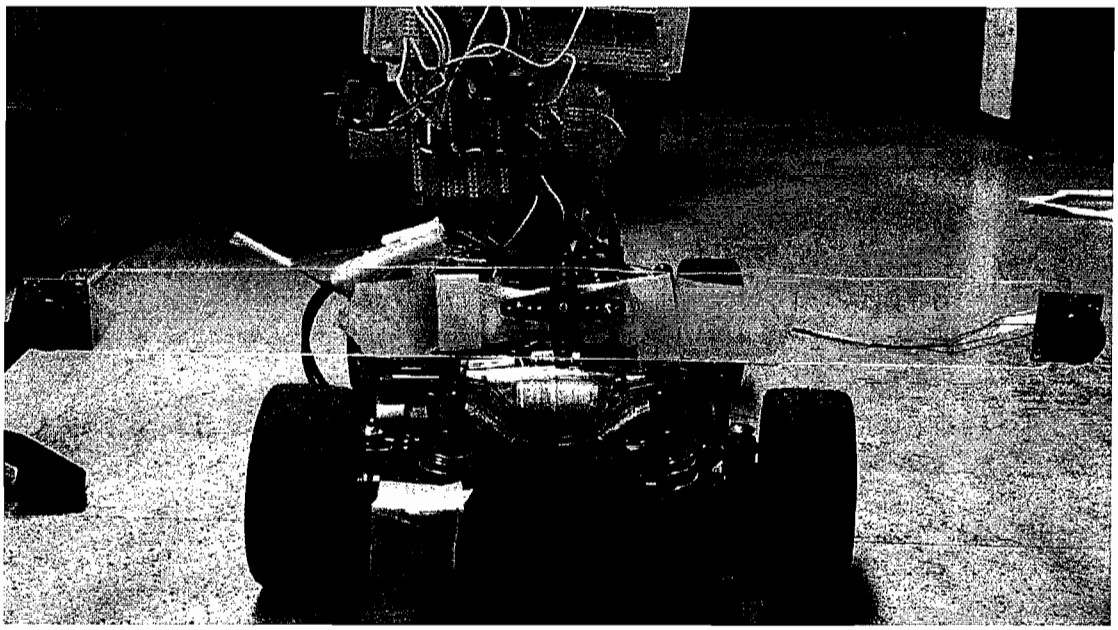
HYSET Example Med

A

STALKER 2007

Seeking **T**riangulating **A**utomated **L**oad-**K**ey **E**ngineered **R**obot
is a motorized car/robot that that will follow a flat surface/object/person. The idea is that when the surface is drawn away from the robot, the robot will be able to direct a path to the surface by which direction the surface turned. This opens multitudes of applications such as shopping carts, baby strollers, and kamikaze bombers without the cost of human lives.

EECS 129B COMP SENIOR PROJECT, klefstad



Adam Clark
Gary Shigemoto
Chi Tran
David Tseng

Description

In short, the project is to create a car/robot that follows another object. In the prototype, a car will follow a flat surface using ultrasonic emitters and detectors. Furthermore, wireless transmission of data from the sensors such as accelerometer, angle steering, and temperature are incorporated into the system. This device is useful in many fields and can be applied to virtually any circumstances. To name a few applications in which this device can be implemented:

1) In travel, luggage will follow one specific person (first person it locks onto)
2) In law enforcement, carry a subject not cooperating
3) In exploration, a helping hand to carry equipment on excavation site
4) In laboratory, a table that carries all commonly used chemicals for easy access
5) In formal events, there will always be a place to put your drink down
6) In military, tagging a target with a RF beacon and letting the bomb follow it's target until a set distance then trigger bomb
7) And much more...

The project itself is divided into 2 realms: (1) **mechanical** and (2) **electrical** (which also includes microcontrollers and software).

The **mechanical** realm includes the basic construction of the mechanical translation and steering of the car/robot itself. It includes the construction of the car and correct gear box in order to step down a high RPM motor. All the mechanical components, including the shells of construction, were assembled from a RC car set.

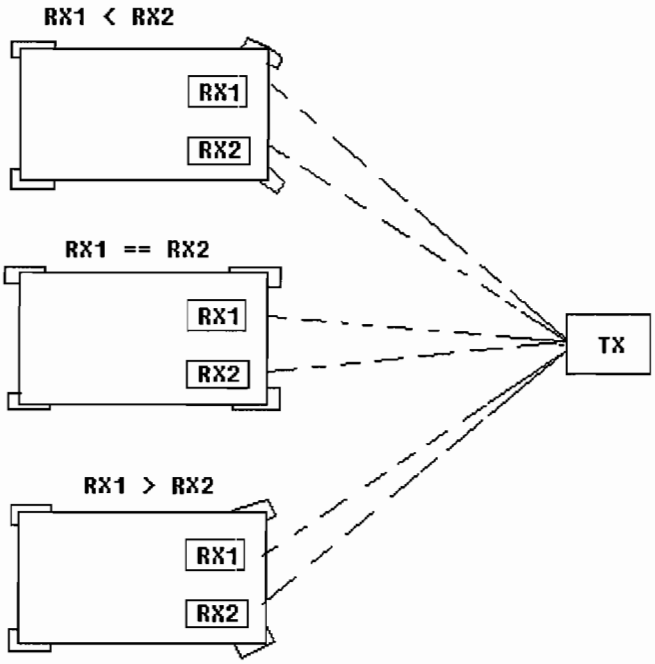
The **electrical** portion of the project is converting the electrical signals and organizing them from the microcontroller to control the servos (which converts an electrical signal to mechanical motion), which will ultimately control the position of the car/robot. Because the servos are controlled via pulse width modulation (digitally), precision in position may be increased with proper calibrations. For movement of the car, an onboard microcontroller will output pulse widths modulated signals to both the steering and the throttle servos based on data from the ultrasonic sensors. A description of the control algorithm will be discussed later. Aside from the control system of the car/robot, onboard sensors including accelerometer and temperature sensor is integrated into the car. The data from these sensors, as well as the positioning system, will be transmitted, via xbee, to a remote location that will monitor the signals.

The last portion of the STALKER 2007 is the signal processing of the electrical signals from the ultrasonic transceivers (sonar) on the car/robot. The strength of the signal and the angle of the object are essential information that must be retrieved, or demodulated, in terms of signal processing. The strength of the signal will give the relative distance of car/robot from the object being followed. The angle of the signal converts the scalar distance into a vector distance so that the car/robot knows the direction of the displacement.

