ABSTRACT

Everything in this modern world is getting affected by automation. Robots are being used for completion of monotonous or tedious tasks of human. Keeping this in mind we programmed Fire Bird v robot by developing its library and used the same to make applications like line follower and odometer. This type of application play very significant role in many projects and systems and can be applied to most of them where autonomous robots are used.

ACKNOWLEDGEMENT

It is my privilege to express my sincerest regards to my project supervisor Prof. Hitesh Patel, for their valuable inputs, able guidance, encouragement, whole-hearted co-operation and direction throughout the duration of our project. I deeply express my sincere thanks to Prof. Hitesh Patel for encouraging and allowing us to pursue summer internship in E-Yantra robotics lab at the department and work with FireBird V Robot.

Jaynil Pandya (16ec060)

Soham Patel (16ec085)

TABLE OF CONTENTS

LIST OF FIGURES

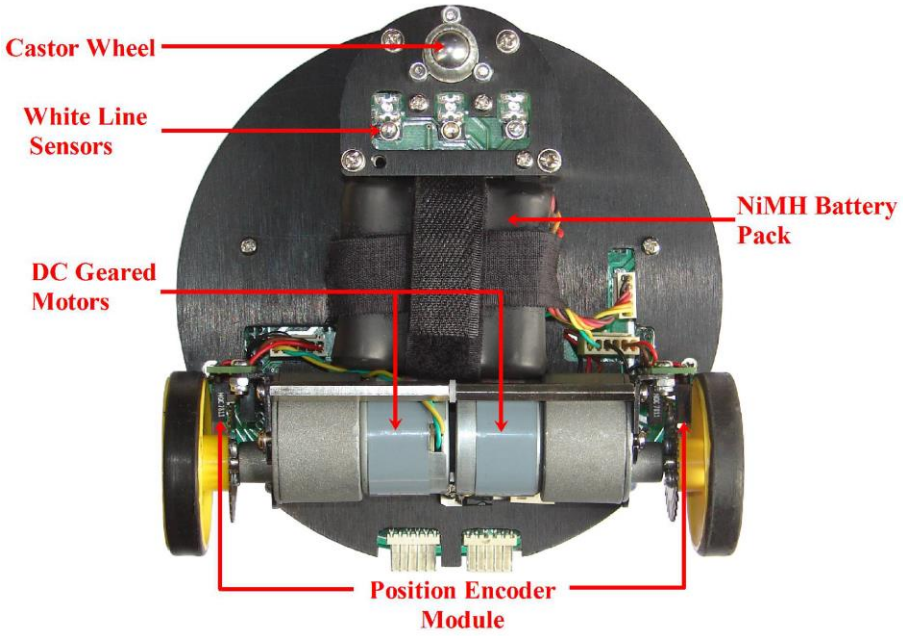
INTRODUCTION

Fire Bird V is robot developed by Nex-Robotics at IIT Bombay. The Fire Bird V robot is the 5th in the Fire Bird series of robots. All the Fire Bird V series robots share the same main board and other accessories. Different family of microcontrollers can be added by simply changing top microcontroller adapter board. Fire Bird V supports ATMEGA2560 (AVR), P89V51RD2 (8051) and LPC2148 (ARM7) microcontroller adapter boards. This modularity in changing the microcontroller adapter boards makes it very versatile. Fire Bird V will help us gain exposure to the world of robotics and embedded systems. With help of its innovative architecture and adoption of the 'Open Source Philosophy' in its software and hardware design, we will be able to create and contribute to complex applications that run on this platform, helping us acquire expertise as we spend more time with them.

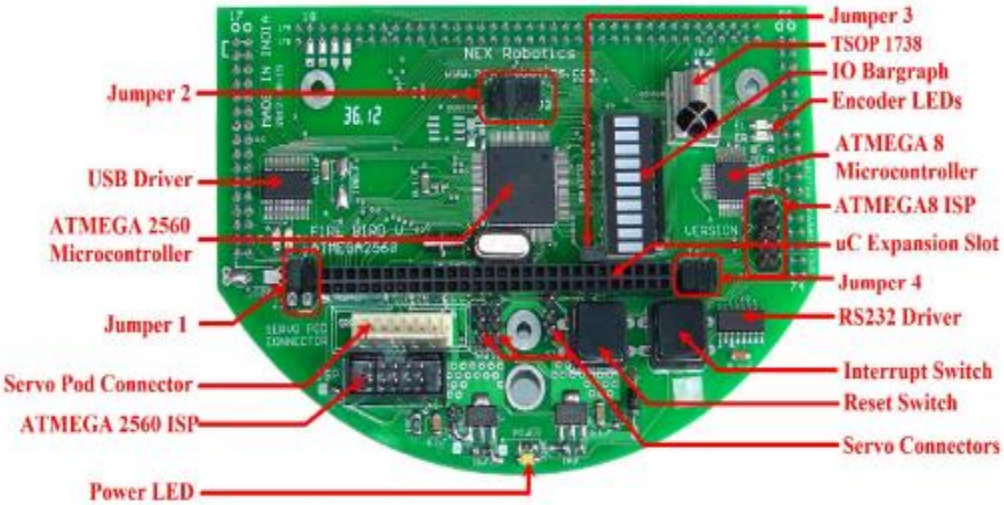
The Fire Bird Robot we are using uses Atmega2560 microtroller as master and Atmega8 as slave.

