Allison Sander, Cameron Canter, Morgan Nishida, Walter Rodenberger, David Long, Misa Jefferson, Bryan Hoang

Go-Baby-Go User Documentation

Human Centered Design Studio

Table of Contents

List of Tables

List of Figures

Introduction

The goal of this project is to modify a Wild-Thing® vehicle into a mobility device for children at a local elementary school. Our main objectives are to create an all-terrain vehicle which could safely allow children to play, learn, and communicate with their peers. Additionally, an important requirement of this project is to allow it to be repeatable by a team of a wide range of skill levels below that of a college engineering team. This document will play a key role in explaining every part of the vehicle modification. The Wild-Thing® was modified by replacing the wheels, seat, and control system and adding a remote on/off control, clipboard table attachment, improved circuitry, and a kickstand for added stability.

Quick Start Manual

Charging and Battery Life

The replacement battery shown in Figure 1 has a longer charge life and is more durable than the battery provided in the Power Wheels vehicle. It may be charged from the back of the car by unplugging it from the circuitry and inserting the charger. If you are not using the car for an extended period, make sure to charge the battery at least once a month.

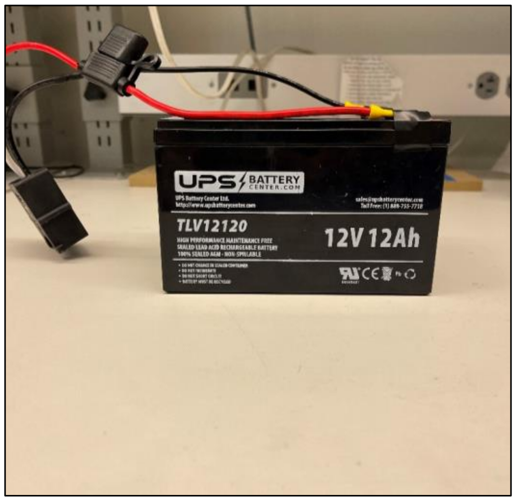


Figure 1: 12V 12Ah Battery Replacement with In-Line Fuse

The charger for the 12V battery included with the Wild-Thing® package and the replacement battery interface as shown in Figure 2 below. There are positive and negative signs on both the battery and charger which show how to connect the two.

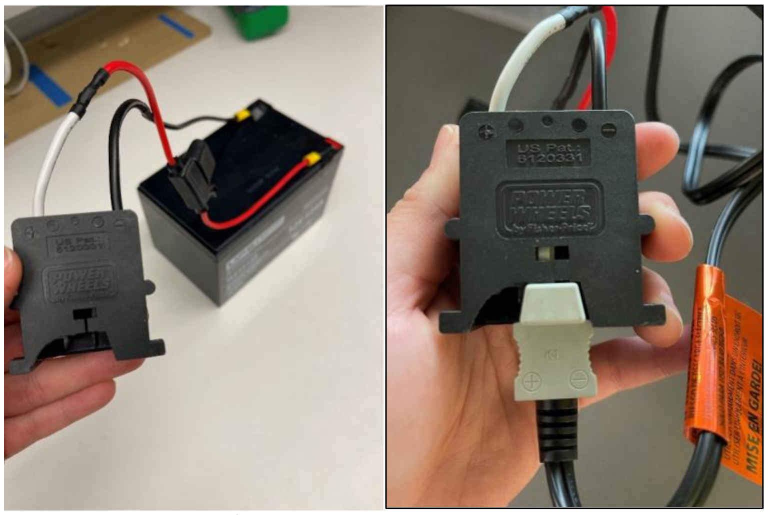


Figure 2: Close up of charger and battery with positive and negative indicators

To best maintain the battery, charge after 7 hours and 45 minutes of continuous use at half speed (this equates to a jogging pace). This should be sufficient for at least one week of recess play. Do not allow the battery to completely drain before charging as this will decrease its lifespan.

Wheels

Check to make sure the tires are filled to 20psi before using the vehicle. Tire pressure should be checked with drastic weather changes and at least every two weeks of use.

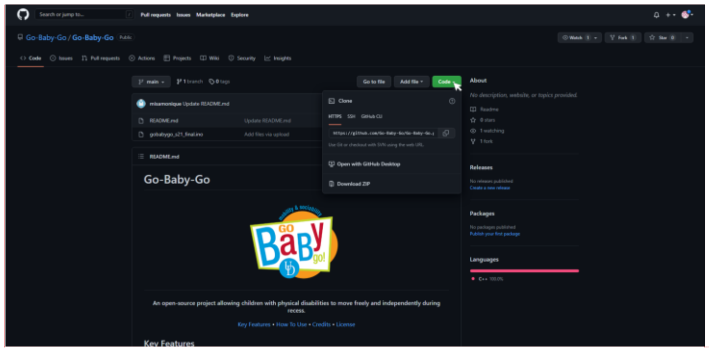
Getting Started with the Code

NOTE: The following instructions are for Windows 10, the process may differ for other operating systems. Additionally, ensure you have the Arduino installed (available at https://www.arduino.cc/en/software) and a program to extract zip folders)

The vehicle electronics were replaced, requiring novel code to be written for the single joystick control. Arduino software is used since it is compatible with the new system and may be altered easily. Take care to make sure the Arduino is not connected to the battery when uploading new code or plugged in to a computer. The final version of the car controlling code limits the vehicle to one action at a time, either forward, reverse, turning left, or turning right. To increase usability for the child and overall safety, the vehicle must come to a complete stop before any change in direction is allowed. This creates a small input delay but prevents the motors from getting out of sync (i.e., decreases the prevalence of drifting). A future version of the program will allow more analog control of the vehicle.

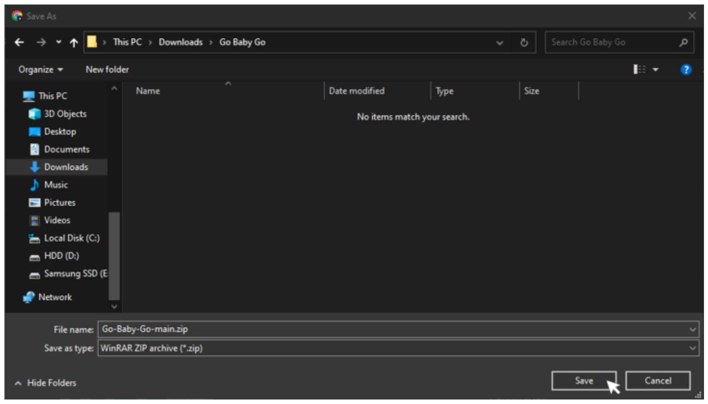
Downloading the Arduino Code

1. To download the code, navigate to the GitHub repository at the following link: https://github.com/Go-Baby-Go/Go-Baby-Go
2. Click the green "Code" button, then press "Download Zip" in the dropdown.

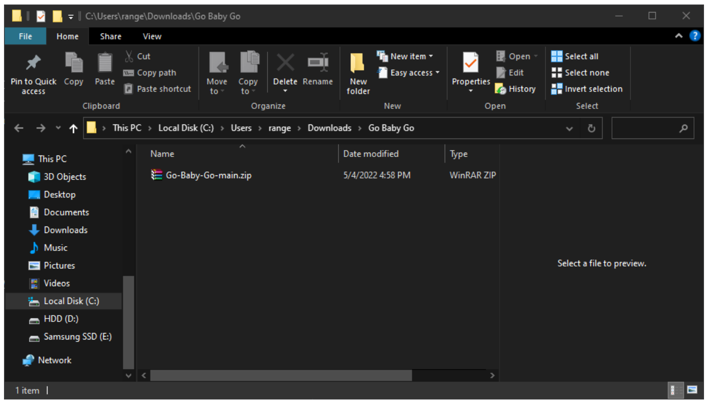


Commented [DL1]: @Chelsea Salinas @Susan Anderson Do I have to label all of these figures? They are just pictures showing the steps to download code and are never referenced directly in the text

3. The file explorer will open. Navigate to the location you wish to save the zip folder to, then click "Save".

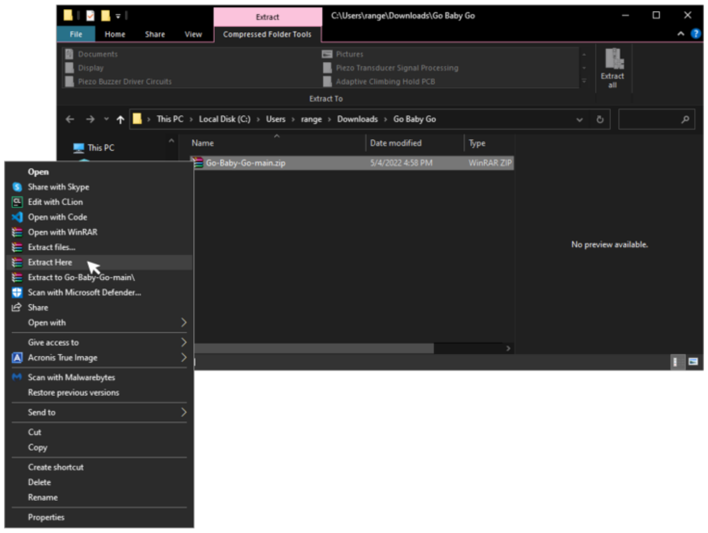


4. Open the file explorer and navigate to the location where you saved the zip folder to.

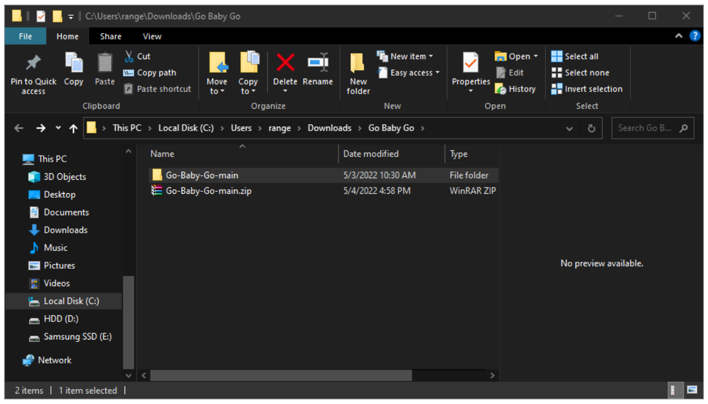


5. Using a program such a WinZip or WinRAR, Extract the contents of the zip folder to the desired location (The following steps for WinRAR are shown below.)

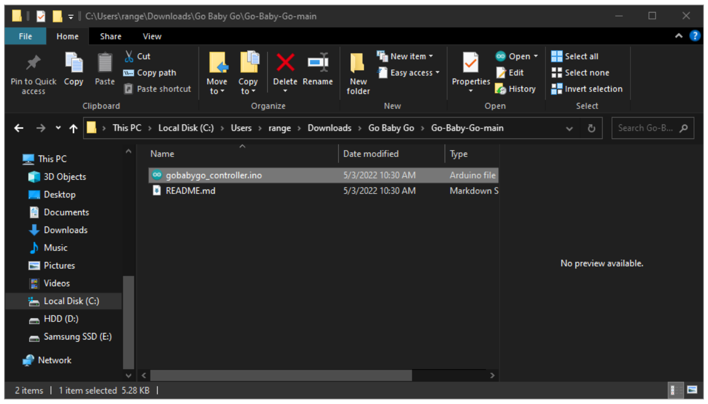
6. Right click on the zip folder and click "Extract Here" from the dropdown.



7. A folder called "Go-Baby-Go-main" will now appear in the location where you saved the zip folder.

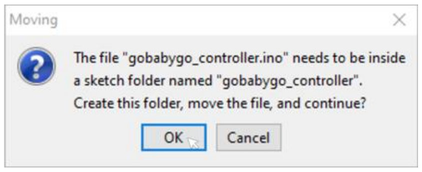


8. Double click on the "Go-Baby-Go-main" folder to open it.



Name	Date modified	Type
gobabygo_controller.ino	5/3/2022 10:30 AM	Arduino file
README.md	5/3/2022 10:30 AM	Markdown S

9. If you already have Arduino installed, double click on the "gobabygo_controller.ino" file. The following pop-up box will appear on the screen:

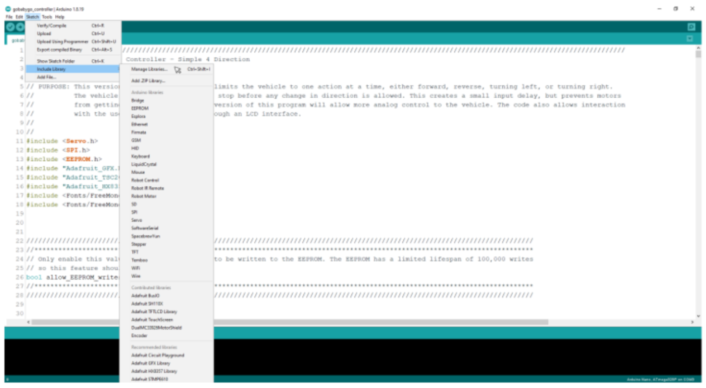

Moving

The file "gobabygo_controller.ino" needs to be inside a sketch folder named "gobabygo_controller". Create this folder, move the file, and continue?

10. Press OK, and the code will open in Arduino.

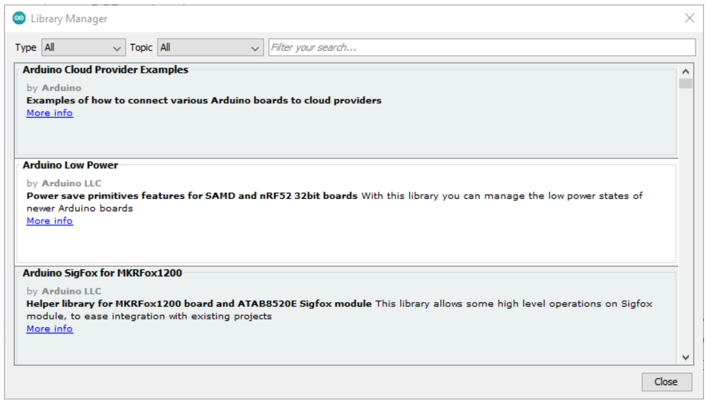
Installing Required Libraries

1. With the "gobabygo_controller" code open, click the "Sketch" tab.
2. Highlight the "Include Library" dropdown then click "Manage Libraries…"



```
//////////////////////////////////////////////////////////////////////////////////////////////////////////
Controller - Simple 4 Direction


// PURPOSE: This versio                limits the vehicle to one action at a time, either forward, reverse, turning left, or turning right.
//          The vehicle                 stop before any change in direction is allowed. This creates a small input delay, but prevents motors
//          from gettin                version of this program will allow more analog control to the vehicle. The code also allows interaction
//          with the us                ough an LCD interface.
//
//
#include <Servo.h>
#include <SPI.h>
#include <EEPROM.h>
#include "Adafruit_GFX.
#include "Adafruit_TSC2
#include "Adafruit_HX83
#include <Fonts/FreeMon
#include <Fonts/FreeMon



/////////////////////////              ////////////////////////////////////////////////////////////////////
//**********************              ********************************************************************
// Only enable this val                to be written to the EEPROM. The EEPROM has a limited lifespan of 100,000 writes
// so this feature shou
bool allow_EEPROM_write
//**********************              ********************************************************************
/////////////////////////              ////////////////////////////////////////////////////////////////////
```

3. The following screen will open.



Type: All
Topic: All
Filter your search...

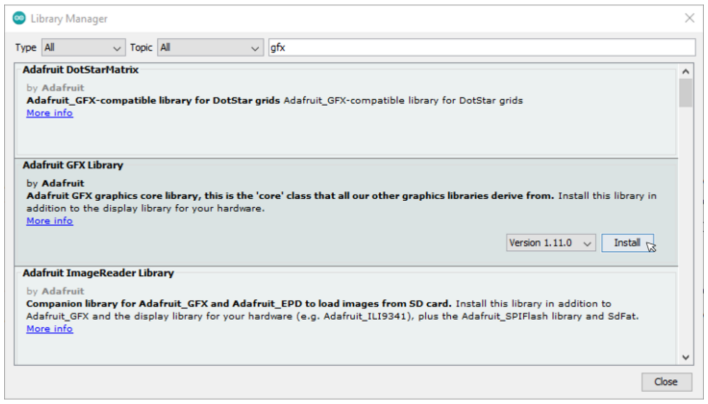
Arduino Cloud Provider Examples
by **Arduino**
Examples of how to connect various Arduino boards to cloud providers
More info

Arduino Low Power
by **Arduino LLC**
Power save primitives features for SAMD and nRF52 32bit boards With this library you can manage the low power states of newer Arduino boards
More info

Arduino SigFox for MKRFox1200
by **Arduino LLC**
Helper library for MKRFox1200 board and ATAB8520E Sigfox module This library allows some high level operations on Sigfox module, to ease integration with existing projects
More info

4. In the search bar at the top right, type "gfx" then wait a moment for the results to appear.

5. Hover over the "Adafruit GFX Library" result, then click "Install".


Library Manager

Type All Topic All gfx

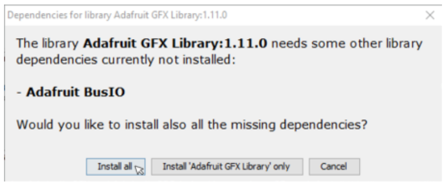
Adafruit DotStarMatrix
by Adafruit
Adafruit_GFX-compatible library for DotStar grids Adafruit_GFX-compatible library for DotStar grids
More info

Adafruit GFX Library
by **Adafruit**
Adafruit GFX graphics core library, this is the 'core' class that all our other graphics libraries derive from. Install this library in addition to the display library for your hardware.
More info
Version 1.11.0 Install

Adafruit ImageReader Library
by Adafruit
Companion library for Adafruit_GFX and Adafruit_EPD to load images from SD card. Install this library in addition to Adafruit_GFX and the display library for your hardware (e.g. Adafruit_ILI9341), plus the Adafruit_SPIFlash library and SdFat.
More info

Close

6. Click "Install all" on the pop-up that appears.


Dependencies for library Adafruit GFX Library:1.11.0

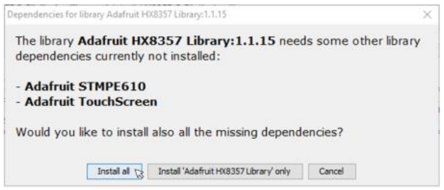
The library **Adafruit GFX Library:1.11.0** needs some other library dependencies currently not installed:

- **Adafruit BusIO**

Would you like to install also all the missing dependencies?

Install all | Install 'Adafruit GFX Library' only | Cancel

7. Back in the search box, type "HX8357" then wait a moment for the results to appear.
8. Highlight the "Adafruit HX8357 Library" result, then click install.
9. Click "Install all" on the pop-up that appears.


Dependencies for library Adafruit HX8357 Library:1.1.15

The library **Adafruit HX8357 Library:1.1.15** needs some other library dependencies currently not installed:

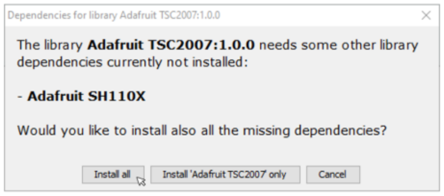
- **Adafruit STMPE610**
- **Adafruit TouchScreen**

Would you like to install also all the missing dependencies?

Install all | Install 'Adafruit HX8357 Library' only | Cancel

10. Back in the search box, type "TSC2007" then wait a moment for the results to appear.
11. Highlight the "Adafruit TSC2007" result, then click install.

