|  |
| --- |
|  |
| Freescale’s Path Finder |
|  |
| In connection with Freescale’s Smart Car Competition |

**Team XLNC**

**Thapar University**

**Patiala**

Contents

Acknowledgment

Abstract

* Introduction 3
* Major components 4
* Control system 5
* Software design 9
* The source code 12
* Development tools 44
* Sampling time 47

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

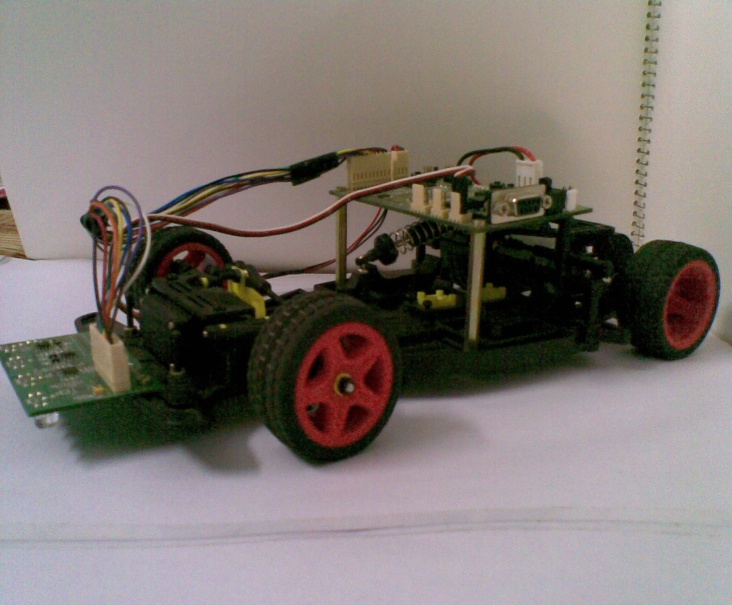
In this report we discuss the SMART CAR, which was designed to participate in the competition path follower meeting the rules and parameters set for the competition. Besides the hardware necessary to sense, he adapted a speed control based on PID technique, implemented in a microcontroller HC12 family.

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.

