

# SudoGlove: Gesture-Based Hardware Control

**Jeremy Blum**

Electrical and Computer Engineering  
jeb373@cornell.edu

**Edgar Alex Garcia**

Computer Science  
eag57@cornell.edu

**Joseph Ballerini**

Electrical and Computer Engineering  
jmb596@cornell.edu

**Tiffany Ng**

Computer Science  
tyn2@cornell.edu

## ABSTRACT

Humans and machines do not interface well. In an attempt to bridge the gap between humans and the systems they interact with, a plethora of input methods have been devised: keyboards, mice, joysticks, game controllers, and touch screens are just a few examples. Unfortunately, none of these devices remove the barrier between man and machine. With the SudoGlove control system, we aim to remove this obstruction by allowing the user to control a hardware device using natural gestures. The SudoGlove takes advantage of a multitude of sensors to capture hand movements and uses this information to control a device – in this case, a modified RC car. Testing showed that novice users were able to wear the glove and control the car with only a small amount of instruction. With some future improvements, it may be possible to remove the learning curve completely.

## INTRODUCTION

The most common thing that people do when frustrated with devices is perform hand gestures to try to show the device what they want it to do. A controller is unnatural and requires getting used to; body language and hand gestures, however, are instinctual. The SudoGlove utilizes various sensors to capture those hand gestures and interpret them as inputs. A touch of the thumb, a waggle of the finger, or a tilting of the hand all act as control inputs. By utilizing the gestures that most make out of simple habit, the hope is that users will forget the control glove is there at all. The device seems to do what the user wants, as though a direct line of communication exists between the user and the hardware to be controlled.

The old way of thinking: machines are there to do the sorts of work that users didn't want to do, like a dishwasher or a Roomba. The new way of thinking: machines are here to help us do the things that we want to do, but are not able to, like a construction yard crane. If users could reach a level of comfort with machines, as though using them was no more than using an extension of their own bodies, perhaps they could get to a point where they thought of themselves as the ones performing the action. That level of comfort and familiarity is not something that can be accomplished

overnight, it would take years, perhaps even generations to reach. Though the change may seem rather daunting, it has to start somewhere, and that's where the SudoGlove comes in, one of the first steps towards superior human-machine integration.

SudoGlove aims to bridge the gap between the user and traditional physical hardware devices. Given the high learning curve in understanding how to use foreign technologies, we hope to break away from conventional control mechanisms and explore an intuitive way to control these devices. SudoGlove will provide a tangible interface that relies on hand gestures to wirelessly control any device or software. By removing the distance between the user and traditional hardware devices, our goal is for SudoGlove to feel more like an extension of the body as opposed to an external machine.

The goal of this project is to capture simple hand gestures from the SudoGlove and use that input to wirelessly control a modified RC car. Controlled variables include speed, steering, lights and sounds using a combination of flex, force, vibration, and gyroscopic sensors. Multiple variables are controlled simultaneously as SudoGlove outputs a constant control signal.

The secondary goal of this project is to reach a level of comfort and precision with SudoGlove currently held by present day controllers. This is an important goal to achieve because if SudoGlove cannot perform at least as well as current controllers, it will be ignored as people will favor the more efficient option.

## RELATED WORK

Relatively recent developments in the field of glove-based input include the BarrettHand [1], a wired glove that can control hardware devices, which is a key feature in our design, though we improved upon it with wireless control to provide free range of motion. The Peregrine Gaming Glove [2] provides software calibration, but has no support for macros and no support for multiple simultaneous key presses, which our prototype handles without trouble. The Reusch



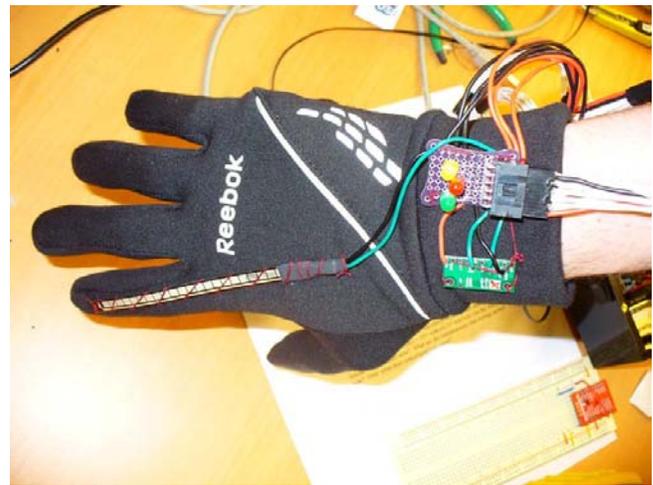
**Figure 1: The Original Powerglove Advertisement**

“Sonic Control” Glove [3] includes nice on-glove displays of volume and bass/treble levels, but is limited to controlling only the iPod, requires the user to press buttons on the glove, and does not sense gestures or pinches. Our prototype handles multiple different gestures and inputs and has an on-board LED array for displaying calibration information and visual feedback. We built a glove whose inputs are general enough to be used to control various devices and applications. Perhaps the model closest to our vision is the PowerGlove Remake [4,5] (Figure 1), which has wireless control, open Arduino architecture, and the ability to map input to multiple devices. Unfortunately, even this notable glove has its limitations and still requires deliberate button presses for its input, which is not the case for our glove. The most recent development is the G-Speak glove input system [6] that provides impressive gesture control, but also does not map to a hardware device and requires external sensors, limiting the range in which the user can roam and still provide input to the system. Our system can be used without external sensors and only requires the glove to be within the range of wireless communication with the hardware device. The G-Speak system seems to be highly intuitive though, an aspect that we strived to include and improve upon in our system by using gestures that do not require line of sight with the slave device but instead are simple and detached enough to be done behind your back. This control scheme is similar to that of the gyroscopic Loop Pointer [7] from Hillcrest Labs, which can be controlled from any hand position without pointing at the screen.

