NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPPALLI

SUMMER PROJECT

Leader Follower Robot

By:
Gudapati Nitish
Mechanical Engineering
NIT Trichy

Contents

1 Project Introduction

1.1 Basic Task

This task involves replication of the path of the Leader robot by the follower robot.

1.2 Advanced Task

The advanced task is localisation using odometer readings derived from the encoder in Leader Robot and the Follower robot has to move to that location.

2 Requirements

2.1 Hardware

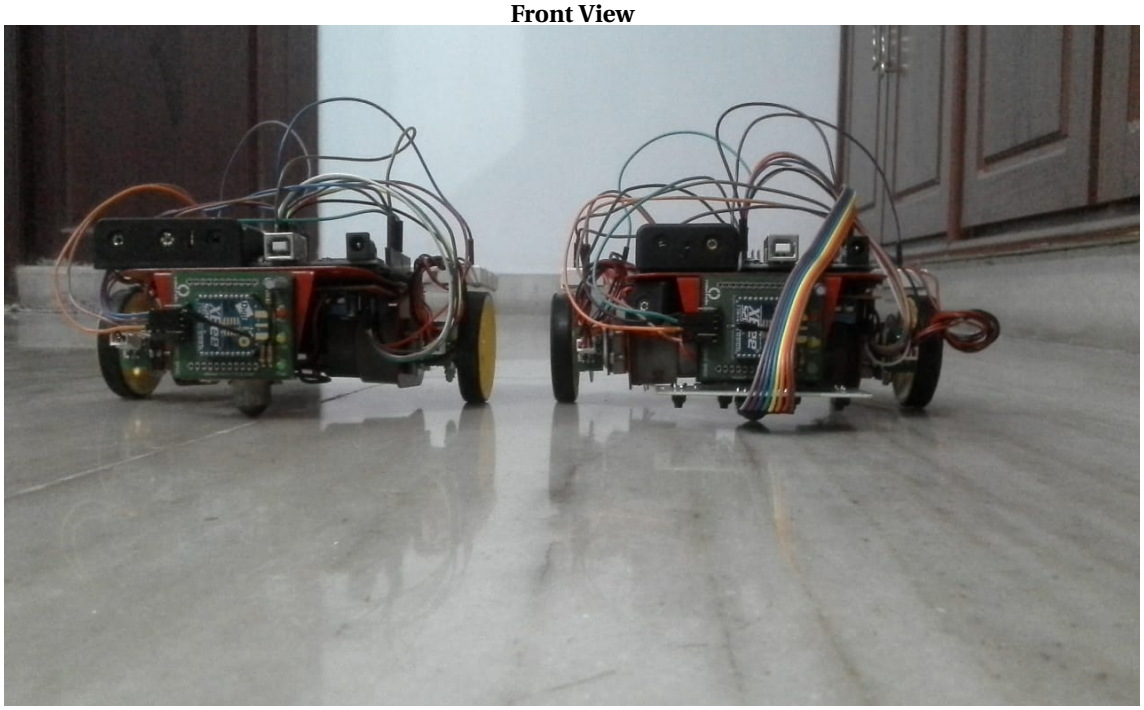
The hardware requirements are three Arduinos and three XBee modules (two for the bots and one for interfacing with the laptop), one IR Array, four DC Motors with each one fitted with wheel encoders (IR Tachometer), two dc motor drivers, two metal chassis

2.2 Software

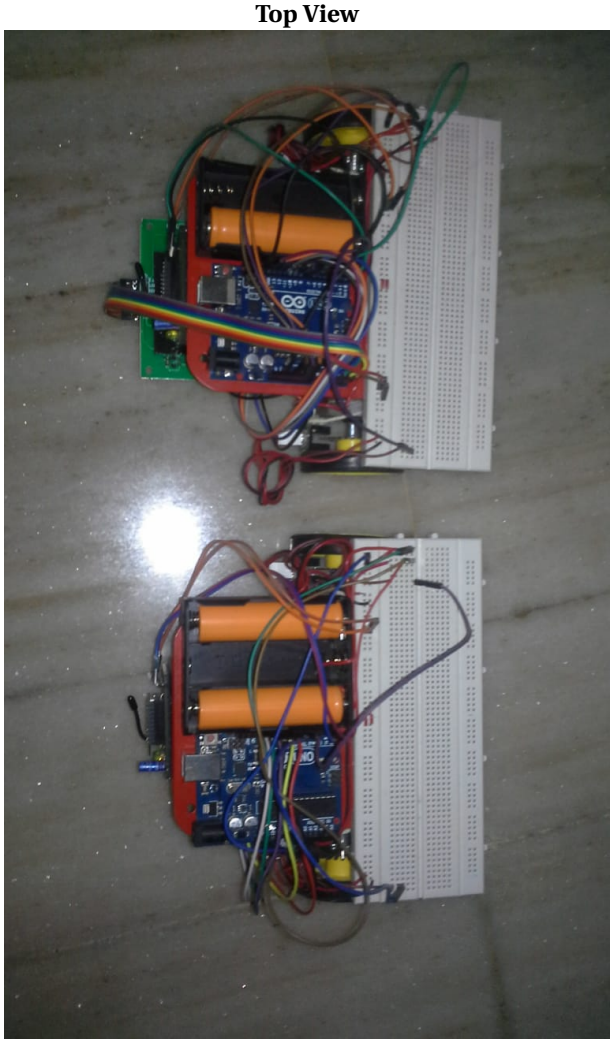
The softwares required are Arduino IDE, XCTU, Python 2, PySerial and VPython Library.

