Spring 2023

Safe Kids Car

Nathan Keenan
njk53@uakron.edu

Jackson Piper
jdp134@uakron.edu

Kyle Law
kpl15@uakron.edu

Anthony Meniru
aum3@uakron.edu

Safe Kids Car:
Safety Systems and Functional Improvements for Ride-On Toy Cars

Senior Design Project Final Report

Design Team 12

Nathan Keenan

Kyle Law

Anthony Meniru

Jackson Piper

Faculty Advisor: Syed Imam Hasan

24 April 2023

Table of Contents

List of Figures

List of Tables

Abstract

Children's ride-on toy vehicles frequently lack critical safety systems and functionality commonly present among modern, full-sized vehicles. One may observe that numerous defining characteristics of modern vehicle technology, namely collision mitigation, end-user variable speed control, and environmental feedback control, are absent from children's ride-on toy vehicles. Alternatively, a scaled integration of modern vehicle technologies retrofitted among ride-on toy vehicles presents a viable solution. Foremost, a project foundation is built upon the fundamental understanding of applicable preexisting technologies. Thus, engineering analysis is thoroughly conducted, particularly among the study of embedded systems, power electronics, sensors, and electromechanical control systems. Subsequently, engineering requirements specifications were developed as necessitated by the quantifiable scope of the project. Ultimately, an accepted technical design is realized to implement the proposed, tailored solution. The addition of safety systems and functional improvements for ride-on toy vehicles will positively impact the user experience for the children who enjoy them and the peace of mind of the guardians who care for them. [JP]

Key features include:

- Collision mitigation system utilizing the latest Ultrasonic Time-of-Flight sensor technology
- Variable speed control analogous to full-sized motor vehicles
- Automatic headlights
- Robust power management system requiring only one rechargeable battery
- Scalable design that is retrofittable to a multitude of ride-on toy vehicles

1. Problem Statement

1.1 Need

There is a need for a way to improve the safety of toy vehicles, as most of these toy vehicles have no safety systems, even though such systems exist in full-sized consumer motor vehicles. Consequently, accidents involving toy vehicles, including collisions with environmental hazards and people, can damage the toy vehicle and cause injury to the young driver. For example, riding toys are responsible for 42% of toy-related injuries [1]. Furthermore, in 2017, 7 of the 13 recorded toy-related child deaths were due to ride-on toys [2]. Clearly, a safety device or system that can help prevent these types of accidents would help to keep toy vehicles functional for longer and prevent injury and death for riders of these toys.

1.2 Objective

To reduce the number of accidents involving toy vehicles, a safety system should be implemented. Such a system would include, but is not limited to, automatic braking, collision detection and lighting (headlights/taillights). The first two parts of the system should be able to stop the toy vehicle when it detects an oncoming hazard such as environmental hazards, people, and cars with the help of an embedded system. The addition of headlights and taillights would also allow for safer use of the toy vehicle in poorer lighting conditions, including in the early morning, evening, and cloudy weather, as they would alert drivers of other vehicles to their presence on the road.

1.3 Background

Electric ride-on toy cars for children lack certain types of advanced safety equipment that would help protect the vehicle and keep the children who ride them safe from injury. Safety features such as headlights, collision avoidance systems, and automatic braking systems are generally missing from most of these ride-on toy vehicle designs. In 2016, a report was released detailing that ride-on toys are responsible for 42% of toy-related injuries (Spivey Law Firm, 2016), and all the ride-on vehicles in the report had no collision avoidance system or active braking features. Adding these safety features to existing ride-on toys would help prevent injury to the children who ride these toys and to other children present around these toy cars.

Collision detection would be the most beneficial addition to the vehicle as it will alert the driver to approaching obstacles. This detection system will be able to detect oncoming traffic (i.e., a car driving down the street towards the vehicle) to avoid any collisions with a moving obstacle. An alarm would then be triggered, either visually or audibly for both the driver and anyone nearby who might be supervising the user. The detection system would then automatically trigger any active brakes on the vehicle to avoid collisions into walls, trees, and other various obstacles. The active braking system will be used to halt all forward motion of the car, essentially bringing the vehicle to a complete stop. Once the object is at a safe enough distance from the vehicle, or the user reorients the vehicle to a position suitable for forward driving, the braking system would allow forward motion once more.

Adding functional headlights to the vehicles would allow them to operate more effectively at times and conditions of low visibility. Along with being easier to drive at night, drivers or pedestrians approaching the vehicle in darker conditions will be better able to spot the toy vehicle and avoid collisions with it. All the added systems would work in tandem with each other to create a far safer product for the youths that will be driving them. [KL]

1.3.1 Existing Solutions

Currently, collision avoidance in full-size vehicles is done in a variety of ways; here, two specific methods will be covered. Firstly, Collision Mitigation by Braking (CMBB) and secondly, a variety of driver warning systems. In a CMBB system, the collision avoidance software triggers an automatic application of the brakes on the vehicle when a specific set of circumstances is observed by the system. Using various sensors such as a "camera with image processing algorithms computing bearing and elevation" and "IR radar measuring bearing elevation, range and range rate" (Jansson, Johansson, & Gustafsson, 2002, p. 198), the system can compute when to apply automatic braking. In the case of a driver warning system, there must be a level of warning frequency that does not either desensitize the driver to the warnings or cause the driver undue stress due to their rarity (Seller, Song and Hedrick, 1998, p. 1334).

The algorithms used in a CMBB system vary by automobile manufacturer, but they all utilize some combination of "relative distance, relative velocity and vehicle velocity" (Seller, Song and Hedrick, 1998, p. 1335) in addition to a critical braking distance mathematical definition in order to allow the system to determine when to apply emergency braking. Similarly, driver warning systems vary by manufacturer, with some using auditory signals in combination with visible lights and others using a mixture of the previous along with some form of vibrotactile system (Seller, Song, Hedrick, 1998, p. 1334) (Meng, et. al., 2015, p. 329).

There are some limitations present in the current designs of collision detection and automatic braking systems and in the technology used in these systems of which it is necessary to be aware. One of these limitations comes from the types of sensors used in current collision detection systems and affects the range at which the chosen optical sensors for the collision warning and detection system can operate. For example, millimeter radar and infrared radar, two common types of sensors used in current forward collision avoidance systems, have a narrow

field of view, causing their individual operation to become unreliable at close distances (Jansson, Johansson, & Gustafsson, 2002, p. 204). The impact of this problem is lessened for full-size vehicles, as the speeds at which they operate necessitate the detection of collisions from farther away, but for ride-on toy vehicles operating at comparatively low speeds, most of the collision detection will need to occur at close range. The usage of multiple types of sensors in current systems, often considered necessary because of the limitations of each individual type of sensor, also leads to measurement discrepancies when the sensors cannot be synchronized with each other, which affects the collision detection model and causes false-positive collision detections (Jansson, Johansson, & Gustafsson, 2002, p. 204). [SA]

The threshold at which the system will warn the user of a collision is also subject to a few design limitations. In current systems, this parameter is carefully considered, as the warnings must be presented to the user in such a way as not to desensitize the user to the warnings nor to startle the user, and potentially the watchful guardians in this case, when the warning does sound (Seller, Song and Hedrick, 1998, p. 1334). A similar safety limitation is placed upon current automatic braking systems. Current automatic braking systems are designed to avoid collisions only when the interference from the system is guaranteed not to cause further harm or when the probability of a collision is calculated to be 1, meaning that a collision is certain (Jansson, Johansson, & Gustafsson, 2002, p. 197). Interference from the automatic braking system can also cause unpredictable and harmful responses from the human driver, such as if the driver is startled by such active interference, further emphasizing the need for current systems to use conservative critical warning distance and critical braking distance algorithms (Seller, Song and Hedrick, 1998, p. 1334). Finally, a limitation for headlights, similarly to those used in electric vehicles, is simply that the battery must be able to power them as well as the motor. [NK]

1.3.2 Comparison Among Proposed Solution and Existing Technologies

There are several similarities and differences among the braking system and battery implementation of ride-on toy vehicles and full-sized front-wheel drive electric vehicles. The braking system works similarly to cooperative braking systems for front-wheel drive electric vehicles. The braking system in the riding toy cars will be regulated by a Motor Control Unit (MCU) similar to the MCUs found within modern electric front-wheel drive vehicles. However, the MCU in a front-wheel drive vehicle adjusts the braking force based on the amount of pressure exerted by the driver through a pressure control system (Zhao 2018). On the other hand, the motor control system adjusts the braking based on the environment surroundings such as incoming vehicles, obstacles, or major inclining roads. The braking system found within a full-sized front-wheel drive electric vehicle is also powered using a rechargeable battery solution, which is to be used in the ride-on toy cars (Zhao 2018).

This battery will also be used to power the LED headlights and taillights for improved visibility. Lithium-ion batteries function similarly to Iron Dicyano Dichloro batteries. Both batteries convert chemical energy into electrical energy through electrochemical process (Nhapulo 2021 & Ramavath 2018). Additionally, both batteries are used for electric applications including machines, electric cars, and light emitting diodes (LED). However, one difference between the lithium and iron diycano dichloro batteries is that lithium is a cathode battery. Iron diycano dichloro batteries operate as both cathode and anode by combining Fe(anode) with DDQ in methansulfonic acid, which is used as the anode. In addition, Fe-DDQ batteries are considered cost effective for powering systems for single use. On the other hand, lithium batteries can operate at less than 3 V, for small applications. [AM]

1.3.3 Discussion of Previously Patented Technologies

The successful design and application of a collision avoidance system implores the usage of existing and patented technologies. Previous automotive technologies focus upon passive

safety systems, which are designed to mitigate the effects of a collision. The safety belt, the air bag, and the safety cage are examples of passive safety systems. In contrast, more modern automotive technologies employ active safety systems, which are designed to reduce the likeliness of a collision. Active safety systems may include various implementations of collision avoidance, collision detection, automatic braking, driver assistance, and adaptive cruise control. Active safety systems, namely collision avoidance, are prominent throughout the modern automotive industry. Currently, an overwhelming majority of major automotive manufacturers, including Chevrolet, Ford, Jeep, and Mercedes-Benz, offer some form of collision avoidance technology within their product lineups (Sinclair, 2021). Volvo Cars, renowned for its introduction of the three-point safety belt in 1959, is regarded as being first to bring a patented collision avoidance system to a substantial portion of the consumer automotive market (Volvo Car Corp., 2022). Andreas Eidehall and Jochen Pohl of Volvo Cars patented their *Method and system for collision avoidance* in 2007, later revised in 2012. This collision avoidance system, principally designed to reduce the risk of unsafe maneuvers while one changes lanes, is capable of estimating future trajectories of detected external objects while computing the trajectory of the vehicle. The system compares these trajectories and actively intercepts the driver's inputs, namely steering and throttle, for appropriate inputs selected by the system to mitigate the risk of a potential collision (Eidehall & Pohl, 2012). The astonishing capabilities of this implementation of a collision avoidance system are built upon previous and notable patented technologies. In 1988, William L. Kelley introduced an early adaptation of a collision avoidance system with his *Collision predicting and avoidance device for moving vehicles*. Kelley's implementation is considered to be the first of its kind, as it is the earliest documented United States patent for such a system. The operation of Kelley's collision avoidance system is briefly described below:

> The apparatus includes at least one microwave pulsed transmitter and receiver for transmitting a scanning beam of pulsed energy which scans a sector of space, at least forward of the vehicle, a check for producing timing pulses, a ranging device connected to the clock and the receiver for measuring the time difference between the transmitted pulses and any echoes received by the receiver. The antenna is pivotally coupled to the vehicle and a scanning motor serves to set the antenna in a scanning motion. A direction device is coupled to the scanning antenna for sensing the direction of the antenna. A computer is connected to ranging device, the clock, the direction device, and computes continuously the last three coordinated for vector to the object and is connected to an annunciator which can speak and/or display a message to the vehicle operator. (Kelley, 1988)

The fundamental operation of Kelley's collision avoidance system serves to be paramount in the development of subsequent collision avoidance systems. One notable instance is Scott Juds's *Collision avoidance system for vehicles*, patented in 1995. Juds's implementation, while similar in fundamental operation, features a more robust approach compared to Kelley's implementation. Dissimilar from Kelley's microwave transmitter and scanning antenna methodology, Juds utilizes an array of infrared light emitting diodes (LEDs) oriented at various directions in tandem with a fixed detection module outfitted with photosensitive detectors (Juds, 1995). Thus, Juds's collision avoidance system ultimately provides uninterrupted monitoring of a wider area of incidence adjacent to the vehicle.

It is feasible to utilize existing and patented instances of collision avoidance systems to design and execute a collision avoidance system for ride-on toy cars. Ride-on toy cars aim to mimic the function and aesthetic of their "full-sized," equivalent vehicle models. The fundamental operation of previous instances of collision avoidance systems may be scaled down

from full-sized vehicles to be suitable for use with ride-on toy cars. In 2004, Peter Reile and Brian L. Bienz introduced their *Children's ride-on vehicle with electronic speed control*, popularized by Fisher Price® Power Wheels®. Their design, while including rudimentary elements of both passive and active safety systems, does not feature any instance of a collision avoidance system (Reile & Bienz, 2004). However, their design does promulgate that one may interface with the electronic motor control to introduce active braking to the ride-on toy car. Reile and Bienz list examples of methods for introducing active braking to the ride-on toy car:

> An illustrative example of a mechanism for actively braking the vehicle is to temporarily short the motor assembly. Another example is for the drive assembly to be configured to automatically brake the vehicle when the motor assembly is not energized. Yet another example is to brake the vehicle by Controller 70 by sending control signals to produce a relatively short pulse of rotational input from the motor assembly in the opposite direction than the vehicle is traveling. (Reile & Bienz, 2004)

The integration of an external object detection system with the preexisting active braking capabilities of the ride-on toy car effectively creates a scalable implementation of a collision avoidance system. Much like its full-sized, analogous system, a successfully executed collision avoidance system for ride-on toy cars will invaluably mitigate the risk of injury, especially for the children who will enjoy them. [JP]

1.4 Marketing Requirements

1. The system should be built to work with existing, repurposed ride-on toy cars.
2. The system should be lightweight and compact.
3. The system should provide collision warnings and collision detection.
4. The system should provide automatic braking for the car to avoid collisions.

5. The system should include headlights and taillights that are controlled automatically.
6. The system should provide an input for variable speed control rather than the existing three-option motor control system.

2. Engineering Analysis

2.1 Circuits

The goal of the power subsystem is to distribute the input voltage (12V DC) to the various components of the ride-on toy vehicle, including motor control, embedded systems (microcontroller), braking, and lighting systems. These components require 5V to work, which is lower than the nominal input voltage from the battery. A Buck Converter is a type of DC-DC Converter that steps the input voltage down to a smaller output.

A buck converter is a switching circuit that consists of an input voltage source, a switch, a diode, an inductor, a capacitor, and a resistor (as shown below). The output voltage V_O, which can be measured from the far end next to the resistor is related to the input voltage V_S.

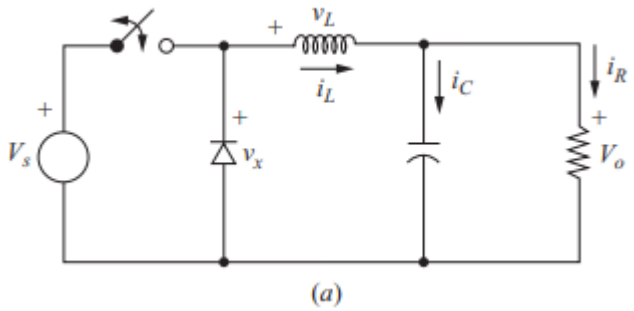

Figure 1: Basic Buck Converter (from Power Electronics by Daniel W. Hart)

The relationship between the input voltage and output voltage can be obtained by analyzing the circuit in two conditions: when the switch is turned on and when the switch is turned off. When the switch is turned on, the current from the input voltage flows through the circuit, including the cathode of the diode. As a result, the diode is opened due to the reverse biased state. The DC-DC converter in this condition is shown in the figure below. This occurs between the times of $0 \leq t \leq DT$.

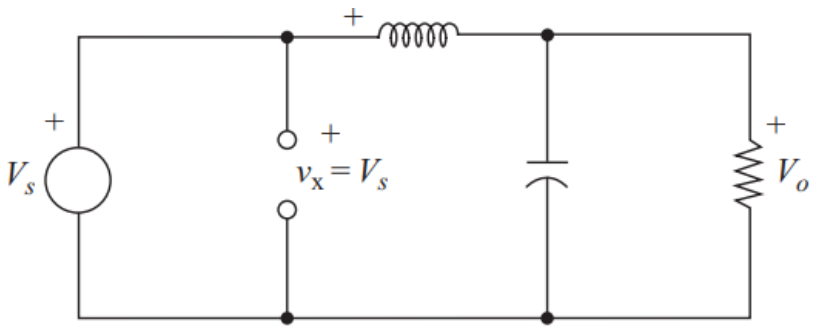

Figure 2: Basic Buck Converter with switch on

(from Power Electronics by Daniel W. Hart)

Kirchhoff's Voltage Law is used to derive the inductor voltage in terms of the input and output voltages. The inductor voltage is used to connect the relationship between the input and output voltages under the two switching conditions.

$$V_{L,closed} = V_S - V_O \quad (1)$$

The second switching condition is when the switch is turned off, which opens the loop at the input voltage. This condition occurs at time t interval [DT, T]. Under this condition, the current flows through the anode of the diode, which ideally shorts the diode and closes the rest of the circuit (as shown below).

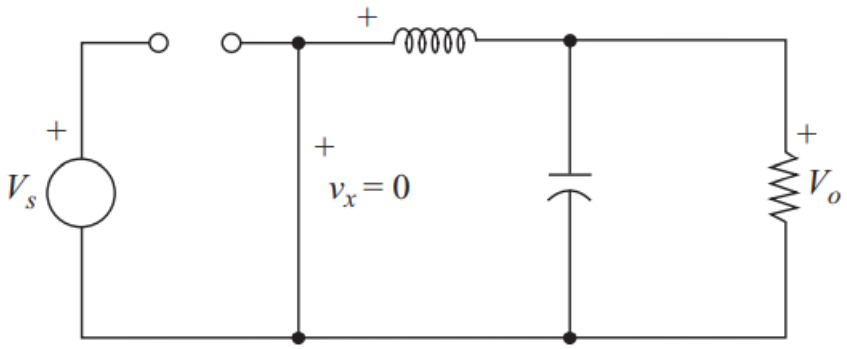

Figure 3: Basic Buck Converter with switch off

(from Power Electronics by Daniel W. Hart)

Similar to the first switching condition, Kirchhoff's Voltage Law is used to obtain a relationship between the inductor voltage and output voltage. Since the source is open, the input voltage is not included in the second KVL equation.

$$V_{L,open} = -V_O \quad (2)$$

The KVL equations under both conditions are combined by setting the average inductor voltage to an integral equation using Equations 1 and 2. This integral equation is set equal to zero because the average inductor voltage is zero. After substituting the terms V_S and V_O using the equations obtained in KVL analysis, the integral equation below is simplified in terms of V_S, V_O,

and D. The simplified equation shown below (Equation 3) is the duty ratio between output voltage and the input voltage. Refer to the appendix for the complete derivation of the duty ratio.

$$V_{L,avg} = \frac{1}{T}\left(\int_{0}^{DT} V_{L,closed}\, dt + \int_{DT}^{T} V_{L,open}\, dt\right) \tag{3}$$

$$D = \frac{V_O}{V_S} \tag{4}$$

The duty ratio depends on the provided input voltage and the desired output voltage. A buck converter is designed to reduce output voltage by setting the duty ratio to obtain an output voltage that is lower than the input. Selecting the correct Buck Converter will depend on the desired output voltage needed to power the components requiring 5V DC from the 12V DC battery. [AM]

2.2 Electronics

Another component that will be used to design the ride-on toy vehicle power system is a noninverting operational amplifier. A noninverting operational amplifier will be used to maintain an output voltage of 5V for components that require more voltage to perform correctly. A simple noninverting operational amplifier configuration consists of a power source that is connected to the positive end of the op amp and two resistors connected as a voltage divider to the negative end of the op amp (shown in figure below). For simplicity, the voltages and currents flowing from the input side of the op amp will be denoted v_+, v_-, i_+ and i_-. The noninverting op amp shown below can be analyzed by assuming the op amp is an ideal op amp. Assuming that the op amp is ideal will simplify the circuit analysis because the currents i_+ and i_- are both zero because of an infinite resistance at the input of the op amp. As a result, the current flowing from the op amp is neglected in the analysis.

