Junkyard Battle Racers

Submitted by:

Group 23

Alex Zenk

Katie Johnson

Jacob Gover

Floyd Bundrant

Mech 307

Mechatronics

Colorado State University

May 6, 2016

Table of Contents

Table of Figures

Design Summary

Junkyard Battle Racers was designed to simulate the video game Mario Kart. The basis of the project was to have two drivers in separate carts be able to interact with one another while the carts were active. In order to create two go-carts that were able to communicate with each other, research was done into the best possible way to easily interact in short distances. The solution chose was infrared (IR) communication. Each cart is enabled with a transmitter and receiver that allows for one cart to contact and affect the other. To create the mystery boxes in Mario Kart, a proximity switch sensor was interfaced.

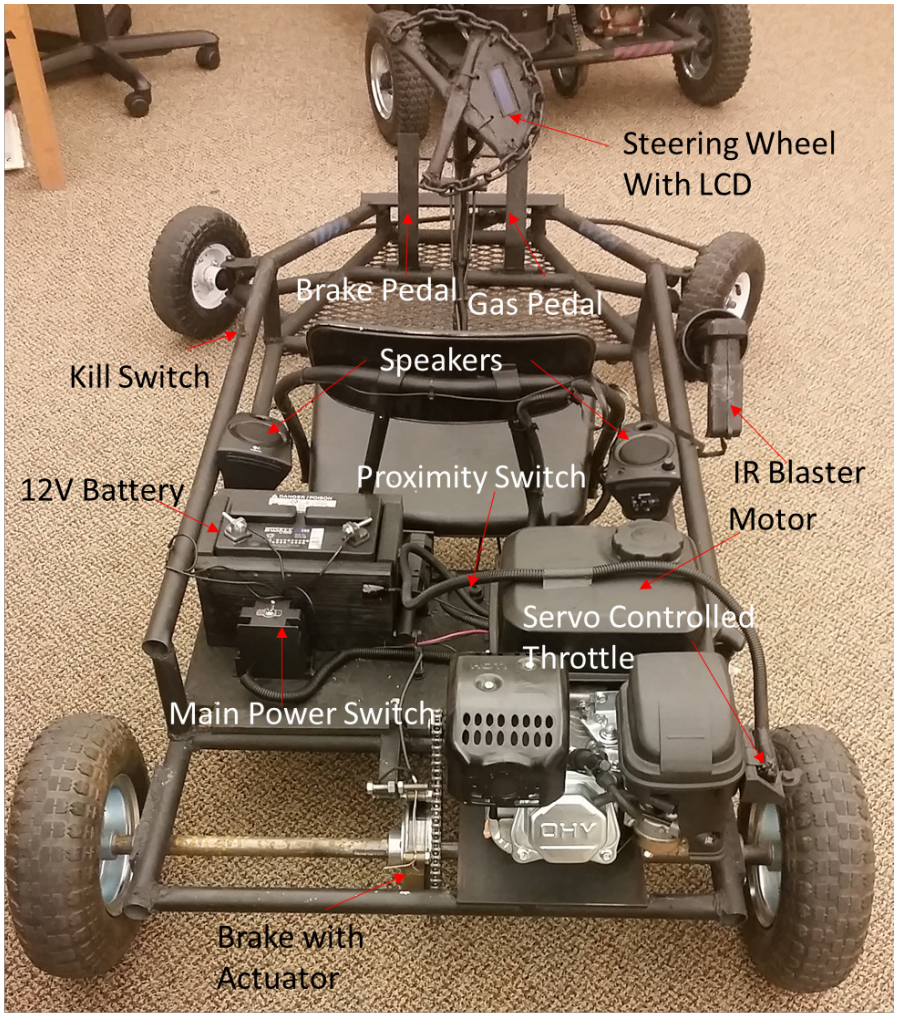


FIGURE 1: JUNKYARD BATTLE RACER CART

The proximity switch enables a “power up” which the driver can then use. There are four different types of power ups that the driver can receive. These power ups affect the speed the driver is going and if the driver is allowed to keep accelerating. To control the speeds of the cart, the pedals of each cart were connected to an actuator and servo. When activated, the actuator can apply the brakes of the cart without input from the driver. The throttle is controlled by the position of a servo motor. In normal driving mode, the usual driving speed before a power up is acquired, the servo only allows the throttle to be open about half way. When a speed boost is acquired, the servo allows the throttle to open as much as the driver wants. For safety concerns, the control of the brakes is never taken away from the driver and there is a kill switch implemented on the cart that will kill the cart motor if switched.

System Details

The aim of this project was to create two functioning go-carts that were electronically and mechanically enhanced to allow for a simulation of the video game Mario Kart. To achieve this goal, mechanical and electrical subsystems were implemented. Figure 2 below shows the overall project with labels of the major components while Figure 3 shows the view of the back of the cart where most of the major circuitry is located.

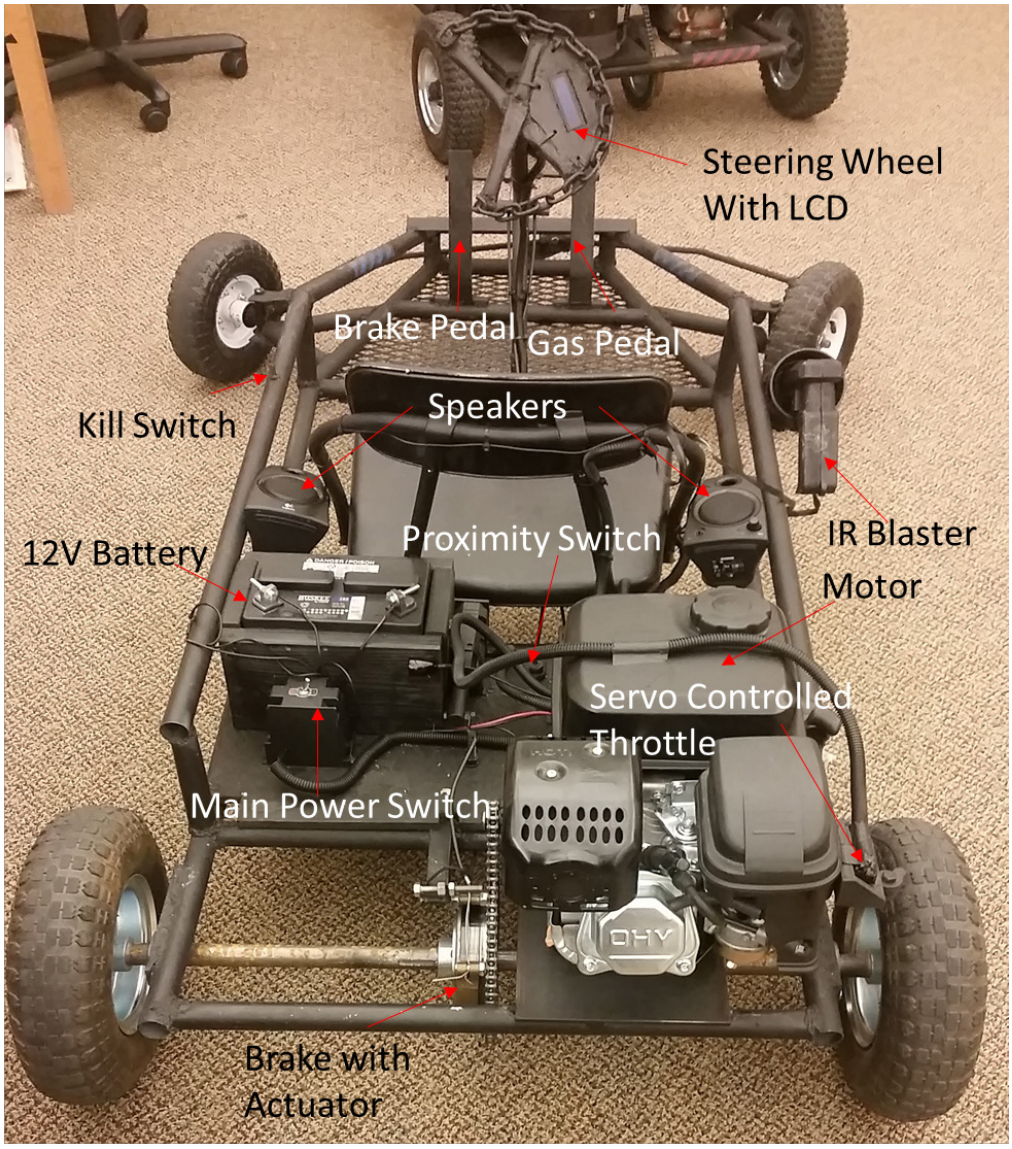


FIGURE 2: OVERVIEW OF PROJECT

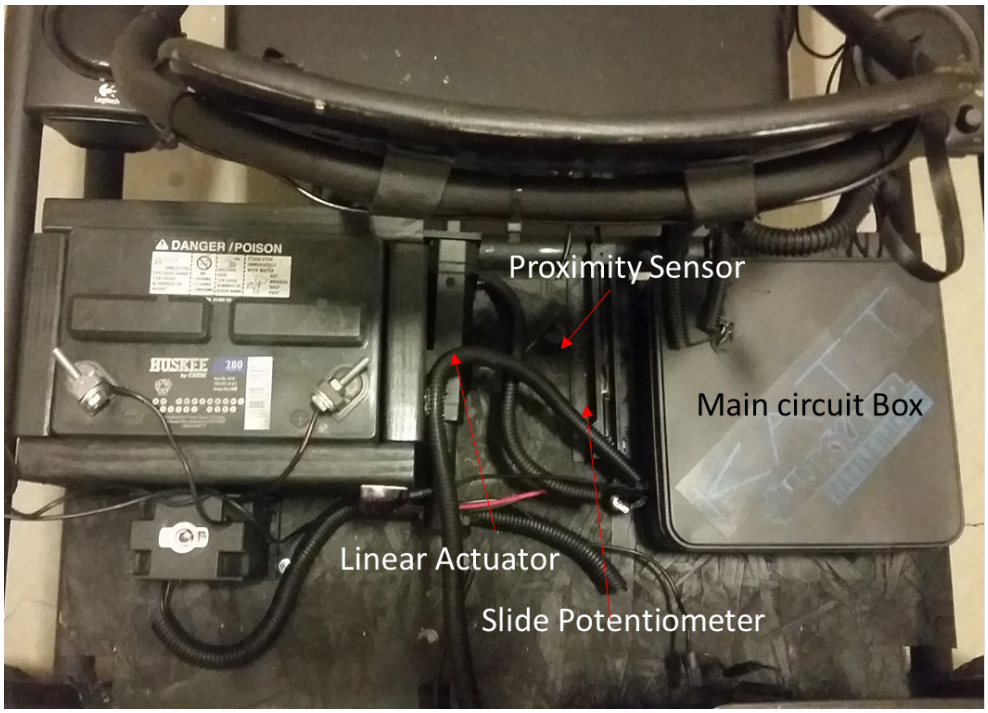


FIGURE 3: BACK OF CART

In order to simulate Mario Kart, certain power ups were designed. Table 1 below shows a breakdown of the power ups and a basic description of each.

POWER UP	DESCRIPTION	REQUIREMENTS
SPEED BOOST	Allows the servo to open more for faster speeds	Slide potentiometer, servo, and throttle linkages
EMP BLAST	Shuts down the opponents throttle	IR transmitter and receiver, servo, and throttle linkages
DEATH BEAM	Actuates the opponents brakes and shuts off the throttle	IR transmitter and receiver, servo, linear actuator, and throttle linkages
SHIELD	Prevents the use of the EMP Blast and Death Beam	Logic Implementation

TABLE 1: POWER UPS AND DESCRIPTIONS

Obtaining a power up is a simple matter of running over a raised base that sets off the proximity switch. For the purposes of this project, simple bases were made from cardboard. This allowed the driver to run over the bases without breaking anything. The power up obtained is randomized with some power ups ranked higher than others. The Death Beam, for example, is more difficult to get than the Speed Boost as it does more damage to the opponent.

