Freescale's Path Finder

In connection with Freescale's Smart Car Competition

Team XLNC
Thapar University
Patiala

Contents

Acknowledgment

At first we would like to thank Freescale for providing us with the opportunity of implementing and improving our knowledge about microcontrollers and providing us the platform to work on your HC12 series microcontrollers

We would also like to thank Thapar University for their support at each and every phase of the Freescale Smart Car Competition.

We are also very thankful to Mr. Vishal P Arora and Mr. Sumit Miglani for supervising our work and helping us with our problems that we faced time to time.

Team XLNC

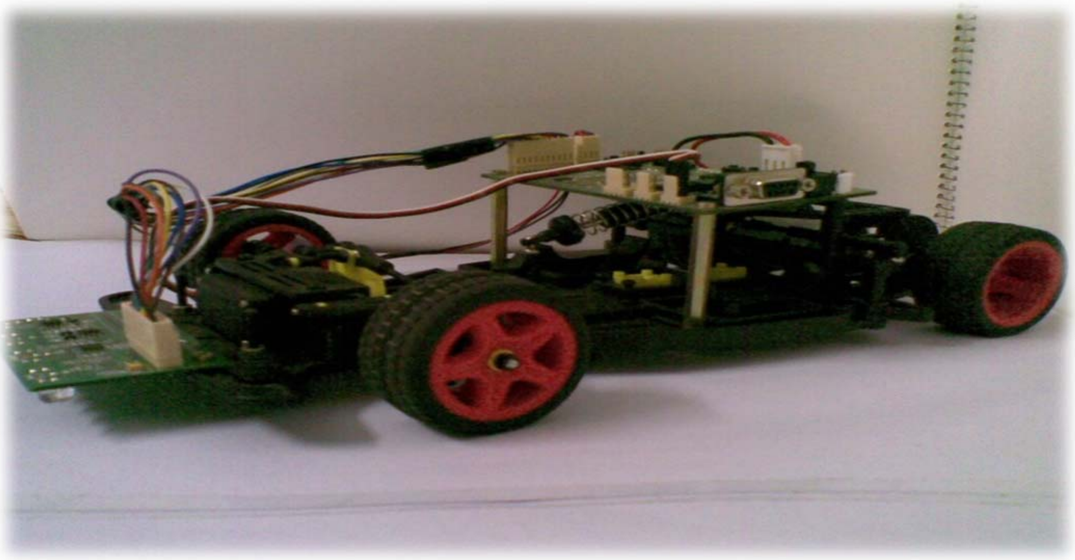
Abstract

In this report we discuss the SMART CAR, which was designed to participate in the competition path follower meeting the rules and parameters set for the competition. Besides the hardware necessary to sense, he adapted a speed control based on PID technique, implemented in a microcontroller HC12 family.
The Smart car is a car line follower consists of the electronics with a microprocessor HC12 family, which is responsible for controlling the car through the interpretation of the signals received from IR sensors, which will track a black line marked on a white track or platform, that road must be traveled in the shortest possible time, in addition to turn and climb a particular slope.

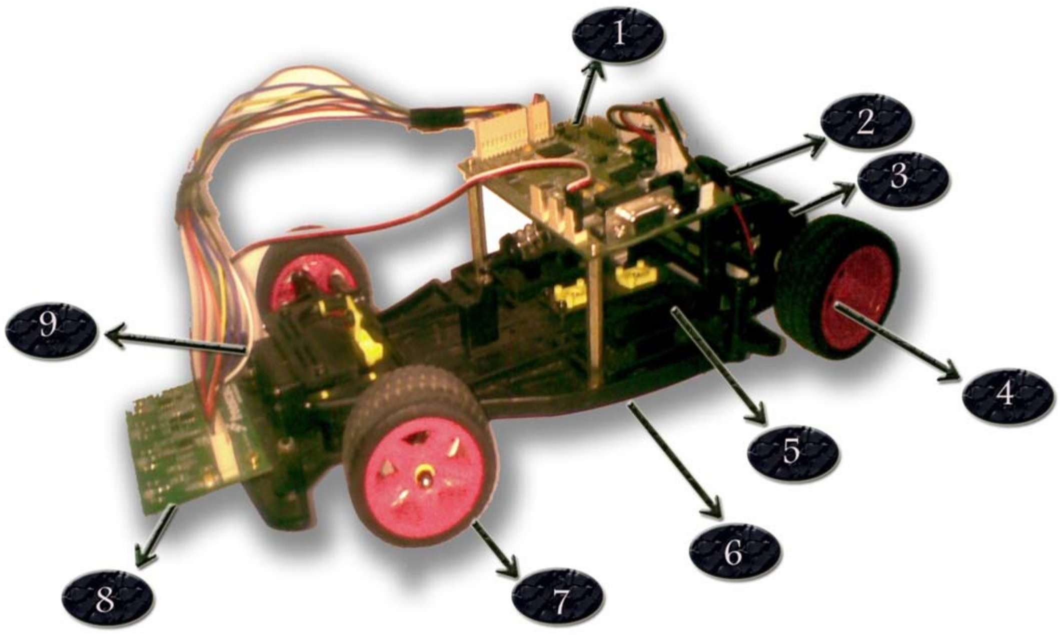
Introduction

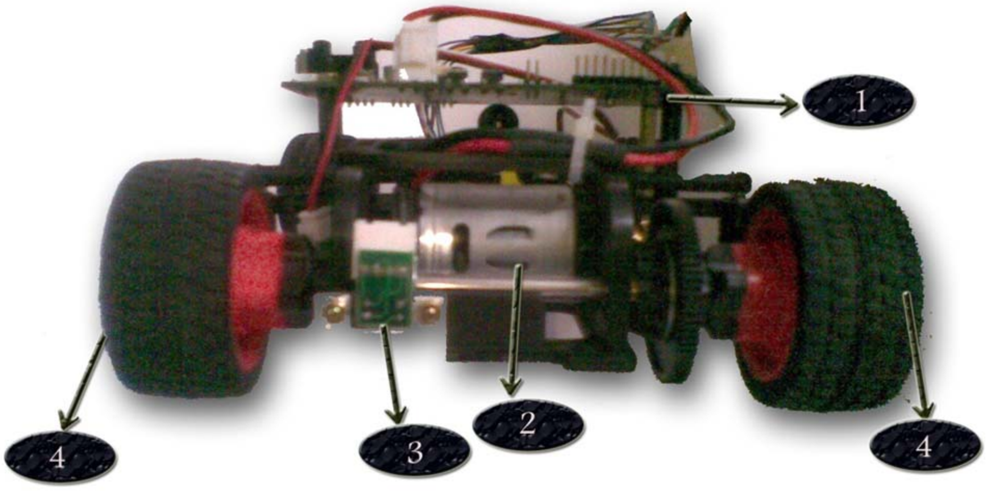
The path follower was made in connection with the Freescale's smart car race -2010. The robot automatically detects its path (in this case a black line) and follow it by positioning itself on the track as per the programme written in it.the black line is sensed by the array of 8 sensors positioned in the front of the robot. The sensor data is then changed into the digital form by the ATD (analog to digital) cconverters. The ATD data is then processed which gives us the current position of the robot with respect to the track and then a command is given to the servo motor connected to the front wheels to guide the car on the specified track. The robot is so designed that it can change its speed according to the sharpness of the turns. It slows down more on sharp turns and runs at a fast speed when it senses a straight path.

The robot uses freescale's mc9s12xd256 microcontroller which has HC12 architecture. The robot is a rear wheel drive driven by a single DC motor and the direction of the robot is controlled by a servo motor which controls the turning of the front wheels. Moreover the robot is provided with a pulse encoder which gives the exact value of the rounds of tires per unit time and can be used effectively to control the speed of the robot or the PWM duty of the DC motor not depending on the battery life remaining .



Major components





The Major components of the smart car are listed below:

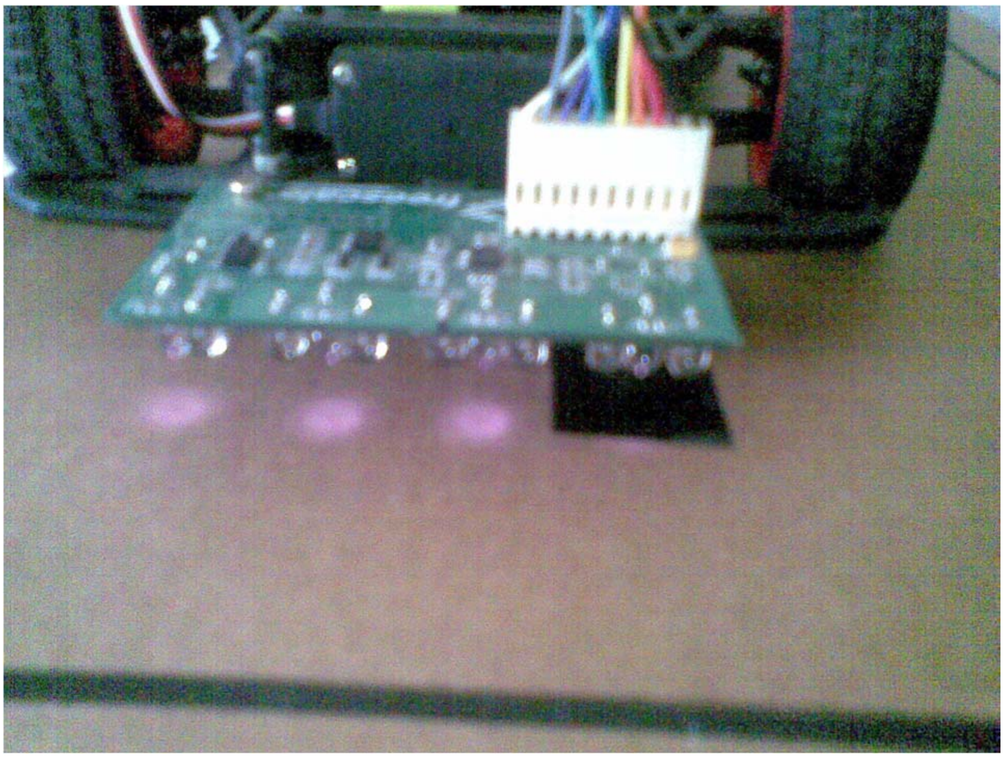
1. Circuit board
2. DC Motor
3. Shaft Encoder
4. Rear wheels
5. Battery holder
6. Chasis
7. Front wheels
8. IR Sensors
9. Servo Motor

Control System

The control system that we used in our car is a PID controller or proportional–integral–derivative controller which is a control loop feedback mechanism. It calculates an “error” value as the difference between a measured process variable and a desired setpoint. In this case the measured process variable is the current position of the black line and the setpoint is the desired position of the black line.

As we are quite new to the fields of robotics so we decided to implement the default algorithm initially and will improve it eventually. The position of any sensor is 2*ATD* sensor index. So the position of the 7 sensors in use is given by the following table

Sensor	Sensor 6	Sensor 5	Sensor 4	Sensor 3	Sensor 2	Sensor 1	Sensor 0
Position	3072	2560	2048	1536	1024	512	0



As observed in the mobile phone camera, only three sensors are on the black line(25 mm). So the position algorithm should contain only these three 'sensitive zones'. Although at first we decided to implement the algorithm without any change, but still logic in mind forced us to reconsider the clumsy and error prone bubble sort algorithm. So we finally come with a very simple algorithm that just finds the max value and assumes that the sensor lies in the middle of the black line

Some significant errors in position were as in case of following data

3	0	0	0	0	51	231

Let the index of sensor having highest ATD value be X, substantially decreasing the time required when sorting with bubble sort and getting the three indexes.

And the position of the black line is given by :

Position of black line = (X)*2*ATD resolution level +(compensated reading of index(X+1) – compensated reading of index (X-1)) [if 0<X<6]

Position of black line = compensated reading of index (1) [if X=0]

Position of black line =3076- compensated reading of index (6)

ATD resolution is 256

The desired position of the black line or the setpoint is: (position of highest index – position of lowest index)/2 = (3072 - 0)/2 = 1536

And the error = (position of center line – setpoint)

The controller attempts to minimize the error by adjusting the process control inputs. But the PID parameters used in the calculation must be tuned according to the nature of the system. The PID controller calculation involves two separate parameters,the proportional, the integral and derivative denoted *P, I and D*. The *proportional* value determines the reaction to the current error, the *integral* value determines the reaction based on the sum of recent errors, and the *derivative* value determines the reaction based on the rate at which the error has been changing .By tuning the three constants in the PID controller algorithm, the controller can provide control action to the servo motor and direct it for an effective path following.