1.1 FireBird V Atmega2560

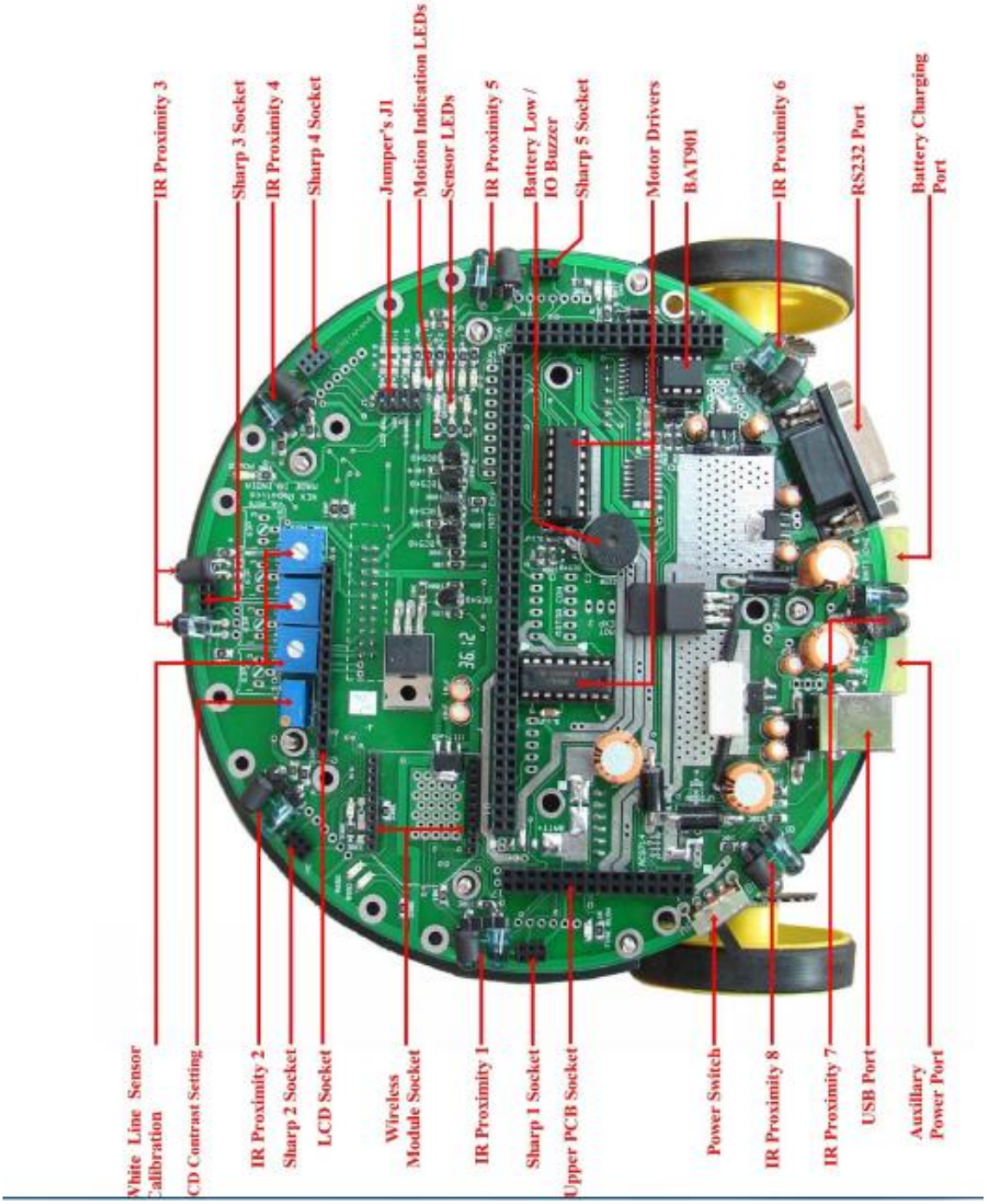
Connections



1.2 Bottom View of Fire Bird



1.3 Fire Bird microcontroller adapter board



1.4 Top view of main board in Fire Bird

Technical Specifications of Fire Bird V Atmega2560

(i)Microcontroller:

Atmel ATMEGA2560 as Master microcontroller (AVR architecture based Microcontroller)

Atmel ATMEGA8 as Slave microcontroller (AVR architecture based Microcontroller)

(ii)Sensors:

a).Three white line sensors (extendable to 7)

b).Five Sharp GP2Y0A02YK IR range sensor (One in default configuration)

c).Eight analog IR proximity sensors

d).Two position encoders (extendable to four)

e).Battery voltage sensing

f).Current Sensing (Optional)

(iii) Indicators:

a).2 x 16 Characters LCD.

b).Buzzer and Indicator LEDs.

(iv) Control:

a).Autonomous Control

b).PC as Master and Robot as Slave in wired or wireless mode

(v) Communication:

a).USB Communication

b).Wired RS232 (serial) communication

c).Wireless ZigBee Communication (2.4GHZ) (if XBee wireless module is in-stalled)

d).Wi-fi communication

e).Bluetooth communication

f).Simplex infrared communication (From infrared remote to robot)

(vi) Dimensions:

a).Diameter: 16cm

b).Height: 8.5cm

c).Weight: 1100gms

(vii) Power:

a).9.6V Nickel Metal Hydride (NiMH) battery pack and external Auxiliary power from

b).battery

c).charger.

d).On Board Battery monitoring and intelligent battery charger.

(viii) Battery Life:

a). 2 Hours, while motors are operational at 75% of timeLocomotion:

Two DC geared motors in differential drive configuration and caster wheel at front