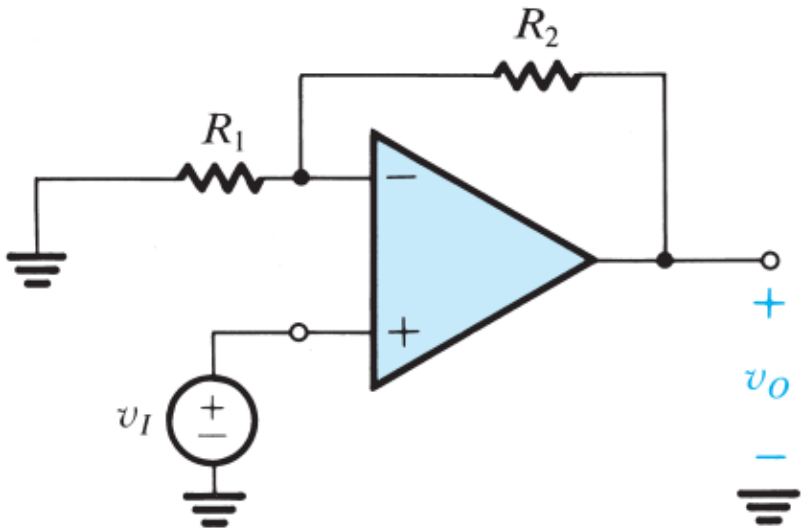

Figure 4: Noninverting Operational Amplifier Configuration

(from Microelectronics Circuits by Adel Sedra and Kenneth Smith)

Using Kirchoff's Voltage Law, the input voltage is the only voltage connected to the positive end of the op amp. As a result, the positive voltage v_+ is equal to the input voltage. Using Kirchoff's Current Law at the node near the negative end of the op amp, an equation using the two resistances, output voltage and the negative voltage v_- (as shown below).

$$\frac{v_-}{R_1} + \frac{v_O - v_-}{R_2} = 0$$

Using the assumption that the op amp is ideal, v_- can be substituted for v_I because v_+ and v_- are the same. Substituting v_I for v_- and performing some algebra, the input and output can be combined into a fraction that is called the gain. The gain can be set equal to sum of 1 and the ratio between R_1 and R_2. The following equation for the gain is written below. The equation below will be used to determine ideally what values the resistances need to obtain an output voltage of around 5V. [AM]

$$\frac{v_O}{v_I} = 1 + \frac{R_2}{R_1}$$

2.3 Embedded Systems

The microcontroller will be the component responsible for much of the calculation and decision-making for the Safe Kids Car. The microcontroller will gather input from the sensors, process this input to determine the likelihood of a collision, and output the appropriate motor control signals to the rest of the system. The microcontroller must have a low current draw and sufficient clock speed in order to maintain a battery life of 3 hours and be able to react to obstructions within the given period of 100ms, in accordance with Engineering Requirements 2 and 5. The microcontroller must be able to accommodate several sensors, be able to output pulse-width modulated signals to drive the DC motors, and be able to read sensor data with a high enough resolution so that no information is lost. The microcontroller should also have a supported C compiler. [NK]

2.4 Sensors and Controls

Throughout all sectors of modern consumer technology, the remarkable ability to quantify external environment conditions and subsequently respond to such inputs autonomously is afforded by the advancement of sensor technology. Notably, this concept continues to be best exemplified by the prominence of active safety systems throughout the modern automotive industry. Recall that previous automotive technologies focus upon passive safety systems, which are designed to mitigate the effects of a collision with technologies such as the safety belt, the air bag, and the safety cage. In contrast, active safety systems are designed to utilize quantifiable environmental inputs to take evasive actions to mitigate the likelihood of a collision through technologies such as collision avoidance, automatic braking, automated headlights, driver assistance, and adaptive cruise control (see **1.3.4 Discussion of Previously Patented Technologies**). Comparatively, such active safety system technologies all require the careful implementation of sensor technology within a closed-loop feedback system to be rendered

effective. When implemented correctly, modern sensor technology will allow for successful execution of active safety systems for the children's ride-on toy car much like its full-sized, analogous counterpart.

2.4.1 Motivating Examples: Distance

Prior to the realization of a functional active safety system, namely an active safety system of collision mitigation, one must first consider the most critical environmental inputs necessary for successful system closed-loop feedback. As previously elaborated upon, numerous collision detection and avoidance schemes have been proposed, patented, and implemented. One may observe that these various collision detection and avoidance schemes, such as those conceived by Kelley (1988) and Juds (1995) exhibit vastly unique practices for obtaining environmental inputs. Consequently, one crucial commonality among these examples of collision mitigation is the procurement of a measure of distance.

The distance, or proximity of an obstacle external of the vehicle, serves as the ultimate quantifiable environmental input to realizing collision detection and avoidance. Timing may be synchronized among the components and operations of a collision mitigation scheme quite easily, but the accurate and precise measurement of distance between a vehicle and potential obstacles truly enables such a system's ability to interact with its surroundings. The collision detection and avoidance algorithm developed within the scope of this project relies heavily upon accurate and precise measurements of distance (refer to **5.2.3 Level 2 Software Behavior Models**). This motivates the thorough engineering analysis of current proximity sensor technology.

2.4.2 Proximity Sensors and the Time-of-Flight Sensor: Theory of Operation

The Time-of-Flight (ToF) sensor scheme enables the detection of range, or proximity, among two bodies through the transmission of modulated pulses of energy, typically light waves, or sound waves. Emitted waves of known periodicity reflect upon the surface of any objects that traverse the specified range of proximity of the ToF sensor. A receiver records the reflections of these emitted waves independent of amplitude. Figure 5 provides a rudimentary illustration of the generic ToF proximity sensor system.

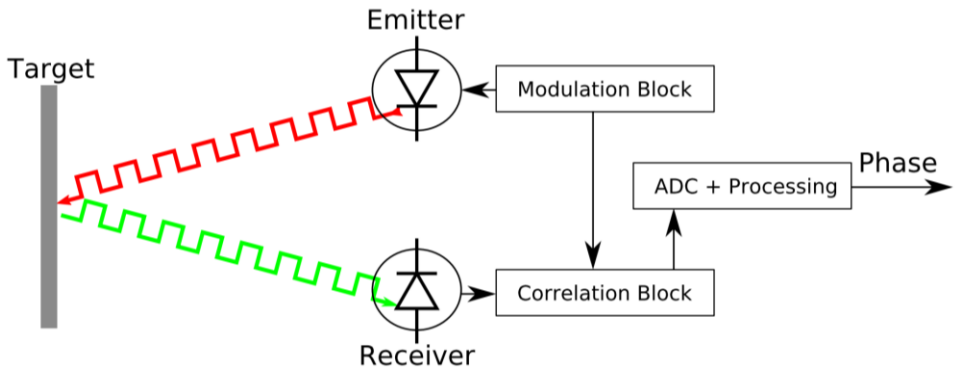

Figure 5: Generic ToF Proximity Sensor System, adapted from *Introduction to Time-of-Flight Long Range Proximity and Distance Sensor System Design* by Texas Instruments

The elapsed time recorded among the emitted and received modulated wave pulses serves to determine the proximity of an object within range of the ToF sensor. Because the periodicity of the incidental waves is known and fixed, the determination of elapsed time and thus absolute distance is derived from the phase modulation observed among the emitted and received waves. Figure 6 demonstrates this theory of operation through the basic illustration of phase modulation of a 10 MHz (100 ns) signal.

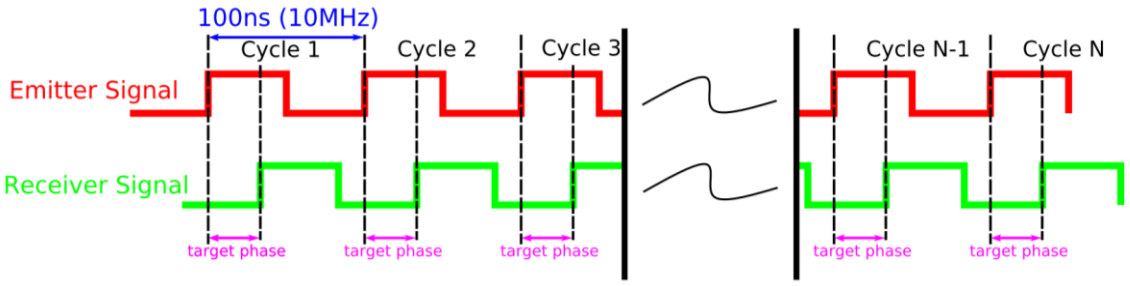

Figure 6: Generic ToF Sensor Timing Diagram, adapted from *Introduction to Time-of-Flight Long Range Proximity and Distance Sensor System Design* by Texas Instruments

One must note that the phase determination is aggregated over several cycles of the periodic light modulation (Texas Instruments).

2.4.3 Available Time of Flight Sensor Implementations

Prior to the final iteration of proximity sensor design, one must consider the appropriate selection of ToF implementation based upon a multitude of factors relevant to the design application. Factors may include, but are not limited to detection range, detection rate, power, and field-of-view (FoV). Engineering analysis reveals the benefits and detriments of current available ToF sensor implementations.

LiDAR (Light Detection & Ranging) ToF sensors comprise many popular variations of current optical proximity sensor technology.

- Pulse-modulated, direct Single Photon Avalanche Diode (SPAD) solutions operate at relatively short distances, namely less than 1 meter (3.28 feet) but boast extreme accuracy at this range, as well as miniscule form factors. One must account for their possible severe performance issues at high ambient and outdoor temperatures, in addition to compromised performance with FoV greater than 30°.
- Discrete time-based direct Avalanche Photodiode (APD) ToF sensors do not compromise accuracy or precision when determining distances on the order of many meters. In consequence, their cost of implementation is comparatively high due to

their need for precision calibration and manufacturing. Some packages require specialized high-voltage biasing.

- Continuous wave-based indirect ToF sensors feature available solutions with medium range capabilities of approximately 2 meters to 15 meters (6.56 feet to 49.21 feet), comparatively high sample rates (framerate), and optimized high ambient light performance to mitigate aliasing. While current product offerings are affordable, this implementation cannot perform within sub-millimeter tolerances.

Alternatively, ultrasonic ToF implementations have seen increased popularity with the introduction of numerous affordable product offerings. Ultrasonic ToF sensors employ a similar theory of operation when compared to LiDAR ToF sensors, but dissimilarly make use of the emittance, reflection, and reception of ultrasonic sound waves with frequencies well beyond the perception of human hearing frequency response (20 Hz to 20 kHz). The typical pulsed echo ultrasonic ToF offerings remain insensitive to any ambient light conditions that plague LiDAR ToF sensors. These solutions are excellent for inclement environmental conditions, such as rain or fog. Effective operation of ultrasonic ToF sensors may be compromised greatly by specular reflection of sound waves upon rigid surfaces, absorption of sound waves upon porous surfaces, and changes in sound wave medium, including humidity and air pressure. Moreover, sample rates are comparatively reduced from those of LiDAR ToF solutions, limited greatly by the speed of sound (Texas Instruments).

Engineering analysis of current available ToF sensor implementations reveals that either optical continuous wave-based indirect ToF sensors or ultrasonic ToF sensors are viable solutions for achieving the engineering requirements within the scope of the proposed design project. Primary considerations include cost, FoV, and depth of range capabilities. Future

simulation and experimental procedures within actual environmental scenarios will aid in determining viability among these proposed proximity sensor implementations.

2.4.4 Ambient Light Sensor Implementation

Ambient light sensor implementations are innumerable within all sectors of modern technology. The prevalence of ambient light sensors is attributed to their robustness and their steadfast technological development for increasing quantifiable perception of environmental inputs. In contrast to the theory of operation of the proximity sensor, the ambient light sensor records the amplitude (intensity) and/or frequency (color) of incidental light waves. Typical ambient light sensor implementations feature a photoelectric detector unit that converts incidental light rays into voltage signals via the photoelectric effect. Ambient light sensors do not require rapid sampling rates for effective implementation, nor do most applications require extreme accuracy or precision, except for niche purposes.

The scope of the proposed modifications to the children's ride-on toy car warrants the use of ambient light sensor implementations for one primary application. That is, similarly to modern, full-scale motor vehicles, ambient light sensor readings shall be used to effectuate the automatic control of the ride-on toy vehicle proposed lighting system.

2.4.5 Hall Effect Sensor Implementation

A Hall effect sensor implementation shall be used to obtain the velocity of the ride-on toy vehicle while in nominal operation by a generic radial velocity sensor (refer to Figure 7 for depicted simplified theory of operation). While numerous Hall effect sensor implementations make use of detecting the modulation of a visible or infrared light source via an ambient light sensor (such as depicted by Figure 7), a permanent magnet and magnetic field intensity sensor provides a more robust and permanent solution.

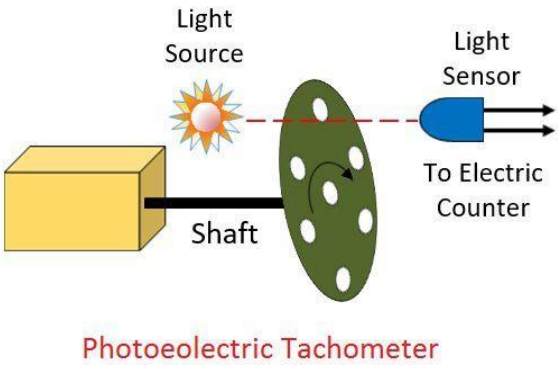

Figure 7: Generic Radial Velocity Sensor with Ambient Light Sensor Implementation

Project applications of this radial velocity sensor implementation include experimental determinations of vehicle dynamics during nominal operation to better optimize various subsystems, as well as an avenue for the collision detection and avoidance routine to mitigate effects of uncontrollable downhill acceleration. It is advisable to make use of a permanent magnet and magnetic field intensity sensor implementation to realize the radial velocity sensor due to its robustness and reluctance to environmental interference. [JP]

2.5 Electromechanics

The electromechanical system will utilize the existing motors and battery to provide forward and backward motion to the vehicle. However, the operation of the existing motors will be modified with a selected Motor Control Unit (MCU). The MCU shall behave in accordance with instructions relayed by the collision avoidance embedded system and sensor subsystem. This section will provide insight into the existing operation and potential design constraints presented by the existing motor system of the preexisting ride-on toy car. Engineering analysis of the existing motor system will be utilized to realize the methodology of the proposed motor control.

2.5.1 DC Motor Theory

The existing DC motors from the Fisher Price® Power Wheels® ride-on toy car are brush DC motors, using a carbon brush to transmit electrical current to create the motor's

rotation. Two of these used synchronously produce the vehicle's forward and backwards motion and can be manipulated via a motor controller unit (MCU).

2.5.2 Existing Motor System Characteristics and Design Constraints

Upon disassembly of the Fisher Price® Power Wheels® ride-on toy car, it was discovered that there are two, identical brushed DC motors both mounted on the rear axle of the vehicle. These motors work synchronously to provide forward or reverse locomotion; however, they do not articulate or control steering. Insights during disassembly did not reveal any specific part number or printed characteristics for the pair of DC motors included. Thus, electromechanical engineering analysis techniques, including experimental procedures, were performed upon the DC motors as a pair.

Per existing functionality of the ride-on toy car, these DC motors require the full 12V voltage supplied by the provided lead-acid battery to operate within steady-state conditions. Ideally, the DC motors will be provided the full nominal voltage of the 12V battery. It is noteworthy that for this design iteration, it is not currently feasible or practical to obtain characteristics of the pair of DC motors while operating under rated load, signified by the inclusion of vehicle friction and the weight of the vehicle and its end-user. First, a simple, single no-load test is performed upon the pair of DC motors at stock operating conditions. Whilst operating under the forward regime, steady state, unloaded conditions with a nominal battery voltage of 12V, it was determined that the pair of motors draws a sustained current of roughly 2.28A with a power draw of about 27.36W (See *Equation 5*).

$$P = IV \tag{5}$$

$$\text{nominal power} = \text{nominal current} \times \text{nominal voltage}$$

$$27.36\text{W} = 2.28\text{A} \times 12\text{V}$$

Subsequently, the no-load test is replicated among an array of fixed DC voltages to characterize the pair of DC motors based upon the resulting total operating current. Table 1 and Figure 8 reveal the results of this experimental procedure.

Table 1: Results of No-Load Test Measuring Current versus Voltage of DC Motors

Voltage (V)	Current (A)
0.5	0.41
1	1.15
1.5	1.22
2	1.30
2.5	1.38
3	1.43
3.5	1.50
4	1.56
4.5	1.63
5	1.71
5.5	1.72
6	1.82
6.5	1.84
7	1.89
7.5	1.92
8	1.95
8.5	1.99
9	2.03
9.5	2.05
10	2.10
10.5	2.14
11	2.16
11.5	2.22
12	2.28
12.5	2.31
13	2.38
13.5	2.39
14	2.41

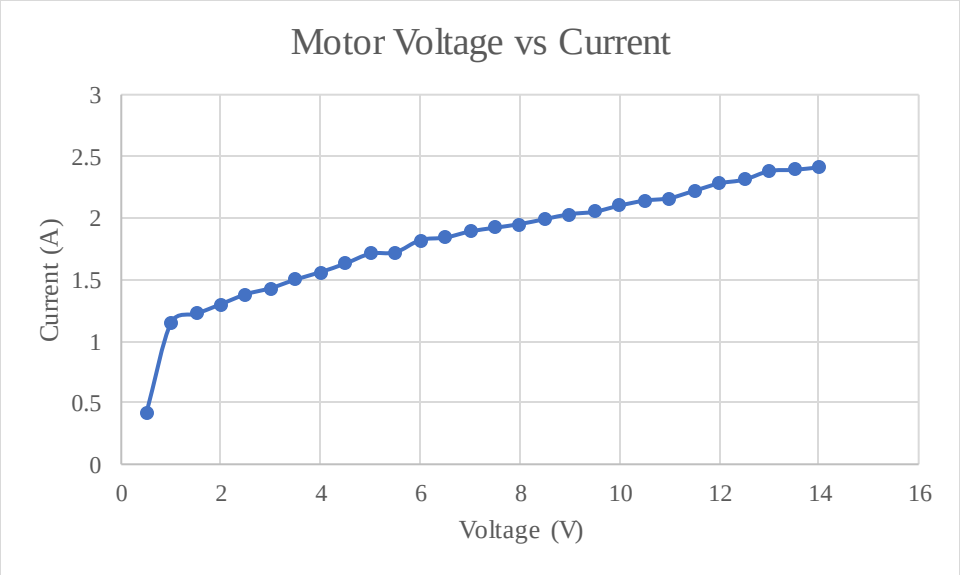

Figure 8: Graph of Current versus Voltage of DC Motors

From the results of the experimental procedure, one may interpret that the DC motors have an expected hysteresis relationship that appears linear in many areas. Whilst the current does gradually increase with voltage, towards the rated voltage of 12V and beyond, there is a clear ceiling that the motors approach for current, which according to the data can be assumed to be 2.5A. For the purposes of this project, this system will not be operating at any voltage higher than 14V (as the battery voltage seems to go above 12V depending on the level of charge).