## DESIGN DESCRIPTION

### Components

Our prototype can be described in three parts: the glove, the controller, and the slave device (an RC car in this scenario).



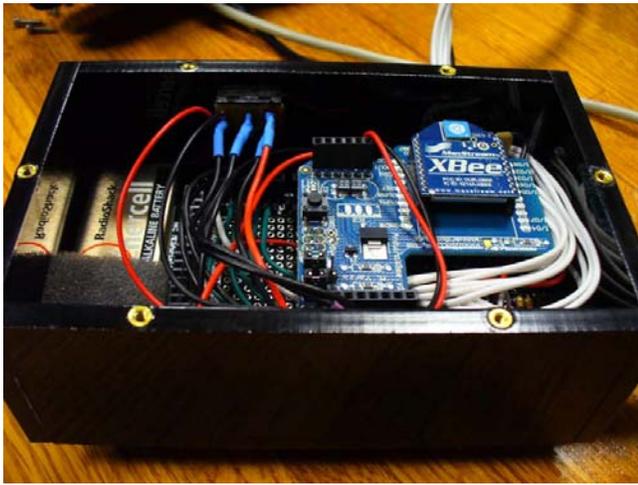
**Figure 2: SudoGlove Gyroscope, LEDs, and Flex Sensor**

### *The Glove*

The glove is fitted with a flex sensor, 2 force sensors, a vibration sensor, a 2D gyroscope, a small protoboard, 3 LEDs, and a connector for the 12-conductor wire that leads to the control unit. Each sensor is carefully sewn into the glove using double stitching to ensure durability. All contacts are all soldered and wrapped in heatshrink tubing to avoid potential shorts. In testing, it was found that the flex sensor was very fragile at its bend joint (near the solder contacts), so this part of the flex sensor was wrapped in heatshrink tubing to prevent it from flexing and undergoing undue stress. The force sensors and flex sensors are sewn directly to the tips of the fingers, but are otherwise only held in place by thread wrapping around their bodies. This permits the sensors to slide up and down on the fingers instead of getting bunched up when the user’s hand is clenched. The gyroscope is sewn to the wrist, adjacent to the protoboard which accepts connections from all the sensors and the cable from the control unit. Three LEDs on the protoboard are configured to inform the user of the position currently detected by the gyroscope (Figure 2).

### *The Control Unit*

The control unit contains an Arduino Mega, a prototyping shield, an XBee wireless unit, and a 9V battery. The components are housed in a laser-cut acrylic belt holster (Figure 3). The base of the container is glued together, and the top can be screwed on. Thermoplastic-molded nuts are inserted into the container base so the top can be screwed and unscrewed easily. An illuminated power switch is located on the side of the box, and will blink when battery power is running low. Located next to the switch is a connector for the glove. The control unit’s prototyping shield accepts connections to and from the glove and holds voltage divider circuits for the sensors. The appropriate resistor values were determined by testing sensor resolution with various values.



**Figure 3: Control Unit Belt Holster**

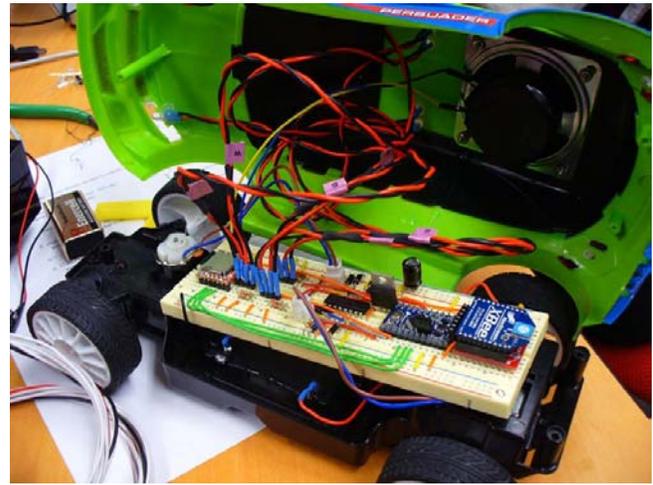
#### *The Modified RC Car*

The car is an RC car with its inner circuitry gutted and replaced with an Arduino mini, H-bridge controlled motors, SOMO audio board and speaker, blue/red/white LEDs, and an XBee module (Figure 4). The car's fake headlights were replaced with white LEDs, and 4 holes were drilled in the top of the car to house 4 blue and red LEDs to act as siren lights. A speaker is screwed to the rear roof of the car, and a radial pattern of holes was cut to allow sounds to be heard clearly. An H-Bridge allows the arduino to control the acceleration and turning motors which were already mounted in the chassis. The car's original 6V battery compartment (4x AA batteries) is wired into the control board. The arduino regulates this to 5V for use with the H-Bridge, and a separate 3.3V regulator is used to supply power to the XBee and SOMO audio module. All of the new circuitry is concealed by the car's original plastic casing.

