**Module M02\_RCDP - Remote Control & Data Processing**

*Supervisor: Shih-Yuan Liu & Andrea Censi*

*Deadline: Sunday Feb 28, 2016, 11:59pm*

Feedback and discussion about this module belong to the Slack channel **#m02-RCDP**, or **comment** directly here. Please **do not** edit directly (even if you can), because sudden changes might be very confusing for other students. If you find inconsistencies, please use the comment function. Instructors: please clearly mark the changes in a different color.

List of exercises:

[RCDP0: Setup the repository to work on the lab branch](#_iptnan6ssm5r)

[RCDP1: Remote control](#_hcwepq2xeejj)

[RCDP2: Logging](#_npt01yfkknmt)

[RCDP3: ROS Basics](#_62ytcu45e1p6)

[RCDP4: Basic Data Processing](#_b1hfyesapsxj)

[RCDP5: Processing data from a log.](#_rx95e5egssij)

[RCDP6: Stateful data processing](#_kcksf0se0542)

TA grading guide: 100/7 points for each exercise.

### RCDP0: Setup the repository to work on the lab branch

Background: please read the chapters in the Git book mentioned in [Pointers to Reference Materials for Git, ROS, etc.](http://drive.google.com/open?id=1eEz4DN_-mXzC3Cu4Wapt1f4rkKkBK4kKcPRvfTfMxDo).

For this module, please work on the branch M02\_RCDP:

duckiebot $ git pull

duckiebot $ git checkout M02\_RCDP

duckiebot $ catkin\_make

Please do not commit to M02\_RCDP.

(Advanced students: please submit pull requests to M02\_RCDP.)

Create your own branch from M02\_RCDP:

duckiebot $ git branch M02\_RCDP-<handle>

duckiebot $ git checkout M02\_RCDP-<handle>

Feel free to *commit* to this branch.

Please do not *push* the branch to the remote repository just yet. (We will explain in the next lectures what you need to know about Git to move your work to a private fork.)

Do you now know about the difference between “commit” and “push”? Time to read the Git book mentioned in [Pointers to Reference Materials for Git, ROS, etc.](http://drive.google.com/open?id=1eEz4DN_-mXzC3Cu4Wapt1f4rkKkBK4kKcPRvfTfMxDo) We guarantee that you will have to learn Git properly -- better do it now and save yourself lots of trouble later.

This is the expected output:

**duckiebot** $ git status

On branch M02\_RCDP-<handle>

nothing to commit, working directory clean

**duckiebot** $ ls ~/duckietown -t

**YES-THIS-IS-THE-RIGHT-ONE** RPi2-Ubuntu circuits

LICENSE.md bootstrap.sh duckietown\_install\_car.sh ros\_diagram setup

README.md catkin\_ws duckietown\_install\_laptop.sh scuderia.yaml

TA grading guide: assume this was done correctly.

### RCDP1**:** Remote control

*Learning objective: Basic Duckiebot operation; video documentation; access to Dropbox.*

1. Create the folder in Dropbox:  
     
    Dropbox:duckietown-data/logs/20160210-M02\_DPRC/<handle>/
2. Setup the Duckiebot in RC mode, using the instructions in [Setup Step 2.1 Joystick + camera output in remote laptop](http://drive.google.com/open?id=1FB25mF8703TtEBUfNR6s8NXYc8_22lCU_7gAfQ_Rw_Y).
3. Make sure your Duckiebot is in the CCC (current conforming configuration).
   * Reminder: the CCC includes the cuteness constraint (Duckie on top).
4. Drive your Duckiebot and take a video of it using an external camera.
   * The Duckiebot should do a nontrivial trajectory that excites all degrees of freedom.
     + The trajectory corresponding to (velocities v = 0, omega) = 0 is trivial
   * Length: 10-15s are enough.
   * Cell phones are great cameras.
   * Our Creative Director Chris Welch hates vertical video. Therefore, vertical video is not valid.
   * Given the dynamic constraints, it is impossible for a Duckiebot in CCC to do wheelies; videos with wheelies are therefore evidence of the duckiebot being not in the CCC.
   * You might want to use a friend to drive the robot or to take the video, unless you have a tripod, three arms, or another robot that you programmed to hold the camera while it takes a video of the Duckiebot.
5. Put the video in *mp4* or *mov* format on Dropbox, with file name:

../20160210-M02\_DPRC/<handle>/201602DD-<handle>-<robot>-RCDP1-external.[mp4|mov]

TA guide: Check that the file above exists in the Dropbox, that it is named appropriately, and that it contains what it should contain.

