ROS BASED ROBOTIC CAR

Debanik Roy

4th to 16th March 2020

—

Mr. Anirban Ghatak (Mentor)

Content

Page No.

Introduction

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. These are explained in greater detail in our Conceptual Overview.

ROS is not a realtime framework, though it is possible to integrate ROS with realtime code. The Willow Garage PR2 robot uses a system called pr2_etherCAT, which transports ROS messages in and out of a realtime process. ROS also has seamless integration with the Orocos Real-time Toolkit.

Abstract

To Investigate the ROS framework and determine the requirement to run it into a raspberry pi 3.

To run a robot on a simulation environment called Rviz.

To Install Linux Ubuntu 18.04 into a laptop.

To configure a Wi-Fi network that is linked to the master node with the laptop or the raspberry pi.

To run a real experiment with Two wheel robot that is using a laptop as the main processing unit.

To Install ROS into the raspberry pi 3.

To run Ros software on both laptop node and the raspberry Pi, in order to do the obstacle avoidance robot using ROS.

1. Basic ROS for mobile Robot

1.1 Understanding the basics

- Nodes: A node is an executable that uses ROS to communicate with other nodes.
- Messages: ROS data type used when subscribing or publishing to a topic.

Topics: Nodes can publish messages to a topic as well as subscribe to a topic to receive
- messages.
- Master: Name service for ROS (i.e. helps nodes find each other)
- rosout: ROS equivalent of stdout/stderr, This is always running as it collects and logs nodes' debugging output.
- roscore: Master + rosout + parameter server (parameter server will be introduced later)
- rossrv: an srv file describes a service. It is composed of two parts: a request and a response.

1.2 Basic Ros cmd

- Filesystem Management Tools
  $ rospack: A tool for inspecting packages.
  $ rospack profile: Fixes path and pluginlib problems.
  $ roscd: Change directory to a package.
  $ rospd/rosd: Pushd equivalent for ROS.
  $ rosls: Lists package or stack information.
  $ rosed: Open requested ROS file in a text editor.
  $ roscp: Copy a file from one place to another.
  $ rosdep: Installs package system dependencies.
  $ roswtf: Displays a errors and warnings about a running ROS system or launch file. catkin create pkg Creates a new ROS stack.
  $ wstool: Manage many repos in workspace. catkin make Builds a ROS catkin workspace.
  $ rqt_dep: Displays package structure and dependencies

- Start-up and Process Launch Tools
  roscore : The basis nodes and programs for ROS-based systems. A roscore must be running for ROS nodes to communicate.
  Usage: $ roscore
  rosrun : Runs a ROS package's executable with minimal typing.
  Usage: $ rosrun package name executable name
  roslaunch : Starts a roscore (if needed), local nodes, remote nodes via SSH, and sets parameter server parameters.
  Usage: $ roslaunch package name file name.launch

- Introspection and Command Tools
- ➢ rosnode
  $ rosnode: ping Test connectivity to node.
  $ rosnode: list List active nodes.
  $ rosnode: info Print information about a node.
  $ rosnode: machine List nodes running on a machine.
  $ rosnode: kill Kill a running node.
- ➢ rostopic
  $ rostopic bw: Display bandwidth used by topic.

$ rostopic echo: Print messages to screen.
$ rostopic find: Find topics by type.
$ rostopic hz: Display publishing rate of topic.
$ rostopic info: Print information about an active topic.
$ rostopic list: List all published topics.
$ rostopic pub: Publish data to topic.
$ rostopic type: Print topic type.

- rosservice
  $ rosservice list: Print information about active services.
  $ rosservice node: Print name of node providing a service.
  $ rosservice call: Call the service with the given args.
  $ rosservice args: List the arguments of a service.
  $ rosservice type: Print the service type.
  $ rosservice uri: Print the service ROSRPC uri.
  $ rosservice find: Find services by service type.
- rosparam
  $ rosparam set: Set a parameter.
  $ rosparam get: Get a parameter.
  $ rosparam load: Load parameters from a file.
  $ rosparam dump: Dump parameters to a file.
  $ rosparam delete: Delete a parameter.
  $ rosparam list: List parameter names.
- rosmsg/ rossrv
  $ rosmsg show: Display the fields in the msg/srv.
  $ rosmsg list: Display names of all msg/srv. rosmsg md5 Display the msg/srv md5 sum.
  $ rosmsg package: List all the msg/srv in a package.
  $ rosmsg packages: List all packages containing the msg/srv.
- rosbag
  $ rosbag record: Record a bag file with specified topics.
  $ rosbag play: Play content of one or more bag files.
  $ rosbag compress: Compress one or more bag files.
  $ rosbag decompress: Decompress one or more bag files.
  $ rosbag filter: Filter the contents of the bag.
- transformation(tf)
  $ rosrun tf tf echo <source frame> <target frame>
- rqt_graph
  $ rqt bag bag file.bag
  $ rqt logger level
  $ rqt graph
  $ rqt dep
  $ rqt plot /topic1/field1 /topic2/field2 & $ rqt image view