#### **Features**

##### *Motion Capture*

The glove itself is nothing more than a Reebok brand neoprene glove bought at a sporting goods store. When fitted with all of its sensors, though, a sophisticated hand-gesture capture device is created. The flex sensor is sewn into the back of the index finger because the index finger is easily bent independent of other fingers, the force sensors are sewn to the tips of the ring and pinky fingers because they are easily pressed with the thumb, the vibration sensor is sewn into the palm to detect clapping, and the 2D gyroscope is sewn into the wrist because that is the part of the hand that rotates. In our first design iteration we attempted to use conductive thread to connect the sensors to the Arduino mega, but that approach ultimately failed as the thread frayed too easily. As a result, we switched to 22-gauge stranded wires and sewed the sensors in with standard thread. With the combined data of these sensors, a fairly detailed motion can be extrapolated. The performance of the motion capture is rated based on ease of



**Figure 4: Inside the RC Car**

use, the natural feeling of the necessary motions, and the recorded sensor readings. The sensor readings were mapped to cover the entire range of motion values to ensure the highest resolution possible. Testing showed that force sensor data was prone to noise; this was compensated for by taking a time-rectified average of these inputs and using that value. "Crosstalk" between sensors was rectified by calculating mathematical models for this noise and using them in the code to reject interference data. For example, the following code was used to eliminate crosstalk from the right force ring (RFR) sensor on the right force pinky (RFP) sensor:  $\text{if}(RFR > 0) \text{RFP} = \text{RFP} - RFR / 5$ . With all of these compensations in place the SudoGlove's motion capture is accurate and reliable.

##### *Wireless Communication*

The Arduino mega communicates wirelessly with the Arduino mini via two XBee modules on the same Personal Area Network (PAN). The performance of the wireless communication was evaluated based on how well the car would respond to wireless commands verses how it responded to commands sent directly over USB. Originally wireless communication was plagued by garbled commands and noise. This was corrected by modeling the wireless communication after the wireless communication used in the Etch-A-Sketch project. The Arduino mini sends out a specific char 'a' when it is ready to start receiving data from the Arduino mega; the Arduino mega reads this char and sends out a packet of numbers preceded by another specific char '.'. When the Arduino mini reads a period, it knows that the Arduino mega sent it a packet of numbers and reads it. After reading the packet and doing whatever the data told it to do, the Arduino mini sends out another char 'a'. Once the special characters were used, wireless communication preformed as well as the USB communication and was deemed successful.

Sensor	X Gyro	Y Gyro	Index Flex	Ring Force	Pinky Force	Vibration
Senses	Wrist Up/Down	Wrist Tilt	Finger Flexing	Ring Finger Pressure	Pinky Finger Pressure	Clapping
Filters	Averaged Acceleration Values	Integrated, Averaged, Filtered Spatial Position	Mapped Analog Values	Averaging	Cross-Talk Filter, Averaging	Averaging
Data Range	000 - 255	000, 127, 255 (Digital Turning)	000 - 255	000 - 255	000 - 255	000 - 255
Controls	N/A	Turning	Acceleration	Forward/Reverse	Lights, Sirens, Sounds	Car Horn

**Table 1: Control Scheme and Sensor Filtering**

### *Controlling a Hardware Device*

The glove component communicates with the RC car via a wireless XBee chip connected to the car's control board. The car interprets commands issued by the glove sent in the form of a period followed by 18 digits. When received by the Arduino, the period signals it is ready to receive a set of commands. The 18-digit command is broken down into six 3-digit chunks that each correspond to an aspect of the car to control. For instance, the second three digits represent an integer from 0-255 specifying what direction the car will steer in (i.e. direction of the front wheels): 000 = Left, 127 = Straight, 255 = Right. The complete 6 sets of commands sent to the car define: steering, speed, headlights, siren lights, siren sounds, and the horn (Table 1).

The program loaded onto the car's Arduino mini can then receive these numeric commands and map them onto the various components of the car's circuit, controlling the front and back motors, LEDs, and speakers. The car's circuitry is based on an Arduino mini connected to the H-bridge, which allows us to easily control the steering and direction of the wheels' motors all at once. We connected the H-bridge to the microcontroller according to its schematic, which includes a 10K-ohm resistor to the Arduino and an array of diodes between the connection to the two front and back motors. We used two white LEDs for the headlights, connected to the board with 220-ohm resistors, two pairs of LEDs (a pair consisting of one red and one blue) with 220-ohm resistors that will alternate in a blinking pattern for the siren lights, and a speaker connected to the SOMO chip on the breadboard that will switch between being used for both the siren sound and the horn.

The performance of the control was evaluated by how the car responded to given commands. Most errors in control were software based and required simple debugging of the code. The 18-digit command itself was verified by allowing a third XBee connected to a computer to listen in on the communication. When we first wired up the H-bridge, we connected the motor directly without any diodes. The absence of diodes led to half-hearted turns and choppy motion, both of which were rectified by implementing the complex diode array. After the debugging of the code and

adding the diode array, the car followed all commands given by SudoGlove correctly.