The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). Hence:

$$\mathrm{MV(t)} = P_{\mathrm{out}} + I_{\mathrm{out}} + D_{\mathrm{out}}$$

Proportional term

The proportional term makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain.

The proportional term is given by:

$$P_{out} = K_p \, e(t)$$

K_p = proportional gain

e(t) = error at present

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable . In contrast, a small gain results in a small output response to a large input error, and a less responsive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

Integral term

The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output. The magnitude of the contribution of the integral term to the overall control action is determined by the integral gain, K_i.

The integral term is given by:

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

where

I_{out}: Integral term of output
K_i: Integral gain, a tuning parameter
e: Error = *Setpoint – process variable*
t: Time or instantaneous time (the present)
τ: a dummy integration variable

The integral term (when added to the proportional term) accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the setpoint value (cross over the setpoint and then create a deviation in the other direction).

Derivative term

The rate of change of the process error is calculated by determining the slope of the error over time (i.e., its first derivative with respect to time) and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term (sometimes called *rate*) to the overall control action is termed the derivative gain, K_d.

The derivative term is given by:

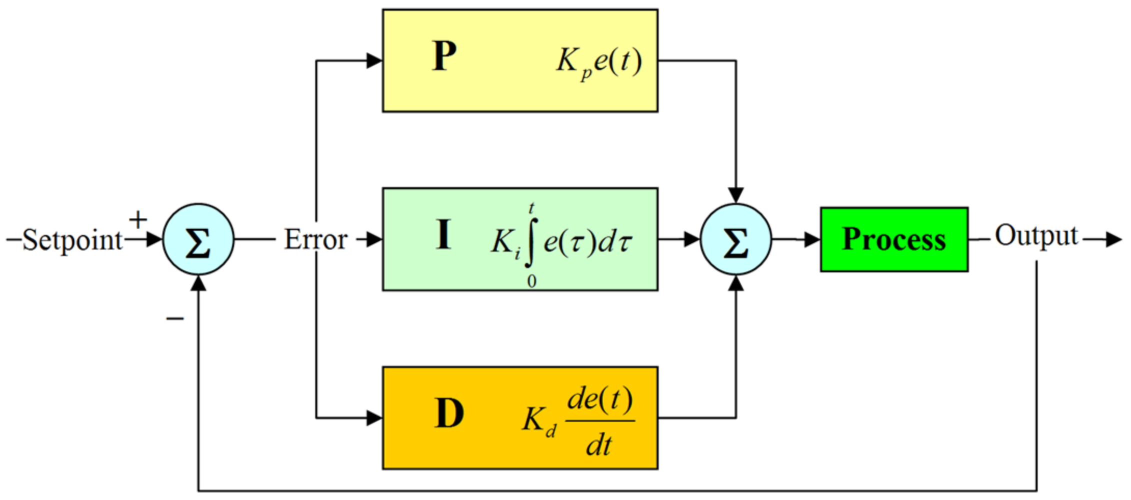
$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

where

D_{out}: Derivative term of output

K_d: Derivative gain, a tuning parameter
e: Error = $SP - PV$
t: Time or instantaneous time (the present)

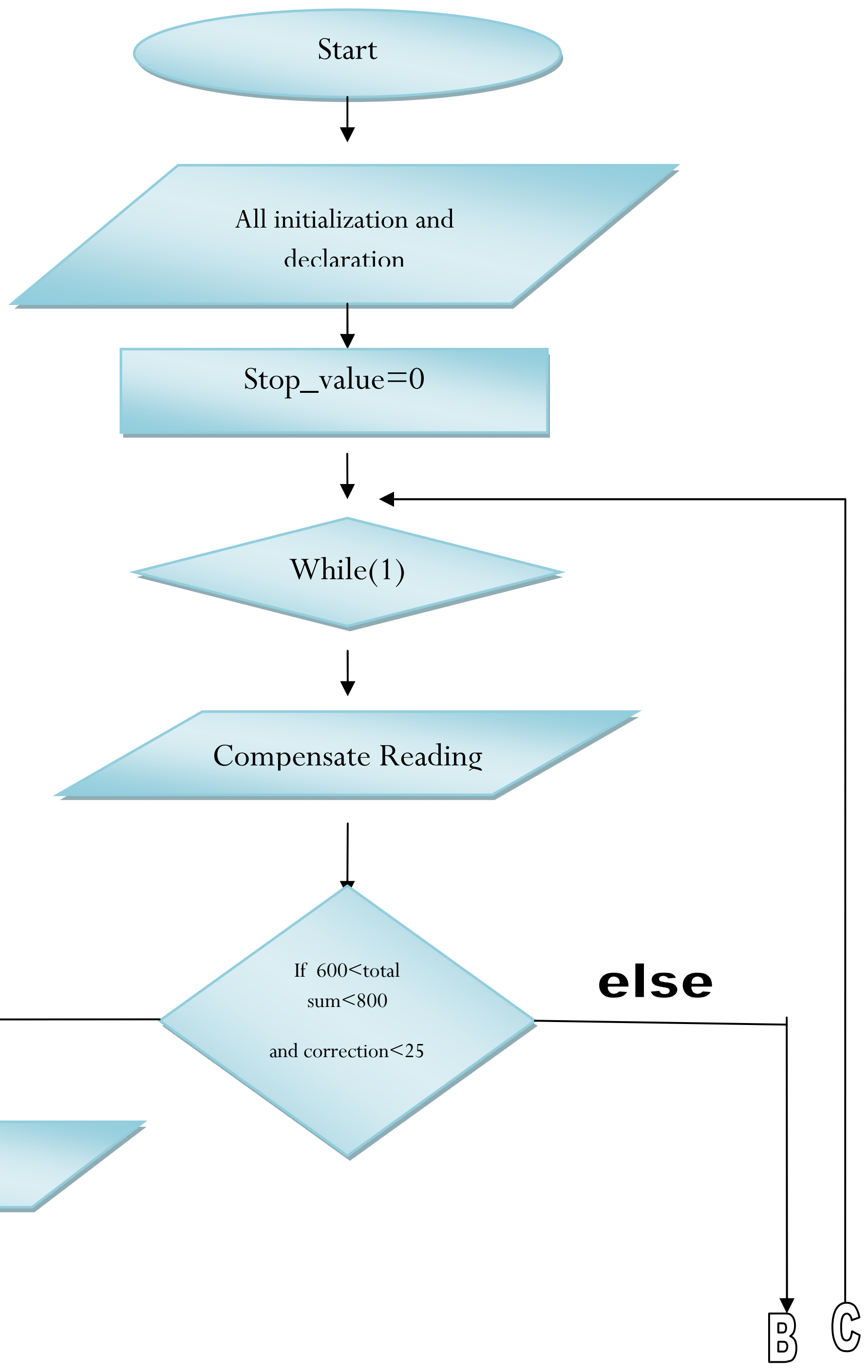
The derivative term slows the rate of change of the controller output and this effect is most noticeable close to the controller setpoint. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability.

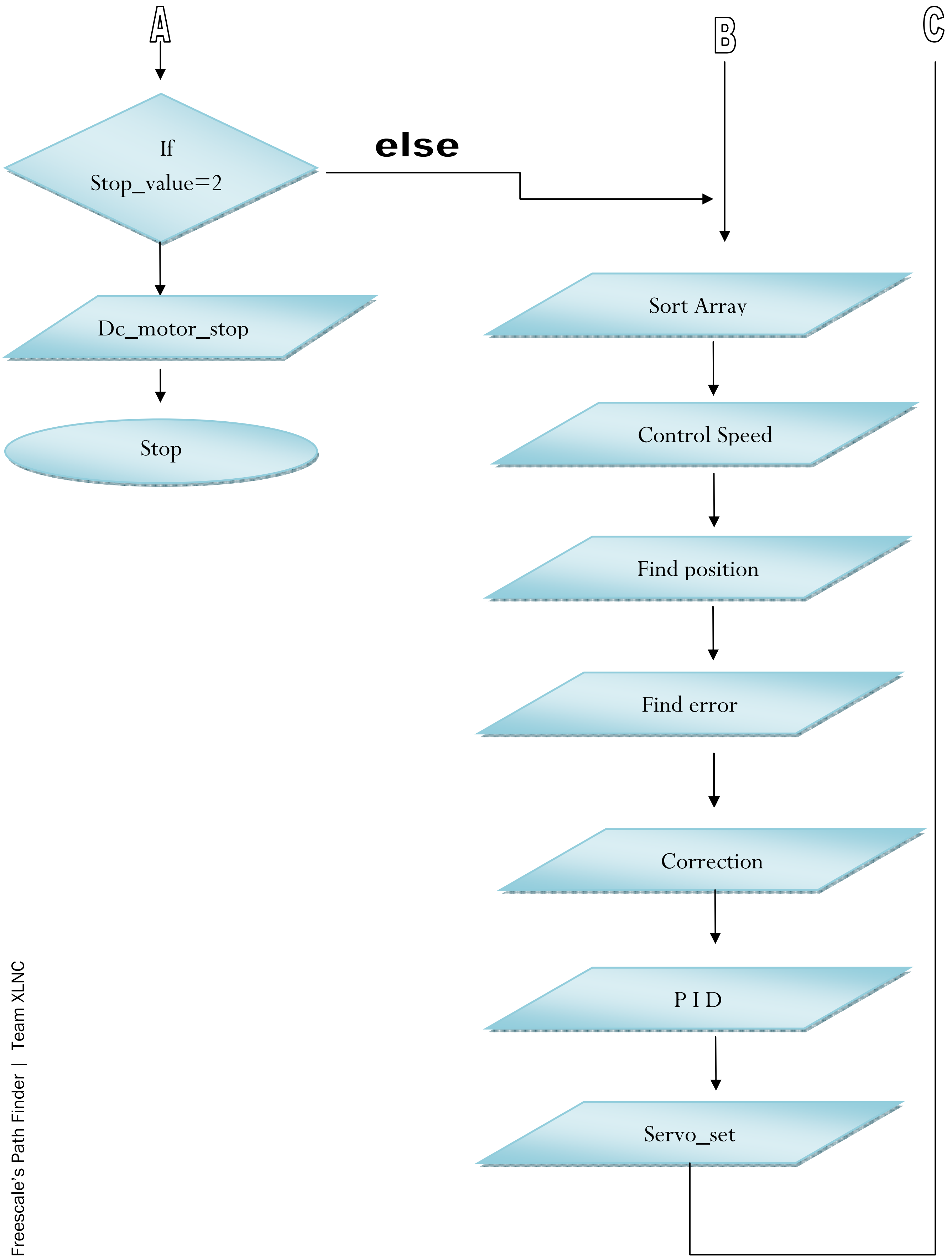


Software Design

The programme starts with initialization process where it initializes all its variables and macros. The a variable stop_value is created and initiated to 0 . This variable limits the robot to run only two laps and then stop. After that the programme goes into an infinite while loop including the main body of the programme. The while loop starts with compensation of the sensor values obtained from the ATD converter. Then the sum of these compensated values is taken and decision is made depending on the total sum of the compensated values. If the total sum is greater that 600 and less than 800 then there may be a start stop line or a cross on track. If the then there is a start stop point and the stop_value is incremented and for stop_vaue=2 the dc motor stops and the programme ends as the robot have already ran 2 laps other wise the programme gives the control to else loop of the previous if condition. The ATD values are stored in an array out of which the sensor showing the maximum reading is slected and its index is returned . And according to the index of the sensor with highest value speed of the robot is set and the position of the car is calculated as per the formula given above. Then the error from the setpoint is calculated which is corrected by the PID control and the servo motor is given the desired duty cycle to guide the robot on its path. For convenient tuning of the car the K_p, K_i and K_d values were stored in EEPROM and it can be changed with the help switch 1 and switch 2. The PD controller was also implemented in the pulse encoder section and it helped to keep the speed of the car constant.

