EECS 129B Winter 2008 Final Project Report

Team Members

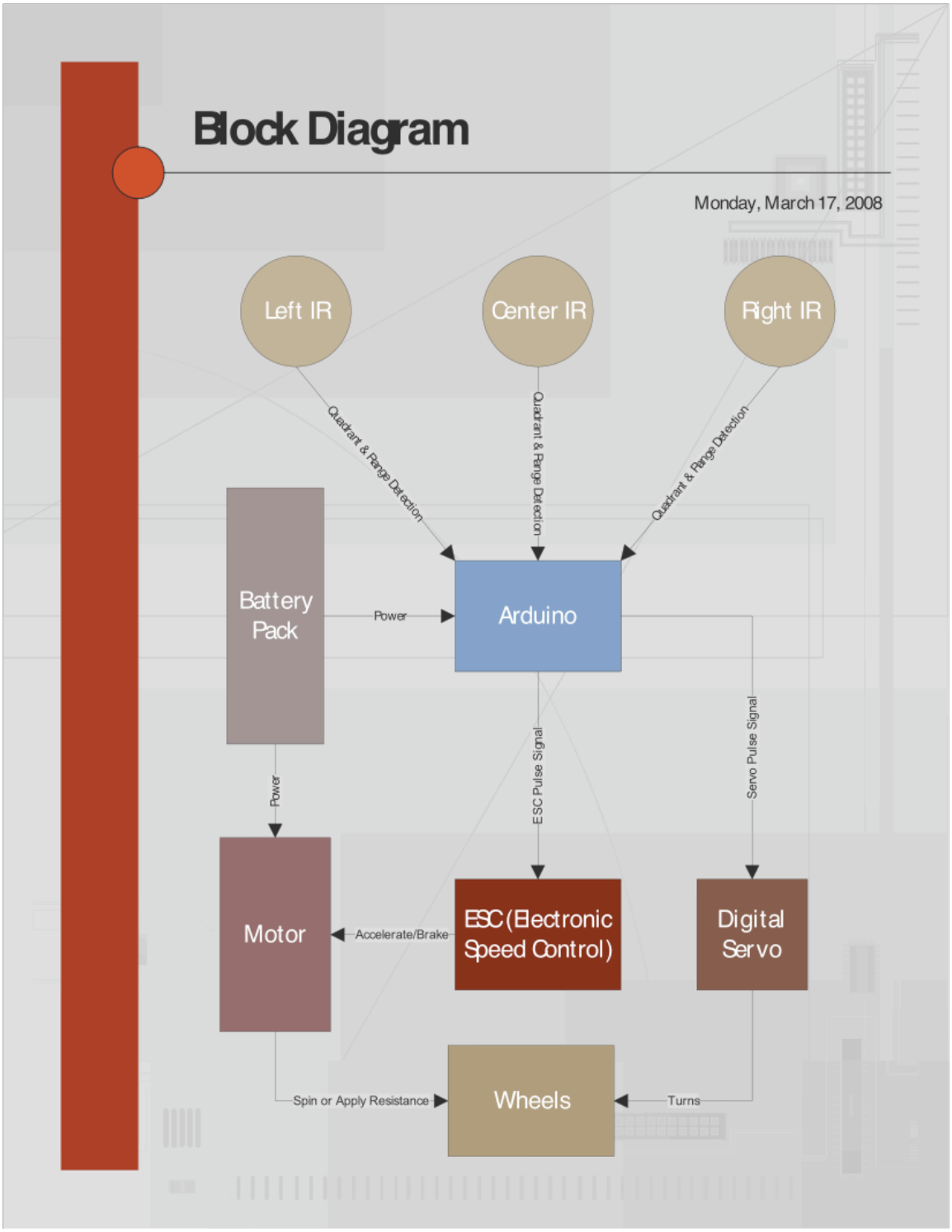
Last Name	First Name
Buchanan	Brian
Keschrumrus	Vin
Snow	Robert

RC Convoy System / ACES: Autonomous Convoy Engagement System

1. Abstract

Our project is a computer-controlled car that follows the path of a human-controlled leader car. The idea is that a pair of vehicles traveling a long distance on a freeway could “caravan” all the way to their destination with only one driver. The following car follows its leader at a safe distance and is able to track it along a path that includes turns that are comparable to a U.S. Freeway and do so without human control. The key hardware used to implement our project is infrared sensors hooked up to an Arduino microcontroller, which are in turn hooked up to the car's electronic speed controller and servo motor. Speed and steering information is processed all on one microcontroller.

2. Project overview (High level)



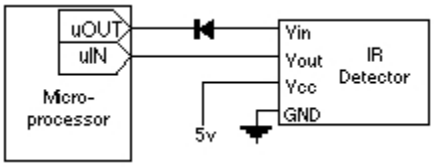
3. Project components

Mechanical engineering

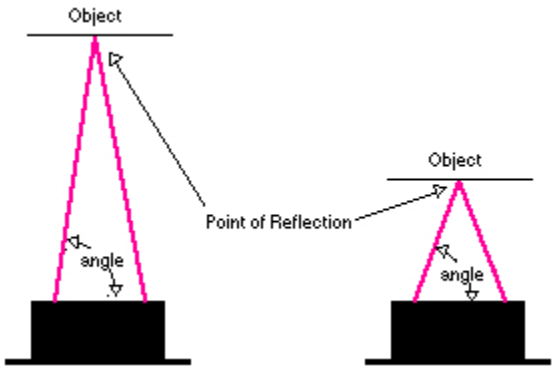
We assembled both RC cars from purchased kits. We had to tune and calibrate the ESC (electronic speed controller) and servo motor for our computer-controlled car. Another important piece of mechanical engineering was the mount for the sensors which proved to be crucial in accurately surveying the field of view to acquire turning information. The mechanical engineering front of this project was light yet time consuming.

