Designing and Building a Remotely Operated Car

Shea Ketsdever

1 Abstract

This project explores creating an Arduino and XBee system for controlling a remotely operated car. User input from a joystick is detected and encoded by a controlling module, which remotely transmits data to a receiving module that decodes these commands and instructs the movement of the vehicle. Each Arduino communicates serially with an XBee radio module that facilitates communication between the controlling and receiving ends.

2 Introduction

2.1 Inspiration and Goal

The goal of this project was to implement an original design for a remotely operated car, focusing especially on the software and electronics. It was inspired by previous student work and a personal interest in becoming more familiar with the electronic and software side of engineering. For this reason, the project had two main goals: to construct a functioning RC car, and to master software and related skills that will no doubt be useful in future endeavors. By constructing the car and writing code from scratch, new suggestions could be made to eliminate redundancy and improve the efficiency of current designs.

This paper was written for Dr. James Dann's Applied Science Research class in the spring of 2015.

2.2 Process

2.2.1 Initial Familiarization and Electronic Prototyping

Initially, a significant amount of time was invested in familiarizing myself with Arduino and prototyping the basic electronic set-up. Instructional manuals and online resources provided an excellent introduction to operating the microcontroller, and sample programs were implemented to gain more insight into the nuances of coding with Arduino. The first major milestone in this regard involved getting an LED to blink on and off repeatedly. Following these simple experiments, a potentiometer was added and tested to begin prototyping the electronics and code involved in manually controlling the car. The potentiometer was connected to the Arduino as an input, and LEDs were attached on a breadboard as outputs. Code was then implemented to turn the LEDs off and on according to directions from the potentiometer; testing indicated that communication was successful.

Next, more electronics were added to the existing setup to test whether a motor could be made to spin based on input from the potentiometer. A transistor was added to the circuit to receive output current from the Arduino when the potentiometer was moved into a certain orientation. When activated, the transistor allowed current from an external power supply to flow through an attached motor, making it spin. After much tweaking, the transistor and motor setup functioned successfully.

At this point, all other motors could have been attached in similar fashion with transistors; and with more work on the mechanical end, the car could have been made to move forwards and even turn. However, this design did not yet accommodate for propelling the car backwards, so the electronics were further adapted to include an H-bridge and allow for the motors to spin in both directions, enabling the car to move both forwards and backwards. First the electronics were isolated and tested on a separate breadboard in the simplest state possible (i.e. no Arduino input). After various tweaks and testing, the H-bridge operated a motor and was deemed successful. These electronics were then attached to the Arduino system so the motor could be flexibly operated based on input from the joystick. Finally, these electronic components

were soldered to a protoboard, minimizing their overall size and allowing them to be integrated more compactly into the overall design of the car. After testing and desoldering some loose connections, the board was deemed reliable enough to add to the car.

2.2.2 Mechanical Construction

Next, the focus shifted to constructing the physical car using components from the DAGU car kit. Four motors were attached inside the frame, and each was outfitted with one wheel. The electronics were then mounted on top of the frame using tape, and some small wood scraps from the laser cutter were used to construct a small mount so the wires would not interfere with the wheels. For testing purposes, a small wood block was acquired and placed underneath the frame of the car to elevate the vehicle off the ground while allowing the wheels to spin freely so that more intricate adjustments could be made.

2.2.3 First Prototype: Wires

With the vehicle functioning, electronics could be finalized, attached, and tested in conjunction with the car. A second H-bridge was added along with the appropriate circuitry so the remaining two motors could also be controlled. 9V batteries were also substituted for the power supplies to eliminate the need for external power and make it possible for the electronics on board to be self-contained.

Longer wires were attached from the joystick to the Arduino on board so the car could be manipulated from a distance away. This made it possible to place the car on the ground and drive it long enough to test steering capabilities. Testing revealed that the car achieved a relatively high velocity and was very responsive to commands from the joystick. While presenting the progress on this project at the check-in meetings conducted at the beginning of each month, a concern was raised about using all four motors to drive the car. It was suggested that this might be counter-productive because if the motors behaved slightly differently, they would act against one another. In light of this, a small wooden axle was constructed that connected the two rear wheels through a ball bearing to prototype two-wheel drive. However, issues did not seem to

arise during initial testing of the four-wheel drive method, so the two-wheel method was abandoned.

2.2.4 Second Prototype: Serial

Although the first prototype was perfectly functional, excessive wiring made the product inelegant, so serial communication was explored and implemented to improve the design. Serial communication required both software and electronic adjustments. Electronically, a second Arduino was added and the joystick attached to create a controlling module. The initial Arduino was repurposed as the receiving module and all components remained intact after the joystick was removed. Serial communication was then facilitated between the controlling and receiving Arduinos by connecting the grounds and the RX and TX pins on the two microcontrollers (RX1 to TX2, RX2 to TX1).

In terms of software, the initial code was split into receiving and controlling code. The controller code interpreted joystick commands, while the receiving code dictated commands to the H-bridges. A small amount of encryption was also implemented to send the data between the controlling and receiving Arduinos.

2.2.5 Final Product: XBee

The XBee radio modules were configured using XCTU, a software program that allows the user to manipulate the values for various features on the XBee. It also provides a console command line feature to facilitate communication with the radios. After installing FTDI drivers so the XBee could be recognized by the computer, the baud rates and addresses were appropriately configured so the XBees could communicate with one another.

Next, the XBees were placed on the Arduinos and connected via the Arduino serial ports. The code was altered slightly to accommodate for XBee replacing the built-in serial function, and a few other tweaks were made to improve efficiency.

In addition, during the final step of implementing XBee, the original electronics began malfunctioning, probably due to some loose connections on the protoboard. Debugging attempts did not solve the issue, so electronics were transferred to a breadboard for the final design.

2.3 History

Nikola Tesla is widely acknowledged for his innovation in the field of radio communication. In 1891, he developed the Tesla coil, a device that enabled wireless transmission of electricity. Large amounts of current are pumped through a capacitor in a primary circuit, which creates a magnetic field that collapses quickly and generates electricity in a surrounding secondary circuit, and sparks are emitted as the voltage rapidly passes through the air between the two coils. Finally, enough charge builds up in the capacitor in the secondary circuit and is released in a huge burst of electric current. Unlike conventional transformers, the Tesla coils are much looser and have a large air gap, enabling them to function at much higher voltages. Ideally, a phenomenon known as resonance is achieved where the timing of energy transfer between the primary and secondary coil is optimized to maximize the energy passed to the secondary coil.

The Tesla coil paved the way for future advancements; when tuned to the same frequency as an incoming radio wave, the resonance of the coil allowed it to magnify the electrical energy of the signal. Through this method, powerful radio signals could be wirelessly received and transmitted through the coil. Tesla used this principle to create the first radio controlled device, a remotely operated boat that he introduced at the Electrical Exhibition of 1898 in New York City's Madison Square Garden. The heavy, four-foot-long steel boat was powered by batteries on board, and switches connected to its propeller and rudder could be activated wirelessly to maneuver the vessel. Most importantly, it had a radio antenna on board that could receive exactly one radio frequency. Tesla created a specialized radio-activated switch, or "coherer," to manage radio communication: a canister containing metal oxide powder that would orient itself in the direction of a magnetic field and become conductive in its presence. When activated, the coherer would advance the position of an adjoined disk that instructed the switches on board to change state and successfully maneuver the boat.