The detailed flow chart of the software is given below :


Start
All initialization and declaration
Stop_value=0
While(1)
Compensate Reading
If 600<total sum<800 and correction<25
else
Stop_value++
A
B
C


A
B
C
If
Stop_value=2
else
Dc_motor_stop
Stop
Sort Array
Control Speed
Find position
Find error
Correction
P I D
Servo_set

The Source Code

- The main file

```
#include <hidef.h>

 #include "mc9s12xd256.h"

 #include "global.h"

 #include "atd.h"

 #include "compensation.h"

 #include "dcmotor.h"

 #include "uart.h"

 #include "servomotor.h"

 #include "eeprom.h"

 #include "pid.h"

 #define MAX_CORRECTION 305

 #define MIN_CORRECTION -305

 #define GUARD_VAL 80

 #define START_CROSS         650

 #define CUT_OFF_LR       250

 #define START_NON90_SURE_START      1450

 #define ALL_WHITE           20

 #define ITERM_LIMIT 20000

 #define SET_POINT 1500

 #define KP_DC 3

 void testmode(void);

 unsigned char com_data[8];

 void main(void)

 {

unsigned char index;

unsigned int position=0;

int error=0,perror=0;

//Declaration related to PID control systems.
```

```
int iterm=0;

unsigned int servo_output=1500;

float correction;

int i_stop;

unsigned int sumleft,sumright,totalsum;

int start_count=0;  //variable to be removed..

// dc pid control

unsigned int pulse_count=0;   //16 bit regsiter for accumulator

int error_dc=0;     //error can be +ve or -ve

int correction_dc=0;         //+ve or -ve

unsigned char speed_dc = 80;

unsigned char count=0;

Leds_and_Switches_Init();

IEE1_Init();

//EEPROM_reset();

EEPROM_read();

ISR_init();

ATD_init();

pulse_counter_init();

dc_motor_init();                    // Current pwm duty is 85

servo_init();

//uart0_9600_init();           //To be removed in the final version of the software.


while(START)

{

    //uart0_tx_string(" ::::::::::::: I'm now in main Loop !! ::::::::::: ");

    //This function compensates the sen_data array and puts the result into com_data array

    compensate_value(sen_data,com_data);

    // Placing it here, will increase the lag between value read and action taken.. hence

    // making the system unresponsive ...

    ATD0CTL5_SCAN=0;         //This will start a new conversion ....

    //code for PID of dc motor
```

```
   if(count >=30)
   {
        count=0;
        pulse_count =  PACN10;
        PACN10=0;    //resets the pulse counter
        error_dc = setpoint - pulse_count;
        correction_dc =(int) ( KP_DC*error_dc);
        if(speed_dc + correction_dc>=200)
        {
          speed_dc=200;
        }
        else if(speed_dc + correction_dc <=0)
        {
          speed_dc =0;
        }
        else
        {
          speed_dc +=correction_dc;
        }
        dc_motor_speed(speed_dc);
   }
   count ++;
//START STOP DETEECTION
//119      15          116         221         134         52          189
    sumleft = com_data[0]+com_data[1]+com_data[2];
    sumright = com_data[4]+com_data[5]+com_data[6];
    totalsum = sumleft+sumright+com_data[3];
     if(totalsum>START_CROSS)    //either CROSS or start ...
    {
          //SUM OF SENSORS IS LARGE OS EITHER A CROSS OR START..
          if(totalsum<START_NON90_SURE_START)
        {
            //SO NOT ALL SENSORS ARE NOT ON BLACK .....
            // SO EITHER WE HAVE A NON 90 CROSS OR START
            if((sumright-sumleft) < CUT_OFF_LR && sumright-sumleft>0 || (sumleft-sumright) < CUT_OFF_LR && sumright-sumleft>0)
            {
                // SUM LEFT AND SUM RIGHT IS ALMOST SAME..
                //SURELY WE HAVE START ..
                start_count ++;
                LED1=~LED1;
                if(start_count>delay_dc)
                {
                    dc_motor_stop();
                }
            }
            else
            {
                // SUM OF LEFT AND RIGHT IS NOT SAME
                // SO THIS IS SURELY A NON 90 CROSS
                start_count=0;
                LED2=~LED2;
            }
        }
        else
        {
            // QUITE A LARGE SUM OF ......
            //SURE 90 CROSS
            //DO NOTHING
            start_count=0;
            LED3=~LED3
        }
    }
    else if(totalsum<ALL_WHITE)
```

```
{

        i_stop++;

        if(i_stop>=100)     //STOP TIME

        {

                dc_motor_stop();

        }

}


else

{

        start_count=0;

        index=sort_array(com_data);

        i_stop=0;                          //To be removed in the final versions of the code


        /*

        if(com_data[index]<15)

        {

                //aa white

        }

        */


        //GUARD 1

        if(index ==0 && com_data[index]<GUARD_VAL)

        {

                servo_set (1500+MIN_CORRECTION);

                LED3=LED_ON;

        }


        //GUARD 2

        else if(index==6 && com_data[index]<GUARD_VAL)

        {
```

```
        servo_set (1500+MAX_CORRECTION);

        LED3=LED_ON;

    }

    else

    {

        //LED3=LED_OFF;

        position =  (index * 512) + (com_data[index+1]-com_data[index-1]) ;

        perror=error;      //stores previous error

        error = 1536 - position;

        //PID

        /*

        Done 10000 from 30000 because iterm once saturates becuse unhanlable .

        */

        // limits bound on iterm.

        if (iterm>=ITERM_LIMIT)       //32767 is max limit of int

        {

            //To buffer the fast responses of propotional control, we have the integral control.

            iterm = ITERM_LIMIT;

            //LED4=~LED4;

        }

        else if (iterm<=-ITERM_LIMIT)       //32767 is max limit of int

        {

            //To buffer the fast responses of propotional control, we have the integral control.

            iterm = -ITERM_LIMIT;

            //LED4=~LED4;

        }


        else

        {

            iterm += error/20;

        }

        correction = (kp*error)  +  (ki *iterm) + (kd*(perror-error));
```

```
                    //anti wind up circutary

                    if (correction >= MAX_CORRECTION)

                    {

                            correction = MAX_CORRECTION;

                    }

                    if (correction <= MIN_CORRECTION)

                    {

                            correction = MIN_CORRECTION;

                    }

                    // 0     1       …     5       6

                    // -ve correction means -ve error (position-6) ie turn right

                    // +ve correction means +ve error (position-0) ie turn left.

                    servo_output = SET_POINT-(unsigned int)correction;

                    servo_set (servo_output);

            }

        }

    }

}
```

Assosiated header files

atd.h

```
void ATD_init()
{
        // New code to implement interrupts

        //ATD0CTL2 registers
        ATD0CTL2_ADPU=1;   //ATD Power Up

        ATD0CTL2_AFFC=1;   //The flag CCF clears automatically if we read the appropriate result register

        //No need to use it. It will decrease the response of the system
        //ATD0CTL2_AFFC=0;   //The flag clears when we read the ATDSTAT1

        ATD0CTL2_ASCIE=1;  //The interrupt will be called when ASCIF is 1. Clearing this flag is necessary.

        //ATD0CTL3
        //ATD0CTL3_S8C=1;          // 8 adc channels sequence

        ATD0CTL3_S8C=0;          // 7 ADC channels sequence
        ATD0CTL3_S4C=1;          // from 0-7
        ATD0CTL3_S2C=1;          //
```

```
                ATD0CTL3_S1C=1;          //

                  //ATD0CTL4
                  ATD0CTL4_SRES8=1;  //8 bit resoluation

                   //Fastest possible settings already
                  ATD0CTL4_SMP=0;     //2 clock cycles.
                  ATD0CTL4_PRS=0;    //divinde BUS clock by 2. However the MAX BUS clock is 4mhz and min is 2mhx

                  //ATD0CTL5
                  ATD0CTL5_DJM=0;     //Left Justified
                  ATD0CTL5_MULT=1;
                  ATD0CTL5_SCAN=0;    //single conversion mode
                  //ATD0CTL5_SCAN=1;    //CONTINUOUS mode
  }
```

Compensation.h

```
/*Compensation ratio = 256.0 / (Black value – White value)*/

float com_ratio[7]=

{

1.75342, 2.0645, 2.39252, 2.3486, 2.4151, 2.0983, 1.9845

};

// MAX VALUES FOR w_val. SEPARATE w_val COMPUTATION ALGORITHM

unsigned char w_val[7]=

{

11, 10, 10, 10, 10, 10, 10

};



// MIN VALUES FOR b_val. SEPARATE b_val COMPUTATION ALGORITHM

unsigned char b_val[7] =

{

157, 134, 117, 119, 116, 132, 139

};

void compensate_value(unsigned char* val,unsigned char* com_val)

{

        int i;

        for(i=0;i<6;i++)

        {

                if(*(val+i)<w_val[i])        //lower bound
                {

                        *(com_val+i)=0;

                }


                else if (*(val+i)> (b_val[i]))    //higher bound

                {

                        *(com_val+i)=255;

                }

                else

                {

                        // This will round off the value properly ….

                        *(com_val+i) = (unsigned char)(((*(val+i) - w_val[i]) * com_ratio[i]) + 0.5);

                }

        }

}

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

Function - sort_array

Input - One compensated array

Output - Null

**********************************************

unsigned char sort_array(unsigned char* a)

{

        int i,index;

        index=0;

        for(i=1;i<7;i++)

        {

                if(*(a+index) < *(a+i))

                {

                        index=i;

                }

        }
```

```
                return index;

            }
```

dcmotor.h

```
        /*

      This function sets the motor speed by the use of PWM.

      The argument to this function val 0<val<=200.

       */

       void dc_motor_speed(unsigned char val)

            {

        //PORTE_PE3=0;        //Disabling the Motor driver

        PWMDTY0 = val;

        PWMDTY1 = 0;

        //PORTE_PE3=0;        //Re-enabling the motor driver

}

void dc_motor_init()

{

  //Drives the motors in forward motion..

       DDRE_DDRE2=1;          // Port E pin 2 & 3 set to output

       DDRE_DDRE3=1;

       DDRP_DDRP0=1;

       DDRP_DDRP1=1;

       PORTE_PE2=0;

       PORTE_PE3=1;

       PWMPOL_PPOL0=1;          //PWM pulse High at begining of Period

       PWMPOL_PPOL1=1;          //PWM pulse High at begining of Period

       PWMCLK_PCLK0=1;          // clock SA as clock source for PWM

       PWMPRCLK =0x00;          //clock A = 2MHz  clockB = 2MHz

       PWMSCLA =5;          //clock  SA = clock A / (2 * 5) = 200KHz

       PWMSCLB =5;          //clock  SB = clock B / (2 * 5) = 200KHz

       PWMPER0 = 200;          // PWM Period          1KHz

       PWMPER1 = 200;          // PWM Period          1KHz
       PWMDTY0 = 70;

       PWMDTY1 = 0;


       PWME_PWME0=1;          //PWM channel 0 Enable

       PWME_PWME1=1;          //PWM channel 1 Enable

}


void dc_motor_stop()

{

 // Stop Motor

       PORTE_PE3=0;


       PTP_PTP0=0;

       PTP_PTP1=0;

       PWME_PWME0=0;          //PWM channel disable

       PWME_PWME1=0;

}


void pulse_counter_init()

{

       PBCTL_PBEN=1;          //16 bit pulse accumulator enable bit (cascading two 8 bit pulse accumulator)


       TCTL4_EDG0B=1;      //configure the edge detector circuits

       TCTL4_EDG0A=0;      //10 is capture on falling edges only..


       DLYCT_DLY0=1;       //introduces delay of 1024 bus clock cycles..

       DLYCT_DLY1=1;

       PACN10=0;   //Sets the data register to 0.

}
```

