The basis of this project is to design a lawnmower that can be controlled with one’s android device. The constituents used to create this include: a lawnmower, motors, motor controller, batteries, wheels, staple components for the frame and wheels, Arduino, WiFly shield, and a wireless camera. The techniques used to create and manipulate the machine include: Soldering of the circuit designed, H-bridge, for the motor controller; Pulse Width Modulation (PWM) in the Arduino Uno for speed and direction control. The overall approach is to design an H-bridge to control the motors, and to use the android phone to control the speed and direction of the motors; a wireless camera will be mounted on top of the lawnmower, and its receiver will be connected to a TV, so one could control the lawnmower indoors. Upon completion of the project, one will be pleased that their lawn could be mowed while inside the comfort of their home.
INTRODUCTION:

Since Edwin Budding invented the world’s first lawnmower in Gloustershire, England in 1827, keeping one’s lawn trimmed and tidy is a laboring task for most people and often time consuming. This is especially true in the summertime when the weather is much hotter or when one has a physical condition that leaves one unable to mow one’s lawn, such as lumbar pain or seasonal allergies that make being outdoors unbearable.

The lawnmower has undergone many incarnations. The first lawnmowers consisted of a simple design, featuring rear-wheel cutting cylinders, with spinning reel blades attached. Later incarnations featured advancements such as gas powered rotary motors, spinning blades, as well as electric mowers for the environmentally inclined users. Now, with the emergence of smartphones and mobile computing platforms such as iOS and Android, life has become even more convenient. One has applications for checking the weather and viewing one’s checking accounts, so why not have an application that can aid in mowing one’s lawn? A lawn mower with a mounted camera that is controllable via an Android phone might be the answer to mowing one’s lawn on those hot summer days.

STEERING SYSTEM:

There are many different types of steering. There is rack and pinion steering, steering by wire, and differential steering. Rack and pinion steering deal with multiple mechanical parts such as a steering wheel, steering shaft, pinion, rack, tie rod, steering arm. The pinion is connected to the steering shaft and with the turning of the steering wheel; this will determine which direction the vehicle will turn. Steering by wire is similar to playing a video game using a joystick. Moving the joystick up will result in the vehicle moving forward. The same process works for going in reverse by having the joystick in the downward position. For moving left or right, the joystick must be tilted in the direction the user desires. For the steering system for the vehicle portion of this project, a differential steering design was chosen. This system worked by having two wheels on opposing sides. In order to be able to move forwards both wheels turn in the same direction and for backwards it is the same process except going in the opposite direction. For turning this would be done if one wheel goes forward and the other wheel is neutral. This would pivot the robot allowing it to go either left or right depending on which wheel is powered and which is neutral. The way it goes left is by having the left wheel being neutral while the right wheel goes in the forward direction and the opposite applies for going right. The way the wheels would move is implemented by two separate motors, one on each side for each wheel. The reason we chose this type of steering instead of the other types is because lawns are usually cut in an up and down pattern. With the pivoting of the robot it allows the vehicle to turn and align itself with the previous part of already cut lawn making it even cuts all across the grass. The diagram below provides an explanation of the axis of rotation. If rack and pinion steering was chosen, this type of steering would desire many mechanical parts that could weigh the robot down and would also raise the question of how to make the bot move without implementing a throttle and braking system. If steering by wire was chosen it would solve the problem of having too many mechanical components by getting rid of them but this system is very complex to the user and does not provide full rotation as differential steering provides.
Figure 1 shows how the robot would pivot around a corner. The bottom wheel would be much faster than the top wheel for the robot to turn left in this case.

**MOTOR CONTROLLER DESIGN:**

“And that was the first time I saw my parents in a month” - Parth Patel

To implement this sort of steering system, independent motor controls were required for the two motors. The motor controllers were required to allow for bidirectional speed control. The type of controller topology that was chosen is called an H-Bridge.\(^5\) The motors that were chosen had a maximum continuous load current of about 12 amps.\(^6\)\(^7\) Therefore, H-Bridge was required to be able to allow for at least 16 amps of continuous load current. While such designs could have been purchased online, this would have been very uneconomical since they were very expensive. As a result, a unique H-Bridge was designed.

The H-Bridge driver operates by allowing current to flow in either of the two directions depending on the inputs that were applied to the switching mechanisms. A general schematic is shown in figure 2.
Figure 2: Structure of an H-bridge

This configuration allowed for bidirectional control of the load current. When S1 and S4 were closed, current flows from left to right on the schematic. When switches S3 and S2 were closed, current will flow from right to left on the schematic. This sort of configuration also allowed for braking and neutral controls. When switches S1 and S3 were closed, the motor breaks since current will try to flow through both sides of the motor causing the motor to stop. When switches S2 and S4 were closed, there will be no current flowing through the load, so the motor will freely spin until it stops. Careful consideration was taken when designing an H-bridge because if switches S1 and S2 were closed, or switches S3 and S4 were closed simultaneously, the circuit would have been shorted. A complete table of possible switching configurations is shown below in table 1.

Table 1: Possible operations of an H-bridge

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Neutral</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Short</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Neutral</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reverse</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Forward</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Brake</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Short</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Short</td>
</tr>
</tbody>
</table>

To implement the switches, Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs) were chosen, because they were most suitable for high power switching purposes compared to Bipolar Junction Transistors (BJT’s). In addition to this, BJT’s require an input current in order to be turned on, whereas MOSFET’s do not require an input current. This allows MOSFETs to operate much more efficiently than BJT’s. A basic transistor level implementation is shown in figure 4.
The circuit in figure 4 was implemented using power MOSFETs for the switching transistors. In addition to this, two additional N-channel MOSFETs were used as logic inverters to prevent any of the short circuiting scenarios, mentioned above, from occurring. This topology also reduced the number of Arduino pins that would have been needed to be used. It was important to minimize the number of Arduino pins that were being used because the Arduino Uno has only six available PWM pins. In this schematic, DC power sources labeled “A” and “B” were used to represent the Analog output pins from the Arduino board.