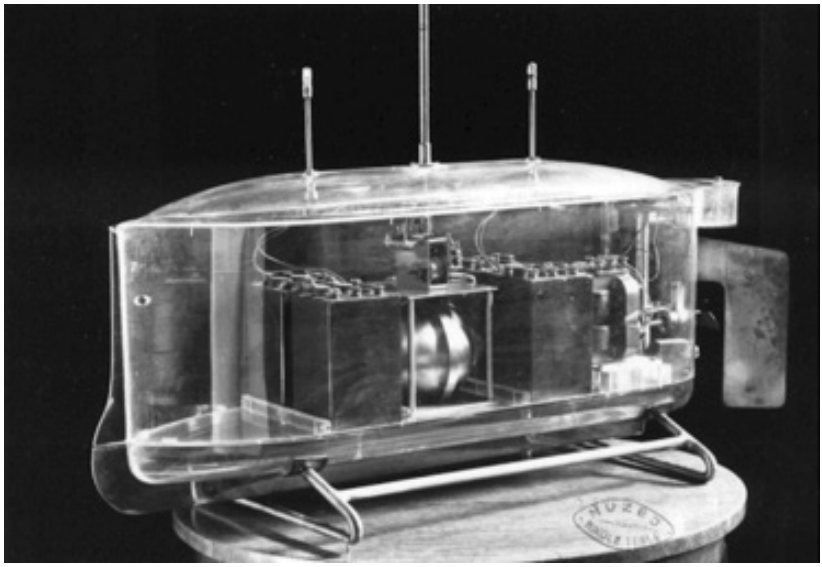

Figure 1: *A model of the "Teleautomaton," Tesla's radio controlled boat.*

Initially, Tesla's technology was not as widely adopted as he expected. Early pioneers included Leonardo Torres-Quevado, a Spanish engineer who used wireless telegraphers to successfully control an engine-powered boat in the early 1900s, and Archibald Low, who implemented radio control in an airplane in 1917. The popularity of radio increased dramatically during the First and Second World Wars when aircraft drones, radio-operated tanks, and guided missiles became instrumental to military operations. The popularization of transistors in the 1960s allowed for significant reductions in the size of electronic equipment and made radio-controlled devices more commercially accessible. Prior to transistors, relays and reed switches were used as toggle switches to control devices; but transistor-based receivers replaced this earlier approach because they took up significantly less space, sometimes reducing the size of electronic components by as much as half. Furthermore, transistors lowered current requirements at low voltages, replacing the need for high voltage batteries that had imposed many limitations. In the 1970s, multi-channel systems introduced further sophistication by making the movements of devices proportional to the movements of control sticks.

Recently, one big improvement has been the introduction of spread spectrum technology. The crystal radio receivers that were previously in use had several limitations; namely, they required that the receiver and transmitter have matching crystals, and they only used one frequency. This presented issues if two systems were operating on the same frequency, because the signals would interfere with one another and inhibit both systems from communicating appropriately. Spread spectrum technology, however, increases the bandwidth of a particular signal and thus significantly reduces interference (while also eliminating the need for matching sets of crystals).

Advances in battery technology have also motivated improvements in the radio industry. Rechargeable NiCd batteries predominated in the early stages of radio-operated devices, and while they were relatively durable, their weight presented significant restrictions. For this reason, lighter NiMh batteries were soon introduced and became especially useful in powering planes and other aircraft that had been weighed down by previous versions. Later on, the introduction of LiPo batteries marked another revolution in battery technology. While more delicate and hazardous than earlier models, LiPo batteries had both more voltage and more amperage per cell than their predecessors, making them ideal power sources for many devices.

The wide applications of radio make the topic and this project especially interesting: gaining an understanding of remote communication could be useful in many fields. Military technology, commercial toys, and wireless communication are just a few options. Gaining more expertise on radio will no doubt be useful in many future endeavors.

3 Design

3.1 Mechanical

The mechanical design of the RC car is intentionally simplistic, to allow for more focus on electronics and to reduce potential for error. For this reason, the car is built using parts from the DAGU electronic multi chassis-4WD (DG012-ATV) kit:

Figure 2: *Image of the DAGU car. Battery containers and wiring are not included in the design for this project.*

More specifically, four 48:1 DC motors are attached to four 78 mm high-profile rubber wheels and placed inside a 2.5 mm thick aluminum frame. The following CAD drawings provide a more detailed visual of the car and components (all dimensions in mm):

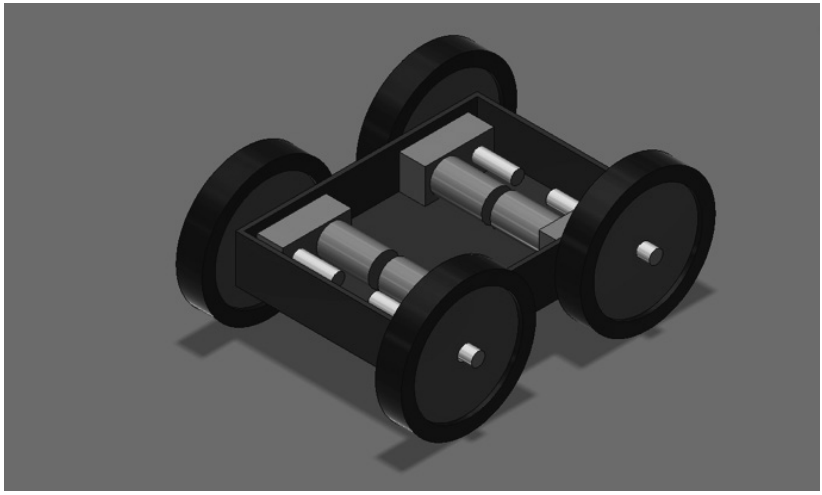

Figure 3: *Slant view of car.*

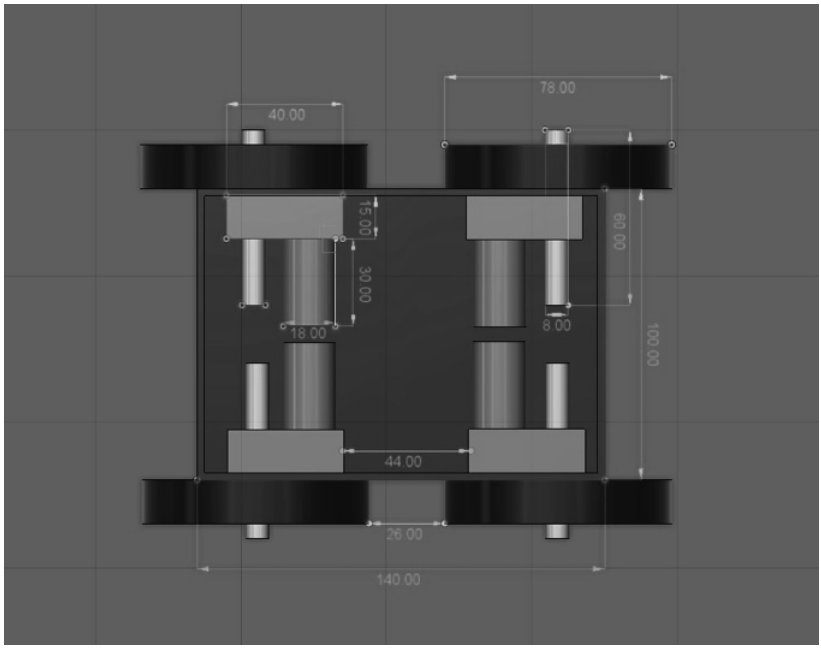

Figure 4: *Top view of car.*

Electronic components including a breadboard, Arduino, and an Xbee receiver and corresponding breakout board are stored on the car, along with three 9V batteries to power the motors and Arduino. Two batteries are used to power the four motors, with one battery corresponding to each H-bridge (which in turn controls two motors); the remaining battery powers the receiving Arduino and XBee module. These components are situated on top of the frame of the car and secured with tape.

