Tekhnê
July - December 2021, Vol. 18, No. 2, pp. 19 – 24

ISSN 1692-8407

TurtleBot3 robot operation for navigation applications using ROS

Manejo del robot TurtleBot3 para aplicaciones de navegación mediante ROS

Fredy H. Martínez S.
Facultad Tecnológica, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia
fhmartinezs@udistrital.edu.co

Logo is a programming language that was born in 1967 as a tool for learning programming. Its concept was simple, assign commands to a virtual turtle to trigger its movement. TurtleBot is a robotics research platform with development based on this concept. However, this platform is both hardware and software oriented and is widely accepted internationally, particularly in SLAM (Simultaneous Localization And Mapping), navigation, and manipulation applications. In fact, it is the standard ROS (Robot Operating System) platform. This paper aims to show the handling and programming of this robot for navigation applications using ROS and Python.

Keywords: Learning, navigation, research, robotics, ROS, TurtleBot3

Logo es un lenguaje de programación que nació en 1967 como herramienta para de aprendizaje en programación. Su concepto era simple, asignar órdenes a una tortuga virtual a fin de provocar su movimiento. TurtleBot es una plataforma de investigación robótica con desarrollo basado en este concepto. Sin embargo, esta plataforma está orientada tanto al hardware como al software, y posee gran aceptación a nivel internacional, particularmente en aplicaciones de SLAM (Simultaneous Localization And Mapping), navegación y manipulación. De hecho, es la plataforma estándar de ROS (Robot Operating System). Este documento pretende mostrar el manejo y programación de este robot para aplicaciones de navegación con el uso de ROS y Python.

Palabras clave: Aprendizaje, investigación, navegación, robótica, ROS, TurtleBot3

Article typology: Research

Received: October 29, 2021
Accepted: November 11, 2021

Research funded by: Universidad Distrital Francisco José de Caldas (Colombia).

How to cite: Martínez, F. (2021). *TurtleBot3 robot operation for navigation applications using ROS*. Tekhnê, 18(2), 19 -24.

ISSN 1692-8407

Introduction

ROS (Robot Operating System) is a framework for study and research in robotics (Lentin, 2018; Moreno & Páez, 2017). The tool was developed in 2007 by the Stanford Artificial Intelligence Lab and is currently being developed and maintained by the Willow Garage with support from a broad group of institutions (Koubaa, 2020). As its name suggests, ROS provides standard services, just as an operating system does, related to hardware abstraction, low-level device control, implementation of commonly-used functionality, inter-process message passing, and package maintenance (Quigley et al., 2009). It is designed for UNIX systems, under BSD licensing terms, so it is widely used in Linux distributions such as Ubuntu, Fedora, and Arch, but it is also used in Mac OS X. The current version is called Melodic Morenia, which was released in 2018 and is supported until May 30, 2023.

TurtleBot is the standard ROS platform, and is the most popular hardware in research, particularly in motion planning strategies. The first prototype, TurtleBot1, was developed in 2010 from the development version of iRobot's Roomba. The second generation, TurtleBot2, was born two years later, in 2012 from the iClebo Kobuki platform (Hou et al., 2020; Poza-Lujan et al., 2019). In 2017 appears the current version, TurtleBot3, which solves the shortcomings of the previous versions and, like the previous versions, maintain full support with ROS (Groß, 2021).

The TurtleBot3 robot has great advantages for use in robotics, both in specific training and research (Al-Mashhadani et al., 2020; Fernandes et al., 2019). It is an affordable platform, widely used in research centers around the world, small and easy to adapt to specific needs, programmable in MatLab and Python, and as mentioned, fully compatible with ROS. Its small size does not sacrifice capability and performance but allows a highly competitive platform to be obtained with a small investment. Thanks to its 360-degree LiDAR (Light Detection And Ranging) sensor, this robot is a perfect choice for traditional SLAM applications in motion planning (Aslan et al., 2021; Ratul et al., 2021).