- ROS distribution Catkin Workspaces
  Create a catkin workspace
  Setup and use a new catkin workspace from scratch.
  Example:
  $ source /opt/ros/kinetic/setup.bash
  $ mkdir -p ~/catkin ws/src
  $ cd ~/catkin ws/src
  $ catkin init workspace

2. Theoretical studies

2.1 Ultrasonic sensor & US_Node:

As the name indicates, ultrasonic / level sensors measure distance by using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. ultrasonic / level sensors measure the distance to the target by measuring the time between the emission and reception.

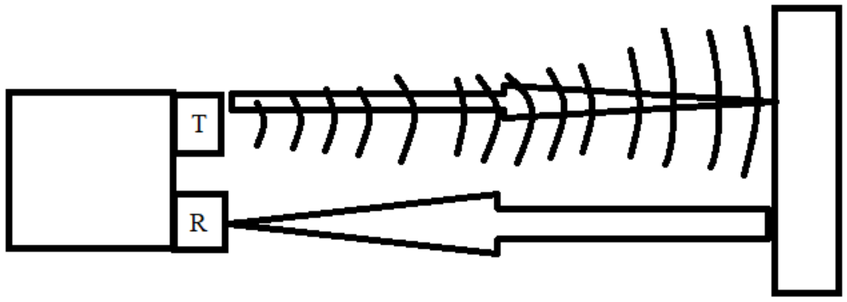


fig 1: ultra sonic obstracle avoidance

An optical sensor has a transmitter and receiver, whereas an ultrasonic / level sensor uses a single ultrasonic element for both emission and reception. In a reflective model ultrasonic / level sensor, a single oscillator emits and receives ultrasonic waves alternately. This enables miniaturisation of the sensor head.s The distance can be calculated with the following formula:

Distance $L = 1/2 \times T \times C$

where L is the distance, T is the time between the emission and reception, and C is the sonic speed. (The value is multiplied by 1/2 because T is the time for go-and-return distance.)

2.3 Wheel motion & motor driver node:

- PWM DC Motor Control

PWM, or pulse width modulation is a technique which allows us to adjust the average value of the voltage that's going to the electronic device by turning on and off the power at a fast rate. The average voltage depends on the duty cycle, or the amount of time the signal is ON versus the amount of time the signal is OFF in a single period of time.

- H-Bridge DC Motor Control

On the other hand, for controlling the rotation direction, we just need to inverse the direction of the current flow through the motor, and the most common method of doing that is by using an H-Bridge. An H-Bridge circuit contains four switching elements, transistors or MOSFETs, with the motor at the center forming an H-like configuration. By activating two particular switches at the same time we can change the direction of the current flow, thus change the rotation direction of the motor.

So if we combine these two methods, the PWM and the H-Bridge, we can have a complete control over the DC motor. There are many DC motor drivers that have these features and the L298N is one of them.

