SAFIRE

EE 382 Final Report
Team D - SAFIRE
Superior Autonomous FireFighting Robot Engineers

Nathan Miller, James Farrell,
Allen Woo, Jack Ramzel

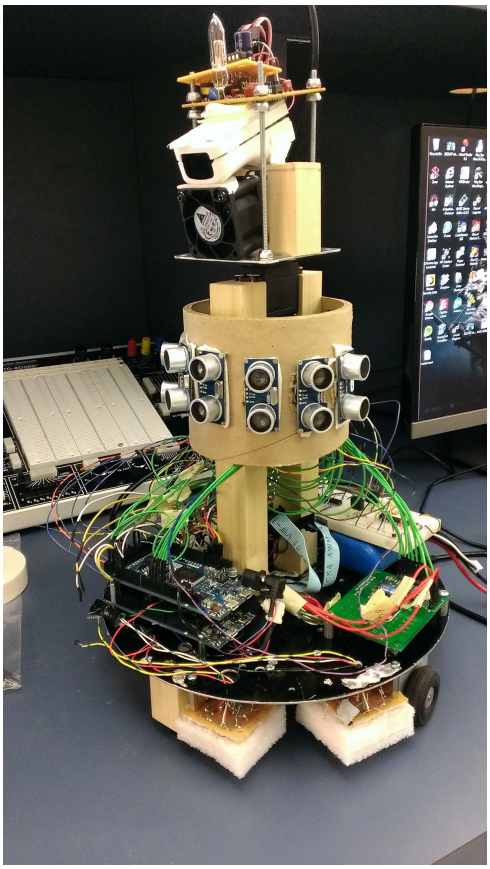

Figure 1 - Robot 0x46 0x49 0x52 0x45 0x21 a.k.a FIRE!

Table of Contents

List of Figures

Abstract

For the Introduction to Design (EE 382) class, groups of students were given two options. One option was to build a beacon finding robot and the other was to build a fire fighting robot. Team SAFIRE chose to design a robot capable of completing the firefighting challenge for the RoboRAVE competition. In order to be considered a success, the robot must extinguish four flames within a 12'x8' field in under three minutes. Three of the candles are blocked by walls that needed to be navigated around. The field is marked by a white border tape with a black stripe.

The robot was constructed on a metal chassis connected to high-torque motors. Sensors included ultrasonic sensors, line sensors, a Wii remote sensor, and a UV-Tron ultraviolet sensor. A panning servo system moved the Wii sensor and UV-Tron sensor in addition to the fan extinguishing system. The Arduino Due was used as the primary interface between all components and the motors. Two different search programs were written in C using the Arduino IDE environment. The robot was not completed for the scheduled demonstration on April 29th. However, the robot was ready to compete for the international RoboRAVE competition on May 3rd. Team SAFIRE placed 4th in the international competition of 25 teams.

Introduction

Team SAFIRE (Superior Autonomous Firefighting Robot Engineers) formed out of the requirements for EE 382 (Intro to Design). The team selected the challenge of designing a robot (see Figure 1 above) capable of completing the RoboRAVE firefighting challenge by autonomously navigating a 12'x8' field and extinguishing four flames within three minutes. The candle flame was held at a height of 10" and the robot was required to be within 8" of the candle base to extinguish them. A ring of white 1" wide tape was used to indicate the 8" distance. The field had three walls 18" wide and 16" tall. The field had a 3" border consisting of a white-black-white tape, each color an inch wide. Hitting a wall carried no penalty, however, hitting or knocking over a candle did. This made it important to prioritize an accurate obstacle avoidance system.

In order to construct a fully autonomous robot, a variety of sensors for range detection, line detection, and flame detection were utilized (see Figure 2 below). The range detectors consisted of an array of seven SRF04 ultrasonics, arranged in the front of the robot giving a 180° field of view. The white/black line detector was designed specifically for this project. The sensor consists of an LED that reflects off a surface and a photoresistor that reads the reflected light level. In order to increase our rate of success, two flame detection methods were implemented at once. The first was a Hamamatsu UV-tron, which detected ultraviolet radiation. The second was an IR camera used in a Nintendo brand Wii-remote to track IR emitters for the gaming console.

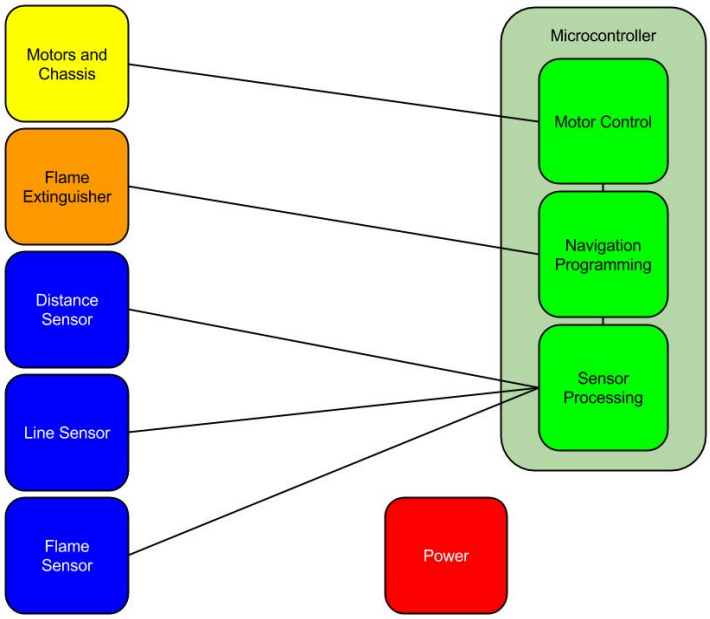

Figure 2 - Block Diagram

All of the components and sensors are housed on board a round 10” diameter chassis. The system is powered with 13.2v provided from two 6.6v batteries in order to provide ample power to all components. The flame sensors and fire extinguishing system are housed atop a servo so that they can be panned over 180° to detect the flames.

The robot would perform what was named a center line search algorithm to locate the flame. This method required position tracking on the field to allow the robot to return to the center after extinguishing a flame or inspecting a barrier the programming deemed suspicious. Additionally, a random searching algorithm was developed due to time constraints. The programming was all done in the Arduino environment which is a C based language. The programming contained a mix of developed code and preconstructed libraries for various sensors and functions.

The components used to form a functional and effective firefighting robot will be described in the sections below.

Design Aspects

Chassis

The SAFIRE chassis (see Figure 3 below) was designed around a 10” circular disk with an aluminum base, two Maxon DC motors with 14:1 gearheads and optical encoders. Two steel ball casters were used at the front and back of the base for stability. The center rise to house the ultrasonic and fire detection circuits were supported by two upright 1”x1”x9” wood beams. Atop the wood is a 5v digital servo which pans the UV-tron, Wii-Mote Camera, and the extinguishing fan mounted to a metal platform.

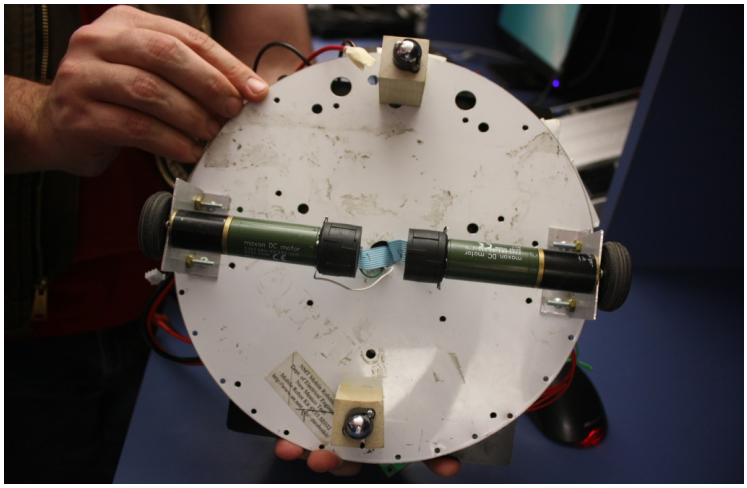

Figure 3 - Locomotion System

Flame Sensors

In order to accurately locate the flames, two fire detection components were implemented; the UV-Tron and the Arduino “Hack” for the Wii-Remote IR camera. The UV-tron is a highly accurate circuit designed to detect flame up to about 25’ away (see Figure 4 below). The circuit is powered by 5V and outputs a 10ms signal when 3 UV waves are detected. The main issue that had to be addressed was the field of view for the sensor. While its vision is dependant on orientation, it is possible to obtain 180° field of

view. The sensor needed to be shielded so the field of vision would be narrowed to roughly 30° in front of the robot. In the end, a simple foam sleeve was used with a forward facing opening.

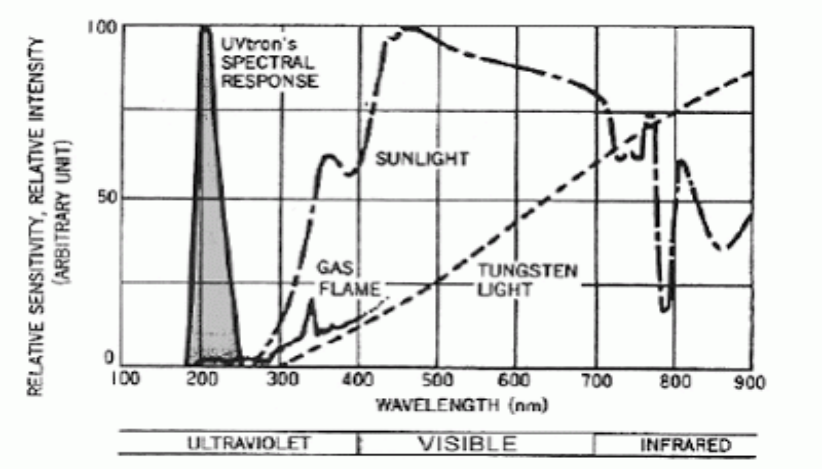

Figure 4 - UV-tron Spectrum

The Wii-Mote IR sensor (see Figure 5 and 6 below) was a modification to the Nintendo Wii-Mote that allowed the robot to use its ability to track up to four of the brightest IR sources in an X and Y coordinate plane. Using a simple timing circuit, the robot was able to receive accurate X and Y coordinates from the camera up to 8'. Once the Wii-Mote camera was implemented, the best mounting positions were tested. It was discovered that the best position was placing the camera slightly angled downward towards the flame. This made it much more accurate and consistent. As a result we chose to mount the camera above our extinguishing fan to provide the extra height.

In order to use the Wii remote, a clock circuit had to be constructed. The circuit was designed to create a square wave at 25 MHz. The circuit diagram and wiring diagram can be seen if Figure 6 below.

