# CSCE574 – Robotics
# Spring 2014 – Notes on Turtlebot robots

## 1 About the Turtlebot

Through the remainder of this class, we'll use ROS to work with a robot platform called the Turtlebot. The next few sections describe how to obtain your Turtlebot kit, how to set up these robots, and how to operate them.

The Turtlebot built from several off-the-shelf components:

1. An iRobot Create mobile base.
2. A small laptop.
3. A Kinect sensor that collects Red-Green-Blue-Depth (RGBD) images.
4. A custom circuit board with both single-axis gyroscope and a power supply for the Kinect.

For this course, we are using the original Turtlebot, which is also called "Turtlebot 1" in the documentation. The newer Turtlebot 2 version is very similar, but has a redesigned mobile base.

### 1.1 Turtlebot is not turtlesim

Note that the first project and many of the tutorials used a very simple robot "simulator" called `turtlesim`, which is designed to help people learn about ROS without worrying too much about the details of real robots. We will not use `turtlesim` again this semester. The fact that both of these names contain the word "turtle" is an unfortunate coincidence.

### 1.2 Turtlebot has two computers

As you are working with the Turtlebot, there are actually two computers you'll use.

- A **netbook** computer that rides atop the robot. Throughout these notes, I'll refer to this computer as "the robot's computer" or "the netbook". Once this computer is set up correctly, it should be very rare for you to remove it from the robot, except to power it on or off. If you find yourself typing on the netbook keyboard regularly after you've got the robot working, you're probably doing something wrong.
- A **desktop or laptop** computer that you'll use to operate the robot. I'll refer to this computer as the "workstation" in these notes. Most all of the typing and clicking you do will be on this computer.

It will be important to keep careful track of the differences between these two computers. Each will have its own commands for you to give and ROS nodes to execute.

### 1.3 Getting your Turtlebot kit

We have access to six Turtlebot robot bases for this class. (Note that there are seven groups, so it will not be possible for everyone to work on the project simultaneously. Please plan accordingly and be considerate to your classmates.) These robots will be stored in the 3D22 lab and should remain there, connected to their chargers, when they're not in use.

In addition to the robot bases, I will distribute to each group a kit of additional hardware to use throughout the rest of the semester. Each kit will include:

1. A netbook, to ride on and control the robot base.
2. A USB stick, to install an operating system on the netbook.
3. A wireless access point (WAP), to enable communication between the netbook and a workstation.
4. A spare battery for the robot bases.
5. Cables for all of these devices.

The idea is that the parts of the robot that need to be configured or charged will be checked out to your group individually. Before you can check out a kit, each member of your group must read and sign a Robot Signout Form, acknowledging responsibility for the equipment and promising to return it in good condition at the end of the semester.

## 2 Netbook setup

The first step toward getting your Turtlebot up and running is to install and configure ROS on the netbook. You should start "from scratch" on this, without assuming that any useful files of any kind are stored on that computer.

### 2.1 Installing the operating system

Before you can install ROS itself, you'll need to install an operating system on the netbook.

1. Using another computer of your choice, download the installation image (that is, the ISO) for the Ubuntu GNU/Linux operating system.
   (a) I strongly suggest that you download Ubuntu 13.04, which is the latest version that works with ROS `hydro`.
   (b) I also suggest that you stick to the 32 bit version; so far 64-bit operating systems often seem to create more trouble than they're worth.

   You may use other Linux distributions if you prefer, but Ubuntu seems to have the best ROS support.
2. Create a bootable USB flash drive from this image.
   (a) Your robot kit includes a USB flash drive that you can use for this purpose.
   (b) The Ubuntu download page has detailed instructions on how to make the drive bootable.

(c) If you are using an Ubuntu computer to create the bootable USB, this command will start the program you need:

```
usb-creator-gtk
```

3. Boot the netbook from the USB drive.

   You may need to configure the netbook's boot sequence (that is, the ordered list of hardware devices from which it tries to load an operating system) to force it to start from the USB drive, rather than from the netbook's hard disk. The details of how to do this vary depending on the specific netbook model you're using, but for the ASUS netbooks we have:

   (a) Pressing F2 to access the BIOS menu when the computer first starts to boot generally seems to be the first step.

   (b) Within the BIOS menu, bootable USB devices (such as your ClearPath USB stick) might show up as hard disks and not, as you might expect, as removable devices.

