BUGGY CHALLENGE

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

Submitted by

Prathit Malik (Regd. No. : 101506128)
Priyum Bansal (Regd. No. : 101506130)
Rahul Garg (Regd. No. : 101506133)
Ridham Arora (Regd. No. : 101506137)

Group: ECE-7

Supervisor
Mr. Sukhwinder Kumar
Lecturer

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

1. 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:

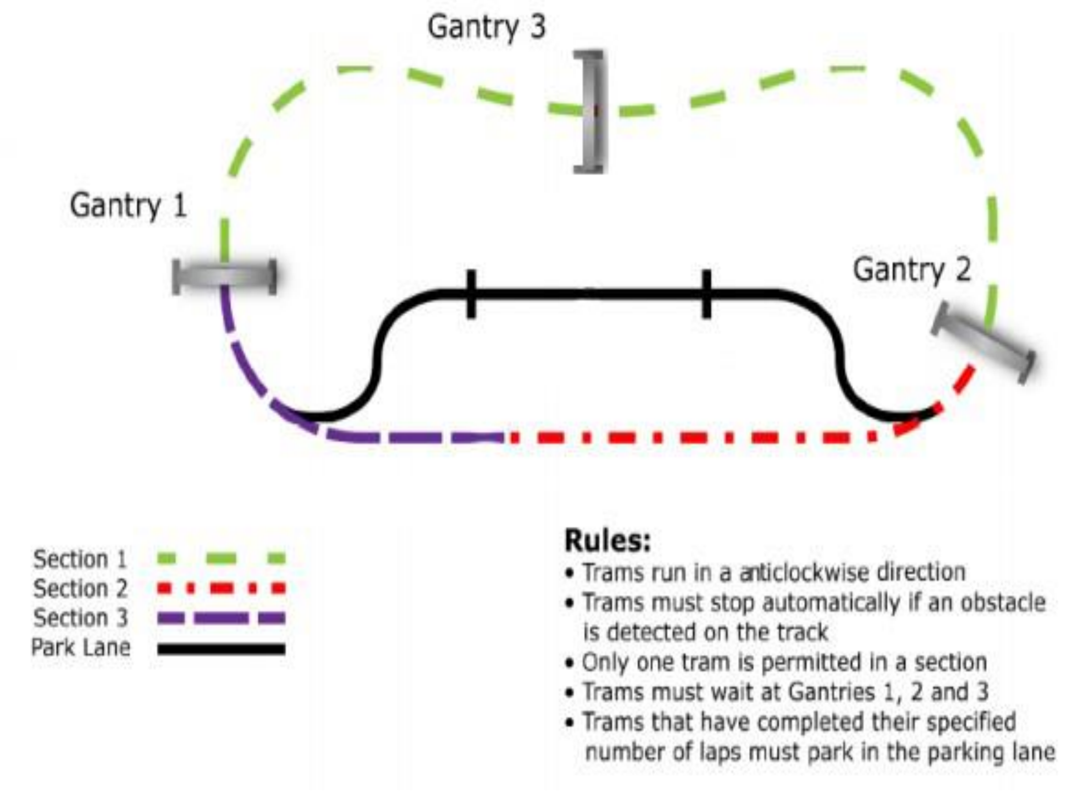
Challenge Silver: The challenge included moving a single buggy on the defined track with predefined events and conditions. The student's teams were required to design buggy along with sensors as per the requirement of challenge and design an algorithm/ program for execution of the defined challenge through 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 PRATHIT MALIK

2.1.1 Gantt Chart:

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					■

2.1.2 Contribution :

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.

i. The interfacing of ultrasonic sensors with arduino.
ii. Writing the code of ultrasonic in program code.
iii. Testing it on buggy.
iv. Configuring zigbee & infrared sensors with arduino.
v. Writing the code of zigbee and infrared sensors in program code.
vi. Making Report

2.2 PRIYUM BANSAL

2.2.1 Gantt Chart:

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 Contribution:

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.

i. The interfacing of ultrasonic sensors with arduino.
ii. Writing the code of ultrasonic in program code.
iii. Testing it on buggy.
iv. Configuring zigbee & infrared sensors with arduino.
v. Writing the code of zigbee and infrared sensors in program code.
vi. Making Report

2.3 RAHUL GARG

2.3.1 Gantt Chart :

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					█

2.4.2 Contribution:

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.

i. The interfacing of ultrasonic sensors with arduino.
ii. Writing the code of ultrasonic in program code.
iii. Testing it on buggy.
iv. Configuring zigbee & infrared sensors with arduino.
v. Writing the code of zigbee and infrared sensors in program code.
vi. Making Report