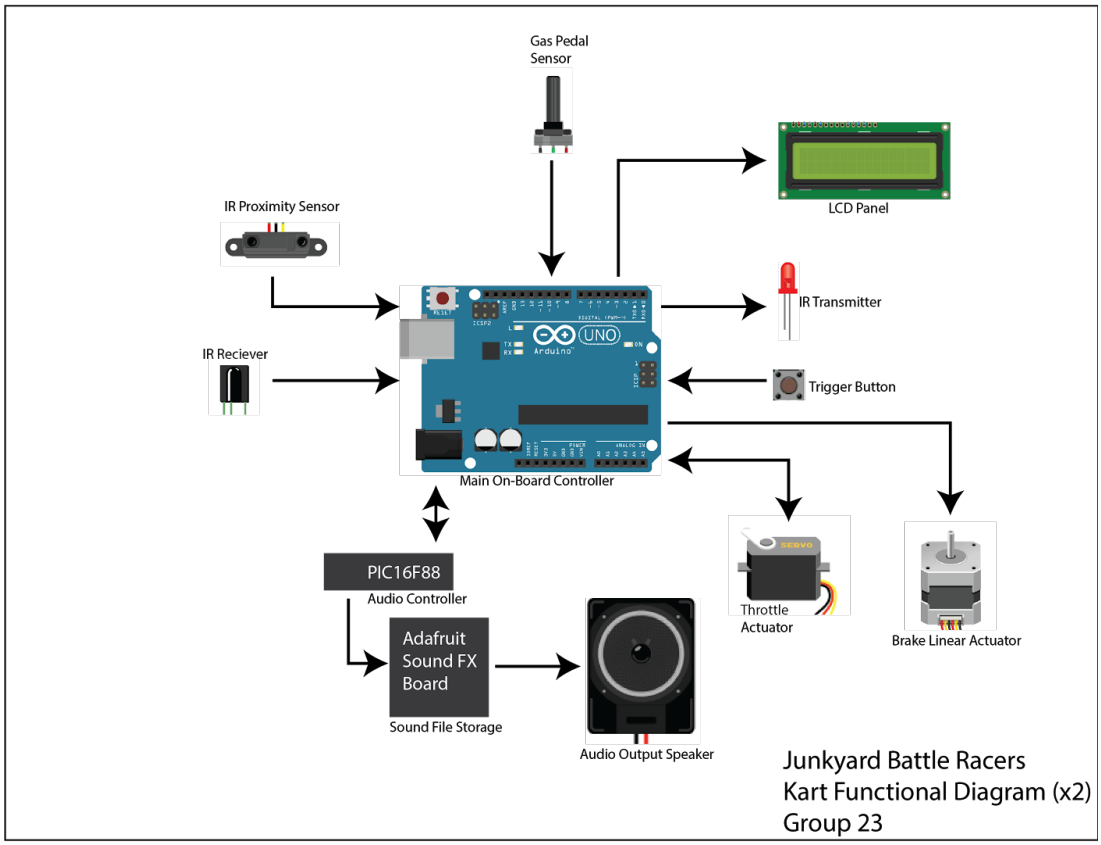


FIGURE 4: FUNCTIONAL DIAGRAM

Figure 4 displays the basic components and how they are interfaced with the microcontrollers. Figure 5 below shows the actual layout of the main circuit board. Most of the electronic and mechanical components are mounted on the cart at various locations, but they are all controlled with the Arduino and PIC at the back of the cart. The complete wiring diagram for both carts can be found in Appendix B.

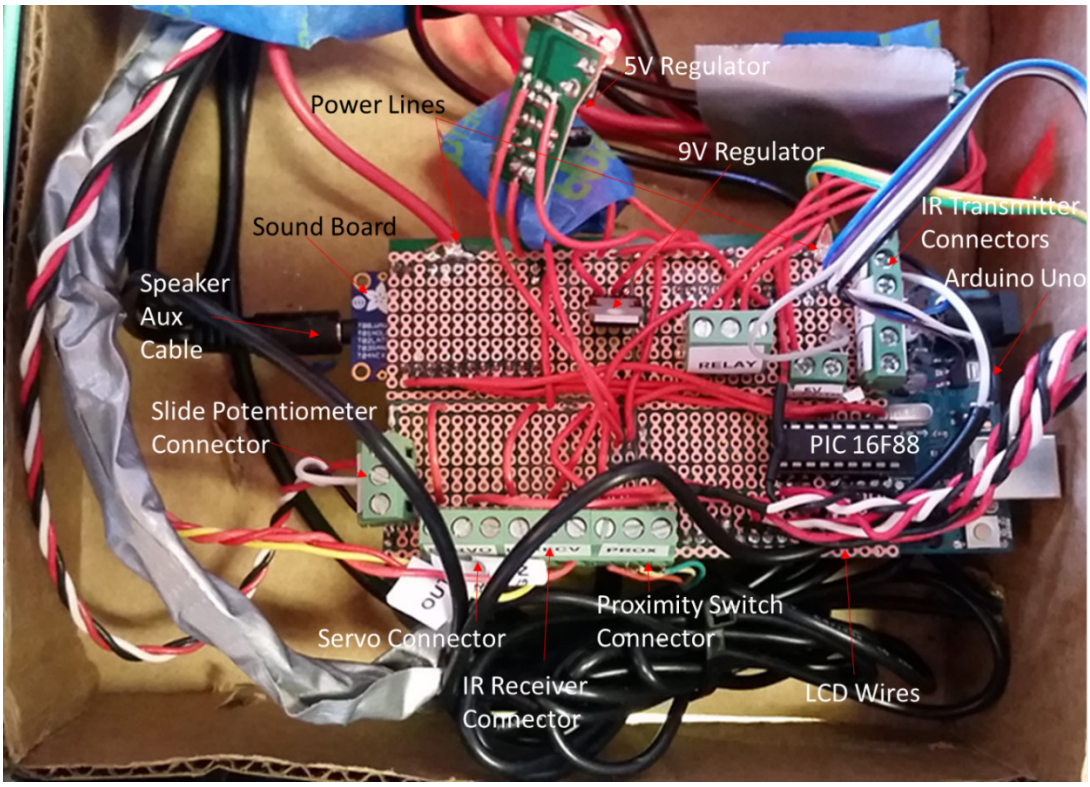


FIGURE 5: MAIN CIRCUIT BOARD

The pedal subsystem consists of an

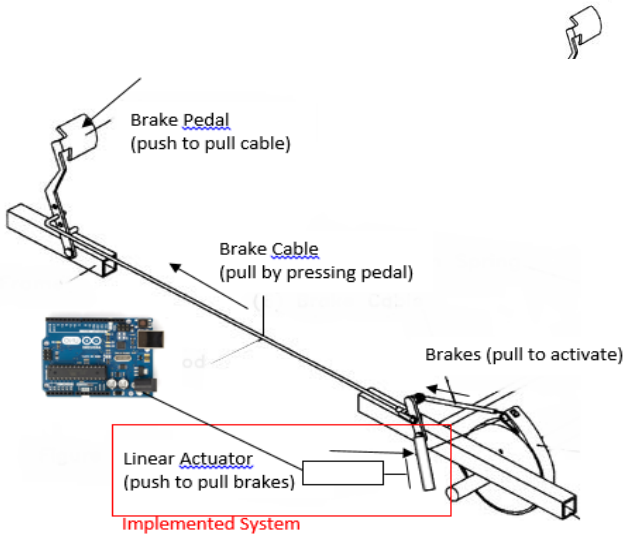
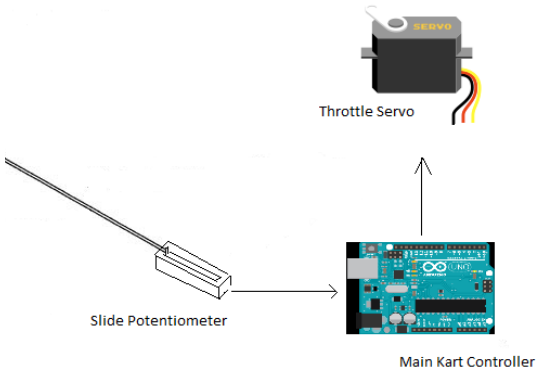


FIGURE 6: OVERVIEW OF THE BRAKE SYSTEM



IGURE 7: OVERVIEW OF THROTTLE PEDAL SYSTEM

actuator for the brakes, and slide potentiometer and servo for the throttle, and electronic controls for both. Figures 6 and 7 show an overview of the two pedals as they are difficult to see from the cart alone. The purpose of this system is to be able to control the brakes and throttle manually and via software. The brake pedal remains mechanically attached to the braking system for safety reasons, but also features a linear actuator which can apply the brakes when triggered by software. The brake pedal system can be seen in Figure 1 however, a clearer conceptual view can be seen in Figure 6. Unlike the brake pedal, the gas pedal is attached to a slide potentiometer, allowing the Arduino to read its position, decode the data and relay appropriate values to the servo controlling the throttle (Figure 7). Figure 8 shows the servo mounted on to the throttle of the motor. The servo is connected to the Arduino which communicates how far the throttle can open, thus controlling the speed. For safety precautions, an emergency kill switch is located on the left side of the go-cart in case of any problems and can be activated at any time by the driver. Additionally, a relay kill switch is attached to the battery which will kill the engine if the battery dies.

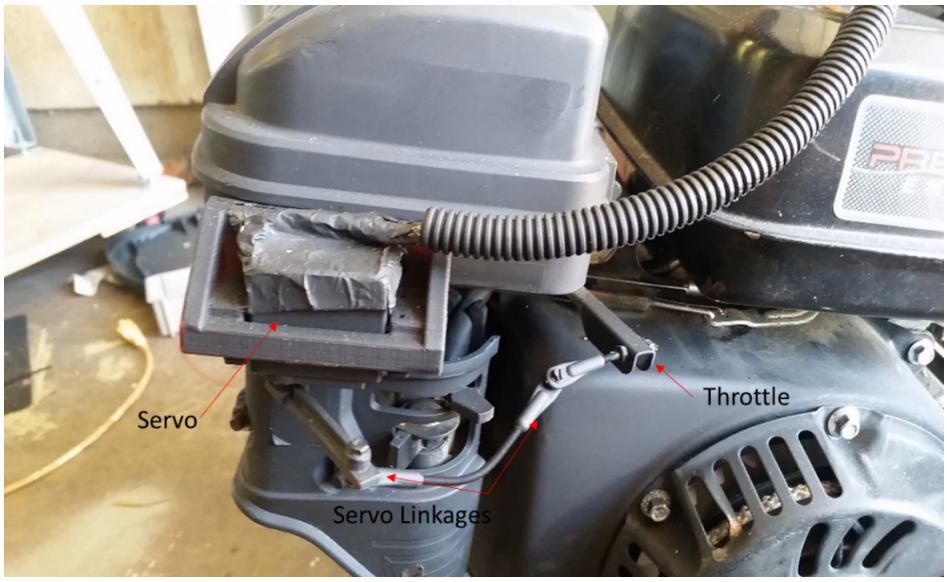


FIGURE 8: SERVO CONTROLLED THROTTLE

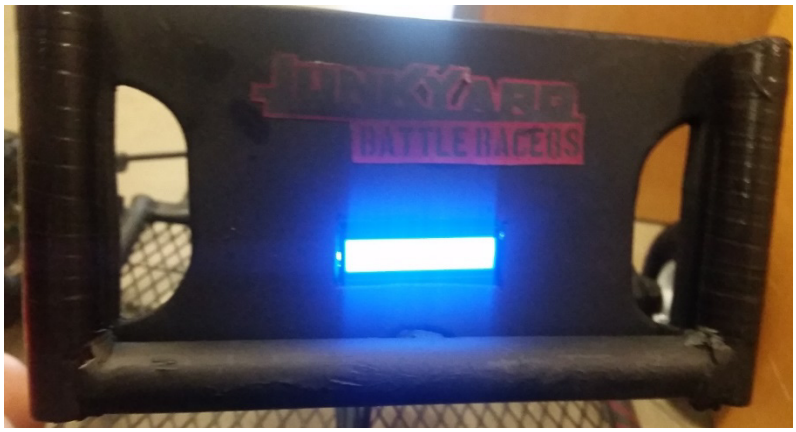


FIGURE 9: STEERING WHEEL WITH LCD

FIGURE 10: CLOSE UP OF LCD SCREEN

Figure 9 shows the LCD display on the steering wheel of the first cart with a close up to show that the LCD displays messages in Figure 10. The LCD is interfaced to the main circuit seen in Figure 5 and the wiring of the LCD can be seen in Figure 11. The wiring was slightly different for each cart. Cart one

required a potentiometer attached to pin 3 of the LCD to set the contrast while cart two did not require a potentiometer and pin 3 was grounded instead. To properly display messages to the LCD, pins 1, 5, and 16 were grounded while pins 2 and 15 were set to 5 volts. The LCD required 6 data lines to the Arduino and were wired to pins 4 through 9 (see Appendix B for the wiring diagram). The LCD wires were long and had to be shielded from the electromagnetic interference that was coming from the motor when it was started. To fix this problem, the LCD wires were braided together, wrapped in foil tape, and covered plastic lining. This solution worked and the LCD displays when the driver has a power up, the time limit of that power up, when the driver is hit by an opponent, when normal driving mode is restored, and the name of the cart.

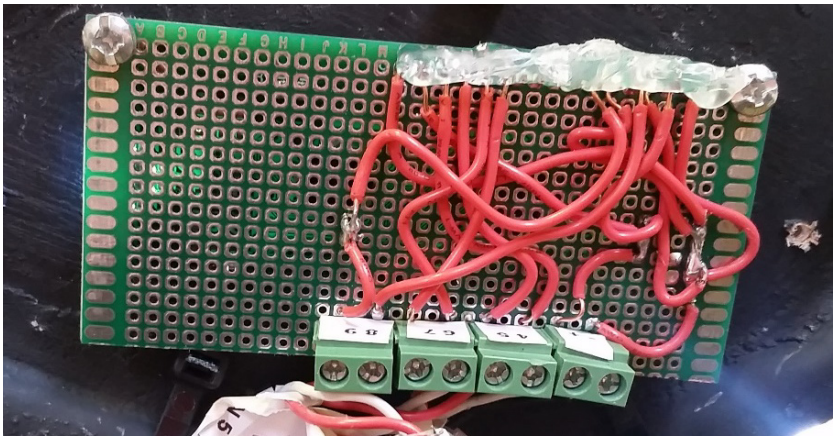


FIGURE 11: WIRING ON THE BACK OF LCD SCREEN

The audio subsystem consists of a sound board, a set of speakers, and a PIC 16F88. The sound board was preloaded with computer generated sounds that play when the PIC microcontroller receives a binary number from the Arduino via serial communication. The PIC then relays which audio bite will be played by grounding a specific pin on the sound board. The speakers are connected to the output (aux cord) of the sound board and are attached to the cart behind the driver (see Figure 1). The speakers are powered with the car battery after being stepped down by the 9V regulator (see Fig. 5). Figure 12 shows the sound card that is mounted to the back of the main circuit and Figure 5 shows the PIC mounted to the front of the main circuit board. The code to make this circuit work will be discussed later.

