REU SUMMER 2005

Final Paper

George Gilmore

August 5, 2005

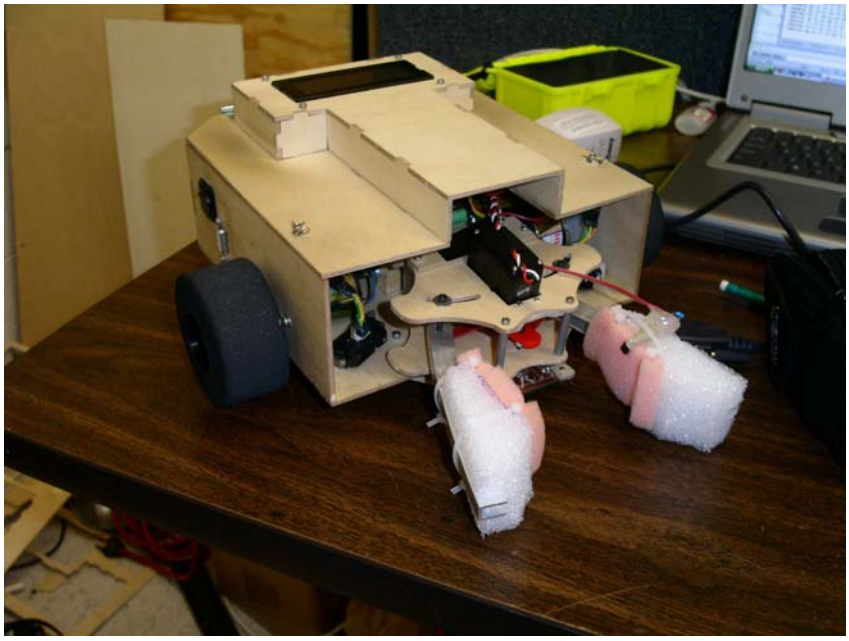


Table of Content

Abstract

294-Kelvin is a robot that sorts cans by temperature. The development of such a robot will relieve the stress accounted at gatherings, where servings of large quantities of beverages need to be served. This paper describes the development of a robot that will be able to detect a can and determine its temperature and sort it accordingly. 294-Kelvin uses the MAVRIC-IIB development board equipped with the Atmega-128 microprocessor. It follows a line and uses various sensors: the AD592 Temperature sensor, Hamamatsu photo reflectors, and GP2D120 Distance Measuring IR sensors to complete its task.

Executive Summary

The goal was to create an autonomous robot that had the ability to detect and sort cans by temperature. Several challenges were posed: detecting the can, accurately determining the temperature of the cans, and finally, lifting the cans. Ideal hardware and software was needed in order to appropriately confront these problems and to generate optimum solutions.

A circular track laid out by black electrical tape on a white background will be the environment of operation. With a very basic understanding of robotics, 294-Kelvin is established as a line-fallowing robot. Although this was is not ideal in real circumstances it was necessary as a starting point for future development.

This needed to be accomplished first, as it was necessary for the robot to execute its task. Concept designs were drawn, and simple mechanical calculations were conducted in order to determine where the task of lifting a full 12 oz. Soda Can could be accomplished. With positive prospects appropriate servos were selected.

294-Kelvin's sensor packages were next to be determined. To achieve desired line following behavior, an array of four of the Hamamatsu photo reflectors were selected. A comparator is integrated into the Hamamatsu photo reflector unit resulting in a digital output signal of high or low, 5V and 0V respectively. The 4 digital signals could simply be read in any I/O pins, which allowed for easy interfacing with the sensors.

Originally, the OPB745 photo reflectors were going to be used in the same arrangement, however, the lack of consistency resulted in a need for a more reliable sensor. However, with other members of the lab testing the OPB745, it was discovered

that the results were inconstant, because of the environment of the black and white surfaces were reflecting with similar values under certain lighting conditions.

The AD592 Temperature sensor used was actually attended as a back up. The DS1620 is as a digital thermometer was originally attended for use. Working, it was going to be a more reliable sensor, however, with not enough acquired experience, interfacing the sensor was too difficult. With limited knowledge, the AD592 sensor was set up in a circuit that would adjust current with temperature change, and thereby varying voltage. With voltage change as a result of temperature, analogue values can be read in through the A/D port on the MAVRIC-IIB. Therefore, a tabled array was created to match particular voltage readings with actual Celsius temperature readings.

Several problems were posed with the AD592 Temperature sensor. In retrospect, I would have gone with a different temperature sensor. With the present circuit established for the AD592 Temperature sensor, changes in the source voltage would have an effect on the voltage readings. To achieve more consistent results, a capacitor was also connected between power and GND of the AD592 Temperature sensor to reduce electronic noise.

GP2D120 Distance Measuring IR sensors were used for diction of the can. Two GP2D120 sensors were setup at different angles; one was attended to detect the can at proximity, while the other was directed to signal when the can is directly in front of the robot. No problems were experienced with these sensors, even when used in places with a lot of ambient light.

The LCD screen played a vital role in the development of 294-Kelvin. Its primary function was to be used as a debugging tool. However, as 294-Kelvin developed the

LCD screen became more used as a device to display information of procedures and information for the user.

At completion, 294-Kelvin could determine between a cold and warm Soda Can. It was able to keep track of how many drinks it had picked up. It went along the circuit and displaced the cold drinks to the left of the line while displacing the warm Soda Cans to the right of the line.

Introduction

These days, after a party is common to find all sorts of articles on the ground and soda cans are often found amongst them. The objective is to create an autonomous robot that will sort these cans by temperature. Having 294-Kelvin at a party, you will know which sodas will have to be refrigerated, and which ones can still be served. To develop this concept for every day use 294-KELVIN will be equipped with a line tracker, a temperature sensor, and IR distance measuring sensors to enable it to complete its task, to sort cans.