2.4 RIDHAM ARORA

2.4.1 Gantt Chart:

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 Contribution:

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.

i. The interfacing of ultrasonic sensors with arduino.
ii. Writing the code of ultrasonic in program code.
iii. Testing it on buggy.
iv. Configuring zigbee & infrared sensors with arduino.
v. Writing the code of zigbee and infrared sensors in program code.
vi. Making Report

3.COMPONENT DETAILS

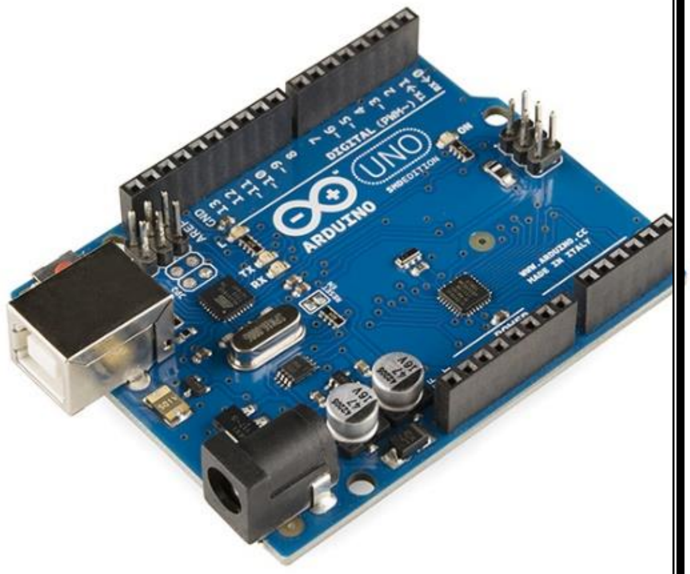
• ARDUINO UNO

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.

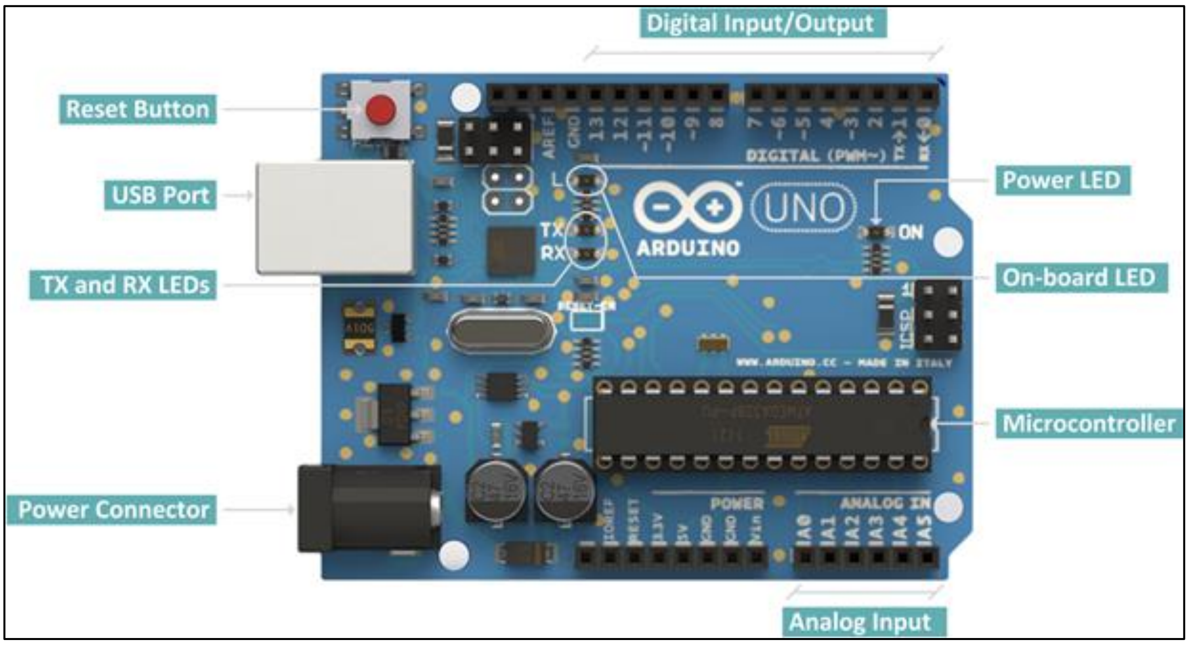
• **3.3V**. A 3.3 V supply generated by the on-board regulator. Maximum current draw is 50 mA.