Electrical

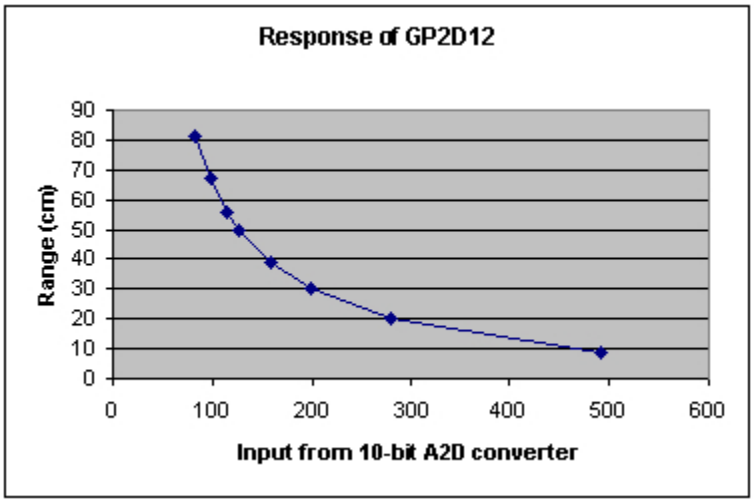
We use an Arduino microcontroller to implement the ACES system. Our board triggers a response from our IR sensors (via a signal wire) every ten microseconds and is echoed an integer value as an input (via input). The values received are processed to determine how the signal that is sent to the ESC and servo will be altered before output via signal wires. The following graphs/pictures are provided by acroname.com for a very similar Sharp IR sensor.

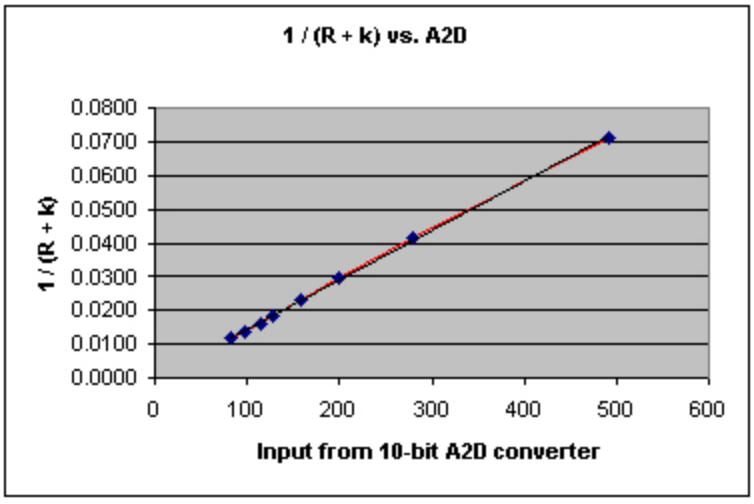


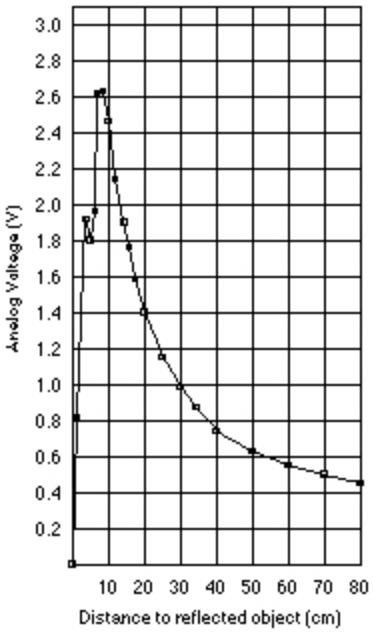
Block Diagram Showing Diode Orientation.



