10/23/2015

Progress Review 2

Individual Lab Report #3

Abhishek Bhatia

Team D: Team HARP (Human Assistive Robotic Picker)
Teammates: Alex Brinkman, Feroze Naina, Lekha Mohan, Rick Shanor

I. Individual Progress

For the Progress Review 2, my primary responsibility was to tele-operate the PR2 using a joystick (X-Box 360 or Sony PS2/3). The primary idea behind this was to test our preliminary suction design along with PR2 and understand the PR2 configuration space. We prepared a video for the demonstration and discussed it with our sponsor Maxim on Thursday. I started with understanding the normal tele-operation in ROS. I figured out that if I can make the turtlesim simulation (Figure 1) work with the wireless X-Box 360 controller (Figure 2) that I had, I can understand the tele-operation properly. I referred the ROS tutorials available to understand the subscriber and publisher combination properly. I also understood the topics that were published during the turtlesim simulation tele-operation using the keyboard.

I worked on interfacing the X-Box controller with the Linux-ROS environment. I used an off-the-shelf 'joy' package to create this interface. The joy package contains joy_node, a node that interfaces a generic Linux joystick to ROS. The joy node publishes a "Joy" message, which contains the current state of each one of the joystick's buttons and axes. This step was fairly straightforward and in no time was I able to output the joystick data on terminal. Then I developed a new package 'turtlesim_joy' which had dependencies on roscpp, rospy, turtlesim, and joy packages.

Figure 1: Representation of turtle roaming in turtlesim simulation
(Image courtesy: ROS-Wiki)

After this I wrote a publisher-subscriber node (Appendix-A) to subscribe to the data published by joy_node and publish the data to be subscribed by turtlesim node. The mapping was done in a way to connect the left axis stick on the joystick to control the linear velocity and angular velocity of the turtle in the simulation. Further, I developed a launch file to launch (Appendix-B) all the nodes such as turtlesim, joy_node and teleop_turtlesim_joy simultaneously. This small experiment gave me a detailed idea about the complete tele-operation on ROS framework.

Figure 2: X-Box 360 wireless controller
(Image courtesy: google.com)

After this, I installed gazebo and started playing with PR2 in simulation (Figure 3). Gazebo already had a PR2 tele-operation using keyboard package. I understood the topics that were being published to make some more sense.

Figure 3: PR2 simulation in Gazebo
(Image courtesy: Google.com)

On Monday morning we met Andrew Dornbush from SBPL lab to get started with operating PR2. He introduced us to the nitty-gritties of operating PR2, setting up an account for Team HARP and explained us the login process. He demonstrated us the PR2 base movement with the Sony-PS2 controller, but he mentioned there was no proper way to tele-operate the PR2 arms.

This was surprising considering I had done some background study on this and I was aware of an off-the-shelf PR2 package 'pr2_teleop_general' that can be used to tele-operate using either the keyboard or the joystick controller. I followed up with our sponsor Maxim and Andrew to understand the requirements for the demonstration and if we could use this package directly. Turns out that this package was already installed on the PR2 machine and it was as simple as launching a launch file to tele-operate the PR2 using PS2 controller. We ported our suction design on PR2's right arm and prepared demonstrations.

Besides this, I helped Alex to interface Kinect2 with ROS-Indigo. Alex was facing issues with accomplishing this task, he was getting the similar error on his different trials on 3 different machines. He had spent huge amount of time trying to debug the error before he asked me to help. I was done early with my task of tele-operation. Alex and I mutually decided that starting from scratch on a clean machine is better than debugging this issue. This also ensured that I got up to speed easily on this task. I followed the instructions from our reference and step-by-step was able to make this work. Later, me and Alex worked together to create a network between the laptop that was used to interface with Kinect2 and our main machine. Our state controller runs on the main machine, but we cannot interface with Kinect2 on the main machine as it runs ROS-Groovy (Ubuntu-12.04 and ROS-Groovy is a PR2 requirement) and Kinect2 is incompatible with ROS-Groovy. Once the network was set up, we worked together to modify the state controller to accept Kinect2 data, hence completing our tasks for this week.

II. Challenges

I faced multiple challenges this week. The first challenge I faced was while working on the turtlesim teleoperation with joystick. I was doing a silly mistake of subscribing the turtlesim node to the wrong publisher and it took me a while before I discovered my mistake. But, this made me more comfortable with using the ROS framework. I debugged this issue using 'rostopic info' and 'rostopic echo' commands, trying to understand the messages each topic was publishing to or subscribing.

