2017 IEEE SoutheastCon Hardware Competition

Nathan Kabat, Electrical Engineering
Ben Geist, Computer Engineering

Project Advisor: Mark Randall
Table of Contents

Introduction
Problem Statement
  Stage 1: Discovering the Unknown
  Stage 2: Lightsaber Duel
  Stage 3: Bring Down the Shields
  Stage 4: Fire the Proton Torpedo’s
Scoring:
Summary of Requirements:
Project Design
  Movement, Navigation, & Communication
Robot 2
  Hardware
  Software
Cost
Statement of Work
  Safety
  Validation
  Manufacturability
  Sustainability
  Environmental Impact
  IEEE Standards
References
Appendices
  I. Detailed Cost Breakdown
  II. Schematics
  III. 3D Models
  IV. Code
    Common
    Robot 1
    Robot 2

List of Figures

Figure 1 - Arena Diagram [1]
Figure 2 - Stage 1: Discovering the Unknown [1]
Figure 3 - Lightsaber & Bobbin [1]
Figure 4 - Division of Robots
Figure 5 - Mecanum Wheel [2], Motor [3], and Robot Directions
Figure 6 - I2C Motor Driver [4]
Figure 7 - Robot Communication Diagram
Figure 8 – Actual Robot Communication Link
Figure 9 - Navigation Diagram
Figure 10 - Stage 1 Apparatus
Figure 11 - Stage 3 Apparatus
Figure 12 - Stage 2 Diagram
Figure 13 - Dream Cheeky iLauncher® [13]
Figure 14 - PCB
Figure 15 - Software Block Diagram
List of Tables

Table 1 - Circuit Elements and Codes [1]
Table 2 - Cost

Special thanks to Mr. Mark Randall (project advisor), Dr. Christina Howe, Dr. Marc Mitchell, Dr. Mohsen Lotfalian, Mr. Jeff Cron, Mr. Ray Shelton, and Mrs. Vicky Hasenour for support, and AFB and the Engineering Department for funding.
Introduction

Autonomous robot competitions have helped to further develop robots that aid in manufacturing processes all the way to exploring deep space. Since 1985, the University of Evansville has competed in the IEEE Southeast Conference Hardware Competition. Each year the competition changes, but it always deals with having an autonomous robot solve some complex task(s). This year’s competition is Star Wars themed and showed how a robot can complete multifaceted tasks that are both unique and challenging.

Problem Statement

The robot must fit within a 12”x12”x12” cube. Once the competition started, the robot had to be completely autonomous and have no communication outside the arena when the competition was in progress. The robot could split into separate sections, but must communicate via a wired link. There also had to be a single labeled start switch. If the robot posed a danger to anyone it would be disqualified.

The course was divided up into four stages. These stages respectively were: Decoding the Unknown, Lightsaber Duel, Bring Down the Shields, and Fire the Proton Torpedo. The arena was made from a 4’x8’ sheet of plywood with 2”x4” lumber for the sides. The starting area was composed of a 15”x15” white starting square with a 1”x8” stipe leading to stage one. Stages one, two, and three were all on the same level. Stage four was raised from the starting level by stairs that increased in size. The step sizes were: 0.5”, 1”, and 2”. This can be seen in Figure 1 [1].
Stage 1: Discovering the Unknown

This stage contained five 0.5” pads centered around a 0.5” center pad inside a 6”x6”x1” shadow box mounted to the side of the arena. The five outside pads were 1.5” from the center and spaced at 0°, 72°, 144°, 216°, and 288° as shown in Figure 2. Between each of the 5 pads contained a passive circuit element. Each element had a number associated with it as shown in Table 1. These components only appeared once. The pads could be accessed in any order along with multiple pads at once. If the robot scratched or dented the pads, it was disqualified.
Stage 2: Lightsaber Duel

In this stage the robot detected an electromagnetic field, which correspond to when the robot could hit the lightsaber. The lightsaber and bobbin was 3D printed from PLA plastic as shown in Figure 3. Each time the magnetic field was energized the robot should strike the bottom half of the lightsaber. The top of the lightsaber had 8 RGB LEDs that glowed blue when the robot hit the lightsaber while the magnetic field was energized. The LEDs glowed red if it was hit when the magnetic field was off. The strikes to the lightsaber was metered via a vibration sensor located in the top half. The electromagnet consisted of forty turns of #20 stranded copper wire and was energized by one amp.

Stage 3: Bring Down the Shields

This stage used the message that was decoded by the robot in stage 1. This stage utilized a quadrature encoder with a built-in RGB LED that resembled a combination lock. This encoder was centered inside a 6”x6”x1” shadow box mounted to the side of the arena. The robot turned the knob clockwise ‘N’ number of turns that corresponded to the first digit of the sequence. To input the second number, the knob was turned counterclockwise ‘N’ number of turns that corresponded to the second digit. The clockwise and counterclockwise turns alternated until all 5 digits were entered. The last five digits

<table>
<thead>
<tr>
<th>Code</th>
<th>Component Type</th>
<th>Component Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wire</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Resister</td>
<td>10K, 10% Tolerance</td>
</tr>
<tr>
<td>3</td>
<td>Capacitor</td>
<td>0.1µF, non Polarized</td>
</tr>
<tr>
<td>4</td>
<td>Inductor</td>
<td>500mH</td>
</tr>
<tr>
<td>5</td>
<td>Diode</td>
<td>IN4001 – Cathode/ Anode can be Oriented in Either Direction</td>
</tr>
</tbody>
</table>
entered were the only digits counted. This allowed the robot to restart if it messed up. Each turn had to be +/- 15° from a complete 360° (345° - 375°) rotation. The knob flashed white at the beginning of the match and until the knob was rotated outside of the +/- 15° range. The knob glowed red while it was turned clockwise and blue while it was turned counterclockwise. Every time the knob was within the +/- 15° range it glowed white.

**Stage 4: Fire the Proton Torpedo’s**

This was the final stage of the competition. The robot had to launch/drop three missiles (Nerf® N-Strike Darts) into a 6”x6” hole located on the side opposite of stage 2. This hole was centered with the arena and raised above the top step 3.5”. The match was over once the third missile was launched or 4 minutes had elapsed.

**Scoring:**

- **Starting; Max 40 Points**
  - 10 points were awarded if the robot showed any sign of motion.
  - 30 points were awarded if any part of the robot crossed over the edge of the starting square.

- **Stage 1: Discover the Unknown; Max 135 Points**
  - 10 points were awarded if the robot touched any part of stage 1.
  - 15 points were awarded for correctly decoding 1 pad, 35 for decoding 2 pads, 60 for decoding 3 pads, 90 for decoding 4 pads, and 125 for decoding all five pads.
  - Points for this stage were awarded by properly decoding stage 3 or by presenting to the judges a display that showed that the stage 1 results were properly decoded. The display
had to be an electronic display (LED or LCD) on the robot that was clearly visible by the judges at the end of the match that indicated the code values determined for each of the five pads on stage 1. The displayed code should be a five-digit number, read left to right, to coincide with stage 1 pad numbers 1 through 5. Note that the display had to be visible on the robot by the judges. Additionally, the display had to be read by the judges before the robot entered the sequestration area, or the points were not counted.

- Stage 2: Lightsaber Duel; Max 290 Points
  - 10 points were awarded if the robot touched any part of stage 2
  - 30 points were awarded for the first hit (to start the duel, as indicated by the lightsaber flashing blue)
  - 65 points were awarded if only one additional hit was registered during the magnetic field activation, 110 if two were registered, 170 if three were registered, and 250 if all four were registered. Note that only one hit per magnetic field activation was counted, so to get more than 2 hits, each must occur during a different magnetic field activation.
  - 50 points were deducted for a hit that occurred when the force field was not activated (not counting the 0.5 second grace period after the field was deactivated). Note that only one penalty per magnetic field deactivation was counted (maximum loss was 200).

- Stage 3: Bring Down the Shields; Max 335 Points
  - 10 points were awarded if the robot touched any part of stage 2
  - 45 points were awarded if one digit was dialed in at the correct location in the sequence, 95 if two were dialed in correctly, 155 if three were dialed in correctly, 230 if four were dialed in correctly and 325 if all five were dialed in correctly (for example, if the code
was 12345, but 12435 was entered then the 1, 2 and 5 were correct so 155 points were awarded).

- Since failure at this stage also meant no points for stage 1, the team should provide a very clear display on the robot that showed the decoded digits from stage 1. If all the digits were entered correctly at stage 3, it was assumed all digits were decoded correctly from stage 1 correctly. If one or more digits were incorrectly entered at stage 3, and the team showed that the values were decoded successfully at stage 1, they would receive full points for the correct digits at stage 1, plus any points for any correctly entered digits at stage 3.

- **Final Stage: Fire the Proton Torpedoes; Max 210 Points**

  - 10 points were awarded if at least one Nerf® missile was fired (regardless of whether it went through the portal).

  - 50 points were awarded if only one missile passed through the portal. 120 if two passed through the portal, and 200 if all three passed through the portal.

  - It was not necessary to successfully complete stages 1, 2 or 3 to be awarded points for the final stage, but once the last Nerf® proton torpedo was fired, the match was over and the robot was not allowed to engage with any of the other stages.