### 

### RCDP2: Logging

*Learning objective: Take a log (called “bag”) with your Duckiebot.*

To learn how to record a bag, refer to the tutorial in [160219 - Lab 03 - ROS tutorial](http://drive.google.com/open?id=1fJe0IZXczCmrez8LEv3s5BxbIO0owAwpc3gZul1hhec).

Instructions:

1. Start RC mode on your Duckiebot (with camera output).
2. Start a log with rosbag and subscribe to all channels (use the -a option).
3. Drive around for at least 1 minute of continuous motion.
4. Open the log using rqt\_bag to check that the data got recorded correctly.
5. Put the bag in Dropbox, and name the file (DD = day of month):  
     
   ../20160210-M02\_DPRC/<handle>/201602DD-<handle>-<robot>-RCDP2.bag

TA guide: Check that the file above exists, that it is named appropriately, and that it contains what it should contain using rqt\_bag.

### 

### 

### RCDP3: ROS Basics

*Learning objective: creating ROS modules.*

**If you were in Lab 03** (Beaverworks Feb 19) **and** **you completed up to Step 2.3**: you are done - please move on to the next exercise.

1. node

If you have not already completed up to Step 2.3, you should replicate the lab **using your robot and your laptop** instead of the ones used in the lab, which were called “Wolverine” and “megaman”, respectively.

These are the differences between the lab setup and your individual setup:

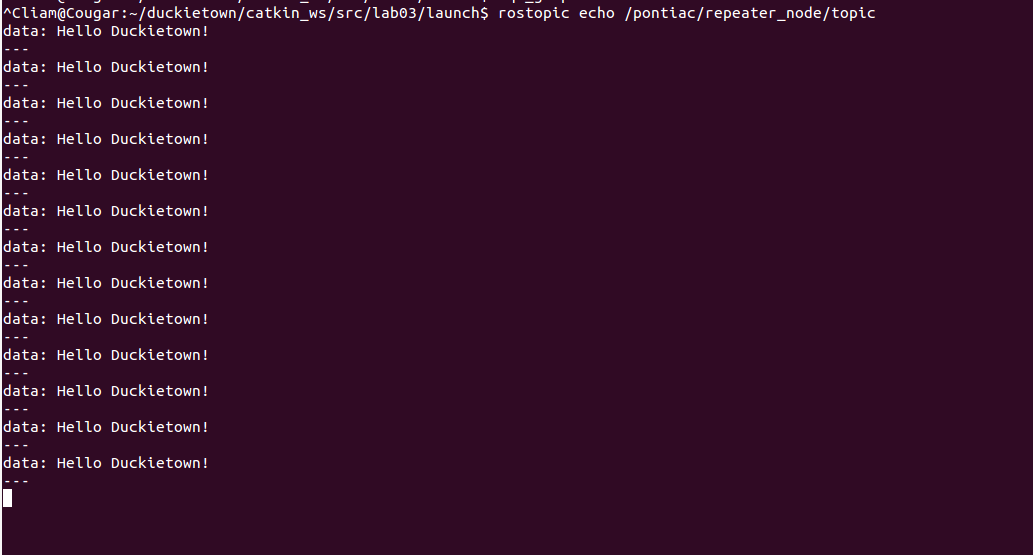
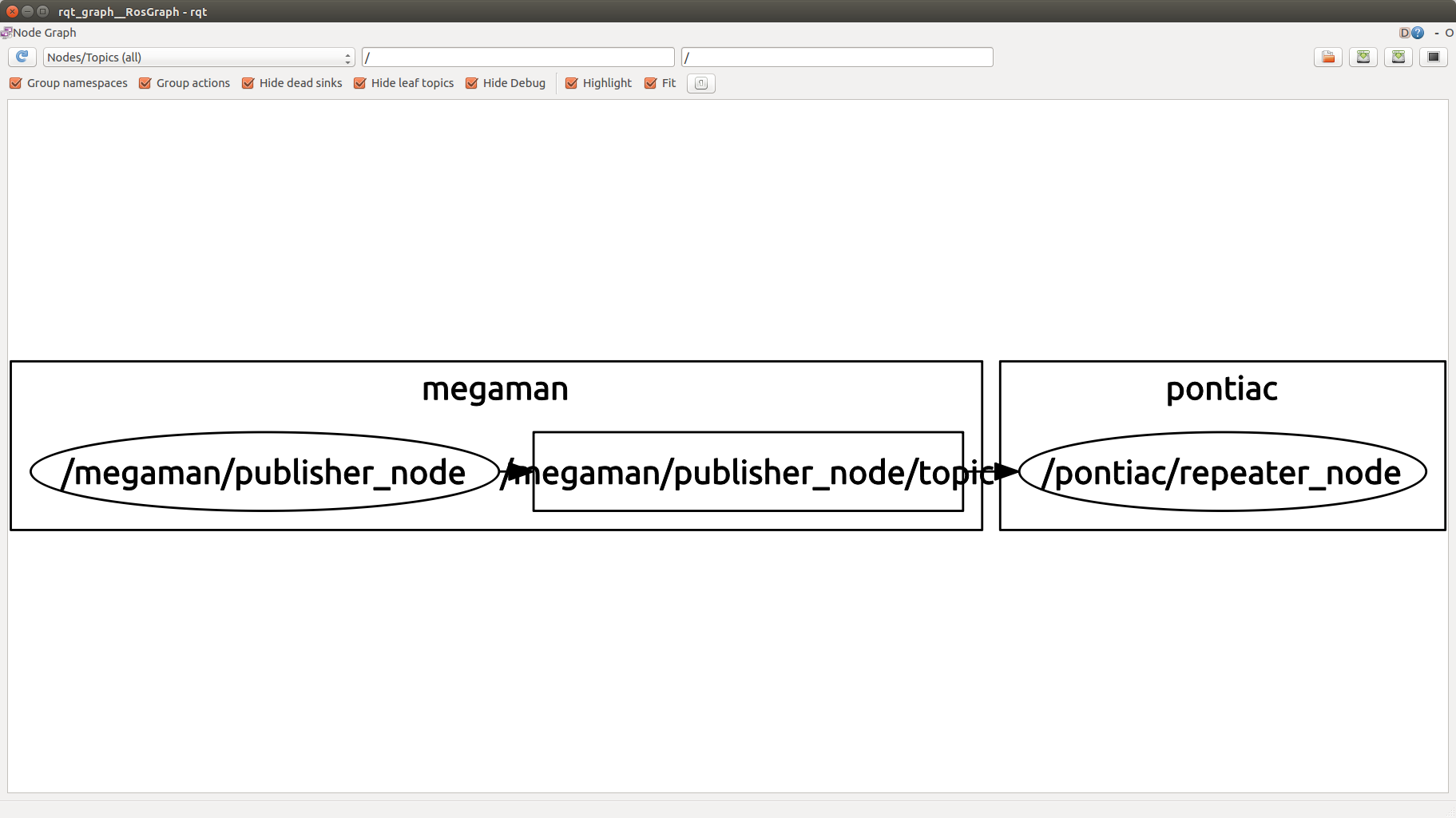
1. Your laptop should be “publisher”.
2. Your robot should be the “repeater” / “subscriber”.
3. Set the ROS\_MASTER\_URI to be your laptop.
4. Complete the steps in [160219 - Lab 03 - ROS tutorial](http://drive.google.com/open?id=1fJe0IZXczCmrez8LEv3s5BxbIO0owAwpc3gZul1hhec).
5. Add ‘1’s to the columns in [160219 - Lab 03 - progress status](http://drive.google.com/open?id=1AxmsAb0gi-LIB869gqitdF-Os280eUiuqh5oC55Xv_s).
6. In Exercise 2.3, run the publisher on your laptop with **veh:=megaman**
7. Run the repeater on your robot with **veh:=your\_robot\_name**

