SIT310 – Task 3

Data in ROS

Overview

This task focuses on managing data in ROS. You will use ROS tools to display, graph, record and playback data. You will then learn about your physical robot, setup your system to interact with it, and program it using the Arduino IDE. Finally, you will access the sensors on the robot from ROS.

Task requirements

a. Ensure you have already completed task 1 and task 2
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

For this lab we will be heavily using tutorials from the technologies we are using. The instructions presented here are cut down versions of these tutorials to allow us to rapidly develop robotic applications. If you want further information, have a look at the "Additional resources" section below.
This task will guide you through creating a complete publish-subscribe ROS application.

1. ROS Data

1.1. We have already seen how we can print data from a topic to the terminal using "rostopic echo name". There are a few more data tricks in ROS to speed up development.

1.2. Rostopic hz

- Rostopic hz topicname report the rate at which data in published.
- Let's try it with turtlesim.
- Make sure roscore is running:

```
$ roscore
```

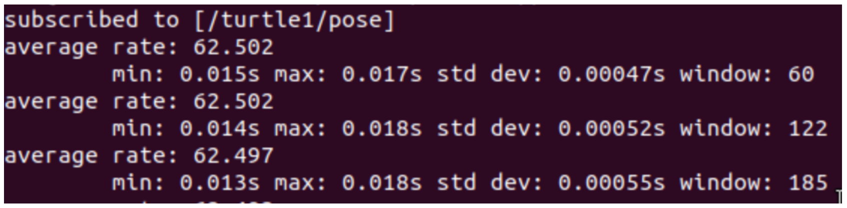
- Run turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

- Run rostopic hz as follows:

```
$  rostopic hz /turtle1/pose
```

- You can see from the output that turtlesim_node is updating the /turtle1/pose at a rate of 60 HZ.



```
subscribed to [/turtle1/pose]
average rate: 62.502
        min: 0.015s max: 0.017s std dev: 0.00047s window: 60
average rate: 62.502
        min: 0.014s max: 0.018s std dev: 0.00052s window: 122
average rate: 62.497
        min: 0.013s max: 0.018s std dev: 0.00055s window: 185
```

1.3. rql_plot

- rql_plot will plot data on a scrolling graph. This is really useful to quickly see the behavour of data.
- run the following to start the rqt_plot program

```
$ rosrun rqt_plot rqt_plot
```

- The + and – buttons will add and remove topics. Add the /turtle1/pose/x and /turtle1/pose/y topics.
- As in the last tutorial, use keyboard control to move the turtle around, as follows.

```
$ rosrun turtlesim turtle_teleop_key
```

- When you move the turtle, you will see the X and Y coordinates of the turtle change.
- ddss

1.4. Recording and playback of data in ROS

- It is possible to record data from ROS topics, and even play it back. This is very useful for testing during development.
- Perform the following to record turtlesim data.
- Make sure roscore is running:

```
$ roscore
```

- Run turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

- Run the keyboard control program

```
$ rosrun turtlesim turtle_teleop_key
```

- To find out what topics are in use, execute the following.

```
rostopic list -v
```

- The important topic is /turtle1/cmd_vel which are the commands from teleop_turtle_key used by turtlesim
- Use the following to record some messages. Once rosbag is running, use your keyboard to move the turtlesim around for around 10 seconds. When you are done, kill the rosbag program by Ctrl-C.

```
$ mkdir ~/bagfiles
$ cd ~/bagfiles
$ rosbag record -a
```

- A file would have been created in the bagfiles folder. You can find information out about the recording by typing the following

