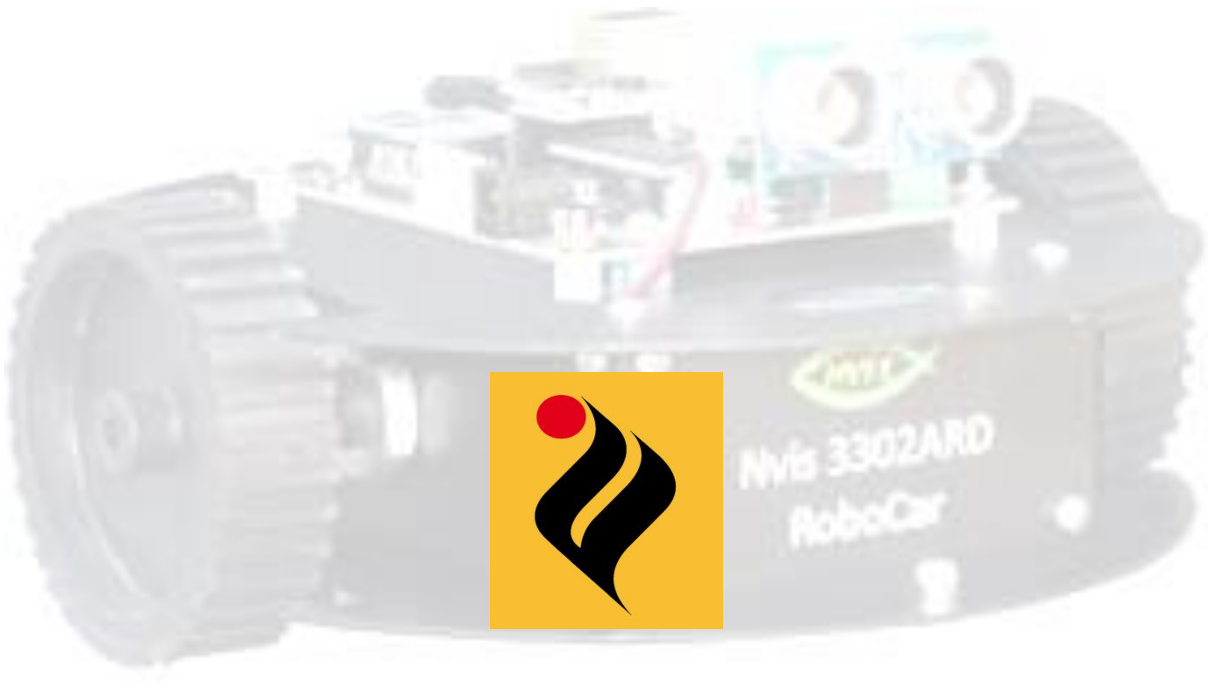
BUGGY CHALLENGE

A Project Report
submitted for UTA 011-Engineering Design –III
(Second Year)

Submitted by
101503207 SHIVAM SABHARWAL
101503208 SHIVAM SHARMA
101503209 SHIVANGI SAREEN
101503210 SHIVEN MEHRU
Group: COE-9



ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT,
COMPUTER SCIENCE ENGINEERING DEPARTMENT,
THAPAR UNIVERSITY, PATIALA-147004, PUNJAB
INDIA
May 2017.

INTRODUCTION

NATURE AND SCOPE:

Engineering Design III) is a very innovative initiative by Thapar University in collaboration with Trinity University, Dublin. All the faculty members have put in great efforts to impart the understanding of Robotic Buggy, various sensors used like the Ultrasonic sensor, Infrared Sensor, XBee and many others.

We had to program the Robotic Buggy using Arduino Software (IDE) to control it and perform the operations of moving it forward, backward, turn left, turn right, ,counting number of obstacles involved, following a given black line, communicate two XBee modules.

The wireless interfacing from buggy (NVIS 3020 Robo car) to buggy should be done using Zigbee.

Two challenges were given to us:

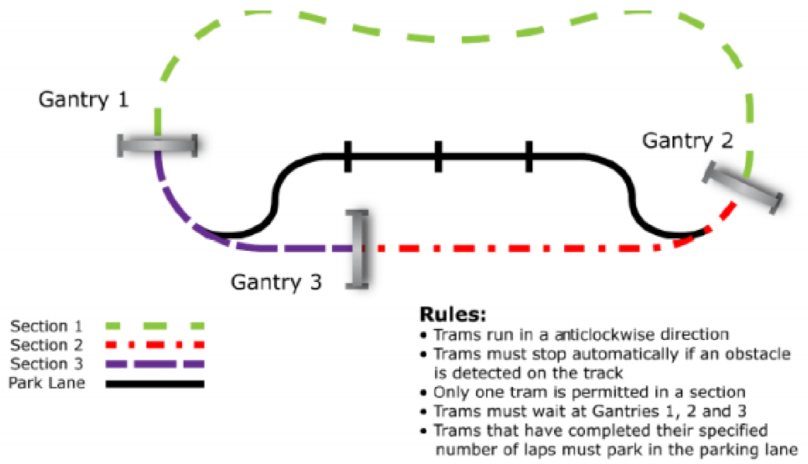
Challenge Silver: The challenge included moving a single buggy on the defined track with predefined events and conditions. The student's teams were required to design buggy along with sensors as per the requirement of challenge and design an algorithm/ program for execution of the defined challenge through buggy. The buggy that started once should not involve any human interface during full round of defined challenge.

This challenge required IR sensor so that our buggy could follow the black strip and move in anticlockwise direction. The buggy had to move from the parking area and complete three rounds and count 9 obstacles in its way using ultrasonic sensor .As soon as the counter is updated at 9 it should come in the parking area.

Challenge Gold: This challenge included moving two buggies on the defined track with predefined events and conditions. Also the buggies should communicate among themselves through Xbee under half duplex mode. The buggy once started should not involve any human interface during full round of defined challenge.

This challenge required communication through ZigBee. As soon as the previous buggy came at the parking area it should signal another buggy. Now this buggy should move in opposite direction and repeat the same procedure followed by buggy1.

