Mini Project 3

Seyedeh Diba Ravanshid Shirazi
School of Electrical and Computer Engineering (ECE)
University of Tehran
Tehran, Iran
Student Number: 810199431

CONTENTS

***Abstract*—**

***Index Terms*—**

I. INSTALLATION AND TUTORIAL

for installing linux on my laptop first I had to download *Oracle*, a virtual machine, which needed vpn for downloading and took me a while to get this done. Then I pressed New and downloaded Ubuntu 20.04. Then I allocated the hardware and hard disk to my virtual machine. Then for starting this system I needed to press start. and setup my linux system. For installing ROS I followed the instructions and had no problem. It asked me to enter my password a lot more than it is shown in the videos. It took me 1 hour to install ROS on my system. After watching the first video I learned how to make files and folders, how to change directories, change permissions of each file. After the second video I learned how to use Ros and how to make directories. First we need to source the setup.bash file. Then we make a directory named catkin ws/src using mkdir command. then we change directory to catkin ws using cd command. After that we write command : textitcatkin make to compile everything. Now we source *devel/setup.bash*. We sun roscore which runs the master node. Now we run *rosrun turtlesim node.* We learn about messages, topics and .We have publisher and subscriber nodes, server and client. For working with these nodes first we have to run *catkin create pkg* and name the package. After going in the package and the src dir, we create the python files and give it access. Also i needed to download Visual Studio to write python codes. Then we write the codes and run them with rosrun package name code.py

A. Error

The only error I had was me having typos. Also working with nano and vim for making files was not that easy and took me so long to understand. While writing the codes in the third video about ROS I could not import the demo pkg.msg; for example, when writing *from demo pkg.msg import demo msg* the text did not appear green. I tried to solve the problem by asking chatgpt but i could not.In the codes here I did not need use this concept. Once i ran a code with the first line being : *!usr/bin/python* and because it was not python3 i got an error. Another time i ran the codes and kept getting error so I realized i didn't source the package correctly.

II. PART 1: DRAW CHARACTER WITH TURTLES!

Turtlesim has a 11*11 page scale and here we need to spawn the turtles in the correct position to form the desired characters. first we define the places to put the turtles by drawing it here:

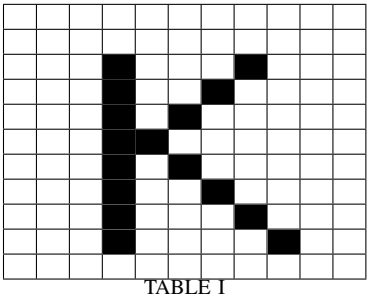

TABLE I
AN 11 BY 11 TABLE FORMING THE SHAPE OF THE LETTER "K"

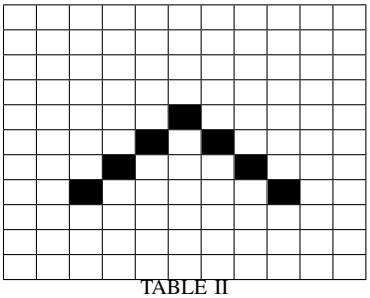

TABLE II
AN 11 BY 11 TABLE FORMING THE SHAPE OF THE CHARACTER CARET.

For making the python codes and create the packages and folders there is explanation in the first section. Let's explain the code used here. First when running the turtlesim there is

a turtle that needs to be killed And then we're going to spawn all the turtles in the code. So we need two functions:

A. Functions

- **kill turtle(name)**: This function is designed to remove a turtle by name from the simulator. It waits for the 'kill' service to be available, then creates a service proxy to make the service call to remove the turtle. If the service call fails, an error is logged.
- **spawn turtle(x, y, theta, name)**: This function spawns a new turtle at specified coordinates with a given name. It waits for the 'spawn' service to be available, creates a service proxy for the service call, and logs an error if the service call fails.

B. Main Function explanation

- **Node Initialization**: The ROS node is initialized with the name *turtle spawner*. This allows the script to interact with the ROS system under this node name.
- **Removing the first default Turtle**: The function *kill turtle* is called with the name *turtle1* to remove the default turtle that appears when *turtlesim* starts. This ensures the space is clear.
- **Spawning Turtles**: It calls *spawn turtle* multiple times with specific coordinates to place turtles in positions that form the shape of the letter K or caret. Each call specifies the x and y coordinates and the turtle's name.
- **Logging Information**: A log message confirms that the turtles have been successfully spawned.

