SIT310 – Task 4

Navigation Strategies in ROS

Overview

This task focuses on using ROS to control your physical robot. You will learn how to control your robot using messages on ROS topics. You will build a complete ROS application that reacts to sensor readings from the robot and then issues commands to the robot via a ROS topic. Your robot will initially avoid walls, but the assessment this week encourages you to expand this to full maze solving.

Task requirements

a. Ensure you have already completed task 1, task 2 and task 3.
b. To complete this task, you will need to use ROS on the raspberry Pi as we will be using your physical robot
c. If you didn't attend the lecture this week, then either watch the lecture online and/or review the PowerPoint slides on the unit site.
d. Read the introduced concepts / theories / recommended readings below.
e. Read the task instructions
f. Complete the activities in the task guide
g. Review your progress with your lab tutor or cloud tutor.

Task guide

So far, we have only received data from the robot. This lab will close the loop of control and reactively issue commands to control the robots motion based on this sensor data. This lab will also go further with ROS and build full applications in ROS. By the end of this lab, your robot should be moving in response to sensor input. You should aspire to have the robot fully acting as a wall-following/maze solving robot.

1. Robot Control

   1.1. We want to be able to control the robot with ROS, so the first step is moving the robot around.

   - Upload the following to your robot using the Arduino IDE. This will have the robot move forward for 1 second, wait for a second, then move backwards and wait for a second. Don't forget to have batteries in the robot and switch the power on.

```
#include <Wire.h>
#include <Zumo32U4.h>

Zumo32U4Motors motors;
Zumo32U4ButtonA buttonA;

void setup()
{
  // Uncomment if necessary to correct motor directions:
  //motors.flipLeftMotor(true);
  //motors.flipRightMotor(true);

  // Wait for the user to press button A.
  buttonA.waitForButton();

  // Delay so that the robot does not move away while the user is
  // still touching it.
  delay(1000);
}

void loop()
{
  // Run robot forward for 1 second.
  motors.setLeftSpeed(100);
  motors.setRightSpeed(100);
  delay(1000);

  // Wait robot for 1 second.
  motors.setLeftSpeed(0);
  motors.setRightSpeed(0);
  delay(1000);

  // Run robot forward for 1 second.
  motors.setLeftSpeed(-100);
  motors.setRightSpeed(-100);
  delay(1000);
  // Wait robot for 1 second.
  motors.setLeftSpeed(0);
  motors.setRightSpeed(0);
  delay(1000);

}
```

1.2. Next, we'd like to be able to control the robot using ROS.

- Upload the following Arduino code

```
#define USE_USBCON

#include <Wire.h>
#include <Zumo32U4.h>

#include <ros.h>
#include <std_msgs/UInt16.h>

ros::NodeHandle  nh;
void ros_handler( const std_msgs::UInt16& cmd_msg){
  if(cmd_msg.data == 0) stop();
  if(cmd_msg.data == 1) forward(1000);
  if(cmd_msg.data == 2) backward(1000);
}


ros::Subscriber<std_msgs::UInt16> sub("control", ros_handler);

Zumo32U4Motors motors;

void setup()
{
  // Uncomment if necessary to correct motor directions:
  //motors.flipLeftMotor(true);
  //motors.flipRightMotor(true);

  nh.initNode();
  nh.subscribe(sub);
}

void forward(int time)
{
  motors.setLeftSpeed(100);
  motors.setRightSpeed(100);
  delay(time);
}

void backward(int time)
{
  motors.setLeftSpeed(-100);
  motors.setRightSpeed(-100);
  delay(time);
}

void stop()
{
  motors.setLeftSpeed(0);
  motors.setRightSpeed(0);
}

void loop()
{
  nh.spinOnce();
  delay(1);
}
```

- Make sure you have roscore running

```
$ roscore
```

- Run the rosserial ROS node

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

- You can, but you don't need to run rostopic on the control topic.

```
$ rostopic echo control
```

- You can now issue commands to your robot using ros.

```
$ rostopic pub -1 /control std_msgs/UInt16 1
```

- If all goes well, your robot should move forward. To get it to stop, issue.

```
$ rostopic pub -1 /control std_msgs/UInt16 0
```

- Issue the following to get it to go backwards and stop.

```
$ rostopic pub -1 /control std_msgs/UInt16 2
$ rostopic pub -1 /control std_msgs/UInt16 0
```

- This is the beginnings of an Arduino Node (like an API) for the robot for ROS.

