TECHNICAL REPORT (TurtleBot)

➢ Aim

TurtleBot is a mobile base that can autonomously move around an environment, we wanted to come up with an experiment that captured, however trivially, the value, utility and possibilities of the base station on its own.

➢ Introduction & TurtleBot

TurtleBot is an open source hardware platform and mobile base. TurtleBot can handle vision, localization, communication and mobility. It can autonomously move anything on top of it to wherever that item needs to go, avoiding obstacles along the way.

ROS is an open-source, meta-operating system for our 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.

- What we will need?
  - ✓ TurtleBot 2 with a Kobuki base.

  - ✓ A Laptop
  - ✓ Power station

TECHNICAL REPORT (TurtleBot)

- ✓ Workstation (PC)
- ✓ A wireless router

Main Technical Keys

Robot Operating System (ROS) is one of frameworks for developing the robot. ROS is running in LINUX operating system, in our environment, we use ROS Indigo which is running well in Ubuntu 14.04 LTS.

The main concept of ROS process is peer-to-peer network. It consists of 'node', 'topic', and 'message'.

✓ ROS Distribution version	~	Indigo
✓ Build system	~	catkin
✓ The ROS Visualization tool	~	RViz
✓ ROS Packages		
✓ ROS Bags		
✓ ROS Analyze working	~	Nodes & Topics
✓ Robot base	~	Kobuki Base
✓ Robot Create	~	iRobot
✓ Testing Robot	~	Kobuki
✓ Operations	~	Keyboard & Joystick
✓ Networking	~	Kobuki Base
✓ Robot	~	Turtlebot2

Packages, Simulation & Basic Control

- ✓ Installing ROS with Debian packages
- ✓ Building ROS Packages with Catkin
- ✓ Ubuntu ROS installation instructions, all ROS packages and metapackages
- ✓ Third-Party ROS Packages ROS Wiki

➢ Project

The first goal for the project is navigate the Turtlebot2. Turtlebot2 is one of robots which is full-supported by ROS for its development. Before doing some development into Turtlebot, it is important to check the base of this Turtlebot (Kobuki). Once everything is checked, the Turtlebot is ready to be developed.

TECHNICAL REPORT (TurtleBot)

- TurtleBot Installation & Testing ROS Installation with TurtleBot and Work Station

roscore

- ✓ If it is working correctly

started core service [/rosout]

- ✓ Move Forward Via Teleoperation

roslaunch turtlebot_bringup minimal.launch

roslaunch turtlebot_teleop keyboard_teleop.launch

- Setting Up Networking

Remotely connect to the TurtleBot computer with SSH build in command.
we made a successful connection between TurtleBot+laptop
and the Workstation. When we faced mainly problem as bashrc file to amend as for our IP address and setting up.

- ✓ Testing with terminal

roslaunch turtlebot_bringup minimal.launch

- ✓ at another terminal

roslaunch turtlebot_teleop keyboard_teleop.launch

- Launch RViz and working with that

roslaunch turtlebot_rviz_launchers view_robot.launch

Then we test Teleoperation with Keyboard and Joystick.

TECHNICAL REPORT (TurtleBot)

- Create a Map Via Teleoperation

  roslaunch turtlebot_bringup minimal.launch

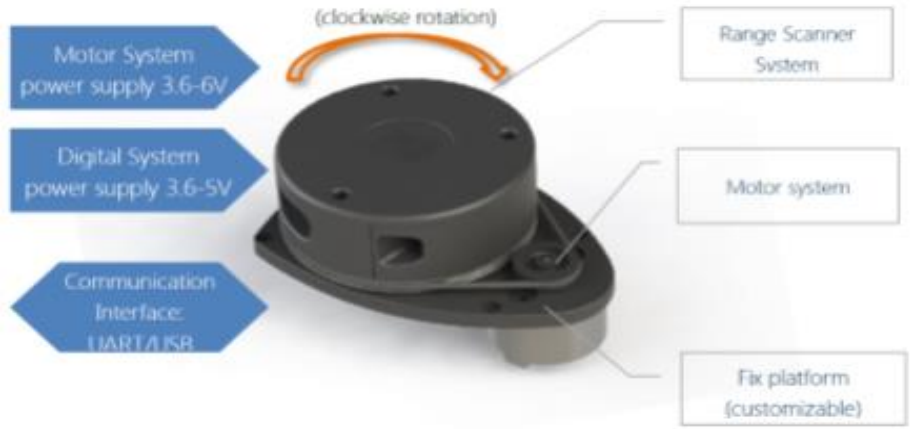
  roslaunch turtlebot_navigation gmapping_demo.launch