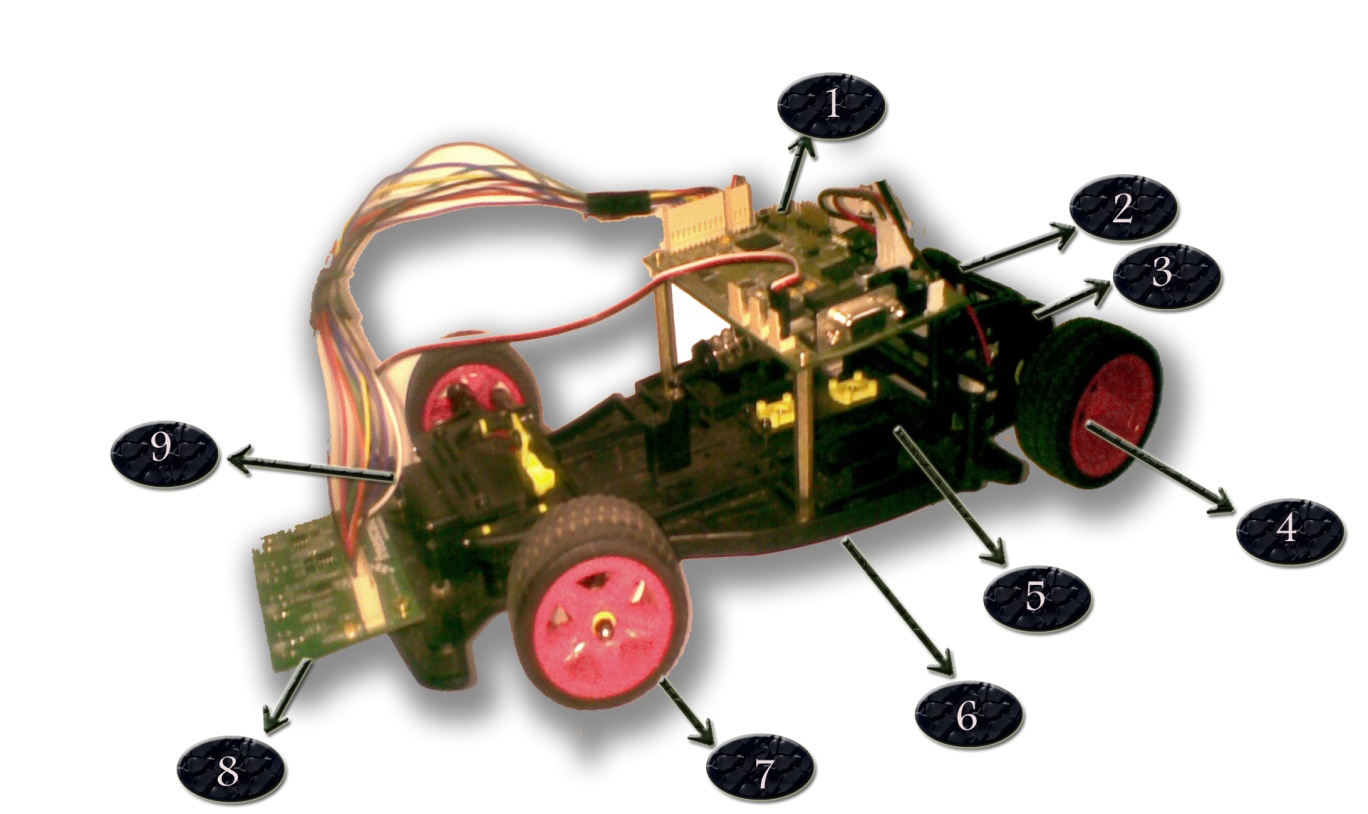
Introduction

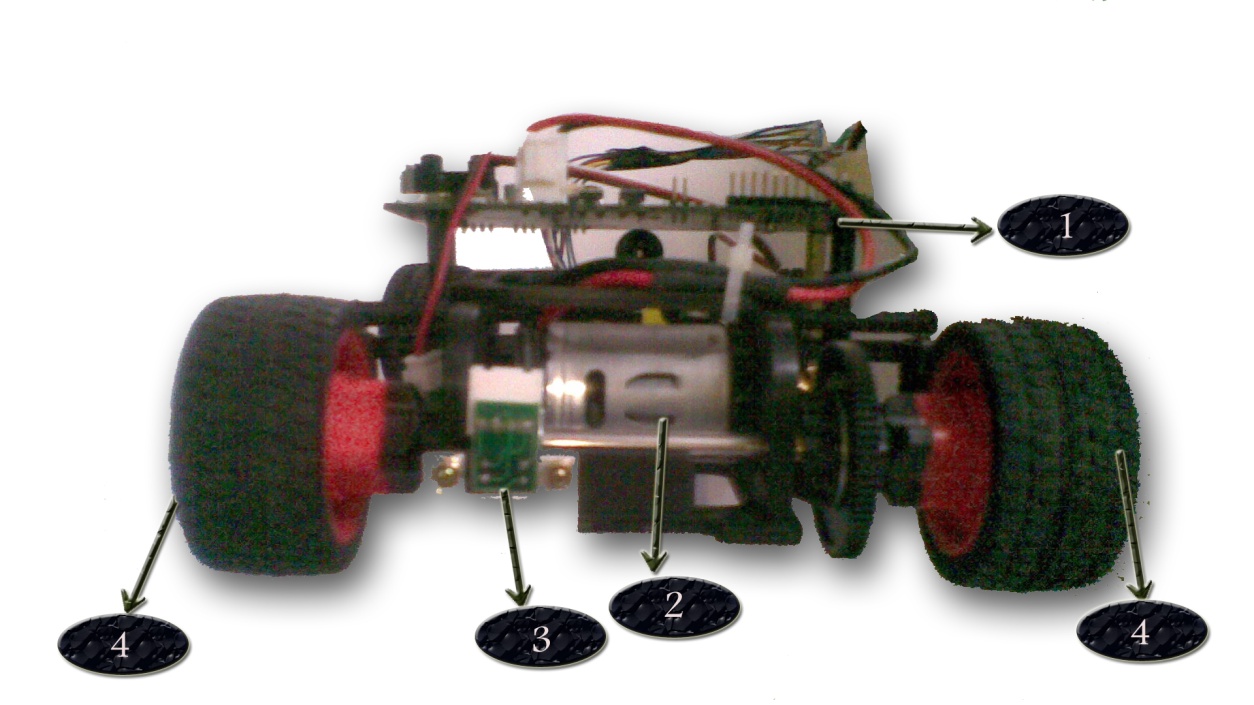
The path follower was made in connection with the Freescale’s smart car race -2010. The robot automatically detects its path (in this case a black line) and follow it by positioning itself on the track as per the programme written in it.the black line is sensed by the array of 8 sensors positioned in 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:

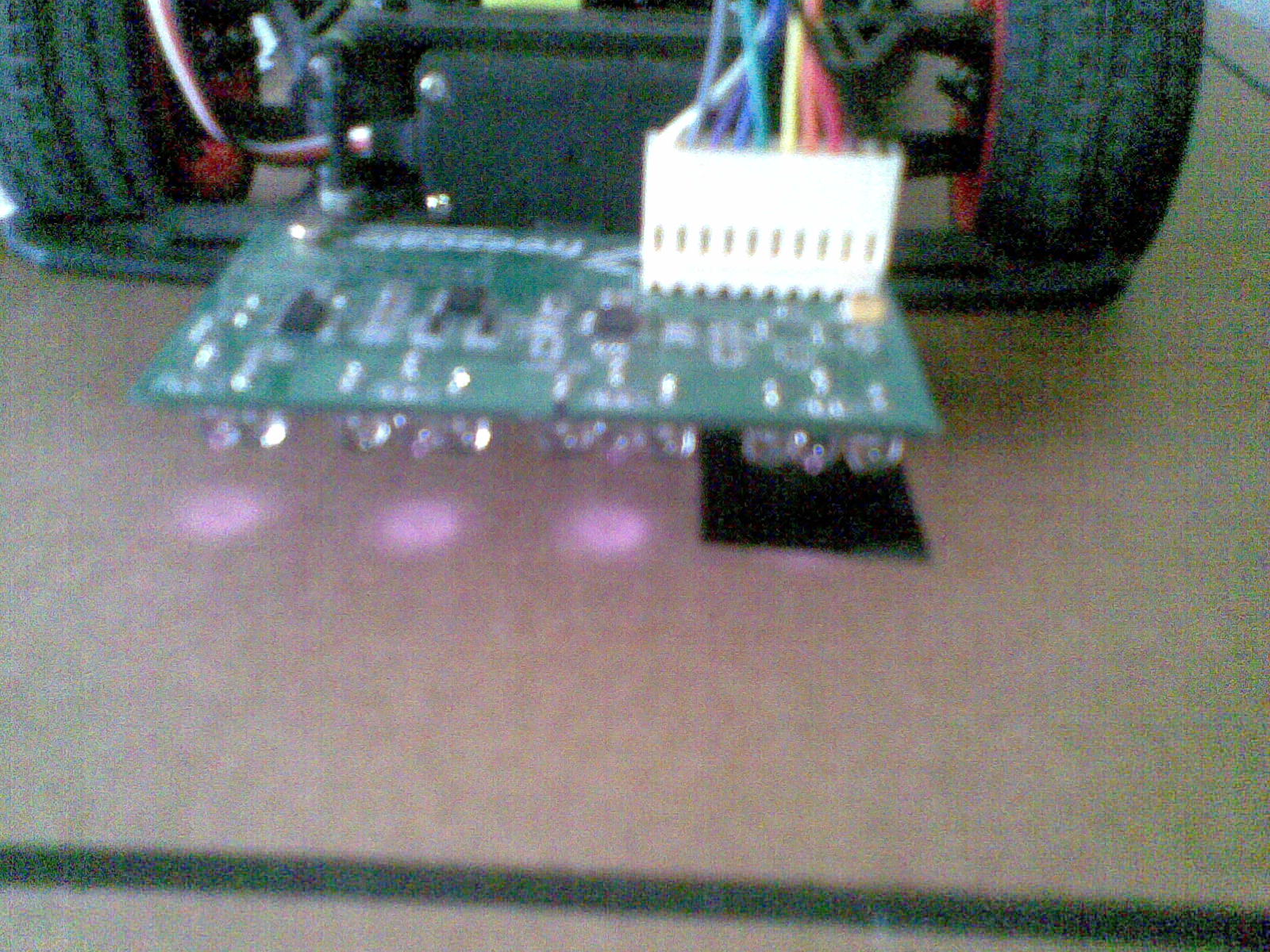
1. Circuit board
2. DC Motor
3. Shaft Encoder
4. Rear wheels
5. Battery holder
6. Chasis
7. Front wheels
8. IR Sensors
9. Servo Motor

Control System

The control system that we used in our car is a PID controller or proportional–integral–derivativecontroller 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.

