
POWER MOBILITY FOR YOUNG STUDENTS

Designed for The Kevin G. Langan School at The Center for Disability Services

Lam Son Ba Ngo, ECBE Department
ECE-499

Professor Cherrice Traver, Project Supervisor

Wednesday 21st March, 2018

1 Summary

Prescribed wheel-chairs in the United States are awfully expensive, which makes getting access to one for training rather difficult for most disabled children and students. The goal of this senior capstone project is to adapt a Fisher-Price mobility toy called Wild thing[®] and construct a controller for the device, which can easily accept inputs from a variable number of standard joysticks, buttons, switches or other sensor inputs. However, due to time constraints, in the first iteration of the project, it will only accept inputs from up to 4 different push-button ability switches. To be considered successful, the controller for the Wild thing[®] needs to be able to control the motors for four directions of movements: forward, backward, left, and right, can be reconfigured on the go, and be able to navigate safely without too much help from supervisors. In addition, it must be able to be adjusted to specific needs of students, to work outdoor, and must stay within the budget of \$500. These requirements are met by using a micro-controller to take in inputs, process them, and drive the motors using Spark motor controllers (replacing the built-in controllers), an LCD touch screen to enable reconfigurability, and two arrays of sensors to ensure safety when driving.

The controller will then be integrated into a modified Wild Thing[®] body with flexible seating and support options to develop a final working prototype. If successful and meeting all design requirements, it can work as a low-cost alternative to a traditional prescribed wheel-chair for two pre-determined students at the Kevin G. Langan School at the Center for Disability Services by the end of Winter term 2018. The working device can also serve as a basis for future senior capstone projects.

Contents

1	Summary	2
2	Introduction	7
3	Background	8
3.1	Related Work	8
3.2	Potential Project Impacts	10
4	Design Requirements	10
4.1	Performance	11
4.2	Compatibility	11
4.3	Debugging	11
4.4	Safety	11
4.5	Energy	11
4.6	Functional Decomposition	12
5	Design Alternatives	13
5.1	Processing unit of the device	13
5.2	Turn on the device	13
5.3	Administrative the device	13
5.4	Administrate the device (for supervisors)	14
5.5	Configure the control interface	14
5.6	Remote Control The device	14
5.7	Max Speed Adjustment	15
5.8	Distance sensing	15
5.9	Safe mode	15
5.10	Display Battery and Speed Information	15
5.11	Emergency stop (kill switch)	15
5.12	Turn on Safe mode	15
6	Preliminary Proposed Design	16
6.1	Micro-controller Design	16
6.2	Motor Controller	18
6.3	Ability Switch input	19
6.4	Display Battery and Speed Information	20
6.5	Remote controller	21
6.6	Safe mode	21
6.6.1	Distance sensing	21
6.6.2	Piezo Buzzer	22
6.6.3	Safe mode Implementation	22
6.7	LCD touch screen - User Inputs	23

7	Final Design and Implementation	24
7.1	Controlling the Motors	25
7.1.1	Acquiring User Configurations via a Graphical User Interface on a Capacitive Touch Screen	25
7.1.2	Input Configuration	28
7.1.3	Drive motors	30
7.2	Ensuring Safety	33
7.2.1	Distance sensing and Obstacle Avoidance	33
7.2.2	Remote Controller and Remote Kill Switch	35
7.2.3	Fail-safe Default	35
7.3	Display Statistical Information	37
7.3.1	Battery Voltage Measurements	37
7.3.2	Max Speed and Current Speed Notifier	38
7.4	Overall Implementation of the Project	40
8	Overall System Performance	42
8.1	Preliminary Design Performance Estimates	42
8.2	Final Implementation Actual Performance	43
8.3	Development and Testing Process	44
9	Potential for Improvements and Future Work	45
10	Production Schedule	47
10.1	Original Production Schedule for Winter Break 2017 and Winter Term 2018	47
10.1.1	Original Winter Break Schedule (6 weeks)	47
10.1.2	Schedule for Winter term	47
10.2	Actual Project Schedule Analysis	48
10.3	Spring Term Schedule	48
11	Cost Analysis	49
12	User's Manual	50
12.1	Assembly and Program Setup	51
12.2	Setups and Operation	52
12.3	Troubleshooting and Adjustments	52
12.4	Storage and Maintenance	53
13	Conclusion	53
13.1	Discussion	53
13.2	Problem	53
13.3	Solutions and Implementation	53
13.4	Results	54
14	Acknowledgement	55
	Appendices	55

A Full Software Codes	55
B Touch Screen Code	72
C LED PWM Code	80
D Student Research Grant	83
E Departmental Presentation	87

List of Tables

1	Detailed Functional Decomposition of the Project.	12
2	Breakout of potential micro-controller candidates.	17
3	Detailed breakout of potential LCD Touch Screen Candidates.	23
4	Detailed Pin out.	41
5	Detailed Winter Term 2018 Schedule.	49
6	Total cost breakdown.	50

List of Figures

1	Fisher-Price's Wild Thing [®] [2]	7
2	RJCooper's Cooper Car[3]	8
3	Medical Engineering Resource Unit's Bugzi [®] [4]	9
4	High-level design block diagram	10
5	System's block diagram	16
6	Arduino Mega [9].	18
7	Spark Motor Controller Pinout [10].	18
8	Push button ability switch [11].	19
9	Battery Tester Circuit.	20
10	Adafruit's NeoPixel Ring [12]	20
11	TSOP 38238 IR receiver.[13]	21
12	HC-SR04 ultrasonic sensor circuit.	21
13	Sharp GP2Y0A21YK IR sensor circuit.	22
14	TFT Touch Shield [14].	24
15	Final implementation block diagram.	25
16	User Interface first screen.	26
17	User Interface second screen.	26
18	User Interface third screen.	27
19	User Interface fourth screen.	27
20	Arduino Mega's handling of input 4 2 Off FBLR 0	28
21	Arduino Mega's handling of input 3 5 On RLF 0.	28
22	Schurter's audio socket.	29
23	Schurter's audio socket internal diagram.	29
24	Ability Switch layout.	30

25	Close up of the Motor Controller.	31
26	The device's power system.	31
28	Motor controller's pulse ranges [10].	31
27	SPARK motor controller's schematic [10].	32
29	Code for moving at 2 miles per hour speed profile	33
30	Sensor layout on the back of the device.	34
31	Flowchart to represent the implementation of proximity sensing.	35
32	Flow chart representing how the remote controller is implemented in software.	36
33	Flowchart representing the implementation of fail-safe default in software.	37
34	Battery measurement voltage (repeated from page 19).	38
35	Three stages of battery level shown on a 24 LED NeoPixel Ring.	39
36	Three different speed profiles shown on a 24 LED NeoPixel Ring.	39
37	Tail light NeoPixel's LEDs incrementally be lit up when device is accelerating forward.	39
38	NeoPixel rings' placements on the device.	40
39	First Iteration of the Project.	40
40	Flowchart representing the overall software implementation of the project.	42
41	SPARK motor controller's LED status code.	44
42	Port Setup for Arduino IDE	52

2 Introduction

In the United States, in order to be granted a fully customized powered wheelchair, a person needs to demonstrate to the committee that she is able to operate the device competently and safely [1]. It is a fair rule, but the problem lies in the fact that buying a wheelchair for practice is completely out of the question for most people, especially young disabled students. Base models of modern wheelchair, which are used for mobility assessment, are awfully expensive, costing more than \$10000 per unit, that is before adding any customizations. To counter this, disability services around the states try to borrow and lend wheelchairs to each other. However, this is only an impermanent solution, as the time frame in which a center can use a wheelchair is often only one week. In addition, the devices come without any customizations, and the counselors and physical trainers have to perform the customization process from the ground up every time they receive a new wheelchair.

Thus, my senior capstone project is to adapt a Fisher-Price toy called the Wild Thing[®], so that it can serve as a power mobility training and assessment tool for younger students, while being low-cost and easy to use. Power mobility means the use of powered wheelchairs and ride-on toys, which run on electricity and provide efficient and autonomous mobility and body support for individuals with limited ability to walk. While my device is not supposed to be a practical replacement for a prescribed wheelchair, it can be utilized as one for young children to learn how to drive. This project is an interdisciplinary one, in which I, a Computer Engineering major, will handle adapting the control interface of the Wild Thing, and Joseph Carruso, a Mechanical Engineering major, will create and construct flexible seating customizations. The entire project is developed and designed to be tested and used at the Kevin G. Langan School At the Center For Disability Services, under the strict supervision of Mr. Jim Luther and Mrs. Laura A Shemo.



Figure 1: Fisher-Price's Wild Thing[®] [2]

The Wild Thing[®] toy was chosen as it is small in size (but still can support up to 44kg), is budget friendly, and has two independently powered motors. In addition, it was built specifically for outdoor performance, such as on pavement and on grass. Its large, rugged tires and forward speeds of up to 5 mph are able to handle most of the surfaces, and there is a built-in stability sensor to help prevent tipping.

However, due to the limited time-frame for a senior project, I have decided to lower the scale of the project. Instead of being able to interface all kinds of different inputs, for now, the device will just interface up to four different push-button ability switches. Also, the mapping of the inputs and their functionalities are tailored to two specific children at the Kevin G. Langan School, who have been carefully chosen to test the project. I will include a detailed manual on how to replicate my project, and students from next years can continue to develop it into a fully functional model and replacement for a prescribed wheelchair.

3 Background

3.1 Related Work

This is not the first time a project like this has been carried out in order to solve this unfortunate dilemma. The very first attempt was the Cooper Car[3], which was based on the toy BOSS[®] by Hedstrom and was sold at \$2000. It was operable by 1 to 4 switches, 1 for each direction, or a joystick.



Figure 2: RJCooper's Cooper Car[3]

However, the device was not able to steer normally, as a direction switch caused the device to spin on its axis. The car was only able to support 22.6kg, which was often not good enough. It was also too bulky and could only work on smooth surfaces indoors. Unfortunately, in 1999, a Hedstrom factory burned

down, and BOSS[®] was discontinued and the Cooper Car project came to an end. Over time, many different prototypes from other companies were introduced, with better specification and performance, but were also much more expensive; Medical Engineering Resource Unit (MERU)'s Bugzi[®] costs about \$5000 and Otto Bock's Skippi[®] costs \$10000. Standing out is MERU's bugzi[®], an award-winning powered indoor wheelchair for children aged one to six[4]. Each Bugzi[®] is tailored and adapted to the individual child's needs, with the capacity to adjust seating and controls as they grow. Compared to the Cooper Car, Bugzi[®] is much more compact, can support more weight and is more flexible in general. In addition, as of current, thanks to the generosity of donors from the charity the QEF Mobility Centre based in Carshalton, Surrey, the William Merritt Disabled Living Centre, and the Cornwall Mobility Centre, MERU is loaning out a number of Bugzis free of charge for children having legal residency in the UK for a minimum of 6 months.



Figure 3: Medical Engineering Resource Unit's Bugzi[®][4]

As commercial powered wheel-chairs are still too expensive, another alternative is Go Baby Go, a non-profit organization, founded by Professor Cole Galloway from the University of Delaware. He invented assistive devices for children with motor impairments, usually adapting mobile toys like the Wild Thing[®]. Soon after it was founded, the organization's popularity quickly spread around the United States and many colleges and hobbyists have started their own program trying to make their own difference. Go Baby Go vehicles are often DIY (do it yourself) and cost less than \$1000, but after reading their reports, their systems, while beautifully crafted, felt somewhat lacking in complexity, and thus could not offer help to a wider range of disabled students.

3.2 Potential Project Impacts

As mentioned in the Introduction section, power mobility means the use of powered wheelchairs and ride-on toys, which run on electricity and provide efficient and autonomous mobility and body support for individuals with limited ability to walk. The goals of these vehicles are to allow their users to move freely within their home and community, to maintain their safety and to conserve energy. Power mobility can enhance quality of life by enabling occupation, improving self-esteem and facilitating social interaction.

For young children, motor skills develop rapidly during the first three years and provide a means for exploration of the environment and for socialization [5]. Thus, for children with severe motor and muscle impairment, exploration and socialization often are limited because of the difficulty these children have on their own. Studies have shown that for these children, power mobility can promote independence and prevent functional limitation and disabilities that the children might encounter [6]. In addition, the ability to move independently has positive influence on their self-awareness, emotional attachment, spatial orientation, fear of heights, and visual/vestibular integration [7] as well as personality traits like motivation and initiation [8].

The fact that this model of a prescribed-wheel chair can be made within a budget of \$1000, for both the mechanical engineering and computer engineering aspects, will lower the entry point for power mobility for most people as even the wheelchairs mentioned in the background sections cost about \$5000 to \$10000. In the future iterations of the project, students will continue to come in and make changes to the project, to make sure that device will have more features, and will behave more and more like a prescribed wheel-chair. Thus, this will introduce power mobility to people who otherwise would not have the chance to, and while it may not be as good as a powered-wheel chair, this is the first step in proving competency to be granted a wheel-chair by the government.

4 Design Requirements

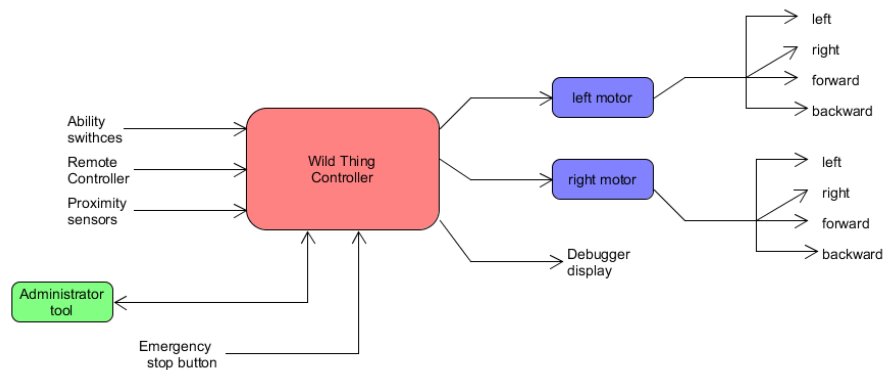


Figure 4: High-level design block diagram

The figure above illustrates the design requirements and constraints that define my project. Following the figure is a set of design specifications which were developed after a conversation with Mr. Luther and Mrs. Shemo from Langan Center for Disability services. All of the below specifications were carefully detailed as to ensure that the device will be safe and easy to use for children.

4.1 Performance

- The Wild thing[®] controller needs to produce individual outputs for each motor, as they are independently powered.
- The vehicle must be able to move forward, backward, steer left and steer right, and brake. Speed acceleration and deceleration need to be smooth to prevent jerky movement.
- The Wild thing[®] controller needs to have a usable administrator tool, with which users can modify the inputs being used, how the inputs control the two motors, the type of inputs, and the maximum speed of the vehicle.

4.2 Compatibility

- The Wild thing[®] Controller must be able to interface a variety of standard ability switches, whose signals are transferred through a 3.5mm jack plug.
- The controller needs to be adjustable to specific user needs.

4.3 Debugging

- The controller needs to have a debugger display available (on the vehicle itself) to the administrator users, which will display important information about the current system, such as: Battery level, current speed, current signals outputted from the switches.

4.4 Safety

- The controller will use a series of proximity sensors to avoid walls and obstacles. This will be activated in a mode called safety mode, which can be turned on or off via the administrator tool.
- An emergency stop button can stop the vehicle once pressed.
- Failure in any subsystems must result in mobility stoppage.
- A remote controller, which can act as a kill switch, must be available for supervisor. The controller should be able to control the device just like a normal set of ability switches.

4.5 Energy

- The system should provide enough power for at least 1.5 hour of operation, both indoor and outdoor.
- There should be a notification when the batteries run low, and the device would be slowly halted.

4.6 Functional Decomposition

Table 1 below details what the system needs to be able to accomplish based on the set of design specifications above.

Subfunctions	Operations
Turn the device on	A module that can turn the device on and off easily.
Administrate the device	A way for the users and their supervisors to reconfigure the device. Including the ability to change the functions of each inputs, the number of inputs, their types and the maximum speed.
Maximum speed adjustment	A way to ensure the maximum speed the device can reach.
Record Speed and Battery Information	A way to know how fast the device is going, and how much capacity the battery has left.
Display Speed and Battery Information	A way to display visually to the users and their supervisors the speed and battery information.
Distance sensing	A way to detect obstacles in the device's path.
Remote control the device	A module that can control the device remotely.
Turn on safe mode	A module that can turn the safe mode on and off.
Safe mode	A mode in which the device can avoid and halts if it is too close to an obstacle
Emergency stop (kill switch)	A way to kill the operations of the device.

Table 1: Detailed Functional Decomposition of the Project.

5 Design Alternatives

5.1 Processing unit of the device

Since the Wild Thing[®] controller needs to be able to take in inputs, and at the same time use these inputs to drive the motors, and be reconfigured on the go, it is best that we construct the device around a micro-controller.

There are many choices to choose from with a micro-controller: A Raspberry Pi, an Arduino micro-controller, or a PIC microcontroller. These hardware stands out as they are low-cost and commercially available. Each of them has their own disadvantages. The Raspberry Pi has a lot of memory capacity and processing power, due to it being a full computer, while the other two do not. However, it only has digital general purpose inputs and outputs, and has limited support for interfacing sensors and external parts. The PIC chip, on the other hand, is cheap, small, has enough processing power, but is hard to debug, and also has limited libraries for interfacing external devices. With these two micro-controllers, most of the time, if we want to connect it to something, we would need to write our own libraries for it to work. This leaves us with an Arduino micro-controller. It is compact, the same-sized as a Raspberry Pi, affordable, and has extensive libraries and tutorials for sensors and external devices, both from its company and also from its ever-rising community.

5.2 Turn on the device

1. Solution: Using a button attached to the body of the toy to turn on the device.
2. Alternative solution: Using the weight sensor presented on the Wild Thing[®].

The Wild Thing[®] toy itself is always-on, which is implemented by a weight sensor. The joysticks and motors would not work until a sufficient weight has been applied to the seating area. This is a nice and futuristic approach to ride-on toys, but it sacrifices energy and battery time. Since the longevity of the operation of the device is required by our customers, it is best to implement a simple on-off button, connected directly to the micro-controller, and mounted on the body of the Wild Thing[®] to turn the device on or off.

5.3 Administrative the device

1. Solution: Using a laptop with Arduino IDE connected directly to the micro-controller.
2. Alternative solution: Do the same thing, but wirelessly with a bluetooth adapter.

In the early versions of the project, to reduce its complexities, I am not going to build an administrative tool for the customers until I have finished with building the whole thing. Instead, I am going to use a laptop with the Arduino IDE running and connect it directly to the micro-controller and perform the changes that an administrative tool should be able to do by re-coding. Doing this wirelessly is an option, but it is not worth the time setting and perfecting the connection.

5.4 Administrate the device (for supervisors)

Since the customers' only real need in modifying the device is to change the mapping of the inputs, the best way to implement it would be to design a simple user interface, and have it either on a touch screen that is attached to the Wild Thing, or an application on a computer that is connected with the device wirelessly.

The application on a computer is simpler to carry out, since it only needs a bluetooth adapter and a graphical JAVA user interface. However, doing it this way would require the costumers to bring their laptops all the time, which is clunky. Thus, it makes sense to try to implement a simple graphical user interface on a touch screen, and have it attached to the body of the toy. This, on the other hand, increases the complexity of the project, as I, myself as a student, have never implemented a touch screen with an Aduino, but the hassle is worth it with the increase of usability and customer's satisfaction.

5.5 Configure the control interface

One of the most important aspects of this project is the ability to reconfigure the Wild Thing controller on the go. The changes would include: choosing the types of inputs being connected, either digital or analog, the number of inputs, the mapping of the inputs to the basic directions: forward, backward, left or right, and the speed limit.

At the early stages of the project, this is done by reprogramming the micro-controller using an IDE on a computer like in section 5.4. After the first round of testings have been done and all operations have been made sure to work, a touch screen mounted on the device will be utilized to reconfigure the Wild Thing Controller, like in section 5.5.

5.6 Remote Control The device

1. Solution: Using an Infrared receiver connected to the micro-controller, and an Infrared remote controller to control it.
2. Alternative solution: Using a bluetooth adapter on the micro-controller, and use a laptop to control it.
3. Alternative solution: Using radio signal with a remote, similar to the system used in cars.
4. Alternative solution: Using the Wireless SD Shield, which allows an Arduino board to communicate wirelessly using a wireless module. It is based on the Xbee modules from Digi.