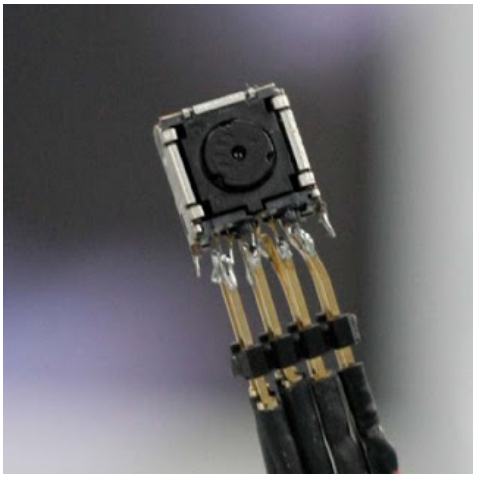

Figure 5 - Wii IR Camera Module

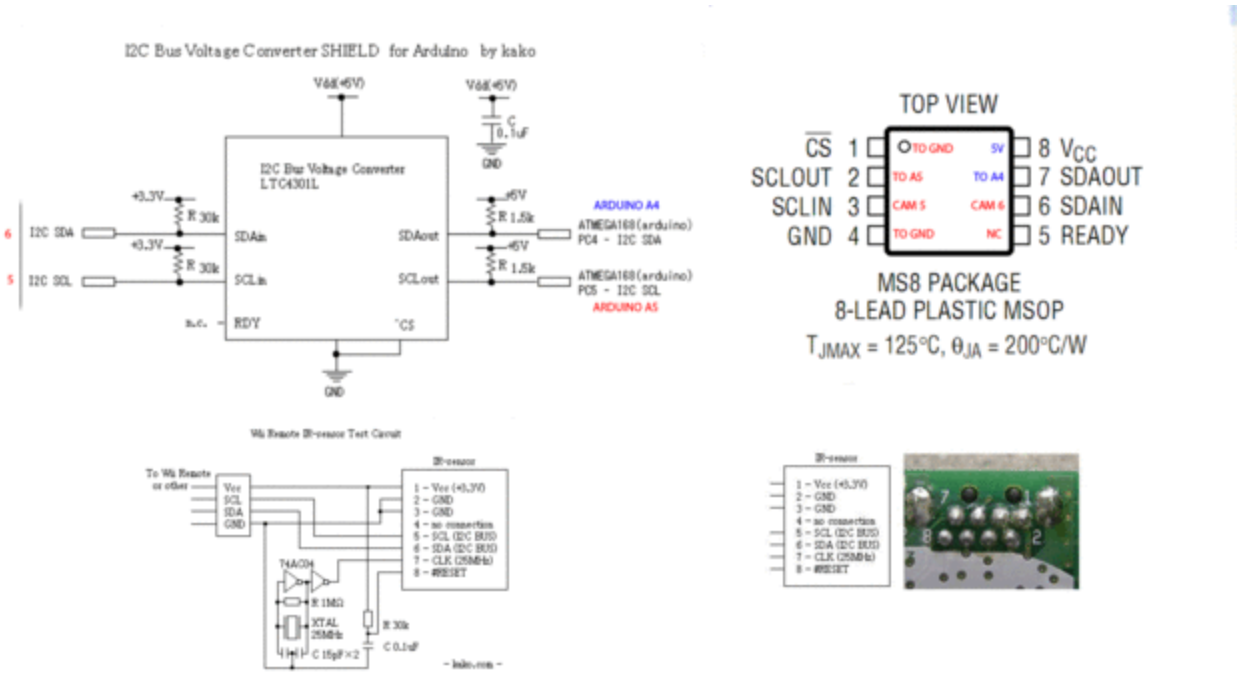

Figure 6 - Wii Clock and Wiring Diagram

A third flame sensor was the dual IR LED sensor. It was the most compact of the flame sensors. It was designed to simply detect and compare infrared light emitted by the

candles. The circuit was composed of two IR LEDs, two trans-impedance amplifiers and two voltage dividers. The device could sense small IR emissions, like the candles, from approximately three feet. It could differentiate between which of the two sensors was closer to the emission and adjust until the the emission was directly in front of the sensors. Due to time constraints, the dual IR LED sensor was not implemented in the final design.

Range Sensors

In the playing field, there were three walls whose dimensions were 18” wide by 16” tall. These walls would obstruct the path and view of the robot. The robot must be able to detect these walls and navigate around them. The solution implemented was seven SRF04 ultrasonic sensors (see Figure 7 below). The detectors emit a high frequency pulse and measure the time it takes for the pulse to return to the sensor. The Arduino then takes that time and calculates what distance it corresponds to in centimeters.

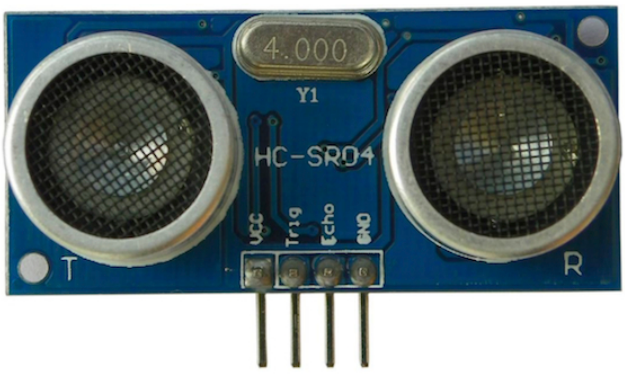

Figure 7 - SRF04 Ultrasonic Sensor

Initially, the plan was to implement three ultrasonic range sensors. However, after testing, it was determined that the initial plan would not be sufficient due to frequent

problems caused by blind spots. To remedy this, the group purchased seven new SRF04 sensors and mounted them in a semicircle around the front half of the center structure on the robot. This allowed the robot to have a sensor every 30°, which eliminated the blind spots. Both the SRF04 sensors, that the group started with, and the SR04 sensors purchased later, had 30° fields of view. This meant that an ultrasonic emission from the sensor could bounce off something up to 15° off axis and accurately return the timing information necessary for the distance calculation. This also meant that with all seven ultrasonics running, the robot's field of view was 180° on the front side. When it was decided that the robot should have seven ultrasonic sensors, the group also considered the possibility that they might interfere with each other, due to their proximity. However, after testing, it was discovered that there was no interference between sensors by staggering which ultrasonic was in operation (so that no two close ultrasonic sensors were used in succession). As with all 5 volt inputs to the Arduino Due, the SRF04 outputs were sent through a SN74LVC245AN (level shifter chip) to drop the voltage from 5V to 3.3V. The results from testing the ultrasonic sensors can be seen in Figure 8 below.

SR04 Brand	Near Range (42cm)	Far Range (120cm)	Placement
Unknown	42cm ±1cm	115cm ±3cm	not used
Vivotech	42cm ±1cm	110cm ±1cm	forward most sensor
SainSmart	43cm ±1cm	120cm ±1cm	90° sensors

Figure 8 - Ultrasonic Sensor Test Results

Line Sensors

In order to have a frame of reference for the the operating field, the robot needed to know when it came across a boundary. The boundaries for the edge of the field were marked by a black line in between two white lines. There are also white rings around each of the candles to mark the maximum distance the robot can be at to extinguish a flame.

To detect the lines, three reflectivity sensors were custom made. The sensors consisted of an LED, an instrumentation amplifier circuit, and a photoresistor (see Figure 9 below). The LED shines on towards the ground and the photoresistor measures the amount of light that gets reflected back and the amplifier translated it to a usable value. The higher the voltage from the photoresistor, the brighter the surface. However, this signal tended to be weak coming directly from the photoresistor which is why the instrumentation amplifier was implemented.

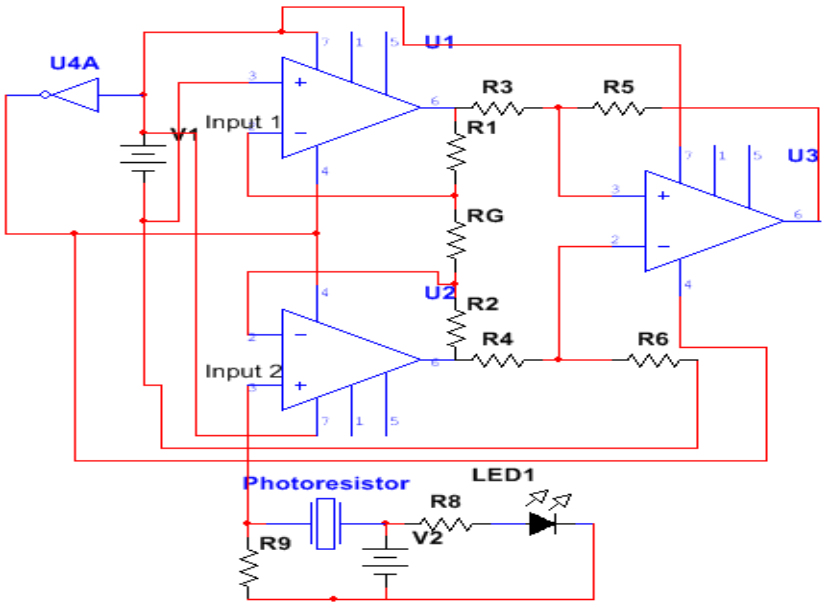

Figure 9 - Custom Line Sensor

The reason an instrumentation amplifier was chosen instead of a regular op-amp was because of the high CMRR. CMRR, or Common Mode Rejection Ratio, is the rejection of unwanted signals. Typically, the higher the CMRR the better. The configuration of an instrumentation amplifier enables a high CMRR. The photoresistor measures the reflected light and sends the signal through the instrumentation amplifier. The signal from the instrumentation amplifier is then read by the analog inputs on the Arduino. The Arduino measures the highest and lowest values for the first five seconds and uses those values as a reference for the current run. Highest values correspond to the white lines and the lowest values correspond to the black line. The values in between correspond to the carpet and will be ignored. A technique that was used to help the calibration was to calibrate the sensor with a gray sheet of paper. This allowed the robot to look for a value that was slightly lower than what the white would represent. The effect was that when the sensor saw a white line, the reflection was more than enough to trip the sensor without fault.

During testing, errors occurred such as a much lower value than expected. The voltage was divided instead of multiplied when run through the instrumentation amplifier. This was determined to be because the photoresistor was connected to “Input 1” on the diagram in Figure 5. Switching the photoresistor to “Input 2” fixed this problem. During competition, changes in the thresholds and initial calibration had to be modified in order to ensure a sufficient white line or black line signal.

Microcontroller

When the project was started several ideas were tossed around for the microcontroller. Several options included: the HC12 board from previous classes, the Raspberry Pi, and Arduino. The decision was made based on the ability to use the Wii-Mote camera with the Arduino. As fas which Arduino to use, one board in particular stood out. The Arduino Due has 54 digital IO pins and 12 Analog IO pins which provide ample pins for sensors and controls (see Figure 10 below). The clock speed of the onboard processor was also much higher than any comparable Arduino, at 84 MHz. In the end, the Arduino Due was chosen.

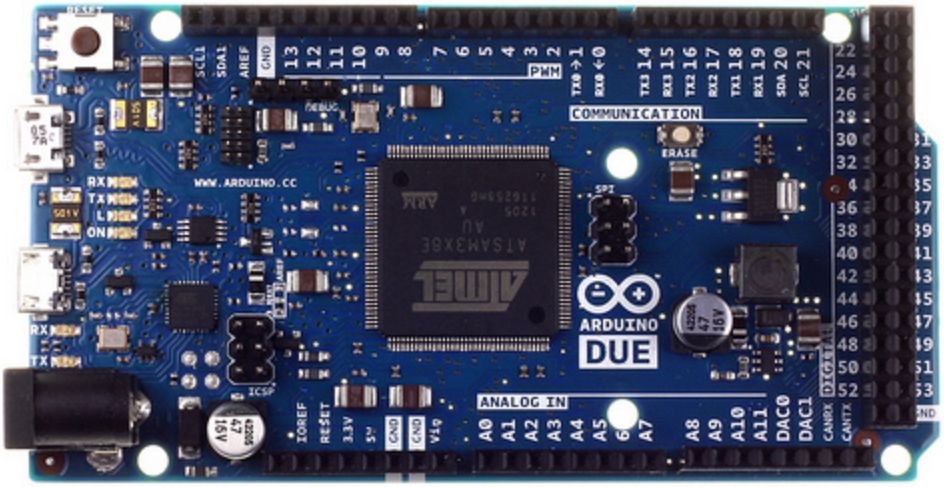

Figure 10 - Arduino Due Microcontroller

The only downside of this decision was that the Due, unlike most Arduinos, runs its pins at 3.3V instead of 5V. This presented a problem in that most sensors provided a 5V response (as with the ultrasonic sensors and UV-tron). To remedy this problem, several SN74LVC245AN (octal bus transceiver) logic level shifting chips were used. This allowed

dropping the 5v signal to 3.3v without losing the incoming data. The circuit configuration used can be seen in Figure 11 below.