Different Angles with Different Distances





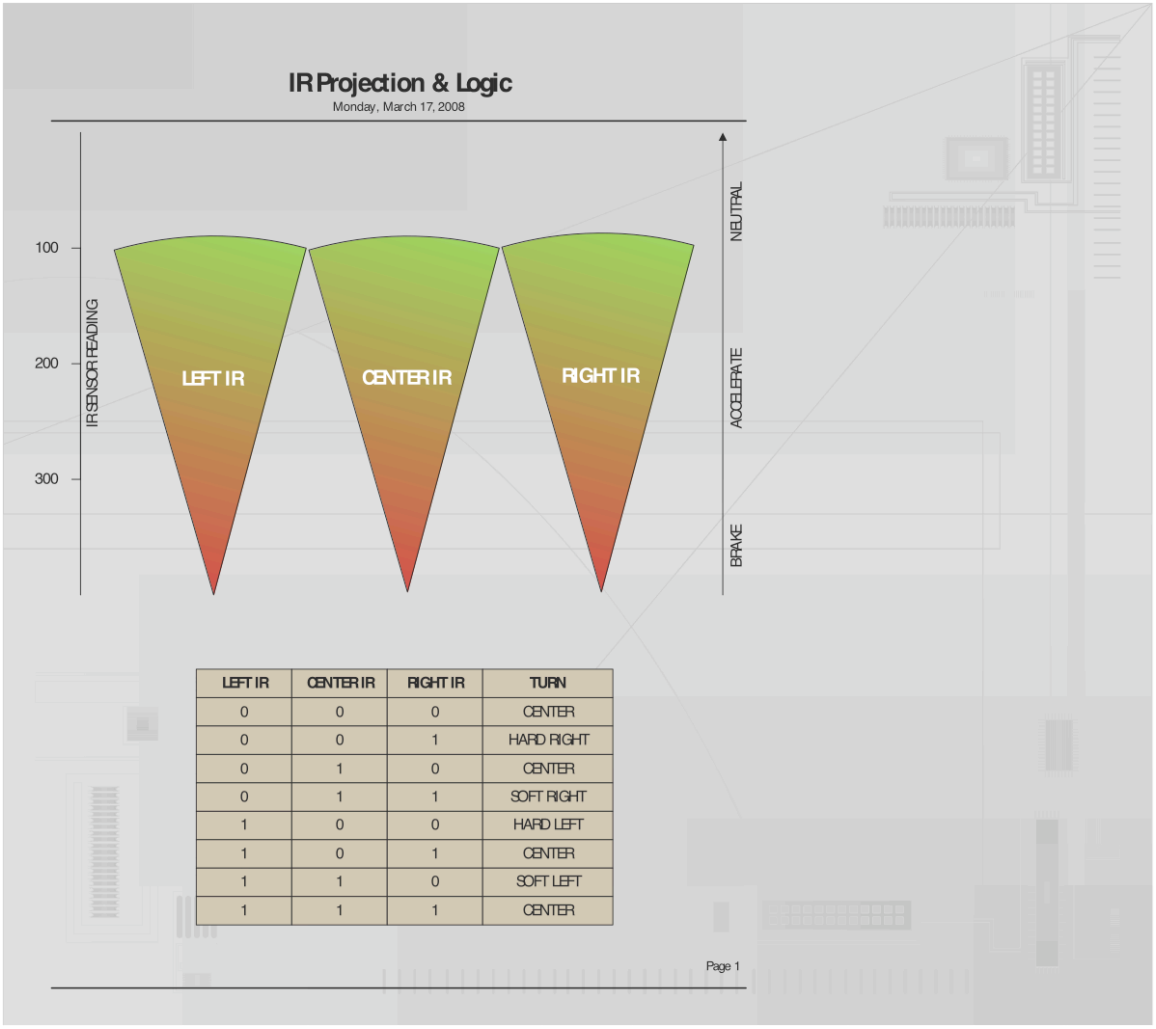


GP2D12 Output Voltage to Distance Curve

The ESC is hooked up to the car's motor and controls its RPMs. The arm of the servo motor turns the wheels left or right or re-centers them, thereby allowing our car to turn in response to the position of the leader in its field of view.