12. Click "Install all" on the pop-up that appears.



Dependencies for library Adafruit TSC2007:1.0.0

The library **Adafruit TSC2007:1.0.0** needs some other library dependencies currently not installed:

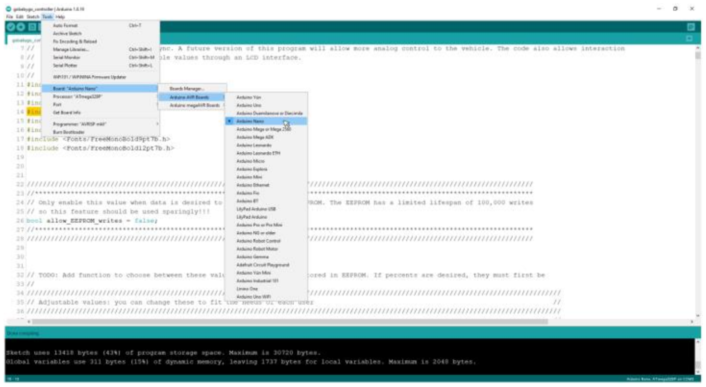
- **Adafruit SH110X**

Would you like to install also all the missing dependencies?

13. All the required libraries are now installed. You may exit the library manager.

Uploading the Code to the Arduino

1. With the "gobabygo_controller" code open, click the "Tools" tab.
2. Hover over the "Board:____" dropdown, then the "Arduino AVR Boards" dropdown, and finally, click "Arduino Nano"

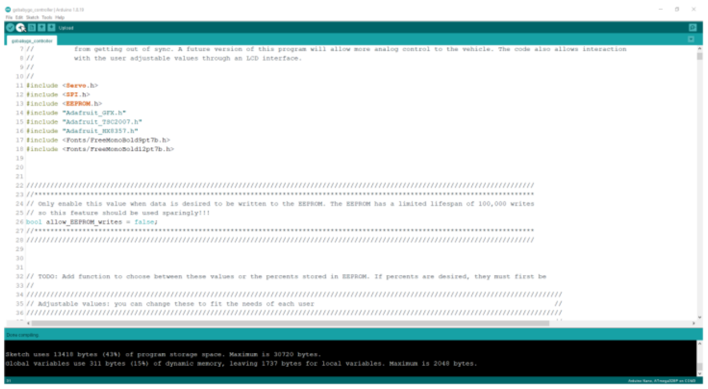


```
#include <Fonts/FreeMonoBold9pt7b.h>
#include <Fonts/FreeMonoBold12pt7b.h>
```

```
Sketch uses 13418 bytes (43%) of program storage space. Maximum is 30720 bytes.
Global variables use 311 bytes (15%) of dynamic memory, leaving 1737 bytes for local variables. Maximum is 2048 bytes.
```

3. Once again, click the "Tools" tab.
4. Hover over the "Port" dropdown and take note of which options are available.
5. Plug the Arduino Nano into your computer using a USB cable.
6. Repeat steps 3 and 4 and you should see that a new COM port option has appeared. Select this COM port.

7. Finally, click the right arrow at the top left to upload the code to the Arduino.


gobabygo_controller | Arduino 1.8.19
File Edit Sketch Tools Help
Upload
gobabygo_controller
Done compiling.
Arduino Nano, ATmega328P on COM3

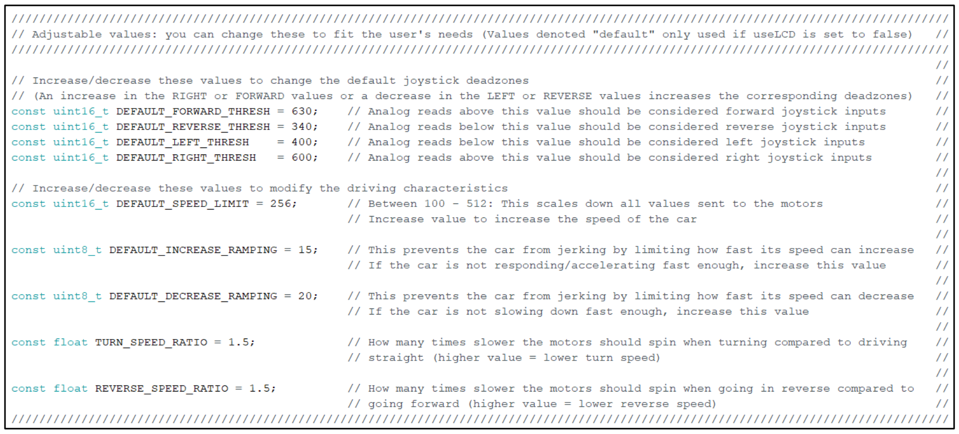
```
//          from getting out of sync. A future version of this program will allow more analog control to the vehicle. The code also allows interaction
//          with the user adjustable values through an LCD interface.
//
//
#include <Servo.h>
#include <SPI.h>
#include <EEPROM.h>
#include "Adafruit_GFX.h"
#include "Adafruit_TSC2007.h"
#include "Adafruit_HX8357.h"
#include <Fonts/FreeMonoBold9pt7b.h>
#include <Fonts/FreeMonoBold12pt7b.h>



////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//**************************************************************************************************************
// Only enable this value when data is desired to be written to the EEPROM. The EEPROM has a limited lifespan of 100,000 writes
// so this feature should be used sparingly!!!
bool allow_EEPROM_writes = false;
//**************************************************************************************************************
////////////////////////////////////////////////////////////////////////////////////////////////////////////////



// TODO: Add function to choose between these values or the percents stored in EEPROM. If percents are desired, they must first be
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Adjustable values: you can change these to fit the needs of each user                                          //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
Sketch uses 13418 bytes (43%) of program storage space. Maximum is 30720 bytes.
Global variables use 311 bytes (15%) of dynamic memory, leaving 1737 bytes for local variables. Maximum is 2048 bytes.
```

Modifying the Code

Before getting started with the vehicle, some modifications to the code may be required depending on your particular hardware, or the user's needs. Values which are adjustable are placed within a block at the top of the code as seen in Figure 3. Any value with the prefix "DEFAULT" is only used if the USE_LCD constant is set to false.

- Joystick deadzones are implemented to adjust the sensitivity of the joystick and are controlled by the directional thresholds, denoted by "*direction*_THRESH". Increasing the RIGHT and FORWARD values or decreasing the LEFT and REVERSE values increases that direction's deadzone. Increasing the deadzones will require the joystick to be pushed further in the corresponding direction before an input is registered and motion of the vehicle begins.
- The SPEED_LIMIT constant controls how fast the vehicle goes. Increasing its value will raise the vehicle's top speed.
- The INCREASE_RAMPING constant controls how fast the vehicle accelerates to the speed limit. Increasing this value will make the vehicle reach top speed faster but will also increase the "jerkiness."
- The DECREASE_RAMPING constant controls how fast the vehicle decelerates to a stop between direction changes or braking. Increasing this value will make the vehicle take less time to come to a stop and thus, respond to direction changes quicker.
- The TURN_SPEED_RATIO constant controls how fast the vehicle turns compared to driving straight. Increasing this value will make the vehicle turn slower.
- The REVERSE_SPEED_RATIO constant controls how fast the vehicle drives in reverse compared to driving forward. Increasing this value will decrease the reverse speed.



```
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Adjustable values: you can change these to fit the user's needs (Values denoted "default" only used if useLCD is set to false)    //
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                                                                                                                     //
// Increase/decrease these values to change the default joystick deadzones                                                           //
// (An increase in the RIGHT or FORWARD values or a decrease in the LEFT or REVERSE values increases the corresponding deadzones)   //
const uint16_t DEFAULT_FORWARD_THRESH = 630;    // Analog reads above this value should be considered forward joystick inputs        //
const uint16_t DEFAULT_REVERSE_THRESH = 340;    // Analog reads below this value should be considered reverse joystick inputs        //
const uint16_t DEFAULT_LEFT_THRESH    = 400;    // Analog reads below this value should be considered left joystick inputs           //
const uint16_t DEFAULT_RIGHT_THRESH   = 600;    // Analog reads above this value should be considered right joystick inputs          //
                                                                                                                                     //
// Increase/decrease these values to modify the driving characteristics                                                              //
const uint16_t DEFAULT_SPEED_LIMIT = 256;       // Between 100 - 512: This scales down all values sent to the motors                 //
                                                // Increase value to increase the speed of the car                                   //
                                                                                                                                     //
const uint8_t DEFAULT_INCREASE_RAMPING = 15;    // This prevents the car from jerking by limiting how fast its speed can increase    //
                                                // If the car is not responding/accelerating fast enough, increase this value        //
                                                                                                                                     //
const uint8_t DEFAULT_DECREASE_RAMPING = 20;    // This prevents the car from jerking by limiting how fast its speed can decrease    //
                                                // If the car is not slowing down fast enough, increase this value                   //
                                                                                                                                     //
const float TURN_SPEED_RATIO = 1.5;             // How many times slower the motors should spin when turning compared to driving     //
                                                // straight (higher value = lower turn speed)                                        //
                                                                                                                                     //
const float REVERSE_SPEED_RATIO = 1.5;          // How many times slower the motors should spin when going in reverse compared to    //
                                                // going forward (higher value = lower reverse speed)                                //
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Figure 3: Adjustable Values Block Arduino Code

To Match Your Hardware:

The motors or joystick for your vehicle may have slightly different behavior than what the code was originally designed for. To get the best experience, you may need to adjust the minimum values for the joystick dead zones and motor speeds.

Speed Limit Calibration Procedure

1. Ensure the battery is completely recharged before performing this test
2. Set the CALIBRATION_MODE_ENABLE constant to true
3. Set the USE_LCD constant to false
4. Upload the code
5. Open the serial monitor and follow the instructions printed to the screen

Dead Zone Calibration Procedure

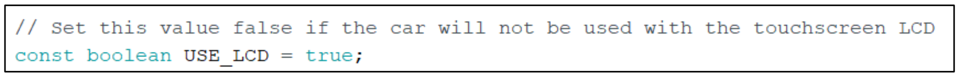
1. Set the vehicle up on an elevated surface where the wheels can freely spin.
2. Make sure the USE_LCD constant is set to false
3. Push the joystick into the right track of the faceplate and then push forward.
4. If the wheels of the vehicle transition from right turn to forward motion, increase the FORWARD_THRESH by 5 and reupload the code.
5. Repeat steps 2 and 3 until the vehicle does not transition from right turn to forward motion while in the right track.
6. Push the joystick into the right track of the faceplate and then pull downward.
7. If the wheels of the vehicle transition from right turn to reverse motion, lower the REVERSE_THRESH by 5 and reupload the code.

8. Repeat steps 6 and 7 until the vehicle does not transition from right turn to reverse motion while in the right track.
9. Push the joystick into the top track of the faceplate and then push to the right.
10. If the wheels of the vehicle transition from forward motion to a right turn, increase the RIGHT_THRESH by 5 and reupload the code.
11. Repeat steps 9 and 10 until the vehicle does not transition from forward motion to a right turn.
12. Push the joystick into the top track of the faceplate and then push to the left.
13. If the wheels of the vehicle transition from forward motion to a left turn, decrease the LEFT_THRESH by 5 and reupload the code.
14. Repeat steps 12 and 13 until the vehicle does not transition from forward motion to a left turn

To Suit the User:

Adjustable values which may be changed to fit the needs of each user are shown in a block at the beginning of the script. These values may need to be adjusted to calibrate the code for the joystick. If you notice the car moves forward or backward when the joystick is in the left or right track of the joystick cover plate, increase the problematic direction's dead zone as discussed above.

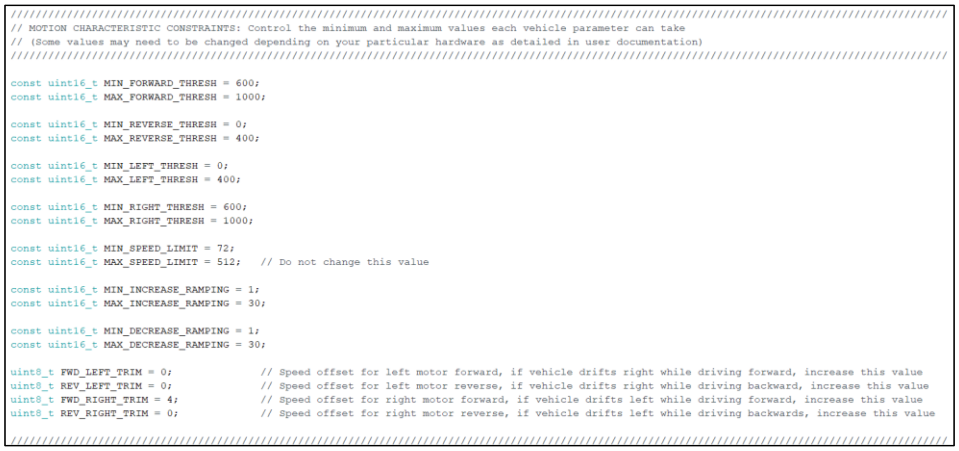
While the code is designed to be interfaced with a touch screen LCD, it may also be run standalone with all sensitivity and motion characteristics modified directly in the code. If this functionality is desired, change the variable labeled "USE_LCD" to false, as seen in Figure 4. This will activate the variables with the prefix DEFAULT within the Adjustable Values block.



```
// Set this value false if the car will not be used with the touchscreen LCD
const boolean USE_LCD = true;
```

Figure 4: Code segment for operating without LCD

If instead, the LCD is desired, then the motion characteristics controllable by the LCD interface will be constrained by the code block labeled MOTION CHARACTERISTIC CONSTRAINTS at the top of the script as seen in Figure 5



```
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// MOTION CHARACTERISTIC CONSTRAINTS: Control the minimum and maximum values each vehicle parameter can take
// (Some values may need to be changed depending on your particular hardware as detailed in user documentation)
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

const uint16_t MIN_FORWARD_THRESH = 600;
const uint16_t MAX_FORWARD_THRESH = 1000;

const uint16_t MIN_REVERSE_THRESH = 0;
const uint16_t MAX_REVERSE_THRESH = 400;

const uint16_t MIN_LEFT_THRESH = 0;
const uint16_t MAX_LEFT_THRESH = 400;

const uint16_t MIN_RIGHT_THRESH = 600;
const uint16_t MAX_RIGHT_THRESH = 1000;

const uint16_t MIN_SPEED_LIMIT = 72;
const uint16_t MAX_SPEED_LIMIT = 512;   // Do not change this value

const uint16_t MIN_INCREASE_RAMPING = 1;
const uint16_t MAX_INCREASE_RAMPING = 30;

const uint16_t MIN_DECREASE_RAMPING = 1;
const uint16_t MAX_DECREASE_RAMPING = 30;

uint8_t FWD_LEFT_TRIM = 0;              // Speed offset for left motor forward, if vehicle drifts right while driving forward, increase this value
uint8_t REV_LEFT_TRIM = 0;              // Speed offset for left motor reverse, if vehicle drifts right while driving backward, increase this value
uint8_t FWD_RIGHT_TRIM = 4;             // Speed offset for right motor forward, if vehicle drifts left while driving forward, increase this value
uint8_t REV_RIGHT_TRIM = 0;             // Speed offset for right motor reverse, if vehicle drifts left while driving backwards, increase this value

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Figure 5: Motion Characteristic Constraints Block

Remote On/Off

To use this safety feature stand within a range of approximately 30’ to the vehicle. This remote has the ability to stop the vehicle using this hand-held remote-control device. This device allows the user to have a sense of freedom by having their own space without sacrificing the safety that adult supervision provides.

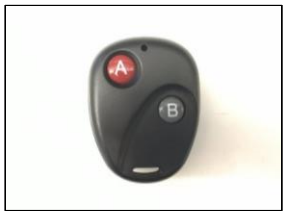
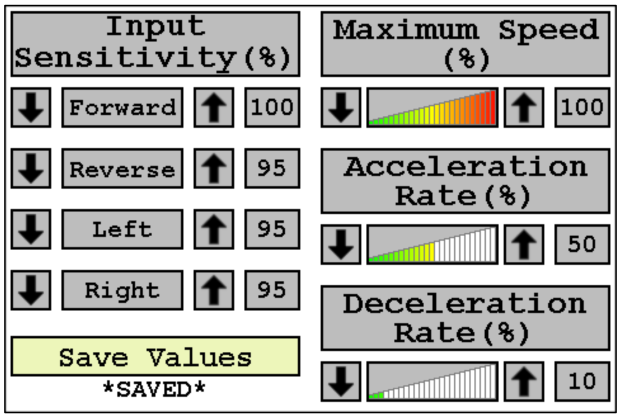


Figure 6: Remote on/off device

LCD Screen

The touchscreen interface shown below allows the adjustment of input sensitivity in the forward, reverse, left, and right directions as well as the maximum speed, acceleration rate, and deceleration rate as a percentage. These values are also able to be saved so that they will be ready for the next use.



Input Sensitivity(%)

Forward 100

Reverse 95

Left 95

Right 95

Save Values

SAVED

Maximum Speed (%) 100

Acceleration Rate(%) 50

Deceleration Rate(%) 10

Figure 7: LCD display

Using the Kickstand

To use the kickstand most effectively, press them downward and towards the vehicle as much as possible with your foot. The kickstands should not be perpendicular to the ground but angled inward towards the car. When finished, push the kickstands out with your foot and rotate them until they are laying on top of the chassis.

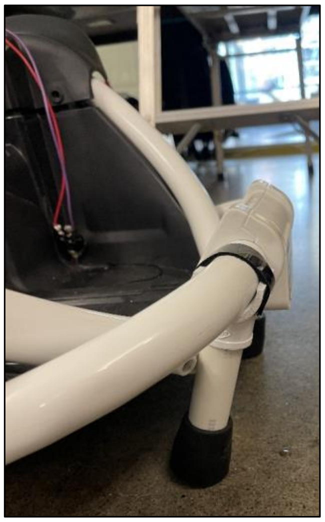


Figure 8: Kickstand in safe position to get in and out of vehicle

Assembly Manual

Bill of Materials

Table 1: Bill of Materials

Part	Company	Unit $	#	Total $	Link to part
Mechanical Components					
8" Pneumatic Swivel Caster	Harbor Freight	$15.99	1	$15.99	Harbor Freight Caster
Caster Fixtures	The Home Depot	$15.00	1	$15.00	
Steel Plate	The Home Depot	$13.64	1	$13.64	Home Depot Steel Plate
2-Pc Rim Pneumatic Wheels	Northern Tool	$69.99	2	$139.98	Pneumatic Wheel Northern Tool
Fixtures	ACE Hardware	$17.74	1	$17.74	Ace Hardware
Car Seat	Walmart	$58.00	1	$58.00	Walmart Car Seat
U-Bolts	The Home Depot	$3.32	4	$13.28	Home Depot U-Bolts
Velcro	The Home Depot	$7.98	1	$7.98	Home Depot Velcro
Storage Clipboard	Amazon-Officemate	$24.63	1	$24.63	Amazon Storage Clipboard
PVC Reducers (1"-1/2")	The Home Depot	$2.53	2	$5.06	Home Depot Reducers
1/2" PVC	The Home Depot	$1.48	1	$1.48	Home Depot PVC
Crutch Ends (4pk)	Amazon	$6.50	1	$6.50	Amazon Crutch Ends
PVC Cement	The Home Depot	$7.45	1	$7.45	Home Depot PVC Cement
Epoxy	The Home Depot	$6.47	1	$6.47	Home Depot Epoxy
Electrical Components					
Remote Control Relay	Amazon	$14.39	1	$14.39	Amazon Relay
SPARK Motor Controller	Rev Robotics	$50.00	2	$100.00	Rev Robotics Motor Controller
Arduino Nano	Arduino	$21.86	1	$21.86	Arduino Nano
1714955 Terminal	Digi-Key	$1.83	7	$12.81	DigiKey Terminal
Analog Joystick	Adafruit	$19.95	1	$19.95	Adafruit Joystick
Power Distribution Bus	Adafruit	$1.95	2	$3.90	Adafruit Bus
On/Off Switch	Hog Slat	$4.97	1	$4.97	Hog Slat Switch
PCB	JLC PCB	$30.00	1	$30.00	https://jlcpcb.com/
12 ga. Stranded Wire (Black)		$1.00	15	$15.00	
12 ga. Stranded Wire (Red)		$1.00	15	$15.00	
12 V 12 Ah Battery Replacement	UPS Battery Center	$45.00	1	$45.00	EBay Battery
In Line Fuse	Amazon	$7.99	1	$7.99	Amazon In-Line Fuse
Male to Female Headers	Amazon	$5.99	1	$5.99	Amazon Header
3.5" 320x480 LCD Touchscreen	Adafruit	$39.95	1	$39.95	Adafruit LCD Screen
Resistive Touchscreen Controller	Adafruit	$4.95	1	$4.95	Adafruit Touchscreen Controller
20 Rectangular Connectors	Digi-Key	$1.42	2	$2.84	DigiKey Connectors
Connector Header	Digi-Key	$8.65	2	$17.30	DigiKey Header
Tin 22-24 AWG Crimp	Digi-Key	$0.06	100	$5.98	DigiKey Crimp
24 AWG Hook-Up wire	Digi-Key	$5.90	1	$5.90	DigiKey Wire
Misc.					
Command Strips	Target	$3.69	1	$3.69	Target Command Strips
Expandable Wire Loom 10ft	Amazon	$7.99	1	$7.99	Amazon Wire Loom
Zip ties	Target	$4.29	1	$4.29	Target Zipties
Solder	The Home Depot	$10.47	1	$10.47	Home Depot Solder
TOTAL				**$733.42**	

Upgraded Wheels

Back Wheel Replacement and Steel Plate

To account for a heightened center of gravity, we implemented a rear wheel to the Wild-Thing® assembly for increased stability. The rear wheel assembly is held together by an aluminum plate that must be machined to support a caster wheel, screw eye, and a bolt that secures it to the Wild-Thing®. The screw eye is used to hold tension in the rear strap of the car seat to prevent movement. The steel plate can be attached to the Wild-Thing® through the same hole that is used to support the original rear wheel.