Integrated Systems

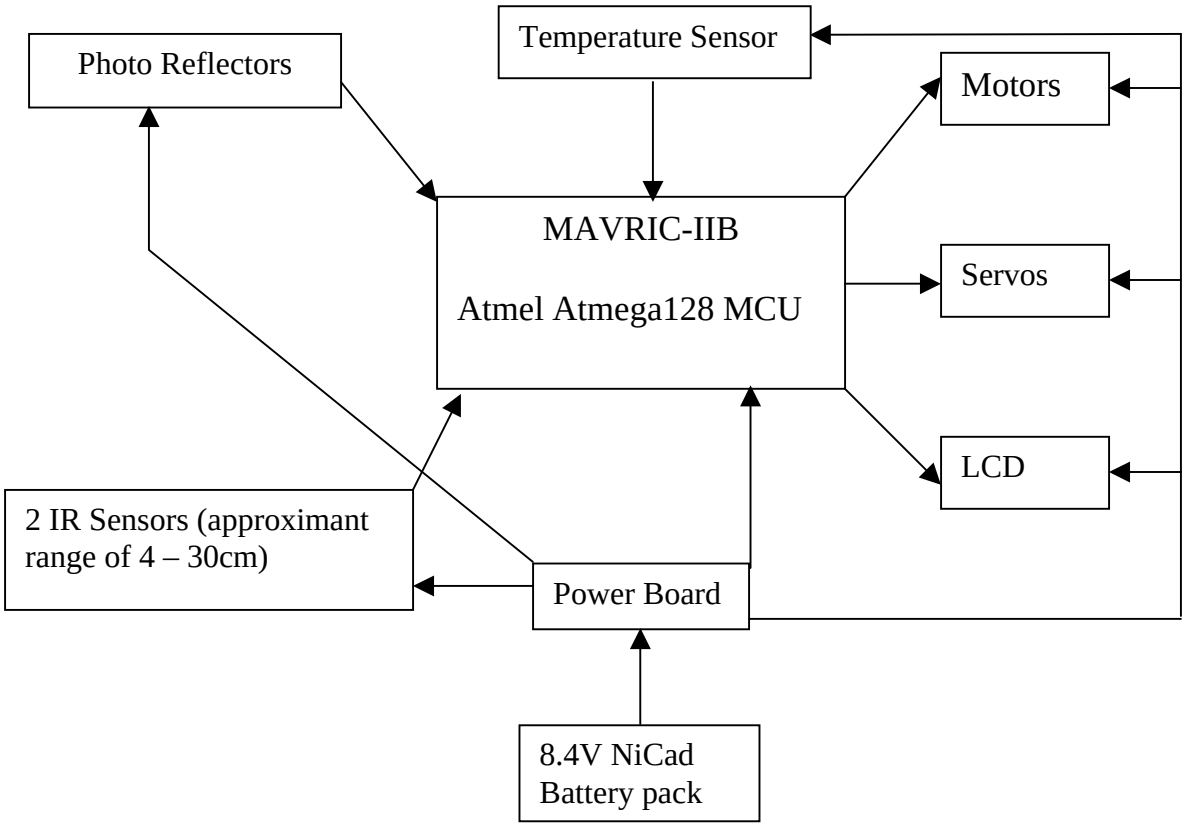
The BDMICRO MAVRIC-IIB microcontroller board based on the Atmega128 MCU was chosen for use on 294-Kelvin. It incorporated all our desired features and more in its original package, which allows for future development. 294-Kelvin's programming was compiled using Programmers Notepad by WinAVR and loaded on to memory using PonyProg. Feature present on the microcontroller such as: up to 51 digital I/O pins, 8 channel, 10-bit A/D converter, and 6 high resolution PWM outputs, make it ideal. More information about the specifications of the BDMICRO MAVRIC-IIB microcontroller board can be found at the BDMICRO website, http://www.bdmicro.com/.

294-Kelvin's sensor suite comprises of an array of four Hamamatsu photo reflectors, the AD592 Temperature sensor, and two GP2D120 Distance Measuring IR sensors. The AD592 and the two GP2D120 sensors' analogue output were simply read in through A/D pins on Port F of the Atmega128. Comparators is integrated into the Hamamatsu photo reflector units resulting in a digital output signal of high or low was read in on standard I/O pins on Port D.

The actuators: two DC motors and two servos were all driving using the built in fast PWM output channels. It was necessary to initialize the appropriate registries to enable the PWM output channels.

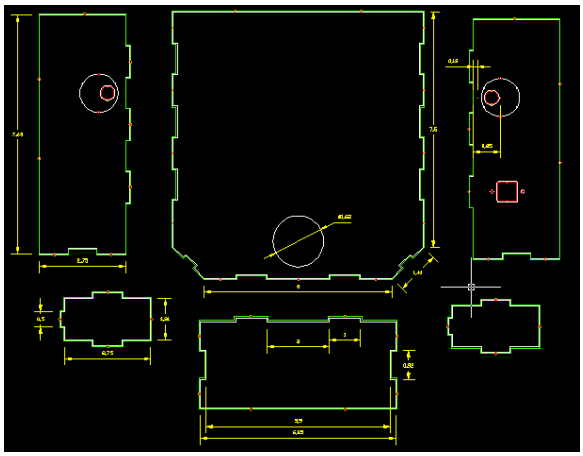
294-Kelvin drew its power from an 8.4V NiCad 2600mAh battery pack. A power board designed by Joseph Gaita, a 2005 University of Florida REU student, provided 4 unregulated power and GND headers, and six 5V regulated Power and GND heads using two 5 Amp regulators. This allowed unregulated power of 8.4V to the motors and the MAVRIC-IIB, while supplying the necessary VCC to sensors, LCD and servos.

Below is a flow chart schematic of the 294-Kelvin's integrated system.



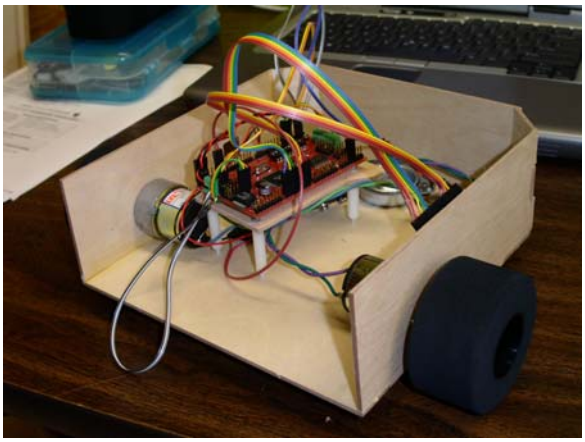
Mobility Platform

294-KELVIN will have a direct differential motor drive system and a third caster wheel mounted at the base rear for stability and mobility. Two 7.2V DC motors are used in a direct differential drive system. Each motor has a RPM of 175, a reduction ratio of 50:1, stall torque of 99.04 oz-inches, and an outside diameter of 37mm. The supporting caster is Flange Mount Ball Transfer Systems 1" Ball with a maximum load capacity of 75lbs. An 8.4V NiCad 2600mAh battery pack will be used as the primary battery source. A prototype of 294-KELVIN will first be constructed out of 3mm plywood using the T-Tech machine.



This is an AutoCAD diagram displaying the pieces that will form the mobility platform.

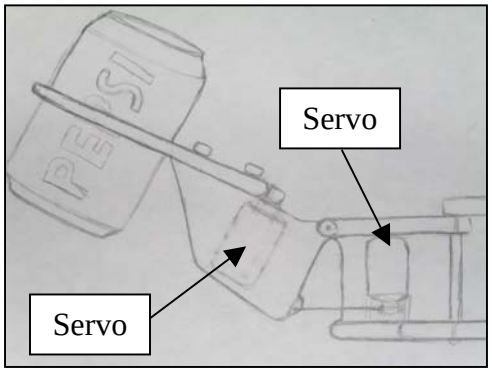
This is a modification to the side panels shown at the top used for guidance mounting the motors.