While the above circuit demonstrated what a transistor level implementation would look like, this schematic was not a functional circuit. This was because MOSFETs require biasing circuits to function in the proper region of operation. An additional level of design needed to take place in order for the circuit to perform properly. MOSFET transistors require biasing circuits, or circuits that limit the range of DC voltages applied to the terminals, that place them in either the triode, or saturation region of operation. In this design, the transistors needed to be biased in the triode, or the linear, region of operation. This is because transistors in the triode region behave as resistors. This gave the capability of being able to use them to emulate switches. The NMOS and PMOS requirements for the triode region of operation are shown below in Equation 1 and Equation 2, respectively.

\[
i_D = K_n \left( v_{GS} - V_{TN} - \frac{v_{DS}}{2} \right) v_{DS} \quad \text{for } v_{GS} - V_{TN} \geq v_{DS} \geq 0
\]

\[
i_D = K_p \left( v_{GS} - V_{TP} - \frac{v_{DS}}{2} \right) v_{DS} \quad \text{for } 0 \leq |v_{DS}| \leq |v_{GS} - V_{TP}|
\]

The main reason that would cause the transistor to be outside the desired region of operation is that when one transistor is switched on, current can be drawn to other transistors that are desired to be off.
The solution to this problem was to introduce a large resistor between the gate and source terminals of the MOSFETs. This allowed the transistors to remain in the triode region of operation, without drawing any current into transistors that were intended to remain off. In addition to this, smaller resistors were placed before the gates of the transistors in order to prevent current from being drawn through the 2N7000 NMOS transistors, since they are not designed to be able to handle large load currents.

The circuit above was the first fully functional design. This design utilized large 51K Ohm resistors between the drain and source of the load transistors to bias the MOSFETS in the triode region of operation and also to prevent current from being drawn away from the load and through the inverters. This design also contains smaller 51 ohm resistors, which were used to prevent any short circuiting that might occur by having direct connections between transistor terminals during the final implementation. Since the motors that were chosen are 24V, and rated for 16 Amp current draw, it was necessary to design the H-bridge such that it could handle at least 16 Amps of current. To accommodate for this, IRFZ40 transistors were chosen for the NMOS transistors and IRF9Z30 transistors were chosen for the PMOS transistors. The NMOS transistors were rated for 50 Amps of continuous drain current, while the PMOS transistors had a rating for 18 Amps of continuous drain current. To prevent the circuit from being limited by the PMOS transistors, the design was modified such that there would be two PMOS transistors in parallel for each switch that was previously implemented by a single PMOS transistor in the previous design. This allows the respective switches to draw twice as much current. The final schematic is shown below:
A bulk capacitor and large resistor was placed in parallel with the entire H-Bridge circuit to regulate voltage spikes from the battery. The bulk capacitor was placed in series with a pull-down resistor, which draws current to ground, preventing accidental discharges from a capacitor. The SPICE models that were used to simulate the IRFZ40, IRF9Z30, and 2N7000 transistors are pasted in the appendix at the end of this document.

The H-Bridge was originally tested on a prototyping board for simplicity; however the final implementation was done on a blank circuit board. The normal prototyping board was insufficient for implementing the circuit because there was a very high resistance between the terminals in the board, which made it a poor choice for the final design. The blank circuit board, on the other hand, did not have any connections between the terminals, therefore, all of the components placed into the board needed to be soldered together.

In order to have a picture of what the final circuit would appear, a schematic was drawn through the website circuits.io. This schematic is below:
In the final implementation, heat sinks were also placed onto the transistors to dissipate heat due to switching losses. Both H-Bridge circuits were soldered onto one circuit board to minimize the amount of space that would be taken up in the final product.

![Figure 7: Picture of Physical H-bridge Circuit](image)

The actual implementation of the first H-Bridge onto the board went relatively smoothly, without any setbacks. However, there were a few problems that came up with the implementation of the second H-Bridge. While testing the circuit using a power supply, two of the PMOS transistors heated up; this led to the conclusion that there was an improper connection in the board. To check for short circuits, all of the connections were probed and the readings were compared to a simulation. After it became evident where the error was, the improper connections were removed using a combination of a solder pump and de-soldering wick. The components were then re-soldered in the correct layout.

The design also allowed for variable speed control of the motors through a method known as Pulse-Width Modulation (PWM). Pulse-width modulation, when applied to motor control, is a way of transferring energy through pulses rather than an analog signal. Increasing or decreasing the pulse width will regulate the energy flow to the motor. Duty cycle is the percent of time that the switch spends in the “on” state as a fraction of the total time under consideration. The longer the switch is on, the higher the power supplied is, and vice versa. The motor speeds were varied by Pulse-Width Modulation. This was controlled by turning the switch between the supply and load on and off. The main advantage in using PWM is efficiency. When a switch is on the “off” state there is almost no current, and when it’s on the “on” state, there is almost no voltage drop across the load. Thus, in both cases, power loss is almost zero. By varying the duty cycle, one effectively changes the average power applied to the motors.
Since the speed of the motors is primarily affected by the amount of power used, varying the average voltage via PWM allows one to have more fine-grained control of the motors. In the Arduino controller, we varied the PWM using the function `analogWrite(PIN, duty cycle in a scale of 0 to 255)`. The PWMs were applied to the inputs A and B in the schematic in figure 8 above.

Varying the PWM to the inputs allowed control of current flow, which dictated the direction of the motors and the speed of rotation.

Some problems that needed to be addressed were the energy losses being dissipated as heat and the speed at which the h-bridge could switch. Since the fall time and rise times of the circuit dictated that switching took at least 0.3 ms, the Arduino controller was not allowed to operate at a switching frequency faster than 3.333 KHz. Figure 9 and Figure 10 show how the rise and fall times were measured through an oscilloscope.
Testing with a function generator, there were heat problems associated with such high frequencies. This was likely due to the larger switching loss associated with faster switching. Testing more with the function generator, an acceptable compromise between heat and switching time was reached at 500 hz.
This meant that the minimum delay time between the controller and the motors was $t = \frac{1}{488 \text{ hz}} = 2\text{ms}$, an acceptable delay.\[^{12}\]

**ARDUINO CONTROL:**

“I have to go give my friend a ride” -Martin

The Arduino was used to control the flow of current in the H-bridge. The direction and speed of the motors were changed by varying the inputs to the switches using pulse-width modulation. To control the Arduino, there was an Android device that sent UDP packets to the Arduino’s WiFly module. UDP packets are used because they have latency and are stateless, making communication very lightweight.