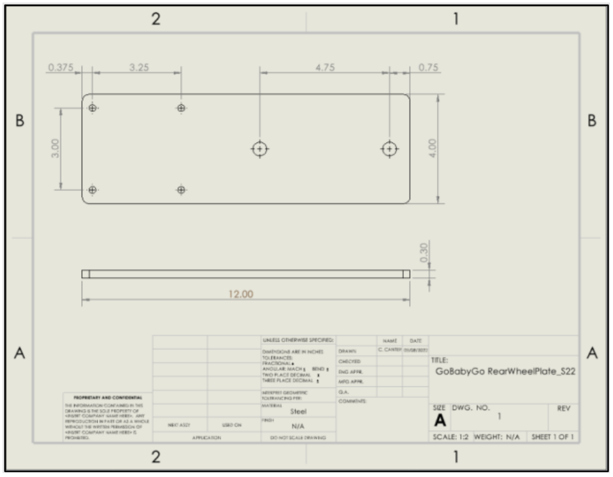


Figure 9: Rear Wheel Plate Drawing

Materials

- 1x 12”x4”x0.25” Steel Plate
- 1x 8” Caster Wheel
  - 4x Bolts
  - 2x Washers
  - 4x Nuts
- 1x Screw Eye
  - 2x Nuts
  - 2x Washers
- 1x Bolt
- 1x Nut

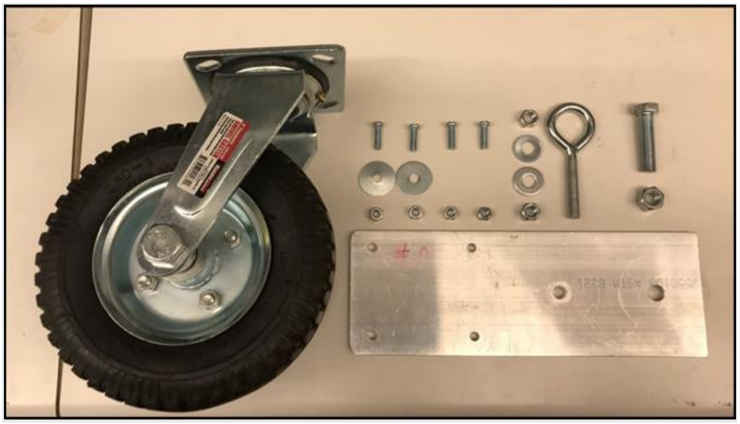


Figure 10: Required Materials

Required Tools

- Phillips Head Screwdriver
- Socket Wrench
- Bandsaw (For Plate Dimensions)
- Drill Press (For Fastener Holes)

Directions

- Machine steel plate to be approximately 12 inches long, 4 inches wide, and 0.3 inches thick. Fillet all sharp edges to mitigate possible user injury.

- Machine 4x through holes into steel plate to match your caster wheels hole pattern as shown in the drawing above. The tolerance of the dimensions of the holes with respect to each other is important to ensure an optimal connection between the plate and caster wheel.

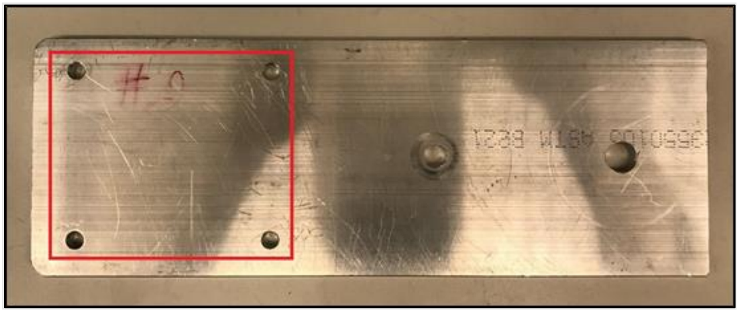


Figure 11: Rear Wheel Plate (Caster Holes)

- Machine a through hole into steel plate for the screw eye ensuring that it is centered with respect to the width as much as possible. The tolerance of this hole with respect to the length is not very important as this will only be used for a strap to fasten the car seat.

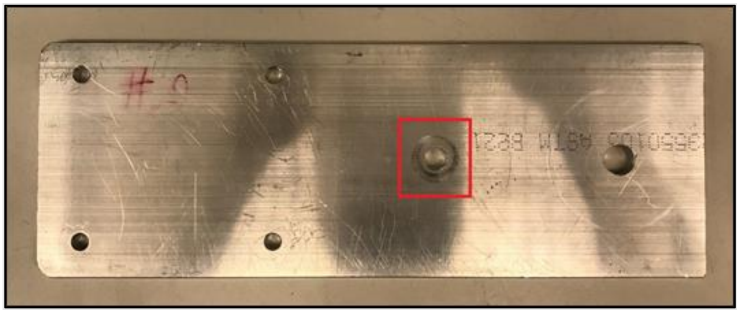


Figure 12: Rear Wheel Plate (Screw Eye Hole)

- Machine a through hole into the steel plate to connect the plate to the Wild-Thing® assembly. This hole must be ~2" or less from the end of the plate to ensure clearance with the Wild-Thing® frame.

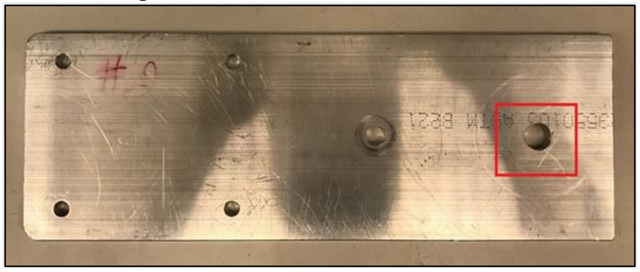


Figure 13: Rear Wheel Plate (Wild-Thing® Connection Hole)

- Connect the caster wheel to the steel plate using 4x bolts on top and 4x washers and 4x bolts on the bottom. Ensure a hand tight connection to prevent disconnections when operating.

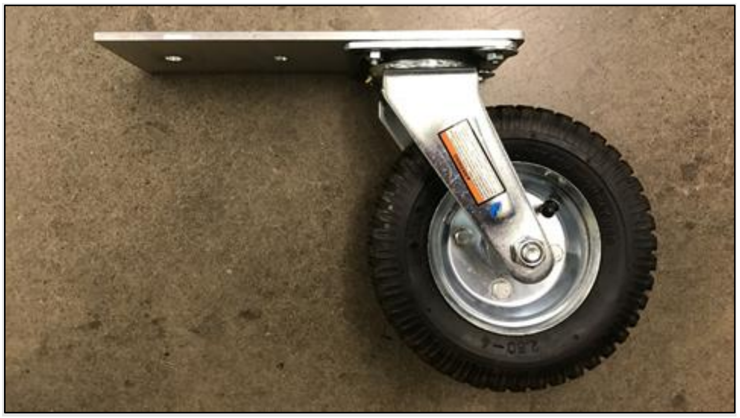


Figure 14: Rear Wheel Plate w/ Caster Wheel

- Connect the screw eye to the center hole using 1x nut and 1x washer on top and 1x nut and 1x washer on the bottom of the plate. Ensure a tight connection so that tension in the car seat strap will not cause them to loosen.

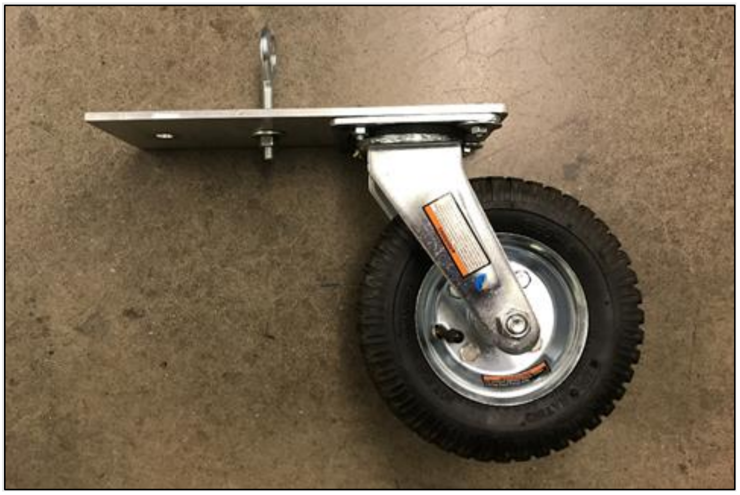


Figure 15: Rear Wheel Plate w/ Caster Wheel and Screw Eye

- Remove the plastic cap from the top of the Wild-Thing® frame above the rear wheelbase to remove the stock rear wheel. This should expose a ~0.5in hole that will be used to fasten the new caster wheel assembly.

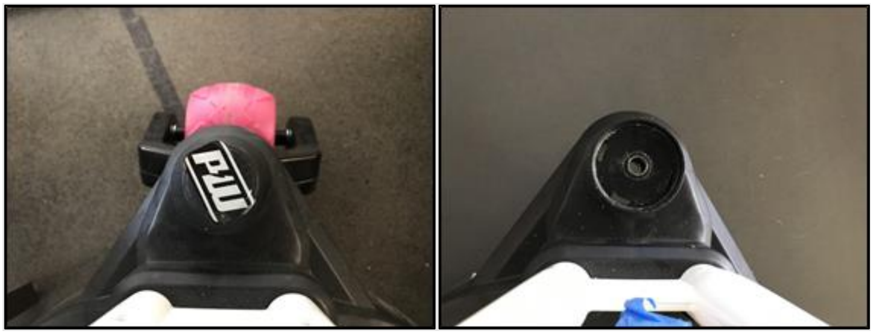


Figure 16: Removal of Wild-Thing® Wheel

- Fasten the steel plate to the Wild-Thing® frame using 1x bolt on top with 1x washer and 1x nut on the bottom. Fasten tightly to ensure that the rear wheel assembly stays straight in relation to the Wild-Thing®. This connection may need to be retightened occasionally after a moderate amount of use.

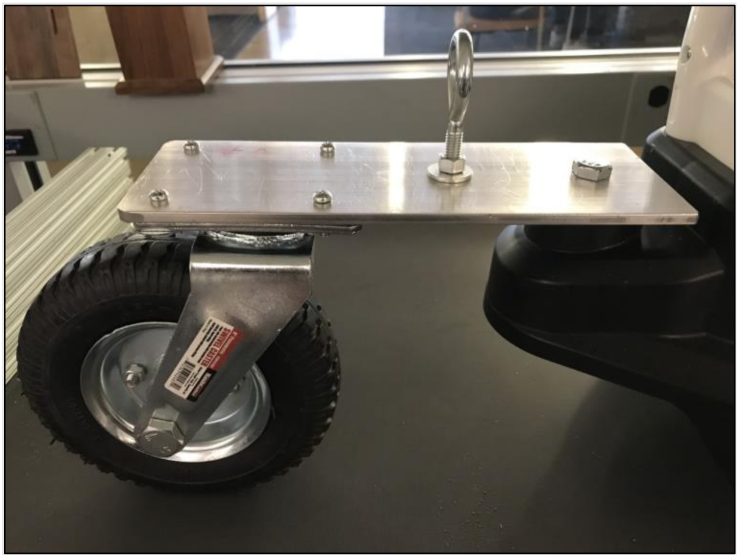


Figure 17: Rear Wheel Assembly Connected to Wild-Thing®

All-Terrain Motor-Controlled Wheels

A main objective of this modification was to make the Wild-Thing**®** vehicle all terrain so that it could be used in the playground and classroom. The team decided the wheels required should have the following features: "zig-zag" or nonlinear tread to rubber prevent cracking, an inner tube to easily refill tires and reduce the instances of flat tires, and a two-piece rim for increased ease of use in replacing the tires when necessary.

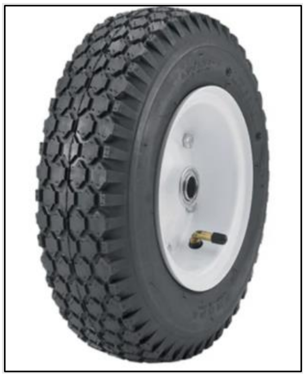
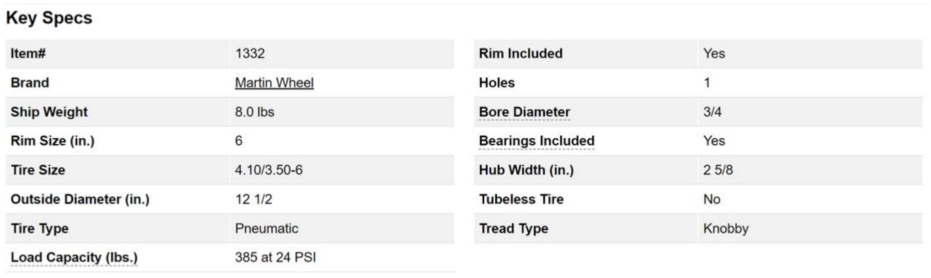


Figure 18: 2-Pc. Split Rim Low-Speed Pneumatic Wheel Assembly by Martin Wheel

Based on these features, we chose the 2-piece split rim low-speed pneumatic wheel assembly by Martin Wheel, which has the specifications shown below.

Table 2: Specifications for All-Terrain Wheels


Key Specs

Item#	1332	**Rim Included**	Yes
Brand	Martin Wheel	**Holes**	1
Ship Weight	8.0 lbs	**Bore Diameter**	3/4
Rim Size (in.)	6	**Bearings Included**	Yes
Tire Size	4.10/3.50-6	**Hub Width (in.)**	2 5/8
Outside Diameter (in.)	12 1/2	**Tubeless Tire**	No
Tire Type	Pneumatic	**Tread Type**	Knobby
Load Capacity (lbs.)	385 at 24 PSI		

The driver piece attached to the existing wheels is needed to interface with the white, steering piece on the axle (seen in Figure 19). This portion must be removed from the existing wheel and attached to the new wheels.

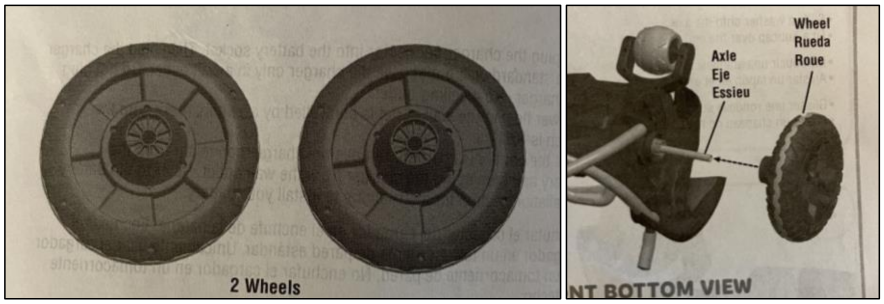


Figure 19: Part Pictures from Power Wheels Manual of Wheel-Axle Interface

Attaching the new wheels thus requires cutting the old wheels. Take caution when performing the following steps as they could result in injury if done without proper safety precautions.

Removing the Driver Piece

Take apart the wheels by unscrewing the bolts of the inner and outer ring of the plastic wheels with a Phillips-head screwdriver. Discard the front piece and rubber center piece and recycle if possible. Cut along the red path shown to remove the driver section seen in Figure 20.

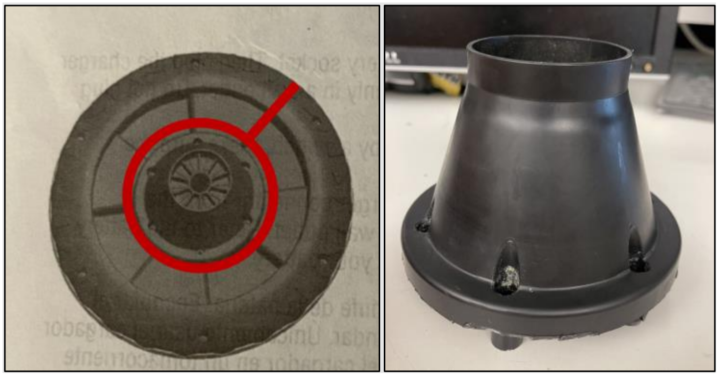


Figure 20: Cutting Path and Driver Piece Removed

Adapting the Driver Piece

Using a 3/8" drill bit, widen the six holes of the driver to create clearance holes. Then, using a box cutter, remove the protruding three plastic sections around the center hole as seen in Figure 21. Each should require a downward and outward cut, and such that the three plastic pieces are close to level with the center hole. These cuts do not have to be precise.

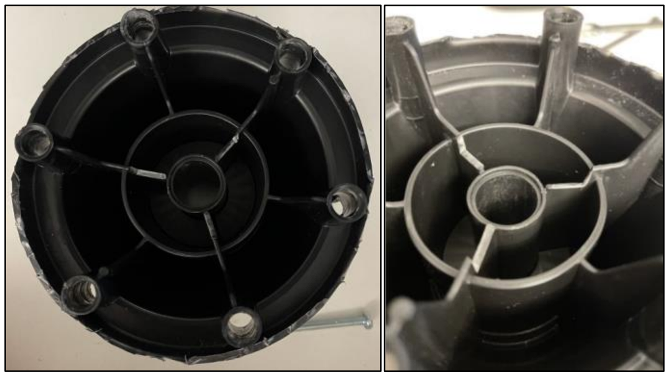


Figure 21: Driver Adaptations

Assembling the New Wheels

Deflate the wheels about halfway to aid in ease of disassembly. Remove the nuts, bolts, and locking washers with a ½” socket wrench and set aside. Purchase the parts shown in Table 3 below, available at the local Ace Hardware.

Table 3: Fixtures for Wheels

Part	Size	Number
Hillman Metric Pan Head Phillips Screws	M5 -.80 x 50mm	12
Hillman Metric Flat Washers	M6	24
Hillman Metric Hex Nuts	M5-0.80	14
Hillman Metric Split Lock Washers	M5	12

Align the driver piece with the back rim (the rim piece without the cutout for inflation) and front rim with the tire deflated. Place washers on each of the 6 holes and insert the Phillips screws in the clearance holes. If conducting the assembly by yourself, it may be helpful to tape the screws in place while working on the other side of the wheel.

