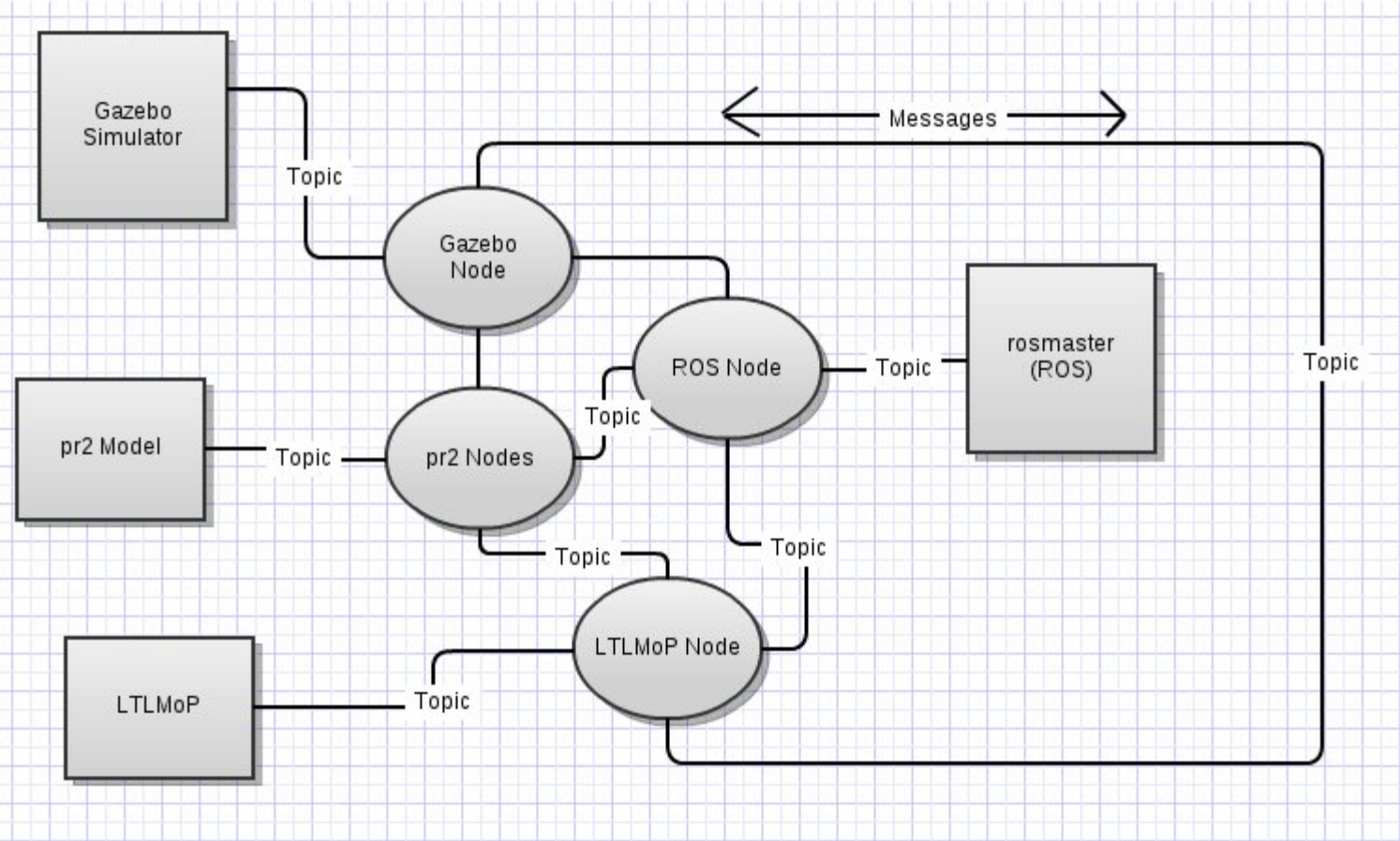
Overview

ROS (Robot Operating System) is a tool for simplifying communication on robotic platforms (In a nutshell). Understanding how it works is helpful for using it with LTLMoP.

