Tank

Kimlin M. Lyons
December 20, 2002

University of Florida
Department of Electrical and Computer Engineering
EEL 5666c
Intelligent Machines Design Laboratory

TABLE OF CONTENTS

ABSTRACT

Tank is an autonomous robot designed to collect data and transmit it via a light signal to a receiver located in 3-dimensions.

While doing obstacle avoidance, Tank collects temperature readings. When enough readings have been collected, Tank (using a vision sensor) locates an array placed somewhere on a wall and transmits the data via a laser. The information from the array is sent to a receiver and then sent to hyperterminal to be viewed.

INTRODUCTION

For Tank to achieve its goals, a variety of sensors were used. Tank incorporated bumper switches and IR's for obstacle avoidance, dc-motors with a motor driver chip for motion, the CMUcam (vision sensor) to locate a target, servos to actuate the laser, laser/laser driver (on Tank) to transmit collected temperature readings, a photodiode array and receiver (off Tank) to receive transmitted data and send to hyperterminal, and finally the ATMega323 chip which functioned as the brains of Tank.

INTEGRATED SYSTEM

The brains of Tank was the ATMega323 chip used on the MegaAVR-Development board. Basic features on the development board are 32K of In-System Programmable FLASH memory, one USART, eight (10 bit) analog inputs, 9 I/0 controlled LED's, and three built in PWM generators.

All code was written in C and compiled using the avrgcc compiler. The compiler is free and was downloaded from avrfreaks.com. The programming environment was avrstudio3.55: this is a platform which included a makefile generator, color coded word processor, and a simple debugger (useful only for simple debugging issues).

MOBILE PLATFORM

I did not want to be concerned with building a platform in AutoCAD, so I opted to use an already constructed platform. I purchased a toy tank from Toys R Us from which I stripped everything except two dc-motors. The motor driver I designed in Protel using the L293NE chip from TI controls the dc-motors. Each flyback diode is rated for 1A.

Since the tank was made for children, I knew its construction was robust, but an issue to deal with was heat. The tanks entire construction is essentially plastic and the MCU (microcontroller) generates approximately 250mW. If the MCU is left in direct contact with the tank for an extended period of time, the plastic would eventually melt and possibly catch fire. To avoid this situation, the MCU was placed outside the tank on a wood base. The wood base conformed to the topology of the tank so it blended in naturally.

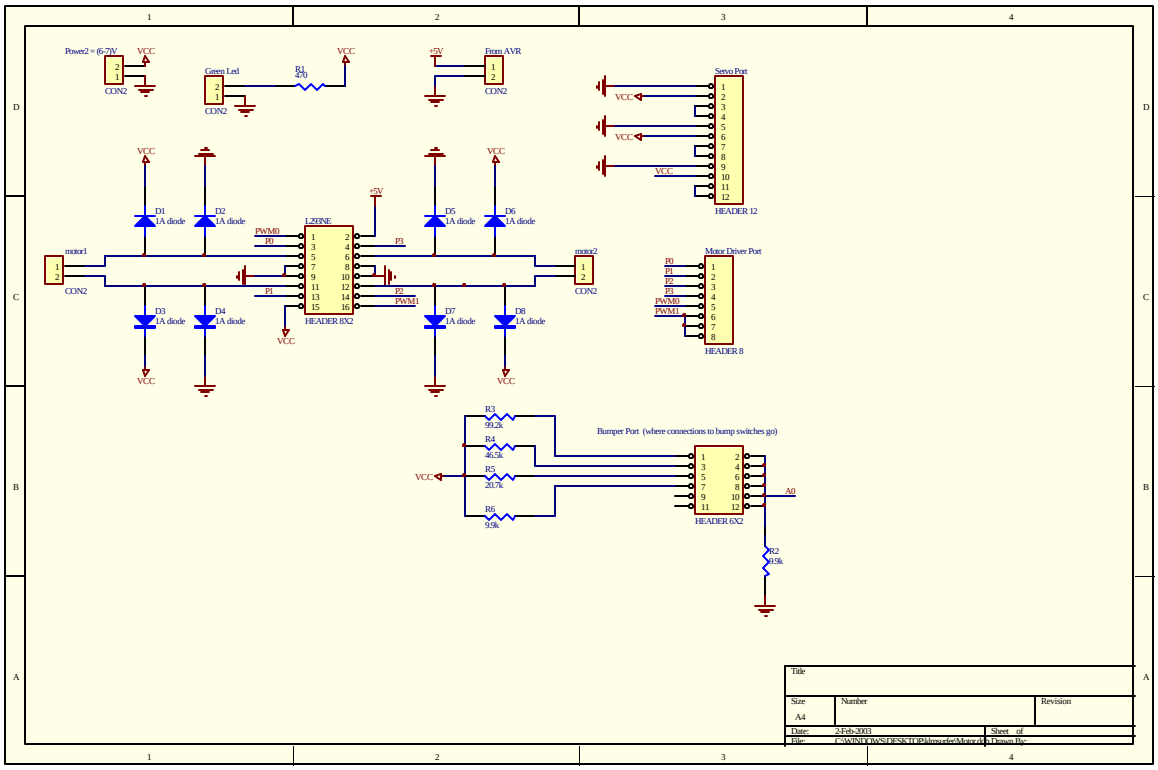
Other electrical parts (two PCB's which I made in Protel and CMUcam) that generated heat would be placed either on the wood base or on spacers elevated from the plastic and in well ventilated areas. With these measurements in place, a safe environment for the plastic platform and electronics was achieved. Because everything was plastic, I thought cutting and shaping it would be simple, it was not. It was alot more work than I expected.