![Figure 11: Picture of the WiFly Module](image)

The WiFly module was responsible for receiving inputs from the Android device. The WiFly module was set-up as a wireless Ad-Hoc network that the Arduino will connect. This allowed direct WiFi communication with the Arduino.\[^{15}\] After the WiFly was connected, the Android sent commands in UDP packets (on the default port 2000 and LAN IP Address 192.168.43.88), which was buffered by the Arduino WiFlyHQ library.\[^{13}\]

The Arduino was powered by a car battery; the car battery’s output voltage was stepped down from 12V to 5V using a linear regulator because higher voltages caused malfunctions with the PWM pins. Jumper wires are used to connect the Arduino to the WiFly module. A regulated 3.3 V from the Arduino powered the WiFly module and communication was done through SoftwareSerial on pins 4 and 7 of the Arduino.\[^{14}\] Hardware serial, which is faster, could have been used but it led to problems with starting and uploading sketches into the Arduino board. The UDP packets had to be processed by the Arduino to tell the motors how much speed to operate and in which direction the wheels should turn. There were PWM pins in the Arduino UNO R3 (3, 5, 6, 9, 10, and 11).
Physical pins 9 and 3 were responsible for controlling the switches in the left motor’s H-bridge and 10 and 11 were responsible for controlling the right motor. Since similar speeds were needed in both motors to move forward or backward, the left pins of each motor operated on the same timer frequency while the right pins on another timer. Operating all the PWMs on the same timer was desirable, but not possible on the UNO. In the case of this microcontroller, pins 9 and 10 were on timer1 and, pins 3 and 11 are on timer2. This means that switching (from a HIGH signal to a LOW signal) for the PWMs was done at the same frequency for each pair of pins, which was necessary to move each wheel at a constant speed. For the motors to have changed speeds, the Arduino varied the duty cycle on the physical pins using analogWrite. When the value of the write was 255, that means the duty cycle was 100% and 0 was 0%. The duty cycle ratio was thus \( \frac{x}{255} \% \), where \( x \) was the value of the write to the analog pin.

For the purposes of this report, the physical pins 9 and 3 were referred to as the left switches of the left pins of the left and right motor, respectively. The physical pins 10 and 11 were referred to the right pins of the left and right motor, respectively. When the brakes on a motor were used, both pins on the motor were set to a 100% duty cycle, meaning 0 voltage was flowing through the motor. To go forward, both motors should have set the right pins to a 0% duty cycle and the left pins to a non-zero duty cycle (in the Arduino code, the values 178, 193, 208, 224, 239, and 255 were used for a range of 70% duty cycle ratio to 100% in steps of 6%). When the lawn mower moved in reverse, the right pins of both motors had a non-zero duty cycle and the left pins of both motors had a 0% duty cycle. When the lawn mower moved forward right or reverse right, the duty cycle of the right motor’s pins were set to 0% and one of the pins for the left motor was non-zero, depending on if the user desired to go forward or reverse. When the lawn mower moved forward left or reverse left, the duty cycle of the left motor’s pins were set to 0% and one of the pins for the right motor was non-zero depending on the direction desired. When the lawn mower was in neutral mode, all the pins were set to a 0% duty cycle.

The lawn mower also had a software failsafe: if the lawn mower did not receive a valid command from the Android device in a certain time quanta (currently, this was set to 1500ms), the lawn mower would hit the brakes. The Android device constantly sent data in small time intervals so that the lawn mower knew that it did not go out of range.

![Figure 12: Picture of the Arduino Uno](image-url)
In order to communicate with the Arduino which controls the H-bridge, a controller has to be used. For the project, an Android phone was used as a controller and communicates with the Arduino by using a custom made application. This application launches the GUI that the user will utilize to steer the lawn mower as well as vary its speed and brake, if necessary. The GUI has gone through much iteration, its first design is shown below in figure 13.

![Figure 13: First Design of GUI](image)

Here the seek bar would alter the speed of the lawn mower and the Left and Right buttons are used for steering. However, this design was considered to be unintuitive and underwent modifications. It was suggested that the controller should resemble a Nintendo or Sega Genesis controller so that it would be more intuitive to use yielding the second design shown below:

![Figure 14: Second Design of GUI (buttons for speeds not shown)](image)
How this would work is if someone held the Up button, the lawn mower would move forward. If the Up and Left buttons were held, the mower would turn left and if the Up and Right buttons were held, the mower would turn right. This design seemed good in theory, but problems were encountered when trying to implement it. Before explaining the problem, one must first understand what an Android View is. A view is simply any component you see on an Android application’s screen that a user can interact with. This includes Text Fields, Buttons, Sliders, etc.

When trying to turn left by pressing Up and Left simultaneously, Android does not register this action. This is because in the Android UI framework, all touch events belong to the View where the touch originated. So if someone touched a Button, all touch events are processed through that Button until that person’s finger is raised up; this includes other touch pointers for multi-touch. While multi-touch can be achieved, it is very unnecessary and would involve having to create a custom view which can be quite complicated. To remedy this problem, a third version of the GUI was designed; one where multi-touch is eliminated entirely.

![Figure 15: Third Design of GUI](image)

Here, the directional buttons are used for steering, the Brake button is to stop the lawn mower, and the 1st and Max buttons set the duty cycle to 50 and 100 percent duty cycle, respectively.

In order to fulfill the request of more speeds for the lawn mower, a fourth and final GUI was designed as show below.
The N button, when held down, puts the lawn mower in a neutral gear and the directional arrows are used for steering. The buttons labeled 1 through 6 vary the speed of the lawn mower to 50, 60, 70, 80, 90, 100 percent duty cycle respectively.

In order for the Android application and Arduino to communicate with each other, a Server-Client model was chosen. Previously, the application acted as the client and the Arduino acted as the server. The Arduino gained WiFi capabilities via the WiFly RN-XV module which created an ad-hoc network for devices to connect to. An ad-hoc network is one that allows wireless devices to communicate with each other in a peer-to-peer manner. However, Android devices do not allow ad-hoc networks to be discoverable and does not allow one to connect to them. In order to remedy this problem, the Android device was chosen to act as the server and the Arduino act as the client.

To do so, the Android device had to create a portable WiFi hotspot. After it was created, the WiFly module could then connect to it. As a result, the Android device can now send commands to the WiFly module and control the lawn mower via UDP packets. The packets are sent using the Java DatagramSocket class’ send(packet) function. The packets were repeatedly sent to the Arduino every 5ms to prevent its software failsafe from activating. This was done by using Java’s TimerTask class’ scheduleAtFixedRate() function.
**FRAME:**

“Don’t put your hand in there!” ~Andrew Willson

In order to put everything together, it had to be mounted onto the frame. The frame design is based off a dolly that is used to move heavy furniture and appliances. This was chosen due to the weight of the lawn mower and batteries being heavy.

![Figure 17: Dolly Frame](image17.png)

Adjustments had to be made to the design of the frame to compensate for moving around in the lawn. Instead of having flat caster wheels in the front, treaded caster wheels replaced the flat wheels. Once in place the lawn mower was mounted onto the frame with L-brackets where each of the wheels used to be.

![Figure 18: Lawn Mower](image18.png)
Once the lawn mower was in place the back wheels had to be designed to work in the differential steering fashion. The back was fitted with two 10 inch wheels connected together by a rear axle. This axle was connected to the frame through more L-brackets and mending plates.