In short, the first phase prototype milestones of the STALKER 2007 is as follows:

1) Construction of all mechanical parts
2) Servo calibration and characterization in terms of pulse width modulation and mechanical displacement
3) Integration of microcontroller into the control system of the servos
4) Signal readings from the ultrasonic transceivers and signal comparisons using the microcontroller
5) The car/robot will then translate forward until the ultrasonic sensors detect a intensity threshold.
6) If the target that it's suppose to follow is not in range, it will simply stay still until it's back in range

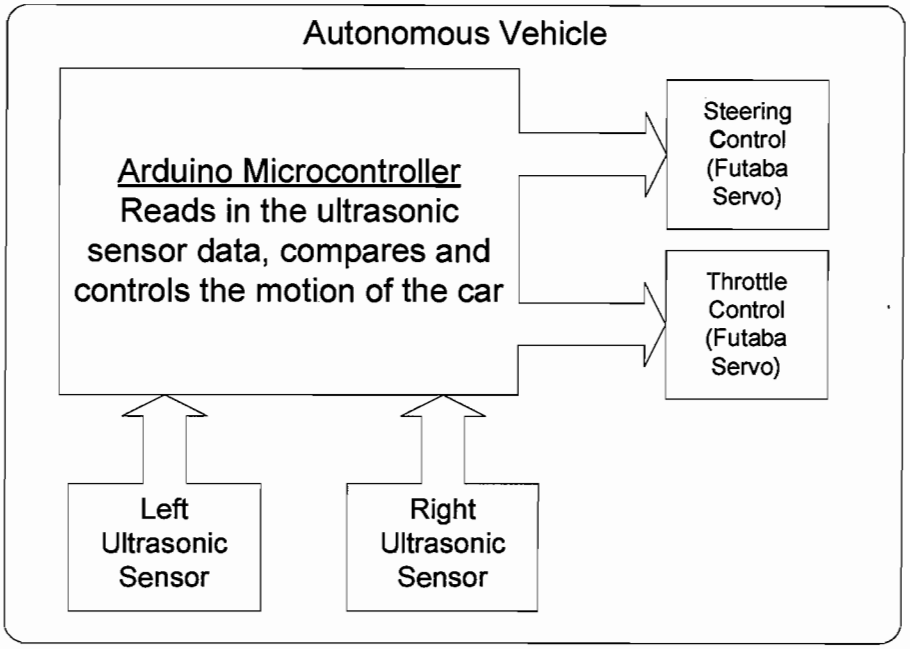
How Directional Control Works

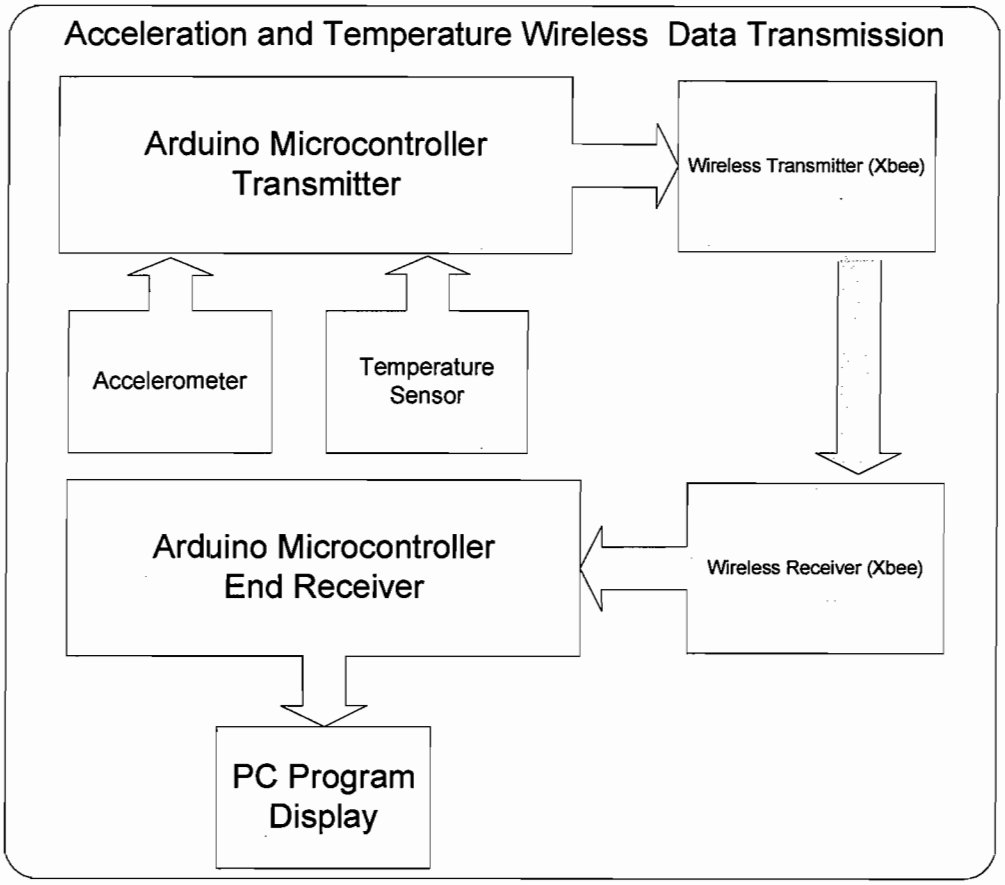


This is the overview of the car/robot in which TX is the "wall" tilted at an angle proportional to turning radius, RX1 and RX2 are the ultrasonic transceivers. When the intensity detected on RX1 < RX2 then the car/robot will know the wall is moving to the right of the current location. When the two intensities are detected to be the same, then the car/robot will know the wall is straight ahead. In this case when the two intensities are the same, the car/robot will proceed foreword. The last condition is when intensity detected in RX1 > RX2 where the car/robot will know the wall is to the left. Notice that the car/robot will only turn clockwise or counterclockwise depending on the intensity. When the difference in intensity between the two transmitters reaches a threshold, then the car/robot will proceed forward (the car/robot will never go in "reverse").