PCB DESIGN

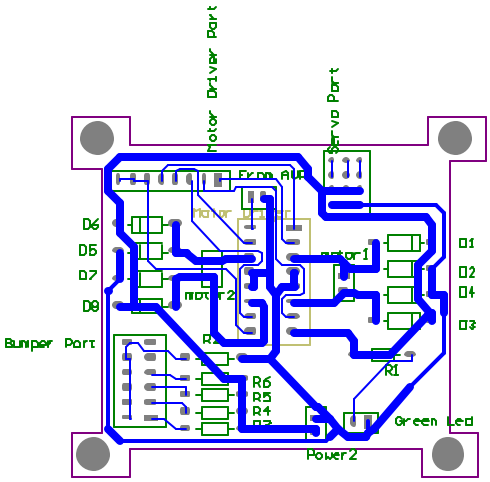
Simply enlarge schematics for a clearer view

1. Motor Driver

(a.) Schematic for Motor Driver (includes servo port, bumper port, and led)

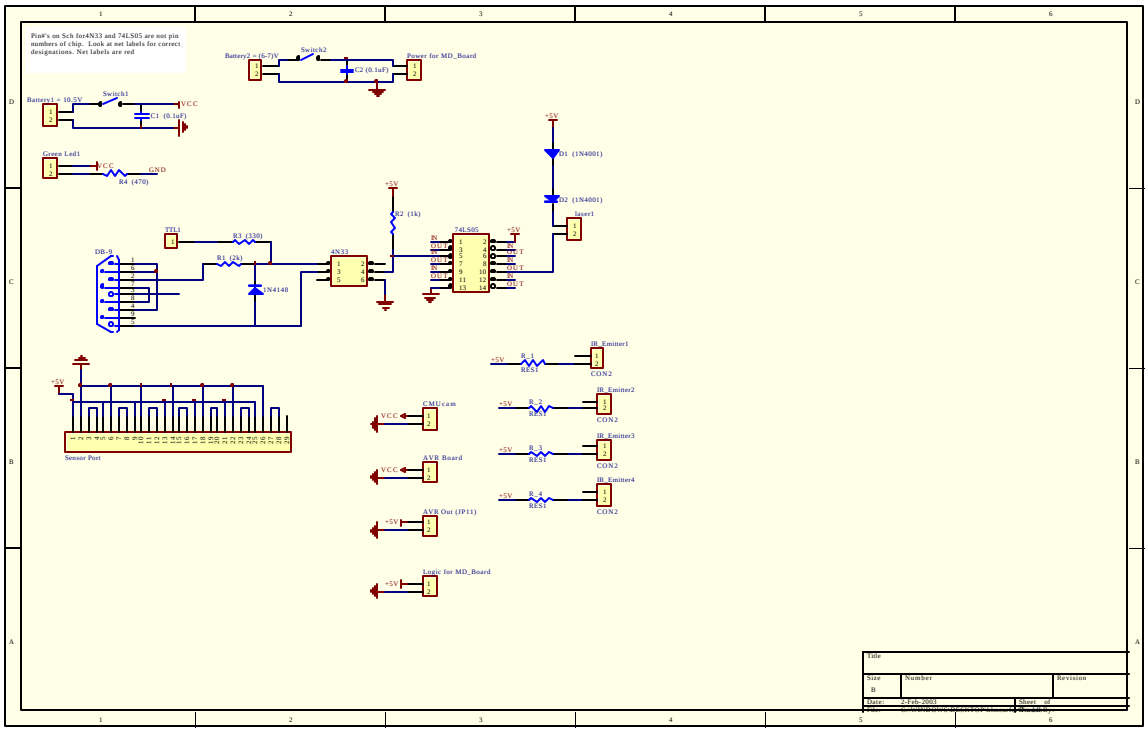


(b.) PCB Layout

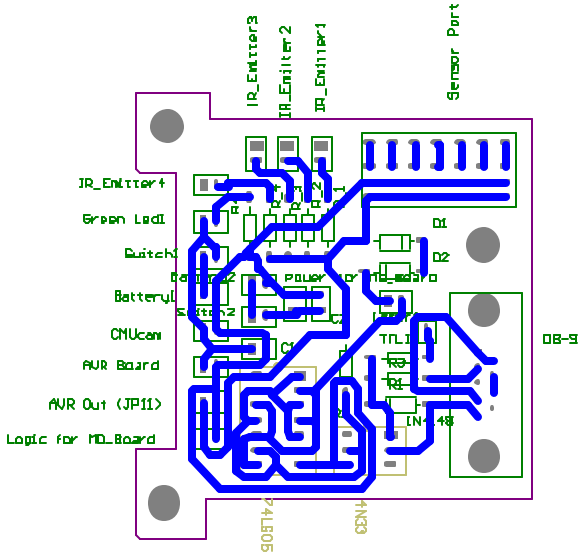


2. Laser Transmitter / Signal Organizer

(a.) Schematic



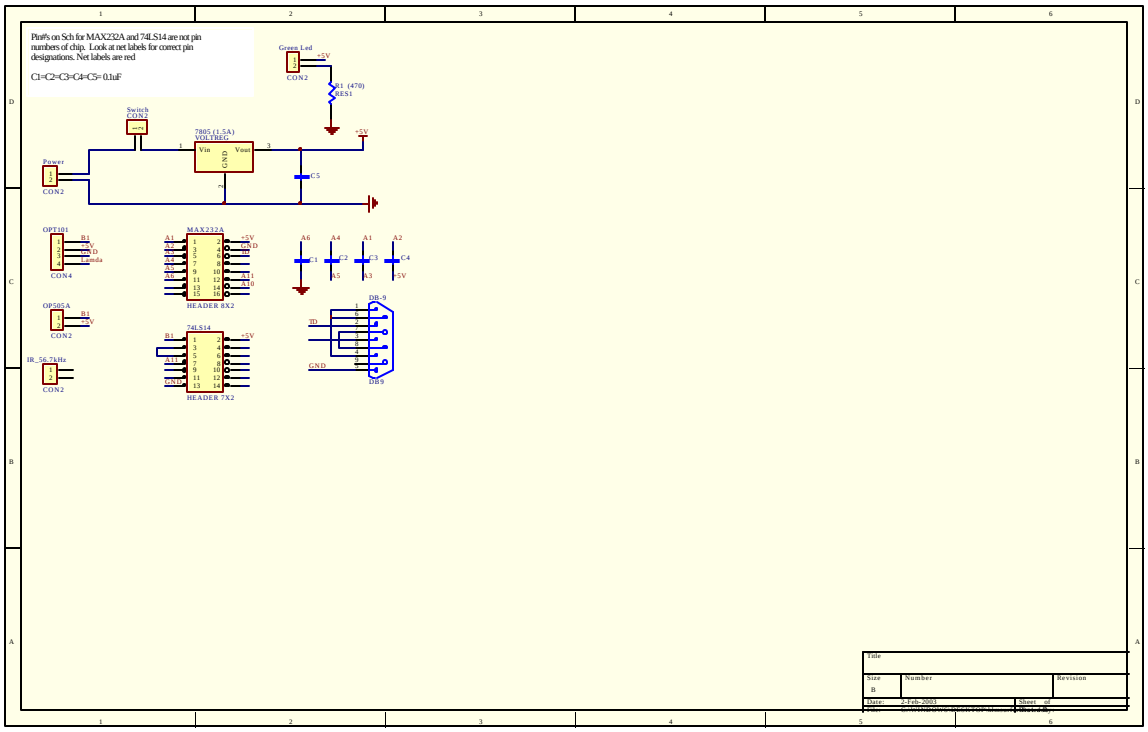
(b.) PCB Layout



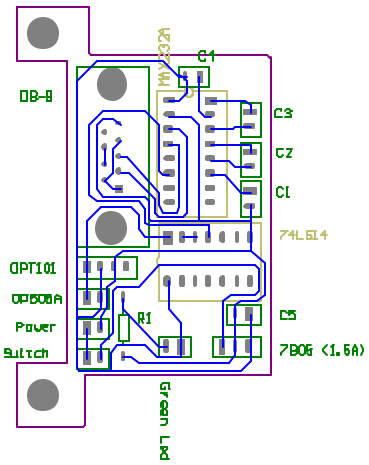
3. Receiver

(a.) Schematic (never used the OPT101 or IR_56.7kHz in schematic)

Note: The array which included an OP505A photodiode and a resistor was put on a breadboard (connects to the receiver through line B1 located on a CON2 which is labeled OP505A) .



(b.) PCB Layout



SENSORS

IR SENSORS

A. Sharp GP2D12 Analog IR

The GP2D12 consists of an IR emitter and a position sensitive detector. It outputs an analog voltage corresponding to the distance measured by the detector. I used two GP2D12's positioned at the front of the tank for obstacle avoidance.

BUMP SENSORS

Bump switches were also used for obstacle avoidance by providing a backup contingency should the IR's fail. Two bump switches placed at the front of the tank to determine if a collision occurs from the left, the right, or head on. Similarly, two bump switches on the back provide the same information except for the opposite direction. By determining the analog voltage, Tank knows where the collision occurred and can adjust its direction of motion appropriately.

TEMPERATURE SENSOR

Tank uses an LM34D (temperature sensor) to collect temperature readings. The LM34D outputs an analog voltage with a scale of +10.0mV/ degree Fahrenheit. The LM34D is calibrated directly in degrees Fahrenheit and can accurately differentiate temperature levels between +5 degrees to +300 degrees Fahrenheit.

LASER TRANSCEIVER

The laser transceiver was designed in Protel and consists of two elements (circuit design was found at www.geocities.com/SiliconValley/Lakes/7156/laser.htm):

