Sensors and Sensing
Lab 2: ROS Arduino and Differential Drive

Todor Stoyanov

September 26, 2018

1 Objectives and Lab Materials

The objective of this lab is to introduce you to using the ROS serialization package for arduino boards and to use that to implement a differential drive robot interface. You will use the interface both for controlling the robot and for obtaining odometry measurements. You will also calibrate the odometry of your robots.

For this lab you are given:

- MakeBlock kit assembly, including a MegaPi board and two DC motors. (we refer to the MegaPi board as **arduino** from now on).
- Raspberry Pi with Ubuntu and the Arduino IDE pre-installed. (we refer to the Raspberry Pi as **pi** from now on).
- A desktop PC with ROS kinetic installed (we refer to this computer as **desktop** from now on).

The following tasks will guide you through setting up ROS serial communication for the arduino and a corresponding C++ node on the pi. We will also setup ROS remote to allow us to communicate with the pi from the dektop. In your report, describe the steps you have taken to solve each task, as well as the results you obtained and the methods you used to verify that the system functions properly. Explain what experiments you performed, how and why. Provide your Arduino sketch and C++ source codes as attachments and demonstrate the final system to the lab assistant. If you think it is necessary, you may take pictures of the system in action, and/or include screenshots.

1.1 Preliminaries

Before starting with this lab please verify you have performed these preliminary steps:

- For this lab, mount the tracks of the robot back on. For the time being, keep the robot onto a stand, so that the tracks do not touch the ground.
- Verify that the arduino is connected via USB to the pi.
- Connect an Ethernet cable between the desktop and the pi.
- Connect the battery pack.
- Using the network manager on your workstation, set up a static IP for the wired connection onto the subnet 192.168.100.255.
- On the pi: Clone the github code for this lab into your workspace.
  `git clone https://github.com/tstoyanov/sensors_lab2_2018.git`
- On the pi: Copy the file `sensors_lab2_2018/src/sensors_lab2_2018_stub.ino` into your arduino sketch directory.

2 Task 1: ROS remote setup (5 points)

In this task we will setup a remote ROS system to simplify testing from the desktop. First you need to edit the `/etc/hosts` file on both the pi and the desktop, to include these lines:

```
192.168.100.2 makeblock-pi
192.168.100.XXX desktop
```

where XXX is the correct IP of your desktop.

Next, you need to make sure both computers are accessible by ssh from each other. To do that, you need to generate ssh keys on both computers.

```
desktop$: ssh-keygen
desktop$: ssh pi@makeblock-pi
pi$: ssh-keygen
pi$: exit
desktop$: cd ~/.ssh
desktop$: scp id_rsa.pub pi@makeblock-pi:~/.ssh/authorized_keys
desktop$: scp pi@makeblock-pi:~/.ssh/id_rsa.pub  ~/.ssh/autorized_keys
```

Test that you can now ssh in between the two computers without having to enter your passwords. Next, we will set up the ROS remote variables. In your `.bashrc` file add the following lines:

```
export ROS_MASTER_URI=http://makeblock-pi:11311
export ROS_IP=makeblock-pi
export ROS_HOSTNAME=makeblock-pi
```

Save and open new terminals (or source the `.bashrc` file). ssh into the pi and start a roscore. In a separate terminal, check that `rostopic list` from the desktop returns a list of topics. Use the rostopic tool to test communication with fixed topics.

3 Task 2: ROS serial communication setup (5 points)

We will now set-up ros serial communication with the arduino. On the pi:

```
pi$: source ~/workspace/devel/setup.bash
pi$: cd ~/Arduino/libraries
pi$: rosrun rosserial_arduino make_libraries.py .
```

Now start the arduino IDE and open the `ros_lib/ServiceServer` example. Compile and upload the sketch to the arduino. We now need to install rosserial-python on the pi. To do that, follow these steps:

```
desktop$: wget http://packages.ros.org/ros/ubuntu/pool/main/r/
  ros-kinetic-rosserial-python/ros-kinetic-rosserial-python_0.7.7
  -0xenial-20180826-025520-0800_armhf.deb rosserial_python.deb
desktop$: scp rosserial_python.deb pi@makeblock-pi:~
desktop$: ssh pi@makeblock-pi
pi$: sudo dpkg -i rosserial_python.deb
```