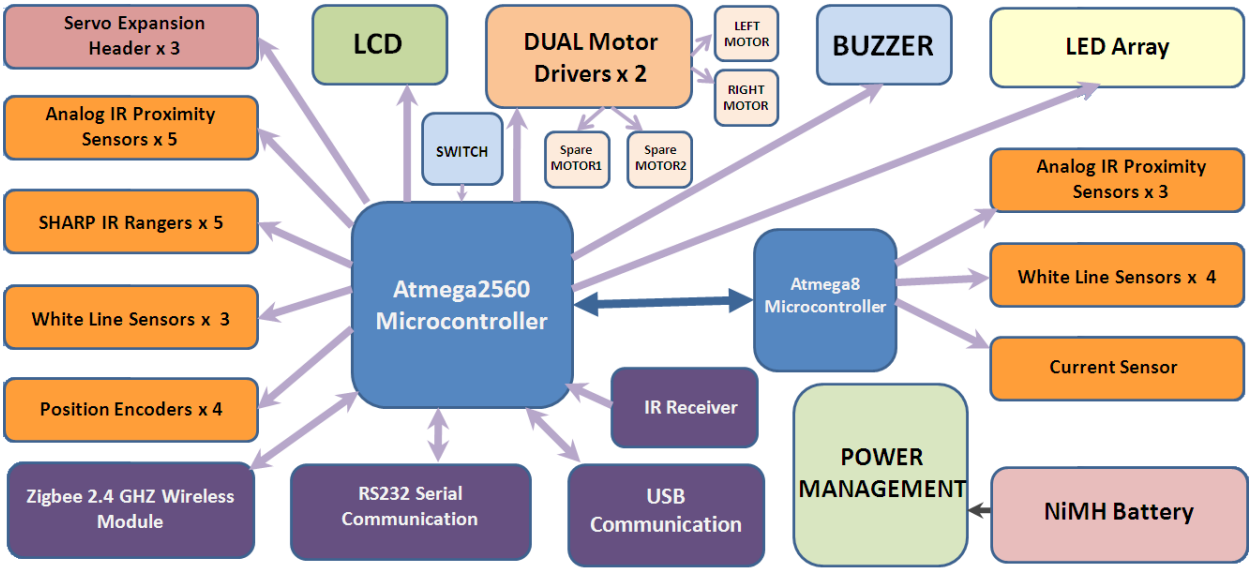
Top Speed: 24 cm / second

Wheel Diameter: 51mm

Position encoder: 30 pulses per revolution

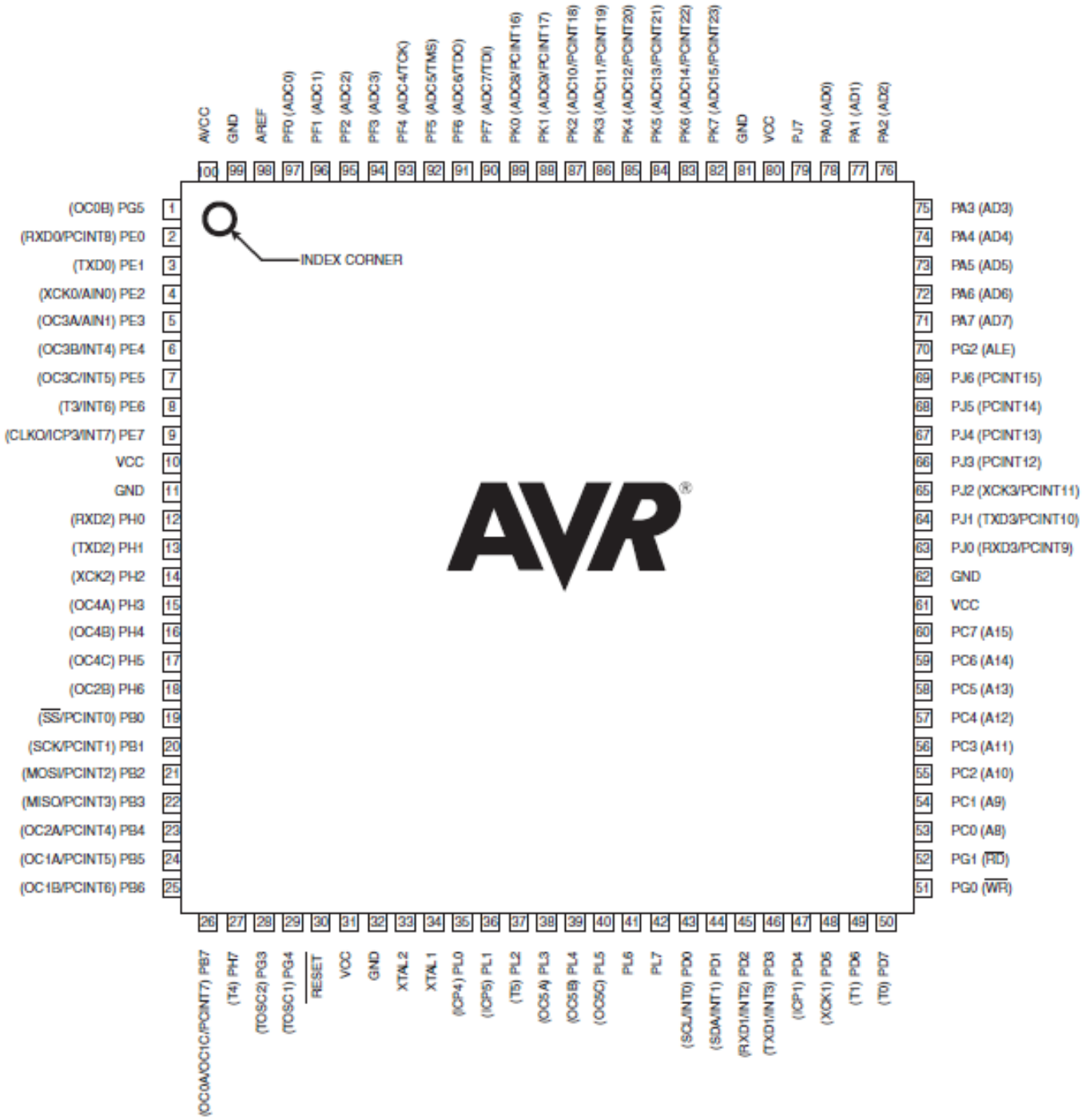
Position encoder resolution: 5.44 mm

Block Diagram



1.5 Fire Bird V ATMEGA2560 robot block diagram

Pin Diagram of Atmega2560 Controller



1.6 Pin Diagram of Atmega2560

Atmega 2560 Pin Configuration Table

Pin No	Pin name	USED FOR	Status
1	(OC0B)PG5	Slave Select (SS) of the SPI expansion port on the main board (refer to figure 3.5)	--
2	RXD0/PCINT8/PE0	UART 0 receive for XBee wireless module (if installed)	Default
3	TXD0/PE1	UART 0 transmit for XBee wireless module (if installed)	Default
4	XCK0/AIN0/PE2	GPIO* (Available on expansion slot of the microcontroller socket)	
5	OC3A/AIN1/PE3	PWM output for C2 motor drive	Output
6	OC3B/INT4/PE4	External Interrupt for the left motor's position encoder	Input
7	OC3C/INT5/PE5	External Interrupt for the right motor's position encoder	Input
8	T3/INT6/PE6	External Interrupt for the C2 motor's position encoder	Input
9	CLK0/ICP3/INT7/ PE7	External Interrupt for Interrupt switch on the microcontroller board, External Interrupt for the C1 motor's position encoder, Connection to TSOP1738 if pad is shorted, can also be used as Boot loading switch ********	Input
10	VCC	5V	
11	GND	Ground	
12	RXD2/PH0	UART 2 receives for USB Communication. For more details refer to section 3.19.7	Default
13	TXD2/PH1	UART 2 transmit for USB Communication. For more details refer to section 3.19.7	Default
14	XCK2/PH2	IR proximity sensors 1 to 8 enable / disable. Turns off these sensors when output is logic 1 *******	Output
15	OC4A / PH3	Sharp IR ranges sensor 1and 5 enable / disable. Turns off these sensors when output is logic 1 *******	Output
16	OC4B / PH4	Connected to Rx pin of 1st Ultrasonic range sensor to trigger the ultrasonic sensor if sensor is mounted. Also Available on expansion slot of the microcontroller socket*********	--
17	OC4C / PH5	Available on expansion slot of the microcontroller socket	--
18	OC2B / PH6	Available on expansion slot of the microcontroller socket	--
19	SS/PCINT0/PB0	ISP (In System Programming), SPI Communication with ATMEGA8 **, Connection to the SPI port on the main board. Also available on expansion slot of the microcontroller socket	
20	SCK/PCINT1/PB1		Output
21	MOSI/PCINT2/PB2		Output
22	MISO/PCINT3/PB3		Input
23	OC2A/PCINT4/PB4	Servo Pod GPIO	--
24	OC1A/PCINT5/PB5	PWM for Servo motor 1. ***	Output
25	OC1B/PCINT6/PB6	PWM for Servo motor 2. ***	Output
26	OC0A/OC1C/PCINT7/PB7	PWM for Servo motor 3. ***	Output

