Technical report:

Motion Control on TurtleBot

By: Kevin Descharrieres, Antoine Merlet

Professor: Dr. Ralph Seulin

May 3, 2017

Contents

I. BRAINSTORM

video pict command line python screenshots: consols -¿ what is started on which computer(ssh) rviz : Part1 Goal 4, Part 2, Part 3

PART 1

BACK AND FORTH mostly from ROS by example Vol. 1 Simple stuff, but explain at least once the process (script -¿ launch) in detail, and then skip it forever. Here we can see that the robot does not end at the starting position. In fact, the twist topic is not sufficient by itself to properly drive the robot, as is just emit Linear and Angular velocities and does not give any feedback. We need to add more sensing to make it viable ˜BACK AND FORTH WITH ODOMERTY Odometry is our savior. It allows us to get information on the internal motion of the robot, and therefore check if the Twist messages are well applied.

II. PRESENTATION OF THE ROBOT

FIG. 1: Kobuki Base

The first version of the TurtleBot was created in 2011 by Willow Garage society. This robot hardware, based on a vacuum cleaner, has been improved over the time and is now considered as a pillar of the robotic field. The aim of this robot is to make it affordable for a beginner in the field while being powerful enough for high-end applications. Moreover, the TurtleBot OS is based on the middle-ware ROS (Robot Operating System), making it a really nice tool as ROS is becoming more and more popular. By now, the TurtleBot became THE reference in education and research.

As any first version of a hardware/software, the TurtleBot could be improved, giving therefore the TurtleBot 2 (the one we are equipped with). Following are presented the main features of this robot:

- A comfortable operating time (2-3 hours);
- A Kinect for 3D sensing;
- A fast charging Dock for idling charge;
- A charging cable in order to charge the base while using it;

- Cliff and drop sensors;
- A laptop;
- And 3 collision sensors.

FIG. 2: TurtleBot 2

For more technical details about the Kobuki base, please refer to [1] (link it to the doc). Most all the work has been done using a very complete book: ROS by example, volume 1, R. Patrick Goebel.

III. HOW TO CONTROL A TURTLEBOT

A. Workspace and network

There are mainly two ways of controlling the TurtleBot. The first option is to directly control the TurtleBot using the laptop connected to it. However, this is inconvenient as the TurtleBot might be moving, but also because the user does not have any comfortable position to use the laptop.

The second option is to remotely connect to the TurtleBot computer. This can be easily done with Ubuntu by using the SSH build-in command. Also, please notice that the network is already optimized in our case (singleton of the couple Workspace+TurtleBot per router), making the SSH really easy on our side. Therefore, you can distinguish two main components: the TurtleBot+laptop and the Workstation. This allows the control of the TurtleBot in an efficient (and comfortable) way. The only thing that is to be done to enable the remote control is to modify the bashrc file of the workstation (located in the Home folder in most of the cases, can be edited by using the command *$ gedit ~ /.bashrc*. The three last lines ('export') need to be uncommented. The first one should contain the IP address of the TurtleBot on your local network (most-likely starting by 192.168.XXX.XXX). The two second line should be filled with the IP of your Workstation. In order to make it easy for daily connection, we suggest that both the Workstation and the TurtleBot laptop are provided static IP addresses.

In all the following, we will specify one which one should be started each command line. If not specified, the main computer (the workstation) should be considered as the default one.

B. Topics and nodes

A node represent a process, doing calculations. With ROS, each node is dedicated to a specific take. For example, one node manages the lidar, another on the Kinect, one the movement of the wheels, and so on. In order to communicate, Nodes are using Topics, by publishing messages into it. This topic has a specific name representing the type of data that it is carrying (the topic "joy" carries information about the joystick input). Node are not only publishing to topics, they can also subscribe to them, so that when a new topic publishes, they receive this data. This concepts are simple as long as the application is basic, otherwise, it is really complicated to figure out which node is publishing to which topic. In order to have a price idea of what is going on at a low level, a very powerful tool can be used: the rqt_graph (presented later).