1.3. Now that we have this working, wouldn't it be cool to have this working with the turtlesim applications?

- Firstly, lets figure out what the messages are passing around for controlling turtlesim.
- Make sure you have roscore running

```
$ roscore
```

- Run turtlesim as before:

```
$ rosrun turtlesim turtlesim_node
```

- You can publish messages using the turtle_teleop_key program. Run it as follows:

```
$ rosrun turtlesim turtle_teleop_key
```

- Now look at the type of messages using

```
$ rostopic type /turtle1/cmd_vel
geometry_msgs/Twist
```

- Great, we know the format the messages are in. You can also see more detail by:

```
$ rostopic echo /turtle1/cmd_vel
```

- It doesn't require a lot to make our robot respond to these messages. Use the following Arduino code.

```
#define USE_USBCON

#include <Wire.h>
#include <Zumo32U4.h>

#include <ros.h>

#include <geometry_msgs/Twist.h>
#include <std_msgs/UInt16.h>

ros::NodeHandle  nh;
void ros_handler( const geometry_msgs::Twist& cmd_msg) {
  float x = cmd_msg.linear.x;
   if(x == -2.0) backward(100);
   if(x == 2.0) forward(100);
  stop();
}

ros::Subscriber<geometry_msgs::Twist> sub("/turtle1/cmd_vel", ros_handler);

Zumo32U4Motors motors;

void setup()
{
  // Uncomment if necessary to correct motor directions:
  //motors.flipLeftMotor(true);
  //motors.flipRightMotor(true);

  nh.initNode();
  nh.subscribe(sub);
}

void forward(int time)
{
  motors.setLeftSpeed(100);
  motors.setRightSpeed(100);
  delay(time);
}

void backward(int time)
{
  motors.setLeftSpeed(-100);
  motors.setRightSpeed(-100);
  delay(time);
}

void stop()
{
  motors.setLeftSpeed(0);
  motors.setRightSpeed(0);
}

void loop()
{
  nh.spinOnce();
  delay(1);
}
```

- Run the rosserial ROS node

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM
```

- Now, when you press up and down in the turtle_teleop_key program window. You should see the turtle move forward in the turtlesim window, and your robot should also move back and forward.

1.4. You can probably guess what this task is. We have only completed basic forward and back functionality. You should complete the remaining functionality by having the robot turn left and right on demand.

2. Managing Multiple topics and data

2.1. Before we move on to creating full ROS applications. We should check that rosserial and the Arduino programs we are writing supports subscribing to and publishing to multiple ROS topics at once. This way we can receive sensor readings and also control the robot at the same time.

2.2. The following code is a merge of the code from this lab 1.3 and the sensor code from Lab 3 task 4.5. Upload this code to your Arduino

```
#define USE_USBCON

#include <Wire.h>
#include <Zumo32U4.h>

#include <ros.h>

#include <geometry_msgs/Twist.h>
#include <std_msgs/Int8.h>

ros::NodeHandle  nh;

std_msgs::Int8 msg;
ros::Publisher pub("sensor", &msg);

void ros_handler( const geometry_msgs::Twist& cmd_msg) {
  float x = cmd_msg.linear.x;
   if(x == -2.0) backward(100);
   if(x == 2.0) forward(100);
  stop();
}

ros::Subscriber<geometry_msgs::Twist> sub("/turtle1/cmd_vel", ros_handler);

Zumo32U4ProximitySensors proxSensors;
```

```
bool proxLeftActive;
bool proxFrontActive;
bool proxRightActive;