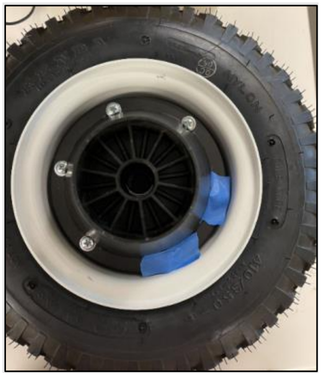


Figure 22: Attaching the Driver to the Wheel

Flip the wheel over to show the opposite rim and the ends of the Phillips screws. On each exposed screw end, place a flat washer, lock washer, and then hex nut. Hand tighten the hex nuts or loosely secure each with a wrench until you feel resistance. The fixtures should appear as in Figure 23. Finally, inflate the wheel to 25psi. Repeat this process with the second wheel

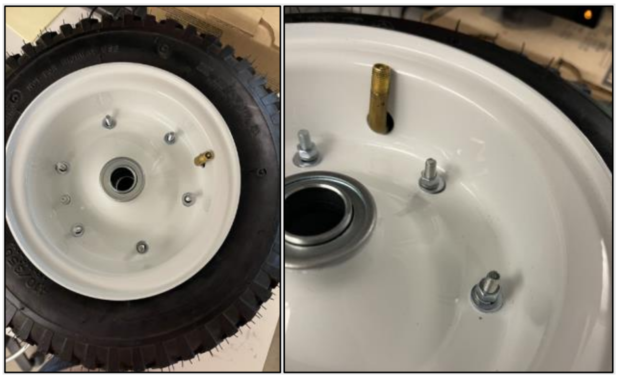


Figure 23: Fixture Order and Configuration

Attaching the Wheels to the Vehicle

Once the wheels and driver pieces are assembled, slide the wheels onto the axles, driver side inward, and align the notches in the driver with the notches in the white plastic piece of the axle. Press until the wheel is secured with about ½" of the axle exposed. Using two of the screws from the old wheels and the two remaining hex nuts, place a set screw in the hole at the end of the axle. This will secure the wheels to the axle.

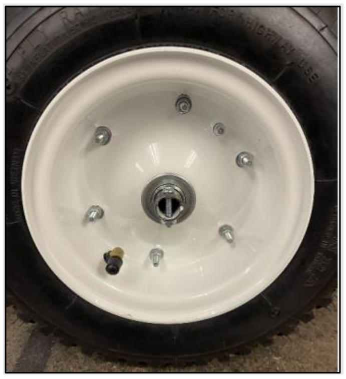


Figure 24: New Wheels on Axle

Improved Seat

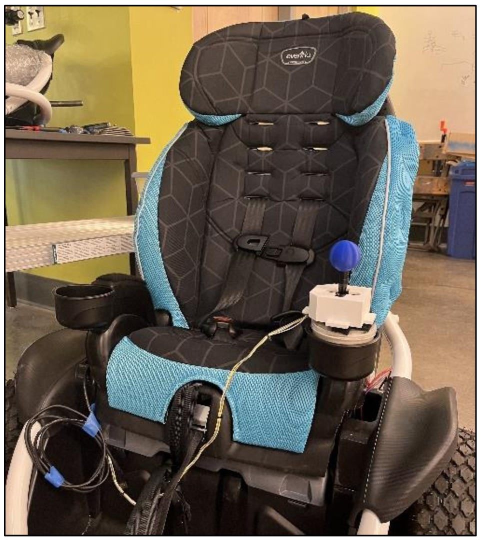


Figure 25: Implementation of Improved Seat

To reach the design requirement of implementing a more comfortable and secure seating system, a modified car seat was added, which supports the back and neck of the child. The car seat was attached to the original seat with U-bolts, and eye bolts were mounted to the vehicle to attach the straps of the car seat and act as an anchor for the seat in the car. One fixture is on the chassis in front of the seat and another mounted on the aluminum plate of the rear wheel. This method was chosen specifically to minimize the amount of custom modification required and to make repeatability for future design teams easier. The car seat model, Evenflo Advanced Chase LX Harness Booster Car Seat, was chosen by the group and client for its safety, durability, cost effectiveness, and reliability. Additional modifications to the design include the use of the cupholders to house the joystick as well as provide a support for the table attachment. The following is a description of all modifications made to the vehicle.

Connecting the Car Seat to the Existing Seat

Four U-bolts were used in the configuration shown above to connect the desired car seat to the plastic interface piece. The plastic piece was adapted from the seat provided with the Wild-Thing® vehicle by removing the backrest and drilling 8 half inch holes for the U-bolt attachments. These bolts allow for a strong, durable connection between the car seat and plastic seat, and the use of the previous seat allows for the original interface between the seat and chassis to remain intact. This will provide the most stability possible at this connection.

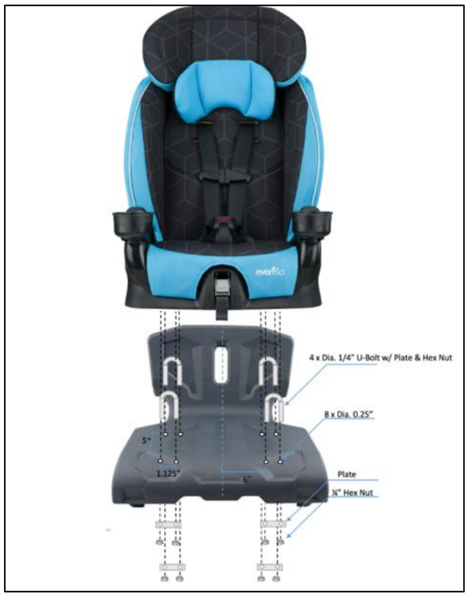


Figure 26: U-Bolt Configuration to Connect Car Seat to Existing Seat

Connecting the Seat Straps to the Chassis

As seen in Figure 27, the straps attached to the car seat were used to provide additional support and stability by connecting them to the chassis and rear metal plate. There is one hole in the aluminum plate and a half inch hole drilled into the foot area of the chassis. These straps are necessary to combat the effects of tilt during turning. Both eye bolts were secured with nuts on either side of the material to lock them in place and mitigate any drift during use. The straps can also be adjusted to maximum tension upon attaching the seat.

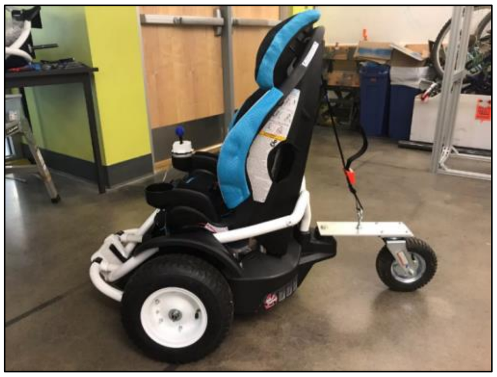


Figure 27: Seat Strap Connection to Chassis

Using the Cup Holders and Arm Rests

The cup holder attachments from the car seat are used to house a 3D printed cylindrical insert that is secured with Velcro and can be moved from either side depending on the dominant hand of the user. The 3D printed insert also has Velcro on its top face to secure the joystick. The table is attached on top of the arm rests and is also connected via Velcro. As shown in figure X, the table has an additional 3D printed insert which allows the joystick to be placed in a more central position for the user and can be adjusted based on the user's reachability.

Storage Clipboard Table Attachment

A storage clipboard is attached to the seat and used as a table and to act as storage for the child's drawings, pencils, crayons, papers, etc. It can be removed easily due to the Velcro attachment at the armrests of the seat. This also allows for varied placement of the table as close or as far as necessary for the child's size. The soft side of the Velcro should be used on the armrests to not hinder their functionality.

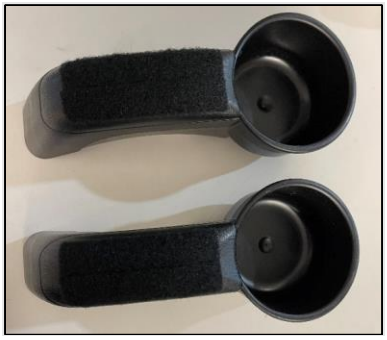


Figure 28: Velcro on Armrests

Any storage clipboard with a length at or above 16" will work for this application. The OIC Carry All Clipboard Storage Box is the team's suggested option because of its durability, rubber rounded corners, and internal organization. Additionally, the clipboard may be taken home with the child at the end of the school day.

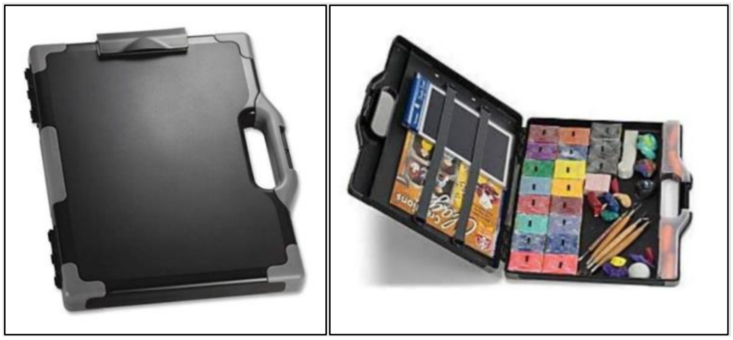


Figure 29: OIC Carry All Clipboard Storage Box

Attaching the Table

Place the clipboard on the car seat armrests and push it as far inward as possible. Make a mark on the underside of the clipboard. Then, adhere Velcro to the outer edges of the board up to this mark (as seen in **Error! Reference source not found.**). The table can be adjusted to as far away as needed depending on the size of the child

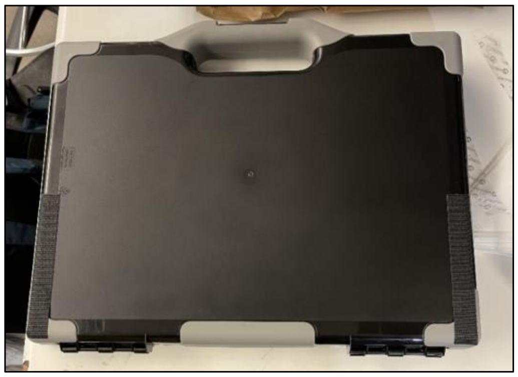


Figure 30: Velcro on Bottom of Clip Board

Attach the clipboard with the handle facing away from the child so a helper may access the contents easily.

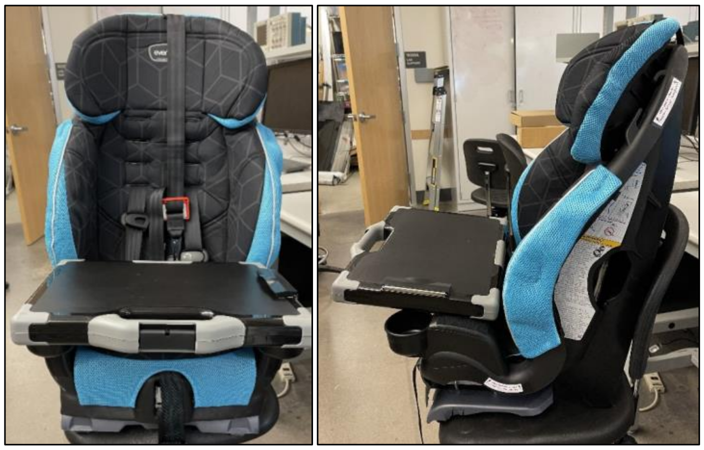


Figure 31: Table on Car Seat

Kickstand

Two kickstands are installed to the front of the Wild-Thing® frame for added stability when users enter and exit the vehicle. The kickstands can be retracted for operation without interfering with the clearance of the vehicle.

Making the Kickstand

The kickstand assembly is comprised of Velcro, PVC, and crutch ends attached with epoxy. PVC reducers (1” to ½”) should be cut with a vertical band saw to about 1 ½" wide and the top cut less than halfway. The cuts described are shown in **Error! Reference source not found.**.

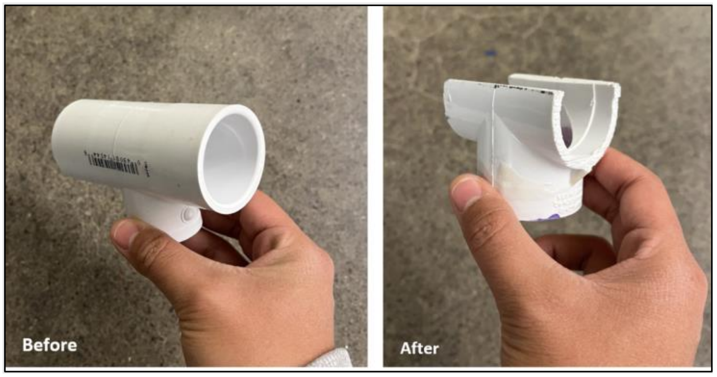


Figure 32: Before and After Cutting Reducer for Attachment

Cut ½" PVC pipe to make two 4” long pieces and connect to the reducer. Using epoxy, attach a crutch end to the end of each ½" PVC pipe. Click the cut reducers onto the front of the Wild-Thing® and attach with Velcro to secure the fit. The kickstands should easily glide around the front bumper and rest inside the vehicle when not in use.

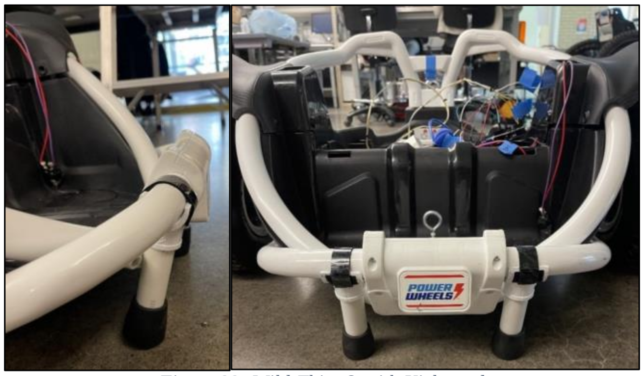


Figure 33: Wild-Thing® with Kickstand

3D Printed Joystick Supports

As previously mentioned, the cup holder attachments from the car seat are used to house the 3D printed part; a rendering of the piece is depicted below in Figure 33.

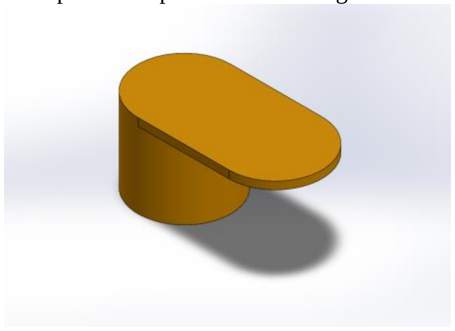
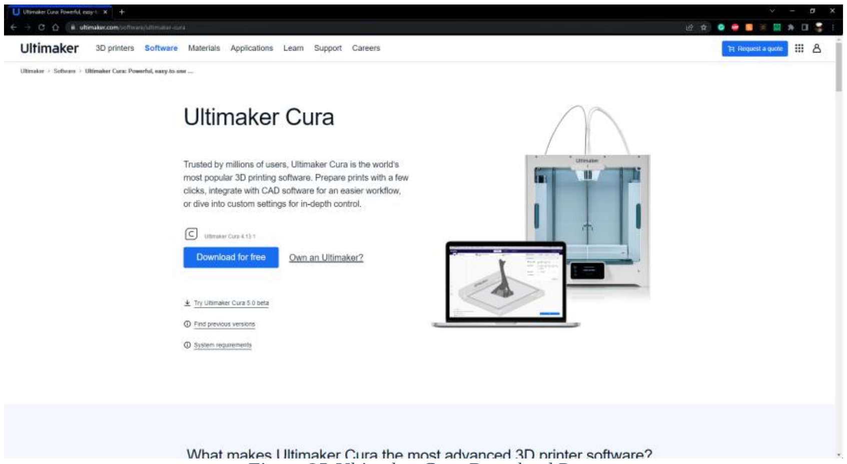


Figure 34: Arm Rest Cylindrical Insert Rendering

If the user has access to a personal 3D printing press or wants to manipulate this piece, they can find the SolidWorks file (a file with the .sldprt extension) here at []. The user can manipulate many dimensions to suit their needs. If the user is not comfortable using SolidWorks, seeking help from an experienced user is highly recommended. Once one is satisfied with the piece, it must be re-saved as a .STL file.

To 3D print, the user must download Ultimaker Cura, or Cura for short. To download Cura, the user must go to Ultimaker's website and navigate to the download page. The direct link to the download page is https://ultimaker.com/software/ultimaker-cura. The download page is depicted below in Figure 34 for reference.

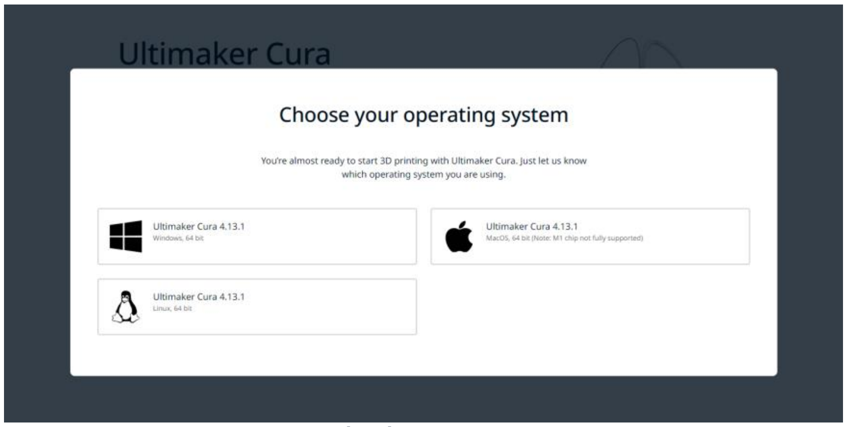

Ultimaker Cura

Trusted by millions of users, Ultimaker Cura is the world's most popular 3D printing software. Prepare prints with a few clicks, integrate with CAD software for an easier workflow, or dive into custom settings for in-depth control.

What makes Ultimaker Cura the most advanced 3D printer software?

Figure 35:Ultimaker Cura Download Page

Click "Download for Free," and the following pop-up should appear.

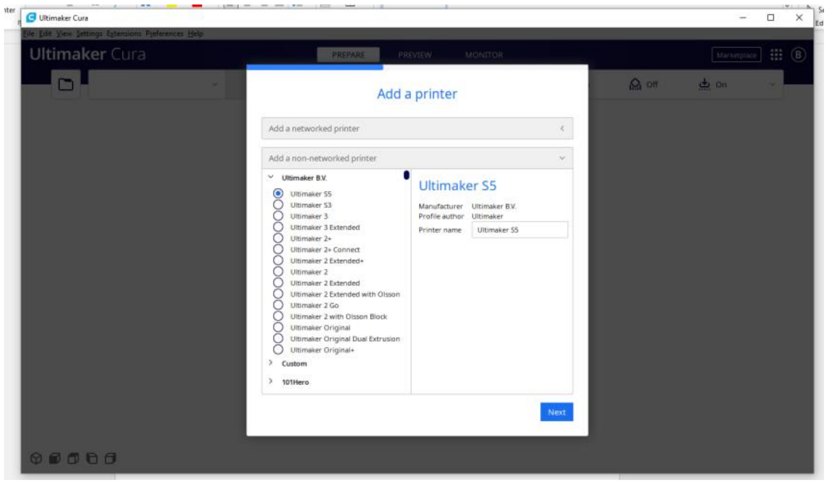

Choose your operating system

You're almost ready to start 3D printing with Ultimaker Cura. Just let us know which operating system you are using.

Figure 36: Ultimaker Cura Operating System

Click on whatever button corresponds to the user's operating system; the programs should not have any functionality differences. Once downloaded, click next for every step and wait for the program to download.