### **FUTURE WORK**

Given what we learned during the making of this project, we have a few ideas on how we would like to improve upon our design for future iterations. First off, a word on aesthetics: we would like to integrate the sensors into the fabric of the glove to achieve a cleaner look, sleeve the wire bus between the glove and control unit, and attach armbands of some sort to the wire bus so it can be worn more easily while providing strain relief. Second, a pair of gloves would make for more and better control options, different gestures, and advanced features. We learned that flex sensors are indispensable in order to take full advantage of the dexterity of a human hand. Future iterations might have many more flex sensors and less force sensors. Another lesson we learned was to take full advantage of the capabilities of your hardware. We were not using our wireless modules at full speed, which we believe was affecting the response time of the system. Higher quality sensors would also improve the sensitivity and response of the system. The ability to customize the interface to control different devices and applications was a key motivation for our project. In the future a software interface for selecting which gestures will control which capabilities on the controlled device or application will make it easy for the user to find a control scheme to fit their preferences. Of course, the control unit cannot simply forget these settings when it is powered off, so the ability to maintain configuration options even after the system is powered off is a must. This could be achieved by writing the configuration to the EEPROM on the Arduino Mega or to an SD card.

### **CONCLUSION**

The SudoGlove system was developed to demonstrate the possibility of intuitive, simple, glove-based input general enough to be extended to other applications, including hardware and software. Based on how well the system controlled the RC car, we are convinced the SudoGlove is a viable user interface with untapped capabilities. Though only a prototype, further development would undoubtedly produce an even more accurate and intuitive design. The SudoGlove has the potential to start bridging the large gap between human ideas and machine responses.

## REFERENCES

1. Barrett Hand Glove Control.  
<http://www.youtube.com/watch?v=Ywo6WR9NUPE>.
2. Peregrine Gaming Glove.  
<http://www.engadget.com/2009/12/18/peregrine-gaming-glove-modeled-calibrated-and-demoed-on-video/>.
3. Sonic Control Glove.  
<http://fibretronic.com/news/Reusch%20Sonic%20Control%20Glove>.
4. Nintendo Power Glove  
[http://en.wikipedia.org/wiki/Power\\_Glove](http://en.wikipedia.org/wiki/Power_Glove).
5. 20<sup>th</sup> Anniversary Power Glove Remake.  
<http://www.instructables.com/id/Power-Glove-20th-Anniversary-Edition/>.
6. Oblong Gesture Computing  
[http://www.readwriteweb.com/archives/minority\\_report\\_in\\_your\\_living\\_room\\_gestural\\_inter.php](http://www.readwriteweb.com/archives/minority_report_in_your_living_room_gestural_inter.php).
7. Hillcrest Labs Loop Pointer  
<http://hillcrestlabs.com/products/loop.php>

## APPENDICES

### Photo Gallery

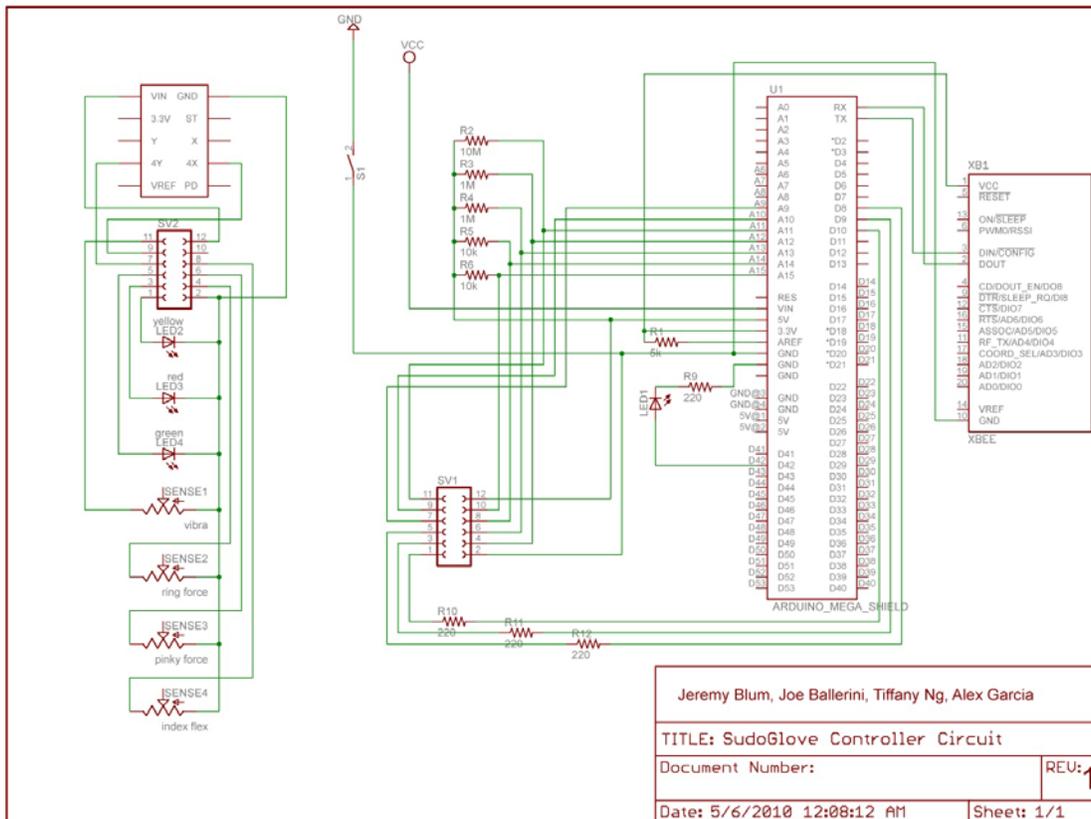
A photo gallery of the project is available at Jeremy Blum's Blog: <http://jeremyblum.com/2010/05/09/sudoglove/>