Work environment

Hardware overview

There are currently two versions of TurtleBot robots, which although quite similar, have some key differences between them. These are the Burger and Waffle models, the differences can be summarized in Table 1.

The Waffle model is slightly faster in forwarding motion and has a higher load capacity. However, it is much slower in turning, much larger has an additional camera, and is also considerably more expensive. For this reason, the research group opted for the Burger platform from which it acquired a couple of robots. The camera may be useful in our research, but it can be integrated into the Burger platform at a later date.

The most important hardware components of the TurtleBot3 Burger robot are:

- LiDAR sensor: The robot uses a 360-degree LDS-01 LiDAR sensor. It is a laser scanner capable of collecting distance data around the robot with a range of 120 mm to 3,500 mm, an angular resolution of 1 degree, with a sampling rate of 1.8 kHz; and allows USB connection for computer and UART for embedded systems. It is located on the top of the robot.

- Raspberry Pi development board: The robot uses a Raspberry Pi 3 Model B development board that is located one level below the LiDAR sensor. Although it does not have large processing power, it is key to implement autonomous algorithms in the robot related to machine learning and digital image processing taking advantage of the ability to connect directly with sensors and actuators.

- OpenCR (Open-source Control Module for ROS) hardware control board: This board was developed to support ROS embedded systems, and is located one level below the Raspberry Pi board. It is implemented around the STM32F7 microcontroller from STMicroelectronics which has an ARM Cortex-M7 core with floating point unit. Development can be done with Arduino IDE and Scratch as well as traditionally with firmware. The board contains an inertial measurement unit (IMU) with a three-axis accelerometer, a gyroscope and a magnetometer, and allows connecting the Raspberry Pi to motors and sensors.

- Additional elements. Other important elements to consider in the robot include the motors and the 11.1 V lithium-polymer (LiPo) battery. It is a differential platform with two independent Dynamixel XL430-W250 motors on each wheel.

Robot Operating System (ROS)

ROS is really a meta-operating system, that is, although it is an operating system for robotics, it runs on top of another operating system (a UNIX system). The advantage of being an operating system for robots is that it allows supervising different parts of the robot (control boards, sensors, and actuators) regardless of the programming language that each of them uses in its implementation.

The structure of ROS is characterized by a file system in which an executable file makes up a node. One node can instruct a sensor to watch for a specific event, while another

ISSN 1692-8407

Table 1

Differences between TurtleBot robot models

	Burger	Waffle
Maximum translational velocity	0.22 m/s	0.26 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)	1.82 rad/s (104.27 deg/s)
Maximum payload	15 kg	30 kg
Size (L × W × H)	138 mm × 178 mm × 192 mm	281 mm × 306 mm × 141 mm
Weight	1.0 kg	1.8 kg

node can specify the movement of an actuator in response to this specific event. The two nodes in this case communicate with each other via message files. A message file can be of different types: topics, services, and actions, and they are grouped with the nodes to work together in a package folder. When this group of nodes is executed, the ROS master server takes on the task of messenger between the nodes, i.e., the node first sends the message to the master, which communicates the message to the receiving node.

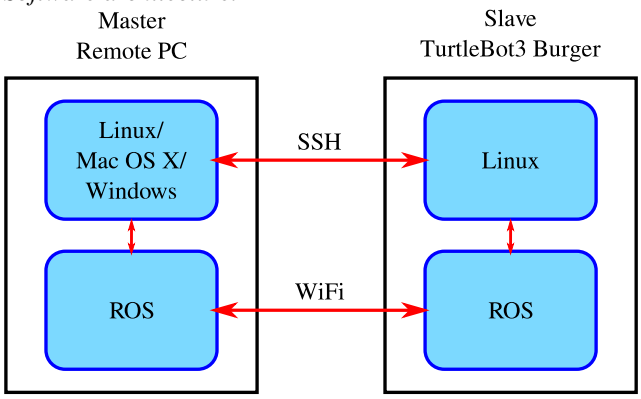
Software overview

The Raspberry Pi board requires an operating system adapted to its processor. Normally a Linux distribution is selected because of its development advantages and low cost. In the laboratory experiments carried out in this research, we used Ubuntu MATE 16.04 because of the extensive documentation, libraries, and repositories. As usual, this distribution was modified by the research group to suit the needs of the project.