3. Implementation details:

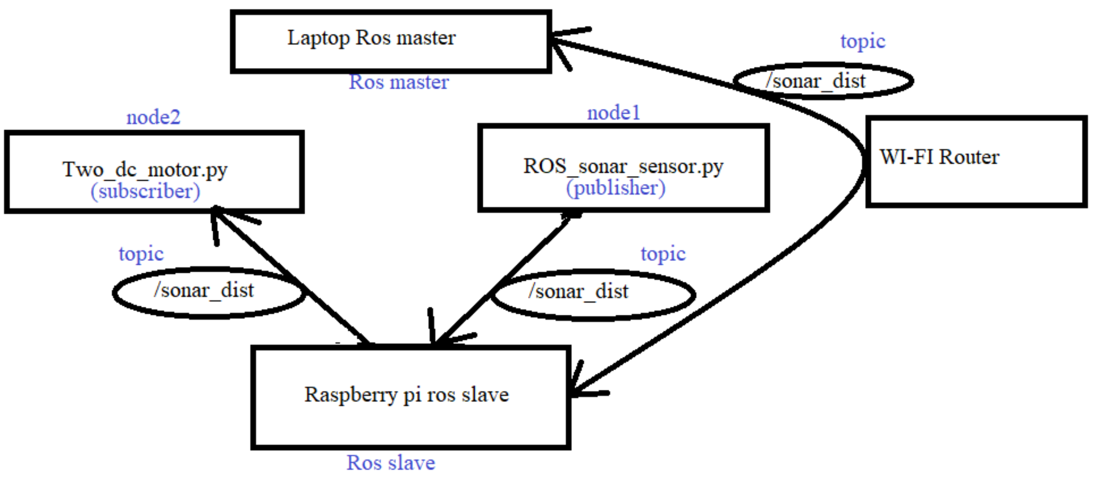


Fig 2: Shows working flow diagram of this project

3.1 Ros Network connection:

ROS is a distributed system. A running ROS system can comprise dozens, even hundreds of nodes, spread across multiple machines. Depending on how the system is configured, any node may need to communicate with any other node, at any time.
As a result, ROS has certain requirements of the network configuration:

- There must be complete, bi-directional connectivity between all pairs of machines, on all ports.
- Each machine must advertise itself by a name that all other machines can resolve.

The primary requirement for getting Raspberry pi 3 model b+ running is connecting to a network so that we can operate it remotely. After both pi3 and Laptop is connected to a network, it is time to set ROS MASTER URI and ROS HOSTNAME. The ROS MASTER URI tells the rest of the nodes at which address they can find the ROS master.

Here we setup the Laptop as a ROS Master and pi3 as a ROS node. Before this is done, we need to determine the IP address of Laptop and Pi3. This can be found by typing "ifconfig" in a terminal. Our IP address are found under wlan0, and it's the numbers proceeding "inet addr:".

3.1.1 Ros Master (Laptop)

- ctrl + Alt + t (Opening new terminal)
- $ hostname -I (to show the ip)
- gedit .bashrc
- go to the bottom of your .bashrc file.
- type

```
$ export ROS_MASTER_URI=http://localhost:11311/
$ export ROS_HOSTNAME=hostname -I
$ export ROS_IP= hostname -I
```

- source .bashrc

3.1.2 Ros Slave (Raspberry pi4)

- ctrl + Alt + t (Opening new terminal)
- $ hostname -I (to show the ip)
- gedit .bashrc
- go to the bottom of your .bashrc file.
- type

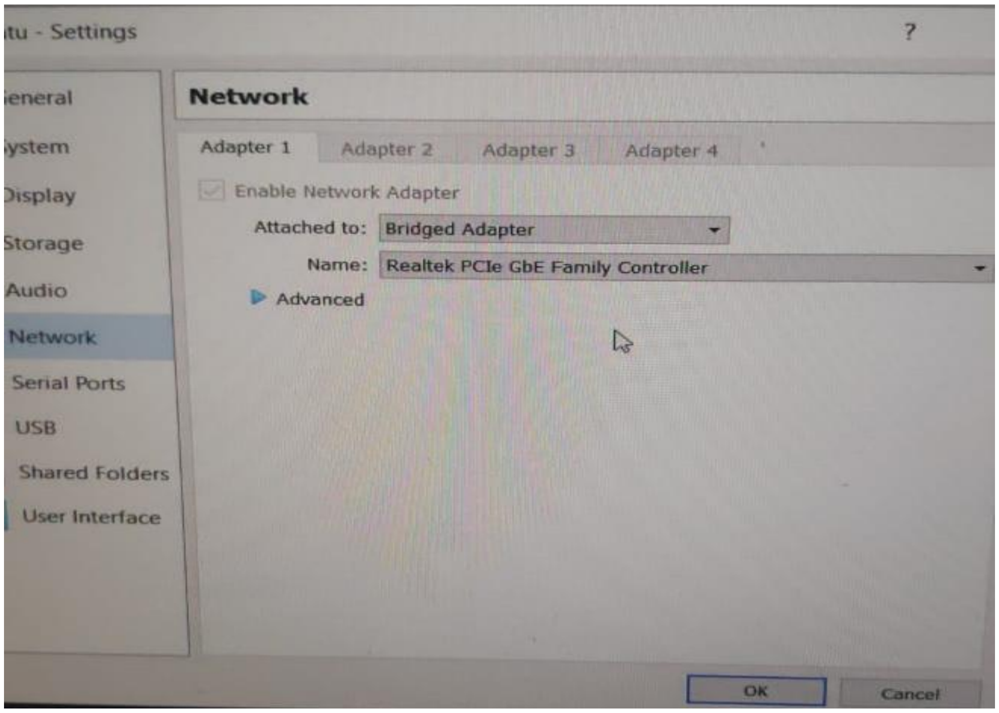
```
$ export ROS_MASTER_URI=http://master_ip:11311/
$ export ROS_HOSTNAME=hostname -I
$ export ROS_IP= hostname -I
```

