Lab 2 - Robot Operating System: Part A

2.1 Workspace setup for Gazebo simulator

Both Ubuntu 18.04 and ROS Melodic have already been installed in the PCs in the lab. You should open a terminal using the keyboard input (Ctrl + Alt + T), in the home directory. Then typing the following commands one by one to prepare the repository:

```
mkdir ros_workspace
cd ros_workspace
source /opt/ros/melodic/setup.bash
mkdir -p src
cd src
catkin_init_workspace
cd ..
catkin_make
```

Above commands should execute without any warnings or errors.

Clone this repository to your workspace:

```
cd src
git clone https://github.com/husarion/rosbot_description.git
```

Install dependencies in *~/M-Drive/ros_workspace*

```
cd ..
rosdep update
rosdep install --from-paths src --ignore-src -r -y
```

Build the workspace in *~/M-Drive/ros_workspace*

```
catkin_make
```

Now your workspace is set up and ready for creating new nodes. You should have three new folders in your workspace: *build* for storing files that are used during compilation, *devel* for storing output files, and *src* for application source files.

Now you can run Gazebo simulator. Please remember that each time you open a new terminal window, you will need to load system variables:

```
source ~/ros_workspace/devel/setup.sh
```

2.2 Creating new package

In ROS, nodes are distributed in packages. To create a node, you need to create a package. Packages are created with command *catkin_create_pkg*, which must be executed in src folder in your workspace. For example, we will create package named *tutorial_pkg* which depends on package *roscpp*. Note that the package *roscpp* is a basic ROS library for C++. Typing the following command:

cd ~/ros_workspace/src
catkin_create_pkg tutorial_pkg roscpp

This will create folder named *tutorial_pkg* and some files in it. Your workspace file structure should now look like below.

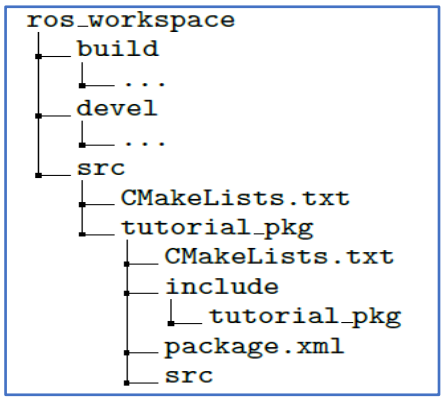

Figure 1.1 Workspace file structure.

Created files are:

- *CMakeLists.txt* - these are *build* instructions for your nodes. You need to edit this file if you want to compile any node, we will do it later.
- *package.xml* - this file contains package metadata like author, description, version, etc. Note that you should adjust this file if you want to publish your package to others.

2.3 Starting Gazebo

To start Gazebo simulator, you should open a terminal and then type the following commands:

cd ~/ros_workspace

source devel/setup.sh

roslaunch rosbot_description rosbot.launch

Now you should see the ROSbot in the Gazebo, similar to the picture below.

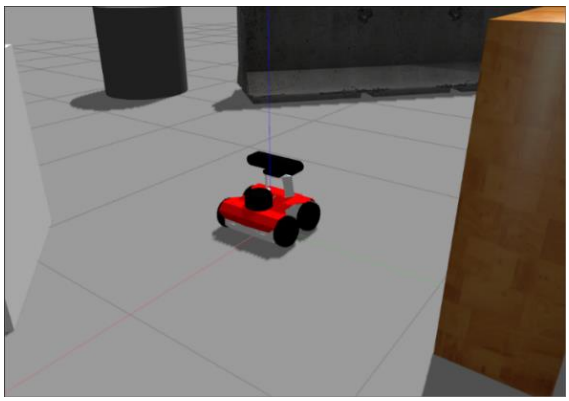

Figure 1.2 ROSbot runs in the Gazebo simulator

2.4 Write your code

Using the following command to create your 1st node, namely *tutorial_pkg_node.cpp* and place it in *src* folder under *tutorial_pkg:*

touch ~/ros_workspace/src/tutorial_pkg/src/tutorial_pkg_node.cpp

Using your favourite text editor to open it and paste the following:

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
using namespace std;

class RobotMove {      //main class
public:
// Tunable parameters
        constexpr const static double FORWARD_SPEED_LOW = 0.1;
        constexpr const static double FORWARD_SPEED_HIGH = 0.2;
        constexpr const static double FORWARD_SPEED_SHIGH = 0.4;
        constexpr const static double FORWARD_SPEED_STOP = 0;
        constexpr const static double TURN_LEFT_SPEED_HIGH = 1.0;
        constexpr const static double TURN_LEFT_SPEED_LOW = 0.3;
        constexpr const static double TURN_RIGHT_SPEED_HIGH = -2.4;
        constexpr const static double TURN_RIGHT_SPEED_LOW = -0.3;
        constexpr const static double TURN_RIGHT_SPEED_MIDDLE = -0.6;
        RobotMove();
        void startMoving();
        void moveForward(double forwardSpeed);
        void moveStop();
        void moveRight(double turn_right_speed = TURN_RIGHT_SPEED_HIGH);
        void moveForwardRight(double forwardSpeed, double turn_right_speed);
private:
        ros::NodeHandle node;
        ros::Publisher commandPub;// Publisher to the robot's velocity command topic
};