### Video Demonstration

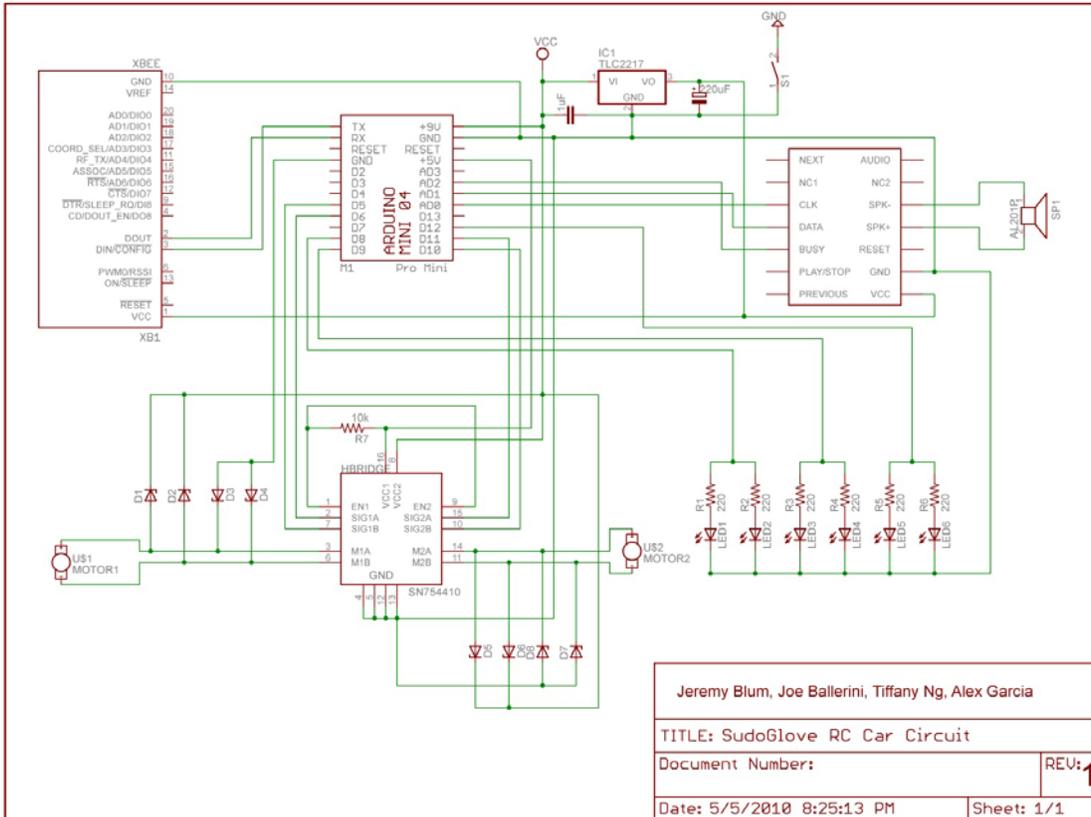
You can view a video demonstration of this project at the following YouTube link:

<http://www.youtube.com/watch?v=RnWPoaLU1i4>

### SudoGlove Controller Circuit



## RC Car Schematic



### SudoGlove Control Unit Arduino Code

```

/*****
**** SUDOGLOVE CONTROLLER PROGRAM
**** INFO 4320 Final Project, Spring 2010
**** Cornell University
**** Copyright: Jeremy Blum, Jow Ballerini, Alex Garcia, and Tiffany Ng
*****/

//Power Switch
const int pwr_led = 42;

//OPERATIONAL MODE
//When debug mode is set to true, pretty values will be printed to the terminal window (for use in USB mode)
//When debug mode is set to false, values are sent wirelessly to car in proper format (for use in XBEE mode)
const boolean debug = false; //False for normal operation
const boolean debug_map = true; //false to display raw sensor values (0-1024), true to display mapped sensor values (0-255) - only valid in debug mode.

//Turning Integration variables
int old_state_turn = 2; //1 is hand left, 2 is middle, 3 is hand right
int new_state_turn = 2;
int angle = 127; //Start in the middle (assumes calibration was successful)

//Gyroscope Bias
//Higher values means it it will take more umph to move to that position.
//Raise values if it tends to always go in that direction.
const int left_bias = 17;
const int right_bias = 7;

//Define the analog input pins for our sensors