![Figure 19: The Lawn Mower Being Mounted onto the Frame](image)

Once this was done a battery mount was made in the back of the frame in order to hold both batteries in place.

![Figure 20: Screwing in the Battery Mount](image)
Lastly to put all the electronics on the frame, they were placed on piece of plywood on the back of the frame and inside a metal box to keep everything in place. Since everything is in the back the electronics can be easily connected to the power distribution strip which is connected to the batteries.

**Connections:**
The power source used for the project was a car battery. The car battery was connected to a power distribution block, which supplied the power needed to the different components of the project. Two switches were connected to the power distribution block, one for the H-bridge and its fan, and the other for the Arduino.
While the H-bridge was directly connected to a switch, the Arduino wasn’t. A step-down regulator was needed first so the Arduino could function properly. After the Arduino was powered properly the WiFly could be powered. The WiFly was powered by having pin 1 of the WiFly connected to the 3.3v pin of the Arduino and pin 10 of the WiFly to the GND pin of the Arduino.

These connections were made using jumper wires. Figure 23 shows how the WiFly Shield was powered by the Arduino.

**Figure 23: Arduino connected to WiFly**
Figure 24 shows how the Arduino was then connected to the two H-Bridges.
<table>
<thead>
<tr>
<th>Qty</th>
<th>Part Name</th>
<th>Unit Cost (USD)</th>
<th>Total(USD)</th>
<th>Total with shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RN-XV WiFly Module</td>
<td>34.95</td>
<td>34.95</td>
<td>34.95</td>
</tr>
<tr>
<td>2</td>
<td>24V 280W Motor</td>
<td>29</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>8</td>
<td>1 ft #25 Chain</td>
<td>2.5</td>
<td>20</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>master link</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>80mm fan</td>
<td>2.49</td>
<td>4.98</td>
<td>5.32</td>
</tr>
<tr>
<td>2</td>
<td>10” Pneumatic Tire</td>
<td>4.99</td>
<td>9.98</td>
<td>47.03</td>
</tr>
<tr>
<td>2</td>
<td>8” Pneumatic Swivel Caster Wheel</td>
<td>13.49</td>
<td>26.98</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3ft 5/8” Steel Rod</td>
<td>6.37</td>
<td>6.37</td>
<td>6.37</td>
</tr>
<tr>
<td>2</td>
<td>2x6x8 Lumber</td>
<td>5.98</td>
<td>11.96</td>
<td>18.97</td>
</tr>
<tr>
<td>1</td>
<td>1x6x8 Lumber</td>
<td>5.77</td>
<td>5.77</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>IRFZ40 N-Channel MOSFET</td>
<td>1.1</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>IRF9Z30 P-Channel MOSFET</td>
<td>1.86</td>
<td>18.6</td>
<td>22.66</td>
</tr>
<tr>
<td>2</td>
<td>6.5in #25 Sprockets</td>
<td>10.45</td>
<td>20.9</td>
<td>26.59</td>
</tr>
<tr>
<td>1</td>
<td>Mini Wireless Color Camera</td>
<td>28.98</td>
<td>28.98</td>
<td>28.98</td>
</tr>
<tr>
<td>1</td>
<td>xbee shield</td>
<td>11.55</td>
<td>11.55</td>
<td>21.26</td>
</tr>
<tr>
<td>2</td>
<td>5/16 bolts</td>
<td>1.37</td>
<td>2.74</td>
<td>2.74</td>
</tr>
<tr>
<td>1</td>
<td>3/8 zinc flat</td>
<td>3.24</td>
<td>3.24</td>
<td>3.24</td>
</tr>
<tr>
<td>4</td>
<td>5/8 hex nut</td>
<td>0.35</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>4</td>
<td>5/8 flat washer</td>
<td>0.33</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>3</td>
<td>3/8 hex bolt</td>
<td>1.21</td>
<td>3.63</td>
<td>3.63</td>
</tr>
<tr>
<td>1</td>
<td>5/16 zinc loc</td>
<td>3.99</td>
<td>3.99</td>
<td>3.99</td>
</tr>
<tr>
<td>1</td>
<td>3 pack 9volt battery</td>
<td>6.47</td>
<td>6.47</td>
<td>6.47</td>
</tr>
<tr>
<td>1</td>
<td>1x2 gpltslt8</td>
<td>3.26</td>
<td>3.26</td>
<td>3.26</td>
</tr>
<tr>
<td>12</td>
<td>3/8 hex nut</td>
<td>0.12</td>
<td>1.44</td>
<td>1.44</td>
</tr>
<tr>
<td>12</td>
<td>5/16 hex nut</td>
<td>0.11</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>1</td>
<td>spark plug</td>
<td>2.09</td>
<td>2.16</td>
<td>2.16</td>
</tr>
<tr>
<td>1</td>
<td>corner brace</td>
<td>2.83</td>
<td>3.03</td>
<td>3.03</td>
</tr>
<tr>
<td>3</td>
<td>blade fuse holder</td>
<td>1.97</td>
<td>5.91</td>
<td>6.32</td>
</tr>
<tr>
<td>1</td>
<td>bike chain delinker</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>Paint Remover</td>
<td>1</td>
<td>6.72</td>
<td>6.72</td>
</tr>
<tr>
<td>1</td>
<td>5/16 bolts</td>
<td>2.02</td>
<td>2.02</td>
<td>2.02</td>
</tr>
<tr>
<td>2</td>
<td>5/16in nut</td>
<td>0.16</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td>6</td>
<td>3.8 in hex nut</td>
<td>0.12</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>6</td>
<td>3/8 in flat washer</td>
<td>0.14</td>
<td>0.84</td>
<td>0.84</td>
</tr>
<tr>
<td>2</td>
<td>6 in corner brace (L-Brakcet)</td>
<td>3.77</td>
<td>7.54</td>
<td>7.54</td>
</tr>
<tr>
<td>3</td>
<td>3/8 hex bolt</td>
<td>1.46</td>
<td>4.38</td>
<td>4.38</td>
</tr>
<tr>
<td>1</td>
<td>3/8 hex bolt</td>
<td>2.52</td>
<td>2.52</td>
<td>2.52</td>
</tr>
<tr>
<td>1</td>
<td>3/8 hex bolt</td>
<td>1.84</td>
<td>1.84</td>
<td>1.84</td>
</tr>
<tr>
<td>1</td>
<td>1-1/2x1-1/2 slotted edge piece</td>
<td>3.26</td>
<td>3.26</td>
<td>3.26</td>
</tr>
<tr>
<td>12</td>
<td>12x1 6 count 1 inch screws</td>
<td>1.18</td>
<td>1.18</td>
<td>1.18</td>
</tr>
<tr>
<td>12</td>
<td>12x3/4 inch screws</td>
<td>1.18</td>
<td>1.18</td>
<td>1.18</td>
</tr>
<tr>
<td>12</td>
<td>12x1-1/4 inch screws</td>
<td>1.18</td>
<td>1.18</td>
<td>1.18</td>
</tr>
<tr>
<td>1</td>
<td>6x1inch screws</td>
<td>1.18</td>
<td>1.18</td>
<td>1.18</td>
</tr>
<tr>
<td>3</td>
<td>8-32 3inch screws</td>
<td>1.18</td>
<td>3.54</td>
<td>3.54</td>
</tr>
<tr>
<td>1</td>
<td>8/32 22 count nuts</td>
<td>1.18</td>
<td>1.18</td>
<td>1.18</td>
</tr>
<tr>
<td>1</td>
<td>1.5x1.625 inch corner flange</td>
<td>0.58</td>
<td>0.58</td>
<td>0.58</td>
</tr>
<tr>
<td>2</td>
<td>Toggle Switch</td>
<td>3.99</td>
<td>7.98</td>
<td>7.98</td>
</tr>
<tr>
<td>4</td>
<td>5/8 hex nut</td>
<td>0.35</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>1</td>
<td>16x8 duct cap</td>
<td>6.24</td>
<td>6.24</td>
<td>6.24</td>
</tr>
<tr>
<td>1</td>
<td>16x8 duct takeoff</td>
<td>8.84</td>
<td>8.84</td>
<td>8.84</td>
</tr>
<tr>
<td>3</td>
<td>1/4 in washer</td>
<td>0.12</td>
<td>0.36</td>
<td>0.36</td>
</tr>
<tr>
<td>1</td>
<td>18 gauge wire</td>
<td>7.99</td>
<td>7.99</td>
<td>7.99</td>
</tr>
<tr>
<td>2</td>
<td>desolder braid</td>
<td>4.49</td>
<td>8.89</td>
<td>8.89</td>
</tr>
<tr>
<td>1</td>
<td>Desolder Bulb</td>
<td>5.99</td>
<td>5.99</td>
<td>5.99</td>
</tr>
<tr>
<td>1</td>
<td>all new pmos and nmos</td>
<td>45.52</td>
<td>45.52</td>
<td>74.96</td>
</tr>
<tr>
<td>1</td>
<td>Rn xy Wifi shield</td>
<td>34.95</td>
<td>34.95</td>
<td>64.47</td>
</tr>
<tr>
<td>1</td>
<td>Roller Chain Breaker</td>
<td>12.53</td>
<td>12.53</td>
<td>12.53</td>
</tr>
<tr>
<td>1</td>
<td>Wireless Camera</td>
<td>28.98</td>
<td>28.98</td>
<td>28.98</td>
</tr>
<tr>
<td>1</td>
<td>Arduino Uno</td>
<td>24.95</td>
<td>24.95</td>
<td>24.95</td>
</tr>
<tr>
<td>2</td>
<td>14x1 1/4in screws</td>
<td>1.18</td>
<td>2.36</td>
<td>2.36</td>
</tr>
<tr>
<td>4</td>
<td>L-Brackets</td>
<td>3.77</td>
<td>15.08</td>
<td>15.08</td>
</tr>
<tr>
<td>6</td>
<td>5/16 in hex nuts</td>
<td>0.11</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>6</td>
<td>5/16 3/4inch bolts</td>
<td>0.16</td>
<td>0.96</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>total</td>
<td></td>
<td>697.27</td>
<td></td>
</tr>
</tbody>
</table>
SUSTAINABILITY

