ROS Basics with C++ Practical Course

Master 2 AII

October 17, 2023

1 Introduction to ROS

ROS (Robot Operating System) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior.

2 Setting Up ROS Environment

2.1 Installing ROS

Follow this link **https://www.youtube.com/@mega_rosbot** to install ROS.

2.2 Creating a Workspace

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

3 Creating a Package

A package in ROS is a directory that contains the necessary files for a specific purpose. It can include libraries, executables, scripts, and configuration files.

In this section, you've already created a package named `tp_rob` using the `catkin_create_pkg` command. The parameters `rospy`, `roscpp`, and `std_msgs` indicate the dependencies of your package.

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tp_rob rospy roscpp std_msgs
```

4 Creating a First C++ ROS Node

A ROS node is a program that uses ROS to communicate with other nodes. In this section, we're creating a simple C++ node named `hello_robots` that publishes a message to the `chatter` topic.

Here's an explanation of the code:

```
#include "ros/ros.h"
#include "std_msgs/String.h"

int main(int argc, char **argv) {
    ros::init(argc, argv, "hello_robots");
    ros::NodeHandle n;
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter",
        1000);
    ros::Rate loop_rate(10);

    while (ros::ok()) {
        std_msgs::String msg;
        msg.data = "Hello, ROS!";
        chatter_pub.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
    }

    return 0;
}
```

4.1 Edit `CMakeLists.txt`

Add the following lines to the `CMakeLists.txt` file:

```
add_executable(hello_robots src/hello_robots.cpp)
target_link_libraries(hello_robots ${catkin_LIBRARIES})
```

5 Compiling the Package

After creating the source files for your package, you need to compile it using `catkin_make` to generate the necessary executables and libraries.

```
$ cd ~/catkin_ws
$ catkin_make
```

6 Launching the Master Node

The ROS master node is a crucial part of ROS. It facilitates communication between different nodes. You can start the master node using the command `roscore`.

```
$ roscore
```

7 Running the Node

You can run the node you created using the `rosrun` command. In this case, you'd run `hello_robots` from the `tp_rob` package.

```
$ rosrun tp_rob hello_robots
```

8 Testing Functionalities

In this section, we will explore additional functionalities in ROS.

8.1 Subscribing to Topics

To subscribe to a topic, you can create another node that listens for messages published on a specific topic. Here's an example code snippet:

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg) {
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
    ros::spin();
    return 0;
}
```

8.2 Using ROS Services

ROS services allow nodes to send a request and receive a response from another node. Here’s an example of defining and using a simple service:

```
#include "ros/ros.h"
#include "tp_rob/MyService.h"

bool myServiceCallback(tp_rob::MyService::Request &req,
    tp_rob::MyService::Response &res) {
    // Process the request and fill the response
    res.result = req.input + 5;
    return true;
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "service_server");
    ros::NodeHandle n;
    ros::ServiceServer service = n.advertiseService("my_service",
        myServiceCallback);
    ros::spin();
    return 0;
}
```