• **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



- **Nvis 3302ARD RoboCar**

Introduction
Nvis 3302ARD is capable of sensing environment using various sensor modules and acts accordingly. Nvis RoboCar is a ready assembled unit consisting of FR4 chassis wheels with different Sensor modules mounted on it. The machine is driven by DC motors which are powered by rechargeable batteries. This Nvis 3302ARD is Atmega328P Microcontroller (Arduino based) RoboCar, is designed for users to start developing smart robot which is capable of accelerometer balancing, Gyroscope angular velocity sensing, Ultrasonic obstacle avoiding/detecting and distance measure and many more. There is Zigbee for wireless control your smart RoboCar.

Features
· Wireless Zigbee RoboCar control
· Ultrasonic RoboCar control
· DC motor interface & control
· Gyroscope Accelerometer Sensor Interface and control
· Expansion Analog connectors for enhancing more experiments

· Expansion PWM connectors for Servo motor Interface
· Expansion connectors for use ISP Programming
· Onboard motor Driver IC to control Motor
· Onboard battery charger
· PC based Programming
· Separate reset switch facility for Zigbee, controller
· Every pin is marked in order to make work easier

- **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.

Ultrasonic sensors are based on measuring the properties of sound waves with frequency above the human audible range. They are based on three physical principles: time of flight, the Doppler effect, and the attenuation of sound waves. Ultrasonic sensors are non-intrusive in that they do not require physical contact with their target, and can detect certain clear or shiny targets otherwise obscured to some vision-based sensors. On the other hand, their measurements are very sensitive to temperature and to the angle of the target.
Ultrasonic sensors “are based on the measurement of the properties of acoustic waves with frequencies above the human audible range,” often at roughly 40 kHz 1). They typically operate

by generating a high-frequency pulse of sound, and then receiving and evaluating the properties of the echo pulse.
Three different properties of the received echo pulse may be evaluated, for different sensing purposes. They are:

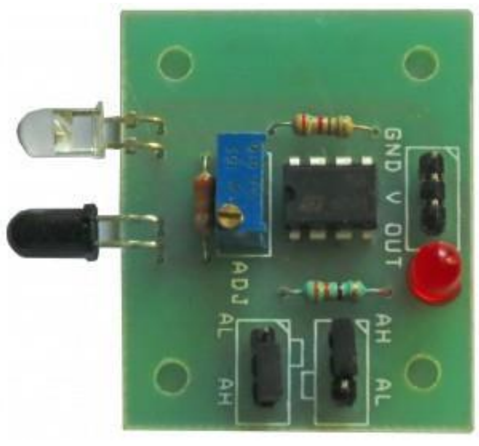
- Time of flight (for sensing distance)
- Doppler shift (for sensing velocity)
- Amplitude attenuation (for sensing distance, directionality, or attenuation coefficient)

Working Principle

We only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.

• INFRARED SENSOR:

An infrared sensor is an electronic device, that emits in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion.These types of sensors measures only infrared radiation, rather than emitting it that is called as a passive IR sensor. Usually in the infrared spectrum, all the objects radiate some form of thermal radiations. These types of radiations are invisible to our eyes, that can be detected by an infrared sensor.The emitter is simply an IR LED (Light Emitting Diode) and the detector is simply an IR photodiode which is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, The resistances and these output voltages, change in proportion to the magnitude of the IR light received.



Working principle
An infrared sensor circuit is one of the basic and popular sensor module in an electronic device. This sensor is analogous to human’s visionary senses, which can be used to detect obstacles

and it is one of the common applications in real time.This circuit comprises of the following components

- • LM358 IC 2 IR transmitter and receiver pair
- • Resistors of the range of kilo ohms.
- • Variable resistors.
- • LED (Light Emitting Diode).

In this project, the transmitter section includes an IR sensor, which transmits continuous IR rays to be received by an IR receiver module. An IR output terminal of the receiver varies depending upon its receiving of IR rays. Since this variation cannot be analyzed as such, therefore this output can be fed to a comparator circuit. Here an operational amplifier (op-amp) of LM 339 is used as comparator circuit.
When the IR receiver does not receive a signal, the potential at the inverting input goes higher than that non-inverting input of the comparator IC (LM339). Thus the output of the comparator goes low, but the LED does not glow. When the IR receiver module receives signal to the potential at the inverting input goes low. Thus the output of the comparator (LM 339) goes high and the LED starts glowing. Resistor R1 (100), R2 (10k) and R3 (330) are used to ensure that minimum 10 mA current passes through the IR LED Devices like Photodiode and normal LEDs respectively. Resistor VR2 (preset=5k) is used to adjust the output terminals. Resistor VR1 (preset=10k) is used to set the sensitivity of the circuit Diagram. Read more about IR sensors.