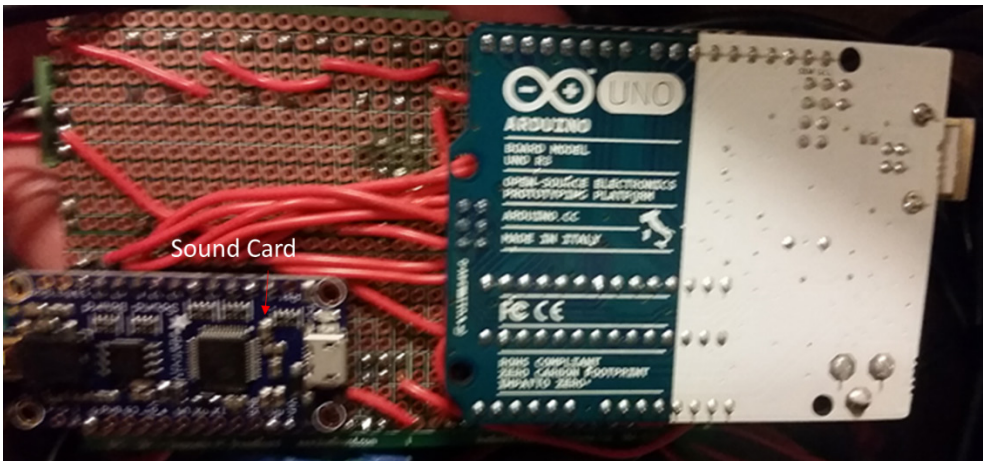


FIGURE 12: AUDIO CIRCUIT

The sensor subsystem in this project required multiple sensors. For communication between carts, IR transmitters were mounted in the 3D printed laser tag gun (Figures 14 and 15) and IR receivers were mounted in the main electronic box on the back of the cart (Figures 16 and 17). To activate an acquired power up, a push button was also mounted in the 3D printed gun that sends a signal to the other cart (Fig. 14). The proximity switch was mounted to the bottom of the cart (Fig. 13) so that objects above a certain height, when driven over, give the driver a power up.

FIGURE 13: PROXIMITY SENSOR AND SLIDE POTENTIOMETER

FIGURE 14: IR TRANSMITTER

FIGURE 15: CLOSE UP OF IR TRANSMITTER

FIGURE 16: COVERED IR RECEIVER

FIGURE 17: UNCOVERED IR RECEIVER

The IR subsystem consisted of a SFH 4544 high power infrared emitter, a TSOP75338WTR Infrared receiver module, lens, and a push button as seen in Figures 14 through 17. This subsystem communicates between the carts and affects the opponent player's driving. The diode modulates at a 38 KHz frequency and sends data with NEC protocol. When the IR receiver reads the output waveform from the IR transmitter, the Arduino decodes it as a serial bit stream. The warning, EMP, and death beam were configured to modulate at three different rates. Depending on the value read, the specified warning, EMP, or Death beam sound activates and logic allows for the correlating systems to become active.

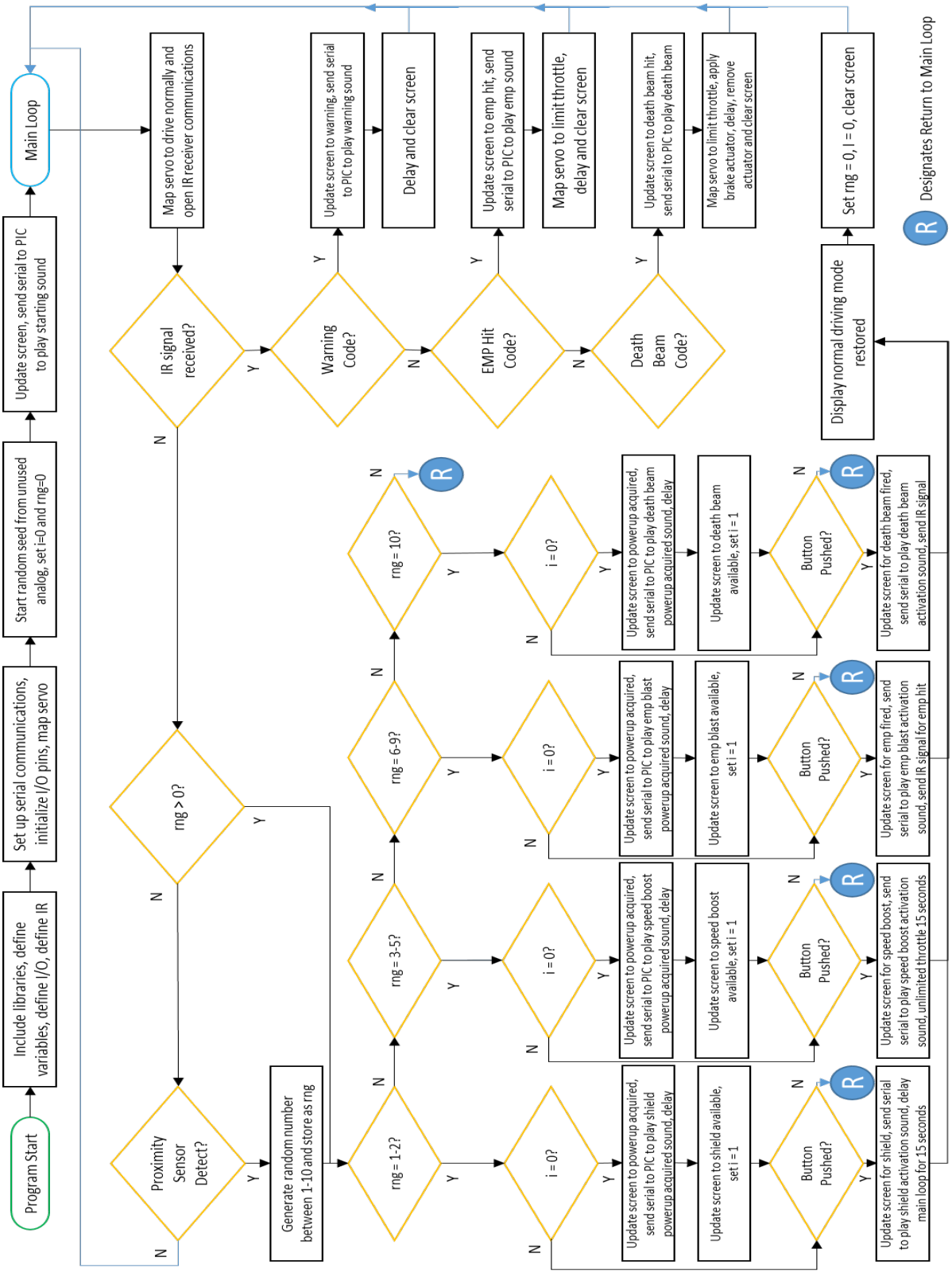


FIGURE 18: ARDUINO SOFTWARE FLOWCHART

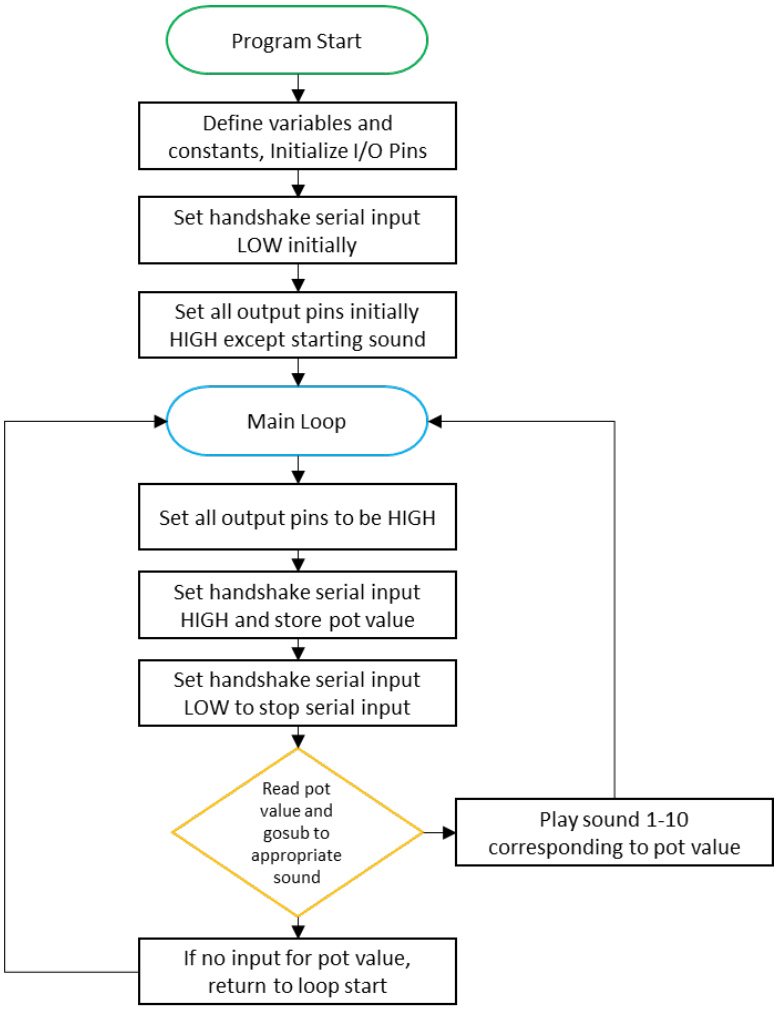


FIGURE 19: PIC SOFTWARE FLOWCHART

The program logic for the go-carts had to be written for both the Arduino Uno (Fig. 18) and the PIC (Fig. 19). The main functionalities of the go-carts were programmed into the Arduino sketch, and the PIC was used to handle the audio output which contained the computer generated sound files. Whenever a sound needs to be played anywhere from the Arduino flowchart in Figure 18, it would send a serial value between 1-10 to the PIC as there were 10 sounds total in our project. The PIC would then read a value 1-10 and ground the corresponding pin on the sound board. The reason the PIC was chosen to do this was because it was able to save a lot of pin space on the main Arduino board so that only one Arduino, which is more expensive, had to be used.

Design Evaluation

The project was able to meet all six functional categories in the project description. For output display, a 16x2 LCD display was mounted in the front of the cart (see Fig. 9) and wired to the main circuit board at the back of the cart (Fig. 5). The research required to implement the LCD screen was not as substantial as some other aspects of the project, but it was tricky to get the LCD to interface properly. The code for the LCD was easy as Arduino has a library for LCD screens but the wiring provided many problems. The LCD on the first cart required a 10k potentiometer for proper functionality while the LCD on the second cart did not require a potentiometer at all. Each LCD has its own circuit board separate from the main board which had to be soldered in addition to the main circuit board. There was a substantial length of wire running from both LCDs to the Arduino, so the eight wires needed were braided and shielded to cut down on interference and to save space. With the adjustments made, both LCDs function and work as required. The LCD displays a message to the driver when a power up is acquired, when the power up is in use, and when the other driver has a power up that could affect their cart.

To fulfill the audio output category, two computer speakers are mounted on the cart. The speakers are powered by the 12V battery running through a 9V regulator and are connected to a sound card that has multiple generated sound effects on it. The sound effects were compiled using Adobe Audition and contained audio from a free online text-to-speech engine and royalty free sound effects from freesounds.com. The completed sound effects were rendered as .ogg files and were then uploaded to the sound board which is connected to a PIC (Fig. 12). The PIC communicates serially with the Arduino to play each sound at a given time. Depending on the power up acquired, the Arduino sends a binary number to the PIC which sets a pin low and plays a sound depending on which pin is set low. This system should more than meet the requirements of the audio output category.

The trigger button in the 3D printed gun functions as the manual user input. The normally open button is connected to the Arduino and interfaces with the IR transmitter. When a power up is acquired, the

driver can trigger the power up with the button which will allow them to gain speed, shield themselves from their opponent, activate their opponent's brakes, or give their opponent no throttle. For safety purposes, the brakes are always operational. No electrical system can compromise the braking system and there is also a kill switch implemented that will kill the cart motor while leaving the brakes intact. The kill switch is only used in emergency, but it also requires input from the driver. The implementation of the button was relatively easy and only took minimal research, but the implementation of the kill switch took slightly more skill and research. We essentially pulled off the original kill switch built into the engine, and spliced onto the wire leads. We found that the kill switch worked by grounding a cable to the motor when switched off, so we ran extensions from this cable to our kill switch and kill relay. The two are wired in parallel, so that if one of the switches is activated, the motor will be grounded. Also worth noting is that to avoid further EMF issues from the motor, the go cart frame was used as a floating ground for the motor kill functionality, while the electronics on the cart were tied to the battery ground, with no connection to the cart frame. This complete physical isolation between the motor and electronics systems completely eliminates the possibility of electromagnetic interference via conduction, which ended up helping eliminate a lot of our EMI problem.