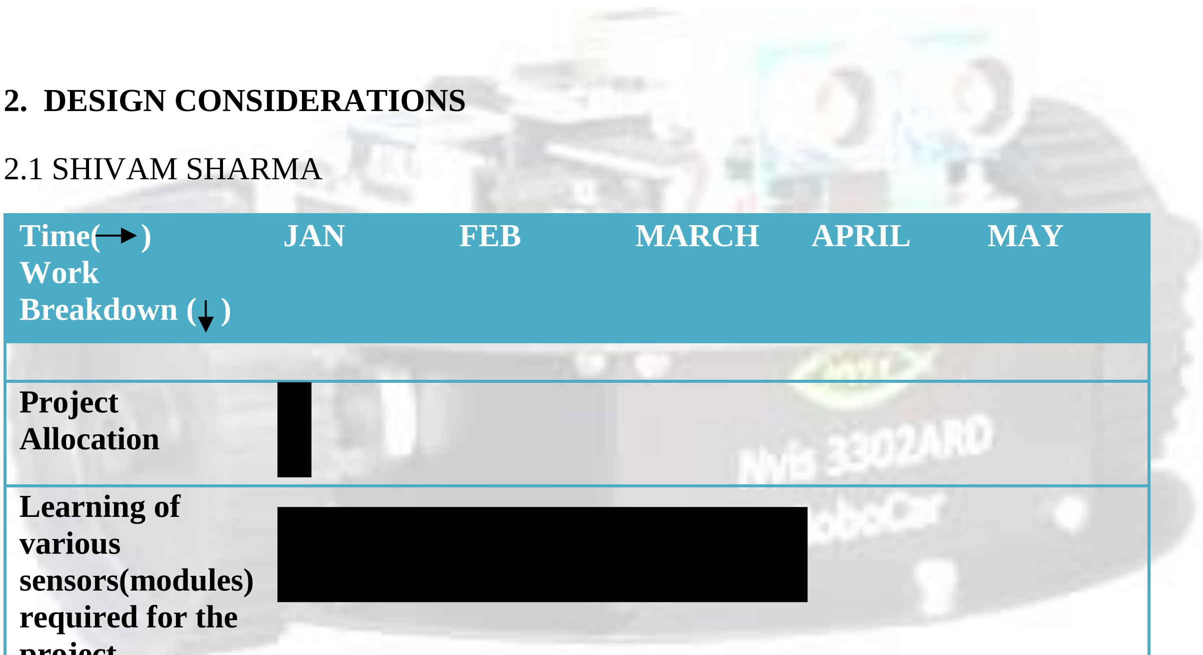
The path is shown below:



Rules:
- Trams run in a anticlockwise direction
- Trams must stop automatically if an obstacle is detected on the track
- Only one tram is permitted in a section
- Trams must wait at Gantries 1, 2 and 3
- Trams that have completed their specified number of laps must park in the parking lane

2. DESIGN CONSIDERATIONS

2.1 SHIVAM SHARMA



Time(→) Work Breakdown (↓)	JAN	FEB	MARCH	APRIL	MAY
Project Allocation	■				
Learning of various sensors(modules) required for the project	■	■	■		
Software Implementation of the project				■	
Testing and correction of the project.				■	
Designing the model				■	
Result Verification					■

2.2.2 He has given his 100% while making the code and the report. Throughout the semester, we have learnt about different sensors, their applications and has given valuable inputs. He helped with the logic of the program.

2.2 SHIVAM SABHARWAL

Time(→) Work Breakdown (↓)	JAN	FEB	MARCH	APRIL	MAY
Project Allocation	█				
Learning of various sensors(modules) required for the project	█	█	█		
Software Implementation of the project				█	
Testing and correction of the project.				█	
Designing the model				█	
Result Verification					█

2.2.2 He has given her 100% while making the report and the code. Throughout the semester, we have learnt about different sensors, their applications and has given valuable inputs. He helped in the programming and analysis of the code.

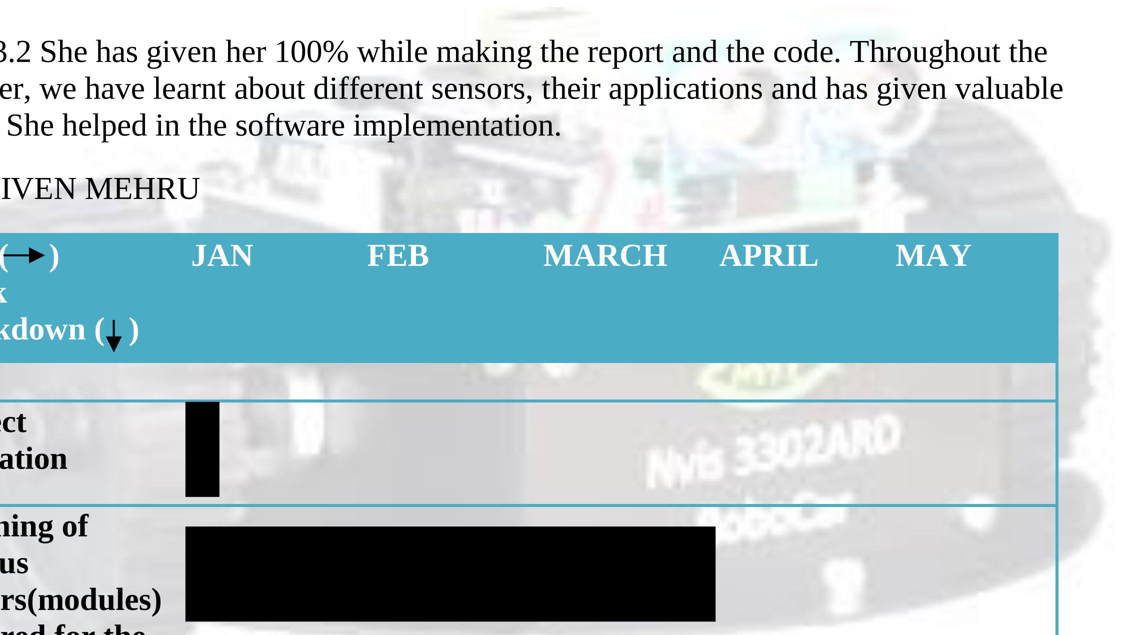
2.3 SHIVANGI SAREEN

Time(→) Work Breakdown (↓)	JAN	FEB	MARCH	APRIL	MAY
Project Allocation	█				
Learning of various sensors(modules)	█	█	█		

required for the project					
Software Implementation of the project				█	
Testing and correction of the project.				█	
Designing the model				█	
Result Verification					█

2.3.2 She has given her 100% while making the report and the code. Throughout the semester, we have learnt about different sensors, their applications and has given valuable inputs. She helped in the software implementation.