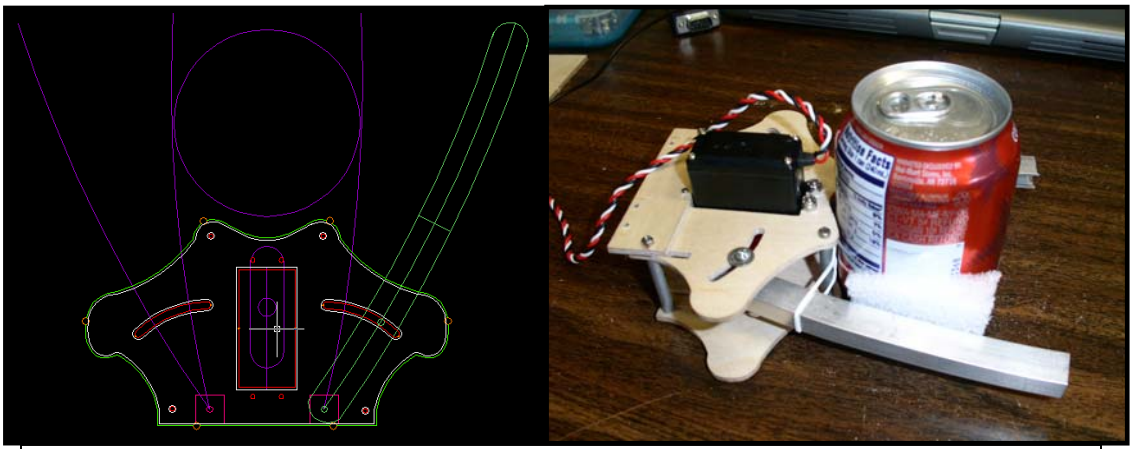
Actuation

Actuation posed as the major mechanical challenge in this project. 294-Kelvin had to be able to lift a full 12 oz. Soda Can. This was generally uncommon for a robot of his size.

Several concept designs using the principles of mechanical advantage were proposed. It was realized that servos were the most appropriate mechanical actuators to be used for the task. After several mechanical calculations were conducted, it was determined that two Hitec HS-645MG High-Torque 2BB MG were going to be able to get the job done. One servo was going to be used to initiate the gripping mechanism, while the second servo, utilizing mechanical advantage was to be used for lifting. This concept can be seen in the diagram to the right.



The final gripping mechanism design and actual product can see seen in the pictures below. 294-Kevlin was fully capable of displacing full 12oz. Soda cans, however, a full mini 8oz. Soda can was used during demonstrations.

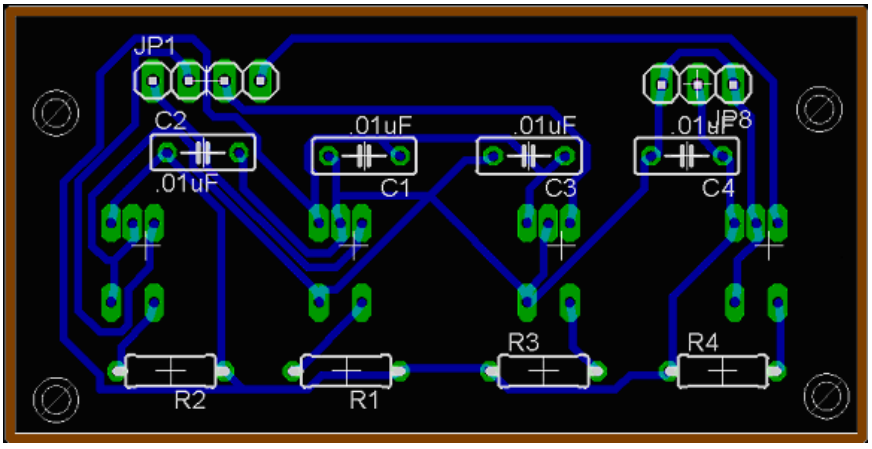


This is a diagram of the AutoCAD drawing to the left and the finished product to the right.

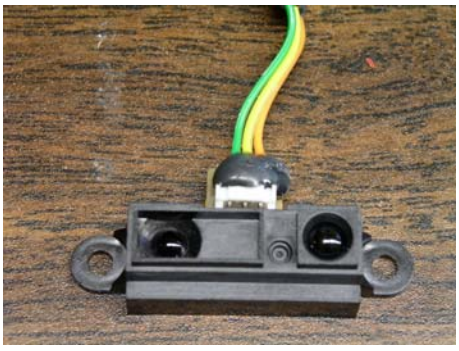
Sensors

Photo reflectors

To achieve line following behavior on the robot, it will be equipped with four Hamamatsu photo reflectors. A comparator is integrated into the Hamamatsu photo reflector unit resulting in a digital output signal of high or low, 5V and 0V respectively. The array of four Hamamatsu photo reflectors will ensure the 294-Kelvin will never navigate off the line at appropriate speeds. The inner to units will handle straight lines and slightly curvy lines while the outer two will be used for sever correction, tight turns and to detect intersections.



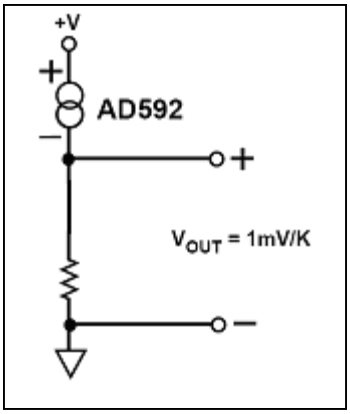
Sharp GP2D120 Distance Measuring Sensor (IR)



The Sharp GP2D120 IR sensor accurately determines range to target between 4cm and 30cm; as well as can be used as a proximity detector to detect objects between 0cm and 50cm. Two of these sensors are used by 294-Kelvin as a proximity

detector. One will be used to detect the can approximately when it's 30 cm away and cause the robot to approach the can at a reduced speed. And which then, the second IR senor will identify when the can is directly in front of the robot, and thereby initiating the gripping mechanism.

Temperature Sensor (Special Sensor)



The AD592 Temperature sensor was used to determine whether a Soda Can was cold or warm (room temperature). The AD592 has a wide range of operating temperatures, which range between -25˚C to +125˚C; with a single supply of 4V to 30V it offers 0.5˚C temperature measurement accuracy. The diagram to the left shows the circuit used for the AD592. I then hooked it up the A/D port on the ATMEGA and read in the change in voltage. A look up table was created with experimental where I would match up the particular voltage reading with a specific temperature. With this setup, 294-Kelvin would give accurate temperature readings I determined to be ± 2ºC.

The initial plan was to use the DS1620 Temperature Sensor. The DS1620 is as a digital thermometer with +/-0.5C accuracy from a 0˚C to +70˚C. The temperature is read out over a 3-wire serial bus in 2's complement format with 9 bits of resolution. However, with not enough acquired experience interfacing the sensor was too difficult. However, if you were able to interface this sensor, I would encourage its use, as it presumes to be a more precise sensor.

Behavior

Line following (obstacle avoidance): 294-KELVIN will run on a closed line track, a black line on a white background. Once turned on, the robot will display on the LCD that you must place it onto the line track. There will be a time delay from the time you switch it on till the motors begin to rotate. The robot will follow the line until is approaches a Soda can.

Object Detection: Once the robot has detected that something is in the way, which will be a can, it will slow down and stop. With its distance measuring sensor it will align itself so that the Soda can will be directly in front it. Once is this established, 294-KELVIN will edge forward until the can hit the bump sensor inside the gripping mechanism and signal to 294-KELVIN to initiate the gripping mechanism.

Gripping Mechanism, Temperature Determination, and Raising Mechanism: Once the gripping mechanism is complete then, 294-KELVIN will initiate the temperature determination, which will record the temperature of the can. The temperature of the can will be displayed to the LCD. The information will be sent to the processor, and with predetermined limits the processor will establish whether it is a warm or a cold Soda can. After this is complete then 294-KELVIN will either displace the can to the left of the line if it cold and to the right if its warm. After this is complete then 294-KELVIN will continue along the line and will follow the same procedures listed.

LCD: An LCD will be fastened to the exterior shell and will output information such as the number of cans that it has picked up, display the temperature and the number of cold and warm cans that were picked up.

