Vehicular Motion Sensor: Developing a Wide-Range Motion Sensing Alarm System

Joshua Isaacson

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Fall 2013

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial
fulfillment of the requirements for graduation from the
Honors Program of Liberty University.

_____
Feng Wang, Ph.D.
Thesis Chair

_____
Carl Pettiford, Ph.D.
Committee Member

_____
Monty Kester, Ph.D.
Committee Member

_____
James H. Nutter, D.A.
Honors Director

_____
Date

Abstract

Vehicular safety has become a serious concern in recent years. Many drivers have difficulty backing out of parking spaces, especially when large trucks and sport utility vehicles block the peripheral view from smaller vehicles. This can lead to accidents and pedestrian injuries when drivers are unable to see or do not pay careful attention to their surroundings.

Vehicular motion sensor systems can alert drivers of approaching obstacles when attempting to back out of parking spaces with limited visibility. This thesis aims to explain the limitations of current systems and the research conducted by the author to develop a prototype for an improved system based on ultrasonic and infrared motion sensing technologies that provides earlier obstacle detection over a larger area.

Vehicular Motion Sensor: Developing a Wide-Range Motion Sensing Alarm System

Vehicular motion sensor alarm systems have become increasingly popular accessories for cars because of the difficulty that many drivers have when backing out of parking spaces. Many injuries and accidents occur when drivers are in a hurry and do not pay proper attention when maneuvering around parking lots and do not see people or vehicles behind them. This is especially true when there are children in the area because they are difficult to see behind or around a vehicle. On average, over 50 children in the United States alone are backed over by vehicles each week, and at least two die from their injuries (Backovers, 2013). This problem is worsened when people own large sport utility vehicles, trucks, and vans. There are large blind spots directly behind these vehicles, and they block the peripheral view of other drivers with smaller vehicles that are lower to the ground. Figure 1 illustrates the significant size of some blind spots behind large vehicles, such as the Chevrolet Suburban pictured (Backovers, 2013).

Vehicular motion sensor alarm systems function as aides to drivers by warning them of approaching obstacles while attempting to back out of a parking space. These systems typically include multiple sensors attached to the rear bumper or trunk of the vehicle that detect objects within a certain range (HY Technologies, n.d.). Usually, these motion sensors are only able to detect obstacles directly behind the vehicle without any ability to sense obstacles to the rear peripherals of the vehicle due to restricted detection angles of the sensors or the location of their placement on the vehicle. When the sensors detect an obstacle, a device inside the vehicle, often mounted to the rearview mirror or dashboard, alerts the driver to the presence and distance of an obstacle. The speed at

which the alert is given once an obstacle is sensed is critical in order to give the driver as much time as possible to stop before hitting the obstacle.

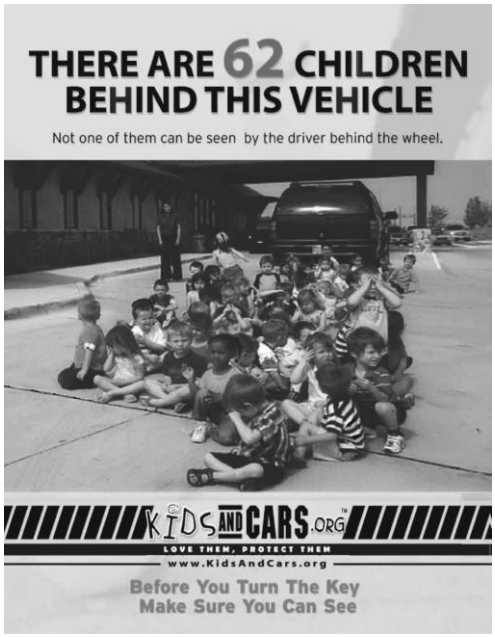


Figure 1. Poster Illustrating the Significant Size of Blind Spots Behind Large Vehicles (Backovers, 2013).

Current Research

Vehicular motion sensor alarm systems consist of two main parts: a collection of sensors that monitor the surrounding area to detect obstacles in the vehicle's path, and a user device that alerts the driver when an obstacle is present. Two commonly used sensors in backup alarm systems available on the commercial market are ultrasonic sensors and infrared motion sensors. Ultrasonic sensors emit and receive high-frequency sound waves in order to determine the distance to an objects (VEX Robotics, n.d.). The distance from the sensor to the object is determined by the length of time it takes for the sound waves to bounce off the object and return to the sensor. This distance is equal to the speed of sound, or 344.2 meters per second, multiplied by one-half the round trip delay of the sound waves (VEX Robotics, n.d.). This research employed the VEX

Ultrasonic Range Finder, which was chosen because of its simple user interface that made it easier to develop an initial prototype. The typical detection range for the VEX Ultrasonic Range Finder, is between three centimeters and three meters (VEX Robotics, n.d.). This research also included a passive infrared motion sensor, which was also selected because of its simple user interface and its ability to detect motion within a larger area than an ultrasonic sensor. This sensor offers a slightly longer detection distance than the ultrasonic sensors of up to five meters and a detection angle of approximately sixty degrees (*35050 PIR motion sensor*, n.d.). Because of the limited detection area covered by ultrasonic and infrared motion sensors, multiple sensor systems are common in order to provide accurate detection within a large enough area behind the vehicle for the system to be a worthwhile purchase for a customer.

The second component of vehicular motion sensor alarm systems is a device that alerts the driver whenever the motion sensors signal that an obstacle has been detected. The device monitors the output from the various sensors and determines whether or not an obstacle was detected. If one was detected, the device alerts the driver of a nearby obstacle and may calculate the distance to that obstacle. A potential problem could arise if false alarms are generated from inaccurate output from the sensors. The device must be programmed in such a way as to limit the frequency of false alarms without eliminating legitimate alarm signals.

Commercially Available Systems

Many systems currently available on the commercial market serve as alarm systems to alert drivers when there are obstacles behind a vehicle. Some systems, such as the 9401T Back Up Sensor from Directed Electronics and the Backup Parking Sensor

System shown in Figure 2 (HY Technologies, n.d.), rely on the detection of ultrasonic sound waves that are emitted from the sensor system and bounce off obstacles behind the vehicle back towards the sensor (Directed Electronics, Inc., 2004). However, this method is inefficient in cases in which the obstacle does not have a flat surface perpendicular to the oncoming ultrasonic waves. In these cases, the ultrasonic waves bounce off the obstacle and away from the sensor, resulting in the obstacle not being detected (Directed Electronics, Inc., 2004). Another issue with these systems is that the detection angle is limited. By placing multiple sensors along the rear bumper, as illustrated in Figure 2, the area directly behind the vehicle is well covered by the detection areas of the sensors. However, because these sensors face directly behind the vehicle and no sensors face outwards from the corners of the bumper at an angle, there is limited detection to the peripheries of the vehicle. This is an issue because visibility concerns in parking lots are not limited to only behind a vehicle, but also to the sides of the vehicle as a car is backing out of a parking space when the driver's view is obstructed by another vehicle or obstacle. Therefore, an improved vehicular motion sensor alarm system is needed.

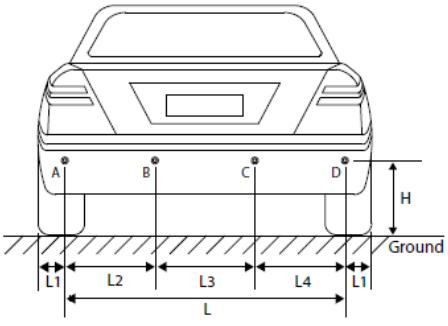


Fig.1 Mounting Height and Position of Sensors
H = 50 ~ 80cm (20" ~ 32")
L1 = 6 ~ 15cm (2" ~ 6")
L2 = L3 = L4 or L2 = L4 = 0.3L, L3 = 0.4L

Figure 2. Sample Mounting Position of Sensors on the Rear Bumper of a Vehicle (HY Technologies, n.d.).

Purpose for Research

This research, completed under the direction of Dr. Feng Wang, Associate Professor of Engineering at the Liberty University School of Engineering and Computational Sciences, focused on developing a prototype for an improved system that can provide earlier obstacle detection over a larger area behind a vehicle as the driver prepares to back out of a parking space. This system will benefit drivers by alerting them to approaching pedestrians, vehicles, and other obstacles outside of their line of vision when backing out of a parking space. The goal was to develop an algorithm that combined two different types of sensor technologies, ultrasonic and infrared, in one system that could not only detect motion, but also determine whether or not the object was approaching the vehicle. The VEX Ultrasonic Range Finder is suitable for calculating the distance to an object based on the time it takes for an ultrasonic wave pulse to travel from the sensor to the object and back to the sensor. Meanwhile, the Passive Infrared Motion Sensor is capable of detection motion across a wider array. By combining these two sensors, the system can detect motion from an object and then calculate the distance to that object once it passes in front of the VEX Ultrasonic Range Finder. By developing a software algorithm to operate these two types of sensors simultaneously, the system will be able to determine whether or not an alert needs to be sent to the user based on the input from the two sensors. This system as a whole will serve as a prototype of a basic implementation of these two types of sensors that can be further developed into a refined, finished system that can be produced and sold on the commercial market.

Systems are currently available on the commercial market, but these systems have several limitations. One limitation is the detection range and area of the sensors. Many sensors can only detect obstacles up to a few meters away and have a limited detection width behind the vehicle (*ReverseGUARD FAQ*, 2013; *Rear object detection systems*, n.d.). This device will attempt to provide a greater detection area, both width around the peripheral of the vehicle and depth behind it, than systems currently available on the commercial market. Another limitation is the process by which the alarm signal is relayed to the driver. Some systems use a series of beeps, with a constant tone meaning that you are very close to the obstacle, to alert the driver of an obstacle and its distance from the vehicle, while others require the driver to attach a device to the rearview mirror or the dashboard to display the distance to the closest obstacle (*Car backup cameras*, 2013). This research focused on implementing a buzzer to alert the driver when an obstacle was detected. This method provides an audible alert whenever an obstacle is detected without requiring a complex user interface. However, both of these limitations can be furthered improved upon through additional research and a final device created from the prototype developed from this research.

VEX Ultrasonic Range Finder

One sensor that this research focused on was the VEX Ultrasonic Range Finder, illustrated in Figure 3, because of its simple user interface that made it easier to develop an initial prototype (VEX Robotics, n.d.). It is also designed to detect and calculate the distance to obstacles in front of it, which is beneficial when determining whether an obstacle is approaching or leaving the vicinity of the vehicle (VEX Robotics, n.d.). Ultrasonic sensors emit and receive high-frequency sound waves in order to determine

the location of objects. The object's distance from the sensor is determined by the length of time it takes for the sound waves to bounce off the object and return to the sensor. This distance is equal to one-half the speed of sound times the round trip delay of the sound wave, or 172.1 meters per second times the round trip delay (VEX Robotics, n.d.). The typical detection range for the VEX Ultrasonic Range Finder is between three centimeters and three meters. Most devices must use multiple sensors in order to detect a large enough area behind the vehicle to be a worthwhile investment for a customer. However, the effectiveness of this type of sensor is limited when the obstacle is not perpendicular to the sensor, causing the ultrasonic waves to bounce off the face of the obstacle at an undesired angle and never return to the sensor.