1. A laser/laser transmitter located on Tank.

   (a.) The laser has an output power $\leq$ 5mW, a wavelength of 650nm $\pm$ 20nm, and requires approximately 3.5V and 40mA.
   (b.) The transmitter can accept both TTL and RS-232 signals. For RS-232 signals, a 4N33 opto-isolator couples the signal to the driver section of the circuit. The driver consists of a 7405 open-collector hex inverter IC with all the outputs of the inverters coupled together so that the laser diode will have enough drive current. The regulated 5V that feeds the laser diode is dropped by two diodes in series to produce 3.6V.

2. An array/receiver located off the tank.

   (a.) The array is simply one OP505A photodiode which changes the laser signal to an analog signal.
   (b.) The receiver consists of one 74LS14 hex inverter with pull down resistors to change the analog signal from the photodiode to a TTL signal. The TTL is then changed to an RS-232 signal via a MAX232A chip. The signal is then sent to the computer where it is viewed on hyperterminal.

CMUcam

The CMUcam is a SX28 MCU interfaced with a 0V6620 Omnivision CMOS camera on a chip that allows simple high level data to be extracted from the cameras streaming video. The board communicates via an RS-232 or a TTL serial port and some of its functionality's include:

-- Track user defined color blobs at 17 Frames Per Second
-- Find the centroid of the blob
-- Gather mean color and variance data
-- Transfer a real-time binary bitmap of the tracked pixels in an image
-- Arbitrary image windowing
-- Adjust the camera's image properties
-- 80x145 Resolution
-- Automatically detect a color and drive a servo to track an object upon startup
-- Ability to control 1 servo or have 1 digital I/O pin

The CMUcam was used to show simple object following and to adjust a laser onto a target. M packets sent from the CMUcam included 8 elements, of which only four were used. These included the middlemass x and y, pixel, and confidence values.