This project fulfills the automatic sensor category with multiple components. The first sensor implemented was a proximity switch (Fig. 13). When the driver runs over an obstacle set a certain height (part of the course), the proximity switch activates a power up. The power up is randomly chosen and can be any of the four power ups. Two of these power ups, the death beam and the EMP blast, will activate one part of the second sensor system, an IR transmitter (Fig. 14). An LED emitting IR light can be activated by either of those power ups. The power up, however, is not activated until the driver pulls the trigger button. During the time the IR transmitter is on and aimed at the other cart, the third sensor, the IR receiver, on the opposing cart detects the IR light from the first cart (Fig. 16). During this time, the cart will play a warning sound so the driver knows their opponent has a power up that can be used

against them. When the driver with the power up pulls the trigger button, their opponent's IR receiver will activate the linear actuator on the brakes or the servo on the throttle. The driver that has been shot will either have their brakes applied or their throttle cut off. This portion of the project required major research and effort to implement. The IR transmission and reception between the two carts took the most time to implement.

This project exceeds expectations in the actuators, mechanisms, and hardware category. The servo interfaces with the Arduino and controls the position of the throttle. In normal driving mode, the servo allows the throttle to be open only a fraction of the full speed. When a speed boost is acquired and activated, the servo allows for full throttle of the cart for 15 seconds. The brake actuator also fulfills this functional category. When the Arduino receives a death beam from the other cart, the actuator is activated and the brakes are applied to the cart. The actuator is connected to a wire that, when activated, pulls the wires to tighten the brake cable and slow the cart. In addition to the actuator and the servo that are implemented in the main circuitry, there are also multiple parts that were 3D printed or made for the carts. The linkages connecting the servo to the throttle were designed and printed in house along with the IR gun, the servo holder, the linear actuator mount, and the box for the throttle potentiometer (Appendix C).

The logic required to run the project was complex and difficult to program. In order to let the drivers have equal opportunity of each power up, each time the proximity switch is triggered a random number is generated which corresponds to each power up. After a power up is received, a digital audio file is played. The Arduino sends a binary number to the PIC using serial communication which grounds a certain pin on the sound board and plays a sound. Along with the randomization and serial communication, IR communication had to be enabled. Each cart contains an IR transmitter and an IR receiver. The transmitter sends a specified hex code to the receiver based on whether the power up has an IR component, has been activated, and the state of the trigger. This data is read by the Arduino and

can be used to actuate the brakes, shut off the throttle, or play a warning sound in response. This part of the logic was by far the hardest as the IR receivers had difficulties picking up the signal transmitted. The code that runs this project should fulfill the requirements of the project description. Most of the logic in the code was not taught in class and required lots of independent research and testing (see Appendices D and E for logic).

Group 23 anticipates a few qualitative adjustments for the project including extra early bird, level of effort, and quality. The group was able to demonstrate functionality of the project by the deadline for extra early bird, so a plus 10 is anticipated. The group also put a lot of work into this project and expect a plus 10 for level of effort. In addition, the final product is appealing, creative, and original so a plus 10 for how presentable and complete the project is also anticipated.

Partial List of Parts

Name	*Vendor*	*Price*	*Description*
Servo	Amazon.com	$11	Controls throttle on motor
Arduino Uno	Arduino	$30	Main microcontroller
Sound Board	Adafruit	$24	Circuit board for audio output
Linear Actuator	Windy Nation	$40	Mechanism for braking system
Proximity Sensor	Geeetech	$8	Sensor to detect power up pads
IR Transmitter	Mouser Electronics	$5.67	Sends IR communication
IR Receiver	Mouser Electronics	$8.90	Module to receive IR communication
Relay	SunFounder US	$6	Switch to automatically turn on actuator and servo
Slide Potentiometer	Data Alchemy	$5	Detects changes in gas pedal position to map to servo
Kill Switch Relay	Amazon.com	$7	Safety switch to kill motor
LCD Display	Amazon.com	$8	Displays power ups and messages to driver

Lessons Learned

As with every project, the team faced multiple obstacles along the way. One of the main issues was time management. The team worked diligently over the entirety of the semester but still faced challenges finishing the project by the due date. Some recommendations for future mechatronics groups would be to elect a leader early on in the project, set up designated times to meet every week, and set up a schedule. Having an elected leader who is in charge of the project can help move the project along faster. Finding a time when everyone can meet can be tricky, but agreeing to a set time early in the semester can help cut down on conflict later on in the project. A basic schedule of due dates and meeting times would have helped keep the project on track and focused.

Another main issue faced was the building and interfacing of circuitry. The main circuit needed multiple wires and connectors soldered to it, which was time consuming and not always effective. One recommendation would be to have circuit boards printed by a company to cut down on the soldering required. The only issue with printed boards would be that once the board is printed, making changes would be difficult. Interfacing to the circuit board was also an issue as the main motor on the project was creating electromagnetic interference. Some components like the LCD screen and the servo were receiving that interference and not functioning properly. This issue was a major time setback because it had to be addressed before moving forward. Shielding wires, especially longer wires, is a good precaution on all future projects. A few great steps for fixing electromagnetic interference issues were braiding wires together and wrapping wires in HVAC foil tape. It is also a good idea to electronically isolate parts of the circuit that may be causing issues--in our case the gas engine.

A major problem faced during the semester was parts breaking or not properly functioning. The LCD screens were a big issue. Even when wired correctly, the screen did not work properly and another screen was shorted and burnt out. To prevent this problem, order extra parts and order parts early. Last minute replacements can be expensive and can prevent completion of projects.

The last major problem faced for this project was the coding. Delegation of components and coding works, but when trying to integrate each of those systems, problems arose. One recommendation is to start compiling and integrating circuits early on to leave time for trouble shooting. A fairly high level of creativity may be required to get all subsystems running together simultaneously. It's important to remember that microcontrollers don't multitask. We ran into some issues with continuously mapping the potentiometer values to our servo while continuing to run all of our other code. As a result we were unable to use any delay functions in our code or the throttle would momentarily become stuck in its last position. To fix this, a custom function had to be called on to add delays. This essentially broke the specified delay time into very tiny components and ran a for loop which contained instructions to read and map the throttle and then delay for a very short amount of time repeatedly for the specified length.

Works Cited

Alciatore, Dave. "MECH307 - Mechatronics and Measurement Systems."*MECH307 - Mechatronics and Measurement Systems*. N.p., n.d. Web. 02 Feb. 2016.

Alciatore, David G., and Michael B. Histand. *Introduction to Mechatronics and Measurement Systems*. 4th ed. New York: McGraw-Hill, 2012. Print.

"Arduino Laser Tag - Duino Tag." *Instructables.com*. N.p., n.d. Web. 15 Apr. 2016.

"Building a Foot Pedal Control." *Welding Design and Fabrication*. N.p., n.d. Web. 10 Apr. 2016.

"Connecting an LCD to the Arduino." *Instructables.com*. N.p., n.d. Web. 23 Apr. 2016.

GillSensors. "Accelerator Pedal Position Sensor." *YouTube*. YouTube, 16 Nov. 2011. Web. 8 Apr. 2016.

"I2C between Arduinos." *Instructables.com*. N.p., n.d. Web. 10 Apr. 2016.

"LM386 Based Stereo Audio Amplifier with Digital Volume Control - Embedded Lab." *Embedded Lab*. N.p., 26 June 2011. Web. 10 Apr. 2016.

"SD/SDHC Card Sound Recorder." *SD/SDHC Card Sound Recorder*. N.p., n.d. Web. 10 Apr. 2016.

"VISHAY TSOP32338SS1V Infrared Receiver, Remote Control, 38 KHz, 45m, 45 °, 2.5 V, 5.5 V, 450 µA." *TSOP32338SS1V*. N.p., n.d. Web. 22 Apr. 2016.

Appendix A: Data Sheets

MICROCHIP

PIC16F87/88

18/20/28-Pin Enhanced Flash MCUs with nanoWatt Technology

Low-Power Features:

- Power-Managed modes:
  - Primary Run: RC oscillator, 76 µA, 1 MHz, 2V
  - RC_RUN: 7 µA, 31.25 kHz, 2V
  - SEC_RUN: 9 µA, 32 kHz, 2V
  - Sleep: 0.1 µA, 2V
- Timer1 Oscillator: 1.8 µA, 32 kHz, 2V
- Watchdog Timer: 2.2 µA, 2V
- Two-Speed Oscillator Start-up

Oscillators:

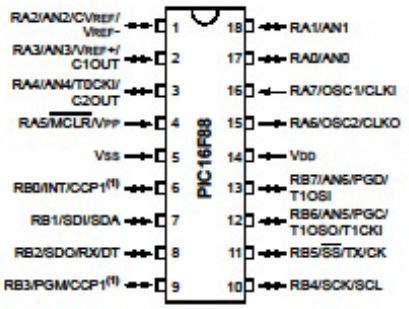
- Three Crystal modes:
  - LP, XT, HS: up to 20 MHz
- Two External RC modes
- One External Clock mode:
  - ECIO: up to 20 MHz
- Internal oscillator block:
  - 8 user selectable frequencies: 31 kHz, 125 kHz, 250 kHz, 500 kHz, 1 MHz, 2 MHz, 4 MHz, 8 MHz

Peripheral Features:

- Capture, Compare, PWM (CCP) module:
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit, 7-channel Analog-to-Digital Converter
- Synchronous Serial Port (SSP) with SPI (Master/Slave) and I^2C™ (Slave)
- Addressable Universal Synchronous Asynchronous Receiver Transmitter (AUSART/SCI) with 9-bit address detection:
  - RS-232 operation using internal oscillator (no external crystal required)
- Dual Analog Comparator module:
  - Programmable on-chip voltage reference
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible

Pin Diagram

18-Pin PDIP, SOIC

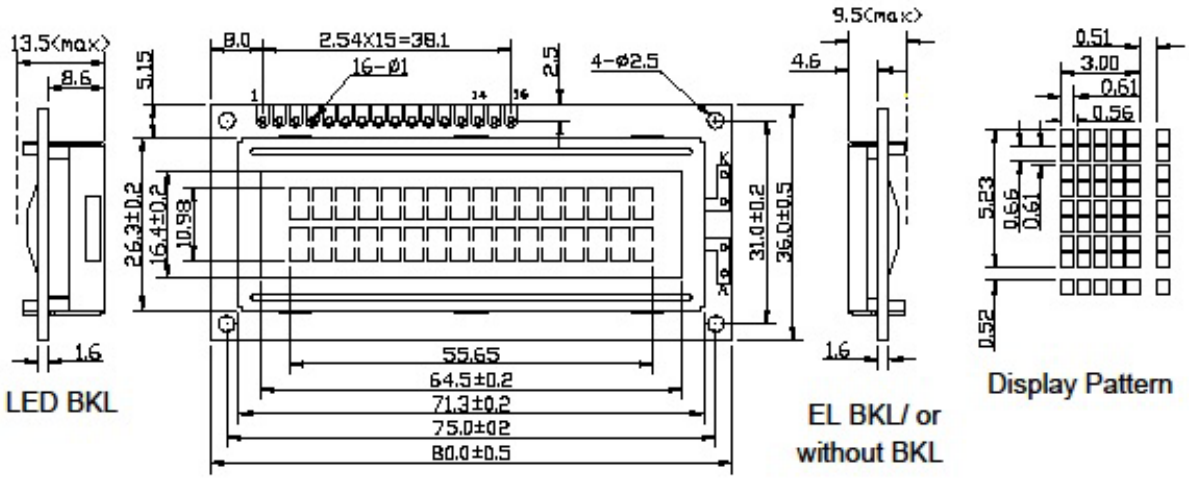


Note 1: The CCP1 pin is determined by the CCPMX bit in Configuration Word 1 register.