2.5.3 Interfacing with Existing Motor System

Whilst this project will employ the existing pair of DC motors within the ride-on toy car, the motors will interface with a procured Motor Control Unit (MCU) that will be designed to precisely control the locomotion of the vehicle, namely acceleration and deceleration. To align with common modern DC motor control practices, pulse-width modulation (PWM) is proposed as the primary method of variable speed control (VSC) for the ride-on toy car. A defined PWM duty cycle will correspond with desired acceleration and deceleration to achieve VSC. Detailed methodology will be developed through simulation and experimental procedure upon the subsequent project design iteration once vehicle dynamics are established. Vehicle dynamics of

the children's ride-on toy car will be realized as a scaled and simplified model of vehicle dynamics for a full-scale electric vehicle.

Secondarily, regenerative braking will be realized for the ride-on toy car. Simply stated, the pair of DC motors is to be subjected to synchronous, gradual pulses in the reverse polarity scheme to realize a decelerative braking effect of the vehicle. The end-user of the ride-on toy car shall be able to apply braking to the vehicle manually with his or her input, for the preexisting ride-on toy car does not employ a direct method for the end-user to brake the vehicle. Alternatively, the automatic collision detection and avoidance subsystems shall be able to interface with the braking scheme of the ride-on toy car as necessary. It is notable that the tires of the existing ride-on toy car are composed of plastic; thus, the tires will not exhibit high contact friction with whichever type of surface they traverse. It is accepted that full-scale vehicles with standard rubber tires traversing a dry, paved road exhibit a coefficient of friction of about 0.7. Conversely, plastic has a coefficient of friction of only about 0.4 for most surfaces. However, when one considers that the top-rated velocity of nominal operation for the ride-on toy car is about 5 mph, this reduction in friction should not pose a detriment to the proposed braking system. As stated before, detailed methodology will be developed through simulation and experimental procedure upon the subsequent project design iteration once vehicle dynamics are established. [KL, JP]

3. Engineering Requirements Specification

The engineering requirements outlined in Table 2 address the technical needs of this design. Including engineering requirements, the table will also describe the reasoning behind each requirement (why it is necessary). These correlate with the Marketing Requirements from the list for this report (see **1.4 Marketing Requirements**). These marketing requirements were used to derive the following requirements.

Table 2: Engineering Requirements

Marketing Requirements	Engineering Requirement	Justification
1, 2	1. System must operate from the existing 12V, 9.5Ah Lead Acid Battery.	Allows the implemented system to use an already existing power supply, reduces complexity.
1	2. System must maintain a battery life of 1-3hrs before recharging.	Typical battery life of ride-on toy vehicles. Maintain a decent drive time even with the added systems.
1, 2	3. Top sustained speed of the vehicle should be 5mph (~7.3ft/s)	Existing top speed, added weight of modifications is negligible.
3	4. System should be able to detect stationary and moving obstacles within its path at a minimum range of 15ft.	The larger range that can be achieved for detection, the more time allowed for the system to respond to oncoming obstacles.
3	5. System should be able to respond to an oncoming obstacle within 100ms of detection.	Affords sufficient time for child driver to react before autonomous collision mitigation.
3	6. An audible warning should be issued to the driver (or a nearby guardian) of an oncoming obstacle (be it stationary or moving).	Will assist the driver in avoiding obstacles, keeping them alert of oncoming danger. If the driver can avoid an obstacle, the autonomous collision mitigation will not need to be activated.

3, 4	7. System should include active braking (both manual and automatic) to avoid collisions and allow the user to stop more effectively. The collision detection system should be able to implement automatic braking.	Existing system has no active braking, only cuts power to the motor when accelerator is not engaged. Introducing active braking will allow the vehicle to stop more effectively.
3, 5	8. System must incorporate headlights/taillights that are automatically activated in low-light situations but can also be manually trigger by the driver (but not turned off in the case of low light situations).	Assist the driver in low visibility situations and allow the vehicle to be more effectively used during dusk or nighttime. Indicates presence of the child and vehicle in low visibility situations.
6	9. A variable user input shall determine vehicle speed in a manner analogous to the accelerator pedal of a full-sized motor vehicle. The rotational speed of the vehicle motors shall vary between approximately 0 and 200 RPM.	Allows the child driver to go at a speed other than previously rated maximum or reverse speeds. The variable input has the potential to reduce power consumption for different speeds (instead of maximum or no power).
4,6	10. The system must be able to provide both forward and reverse analog speeds to the motors. The motor control unit will adjust these speeds according to the user and sensor inputs.	Adjustable outputs are mandatory for user control of the toy vehicle. Allows the toy vehicle to respond to various environments.

4. Engineering Standards Specifications

4.1 Safety

This project requires work in the laboratory involving voltages no higher than 24V for testing purposes. The highest rated voltage of the system will be 12V DC, however proper safety standards must be maintained in the laboratory. Wires and circuits must be kept well maintained and not exposed to avoid accidental contact with electrified circuits. Even at a low voltage, there is still a risk of damage to those operating in the laboratory.

For the actual system, all electronics and hardware must be positioned and maintained in such a way as to avoid any electrical contact with the operator of the vehicle. Ideally, any electronics should be contained in a solid compartment where the driver (youth) cannot access any electrical hardware without the assistance of an adult guardian. These various components should also be securely fastened to the vehicle, to limit damage from any impacts that the vehicle might suffer. With this, the system must also be reliable and simple to repair in case of a breakdown. The various subsystems and components must have fail-safes and backups in case of malfunction. In the event of a malfunction, the vehicle should still be able to operate, even if operational capacity is limited. Regardless, designs must be implemented to minimize the possibility of any system failure.

4.2 Data Formats

For the purposes of this project, a 32-bit microcontroller will be implemented into the design. Whilst there is no specific microcontroller in mind presently, research and analysis are being done to find a suitable microcontroller for this project. A viable instance is the Microchip PIC24FJ128GA010 with the Explorer 16/32 development board, but the development board itself has limitations that must be analyzed fully before it is implemented.

4.3 Design Methods

There were many steps in the design process for developing this project. Defining a problem statement in the form of a need, objective and a set of marketing requirements are defined in Section 1 of this document. Research on existing technologies for both full-sized vehicles and for Fisher Price® Power Wheels® was performed to determine the scope and progression of the project. From this research, initial engineering analysis and refinement of the project idea was conducted. Before any true analysis was done, block diagrams and flowcharts were created to give a stronger idea of how the project systems would manifest and identify crucial inputs and outputs. After the block diagrams were created, functional requirement tables were also created to further describe the functions of the various subsystems. More in-depth engineering analysis followed. Standards, calculations, and design constraints were all produced and used to form more detailed block diagrams and functional requirements tables. With the inclusion of all of this, a realizable and organized project will be produced.

4.4 Programming Languages

This project will be programmed using the C programming language. The C programming language can be used to program many processors, including the PIC24 family of processors, as there exist many different C compilers. C is also designed to facilitate procedural programming, with the creation and use of functions being incredibly easy, and it is not object-oriented, so it helps one to avoid the bloat that comes with object-oriented programming by forcing adherence to more space-efficient programming paradigms. [NK]

4.5 Connector Standards

The existing motors use a series of spade connectors to connect to the present 'control' stick which only operates at three set velocities. These spade connections will be maintained and adapted to interface with the new Motor Control Unit (MCU).

The microcontroller (embedded system) itself will be connected to the MCU via soldered, wired connections of suitable gauge to ensure reliability.

The proprietary power connection of the pre-existing 12V lead-acid battery shall be modified to supply power to appropriate applications among the subsystems of the design as necessary. Soldered, wired connections of suitable gauge shall be employed.

Power shall be supplied to the selected development board containing the microprocessor (embedded system) via a USB Type A connection. This connection standard ensures a steady 5V power connection with tight tolerances among the Buck Converter (DC-DC converter) and development board.

5. Accepted Technical Design

The following block diagrams and functional requirement tables will outline the major subsystems of the ride-on toy vehicle accepted technical design. The block diagrams will identify key inputs, outputs and paths within each subsystem and allow for a decent visual picture of how the system will operate. The diagrams themselves are split into various levels, with subsequent levels providing increased detail and specifics. Accompanying the block diagrams will be functional requirement tables that will outline module descriptions, operations, and functionality.

5.1 Hardware Design

The various hardware diagrams will demonstrate how the various components of the system will interact and process information and inputs.

5.1.1 Block Diagram Level 0 and Functional Requirement Table

The following figure shows the Level 0 Hardware Block Diagram of the ride-on toy vehicle retrofit. Major inputs include environmental surroundings, vehicle speed (and direction) and power input. The inputs will affect how the system responds and how it triggers the various signals as displayed in the outputs (motor control, headlight control and collision warning).

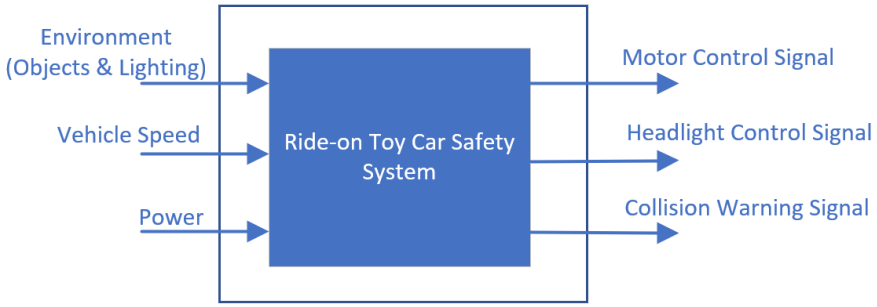

Figure 9: Level 0 Hardware Block Diagram

Table 3: Functional Requirements

Module	**Ride-On Toy Car Safety System**

Designer	Nathan Keenan, Kyle Law, Anthony Meniru, Jackson Piper
Inputs	Environment (objects and lighting) Vehicle speed Power: on-board battery, 12V DC
Outputs	Motor control signal Headlight control signal Collision warning signal
Description	Interprets data from the environment and activates motor cut-off or braking if vehicle is in motion. Activates headlights and taillights when needed.

5.1.2 Level 1 Hardware Diagram

Figure 10 shows the Level 1 Hardware Block Diagram of the ride-on toy vehicle retrofit. Essentially, this provides a more detailed view of the system compared to the Level 0 Diagram, whilst including the previous inputs and outputs. This diagram includes all five of the major subsystems: Power, Sensors, Microcontroller, Motor Control Unit, and Lighting. Along with displaying the systems, this diagram also shows in greater detail, the various paths connecting each subsystem.

As shown by Figure 10, the battery acts as an input to the power module, which then distributes the various DC voltages to the systems that require it at the time. The sensors always monitor the surrounding environment near the vehicle, assessing the situation and delivering signals for the microcontroller to interpret. These signals, both analog and digital, will be processed by the microcontroller. Once the data is processed by the microcontroller, both from the sensors and any user input, the microcontroller will deliver control signals to the motor control unit, and the lighting systems (if applicable). As shown, the motor control unit will influence the movement of the vehicle whilst the lighting system, will of course, adjust lighting accordingly.

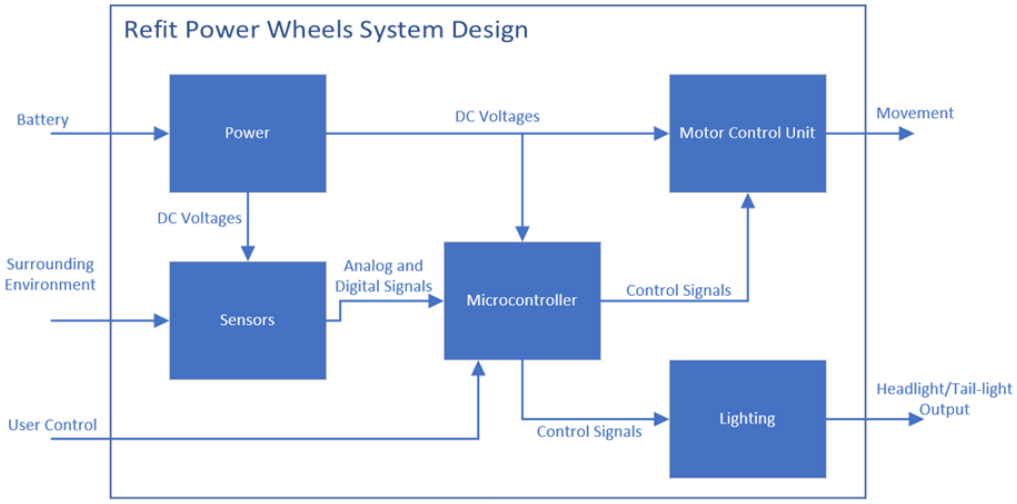

Figure 10: Level 1 Hardware Block Diagram

The following tables will provide detailed descriptions of the functional requirements of each respective module within the system.

Table 4: Level 1 Power Module Functional Requirements

Module	**Power**
Designer	Anthony Meniru
Inputs	12V Battery
Outputs	DC Voltage: Multiple voltage levels for powering various components and subsystems (4.75V – 5.25V)
Description	Converts and steps down voltages from the battery to the needed voltage levels for the various subsystem modules.

Table 5: Level 1 Sensors Module Functional Requirements

Module	**Sensors**
Designer	Nathan Keenan, Jackson Piper
Inputs	Surrounding Environment: ambient light, obstacles, terrain DC Voltages: multiple voltage levels for powering various components (3.3V, 5V)
Outputs	Analog and Digital Signals: 0 – 5V analog signals and 3.3V – 5V logic signals
Description	Monitor, track and measure the surrounding environment to avoid collision with obstacles and track ambient light for lighting scheme control purposes.

Table 6: Level 1 Microcontroller Functional Requirements

Module	**Microcontroller**
Designer	Nathan Keenan
Inputs	DC Voltage: required voltage level to power the microcontroller (3.3V) Analog and Digital signals: 0 – 5V analog signals and 3.3V – 5V logic signals User Control: acceleration, velocity, and manual braking
Outputs	Control Signals: 3.3V logic and pulse width modulated signals from the microcontroller to drive the DC motors and to activate or deactivate the headlights
Description	Process inputs from the sensors and user inputs to provide control signals for the motors and lighting system according to the system state.

Table 7: Level 1 Motor Control Unit Functional Requirements

Module	**Motor Control Unit**
Designer	Kyle Law, Jackson Piper
Inputs	12V DC Direct from power module Control Signals: 3.3V logic and pulse width modulated signals from the microcontroller to drive the DC motors
Outputs	Movement: bidirectional (forwards or backwards) movement with variable speed control
Description	The dual rear axle motors working in tandem provide either forward or backwards movement. Steering will be controlled via the front axle which is unpowered. The rear axle will be controlled via the microcontroller.

Table 8: Level 1 Lighting Module Functional Requirements

Module	Lighting
Designer	Jackson Piper
Inputs	DC Voltages: multiple voltage levels for various light intensities (3.3V – 5V) Control Signals: 3.3V logic and pulse width modulated signals from the microcontroller to activate or deactivate the lights.
Outputs	Headlights and Taillights (LED's at various rates of luminosity)
Description	Lighting will be activated in low-light environments, providing improved environmental visibility to both the end-user and surrounding pedestrians.

5.1.3 Level 2 Hardware Block Diagrams

The following figure is the Level 2 Block Diagram for power conversion, a slightly more detailed view of how power is converted and then transmitted to the other subsystems. Accompanying that are Functional Requirement tables explaining the details of the module further.

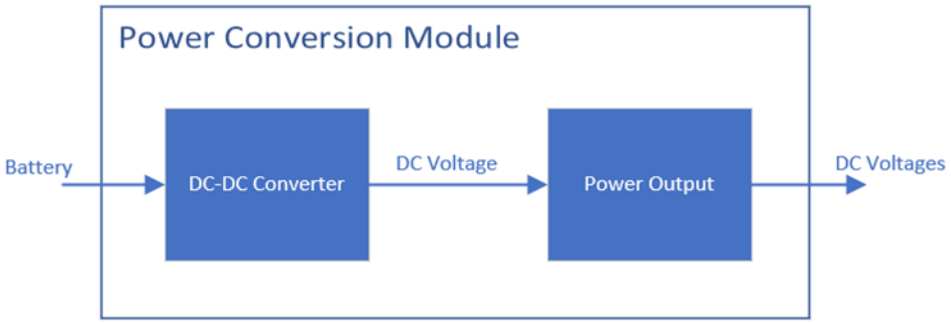

Figure 11: Level 2 Power Module Block Diagram

Table 9: Level 2 DC-DC Converter Module Functional Requirements

Module	DC-DC Converter
Designer	Anthony Meniru
Inputs	12V Battery
Outputs	DC Voltage: Multiple voltage levels for powering various components and subsystems (4.75V – 5.25V)
Description	Steps down and regulates voltages from the battery to the needed voltage levels for the various subsystem modules.

Table 10: Level 2 Power Output Module Functional Requirements

Module	Power Output

Designer	Anthony Meniru
Inputs	DC Voltages (from DC-DC converter)
Outputs	DC Voltages: multiple voltage levels (4.75V – 5.25V) for powering various components
Description	Transmits stepped down voltages from DC-DC converter to the various subsystems.

The following figure represents a detailed view of the motor control unit, separated into two separate components: the motor controller chipset, and the actual DC motors. Accompanying this are two Functional Requirements tables giving a description of each component.

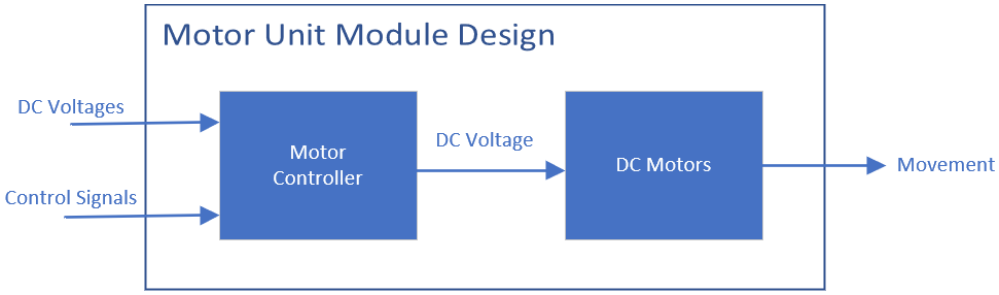

Figure 12: Level 2 Motor Unit Module Design

Table 11: Level 2 Motor Unit Module Functional Requirements

Module	**Motor Controller**
Designer	Kyle Law
Inputs	DC Voltages: 12V (from DC-DC converter) Control Signals: 3.3V logic and pulse width modulated signals from the microcontroller to drive the DC motors.
Outputs	DC Voltages: 12V to the motors
Description	Adjusts the duty cycle of the PWM delivered to motors to modify speed, acceleration and overall motion of the motors.

Table 12: Level 2 DC Motors Module Functional Requirements

Module	**DC Motors**
Designer	Kyle Law (Pre-Existing Motors)
Inputs	DC Voltages: 12V from the Motor Controller
Outputs	Bidirectional movement (forwards or backwards) and variable speed
Description	Takes the variable inputted voltage from the motor controller to then create rotation in the motors and provide variable speed.

The last Level 2 figure is the sensors module, which outlines how the system will obtain and accumulate data from the environment and transmit the data to the microcontroller using both analog and digital signals. This is also accompanied by a functional requirements table.