The following shows how ROS is set up to communicate with the Pr2, Gazebo (the simulator), and LTLMoP:



In ROS, you must initiate **nodes** in your software. These are platforms with which you can communicate complicated data between different pieces of software or equipment. ROS is modular in the sense that you can add or subtract components as you wish. These components (software or hardware) create their own node (or nodes) to communicate with the rest of the system.

Each Node establishes **topics** with which it can publish certain kinds of data across the ROS network tree. This data is in the form of **messages**, which take multiple forms, but are generally comparable to an object in an object oriented language like python or c++.

Nodes can only communicate with their creator (the piece of software or hardware) and other nodes. This can be seen in the diagram.

For this project, LTLMoP's init handler interface creates a node to establish connection to the ROS framework after it launches much of the what you see in the diagram above. The pr2 is used as an example, but is not the only robot you can use. The robot usually creates multiple nodes for its different components. You can see a more detailed version of the above map by launching $rxgraph while the simulator is up and running.

The Gazebo Simulator launches with a world file and a robot. Both of which can be specified in LTLMoP.

Installation
To begin, you will need to install Ubuntu
There are plenty of instructions online for this.

Please bring your system up to date **before** running our install script:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
```

Get the latest version of LTLMoP for ROS from GitHub

Change the directory to the main folder

Then:

```
cd src/lib/handlers/robots/ROS/
sudo chmod 777 install.sh
./install.sh
```

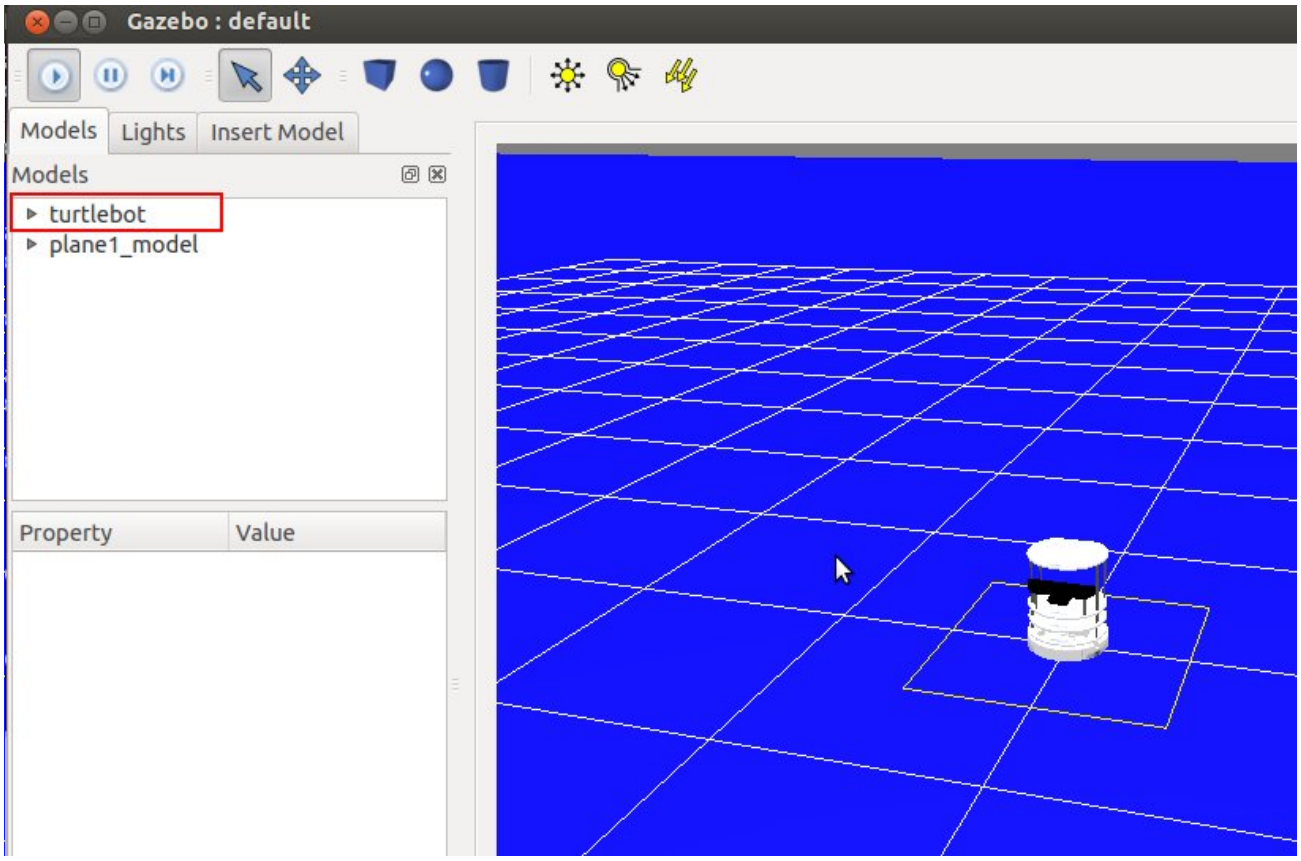
This will take a long time. It should install ros-fuerte and ros-fuerte-pr2 as well as copy the necessary files from the ROS folder to the required directories.
Make sure you check it every once and awhile as it will prompt for inputs like: install??? (y or n): [choose yes]
It will also prompt for the sudo password (you should be on an admin account by the way)

If for any reason my install script breaks (or it doesn't work), open it in a text editor and you can see it is just bash commands in order. You can try to run them one by one to see which one breaks it and then determine where to go from there.

Running
This assumes you have used LTLMoP before. If not, please refer to this tutorial
It also might be a good idea to go through the ROS tutorials.

1. Run LTLMoP and open your favorite spec.
2. Make sure your region file has calibration points.
3. Run Configure Simulation

4. Add a ROS type robot and give it a name
5. *Init Handler* -
    a. Specify an initial region
    b. If you want to use the ltlmop region map as the floor of the world then:
        i. Specify 'ltlmop_map.world' as the world file
        ii. Specify the pixel width of the robot (think the size of the dot in the simulation window of LTLMoP)
        iii. Specify the actual width of the robot in meters
        iv. These allow the initial map size to be set before calibration
    c. If you want to use a different world file then:
        i. Specify the world file
        ii. Pixel width and robot size don't matter for other world sizes
    d. Specify the robot package (generally it is "'robot_name'_gazebo", so ex. pr2_gazebo)
    e. Specify the robot launch file. To determine this, try:
    `roscd package_name`
    ex. `roscd pr2_gazebo`
    This will 'cd' into the package, then do:
    `cd launch`
    `ls`
    You should then see all the launch files associated with the robot. Try and determine which one will launch just the robot (without an empty world, as LTLMoP will launch the world for you) and use that as the launch file.
6. *Pose Handler* -
    a. Specify the name of the robot in gazebo (generally just the name of the type of robot in lowercase) examples: pr2, turtlebot
        i. If you don't know, try launching the robot in gazebo:
        `roslaunch $robot$_gazebo $robot$_empty_world.launch`
        where $robot$ is the name of the robot you are using. This should bring up gazebo with the robot in an 'empty' world. In gazebo, you should see the name of the robot.
        `roslaunch turtlebot_gazebo turtlebot_empty_world.launch`

ii. 

   b. Run the Calibration Tool
      i. This should bring up Gazebo with the robot
      ii. At this point the calibration tool asks for you to move the robot to the given calibration points, you will need to move the robot
      iii. Most robots have a keyboard_teleop utility, for the pr2, in a new window, you can do the following:
      ```
      roslaunch pr2_teleop teleop_keyboard.launch
      ```
      iv. You must be active in the new terminal to control the robot, directions are given for control.
      v. Upon completion, quit the calibration utility and save your spec.
      vi. It is a good idea to kill specEditor at this point as the gazebo window will still be open. If you kill it, you can then restart
      vii. Also kill the teleop program.
7. *Drive Handler* -
   a. Specify the d value (half the length of the robot)
8. Now go to Edit Proposition Mapping
   a. There is one example sensor (the tilt scanner on the pr2) and two templates.
      i. The example takes an operator and a value to test. It tests whether the data is 'operator' the 'value', so if the operator is '>' and the value is '9', and the data is 3, then it will return False