C. Package

A way to control the TurtleBot is to create a package. This allows to store several behaviours for the TurtleBot. In order to create such a package, one could either use the command *$*

catkin_create_pkg <package _name>[depend1] [depend2] [depend3] by replacing the given parameters by the wanted name and dependencies. It is also possible to create manually a package from scratch, mostly for advanced users with a wide knowledge of ROS. In our case, we used the command *$ catkin_create_pkg project rospy std_msgs*. Therefore, the package is named 'project' and is using two external libraries. Rospy is used to be able to create and run python scripts with the TurtleBot, and std_msgs allowing to use standard data-types. The package also comes with a file named 'CMakeLists.txt', which we are not changing. The package also comes along with folders, such a scripts and launch files.

D. Script

In order to create your own routines for the robot, Scripts are used. They are mainly written in Python or C++ language, which are widely used in IT science, making it somewhat easy to create. Following is presented the basic commands to create a Python script for ROS.

```
# Always import rospy for the script to work
import rospy

# Initializing a Self-created node called MyNode
rospy.init_node('MyNode')

# Declaring a new publisher to the Topic /cmd_vel.
self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=1)

# Initialize the movement command that will be emitted to the publisher
move_cmd = Twist()

# Publishing the move_cmd message to the publisher (being /cmd_vel)
self.cmd_vel.publish(move_cmd)

# Putting the robot in sleep mode (usually when the script is done)
rospy.sleep(1)
```

Once the TurtleBot is started with a ROScore, it is possible to start scripts. However, having to start the ROScore with all the scripts needed for the experiment can be a bit tedious. Launch files can overcome this problem.

E. Launch files

Script file are simple to use, but not very convenient. Launch files bring the possibility to start several scripts and launch file as well as allowing to use custom settings. The following example show all of this.

```
<launch>
    <!-- Will use the launch File for the joystick-->
    <include file="$(find turtlebot_teleop)/launch/logitech.
       launch"/>

    <!-- Create a Node instance joy_Node -->
    <node pkg="joy" type="joy_Node" name="turtle_joy" >
        <!-- Setting specific Parameters of the instance-->
        <param name="deadzone" value="0.12" />
        <param name="autorepeat_rate" value="2" />
    </node>

    <!-- Create a Node instance according to a Script File.
       the Args are use to redirect the publication to the /
       cdm-topic to another topic (cmd not wortking on real
       robot)-->
    <node name="Joy2twist" pkg="joy2twist" type="NewJoy2Twist.
       py" args="cmd_vel:=cmd_vel_mux/input/teleop" />
</launch>
```

As we can see on the first few lines, this launch file is starting another launch file. This included

file may or may not include again other launch files, et caetera. From this, we can obviously see a big source of problem: any time you there is a need to debug one application, all launch files must be examined recursively. It is not unusual to have around seven layers of launch file. In the launch files, it is also possible to set the values of parameters from the included nodes. Finally, it is also possible to remap some topics, transferring the published data to another topic.

F. Bag files

ROS is equipped with a very powerful tool to record all the topics and nodes while the robot is processing them: the bag files. It allows the user to save the robot state during an experiment so that it can be simulated later, or even re-run on the robot. We consider this tool really important, that is why we chose to provide a bag file corresponding to each script that we produced. To record a bag file with all the running topics, one can use the console command *$ rosbag record -a*. For more documentation about the rosbag command, please follow this link

IV. OUR PROJECT

A. Purpose

B. Simple motions, Twist and Odometry

C. Adding and controlling sensors

D. SLAM and ACML

V. MERGING TURTLEBOT AND PHANTOMX ARM PINCHER

VI. CHALLENGES ENCOUNTERED

A. Hardware malfunctions

battery, desync, kinect, arm, arduino board

B. Python

unknown libraries

C. Debugging the TurtleBot

VII. CONCLUSION

VIII. REFERENCES

IX. APPENDIX

X. OLD STUFF

The first move is the come-and-back which is a low level programming.

