ROS Crash Course

Class 7

ROS

UNLV | Howard R. Hughes College of Engineering

DASL

Drones and Autonomous Systems Lab @ UNLV

Agenda

-Multiple publishers and subscribers in one node

-Using ROS over multiple systems

-rosbag

-SSH

-Optional HW

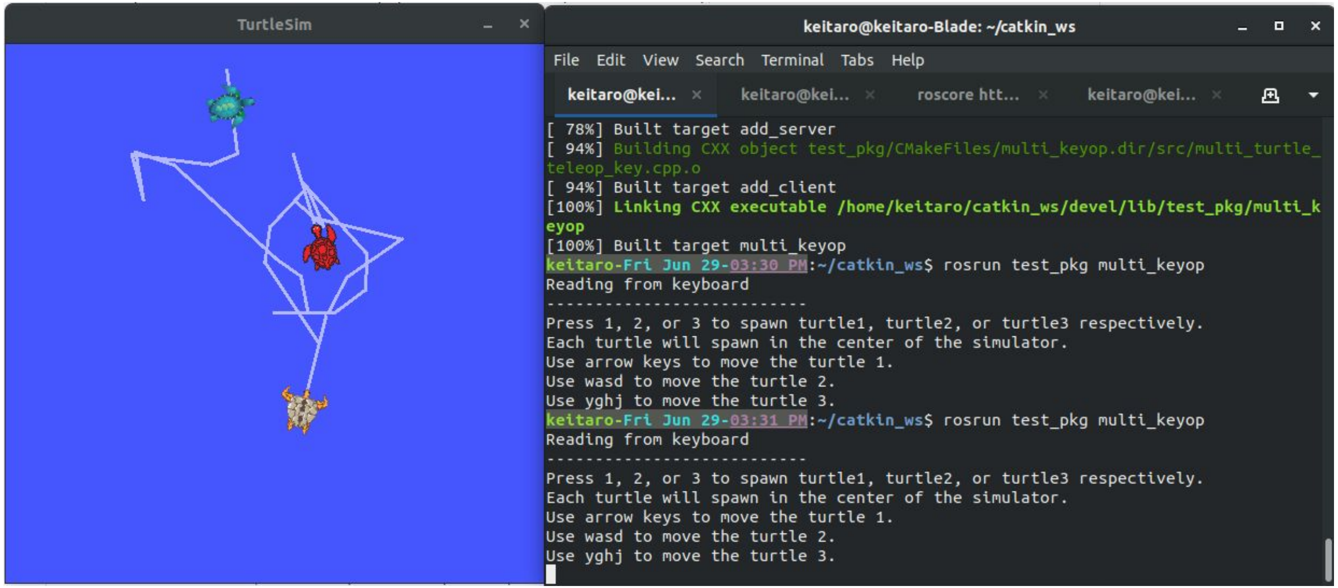
Multiple publishers and subscribers in one node

-Each node is not limited to any amount of publishers or subscribers that can be in it.

-You just need to define and initialize each publisher/subscriber separately and be sure to keep track of which msg goes to which one.

Multiple publishers and subscribers in one node

-Using the turtle_teleop_key.cpp as a base a node to control 3 different turtles at the same time will be made


TurtleSim
keitaro@keitaro-Blade: ~/catkin_ws
File Edit View Search Terminal Tabs Help
keitaro@kei...
keitaro@kei...
roscore htt...
keitaro@kei...

```
[ 78%] Built target add_server
[ 94%] Building CXX object test_pkg/CMakeFiles/multi_keyop.dir/src/multi_turtle_
teleop_key.cpp.o
[ 94%] Built target add_client
[100%] Linking CXX executable /home/keitaro/catkin_ws/devel/lib/test_pkg/multi_k
eyop
[100%] Built target multi_keyop
keitaro-Fri Jun 29-03:30 PM:~/catkin_ws$ rosrun test_pkg multi_keyop
Reading from keyboard
---------------------------
Press 1, 2, or 3 to spawn turtle1, turtle2, or turtle3 respectively.
Each turtle will spawn in the center of the simulator.
Use arrow keys to move the turtle 1.
Use wasd to move the turtle 2.
Use yghj to move the turtle 3.
keitaro-Fri Jun 29-03:31 PM:~/catkin_ws$ rosrun test_pkg multi_keyop
Reading from keyboard
---------------------------
Press 1, 2, or 3 to spawn turtle1, turtle2, or turtle3 respectively.
Each turtle will spawn in the center of the simulator.
Use arrow keys to move the turtle 1.
Use wasd to move the turtle 2.
Use yghj to move the turtle 3.
```

Multiple publishers and subscribers in one node

-Defining multiple publishers is the same as multiple subscribers.