2.4 SHIVEN MEHRU



Time(→) Work Breakdown (↓)	**JAN**	**FEB**	**MARCH**	**APRIL**	**MAY**
Project Allocation	█				
Learning of various sensors(modules) required for the project	█	█	█		
Software Implementation of the project				█	
Testing and correction of the project.				█	
Designing the model				█	
Result Verification					█

2.3.2 He has given his 100% while making the report and the code. Throughout the semester, we have learnt about different sensors, their applications and has given valuable inputs throughout.

COMPONENT DETAILS

- **ARDUINO UNO**

What is a Microcontroller?

Wikipedia1 says: A micro-controller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals The important part for us is that a micro-controller contains the processor (which all computers have) and memory, and some input/output pins that you can control. (Often called GPIO - General Purpose Input Output Pins).

Technical specs:

- Microcontroller ATmega328
- Operating Voltage 5V
- Input Voltage (recommended) 7-12V
- Input Voltage (limits) 6-20V Digital
- I/O Pins 14 (of which 6 provide PWM output)
- Analog Input Pins 6
- DC Current per I/O Pin 40 Ma
- DC Current for 3.3V Pin 50 mA
- Clock speed 16 mhz
- Flash Memory 32 KB of which 0.5 KB.
- SRAM 2 KB
- EEPROM 1 KB

PIN CONFIGURATION:

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

The power pins are as follows:

• **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

• **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.

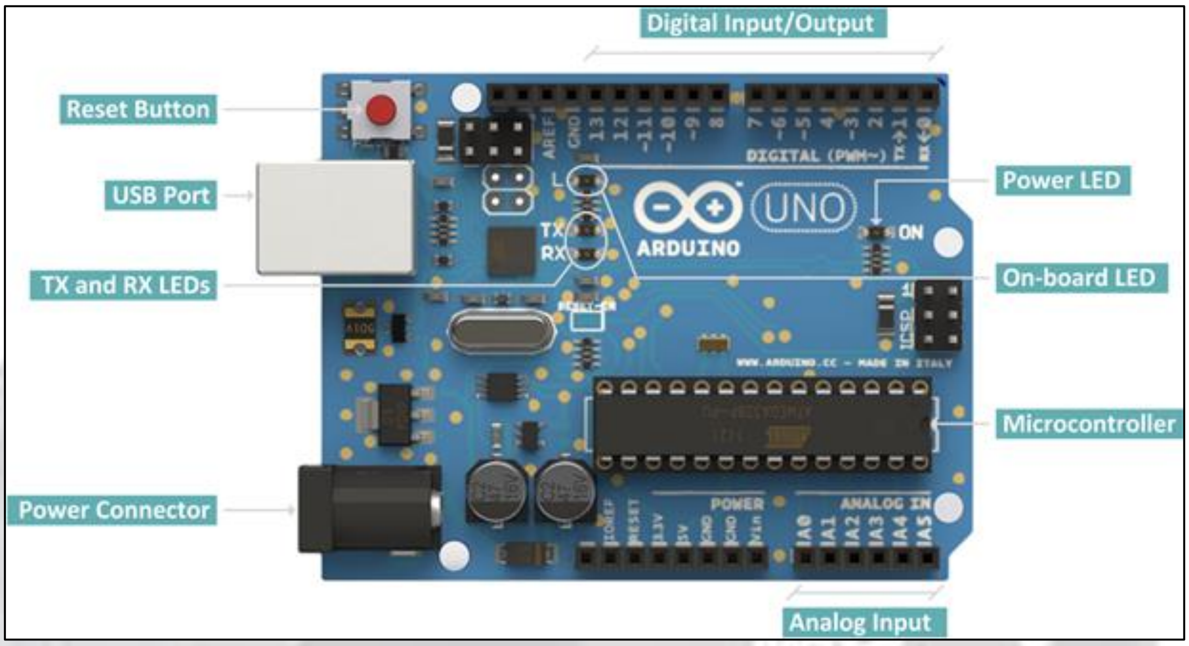
• **3.3V**. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

• **GND**. Ground pins.

• **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip. • External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

• **PWM:** 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite () function.

• **LED: 13**. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off



- **ULTRASONIC SENSOR (HC-SR04):**

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. From 2cm to 400 cm or 1” to 13 feet. It operation is not affected by sunlight or black material like Sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module.The HC-SR04 Ultrasonic Sensor is a very affordable proximity/distance sensor that has been used mainly for object avoidance in various robotics projects. It essentially gives your Arduino eyes / spacial awareness and can prevent your robot from crashing or falling off a table. It has also been used in turret applications, water level sensing, and even as a parking sensor.

- **INFRARED SENSOR:**

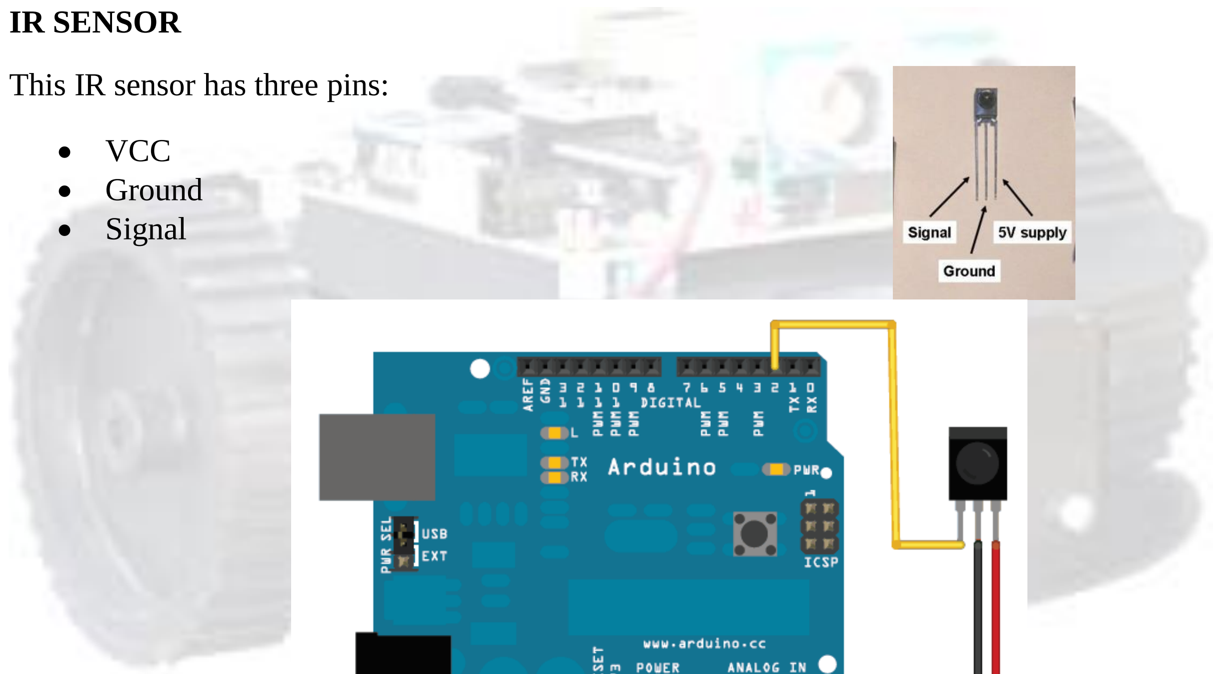
We used IR Transmitters and IR receivers also called photo diodes. They are used for sending and receiving light. IR transmits infrared lights. When infrared rays fall on white surface, it's reflected back and caught by photodiodes which generates some voltage changes. When IR light falls on a black surface, light is absorbed by the black surface and no rays are reflected back, thus photo diode does not receive any light or rays.