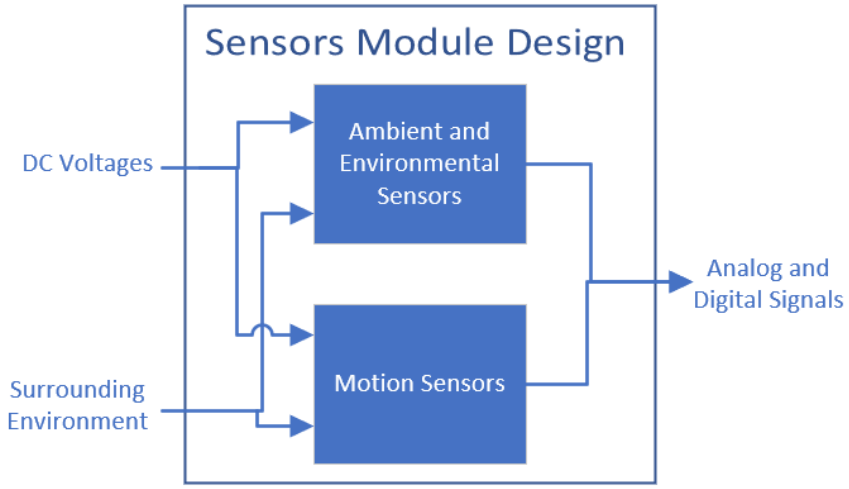

Figure 13: Level 2 Sensors Module Diagram

Table 13: Level 2 Sensors Module Functional Requirements

Module	**Sensors**
Designer	Nathan Keenan, Jackson Piper
Inputs	DC Voltage: Multiple voltage levels for powering various components and subsystems (3.3V, 5V) Surrounding Environment: Obstacles, hazards and light
Outputs	Analog and digital signals transmitted to the microcontroller
Description	Sensors will take in data from the surrounding environment to detect obstacles, ambient light conditions, and any anomalies with vehicle speed

5.1.4 Level 3 Hardware Block Diagrams

The following figures and tables represent an increased level of detail (level 3) of the various subsystem modules and how they will function. Whilst these will not yet include specific schematics, it is intended to provide a more detailed overview of how each hardware subsystem behaves. This iteration begins again with power conversion; how power from the battery will be converted and transmitted into voltages for the various subsystems.

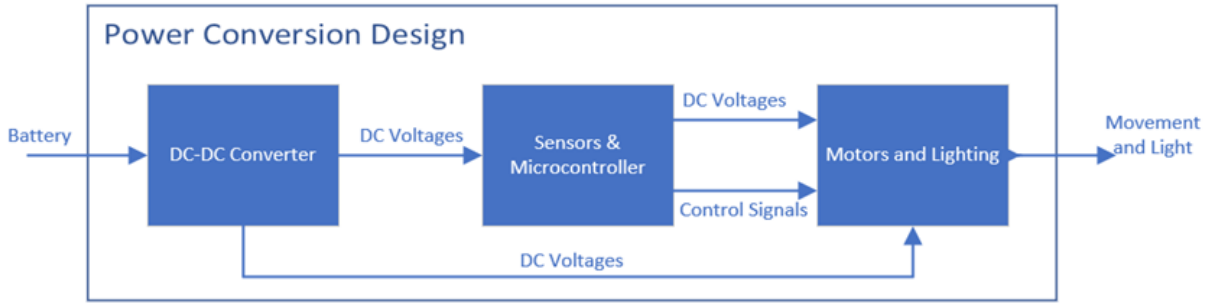

Figure 14: Level 3 Power Conversion Flowchart

Table 14: Level 3 Power Conversion Functional Requirements

Module	**Power Conversion**
Designer	Anthony Meniru
Inputs	12V Battery
Outputs	DC Voltage: Multiple voltage levels for powering various components and subsystems (4.75V – 5.25V)
Description	Converts and steps down voltages from the battery to the needed voltage levels for the various system modules. Responsible for the entire power supply of the system.

The MCU is displayed next, showing how the motor controller itself will receive information and then activate the motors accordingly. It is important to note both motors work synchronously with each other.

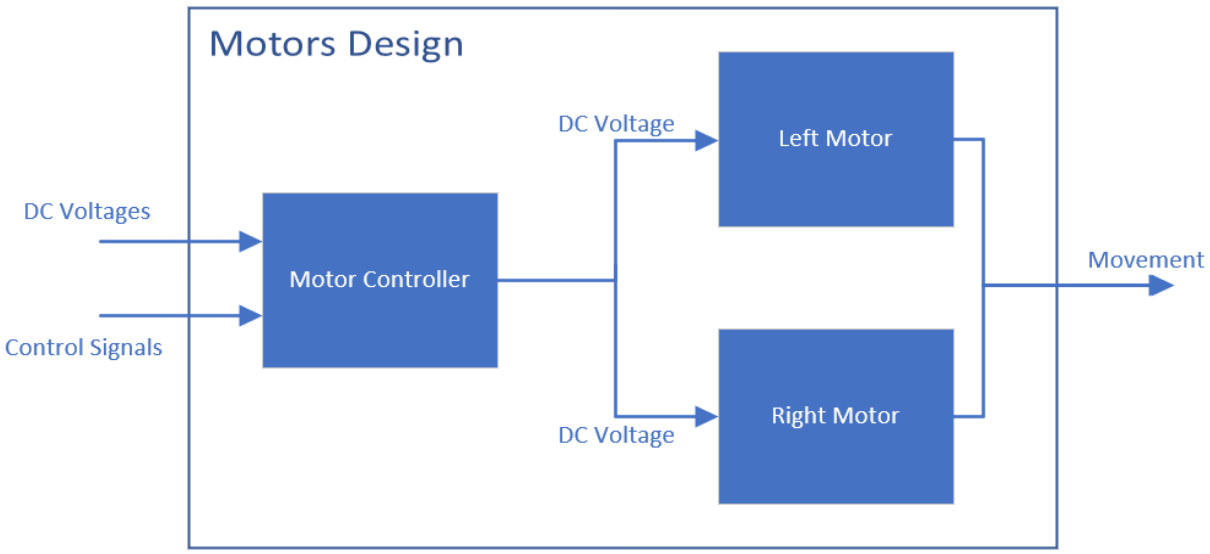

Figure 15: Level 3 Motor Control Unit Diagram

Table 15: Level 3 Motor Control Unit Functional Requirements

Module	**Motor Control Unit**
Designer	Kyle Law
Inputs	DC Voltages: 12V from power module Control Signals: 3.3V logic and pulse width modulated signals from the microcontroller to drive the DC motors.
Outputs	Bidirectional movement (forwards or backwards) and variable speed
Description	The control unit receives power and then information from the microcontroller including movement commands. Motor controller then sends voltages to the right and left motor, providing forward or backwards motion. Both motors function synchronously to create movement. The motors do not articulate or rotate, steering is relegated to the unpowered front axle.

Lastly, the sensors subsystem is shown. This demonstrates how the system will receive information about the environment surrounding the vehicle and send that information to the microcontroller to be processed. Without the sensors, the microcontroller is rendered inoperable to the closed-loop feedback of the entire accepted technical design of the system.

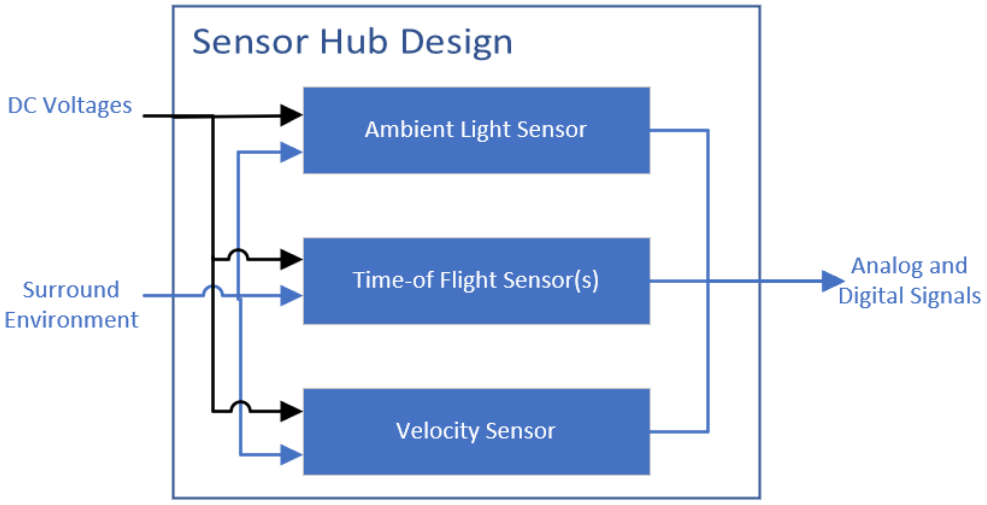

Figure 16: Level 3 Sensor Hub Diagram

Table 16: Level 3 Sensor Hub Functional Requirements

Module	**Sensor Hub**
Designer	Nathan Keenan, Jackson Piper
Inputs	DC Voltage: Multiple voltage levels for powering various components and subsystems (3.3V, 5V) Surrounding Environment: Light intensity, motion of the vehicle, obstacles, and hazards
Outputs	Analog and digital signals transmitted to the microcontroller
Description	An ambient light sensor will detect light intensity surrounding the vehicle and create signals to send to the microcontroller to activate or deactivate the lights. Time of flight sensor(s) will act as the “eyes and ears” of the vehicle, transmitting environmental data to the microcontroller. Hall effect sensor will detect the vehicles velocity and relate that to oncoming obstacles or hazards, as well as detect and mitigate uncontrollable downhill acceleration.

5.1.5 Schematics with Explanations

The following figures are the schematics of each subsystem used to design the complete retrofit system for the ride-on toy vehicle. The first subsystem considered is the Power Regulation system, which consists of one DC Regulator that will deliver 5V DC to the Explorer 16/32 development board. The LM7805C is a buck converter that can transmit an output voltage range from 4.75V to 5.25V when it is powered with an input voltage between 7.5V and 20V. The following figure shows the schematic diagram of the regulator with a variety of embedded components such as resistors, Zener diodes, capacitors, and transistors.

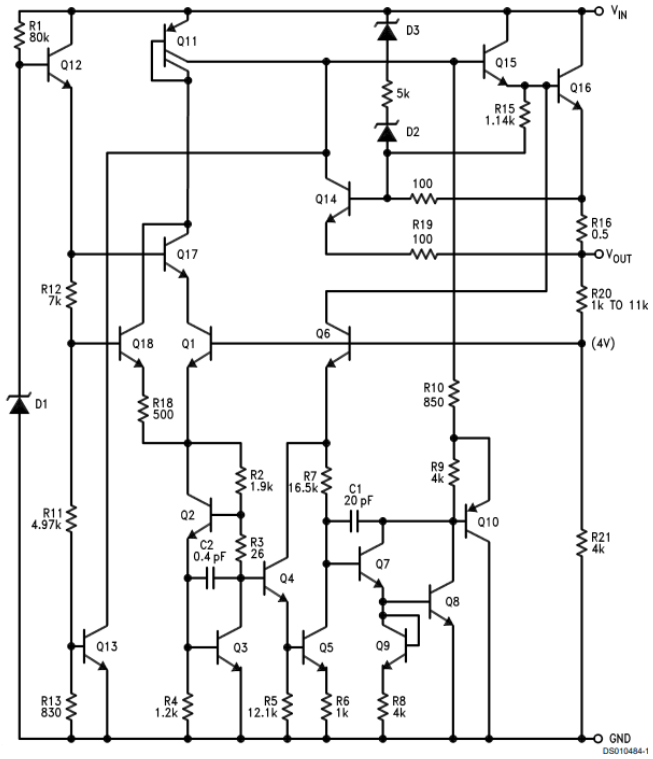

Figure 17: Schematic Diagram of LM7805C Regulator

(from National Semiconductor datasheet)

Outside of the regulator, there are three capacitors that are connected between the regulator, ground, input, and output of the circuit (as shown in the figure below). A 0.33µF capacitor is connected to the 12V battery and the input pin of the LM7805C. The 0.1µF and

10µF capacitors are connected in parallel to the output pin of the DC regulator and the output of the circuit. In addition, all capacitors are shorted together with the ground pin to ground. The figure below shows the simulation of the DC regulator circuit on LTSPICE. The Vout flag represents the output load which is where the 5V (approximately 5.009V from LTSPICE waveform on Figure 19) is being delivered.

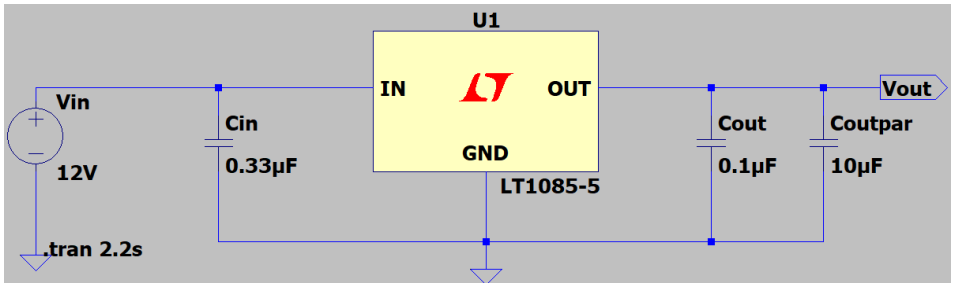

Figure 18: LTSPICE Schematic Diagram of DC Regulator

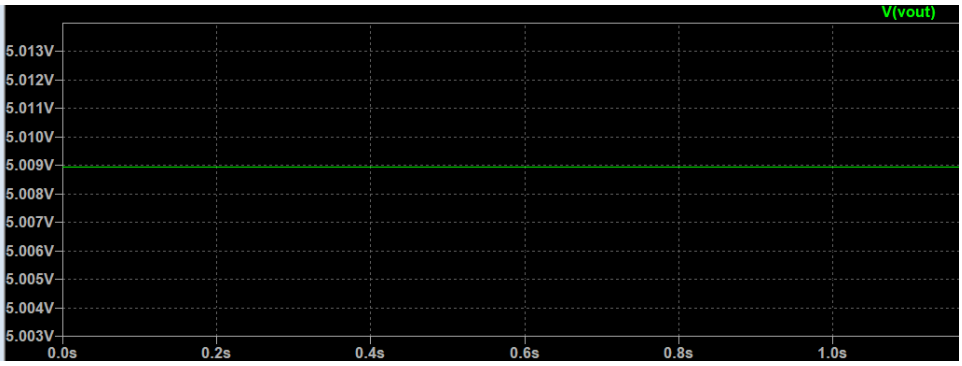

Figure 19: Output Voltage waveform (from LTSPICE Simulation)

The actual implementation is tested by connecting the capacitors and the LM7805C regulator to a breadboard circuit following the same scheme as the schematic. An external DC power supply is connected to the input rail of the board to act as the battery for the circuit. The power supply is set to 12V which is the same as the SafeAMP battery being used in the project. The output voltage is measured using a multimeter, which reads a voltage of 5.09V. Additionally, the regulator circuit was tested at other voltage ranges to evaluate how well the circuit regulates the voltage when the input is not strictly 12V. The regulator can maintain the same output voltage

between an input voltage range of 8V (low power mode) and 14.5V (fully charged). Lastly, the circuit was tested with a red LED to ensure it can deliver power to a load. The regulator can power the LED with the same output voltage at the same input voltage range.

After successfully testing the circuit on a breadboard, the same components are connected to a solder board to test the same conditions. After testing regulator power circuit on its own, a USB adapter is soldered to the power circuit between both the output and ground connections. A USB type A plug is plugged into the Explorer 16/32 board to deliver a regulated 5V to the PIC24FJ128GA010. The power circuit is connected to the Explorer development board to act as a regulator to deliver 5V to electronic components that require less than the battery voltage. [AM]

For the Motor Control Unit subsystem, after various chips were considered and different types of circuits were tested, the Sabertooth 2x25 Dual Motor Driver was used to implement analog motor control compared to the original ride-on toy vehicle design of simply turning the motors on or off. This Motor Control Unit arrived fully built with supporting electronic components embedded. The only modification that was required was the addition of a lowpass filter circuit (Resistor-Capacitor Circuit) to properly implement the pulse-width modulation (PWM) from the PIC24 microprocessor to use as an analog input signal. The battery and two DC motors directly connect to this motor driver, with each motor able to be supplied 25A sustained, which is more than enough for the purposes of this project.

Figure 20: Sabertooth 2x25 Dual Motor Driver (3D Model)

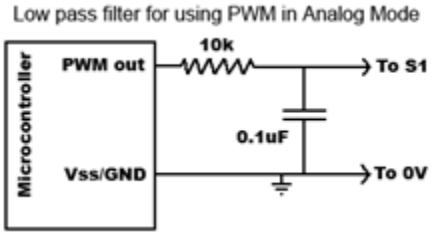

Figure 21: Required Low pass filter for PWM Analog Control

Connecting the driver and lowpass filter to the PIC24 allows for full analog control in both forward and reverse using the programming code that has been developed. Based on the average value of the PWM (which the lowpass filter creates), the motors will either turn forward (counterclockwise for the left, clockwise for the right), reverse (CW/CCW), idle, or brake (idle and then short bursts of reverse or forward to act as a braking mechanism). This design and program have been tested and implemented successfully, no further modification or design is required. [KL]

The design of the lighting subsystem of the retrofitted ride-on toy vehicle may be abstracted into two mechanisms: 1) environmental feedback via ambient light sensor, and 2) actual electrical control of individual lighting schemes.

As discussed in **5.1.4 Level 3 Hardware Block Diagrams**, an ambient light sensor detects environment light intensity surrounding the ride-on toy vehicle and delivers light intensity data to the microcontroller subsystem in real time. To quantify light intensity data reliably and accurately, a digital ambient light sensor is selected. Additionally, the use of a digital ambient light sensor allows for seamless integration with the selected microcontroller, for the microcontroller is confirmed compatible with multiple digital signal communication protocols. Ultimately, the Vishay Semiconductors VEML7700 ambient light sensor is selected. The surface-mounted package of the selected ambient light sensor necessitates the use of a prefabricated adaptor board for proper integration, supplied by Adafruit Industries. The VEML7700 makes use of the Inter-Integrated Circuit (I^2C) digital communication protocol to communicate with the microcontroller. See **Appendix** for sufficiently commented code and functions outlining the implementation of the I^2C protocol with the Microchip PIC24 microprocessor. Upon consulting the datasheet of the VEML7700 ambient light sensor, functions are developed to convert the raw digital data transmitted from the sensor (in the form of 2-byte words) into absolute values of lux, the SI unit of light intensity. An external laboratory lux meter is utilized to calibrate the sensor accordingly.

Subsequently, individual lighting schemes for the ride-on toy vehicle are devised, along with necessary supporting electrical control methods. Much like its analogous, full-sized vehicle counterpart, the ride-on toy vehicle shall feature functional white headlights, white fog lights, red braking taillights, amber running lights, and white backup lights. Within software, lux thresholds are defined for which the headlights and fog lights shall autonomously illuminate. For example,

when environmental lighting conditions are below the threshold of 150 lux as read by the ambient light sensor (approximately the light intensity observed during a dark, overcast sky at dawn or dusk), the fog lights will activate at full brightness. Amber running lights shall be illuminated continuously whenever the vehicle is powered and operational. The red braking taillights shall be illuminated when the vehicle employs active braking. The white backup lights shall be illuminated when the vehicle operates in the reverse direction scheme.