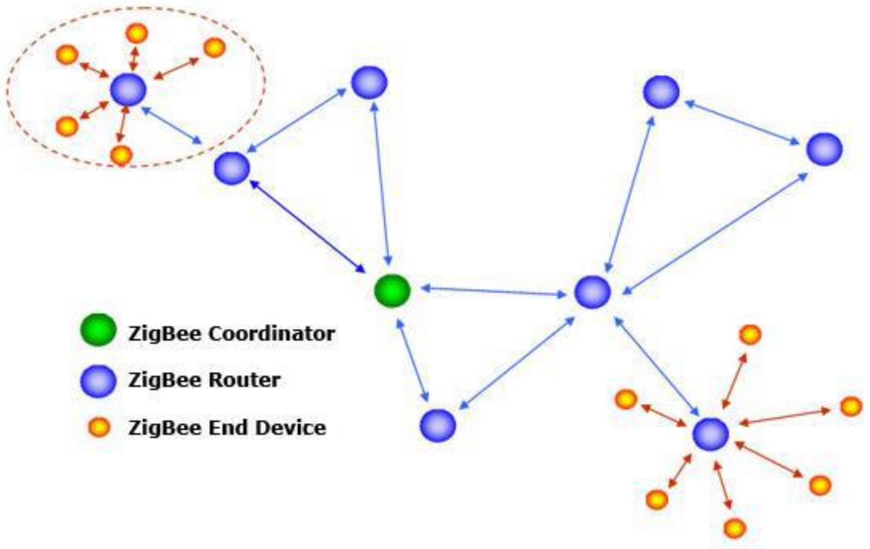
• ZigBee

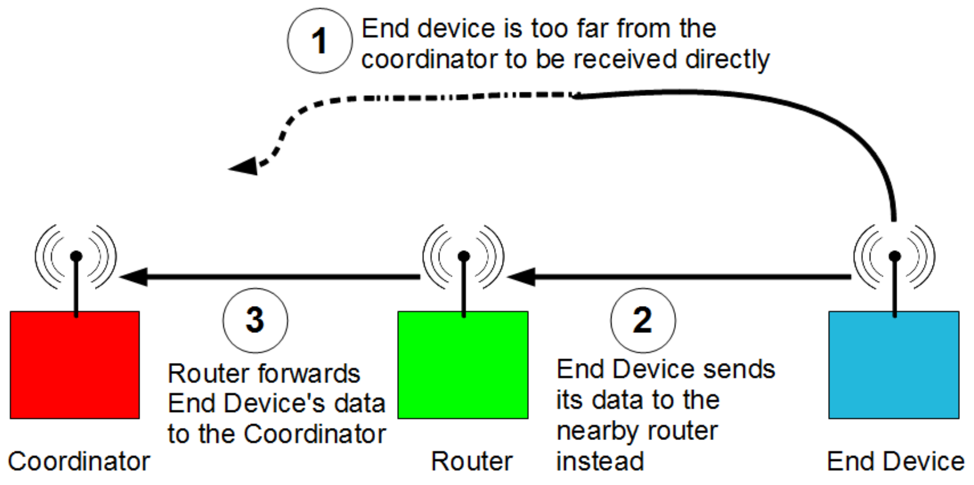
Zigbee is a wireless communication module which use IEEE 802.15.4 standard. 802.15.4 is a IEEE standard for low power applications of radio frequency. It used in many products now a days for wireless communication functionality. It can be used as a transmitter and receiver both. It used serial communication to send and receive data. It have two series, series1 and series 2. Series 1 is comparatively easy to use and it is recommended for beginners. Series 1 zigbee module can not work in mesh network. Mean it can not talk to more than one zigbees buddies.
As I have already mentioned it use serial port to send and receive data. So its mean it can be easily interface with Arduino Uno R3, any type of microcontroller and computer. Because they all support serial communication and they all have serial port to send and receive data. It can also communicate with other Zigbee to form a mesh. Zigbee can also be used to make a local area network. It have many applications. But some of the famous applications of Zigbee is given below.

Applications of Zigbee :
- Wireless communication
- wirelessly controlled robot
- Remote monitoring system
- Wireless home automation system
- wireless temperature sensor and many others.