You can now test the serial communication by starting the serial node on your pi:

```
pi$: roscore
pi$: rosrun rosserial_python serial_node.py /dev/ttyUSB0
pi$: rostopic echo /chatter
```

Check that the chatter topic is also visible from your desktop PC. Once done with this setup, you can now start with the core of the lab.

4 Task 3: Implementing a Differential Drive (10 points)

As a first step, we will port the code from Lab1 to use ros_serial communication. Open the newly provided sketch file `sensors_lab2_2018_stub.ino` and take some time to familiarize yourself with the contents. You will notice that the main functionalities: the pid controller and encoder counters can be directly copied from the previous lab. What you have to do, in order to replicate your previous results, is to implement the new (and much simpler to write) communication functions. To do that:

- In the loop function, fill in the `state` varriable with data from the left and right motor encoder. Set a sequence number and time of message generation.
- Implement the SetPID callback function which changes the PID values for a specific motor.
- Test with a hardcoded velocity target, or alternatively, (for +2 bonus points) define and implement a `setVelocity` service. In either case, make sure that you can read motor states from ROS and that you can re-set PID parameters.

Now that you can move the motors using the new interface, it is time to implement a differential drive controller. To do that:

- Implement a service to set the relevant parameters for a differential drive kinematic.
- Implement a service to re-set the odometry to zero.
- Subscribe to a topic `\cmd_vel` which accepts a `geometry_msgs/Twist` message and sets appropriate speeds to the left and right wheels. Use the PID controllers to track the desired wheel velocities. You will need to solve for the inverse kinematics problem here: i.e., how to convert from a linear velocity and angular velocity to desired velocitites of the left and right wheels.
- Make sure that if no velocity command arrives for a set interval of time the motors are set to zero velocity (for safety).
- Implement a publisher which sends `nav_msgs/Odometry` message for every feedback cycle. Use the measured elapsed time and velocity of each wheel to integrate measurements and obtain a relative x, y, θ with respect to the previous pose. Use homogeneous transformations to add the new relative measurement to the pose reported in the previous cycle. In order to publish an Odometry message, you will need to convert from a 2D rotation matrix to a quaternion, which is a different representation for orientations. A quaternion for rotation of angle α around a normalized vector $v = [v_x, v_y, v_z]^T$ is represented as

  $$q = \begin{bmatrix} v_x sin(\alpha/2) \\ v_y sin(\alpha/2) \\ v_z sin(\alpha/2) \\ cos(\alpha/2) \end{bmatrix}$$

  For this lab, you only need to fill in the position and orientation parts of the odometry message, and you can ignore the covariance and twist.
- Use a ruler to measure the parameters of your odometry model and set them to the arduino. Test your odometry module.

5 Task 4: Odometry Calibration (5 points + 5 bonus points)

For this task, you should perform a calibration of the odometry module you have just devised. This is an open task: it is up to you to design a calibration procedure and then carry it out. You may want to use a measuring tape to obtain ground truth measurements. Use a package to command your robot from the keyboard, for example `http://wiki.ros.org/teleop_twist_keyboard` (which should be installed on the desktop). There are two options:

- To obtain the full 10 points: In order to optimize the relevant kinematic parameters, you will need to record the wheel velocities directly (use the motor state topic) and then perform a non-linear optimization to find the best fitting parameters. You can use octave to perform the optimization, and use the functions `fminsearch` or `fminunc` to find a solution. Try different starting points for your minimizer and see how the solution converges.

- For five points: If this task is too difficult, you can devise a more ad-hoc method to calibrate parameters separately. For example, you can try driving a straight line of known distance a number of times and re-scale the wheel radii accordingly. You can then try to devise a method to estimate the remaining parameter.