BEHAVIORS

Using the sensors described previously, Tank can:

1. Collect 400 temperature readings while doing obstacle avoidance. When 400 temperature readings have been collected (after ~ 1 minuet), Tank signals that its ready to transmit the data.

2. Transmit the collected temperature readings via a laser to a receiver which then sends the information to a computer to be viewed in hyperterminal. This part requires that I physically aim the laser onto the photodiode.

3. Position a laser onto a 1.25in x 1.25in orange square placed within a specific boundary using vision processing (the orange square is an imaginary array which would be used to receive data transmitted by the laser).

4. Follow a colored object using vision processing.

EXPERIMENTAL LAYOUTS AND RESULTS

CMUcam

The vision software (CMUcamGUI) found at www-2.cs.cmu.edu/~cmucam/downloads.html was used extensively. Specific features utilized where frame dumps and color picker. Color picker weights the RGB values from 0-240 over an entire frame. Based on these values, you can enter specific TC selections so as to lock onto specific targets. The light source and the orientation of the target proved critical in achieving a consistent lock on both the laser and target.

Achieving consistent lock on the laser proved difficult. Using the CMUcamGUI to view the laser, there would be a thin outer fringe of redness (it was almost impossible to get a consistent lock on this part) and then mostly white light ranging from approximately 150-240 spanning the RGB. By choosing a background that had RGB values of 170 or less and keeping the target within a given boundary (usually between
(x1, y1, x2, y2) = (20, 40, 60, 100)), the CMUcam was able to get a consistent lock on the laser using "TC 200 240 170 240 130 240".
Another method which I later thought of but did not experiment with would have been to place the target on a surface that strongly scattered the laser light, this may have produced more red than previously seen.

Bump Sensors

The output of the bump switch network was connected to an analog pin and the values for the different voltages were seen in hyperterminal. The Motor Driver schematic in
PCB DESIGN has a schematic of the bumper port. The box connecting the resistors and line A0 is where the connectors for the bumper switches go.

The values for each voltage were found using a voltage divider circuit and verified using a multimeter. The equation used is $V = 5*[R2 / (R2 + Ri,eq)]$ where Ri is used when one bumper switch is pressed and Req is used when two bumper switches are pressed; a head on collision either in the forward or backward direction puts resistors R3 and R4, or R5 and R6 in parallel with R2.

Servos

Two unhacked Futaba FP-S148 servos were used in positioning the laser onto the target. In order for the servos to accurately move the laser onto a 1.25in x 1.25in orange square target, they would need to have a high degree of resolution. This means that for every tick, the servo would turn with a very small angle.

Unfortunately, I was only able to get 3.5 degrees/tick when the servo functioned as a servo (by using the appropriate frequency and values in the output compare register, a PWM is generated that corresponds to a specific angle for the servo). For the laser to land on the target accurately, I would need a servo that had a resolution of roughly
0.5 degrees/tick.

In order to achieve this, I first used the servos in 8-bit mode (look at my code to see what this means) to adjust the laser to an appropriate location (the servos functioning as servos). Then I switch to 10-bit mode and change the output compare value appropriately which moves the

servos at 0.2 degrees/tick in one direction (servos functioning as hacked servos). Thus by switching between two different PWM's, I can locate where the laser is and position it accurately onto the target.

Temperature Sensor

The LM34D was connected directly to the ADC on the Atmega323. By comparing the temperature given by the LM34D and a thermometer, it was found that the values given by the LM34D were approximately 34 points off from the correct temperature. Thus 34 points was added to every value given by the LM34D to attain the correct temperature reading.

CONCLUSION

Many long hours of trial and error was a must, not to mention patience. Due to time and financial constraints, the robot was not completely realizable. The robots behaviors are broken up into 2 sections:

A. Section 1 consists of behaviors 1 and 2. That is, the robot collects 400 temperature readings while doing obstacle avoidance, signals when its done, then transmits the temperature readings to a receiver via a laser (I physically aim the laser onto the receiver) at which time the receiver sends the data to the computer to be viewed on hyperterminal.

B. Section 2 consists of behaviors 3 and 4. That is, the robot first follows a moving colored object using vision. When the robot looses lock on the object for more than 10 seconds, it starts its second routine of positioning a laser onto a 1.25in x 1.25in orange square. It does this four times, each time I place the orange square at a new location and wait for the robot to adjust the laser onto the square.

Things I would change:

1. I would use stepper motors to actuate the laser.

2. I would start work on the CMUcam much sooner due to all the unexpected problems.

3. I would concentrate more on making a robust tracking system.

4. I would spend more time learning AutoCAD.

DOCUMENTATION

Anita M. Flynn, Bruce A. Seiger, and Joseph L. Jones. Mobile Robots: Inspiration to Implementation; A.K. Peters, 1999.