27	T4/PH7	GPIO (Available On Expansion Slot)	--
28	TOSC2/PG3	RTC (Real Time Clock)****	
29	TOSC1/PG4		
30	RESET	Microcontroller reset	
31	VCC	5V	
32	GND	Ground	
33	XTAL2	Crystal 14.7456 MHz	
34	XTAL1		
35	ICP4/PL0	Connected to RSSI pin of XBee module. Also Available on expansion slot of the microcontroller socket.	--
36	ICP5/PL1	Available on expansion slot of the microcontroller socket.	--
37	TS/PL2	Available on expansion slot of the microcontroller socket.	--
38	OC5A/PL3	PWM for left motor.	Output
39	OC5B/PL4	PWM for right motor.	Output
40	OC5C/PL5	PWM for C1 motor.	Output
41	PL6	GPIO* (Available on expansion slot of the microcontroller socket)	--
42	PL7		--
43	SCL/INT0/PD0	I2C bus / GPIOs (Available on expansion slot of the microcontroller socket)	--
44	SDA/INT1/PD1		--
45	RXD1/INT2/PD2	UART1 receive for RS232 serial communication	Default
46	TXD1/INT3/PD3	UART1 transmit for RS232 serial communication	Default
47	ICP1/PD4	GPIO* (Available on expansion slot of the microcontroller socket)	--
48	XCK1/PD5		--
49	T1/PD6		--
50	T0/PD7		--
51	PG0/WR	GPIO* (Available on expansion slot of the microcontroller socket)	--
52	PG1/RD		--
53	PC0	LCD control line RS (Register Select)	Output
54	PC1	LCD control line RW(Read/Write Select)	Output
55	PC2	LCD control line EN(Enable Signal)	Output
56	PC3	Buzzer	Output
57	PC4	LCD data lines (4-bit mode)	Output
58	PC5		
59	PC6		
60	PC7		
61	VCC	5V	
62	GND	Ground	
63	PJ0/RXD3/PCINT9	LED bargraph display and GPIO* (Available on expansion slot of the microcontroller socket)	Output
64	PJ1/TXD3/PCINT10		
65	PJ2/XCK3/PCINT11		
66	PJ3/PCINT12		
67	PJ4/PCINT13		
68	PJ5/PCINT14		
69	PJ6/PCINT15		
70	PG2/ALE	Red LEDs of white line sensor enable/disable. ******* Turns off these sensors when output is logic 1	Output
71	PA7 C2-2	Logic input 2 for C2 motor drive	Output

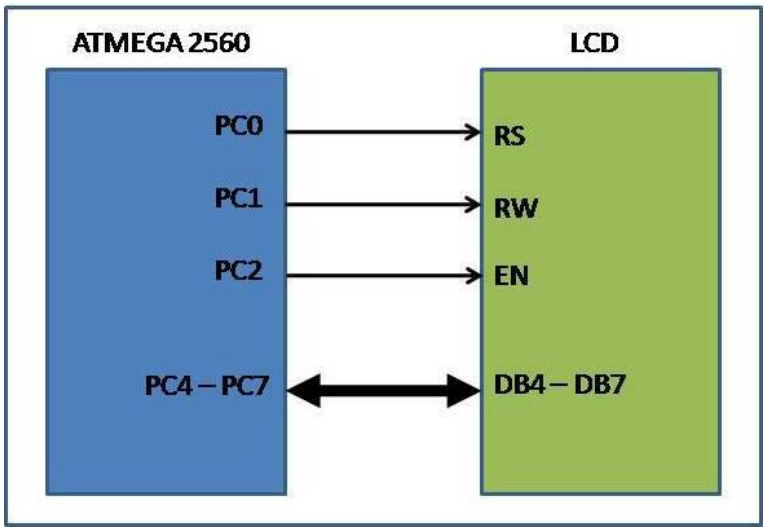
72	PA6 C2-1	Logic input 1 for C2 motor drive	Output
73	PA5 C1-2	Logic input 2 for C1 motor drive	Output
74	PA4 C1-1	Logic input 1 for C1 motor drive	Output
75	PA3	Logic input 1 for Right motor (Right back)	Output
76	PA2	Logic input 2 for Right motor (Right forward)	Output
77	PA1	Logic input 2 for Left motor (Left forward)	Output
78	PA0	Logic input 1 for Left motor (Left back)	Output
79	PJ7	LED Bar Graph and GPIO* (Available on expansion slot of the microcontroller socket)	
80	VCC	5V	
81	GND	Ground	
82	PK7/ADC15/PCINT23	ADC Input For Servo Pod 2	Input (Floating)
83	PK6/ADC14/PCINT22	ADC Input For Servo Pod 1	Input (Floating)
84	PK5/ADC13/PCINT21	ADC input for Sharp IR range sensor 5	Input (Floating)
85	PK4/ADC12/PCINT20	ADC input for Sharp IR range sensor 4	Input (Floating)
86	PK3/ADC11/PCINT19	ADC input for Sharp IR range sensor 3	Input (Floating)
87	PK2/ADC10/PCINT18	ADC input for Sharp IR range sensor 2	Input (Floating)
88	PK1/ADC9/PCINT17	ADC input for Sharp IR range sensor 1	Input (Floating)
89	PK0/ADC8/PCINT16	ADC input for IR proximity analog sensor 5	Input (Floating)
90	PF7(ADC7/TDI)	ADC input for IR proximity analog sensor 4*****	Input (Floating)
91	PF6/(ADC6/TD0)	ADC input for IR proximity analog sensor 3*****	Input (Floating)
92	PF5(ADC5/TMS)	ADC input for IR proximity analog sensor 2*****	Input (Floating)
93	PF4/ADC4/TCK	ADC input for IR proximity analog sensor 1*****	Input (Floating)
94	PF3/ADC3	ADC input for white line sensor 1	Input (Floating)
95	PF2/ADC2	ADC input for white line sensor 2	Input (Floating)
96	PF1/ADC1	ADC input for white line sensor 3	Input (Floating)
97	PF0/ADC0	ADC input for battery voltage monitoring	Input (Floating)
98	AREF	ADC reference voltage pin (5V external) ******	
99	GND	Ground	
100	AVCC	5V	

LITERATURE ANALYSIS

The major components interfaced with Fire Bird used in our internship are :
1. LCD
2. Position Encoders
3. White Line Sensors

LCD

To interface LCD with the microcontroller in default configuration requires 3 control signals and 8 data lines. This is known as 8 bit interfacing mode which requires total 11 I/O lines. To reduce the number of I/Os required for LCD interfacing we can use 4 bit interfacing mode which requires 3 control signals with 4 data lines. In this mode upper nibble and lower nibble of commands/data set needs to be sent separately. Figure below shows LCD interfacing in 4 bit mode.
The three control lines are referred to as EN, RS, and RW.