Take a screenshot of   
1) the graph created by rqt\_graph; and

2) the output of running this on your laptop:

**laptop** $ rostopic echo /your\_robot\_name/repeater\_node/topic

The screenshots should look something like this:



Post both of these to the Slack channel #lab03-feb19.

TA guide: Check that either:

1. they were present at the lab and completed the exercise; or
2. the two screenshots were posted to the Slack channel.

### RCDP4: Basic Data Processing

*Learning objective: creating a node that can receive and republish data.*

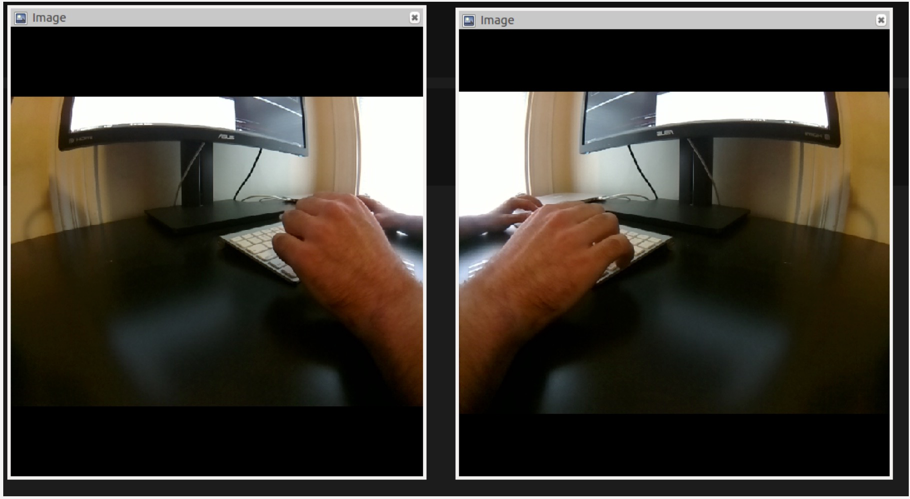
The simplest example of a data processing node is a node that receives an image, makes a little modification to it, and then republishes it. Let us call this node “virtual\_mirror-<handle>”.

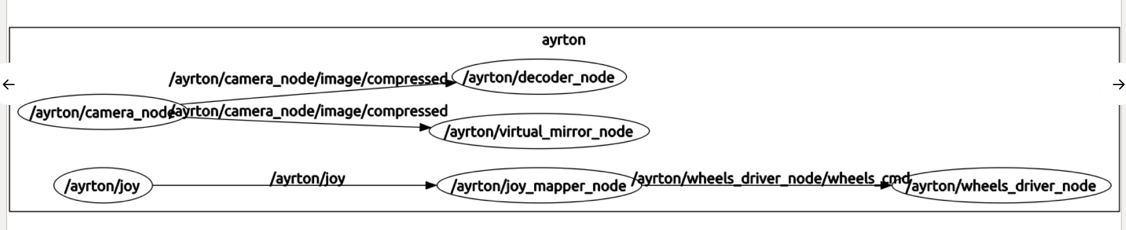
1. Create virtual\_mirror-<handle> according to the conventions in the ROS tutorial.   
   The package should be in:  
    duckietown/catkin\_ws/src/virtual\_mirror-<handle>
2. This is the specification of the functionality.  
     
   Let H,W be the height and width of an image. The array has thus size H x W x 3.   
     