The Raspberry Pi of each robot (and the computers of the research group) is connected to a central wireless access point (AP). This scheme facilitates the configuration of the algorithm, as well as the conceptualization of the algorithm architecture in multi-agent schemes. However, it is not the recommended working strategy when working with more than five robots since it presents a large latency in the communication that hinders the behavior of the algorithms. A change to the proposed working scheme for future implementations of swarm algorithms is to configure the Raspberry Pi board of each robot as an AP. During the tests with the multi-agent algorithms, there were connection problems due to the number of devices connected to the router, even though only two robots were used as maximum in the tests. Allowing the robots to have their own AP with fixed IP decreases the network load, avoiding communication problems. This also changes the implementation strategy of the algorithms, which is why it was not done at this stage of the project but is raised as a necessity for future work.

Figure 1

Software architecture.



To reduce the processing and memory requirements of the Raspberry Pi we removed unnecessary software included by default in the Linux distribution (office software and some system tools). Also, the graphical interface and libraries for handling peripherals such as printers were removed. ROS was installed in this tweaked version of Linux as well as important development tools in our algorithms such as Python and OpenCV. Although the master computer used in this research is also a Linux machine (MX Linux 21 Wildflower 64 bits with Kernel 5.10), it is also possible to use Windows OS servers (Fig. 1).

TurtleBot3 Burger control

The control code presented here was developed in Python for the kinematics of the TurtleBot3 Burger robot and assumes that ROS is installed on both the server computer and the robot's Raspberry Pi. Detailed information can be found on the website of ROBOTIS, the robot's manufacturer.

https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

The detailed model of this robot can be found in this paper (Lee et al., 2000).

ISSN 1692-8407

Figure 2

Libraries.

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from math import pow, atan2, sqrt, sin, cos
from tf.transformations import
    euler_from_quaternion
import numpy as np
```

Code breaking down

The first block of code corresponds to the libraries used in the node. Each node in ROS carries the

```
#!/usr/bin/env python
```

declaration at the top, this ensures that the code is executed as a Python script. ROS messages must be imported for use as any Python library. The first library imported is `Twist` from `geometry_msgs.msg`, which allows controlling the movement of the robot, for which a message is published with two variables, the amount of rotational (angular) movement and the amount of linear movement. The third library corresponds to `Odometry` from `nav_msgs.msg`, this allows to inform the position and velocity of the robot. A set of libraries is also imported for mathematical calculations and to convert a quaternion rotation (robot orientation) to the corresponding Euler angles. Finally, NumPy is included for vector and matrix handling (Fig. 2).

The next section of the code is the block corresponding to the initialization of the ROS connections. In this block, the nodes are asked to subscribe and to publish messages under the corresponding topics, in our case-specific velocity and odometry topics. Also in this section, the pose values are reported (Fig. 3).

The next block of code defines the use of the sensor values in the robot model to estimate the position in the navigation environment, and then produce the movement of the robot if it is not yet within the tolerance range of the target point. This is the general control code of the robot, this code can be modified so that the target point corresponds to the estimation made by the robot itself from the location of neighboring robots, which can be detected from the LiDAR, as advanced in our research, or from a more complex sensor such as a digital camera, which is proposed as the future development of our work.

Two gains, K1 and K2 are used to adjust the robot velocity (linear and rotational), these values can be adjusted according to the laboratory behavior of the robot. The first task of this section of the code is to estimate the values of rho (distance between the robot and the target position) and phi (error between the direction of the robot and the direction from the origin to the target position) from the robot model. This can be done concerning a global reference, or a local one set with respect to the robot readings. After defining its location, control signals are sent to the robot to reduce the error to the target position. Finally, a message is sent when the robot is within the tolerance region of the target point (Fig. 4).

At the end is the `Main` section of the code. The objective is to keep the robot moving until the task is completed. Here you can include conditions or events that force the robot to leave the target point, for example, if the grouping criterion by QS (activation of behaviors by population size) has not been met or if you want to activate a new behavior (Fig. 5).

Gazebo is a 3D simulator that can show the behavior of robots according to the ROS code. To implement this simulation with the above code can be run on the terminal:

```
$ roslaunch turtlebot3_gazebo
turtlebot3_empty_world.launch
```

The code shown is only the structure that allows to read the sensors, estimate the position of the robot, and send the motion signals. However, it is a complete code that can be modified to implement the motion algorithms according to the navigation strategy to be evaluated in the robot.

Conclusion

This article presents the most important implementation details when developing motion planning schemes on the TurtleBot3 Burger platform using ROS. Both the basic information about this platform and some useful Python code structures for its management are presented. This work was developed during the implementation of flocking schemes with the robot, for which the performance of algorithms was evaluated on one or two robots, and then scaling the behavior to dozens of agents. During these tests, several difficulties were encountered, such as synchronization problems, and message sending between robots and control computer (network latency), which although not explicitly detailed because they do not affect the performance of the algorithm, they do raise the need to continue working on the implementation on the platform. Leaving these elements aside, the flocking behavior, about the strategy proposed for its implementation in our research, is feasible from the results, particularly with this robot, and formulates new lines of work that include the possibility of new sensors in the agents (digital cameras), new customized agents with similar functionality (agents built in our laboratory), and even study of the interaction between different populations, with different objectives.

ISSN 1692-8407

Figure 3

ROS connection.

```
class turtlebot():
    def __init__(self):
        # Create node, publisher and subscriber
        rospy.init_node('turtlebot_controller', anonymous=True)
        self.velocity_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        self.pose_subscriber = rospy.Subscriber('/odom', Odometry, self.callback)
        self.pose = Odometry()
        self.rate = rospy.Rate(10)