      ii. The templates are for subscribers to topics and service calls.

b. You can specify a couple different actions, if you are familiar with ROS, you can publish on a topic. There are probably a lot of things the robot can do, but to best understand them, you will need to see the ROS tutorials for the robot.
c. For right now, there are two example functions that can be used and a script call example
    i. For gripper, specify which hand you want to use, the new position and the force to be applied
    ii. For the head, specify the frame_id (http://www.ros.org/wiki/pr2_description), the position in the new frame and the minimum travel duration, this one represents a more advanced actuation.
    iii. The script call is kind of unique, but navigation is much easier with the arms down, so I suggest you perform this action when you start. This function is simple, but the script it runs is complicated. For future actuation of the arms, I suggest looking at this script.
    iv. These will serve as good examples if you are to create new functions
d. There are also two template functions in the file. These are very well documented and can be used for future creation of actuation functions. The following might be helpful in this process:
    i. To list the topics (while the robot is running in gazebo) that can be published on, try:
    `rostopic list`
    ii. To determine the msg type of a topic try:
    `rostopic info 'topic'`
    iii. To determine the attributes of a message type, try:
    `rosmsg show 'message'`

9. Save the configuration, compile and try running.
10. It will take a second to come up and it will be very chatty.
    a. Exception AttributeError: AttributeError("'_DummyThread' object has no attribute '_Thread__block'",) in <module 'threading' from '/usr/lib/python2.7/threading.pyc'> ignored
    is an open bug in python 2.7 and should not be of any concern.
    It **Will** print many times, including in the simulation output window. You can ignore it.
    b. Anything in orange is also not an issue and generally deals with the pr2 not being updated to SDF format: http://gazebosim.org/wiki/sdf_format
    The files LTLMoP uses are all up to date in this format (unless they change the format), so in the future these errors should decrease as the pr2's model gets updated to this new version.
    c. Anything in red should be noted, but there are a few messages that will print but not cause problems. If gazebo does not launch, kill it, wait a little bit and restart, you probably started gazebo too quickly after killing it and not all of its components were able to terminate.

11. At this point the simulation window should come up, and gazebo should be running with the robot in the specified initial region. The mouse has left-zoom, right-pan3d, and middle-pan2d, so use that to gain a good view of your map.
12. You will need to start the simulation without waiting too long, some sockets might crash if you wait too long (LTLMoP - so this may be able to be fixed)
13. Upon starting, everything should look normal minus a couple of random outputs in the log window. (like the python error)
14. If there are issues at any point during the process of running everything, crtl-c the terminal window and restart. Make sure all python processes are dead before restarting ltlmop, try:
    ```
    pgrep python
    ```
    If this has output, you have python processes still running, to kill, try:
    ```
    killall -9 python
    ```
15. Collisions in gazebo are fairly strange at the moment. If you are having problems with collisions, look at the world file and check the 'contact_max_velocity' value. Values less than 5 look most normal for when the robot runs into walls and such; however, those values cause the robot to shake and actuations tend to not work well. Values between 10 and 100 look the best as long as you don't collide with walls and objects.