Special Microcontroller Features:

- 100,000 erase/write cycles Enhanced Flash program memory typical
- 1,000,000 typical erase/write cycles EEPROM data memory typical
- EEPROM Data Retention: > 40 years
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Processor read/write access to program memory
- Low-Voltage Programming
- In-Circuit Debugging via two pins
- Extended Watchdog Timer (WDT):
  - Programmable period from 1 ms to 268s
- Wide operating voltage range: 2.0V to 5.5V

Device	Program Memory		Data Memory		I/O Pins	10-bit A/D (ch)	CCP (PWM)	AUSART	Comparators	SSP	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)							
PIC16F87	7168	4096	368	256	16	N/A	1	Y	2	Y	2/1
PIC16F88	7168	4096	368	256	16	1	1	Y	2	Y	2/1



Feature

1. 5X8 dots with cursor
2. Built-in controller (KS0066U or Equivalent)
3. +5V power supply(Also available for +3.0V)
4. 1/16 duty
5. BKL to be driven by pin1,pin2,or pin15,pin16,or A,K
6. N.V.optional

PIN NO	Symbol	Fuction
1	VSS	GND
2	VDD	+5V
3	V0	Contrast adjustment
4	RS	H/L Register select signal
5	R/W	H/L Read/Write signal
6	E	H/L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line
15	A	+4.2V for LED
16	K	Power supply for BKL(0V)

Mechanical Data

Item	Standard	Unit
Module dimension	80.0x36.0	mm
Viewing area	64.5x16.4	mm
Dot size	0.56x0.61	mm
Character size	3.00x5.23	mm

Absolute Maximum Rating

Item	Symbol	Standard			Unit
		Min	Typ	Max	
Power supply	VDD-VSS	-0.3	——	5.5	V
Input voltage	VI	-0.3	——	VDD	

Electronical characteristics

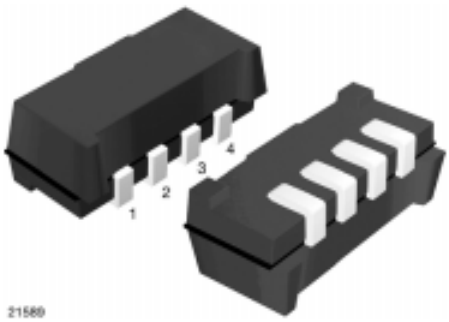
Item	Symbol	Condition	Standard			Unit
			Min	Typ	Max	
Input voltage	VDD	+5V	4.7	5.0	5.5	V
		+3.3V	2.7	3.0	5.3	V
Supply current	IDD	VDD=5V	——	1.5	4	mA
Recommended LCD riling voltage for normal temp version module	VDD-V0	-20°C	——	——	——	V
		0 °C	4.7	5.0	5.5	
		25°C	4.3	4.5	4.7	
		50°C	4.1	4.3	4.5	
		70°C	——	——	——	
LED forward voltage	VF	25°C	——	4.2	4.6	V
LED forward current	IF	25°C	——	120	160	mA
EL power supply current	IEL	VIN=110V AC 400Hz	——	——	——	mA

Display character address code:

Display position

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	—	—	—	—	—								0FH
DDRAM address	40	41	42	—	—	—	—	—								4FH

IR Receiver Modules for Remote Control Systems



FEATURES

- Very low supply current
- Photo detector and preamplifier in one package
- Compatible also with short burst dataformats
- Supply voltage: 2.5 V to 5.5 V
- Improved immunity against ambient light
- Capable of side or top view
- Low profile 2.35 mm
- Insensitive to supply voltage ripple and noise
- Material categorization: for definitions of compliance please see www.vishay.com/doc?99912

RoHS COMPLIANT
HALOGEN FREE
GREEN (5-2008)

MECHANICAL DATA

Pinning:

1, 4 = GND, 2 = V_S, 3 = OUT

ORDERING CODE

Taping:

TSOP75...WTT - top view taped

TSOP75...WTR - side view taped

DESCRIPTION

The TSOP753..W, TSOP755..W series are a miniaturized receiver module for infrared remote control systems. Two PIN diodes and a preamplifier are assembled on a leadframe, the epoxy package contains an IR filter.

The TSOP753..W..series is optimized to suppress almost all spurious pulses from energy saving lamps like CFLs. AGC3 may also suppress some data signals if continuously transmitted.

The TSOP755..W series contain a very robust AGC5. This series should only be used for critically noisy environments.

This component has not been qualified according to automotive specifications.

PARTS TABLE			
AGC		NOISY ENVIRONMENTS AND SHORT BURSTS (AGC3)	VERY NOISY ENVIRONMENTS AND SHORT BURSTS (AGC5)
Carrier frequency	30 kHz	TSOP75330W	TSOP75530W
	33 kHz	TSOP75333W	TSOP75533W
	36 kHz	TSOP75336W (1)	TSOP75536W
	38 kHz	TSOP75338W (2)(3)(4)(5)	TSOP75538W
	40 kHz	TSOP75340W	TSOP75540W
	56 kHz	TSOP75356W	TSOP75556W
Package		Heimdall no lens	
Pinning		1, 4 = GND, 2 = V_S, 3 = OUT	
Dimensions (mm)		6.8 W x 3.0 H x 2.35 D	
Mounting		SMD	
Application		Remote control	
Best remote control code		(1) MCIR (2) Mitsubishi (3) RECS-80 Code (4) r-map (5) XMP-1, XMP-2	

Infrared Emitter (940 nm)
Version 1.3

SFH 4544

Features:

- Wavelength 950nm
- Narrow half angle ± 10°
- Short switching times
- UL version available (ordering code & test conditions on request)

Applications

- Infrared Illumination for cameras
- Sensor technology
- Data transmission

Notes

Depending on the mode of operation, these devices emit highly concentrated non visible infrared light which can be hazardous to the human eye. Products which incorporate these devices have to follow the safety precautions given in IEC 60825-1 and IEC 62471.

Ordering Information

Type:	Radiant Intensity I_e [mW/sr] **I_F= 100 mA, t_p= 20 ms**	Ordering Code
SFH 4544	550 (≥ 250)	Q65111A4886

Note: Measured at a solid angle of Ω = 0.001 sr

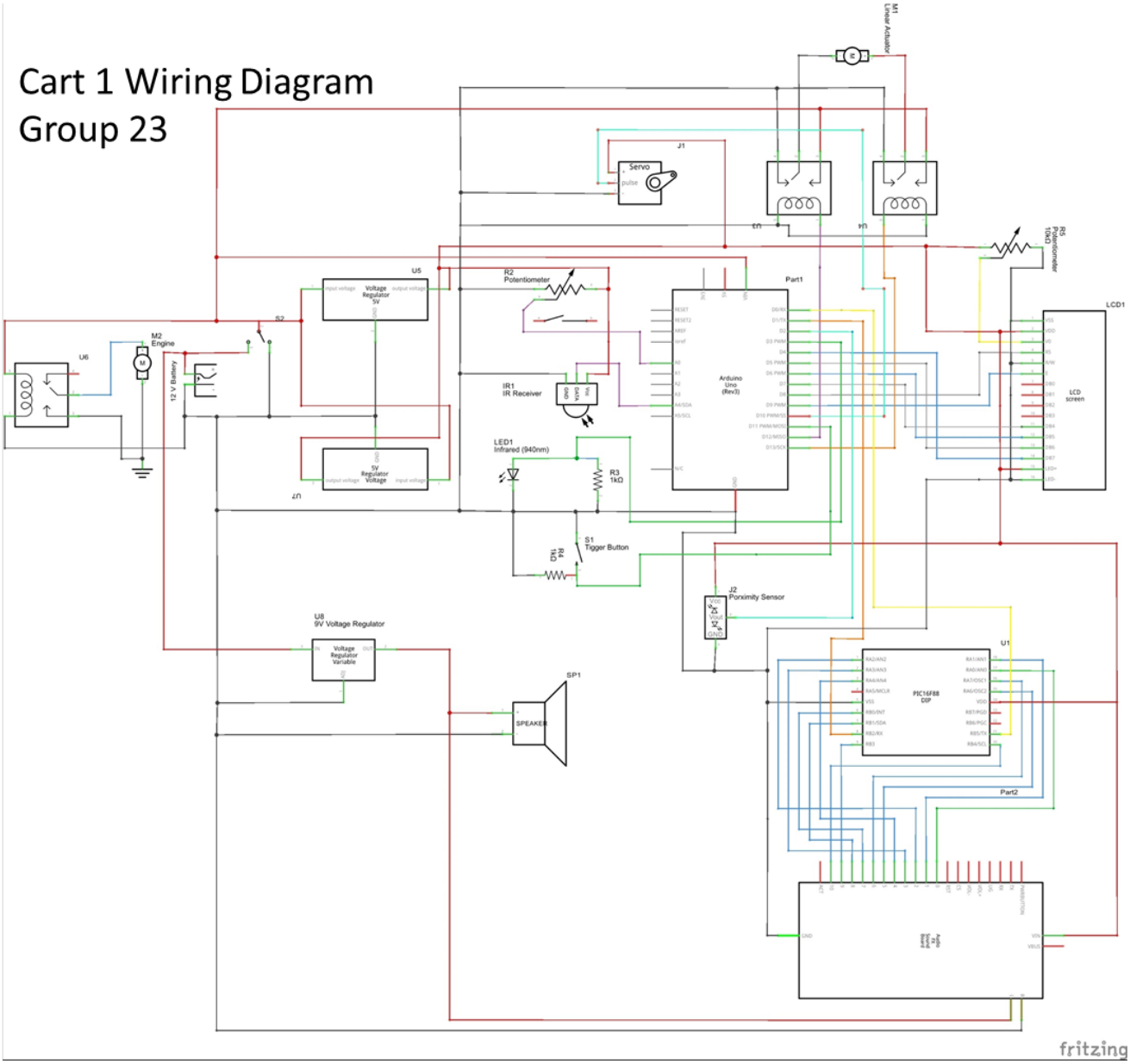
Maximum Ratings (T_A = 25 °C)

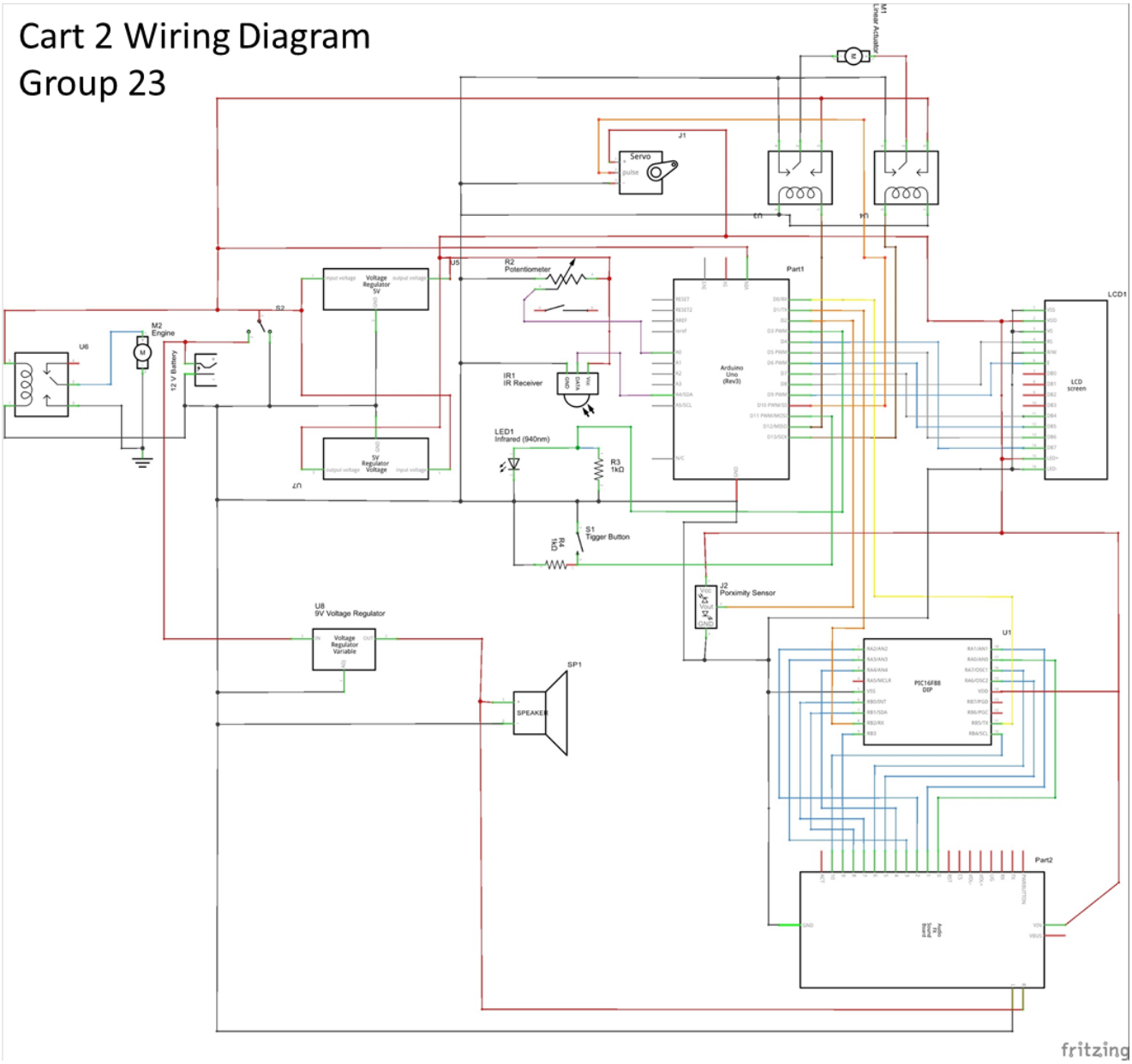
Parameter	Symbol	Values	Unit
Operation and storage temperature range	T_{op}; T_{stg}	-40 ... 100	°C
Reverse voltage	V_R	5	V
Forward current	I_F	100	mA
Surge current (t_p ≤ 200 µs, D = 0)	I_{FSM}	1	A
Power consumption	P_{tot}	200	mW
ESD withstand voltage (acc. to ANSI/ ESDA/ JEDEC JS-001 - HBM)	V_{ESD}	2	kV
Thermal resistance junction - ambient 1) page 8	R_{thJA}	430	K / W
Thermal resistance junction - soldering point	R_{thJS}	240	K / W