Dimensions and Specifications

Length: 330mm (13 in)

Width: 287mm (11.25 in)

Height: 120mm (4.75 in)

Weight (w/o battery pack): 1311g (46.25 oz.)

Power Source: 8.4V NiCad/NiMH

Parts List

Below is a table listing of all the parts that were used in creating 294-Kelvin. I was aiming for a budget of $250 dollars.

Description	Part #	Quantity	Unit Price	Price
MAVRIC-IIB Development Board	MAV2BPH	1	$100	$100.00
Serial Port Dongle Programmer for STK Port	AVR-PG1	1	$11.95	$11.95
Gear Head Motor - 7.2V DC, 50:1, 175 RPM (6mm Shaft)	GHM-04	2	$21.95	$43.90
Hitec HS-645MG High-Torque 2BB MG Servo J	LXUZ90	2	$36.99	$73.98
20x4 LCD Display with LED Backlight	#36	1	$16.00	$16.00
Sharp GP2D120 Distance Measuring Sensor	GP2D120	2	$8.25	$16.50
Flange Mount Ball Transfer Systems 1" Ball	NFMC	1	$7.62	$7.62
Universal Hub - 6mm (pair)	HUB-02	1	$8.00	$8.00
Green Dot Sumo Tire - 2.88"D x 1.5"W (pair)	TSR-04	1	$19.95	$19.95
	TOTAL			$297.90

Conclusion

Project complete. The project took approximate 2.5 months to complete. 294-Kelvin was the first generation can sorting robot by temperature. Some day, I hope to develop on the idea and come up with a concept that would actually have practical use in the world. However, it was a great learning experience. Being a mechanical engineer, it was the first time I have ever working to extensively with electronics and software programming. Special thanks to Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz, and William Dubel for guidance during REU Summer 2005 at the Machine Intelligence Lab, University of Florida.

Documentation

Special thanks and acknowledgment need to be award to persons how have greatly aided in the development of software models or have granted me permission of use of particular software.

Main Program

```
////////////////////////////////////////////////////////////////////
//
//          FILE NAME: 294_Kelvin_Main.c
//          AUTHOR: George Gilmore
//          Date: Summer 2005
//          SUMMER REU PROGRAM
//          DISCRIPTION:
//
//
////////////////////////////////////////////////////////////////////


//*************************INCLUDE STATEMENTS*********************

#include <inttypes.h>
#include <stdarg.h>
#include <avr/io.h>
#include <avr/signal.h>
#include <stdlib.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/delay.h>

#include "Headers/GLOBALS.h"
#include "Headers/PHOTO.h"
#include "Headers/adc.h"
#include "Headers/DELAY.h"
#include "Headers/TRACK.h"
#include "Headers/TEMP.h"
#include "Headers/Temp_Table.h"
#include "Headers/MOTOR.h"
#include "Headers/SERVO.h"
#include "Headers/LCD.h"

//***********************END OF INCLUDE STATEMENTS*****************


////////////////////////////////////////////////////////////////////
```

```
//    Name:  BANNER
//   Input:  None
//  Output:  None
// Fuction:  Displays introductory messages when 294-Kelvin is
//           turned on
////////////////////////////////////////////////////////////////////
void BANNER()
 {
  LCD_SEND_COMMAND(0x01);
  delay_1ms(5);
  SETPOS(1,0);
  LCD_TEXT("     294-KELVIN");
  SETPOS(3,4);
  LCD_TEXT("HERE TO PICK");
  SETPOS(4,4);
  LCD_TEXT("UP YOUR CANS");
  delay_100ms(100);
  delay_100ms(100);
  CLEAR_LINE(3);
  SETPOS(3,3);
  LCD_TEXT("PLEASE PLACE ME");
  CLEAR_LINE(4);
  SETPOS(4,4);
  LCD_TEXT("ON THE TRACK");
  for (int i=0; i<4; i++)
    {
     delay_100ms(100);
    }
  LCD_SEND_COMMAND(0x01);
 }

////////////////////////////////////////////////////////////////////
//     Name:  INIT_DEVICES
//    Input:  None
//   Output:  None
// Function:  Initiallizes all the devices that will used
////////////////////////////////////////////////////////////////////
void INIT_DEVICES()
{
 DDRA=0xFF;                    // Setting PORT A to Output
 delay_1ms(5);
 LCD_INIT();                   // Initialize the LCD
 delay_1ms(250);
 SERVO_INIT();               // Initialize the servos
 delay_100ms(10);
 delay_100ms(10);
 MOTOR_INIT();               // Initialize the the motors
 delay_1ms(250);
 adc_init();             // Initialize A/D Converter
 delay_1ms(250);
 PHOTO_INIT ();                // Initialize Photo reflectors
 delay_1ms(250);
 SET_TABLE();                  // Sets up the lookup table used to
determine the degree of Temperature
 delay_1ms(250);
}
```

```
void IR ()
{
 IR_L = adc_readn(0, 25);           // Samples Pin 4, PORT F 25 times
then takes the average
 IR_R = adc_readn(1, 25);           // Samples Pin 5, PORT F 25 times
then takes the average
}

/*////////////////////SERVO VALUES/////////////////////////////////////

 SERVO_CONTROL(1,2000);       // Claw is fully open
 SERVO_CONTROL(2,2000);       // Claw is fully raised
 SERVO_CONTROL(2,750);        // Claw is fully lowered
 SERVO_CONTROL(1,1450);       // Can is gripped
 SERVO_CONTROL(2,2000);       // Claw is fully raised
 SERVO_CONTROL(2,750);        // Claw is fully lowered
 SERVO_CONTROL(1,2000);       // Claw is fully open
 SERVO_CONTROL(2,2000);       // Claw is fully raised

 *//////////////////////////END OF SERVO VALUES//////////////////////


////////////////////////////////////////////////////////////////////
//     Name:  SET_CAN
//    Input:  None
//   Output:  None
// Function:  Routine that positions the can and takes temperature
//            the reading
////////////////////////////////////////////////////////////////////
void SET_CAN()
{
 CLEAR_LINE(2);

 SERVO_CONTROL(2,750);              // Claw is fully lowered

 delay_100ms(100);
 delay_100ms(100);

 TEMP();
 CLEAR_LINE(2);
 SETPOS(2,0);
 LCD_TEXT("TEMP: ");
 Temper = TEMP_DISPLAY(T);
 delay_1ms(5);
 LCD_SEND_INT(Temper);
 LCD_SEND_DATA(0xDF);
 LCD_TEXT("C");

 SERVO_CONTROL(1,1750);                    // Claw is closing in
 MOTOR_CONTROL(900,900,FWD);
 delay_100ms(100);
 MOTOR_CONTROL(900,900,FWD);
 delay_100ms(100);
 MOTOR_CONTROL(0,0,FWD);
 delay_100ms(100);
 SERVO_CONTROL(1,1550);                    // Can is gripped
```

```
 for ( int j=0; j<8; j++)                  // This for loop is used to
run the update Temperature routine 8 times
   {
    TEMP_2();
    CLEAR_LINE(3);
      SETPOS(3,0);
      LCD_TEXT("UPDATE:");
      Temper = TEMP_DISPLAY(T_2);
      delay_1ms(5);
      LCD_SEND_INT(Temper);
      LCD_SEND_DATA(0xDF);
      LCD_TEXT("C");
      for (int i=0; i<4; i++)
        {
         delay_100ms(100);
        }
   }
 TEMP_2();
 TEMP_COMPARE();                      // Function that compares T and T_2
to deterement condition of can
 delay_100ms(100);
 SERVO_CONTROL(1,1450);               // Can is gripped
 delay_100ms(100);
 delay_100ms(100);
 SERVO_CONTROL(2,2000);               // Claw is fully raised
 can++;     // Counter that keeps track of cans is incremented by one
 SETPOS(1,15);
 LCD_SEND_INT(can);
 delay_100ms(100);
 delay_100ms(100);
 do
   {
    TRACK_SLOW();
   }
 while (LL==1 && RR==1);
   {
    MOTOR_CONTROL(0,0,FWD);
      CLEAR_LINE(1);
      SETPOS(1,3);
      LCD_TEXT("INTERSECTION");
      if (TURN == RIGHT)
        {
         INTER_TURN_RIGHT();
         LEFT_TURN_BACK();
        }
      else if (TURN == LEFT)
        {
         INTER_TURN_LEFT();
         RIGHT_TURN_BACK();
        }
   }
}


void DETECT ()
{
```

```
 IR();
 if (IR_L >= 225)
   {
      do{
            CLEAR_LINE(1);
            SETPOS(1,0);
            LCD_TEXT("CAN DETECTED");
          IR();
          TRACK_SLOW();
         } while(IR_R <=280);

    MOTOR_CONTROL(0,0,FWD);
      CLEAR_LINE(1);
      SETPOS(1,0);
      LCD_TEXT("CAN FOUND");
      SET_CAN();
   }
}

void BEHAVIOR()
{
 PHOTO();
 TRACK();
 //TRACK_FAST();
 DETECT();
}


int main(void)
  {
   INIT_DEVICES();                        // Calls a function that
initializes all the devices

   SERVO_CONTROL(2,2000);           // Claw is fully raised
   SERVO_CONTROL(1,2000);           // Claw is fully open
   can = 0;                               // Initiallizes the can
counter at 0
   PORTB ^=0X01;                    // Toggle LED
   BANNER();
   CLEAR_LINE(1);
   SETPOS(1,15);
   LCD_SEND_INT(can);               // Displays can counter value

   while (TRUE)
     {
      BEHAVIOR();
       }

   return 0;
}
```