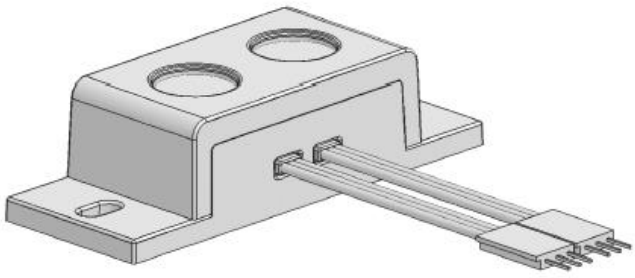


Figure 3. VEX Ultrasonic Sensor (VEX Robotics, n.d.).

To detect an obstacle, the VEX sensor must go through a specific process. First, a 10 microsecond pulse is sent to the sensor to tell the sensor that it needs to send the ultrasonic sound waves used to detect the object (VEX Robotics, n.d.). After the pulse, the sensor outputs a signal of 5 volts, which corresponds to a logical *high* value (VEX Robotics, n.d.). This *high* output signal is maintained until the sensor detects the ultrasonic wave that bounced off an object and returned to the sensor. The sensor then outputs a signal of 0 volts, or a logical *low* value, which is maintained until the next 10

microsecond pulse is generated (VEX Robotics, n.d.). The period of time that the VEX ultrasonic sensor outputs a logical *high* value is equal to the round-trip delay of the sound wave. To determine the distance of the object from the sensor, the round-trip delay must be multiplied by one-half the speed of sound, or 172.1 meters per second. This entire process is illustrated in Figure 4, which depicts the ultrasonic sound waves emitted from the sensor and bouncing off an obstacle before being detected by the sensor (VEX Robotics, n.d.).

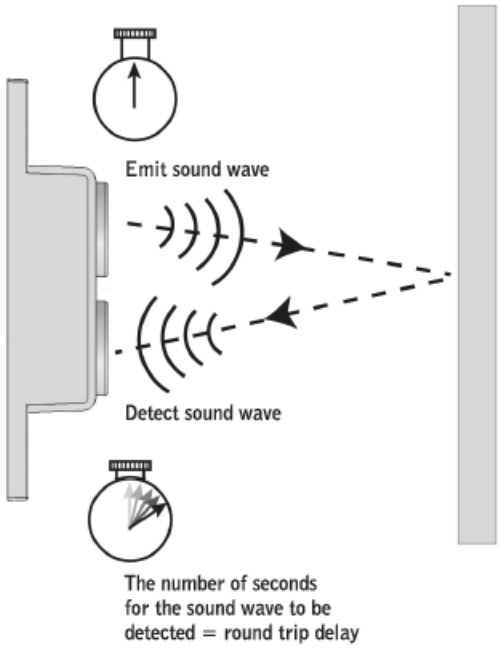


Figure 4. Operation of VEX Ultrasonic Sensor (VEX Robotics, n.d.).

In order to determine whether or not an obstacle has been detected by the VEX Ultrasonic Range Finder, a separate device must read in the output from the sensor and, if detected, calculate the distance to the obstacle. For this sensor, an obstacle is known to be detected if the period of time that the sensor outputs a logical *high* value is less than or equal to the period of time it takes for the ultrasonic waves to travel from the sensor to its

maximum detection point and back again. The VEX sensor has a maximum detection distance of approximately three meters, which would result in a maximum duration of a logical *high* value output from the sensor of approximately 17.4 milliseconds when a legitimate obstacle is detected. The device must be able to factor out times that exceed this threshold, while also calculating the distance to an obstacle for any duration under the threshold. This calculation is performed by a microcontroller that can be programmed to accept multiple sensor inputs and calculate the correct distance and location of the obstacle before outputting that result as an alert to the driver. The current problem with most commercial systems is the frequency of false alarms caused by inaccurate output from the sensors. The microcontroller must be programmed in such a way as to limit the frequency of false alarms without eliminating actual alarms. A Microchip PIC18F242 microcontroller, which can accept input voltage values up to 5 volts, was used for this project because of its simplicity and available functionality (Microchip Technology, 2006). It contains several hardware modules that can generate system interrupts whenever the input value from a sensor changes. Because the VEX Ultrasonic Range Finder outputs value between 0 and 5 volts, no voltage amplifier is needed to scale the output voltage of the VEX sensor to an acceptable input value for the microcontroller. This functionality, in addition to its use in other engineering courses at Liberty University, made the microcontroller easy to use in a system prototype.

The VEX Ultrasonic Range Finder has a simple user interface consisting of six pins: three for the input control and three for the output control. Two pins, one for both the input and output control segments, are connected to ground and two pins are connected to V_{DD}, which equals positive 5 volts power (Ultrasonic range finder, 2012).

The other two pins are control signals that are connected to the microcontroller (Ultrasonic range finder, 2012). The input pin for the ultrasonic sensor accepts the output signal from the microcontroller to begin sensing, while the output pin relays the current state of the sensor to the microcontroller.

Passive Infrared Motion Sensor

Passive infrared motion sensors, such as the one shown in Figure 5, are also frequently used in motion detection systems (*PIR sensor*, 2010). These sensors operate by collecting infrared signals radiated by objects. The passive infrared motion sensor used in this project consists of a pyroelectric sensor which "combines a crystal and a filter to produce an electric charge when exposed to infrared radiation" (*PIR sensor*, 2010, p. 4). However, the sensor is sensitive to a wide array of radiation, so a filter must be used to select the specific range of values that pertain to the radiation emitted from a human body (*PIR sensor*, 2010). This filter allows the system to specifically detect motion from humans. In a completed vehicular motion sensor alarm system marketed commercially, a different passive infrared motion sensor may be necessary to guarantee sensitivity to animals, pedestrians, and vehicles.

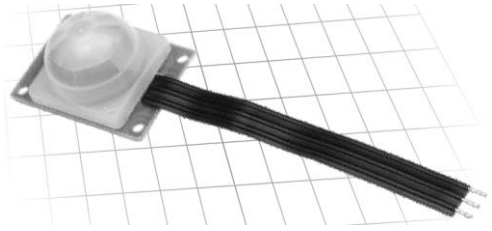


Figure 5. Passive Infrared Motion Sensor (*PIR sensor*, 2010).

Though the sensor has a filter to select a specific range of radiation that is emitted by humans, the sensor can still be influenced by changes in temperature (*PIR sensor*,

2010). To counteract this unwanted interference, the sensor has two sensor regions. "A body passing in front of the sensor will activate first one and then the other element whereas other sources will affect both elements simultaneously" (*PIR sensor*, 2010, p. 4). This helps to eliminate the effect of undesired signals interfering with the device.

The passive infrared motion sensor that was used in this project has simple interface that the user can interact with. There are only three input pins, as seen in the wiring diagram in Figure 6: ground, positive 5 volts power, and the output signal (*PIR sensor*, 2010). This sensor is much simpler than the VEX Ultrasonic Ranger Finder in that it does not require any input signal from the microcontroller to initiate the detection sequence. However, the sensor does require approximately one minute for the device to initialize and become operational after it is connected to the power and ground inputs (*PIR sensor*, 2010). This means that when using this sensor, the entire system must wait for a full minute until the sensor is initialized. This could potentially pose a problem for a system that is turned on once a driver starts his or her vehicle. Drivers do not want to be required to wait an additional minute once they have started their vehicles before being able to use the device. On the other hand, if the sensor is connected to a power source that remains on, even when the vehicle is not running, then the sensor would not have any initialization delay after the first minute when the power source is initially connected. After the sensor is initialized, it outputs a positive 5 volts signal to the microcontroller that lasts for approximately one second whenever motion is detected (*PIR sensor*, 2010).

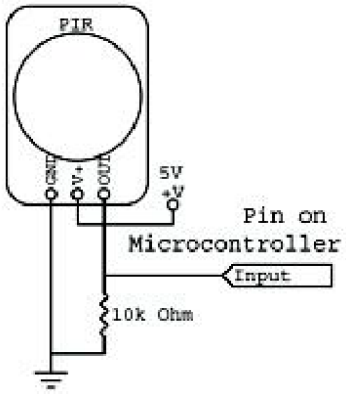


Figure 6. Passive Infrared Motion Sensor Wiring Diagram (*PIR sensor*, 2010).

The sensor has an approximate detection angle of 60 degrees from left edge to right edge and a maximum detection distance of approximately five meters (*PIR sensor*, 2010). The maximum detection distance directly in front of the sensor is approximately five meters; however, the detection range decreases drastically as the angle from the center of the sensor widens to a maximum of 30 degrees, as seen in Figure 7 (*35050 PIR motion sensor*, n.d.). A detection distance of five meters is an acceptable distance to provide drivers enough time to react if an approaching obstacle is detected that far away. However, the detection range on the outer limits of the sensor's maximum angle of detection does not provide much advance notice to the driver for an obstacle approaching from the side because it can only detect up to two and a half meters away. In order to accommodate this weakness, it may be necessary to put the sensor at an angle on the corner of the rear bumper so that it can detect motion at greater distances outside of the driver's line of vision directly behind the vehicle.

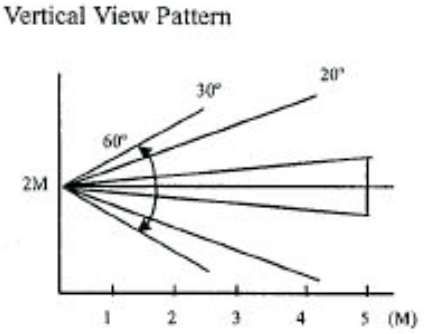


Figure 7. Detection Range of Passive Infrared Motion Sensor (*35050 PIR motion sensor*, n.d.).

System Design

The overall design of the system consists of both hardware and software components. The hardware design of the system includes the main circuit that is implemented to connect three sensors, two passive infrared motion sensors and one VEX Ultrasonic Range Finder, to the microcontroller that stores the software program. The software design includes the C program implementation of an algorithm that operates all three sensors simultaneously and outputs an alert signal to the user whenever motion from an approaching obstacle is detected.