The above chart gives an overview of what the total costs of this project were. A significant portion of the cost was from parts that had to be ordered last minute due to part malfunctions that occurred on the last week. If this sort of device were to be mass produced, there would be a few factors that may contribute towards environmental damage. This project is constructed from wood, which may contribute to deforestation if it were to be mass produced. In addition to this, the lawnmower uses gasoline, which can be expensive and can cause environmental damage if spilled. To help balance this out, design considerations may include either making this design solar powered, or using an electric lawnmower.

While the total cost of the lawn mower is $697.27, the costs could be reduced. The results of losing the WiFi module and burning transistors significantly set back the costs of producing this project by $139.43. Removing the wireless camera reduces the project price by $28.98. Not using an XBee Shield and simply connecting the WiFi module directly to the Arduino reduces costs by $21.26. Due to inexperience with the mechanical construction of the frame and having to buy parts in bulk (such as purchasing packs of fuses when only two are required), there was an excess amount of parts bought for the frame. This can be estimated to be $60. Removing the cost of the 9-volt battery for the camera is $6.47. If one cuts one’s own lumber, this reduces the cost by $18.97. Trading range for price, one could switch the WiFi module with a Bluetooth module and reduce the cost by $25, with the cost of a Bluetooth module being $10. The total reduction in price is $139.43 + $29.98 + $21.26 + $60 + $6.47 + $18.97 + $25 = $301.11. The price per production of an Android Controlled Lawnmower is thus $697.97 - $301.11 = $396.86. Considering that the average price of a new lawn mower is $300, one could assemble an Android Controlled Lawnmower using old lawnmower parts for only $100 more. The Android Controlled Lawnmower is also more functional and easier to use than a brand new push mower.
<table>
<thead>
<tr>
<th>MEMBER CONTRIBUTION</th>
<th>Parth</th>
<th>Andrew</th>
<th>Andre</th>
<th>Martin</th>
<th>Kevin</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>H-bridge</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Designed circuit</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Bread board assembly and test</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Tested motors on batteries</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Soldered H-bridge onto PCB board and test with 400mA power supplies</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Test H-Bridge with Motors and 5A power supplies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>6. Tested Arduino code/frequencies to test if PWM pins needed the same timers Looked at switching rate vs. heat generated by the H-Bridges</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>7. Tested with car battery</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>8. Soldered Capacitor and pull down resistor</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Soldered fuses</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>1. Tested with android, arduino, and battery</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>

| **Android**                         |       |        |       |        |       |
| 1. Designed first GUI               |       |        |       | ✔️     |       |
| 2. Designed second GUI iteration    | ✔️    | ✔️     |       |        |       |
| 3. Wrote Android TCP client code that communicates with mock server |       | ✔️     |       |        |       |
| 4. Designed third GUI iteration that eliminated multitouch |       |        |       | ✔️     |       |
| 5. Revised Android app to use UDP protocol instead of TCP |       |        | ✔️     | ✔️     |        |
| 6. Designed fourth and last GUI iteration that included more speeds |       | ✔️     | ✔️     | ✔️     | ✔️    |
| 7. Implemented a timer that sends UDP packets periodically to Arduino |       |       | ✔️     | ✔️     | ✔️    |
| 8. Fixed Android app code that eliminated new threads being created everytime the user sends a different packet |       | ✔️     | ✔️     | ✔️     | ✔️    |
| 9. Tested the distance the Android device and Arduino can communicate with each other |       | ✔️     | ✔️     | ✔️     | ✔️    |
### Arduino&Wifly