Pulse width modulation is used to control the two servos which controls the direction of the car. The period of the pulse is 18.08ms. Neutral throttle pulse width on the throttle servo is 1500µs and forward is an additional 150µs to the neutral pulse width while reverse is decreasing the neutral pulse width by 150µs. For the steering servo, straight is 1385µs. Steering right and left is proportional to adding and subtracting proportionally to the straight pulse width. Pulse widths are outputted from a microcontroller to control both the steering and throttle of the car. For this particular project, every forward motion is accompanied by a backward motion to maintain a controlled, accurate and steady motion. Therefore, the car's response is

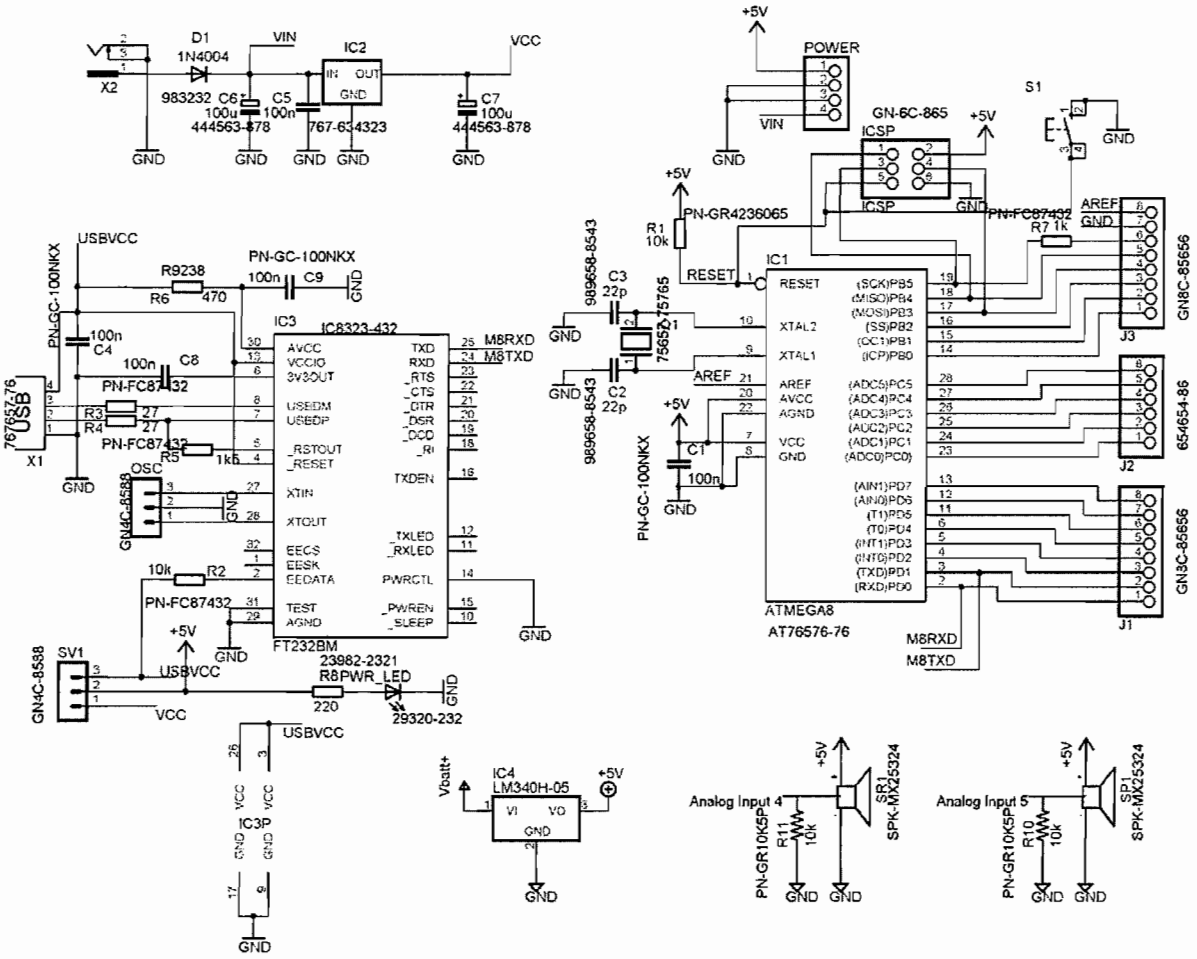
System Level Block Diagram



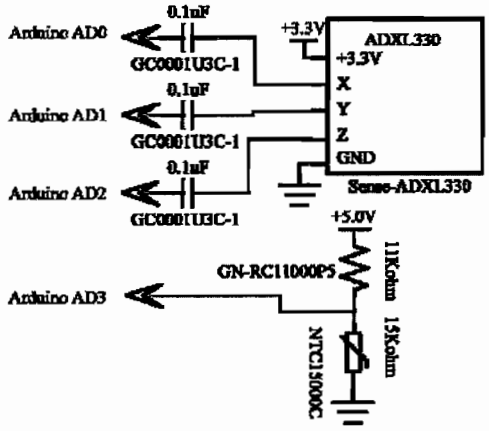


Circuit Level Block Diagram (ALL the details, including part #s, pin #s, resistor values, etc.)