There are various choices when it comes to remote controlling the device. However, in terms of priority, this is not the most important, as it is not essential in the core operations of the device. Thus, to reduce complexity, an IR remote and an IR receiver are the most straightforward. The remote control's operations are simple: turn on safe mode, turn off safe mode, turn on/off the device, and control the motors. It is not worth the hassle in the early stages to implement a more complex solution. While using IR outdoor is not optimal, in the first round of testing and demoing, the device will be mostly operated indoor. If the project is successful and the first implementation of the remote controller is good enough and time allows, I would proceed to make a remote controller using the Arduino Wireless SD Shield. This module is superior to the IR remote controller since it can communicate up to 100 feet indoors or 300 feet outdoors with line of sight.

5.7 Max Speed Adjustment

1. Solution: Use a potentiometer to change the value of the max speed variable in the program, as the motors of the Wild Thing can be controlled using Pulse Width Modulation.

5.8 Distance sensing

To implement distance sensing capability on the device, two arrays of IR sensors and ultrasonic sensors would be used, one in the front and one in the back. Both an IR sensor and an ultrasonic sensor are used to increase the range, and the accuracy of the sensing information (from 1cm to 100cm).

5.9 Safe mode

As required by the customers, the device will need to be able to avoid obstacles in a mode called the Safe mode. This will be implemented utilizing the distance sensing capabilities mentioned above. When the device is in close proximity to an obstacle, at around 80 cm, the device will use a Piezo buzzer to make a loud sound to alert the users and the supervisors. When the device is too close to an obstacle, at 30 cm, the device will automatically halt if the user has not decided to do though. The 30 cm distance is used to ensure smooth halting and prevent injuries.

5.10 Display Battery and Speed Information

1. Solution: Use 2 LEDs string to display the information.
2. Alternative solution: Use an LCD display to display the numerical version of the information.

Since we have already installed an LCD touch screen on the device to be used as the administrative tool, install another LCD to display the device's information seems impractical. Thus, we decided to go with 2 LED strings with multiple LEDs to display the two information. The number of LEDs being lit up in the array directly corresponds to how fast the device is going, or how much capacity the device's battery still has left. This will be more straight-forward to carry out, and would give a better visual representation of the battery and speed information, as even from far away, the supervisors can still see the two indicators.

5.11 Emergency stop (kill switch)

1. Solution: Install a button on the device and a button on the remote controller.

5.12 Turn on Safe mode

1. Solution: Install a button on the device and a button on the remote controller.

6 Preliminary Proposed Design

The figure below is the detailed block diagram of the project. It includes parts, their operations and how they are connected together.

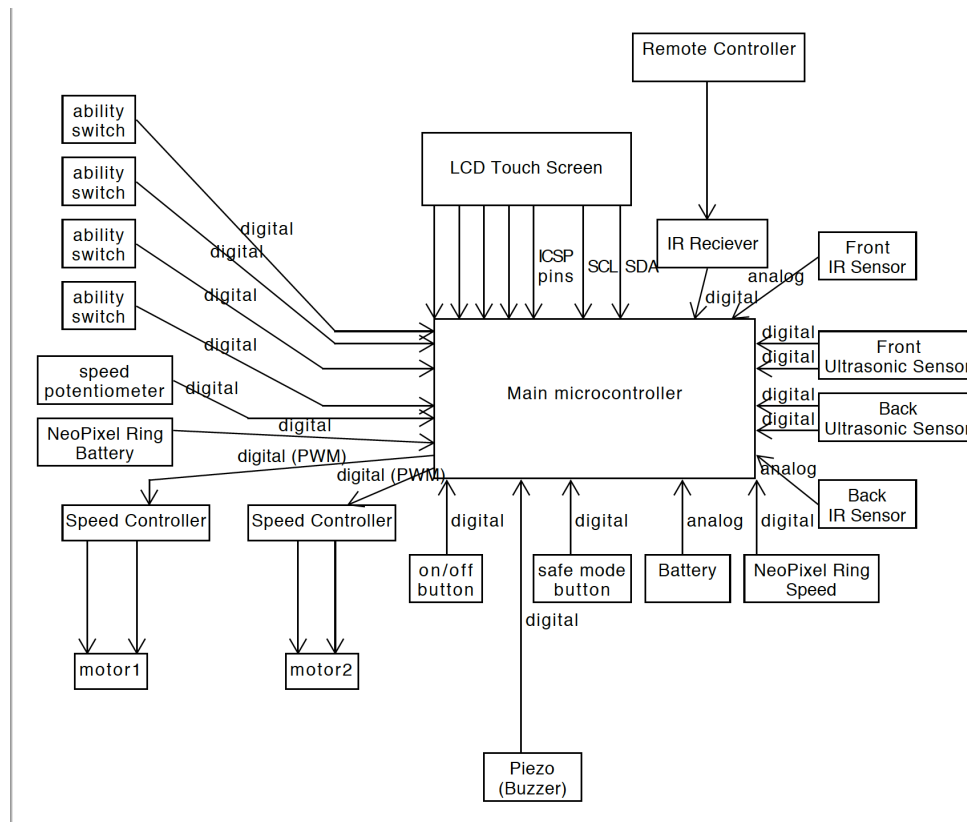


Figure 5: System's block diagram

6.1 Micro-controller Design

The heart of this project is a micro-controller, which needs to fit under the seating area of the Wild Thing toy, in a 16 cm by 8 cm area. Here are the micro-controller design constraints:

- It needs to fit in a 16 cm x8 cm space.
- It has to have at least 14 digital general purpose input, 6 of which are PWM inputs and 5 analog input.
- It has to have at least 5 interrupt pins.
- It has to have a large memory capacity to store code.
- It needs to support basic communications: SPI, I2C, and Serial.
- It needs to be easy to program and debug.

	Adafruit Trinket Pro	Arduino Micro	Raspberry Pi zero	Arduino Zero	Arduino Mega (Chosen)
Pros	Small and really cheap.	Compact.	Compact and a Full computer.	None	None
	Operates at 5V.	Has both 3.3V and 5V variants.	Operates at 3.3V.	Operating Voltage 3.3V.	Operates at 5V.
	18 GPIOs (6 as analog) and 2 additional analog.	20 digital GPIO pins, in which 12 can be used as analog and 7 can be used as PWM.	28 digital GPIO pins.	6 analog pins and total 20 GPIOs.	54 digital GPIO pins. 16 analog GPIOs.
	2 external interrupts	External Interrupts on 4 pins	None	All support external interrupts	6 pins support external interrupt.
	28K of flas and 2K of Ram	32kb of flash and 2.5kb SRAM. Has EEPROM. 16 Mhz Cyrstal oscillator	512Mb Ram and supports SD cards	256Kb Flash memory and 32Kb of SRAM	256Kb Flash memory, 4Kb of SRAM and 4Kb of EEPROM
	Supports I2C, SPI	Support I2C and SPI	Support I2C, SPI, Serial	Supports I2C, SPI, Serial	Support I2C, SPI, Serial
	Dimension: 3.81cm by 1.778cm.	Dimensions: 4.8cm by 1.77cm	Dimensions: 6.5cm by 3cm	Dimensions: 6.86cm by 5.334cm	Dimensions: 10cm by 5.3cm
	Can be used with both Arduino IDE and Python	None	Cheap and support the Raspberry Pi touch screen	Powerful and has embedded debugger	None
Cons	No support for Serial.	Fragile and easy to break	Little library support	Only 14 digital pins	Longer than most of the other controllers
	Only 2 external Interrupts	Has the least memory.	Many unnecessary ports.	None	None
	None	None	All GPIOs are digital (needs ADCs)	None	None

Table 2: Breakout of potential micro-controller candidates.

The table above represents the different choices of micro-controllers short-listed for this project. Each micro-controller is the best one in its size category. The categories were used to choose the micro-controllers are: cost, operating voltage, number of GPIOs, external interrupt capability, memory capacity, supported communications, and dimensions.

Originally, a smaller variant of an Arduino, an Arduino Micro, was planned to be used. However, it does not have enough GPIOs and external interrupts to support the project. Also, upon reconsideration of the dimensions of the Wild Thing[®], small size is not completely necessary, and with careful wiring

and placement, a full-size Arduino is also suitable.

Thus, we resort to an Arduino Mega, which has 54 GPIOs and 6 external interrupts. This has way more ports than what the project needs, but it facilitates adding more features in the future if the customers demand it. It is a newer version of ATmega1280, one of the faster mobile CPUs. It has 128Kb of flash memory to store codes, 4kb of which is used for bootloader. In addition, it has 4kb of electrically erasable programmable read-only memory, which will be useful along the way if we want to implement some kinds of identifications with the project. It supports all of the required communication systems.

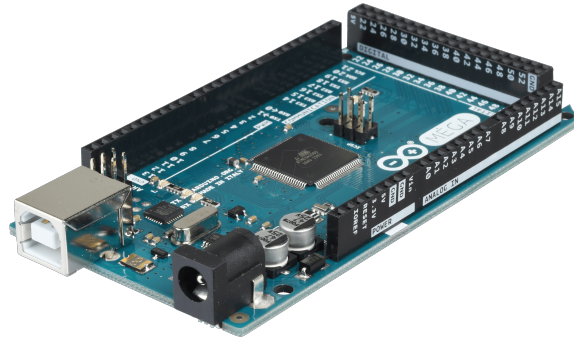


Figure 6: Arduino Mega [9].

However, one downside is that the Arduino Mega is one of the bigger Arduinos, with its dimensions being 10cm by 5.3cm. It will require careful and smart wiring to make sure that it fits in the designated space.

6.2 Motor Controller

The Wild Thing itself has its own motor-controllers, but it does not work well with Arduino micro-controllers, which was chosen for the project, and has poor documentation online. Hence, to reduce the complexity of the project, it was decided that we would use a Spark motor controller to replace the built-in ones. One advantage the Spark motor controllers have is that it has been proven to work with Arduino and the Wild Thing [cite].

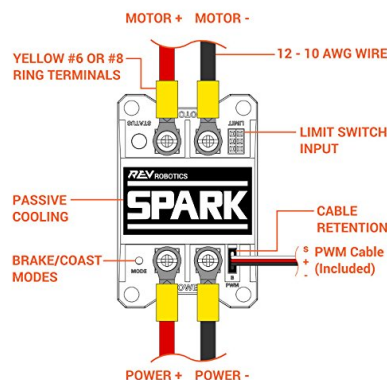


Figure 7: Spark Motor Controller Pinout [10].

From figure 9, a particular Spark motor controller has PWM inputs, Power inputs, and Motor output. For Pulse Width Modulation, the “S” pin is wired to a PWM port on the Arduino and the “-” pin is connected to the Mega’s ground. We do not need to connect the “+” pin, as it is not connected to anything internally. The motor “+” and “-” are connected directly to the two wires coming from the motors. Power “+” and power “-” are connected to the battery. The motor controller can operate between 5.5 V and 24V, and the ideal amount is 12V, which can be supplied using the Wild Thing’s built-in rechargeable battery.

6.3 Ability Switch input

Different from the initial pitch, which called for the ability to interface all kinds of inputs, we have decided to reduce the scale of the project, and tailor it to work specifically with two students from Kevin G. Langan School. After meeting with their physical trainer, Mrs. Laura A. Shemo, our inputs are chosen to be four push button switches.

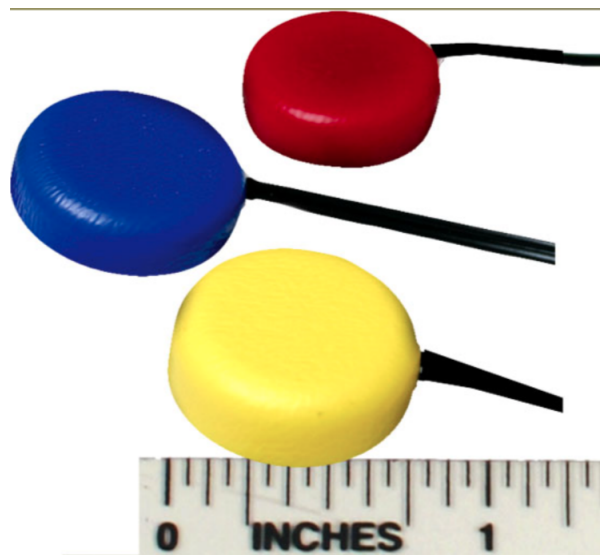


Figure 8: Push button ability switch [11].

Ability switches are designed for people with physical abilities, offering them an alternative means to interact with computers, speech generating devices, appliance controllers, and so on. It has a 3.5mm jack, and in order to use one with a micro-controller, we need to use an adapter, which makes it work like a 2-pin push button.

To connect it to the Mega, one side of the button is connected to Ground, and the other side is connected to a digital Arduino pin, which needs to be configured as a digital input with pull-up enabled. If the button is push-to-make, a digitalRead from that pin will return LOW when the button is pressed, and HIGH when it is not pressed. If the pushbutton is press-to-break, this will be reversed. It can also be used to generate pulse width modulation, which in turn can control our motor controllers.

6.4 Display Battery and Speed Information

When we are using a battery powered Arduino with other parts such as motor controllers and LCD touchscreen, it is necessary that we check the battery voltage to see if it needs to be charged or replaced. An Arduino Mega needs 5 V power to run on, thus our 12 V battery is sufficient.

To record the battery level, we need a battery tester circuit, which has a common ground between the controller and the battery and an analog pin. Since Arduino pins support only a maximum of 5 V, we need a voltage divider to reduce the battery from 12V to under 5 V. This circuit, however, only measures the voltage level. For a more elaborate information on the condition of the battery, current and power dissipation also need to be considered. An example circuit can be seen below.

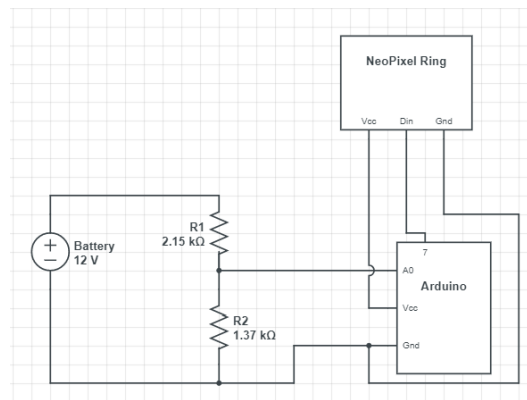


Figure 9: Battery Tester Circuit.

The Wild Thing has a built-in speed indicator, which we can use to see how fast the device is moving. To display this two information (speed and battery) to the users, two NeoPixel Ring will be used. NeoPixel is an Adafruit product, and it is essentially an array of LEDs. The ring's outputs are chained, i.e, the output of one pin is connected to the input pin of another. Thus, a micro-controller only needs one digital pin to control the ring. This is really advantageous as we can reserve pins for other parts.



Figure 10: Adafruit's NeoPixel Ring [12]

When the battery is full or the device is moving at maximum speed, all LEDs in the ring will be lit up. As the battery is slowly discharging and the device is moving slower, the LEDs will gradually be

turned off to signify the users.

6.5 Remote controller

A TSOP 38238 IR receiver diode was chosen for this project. It has low power consumption, and is straightforward to use. A pin diode and a preamplifier are assembled on a lead frame while the epoxy package acts as an IR filter. The demodulated output signals can be directly decoded by an Arduino micro-controller, which makes the TSOP 38238 compatible with all common IR remote control data formats.



Figure 11: TSOP 38238 IR receiver.[13]

6.6 Safe mode

6.6.1 Distance sensing

As mentioned in the Design Alternative section, for distance sensing, two arrays of ultrasonic and IR sensors are going to be used, one in the front and one in the back.

For the ultrasonic sensor, we decided to go with the HC-SR04, the most commonly used ultrasonic sensor for the Arduino. It works in the range from 2 cm to 400 cm. It is cost-effective (\$4.00) and easy to use. Its operating voltage is 5V, which is suitable for the Arduino Mega, and the operating current is 15 mA. It needs 4 pins: power, ground, trigger pulse input (digital) and echo pulse output (digital).

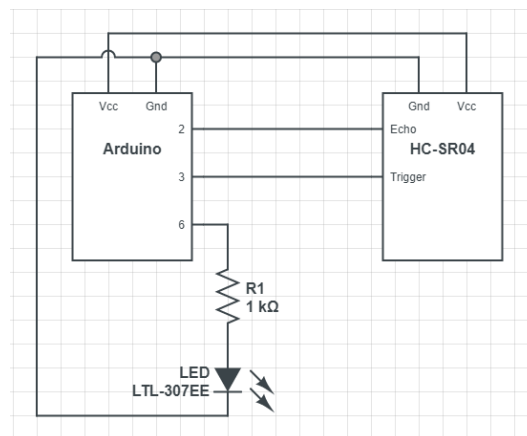


Figure 12: HC-SR04 ultrasonic sensor circuit.

For the Infrared sensor, a Sharp GP2Y0A21YK is used, with the working range from 10 cm to 80 cm. It is more expensive than its ultrasonic counterpart at \$14.95, but is more accurate. Its operating voltage is 5V, which is suitable. It needs three pins: Power, ground, and an analog pin.

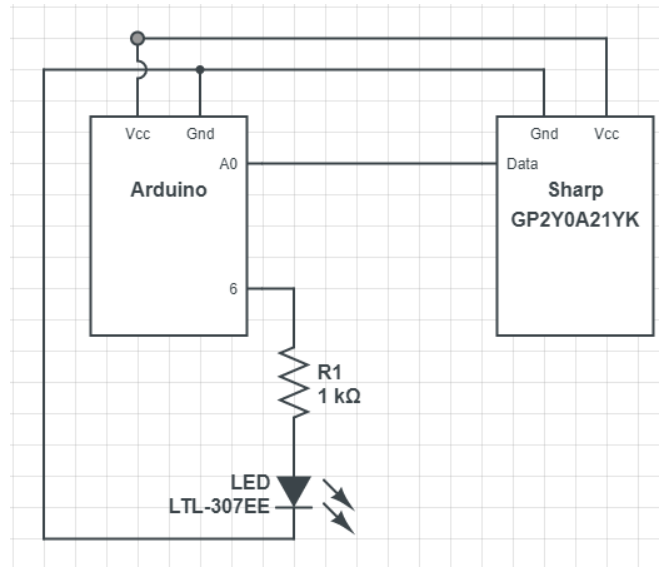


Figure 13: Sharp GP2Y0A21YK IR sensor circuit.

6.6.2 Piezo Buzzer

A module to play an alert sound is needed to implement the safe mode, and a piezo buzzer is used. A Piezo buzzer is used for making beeps, tones and alerts. The sound is decently loud, and it is cheap, only \$1.50. It has two pins and to use, one pin (either one) is connected to ground and the other pin is connected to a square wave out from a micro-controller. For the loudest tones, we need to stay around 4 KHz, but the buzzer works quite well from 2KHz to 10KHz. For extra loudness, we can connect both pins to a micro-controller and swap which pin is high or low ("differential drive") to double the volume.

6.6.3 Safe mode Implementation

The two ultrasonic sensors are connected to interrupt pins. When the safe mode is turned on, whenever it detects an obstacle 80 cm away from the device, the micro-controller will be interrupted, and the information will be checked with the IR sensor to ensure optimal accuracy. Then, the piezo buzzer will beep constantly to alert the users and the supervisors. It will keep beeping until the close proximity of the device is cleared of obstacles. When the obstacle is too close, at 30 cm, and the user has not decided to stop the device, the micro-controller will be interrupted, and the device will smoothly halt to a full stop. It will not move again unless the 30 cm proximity of the car is cleared.

6.7 LCD touch screen - User Inputs

This is the most challenging section of the project, as I have not had prior experience with interfacing an LCD touch screen with a micro-controller.

	TFT Touch Shield (chosen)	Raspberry Pi Touch Display
Pros	Capacitive	Capacitive
	Bright and displays good color.	Big touch screen and higher resolution,
	Has detailed tutorials on Adafruit.	Has good tutorials online.
	Use SPI and I2C (common)	Raspberry Pi proprietary protocol
	Built-in controller with RAM buffering	More demanding on microcontroller.
	Operating Voltage: 3.3V (works with 5V)	Operating Voltage (5V)
	Operating current with backlight: 100mA	Operating current with backlight: 600mA
	Operating current without backlight: 100mA	Operating current with backlight: 200mA
Cons	Small	None
	Needs 5 SPI pins for the display and 2 I2C pins for the controller.	Only works with Raspberry Pi
	Expensive: \$50.	Good price for value.