Laser circuitry -- www.geocities.com/SiliconValley/Lakes/7156/laser.htm

C programming info -- www.cs.cf.ac.uk/Dave/C/CE.html

Atmega323 programming info -- www.avrfreak.com

ACKNOWLEDGEMENTS

Along with the written documents I would like to thank everybody who gave their assistance:

- IMDL TA, Jason Plew, for his suggestions and insight (not to mention patience)

-The students in IMDL who have helped and offered suggestions along the way

SOURCES FOR PARTS

Progressive Resources LLC - www.prllc.com

Digi-Key - www.digikey.com

Acroname - www.acroname.com

OfficeMax - 3642 SW Archer Rd. 352-378-2353

Lowes Home Improvement - 3500 SW Archer Rd. 352-376-9900

Radio Shack - Archer Rd. 352-375-2426

Electronics Plus - 2026 SW 34th St. 352-371-3223

Michaels Arts and Crafts - 3644 SW Archer Rd. 352-377-9797

Toy R Us - 6711 W Newberry Rd. 352-331-7778

APPENDICES

SECTION 1 --- This code refers to Section 1 as discussed in Conclusion

```
// Written by Kimlin Lyons
////////////////////////////Modified functions/////////////////////////////////////////////////////////////////////////
//  ADC_getreading( )  -- written by Kristen Allen, modified by Kimlin Lyons

/*****************************includes*********************************/
#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>
#include <stdlib.h>

/**********************************************************************/
/*****************************defines**********************************/
#define F_CPU                       6000000     //6Mhz
#define UART_BAUD_RATE     19200         //19200 baud

#define UART_BAUD_SELECT (F_CPU/(UART_BAUD_RATE*16l)-1)

//determinces which channel to read from on the A/D
#define RIGHT_IR          0
#define LEFT_IR           1
#define BUMP              2
#define TEMPERTURE    3

#define forward         0xA4
#define backward        0x54
#define right           0x64
#define left            0x94
#define stop            0x00

#define Threshold   48
/**********************************************************************/

/*****************************reference variable data*****************/
typedef unsigned char   u08;       //unsigned char is int from 0-255
typedef              char  s08;
typedef unsigned short  u16;
typedef               short s16;

/**********************************************************************/

/*****************************globals**********************************/
//global variables are seen by everyone and updated as they change in program
u08 avg_rightIR=0,
```

```
      avg_leftIR=0,
      tempert=0,
      rightIR=0,
      leftIR=0,
      bump=0;

u16 sum_rightIR=0,
      sum_leftIR=0;

/*********************************************************************/

/*********************delay*********************************************/
void delay(u16 delay_time) {
   do {
     u08 i=0;
     do {
     asm volatile("nop\n\t"
        "nop\n\t"
        "nop\n\t"
        "nop\n\t"
        ::);
     } while(--i);
   } while(--delay_time);
}

/*********************************************************************/

/*****************************************light*************************/
void light(u08 check)
{
   if(check==0)  {
      cbi(PORTC, PINC0);
      delay(0x5ff);
      sbi(PORTC, PINC0);
   }
   else if(check==1)  {
      cbi(PORTC, PINC1);
      delay(0x5ff);
      sbi(PORTC, PINC1);
   }
   else if(check==2)  {
      cbi(PORTC, PINC2);
      delay(0x5ff);
      sbi(PORTC, PINC2);
   }
   else if(check==3)  {
      cbi(PORTC, PINC3);
      delay(0x5ff);
```

```
      sbi(PORTC, PINC3);
   }
   else if(check==4)  {
      cbi(PORTC, PINC4);
      delay(0x5ff);
      sbi(PORTC, PINC4);
   }
   else if(check==5)  {
      cbi(PORTC, PINC5);
      delay(0x5ff);
      sbi(PORTC, PINC5);
   }

}

/*************************************************************************/
/*********************************send*********************************/
void send(u08 *ptr)
{
   while(*ptr != '\0')  {
      while ( !(UCSRA & (1<<UDRE)) ) {};  //wait till transmit buffer is empty
      UDR = *ptr;
      delay(0x8f);     //not using interrupt, so put delay ( ? < delay < 0x2ff)
      ptr++;
   }
}
/*************************************************************************/
/**********************UART_init( )***********************************/
void UART_init(void)
{  //initialize uart
   UBRRH = 0x00;
   UBRRL = UART_BAUD_SELECT;
   UCSRB = _BV(RXCIE)|_BV(RXEN)|_BV(TXEN);
   UCSRC = _BV(URSEL)|_BV(UCSZ1)|_BV(UCSZ0);
   UCSRA = 0x00;
}

/*************************************************************************/

/**********************ADC_init***************************************/
// Free running mode, sampling rate is (cpu clk)/prescale = 93.7 kHz

void ADC_init(void)
{
   //ADFR - puts A/D in free running mode,  ADPS210 - 101, 6MHz/32 =187.5kHz
   ADCSR = _BV(ADEN)|_BV(ADFR)|_BV(ADPS2)|_BV(ADPS1);
}
```

```
/*********************************************************************/

/*************************************MOTOR_int*********************/
void MOTOR_init(void)
{
   PORTB = forward;        //start off in the forward direction
}

/*********************************************************************/