- **Maximum points for the competition is 1010.**

**Summary of Requirements:**

- Robot must be completely autonomous and fit in a 12”x12”x12” cube at the start

- Robot could split into separate robots

- No compressed air over 30 pounds per square inch
• Single clearly visible and labeled start switch
• The robot could not present any danger to anyone
• The course had four different stages the robot must complete:
  ○ Discovering the Unknown, Lightsaber Duel, Bringing Down the Shields, and Firing the Proton Torpedoes

**Project Design**

The complex tasks that were set forth in this competition required high tech robots. Therefore, there were two separate robots that completed the course in parallel. One robot completed stages 1 and 3 while the other robot completed stages 2 and 4. This ensured finishing on time would not be an issue. Each robot was 11.5”x5.25”x10” to stay under the size requirements. Each short side of the robot contained the tools needed to complete the stage associated with it as shown in Figure 4.

![Figure 4 - Division of Robots](image)

**Movement, Navigation, & Communication**

Each robot had four mecanum wheels [2] along with four DC motors with gearboxes [3]. This allowed for easy and accurate positioning of the robot on the course. Figure 5 shows the mecanum wheel design, motor with gearbox, and some possible directions the robot could move. The motors were driven by an I2C motor driver [4]. Using I2C to control all four motors and actuators required only two pins on the microcontroller. This took the computation of generating the PWM signal to an external device so the
main microcontroller was used for navigation. Each I2C motor driver controlled two motors and worked with voltages up to 15V and outputted 2A per channel. The I2C motor driver is shown in Figure 6.

The robots used ultrasonic range sensors for navigation, since there was only one line to follow and wall following was too slow [5]. These sensors allowed the robots to find their x and y coordinates on the course. The ultrasonic range sensors sent a pulse width to the microcontroller that corresponded to the distance to the closest object. The microcontroller measured how long the pulse was and determined the distance in millimeters. The sensors returned distances with millimeter resolution and had a repeatability of 40 Hz or 25 mS. Only one sensor could be ranged at a time regardless of the robot due to interference. To get around this issue, both robots had to communicate when they were moving/ranging.
Since wireless communication was not allowed, the two robots had to communicate via a wired link. This is accomplished by using a stretchable key chain [6] with silicon stranded wire [7] dangled from the key chain. The stretchy chain allowed the wire to stay in the air no matter the location of the robots on the course. As the robots moved around, the droops in the cable got smaller and larger depending on if the robots were getting closer or further apart. This can be seen in Figure 7 and Figure 8. The cables were attached to a linear actuator [8] that extended into the air as soon as the competition started. This allowed the cables to rise above any obstructions that were on the robots.

![Figure 7 - Robot Communication Diagram](image1)

![Figure 8 – Actual Robot Communication Link](image2)
The two robots started side by side in the white 15”x15” starting square as shown in Figure 9 [A]. Once the start switch was activated, robot 2 began navigating to stage 4 as shown in Figure 9 [B]. Once at stage 4, robot 2 signaled robot 1 to move to Stage 1 as shown in Figure 9 [C]. While robot 1 was navigating to stage 1, robot 2 fired 2 darts towards stage 4 and then waited. Once robot 1 completed stage 1, it navigated to stage 3 as shown in Figure 9 [D]. To make final adjustments at stages 1 and 3, two IR sensors were used to line up with the white frames associated with each stage. After navigating to stage 3, robot 1 signaled robot 2 that it could move again. Robot 2 then navigated to stage 2 as shown in Figure 9 [E]. Once the battle was over, robot 2 navigated to stage 4 as shown in Figure 9 [F]. Once at stage 4, the final dart was fired and the competition was over.
**Robot 1**

The apparatus to complete stage 1, shown in Figure 10, was custom made and 3D printed. It was attached to a linear slide [9] that moved out of the robot via a track actuator [10] once at stage 1. The apparatus was comprised of six spring loaded probes [11]; five of the probes were mounted at 0°, 72°, 144°, 216°, and 288° 1.5 inches from the center and one probe at the center. The apparatus also had a bump sensor attached to it. This sensor was used to determine how far the track actuator needed to move the apparatus. This ensured the probes did not damage the pads. The robot inputted a step response to determine which passive circuit component lied between one of the five outer and common center pads. This process was repeated until all five pads have been read and the values have been stored. The robot outputted the sequence to an LCD display to ensure points were received.

To complete stage 3, the robot had similar hardware as stage 1. The apparatus to complete stage 3, shown in Figure 11, was attached to a linear slide [9] that moved via a track actuator [10]. The custom 3D printed apparatus was lined with Plasti Dip® to make sure the inside of the apparatus had a rubbery surface. This allowed for a better grip of the knob. The apparatus was attached to a 360° smart servo [12] which allowed continuous rotation. One complete rotation was measured by a Hall effect sensor.
Robot 2

Stage 2 had a rod attached to a smart servo [12] that swung out from the chassis and hit the hilt of the light saber when the magnetic field was turned on. This can be seen in Figure 12. A hit was measured by a vibration sensor mounted to the rod. The magnetic field was measured using an analog Hall effect sensor.

The robot utilized a Dream Cheeky iLauncher® [13], shown in Figure 13, to fire Nerf® missiles through the hole of stage 4. The launcher was raised out of the top of the robot by two track actuators [10]. There was a custom 3D printed base that the launcher was attached to that span the distance between the two track actuators [10].

Hardware

Each robot had a common PCB board, which lowered the cost. Each PCB was populated with only what was necessary for that specific robot. All the external parts to the board had JST connectors to allow easy removal if a part failed. This also allowed the board to be easily removed if it need to be replaced. Every item on both robots were easily taken off and replaced, due to the way the robots were designed. Both robots were completely 3D modeled before any of the parts were 3D printed. The 3D models can be seen in Appendix III. All the chassis and covers were 3D printed from ABS plastic. Figure 14 shows what the PCB board looked like before it was printed.

---

Figure 12 - Stage 2 Diagram

Figure 13 - Dream Cheeky iLauncher® [13]
Below is a short list of the major hardware used along with its part number:

- **Processor**: Atmel AT89C51CC03 microprocessor [14]. The processor had many features such as three 16-bit timers, 32 + 4 digital I/O lines, five-channel 16-bit PCA (PWM, Timer and Edge Capture), and a 10-bit analog to digital Converter with eight multiplexed channels, to name a few.

- **Track Actuator**: Actuonix T16-100-64-12-S [10]. Four of these were used in both robots. They were used for moving stage 1 and stage 3 apparatus out of the robot. There were two that raised the launcher out of robot 2.

- **Linear Slide**: Actuonix S9-100 Linear Slide Rail [9]. Two of these were used to mount stage 1 and stage 3’s apparatus. This ensured there was not too much force on the track actuators, which would cause them to break.

- **Linear Actuator**: Actuonix L16-100-63-12-S [8]. Two of these were used for moving the communication cable above the robots.

- **Motors**: Pololu (#995) 250:1 Micro Metal Gearmotor HP 6V [3]. Four of these were used on each robot for a total of eight. These motors allowed the robots to be small and still move.

- **Wheels**: Robot Shop (RB-Ftr-40) mecanum wheel 4 pack (w/ metal hubs) [2]. Four of these were used on each robot for a total of eight.
• **Drivers:** Seed Studio (105020001) Power Management IC Development Tools Grove – I2C Motor Driver [4]. Three of these were used in each robot for a total of six. They controlled all the motors, track actuators, and linear actuators.

• **Distance Sensors:** Maxbotix MB1013 HRLV-MaxSonar®-EZ1™ [5]. These sensors gave back millimeter resolution. There were four on each robot, one on the front and back and two on one side.

• **Servo:** Rev Robotics (REV-41-1097) Smart Robot Servo [12]. The smart servo was programmed to be either a 280° servo or a continuous servo. Stage 2 utilized the 280° servo function while stage 3 used it as a continuous servo.

• **Launcher:** Dream Cheeky iLauncher® (#358) [13]. The iLauncher® was used to fire the Nerf® Darts for stage 4.

• **Mux:** Two different multiplexers were used in the project:
  
  o Texas Instrument (CD4051B) [15] is an 8:1 multiplexer that was used to connect all 5 probes for stage 1 to one port pin on the processor.

  o Texas Instrument (CD4052B) [15] is an 8:2 multiplexer that was used to communicate to the distance sensors. These worked perfect for this application because each sensor had to have a send and receive line that came back to the processor.

• **Vibration Sensor:** Adafruit (#2384) Medium Vibration Sensor Switch [16]. One of these were used to detect when the servo swung far enough to hit the lightsaber.

• **Hall Effect Sensor:** Texas Instrument (DRV5053VAQLPGM1) analog Hall effect sensor [17]. Two of these were used to measure the magnetic field at stage 2.

• **Probes:** Smith Connectors (S-3-J-4-G S/C SS SPGS) contact probes headless radius [18]. Six of these were used in the stage 1 apparatus. These were spring loaded and allowed the robot to touch the pads without damaging them.