The entire *main* function is enclosed in a try-except block to catch and handle any unexpected errors during execution and make it easier to debug. Following the code done in the third video, we wrote these functions and named the code: spwaner.py. Here is how we run it and what it outputs:

```
diba@diba-VirtualBox:~/catkin_ws/src/demo_pkg/src$ rosrun demo_pkg spawner.py
Everything's ok!
[INFO] [1718216185.881833]: Turtles spawned.
```

Fig. 1. Output of spawner.py

We run the code two times and get the following outputs:

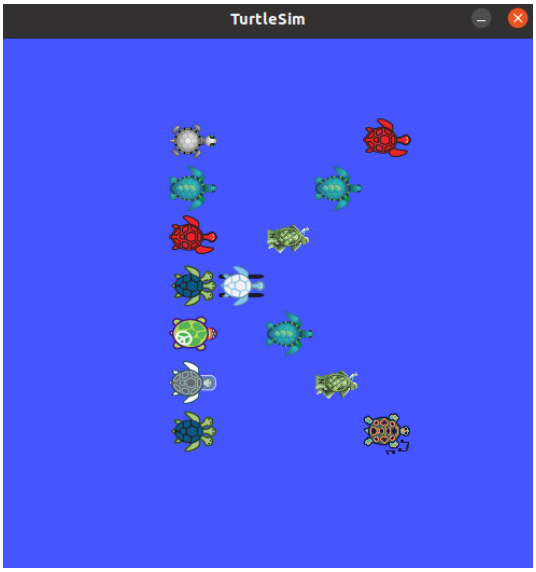

Fig. 2. K letter with turtles

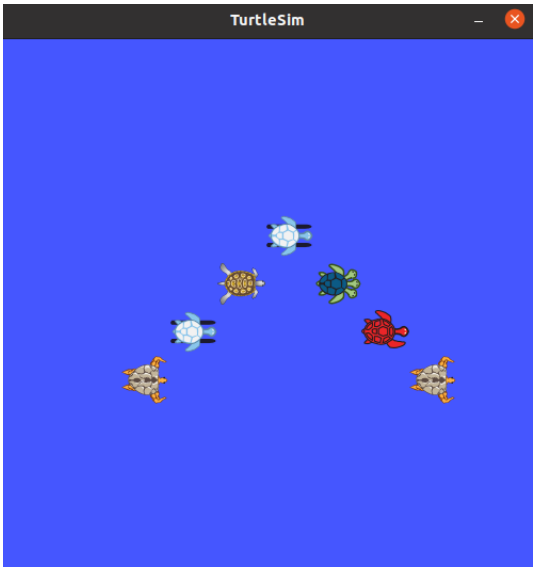

Fig. 3. caret character with turtles

We have successfully spawned all the turtles.

C. Error

There were no errors for this part because i could follow along with the video. The code has a commented part that can be uncommented for drawing a different character, which i think can be implemented in a better way.

```
diba@diba-VirtualBox:~/catkin_ws/src/demo_pkg/src$ rostopic
rostopic is a command-line tool for printing information about ROS Topics.

Commands:
        rostopic bw     display bandwidth used by topic
        rostopic delay  display delay of topic from timestamp in header
        rostopic echo   print messages to screen
        rostopic find   find topics by type
        rostopic hz     display publishing rate of topic
        rostopic info   print information about active topic
        rostopic list   list active topics
        rostopic pub    publish data to topic
        rostopic type   print topic or field type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

Fig. 4. rostopic in command output

For running the rostopic list, we have to open the turtlesim and spawn the turtles first. If we don't do that we will only get rosout command available.

III. Part2: Rotating the turtles!

First as the instruction says we have to see the rostopic list. first if we type rostopic itself we will get a list of all the command we can give each turtle after its name:

```
diba@diba-VirtualBox:~/catkin_ws/src/demo_pkg/src$ rostopic list
/k1/cmd_vel
/k1/color_sensor
/k1/pose
/k10/cmd_vel
/k10/color_sensor
/k10/pose
/k11/cmd_vel
/k11/color_sensor
/k11/pose
/k12/cmd_vel
/k12/color_sensor
/k12/pose
/k13/cmd_vel
/k13/color_sensor
/k13/pose
/k14/cmd_vel
/k14/color_sensor
/k14/pose
/k2/cmd_vel
/k2/color_sensor
/k2/pose
```

Fig. 5. rostopic list

So here we see that we have cmd-vel topic after the turtle's name to control its velocity. In geometry-msgs we have twist. So as it shows we will be needing to find the turtle's names and also give them the message to rotate in a loop so they won't stop untill we press ctrl+c. Here is the detailed explanation of the code:

- **Initialization**: The script initializes a ROS node named 'turtle_rotator' using *rospy.init_node*..
- **Publisher Setup**: For each turtle, a publisher (*rospy.Publisher*) is set up to the *"/turtle_name/cmd_vel"* topic. This allows sending velocity commands to control the turtles' movements.
- **Velocity Commands**: A *Twist* message is created with *angular.z* set to 1.0, which determines the counter-clockwise angular velocity of the turtles.
- **Continuous Rotation**: Inside a *while* loop that runs as long as the ROS node is not shut down, the script continuously publishes the *Twist* message to all turtles' *cmd_vel* topics. The rate of publishing is controlled by a *Rate* set to 10 Hz.
- **Stop**: The script runs indefinitely, rotating the turtles until manually stopped by terminating the ROS node.

A. Main

The main function has the list of turtle names. It then calls the *rotate_turtle* function.

B. Bonus

For accessing keyboard inputs we need *pynput* library. For using pip install first we need to install pip. Then we install pynput(There were some errors that is discussed in error subsection after this subsection.):

C. Error

For installing pynput i think there is an network issue that can be solved using proxy or vpn. I tried to change the servers in software and update part in ubuntu, but it didn't work. I changed the proxies in setting of windows but it also didn't work.