Table 3: Detailed breakout of potential LCD Touch Screen Candidates.

The table above is the two touch screens considered for this project: one specifically to work with Arduino micro-controllers, and one designed to work with a Raspberry Pi. Since we did not go with a Raspberry Pi, the TFT Touch Shield is a logical choice. Touted by the Arduino community to be the best suited touch screen for the Arduino, it has a capacitive touch screen and a bright and colorful display. It uses SPI and I2C for data communications, which are supported by the Arduino Mega. It has a built-in controller with RAM buffering, thus, most of the processing is done on the touch screen itself, reducing the load on the main micro-controller. Its operating voltage is 3.3V, but it works with 5V micro-controllers. However, the touchscreen itself has some disadvantages: it is small only 2.8 inches, and it requires a lot of ports: 5 SPI pins for the display and 2 SPI pins for the controller.

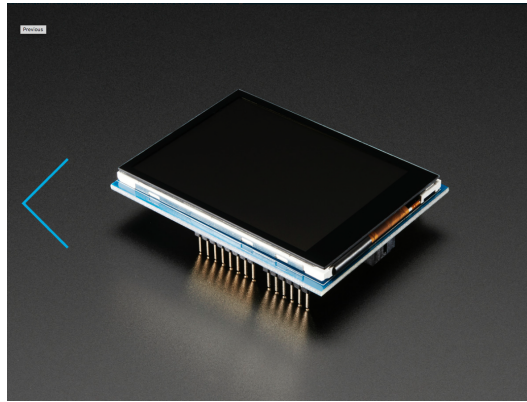


Figure 14: TFT Touch Shield [14].

7 Final Design and Implementation

The final design of the system differs mainly from the preliminary design in that since the TFT capacitive touch screen came embedded in a shield and required more ports than what were available on the Arduino Mega, an additional micro-controller was added. The touch screen is controlled by an Arduino Uno R3, which communicates with the main micro-controller, the Arduino Mega, through Serial Communication. More information on how this was implemented will be provided in section 7.1.1. In addition, on/off and safemode buttons alongside with the speed potentiometer are not implemented, as they can be accomplished through the user interface on the touch screen and the remote controller.

A block diagram detailing how all included electronic parts are connected can be found in figure 15 below.

The system can be broken down into three main sub-functions, each of which attempts to meet a set of design specifications and requirements planned out by the customers over at the Center for Disability Services (Section 4):

- Controlling the motors for four directions of movements:
 - Receive configurations from users through the touch screen (operated by an external micro-controller) and propagate the information back to the main micro-controller.
 - Configure inputs accordingly and send PWM signals to drive the motors in four different directions: forward, backward, left and right.
- Ensuring users' safety:
 - Safe-mode: obstacle avoidance and interrupts any ongoing operations if blocked by an obstacle.
 - Remote kill switch: override all ongoing operations and shut down the entire system at the supervisor's discretion.
 - Fail-safe default: check for the availability of important electronic parts before executing commands on them. Failures in subsystems would result in complete mobility stoppage.
- Display statistical information:

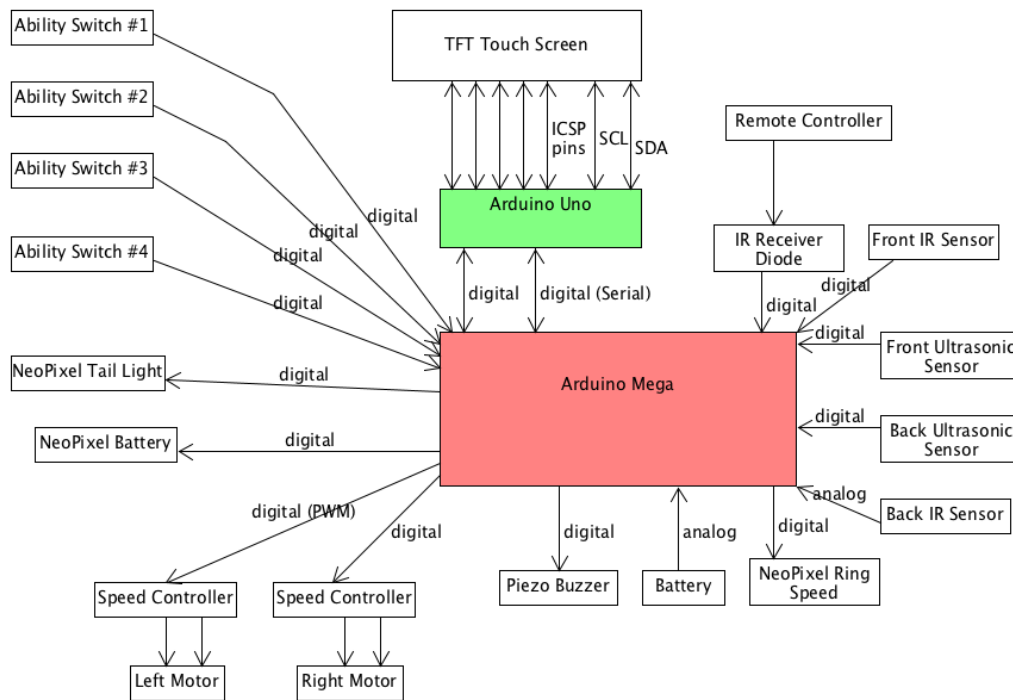


Figure 15: Final implementation block diagram.

- Display battery level information.
- Display speed information.

Each of the sub-functions was accomplished by a combination of hardware and software solutions. Details about how each of them was developed and implemented will be provided in the following sub sections.

7.1 Controlling the Motors

7.1.1 Acquiring User Configurations via a Graphical User Interface on a Capacitive Touch Screen

The TFT touch screen, as detailed in section 6.7, is controlled by an Arduino Uno R3, which is in serial communication with the main micro-controller, Arduino Mega.

Since an Arduino Uno R3 was the only micro-controller available during the development process and was able to work with the TFT touch screen out-of-the-box, it was utilized for the first iteration of this project, but it is not an ideal solution. For more information on how to improve this, refers to the Future Work section.

The GUIslice library was used to design the user interface for the system. It is a lightweight GUI framework suitable for embedded displays, and it works with the purchased TFT capacitive touch screen. However, the baseline Arduino Uno R3 with the ATmega328P chip set has limited SRAM (2KB SRAM, 32KB Flash). Therefore, to design an usable user interface on the touchscreen with the Arduino R3, it is important and essential that GUI elements are stored in FLASH memory whenever possible.

Specifically, in this system, per customers' desires, there are 4 different pages in total, thus to increase performance, all pages and their respective elements are stored in FLASH. Failure to do so would result in the touch screen being unresponsive or not showing enough graphical elements.

- Starter page:
 - Allows users to decide between different number of inputs: 1, 2, 3, 4.
 - Allows users to choose a preferred speed profile: 1 mile per hour, 2 miles per hour, 3 miles per hour.
 - Allows users to toggle safe mode on or off.



Figure 16: User Interface first screen.

- Input mapping page: based on the number of inputs decided in the previous pages, users can decide the capability of each input between forward, backward, left and right.



Figure 17: User Interface second screen.

- If the device is ready, a screen with a ready button will be shown, and if users click on Ready, the device can be used immediately.



Figure 18: User Interface third screen.

- If the device is not ready, a screen with a not ready button will be shown. Clicking on the not ready button will return the touch screen to the starter page. Users will need to check the device for any part failures. For more information on how to do this, refers to the user manual section.

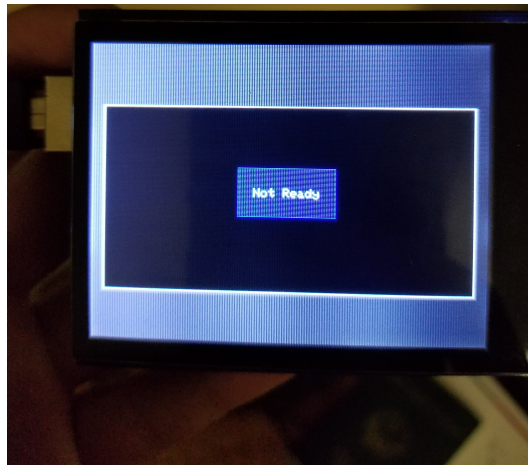
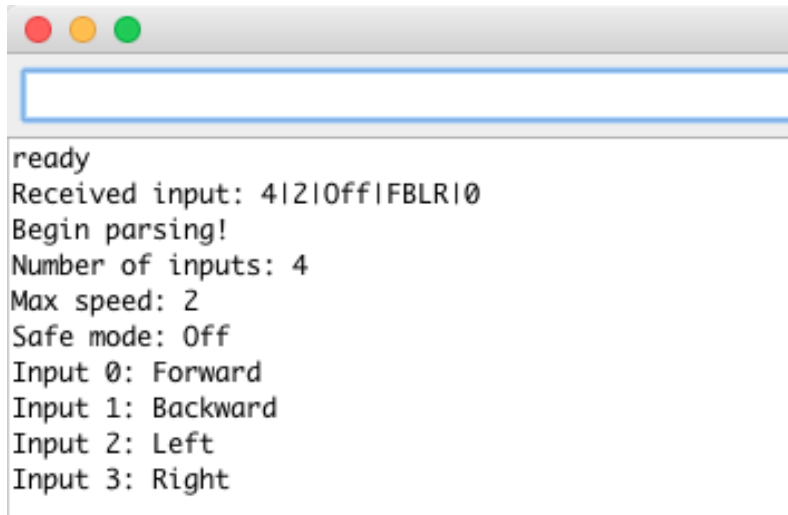


Figure 19: User Interface fourth screen.

After gathering the necessary information, the Arduino Uno will communicate with the Arduino Mega (through pin 0 and 1 on each micro controller) through Serial Communication, and sends users' decisions as one long string, with each decision separated by a "|" to act as a delimiter. |0 will be included at the end as a termination. The string will be split based on | using String's *substring* function. For example, given the string $a = \text{"abc|"}$, $a.substring(0, 3)$ would return "abc". The first index is inclusive and the second index is exclusive. String's *indexOf* function can be used to find the index of the first delimiter |. Sample communication sessions can be found in figure 20 and 21 below.



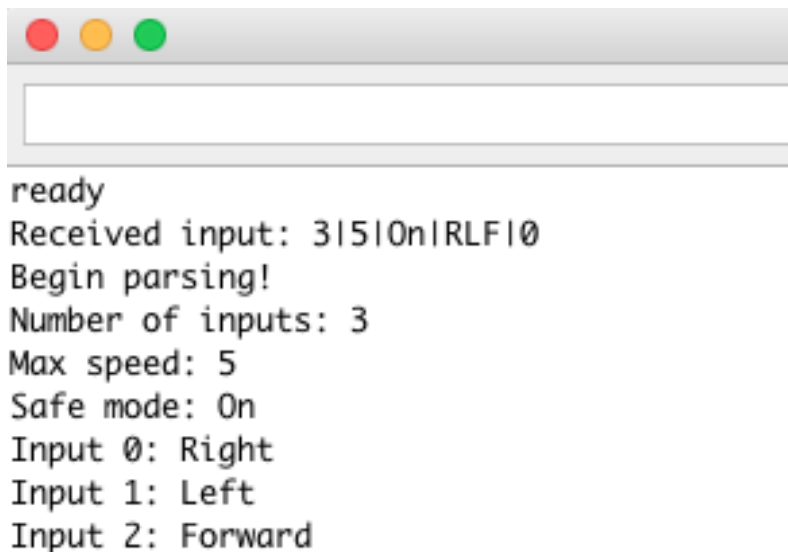
```

ready
Received input: 4|2|Off|FBLR|0
Begin parsing!
Number of inputs: 4
Max speed: 2
Safe mode: Off
Input 0: Forward
Input 1: Backward
Input 2: Left
Input 3: Right

```

Figure 20: Arduino Mega's handling of input 4|2| Off | FBLR |0

The input means that there are 4 inputs, the maximum speed is 2 miles per hour, safe mode is off and mapping for inputs is forward backward left right respectively.



```

ready
Received input: 3|5|On|RLF|0
Begin parsing!
Number of inputs: 3
Max speed: 5
Safe mode: On
Input 0: Right
Input 1: Left
Input 2: Forward

```

Figure 21: Arduino Mega's handling of input 3|5| On | RLF |0.

The input means that there are 3 inputs, the maximum speed is 5 miles per hour, safe mode is on and mapping for inputs is right left forward respectively.

7.1.2 Input Configuration

With user settings received from the Arduino Uno and the TFT touch screen, input configuration will be carried out on the main micro-controller. Per section 6.3, users will control the devices through a set

of push button ability switches (up to 4 switches). Ability switches help disabled people interact with electronic devices, and their signals are transferred through a 3.5 mm jack. Thus, in order for them to work with a traditional micro-controller like the Arduino Mega, an adapter was needed. In this project, an audio socket is used (figure 22 and 23).



Figure 22: Schurter's audio socket.

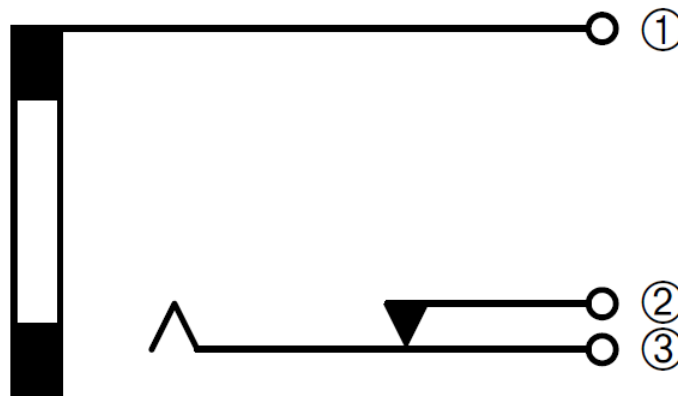


Figure 23: Schurter's audio socket internal diagram.

When an ability switches is inserted, pin 1 and 2 would act as an usual 2-pin push button (normally used in classes such as ECE-118 or ECE-218), and pin 3 would act as a detector of input (whether an ability is plugged into the socket or not). There are four sockets mounted on the device, which allow users to use their own sets of ability switches. For testing purposes, a set of 3 push button switches are used, but in principle, the system would work with any type of ability switches.

During what is called the set up stage, which happens whenever the device is stopped, after user manual settings is received through the touch screen, the number of switches to be used will be changed and each of the switches will be mapped to a certain direction. This setting is configurable to increase the modularity of the project, as during each session of the device, different sets of ability switches can be used to tailor to the capabilities of a specific disabled student.



Figure 24: Ability Switch layout.

7.1.3 Drive motors

As can be seen from figure 25 below, motor output terminals are located above the SPARK logo and are marked with raised lettering. A raised "+" and "-" sign indicate the polarity of the motor terminals. When the SPARK controller is driving the output in the "forward" direction, the output polarity is positive from M+ to M-. When driving in "reverse", the output polarity is reversed [10].

Power input terminals are located below the SPARK logo and are marked with raised lettering. A raised "+" and "-" sign indicate the polarity of the power terminals. As a way to power the two motor controllers and also the two micro-controllers, they were chained in series, which can be seen in figure 26, to a 12V battery, which came with the Wild Thing toy. The SPARK motor controller is intended to operate in a 12V DC system, and the Mega and Uno have built-in voltage regulators which handle stepping down the input voltage to a more manageable value.

The motor controllers accept servo PWM signal through a standard 3 wire PWM cable in the port marked PWM in figure 27. The "-" goes to ground and "s" goes to port 6 (left) and 7 (right) on the Arduino Mega.

The motors can be driven using the Arduino's built-in servo library, specifically the *writeMicroSecond* command. The SPARK responds to a factory default pulse range of $1000\mu s$ to $2000\mu s$. These pulses correspond to full reverse and full forward rotation, respectively, with $1500\mu s$ as the neutral position, that is no rotation. Figure 28 below describes how the pulse range maps to output voltage, motor speed, and motor direction.



Figure 25: Close up of the Motor Controller.

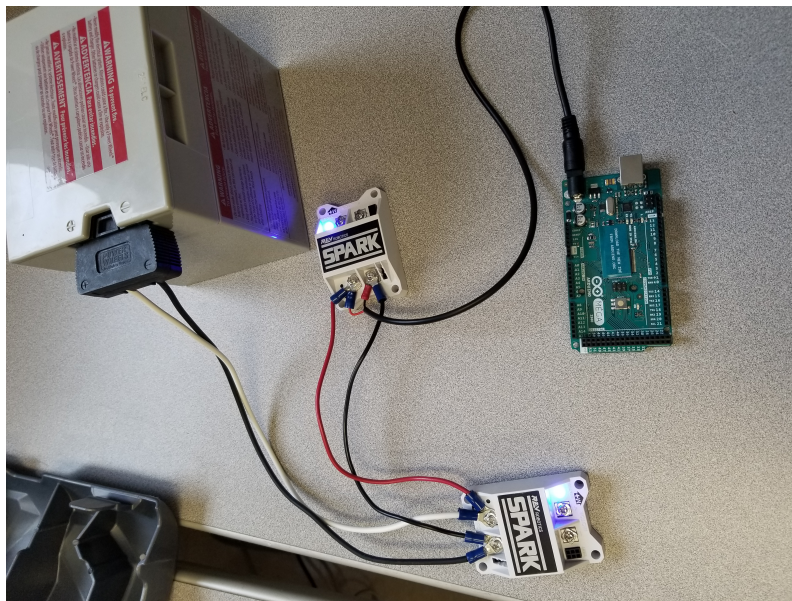


Figure 26: The device's power system.

For input pulse p	Motor Speed and Direction				
	Full Reverse	Prop. Reverse	Neutral	Prop. Forward	Full Forward
Output voltage V_{OUT} (V)	$V_{OUT} = -V_{IN}$	$-V_{IN} < V_{OUT} < 0$	$V_{OUT} = 0$	$0 < V_{OUT} < +V_{IN}$	$V_{OUT} = +V_{IN}$
Default pulse width (μs)	$p \leq 1000$	$1000 < p < 1460$	$1460 \leq p \leq 1540$	$1540 < p < 2000$	$2000 \leq p$
Maximum pulse range (μs)	$500 \leq p \leq 2500$				

Figure 28: Motor controller's pulse ranges [10].

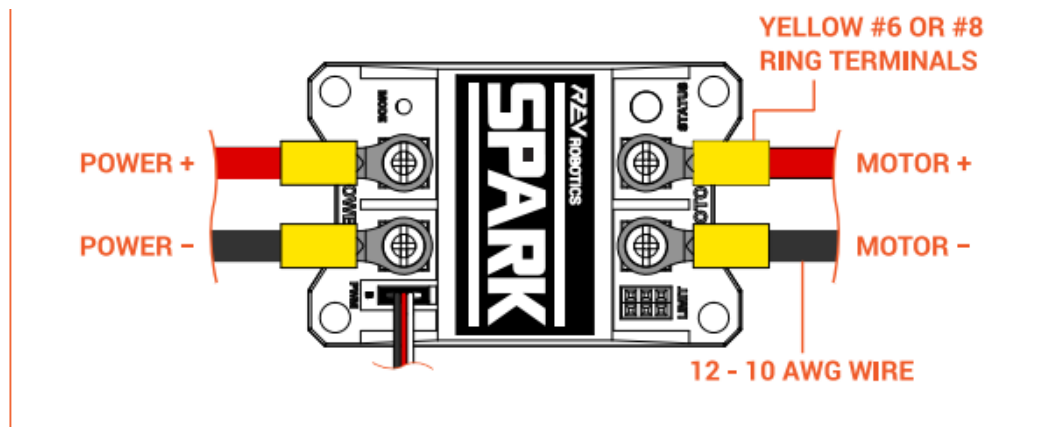


Figure 27: SPARK motor controller's schematic [10].

Using figure 28, three different unloaded speed profiles are created: 1 mph, 2 mph and 5 mph. Unloaded means that this is the speed without an user operating the device. To guarantee a speed profile, an encoder will need to be added and work with with motor controller to ensure the certain speed. More information will be given on this during the Future Work section.

- 1 mph:
 - Forward: 1540 to 1580.
 - Backward: 1460 to 1420.
- 2 mph
 - Forward: 1540 to 1600.
 - Backward: 1460 to 1400.
- 5 mph
 - Forward: 1540 to 1650.
 - Backward: 1460 to 1350.