Once Cura is launched, the user must create an Ultimaker account; do not worry, it is free. After creating an account and signing in, the following prompt will appear:

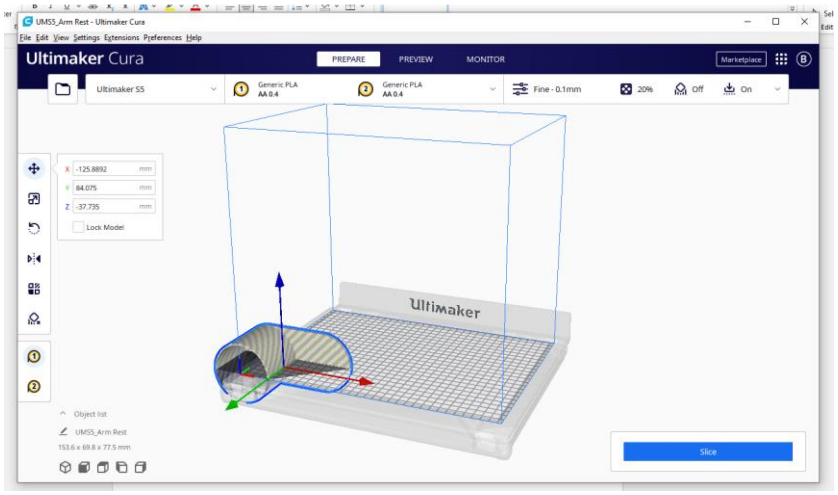


Ultimaker S5

Manufacturer Ultimaker B.V.
Profile author Ultimaker
Printer name Ultimaker S5

Figure 37: Cura Printer Dialogue

Click the “Add a non-network printer,” and a dropdown menu will appear; this can be seen above in Figure 36. These are all the many 3D printing machines available. Click on your 3D printing press model. Next, drag and drop the .STL file you created onto the app. It should look like this if done correctly:


UMS5_Arm Rest - Ultimaker Cura
File Edit View Settings Extensions Preferences Help
Ultimaker Cura
PREPARE
PREVIEW
MONITOR
Marketplace
Ultimaker
Object list
Slice

Ultimaker S5 | Generic PLA AA 0.4 | Generic PLA AA 0.4 | Fine - 0.1mm | 20% | Off | On

X -125.8892 mm
Y 84.075 mm
Z -37.735 mm
Lock Model

UMS5_Arm Rest
153.6 x 69.8 x 77.5 mm

Figure 38: .STL File in Cura

On the left-hand side is a toolbar with many different icons. The very top icon, the crossroads, is called move. Click on any of the colored arrows depicted above in Figure 37 to move the piece around the workspace. The 3rd icon down, with the counterclockwise arrow, is called rotate. Click on rotate, and a bunch of circles should appear around the part, and another pop-up menu should open.

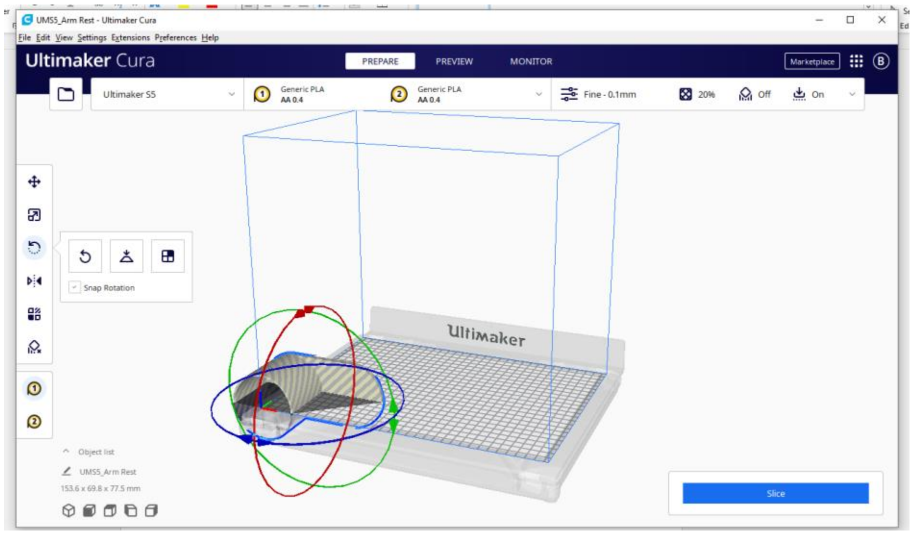


Figure 39: Cura Rotate

These circles can be clicked and dragged to change how the machine prints a part; there are infinite ways a piece can be 3D printed. However, to print a part in the quickest way possible, it must be oriented so there are no hanging ledges. An example of a hanging ledge is pictured below.

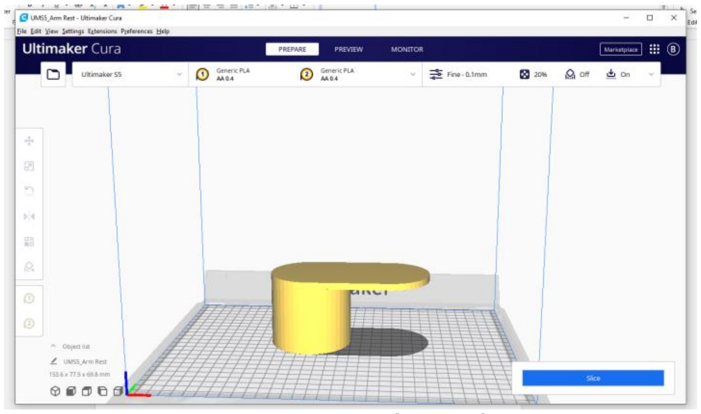


Figure 40:Hanging Ledge Example

Hanging ledges are bad because they significantly increase the amount of time required to print a piece and allow the possibility of a misprint. The most optimal way to place this piece is depicted below in Figure 40.

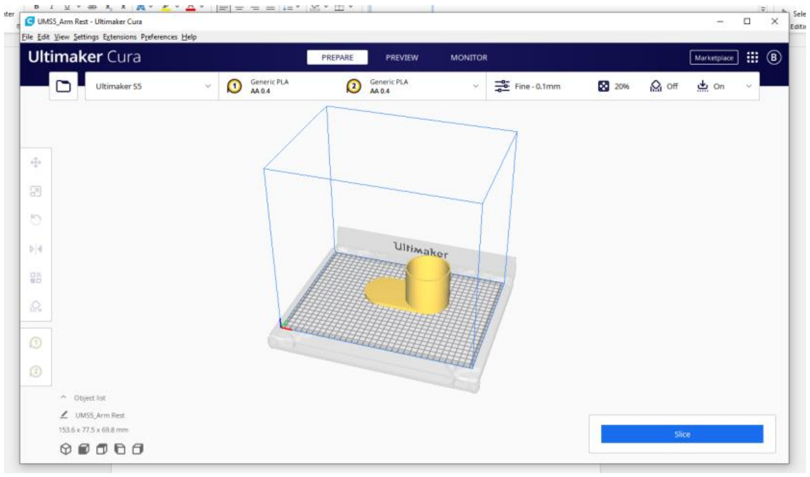


Figure 41: Optimal Printing Orientation

Click the "select face to align to build plate" button within the rotate pop-up to achieve the most optimal print orientation. It should be the rightmost icon. Click the side you want to lay flat on the workspace. Obviously, the face chosen changes from piece to piece but one should look to limit hanging ledges as much as possible.

The print speed will depend on the material, infill density, print speed, temperature, top/bottom, and wall thicknesses. These variables can be seen by clicking the bar in the top-right corner. A dropdown menu should drop:

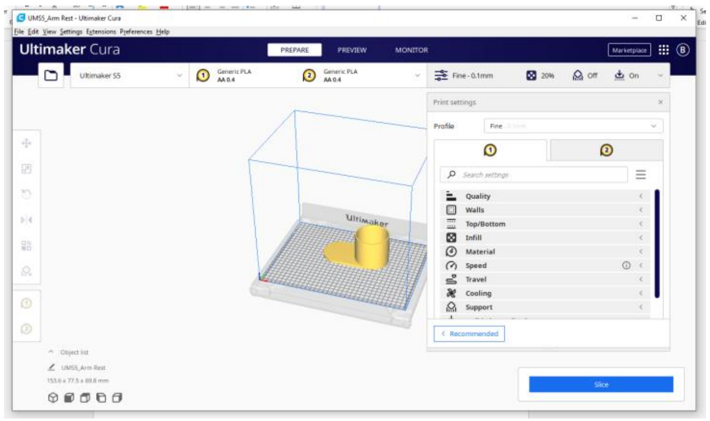


Figure 42: 3D printing variables

At this point, it is highly recommended that help is sought after for this section if one is not familiar. If the user is comfortable or has experience using 3D printing machines, they may experiment with the variables in this section. Once the user is satisfied with the printing variables, click slice and save the file onto a removable disk. The removable disk may then be inserted into the machine to print.

Electronics

Initial modifications

The initial Wild Thing vehicle contains a battery, a thinking motor control board, and a set of driving controls. The modification only requires the built-in motor's wires to remain. To implement this system, everything but those can be removed in any order. This will require unscrewing all the plastic wire guides and the factory control circuit board. Removing the wires from the board is as easy as sliding the connectors apart. In the image below the retained parts are shown, a set of colored wires for each motor, and a black and white wire coming out of the battery. Additionally, the motor wires had crimp connectors added for connection to the motor control. There is no specific order of removal.

Battery Replacement

The initial battery is replaced by a 12V Sealed Lead Acid Battery with F1 Terminals. This battery will allow the vehicle to be charged without removing the seat while providing a more efficient and reliable power source to be used daily for hours at a time. The following link is a step-by-step video of how to replace the Wild Thing battery: https://youtu.be/nQdEdroassI

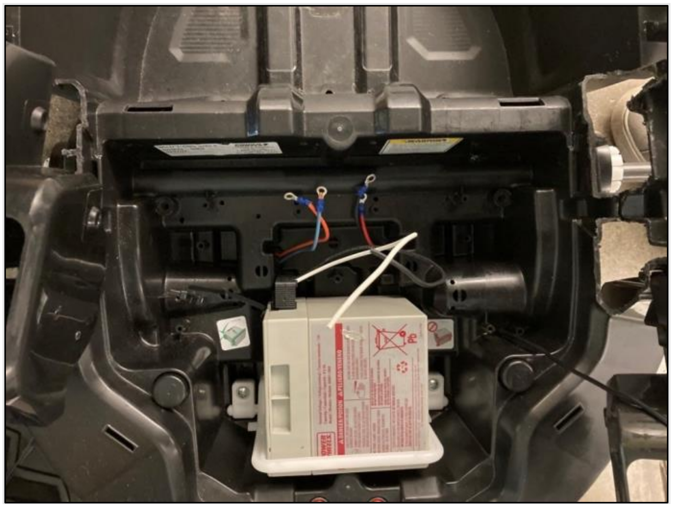


Figure 43: Initial configuration after removing factory parts.

Remote on/off

This device was purchased directly from a seller, listed in the bill of materials. It is a standard RF on/off relay with a remote control. If this specific one is unavailable, there are many substitute products. Most importantly, it is a remote-control relay with a normally-closed wiring available. Wiring this directly after the main switch in the normally-closed position allows the vehicle to be shut off remotely by a teacher or aide but does not prevent the vehicle from being turned back on by the user, for safety concerns. To simplify, a relay acts exactly like a switch in a circuit so it is wired in series right after the switch. Cut the wire after the switch, and reconnect the cut together with the relay, using the common terminal and the normally-closed terminal. The other two wires in Figure 44 below are being used to power the relay itself. This means the positive terminal of the relay is connected to the positive bus and the negative terminal is connected to the negative bus. Shown in the figure below, the black arrow points to the negative power for the relay, the red arrow points to the positive power for the relay, the green arrow points to the common terminal, and the orange arrow points to the normally-closed terminal.

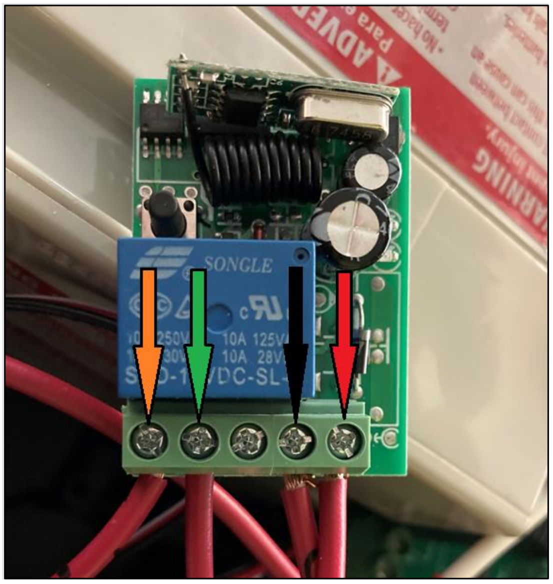


Figure 44: The wired remote-control relay

The styles of remote switch may vary from this exact one, it is just most important to wire it into series after the main physical switch through the NC (normally closed) and common. Additionally, the power in from the battery is split here to both power the relay itself and the entire vehicle.

Arduino Nano

The Arduino Nano was soldered into headers and attached to a custom PCB to easily interface with all parts of the circuit. This could be modified to have direct soldered connections; however, this would make future modifications much more difficult. Alternatively, an Arduino Uno may be used for prototyping. The Arduino Nano was chosen for its size, processing power, and functionality.

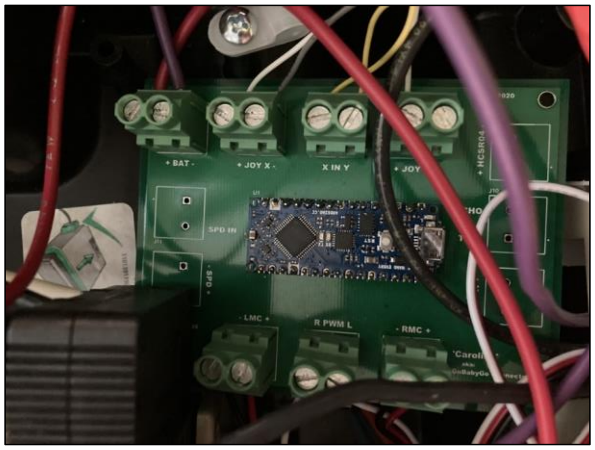


Figure 45: Arduino Board within the vehicle

Motor Control

Each motor of the vehicle has its own motor control, the diagram below shows a simplified design of a combined motor control. Each device takes in power from the battery, the motor wires, and signals from the Nano to control the motors accurately.

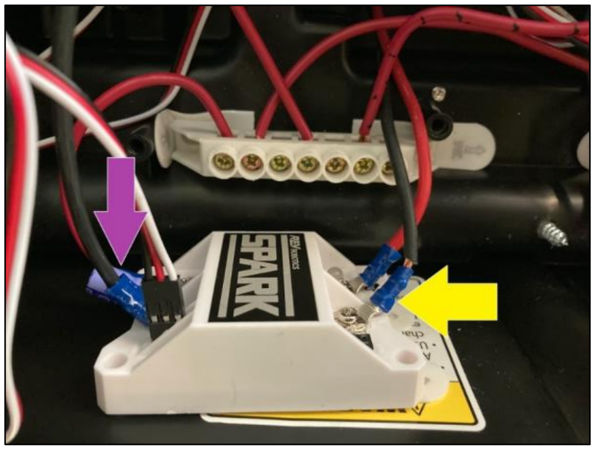


Figure 46: Motor control and power bus, both mounted with command strips.

It is also important to note that each of the motor controls has a capacitor (details listed in the bill of materials) bridging the power input wires. NOT the wires going to the motor. This is marked in Figure 46 with the purple arrow. The yellow arrow marks the motor wires, which is also written on the motor control itself, with an “M”.

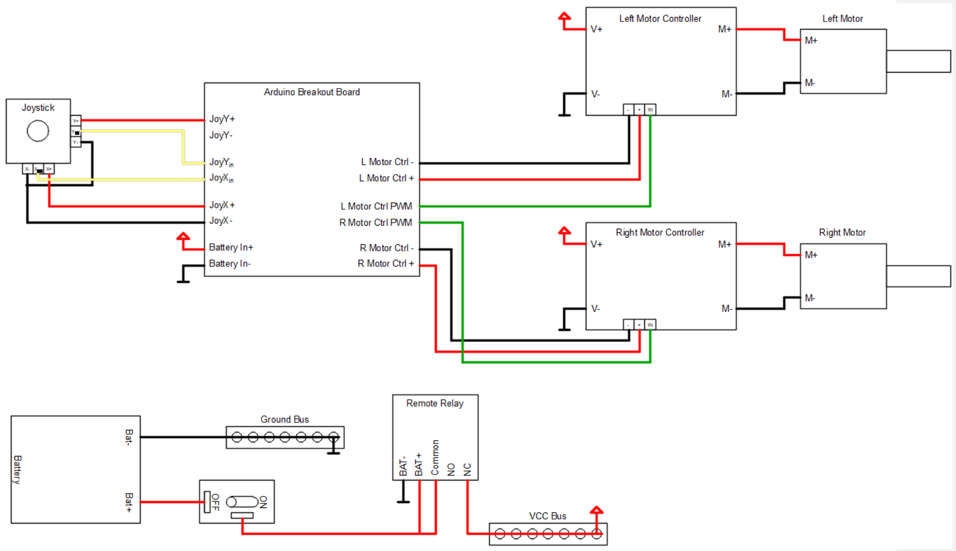


Figure 47: Circuit Flow Diagram for Implemented Circuitry

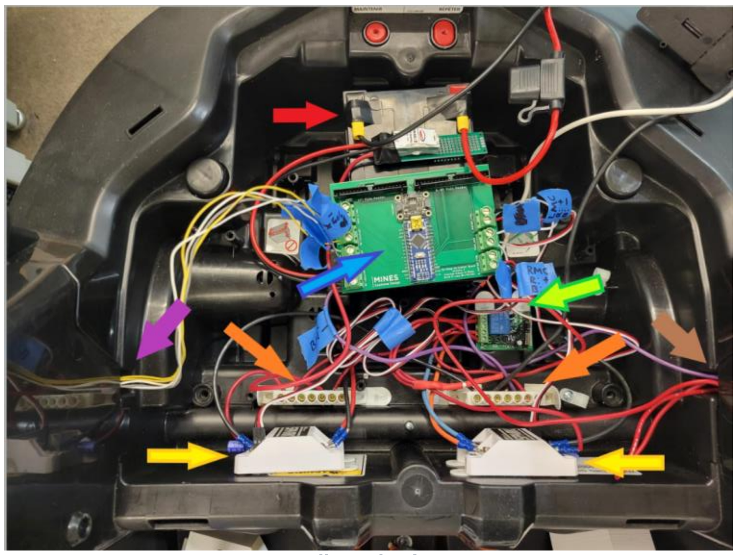


Figure 48: Full completed new circuitry.

The above figure shows the fully completed circuitry, labeled by colored arrows. The red arrow is the new battery, the blue arrow points to the Arduino control, the orange arrows point to the power busses, the yellow arrows point to the motor controls, the brown arrow points to the on/off switch wiring, the green arrow points to the remote on/off relay, and the purple arrow points to the joystick wiring.

