Introduction to Cognitive Robotics

Module 2: The Robot Operating System (ROS)

Lecture 3: Writing ROS software in C++: Subscribers

David Vernon
Carnegie Mellon University Africa

www.vernon.eu

Writing ROS Software

- Creating a ROS workspace and a ROS package

- Writing ROS programs

  1. Example program: Hello World!

  2. Example program to publish messages

     Send velocity messages on `/turtle1/cmd_vel`

Writing ROS Software

- Writing ROS programs

3. Example program to subscribe to messages

   Receive pose messages on `/turtle1/pose`

4. Example program to use services

```
/reset
/clear
/turtle1/set_pen
/turtle1/teleport_absolute
```

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Make sure you are in the **agitr** sub-directory

```
~/workspace/ros/src$ cd ~/workspace/ros/src/agitr
```

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Edit CMakeLists.txt

Add the following lines at the end of the file

```
add_executable(${PROJECT_NAME}_subpose src/subpose.cpp)
set_target_properties(${PROJECT_NAME}_subpose PROPERTIES OUTPUT_NAME subpose  PREFIX "")
target_link_libraries(${PROJECT_NAME}_subpose ${catkin_LIBRARIES})
```

This avoids name pollution and allows different packages
to have ROS nodes with the same name

But the executable is still just this
so that you can execute it with rosrun package_name node,
i.e. rosrun agitr subpose

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Move to the agitr/src sub-directory

```
~/workspace/ros/src/agitr$ cd src
~/workspace/ros/src/agitr/src$
```

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Edit subpose.cpp and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
   ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
                "position=(" << msg.x << "," << msg.y << ")" <<
                " direction=" << msg.theta);
}

int main(int argc, char **argv) {

  /* Initialize the ROS system and become a node */
   ros::init(argc, argv, "subscribe_to_pose");
   ros::NodeHandle nh;

   /* Create a subscriber object */
   ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

   /* Let ROS take over */
   ros::spin();
}
```

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Edit subpose.cpp and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
   ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
                "position=(" << msg.x << "," << msg.y << ")" <<
                " direction=" << msg.theta);
}

int main(int argc, char **argv) {

  /* Initialize the ROS system and become a node */
   ros::init(argc, argv, "subscribe_to_pose");
   ros::NodeHandle nh;

   /* Create a subscriber object */
   ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

   /* Let ROS take over */
   ros::spin();
}
```

The callback function is called every time a message is received
We have to put the code to handle the message in this function

The parameter is the message that arrives.
The type of the message is defined in the header file `<turtlesim/Pose.h>`

Here, we simply print the values to the terminal

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Edit subpose.cpp and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
   ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
                "position=(" << msg.x << "," << msg.y << ")" <<
                " direction=" << msg.theta);
}

int main(int argc, char **argv) {

  /* Initialize the ROS system and become a node */
   ros::init(argc, argv, "subscribe_to_pose");
   ros::NodeHandle nh;

   /* Create a subscriber object */
   ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

   /* Let ROS take over */
   ros::spin();
}
```

Instantiate a subscriber object

Initialize it by calling the subscribe method of the node handle object

Subscribe on the `turtle1/pose` topic

using a queue that can handle 1000 messages

Pass a pointer to the callback function that is to be called when messages arrive

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Edit subpose.cpp and insert the following code

```
/* This program subscribes to turtle1/pose and shows its messages on the screen */

#include <ros/ros.h>
#include <turtlesim/Pose.h>
#include <iomanip> // for std::setprecision and std::fixed

/* A callback function. Executed each time a new pose message arrives */
void poseMessageReceived(const turtlesim::Pose& msg) {
   ROS_INFO_STREAM(std::setprecision(2) << std::fixed <<
                "position=(" << msg.x << "," << msg.y << ")" <<
                " direction=" << msg.theta);
}

int main(int argc, char **argv) {

  /* Initialize the ROS system and become a node */
   ros::init(argc, argv, "subscribe_to_pose");
   ros::NodeHandle nh;

   /* Create a subscriber object */
   ros::Subscriber sub = nh.subscribe("turtle1/pose", 1000, &poseMessageReceived);

   /* Let ROS take over */
   ros::spin();
}
```

Allow ROS to service the callback by calling `ros::spin()`
Note: won't return control to this main() function; it will just carry on servicing the call the callback function
If you want to do more work here, call `ros::spinOnce()`
This will allow ROS to execute all pending callback calls and then return control to here
You'll probably embed this call in a loop so that you iteratively service the callback and then do some work

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

Build the workspace to compile the program

- Make sure you are in the workspace directory

```
~/workspace/ros/src/agitr/src$ cd ~/workspace/ros
~/workspace/rosr$
```

- Run catkin_make

```
~/workspace/ros$ catkin_make
```

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

If you have not already done it, open a terminal and enter

```
~$ roscore
```

If you have not already done it, open a second terminal and enter