4. Follow the on-screen prompts to install Ubuntu.

   (a) You should see a screen that has options to "Try Ubuntu" or "Install Ubuntu". If you get a typical Linux login screen instead, it's likely that something went wrong in the previous step.

   (b) Choose "Install Ubuntu".

   (c) When you're asked, I suggest choosing both "Download updates while installing" and "Install third-party software".

   (d) If you are asked about operating systems that are already installed, I recommend that you *vaporize* anything that is already on the netbook, so that you new Ubuntu 13.04 installation uses the entire hard disk.

   (e) During the installation process, you will also be prompted to provide a name for the computer and to create a user account. Choose a memorable name (suggestion: `turtlebot`) and a reasonably secure password for this account.

   (f) Don't forget to remove the USB drive during the reboot, after the installation is finished. Otherwise, you'll find yourself back at the beginning of the installation process.

## 2.2 Installing basic software

You will likely want to install other software beyond the packages pre-installed with Ubuntu. Use the `apt-get` command-line tool—You might want to ask Google about `apt-get` if you haven't used it before—to install and manage packages.

There are two "standard" (that is, non-ROS) packages that you will definitely want to install:

1. The package `openssh-server` provides an SSH server, so that you can execute commands on the netbook remotely over the network via `ssh`.

2. The package `chrony` package provides a clock synchronization service. This is important because, as you have seen, ROS relies on timestamps in many of its messages.

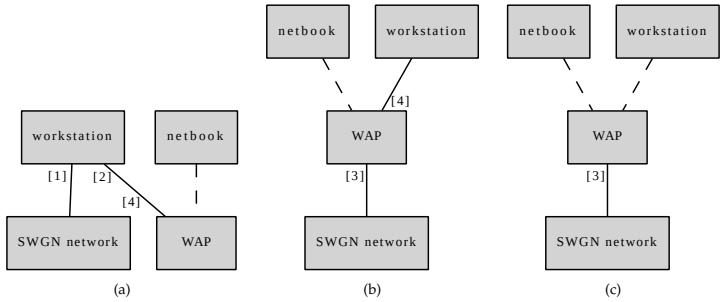


Figure 1: Physical network setup. (a) Recommended setup for 3d22 lab computers. (b, c) Alternatives for using a laptop as the workstation. Dashed lines indicate wireless connections. Solid lines indicate wired connections. [1] Workstation primary ethernet; [2] Workstation secondary ethernet; [3] WAP "Internet" or "WAN" port; [4] WAP "ethernet" or "LAN" port

## 2.3 Installing ROS

Now that you have a working Ubuntu installation on the robot's netbook, the next step is to install ROS itself.

1. First, install the core ROS packages. Section 2.1 of the textbook has instructions on how to do this.
2. Second, there are a few packages that are specific to the Turtlebot that you'll need as well. They are:
   - `ros-hydro-turtlebot`
   - `ros-hydro-turtlebot-apps`
   - `ros-hydro-turtlebot-create`
   - `ros-hydro-turtlebot-create-desktop`
   - `ros-hydro-turtlebot-dashboard`
   - `ros-hydro-turtlebot-simulator`
   - `ros-hydro-turtlebot-viz`

   These can be installed with the usual `apt-get install` command. (You may not use all of these, but it's harmless to install of them.)

## 2.4 Physical setup

There are several choices for how to connect things to enable the workstation and the netbook to talk to each other, but the recommended option for the 3d22 lab computers appears in Figure 1a. Figures 1b and 1c show alternatives that might make sense if your workstation is a laptop.

## 2.5 Wireless access point configuration

Your robot kit includes a wireless access point (WAP) that you should use to allow the netbook to connect to the outside world. Since WAPs vary, I omit detailed instructions on how to configure this. At a minimum, you should

1. set the WAP's SSID (sometimes called "network name") to the name of the WAP itself. (Example: `turtlebot5`)
2. enable some form of password protection on the wireless network, and
3. set the WAP's administrator password to some memorable but reasonably secure password.

Some students in the past have also had success changing the wireless channel, to avoid interference with the wireless signals used by other groups.

On the netbook side, you can attempt to connect to a wireless network using this icon in the top right of the screen:  . You'll know that you've succeeded at this step if the netbook is able to connect correctly to the wireless network provided by your WAP.