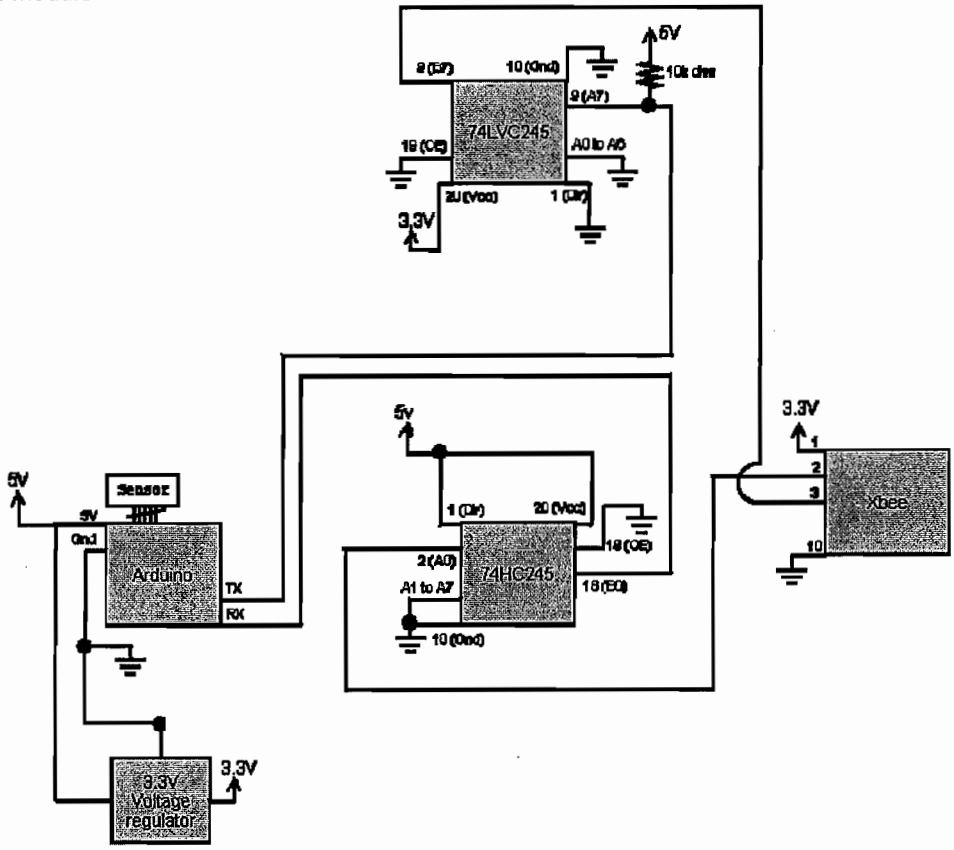
Car control system



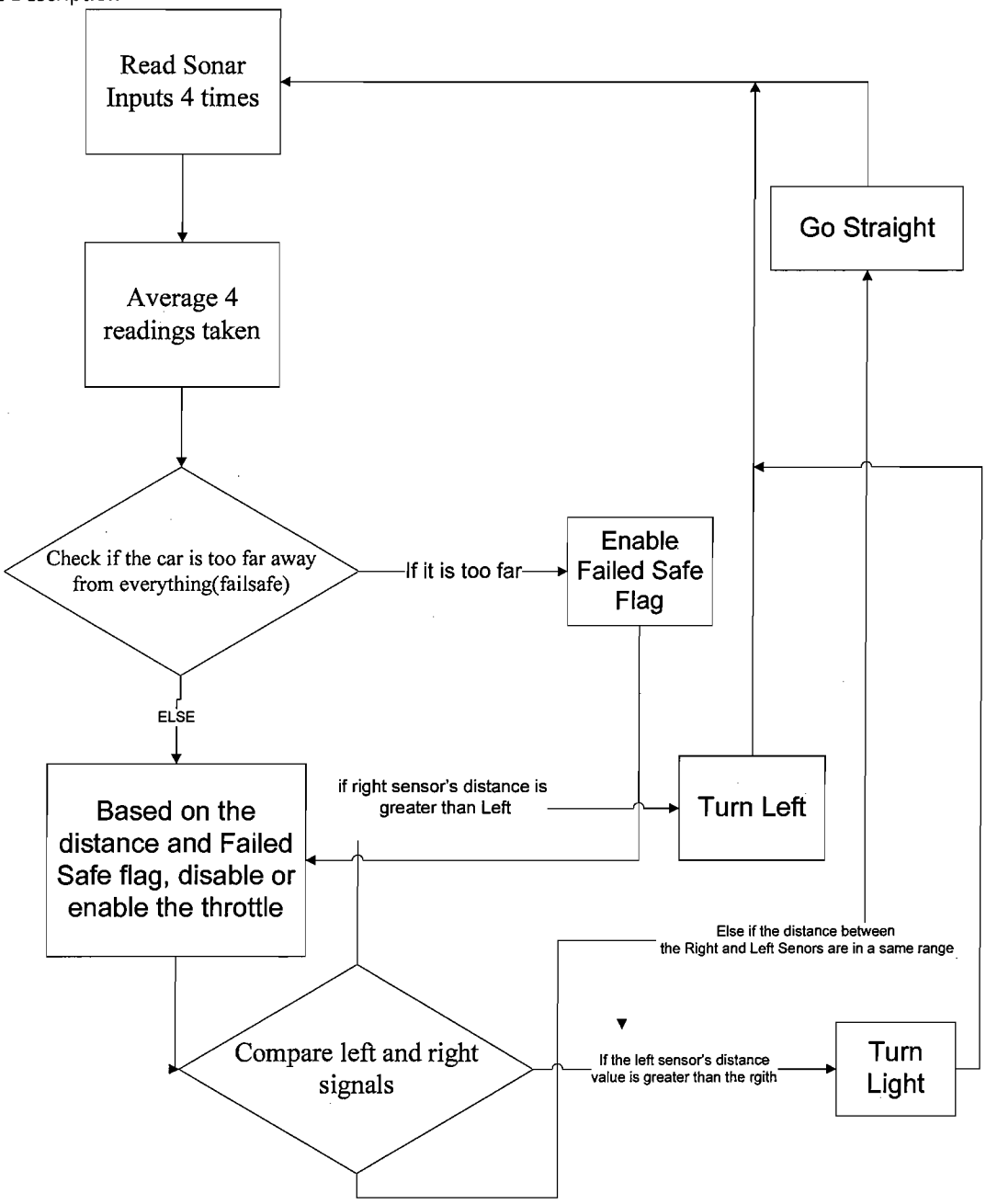
Sensor system (accelerometer and thermistor)