Contact Info:

- Allison Sander: asander@mines.edu, (210) 202-7371
- James "Cameron" Canter: jccanter@mines.edu
- Morgan Nishida: mknishida@mines.edu
- Walter Rodenberger: rodenberger@mines.edu
- David Long: dlong1@mines.edu
- Misa Jefferson: mjefferson@mines.edu
- Bryan Hoang: bryanhoang@mines.edu
- Human Centered Design Studio: hcds@mines.edu

Appendix

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// TITLE:   GoBabyGo Car Controller - Simple 4 Direction
// AUTHOR:  David Long
//
// PURPOSE: This version of the car controller limits the vehicle to one action at a time, either forward, reverse, turning left, or turning right.
//          The vehicle must come to a complete stop before any change in direction is allowed. This creates a small input delay, but prevents motors
//          from getting out of sync. A future version of this program will allow more analog control to the vehicle. The code also allows interaction
//          with the user adjustable values through an LCD interface.
//
//
#include <Servo.h>
#include <SPI.h>
#include <EEPROM.h>
#include "Adafruit_GFX.h"
#include "Adafruit_TSC2007.h"
#include "Adafruit_HX8357.h"
#include <Fonts/FreeMonoBold9pt7b.h>
#include <Fonts/FreeMonoBold12pt7b.h>



//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//************************************************************************************************************************************** **
// Only enable this value when data is desired to be written to the EEPROM. The EEPROM has a limited lifespan of 100,000 writes
// so this feature should be used sparingly!!!
bool allow_EEPROM_writes = false;
//************************************************************************************************************************************** **
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////



// TODO: Add function to choose between these values or the percents stored in EEPROM. If percents are desired, they must first be
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Adjustable values: you can change these to fit the needs of each user                                                  //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                                                                                                    //
// Increase/decrease these values to change the joystick deadzones                                                       //
// (An increase in the RIGHT or FORWARD values or a decrease in the LEFT or REVERSE values increases the corresponding deadzones)   //
const uint16_t RIGHT_THRESH = 600;      // Analog reads above this value should be considered right joystick inputs                //
const uint16_t LEFT_THRESH = 400;       // Analog reads below this value should be considered left joystick inputs                 //
const uint16_t FORWARD_THRESH = 630;    // Analog reads above this value should be considered forward joystick inputs              //
const uint16_t REVERSE_THRESH = 340;    // Analog reads below this value should be considered reverse joystick inputs              //
                                                                                                                    //
// Increase/decrease these values to modify the driving characteristics                                                  //
const uint16_t SPEED_LIMIT = 206;       // Between 100 - 512: This scales down all values sent to the motors                      //
                                        // Increase value to increase the speed of the car                               //
                                                                                                                    //
const uint8_t INCREASE_RAMPING = 5;     // This prevents the car from jerking by limiting how fast its speed can increase          //
                                        // If the car is not responding/accelerating fast enough, increase this value             //
                                                                                                                    //
const uint8_t DECREASE_RAMPING = 15;    // This prevents the car from jerking by limiting how fast its speed can decrease          //
                                        // If the car is not slowing down fast enough, increase this value                       //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Set this value true to enable debug Over Serial (Make sure battery is disconnected from Arduino.
const boolean DEBUG = false;

// Set this value false if the car will not be used with the touchscreen LCD
const boolean USE_LCD = false;


// Constants for Rev Robotics Spark Motor Controller
// | Full Reverse | Proportional Reverse |      Neutral      | Proportional Forward | Full Forward |
// |  p <= 1000   |   1000 < p < 1460    | 1460 <= p <= 1540 |   1540 < p < 2000    |  2000 <= p   |
const uint16_t REVERSE_PULSE = 1000;   // Default pulse width for full reverse is 1000us
const uint16_t FORWARD_PULSE = 2000;   // Default pulse width for full forward is 2000us


// Invert left or right motor
// (If true, CW rotation as viewed from the front of the motor will be considered forward drive)
// (If false, CCW rotation as viewed from the front of the motor will be considered forward drive)
const boolean INVERT_LEFT_MOTOR = false;
const boolean INVERT_RIGHT_MOTOR = true;


// Pin Assignments
const int JOYSTICK_X = A7;       // Joystick X-input
const int JOYSTICK_Y = A6;       // Joystick Y-input
const int TFT_CS = A3;           // LCD chip select pin
const int TFT_DC = A2;           // LCD data control pin
const int TFT_RST = -1;          // Reset pins tied between LCD and Arduino
const int TOUCH_INT = 2;         // Touch screen interrupt pin
const int LEFT_MOTOR_PIN   = 9;  // digital pin 6 (PF4)
const int RIGHT_MOTOR_PIN  = 10; // digital pin 5 (PB2)


// LCD Colors
const uint16_t BLACK    = 0x0000;
const uint16_t WHITE    = 0xFFFF;
const uint16_t GREY     = 0xBDF7;
const uint16_t GREEN    = 0x07E0;
const uint16_t L_YELLOW = 0xEFB7;


// LCD Menu Elements
const uint8_t L_MENU_LINES_X0         = 5;    // X-coordinate of top left of all 4 input sensitivity menu lines
const uint8_t L_MENU_LINE1_Y0         = 65;   // Y-coordinate of top left of input sensitivity menu line 1 (forward sensitivity)
const uint8_t L_MENU_LINE2_Y0         = 113;  // Y-coordinate of top left of input sensitivity menu line 2 (reverse sensitivity)
const uint8_t L_MENU_LINE3_Y0         = 161;  // Y-coordinate of top left of input sensitivity menu line 3 (left sensitivity)
const uint8_t L_MENU_LINE4_Y0         = 209;  // Y-coordinate of top left of input sensitivity menu line 5 (right sensitivity)

const uint8_t R_MENU_LINES_X0         = 249;  // X-coordinate of top left of all 3 motion characteristics menu lines
const uint8_t R_MENU_LINE1_Y0         = 65;   // Y-coordinate of top left of motion characteristics menu line 1 (maximum speed)
const uint8_t R_MENU_LINE2_Y0         = 173;  // Y-coordinate of top left of motion characteristics menu line 2 (acceleration rate)
const uint16_t R_MENU_LINE3_Y0        = 281;  // Y-coordinate of top left of motion characteristics menu line 3 (deceleration rate)

const uint8_t MENU_HEADER_HEIGHT      = 51;   // Height (in pixels) of the header boxes for both the input sensitivity and motion characteristics menus
const uint8_t MENU_HEADER_WIDTH       = 227;  // Width (in pixels) of the header boxes for both the input sensitivity and motion characteristics menus

const uint8_t MENU_LINE_HEIGHT        = 31;   // Height (in pixels) of each box in the menu lines of both the input sensitivity and motion characteristics menus

const uint8_t SENS_TYPE_BOX_WIDTH     = 95;   // Width (in pixels) of the input sensitivity menu line sensitivity type text box (2nd column)
const uint8_t SENS_TYPE_BOX_X_OFFSET  = 40;   // Pixel offset in x-direction from top left corner of menu line to the top left of the sensitivity type textbox
const uint8_t SENS_TYPE_TEXT_X_OFFSET = 47;   // Pixel offset in x-direction from top left corner of menu line to the bottom left of the sensitivity type text
const uint8_t SENS_TYPE_TEXT_Y_OFFSET = 21;   // Pixel offset in y-direction from top left corner of menu line to the bottom left of the sensitivity type text

const uint8_t MAG_BAR_BOX_WIDTH       = 103;  // Width (in pixels) of the motion characteristics menu line magnitude bar bounding box (2nd Column)
const uint8_t MAG_BAR_BOX_X_OFFSET    = 36;   // Pixel offset in x-direction from top left corner of motion characteristics menu line to the magnitude bar bounding box

const uint8_t BUTTON_WIDTH            = 31;   // Width (in pixels) of the up and down buttons for both the input sensitivity and motion characteristics menus (1st and 3rd column)
const uint8_t UP_BUTTON_X_OFFSET      = 144;  // Pixel offset in x-direction from top left corner of the menu line to the top left corner of the up button (both menus)

const uint8_t READOUT_BOX_WIDTH       = 43;   // Width (in pixels) of the data readout textbox for both input sensitivity and motion characteristics menu lines (4th column)
const uint8_t READOUT_BOX_X_OFFSET    = 184;  // Pixel offset in x-direction from top left corner of menu line to the top left of the data readout box
const uint8_t READOUT_TEXT_X_OFFSET   = 189;  // Pixel offset in x-direction from top left corner of menu line to the bottom left of the data readout text
const uint8_t READOUT_TEXT_Y_OFFSET   = 21;   // Pixel offset in y-direction from top left corner of menu line to the bottom left of the data readout text

const uint16_t SAVE_BOX_Y0            = 261;  // Y_coordinate of top left of save values button


// Input sensitivity menu button touch screen extents
const uint16_t L_BUTTON_R1_TOUCH_X1 = 1000;  // X value of touchscreen corresponding to top left corner of buttons in row 1 of the left menu screen (buttons for forward sensitivity row)
const uint16_t L_BUTTON_R1_TOUCH_X2 = 1300;  // X value of touchscreen corresponding to bottom right corner of buttons in row 1 of the left menu screen (buttons for forward sensitivity row)

const uint16_t L_BUTTON_R2_TOUCH_X1 = 1550;  // X value of touchscreen corresponding to top left corner of buttons in row 2 of the left menu screen (buttons for reverse sensitivity row)
const uint16_t L_BUTTON_R2_TOUCH_X2 = 1850;  // X value of touchscreen corresponding to bottom right corner of buttons in row 2 of the left menu screen (buttons for reverse sensitivity row)

const uint16_t L_BUTTON_R3_TOUCH_X1 = 2100;  // X value of touchscreen corresponding to top left corner of buttons in row 3 of the left menu screen (buttons for left sensitivity row)
const uint16_t L_BUTTON_R3_TOUCH_X2 = 2400;  // X value of touchscreen corresponding to bottom right corner of buttons in row 3 of the left menu screen (buttons for left sensitivity row)

const uint16_t L_BUTTON_R4_TOUCH_X1 = 2650;  // X value of touchscreen corresponding to top left corner of buttons in row 4 of the left menu screen (buttons for right sensitivity row)
const uint16_t L_BUTTON_R4_TOUCH_X2 = 2950;  // X value of touchscreen corresponding to bottom right corner of buttons in row 4 of the left menu screen (buttons for right sensitivity row)

const uint16_t L_BUTTON_C1_TOUCH_Y1 = 3870;  // Y value of touchscreen corresponding to top left corner of buttons in column 1 of the left menu screen (up buttons for input sensitivity menu)
const uint16_t L_BUTTON_C1_TOUCH_Y2 = 3650;  // Y value of touchscreen corresponding to bottom right corner of buttons in column 1 of the left menu screen (up buttons for input sensitivity menu)

const uint16_t L_BUTTON_C2_TOUCH_Y1 = 2800;  // Y value of touchscreen corresponding to top left corner of buttons in column 2 of the left menu screen (down buttons for input sensitivity menu)
const uint16_t L_BUTTON_C2_TOUCH_Y2 = 2550;  // Y value of touchscreen corresponding to bottom right corner of buttons in column 2 of the left menu screen (down buttons for input sensitivity menu)


// Motion characteristics menu button touch screen extents
const uint16_t R_BUTTON_R1_TOUCH_X1 = 1000;  // X value of touchscreen corresponding to top left corner of buttons in row 1 of the right menu screen (buttons for maximum speed row)
const uint16_t R_BUTTON_R1_TOUCH_X2 = 1300;  // X value of touchscreen corresponding to bottom right corner of buttons in row 1 of the right menu screen (buttons for maximum speed row)

const uint16_t R_BUTTON_R2_TOUCH_X1 = 2200;  // X value of touchscreen corresponding to top left corner of buttons in row 2 of the right menu screen (buttons for acceleration rate row)
const uint16_t R_BUTTON_R2_TOUCH_X2 = 2550;  // X value of touchscreen corresponding to bottom right corner of buttons in row 2 of the right menu screen (buttons for acceleration rate row)

const uint16_t R_BUTTON_R3_TOUCH_X1 = 3400;  // X value of touchscreen corresponding to top left corner of buttons in row 1 of the right menu screen (buttons for deceleration rate row)
const uint16_t R_BUTTON_R3_TOUCH_X2 = 3700;  // X value of touchscreen corresponding to bottom right corner of buttons in row 1 of the right menu screen (buttons for deceleration rate row)

const uint16_t R_BUTTON_C1_TOUCH_Y1 = 2000;  // Y value of touchscreen corresponding to top left corner of buttons in column 1 of the right menu screen (down buttons for motion characteristics menu)
const uint16_t R_BUTTON_C1_TOUCH_Y2 = 1770;  // Y value of touchscreen corresponding to bottom right corner of buttons in column 1 of the right menu screen (down buttons for motion characteristics menu)

const uint16_t R_BUTTON_C2_TOUCH_Y1 = 860;  // Y value of touchscreen corresponding to top left corner of buttons in column 2 of the right menu screen (up buttons for motion characteristics menu)
const uint16_t R_BUTTON_C2_TOUCH_Y2 = 650;  // Y value of touchscreen corresponding to bottom right corner of buttons in column 2 of the right menu screen (up buttons for motion characteristics menu)


// Save values button touch screen extents
const uint16_t SAVE_VAL_TOUCH_X1 = 3200;
const uint16_t SAVE_VAL_TOUCH_X2 = 3500;
const uint16_t SAVE_VAL_TOUCH_Y1 = 3870;
const uint16_t SAVE_VAL_TOUCH_Y2 = 2190;


// Create LCD display object using hardware SPI
Adafruit_HX8357 tft = Adafruit_HX8357(TFT_CS, TFT_DC, TFT_RST);


// Touch screen variables
Adafruit_TSC2007 touch;
uint16_t touch_x  = 0;   // X coordinate of touch input
uint16_t touch_y  = 0;   // Y coordinate of touch input
uint16_t touch_z1 = 0;   // Pressure value 1 of touch input
```

```
uint16_t touch_z2 = 0;   // Pressure value 2 of touch input


// Time related variables
const uint16_t CONSECUTIVE_TOUCH_DELAY = 100;   // Number of milliseconds that must past before another touch input should be accepted
unsigned long timePressed;                      // The time the most recent touch input was detected
unsigned long prevTimePressed = millis();       // The time the previous touch input was detected

uint8_t forwardSens;
uint8_t reverseSens;
uint8_t leftSens;
uint8_t rightSens;
uint8_t maxSpeed;
uint8_t accRate;
uint8_t decRate;

// EEPROM Memory Addresses
uint16_t FORWARD_SENS_ADDRESS = 1;
uint16_t REVERSE_SENS_ADDRESS = 2;
uint16_t LEFT_SENS_ADDRESS = 3;
uint16_t RIGHT_SENS_ADDRESS = 4;
uint16_t MAX_SPEED_ADDRESS = 5;
uint16_t ACC_RATE_ADDRESS = 6;
uint16_t DEC_RATE_ADDRESS = 7;


// Create Servo object for both motors
Servo leftMotor;
Servo rightMotor;


void setup() {

  // Pin configurations
  pinMode(JOYSTICK_X, INPUT);
  pinMode(JOYSTICK_Y, INPUT);
  pinMode(LEFT_MOTOR_PIN,OUTPUT);
  pinMode(RIGHT_MOTOR_PIN,OUTPUT);
  pinMode(TOUCH_INT, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(TOUCH_INT), touchDetected, FALLING);

  leftMotor.attach(LEFT_MOTOR_PIN);
  rightMotor.attach(RIGHT_MOTOR_PIN);


  // If the LCD should be used
  if (USE_LCD) {
    // Read in the sensitivity and motion characteristic values from the Arduino EEPROM
    uint8_t forwardSens = EEPROM.read(FORWARD_SENS_ADDRESS);
    uint8_t reverseSens = EEPROM.read(REVERSE_SENS_ADDRESS);
    uint8_t leftSens    = EEPROM.read(LEFT_SENS_ADDRESS);
    uint8_t rightSens   = EEPROM.read(RIGHT_SENS_ADDRESS);
    uint8_t maxSpeed    = EEPROM.read(MAX_SPEED_ADDRESS);
    uint8_t accRate     = EEPROM.read(ACC_RATE_ADDRESS);
    uint8_t decRate     = EEPROM.read(DEC_RATE_ADDRESS);

    // If any of the values read from the EEPROM were >100, set them to 100. This is useful for the first program run after installing a new Arduino
    // where the EEPROM may not have been written to before (Data defaults to 255 at every unwritten EEPROM address)
    if (forwardSens > 100) forwardSens = 100;
    if (reverseSens > 100) reverseSens = 100;
    if (leftSens > 100) leftSens = 100;
    if (rightSens > 100) rightSens = 100;
    if (maxSpeed > 100) maxSpeed = 100;
    if (accRate > 100) accRate = 100;
    if (decRate > 100) decRate = 100;

    // Wait for Serial connection to be made
    Serial.begin(9600);
    while (!Serial) delay(10);

    // Wait for touchscreen to connect
    if (!touch.begin()) {
      while (1) delay(10);
    }

    ////////////////////////////////////////////////////////////////////////////////
    // Draw Initial Screen Layout
    ////////////////////////////////////////////////////////////////////////////////
    tft.begin();
    tft.setRotation(1);         // Rotate screen to landscape mode
    tft.fillScreen(WHITE);      // Set screen background to white
    tft.setTextColor(BLACK);    // Set default text color to black

    ////////////////////////////////
    // Input Sensitivity (%) Header
    ////////////////////////////////
    drawMenuBox(5, 5, MENU_HEADER_WIDTH, MENU_HEADER_HEIGHT, BLACK, GREY);
    tft.setFont(&FreeMonoBold12pt7b);
    tft.setCursor(81, 23);
    tft.println("Input");
    tft.setCursor(23, 46);
    tft.println("Sensitivity(%)");
```

```
        ////////////////////////////////
        // Forward Sensitivity Menu Line
        ////////////////////////////////
        drawLeftMenuLine(L_MENU_LINES_X0, L_MENU_LINE1_Y0, BLACK, GREY, "Forward", forwardSens);

        ////////////////////////////////
        // Reverse Sensitivity Menu Line
        ////////////////////////////////
        drawLeftMenuLine(L_MENU_LINES_X0, L_MENU_LINE2_Y0, BLACK, GREY, "Reverse", reverseSens);

        ////////////////////////////////
        // Left Sensitivity Menu Line
        ////////////////////////////////
        drawLeftMenuLine(L_MENU_LINES_X0, L_MENU_LINE3_Y0, BLACK, GREY, "Left", leftSens);

        ////////////////////////////////
        // Right Sensitivity Menu Line
        ////////////////////////////////
        drawLeftMenuLine(L_MENU_LINES_X0, L_MENU_LINE4_Y0, BLACK, GREY, "Right", rightSens);

        ////////////////////////////////
        // Save Values Button
        ////////////////////////////////
        drawMenuBox(L_MENU_LINES_X0, SAVE_BOX_Y0, MENU_HEADER_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);
        tft.setFont(&FreeMonoBold12pt7b);
        tft.setCursor(41, 284);
        tft.println("Save Values");

        ////////////////////////////////////////////////
        // Maximum Speed (%) Header, Control, and Readout
        ////////////////////////////////////////////////
        drawMenuBox(249, 5, MENU_HEADER_WIDTH, MENU_HEADER_HEIGHT, BLACK, GREY);
        tft.setFont(&FreeMonoBold12pt7b);
        tft.setCursor(273, 23);
        tft.println("Maximum Speed");
        tft.setCursor(340, 46);
        tft.println("(%)");
        drawRightMenuLine(R_MENU_LINES_X0, R_MENU_LINE1_Y0, BLACK, GREY, maxSpeed);

        ////////////////////////////////////////////////////
        // Acceleration Rate (%) Header, Control, and Readout
        ////////////////////////////////////////////////////
        drawMenuBox(249, 113, MENU_HEADER_WIDTH, MENU_HEADER_HEIGHT, BLACK, GREY);
        tft.setFont(&FreeMonoBold12pt7b);
        tft.setCursor(281, 131);
        tft.println("Acceleration");
        tft.setCursor(322, 153);
        tft.println("Rate(%)");
        drawRightMenuLine(R_MENU_LINES_X0, R_MENU_LINE2_Y0, BLACK, GREY, accRate);