Hardware Design

The hardware design uses three primary components: a PIC18F242 microcontroller, a VEX Ultrasonic Range Finder, and two Passive Infrared Motion Sensors. The microcontroller operates at a frequency dependent upon an external crystal. The frequency of the microcontroller is four times the frequency of the crystal, which has a frequency of 7.3725 MHz, or millions of cycles per second. The microcontroller operates at an input voltage of 5 volts, which is provided from an AC/DC power adapter and stepped down using a 5 volt voltage regulator. The microcontroller also has hardware capability to perform asynchronous serial communication (USART). The

microcontroller has both a transmit (TX) pin, which is located at port RC6, and a receive (RX) pin, which is located at port RC7 (Microchip Technology, 2006). The transmit and receive pins allow the microcontroller to output data to a terminal screen on an external computer. This is important in order to determine whether or not the sensor is functioning properly and the distance calculations are being performed correctly. However, the serial communication on the external computer side must be performed at a greater voltage than the 5 volts that operate the PIC18F242. To solve this problem, a MAXIM 202/RS232 transceiver was used to convert RS232 voltage levels to digital levels and vice-versa. The transceiver stepped up the output voltage from the microcontroller before the signal was sent through a DB9 connector to the external computer and stepped down the voltage when transferring data from the computer to the microcontroller.

The VEX Ultrasonic Range Finder has six pins. Two pins are connected to ground and two are connected to V_{DD}, which equals 5 volts (VEX Robotics, n.d.). The other two pins are control signals that are connected to the microcontroller (VEX Robotics, n.d.). The input pin for the ultrasonic sensor is connected to RB3 of PORTB. The output pin for the ultrasonic sensor is connected to RB4 of PORTB. These ports are configured in software from the perspective of the microcontroller: RB4 is an input from the sensor to the microcontroller and RB3 is an output from the microcontroller to the sensor.

The Passive Infrared Motion Sensor has three pins. The leftmost pin is connected to the ground, the middle pin is connected to V_{DD}, which equals 5 volts, and the rightmost pin is the output signal from the sensor, which is connected to the input pin of the

microcontroller. The PIC18F242 microcontroller has two input ports that form a Capture/Compare/Pulse Width Modulation (CCP) module. These two ports, labeled CCP1 and CCP2, are enabled in the software algorithm to operate in *Capture* mode. This allows the microcontroller to capture a rising or falling edge of the output signal from the Passive Infrared Motion Sensor to determine when motion has been detected. Since two Passive Infrared Motion Sensors are used in this system, one is connected to CCP1 and one is connected to CCP2.

In order to alert the user when motion has been detected, a buzzer is connected to the microcontroller. The buzzer has two pins, one marked with a plus (+) sign that indicates the input pin and one that is connected to ground. The input pin of the buzzer is connected to PORTB of the microcontroller using pin RB0. The microcontroller outputs a high logic signal of 5 volts whenever an alert is generated. This results in an audible alert given by the buzzer. The schematic of the entire hardware design that was created in Eagle can be seen in Appendix A and additional images of the implemented circuit, which is shown in Figure 8, can be seen in Appendix B.

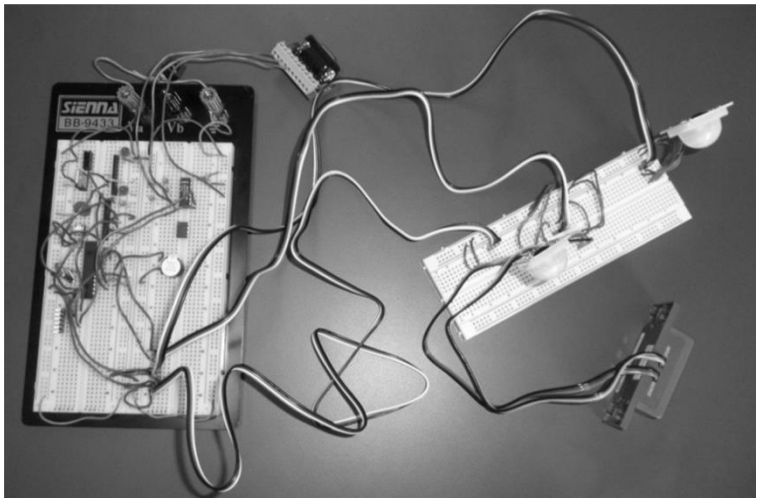


Figure 8. Implemented Circuit on Prototyping Board.

Software Algorithm

The software algorithm developed for the vehicular backup alarm system has three main parts: the Passive Infrared Motion Sensor algorithm, the VEX Ultrasonic Range Finder algorithm, and the event handling and alert algorithm. The two algorithms that operate the three sensors are performed simultaneously while the event handling and alert algorithm is performed at the end of each sensor check. The source code for the entire algorithm can be seen in Appendix C.

Passive infrared motion sensor algorithm. The Passive Infrared Motion Sensor algorithm was relatively simple to implement in software. The microcontroller captures the output signal from the Passive Infrared Motion Sensor using the CCP module. The CCP module will generate an interrupt whenever the value at the input pin on the microcontroller satisfies a certain condition based on the mode it is set to (Microchip Technology, 2006). This algorithm uses two different modes, one to capture the rising edge of the signal and one to capture the falling edge. Because there are two Passive Infrared Motion Sensors, the algorithm uses both the CCP1 and CCP2 input modules. At the beginning of the program, each CCP module is configured to generate an interrupt on the rising edge of the input signal. In order for the module to generate an interrupt based on the given mode, interrupts must be enabled for the CCP modules. This is done by setting CCP1IE and CCP2IE each to '1'. However, interrupts are also used for the VEX Ultrasonic Range Finder. It is possible for interrupts from multiple sensors to occur simultaneously. Because of this, the CCP1 and CCP2 modules are set to high priority interrupts while the PORTB interrupt, which is used for the VEX Ultrasonic Range Finder, is set to low priority because the Passive Infrared Motion Sensors are more

effective in detecting motion. The VEX Ultrasonic Range Finder is limited to calculating the distance to an object directly in front of it, whereas the Passive Infrared Motion Sensors are capable of detecting motion across a wider area, though it cannot determine the distance to the object. The CCP module mode used to generate an interrupt is set to the rising edge of the input signal from the sensor because the Passive Infrared Motion Sensor outputs a 5 volts pulse for approximately one second every time motion is detected. By capturing the rising edge, the system can identify the beginning of every positive detection pulse. When a new pulse is detected, an internal signal flag is set notifying the system that motion has been detected from the corresponding CCP module. At the end of the detection sequence for the three sensors, the system performs an event handling routine that prints out a message to the user stating which sensors detected motion. This whole process is repeated infinitely while the system is powered on.

VEX ultrasonic range finder algorithm. The software algorithm for the VEX Ultrasonic Range Finder was more complex than that of the Passive Infrared Motion Sensor. The main difficulty was determining the most effective way of calculating the distance to an object based on the duration of the high output signal from the microcontroller. Initially, the polling method was used to continually check to see when the microcontroller received a high input value from the sensor and count the number of clock cycles that elapsed until the sensor output a low value. This method was not efficient because the microcontroller could not do anything else while the polling was taking place.

A more efficient method for calculating the distance to an object was to use interrupts. Three types of interrupts were used in the program. The first interrupt that

was used was TMR2. This interrupt is based on a timer that can be modified in the software program to occur after a certain number of clock cycles. TMR2 is an 8-bit timer that has an 8-bit period register (Microchip Technology, 2006). This period register can be set to any 8-bit value and an interrupt will occur whenever the timer overflows past the period register value. For this program, the period register was set to 0x18, which is equal to 24 clock cycles, or approximately a 10 microsecond delay. This is used when sending the 10 microsecond control signal to the sensor to alert it that it needs to emit and prepare to receive the ultrasonic sound waves. The second interrupt that was used was the PORTB change interrupt. This interrupt occurs every time the values of pins RB4 through RB7 change. This is useful because the interrupt will occur on both the rising and falling edges of the pin value instead of just a single edge selected in software. This interrupt was used to monitor the output value from the VEX Ultrasonic Range Finder. When the interrupt occurred and the value of RB4, the input from the sensor, changed from low to high, the timer counting the round trip delay of the ultrasonic waves started. The timer stopped when the interrupt occurred again and the value of RB4 went from high to low. The third interrupt that was used was TMR0. This interrupt, like TMR2, was used to count the number of clock cycles that elapsed. TMR0 can be selected in software as an 8-bit or 16-bit timer, but it does not have a period register (Microchip Technology, 2006). This program selected TMR0 as an 8-bit timer, so the interrupt occurs whenever the value overflows the maximum 8-bit number, 255, and resets to 0. The total round trip delay was calculated as the number of times the TMR0 interrupt occurred times 2^8, or 256, plus the remaining clock cycles from the timer after the final occurrence of the interrupt. The round trip delay was then converted from clock cycles to

microseconds using the frequency of the microprocessor, 29.49 MHz. This round trip delay was used to calculate the distance to the object.

The distance from the sensor to the obstacle is calculated using the formula ½ * speed of sound * round trip delay of the sound wave (VEX Robotics, n.d.). The speed of sound is equal to 344.2 meters per second or 1,135 feet per second (VEX Robotics, n.d.). The round trip delay, initially in microseconds, is converted into seconds and then multiplied by one-half the speed of sound. The resulting value is the distance to the object in feet or meters, depending on which speed of sound value was used.

After an object is detected and a distance is calculated, it must then be compared to distances of several recent calculations. This is the part of the algorithm that is the most difficult. Because the ultrasonic sound waves do not always bounce of the object the same way every time, slight variations in the round trip delay are inevitable. It is important that the algorithm takes this into account in order to limit the number of false alarms that are output to the user. When backing out of a parking space, drivers typically do not exceed speeds of approximately five to ten miles per hour due to limited visibility and small areas to maneuver their vehicles. This speed can be used to determine whether or not the distance to an object is decreasing faster than the speed the vehicle is going in reverse. If the distance to an object is indeed decreasing faster than the distance traveled by the vehicle in reverse based on the arbitrary average speed traveled in reverse, then the object is approaching the vehicle and an alert needs to be sent to the user. If not, then the object is either leaving or stationary. However, if the object is within a threshold distance of the vehicle, an alert must be given to the user to warn them that the car is approaching an obstacle. The algorithm itself is not difficult to understand, but the values that need to

be programmed cannot be determined without a lot of testing. As mentioned earlier, the sensor does not always output the same exact value when the sensor and object are stationary. Because of this, if the program only checks whether or not the current distance is greater than or less than the last distance, then a false alarm may be given when the object is not a threat and the only movement was negligible. So, the best way to solve this problem is to have a threshold difference value to determine whether or not the distance to the object has changed by an amount that is worth alerting the user about. This threshold value is difficult to determine because the program should alert the user as quickly as possible when an approaching obstacle is detected, but the program should limit the frequency of false alarms. The threshold value was set to 0.1176 meters, or 4.4 inches, which is equivalent to the distance a car will travel in 50 milliseconds at a speed of 5.0 miles per hour. This value was chosen as an arbitrary speed for a vehicle backing out of a parking spot. So, it is important to be able to gauge how quickly the distance between the vehicle and the object is changing relative to the vehicle. The elapsed time of 50 milliseconds was selected based on the length of time needed to complete one detection and calculation cycle. Both the speed of the vehicle and the program time can be modified as global variables at the beginning of the source code.