## 2.6 Netbook and workstation network configuration

ROS requires three things to be true in order to work correctly over a network:

1. **Each computer must use a name and IP address that can be resolved by each other computer.** The easiest way to accomplish this is to edit the file `/etc/hosts` on both the workstation and the netbook, adding a line listing the IP address and hostname of the other computer.

   For example, here's the hosts file from my workstation, whose name is `raphael` and whose IP address is `192.168.1.100`:

   ```
   127.0.0.1 localhost
   127.0.1.1 raphael
   192.168.1.101 turtlebot1
   ```

   Here's the hosts file from my netbook, whose name is `turtlebot1` and whose IP address is `192.168.1.101`:

   ```
   127.0.0.1 localhost
   127.0.1.1 turtlebot1
   192.168.1.100 raphael
   ```

   You can determine the specific IP address assigned to a computer using the `ifconfig` command, or possibly directly from the administration interface of your WAP. Note that these IP addresses are assigned dynamically by the WAP, so you should verify that the hosts files are set up correctly each time you start working with the robot.

   You can verify that this step has been completed correctly by using the `ping` command. Be sure to use hostnames (rather than IP addresses), since ROS uses hostnames internally.

   (The ROS documentation also mentions an alternative method that uses environment variables called `ROS_IP` and `ROS_HOSTNAME`. This approach seems less reliable in my experience.)

2. **Each computer must be able to contact the same ROS master.** Recall that there should be exactly one master, regardless of how many computers you use. The location of the master is controlled by the environment variable `ROS_MASTER_URI`. For a Turtlebot, it is recommended to run the master on the netbook. For example, here's the command I used:

   ```
   export ROS_MASTER_URI=http://turtlebot1:11311
   ```

The number at the end of this URI is a port number; 11311 is the default port for the ROS master. Don't forget to ensure that this variable is set in every terminal you use on the workstation.

3. **Each computer can initiate a connection to any port on any other computer.** This is governed by the configuration of the network hardware. In the setups shown in Figure 1, you should get this for free, but in more complex scenarios, it can become tricky.

**References**

- `http://www.ros.org/wiki/ROS/NetworkSetup`
- `http://www.ros.org/wiki/ROS/Tutorials/MultipleMachines`
- `http://ros.org/wiki/ROS/EnvironmentVariables`

## 2.7 Testing the network configuration

After completing the network configuration, I strongly suggest that you do some testing to make sure ROS works correctly in your setup, before you continue. As a simple example, you should be able to start `roscore` on the netbook, and then run `turtlesim_node` and `turtle_teleop_key` on the two different machines—Try it in both directions!—and control the simulated robot on one computer from the other.

## 2.8 Connecting to the netbook via SSH

By now you should be able to connect to the netbook via SSH from your workstation. The command should be something like:

```
ssh turtlebot1@turtlebot1
```

If this works correctly, you should get a command prompt on the netbook. This is crucial because the netbook will be mounted on the robot—with its keyboard quite inaccessible—most of the time.

## 2.9 Sharing files between the workstation and netbook

It's important to note that ROS does not transfer files between the computers. Therefore, you will want to find a way to develop software on your workstation and transfer that software the netbook to execute there. One possible option is use `scp` to copy files; another is to use Dropbox to sync things automatically; still another is to use `sshfs` (which does not require any new software on the workstation side) to mount the workstation's ROS workspace on the netbook, or `rsync` (my favorite) to rapidly synchronize only the files that have changed. There are other solutions as well.

## 2.10 How to compile on workstation and run on netbook

So far, we've used the `devel/setup.bash` script generated by `catkin_make` to configure the environment variables that ROS needs to find the nodes that we write. However, this method probably won't work

if attempt to compile your code on your workstation and run it on the netbook, because `setup.bash` hard-codes the path to your ROS workspace, which is unlikely to be the same between the two computers.

The easiest solution is to use the `install` features of catkin, which create a `setup.bash` that works correctly, even across multiple computers with different directory structures. There are a few steps.