Forward and backward direction can be achieved at a certain speed using the above speed profiles. For turning left and right, for each speed profile, proportional values are given to each of the motors. Refers to figure 29 to see how it is implemented. When an ability switch is pressed, in this case, the one mapped to the forward direction, a float variable named *speed* will be slowly incremented as long as an ability switch is pressed by 0.0005 each time to make sure acceleration is as smooth as possible. At the end of each iteration, *speed* will be sent to the motors using *writeMicroseconds*. When it exceeds the current maximum value, that is $speed > 1600$, it stays at that level. When the button is released, which would cause the program to exit the accelerate loop, before the system does anything else, it slowly decelerate to the neutral state (refers to the pulse range figure above) by decrementing *speed* by 0.005 each time for a smooth transition.

```
if (digitalRead(forward) == LOW){
    speed = 1560;

    //accelerate
    while (digitalRead(forward) == LOW){
        if (speed >= 1600){
            speed = 1600;
        }
        leftMotor.writeMicroseconds(speed);
        rightMotor.writeMicroseconds(speed);
        speed = speed + 0.0005;
    }

    //decelerate
    while (speed > 1525){
        leftMotor.writeMicroseconds(speed);
        rightMotor.writeMicroseconds(speed);
        speed = speed - 0.005;
    }
}
```

Figure 29: Code for moving at 2 miles per hour speed profile

7.2 Ensuring Safety

7.2.1 Distance sensing and Obstacle Avoidance

Two arrays of sensors (one ultrasonic and one IR) are used, on the front and the back of the device, specifically for detecting obstacle. Two HC-SR04 ultrasonic sensors are employed as they have a long working range, from 4 cm to 400 cm and two Sharp *GP2Y0A21YK* IR sensors are utilized as they have better accuracy when it comes to low range (0.5 cm to 10 cm).

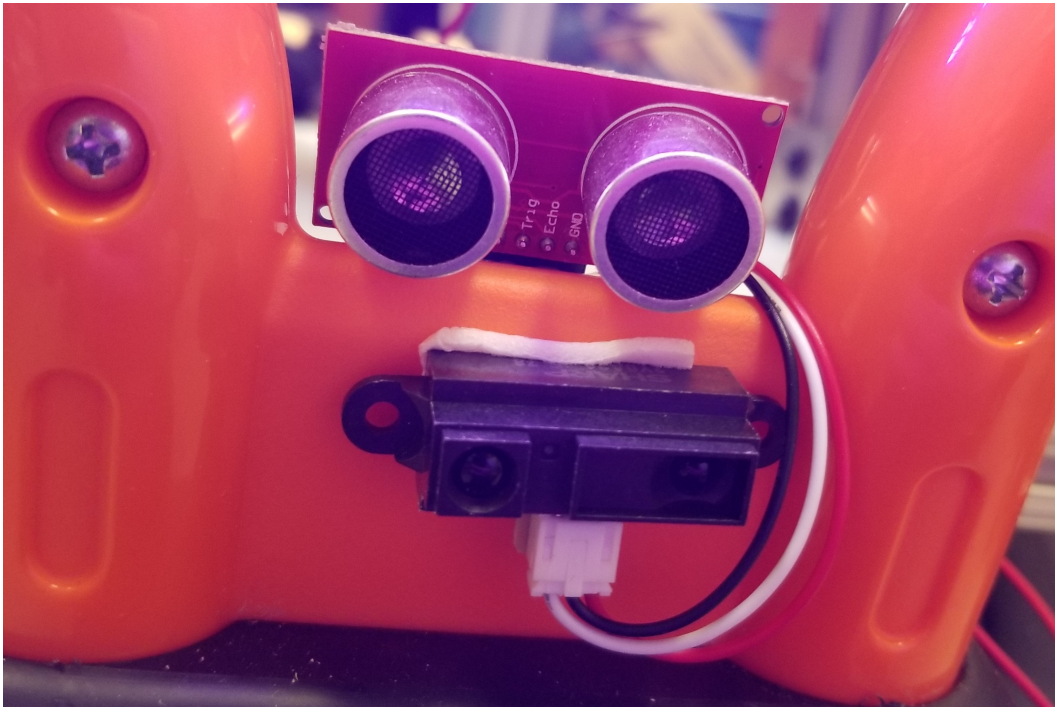


Figure 30: Sensor layout on the back of the device.

An ultrasonic sensor has a trigger pin and an echo pin. When the trigger goes HIGH (1) for a specific amount of time and then LOW (0), the sensor sends out a high frequency sound. When the signal hits an object, it bounces back. The transmitter, or the echo pin, receives it, and has a built-in timer to count how long the sound takes to get back, we will call it t . The distance d in cm from the sensor to the object is calculated as follows:

$$d = \frac{t}{2} \cdot 29.1$$

Since the sensor only works if the trigger pin is set HIGH for a specific amount, there needs to be a way for the ultrasonic sensor to do it constantly to detect any obstacle when the device is the move. The solution is to use the Arduino Mega's built-in Timer 1 through the timer1 library and polls every $50\mu s$ to set the trigger pin, and have the echo pin on an external interrupt pin (2 for the front ultrasonic sensor and 3 for the back ultrasonic sensor), so that when the feedback comes back, the echo pin will interrupt the main program, calculate the distance, compare it with the reading of the analog sensors (through Arduino's *analogRead* and act accordingly:

- If the obstacle is in 80cm range, a piezo buzzer (connected to pin 24) will play a sound to alert the user and her supervisor.
- If the obstacle is in 30cm range, this will result in complete mobility stoppage. 30cm range is chosen for a smooth deceleration of the vehicle. The device will not be operational until it is clear of obstacle in the 30cm range.

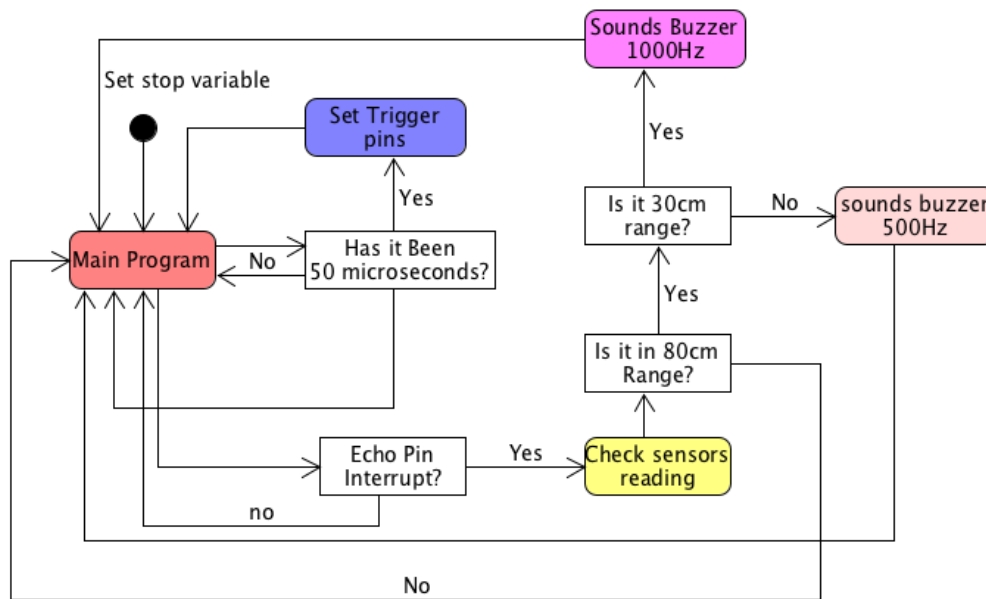


Figure 31: Flowchart to represent the implementation of proximity sensing.

7.2.2 Remote Controller and Remote Kill Switch

Per customers' request, a remote controller was developed for this project. A TSOP 38238 IR receiver diode is used to read and decode signals coming from an IR remote. The receiver is extremely robust such that it can interface signals from every type of IR remotes. For this project, three different types of remote from different manufacturer (Sharp, Sony and Samsung) were tested. The receiver diode is connected directly to an external interrupt pin, and as it receives a signal, it will interrupt the main program, decode the signal and act accordingly (figure 32):

- If the power button is pressed (HEX value: *C573E684*), the device will slowly decelerate and will not be operational again until the button is pressed again. This is achieved by changing a boolean variable (a variable which can only be either True or False). The code is only run if the variable is set to True, and vice versa.
- The remote controller can also be used to control the device by using the four directional buttons (HEX value: forward - *2DD37549*, backward - *EBEDBD06*, left - *A6B89AAB* and right - *B4a116F2*). However, this is only possible when the on-board switches are not used to prevent any conflicts. Likewise, the on-board switches are not operational when the remote controller's directional buttons are in use.

7.2.3 Fail-safe Default

Fail-safe default is a common security design principle which helps applications and programs to fail gracefully in the event of a dangerous error. In the concept of this project, this design principle

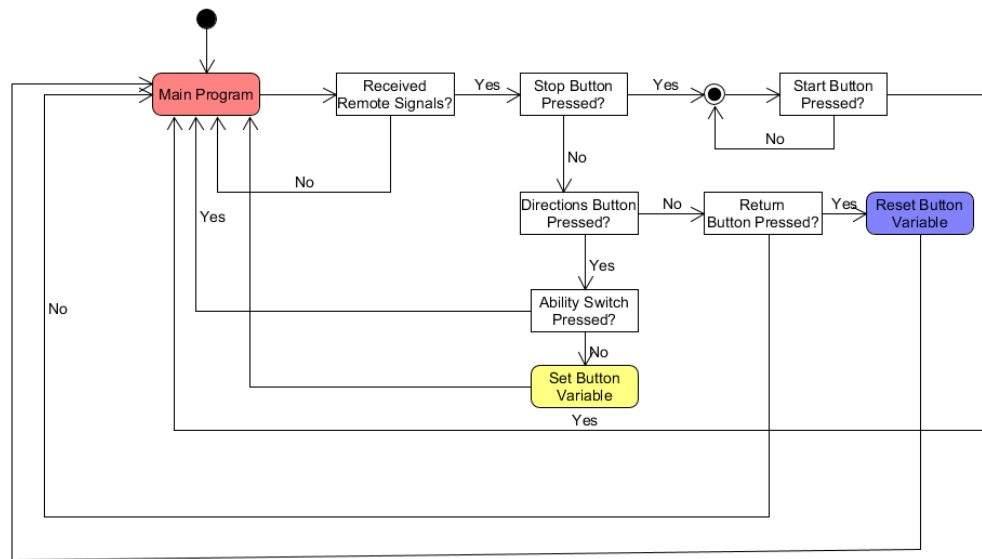


Figure 32: Flow chart representing how the remote controller is implemented in software.

aided in constructing a system which would be shutdown once one of its core electronic parts is not functional. This is necessary as this project will be used by small children, and thus, safety is of the utmost importance.

In a normal Arduino program, there is a setup function that will be executed once at the beginning of the application life cycle and a loop function that will be executed consecutively until the application is interrupted or shutdown. In the setup function, after the Mega receives user initial settings from the touch screen (section 7.1.1), it will check if all of the parts are functional (detailed in the list below) and sends a "READY" or "NOT READY" to the touchscreen to notify the users accordingly.

At the subsequent iterations of the loop program. before executing any commands, the micro-controller will check:

- Ability Switches Input: Check to see if all the desire switch sockets are present and connected (utilizing pin 3 of the sockets, for more information, refers to section 7.1.2). The device is only operation if all designated inputs are present. One input going offline will result in complete mobility stoppage immediately.
- Proximity Sensors: if the trigger pins (both front and back sensor's) cannot be set or the echo pin is not operational, that is, the reading of the echo pin returns 0 repeatedly, the device will not be operational. The IR sensors are not checked, as their failures are not fatal to the device's operation.
- Motors: If the read() function call from the Arduino's servo library - which returns the current angle of the servo (the value passed to the lass call to write(), which is called to set the angle to 90 degree after each time the motor is used (*)) - returns 0, the device is stopped as it is not the intended value.
- If battery level is too low, the device will not be operational. For more details on this information is gathered, refers to section 7.3.1.

- During initial and subsequent setups, the touch screen will not say that it is ready if all three conditions above are satisfied. In each case, the device will slowly decelerate to ensure the safety and comfort of the current user.
- During the run time of the program, if any of the check fails, the program will be halted, and a “NOT READY” sign will be sent to the touch screen to alert users and their supervisors.

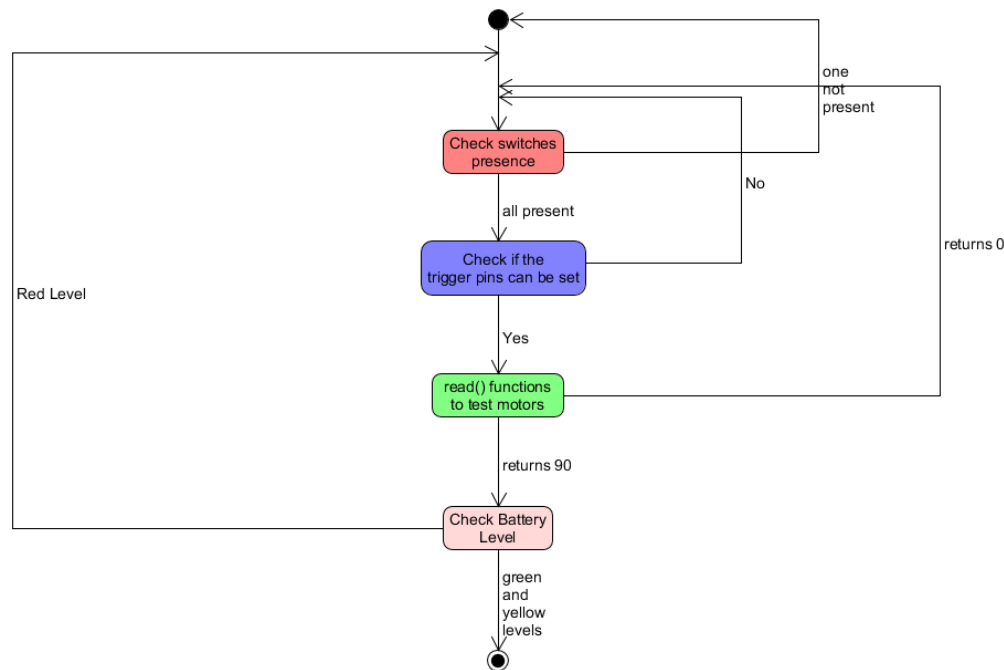


Figure 33: Flowchart representing the implementation of fail-safe default in software.

Note: *write()* does change the value that *read()* returns, but does not do anything to the motor, which is why it was used to test the availability of the motors.

7.3 Display Statistical Information

7.3.1 Battery Voltage Measurements

The Wild Thing toy comes with a 12V rechargeable battery, which powers the two micro-controllers including all of the parts that connect to them, the two motor controllers and the two motors. Thus, it is necessary that users and their supervisor are able to know when the battery is running out, and interrupt the session to recharge it. For the first iteration of the project, this is accomplished by having the Arduino Mega measure the voltage level of the battery. Since analog inputs for the Mega can only accept voltages up to 5V, the voltage from the battery needs to be scaled down with a simple voltage divider circuit (simple resistive divider). The scaled down value then can be fed into the Mega’s ADC (analog pin A2). The battery and the Arduino share the same common ground.

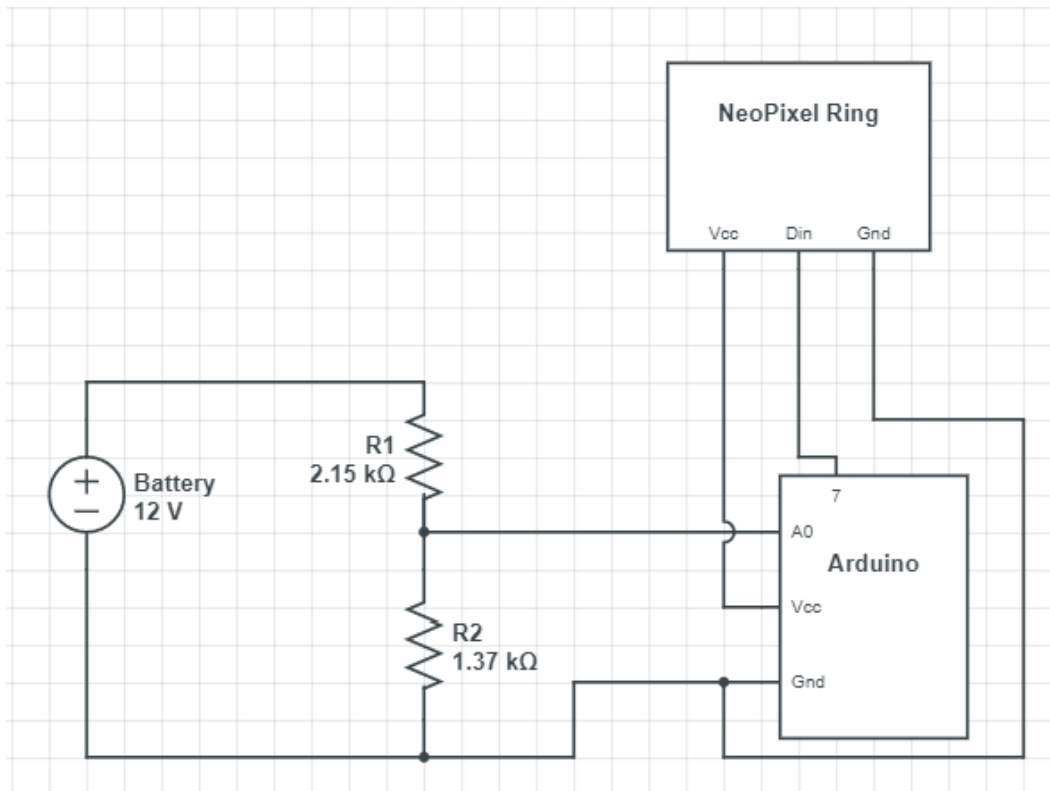


Figure 34: Battery measurement voltage (repeated from page 19).

Divider circuit equation:

$$V_o = V_i \cdot \frac{1370}{1370 + 2150} (V)$$

With this equation, the maximum voltage 12.8V corresponds to 5V. When the voltage of the battery drops down to around 11V, the battery is already “exhausted”, and cannot operate normally. Obvious signs are motors moving much slower and buzzer sometimes not making sound when it should. A NeoPixel Ring with 24 LEDs is used to display this statistical information. The inputs of LEDs on a ring are chained together from one LED to the next, so it only requires one PWM pin from the Arduino Mega.

- When the battery voltage level is above 11.5V, the ring will be green.
- When the battery voltage level is between 11.1V and 11.5V, the ring will be yellow.
- When the battery voltage level is at or below 11.1V, the ring will be red.

7.3.2 Max Speed and Current Speed Notifier

There will be two NeoPixel rings, one with 24 LEDs and one with 16 LEDs, to display these two information. As mentioned in section 7.1.3, there are three different speed profiles, each of which corresponds to a maximum unloaded speed that the device can attain. One NeoPixel Ring with 24 LEDs will be dedicated to this, which will display blue for 1 mile per hour, green for 2 miles per hour and purple for 5 miles per hour.



Figure 35: Three stages of battery level shown on a 24 LED NeoPixel Ring.



Figure 36: Three different speed profiles shown on a 24 LED NeoPixel Ring.

Another NeoPixel Ring with 16 LEDs will act similar to a tail light for a car. If the device is accelerating forward, the LEDs will incrementally be lit up and the color is green. If the device is decelerating backward, the LEDs will incrementally be lit up and the color is red. When the device is stopped either because of fail safe default, or because of an obstacle, the ring will blink red consistently to grab the supervisors' attention.



Figure 37: Tail light NeoPixel's LEDs incrementally be lit up when device is accelerating forward.

The placements of the three NeoPixel rings on the device (captured from the back) can be found in figure 38 below. Left: battery NeoPixel, Right: speed NeoPixel, Center: Tail light NeoPixel.



Figure 38: NeoPixel rings' placements on the device.

7.4 Overall Implementation of the Project

The first iteration of the project, as seen in figure 39 below, is a combination of sections 7.1, 7.2 and 7.3 in both hardware and software.



Figure 39: First Iteration of the Project.