Event handling and alert algorithm. The last part of the program involves handling the different event cases from the input sensor signals and determining whether or not an alert needs to be generated. Because there are three sensors which can each detect an obstacle, there are eight possible combinations of outcomes from the group. There are different event statements printed to the terminal screen via serial communication to alert the user to which case was handled. Any case that involves

motion detected from at least two of the three sensors generated an alert. When an alert is generated, the microcontroller also sends a high output signal to the buzzer to trigger an audible alert. The buzzer signal lasts for approximately one second before it is cleared and the detection cycle is repeated.

Results

The goal of this research project was to construct a working prototype for an improved vehicular backup alarm system based on ultrasonic and infrared motion sensing technologies that provides earlier obstacle detection over a larger area. This prototype implements an algorithm that operates three sensors: two Passive Infrared Motion Sensors and one VEX Ultrasonic Range Finder. When motion is detected by these sensors, an output signal sent to the PIC18F242 microcontroller triggers an interrupt that notifies the microcontroller that motion was detected. Whenever motion is detected from an approaching object, an alert should be generated. In this prototype, a buzzer was used to sound an alarm whenever motion from an approaching object is detected.

The final prototype of the system is programmed using a PICKit2 programmer. The program, which was written using the MPLAB X IDE developed by Microchip, is loaded onto the PIC18F242 microcontroller. The three sensors are placed on a smaller board and connected to the microcontroller located on a separate circuit board using long wires so that the sensors can be placed in different places around the main prototyping circuit board. This increases the flexibility of the prototype design. The microcontroller is also connected to a computer and transmits output data to a terminal window via serial communication. This allows the user to view the output from each of the sensors and any text alerts that are generated. Whenever an alert is triggered, the microcontroller also

outputs a high signal to turn on a buzzer. This audible alert provides another means of notifying the user that motion was detected.

The system functions properly in operating the three sensors and generating alerts based on the eight possible combinations of outputs from the motion sensors. However, there are a few weaknesses of the design that limit the overall effectiveness of the prototype. One weakness is the quality of the sensors that are used. The Passive Infrared Motion Sensors and the VEX Ultrasonic Range Finder are all low quality sensors. The VEX Ultrasonic Range Finder is meant to be used for robotic devices developed by middle and high school students for competitions (VEX Robotics, n.d.). This sensor is not supposed to be used for systems requiring high precision measurements, which is what this application requires in order to accurately calculate the distance to an obstacle and determine whether it is approaching or leaving. The Passive Infrared Motion Sensors are also inadequate for use in the final design. These sensors require a full minute to be initialized before they will begin outputting pulses for motion detection (*PIR sensor*, 2010). Also, the sensors output a pulse for a full second whenever motion is detected (*PIR sensor*, 2010). Because of this, the next detection cycle cannot begin until after the Passive Infrared Motion Sensor has stopped outputting the high motion detection pulse. If the next detection cycle proceeds before the output from the sensor returns to the low position again, then the Passive Infrared Motion Sensor will not be able to detect motion for that cycle. This severely limits the operation of the system as a whole. The higher frequency at which the detection cycles are run, the more accurate the system is in detecting motion and alerting the user. However, because of the long output pulse of the Passive Infrared Motion Sensor, operating the system with quick detection cycles is not a

feasible option. Therefore, an ideal system would utilize higher quality and faster motion sensors.

Another weakness of the system is that the Passive Infrared Motion Sensors can only detect motion while the VEX Ultrasonic Range Finder can only calculate the distance to an obstacle directly in front of it. Because of this, the VEX Ultrasonic Range Finder by itself is not too useful since it could possibly calculate the distance to an object that is not moving, but within range directly in front of the sensor. However, if combined with a Passive Infrared Motion Sensor, then the VEX Ultrasonic Range Finder could calculate the distance to a moving object that is detected by the motion sensor. In this system, due to limited resources and having access to only a single VEX Ultrasonic Range Finder, the VEX sensor was placed in the center of the system, between the two Passive Infrared Motion Sensors. The theory is that this would enable the system to detect motion of an object from one side of the system and, as it crosses into the center of the system, it will be detected by the VEX Ultrasonic Range Finder and its distance will be calculated. However, with this setup, it is difficult to provide accurate results due to the wide detection range of the Passive Infrared Motion Sensors and the narrow detection range of the VEX Ultrasonic Range Finder.

A better solution, given additional resources, would be to group a Passive Infrared Motion Sensor and a VEX Ultrasonic Range Finder in a pair. This pair would be mounted together and face the same direction. When an object was detected by the motion sensor, it should be detected by the VEX Ultrasonic Range Finder shortly thereafter as it crosses across the detection area of the Passive Infrared Motion Sensor. The entire system would be comprised of several of these sensor pairs. Each pair would

be placed in a different location along the rear bumper of the vehicle so that the entire rear peripherals can be detected by the Passive Infrared Motion Sensors. The ideal system would most likely include four pairs: one in each corner of the rear bumper placed at a 45 degree angle with the rear bumper and two along the rear bumper placed approximately one-third of the total bumper distance away from the corner sensors. This would provide a full detection area directly behind the vehicle as well as the rear side peripherals of the vehicle. This is important because the system should detect obstacles prior to them moving directly behind the vehicle so that the driver is given advanced warning of an approaching vehicle or pedestrian.

Further Development

This research is only the beginning of the process of creating an effective vehicular motion sensor alarm system that drivers can attach to rear of their vehicles. As a result, there are several aspects of this project that can be further developed before creating a final system that can be implemented. One aspect of the project that can be further developed is the type of sensors used in the final system design. The VEX Ultrasonic Range Finder and the Passive Infrared Motion Sensor, while acceptable for prototyping a design, are not the best sensors to use in a final product. These sensors are large and bulky and would be an eye sore for a driver if attached to the rear bumper of his or her vehicle. The sensor system should be small in size and not noticeable when attached to the vehicle. Otherwise, customers would not be willing to purchase the system because it would not be an aesthetically appealing addition to their vehicles. Also, the two sensors used in this project are stand-alone units that can be attached to a prototyping breadboard. However, the final design would require mounting these

components to a printed circuit board that would include multiple sensors and other circuit components, not just a single sensor. The Passive Infrared Motion Sensors also have one weakness that is hard to ignore. These sensors require an initialization period of one minute once power and ground are connected before it will function properly. This poses a significant problem because customers will not want to wait for a minute when they turn on the system before they can back out of their parking space. By designing a better sensor, hopefully the initialization delay required will be reduced.

Another aspect of the project that can be further developed is implementing additional sensors and determining the most effective arrangement of sensors across the rear of the vehicle. This project focused on implementing three sensors: two Passive Infrared Motion Sensors and one VEX Ultrasonic Range Finder. These sensors were arranged with the VEX Ultrasonic Range Finder in the center flanked on both sides by the two Passive Infrared Motion Sensors. A future design may seek to encompass additional sensors to better detect motion over a larger area and accurately determine the distance to that obstacle. This could be done by placing the sensors in pairs, with one Passive Infrared Motion Sensor and one VEX Ultrasonic Range Finder, to allow the two different technologies to operate in tandem. This would enable the system to detect motion and the distance to that moving obstacle at each sensor placement.

A third improvement to the project could include refining the algorithm to operate more efficiently. One downside to the current implementation is that there is a one minute delay at the beginning of the program to allow the Passive Infrared Motion Sensors to initialize once the power and ground inputs are connected. A better means of limiting the delay time may involve modifying the program to enter sleep mode or not

have the device shut off completely so that the Passive Infrared Motion Sensors do not have to reinitialize every time the program is executed. This would save a lot of time during both testing and operation, as well as make the process smoother for customers so that it is not necessary to wait a minute before using the device every time they enter their vehicles. Also, the program currently starts all three sensors at the same time, but the VEX Ultrasonic Range Finder completes its pulse and calculation so quickly that it is difficult to detect motion and obtain the pulse from the Passive Infrared Motion Sensors in that period of time. The current algorithm delays slightly after the VEX Ultrasonic Range Finder finishes its pulse in order to properly detect a pulse from the Passive Infrared Motion Sensors, if they detected motion, which lasts for approximately one second. Because of this, the VEX Ultrasonic Range Finder is left idling for part of every check since the time required for it to detect the distance to the nearest object is so short. An improved algorithm would determine a better way of operating all of the sensors more efficiently while still checking their outputs simultaneously.

Finally, the design could be improved by developing a means of alerting the user whenever motion is detected. The current system employs the use of a buzzer, which sounds every time an alert is generated due to motion. A further development could involve designing a mobile phone application that can communicate with the system via Bluetooth and alert the user. This application could include text alerts and display data regarding the sensor or sensors that triggered the alert as well as an audio alert. The algorithm for motion detection would remain relatively the same, but additional hardware and software functionality would need to be implemented to allow the system to communicate with a mobile device, such as a smart phone.

Application of this Research

This research is beneficial because it can lead to the development of an improved vehicular backup alarm system. There is a need for a system that can warn drivers of approaching pedestrians, vehicles, and other obstacles as they prepare to back out of a parking space. With the number of large vehicles that flood roads today, those who drive smaller cars often have such a restricted view when attempting to back out of a parking space that they cannot see beyond what is directly behind their vehicle. Even those who drive larger vehicles often have restricted views of the peripherals of their vehicles as they back out of parking spaces. The combination of this restricted view and people travelling too fast in small, heavily trafficked areas has led to many accidents and injuries. Not only have cars hit other cars, but they have hit pedestrians, some of whom have been killed (Backovers, 2013). Clearly, there is a need for a device that can alert drivers of approaching obstacles as they attempt to back out of parking spaces. This research is the beginning of the development of a new system to solve this problem. The focus of this project was to develop a prototype for an improved system based on ultrasonic and infrared motion sensing technologies that provides earlier obstacle detection over a larger area. The prototype includes an algorithm for detecting moving obstacles behind a vehicle, determining whether they are leaving or approaching, and generating alerts to the user if the obstacles are approaching. This implementation used the VEX Ultrasonic Range Finder and Passive Infrared Motion Sensors in tandem. This research can be further developed to create an effective system that can make a positive impact by decreasing the number of accidents and pedestrian injuries that occur when vehicles back out of parking spaces.