Second issue I faced was during the PR2 tele-operation. I discovered that tele-operating PR2 was extremely difficult, primarily because we were restricting the 7-DOF PR2 arm to work in a 3-D Cartesian space because of limited mappings on the controller. I spent couple of hours practising/playing with PR2 trying to understand the best possible paths to grasp objects that were kept on the table, eventually being able to prepare a successful demonstration.

Later, while working with Alex, I faced difficulty during debugging the state controller. Alex had spent considerable amount of time working on the state controller, but since I was not up to speed on it right now, I spent some time understanding the 'smach' state controller in detail.

III. Teamwork

Alex: This week Alex worked completely on getting the preliminary 'smach' state controller up. Besides he worked on interfacing Kinect2 with ROS-Indigo and worked with Rick to create the portable gripper design (mount that will be help by PR2 gripper, suction cup and hose will be mounted on this).
Feroze: Feroze focussed all his energies on getting the PR2 arm to move from one point to another in simulation. He was able to make this work using inverse kinematics.
Lekha: Lekha worked on large scale 3D recognition test. She interacted with Venkat from SBPL lab to converge on the initial vision algorithm.
Rick: Rick worked with Alex in creating the preliminary suction design. Later, Alex, Rick, and I worked on generating the demonstration, testing our suction design with PR2. Rick also worked with Lekha on Image Segmentation.
Abhishek: I focussed on PR2 tele-operation and generating the final demonstration. Later, I helped Alex with interfacing Kinect2 and debugging state controller.

IV. Future Plans

I have already started working on my next task to control the PR2 base (navigation planning and execution for PR2). During the next few weeks, I will be working on navigation planning and execution of the PR2 base and integrate the algorithm with our state controller. Besides, I will be working with Feroze during the next week to finalize Electronic Design for our suction gripper. Lastly, I want to spend some time on our Preliminary Design Review which is due next week.

V. Appendix

Appendix – A

```
// %Tag(FULL)%
// %Tag(INCLUDE)%
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <sensor_msgs/Joy.h>
// %EndTag(INCLUDE)%
```

```
// %Tag(CLASSDEF)%
class TeleopTurtle
{
public:
  TeleopTurtle();

private:
  void joyCallback(const sensor_msgs::Joy::ConstPtr& joy);

  ros::NodeHandle nh_;

  int linear_, angular_;
  double l_scale_, a_scale_;
  ros::Publisher twist_pub_;
  ros::Subscriber joy_sub_;

};
// %EndTag(CLASSDEF)%
// %Tag(PARAMS)%
TeleopTurtle::TeleopTurtle():
  linear_(1),
  angular_(2)
{

  nh_.param("axis_linear", linear_, linear_);
  nh_.param("axis_angular", angular_, angular_);
  nh_.param("scale_angular", a_scale_, a_scale_);
  nh_.param("scale_linear", l_scale_, l_scale_);
// %EndTag(PARAMS)%
// %Tag(PUB)%
  twist_pub_ = nh_.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1);
// %EndTag(PUB)%
// %Tag(SUB)%
  joy_sub_ = nh_.subscribe<sensor_msgs::Joy>("joy", 10, &TeleopTurtle::joyCallback, this);
// %EndTag(SUB)%
}
// %Tag(CALLBACK)%
void TeleopTurtle::joyCallback(const sensor_msgs::Joy::ConstPtr& joy)
{
  geometry_msgs::Twist twist;
  twist.angular.z = a_scale_*joy->axes[angular_];
```

```
  twist.linear.x = l_scale_*joy->axes[linear_];
  twist_pub_.publish(twist);
}
// %EndTag(CALLBACK)%
// %Tag(MAIN)%
int main(int argc, char** argv)
{
  ros::init(argc, argv, "teleop_turtle");
  TeleopTurtle teleop_turtle;

  ros::spin();
}
// %EndTag(MAIN)%
// %EndTag(FULL)%
```

Appendix - B

```
<launch>

 <!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>


 <!-- joy node -->
  <node respawn="true" pkg="joy"
        type="joy_node" name="turtle_joy" >
    <param name="dev" type="string" value="/dev/input/js0" />
    <param name="deadzone" value="0.12" />
  </node>

 <!-- Axes -->
  <param name="axis_linear" value="1" type="int"/>
  <param name="axis_angular" value="0" type="int"/>
  <param name="scale_linear" value="2" type="double"/>
  <param name="scale_angular" value="2" type="double"/>

  <node pkg="turtlesim_joy" type="teleop_turtle_joy" name="teleop"/>

</launch>
```