eeprom.h

```
//Global definations

#define ON 1;

#define OFF 0;

#define LED_ON 0

#define LED_OFF 1

#define LED1 PTT_PTT4

#define LED2 PTT_PTT5

#define LED3 PTT_PTT6

#define LED4 PTT_PTT7

#define ADT_LED  LED1

#define BUTTON_LED  LED2

#define SW2  PTP_PTP5

#define SW3  PTP_PTP7

// EEPROMs VETOR ADDRESS

#define EEPROM_KP ((IEE1_TAddress)0x13F000)

#define EEPROM_KI ((IEE1_TAddress)0x13F010)     // 2 byte above

#define EEPROM_KD ((IEE1_TAddress)0x13F020)     // 2 byte above

#define EEPROM_DELAYDC ((IEE1_TAddress)0x13F030)     //Need 1 byte address..

#define EEPROM_SETPOINT ((IEE1_TAddress)0x13F038)     //Need 1 byte address..

#define EEPROM_START ((IEE1_TAddress)0x13F040)     //Need 1 byte address..

//Global Variables

union long_to_float

{

    unsigned long long_val;

    float float_val;

};

unsigned char START=1;     //Variable that starts the car..

// SWITCH interrupt CONTROL EEPROM VARIABLES. ...

float kp=0.0,ki=0.0,kd=0.0;

unsigned char delay_dc=3;   //variable related to START_CROSS detection

unsigned char setpoint =1;   //set point is +ve and generally small values..  VAR RELATED TO PULSE counter

void Leds_and_Switches_Init()
```

```
{
        DDRT=0xF0;              // Setting Port T pin 4,5,6,7 as output
        DDRP_DDRP5=0;               //Port P Pin 5 & 7 set to input
        DDRP_DDRP7=0;
        PERP_PERP5=1;             //Port P Pin 5 & 7 Pullup Enable
        PERP_PERP7=1;

        LED1=LED_OFF;
        LED2=LED_OFF;
        LED3=LED_OFF;
        LED4=LED_OFF;

}
void Delay(unsigned char a)
{
        unsigned int i,j;
        for(j=1;j<=a;j++)          //Delay
        for(i=0;i<=60000;i++);
}
```

pid.h

```
#define __DI()  { asm sei; }     /* Disable global interrupts  */
#define __EI()  { asm cli; }     /* Enable global interrupts */
#define ISR(x) __interrupt void x(void)
// variable assignments...
unsigned char sen_data[8]=
{0, 0, 0,0, 0, 0, 0,0};
/* float akps=0.11,skis=0.005;        //0.220
*/
unsigned char flag_sm=0;
char selection = -1;
int i=0;
```

```
void ISR_init()
{
        __EI()    //enable interrupts
          /* IRQCR: IRQEN=0 */
        IRQCR &= (unsigned char)~64;
        /* PIEP: PIEP7=1,PIEP6=0,PIEP5=1,PIEP4=0,PIEP3=0,PIEP2=0,PIEP1=0,PIEP0=0 */
        PIEP = 160;
        //new code ends
}
ISR (isr_default)
{}
ISR (isrVatd0)
{
        //Clearing ASCIF is necessary as this will not cause reinitiating
        //uart0_tx_newline();
        //uart0_tx_newline();
        //uart0_tx_newline();
        //uart0_tx_string(" :::: Interrupt ADC complete haha !! :::");
        //uart0_tx_newline();
        //uart0_tx_newline();
        //uart0_tx_newline();
        // See how many reads are req to clear the reg. It is conjectured that atleast one may req..
        //ADT_LED=~ADT_LED;
        //Actual code
        sen_data[0] =  ATD0DR0H;            //This will clear the Appropriate FLAG .....
        sen_data[1] =  ATD0DR1H;
        sen_data[2] =  ATD0DR2H;
        sen_data[3] =  ATD0DR3H;
        sen_data[4] =  ATD0DR4H;
        sen_data[5] =  ATD0DR5H;
        sen_data[6] =  ATD0DR6H;
        sen_data[7] =  ATD0DR7H;
```

```
}
ISR (simar)
{
    if(flag_sm!=0)          //flag_SELECTION_MADE
      {
        //selection has been made    ((( :::: button5 is INC   and button7 is DEC :::: )))
          switch(selection)
          {
              case 0:
              {
                  //START
                  if(PIFP&(1<<5))                    // (Active Low)  on Button 2 (PIN 5) has occured
                  {
                      START=~START;
                      // No need to toggle the LED as feedback will be from from the motors..
                  }
                  else
                  {
                      //writes to EEPROM..
                      unsigned char return_val;
                      return_val = IEE1_SetByte(EEPROM_START,START);
                      if(return_val == ERR_OK)
                      {
                          LED1=LED_OFF;
                          Delay(2);
                          LED1=LED_ON;
                      }
                  }
              }
              case 1:
              {
```

```
        // KI
        union long_to_float simar_read;
        unsigned char return_val;
        if(PIFP&(1<<5))                    // (Active Low)  on Button 2 (PIN 5) has occured (INC)
        {
            //uart0_tx_string(" Interrupt !! ... BUTTON 2  :::: ::::: ");
            //uart0_tx_newline();
            // Clears flag. Writes logical one to the 5h
            //PIFP = 32;
            ki=ki+0.000250;
        }
        else
        {
            //uart0_tx_string(" Interrupt !! ... BUTTON 3  :::: ::::: ");
            //uart0_tx_newline();
            //PIFP = 128;                    // Clears flag. Writes logical one to the 7h
            ki=ki-0.000250;
        }

        //PIFP = 160;                    // Clears flag. Writes logical one to the 5h pin n 7th pin..
        // This code writes the new value of KP to EEPROM
        simar_read.float_val = ki;
        return_val = IEE1_SetLong(EEPROM_KI,simar_read.long_val);
        if(return_val == ERR_OK)
        {
            LED1=LED_OFF;
            Delay(2);
            LED1=LED_ON;
        }
        //ATD0CTL5_SCAN=0;
        break;
    }
```

```
case 3:

{

    //KP

    union long_to_float simar_read;

    unsigned char return_val;

    if(PIFP&(1<<5))                    // (Active Low)  on Button 2 (PIN 5) has occured

    {

        //uart0_tx_string(" Interrupt !! ... BUTTON 2  :::: ::::: ");

        //uart0_tx_newline();

        // Clears flag. Writes logical one to the 5h

        //PIFP = 32;

        kp=kp+0.0025;

    }

    else

    {

        //uart0_tx_string(" Interrupt !! ... BUTTON 3  :::: ::::: ");

        //uart0_tx_newline();

        //PIFP = 128;                    // Clears flag. Writes logical one to the 7h

        kp=kp-0.0025;

    }

    //PIFP = 160;                    // Clears flag. Writes logical one to the 5h pin n 7th pin..

    // This code writes the new value of KP to EEPROM

    simar_read.float_val = kp;

    return_val = IEE1_SetLong(EEPROM_KP,simar_read.long_val);

    if(return_val == ERR_OK)

    {

        LED2=LED_OFF;

        Delay(2);

        LED2=LED_ON;

    }

    break;
```

```
}

case 4:

{

        // QNAN EEPROM RESET

        if(PIFP&(1<<7))     //Button 3          //decrement

        {

                EEPROM_reset();

        }

        break;

}

case 5:                //LED 3

{

        // DELAY_DC

        unsigned char return_val;

        if(PIFP&(1<<5))                         // (Active Low)  on Button 2 (PIN 5) has occured

        {

                //uart0_tx_string(" Interrupt !! ... BUTTON 2  :::: ::::: ");

                //uart0_tx_newline();

                // Clears flag. Writes logical one to the 5h

                //PIFP = 32;

                delay_dc++;

        }

        else

        {

                //uart0_tx_string(" Interrupt !! ... BUTTON 3  :::: ::::: ");

                //uart0_tx_newline();

                //PIFP = 128;                         // Clears flag. Writes logical one to the 7h

                delay_dc--;

        }

        //PIFP = 160;                         // Clears flag. Writes logical one to the 5h pin n 7th pin..

        // This code writes the new value of KP to EEPROM

        return_val = IEE1_SetByte(EEPROM_DELAYDC,delay_dc);
```

```
                if(return_val == ERR_OK)
                {
                    LED3=LED_OFF;
                    Delay(2);
                    LED3=LED_ON;
                }
                break;
            }
            case 6:             //LED2
            {
                // SETPOINT
                unsigned char return_val;
                if(PIFP&(1<<5))                 // (Active Low)  on Button 2 (PIN 5) has occured
                {
                    //uart0_tx_string(" Interrupt !! ... BUTTON 2  :::: ::::: ");
                    //uart0_tx_newline();
                    // Clears flag. Writes logical one to the 5h
                    //PIFP = 32;
                    setpoint++;
                }
                else
                {
                    //uart0_tx_string(" Interrupt !! ... BUTTON 3  :::: ::::: ");
                    //uart0_tx_newline();
                    //PIFP = 128;             // Clears flag. Writes logical one to the 7h
                    setpoint--;
                }

                //PIFP = 160;             // Clears flag. Writes logical one to the 5h pin n 7th pin..
                // This code writes the new value of KP to EEPROM
                return_val = IEE1_SetByte(EEPROM_SETPOINT,setpoint);
```

```
        if(return_val == ERR_OK)

        {

            LED2=LED_OFF;

            Delay(2);

            LED2=LED_ON;

        }

        break;

    }

    case 7:          //black compensattion

    {

        START=0;          //STOPs the car.

        if(PIFP&(1<<5))                  // (Active Low)  on Button 2 (PIN 5) has occured

        {

            //Start a new conversion

            for(i=0;i<=6;i++)

            {

                if(b_val[i]<sen_data[i])

                {

                    //write new max value to b_val[i]

                    b_val[i]=sen_data[i];

                }

            }


            ATD0CTL5_SCAN=0;        //This will start a new conversion ....

        }


        else

        {

            //make it final

            //write the data to eeprom

            //test it on computer before writing this code...

        }
```

```
                        break;
                    }
                    case 8:         //white compensattion
                    {
                        START=0;        //STOPs the car.
                        if(PIFP&(1<<5))             // (Active Low)  on Button 2 (PIN 5) has occured
                        {
                            //Start a new conversion
                            for(i=0;i<=6;i++)
                            {
                                if(w_val[i]<sen_data[i])
                                {
                                    //write new max value to b_val[i]
                                    w_val[i]=sen_data[i];
                                }
                            }
                            ATD0CTL5_SCAN=0;        //This will start a new conversion ….
                        }
                        else
                        {
                            //make it final
                            //write the data to eeprom
                            //test it on computer before writing this code…
                        }
                        break;
                    }
                }
        }
        else
        {
            if(PIFP&(1<<5))             // MENU SCROLLER
```

```
{
    if(selection>=5)    //Rotating menu..
    {
        selection=0;
    }
    else
    {
        selection++;
    }
    switch(selection)
    {
        case 0:                          //LED1     //start
        {
            LED1=LED_ON;
            LED2=LED_OFF;
            LED3=LED_OFF;
            LED4=LED_OFF;
            break;
        }
        case 1:                 //LED2                  ki
        {
            LED1=LED_OFF;
            LED2=LED_ON;
            LED3=LED_OFF;
            LED4=LED_OFF;
            break;
        }
        case 2:   //                        LED3           kp
        {
            LED1=LED_OFF;
            LED2=LED_OFF;
            LED3=LED_ON;
        LED4=LED_OFF;

        break;

}

case 3:         //kd

{

        LED1=LED_OFF;

        LED2=LED_OFF;

        LED3=LED_OFF;

        LED4=LED_ON;

        break;

}

case 4:     //QNAN                  LED4     ToDo change in reset

{

        //DONT CHANGE ITS INDEX NO ... EEMPROM_RESET HAS TO BE CHANGES

        LED1=LED_OFF;

        LED2=LED_OFF;

        LED3=LED_ON;

        LED4=LED_OFF;

        break;

}

case 5:   //bounce back mechanism      LED2     delay_dc

{

        LED1=LED_OFF;

        LED2=LED_ON;

        LED3=LED_OFF;

        LED4=LED_OFF;

        break;

}

case 6:   //bounce back mechanism      LED1             setpoint

{

        LED1=LED_ON;
```

```
                LED2=LED_OFF;

                LED3=LED_OFF;

                LED4=LED_OFF;

            }

            case 7:   //bounce back mechanism      LED2              black   compensation

            {

                LED1=LED_OFF;

                LED2=LED_ON;

                LED3=LED_OFF;

                LED4=LED_OFF;


                //start ADT conversion

                ATD0CTL5_SCAN=0;        //This will start a new conversion ....

                break;

                //TAKE THE PRECAUTION OF KEEPING THE CAR ON THE TRACK RIGHT BEFORE

                //SELECTING THE MENU ....

            }

            case 8:   //bounce back mechanism      LED3              white  compensation

            {

                LED1=LED_OFF;

                LED2=LED_OFF;

                LED3=LED_ON;

                LED4=LED_OFF;

                //start ADT conversion

                ATD0CTL5_SCAN=0;        //This will start a new conversion ....

                break;

                //TAKE THE PRECAUTION OF KEEPING THE CAR ON THE TRACK RIGHT BEFORE

                //SELECTING THE MENU ....

            }

        }

    }

    else
```

```
                {
                        //7th button is menu SELECTOR
                        flag_sm=1;
                }
        }
        Delay(1);
        PIFP = 160;                     // Clears flag. Writes logical one to the 5h pin n 7th pin..
        Delay(1);
}


/* Interrupt vector table */
/* ISR prototype */
typedef void (*near tIsrFunc)(void);
#ifndef UNASSIGNED_ISR
  #define UNASSIGNED_ISR isr_default   /* unassigned interrupt service routine */
#endif
const tIsrFunc _InterruptVectorTable[] @0xFF10 = { /* Interrupt vector table */
  /* ISR name                              No.  Address Pri XGATE Name         Description */
  UNASSIGNED_ISR,                    /* 0x08  0xFF10   -   -    ivVsi          unused by PE */
  isrVatd0,                     /* 0x69  0xFFD2   1   no   ivVatd0       used by PE */
  UNASSIGNED_ISR,                    /* 0x6A  0xFFD4   1   no   ivVsci1       unused by PE */
  UNASSIGNED_ISR,                    /* 0x6B  0xFFD6   1   no   ivVsci0       unused by PE */
  UNASSIGNED_ISR,                    /* 0x6C  0xFFD8   1   no   ivVspi0       unused by PE */
  UNASSIGNED_ISR,                    /* 0x6D  0xFFDA   1   no   ivVtimpaie    unused by PE */
  UNASSIGNED_ISR,                    /* 0x6E  0xFFDC   1   no   ivVtimpaaovf  unused by PE */
  UNASSIGNED_ISR,                    /* 0x6F  0xFFDE   1   no   ivVtimovf     unused by PE */
  UNASSIGNED_ISR,                    /* 0x70  0xFFE0   1   no   ivVtimch7     unused by PE */
};          //NOT COMPLETE
```