Conclusion

This directed research project sought to develop a prototype of a vehicular backup alarm system that provides earlier obstacle detection over a larger area by using two different types of sensor technologies: infrared and ultrasonic. By employing these two types of sensors, a system could be designed to provide an alert for drivers whenever a pedestrian or vehicle approaches their vehicles as they back out of a parking space. A circuit was designed to combine two Passive Infrared Motion Sensors and one VEX Ultrasonic Range Finder connected to a PIC18F242 microcontroller. This microcontroller operated these sensors simultaneously, accepting input from each in order to determine whether motion was detected. The software algorithm developed for this process also handled the different input events from the sensors and determined when to trigger an alert, which was simulated using a buzzer. This operational prototype proves that these sensors can be combined to detect motion and alert a user of an approaching obstacle. The next step in the development of a complete system is to implement this prototype in a complete system using multiple sensors placed along the rear bumper of a vehicle and alerting a driver of an approaching obstacle. This can be completed by another student who, along with Dr. Wang, can develop a more advanced sensor system that combines the technologies of both the infrared motion sensor to detect an obstacle and the ultrasonic sensor to determine the distance to it. This system would be much more accurate than the VEX Ultrasonic Range Finder and Passive Infrared Motion sensor system used in the prototype, but would follow the basic premise of the algorithm employed in the prototype. The ultimate conclusion to the research compiled in this project and further refined by a future student is the creation, implementation, and

production of a vehicular backup alarm system that employs infrared and ultrasonic motion sensing technologies to provide a larger and more accurate detection area behind and extending to the sides of the vehicle. Therefore, this research has provided a firm foundation for the creation of an improved vehicular backup alarm system that can be made available on the commercial market in the near future.

References

35050 PIR motion sensor [Pamphlet; PDF]. (n.d.). Retrieved from https://solarbotics.com/product/35050/

Backovers. (2013). Retrieved September 9, 2013, from http://www.kidsandcars.org/back-overs.html

Car backup cameras. (2013, February). Retrieved from http://www.consumerreports.org/cro/2012/12/car-backup-cameras/index.htm

Directed Electronics, Inc. (2004). *9401T back up sensor* [Pamphlet; PDF].

HY Technologies. (n.d.). *Backup parking sensor system: Model 4015N* [Pamphlet; PDF]. Retrieved from http://parkingsensors.net

Microchip Technology, Inc. (2006). *PIC18FXX2 data sheet* [Pamphlet; PDF]. Retrieved from http://www.microchip.com

PIR sensor [Pamphlet; PDF]. (2010). Retrieved from https://solarbotics.com/product/35050/

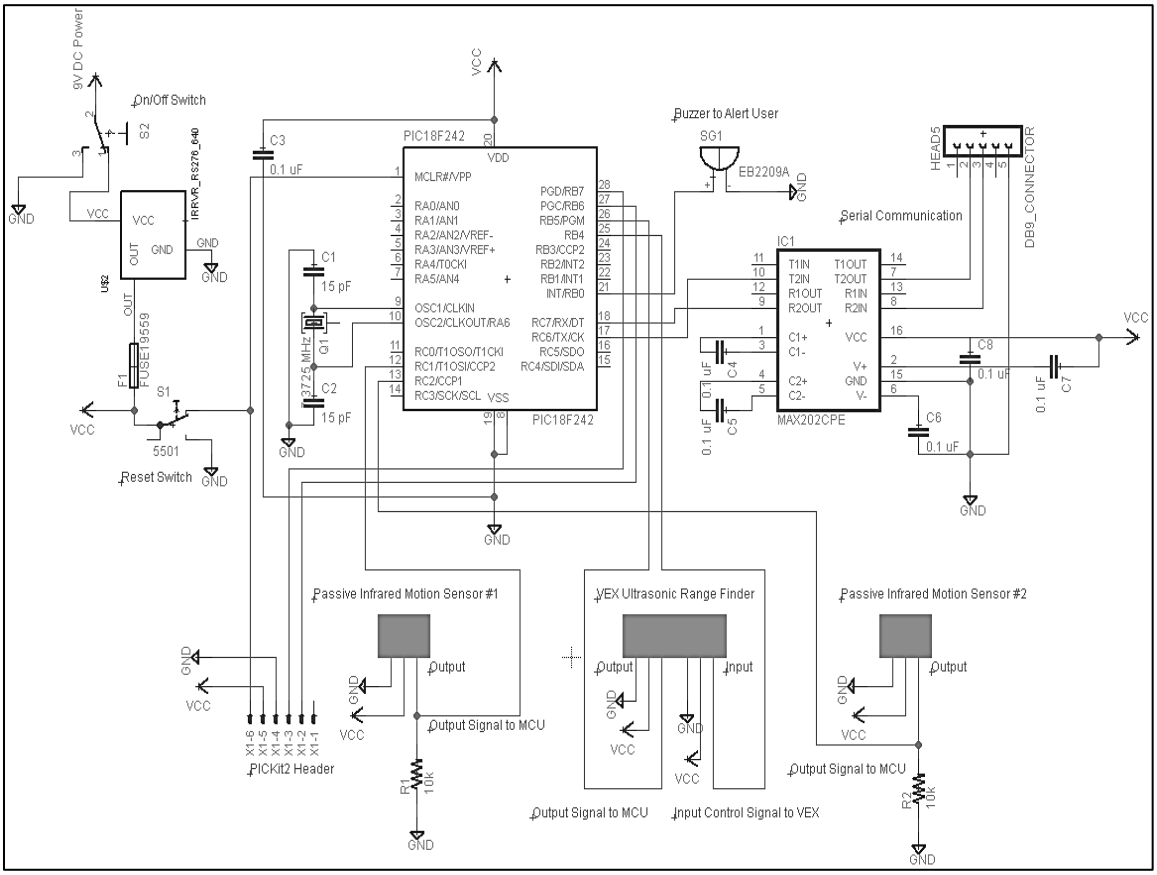
Rear object detection systems. (n.d.). Retrieved from http://www.fmcsa.dot.gov/facts-research/systems-technology/product-guides/rear-object-detection.htm

ReverseGUARD FAQ. (2013). Retrieved from http://americanroadproducts.com

Ultrasonic range finder. (2012, May 4). Retrieved December 8, 2012, from http://www.vexforum.com/wiki/index.php/Ultrasonic_Range_Finder

VEX Robotics. (n.d.). *Ultrasonic sensor* [Pamphlet; PDF]. Retrieved from http://www.vexrobotics.com

Appendix A: Prototype Circuit Schematic



Appendix B: Pictures of Circuit Prototype

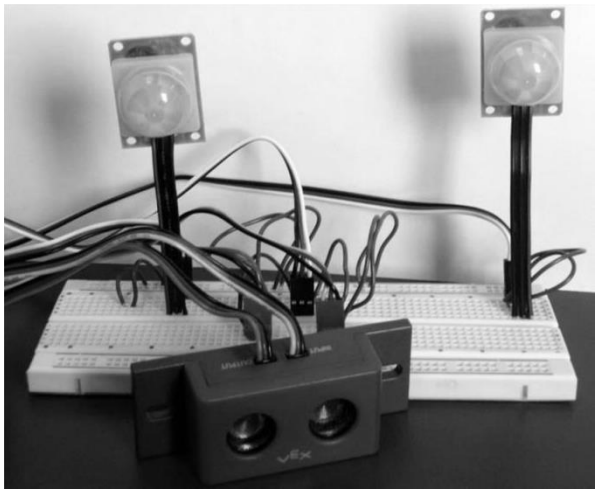


Figure 9. The Two Passive Infrared Motion Sensors and the VEX Ultrasonic Range Finder Used in the Prototype.

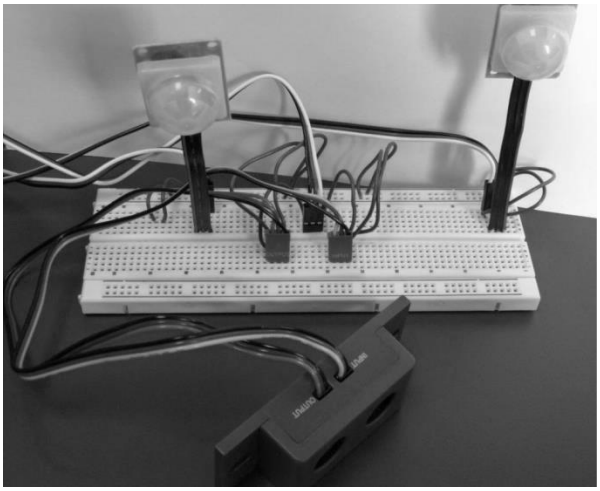


Figure 10. The Three Sensors Used for Motion Detection Connected to a Separate Circuit Board.

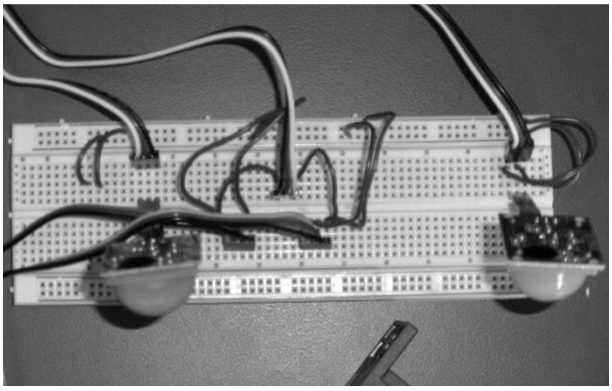


Figure 11. Overhead View of Sensor Circuit Board.

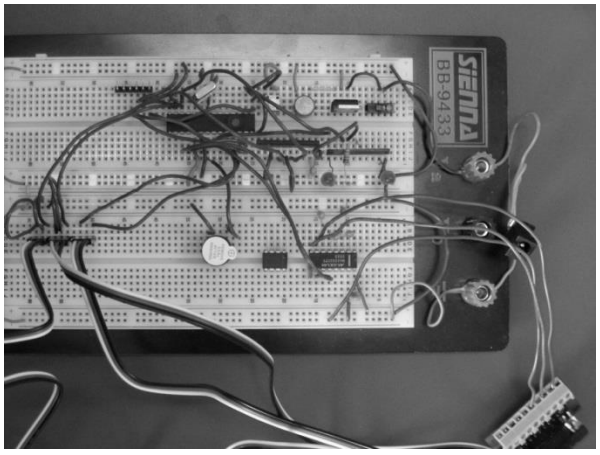


Figure 12. The Main Circuit Board with Microcontroller, Buzzer, and Serial Connector.

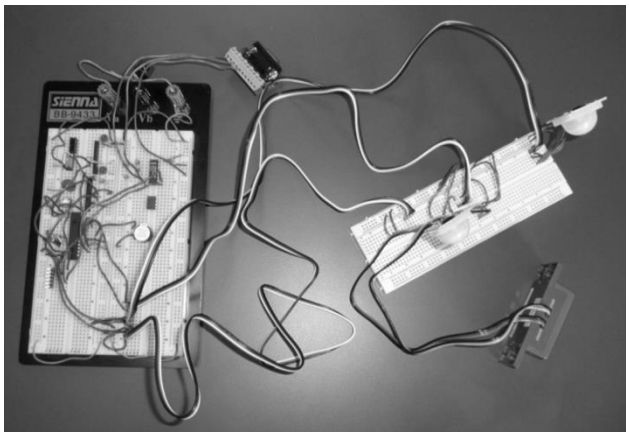


Figure 13. Overhead View of Entire System Consisting of Two Circuit Boards.