• **Extrusion:** Rev Robotics (REV-41-1017) 15mm extrusion [19]. This material was used in stages 1, 2, and 3. This material allowed for easy horizontal/vertical adjustments in the stages apparatuses.
• **Shaft**: Servo City (634048) ¼” Stainless Steel D-Shaft [20]. It was used to hit the lightsaber in stage 2. The shaft was cut and welded to form a 90-degree bend. The shaft was also used in stage 3. The shaft was attached to the servo and the custom knob turner. In both cases, the shaft was inserted into a servo spline to allow the servo to control the shaft.

• **Servo Spline**: Servo City (525134) C1 Servo Spline to ¼” Shaft Coupler (w/ Set Screw) [21]. This was used to connect the ¼” shaft to the smart servos.

• **RJ12 Plug**: L-com (ECJ504-6) Modular Panel RJ12 Jack [22]. Two of these were used for fast connection/removal of the communication cable.

**Software**

Each robot had a separate microcontroller that acted independently of each other except a few communications for timing purposes. Each microcontroller controlled three I2C motor driver boards that controlled all the motors and linear/track actuators on each robot. This eliminated the need to constantly use the 8051 to send PWM to the motors, which slowed it down.

All the code was edited in Keil µVision 5 IDE and written in C. A few external libraries were used to aid in the coding process. The “stdio” library was used on Robot 1 to take the code and output it to the LCD screen. This was accomplished with the sprintf function from the library. The library was also used to print out messages while debugging the code.

The navigation algorithm was by far the most complex used on the robots. Due to mechanical inconsistencies, the navigation algorithm needed to guarantee the robot made it to each stage. Two sensors were mounted on one side of each Robot, to make sure the robot was not crooked. These sensors were placed as close to the wheels as possible to get the most accurate readings. The difference in the two sensors were sent into a custom algorithm to level the robot out when needed. Figure 15 shows the order of events the code followed.
Start

Robot 1

No Signal

Wait

Signal Received

Navigate to Stage 1

Discovering the Unknown

Navigate to Stage 3

Signal Robot 2 to Move

Bring Down the Shield

Robot 2

Navigate to Stage 4

Signal Robot 1 to Move

Fire 2 Nerf Darts®

Wait

No Signal

Signal Received

Navigate to Stage 2

Lightsaber Battle

Navigate to Stage 4

Fire 1 Nerf® Dart

End

Figure 15 - Software Block Diagram
The cost of this project was justified since there were two robots fabricated. Each robot had separate motors, drivers, actuators, and processors. Even though there were two robots, each robot had specific tasks associated with each one. Therefore, they were both necessary to finish the course.

<table>
<thead>
<tr>
<th>Category</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course/Track</td>
<td>$163</td>
</tr>
<tr>
<td>Robot for Stages 1 &amp; 3</td>
<td>$2,626</td>
</tr>
<tr>
<td>Travel (Charlotte, NC)</td>
<td>$1,600</td>
</tr>
<tr>
<td>Total</td>
<td>$4,389</td>
</tr>
</tbody>
</table>

**Statement of Work**

**Safety**

Both robots were very complex in the mechanical design. This caused a concern for safety. Below are some problems that could arise along with some solutions:

- The missile launcher could shoot in the wrong direction and cause injury to someone. By using angles lower than eye level the launcher was not able to hit someone above the waste.
- There were many pinch points associated with the robot. By enclosing the whole robot this decreased the chance of pinching oneself.
- The I2C motor controller could get very hot when using high current which could cause damage to other components leading to catastrophic meltdown of the robot. This was eliminated by isolating the controller from the rest of the components.
- The Li-Po batteries could catch fire if used incorrectly. To reverse this, the batteries were used to manufacture specifications.
Validation

This project was based on the rules given for the 2017 IEEE SoutheastCon Hardware Competition. Following these rules guaranteed the robot to compete in the competition along with satisfying our client requirements.

Manufacturability

All chassis and mounts were 3D printed, guaranteeing repeatability. Also, the circuity was propagated on PCB boards so the circuit was always the same. All other parts were readily available from online vendors.

Sustainability

The robot was sustainable if the batteries stay charged. Since the robot was made to complete a certain set of tasks, it would not perform outside of what it was intended for.

Environmental Impact

The robot used rechargeable batteries, which saved space in landfills and did not produce hazardous emissions. The ABS plastic that was used for the chassis could be ground up and reused for other 3D prints.

IEEE Standards

The 1872-2015 IEEE standard for Robotics and Automation was taken into consideration during this project to ensure a safe interaction between the robots and humans.

Results

The robots navigated the course and finished the four stages successfully. The average time the two robots took to complete the course was 3 min. and 10 sec. The fastest time the robots completed the course was 2 min. and 17 sec.
References


## Appendices

### I. Detailed Cost Breakdown

<table>
<thead>
<tr>
<th>Items Bought</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menards (Track Supplies)</td>
<td>$109.16</td>
</tr>
<tr>
<td>Robot Shop (Wheels)</td>
<td>$96.50</td>
</tr>
<tr>
<td>Pololu (Motors Wrong)</td>
<td>$124.93</td>
</tr>
<tr>
<td>Pololu Motors Right</td>
<td>$132.70</td>
</tr>
<tr>
<td>Newegg (iLauncher)</td>
<td>$49.99</td>
</tr>
<tr>
<td>Robot Shop (I2C Sensor)</td>
<td>$43.95</td>
</tr>
<tr>
<td>Adafruit (Neopixel &amp; USB Connector)</td>
<td>$17.75</td>
</tr>
<tr>
<td>Sparkfun (Encoder, Knob, Wire)</td>
<td>$21.40</td>
</tr>
<tr>
<td>Digikey (2 Female Plugs, 2 MIDI Connectors)</td>
<td>$7.87</td>
</tr>
<tr>
<td>Actuonix (Linear Actuator, Track Actuator, )</td>
<td>$94.63</td>
</tr>
<tr>
<td>Mouser (Power Management IC, Rocker Switches, I2C Multiplexer Switch IC, Marbotix Sensor, Serial to Parallel Converters)</td>
<td>$131.49</td>
</tr>
<tr>
<td>HobbyKing (Batteries &amp; Connectors)</td>
<td>$114.74</td>
</tr>
<tr>
<td>Rev Robotics (Extrusion, Hex Pillow Black, Inside Corner, Lap Corer, Servo Bracket, Hex Shaft, Nuts, Screws, Shaft Collars)</td>
<td>$84.95</td>
</tr>
<tr>
<td>Amazon (Wire, Ribbon Cable Connector, JST plugs, Headers)</td>
<td>$112.16</td>
</tr>
<tr>
<td>Mouser (I2C Motor Drivers, Magnetic Sensors, 1/4&quot; Plug, Resistor, Fuse, Probe, Octocopter)</td>
<td>$104.35</td>
</tr>
<tr>
<td>Servo City (C1 Servo Spline, D-Shaft, Collars, Serco Arm)</td>
<td>$24.04</td>
</tr>
<tr>
<td>Robot Shop (Wheels, Flat Bearing Mount)</td>
<td>$69.48</td>
</tr>
<tr>
<td>Almark Enterprises Inc. (Retractable Cable Reel)</td>
<td>$51.95</td>
</tr>
<tr>
<td>Pololu (Motors, Voltage Regulators, Reverse Voltage Protection, Brackets, Screws, Encoders)</td>
<td>$116.69</td>
</tr>
<tr>
<td>Servo City (2ST Spine, D-Shafts, Collars)</td>
<td>$24.18</td>
</tr>
<tr>
<td>Mouser (Resister Network, Driver Cables, Hall Effect Sensors, Mounting Clips)</td>
<td>$39.65</td>
</tr>
<tr>
<td>Hobby Lobby (Epoxy)</td>
<td>$3.99</td>
</tr>
<tr>
<td>Actuonix (Linear Actuator, Track Actuators, Slide Rail, Slide Block)</td>
<td>$270.91</td>
</tr>
<tr>
<td>Amazon (JST plugs and Standoffs)</td>
<td>$37.75</td>
</tr>
<tr>
<td>Advanced Circuits (Trial PCB Board)</td>
<td>$53.19</td>
</tr>
<tr>
<td>Adafruit (Wire and TTL Breakout Cable)</td>
<td>$73.05</td>
</tr>
<tr>
<td>Advanced Circuits (2nd PCB Board)</td>
<td>$53.74</td>
</tr>
<tr>
<td>L-Com (RJ12 Connectors)</td>
<td>$24.48</td>
</tr>
<tr>
<td>Tanis (3&quot;, 2&quot;, &amp;1&quot; Brushes)</td>
<td>$70.03</td>
</tr>
<tr>
<td>Windy City Novelties (Blue Plastic 23&quot; Coil Key Chain with Clip – 12 Pack)</td>
<td>$10.79</td>
</tr>
<tr>
<td>Hobby Lobby (Paint and Supplies)</td>
<td>$52.88</td>
</tr>
<tr>
<td>Adafruit (ToF Sensors)</td>
<td>$51.99</td>
</tr>
<tr>
<td>Amazon (Resister and RJ12 Cables)</td>
<td>$25.39</td>
</tr>
<tr>
<td>Advanced Circuits (2 New Boards)</td>
<td>$86.76</td>
</tr>
<tr>
<td>Mouser (Analog Sensor, Probes, 5.1K Resistors, 10K Resistors, Rotary Encoder Board, 820pF)</td>
<td>$49.86</td>
</tr>
<tr>
<td>Amazon (2 Pin JST Connectors) - Nathan Bought</td>
<td>$14.06</td>
</tr>
<tr>
<td>Newegg (iLauncher) - Nathan Bought</td>
<td>$20.32</td>
</tr>
<tr>
<td>Pololu (x6 spare motors, 5V/3.3V Reg, Encoders) - Nathan Bought</td>
<td>$153.45</td>
</tr>
</tbody>
</table>

| Total                                                                      | $2,625.20 |
II. Schematics