3 Robot Design

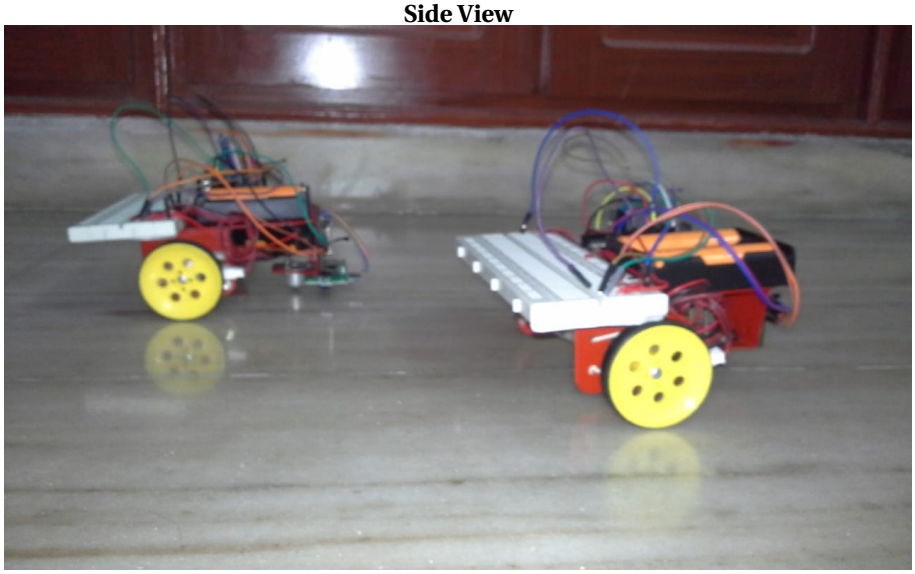
Front View



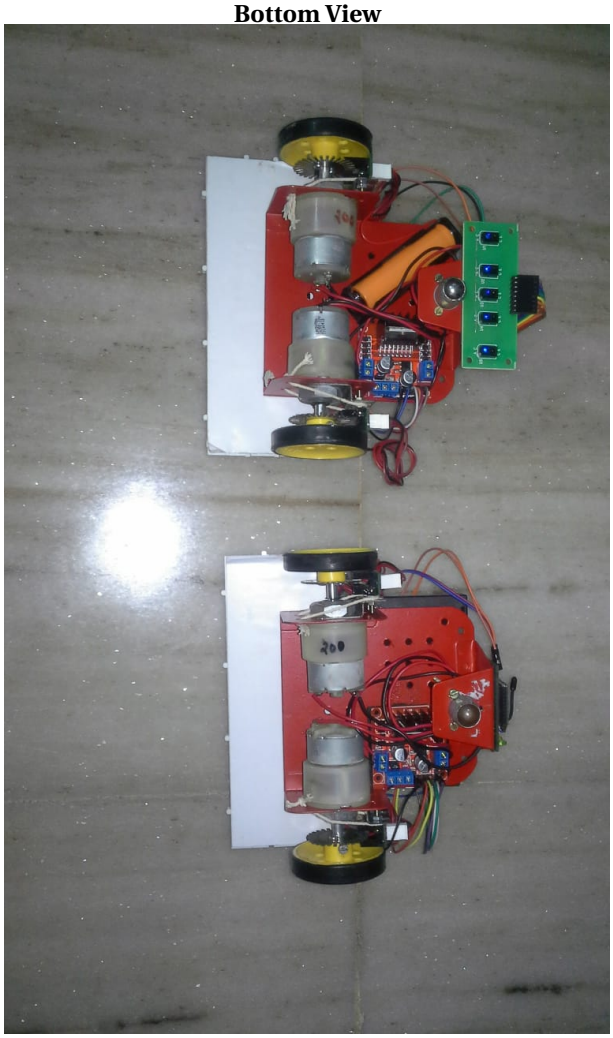
Top View



Side View



Bottom View



Back View

4 Description

4.1 XBee

In-order to send the processed data between the robots and computer, Xbee devices were used. Xbees are the devices which are used for setting up wireless communication. These devices work on Zigbee wireless protocol. Another advantage of Zigbee protocol is that the Xbees can be designed to form a mesh network. Using mesh network, communication between Xbees which are located farther than their range can communicate easily. They can also be set up as point to multi-point network topology.