Header Files

GLOBALS.h

```
#define EN 0x80
#define RS 0x20

#define MotorR   OCR3A
#define MotorL   OCR3B
#define FWD      0x02
#define REV        0x01
#define Clockw   0x03
#define cClockw  0x00


uint16_t IR_L;
uint16_t IR_R;
uint16_t PR_1;
uint16_t PR_2;
uint16_t PR_3;
uint16_t PR_4;
uint16_t T;
uint16_t T_2;
uint8_t can;

const int maxtext=40;
//unsigned int TEMP_READ(void);

int oldDir, oldMR, oldML;
int LL, L, R, RR;

int TURN;
int RIGHT=1;
int LEFT=2;
int NONE=0;

int TRUE=1;       // Used for while loop

int TEMP_TABLE[25];
int Tem;
int Temper;

void delay_1ms(uint8_t);
void delay_1us(uint8_t);
void delay_100ms(uint8_t);

void LCD_TEXT(char[]);
void LCD_SEND_INT(int);
void LCD_SEND_COMMAND(unsigned char);
void LCD_SEND_DATA(unsigned char);
void CLEAR_LINE(int);
void BANNER(void);
void SETPOS(int,int);

void LCD_INIT(void);
```

```
void SERVO_INIT(void);
void MOTOR_INIT(void);
void PHOTO_INIT(void);
void INIT_DEVICES(void);

void MOTOR_CONTROL (int, int, int);
void SERVO_CONTROL(int,int);

void PHOTO (void);
void IR (void);
void TEMP(void);
void TEMP_2(void);
void TEMP_COMPARE(void);

void TRACK(void);
void TRACK_FAST(void);
void TRACK_SLOW(void);

void DETECT(void);
void SET_CAN(void);
void BEHAVIOR(void);

void INTER_TURN_LEFT(void);
void INTER_TURN_RIGHT(void);
void LEFT_TURN_BACK(void);
void RIGHT_TURN_BACK(void);
```

PHOTO.h

```
void PHOTO_INIT()
{
 DDRD &= 0x0F;                        // Initialize photo reflector
inputs on MCU
 PORTD |= 0xF0;                 // Enable pull up resistors for photo
reflectors
}


void PHOTO ()
{
 // 1 = white
 // 0 = black

 RR = PIND >> 4;
 R = PIND >> 5;
 LL = PIND >> 6;
 L = PIND >> 7;

 RR &= 0x01;
 R &= 0x01;
 LL &= 0x01;
 L &= 0x01;

 CLEAR_LINE (4);
```

```
 SETPOS(4,0);
 LCD_SEND_INT(LL);
 LCD_SEND_INT(L);
 LCD_SEND_INT(R);
 LCD_SEND_INT(RR);
}
```

adc.h

Acknowledgment: BD-MICRO open source code for use of adc.h

```
/*
 * $Id: adc.h,v 1.1 2003/12/11 01:35:00 bsd Exp $
 */

#ifndef __adc_h__
#define __adc_h__

void     adc_init(void);

void     adc_chsel(uint8_t channel);

void     adc_wait(void);

void     adc_start(void);

uint16_t adc_read(void);

uint16_t adc_readn(uint8_t channel, uint8_t n);

#endif
```

```
/*
 * $Id: adc.c,v 1.3 2005/03/22 19:12:10 bsd Exp $
 */

/*
 * ATmega128 A/D Converter utility routines
 */

#include <avr/io.h>
#include <stdio.h>


/*
 * adc_init() - initialize A/D converter
 *
 * Initialize A/D converter to free running, start conversion, use
 * internal 5.0V reference, pre-scale ADC clock to 125 kHz (assuming
 * 16 MHz MCU clock)
 */
```

```
void adc_init(void)
{
  /* configure ADC port (PORTF) as input */
  DDRF  = 0x00;
  PORTF = 0x00;

  ADMUX = _BV(REFS0);
  ADCSR = _BV(ADEN)|_BV(ADSC)|_BV(ADFR) |
_BV(ADPS2)|_BV(ADPS1)|_BV(ADPS0);
}


/*
 * adc_chsel() - A/D Channel Select
 *
 * Select the specified A/D channel for the next conversion
 */
void adc_chsel(uint8_t channel)
{
  /* select channel */
  ADMUX = (ADMUX & 0xe0) | (channel & 0x07);
}


/*
 * adc_wait() - A/D Wait for conversion
 *
 * Wait for conversion complete.
 */
void adc_wait(void)
{
  /* wait for last conversion to complete */
  while ((ADCSR & _BV(ADIF)) == 0)
    ;
}


/*
 * adc_start() - A/D start conversion
 *
 * Start an A/D conversion on the selected channel
 */
void adc_start(void)
{
  /* clear conversion, start another conversion */
  ADCSR |= _BV(ADIF);
}


/*
 * adc_read() - A/D Converter - read channel
 *
 * Read the currently selected A/D Converter channel.
 */
uint16_t adc_read(void)
{
  return ADC;
```

```
}


/*
 * adc_readn() - A/D Converter, read multiple times and average
 *
 * Read the specified A/D channel 'n' times and return the average of
 * the samples
 */
uint16_t adc_readn(uint8_t channel, uint8_t n)
{
  uint16_t t;
  uint8_t i;

  adc_chsel(channel);
  adc_start();
  adc_wait();

  adc_start();

  /* sample selected channel n times, take the average */
  t = 0;
  for (i=0; i<n; i++) {
    adc_wait();
    t += adc_read();
    adc_start();
  }

  /* return the average of n samples */
  return t / n;
}
```