Main 8051 Circuit
Power Distribution Circuit

I2C Driver Circuit
III. 3D Models

Robot 1
Robot 2
### IV. Code

#### Common

```c
//ADCRead.c
//This file contains the functions needed to do a 8 bit ADC read
#include <at89c51cc03.h>
#include "ADCRead.h"

//This function does a 8 bit ADC read from the given channel
unsigned char ADC8BIT (unsigned char channel)
{
    unsigned char value;
    ADCON &= 0x80; // Rest ADC Channel Select
    ADCON |= channel; // channel select
    ADCON &= 0x80; // starts ADC conversion 8-Bit
    while(!((ADCON & 0x10))); // loop until done
    value = ADDH; // load ADDH value
    ADCON &= 0x80; // clear ADEOC = 0
    ADCON &= 0x80; // Rest ADC Channel Select
    return value;
}

//Delay.c
//This file contains all the dealy functons
#include <at89c51cc03.h>
#include "Delay.h"

//This function will delay for 10 milliseconds * the number passed
void delay10MILL (unsigned int time)
{
    int i;
    for(i=0;i<time;i++)
    {
        TRL = (65536)>>8); // Load timer high byte
        TLL = (65536); // Load timer low byte
        TR1 = 1; // Start timer 0
        TF1 = 0; // Reset overflow flag
        while (TF1 == 0); // Wait for timer 0 to time out
        TF1 = 0; // Turn off timer 0
    }
}

//This function will delay for 1ms per second * the number passed
void delay1MILL (unsigned int time)
{
    int i;
    for(i=0;i<time;i++)
    {
        TML = (65536)>>8); // Load timer high byte
        TLL = (65536); // Load timer low byte
        TR1 = 1; // Start timer 0
        TF1 = 0; // Reset overflow flag
        while (TF1 == 0); // Wait for timer 0 to time out
        TF1 = 0; // Turn off timer 0
    }
}

//This function will delay for 1microsecond * the number passed
void delay1MICR (unsigned int time)
{
    int i;
    for(i=0;i<time;i++)
    {
        TML = (65536)>>8); // Load timer high byte
        TLL = (65536); // Load timer low byte
        TR1 = 1; // Start timer 0
        TF1 = 0; // Reset overflow flag
        while (TF1 == 0); // Wait for timer 0 to time out
        TF1 = 0; // Turn off timer 0
    }
}
```

// This file contains all the functions for I2C communications
#include "st69c51cc03.h"
#include "I2C.h"
#include "Pin Assignments.h"

// Function Prototypes
void Delay (unsigned int tick);
void write(unsigned char I2CData);
unsigned char I2C_Clock = 1;

// bit addressable variable that will be used for shifting in out I2C data
unsigned char bdata a;

// MSB and LSB correspond to the most/least significant bit of 'a' respectfully
sbit LSB = a ^ 0;
sbit MSB = a ^ 7;

// I2C master starts communication by sending Slave address
void beginTransmission( unsigned char Slave_address ); // Start Connection to Slave_address
{
  scl = 1;
  Delay(I2C_Clock);
  sda = 0;
  Delay(I2C_Clock);
  write(Slave_address);
}

void endTransmission(); // End I2C conversation with connected slave device
{
  sda = 0;
  scl = 1;
  Delay(I2C_Clock |);
  sda = 1;
}

// Sends a byte of data to the slave device.
void write(unsigned char I2CData); // send byte of data on I2C bus
{
  unsigned char count;
  a = I2CData;
  for(count=0; count<8; count++)
  {
    scl = 0;
    Delay(I2C_Clock |); 
    sda = MSB;
    a = a << 1;
    Delay(I2C_Clock |);
    scl = 1;
    Delay(I2C_Clock |);
    scl = 0;
  }
  scl = 1;
  Delay(I2C_Clock |);
  scl = 0;
  Delay(I2C_Clock |);
  scl = 1;  // Added this since uploading to Motor Driver
}

// Delay used for increasing or decreasing I2C clock.
void Delay (unsigned int tick)
{
  unsigned int t, tock;
  for(t=0;t<tick;t++)
  {
    for(tock=0;tock<50;tock++);
  }
}
//LCD.c
//This file contains the functions needed to output to the LCD screen
#include <at89c51cc03.h>
#include "LCD.h"

xdata unsigned char msg[50];

// print function for serial command to LCD
void print (char* msg)
{
    unsigned char i;
    i = 0;
    while(msg[i] != 0) //Char string ends in 0
    {
        TI = 0;
        SHIF = msg[i];
        i++;
        while (TI == 0); // Wait for write to be done
        TI = 0;
    }
}

//Mux Select.c
//This file contains the function used to select the mux channels
#include <at89c51cc03.h>
#include "Mux Select.h"
#include "Pin Assignments.h"

//This function sets the mux to the correct channel given
extern void select (unsigned char channel)
{
    switch (channel)
    {
    case 0:
        select_A = 0;
        select_B = 0;
        select_C = 0;
        break;
    case 1:
        select_a = 1;
        select_b = 0;
        select_C = 0;
        break;
    case 2:
        select_a = 0;
        select_b = 1;
        select_C = 0;
        break;
    case 3:
        select_a = 1;
        select_b = 1;
        select_C = 0;
        break;
    case 4:
        select_a = 0;
        select_b = 0;
        select_C = 1;
        break;
    case 5:
        select_a = 1;
        select_b = 0;
        select_C = 1;
        break;
    case 6:
        select_a = 0;
        select_b = 1;
        select_C = 1;
        break;
    case 7:
        select_a = 1;
        select_b = 1;
        select_C = 1;
        break;
    case 8:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    case 9:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    case 10:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    case 11:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    case 12:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    case 13:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    case 14:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    case 15:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    default:
        select_a = 0;
        select_b = 0;
        select_C = 0;
        break;
    }
}
// This file contains all the functions used to move the motors
#include "MotorDriver.h"
#include "delay.h"
#include "MotorControl.h"
#include "Pin Assignments.h"

VF2MotorController();

// Function prototypes
void MotorDirectionSet (unsigned char Address, unsigned char MotorSpeedA, unsigned char MotorSpeedB);
void MotorDirectionSet (unsigned char Address, unsigned char direction);
unsigned char map(unsigned char x, unsigned char in_min, unsigned char in_max, unsigned char out_min, unsigned char out_max);

// This function moves all four wheels to get the desired direction and speed
void move (unsigned char direction, unsigned char speed)
{
    switch (direction)
    {
    case 'f':
        MotorSpeedSetAB(I2CMotorDriverAddrFront, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrFront, 0x06); // 0110
        MotorSpeedSetAB(I2CMotorDriverAddrRear, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrRear, 0x06); // 0110
        break;
    case 'b':
        MotorSpeedSetAB(I2CMotorDriverAddrFront, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrFront, 0x09); // 1001
        MotorSpeedSetAB(I2CMotorDriverAddrRear, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrRear, 0x09); // 1001
        break;
    case 'r':
        MotorSpeedSetAB(I2CMotorDriverAddrFront, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrFront, 0x0A); // 1010
        MotorSpeedSetAB(I2CMotorDriverAddrRear, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrRear, 0x0A); // 1010
        break;
    case 'l':
        MotorSpeedSetAB(I2CMotorDriverAddrFront, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrFront, 0x05); // 0101
        MotorSpeedSetAB(I2CMotorDriverAddrRear, speed, speed);
        MotorDirectionSet(I2CMotorDriverAddrRear, 0x05); // 0101
        break;
    case 1: // Move NW
        MotorSpeedSetAB(I2CMotorDriverAddrFront, speed, 0);
        MotorDirectionSet(I2CMotorDriverAddrFront, 0x02); // XX10
        MotorSpeedSetAB(I2CMotorDriverAddrRear, speed, 0);
        MotorDirectionSet(I2CMotorDriverAddrRear, 0x04); // 01XX
        break;
    case 2: // Move NE
        MotorSpeedSetAB(I2CMotorDriverAddrFront, 0, speed);
        MotorDirectionSet(I2CMotorDriverAddrFront, 0x04); // 01XX
        MotorSpeedSetAB(I2CMotorDriverAddrRear, speed, 0);
        MotorDirectionSet(I2CMotorDriverAddrRear, 0x02); // XX10
        break;
    case 3: // Move SW
        break;
    }
MotorSpeedSetAB(12CMotorDriverAddFront, 0, speed);
MotorDirectionSet(12CMotorDriverAddFront, 0x00); // 10XX
MotorSpeedSetAB(12CMotorDriverAddRear, speed, 0);
MotorDirectionSet(12CMotorDriverAddRear, 0x01); // XX01
break;