2.1 LCD interfacing with FireBird (4-bit mode)

LCD Code

```
void lcd_begin(void)
{
   LCD_BUZZER_DIR=0xff; //initialize buzzer and lcd as output
   LCD_PORT=0x00; //No input to lcd
   _delay_ms(5);
   lcd_ctrl(0x33);
   lcd_ctrl(0x32);
```

```
   lcd_ctrl(0x28);
   lcd_ctrl(0x0C); //cursor off
   lcd_ctrl(0x06); //move l to r
   lcd_ctrl(0x01); //clear
   lcd_ctrl(0x80); //starting address till 8f
}
void lcd_clear(void)
{
   lcd_ctrl(0x01);
   lcd_ctrl(0x80);
}
void lcd_setcursor(unsigned char a,unsigned char b)
{
   //a is col and b is row
   if(b==0)
   lcd_ctrl((0x80)+a);
   if(b==1)
   lcd_ctrl((0xc0)+a); //add of row 1 c0 to cf
   if(LCD_ROW==4 && LCD_COL==20)
   {
      if(b==2)
      lcd_ctrl((0x94)+a);
      if(b==3)
      lcd_ctrl((0xd4)+a);
   }
}

void lcd_cursor(void)
{
   lcd_ctrl(0x0E); //cursor on
}
void lcd_cursoroff(void)
{
   lcd_ctrl(0x0C);
}
void lcd_ctrl(unsigned char cmd)
{
   LCD_PORT&=~(1<<RS); //RS is  0 for commands
   LCD_PORT=(LCD_PORT & 0x0f) | (cmd & 0xf0); //sending upper
nibble
   LCD_PORT|=1<<EN;
   _delay_ms(1);
   LCD_PORT&=~(1<<EN);
   _delay_ms(1);
   cmd=cmd<<4;
   LCD_PORT=(LCD_PORT & 0x0f) | (cmd & 0xf0); //sending lower
nibble
```

```
   LCD_PORT|=1<<EN;
   _delay_ms(1);
   LCD_PORT&=~(1<<EN);
   _delay_ms(1);
}
void lcd_print(unsigned char *data)
{
   LCD_PORT|=1<<RS; //RS is 1 for data
   while(*data !='\0')
   {
      lcd_printa(*data);
      *data++;
   }
}
void lcd_printa(unsigned char ascii) //ascii value directly
{
   LCD_PORT|=1<<RS;
   LCD_PORT=(LCD_PORT & 0x0f) | (ascii & 0xf0); //sending upper
nibble
   LCD_PORT|=1<<EN;
   _delay_ms(1);
   LCD_PORT&=~(1<<EN);
   _delay_ms(1);
   ascii=ascii<<4;
   LCD_PORT=(LCD_PORT & 0x0f) | (ascii & 0xf0); //sending lower
nibble
   LCD_PORT|=1<<EN;
   _delay_ms(1);
   LCD_PORT&=~(1<<EN);
   _delay_ms(1);
}
```

Position Encoder

Interrupt 4 (INT4) and interrupt 5 (INT5) are connected to the robot's position encoder. Position encoders give position / velocity feedback to the robot. It is used in closed loop to control robot's position and velocity. Position encoder consists of optical encoder and slotted disc assembly. When this slotted disc moves in between the optical encoder we get square wave signal whose pulse count indicates position and time period indicates velocity.

Wheel diameter: 5.1cm
Wheel circumference: 5.1cm * 3.14 = 16.014cm = 160.14mm

Number slots on the encoder disc: 30
Position encoder resolution: 163.2 mm / 30 = 5.44mm / pulse

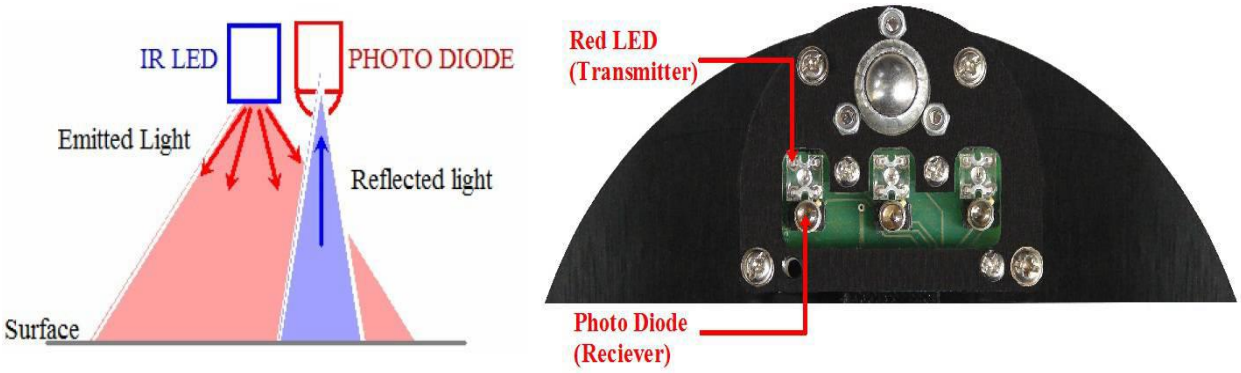
Code for measuring distance using Position Encoder

```
ISR(INT5_vect)
{
   posr++;
   sei();
}
ISR(INT4_vect)
{
   posl++;
   sei();
}
void lcd_print_dist(unsigned char col,unsigned char row)
{
   lcd_setcursor(col,row);
   distance=((posr+posl)/2)*0.544;
   dis_data_op();
   for(cnt=MAX_DIGITS-1;cnt>=0;cnt--)
   {
      logic+=distance_data[cnt];
      if(logic!=0)                              \
      {
         lcd_printa(48+(distance_data[cnt]));
         if(unit==1 && cnt==2)
         {
            lcd_printa(0x2E); //ascii of decimal point
         }
      }
   }
   logic=0;
   if(unit)
      lcd_print("m");
   else
      lcd_print("cm");
}
void dis_data_op(void)
{
   //distance_data[6]=(distance%10000000)/1000000;
   //distance_data[5]=(distance%1000000)/100000;
   //distance_data[4]=(distance%100000)/10000;
   distance_data[3]=(distance%10000)/1000;
   distance_data[2]=(distance%1000)/100;
   distance_data[1]=(distance%100)/10;
```

```
    distance_data[0]=(distance%10);
    if(distance_data[2]>0)
    {
       unit=1;
    }
}
```

White Line Sensor

White line sensors are used for detecting white line on the ground surface. White lines are used to give robot sense of localization. White line sensor consists of a highly directional photo transistor for line sensing and bright red LED for the illumination. Due to the directional nature of the photo diode it does not get affected with ambient light unless it is very bright.