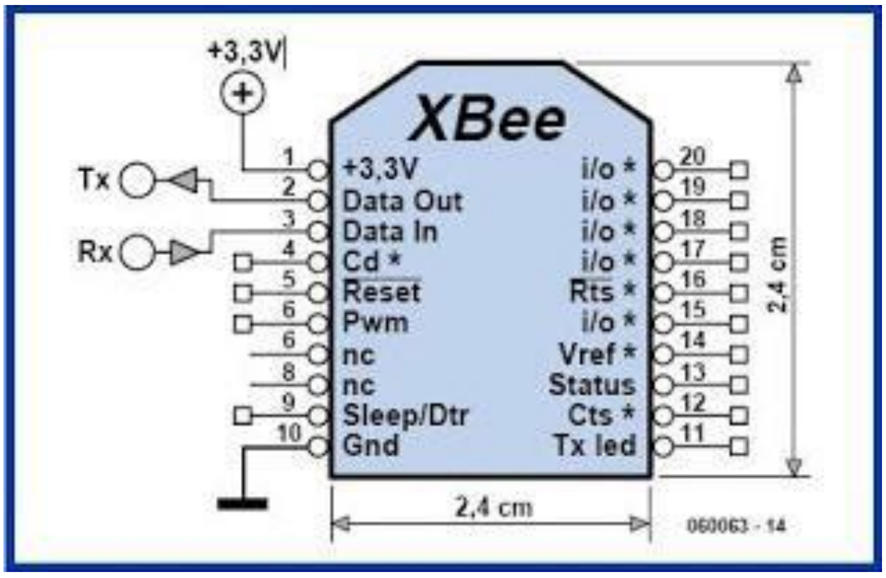
Zigbee alone can’t do any thing. You have to interface it with some intelligent device like microcontrollers, Arduino and computer. These devices will tell it what to do or what no to do through alrady fed program inside microcontrollers and Arduino Uno R3. These digital devices are no such intelligent. But you can make them intelligent by writing few lines of instructions. Let’s move forward and learn how to interface Zigbee with Arduino.

Zigbee Network Formations:
Zigbee defines three difference device types: coordinator, router and end device.
Coordinator: Start a new personal area network (PAN) by selecting the channel and PAN ID. Allow routers and end devices to join the PAN, transmit and receive RF data transmission and route the data through the mesh network. In charge of setting up the network. Can never sleep.
Router: Transmit and receive RF data transmission, and route data packet through the network. Can relay signals from other routers/EPs. Can never sleep.
End Device: Cannot assist in routing the data transmission but transmit or receive RF data transmission and intended to be battery powered devices. Cannot relay signals. Can sleep to save power
Transceiver: Transceiver is Transmitter and Receiver as one module, XBee can transmit data and receive data. Arduino(A) transmit data to XBee(A) via wire and XBee(A) transmit that data wirelessly to XBee(B); upon receiving data from XBee(A) wirelessly, XBee(B) will then transmit that data to Arduino(B) via wire. Same applied when data is transmitted from Arduino (B).



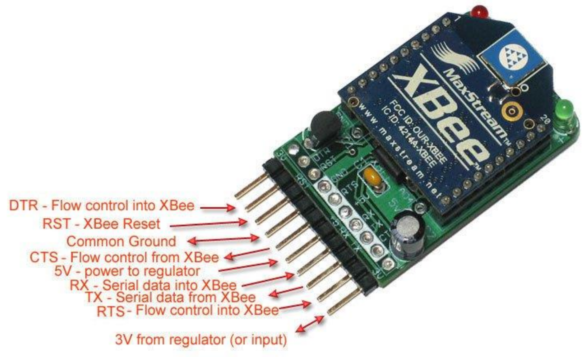

1
End device is too far from the coordinator to be received directly
3
Router forwards End Device's data to the Coordinator
2
End Device sends its data to the nearby router instead
Coordinator
Router
End Device

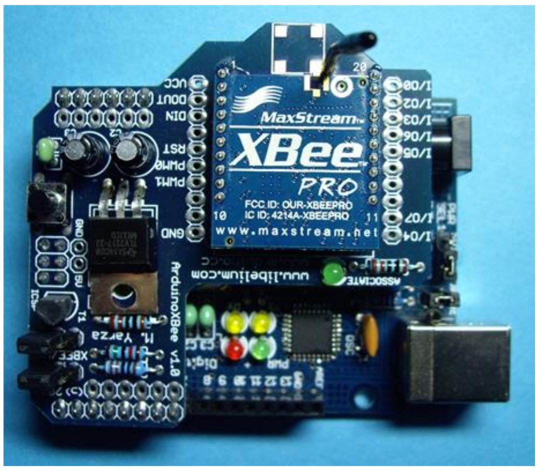
INTERFACING WITH ARDUINO:
Two Zigbee modules can talk with each other if both are of same type. To communicate to it module with each other, obtain to its modules. Connect one module to Arduino and other module to either sensor or any microcontroller or computer.
API and AT Modes:
The XBee modules can 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.