XBee works in two modes - transparent mode (AT) and application program interface mode (API). API mode provides a structured interface where data is communicated through the serial interface in organised packets and in a determined order. By default, XBee devices are configured to work in transparent mode: all data received through the serial input is queued up for radio transmission and data received wirelessly is sent to the serial output exactly as it is received, with no additional information. To read or write the configuration of an device in Transparent mode, the device is first transitioned into Command mode.

API mode provides a structured interface where data is communicated through the serial interface in organised packets and in a determined order. Since the data destination is included as part of the API frame structure, API mode can be used to transmit messages to multiple devices. The API frame includes the source of the message so it is easy to identify where data is coming from. It can receive success/failure status of each transmitted packet and obtain the signal strength of any received packet. In this project, API mode is used along with ZB TH reg firmware.

API frame structure

The structured data packets in API mode are called frames. They are sent and received through the serial interface of the device and contain the wireless message itself as well as some extra information such as the destination/source of the data or the signal quality. When a device is in API mode, all data entering and leaving the module through the serial interface is contained in frames that define operations or events within the device. An API frame has the following structure:

Start delimiter	Length		Frame data: Frame type	Frame data: Data							Checksum
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB	API frame type	Frame-type-specific data							Single byte

Any data received through the serial interface prior to the start delimiter is silently discarded by the XBee. If the frame is not received correctly, or if the checksum fails, the data is also discarded and the module indicates the nature of the failure by replying with another frame.

Start delimiter:
The start delimiter is the first byte of a frame consisting of a special sequence of bits that indicate the beginning of a data frame. Its value is always 0x7E. This allows for easy detection of a new incoming frame.

Length

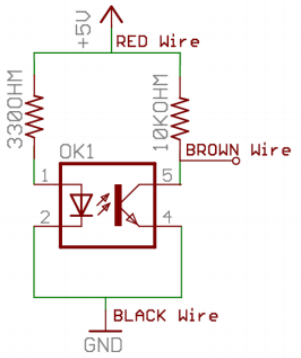
The length field specifies the total number of bytes included in the frame data field. Its two-byte value excludes the start delimiter, the length, and the checksum.

Frame data:
This field contains the information received or to be transmitted. Frame data is structured based on the purpose of the API frame.
Frame type is the API frame type identifier. It determines the type of API frame and indicates how the information is organized in the Data field.
Data contains the data itself. The information included here and its order depends on the type of frame defined in the Frame type field.

Checksum:
Checksum is the last byte of the frame and helps test data integrity. It is calculated by taking the hash sum of all the API frame bytes that came before it, excluding the first three bytes (start delimiter and length).

Note: Frames sent through the serial interface with incorrect checksums will never be processed by the module and the data will be ignored.
Calculate the checksum of an API frame
Add all bytes of the packet, excluding the start delimiter 0x7E and the length (the second and third bytes). From the result, keep only the lowest 8 bits. Subtract this quantity from 0xFF.