<table>
<thead>
<tr>
<th>Task</th>
<th>Parth</th>
<th>Andrew</th>
<th>Andre</th>
<th>Martin</th>
<th>Kevin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Tested WiFly with various libraries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Learned how communication works between WiFly and Arduino</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Tested WiFly under WiFi and created a sample UDP Server/Client. Figured out it receives on port 2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Tested WiFly under Ad-Hoc mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Wrote preliminary PWM serving code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Implemented a software failsafe so that the motors would stop since communication is via UDP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Implemented auto-reconnect code so that the WiFly would keep looking for the tablet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Implemented flexible speed code for the PWMs and cleaned up the code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Added workarounds for the auto-reconnect code since the hardware was buggy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Cleaned up code and matched pins to corresponding timer frequencies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frame

<table>
<thead>
<tr>
<th>Task</th>
<th>Parth</th>
<th>Andrew</th>
<th>Andre</th>
<th>Martin</th>
<th>Kevin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Measured lawn mower dimensions for frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Build test frame with 2x4 lumber</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Fixed lawn mower</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Cut 2x6 and 1x6 for frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Assemble caster wheels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Mounted lawn mower</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Assembled structure for back wheels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Created battery mount</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Mounted the motor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Mount electronics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Test software on frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Mount camera</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Attach chains</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CONCLUSIONS AND FUTURE WORKS:
The objective in this project was to design an Android controlled lawnmower. The design used a simple interface between an Android platform, a WiFly shield, an Arduino Uno, and an H-Bridge. In this interface, the Android device connected to the WiFly shield via an ad-hoc network. This allowed input signals from the Android device to be read by the Arduino which could be translated into speed and directional control through differential steering.

There were a number of improvements that could have been made in order to improve the design. In terms of the electronics portion, the biggest improvement that could be made would be to have the H-Bridge circuit etched onto a Printed Circuit Board (PCB). Having the circuit etched would be much more expensive, but it would be a much more reliable design. Having to solder the H-Bridge by hand was a very tedious process. One needed to be very meticulous in ensuring that the components were connected properly. This often led to improper connections and improper grounding.

After consulting a Mechanical Engineer about the structure of our project, various improvements that can be made on the body of the lawnmower were found. These improvements range from force distribution to material choices. The material choices made throughout this project, for the body, were based on available resources and cost efficiency. However, in an ideal situation, the supporting frame of lawnmower would be made out of a recycled steel to control cost and maximize support. The steel would provide a greater ultimate strength and a better lifespan. In terms of the force distribution, the structure would be arranged differently to provide maximum support (with any material). This design lawnmower was based on a dolly frame and then given support beams for the electrical components. The wheels would have more support if the wooden planks were put in vertical on the sides with the axels through them. These vertical planks would then be closed up on the front and back with two more vertical planks joining them. This would localize the force per area distribution and essentially put less force on the wheels. With the body structured this way, the lawnmower would then be supports with L-brackets along the vertical portion of the beams on either side. This would provide greater stability to the lawnmower when moving around.

A software side improvement to this project would be to develop a method of streaming the video from the wireless camera onto the Android application. This would work by having the camera stream the video onto a website containing a live stream. The application could then link to the stream, allowing the video to appear on the Android application. This would allow for the video to be integrated into the application, eliminating the need for a separate monitor to view the camera feed.

This projected proved to be a challenging yet rewarding experience. Through this project, it was proven that it is possible to use an Android device to automate complicated tasks that can normally be physically demanding, or otherwise impossible. While this design would need refining before being mass produced, it is possible that a similar design to make mowing one’s lawn a much more relaxing task.
APPENDIX:

**Spice Models:**

```plaintext
.SUBCKT 2N7000 3 4 5
* Nodes D G S
M1 3 2 5 5 MOD1
RG 4 2 343
RL 3 5 6E6
D1 5 3 DIODE1
.MODEL MOD1 NMOS VTO=2.474 RS=1.68 RD=0.0 IS1E-15 KP=0.296
+CGSO=23.5F CGDO=4.5F CBD=53.5F PE=1 LAMBDA=267E-6
.MODEL DIODE1 D IS=1.254E-13 N=1.0207 RS=0.222
.ENDS 2N7000

.model IRFZ40
NMOS(Level=3 Gamma=0 Delta=0 Eta=0 Theta=0 Kappa=0.2 Vmax=0 Xj=0
+ Tx=100n Uc=600 Phi=.6 Rs=15.03m Kp=20.84u W=1.8 L=2u Vto=3.408
+ Rd=2.122m Rds=222.2K Cbd=2.805n Pb=.8 Mj=.5 Fc=.5 Cgso=1.836n
+ Cgdo=294.3p Rg=1.028 Is=28.65p N=1 Tt=215n)

.model IRF9230
PMOS(Level=3 Gamma=0 Delta=0 Eta=0 Theta=0 Kappa=0.2 Vmax=0 Xj=0
+ Tx=100n Uc=300 Phi=.6 Rs=9.16m Kp=10.36u W=.79 L=2u
+ Vto=3.587 Rd=1.997m Rds=222.2K Cbd=2.385n Pb=.8 Mj=.5 Fc=.5
+ Cgso=714p Cgdo=722.9p Rg=20.9l Is=77.38f N=2 Tt=102n)
```
public class MainActivity extends AppCompatActivity implements OnTouchListener, android.view.View.OnClickListener {

    ImageButton upleft, up, upright, downleft, down, downright, neutral;
    ImageButton btnGear1, btnGear2, btnGear3, btnGear4, btnGear5, btnGear6, btnBrake;
    TextView txtSpeed, txtGear;

    //Client Code Variables
    String haddr = "192.168.43.88";
    String packet;
    String direction;
    String speed;
    byte[] data;
    DatagramSocket clientSocket;
    DatagramPacket sendPacket;
    InetAddress IPAddress;
    int port = 2000;
    Timer repeatSend = new Timer();
    String initDirection;
    clientSendInfo client;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        setContentView(R.layout.main);

        txtSpeed = (TextView) findViewById(R.id.txtSpeed);
        txtGear = (TextView) findViewById(R.id.txtGear);
        loadButtons();
    }
}
data = new byte[4];
direction = "d3";
speed = "$0";
//Start on Neutral with Zero speed
client = new clientSendInfo();
client.execute();