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

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



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:

**C:\Users\khaira\Desktop\6ae99388a627a49f06b3206da2160a89.png**

**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 *Kp*, called the proportional gain.

The proportional term is given by:

*Pout = Kp e(t)*

Kp = 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, *Ki*.

The integral term is given by:

I_{\mathrm{out}}=K_{i}\int_{0}^{t}{e(\tau)}\,{d\tau}

where

*I*out: Integral term of output

*Ki*: 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 *Kd*. The magnitude of the contribution of the derivative term (sometimes called *rate*) to the overall control action is termed the derivative gain, *Kd*.

The derivative term is given by:

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

where

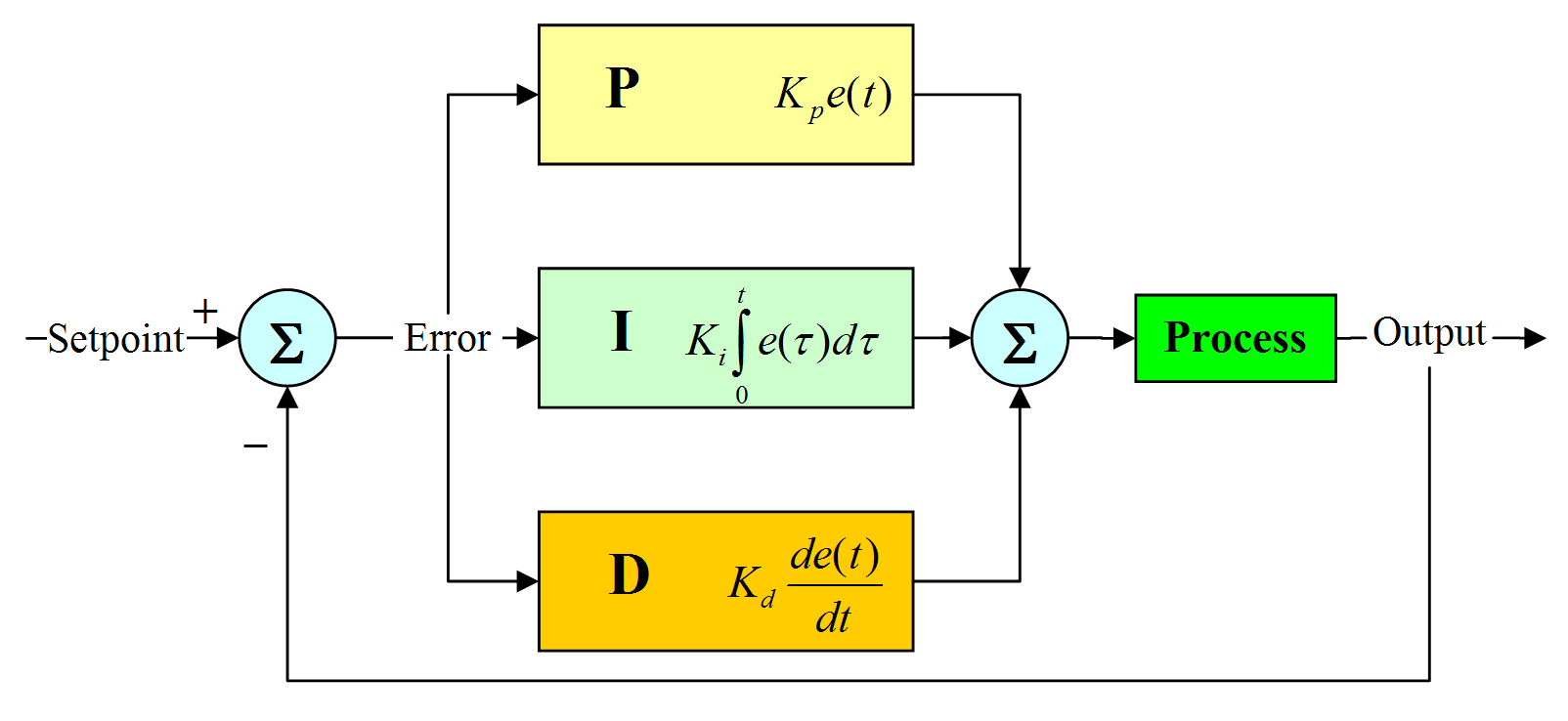
*D*out: Derivative term of output

*Kd*: 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 Kp, Ki and Kd 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 :

All initialization and declaration

Stop\_value=0

While(1)