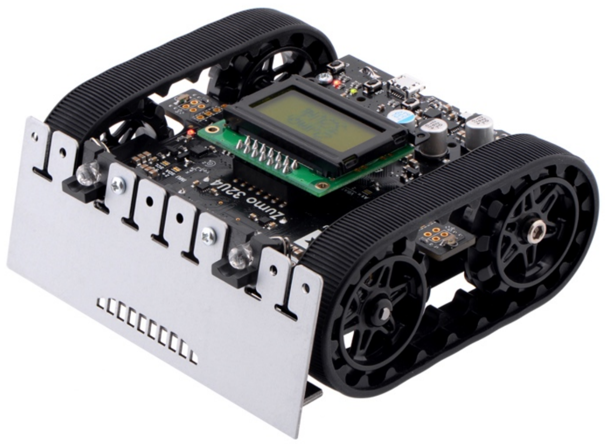
```
$ rosbag info <your bagfile>
```

- You can play back your recording very easily with:

```
$ rosbag play <your bagfile>
```

- Just should restart your turtlesim, then play back your recording. You'll see the turtle move around. Very useful!

2. Your robot (**Pololu Zumo 32U4)**



2.1. The robot you are using is a Pololu Zumo 32U4. This is an extremely capable platform, with a lot of sensors, a low power Arduino compatible microcontroller, and powerful servos. It is an extremely open platform, which has extensive documentation and is very extendable. You should spend some time looking at the documentation at https://www.pololu.com/docs/0J63. Some of it is very in depth though, so just browse the content that is most of interest to you. Chat to your peer and your lab tutor. You learn a lot from exploring these resources.

2.2. Building your robot

2.2.1. If you bought a Zumo 32U4 kit (and 2 motors) to save some money, then you have a bit of work to do. Follow the instructions carefully here: https://www.pololu.com/docs/0J63/4

- Details of the kit are here: https://www.robotgear.com.au/Product.aspx/Details/4631-Zumo-32U4-Robot-Kit-No-Motors

2.2.2. If you bought yours fully assembled, then awesome, you are ready to go.

2.3. Programming your robots

2.3.1. We are going to test the robot using the Arduino IDE to start with. This will also be useful for quick prototyping.

2.3.2. We need to install a recent version of Arduino (not the default Mate one).

- In your web browser, go to **www.arduino.cc/en/Main/Software**
- Download the main Linux Arm version. At the time of writing this was 1.8.8.
- Open a terminal and type of the following.

```
$ cd ~/Downloads
(remember you can use tab completion here)
$ tar -xvf arduino-1.8.8-linuxarm.tar.xz
$ sudo mv arduino-1.8.8 /opt
$ cd /opt/arduino-1.8.8
$ chmod +x install.sh
$ ./install.sh
```

2.3.3. When it is finished, load the Arduino IDE from the menu. It will be under "Programming -> Arduino IDE".

2.3.4. You will need to give yourself permissions to access serial ports – in the terminal type the following. Note that you will have to logout and log back for this to work.

```
$ sudo usermod -a -G dialout $USER
```

2.3.5. We need to add support for the Arduino on the robot.

- In the Arduino IDE, click File->Preferences. In the "Additional Boards Manager URLs:" copy the following text. Click OK. This will add a new sources of board types. **https://files.pololu.com/arduino/package_pololu_index.json**
- Select "Tools" -> "Board: Arduino…" ->"Board Manager"
- It will update some information, then in the search box, type "polou". A single option should appear, just click install. When it is finished, click "close".
- You should now see the board listed in "Tools"->"Board"->"Pololu A-Star 32U4". You can select it.

2.3.6. We don't want to waste any time writing small programs to learn the functionality of the Zumo 32U4 – Pololu have some example programs to help us.

- Go to https://github.com/pololu/zumo-32u4-arduino-library
- Click Releases. Click the latest version (1.1.4 at the time of writing). And download the zip file. Save it to your downloads folder as usual.
- Move it to the Arduino library folder as follows:

```
$ cd ~/Downloads
$ unzip zumo-32u4-arduino-library-1.1.4.zip
(remember you can use tab completion here)
$ mv zumo-32u4-arduino-library-1.1.4 ~/Arduino/libraries/Zumo32U4
```

2.3.7. Close and restart the Arduino IDE from the applications menu

2.3.8. You should see a series of examples for the Zumo32U4, in "File"->"Examples"->"Zumo32U4" (at the bottom)

2.4. Testing your robot

2.4.1. Open the Zumo32U4 example, BlinkLEDs. Connect your robot via usb cable to your Raspberry pi. Ensure the Board is "Pololu A-Star 32U4". The port will probably be "/dev/ttyACM0". Click "Upload" on the menu bar. The lights will flash on the robot. Have a read through the code to understand what is going on.

