ROS Programming (C/C++)

Ivan Marković Matko Orsag Damjan Miklić
(Srećko Jurić-Kavelj)

University of Zagreb, Faculty of Electrical Engineering and Computing,
Departement of Control and Computer Engineering

2015

University of Zagreb
Faculty of Electrical Engineering
and Computing

Before we begin

- You have all the required packages already installed for this class
- We will go through the process of creating and building a C/C++ ROS package
- We will write a C/C++ publisher and a subscriber that will work with the `turtlesim` simulator

Creating the package

- Navigate in your `/src` catkin workspace folder
- Create `turtle_lawnmower` package that we will use in this class

```
$ catkin_create_pkg turtle_lawnmower turtlesim roscpp \
 geometry_msgs
```

- Refresh the package list and `roscd` into the project

```
$ rospack profile
$ roscd turtle_lawnmower
```

Writing the subscriber in C/C++

- Create a `turtle_lawnmower_node.cpp` file in the `/src` folder of your `turtle_lawnmower` package
- Program a node that will subscibe to the turtle's `turtlesim/Pose` message and write it into the console (use `ROS_INFO`)

```
// Required headers for the node
#include "ros/ros.h"
#include "geometry_msgs/Twist.h" // turtle's cmd_vel
#include "turtlesim/Pose.h" // reading turtle's position
```

Writing the subscriber in C/C++

```
int main(int argc, char **argv)
{
  // Initialized the node, setup the NodeHandle
  // for handling the communication with the ROS system
  ros::init(argc, argv, "turtle_lawnmower_node");
  ros::NodeHandle nh;

  // Define the subscriber to turtle's position
  ros::Subscriber sub = nh.subscribe
   ("turtle1/pose", 1, turtleCallback);

  ros::spin();

  return 0;
}
```

Writing the subscriber in C/C++

- Write the callback function `turtleCallback` that will be called each time a message is published on the `turtle1/pose` topic (above the `main` function)

```
void turtleCallback(const turtlesim::Pose::ConstPtr& msg)
  {
    ROS_INFO("Turtle lawnmower@[%f, %f, %f]",
      msg->x, msg->y, msg->theta);
  }
```

- The message has been passed in a `boost_shared_ptr` and member of the class being pointed to can be accessed using the dereferencing operator '`->`'

Building the project

- Open the `CMakeLists.txt`
- Check that `find_package` looks for all dependencies
- This will create variables needed in the linking stage (including headers and linking libraries)
- Add your node as an executable (uncomment `add_executable`)
- Link with the required libraries (uncomment `target_link_libraries`)

Building the project

- In the end your `CMakeLists.txt` should look like this

```
cmake_minimum_required(VERSION 2.8.3)
project(turtle_lawnmower)
## Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS
  roscpp
  geometry_msgs
  turtlesim
)
catkin_package()
# include_directories(include)
include_directories(
  ${catkin_INCLUDE_DIRS}
)
```

Building the project

- `CMakeLists.txt` continued ...

```
## Declare a cpp executable
add_executable(
turtle_lawnmower_node src/turtle_lawnmower_node.cpp
)
## Specify libraries to link a library or
## executable target against
target_link_libraries(turtle_lawnmower_node
   ${catkin_LIBRARIES}
)
```

- Call `catkin_make` from the workspace root folder and build the `turtle_lawnmower` project

Testing the node

- Run the `turtlesim` and then your node to see if the callback is running

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtle_lawnmower turtle_lawnmower_node
```

- In the terminal your program should be outputing the turtle's pose (position + orientation)

Publishing velocity commands

- Now we need to add a publisher to our node
- Since we already have a subsciber, and we want to publish in the callback function, we will need to declare the publisher as a global variable
- Above the `turtleCallback` function declare the publisher

  ```
  ros::Publisher pub;
  ```

- Now setup the publisher within the `main` function (e.g. just after the subscriber)

  ```
  pub = nh.advertise<geometry_msgs::Twist>
  ("turtle1/cmd_vel", 1);
  ```

Publishing velocity commands

- In the `turtleCallback` function define the command velocities and publish them

```
geometry_msgs::Twist turtle_cmd_vel;

turtle_cmd_vel.linear.x = 1;
pub.publish(turtle_cmd_vel);
```

- The robot will be moving forward with the designated velocity
- A better way to implement this, and avoid global variables, would be to use class method as a callback function (See here and here for info on callback types)

Homework

For homework you will need to program a landmower algorithm. The turtle should start from the lower left corner and continue straight until it reaches the end and then turn in a small circular arc and continue in the opposite direction. This should go on until turtle covers the whole area. You can assume that the position of the turtle is known (`turtle1/Pose`) and that the size of the environment is known (an 11×11 square). We are not asking you to implement a complete coverage algorithm! (Hint: detect when robot is close to the edge, then make it turn, otherwise maintain straight heading).

Assignments

1. Adapt the `turtle_lawnmower_node` so that the turtle behaves as described in the homework text (you can use global variables)
2. Send us the node source code and a picture of the turtle's path in the `turtlesim` simulator

Homework