- Add `install` commands to your `CMakeLists.txt`, mentioning all of the final products (executables, launch files, data files, etc.) that should be available on the netbook.
  - To install executables, use code like this:

    ```
    install(TARGETS executable-names
       RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
    )
    ```

  - To install all of the launch files contained in a `launch` subdirectory inside your package folder, use code like this:

    ```
    install(DIRECTORY launch/
       DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}/launch
    )
    ```

- After making these changes, you can use this command on the workstation to build everything and create the install space:

  ```
  catkin_make install
  ```

- Finally, after transferring the fully-built workspace to your netbook, you can use this command to configure your environment to find the installed packages:

  ```
  source install/setup.bash
  ```

  This command replaces the usual `source devel/setup.bash` that you're accustomed to.

### 2.11 Where should nodes run?

Now that there are multiple computers in your system, you'll need to decide which computer should run each node. There are several factors that can influence this decision, but a general rule of thumb is that any nodes that have GUI components should run on the workstation, and all others should run on the netbook.

## 3 Turtlebot operation

You should now be ready to start operating a Turtlebot. This section has some details on how to do that. As usual, ROS has an abundance of documentation, of which much is directly useful to us. Two useful starting places are the Turtlebot tutorial page:

- `http://www.ros.org/wiki/turtlebot/Tutorials`

and the Turtlebot "Care and Feeding" page:

- `http://www.ros.org/wiki/turtlebot/Tutorials/TurtleBot%20Care%20and%20Feeding`

## 3.1 Physical connections

There are a two steps to take each time you want to use the robot.

1. Place the powered-on netbook in the robot, oriented so that the side with two USB ports is in the back. (You may need to configure Ubuntu's power settings to prevent the netbook from powering down when you close it.) Plug in both USB cables. One of these cables connects the robot to the iRobot Create mobile base; the other connects it to the Kinect RGBD sensor.
2. Turn on the iRobot Create base. There's a button inside the lowest plate that looks a bit like this: ⏻. If you hear a four-note sad song, this indicates that the Create's battery is too low to operate correctly, and you'll need to recharge or swap out the battery.

## 3.2 Basic Turtlebot Nodes

This section introduces several nodes that you'll want to use every time you operate the Turtlebot. (In the `turtlebot_bringup` package, you can see an example launch file called `minimal.launch` that uses most of these nodes. However, you should construct your own launch file, to ensure that you understand what's going on inside your robot.)

Be sure to execute these nodes on the robot's netbook, rather than on your workstation.

### 3.2.1 Driver for the Create Base

The `create_node` package provides a node called `turtlebot_node.py`. This node is the program that communicates with the iRobot Create mobile base using the white serial cable. (If you've taken CSCE374 recently, then you should have some idea about how this program works.)

Before the usual `rosrun` command (or `roslaunch` `node` element), there are two additional steps on the netbook that are needed to get this node working correctly:

1. First, you need to make sure that you have permission to read from and write to the robot's serial connection. This device is called `/dev/ttyUSB0`, and its `ls -l` entry looks like this:

   ```
   crw-rw---- 1 root dialout 188, 0 Mar 1 13:52 /dev/ttyUSB0
   ```

   The important thing here is that the device is accessible to either `root` (that is, the system's "superuser" administrator account) or to members of the group `dialout`. Since you don't want to run your ROS code as `root`, you must instead ensure that the account that you'll use is in the `dialout` group. You can get a list of the groups a user is in using this command:

   ```
   groups
   ```

If you don't see `dialout` in that list, then you'll need to add it by editing the file named `/etc/group`. In that file, find the `dialout` line and add the user name at the end. For example, my `/etc/group` file has this line:

```
dialout:x:20:jokane
```

You may need to log out and back in for this change to take effect.

2. Second, you'll need to update a file in the standard ROS distribution to correct a small but fatal bug. Specifically, you must (as `root`), comment out line 213 in this file:

```
/opt/ros/hydro/lib/python2.7/dist-packages/create_driver/create_driver.py
```

Once you have `turtlebot_node.py` running, you'll see that it subscribes to a topic called `cmd_vel` that works in much the same way that it does for `turtlesim`. By publishing messages on this topic, you can move the robot around. This node also publishes on several topics, most of which we'll discuss below in the context of other nodes that subscribe to them.

#### 3.2.2 Velocity Command Multiplexer

As you control the robot, you may have several different nodes that want to publish `cmd_vel` messages. Which one should have control of the robot? If everyone publishes directly to `cmd_vel`, then the robot will always try to execute the command in the most recent message it has received. This is obviously not a good solution if, for example, you'd like to take teleoperative control of the robot to override automatically generated commands sent by your software.