   Let rgb\_in[u1,v1,w1] be the image received by the module (pixel y=u1, x=v1, channel w1) and let rgb\_out[u2,v2,w2] be the image published by the module. (pixel y=u1, x=v1, channel w1).  
   vi  
   Then this should be true for all u,v in the range of the image:  
     
    rgb\_out[**u**, **v**, **w**] == rgb\_in[**u**, **W** - **v**, **w**]  
     
   (Note: **W** ≠ **w**) This corresponds to flipping the image along the horizontal axis (virtual mirror).
3. Create a launch file that makes the node subscribe to the Duckiebot image. Please refer to the ROS Diagram in the Duckietown-public:design/ folder for the name of the topic.  
     
   The launch file is in   
    duckietown/catkin\_ws/src/virtual\_mirror-<handle>/launch/virtual\_mirror\_node.launch  
     
   (Note: You will have to run the camera launch file separately) The node should be launchable using:  
   $ roslaunch virtual\_mirror-<handle> virtual\_mirror\_node.launch veh:=<duckiebot>v
4. Run this node and subscribe to both images from RViz in your laptop.
5. Take a screenshots and post to #m02-RCDP:
   1. the rqt\_graph
   2. the two images in RViz

TA grading guide: Check that the two screenshots were posted to the Slack channel.

Food for thought: Why do mirrors reverse left and right, but not top and bottom?

Expected result (from Takke):





### 

### 

### RCDP5: Processing data from a log.

Learning objective: ability to process data from a log.

In exercise RCDP2 you created a bag file, by the name:

[...]/20160210-M02\_DPRC/<handle>/201602DD-<handle>-<robot>-RCDP2.bag

Call this “log\_in”. In bash (it needs to be the full path):

log\_in=[...]/20160210-M02\_DPRC/<handle>/201602DD-<handle>-<robot>-RCDP2.bag

In this exercise we will create another bag file, which we are going to call “...**-RCDP5-log\_out**.bag”:

log\_out==[...]/20160210-M02\_DPRC/<handle>/201602DD-<handle>-<robot>**-RCDP5-log\_out**.bag

Create a launch file virtual\_mirror\_test.launch.  
  
This launch file should use two variables, log\_in and log\_out, such that when we run   
  
$ roslaunch virtual\_mirror-<handle> virtual\_mirror\_test.launch veh:=<duckiebot> log\_in:=$log\_in log\_out:=$log\_out

this is what happens:

1) The recorded image data in $log\_in is pushed through virtual\_mirror  
2) All the topics are saved in $log\_out.

*Hint: have a look in* ~/duckietown/catkin\_ws/src/duckietown\_unit\_test/launch *for some launch files that run rosbag in this way.*

Consequently, log\_out contains two image streams: the initial one and the flipped one.

Deliverable: the resulting file log\_out in 20160210-M02\_DPRC/<handle>/201602DD-<handle>-<robot>**-RCDP5-log\_out**.bag

**Update (Feb 24): You should know that ``rqt\_bag`` has a bug for which the colors are not decoded right. If you use ``rqt\_bag`` in our logs, the logs look blueish, but in fact, the image inside the log is correct.**

TA grading guide: Check that the file exists and that it contains the two streams of images.

### 

### RCDP6: Stateful data processing

*Learning objective: learn how to create a module that does data processing in a stateful way (i.e. with an internal state).*

Create a package “image\_average-<handle>” according the the conventions. This package reads an image rgb\_in and cpwrites an image rgb\_out.

The image rgb\_out in this case is the temporal average of rgb\_in. Formally:

rgb[u,w,v] at time t1 == average of rgb\_in[u,w,v] between t0 and t1

Create a launch file that takes log\_in and log\_out as parameters.

Run this on the bag file provided in

20160210-M02\_RCDP/censi/20160122-censi-ferrari-RCDP2.bag

Put the output bag file in

20160210-M02\_DPRC/<handle>/20160122-censi-ferrari-RCDP6-<handle>.bag

Open the bag using rqt\_bag and take a screenshot of the last image; post it on Slack in #m02-rcdp.

TA grading guide: Check that the file above exists and that it contains the two streams of images; look for the screenshot in Slack.