DELAY.h

```
void delay_1us(uint8_t i) //method which waits i us.
{
 uint8_t j;
 while ( i )
 {
  for (j=0 ; j<4 ; j++ );
  i--;
 }
}

void delay_1ms(uint8_t i) //method which waits i ms.
{
 uint8_t j, k;
 while ( i )
 {
  for (j=0 ; j<200 ; j++ )
 {
  for ( k=0 ; k < 2; k++);
 }
 i--;
```

```
 }
}

void delay_100ms(uint8_t a)
{
 uint8_t i;
 for ( i=0; i<a; i++)
 {
  delay_1ms(100);
 }
}
```

TRACK.h

```
void TRACK()
{
 IR ();
 PHOTO();

 if (LL== 0 && L==0 && R == 0 && RR==0)   // Place on track
   {
    MOTOR_CONTROL (0,0,FWD);
      CLEAR_LINE(1);
      CLEAR_LINE(2);
      CLEAR_LINE(3);
      SETPOS(1,0);
      LCD_TEXT ("PLACE ON TRACK");
      delay_100ms(25);
      CLEAR_LINE(1);
      TRUE=1;
   }
 if (RR==1 && L==0 && R==0 && RR==1)
   {
    MOTOR_CONTROL(2000,2000,FWD);                    // Forward
   }
 else if (L==0 && R==1)                              // Adjusting Left
   {
      MOTOR_CONTROL(2000,2550,FWD);
      delay_1ms(8);
      MOTOR_CONTROL(2000,2250,FWD);
   }

 else if (L==1 && R==0)                              // Adjusting
Right
   {
      MOTOR_CONTROL(2550,2000,FWD);
      delay_1ms(8);
      MOTOR_CONTROL(2250,2000,FWD);
   }

 else if (LL== 1 && RR==0)          // Adjusting Strongly Right
   {
      do
        {
```

```
        MOTOR_CONTROL (1600,200,FWD);
        delay_1ms(14);
        MOTOR_CONTROL (1800,350,FWD);
        PHOTO();
       }
     while (R==1);
       {
        MOTOR_CONTROL (2050,1800,FWD);
       }
  }

 else if (LL==0 && RR==1)                 // Adjusting Strongly Left
   {
    do
      {
         MOTOR_CONTROL (200, 1600, FWD);
       delay_1ms(14);
       MOTOR_CONTROL (350,1800, FWD);
       PHOTO();
      }
    while (L==1);
      {
       MOTOR_CONTROL (1800,2050,FWD);
      }
  }

 else if (LL== 1 && L==1 && R == 1 && RR==1)    // Place on track
   {
    MOTOR_CONTROL (0,0,FWD);
      CLEAR_LINE(1);
      CLEAR_LINE(2);
      CLEAR_LINE(3);
      SETPOS(1,0);
      LCD_TEXT ("PLACE ON TRACK");
      delay_100ms(25);
      CLEAR_LINE(1);
      TRUE=1;
   }
}

void TRACK_FAST()
{
 IR ();
 PHOTO();

 if (LL== 0 && L==0 && R == 0 && RR==0)   // Place on track
   {
    MOTOR_CONTROL (0,0,FWD);
      CLEAR_LINE(1);
      CLEAR_LINE(2);
      CLEAR_LINE(3);
      SETPOS(1,0);
      LCD_TEXT ("PLACE ON TRACK");
      delay_100ms(25);
      CLEAR_LINE(1);
      TRUE=1;
   }
```

```
 if (RR==1 && L==0 && R==0 && RR==1)
   {
    MOTOR_CONTROL(3500,3500,FWD);                    // Forward
   }
 else if (L==0 && R==1)                              // Adjusting Left
   {
      MOTOR_CONTROL(3500,4000,FWD);
      delay_1ms(8);
      MOTOR_CONTROL(3500,3800,FWD);
   }

 else if (L==1 && R==0)                              // Adjusting
Right
   {
      MOTOR_CONTROL(4000,3500,FWD);
      delay_1ms(8);
      MOTOR_CONTROL(3800,3500,FWD);
   }

 else if (LL== 1 && RR==0)          // Adjusting Strongly Right
   {
      do
        {
         MOTOR_CONTROL (1600,200,FWD);
         delay_1ms(14);
         MOTOR_CONTROL (1800,350,FWD);
         PHOTO();
        }
      while (R==1);
        {
         MOTOR_CONTROL (2050,1800,FWD);
        }
   }

 else if (LL==0 && RR==1)                 // Adjusting Strongly Left
   {
    do
      {
         MOTOR_CONTROL (200, 1600, FWD);
       delay_1ms(14);
       MOTOR_CONTROL (350,1800, FWD);
       PHOTO();
      }
    while (L==1);
      {
       MOTOR_CONTROL (1800,2050,FWD);
      }
   }

 else if (LL== 1 && L==1 && R == 1 && RR==1)    // Place on track
   {
    MOTOR_CONTROL (0,0,FWD);
      CLEAR_LINE(1);
      CLEAR_LINE(2);
      CLEAR_LINE(3);
      SETPOS(1,0);
      LCD_TEXT ("PLACE ON TRACK");
```

```
      delay_100ms(25);
      CLEAR_LINE(1);
      TRUE=1;
   }
}


void TRACK_SLOW()
{
 PHOTO();
 if (L==0 && R==0)                                    // FORWARD SLOWLY
   {
    MOTOR_CONTROL(1000,1000,FWD);
   }

 if (L==0 && R==1)                                    // Adjusting Left
Slowly
   {
      MOTOR_CONTROL(1000,1550,FWD);
      delay_1ms(8);
      MOTOR_CONTROL(1000,1250,FWD);
   }

 if (L==1 && R==0)                                    // Adjusting
Right Slowly
   {
      MOTOR_CONTROL(1550,1000,FWD);
      delay_1ms(8);
      MOTOR_CONTROL(1250,1000,FWD);
   }

 else if (LL== 1 && RR==0)                 // Adjusting Strongly Right
   {
    do
      {
       MOTOR_CONTROL (1600,200,FWD);
       delay_1ms(14);
         MOTOR_CONTROL (1800,350,FWD);
         PHOTO();
        }
      while (R==1);
        {
         MOTOR_CONTROL (1050,800,FWD);
      }
   }

 else if (LL==0 && RR==1 )                 // Adjusting Strongly Left
   {
      do
        {
         MOTOR_CONTROL (200, 1600, FWD);
         delay_1ms(14);
         MOTOR_CONTROL (350,1800, FWD);
       PHOTO();
        }
      while (L==1);
        {
```

```
         MOTOR_CONTROL (800,1050,FWD);
        }
   }
}


void INTER_TURN_RIGHT()
{

 MOTOR_CONTROL(900,900,FWD);
 CLEAR_LINE(1);
 SETPOS(1,0);
 LCD_TEXT ("INCH FORWARD");
 delay_100ms(100);
 delay_100ms(100);
 delay_100ms(100);
 delay_100ms(30);
 MOTOR_CONTROL(0,0,FWD);
 MOTOR_CONTROL(1500,1500,Clockw);
 delay_100ms(100);
 delay_100ms(100);

 do{
      MOTOR_CONTROL(1700,1700,Clockw);
      PHOTO();
      }while (L==1);

       MOTOR_CONTROL(0,0,FWD);
       MOTOR_CONTROL(1100,1100,FWD);

       delay_100ms(100);
       delay_100ms(100);
       delay_100ms(100);
       delay_100ms(100);

       MOTOR_CONTROL(0,0,FWD);
       delay_100ms(25);
       SERVO_CONTROL(2,750);                    // Claw is fully
lowered
       delay_100ms(100);
       SERVO_CONTROL(1,2000);                   // Claw is fully open
       delay_100ms(100);
       SERVO_CONTROL(2,2000);                   // Claw is fully raised
       MOTOR_CONTROL(1100,1100,REV);

       delay_100ms(100);
       delay_100ms(100);
       delay_100ms(100);
       delay_100ms(100);

       MOTOR_CONTROL(0,0,FWD);
}

void INTER_TURN_LEFT()
{
 MOTOR_CONTROL(900,900,FWD);
 CLEAR_LINE(1);
```

```
 SETPOS(1,0);
 LCD_TEXT ("INCH FORWARD");
 delay_100ms(100);
 delay_100ms(100);
 delay_100ms(100);
 delay_100ms(30);
 MOTOR_CONTROL(0,0,FWD);
 MOTOR_CONTROL(1500,1500,cClockw);
 delay_100ms(100);
 delay_100ms(100);
 do
   {
      MOTOR_CONTROL(1700,1700,cClockw);
      PHOTO();
      }
 while (R==1);
      {
       MOTOR_CONTROL(0,0,FWD);
       MOTOR_CONTROL(1100,1100,FWD);

       delay_100ms(100);
       delay_100ms(100);
       delay_100ms(100);
       delay_100ms(100);

       MOTOR_CONTROL(0,0,FWD);
       delay_100ms(25);
       SERVO_CONTROL(2,750);                    // Claw is fully
lowered
       delay_100ms(100);
       SERVO_CONTROL(1,2000);                   // Claw is fully open
       delay_100ms(100);
       SERVO_CONTROL(2,2000);                   // Claw is fully raised
       MOTOR_CONTROL(1100,1100,REV);

       delay_100ms(100);
       delay_100ms(100);
       delay_100ms(100);

       delay_100ms(100);
       MOTOR_CONTROL(0,0,FWD);
    }
}


void LEFT_TURN_BACK()
{
 MOTOR_CONTROL(1500,1500,cClockw);
 delay_100ms(100);
 delay_100ms(100);
 do
   {
      MOTOR_CONTROL(1500,1500,cClockw);
      PHOTO();
   }
 while (L==1);
```

```
      MOTOR_CONTROL(0,0,FWD);
}

void RIGHT_TURN_BACK()
{
 MOTOR_CONTROL(1500,1500,Clockw);
 delay_100ms(100);
 delay_100ms(100);
 do{
      MOTOR_CONTROL(1500,1500,Clockw);
      PHOTO();
   }while (R==1);

    MOTOR_CONTROL(0,0,FWD);
}
```