3.2 Electrical and Software

Communication with the car is facilitated by the Arduino microcontroller and software. A potentiometer (joystick), attached as an analog input, sends the voltages corresponding to its horizontal and vertical orientation to the controlling Arduino. These values are compared to high and low cut-off thresholds to determine whether the joystick is displaced from its initial position to the extent that warrants action.

More specifically, testing revealed that the Arduino reads in potentiometer voltage values from 0-1023; the horizontal and vertical values are both approximately 500 when at rest. To account for variation in these voltage values and ensure that small accidental movements of the joystick were not interpreted as instructions for the car to move, cut-off values of 200 above and below this equilibrium position were selected. The Arduino therefore responds to voltages above 700 and below 300 and disregards values in the intermediate range as noise. These voltage inputs from the potentiometer correspond to nine specific actions; however, only seven are executable. Moving directly to the right or to the left is unattainable because it is not possible for the car to move completely perpendicular to its current trajectory. So if the joystick is maneuvered directly to the right or left, the Arduino defaults to moving forward-right or backward-right:

Vertical voltage	**Horizontal voltage**	**Action**
Between 300 and 700	Between 300 and 700	No action
< 300	Between 300 and 700	Move backward
> 700	Between 300 and 700	Move forward
< 300	< 300	Move backward-left
> 300	> 700	Move backward-right
> 700	< 300	Move forward-left
> 700	> 700	Move forward-right

This logic table is translated into a series of "if" statements in the Arduino code that determines which action, if any, should be performed based on the current input from the potentiometer. The controlling Arduino encodes these action commands into a series of numbers ranging from 0 to 6. Each number corresponds to one of the seven executable commands:

Encoded value	Action
0	No movement
1	Forward
2	Forward-left
3	Forward-right
4	Backward
5	Backward-left
6	Backward-right

These numbers are then sent remotely from the controlling Arduino to the receiving Arduino by way of XBee. Each XBee uses serial communication to connect to its respective Arduino, and wireless radio communication to transmit and receive data with its partner. Essentially, data travels serially from the controlling Arduino through its XBee module, then wirelessly to the other XBee module, and finally serially to the receiving Arduino (see section 4.5, "Serial communication and XBee" for specific details). The receiving Arduino then reads in this data and decodes the numbers to determine which action they refer to. If an action is required, the actions of the four motors on board must combine to create the desired effect. Each motor is instructed to move forwards, backwards, or not at all:

Desired Action	M1 (front left)	M2 (front right)	M3 (back left)	M4 (back right)
No action	No action	No action	No action	No action
Move backward	Backward	Backward	Backward	Backward
Move forward	Forward	Forward	Forward	Forward
Move backward-left	Backward	Forward	Backward	Backward
Move backward-right	Forward	Backward	Backward	Backward
Move forward-left	Backward	Forward	Forward	Forward
Move forward-right	Forward	Backward	Forward	Forward

It should be noted that there are many possible steering techniques that can be used when maneuvering a car. In front-wheel drive, the front two tires rotate during turns while the back two tires continue going either straight forward or backward. Conversely, back-wheel drive involves the back wheels turning while the front wheels remain straight. Front-wheel drive is implemented in this project, although at this point the choice is mainly arbitrary. Testing does not suggest flaws in this strategy, but back-wheel drive is an equally valid alternative.

After translating the voltage from the potentiometer into an action and determining the corresponding instructions for each motor, the receiving Arduino sends voltage outputs to the logic pins of the two H-bridges controlling the motors (two motors per bridge). When a voltage difference is successfully created between these pins, the motor spins in the desired direction and propels the car. One of the advantages of H-bridges is that their circuitry makes it possible to rotate the motors in either direction (described in greater detail below). In this context, this means that the Arduino raises one logic pin high (outputs 5V) and one low (0V) or vice versa to make the motor spin in the opposite direction. H-bridges also utilize built-in transistors to handle higher current flow required to power the motors.