For the most effective lighting implementation, 12V light-emitting diode (LED) signal lights are integrated into the design of the lighting subsystem. This selection is made in consideration of the minimal power draw of the LEDs, as well as their exceptional luminosity and lifespan. Consequently, while the selected LED signal lights operate most effectively with a 12V power source (that is, the preexisting lead-acid battery of the ride-on toy vehicle), they cannot be independently controlled by the microcontroller without supporting driver circuitry. The input and output ports of the Microchip PIC24 microcontroller operate at a nominal voltage of 3.3V, while exhibiting a voltage of 1.3V when under load. Additionally, one must consider the driving current limitations of the microcontroller. Designs making use of relays to drive the lighting schemes are rendered inoperable because the minimal current that can be delivered by the output pins of the microcontroller is unable to drive the closing coil of even the most sensitive Single Pole Single Throw (SPST) relays.

Alternatively, general purpose NPN bipolar junction transistors (BJTs) allow for integration among the lighting schemes and the microcontroller without unwarranted complexity. The general purpose 2N2222A is selected for its nominal base-emitter threshold voltage (V_{BE}) of approximately 0.7V, which is ideal for use with the input and output ports of the Microchip PIC24 microcontroller. Essentially, for each circuit associated with each respective lighting scheme, the BJT acts as a robust “switch” that can still supply each signal light with its required

12V power while maintaining functionality with the microcontroller. For each lighting scheme driver circuit, 12V is fed from the terminal block of the battery supply to the collector pin of the BJT. When a high output signal is given by the microcontroller to the base of the BJT (that is, the associated output pin of the microcontroller is brought high to signal illuminating a lighting scheme), the "switch" of the BJT is closed, allowing for power to be delivered to the appropriate lighting scheme from the battery supply. The emitter of the BJT is connected to chassis ground. The signal ground associated with the base of the BJT is also connected to chassis ground. See Figure 22 for the topology of headlights lighting scheme circuit for example. This circuit is employed similarly with each respective lighting scheme. [JP]

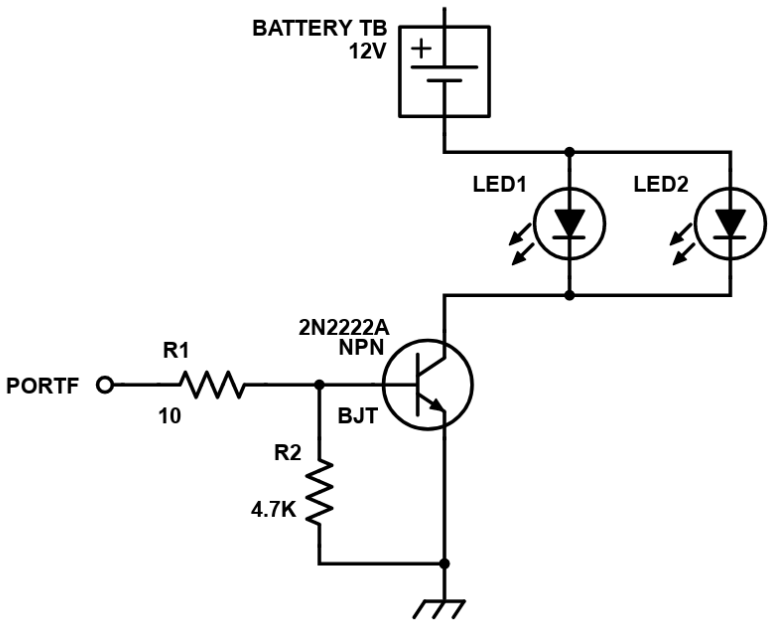

Figure 22: General Schematic of Lighting Scheme

5.2 Software Design

The program responsible for controlling the system will adhere to a more procedural programming paradigm, where the various jobs that the program performs can be split into several neatly defined subroutines. This programming paradigm makes each section of the code more reusable and easier to modify and debug while also avoiding the code bloat that comes with other programming paradigms such as state-machines and object-oriented programming. This will be especially useful for this program, where its internal state does not change greatly.

5.2.1 Level 0 Software Behavior Models

The entry point to the program will be a main function that begins with any necessary initialization of the system and immediately transitions into an infinite loop that will govern the rest of the program. This loop will call functions to gather sensor data, perform any necessary calculations, make any necessary updates to the minimal internal state of the program, and pause for a specified period to match the desired framerate.

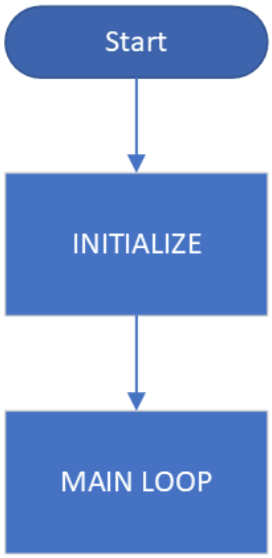

Figure 23: Level 0 System Software Block Diagram

Table 17: Level 0 Software Functional Requirements

Routine	**Overall System Software**
Designer	Nathan Keenan
Inputs	Analog and Digital Inputs from sensors DC Voltages: 3.3V, 5V from DC-DC converter for various components of microcontroller
Outputs	Control Signals: 3.3V logic and pulse width modulated signals from the microcontroller to drive the DC motors and activate/deactivate lights.
Description	System software will operate using the microcontroller to process and deliver information and commands to the various subsystems.

5.2.2 Level 1 Software Behavior Models

The main loop, which consists of the driving code after the initialization, can be broken down into the different subroutines that it calls. These subroutines include SENSORS, CALCULATIONS, COLLISION, VELOCITY, PEDAL, and LIGHTS. The first three subroutines make up the main collision detection system. In SENSORS, the sensors are read and their output converted into manipulable distances. Then in CALCULATIONS, the time-to-collision values are calculated. Finally, in COLLISION, it is determined whether the vehicle is on a collision path and whether the user should be warned of the collision or automatic braking should be employed. The next two subroutines round out the speed control system, which is only active whenever the user has control of the vehicle. In VELOCITY, the velocity of the vehicle is calculated and it is determined whether or not the vehicle must be slowed in order to keep the vehicle's velocity below a predetermined safety level. Next, in PEDAL, the appropriate PWM signal is sent to the motor controller based on the position of the potentiometer foot pedal and the other motor control signals. Lastly, in LIGHTS, the control signals for the headlights are determined. Between some of these subroutine calls, the main loop must also update the internal state of the program, including the current frame's distance sensor readings and the previous frame's distance sensor readings, as well as some of the motor control signals. The pause for framerate synchronization is also shown in this flowchart.

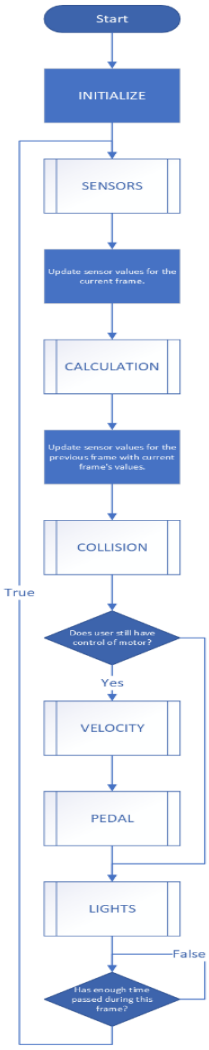

Figure 24: Level 1 System Software Block Diagram

Table 18: Level 1 Software Functional Requirements

Routine	**Overall System Software**
Designer	Nathan Keenan
Inputs	Analog and Digital Inputs from sensors DC Voltages: 3.3V, 5V from DC-DC converter for various components of microcontroller
Outputs	Control Signals: 3.3V logic and pulse width modulated signals from the microcontroller to drive the DC motors and activate/deactivate lights.
Description	System software will operate using the microcontroller to process and deliver information and commands to the various subsystems. Utilizes three separate subroutines to determine the state of the system and activate collision warnings.

5.2.3 Level 2 Software Behavior Models

When the program begins, any required initialization will be run, including some preliminary sensor readings and calibration. The program will then enter its infinite loop, reading sensor input, performing delta and collision calculations, and outputting control signals until the program is terminated. The program will then determine the time until a collision is set to happen using the input from each individual sensor. The velocity of an object is defined by the following equation:

$$v = \frac{d}{t}$$

Where v is the velocity, d is the displacement, and t is the change in time. This equation can be rearranged to produce the following equation:

$$t = \frac{d}{v}$$

This gives us a method for calculating the time t that it would take an object to traverse a displacement of d at a velocity of v. The distance from the vehicle to the obstacle can be found directly from the sensor input, but the velocity requires some calculation. Fortunately, this calculation can be made quite easily. Assume we run the program at a specified framerate f_0, such that the time between frames is $\frac{1}{f_0} = t_0$, checking the sensors once every frame. Our velocity relative to the object detected by the sensor can therefore be calculated in the following manner:

$$v = \frac{d_n - d_{n-1}}{\frac{1}{f_0}} = (d_n - d_{n-1})f_0 = \frac{d_n - d_{n-1}}{t_0}$$

Where d_n is the distance to the obstacle given by one sensor on frame n, and d_{n-1} is the distance to the obstacle given by that same sensor the frame before. Of course, it is easy to see that the velocity will become negative as the vehicle approaches an object since d_n will be less than d_{n-1}, which may interfere with time-to-collision calculations. It is therefore desirable to use

negative velocity in our calculations so that the collision detection logic will work properly. Thus, the time until the vehicle is set to collide with the obstacle can be calculated as follows:

$$t = \frac{d}{-v} = \frac{d_n}{\frac{-(d_n - d_{n-1})}{\frac{1}{f_0}}} = \frac{d_n}{(d_{n-1} - d_n)f_0} = \frac{d_n}{\frac{d_{n-1} - d_n}{t_0}} = \frac{d_n t_0}{d_{n-1} - d_n}$$

Thus, if the vehicle is approaching an object (d_n is less than d_{n-1}), the calculation for time-to-collision will be positive and can be checked against a threshold by checking if it is less than our allotted time. If this calculation produces a negative result, it can safely be ignored. However, the check for a negative result should occur before the check against the allotted collision time to ensure that there are no false positives.

One potential problem with this method of calculation is the possibility of a division by zero if a sensor happens to return the same exact value for two adjacent frames. To avoid this disturbing the system, the difference $d_{n-1} - d_n$ should be compared to zero before it is used in calculations. If it is equal to zero, the time-to-collision calculation will be skipped entirely, and it will be automatically determined that no collision is imminent for that frame.

The program must also be able to determine the absolute velocity of the vehicle in order to ensure that it stays below a predetermined safety threshold. Assuming that there is only one Hall effect sensor on the vehicle, the velocity of the vehicle can be calculated in the following manner:

$$v = \frac{d}{t} = \frac{2\pi r}{t - t_h}$$

Where r is the radius of the wheels of the vehicle, t is the current system time, and t_h is the system time recorded the last time the Hall effect sensor was noted to be in a high state. Using this equation, we can determine the velocity of the vehicle independent of its surroundings, allowing us to implement accurate safety measures.

This program must also use the analog-to-digital converter to convert the voltage across the potentiometer foot pedal into a digital signal such that the voltage across the potentiometer foot pedal when the pedal is not depressed at all produces a PWM signal with a duty cycle of 0% and the voltage across the potentiometer foot pedal when the pedal is completely depressed produces a PWM signal with a duty cycle of 100%. This PWM signal is sent to the motor controller, but only if there are no other signals overriding it.

It should also be noted that small inconsistencies in the sensors may cause collision calculations to be inaccurate from time to time. To avoid an inconsistent calculation triggering the safety systems when not necessary, the program should implement a sort of debounce, where the collision warning or automatic braking is only employed when multiple frames of calculations produce consistent results. This same tactic will also be used to determine whether the vehicle's path is clear after the automatic braking system has been employed.

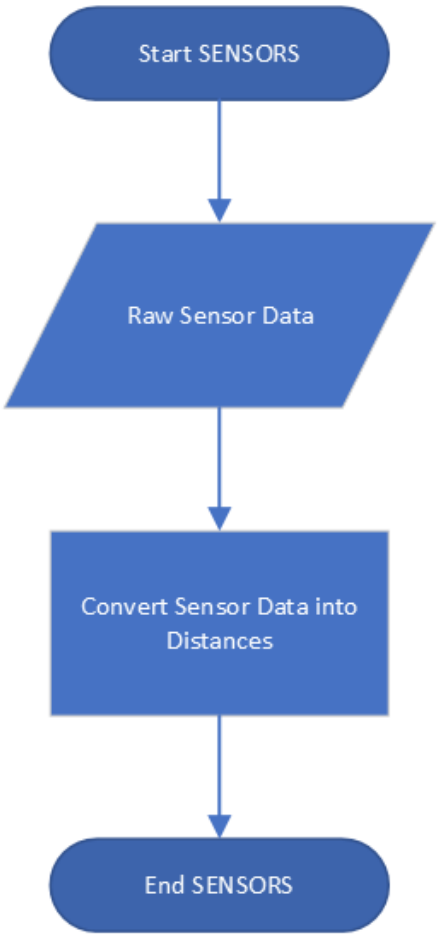

Figure 25: Level 2 SENSORS Subroutine Block Diagram

Table 19: Level 2 SENSORS Subroutine Functional Requirements

Subroutine	**SENSORS**
Designer	Nathan Keenan
Inputs	Raw sensor data
Outputs	Processed sensor data
Description	This subroutine will take raw sensor data and process it into manipulable distances that will be saved to the program's internal state and used in the following subroutine (CALCULATION). The data output from this subroutine will be distances to collision.

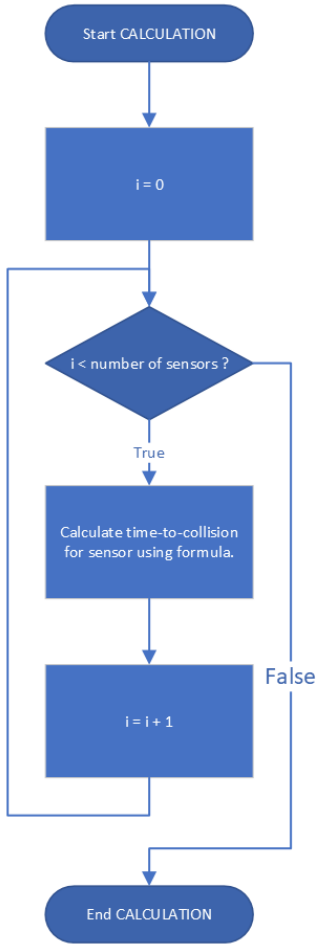

Figure 26: Level 2 CALCULATION Subroutine Block Diagram

Table 20: Level 2 CALCULATION Subroutine Functional Requirements

Subroutine	**CALCULATION**
Designer	Nathan Keenan
Inputs	Processed sensor data (distance to obstacle)
Outputs	Calculated time to collision
Description	This subroutine will take the processed and converted data from the SENSORS subroutine and make calculations for time-to-collision based off the distance to obstacle in the current frame and the distance to obstacle from the previous frame.

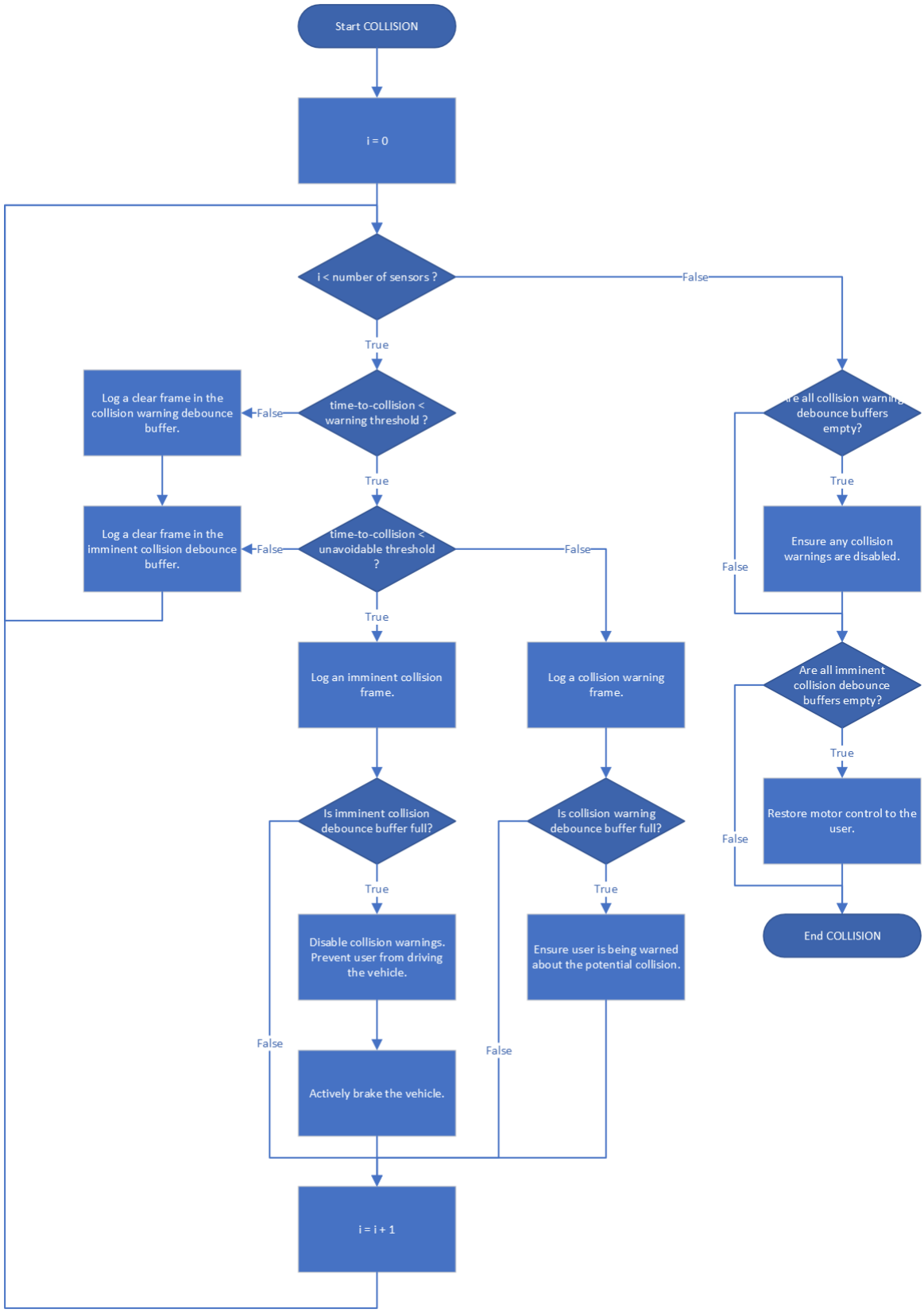

Figure 27: Level 2 COLLISION Subroutine Block Diagram

Table 21: Level 2 COLLISION Subroutine Functional Requirements

Subroutine	**COLLISION**
Designer	Nathan Keenan
Inputs	Time-to-collision (from CALCULATION subroutine)
Outputs	Collision warning signal Motor control signal (brake or continue normal operations)
Description	This subroutine is the most complex and will take into account the calculated time-to-collision from the previous subroutines. Based off the calculation this subroutine will make decisions to stop the vehicle or let it continue operations as normal.