Characteristics (T_A = 25 °C)

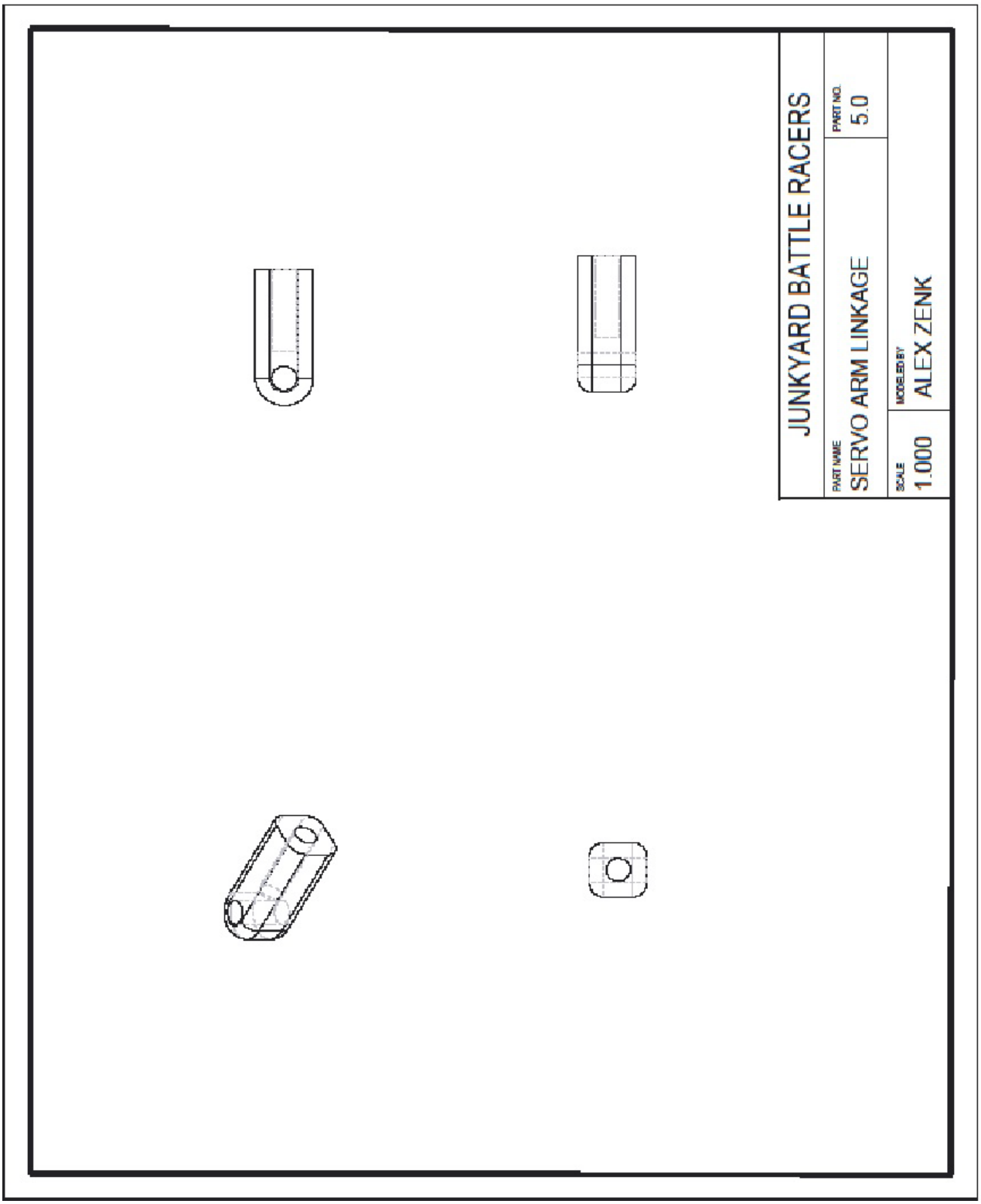
Parameter		Symbol	Values	Unit
Peak wavelength (I_F = 100 mA, t_p = 20 ms)	(typ)	λ_{peak}	950	nm
Centroid wavelength (I_F = 100 mA, t_p = 20 ms)	(typ)	$\lambda_{centroid}$	940	nm
Spectral bandwidth at 50% of I_{max} (I_F = 100 mA, t_p = 20 ms)	(typ)	Δλ	42	nm
Half angle	(typ)	φ	± 10	°
Dimensions of active chip area	(typ)	L x W	0.3 x 0.3	mm x mm
Rise and fall time of I_e (10% and 90% of $I_{e\,max}$) (I_F = 100 mA, R_L = 50 Ω)	(typ)	t_r, t_f	12	ns
Forward voltage (I_F = 100 mA, t_p = 20 ms)	(typ (max))	V_F	1.6 (≤ 1.9)	V
Forward voltage (I_F = 1 A, t_p = 100 µs)	(typ (max))	V_F	3.6 (≤ 4.6)	V
Reverse current (V_R = 5 V)		I_R	not designed for reverse operation	µA
Total radiant flux (I_F=100 mA, t_p=20 ms)	(typ)	Φ_e	75	mW

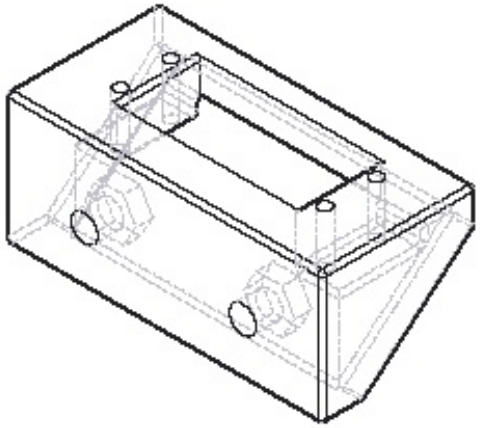
Appendix B: Wiring Diagrams

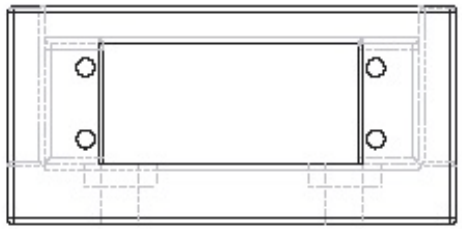


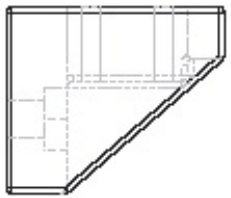

Cart 2 Wiring Diagram
Group 23
M1
Linear Actuator
J1
Servo
pulse
U3
U4
U5
R2
Potentiometer
Voltage
Regulator
5V
S2
M2
Engine
U6
12 V Battery
IR1
IR Receiver
GND
DATA
Vcc
Part1
Arduino
Uno
(Rev3)
LCD1
LCD
screen
LED1
Infrared (940nm)
R3
1kΩ
U7
5V
Regulator
Voltage
R4
1kΩ
S1
Tigger Button
J2
Ponximity Sensor
U8
9V Voltage Regulator
Voltage
Regulator
Variable
SP1
SPEAKER
U1
PIC16F88
DIP
Part2
fritzing

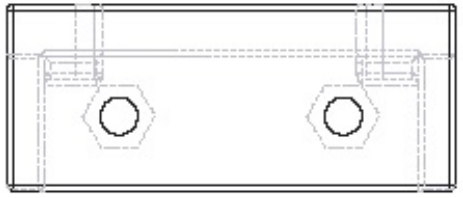
Appendix C: 3D Printed Parts Drawing



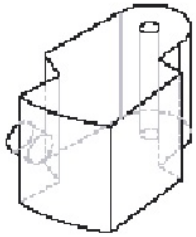


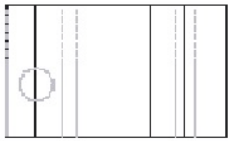




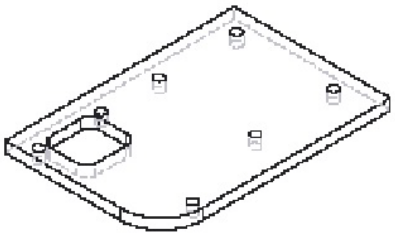


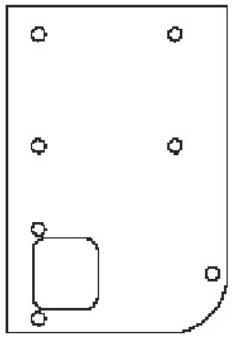
JUNKYARD BATTLE RACERS	
PART NAME THROTTLE SERVO MOUNT	PART NO. 2.0
SCALE 1.000	MODELED BY ALEX ZENK

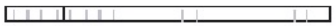




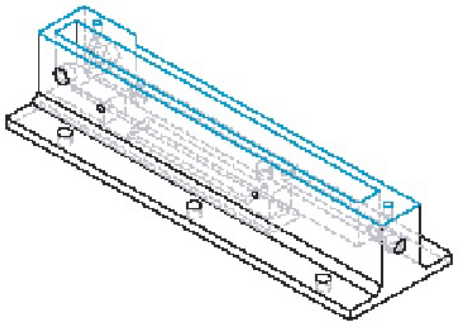
JUNKYARD BATTLE RACERS	
PART NAME: LASER TAG GUN (TRIGGER)	PART NO.: 3.3
SCALE: 1.000	MODELED BY: ALEX ZENK

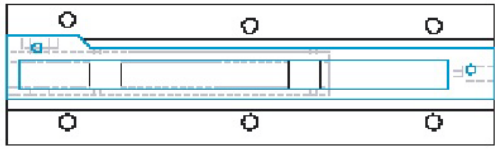




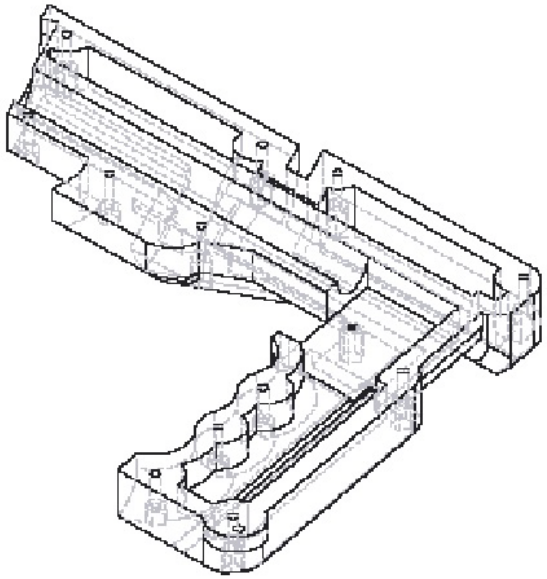


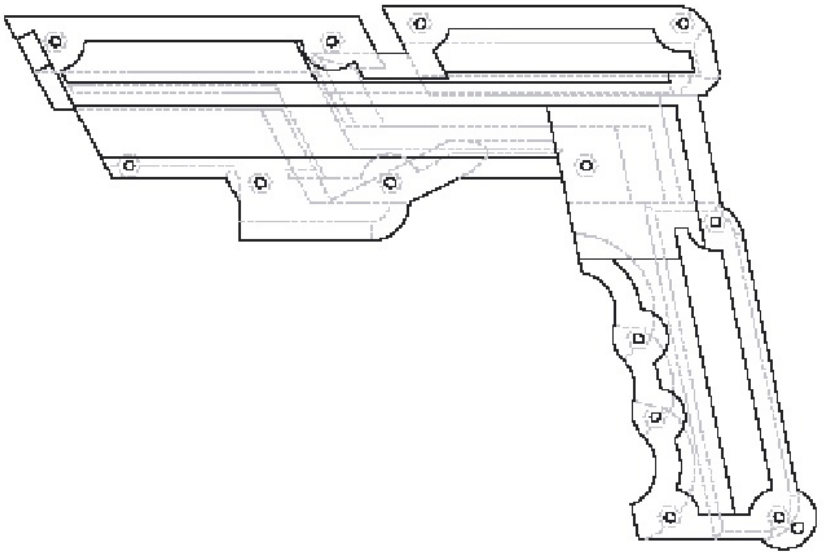
JUNKYARD BATTLE RACERS	
PART NAME: LINEAR ACTUATOR MOUNT	PART NO.: 4.0
SCALE: 0.500	MODELED BY: ALEX ZENK