servomotor.h

```
void servo_centre()
```

```
{
            //          Values of registers is 1415
             PWMDTY2 = 0x05;            // Pulse Width 1.5ms: Center Position
//           PWMDTY3 = 0x87;            //1.415ms
             // Value of register os 1500….
             PWMDTY3 = 0xDC;             //1.5ms
}
void servo_init()
{
      PWMCTL_CON23=1;
      PWME_PWME2=1;             //PWM channel 2 Enable
      PWME_PWME3=1;             //PWM channel 3 Enable
      PWMPOL_PPOL2=1;            //PWM pulse High at begining of Period
      PWMPOL_PPOL3=1;            //PWM pulse High at begining of Period
      PWMCLK_PCLK2=1;            // clock SA as clock source for PWM
      PWMCLK_PCLK3=1;            // clock SB as clock source for PWM
      PWMPRCLK =0x00;           //clock A = 2MHz  clockB = 2MHz
      // Time of this clock pulse is 1 us
      PWMSCLA =1;             //clock  SA = clock A / (2 * 1) = 1MHz
      PWMSCLB =1;             //clock  SB = clock B / (2 * 1) = 1MHz
      // Value of the register as PWMPER2-PWMPER3 is 20,000.
      // So the value of period of the flipping is 20ms
      PWMPER2 = 0x4E;           // PWM Period 20ms        50Hz
      PWMPER3 = 0x20;           // PWM Period 20ms        50Hz
    servo_centre();
}

void servo_left()
{
       //   Values of register is 1100
       PWMDTY2 = 0x04;            //Pulse Width: Left Position
       PWMDTY3 = 0x6A;            //1.1ms
}

void servo_right()

{

        //   Values of register is 1700

        PWMDTY2 = 0x06;              //Pulse Width 1.7 ms: Right Position

        PWMDTY3 = 0xA4;

}

void servo_set(unsigned int a)

{

        //   a should range somewhere in 1100 to 1700

        PWMDTY2 = (a>>8);

        a=a<<8;

        a=a>>8;

                PWMDTY3 = a;

}
```

Development tool and debugging process

As debugging in embedded system is a very important part, we tried to put our maximum effort here.

We created software like Data Logger when we felt like exporting sensor values to excel to study them better using mathematical tool there. We plotted graphs using the sensor position data and identified certain very significant loopholes in the current position finding algorithm. The quantized nature of the default position finding algorithm drove us towards introducing some other better version but unfortunately, time did not allowed us to do so. However the software did helped us a lot in understanding the sensor values, finding mix, max and in the end proved to be a boon to us.

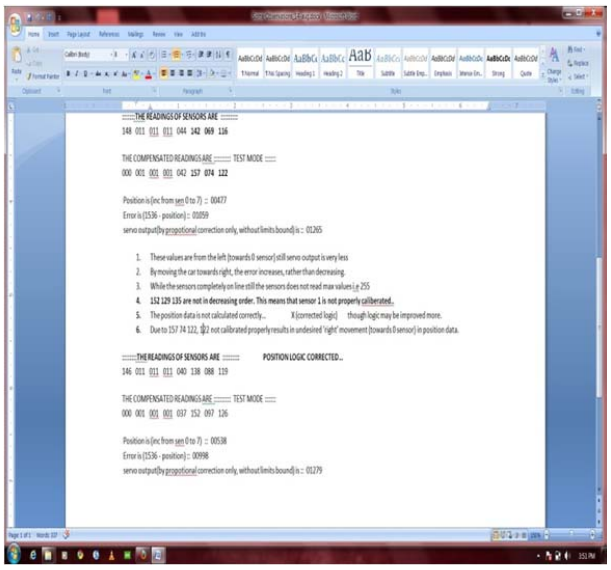
The main problem with the Data Logger was that, we were not able to see the data in real time and also the software is not so portable. As need is the force to creation and innovation of ideas, further exploration towards the end, we encountered code warrior virtualization tool. So we had another powerful tool and it helped us to better debug the software.

- Codewarrior
- Freescale's student development module
- Data Logger – made using visual basic
- Codewarriors's virtual tool

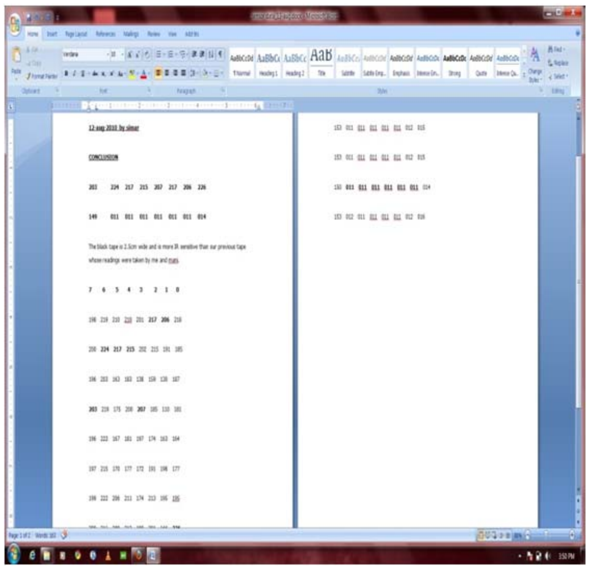
While the codewarrior was provided to us by Freescale itself. The data logger software was made exclusively for logging the sensor data coming from the ATD converter into a spreadsheet from where the desired functions on the data can be implemented.

Starting with simple algorithm to run the robot on straight path, the code became more and more complex to run the robot effectively on an oval track. Further improvements involve increasing the speed of the robot and implementing a better control system to minimize the oscillations and maximize the speed .

SCREENSHOTS:



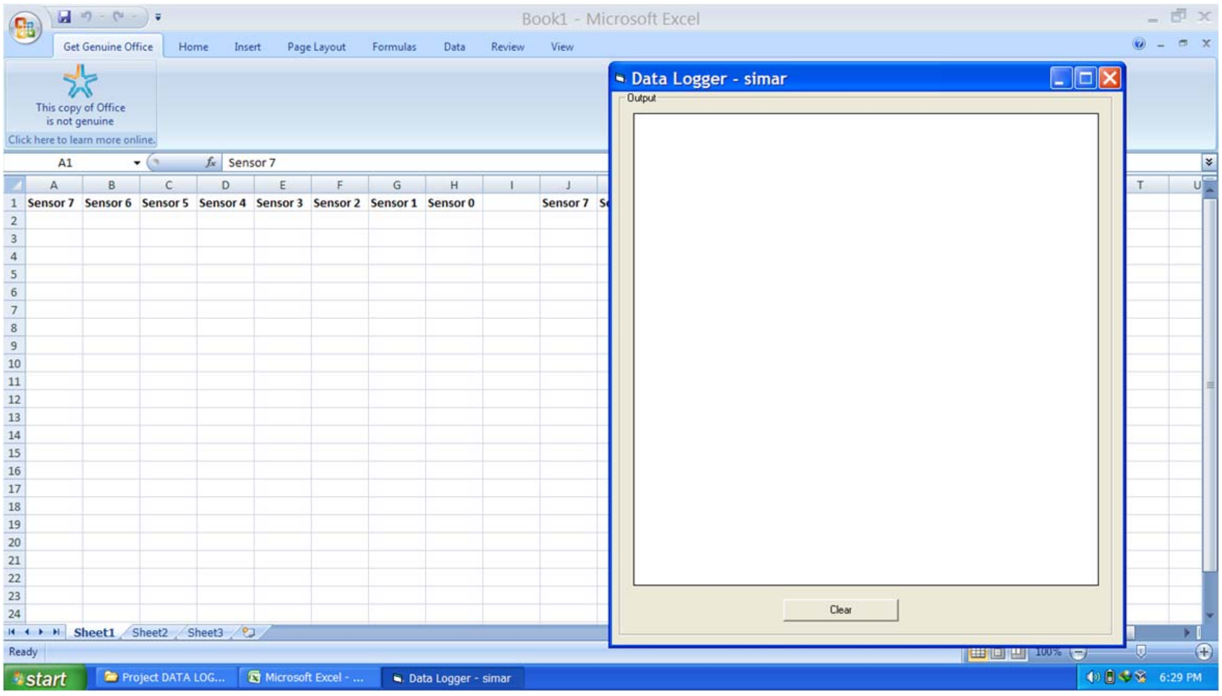
```
::::::THE READINGS OF SENSORS ARE ::::::
148  011  011  011  044  142  069  116

THE COMPENSATED READINGS ARE :::::: TEST MODE ::::
000  001  001  001  042  157  074  122

Position is (inc from sen 0 to 7) :: 00477
Error is (1536 - position) :: 01059
servo output(by propotional correction only, without limits bound) is :: 01265

   1.  These values are from the left (towards 0 sensor) still servo output is very less
   2.  By moving the car towards right, the error increases, rather than decreasing.
   3.  While the sensors completely on line still the sensors does not read max values i.e 255
   4.  152 129 135 are not in decreasing order. This means that sensor 1 is not properly caliberated..
   5.  The position data is not calculated correctly...       X (corrected logic)    though logic may be improved more.
   6.  Due to 157 74 122, 122 not calibrated properly results in undesired 'right' movement (towards 0 sensor) in position data.

::::::THE READINGS OF SENSORS ARE ::::::        POSITION LOGIC CORRECTED...
146  011  011  011  040  138  088  119

THE COMPENSATED READINGS ARE :::::: TEST MODE ::::
000  001  001  001  037  152  097  126

Position is (inc from sen 0 to 7) :: 00538
Error is (1536 - position) :: 00998
servo output(by propotional correction only, without limits bound) is :: 01279
```