Wireless module



Software Description



The following is the actual software with a lot of comments describing the functions of each module of code. This includes the physical response, the interface with pins and sensors and control units etc.

```
/* Ultrasound Sensor
*-------------------
* Written by: Adam Clark and Chi Tran
*
* Description: The program briefly pulses a control pin on both sonar sensors, making
*                                    them measure the distance to the surface in front of them. The sensors
*                                    then return an analog voltage, which is linearly proportional to the
*                                    distance. The sensors are pulsed 4 times, after which the readings
```

```
 *                          are averaged to produce a smoother signal.  The program then determines
 *                          if the distances are both greater than a pre-set threshold, in which case
 *                          the car will accelerate to keep up with its target; the throttle is held on
 *                          for the amount of time that it takes to read in the sonar values, or about
 *                          250ms.  Finally, the program calculates the difference between the left and
 *                          right values, and scales the steering proportionally to the difference.
 *
 *
 *
 *
 *
 */
#include <math.h>

#define ledPin 13                       // LED connected to digital pin 13
#define pwmSteerServo 10                //Steering Servo PWM Control
#define throttleServo 9                 //Throttle Servo PWM Control
#define SonarControlPin 12              //Control pin for telling the sonar beacons to check the distance

#define leftSonarSignal 0               // Left Ultrasound signal pin
#define rightSonarSignal 1              // Right Ultrasound signal pin

#define zeroValue 5                     // the difference threshold necessary for the car to turn

#define NEUTRAL_STEER 1385              // neutral pulse length for steering
#define NEUTRAL_THROTTLE 1500           // neutral pulse length for throttle

#define STOP_THRESHOLD 4                // threshold for throttle differences
#define maximumDifference 10            // the maximum difference between left and right values

#define throttleThreshold 22            // The threshold for the throttle turning on
#define throttleController 5            // input for a potentiometer for real-time throttle control
#define failSafe 144                    // distance for fail-safe control system
#define counterValue 10                 // counter for throttle control

int leftReading = 0;                    // used to store left distance reading; twice the actual distance in inches
int rightReading = 0;                   // used to store right distance reading; twice the actual distance in inches

int difference = 0;                     // difference between left and right readings

int steerValue = 0;                     // the value that the steering pulse length is changed by
int selTarget = 0;                      // the selected sensor for one part of the distance comparison
int lastTarget = 0;                     // the distance value from the previous loop

bool throttleOn = false;                // tracks whether or not the throttle was on from the last loop
bool tooFarAway = false;                // check to see if the car is too far away from everything
bool timeToStop = false;                // tracking so the car won't ram the lead car
bool turnThrottleOn = false;            // flag for tracking the throttle

int counterTracker = 0;                 // used to track throttle changes
int counter = 0;                        // used to track the time that the throttle has been on


void setup()
{
        Serial.begin(9600);                             // Sets the baud rate to 9600
        pinMode(ledPin, OUTPUT);                        // Sets the digital pin as output

        pinMode(pwmSteerServo, OUTPUT);         // sets the pin for controlling the steering servo as an output
        pinMode(throttleServo, OUTPUT);         // sets the pin for controlling the throttle servo as an output

        TCCR1A = 0x00;                          // sets timer control bits to PWM Phase and Frequency Correct mode
        TCCR1B = 0x12;                          // sets timer control bits to Prescaler N = 8
        ICR1 = 0x46A0;                          // Upper Timer Limit = 18080 (in hex) equals 18080usb

        digitalWrite(SonarControlPin, LOW);     // writes both control pins low, to ensure they don't

}
```

```
void loop()
{
        leftReading = 0;                                    // clears the readings to make sure things start right
        rightReading = 0;
        //counter = 0;

        //variableThrottle = analogRead(throttleController);
        //variableThrottle = ((variableThrottle * 6) / 1024) + 0.5;
        //variableThrottle += 8;

        ReadSonarInputs();                                  // read the inputs

        ConvertReadings(leftReading, rightReading);         //average the input readings

        tooFarAway = FailSafeCheck(leftReading, rightReading);  // check if the failsafe is met

        //check if we need to go forward (gas)
  turnThrottleOn = ThrottleFlags(leftReading, rightReading, selTarget, lastTarget);

        ThrottleActivation();                               //activate the throttle

        SteeringControl(leftReading, rightReading);         //determine the turn

        //delay(500);                                       // delay for debugging purposes
}


// Function ReadSOnarInputs reads the sonar sensors 4 times and records those values.
// The sonar sensors can be read every 50ms, and the delays are broken for timing.
// The variable counter is incremented once for every 25ms elapsed, and is used by
// CountTheThrottle to control throttle deactivation.
void ReadSonarInputs()
{
        for (int i=0; i < 4; i++)                           // take 4 readings so that they can be averaged
        {
                digitalWrite(SonarControlPin, HIGH);        // pulse the control pin high
                delay(5);
                digitalWrite(SonarControlPin, LOW);         // set the contorl pin back to low
                delay(20);

                counter += 1;                               // increment counter

                ThrottleControl(counter, throttleOn);       //check counter for throttle control

                delay(20);

                leftReading += analogRead(leftSonarSignal); // read in the values from both sensors
                rightReading += analogRead(rightSonarSignal);

    delay(5);

                counter += 1;                               // increment counter

                ThrottleControl(counter, throttleOn);       // check counter for throttle control
        }
}


// Function ConvertReadings takes in values for the left and right sensors, where the values
// represent 4 readings added together.  They are divided by 4 for calculations, then divided
// by 2 so that the values printed through serial are actual distances in inches.  Finally,
// selTarget and lastTarget are set for distance calculations
void ConvertReadings(int left, int right)
{
```

```
        leftReading = left >> 2;                    // divide the left reading by 4 to average it
        rightReading = right >> 2;                  // divide the right reading by 4 to average it

        int tempLeftReading = leftReading >> 1;     // divide left value by 2 to give the actual reading in inches
        int tempRightReading = rightReading >> 1;   // divide right value by 2 to give the actual reading in inches

        Serial.print("left: ");                     // print out the left value in inches
        Serial.print(tempLeftReading, DEC);
        Serial.print("  right: ");                  // print out the right value in inches
        Serial.println(tempRightReading, DEC);

        selTarget = leftReading;                    // We check the left sensor for general distance calculations

  if (lastTarget == 0)                              // if we're at the beginning
    lastTarget = selTarget;                         // save this reading as the last loop's reading
}

// Function FailSafeCheck checks both the left and right readings, and if either is above the
// failsafe threshold, they are both rounded down, and the function will return true.  Otherwise,
// the function will return false.
boolean FailSafeCheck(int left, int right)
{

        bool failSafeBroken = ((right >= failSafe) || (left >= failSafe));

        if (failSafeBroken)                         //if either value is above the failsafe threshold,
        {
                leftReading = 100;                  //round them down
                rightReading = 100;
                return true;                        // the car is too far away
        }

        else
                return false;                       //otherwise, we're ok, and return false

}

// Function ThrottleFlags takes in the leftReading, rightReading, selected target and previous
// target and performs the necessary comparisons necessary for ThrottleActivation.  it returns
// true if the throttle should be turned on, and false if the throttle should remain off.
boolean ThrottleFlags(int left, int right, int sel, int last)
{

        // if either side is over the throttlethreshold, then catchUp is true
        bool catchUp = ((left >= throttleThreshold) && (right >= throttleThreshold));
        // if the distance increased more than the stop threshold, distanceChange is true
        bool distanceChange = (STOP_THRESHOLD < (sel - last));

        lastTarget = selTarget;                                 // save this distance reading for next time

        if (catchUp || distanceChange)                          // if either are true,
        {
                timeToStop = true;                                      // the throttle should be turned on
                return true;                                            // return true to signify that
        }

        else
                return false;                                           // otherwise, return false

}

// Function ThrottleActivation controls turning on the throttle.  If turnThrottleOn is true
// and tooFarAway passed the failsafe, then the throttle is turned on.  If the throttle shouldn't
// be on, and it's timeToStop, then reverse for 300ms to decelerate, then come to a stop.
// otherwise, the car was not moving during the last loop, and should stay stopped.
void ThrottleActivation()
```

```
{

if (turnThrottleOn && (tooFarAway == false))                    // if the throttle needs to be on
        {
                analogWrite(throttleServo, 1650);               // increase the pulse length for the throttle
                throttleOn = true;                              // indicate that the throttle is on

          Serial.println("Throttle is on.");// print that it's on

        }

        else if ((turnThrottleOn == false) && (timeToStop == true))     //otherwise,
        {

                analogWrite(throttleServo, 1350);               // reversse for 300ms
                delay(300);
                analogWrite(throttleServo, 1500);               //then stop
                throttleOn = false;                             // indicate that the throttle was on
                timeToStop = false;                             // clear the brake flag

          Serial.println("Throttle is off.");                   // print that it's of

        }

        else
        {
                analogWrite(throttleServo, 1500);               // reset the throttle to neutral
                throttleOn = false;                             // indicate that the throttle is off

          Serial.println("Throttle is off.");                   // print that it's off
        }
}

// Function ThrottleControl takes in a counter value, and the flag throttleInput.
// if the count is equal to the defined value, the throttle will either be turned off
// or paused momentarily, depending on the value of throttleInput.
void ThrottleControl(int count, bool throttleInput)
{

        if (count == counterValue)                              // if the count is correct,
        {
                counter = 0;                                    // clear the counter

                if (throttleInput == true)                      // then, if the throttle is on from the previous loop
                {
                        delay(20);
                        analogWrite(throttleServo, 1350);       // throw the car into reverse for 125ms
                        delay(125);
                        analogWrite(throttleServo, 1650);       // then turn the throttle back on
                        throttleOn = false;                     // clear the throttle check
                }

                else                                                    // otherwise,
                {
                        //delay(15);
                        analogWrite(throttleServo, 1500);               // just make sure the throttle is off
                        throttleOn = false;                             // and the check is cleared
                        //delay(35);
                }
        }
}

// Function SteeringControl controls how far and in which direction the car should turn.
// If the difference is less than the zeroValue threshold, then the car shouldn't turn.
// If the difference is greater than the threshold, then the car should turn right.
// If the difference is less than the negative value of the threshold, then the car should turn left.
```

```
// THe amount the car turns to either side is controlled by the value of the difference.
void SteeringControl(int left, int right)
{
        difference = left - right;                          // the difference is left - right (so, positive if it's turning right)

        Serial.print("difference is: ");                    // print out the difference
        Serial.print(difference, DEC);
        Serial.print("    ");

        if ((difference < zeroValue) && (difference > (zeroValue*(-1))))    // if the sides are reasonably close together
        {
                Serial.println("Go Straight.");
                analogWrite(pwmSteerServo, NEUTRAL_STEER);  // go straight
        }

        else if (difference >= zeroValue)                   // if the left side is farther away
        {

                if (difference > maximumDifference)         // if the difference is very large,
                        difference = 170;                   // make the sharpest turn

                else                                        // otherwise,
                        difference *= 17;                   // convert the difference into a steering value

                analogWrite(pwmSteerServo, NEUTRAL_STEER - difference);     // turn right

                Serial.println("Turn right.");              // Serial output
        }

        else if (difference <= ((-1)*zeroValue))        // if the right side is farther away
        {
                difference *= -1;                           // make the difference positive

                if (difference > maximumDifference)         // if the difference is very large,
                        difference = 225;                   // turn as sharply as allowed
                else
                        difference *= 22.5;

                analogWrite(pwmSteerServo, NEUTRAL_STEER + difference);     // turn left

                Serial.println("Turn left.");               // Serial output
        }

}
```