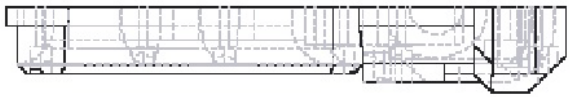




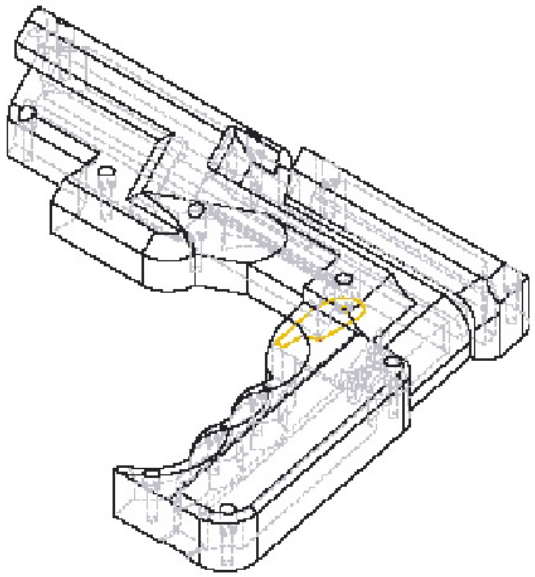
JUNKYARD BATTLE RACERS	
PART NAME: THROTTLE POT BOX	PART NO.: 1.0
SCALE: 0.5	MODELED BY: ALEX ZENK

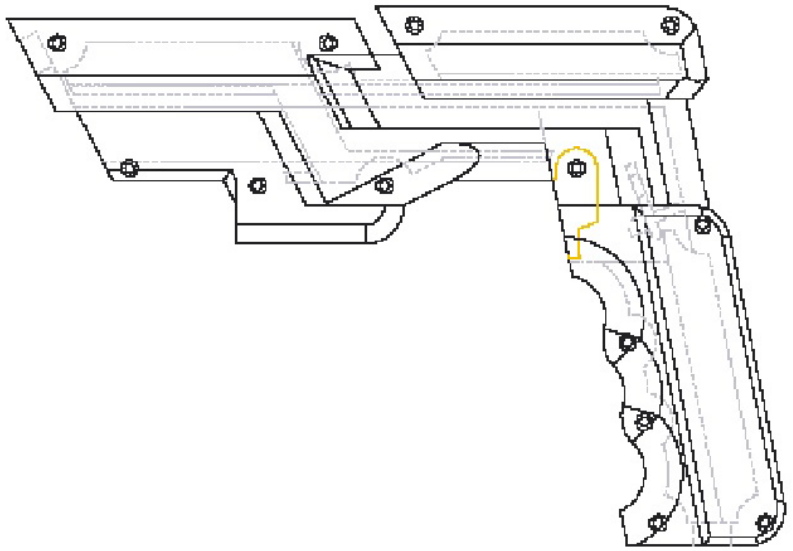






JUNKYARD BATTLE RACERS	
PART NAME: LASER TAG GUN (PART 1)	PART NO.: 3.1
SCALE: 0.500	MODELED BY: ALEX ZENK





JUNKYARD BATTLE RACERS	
PART NAME LASER TAG GUN (PART 2)	PART NO. 3.2
SCALE 0.500	MODELED BY ALEX ZENK

Appendix D: Arduino Code

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

   Junkyard Battle Racers: Kart 1 Firmware

   Final Build Version V.4.0

   Mechatronics Spring 2016: Group 23

   Alex Zenk

   Katie Johnson

   Floyd Bundrant

   Jake Gover

/*********************************************************************************
 PRE-PROGRAM INITIALIZATION
*********************************************************************************/
#include <IRremote.h>
#include <IRremoteInt.h>
#include <Servo.h> //includes library for servo
#include <LiquidCrystal.h> //includes library for lcd
LiquidCrystal lcd(8, 9 , 7, 6, 5 , 4); //what pins are on the lcd
Servo throttle; //create servo object to control a servo
const int i0 = 0; //initial iteration value as 0
//int i = 0; //initial iteration declared as 0 initially
const int rng0 = 0; //initial rng value as 0
//int rng = 0 //set the randomly generated number value to be 0 initially
const int powerupsense = 2; //pin 2 is connected to normally high output of proximity sensor
const int trigger = 11; //pin 11 is connected to gun trigger NO button
const int relay_a = 12; //pin 12 is connected to actuator relay A
const int relay_b = 13; //pin 13 is connected to actuayor relay B
const int IRtransmit = 3; //pin 10 is PWM and is connected to the IR transmitter
const int IRreceive = A4; //Analog pin A4 is connected to the IR receiver
const int gaspedal = A0; //Analog pin A0 is connected to the gas pedal potentiometer

IRrecv irrecv(IRreceive); //initiate IR libraries.
IRsend irsend;
decode_results results; //define results  for IR receiver
int powerupstate; //variable to read powerup state.
int gasval; //variable to read gas potentiometer value
int triggerstate; //variable to read trigger state.
int rng; //variable to store random item number
int i; //variable to store whether powerup is available.
/*********************************************************************************
 MAIN PROGRAM SETUP
 *********************************************************************************/
void setup() {
```

```
 Serial.begin(9600); //initialize serial communication with a baud rate of 9600
  Serial.write(0); //Plays starting song, sound 0
 //Initialize I/O pins
 pinMode (powerupsense, INPUT); //powerup sensor is an input
 pinMode (trigger, INPUT); //blaster trigger is an input
 pinMode (relay_a, OUTPUT); //linear actuator relay a is an output
 pinMode (relay_b, OUTPUT);//linear actuator relay b is an output
 pinMode (IRtransmit, OUTPUT); //IR transmitter is an output
 //pinMode (IRreceive, INPUT); //IR receiver is an input
 throttle.attach(10); //servo control on pin 10
 throttle.writeMicroseconds(2200); //initially set throttle to idle position
 int i = i0; //initially set i to 0 (no power-up)
 int rng = rng0; //initially rng is set as 0
 lcd.begin(16, 2); //how many rows and columns are on the display
 lcd.clear(); //clear lcd
 randomSeed(analogRead(1)); //use A1 to generate random numbers

 //Start serial communications and IR reveiver. Play opening song
 lcd.setCursor(1, 0); //set lcd cursor
 lcd.print("Powering on..."); //print to lcd
 digitalWrite(relay_b, LOW); //disengage brakes
 delay(100);
 digitalWrite(relay_a, HIGH); //disengage brakes
 delay(5000); //delay 5 seconds
  lcd.clear(); //clear lcd
 lcd.setCursor(0, 0); //set lcd cursor
 lcd.print("Junkyard Battle"); // print to lcd
 lcd.setCursor(4, 1); //set lcd cursor
 lcd.print("Racer #1"); //print to lcd
 delay (3000); //Wait 3 seconds
 irrecv.enableIRIn(); //Start the IR receiver.
}
/*********************************************************************************
 MAIN PROGRAM LOOP
 *********************************************************************************/
void loop() {
 normaldriving(); //run normal driving mode function
 //check for previously stored rng value (is it not zero?)
 if (rng > 0) {
   rngexecute(); //run rng execute code
   fancydelay(50); //run fancy delay function
 }
 //check for Proximity Receiver Signal Low?
 powerupstate = digitalRead(powerupsense);
 if (powerupstate == LOW) {
   lcd.begin(16, 2); //reset lcd display
   rng = random(1, 11);//generate random number 1-10
   rngexecute();//run rng execute code
```

```
   fancydelay(500); //run fancy delay so that rng sequence is ran only once
  }
  fancydelay(50); //run fancy delay function
  //return; //end of checks, return to beginning of loop
}
/*****************************************************************************
 RANDOM NUMBER GENERATOR FUNCTION
 *****************************************************************************/
void rngexecute() {
  //determine which to execute based on rng value
  if (rng >= 1 && rng <= 2) {
    shield(); //execute shield program
  }
  if (rng >= 3 && rng <= 5) {
    boost();//execute speed bost program
  }
  if (rng >= 6 && rng <= 9) {
    empblaster(); //execute emp blaster program
  }
  if (rng == 10) {
    deathbeam(); //execute death beam program
  }
  else if (rng == 0) {
    loop(); //return to main loop
  }
  fancydelay(50); //run fancy delay function
}
/*****************************************************************************
 SHIELD ACQUISITION SEQUENCE
 *****************************************************************************/
void shield() {
  if (i == 0) {  //if powerup was just acquired run acquisition sequence
    lcd.begin(16, 2); //reset lcd
    lcd.clear(); //clear lcd
    lcd.setCursor(0, 0); //set lcd cursor
    lcd.print("Powerup Acquired"); //print to lcd
    Serial.write(3); //play shield acquired sound effect
    fancydelay(2500); //fancy delay for 2.5 seconds
    lcd.begin(16, 2); //reset lcd
    lcd.clear(); //clear lcd
    lcd.setCursor(0, 0); //set lcd cursor
    lcd.print("Shield Available"); //print to lcd
    fancydelay(4000); //fancy delay for 4 seconds
    i = 1; //set i to 1, so that acuisition sequence won't be run again
  }
  if (i == 1) {
    triggerstate = digitalRead(trigger); //read trigger and set state
```

```
   if (triggerstate == HIGH) {  //if blaster trigger is pressed, run shield sequence
    Serial.write(6); //play blaster gun sound effect
    lcd.begin(16, 2); //reset lcd
    lcd.clear(); //clear lcd
    lcd.setCursor(0, 0); //set lcd cursor
    lcd.print("Shield Activated"); //print to lcd
    shielddelay(1300);
    for ( int x = 15 ; x > 0; x--) { //loop for lcd timer countdown
     lcd.begin(16, 2); //reset lcd
     lcd.clear(); //clear lcd
     lcd.setCursor(0, 0); //set lcd cursor
     lcd.print("Shield Activated"); //print to lcd
     lcd.setCursor(7, 1); //set lcd cursor
     lcd.print(x, DEC); //print seconds remaining on lcd
     shielddelay(900); //shield delay for 1 second
     irrecv.decode(&results);
     irrecv.resume();
    }
    lcd.begin(16, 2); //reset lcd
    lcd.clear(); //clear lcd
    lcd.setCursor(1, 0); //set lcd cursor
    lcd.print("Normal Driving"); //print to lcd
    lcd.setCursor(1, 1); //set lcd cursor
    lcd.print("Mode Restored"); //print to lcd
    fancydelay(1700); //fancy delay for 1 second
    lcd.begin(16, 2); //reset lcd
    lcd.clear(); //clear lcd
    lcd.setCursor(0, 0); //set lcd cursor
    lcd.print("Junkyard Battle"); //print to lcd
    lcd.setCursor(4, 1); //set lcd cursor
    lcd.print("Racer #1"); //print to lcd
    fancydelay(100); //fancy delay for 1/10th second
    i = 0; //set powerup state to used
    rng = 0; //set powerup state to used
    //return;
   }
   else if (triggerstate == LOW) { //if trigger is not pressed return to main loop
    //return;
   }
  }
}
/*********************************************************************************
 SPEED BOOST ACQUISITION SEQUENCE
 *********************************************************************************/
void boost() {
  if (i == 0) {  //if powerup was just acquired run acquisition sequence
   lcd.begin(16, 2); //reset lcd
   lcd.clear(); //clear lcd
```

```
  lcd.setCursor(0, 0); //set lcd cursor
  lcd.print("Powerup Acquired"); //print to lcd
  Serial.write(4); //play boost acquired sound effect
  fancydelay(2500); //fancy delay for 2.5 seconds
  lcd.begin(16, 2); //reset lcd
  lcd.clear(); //clear lcd
  lcd.setCursor(2, 0); //set lcd cursor
  lcd.print("Speed Boost"); //print to lcd
  lcd.setCursor(3, 1); //set lcd cursor
  lcd.print("Available"); //print to lcd
  fancydelay(7000); //fancy delay for 7 seconds
  i = 1;  //set i to 1, so that acuisition sequence won't be run again
 }
 if (i == 1) {
  triggerstate = digitalRead(trigger);
  if (triggerstate == HIGH) { //if blaster trigger is pressed, run boost sequence
    Serial.write(7); //play speed boost sound effect
    lcd.begin(16, 2); //reset lcd
    lcd.clear(); //clear lcd
    lcd.setCursor(0, 0); //set lcd cursor
    lcd.print("Boost Activated"); //print to lcd
    for ( int x = 15 ; x > 0; x--) {
      lcd.begin(16, 2); //reset lcd
      lcd.clear(); //clear lcd
      lcd.setCursor(0, 0); //set lcd cursor
      lcd.print("Boost Activated"); //print to lcd
      lcd.setCursor(7, 1);
      lcd.print(x, DEC);
      boostdelay(900);
      irrecv.decode(&results);
      irrecv.resume();
    }
    lcd.begin(16, 2); //reset lcd
    lcd.clear();
    lcd.setCursor(1, 0);
    lcd.print("Normal Driving");
    lcd.setCursor(1, 1);
    lcd.print("Mode Restored");
    fancydelay(1700);
    lcd.begin(16, 2); //reset lcd
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Junkyard Battle");
    lcd.setCursor(4, 1);
    lcd.print("Racer #1");
    fancydelay(100);
    i = 0;
    rng = 0;
```