In terms of hardware, the micro-controller, the battery and the two motor controllers are placed

under the chair and is covered by a white plastic piece as shown in the figure. The four sockets are mounted under the wooden tray with wires going along the rails of the chair to the micro-controller. The sensors are mounted on the front and on the back, looking straight up for a good performance. The three NeoPixel Rings and the touch screen are mounted behind the chair. The touch screen is mounted behind the chair is to prevent the children from accidentally altering the settings. Note: on the first iteration of the project, all electronic parts on the outside are currently only glued to the toy. This is because before being permanently mounted on the second iteration, optimum placements for them must be decided first. Detail pin out of the micro-controller can be found in table 2 below. 21 digital pins out of 54 digital pins are used. Out of which, 8 PWM pins out of 15 PWM pins and 3 interrupt pins out 6 interrupt pins are used. 3 analog pins out of 16 analog pins are connected.

Type	Part	Pin
Ultrasonic sensor	Front echo	2, digital, Interrupt
	Front Trigger	4, digital
	Back echo	3, digital, interrupt
	Back Trigger	5, digital
IR sensor	front IR sensor	A1, analog
	back IR sensor	A0, analog
Battery measurement	Battery measurement circuit	A2, analog
Ability switches	Switch 1	11, digital, PWM
	Switch 2	12, digital, PWM
	Switch 3	13, digital, PWM
	Switch 4	22, digital
	Switch 1 present	23, digital, PWM
	Switch 2 present	24, digital, PWM
	Switch 3 present	25, digital, PWM
	Switch 4 present	26, digital, PWM
Motor Controller	Left motor controller	6, digital, PWM
	Right motor controller	7, digital, PWM
LED arrays	Speed NeoPixel Ring	8, digital, PWM
	Battery NeoPixel Ring	9, digital, PWM
	Tail Light NeoPixel Ring	10, digital, PWM
Buzzer	Piezo Buzer	24, digital
Remote controller	IR Receiver	18, digital, interrupt
Serial Communications	Receiver	0, digital
	Transmitter	1, digital

Table 4: Detailed Pin out.

Software wise, the project integrates the logics of proximity sensing, fail-safe default, remote controlling, input configuration, communications with the touchscreen, driving the motor into one single program. The UML activity diagram in figure 40 belows details how they are amalgamated. The fully commented code can be found in Appendix A. To accomplish these logics, numerous third party libraries were used:

- Servo library.

- GUIslice library.
- IR Remote library.
- NeoPixel library.
- Timer1 library.
- Arduino base libraries which include the tone library for the Piezo buzzer.

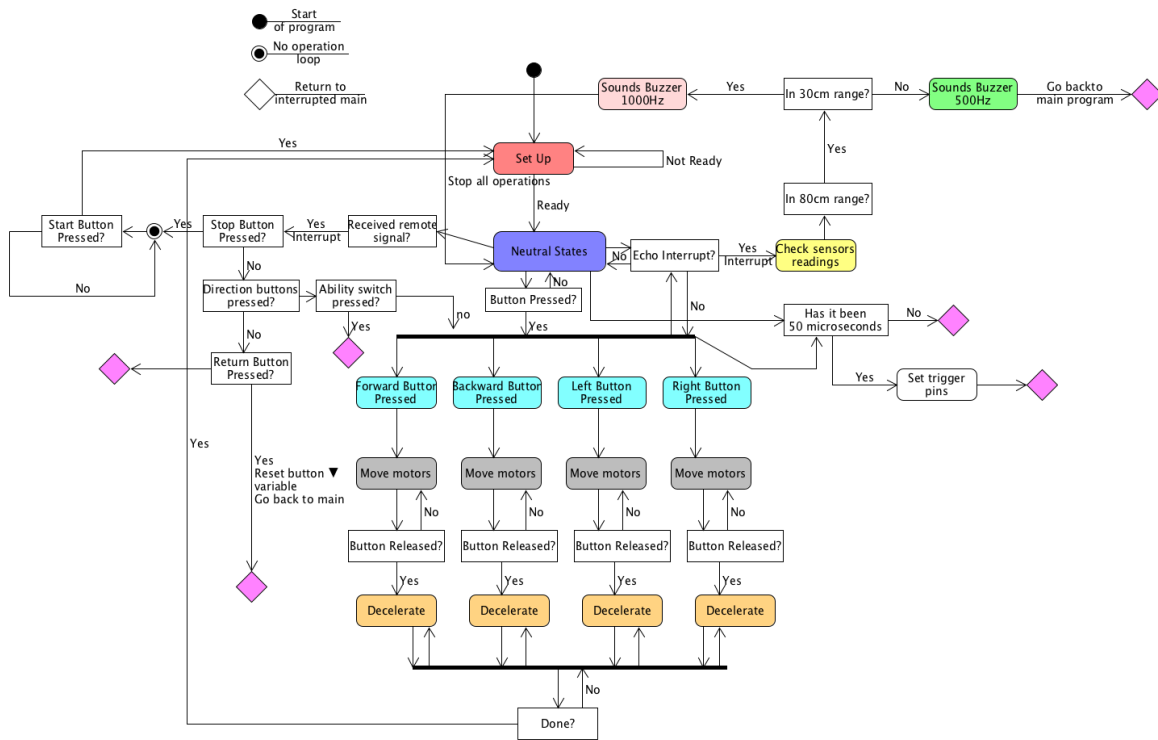


Figure 40: Flowchart representing the overall software implementation of the project.

8 Overall System Performance

Within this section, the expected performance of the preliminary design will be discussed and compared with the performance of the final design. The level of satisfaction of the design requirements will be discussed. Finally the testing process will be described in detail and recommendations to improve performance and future work will be addressed in the following section (section 9).

8.1 Preliminary Design Performance Estimates

The preliminary design was expected to provide a smooth wheelchair driving experience for small disabled children. It was supposed to interface any set of different ability switches (up to 4 switches) and control the Wild Thing toy for four directions of movements: Forward, Backward, Left and Right.

It was expected to be modular, i.e, it could be configured on the go and allow disabled children to navigate freely without significant help from the supervisors. The driving experience was supposed to be smooth (accelerating and decelerating) and wiring should be simple and easy to follow. In addition, the design was expected to be easily maintainable and upgradeable, so that the Kevin G. Langan School at the Center for Disabilities Services and any upcoming senior engineering students can work on and improve the system.

8.2 Final Implementation Actual Performance

Note: refers to section 4 for the set of design requirements mentioned explicitly in this section.

The first iteration of the project (completed in Winter 2018 and considered Final Implementation in this paper) matches the preliminary design performance estimates quite well. While it manages to match the goals, the implementation of the system was changed quite significantly (the complexity of the software were increased greatly) for this to happen. Thanks to the 3.5mm audio sockets, the controller can interface any set of different ability switches, since they all work similar to a push button switches. However, more testing will need to be carried out to confirm this assessment, and the code will need to be modified for people with different disabilities. The modularity when it comes to input to the system is achieved and the compatibility design requirements were met.

The complexities of the project were increased greatly when an additional micro-controller was needed to control the touch screen. Because it was a baseline Arduino, it has limited SRAM, and thus an extended amount of work-arounds had to be accomplished for it to control the GUI well enough for a good user experience. In addition, another overhead was introduced as there needed to be a good way for the two micro-controllers to communicate and transfer information. The current solution is to send a string over serial communication with a "|" acting as a delimiter, but the performance is much slower than having the touch screen operating natively on the main micro-controller, due to the fact that Serial Communication transfers strings a byte at a time. Nevertheless, the implementation of the touch screen was successful, and the device can be reconfigured whenever it is stopped. It just takes close to 30 seconds for the system to be ready to be used again. User settings and input configurations work as expected. The reconfigurability of the performance design requirements were met.

In the preliminary design, the sensor was supposed to be connected to an interrupt pin, and would interrupt the main program whenever it detected an obstacle. However, the ultrasonic sensors cannot work this way, as does a normal digital proximity sensor, since the trigger pin needs to be set manually. A solution is to use a timer to interrupt the program every $50\mu\text{s}$ to set the trigger pin. The performance is not affected, but it results in the code being much longer and more intricate. Obstacle avoidance and remote controller work as expected, that is, it is able to control and kill the device if need be. The safety design requirement were achieved successfully.

Without any real users, the device can attain the wanted speed in all three speed profiles, and can accelerate and decelerate smoothly and safely. Nonetheless, when an user is on the device which increases the weight applied on the motors, speed varied a lot. In order for this to be improved, an encoder would need to be added. Refers to section 8.3 for more details on this. The battery on a heavy load, however, performs significantly worse than expected. It was intended to last for 2 hours, but in reality, can only support the system for 45 minutes on average, before a real decrease in performance is noticeable. The performance design requirements were met on an acceptable level, but more work will need to be put in to perfect it. There are LEDs to tell when the battery is going low, the currently active speed profile and the movement of the device, which satisfy the debugging design requirement.

8.3 Development and Testing Process

This project was tested and developed incrementally as each parts came in.

- Tested interfacing ability switches with the Arduino Mega micro-controller using the audio sockets:
 - Recorded the *digitalRead* values.
 - Utilized *digitalRead* of the switches to manipulate the brightness of a LED. Snippet of the codes can be found in Appendix C.
- Received and tested the SPARK motor controllers:
 - Created a power system where two motor controllers and an Arduino Mega were powered in series by a 12V battery.
 - Sent PWM signals to the motor controllers, and understood how the signals worked by looking at their informational LEDs.

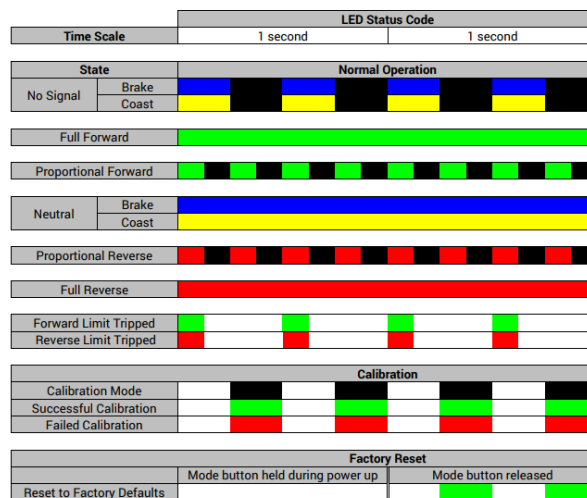


Figure 41: SPARK motor controller's LED status code.

- Connected the motor controllers to the two motors, and used the ability switches to send PWM signals to drive the motors using *analogWrite*.
- Used the ability switches to send servo PWM signals to drive the motors.
- Received and tested the ultrasonic sensors and IR sensors.
 - Tested basic proximity sensing with both of the sensors.
 - Integrated them with a piezo buzzer to implement safe mode.
- Received and tested the IR receiver diode.
 - Tested reading signals from an IR remote.

- Designed and tested the remote controller and remote kill switch.
- Received and tested the touch screen.
 - Connected the TFT touch shield to a micro-controller and implemented a simple screen with a interactive button.
 - Implemented a drawing pad for the touch screen.
 - Implemented a front-end only multi-page touch screen for the project.
 - Connected the touch screen to an Arduino Uno and had it communicate with the main micro-controllers.

When the system was fully developed and the flexible seating customizations from Joseph Caruso were finished, extensive testing were promptly carried out. The device were driven around (without any users) substantially to fine tune the smoothness of acceleration and deceleration. Safe mode was also tested by having multiple obstacles on the device's path. In all cases, the device managed to either detect or stop gracefully when encountered an obstacle.

The device was first demonstrated to Mr. Jim Luther and Mrs. Gillian Whelan (who replaced Mrs. Laura Shemo as my clinical contact) from the Kevin G. Langan School at the Center for Disability Services on February 28th, 2018. Professor Cherrice Traver, Professor William Keat and Joseph Caruso also attended the meeting. The device received excellent feedbacks and various comments on potential improvement, which will be discussed in details in the Future Work section.

The device was demonstrated to Union College as a part of ECE 499 on March 13th 2018. A presentation on the project to fellow Electrical and Computer Engineering professors and students was also given on February 24th. In addition, I will also hold a poster session on the second iteration of the project during 2018 Steinmetz Symposium.

9 Potential for Improvements and Future Work

While the first iteration of the project proved to be successful, substantial improvements can still be made. Firstly, as mentioned in section 7.1.1, the baseline Arduino we used for the TFT touch screen was not a good and efficient solution. The micro-controller had limited SRAM available, which hinders the real-time performance of the touch screen. Thus, in the next iteration of the project, one of the following alternative solutions can be implemented.

- The most obvious solution is to replace the Arduino Uno with a more capable Arduino Mega, which would solve the memory and performance problem. This would be really easy to implement, but, with this solution, two different micro-controllers are still needed for the project, which means a more complex and harder to maintain system.
- The second solution is to figure out which pins on the TFT touch screen shield are necessary for the project, and attempt to control the touch screen via the main micro-controller. However, this would increase the complexity of the current software running on the micro-controller, which is already a long program and currently it is not guaranteed that the main micro-controller can sufficiently manage the touch screen.

- The third solution is to discard the physical touch screen and develop a mobile application that can handle the job of getting users' preferred settings and configurations. The advantage of this solution is that there is no need for a second micro-controller, and the supervisors at the school all have smart phones with blue-tooth and Wifi capability. In addition, if implemented well, it can also remove the need for an IR remote controller and informational LEDs, since it can be accomplished with the same mobile application on a smart phone. However, this would also increase the complexity of the project. I would need to develop and deploy fully functional iOS and Android applications (cross-platform is necessary) for the system via Apple's Xcode and Google's Android SDK. It would also require acquiring additional hardware for the Arduino Mega, as the base model does not have blue-tooth and Wifi capability. There are two routes for this solution, each requires a different hardware and software approach:
 - Wifi and the Internet: this approach requires a wifi module for the Arduinos. With it, the Mega and the mobile application can communicate and transfer information through Amazon Web Services using a lambda function. Previous data can also be saved into an online relational database for analytical purposes. The disadvantage of this solution is that it requires a constant Internet connection for the device to operate normally.
 - Bluetooth: this approach requires a bluetooth module for the Arduinos. The most commonly used one is a HC-05 Bluetooth module. Bluetooth is one of the popular wireless communication technologies because of its low power consumption, low cost and a light stack but compensates on range. The application on the smartphone would communicate with the bluetooth module, which then sends the information to the microcontroller via Serial Communication. The advantage of this approach over the previous one is that it can work anywhere and does not require an internet connection. Nonetheless, there might be a noticeable delay in real-time performance due to its reliance on Serial Communication.

During the demonstration during week 9 with the customers over at the Kevin G. Langan School, they recommended adding features which would improve the device's modularity and its compatibility with different disabled students. Some students are not able to register a nice firm push on an ability switch instantly (they would hit it multiple times before resting their hands firmly on it). Thus, it would be more preferable to add an option that the supervisors can choose during setup stages on the touch screen which tells the micro-controller to not accept any inputs until a firm press has been registered. This would also prevent snappy movements of the motors, which could be potentially harmful to a disabled student. In addition, for many students, after they have firmly rested their hands on an ability switch, it is exceptionally difficult for them to raise their hands off it to stop the device. For this specific reasons, to prevent the cases where the device cannot be stopped, a timeout option (that can be chosen during setup) to slowly stop the motors after a configurable amount of time needs to be added. Furthermore, encoders for the motors are essential in ensuring that the device is moving at the intended speed.

To make the device more modular and safer to use, support for analog joysticks will need to be added, as they are what most commercial wheelchairs use. Also, more extensive testing and fine tuning of the software to improve the driving experience are preferable. A more elaborate implementation of safe mode to aid students in operating the device with added features such as auto-alignment and curve detection are requested by our contacts at the Kevin G. Langan School. Moreover, when being reproduced, the device will model and follow closely the existing standards and protocols for powered wheelchairs.

10 Production Schedule

10.1 Original Production Schedule for Winter Break 2017 and Winter Term 2018

At the end of Fall Term 2017, I originally intended to finish the testing of all individual parts of the system during the winter break, so that Winter Term 2018 could be spent to combine them into a fully functional working prototype and to perform extensive testing. Joseph Caruso and I planned to have the device fully working and demonstrate it to the Center for Disability Services at the beginning of week 8 of Winter Term 2018.

10.1.1 Original Winter Break Schedule (6 weeks)

- Implement On-off button. (Week 1).
- Test motor controllers: (Week 2 and 3).
 - Install them into the Wild Thing, replacing the old motor-controllers.
 - Test their signals.
 - Test driving the motors using Pulse Width Modulation.
 - Test changing the speed with a potentiometer.
- Interface ability switches with the Arduino Mega (Week 3 and 4):
 - Test generating PWM using them.
 - * Test using PWM to change LED brightness.
 - Test generating PWM to control the motors.
- Implement Safe mode (Week 5 and 6):
 - Test ultrasonic and IR sensors.
 - Implement and test remote controller.
 - Interface sensors with Piezo.
 - Combine the parts together to construct safe mode.

10.1.2 Schedule for Winter term

- Combine the existing parts. (Week 1)
- Perform another round of testing for the individual parts. (Week 1 and 2).
- Interface NeoPixel Ring for debugging. (Week 2).
- Implement the touch screen: (Week 3 and 4).
 - Test getting input.
 - Test getting input and changing states.
 - Interface with the rest of the system.

- Perform extensive testing on the whole system. The battery lasts 2 hour on default, and with the system, expected battery life is 1.5 hour. (Week 5 and 6).
- Integrate into a full device with Joseph Carruso from the Mechanical Engineering department, and have a demo test at the Kevin G. Langan School at the Center for Disability Services. This should be done by week 8. (Week 7 and 8)
- Write instructions on how to operate the device. (Week 8).
- Write a full manual on how to replicate the device. (Week 9).
- Write the ECE-499 final report. (Week 9 and 10).

10.2 Actual Project Schedule Analysis

While it was a good idea to set tight deadlines to encourage a fast and on-time project, the timeline did prove itself fairly ambitious. Most of the parts arrived on time for the winter break work load. However, during that time, I was without Professor Traver's guidance, and could not finish everything that was originally planned. Everything on the original schedule was finished except for the motor controllers. I had difficulties with them since in order to install the new motor controllers into the Wild Thing, I would need to remove the joy sticks, the many protective layers and the original motherboard. Due to this, I did not have enough time during the winter break and had to delay it until the new term began.

Overall, the original plan for Winter Term 2018 was followed closely, and every presentation and demonstration deadlines were met. Nonetheless had more work been done during Winter Break 2017, Winter Term would not have been as rushed, which would allow for more time to extensively test and improve the project. The actual project schedule can be found in table 5 below.

10.3 Spring Term Schedule

The project will continue into Spring term 2018 (no course credit), in which a second and improved iteration of the project will be produced, software wise. Joseph Caruso will not continue, thus the flexible seating options will not be updated. The changes mentioned in section 9 will be made, and the project will be tested by a human subject on campus first, then tested at the Kevin G. Langan School. After finals changes are made based on received feedbacks, the project will be deployed and used at the Center for Disability Services.

Week	Joseph Caruso	Lam Ngo
3	-Finalize dimensions and chair design	Take off the default processing unit, joysticks, and protective layers.
	-Order parts including tilt n space brackets	Mount wires and connector for motor controllers
	-Build dimensional mockup	Mount wires and connector for motor drivers
4		Connect motor drivers, controllers and battery
	-Analyze tilt n space brackets	Motor controllers extensive testing
	-Determine any modification needed for brackets	Connect all individual parts
		Substantial Individual Part Testing
5		Start developing Arduino Software
	-Begin assembly of chair and mounting frame	Finish part ordering (NeoPixel and Touch Screen)
		Test new parts as they come in
6		Continue software development
	-Finalize assembly of chair and test chair functions	Finalize the software, and work with Joseph Carruso to mount all the necessary part
		Start testing device's operations
7		Fix bugs as they occur
	-Initial testing of at Langan School and receive their input on the design	Continue Testing
		Implement the LCD touch screen
8		Departmental Presentation
	-Finalize design based on input received from the Langan School	Fix any remaining and finalizing the LCD touch screen implementation
9		Demonstration with the Kevin G. Langan School
	-Project report and written design report	Document feedbacks
10		Public demonstration of the project
	-Present finalized design to the Langan School for their use	Write final report
		Finalize the software and the first iteration of the project
		Finalize project website.

Table 5: Detailed Winter Term 2018 Schedule.

