1 Dates

- Assignment 1 handin due: Friday, September 25, 2009

2 Introduction

This project is structured to acquaint you with the basics of writing robot controllers using ROS (the Robot Operating System). ROS runs onboard a single robot and provides an interface to the robot's sensors and actuators over an IP network. Many interesting research-level projects can be implemented using ROS. From the ROS website:

> ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.[1]

3 ROS Framework and the Brown ROS Package

The basic building block in ROS is a *node*. Each node is a modular process with some specialized purpose. For example, there may be a node that handles the basic movement of the robot, and there may be another node that simply publishes information from a camera. From these simple nodes, more complex nodes can be created. For example, there may be another node responsible for blob finding that requires information from a camera driver node. A primary motivation of ROS is to promote code reuse, and the modularity and platform-independence of nodes helps serve this purpose.

A complex program could potentially consist of tens or hundreds of nodes, and these nodes need some efficient way to communicate with each other. Thus, every program in ROS is run with a special node called the *master node*. Conceptually, this node is a matchmaker: it is responsible for listening to node offers and requests, and then introducing relevant nodes to each other so that they can communicate.

The Brown ROS package is a set of nodes that provides basic sensing and movement functionality for the iRobot Create. For this assignment, you will be using the `irobot_create_controller` node, which provides simple methods to determine whether the robot's left or right bumper is pressed, as well as to set the robot's speed and rotation.

4 Instructions

For this assignment, you will be expected to work with our installation of ROS on the EeePC to write a simple random traversal controller. Before writing your own controller, you must first setup the robot and

[1] From `http://www.ros.org/wiki/ROS/Introduction`

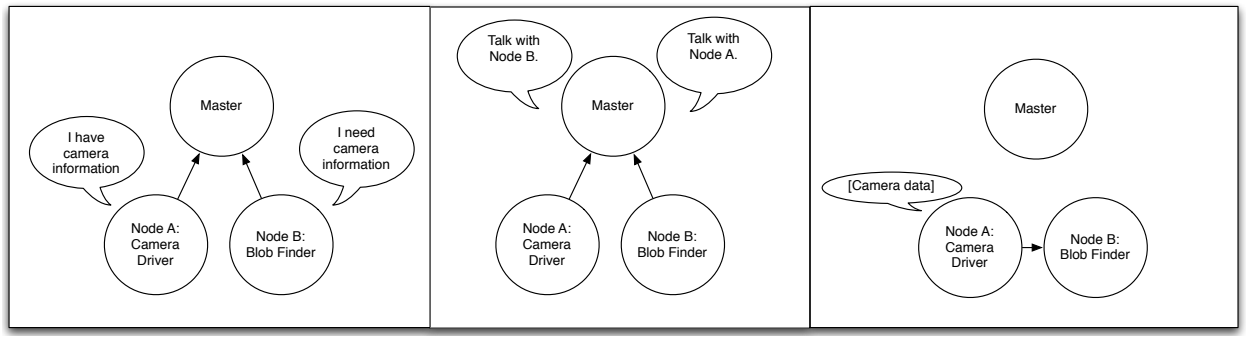

Figure 1: A conceptual view of a higher-level node calling a lower-level node. The master node is responsible for matching nodes to enable communication between them. For a more technical version of this diagram, see below and visit the ROS website.