ROS provides a solution to this problem in the form of a `multiplexer node`. Each node that wants to send movement commands to the robot, instead of publishing directly to `cmd_vel`, publishes messages on one of three different topics:

- `/cmd_vel_mux/input/navi`
- `/cmd_vel_mux/input/teleop`
- `/cmd_vel_mux/input/safety_controller`

(Note that these specific topic names are determined by a configuration file; there's nothing stopping you from changing them, or adding others if you like.) When messages arrive on any of these topics, `cmd_vel_mux` decides which should take the highest priority, and forwards the corresponding messages to the `turtlebot_node` via `cmd_vel`.

Here's a launch file entry for this node:

```
<node
   pkg="nodelet"
   type="nodelet"
   name="cmd_vel_mux"
   args="standalone yocs_cmd_vel_mux/CmdVelMuxNodelet"
>
   <param name="yaml_cfg_file" value="$(find turtlebot_bringup)/param/mux.yaml"/>
```

```
    <remap from="cmd_vel_mux/input/teleop" to="turtlebot_teleop/cmd_vel"/>
    <remap from="cmd_vel_mux/output" to="cmd_vel"/>
</node>
```

There are three noteworthy things here.

1. First and most noticeably, the `cmd_vel_mux` functionality is actually provided by a *nodelet* rather than a full-fledged node. The idea is, when nodes are very small and very simple, to reduce overhead by combining the functionality of several nodes into a single process. Usually, we would first start a *nodelet manager*, and then *load* one or more nodelets into that manager. In this case, however, there's only one nodelet, so we can launch it as a `standalone` node, without a separate manager. Later, we'll load multiple nodelets into the same manager.
2. Second, we must provide a configuration file, which defines the input and output topics for the multiplexer, along with a priority level and a timeout for each input topic. You might understand the role of `cmd_vel_mux` better if you examine this configuration file.
3. Finally, we use several `remap` entries to modify the topic names used by this node, to ensure that the correct connections are made with the other nodes.

#### 3.2.3 Teleoperation Node

Once the `turtlebot_node` and the `cmd_vel_mux` nodes are running, you can teleoperate the robot using a command like this on your workstation:

```
rosrun turtlebot_teleop turtlebot_teleop_key
```

#### 3.2.4 Robot Description and Robot State Publisher

It will be useful to be able to display the robot itself in rviz. Fortunately, rviz supports a RobotModel display that does just this. However, there are two preliminary steps that we need to take to get rviz the information it needs to do this correctly.

- First, we must set a parameter called `robot_description`. (See Chapter 7 of the textbook for details on what parameters are and how to use them.) This parameter is a (rather long) string that describes, in a format called URDF, each component of the robot, along with the relationships between those parts.

  Fortunately, someone else has already written this description for our Turtlebots, so we only need to ensure that the `robot_description` parameter is set correctly. A launch element like this should do the job:

```
<param
   name="robot_description"
   command="$(find xacro)/xacro.py
      $(find turtlebot_description)/robots/create_circles_kinect.urdf.xacro"
/>
```

- Second, you'll want to start a node called `robot_state_publisher`, which lives in the `robot_state_publisher` package. This node's job is to publish `tf` transforms—that is, information about the relationships between the various coordinate systems that the robot uses—between all of the robot's parts. (Note that, since this robot does not have any movable joints, these transformations won't change. We still need the `robot_state_publisher` because this system is designed to also work for robots that do have joints.)

  This node needs information from both the `robot_description` parameter (see above) and the messages published by the Create driver on `/joint_states`.

After completing these two steps, you should be able to use a RobotModel display to see the robot in rviz.

#### 3.2.5 Diagnostic Aggregator

You should also start an `aggregator_node` node from the `diagnostics_aggregator` package. This node subscribes to `/diagnostics`, a topic published by the create driver, organizes that data according to the robot's subsystems (power system, sensors, etc) are re-publishes it at a lower rate on `/diagnostics_agg`.

A launch file entry for `aggregator_node` might look like this:

```
<node
   pkg="diagnostic_aggregator"
   type="aggregator_node"
   name="diagnostic_aggregator"
>
   <rosparam
      command="load"
      file="$(find turtlebot_bringup)/param/create/diagnostics.yaml"
   />
</node>
```

Here we're using a `rosparam` file (which is in YAML format) to set several parameters at once. In this case, those parameters describe how the `diagnostics` messages should be organized.