/*************************************SENSORS_int*******************/
//  initialize sensors/actuators

void SENSORS_init(void)
{
   UART_init();
   ADC_init();
   MOTOR_init();
}

/*********************************************************************/

/***********************************ADC_getreading*******************/
//Function to read a specific channel with the desired reference voltage, then start
// the uart for transmission of the reference voltage value (changed to a decimal number)
// to the screen.

unsigned char ADC_getreading(int channel)
{
   u08 temp_valueH;

   outp((1<<REFS0)|(1<<ADLAR), ADMUX);

   if (channel == 0)  {}
     else if(channel == 1)  {
          sbi(ADMUX, MUX0);
     }
     else if(channel == 2)  {
          sbi(ADMUX, MUX1);
     }
     else if(channel == 3)  {
          sbi(ADMUX, MUX0);
          sbi(ADMUX, MUX1);
     }

   sbi(ADCSR, ADSC);                        //starts conversion
   while ( !(ADCSR & (1<<ADIF)) )  {};    //ADIF set when an ADC conversion done
   delay(0x2);         //without this delay, values at hyper are not correct
```

```
   temp_valueH = inp(ADCH);    //need only ADCH since 8-bit resolution. Get conversion
   sbi(ADCSR, ADIF);           //clear flag (flag sets when conversion is complete)

   return temp_valueH;

}

/*********************************************************************/

/*********************************motor*********************************/
void motor(u08 dir1, u16 delay1, u08 dir2)
{
  PORTB = dir1;
  delay(delay1);
  PORTB = dir2;
}
/*********************************************************************/

/*********************************motor_control*********************************/

void BUMP_IR(void)
{
   u16 time=3000, angle=900;

//Bumper check
   if(bump >= 10)  {
      if(bump>=25 && bump<=35)  {    //F_Left Bumper
         motor(backward, time, left);
      }
      else if((bump>=53 && bump<=63) || (bump>=74 && bump<=84))  {
         motor(backward, time, right);          //F_Right, F_HeadOn Bumper
      }
      else if((bump>=100 && bump<=110) || (bump>=195 && bump<=205))  {
         motor(forward, time, right);       //B_Left Bumper, B_HeadOn Bumper
      }
      else if(bump>=153 && bump<=163)  {       //B_Right Bumper
         motor(forward, time, left);
      }
      delay(angle);
      PORTB = forward;
   }

//IR check
   if( ((avg_rightIR - avg_leftIR) <= 2)  && (avg_leftIR >= Threshold+8))  {
      motor(right,0x1f,forward);       //if difference is small, both IR values ~ equal
   }
   else if((avg_rightIR >= Threshold) && (avg_leftIR < Threshold))  {
```

```
      motor(left,0x1f,forward);
   }
   else if((avg_rightIR < Threshold) && (avg_leftIR >= Threshold))  {
      motor(right,0x1f,forward);
   }
   PORTB = forward;

} //end of motor_control()

/*****************************************************************/

/*************************SENSORS_check*************************/
//check sensors
void SENSORS_check(void)
{
   u16 Heat[401];
   u08 static ctr1=0, ctr2=0;
   u16 static u=0;

   tempert = ADC_getreading(TEMPERTURE);
   rightIR = ADC_getreading(RIGHT_IR);
   leftIR = ADC_getreading(LEFT_IR);
   bump = ADC_getreading(BUMP);

   sum_rightIR += rightIR;
   sum_leftIR += leftIR;

   if(ctr1 >= 50)  {
      ctr1=0;
      avg_leftIR = sum_leftIR/50;
      avg_rightIR = sum_rightIR/50;

      sum_rightIR=0;
      sum_leftIR=0;
   }
   ctr1++;

//these values are updated roughly (400 times)/(1.10 min)
//PORTC lights up when array Heat is full

   if(ctr2 >= 50)  {
      ctr2=0;
      Heat[u]=tempert + 34;  //values stored in raw form, each u corresponds to u08 type
      u++;

      if(u>=400)  {
         PORTC=0;         //ready to Tx temperture readings to hyperterminal
         PORTB = stop;  //stop tank
```

```
        delay(0x3fff);
        delay(0x2fff);
        while(1)  {
          u=0;

    send("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\r");
    send("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\r\n\n");
    send("Temperture readings are in degrees Fahrenheit\r\n\n");
          delay(0x9ff);
          while(u<=400)  {
            u08 buff3[10];
            itoa(Heat[u], buff3, 10);
            send(buff3);
            send("\r");
            u++;
          }
        }
      } //end of nested if
    } //end of if
    ctr2++;


} //end of SENSORS_check

/*************************************************************************/

/*************************************main*************************/
int main(void)
{
    outp(0xff,DDRD);     //use all pins on portD for output
    outp(0xff,DDRC);     //used to see leds
    outp(0xff,DDRB);     //use PORTB to control motor direction
    PORTB=0;            //init PORTB
    PORTC=0xff;          //init PORTC

    delay(0x1fff);              // Gets rid of error message when program compiled
    SENSORS_init();        // Turn on UART_init(), ADC_init(), MOTOR_init()

    sei();              //Set global interrupt enable
    delay(0x2ff);    //good to do in lower level languages

    for(;;)  {

      SENSORS_check();
      BUMP_IR();

    } //end of  for loop
```

```
} //end of main

/*********************************************************************/
```

SECTION 2 --- This code refers to Section 2 as discussed in Conclusion

```
// Written by Kimlin Lyons
////////////////////////////Modified functions/////////////////////////////////////////////////////////////////////////
//  follow_target( )  -- written by unknown (source code can be found at www.seattlerobotics.com/video.htm), modified by Kimlin Lyons
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/*
   The elements of the laser are kept in the temp array.  Not sure why, but 1st element of
   M packet for laser is temp[6]=mx and last element is temp[13]=confidence. Since I'm
   using pointer indexing on temp array now, I just use the index n=(0,8) for elements in
   M packet

  CPU clock runs at 6MHz (6 million clk cycles per sec). By prescaling cpu clk by 1024,
  each clk cycle takes 6*10^6/1024 ~ 5.86kHz --> 0.171ms. T/C0 overflows every 255
  clk cycles and I'm looping n times, then delay in sec = { 1/[(cpu clk)/prescale] }*255*n
  = (0.171ms)*255*n = x sec.
*/


#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>
#include <string-avr.h>
#include <stdlib.h>

#define F_CPU                   6000000     //6Mhz
#define UART_BAUD_RATE     19200        //19200 baud

#define UART_BAUD_SELECT (F_CPU/(UART_BAUD_RATE*16l)-1)

#define Target          1
#define Laser           2
#define x_sel           3
#define y_sel           4
#define X               5
#define follow          6

#define forward         0xA0
#define backward        0x50
#define left            0x60
#define right           0x90
```

```
#define stop                0x00

/*******************************typedef   designations*******************/
typedef unsigned char  u08;          //8-bit int from (0,255)
typedef              char  s08;          //8-bit int from (-127,127)
typedef unsigned short u16;
typedef              short s16;
/************************************************************************/

/*******************************global variables***********************/
//global variables are seen by everyone and updated as they change
volatile s16 temp[40];      //volatile allows an interrupt to modify a variable
volatile u16 i;
volatile u08 lock1;

s16 *const ptrtemp;      //ptrL always pts to the same address
s16 T[12];

u08 m=0;
/************************************************************************/

/*******************************delay**********************************/
void delay(u16 delay_time) {
   do {
     u08 j=0;
     do {
     asm volatile("nop\n\t"
         "nop\n\t"
         "nop\n\t"
         "nop\n\t"
         ::);
     } while(--j);
   } while(--delay_time);
}

/************************************************************************/

/*****************************************light*************************/
void light(u08 check)
{

   if(check==0)  {
      cbi(PORTC, PINC0);
      delay(0x5ff);
      sbi(PORTC, PINC0);
   }
   else if(check==1)  {
      cbi(PORTC, PINC1);
```

```
      delay(0x5ff);
      sbi(PORTC, PINC1);
   }
   else if(check==2)  {
      cbi(PORTC, PINC2);
      delay(0x5ff);
      sbi(PORTC, PINC2);
   }
   else if(check==4)  {
      cbi(PORTC, PINC4);
      delay(0x2fff);
      sbi(PORTC, PINC4);
   }
   else if(check==5)  {
      cbi(PORTC, PINC5);
      delay(0x5ff);
      sbi(PORTC, PINC5);
   }

}

/*************************************************************************/

/**********************************SIGNAL(SIG_OVERFLOW0)***********/
SIGNAL(SIG_OVERFLOW0)          // signal handler for tcnt0 overflow interrupt
{
   u08 static ctr=23;

   if(ctr==0)  {
      ctr = 23;
      lock1++;
      light(0);
   }
   ctr--;
   TCNT0=0;

}

/*************************************************************************/

/******************SIGNAL(SIG_UART_RECV)*************************/
SIGNAL(SIG_UART_RECV)
// signal handler for receive complete interrupt

{
   register s16 led;

   led = UDR;
```

```
   temp[i] = led;
   i++;

}

/*********************************************************************/

/**********************SERVO_init*********************************/
void SERVO_init(void)
{

//This is all one line
_____________________________________________________________________
TCCR1A =
_BV(COM1A1)|_BV(COM1A0)|_BV(COM1B1)|_BV(COM1B0)|_BV(PWM10);
_____________________________________________________________________
  TCCR1B = _BV(CS12);     //TCNT1 counts with [(cpu clk)/256] = 23.256kHz
  TCNT1 = 0;                  //16-bit counter, only using 8-bit portion since PWM is 8-bit

  //init laser to point staright ahead in the xy-plane
  OCR1AL = 238;   //signal on PD6
  OCR1AH = 0;
  OCR1BL = 248;   //signal on PD5
  OCR1BH = 0;

  delay(0x1fff);

}

/*********************************************************************/

/*******************************UART_init*****************************/
void UART_init(void)
//initialize uart
{
   UBRRH = 0x00;
   UBRRL = UART_BAUD_SELECT;
   UCSRB = _BV(RXCIE)|_BV(RXEN)|_BV(TXEN);
   UCSRC = _BV(URSEL)|_BV(UCSZ1)|_BV(UCSZ0);
   UCSRA = 0x00;
}

/*********************************************************************/

/*******************************SENSORS_init**************************/
//  initialize sensors/actuators

void SENSORS_init(void)
```

```
{
   UART_init();
   SERVO_init();
}

/*************************************************************************/

/*************************************send*****************************/
void send(u08 *ptr)
{
   while(*ptr != '\0')  {
      while ( !(UCSRA & (1<<UDRE)) ) {};  //wait till transmit buffer is empty
      UDR = *ptr;
      delay(0x1ff);     //not using interrupt, so put delay ( ? < delay < 0x2ff)
      ptr++;
   }

}

/*************************************************************************/

/*************************updateL*************************************/
void update_temp(void)      //get new M packet
{
   i=0;
   send("TC\r");
   delay(0x2ff);

   u08 j;
   for(j=0; temp[j] != 'M'; j++)  {}      //set up data in temp for easy access
   j++;                           //inc so M (77) not stored
   ptrtemp = &temp[j];      //assign const ptrtemp to address of mx in temp

}

/*************************************************************************/

/*************************************OBJECT_init**********************/
void OBJECT_init(u08 object)
{
   u08 static ctr=0;
   //these arrays automatically have a null char (\0) at the end