Zumo32U4Motors motors;

void setup()
{
  // Uncomment if necessary to correct motor directions:
  //motors.flipLeftMotor(true);
  //motors.flipRightMotor(true);

  nh.initNode();
  nh.subscribe(sub);

  proxSensors.initThreeSensors();

  nh.initNode();
  nh.advertise(pub);
}

void forward(int time)
{
  motors.setLeftSpeed(100);
  motors.setRightSpeed(100);
  delay(time);
}

void backward(int time)
{
  motors.setLeftSpeed(-100);
  motors.setRightSpeed(-100);
  delay(time);
}

void stop()
{
  motors.setLeftSpeed(0);
  motors.setRightSpeed(0);
}

void publishReadings()
{
  msg.data = proxSensors.countsFrontWithLeftLeds();
  pub.publish( &msg);
  nh.spinOnce();
  delay(100);
}

void loop()
{
static uint16_t lastSampleTime = 0;

  if ((uint16_t)(millis() - lastSampleTime) >= 100)
  {
    lastSampleTime = millis();

    // Send IR pulses and read the proximity sensors.
    proxSensors.read();

    // Just read the proximity sensors without sending pulses.
    proxLeftActive = proxSensors.readBasicLeft();
    proxFrontActive = proxSensors.readBasicFront();
    proxRightActive = proxSensors.readBasicRight();

    publishReadings();

  }
  nh.spinOnce();
  delay(1);
}
```

- To fully demo the above. You'll need to do the following.
- Make sure you have roscore running

```
$ roscore
```

- Run the rosserial ROS node

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

- Run turtlesim as before:

```
$ rosrun turtlesim turtlesim_node
```

- You can publish messages using the turtle_teleop_key program. Run it as follows:

```
$ rosrun turtlesim turtle_teleop_key
```

- Echo the turtle topic.

```
$ rostopic echo /turtle1/cmd_vel
```

- Echo the sensor topic.

```
$ rostopic echo sensor
```

- Graph the output of the sensor

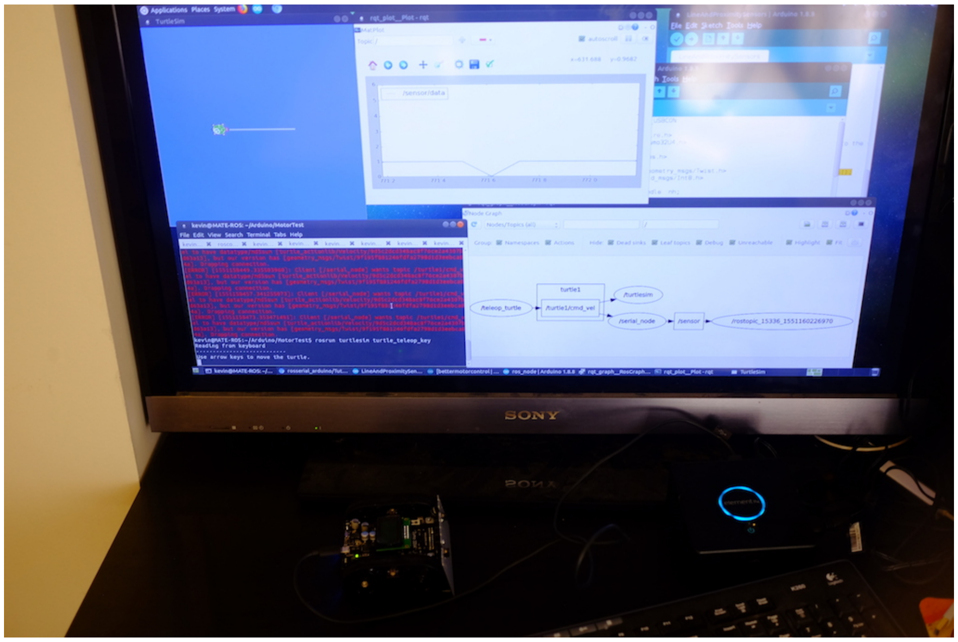
```
$ rosrun rqt_graph rqt_graph
```

- Display the node information.

```
$ rosrun rqt_plot rqt_plot
```

- When you move around the robot with the turtle_teleop_key, you should get messages displayed, the robot moving, turtlesim moving and it you have a suitable object in near the robot, changing sensor readings.

- Something like this:



```
kevin@MATE-ROS:~/Arduino/MotorTest$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

2.3. You should improve your Arduino code in 2.2 to fully control the robot and also read from multiple sensors into multiple topics. We will be using this code in the future. Call it something sensible like ROS_Node.ino

3. Create a project

3.1. Create a project.

- Navigate to your carkin_ws folder, create a project and test build it.