2.4.2. Open the Zumo32U4 example, Buttons. Upload this. Open the serial console by clicking the icon in the top right. Read through the code to understand what is going on when you press button B or C.

2.4.3. Open the Zumo32U4 example, LineAndProximitySensors. Upload this. After an initial calibration, it will use the LCD to display proximity to e.g. your hand. You can also use the serial console as it prints to this. Read through the code to understand what is going on.

2.4.4. Open the Zumo32U4 example, MotorTest. Upload this. Hmm, why isn't anything happening? Yes, the motors are driven from the batteries. Insert 4 AA rechargeable batteries into the compartment at the bottom. Hmm, nothing happening yet? It stumped your Unit Chair, until he noticed the battery switch near the USB cable. Switch it. Unplug your USB cable. Put the robot on the floor and hit A. Read through the code to understand how its moving. It's a little tricky and you might have to refer to the user guide: https://www.pololu.com/docs/0J63/4

3. Using your robot in ROS

Using the Arduino IDE, and Arduino is fine for simple problems. But quickly becomes unrealistic for difficult problems. We want to be able to use our nice robot in a more powerful development environment. We will now concentrate on using our robot in ROS.

3.1. rosserial

3.1.1. There are many ways to connect your robot to ROS, but one of the simplest is to have a small program that sits on the Arduino and listens from commands from ROS or sends sensor data to ROS. The most straightforward way of doing this is using rosserial. rosserial is a ROS package, see here - http://wiki.ros.org/rosserial. We will be using the rosserial_arduino package with communication using serial over USB.

3.2. rosserial hello world

3.2.1. Open a terminal and type the following. This will install the roserial packages for the Arduino IDE.

```
$ apt-get install ros-kinetic-rosserial-arduino ros-kinetic-rosserial
```

3.2.2. The following actually puts the rosserial libraries in the right location for the Arduino IDE to find them.

```
$ cd ~/Arduino/libraries
$ rm -rf ros_lib
$ rosrun rosserial_arduino make_libraries.py .
```

3.2.3. Open the Arduino IDE – or restart it if you had it open already. You will find all the ros_lib examples in "File"->"Examples"->"ros_lib" at the bottom

3.2.4. Open the ros_lib example sketch "HelloWorld". Add the following to just before the include statements, you'll then have to save the sketch.

```
#define USE_USBCON
```

3.2.5. Connect your robot, and upload it as normal.

3.2.6. Make sure you have roscore running in a terminal. In another terminal, execute the following:

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

3.2.7. And in another terminal, execute the following:

```
$ rostopic echo chatter
```

3.2.8. You should recognise this as its just printing anything on the ROS topic chatter. You should get data appearing. You should recognise this as quite important, as it allows us to get data from your robot to ROS.

4. Sensor Data from the robot in ROS

4.1. Let's do something a bit more advanced. We'd like to be able to read sensor data from the proximity sensors on the robot, feed it into ROS and graph it using rqt_plot

CHALLENGE: If you feel up for it. Try and achieve the above yourself! Tell your lab tutor and they will guide you slowly through the steps. STOP READING HERE.

4.2. For the rest of us, we'll go through a few simple steps to achieve this. Let's start with the basics of reading the proximity sensors. There are 3 IR sensors mounted on the front sensor bar. They are the black plastic parts you can see when you flip the robot over. There are 2 big IR emitters on the front – they look like the ones you have on IR remote controls. There is also an IR emitter either side of the main board in between the wheels. So that robot is well equipped with sensors. See here for more details: https://www.pololu.com/docs/0J63/3.6

4.3. To read from these sensors, the following is a modified version of the LineAndProximitySensors example. Create a new sketch and paste this code.