```

```

const int right_gyro_4Y    = 9;
const int right_gyro_4X    = 10;
const int right_flex_index = 14;
const int right_flex_middle = 15; //NOTE: not in use
const int right_force_ring = 12;
const int right_force_pinky = 13;
const int right_vibra_palm = 11;
//Define the Analog Ouput Pins (PWM pins) (LEDs)
const int right_LED_green  = 8;
const int right_LED_red    = 9;
const int right_LED_yellow = 10;

//Define Variables for holding sensor data
int rgy; //Right Gyro Y
int rgx; //Right Gyro X
int rfi; //Right Flex Index
int rfm; //Right Flex Middle
int rfr; //Right Force Ring
int rfp; //Right Force Pinky
int rvp; //Right Vibra Palm

//Equilibrium Values
//NOTE: Gyro Values are set in calibration stage, but declared as globals now.
    int rgy_mid;
    int rgy_low;
    int rgy_high;
    int rgx_mid;
    int rgx_low;
    int rgx_high;
const int rfi_straight = 525;
const int rfi_curled   = 730;
const int rfr_low      = 1023;
const int rfr_high     = 600;
const int rfp_low      = 1023;
const int rfp_high     = 700;
const int rvp_low      = 1023;
const int rvp_high     = 400;

//This function will format and print number to be sent over XBee
void print_pretty(int val)
{
    int new_val;
    //All Numbers must range from zero to 255.
    if (val<0)    new_val = 0;
    else if (val>255) new_val = 255;
    else        new_val = val;
    //Add Leading zeros
    if (new_val < 10)
    {
        Serial.print(0);
        Serial.print(0);
        Serial.print(new_val);
    }
    else if (new_val >= 10 && new_val <100)
    {
        Serial.print(0);
        Serial.print(new_val);
    }
    else Serial.print(new_val);
}

void calibrate_gyro(int y_axis_pin, int x_axis_pin)
{
    //If we are talking to the gyroscope sensors, then equilibrium value is at the middle.
    //We will collect 20 data samples in 2 seconds

```

```

//Do some dummy reads first.  This seems to be necessary.
analogRead(x_axis_pin);
delay (500);
analogRead(y_axis_pin);
delay(500);

//Read Values and do some average to determine sensor sensor calibration states
int sumy = 0;
int sumx = 0;
for (int i = 0; i<20; i++)
{
    sumy = sumy + analogRead(y_axis_pin);
    sumx = sumx + analogRead(x_axis_pin);
    delay (100);
}
double avgy = round(sumy/20);
double avgx = round(sumx/20);

rgy_mid = avgy;
rgy_low = 0;
rgy_high = avgy*2;
rgx_mid = avgx;
rgx_low = 0;
rgx_high = avgx*2;
}

void setup()
{
    //Set Pin Directions
    pinMode (pwr_led,          OUTPUT);
    pinMode (right_gyro_4Y,    INPUT);
    pinMode (right_gyro_4X,    INPUT);
    pinMode (right_flex_index,  INPUT);
    pinMode (right_flex_middle, INPUT);
    pinMode (right_force_ring,  INPUT);
    pinMode (right_force_pinky, INPUT);
    pinMode (right_vibra_palm,  INPUT);
    pinMode (right_LED_green,   OUTPUT);
    pinMode (right_LED_red,     OUTPUT);
    pinMode (right_LED_yellow,  OUTPUT);

    //Turn on Pwr LED
    digitalWrite(pwr_led, HIGH);

    //Setup Serial Connection to Computer and Xbee
    Serial.begin (9600);

    //PERFORM 1-TIME SENSOR CALIBRATION
    //Warn the User that synchronization will occur by counting down with LEDs.
    digitalWrite(right_LED_red, HIGH);
    delay(500);
    digitalWrite(right_LED_red, LOW);
    digitalWrite(right_LED_yellow,HIGH);
    delay(500);
    digitalWrite(right_LED_yellow,LOW);
    digitalWrite(right_LED_green,HIGH);
    delay(500);
    digitalWrite(right_LED_yellow,HIGH);
    digitalWrite(right_LED_red, HIGH);
    //The User has been warned, and we will now start calibrating
    calibrate_gyro (right_gyro_4Y, right_gyro_4X);
    //Calibration Has Finished.  Turn off Warning LEDs, and prepare for normal operation
    digitalWrite(right_LED_green, LOW);
    delay(500);

```

```

digitalWrite(right_LED_yellow, LOW);
delay(500);
digitalWrite(right_LED_red, LOW);
delay(500);
//Go!
}

void loop()
{
  //Read 3.3V Sensor Values
  analogReference(EXTERNAL);           //3.3V Reference
  rgy = analogRead(right_gyro_4Y);     //Right Gyroscope Y Axis
  rgx = analogRead(right_gyro_4X);     //Right Gyroscope X Axis
  //Read 5V Sensor Values
  analogReference(DEFAULT);           //5V Reference
  rfi = analogRead(right_flex_index);  //Right Flex Index
  rfm = analogRead(right_flex_middle); //Right Flex Middle
  //NOTE: Force Sensors have some noise problems, so we will average them
  int sumr = 0;
  int sump = 0;
  for (int i=0; i<10; i++)
  {
    sumr = sumr + analogRead(right_force_ring); //Right Force Ring
    sump = sump + analogRead(right_force_pinky); //Right Force Pinky
  }
  rfr = round(sumr/10); //Right Force Ring
  rfp = round(sump/10); //Right Force Pinky
  //NOTE: vibration sensors have some noise problems, so we will average them
  int sumv = 0;
  for (int i=0; i<5; i++)
  {
    sumv = sumv + analogRead(right_vibra_palm); //Right Vibra Palm
  }
  rvp = round(sumv/5); //Right Vibra Palm

  //Map Values for Transmission
  int RGX = map(rgx, rgx_low,      rgx_high,  0, 255);
  int RGY = map(rgy, rgy_low,      rgy_high,  0, 255);
  int RFI = map(rfi, rfi_straight, rfi_curled, 0, 255);
  int RFR = map(rfr, rfr_low,      rfr_high,  0, 255);
  int RFP = map(rfp, rfp_low,      rfp_high,  0, 255);
  int RVP = map(rvp, rvp_low,      rvp_high,  0, 255);

  //Corrects for a Sensor Crossover Problem
  if(RFR > 0) RFP = RFP - RFR/5;
  //if(RGY > (127+20) || RGY < (127-20)) RVP = RVP - RGY/5;
  //else if(RGX > (127+20) || RGX < (127-20)) RVP = RVP - RGX/5;

  //We integrate data from the Y acceleration to determine a rough position vector
  //What is the current Turning State?
  if (RGY > (127-left_bias) && RGY < (127+right_bias) && old_state_turn == 1)
new_state_turn = 1;
  else if (RGY > (127-left_bias) && RGY < (127+right_bias) && old_state_turn == 2)
new_state_turn = 2;
  else if (RGY > (127-left_bias) && RGY < (127+right_bias) && old_state_turn == 3)
new_state_turn = 3;

  else if (RGY >= (127+right_bias) && old_state_turn == 1) new_state_turn = 2;
  else if (RGY >= (127+right_bias) && old_state_turn == 2) new_state_turn = 3;
  else if (RGY >= (127+right_bias) && old_state_turn == 3) new_state_turn = 3;

  else if (RGY <= (127-left_bias) && old_state_turn == 1) new_state_turn = 1;
  else if (RGY <= (127-left_bias) && old_state_turn == 2) new_state_turn = 1;
  else if (RGY <= (127-left_bias) && old_state_turn == 3) new_state_turn = 2;