        ////////////////////////////////////////////////////
        // Deceleration Rate (%) Header, Control, and Readout
        ////////////////////////////////////////////////////
        drawMenuBox(249, 221, MENU_HEADER_WIDTH, MENU_HEADER_HEIGHT, BLACK, GREY);
        tft.setFont(&FreeMonoBold12pt7b);
        tft.setCursor(281, 239);
        tft.println("Deceleration");
        tft.setCursor(322, 262);
        tft.println("Rate(%)");
        drawRightMenuLine(R_MENU_LINES_X0, R_MENU_LINE3_Y0, BLACK, GREY, decRate);
        delay(100);
        ////////////////////////////////////////////////////////////////////////////////
    } // End if
} // End setup()




void loop() {
    if (USE_LCD) {
        // TODO: May need to add flag that makes this section run only until user presses a button on the touch screen
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        // Touchscreen Interaction
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        // If the touchDetected ISR was triggered and the appropriate ammount of time has passed since the last input, register a new touch input
        if (timePressed > prevTimePressed + CONSECUTIVE_TOUCH_DELAY) {
            touch.read_touch(&touch_x, &touch_y, &touch_z1, &touch_z2);
            prevTimePressed = timePressed;

            // Determine which (if any) button was pressed
            if (touch_y < L_BUTTON_C1_TOUCH_Y1 && touch_y > L_BUTTON_C1_TOUCH_Y2) {          // A decrease button was pressed in the input sensitivity menu
                if (touch_x > L_BUTTON_R1_TOUCH_X1 && touch_x < L_BUTTON_R1_TOUCH_X2) {          // Decrease forward sensitivity
                    updateSensitivityReadout(1, 'D');
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE1_Y0, 'D', BLACK, L_YELLOW);          // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE1_Y0, 'D', BLACK, GREY);

                } else if (touch_x > L_BUTTON_R2_TOUCH_X1 && touch_x < L_BUTTON_R2_TOUCH_X2) {     // Decrease reverse sensitivity
                    updateSensitivityReadout(2, 'D');
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE2_Y0, 'D', BLACK, L_YELLOW);          // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE2_Y0, 'D', BLACK, GREY);
                } else if (touch_x > L_BUTTON_R3_TOUCH_X1 && touch_x < L_BUTTON_R3_TOUCH_X2) {      // Decrease left sensitivity
                    updateSensitivityReadout(3, 'D');
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE3_Y0, 'D', BLACK, L_YELLOW);             // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE3_Y0, 'D', BLACK, GREY);

                } else if (touch_x > L_BUTTON_R4_TOUCH_X1 && touch_x < L_BUTTON_R4_TOUCH_X2) {      // Decrease right sensitivity
                    updateSensitivityReadout(4, 'D');
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE4_Y0, 'D', BLACK, L_YELLOW);             // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0, L_MENU_LINE4_Y0, 'D', BLACK, GREY);
                }
                tft.fillRect(5, 293, 228, 27, WHITE);   // Clear the *SAVED* or *SAVING DISABLED* message area by filling with white rectangle

            } else if (touch_y < L_BUTTON_C2_TOUCH_Y1 && touch_y > L_BUTTON_C2_TOUCH_Y2) {                  // An increase button was pressed in the input sensitivity menu
                if (touch_x > L_BUTTON_R1_TOUCH_X1 && touch_x < L_BUTTON_R1_TOUCH_X2) {                         // Increase forward sensitivity
                    updateSensitivityReadout(1, 'U');
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE1_Y0, 'U', BLACK, L_YELLOW);    // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE1_Y0, 'U', BLACK, GREY);

                } else if (touch_x > L_BUTTON_R2_TOUCH_X1 && touch_x < L_BUTTON_R2_TOUCH_X2) {                  // Increase reverse sensitivity
                    updateSensitivityReadout(2, 'U');
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE2_Y0, 'U', BLACK, L_YELLOW);    // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE2_Y0, 'U', BLACK, GREY);

                } else if (touch_x > L_BUTTON_R3_TOUCH_X1 && touch_x < L_BUTTON_R3_TOUCH_X2) {                  // Increase left sensitivity
                    updateSensitivityReadout(3, 'U');
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE3_Y0, 'U', BLACK, L_YELLOW);    // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE3_Y0, 'U', BLACK, GREY);

                } else if (touch_x > L_BUTTON_R4_TOUCH_X1 && touch_x < L_BUTTON_R4_TOUCH_X2) {                  // Increase right sensitivity
                    updateSensitivityReadout(4, 'U');
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE4_Y0, 'U', BLACK, L_YELLOW);    // Flash button
                    delay(100);
                    drawButton(L_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, L_MENU_LINE4_Y0, 'U', BLACK, GREY);
                }
                tft.fillRect(5, 293, 228, 27, WHITE);   // Clear the *SAVED* or *SAVING DISABLED* message area by filling with white rectangle

            } else if (touch_y < R_BUTTON_C1_TOUCH_Y1 && touch_y > R_BUTTON_C1_TOUCH_Y2) {      // A decrease button was pressed in the motion characteristics menu
                if (touch_x > R_BUTTON_R1_TOUCH_X1 && touch_x < R_BUTTON_R1_TOUCH_X2) {            // Decrease maximum speed
                    updateMotionCharReadout(1, 'D');
                    drawButton(R_MENU_LINES_X0, R_MENU_LINE1_Y0, 'D', BLACK, L_YELLOW);             // Flash button
                    delay(100);
                    drawButton(R_MENU_LINES_X0, R_MENU_LINE1_Y0, 'D', BLACK, GREY);

                } else if (touch_x > R_BUTTON_R2_TOUCH_X1 && touch_x < R_BUTTON_R2_TOUCH_X2) {      // Decrease acceleration rate
                    updateMotionCharReadout(2, 'D');
                    drawButton(R_MENU_LINES_X0, R_MENU_LINE2_Y0, 'D', BLACK, L_YELLOW);             // Flash button
                    delay(100);
                    drawButton(R_MENU_LINES_X0, R_MENU_LINE2_Y0, 'D', BLACK, GREY);

                } else if (touch_x > R_BUTTON_R3_TOUCH_X1 && touch_x < R_BUTTON_R3_TOUCH_X2) {      // Decrease deceleration rate
                    updateMotionCharReadout(3, 'D');
                    drawButton(R_MENU_LINES_X0, R_MENU_LINE3_Y0, 'D', BLACK, L_YELLOW);             // Flash button
                    delay(100);
                    drawButton(R_MENU_LINES_X0, R_MENU_LINE3_Y0, 'D', BLACK, GREY);
                }
                tft.fillRect(5, 293, 228, 27, WHITE);   // Clear the *SAVED* or *SAVING DISABLED* message area by filling with white rectangle

            } else if (touch_y < R_BUTTON_C2_TOUCH_Y1 && touch_y > R_BUTTON_C2_TOUCH_Y2) {                      // An increase button was pressed in the motion characteristics menu
                if (touch_x > R_BUTTON_R1_TOUCH_X1 && touch_x < R_BUTTON_R1_TOUCH_X2) {                             // Increase maximum speed
                    updateMotionCharReadout(1, 'U');
                    drawButton(R_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, R_MENU_LINE1_Y0, 'U', BLACK, L_YELLOW);        // Flash button
                    delay(100);
                    drawButton(R_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, R_MENU_LINE1_Y0, 'U', BLACK, GREY);

                } else if (touch_x > R_BUTTON_R2_TOUCH_X1 && touch_x < R_BUTTON_R2_TOUCH_X2) {                  // Increase acceleration rate
                    updateMotionCharReadout(2, 'U');
                    drawButton(R_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, R_MENU_LINE2_Y0, 'U', BLACK, L_YELLOW);        // Flash button
                    delay(100);
                    drawButton(R_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, R_MENU_LINE2_Y0, 'U', BLACK, GREY);

                } else if (touch_x > R_BUTTON_R3_TOUCH_X1 && touch_x < R_BUTTON_R3_TOUCH_X2) {                  // Increase deceleration rate
                    updateMotionCharReadout(3, 'U');
                    drawButton(R_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, R_MENU_LINE3_Y0, 'U', BLACK, L_YELLOW);        // Flash button
                    delay(100);
                    drawButton(R_MENU_LINES_X0 + UP_BUTTON_X_OFFSET, R_MENU_LINE3_Y0, 'U', BLACK, GREY);
                }
                tft.fillRect(5, 293, 228, 27, WHITE);   // Clear the *SAVED* or *SAVING DISABLED* message area by filling with white rectangle
            }

            if (touch_x > SAVE_VAL_TOUCH_X1 && touch_x < SAVE_VAL_TOUCH_X2 && touch_y < SAVE_VAL_TOUCH_Y1 && touch_y > SAVE_VAL_TOUCH_Y2) {     // The save values button was pressed
                // Flash the touch box momentarily
                drawMenuBox(L_MENU_LINES_X0, SAVE_BOX_Y0, MENU_HEADER_WIDTH, MENU_LINE_HEIGHT, BLACK, L_YELLOW);
                tft.setFont(&FreeMonoBold12pt7b);
                tft.setCursor(44, 284);
                tft.println("Save Values");
                delay(150);
                drawMenuBox(L_MENU_LINES_X0, SAVE_BOX_Y0, MENU_HEADER_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);
                tft.setCursor(44, 284);
                tft.println("Save Values");

                // Save values to EEPROM
                // *CAREFUL* The EEPROM has a limited lifespan of 100,000 writes. Do not overuse this function!
                if (allow_EEPROM_writes) {
                    tft.setFont(&FreeMonoBold9pt7b);
                    tft.setCursor(78, 308);
                    tft.println("*SAVED*");
                    EEPROM.update(FORWARD_SENS_ADDRESS, forwardSens);
                    EEPROM.update(REVERSE_SENS_ADDRESS, reverseSens);
                    EEPROM.update(LEFT_SENS_ADDRESS, leftSens);
                    EEPROM.update(RIGHT_SENS_ADDRESS, rightSens);
                    EEPROM.update(MAX_SPEED_ADDRESS, maxSpeed);
                    EEPROM.update(ACC_RATE_ADDRESS, accRate);
                    EEPROM.update(DEC_RATE_ADDRESS, decRate);
                } else {
                    tft.setFont(&FreeMonoBold9pt7b);
                    tft.setCursor(25, 308);
                    tft.println("*SAVING DISABLED*");
                }
            }
          }
          ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
      } // End if(USE_LCD)


      //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
      // Vehicle operation
      //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

      // Convert the joystick axis voltages to directions
      int x_p = analogRead(JOYSTICK_X) > RIGHT_THRESH ? 1 : 0;
      int x_n = analogRead(JOYSTICK_X) < LEFT_THRESH ? 1 : 0;
      int y_p = analogRead(JOYSTICK_Y) > FORWARD_THRESH ? 1 : 0;
      int y_n = analogRead(JOYSTICK_Y) < REVERSE_THRESH ? 1 : 0;

      /////////////////////////////////////////////
      // JOYSTICK INPUT DIAGRAM                    //
      //                                         //
      //              + Y Input                  //
      //                 ____                    //
      //                |    |                   //
      //                |    |                   //
      //         _______|    |_______            //
      //  -X Input |       /``\       | + X Input  //
      //          |_______ \__/ _______|            //
      //                |    |                   //
      //                |    |                   //
      //                |____|                   //
      //                                         //
      //              -Y Input                   //
      //                                         //
      /////////////////////////////////////////////

      // set x and y to either 0, 512, or 1023 based on the joystick values where 0 is negative input, 512 is neutral (no input), and 1023 is positive input
      // Truth Table for X, Y output (D/C = Don't Care, N/P = Not Possible)
      //  _____________________________________
      // | x_p | x_n | y_p | y_n ||  x  |  y  |
      // |  0  |  0  | D/C | D/C || 512 | D/C |
      // |  0  |  1  | D/C | D/C ||  0  | D/C |
      // |  1  |  0  | D/C | D/C || 1023| D/C |
      // |  1  |  1  | D/C | D/C || N/P | D/C |
      // | D/C | D/C |  0  |  0  || D/C | 512 |
      // | D/C | D/C |  0  |  1  || D/C |  0  |
      // | D/C | D/C |  1  |  0  || D/C | 1023|
      // | D/C | D/C |  1  |  1  || D/C | N/P |
      // ``````````````````````````````````````

      // Set x and y based on truth table above
      int x = x_p ? 1023 : (x_n ? 0 : 512);
      int y = y_p ? 1023 : (y_n ? 0 : 512);

      // TODO: May be able to remove map function here
      // Map speeds to within speed limit
      x = map(x, 0, 1023, 512 - SPEED_LIMIT, 512 + SPEED_LIMIT);
      y = map(y, 0, 1023, 512 - SPEED_LIMIT, 512 + SPEED_LIMIT);

      int moveValue = 0;
      if (y > 512) {
        moveValue = y - 512;
      } else {
        moveValue = -(512 - y);
      }

      int rotateValue = 0;
      if (x > 512) {
        rotateValue = x - 512;
      } else {
        rotateValue = -(512 - x);
      }

      setLeftRightMotorSpeeds(moveValue, rotateValue);
```

```
    delay(30); // Make loop run approximately 50hz
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
} // End loop()




////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   Draws a box with a 2 pixel wide border and solid fill color
 *@param:   x0 -> Top left corner x coordinate
 *@param:   y0 -> Top left corner y coordinate
 *@param    w  -> Width in pixels
 *@param    h  -> Height in pixels
 *@param    border_color -> 16-bit 5-6-5 Color to draw border with
 *@param    fill_color -> 16-bit 5-6-5 Color to fill with
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void drawMenuBox(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t border_color, uint16_t fill_color) {
    // Draw 2-pixel thick border rectangle
    tft.drawRect(x0, y0, w, h, border_color);
    tft.drawRect(x0+1, y0+1, w-2, h-2, border_color);

    // Fill in border with color
    tft.fillRect(x0+2, y0+2, w-4, h-4, fill_color);

} // End drawMenuBox()




////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   Draws initial menu items for the left side of the screen
 *@param:   x0 -> Top left corner x coordinate
 *@param:   y0 -> Top left corner y coordinate
 *@param    border_color  -> the color the box borders should be drawn with
 *@param    fill_color  -> the color the boxes should be filled with
 *@param    menuDataType -> The text that should be printed to show what value the menu item line is for
 *@param    menuDataVal -> The initial value that should be displayed in the data readout textbox
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void drawLeftMenuLine(uint16_t x0, uint16_t y0, uint16_t border_color, uint16_t fill_color, String menuDataType, uint8_t menuDataVal) {
    tft.setTextColor(BLACK);
    tft.setFont(&FreeMonoBold9pt7b);

    drawButton(x0, y0, 'D', border_color, fill_color);                                                        // Draw decrease button

    drawMenuBox(x0 + SENS_TYPE_BOX_X_OFFSET, y0, SENS_TYPE_BOX_WIDTH, MENU_LINE_HEIGHT, border_color, fill_color);  // Draw data type textBox
    tft.setCursor(x0 + SENS_TYPE_TEXT_X_OFFSET, y0 + SENS_TYPE_TEXT_Y_OFFSET);                                    // Set cursor to proper position in data type textBox
    tft.println(menuDataType);                                                                                  // Print menuDataType string to screen

    drawButton(x0 + UP_BUTTON_X_OFFSET, y0, 'U', border_color, fill_color);                                     // Draw increase button

    drawMenuBox(x0 + READOUT_BOX_X_OFFSET, y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, border_color, fill_color);      // Draw value readout textBox
    tft.setCursor(x0 + READOUT_TEXT_X_OFFSET, y0 + READOUT_TEXT_Y_OFFSET);                                        // Set cursor to proper position in data readout textBox
    tft.println(String(menuDataVal));                                                                           // Convert menuDataVal to string and print to screen

} // End drawLeftMenuLine()




////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   Draws initial menu items for the right side of the screen
 *@param:   x0 -> Top left corner x coordinate
 *@param:   y0 -> Top left corner y coordinate
 *@param    border_color  -> the color the box borders should be drawn with
 *@param    fill_color  -> the color the boxes be filled with
 *@param    menuDataVal -> The initial value that should be displayed in the data readout textbox
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void drawRightMenuLine(uint16_t x0, uint16_t y0, uint16_t border_color, uint16_t fill_color, uint8_t menuDataVal) {
    tft.setTextColor(BLACK);
    tft.setFont(&FreeMonoBold9pt7b);

    drawButton(x0, y0, 'D', border_color, fill_color);                                                       // Draw decrease button

    drawMenuBox(x0 + MAG_BAR_BOX_X_OFFSET, y0, MAG_BAR_BOX_WIDTH, MENU_LINE_HEIGHT, border_color, fill_color);  // Draw data type textBox
    drawMagnitudeBar(x0 + MAG_BAR_BOX_X_OFFSET, y0, menuDataVal);

    drawButton(x0 + UP_BUTTON_X_OFFSET, y0, 'U', border_color, fill_color);                                  // Draw increase button

    drawMenuBox(x0 + READOUT_BOX_X_OFFSET, y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, border_color, fill_color);  // Draw value readout textBox
    tft.setCursor(x0 + READOUT_TEXT_X_OFFSET, y0 + READOUT_TEXT_Y_OFFSET);                                     // Set cursor to proper position in data readout textBox
    tft.println(String(menuDataVal));                                                                        // Print menuDataVal string to screen
} // End drawRightMenuLine()




////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   Draws an arrow button at given coordinates
 *@param:   x0 -> Top left corner x coordinate
 *@param:   y0 -> Top left corner y coordinate
 *@param    buttonType -> What button to draw ('U' = Up arrow, 'D' = down arrow)
 *@param    border_color  -> the color the bounding box borders and button arrow should be drawn with
 *@param    fill_color  -> the color the bounding box should be filled with
*/
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void drawButton(uint16_t x0, uint16_t y0, char buttonType, uint16_t border_color, uint16_t fill_color) {
   drawMenuBox(x0, y0, BUTTON_WIDTH, MENU_LINE_HEIGHT, border_color, fill_color);
   if (buttonType == 'D') {                                                          // Draw decrease button
      tft.fillTriangle(x0 + 6, y0 + 17, x0 + 25, y0 + 17, x0 + 15, y0 + 27, BLACK);          // Draw down arrow triangle
      tft.fillRect(x0 + 12, y0 + 5, 8, 13, BLACK);                                          // Draw down arrow rectangle
   } else if (buttonType == 'U') {                                                   // Draw increase button
      tft.fillTriangle(x0 + 6, y0 + 14, x0 + 25, y0 + 14, x0 + 15, y0 + 4, BLACK);         // Draw up arrow triangle
      tft.fillRect(x0 + 12, y0 + 14, 8, 13, BLACK);                                         // Draw up arrow rectangle
   }
} // End drawButton()




/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   Draws the magnitude bar made up of 25 segments where the number of segments filled with a color gradient represents the %magnitude
 *@param:   x0 -> Top left corner x coordinate
 *@param:   y0 -> Top left corner y coordinate
 *@param    percentMagnitude -> The percentage magnitude the bar should represent (how many segments should be filled with a gradient)
*/
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void drawMagnitudeBar(uint16_t x0, uint16_t y0, uint8_t percentMagnitude) {

   // Convert percentage into number of bars where each bar = 4% (cast to float to get better rounding then cast back to int)
   uint8_t numSegmentsToFill = int(round(float(percentMagnitude) / 4.0));

   // If percentMagnitude is not zero, at least one bar should be filled
   if (percentMagnitude > 0 && numSegmentsToFill == 0) {
      numSegmentsToFill = 1;
   }

   uint8_t currBar = 1;
   uint16_t currBarX0 = x0 + 2;
   uint16_t currBarY0 = y0 + 27;
   uint8_t currBarHeight = 2;
   uint16_t currBarColor = GREEN;