11 Cost Analysis

Table 6 below details the overall cost for all needed components. Since the Wild Thing[®] toy is donated to Union College by the Center for Disability Services, it is not included in the final cost.

- Total component cost: \$342.90
- Total Shipping cost: \$22.63

- From vendor Adafruit: \$7.17.
 - From vendor Sparkfun: \$0 since order is more than \$75.
 - From vendor RevRobotics: \$5.46.
 - From vendor Enabling Devices: \$10
 - **Final cost:** \$365.53.
- Student Research Grant: \$344.00. The remaining \$21.53 is funded by the Electrical and Computer Engineering department. This is due to the additional purchase of an Arduino Uno for the TFT touch screen.

Type	Part Number/Description	Link	Price	Note	Quantity	Total
Microcontroller	Arduino Mega 2560 Rev3	Arduino	\$38.50		1	\$38.50
LCD Touch Screen	TFT Touch Shield With Capacitive Touch	Adafruit	\$44.95		1	\$44.95
IR Receiver	IR Receiver Diode	Sparkfun	\$1.95		1	\$1.95
Buzzer	Piezo Buzzer	Adafruit	\$1.95		1	\$1.95
Distance Sensor	HC-SR04	Adafruit	\$3.95		2	\$7.90
	Sharp GP2Y0A21YK	Sparkfun	\$14.95		2	\$27.90
LED arrays	Neo-pixel Ring	Adafruit	\$16.95		3	\$50.85
Speed Controller	Spark Motor Controller	RevRobotics	\$45		2	\$90
Ability Switches	PushButton Switches, set of 3	enabling devices	\$65.95	This was donated to Union College	1	\$65.95
Ride-on Toy	Wild Thing	Jet.com	\$249.99		1	\$249.99
Microcontroller	Arduino Uno R3	Arduino	\$22.00	Additional buy for the TFT touch screen	1	\$22.00
					Total	\$592.89

Table 6: Total cost breakdown.

12 User's Manual

This section provides detailed instructions on how to assemble, operate, and maintain the system. It is assumed that the device comes with the NeoPixel, sensors, touch screen, Arduino Mega, battery,

power rails already mounted, the Arduino sketch and all necessary libraries are on the user's computer. If not, they can refer to the appendices of this report, or my public github repository: <https://github.com/lamsonbango/Power-Mobility>.

12.1 Assembly and Program Setup

1. Disconnect the battery and the power plug from the micro-controller.
2. Remove the chair and the cover lid per Joseph Caruso's instruction.
3. Connect 5V and Ground from the Arduino to the power rails.
4. Connect PWM pins from the two motor controllers to the Arduino micro-controller and the ground pins to the power rails. Refers to the detailed pin out table. Ground is the one close to the raised letter *B* and the PWM pin is on the other side. The middle pin is left unconnected.
5. Put the chair back on top of the device and screw them in place per Joseph Caruso's instruction.
6. The chair should have the wire already going through its rails like the picture. If not, one should attempt to wire them like that before the chair is put back on.
7. The wires' ends should have tags saying which part they belong to. Connect all red and black wires to power rails.
8. Connect green and blue wires to the micro-controller per the pinout table. Once finished, check to make sure there is no loose connect and no short (red wires plugged into "-" rail and vice versa.)
9. Reconnect the battery and plug the power cord into the main micro-controller and the second micro-controller. If successful, you should see the motor controllers' leds being blue and the touch screen lit up and functional. Wait for 30 seconds for the device to finish initial setup (starting up Serial communication). If not successful, double check each wire connection to the main micro-controller. The other ends of the wires are glued to the parts, so it is impossible for them to be faulty.
10. If the touch-screen is lit up, but not showing anything, that means the program has not yet been loaded. Disconnect the power cord and connect the second Arduino to a computer with Arduino IDE installed using the included cable. If the software has not been downloaded, head to my github repository, click Download as zip and unzip. Go into the folder "Libraries" and copy all of the folders in there into your Arduino library folder. On macOS, it should be default to Document\Arduino\libraries. Go into the folder "Touch Screen", open the file "touchscreen.ino" with Arduino IDE. Make sure in "Tools", "Port" is set to Arduino Uno and there is a correct port connection (figure 42). Click verify and click upload. Wait for 30 seconds. Disconnect the cable and reconnect the power cord.
11. Disconnect the power cord and connect the main Arduino to a computer with Arduino IDE installed using the included cable. If the software has not been downloaded, head to my github repository, click Download as zip and unzip. Go into the folder "Power Mobility", open the file "powermobility.ino". Make sure in "Tools", "Port" is set to Arduino Mega and there is a correct port connection. Click verify and click upload. Wait for 30 seconds. Disconnect the cable and reconnect the power cord.

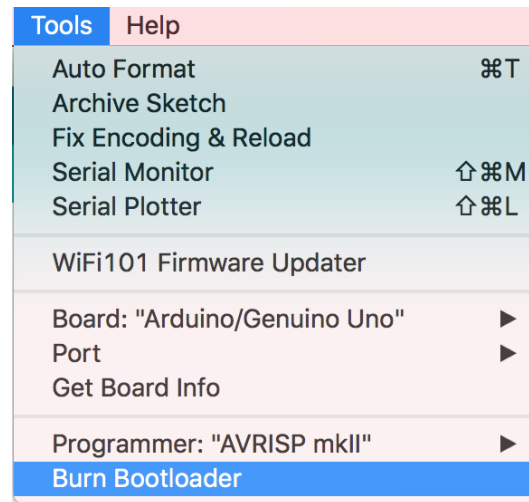


Figure 42: Port Setup for Arduino IDE

12. The touch screen should now be showing the first user configuration screen and the two LEDs for battery and speed profile should be lit up. The device is now ready to be set up and used.
13. Connect your ability switches to the sockets underneath the trays (up to 4).

12.2 Setups and Operation

1. To turn on, click the power button on the included remote controller.
2. To turn off the device operation temporarily, click the power button again on the remote controller.
3. Enter your desired user settings and configurations on the touch screen.
4. If the touch screen says ready, the device is now ready to be operated.
5. If the touch screen says not ready, go to step 9 of Assembly and program set up.
6. Use the ability switches to to drive the vehicle.
7. If the device is not moving, clear the path of any obstacles.

12.3 Troubleshooting and Adjustments

1. If the device is slowing down/not functioning property, check the battery level and charge accordingly.
2. To charge, open the lid per Joseph Caruso's instructions, and disconnect the current cable from the battery. Plug in the charger. Note: the device will not be operational when it is being charged.
3. If any of the parts fail to work:

- (a) If the buzzer is not making any sounds, the ultrasonic sensors or the buzzer are malfunctioning. Check to see if they are connected properly following the assembly and program setup guide and the pinout table.
- (b) If the device is not moving, the ability switches or the motor controllers might be at fault. Check their wiring following the assembly and program setup guide and the pinout table.
- (c) If the remote controller is not working, make sure you are pointing directly behind the vehicle as that is where the IR receiver diode is. Else, check the wiring of the receiver diode and the battery of the remote controller.

12.4 Storage and Maintenance

1. The battery life of the whole system is approximately 45 minutes of constant use. To ensure the batteries are not drained, when not in use, completely disconnect the power plug from the battery. When not in use, the battery can be recharged. For safety purposes, do not leave the battery plugged in to charge for an extended amount of time.
2. Avoid contact with water and other liquids, as the system is not water resistant.
3. During storage, battery and the two micro-controllers need to be unplugged for safety purposes.

13 Conclusion

13.1 Discussion

After two terms of hard and consistent work, a device based on the Wild Thing[®] toy which can serve as a power mobility training and assessment tool was designed and developed. The system, which closely meets the design requirements and specifications planned out by the Kevin G. Langan School at the Center for Disability Services, solves an unfortunate problem faced by many disabled people in the United States.

13.2 Problem

In the United States, in order to get a prescription for a manually controlled and customized wheelchair, a person would need to prove that their health limits their mobility and that they can operate the wheelchair competently. However, commercial wheelchairs are very expensive, usually costing more than \$10000, which makes getting access to one for training impractical for most young disabled children.

13.3 Solutions and Implementation

Hence, the goal of this senior project is to adapt a low-cost Fisher-Price ride-on toy called the Wild Thing[®] to serve as a mobility assessment and training tool, which can take in inputs from standard ability switches to control the motors both indoor and outdoor. Different set and combinations of ability switches can be tailored to the capabilities of a specific student. The overall cost of the control interface to be just over \$592.89, which includes the purchasing of the Wild Thing[®] toy for \$250 (which was donated to Union College by the Center For Disability Services). This price tag is much more affordable

than the usual \$10000 cost. While not replacing traditional wheelchairs, the device can help lower the cost of entry to power mobility, which provides young disabled children with a means of autonomous movement and sufficient body support. By lowering the entry cost, my project can help enhance quality of life by enabling occupation, improving self-esteem and facilitating social interactions. Especially for young children, which are my targeted customers, studies have shown that powered mobility can promote independence and prevent functional limitations and disabilities that they might encounter. In addition, the ability to move autonomously has positive influence on their “self-awareness, emotional attachment, spatial orientation and visual/vestibular integration” [8] as well as “personality traits like motivation and initiation” [7]

This is an interdisciplinary project, in which I modified the control interface of the Wild Thing and Joseph Caruso from the Mechanical Engineering department constructed flexible seating customizations. The entire project is developed and designed to be tested and used at the Kevin G. Langan School at the Center for Disability Services, under the guidance of Mr. Jim Luther and Mrs. Gillian Whelan. The user interface options can be tailored to the capabilities of a specific young child with disabilities through a variety of different ability switches, which can be used to control the motors.

13.4 Results

The first iteration of the project, which is broken down in details by this paper, met or exceeded all but one of the design requirements as outlined in section 4. The device is able to interface any different set of ability switches through a set of 4 audio sockets, and thus can be adjustable to specific user needs. Inputs from these ability switches can be used to produce individual outputs for each motor, as they are independently powered. Within a touch screen mounted on the back of the device, users can configure the device to how they want it to be: the number of inputs being used, how the inputs should control the motor, the maximum speed of the vehicle and enabling/disabling safe mode. Safe mode is the integrated use of two arrays of proximity sensors to detect and avoid walls and obstacles. In addition to the onboard switches, the device can also be controlled through the use of a remote controller, which can either control the motors directly or act as a remote kill switch in case of an emergency. The device is safe in the way that failure in any subsystems results in complete mobility stoppage. The downsides of the project are the overly long and overly complex Arduino program and the battery life, which only lasts 45 minutes of constant use.

The system was designed and implemented with the fact that it is going to be deployed and used full time at the Center for Disability Services full time in mind. All electronic parts are chosen to be plug-and-play without any additional circuitry involved. This way, the users and their supervisors can easily maintain the device and perform trouble shooting as errors occur. The Arduino Mega still has many unused digital (both PWM and external interrupt) and analog ports, which can be used to install upgrades and added features. The one thing that will need to be improved to increase maintainability is to shorten and simplify the code. Right now it is fully documented, but it can arguably hard for people who are unfamiliar with the C and C++ programming language to follow. Included in this paper is also the first draft of a detailed manual which can help users disassemble (for storage as an example) and assemble them back in again. During Spring Term 2018, a detailed manual on how to build the project from scratch will also be produced. This can be helpful for upcoming seniors and summer research students to either upgrade my design or come up with design of their own to solve the problems presented in this paper.

14 Acknowledgement

Professor Cherrice Traver for her patient and wonderful guidance during the two terms of this senior capstone project.

Joseph Caruso and Professor William Keat of the Mechanical Engineering Department for working on the flexible seating customization for the device and completing this ambitious project with me.

Professor Helen Hanson for teaching me how to choose a project, how to write a senior capstone report paper and the engineering's code of ethics during ECE 497.

Union College, the Student Research Grant Committee and the ECBE department for giving me the incredible education foundation, enabling me access to necessary tools and required funding to finish this project successfully.

Gene Davidson for helping me find the correct terminals for my power wires and for printing my poster.

Appendices

A Full Software Codes

```
//Author: Lam Ngo
```

```
#include <TimerOne.h>
```

```
#include <Servo.h>
```

```
#include <IRremote.h>
```

```
#include <Adafruit_NeoPixel.h>
```

```
//define pins
```

```
#define frontTrigPin 4
```

```
#define frontEchoPin 2
```

```
#define backTrigPin 5
```

```
#define backEchoPin 3
```

```
#define buzzer 24
```

```
//define four installed sockets
#define first 11
#define second 12
#define third 13
#define fourth 22

#define firstPresent 23
#define secondPresent 24
#define thirdPresent 25
#define fourthPresent 26

int input_list[] = {11,12,13,22};
//Statiscal LEDs
#define ring_speed 8
#define ring_battery 9
#define ring_status 10

//IR sensors
#define frontAnalog A1
#define backAnalog A2

#define battery_level A2
#define ir 18

//additional constants
#define NUMPIXELS.1 16
#define NUMPIXELS.2 24
```

```
#define echo_int 0

#define TIMER_US 50
#define TICK_COUNTS 4000

volatile long echo_start = 0;
volatile long echo_end = 0;
volatile long echo_duration = 0;
volatile int trigger_time_count = 0;

float speed;
boolean stop = true;
float led;
int max = 40;
int speedprofile = -1;

int forward_speed[][] = {{1540,1580},{1540,1600},{1530,1650}}
int backward_speed[][] = {{1460,1420},{1460,1400},{1460,1350}}

int forward = -1;
int backward = -1;
int left = -1;
int right = -1;

bool run = true;
boolean safemode = true;
int light_state = 0;
```

```
Servo leftMotor ;
Servo rightMotor ;

Adafruit_NeoPixel pixels_speed = Adafruit_NeoPixel(NUMPIXELS2, ring_speed , NEO_GRB +
Adafruit_NeoPixel pixels_battery = Adafruit_NeoPixel(NUMPIXELS2, ring_battery , NEO.C
Adafruit_NeoPixel pixels_tail = Adafruit_NeoPixel(NUMPIXELS1, ring_status , NEO_GRB +

IRrecv irr(ir);
decode_results results ;

int number_of_inputs = 0;
void setup() {
  // initialize inputs
  pinMode(frontTrigPin , OUTPUT);
  pinMode(frontEchoPin , INPUT);

  pinMode(backTrigPin , OUTPUT);
  pinMode(backEchoPin , INPUT);

  pinMode(first ,INPUT_PULLUP);
  pinMode(second ,INPUT_PULLUP);
  pinMode(third ,INPUT_PULLUP);
  pinMode(fourth ,INPUT_PULLUP);

  Timer1.initialize(TIMER_US);
  Timer1.attachInterrupt(timerIsr);
  attachInterrupt(echo_int , echo_interrupt , CHANGE);
```



```
attachInterrupt(ir , decode);

leftMotor.attach(6);
leftMotor.write(90);
rightMotor.attach(7);
rightMotor.write(90);

pixels_speed.begin();
pixels_speed.show();

pixels_battery.begin();
pixels_battery.show();

pixels_tail.begin();
pixels_tail.show();

Serial.begin(9600);

setup_stage();
  time_t t = now();
}

void loop() {
  //update battery level every 10 minutes
  if (minute(now()) - minute(t) >= 10){
    battery_light(analogRead(battery_level));
    t = now();
  }
}
```

```
if (run){
  //neutral state
  setup_stage();
  leftMotor.writeMicroseconds(1510);
  rightMotor.writeMicroseconds(1510);

  //move forward
  if (failsafe() && forward != -1 && digitalRead(forward) == LOW){
    speed = forward_speed[speed_profile][0];

    //slowly accelarating
    while (digitalRead(forward) == LOW){
      if (speed >= forward_speed[speed_profile][1]){
        speed = forward_speed[speed_profile][1];
      }
      leftMotor.writeMicroseconds(speed+2);
      rightMotor.writeMicroseconds(speed);
      speed = speed + 0.0005;

      proportial_light(speed, forward_speed[speed_profile][1] - forward_speed[speed_profile][0]);
    }

    //slow decelerating
    while (speed > 1525){
      leftMotor.writeMicroseconds(speed);
      rightMotor.writeMicroseconds(speed);
      speed = speed - 0.005;
    }
  }
}
```

```
    reset_tail_light ();
}

//move backward
if ( failsafe () && backward != -1 && digitalRead (backward) == LOW){
    speed = backward_speed[speed_profile][0];

    while ( digitalRead (backward) == LOW && !stop){
        if (speed <= backward_speed[speed_profile][1]){
            speed = backward_speed[speed_profile][1];
        }
        leftMotor.writeMicroseconds(speed);
        rightMotor.writeMicroseconds(speed);
        speed = speed - 0.0005;
        proportial_light(speed, backward_speed[speed_profile][0] - backward_speed[spe
    }

    while (speed < 1475){
        leftMotor.writeMicroseconds(speed);
        rightMotor.writeMicroseconds(speed);
        speed = speed + 0.001;
    }

    reset_tail_light ();
}

//move left
```

```
if (failsafe() && left != -1 && digitalRead(right) == LOW){
    float speedL = forward_speed[speed_profile][0];
    float speedR = backward_speed[speed_profile][0];

    while (digitalRead(right) == LOW && !stop){
        if (speedL >= forward_speed[speed_profile][1]){
            speedL = forward_speed[speed_profile][1];
        }

        if (speedR <= backward_speed[speed_profile][1]){
            speedR = backward_speed[speed_profile][1];
        }

        leftMotor.writeMicroseconds(speedL);
        rightMotor.writeMicroseconds(speedR);
        speedL = speedL + 0.002;
        speedR = speedR - 0.002;
        proportional_light(speedL, forward_speed[speed_profile][1] - forward_speed[speed_profile][0]);
    }

    while (speedL > 1525 && speedR < 1475){
        leftMotor.writeMicroseconds(speedL);
        rightMotor.writeMicroseconds(speedR);
        speedL = speedL - 0.001;
        speedR = speedR + 0.001;
    }

    reset_tail_light();
}
```

```
}

//move right
if (failsafe() && right != -1 && digitalRead(left) == LOW){
    float speedL = backward_speed[speed_profile][0];
    float speedR = forward_speed[speed_profile][0];

    while (digitalRead(right) == LOW && !stop){
        if (speedR >= forward_speed[speed_profile][1]){
            speedR = forward_speed[speed_profile][1];
        }

        if (speedL <= backward_speed[speed_profile][1]){
            speedL = backward_speed[speed_profile][1];
        }

        leftMotor.writeMicroseconds(speedL);
        rightMotor.writeMicroseconds(speedR);
        speedR = speedR + 0.002;
        speedL = speedL - 0.002;

        proportial_light(speedL, backward_speed[speed_profile][0] - backward_speed[speed_profile][1]);
    }

    while (speedR > 1525 && speedL < 1475){
        leftMotor.writeMicroseconds(speedL);
        rightMotor.writeMicroseconds(speedR);
        speedR = speedR - 0.001;
```

```
        speedL = speedL + 0.001;
    }

    reset_tail_light();
}
}

boolean failsafe(){
    if (leftMotor.read() == 0){
        Serial.write("Not Ready");
        return false;
    }

    if (rightMotor.read() == 0){
        Serial.write("Not Ready");
        return false;
    }

    if (digitalRead(firstPresent) == HIGH || digitalRead(secondPresent) == HIGH
        || digitalRead(thirdPresent) == HIGH || digitalRead(fourthPresent) == LOW){
        Serial.write("Not Ready");
        return false;
    }

    Serial.write("Ready");
    return true;
}
```

```
//Read from serial and setup
void setup_stage(){
    if (Serial.avialable() > 0){
        String message = Serial.readString();

        String v = message;
        int count = 1;

        while (v.indexOf("|") != -1){
            if (count == 1){
                number_of_inputs = int(v.substring(0,v.indexOf("|")));
            } else if (count == 2){
                speed_profile = int(v.substring(0,v.indexOf("|")));
                if (speed_profile == 1){
                    speed_profile = 0;
                } else if (speed_profile == 2){
                    speed_profile = 1;
                } else{
                    speed_profile = 2;
                }
            }
            speed_light(speed_profile);
        } else if (count == 3){
            r = int(v.substring(0,v.indexOf("|")));
            if (r == "On"){
                safemode = true;
            } else {
                safemode = false;
            }
        }
    }
}
```

```
    }
} else if (count == 4){
    String a = v.substring(0,v.indexOf("|"));
    for (int i = 0; i < number_of_inputs; i++){
        if (a.charAt(i) == 'F'){
            forward = input_list[i];
        } else if (a.charAt(i) == "B"){
            backward = input_list[i];
        } else if (a.charAt(i) == "L"){
            left = input_list[i];
        } else if (a.charAt(i) == "R"){
            right = input_list[i];
        }
    }
}
}
}
}

void timerIsr(){
    trigger_pulse();
}

//incrementally light up the tail light neopixel
void proportial_light(int currentspeed, int max, int direction){
    int to_lit = (currentspeed/max)*NUMPIXELS_1;
    if (direction == 0){
        pixels_tail.setPixelColor(to_lit, pixels_tail.Color(0,150,0));
    }
}
```



```
    } else if (direction == 1){
        pixels_tail.setPixelColor(to_lit , pixels_tail.Color(150,0,0));
    }
    pixels_tail.show();
}

//turn off neopixel
void reset_tail_light(){
    for (int i = 0; i < NUMPIXELS_1; i++){
        pixels_tail.setPixelColor(i, pixels_tail.Color(0,0,0));
    }
    pixels_tail.show();
}

//light up speed light
void speed_light(int speed_profile){
    if (speed_profile == 0){
        int color[] = {0,0,255};
    } else if (speed_profile == 1){
        int color[] = {0,150,0};
    } else if (speed_profile == 2) {
        int color[] = {238,130,238}
    }
    for (int i = 0; i < NUMPIXELS_2; i++){
        pixels_speed.setPixelColor(i, pixels_speed.Color(color[0],color[1],color[2]));
    }
    pixels_speed.show();
}
```

```
void battery_light(int batterylevel){
    if (batterylevel > 4.47){
        for (int i = 0; i < NUMPIXELS2; i++){
            pixels_battery.setPixelColor(i, pixels_speed.Color(0,150,0));
        }
        pixels_battery.show();
    } else if (batterylevel > 4.32 and batterylevel <= 4.47){
        for (int i = 0; i < NUMPIXELS2; i++){
            pixels_battery.setPixelColor(i, pixels_speed.Color(255,255,0));
        }
        pixels_battery.show();
    } else if (batterylevel <= 4.32){
        for (int i = 0; i < NUMPIXELS2; i++){
            pixels_battery.setPixelColor(i, pixels_speed.Color(150,0,0));
        }
        pixels_battery.show();
    }
}
```

```
//blink red
```

```
void tail_blink(int state){
    if (state == 0){
        for (int i = 0; i < NUMPIXELS1; i++){
            pixels_tail.setPixelColor(i, pixels_tail.Color(0,0,0));
        }
        pixels_tail.show();
        delay(500);
    }
}
```

```
    } else{
        for (int i = 0; i < NUMPIXELS-1; i++){
            pixels_tail.setPixelColor(i, pixels_tail.Color(150,0,0));
        }
        pixels_tail.show();
        delay(500);
    }
}
```

```
void trigger_pulse(){
    static volatile int state = 0;

    if (!(--trigger_time_count)){
        trigger_time_count = TICK_COUNTS;
        state = 1;
    }

    switch (state) {
        case 0: break;
        case 1:
            digitalWrite(frontTrigPin ,LOW);
            delayMicroseconds(5);
            digitalWrite(frontTrigPin , HIGH);
            state = 2;
            break;
        case 2:
        default:
            digitalWrite(frontTrigPin ,LOW);
```

```
        state = 0;
        break;
    }
}

//decode upcoming signal from IR sensors
void decode(){
    if (irr.decode(&results)){
        if (results.value == "C573E684" && run){
            run = false;
        } else if (results.value == "C573E684" && !run){
            run = true;
        }
    }
}

void echo_interrupt(){
    switch(digitalRead(frontEchoPin)){
        case HIGH:
            echo_end = 0;
            echo_start = micros();
            break;
        case LOW:
            echo_end = micros();
            echo_duration = echo_end - echo_start;
            if (echo_duration/58 == 0){
                stop = true;
                Serial.write("Not Ready");
            }
        }
    }
}
```

```
}
if (safemode){
  if (echo_duration/58 < 80 && echo_duration/58 != 0) {
    //distance in cm
    if (echo_duration/58 < 50 && echo_duration/58 != 0){
      // distance < 50 and stop the device
      tone(buzzer,1000);
      if (light_state == 0){
        light_state = 1;
        tail_blink(light_state);
      } else{
        light_state = 0;
        tail_blink(light_state);
      }
      stop = true;
    }else{
      // < 80 sound the alarm to alert
      tone(buzzer,500);
      stop = false;
    }
  } else{
    //do nothing
    noTone(buzzer);
    stop = false;
  }
}
break;
}
```

```
}
```