// Move SE
MotorSpeedSetAB(12CMotorDriverAddFront, speed, 0);
MotorDirectionSet(12CMotorDriverAddFront, 0x01); // XX01
MotorSpeedSetAB(12CMotorDriverAddRear, speed, 0);
MotorDirectionSet(12CMotorDriverAddRear, 0x00); // 10XX
break;

// Turn Right
MotorSpeedSetAB(12CMotorDriverAddFront, (speed-70), speed);
MotorDirectionSet(12CMotorDriverAddFront, 0x06); // 0110
MotorSpeedSetAB(12CMotorDriverAddRear, (speed-70), speed);
MotorDirectionSet(12CMotorDriverAddRear, 0x06); // 0110
break;

// Turn Left
MotorSpeedSetAB(12CMotorDriverAddFront, speed, (speed-70));
MotorDirectionSet(12CMotorDriverAddFront, 0x06); // 0110
MotorSpeedSetAB(12CMotorDriverAddRear, speed, (speed-70));
MotorDirectionSet(12CMotorDriverAddRear, 0x06); // 0110
break;

// 360 CCW
MotorSpeedSetAB(12CMotorDriverAddFront, speed, speed);
MotorDirectionSet(12CMotorDriverAddFront, 0x0A); // 1010
MotorSpeedSetAB(12CMotorDriverAddRear, speed, speed);
MotorDirectionSet(12CMotorDriverAddRear, 0x0A); // 1010
break;

// 360 CW
MotorSpeedSetAB(12CMotorDriverAddFront, speed, speed);
MotorDirectionSet(12CMotorDriverAddFront, 0x05); // 0101
MotorSpeedSetAB(12CMotorDriverAddRear, speed, speed);
MotorDirectionSet(12CMotorDriverAddRear, 0x05); // 0101
break;

void stop (){
    MotorSpeedSetAB(12CMotorDriverAddFront, 0, 0);
    MotorDirectionSet(12CMotorDriverAddFront, 0x00);
    MotorSpeedSetAB(12CMotorDriverAddRear, 0, 0);
    MotorDirectionSet(12CMotorDriverAddRear, 0x00);
}

void LinearActuator(unsigned char stage, unsigned char direction)
{
    MotorSpeedSetAB(12CMotorDriverAddMiddle, 90, 90);
    delay(1MILLI);
    switch (direction)
    {
    case 'r':
        MotorDirectionSet(12CMotorDriverAddMiddle, 0x01<<stage-1); //0001 or 0100
        delay(MILLI4);
        break;
    case 'l':
        MotorDirectionSet(12CMotorDriverAddMiddle, 0x02<<stage-1); //0010 or 0100
        delay(MILLI4);
        break;
    case 'b':
        MotorDirectionSet(12CMotorDriverAddMiddle, 0x05); //0101
        delay(MILLI4);
    }
break;
    case 'a':
        MotorDirectionSet(I2CMotorDriverAddMiddle, 0x0A); //1010
        delayMILLI(4);
        case 's':
        MotorDirectionSet(I2CMotorDriverAddMiddle, 0x00); //0000
        delayMILLI(4);
        break;
    }

    //This function is used to control the motor driver boards
    void MotorSpeedSetAB(unsigned char Address, unsigned char MotorSpeedA, unsigned char MotorSpeedB)
    {
        MotorSpeedA = map(MotorSpeedA, 0, 100, 0, 255);
        MotorSpeedB = map(MotorSpeedB, 0, 100, 0, 255);
        beginTransmission(Address); // transmit address to I2C device
        write(MotorSpeedSetA);       // set pan header
        write(MotorSpeedA);          // send pwm
        write(MotorSpeedB);          // send pnm
        endTransmission();           // stop transmitting
    }

    void MotorDirectionSet (unsigned char Address, unsigned char Direction) // Adjust the direction of the motors 0b0000 14 13 12 11
    {
        beginTransmission(Address);
        write(DirectionSet);
        write(Direction);
        write (Nothing);
        endTransmission();
    }

    //This function maps the speed from 0-100 into 0-255
    unsigned char map(unsigned char x, unsigned char in_min, unsigned char in_max, unsigned char out_min,
        unsigned char out_max)
    {
        return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
```c
#define _Pin_Assignments_h

#define prox_1  P0_0
#define prox_2  P0_1
#define com_2   P0_3
#define com_4   P0_4
#define start_sw P0_7
#define receive_range P1_7 // (receive_range) Pulse Width Measurement

#define light_1 P1_2
#define pca_0   P1_3
#define P1_4
#define P1_5
#define P1_6
#define select_a P2_0
#define select_b P2_1
#define select_c P2_2
#define send_range P2_3
#define inhibit P2_4
#define scl P2_5
#define sda P2_6
#define V_switch P2_7
#define P3_0 tx
#define P3_1 Rx
#define rot_1 P3_2
#define shooter_1 P3_4
#define light_2 P3_5 // was shooter_2
#define P3_6
#define P3_7
#define P4_0
#define P4_1

#endif
```
xdata unsigned int pulse_width_old, pulse_width_new = 0;

bit pulse_done;

//This function gets the distance in mm from the given sensor
unsigned int getDistance (unsigned char sensor)
{
    EC = 1; // Enable PCA Interrupt
    inhibit = 0; // Enable Multiplexer
    select(sensor); // Select Sensor From Mux
    send_range = 1; // Bring high to command a reading
    delayMICR(20); // delay
    send_range = 0; // Bring low to stop reading
    while(pulse_done == 0); // Wait until the conversion has been completed
    pulse_done = 0;
    inhibit = 1; // disable Multiplexer
    EC = 0; // Disable PCA Interrupt
    delayMILL(20); // delay
    return pulse_width_new; // return
}

#pragma save // Remember current registerbank
#pragma REGISTERBANK(2) // Tell C51 base address of current registerbank.
void pulseWidth(void)
{
    CCF4 = 0; // Clear Interrupt Flag
    if(receive_range)
    {
        pulse_done = 0;
        CR = 1; // Start Timer
        pulse_width_old = (CCAP4H<<8)+CCAP4L; // Load Timer Values
    }
    if(!receive_range)
    {
        pulse_width_old = ((CCAP4H<<8)+CCAP4L)-pulse_width_old; // Calculate how many counts have passed
        pulse_width_new = (pulse_width_old*10)/47;
        pulse_done = 1;
    }
}
#pragma restore // Put back to original registerbank
// This file contains all the functions used to implement the real time clock
#include <at89c51cc03.h>
#include "RTC.h"

xdata unsigned char minutes;
xdata unsigned char seconds;
xdata unsigned char tenthCounter;
xdata unsigned char milliSeconds;

// This function will start the real time clock
extern void startRTC (void)
{
    minutes=0; seconds=0; tenthCounter=0; milliSeconds=0; // Reset RTC
    TL0 = Ox3A; // Timer 0 set to ED3 - 60835
    TR0 = 1; // Start Timer 0
    ET0 = 1; // Timer 0 interrupt enable

    // This function will stop the real time clock
    extern void stopRTC (void)
    {
        TR0 = 0; // Stop Timer 0
        ET0 = 0; // Timer 0 interrupt disable
    }

    // This interrupt is what inc the real time clock
    void T0Int() interrupt 1 using 3
    {
        TL0 = Ox3A; // Timer 0 set to ED3 - 60835
        if(milliSeconds > 99) // .1 sec = 100 msec
        {
            milliSeconds = 0;
            tenthCounter++;
        }
        else // For each 10 update seconds
        {
            if(tenthCounter > 9) // For each ten update seconds
            {
                tenthCounter = 0;
                seconds++;
                if(seconds > 59) // For each 60 seconds update minutes
                {
                    seconds = 0;
                    minutes++; // minutes can overflow
                }
            }
        }
    }
}
```c
// Nathan Kabat and Ben Geist
// This is the main code for the IEEE Robot 1
// Function Prototypes
void setup(void);

#include <stdio.h>
#include <stdlib.h>
#include "MotorControl.h"
#include "Navigation.h"
#include "Delay.h"
#include "Stage 1 Decode Message.h"
#include "LCD.h"
#include "Pulse Width Measurement.h"
#include "Servo.h"
#include "Stage 3.h"
#include "Pin Assignments.h"
#include "RTC.h"

// Pin Assignments
/*
PLEASE LOOK AT Pin Assignments2.pcn
*/

// Main Program
void main ()
{
    unsigned char codeMsg[5];
    setup();
    while((com_2==1&&start_sw==1)) // wait on com 2 or start_sw
    {
        FindStage1(); // move to stage 1
        LineActuator1('e'); // extend stage 1 apparatus
        while((prox_1==1)) // wait on stage 1 prox sensor
        {
            delayMicros(15); // delay to make sure prods are in
        }
        LineActuator1('s'); // stop extending stage 1 apparatus
        getCode(codeMsg); // read in the code
        delay10MILLI(20); // delay
        LineActuator1('r'); // retract stage 1 apparatus
        delay10MILLI(100); // delay
        FindStage3(); // move to stage 3
        com_3=0; // signal com 3
        zeroOut(); // turn stage 3 apparatus to home position
        LineActuator3('e'); // extend stage 3 apparatus
        startRTC(); // start real time clock
        while((prox_2==1&&seconds<9)) // wait until stage 3 prox or 9 sec pass
        {
            stopRTC(); // stop real time clock
            delay10MILLI(75); // delay
        }
        LineActuator3('s'); // stop extending stage 3 apparatus
        inputCode(codeMsg); // input the code
        delay10MILLI(100); // delay
        LineActuator3('r'); // retract stage 3 apparatus
        delay10MILLI(100); // delay
        moveAway(); // moves stage 3 apparatus to set position
        sprintf(msg,"%d,%d,%d,%d,%d", (int)codeMsg[0],(int)codeMsg[1],(int)codeMsg[2],(int)codeMsg[3],(int)codeMsg[4]);
        print(msg); // output the code to LCD
        delay10MILLI(800); // delay
        LineActuator69('s'); // stop sending retract signal
        while(1); // wait forever
    }
}

void setup()
{
    unsigned char i;
```


