TECHNICAL REPORT:

Getting Started with ROS Programming

I. Introduction

Paket ROS adalah unit dasar program ROS. Kita dapat membuat paket ROS, membangunnya, dan merilisnya ke publik. Distribusi ROS yang kami gunakan saat ini adalah Noetic Ninjemys. Kami menggunakan sistem build catkin untuk membangun paket ROS. Sistem build bertanggung jawab untuk menghasilkan target (yang dapat dieksekusi/pustaka) dari kode sumber tekstual yang dapat digunakan oleh pengguna akhir. Dalam distribusi lama, seperti Electric dan Fuerte, rosbuild adalah sistem pembangunannya. Karena berbagai kekurangan rosbuild, muncullah catkin. Hal ini juga memungkinkan kami untuk memindahkan sistem kompilasi ROS lebih dekat ke Cross Platform Make (CMake). Ini memiliki banyak keuntungan, seperti mem-porting paket ke OS lain, seperti Windows. Jika OS mendukung CMake dan Python, paket berbasis catkin dapat di-porting ke OS tersebut.

II. Instructions

1. Creating ROS Package

The first requirement for working with ROS packages is to create a ROS catkin workspace. After installing ROS, we can create and build a catkinworkspace called

catkin_ws:

mkdir -p ~/catkin_ws/src

To compile this workspace, we should source the ROS environment to get access to ROS

functions:

source /opt/ros/noetic/setup.bash

Switch to the source src folder that we created previously:

cd ~/catkin_ws/src

Initialize a new catkin workspace:

catkin_init_workspace

We can build the workspace even if there are no packages. We can use the following

command to switch to the workspace folder:

cd ~/catkin_ws

The catkin_make command will build the following workspace:

catkin_make

This command will create a devel and a build directory in your catkin workspace. Different setup files are located inside the devel folder. To add the created ROS workspace to the ROS environment, we should source one of these files. In addition, we can source the setup file of this workspace every time a new bash session starts with the following command:

echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc

source ~/.bashrc

After setting the catkin workspace, we can create our own package that has sample nodes to demonstrate the working of ROS topics, messages, services, and actionlib. Note that if you haven't set up the workspace correctly, then you won't be able to use any ROS commands. The catkin_create_pkg command is the most convenient way to create a ROS package. This command is used to create our package, in which we are going to create demos of various ROS concepts. Switch to the catkin workspace's src folder and create the package by using the following command**:**

catkin_create_pkg package_name [dependency1] [dependency2]

Source code folder: All ROS packages, either created from scratch or downloaded from other code repositories, must be placed in the src folder of the ROS workspace; otherwise, they won't be recognized by the ROS system and be compiled. Here is the command for creating the sample ROS package: **catkin_create_pkg mastering_ros_demo_pkg roscpp std_msgs actionlib actionlib_msgs**

2. **Working with ROS Topics**
    A. **Creating ROS Nodes**

        The first node we are going to discuss is demo_topic_publisher.cpp and demo_topic_subscriber.cpp. Copy the current code into a new package or

use this existing file from this book's code repository. The following code: Robotika/UAS-Robotika at main · chismack/Robotika (github.com)

B. Adding Custom .msg and .srv files

We will start with message definitions. Message definitions must be written in the .msg file and must be kept in the msg folder, which is inside the package. We are going to create a message file called demo_msg.msg with the following definition:

string greeting

int32 number

So far, we have only worked with standard message definitions. Now, we have created our own definitions, which means we can learn how to use them in our code.

The first step is to edit the package.xml file of the current package and uncomment the **<build_depend>message_generation</build_depend> and <exec_ depend>message_runtime</exec_depend>** lines. Edit the current CMakeLists.txt file and add the message_generation line, as follows:

Robotika/UAS-Robotika at main · chismack/Robotika (github.com)

3. **Working with ROS Services**

In this section, we are going to create ROS nodes, which can use the service definition that we defined already. The service nodes we are going to create can send a string message as a request to the server; then, the server node will send another message as a response. Navigate to mastering_ros_demo_pkg/src and find the demo_service_ server.cpp and demo_service_client.cpp nodes. demo_service_server.cpp is the server, and its definition is as follows: Robotika/UAS-Robotika at main · chismack/Robotika (github.com)

4. **Creating Launch File**

Let's start by creating the launch files. Switch to the package folder and create a new launch file called demo_topic.launch to launch two ROS nodes for publishing and subscribing to an integer value. We will keep the launch files in

the launch folder, which is inside the package: Robotika/UAS-Robotika at main · chismack/Robotika (github.com)