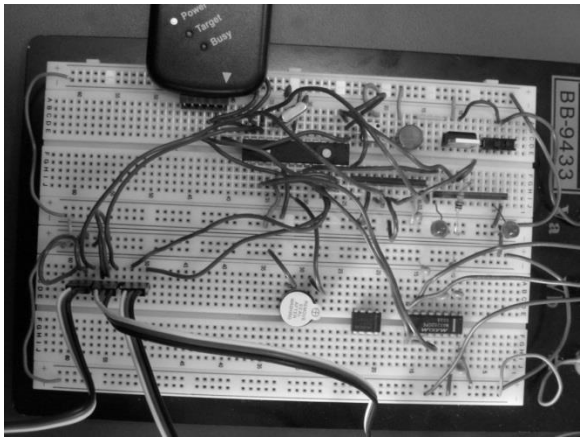


Figure 14. The PICKit2 Programmer Connected to the Main Circuit Board.

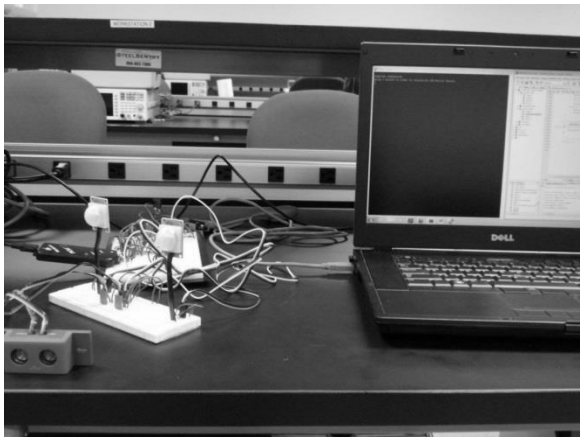


Figure 15. Circuit Running with Terminal Window Open on Left Side of Laptop Screen.

Appendix C: Algorithm Source Code

Buffer.h

```
/*
 * File:       buffer.h
 * Author:  Joshua Isaacson
 * Date:      23 April 2013
 */

#ifndef _BUFFER_H_
#define _BUFFER_H_

#include "config.h"

/* Buffer configuration */
#define BUFMAX 20          // Maximum number of characters in the buffer

/* Buffer status conditions */
#define BUF_STAT_READY      0   // The buffer is ready to receive more data
#define BUF_STAT_FULL       1   // No more data may be written to the buffer without destroying
                                //    existing data
#define BUF_STAT_EMPTY      2   // All data has already been read from the buffer

/* Structure containing the variables needed to maintain a wrap-around buffer */
struct Buffer
{
   unsigned char buf[BUFMAX]; // Array representing the data buffer
   unsigned char wpos;        // Write functions's position in the buffer
   unsigned char rpos;        // Read function's position in the buffer
};

INLINE_IF_POSSIBLE void buf_init( struct Buffer * buf );
INLINE_IF_POSSIBLE void buf_write( struct Buffer * buf, unsigned char c );
INLINE_IF_POSSIBLE unsigned char buf_read( struct Buffer * buf );
INLINE_IF_POSSIBLE unsigned char buf_status( struct Buffer * buf );

#endif // _BUFFER_H_
```

Buffer.c

```
/*
 * File:       buffer.c
 * Author:  Joshua Isaacson
 * Date:      23 April 2013
 */

#include "buffer.h"

/*
 * Get the buffer ready for use.
 */
INLINE_IF_POSSIBLE void buf_init( struct Buffer * buf )
{
   buf->wpos = 0;
   buf->rpos = 0;
}

/*
 * Write the specified character to the buffer.
 */
INLINE_IF_POSSIBLE void buf_write( struct Buffer * buf, unsigned char c )
{
   if( ++buf->wpos == BUFMAX ) buf->wpos = 0;
   buf->buf[buf->wpos] = c;
}

/*
 * Read the next character from the buffer.
 */
INLINE_IF_POSSIBLE unsigned char buf_read( struct Buffer * buf )
{
   if( ++buf->rpos == BUFMAX ) buf->rpos = 0;
   return buf->buf[buf->rpos];
}

/*
 * Return the current status of the buffer based on the BUF_STAT_* definitions.
 */
INLINE_IF_POSSIBLE unsigned char buf_status( struct Buffer * buf )
{
   if( buf->rpos == buf->wpos ) return BUF_STAT_EMPTY;
   if( buf->rpos == ((buf->wpos == 0) ? BUFMAX - 1 : buf->wpos) ) return BUF_STAT_FULL;
   return BUF_STAT_READY;
}
```

Config.h

```
/*
 * File:       config.h
 * Author:  Joshua Isaacson
 * Date:      23 April 2013
 */

#ifndef CONFIG_H
#define CONFIG_H

#include <htc.h>

/* PIC18F242 configuration details */
#pragma config OSC=HSPLL
#pragma config BOR=OFF
#pragma config PWRT=OFF
#pragma config WDT=OFF
#pragma config DEBUG=OFF
#pragma config LVP=OFF

/* Bit manipulation macros suggested by the HI-TECH C manual */
#define bitset(var,bitno) ((var) |= (1 << (bitno)))
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))
#define bittst(var,bitno) (var & (1 << (bitno)))

/* Internal configuration options */
#ifdef DEBUG
#define INLINE_IF_POSSIBLE
#else // !DEBUG
#define INLINE_IF_POSSIBLE inline
#endif // DEBUG

#endif  // CONFIG_H
```

Delay.h

```
/*
 * File:       delay.h
 * Author:  Joshua Isaacson
 * Date:       23 April 2013
 *
 * This file was based on delay.h from the example code provided on the CD with
 * "MICROPROCESSORS FROM ASSEMBLY LANGUAGE TO C USING THE PIC18FXX2" by
 * ROBERT B. REESE. Any modifications were made by the authors above to suit
 * their project specifications.
 */

#ifndef DELAY_H
#define DELAY_H

/* Clock speed definitions for the PIC18F242 */
#define FOSC    29490L    // Internal clock freq in kHz
#define MHZ     *1000L    // number of kHz in 1 MHz
#define KHZ      *1           // number of kHz in 1 kHz

/* Global Variables */
extern unsigned char kill_delay;      // Force DelayMsKill() to return even if the time has not expired.

/* Macros */
#define DelayUs( x ) \
{ \
   unsigned char _dcnt; \
   _dcnt = ((x * FOSC)/(12MHZ)); \
   while( --_dcnt != 0 ) continue; \
}

/* Delay function declarations */
void DelayMs( unsigned char cnt );
void ISR_DelayMs( unsigned char cnt );
void DelayMsKill( unsigned char cnt );
void Delay( unsigned char cnt );

#endif // DELAY_H
```

Delay.c

```
/*
 * File:       delay.c
 * Author:  Joshua Isaacson
 * Date:      23 April 2013
 *
 * This file was based on delay.h from the example code provided on the CD with
 * "MICROPROCESSORS FROM ASSEMBLY LANGUAGE TO C USING THE PIC18FXX2" by
 * ROBERT B. REESE. Any modifications were made by the authors above to suit
 * their project specifications.
 */

#include "config.h"
#include "delay.h"

/* Global Variables */
unsigned char kill_delay;   // Force DelayMsKill() to return even if the time has not expired.

/*
 * Delay for between 1 and 255 milliseconds.
 */
void DelayMs( unsigned char cnt )
{
   unsigned char i;
   // Delay 1 ms for cnt times
   do
   {
      i = 20;
      do
      {
         DelayUs( 50 );
      } while( --i );
   } while( --cnt );
}

/*
 * Delay loop from an ISR (otherwise the same as DelayMs().
 * It is not recommended to delay within an ISR, but use this function if you must.
 */
void ISR_DelayMs( unsigned char cnt )
{
   unsigned char i;
   // Delay 1 ms for cnt times
   do
   {
      i = 20;
      do
      {
         DelayUs( 50 );
      } while( --i );
   } while( --cnt );
}
```

```
/*
 * Delay for between 1 and 255 milliseconds or when kill_delay is set.
 */
void DelayMsKill( unsigned char cnt )
{
   unsigned char i;
   kill_delay = 0;
   // Delay 1 ms for cnt times
   do
   {
      i = 20;
      do
      {
         DelayUs( 50 );
      } while( --i && !kill_delay );
   } while( --cnt && !kill_delay );
}

/*
 * Delay for between 1 and 255 seconds.
 */
void Delay( unsigned char cnt )
{
   unsigned char i, k;
   // Delay 1 second cnt times
   for( k = 0; k < cnt; k++ )
      for( i = 0; i < 10; i++ ) DelayMs( 100 );
}
```

Serial.h

```
/*
 * File:       serial.h
 * Author:  Joshua Isaacson
 * Date:      23 April 2013
 */

#ifndef _SERIAL_H_
#define _SERIAL_H_

#include "config.h"

/* Port definitions */
#define TX RC6  // Port used to send data
#define RX RC7  // Port used to receive data

INLINE_IF_POSSIBLE void serial_buffer_char();
INLINE_IF_POSSIBLE void serial_init();
INLINE_IF_POSSIBLE void enable_interrupts();

#endif // _SERIAL_H_
```

Serial.c

```
/*
 * File:       serial.c
 * Author:  Joshua Isaacson
 * Date:      23 April 2013
 */

#include "serial.h"
#include "buffer.h"
#include "delay.h"

#include <stdio.h>

/* Global variables */
static volatile struct Buffer rc_buf; // Characters received over serial

/*
 * Low-level function to place a character received over serial in our buffer.
 *
 * Remarks:
 *      Only call this function from your interrupt function IF you have your
 *      program set to use interrupts. Otherwise make sure you have
 *      SERIAL_NO_INTERRUPTS defined in your global config. ("Global config"
 *      refers to either config.h or your compiler's flag definition feature.)
 */
INLINE_IF_POSSIBLE void serial_buffer_char()
{
   buf_write( &rc_buf, RCREG );
}

/*
 * Send a character over serial.
 */
void putch( char c )
{
   while( !TXIF ) continue;
   TXREG = c;
}

/*
 * Get a character over serial.
 */
char getch()
{
#ifdef SERIAL_NO_INTERRUPTS
   while( !RCIF ) continue;
   serial_buffer_char();
#else // !SERIAL_NO_INTERRUPTS
   while( rc_buf.rpos == rc_buf.wpos )
   {
      DelayMs( 100 );
      //CLRWDT();
   };
```

```
#endif // SERIAL_NO_INTERRUPTS
   return buf_read( &rc_buf );
}

/*
 * Get a character over serial and echo it.
 */
char getche()
{
   char c; // Character recieved
   c = getch();
   putch( c );
   return c;
}

/*
 * Initialize the software serial connection.
 */
INLINE_IF_POSSIBLE void serial_init()
{
   // Setup I/O ports.
   TRISC = 0xBF;
   TX9 = 0;
   RX9 = 0;
   // Configure hardware serial settings.
   TXEN = 1;
   SYNC = 0;
   BRGH = 1;
   SPBRG = 15;   // BAUD 115200 hi-speed
   // Enable hardware serial support.
   SPEN = 1;
   CREN = 0;
   CREN = 1;
   // Initialize the buffer.
   buf_init( &rc_buf );
}

/*
 * Enable interrupts.
 */
INLINE_IF_POSSIBLE void enable_interrupts()
{
   // Enable priority interrupts for Sensors.
   IPEN = 1;
   // Enable PORTC interrupts for serial communication.
   RCIE = 1;
   // Enable all low priority peripheral interrupts when IPEN = 1;
   //    else enable all unmasked peripheral interrupts
   PEIE = 1;
   // Enable all high priority interrupts when IPEN = 1;
   //    else enable all unmasked interrupts
   GIE = 1;
}
```