```

```

else new_state_turn = 2;

//Perform Turn Angle Integration
if (new_state_turn == 1) //Glove is left
{
    angle = 0;
    digitalWrite(right_LED_red,    LOW);
    digitalWrite(right_LED_green,  HIGH);
    digitalWrite(right_LED_yellow, LOW);
}
else if (new_state_turn == 2) //Glove is in middle
{
    angle = 127;
    digitalWrite(right_LED_red,    HIGH);
    digitalWrite(right_LED_green,  LOW);
    digitalWrite(right_LED_yellow, LOW);
}
else //Glove is right
{
    angle = 255;
    digitalWrite(right_LED_red,    LOW);
    digitalWrite(right_LED_green,  LOW);
    digitalWrite(right_LED_yellow, HIGH);
}

//Be Ready to get the new state in the next iteration of the loop
old_state_turn = new_state_turn;

//SEND OUT DATA...
//Send data over XBee
if (debug == false)
{
    Serial.print('.');
    print_pretty(RGX);
    print_pretty(angle);
    print_pretty(RFI);
    print_pretty(RFR);
    print_pretty(RFP);
    print_pretty(RVP);
    while (Serial.read() != 'a');
}
//Print Mapped Debug Info
else if (debug == true && debug_map == true)
{
    print_pretty(RGX);
    Serial.print("  ");
    print_pretty(angle);
    Serial.print("  ");
    print_pretty(RFI);
    Serial.print("  ");
    print_pretty(RFR);
    Serial.print("  ");
    print_pretty(RFP);
    Serial.print("  ");
    print_pretty(RVP);
    Serial.println();
    delay(200);
}
//Print Raw debug info
else if (debug == true && debug_map == false)
{
    Serial.print(rgx);
    Serial.print("  ");
    Serial.print(rgy);
}

```

```

    Serial.print("    ");
    Serial.print(rfi);
    Serial.print("    ");
    Serial.print(rfr);
    Serial.print("    ");
    Serial.print(rfp);
    Serial.print("    ");
    Serial.print(rvp);
    Serial.println();
    delay(800);
}
}

```

## RC Car Arduino Code

```

/*****
**** SUDOGLOVE RC CAR PROGRAM ****
**** INFO 4320 Final Project, Spring 2010 ****
**** Cornell University ****
**** Copyright: Jeremy Blum, Jow Ballerini, Alex Garcia, and Tiffany Ng ****
*****/

```

```

unsigned long millis(void);

```

```

//H-Bridge Pins

```

```

const int mPin1 = 10;
const int mPin2 = 11;
const int mPin3 = 5;
const int mPin4 = 6;

```

```

//SOMO pins and constants

```

```

const int clk = 14;
const int data = 15;
const int busy = 16;
const unsigned int VOLUME_0 = 0xFFF0;
const unsigned int VOLUME_1 = 0xFFF1;
const unsigned int VOLUME_2 = 0xFFF2;
const unsigned int VOLUME_3 = 0xFFF3;
const unsigned int VOLUME_4 = 0xFFF4;
const unsigned int VOLUME_5 = 0xFFF5;
const unsigned int VOLUME_6 = 0xFFF6;
const unsigned int VOLUME_7 = 0xFFF7;
const unsigned int PLAY_PAUSE = 0xFFFE;
const unsigned int STOP = 0xFFFF;

```

```

//Headlights Pin

```

```

const int headlight_pin = 12;

```

```

//Siren Lights Pins

```

```

const int sPin1 = 8;
const int sPin2 = 9;

```

```

//Hold Serial commands

```

```

char buffer[18]; //There are 6 3-digit commands
const int buffer_len = 18;

```

```

//State variables

```

```

boolean characterSent = false;
boolean sirenOn = false;
boolean hlightsOn = false;
boolean sounds = false;
boolean leftRight = false;
boolean soundsH = false;

```

```

//Sensor Values

```

```

int something = 0;