- **DC MOTOR:**

A DC motor converts direct current electrical energy into mechanical energy, with a help of an oscillating electric field. This oscillation is done with the help of some internal mechanism, either electronic or electromechanical. These motors run on a high voltage, which is more than the voltage used in electronic circuits. L293D IC acts as an interface and also solves the above-mentioned problem by acting as an amplifier.

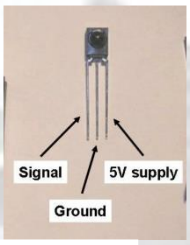
CONNECTION DIAGRAM WITH ARDUINO

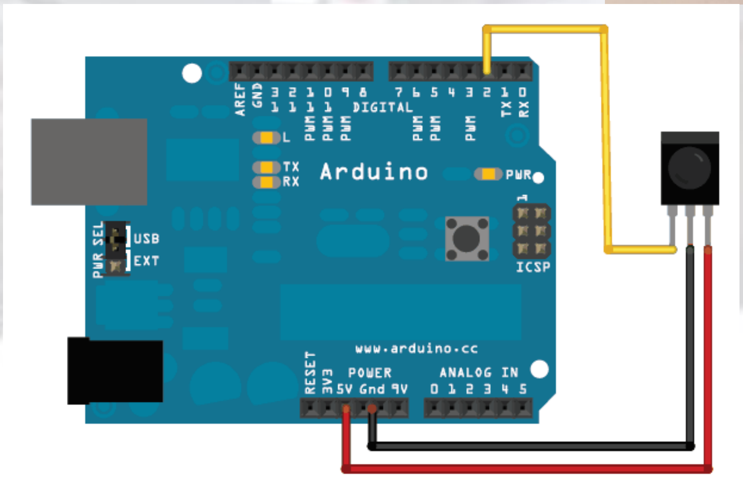
IR SENSOR



This IR sensor has three pins:

- VCC
- Ground
- Signal

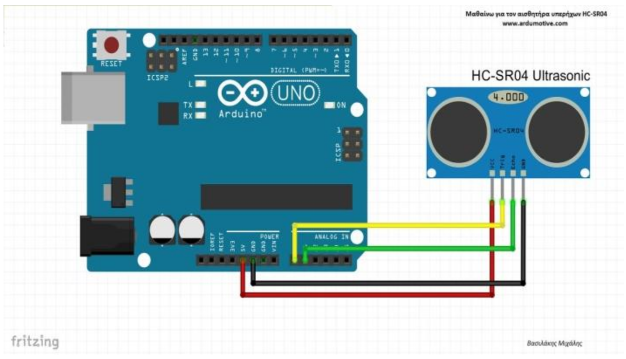




ULTRASONIC SENSOR

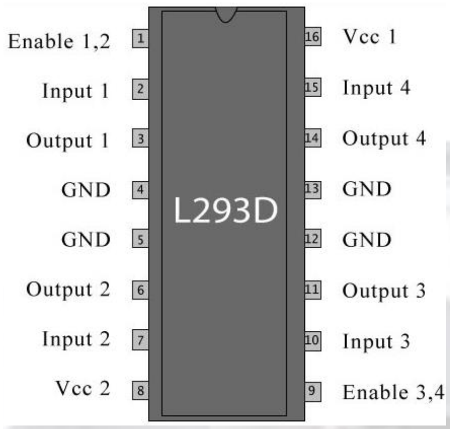
Pins:

- VCC: +5VDC
- Trig : Trigger (INPUT)
- Echo: Echo (OUTPUT)
- GND: GND



DC MOTOR:

The IC has 16 terminals, 8 on each side, and can operate two motors at a time. It has 2 H-bridge circuits inside of it. The detailed layout of the IC is given below:



Vcc1 is the internal voltage for the IC.

Vcc2 is the supply for the motor (max=36V).

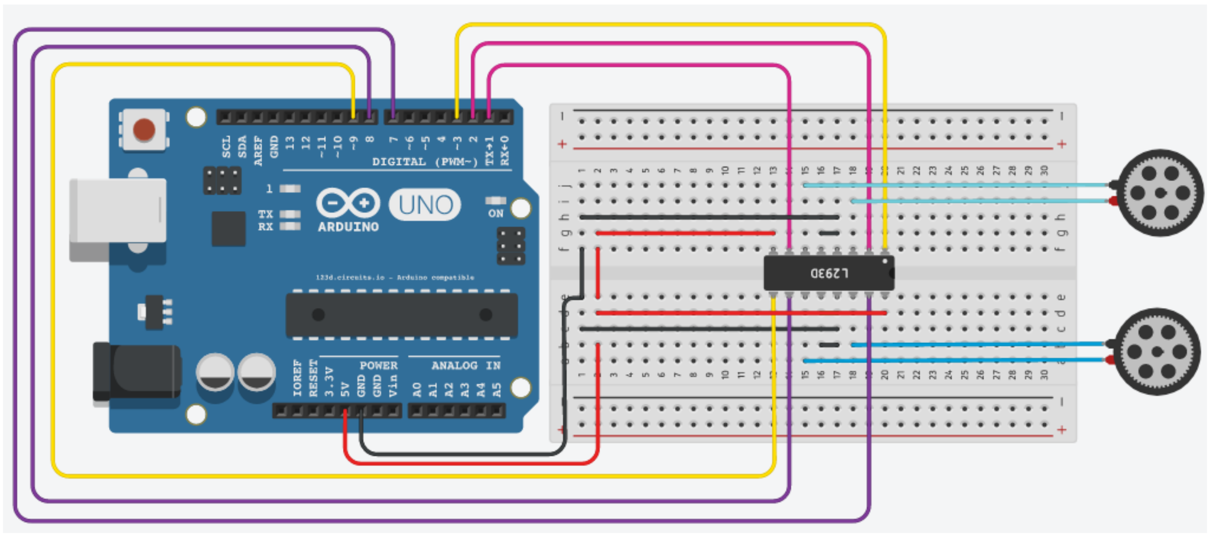
Enable acts as a switch, and has to be high for the motor to work.