- source .bashrc

3.1.3 Ros master in virtual box

change the network connection according to the fig2.
Select virtual box Network settings select

- Attached to: Bridged Adapter
- Name : Realtek PCIe GbE Family Controller



Network

Adapter 1

Enable Network Adapter

Attached to: Bridged Adapter

Name: Realtek PCIe GbE Family Controller

Fig 3: shows the network changing option in virtual box for ros network

- ctrl + Alt + t (Opening new terminal)
- $ hostname -I (to show the ip)
- gedit .bashrc
- go to the bottom of your .bashrc file.
- type

```
$ export ROS_MASTER_URI=http://localhost:11311/
$ export ROS_HOSTNAME=hostname -I
$ export ROS_IP= hostname -I
```

- source .bashrc

3.2 ROS Node with topic

3.2.1 Ros_Sonar_sensor Node (node 1)

```
$ cd catkin_ws/src/ros_rover/script/
$ touch Ros_Sonar_sensor.py
$ sudo chmod +x Ros_Sonar_sensor.py
$ gedit Ros_Sonar_sensor.py
```

```
#!/usr/bin/env python

import RPi.GPIO as gpio
import time
import sys
import signal
import rospy
from std_msgs.msg import Float32

def signal_handler(signal, frame): # ctrl + c -> exit program
        print('You pressed Ctrl+C!')
        sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)


class sonar():
    def __init__(self):
        rospy.init_node('sonar', anonymous=True)
        self.distance_publisher = rospy.Publisher('/sonar_dist',Float32, queue_size=1)
        self.r = rospy.Rate(30)
    def dist_sendor(self,dist):
        data = Float32()
        data.data=dist
        self.distance_publisher.publish(data)


gpio.setmode(gpio.BCM)
trig = 23 # 7th
echo = 24 # 6th

gpio.setup(trig, gpio.OUT)
gpio.setup(echo, gpio.IN)

sensor=sonar()
time.sleep(0.5)
print ('-----------------------------------------------------------------------sonar start')
try :
    while True :
        gpio.output(trig, False)
        time.sleep(0.1)
        gpio.output(trig, True)
        time.sleep(0.00001)
        gpio.output(trig, False)
        while gpio.input(echo) == 0 :
            pulse_start = time.time()
        while gpio.input(echo) == 1 :
            pulse_end = time.time()
        pulse_duration = pulse_end - pulse_start
        distance = pulse_duration * 17000
        if pulse_duration >=0.01746:
            #print('time out')
            continue
        elif distance > 300 or distance==0:
            #print('out of range')
            continue
        distance = round(distance, 3)
        #print ('Distance : %f cm'%distance)
        sensor.dist_sendor(distance)

        sensor.r.sleep()

except (KeyboardInterrupt, SystemExit):
    gpio.cleanup()
    sys.exit(0)
except:
    gpio.cleanup()
```