```
diba@diba-VirtualBox:~/catkin_ws/src/demo_pkg/src$ pip install pynput
Collecting pynput
  WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, sta
tus=None)) after connection broken by 'ReadTimeoutError("HTTPSConnectionPool(ho
st='files.pythonhosted.org', port=443): Read timed out. (read timeout=15)")': /
packages/ef/1d/fdef3fdc9dc8dedc65898c8ad0e8922a914bb89c5308887e45f9aafaec36/pyn
put-1.7.7-py2.py3-none-any.whl
  WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, sta
tus=None)) after connection broken by 'ReadTimeoutError("HTTPSConnectionPool(ho
st='files.pythonhosted.org', port=443): Read timed out. (read timeout=15)")': /
packages/ef/1d/fdef3fdc9dc8dedc65898c8ad0e8922a914bb89c5308887e45f9aafaec36/pyn
put-1.7.7-py2.py3-none-any.whl
  WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None, sta
```

Fig. 6. Install pynputs ran to error.

IV. Part3: Launch files

Here we need a launch file with.launch that helps us run all the codes together. I made some typo mistakes. But it was easy. You should just run the turtlesim-node at first and them the py files that you wrote. After launching this file the letter would appear and the turtles will start rotating at the same time. Here are the nodes using rqt-graph

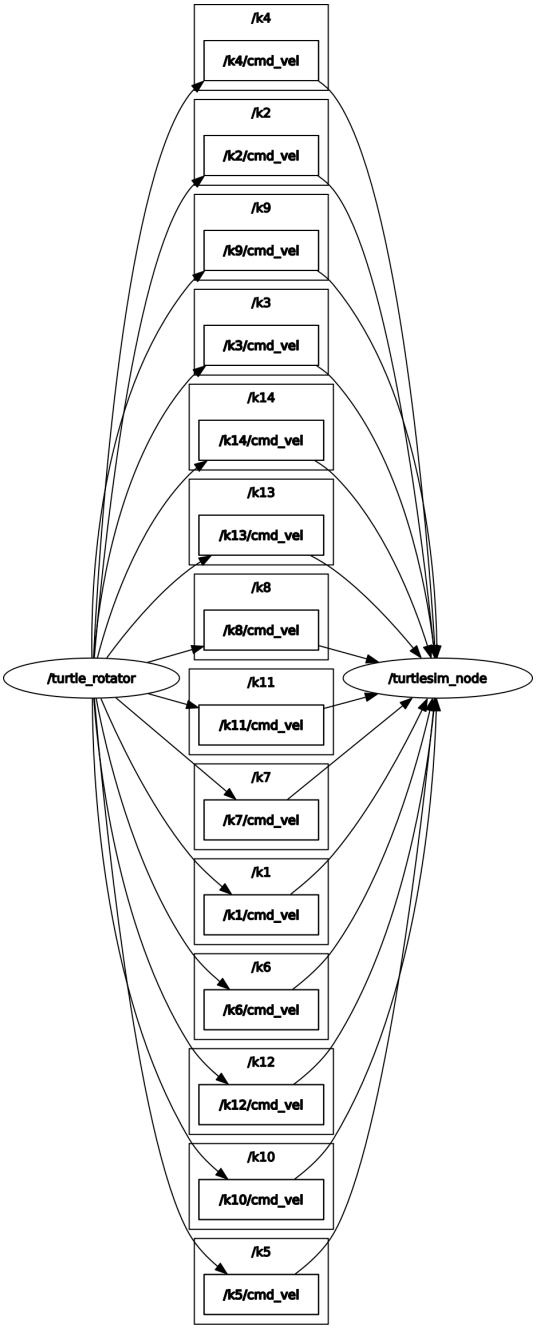

Fig. 7. The graph of all the nodes.

Thank you for your attention. This project took me 25 hours to complete including the times for installation and watching the videos.