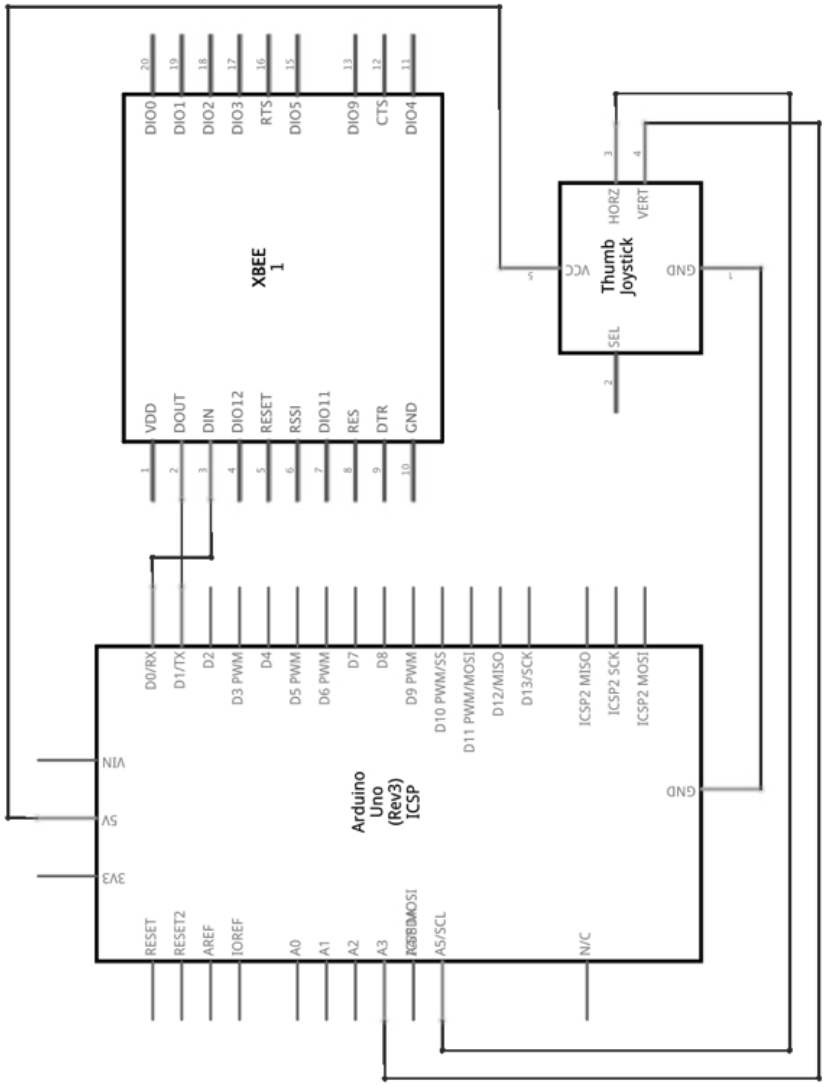

Figure 5: *Circuit diagram for controller with Arduino, joystick, and XBee.*

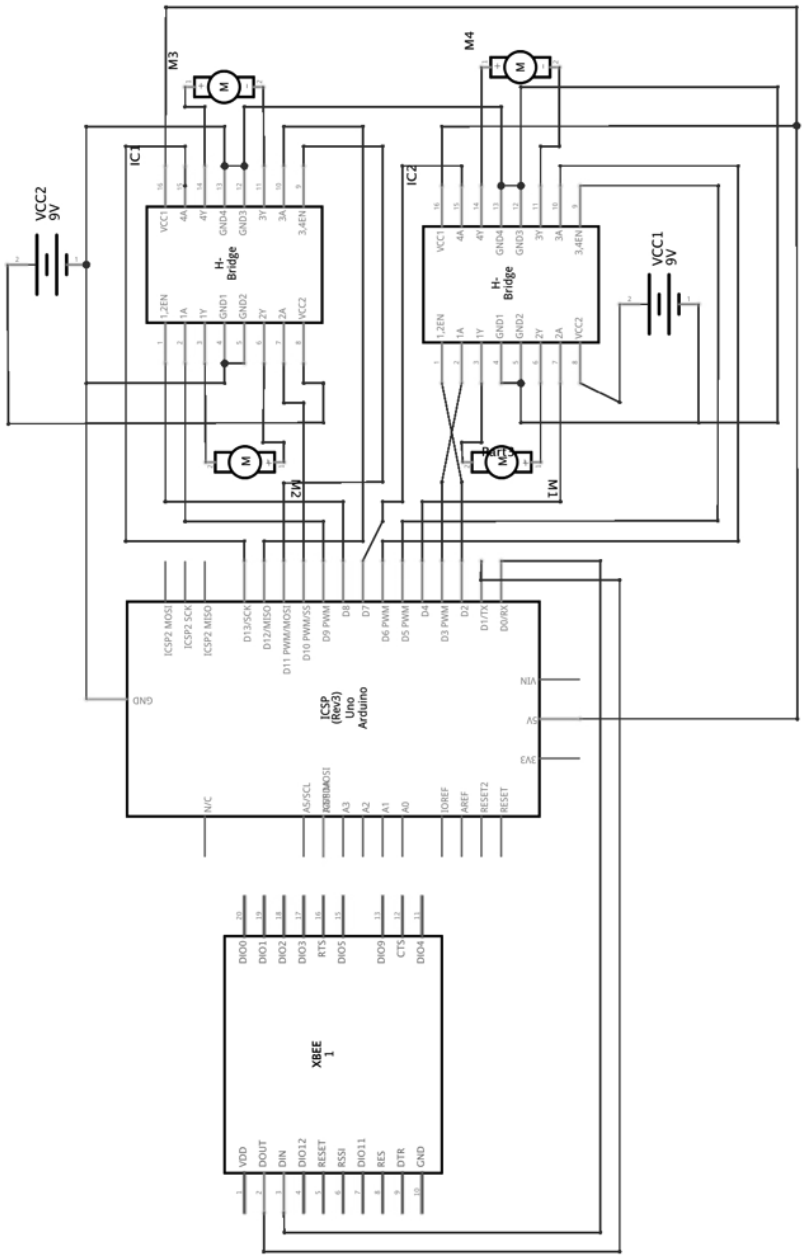

Figure 6: *Circuit diagram for receiver with Arduino, H-bridges, motors, batteries, and XBee.*

4 Theory

4.1 Potentiometer

A potentiometer is a three-terminal resistor that utilizes a sliding component to create an adjustable voltage divider. Potentiometers have three terminals: two "ends" and one "wiper" in the middle. The wiper is a moveable contact that slides along a resistive element, most commonly graphite. A mechanism attached to the wiper can be manipulated externally (e.g., toggling a joystick back and forth) to move the wiper along the resistive element. The wiper is also connected to one or both of the "ends," so as it moves across the resistive element the changing voltage can be measured. The closer the wiper gets to an end, the distance between the two decreases, which decreases the path of the current and therefore the resistance.

Figure 7: *An image of a standard potentiometer.*

In this project, potentiometers are present in the design of the SparkFun Thumb Joystick that enables the user to navigate the car. The joystick contains two potentiometers, one for horizontal movement and one for vertical movement:

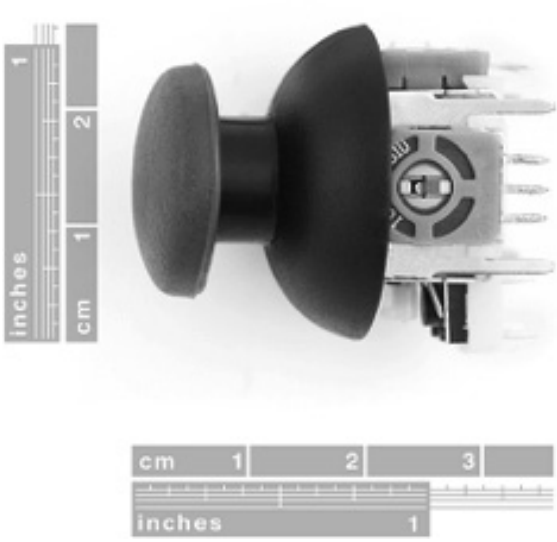

Figure 8: *The SparkFun Thumb Joystick used to direct the car, with scale markings to indicate size.*

4.2 Arduino Microcontroller

Microcontrollers are tiny computers. They contain a processor core, memory, and input/output channels that enable the device to receive, store, manipulate, and respond to data. Input is collected from the environment through temperature sensors, motion detectors, etc. that react to the environment to produce a voltage that the microcontroller can detect. This information is stored in the memory of the microcontroller, and the processor core executes various pre-determined tasks (specified in a program) to manipulate the data. In response, output signals can be sent to control various devices connected to the microcontroller.

The Arduino microprocessor integrates its processor core, memory, and input/output slots into a small control board. As with all avr-based Arduino boards, the Uno operates on three pools of memory: flash memory, where the program sketch is stored; SRAM (static random access memory), where the sketch stores and manipulates variables as the program runs; and EEPROM where permanent long-term memory is stored. When power is removed from the control board, SRAM is lost while EEPROM and Flash remain. Flash will only persist, however, until a new sketch is uploaded to the board in replacement. The Arduino Uno in this project utilizes a 10-bit processor chip known as the ATmega238 that has 32KB of Flash, 2KB of SPAM, and 1KB of EEPROM. Because it has relatively little SPAM memory, the Uno sometimes encounters issues storing large variables like strings and can occasionally fail in unexpected ways if it reaches its limit.