System Test Plan

Mechanical control using microcontroller [Steering servo & Throttle servo		
	Cause	**Effect**
	Connect the controller and radio receiver to the throttle servos. Signal servo to full forward speed. Then connect controller to oscilloscope.	servo is responding mechanically in a unidirection. Oscilloscope reading of controller should match profile
	signal throttle servo to full backward speed and check if servo is responding mechanically. Then connect controller to oscilloscope.	servo is responding mechanically in a unidirection fashion (the opposite direction of forward). Oscilloscope reading of controller should match profile
	Connect the controller and radio receiver to the steering servos. Signal servo to turn right. Then connect controller to oscilloscope.	servo is responding mechanically in a unidirection but continuous linear response. Oscilloscope reading of controller should match profile
	Signal steering servo to turn left and check if servo is responding mechanically. Then connect controller to oscilloscope.	servo is responding mechanically in a unidirection fashion (the opposite direction of right). Oscilloscope reading of controller should match profile
Ultrasonic transceivers		
	Cause	**Effect**
	Ultrasonic transceiver's pin layout is as follows: ground (pin 1), 5V (pin 2), trigger (pin 4), analog output (pin 5). With these pins, power on the ultrasonic transceiver by connecting pin 1 and 2 to corresponding voltages. Trigger the distance reading using pin 4 by putting it high and it will cause an analog reading on pin 5. Put object in front of sensor at different distances	Using a volt meter, read the voltages and it should correspond to the following: lower voltage when object is closer to the ultrasonic transceiver and higher voltage when object is out of range or further away.
	To test if the microcontroller is responding correctly to the signals, hook up a two channel DC power supply to pin 4 and 5. By sending voltages between 0 and 5V with each channel	Steering servo should respond relatively to each channel and the throttle will respond to individual intensity. The throttle will trigger at high DC voltage and neutral at low voltages
Sensing units		
	Cause	**Effect**
Accelerometer	Connect the the accelerometer to the following pin layout : analog X, analog Y, analog Z, 3.3V, GND.	The device should power on.
	Use a voltmeter to measure the voltages of the pin Z, Y, and X asix.	The readings of each pin layout should be less than 3.3V
	Rotate the accelerometer so that the X axis is perpendicular to ground.	The X analog reading should be about 3.3V. Y and Z axis pin should be 0V. The concept should follow the Y and Z axis.
Themistor	Connect the variable resistor to one of the input pins of the microcontroller and ground.	This should allow the device to operate.
	Use the microcontroller program to read out the outputs of the analog pin.	The reading on the thermistor is another analog reading in which the