4.2 Wheel Encoder (IR Tachometer)



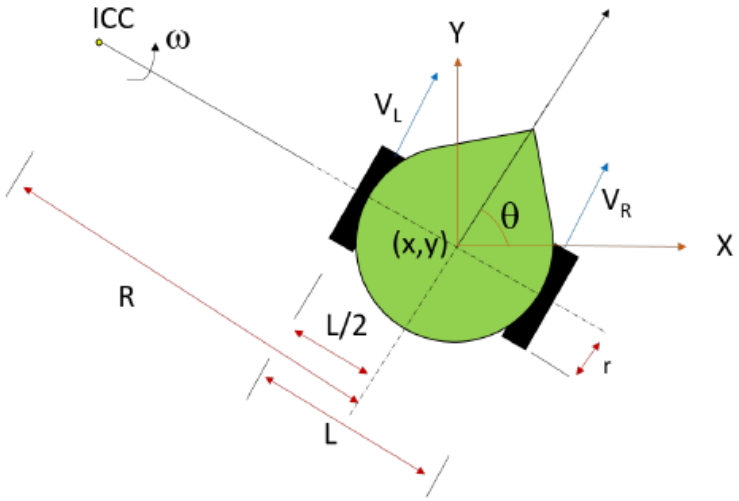
4.3 PID Controller

Proportional, Integral, Derivative PID controller is a feedback controller that helps to attain a set point irrespective of disturbances or any variation in characteristics of the plant of any form. It calculates its output based on the measured error and the three controller gains - proportional gain Kp, integral gain Ki, and derivative gain Kd. The proportional gain simply multiplies the error by a factor Kp. This reacts based on how big the error is. The integral term is a multiplication of the integral gain and the sum of the recent errors. The integral term helps in getting rid of the steady state error and causes the system to catch up with the desired set point. The derivative controller determines the reaction to the rate of which the error has been changing. In most of the systems, it is not necessary to use the derivative part of the PID, hence in this project, only PI controller has been designed and used. The final output of the controller (U) is calculated using the following equation:

$$U = Kp * e(t) + Ki \int e\,dt + Kd \frac{de}{dt}$$

Parameter Increase	Rise time	Overshoot	Settling Time	Steady-state error
Kp	↓	↑	Small Change	↓
Ki	↓	↑	↑	Great reduce
Kd	Small Change	↓	↓	Small Change

4.4 Odometry



$$c = \frac{2 \times \pi}{\frac{Tickspperrevolution \times dt}{1000000.0}}$$

$$\omega_l = (Ticks_l - TicksPrev_l) \times c$$

$$\omega_r = (Ticks_r - TicksPrev_r) \times c$$

$$V_l = \omega_l \times R$$

$$V_r = \omega_r \times R$$

$$V = \frac{V_r + V_l}{2.0}$$

$$\omega = \frac{V_r - V_l}{L}$$

$$\theta_{next} = \theta + (dt \times w)$$

$$x_{next} = x_c + \frac{dt \times v \times cos(\theta_{next})}{10.0}$$

$$y_{next} = y_c + \frac{dt \times v \times sin(\theta_{next})}{10.0}$$

$$distance = \sqrt{x_c^2 + y_c^2}$$

5 Basic Task

This task involves replication of the path of the Leader robot by the follower robot. There are two ways of solving the problem. Firstly, The leader robot tracks a line and and sends the rpm values of each of its wheel. The corresponding wheels of the follower are set to run at the same rpm as that of the leader using PID control algorithm. Thereby, the follower replicates the leader's path. The second way is by sending the velocity of the bot and its orientation. In this project the first method is implemented. First, the Leader robot tracks a line using IR Array Sensor. Simultaneously, it calculates the wheel rpm and it's x,y coordinates and orientation. All the above data along with the wheel direction is sent as a packet through XBee.

The follower uses proportional controller to set the rpm and direction of each of the wheels. It then sends the x,y,theta of both the leader and the follower bots to the Laptop Xbee. The paths are then plotted in Python Serial.

6 Advanced Task

In this task, a path is hard coded into the leader robot and it moves in that path as a differential drive robot. Simultaneously, it calculates the odometric values i.e, it's x,y coordinates and orientation. All the above data through XBee.

The follower gets the location of the leader and uses kinematics to find the direction and distance of the leader and then moves to the location. It then sends the x,y,theta of both the leader and the follower bots to the Laptop Xbee. The paths are then plotted in Python Serial.

7 Source Code

GitHub : Leader-Follower Robotic System

8 Working Videos

Google Drive : Leader Follower Robotic System

9 Problems Faced

The main problem I faced is communicating between the XBee devices. There have been a lot of data packet losses at high frequency transmission. I consumed a lot of time in identifying a suitable firmware and configuration. The firmware I have used initially is 802.15.4 which is the firmware of series1 xbee. Zigbee Th reg is the firmware of series2. That's why series 2 code didn't work for 802 firmware. I tried series1 code with 802 firmware, but it failed. I changed configurations in xctu from 802.15.4 to zigbee th reg firmware with series2 code. With this firmware in broadcast, the data is transmitted but the frequency was a problem. Broadcast takes a lot of time as it makes roughly eight transmissions per packet and hence for three XBees make it twenty-four transmissions. So, instead of broadcasting, I used unicast using the MAC addresses. The accuracy of broadcast is one msg in 1.5 sec, whereas in unicast, it is once in 20 ms when tested in xctu. But it doesn't work with same accuracy in arduino sketch in api mode 1, so I changed it to api2. Also the data packet size also determines the time interval between two transmissions. I sent the data from leader to follower in unicast at 20ms delay. From follower I sent the same data as and when received to the laptop in unicast again. Now the problem is solved.

Another problem I faced is that the ir tachometer interrupt bounces too often. To solve this first, I used a comparator circuit, but that too failed. Finally, Schmitt Trigger IC solved the problem.