Input receives the output signal given from the controller (Arduino), and Output gives the output to the DC motor.

GND need to be grounded.

Input 1 (pin 2)	Input 2 (pin 7)	Effect on motor
0	0	No rotation
0	1	Anticlockwise rotation
1	0	Clockwise rotation
1	1	No rotation

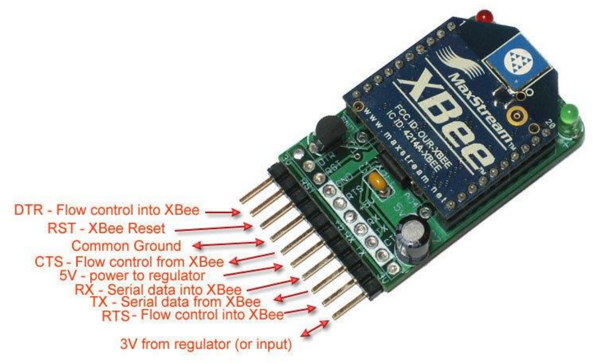
CIRCUIT DIAGRAM:

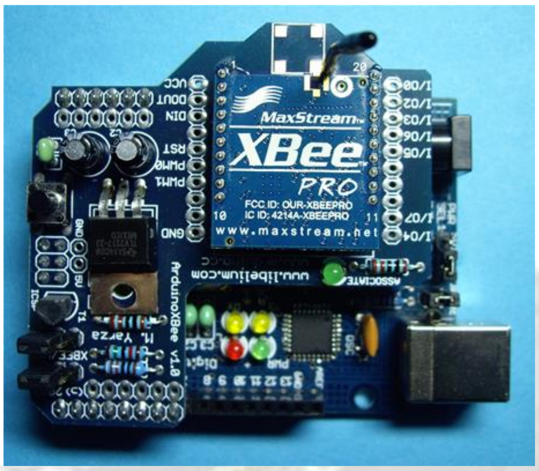


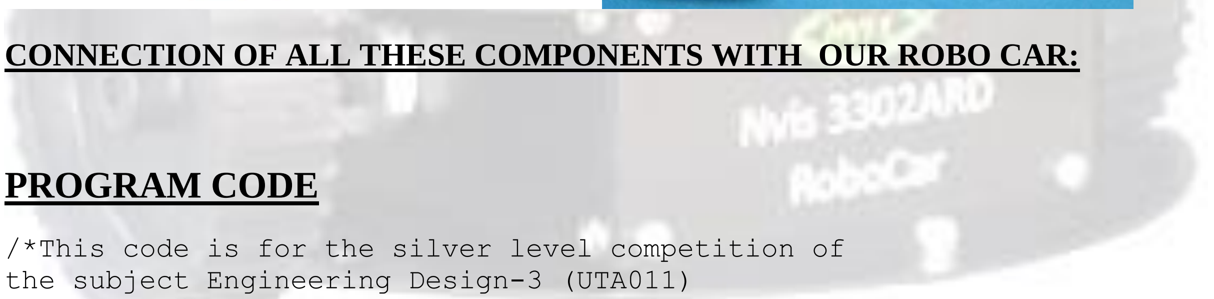
ZIGBEE

API and AT Modes:

The XBee modules can be configured in two ways: Transparent Mode (AT) and API Mode (API).In AT mode you are limited to point-to-point communication between two XBees. In API mode, we can trivally send and receive from both the COORDINATOR and many many XBees out in the world. Additionally, API mode will expose a variety of additional information encoded in each packet.


DTR - Flow control into XBee
RST - XBee Reset
Common Ground
CTS - Flow control from XBee
5V - power to regulator
RX - Serial data into XBee
TX - Serial data from XBee
RTS - Flow control into XBee
3V from regulator (or input)
MaxStream
XBee



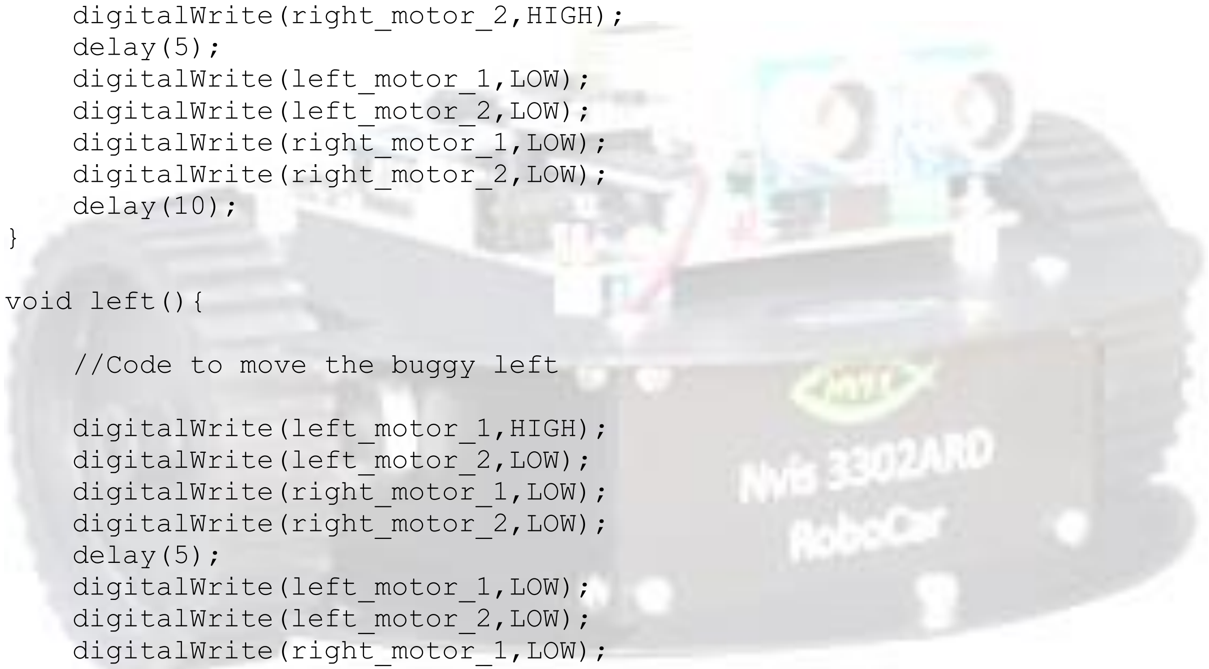


CONNECTION OF ALL THESE COMPONENTS WITH OUR ROBO CAR:

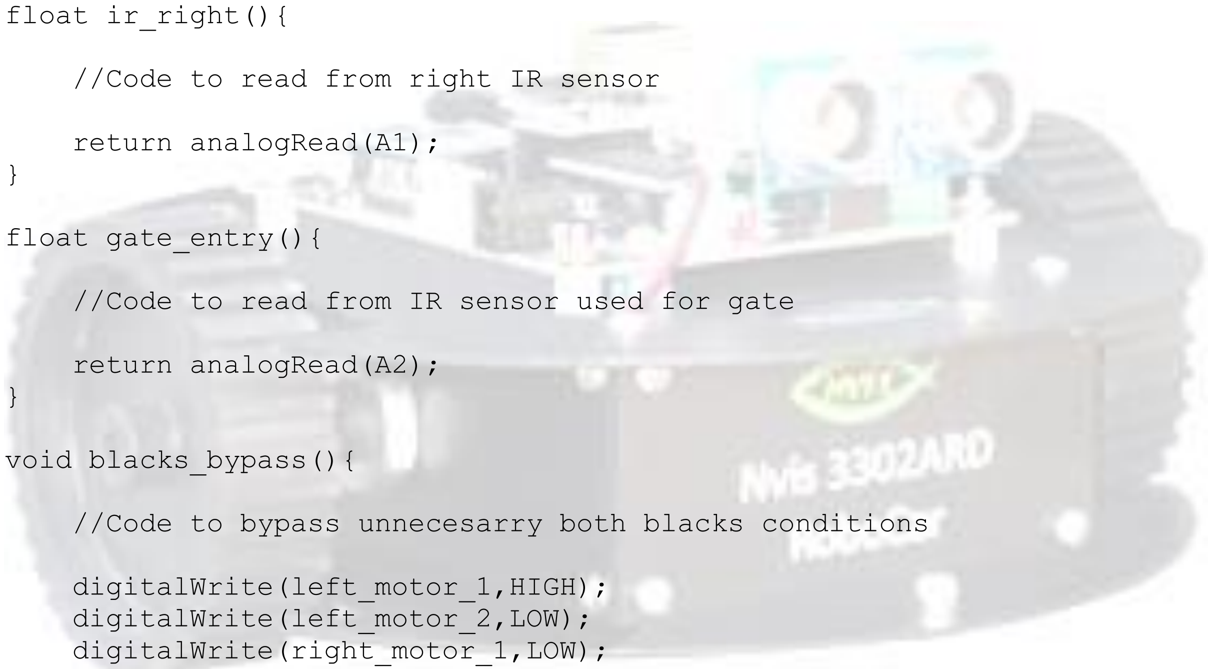
PROGRAM CODE

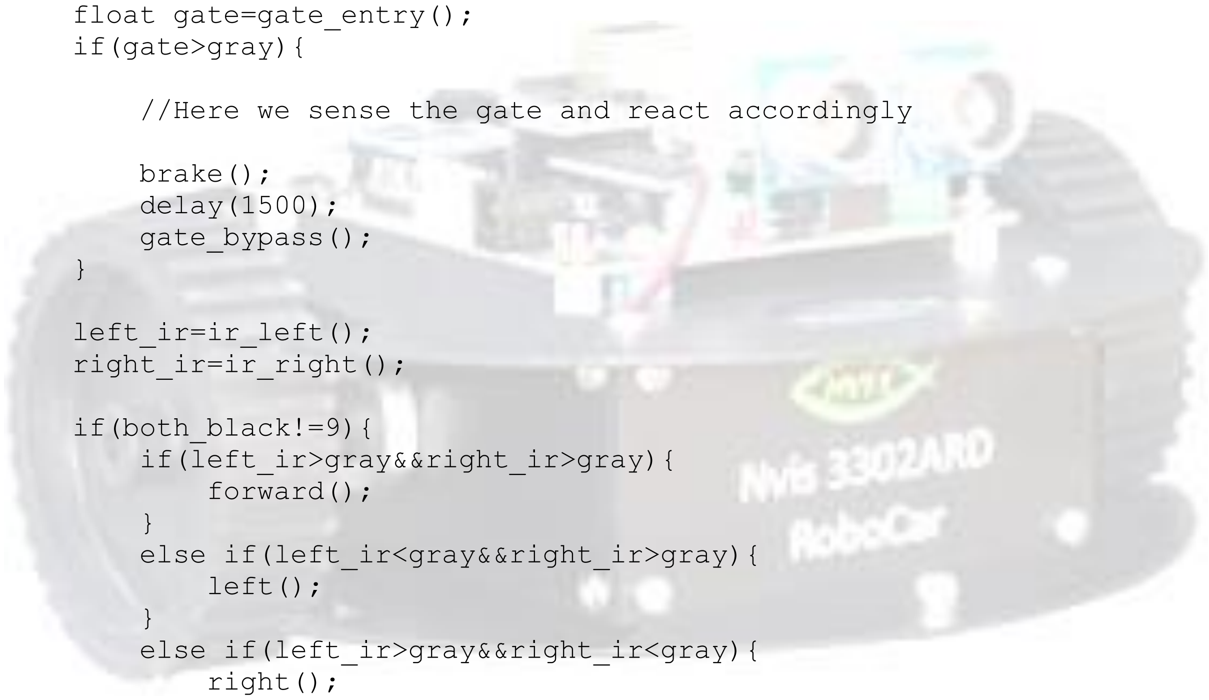
```
/*This code is for the silver level competition of
the subject Engineering Design-3 (UTA011)
for this to work this you need to put two IR sensors
beneath the buggy to sense the black line and
one IR sensor above the buggy to sense the gates in the path.

Initially both the sensors below are to be kept on white
or one of them on black and other on white and is started
behind the first line on parking lane then without
any intervention the buggy will complete one round
anti-clockwise and in the second round it will
enter the parking lane and stop on the third line.
*/

int left_motor_1=5;
int left_motor_2=6;
int right_motor_1=7;
int right_motor_2=8;
float duration;
int gray=400;
int both_black=0;
void setup()
```

```
{
    pinMode(left_motor_1,OUTPUT);
    pinMode(left_motor_2,OUTPUT);
    pinMode(right_motor_1,OUTPUT);
    pinMode(right_motor_2,OUTPUT);
    pinMode(A0,INPUT);              //This pin is for left IR sensor
    pinMode(A1,INPUT);              //This pin is for right IR sensor
    pinMode(A2,INPUT);              //This pin is for the IR used for
gate sensing
}

void forward(){

    //Code to move the buggy forward

    digitalWrite(left_motor_1,HIGH);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,HIGH);
    delay(5);
    digitalWrite(left_motor_1,LOW);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,LOW);
    delay(10);
}

void left(){

    //Code to move the buggy left

    digitalWrite(left_motor_1,HIGH);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,LOW);
    delay(5);
    digitalWrite(left_motor_1,LOW);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,LOW);
    delay(10);
}

void right(){

    //Code to move the buggy right

    digitalWrite(left_motor_1,LOW);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,HIGH);
    delay(2);
    digitalWrite(left_motor_1,LOW);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,LOW);
    delay(5);
}
```