B Touch Screen Code

```
//Author: Lam Ngo

#include "GUIslice.h"
#include "GUIslice_ex.h"
#include "GUIslice_drv.h"

// Defines for resources

// Enumerations for pages, elements, fonts, images
enum {E_PG_1,E_PG_2, E_PG_3};
enum {E_ELEM_BOX,E_ELEM_BTN_1, E_ELEM_BTN_2};
enum {E_FONT_BTN};

bool    m_bQuit = false;

// Instantiate the GUI
#define MAX_PAGE 3
#define MAX_FONT 1
#define MAX_ELEM_PG_2 22
#define MAX_ELEM_PG_1 14
#define MAX_ELEM_PG_3 2

// Define the maximum number of elements per page
// - To enable the same code to run on devices that support storing
//   data into Flash (PROGMEM) and those that don't, we can make the
```

```

//   number of elements in Flash dependent upon GSLC_USE_PROGMEM
// - This should allow both Arduino and ARM Cortex to use the same code
#if (GSLC_USE_PROGMEM)
    #define MAX_ELEM_PG_2_PROG    22                // # Elems
    #define MAX_ELEM_PG_1_PROG    14
    #define MAX_ELEM_PG_3_PROG    2
#else
    #define MAX_ELEM_PG_1_PROG    0                // # Elems i
    #define MAX_ELEM_PG_2_PROG    0                // # Elems i
    #define MAX_ELEM_PG_3_PROG    0                // # Elems i
#endif

#define MAX_ELEM_PG_1_RAM        MAX_ELEM_PG_1 - MAX_ELEM_PG_1_PROG // # Elems in RAM
#define MAX_ELEM_PG_2_RAM        MAX_ELEM_PG_2 - MAX_ELEM_PG_2_PROG // # Elems in RAM
#define MAX_ELEM_PG_3_RAM        MAX_ELEM_PG_3 - MAX_ELEM_PG_3_PROG // # Elems in RAM

gslc_tsGui                        m_gui;
gslc_tsDriver                    m_drv;
gslc_tsFont                      m_asFont[MAX_FONT];
gslc_tsPage                      m_asPage[MAX_PAGE];

gslc_tsElem                      m_asFirstElem[MAX_ELEM_PG_1_RAM]; // Storage for all el
gslc_tsElemRef                  m_asFirstElemRef[MAX_ELEM_PG_2];

gslc_tsElem                      m_asSecondElem[MAX_ELEM_PG_2_RAM];
gslc_tsElemRef                  m_asSecondElemRef[MAX_ELEM_PG_2];

gslc_tsElem                      m_asThirdElem[MAX_ELEM_PG_3_RAM];
gslc_tsElemRef                  m_asThirdElemRef[MAX_ELEM_PG_2];

```

```
// Define debug message function
static int16_t DebugOut(char ch) { Serial.write(ch); return 0; }

// Button callbacks
bool First(void* pvGui,void *pvElemRef,gslc_teTouch eTouch,int16_t nX,int16_t nY)
{
// if (eTouch == GSLC_TOUCH_UP_IN) {
//   gslc_SetPageCur(&m_gui,E_PG_2);
// }
  gslc_SetPageCur(&m_gui,E_PG_2);
  return true;
}

// Button callbacks
bool Second(void* pvGui,void *pvElemRef,gslc_teTouch eTouch,int16_t nX,int16_t nY)
{
// if (eTouch == GSLC_TOUCH_UP_IN) {
//   gslc_SetPageCur(&m_gui,E_PG_2);
// }
  gslc_SetPageCur(&m_gui,E_PG_3);
  return true;
}

// Button callbacks
bool Third(void* pvGui,void *pvElemRef,gslc_teTouch eTouch,int16_t nX,int16_t nY)
```



```
{
//  if (eTouch == GSLC_TOUCH_UP_IN) {
//      gslc_SetPageCur(&m_gui,E_PG_2);
//  }
    gslc_SetPageCur(&m_gui,E_PG_1);
    return true;
}

bool InitOverlays(){
    gslc_tsElemRef* pElemRef = NULL;

    gslc_PageAdd(&m_gui,E_PG_1,m_asFirstElem,MAX_ELEM_PG_1_RAM,m_asFirstElemRef,MAX_ELE
    gslc_PageAdd(&m_gui,E_PG_2,m_asSecondElem,MAX_ELEM_PG_2_RAM,m_asSecondElemRef,MAX_E
    gslc_PageAdd(&m_gui,E_PG_3,m_asThirdElem,MAX_ELEM_PG_3_RAM,m_asThirdElemRef,MAX_ELE

    gslc_SetBkgndColor(&m_gui,GSLC_COL_GRAY_DK2);

// PAGE: 1

// Create background box
gslc_ElemCreateBox_P(&m_gui,100,E_PG_1,20,50,280,150,GSLC_COL_WHITE,GSLC_COL_BLACK,

//      // Create title
//  gslc_ElemCreateTxt_P(&m_gui,101,E_PG_1,10,10,310,40,"Configuration",&m_asFont[2],
//      GSLC_COL_WHITE,GSLC_COL_BLACK,GSLC_COL_BLACK,GSLC_ALIGN_MID_MID,false,fal

// Create Quit button with text label
gslc_ElemCreateBtnTxt_P(&m_gui,101,E_PG_1,120,100,80,40,"GO",&m_asFont[0],
    GSLC_COL_WHITE,GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK4,GSLC_COL_BLUE_DK2,
```

```
GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, &First, NULL);

// PAGE: 2
// Background flat color
//  gslc_SetBkgndColor(&m_gui, GSLC_COL_GRAY_DK2);

// Create background box
gslc_ElemCreateBox_P(&m_gui, 102, E_PG_2, 10, 50, 300, 150, GSLC_COL_WHITE, GSLC_COL_BLACK,

gslc_ElemCreateTxt_P(&m_gui, 201, E_PG_2, 20, 60, 10, 10, "Input 1", &m_asFont[1], // E_FON
    GSLC_COL_YELLOW, GSLC_COL_BLACK, GSLC_COL_BLACK, GSLC_ALIGN_MID_LEFT, false, true

// Create Quit button with text label
gslc_ElemCreateBtnTxt_P(&m_gui, 222, E_PG_2, 70, 60, 50, 20, "Forward", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 223, E_PG_2, 130, 60, 50, 20, "Backward", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 202, E_PG_2, 190, 60, 50, 20, "Left", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 203, E_PG_2, 250, 60, 50, 20, "Right", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);
```

```
gslc_ElemCreateTxt_P(&m_gui,204,E_PG_2,20,90,10,10,"Input 2",&m_asFont[1], // E_FON
    GSLC_COL_YELLOW,GSLC_COL_BLACK,GSLC_COL_BLACK,
    GSLC_ALIGN_MID_LEFT,false,true);
// Create Quit button with text label
gslc_ElemCreateBtnTxt_P(&m_gui,205,E_PG_2,70,90,50,20,"Forward",&m_asFont[0],
    GSLC_COL_WHITE,GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK1,GSLC_ALIGN_MID_MID,true,true,NULL,NULL);

gslc_ElemCreateBtnTxt_P(&m_gui,206,E_PG_2,130,90,50,20,"Backward",&m_asFont[0],
    GSLC_COL_WHITE,GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK1,GSLC_ALIGN_MID_MID,true,true,NULL,NULL);

gslc_ElemCreateBtnTxt_P(&m_gui,207,E_PG_2,190,90,50,20,"Left",&m_asFont[0],
    GSLC_COL_WHITE,GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK1,GSLC_ALIGN_MID_MID,true,true,NULL,NULL);

gslc_ElemCreateBtnTxt_P(&m_gui,208,E_PG_2,250,90,50,20,"Right",&m_asFont[0],
    GSLC_COL_WHITE,GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK1,GSLC_ALIGN_MID_MID,true,true,NULL,NULL);

gslc_ElemCreateTxt_P(&m_gui,209,E_PG_2,20,120,10,10,"Input 3",&m_asFont[1], // E_FON
    GSLC_COL_YELLOW,GSLC_COL_BLACK,GSLC_COL_BLACK,GSLC_ALIGN_MID_LEFT,false,true);
// Create Quit button with text label
gslc_ElemCreateBtnTxt_P(&m_gui,210,E_PG_2,70,120,50,20,"Forward",&m_asFont[0],
    GSLC_COL_WHITE,GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2,GSLC_COL_BLUE_DK1,GSLC_ALIGN_MID_MID,true,true,NULL,NULL);
```

```
gslc_ElemCreateBtnTxt_P(&m_gui, 211, E_PG_2, 130, 120, 50, 20, "Backward", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4
    , GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 212, E_PG_2, 190, 120, 50, 20, "Left", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4
    , GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 213, E_PG_2, 250, 120, 50, 20, "Right", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateTxt_P(&m_gui, 214, E_PG_2, 20, 150, 10, 10, "Input 4", &m_asFont[1], // E_FC
    GSLC_COL_YELLOW, GSLC_COL_BLACK, GSLC_COL_BLACK
    , GSLC_ALIGN_MID_LEFT, false, true);

// Create Quit button with text label
gslc_ElemCreateBtnTxt_P(&m_gui, 215, E_PG_2, 70, 150, 50, 20, "Forward", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 216, E_PG_2, 130, 150, 50, 20, "Backward", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 217, E_PG_2, 190, 150, 50, 20, "Left", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);
```

```
gslc_ElemCreateBtnTxt_P(&m_gui, 218, E_PG_2, 250, 150, 50, 20, "Right", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, NULL, NULL);

gslc_ElemCreateBtnTxt_P(&m_gui, 219, E_PG_2, 250, 175, 100, 50, "Next", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, &Second, NULL);

//PAGE 3:

// Create background box
gslc_ElemCreateBox_P(&m_gui, 300, E_PG_3, 10, 50, 300, 150, GSLC_COL_WHITE, GSLC_COL_BLACK,

// Create Quit button with text label
gslc_ElemCreateBtnTxt_P(&m_gui, 301, E_PG_3, 120, 100, 80, 40, "READY", &m_asFont[0],
    GSLC_COL_WHITE, GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK4,
    GSLC_COL_BLUE_DK2, GSLC_COL_BLUE_DK1, GSLC_ALIGN_MID_MID, true, true, &Third, NULL);

return true;
}

void setup()
{
    Serial.begin(9600);
    gslc_InitDebug(&DebugOut);

    if (!gslc_Init(&m_gui, &m_drv, m_asPage, MAX_PAGE, m_asFont, MAX_FONT)) { return; }
```

```
if (!gslc_FontAdd(&m_gui,E_FONT_BTN,GSLC_FONTREF_PTR,NULL,1)) { return; } // m_a
//if (!gslc_FontAdd(&m_gui,E_FONT_TXT,GSLC_FONTREF_PTR,NULL,1)) { return; } // m
//if (!gslc_FontAdd(&m_gui,E_FONT_TITLE,GSLC_FONTREF_PTR,NULL,1)) { return; } // m

InitOverlays();

gslc_SetPageCur(&m_gui,E_PG_1);

}

void loop()
{
  // Periodically call GUIslice update function
  gslc_Update(&m_gui);

  // In a real program, we would detect the button press and take an action.
  // For this Arduino demo, we will pretend to exit by emulating it with an
  // infinite loop. Note that interrupts are not disabled so that any debug
  // messages via Serial have an opportunity to be transmitted.
  if (m_bQuit) {
    gslc_Quit(&m_gui);
    while (1) { }
  }
}
```

C LED PWM Code

```
//Author: Lam Ngo
int ledPin = 9;    // LED connected to digital pin 9
int aSwitch = 6;

void setup() {
  pinMode(aSwitch, INPUT_PULLUP)
  int fadeValue = 0;
}

void loop() {
  while (digitalRead(aSwitch) == LOW){
    // fade in from min to max in increments of 5 points:
    if (fadeValue >= 255){
      fadeValue = 255;
    }
    analogWrite(ledPin, fadeValue);
    delay(30);
    feadeValue += 5;
  }

  while (digitalRead(aSwitch) == HIGH){
    if (fadeValue <= 0){
      fadeValue = 0;
    }

    analogWrite(ledPin, fadeValue);
    delay(30);
    feadeValue -= 5;
  }
}
```

}

}

D Student Research Grant

Union College Senior Capstone Project Powered Mobility For Young Disabled Students Designed for the Kevin G. Langan School At the Center for Disability Services

Lam Ngo, Student ID: 2463971
Advisor: Professor Cherrice Traver

October 29, 2017

1 Introduction

In the United States, in order to be granted a fully customized powered wheelchair, a person needs to demonstrate to the committee that she is able to operate the device competently and safely [1]. It is a fair rule, but the problem lies in the fact that buying a wheelchair for practice is completely out of the question for most people, especially young disabled students. Base models of modern wheelchair, which are used for mobility assessment, are awfully expensive, costing more than \$10000 per unit, that is before adding any customizations. To counter this, disability services around the states try to borrow and lend wheelchairs to each other. However, this is only an impermanent solution, as the time frame in which a center can use a wheelchair is often only one week. In addition, the devices come without any customizations, and the counselors and physical trainers have to perform the customization process from the ground up every time they receive a new wheelchair.

Thus, my senior capstone project is to adapt a Fisher-Price toy called Wild Thing[®], so that it can serve as a power mobility assessment tool for younger students, while being low-cost and easy to use. Power mobility means the use of powered wheelchairs and ride-on toys, which run on electricity and provide efficient and autonomous mobility and body support for individuals with limited ability to walk. While my device is not supposed to be a practical replacement for a prescribed wheelchair, it can be utilized as one for young children to learn how to drive. This project is a interdisciplinary one, in which I, a Computer Engineering major, will handle adapting the control interface of the Wild Thing, and Joseph Carruso, a Mechanical Engineering major, will create and construct flexible seating customizations. The entire project is developed and designed to be tested and used at the Kevin G. Langan School At the Center For Disability Services, under the strict supervision of Mr. Jim Luther and Mrs. Laura A Shemo.

2 Goals

The overall goal of this senior capstone project is to be able to adapt and construct a control interface for the toy Wild Thing[®], which can take in inputs from standard joysticks, buttons, or other sensor inputs to control the motors both indoor and outdoor. These different control inputs and schemes can be tailored to the capabilities of a specific student. Also, as mentioned above, the device needs to be safe, intuitive, and relatively low-cost. I am approximating the overall cost of the control interface to be just over \$550, which includes the purchasing of the Wild Thing[®] toy for \$250 (which was donated to Union College by the Center For Disability Services). This price tag is much more affordable than the usual \$10000 cost.

However, due to the limited time-frame for a senior project, I have decided to lower the scale of the project. Instead of being able to interface all kinds of different inputs, for now the device will just interface up to four different push-button ability switches. In addition, the mapping of the inputs and their functionalities are tailored to two specific children at the Kevin G. Langan School, who have been carefully chosen to test the project. I will include a detailed manual on how to replicate my project, and students from next years can continue to develop it into a fully functional model and replacement for a prescribed wheelchair.

3 Design Methodology

The Wild Thing[®] toy was chosen as it is small in size (but still can support up to 44kg), is budget friendly, and has two independently powered motors. In addition, it was built specifically for outdoor performance, such as on pavement and on grass. Its large, rugged tires and forward speeds of up to 5 mph are able to handle most of the surfaces, and there is a built-in stability sensor to help prevent tipping.

After several conversations with my customers over at the Center for Disability Services, we agreed upon a set of design specifications that my adapted Wild Thing control interface is required to be able to accomplish.