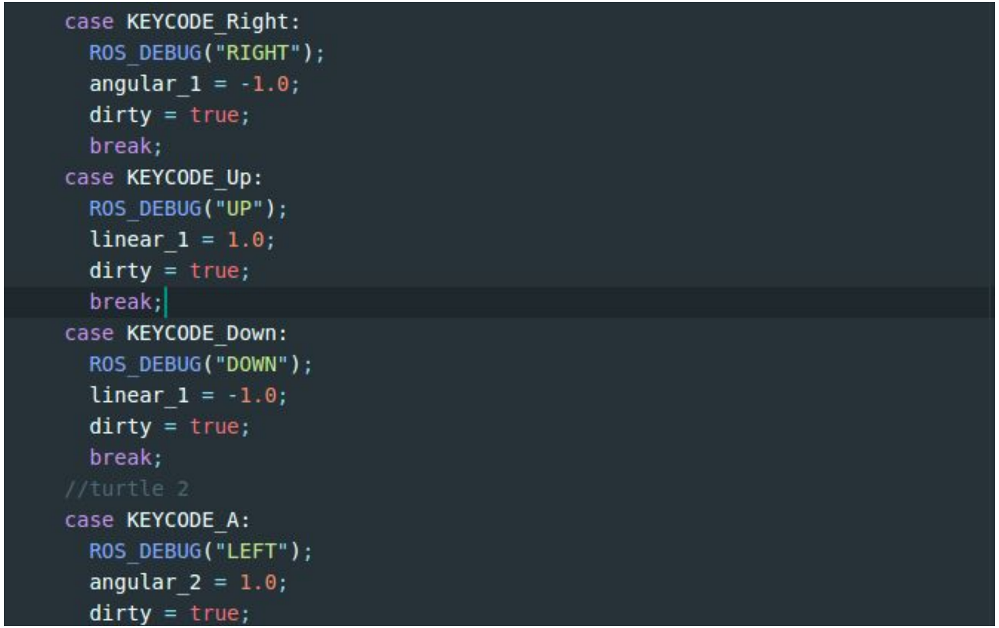
```
ros::Publisher twist_pub_1, twist_pub_2, twist_pub_3;

twist_pub_1 = nh_.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1);
twist_pub_2 = nh_.advertise<geometry_msgs::Twist>("turtle2/cmd_vel", 1);
twist_pub_3 = nh_.advertise<geometry_msgs::Twist>("turtle3/cmd_vel", 1);
```

-For this example code the publishers are global variables, which is why they are used throughout the code

Multiple publishers and subscribers in one node

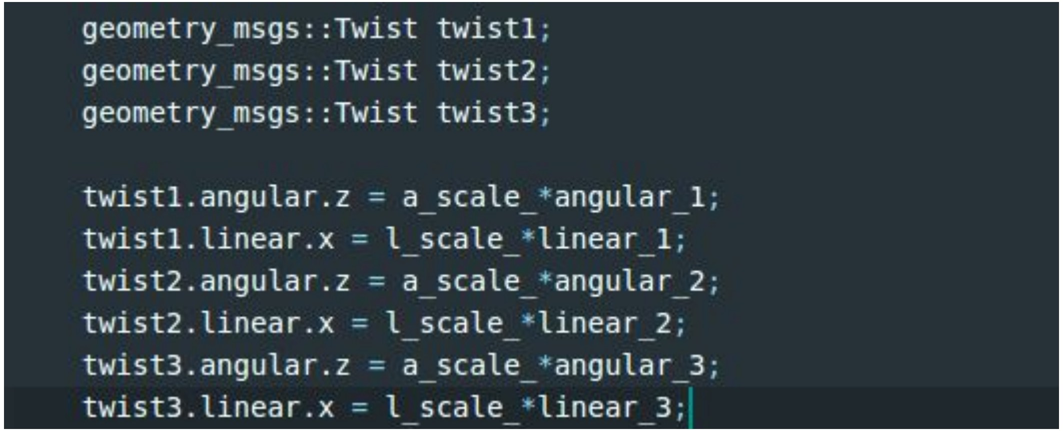
-Notice how for each key input there are different angular and linear variables for each turtle.



```
    case KEYCODE_Right:
      ROS_DEBUG("RIGHT");
      angular_1 = -1.0;
      dirty = true;
      break;
    case KEYCODE_Up:
      ROS_DEBUG("UP");
      linear_1 = 1.0;
      dirty = true;
      break;
    case KEYCODE_Down:
      ROS_DEBUG("DOWN");
      linear_1 = -1.0;
      dirty = true;
      break;
    //turtle 2
    case KEYCODE_A:
      ROS_DEBUG("LEFT");
      angular_2 = 1.0;
      dirty = true;
```