  at another terminal

  roslaunch turtlebot_rviz_launchers view_navigation.launch

  roslaunch turtlebot_teleop keyboard_teleop.launch

- RP-LIDAR



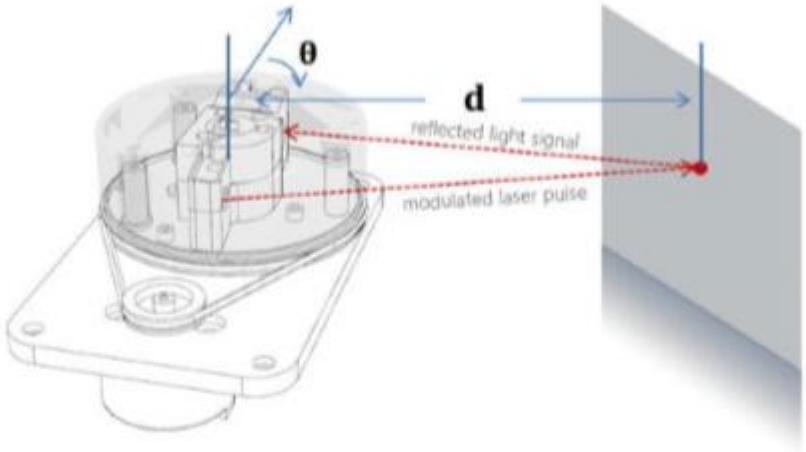
The different RP-LIDAR's parts

RP-LIDAR is a low-cost 360 degree 2D Laser Scanner (LIDAR) system.

It can perform scanning with a frequency of5.5Hzwhen sampling 360 points within 6-meter range.

RP-LIDAR is based on laser triangulation ranging principle: It emits modulated infrared laser signal and the laser signal is then reflected by the object to be detected. The returning signal is sampled by vision acquisition system in RP-LIDAR and the DSP embedded in RP-LIDAR start processing the sample data, output distance value and angle value between object and RP-LIDAR through communication interface.

TECHNICAL REPORT (TurtleBot)



Graphical representation of laser triangulation

The system can measure distance data in more than 2000 times' per second and with high resolution distance output (<1% of the distance):

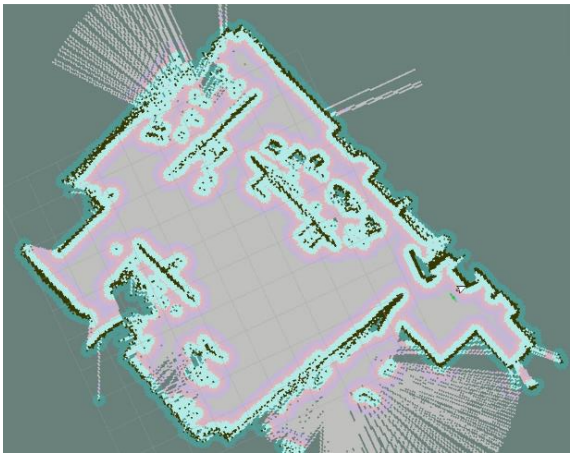
Data Type	Unit	Description
Distance	mm	Current measured distance value
Heading	degree	Current heading angle of the measurement
Quality	level	Quality of the measurement
Start Flag	(Boolean)	Flag of a new scan

Output data given by RP-LIDAR

- SLAM using the gmapping and amcl ROS Packages

At an even higher level, ROS enables our robot to create a map of its environment using the SLAM gmapping package. Once a map of the environment is available, ROS provides the amcl package (adaptive Monte Carlo localization) for automatically localizing the robot based on its current scan and odometry data.

