Real Turtlebot Lab

This section should be followed only if you are working on a Turtlebot with a Turtlebot laptop.

Please keep your laptops in the lockers and please plug them in so that they get charged.

You can login to laptops using

Username: robot

Password: robot

You can connect to the internet through eduroam.

Since you will be working on this laptop with your group's members, please configure the wireless eduroam settings such that it does not save your password. You can do this by clicking on the WiFi symbol at the top right of the Ubuntu menu, and choosing "Edit connections…". Then, in the new window, select "eduroam" and click Edit. Then choose "Wi-Fi Security" and check the box saying "Ask for this password every time". Also go to "IPv6 Settings" and change the Method to "Ignore". Then Save and close the window. You can ignore any messages about certificates.

TurtleBot introduction

You should be reading this while in front of the TurtleBot.

1. Cables:

On the TurtleBot base there will be two USB cables you should see. They need to be plugged into the TurtleBot's laptop to be operational.

2. Charging:

A seperate cable and powerpack are used to charge the TurtleBot, they will be located near the TurtleBots. When you are finished with the TurtleBot, plug the robots charger into the base (near the power switch for the base). This will make sure that the robot is charged for the next group to use it.

Similarly, when you are done, please place the laptop into one of the lockers and plug it in so that it can be charged.

3. Laptop and seating the laptop on the TurtleBot:

The notebooks are seated on top of the TurtleBot. They should be seated so the the laptop screen is facing the opposite direction to the 3d sensor is facing. They are held in place by the velcro squares on top of the robot and the one on the base of the laptop, make sure that you line up the squares as best as possible so the laptop is seated securely on the top.

4. Turning the robot on:

Use the power switch on the mobile base to turn it on. Then, turn the laptop on.

Setup Turtlebot laptop for your group

The Turtlebot laptops do not keep your files on disk. Therefore, everytime you log in, you will need to retrieve any files you want to work with from your group's git repo. To configure the laptop for a specific group, simply open a terminal window, and run the command below with your group's name:

```
setup_group <your-group-name>
```

For example:

```
setup_group Group1
```

This script does something simple: it clones your group's git repo under $HOME/catkin_ws on the Turtlebot laptop. You can see the directories and files under $HOME/catkin_ws/src .

Note that you will need to call this setup script **every time** you logon to the Turtlebot laptop.

!!! IMPORTANT !!!: Turtlebot laptops erase all changes and revert to a clean state once you log off. Therefore, if you edit/create any files on the Turtlebot laptop, and if you do not want to lose your changes/files, then push them to your group's git repo before logging off. We recommend pushing your changes to git frequently, in case the laptop accidentally turns off, e.g. in case the battery is drained, since you will lose your changes on the laptop if they are not pushed to git.

For example, if you want to push your changes as you work on realrobotlab:

```
cd $HOME/catkin_ws/src/realrobotlab/src/

git add <list-of-files-you-want-to-push-to-git>

git commit -m "your message"

git push
```

Running TurtleBot software

To get the laptop to command the robot, you need to start the TurtleBot software. In a terminal window, run:

```
roslaunch turtlebot_bringup minimal.launch
```

This will bring up the nodes that provide the basic functionality of the TurtleBot. For example, this will bring up the controller for the TurtleBot's mobile base, drivers for TurtleBot's sensors, etc. You can inspect the nodes and topics launched. If you want to see a list of all the ROS nodes that are running, in a new terminal do this:

```
rosnode list
```

and inspect the result.

Similarly, to see all the ROS topics that are created, do this:

```
rostopic list
```

These are all the topics required for the complete functioning of the TurtleBot.

Moving the robot

You will find the file firstwalk.py in realrobotlab directory. Open and inspect it. Particularly notice that since we need access to the geometry_msgs/Twist message type, the script imports it like this:

```
from geometry_msgs.msg import Twist
```

You always need to import a message type if you are going to use it in a script.

Notice how the script is publishing to the /mobile_base/commands/velocity topic.

Also notice how we set the x component of the linear velocity to a desired value.

Run this script and see what happens (you might need to make the script executable before you run it - chmod +x file):

```
rosrun realrobotlab firstwalk.py
```

Stop the script (use ctrl-c) and you can edit the velocity values to change the speed the robot is moving.