TEMP.h

```
void TEMP()
{
 T = adc_readn(7, 75);                          // Samples PIN 7, PORT
F 40 times then takes the average
}

void TEMP_2()
{
 T_2 = adc_readn(7, 75);                                    // Samples
PIN 3, PORT F 10 times then takes the average
}

void TEMP_COMPARE()
{
  int T_3 = (T_2 - T);
  if (T_3 <= -2)
 {
  delay_1ms(100);
  CLEAR_LINE(1);
  SETPOS(1,0);
  LCD_TEXT("COLD DRINK");
  CLEAR_LINE(2);
  CLEAR_LINE(3);
  //SETPOS(2,0);
  //LCD_SEND_INT(T_3);
  TURN = LEFT;
  delay_100ms(100);
 }

 else
 {
  delay_1ms(100);
  CLEAR_LINE(1);
  SETPOS(1,0);
  LCD_TEXT("WARM DRINK");
  CLEAR_LINE(2);
```

```
  CLEAR_LINE(3);
  //SETPOS(2,0);
  //LCD_SEND_INT(T_3);
  TURN = RIGHT;
  delay_100ms(100);
 }
}
```

Temp_Table.h

```
void SET_TABLE(void)
{
 TEMP_TABLE[0] = 780;
 TEMP_TABLE[1] = 781;
 TEMP_TABLE[2] = 782;
 TEMP_TABLE[3] = 783;//805;
 TEMP_TABLE[4] = 784;//804;
 TEMP_TABLE[5] = 785;///803;
 TEMP_TABLE[6] = 786;///802;
 TEMP_TABLE[7] = 787;///801;
 TEMP_TABLE[8] = 788;//800;
 TEMP_TABLE[9] = 789;//799;
 TEMP_TABLE[10] = 790;//798;
 TEMP_TABLE[11] = 791;//797;
 TEMP_TABLE[12] = 792;//798;
 TEMP_TABLE[13] = 793;//797;
 TEMP_TABLE[14] = 794;//796;
 TEMP_TABLE[15] = 795;//795;
 TEMP_TABLE[16] = 796;//794;
 TEMP_TABLE[17] = 795;//793;
 TEMP_TABLE[18] = 796;//792;
 TEMP_TABLE[19] = 797;//791;
 TEMP_TABLE[20] = 798;//790;
 TEMP_TABLE[21] = 799;//789;
 TEMP_TABLE[22] = 800;//788;
 TEMP_TABLE[23] = 801;//787;
 TEMP_TABLE[24] = 802;//786;
 TEMP_TABLE[25] = 803;//785;
 TEMP_TABLE[26] = 804;//784;
 TEMP_TABLE[27] = 805;//783;
}

/*
int TEMP_DISPLAY (int Tem)
{
 if (Tem > 805) return 0;
 if (Tem < 780) return 100;

 for (int i=0; i<28; i++)
   {
    if (((TEMP_TABLE[i] >=Tem)) && (Tem >= TEMP_TABLE[i+1]))
      {
       if ((TEMP_TABLE[i] -Tem) >= (TEMP_TABLE[i+1] -Tem))
         {
```

```
          return (i+1);
         }
       else
         {
          return (i);
         }
      }
   }or before '}' token
Headers/Temp_Table.h:51: error: parse error before "return"
In file included from 294_Kelvin_Main.c:31:
Headers/Temp_Table.h:78:3: warning: no newline at end of file
make.exe: *** [294_Kelvin_Main.o] Error 1

> Process Exit Code: 2

 return 0;
}
*/

int TEMP_DISPLAY (int Tem)
{
 int i=0;

 if (Tem > 805) return 0;
 if (Tem < 780) return 100;

 do{
    if (Tem == TEMP_TABLE[i])
      {
         return i;
       }
    else
      {
         i++;
      }
   }while (Tem != TEMP_TABLE[i]);
   return i;
}
```