Compensate Reading

If 600<total sum<800

and correction<25

Stop\_value++

If Stop\_value=2

Dc\_motor\_stop

Servo\_set

P I D

Correction

Find error

Find position

Control Speed

Sort Array

**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.

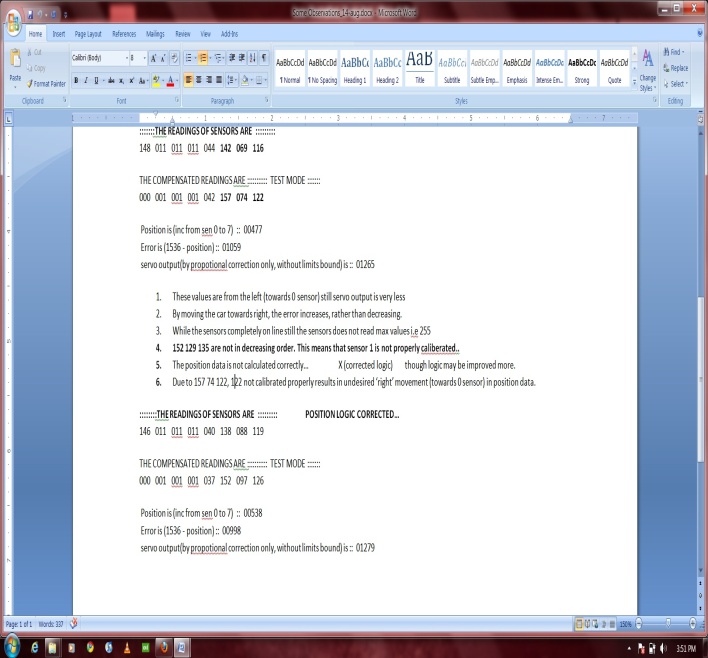
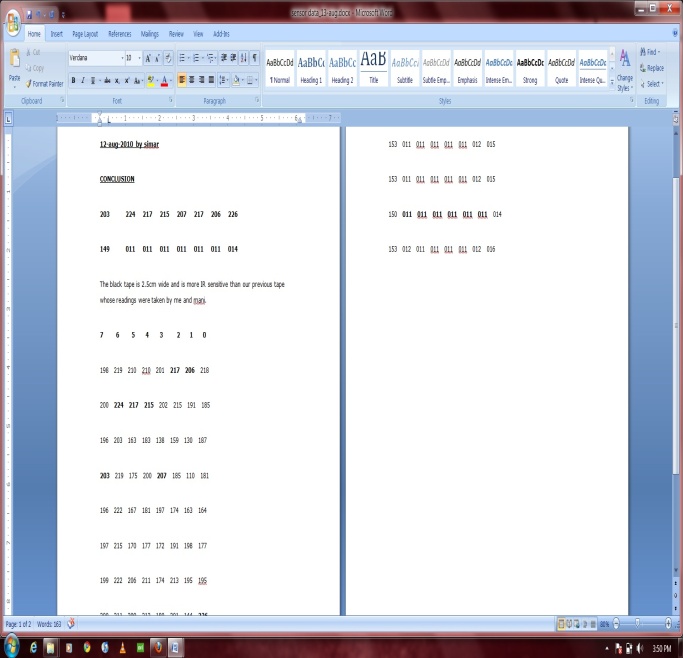
The main problem with the Data Logger was that, we were not able to see the data in real time and also the software is not so portable. As need is the force to creation and innovation of ideas, further exploration towards the end, we encountered code warrior virtualization tool. So we had another powerful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

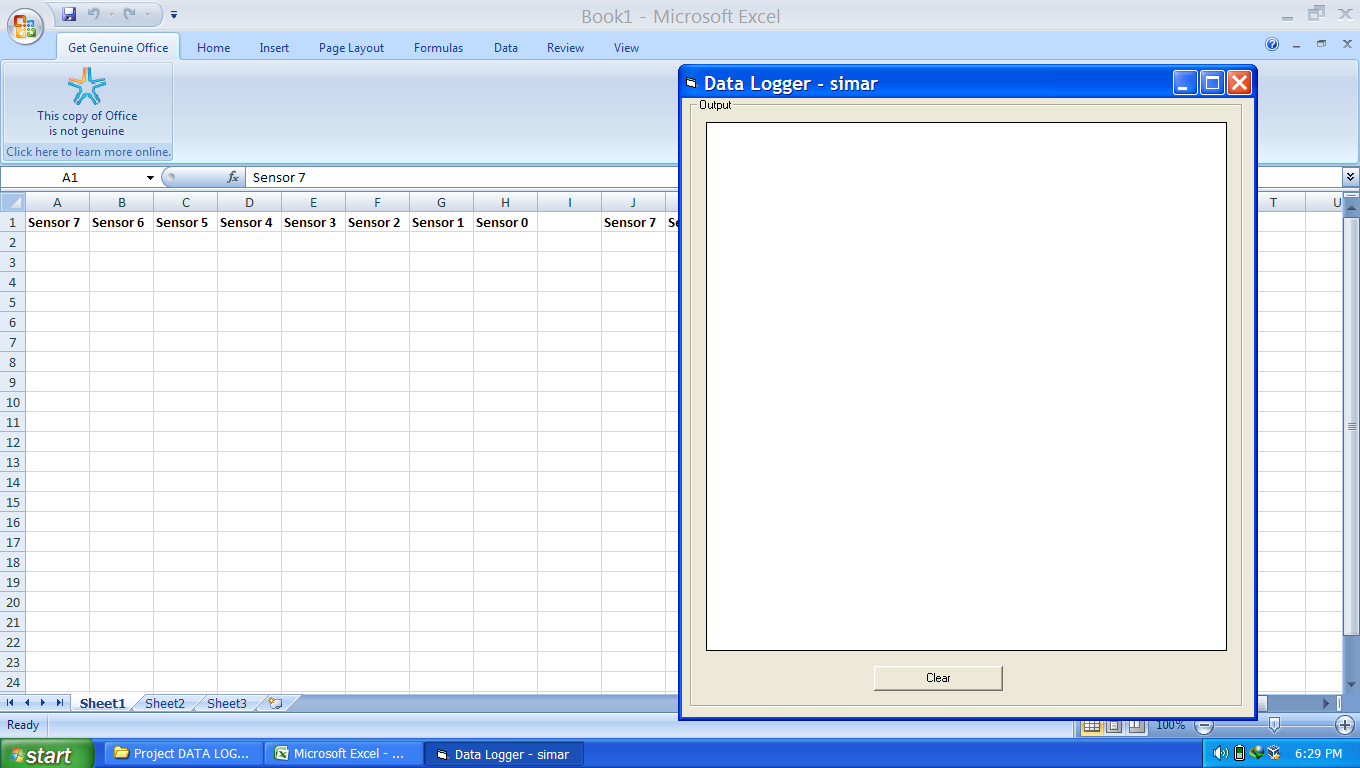
While the codewarrior was provided to us by Freescale itself. The data logger software was made exclusively for logging the sensor data coming from the ATD converter into a spreadsheet from where the desired functions on the data can be implemented.

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 .