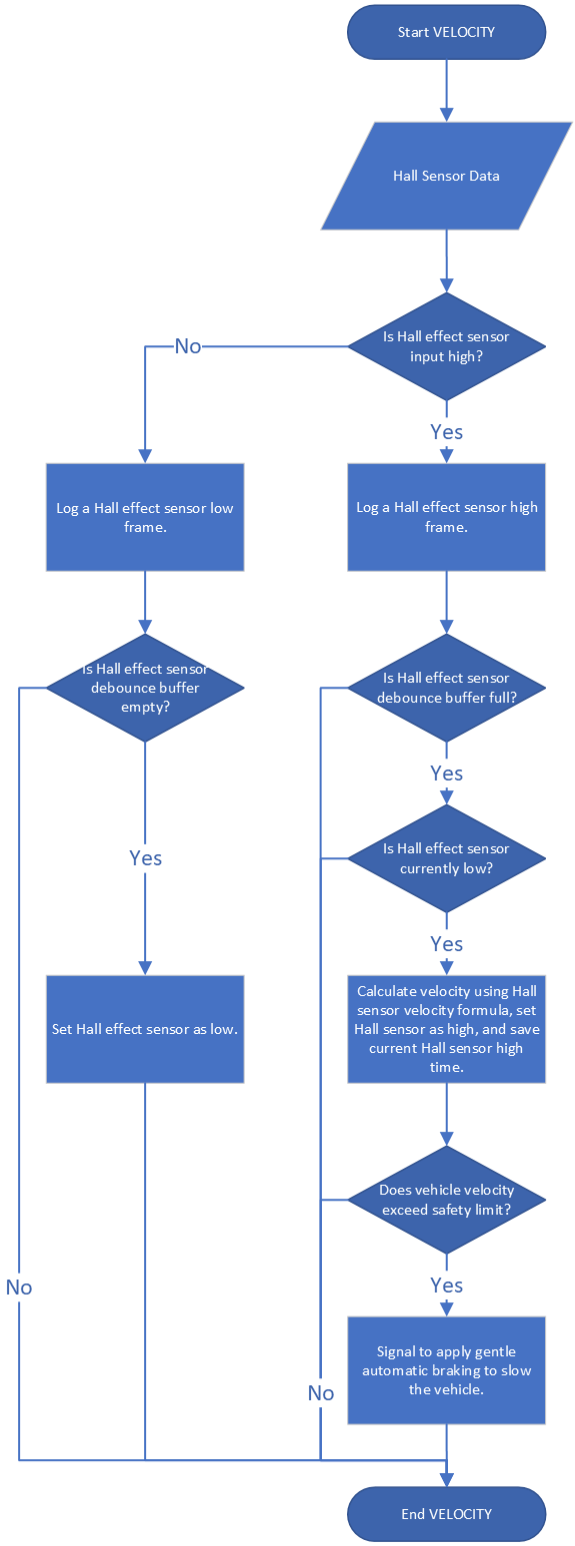

Figure 28: Level 2 VELOCITY Subroutine Block Diagram

Table 22: Level 2 VELOCITY Subroutine Functional Requirements

Subroutine	**VELOCITY**
Designer	Nathan Keenan
Inputs	Hall Effect Sensor Data
Outputs	Motor control signal (brake or continue normal operations)
Description	This subroutine will take the input data from the Hall effect sensor and determine whether or not continuous automatic braking of the vehicle is necessary in order to keep the velocity of the vehicle below a predetermined safety threshold.

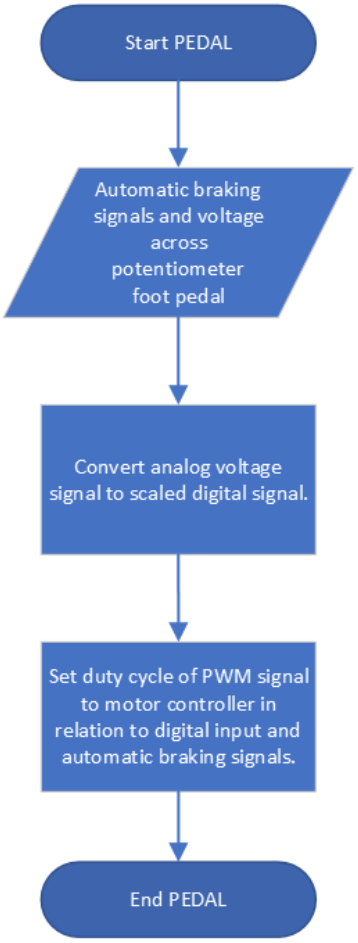

Figure 29: Level 2 PEDAL Subroutine Block Diagram

Table 23: Level 2 PEDAL Subroutine Functional Requirements

Subroutine	**PEDAL**
Designer	Nathan Keenan
Inputs	Automatic Braking Signal (from VELOCITY subroutine) Potentiometer Foot Pedal Voltage
Outputs	PWM Signal to Motor Controller
Description	This subroutine will take the input data from the potentiometer foot pedal and output the appropriate PWM signal to the motor controller to ensure that the vehicle is moving at the speed the user desires. This subroutine will also take into account whether or not the vehicle must be slowed down for safety reasons.

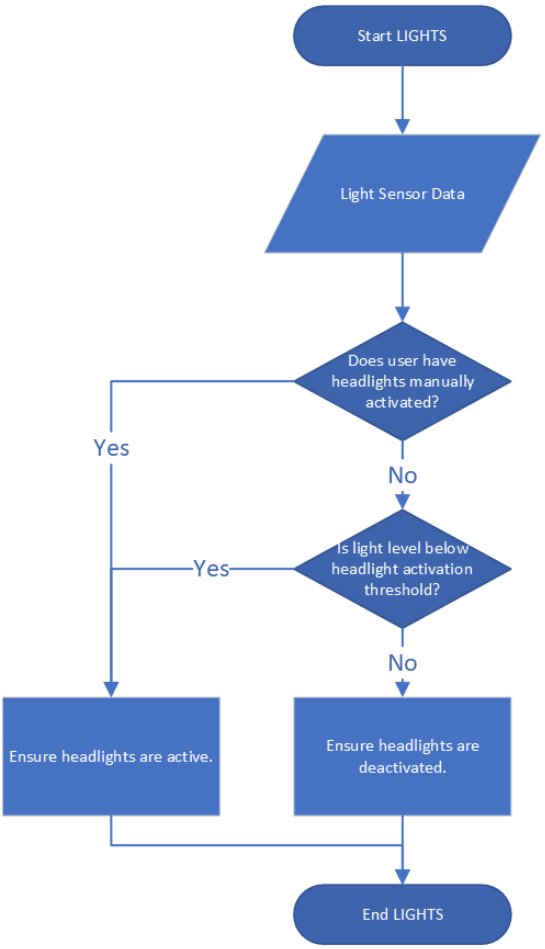

Figure 30: Level 2 LIGHTS Subroutine Block Diagram

Table 24: Level 2 LIGHTS Subroutine Functional Requirements

Subroutine	**COLLISION**
Designer	Nathan Keenan
Inputs	Light sensor data User headlight control signal
Outputs	Headlight control signal
Description	This subroutine simply checks sensor and user data to determine whether the headlights should be activated. The structure of the subroutine ensures that the headlights are always activated in dark conditions, regardless of user control.

6. Mechanical Sketch

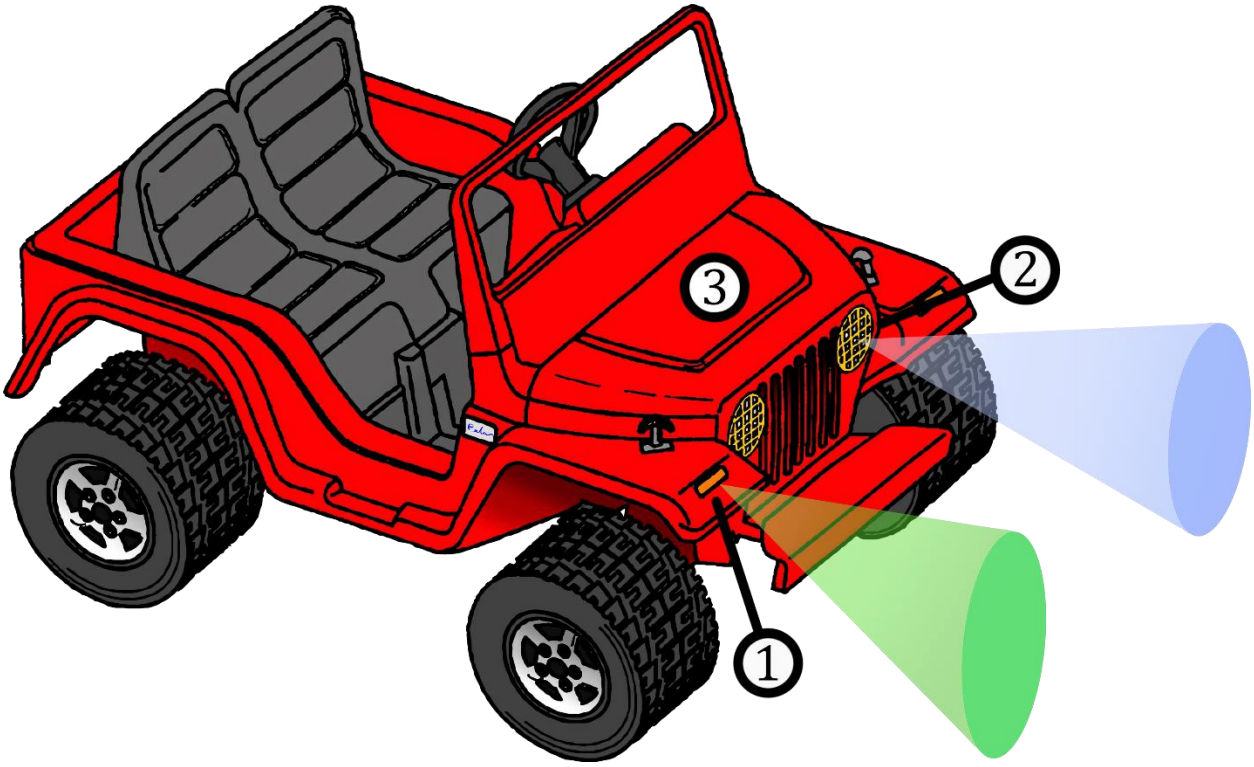

Figure 31: Mechanical Sketch

1. Ultrasonic Time-of-Flight sensor (approximate field-of-view and mounted location of module, one of four identical modules).

2. Automatic LED (Light Emitting Diodes) headlights (approximate field of view of beam and location of lamp, one of two identical lamps).

3. 12V rechargeable lead-acid battery with supporting power electronics and embedded system (concealed beneath removable hood of vehicle for ease of access).

Note: Image not to scale.

[JP]

7. Parts List

Purchased (Based on Internal Records, Does Not Include All Parts from Previous Semester)

- 1 L6229DTR Motor Control Chip (1st M.C.C. Iteration) $8.13
- 1 TB67H303HG IC Chip (2nd M.C.C. Iteration) $8.58
- 1 LTC3851AEMSE#TRPBF DC-DC Buck Converter $5.87
- 1 16MSOP surface mount adapter $6.79
- 1 OP747 Operational Amplifier IC Chip $10.42
- 1 14SOIC surface mount adapter for op amp IC $3.49
- 1 Adafruit VEML7700 Ambient Light Sensor $4.95
- 2 ST Microelectronics VL53L1X-SATEL ToF Sensor $30.27
- 1 Littelfuse 55100-2M-02-A Hall Effect Sensor $8.77
- 2 Toshiba Tk090E65Z,S1X Power MOSFETs $10.72
- 5 STB45NF06T4 N-Channel MOSFTETs $9.35
- 5 PA0184 TO-263-3 to DIP-6 SMT Adapter Boards $20.95
- 1 YK-B-337 Electric Gas Pedal $17.99
- 1 DC-2020702ZW Strip Lights $8.29
- 1 CAR-029-RAW Head, Tail & Signal Lights (12V) $19.99
- 8 TB67H420FTG,EL CMOS Motor Drivers $37.08
- 4 PA0073 48-DIP to 48-SMT Adapter Boards $50.36

Budget Allocated: $600.00

Budget Remaining: $231.53

Budget Used: $368.47 [KL]

Non-Purchased

- 1 Microchip PIC24FJ128GA010 Microcontroller with Explorer 16/32 Development Board
- 1 Sabertooth 2x25 Motor Controller from Dimension Engineering
- Repurposed Fisher Price® Power Wheels® chassis, Motors and other Recycled Parts
- 12V SafeAMP 9.5Ah Lead-Acid Battery
- 4 HC-SR04 Ultrasonic Time-of-Flight Sensors
- 5 2N2222A General Purpose NPN Bipolar Junction Transistors
- Resistors, Inductors, and Capacitors Not Listed

8. Project Schedules

	SDP I 2022			
	Project Design	**95 days**	**Wed 8/24/22**	**Sun 11/27/22**
	Midterm Report	46 days	Wed 8/24/22	Sun 10/9/22
	Cover page	46 days	Wed 8/24/22	Sun 10/9/22
	T of C, L of T, L of F	46 days	Wed 8/24/22	Sun 10/9/22
	Problem Statement	**46 days**	**Wed 8/24/22**	**Sun 10/9/22**
	Need	46 days	Wed 8/24/22	Sun 10/9/22
	Objective	46 days	Wed 8/24/22	Sun 10/9/22
	Background	46 days	Wed 8/24/22	Sun 10/9/22
	Marketing Requirements	46 days	Wed 8/24/22	Sun 10/9/22
	Engineering Requirements Specification	46 days	Wed 8/24/22	Sun 10/9/22
	Engineering Analysis	**46 days**	**Wed 8/24/22**	**Sun 10/9/22**
	Circuits (DC, AC, Power, ...)	46 days	Wed 8/24/22	Sun 10/9/22
	Electronics (analog and digital)	46 days	Wed 8/24/22	Sun 10/9/22
	Signal Processing	46 days	Wed 8/24/22	Sun 10/9/22
	Communications (analog and digital)	46 days	Wed 8/24/22	Sun 10/9/22
	Electromechanics	46 days	Wed 8/24/22	Sun 10/9/22
	Computer Networks	46 days	Wed 8/24/22	Sun 10/9/22
	Embedded Systems	46 days	Wed 8/24/22	Sun 10/9/22
	Controls	46 days	Wed 8/24/22	Sun 10/9/22
	Accepted Technical Design	**46 days**	**Wed 8/24/22**	**Sun 10/9/22**
	Hardware Design: Phase 1	**46 days**	**Wed 8/24/22**	**Sun 10/9/22**
	Hardware Block Diagrams Levels 0 thru N (w/ FR tables)	46 days	Wed 8/24/22	Sun 10/9/22
	Software Design: Phase 1	**46 days**	**Wed 8/24/22**	**Sun 10/9/22**
	Software Behavior Models Levels 0 thru N (w/FR tables)	46 days	Wed 8/24/22	Sun 10/9/22
	Mechanical Sketch	46 days	Wed 8/24/22	Sun 10/9/22
	Team information	46 days	Wed 8/24/22	Sun 10/9/22
	Project Schedules	**46 days**	**Wed 8/24/22**	**Sun 10/9/22**
	Midterm Design Gantt Chart	19 days	Wed 8/24/22	Mon 9/12/22
	References	46 days	Wed 8/24/22	Sun 10/9/22
	Midterm Parts Request Form	50 days	Wed 8/24/22	Thu 10/13/22
	Midterm presentation file submission	33 days	Wed 8/24/22	Mon 9/26/22
	Midterm Design Presentations Day 1	0 days	Wed 9/28/22	Wed 9/28/22
	Midterm Design Presentations Day 2	0 days	Wed 10/5/22	Wed 10/5/22
	Project Poster	14 days	Tue 10/11/22	Tue 10/25/22
	Final Design Report	47 days	Tue 10/11/22	Sun 11/27/22
	Abstract	47 days	Tue 10/11/22	Sun 11/27/22
	Hardware Design: Phase 2	**47 days**	**Tue 10/11/22**	**Sun 11/27/22**
	Modules 1...n	**47 days**	**Tue 10/11/22**	**Sun 11/27/22**
	Simulations	47 days	Tue 10/11/22	Sun 11/27/22
	Schematics	47 days	Tue 10/11/22	Sun 11/27/22
	Software Design: Phase 2	**47 days**	**Tue 10/11/22**	**Sun 11/27/22**
	Modules 1...n	**47 days**	**Tue 10/11/22**	**Sun 11/27/22**
	Code (working subsystems)	47 days	Tue 10/11/22	Sun 11/27/22
	System integration Behavior Models	47 days	Tue 10/11/22	Sun 11/27/22
	Parts Lists	**47 days**	**Tue 10/11/22**	**Sun 11/27/22**
	Parts list(s) for Schematics	47 days	Tue 10/11/22	Sun 11/27/22
	Materials Budget list	47 days	Tue 10/11/22	Sun 11/27/22
	Proposed Implementation Gantt Chart	47 days	Tue 10/11/22	Sun 11/27/22
	Conclusions and Recommendations	47 days	Tue 10/11/22	Sun 11/27/22
	Parts Request Form for Subsystems	32 days	Wed 9/21/22	Sun 10/23/22
	Subsystems Demonstrations Day 1	0 days	Wed 11/9/22	Wed 11/9/22
	Subsystems Demonstrations Day 2	0 days	Wed 11/16/22	Wed 11/16/22
	Parts Request Form for Spring Semester	0 days	Fri 12/2/22	Fri 12/2/22

Figure 32: Gantt Chart for SDP1 (Fall 2022)

		SDP2 Implementation 2023	**453 days**	**Mon 1/9/23**	**Fri 4/5/24**	
		Revise Gantt Chart	14 days	**Mon 1/9/23**	Sun 1/22/23	
		Implement Project Design	**92 days?**	**Mon 1/9/23**	**Mon 4/10/23**	
		Hardware Implementation	**43 days?**	**Mon 1/9/23**	**Tue 2/21/23**	
		Breadboard Components	**21 days?**	**Mon 1/9/23**	**Sun 1/29/23**	
		Sensors	**21 days?**	**Mon 1/9/23**	**Sun 1/29/23**	**Jackson, Nathan**
		Hall Effect	21 days	**Mon 1/9/23**	Sun 1/29/23	
		Ambient Light	21 days	**Mon 1/9/23**	Sun 1/29/23	
		LIDAR	21 days	**Mon 1/9/23**	Sun 1/29/23	
		DC Buck Converter	**21 days**	**Mon 1/9/23**	**Sun 1/29/23**	**Anthony**
		Converter	21 days	Mon 1/9/23	Sun 1/29/23	
		Op-Amp	21 days	Mon 1/9/23	Sun 1/29/23	
		Motor Controller	**21 days?**	**Mon 1/9/23**	**Sun 1/29/23**	**Kyle**
		IC Chip and Circuit	21 days	Mon 1/9/23	Sun 1/29/23	
		Connection to Motors	21 days	Mon 1/9/23	Sun 1/29/23	
		Lights	**21 days?**	**Mon 1/9/23**	**Sun 1/29/23**	**Jackson, Anthony, Kyle**
		Headlights/Taillights	21 days	Mon 1/9/23	Sun 1/29/23	
		Strip Lights	21 days	Mon 1/9/23	Sun 1/29/23	
		Layout and Generate PCB(s)	**21 days**	**Mon 1/9/23**	**Sun 1/29/23**	
		Sensor Module (All Sensors in One)	21 days	Mon 1/9/23	Sun 1/29/23	Jackson
		Power Module (Battery Connection and Voltage Regulator)	21 days	Mon 1/9/23	Sun 1/29/23	Anthony
		Motor Control Module (MCU and Motor Connections)	21 days	Mon 1/9/23	Sun 1/29/23	Kyle
		Lighting Connections	21 days	Mon 1/9/23	Sun 1/29/23	Anthony,Jackson,Kyle
		Assemble Hardware	**14 days**	**Mon 1/30/23**	**Sun 2/12/23**	**23**
		Sensor Module (Connect to MCC)	14 days	Mon 1/30/23	Sun 2/12/23	23 Jackson
		Power Module (Connect to All Systems)	14 days	Mon 1/30/23	Sun 2/12/23	23 Anthony
		MCH (Connect to MCC and Power)	14 days	Mon 1/30/23	Sun 2/12/23	23 Kyle
		Preliminary Assembly	14 days	Mon 1/30/23	Sun 2/12/23	23 Anthony,Jackson,Kyle
		Test Hardware	**7 days**	**Mon 2/13/23**	**Sun 2/19/23**	**28**
		Sensors	7 days	Mon 2/13/23	Sun 2/19/23	28 Jackson
		Power	7 days	Mon 2/13/23	Sun 2/19/23	28 Anthony
		Motor Control	7 days	Mon 2/13/23	Sun 2/19/23	28 Kyle
		Lighting	7 days	Mon 2/13/23	Sun 2/19/23	28 Anthony,Jackson,Kyle

Figure 33: Gantt Chart for SDP2 (Spring 2023)

Note: Figure 33: Gantt Chart for SDP2 (Spring 2023) is continued on page 70.