Please never command the robot to drive faster than 0.2 m/s, otherwise you can hurt the robot or some other item in the room, when you are running the real robot.

Bumper input

When you start up the TurtleBot software it does not only start the controller for the mobile base, but it also starts drivers for sensors. If you look at the output of 'rostopic list' again, you can see different sensor topics. Here we will mention only one, but you can experiment with others as well. The topic "/mobile_base/events/bumper" is a topic that outputs values based on the bumpers on the TurtleBot base. If you have the real Turtlebot in front of you, please locate the two bumpers on the left and right of the robot. Use rostopic info to inspect this topic.

```
rostopic info /mobile_base/events/bumper
```

You will see that the messages are of type kobuki_msgs/BumperEvent. You should use rosmsg show to inspect the details of the BumperEvent message type.

One quick way to make sure this topic works as expected is to listen to it on command line using "rostopic echo":

```
rostopic echo /mobile_base/events/bumper
```

(Ignore any warnings you might get about “/clock”)

Now, press on the bumpers with your hand, while you keep your eye on this terminal window. You should see that the state variable is set to 1, when bumper is pressed.

Retrieving Images from Robot Camera

There are two cameras on the TurtleBot. One camera is the 3D sensor's camera. The 3D sensor provides both RGB and depth values which are published as separate ROS topics. The other camera is the laptop's camera, which you are also welcome to use. The laptop camera exists only on the real Turtlebot, not in simulation.

3D sensor – Real Robot

Since the 3D sensor consumes lots of battery power, the minimal.launch does not start it. To start the 3D sensor, you need to run a new launch file:

```
roslaunch astra_launch astra.launch
```

Now look at the list of topics again ("rostopic list"). You should see topic names which did not exist before. There should be plenty of /camera/ topics including /rgb/ and /depth/. For example, /camera/rgb/image_raw is the topic for the raw RGB image from the camera. Just like in simulation, run:

```
rosrun image_view image_view image:=/camera/rgb/image_raw
```

You should see the RGB image from the robot’s 3D sensor in a new window.

The 3D sensor also outputs depth (distance) values. The topic "/camera/depth_registered/points" include the depth data. Just like we did in the simulated environment, we will now use rviz to visualise depth data by running the following command:

```
rosrun rviz rviz
```

As described in Section 4 of the rviz user guide, add a new display for the PointCloud2 message type. On the left pane, enter the topic name for this display as /camera/depth_registered/points. Change the value of the "Fixed Frame" field under "Global Options", and select "odom". You should now see the 3D point cloud from the 3D sensor. You can use your mouse buttons to move around in the 3D display. After you are done, quit rviz.

Laptop camera

You should be able to launch the laptop camera using this command:

```
roslaunch libuvc_camera launchcam.launch
```

You should now see the topic /laptop_camera/image_raw being published. Use image_view to inspect the image.

Convert from ROS to OpenCV and display image

Now let’s focus on getting your python node capable of reading the camera data and then getting it to display to a screen. Open the file convert_image.py.

OpenCV

To process images and make decisions based on them we will be using a library called OpenCV (Computer Vision). Specifically, we will be using OpenCV2 so in any python scripts that you want to handle images you need to import cv2:

```
import cv2
```

Since we are handling ROS we also need the use of a library called cv_bridge, which can translate ROS images into images readable by OpenCV. (More info abut cv_bridge is here: http://wiki.ros.org/cv_bridge/Tutorials)

Importing necessary modules

We always start by importing the necessary python modules and classes.

```
import cv2

import numpy as np

import rospy

from sensor_msgs.msg import Image

from cv_bridge import CvBridge, CvBridgeError
```

Subscribing to the image topic

In order to receive and process image data from the cameras we must create a subscriber to the topic that our camera outputs to. The RGB image from the 3D sensor is on the topic: camera/rgb/image_raw. Create a subscriber for this topic in the constructor (__init__) of the colourIdentifier class, and specify the callback function of the colourIdentifier class as the callback of the image topic.

Converting between openCV and ROS image formats

The image from the camera arrives in the ROS type Image. To manipulate it in OpenCV, we must convert the image from the ROS format of Image into an OpenCV image. Luckily OpenCV has buit in functions to do this for us. Call the function imgmsg_to_cv2(data, "bgr8") on a CvBridge object you have created in your class. It's always a wise idea to wrap calls to this method in an exception handler in case anything is wrong with the camera feed.