```
    //return;
   }
   else if (triggerstate == LOW) {  //if trigger is not pressed return to main loop
    //return;
   }
  }
}
/*****************************************************************************
 EMP BLASTER ACQUISITION SEQUENCE
 *****************************************************************************/
void empblaster() {
 if (i == 0) {  //if powerup was just acquired run acquisition sequence
  lcd.begin(16, 2); //reset lcd
  lcd.clear(); //clear lcd
  lcd.setCursor(0, 0); //set lcd cursor
  lcd.print("Powerup Acquired"); //print to lcd
  Serial.write(2);
  fancydelay(2500);
  lcd.begin(16, 2); //reset lcd
  lcd.clear(); //clear lcd
  lcd.setCursor(0, 0); //set lcd cursor
  lcd.print("Powerup Acquired"); //print to lcd
  lcd.begin(16, 2); //reset lcd
  lcd.clear();
  lcd.setCursor(3, 0);
  lcd.print("EMP Blaster");
  lcd.setCursor(4, 1);
  lcd.print("Available");
  fancydelay(9000);
  i = 1;  //set i to 1, so that acuisition sequence won't be run again
 }
 if (i == 1) {
  triggerstate = digitalRead(trigger);
  if (triggerstate == HIGH) {  //if blaster trigger is pressed, run EMP blaster sequence
    lcd.begin(16, 2); //reset lcd
   lcd.clear();
   lcd.setCursor(4, 0);
   lcd.print("EMP Fired");
   irsend.sendNEC(0x40BE9867, 32); //send EMP blast hex signal
   fancydelay(100);
   irsend.sendNEC(0x40BE9867, 32); //send EMP blast hex signal
   fancydelay(100);
   irsend.sendNEC(0x40BE9867, 32); //send EMP blast hex signal
   Serial.write(5);
   fancydelay(1500);
   lcd.begin(16, 2); //reset lcd
   lcd.clear();
   lcd.setCursor(0, 0);
```

```
    lcd.print("Junkyard Battle");
    lcd.setCursor(4, 1);
    lcd.print("Racer #1");
    i = 0;
    rng = 0;
    irrecv.enableIRIn();
    //return;
   }
   else if (triggerstate == LOW) {  //If trigger is not pressed, send warning IR and return to main loop
    irsend.sendNEC(0x40BE30DD, 32);
    fancydelay(500);
    irrecv.enableIRIn();
    //return;
   }
  }
}
/*********************************************************************************
 DEATH BEAM ACQUISITION SEQUENCE
 *********************************************************************************/
void deathbeam() {
  if (i == 0) {  //if powerup was just acquired run acquisition sequence
   Serial.write(1);
   lcd.begin(16, 2); //reset lcd
   lcd.clear(); //clear lcd
   lcd.setCursor(0, 0); //set lcd cursor
   lcd.print("Powerup Acquired"); //print to lcd
   lcd.begin(16, 2); //reset lcd
   lcd.clear();
   lcd.setCursor(3, 0);
   lcd.print("Death Beam");
   lcd.setCursor(4, 1);
   lcd.print("Available");
   fancydelay(9000);
   i = 1;  //set i to 1, so that acuisition sequence won't be run again
  }
  if (i == 1) {
   triggerstate = digitalRead(trigger);
   if (triggerstate == HIGH) {  //if blaster trigger is pressed, run death beam sequence
    lcd.begin(16, 2); //reset lcd
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Death Beam Fired");
    irsend.sendNEC(0x40BE30CF, 32); //send death beam blast hex signal
    fancydelay(100);
    irsend.sendNEC(0x40BE30CF, 32); //send death beam blast hex signal
    fancydelay(100);
    irsend.sendNEC(0x40BE30CF, 32); //send death beam blast hex signal
    Serial.write(5);
```

```
    fancydelay(1500);
    lcd.begin(16, 2); //reset lcd
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Junkyard Battle");
    lcd.setCursor(4, 1);
    lcd.print("Racer #1");
    i = 0;
    rng = 0;
    irrecv.enableIRIn();
    //return;
   }
   else if (triggerstate == LOW) {  //If trigger is not pressed, send warning IR and return to main loop
    irsend.sendNEC(0x40BE30DD, 32);
    fancydelay(500);
    irrecv.enableIRIn();
    //return;
   }
  }
}
/*********************************************************************************
 WARNING IR RECEIVED SEQUENCE
 *********************************************************************************/
void warning() {
 Serial.write(10); //play warning sound effect
 //lcd.begin(16, 2); //reset lcd
 //lcd.clear(); //clear lcd
 //lcd.setCursor(4, 0);
 //lcd.print("Warning!"); //print to lcd
 //fancydelay(1000); //fancy delay for 1 second
 //lcd.clear(); //clear lcd
 //lcd.print("Junkyard Battle"); //print to lcd
 //lcd.setCursor(4, 1); //set lcd cursor
 //lcd.print("Racer #1"); //print to lcd
}
/*********************************************************************************
 DEATH BEAM HIT SEQUENCE
 *********************************************************************************/
void deathbeamhit() {
 lcd.begin(16, 2); //reset lcd
 lcd.clear(); //clear lcd
 delay(150);
 Serial.write(8); //play death beam hit sound effect
 lcd.print("Death Beam Hit!"); //print to lcd
 throttle.writeMicroseconds(2592); //override gas pedal and set throttle to idle
 delay(100);
 digitalWrite(relay_b, HIGH); //engage brakes
```

```
  delay(100);
  digitalWrite(relay_a, LOW); //engage brakes
  delay(10700); //wait 11 seconds
  digitalWrite(relay_b, LOW); //disengage brakes
  delay(100);
  digitalWrite(relay_a, HIGH); //disengage brakes
  delay(6000); //wait 6 seconds
  lcd.begin(16, 2); //reset lcd
  delay(100);
  lcd.clear(); //clear lcd
  lcd.print("Junkyard Battle"); //print to lcd
  lcd.setCursor(4, 1); //set lcd cursor
  lcd.print("Racer #1"); //print to lcd
  rng = 0;
  i = 0;
}
/*****************************************************************************
 EMP HIT SEQUENCE
 *****************************************************************************/
void emphit() {
  lcd.begin(16, 2); //reset lcd
  lcd.clear(); //clear lcd
  delay(150);
  Serial.write(9); //play EMP hit sound effect
  lcd.setCursor(4, 0);
  lcd.print("EMP Hit!"); //print to lcd
  throttle.writeMicroseconds(2592); //override gas pedal and set throttle to idle
  delay(16000); //wait for 16 seconds
  lcd.begin(16, 2); //reset lcd
  lcd.clear(); //clear lcd
  lcd.print("Junkyard Battle"); //print to lcd
  lcd.setCursor(4, 1); //set lcd cursor
  lcd.print("Racer #1"); //print to lcd
  rng = 0;
  i = 0;
}
/*****************************************************************************
 NORMAL DRIVING MODE STATE
 *****************************************************************************/
void normaldriving() {
  int lasttriggerstate;
  gasval = analogRead(gaspedal);  // read the value of the gas pedal potentiometer (value between 0 and
1023) and define it as gasval
  gasval = map(gasval, 511, 1023, 2200, 1900); // neglect all gasval values below 511 (to account for
spring yield) and rescale vals for servo (up to value between 1219 and 2200).
  throttle.writeMicroseconds(gasval); // sets the throttle position according to the re-scaled gasval
  triggerstate = digitalRead(trigger);
```

```
 if (lasttriggerstate == HIGH && triggerstate == LOW) {
  irrecv.enableIRIn(); // Re-enable IR receiver
 }
 if (irrecv.decode(&results)) {
  if (results.value == 0x40BE30DD) {  //if warning IR signal is received go to warning sequence
   warning();
  }
  if (results.value == 0x40BE30CF) {  //if death beam IR signal is received go to death beam hit sequence
   deathbeamhit();
  }
  if (results.value == 0x40BE9867) {   //if EMP blast IR signal is received go to EMP hit sequence
   emphit();
  }
  irrecv.resume(); // resume receiver
 }
lasttriggerstate = triggerstate;
}
/*****************************************************************************
 SPEED BOOST DRIVING MODE STATE
 *****************************************************************************/
void boostdrive() {
 int gasvalboost = analogRead(gaspedal);  // read the value of the gas pedal potentiometer (value between 0 and 1023) and define it as gasval
 gasvalboost = map(gasvalboost, 511, 1023, 2200, 1219); // neglect all gasval values below 511 (to account for spring yield) and rescale vals for servo (full range for boost).
 throttle.writeMicroseconds(gasvalboost); // sets the throttle position according to the re-scaled gasval
}
/*****************************************************************************
 SHIELD DRIVING MODE STATE
 *****************************************************************************/
void shielddrive() {
 gasval = analogRead(gaspedal);  // read the value of the gas pedal potentiometer (value between 0 and 1023) and define it as gasval
 gasval = map(gasval, 511, 1023, 2200, 1900); // neglect all gasval values below 511 (to account for spring yield) and rescale vals for servo (value between 1219 and 2200).
 throttle.writeMicroseconds(gasval); // sets the throttle position according to the re-scaled gasval
 //gets rid of ability to be shot at.
}

/*****************************************************************************
 FANCY DELAY FUNCTION
 *****************************************************************************/
void fancydelay(int time) {
 for ( int x = (time) ; x > 0; x--) { //break requested delay time into 1 millisecond pieces
  normaldriving(); //run normal driving mode throughout requested delay
  delay(1);
 }
```

```
}
/*****************************************************************************
 BOOST DELAY FUNCTION
 *****************************************************************************/
void boostdelay(int time) {
  for ( int x = (time) ; x > 0; x--) {  //break requested delay time into 1 millisecond pieces
    boostdrive(); //run boost driving mode throughout requested delay
    delay(1);
  }
}
/*****************************************************************************
 SHIELD DELAY FUNCTION
 *****************************************************************************/
void shielddelay(int time) {
  for ( int x = (time) ; x > 0; x--) {  //break requested delay time into 1 millisecond pieces
    shielddrive(); //run shield driving mode throughout requested delay.
    delay(1);
  }
}
```

Appendix E: PIC Code

```
' receiver.bas
' Code for the receiver PIC in an example illustrating A/D conversion,
' hand shaking, serial communication, and LCD ouput
' Define non-default configuration settings (from the PIC16F88 code template)
#CONFIG
 __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#endconfig
' Set the internal oscillator frequency to 8 MHz
define OSC 8
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1
' Turn off the A/D converters
ANSEL = 0
' Define variables and constants
sound0 Var PORTA.0
sound1 Var PORTA.1
sound2 Var PORTA.2
sound3 Var PORTA.3
sound4 Var PORTA.4
sound5 Var PORTA.6
sound6 Var PORTA.7
sound7 Var PORTB.0
sound8 Var PORTB.1
sound9 Var PORTB.3
sound10 Var PORTB.4
hand_shake Var PORTB.5 ' handshake line on pin RB6
serial Var PORTB.2 ' serial communication through pin RB0
pot_value Var Byte ' POT value received from sender PIC
baud_rate Con 2 ' 9600 baud-rate mode for serial communication
' Blink the LED three times to indicate the PIC is running
'Gosub Blink1 : Gosub Blink1 : Gosub Blink1
' Wait 1/2 sec for the LCD to power up, and clear the LCD
'Pause 500
'Initialize I/O pins
TRISA = %00000000       'all PORTA pins initialized as outputs
                        ' (although only pins 0,1, and 2 are used)
TRISB = %00000100       'PORTB.2,5 pins are initialized as inputs
                        ' (RB4: reset, RB5: increment, RB6: decrement)
                        'all other PORTB pins initialized as outputs

' Make sure the handshake line is off initially
Low hand_shake
low sound0
high sound1
high sound2
```

```
high sound3
high sound4
high sound5
high sound6
high sound7
high sound8
high sound9
high sound10
 pause 5000  'wait 5 seconds
' Main program loop
start:
 ' Handshake with the sender PIC and receive the POT value serially
 high sound0
 high sound1
 high sound2
 high sound3
 high sound4
 high sound5
 high sound6
 high sound7
 high sound8
 high sound9
 high sound10
 High hand_shake ' signals the sender PIC to send
 Serin serial, baud_rate, pot_value ' wait for and receive POT value
 Low hand_shake
 ' Display the received POT value on the LCD and blink the LED
 IF pot_value = 0 Then Gosub Play0
 IF pot_value = 1 Then Gosub Play1
 if pot_value = 2 then Gosub Play2
 if pot_value = 3 then Gosub Play3
 IF pot_value = 4 Then Gosub Play4
 IF pot_value = 5 Then Gosub Play5
 IF pot_value = 6 Then Gosub Play6
 IF pot_value = 7 Then Gosub Play7
 IF pot_value = 8 Then Gosub Play8
 IF pot_value = 9 Then Gosub Play9
 IF pot_value = 10 Then Gosub Play10
 Goto start
End ' end of main program (not required since never reached)

Play0:
 low sound0 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
Play1:
 low sound1 ' turn on the LED
 Pause 1000 ' wait 1/4 second
```

```
Return
Play2:
 low sound2 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
Play3:
 low sound3 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
Play4:
 low sound4 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
Play5:
 low sound5 ' turn on the LED
 Pause 500 ' wait 1/4 second
Return
Play6:
 low sound6 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return

Play7:
 low sound7 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
Play8:
 low sound8 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
Play9:
 low sound9 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
Play10:
 low sound10 ' turn on the LED
 Pause 1000 ' wait 1/4 second
Return
```