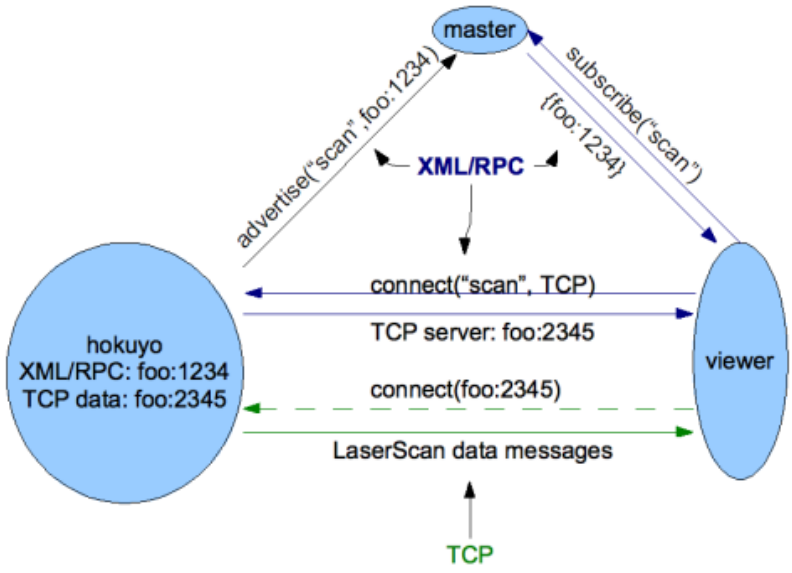

Figure 2: A more technically correct version of the above diagram. Details can be found at the ROS website.

verify that it is running properly. Once your controller is completed, you will run a number of experiments to evaluate how well your robot can escape from an enclosed area.

- Get a robot, and make sure it runs:
  - Ask one of the TAs for access to the Roomba Lab (CIT 404), an iRobot Create, and an IP address. Also ask them for your svn username/password and verify your group name, which you'll need for the second part of the assignment.
  - Turn on the iRobot Create and the mounted EeePC computer. Make sure that the EeePC is connected to the AIBO wireless network (no password is necessary).
  - Each EeePC has an IP address in the form 10.100.0.x, where x is posted next to each robot name on the whiteboard in the Roobma Lab. Determine the IP address for the robot you are currently

working with by looking at the whiteboard or by running the command `/sbin/ifconfig` on the EeePC.

- From a computer in the Roomba Lab connected to the AIBO network,
  - Open a terminal on one of the two computers in 404 and type `ssh obot@IPADDRESS`, where `IPADDRESS` is the IP address of the robot you are currently working with. This will allow you to remotely connect to your EeePC, so that you do not have to do your coding on a small laptop.
  - From a terminal remotely connected to your EeePC, enter the command:
    `roscore`
    This command launches the ROS Master, which must be running for other ROS nodes to locate each other and communicate.
  - From another terminal remotely connected to your EeePC, enter the command:
    `rosrun irobot_create driver`
    This command runs the iRobot Create driver node from the Brown ROS package. The purpose of this node is to provide other nodes with basic sensor information (e.g. whether or not a bumper has been pressed), as well as to provide basic movement control to other nodes for the iRobot Create.
  - From another terminal remotely connected to your EeePC, enter the command:
    `rosrun irobot_create controller_gui`
    This command runs a node that opens a GUI to control the iRobot Create. This node will send messages to the driver node requesting that the robot move with a particular speed and direction, based on user input from the GUI.
- After you start the `controller_gui` node, a window will pop up with a circular grid. You can now control the robot by dragging the marker in the center around in different directions. Moving the marker above/below the center of the grid makes the robot move forward/backwards, and moving the marker left/right of the center of the grid makes the robot turn left/right. The further the marker is from the center, the faster the robot moves. If you are successfully able to control the robot, you are ready to write your own controller.

- Write your own random traversal controller:
  - From a terminal remotely connected to your EeePC, move into the projects directory:
    `cd /home/obot/ros/projects`
  - Create a ROS package for enclosure escape by entering the following command:
    `roscreate-pkg enclosure_escape roscpp std_msgs irobot_create`
    This automatically creates a directory called `enclosure_escape` with dependencies on three other packages (`roscpp, std_msgs,` and `irobot_create`) and generates a number of files for building the new package. Move into this `enclosure_escape` directory: `cd enclosure_escape`
  - Check out your group's svn repository by running:
    `svn checkout svn://foxwood/groupname/trunk src`
  - Move into the repository: `cd src`
  - Pull the skeleton code from the repository into your trunk:
    `svn merge svn://foxwood/groupname/branches/asgn1_ros_skel`
    For detailed instructions on checking out and committing code, refer to the svn tutorial on the course website.
  - We have just checked out a simple controller from the subversion repository, but we need to adjust the `CMakeLists` file so that the controller builds properly. Open the file `enclosure_escape/CMakeLists.txt` and at the end of the file, add the following line:

`rospack_add_executable(escape src/escape.cpp)`
This causes an executable named `escape` to be created whenever you run the command `rosmake enclosure_escape`. For more information on CMake, you can refer to the CMake tutorial on the course website, but most of the work in creating `CMakeLists` is done automatically by ROS.

- Open the file `enclosure_escape/src/escape.cpp` and look through the code for this client. It currently only rotates the robot in place until Ctrl-C is pressed. Before making any changes to the code, verify that you can run this client:
  - Compile the code by entering the command `rosmake enclosure_escape`. This will take a few minutes.
  - If they are not still running, enter the commands `roscore` and `rosrun irobot_create driver` like you did in the previous section.
  - Run the project code by entering the command `rosrun enclosure_escape escape`
  - Before continuing, verify that the robot spins around in place.
- Edit the `mainControlLoop()` method to make the robot move randomly around the room, intelligently reacting to obstacles using its bumper sensors. Your code will use the `irobot_create driver` node to receive bumper state information and move the robot. You can examine (but should not have to edit) this node by viewing the following file:
  `~/ros/brown-ros-pkg/rlab/irobot_create/src/irobot_create_controller/controller.cpp`
- When you are finished, commit the changes you made to `escape.cpp` to the svn repository:
  `svn commit escape.cpp -m 'Added random movement and bumper reaction'`

- Create a written report documenting the work you did for this assignment. Specifically, you should perform an in-depth analysis showing how effective your algorithm is at escaping from various enclosures. You are welcome to experiment with various enclosure escape algorithms and evaluate the relative performance of each. The details of how this report is graded are discussed in the following section. Example reports are provided in the "Documents" section of the course website. When finished, your report should be committed to the `docs` directory of the repository that you checked out.

5 Grading

Your grade for Assignment 1 will be determined by equal weighting of your group's implementation (50%) and your individual written report (50%). The weighted breakdown of grading factors for this assignment are as follows:

Project Implementation	
- Movement Control → Does your robot move smoothly in the environment?	20%
- Obstacle Reaction → Does your robot reasonably detect and respond to obstacles?	20%
- Controller Robustness → Does your controller run without interruption?	10%
Written Report	
- Introduction and Problem Statement → What is your problem? → Why is it interesting?	7%
- Approach and Methods → What is your approach to the problem? → How did you implement your approach and algorithms? → Could someone reproduce your algorithms?	15%
- Experiments and Results → How did you validate your methods? → Describe your variables, controls, and specific tests. → Could someone reproduce your results?	20%
- Conclusion and Discussion → What conclusions can be reached about your problem and approach? → What are the strengths of your approach? → What are the shortcomings of your approach?	8%

6 Appendix: Basic commands

- `roscore`
  - Launches the ROS Master, which must be running for other ROS nodes to locate each other and communicate. For more info, see `http://www.ros.org/wiki/roscore`
- `rosmake enclosure_escape`
  - A tool to assist with building ROS packages. It builds the input package along with any of its dependencies. For more info, see `http://www.ros.org/wiki/rosmake`
- `rosrun enclosure_escape escape`
  - Use this to run a ROS node.
- `svn checkout svn://foxwood/GROUPNAME/asgn1_ros_skel src`
  - Checks out the basic skeleton code for this assignment from the svn repository.
- `svn commit escape.cpp -m 'Added random movement and bumper reaction'`
  - Commits your changes to the svn repository.