MOTOR.h

Acknowledgment: Julio Suarez for used of his motor code

```
void MOTOR_INIT()
{
 MotorR = 0;            // Motors set to stop
 MotorL = 0;

 TCCR3A = 0xA2;   // Use OC3A and OC3B
 TCCR3B = 0x1A;   // Fast PWM mode with division by 8
 ICR3 = 10000;          // max counter value
 DDRE |= 0x18;          // Set up outputs
```

```
 DDRC |= 0x03;        // PC0 = MR; PC1 = ML
                           // Direction pins initialized

 oldDir = 0x04;   // value that does not represent a direction
}


void MOTOR_CONTROL(int MR,int ML, int dir)            // actually it's
(ML,MR,dir) chooses are: FWD, REV,Clockw,cClockw
{
 int TempMR, TempML;
 int incMR, incML;
 int decMR, decML;


 if (oldDir != dir)
   {
      MotorR = 0;
      MotorL = 0;
      oldMR = 0;
      oldML = 0;
      delay_1ms(50);

      PORTC &= 0xFC;
      PORTC |= dir;
   }


 if(MR > oldMR)
   {
      TempMR = (MR - oldMR);
      TempMR = (TempMR/4);

      MotorR = (oldMR + TempMR);
      incMR = (oldMR + TempMR);
      delay_1ms(12);
      MotorR = (incMR + TempMR);
      incMR += TempMR;
      delay_1ms(12);
      MotorR = (incMR + TempMR);
      incMR += TempMR;
      delay_1ms(12);
      MotorR = MR;
   }


 if (ML > oldML)
   {
      TempML = (ML - oldML);
      TempML = (TempML/4);

      MotorL = (oldML + TempML);
      incML = (oldML + TempML);
      delay_1ms(12);
      MotorL = (incML + TempML);
      incML += TempML;
      delay_1ms(12);
```

```
      MotorL = (incML + TempML);
      incML += TempML;
      delay_1ms(12);
      MotorL = ML;
   }

 if(MR < oldMR)
   {
      TempMR = (oldMR - MR);
      TempMR = (TempMR/4);

      MotorR = (oldMR - TempMR);
      decMR = (oldMR - TempMR);
      delay_1ms(12);
      MotorR = (decMR - TempMR);
      decMR -= TempMR;
      delay_1ms(12);
      MotorR = (decMR - TempMR);
      decMR -= TempMR;
      delay_1ms(12);
      MotorR = MR;
   }

 if (ML < oldML)
   {
      TempML = (oldML - ML);
      TempML = (TempML/4);

      MotorL = (oldML - TempML);
      decML = (oldML - TempML);
      delay_1ms(12);
      MotorL = (decML - TempML);
      decML -= TempML;
      delay_1ms(12);
      MotorL = (decML - TempML);
      decML -= TempML;
      delay_1ms(12);
      MotorL = ML;
   }

 if (MR == oldMR)
   {
      MotorR = MR;
   }

 if (ML == oldML)
   {
      MotorL = ML;
   }

 oldMR = MR;
 oldML = ML;
 oldDir = dir;
}
```

SERVO.h

```
void SERVO_INIT()
{
 DDRB=0xFF;              // Setting PORT E to output
 TCCR1A=0xA8;            // TCCR- is timer counter control register for
timer 1
 TCCR1B=0x12;            // TCCR- is timer counter control register for
timer 1
 ICR1=20000;             // Setting a maximum value, which has the same
effect as setting the max frequency...after gets to 200000 resets to 0
}

void SERVO_CONTROL(int num,int val)
{
 //LCD_SEND_INT(num);               // num is the servo number
 //LCD_SEND_INT(val);               // val is the position of the servo
the range is between 750 and 2000

 if(num==1)
   {
    OCR1A=val;
   }

 else if (num==2)
   {
    OCR1B=val;
   }

 else
   {
    OCR1C=val;
   }
}
```

LCD.h

Acknowledgment: Clint Doriot for use of his SETPOS function

```
////////////////////////////////////////////////////////////////
// Name:   SETPOS
// Input:  int, int - the line and space respectively
// Output: None
// Action: sets the position of the LCD's cursor to a specific spot
////////////////////////////////////////////////////////////////
```

```
/*----- LCD VALUES --------------------------------------------*/
const unsigned char CLEAR = 0x01;
const unsigned char CURSOR_SET = 0x80;
const unsigned char LINE1 = 0x00;
const unsigned char LINE2 = 0x40;
const unsigned char LINE3 = 0x14;
const unsigned char LINE4 = 0x54;


void SETPOS(int row, int col)
{
      // Create basic cursor position command that will be used to set
the position
      unsigned char cursorPosCmd = 0x00;

      // Change the command to account for the specified row
      if (row == 1) cursorPosCmd = LINE1;
      else if (row == 2) cursorPosCmd = LINE2;
      else if (row == 3) cursorPosCmd = LINE3;
      else cursorPosCmd = LINE4;

      // Change the command to account for the column by offsetting the
address
      cursorPosCmd += col;

      // Change the command to be Write Cursor Position command by
      cursorPosCmd |= CURSOR_SET;

      // Change the cursor
      LCD_SEND_COMMAND (cursorPosCmd);

}

void CLEAR_LINE(int L)
 {
  if (L==1)
    {
     SETPOS(1,0);
     LCD_TEXT("               ");
    }

  else if (L==2)
    {
     SETPOS(2,0);
     LCD_TEXT("                    ");
    }

  else if (L==3)
    {
     SETPOS(3,0);
     LCD_TEXT("                    ");
    }

  else if (L==4)
    {
     SETPOS(4,0);
```

```
     LCD_TEXT("                    ");
    }
}


void LCD_SEND_COMMAND (unsigned char b)
{
 PORTA=(b>>4)|EN;
 delay_1ms(5);

 PORTA=(b>>4);
 delay_1ms(5);

 PORTA=(b & 0x0F)|EN;
 delay_1ms(5);

 PORTA=(b & 0x0F);
 delay_1ms(5);
}


void LCD_SEND_DATA (unsigned char b)
{
 PORTA=(b>>4)|EN|RS;
 delay_1ms(5);

 PORTA=(b>>4)|RS;
 delay_1ms(5);

 PORTA=(b & 0x0F)|EN|RS;
 delay_1ms(5);

 PORTA=(b & 0x0F);
 delay_1ms(5);
}

void LCD_SEND_INT(int val)
{
 LCD_SEND_DATA(val/10000 + 0x30);
 LCD_SEND_DATA(((val%10000)/1000)+0x30);
 LCD_SEND_DATA(((val%1000)/100)+0x30);
 LCD_SEND_DATA(((val%100)/10)+0x30);
 LCD_SEND_DATA(val%10+0x30);
}


void LCD_TEXT (char text[maxtext])
{
 int i=0;
 char b;
 while ((i < (maxtext+1)) & (text[i] != 0x00))
   {
    b=text[i];
    LCD_SEND_DATA (b);
    i++;
   }
 LCD_SEND_COMMAND(0x0C);
```

```
}


void LCD_INIT()
{

 /*Hardare setup information for LCD
 *PA7 = Enable on LCD PIN 6 on LCD
 *PA6 = Not connected
 *PA5 = RS signal Pin4 on LCD
 *PA4 = RW pin5 on LCD
 *PA3 .. PA0 = LCD7..4
 //end of LCD hardwar?e setup information*/

 delay_1ms(15);
 LCD_SEND_COMMAND (0X33);
 LCD_SEND_COMMAND (0X32);
 LCD_SEND_COMMAND (0X28);
 LCD_SEND_COMMAND (0X0F);
 LCD_SEND_COMMAND (0X01);
}
```