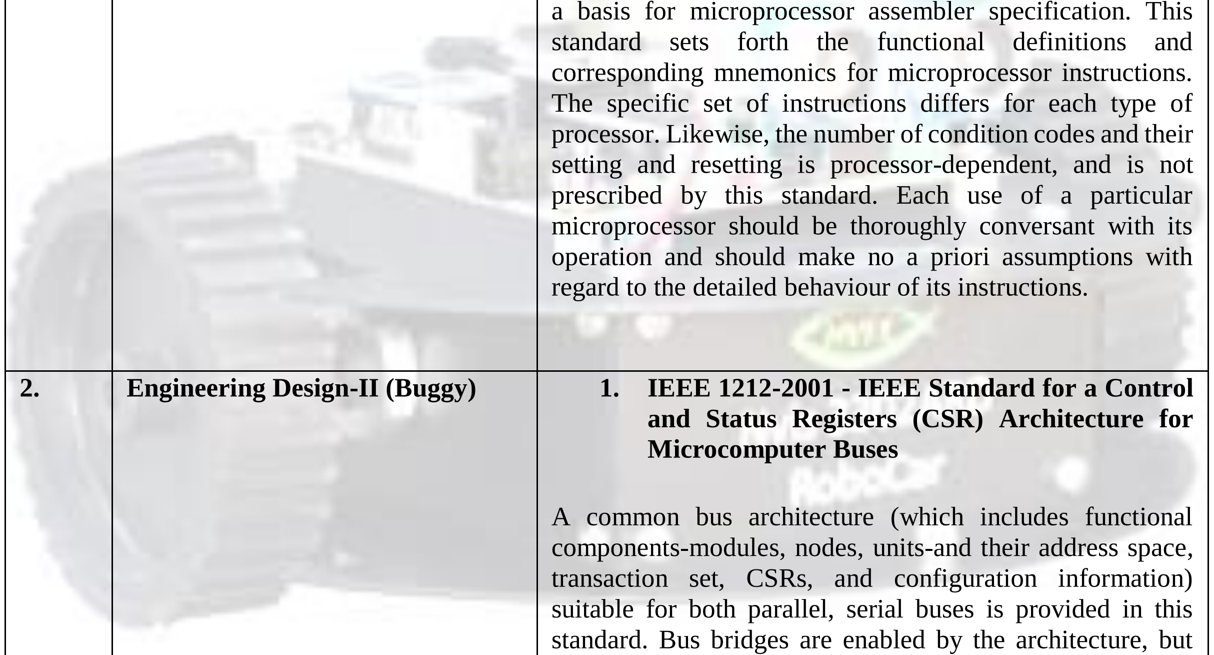
```
void brake(){

    //Code to stop the buggy

    digitalWrite(left_motor_1,LOW);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,LOW);
}

float ir_left(){

    //Code to read from left IR sensor

    return analogRead(A0);
}

float ir_right(){

    //Code to read from right IR sensor

    return analogRead(A1);
}

float gate_entry(){

    //Code to read from IR sensor used for gate

    return analogRead(A2);
}

void blacks_bypass(){

    //Code to bypass unnecesarry both blacks conditions

    digitalWrite(left_motor_1,HIGH);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,HIGH);
    delay(100);
    digitalWrite(left_motor_1,LOW);
    digitalWrite(left_motor_2,LOW);
    digitalWrite(right_motor_1,LOW);
    digitalWrite(right_motor_2,LOW);
    delay(50);
}

void gate_bypass(){

    //Code to bypass gate

    left_ir=ir_left();
    right_ir=ir_right();
    gate=gate_entry();
    while(gate>gray){
        if(left_ir>gray&&right_ir>gray){
            forward();
```