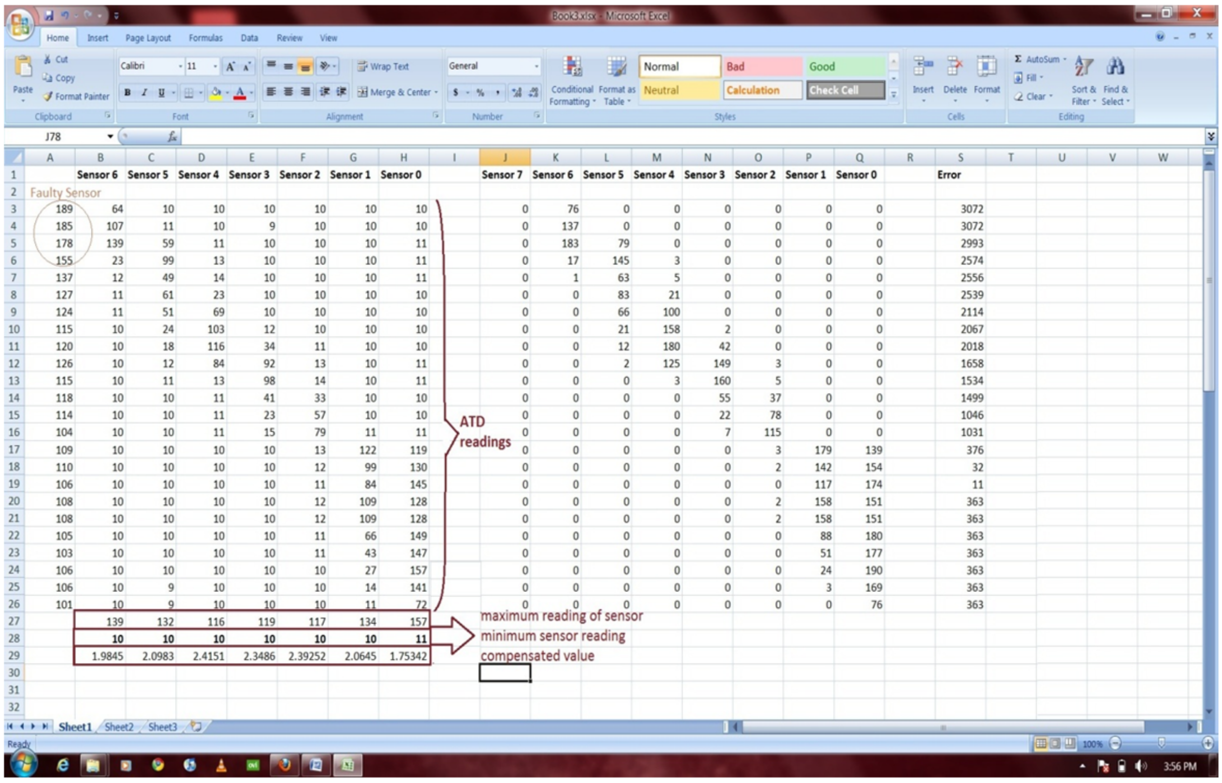
```
12-aug-2010 by simar

CONCLUSION

203     224  217  215  207  217  206  226

149     011  001  001  001  001  001  014

The black tape is 2.5cm wide and is more IR sensitive than our previous tape
whose readings were taken by me and mani.

7   6   5   4   3   2   1   0

198 219 210 210 201 217 206 218

200 224 217 215 202 215 191 185

196 203 163 183 138 159 128 167

203 219 175 208 207 185 133 181

196 223 167 181 197 174 163 164

197 215 170 177 172 191 198 177

198 222 206 211 174 213 195 195
```

```
153 011 011 011 011 011 012 015

153 011 011 011 011 011 012 015

150 011 011 011 011 011 011 014

153 012 011 011 011 011 012 016
```

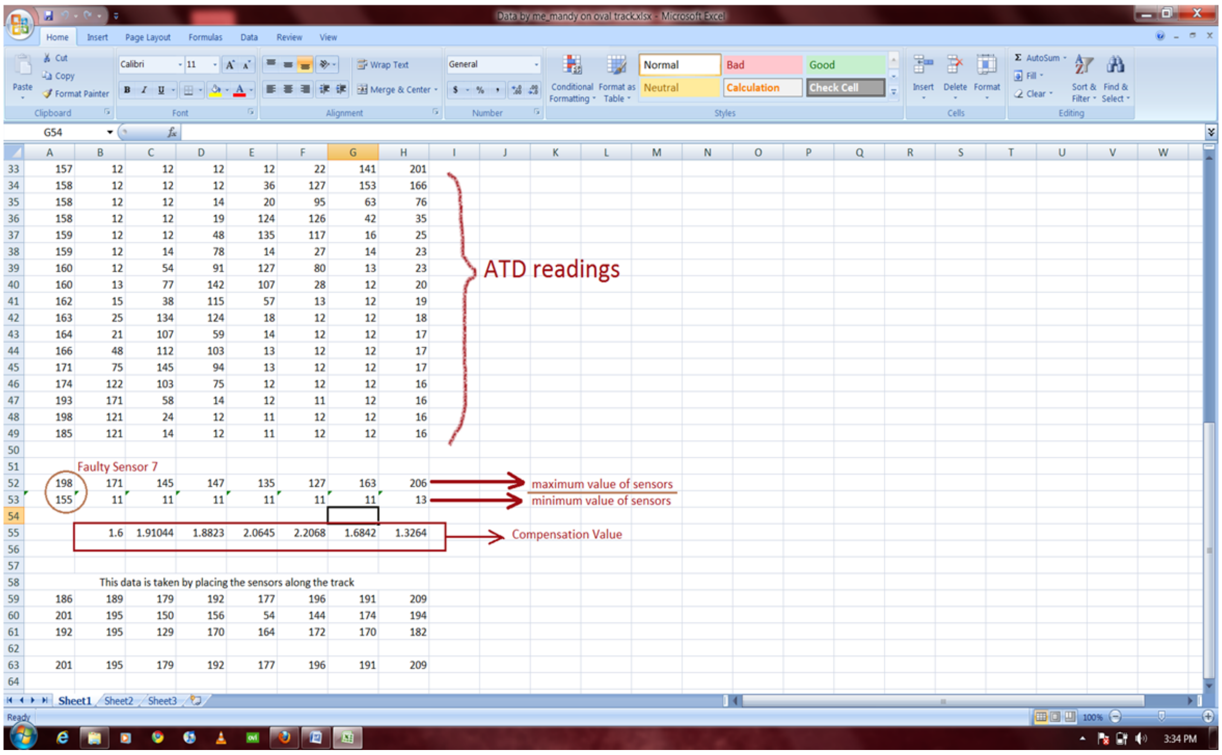
Observing the values from the serial port and deriving conclusions



Data Logger software interface

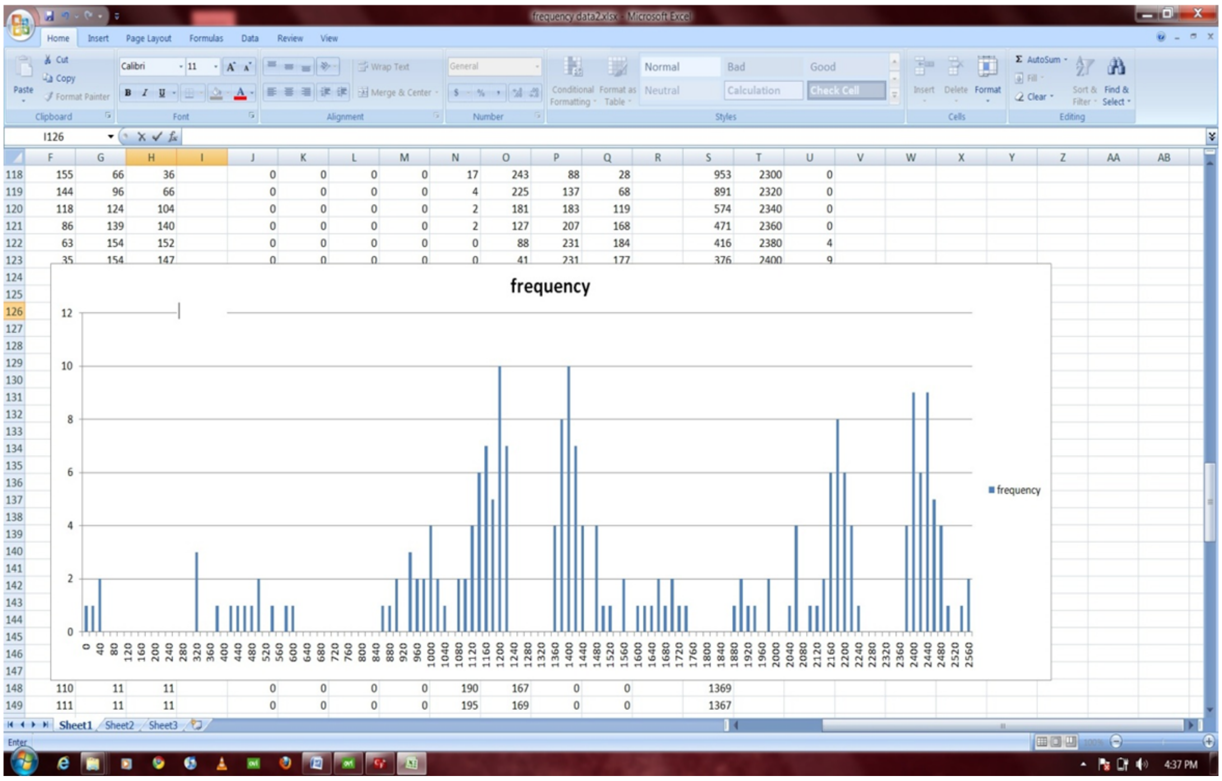


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1		Sensor 6	Sensor 5	Sensor 4	Sensor 3	Sensor 2	Sensor 1	Sensor 0		Sensor 7	Sensor 6	Sensor 5	Sensor 4	Sensor 3	Sensor 2	Sensor 1	Sensor 0		Error
2	Faulty Sensor																		
3	189	64	10	10	10	10	10	10		0	76	0	0	0	0	0	0		3072
4	185	107	11	10	9	10	10	10		0	137	0	0	0	0	0	0		3072
5	178	139	59	11	10	10	10	11		0	183	79	0	0	0	0	0		2993
6	155	23	99	13	10	10	10	11		0	17	145	3	0	0	0	0		2574
7	137	12	49	14	10	10	10	11		0	1	63	5	0	0	0	0		2556
8	127	11	61	23	10	10	10	10		0	0	83	21	0	0	0	0		2539
9	124	11	51	69	10	10	10	10		0	0	66	100	0	0	0	0		2114
10	115	10	24	103	12	10	10	10		0	0	21	158	2	0	0	0		2067
11	120	10	18	116	34	11	10	10		0	0	12	180	42	0	0	0		2018
12	126	10	12	84	92	13	10	11		0	0	2	125	149	3	0	0		1658
13	115	10	11	13	98	14	10	11		0	0	0	3	160	5	0	0		1534
14	118	10	10	11	41	33	10	10		0	0	0	0	55	37	0	0		1499
15	114	10	10	11	23	57	10	10		0	0	0	0	22	78	0	0		1046
16	104	10	10	11	15	79	11	11		0	0	0	0	7	115	0	0		1031
17	109	10	10	10	10	13	122	119		0	0	0	0	0	3	179	139		376
18	110	10	10	10	10	12	99	130		0	0	0	0	0	2	142	154		32
19	106	10	10	10	10	11	84	145		0	0	0	0	0	0	117	174		11
20	108	10	10	10	10	12	109	128		0	0	0	0	0	2	158	151		363
21	108	10	10	10	10	12	109	128		0	0	0	0	0	2	158	151		363
22	105	10	10	10	10	11	66	149		0	0	0	0	0	0	88	180		363
23	103	10	10	10	10	11	43	147		0	0	0	0	0	0	51	177		363
24	106	10	10	10	10	10	27	157		0	0	0	0	0	0	24	190		363
25	106	10	9	10	10	10	14	141		0	0	0	0	0	0	3	169		363
26	101	10	9	10	10	10	11	72		0	0	0	0	0	0	0	76		363
27		139	132	116	119	117	134	157											
28		10	10	10	10	10	10	11											
29		1.9845	2.0983	2.4151	2.3486	2.39252	2.0645	1.75342											