Multiple publishers and subscribers in one node

-This was done so that I wouldn’t get confused which variable was meant for which publisher. There are obviously more elegant solutions but this is quick and dirty.

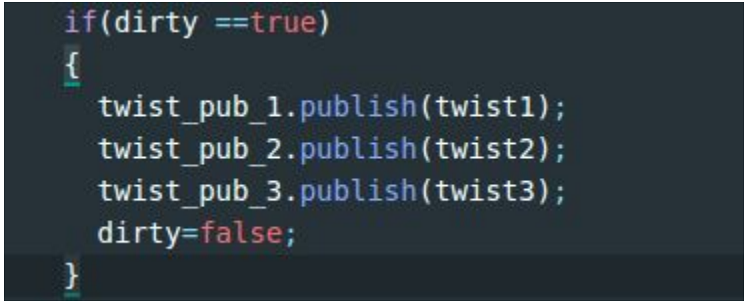


```
    geometry_msgs::Twist twist1;
    geometry_msgs::Twist twist2;
    geometry_msgs::Twist twist3;

    twist1.angular.z = a_scale_*angular_1;
    twist1.linear.x = l_scale_*linear_1;
    twist2.angular.z = a_scale_*angular_2;
    twist2.linear.x = l_scale_*linear_2;
    twist3.angular.z = a_scale_*angular_3;
    twist3.linear.x = l_scale_*linear_3;
```

Multiple publishers and subscribers in one node

-Notice how all three publishers are publishing at the same time as well. This was just done for convenience. There is no reason why each publisher should have a different flag to make sure that unwanted messages aren’t sent.



```
if(dirty ==true)
{
  twist_pub_1.publish(twist1);
  twist_pub_2.publish(twist2);
  twist_pub_3.publish(twist3);
  dirty=false;
}
```

Multiple publishers and subscribers in one node

-Add the example code to your packages and try to run them.

    -don’t forget to add it to your cmake file

-Is there anything you notice about the functionality of the code?

Using ROS over multiple systems

-You can have an infinite amount of machines communicate over a network using ROS but you can only have **1 ROSMASTER** running at a time. This means that every computer must know where the rosmaster is and where each of the nodes reside relative to the network.

Using ROS over multiple systems

-ROS_MASTER_URI

-This is a ROS environment variable that is used to specify where the ros master is running in the network

IP address :port number

```
-eg $ export ROS_MASTER_URI=http://localhost:11311
```

-ROS_IP

-This tells the rosmaster where each node is coming from in the network. This is done on each computer to point to their own IP address

```
Eg $ export ROS_IP=http://localhost:11311
```

Using ROS over multiple systems

-Both the ROS_MASTER_URI and ROS_IP variables can be changed by using shell commands as shown in the previous slide.

-These can also be set through the roslaunch file as well.

rosbag

-This is a command that allows you to save, playback, and work with ros data after actually running the nodes.

-This is how you can playback exact inputs or export data to matlab or other software.

-Though this command creates a .bag file there are python scripts available online which convert .bag files into .csv files.

rosbag

-First make a directory where you want to save your bag files. They are named after the date and time you started them so organization is important.

-Then run roscore, turtlesim_node, and a teleop_key node.

-Open a new shell and cd into the rosbag directory you made and run

```
    $ rosbag record -a
```

-and move the turtle(s) around for a bit

rosbag

-Finally end the rosbag by going back to its shell and pressing ctrl+c

-Then use the command

```
$ rosbag play [the .bag file]
```

To see the inputs played back on the turtlesim.

SSH

-Secure Shell is a protocol used to “remote desktop” into other systems on the same network.

-However, unlike “remote desktop” it is a purely text interface. This is often used to work on robots with integrated or hard to get to/remove computers. It is just like logging into your personal laptop through terminal. You just need to know the IP address of the computer, username, and pw for the username.

Optional HW

-Do Gazebo beginner tutorials (http://gazebosim.org/tutorials)