After that we created my_map.pgm & my_ma.yaml files and saved in TurtleBot.



TECHNICAL REPORT (TurtleBot)

- Autonomous Driving

roslaunch turtlebot_bringup minimal.launch

roslaunch turtlebot_navigation amcl_demo.launch map_file:=/location/my_map.yaml

testing and running with rviz

roslaunch turtlebot_rviz_launchers view_navigation.launch --screen

After setting the estimated pose, select “2D Nav Goal” and click the location where we want TurtleBot to go and specify a goal orientation using the same technique we used with “2D Pose Estimate”.TurtleBot should now be driving around autonomously based on our goals.

- Scripts

The main goal of this project is to learn ROS buy producing code to control the TurtleBot. The first step to perform in order to control the TurtleBot is to implement methods for the movement: moving forward Then bring TurtleBot to the specific location. Finally return to initial position.

Going to a Specific Location on our Map with Using Code

Launch script.

python go_to_specific_point_on_map.py

```
import rospy
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, Point, Quaternion

class GoToPose():
    def __init__(self):

        self.goal_sent = False
```

TECHNICAL REPORT (TurtleBot)

```
        rospy.on_shutdown(self.shutdown)

        # Tell the action client that we want to spin a thread by default
        self.move_base = actionlib.SimpleActionClient("move_base",
MoveBaseAction)
        rospy.loginfo("Wait for the action server to come up")

        # Allow up to 5 seconds for the action server to come up
        self.move_base.wait_for_server(rospy.Duration(5))

    def goto(self, pos, quat):

        # Send a goal
        self.goal_sent = True
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()
        goal.target_pose.pose = Pose(Point(pos['x'], pos['y'], 0.000),
                                     Quaternion(quat['r1'], quat['r2'], quat['r3'], quat['r4']))

        # Start moving
        self.move_base.send_goal(goal)

        # Allow TurtleBot up to 60 seconds to complete task
        success = self.move_base.wait_for_result(rospy.Duration(60))

        state = self.move_base.get_state()
        result = False

        if success and state == GoalStatus.SUCCEEDED:
            # We made it!
            result = True
        else:
            self.move_base.cancel_goal()

        self.goal_sent = False
        return result

    def shutdown(self):
        if self.goal_sent:
            self.move_base.cancel_goal()
```

TECHNICAL REPORT (TurtleBot)

```
        rospy.loginfo("Stop")
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        rospy.init_node('nav_test', anonymous=False)
        navigator = GoToPose()

        # Customized our location
        position = {'x': -1.925, 'y' : 1.829}
        quaternion = {'r1' : 0.000, 'r2' : 0.000, 'r3' : 0.979, 'r4' : -0.202}

        rospy.loginfo("Go to (%s, %s) pose", position['x'], position['y'])
        success = navigator.goto(position, quaternion)

        if success:
            rospy.loginfo("Hooray, reached the desired pose")
        else:
            rospy.loginfo("The base failed to reach the desired pose")

        # Sleep to give the last log messages time to be sent
        rospy.sleep(1)

    except rospy.ROSInterruptException:
        rospy.loginfo("Ctrl-C caught. Quitting")
```

And I implement the same code but changed initial position to return back.

Launch script.

```
        python go_back_to_specific_point_on_map.py
```

```
import rospy
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, Point, Quaternion

class GoToPose():
    def __init__(self):
```