2.2 White line sensor

Code for ADC(White line sensor)

```
void mosfet_switch_config(void)
{
     DDRG = DDRG | 0x04;
     PORTG = PORTG & 0xfb;
}
void interrupt_begin(void)
{
     cli();
     INT|=(1<<POSENL)|(1<<POSENR)|(1<<BUTTON);  //activate  pull-
up resistor for both position encoders as well as for pushbutton
     EICRB|=(1<<ISC71)|(1<<ISC51)|(1<<ISC41);   //falling   edge
interrupt
     EIMSK|=(1<<INT7)|(1<<INT5)|(1<<INT4);
     EIFR|=(1<<INTF7); //requires clearing of interrupt flag for
edge trigerred interrupts
```

```
     //intf7 is 1 when isr is running and 0 after finishing isr
     sei();
}

void adc_begin(void)
{
     ADCSRA = 0x00;
     ADCSRB = 0x00;      //MUX5 = 0
     ADMUX = 0x20;       //Vref=5V  external  ---  ADLAR=1  ---
MUX4:0 = 0000
     ACSR = 0x80;
     ADCSRA = 0x86;      //ADEN=1 --- ADIE=1 --- ADPS2:0 = 1 1 0
}

unsigned int adc_conv(unsigned char Ch)
{
          unsigned char a;
          if(Ch>7)
          {
               ADCSRB = 0x08;
          }
          Ch = Ch & 0x07;
          ADMUX= 0x20| Ch;
          ADCSRA = ADCSRA | 0x40;       //Set  start  conversion
bit
          while((ADCSRA&0x10)==0); //Wait  for  ADC  conversion  to
complete
          a=ADCH;
          ADCSRA  =  ADCSRA|0x10;  //clear  ADIF  (ADC  Interrupt
Flag) by writing 1 to it
          ADCSRB = 0x00;
          return a;
}

void     print_sensor(unsigned     char     row,unsigned     char
coloumn,unsigned char channel)
{
     ADC_Value = adc_conv(channel);
     lcd_setcursor(coloumn,row);
     lcd_printa(48+((ADC_Value/100)%10));
     lcd_printa(48+((ADC_Value%100)/10));
     lcd_printa(48+((ADC_Value%100)%10));
}

void read_all(void)
{
     a = adc_conv(3);  //Prints value of White Line Sensor1
```

```
    b = adc_conv(2);  //Prints Value of White Line Sensor2
    c = adc_conv(1);  //Prints Value of White Line Sensor3

    print_sensor(1,2,3);    //WL Sensor a-L 1
    print_sensor(1,6,2);    //WL Sensor b-M 2
    print_sensor(1,10,1);   //WL Sensor c-R 3
}
```

PROGRAM

LIBRARY

firebird_atmega2560.h

```
#ifndef FIREBIRDV_ATMEGA2560_H_
#define FIREBIRDV_ATMEGA2560_H_

#define IR_PORT PORTF
#define IR_SENSOR1 4

#define INT PORTE
#define POSENL 4
#define POSENR 5
#define BUTTON 7
#define MAX_DIGITS 7

#define MOTOR_PORT_DIR DDRA
#define MOTOR_PWM_PORT_DIR DDRL
#define MOTOR_PORT PORTA
#define MOTOR_PWM_PORT PORTL
#define LFWD 1
#define LBWD 0
#define RFWD 2
#define RBWD 3
#define LEFT 3
#define RIGHT 4

#define BUZZER 3
#define BUZZER_PORT PORTC

#define LCD_BUZZER_DIR DDRC
#define LCD_COL 16
#define LCD_ROW 2
#define LCD_PORT PORTC
#define RS 0
#define RW 1
#define EN 2

#define LED_BAR_GRAPH_PORT PORTJ
#define LED_BAR_GRAPH_DIR DDRJ

void lcd_print_test(unsigned char,unsigned char,unsigned char);
```

```
void firebird_init(void);
void velocity_control(void);
void motorspeed_config(void);
void velocity(void);
void lcd_print_dist(unsigned char,unsigned char);
void dis_data_op(void);
void interrupt_begin(void);
void read_all(void);
void print_sensor(unsigned char,unsigned char,unsigned char);
void adc_begin(void);
unsigned int adc_conv(unsigned char); // enter ADC number
void motor_begin(void);
void motion_set(unsigned char);
void forward(void);
void backward(void);
void stop(void);
void left(void);
void softleft(void);
void softleft2(void);
void right(void);
void softright(void);
void softright2(void);
void buzzer_on(void);
void buzzer_off(void);
void ledbarlevel(unsigned char); //enter no of led bars to glow
void lcd_begin(void);
void lcd_clear(void); //clear lcd
void lcd_setcursor(unsigned char,unsigned char); //col,row
void lcd_cursor(void); // turn on cursor on lcd
void lcd_cursoroff(void); // turn off cursor on lcd
void lcd_ctrl(unsigned char);
void lcd_print(unsigned char*); // "Hello World!"
void lcd_printa(unsigned char); // enter ascii value directly
void mosfet_switch_config(void); //initialize white line sensors

unsigned char logic,stopper,unit,ADC_Value,flag;
unsigned long posl,posr;
unsigned int a,b,c,i;
int cnt;
long distance,lft,rgt;
char distance_data[MAX_DIGITS];
#endif
```