The only thing left to do is to output the camera feed to the screen.

Displaying image on a new window

We declare that we want to have a named window called 'camera feed' then we show it.

```
cv2.namedWindow('Camera_Feed')

cv2.imshow('Camera_Feed', <image name>)
```

Try to run your script and see if it works.

Appendix

Earlier in this lab you will have noticed that we were instructing you to examine and analyse topics and the messages they use via the rostopic echo and rosmsg show commands. This is a very helpful method for understanding how the different topics communicate with each other over ROS. When writing publishers and subscribers you will definitely want to use this method of examination to find out about the messages you need to be handling or sending in your custom nodes. A follow on point to that method was that you can also use rostopic echo to examine messages arriving at the topic, allowing you see the kind of messages that arrive if you do particular things to the robot.

rostopic info <topic_name>

rostopic echo <topic_name>

rosmsg show <message_type>

Remember these commands, they will become **very useful** for your future work within ROS. You can also call "rostopic --help" to see other useful functionality within the rostopic command. Same applies to rosmsg and rosnode commands.

Below, we collect the topics on the TurtleBot that are worth knowing about and we encourage you to examine them using this method and anything else we mention:

/camera/depth_registered/points - This is a depth pointcloud produced by the 3dsensor that you can view in rviz visualiser

rosrun rviz rviz

This will be a useful topic to subscribe to for when you want to program nodes that react to objects based on depth and how far away or close an object may be (following, for example).

/camera/rgb/image_raw - The raw image data being input from the sensor, you can view this using a package called image_view

rosrun image_view image_view image:=/camera/rgb/image_raw

This can be useful for identifying the colour of specific objects in the sensors field of view.

/laptop_charge - Keeps track of the battery levels in the turtlebots laptop, you can inspect this topic using the rostopic info and rosmsg show method you used earlier.

rostopic info /laptop_charge

rostopic echo /laptop_charge

You could write a node to monitor this level and give warnings to the user when the battery levels are running low.

/mobile_base/events/bumper - Records events concerning the bumpers on the robots base, you can inspect this topic using the rostopic info and rosmsg show method you used earlier.

rostopic info /mobile_base/events/bumper rosmsg show BumperEvent

rostopic echo /mobile_base/events/bumper

/mobile_base/events/cliff - Records events concerning when the robot can or cannot detect a floor beneath itself, you can inspect this topic using the rostopic info and rosmsg show method you used earlier.

rostopic info /mobile_base/events/cliff

rosmsg show CliffEvent

rostopic echo /mobile_base/events/cliff

Try running rosotpic echo and examine the messages produced when you pick up and put down the robot.

/mobile_base/events/robot_state - Indicates the state of the robot, on or off, you can inspect this topic using the rostopic info and rosmsg show method you used earlier.

rostopic info /mobile_base/events/robot_state

rosmsg show RobotStateEvent

rostopic echo /mobile_base/events/robot_state

Run rostopic echo on this topic, it should show you one message, you can guess what it means.

/mobile_base/events/wheel_drop - Records events concerning whether or not the wheels have dropped within their seating either due to the floor becoming angular or due to the robot being picked up.

You can inspect this topic using the rostopic info and rosmsg show method you used earlier.

rostopic info /mobile_base/events/wheel_drop

rosmsg show WheelDropEvent

rostopic echo /mobile_base/events/wheel_drop

Run rostopic echo on this topic and try picking the robot up and placing it down again. You'll notice that two seperate messages are produced when you do this, one for each wheel.

Try and figure out which wheel is wheel 0 and which wheel is wheel 1.

/mobile_base/sensors/imu_data - The imu collects data concerning linear acceleration and angular velocity and sends it to the main processor, you can inspect this topic using the rostopic info and rosmsg show method you used earlier.

```
rostopic info /mobile_base/sensors/imu_data
```

```
rosmsg show
```

```
rostopic echo /mobile_base/sensors/imu_data
```

Run rostopic echo on this topic and try nudging the robot or turning it on the spot. You'll notice how sensitive the robot is to changes in its own velocity.