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg scared_robot std_msgs geometry_msgs rospy
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash
```

- Navigate to the new project and create a scripts folder.

```
$ roscd scared_robot
$ mkdir scripts
$ cd scripts
```

- Create a file, with the following code, and save it in the scripts folder of scared_robot as react.py

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int8
from geometry_msgs.msg import Twist

rospy.init_node('scared_robot', anonymous=True)
pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)


def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'I heard %d', data.data)
    if data.data > 4:
        vel_msg = Twist()
        vel_msg.linear.x = -2
        #Since we are moving just in x-axis
        vel_msg.linear.y = 0
        vel_msg.linear.z = 0
        vel_msg.angular.x = 0
        vel_msg.angular.y = 0
        vel_msg.angular.z = 0
        pub.publish(vel_msg)


def listener():

#    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber('sensor', Int8, callback)
    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

- This code will consume sensor readings from the front sensor and when it detects an obstacle close to its front, it will back off like a scared little robot.
- Because we have created this as part of ROS project, you can run the above ROS program as follows, from any terminal.

```
$ rosrun scared_robot react.py
```

- This is our first useful-ish program.

4. A note about editors.

   4.1. There are many editors suitable for editing programs in Linux.

   4.2. You can use a command line editor such as emacs (or vi). e.g.

```
$ sudo apt-get install emacs-nox
$ emacs filename
A guide is here: http://www.jesshamrick.com/2012/09/10/absolute-beginners-gui
de-to-emacs/
```

   4.3. There are a lot of editors, for example, see:

- https://www.tecmint.com/best-python-ides-for-linux-programmers/
- And those suggested by ROS: http://wiki.ros.org/IDEs
- Most can be installed through apt-get at the command line. The following installs the Eric and Spyder editors for you to try. Once installed, they are in the programming menu.

```
$ sudo apt-get install eric spyder
```

- If you find a more suitable editor, post of the unit site discussion forum.

5. Wall avoidance navigation strategy.

   5.1. You've now got all the pieces in place to build useful ROS applications with your robot. The rest of this week and the 2nd assessment is to build a simple navigation robot, that just uses wall avoidance as its basic strategy.

   5.2. Start from scratch, or adapt the programs you have so far. Your robot should constantly move forward slowly, but when it detects a wall to its left - it will move right, and when it detects a wall to its right it will turn left. Else it will continue forwards.

   5.3. Use the ROS and Zumo resources online to help you. You should try to achieve as much as possible of the assessment as follows.

ASSESSMENT

This week's lab assessment is based on the output of this lab as follows. You should start this now. Depending on what you achieve, you should upload a short paragraph of explanation of what you have done, your code in a zip and a short video to the quiz "Lab Assessment 2" on the unit site. Leave the quiz questions for more advanced functionality blank. Don't worry, there is plenty of time to get great grades in the unit!

- P – Robot controlled with turtlesim key commands, including left and right.
- C – Robot avoiding walls on one side
- D – Robot capable of avoiding walls on both sides
- HD – Accurate wall avoidance using the Zumo's full sensor array. Advanced solutions can also consider all IR emitters to each sensor.

You've made a lot of progress with ROS so far. In the next 2 weeks we will really start to use the ROS abilities by using the ROS navigation stack.

6. Additional resources

   6.1. Learn about linux

   - https://www.computerworld.com/article/2598082/linux/linux-linux-command-line-cheat-sheet.html

-

6.2. Learn about ROS.

- http://wiki.ros.org
- A Gentle Introduction to ROS: https://www.cse.sc.edu/~jokane/agitr/
- http://wiki.ros.org/kinetic/Installation/Ubuntu
- http://wiki.ros.org/ROS/Tutorials
- https://www.pololu.com/docs/0J63/4
- http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup
- http://wiki.ros.org/rosserial_arduino/Tutorials/Hello%20World