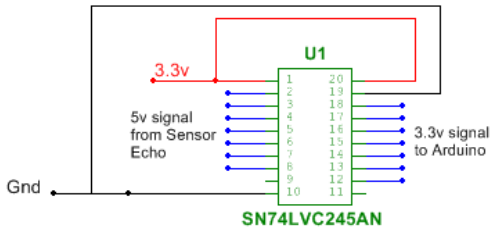

Figure 11 - Logic Level Shifter

Flame Extinguisher

The flame extinguishing subsystem was composed of a small but high powered fan driven at maximum available voltage. After testing several small computer fans and motors, a small computer fan was chosen that accepts 12v and uses up to 0.60A. Due to this high power draw, the Arduino Due was unable to provide the power directly. This meant that some sort of switching circuit need to be developed. Both MOSFET switches and relays were considered. However, in the end it was decided that a MOSFET would be needed to toggle the relay in order to reliably turn on the fan due to the limited voltage level and current output of the Arduino Due pins (see figure 12 below).

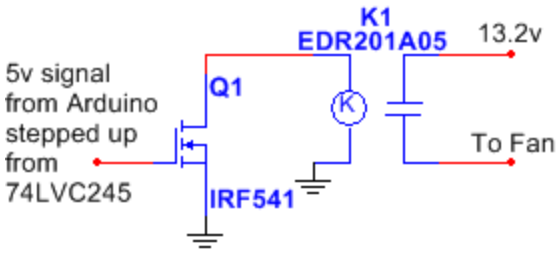

Figure 12 - MOSFET/Relay Circuit

To provide enough voltage to toggle the MOSFET, one of the SN74LVC245AN was used to step up the 3.3V from the Due to 5V. The group originally tested this using a relay in place of the MOSFET but the Due alone was not enough to toggle the relay. However, the voltage and current supplied by the chip was not enough to fully turn on the MOSFET, so the MOSFET was used to turn on the relay.

Power

In order to safely power the Arduino, the group needed to supply between 7v and 20v. To do this, two 6.6v A123 LiFe batteries (see Figure 13 below) were placed in series to provide 13.2v and 2300mAh. The only components that take the full 13.2v input were the Arduino Due, the motors, and the fan. The batteries provide roughly 90 minutes of continuous use. These batteries were the first choice because of their total output and availability. In order to power the 3.3V and 5V systems, the voltage regulators built into the Arduino Due were utilized. However, in testing, capacitors were added to the outputs in order to minimize fluctuations from the voltage regulators. Additionally, one Due was used to provide 5V and the other 3.3V in order to reduce strain on each Due.

Figure 13 - A123 LiFe Battery

Motors

Two Maxon DC Motors (see Figure 14 below) were obtained from the robot jail. They were already equipped with 14:1 ratio gearheads and HP optical quadrature encoders. The motors are capable of operating at 18v which is significantly above the 13.2v battery supply. The motors are able to supply much more torque than what was required to move the robot granting the motors the ability to operate at varying PWM inputs. The motors are relatively small for their output and easily fit under the platform without causing issues. Considering the size of the motors the group had the option to use large wheels but opted to stay with the 2” diameter wheels that were also already with the motors.

Figure 14 - Maxon DC Motor

H-Bridge

The H-Bridge used to control and power the motors was of standard configuration (see Figure 15). Its most impressive aspect is that it can run two channels at up to 46V, 2A.

However, this was well past what was being using at 13.2V making it more than sufficient. The board itself had a positive and negative line out to power each motor. Each motor also had an enable pin and a second set of pins labeled (+) input and (-) input. Its control protocol was implemented by the enable and input pins. The enable pin, when high, would allow control actions to take place. Then pulling the input pins high or low would allow control over the motors on a lower level. this can easily be seen in Figure 16 below.

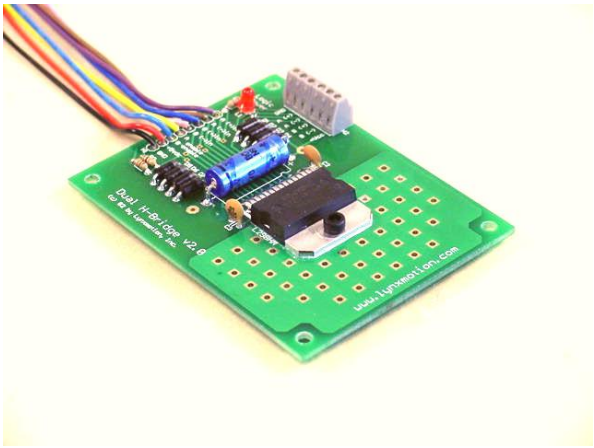

Figure 15 - L298N H-Bridge

E	+	-	Motor Status
L	X	X	Power Off
H	L	L	Stop(Brake)
H	H	L	Rotate CW(Fast)
H	L	H	Rotate CCW(Fast)
H	H	H	Stop(Break)
P	H	L	Rotate CW(Slow)
P	L	H	Rotate CCW(Slow)

E=Enable H=High L=Low P=Pulse X=Don't Care

Figure 16 - H-Bridge Truth Table

Even more precise actions can be implemented through the H-bridge via PWM. Things such as cruise control, acceleration, and speed changes can be done through PWM utilization alongside encoder implementation. The group in particular, needed and implemented cruise control to make the motors match pace with each other. This helped the robot perform precision steering and would have been particularly useful if it had been possible to implement the mapping algorithm.

Encoders

The encoders, like the gearheads, were already mounted when they were obtained with the motors. They pulse at 200 counts per revolution (which account for ten high state changes) HP quadrature encoders and with the 14:1 ratio of the gearheads, every rotation had ~28,000 state changes per motor. Each encoder has two internal sensors that are mounted side by side. This is done so that the signals that are output in the two data lines (one from each sensor) are offset by 45 degrees. With the offset monitored carefully it was possible to obtain directional information that can help guide the robot.

The data from each of the encoders needs to be divided so the microcontroller can read and use it without much effort. For this task it was decided that a divider of 16 would slow the information down enough to be read by the microcontroller. In order to obtain a division of 16 the group decided that either a ripple counter circuit or an FPGA would be able to handle the task. The flip-flop based ripple counter would use four IC's (74HC4040), one for each line of incoming data (see Figure 17 below). However, a problem arose in its

operation. Implementing the ripple counter unfortunately came at the cost of the directional information as the phase information was not retained when changing directions. The solution to this problem was to replace the ripple counter with an FPGA. The group was able to obtain the DE0-nano, a Cyclone 4 based FPGA. By coding in four 4-bit registers that increment every high edge state change and outputting the most significant bit of each register, the speed of the encoder outputs are slowed by 16. By having particular registers increment based on a check of highs and lows in tandem directional information can be retained and utilized. However, due to time constraints, the DE0-nano was not able to be implemented in time for the competition.

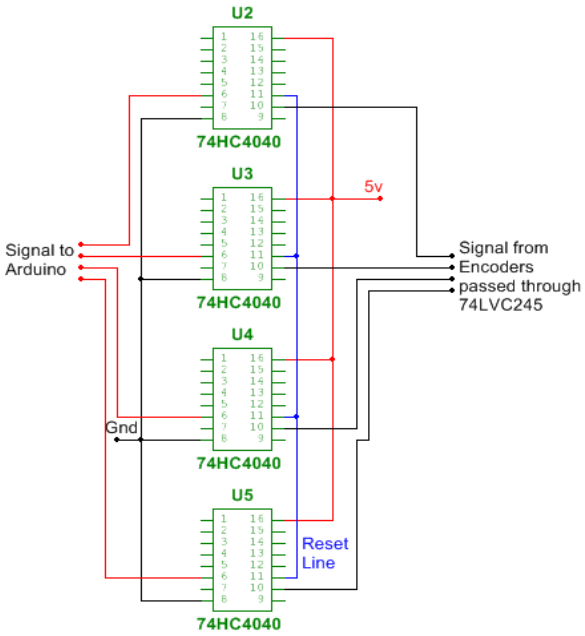

Figure 17 - Ripple Counter

Fuses

In order to protect the teams Arduino and other circuitry two fuses were added in strategic locations to attempt to protect the most important components. The Due is

capable of pulling up to 800mA safely so a 750mA fuse was placed in line before the microcontroller to prevent its potential damage. The group also decided that the batteries represented the next most important component in need of protection. After estimating what was believed to be a modest current consumption, a 3A fuse was decided upon and placed between the batteries and everything else. While there were not any blown fuses due wiring issues, one of each fuse blew up while unplugging the robot or accidentally touching hot wires. Thus the fuses served their purpose by protecting the robot from human error.

Programming

The programming for the Firefighting robot was divided into two separate programs, one for a master microcontroller and another for a slave microcontroller. Both microcontrollers were the Arduino Due, utilizing the Arduino IDE programming environment (utilizing C and C++). The master microcontroller was programmed to receive multiple inputs from sensor data and react based upon that information. The slave microcontroller's purpose was to read data from the encoders and send the information upon request to the master microcontroller. The microcontroller's and the programs associated with them will be described in further detail below.

The organization of the program was designed to maximize readability by separating the repeating sections of code into functions. The functions each perform a specific task. Changing variables, needed by multiple functions, were declared as global variables to ensure proper scope across all functions. Temporary variables were declared within the needed scope. As needed, variables were initialized to ensure desired values would be set to known values. Code was commented as needed to allow for easier troubleshooting.

Master Arduino Due

The Master Arduino Due was responsible for analyzing the sensor data and responding accordingly. First, the program initialized required variables and included desired libraries (see Appendix A below). Next, in the setup loop, the program begins communication between the various sensors and the I2C. The I2C is connected to the

slave Arduino Due and to the Wii camera. The program waits in the setup loop until the start button is pressed and then finishes the setup commands.

The main loop simply calls the desired programs in a systematic order. Each function is described in further detail in Appendix A below. The sensor functions were designed to check the values, request data from the desired sensors and save the data to the global variables. The random search and robot search algorithms called desired response functions based upon the data information. The response functions controlled the motors, servo, and fan functions.

The robot search algorithm was designed to perform a center search procedure. The robot would start in the corner and proceed to the center line. The robot would then drive up and down the center line. A fire override would take place whenever the UV sensor or the Wii camera detected a fire. Additionally, suspicious unexplored walls would be examined to check for hidden candles. In order to keep track of position, basic trigonometry was performed to keep track of x and y coordinates as well as angular direction. This information was derived from the encoder distance data.

The random search algorithm was developed as integrating the hardware with the programming became very challenging and time consuming. The random search algorithm would simply avoid obstacles, stay within the field boundary, and look for fire. Once fire was detected, the robot would home in on the fire and put it out. The disadvantage of this simple program was that there was no guarantee all areas of the field could or would be searched thoroughly.

Both search algorithms set flags and variables to determine desired speeds and direction of the robot. The speed of the robot was controlled using a PID (proportional integrative derivative) method of controlling the speed of the motors. Using trial and error, values for the PID were set to allow a nearly constant speed from both wheels. The result was that the robot had little left or right drift when travelling straight.

The Wii camera and slave Arduino Due were connected using the I2C communication protocol. When information from one of the slave devices was needed, the program utilized the Arduino wire library to request and receive the information. The information was analyzed and saved to the appropriate variables.

Prior to writing the center search algorithm, the concept of the searching method was tested theoretically using Matlab. See Appendix C below for the Matlab program code, Figure 18 below, or go to http://goo.gl/YKn7G9. The program creates a “playing field” using preset barriers. The robot then searches the field, navigating barriers and staying within the line boundaries. Once the opposite side of the field is reached, the robot reverses direction and travels back down the middle of the field, again avoiding barriers. The test program performed as expected and demonstrated the possibility of utilizing a center line search algorithm.