#ifdef _ADCRead_h || defined(_Stage_1_Decode_Message_h)
//ADC set up
ADCON |= 0x20; // Enable ADC Function
ADCLK = 0x08; // Set AD Clock speed
ADCON &= 0xF8; // Rest ADC Channel Select
//ADC set up for P1.0
ADCF |= 0x01;
//ADCF |= 0x02;
#endif

#ifndef _Launcher_h
// For ILauncher
shooter_1 = 1;
shooter_2 = 0;
#endif

#ifndef _Mux_Select_h
// for Mux Select
inhibit = 1; // Disable Multiplexer
#endif

#ifndef _Servo_h
// For Servo using 16-bit compare mode for 0.02 deg/tick
CCP4M0 |= 0x48; // 0100 1000 Compare Bit & Match Bit
CCP4M0 |= 0x01; // 0000 0001 Enable CFP0 Interrupt Bit
pce_0 = 1;
#endif

#ifndef _Stage_3_h
// Declare External Interrupt
IT0 = 1; // External Interrupt 0 Falling Edge Triggered
#endif

#ifndef _LCD_h || defined(_TTL_Distance_Sensor_h)
// Serial Communication RX and TX @ 9600 Baud
SCON = 0x40; // Mode 2, 9 bit uart transmit only, uses T2
REN=1; // Transmit and Receive
RCLK=1; // Turn on receive clock in T2CON
TCLK=1; // Turn on transmit clock in T2CON
//Baud rate = fCrystal/(32*65536 - (RCAPI, RCAPI)) for x1 mode
//Baud rate = fCrystal/(16*65536 - (RCAPI, RCAPI)) for x2 mode
//1200 = 28.2076 x 10^6/(16*65,536 = (RCAPI, RCAPI))
//4800 = 28.2076 x 10^6/(16*65,536 = (RCAPI, RCAPI))
//9600 = 28.2076 x 10^6/(16*65,536 = (RCAPI, RCAPI))
//19200 = 28.2076 x 10^6/(16*65,536 = (RCAPI, RCAPI))
//38400 = 28.2076 x 10^6/(16*65,536 = (RCAPI, RCAPI))
//76800 = 28.2076 x 10^6/(16*65,536 = (RCAPI, RCAPI))
//48000 = 28.2076 x 10^6/(16*65,536 = (RCAPI, RCAPI))
//96000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//192000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//384000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//768000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//1536000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//3072000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//6144000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//12288000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//24576000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//49152000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//98304000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//196608000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//393216000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//786432000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//1572864000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//3145728000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//6291456000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//12582912000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//25165824000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//50331648000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//100663296000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//201326592000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//402653184000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//805306368000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//1610612736000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//3221225472000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//6442450944000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)
//12884901888000 = 28.2076 x 10^6/16*65,536 = (RCAPI, RCAPI)

RCAP2H=0XFE; // 4800 baud @ 28.2076 Mhz
RCAP2L=0x01; //
TR2=1; // T2CON bit to start Timer 2.
RT = 0; // Clear the receive interrupt flag
EA = 1; // Turn on global interrupt flag
//ES = 1; // Turn on serial interrupt
#endif

delay1MILL(50);
#ifndef _MotorControl_h
// Retract all the Actuators
LinearActuator(69, 'a');
delay1MILL(30);
LinearActuator(69, 'b');
#endif

// Navigation.c
// This file contains all the functions used for specific navigation on Robot 1
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "MotorControl.h"
#include "Delay.h"
#include "Navigation.h"
#include "LCD.h"
#include "Pulse Width Measurement.h"
#include "Pin Assignments.h"

// Function prototypes
void MakeLevel1(unsigned char sen);
unsigned int moveTo(unsigned int distance, unsigned dir, unsigned char speed, unsigned char sensor);
void MakeLevel2(unsigned char sen);

// This function will navigate Robot 1 to stage 1
void FindStage1()
{
    int diff = 0;
    unsigned char power = 0;
    int dist = 0;
    unsigned char i;
    int rightWall = 638;
    moveTo(670, 'b', 50, 3);
    stop();
    MakeLevel1(1);
    moveTo(600, 'r', 65, 2);
    stop();
    MakeLevel2(1);
    while(dist < rightWall) // This loop ensures that Robot did not overshoot
    {
        diff = rightWall - dist;
        power = (5 * (diff + 5)) / 3 + 6;
        dist = moveTicks(rightWall, 'r', power, 2);
        MakeLevel2(1);
        dist = 0;
        for (i = 0; i < 2; i++)
        {
            dist += getDistance(2);
        }
        dist = dist / 2;
        while (dist < rightWall);
        if (dist > rightWall + 5)
        {
            dist = moveTicks(rightWall - 10, 'l', 60, 2);
            MakeLevel2(1);
            dist = 0;
            for (i = 0; i < 2; i++)
            {
                dist += getDistance(2);
            }
            dist = dist / 2;
        }
    }
    MakeLevel1(1);
    moveTo(785, 'b', 65, 3);
    stop();
    moveTicks(850, 'b', 50, 3);
    while (light_1 == 0) { // Use the IR sensor to find the stage's side
        move('r', 70);
        delayMICR(400);
        stop();
        move('b', 60);
        delayMICR(600);
        stop();
    }
    void FindStage3()
    {
        int diff = 0;
        unsigned char power = 0;
    }
unsigned char i;
int dist=0;
int rightWall =648;
move('f',50);
delayMicroseconds(250);
stop();
delayMicroseconds(25);
move('l',80);
delayMicroseconds(70);
stop();
delayMicroseconds(25);
getDistance(1);
getDistance(2);
MakeLevel2(1);
getDistance(0);
moveTo(660, 'f', 50, 0);
stop();
move('f', 50);
delayMicroseconds(15);
stop();
MakeLevel1();
moveTo(615, 'r', 65, 1);
move('f', 50);
delayMicroseconds(20);
stop();
while(dist<rightWall){ //This loop makes sure Robot 1 did not overshoot
do{
diff = rightWall-dist;
power = (2*(diff+5))/11 +50;
dist=moveTicks(rightWall,'r',power,1);
MakeLevel2();
} while(dist<rightWall);
if(dist<rightWall){
dist=moveTicks(rightWall-10,'l',60,1);
MakeLevel2();
dist=getDistance(1);
}
MakeLevel1();
moveTo(770, 'f', 80, 0);
stop();
moveticks(840, 'f', 50, 0);
while(light_2==0){ //use IR sensors to find stage 3' wall
move('r',70);
delayMicroseconds(400);
stop();
moveticks('f',60);
delayMicroseconds(10);
stop();
for(i=0;i<4;i++)
move('l',70);
delayMicroseconds(400);
stop();
moveticks('f',60);
delayMicroseconds(600);
stop();
}
//This function is used to make the robot level
void MakeLevel2(unsigned char sen){
unsigned char power=40;
unsigned int value1, value2, adv1, adv2;
unsigned char i;
unsigned char samples=1;
int value3 = 0;
unsigned char times = 0;
bit dir, pastDir;

if (sen == 0)
samples = 2;
do{
value1 = 0;
value2 = 0;
adv1 = 0;
adv2 = 0;
for (i = 0; i < samples; i++) {
    value1 += getDistance(1);
}
adv1 = value1 / samples;
delay10MILL(1);
for (i = 0; i < samples; i++) {
    value2 += getDistance(2);
}
adv2 = value2 / samples;
value3 = adv1 - adv2;
power = (value3 * (value3 + 1)) / 2 + 39;
if (power < 40)
    power = 40;
if (value3 > sen){
    move(dir, power);
    dir = 0;
} else if (value3 < sen){
    move(dir, power);
    dir = 1;
} else if (dir != 'p'){
    times++;
    dir = 'p';
    stop();
    break;
} else{
    pastDir = dir;
    if (value3 > 15 || value3 < -15)
delay10MILL(3);
    else
    delayMICRO(500);
    stop();
} while (value3 > sen || value3 < sen);