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

Zumo32U4ProximitySensors proxSensors;

//uint16_t lineSensorValues[5] = { 0, 0, 0, 0, 0 };

bool proxLeftActive;
bool proxFrontActive;
bool proxRightActive;

void setup()
{
  /* Configuration:
   * - 3 proximity sensors (left, front, right)
   *
   */
  proxSensors.initThreeSensors();

}
```

```
// Prints a line with all the sensor readings to the serial
// monitor.
void printReadingsToSerial()
{
  static char buffer[80];
  sprintf(buffer, "%d %d %d %d\n",
    proxSensors.countsLeftWithLeftLeds(),
    proxSensors.countsFrontWithLeftLeds(),
    proxSensors.countsFrontWithRightLeds(),
    proxSensors.countsRightWithRightLeds()
  );
  Serial.print(buffer);
}

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();

    printReadingsToSerial();
  }
}
```

4.4. Connect your robot, Click upload. Click the button for the serial monitor. You should be able to alter ranges of values presented on the screen by moving the robot or your hand around. Try to understand the above code. Close the serial monitor when you're done.

4.5. Next, we want to pass this data into ROS. Create a new sketch and use the following. Upload it to your robot. Have a read through it and try to understand what is going on.

```
#define USE_USBCON

#include <ros.h>
#include <std_msgs/Int8.h>

ros::NodeHandle nh;

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

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

Zumo32U4ProximitySensors proxSensors;

bool proxLeftActive;
bool proxFrontActive;
bool proxRightActive;

void setup()
{
  /* Configuration:
   * - 3 proximity sensors (left, front, right)
   *
   */
  proxSensors.initThreeSensors();

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

}

// Prints a line with all the sensor readings to the serial
// monitor.
void printReadingsToSerial()
{
  msg.data = proxSensors.countsLeftWithLeftLeds();
  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();

    printReadingsToSerial();
  }
}
```

4.6. To get our program working, do the following.

- Make sure you have roscore running

```
$ roscore
```

- Run the rosserial ROS node

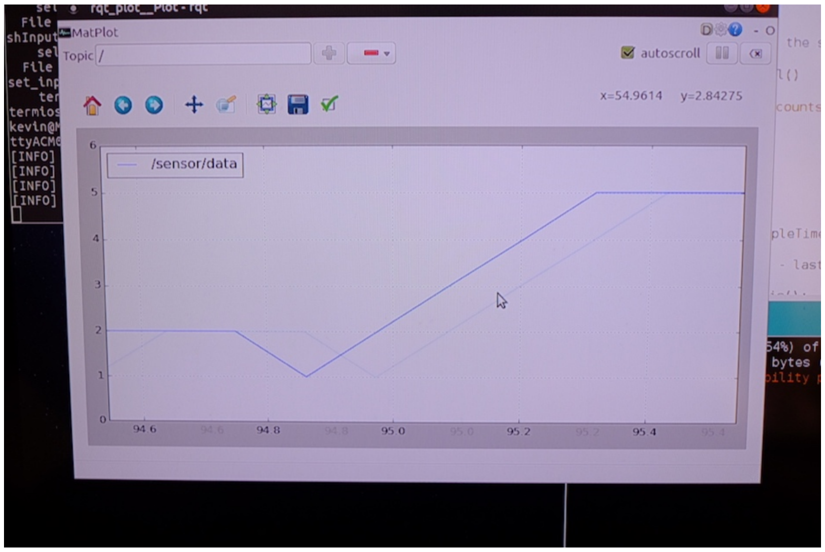
```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

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

```
$ rostopic echo sensor
```

- Run rqt_plot and type sensor to see the data graphed.

```
$ rosrun rqt_plot rqt_plot
```



- Adjust the delay in the Arduino program to change the pace the data is sent. You'll have to restart the ROS rosserial node as it will crash during reprogramming of the Arduino.

- This is pretty cool. We can now reliably use the sensors on the robot in ROS. In the next task we'll move on to writing our own ROS programs and manipulating the robot motors.

5. Additional resources

    5.1. Learn about linux

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

    5.2. 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