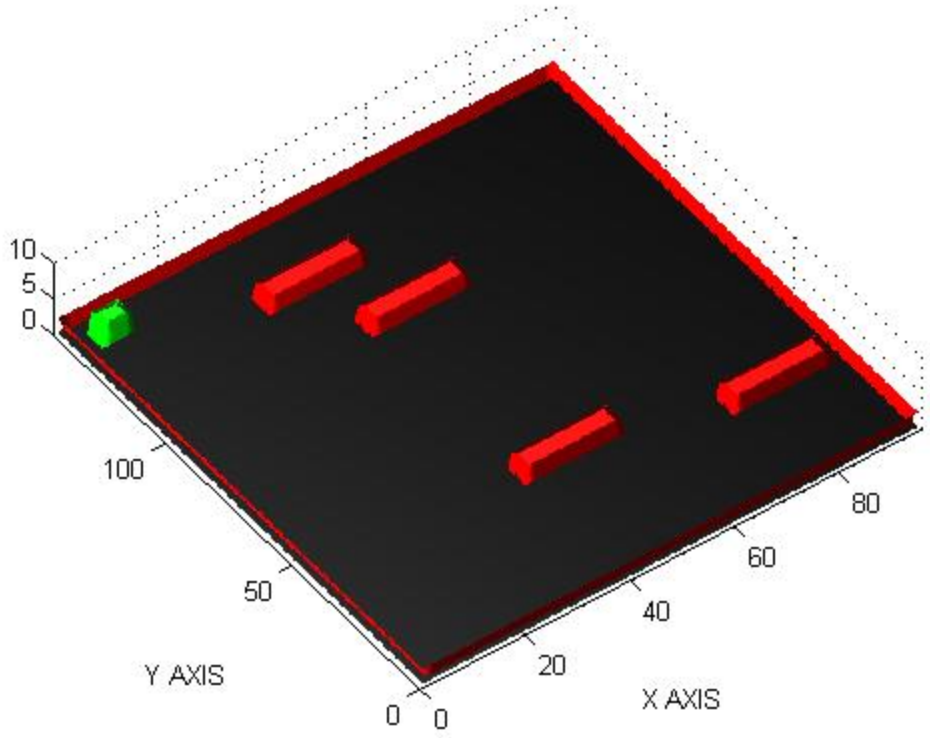

Figure 18 - Matlab Simulation of Center Line Search Program

The Matlab simulation provided the building blocks that were implemented within the robot search algorithm. The additional components that were added provided overrides for fire detection and suspicious barrier exploration. Whenever a fire was seen, the robot was programmed to go directly towards the fire and extinguish the fire once within the 8 inch circle. Having kept track of location information, the robot would then return to the center line search algorithm. The suspicious barrier mode had lower precedence than the fire detection but would override the center line algorithm. The concept was to circle any barriers that were within the boundaries of the field, as determined by the ultrasonic

sensors, and extinguish any observed flames.

Due to hardware issues and time constraints, the center search algorithm was abandoned before completion. The finished code is included in Appendix A. Instead, a random search algorithm was incorporated. The random search algorithm was designed to keep the robot within the boundaries at all times and avoid obstacles. Whenever a boundary or barrier was detected, the robot turned a predetermined amount to avoid the obstacle or to stay within the boundary. When fire was detected, the same algorithm as in the line search was used. The fire detection took precedence over the random search algorithm. The line sensors again detected when the robot was within range of the candle and the fire extinguishing system was activated.

The fire detection system was comprised of the Wii camera and the UV-Tron. The Wii camera data was obtained using the I2C. Only the x coordinate of the brightest flame was used as less bright flames and y coordinate information did not matter. When the robot was relatively centered on the candle, the robot would continue forwards. If the candle moved off center, the robot would turn accordingly to correct. Additionally, in case the robot over corrected or was moving too quickly, the last location of the candle within a short period of time was remembered and the robot turned in the suspected direction of the last candle. The UV-Tron was connected to an interrupt. The interrupt would run whenever a pulse was detected from the UV-Tron. The pulse would indicate a flame detection. However, this process was slow and was ultimately used as a backup sensor to help ensure positive fire detection.

The fire extinguishing system panned the servo mount and turned on the fan motor for a predetermined amount of time. After attempting to blow out the candle, the program would first return to the fire detection portion of the algorithm. If the fire was still detected, the extinguishing system would run again. The entire process would repeat until the robot was stopped by resetting the microcontroller.

Slave Arduino Due

The Slave Arduino Due was designed with a single purpose of detecting pulses from the encoders, saving the information, and then outputting the information to the Master Arduino Due via I2C (see Appendix B below). Due to the computational power required by a high frequency of encoder pulses, the Arduino Due was dedicated to the encoders. The encoders each outputted two signals out of phase. Whenever a rising or falling edge was detected by the Arduino Due, the encoder count was incremented. The count was stored in the memory of the Due until requested by the Master. The Master Arduino Due interpreted the counts by scaling them based upon the number of counts per revolution generated by the encoders. As the number of counts was still to high for the slave Due, an array of ripple counters reduced the number of pulses by a factor of 16, which was accounted for in the Master Due interpretation of the data.

In testing the search algorithm, there were several bugs that will need to be worked out. First, the line sensors require calibration. At the RoboRave competition, the line sensors had difficulty in untested environments. Further calibration and testing would be necessary to ensure solid operation from the sensors. Several bugs in the code resulted in

periodic spinning motion of the robot. With more troubleshooting and testing, these errors can be corrected.

Conclusion

Even though the robot was never fully functional, it still managed to place 4th in the International RoboRAVE Firefighting Competition thanks to consistency in blowing out the first candle. This success is also due to having put in the effort to complete the pre-event points; the paper, video, and sponsorship letters that were covered in the competition rules.

In the end, the center line search algorithm, that was believed to be very successful approach, had to be abandoned. To program such a complex code and with the last minute hardware failures that were experienced required too much time. Accuracy had to be sacrificed for a chance at multiple candles.

The Arduino Due was a very reliable and easy to use microcontroller, but it is recommended to consider a platform that has more refined I2C communication lines. The Due, as was found out late in the semester, has some fairly prevalent issues with consistency in these channels.

The Encoders were another challenge themselves with roughly 30,000 counts per revolution, they sent much more data than the Arduino could reliably handle at higher speeds. Several methods were tried to remedy this problem, unfortunately some of the methods were too late in coming to be implemented.

The Ultrasonic range sensors had no real issues individually. When seven were implemented at once, they started to cause a noticeable delay. Each sensor could take up to 32ms to complete its function. So if no sensors had a return signal, the robot would spend approximately 0.2 seconds just checking the range around the robot. During the

competition, the group choose to disable a few of the sensors that were not necessary to function outside the centerline search.

The UV-tron was originally planned to be used as a confirmation for the Wii-Mote. When the camera failed, the group attempted to use the UV-tron as the primary detection source. However, it was just too slow to be used effectively.

All in all there was a massive amount about troubleshooting, integration, and teamwork on this project that will give us a good basis to start our senior design class in the right foot.

Future Work

In the future, several important features and lessons will be implemented to build a better firefighting robot. First, a simpler design will be implemented. While in theory and separate testing, the sensors performed exceptionally, when finally integrating them all onto one device, the challenges proved time consuming. Secondly, a simpler initial search method will be used. Many hours were used writing code to keep track of positional information. In the end, this program had to be laid aside due to time. Starting simple and working up to a more complex algorithm would have saved much effort. Third, using common components rather than custom or hacked components could save time and effort. Although the Wii camera was potentially extremely powerful and useful, it required extra effort in order to implement. Even though the analog component was a requirement, it made building the line sensors more challenging and potentially less accurate than conventional products. Finally, the entire project should be started considerably earlier. The difficulty of the programing dictates a longer period of time needed for the process. Thus, future projects would dedicate more time to the programing in order to produce effective code.

Budget

Parts that were available to us at no cost.

Part	Quantity	Price(each)	Total Cost
Maxon Motors with gears and encoders	2	$150.00	$300.00
Chassis	1	$10.00	$10.00
Wheels	2	$2.50	$5.00
UV tron	1	$50.00	$50.00
Dual H Bridge	1	$5.00	$5.00
LM411	3	$0.50	$1.50
SRF04 Sensors	3	$5.00	$15.00
Not Gate 7404	1	$0.50	$0.50
Discrete Components		$3.00	$3.00
Batteries	2	$40.00	$80.00
Fan	1	$10.00	$10.00
			$475.00

Figure 19: Supplied items

Parts that we ordered using our $325 budget.

Parts Ordered	Quantity	Price (each)	Total Cost (Including Shipping)
Arduino Due	1	$39.95	$39.95
Ball Caster	2	$1.99	$5.77
5v-3.3v Chip	4	$0.50	$2.98
Wii Remote	1	$13.79	$13.79
Super Bright LED	3	$2.95	$5.90
Surface Mount DIP	2	$5.00	$12.00
Green Wires	4	$5.00	$20.00
Servo(Plastic)	1	$8.95	$8.95
Servo(Metal)	1	$13.99	$13.99
Vivotech Ultrasonics	4	$5.65	$22.60
SainSmart Ultrasonics	3	$5.50	$16.50
Perf-Board	3	$0.75	$2.25
Multi OpAmp DIP	3	$2.00	$6.00
Inverter DIP	3	$3.00	$9.00
Bread Board	1	$5.00	$5.00
Siren	1	$8.25	$8.25
Male Male Wire Kit	1	$7.00	$7.00
		Total	$199.93

Figure 20: Purchased Items

Overall we have spent $199.93 of our $325.00 budget. The parts we were given or salvaged from robot jail amounted to $475.00. In total our robot would cost $674.93 to completely replicate.

Estimated Power Budget

Item	Power Each	Number Used	Power Total
Motors	6 W	2	12 W
Arduino Due	3.6 W	2	7.2 W
Line Sensor	0.3 W	3	1 W
Ultrasonic	0.03 W	7	0.21 W
Wii-mote Flame Sensor	0.15 W	1	0.15 W
UVtron	0.048 W	1	0.05 W
Fan	1.2 W	1	1.2 W
Octal Bus Transceiver	0.01 W	3	0.03 W
Ripple Counter	0.01 W	3	0.03 W
Misc			0.25
		Total	22.12 W

Figure 21: Estimated Power Budget

References

"2001 Groups and Final Reports." *2001 Groups and Final Reports*. NMT Electrical Engineering Department, May-June 2001. Web. Feb.-Mar. 2014.

"Fire Fighting Challenge 2014." *RoboRAVE International*. N.p., n.d. Web. Feb.-Mar. 2014.

"Wii IR Camera - RoboTeen." *RoboTeen*. N.p., n.d. Web. Feb.-Mar. 2014. <http://roboteen.org/wordpress/wii-ir-camera/>

"Wii Remote IR Camera Hack with Arduino Interface." *Instructables.com*. N.p., Jan.-Feb. 2010. Web. Feb.-Mar. 2014.

Spectrex, Inc. "Flame Detector Types." N.p., May 2008. Web. Jan.-Feb. 2014. <http://spectrex-inc.com/files/sharpeye/presentations/firedetectiontypes_may08.pdf>.

"How to Build a Robot Tutorials - Society of Robots." *How to Build a Robot Tutorials - Society of Robots*. N.p., 2005. Web. Jan.-Feb. 2014.

"Arduino - Libraries." *Arduino - Libraries*. Arduino, n.d. Web. Mar.-Apr. 2014.

Texas Instruments. "A Single-Supply Op-Amp Circuit Collection." N.p., Nov. 2000. Web. Jan.-Feb. 2014. <http://www.eng.yale.edu/ee-labs/morse/compo/sloa058.pdf>

"Hp HEDS - 5000 Series Datasheet." N.p., n.d. Web. Feb.-Mar. 2014. <http://www.telusplanet.net/public/mcpheej/HP_HEDS_5000_Series/HEDS_5000_Optical_Encoders.pdf>.

"SRF04 Technical Documentation." *SRF04 Technical Documentation*. N.p., May 2005. Web. Feb.-Mar. 2014. <http://www.robot-electronics.co.uk/htm/srf04tech.htm>.

"OCTAL BUS TRANSCEIVER WITH 3-STATE OUTPUTS." N.p., Sept. 2010. Web. Feb.-Mar. 2014. <http://www.adafruit.com/datasheets/sn74lvc245a.pdf>.

"Maxon DC Motor." N.p., Apr. 2000. Web. Feb.-Mar. 2014. <http://www.ee.nmt.edu/~wedeward/EE382/SP01/2322_18V_motor.pdf>

"Dual H -Bridge DC Motor Driver." N.p., Aug. 1998. Web. Feb.-Mar. 2014. <http://www.cs.unca.edu/~bruce/Spring07/180/dhb-v2.pdf>.

"DE0-Nano Development and Education Board." *Altera News*. N.p., Dec. 2005. Web. 05 May 2014. <http://www.altera.com/education/univ/materials/boards/de0-nano/unv-de0-nano-board.html>.