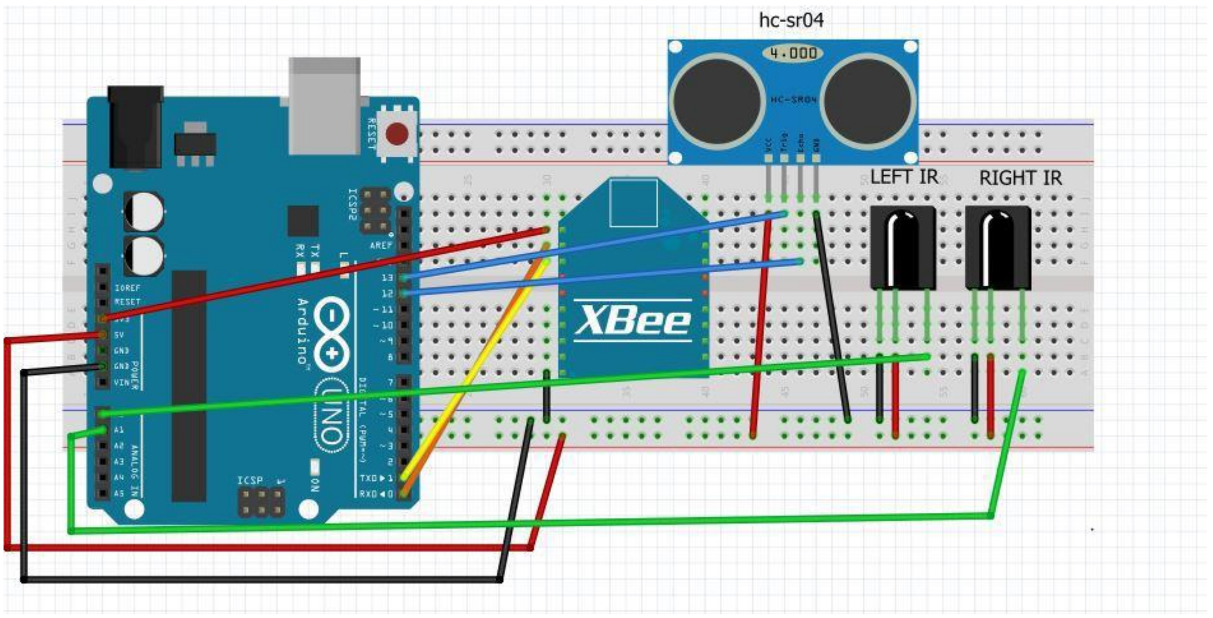
API and AT Modes:

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)



4.CONNECTION DIAGRAM WITH ARDUINO



5.PROGRAM CODE

Silver Challenge:

Buggy -1 : Anticlockwise Track Following Code

```
int leftIR1, rightIR1, leftIR2, rightIR2, obs_IR; // rightIR:0, leftIR:1, obs_IR:3 int rightf = 5,
rightb = 6;

int leftb = 7, leftf = 8;

int count = 0, flag = 1, pole = -1;

void setup()

{

for (int i = 5; i < 9; i++)

{

pinMode(i, OUTPUT);

}

Serial.begin(9600);

stop();

}

void loop()

{

take_input(&rightIR1, &leftIR1); delay(5);

take_input(&rightIR2, &leftIR2);

if ((leftIR1 == leftIR2) && (rightIR1 == rightIR2))

    {

    if (rightIR1 == 1 && leftIR1 == 1)

    {

    forward();

    if (pole == 0)

    {
```

```
pole = 1;
  }
}
else if (rightIR1 == 1 && leftIR1 == 0) left();
else if (rightIR1 == 0 && leftIR1 == 1) right();
else if (rightIR1 == 0 && leftIR1 == 0)
{
  if (pole == -9)
  {
  stop(); for (;;);
  }
  if (count == 9)
  {
  if (pole == 1) //Handling Parking Turn
  {
  right();
  delay(500); count = 0; pole = -9;
  }
    else
    {
    left();
    forward();
    delay(50); pole = 0;
    }
    }
else if (count > 0) //Handling No Parking Turn

  {
```

```
left();
forward();

delay(50);

}
else forward();
}
}
Else
    {
    stop();
    }

    obstacle();

  }
void obstacle()

{

  obs_IR = analogRead(3); if (obs_IR > 10)
  {

    if (flag == 1)

    {

      count++;

      stop();

      delay(1000); flag = 0;
    }
  }
  else
  {
    flag = 1;
  }
}
void stop()
{
  digitalWrite(rightf, LOW);
   digitalWrite(rightb, LOW);
  digitalWrite(leftf, LOW);
   digitalWrite(leftb, LOW);
}

void forward()
{
  digitalWrite(rightf, HIGH);
  digitalWrite(rightb, LOW);
   digitalWrite(leftf, HIGH);
   digitalWrite(leftb, LOW); // delay(20);

}
```

```
void left()

{

  digitalWrite(rightf, HIGH);

  digitalWrite(rightb, LOW);

  digitalWrite(leftf, LOW);

   digitalWrite(leftb, HIGH); // delay(20);

}
void right()

{

  digitalWrite(rightf, LOW);

  digitalWrite(rightb, HIGH);

  digitalWrite(leftf, HIGH);

   digitalWrite(leftb, LOW); // delay(20);

}
void take_input(int *rightIR, int *leftIR)

{

  *rightIR = analogRead(0);

  *leftIR = analogRead(1); //serial_out(*rightIR,*leftIR);

  mapping(rightIR, leftIR);

}
void serial_out(int rightIR, int leftIR)

{
  Serial.print("rightIR : ");

  Serial.println(rightIR);

  Serial.print("leftIR : ");

  erial.println(leftIR);

  delay(500);

}
void mapping(int *rightIR, int *leftIR)

{

  if (*rightIR < 10) *rightIR = 0;

  else
  *rightIR = 1; if (*leftIR < 10) *leftIR = 0;

  else

    *leftIR = 1;

}
```