		Revise Hardware	**7 days**	**Mon 2/20/23**	**Sun 2/26/23**	**33**	
		Sensor Module	7 days	Mon 2/20/23	Sun 2/26/23	33	Jackson
		Power Module	7 days	Mon 2/20/23	Sun 2/26/23	33	Anthony
		Motor Control Module	7 days	Mon 2/20/23	Sun 2/26/23	33	Kyle
		Lights	7 days	Mon 2/20/23	Sun 2/26/23	33	Anthony,Jackson,Kyle
		MIDTERM: Demonstrate Hardware Subs	0 days	Tue 2/21/23	Tue 2/21/23		All
		Software Implementation	**43 days**	**Mon 1/9/23**	**Tue 2/21/23**		
		Develop Software	**21 days**	**Mon 1/9/23**	**Sun 1/29/23**		
		Receiving Inputs from Sensors	21 days	Mon 1/9/23	Sun 1/29/23		Jackson,Nathan
		Ambient Light Calculations	14 days	Mon 1/9/23	Sun 1/22/23		Jackson
		Velocity & Collision Time Calculation	21 days	Mon 1/9/23	Sun 1/29/23		Nathan
		Determining Appropriate Control Signals	21 days	Mon 1/9/23	Sun 1/29/23		Nathan
		Test Software	**14 days**	**Sun 1/29/23**	**Sat 2/11/23**		
		Sensor Inputs	14 days	Sun 1/29/23	Sat 2/11/23		Jackson,Nathan
		Ambient Light Calculation	14 days	Sun 1/29/23	Sat 2/11/23		Jackson
		Velocity and Collision Time	14 days	Sun 1/29/23	Sat 2/11/23		Nathan
		Control Signal Outputs	14 days	Sun 1/29/23	Sat 2/11/23		Nathan,Kyle
		Revise Software	**14 days**	**Mon 1/30/23**	**Sun 2/12/23**	**41**	
		Receiving Inputs from Sensors	14 days	Mon 1/30/23	Sun 2/12/23	41	Jackson,Nathan
		Ambient Light Calculations	14 days	Mon 1/30/23	Sun 2/12/23	41	Jackson
		Velocity & Collision Time Calculation	14 days	Mon 1/30/23	Sun 2/12/23	41	Nathan
		Determining Appropriate Control Signals	14 days	Mon 1/30/23	Sun 2/12/23	41	Nathan
		MIDTERM: Demonstrate Software Subsystems	0 days	Tue 2/21/23	Tue 2/21/23		Nathan
		System Integration	**49 days**	**Tue 2/21/23**	**Mon 4/10/23**		
		Assemble Complete System Integration	**14 days**	**Tue 2/21/23**	**Mon 3/6/23**	**56**	
		Processor and Sensors	14 days	Tue 2/21/23	Mon 3/6/23	56	Nathan,Jackson
		Power	14 days	Tue 2/21/23	Mon 3/6/23	56	Anthony
		Motor Control Unit	14 days	Tue 2/21/23	Mon 3/6/23	56	Kyle
		Lights	14 days	Tue 2/21/23	Mon 3/6/23	56	All
		Full Assembly (Prototype)	14 days	Tue 2/21/23	Mon 3/6/23	56	All
		Test Complete System Integration	**7 days**	**Tue 3/7/23**	**Mon 3/13/23**	**58**	
		Inputs and Outputs of Processor	7 days	Tue 3/7/23	Mon 3/13/23	58	Jackson,Nathan
		Power Flow	7 days	Tue 3/7/23	Mon 3/13/23	58	Anthony,Nathan
		Motor Control (Directional and Speed)	7 days	Tue 3/7/23	Mon 3/13/23	58	Kyle,Nathan
		Complete System Test	7 days	Tue 3/7/23	Mon 3/13/23	58	All
		Revise Complete System Integration	**24 days**	**Tue 3/14/23**	**Thu 4/6/23**	**64**	
		Processor and Sensors	24 days	Tue 3/14/23	Thu 4/6/23	64	Jackson,Nathan
		Power	24 days	Tue 3/14/23	Thu 4/6/23	64	Anthony
		Motor Control Unit	24 days	Tue 3/14/23	Thu 4/6/23	64	Kyle
		Lights	24 days	Tue 3/14/23	Thu 4/6/23	64	Anthony,Jackson,Kyle
		Final Prototype	24 days	Tue 3/14/23	Thu 4/6/23	64	All
		Preliminary Demonstration of Complete	4 days	Fri 4/7/23	Mon 4/10/23	74	All
		Develop Final Report	**106 days**	**Mon 1/9/23**	**Mon 4/24/23**		
		Write Final Report	106 days	Mon 1/9/23	Mon 4/24/23		All
		Submit Final Report	0 days	Mon 4/24/23	Mon 4/24/23	77	All
		Spring Recess	7 days	Mon 3/20/23	Sun 3/26/23		All
		Project Demonstration and Presentation	0 days	Mon 4/17/23	Mon 4/17/23		All

9. Design Team Information

Nathan Keenan, Software Manager, CE

Jackson Piper, Team Lead, Project Manager, EE

Anthony Meniru, Hardware Manager, EE

Kyle P. Law, Engineering Data Manager, EE

10. Conclusion and Recommendations

The project has successfully yielded viable safety systems and functional improvements for ride-on toy vehicles. This semester highly focused on design implementation and troubleshooting based on the previous semester's analysis, research and initial design. The goals to implement, motor control, collision detection and automatic lights have been mixed result however. Motor control has been fully realized with an analog input to control speed based off the input of the electric pedal and a direction control switch into the PIC24. Collision detection using 4 ultrasonic sensors (2 in front and 2 in the rear) has been partially successful.
Based on the developed code, only two of the four ultrasonic sensors work at any given time, due to limitations of the PIC24's sampling and response time. In the case of lighting, issues have arisen and have not been amended with the I2C used by the ambient light sensor and the PIC24. Whilst the code for the ambient light works on its own as a separate subsystem, when tied together with the rest of the code, errors pop up with no indication of what is causing such issues. Given more time perhaps the issues could be resolved but, that is not a luxury that was given. [K.L.]

11. References

Eidehall, A. & Pohl, J. (2012). *Method and system for collision avoidance* (U.S. Patent No. 8,112,225 B2). U.S. Patent and Trademark Office. https://patents.google.com/patent/US8112225B2/en [JP]

Jansson, J., Johansson, J., & Gustafsson, F. (2002). Decision Making for Collision Avoidance Systems. SAE Transactions, 111, 197–204. http://www.jstor.org/stable/44699413 [SA]

Juds, S. (1995). *Collision avoidance system for vehicles* (U.S. Patent No. 5,463,384). U.S. Patent and Trademark Office. https://patents.google.com/patent/US5463384 [JP]

Kelley, W. (1988). *Collision predicting and avoidance device for moving vehicles* (U.S. Patent No. 4,926,171). U.S. Patent and Trademark Office. https://patents.google.com/patent/US4926171A/en [JP]

Meng, F., Gray, R., Ho, C., Ahtamad, M., & Spence, C. (2015). Dynamic vibrotactile signals for forward collision avoidance warning systems. Human factors, 57(2), 329–346. https://doi.org/10.1177/0018720814542651 [SA]

National Semiconductor. (1999, July). LM341/LM78MXX Series 3-Terminal Positive Voltage Regulators. LM78M05C datasheet. Retrieved April 22, 2023, from https://datasheetspdf.com/pdf-file/1047748/NationalSemiconductor/LM78M05C/1 [AM]

Nhapulo, S. L., & de Almeida, J. S. (2021). Modeling electrochemical properties of LiMn1 -xCoxBO3 for cathode materials in lithium-ion rechargeable batteries. Scientific Reports, 11(1), 1–7. https://doi.org/10.1038/s41598-021-90317-0 [AM]

Ramavath, J. N., Ramachandra, C., & Ramanujam, K. (2018). Iron-Dicyano Dichloro Quinone Primary Battery. ChemistrySelect, 3(37), 10281–10286. https://doi.org/10.1002/slct.201801878 [AM]

Reile, P. & Bienz, B. (2004). *Children's ride-on vehicle with electronic speed control* (U.S. Patent No. 6,771,034 B2). U.S. Patent and Trademark Office. https://patents.google.com/patent/US6771034B2/en [JP]

Seiler, P., Song, B., & Hedrick, J. K. (1998). Development of a Collision Avoidance System. SAE Transactions, 107, 1334–1340. http://www.jstor.org/stable/44741070 [SA]

Sinclair, S. (2021, May 14). *Guide to Cars with Advanced Safety Systems*. Consumer Reports. https://www.consumerreports.org/car-safety/cars-with-advanced-safety-systems-a7292621135/ [JP]

Spivey Law Firm. (2016, Jan 18). *Ride on toys Responsible for Almost Half of Toy-Related Injuries*. https://www.spiveylaw.com/blog/ride-on-toys-responsible-for-almost-half-of-toy-related-injuries/ [KL]

Texas Instruments. (2019, Aug). *Introduction to Time-of-Flight Long Range Proximity and Distance Sensor System Design* [White paper]. https://www.ti.com/lit/sbau305 [JP]

Volvo Car Corp. (2022). *Car Safety Innovation*. https://www.volvocars.com/us/about-volvo/human-innovation/future-of-driving/safety [JP]

Zhao, D., Chu, L., Xu, N., Sun, C., & Xu, Y. (2018). Development of a Cooperative Braking System for Front-Wheel Drive Electric Vehicles. Energies (19961073), 11(2), 378. https://doi.org/10.3390/en11020378 [AM]

12. Appendix

Complete Derivation of a Buck DC-DC Converter:

When switch is on, and diode is off: $0 \leq t \leq DT$ using KVL:

$$V_S - V_{L,closed} - V_O = 0$$

$$V_{L\_closed} = V_S - V_O$$

When switch is off, and diode is on at $DT \leq t \leq T$ (using KVL):

$$V_{L,open} + V_O = 0$$

$$V_{L,open} = -V_O$$

Integration:

$$V_{L,avg} = \frac{1}{T}\left(\int_0^{DT} V_{L,closed}\, dt + \int_{DT}^{T} V_{L,open}\, dt\right) = 0$$

$$\frac{1}{T}\left(\int_0^{DT} V_S - V_O\, dt + \int_{DT}^{T} -V_O\, dt\right) = 0$$

$$\frac{1}{T}(V_S - V_O)DT - V_O(T - DT) = 0$$

$$\frac{1}{T}(V_S - V_O)DT - V_O T(1 - D) = 0$$

$$V_S D - V_O D - V_O + V_O D = 0$$

$$V_S D = V_O,\ D = \frac{V_O}{V_S}$$

Complete Derivation of Gain Equation for the Noninverting Operational Amplifier:

Ideal Op Amp Properties:

$$i_+ = i_- = 0 \ \& \ v_+ = v_-$$

KVL at v_+:

$$v_+ = v_I$$

KCL at v_- node:

$$\frac{v_- - 0V}{R_1} + \frac{v_O - v_-}{R_2} = 0$$

$$\frac{v_I}{R_1} + \frac{v_O - v_I}{R_2} = 0$$

$$\frac{v_I}{R_1} + \frac{v_O}{R_2} - \frac{V_I}{R_2} = 0$$

$$\frac{v_O}{R_2} = \frac{-v_I}{R_1} + \frac{v_I}{R_2}$$

$$v_O = v_I\left(1 + \frac{R_2}{R_1}\right)$$

$$\frac{v_O}{v_I} = 1 + \frac{R_2}{R_1}$$

I^2C Functions with Commented Code:

```
void initI2C(int BRG)
{
    T3CON = 0x8000; //initialize Timer3 in 16-bit mode
    I2C1BRG = BRG; //assign value of variable that was passed to
           baudrate generator register
    while(I2C1STATbits.P);
    I2C1CONbits.A10M = 0;
    I2C1CONbits.I2CEN = 1; //enable the I2C module
}

void startI2C(void) //initiate start event
{
    TMR3 = 0;while(TMR3 < 160); //implement 10 us delay
    I2C1CONbits.SEN = 1; //set start enable bit to 1
    while(I2C1CONbits.SEN); //wait until the start event is over
    TMR3 = 0;while(TMR3 < 160); //implement 10 us delay
}

void stopI2C(void) //initiate stop event
{
    TMR3 = 0;while(TMR3 < 160); //implement 10 us delay
    I2C1CONbits.PEN = 1; //set the stop enable bit to 1
    while(I2C1CONbits.PEN); //wait until the stop event is over
    TMR3 = 0;while(TMR3 < 160); //implement 10 us delay
}

void sendbyteI2C(char data) //have microcontroller write a frame
{
    while(I2C1STATbits.TBF); //wait until transmit register is
           available by checking transmit buffer full status bit
    I2C1TRN = data; //assign the argument 'data'
    TMR3 = 0;while(TMR3 < 160); //implement 10 us delay
}

char getbyteI2C(void) //have microcontroller read a frame
{
    I2C1CONbits.RCEN = 1; //enable the receive register
    while(!I2C1STATbits.RBF); //wait until receive register is
           available by checking receive buffer full status bit
    I2C1CONbits.ACKEN = 1; //acknowledge the 'receive'
    TMR3 = 0;while(TMR3 < 160); //implement 10 us delay
    return(I2C1RCV); //return the received value
}
```

```
// Sends master NACK
uint8_t I2Clastgetbyte(void)
{
    I2C1CONbits.RCEN =1; // Set RCEN, Enables I2C Receive mode
    while (!I2C1STATbits.RBF); //wait for byte to shift into I2C1RCV
          register
    I2C1CONbits.ACKEN = 1; // Master sends No acknowledge
    I2C1CONbits.ACKDT = 1;
    return(I2C1RCV);

}
```

sonicS.c

```
#include "mcc_generated_files/system.h"
#include "sonicS.h"

#define PULSE 160
#define CALIBRATION_1_TO_1 464
#define CALIBRATION_1_TO_8 58
#define MAX_16BIT_INTEGER 65535

/*
enum SensorStage
{
  PRELIMINARY,
  AWAITING_ECHO,
  DONE
};
*/

void tenUsDelay(void)
{
  T3CON = 0x8000; //initialize Timer3 in 16-bit mode
  TMR3 = 0;
  while(TMR3 < 160); //implement 10 us delay
}

void setB6(char value) { _RB6 = value; tenUsDelay(); }
void setB4(char value) { _RB4 = value; tenUsDelay(); }
void setB2(char value) { _RB2 = value; tenUsDelay(); }
void setB0(char value) { _RB0 = value; tenUsDelay(); }

char getB7(void) { return _RB7; }
char getB5(void) { return _RB5; }
char getB3(void) { return _RB3; }
```

```
char getB1(void) { return _RB1; }

char bypass = 0;

//long m[4] = {0};

void initSensor(void)
{
   T1CON = 0x8010;
   PR1 = MAX_16BIT_INTEGER;
}

//sends a 10 us pulse to the trigger pin
void sendTrig(SonicSensor sensor)
{
   sensor.setTrigger(1);
   TMR1 = 0;
   while (TMR1 < PULSE);
   sensor.setTrigger(0);
}

void bypassSensorCheck()
{
   bypass = 1;
}

long getEcho(SonicSensor sensor)
{
   long m = 0x00000000;
   TMR1 = 0;
   while (sensor.getEcho() == 0 && !bypass);

   int previousTMR1 = 0;
   TMR1 = 0;
   char overflowed = 0;
   while (sensor.getEcho() == 1 && !bypass)
   {
      if (TMR1 < previousTMR1)
      {
         overflowed = 1;
         break;
      }

      previousTMR1 = TMR1;
   }
   m = TMR1;

   if (bypass)
```

```
  {
    bypass = 0;
    return MAX_16BIT_INTEGER;
  }

  if(!overflowed)
    return m;
  else
    return MAX_16BIT_INTEGER;
}

//calculates the distance in cm
double calcDist(long echo)
{
  return (double)echo/(CALIBRATION_1_TO_8);
}
```

sonicS.h

```
#include "mcc_generated_files/system.h"

#ifndef SONICS_H
#define SONICS_H

typedef struct SonicSensor
{
    void (*setTrigger)(char);
    char (*getEcho)(void);
} SonicSensor;

void setB6(char value);
void setB4(char value);
void setB2(char value);
void setB0(char value);

char getB7(void);
char getB5(void);
char getB3(void);
char getB1(void);

void initSensor(void);

void sendTrig(SonicSensor sensor);

long getEcho(SonicSensor sensor);
//void getAllEchoes(SonicSensor* sensors);
//long returnEchoIndex(int index);
```

```
double calcDist(long);

//double calcHigh(long);

#endif
```