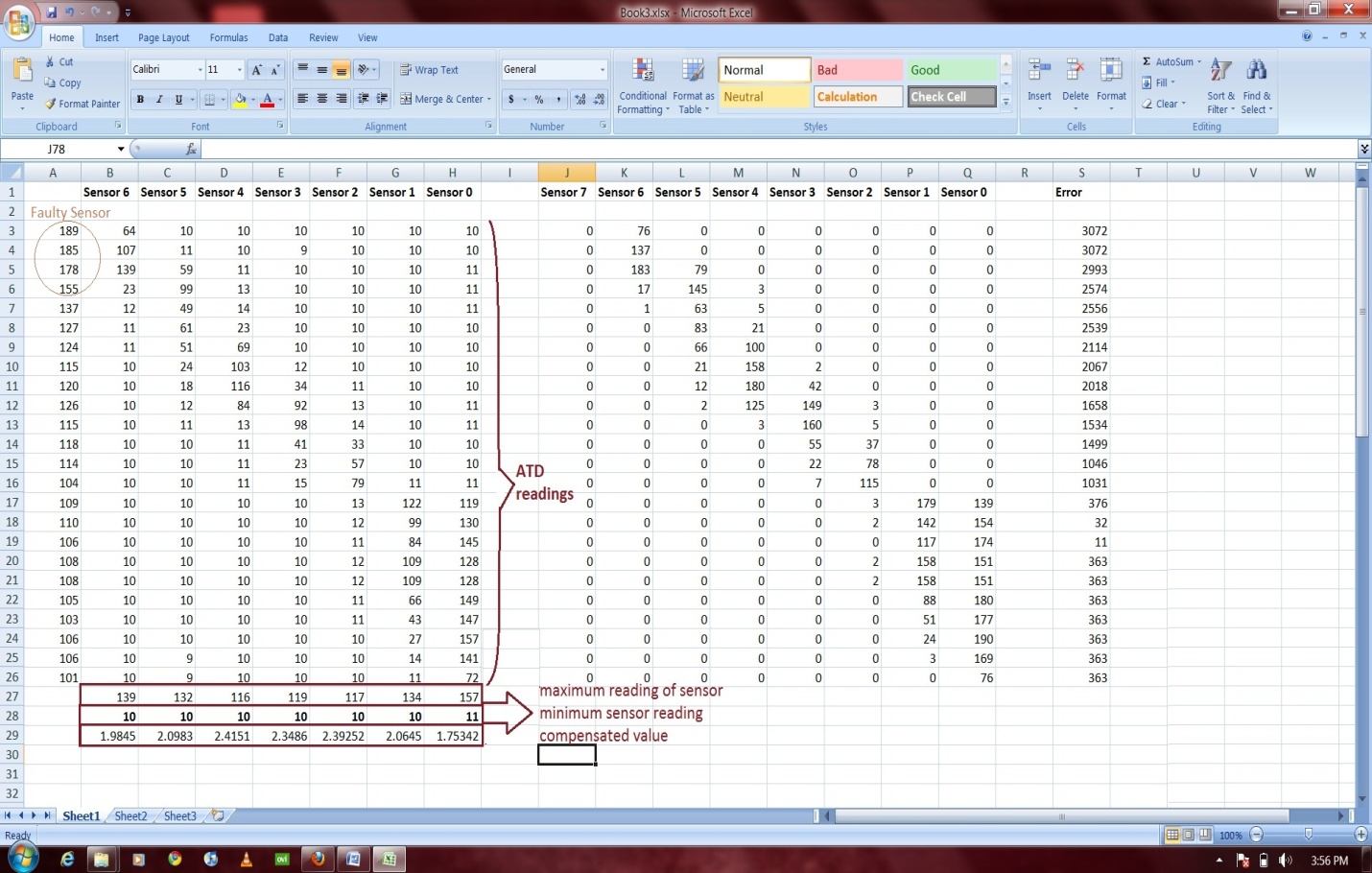
**SCREENSHOTS:**

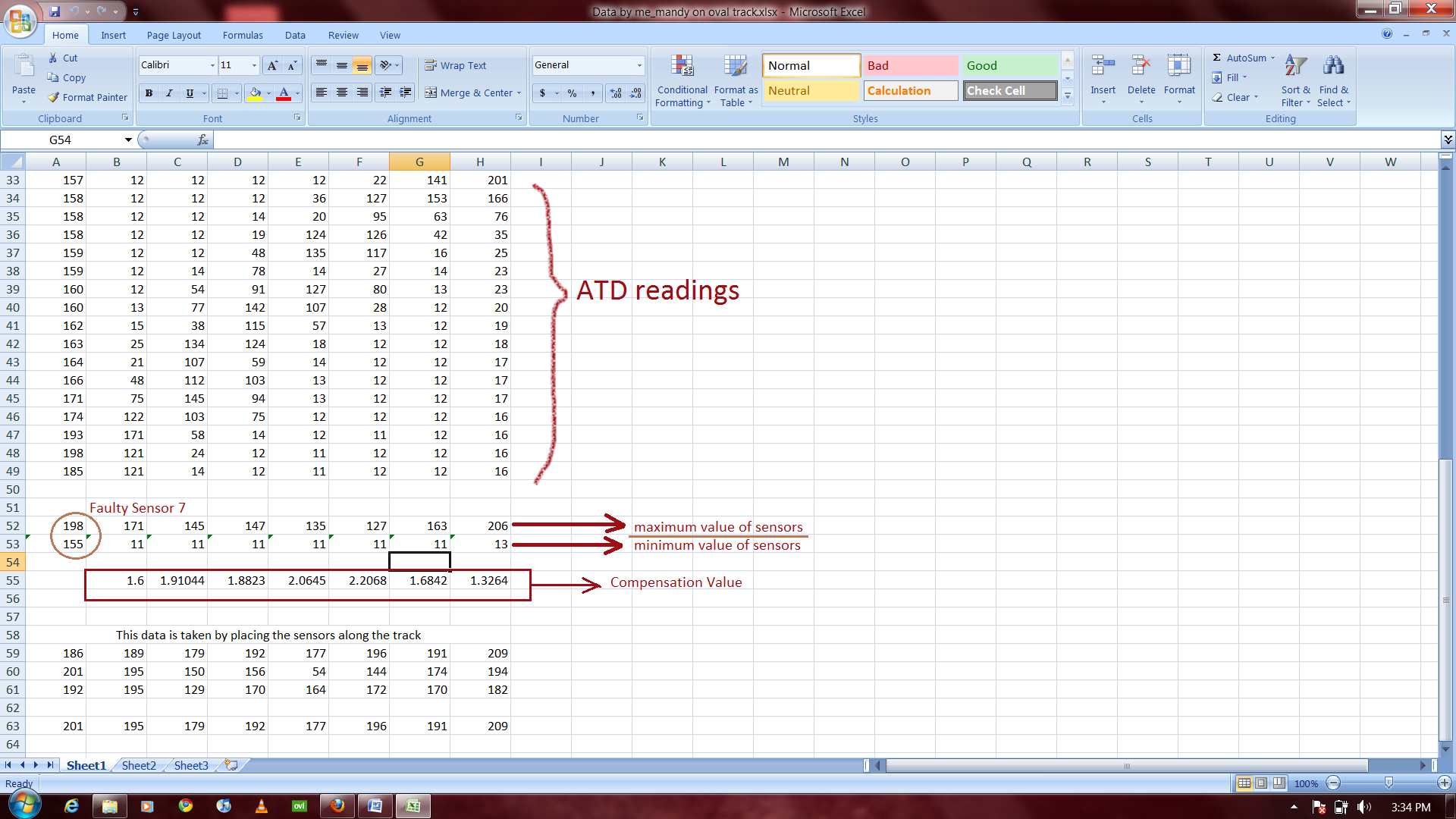
** **

Observing the values from the serial port and deriving conclusions