Vehicular_Alarm_System.c

```
/*
 * File:       vehicular_alarm_system.c
 * Author:  Joshua Isaacson
 * Date:       23 April 2013
 */

#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <ctype.h>

#include "delay.h"
#include "serial.h"


/* Definitions used for debugging to easily activate/deactivate PIR and Ultrasonic sensor code */
#define PIR_SENSOR              // When defined, all PIR code will be active and compiled
#define ULTRASONIC_SENSOR       // When defined, all Ultrasonic code will be active and compiled

/* Delay and Timeout Constants */
#define MAX_DELAY               2           // Number of half seconds to delay between tests
#define TMR0_TIMEOUT            20000       // Arbitrary # of TMR0 interrupts before timeout of test
#define DURATION_MAX            999999  // Set a max value for when the VEX timer times out
#define MICROSECONDS_PER_SECOND     1000000     // The number of microsoeconds in 1 second

/* VEX SENSOR DECLARATIONS */
#define FOSC                    29490L  // Internal clock freq in KHz
#define VEX_TIMEOUT             1000        // # of TMR0 interrupts before timeout of Ultrasonic sensor
#define PAST_ARRAY_LEN          3           // Length of array past_durations

/* Declared own type MYLONG, which is an unsigned long long */
typedef unsigned long long MYLONG;

/* Distance Threshold Variables */
#define MPH     5               // Speed of car in reverse used for threshold calculation
#define TIME    50              // Arbitrary estimated time in ms for entire program to run

/* Port definitions */
#define TX                  RC6         // Port used to send data
#define RX                  RC7         // Port used to receive data
#define IN_VEX              RB4         // Port used to monitor input from VEX Ultrasonic Sensor
#define OUT_VEX             LB3         // Port used to send start signal to VEX Ultrasonic Sensor
#define BUZZER              LB0         // Port used to send high signal to sound buzzer

/* Timer Flags */
#define FLAG_NONE       0
#define FLAG_PULSE      1       // Ultrasonic Pulse being used
#define FLAG_TO         2       // Timeout flag for distance calculation

/* RBIF Flags */
#define FLAG_HIGH_TO_LOW    4   // Flag for IN_VEX currently high - just changed to low
#define FLAG_LOW_TO_HIGH    8   // Flag for IN_VEX currently low - just changed to high
```

```
/* Interrupt Variables */
volatile unsigned long timer0_overflows;          // Number of times TMR0 interrupt occurred on overflow
volatile unsigned long timer0_count_ultrasonic;   // Number of times TMR0 interrupt occurred during
                                                  //    Ultrasonic sensing period
volatile unsigned long timer0_rem_ultrasonic;     // Remaining cycles of TMR0 after last interrupt during
                                                  //    Ultrasonic sensing period
volatile unsigned long flags = FLAG_NONE;         // Flag value for state of Ultrasonic sensor


/* PIR MOTION SENSOR DECLARATIONS */

#ifdef PIR_SENSOR

/* CCP1CON/CCP2CON Mode Settings (Excluding Pulse Width Modulation) */
#define CCPCON_MODE_DISABLED    0x0  // Capture/Compare/PWM disabled (resets CCPx
                                     //    module)
#define CCPCON_MODE_OUTPUT      0x2  // In compare mode set output on match (meaning the
                                     //    CCPxIF bit is set)
#define CCPCON_MODE_FALLING     0x4  // Capture on every falling edge
#define CCPCON_MODE_RISING      0x5  // Capture on every rising edge
#define CCPCON_MODE_RISING4     0x6  // Capture on every 4th rising edge
#define CCPCON_MODE_RISING16    0x7  // Capture on every 16th rising edge
#define CCPCON_MODE_COMP_LOW    0x8  // Initialize the CCP pin Low and set it High on compare
                                     //    match (CCPIF bit is set)
#define CCPCON_MODE_COMP_HIGH   0x9  // Initialize the CCP pin High and set it Low on compare
                                     //    match (CCPIF bit is unset)
#define CCPCON_MODE_INTERRUPT   0xA  // Generate an interrupt on compare match (CCPIF bit is
                                     //    set, but CCP pin is unaffected)
#define CCPCON_MODE_TRIGGER     0xB  // Trigger a special event on compare match (CCPIF bit
                                     //    is set)

#endif /* PIR_SENSOR */

/* Motion Sensor Capture Variables */
volatile unsigned char ccp1_detect;    // Did CCP1 capture a high pulse?
volatile unsigned char ccp2_detect;    // Did CCP2 capture a high pulse?


/*
 * Interrupt processing routine for hardware serial input.
 */
void interrupt pic_isr()
{
   // Handle Serial Buffer Interrupts.
   if( RCIF )
   {
      serial_buffer_char();
   };

   //
   // Handle PIR Sensor Interrupts
   //
```

```
#ifdef PIR_SENSOR
   // Handle CCP1 Interrupt for Rising and Falling Edges.
   //   This captures the High Pulse from the PIR Motion Sensor after motion is detected.
   if( CCP1IF )
   {
      if( (CCP1CON & 0x0F) == CCPCON_MODE_FALLING )
      {
         // Print capital F for falling edge of CCP1
         printf( "F" );
      }
      else
      {
         // After a Rising Edge is detected, set the capture mode to a Falling
         //   Edge to determine end of motion detection.
         CCP1CON = CCPCON_MODE_DISABLED;
         CCP1CON = CCPCON_MODE_FALLING;
         ccp1_detect = 1;

         // Print capital R for rising edge of CCP1
         printf( "R" );
      };

      CCP1IF = 0;
   };

   // Handle CCP2 Interrupt for Rising and Falling Edges.
   //   This captures the High Pulse from the PIR Motion Sensor after motion is detected.
   if( CCP2IF )
   {
      if( (CCP2CON & 0x0F) == CCPCON_MODE_FALLING )
      {
         // Print lowercase f for falling edge of CCP2
         printf( "f" );
      }
      else
      {
         // After a Rising Edge is detected, set the capture mode to a Falling
         //   Edge to determine end of motion detection.
         CCP2CON = CCPCON_MODE_DISABLED;
         CCP2CON = CCPCON_MODE_FALLING;
         ccp2_detect = 1;

         // Print lowercase r for rising edge of CCP2
         printf( "r" );
      };

      CCP2IF = 0;
   };
#endif /* PIR_SENSOR */


   //
   // Handle VEX Ultrasonic Sensor Interrupts
   //
```

```
   // Timer 0 is handling timing for both the distance calculation and the
   //      overall period of time that the PIR sensors are sensing.
   if( TMR0IF )
   {
      // Increment counter timer0_overflows each time TMR0 overflows
      timer0_overflows++;

      // Reset TMR0
      TMR0IF = 0;
      TMR0IE = 0;
      TMR0IE = 1;
   }

#ifdef ULTRASONIC_SENSOR
   // Timer 2 is handling timing for ultrasonic pulse.
   if( TMR2IF )
   {
      // End of 10 us control signal for ultrasonic pulse - turn off TMR2
      OUT_VEX = 0;
      TMR2IF = 0;
      TMR2IE = 0;
      TMR2ON = 0;
      flags = FLAG_LOW_TO_HIGH;
   }

   // RBIE is the interrupt-on-change for RB4 that is used to determine
   //    the start and end of the distance calculation.
   if( RBIF && RBIE )
   {
      RBIF = 0;

      // Interrupt-on-change from low to high value (start of sensing time)
      if( flags & FLAG_LOW_TO_HIGH )
      {
         timer0_overflows = 0;
         timer0_count_ultrasonic = 0;

         // Reset TMR0
         TMR0IF = 0;
         TMR0IE = 0;
         TMR0IE = 1;
         flags = FLAG_HIGH_TO_LOW;
      }

      // Interrupt-on-change from high to low value (end of sensing time)
      else if( !IN_VEX )
      {
         // Read in the last value of TMR0
         timer0_rem_ultrasonic = TMR0;

         // Read in the number of TMR0 overflows during sensing period
         timer0_count_ultrasonic = timer0_overflows;
```

```
            // Disable RBIE
            RBIE = 0;
            flags = FLAG_NONE;
        }
    };
#endif /* ULTRASONIC_SENSOR */
}

/*
 * Delays 0.5 seconds.
 */
void delay_half_second()
{
    for( char i = 0; i < 2; i++ ) DelayMs( 250 );
}

/*
 * Convert number of microseconds delay to a number of clock cycles.
 */
MYLONG microsecondsToClockCycles( unsigned long timeout )
{
    return ((MYLONG) timeout) * FOSC / 4000L;
}

/*
 * Convert number of clock cycles delay to a number of microseconds.
 */
MYLONG clockCyclesToMicroseconds( MYLONG cycles )
{
    return (cycles * 4000L) / ((MYLONG) FOSC);
}

/*
 * Calculate distance in meters given sensor duration in microseconds.
 */
float calc_distance( MYLONG duration )
{
    // distance in meters = 172.1 m/s * round trip delay
    // delay is in us, so need to divide delay by 10 ^ 6 to get seconds
    return (float) ( 172.1 * (float) duration / (float) MICROSECONDS_PER_SECOND );
}

/*
 * Calculate threshold duration (us) based on speed of car and program time.
 *   This duration corresponds to the distance that the vehicle will move
 *   in reverse during the time it takes for the test.  This is used to determine
 *   whether or not the object is approaching, leaving, or in the same location.
 */
MYLONG calc_threshold()
{
    // 1 mph = 0.00044704 m/ms
    // Multiply by TIME (in ms) of entire program to calculate threshold distance
    // Convert distance into duration in us based on calc_distance() formula
```

```
   return (MYLONG) ((float) (( .00044704 * MPH * TIME ) / 172.1) * (float)
MICROSECONDS_PER_SECOND);
}

/*
 * Convert distance in meters to feet.
 */
float convert_meters_to_feet(float meters)
{
   // cm = m * 100
   // in = cm / 2.54
   // ft = in / 12
   return (((meters * 100)/2.54)/12.0);
}

/*
 * Convert a float to two integers (int and decimal) so that it can be printed.
 */
void float_to_int(float num, unsigned int * int_portion, unsigned int * decimal_portion)
{
   *int_portion = (unsigned int) num;
   *decimal_portion = (unsigned int) ((num - *int_portion) * 100000);