		voltage is inversely proportional to the temperature. To test, use the ohmmeter.
Wireless transmission	The pin-layouts are as follow on the wireless Xbee module: on the wireless sender (TX) module, connect pin 2 from the Xbee module to the Arduino sender-microcontroller's pin TX. For both Xbee's (TX and RX) with its own Arudino microcontroller, connect the Arduino's 5V output pin and ground to the pin 1 and pin 20 of the Xbee, respectively.	This will enable the device to turn on. The TX receiver leds should be blinking yellow to indicate values sent.
	Connect the usb of the receiver-microcontroller to a pc for serial data transfer.	Check values sent between the PC's through the wireless based on the string sent.
Overall Testing		
	Cause	**Effect**
	Programmed microcontroller1 for control of the car and microcontroller2 the sensing and wireless data collection.	Leds should flash multiple times during the process of flashing the microcontroller
	Microcontroller1 is connected to both of the ultrasonic transmitters, steering servo and throttle servo, which is powered by 5 volt voltage regulator.	Once connected, the car should try steering and gas based on the AI programmed on to microcontroller1.
	Microcontroller2 is connected to the thermistor, xbee (3.3V voltage regulated), and accelerometer which is powered by a 5 volt voltage regulator.	This should microcontroller's TX LED should blink to indicate data transfer.

The following is a more detailed testing procedure and setup procedure

Module testing:

Mechanical control using microcontroller [Steering servo & Throttle servo]

Connect the controller and radio receiver to the servos. Put throttle servo to full forward speed and check if servo is responding mechanically. Put throttle servo to full backward speed and check if servo is responding mechanically. Same test can be done to the steering servo in which left correlates to forward and right correlates to backwards.

Measure the pulse widths output from the controller/radio receiver using an oscilloscope in order to characterize each motion (in the case of forward, it has a certain pulse width signature while right turn has a continuous range of signature depending the radius of the right turn). Make sure you know what each wire on the servos mean: ground, high, pulse width is the order from left to right on standard servos.

Measure the output control signals from the microcontroller, where throttle is pin 9 and steering is pin 10) and make sure it matches the signature characterization of the servos. If everything correlates, then the system is in working condition.

Ultrasonic transceivers [sonar]

Ultrasonic transceiver's pin layout is as follows: ground (pin 1), 5V (pin 2), trigger (pin 4), analog output (pin 5). With these pins, power on the ultrasonic transceiver by connecting pin 1 and 2 to corresponding voltages. Trigger the distance reading using pin 4 by putting it high and it will cause an analog reading on pin 5. Using a volt meter, read the voltages and it should correspond to the following: lower voltage when object is closer to the ultrasonic transceiver and higher voltage when object is out of range or further away.

To test if the microcontroller is responding correctly to the signals, hook up a two channel DC power supply to pin 4 and 5. By sending voltages between 0 and 5V with each channel, the steering servo should respond relatively to each channel and the throttle will respond to individual intensity. Every forward motion is accompanied by a backward pulse so that the car does not accelerate out of control.

Sensing units

The accelerometer pin layout is as follows analog X, analog Y, analog Z, 3.3V, GND. This unit can be easily tested in that once powered on, the analog X,Y,Z can be read by a voltmeter. By rotating the accelerometer so that the X axis is perpendicular to ground, the X analog reading should be at 3.3V and Y and Z axis should be 0V. The concept should follow the Y and Z axis.

The thermistor is basically a variable resistor that's sensitive to temperature. The reading on the thermistor is another analog reading in which the voltage is inversely proportional to the temperature. To test if a thermistor works, measure with ohmmeter with respect to temperature.

Wireless

The pin-layouts are as follow on the wireless Xbee module: on the wireless sender (TX) module, connect pin 2 from the Xbee module to the Arduino sender-microcontroller's pin TX. On the wireless-receiver module, connect pin 3 to the Arduino receiver-microcontroller's pin RX. For both Xbee's (TX and RX) with its own Arudino microcontroller, connect the Arduino's 5V output pin and ground to the pin 1 and pin 20 of the Xbee, respectively. Connect the usb of the receiver-microcontroller to a pc for serial data transfer.

Once both wireless modules are powered up and booted, the led from the TX receiver should light up yellow. Once that happens, check on the pc to see if the data are outputted on the serial screen correctly.

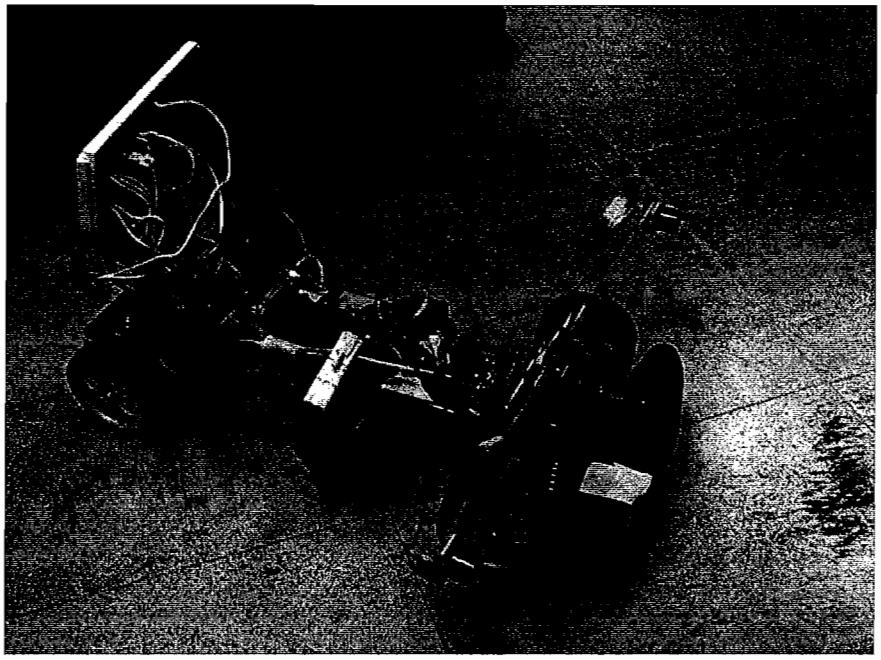
Overall Testing

Once all the components are connected correctly, microcontroller1 is designated for control of the car and microcontroller2 is designated for the sensing and wireless data collection. Microcontroller1 is connected to both of the ultrasonic transmitters, steering servo and throttle servo, which is powered by 5 volt voltage regulator. Microcontroller2 is connected to the thermistor, xbee (3.3V voltage regulated), and accelerometer which is powered by a 5 volt voltage regulator. Once all the hardware is connected, the following is the procedure for testing:

1) Turn on a separate remote control car that it shall follow
2) Hook up laptop to the receiving xbee/arduino module and make sure it's reading serial inputs.
3) Boot up The STALKER
    a. Drive the car right and the STALKER will follow the remote control car to the right, the STALKER will then stop when the distance between the car and STALKER reaches a threshold
    b. Drive the car left and the STALKER will follow the remote control car to the left, the STALKER will then stop when the distance between the car and STALKER reaches a threshold
    c. Go straight and the STALKER will go straight, it will then stop when the distance between the car and STALKER reaches a threshold
    d. Drive the remote control car away at full speed (out of range of the STALKER) and the STALKER should stop because the stalking object is no longer in sight.
    e. Repeat steps a-d with an actual human. Although the stalking is not perfected for an actual human stalking, keep in mind that this unit is only a prototype. With further funding, this may be improved.

Cost Analysis

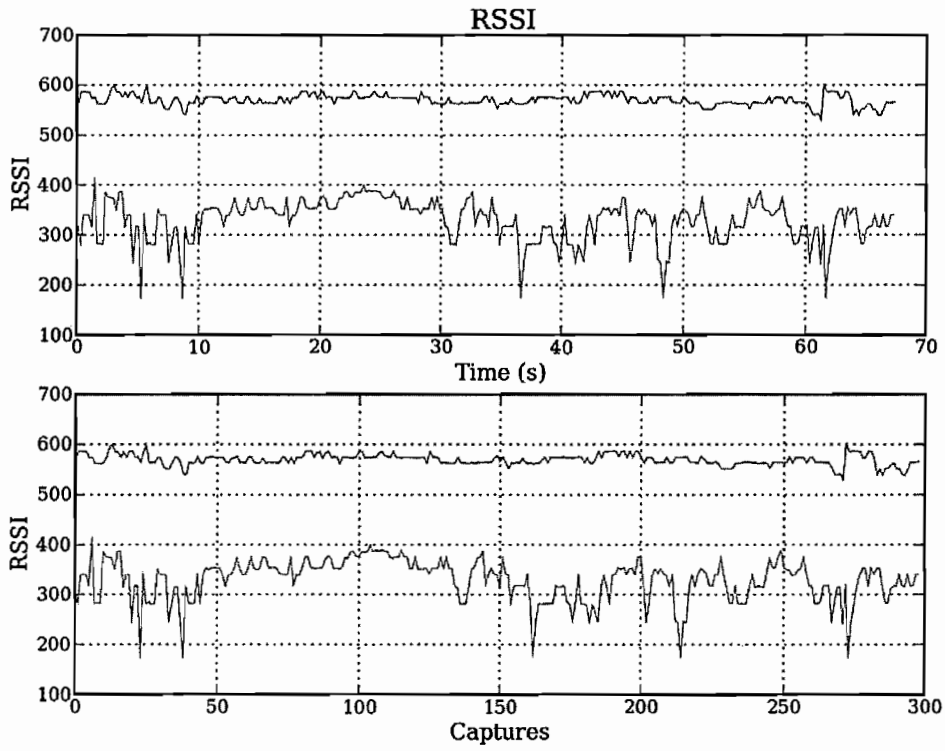
Part	Amount	Cost	Part Total
RC Car with servo control	1	$150.00	$ 150.00
Arduino USB board with Atmel Atmega8	2	$ 31.95	$ 63.90
Breadboard (Experimentor 350)	3	$ 8.39	$ 25.17
1k resistor	2	$ 0.65	$ 1.29
47k resistor	5	$ 0.20	$ 0.99
wire (30 gauge)	2	$ 5.99	$ 11.98
XBee wireless module	1	$ 19.00	$ 19.00
Xbee PRO wireless module	1	$ 32.00	$ 32.00
5V Regulator	2	$ 0.95	$ 1.90
IC LDO REGULATOR +3.3V TO-220	2	$ 0.77	$ 1.54
IC TXRX OCTAL BUS 3TATE 20DIP	2	$ 0.58	$ 1.16
IC OCT BUS XCVR TRI-ST 20-DIP	2	$ 0.60	$ 1.20
20 PIN FEMALE HEADERS	2	$ 2.85	$ 5.70
USB, a to b, cable	2	$ 1.55	$ 3.10
Ultrasonic Range Finder - Maxbotix LV-EZ1	2	$ 24.95	$ 49.90
Thermistor,NTC,K(+/-10%)	1	$ 0.52	$ 0.52
Accelerometer Breakout Board - ADXL202JE +/-2g	1	$ 34.95	$ 34.95
Subtotal			$ 404.30
Tax	Rate:	7.75%	$ 31.33
Shipping			$ 73.69
TOTAL			$ 509.32



Summary

The final project turned out to be a robot/car the follows an object through ultrasonic readings. The most ideal case is when the object that the car follows is a flat wall attached another car (as in a convoy suggested by the professor). This utilizes the comparison of two ultrasonic signals and the direction of the car is determined by these two readings. Although the signal comparison is original to the proposal from last year, the way the signal is collected is different.

In the original proposed project, the triangulation (the control system used to control the direction of the car) is suppose to be done by using radio frequency domain with 3 wireless transceivers. However, after many weeks of experimentation, we realized that we were limited by the accuracy of the RSSI (relative signal strength) signal output. Below are some data taken that provides evidence for the instability of RSSI.



Red curve and green curve corresponds to the right xbee and left xbee data respectively. This data was collected when the beacon is at a constant distance from the two receivers. As illustrated, the signal is too unstable for any consistent control in the use of triangulation system.

In order to resolve this hardware limitation and because the total spending on the project was exceeding $600.00, we needed a way to overcome this hardware issue without having to invest more on hardware. Using ultrasonic transceivers to measure distance was the best substitute because it also measures distance with respect to an object and therefore we could still use the triangulation algorithm/theory. The only difference between the two approaches (RF vs ultrasonic) is the signature on the object it follows. In other words, the ultrasonic approach follows any solid object while the RF approach only follows the person/moving object that has the beacon.

Besides the hardware limitations, we accomplished everything we proposed + MORE. This includes a robot/car that follows an object and data from sensors (accelerometer and thermistor) that is transmitted wirelessly to a xbee that is plugged into a personal computer.