Appendix A

Master Arduino Due Code

```
/* TEAM SAFIRE Encoder Arduino Due Master
 *
 * By Nathaniel Miller
 * Last edited 5 - 5 - 2014
 *
 */

//*********************************************************************************************
//*********************************************************************************************
//include needed files
#include <Average.h>
#include <Wire.h>
#include <PID_v1.h>
#include <DistanceSRF04.h>
#include <Servo.h>

//*********************************************************************************************
//*********************************************************************************************
//pin definitions
#define Ap 2                    //set pins for motor direction control
#define An 3                    //A -> Right Motor
#define Bp  4                   //B -> Left Motor
#define Bn  5
#define Aenable  6              //set pins for motor speed control
#define Benable  7
#define button_start_pin 8      //button press for start
#define echo90 22               //echo and trigger pins for ultrasonics
#define trig90 23
#define echo60 24
#define trig60 25
#define echo30 26
#define trig30 27
#define echo0 28
#define trig0 29
#define echo_30 42
#define trig_30 43
#define echo_60 44
#define trig_60 45
#define echo_90 46
#define trig_90 47

//constant and factor definitions
#define qnum 16                 //factor reduced by ripple counter (2^Q)
#define forward 4095.0          //quarter speed of max of 4096
#define correction_factor 50.0  //corrects for off course
#define wheel_diameter 9.6063   //distance between wheels
#define danger_threshold_fwd 15.0   //dangerous distances
#define danger_threshold_side 10.0
#define minSide 10.0            //minimum distances
#define minForward 15.0
#define min_move_past 10        //minimum move distance
#define max_angle_off  0.2      //max angle off
#define size_y  100             //100/96 factor
#define mindist 200             //minimum ultrasonic distance in centimeters
#define iterations 3            //number of iterations for wii camera
#define max_time_between_fire 100  //maximum time to elapse between fire detects
#define line_trip 250                   //white line trip threshold

//*********************************************************************************************
```

```
//*********************************************************************************
//Function Prototypes - see functions for descriptions
void getDistance(void);
void updatePID(void);
void saveOldData(void);
void calculateLocation(void);
void toPrint(void);
void ultrasonicDistances(void);
void closestObject();
void turnTo(void);
void robotSearch(void);
void setFactor(void);
void wiiCam(void);
void servoSearch(void);
void servoBlow(void);
void trackFire(void);
void lineCheck(void);

//*********************************************************************************
//*********************************************************************************
//distance variables
byte from_slave[8] = {0};              //data recieved from slave

long newLeft = 0;                      //variables for distance
long oldLeft = 0;
long newRight = 0;
long oldRight = 0;

double factor = .0001699389 * qnum;     //inches per state change
double dist_trav_left = 0.0;            //distance travelled by wheel
double dist_trav_right = 0.0;
double dist_trav_left_old = 0.0;        //old distance travelled by wheel
double dist_trav_right_old = 0.0;
double delta_trav_left = 0.0;           //distance travelled since last update
double delta_trav_right = 0.0;
double difference_trav = 0.0;           //difference between delta trav
double delta_trav_left_old = 0.0;       //old delta trav
double delta_trav_right_old = 0.0;

//*********************************************************************************
//wii camera variables
byte data_buf[16];                     //buffer variable
int i;                                 //counter
int Ix[4];                             //actual values
int Iy[4];
int s;
int data_x_ave;                        //averaged values
int data_y_ave;
int data_x_under[iterations];          //data to be averaged
int data_y_under[iterations];
int count;                             //counter

//*********************************************************************************
//servo variables
volatile int servopos = 0;             //servo position
int toadd = 1;                         //positions to increment
Servo myservo;                         / create servo object to control a servo
#define num_servo_pos 5                //size of servo position array
int servo_array[num_servo_pos] = {10, 45, 90, 135, 170};          //servo position

//*********************************************************************************
//PID and velocity variables
double Setpoint_Left = 0.0;            //variables for PID
double Setpoint_Right = 0.0;
double Velocity_Left = 0.0;
double Velocity_Right = 0.0;
```

```
double Output_Left = 0.0;
double Output_Right = 0.0;

//Define the aggressive and conservative Tuning Parameters
double aggKp = 100.0;                  //variables for PID
double aggKi = 20.0;
double aggKd = 2.0;
double consKp = 25.0;
double consKi = 2.0;
double consKd = 0.2;

//Specify the links and initial tuning parameters
PID PID_Left(&Velocity_Left, &Output_Left, &Setpoint_Left, consKp, consKi, consKd, DIRECT);
PID PID_Right(&Velocity_Right, &Output_Right, &Setpoint_Right, consKp, consKi, consKd, DIRECT);

int right_freq = forward;              //initialize speed to full speed
int left_freq = forward;

long start_time_left = 100.0;           //variables for velocity
long start_time_right = 100.0;

double velocity_left = 15.0;            //velocity calculations
double velocity_right = 15.0;
double delta_v_left = 0.0, delta_v_right = 0.0;    //change in velocity

//*********************************************************************************************
//position variables
boolean turn_in_progress = false;         //flag for turning

int finish_turn = 0;                   //status of turn

double xCoord = 0.0;                    //x and y coordinates for current location
double yCoord = 0.0;
double deltax = 0.0;                    //change in x/y
double deltay = 0.0;
double theta = 0.0;                    //angle relative to start
double desired_theta = 0.0;
double theta_new = 0.0;                  //new angle
double radius_prime = 0.0;               //radius of curve
double magnitude = 0.0;                  //distance travelled
double temp = 0.0;

//*********************************************************************************************
//ultrasonic variables

DistanceSRF04 Dist0;                     //initialize modules for each sensor
DistanceSRF04 Dist30;
DistanceSRF04 Dist60;
DistanceSRF04 Dist90;
DistanceSRF04 Dist_30;
DistanceSRF04 Dist_60;
DistanceSRF04 Dist_90;

double  distance0 = 80;                  //distance variables
double  distance30 = 80;                 //note all distances scaled by 1000 (1000 units = 1 inch)
double  distance60 = 80;
double  distance90 = 80;
double  distance_30 = 80;
double  distance_60 = 80;
double  distance_90 = 80;

double closestLeft = 0;                  //closest object
double closestRight = 0;
double closestForward = 0;

//*********************************************************************************************
```

```
//motor direction

double right_direction = 0;                //sets direction of motors
double left_direction = 0;

//*********************************************************************************
//motor speed variables
boolean set_Ap = false;                   //H-Bridge enable pins
boolean set_An = false;
boolean set_Bp = false;
boolean set_Bn = false;

//*********************************************************************************
//robotSearch
int allow_y = 1;                          //allow y movement
int scan_up = 1;                          //scan direction
int negate = 1;                          //case for negation
int flip = 0;                           //for flipped direction
int check_other = 0;                      //check opposite side
int x_pos_clear = 0;                      //directions clear
int x_neg_clear = 0;
int y_pos_clear = 0;
int y_neg_clear = 0;

//*********************************************************************************
//uv stuff
int turn_direction = 1;

volatile int i_see_fire = 0;
volatile int run_once = 0;
volatile int fire_servo_pos = -1;

double time_since_fire = 0.0;
double last_fire_time = 0.0;

//*********************************************************************************

volatile int last_wii = 0;

const int sensorPin = A0;    // pin that the sensor is attached to
const int sensorPinLeft = A1;    // pin that the sensor is attached to
const int sensorPinRight = A2;    // pin that the sensor is attached to

// variables for line sensors
int sensorValue = 0;        // the sensor value
int sensorMin = 500;        // minimum sensor value
int sensorMax = 0;          // maximum sensor value

int sensorValueLeft = 0;        // the sensor value
int sensorMinLeft = 500;        // minimum sensor value
int sensorMaxLeft = 0;          // maximum sensor value

int sensorValueRight = 0;        // the sensor value
int sensorMinRight = 500;        // minimum sensor value
int sensorMaxRight = 0;          // maximum sensor value

int line_stop = 0;                      //stop due to line sensor

//*********************************************************************************
//*********************************************************************************
void setup()
{
 Serial.begin(115200);                  //setup computer serial monitor
 Serial.println("Start Program");

 while (millis() < 5000) {                //calibrate sensors
```

```
  sensorValue = analogRead(sensorPin);  //read initial value
  sensorValueLeft = analogRead(sensorPinLeft);
  sensorValueRight = analogRead(sensorPinRight);


  // record the maximum sensor value
  if (sensorValue > sensorMax) {
    sensorMax = sensorValue;
  }
     if (sensorValueLeft > sensorMaxLeft) {
    sensorMaxLeft = sensorValueLeft;
  }
     if (sensorValueRight > sensorMaxRight) {
    sensorMaxRight = sensorValueRight;
  }

  // record the minimum sensor value
  if (sensorValue < sensorMin) {
    sensorMin = sensorValue;
  }
     if (sensorValueLeft < sensorMinLeft) {
    sensorMinLeft = sensorValueLeft;
  }
     if (sensorValueRight < sensorMinRight) {
    sensorMinRight = sensorValueRight;
  }
 }

 pinMode(Ap, OUTPUT);                    //set motor control pins for output
 pinMode(An, OUTPUT);
 pinMode(Bp, OUTPUT);
 pinMode(Bn, OUTPUT);
 pinMode(Aenable, OUTPUT);
 pinMode(Benable, OUTPUT);
 pinMode(button_start_pin, INPUT);          //set button pin to input

 analogWriteResolution(12);                //set high pwm resolution max = 4095

slaveAddress = IRsensorAddress >> 1;   // Address of camera

 Velocity_Left = 0.0;                    //initialize PID variables
 Velocity_Right = 0.0;
 Setpoint_Left = 40.0;
 Setpoint_Right = 40.0;

 set_Ap = false;                         //set H-Bridge to stopped position
 set_An = false;
 set_Bp = false;
 set_Bn = false;
 digitalWrite(Ap, set_Ap);                //write to H-Bridge pins
 digitalWrite(An, set_An);
 digitalWrite(Bp, set_Bp);
 digitalWrite(Bn, set_Bn);
 analogWrite(Aenable, forward - 2000);      //set starting motor speed to 1/2 max
 analogWrite(Benable, forward - 2000);

 PID_Left.SetMode(AUTOMATIC);              //turns on PID
 PID_Right.SetMode(AUTOMATIC);
 PID_Left.SetOutputLimits(0, forward - 1.0);   //set max/min changes
 PID_Right.SetOutputLimits(0, forward - 1.0);
 PID_Left.SetSampleTime(20);               //evaluate PID more often
 PID_Right.SetSampleTime(20);

 Dist90.begin(echo90, trig90);             //start ultrasonics
 Dist60.begin(echo60, trig60);
 Dist30.begin(echo30, trig30);
```

```
 Dist0.begin(echo0, trig0);
 Dist_30.begin(echo_30, trig_30);
 Dist_60.begin(echo_60, trig_60);
 Dist_90.begin(echo_90, trig_90);

 delay(500);                          //wait for other due and sensors to setup

 Wire.begin();                        //setup as master for I2C

Write_2bytes(0x30, 0x01); delay(10); //Control byte, allows modification of settings
 Write_2bytes(0x06, 0xc8); delay(10); // 10 MAXSIZE - Maximum blob size. Wii uses values from 0x62 to 0xc8.
 Write_2bytes(0x08, 0xc0); delay(10); // 15 GAIN - Sensor Gain. Smaller values = higher gain. Numerical gain
 Write_2bytes(0x1A, 0x40); delay(10); // 10 GAINLIMIT - Sensor Gain Limit. Must be less than GAIN for camera to function
 Write_2bytes(0x1B,0x03); delay(10); // * MINSIZE - Minimum blob size. Wii uses values from 3 to 5
 Write_2bytes(0x33, 0x33); delay(10); //
 Write_2bytes(0x30, 0x08); delay(10); //Was Out of order, needs to be at end
 delay(100);

 while (digitalRead(button_start_pin) == LOW);  //wait for button press

 set_Ap = true;                       //start moving forward
 set_Bp = true;
 digitalWrite(Ap, set_Ap);                //turn on motors
 digitalWrite(Bp, set_Bp);
}

//*****************************************************************************
//*****************************************************************************