firebird_atmega2560.c

```
#define F_CPU 14745600 //working frequency of firebirdv
atmega2560 is 14.7456Mhz

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "firebirdv_atmega2560.h"

ISR(INT7_vect)
{
     if(stopper)
     {
          forward();
          stopper=0;
     }
     else
     {
          stop();
          stopper=1;
     }
     sei();
}

ISR(INT5_vect)
{
     posr++;
     sei();
}

ISR(INT4_vect)
{
     posl++;
     sei();
}

void firebird_init(void)
{
     lcd_begin();
     motor_begin();
     adc_begin();
     LED_BAR_GRAPH_DIR=0xff; //initialize led bar graph (8 leds)
     motorspeed_config();
     interrupt_begin();
     mosfet_switch_config();
}
```

```
void mosfet_switch_config(void)
{
     DDRG = DDRG | 0x04;
     PORTG = PORTG & 0xfb;
}
void motorspeed_config(void)
{
     TCCR5A=0xa1;
     TCCR5B=0x0b;

     OCR5AL=0xff; //full velocity left motor 100% duty cycle
     OCR5BL=0xff; //full velocity right motor 100% duty cycle

     MOTOR_PWM_PORT_DIR=(1<<LEFT)|(1<<RIGHT);//pwm pins as
output
     lft=255;
     rgt=255;
}
void velocity(void)
{
     OCR5AL=lft;
     OCR5BL=rgt;
}
void velocity_control(void)
{
     if(posr>posl)
     {
          if(lft<245)
          lft+=1;
          else
          rgt-=1;
     }
     else
     {
          if(rgt<245)
          rgt+=1;
          else
          lft-=1;
     }
     velocity();
}
void lcd_print_dist(unsigned char col,unsigned char row)
{
     lcd_setcursor(col,row);
     distance=((posl + posr)/2)*0.544;
     dis_data_op();
     for(cnt=MAX_DIGITS-1;cnt>=0;cnt--)
```

```
	{
		logic+=distance_data[cnt];
		if(logic!=0)
		{
			lcd_printa(48+(distance_data[cnt]));
			if(unit==1 && cnt==2)
			{
				lcd_printa(0x2E); //ascii of decimal point
			}
		}
	}
	logic=0;
	if(unit)
	lcd_print("m");
	else
	lcd_print("cm");
}

void dis_data_op(void)
{
	//distance_data[6]=(distance%10000000)/1000000;
	//distance_data[5]=(distance%1000000)/100000;
	distance_data[4]=(distance%100000)/10000;
	distance_data[3]=(distance%10000)/1000;
	distance_data[2]=(distance%1000)/100;
	distance_data[1]=(distance%100)/10;
	distance_data[0]=(distance%10);
	if(distance_data[2]>0)
	{
		unit=1;
	}
}

void interrupt_begin(void)
{
	cli();
	INT|=(1<<POSENL)|(1<<POSENR)|(1<<BUTTON); //activate pull-
up resistor for both position encoders as well as for pushbutton
	EICRB|=(1<<ISC71)|(1<<ISC51)|(1<<ISC41); //falling edge
interrupt
	EIMSK|=(1<<INT7)|(1<<INT5)|(1<<INT4);
	EIFR|=(1<<INTF7); //requires clearing of interrupt flag for
edge trigerred interrupts
	//intf7 is 1 when isr is running and 0 after finishing isr
	sei();
}
```

```
void adc_begin(void)
{
     ADCSRA = 0x00;
     ADCSRB = 0x00;      //MUX5 = 0
     ADMUX = 0x20;       //Vref=5V external --- ADLAR=1 ---
MUX4:0 = 0000
     ACSR = 0x80;
     ADCSRA = 0x86;      //ADEN=1 --- ADIE=1 --- ADPS2:0 = 1 1 0
}

unsigned int adc_conv(unsigned char Ch)
{
          unsigned char a;
          if(Ch>7)
          {
               ADCSRB = 0x08;
          }
          Ch = Ch & 0x07;
          ADMUX= 0x20| Ch;
          ADCSRA = ADCSRA | 0x40;       //Set start conversion
bit
          while((ADCSRA&0x10)==0); //Wait for ADC conversion to
complete
          a=ADCH;
          ADCSRA = ADCSRA|0x10; //clear ADIF (ADC Interrupt
Flag) by writing 1 to it
          ADCSRB = 0x00;
          return a;
}

void print_sensor(unsigned char row,unsigned char
coloumn,unsigned char channel)
{
     ADC_Value = adc_conv(channel);
     lcd_setcursor(coloumn,row);
     lcd_printa(48+((ADC_Value/100)%10));
     lcd_printa(48+((ADC_Value%100)/10));
     lcd_printa(48+((ADC_Value%100)%10));
}

void read_all(void)
{
     a = adc_conv(3);  //Prints value of White Line Sensor1
     b = adc_conv(2);  //Prints Value of White Line Sensor2
     c = adc_conv(1);  //Prints Value of White Line Sensor3

     print_sensor(1,2,3);    //WL Sensor a-L 1
```

```
     print_sensor(1,6,2);    //WL Sensor b-M 2
     print_sensor(1,10,1);   //WL Sensor c-R 3
}

void forward(void)
{
     motion_set((1<<LFWD)|(1<<RFWD));
}
void backward(void)
{
     motion_set((1<<LBWD)|(1<<RBWD));
}
void left(void)
{
     motion_set((1<<RFWD)|(1<<LBWD));
}
void softleft2(void)
{
     motion_set((1<<LBWD));
}
void right(void)
{
     motion_set((1<<LFWD)|(1<<RBWD));
}
void softright2(void)
{
     motion_set((1<<RBWD));
}
void softleft(void)
{
     motion_set((1<<RFWD));
}
void softright(void)
{
     motion_set((1<<LFWD));
}
void stop(void)
{
     motion_set(0x00);
}

void motion_set(unsigned char data)
{
     MOTOR_PORT= ( (MOTOR_PORT & 0xf0) | (data) );
}

void motor_begin(void)
```

```
{
     MOTOR_PORT_DIR|=(1<<LFWD)|(1<<LBWD)|(1<<RFWD)|(1<<RBWD);
//initialize inputs for left and right dc-motors connected to
L293D IC
     MOTOR_PORT&=0xf0; //No direction of movement to DC motors
at startup
     MOTOR_PWM_PORT_DIR|=(1<<LEFT)|(1<<RIGHT); //initialize
velocity/PWM inputs for left and right dc-motors connected to
L293D IC
     MOTOR_PWM_PORT&=0xe7;
     MOTOR_PWM_PORT|=0x18;
}

void buzzer_on(void)
{
     BUZZER_PORT|=1<<BUZZER;
}

void buzzer_off(void)
{
     BUZZER_PORT&=(~(1<<BUZZER));
}

void ledbarlevel(unsigned char level)
{
     switch(level)
     {
          case 0:
          LED_BAR_GRAPH_PORT=0x00;
          break;
          case 1:
          LED_BAR_GRAPH_PORT=0x01;
          break;
          case 2:
          LED_BAR_GRAPH_PORT=0x03;
          break;
          case 3:
          LED_BAR_GRAPH_PORT=0x07;
          break;
          case 4:
          LED_BAR_GRAPH_PORT=0x0f;
          break;
          case 5:
          LED_BAR_GRAPH_PORT=0x1f;
          break;
          case 6:
          LED_BAR_GRAPH_PORT=0x3f;
```

```
            break;
            case 7:
            LED_BAR_GRAPH_PORT=0x7f;
            break;
            case 8:
            LED_BAR_GRAPH_PORT=0xff;
            break;
            default:
            LED_BAR_GRAPH_PORT=0x00;
            break;
      }
}

void lcd_begin(void)
{
      LCD_BUZZER_DIR=0xff; //initialize buzzer and lcd as output
      LCD_PORT=0x00; //No input to lcd
      _delay_ms(5);

      lcd_ctrl(0x33);
      lcd_ctrl(0x32);
      lcd_ctrl(0x28);

      lcd_ctrl(0x0C); //cursor off
      lcd_ctrl(0x06); //move l to r
      lcd_ctrl(0x01); //clear
      lcd_ctrl(0x80); //starting address till 8f
}
void lcd_clear(void)
{
      lcd_ctrl(0x01);
      lcd_ctrl(0x80);
}
void lcd_setcursor(unsigned char a,unsigned char b)
{
      //a is col and b is row
      if(b==0)
      lcd_ctrl((0x80)+a);
      if(b==1)
      lcd_ctrl((0xc0)+a); //add of row 1 c0 to cf
      if(LCD_ROW==4 && LCD_COL==20)
      {
            if(b==2)
            lcd_ctrl((0x94)+a);
            if(b==3)
            lcd_ctrl((0xd4)+a);
      }
```

```
}

void lcd_cursor(void)
{
     lcd_ctrl(0x0E); //cursor on
}
void lcd_cursoroff(void)
{
     lcd_ctrl(0x0C);
}
void lcd_ctrl(unsigned char cmd)
{
     LCD_PORT&=~(1<<RS); //RS is  0 for commands
     LCD_PORT=(LCD_PORT & 0x0f) | (cmd & 0xf0); //sending upper
nibble
     LCD_PORT|=1<<EN;
     _delay_ms(1);
     LCD_PORT&=~(1<<EN);
     _delay_ms(1);
     cmd=cmd<<4;
     LCD_PORT=(LCD_PORT & 0x0f) | (cmd & 0xf0); //sending lower
nibble
     LCD_PORT|=1<<EN;
     _delay_ms(1);
     LCD_PORT&=~(1<<EN);
     _delay_ms(1);
}
void lcd_print(unsigned char *data)
{
     LCD_PORT|=1<<RS; //RS is 1 for data
     while(*data !='\0')
     {
          lcd_printa(*data);
          *data++;
     }
}
void lcd_printa(unsigned char ascii) //ascii value directly
{
     LCD_PORT|=1<<RS;
     LCD_PORT=(LCD_PORT & 0x0f) | (ascii & 0xf0); //sending
upper nibble
     LCD_PORT|=1<<EN;
     _delay_ms(1);
     LCD_PORT&=~(1<<EN);
     _delay_ms(1);
     ascii=ascii<<4;
```

```
     LCD_PORT=(LCD_PORT & 0x0f) | (ascii & 0xf0); //sending
lower nibble
     LCD_PORT|=1<<EN;
     _delay_ms(1);
     LCD_PORT&=~(1<<EN);
     _delay_ms(1);
}
```