```
~$ rosrun turtlesim turtlesim_node
```

If you have not already done it, open a third terminal and enter

```
~$ rosrun agitr pubvel
```

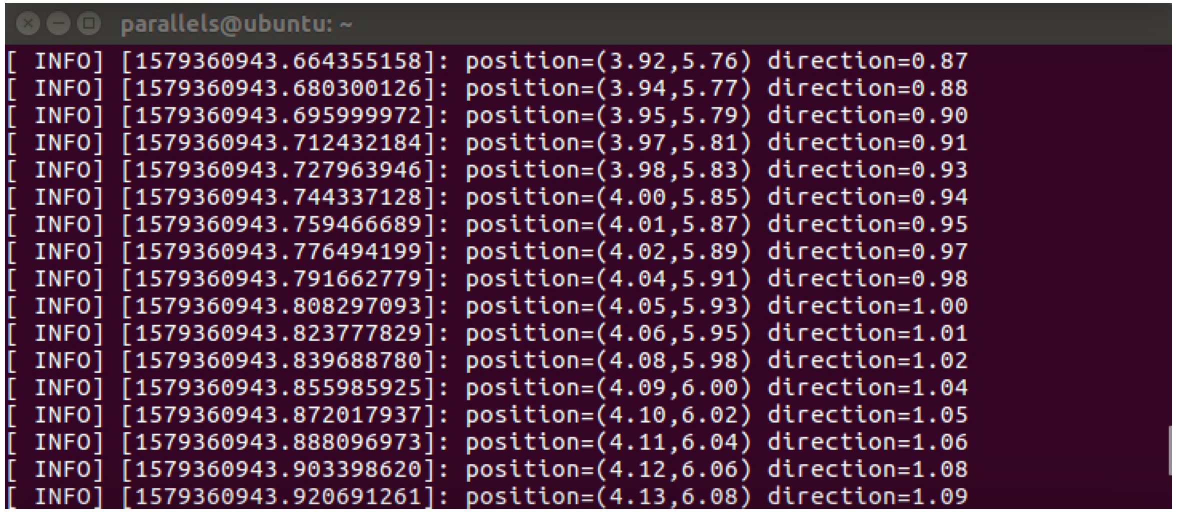
Open a fourth terminal and enter

```
~$ rosrun agitr subpose
```

Example Program to Subscribe to Messages

Receive pose messages on `/turtle1/pose`

If everything works correctly, you should see messages in the fourth terminal detailing the pose of the turtle



```
[ INFO] [1579360943.664355158]: position=(3.92,5.76) direction=0.87
[ INFO] [1579360943.680300126]: position=(3.94,5.77) direction=0.88
[ INFO] [1579360943.695999972]: position=(3.95,5.79) direction=0.90
[ INFO] [1579360943.712432184]: position=(3.97,5.81) direction=0.91
[ INFO] [1579360943.727963946]: position=(3.98,5.83) direction=0.93
[ INFO] [1579360943.744337128]: position=(4.00,5.85) direction=0.94
[ INFO] [1579360943.759466689]: position=(4.01,5.87) direction=0.95
[ INFO] [1579360943.776494199]: position=(4.02,5.89) direction=0.97
[ INFO] [1579360943.791662779]: position=(4.04,5.91) direction=0.98
[ INFO] [1579360943.808297093]: position=(4.05,5.93) direction=1.00
[ INFO] [1579360943.823777829]: position=(4.06,5.95) direction=1.01
[ INFO] [1579360943.839688780]: position=(4.08,5.98) direction=1.02
[ INFO] [1579360943.855985925]: position=(4.09,6.00) direction=1.04
[ INFO] [1579360943.872017937]: position=(4.10,6.02) direction=1.05
[ INFO] [1579360943.888096973]: position=(4.11,6.04) direction=1.06
[ INFO] [1579360943.903398620]: position=(4.12,6.06) direction=1.08
[ INFO] [1579360943.920691261]: position=(4.13,6.08) direction=1.09
```

ROS Resources

Wiki	http://wiki.ros.org/
Installation	http://wiki.ros.org/ROS/Installation
Tutorials	http://wiki.ros.org/ROS/Tutorials
Tutorial Videos	http://www.youtube.com/playlist?list=PLDC89965A56E6A8D6
ROS Cheat Sheet	http://www.tedusar.eu/files/summerschool2013/ROScheatsheet.pdf

Recommended Reading

`http://wiki.ros.org/catkin/Tutorials/create_a_workspace`

`http://wiki.ros.org/ROS/Tutorials/CreatingPackage`

`http://wiki.ros.org/roscpp/Overview/InitializationandShutdown`

`http://wiki.ros.org/roscpp/Overview/NodeHandles`

`http://wiki.ros.org/ROS/Tutorials/BuildingPackages`

`http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c++)`

J. M. O'Kane, A Gentle Introduction to ROS, 2014.
`https://cse.sc.edu/~jokane/agitr/`