} // This function is used to move the Robot to a given distance in mm, in the given direction, with the
// given speed and using the given sensor
unsigned int moveTo(unsigned int distance, unsigned int dir, unsigned char dir, unsigned int speed, unsigned char sensor) {
    unsigned int value1 = 0;
    value1 = getDistance(sensor);
    move(dir, speed);
    if (dir == 'r') || (dir == 'l') || (dir == 'b'){
    while (value1 < distance){
        value1 = getDistance(sensor);
        move(dir, speed);
    } else if (dir == 'l'){
    while (value1 < distance){
        move(dir, speed);
        value1 = getDistance(sensor);
    } else if (dir == 'b'){
    while (value1 < distance){
        move(dir, speed);
        value1 = getDistance(sensor);
    } else if (dir == 'w'){
    value1 = getDistance(sensor);
    return value1;
}
/*
  Value1 Value2

  Diode Backwards - 240-255 0-20
  Diode Forwards - 63-75 0-20
  Capacitor - 200-255 105-170
  Resistor - 150-170 0-20
  Inductor - 30-62 0-20
  Wire - 0-25 0-20
*/

int repeatCount = 0;

// This function will use the step response to get the code that is between each pad and also check to make sure that the got it got is a valid code.
// If the code is invalid it will try again.
void decode(unsigned char array[3]) {
  unsigned char temp, count;
  if (repeatCount > 10) {
    repeatCount = 0;
    LinearActuator[1, 'e'];
    delayMILL(180); // x2 since it always had to do twice, was 25
    LinearActuator[1, 'e'];
    for (i = 0; i < 5; i++) {
      array[i] = getComponent(i);
      delayMILL(5);
    }
    print("\n");
    count = 0;
    for (j = 0; j < 5; j++) {
      temp = array[j];
      for (k = 0; k < 5; k++) {
        if ([temp = array[k]] || (temp <= 0) || (temp > 5)) {
          count++;
        }
      }
    }
    if (count > 1) {
      repeatCount++;
      decode(array);
    } else {
      count = 0;
    }
    sprintf(msg, "%d, %d, %d, %d, %d", (int)array[0], (int)array[1], (int)array[2], (int)array[3], (int)array[4]);
    print(msg);
  } else {
    unsigned char i;
    unsigned int value1 = 0, value2 = 0;
    unsigned char ret;
  }
}
//Stage3.c
//This file contains all the functions used to input the code found
// from stage 1 into stage 3
#include <stdio.h>
#include "Delay.h"
#include "Stage 3.3.h"
#include "Servo.h"
#include "LCD.h"
#include "Fin Assignments.h"

unsigned int count;

//This function is used to move the servo to the home position
// before the code is inputed
void zeroOut (void)
{
    EX0 = 1; // Enable External Interrupt 0
    count = 0;
servoMove('b');
delay1MILL(5);
while(count<count);servoMove('s');
delay10MILL(50);
servoMove('f');
servoMove('s');
servoMove('q');
EX0 = 0; // Disable External Interrupt 0
}

//This function is used to move the servo to a set location after
// the code is inputed
void moveAway (void)
{
    servoMove('b');
delay10MILL(50);
servoMove('s');
servoMove('q');
}

//This function will move the servo so to input the code into
// stage 3
void inputCode (unsigned char codeMsg[5])
{
    bit dir=1;
    unsigned char i;
    EX0 = 1; // Enable External Interrupt 0
    count = 0;
    for(i=0;i<5;i++)
    {
        if(dir)
            servoMove('f');
        else
            servoMove('b');
delay10MILL(75);
count=0;
while(count<countMsg[i]);
servoMove('s');
delay10MILL(50);
dir = ~dir;
count = 0;
}
servoMove('s');
servoMove('q');
EX0 = 0; // Disable External Interrupt 0
}

void EX9(void) interrupt 0
{
    RX0 = 0; // Clear Interrupt Flag
    count++;
}

Robot 2

// Nathan Kabat and Ben Geist
// This is the Main code for IREE Robot 2

// Function Prototypes
void setup(void);

#include <Wire.h>
#include "MotorControl.h"
#include "Navigation.h"
#include "Delay.h"
#include "Pulse Width Measurement.h"
#include "Servo.h"
#include "ilauncher.h"
#include "Stage 2.5.h"
#include "Pin Assignments.h"

// Pin Assignments

/*

PLEASE LOOK AT Pin Assignments2.pn
*/

// Main Program
void main ()
{
    //setup();

    while(start_sw==1);  //Wait for start switch
    LinearActuator3(300,250);  //extend communication actuators
    delay10MILLI(100);  //wait 1 sec
    MoveOutOfWay();  //move to stage 4
    com 2=0;  //signal 2
    LinearActuator1(1,0);  //extend linear track actuators for stage 4
    delay1MILLI(1000);  //wait 1 sec
    shoot();  // Shoot 2 darts at this time
    while(com 3==1 && start_sw==1);  //wait on com 3 or start switch
    FindStage2();  //move to stage 2
    Stage2BATTLE();  //do stage 2
    FindStage4();  //move back to stage 4
    LinearActuator1(1,1);  //extend linear track actuators for stage 4
    shoot();  //shoot last dart
    LinearActuator(65,1);  //retract communication actuators
    while(1);  //wait forever
}

//This function sets up everything
void setup();
{
    unsigned char i;
    CKCON=0x01;  // Double Clocked
    TMD=0x11;  // Timer 0a1 set up in 16-bit Timer Mode
    EA = 1;  // Enable All Interrupts
    ES=0;  // Turn off serial interrupt

    #if defined(ADCRD Read.h) || defined(_Stags 1 Decode Message.h)
    //ADC set up
    ADCON = 0x28;  // Enable ADC Function
    ADCCLK = 0x08;  // Set ADC Clock speed
    ADCON &= 0x08; // Rest ADC Channel Select
    ADCON set up for P1.0 & P1.1
    ADCF |= 0x01;
    ADCF |= 0x02;
    #endif

    #ifdef ilauncheer_h
73 // For iLauncher
74 shooter_1 = 1;
75 shooter_2 = 0;
76 #endif
77
78 #ifdef Mux_Select_h
79 // For Mux Select
80 inhibit = 1; // Disable Multiplexer
81 #endif
82
83 #ifdef servo_h
84 // For Servo Using 16-bit compare mode for 0.02 deg/tick
85 CCAPM0 = 0x00; // 0100 1000 Compare Bit & Match Bit
86 CCAPM1 = 0x01; // 0000 0001 Enable CCP3 Interrupt Bit
87 pca_0 = 1;
88 #endif
89
90 #ifdef stage_3_h
91 // Declare External Interrupt
92 IT0 = 1; // External Interrupt 0 Falling Edge Triggered
93 #endif
94
95 #if defined(_LCD_h) && defined(_TTL_Distance_Sensor_h)
96 // Serial Communicating RX and TX @ 9600 Baud
97 SCON = 0x40; // Mode 2, 8 bit uart transmit only, uses T2
98 REN=1; // Transmit and Receive
99 RCLK=1; // Turn on receive clock in T2CON
100 TCLK=1; // Turn on transmit clock in T2CON
101 // S1 baud rate = fCrystal/(32*(165536 - RCPP2H, RCPP2L)) for x1 mode
102 // S2 baud rate = fCrystal/(16*(65536 - (CCAP2H, CCAP2L)) for x2 mode
103 // 1200 = 28.2076 x 10^6/(16*(65536 - (CCAP2H, CCAP2L))
104 // 2400 = 28.2076 x 10^6/(16*(65536 - (CCAP2H, CCAP2L))
105 // 4800 = 28.2076 x 10^6/(16*(65536 - (CCAP2H, CCAP2L))
106 // 9600 = 28.2076 x 10^6/(16*(65536 - (CCAP2H, CCAP2L))
107 // 19200 = 28.2076 x 10^6/(16*(65536 - (CCAP2H, CCAP2L))
108 // 2400 = 28.2076 x 10^6/(16*(65536 - (CCAP2H, CCAP2L))
109 // TX2 = 0xFF;
110 // RX1 = 0x48;
111 RCCP2HX=0x0F;
112 RCCP2LX=0x0B;
113 // 4800 baud @ 28.2076 MHz
114 TR2=1; // TCON bit to start timer 2.
115 RX = 0; // Clear the receive interrupt flag
116 // ES = 1; // Turn on serial interrupt
117 #endif
118
119 #ifdef servo_h
120 servoMove('h'); // Make sure servo is at the home position
121 #endif
122
123 delay10MILLIS(50);
124
125 #ifdef MotorControl_h
126 // Retract all the Actuators
127 LinearActuator(69, 'a');
128 delay10MILLIS(300);
129 LinearActuator(69, 'b');
130 #endif
131
132 #ifdef servo_h
133 // servoMove('q'); // Set Pulse Width
134 // pca_0 = 1; // Set Servo Signal Pin High
135 #endif
136
137 #ifdef Pulse_Width_Measurement_h
138 // For Sonar Sensor
139 send_range = 0;
140 // For Sonar Sensor using Pulse Width
141 CCAPM1 = 0x03; // 0011 0000 Capture Positive Edge & Negative Edge on P1.7
142 CCAPM1 = 0x01; // 0000 0001 Enable CCP4 Interrupt Bit
143 for(i=0;i<4;i++)
144 getDistance();
145 #endif
// This file contains all the functions used in navigation for Robot 2
#include <stdlib.h>
#include "MotorControl.h"
#include "Delay.h"
#include "Navigation.h"
#include "Pulse Width Measurement.h"
#include "Pin Assignments.h"
#include "RTC.h"