void loop()
{
 setFactor();
 getDistance();
 updatePID();
 calculateLocation();
 ultrasonicDistances();
 closestObject();
 toPrint();
 saveOldData();
 wiiCam();
 robotSearch();
 turnTo();
 servoSearch();
 trackFire();
}

/*****************************************************************************
*
*calculateLocation function
*input none
*output none
*uses global variables
*determine x, y, and theta location of robot
*
*/
void calculateLocation(void)
{
 if (delta_trav_left != 0 || delta_trav_right != 0)         //check for new data
 {
  difference_trav = delta_trav_left - delta_trav_right;     //set difference travelled
  difference_trav = abs(difference_trav);                   //take absolute of distance travelled

  theta_new = (difference_trav / wheel_diameter);           //determine new theta

  if ((delta_trav_left > delta_trav_right) && theta_new != 0)  //check for right turn
  {
   radius_prime = (delta_trav_right / theta_new) + (0.5 * wheel_diameter); //radius of turn
```

```
    temp = cos(theta_new);
    deltay = radius_prime * (1.0 - temp);                //change y for relative axis
    temp = sin(theta_new);
    deltax = radius_prime * temp;                        //change x for relative axis

    if (deltax == 0)                                     //check for change in x
      temp = 0;                                         //set angle addition to 0
    else
      temp = atan(deltay / deltax);                      //calculate angle change

    if (deltay < 0 && deltax < 0 && temp != 0)           //determine angle quadrant
      temp = temp - 3.1416;
    else if (deltax < 0 && deltay > 0 && temp != 0)
      temp = temp + 3.1416;

    theta = theta + temp;                                //updated angle
    magnitude = sqrt(deltax * deltax + deltay * deltax);    //magnitude of distance travelled
    temp = sin(theta);
    yCoord = yCoord - (magnitude * temp);                //change in y
    temp = cos(theta);
    xCoord = xCoord + (magnitude * temp);                //change in x
   }

   else if (theta_new != 0)                              //case for left turn
   {
    radius_prime = (delta_trav_left / theta_new) + (0.5 * wheel_diameter);  //radius of turn
    temp = cos(theta_new);
    deltay = radius_prime * (1.0 - temp);                //change y for relative axis
    temp = sin(theta_new);
    deltax = radius_prime * temp;                        //change x for relative axis

    if (deltax == 0)                                     //check for change in x
      temp = 0;                                         //set angle addition to 0
    else
      temp = atan(deltay / deltax);                      //calculate angle change

    if (deltay < 0 && deltax < 0 && temp != 0)           //determine angle quadrant
      temp = temp - 3.1416;
    else if (deltax < 0 && deltay > 0 && temp != 0)
      temp = temp + 3.1416;

    theta = theta - temp;                                //updated angle
    magnitude = sqrt(deltax * deltax + deltay * deltay);    //magnitude of distance travelled
    temp = sin(theta);
    yCoord = yCoord - (magnitude * temp);                //change in y
    temp = cos(theta);
    xCoord = xCoord + (magnitude * temp);                //change in x
   }
  }
}

//*****************************************************************************
/*****************************************************************************
*
*closestObject function
*input none
*output none
*uses global variables
*sets closest objects based on ultrasonics
*
*/

void closestObject(void)
{
  int  temp = 0;
```

```
  //find closes forward object on front three ultrasonics
  closestForward = distance0;              //guess front ultrasonic sees closest object

  temp = distance30 * 0.866;               //set temporary value to check
  if (temp < closestForward)               //case for right angled ultrasonic closer
  {
   closestForward = temp;                  //set new value as closest
  }

  temp = distance_30 * 0.866;              //set temporary value to check
  if (temp < closestForward)               //case for left angled ultrasonic closer
  {
   closestForward = temp;                  //set new value as closest
  }

  //find closes left side object on side two ultrasonics, comments similar and ommited
  closestLeft = distance_90;

  temp = distance_60 * 0.866;
  if (temp < closestLeft)
  {
   closestLeft = temp;
  }

  //find closes right side object on side two ultrasonics
  closestRight = distance90;

  temp = distance60 * 0.866;
  if (temp < closestRight)
  {
   closestRight = temp;
  }
}
//*****************************************************************************
/*****************************************************************************
*
*getDistance function
*input none
*output none
*uses global variables
*updates rotation counter by taking data from slave Due via I2C
*also updates distances
*
*/
void getDistance(void)
{
  int counter = 0;                  //variable for for loops
  double temp = 0.0;                //temporary variable

  Wire.requestFrom(2, 8);           //request data from slave Due

  while (Wire.available())          //able to transmit
  {
   from_slave[counter] = Wire.read();  //save incoming array to variable
   counter++;                       //increment counter
  }

  Wire.endTransmission();           //finish recieving

  union newLeft_tag {               //union to convert array to long
   byte newLeft_b[4];               //array of 4 bytes
   long newLeft_fval;               //long int is final value
  } newLeft_U;                      //name of union

  union newRight_tag {              //union for right side
   byte newRight_b[4];
```

```
  long newRight_fval;
 } newRight_U;

 for (counter = 0; counter < 4; counter++)    //walk through data
 {
  newLeft_U.newLeft_b[counter] = from_slave[counter];  //populate union with array elements
 }
 newLeft = newLeft_U.newLeft_fval;    //creating long from transmitted data

 for (counter = 0; counter < 4; counter++)    //walk through remaining data
 {
  newRight_U.newRight_b[counter] = from_slave[counter + 4]; //populate union with array elements
 }
 newRight = newRight_U.newRight_fval; //creating long from transmitted data

 temp = newLeft - oldLeft;            //determine delta from counters
 temp = abs(temp);                    //absolute of counters
 temp = temp * factor * left_direction;  //change based on direction information and offset factor
 dist_trav_left += temp;              //update distance travelled

 temp = newRight - oldRight;          //same as above for right side
 temp = abs(temp);
 temp = temp * factor * right_direction;
 dist_trav_right += temp;

 delta_trav_left = (dist_trav_left - dist_trav_left_old);    //set delta travelled
 delta_trav_right = (dist_trav_right - dist_trav_right_old);
 delta_trav_left = abs(delta_trav_left);                     //absolute of distance travelled
 delta_trav_right = abs(delta_trav_right);
}
//***************************************************************************
/***************************************************************************
*
*robotSearch function
*input none
*output none
*uses global variables
*moves around field in center search algorithm mode
*
*/
void robotSearch(void)
{

 //below center line
 if (((yCoord + wheel_diameter * .5) < size_y / 2) && (y_pos_clear == 0) && allow_y == 1)
 {
  if ((theta > (1.57 + max_angle_off)) || (theta < (1.57 - max_angle_off)))  //correct if not moving in desired direction
  {
   desired_theta = negate * 1.57;        //sets desired direction
  }
 }

 //above center line
 else if (((yCoord - wheel_diameter * .5) > size_y / 2) && (y_neg_clear == 0) && allow_y == 1)
 {
  if ((theta > (-1.57 + max_angle_off)) || (theta < (-1.57 - max_angle_off)))  //correct if not moving in desired direction
  {
   desired_theta = negate * -1.57;       //sets desired direction
  }
 }

 //when at center line
 else if (scan_up == 1)                  //scan up field
 {

  //x direction is clear
```

```
 if (x_pos_clear == 0)
 {
  if ((theta > (max_angle_off)) || (theta < (max_angle_off)))    //correct if not moving in desired direction
  {
   desired_theta = 0 + ((negate + 1) / 2) * 3.14;   //sets desired direction
  }

  else                              //when x direction not clear, move in y direction
  {
   allow_y -= 1;                    //sets y direction flag
  }

  if (allow_y < min_move_past)         //when moved sufficiently in y direction past barrier
   allow_y = 1;                     //reset y flag
 }

 //x direction not clear
 else if ((y_pos_clear == 0) && (check_other == 0))
 {
  //move in positive y direction until clear
  if ((theta > (1.57 + max_angle_off)) || (theta < (1.57 - max_angle_off))) //correct if not moving in desired direction
  {
   desired_theta = 1.57;              //sets desired direction
  }
  allow_y = 0;                      //do not allow y movement
 }

 //x not clear and positive y not clear
 else if ((x_pos_clear == 1) && (y_pos_clear == 1))
 {
  check_other = 1;                   //set opposite y direction flag
  allow_y = 0;                      //reset movement in y to not allowed
 }

 //move in negative y direction
 if (check_other == 1)
 {

  //if y negative clear
  if ((x_pos_clear == 1) && (y_neg_clear == 0))
  {
   if ((theta > (-1.57 + max_angle_off)) || (theta < (-1.57 - max_angle_off)))  //correct if not moving in desired direction
   {
    desired_theta = -1.57;           //sets desired direction
   }
  }

  //cannot move in positive x direction whatsoever
  else if ((x_pos_clear == 1) && (y_neg_clear == 1) && (flip == 0))
  {
   check_other = 0;                  //reset flag to check both y directions
   allow_y = 1;                     //reallow y movement
   flip = 1;                        //flip directions
   negate = negate * -1;             //move in opposite direction
  }

  //cannot move in positive x direction whatsoever
  else if ((x_pos_clear == 1) && (y_neg_clear == 1) && (flip == 1))
  {
   check_other = 0;                  //reset flag to check both y directions
   allow_y = 1;                     //reallow y movement
   flip = 0;                        //flip directions
   negate = negate * -1;             //move in opposite direction
  }
 }
}
```

```
}
//*****************************************************************************
/*****************************************************************************
*
*saveOldData function
*input none
*output none
*uses global variables
*saves distance and count data to variables for later reference to determine deltas
*
*/
void saveOldData(void)
{
 dist_trav_left_old = dist_trav_left;          //save distance travelled
 dist_trav_right_old = dist_trav_right;

 oldLeft = newLeft;                          //save old data encoder data
 oldRight = newRight;

 delta_trav_left_old = delta_trav_left;        //save old deltas
 delta_trav_right_old = delta_trav_right;
}
//*****************************************************************************
/*****************************************************************************
*
*setFactor function
*input none
*output none
*uses global variables
*sets directional factor
*
*/
void setFactor(void)
{
 right_direction = ((-1 * set_An) + set_Ap);      //set direction of right wheel movememnt based upon enable pins
 left_direction = ((-1 * set_Bn) + set_Bp);       //set direction of left wheel movememnt based upon enable pins
}
//*****************************************************************************
/*****************************************************************************
*
*toPrint function
*input none
*output none
*uses global variables
*prints desired variables
*format for printing: Serial.print(" your string here ");
*                    Serial.println(your_variable_here);
*comment or uncomment desired information to print to screen
*
*/
void toPrint(void)
{

 //print statements in this loop only print when new data from the encoders is recieved
 if (newLeft != oldLeft || newRight != oldRight)
 {
  Serial.print(newLeft);
  Serial.print(" - ");
  Serial.print(newRight);
  Serial.print(" - ");
  Serial.print(dist_trav_left, 7);
  Serial.print(" - ");
  Serial.print(dist_trav_right, 7);
  Serial.print(" - ");
  Serial.print(xCoord, 7);
  Serial.print(" - ");
```

```
  Serial.print(yCoord, 7);
  Serial.print(" - ");
  Serial.println(theta, 7);

  Serial.print(Velocity_Left, 7);
  Serial.print("  -  ");
  Serial.print(Velocity_Right, 7);
  Serial.print("  -  ");
  Serial.print(Output_Left, 7);
  Serial.print("  -  ");
  Serial.print(Output_Right, 7);
  Serial.print("  -  ");
  Serial.print(delta_trav_left, 7);
  Serial.print("  -  ");
  Serial.println(delta_trav_right, 7);

 }

 Serial.print(distance_90);
 Serial.print(" - ");
 Serial.print(distance_60);
 Serial.print(" - ");
 Serial.print(distance_30);
 Serial.print(" - ");
 Serial.print(distance0);
 Serial.print(" - ");
 Serial.print(distance30);
 Serial.print(" - ");
 Serial.print(distance60);
 Serial.print(" - ");
 Serial.println(distance90);

 Serial.print(closestLeft);
 Serial.print(" - ");
 Serial.print(closestRight);
 Serial.print(" - ");
 Serial.println(closestForward);


}