The Arduino utilizes two types of input/output (I/O) pins: analog and digital. Analog pins are used only as inputs and can be connected to sensors or other devices that produce a voltage in response to environmental factors or manipulation. Analog itself is a continuous voltage range, but computers can only operate with 0's and 1's, so the value detected by the analog pins must be converted into a numerical value that the processor can understand. Although the ATmega238 only has an 8-bit processor core, the Arduino board has a 6-channel, 10-bit analog to digital converter built in, allowing the Uno to accept 1024 analog input values in the range from 0-1023. Digital pins work similarly to analog pins, but they can be used for both input and output, and have only 1 bit. This means that they have a resolution of 21 and can only operate in two states: on or off.

The Arduino platform also has its own software, called Arduino Integrated Development Environment, or IDE, which is very similar to C++ (see section 8.3 for current code).

4.3 H-Bridges

An H-bridge is an electronic circuit comprised of switching elements that allow a voltage difference to be applied across a load in either direction.

This load is typically a brushless DC motor, but H-bridges are also often used to power bipolar stepper motors. The circuit is relatively simple:

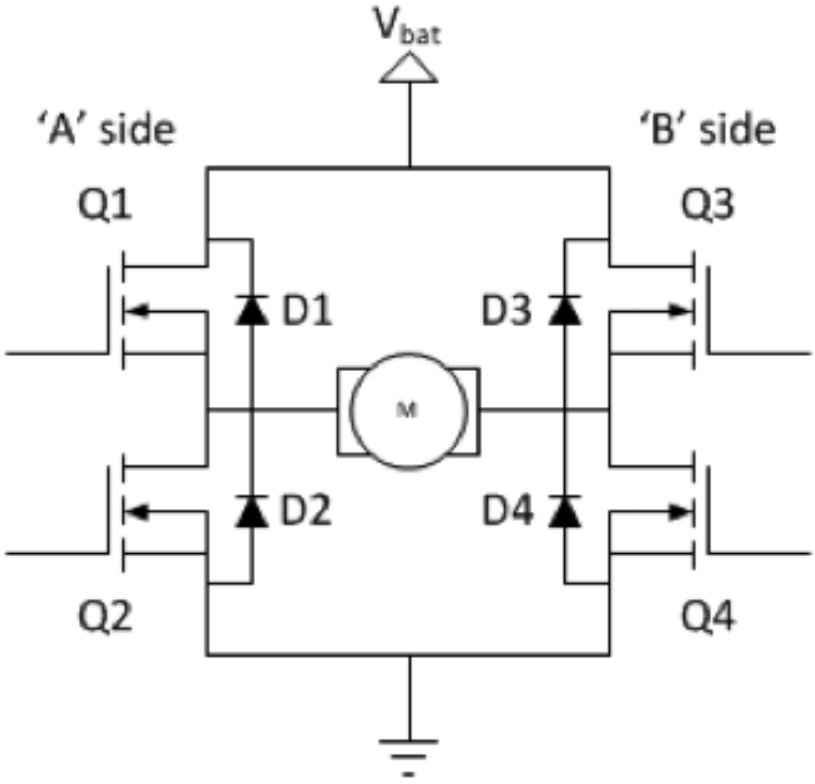

Figure 9: *Circuit diagram for an H-bridge.*

The switches can be turned on and off independently to affect the behavior of the load. To energize the load, one "pair" of switches must be on while the other is not. There are two pairs of switches in an H-bridge (in the diagram, the pairs are S1 & S4 and S2 & S3), so there are two ways in which the load can be energized. If S1 and S4 are turned on,

the left lead on the load will be connected to power and the right will be connected to ground, creating a voltage difference across the load in one direction. When S2 and S3 are activated, the voltage difference will be created in the opposite direction, energizing the load in reverse. It is not necessary that the switches act only in pairs, although this provides the most commonly desired results of energizing the motor in one direction or another. However, the high-side (S1 and S3) and low-side (S2 and S4) switches should not be activated at the same time, otherwise the path between power and ground would have very low resistance and result in a "shoot-through." In this scenario, either a short-circuit will lead to a loss of power, or excessive amounts of current will heat up the H-bridge and most likely cause damage to the electronics. Possible permutations of switching states include:

S1	**S2**	**S3**	**S4**	**Result**
1	0	0	1	Voltage applied in one direction
0	1	1	0	Voltage applied in other direction
0	0	0	0	Free run
1	0	1	0	Stop
0	1	0	1	Stop
1	1	1	1	Shoot-through
1	1	0	0	Shoot-through
0	0	1	1	Shoot-through

A dual L238D H-bridge is used to control the motors in this project. The L238D has two bridges built in, so it can control up to two motors at once (one on each side). Controlling multiple motors with the same H-bridge is very useful because it significantly decreases the size of the electronics and simplifies the circuitry:

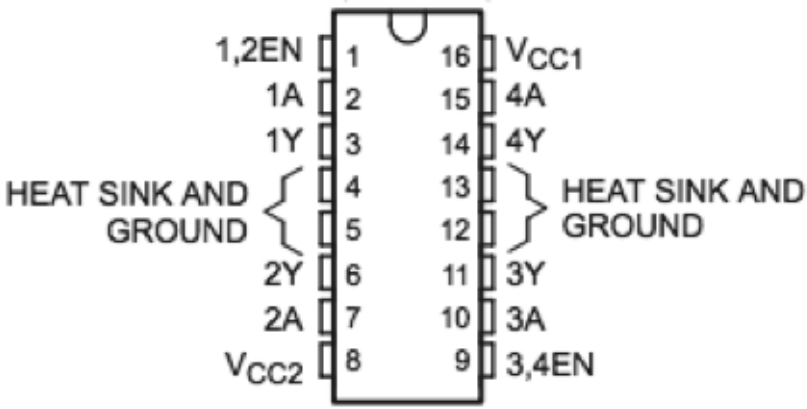

Figure 10: *A diagram from the data sheet for the L239D H-bridge used in this project. Vcc2 is the external power supply for the motor. Vcc1 is the 5V power from the Arduino. The left side controls one motor and the right side controls another. Each has an enable pin (1,2EN and 3,4EN) that must be activated for the bridge to be functional. 1A, 2A, 3A, and 4A are logic pins that take output voltages from the Arduino. Raising one high and the other low creates a voltage difference that turns the motors, which are connected to 1Y, 2Y, 3Y, and 4Y. Each motor requires two logic pins and two motor pins, consistent with the 4-switch circuitry of the H-bridge that controls it.*

4.4 DC Motors

Electric motors turn electrical energy into mechanical energy through fundamental physical laws such as magnetism. The premise is that moving electrical charge, or current, creates a magnetic field directed perpendicular to the direction of its movement. This concept is utilized in the electromagnets of the motors, which are comprised of numerous coils of wire wound tightly together around a core. A battery or other power source is connected to a circuit containing the electromagnet, and the voltage difference across this power supply causes current to

flow through the circuit and the electromagnet, thus creating a relatively consistent and uniformly directed magnetic field in the coils of wire. Specifically, this project uses the mini DC GearBox DG01D dual-axis drive gears from the DAGU car kit:

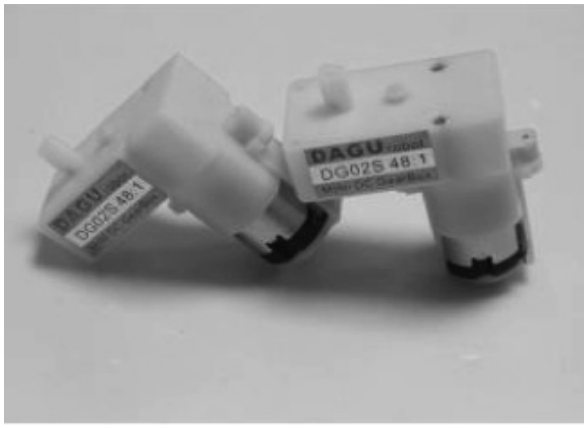

Figure 11: *Two of the DAGU motors used in this project.*

4.5 Serial Communication and XBee

Serial processing involves sending data sequentially one bit at a time over a communication channel. It is often used as an alternative to parallel communication, which involves sending several bits at once linked across several parallel channels. Serial communication is often preferable when sending data over a long period of time, or when it is challenging to effectively synchronize cables. This is because serial communication can often be accomplished using only one wire, which reduces the cost of cable and simplifies many designs.

The Arduino microcontroller uses serial processing to communicate between the board and external devices (computers, external modems, etc.). Arduino boards all have at least one serial port (known as UART or USART), which communicates using digital pins 0 (RX) and 1 (TX). When connected to a computer, Arduino software includes a built-in serial monitor that is useful for run-time analysis and debugging. This

serial port can also be repurposed to enable serial communication between the Arduino and any other device.

In this project, serial communication is instrumental in connecting each Arduino with its respective XBee, which is in turn responsible for wirelessly transmitting and receiving data. The most straightforward way to facilitate Arduino-XBee communication is to use a breakout board that internally connects RX and TX pins for the Arduino UART port to the corresponding pins on XBee. These pins could hypothetically be connected manually, but the breakout board has the advantage of simplifying design, and it also handles several other functions like converting from the Arduino's 5V to the 3.3V required for XBee.

After serial communication is established between Arduino and XBee, the two XBees must be configured to communicate with one another. Configuration serves two main purposes: ensuring that each XBee is individually capable of sending and receiving data, and synchronizing their settings so they are able to communicate with their intended partners. Specifically, the configuration process involves setting the baud rates and parity modes, along with many other features. This project utilizes the XCTU software program to configure XBee; XCTU is essentially a terminal program that allows the user to detect and set basic XBee settings like those mentioned above. Each XBee must be configured separately, and they can then be attached to their respective Arduinos, where they receive or transmit data wirelessly to their XBee partner and serially to their Arduino.

5 Results

The goal of this project was to gain familiarity with the software and electronics behind the operation of the car, so measurable results are not the primary focus. That said, a few metrics regarding the performance of the vehicle are included to give an idea of its functionality and explain some programming choices.

In the code for the receiver, there is a "delay()" command that tells the Arduino to pause momentarily. This line is inserted to provide the motors enough time to actually begin moving in the intended direction before they are reset and cease rotating. The length of this delay was observed to affect the speed at which the car responded to the controller, as well as the velocity of the car, so measurements were taken of both response time and velocity for different delay times to examine these relationships.

To test velocity, a distance of two meters was measured out and marked with tape. The car was placed at the start line and the time it took for the car to travel the two meter distance was recorded. This data was used to calculate the velocity of the car, and the process was then repeated for each different delay time. To test response time, the joystick was manipulated into the forward position and the time it took for the motor to begin rotating was recorded (see section 8.2 for data tables).