// Function prototypes
void MakeLevel1(char sen);
void MakeLevel2(char sen);

// This function moves Robot 2 out of the way of Robot 1 and
// lines it up to shoot at stage 4
void MoveOutOfWay(){
  int diff = 0;
  unsigned char power = 0;
  int dist = 0;
  int sen = 2;
  int i = 0;
  unsigned int Wall = 495;
  move(1', 75);
  delayOMILL(200);
  stop();
  moveTo(435, 'b', 50, 0);
  stop();
  Turn90Deg();
  MakeLevel2(sen);
  moveTo(305, 'b', 50, 0);
  stop();
  moveTo(Wall-70, '1', 50, 1);
  stop();
  while (dist < Wall){
    do{
      diff = Wall - dist;
      power = (12*(diff+5))/9 + 15;
      moveTicks(Wall, '1', power, 2);
      MakeLevel2(sen);
      dist = 0;
      for (i = 0; i < 10; i ++)
        dist += getDistance(2);
    } while (dist < Wall);
    if (dist < Wall + 5){
      moveTicks(Wall-10, '1', 80, 2);
      MakeLevel2(sen);
      dist = 0;
      for (i = 0; i < 10; i ++)
        dist += getDistance(2);
    } else
      dist = dist / 2;
  } while (dist <= Wall);
  MakeLevel1(1);
  moveTo(415, 'b', 50, 0);
  stop();
}

// This function will turn the robot 90 deg.
void Turn90Deg(){
  move('(', 60);
  delayOMILL(230);
  stop();
}

// This function will fine stage 2 from stage 4
void FindStage2(){
  int diff = 0;
  unsigned char power = 0;
  int dist = 0;
int i=0;
int LeftWall = 485;
getDistance(3);
move('f', 50);
delay(OMILL(150));
stop();
MakeLevel2(2);
move('s', 65);
delayFor(i=0; i<2; i++)
stop();
MakeLevel2(2);
moveto(600, 'f', 50, 3);
stop();
moveto(925, 'f', 50, 3);
stop();
moveto(440, 'l', 50, 2);
stop();
while(dist<LeftWall){
dist=0;
for(i=0; i<2; i++)
dist+=getDistance(2);
}
dist = dist/2;
diff = LeftWall-dist;
power =((12*(diff+5))/9) +45;
moveticks(LeftWall, 'l', power, 2);
MakeLevel2(2);
while(dist<LeftWall);
if(dist<LeftWall+5)
do{
moveticks(LeftWall-10, 'r', 60, 2);
MakeLevel2(2);
dist=0;
for(i=0; i<2; i++)
dist+=getDistance(2);
}
dist = dist/2;
while(dist<LeftWall+5);
}
MakeLevel1(1);
moveto(3090, 'f', 30, 3);
stop();
//This function will find stage 4 from stage 2
void FindStage4(){
int diff = 0;
unsigned char power = 0;
int dist = 0;
int sen = 2;
int l = 0;
unsigned int Wall = 495;
getDistance(0);
moveto(650, 'b', 50, 0);
stop();
moveto(980, 'b', 50, 0);
delay(OMILL(25));
stop();
MakeLevel2(sen);
moveto(980, 'b', 50, 0);
stop();
MakeLevel2(sen);
moveto(Wall-70, 'l', power, 2);
while(dist<Wall){
diff = Wall-dist;
power = ((12*(diff+5))/9) + 45;
moveticks(Wall, 'l', power, 2);
MakeLevel2(sen);
dist=0;
for(i=0;i<2;i++)
    dist+=getDistance(2);

while(dist<2);
if(dist=Wall+5){
    moveTicks(Moment-10,'r',60,2);
    MakeLevel2(sen);
    dist=0;
    for(i=0;i<2;i++)
        dist+=getDistance(2);
}

MakesMove(1);
motor(1070,'b',50,0);
stop();

//This function is used to level out the Robot
void MakeLevel2(unsigned char sen){
    unsigned char power=40;
    unsigned int value1,value2,adv1,adv2;
    int samples=1;
    int value3 = 0;
    unsigned char times=0;
    bit dir, pastDir;

    if(sen == 0)
        samples = 2;
    do{
        adv1=0;
        adv2=0;
        value1=0;
        value2=0;
        for(i=0;i<samples;i++)
            value1+=getDistance(1);
        adv1=value1/samples;
        //delay10MILL(2);
        for(i=0;i<samples;i++)
            value2+=getDistance(2);
        adv2=value2/samples;
        value3=adv1-adv2;
        power=(45*(value3+1))/39;
        if(power>80)
            power=80;
        else if(power<40)
            power=40;
        else if(value3<sen)
            move[1],power;
        dir=0;
    }else if(value3<sen)
        move[5],power;
    dir=1;
}
if(dir=pastDir)
    times++;
else
    times=0;
if(times>0){
    stop();
    break;
}
pastDir=dir;
delayMICR(130);
stop();
while(value3<sen)|value3<sen);
// This function is used to move the Robot to the given distance and direction with the given speed, and using the given sensor
unsigned int moveTo(unsigned int distance, unsigned char dir, unsigned char speed, unsigned char sensor) {
    unsigned int value = 0;
    move(dir, speed);
    if (dir == 'l' || dir == 'r') {
        while (value < distance) {
            value = getDistance(sensor);
        }
    } else {
        while (value < distance) {
            value = getDistance(sensor);
        }
    }
    return value;
}

// This function will move the Robot very slowly to the given distance with the given speed and using the given sensor
unsigned int moveTicks(unsigned int distance, unsigned char dir, unsigned char speed, unsigned char sensor) {
    unsigned char times = 0;
    value = getDistance(sensor);
    if ((dir == 'l' || dir == 'f') || (dir == 'b')) {
        while (value < distance) {
            if (value > distance) {
                break;
            } else {
                move(dir, speed);
                delayMICR(400);
                delayMILL(10);
                value = getDistance(sensor);
                if (speed <= 40) {
                    times++;
                } else {
                    times = 2;
                    break;
                }
            }
        }
    }
    return value;
}

// This function is used to make very fine tune adjustments to the robot
void makelevel1(char sen) {
    char power = 35;
    unsigned int value1, value2, advl, adv2;
    unsigned char ch1;
    int value3 = 0;
    do {
        advl = 0;
        value1 = getDistance(sensor);
        return value1;
    }
    while (value1 < distance);
//iLauncher.c
//This file contains all the functions used to control the iLauncher

#include <at89c51cc03.h>
#include "Delay.h"
#include "iLauncher.h"
#include "Pin Assignments.h"

//This function will shoot the given number of missiles
void shoot(unsigned char rounds)
{
    unsigned char i;
    for(i=0;i<rounds;i++)
    {
        shooter_1 = 0;
        delay10MILL(200);
        while(shooter_2 == 0);
        shooter_1 = 1;
        delay10MILL(50);
    }
}
//Stage 2.c
//This file contains all the function used to do stage 2
#include <at89c51rc03.h>
#include <stdio.h>
#include "Stage 2.h"
#include "ADCRead.h"
#include "Flin Assignments.h"
#include "LCD.h"
#include "Servo.h"
#include "Delay.h"
#include "RTC.h"
#include "Navigation.h"
#include "MotorControl.h"

//Function prototypes
unsigned char readSAG(unsigned char samples, unsigned char sensor);
void hitSaber(void);

//This function will do the 30 sec lightsaber better for stage 2
void Stage2BATTLE (void)
{
    unsigned char sensor0OFF, sensor1OFF;
    unsigned char sensor0New, sensor1New, swings=0;
    unsigned char threshold = 6;

    servoMove('a'); // Move servo to ready "arm" position.
    move('f',50);
    while(prox1 == 1);
    startRTC();
    stop();
    delay10MILL(20);

    // Take sensor OFF values
    sensor0OFF = readSAG(2,0);
    sensor1OFF = readSAG(2,1);

    while(seconds < 29)
    {
        sensor0New = readSAG(2,0);
        sensor1New = readSAG(2,1);

        if((sensor0New-sensor0OFF-threshold)|| (sensor1New-sensor1OFF-threshold) ||
           (sensor0New-sensor0OFF+threshold) || (sensor1New-sensor1OFF+threshold))
        {
            hitSaber();
            swings++;
            if(swings>3)
                break;
        delay10MILL(200);
    }

    if(swings<4)
        hitSaber();
    stopRTC(); // Stop RTC

    // Move Servo Arm back home
    servoMove('h');
}

//This function used the 8 bit ADC function to read the magnetic field
unsigned char readSAG(unsigned char samples, unsigned char sensor)
{
    unsigned char tmp;
    unsigned char i;
    tmp = 0;
    for(i=0;i<samples;i++)
    {
        tmp += ADC8BIT (sensor);
72    }
73    tmp = tmp/samples;
74    return tmp;
75    }
76    //This function moves the servo so that the servo arm hits the saber
77    void hitSaber(void)
78    {
79        servoMove('s');
80        while(vib_sens != 1);
81        servoMove('s');
82    }