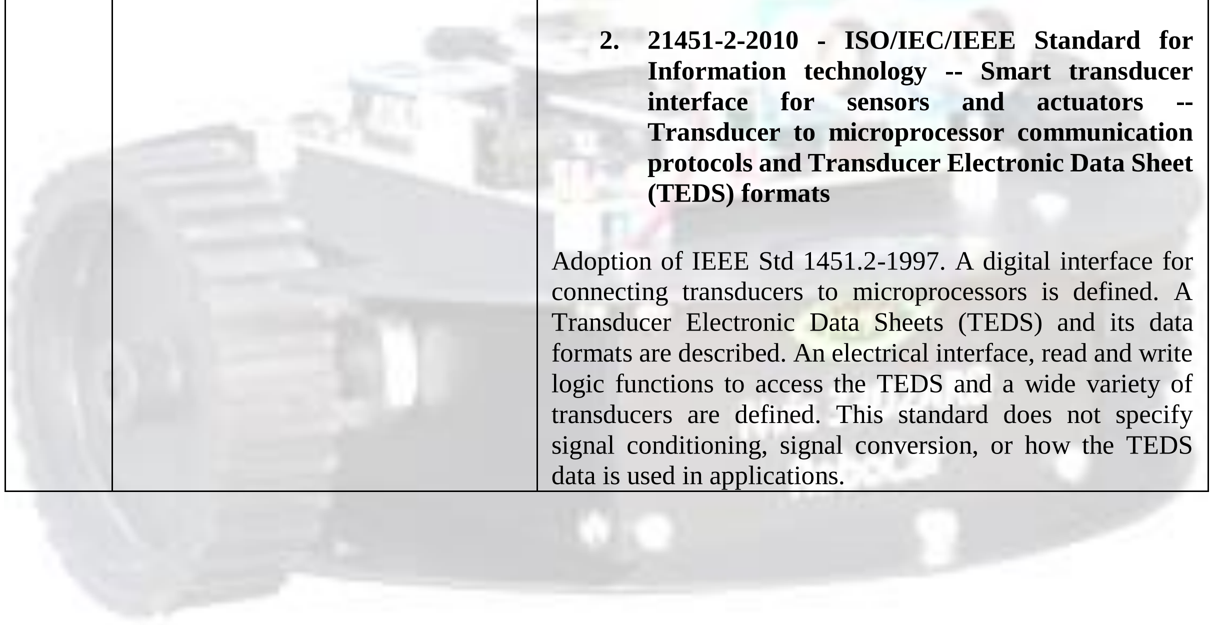
```
        }
        else if(left_ir<gray&&right_ir>gray){
            left();
        }
        else if(left_ir>gray&&right_ir<gray){
            right();
        }
        left_ir=ir_left();
        right_ir=ir_right();
        gate=gate_entry();
    }
}

void loop()
{
    float left_ir=ir_left();
    float right_ir=ir_right();

    float gate=gate_entry();
    if(gate>gray){

        //Here we sense the gate and react accordingly

        brake();
        delay(1500);
        gate_bypass();
    }

    left_ir=ir_left();
    right_ir=ir_right();

    if(both_black!=9){
        if(left_ir>gray&&right_ir>gray){
            forward();
        }
        else if(left_ir<gray&&right_ir>gray){
            left();
        }
        else if(left_ir>gray&&right_ir<gray){
            right();
        }
        else if(both_black==1&&left_ir<gray&&right_ir<gray){

            //This is for entering into the main lane from parking lane

            both_black++;
            digitalWrite(left_motor_1,HIGH);
            digitalWrite(left_motor_2,LOW);
            digitalWrite(right_motor_1,LOW);
            digitalWrite(right_motor_2,HIGH);
            delay(20);
            digitalWrite(left_motor_1,HIGH);
            digitalWrite(left_motor_2,LOW);
            digitalWrite(right_motor_1,LOW);
            digitalWrite(right_motor_2,LOW);
            delay(800);
        }
        else 
if(both_black!=5&&both_black!=1&&left_ir<gray&&right_ir<gray){

            //Here we bypass all the useless double bypass conditions

            both_black++;
            blacks_bypass();
        }
        else if(both_black==5&&left_ir<gray&&right_ir<gray){

            //This is for entering into the parking lane from main lane

            both_black++;
            digitalWrite(left_motor_1,HIGH);
            digitalWrite(left_motor_2,LOW);
            digitalWrite(right_motor_1,LOW);
            digitalWrite(right_motor_2,LOW);
            delay(700);
        }
    }
    else{
        brake();
    }
}
```

STANDARD USED



S. No.	Subject	Standards
1.	**Microprocessor and their Applications**	**1. 1754-1994 - IEEE Standard for a 32-bit Microprocessor Architecture** A 32-bit microprocessor architecture, available to a wide variety of manufacturers and users, is defined. The standard includes the definition of the instruction set, register model, data types, instruction op-codes, and coprocessor interface. **2. IEEE 694-1985 - IEEE Standard for Microprocessor Assembly Language** The intent of this standard is to name a common set of instructions used by most general purpose microprocessors, to provide rules for the naming of new instructions and the derivation of new mnemonics, and to establish assembly language conventions and syntax. It is intended to be used as a basis for microprocessor assembler specification. This standard sets forth the functional definitions and corresponding mnemonics for microprocessor instructions. The specific set of instructions differs for each type of processor. Likewise, the number of condition codes and their setting and resetting is processor-dependent, and is not prescribed by this standard. Each use of a particular microprocessor should be thoroughly conversant with its operation and should make no a priori assumptions with regard to the detailed behaviour of its instructions.
2.	**Engineering Design-II (Buggy)**	**1. IEEE 1212-2001 - IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses** A common bus architecture (which includes functional components-modules, nodes, units-and their address space, transaction set, CSRs, and configuration information) suitable for both parallel, serial buses is provided in this standard. Bus bridges are enabled by the architecture, but their details are beyond its scope. Configuration information is self-administered by vendors, organizations based upon IEEE Registration Authority company id. **2. 1451.2-1997 - IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats** A digital interface for connecting transducers to microprocessors is defined. A TEDS and its data formats are described. An electrical interface, read and write logic functions to access the TEDS and a wide variety of transducers are defined. This standard does not specify signal conditioning, signal conversion, or how the TEDS data is used in applications.

3.	**Microcontrollers and Embedded Systems**	**1. IEEE 1275.1-1994 - IEEE Standard for Boot (Initialization Configuration) Firmware: Instruction Set Architecture (ISA) Supplement for IEEE 1754** Firmware is the read-only-memory (ROM)-based software that controls a computer between the time it is turned on and the time the primary operating system takes control of the machine. Firmware's responsibilities include testing and initializing the hardware, determining the hardware configuration, loading (or booting) the operating system, and providing interactive debugging facilities in case of faulty hardware or software. The core requirements and practices specified by IEEE Std 1275-1994 must be supplemented by system-specific requirements to form a complete specification for the firmware for a particular system. This standard establishes such additional requirements pertaining to the instruction set architecture (ISA) defined by IEEE Std 1754-1944, IEEE Standard for a 32-bit Microprocessor Architecture. **2. 21451-2-2010 - ISO/IEC/IEEE Standard for Information technology -- Smart transducer interface for sensors and actuators -- Transducer to microprocessor communication protocols and Transducer Electronic Data Sheet (TEDS) formats** Adoption of IEEE Std 1451.2-1997. A digital interface for connecting transducers to microprocessors is defined. A Transducer Electronic Data Sheets (TEDS) and its data formats are described. An electrical interface, read and write logic functions to access the TEDS and a wide variety of transducers are defined. This standard does not specify signal conditioning, signal conversion, or how the TEDS data is used in applications.



RESULT

We have become familiar with the understanding of processor/ micro-controllers, their programming, controlling algorithms, Interfacing of sensors like Ultrasonic, IR, accelerometer, gyroscope,understanding wireless devices like Zigbee , their operation controlling using microcontroller and PC.The buggy1 could follow the path and was able to park itself after completing three rounds.