```

```

int accel = 0;
int turn = 0;
int sirenNLights = 0;
int honk = 0;
int reverse = 0;

//Threshold values for switching commands
const int headlightsThreshold = 30; //30
const int sirenLightsThreshold = 80; //90
const int sirenSoundThreshold = 160; //200
const int surfaceThreshold = 190; // 190 for carpet, 150 for hard surface
const int honkThreshold = 200;
const int reverseThreshold = 100;

//Clears the buffer coming in from signal
void clearBuffer(){
  for(int i = 0; i < buffer_len; i++){
    buffer[i] = 0;
  }
}

//Send command to SOMO for sounds
void sendCommand(unsigned int command) {
  // start bit
  digitalWrite(clk, LOW);
  delay(2);

  // bit15, bit14, ... bit0
  for (unsigned int mask = 0x8000; mask > 0; mask >>= 1) {
    if (command & mask) {
      digitalWrite(data, HIGH);
    }
    else {
      digitalWrite(data, LOW);
    }
    // clock low
    digitalWrite(clk, LOW);
    delayMicroseconds(200);

    // clock high
    digitalWrite(clk, HIGH);
    delayMicroseconds(200);
  }

  // stop bit
  delay(2);
}

void setup(){
  // //These Pins are Outputs
  pinMode(sPin1, OUTPUT);
  pinMode(sPin2, OUTPUT);
  pinMode(clk, OUTPUT);
  pinMode(data, OUTPUT);
  pinMode(busy, INPUT);
  pinMode(headlight_pin, OUTPUT);

  Serial.begin(9600);
  clearBuffer();

  //Keep motor off on initialization
  analogWrite(mPin1, 0);
  analogWrite(mPin2, 0);
  analogWrite(mPin3, 0);
  analogWrite(mPin4, 0);

```

```

//Set Default Values for SOMO
digitalWrite(clk, HIGH);
digitalWrite(data, LOW);
sendCommand(VOLUME_4);
delay(50);
sendCommand(0x0002);
delay(50);
while(digitalRead(busy) == HIGH);
delay(2000);
Serial.flush();
}

void loop(){
//Sending a character tells the glove unit to deliver more data
if(!characterSent){
    Serial.print('a');
    characterSent = true;
}

//Get commands
if(Serial.available()){
    characterSent = false;
    char check = Serial.read();

    if(check == '.')
    {
        for (int i=0; i < Serial.read();
            delay(10);
        }

//Get the data from the XBee Transmission
char driveTrain[3] = {
    buffer[0], buffer[1], buffer[2]    };
char steering[3] = {
    buffer[3], buffer[4], buffer[5]    };
char headlights[3] = {
    buffer[6], buffer[7], buffer[8]    };
char sirenlights[3] = {
    buffer[9], buffer[10], buffer[11]   };
char sirenSounds[3] = {
    buffer[12], buffer[13], buffer[14]  };
char horn[3] = {
    buffer[15], buffer[16], buffer[17]  };

//Parse buffer into our sensor values
something    = 100*(driveTrain[0] - '0') + 10*(driveTrain[1]-'0') + (driveTrain[2]-
'0');
turn        = 100*(steering[0] - '0') + 10*(steering[1]-'0') + (steering[2]-'0');
accel       = 100*(headlights[0] - '0') + 10*(headlights[1]-'0') + (headlights[2]-
'0');
reverse     = 100*(sirenlights[0] - '0') + 10*(sirenlights[1]-'0') + (sirenlights[2]-
'0');
sirenNLights = 100*(sirenSounds[0] - '0') + 10*(sirenSounds[1]-'0') + (sirenSounds[2]-
'0');
honk        = 100*(horn[0] - '0') + 10*(horn[1]-'0') + (horn[2]-'0');

//Turn on headlights
if(sirenNLights < headlightsThreshold)
{
    hlightsOn=false;
    sirenOn=false;
    sounds=false;
}
}

```

```

}
else if((sirenNLights >= headlightsThreshold) && (sirenNLights < sirenLightsThreshold))
{
  hlightsOn = true;
  sirenOn=false;
  sounds = false;
}
//turn on siren lights by pressing pinky twice as hard
else if((sirenNLights >= sirenLightsThreshold) && (sirenNLights <
sirenSoundThreshold))
{
  hlightsOn=true;
  sirenOn=true;
  sounds = false;
}
//turn on siren sounds by pressing pinky three times as hard
else if(sirenNLights >= sirenSoundThreshold)
{
  hlightsOn=true;
  sirenOn=true;
  sounds=true;
}

// Control Headlights
if(hlightsOn){
  digitalWrite(headlight_pin, HIGH);
}else{
  digitalWrite(headlight_pin, LOW);
}

// Control Siren Lights
if(sirenOn) {
  if(leftRight) {
    digitalWrite(sPin1, HIGH);
    digitalWrite(sPin2, LOW);
  }
  else {
    digitalWrite(sPin1, LOW);
    digitalWrite(sPin2, HIGH);
  }
}

leftRight = !leftRight;
} else if(!sirenOn){
  digitalWrite(sPin1, LOW);
  digitalWrite(sPin2, LOW);
}

//Acceleration Commands
if(accel < surfaceThreshold){
  analogWrite(mPin1, 0);
  analogWrite(mPin2, 0);
}else{
  if(reverse > reverseThreshold){
    analogWrite(mPin1, 0);
    analogWrite(mPin2, accel);
  } else {
    analogWrite(mPin1, accel);
    analogWrite(mPin2, 0);
  }
}

//Turning Commands

```