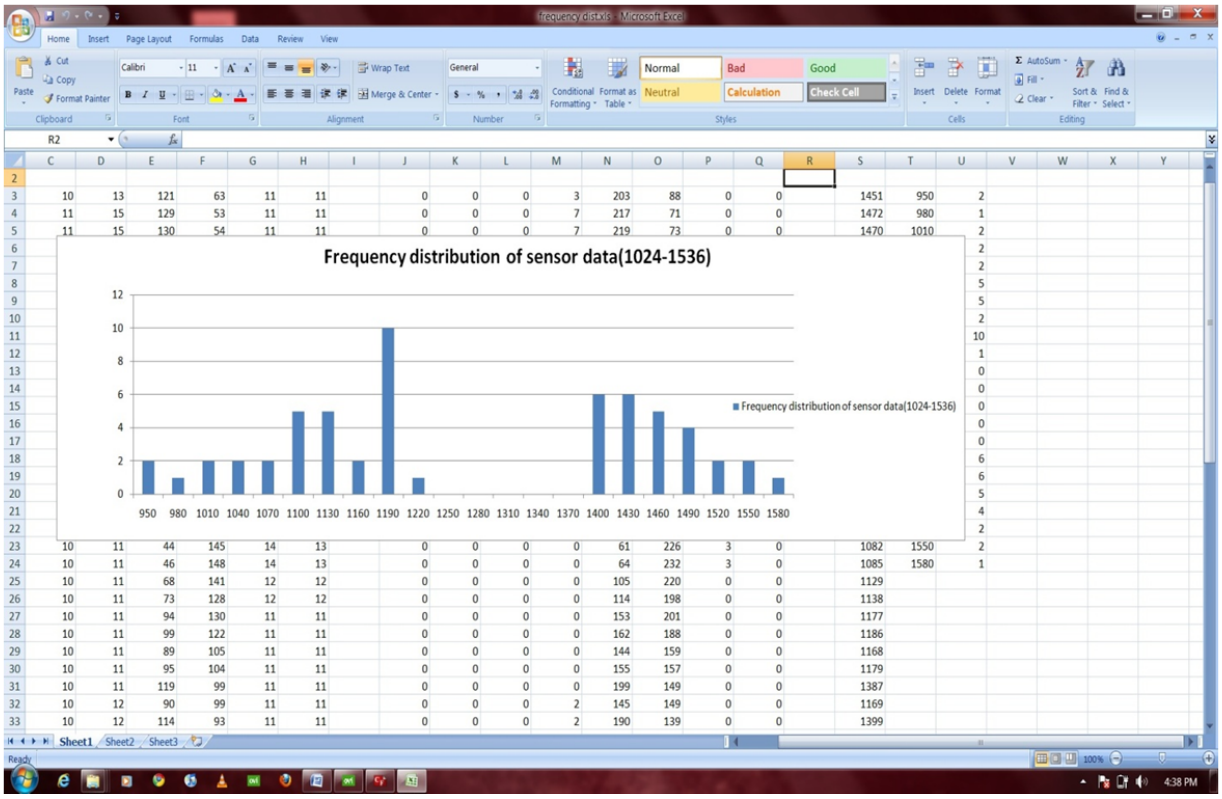


	A	B	C	D	E	F	G	H
33	157	12	12	12	12	22	141	201
34	158	12	12	12	36	127	153	166
35	158	12	12	14	20	95	63	76
36	158	12	12	19	124	126	42	35
37	159	12	12	48	135	117	16	25
38	159	12	14	78	14	27	14	23
39	160	12	54	91	127	80	13	23
40	160	13	77	142	107	28	12	20
41	162	15	38	115	57	13	12	19
42	163	25	134	124	18	12	12	18
43	164	21	107	59	14	12	12	17
44	166	48	112	103	13	12	12	17
45	171	75	145	94	13	12	12	17
46	174	122	103	75	12	12	12	16
47	193	171	58	14	12	11	12	16
48	198	121	24	12	11	12	12	16
49	185	121	14	12	11	12	12	16
50								
51		Faulty Sensor 7						
52	198	171	145	147	135	127	163	206
53	155	11	11	11	11	11	11	13
54								
55		1.6	1.91044	1.8823	2.0645	2.2068	1.6842	1.3264
56								
57								
58		This data is taken by placing the sensors along the track						
59	186	189	179	192	177	196	191	209
60	201	195	150	156	54	144	174	194
61	192	195	129	170	164	172	170	182
62								
63	201	195	179	192	177	196	191	209
64								

Date logged into a spreadsheet by data logger

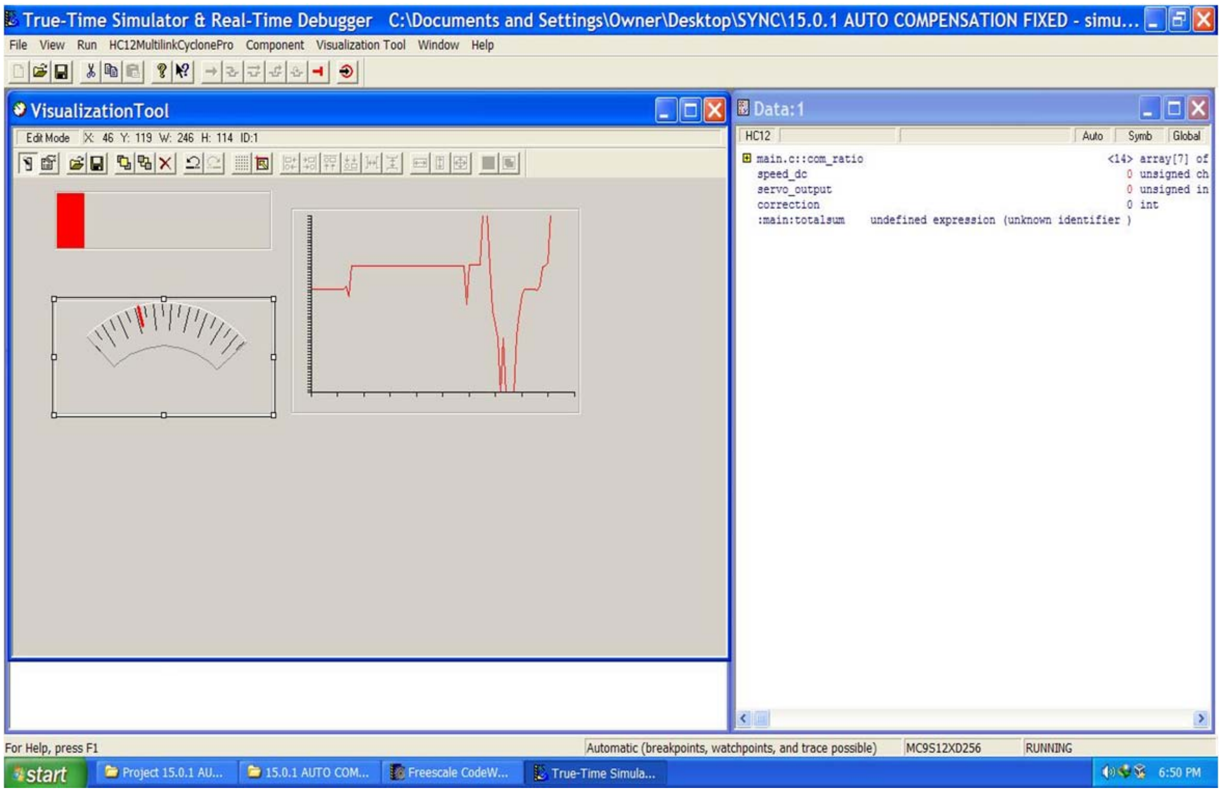


	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
118	155	66	36		0	0	0	0	17	243	88	28		953	2300	0
119	144	96	66		0	0	0	0	4	225	137	68		891	2320	0
120	118	124	104		0	0	0	0	2	181	183	119		574	2340	0
121	86	139	140		0	0	0	0	2	127	207	168		471	2360	0
122	63	154	152		0	0	0	0	0	88	231	184		416	2380	4
123	35	154	147		0	0	0	0	0	41	231	177		376	2400	9
148	110	11	11		0	0	0	0	190	167	0	0		1369		
149	111	11	11		0	0	0	0	195	169	0	0		1367		



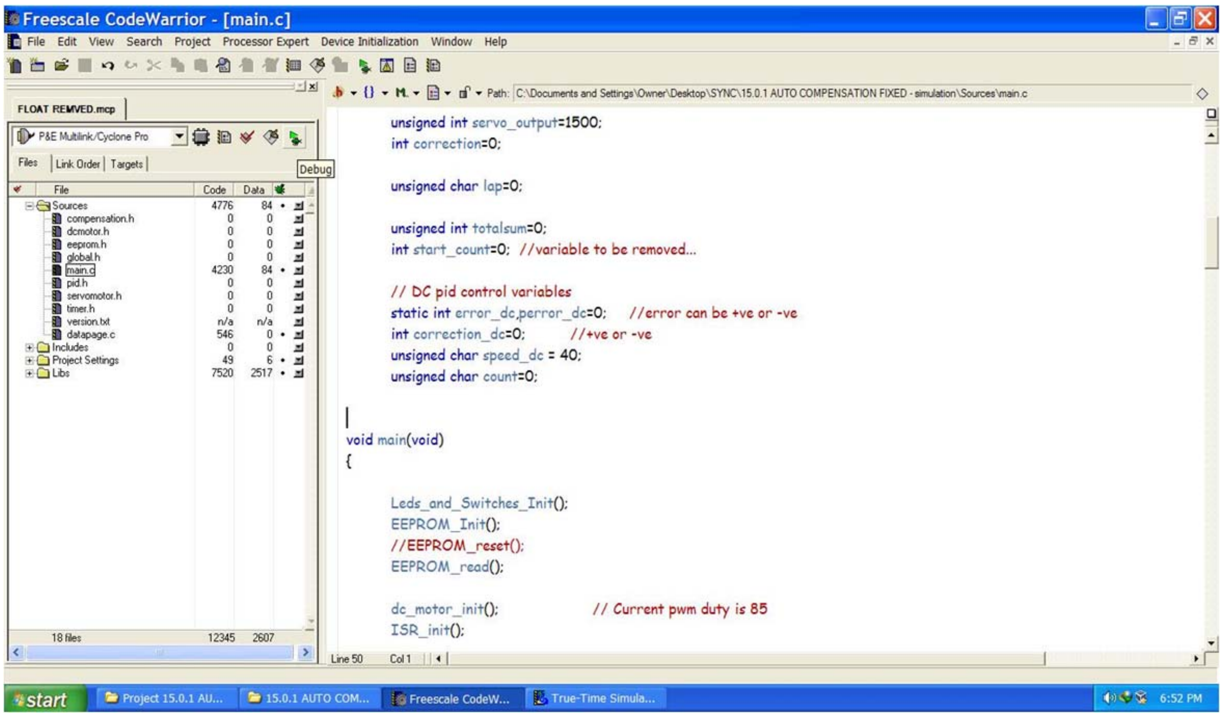
	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
2																			
3	10	13	121	63	11	11		0	0	0	3	203	88	0	0		1451	950	2
4	11	15	129	53	11	11		0	0	0	7	217	71	0	0		1472	980	1
5	11	15	130	54	11	11		0	0	0	7	219	73	0	0		1470	1010	2
6																			2
7																			2
8																			5
9																			5
10																			2
11																			10
12																			1
13																			0
14																			0
15																			0
16																			0
17																			0
18																			6
19																			6
20																			5
21																			4
22																			2
23	10	11	44	145	14	13		0	0	0	0	61	226	3	0		1082	1550	2
24	10	11	46	148	14	13		0	0	0	0	64	232	3	0		1085	1580	1
25	10	11	68	141	12	12		0	0	0	0	105	220	0	0		1129		
26	10	11	73	128	12	12		0	0	0	0	114	198	0	0		1138		
27	10	11	94	130	11	11		0	0	0	0	153	201	0	0		1177		
28	10	11	99	122	11	11		0	0	0	0	162	188	0	0		1186		
29	10	11	89	105	11	11		0	0	0	0	144	159	0	0		1168		
30	10	11	95	104	11	11		0	0	0	0	155	157	0	0		1179		
31	10	11	119	99	11	11		0	0	0	0	199	149	0	0		1387		
32	10	12	90	99	11	11		0	0	0	2	145	149	0	0		1169		
33	10	12	114	93	11	11		0	0	0	2	190	139	0	0		1399		