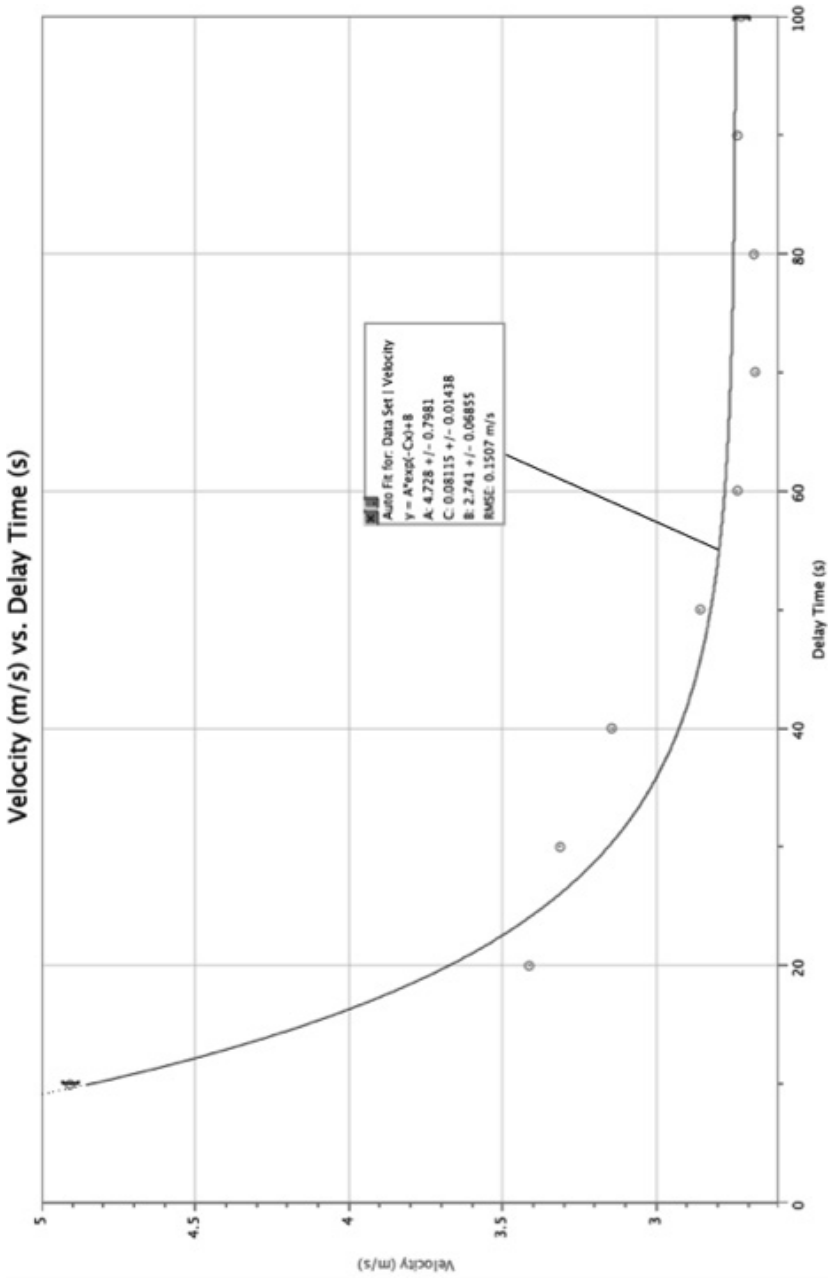

Figure 12: *Relationship between velocity and delay time.*

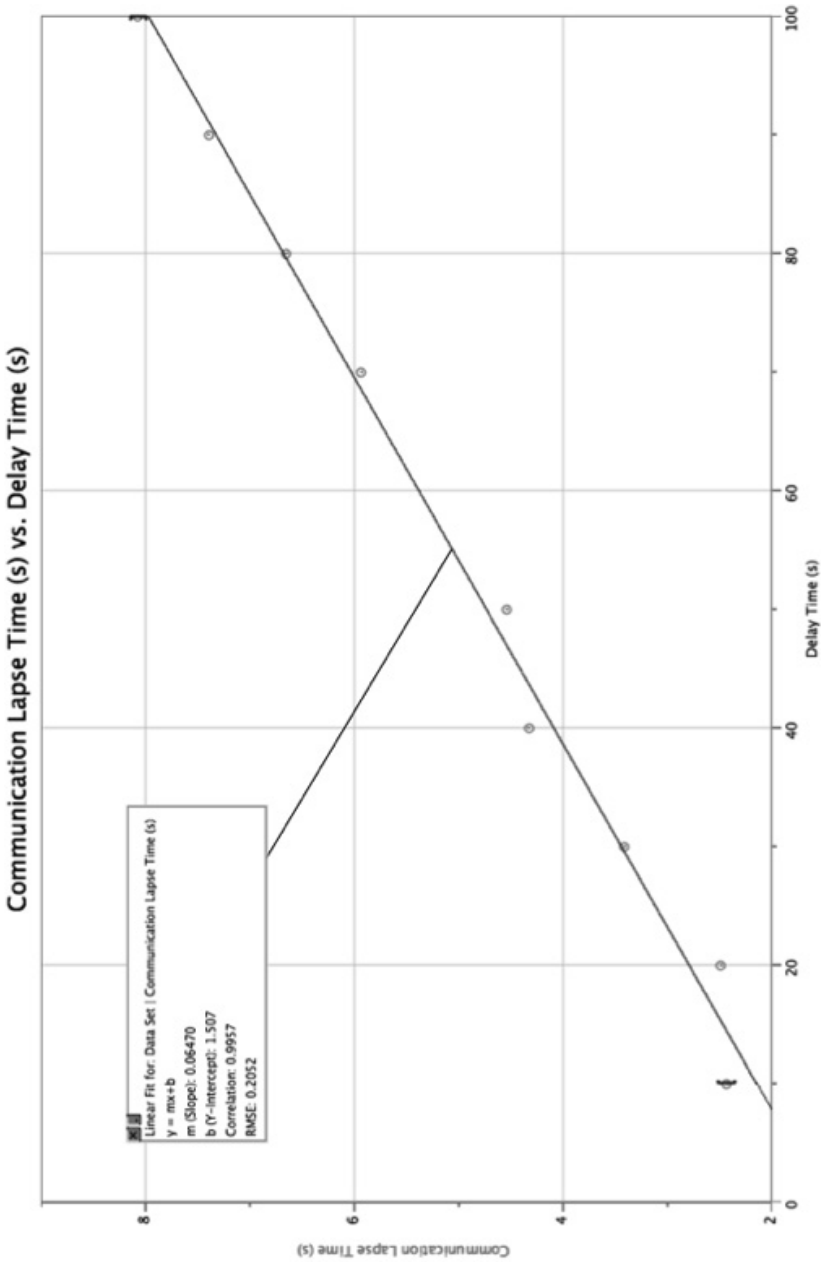

Figure 13: *Relationship between length of communication lapse between receiver and controller and delay time.*

As anticipated, velocity and lapse in communication appear to be directly proportional to delay time. However, it can be observed that the velocity reaches a plateau at approximately 2.7m/s, and hovers around this value for delay times of ~60s and greater. This indicates that increasing the delay time further would not have any positive impact on velocity, and would only result in longer communication delays. Therefore, to optimize velocity and the speed of communication between the controller and receiver, the delay time was set at 60 seconds (the point where velocity began to plateau).

6 Acknowledgements

Dr. Dann: thank you for all your tips throughout the painful process of debugging electronics, and for ordering those snazzy new XBee parts.

Whoever introduced me to the black soldering iron with the really small tip: that is my new favorite piece of equipment in the Whitaker Lab and saved me from short-circuiting everything, so thank you thank you thank you! ●

7 Bibliography

Carrino, Nico. *Coding, Developing, and Controlling a Radio Controlled Car*. 2013

http://www.arduino.cc/en/Reference/Serial

http://www.livescience.com/46745-how-tesla-coil-works.html

http://www.pbs.org/tesla/ll/ll_whoradio.html

http://www.pbs.org/tesla/ins/lab_remotec.html

http://www.tfcbooks.com/articles/muzej.htm

http://science.howstuffworks.com/innovation/repurposed-inventions/history-of-remote-control.htm

http://rcflightline.com/rc-history/

http://www.rlx.sk/sk/robot/2582-multi-chassis-4wd-kit-atv-sparkfun-rob-12090.html

http://www.brighthubengineering.com/commercial-electrical-applications/47625-potentiometers-explored-construction-and-working-principles/

http://en.wikipedia.org/wiki/Potentiometer#mediaviewer/File:Potentiometer.jpg

https://www.sparkfun.com/products/9032

http://electronics.howstuffworks.com/microcontroller.htm

http://arduino.cc/en/Tutorial/Memory

http://www.societyofrobots.com/microcontroller_tutorial.shtml

http://www.atmel.com/devices/atmega328p.aspx

http://arduino.cc/en/pmwiki.php?n=Reference/AnalogRead

http://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/

http://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridge-control/

http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_1341966_-1

https://itp.nyu.edu/physcomp/labs/motors-and-transistors/dc-motor-control-using-an-h-bridge/

http://www.dagurobot.com/goods.php?id=86

https://learn.sparkfun.com/tutorials/serial-communication

https://learn.sparkfun.com/tutorials/exploring-xbees-and-xctu

8 Appendix

8.1 Parts List

Part Description	Quantity	Use	Cost	Provider
Car kit (includes frame, wheels, motors, various attachment pins)	1	Components for physical car	Already own it	DAGU (http://www.rlx.sk/sk/robot/2582-multi-chassis-4wd-kit-atv-sparkfun-rob-12090.html; other sources also exist)
Thumb Joystick	1	Allows user to direct movement of car	Already own it	SparkFun (https://www.sparkfun.com/products/9032)
L239D H-bridge	2	Electronic component controlling current to motors	$2.95/unit	Jameco (http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_1341966_-1)
SB300 Solderable PC Breadboard	1	Small protoboard to compactly house circuitry	Already own it	Jameco (http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_1341966_-1)
Male-male and male-female hookup wires	Several	Basic wiring for circuit	Already own it	SparkFun (https://www.sparkfun.com/products/9194)

9V batteries	3	Power supply for Arduino and motors	$8.99/ 4 batteries	Amazon (http://www.amazon.com/Energizer-E522-Alkaline-battery-later/dp/B00MNRYY0A/ref=sr_1_10?s=hpc&ie=UTF8&qid=1426618205&sr=1-10)
XBee pro	2	XBee device to facilitate radio communication	$37.95	SparkFun (https://www.sparkfun.com/products/8742)
XBee shield	2	Breakout shield that allows the XBee to interface with the Arduino	$14.95	SparkFun (https://www.sparkfun.com/products/12847)
XBee explorer dongle	1	Smaller breakout board with USB dongle to connect XBee with computer (used for testing)	$24.95	SparkFun (https://www.sparkfun.com/products/11697)

