

MICHIGAN STATE UNIVERSITY COLLEGE OF ENGINEERING

Autonomous Target Tracking Robot

ECE 480 Design Team 7

Victor Abreu, Matthew Beutler, Brent Eisenmann

Hisham Hassan, Thomas Schriefer, Peng Xie

Course Instructor: Timothy Grotjohn

Design Team Faculty Facilitator: Jian Ren

April 26, 2013

DEPARTMENT OF ECE
ELECTRICAL AND COMPUTER ENGINEERING

Executive Summary

The Michigan State Department of Electrical and Computer Engineering has asked Design Team 7 to create a robot that is capable of autonomously tracking a predefined target. The primary function for this device is for military use, but could also be marketed and utilized in the private sector. This type of robot is essential for jobs such as detecting bombs, or tracking a target to an area that would not be possible or safe with an actual person. Design Team 7 must create a design that is capable of having both manual and autonomous control modes. After considering the problem and analyzing the predefined design requirements, Design Team 7 chose to go with a tank style prototype. This prototype will be remotely controlled by an onboard microcontroller, and use Bluetooth as a mode of wireless control and data transfer. Once a target is defined, the robot will follow the object for a period of at least one hour at various speeds. If the target is lost, a manual override will be available to drive the robot to the target location. Team 7 was successfully able to create a design that met the customer requirements.

Acknowledgement

Electrical and Computer Engineering Department: A Special thanks to the Michigan State ECE department for sponsoring this senior design project.

Dr. Jian Ren: A special thanks to Dr. Ren, the faculty facilitator, for assisting in the overall design process.

Dr. Timothy Grotjohn: Special thanks to Dr. Grotjohn, the professor of this year's ECE 480 Senior Design class, as well as the sponsor representative for the ECE department for Design Team 7.

Gregg Motter: Thanks to Mr. Motter of Dow Chemical, for providing valuable information during the in class lectures, especially those pertaining to the Six Sigma strategies for problem solving.

Electrical and Computer Engineering Shop: Thanks to the ECE Shop for providing necessary parts used in the circuit design.

This was a wonderful learning experience for all of Design team 7, and was all made possible by all those mentioned above.

TABLE OF CONTENTS

CHAPTER 1	6
INTRODUCTION	6
BACKGROUND	7
CHAPTER 2	8
SELECTING A DESIGN APPROACH	8
DESIGN SPECIFICATIONS	8
FAST DIAGRAM	9
CONCEPTUAL DESIGNS	10
<i>Quad-copter</i>	10
<i>Four Wheel Independent Axis Remote Control Car</i>	10
<i>Tank</i>	11
<i>Control</i>	11
<i>Tracking</i>	11
SOLUTION	12
PROJECT SCHEDULE	13
PROJECT BUDGET	14
CHAPTER 3	15
HARDWARE DESIGN EFFORTS	15
ARDUINO UNO	15
DFRDUINO MOTOR SHIELD	16
DEVANTECH SRF05 ULTRASONIC SENSORS	17
BLUETOOTH MATE SILVER	20
LI-ION BATTERY & CHARGER	21
D-LINK WIRELESS ENABLED CAMERA	23
HARDWARE IMPLEMENTATION & PHOTO IDENTIFICATION	24
HARDWARE DESIGN DIAGRAM	24
ARDUINO UNO	25
DFRDUINO MOTOR SHIELD	26
DEVANTECH SRF05 ULTRASONIC SENSORS	28
BLUETOOTH MATE SILVER	27
LI-ION BATTERY & CHARGER	30
D-LINK WIRELESS ENABLED CAMERA	34
SOFTWARE & INTERFACE DESIGN REQUIREMENTS	35
SOFTWARE DESIGN DIAGRAM	35
SOFTWARE IMPLEMENTATION	36
C# CODE	36
ARDUINO CODE	37
ISSUES ENCOUNTERED	41
CHAPTER 4	42
ARDUINO FUNCTIONALITY TESTING	42
SOFTWARE TESTING	43

TABLE OF CONTENTS (cont)

CHAPTER 5	44
FINAL COST.....	44
SCHEDULE	45
SUMMARY AND CONCLUSIONS	45
APPENDIX 1	47
TECHNICAL ROLES, RESPONSIBILITIES, WORK ACCOMPLISHED.....	47
VICTOR ABREU.....	47
MATTHEW BEUTLER.....	48
BRENT EISENMANN.....	49
HISHAM HASSAN	50
THOMAS SCHRIEFER.....	51
PENG XIE.....	52
APPENDIX 2	53
LITERATURE AND WEBSITE REFERENCES	53
APPENDIX 3	55
SOLUTION SELECTION MATRICES.....	55
CRITICAL CUSTOMER REQUIREMENTS.....	55
DECISION MATRIX.....	56
GANTT CHART	57
WIRE SCHEMATICS	60
MODE 1 OPERATION DEVANTECH SRF05 ULTRASONIC SENSOR.....	62
WIRE CONNECTIONS.....	63
INITIAL PROTOTYPE DESIGN	65
C# SOURCE CODE.....	66
ARDUINO SOURCE CODE.....	74

CHAPTER 1

INTRODUCTION

The design team, consisting of seniors majoring in electrical or computer engineering, has been assigned the task to design and create an autonomous robot that is capable of following a predefined target. This technology could be utilized and implemented in a number of different areas, such as military applications and child monitoring. The sponsor for this project is the Michigan State Electrical and Computer Engineering Department. No initial components were provided to the team by the sponsor, so significant research had to be done on every part.

There were many project design requirements that were specified to the team by the sponsor. The sponsor also listed what components were mandatory, such as rangefinders and a camera. The design team first specified the object to be tracked as a brightly colored ball for simplicity and ease of testing. The robot must have both autonomous and manual modes of operation, and be wirelessly connected to a base station laptop. Once the target is acquired by the camera, it will close to a certain distance from the object, and remain at that distance through the autonomous operation mode. One of the most important components is the battery, because it has to power all other devices for an extended period of time. The final prototype must be built to survive in normal indoor environments, as well as protect the internal electrical components.

The design team created a prototype solution to meet the requirements with a budget of \$500. This project includes many aspects of both electrical and computer engineering, such as wireless network communication, microcontroller communication, motor control, as well as circuit and power design.

BACKGROUND

The first instance of modern robotics appeared in 1948 when George Devol and Joe Engleberger created a mechanical arm. The two later created the first robotics company in 1956. In 1979, the Stanford Cart robot successfully crossed a chair filled room with no human assistance. It did this by using a camera to take pictures and then sent them to a computer for analysis. [1] The robot could travel at a rate of about one meter every 15-20 minutes, at which point it would then take pictures and reassess the surroundings. It did this by utilizing an onboard computer to make the necessary calculations. This computer is comparable to today's microcontrollers. [2] The first real rugged autonomous vehicle was the Terregater, which was created in 1984 for the primary purposes of road mapping and mine exploration. This was the first instance where an autonomous robot was created for a specific industrial function. [3]

The idea of an autonomous tracking robot is relatively new technology. Robots have been around for decades, but having one that can essentially think on its own has not been explored until recent developments in microcontroller technology. Recent developments in Wi-Fi and Bluetooth technologies have also allowed for exploration into wireless control and data transfer. Due to these factors, interest in this field is beginning to explode, especially because of a recent mandate set forth by the US military. "The U.S. Congress has mandated that by the year 2015, one-third of ground combat vehicles will be unmanned, and the DOD is now developing a multitude of unmanned systems that it intends to rapidly field". [4] Additional importance is being placed on the space rovers used by NASA. These new rovers, especially the ones currently on Mars, must have an autonomous capability to help adapt to the terrain.

There are many designs today that all provide similar functions. This robot will be different than most because it combines many different components to make up the final design. As shown by the Stanford Cart robot, cameras have been used on robots for over 40 years. Other devices, such as the self-cleaning sweeper use range finders to detect objects before it runs into them. This project is unique because it combines real time video tracking via Wi-Fi, data transfer with Bluetooth, and range detection using ultrasonic sensors.

CHAPTER 2

SELECTING A DESIGN APPROACH

DESIGN SPECIFICATIONS

As stated before, the design team specified the object to be tracked as a brightly colored ball. The robot must be designed to operate in a normal indoor environment, at room temperature. The robot can be manually guided to the target area, although it must be wirelessly controlled. Once the robot is within three feet of the target, it must be capable of having 360 degrees of rotation to autonomously find the object using a wireless camera. This range of motion requires that the robot has a zero degree turn radius. Once the target is acquired, the robot must move to a maximum distance of three feet from the object, and remain within that range for the duration of the time. Run time must elapse at least one hour, at a speed of five miles per hour. Additionally, sensors must be implemented to prevent collision of the robot and the target, and other surrounding objects.

A microcontroller must be used to wirelessly communicate and relay information back to a base computer, as well as provide complete control over the speed and motion of the robot. This information must be saved automatically into a database. A manual shut off switch must be hard wired into the robot to cut off power to all components for safety. Additional features could be added to the design that improves upon the requirements. The final delivery date for this design is April 26, 2013.

FAST DIAGRAM

The Function Analysis System Technique, or FAST Diagram, is an effective method used to determine the essential tasks a system is required to accomplish. Reading the FAST diagram shown in **Figure 1** from left to right and top to bottom shows how the relationship is related to the task. Reading from right to left and bottom to top indicates why the relationship is related to the task. By using this method, it was much easier to create conceptual designs, and visualize how components would interact with one another.

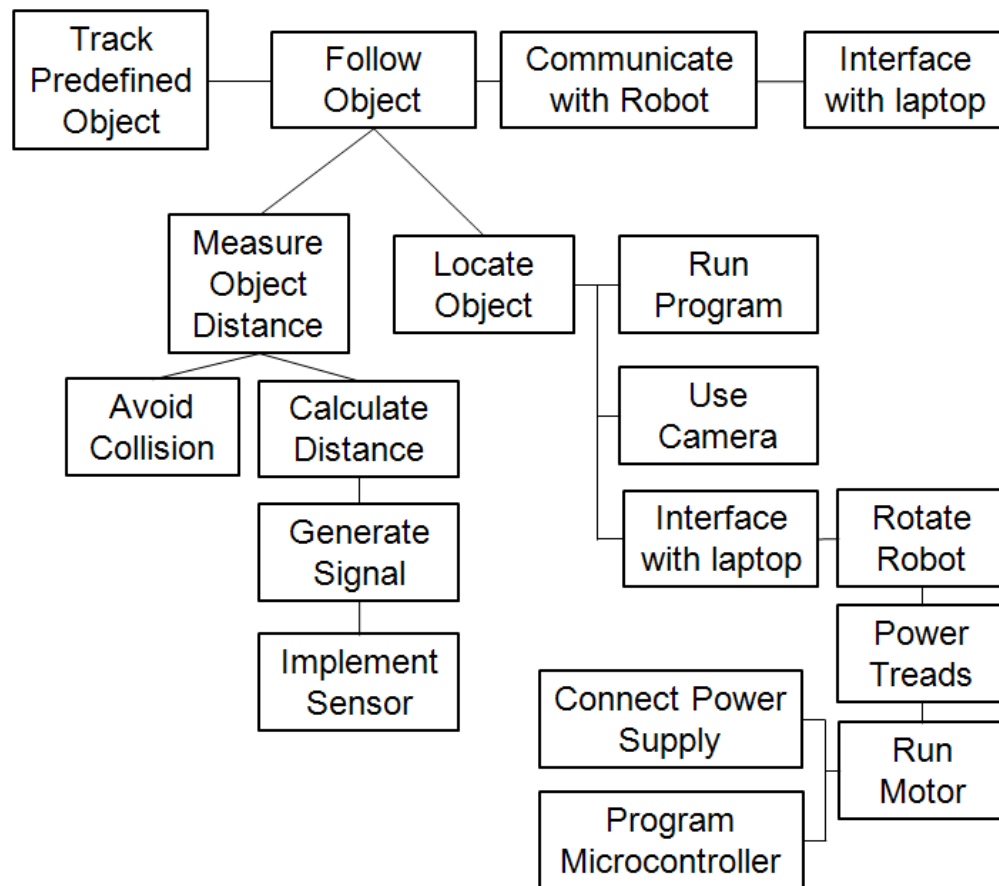


Figure 1: Function Analysis System Technique (FAST) Diagram

CONCEPTUAL DESIGNS

There were many factors that contributed to the first conceptual designs for the autonomous target tracking robot. The appearance and maneuverability were large factors that greatly influenced the initial designs, as well as cost. The three initial prototype designs were a quad-copter, a tank, and a four wheeled, independent axis remote control car.

QUAD-COPTER

The quad-copter design would have an appealing looking body, and would be a type of tracking robot that has not been explored in detail. This design would give the best maneuverability, but it would also have the highest cost. The idea was scrapped mainly for the reason that the runtime has to be at least one hour and this would be difficult to achieve given the design constraints. The design team determined that running the motors continuously for one hour on the quad-copter would require a new battery, which when added with the motor shield and microcontroller would have made the robot too heavy to operate effectively. An additional drawback is the difficulty in mapping out a three dimensional representations of the area below the robot compared to the other design concepts.

FOUR WHEEL INDEPENDENT AXIS REMOTE CONTROL CAR

The next design option was the remote control car with four independent wheels. This would allow for each side of wheels to run on one motor, allowing for the zero degree turning radius to be achieved. It would also have the capabilities of speed that other options may not. This design would require modifying a current remote control car, which may be difficult because most current models run with the front or rear wheels connected by a single axis.

TANK

The tank style design was chosen as the final prototype after weighing the appearance, maneuverability, ease of use, and cost. What the design team liked most is that the tank style design already has a zero degree turn radius built into its design. There are also many preexisting designs that the team could modify so that it would operate in the desired way. Although the treads may reduce the overall speed, this design requirement was deemed to be the least vital.

CONTROL

The two options that were researched as ways to manually control the robot were a smart phone and a joystick videogame controller. The team feels that the better option is a videogame controller which allows for easier movement. However, later on in the semester, it was decided that the laptop will provide the manual control for the robot. This was chosen for simplicity, due to the fact that an external command station/laptop is already needed. The established communication from the microcontroller to the base station was discussed to be either Bluetooth or Wi-Fi. After researching both methods, Bluetooth was chosen for its speed and reliability. Additional reasons why Bluetooth is implemented is for its compatibility with laptop connections, as well as the range capabilities.

TRACKING

The portion of the design created to meet the tracking requirements was the most crucial for the project, and many different options were explored. It is stated in the design requirements that a camera must be included in the project, so that was something the design team had to take into consideration. A webcam is utilized to visually track the predetermined object by recognizing its color and shape by using the Open CV library. Although radar is capable of detecting objects, the team felt that it would be more efficient to install and use ultrasonic range finders because of their wide detection angle. These sensors are used to measure the distance of the tracked object, and avoid collisions with obstacles in the surroundings.

SOLUTION

One helpful method in determining the best prototype design is the Critical Customer Requirements Matrix. This can be seen in **Table 1A of Appendix 3**. This table compares all of the designs and possible design solutions. These options are then graded from one to five on order of importance, and these values are multiplied by a weighted factor. Once the table is filled out, all of the values are added together and each design solution is given a final grade. The component with the highest score is most commonly the suggested design solution. By using this method, the design team was able to effectively compare and contrast many different options to finally decide on the overall design.

The design team chose to go with a tank style robot that is created by modifying a current remote control tank that is on the market. Initially the method of control was chosen to be a joystick controller, which was later changed to using the keyboard on a PC client. Wi-Fi and Bluetooth were both chosen as methods of communication, and ultrasonic sensors were chosen for their accurate distance measurements. Buying a premade vehicle restricts the team's flexibility and creativity in the design. However, building from the ground would add unwanted complications to the project. Therefore, a mix of buying and building would allow for maximum creativity and simplicity.

The final design of the robot to be built was determined using the decision matrix, which can be seen in **Table 2A in Appendix 3**. The obvious goal of any design project is to meet all of the customer requirements. However, some aspects are more crucial and are of higher importance. Using this matrix allowed the design team to rank the most important design requirements by seeing how essential each component is to fulfilling each prototype specification.

PROJECT SCHEDULE

This semester, a Gantt chart was utilized to assist in the creation of a project timeline. This timeline was created to allow the design team to track their progress, and to ensure that all deadlines are met and the final prototype is delivered on time. This semester, each team member was assigned both a technical and non-technical role, and this chart shows what jobs were supposed to be completed by which individuals. The Gantt chart begins on the week of January 28, and breaks the semester down into six distinct time intervals. Table 3 below breaks down these six time intervals and gives a brief explanation as to what was accomplished during that period. See **Table 3A & Figures 1A & 2A in Appendix 3** for a detailed view of the Gantt chart.

1/28-2/14	Gather data, research conceptual designs & possible parts
2/14-3/1	Order parts & begin prototyping upon arrival
3/11-3/22	Build prototype, & begin testing/troubleshooting
4/1-4/5	Testing & Troubleshooting
4/8-4/12	Final design adjustments
4/15-4/25	Prepare for design day
4/26	Design Day – Prototype delivery

Table 1: Gantt Chart Summary

PROJECT BUDGET

As was stated earlier, Design team 7 began the semester with a \$500 budget, which had to cover all electrical, mechanical, and software components for the final prototype. After researching possible devices that could be used to meet the design requirements, a preliminary budget was created. **Table 2** shows a detailed analysis of the initial budget estimate. It should be noted that this budget did change over time, which will be explained later in Chapter 5.

Projected Budget	
Part	Cost
Robot Body	
Tamiya Double Gearbox	\$14.48
Tamiya Track and Wheel Set	\$7.84
Tamiya Universal Plate	\$5.33
Data Acquisition	
Wireless Wi-Fi Enabled Webcam	\$58.00
4x Ultrasonic Range Finder	\$118.00
Microcontroller and Electronics	
Arduino Uno Board	\$20.95
Motor shield for Arduino	\$24.99
Sharp Infrared Sensor	\$14.95
DFRobot Serial Bluetooth Module	\$21.90
Li-Ion 18650 Battery + Charger	\$69.90
Total	\$328.69

Table 2: Projected Prototype Design Budget Breakdown

CHAPTER 3

HARDWARE DESIGN EFFORTS

ARDUINO UNO

The microcontroller chosen for the project is the Arduino UNO, shown in **Figure 2**. It has 14 digital input/output pins, where 6 can be used as pulse width modulation outputs, 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an in-circuit serial programming header, and a reset button. It can easily be powered by connecting it to a computer with a USB cable, attaching it to an AC-to-DC adapter, or to a battery. This Arduino features an 8-bit microcontroller Atmega16U2 programmed as a USB-to-serial converter. This revision of the UNO board has a resistor that pulls the 8U2 HWB line to ground, making it easier to put into the DFU mode (Device Firmware Update). The operating voltage is 5V while the recommended input voltage is 7V to 12V with limits of 6V and 20V. The DC current per I/O Pin is 40mA and while the DC current for 3.3V Pin is 50 mA. Additionally the Flash memory in the ATmega328 is 32KB where 0.5KB is used by the boot loader. The SRAM in the controller is 2KB while the EEPROM provides 1KB. Lastly the clock speed is 16MHz. [5] Please see **Appendix 3 Figure 3A** for a detailed view of the electrical components of the Arduino UNO.

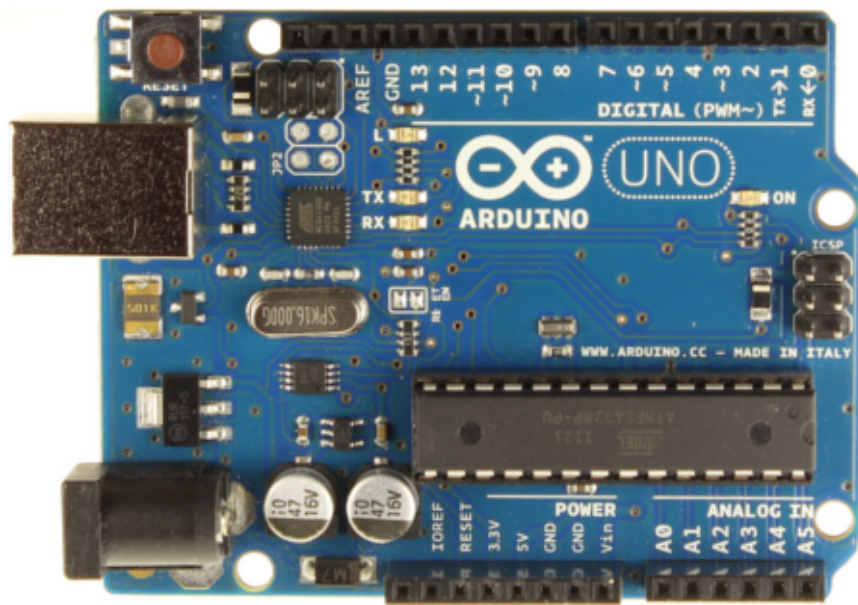


Figure 2: Arduino UNO Microcontroller [5]

DFRDUINO MOTOR SHIELD

The motor shield used is a DFRobot Arduino Compatible Motor Shield that uses an L298P chip to drive two 7V to 12V DC motors with a maximum 2A of current. See **Figure 3**. Speed control is achieved through a conventional PWM found in the Arduino's output pins 5 and 6. Digital pins 4 and 7 are used as an enabling and disabling function for the motor control. The motor shield can be powered directly from the Arduino or from an external power source. It is strongly recommended that an external power supply of at least 7V is used to power the motor shield. The logic control voltage from the Arduino is 5V while the motor driven voltage, from an external power source, is 6.5V to 12V. This is a two-way motor drive with a logic supply current of less than 36mA and a motor driven current of less than 2A. This is important because the motors to be used must have an operating current of under 2A, or they will not run. The size of the board is 55x55mm, which is nearly the same size as the UNO. [6] Please see **Appendix 3 Figure 4A** for a detailed view of the electrical components of the DFRduino 2A Motor Shield.



Figure 3: DFRduino 2A Motor Shield [6]

DEVANTECH SRF05 ULTRASONIC SENSORS

One of the most crucial aspects of this project is the detection of not only the marked target, but other objects in the surroundings as well. Because of the wide area that an object can be in, it is important for the prototype to have a range finding sensor that has a wide detection array as well. This is the reason that the design team chose to use ultrasonic sensors. The SRF05 model is manufactured to have a detection angle of at least 45 degrees. This provides a wide enough angle that when two sensors are placed on the front of the robot, any object in front of it should be detected. The beam pattern for an SRF05 sensor is shown in **Figure 4. [7]**

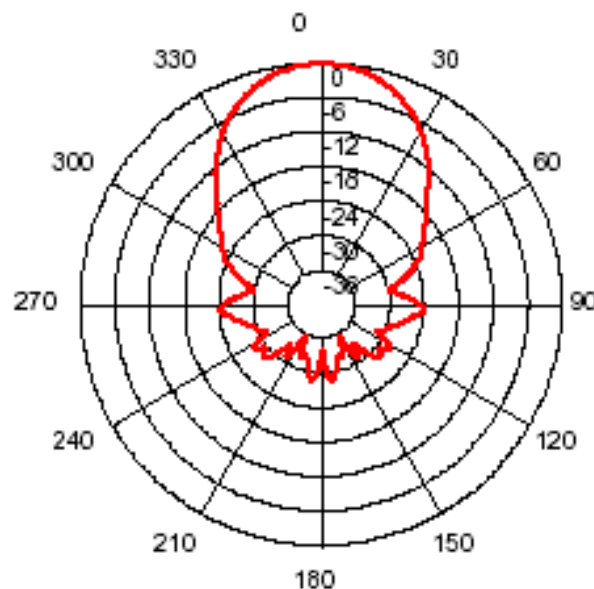
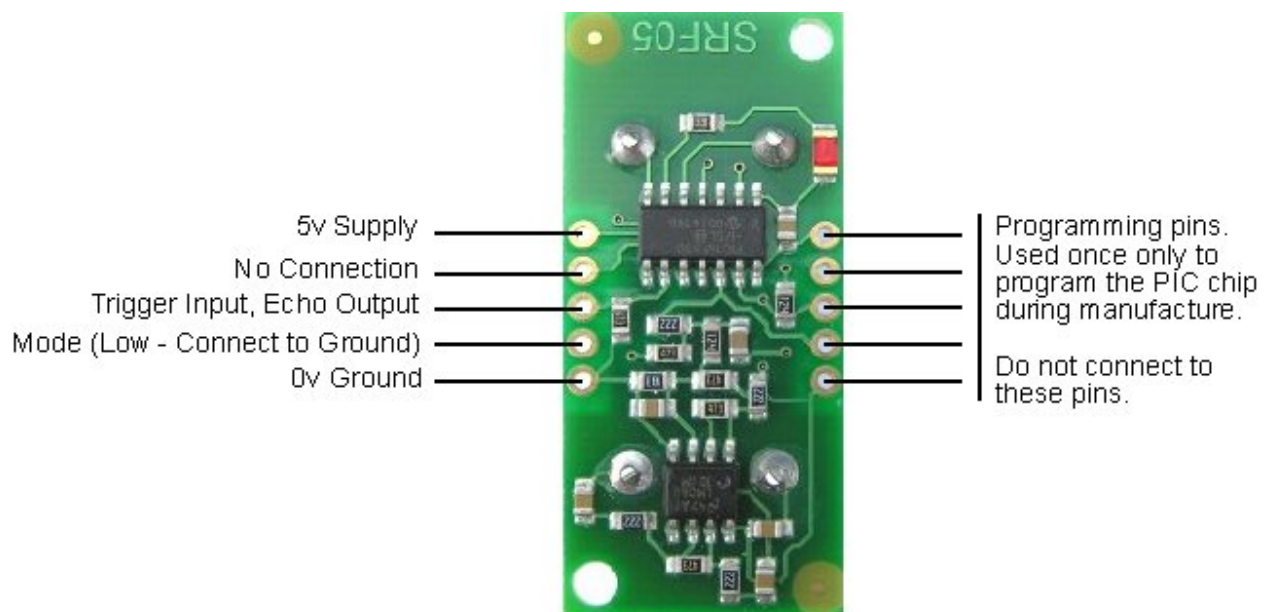


Figure 4: Beam Pattern for SRF05 Ultrasonic Sensor [7]

There are a variety of these types of sensors on the market, and each performs similar functions. The major difference however is the detection range. For this project, the goal is for the robot to stay within three feet of the target, so a sensor with a range longer than this distance is needed. While doing research on the topic of ultrasonic sensors, it was found that the Devantech SRF05 Ultrasonic Sensor is often used with Arduino microcontrollers. This is important because it let the design team know that the two devices are compatible with each other without having to find out by trial and error. [7] The SRF05 sensor is rated for detection range of three centimeters to four meters, which fits well with the design requirements.

For ultrasonic sensors, the user triggers a sonic pulse that is sent out, and an echo pin measures the time until the signal returns from bouncing off of an object. This time measurement is then converted into a physical length. There are 10 total pins on an SRF05 sensor, where five are programming pins used by the manufacturer, and five are used by the customer. There are two modes of operation for the SRF05 sensors, and each has the exact same function. It is simply up to the user to decide which one works best for their design. The only difference between the two is that in Operation Mode 1, the trigger and echo pins are separate. See **Appendix 3 Figures 5A & 6A** for more information concerning the Operating Mode 1. In Operating Mode 2, shown in **Figure 5**, the echo and trigger pins are combined, so that only one I/O pin is used on the microcontroller. Since the Arduino UNO only has 14 I/O pins, the design team decided that using Mode 2 to operate the sensors would be more efficient. It also ensures that there are enough I/O pins left for other devices in the prototype.



Connections for single pin Trigger/Echo Mode

Figure 5: Devantech SRF05 Ultrasonic Sensor operation Mode 2 [7]

In **Figure 6**, the timing diagram for Operation Mode 2 is shown. When it is examined, it is shown that the trigger and echo pulses are on the same line, which is an indication that they both are seen at the same pin. The second line shows the 8 cycle, 10 μ s sonic pulse that is sent out by the sensor. The pulse sent out is at a frequency of 40kHz, which is outside of the human hearing range of 20 to 20kHz. If nothing is detected after 30ms, the sensor defaults a return value of the maximum 400cm. Due to this, it is suggested that when using multiple sensors, a 50ms delay be placed between each trigger, so that the ultrasonic beep has faded away and will not cause an incorrect reading on the next measurement. [7]

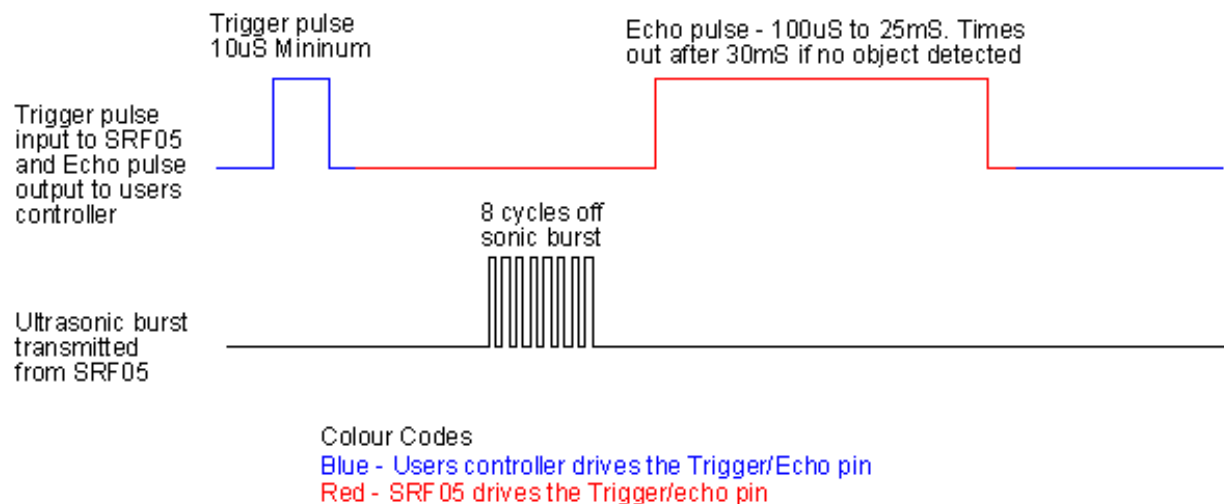


Figure 6: SRF05 Mode 2 Timing Diagram [7]

BLUETOOTH MATE SILVER

The design team initially decided to go with an alternative, less expensive Bluetooth module. Upon further testing and research when it arrived, the team realized that there are known communication issues between that model and the Arduino UNO. Due to this, the team immediately looked to an alternative model so as to not waste valuable design time. The model that was found to work well with the UNO was the Bluetooth Mate Silver. See **Figure 7**. It is more expensive than the previous option, but it is known to work well with the UNO. This Bluetooth has a maximum serial stream of 115200 bits per second, which is sufficient for transmitting continuously updating data. It is capable of running off of 3.3V to 6V, which works well with this design because the UNO has a 5V power output pin. The small size is also great this prototype because it will not take up a lot of space inside of the tank. Additionally, it has a really low power consumption of only 25mA, which is important to preserving the lifetime of the battery. The antenna is built into the device itself, which means that there is nothing that sticks off of the board that could potentially break off. Finally, this device is unique because it is able to operate in electronically noisy environments, such as areas with multiple Wi-Fi connections. [8]

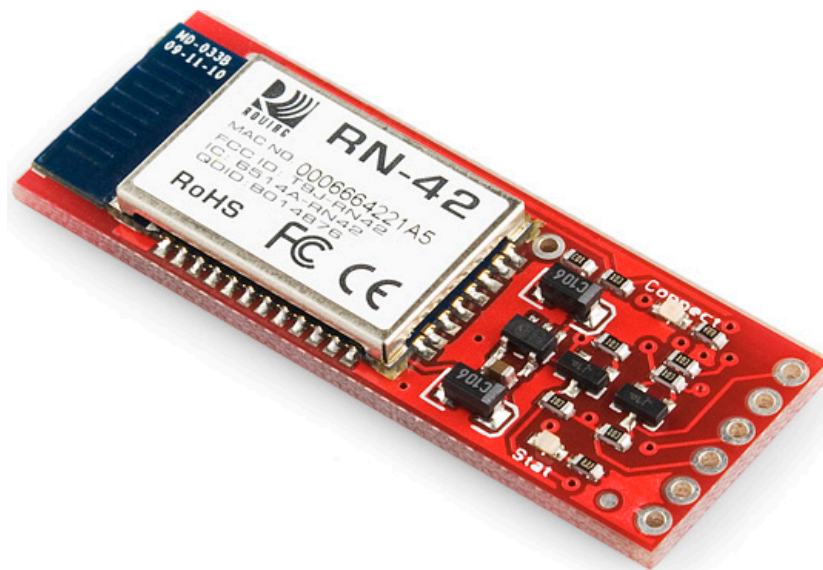


Figure 7: Bluetooth Mate Silver [8]

LI-ION BATTERY CHARGER

The tracking robot developed operates off of two battery packs, one strictly for running the Arduino and the other to supply power for the motors and webcam. Nickel cadmium (NiCad) batteries, supplied with our purchase of the tank body, are rated at 7.2V and 1700mAh. This battery has enough voltage to run the microprocessor, but does not have a large enough capacity to run the rest of the hardware. The lithium ion battery pack purchased, shown in **Figure 8**, holds 6600mAh and is nominally rated at 7.4V, which can operate the tracking robot for significantly longer. The lithium ion battery came with protection circuitry which limits overcharging and over-discharging. It can supply a maximum current of 1.75A, which gives more than enough power to the motors for the required speed. The lithium ion battery pack however, did not come with a charger. To add a level of technicality to the project, a custom charger circuit was designed for quick and safe charging.



Figure 8: 6600mAh Li-Ion Battery

When designing the charger, the first consideration was the size and rating of the battery pack. The voltage is the main concern, but the C rating and capacity is very important to determine the max charge rate which can be safely applied to the battery. The battery pack has a nominal voltage of 7.4V and a 6600mAh capacity, the C rating defines the maximum charge and discharge of the pack. The data sheet gives a maximum discharge current as .75C, and a maximum charge current as .265C. This means the battery pack can supply up to 5 amps of current to the application it is being used for, while the maximum charge current is 1.75A. [11]

When charging lithium ion batteries, there are two stages of charging: constant current and constant voltage. During the constant current stage, the maximum current allowed (or designed for) will flow into the battery as the voltage level increases to the rated voltage of the battery pack. Once the voltage is reached, current will continue to decrease as the capacity of the battery is filled; during this stage the voltage stays constant. See **Figure 9**.

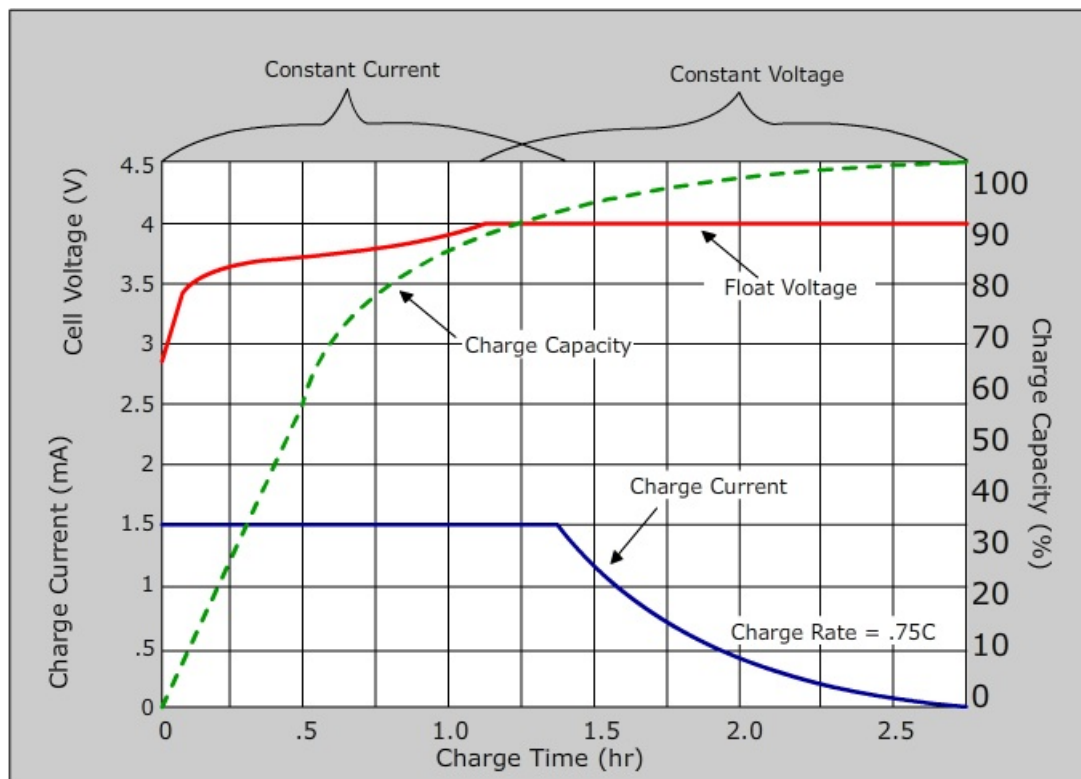


Figure 9: Charging Capacity

Some needed components for this battery pack charger include a transformer, bridge rectifier, voltage regulator and current regulator. For extra safety, a temperature sensor was included in the design, which required an extra voltage regulator for a reference voltage, as well as an indicator circuit, which lights up an LED when the battery pack is charged to approximately 80% of its full capacity. This was done because storing full capacity at high voltage creates stress on the batteries, which decreases longevity.

D-LINK WIRELESS ENABLED CAMERA

A drawback with using Bluetooth as a method of communication is that it does not stream real time video well, due to a low data transfer rate. As it is essential for the microcontroller and laptop to be evaluating current video data, other methods were needed to be explored for the camera to be effective. After researching this topic extensively, the design team was able to find a webcam that is wireless and transmits data over a Wi-Fi connection. It is the D-Link Wireless Enabled Webcam, shown in **Figure 10**. This will allow the device to operate as long as there is an internet connection nearby. If there is no internet available, a mobile router can be used which will provide the necessary connection. The camera runs on a 5V DC supply, which is important because it is within the available amount provided by the two batteries. [9]



Figure 10: D-Link Wireless Enabled Camera [9]

HARDWARE IMPLEMENTATION & PHOTO IDENTIFICATION

HARDWARE DESIGN DIAGRAM

The final step in the design process is mapping out how all of the electrical components will interact and work together. **Figure 11** below shows the design diagram for the hardware. It is important to remember that because the motor shield is stacked right on top of the UNO, all connections made to the UNO are actually made in the motor shield pins. In the following sections, the electrical connections of all components will be explained and shown. Detailed wire connections can be seen in **Figures 7A, 8A, & 9A** in **Appendix 3**.

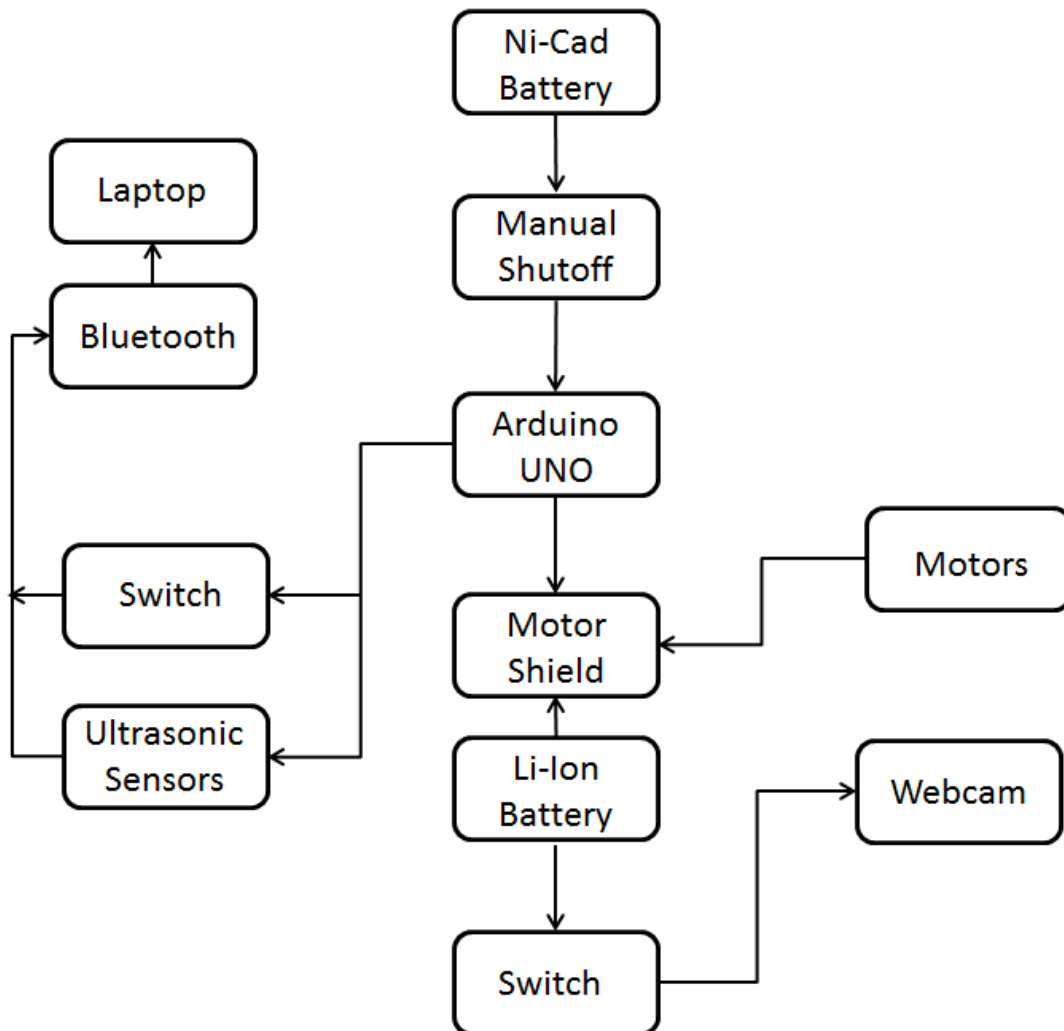


Figure 11: Hardware Design Diagram

ARDUINO UNO

The reason this microcontroller was chosen was for its Bluetooth compatibility, easy programmability, and available resources. Its Tx and Rx pins are made especially to operate with Bluetooth devices. Bluetooth technology allows the microcontroller to wirelessly communicate with the base station, which in this project is a laptop PC. The UNO also allows the team to connect a motor shield to control two DC motors found on the robot. A safety switch has been placed between the 7.2V Ni-Cad battery and the UNO that will cut power to the motors. The USB connection of the Arduino is neatly placed onto the rear of the robot for connection and power to a computer without having to take the robot apart. **See Figure 12.** This allows the program to be easily changed in situations where the environment or target may change.



Figure 12: PICTURE OF BACK OF TANK WITH USB PLUG

MOTOR SHIELD

As previously explained, and shown in **Figure 13**, the motor shield pins fit right into all of the pins of the UNO. The Arduino UNO is blue, while the motor shield is black. The motor shield pins 4, 5, 6, and 7 are occupied to control the speeds of the motors, so no other devices can be placed in those locations. Pins 9, 10, and 11 are used as the trigger/echo pins for each of the individual ultrasonic sensors. Pins 12 and 13 are connected to 100 Ω resistors that are in series with two parallel LED's, which are the headlights and taillights on the robot. The shield's positive power input is connected in parallel to a switch and the 7.4V, 6600mAh DC battery. The Tx and Rx pins are used for connecting the Bluetooth Mate Silver. Pins M2+ and M2- are directly connected to the positive and negative connections of one motor. Pins M1- and M2+ are connected to the other motor in the same fashion.

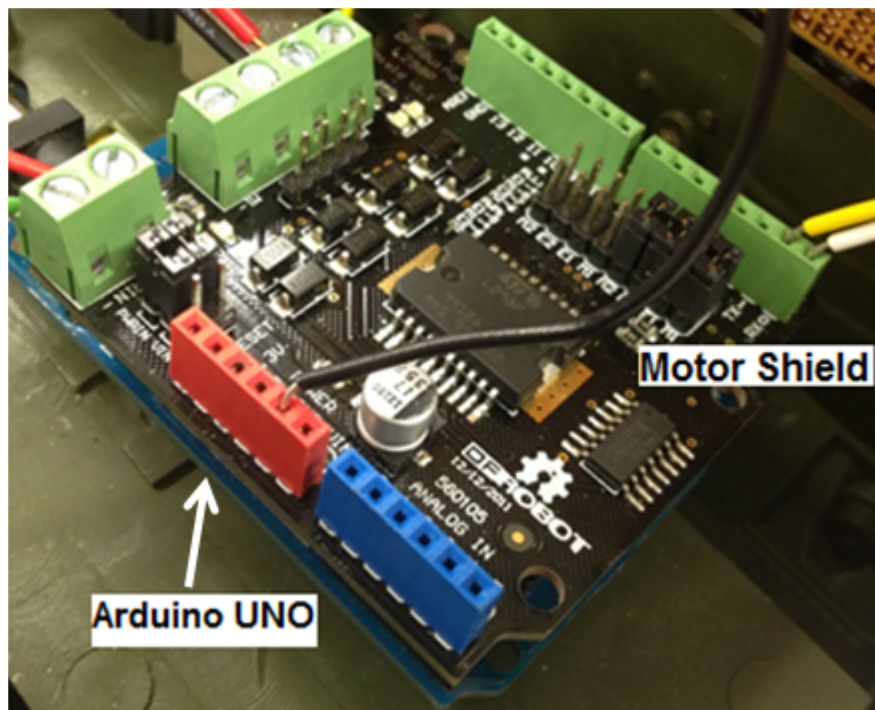


Figure 13: Motor shield stacked on top of Arduino

SRF05 ULTRASONIC SENSORS

As mentioned above, the trigger/echo pins of the sensors are specified as pins 9,10, and 11 of the Arduino. Each sensor runs off of a 5V power supply, so the team is able to connect all of these to the 5V output power pin on the UNO. For operating in Mode 2, the operating mode pins, and the ground pins of the sensors were connected in parallel. They were all then connected to the ground pin of the UNO. The full connection process is shown below in **Figure 14**.

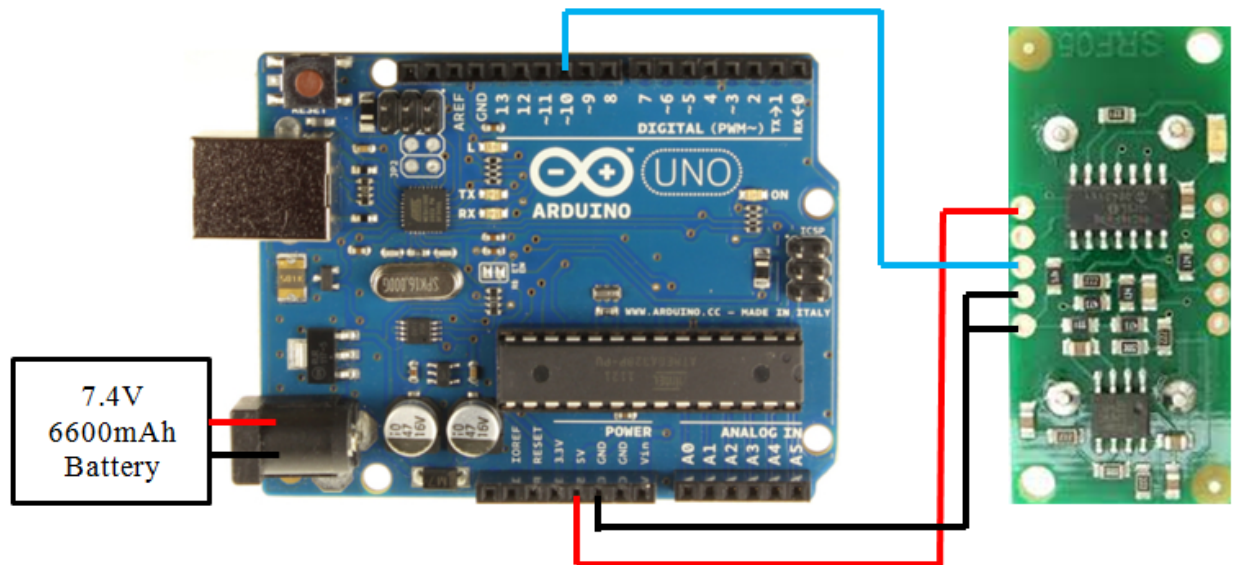


Figure 14: Example of Operating Mode 2 Wire Connections [5]

The design team decided to place two sensors at the front of the tank, and one sensor in the back. The two in front point slightly in, so that they are able to find the marked target together. The back sensor is primarily used to ensure that the robot does not run into anything while backing up. The original plan was to have two sensors in the back as well, but the design team decided that one sensor will provide the functionality needed. **Figure 15** shows an overhead view of the detection area of the robot that is created by the three ultrasonic sensors.



Figure 15: Overhead view of tank and detection angles

BLUETOOTH MATE SILVER

The Rx-1 input of the Bluetooth Mate Silver is directly connected to the Tx pin of the UNO, while the Rx pin is directly connected to the Tx-0 pin of the Bluetooth Mate Silver. It should be noted that the Tx and Rx connections are backwards. This is due to how the Bluetooth Mate is programmed. The power pin is connected directly to the 5V power output pin of the UNO, while the ground pin is connected to the UNO ground pin. **Figure 16** below shows the connections for the Bluetooth module.

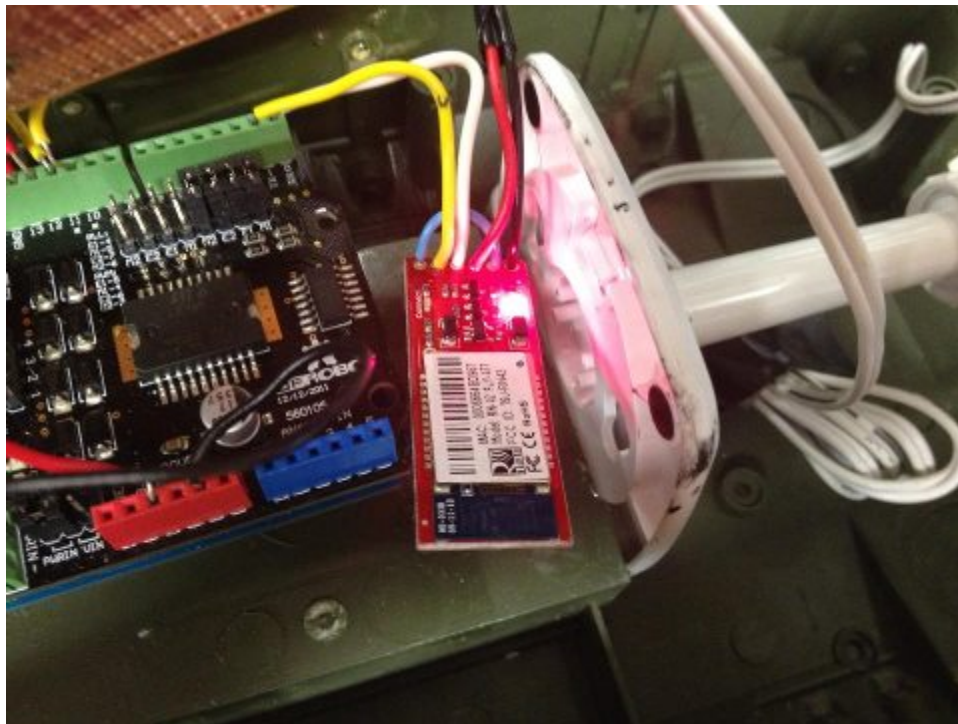


Figure 16: Wire Connections for the Bluetooth Mate Silver

LI-ION BATTERY CHARGER

A battery operates on a DC voltage, and since this design includes the use of a wall outlet, it requires the transformation from 120V AC, to a usable DC voltage. The regulators used in this design allow for 15V to be dropped across them without affecting the current limit, so a standard transformer was used to bring the wall outlet voltage down to 12V AC. See **Figure 17**.

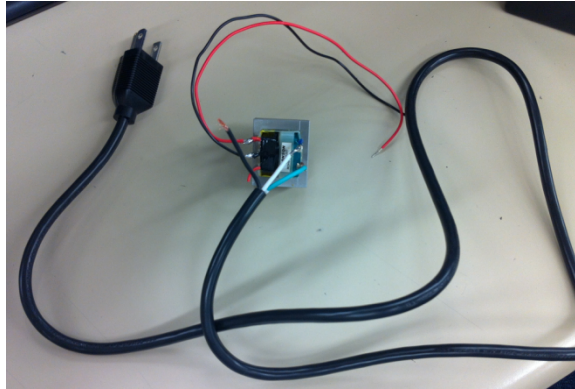
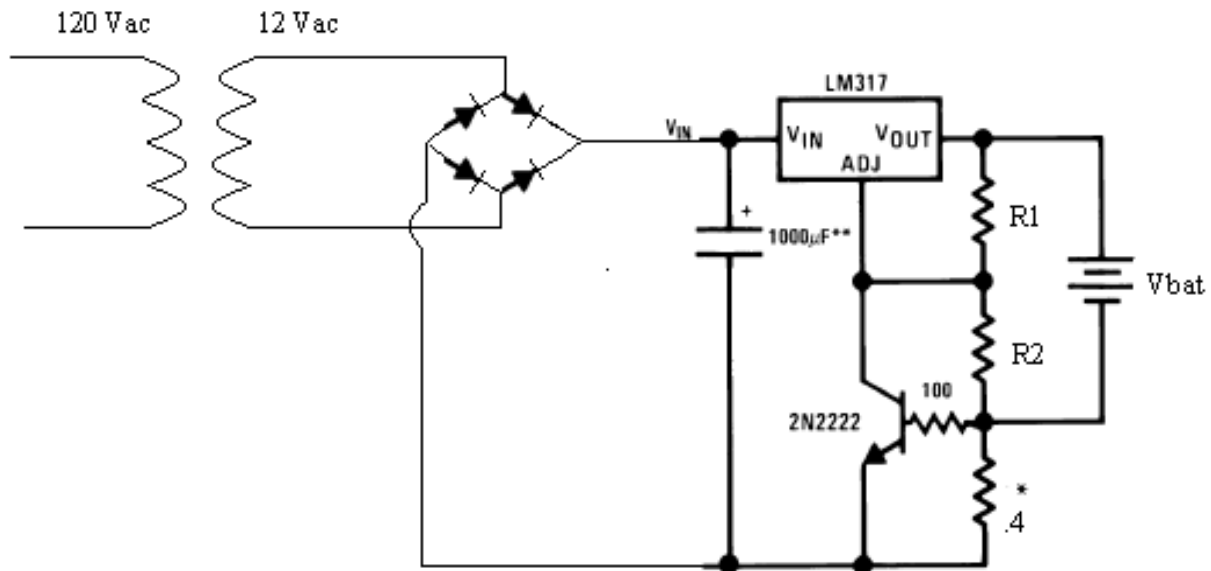


Figure 17: Transformer

Next, a bridge rectifier is used to acquire DC voltage, and a large capacitor is placed across the output to reduce the voltage ripple. The LM317 is a regulator that can be used to set a voltage or limit current. In the data sheet several examples are shown, one in particular shows its use as a battery charger. [13] Using this example with a few modifications, the team designed a custom battery charger with temperature protection. Setting the voltage desired to charge the battery pack is the first consideration to make. This depends on the values of the resistors R_1 and R_2 as shown in the schematic in **Figure 18**. The voltage of the battery, V_{bat} , can be used to solve for R_1 and R_2 .



*Sets peak current

**The 1000µF is recommended to filter out input transients

Figure 18: Battery Charger Schematic

Since the application of this battery pack is to power an RC car, a higher voltage is desirable. The battery charger is designed for 7.8V. Each lithium ion cell can be charged to between 3V and 4.2V.

Equation 1:
$$V_{bat} = 1.25 \left(1 + \frac{R_2}{R_1} \right) = 7.8$$

Solving **Equation 1** for R_1 and R_2 , the ratio of R_2/R_1 is 5.25.

The team chose $R_2=1600\ \Omega$ and $R_1=300\ \Omega$, which gives a ratio of 5.33. This will set the voltage across the battery pack as 7.9125V, which is still under the maximum rating of the battery pack.

The LM317 can provide currents above 2.2A, so if the battery pack is not rated for that amount, a current limiting resistor must be added from the negative side of the battery pack to ground. This charger was designed to limit current to 1.5A because of the .4 Ω resistor. This resistor will have a considerable amount of current flowing through it, and therefore must be able to handle a considerable power load. The resistor used is rated

for 5W, though a 3W power rating should suffice. A temperature sensor is placed near where the battery pack will be charging. By taking the output of the temperature sensor and comparing it to a set threshold voltage with a comparator, the team can use the output to control a BJT as a switch. This is needed to stop the charger if the battery pack gets too hot. See **Figure 19**.



Figure 19: Battery Pack and Charger with Temperature Sensor

The LM34 outputs the solution of 10mV multiplied by the temperature reading in degrees Fahrenheit. A room temperature at 70 degrees will give an output reading of 0.7V. The data sheet for the lithium ion battery pack gave temperature recommendations. Therefore, the charger is designed to shut off once a temperature of 137 degrees is reached. In order to have the temperature sensor compare the current and threshold temperatures, a DC voltage regulator was used to create the 1.37V reference, which serves as the positive input for the comparator. Again the LM317 is used and an example in the data sheet showed an adjustable regulator with improved ripple rejection. The output voltage of 1.37V is reached by setting $R_1=466\Omega$, and $R_2=55\Omega$. See **Figure 20** for the final battery charger schematic. [12]

D-LINK WIRELESS ENABLED CAMERA

The webcam functions at a DC voltage of 5V, with a 1.5A current. To achieve this desired current from the 7.4V Li-Ion battery, a voltage regulator had to be used. IN this case, the design team used a LM7805 regulator, and the connections are shown in **Figure 22**. It should also be noted that between the 7.4V battery and the input terminal of the regulator, the design team placed a simple throw switch. This allows the user to turn the webcam on and off when the robot is not in use, which will limit unneeded current draw from the battery. This is needed because the webcam is on a different battery than the 7.2 Ni-Cd battery that powers the UNO. The hope was that this would lengthen the battery life of the UNO battery.

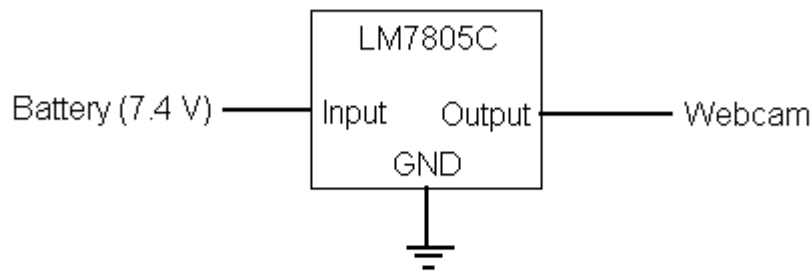


Figure 22: Webcam Voltage Regulator Schematic

SOFTWARE & INTERFACE DESIGN REQUIREMENTS

SOFTWARE DESIGN DIAGRAM

Now that the hardware has been installed, the next process was writing a program that will make the robot function to meet the design requirements. It is specified that the robot must be capable of two way communication with a base station to transmit and receive data and commands. To start this design process, the team took a similar approach to mapping out the hardware, by creating a software design diagram shown in **Figure 23**. This software will combine the use of both Wi-Fi and bluetooth communication, as well as transmitting, receiving, and saving data. The full **C# and Arduino source codes** can be found in **Appendix 3**.

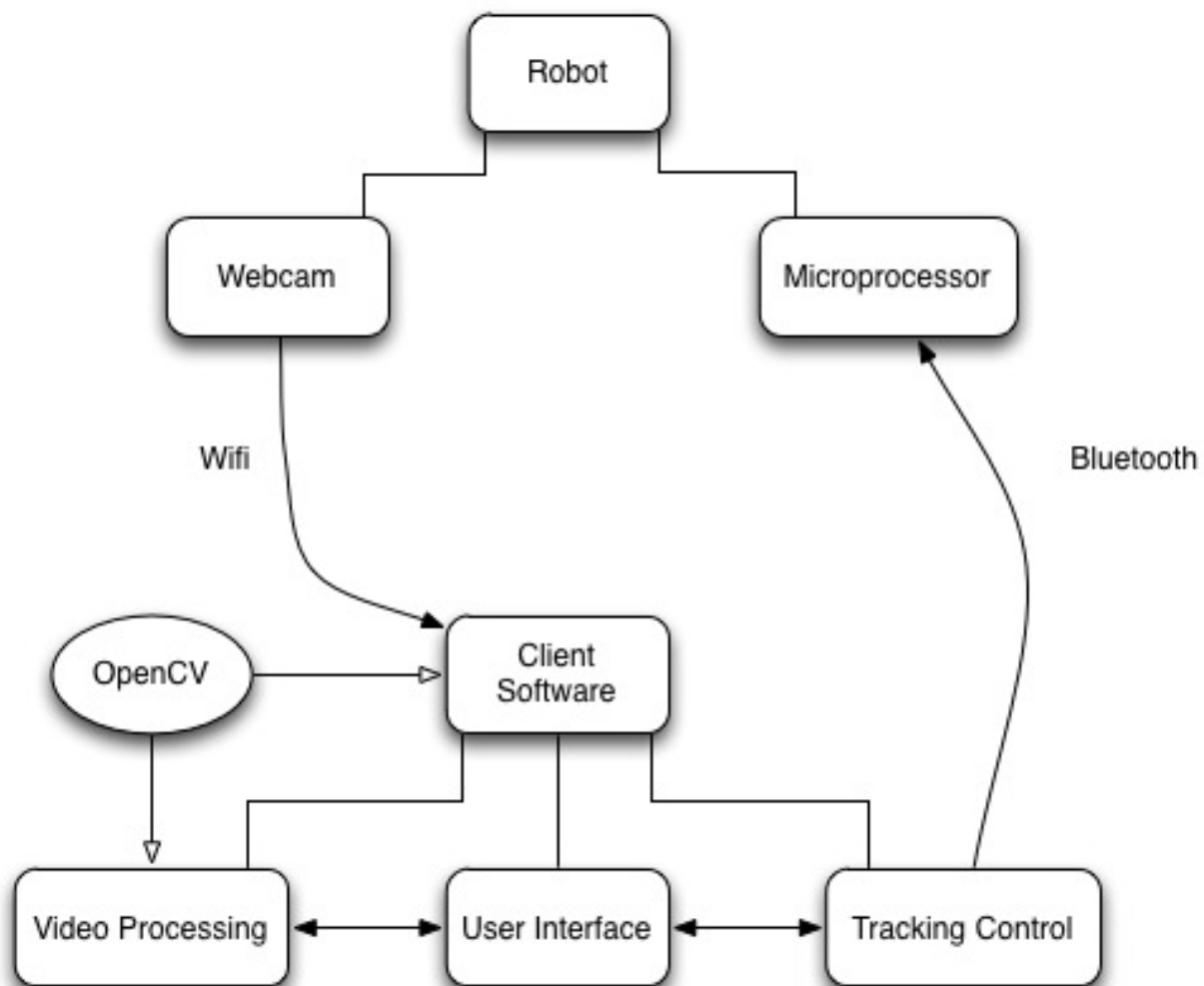


Figure 23: Software Design Diagram

SOFTWARE IMPLEMENTATION

C# CODE

The PC client software is designed in C# with Emgu CV, which is a .NET framework wrapper for Open CV library. The software provides a GUI for controlling and monitoring the robot's movements. The design consists 3 main parts.

The first part is the video processing component. In the GUI, it has two image boxes which will display both the unprocessed and processed real-time video from the robot. It also contains a text box that has the information of the tracking object's position and some editable fields to let user changes the RGB values for tracking color range and webcam URL. Underneath the GUI, some helper methods are defined in the code to process the incoming video. The method "processFrameAndUpdateGUI" processes the video stream by applying color filter layers. If the object's RGB color lies between the lower and higher limits of the RGB filter values, the object will show up as white in the image box labeled processed image. Once the object is detected, the method will draw a green dot in the center of the object and a red circle around the object. In the meantime, it will also update the text box to display the current position of the object. Upon acquiring the position of the object, if the robot is currently in autonomous mode, the method will check the object's relative position to the robot and send movement commands accordingly. If the object is in front of the robot, a "move forward" command will be sent. The method will not send a command if the current control mode is manual. To improve the response time of the robot and prevent sending duplicated commands, a string variable "previousCMD" is created to record the last command sent to the robot.

The second portion is the user control part which consists of several buttons for manual control and for changing the control mode. In manual mode, the user can use direction buttons to control the robot's movement and turn on and off the LEDs. The user can also change between the manual and autonomous modes using the corresponding button. As mentioned before, the image processing component will send control commands to the robot in autonomous mode.

The last component is the serial communication. It has two drop down combination boxes, a text box, start and stop buttons, and codes to support the operations. Users can use the combination boxes to choose the COM port and baud rate for serial communication. The team decided to implement a separate button to start serial communication in order to provide the user a “video only” mode. On event of pressing the start button, the component will establish the communication using the COM port and baud rate specified by the user. The text box will display the data received through the COM port. This component plays an important role in the whole design. It connects all parts of the software together and makes it possible to communicate with the robot wirelessly.

ARDUINO CODE

The purpose of the Arduino code is to enable communication between the microcontroller and a PC client via a Bluetooth modem wired to the Arduino and a Bluetooth adapter attached to the client’s USB. This allows the microcontroller to receive inputs related to manual or autonomous control as well as send information such as distance measurements back to the client. The code interprets any manual control inputs received wirelessly from the software GUI and executes movement by driving the motors accordingly. The code also allows the Arduino to take into account inputs received from the ultrasonic range sensors to avoid bumping into obstacles.

The code was built on several main ideas:

- Modularity
- Room for improvement
- Flexibility.
- Easily relatable to the schematic.

The first part of the code calls the `setup()` function, which will only run once after each power up or reset of the Arduino. It is then used to initialize relevant variables that store information such as sensor readings, and initialize pin numbers to allow the board to interface with the motors, sensors, and Bluetooth modem.

The motors are driven using pulse width modulation to adjust for speed. This is done by calling the function `analogWrite`, which generates a steady square wave of the specified duty cycle. The frequency of the PWM signal is 490 Hz. The values of the duty cycle range from 255, which is equivalent to always on, and a minimum of zero, always off. A value of 255 is initialized to drive the motors at maximum speed, and can be adjusted as needed.

The second part of the code is concerned with the `loop()` function. This function loops consecutively after the Arduino is done with `setup`, and allows for active control of the board. The loop first clears the ultrasonic sensors by sending them a LOW signal. The `pulseIn()` function is then called, which causes the sensors to send out a pulse. The `pulseIn()` function then reads the bounce back time. This is done at intervals of 200ms between each sensor, because the Arduino cannot read from sensors concurrently. The bounce back time is used to calculate the distance by using the speed of sound in relevant calculations. These values are used to move away from an obstacle if it is within a certain range. The information is stored and sent wirelessly to the client PC, which displays the readings on a GUI.

Manual movement is achieved when the loop checks if there is any data available to read from the serial port. If there is, this will be the user input entered in the client PC related to manual control, which moves the robot accordingly by controlling the left and right motors. Spinning them in the same direction causes the robot to move forward or backward. Spinning the right motor forward and the left motor backwards causes the robot to turn left, while the opposite causes it to move right. If the user decides to engage autonomous control, then the PC software takes care of the robot movement. The following pseudo code can effectively model what occurs in the Arduino microcontroller.

Setup pins, variables

Setup serial communication via Bluetooth

LOOP

- clear right sensor

- pulse right sensor

- calculate and store readings for right sensor

- send information to client

- wait 200ms

- repeat for left sensor*

- repeat for back sensor*

- IF data available to read from serial port

 - Interpret data and move accordingly

- IF sensor readings show object is too close

 - Move backwards

 - wait 200ms

 - stop

The final communication map can be seen below in **Figure 24**.

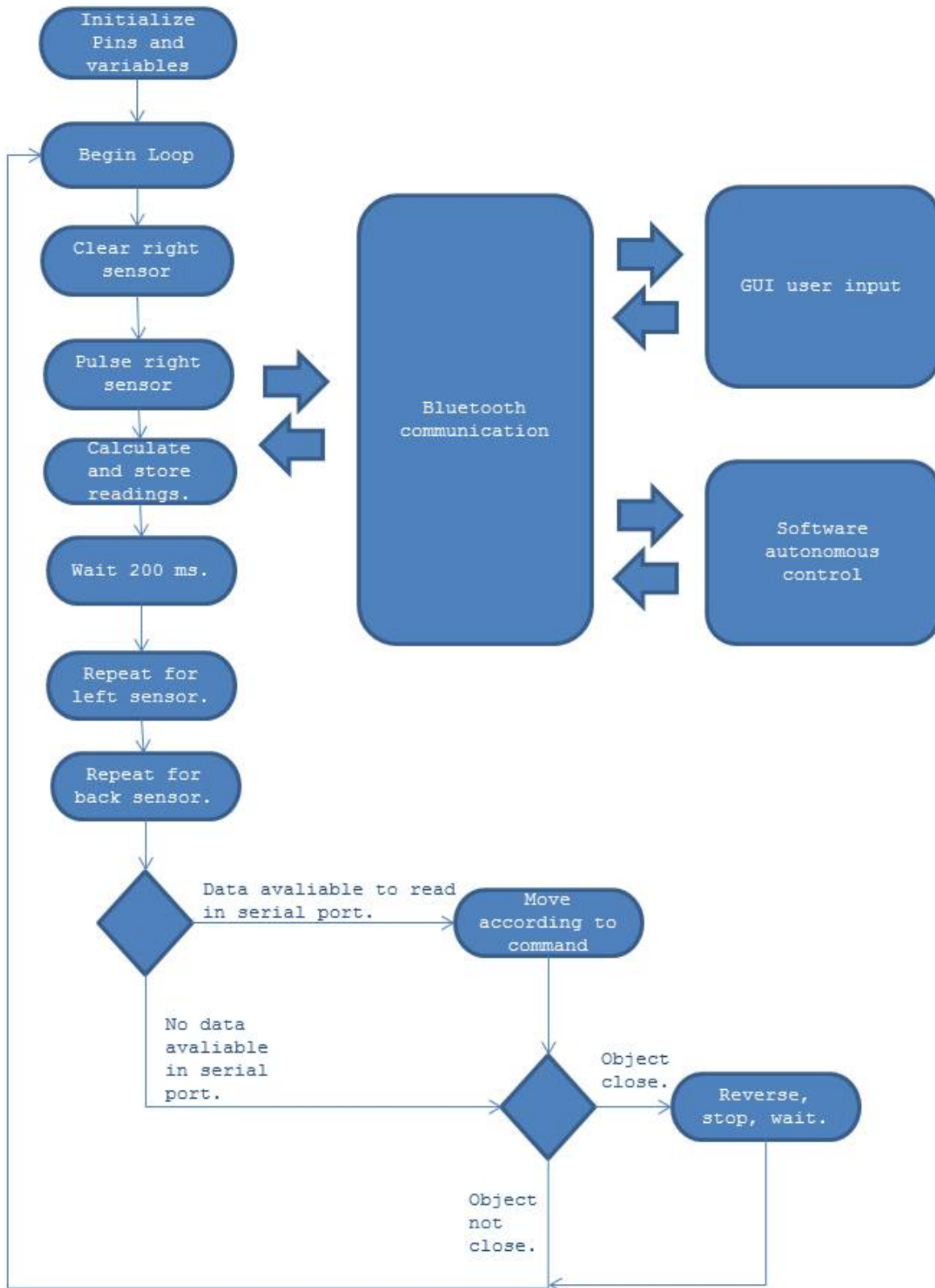


Figure 24: Loop of Communication

ISSUES ENCOUNTERED

There were several issues encountered with the designing, building, and programming of this project. The initial issue was the size of the first prototype ordered, which is shown in **Figure 10A in Appendix 3**. When the robot arrived, the team was skeptical about its ability to hold all of the components that would later be installed. The team attempted to make room by shortening and zip tying the wires, and decreasing the free space between components. In the end, the team saw that it would be impossible to fit all of the necessary hardware onto this style of robot. The solution was to order a new, and larger sized mobile tank. This tank was large enough to store all of the electrical hardware and more if needed.

Another issue was discovered after the Bluetooth Transceiver was ordered. The hardware that was to be attached to the UNO and Motor Shield had a lot of trouble connecting with the Bluetooth Dongle attached to the laptop. Additionally there was trouble connecting the laptop to the transceiver. After doing further research, it was discovered that this specific device was incompatible with the UNO. When more research was done, a compatible device called the Bluetooth Mate Silver was discovered. Once the part arrived the connection was made almost immediately after installation.

The Open CV software tracking program created another issue for the project. The program will sometimes track multiple objects. This is an issue because if the robot detects more than one object it can become confused and the tank will attempt to track down multiple targets at the same time. The issue cannot be resolved through the current programming process. Therefore, this would be a feature to be improved on in future designs.

Two weeks before design day the team encountered an additional issue. One of the motors that drive the robot treads had burned out. This may have been from not effectively limiting the current sent to the motor. Unfortunately the motor came with the tank ordered earlier in the semester, and the size and number of teeth for the gears was not mentioned in the specifications. After measuring the length of the shaft, as well as the number of teeth on the motor head, the team was able to order a replacement motor. This new motor to be ordered online had to have the same voltage and current specifications of the previous motors.

CHAPTER 4

Testing was divided into three main sections:

- 1) Testing Arduino functionalities.
- 2) Testing software functionalities.
- 3) Testing the charger.

ARDUINO FUNCTIONALITY TESTING

Phase 1: Testing proper Bluetooth communication to and from the client PC. This was done by writing simple code that would wireless turn an LED on or off.

Phase 2: Test if Arduino can store sensor readings, send the values to a client PC, and respond by moving accordingly away from the obstacle. This was done by adding a form in the GUI that would display sensor readings, and matching the readings with the movement of the robot.

Phase 3: Test if Arduino can drive the correct motors in response to Manual Control mode input. This was done by sending input from the client software wirelessly and observing if the Arduino responded properly to the input commands.

SOFTWARE TESTING

The software was then tested after all Arduino functionalities were established. The testing was done in 3 phases, and was mainly related to the three iterations the code went through:

Phase 1: Test if software can detect an object with a certain color through the default camera in the client PC. This was difficult as the software would behave erratically and would not always detect the presence of an object. This was rectified by allowing a larger range of colors to be allowed through the color filter layer.

Phase 2: Added position functionality, and tested to see if software would return correct object position values and would display correct positional coordinates.

Phase 3: In this iteration, the wireless camera was mounted to the tank body. Testing for autonomous control was done during this phase by testing to see if the robot would follow an object as it moved in front of the wireless camera.

Testing proceeded smoothly and was successful for all code iterations.

CHAPTER 5

FINAL COSTS

As was stated previously, Design Team 7 was granted a budget of \$500 for this project. Below, in **Table 3**, a detailed analysis of the total cost is shown. The final prototype design cost came in at \$386.40. However, the design team also had a previous prototype design that has a total cost of \$53.36. These costs, plus additional spare motors and shipping brought the semester's total cost to be \$561.81. This project did put the design team slightly over budget. However, the final prototype design cost is still well under \$500.

Part	Cost
Wireless Wi-Fi Enabled Webcam	\$58.00
3x Ultrasonic Range Finder	\$88.50
Arduino Uno Board	\$21.95
Motor shield for Arduino	\$21.95
Mini Bluetooth Dongle	\$12.99
RC Snow Leopard Tank	\$89.31
USB to TTL Converter	\$7.95
Tenergy Li-Ion 18650 battery	\$35.49
Bluetooth Mate Silver	\$39.95
2x DC motor	\$9.50
Final Design Total	\$386.40
First Pass Design Parts	
Tamiya Universal Plate	\$4.49
Tamiya double gearbox	\$14.48
Tamiya Track and Wheel Set	\$12.49
DFRobot serial Bluetooth module	\$21.90
Total for first pass design	\$53.36
Spare Parts	
2x Replacement DC Motors	\$9.50
1x Replacement Range Finder	
Shipping	\$83.86
Total	\$561.81

Table 3: Final Budget Analysis

SCHEDULE

Every week, the design team had a meeting with the facilitator, Dr. Ren, to update him on the progression of the project, as well as to do a check of the Gantt chart. Throughout the semester, the team was able to stay on track with the target timeline, which was important in the completion of the project. Having the weekly meetings with the facilitator forced the team to work hard to keep on schedule, and held everyone accountable. The team was not only accountable together, but individually as well, as each person strove to complete their technical portion of the project. Being on time with the schedule also allowed the team to be able to do more work on the final presentation and poster, and not spend the last week struggling to complete tasks.

For more information on the Gantt chart, please see **Table 3A**, and **Figures 1A** and **2A** in **Appendix 3**.

SUMMARY AND CONCLUSIONS

Overall, this project was a success even though there were many challenges and trials that the group faced in order for its completion. Throughout the semester there were issues with the components ordered and designed, as well as damage to devices towards the end of the semester. Additional programming issues were discovered and dealt with concerning the Open CV library. However, the team was able to successfully create a prototype with a similar design and functions as originally planned. The team successfully designed and built an autonomous target tracking robot.

In the beginning of the semester a land terrain vehicle was chosen rather than the aeronautical design for the robot due to battery lifetime constraints. Once the type of mobile robot was chosen the team ordered and worked on an existing tank style robot. After additional work was performed it was discovered that the design was too small to contain all the hardware devices needed. For this reason, the team ordered a larger and more traditional style remote control tank. The Bluetooth transceiver ordered also appeared to be a problem after attempting to communicate with the Arduino UNO and the USB Dongle. To fix this, more research was performed and the team discovered there is a more compatible transceiver available. The tracking program also developed

issues due to its inability to limit the number of targets acquired. Finally, during the week before design day, one of the robot's motors burned out. Another pair of motors was ordered and placing a resistor in series solved this problem by limiting the current. Despite all of the issues the team was able to successfully create a functioning robot to track and follow a predetermined object and avoid obstacles along the way. The team established a means of communication between the base station (laptop) and robot as expected.

Although this project is considered a success, improvements can always be made. In the future it is recommended to work on creating a mobile form base station such as an Android phone or iPhone. This would allow for a more user friendly platform for communication, as well as making the system more mobile. Also, the tracking could be improved to sense more difficult objects as well as limit the number of objects detected. Additionally, the body could be improved for stronger terrain and weather conditions. This current design is limited to indoor use on a flat plane, which limits its real life applications. Simple cosmetic changes could make this prototype more marketable. Having a tank body that built specifically for this robot would make it more appealing to customers. Finally, additional coding could be done to clean up the overall tracking process of the robot.

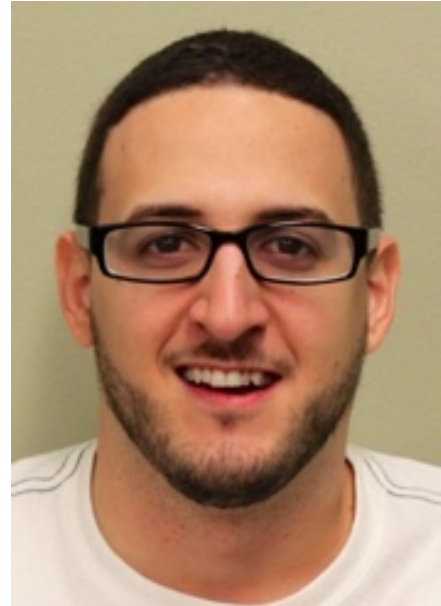
This project served as a great learning experience for the members of Design Team 7. The team learned how to design a product from beginning to end: starting with brainstorming and creating a first pass design, to improving this design, and finally to building a working prototype that could be viable on the open market. The knowledge gained from the experiences throughout the semester will be used to help the team members grow in their future careers as professional engineers from Michigan State University.

Appendix 1

TECHNICAL ROLES, RESPONSIBILITIES, WORK ACCOMPLISHED

Victor Abreu – Project Manager

Victor's technical role was as a Testing Engineer where he would check results of the microcontroller's communication to the motors, lights, and other electrical components of the mobile vehicle. Upon checking the results, he would provide solutions to any issues that may arise to complete the project in time. Throughout the semester his role merged with other roles in order to move forward with the project and keep the team on schedule. Victor created a circuit schematic and installed and soldered LED and resistor components in the robot to make headlights and taillights. The headlights are meant to help the camera's view of the tracked object's color, and to give the mobile robot a more appealing look. He also helped install the Arduino UNO software and hardware onto the PC and robot. From there, Victor participated in testing the sensors implemented in the robot for collision detection. He also helped install the software components for the Bluetooth connectivity from the PC to the laptop. He also assisted with the testing of the Bluetooth connection to the laptop, and found the problem that forced the team to order a new Bluetooth component. The Bluetooth Mate Silver is more compatible with the microcontroller installed onto the robot, the Arduino UNO. Victor then continued to test the new Bluetooth component with the PC and microcontroller. Once the software components were installed and completed he helped create a schematic to solder the circuit to the breadboard. In terms of the body, Victor made modifications to portions of the tank to allow space for a battery, USB, and connection for the microcontroller. Later on in the project only one of the tracks would rotate. In order to discover the issue Victor helped test the motor and motor shield in order to discover which part had issues. Once the motor was discovered to have an issue, he switched out the capacitors in order to further test the part.



Matthew Beutler – Presentation Preparation

Matt's primary technical role this semester was as the Mechanical Design Engineer

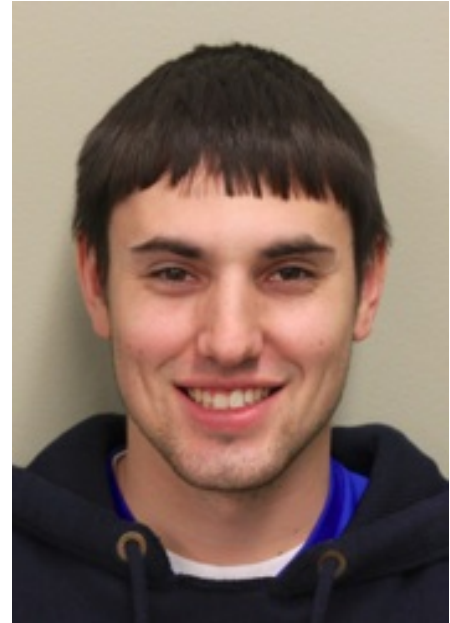
As the Mechanical Design Engineer for the autonomous tracking robot, he took charge of the body of the robot. The first prototype assembly was quickly realized to be insufficient to meet the design needs. Once this was realized, he worked designing a body from the ground up, using some parts from old RC cars as well as some purchased components. He also developed designs in 3D modeling software, measured lengths of parts that were available, and showed



the team what he had come up with. Because of unfamiliarity with mechanical construction, the team decided it was better to purchase a finished product. With team member support, the team purchased a more rugged, complete tank assembly. The tank was fully assembled and had many components unneeded in the project. With access and familiarity with tools in the open lab next to the 480 lab and familiarity in the machine shop, Matt and other team members removed unnecessary parts of the tank that was purchased, such as the cannon and pellet shooting mechanism. Matt was also in charge of where and how to mount the sensors, battery, and microcontroller. Using plastic parts that were surplus in the labs and cutting them down to specifications, he created mounts for the sensors and using nuts and screws, mounted them to the tank. Using some of the surplus plastic and screws, he also created the box for the battery charger by cutting parts out, assembling them, mounting the connection to the battery, and mounting the circuits inside. One other major contribution he made was the circuit design, realization, and testing of the battery charger. Through many hours of prototyping, testing, transferring from proto board to bread board and soldering, testing, and going through the cycle all over again, Matt achieved the design goal for a safe and quick charging battery circuit.

Brent Eisenmann – Document Preparation

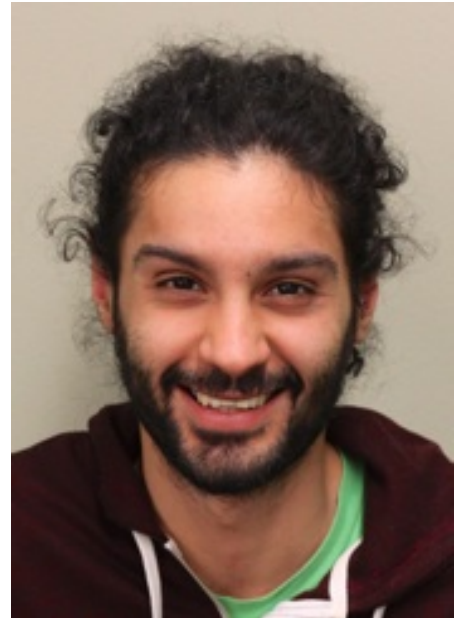
Brent's primary role this semester was the Sensor Technician, where he was in charge of choosing, implementing, and programming the Devantech SRF05 ultrasonic sensors. The SRF05 sensors that he chose were picked because of their compatibility with the design requirements. These included compatibility with the Arduino UNO microcontroller, the detection range of up to four meters, ease of programmability, and cost. When the sensors first arrived, the initial step was to download the Arduino microcontroller software. Once the software was available, the next vital step was choosing which mode



of operation to use for the sensors. Brent chose mode 2 because it allowed the team to combine the trigger and echo pins into one, so valuable I/O pins could be saved for other devices. Once this mode was chosen, research was done on how to properly write code for PING sensors such as these. Fortunately, the Arduino software has a PING sensor example program that Brent was able to build on. Programming included initializing the I/O pin as a PING sensor pin, and declaring all of the necessary distance variables. The distance measurements used for this project are both inches and centimeters, to make this prototype available to people in different regions of the world. Next, a prewritten code found on the SRF05 sensor datasheet was used to convert the measured time response from the sensor ping into a length. Brent then wrote code for the Arduino to print out all of the distance measurements so they could be viewed using the Serial Monitor in the Arduino software. The final step was actually installing and testing the sensors. Once proper mounts were selected by the team, Brent installed three of the SRF05 sensors onto the tank in the manner shown above. After connecting the proper pins to ground, 5V power, and the sensor pin, the final stage was testing. This was done by placing an object in front of each sensor, and verifying that the measured distance matched the physical distance. The angle of measurement was measured as well, to ensure the front sensors would be able to find the tracked target.

Hisham Hassan – Presentation Preparation

Hisham was assigned the duty of Networking Administrator. His understanding of computer networks allowed him to work on setting up the Bluetooth network communication between the robot and a remote laptop. This was done by setting up the necessary wiring between the Arduino Microcontroller and a Bluetooth module, as well as writing and maintaining the necessary C++ code that would allow the Microcontroller to use the Bluetooth connection to interface with the laptop. Hisham's background in computer engineering also helped him write, test, and maintain the code necessary to achieve collision avoidance. He also worked on connecting the Arduino Microcontroller to the Ultrasonic Range Sensors, and programming the microcontroller using the Arduino IDE to use the sensor readings to avoid collision with obstacles. Hisham also collaborated with Peng Xie in order to achieve proper manual and autonomous robot control. This was achieved by using C# and the open CV library to allow a Wi-Fi camera mounted on the robot to recognize a certain range of colors and shapes. The camera would also stream video to the laptop. Once this portion was completed, he worked on writing, testing, and maintaining the code that allowed the robot to track the object and move autonomously. Manual control was also achieved by writing and testing code accordingly. Hisham also contributed to the design of an easy to use graphical user interface that allows the user to take control over the robot at any time, thereby over-riding the autonomous movement of the robot. He additionally worked on implementing other functionalities in the GUI, such as changing the parameters of object detection by changing the color, and giving the robot the ability to connect to other Wi-Fi enabled cameras that may be connected to the network. Hisham also worked on testing the code and making any necessary adjustments that were needed in order to allow for smoother movement by the robot.



Thomas Schriefer – Lab Coordinator

Thomas' technical role for this project was the data analyzer. Early on, this meant running the numbers to make sure that the range finders could detect objects within the desired range, and that the battery contained enough power to run the robot for at least one hour. This also included testing that the motors were powerful enough to allow the robot to travel at sufficient speeds, and a camera and Bluetooth transceiver that were capable of communicating with the computer at acceptable distances. As the semester went on, the need for a data analyzer diminished, and he spent more time designing, debugging, and building analog circuits for the robot and battery charger. The first circuit that he designed enabled the webcam, which is normally powered by a wall outlet, to be powered by the on board battery of the robot. The webcam itself runs off of 5V DC and can draw up to 1.5A of current. The circuit was realized by splicing the power adapter that came with the webcam to a DC regulator. Next, he designed a circuit that powered the LEDs in the front of the robot. This was completed by soldering the cathodes of the LEDs to ground and the anodes to a resistor, which was soldered to the V_{cc} bus of the protoboard. This turned on the LEDs whenever the power was applied. While not critical, this circuit helped to improve the presentation of the project. Most recently, he has designed and built a circuit that will light a green LED when the battery is finished charging. This circuit uses a comparator to measure the voltage across the battery, and compares it to a reference voltage. When the battery voltage exceeds the reference voltage, a green LED is turned on, indicating that the battery is sufficiently charged.



Peng Xie - Webmaster

Peng was selected to be the software developer for the project because of his experience in software developing and interest in designing target tracking software. His task was to design the client software on a Windows operating system for the robot. To achieve the objective of target tracking, he researched several programming libraries that support target tracking and finally chose the Open CV library after some discussions with the rest of the design team. The software is developed in C# with Emgu CV, which is a .NET framework for Open CV library. Before starting the design of the final



software, Peng made several prototypes to make sure each of the individual components was working correctly. One prototype he made used Bluetooth serial communication to control the movement of the robot. Another prototype is an application that shows real-time video from the webcam and tracks a certain target in the video. Based on the second prototype, Peng added the function of automatic and manual robot control through the Bluetooth serial communication port, which made it the first prototype of the final software design. After some testing with teammates, Peng added more elements in the GUI to make the software more user-friendly and give users more options to control the robot and target tracking. Peng also collaborated with Hisham Hassan to make sure the manual and automatic control worked correctly. Peng also helped on the designing and debugging of the Arduino code with Hisham. To make the client software work flawlessly with the robot hardware, Peng worked closely with other teammates that were in charge of building circuits for the robot. He built a final design with Hisham, which puts the target tracking function in client software and distance measuring function in the Arduino program. With the help from other teammates, Peng finalized the GUI and functions of the client software.

Appendix 2

LITERATURE AND WEBSITE REFERENCES

- [1] Bellis , Mary. "Timeline of Robots." *About.com*. About.com, n.d. Web. 18 Feb 2013.<<http://inventors.about.com/od/roboticsrobots/a/RoboTimeline.htm>>.
- [2] Moravec, Hans. "The Stanford Cart and The CMU Rover." . Carnegie-Mellon University, n.d. Web. 18 Feb 2013.
<<http://www.frc.ri.cmu.edu/~hpm/project.archive/robot.papers/1983>>
- [3] "Robot Timeline - Robotic History." *RobotWorx*. RobotWorx, n.d. Web. 18 Feb 2013.
<[http://www.used-robots.com/robot-education.php?page=robot timeline](http://www.used-robots.com/robot-education.php?page=robot%20timeline)>.
- [4] Weiss, Lora. "Autonomous Robots in the Fog of War." *ieee Spectrum*. (2011): n. page. Web. 13 Feb. 2013. <<http://spectrum.ieee.org/robotics/military-robots/autonomous-robots-in-the-fog-of-war/1>>.
- [5] "Arduino UNO." *Arduino*. Arduino, n.d. Web. 17 Apr 2013.
<<http://arduino.cc/en/Main/arduinoBoardUno>>.
- [6] "2A Motor Shield for Arduino." *DFRobot*. DFRobot, n.d. Web. 17 Apr 2013.
<http://www.dfrobot.com/index.php?route=product/product&product_id=69>
- [7] "SRF05 - Ultra-Sonic Ranger." *Robot Electronics*. Devantech, n.d. Web. 17 Apr 2013. <<http://www.robot-electronics.co.uk/htm/srf05tech.htm>>.
- [8] "Bluetooth Mate Silver." *Sparkfun Electronics*. Sparkfun Electronics, n.d. Web. 17 Apr 2013. <<https://www.sparkfun.com/products/10393>>.
- [9] "mydlink-enabled Wireless N Network Camera." *DCS-930L*. D-Link Corporation, n.d. Web. 17 Apr 2013. <<https://www.sparkfun.com/products/10393>>.
- [10] "Tamiya 70100 Track and wheel set." *Micro Robotics*. Micro Robotics, n.d. Web. 17 Apr 2013. <<http://robotics.org.za/index.php?route=product/product&product>>.

- [11] "LM317 3-terminal Positive Adjustable Regulator." . Fairchild Semiconductor Corporation, n.d. Web. 22 Apr 2013. <www.fairchildsemi.com>.
- [12] "LM34 Precision Fahrenheit Temperature Sensors Regulator." . Texas Instruments, n.d. Web. 22 Apr 2013. <<http://www.ti.com/lit/ds/symlink/lm34.pdf>. >.
- [13] "TIP31/TIP31A/TIP31/B/TIP31C NPN Epitaxial Silicon Transistor." . Fairchild Semiconductor Corporation, n.d. Web. 22 Apr 2013. <<http://www.fairchildsemi.com/ds/TI/TIP31C.pdf>>.

Appendix 3

DETAILED TECHNICAL ATTACHMENTS SOLUTION SELECTION METRICS

		Look/Style		Maneuverability		Ease of Use		Cost		Total Score
		out of 5	Relative Weight (.15)	out of 5	Relative Weight (.35)	out of 5	Relative Weight (.20)	out of 5	Relative Weight (.30)	
Appearance/ Maneuverability	4 Wheel RC Car	2	0.3	3	1.05	4	0.8	3	0.9	3.05
	Tank	3	0.45	3	1.05	4	0.8	3	0.9	3.2
	Quadcopter	5	0.75	5	1.75	1	0.2	1	0.3	3
	Smart Phone	3	0.45	2	0.7	2	0.4	1	0.3	1.85
Control	Videogame Controller	3	0.45	3	1.05	4	0.8	4	1.2	3.5
Communication	Bluetooth	-	-	-	-	4	0.8	4	1.2	2
	Wifi	-	-	-	-	2	0.4	2	0.6	1
Tracking	Radar	-	-	-	-	1	0.2	4	1.2	1.4
	Webcam	-	-	-	-	3	0.6	4	1.2	1.8
	UltraSonic Range Finder	-	-	-	-	3	0.6	5	1.5	2.1
	Buying Premade	2	0.3	-	-	5	1	3	0.9	2.2
Body	Building from the Ground	5	0.75	-	-	1	0.2	2	0.6	1.55
	Mix of Buying and Building	4	0.6	-	-	3	0.6	4	1.2	2.4

Primary	Secondary	Manual Controller	Webcam with OpenCV	Ultra-Sonic Sensor	Battery	Tamiya Gearbox	Tank L
	Remotely controllable to and from the target area	9				3	
	Autonomous mode to search 360 degrees for a predefined target		9				
	Close to within three feet of target without ...			9			

Table 2A: Decision Matrix

Primary and Secondary follow marked target using camera	movements				3		
	Self-powered for one hour		1	1	9	1	
	Five mph speed				9	9	
	Zero degree turning radius	1				3	9
	An electrical package including significant computing power, power supply, & motor control	3	3	3	9	3	
	Two-way communication	1	9	3	3		

GANTT CHART

	1 Task Mode	Name	Leveling Delay	Duration	Start	Finish	Successors
1	✓	Gant Chart	0 edays 3 days		Wed 1/30/13	Fri 2/1/13	
2	✓	Pre-Proposal	0 edays 3 days		Wed 1/30/13	Fri 2/1/13 7	
3	✓	Research	0 edays 8.5 days		Wed 1/30/13	Mon 2/11/13 16,7	
4	✓	Voice of customer assign	0 edays 6 days		Wed 1/30/13	Wed 2/6/13	
5	✓	Web page design	0 edays 4 days		Wed 1/30/13	Sun 2/3/13	
6	✓	EAET Program	0 edays 5 days		Mon 2/4/13	Fri 2/8/13	

Table 3A: Gantt Chart

16	✓	Order Parts	0 edays 3 days		Thu 2/14/13	Sun 2/17/13 17	
17	✓	Prototyping	0 edays 3 wks		Mon 2/18/13	Fri 3/8/13 18	
7	✓	Final Proposal	0 edays 4 days		Tue 2/19/13	Fri 2/22/13	
15	✓	Spring Break	0 edays 7 days		Sat 3/2/13	Sun 3/10/13	
9	✓	Progress Report	0 edays 3 days		Mon 3/11/13	Wed 3/13/13	
18	✓	Testing and Troubleshoo	0 edays 10 days		Mon 3/11/13	Fri 3/22/13 19	
19	✓	Assembly	0 edays 1 wk		Mon 3/25/13	Fri 3/29/13 20	
20	✓	Testing and Troubleshoo	0 edays 5 days		Mon 4/1/13	Fri 4/5/13 21,10	
11	✓	Progress Report 2	0 edays 4 days		Fri 4/5/13	Wed 4/10/13 14	
10	✓	Design Issues Paper	0 edays 5 days		Mon 4/8/13	Fri 4/12/13	
21	✓	Modifications	0 edays 1 wk		Mon 4/8/13	Fri 4/12/13 14,12,13	
12	✓	Final Report	0 edays 6 days		Mon 4/15/13	Mon 4/22/13	

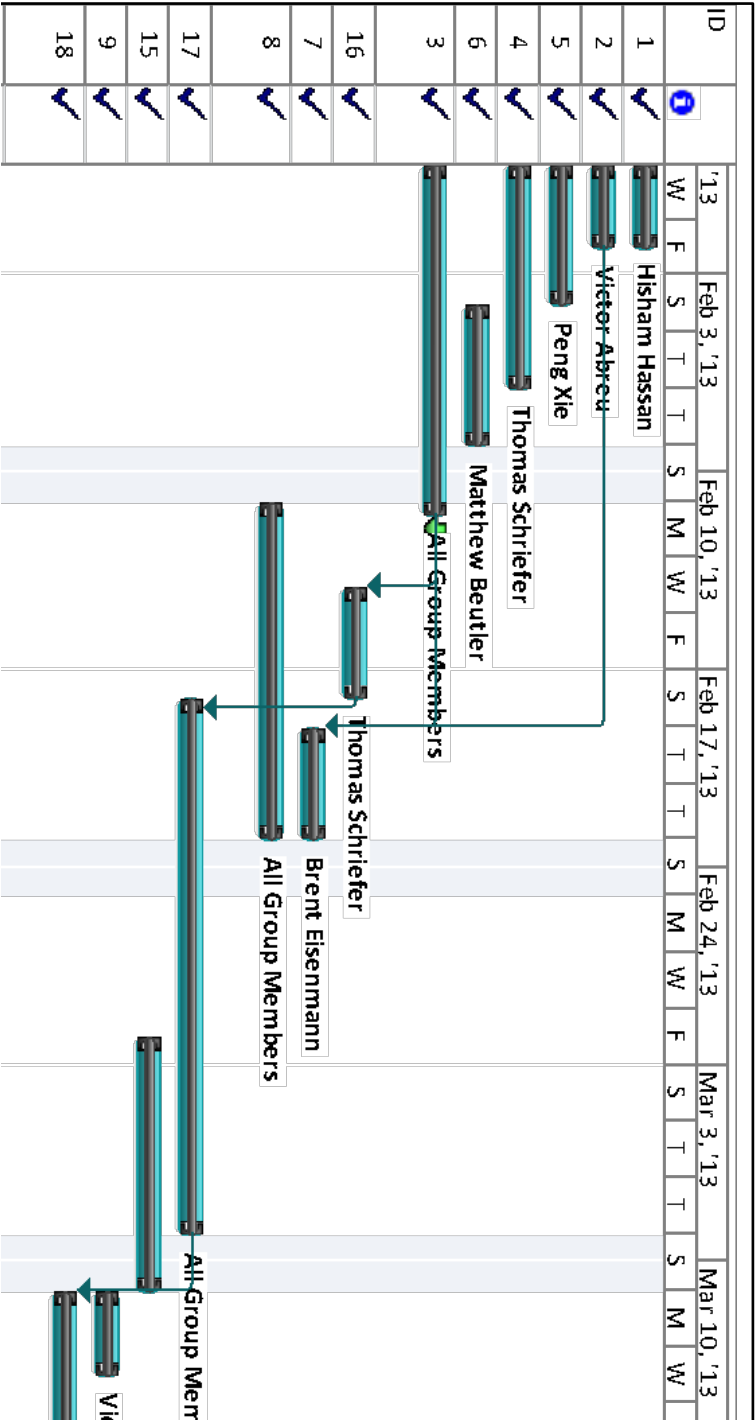


Figure 1A: Gantt Chart Page 1

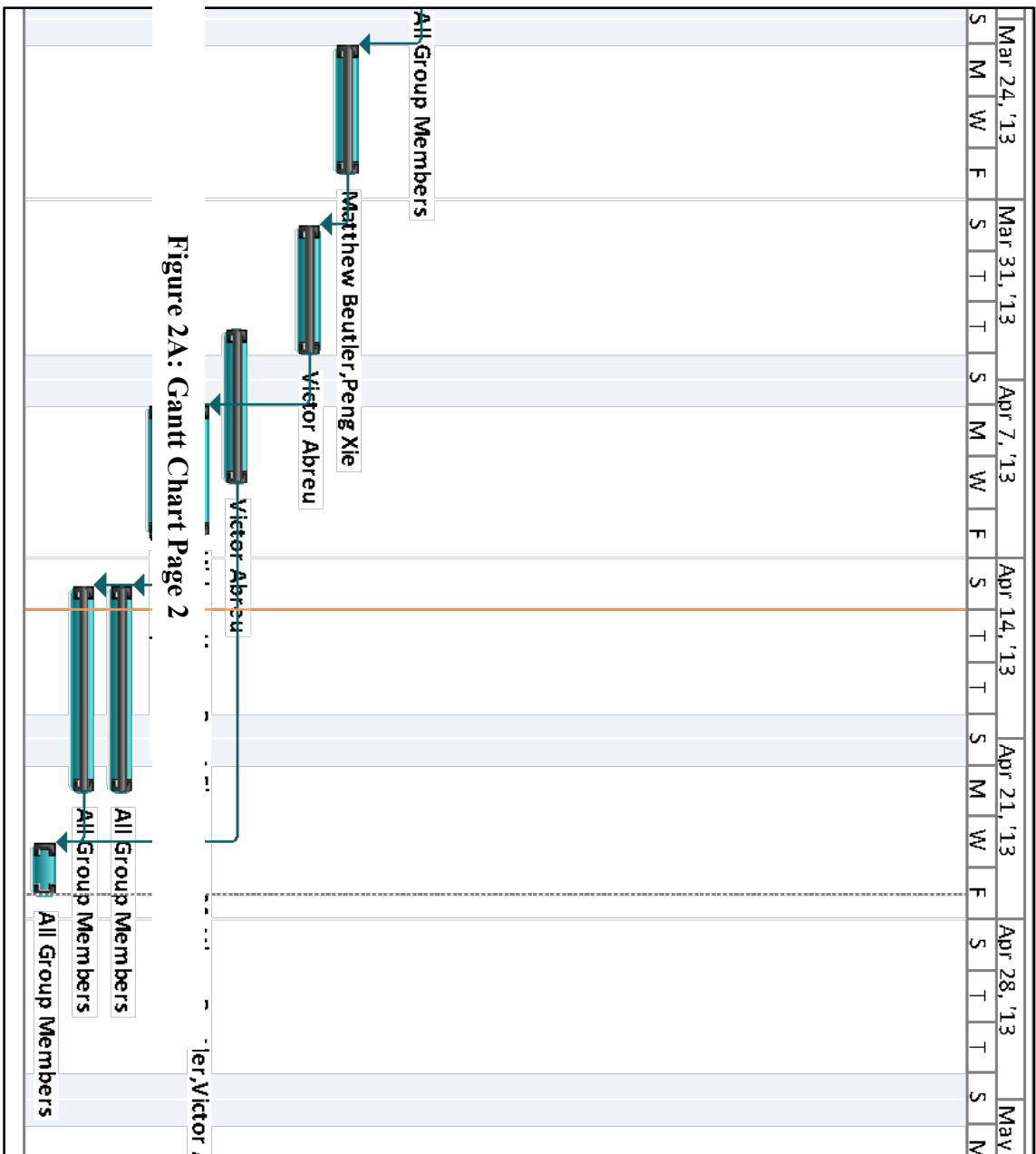


Figure 2A: Gantt Chart Page 2

WIRE SCHEMATICS

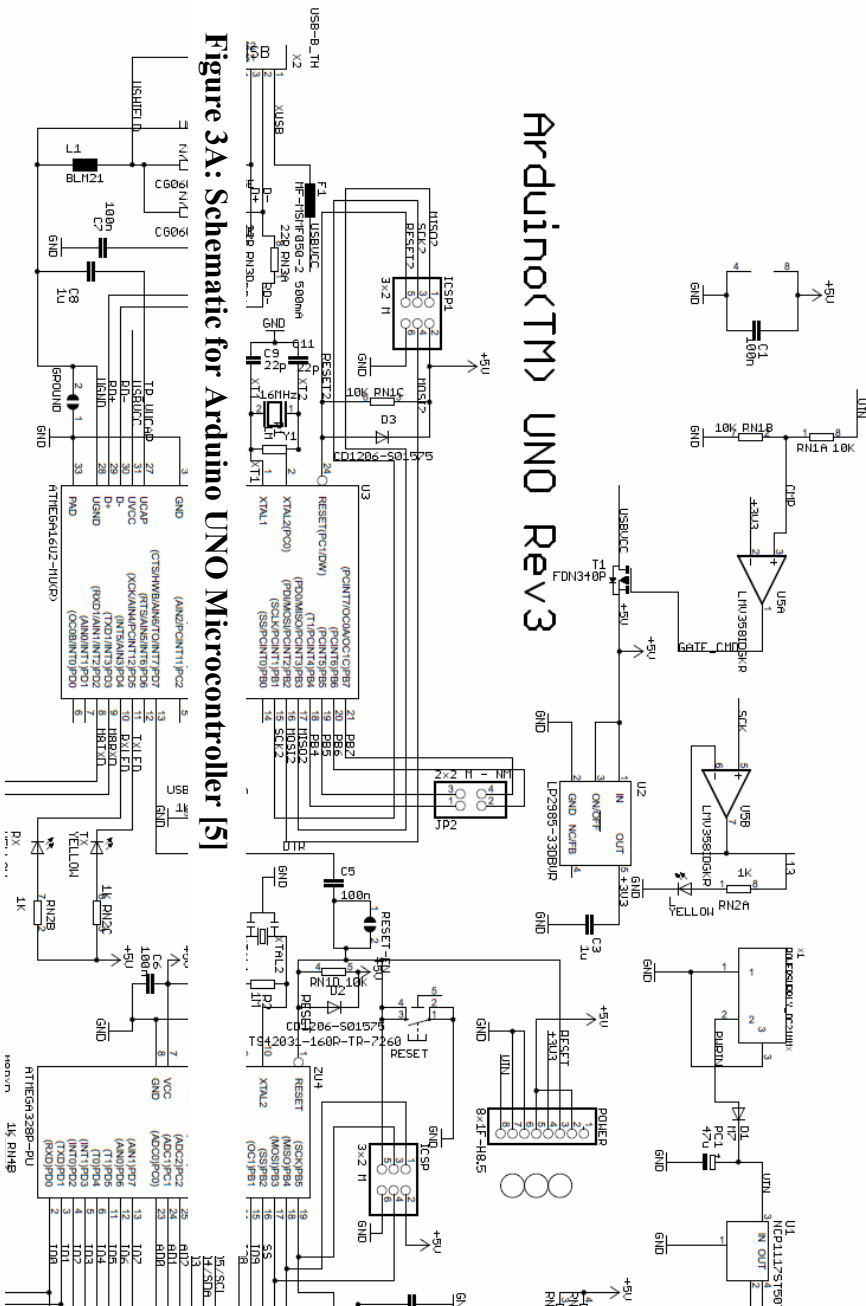


Figure 3A: Schematic for Arduino UNO Microcontroller [5]

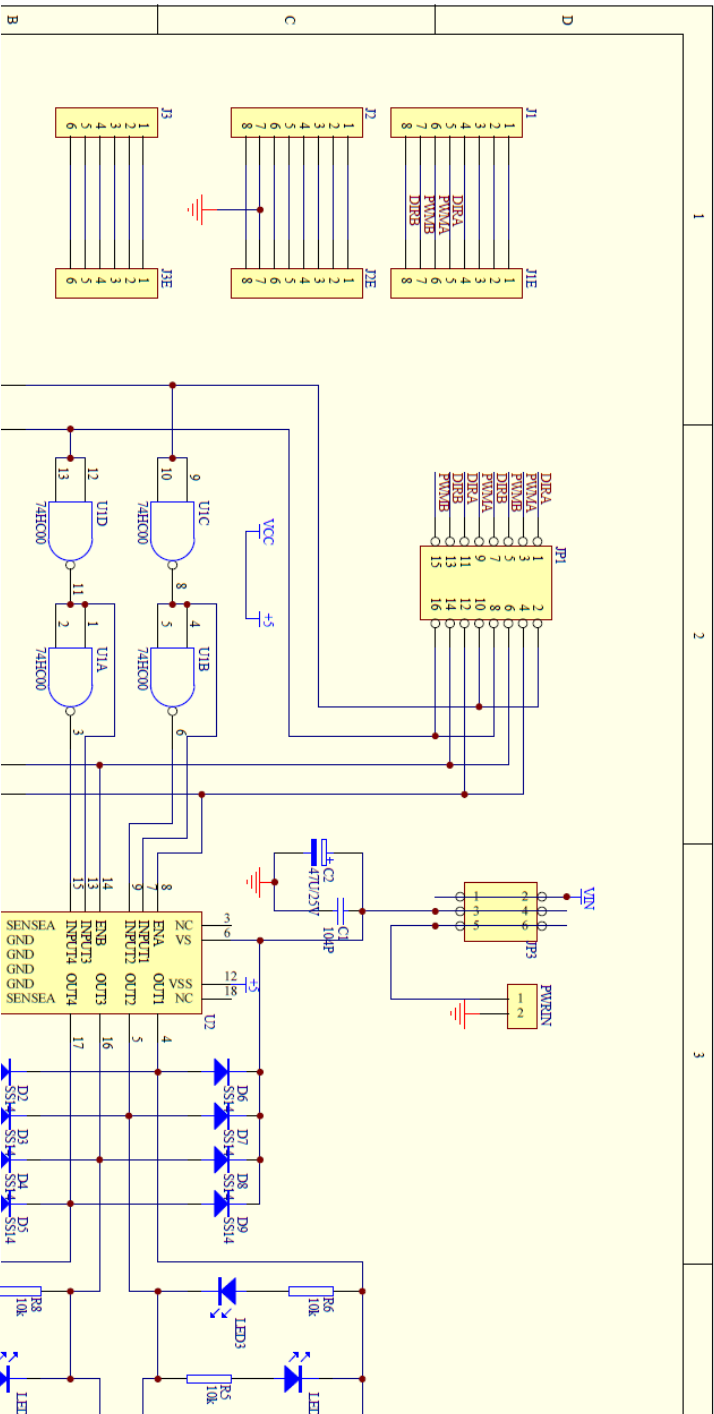
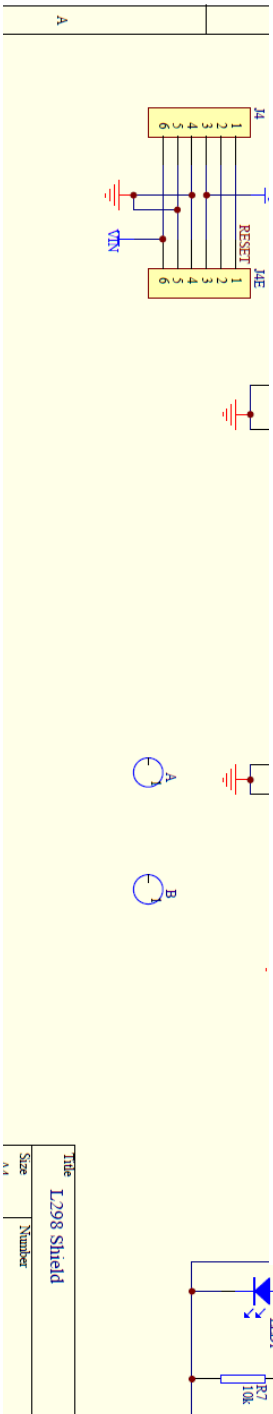
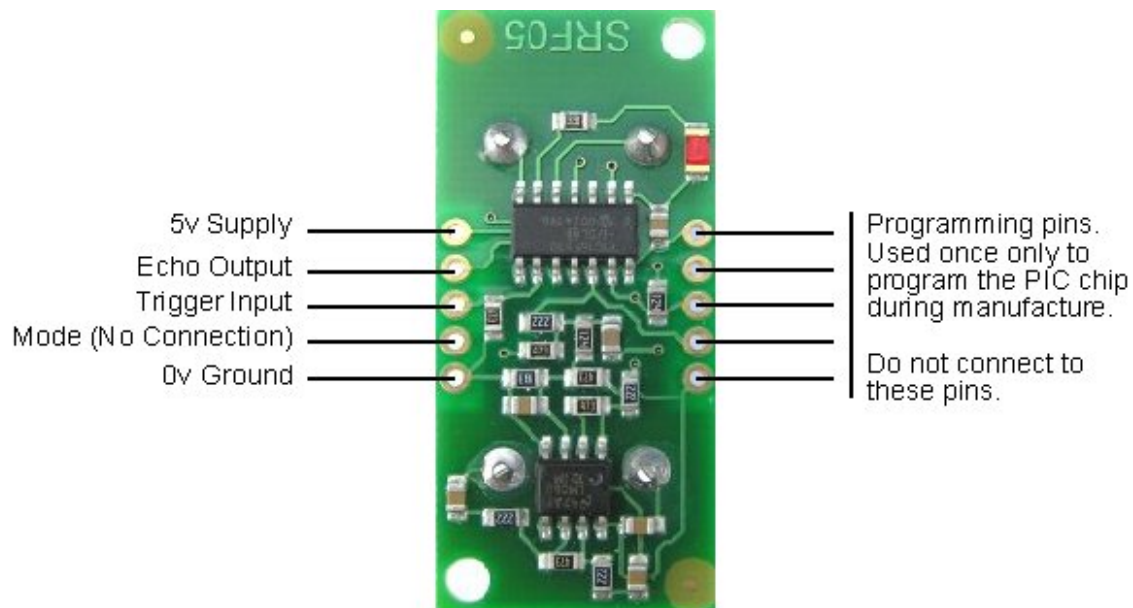


Figure 4A: Schematic for Motor Shield [6]



MODE 1 OPERATION DEVANTECH SRF05 ULTRASONIC SENSOR



Connections for 2-pin Trigger/Echo Mode (SRF04 compatible)

Figure 5A: Devantech SRF05 Ultrasonic Sensor Operation Mode 1

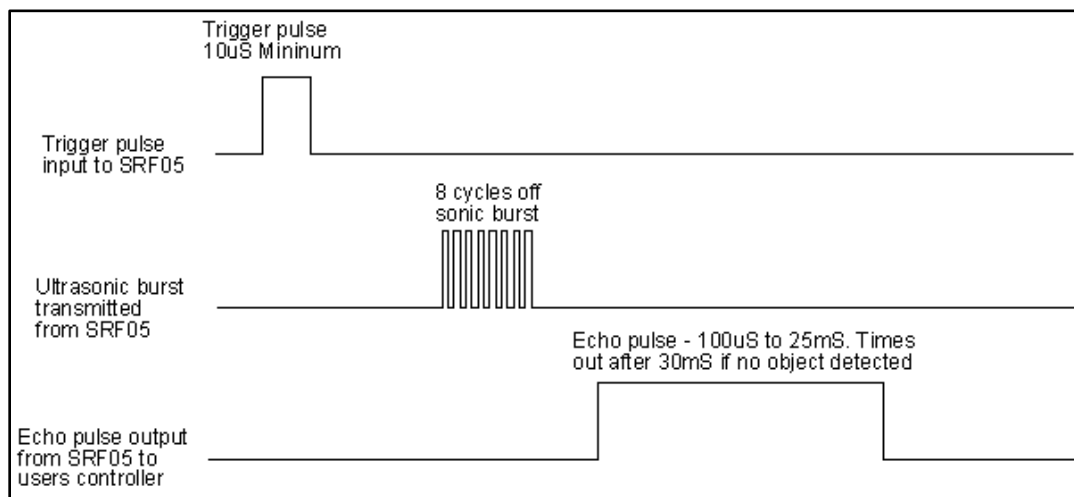
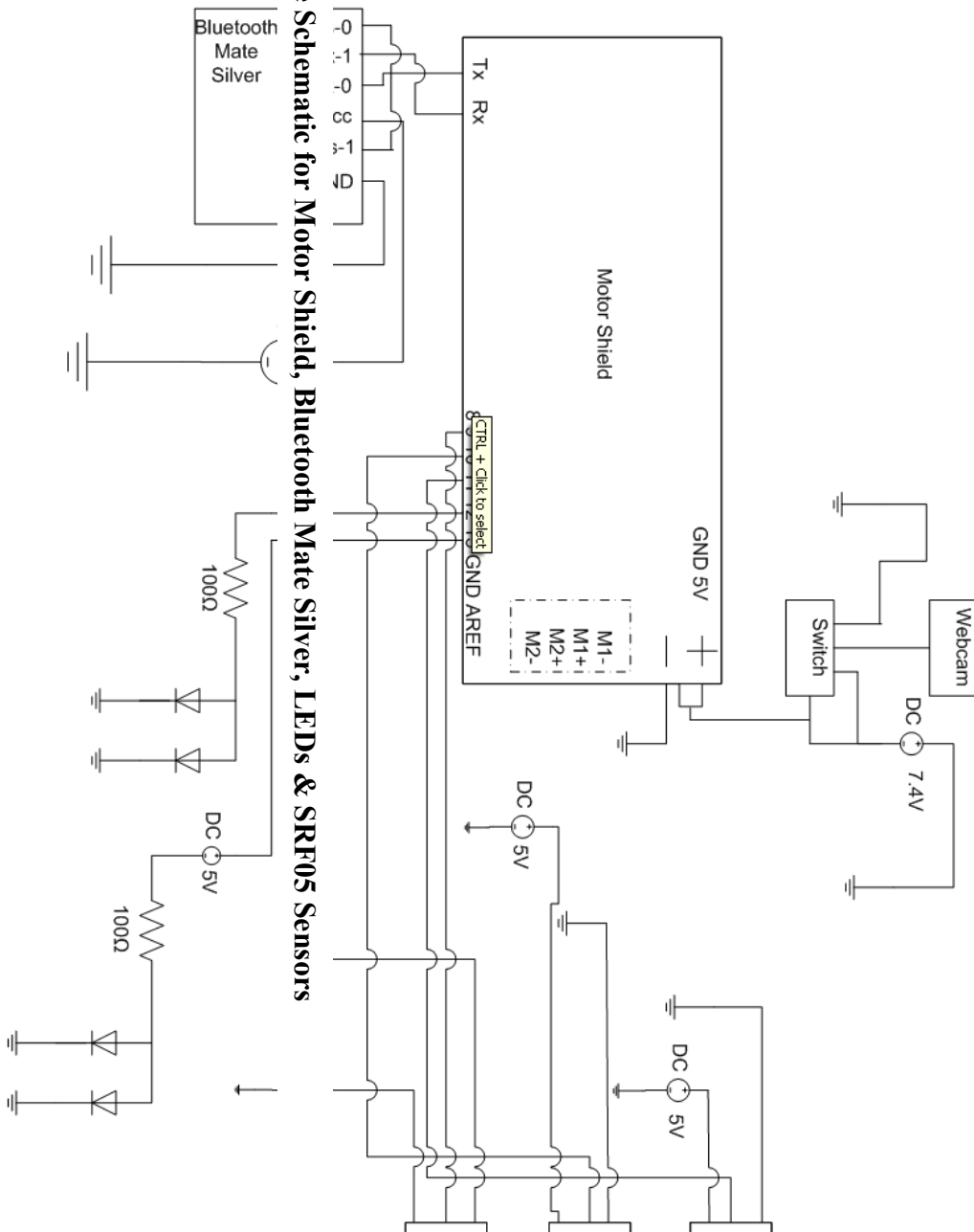


Figure 6A: SRF05 Mode 1 Timing Diagram

<http://www.robot-electronics.co.uk/htm/srf05tech.htm>

WIRE CONNECTIONS

Figure 7A: re Schematic for Motor Shield, Bluetooth Mate Silver, LEDs & SRF05 Sensors



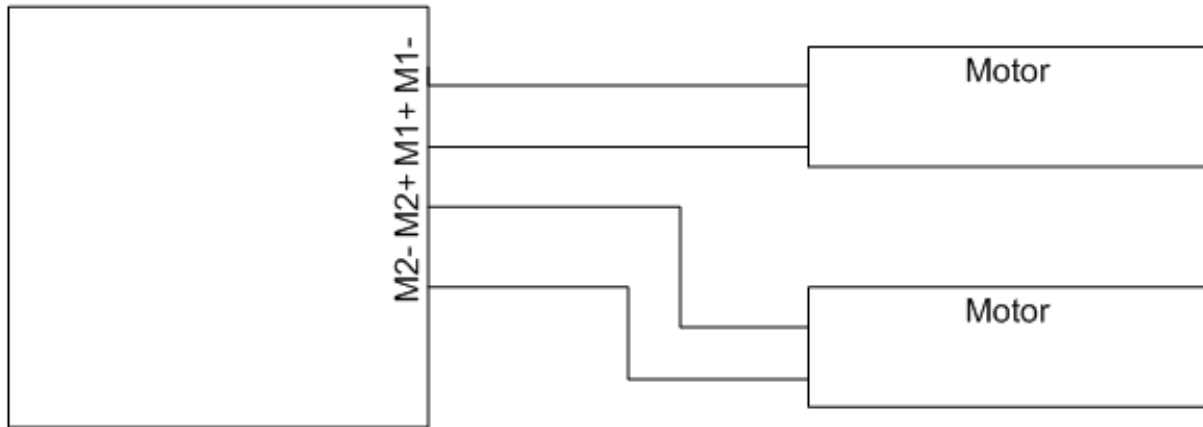


Figure 8A: Connection of 2 Motors to DFRduino Motor Shield

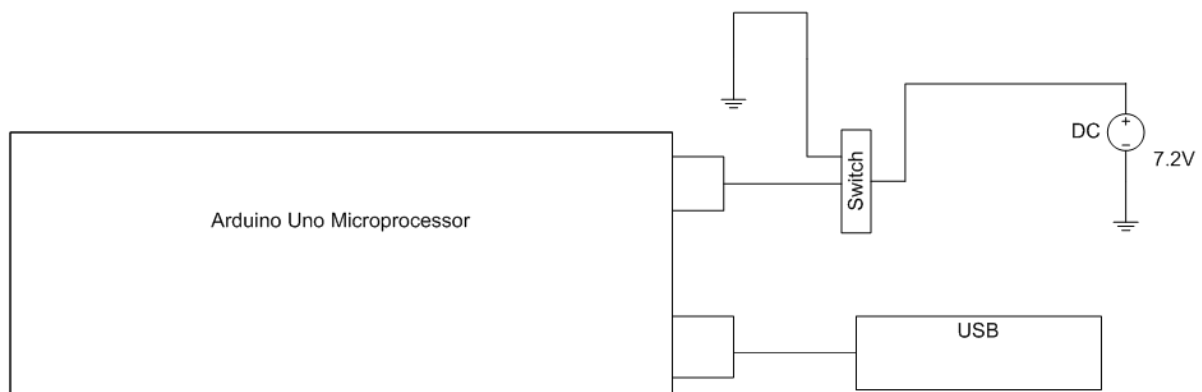


Figure 9A: Connection of Power and USB Programming Port to Arduino UNO Microcontroller

INITIAL PROTOTYPE DESIGN

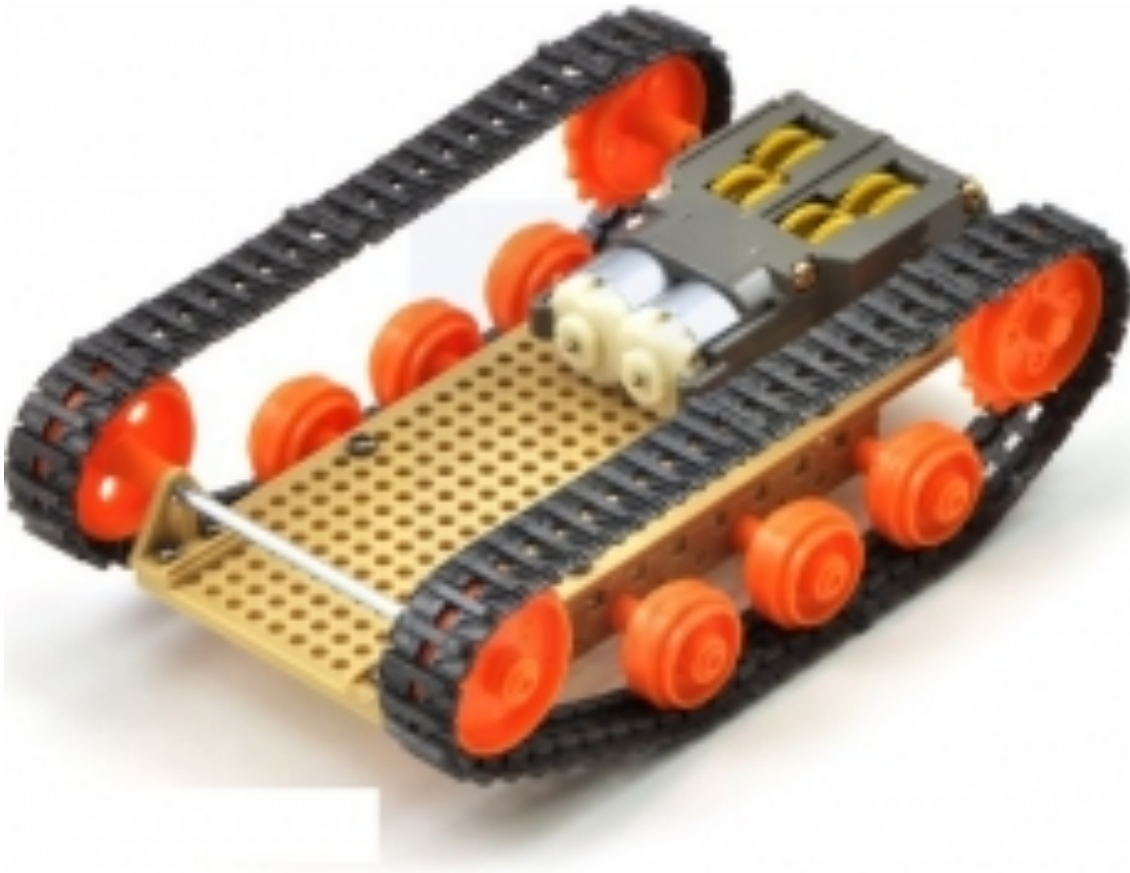


Figure 10A: Timaya 70100 Tank-Initial Prototype Design [10]

C# SOURCE CODES

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.CV.UI;

namespace Emgu_Test
{
    public partial class Form1 : Form
    {
        Capture capWebCam = null; //init the object for webcam video capture
        bool IsCapturing = false; //setup flag to indicate if the capture is
        running or not
        bool IsManual = true; //setup flag to indicate if it's manual control or
        auto control
        Image<Bgr, Byte> imageOriginal; //init the object for original image (un-
        processed video stream from webcam)
        Image<Gray, Byte> imageProcessed; //init the object for processed image
        string previousCMD = ""; //init an string to record previous command in
        order to prevent sending same command repeatedly

        int minB_Val, minG_Val, minR_Val, maxB_Val, maxG_Val, maxR_Val; //init
        integer values to define color range for tracking

        //for serial COM
        string RxString; //string containing incoming data from bluetooth device

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Get a list of serial port names.
            string[] ports = SerialPort.GetPortNames();
            // add string array to combobox
            comboBox1.Items.AddRange(ports);

            // Make a array of baud rates
            string[] rates = new string[] { "9600", "38400", "115200" };
            // add array to combobox
            comboBox2.Items.AddRange(rates);
        }

        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            // Clean up for closing
        }
    }
}
```

```

        if (capWebCam != null)
        {
            serialPort1.Write("s");
            serialPort1.Dispose();
            capWebCam.Dispose();
        }
    }

    // Method for video processing
    void processFrameAndUpdateGUI(object sender, EventArgs e)
    {
        imageOriginal = capWebCam.QueryFrame();
        if (imageOriginal == null) return;

        // Processed image will only show color objects in color range (as
        white on screen)
        imageProcessed = imageOriginal.InRange(new Bgr(minB_Val, minG_Val,
        minR_Val), //lower limit value
            new Bgr(maxB_Val, maxG_Val, maxR_Val)); //higher limit value

        imageProcessed = imageProcessed.SmoothGaussian(15);

        // Define circles to mark objects in video
        CircleF[] circles = imageProcessed.HoughCircles(new Gray(100), new
        Gray(50), 2, imageProcessed.Height / 4, 10, 200)[0];

        // Auto mode: webcam sends commands
        if (!IsManual)
        {
            foreach (CircleF circle in circles)
            {
                if (IsManual) break;

                if (txt_xyValues.Text != "")
                txt_xyValues.AppendText(Environment.NewLine);

                // Display object position in text box
                txt_xyValues.AppendText("Ball position = x" +
                circle.Center.X.ToString().PadLeft(4) +
                ". y =" + circle.Center.Y.ToString().PadLeft(4) +
                ". radius =" +
                circle.Radius.ToString("###.000").PadLeft(7));

                txt_xyValues.ScrollToCaret();

                CvInvoke.cvCircle(imageOriginal,
                    new Point((int)circle.Center.X, (int)circle.Center.Y),
                    3,
                    new MCvScalar(0, 255, 0),
                    -1,
                    LINE_TYPE.CV_AA,
                    0);
                imageOriginal.Draw(circle,
                    new Bgr(Color.Red),
                    3);

                // sending cmd
                if (serialPort1.IsOpen && circle.Center.X < 640 &&
                circle.Center.X > 0)

```

```

{
    // Object moving to left, turn left.
    if (circle.Center.X > 426 && !previousCMD.Equals("a"))
    {
        serialPort1.Write("a");
        previousCMD = "a";
    }

    // Object moving to right, turn right
    else if (circle.Center.X < 213 &&
!previousCMD.Equals("d"))
    {
        for (int i = 0; i < 1000000; i++) ;
        serialPort1.Write("d");
        previousCMD = "d";
    }

    // Object in center, go forward
    else if (circle.Center.X >= 213 && circle.Center.X <= 426
&& !previousCMD.Equals("x"))
    {
        for (int i = 0; i < 1000000; i++) ;
        serialPort1.Write("x");
        previousCMD = "x";
    }
}
}

//Manual Mode: User sending command using buttons in UI
else if (IsManual)
{
    foreach (CircleF circle in circles)
    {
        if (!IsManual) break;

        // Clear outgoing buffer for serial communication
        if (serialPort1.IsOpen)
        {
            serialPort1.DiscardOutBuffer();
        }

        if (txt_xyValues.Text != "")
txt_xyValues.AppendText(Environment.NewLine);

        // Display object position in text box
        txt_xyValues.AppendText("Ball position = x" +
circle.Center.X.ToString().PadLeft(4) +
        ". y =" + circle.Center.Y.ToString().PadLeft(4) +
        ". radius =" +
circle.Radius.ToString("###.000").PadLeft(7));

        txt_xyValues.ScrollToCaret();

        CvInvoke.cvCircle(imageOriginal,
            new Point((int)circle.Center.X, (int)circle.Center.Y),
            3,
            new MCvScalar(0, 255, 0),
            -1,

```

```

        LINE_TYPE.CV_AA,
        0);
        imageOriginal.Draw(circle,
        new Bgr(Color.Red),
        3);
    }
}

ib_Original.Image = imageOriginal;
ib_Processed.Image = imageProcessed;
}

// Button to pause or resume the video stream
private void btn_Control_Click(object sender, EventArgs e)
{
    if (IsCapturing == true)
    {
        Application.Idle -= processFrameAndUpdateGUI;
        IsCapturing = false;
        btn_Control.Text = "Resume";
    }
    else
    {
        Application.Idle += processFrameAndUpdateGUI;
        IsCapturing = true;
        btn_Control.Text = "Pause";
    }
}

// Button to open COM port and start communication
private void buttonStart_Click(object sender, EventArgs e)
{
    if (comboBox1.Text != "" && comboBox2.Text != "")
    {
        textBox1.Clear();

        serialPort1.PortName = comboBox1.Text;
        serialPort1.BaudRate = int.Parse(comboBox2.Text);

        serialPort1.Open();
        if (serialPort1.IsOpen)
        {
            buttonStart.Enabled = false;
            buttonStop.Enabled = true;
            textBox1.ReadOnly = false;
        }
    }
    else
    {
        textBox1.Clear();
        textBox1.AppendText("Error: Empty COM Port or Baud Rate.");
    }
}

// Button to close COM port
private void buttonStop_Click(object sender, EventArgs e)
{
    if (serialPort1.IsOpen)

```

```

        {
            serialPort1.Close();
            buttonStart.Enabled = true;
            buttonStop.Enabled = false;
            textBox1.Clear();
            textBox1.ReadOnly = true;
        }
    }

    // Event for processing COM text box inputs. Currently disabled.
    private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
    {
        //Code for this block is commented out because sending out commands
        using text box is not needed.
        //Uncomment following codes to enable sending commands using text box.
        /*
        // If the port is closed, don't try to send a character.

        if (!serialPort1.IsOpen) return;

        // If the port is Open, declare a char[] array with one element.
        char[] buff = new char[1];

        // Load element 0 with the key character.

        buff[0] = e.KeyChar;

        // Send the one character buffer.
        serialPort1.Write(buff, 0, 1);

        // Set the KeyPress event as handled so the character won't
        // display locally. If you want it to display, omit the next line.
        //e.Handled = true;
        */
    }

    // Display incoming text in COM textbox
    private void DisplayText(object sender, EventArgs e)
    {
        textBox1.AppendText(RxString);
    }

    // Event to display incoming data
    private void serialPort1_DataReceived(object sender,
    System.IO.Ports.SerialDataReceivedEventArgs e)
    {
        RxString = serialPort1.ReadExisting();
        this.Invoke(new EventHandler(DisplayText));
    }

    // Close serial port on closing
    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (serialPort1.IsOpen) serialPort1.Close();
    }

    // Events to handle button pushing for manual control
    private void buttonForward_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen) return;
    }

```

```

        serialPort1.Write("x");
    }

    private void buttonPark_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen) return;
        serialPort1.Write("s");
    }

    private void buttonReverse_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen) return;
        serialPort1.Write("w");
    }

    private void buttonLeft_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen) return;
        serialPort1.Write("d");
    }

    private void buttonRight_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen) return;
        serialPort1.Write("a");
    }

    private void buttonLEDon_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen) return;
        serialPort1.Write("1");
    }

    private void buttonLEDOff_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen) return;
        serialPort1.Write("0");
    }

    // Events to handle color range changes
    private void MinB_TextChanged(object sender, EventArgs e)
    {
        if (MinB.Text == "") return;
        if (int.Parse(MinB.Text) >= 0 && int.Parse(MinB.Text) <= 256)
        {
            minB_Val = int.Parse(MinB.Text);
        }
    }

    private void MinG_TextChanged(object sender, EventArgs e)
    {
        if (MinG.Text == "") return;
        if (int.Parse(MinG.Text) >= 0 && int.Parse(MinG.Text) <= 256)
        {
            minG_Val = int.Parse(MinG.Text);
        }
    }

    private void MinR_TextChanged(object sender, EventArgs e)
    {

```

```

        if (MinR.Text == "") return;
        if (int.Parse(MinR.Text) >= 0 && int.Parse(MinR.Text) <= 256)
        {
            minR_Val = int.Parse(MinR.Text);
        }
    }

    private void MaxB_TextChanged(object sender, EventArgs e)
    {
        if (MaxB.Text == "") return;
        if (int.Parse(MaxB.Text) >= 0 && int.Parse(MaxB.Text) <= 256)
        {
            maxB_Val = int.Parse(MaxB.Text);
        }
    }

    private void MaxG_TextChanged(object sender, EventArgs e)
    {
        if (MaxG.Text == "") return;
        if (int.Parse(MaxG.Text) >= 0 && int.Parse(MaxG.Text) <= 256)
        {
            maxG_Val = int.Parse(MaxG.Text);
        }
    }

    private void MaxR_TextChanged(object sender, EventArgs e)
    {
        if (MaxR.Text == "") return;
        if (int.Parse(MaxR.Text) >= 0 && int.Parse(MaxR.Text) <= 256)
        {
            maxR_Val = int.Parse(MaxR.Text);
        }
    }

    // Event to switch between auto and manual control
    private void buttonMA_Click(object sender, EventArgs e)
    {
        if (IsManual == true)
        {
            IsManual = false;
            if (serialPort1.IsOpen)
            {
                serialPort1.DiscardOutBuffer();
                serialPort1.DiscardInBuffer();
                serialPort1.Write("s");
            }
            buttonMA.Text = "Manual";
        }
        else
        {
            IsManual = true;
            if (serialPort1.IsOpen)
            {
                serialPort1.Write("s");
            }
            buttonMA.Text = "Auto";
        }
    }

    // Button to start video capture

```

```

private void ConnectBTN_Click(object sender, EventArgs e)
{
    if (IsCapturing == false)
    {
        try
        {
            if (URL_Box.Text == "")
            {
                capWebCam = new Capture();
            }
            else capWebCam = new Capture(URL_Box.Text);

            URL_Box.Enabled = false;
            ConnectBTN.Enabled = false;

        }
        catch (NullReferenceException except)
        {
            txt_xyValues.Text = except.Message;
            URL_Box.Enabled = true;
            return;
        }

        imageOriginal = capWebCam.QueryFrame();
        if (imageOriginal == null) return;

        minB_Val = int.Parse(MinB.Text);
        minG_Val = int.Parse(MinG.Text);
        minR_Val = int.Parse(MinR.Text);

        maxB_Val = int.Parse(MaxB.Text);
        maxG_Val = int.Parse(MaxG.Text);
        maxR_Val = int.Parse(MaxR.Text);

        Application.Idle += processFrameAndUpdateGUI;
        IsCapturing = true;
    }
}
}

```

ARDUINO SOURCE CODE

```
// Set up sensor pins
const int sensorL = 9; // left sensor
const int sensorR = 10; // right sensor
const int sensorB = 11; // back sensor

//Set up motor pin numbers
int IN2 = 4;
int EN2 = 5;
int EN1 = 6;
int IN1 = 7;

//Maximum speed - used for PWM control
int speed = 255;

//store sensor readings
//durationL for left sensor bounce-back time
//inchesL, cmL to store calculated object distances

long durationL, inchesL, cmL;
long durationR, inchesR, cmR;
long durationB, inchesB, cmB;

//Bluetooth Test LED connected to pin 2
//int ledpin = 2;

// control flag 0--Manual 1--Autonomous
int control = 0;

//store readings for manual control
char val;
```

```

void setup()
{
    //set motor and LED pins as outputs
    int i;
    for(i=4;i<=7;i++) pinMode(i, OUTPUT);

    //Set tank LED pins as outputs
    pinMode(ledpin = 12, OUTPUT);
    pinMode(ledpin = 13, OUTPUT);

    //Begin Bluetooth communication through the serial port
    //Bluetooth baud rate: 115200
    Serial.begin(115200);
}

void loop()
{

    //Make sure left sensor is clear
    pinMode(sensorL, OUTPUT);
    digitalWrite(sensorL, LOW);
    delayMicroseconds(2);

    //Send a pulse from left sensor
    digitalWrite(sensorL, HIGH);
    delayMicroseconds(5);
    digitalWrite(sensorR, LOW);

    //Measure bounce-back duration from left sensor
    pinMode(sensorL, INPUT);
    durationL = pulseIn(sensorL, HIGH);
}

```

```

//Convert to distance measurements
inchesL = microsecondsToInches(durationL);
cmL = microsecondsToCentimeters(durationL);

//Send measurements to PC client
Serial.print("Pin Sensor L: ");
Serial.print(inchesL);
Serial.print("in, ");
Serial.print(cmL);
Serial.print("cm");
Serial.println();

//Arduino cannot read from all sensors simultaneously
//Therefore delay before using next ultrasonic sensor
delay(200);

//Repeat algorithm for sensor R
pinMode(sensorR, OUTPUT);
digitalWrite(sensorR, LOW);
delayMicroseconds(2);
digitalWrite(sensorR, HIGH);
delayMicroseconds(5);
digitalWrite(sensorR, LOW);
pinMode(sensorR, INPUT);
durationR = pulseIn(sensorR, HIGH);
inchesR = microsecondsToInches(durationR);
cmR = microsecondsToCentimeters(durationR);
Serial.print("Pin Sensor R: ");
Serial.print(inchesR);
Serial.print("in, ");

```

```

Serial.print(cmR);
Serial.print("cm");
Serial.println();
delay(200);

//Repeat algorithm for sensor B
pinMode(sensorB, OUTPUT);
digitalWrite(sensorB, LOW);
delayMicroseconds(2);
digitalWrite(sensorB, HIGH);
delayMicroseconds(5);
digitalWrite(sensorB, LOW);
pinMode(sensorB, INPUT);
durationB = pulseIn(sensorB, HIGH);
inchesB = microsecondsToInches(durationB);
cmB = microsecondsToCentimeters(durationB);
Serial.print("Pin Sensor B: ");
Serial.print(inchesB);
Serial.print("in, ");
Serial.print(cmB);
Serial.print("cm");
Serial.println();
delay(200);

//If data is available to read from serial communication
if(Serial.available())
{
    // read it and store it in 'val'
    val = Serial.read();
}

```

```

// if no objects are close to sensors (no collision
// possibility), read inputs from 'val'.
if(val != -1 && control == 0 && !(cmL <= 12 || cmR <= 12 ||
cmB <= 12))
{
    switch(val)
    {
        //LED on
        case '1':
            digitalWrite(ledpin = 13, HIGH);
            digitalWrite(ledpin = 12, HIGH);
            Serial.println("12 on");
            Serial.println("13 on");
            delay(1000);
            break;

        //LED off
        case '0':
            digitalWrite(ledpin = 13, LOW);
            digitalWrite(ledpin = 12, LOW);
            Serial.println("12 off");
            Serial.println("13 off");
            delay(1000);
            break;

        //Move backwards
        case 'w':
            Motor1(speed,true);
            Motor2(speed,true);
            Serial.println("motor w");
            break;
    }
}

```

```

//Move forwards
case 'x':
    Motor1(speed,false);
    Motor2(speed,false);
    Serial.println("motor x");
    break;

//Turn left. Motor1 moves backwards, Motor2 forwards
case 'a':
    Motor1(200,false);
    Motor2(200,true);
    Serial.println("motor a");

    //Stop to allow camera to re-target object
    //if lost during movement
    delay(100);
    Motor1(0,true);
    Motor2(0,true);
    delay(500);
    break;

//Turn right. Motor1 moves forwards, Motor2 backwards
case 'd':
    Motor1(200,true);
    Motor2(200,false);
    Serial.println("motor d");

    //Stop to allow camera to re-target object
    //if lost during movement
    delay(100);
    Motor1(0,true);

```

```

        Motor2(0,true);
        delay(500);
        break;

//stop movement
case 's':
    Motor1(0,false);
    Motor2(0,false);
    Serial.println("motor s");
    break;

//change from Manual control to Autonomous control
case 'o':
    control = 1;
    Serial.println("control 1");
    Break;
}
}

// Collision detection
//object too close to back sensor
if (cmB <= 12)
{
    Serial.println("sensor forward");
    // move forwards (away from object)
    Motor1(speed,false);
    Motor2(speed,false);
    delay(500);
    //Stop
    Motor1(0,false);
    Motor2(0,false);
    delay (200);
}

```

```

    }

    //If object too close to sensors in front of tank
    if (cmL <= 12 || cmR <= 12)
    {
        Serial.println("sensor backward");
        //Move backwards (away from object)
        Motor1(speed,true);
        Motor2(speed,true);
        delay(500);
        //Stop
        Motor1(0,false);
        Motor2(0,false);
        delay (200);
    }
} //End of loop function

//PWM functions to control motor speed and movement
//Motor1 function
void Motor1(int pwm, boolean reverse)
{
    analogWrite(EN1,pwm);
    //set pwm control, 0 for stop, and 255 for maximum speed
    if(reverse)
    {
        //High for backwards movement
        digitalWrite(IN1,HIGH);
    }
    else
    {
        //Low for forwards movement
        digitalWrite(IN1,LOW);
    }
}

```

```

    }
}

//Motor2 function
void Motor2(int pwm, boolean reverse)
{
    analogWrite(EN2,pwm);
    if(reverse)
    {
        digitalWrite(IN2,HIGH);
    }
    else
    {
        digitalWrite(IN2,LOW);
    }
}

//Function to convert bounce-back time to distance in inches.
/*Sound travels at 1130 feet/second, or 74 microseconds/inch.
This gives the distance travelled by the pulse, outbound and
return. Therefore, dividing by 2 gives the distance*/
long microsecondsToInches(long microseconds)
{
    return microseconds / 74 / 2;
}

//Function to convert bounce-back time to distance in cm.
/*The speed of sound is 340 m/s or 29 microseconds per
centimeter. Dividing by 2 again to find distance */
long microsecondsToCentimeters(long microseconds)
{
    return microseconds / 29 / 2;
}

```

}