Gold Challenge:

(Buggy -1 : Anticlockwise Track Following Code)

```
int leftIR1, rightIR1, leftIR2, rightIR2, obs_IR; // rightIR:0, leftIR:1, obs_IR:3 int rightf = 5,
rightb = 6;

int leftb = 7, leftf = 8;

int count = 0, flag = 1, pole = -1;
void setup()
{
  for (int i = 5; i < 9; i++)
{
pinMode(i, OUTPUT);

  }

  Serial.begin(9600);

  stop();

}
void loop()

{
if (Serial.available() > 0)
{
char s = Serial.read();
  }

  if (s == 'X')

  {
```

```
take_input(&rightIR1, &leftIR1); delay(5);

take_input(&rightIR2, &leftIR2);

if ((leftIR1 == leftIR2) && (rightIR1 == rightIR2))

{

  if (rightIR1 == 1 && leftIR1 == 1)

  {

    forward();

    if (pole == 0)

      { ole = 1;

    }

  }

  else if (rightIR1 == 1 && leftIR1 == 0) left();

  else if (rightIR1 == 0 && leftIR1 == 1) right();

  else if (rightIR1 == 0 && leftIR1 == 0)

  {

    if (pole == -9)

    {
    stop(); for (;;);
```

```
 }
 if (count == 9)

 {
 if (pole == 1) //Handling Parking Turn
 {
 right();
 delay(500); count = 0; pole = -9;
   }
   else
   {
   left();
   forward();
   delay(50); pole = 0;
   }

 }

 else if (count > 0) //Handling No Parking Turn

 {

   left();

   forward();

   delay(50);

 }
 else forward();
}
```

```
    }

    else

    {
stop();
    }
obstacle();
}
}
void obstacle()
{
obs_IR = analogRead(3); if (obs_IR > 10)
  {
  if (flag == 1)
  {
  count++;
  stop();
delay(1000); flag = 0;
    }

  }

  else

  {

    flag = 1;

  }

}
```

```
void stop()

{


  digitalWrite(rightf, LOW);

   digitalWrite(rightb, LOW);

  digitalWrite(leftf, LOW);

  digitalWrite(leftb, LOW);

void forward()

{

  digitalWrite(rightf, HIGH); digitalWrite(rightb, LOW); digitalWrite(leftf, HIGH);
  digitalWrite(leftb, LOW); // delay(20);

}

void left()

{

  digitalWrite(rightf, HIGH); digitalWrite(rightb, LOW); digitalWrite(leftf, LOW);
  digitalWrite(leftb, HIGH); // delay(20);

}

void right()

{

  digitalWrite(rightf, LOW); digitalWrite(rightb, HIGH); digitalWrite(leftf,
  HIGH); digitalWrite(leftb, LOW); // delay(20);

  }

void take_input(int *rightIR, int *leftIR)

{

  *rightIR = analogRead(0);

  *leftIR = analogRead(1); //serial_out(*rightIR,*leftIR);

  mapping(rightIR, leftIR);

}
```

```
void serial_out(int rightIR, int leftIR)

{
  Serial.print("rightIR : ");
  Serial.println(rightIR);
  Serial.print("leftIR : ");
  Serial.println(leftIR); delay(500);

}
void mapping(int *rightIR, int *leftIR)
{
  if (*rightIR < 10) *rightIR = 0;

  else
   *rightIR = 1; if (*leftIR < 10)
                 *leftIR = 0;

                        else

    *leftIR = 1;


}
```