@Override
protected void onDestroy() {
    super.onDestroy();
    if(repeatSend!=null)
    {
        repeatSend.cancel();
    }
}

public void loadButtons()
{
    upleft=(ImageButton) findViewById(R.id.btnUpLeft);
    up=(ImageButton) findViewById(R.id.btnUp);
    upright=(ImageButton) findViewById(R.id.btnUpRight);
    downleft=(ImageButton) findViewById(R.id.btnDownLeft);
    down=(ImageButton) findViewById(R.id.btnDown);
    downright=(ImageButton) findViewById(R.id.btnDownRight);
    neutral=(ImageButton) findViewById(R.id.btnNeutral);
    upleft.setOnTouchListener(this);
    up.setOnTouchListener(this);
    upright.setOnTouchListener(this);
    downleft.setOnTouchListener(this);
    down.setOnTouchListener(this);
    downright.setOnTouchListener(this);
    neutral.setOnTouchListener(this);
    btnGear1=(ImageButton) findViewById(R.id.btnGear1);
    btnGear2=(ImageButton) findViewById(R.id.btnGear2);
    btnGear3=(ImageButton) findViewById(R.id.btnGear3);
    btnGear4=(ImageButton) findViewById(R.id.btnGear4);
    btnGear5=(ImageButton) findViewById(R.id.btnGear5);
    btnGear6=(ImageButton) findViewById(R.id.btnGear6);
    btnBrake=(ImageButton) findViewById(R.id.btnBrake);
    btnGear1.setOnClickListener(this);
    btnGear2.setOnClickListener(this);
    btnGear3.setOnClickListener(this);
    btnGear4.setOnClickListener(this);
    btnGear5.setOnClickListener(this);
    btnGear6.setOnClickListener(this);
    btnBrake.setOnClickListener(this);
}
@Override
public boolean onTouch(View v, MotionEvent event) {
    switch(v.getId())
    {
        //Turn Left button press
        case R.id.btnUpLeft:
            
            if(event.getAction()==android.view.MotionEvent.ACTION_DOWN)
            {
                direction="d5";
            }

            if(event.getAction()==android.view.MotionEvent.ACTION_UP)
            {
                direction="d3";
            }
            break;
        //Up Button Press
        case R.id.btnUp:
            
            if(event.getAction()==android.view.MotionEvent.ACTION_DOWN)
            {
                direction="d4";
            }

            if(event.getAction()==android.view.MotionEvent.ACTION_UP)
            {
                direction="d3";
            }
            break;
        //Turn Right Press
        case R.id.btnUpRight:
            
            if(event.getAction()==android.view.MotionEvent.ACTION_DOWN)
            {
                direction="d6";
            }

            if(event.getAction()==android.view.MotionEvent.ACTION_UP)
            {
                direction="d3";
            }
            break;
        //Down Left Button
        case R.id.btnDownLeft:
            {
            
            }}
if(event.getAction()==android.view.MotionEvent.ACTION_DOWN)
    {
        direction="d1";
    }

if(event.getAction()==android.view.MotionEvent.ACTION_UP)
    {
        direction="d3";
        break;
    }

//Down Button
    case R.id.btnDown:
    {
        if(event.getAction()==android.view.MotionEvent.ACTION_DOWN)
            {
                direction="d0";
            }
        if(event.getAction()==android.view.MotionEvent.ACTION_UP)
            {
                direction="d3";
                break;
            }
    }

//Down Right Button
    case R.id.btnDownRight:
    {
        if(event.getAction()==android.view.MotionEvent.ACTION_DOWN)
            {
                direction="d2";
            }
        if(event.getAction()==android.view.MotionEvent.ACTION_UP)
            {
                direction="d3";
                break;
            }
    }

    //neutral button
    case R.id.btnNeutral:
    {
        if(event.getAction()==android.view.MotionEvent.ACTION_DOWN)
            {
                direction="d3";
                break;
            }
return true;
}

@Override
public void onClick(View v) {
    switch(v.getId())
    {
    case R.id.btnBrake:
    {
        speed="$0";
        txtSpeed.setText("0% Speed");
        break;
    }
    case R.id.btnGear1:
    {
        speed="$1";
        txtSpeed.setText("50% Speed");
        break;
    }
    case R.id.btnGear2:
    {
        txtSpeed.setText("60% Speed");
        speed="$2";
        break;
    }
    case R.id.btnGear3:
    {
        txtSpeed.setText("70% Speed");
        speed="$3";
        break;
    }
    case R.id.btnGear4:
    {
        txtSpeed.setText("80% Speed");
        speed="$4";
        break;
    }
    case R.id.btnGear5:
    {
        txtSpeed.setText("90% Speed");
        speed="$5";
        break;
    }
    case R.id.btnGear6:
    {
        txtSpeed.setText("100% Speed");
        speed="$6";
        break;
    }
    }
//Extending AsyncTask allows you to perform network operations in a
//background thread
//so you don’t freeze the UI Thread
private class clientSendInfo extends AsyncTask<byte[], Void, Void> {

    public void sendCommand() {
        try {
            data = (direction+speed).getBytes();
            clientSocket=new DatagramSocket();
            IPaddress=InetAddress.getByName(haddr);
            sendPacket=new DatagramPacket(data, data.length, IPaddress, port);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected Void doInBackground(byte[]... params) {
        repeatSend.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                try {
                    sendCommand();
                    clientSocket.send(sendPacket);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }, 0, 500); //Repeatedly send command every 5 ms

        return null;
    }
}

}
ARDUINO CODE:

/**********************************************************
* This code uses the following library from:
* Title: WiFly RN-XV Arduino Library
* Author: Darran Hunt
* Date: 2013
* Code Version: 0.3
* Availability: https://github.com/harlequin-tech/WiFlyHQ
*****************************************************/

#include "WiFlyHQ.h"

#define reverse 0
#define Rleft 1
#define Rright 2
#define neutral 3
#define forward 4
#define Fleft 5
#define Fright 6
#define leftMotorLeftPIN 9
#define leftMotorRightPIN 3//10
#define rightMotorLeftPIN 10//6//11
#define rightMotorRightPIN 11//3//13//3//13

//PWM: 3, 5, 6, 9, 10, 11
//9 and 10 are on the same timer
//3 and 11 are on the same timer
int speed;
// 0 = stop/brake, 1 = 50%, 2 = 60%
// 3 = 70%, 4 = 80%, 5 = 90%, 6 = 100%

int direction;
// 0 = reverse, 1 = left-reverse, 2 = right-reverse, 3 = neutral, 4 = forward, 5 = forward-left, 6 = forward-right

WiFly wifly;
char buf[32];