1. Size constraints: main processing units need to fit inside a 16 cm by 8 cm area under the chair of the Wild Thing[®].
2. Movement: the vehicle must be able to move forward, backward, steer left, right and brake. Speed acceleration needs to be smooth, and speed can be modified using a potentiometer.
3. Inputs: the vehicle must be able to interface a variety of standard ability switches, whose signals are transferred through a 3.5mm jack plug. These inputs are mapped to the four directions of movement, and can be reconfigured at any time, using a LCD touch screen. The total number of inputs and their types are also touchconfigurable with the touch screen.
4. Debugging: The vehicle must display to users and their supervisors vital information about the system: battery level and current speed.
5. Safety:
 - In a special mode called safety mode, which can be turned on and off via the LCD screen and a physical button, the vehicle must be able to avoid walls and obstacles.
 - Failure in any subsystems must result in mobility stoppage.
 - A Remote controller, which can turn the device on and off and remotely kill the device, is required.
6. Energy: the system needs to provide enough power for at least one and a half hour of operation, both indoor and outdoor.

The design specifications above are chosen carefully to ensure that the operation of the device is safe, intuitive and modular for young disabled students and children. They were also based on the functionalities and safety features of a full-fledged prescribed wheelchair. Figure 1 below is the high level block diagram of my design,

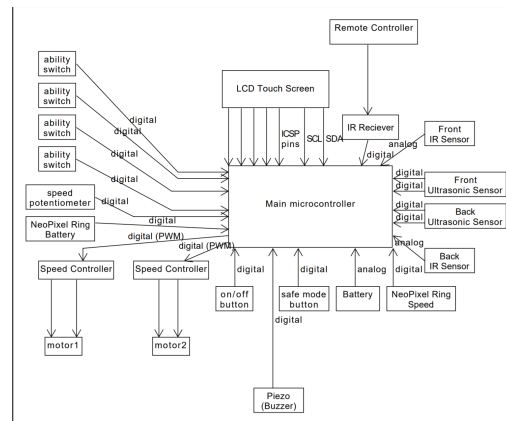


Figure 1: High Level Block Diagram

which is tailored to directly satisfy the design requirements outlined above. The processing unit of the entire system is a micro-controller. I decided to do this on the Arduino platform, for ease of development and of debugging. The micro-controller, which was chosen, is an Arduino Mega 2560 Rev3. It is perfectly suitable

for our project thanks to having 54 digital inputs, 16 analog inputs, and a large memory capacity. Though on the larger side of Arduino micro-controllers, it should fit nicely into the designated 16 cm by 8 cm spot. The inputs can be up to 4 ability switches, which act as digital inputs to the micro-controller. The two motors are powered and controlled by two Spark Rev motor controllers, which replace the current motor controllers inside the Wild Thing, which do not support Arduino integration. These Spark motor controllers have been proven to work well with the Arduino platform. The inputs to the two Rev motor are digital pulse width modulation signals, and can be generated using our ability switch inputs. The switches will be mapped, using software, to control the two motors to create four direction of movements: Forward, Backward, Left, and Right.

A touchscreen is utilized to reconfigure the inputs mapping, and increase or decrease the speed. A 2.8 inch capacitive LCD touchscreen called TFT Touch Shield is chosen, as it has been proven among the Arduino community to be the most intuitive and best performing touch screen for micro-controllers. The touch screen communicates with micro-controllers using designated ICSP pins (for SPI communications) and I2C communications. One good thing is that the touch screen itself has its own controller with RAM buffering, thus most of the processing is done on the LCD, which helps to reduce the load on the micro-controller.

For debugging, to display vital information about the system, I use two different NeoPixel Rings from Adafruit. They are essentially just an array of LEDs, but only require 1 digital input. One ring is used to display the battery level, which are recorded using a battery tester circuit with an analog input to the micro-controller, and one ring is for the speed, which is an information kept inside the software. Remotely controlling the device is done, in the early stages of the project, using an IR Receiver diode and an IR remote controller. However, since it is Infrared, its performance outdoor is not going to be optimal. Thus, if time allows, I would want to implement this future using Xbee, which provides communication through radio signals.

Safety mode is turned on using a button, or using the remote controller. It is implemented using two arrays of proximity sensors, one on the front and one on the back. Each array consists of an ultrasonic distance sensor, specifically HC-SR04 with a working range from 2cm to 400cm and an Infrared distance sensor, Sharp GP2Y0A21YK with a working range from 10cm to 80cm. These two types of sensors are integrated together in order to improve the overall accuracy. Whenever the sensor arrays detect an obstacle, there are two scenarios that would happen. If it is only 100 centimeters from the obstacle, a piezo buzzer will be used to alert the users and the supervisors. If it is only 30 centimeters from the obstacle, the device will smoothly halt. In software, whenever an error occurs, the whole system will be shutdown for safety purposes.

Detailed information about cost, and where to buy the parts are included in my budget section.

4 Reasons for Funding

This is my passion project, and I believe that were it to be successful, the adapted Wild Thing would provide many benefits and possibilities for the young disabled students at the Kevin G. Langan school. The device will give them a platform to practice, introduce them to power mobility and to show that they can operate a prescribed wheelchair competently and eventually granted one. This would allow them to move freely within their home and community, to maintain their safety and to conserve energy. Power mobility can enhance the quality of life by enabling occupation, improving self-esteem and facilitating social interaction. Especially with children who are still developing, studies have shown that the ability to move independently has positive influence on their self-awareness, emotional attachment, spatial orientation, fear of heights, and visual/vestibular integration [2] as well as personality traits like motivation and initiation [3].

References

- (1) NYS State Law NEW YORK STATE MEDICAID WHEELED MOBILITY EQUIPMENT GUIDELINES., https://www.emedny.org/ProviderManuals/DME/PDFS/DME_Wheeled_Mobility_Equipment_Guidelines.pdf, Accessed: 2017-06-08.
- (2) Furumasu, J., *Pediatric powered mobility: Developmental perspectives, technical issues, clinical approaches*; RESNA/Rehabilitation Engineering and Assistive Technology Society of North America: 1997.
- (3) Campos, J. J.; Anderson, D. I.; Barbu-Roth, M. A.; Hubbard, E. M.; Hertenstein, M. J.; Witherington, D. *Infancy* **2000**, *1*, 149–219.

5 Budget Listing

Figure 2 below details the overall cost for the all the components needed for my projects. Since the Wild Thing[®] toy is donated to Union College by the Center for Disability Services, it is not included in the final cost.

Type	Part Number/Description	Link	Price	Note	Quantity	Total
Microcontroller	Arduino Mega 2560 Rev3	Sparkfun	\$38.50		1	\$38.50
LCD Touch Screen	TFT Touch Shield With Capacitive Touch	Adafruit	\$44.95		1	\$44.95
IR Receiver	IR Receiver Diode	Sparkfun	\$1.95		1	\$1.95
Buzzer	Piezo Buzzer	Adafruit	\$1.95		1	\$1.95
Distance Sensor	HC-SR04	Sparkfun	\$3.95		2	\$7.90
	Sharp GP2Y0A21YK	Sparkfun	\$14.95		2	\$27.90
LED arrays	Neopixel Ring	Adafruit	\$16.95		2	\$33.90
Speed Controller	Spark Motor Controller	RevRobotics	\$45		2	\$90
Jumper Wires	Female/Female	Adafruit	\$3.95		2	\$7.90
Ability Switches	PushButton Switches, set of 3	Enabling devices	\$65.95		1	\$65.95
Ride-on Toy	Wild Thing	Jet.com	\$249.99	This was donated to Union College.	1	\$249.99

Figure 2: Detailed budget of the project

- Total component Cost: \$320.90
- Total Shipping cost: \$22.63
 - From vendor Adafruit: \$7.17.
 - From vendor Sparkfun: \$0 since order is more than \$75.
 - From vendor RevRobotics: \$5.46.
 - From vendor Enabling Devices: \$10
- **Final cost: \$343.53.**

The importance of each component is mentioned in my Design Methodology section, but here is a brief explanation for each one.

- Arduino Mega 2560 Rev3 (Reusable): the main processing unit of the system, it drives the motor and takes in inputs.
- TFT Touch Shield (Reusable): it handles getting user inputs.
- IR receiver (Reusable): it enables the communication between a remote and the microcontroller.
- Piezo buzz (Reusable): it is used to play sound when the device gets too close to an obstacle.
- HC-SR04 and Sharp GP2Y0A21YK (Reusable): they are used to detect when the device gets too close to an obstacle.
- NeoPixel Ring (Reusable): it is used to display battery and speed information.
- Spark Motor Controllers (Reusable): they are used to control the motors.
- Jumper wires (Reusable): they are used to connect components to the microcontroller.
- Ability Switches (Reusable): they are used as input for disabled students.
- The Wild Thing[®] (Reusable): it is adapted to be used as a mobility assessment tool.

E Departmental Presentation

3/24/2018

POWERED MOBILITY FOR YOUNG STUDENTS.

Designed for The Kevin G. Langan School at The Center for Disability Services
 LAM NGO – ECE 499
 Project Supervisor: Professor Cherie Traver

PROBLEM DESCRIPTION

- Competency in driving is needed to be granted a wheelchair.
- Normal expenses: > \$10,000.
- Difficult for many young disabled students to get access to one for training.

• Base model: \$14,000.



GOALS

- Solution: Create a responsive, safe, easy-to-use, and low-cost vehicle to serve as a powered mobility assessment and training tool.
- Implementation: Adapt the mobility control interface options of a Fisher Price toy; The Wild Thing.

WHY THE WILD THING?

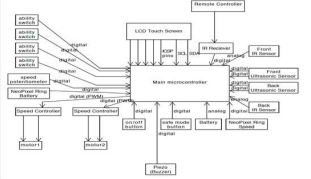
- Small sized and budget-friendly.
- The seat can support up to 44kg/97lbs.
- Two independently powered motors.
- Still in market, so we can buy one easily.



DESIGN SPECIFICATIONS

1. Able to control motors for 4 directions of movement.
2. Able to interface a variety of ability switches as inputs.
3. Able to be adjusted to specific needs.
4. Able to be modified and reconfigured on the go.
5. Show battery and speed information for debugging
6. Safety: Obstacle avoidance, emergency button and remote kill switch.
7. Safety: Failure in any parts results in mobility stoppage.
8. Needs to work outdoors!
9. Affordable, planned budget to be \$500.


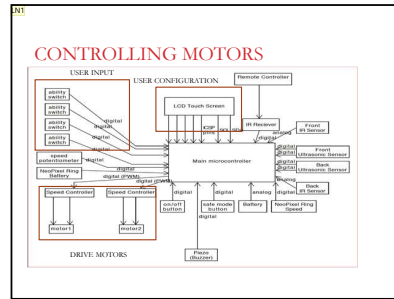
DESIGN OVERVIEW



3/24/2018

Microcontroller

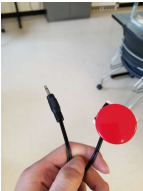
- **Three Sub-functions:**
 - **Controlling motors:**
 - Receive configurations from users.
 - Configure inputs and drive motors.
 - **Ensuring safety:**
 - Safe-mode: obstacle avoidance.
 - Remote kill switch.
 - Fail-safe default.
 - **Display statistical information:**
 - Battery information.
 - Speed information.


Ability Switches

- Interact with electronic devices.
- Many different types.
- Used in the project: Push button
- Signals are transferred through 3.5 mm jack.

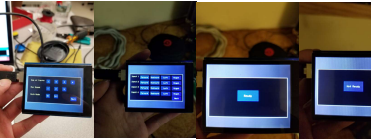
⇒ Need to figure out a way for the Mega to receive input signals.



Ability Switches Adapter



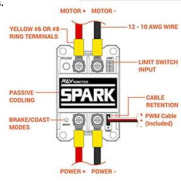
TFT Touch Screen



- **Controlled by an Arduino Uno.**
 - Not enough ports on the Mega.
 - Independent and native support.
 - Serial communication between the two.

Spark Motor Controller

- Controls the two motors.
- 5.5-24V input Voltage.
- Informational LEDs.
- Accepts servo-identical PWM signals.



Slide 8

LN1 Lam Ngo, 2/23/2018

3/24/2018

CONTROLLING MOTORS

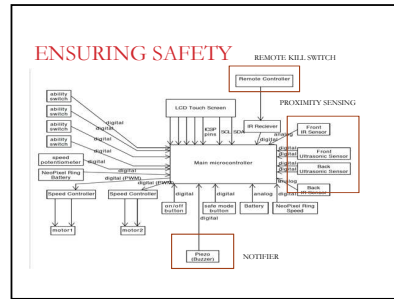
Control Motor for 4 directions of movements:

1. First implementation: Arduino's analogWrite
 1. Disadvantage: cannot specify the needed period.
2. Final implementation: Arduino's built-in servo library.

Table 2-2 Input Pulse Mapping

For input pulse p	Motor Speed and Direction				
	Full Reverse	Prop. Reverse	Neutral	Prop. Forward	Full Forward
Output voltage V_{out} (V)	$V_{cc} - V_{in}$	$-V_{in} < V_{cc} < 0$	$V_{in} = 0$	$0 < V_{in} < V_{cc}$	$V_{in} = V_{cc}$
Default pulse width (μs)	$p \pm 1000$	$1000 - p \pm 1460$	$1460 - p \pm 1540$	$1540 - p \pm 2000$	$2000 + p$
Maximum pulse range (μs)	$500 + p \pm 7500$				

3. 3 different speeds: 1mph (1600) 2 mph(1640) and 5 mph (1700)



Proximity Sensing

- Use two arrays of sensors: one ultrasonic and one IR.
 - For faster and better accuracy.
- Ultrasonic: HC-SR04. Long range and affordable (\$10).
- IR Sensor: Sharp GP2Y0A21YK. Shorter range, but have better information indoors (\$20).

ENSURING SAFETY

Obstacle avoidance:

1. Ultrasonic sensor:
 1. Trigger pin goes HIGH then LOW, sends an audio and counts how long the echo takes to go back.
 2. Problem: only work if trigger pin is set.
 3. Solution:
 1. Use Mega's Timer 1 and polls every 50us for feedbacks (interrupt).
2. Analog sensor:
 1. Use for better accuracy each time echo signal comes back.
3. Implementation:
 1. If back and/or sensors detect obstacles:
 1. In 80cm range: piezo buzzer plays a sound.
 2. In 30cm: mobility stoppage.

ENSURING SAFETY

Remote Kill Switch:

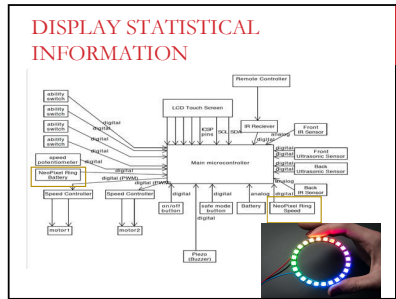
1. TSOP 38238 IR Receiver Diode:
 1. Used on an external interrupt pin.
 2. When received a signal, decode signal and act accordingly.
 1. If stop button is pressed: mobility stoppage.
 2. Can control vehicle (only possible when motors are not moving).
 3. Works with any IR remote.

ENSURING SAFETY

FAIL-SAFE DEFAULT:

1. Check present pins for ability switches:
 1. Only operate if ALL designated inputs are present.
 2. Mobility stoppage as soon as one goes offline.
2. Sensors:
 1. If trigger pin (both front and back ultrasonic) cannot be set: stop the vehicle.
3. Motors:
 1. If receive no signals from the motors, stop the vehicle. (NEEDS WORK).
4. During initial and subsequent setups, touch screen will only say ready if all three are satisfied.

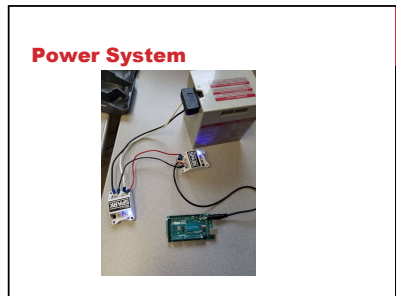
3/24/2018



Pinout Table

18/54 digital inputs used
4/6 interrupt inputs used
4/6 analog inputs used.

Type	Part	Pin
Ultrasonic sensor	Front ultrasonic sensor	2, digital, interrupt
	Back ultrasonic sensor	3, digital, interrupt
IR sensor	front IR sensor	A1, analog
	back IR sensor	A0, analog
Battery tester	Battery tester circuit	A2, analog
Potentiometer	Speed potentiometer	A3, analog
	Ability switches	Switch 1
Ability switches	Switch 2	7, digital
	Switch 3	9, digital
	Switch 4	22, digital
Motor Controller	Left motor controller	10, digital, PWM
	Right motor controller	11, digital, PWM
LED arrays	Speed NeoPixel Ring	Pin 13, digital, PWM
	Battery NeoPixel Ring	Pin 8, digital, PWM
Buzzer	Pinzo Buzzer	Pin 24, digital
Button	On/off button	Pin 25, digital
	Safemode button	Pin 26, digital
Remote controller	Emergency button	Pin 19, digital, interrupt
	IR Receiver	Pin 18, digital, interrupt



- ### STATUS REPORT
- Implementation of all three sub-functions are finished.
 - First round of testing with a human (permission attained) Wednesday February 28th.
 - Second round of testing with the Center for Disability Services Wednesday March 7th.
 - Demonstration: Week 10.
 - Final round of testing with the Center for Disability Services: TBD.
 - Final report (Finals Week).
 - User Manual (Finals Week).

- ### CONCLUSIONS
- Project can be replicated:
 - Hardware is off the shelf.
 - All libraries used are public and code is carefully documented.
 - Detailed manual is included.
 - Project is upgradable:
 - Many ports are still open on the Arduino Mega.
 - Mechanical seat is removable.
 - Software is public.

- ### CONCLUSIONS
- Project can be replicated.
 - Project is upgradable.
 - Potential Impacts:
 - Allow students to practice for prescribed wheelchairs, which would help them move freely within their home and community.
 - Lower the entry for power mobility.
 - Increase quality of life.

3/24/2018

Future Work

1. More elaborate implementation of safety mode.
2. Remove the need of a second microcontroller.
3. Include mobile support.
4. Lower the price.

Questions?

References

- [1] *NYS State Law new york state medicaid wheeled mobility equipment guidelines*, https://www.emedny.org/ProviderManuals/DME/PDFS/DME_Wheeled_Mobility_Equipment_Guidelines.pdf, Accessed: 2017-06-08.
- [2] *Fisher-Pricepower wheels wild thing*, <http://fisher-price.mattel.com/shop/en-us/fp/power-wheels/power-wheels-wild-thing-fgf77>, Accessed: 2017-06-08.
- [3] *Assistive Technology the cooper car*, <http://www.rjcooper.com/coopercar/index.html>, Accessed: 2017-06-08.
- [4] *Powered Mobility medical engineering resource unit's award winning assistive device*, <http://meru.org.uk/what-we-do/bugzi/>, Accessed: 2017-06-08.
- [5] M. A. Jones, I. R. McEwen, and L. Hansen, "Use of power mobility for a young child with spinal muscular atrophy", *Physical Therapy*, vol. 83, no. 3, pp. 253–262, 2003.
- [6] W. B. Mortenson, W. C. Miller, J. Boily, B. Steele, L. Odell, E. M. Crawford, and G. Desharnais, "Perceptions of power mobility use and safety within residential facilities", *Canadian Journal of Occupational Therapy*, vol. 72, no. 3, pp. 142–152, 2005.
- [7] J. Furumasu, *Pediatric powered mobility: Developmental perspectives, technical issues, clinical approaches*. RESNA/Rehabilitation Engineering and Assistive Technology Society of North America, 1997.
- [8] J. J. Campos, D. I. Anderson, M. A. Barbu-Roth, E. M. Hubbard, M. J. Hertenstein, and D. Witherington, "Travel broadens the mind", *Infancy*, vol. 1, no. 2, pp. 149–219, 2000.
- [9] *Arduinoarduino mega 2560 rev 3*, <https://store.arduino.cc/usa/arduino-mega-2560-rev3>, Accessed: 2017-11-13.
- [10] *RevRoboticsspark motor controller*, <http://www.revrobotics.com/spark/>, Accessed: 2017-06-08.
- [11] *Enabling Devicecompact switches – set of 3*, <https://enablingdevices.com/product/compact-switches/>, Accessed: 2017-11-13.
- [12] *Adafruitneopixel ring - 12 x 5050 rgb led with integrated drivers*, <https://www.adafruit.com/product/1643>, Accessed: 2017-11-13.
- [13] *Sparkfunir receiver diode - tsop38238*, <https://www.sparkfun.com/products/10266>, Accessed: 2017-11-19.
- [14] *Adafruit2.8" tft touch shield for arduino w/capacitive touch*, <https://www.adafruit.com/product/1947>, Accessed: 2017-12-13.