TECHNICAL REPORT (TurtleBot)

```
    self.goal_sent = False

        # What to do if shut down (e.g. Ctrl-C or failure)
        rospy.on_shutdown(self.shutdown)

        # Tell the action client that we want to spin a thread by default
        self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)
        rospy.loginfo("Wait for the action server to come up")

        # Allow up to 5 seconds for the action server to come up
        self.move_base.wait_for_server(rospy.Duration(5))

def goto(self, pos, quat):

    # Send a goal
    self.goal_sent = True
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose = Pose(Point(pos['x'], pos['y'], 0.000),
                                 Quaternion(quat['r1'], quat['r2'], quat['r3'], quat['r4']))

        # Start moving
    self.move_base.send_goal(goal)

        # Allow TurtleBot up to 60 seconds to complete task
        success = self.move_base.wait_for_result(rospy.Duration(60))

    state = self.move_base.get_state()
    result = False

    if success and state == GoalStatus.SUCCEEDED:
        # We made it!
        result = True
    else:
        self.move_base.cancel_goal()

    self.goal_sent = False
    return result

def shutdown(self):
    if self.goal_sent:
        self.move_base.cancel_goal()
    rospy.loginfo("Stop")
    rospy.sleep(1)
```

TECHNICAL REPORT (TurtleBot)

```
if __name__ == '__main__':
    try:
        rospy.init_node('nav_test', anonymous=False)
        navigator = GoToPose()

        # Customized our location to back to normal place
        position = {'x': 4.237, 'y' : 6.213}
        quaternion = {'r1' : 0.000, 'r2' : 0.000, 'r3' : 0.979, 'r4' : -0.202}

        rospy.loginfo("Go to (%s, %s) pose", position['x'], position['y'])
        success = navigator.goto(position, quaternion)

        if success:
            rospy.loginfo("Hooray, reached the desired pose")
        else:
            rospy.loginfo("The base failed to reach the desired pose")

        # Sleep to give the last log messages time to be sent
        rospy.sleep(1)

    except rospy.ROSInterruptException:
        rospy.loginfo("Ctrl-C caught. Quitting")
```

- Setting up initial pose without *rviz*

```
bscv@CO-ROBOT01:~$ rostopic echo /initialpose
header:
  seq: 1
  stamp:
    secs: 1525877602
    nsecs:  46830625
  frame_id: map
pose:
  pose:
    position:
      x: 4.41164684296
      y: 6.0491361618
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.973755707403
      w: 0.227595743151
  covariance: [0.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.25, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.06853891945200942]
```

➢ Other Technical summary from the Project.

The first goal for the project is navigate the Turtlebot2. Turtlebot2 is one of robots which is full-supported by ROS for its development. Before doing some development into Turtlebot, it is important to check the base of this Turtlebot (Kobuki). Once everything is checked, the Turtlebot is ready to be developed.

TECHNICAL REPORT (TurtleBot)

Developing Turtlebot to make Turtlebot doing a specific task with ROS should not be done directly in the real Turtlebot, since it will cause many troubles which can damage the robot. In ROS, there are many simulations which can be used for doing this task, such as 'Bag' and 'Gazebo'. Bag is the recorded data of the previous robot execution whereas Gazebo is the simulation package which represent the real world of the Turtlebot and its environment.

Running the turtlebot both in bag and gazebo are using the 'fake' turtlebot by launching some launch files of the fake_turtlebot and gazebo_turtlebot packages. Once the fake turtlebot is running, it will publish some topics which can be seen in terminal LINUX by rostopic command. All topics, the interaction between nodes that happens in Turtlebot, can be represented into graph by using rqt_graph command, and the visualization of the turtlebot can be seen by using rviz command.

➢ Issues encountered

Battery life time down without we know	Then we must wait few hours
Default robot speed is high	we limit for safety and control
I used rviz to set initial position	then I used *rostopic echo/initialpose*
RPLIDAR is not working properly when is low power	Because laptop power down
Sometimes robot lost position at mid-way	May be RPLIDAR

➢ Conclusion

ROS includes packages to run a number of robots in simulation so that we can test our programs before venturing into the real world. ROS help us to work and do fantastic tasks on Robot. This is technically very interesting for study Robotic Engineering.

TECHNICAL REPORT (TurtleBot)

The TurtleBot gives a new dimension of possibilities to study all above this project. And our aim of this project is to develop an autonomous application for controlling with TurtleBot.

The goal was extending to study and experience when we doing the simulation and testing in every lab. we faced problems during the lab and professors help me out to continue the success in this project.

"Man is striving to make the machines do tasks that humans can do or try to do through embedded distributed intelligence."