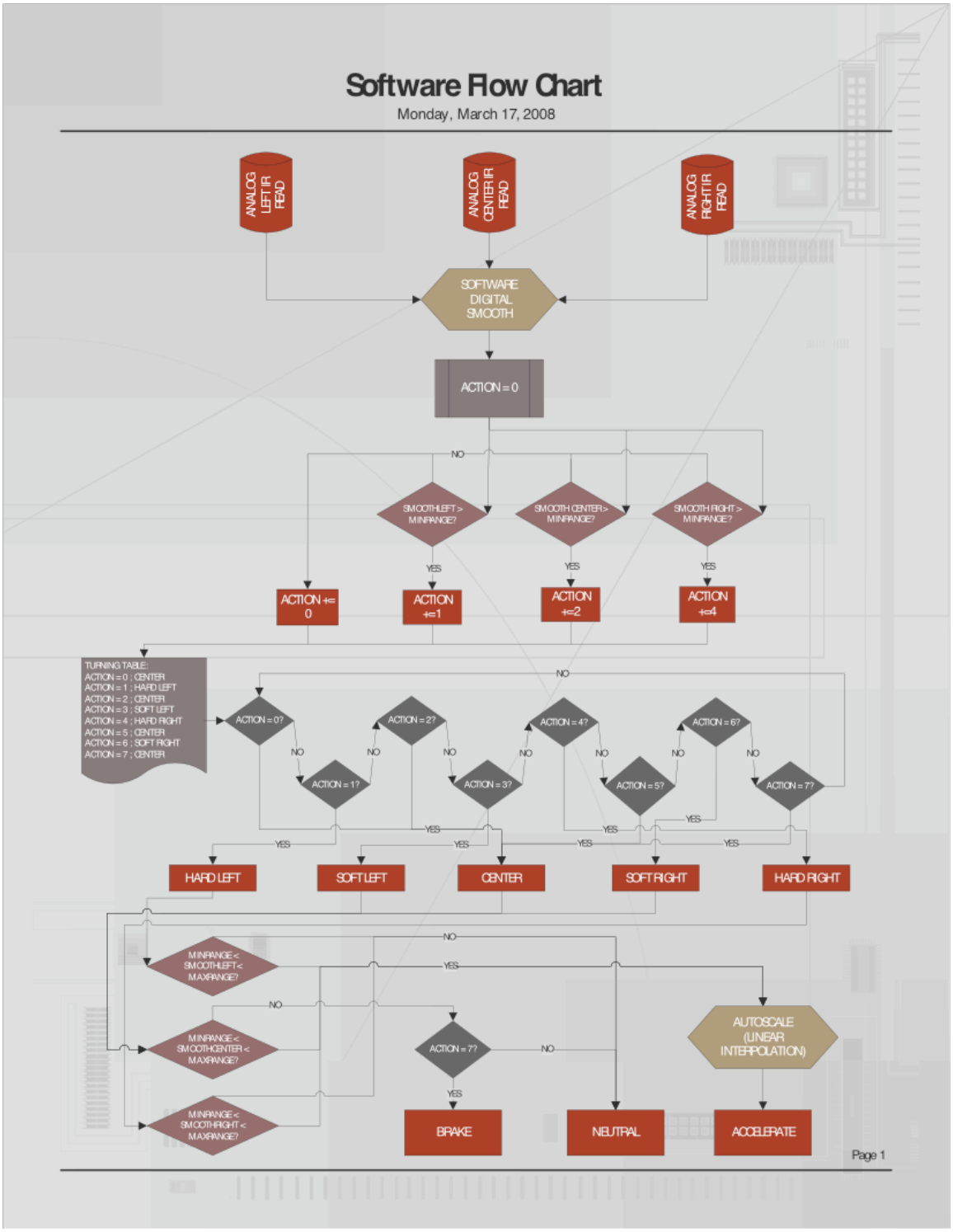
Software

We designed one elegant software component to implement our convoy system. It is programmed on the chip of the Arduino microcontroller which thereby controls the motor and the turning direction or correction.



LEFT IR	CENTER IR	RIGHT IR	TURN
0	0	0	CENTER
0	0	1	HARD RIGHT
0	1	0	CENTER
0	1	1	SOFT RIGHT
1	0	0	HARD LEFT
1	0	1	CENTER
1	1	0	SOFT LEFT
1	1	1	CENTER

Truth Table Showing Turning Decision Based On IR Detection



Software Flow Chart

Source Code

```
/*
* ACES Project (Autonomous Convoy Engagement System)
* Sharp IR, ESC, and Servo Control
*
```

```
This program takes readings from 3 analog Sharp IR sensors and runs them
through a Smoothing Function to even out the values and throw out the erroneous
values. Those values are then used to determine which IR sensor (Left, Center,
or Right) detects an object based on a given threshold. A combination of which
sensor is triggered results in a constant number being added together to
determine the action to be taken (0-7). A table is provided later in the code
to show which action number represents what. A switch clause is then entered to
perform the necessary actions. The speed/acceleration mechanism uses the range
information gathered by the IR sensor and applies linear interpolation so that
those values may fit into a ranged value (1400-1460) in which the ESC
(Electronic Speed Controller) understands (Using Pulse Width Modulation).
Throughout the process, LEDs are triggered accordingly to the IR sensor
quadrant detection and whether the vehicle is in the accelerate/neutral/brake
state. Figuring out the timing delays was a major challenge. The 2 main
functions of Digital Smoothing and Autoscale (Linear Interpolation) are
provided by Paul Badger from http://www.arduino.cc/playground/Main/ .
*
* By Vin Keschrumrus & Brian Buchanan
*
*/

#define filterSamples 13  // filterSamples should  be an odd number, no smaller
than 3

#define minRange  300  // minimum range for IR detection, higher the value the
closer
#define maxRange  110  // maximum range for IR detection, lower the value the
further
#define minAccel  1460 // minimum acceleration range, 1460 is beginning of
accelerations stage
#define maxAccel  1400 // maximum acceleration range, lower than 1400 goes too
fast

int sensSmoothArray1 [filterSamples];   // array for holding raw sensor values
for sensor1
int sensSmoothArray2 [filterSamples];   // array for holding raw sensor values
for sensor2
int sensSmoothArray3 [filterSamples];   // array for holding raw sensor values
for sensor3

int escPin = 6;  // ESC Digital PWM Output Pin
int servoPin = 9;  // Digital Servo Digital PWM Servo Output Pin


int leftLED = 3; // Left IR LED Digital Output Pin
int centerLED = 4;  // Center IR LED Digital Output Pin
int rightLED = 5;  // Right IR LED Digital Output Pin

int accelerateLED = 11;  // Accelerate LED Digital Output Pin
int neutralLED = 12;  // Neutral LED Digital Output Pin
int brakeLED = 13;  // Brake LED Digital Output Pin

int signalLeft = 0;  // Left Sharp IR Analog Input Pin
int signalCenter = 1;  // Center Sharp IR Analog Input Pin
int signalRight = 2;  // Right Sharp IR Analog Input Pin

int tempLeft = 0; // Initial Analog reading from Left Sharp IR sensor
int tempRight = 0;  // Initial Analog reading from Right Sharp IR sensor
int tempCenter = 0;  // Initial Analog reading from Center Sharp IR sensor

int smoothCenter = 0;  // variable for Center IR value after Digital Smoothing
Function
int smoothLeft = 0;  // variable for Left IR value after Digital Smoothing
Function
```

```
int smoothRight = 0;  // variable for Right IR value after Digital Smoothing
Function

int action = 0;  // Variable to determine which action to take
int turnPulse = 0;  // Variable to store Pulse value for turning to be sent to
Servo
int speedPulse = 0;  // Variable to store Pulse value for speed to be send to
ESC

// Variables used for AutoScale Function
int j;
long scaledResult;

void setup() {

  pinMode(servoPin, OUTPUT);
  pinMode(escPin, OUTPUT);

  pinMode(leftLED, OUTPUT);
  pinMode(centerLED, OUTPUT);
  pinMode(rightLED, OUTPUT);

  pinMode(accelerateLED, OUTPUT);
  pinMode(neutralLED, OUTPUT);
  pinMode(brakeLED, OUTPUT);

  pinMode(signalCenter, INPUT);
  pinMode(signalLeft, INPUT);
  pinMode(signalRight, INPUT);

  Serial.begin(115200);
}

// Servo Pulse Function that sends a signal to the Servo
void servoPulse(int pulse)
{
  // range for our setup seems to be from 1150 - 1750 (extreme left to right
looking at servo)
  // with the neutral region being from about 1460~1560
  digitalWrite(servoPin, HIGH);  // set servo high
  delayMicroseconds(pulse);  // wait for time specified to determine turn angle
  digitalWrite(servoPin, LOW);  // set servo low
  delay(2);  // delay for 2 ms
}

// ESC Pulse Function that sends a signal to the ESC
void escPulse(int escPin, long pulse)
{
  digitalWrite(escPin, HIGH);
  delayMicroseconds(pulse);
  digitalWrite(escPin, LOW);
  delay(2);  // delay for 2 ms
}

// AutoScale function that uses Linear Interpolation to smoothly control
acceleration
int autoScale( int originalMin, int originalMax, int newBegin, int newEnd, int
inputValue)
{
  long zeroRefOriginalMax = 0;
  long zeroRefnewEnd = 0;
  long zeroRefCurVal = 0;
  long rangedValue = 0;
  boolean invFlag = 0;
```

```
  // Check for out of range inputValues
  if (inputValue < originalMin) {
    inputValue = originalMin;
  }
  if (inputValue > originalMax) {
    inputValue = originalMax;
  }

  // Zero Refference the values
  zeroRefOriginalMax = originalMax - originalMin;

  if (newEnd > newBegin){
    zeroRefnewEnd = newEnd - newBegin;
  }
  else
  {
    zeroRefnewEnd = newBegin - newEnd;
    invFlag = 1;
  }

  zeroRefCurVal = inputValue - originalMin;

 // Check for originalMin > originalMax  - the math for all other cases i.e.
negative numbers seems to work out fine
  if (originalMin > originalMax ) {
    return 0;
  }

  if (invFlag == 0){
    rangedValue =  ((zeroRefCurVal * zeroRefnewEnd) /
      zeroRefOriginalMax) + newBegin ;
  }
  else     // invert the ranges
  {
    rangedValue =  newBegin - ((zeroRefCurVal * zeroRefnewEnd) /
      zeroRefOriginalMax)  ;
  }

  return rangedValue;
}


// Digital Smooth Function that smooths out the values of the Sharp IR analog
readings from spikes
int digitalSmooth(int rawIn, int *sensSmoothArray)
{
  int j, k, temp, top, bottom;
  long total;
  static int i;
  static int sorted[filterSamples];
  boolean done;

  i = (i + 1) % filterSamples;    // increment counter and roll over if necc. -
% (modulo operator) rolls over variable
  sensSmoothArray[i] = rawIn;                 // input new data into the oldest
slot

  for (j=0; j<filterSamples; j++){     // transfer data array into anther array
for sorting and averaging
    sorted[j] = sensSmoothArray[j];
  }

  done = 0;                // flag to know when we're done sorting
```

```
  while(done != 1){        // simple swap sort, sorts numbers from lowest to
highest
    done = 1;
    for (j = 0; j < (filterSamples - 1); j++){
      if (sorted[j] > sorted[j + 1]){     // numbers are out of order - swap
        temp = sorted[j + 1];
        sorted [j+1] =  sorted[j] ;
        sorted [j] = temp;
        done = 0;
      }
    }
  }

  // throw out top and bottom 15% of samples - limit to throw out at least one
from top and bottom
  bottom = max(((filterSamples * 15)  / 100), 1);
  top = min((((filterSamples * 85) / 100) + 1  ), (filterSamples - 1));   //
the + 1 is to make up for asymmetry caused by integer rounding
  k = 0;
  total = 0;
  for ( j = bottom; j< top; j++){
    total += sorted[j];  // total remaining indices
    k++;
  }

  return total / k;    // divide by number of samples
}

// Main loop of program
void loop() {

  // Reset all IR readings and action
  action = 0;
  tempLeft = 0;
  tempRight = 0;
  tempCenter = 0;

  // Get readings from each IR sensor and smooth out the values
  tempLeft = analogRead(signalLeft);
  smoothLeft = digitalSmooth(tempLeft, sensSmoothArray1);

  tempCenter = analogRead(signalCenter);
  smoothCenter = digitalSmooth(tempCenter, sensSmoothArray2);

  tempRight = analogRead(signalRight);
  smoothRight = digitalSmooth(tempRight, sensSmoothArray3);

  // Set all LED low
  digitalWrite(leftLED,LOW);
  digitalWrite(centerLED,LOW);
  digitalWrite(rightLED,LOW);

  digitalWrite(accelerateLED,LOW);
  digitalWrite(neutralLED,LOW);
  digitalWrite(brakeLED,LOW);

  // Left IR Sensor detection
  if (smoothLeft > maxRange)
  {
     action = action + 1;
     digitalWrite(leftLED,HIGH);
  }

  // Center IR Sensor detection
```

```
  if (smoothCenter > maxRange)
  {
     action = action + 2;
     digitalWrite(centerLED,HIGH);
  }

  // Rigth IR Sensor detection
  if (smoothRight > maxRange)
  {
     action = action + 4 ;
     digitalWrite(rightLED,HIGH);
  }
  // ACTION TABLE:
  // action = 0 ; center // L C R IR out of range
  // action = 1 ; hard left // L IR only
  // action = 2 ; center // C IR only
  // action = 3 ; soft left // L C IR
  // action = 4 ; hard right // R IR only
  // action = 5 ; center // L R IR
  // action = 6 ; soft right // C R IR
  // action = 7 ; center // L C R IR

  // Switch clause to perform appropriate instructions for each action
  switch(action)
  {
    case 1: // Hard Left turn and use Left IR for range information
     turnPulse = 1700;
      if ((smoothLeft > maxRange) & (smoothLeft < minRange))
      {
          speedPulse = autoScale(maxRange, minRange, maxAccel, minAccel,
smoothLeft);
      }
      else
      {
        speedPulse = 1500; // netural speed
      }
     break;

    case 2:  // Center turn and use Center IR for range information
      turnPulse = 1500;
      if ((smoothCenter > maxRange) & (smoothCenter < minRange))
      {
          speedPulse = autoScale(maxRange, minRange, maxAccel, minAccel,
smoothCenter);
      }
      else
      {
        speedPulse = 1500;
      }
      break;

    case 3:  // Soft Left turn and use Center IR for range information
      turnPulse = 1560;
      if ((smoothCenter > maxRange) & (smoothCenter < minRange))
      {
          speedPulse = autoScale(maxRange, minRange, maxAccel, minAccel,
smoothCenter);
      }
      else
      {
        speedPulse = 1500;
      }
      break;
```

```
    case 4:  // Hard Right turn and use Right IR for range information
     turnPulse = 1275;
      if ((smoothRight > maxRange) & (smoothRight < minRange))
      {
          speedPulse = autoScale(maxRange, minRange, maxAccel, minAccel,
smoothRight);
      }
      else
      {
        speedPulse = 1500;
      }
     break;


    case 6:  // Soft Right turn and use Center IR for range information
     turnPulse = 1425;
      if ((smoothCenter > maxRange) & (smoothCenter < minRange))
      {
          speedPulse = autoScale(maxRange, minRange, maxAccel, minAccel,
smoothCenter);
      }
      else
      {
        speedPulse = 1500;
      }
     break;

    case 7:  // Center turn and use Center IR for range information
      turnPulse = 1500;
      if ((smoothCenter > maxRange) & (smoothCenter < minRange))
      {
          speedPulse = autoScale(maxRange, minRange, maxAccel, minAccel,
smoothCenter);
      }
      else
      {
        speedPulse = 1800; // All 3 IR sensors detected and object is too
close, therefore brake
      }
      break;

    default:  // Cases 0 and 5 fall under this category, want to set to Center
turn and set speed to neutral
      turnPulse = 1500;
      speedPulse = 1500;
  }

  servoPulse(turnPulse); // send turn pulse
  escPulse(escPin, speedPulse); // send speed pulse

  // Case Statements to set LED accelerate/brake/neutral
  if ((speedPulse > 1460) & (speedPulse < 1560))
  {
      digitalWrite(neutralLED,HIGH);
  }
  else
  {
    if (speedPulse > 1560)
    {
      digitalWrite(brakeLED,HIGH);

    }
    else
    {
```

```
      digitalWrite(accelerateLED,HIGH);
    }
  }
}
```