3.2.2 Two_dc_motor Node (node 2)

```
$ touch Two_dc_motor.py
$ sudo chmod +x Two_dc_motor.py
$ gedit Two_dc_motor.py
```

```
#!/usr/bin/env python
import sys
import signal
import rospy
from std_msgs.msg import Float32
import RPi.GPIO as GPIO
from time import sleep


class Motion_vector():
    def __init__(self):

        GPIO.setmode(GPIO.BOARD)

        self.Motor1A = 8
        self.Motor1B = 10
        self.Motor1E = 12

        self.Motor2A = 13
        self.Motor2B = 15
        self.Motor2E = 11

        GPIO.setup(self.Motor1A,GPIO.OUT)
        GPIO.setup(self.Motor1B,GPIO.OUT)
        GPIO.setup(self.Motor1E,GPIO.OUT)

        GPIO.setup(self.Motor2A,GPIO.OUT)
        GPIO.setup(self.Motor2B,GPIO.OUT)
        GPIO.setup(self.Motor2E,GPIO.OUT)
    def forword(self):
        print("Going forword")
        GPIO.output(self.Motor1A,GPIO.HIGH)
        GPIO.output(self.Motor1B,GPIO.LOW)
        GPIO.output(self.Motor1E,GPIO.HIGH)

        GPIO.output(self.Motor2A,GPIO.HIGH)
        GPIO.output(self.Motor2B,GPIO.LOW)
        GPIO.output(self.Motor2E,GPIO.HIGH)

    def backword(self):
        GPIO.output(self.Motor1A,GPIO.LOW)
        GPIO.output(self.Motor1B,GPIO.HIGH)
        GPIO.output(self.Motor1E,GPIO.HIGH)

        GPIO.output(self.Motor2A,GPIO.LOW)
        GPIO.output(self.Motor2B,GPIO.HIGH)
        GPIO.output(self.Motor2E,GPIO.HIGH)
    def left(self):
        print("Going left")
        GPIO.output(self.Motor1A,GPIO.HIGH)
        GPIO.output(self.Motor1B,GPIO.LOW)
        GPIO.output(self.Motor1E,GPIO.HIGH)

        GPIO.output(self.Motor2A,GPIO.LOW)
        GPIO.output(self.Motor2B,GPIO.LOW)
        GPIO.output(self.Motor2E,GPIO.LOW)

    def right(self):
        print("Going right")
        GPIO.output(self.Motor1A,GPIO.LOW)
        GPIO.output(self.Motor1B,GPIO.LOW)
        GPIO.output(self.Motor1E,GPIO.LOW)

        GPIO.output(self.Motor2A,GPIO.HIGH)
        GPIO.output(self.Motor2B,GPIO.LOW)
        GPIO.output(self.Motor2E,GPIO.HIGH)
    def all_stop(self):
        print("All stop")
        GPIO.output(self.Motor1A,GPIO.LOW)
        GPIO.output(self.Motor1B,GPIO.LOW)
        GPIO.output(self.Motor1E,GPIO.LOW)

        GPIO.output(self.Motor2A,GPIO.LOW)
        GPIO.output(self.Motor2B,GPIO.LOW)
        GPIO.output(self.Motor2E,GPIO.LOW)

m_v = Motion_vector()

def callback(data):

    rospy.loginfo(rospy.get_caller_id() + "Sonar data %s", data.data)
    if data.data > 18.2:
        m_v.forword()
    elif data.data < 18.2:
        m_v.right()
        sleep(0.02)
        m_v.all_stop()




def sonar_motor():
    rospy.init_node('two_dc_motor', anonymous=True)
    p = rospy.Subscriber('/sonar_dist', Float32, callback)

    rospy.spin()
if __name__ == '__main__':
    sonar_motor()
```

3.3 Ros code execution

3.3.1 In Laptop (Ros master)

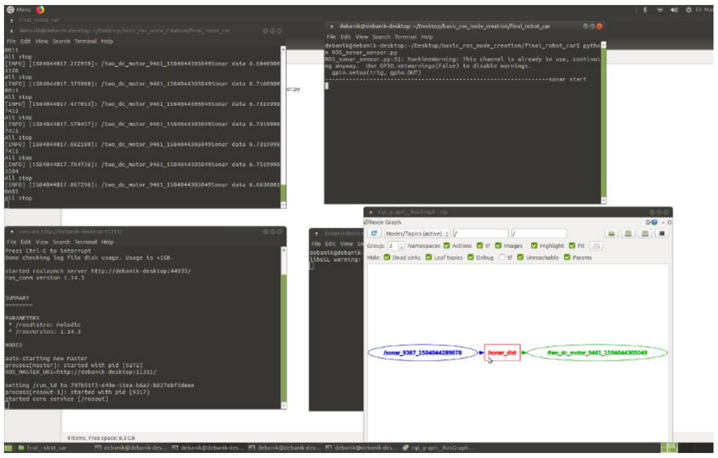
- ctrl + Alt + t (Opening new terminal)
- roscore