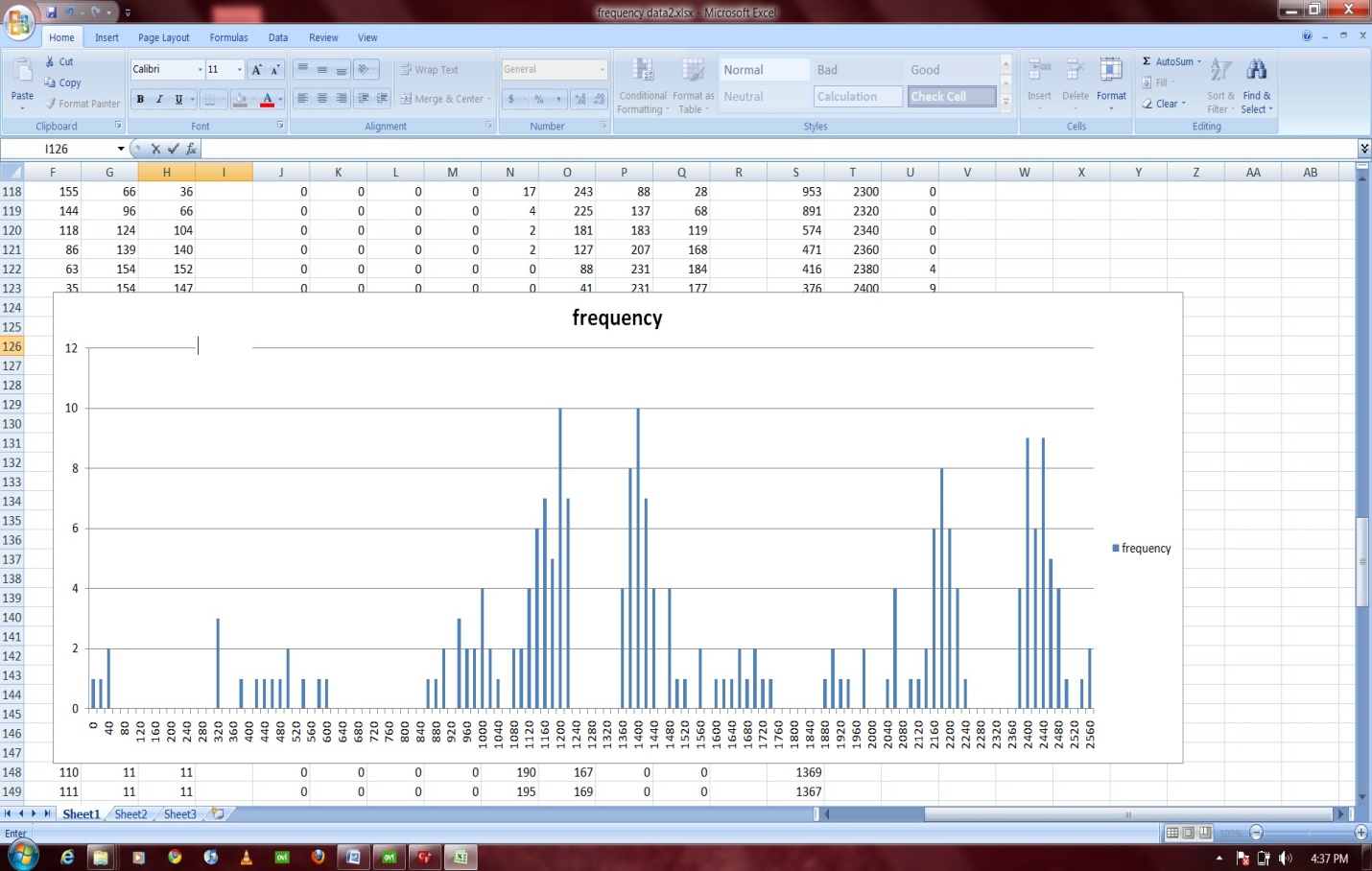


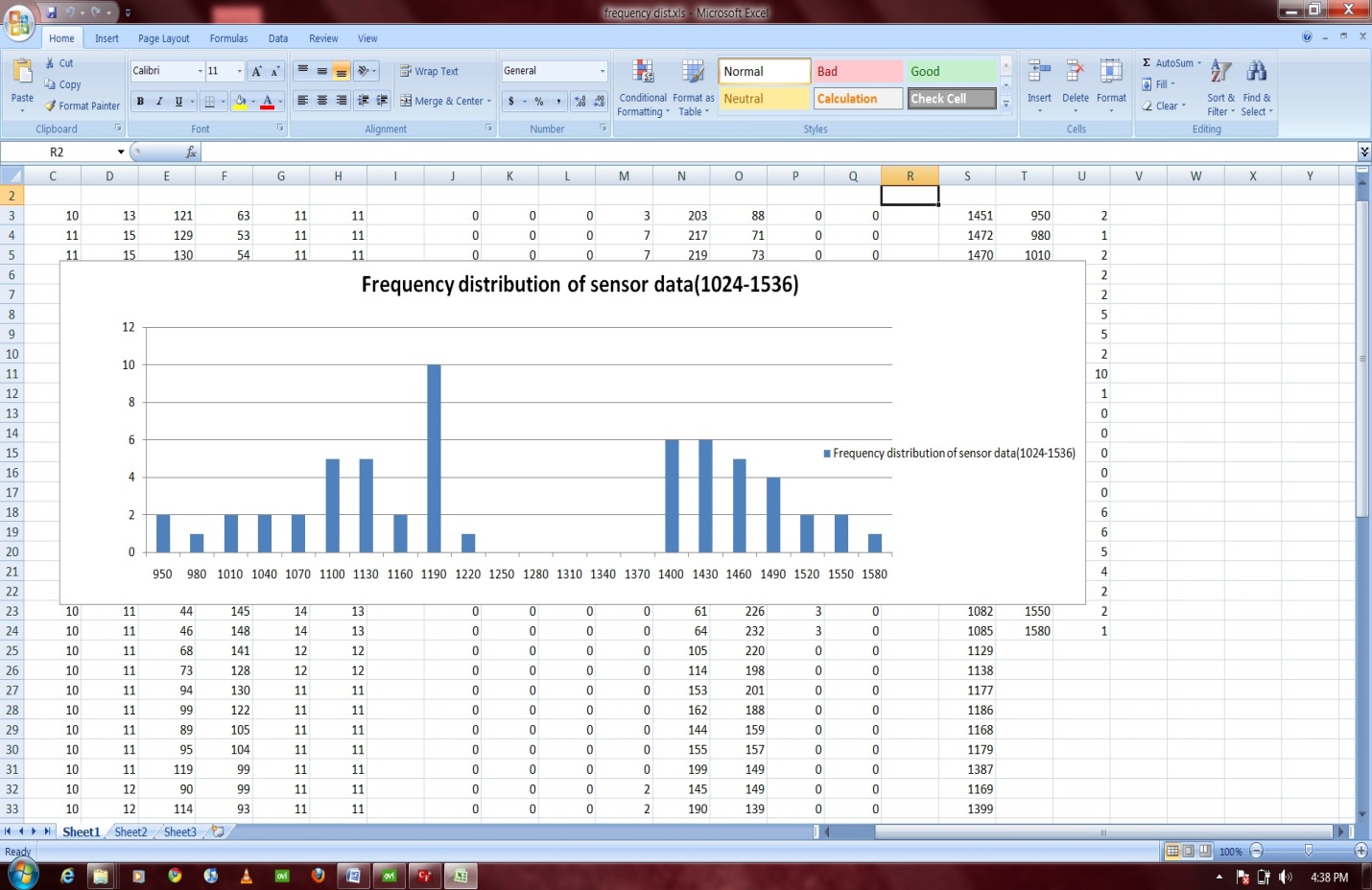
Data Logger software interface