Drawing conclusions from the data logged



```
main.c::com_ratio                                    <14> array[7] of
speed_dc                                                0 unsigned ch
servo_output                                            0 unsigned in
correction                                              0 int
:main:totalsum    undefined expression (unknown identifier )
```

For Help, press F1   Automatic (breakpoints, watchpoints, and trace possible)   MC9S12XD256   RUNNING



Freescale CodeWarrior - [main.c]

FLOAT REMVED.mcp

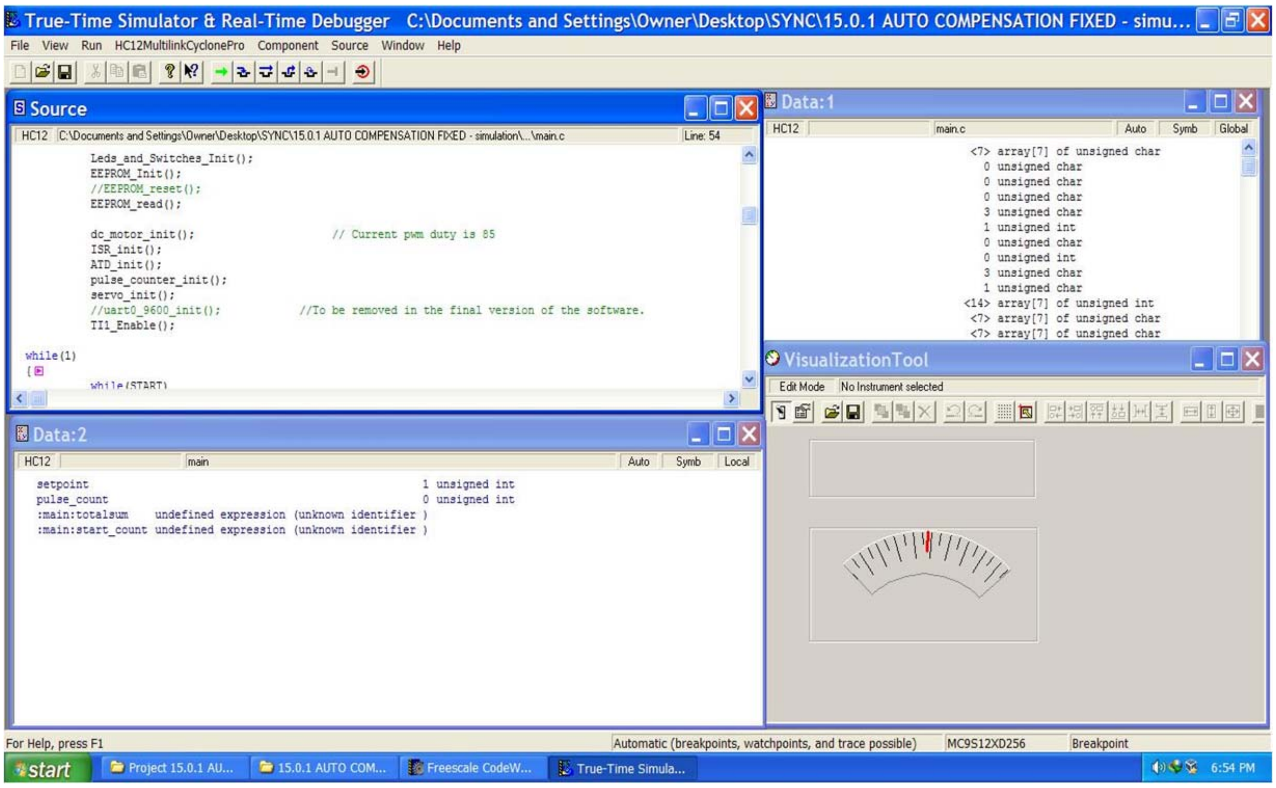
File	Code	Data
Sources	4776	84
compensation.h	0	0
dcmotor.h	0	0
eeprom.h	0	0
global.h	0	0
main.c	4230	84
pid.h	0	0
servomotor.h	0	0
timer.h	0	0
version.txt	n/a	n/a
datapage.c	546	0
Includes	0	0
Project Settings	49	6
Libs	7520	2517
18 files	12345	2607

Path: C:\Documents and Settings\Owner\Desktop\SYNC\15.0.1 AUTO COMPENSATION FIXED - simulation\Sources\main.c

```
        unsigned int servo_output=1500;
        int correction=0;

        unsigned char lap=0;

        unsigned int totalsum=0;
        int start_count=0;  //variable to be removed...

        // DC pid control variables
        static int error_dc,perror_dc=0;     //error can be +ve or -ve
        int correction_dc=0;          //+ve or -ve
        unsigned char speed_dc = 40;
        unsigned char count=0;


void main(void)
{

        Leds_and_Switches_Init();
        EEPROM_Init();
        //EEPROM_reset();
        EEPROM_read();

        dc_motor_init();                // Current pwm duty is 85
        ISR_init();
```

Line 50 Col 1

Codewarrior coding section



True-Time Simulator & Real-Time Debugger C:\Documents and Settings\Owner\Desktop\SYNC\15.0.1 AUTO COMPENSATION FIXED - simu...

Source — HC12 C:\Documents and Settings\Owner\Desktop\SYNC\15.0.1 AUTO COMPENSATION FIXED - simulation\...\main.c Line: 54

```
        Leds_and_Switches_Init();
        EEPROM_Init();
        //EEPROM_reset();
        EEPROM_read();

        dc_motor_init();                    // Current pwm duty is 85
        ISR_init();
        ATD_init();
        pulse_counter_init();
        servo_init();
        //uart0_9600_init();            //To be removed in the final version of the software.
        TI1_Enable();

 while(1)
 {

        while(START)
```

Data:1 — HC12 main.c Auto Symb Global

```
 <7> array[7] of unsigned char
   0 unsigned char
   0 unsigned char
   0 unsigned char
   3 unsigned char
   1 unsigned int
   0 unsigned char
   0 unsigned int
   3 unsigned char
   1 unsigned char
<14> array[7] of unsigned int
 <7> array[7] of unsigned char
 <7> array[7] of unsigned char
```

Data:2 — HC12 main Auto Symb Local

```
setpoint                                        1 unsigned int
pulse_count                                     0 unsigned int
:main:totalsum    undefined expression (unknown identifier )
:main:start_count undefined expression (unknown identifier )
```

VisualizationTool — Edit Mode No Instrument selected

For Help, press F1 — Automatic (breakpoints, watchpoints, and trace possible) — MC9S12XD256 — Breakpoint

Codewarrior debugging tool

Stages of debugging

The code was written and compiled in C using the codewarriors in 10 different stages:

STAGE 1: getting familiar with the servomotor

Stage 1 started with taking control over the servo motor. The servo motor was given PWM duty cycle using serial cable and by incrementing and decrementing the PWM duty cycle to get the value for rightmost and leftmost turn. The value was actually a 16 bit value stored in two registers by right and left shifting using the code:

```
void servo_set(unsigned int a)

{

        // a should range somewhere in 1100 to 1700

        PWMDTY2 = (a>>8);

        a=a<<8;

        a=a>>8;

                PWMDTY3 = a;

}
```

The leftmost value truned out to be : 1205

Rightmost value =1815

Central value = 1510

STAGE 2: compensating the sensor reading

Stage 2 included taking sensor values from the ATD converter through serial cable connection by the help of data logger. The sensor values were collected and calibrated.

STAGE 3: implementing PI controller

Stage 3 comprises of implementing PI control and running the robot with PI controller. The error values were checked in this stage to be right and according to the used algorithm.

STAGE 4: completing oval track

Stage 4 include first completely working code running the robot on an oval track. in this stage the robot ran around an oval track for the first time.

STAGE 5: implementing writing to EEPROM

In stage 5 writing on EEPROM was implementing so that the tuning can be done in real time and all the P and I values were strored in the EEPROM and they could be changed by using the buttons provided and auto compensation of sensor values was done.

STAGE 6: upgrading to PID controller

In this stage PI control was upgraded to PID controller.

STAGE 7: Programming to stop after 2 laps

in this stage the programme was provided with a code section to stop the car after 2 laps.

STAGE 8: Menu Interface

Stage 8 included making a menu interface to effectively tune the car and store the result in EEPROM. The menu works with the help of two buttons .The button 2 was used to scroll through the menu items and a corresponding LED pattern is displayed. Once the desired option is obtained the button 3 is pressed to select it and then button 3 is pressed for further requirements. But while increasing the Kp, Ki or Kd values button 3 decrements and button 2 increments these values.

Menu interface:

Menu Scrolling by Button 2	Button 3	Button3	Button2
compensation	selects	Black compensation	White compensation
Ki	Selects	Decrement	Increment
Kd	Selects	Decrement	Increment
Kp	Selects	Decrement	increment
EEPROM resets	selects	confirms	Do nothing

Delay dc (for tuning the stopping point)	selects	Decrement	increment

STAGE 9: Using pulse encoder

We were facing a big problem in the previous stages that whenever we tune the robot at one speed perfectly then after sometime the battery power will go down and the spedd at which we were running would vary and hence the tuning has to be all over again. So to overcome this problem we used the pulse encoder and implemented PD controller in it to derive constant power from the battery.

STAGE 10: Floating point emulation

We tested using Real time simulation in code warrior to calculate our sampling time. What was found was a very interesting point. Using float in the program decrease the speed of processing of calculation by 100 times for one float.

This is a very significant issue so we decided to implement fixed point algorithm. Further exploring the net we discovered that the use of complex fixed point algorithm can be avoided in this processor by dividing the result by a particular number as here.

As MC9S12 series are equipped with hardware division circuitry, dividing doesn't consume any unnecessary clock cycles and operates quite good in speed. This way we were able to increase our sampling time from 1.6ms to 0.5ms.

Sampling Time

Overall Sampling time: 0.5ms

ATD Interrupt time : Time spent in the ATD interrupt. Since this interrupt is very frequent, the time of execution of this interrupt has to as small as possible. We made it the smallest possible and the time is 0.084ms.

Things Learnt

Documentation

The documentation from the datasheet must be done using screenshots. Even if the datasheet is very concise, taking screenshots and writing some specks in them is very beneficial and easy to understand.

The documentation must also be made and managed in a revision control as well.

Code

The code has to necessarily written in the form of revision control system. This is very important especially when continuous improvement or fixing of bugs is done over a time span. When the project has to undergo some more than 10 revisions then the revision control system is very necessary.

The patch system should be chosen and entries should be done about the changes that fix the bugs or introduces some improvement in software.

The main loop should be small, up to the level that everything is clearly visible in the loop itself without the need of scrolling. Although time taken to switch between functions should be considered and noted in the particular platform in which we intended to work in.

Not only the main loop, but the whole code should be properly modularized so the readability of code can be enhanced and bug detection and removal can be easily done.

Code representations

Code has to be properly represented in flowcharts, algorithms and state diagrams for each revision of the software. So either, get some software that will map the flowcharts in the computer for each separate revision or get this in a separate notebook.

As identifying bugs and then fixing them is a great pain and take a lot of time in code writing, sometimes situation comes such that a person fears even to touch the code because he thinks some bug will get into it. A separate notebook should be maintained for each project code. This notebook should contain some important information about the bugs found in the software and some other sensitive code related issues.

Team

Coding has to be mentored or mastered by one person, so all the software maintenance lies in single hand but two to three people can collaborate in software bug removal and testing. That's a big relief. People who want to contribute to code must submit patches as in Ubuntu, the master coder can choose to apply, reject or merge a particular patch.

After making one version of the code, the code should be send to the **testing team** so that it can test the code and find possible errors and bugs/improvements and documents the bugs in a report along with the result of applying these on the codes. Meanwhile the software writer can work on some more improvements in the code or some alternate code writing.

This way one person can convert the brainstorming into code and possible practical solutions while the testing team will test each version of the code and in the last the best possible solution can be chosen for the finals.

At competition

During the competition, we should try our most optimal and the most precise code at first. This will build the required confidence and so we can later try more speed in the later tries. Also we should focus on testing there and no code without testing should be introduced in the final software.

Also another important thing is that we should not add some new feature in the code on the competition day. If some very important need is there then there should be parallel testing going on by the testing team.