RobotMove::RobotMove(){
//Advertise a new publisher for the simulated robot's velocity command topic at 10Hz
        commandPub = node.advertise<geometry_msgs::Twist>("cmd_vel", 10);
}

//send a velocity command
void RobotMove::moveForward(double forwardSpeed){
        geometry_msgs::Twist msg;//The default constructor to set all commands to 0
        msg.linear.x = forwardSpeed; //Drive forward at a given speed along the x-axis.
        commandPub.publish(msg);
}

void RobotMove::moveStop(){
        geometry_msgs::Twist msg;
        msg.linear.x = FORWARD_SPEED_STOP;
        commandPub.publish(msg);
}

void RobotMove::moveRight(double turn_right_speed){
        geometry_msgs::Twist msg;
        commandPub.publish(msg);
```

```
}

void RobotMove::startMoving(){
        ros::Rate rate(20);   //Define rate for repeatable operations.
        ROS_INFO("Start moving");
        // keep spinning loop until user presses Ctrl+C
        while (ros::ok()){
            // Check if ROS is working. If ROS master is stopped or there was sent signal
            // to stop the system, ros::ok() will return false.
            moveForward(FORWARD_SPEED_LOW);
            ROS_INFO_STREAM("Robot speed: " << FORWARD_SPEED_LOW);
            ros::spinOnce(); // Allow ROS to process incoming messages
            rate.sleep();   // Wait until defined time passes.
        }
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "RobotMove");//Initiate new ROS node named "RobotMove"
    RobotMove RobotMove;            // Create new RobotMove object
    RobotMove.startMoving();        // Start the movement
    return 0;
}
```

2.5 Building your node

Go to *~/ros_workspace/src/tutorial_pkg* directory, you need to edit *CMakeLists.txt* before building the node. Open it using your favourite text editor.

Find line: *# add_compile_options(-std=c++11)*

and uncomment it (remove # sign). This will allow to use C++11 standard of C++.

You should also find and uncomment line:

```
# add_executable(${PROJECT_NAME}_node src/tutorial_pkg_node.cpp)
```

This will enable the compiler to create executable file from the defined source, which will be your node. Variable *PROJECT_NAME* is defined by line *project (tutorial_pkg)*. This results in *tutorial_pkg_node* as the name of the executable. You can adjust it to your needs.

Then, you should find and uncomment the following lines:

```
#  target_link_libraries(${PROJECT_NAME}_node
#  ${catkin_LIBRARIES}
# )
```

The compiler links libraries required by your node, save the changes and close editor.
Open the terminal, move to the main directory:

```
cd ~/ros_workspace
```

and build your project using

```
catkin_make
```

You should get output like Figure 1.3.

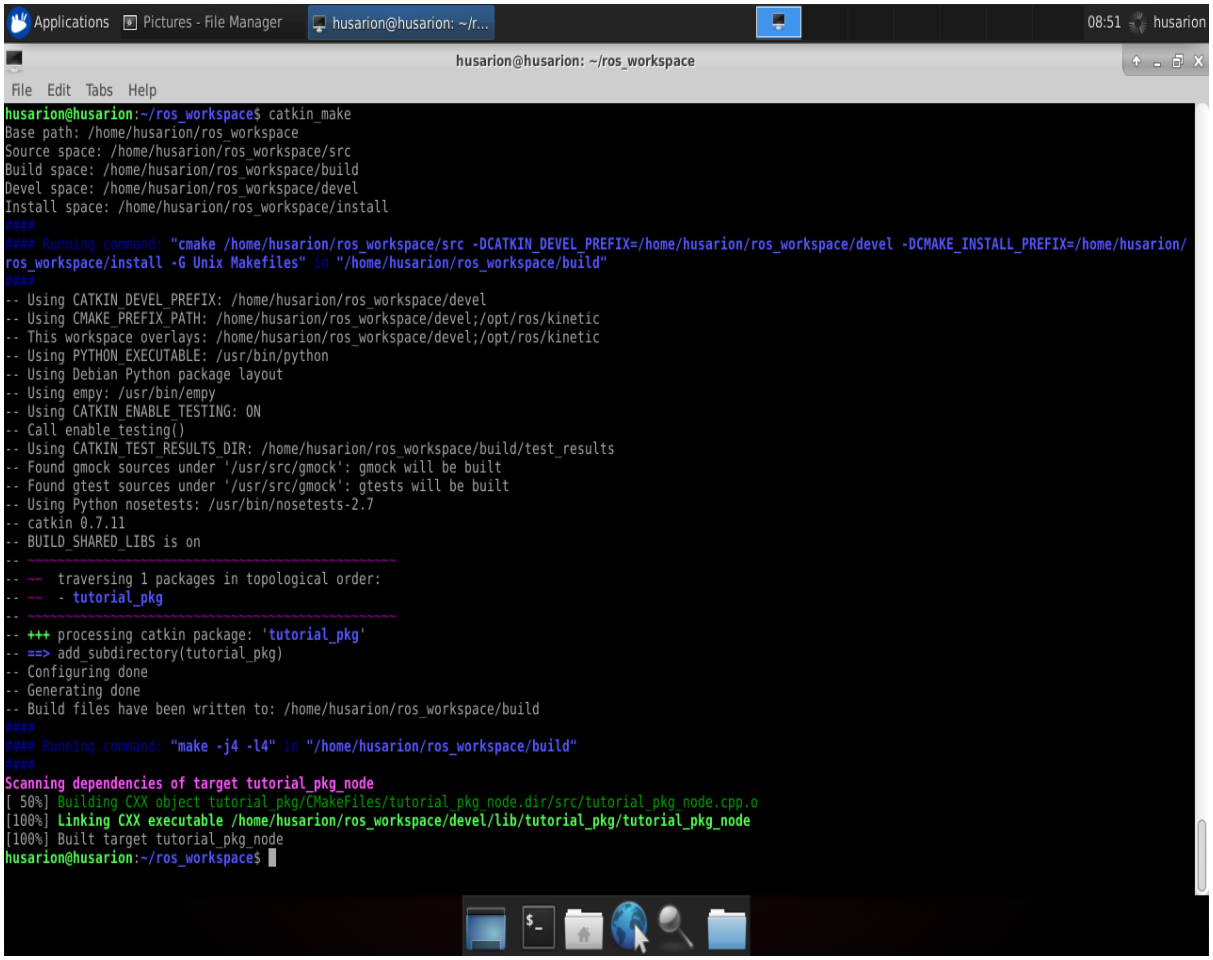

Figure 1.3 Build the project using '*catkin_make*' tool

2.6 Run your node

Your node is built and ready for running by now. However, you should type the following command to load environment variables to run the node no matter which directory you are in:

source ~/ros_workspace/devel/setup.sh

You must load it every time you open new a terminal, unless you have added it into your *.bashrc* file so that every time you open a new terminal, the environment variables will be loaded automatically.

Task 2.1 To run your node, you can use command line or *.launch* file. Remember that package is *tutorial_pkg* and node is *tutorial_pkg_node*. Then use *rosnode* and *rqt_graph* tools to examine system and check if your node is visible in the system.

Using command line

In terminal 1: *roscore*

In terminal 2: *rosrun tutorial_pkg tutorial_pkg_node*

You will see the data output in terminal 2. Note: You can use "Ctrl-C" to stop it.

Using *.launch* files

You should type the following command to create *launch* directory:

mkdir ~/ros_workspace/src/tutorial_pkg/launch

Typing the following command to create a launch file *tutorial_pkg_node.launch*:

touch ~/ros_workspace/src/tutorial_pkg/launch/tutorial_pkg_node.launch

Using *gedit* text editor to open the launch file and paste the following:

```
<launch>
    <node pkg="tutorial_pkg" type="tutorial_pkg_node"
    name="tutorial_pkg_node" output="screen">
     </node>
</launch>
```

Save it in *~/M-Drive/ros_workspace/src/tutorial_pkg/launch* directory and launch it:

roslaunch tutorial_pkg tutorial_pkg_node.launch

Using *rosnode* *and* *rqt_graph* tools to examine the system

In terminal 1: you can use the following command to launch *tutorial_pkg*:

cd ~/ros_workspace/src/tutorial_pkg/launch

roslaunch tutorial_pkg tutorial_pkg_node.launch

In terminal 2: you can use the following command to see active nodes

cd ~/ros_workspace/src/tutorial_pkg/launch

rosnode list

Using the following command to retrieve information about the *tutorial_pkg_node:*

rosnode info tutorial_pkg_node

Then using the following graphical tool for visualization of data flow across nodes.

rqt_graph

Then you should see a GUI graph that shows data flow among ROS nodes and topics.

2.7 Further References

ROS Wiki	*http://wiki.ros.org/*
Installation	*http://wiki.ros.org/ROS/Installation*
Tutorials	*http://wiki.ros.org/ROS/Tutorials*
Available packages	*http://www.ros.org/browse/*

ROS Cheat Sheet

https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index

ROS Best Practices

https://github.com/leggedrobotics/ros_best_practices/wiki

ROS Package Template

https://github.com/leggedrobotics/ros_best_practices/tree/master/ros_package_template