The value of the aggregator is that you can view the output from this process, which can be useful for confirming that your robot is working correctly, using the `rqt` tool (a more general form of the `rqt_graph` command that you've used already). Use this command on your workstation:

```
rqt
```

Then use the Plugins menu to add a "Runtime Monitor".

#### 3.2.6 Extended Kalman Filter

Next, you'll want to create a node of type `robot_pose_ekf`, from the package named `robot_pose_ekf`. The node subscribes to `/odom` (on which the create driver publishes raw estimates of its movements based on its odometers) and `/imu/data` (on which the create driver publishes measurements of its rotation, taken

from its gyroscope). It combines these measurements, using an algorithm called the Extended Kalman Filter, to form both an estimated pose for the robot and a covariance matrix indicating how certain the robot is about that estimate. The EKF node publishes the result of this estimation process on `/robot_pose_ekf/odom`, and as a `tf` frame.

This node needs several parameters to work correctly. Here's a reasonable launch file entry:

```
<node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf">
   <remap from="imu_data" to="imu/data"/>
   <param name="freq" value="10.0"/>
   <param name="sensor_timeout" value="1.0"/>
   <param name="publish_tf" value="true"/>
   <param name="odom_used" value="true"/>
   <param name="imu_used" value="true"/>
   <param name="vo_used" value="false"/>
   <param name="output_frame" value="odom"/>
</node>
```

Note also the `remap` tag, which is used to rename the output topic.

## 3.3 Kinect power supply

For the Kinect sensor to operate correctly, it must draw power from the Create's battery. (The 5V available from the USB connection to the netbook is not sufficient; the Kinect needs at least 8V. Note that seeing blinking lights in the Kinect is, in my experience, *not* enough to know for sure that the Kinect is receiving the power it needs.)

Your Turtlebot has a cable connecting the Kinect to the Create for this purpose, but by default that power supply is disabled. To enable it, use a node like this:

```
<node
   pkg="create_node"
   type="kinect_breaker_enabler.py"
   name="kinect_breaker_enabler"
/>
```

This script will call a service called `set_operation_mode`, which is offered by `turtlebot_node`. This service puts the robot into "Full" mode (as its name suggests) and also enables the breaker that supplies power to the Kinect. After successfully calling that service, this node will terminate.

## 3.4 Kinect driver

You will want to launch several nodelets whose job is to publish data from the robot's Kinect RGBD sensor. The standard ROS distribution contains a launch file, which you can import into your own launch files, that does what we need. Here's an example:

```
<include file="$(find openni_launch)/launch/openni.launch">
   <arg name="camera" value="camera"/>
   <arg name="publish_tf" value="true"/>
   <arg name="depth_registration" value="false"/>
```

```
    <arg name="num_worker_threads" value="4" />
    <arg name="rgb_processing" value="true"/>
    <arg name="ir_processing" value="false"/>
    <arg name="depth_processing" value="false"/>
    <arg name="depth_registered_processing" value="false"/>
    <arg name="disparity_processing" value="false"/>
    <arg name="disparity_registered_processing" value="false"/>
</include>
```

Notice that this launch file accepts several arguments, most of which enable or disable nodelets that publish various kinds of data from the sensor. The idea is that you can reduce the amount computation by loading only the nodelets that you need. It's not likely that you'll need to modify these parameters.

### 3.5 Fake laser scans

Finally, let's add a node that converts the Kinect's sensor data to a format that we're already familiar with, namely `sensor_msgs/LaserScan`. To do this, we can use a `depthimage_to_laserscan` node, like this:

```
<node
    pkg="nodelet"
    type="nodelet"
    name="depthimage_to_laserscan"
    args="standalone depthimage_to_laserscan/DepthImageToLaserScanNodelet"
>
    <param name="scan_height" value="10"/>
    <param name="output_frame_id" value="camera_depth_frame"/>
    <param name="range_min" value="0.45"/>
    <remap from="image" to="camera/depth/image_raw"/>
</node>
```

This node will subscribe to the depth images published by the Kinect, slice the middle 10 rows from those images, choose the smallest distance in each column, and publish the results as a "laserscan". The quotation marks are important here because there is, of course, no laser involved.