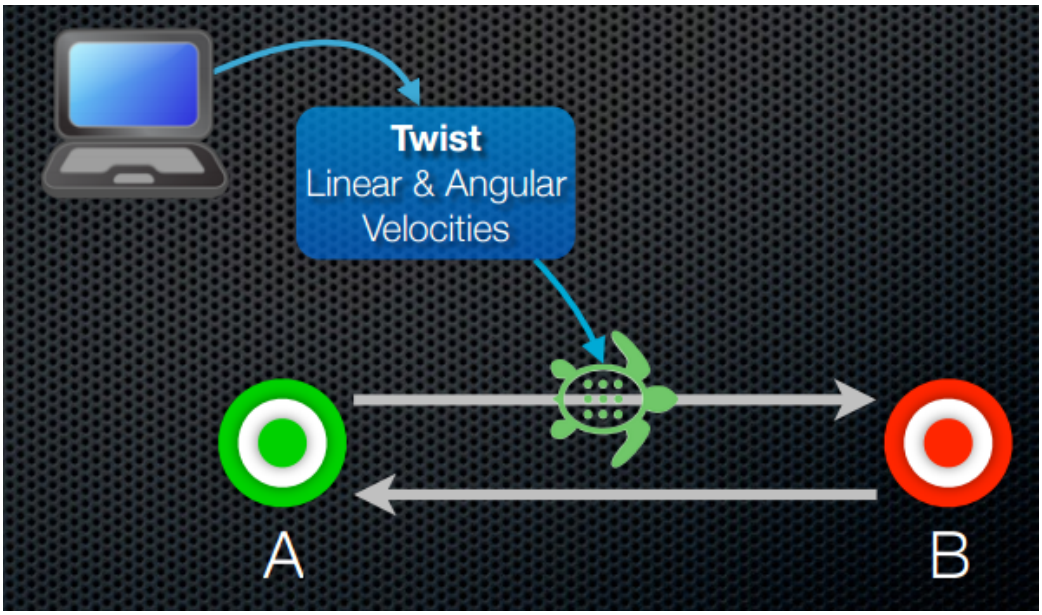

FIG. 3: Come and Back movement

The second one is the same movement as previously using the odometry data.

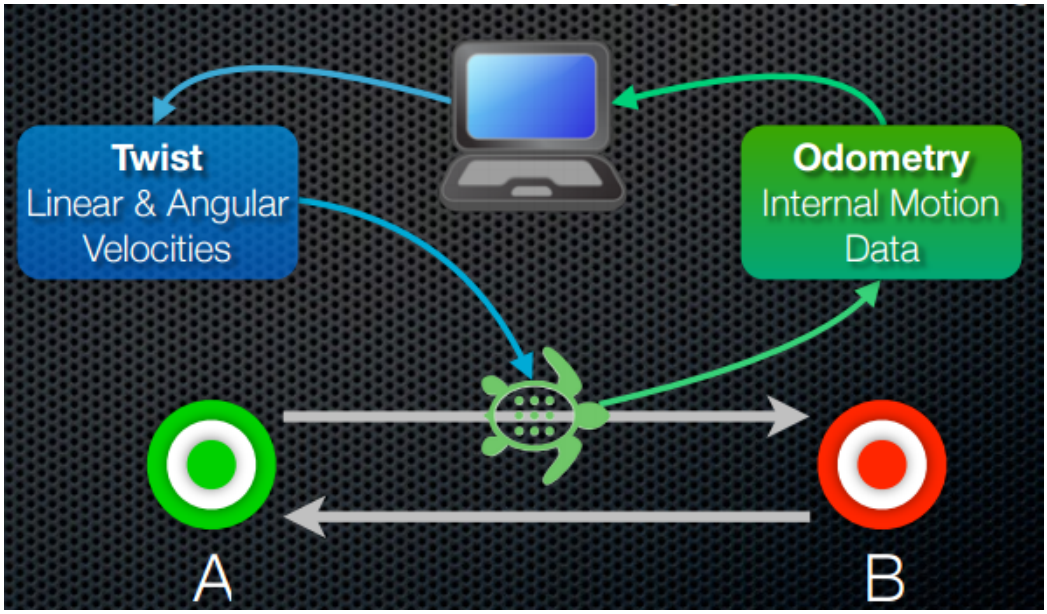

FIG. 4: Come and Back movement using odometry

The third movement is navigating a square using twist and odometry.