//*****************************************************************************
/*****************************************************************************
*
*turnTo function
*input none
*output none
*turn based on set desired angle
*
*/
void turnTo(void)
{
 //turn left
 if ((theta > (desired_theta)) && (turn_in_progress == true) && (finish_turn <= 0))
 {
  set_Ap = true;              //set enables for left turn
  set_An = false;
  set_Bp = false;
  set_Bn = false;
  digitalWrite(Ap, set_Ap);
  digitalWrite(An, set_An);
  digitalWrite(Bp, set_Bp);
  digitalWrite(Bn, set_Bn);

  Setpoint_Left = 40;          //set speed for turn
  Setpoint_Right = 40;
  finish_turn = -2;            //set last turn indicator
```

```
  }

  //turn right
  else if ((theta < desired_theta) && (turn_in_progress == true) && (finish_turn >= 0))
  {
   set_Ap = false;              //set enables for right turn
   set_An = false;
   set_Bp = true;
   set_Bn = false;
   digitalWrite(Ap, set_Ap);
   digitalWrite(An, set_An);
   digitalWrite(Bp, set_Bp);
   digitalWrite(Bn, set_Bn);

   Setpoint_Left = 40;          //set speed for turn
   Setpoint_Right = 40;
   finish_turn = 2;             //set last turn indicator
  }

  //turn just completed
  else if (turn_in_progress == true)
  {
   set_Ap = true;                //reset to forward direction
   set_An = false;
   set_Bp = true;
   set_Bn = false;
   digitalWrite(Ap, set_Ap);
   digitalWrite(An, set_An);
   digitalWrite(Bp, set_Bp);
   digitalWrite(Bn, set_Bn);

   turn_in_progress = false;       //reset turn flag
   finish_turn = 0;               //reset last turn indicator
   Setpoint_Left = 40;             //set speed
   Setpoint_Right = 40;
  }

  //ensure forward direction
  else
  {
   set_Ap = true;                //reset to forward direction
   set_An = false;
   set_Bp = true;
   set_Bn = false;
   digitalWrite(Ap, set_Ap);
   digitalWrite(An, set_An);
   digitalWrite(Bp, set_Bp);
   digitalWrite(Bn, set_Bn);

   Setpoint_Left = 40;             //set speed
   Setpoint_Right = 40;
  }
}

//*****************************************************************************
/*
* ultrasonicDistances function
* inputs none
* outputs none
* uses global variables
* finds and returns distance information
* uses staggered method to ensure reflections do not affect other sensors
*
*           0
*          _|_|_
*    -30 =/     \= 30
```

```
*   -60 =/       \= 60
*  -90 =|        |= 90
*
*/
void ultrasonicDistances(void)
{
 int temp = 0;

 //front sensor
 temp = Dist0.getDistanceCentimeter();       //get distance from sensor in centimeters
 if (temp > 0 && temp < mindist)             //when non-zero and less than 200 centimeters
   distance0 = .4 * temp;                    //convert and save in inches
 else
   distance0 = 80;                           //assume greater than 80 inches

 //left 90
 temp = Dist_90.getDistanceCentimeter();
 if (temp > 0 && temp < mindist)
   distance_90 = .4 * temp;
 else
   distance_90 = 80;

 //right 30
 temp = Dist30.getDistanceCentimeter();
 if (temp > 0 && temp < mindist)
   distance30 = .4 * temp;
 else
   distance30 = 80;

 //left 60
 temp = Dist_60.getDistanceCentimeter();
 if (temp > 0 && temp < mindist)
   distance_60 = .4 * temp;
 else
   distance_60 = 80;

 //right 60
 temp = Dist60.getDistanceCentimeter();
 if (temp > 0 && temp < mindist)
   distance60 = .4 * temp;
 else
   distance60 = 80;

 //left 30
 temp = Dist_30.getDistanceCentimeter();
 if (temp > 0 && temp < mindist)
   distance_30 = .4 * temp;
 else
   distance_30 = 80;

 //right 90
 temp = Dist90.getDistanceCentimeter();
 if (temp > 0 && temp < mindist)
   distance90 = .4 * temp;
 else
   distance90 = 80;
}
//*****************************************************************************
/*****************************************************************************
*
*updatePID function
*input none
*output none
*uses global variables
*updates PID values and correct motor speeds
*also updates velocity, delta d travelled
```

```
*
*/
void updatePID(void)
{
 Velocity_Left = abs(1000.0 * (delta_trav_left) / ((double)(millis() - start_time_left)));     //determine velocity of left wheel
 start_time_left = millis();     //reset time counter

 Velocity_Right = abs(1000.0 * (delta_trav_right) / ((double)(millis() - start_time_right)));  //determine velocity of right wheel
 start_time_right = millis();    //reset time counter


 delta_v_left = Setpoint_Left - Velocity_Left;     //determine delta velocities
 delta_v_right = Setpoint_Right - Velocity_Right;

 double gap_Left = abs(delta_v_left);      //determine offset from desired velocity
 double gap_Right = abs(delta_v_right);


 if (gap_Left < .0001)                    //close case
 {
  PID_Left.SetTunings(consKp, consKi, consKd);    //run PID correction for left wheel
 }
 else                                     //far off case
 {
  PID_Left.SetTunings(aggKp, aggKi, aggKd);       //run PID correction for left wheel
 }

 if (gap_Right < .0001)                   //close case
 {
  PID_Right.SetTunings(consKp, consKi, consKd);   //run PID correction for right wheel
 }
 else                                     //far off case
 {
  PID_Right.SetTunings(aggKp, aggKi, aggKd);      //run PID correction for right wheel
 }

 PID_Left.Compute();                      //calculate PID equation for new output
 PID_Right.Compute();

 analogWrite(Benable, Output_Left);       //sets speed sent to H-Bridge
 analogWrite(Aenable, Output_Right);
}
//*****************************************************************************
/*****************************************************************************
*
*wiiCam function
*input none
*output none
*uses global variables
*recieve wii data input
*
*/
void wiiCam(void)
{

 Wire.beginTransmission(slaveAddress);     //IR sensor read
 Wire.write(0x36);
 Wire.endTransmission();

 Wire.requestFrom(slaveAddress, 16);       // Request the 2 byte heading (MSB comes first)

 for (i = 0; i < 16; i++)                  // initialize buffer
 {
  data_buf[i] = 0;
 }
 i = 0;
```

```
 while (Wire.available() && i < 16) {       //Read data from wii cam
  data_buf[i] = Wire.read();
  i++;
 }

 Ix[0] = data_buf[1];                      //sort x data
 s   = data_buf[3];
 Ix[0] += (s & 0x30) << 4;

 if (count < iterations - 1)  {             //get normal x data to be averaged
  if (Ix[0] < 1023) {
   data_x_under[data_x_under[iterations - 1]] = Ix[0];  //save to variable to be averaged
   data_x_under[iterations - 1]++;         //increment counter
  }
  count++;
 }

 else                                      //average values obtained
 {
  if (data_x_under[iterations - 1] > 0) {  //case for valid data
    for (i = 0; i < data_x_under[iterations - 1]; i++) {
      data_x_ave = data_x_ave + data_x_under[i];  //sum values
    }
    data_x_ave = data_x_ave / data_x_under[iterations - 1];   //average values
  }
  else                                     //no values below 1023 or above 0
    data_x_ave = 1023;

  Ix[0] = data_x_ave;                      //set to average data

  for (i = 0; i < 4; i++)                  //print data to screen
  {
    if (Ix[i] < 1000)
      Serial.print(" ");
    if (Ix[i] < 100)
      Serial.print(" ");
    if (Ix[i] < 10)
      Serial.print(" ");
    Serial.print( int(Ix[i]) );
  }
  Serial.println("");

  count = 0;                               //reset variables
  data_x_ave = 0;
  for (i = 0; i < iterations; i++) {
    data_x_under[i] = 0;
  }
 }
}


void Write_2bytes(byte d1, byte d2)
{
 Wire.beginTransmission(slaveAddress);
 Wire.write(d1); Wire.write(d2);
 Wire.endTransmission();
}

/*****************************************************************************
*
*trackFire function
*input none
*output none
*uses
*record servo position when fire seen
```

```
*
*/
void trackFire(void)
{
    if(fire_servo_pos >=0 || servopos == 0)      //fire seen
    {
     myservo.write(servo_array[fire_servo_pos]);    //move to fire position
     delay(4000);
     fire_servo_pos = -1;                                        //set fire flag
    }
}
/*****************************************************************************
*
*servoSearch function
*input none
*output none
*uses global variables
*pans servo and checks UV tron
*
*/
void servoSearch(void)
{
 myservo.write(servo_array[servopos]);     //pan servo
 if (servopos == 0)
   delay(1000);
 servopos = servopos + toadd;                  //increment servo position
 if (servopos > (num_servo_pos - 1))         //reset servo counter
 {
  servopos = 0;
 }
 checkUV();                                                              //run UV tron function
}
/*****************************************************************************
*
*servoBlow function
*input none
*output none
*uses global variables
*pans servo and blows fan
*
*/
void servoBlow(void)
{
 int pos;
 digitalWrite(fan_pin, HIGH);       //turn on fan
   for(pos = 0; pos <= 180; pos += 1) // goes from 0 degrees to 180 degrees
 {                                  // in steps of 1 degree
  myservo.write(pos);              // tell servo to go to position in variable 'pos'
  delay(25);                       // waits 15ms for the servo to reach the position
 }
 for(pos = 180; pos>=0; pos-=1)     // goes from 180 degrees to 0 degrees
 {
  myservo.write(pos);              // tell servo to go to position in variable 'pos'
  delay(25);                       // waits 15ms for the servo to reach the position
 }
 digitalWrite(fan_pin, LOW); //turns off fan
 myservo.write(90);  //resets fan
 delay(25);
}

/*****************************************************************************
*
*getDistance function
*input none
*output none
*uses global variables
```

```
*randomly searches for fire
*
*/
void randomSearch(void)
{
  //fire seen
  if (Ix[0] != 1023 || Ix[0] != 0)
  {
    //fire close
    if (distance0 < 10.0 || distance30 < 10.0 || distance60 < 10.0 || distance90 < 10.0 || distance_30 < 10.0 || distance_60 < 10.0 ||
distance_90 < 10.0)
    {
      set_Ap = false;                   //stop
      set_Bp = false;
      digitalWrite(Ap, set_Ap);
      digitalWrite(Bp, set_Bp);

      servoBlow();                      //blow out fire

      Setpoint_Left = 30.0;             //reset speeds
      Setpoint_Right = 30.0;

      set_Ap = true;                    //resume movement
      set_Bp = true;
      digitalWrite(Ap, set_Ap);
      digitalWrite(Bp, set_Bp);

      myservo.write(90);                //reset servo
    }

    else if (Ix[0] < 100)               //fire seen and off angle
    {
      delay_time = 200;                 //turn to correct
      turnTo();
      delay_time = 500;
      last_wii = -1;
    }

    else if (Ix[0] > 900)               //fire seen and off angle
    {
      delay_time = 100;                 //turn to correct
      turn_direction = -1;
      turnTo();
      delay_time = 200;
      last_wii = 1;
    }

    else if (last_wii == 1)             //fire lost recently
    {
      delay_time = 300;                 //turn to last known location
      turn_direction = -1;
      turnTo();
      delay_time = 500;
      last_wii = 0;
    }

    else if (last_wii == -1)            //fire lost recently
    {
      delay_time = 300;                 //turn to last known location
      turn_direction = 1;
      turnTo();
      delay_time = 500;
      last_wii = 0;
    }
  }
```

```
  else if (line_stop == 1)              //hit line and then rotate
  {
    turnTo();
    turnTo();
  }

  //object too close
  else if (distance0 < 10.0 || distance30 < 10.0 || distance60 < 10.0 || distance90 < 10.0 || distance_30 < 10.0 || distance_60 < 10.0 ||
distance_90 < 10.0)
  {
    delay_time = 500;
    turnTo();
  }

/*********************************************************************
*
*checkUV function
*input none
*output none
*uses global variables
*check line sensors
*
*/

void lineCheck(void)
{
  sensorValue = analogRead(sensorPin);      //read initial variables
  sensorValueLeft = analogRead(sensorPinLeft);
  sensorValueRight = analogRead(sensorPinRight);


  // apply the calibration to the sensor reading
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 500);
  sensorValueLeft = map(sensorValueLeft, sensorMinLeft, sensorMaxLeft, 0, 500);
  sensorValueRight = map(sensorValueRight, sensorMinRight, sensorMaxRight, 0, 500);


  // in case the sensor value is outside the range seen during calibration
  sensorValue = constrain(sensorValue, 0, 500);
  sensorValueLeft = constrain(sensorValueLeft, 0, 500);
  sensorValueRight = constrain(sensorValueRight, 0, 500);

  if ((sensorValue > line_trip) || (sensorValueLeft > line_trip) || (sensorValueRight > line_trip) )  //case for line detected
    line_stop = 1;
  else
    line_stop = 0;

}

/*********************************************************************
*
*checkUV function
*input none
*output none
*uses global variables
*check UV Tron
*
*/

void checkUV(void)
{

  if(i_see_fire == 1)   //case for fire
  {
      i_see_fire = 0;     //reset fire flag
      fire_servo_pos = fire_servo_pos - 1;  //set fire position
```

```
    if(fire_servo_pos < 0)
    fire_servo_pos =170;
    myservo.write(servo_array[fire_servo_pos]);  //move to fire angle on servo
    delay(5000);
  }
}
```