   // If there are bars to fill, the initial color is green, otherwise the initial color is white.
   if (numSegmentsToFill == 0) {
      currBarColor = WHITE;
   } else {
      currBarColor = GREEN;
   }
   // Logic to determine what color each bar should be filled with (B denotes a binary, D a decimal, and 0x a hexadecimal value)
   //
   // Colors on the LCD are packed into a single 16-bit number where the the 5 most significant bits are for red, the middle 6 bits are for green, and the 5 least significant bits are for blue.
   // The magnitude bar is filled with a gradient from green to red where the first color is (Rd=B00000 Gr=B111111 Bl=B00000 -> 0x07E0) the color at the middle of the gradient is (Rd=B11111 Gn=B111111 Bl=B00000)
   // and the last color is (Rd=B11111 Gn=B000000 Bl=B00000). Since there are 25 segments, the transition from start to center of the gradient should take 12 segments and the transition from center to end
   // of the gradient should take the remaining 13 segments.
   //
   // A transition from Rd=B00000=D0 to Rd=B11111=D31 over 12 segments requires a change of 2.58 per segment. The value can only be increased by integers, however, so the red values of the first 2 segments
   // of the transition will be increased by 2 and the last 10 will be increased by 3. Since the red portion of the packed 16-bit RGB color is represented by the 5 most significant bits, an increase of 2 in
   // the red value is equivalent to an increase of (B0001000000000000=0x1000) for the packed color value and an increase of 3 in the red value is equivalent to an increase of (B0001100000000000=0x1800) to
   // the packed color value.
   //
   // The transition from the center to the end of the gradient represents a change from Bl=B111111=D63 to Bl=000000=D0 over 13 segments requires a change of 4.85 per segment. This will be split up such that
   // the blue values of the first 11 segments will be decreased by 5 and the last 2 will be decreased by 4. A decrease of 5 in the blue value is equivalent to a decrease of (B0000000010100000=0x00A0) for
   // the packed color value and a decrease of 4 in the blue value is equivalent to a decrease of (B0000000010000000=0x0080) for the the packed color value.
   //
   // The gradient stops and the remaining segments are filled with white once the desired number of segments are filled

   for (currBar = 1; currBar <= 25; currBar++) {
      // Draw the segment
      tft.fillRect(currBarX0, currBarY0, 3, currBarHeight, currBarColor);

      // Increase the dimensions of the next segment to be drawn
      currBarX0 += 4;
      currBarY0 -= 1;
      currBarHeight += 1;

      // Find what color to make the next segment
      if (currBar < numSegmentsToFill) {  // If there are still more bars to fill with the gradient
         if (currBar <= 2) {                // Increase the red value of segments [1-2] by D2
            currBarColor += 0x1000;
         } else if (currBar < 12) {         // Increase the red value of segments [3-12] by D3
            currBarColor += 0x1800;
         } else if (currBar < 23) {         // Decrease the blue value of segments [13-23] by D5
            currBarColor -= 0x00A0;
         } else {                           // Decrease the blue value of segments [24-25] by D4
            currBarColor -= 0x0080;
         }
      } else {                              // Else if there are no more bars to fill with the gradient
         currBarColor = WHITE;              // Fill remaining bars with white
      }
   }
} // End drawMagnitudeBar()
```

```
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   Increases or decreases sensitivity value and updates it cooresponding readout on the screen
 *@param:   row -> Which row of the input sensitivity menu to update (1=forward, 2=reverse, 3=left, 4=right)
 *@param:   upOrDown -> Whether to increase or decrease the data at row ('U' = increase, 'D' = decrease)
*/
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void updateSensitivityReadout(uint8_t row, char upOrDown) {
  tft.setTextColor(BLACK);
  tft.setFont(&FreeMonoBold9pt7b);
  switch (row) {
    // Update forward sensitivity readout
    case 1:
      if (upOrDown == 'U' && forwardSens != 100) {    // If the data should be increased
        forwardSens += 1;                             // Increase by 1% with maximum of 100%
      }
      else if (upOrDown == 'D' && forwardSens != 0) { // If the data should be decreased
        forwardSens -= 1;                             // Decrease by 1% with minimum of 0%
      }
      drawMenuBox(L_MENU_LINES_X0 + READOUT_BOX_X_OFFSET, L_MENU_LINE1_Y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);  // Clear text
      tft.setCursor(L_MENU_LINES_X0 + READOUT_TEXT_X_OFFSET, L_MENU_LINE1_Y0 + READOUT_TEXT_Y_OFFSET);     // Set curser to correct location
      tft.println(forwardSens);  // Print new data to the LCD
      break;

    // Update reverse sensitivity readout
    case 2:
      if (upOrDown == 'U' && reverseSens != 100) {    // If the data should be increased
        reverseSens += 1;                             // Increase by 1% with maximum of 100%
      }
      else if (upOrDown == 'D' && reverseSens != 0) { // If the data should be decreased
        reverseSens -= 1;                             // Decrease by 1% with minimum of 0%
      }
      drawMenuBox(L_MENU_LINES_X0 + READOUT_BOX_X_OFFSET, L_MENU_LINE2_Y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);  // Clear text
      tft.setCursor(L_MENU_LINES_X0 + READOUT_TEXT_X_OFFSET, L_MENU_LINE2_Y0 + READOUT_TEXT_Y_OFFSET);     // Set curser to correct location
      tft.println(reverseSens);  // Print new data to the LCD
      break;

    // Update left sensitivity readout
    case 3:
      if (upOrDown == 'U' && leftSens != 100) {       // If the data should be increased
        leftSens += 1;                                // Increase by 1% with maximum of 100%
      }
      else if (upOrDown == 'D' && leftSens != 0) {    // If the data should be decreased
        leftSens -= 1;                                // Decrease by 1% with minimum of 0%
      }
      drawMenuBox(L_MENU_LINES_X0 + READOUT_BOX_X_OFFSET, L_MENU_LINE3_Y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);  // Clear text
      tft.setCursor(L_MENU_LINES_X0 + READOUT_TEXT_X_OFFSET, L_MENU_LINE3_Y0 + READOUT_TEXT_Y_OFFSET);     // Set curser to correct location
      tft.println(leftSens);  // Print new data to the LCD
      break;

    // Update right sensitivity readout
    case 4:
      if (upOrDown == 'U' && rightSens != 100) {      // If the data should be increased
        rightSens += 1;                               // Increase by 1% with maximum of 100%
      }
      else if (upOrDown == 'D' && rightSens != 0) {   // If the data should be decreased
        rightSens -= 1;                               // Decrease by 1% with minimum of 0%
      }
      drawMenuBox(L_MENU_LINES_X0 + READOUT_BOX_X_OFFSET, L_MENU_LINE4_Y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);  // Clear text
      tft.setCursor(L_MENU_LINES_X0 + READOUT_TEXT_X_OFFSET, L_MENU_LINE4_Y0 + READOUT_TEXT_Y_OFFSET);     // Set curser to correct location
      tft.println(rightSens);  // Print new data to the LCD
      break;
  } // End Switch
} // End updateSensitivityReadout()




///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   Increases or decreases motion characteristic value and updates it cooresponding readout on the screen. The Magnitude bar of the chosen row is also
 *          updated to reflect the new value.
 *@param:   row -> Which row of the motion characteristics menu to update (1=max speed, 2=acceleration rate, 3=deceleration rate)
 *@param:   upOrDown -> Whether to increase or decrease the data at row ('U' = increase, 'D' = decrease)
*/
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void updateMotionCharReadout(uint8_t row, char upOrDown) {
  tft.setTextColor(BLACK);
  tft.setFont(&FreeMonoBold9pt7b);

  switch (row) {
    // Update maximum speed readout
    case 1:
      if (upOrDown == 'U' && maxSpeed != 100) {    // If the data should be increased
        maxSpeed += 1;                             // Increase by 1% with maximum of 100%
      }
      else if (upOrDown == 'D' && maxSpeed != 0) { // If the data should be decreased
        maxSpeed -= 1;                             // Decrease by 1% with minimum of 0%
      }
      drawMenuBox(R_MENU_LINES_X0 + READOUT_BOX_X_OFFSET, R_MENU_LINE1_Y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);  // Clear text
      tft.setCursor(R_MENU_LINES_X0 + READOUT_TEXT_X_OFFSET, R_MENU_LINE1_Y0 + READOUT_TEXT_Y_OFFSET);     // Set curser to correct location
      tft.println(maxSpeed);  // Print new data to the LCD
      drawMagnitudeBar(R_MENU_LINES_X0 + MAG_BAR_BOX_X_OFFSET, R_MENU_LINE1_Y0, maxSpeed);    // Update the maxSpeed magnitude bar
```

```
            break;

        // Update acceleration rate readout
        case 2:
            if (upOrDown == 'U' && accRate != 100) {    // If the data should be increased
                accRate += 1;                           // Increase by 1% with maximum of 100%
            }
            else if (upOrDown == 'D' && accRate != 0) { // If the data should be decreased
                accRate -= 1;                           // Decrease by 1% with minimum of 0%
            }
            drawMenuBox(R_MENU_LINES_X0 + READOUT_BOX_X_OFFSET, R_MENU_LINE2_Y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);  // Clear text
            tft.setCursor(R_MENU_LINES_X0 + READOUT_TEXT_X_OFFSET, R_MENU_LINE2_Y0 + READOUT_TEXT_Y_OFFSET);     // Set curser to correct location
            tft.println(accRate);  // Print new data to the LCD
            drawMagnitudeBar(R_MENU_LINES_X0 + MAG_BAR_BOX_X_OFFSET, R_MENU_LINE2_Y0, accRate);    // Update the maxSpeed magnitude bar
            break;

        // Update deceleration rate readout
        case 3:
            if (upOrDown == 'U' && decRate != 100) {       // If the data should be increased
                decRate += 1;                              // Increase by 1% with maximum of 100%
            }
            else if (upOrDown == 'D' && decRate != 0) {    // If the data should be decreased
                decRate -= 1;                              // Decrease by 1% with minimum of 0%
            }
            drawMenuBox(R_MENU_LINES_X0 + READOUT_BOX_X_OFFSET, R_MENU_LINE3_Y0, READOUT_BOX_WIDTH, MENU_LINE_HEIGHT, BLACK, GREY);  // Clear text
            tft.setCursor(R_MENU_LINES_X0 + READOUT_TEXT_X_OFFSET, R_MENU_LINE3_Y0 + READOUT_TEXT_Y_OFFSET);     // Set curser to correct location
            tft.println(decRate);  // Print new data to the LCD
            drawMagnitudeBar(R_MENU_LINES_X0 + MAG_BAR_BOX_X_OFFSET, R_MENU_LINE3_Y0, decRate);    // Update the maxSpeed magnitude bar
            break;
    } // End switch
} // End updateMotionCharReadout()




////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*@brief:   ISR called when touch is detected. Finds the time at which the touch occured
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void touchDetected() {
    timePressed = millis();
} // End touchDetected()




//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* @brief:  Get left and right motor speeds
 * @desc :  Convert moveValue and rotateValue to left and right motor speeds to send to the drive function
 * @param:  moveValue   -> Value indicating speed and direction of movement, + value indicates moving forward, - value indicates moving backwards
 *                      ranges from [512 - SPEED_LIMIT, 512 + SPEED_LIMIT]
 * @param:  rotateValue -> Value indicating speed and direction of rotation, + value indicates CW rotation, - value indicates CCW rotation
 *                      ranges from [512 - SPEED_LIMIT, 512 + SPEED_LIMIT]
 * @return: None
 */
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// TODO: Try making these values statics (if they're only changed by the setLeftRightMotorSpeeds function)
// These variables are used to keep track of changes in direction in order to reset the speeds sent to the motor before allowing additional input
int prevLeftMotorSpeed = 0;
int prevRightMotorSpeed = 0;
bool directionChanged = false;
bool leftResetComplete = true;
bool rightResetComplete = true;

void setLeftRightMotorSpeeds(int moveValue, int rotateValue) {
    int leftMotorSpeed = 0;
    int rightMotorSpeed = 0;

    // Driving forward
    if (moveValue > 0.0) {
        leftMotorSpeed = moveValue;
        rightMotorSpeed = moveValue;
    }
    // Driving backwards
    else if (moveValue < 0.0) {
        leftMotorSpeed = moveValue;
        rightMotorSpeed = moveValue;
    }
    // Turning left
    else if (rotateValue < 0.0) {
        leftMotorSpeed = rotateValue;
        rightMotorSpeed = -rotateValue;
    }
    // Turning right
    else if (rotateValue > 0.0) {
        leftMotorSpeed = rotateValue;
        rightMotorSpeed = -rotateValue;
    }

    // Find out if a direction change occured, then store the previous speed values
    if (leftMotorSpeed != prevLeftMotorSpeed || rightMotorSpeed != prevRightMotorSpeed) {
        directionChanged = true;
        leftResetComplete = false;
        rightResetComplete = false;
    }
    prevLeftMotorSpeed = leftMotorSpeed;
    prevRightMotorSpeed = rightMotorSpeed;

    // Remap Motor Speed values to range [0, 1023] (Negative values map to [0, 512], positive values map to [513, 1023])
    int remapLeftMotorSpeed = map(leftMotorSpeed, -512, 512, 0, 1023);
    int remapRightMotorSpeed = map(rightMotorSpeed, -512, 512, 0, 1023);

    // Drive the motors at the given speeds
    drive(remapLeftMotorSpeed, remapRightMotorSpeed);

} // End setLeftRightMotorSpeeds




////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* @brief: Send info to motors to drive the vehicle
 * @desc:  Add or subtract nextLeftSpeed and nextRightSpeed pulse width offsets from default pulse widths to control speed and direction of each
 *         motor. See table above for coorelation between pulse width and motor direction/speed. Acceleration is limited by a ramping constant.
 * @param: nextLeftSpeed  -> The pulse width offset representing the next speed the left motor should drive at
 * @param: nextRightSpeed -> The pulse width offset representing the next speed the right motor should drive at
 */
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// TODO: Try making these values statics (if they're only changed by the drive function)
// These two values are used by the drive function to track the previous cycle speeds in order to ramp the speed up or down
int prevLeft = 500;
int prevRight = 500;
const int NEUTRAL_SPEED = 500;    // Speed value that represents motor neutral (no motion)

void drive(int nextLeftSpeed, int nextRightSpeed) {
    int leftResetSpeed = prevLeft;      // The current speed of the left motor while it is in its reset cycle
    int rightResetSpeed = prevRight;    // The current speed of the right motor while it is in its reset cycle
    int leftMotorPulse = 0;             // Pulsewidth (in us) to send to left motor controller
    int rightMotorPulse = 0;            // Pulsewidth (in us) to send to right motor controller

    // If a direction change has occured, reset motor speeds to neutral before accepting new input from the joystick
    if (directionChanged) {
        resetMotorSpeeds();
    }

    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // LEFT MOTOR CONTROL
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

    // variable controls the next speed the left motor should drive at
    int leftDriveSpeed = map(nextLeftSpeed, 0, 1023, 0, FORWARD_PULSE - REVERSE_PULSE);

    // If left motor is signaled to increase from its previous speed by at least RAMPING constant, increase its speed by RAMPING constant
    if(leftDriveSpeed > prevLeft + INCREASE_RAMPING) {
        leftDriveSpeed = prevLeft + INCREASE_RAMPING;
    }
    // If left motor is signaled to decrease from its previous speed by at least RAMPING constant, decrease its speed by RAMPING constant
    else if(leftDriveSpeed < prevLeft - INCREASE_RAMPING) {
        leftDriveSpeed = prevLeft - INCREASE_RAMPING;
    }

    // Calculate pulse width to send to right motor controller
    if(INVERT_LEFT_MOTOR) {  // If left motor is inverted
        leftMotorPulse = FORWARD_PULSE - leftDriveSpeed;
    } else {
        leftMotorPulse = REVERSE_PULSE + leftDriveSpeed;
    }

    // Send PWM signal to the left motor (As long as DEBUG mode is not enabled)
    if(!DEBUG) {
        leftMotor.writeMicroseconds(leftMotorPulse);
    }

    // Store previous speed value of left motor
    prevLeft = leftDriveSpeed;
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // RIGHT MOTOR CONTROL
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

    // variable controls the next speed the right motor should drive at
    int rightDriveSpeed = map(nextRightSpeed, 0, 1023, 0, FORWARD_PULSE - REVERSE_PULSE);

    // If right motor is signaled to increase from previous speed by at least RAMPING constant, increase speed by RAMPING constant
    if(rightDriveSpeed > prevRight + INCREASE_RAMPING) {
        rightDriveSpeed = prevRight + INCREASE_RAMPING;
    }
    // If right motor is signaled to decrease from previous speed by at least RAMPING constant, decrease speed by RAMPING constant
    else if(rightDriveSpeed < prevRight - INCREASE_RAMPING) {
        rightDriveSpeed = prevRight - INCREASE_RAMPING;
    }

    // Calculate pulse width to send to right motor controller
    if(INVERT_RIGHT_MOTOR) {  // If right motor is inverted
```

```
        rightMotorPulse = FORWARD_PULSE - rightDriveSpeed;
    } else {
        rightMotorPulse = REVERSE_PULSE + rightDriveSpeed;
    }

    // Send PWM signal to the right motor (As long as DEBUG mode is not enabled)
    if (!DEBUG) {
        rightMotor.writeMicroseconds(rightMotorPulse);
    }
    // Store previous speed value of right motor
    prevRight = rightDriveSpeed;
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
} // End drive()



////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* @brief: Bring left and right motors to a stop
 * @desc:  Similar to the drive function, but contains its own loop to allow the speed of both left and right motors to be completely reset to
 *         neutral before accepting further inputs. This function is called when a direction change is signaled by the joystick inputs. This keeps
 *         both motors in sync with eachother to avoid drifting of the vehicle
 */
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void resetMotorSpeeds() {
    int leftResetSpeed = prevLeft;      // The current speed of the left motor while it is in its reset cycle
    int rightResetSpeed = prevRight;    // The current speed of the right motor while it is in its reset cycle

    // While direction changed flag is true, reset left and right motors to zero speed before allowing more speed input
    while (directionChanged) {

        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        // RESET LEFT MOTOR
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        if (leftResetSpeed > NEUTRAL_SPEED + DECREASE_RAMPING) {
            leftResetSpeed = prevLeft - DECREASE_RAMPING;
        } else if (leftResetSpeed < NEUTRAL_SPEED - DECREASE_RAMPING) {
            leftResetSpeed = prevLeft + DECREASE_RAMPING;
        } else {
            leftResetComplete = true;   // Signal that the left motor has come to a stop
        }
        // Calculate pulse width to send to right motor controller
        int leftMotorPulse = 0;
        if(INVERT_LEFT_MOTOR) {  // If left motor is inverted
            leftMotorPulse = FORWARD_PULSE - leftResetSpeed;
        } else {
            leftMotorPulse = REVERSE_PULSE + leftResetSpeed;
        }
        // Send PWM signal to the left motor (As long as DEBUG mode is not enabled)
        if(!DEBUG) {
            leftMotor.writeMicroseconds(leftMotorPulse);
        }
        // Store previous speed value of left motor
        prevLeft = leftResetSpeed;
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        // RESET RIGHT MOTOR
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        if (rightResetSpeed > NEUTRAL_SPEED + DECREASE_RAMPING) {
            rightResetSpeed = prevRight - DECREASE_RAMPING;
        } else if (rightResetSpeed < NEUTRAL_SPEED - DECREASE_RAMPING) {
            rightResetSpeed = prevRight + DECREASE_RAMPING;
        } else {
            rightResetComplete = true;  // Signal that the right motor has come to a stop
        }
        // Calculate pulse width to send to right motor controller
        int rightMotorPulse = 0;
        if(INVERT_RIGHT_MOTOR) {  // If right motor is inverted
            rightMotorPulse = FORWARD_PULSE - rightResetSpeed;
        } else {
            rightMotorPulse = REVERSE_PULSE + rightResetSpeed;
        }
        // Send PWM signal to the right motor (As long as DEBUG mode is not enabled)
        if(!DEBUG) {
            rightMotor.writeMicroseconds(rightMotorPulse);
        }
        // Store previous speed value of left motor
        prevRight = rightResetSpeed;
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

        // If both left and right motors have come to a stop, signal that they are ready to receive new inputs
        if (leftResetComplete && rightResetComplete) {
            directionChanged = false;
        }
        delay(30);  // Make loop run at approximately 50Hz
    }
} // End resetMotorSpeeds()



////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* @brief: Print debug information to the serial monitor
```

```
 */
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void debug(String s, int value){
  if(DEBUG){
    Serial.print(s);
    Serial.print(": ");
    Serial.println(value);
  }
} // End debug()
```