main.c

```
/**
  Section: Included Files
*/
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/tmr1.h"
#include "sonicS.h"
#include "lightSensor.h"
//#include "main_library.h"

#define MAX_16BIT_INTEGER   65535
#define PI                  3.14159265358979

#define SECONDS_60FPS       0.016
#define SECONDS_120FPS      0.008

#define DEBOUNCE_DEPTH      5
#define WARNING_THRESHOLD   5.0
#define COLLISION_THRESHOLD 1.0

#define MAX_ALLOWED_SPEED   7.333333

#define MODERATE_SHADE      300
#define OVERCAST_LIGHT      150

#define WHEEL_DIAMETER      0.25

#define NUMBER_OF_SENSORS   1

#define HUNDRED_PERCENT     800

#define BEEPER              _RA7

#define FORWARD_SIGNAL      _RB9
#define BACKWARD_SIGNAL     _RB8

#define FOG_BEAMS           _RF2
#define HEADLIGHT_BEAMS     _RF3
#define AMBERS              _RF4
```

```
#define BRAKE_LIGHTS        _RF5
#define REVERSE_LIGHTS      _RF6

#define SLOWDOWN_TIME       1.0
#define STOP_TIME           0.5

enum ReverseTransition { SLOWDOWN, STOP, REVERSE };

// set these values to customize program
double frameTime = SECONDS_120FPS;
double beeperTime = 0.5;
double beeperTimeHigh = 0.25;

char numberOfInterrupts = 0;
unsigned long long int timeMillis = 0;
char timerTurnaround = 0;
double wheelDiameter = 1;

unsigned long long int sensorCheckTime = 0;

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

enum ReverseTransition motorTransition = SLOWDOWN;

// initialize variables to be used for the collision detection process
int distances[4] = {0, 0, 0, 0};
int previousDistances[4] = {0, 0, 0, 0};
double timesToCollision[4] = {0, 0, 0, 0};
char warningDebounce[4][DEBOUNCE_DEPTH] = {{0}};
char collisionDebounce[4][DEBOUNCE_DEPTH] = {{0}};
char onWarningCourses[4] = {0, 0, 0, 0};
char onCollisionCourses[4] = {0, 0, 0, 0};
char onWarningCourse = 0;
char onCollisionCourse = 0;

char forward;
char backward;
char lastForward;
char lastBackward;

// initialize variables to be used for speed detection
char hallEffectDebounce[DEBOUNCE_DEPTH] = {0};
char isHallEffect = 0;
char lastIsHallEffect = 0;
unsigned long long int lastTimeHigh = 0;
```

```
double speed = 0;
double lastSpeed = 0;
double acceleration = 0;

// initialize variables to be used as final control determiners
double pedalDutyCycle = 0;
char isTooFast = 0;
char isDark = 0;

// initialize internal control signals
char backwardDebounce[DEBOUNCE_DEPTH] = {0};
char forwardDebounce[DEBOUNCE_DEPTH] = {0};
char triggerForwardToBackward = 0;
char triggerBackwardToForward = 0;
double forwardSpeedMultiplier = 1;
double reverseSpeedMultiplier = 1;
double stopTime = 1;

double beeperTimer = 0;
char triggerWarning = 0;
char triggerStop = 0;
char triggerSlowdown = 0;
unsigned long long int stopTriggerTime = 0;

// main program loop
unsigned long long int lastFrameTime;
double deltaTime = 0;

SonicSensor sensors[4];

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

void initADC(void)
{
    AD1PCFG = 0x3fff;
    AD1CON1 = 0b0000000000000000;
    AD1CON2 = 0b0000000000000000;
    AD1CON3 = 0x0000000111111111;
}

void initTimer2(void)
{
    TRISD = 0xfffe;
    T2CON = 0x8000;
```

```
    PR2 = HUNDRED_PERCENT - 1; // 20kHz, this was at 400 - 1, with 1:1
prescaler
    _T2IF = 0;
    _T2IE = 1;
    OC1R = OC1RS = 4; // duty cycle of 0%
    OC1CON = 0x0006;  // use Timer 2

    OC1RS = HUNDRED_PERCENT;
}

void _ISRFAST _T2Interrupt(void)
{
    ++numberOfInterrupts; // prescale by 20

    if (numberOfInterrupts == 20)
    {
        numberOfInterrupts = 0;
        ++timeMillis;
    }

    if (timeMillis > sensorCheckTime + 50) // 24 ms, calculation based
on maximum operating distance
    {
        bypassSensorCheck();

        PORTA = 0b00000001;
        tenUsDelay();
    }

    if (timeMillis == 0)
        timerTurnaround = 1;

    _T2IF = 0;
}

double getCollisionTime(int previousDistance, int currentDistance,
double frameTime)
{
    if (previousDistance <= currentDistance || frameTime == 0) // this
term here on the right can control noise filtration, with debounce, it
might not be necessary
        return -1;
    else
        return (currentDistance * frameTime) / (previousDistance -
currentDistance);
}
```

```
void pushAndForget(char* array, int size, char value)
{
    for (int i = size - 1; i > 0; i--)
        array[i] = array[i - 1];

    array[0] = value;
}

char evaluateDebounce(char* array, int size)
{
    char containsTrue = 0;
    char containsFalse = 0;

    for (int i = 0; i < size; i++)
    {
        if (array[i])
            containsTrue = 1;
        else
            containsFalse = 1;
    }

    if (containsTrue && !containsFalse)
        return 1;
    else if (!containsTrue && containsFalse)
        return 0;
    else
        return -1;
}

unsigned int getADCValue(char channel)
{
    _CH0SA = channel;
    _ADON = 1;

    _SAMP = 1;
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
    tenUsDelay();
```

```
    _SAMP = 0;

    while(_DONE == 0);

    return ADC1BUF0;
}

char anyTrue(char* array, int size)
{
    for (int i = 0; i < size; i++)
    {
        if(array[i])
            return 1;
    }

    return 0;
}

void setMotorSignal(char forward, double relativeDutyCycle)
{
    if(relativeDutyCycle > 0.99)
        relativeDutyCycle = 0.99;

    double rampedDutyCycle = (relativeDutyCycle) *
(relativeDutyCycle); // ramping function, x^2

    if(forward)
        OC1RS = (int) (HUNDRED_PERCENT * 0.775 - HUNDRED_PERCENT *
rampedDutyCycle * 0.17); // last coeff should be 0.19, 0.775 may be 0
or a little different?
    else
        OC1RS = (int) (HUNDRED_PERCENT * 0.775 + HUNDRED_PERCENT *
rampedDutyCycle * 0.17);

    tenUsDelay();
}

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

// USE THESE TO BREAK UP THE MAIN FUNCTION

void initMain()
{
    // initialize variables to determine desired motor direction
    forward = FORWARD_SIGNAL;
```

```
    backward = BACKWARD_SIGNAL;
    lastForward = FORWARD_SIGNAL;
    lastBackward = BACKWARD_SIGNAL;

    lastFrameTime = timeMillis;

    /*
    sensors[0].setTrigger = setB0;
    sensors[0].getEcho = getB1;

    sensors[1].setTrigger = setB2;
    sensors[1].getEcho = getB3;

    sensors[2].setTrigger = setB4;
    sensors[2].getEcho = getB5;

    sensors[3].setTrigger = setB6;
    sensors[3].getEcho = getB7;
    */

    sensors[0].setTrigger = setB0; // front
    sensors[0].getEcho = getB1;

    sensors[1].setTrigger = setB4; // back
    sensors[1].getEcho = getB5;

    sensors[2].setTrigger = setB2;
    sensors[2].getEcho = getB3;

    sensors[3].setTrigger = setB6;
    sensors[3].getEcho = getB7;
}

void checkSonicSensors()
{
    // get sensor input
    for (int i = 0; i < NUMBER_OF_SENSORS; i++)
    {
        sensorCheckTime = timeMillis;
        sendTrig(sensors[i]);
        distances[i] = (int) (calcDist(getEcho(sensors[i])) * 10);
    }

    /*
    sensorCheckTime = timeMillis;
    getAllEchoes(sensors);
```

```
    for (int i = 0; i < 4; i++)
            distances[i] = (int) (calcDist(returnEchoIndex(i)) * 10);
    */

    // calculate collision time
    for (int i = 0; i < NUMBER_OF_SENSORS; i++)
    {
        timesToCollision[i] = getCollisionTime(previousDistances[i],
distances[i], deltaTime); // with prescaler we at 1 MHz
        previousDistances[i] = distances[i];
    }

    // check for warning and collision conditions
    for (int i = 0; i < NUMBER_OF_SENSORS; i++)
    {
        if (timesToCollision[i] >= 0 && timesToCollision[i] <
COLLISION_THRESHOLD)
        {
            pushAndForget(warningDebounce[i], DEBOUNCE_DEPTH, 0);
            pushAndForget(collisionDebounce[i], DEBOUNCE_DEPTH, 1);
        }
        else if (timesToCollision[i] >= 0 && timesToCollision[i] <
WARNING_THRESHOLD)
        {
            pushAndForget(warningDebounce[i], DEBOUNCE_DEPTH, 1);
            pushAndForget(collisionDebounce[i], DEBOUNCE_DEPTH, 0);
        }
        else
        {
            pushAndForget(warningDebounce[i], DEBOUNCE_DEPTH, 0);
            pushAndForget(collisionDebounce[i], DEBOUNCE_DEPTH, 0);
        }

        if(evaluateDebounce(warningDebounce[i], DEBOUNCE_DEPTH) != -1)
            onWarningCourses[i] = evaluateDebounce(warningDebounce[i],
DEBOUNCE_DEPTH);
        if(evaluateDebounce(collisionDebounce[i], DEBOUNCE_DEPTH) != -
1)
            onCollisionCourses[i] =
evaluateDebounce(collisionDebounce[i], DEBOUNCE_DEPTH);
    }

    // make final determination on any warning or collision course
    onWarningCourse = anyTrue(onWarningCourses, 4);
    onCollisionCourse = anyTrue(onCollisionCourses, 4);
}
```

```
void checkHallEffect()
{
    // evaluate speed to determine if we are going too fast
    unsigned int hallBuffer = getADCValue(15);
    pushAndForget(hallEffectDebounce, DEBOUNCE_DEPTH, hallBuffer <
0x3ff);
    if(evaluateDebounce(hallEffectDebounce, DEBOUNCE_DEPTH) != -1)
        isHallEffect = evaluateDebounce(hallEffectDebounce,
DEBOUNCE_DEPTH);
    if(isHallEffect && !lastIsHallEffect)
    {
        speed = (WHEEL_DIAMETER * PI) / ((timeMillis - lastTimeHigh) /
1000.0);
        acceleration = speed - lastSpeed;
        lastSpeed = speed;

        if (speed > MAX_ALLOWED_SPEED)
            isTooFast = 1;
        else
            isTooFast = 0;
        lastTimeHigh = timeMillis;
    }
    lastIsHallEffect = isHallEffect;
}

void setLights()
{
    // get ambient light level
    uint16_t lux = 0;// getAmbientLightLevel();

    if (lux < OVERCAST_LIGHT)
    {
        TRISFbits.TRISF3 = 0; //set RF2 as an output to trigger fog
beams, P52
        HEADLIGHT_BEAMS = 0;
        tenUsDelay();

        TRISFbits.TRISF2 = 0; //set RF2 as an output to trigger fog
beams, P52
        FOG_BEAMS = 1;
        tenUsDelay();
    }
    else if (lux < MODERATE_SHADE)
    {
```

```
        TRISFbits.TRISF3 = 0; //set RF2 as an output to trigger fog
beams, P52
        HEADLIGHT_BEAMS = 1;
        tenUsDelay();

        TRISFbits.TRISF2 = 0; //set RF2 as an output to trigger fog
beams, P52
        FOG_BEAMS = 0;
        tenUsDelay();
    }
    else
    {
        TRISFbits.TRISF3 = 0; //set RF2 as an output to trigger fog
beams, P52
        HEADLIGHT_BEAMS = 0;
        tenUsDelay();

        TRISFbits.TRISF2 = 0; //set RF2 as an output to trigger fog
beams, P52
        FOG_BEAMS = 0;
        tenUsDelay();
    }

    TRISFbits.TRISF4 = 0; //set RF4 as an output to trigger amber
running lights, P49
    AMBERS = 1;
    tenUsDelay();
}

void checkPedal()
{
    // calculate user duty cycle based on pedal
    long buffer = getADCValue(14);
    double dutyCycle = (double) (buffer - 288) / (1023 - 288); //
range was 0x0114 to 0x03ff
    if (dutyCycle < 0.0)
        dutyCycle = 0.0;
    if (dutyCycle > 1.0)
        dutyCycle = 1.0;
    pedalDutyCycle = dutyCycle;
}

void setMotorDirection()
{
    // determine motor direction
    pushAndForget(forwardDebounce, DEBOUNCE_DEPTH, FORWARD_SIGNAL);
```

```
    pushAndForget(backwardDebounce, DEBOUNCE_DEPTH, BACKWARD_SIGNAL);

    if(evaluateDebounce(forwardDebounce, DEBOUNCE_DEPTH) != -1)
        forward = evaluateDebounce(forwardDebounce, DEBOUNCE_DEPTH);

    if(evaluateDebounce(backwardDebounce, DEBOUNCE_DEPTH) != -1)
        backward = evaluateDebounce(backwardDebounce, DEBOUNCE_DEPTH);

    if (forward && !lastForward)
    {
        if (!triggerForwardToBackward)
        {
            triggerBackwardToForward = 1;
            forwardSpeedMultiplier = 0;
            reverseSpeedMultiplier = 1;
            stopTime = STOP_TIME;
            motorTransition = SLOWDOWN;
        }
        else
        {
            triggerForwardToBackward = 0;

            triggerBackwardToForward = 1;
            if (motorTransition == SLOWDOWN)
            {
                motorTransition = REVERSE;
                stopTime = 0;
            }
            else if (motorTransition == STOP)
            {
                stopTime = STOP_TIME / 2;
            }
            else
            {
                motorTransition = SLOWDOWN;
                stopTime = STOP_TIME;
            }
        }
    }
    if (backward && !lastBackward)
    {
        if (!triggerBackwardToForward)
        {
            triggerForwardToBackward = 1;
            forwardSpeedMultiplier = 1;
            reverseSpeedMultiplier = 0;
```

```
            stopTime = STOP_TIME;
            motorTransition = SLOWDOWN;
        }
        else
        {
            triggerBackwardToForward = 0;

            triggerForwardToBackward = 1;
            if (motorTransition == SLOWDOWN)
            {
                motorTransition = REVERSE;
                stopTime = 0;
            }
            else if (motorTransition == STOP)
            {
                stopTime = STOP_TIME / 2;
            }
            else
            {
                motorTransition = SLOWDOWN;
                stopTime = STOP_TIME;
            }
        }
    }

    lastForward = forward;
    lastBackward = backward;
}

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

// TODO take out switch latches
/*
                         Main application
*/
int main(void)
{
    // initialization
    SYSTEM_Initialize();
    initMain();
    initSensor();
    initTimer2();
    initADC();
    initI2C(0x9D); //baud rate generator value set to 157
    //initLightSensor();
```

```
    TRISA = 0x0000;
    TRISB = 0xC3AA;
    TRISF = 0x00;

    TRISFbits.TRISF2 = 0; //set RF2 as an output to trigger fog beams,
P52
    FOG_BEAMS = 0; //use RF2 as triggering signal, initially low
    tenUsDelay();

    TRISFbits.TRISF3 = 0; //set RF3 as an output to trigger headlight
beams, P51
    HEADLIGHT_BEAMS = 0; //use RF3 as triggering signal, initially low
    tenUsDelay();

    TRISFbits.TRISF4 = 0; //set RF4 as an output to trigger amber
running lights, P49
    AMBERS = 0; //use RF4 as triggering signal, initially low
    tenUsDelay();

    TRISFbits.TRISF5 = 0; //set RF5 as an output to trigger red brake
lights, P50
    BRAKE_LIGHTS = 0; //use RF5 as triggering signal, initially low
    tenUsDelay();

    TRISFbits.TRISF6 = 0; //set RF6 as an output to trigger reverse
lights, P55
    REVERSE_LIGHTS = 0; //use RF6 as triggering signal, initially low
    tenUsDelay();

    TRISDbits.TRISD6 = 1; //set RD6 as button S3 to toggle headlight
beams, active low
    TRISDbits.TRISD7 = 1; //set RD7 as button S6 to toggle fog beams,
active low

    setMotorSignal(1, 0);

    while (1)
    {
        checkSonicSensors();
        checkHallEffect();

        if (speed < 0.25)
            PORTA = 0x00;
        else if (speed < 0.5)
            PORTA = 0x01;
```

```
else if (speed < 0.75)
    PORTA = 0x03;
else if (speed < 0.1)
    PORTA = 0x07;
else if (speed < 1.25)
    PORTA = 0x0f;
else if (speed < 1.5)
    PORTA = 0x1f;
else if (speed < 1.75)
    PORTA = 0x3f;
else if (speed < 2.00)
    PORTA = 0x7f;
else if (speed < 2.25)
    PORTA = 0xff;

//setLights();
AMBERS = 1;
tenUsDelay();


checkPedal();
setMotorDirection();

char beeperVal = 0;
if (onWarningCourse)
{
    beeperTimer -= frameTime;

    if (beeperTimer < 0)
        beeperTimer = beeperTime;

    if (beeperTimer < beeperTimeHigh)
        beeperVal = 1;
    else
        beeperVal = 0;
}

BEEPER = beeperVal;
tenUsDelay();

if (onCollisionCourse)
{
    triggerStop = 1;
    stopTime = 1;
}
```

```
        if (isTooFast)
            triggerSlowdown = 1;


        // TODO A COMPLETE REWORK OF THE SLOWDOWN SYSTEM

        // determine final motor control signal
        // need either a direction control, or need to determine which
direction we are moving
        if (triggerStop)
        {
            // WARNING: THIS CODE SEEMS TO HAVE CAUSED A SHORT IN THE
MOTOR CHIP
            setMotorSignal(1, 0);

            BRAKE_LIGHTS = 1;
            tenUsDelay();
            stopTime -= deltaTime;

            if(stopTime <= 0) // if we are stopped for one second
            {
                triggerStop = 0;

                BRAKE_LIGHTS = 1;
                tenUsDelay();

                if (forward)
                {
                    triggerBackwardToForward = 1;
                    forwardSpeedMultiplier = 0;
                }
                else
                    triggerForwardToBackward = 1;
                    reverseSpeedMultiplier = 0;

                motorTransition = REVERSE;
            }
        }
        else if (triggerSlowdown)
        {
            /*
            OUT_A1 = 1; // forward motion
            OUT_A2 = 0;
            OUT_B1 = 1;
            OUT_B2 = 0;
```

```
            OC1RS = 0; // stop forward motion

            if(speed < MAX_ALLOWED_SPEED || timeMillis - lastTimeHigh
> 1000 || acceleration > 0)
                triggerSlowdown = 0;
            */
        }
        else if (triggerBackwardToForward)
        {
            switch (motorTransition)
            {
            case SLOWDOWN:
                if (reverseSpeedMultiplier > 0)
                    reverseSpeedMultiplier -= deltaTime /
SLOWDOWN_TIME;
                else
                {
                    reverseSpeedMultiplier = 0;
                    motorTransition = STOP;
                }

                setMotorSignal(0, pedalDutyCycle *
reverseSpeedMultiplier);
                break;

            case STOP:
                if (stopTime > 0)
                {
                    stopTime -= deltaTime;
                    setMotorSignal(1, 0);
                }
                else
                    motorTransition = REVERSE;
                break;

            case REVERSE:
                if (forwardSpeedMultiplier < 1)
                    forwardSpeedMultiplier += deltaTime /
SLOWDOWN_TIME;
                else
                {
                    forwardSpeedMultiplier = 1;
                    motorTransition = SLOWDOWN;
                    triggerBackwardToForward = 0;
                }
```

```
                setMotorSignal(1, pedalDutyCycle *
forwardSpeedMultiplier);
                break;
            }
        }
        else if (triggerForwardToBackward)
        {
            switch (motorTransition)
            {
            case SLOWDOWN:
                if (forwardSpeedMultiplier > 0)
                    forwardSpeedMultiplier -= deltaTime /
SLOWDOWN_TIME;
                else
                {
                    forwardSpeedMultiplier = 0;
                    motorTransition = STOP;
                }

                setMotorSignal(1, pedalDutyCycle *
forwardSpeedMultiplier);
                break;

            case STOP:
                if (stopTime > 0)
                {
                    stopTime -= deltaTime;
                    setMotorSignal(0, 0);
                }
                else
                    motorTransition = REVERSE;
                break;

            case REVERSE:
                if (reverseSpeedMultiplier < 1)
                    reverseSpeedMultiplier += deltaTime /
SLOWDOWN_TIME;
                else
                {
                    reverseSpeedMultiplier = 1;
                    motorTransition = SLOWDOWN;
                    triggerForwardToBackward = 0;
                }

                setMotorSignal(0, pedalDutyCycle *
reverseSpeedMultiplier);
```

```
                break;
            }
        }
        else
        {
            BRAKE_LIGHTS = 0;
            tenUsDelay();

            if (forward)
                setMotorSignal(1, pedalDutyCycle);
            else if (backward)
                setMotorSignal(0, pedalDutyCycle);
        }

        // make sure tris is set before each operation and a nop is
called, follow this pattern
        /*
        TRISA = 0;
        Nop();
        PORTA = 0b00000001;
        */

        // wait
        while((deltaTime = (timeMillis - lastFrameTime) / 1000.0) <
frameTime);
        lastFrameTime = timeMillis;
    }

    return 1;
}

/**
End of File
*/
```