Appendix B

Slave Arduino Due Code

```
/* TEAM SAFIRE Encoder Arduino Due Slave
 *
 * By Nathaniel Miller
 * Last edited 5 - 5 - 2014
 *
 */

//*********************************************************************************************
//include needed files
#include <Encoder.h>
#include <Wire.h>

//*********************************************************************************************
//definitions
#define encoder_address 2             //encoder slave address

#define leftA 31;                     //encoder pins
#define leftB 33;
#define rightA 35;
#define rightB 37;

Encoder knobLeft(leftA, leftB);       //left encoder setup
Encoder knobRight(rightA, rightB);    //right encoder setup

volatile byte* INPUT1FloatPtr;        //pointers for array to transmit
volatile byte* INPUT2FloatPtr;

long newLeft, newRight;               //new encoder data

void setup()
{
  Wire.begin(encoder_address);        //set as slave for I2C
  Wire.onRequest(requestEvent);       //function for sending
}

void loop()
{
  newLeft = abs(knobLeft.read());          //get new position taken by interrupts
  newRight = abs(knobRight.read());
}

//function executes when data requested
void requestEvent()
{
  byte Data[8];                       //data array to send

  INPUT1FloatPtr = (byte*) &newLeft;    //pointer to location of data to transmit
  INPUT2FloatPtr = (byte*) &newRight;

  Data[0] = INPUT1FloatPtr[0];         //transfer long to byte array
  Data[1] = INPUT1FloatPtr[1];
  Data[2] = INPUT1FloatPtr[2];
  Data[3] = INPUT1FloatPtr[3];
```

```
  Data[4] = INPUT2FloatPtr[0];
  Data[5] = INPUT2FloatPtr[1];
  Data[6] = INPUT2FloatPtr[2];
  Data[7] = INPUT2FloatPtr[3];

  Wire.write(Data, 8);                //transmit requested data
}
```

Appendix C

Matlab Simulation Code

```
%% SAFIRE Search Program
%% By Nathaniel Miller
%% Last edited 3-12-14

clear all
close all

global count_i

%%Field setup
%%=========================================================================

%Field is 8' x 12'
factor = 1;
resolutionx = 96*factor; %number of inches
resolutiony = 144*factor; %number of inches

%let a zero in the grid indicate an open spot with no walls around the spot
%and free to navigate
field_height = zeros(resolutionx, resolutiony);
barrier_height = field_height;
ground = field_height;

%setup boundary walls indicated by white and black lines on the playing
%field

for i=1:resolutionx
   for j=1:resolutiony
      if(i==1 || j==1 || i==resolutionx || j==resolutiony)
         field_height(i,j) = 2;
      end
   end
end

%set barriers onto map
%barriers can be 18" to 36"
%assume 6" wide barrier

%position

xlength = 18*factor;
ylength = 6*factor;

x_1 = 40*factor;
y_1 = 40*factor;

x_2 = 75*factor;
y_2 = 30*factor;

x_3 = 40*factor;
y_3 = 100*factor;

x_4 = 30*factor;
y_4 = 120*factor;
```

```
for i=1:resolutionx
   for j=1:resolutiony
      if((x_1<=i && i<=(x_1+xlength) && y_1<=j && j<=(y_1+ylength)) || (x_2<=i && i<=(x_2+xlength) && y_2<=j &&
j<=(y_2+ylength)) || (x_3<=i && i<=(x_3+xlength) && y_3<=j && j<=(y_3+ylength)) || (x_4<=i && i<=(x_4+xlength) && y_4<=j
&& j<=(y_4+ylength)) )
         field_height(i,j) = 2;
      end
   end
end

%plot walls
figure(1)
surf(field_height.', 'FaceColor','red','EdgeColor','none');
camlight left; lighting phong
axis([0 resolutionx 0 resolutiony 0 10]);
xlabel('X AXIS');
ylabel('Y AXIS');
hold on

surf(ground.', 'FaceColor','black','EdgeColor','none');
camlight left; lighting phong

%%=====================================================================
%Robot initialize

robot(10,10) = -1;
robot(3,3) = -1;
robot(3,10) = -1;
robot(10,3) = -1;

robotX = [3 4 5 6 7 8 9 10; 3 4 5 6 7 8 9 10; 3 4 5 6 7 8 9 10; 3 4 5 6 7 8 9 10; 3 4 5 6 7 8 9 10; 3 4 5 6 7 8 9 10; 3
4 5 6 7 8 9 10];
robotY = [3 3 3 3 3 3 3 3; 4 4 4 4 4 4 4 4; 5 5 5 5 5 5 5 5; 6 6 6 6 6 6 6 6; 7 7 7 7 7 7 7 7; 8 8 8 8 8 8 8 8; 9 9 9 9 9 9 9 9; 10 10 10
10 10 10 10 10];
robotZ = [-.1 -.1 -.1 -.1 -.1 -.1 -.1 -.1; -.1 3 3 3 3 3 3 -.1; -.1 3 3 3 3 3 3 -.1; -.1 3 3 3 3 3 3 -.1; -.1 3 3 3 3 3 3 -.1; -.1 3 3 3 3 3 3 -.1; -.1 3 3 3 3 3 3 -.1;
-.1 3 3 3 3 3 3 -.1; -.1 -.1 -.1 -.1 -.1 -.1 -.1 -.1];

for i=4:9
   for j=4:9
      robot(i,j) = 3;
   end
end

robotX = robotX + ones(size(robotX));
robotY = robotY + ones(size(robotY));

stop = 0;
k=1;

surf_handle = surf(robotX, robotY, robotZ, 'FaceColor','green','EdgeColor','none');
camlight left; lighting phong

campos([25,25,30])

direction = 1;
%1 = +x
```

```
%2 = +y
%3 = -x
%4 = -y

allow_x = 1; %%stop x motion if 0 to move up past barrier
scan_up = 1; %%scan in y+ if one
negate = 1; %%
check_other = 0; %%go other direction to check
count_i=1;
flip = 0; %%change direction
n = 1;

while(stop ~= 1)

   [robotX, robotY, direction, allow_x, scan_up, negate, check_other, flip] = robot_search(robotX, robotY, direction, field_height,
resolutionx, resolutiony, allow_x, scan_up, negate, check_other, flip);

   pause(.005);

   set(surf_handle,'XData',robotX, 'YData',robotY); % 'ZData', robot);

   M(n)=getframe;
   n = n+1;

   if(n == 1000)
      stop = 1;
   end

end

numtimes=3;
fps=10;
movie2avi(M, 'robotSearch.avi', 'compression', 'none');

%%*****************************************************************************************
%%*****************************************************************************************
function [ robotX, robotY, direction, allow_x, scan_up, negate, check_other, flip ] = robot_search(robotX, robotY, direction,
field_height, resolutionx, resolutiony, allow_x, scan_up, negate, check_other, flip)
%function takes robots location from matrix robot and uses direction
%to plot new point based upon obstacle avoidance

global count_i

%based on field resolution
scaler_num = 1;

count_i = count_i + 1;

%check clear directions
[x_pos, x_neg, y_pos, y_neg] = is_clear(robotX, robotY, field_height);

%case of x clear, y clear, x motion allowed, and robot past centerline
if(robotX(1,1)+length(robotX)/2 > resolutionx/2 && x_neg == 0 && allow_x == 1)
   robotX = robotX - scaler_num*ones(size(robotX));

   %case of x clear, x motion allowed, robot less than center line
elseif(robotX(1,1)+length(robotX)/2 < resolutionx/2 && x_pos == 0 && allow_x == 1)
```

```
    robotX = robotX + scaler_num*ones(size(robotX));

    %x movement not allowed or centered but moving up y+
elseif(scan_up == 1)

    if(flip == 1)
        [x_neg, x_pos, y_neg, y_pos] = is_clear(robotX, robotY, field_height);
    end

    %not to max y and y clear
    if(y_pos == 0) %(robotY(1,1)+length(robotY) < resolutiony) && (y_pos == 0))
        robotY = robotY + negate * scaler_num*ones(size(robotY));
        allow_x = allow_x - 1;

        %will reallow x after moving past barrier
        if(allow_x < -4)
            allow_x = 1;
        end

        %y not free - move in x around barrier
    elseif(x_pos == 0 && check_other == 0)% && robotX(1,1)+length(robotX)<resolutionx)
        robotX = robotX + negate * scaler_num*ones(size(robotX));
        allow_x = 0;

        %set negate for when x and y are blocked
    elseif(y_pos == 1 && x_pos == 1)
        fprintf('yes ');
        check_other = 1;
        allow_x = 0;
    end

    if(check_other == 1)
        if(y_pos == 1 && x_neg == 0)
            robotX = robotX - negate *scaler_num*ones(size(robotX));

        elseif(y_pos==1 && x_neg == 1 && flip ==0)
            %scan_up = 0;
            check_other = 0;
            allow_x = 1;
            flip = 1;
            negate = negate * -1;

        elseif(y_pos == 1 && x_neg == 1 && flip == 1)
            check_other = 0;
            allow_x = 1;
            flip = 0;
            negate = negate *-1;
        end

    end

end

%%*************************************************************************************************
%%*************************************************************************************************
function [ x_pos, x_neg, y_pos, y_neg ] = is_clear( robotX, robotY, field_height )
%returns directions clear
```

```
%one indicates not clear

isclear = 0;
for i=0:3
   for j=0:length(robotY)-1
      isclear = isclear + field_height(robotX(1,1) + length(robotX) + i, robotY(1,1) + j);
   end
end

if(isclear == 0)
   x_pos = 0;
else
   x_pos = 1;
end

%%-------------------------
isclear = 0;
for i=0:3
   for j=0:length(robotX)-1
      isclear = isclear + field_height(robotX(1,1) + j, robotY(1,1) + length(robotY) + i);
   end
end

if(isclear == 0)
   y_pos = 0;
else
   y_pos = 1;
end

%%-------------------------
isclear = 0;
for i=0:3
   for j=0:length(robotY)-1
      isclear = isclear + field_height(robotX(1,1) - i, robotY(1,1) + j);
   end
end

if(isclear == 0)
   x_neg = 0;
else
   x_neg = 1;
end

%%-----------------------
isclear = 0;
for i=0:3
   for j=0:length(robotY)-1
      isclear = isclear + field_height(robotX(1,1) + j, robotY(1,1) - i);
   end
end

if(isclear == 0)
   y_neg = 0;
else
   y_neg = 1;
end
end
```