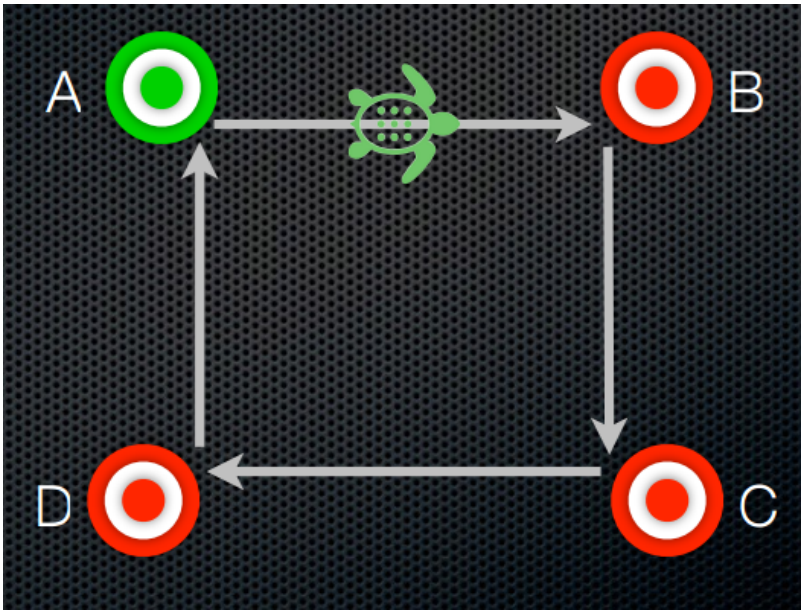

FIG. 5: Square Movement

The forth one is to follow a curve between a starting point and a target and navigation with path planing.

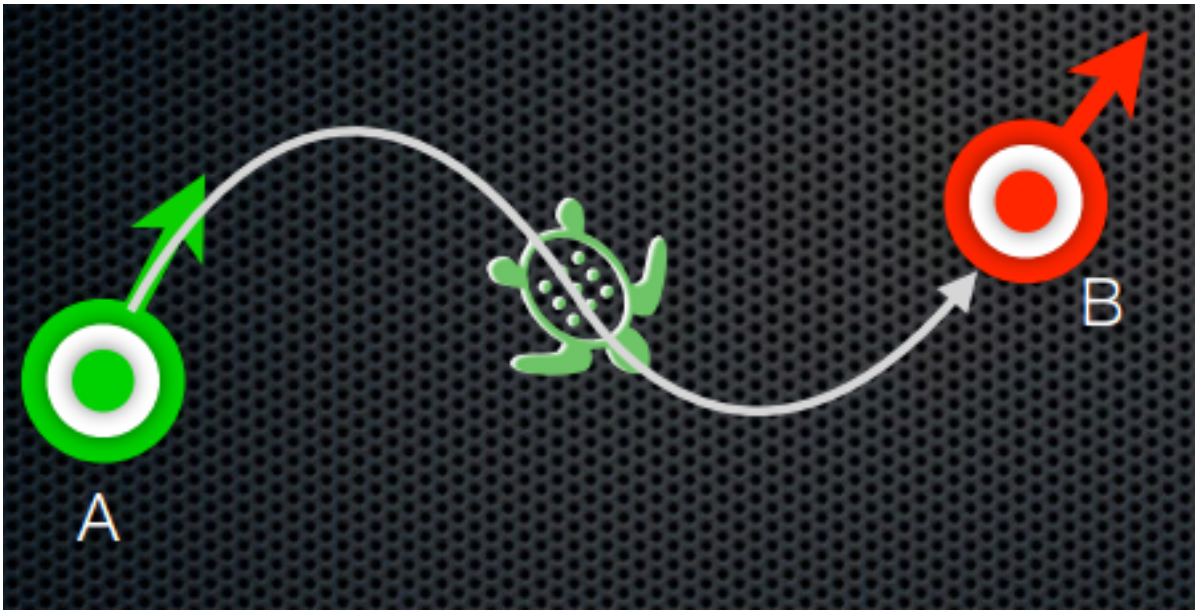

FIG. 6: Curve Movement

XI. CHALLENGES

A. Twist

The first main problem in this exercise was how to twist the TurtleBot in real time. For that, we had in a first time try to give it a rotation angle, for example 90 degrees, which is an angle easily recognizable. We saw quickly the problem, the describe angle was never 90 degrees, it was between 80 and 105 degrees. During the come-and-back demonstration, the move was great except the exact angle during the twist. This exercise demonstrated the fact that program a robot by giving it movement angle is not a good way if we want a precise result at the angle. Furthermore, it is useless to try to correct the angle by hand in the program because the describe angle will not be tow times the same, it changed at every step.

B. Odometry

C. Basic Moves

XII. ROSIFICATION

A.

B.