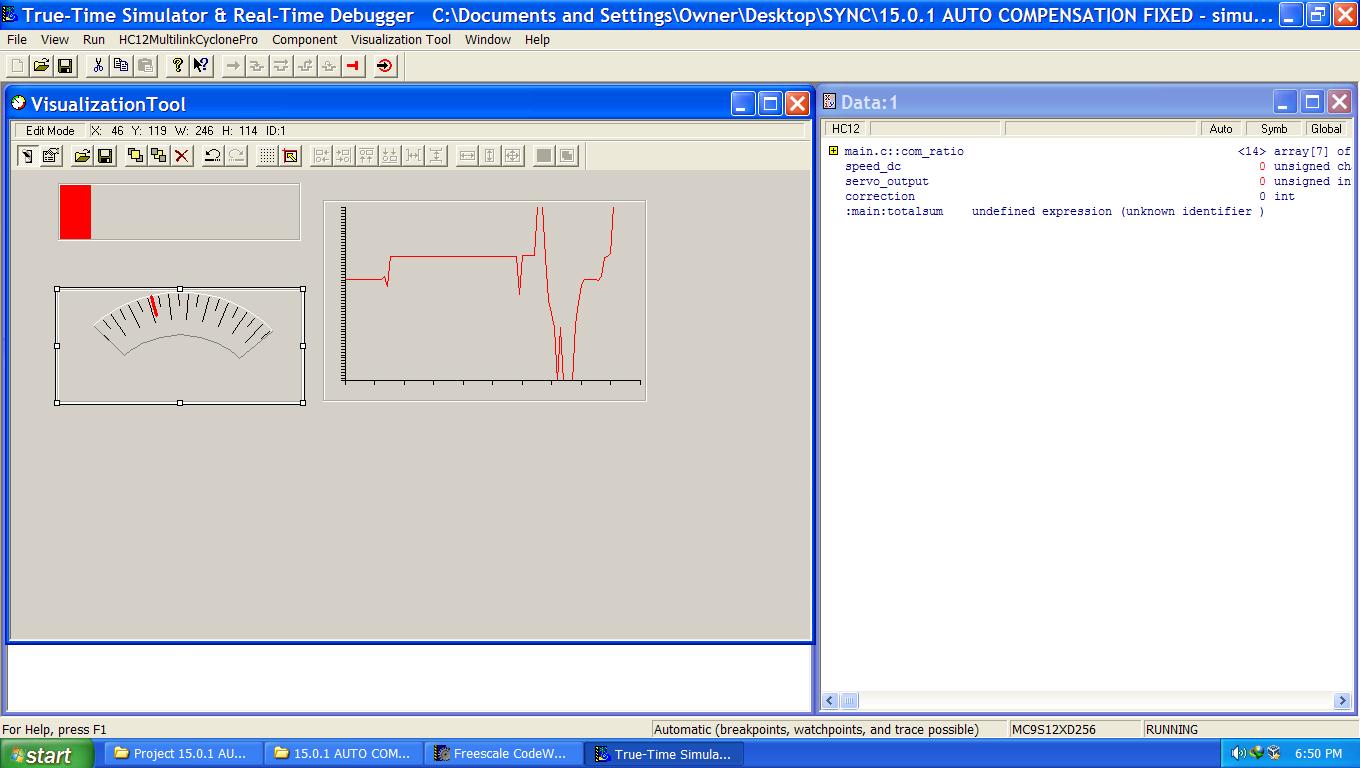


Date logged into a spreadsheet by data logger

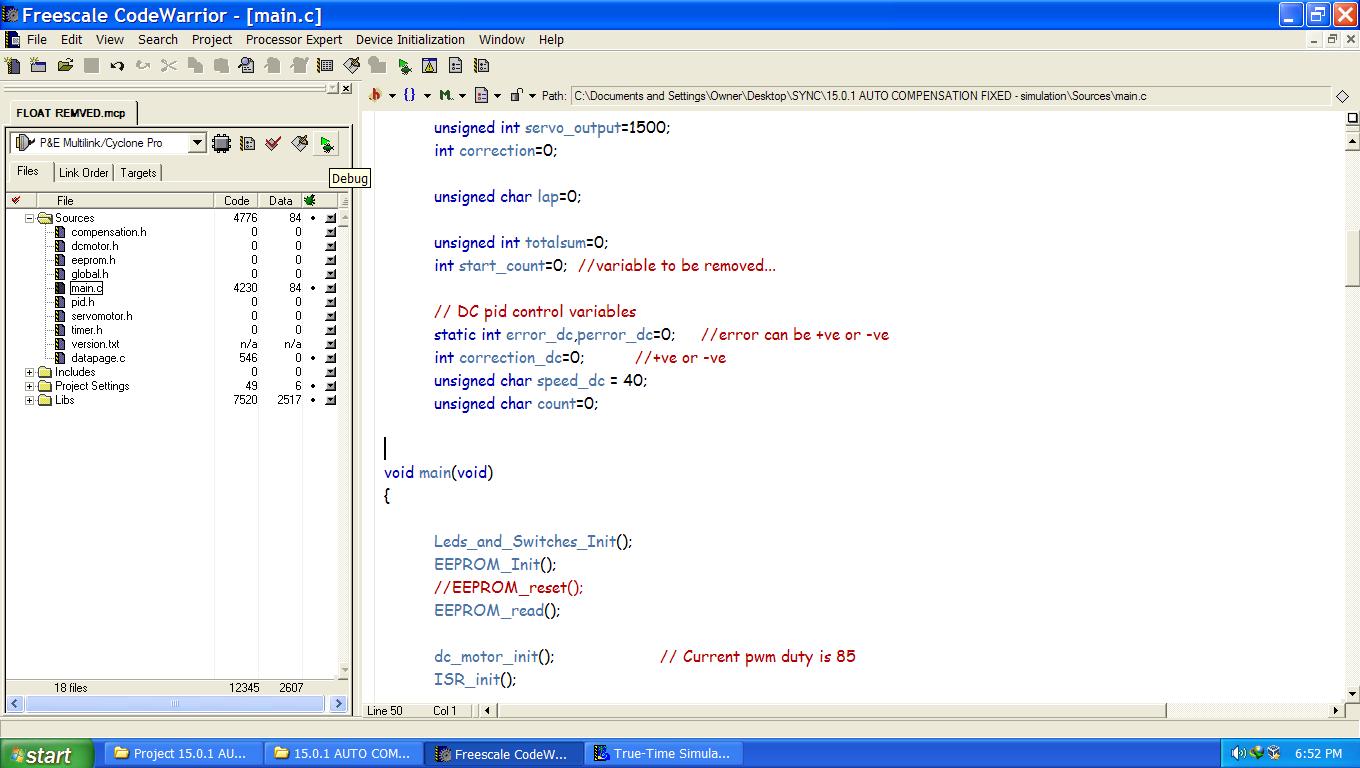
****

****

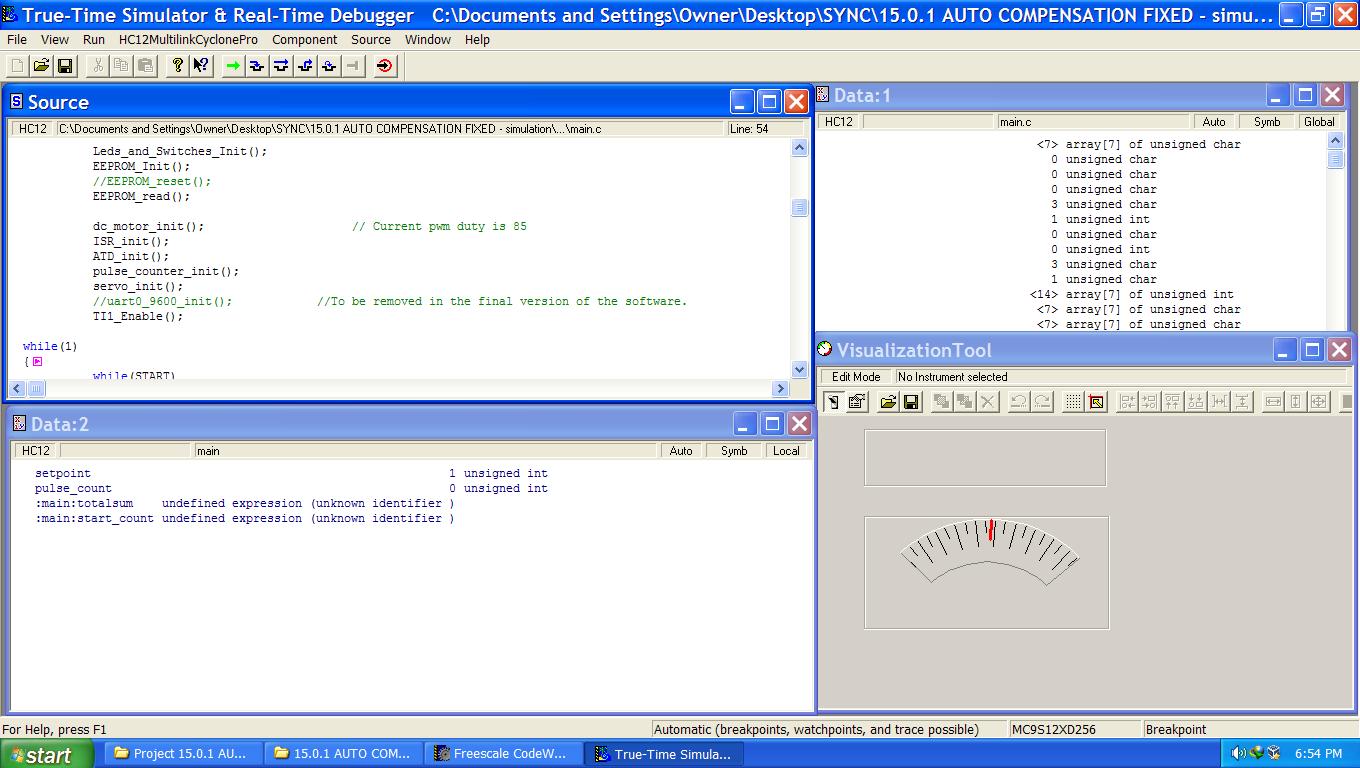
Drawing conclusions from the data logged



Virtualization Tool



Codewarrior coding section



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.