// keeps track of polling for failsafe
int count;

// keeps track of how many cycles we hit the failsafe
int failcount;

char *wssid = "KevinSHARTY2"; /* SSID of Access Point */
char *wpass = ""; /* password of Access Point */
int usesPassword = 0;
// 0 = no password, 1 = WEP, 2 = WPA1/2

#define sserialtype sSerial
// type of serial, change to serial1 for Leonardo
#include <SoftwareSerial.h>
SoftwareSerial sSerial(4,7);
void setup() {
    pinMode(leftMotorLeftPIN,OUTPUT);
pinMode(leftMotorRightPIN, OUTPUT);
pinMode(rightMotorLeftPIN, OUTPUT);
pinMode(rightMotorRightPIN, OUTPUT);
pinMode(13, OUTPUT);
digitalWrite(13, LOW);
speed = 1; direction = 3; // Start in neutral
changeDirection(speed, direction);
resetCount();
serialtype.begin(9600);

//while (!sSerial);
//while (!Serial1 || !Serial);
wifly.begin(&serialtype, NULL);
//wifly.factoryRestore();
//while (!wifly.begin(&sSerial, NULL))
//delay(200);

// begins the Wifly using SoftwareSerial on pins 4 & 7
// Hardware pins caused problems during startup for the UNO
//wifly.reboot();
if (!wifly.isAssociated())
    setupConnection();
/*else
    Serial.println(F("Already connected"));
    Serial.print(F("MAC: "));
    Serial.println(wifly.getMAC(buf, sizeof(buf)));
    Serial.print(F("DeviceID: "));
    Serial.println(wifly.getDeviceID(buf, sizeof(buf)));
    Serial.println(F("WiFly ready"));*/

wifly.setIpProtocol(WIFLY_PROTOCOL_UDP);
wifly.setDeviceID("Wifly-UDPsuckit");
speed = 1; direction = 3; // Start in neutral
changeDirection(speed, direction);
delay(4000);

int c;
int offset;

void loop2() {}

void loop() {
    if (count == 600) { // FAILSAFE
        resetMotors();
        count = 0;
        failcount++;
        //Serial.println("FAILSAFE");
    }
    count++;
    if (wifly.isAssociated() && failcount < 30) { // check if it hasn't failed 30 times
        while (wifly.available() > 0) {
            while (wifly.available() > 0) {
                c = wifly.read();
                if (c == 'd' || c == '$')
                    goto next;
            }
            next:
            if (wifly.available() > 0) {
                
            }
        }
    }
}
if (c == '$') { // if there is a char with $ for speed
c = wifly.read(); // it reads the character after it
offset = (int)(c - '0'); // pull out the offset
digitalWrite(13, HIGH);
if (offset >= 0 && offset <= 6 && offset != speed) {
speed = offset;
changeDirection(speed, direction);
resetCount();
} else if (offset == speed) {
resetCount();
//Serial.println("Same speed");
}
} else if (c == 'd') {
c = wifly.read();
offset = (int)(c - '0'); // pull out the offset
digitalWrite(13, HIGH);
if (offset >= 0 && offset <= 6 && offset != direction) {
direction = offset;
changeDirection(speed, direction);
resetCount();
} else if (offset == direction) {
resetCount();
//Serial.println("Same direction");
}
} else {
if (wifly.isAssociated() && failcount < 60) { // after failing 30 cycles, increase wait time
delay(2000);
failcount++;
setupConnection();
} else {
if (!wifly.isAssociated()) {
delay(2000);
setupConnection();
} else {
delay(10000); // Delay for 10 seconds before trying again
setupConnection();
}
}
delay(3);
digitalWrite(13, LOW);
}

void setupConnection() {
//Serial.println(F("Not connected to a network..."));
wifly.reboot();
delay(2000);
count = 0;
wifly.setSSID(wssid);

switch (usesPassword) {
case 2:
    wifly.setPassphrase(wpass);
    break; /* For WPA */
case 1:
    wifly.setKey(wpass);
    break; /* For WEP, CHOOSE CORRECTLY ONE OF THESE TWO */
}
wifly.enableDHCP();
wifly.join();
if (wifly.isAssociated()) // Reset to start processing input again
    failcount = 0;
}

void setupAdhoc() {
    wifly.setSSID(wssid);
    wifly.createAdhocNetwork(wssid, 4);
}

void resetCount() {
    count = 0;
    failcount = 0;
}

void resetMotors() {
    speed = 0; // stopped
    direction = 3; // neutral
    changeDirection(speed, direction);
}

void writePINS(int leftMLP, int leftMRP, int rightMLP, int rightMRP) {
    analogWrite(leftMotorLeftPIN, leftMLP);
    analogWrite(leftMotorRightPIN, leftMRP);
    analogWrite(rightMotorLeftPIN, rightMLP);
    analogWrite(rightMotorRightPIN, rightMRP);
}

void changeDirection(int speed, int direction) {
    // printstuff(speed,direction);
    if (speed == 0) { // Stop
        writePINS(255, 255, 255, 255);
    } else {
        // Gives the speeds from 50-100% with 10% granularity
        int scale = 178 + (int)((speed-1)*15.4); // 128 + (int)((speed-1)*25.5);
        int isForward = (direction > neutral) ? 1 : 0;
        int isLeft = direction % 2;
        switch (direction) {
            case neutral: writePINS(0,0,0,0); break;
            case forward:
                case reverse: writePINS(scale*isForward, scale*(1-isForward), scale*isForward, scale*(1-isForward)); break;
case Rleft:
case Rright:
case Fleft:
case Fright:
    writePINS(scale*isForward*(1-isLeft),
                scale*(1-isForward)*(1-isLeft),
                scale*isForward*isLeft, scale*(1-
                isForward)*isLeft);
}
}
}

/* For debug use */
void connectInfo() {
    Serial.print(F("Connected"));
    Serial.print(F("MAC: "));
    Serial.println(wifly.getMAC(buf, sizeof(buf)));
    Serial.print(F("IP: "));
    Serial.println(wifly.getIP(buf, sizeof(buf)));
    Serial.print(F("Netmask: "));
    Serial.println(wifly.getNetmask(buf, sizeof(buf)));
    Serial.print(F("Gateway: "));
    Serial.println(wifly.getGateway(buf, sizeof(buf)));
}

void printstuff(int speed, int direction) {
    Serial.print("Speed: ");
    Serial.println(speed);
    Serial.print("Direction: ");
    Serial.println(direction);
    Serial.println("0 = reverse, 1 = left-reverse, 2 = right-reverse, 3 = neutral\n4 = forward, 5 = forward-left, 6 = forward-right");
} */
REFERENCES:


