AUTORIDE
FINAL REPORT

By

ANGEL NUNEZ

IMDL Spring 2008
Dr. Arroyo
Dr. Schwartz

Autoride is a simple line following robot that is also integrated with sonar technology in order to avoid obstacles. The sonar sensors allow the robot to move freely through a system of obstacles without the help of the line tracking sensor. The machine is also equipped with a Bluetooth receiver that allows me to exchange information with a computer. In this particular case I am using this technique to pair webcam information with the robots board. All of this technology is hidden underneath a very good looking 2005 mustang body. Enjoy!!!

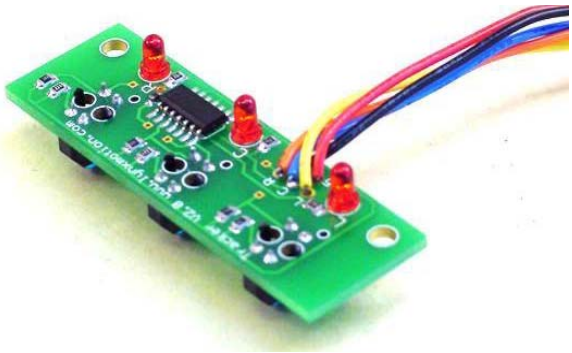
SENSORS

BLUETOOTH RECEIVER

This sensor from sparks.com is perfect for interfacing a webcam with the MAVRIC IIB. A computer gathers information from the webcam and send a wireless signal to the bluetooth receiver. The webcam is connected to matlab and the moment a light goes on the BW image, matlab outputs 1 instead of 0.

The board is then programmed to act in different ways according to the information received by the bluetooth.

THE TRACKER



The Tracker is an Infrared reflective sensor trio, which can be attached to the front of mobile wheeled robots. The sensor delivers three digital signals to a microcontroller so the robot

can reliably follow a black line on a white background, or vise versa. The circuitry could also be useful as an edge detector to keep your robot from going off the edge of a table, or tumbling down the stairs. This is a time tested rock solid circuit that will perform flawlessly with no tedious set-up or calibration.

The three reflective sensors are made from one piece Infrared LED and photo detector pairs that are directed at the surface in front of the robot. Each of the three sensors looks for reflected IR light. When one of the sensors is positioned over a dark or black surface its output will be low. When it is moved to a light or white surface its output will be high. The microcontroller monitors these signals and moves the robot according to the diagram below.

The Tracker works great with line thickness from 1/4" to 3/4". Lines made with electrical tape are perfect. The track can be black tape on a white background or white tape on a black background, as long as the dark surface has a matte (not shiny) finish. A track can be made from plastic such as Sintra, cut into 12" x 12" pieces. You then add the lines to each square making some turn and some go straight. The placement of the panels can be changed to make different courses. For more challenging courses some of the panels can have gaps or wiggles in the line.

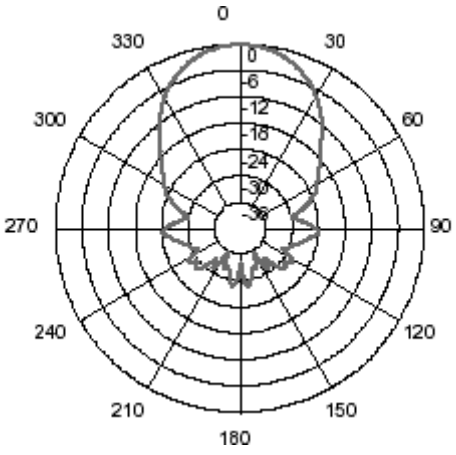
One of the things that I like best about the tracker, besides that it comes already assemble with pins and wires, is that it is very easy to program. For programming illiterate people, such as me, this can be very handy and can save a lot of time. The tracker with the help of the MAVRIC-IIB board can be interfaced with servos very easily. In Autorides case the tracker is being interfaced with one servo that is in charge of the steering and the LCD screen as well. This sensor is a little pricy compared with other companies' products but at the end it's totally worth the money.

SONAR

This project is being equipped with sonar sensor. The sonar where purchased from acroname.com and their model number is Devantech SRF05. Sonar is being used for obstacle avoidance, this specific device works perfect for our application since it has a range of about 4m and I will only be needing it to avoid objects a merely 50cm or even less. The full capacity of the sensor is not being challenged which I think is a good thing, nobody likes unexpected malfunctions.

Specifications	
Frequency	40kHz
Max Range	4 meters
Min Range	3 centimeters
Input Trigger	10uSec minimum, TTL level pulse
Echo Pulse	Positive TTL level signal, proportional to range

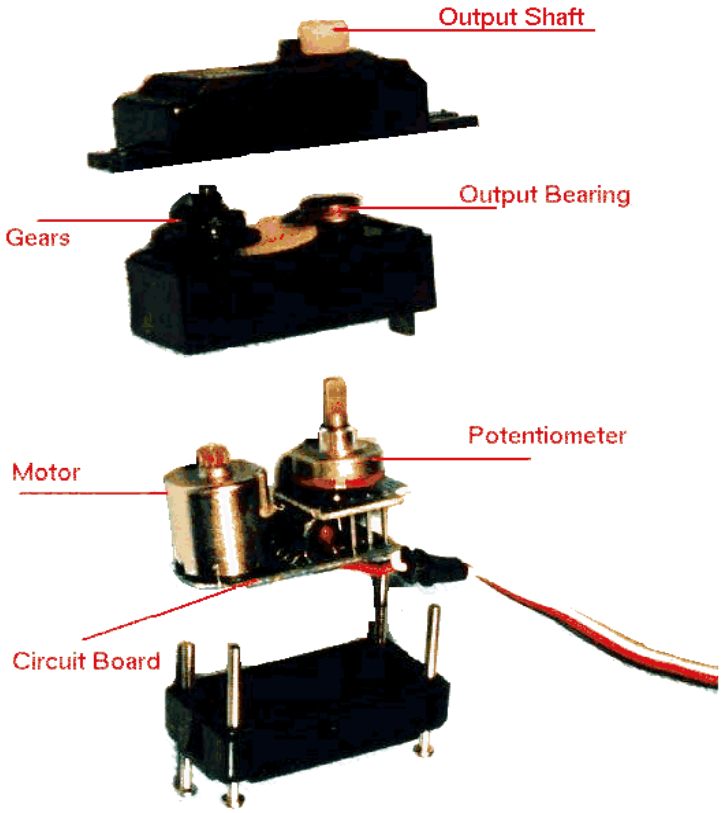
Autoride will be equipped with two sonar sensors in the front; the reason for using two is that the functioning angle of the sensor is limited, as shown in the figure bellow. The sonar sensors where not as handy as the tracking sensors from Lynxmotion, they came bare and with little instructions. I had to solder the pins on and make new cables for them, but none the less it was a good deal.



DRIVE TRAIN

The robot will be powered by a single DC motor that will be connected through a driveshaft to a differential that will then distribute torque to both wheels evenly. The motor is manufactured by Jameco with a maximum torque of 350g-cm and a speed of 600 rpm.

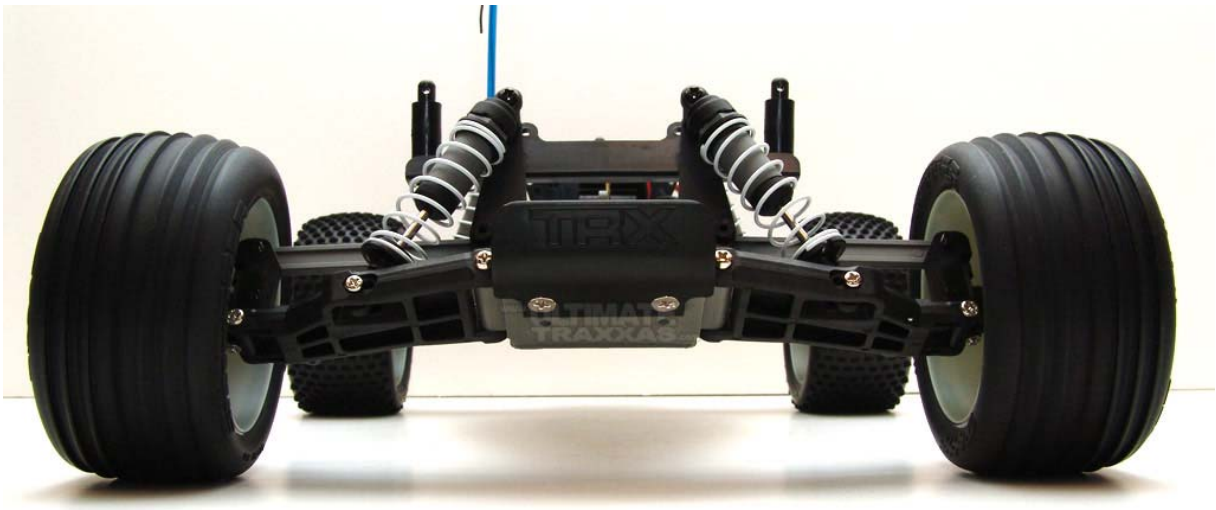
The car will be steered by a manual steering system powered by a servo. This will allow smooth steering and prevent any stalls that can be caused by other types of steering. The servo will most probably be obtained from an economic remote control car since a more expensive servo is not required for this application.



Together this system, combined with the MAVRIC IIB board and some computer programming, will power the robot through the task that it is designed to accomplish.

PLATFORM

Since the design of the robot is of a normal car, the platform is going to be obtained from a TRAXXAS Nitro RC car. The platform is a flat CNC design made out of aluminum alloy which makes it just as light weight as plywood but much stronger. Attached to this is going to be a all four wheel independent suspension equipped with gas shock that will reduce the shock on the robot and thus also reduce the risk of circuit failure due to vibration.



The whole robot will be disguised by a body of a car, most likely one to resemble a muscle car which is my personal preference in collection.

CODE

```
#include <avr/interrupt.h>

#define DIR_L PORTE2
#define DIR_R PORTE5
#define MOTOR_L OCR3A          //pin3 port E
#define MOTOR_R OCR3B          //pin4 port E


void init_timer();
void motor_move();
void dc_move();

// initialize PWM timer

#include <avr/io.h>

int stop = 0;

#define S1R (1<<0) // Echo output 1
#define S1 (1<<1) // Trigger input 1
#define S2R (1<<6) // Echo output 2
#define S2 (1<<7) // Trigger input 2
#define TRAILS 10
// PORTD , PIND == PORTA, PINA
#define MIN(x,y) ((x) < (y)) ? (x) :(y)
#define MAX(x,y) ((x) > (y)) ? (x) :(y)

void lcd_delay();          // short delay (50000 clocks)
void lcd_init();           // sets lcd in 4 bit mode, 2-line mode, with cursor on and
set to blink
void lcd_cmd();                  // use to send commands to lcd
void lcd_disp();           // use to display text on lcd
void lcd_clear();          // use to clear LCD and return cursor to home position
void lcd_row(int row);     // use to put the LCD at the desired row

int getSonar1();           // use to read value from sonar module
int getSonar2();

/* IMPORTDNT!
```

Before using this code make sure your LCD is wired up the same way mine is, or change the code to
match the wiring of your own LCD. My LCD is connected to PortC of the At-Mega128 in the following manner:

PortC bit 7 : LCD data bit 7 (MSB)
PortC bit 6 : LCD data bit 6
PortC bit 5 : LCD data bit 5
PortC bit 4 : LCD data bit 4 (LSB)

PortC bit 3 : (not connected)
PortC bit 2 : LCD enable pin (clock)
PortC bit 1 : LCD R/W (read / write) signal
PortC bit 0 : LCD RS (register select) pin

Also remember you must connect a potentiometer (variable resistor) to the vcc, gnd, and contrast pins on the LCD.
The output of the pot (middle pin) should be connected to the contrast pin. The other two can be on either pin.

***/**

```
void port_init() {
      DDRC = 0xFF;                  // set portC to output  (could also use DDRC =
0b11111111)
      DDRB = 0xFF;                  // set portB to output
      DDRE = 0xFF;                  // set portE to output
      DDRF = 0b00000000;              // set port F to all input
}

void interrupt_init() {
      UCSR0B = 0b10011000;              // enable tx and rx
      UCSR0C = 0b00000110;              // asynch, no parity, 1 stop bit, 8 bit characters
      UBRR0L = 95;                      // initialize transfer speed for 38400 baud
      sei();                          // enable all interupts
}

void init_timer()
{
    //initializes motors, enable OC3A, B, and C

    DDRE = 0xFF         ;       //PORT E out
    //DDR_OC3A = 0b1;
    //DDR_OC3B = 0b1;
```

```
    TCCR3A = 0xA8;     //10101000 clear OCA,B,C on campare match- non-inv PWM
    TCCR3B = 0x12;     //00010010 mode  8 - PWM phase/freq correct, clk(io)/8
    ICR3 = 18432;      //PWM = 50hz.  1/(14.7456 MHz/8)*2*18432 = 1/50
    TCNT3 = 0x00;      //

    MOTOR_L = 0;       //not moving
    MOTOR_R = 0;

    PORTE = 0x0;


}



void motor_move(uint16_t speed)
{
    int increment_R = 0;


    if(MOTOR_R > speed)
    {
        increment_R = -1;
    }
    else if (MOTOR_R < speed)
    {
        increment_R = 1;
    }
    else
    {
        increment_R = 0;
    }

    while (MOTOR_R != speed) // if either motor does not equal speed, run this loop
    {
        if (MOTOR_R != speed)
        {
            MOTOR_R =MOTOR_R + increment_R;
        }


    }
}
```

```
void dc_move(uint16_t speed)
{

    int increment_L = 0;

    if(stop)
            return;

    if(MOTOR_L > speed)
    {
            increment_L = -1;
    }
    else if (MOTOR_L < speed)
    {
            increment_L = 1;
    }
    else
    {
            increment_L = 0;
    }



    while (MOTOR_L != speed) // if either motor does not equal speed, run this loop
    {
            if (MOTOR_L != speed)
            {
                    MOTOR_L = MOTOR_L + increment_L;
            }


    }
}


void lcd_delay()            // delay for 10000 clock cycles
{
    long int ms_count = 0;
    while (ms_count < 1000)
    {
            ms_count = ms_count + 1;
    }
}

void lcd_cmd( unsigned int myData )
{
```

```
    /* READ THIS!!!

    The & and | functions are the BITWISE AND and BITWISE OR functions respectively.  DO NOT
    confuse these with the && and || functions (which are the LOGICAL AND and LOGICAL OR functions).

    The logical functions will only return a single 1 or 0 value, thus they do not work in this scenario
    since we need the 8-bit value passed to this function to be preserved as 8-bits
    */


    unsigned int temp_data = 0;

    temp_data = ( myData | 0b00000100 );      // these two lines leave the upper nibble as-is, and set
    temp_data = ( temp_data & 0b11110100 );          // the appropriate control bits in the lower nibble
    PORTC = temp_data;
    lcd_delay();
    PORTC = (temp_data & 0b11110000);                // we have written upper nibble to the LCD

    temp_data = ( myData << 4 );                     // here, we reload myData into our temp. variable and shift the bits
                                                                                    // to the left 4 times.  This puts the lower nibble into the upper 4 bits

    temp_data = (temp_data & 0b11110100);  // temp_data now contains the original
    temp_data = (temp_data | 0b00000100);    // lower nibble plus high clock signal

    PORTC = temp_data;                                                  // write the data to PortC
    lcd_delay();
    PORTC = (temp_data & 0b11110000);                // re-write the data to PortC with the clock signal low (thus creating the falling edge)
    lcd_delay();

}

void lcd_disp(unsigned int disp)
{

    /*
```

```
    This function is identical to the lcd_cmd function with only one exception.  This least
significant bit of
    PortC is forced high so the LCD interprets the values written to is as data instead of
a command.

    */

    unsigned int temp_data = 0;

    temp_data = ( disp & 0b11110000 );
    temp_data = ( temp_data | 0b00000101 );
    PORTC = temp_data;
    lcd_delay();
    PORTC = (temp_data & 0b11110001);
    lcd_delay();                                        // upper nibble

    temp_data = (disp << 4 );
    temp_data = ( temp_data & 0b11110000 );
    temp_data = ( temp_data | 0b00000101 );
    PORTC = temp_data;
    lcd_delay();
    PORTC = (temp_data & 0b11110001);
    lcd_delay();                                        // lower nibble

}


void lcd_init()
{
    lcd_cmd(0x33);              // writing 0x33 followed by
    lcd_cmd(0x32);              // 0x32 puts the LCD in 4-bit mode

    lcd_cmd(0x28);              // writing 0x28 puts the LCD in 2-line mode

    lcd_cmd(0x0F);              // writing 0x0F turns the display on, curson on, and puts
the cursor in blink mode

    lcd_cmd(0x01);              // writing 0x01 clears the LCD and sets the cursor to the
home (top left) position

    //LCD is on... ready to write

}

void lcd_string(char *a)
```

```
{

    /*

    This function writes a string to the LCD.  LCDs can only print one character at a time so we need to
    print each letter or number in the string one at a time.  This is accomplished by creating a pointer to
    the beginning of the string (which logically points to the first character).  It is imPORTDnt to understand
    that all strings in C end with the "null" character which is interpreted by the language as a 0.  So to print
    an entire string to the LCD we point to the beginning of the string, print the first letter, then we increment
    the pointer (thus making it point to the second letter), print that letter, and keep incrementing until we reach
    the "null" character".  This can all be easily done by using a while loop that continuously prints a letter and
    increments the pointer as long as a 0 is not what the pointer points to.

    */

    while (*a != 0)
    {
            lcd_disp((unsigned int) *a); // display the character that our pointer (a) is pointing to
            a++;                                                    // increment a
    }
    return;

}

void lcd_int(int value)
{

    /*
    This routine will take an integer and display it in the proper order on
    your LCD.  Thanks to Josh Hartman (IMDL Spring 2007) for writing this in lab
    */
    int temp_val;
    int x = 10000;                  // since integers only go up to 32768, we only need to worry about
                                                    // numbers containing at most a ten-thousands place
```

```
    while (value / x == 0)// the purpose of this loop is to find out the largest position (in
decimal)
    {                                       // that our integer contains.  As soon as we
get a non-zero value, we know
            x/=10;                  // how many positions there are int the int and x
will be properly initialized to the largest
    }                                       // power of 10 that will return a non-zero
value when our integer is divided by x.


    while (x >= 1)                  // this loop is where the printing to the LCD
takes place.  First, we divide
    {                                                   // our integer by x (properly
initialized by the last loop) and store it in
            temp_val = value / x;   // a temporary variable so our original value is
preserved.  Next we subtract the
            value -= temp_val * x;          // temp. variable times x from our original
value.  This will "pull" off the most
            lcd_disp(temp_val+ 0x30);  // significant digit from our original integer but
leave all the remaining digits alone.
                                                    // After this, we add a hex
30 to our temp. variable because ASCII values for integers
            x /= 10;                                // 0 through 9 correspond to hex
numbers 30 through 39.  We then send this value to the
    }                                               // LCD (which understands
ASCII).  Finally, we divide x by 10 and repeat the process
                                                    // until we get a zero value
(note: since our value is an integer, any decimal value
    return;                                         // less than 1 will be
truncated to a 0)

}

void lcd_clear()        // this function clears the LCD and sets the cursor to the home
(upper left) position
{
    lcd_cmd(0x01);

    return;
}

void lcd_row(int row)   // this function moves the cursor to the beginning of the
specified row without changing
{                                       // any of the current text on the LCD.

    switch(row)
```

```
    {

            case 0: lcd_cmd(0x02);
            case 1: lcd_cmd(0xC0);

    }

    return;
}

void sonar_init()
{
    DDRA = S1|S2;
}

int getSonar1()
{

    unsigned long total = 0;
    unsigned int n,i;
    unsigned int min,max=0;

    min = 0xffff;

    for (i = 0;i < TRAILS; i++)
    {
            n = 0;
            PORTA =  (PINA & (~S1)); //(PIND & 0x7F);      // these two lines create a
rising edge
            PORTA =  (PINA | S1); //(PIND | 0x80);    // on PORTD pin 7 (2)


            while (n < 40)
            {
                    //waste enough clock cycles for at least 10us to pass
                    n += 1;
                    n++;
            //      lcd_clear();
            //      lcd_int(n);
            }

            PORTA =  (PINA & (~S1));//          PORTD = (PIND & 0x7F);  // force
PORTD pin 7(2) low to create a falling edge
                                                // this sends out the trigger

            while (!(PINA & S1R))
```

```
            {
                    // do nothing as long as echo line is low
            }

            n = 0;  //re-use our dummy variable for counting

            while (PINA & S1R)
            {
                    n += 1;                 // add 1 to n as long as PORTD pin 0 is high
            }

            //when we get here, the falling edge has occured
            total +=n;
            min = MIN (n,min);
            max = MAX (n,max);
    }
    total-= (min + max);
    n = (int) (total/TRAILS - 2);

    return n;

}

int getSonar2()
{

    unsigned long total = 0;
    unsigned int n,i;
    unsigned int min,max=0;

    min = 0xffff;

    for (i = 0;i < TRAILS; i++)
    {
            n = 0;
            PORTA =  (PINA & (~S2)); //(PIND & 0x7F);      // these two lines create a
rising edge
            PORTA =  (PINA | S2); //(PIND | 0x80);    // on PORTD pin 7 (2)



            while (n < 40)
            {
                    //waste enough clock cycles for at least 10us to pass
                    n += 1;
                    n++;
```

```
            //      lcd_clear();
            //      lcd_int(n);
            }

            PORTA =  (PINA & (~S2));//       PORTD = (PIND & 0x7F);  // force
PORTD pin 7(2) low to create a falling edge
                                            // this sends out the trigger

            while (!(PINA & S2R))
            {
                  // do nothing as long as echo line is low
            }

            n = 0;  //re-use our dummy variable for counting

            while (PINA & S2R)
            {
                  n += 1;                   // add 1 to n as long as PORTD pin 0 is high
            }

            //when we get here, the falling edge has occured
            total +=n;
            min = MIN (n,min);
            max = MAX (n,max);
      }
      total-= (min + max);
      n = (int) (total/TRAILS - 2);

      return n;

}

int main(void)
{

      int i; long SonarVal1 = 0, SonarVal2 = 0;

      sonar_init(); //0x80;  // sonar is connected with trigger on PORTD.7 and echo on
PORTD.0

      DDRC = 0xFF;        // set all pins on portC to output  (could also use DDRC =
0b11111111)

      lcd_init();         // set lcd in 4 bit mode, 2-line mode, with cursor on and set to
blink
```

```
    DDRD = 0b00000000;


    unsigned int Dvalue;

    lcd_string("Your LCD is working.");        // if your LCD is wired up correctly, you will see this text
                                                                                                    // on it when you power up your Micro-controller board.

    lcd_row(1);

    lcd_string("Good luck! :)");

    init_timer();



    interrupt_init();
    port_init();

    while(1)
    {
            SonarVal1 = getSonar1();
            SonarVal2 = getSonar2();
            lcd_clear();
            lcd_string("S:");
            lcd_int(SonarVal1);
            lcd_string(" ");
            lcd_int(SonarVal2);
            Dvalue = PIND & 0b00000111;
            for (i=0; i<10; i++)
            {
                    lcd_delay();
            }

            if (Dvalue == 0b00000101)
            {
            motor_move(1550);//1300 was right, 1500 was center, 1700 was left
            }
            else if (Dvalue == 0b00000110)
            {
            motor_move(1800);
            }
            else if (Dvalue == 0b00000011)
            {
```

```
            motor_move(1200);
            }
            else if (SonarVal1 < 1000)
            {
            motor_move(1800);
            }
            else if (SonarVal2 < 1000)
            {
            motor_move(1250);
            }
            if (Dvalue == 0b00000000)
            {
            dc_move(1725);
            }
            else
            {
            dc_move (2600);
            }

    }



    return 0;


}

    int count = 0;

    SIGNAL(SIG_UART0_RECV)
    {


    char c = UDR0;
            if (c == 'Z')
                    {
                    dc_move(1725);
                    stop =  1;

                    count++;

                    }
            else
             {
                    stop = 0;
```

```
                dc_move(2600);

                count = 0;

            }

}
```

CONCLUSION

This robot has been assembled and tested to perfection, it has not been an easy task to finish this but I prevailed. I acquired the knowledge and skills required to complete a great machine that does exactly what I originally intended for it to do. I will like to take this opportunity to thank all of the teacher assistants for the class I couldn’t have done it with out you. Special thanks to Adam that really took the time to explain every question I had. In the

end the class has shown me a wide variety of new things and I had a painful but very fun experience.