   u08 Tt[]="TC 230 240 93 165 16 60\r";              //lock onto target (orange square)
   u08 Tl[]="TC 200 240 170 240 130 240\r";        //lock onto laser

   if(ctr= =0)  {       //init CMUcam once
      i=0;
      ctr++;
```

```
      send("\r");
      light(4);
      send("PM 1\r");       //get one packet after sending TC command
      light(4);
      send("RM 1\r");      //receive information in raw form
      light(4);
   }

   i=0;    //set temp back to 0th element
   if(object= =Target)  {  //init CMU for lock on target (orange square)
      send(Tt);         //CMU sends data to AVR, Rx interrupt stores incoming data to temp
      delay(0x2ff);
      update_temp();


     while(*(ptrtemp + 7) <= 35)  {          //get lock on target before preceding
        update_temp();
        light(5);
      }
      u16 n=0;
      while( *(ptrtemp + n) != '\0')  {  //store values of locked target for later use
         T[n] = *(ptrtemp + n);
         n++;
      }
   }
   else if(object = =follow)  {  //init CMU for lock onto moving target
      send("TW\r");   //TW locks CMU onto 1st color it sees in middle of view
      delay(0x2ff);
      update_temp();
   }
   else  if(object= =Laser)  {  //init CMU for laser lock
      send(Tl);
   }

   delay(0x4ff);

}

/*****************************************************************************/

/*****************************servo_contr*****************************/
void servo_contr(u08 pwm_mode, u08 x_resol, u08 y_resol)
{  //for OCR1X=0 in both pwm_modes, servos do not move

   if(pwm_mode= =10)   {
      sbi(TCCR1A, PWM11);    //10-bit PWM
   }
```

```
   else if(pwm_mode= = 8)  {
      cbi(TCCR1A, PWM11);    //8-bit PWM
   }

   OCR1A=x_resol;
   OCR1B=y_resol;

   delay(0x1ff);
}

/*************************************************************************/

/*********************************adj_laser**************************/

void adj_laser(u08 select, u08 xy)
{   //Adjusts laser based on mx an my positions

   u16 n=0;
   u08 epsolon;  //determines how soon to break from loop when using either mx or my

   if(select= =x_sel)  {  //for x
      n=0;
      epsolon=3;

       //loop determines if laser should go left or right
       while( ((*(ptrtemp + n) - T[n]) < -1) && *(ptrtemp + n) != 0 )  {
         xy++;
         servo_contr(8,xy,0);      //8-bit PWM
         update_temp();
         delay(0x8ff);
      }
      servo_contr(10,1,0);
   }
   else if(select= =y_sel)  {    //for y
      n=1;
      epsolon=5;
       //CMU must have laser lock for this loop to work or else laser just goes straight up
      servo_contr(10,0,8);
       while(*(ptrtemp + 7) <= 35)  {  //when loop ends, you know CMU has lock on laser
         update_temp();
      }
   }

   while(abs(*(ptrtemp + n) - T[n]) > epsolon)  {     //keep moving laser until condition
      update_temp();
   }
   servo_contr(X,0,0);
```

```
}

/*********************************************************************/

/***********************************follow_target*********************/
void follow_target(void)
{   //this funtions tracts a moving target and moves tank in direction of moving target
   u08 check1=0;
   while(!check1)  {
      update_temp();
      if(*(ptrtemp + 0) > 55 && *(ptrtemp + 7) > 25)  {    //(ptrtemp+0)= mx
         PORTB = right;                                    //(ptrtemp+7) = confidence
      }
      else if(*(ptrtemp + 0) < 35 && *(ptrtemp + 7) > 25)  {
         PORTB = left;
      }
      else if(*(ptrtemp + 6) > 250 && *(ptrtemp + 7) > 50)  {    //(ptrtemp+6) = pixel
         PORTB = backward;
         delay(0x100);
      }
      else if(*(ptrtemp + 6) < 100 && *(ptrtemp + 7) > 25)  {
         PORTB = forward;
         delay(0x100);
      }
      else if(*(ptrtemp + 7) <= 25)  {
         lock1=0;  //init lock1
         while(*(ptrtemp + 7) <= 25)  {
            PORTB = stop;
            update_temp();

            TIMSK = _BV(TOIE0);   //enable TOIE0 in TIMSK
            TCNT0 = 0;            //start counting in TCNT0 (T/C0) at $00
            TCCR0= _BV(CS02)|_BV(CS00);     //count at cpu/1024 or 0.17ms/count
            if(lock1 >= 10)  {
               cbi(TIMSK, TOIE0);  //disable interrupt
               check1=1;
               break;
            }
         } //end of 2nd while
         lock1=0;  //if lock1 is incremented but does not reach 10, set back to zero
         cbi(TIMSK, TOIE0);  //disable interrupt if lock1 < 10 so don't see PINC0 blinking

      } //end of last else if
      delay(0xff);
      PORTB=stop;
   } //end of 1st while

} //end of follow_target
```

```
/*********************************************************************/

/*****************************************main**********************/
int main(void)
{
   DDRB = 0xff;   //use all pins on portB for output  (used to control dc-motors)
   DDRD = 0xff;   //use all pins on portD for output  (used for servo control)
   DDRC = 0xff;   //use all pins on portC for output
   PORTC= 0xff;   //turn off all leds (by default, leds all turn on when DDRC=0xff)

   delay(0x1fff); //use before initializing uart, otherwise will get an error
   SENSORS_init();

   delay(0x5ff);
   sei();          //Set global interrupt enable

   delay(0x2fff);  //delay so I can change cords on AVR board
   delay(0x2fff);
   m=1;

   for(;;)  {

      if(m= =1)  {
         //For following target
         OBJECT_init(follow);
         update_temp();
         follow_target();

         //For adjusting laser onto target
         u08 ctr=5;
         while(ctr--)  {  //blink lights before moving on to next step
            light(2);
            delay(0x1fff);
         }

         ctr=4;
         while(ctr--)  {  //find target 4 times

            //For adjusting laser onto target
            servo_contr(8,238,248);

            delay(0x1fff);
            OBJECT_init(Target);
            delay(0x1fff);
            OBJECT_init(Laser);

            delay(0x5ff);
            adj_laser(y_sel,X);    //move laser in y-dir
```

```
            adj_laser(x_sel,238);  //move laser in x-dir

            u08 ctr2=5;
            while(ctr2--)  {
               light(2);
               delay(0x1fff);
            }

         }
         m++;
         PORTC=0;

      } //end of if

   } //end of for

}  //end of main

/*********************************************************************/
```

END OF REPORT