4. Design description

The program takes readings from 3 analog Sharp IR sensors and runs them through a Smoothing Function to even out the values and throw out the erroneous values. Those values are then used to determine which IR sensor (Left, Center, or Right) detects an object based on a given

threshold. A combination of which sensor is triggered results in a constant number being added together to determine the action to be taken (0-7). A table is provided later in the code to show which action number represents what. A switch clause is then entered to perform the necessary actions. The speed/acceleration mechanism uses the range information gathered by the IR sensor and applies linear interpolation so that those values may fit into a ranged value (1400-1460) in which the ESC (Electronic Speed Controller) understands (Using Pulse Width Modulation). Throughout the process, LEDs are triggered accordingly to the IR sensor quadrant detection and whether the vehicle is in the accelerate/neutral/brake state. Figuring out the timing delays was a major challenge. The 2 main functions of Digital Smoothing and Autoscale (Linear Interpolation) are provided by Paul Badger from http://www.arduino.cc/playground/Main/ .

5. System test plan

Test Number	Description of Set-up	Input or Stimulus	Expected Behavior

1	Kill switch test. If ultrasonic sensor detects that the lead car is out of range, the motor is killed so that the ACES car does not go out of control.	Accelerating the lead car or taking it out of range manually.	ACES car is killed/stopped.
2	Ability to follow an object in a straight line. Demonstrates the functionality of the ESC (Electronic Speed Control) and the ultrasonic sensor.	Start the ACES car from rest. Then move the lead object away at a reasonable rate. Also have the lead object be moving at a certain rate and have it come to a rest.	Should be able to accelerate and decelerate accordingly based on the range of the lead object. Should not lose track if within the specified safe zone.
3	Steering test. Be able to control the servo properly based on the IRPD sensor quadrant detection. The IRPD will sense the quadrant in which the lead object has entered and should turn accordingly until the IRPD is in the center quadrant.	Have a lead object move from left to right at a pace to see if the servo will turn according based on the quadrant it is detected in.	Servo should turn accordingly and reset to the center position if the IRPD detects the lead object in it's center quadrant.

4	Combine the speed control with the turning ability. Make sure the two work in tandem for reasonable velocities and turning angles.	Have a lead object be moving at a constant rate and turn the object left and right at small angles. Even take the lead object out of range of the ACES car IRPD range to see if it will continue turning accordingly	The ACES car should be able to keep up with the lead object and turning accordingly even when the IRPD does not detect any object in any quadrants (ex. 2 consecutive sharp turns). The car should not kill itself unless it can't re-center within 2 seconds.
5	Safe braking Test. Make sure that the ACES car is able to stop in time if the lead object comes to a near dead stop. This is important so that a collision does not occur.	Have the lead car travel at a constant velocity then come to an immediate halt. The ACES car should detect the sudden change in range and apply full brakes.	The ACES car should not collide with the lead object and come to a halt within a safe distance.
6	Mount Test. Make sure that every piece of equipment works properly when mounted in the configuration for presentation. This is important because many calculations, formulas may need to change based on the positions of the sensors.	Follow test cases 1-5 when everything is mounted in the presentation configuration.	Everything is to behave as expected and the sensing capabilities should remain as precise.

7	Self-Correcting code. Should include code that accommodates for unexpected but typical events. An example is for the ACES car's ability to know the direction to turn the servo if vision is lost from the lead object. Another example is the ACES car ability to try a last chance operation that attempts to get vision back of the lead car; otherwise a kill operation will be executed.	Basically, we will have the lead car make sharp turns and accelerated/ decelerate at steep rates.	The ACES car should ideally be able to keep up at all times and activate the kill switch properly in dangerous situations.
8	Constant Speed test. Make sure that the ACES car is fully able to maintain a constant speed with the lead car within a safe zone range.	Have the lead car accelerate and maintain a constant velocity.	The ACES car must be able to keep up with the lead car without skipping a beat.

6. Project timeline

10/1: Ordered Arduino Boards, wires, LEDs and other starter materials

12/4: Project chosen

1/17: Purchased the follower RC kit

1/18: Ordered Sonic Range-Finder

1/22: Assembled RC car and calibrated speed and steering

1/24: Wired and tested SRF

1/24: Ordered 3-Quadrant IR sensor

1/29: Programmed speed Arduino to control the wheel speed based on SRF readings

2/5: Wired and successfully tested IR sensor

2/21: Temporarily mounted SRF and achieved our minimum subset

2/26: Temporarily mounted IRPD sensor and concluded that we should change implementation

2/26: Purchased 3 new IR sensors that detect range

3/6: Achieved medium subset

3/8: Completed the ACES project

7. Division of work

Vin Keschrumrus:

- Programmed following algorithm
- Involved in debugging and testing the following algorithm
- Involved in debugging and testing the turning algorithm
- Helped wire microcontroller and components
- Helped with mounting of sensors and microcontroller
- Made Charts/Slides and prepare Final Project Documentation
- Help make Demo Video