8.2 Results: Data Tables

8.2.1 Delay Time (s) vs. Velocity (m/s)

Delay Time (s)	Time to Travel 2 Meters (s)	Average Velocity (m/s)
10	4.93	4.907
	4.81	
	4.98	
20	3.38	3.413
	3.40	
	3.46	
30	3.36	3.313
	3.28	
	3.30	
40	3.27	3.143
	3.10	
	3.06	
50	2.83	2.857
	2.76	
	2.98	
60	2.80	2.737
	2.66	
	2.75	
70	2.76	2.677
	2.61	
	2.66	
80	2.56	2.680
	2.68	
	2.80	
90	2.73	2.737
	2.70	
	2.78	
100	2.68	2.723
	2.78	
	2.71	

8.2.2 Delay Time (s) vs. Communication Delay Time (s)

Delay Time (s)	Lapse In Response Time (s)	Average Communication Delay Time (s)
10	2.51	2.440
	2.33	
	2.48	
20	2.43	2.497
	2.61	
	2.45	
30	3.47	3.417
	3.35	
	3.43	
40	4.31	4.327
	4.36	
	4.31	
50	4.52	4.537
	4.58	
	4.51	
70	5.90	5.933
	5.87	
	6.03	
80	6.67	6.650
	6.60	
	6.68	
90	7.43	7.393
	7.35	
	7.40	
100	8.12	8.067
	8.10	
	7.98	

8.3 Code

8.3.1 Controller

```
// include xbee
#include <SoftwareSerial.h>
SoftwareSerial XBee(2, 3); // RX, TX

void setup()
{
 XBee.begin(9600);
 Serial.begin(9600);
}

void loop()
{
 // read in values from joystick
 int verticalValue = analogRead(3);
 int horizontalValue = analogRead(5);

 // voltage cutoffs
 int lowCutoff = 300;
 int highCutoff = 700;

 // direction key
 int noMovement = 0;
 int forward = 1;
 int forwardLeft = 2;
 int forwardRight = 3;
 int backward = 4;
 int backwardLeft = 5;
 int backwardRight = 6;

 if (verticalValue < lowCutoff)
 {
  if (horizontalValue < lowCutoff) // move forward-left
  {
```

```
  XBee.write(forwardLeft);
 }
 else  if (horizontalValue > highCutoff) // move forward-right
 {
  XBee.write(forwardRight);
 }
 else // move straight forward
 {
  XBee.write(forward);
 }
}
else if (verticalValue > highCutoff)
{
 if (horizontalValue < lowCutoff) // move backward-left
 {
  XBee.write(backwardLeft);
 }
 else  if (horizontalValue > highCutoff) // move backward-right
 {
  XBee.write(backwardRight);
 }
 else // move straight backward
 {
  XBee.write(backward);
 }
}
else // no vertical movement, but could move "straight" left or right
    // (translated as forward-left or forward-right)
{
 if (horizontalValue < lowCutoff) // move forward-left
 {
  XBee.write(forwardLeft);
 }
 else  if (horizontalValue > highCutoff) // move forward-right
 {
  XBee.write(forwardRight);
 }
```

```
   else // no vertical or horizontal movement
   {
    XBee.write(noMovement);
   }
  }
}
```

8.3.2 Receiver

```
// include XBee
#include <SoftwareSerial.h>
SoftwareSerial XBee(2, 3); // RX, TX

void setup()
{
 Serial.begin(9600);
 XBee.begin(9600);

 // establish motor pins as outputs

 pinMode(4, OUTPUT);
 pinMode(5, OUTPUT);
 pinMode(6, OUTPUT);
 pinMode(7, OUTPUT);
 pinMode(8, OUTPUT);
 pinMode(9, OUTPUT);
 pinMode(10, OUTPUT);
 pinMode(11, OUTPUT);
}

void loop()
{
 // motor pin key
 int motor1Pin1 = 4;
 int motor1Pin2 = 5;
 int motor2Pin1 = 6;
 int motor2Pin2 = 7;
```

```
int motor3Pin1 = 8;
int motor3Pin2 = 9;
int motor4Pin1 = 10;
int motor4Pin2 = 11;

// direction key
int noMovement = 0;
int forward = 1;
int forwardLeft = 2;
int forwardRight = 3;
int backward = 4;
int backwardLeft = 5;
int backwardRight = 6;

// read info from xbee
int value = 0;
if (XBee.available())
{
  Serial.println("xbee available");
  value = XBee.read();
}
Serial.println(value);

if (value == forward) // straight forward
{
  Serial.println("forward");
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);
  digitalWrite(motor3Pin1, HIGH);
  digitalWrite(motor3Pin2, LOW);
  digitalWrite(motor4Pin1, HIGH);
  digitalWrite(motor4Pin2, LOW);
}
else if (value == forwardLeft) // forward-left
{
```

```
  Serial.println("forwardLeft");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);

  digitalWrite(motor3Pin1, HIGH);
  digitalWrite(motor3Pin2, LOW);
  digitalWrite(motor4Pin1, HIGH);
  digitalWrite(motor4Pin2, LOW);
}
else if (value == forwardRight) // forward-right
{
  Serial.println("forwardRight");
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);

  digitalWrite(motor3Pin1, HIGH);
  digitalWrite(motor3Pin2, LOW);
  digitalWrite(motor4Pin1, HIGH);
  digitalWrite(motor4Pin2, LOW);
}
else if (value == backward) // straight backward
{
  Serial.println("backward");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);

  digitalWrite(motor3Pin1, LOW);
  digitalWrite(motor3Pin2, HIGH);
  digitalWrite(motor4Pin1, LOW);
```

```
  digitalWrite(motor4Pin2, HIGH);
}
else if (value == backwardLeft) // backward-left
{
  Serial.println("backwardLeft");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);

  digitalWrite(motor3Pin1, LOW);
  digitalWrite(motor3Pin2, HIGH);
  digitalWrite(motor4Pin1, LOW);
  digitalWrite(motor4Pin2, HIGH);
}
else if (value == backwardRight) // backward-right
{
  Serial.println("backwardRight");
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);

  digitalWrite(motor3Pin1, LOW);
  digitalWrite(motor3Pin2, HIGH);
  digitalWrite(motor4Pin1, LOW);
  digitalWrite(motor4Pin2, HIGH);
}
else // value == noMovement
{
  Serial.println("noMovement");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
  digitalWrite(motor3Pin1, LOW);
```

```
    digitalWrite(motor3Pin2, LOW);
    digitalWrite(motor4Pin1, LOW);
    digitalWrite(motor4Pin2, LOW);
  }
  delay(60);
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
  digitalWrite(motor3Pin1, LOW);
  digitalWrite(motor3Pin2, LOW);
  digitalWrite(motor4Pin1, LOW);
  digitalWrite(motor4Pin2, LOW);
}
```