}


/*
 * Program for the PIC18F242 that uses a PIR Motion Sensor
 *   to detect a nearby object and a VEX Ultrasonic Sensor to
 *   determine the distance to that object.
 */
void main()
{
   serial_init();
   enable_interrupts();

   printf( "\r\nSTARTING SIMULATION.\r\n" );

   // Set high priority for CCP1,2 interrupts and low priority for RB interrupts
   CCP1IP = 1;
   CCP2IP = 1;
   RBIP = 0;


   // Set RB4 to interrupt-on-change input for Ultrasonic sensor,
   //   RB3 is output to Ultrasonic sensor, RB0 is output to buzzer.
   TRISB = 0xF6;
   // Set PR2 register to 0x18 so that timeout occurs at ~10 us
   PR2 = 0x18;
   // Set clock source internal instruction cycle clock
   T0CS = 0;

   // Declare local variables necessary for sensor tests.
   MYLONG duration;
   float distance;
```

```
   float ft_distance;
   MYLONG threshold = calc_threshold();
   MYLONG past_durations[PAST_ARRAY_LEN];
   unsigned char past_count;
   unsigned char ultrasonic_detect = 0;
   unsigned long test_num = 0;


   // Declare variable to determine which event case occurred during test
   unsigned char event = 0;

   // Declare local variable to count the number of half second delays between tests
   unsigned char num_delay = 0;

#ifdef ULTRASONIC_SENSOR
   for( past_count = 0; past_count < PAST_ARRAY_LEN; past_count++)
   {
      past_durations[past_count] = 0;
   }
#endif /* ULTRASONIC_SENSOR */

#ifdef PIR_SENSOR
   printf( "Delay 1 minute in order to initialize PIR Motion Sensor.\r\n" );

   // Delay 60 seconds to initialize the PIR Motion Sensors
   for( num_delay = 0; num_delay < 120; num_delay++ )
   {
      delay_half_second();
   }
#endif /* PIR_SENSOR */

   while( 1 )
   {
      printf( "\r\nTEST: %d\r\n", test_num );
      test_num++;

      //
      // Start Sensors
      //

      printf( "\r\nStarting sensors.\r\n" );

      // Reset all interrupt detection flags and timer0_overflows counter.
      ccp1_detect = 0;
      ccp2_detect = 0;
      ultrasonic_detect = 0;
      timer0_overflows = 0;

#ifdef PIR_SENSOR
      // Configure CCP1 - PIR Motion Sensor
      CCP1IE = 0;
      CCP1CON = CCPCON_MODE_DISABLED;
      CCP1CON = CCPCON_MODE_RISING;
      CCP1IF = 0;
```

```
        // Configure CCP2 - PIR Motion Sensor
        CCP2IE = 0;
        CCP2CON = CCPCON_MODE_DISABLED;
        CCP2CON = CCPCON_MODE_RISING;
        CCP2IF = 0;

        // Enable CCP1 and CCP2 Interrupts - PIR Motion Sensors
        CCP1IE = 1;
        CCP2IE = 1;
#endif /* PIR_SENSOR */

        // Start TMR0
        TMR0IF = 0;
        TMR0IE = 1;
        TMR0ON = 1;

#ifdef ULTRASONIC_SENSOR
        // Prepare to send control signal pulse (Ultrasonic)
        flags = FLAG_PULSE;
        OUT_VEX = 0;
        DelayUs( 2 );

        // Start ultrasonic pulse and turn on TMR2 - interrupt will occur at ~10 us
        OUT_VEX = 1;
        TMR2IE = 1;
        TMR2ON = 1;
        RBIF = 0;
        RBIE = 1;

        // Wait for ultrasonic pulse to end.
        while( flags & FLAG_PULSE ) continue;

        // Wait for initial interrupt on change for IN_VEX to go high.
        while( flags & FLAG_LOW_TO_HIGH ) continue;

        // Distance timer about to start.
        if( flags & FLAG_LOW_TO_HIGH )
           while( flags & FLAG_LOW_TO_HIGH ) continue;

        // Wait until IN_VEX goes high (sensing has started).
        while( !( flags & FLAG_HIGH_TO_LOW ) ) continue;

        // IN_VEX is high, so keep counting.
        while( (flags & FLAG_HIGH_TO_LOW) && IN_VEX )
        {
           if( timer0_overflows >= VEX_TIMEOUT )
           {
              flags = FLAG_TO;
           }
        }
#endif /* ULTRASONIC_SENSOR */

        // If Timeout occurred during distance calculation,
```

```
    //    set duration to DURATION_MAX and skip distance calculation.
    if( flags & FLAG_TO )
    {
#ifdef ULTRASONIC_SENSOR
        printf( "Ultrasonic sensor timed out.\r\n" );
        duration = DURATION_MAX;
#endif /* ULTRASONIC_SENSOR */
    }
    // If no Timeout, then calculate the duration and distance.
    else
    {
#ifdef ULTRASONIC_SENSOR
        printf( "Calculating distance." );
        // IN_VEX is low, so stop counting and calculate duration and distance.
        // Calculate sensor pulse duration and object distance.
        duration = clockCyclesToMicroseconds((timer0_count_ultrasonic * 256) + timer0_rem_ultrasonic);
        if( duration > VEX_TIMEOUT * 256 )
        {
          ultrasonic_detect = 0;
          printf( "Ultrasonic sensor timed out...No object in range.\r\n" );
        }
        else
        {
          ultrasonic_detect = 1;

          printf( "Duration (us): %d\r\n", duration );

          distance = calc_distance( duration );

          ft_distance = convert_meters_to_feet(distance);

          unsigned int i1, i2;
          float_to_int( distance, &i1, &i2 );
          printf( "Distance (m) = %d.%d\r\n", i1, i2 );
          float_to_int( ft_distance, &i1, &i2 );
          printf( "Distance (ft) = %d.%d\r\n", i1, i2 );
        }
#endif /* ULTRASONIC_SENSOR */
    }


    // Wait until max time for sensing is elapsed
    while( timer0_overflows <= TMR0_TIMEOUT ) continue;

#ifdef PIR_SENSOR
    // Stop Motion Sensors - CCP1 and CCP2
    CCP1IE = 0;
    CCP2IE = 0;
#endif /* PIR_SENSOR */

    // Stop TMR0
    TMR0IF = 0;
    TMR0IE = 0;
    TMR0ON = 0;
```

```
        DelayMs( 20 );

#ifdef PIR_SENSOR
        if( ccp1_detect != 0 )
        {
           printf( "\r\nMotion detected from CCP1.\r\n" );
        }

        if( ccp2_detect != 0 )
        {
           printf( "\r\nMotion detected from CCP2.\r\n" );
        }
#endif /* PIR_SENSOR */

#ifdef ULTRASONIC_SENSOR

        // When backing up at a speed of MPH:
        //
        // If object is at same distance or farther, then object is leaving
        // Else if duration is less than the old duration minus
        //      threshold, then the object is approaching
        // Else, the object is moving away slower than the car is moving towards it
        //      or the object didn't move.


        // Calculate whether object is approaching or leaving
        //   Turn ON Buzzer when there is an Alert
        if( duration > ( past_durations[0]) )
        {
           printf( "Object leaving.\r\n" );
           BUZZER = 0;
        }
        else if( duration < ( past_durations[0] - threshold ) )
        {
           printf( "ALERT: Object approaching.\r\n" );
           BUZZER = 1;
        }
        else
        {
           printf( "Object is slowly approaching.\r\n" );
           BUZZER = 0;
        }

        // Alert user if object is less than 1 meter away
        if ( distance < 1.0 )
        {
           printf( "ALERT: Object less than 1 meter away.\r\n" );
           BUZZER = 1;
        }

        // Update past_durations to include most recent PAST_ARRAY_LEN duration times.
        for( past_count = PAST_ARRAY_LEN - 1; past_count > 0; past_count--)
        {
```

```
            past_durations[past_count] = past_durations[past_count - 1];
        }

        past_durations[0] = duration;
#endif /* ULTRASONIC_SENSOR */

        // Reset flags to NONE.
        flags = FLAG_NONE;


        //
        // Event Handling
        //

        // CASES
        // CCP1    Ultra     CCP2     Result
        //  0        0         0       No         (0)
        //  0        0         1       Probably   (1)
        //  0        1         0       Doubtful   (2)
        //  0        1         1       Most Likely (3)
        //  1        0         0       Probably   (4)
        //  1        0         1       Yes        (5)
        //  1        1         0       Most Likely (6)
        //  1        1         1       Yes        (7)


        // NOTE: CCP1 and CCP2 (PIR Motion Sensors) are given a higher weight
        //    than the Ultrasonic Sensor because they have a wider detection
        //    range and detect motion instead of just any object within a certain distance.

        event = (ccp1_detect << 2) | (ultrasonic_detect << 1) | (ccp2_detect);

        switch ( event )
        {
            case 0:
                printf( "\r\nMotion was not detected at all.\r\n" );
                break;
            case 1:
                printf( "\r\nMotion was probably detected - only CCP2.\r\n" );
                break;
            case 2:
                printf( "\r\nDoubtful that accurate motion was detected - only Ultrasonic.\r\n" );
                break;
            case 3:
                printf( "\r\nMotion was most likely detected - CCP2 and Ultrasonic.\r\n" );
                printf( "ALERT!\r\n" );
                BUZZER = 1;
                break;
            case 4:
                printf( "\r\nMotion was probably detected - only CCP1.\r\n" );
                break;
            case 5:
                printf( "\r\nMotion was definitely detected - CCP1 and CCP2.\r\n" );
                printf( "ALERT!\r\n" );
```

```
            BUZZER = 1;
            break;
         case 6:
            printf( "\r\nMotion was most likely detected - CCP1 and Ultrasonic.\r\n" );
            printf( "ALERT!\r\n" );
            BUZZER = 1;
            break;
         case 7:
            printf( "\r\nMotion was definitely detected - All 3 Sensors.\r\n" );
            printf( "ALERT!\r\n" );
            BUZZER = 1;
            break;
         default:
            printf( "\r\nERROR: Bad Event Calculation.\r\n" );
            break;
      }

      // Delay MAX_DELAY number of half seconds between tests during debug process
      for( num_delay = 0; num_delay < MAX_DELAY; num_delay++ )
      {
         delay_half_second();
      }

      // Turn off buzzer
      BUZZER = 0;
   }
}
```