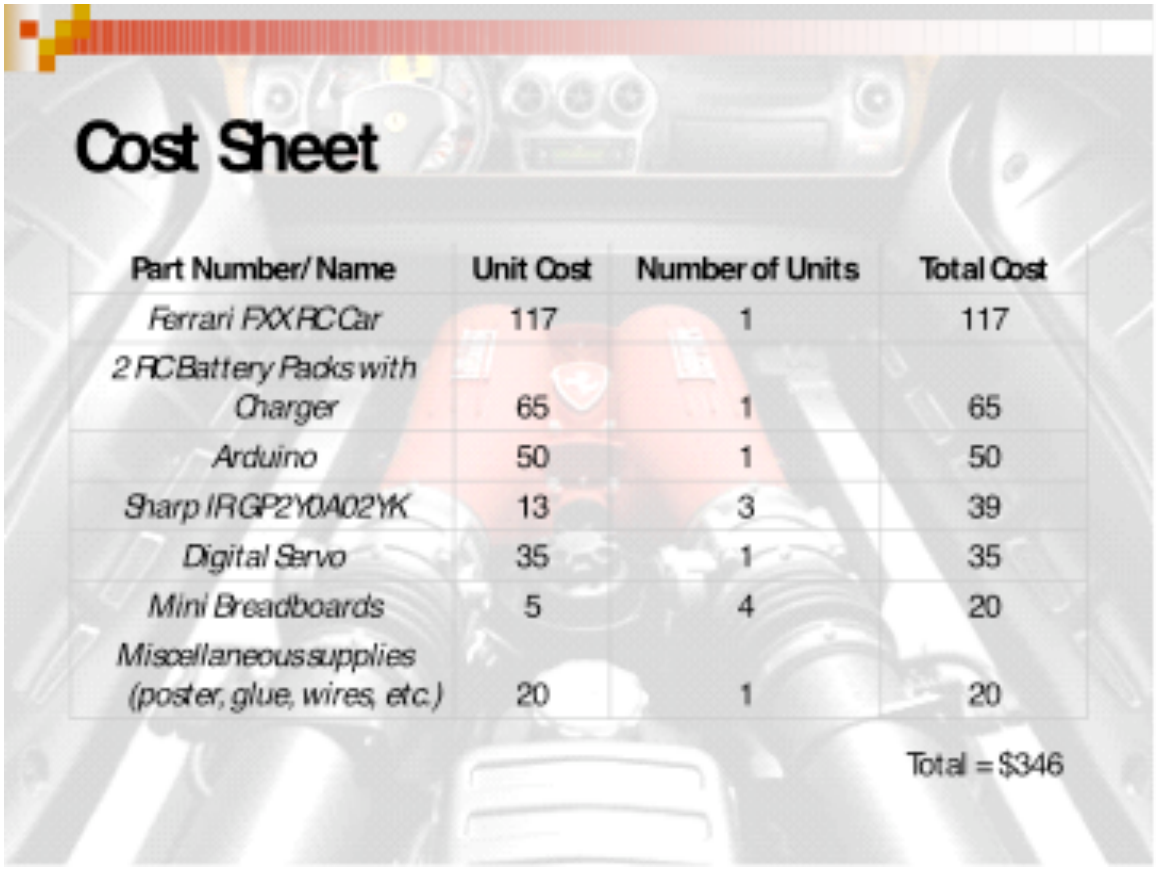
Brian Buchanan:

- Programmed following algorithm
- Involved in debugging and testing the following algorithm
- Involved in debugging and testing the turning algorithm
- Helped wire microcontroller and components
- Helped with mounting of sensors and microcontroller
- Help make Slides and prepare Final Project Documentation
- Help make Demo Video

Robert Snow:

- Helped with mounting of sensors and microcontroller
- Help Make and Edit Demo Video
- Helped with Hardware aspects of project

8. Cost



Total cost of project: $346

9. Problem encountered and comments

We encountered several problems during the more than 200 hours of work. The first and most significant problem occurred during the early stages of design. We simply purchased parts that were not going to allow us to successfully achieve our medium subset. We originally were implementing ACES with an ultrasonic range-finder module to control the speed and a proximity infrared detector module to control the turning. Our three-quadrant proximity IR sensor only detected the presence of an object or an absence of an object and was not very accurate beyond about 20 inches. In early testing, when the leader took a turn while moving, the IR would sense the turn, but the SRF would lose track of the car and shut down processing of speed information. This was unacceptable, and we decided to change our implementation.

We purchased three Sharp IR sensors that returned range information also, which proved to be the solution to our ACES turning problems. These IR sensors were far more accurate and robust. We found that the IR sensors were reading erroneous value spikes that caused the system to make the car twitch. We solved this problem with the digital smoothing function and by adding a capacitor to the circuit by the signal wires. We also

had a software problem with trying to smooth out the acceleration based on the distance information from the IRs. We tried numerous things but eventually came across the idea and function of Linear Interpolation which auto scales a range of values to another range of values. This helped tremendously with our acceleration region.

Another problem we encountered was the mounting of the IR sensors due to their precision and narrow beam angle. At first we used cardboard and tape to mount the sensors. Later we decided to construct a sound and sturdy mount for the sensors. We used high quality thick plastics, screws, and plastic bracket mounts to construct the housing/ mount for the 3 IR sensors. We were then able to precisely angle the side IRs for maximum performance in respects to turning capabilities.

Overall, this was a very enjoyable project and we learned many new things. This is probably one of the best classes because we were able to utilize some of our previous knowledge and apply some engineering problem solving skills when we encountered an obstacle. We plan to continue this project further in the future to improve it's performance (smoother acceleration and more robust turning capabilities) and functionality (obstacle avoidance techniques, one-object locking/calibration techniques, formation capability with multiple ACES equipped vehicles) for our own learning purposes. We would also like to thank Professor Klefstad and Shruti Gorappa of UC Irvine for providing guidance and assistance.