Buggy -2 : Clockwise Track Following Code

```
int leftIR1, rightIR1, leftIR2, rightIR2, obs_IR; // rightIR:0, leftIR:1, obs_IR:3 int rightf = 5,
rightb = 6;

int leftb = 7, leftf = 8;
int count = 0, flag = 1, pole = -1;
void setup()
{
  for (int i = 5; i < 9; i++)
  {
```

```
  pinMode(i, OUTPUT);
  }
  serial.begin(9600);
  stop();
}
void loop()
{
  take_input(&rightIR1, &leftIR1); delay(5);
  take_input(&rightIR2, &leftIR2);
  if ((leftIR1 == leftIR2) && (rightIR1 == rightIR2))
{
if (rightIR1 == 1 && leftIR1 == 1)
{
forward();
if (pole == 0)
{
pole = 1;
}
}
else if (rightIR1 == 1 && leftIR1 == 0) left();
else if (rightIR1 == 0 && leftIR1 == 1) right();
else if (rightIR1 == 0 && leftIR1 == 0)
{
if (pole == -9)
{
stop();
Serial.print('X'); for (;;);

      }
```

```
    if (count == 9)
    {
       if (pole == 1) //Handling Parking Turn
       left();
       delay(500);
       count = 0;
       pole = -9;
     }
     else
     {
     right();
     forward();
     delay(50); pole = 0;
     }
     }
    else if (count > 0) //Handling No Parking Turn
{
right();
forward();
delay(50);
}
else forward();
}
}
else
{
stop();
```

```
  }
obstacle();
}
void obstacle()
{
obs_IR = analogRead(3);
  (obs_IR > 10)

  {

    if (flag == 1)

    {

      count++;

      stop();

      delay(1000); flag = 0;

    }

  }

  else

  {

    flag = 1;

  }

}
```

```
void stop()
{
  digitalWrite(rightf, LOW);
   digitalWrite(rightb, LOW);
  digitalWrite(leftf, LOW);
  digitalWrite(leftb, LOW);
}
void forward()
{
  digitalWrite(rightf, HIGH);
   digitalWrite(rightb, LOW);
   digitalWrite(leftf, HIGH);
   digitalWrite(leftb, LOW); // delay(20);
}
void left()
  digitalWrite(rightf, HIGH);
  digitalWrite(rightb, LOW);
   digitalWrite(leftf, LOW);
  digitalWrite(leftb, HIGH); // delay(20);
}
void right()

{
  digitalWrite(rightf, LOW);
  digitalWrite(rightb, HIGH);
  digitalWrite(leftf, HIGH);
  digitalWrite(leftb, LOW); // delay(20);

}
void take_input(int *rightIR, int *leftIR)

{

  *rightIR = analogRead(0);

  *leftIR = analogRead(1); //serial_out(*rightIR,*leftIR);

   mapping(rightIR, leftIR);

}
void serial_out(int rightIR, int leftIR)

{

  Serial.print("rightIR : ");

  Serial.println(rightIR);

  Serial.print("leftIR : ");

  Serial.println(leftIR); delay(500);

}

void mapping(int *rightIR, int *leftIR)

{

  if (*rightIR < 10) *rightIR = 0;

  else

                        *rightIR = 1; if (*leftIR < 10) *leftIR = 0;
  else
    *leftIR = 1;
}
```

6.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.

7.RESULT

We were able to successfully interface Buggy with various components. We learned how to interface our buggy with IR sensor and based on the data sent by IR we were able to make a simple line follower. We mounted an Ultrasonic sensor on the top of our buggy and we able to detect the obstacles on our track and take actions as required.

To communicate between two buggies we used Xbee working on ZigBee protocol using XCTU software and as a result we were successful in establishing a wireless connection between two buggies.

We made one Xbee as coordinator and mounted it on top of the buggy and made the other as Router and mounted it on top of another buggy. We used it to perform the action that when one buggy just parked itself on the marked location another buggy became aware of this and started moving as programmed. Hence as a result we also learned how wireless communication can take place between two buggies and program events that follow.