MAIN PROGRAM

main.c

```
#define F_CPU 14745600 //working frequency of firebirdv
atmega2560 is 14.7456Mhz

#include <avr/io.h>
#include "firebirdv_atmega2560.h"
#include <util/delay.h>
#include <avr/interrupt.h>

unsigned char flag=2;

void turn_left()
{
          for(i=0;i<20;i++)
//loop for 90'
          {
               lcd_print_dist(5,0);
               lft=150;
               rgt=150;
               velocity();
               left();
               _delay_ms(90);
               stop();
               _delay_ms(20);
               read_all();
               if(b<11)
//if b is less then 7 make flag 0
               {
                    flag=0;
                    break;
               }
          }
}
void turn_right()
{
```

```
          for(i=0;i<40;i++)             //loop for 180'
          {
               lcd_print_dist(5,0);
               lft=150;
               rgt=150;
               velocity();
               right();
               _delay_ms(90);
               stop();
               _delay_ms(20);
               read_all();
               if(b<11)
//if b is less then 7 make flag 1
               {
                    flag=1;
                    break;
               }
          }

}

int main(void)
{
     firebird_init();
     lcd_print("Dist:");
    while (1)
    {
          lcd_print_dist(5,0);
          while(b<=12)
//until the mid white line sensor reading is less then or equal
to 7
          {
               lcd_print_dist(5,0);
               /*if (a==c && a==b)
               {
                    lcd_print_dist(5,0);
                    lft=250;
                    rgt=250;
                    velocity();
                    left();
                    _delay_ms(50);
                    stop();
                    _delay_ms(20);
                    read_all();
               }
               else
               {*/
```

```
                    lcd_print_dist(5,0);
                    lft=125;
                    rgt=130;
                    velocity();
                    forward();
                    _delay_ms(50);
                    read_all();
                    /*if(b>100)
//if b is grater then 80 then check flag if flag is 0 then turn
left and 1 then turn right
                    {
                         if(flag==0)
                         turn_left();
                         else if(flag==1)
                         turn_right();
                    }*/
               //}

          }
          while(b>100)
//until the mid white line sensor reading is grater then 80
          {
               lcd_print_dist(5,0);
               if(flag==0 || flag==2)
               turn_left();
               if(flag==1 || flag==2)
               turn_right();
          }
          while(b>12 && b<100)
//until the mid white line sensor reading is grater then 6 and
less then 80
          {
               lcd_print_dist(5,0);
               while(a<c && b>12 && b<100)
//take soft left until sensor a is less then sensor c
               {
                    lcd_print_dist(5,0);
                    lft=200;
                    rgt=200;
                    velocity();
                    softleft();
                    _delay_ms(100);
                    stop();
                    _delay_ms(20);
                    read_all();
               }
```

```
                while(c<a && b>12 && b<100)
//take soft right until sensor a is less then sensor c
                {
                     lcd_print_dist(5,0);
                     lft=200;
                     rgt=200;
                     velocity();
                     softright();
                     _delay_ms(100);
                     stop();
                     _delay_ms(20);
                     read_all();
                }
                for(i=0;i<20 && a==c && a==b;i++)
//when no white line is ahead turn left 90' or till white line
is detected
                {
                     lcd_print_dist(5,0);
                     lft=250;
                     rgt=250;
                     velocity();
                     left();
                     _delay_ms(80);
                     stop();
                     _delay_ms(20);
                     read_all();
                }
                for(i=0;i<40 && a==c && a==b;i++)
//when no white line is ahead turn right 180' or till white line
is detected
                {
                     lft=250;
                     rgt=250;
                     velocity();
                     right();
                     _delay_ms(80);
                     stop();
                     _delay_ms(20);
                     read_all();
                }
                read_all();
                stop();
          }
          read_all();
    }
}
```

CONCLUSION

This project made us work with Atmega2560 and helped us learn to use datasheet. We also learned about IR sensors and position encoder and their importance in such projects. We realized that working with FireBird robot was fun and such project finds application into many real time problems.