3.3.2 In Raspberry Pi (Ros slave)

- ctrl + Alt + t (Opening new terminal)
- rostopic list
- If topics are showing
- then type
- at T1 (terminal 1)
- rosrun ros_rover Ros_Sonar_sensor.py
- at T2 (terminal 2)
- rosrun ros_rover Two_dc_motor.py
- rostopic list

3.3.3 In Laptop (Ros master)

- at T2 (terminal 2)
- rostopic list
- It would show " /sonar_dist " topic
- at T3 (terminal 3)
- rostopic echo /sonar_dist #you see the published distance by sonar sensor
- at T4 (terminal 4)
- rosrun rqt_graph rqt_graph
- rqt_graph will be appeared.
- Fig 4 show distributed node execution including rqt graph

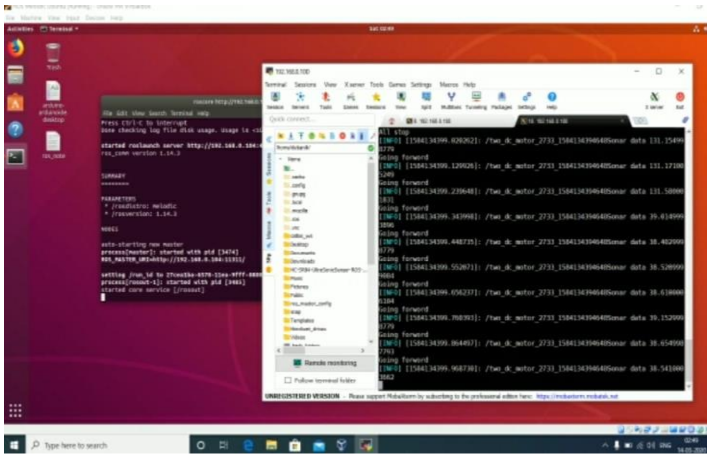


```
8011
All stop
[INFO] [1584044817.272959]: /two_dc_motor_9461_1584044305049Sonar data 6.6840000
1526
All stop
[INFO] [1584044817.375988]: /two_dc_motor_9461_1584044305049Sonar data 6.7160000
8011
All stop
[INFO] [1584044817.477853]: /two_dc_motor_9461_1584044305049Sonar data 6.7319998
7411
All stop
[INFO] [1584044817.579457]: /two_dc_motor_9461_1584044305049Sonar data 6.7319998
7411
All stop
[INFO] [1584044817.682180]: /two_dc_motor_9461_1584044305049Sonar data 6.7319998
7411
All stop
[INFO] [1584044817.784726]: /two_dc_motor_9461_1584044305049Sonar data 6.7519998
5504
All stop
[INFO] [1584044817.887298]: /two_dc_motor_9461_1584044305049Sonar data 6.6630001
6681
All stop
```

```
debanik@debanik-desktop:~/Desktop/basic_ros_node_creation/final_robot_car$ pytho
n ROS_sonar_sensor.py
ROS_sonar_sensor.py:31: RuntimeWarning: This channel is already in use, continui
ng anyway.  Use GPIO.setwarnings(False) to disable warnings.
  gpio.setup(trig, gpio.OUT)
------------------------------------------------------------------sonar start
```

```
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://debanik-desktop:44935/
ros_comm version 1.14.3


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.3

NODES

auto-starting new master
process[master]: started with pid [9272]
ROS_MASTER_URI=http://debanik-desktop:11311/

setting /run_id to 797b51f2-649e-11ea-b6a2-b827ebf5deee
process[rosout-1]: started with pid [9317]
started core service [/rosout]
```

```
debanik@debanik
libEGL warning:
```



```
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1
started roslaunch server http://192.168.0.104:
ros_comm version 1.14.3


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.3

NODES

auto-starting new master
process[master]: started with pid [3474]
ROS_MASTER_URI=http://192.168.0.104:11311/

setting /run_id to 27cea1ba-6578-11ea-9fff-080
process[rosout-1]: started with pid [3485]
started core service [/rosout]
```

```
All stop
[INFO] [1584134399.020262]: /two_dc_motor_2733_1584134394648Sonar data 131.15499
8779
Going forward
[INFO] [1584134399.129926]: /two_dc_motor_2733_1584134394648Sonar data 131.17100
5249
Going forward
[INFO] [1584134399.239648]: /two_dc_motor_2733_1584134394648Sonar data 131.50000
1831
Going forward
[INFO] [1584134399.343998]: /two_dc_motor_2733_1584134394648Sonar data 39.014999
3896
Going forward
[INFO] [1584134399.448735]: /two_dc_motor_2733_1584134394648Sonar data 38.402999
8779
Going forward
[INFO] [1584134399.552071]: /two_dc_motor_2733_1584134394648Sonar data 38.520999
9084
Going forward
[INFO] [1584134399.656237]: /two_dc_motor_2733_1584134394648Sonar data 38.618000
6104
Going forward
[INFO] [1584134399.760393]: /two_dc_motor_2733_1584134394648Sonar data 39.152999
8779
Going forward
[INFO] [1584134399.864497]: /two_dc_motor_2733_1584134394648Sonar data 38.654998
7793
Going forward
[INFO] [1584134399.968730]: /two_dc_motor_2733_1584134394648Sonar data 38.541000
3662
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
Type here to search

fig 4: distributed node execution including rqt graph

Here, rqt_graph shows the sonar data publishing the distance data topic through /sonar_dist topic and two_motor wheel will act accordingly to the published data and ros master computer can see the publish data.

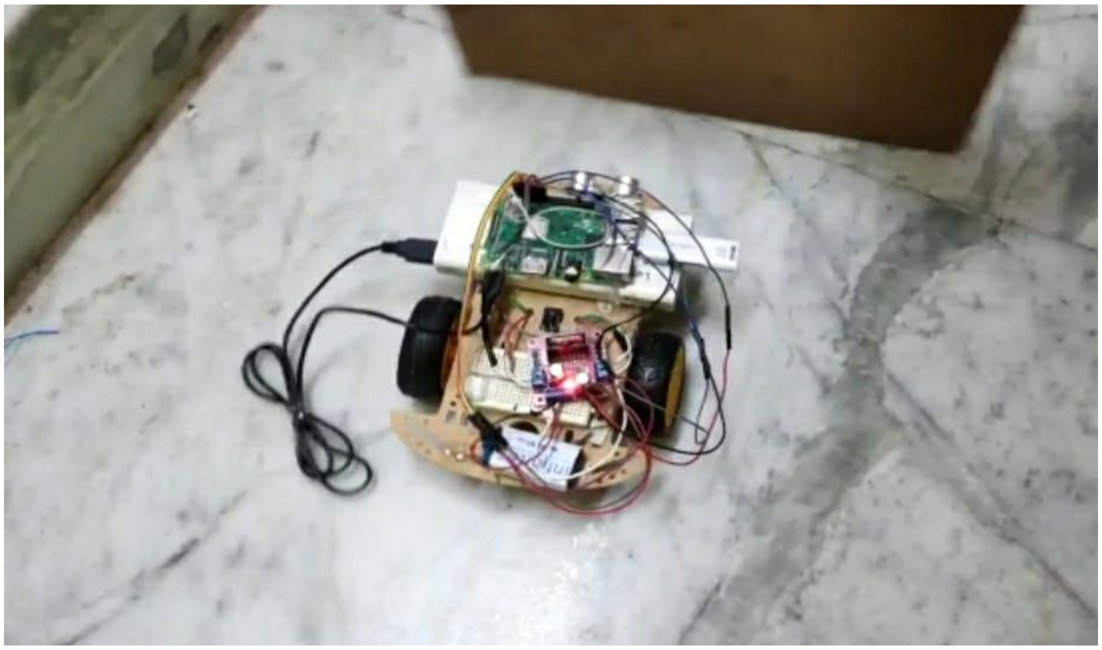


fig 5: This Project ros based realtime robotic car

Conclusion

This study has various further study options, such as, analyzing the power usage for the robot after using the Pi, using the raspberry pi that has ROS into it to run more ROS applications, which could be a challenging. in the near future, if there are a newer raspberry pi released with higher specification or other type of single board computer, with higher processing power and RAM that will allow the project to give better performance. In future slam and computer vision with ros will be introduced.