    # Response value of pose received
    def callback(self, data):
        self.pose = data.pose.pose.position
        self.orient = data.pose.pose.orientation
        self.pose.x = round(self.pose.x, 4)
        self.pose.y = round(self.pose.y, 4)
```

Acknowledgment

This paper has been funded by the Facultad Tecnológica of the Universidad Distrital Francisco José de Caldas. The author would like to thank the ARMOS research group for the laboratory implementations of algorithms and prototypes.

References

Al-Mashhadani, Z., Mainampati, M., & Chandrasekaran, B. Autonomous exploring map and navigation for an agricultural robot. In: *2020 3rd international conference on control and robots (ICCR)*. 2020, 1–6. https://doi.org/10.1109/ICCR51572.2020.9344404.

Aslan, M. F., Durdu, A., Yusefi, A., Sabanci, K., & Sungur, C. (2021). A tutorial: Mobile robotics, SLAM, bayesian filter, keyframe bundle adjustment and ROS applications. *Studies in computational intelligence* (pp. 227–269). Springer International Publishing. https://doi.org/10.1007/978-3-030-75472-3_7

Fernandes, J., Li, K., Mirabile, J., & Vesonder, G. Application of robot operating system in robot flocks. In: *2019 IEEE 10th annual ubiquitous computing, electronics & mobile communication conference (UEMCON)*. 2019, 1–6. https://doi.org/10.1109/UEMCON47517.2019.8993017.

Groß, D. T. (2021). *An implementation approach of the gap navigation tree using the turtlebot 3 burger and ros kinetic* (Master's thesis). FH Vorarlberg (Fachhochschule Vorarlberg). FH Vorarlberg (Fachhochschule Vorarlberg). https://doi.org/10.25924/opus-3888

Hou, Y. C., Sahari, K. S. M., Weng, L. Y., Foo, H. K., Rahman, N. A. A., Atikah, N. A., & Homod, R. Z. (2020). Development of collision avoidance system for multiple autonomous mobile robots. *International Journal of Advanced Robotic Systems*, *17*(4), 172988142092396. https://doi.org/10.1177/1729881420923967

Koubaa, A. (2020). *Robot operating system - the complete reference* (Vol. 4). Springer.

Lee, S.-O., Cho, Y.-J., Hwang-Bo, M., You, B.-J., & Oh, S.-R. A stable target-tracking control for unicycle mobile robots. In: *Proceedings. 2000 IEEE/RSJ international conference on intelligent robots and systems (IROS 2000) (cat. no.00ch37113)*. 2000, 1–6. https://doi.org/10.1109/IROS.2000.895236.

Lentin, J. (2018). *Robot operating system (ros) for absolute beginners*. Apress.

Moreno, A., & Páez, D. (2017). Performance evaluation of ros on the raspberry pi platform as os for small robots. *Tekhnê*, *14*(1), 61–72.

Poza-Lujan, J.-L., Posadas-Yagüe, J.-L., Munera, E., Simó, J. E., & Blanes, F. Object recognition: Distributed architecture based on heterogeneous devices to integrate sensor information. In: *Distributed computing and artificial intelligence, 16th international conference*. 2019, 181–188. https://doi.org/10.1007/978-3-030-23887-2_21.

Quigley, M., Conley, K., Gerkey, B., & Faust, J. Ros: An open-source robot operating system. In: *Icra workshop on open source software*. 2009, 1–6.

Ratul, M. T. A., Mahmud, M. S. A., Abidin, M. S. Z., & Ayop, R. Design and development of GMapping based SLAM algorithm in virtual agricultural

ISSN 1692-8407

Figure 4

Sensing, position estimation and motion.

```
    def move2goal(self):
        K1=0.5
        K2=0.5
        goal_pose_ = Odometry()
        goal_pose = goal_pose_.pose.pose.position
        goal_pose.x = input("Set␣x␣goal:")
        goal_pose.y = input("Set␣y␣goal:")
        distance_tolerance = input("Tolerance:")
        vel_msg = Twist()
        r = sqrt(pow((goal_pose.x - self.pose.x), 2) +
            pow((goal_pose.y - self.pose.y), 2))
        while r >= distance_tolerance:

            r = sqrt(pow((goal_pose.x - self.pose.x), 2) +
                pow((goal_pose.y - self.pose.y), 2))
            psi = atan2(goal_pose.y - self.pose.y, goal_pose.x - self.pose.x)
            orientation_list = [self.orient.x, self.orient.y, self.orient.z,
                self.orient.w]

            # Adjust according to navigation algorithm
            (roll, pitch, yaw) = euler_from_quaternion (orientation_list)
            theta = yaw
            phi = theta - psi
            if phi > np.pi:
            phi = phi - 2*np.pi
            if phi < -np.pi:
            phi = phi + 2*np.pi

            vel_msg.linear.x = K1*r*cos(phi)
            vel_msg.angular.z = -K1*sin(phi)*cos(phi)-(K2*phi)

            # Publishing input
            self.velocity_publisher.publish(vel_msg)
            self.rate.sleep()
        # Stopping the robot after the movement is over
        vel_msg.linear.x = 0
        vel_msg.angular.z =0
        self.velocity_publisher.publish(vel_msg)
```

Figure 5

Continuous robot navigation.

```
if __name__ == '__main__':
    x = turtlebot()
    while 1:
        try:
        x.move2goal()
    except:
        pass
```

environment. In: *2021 11th IEEE international conference on control system, computing and engineering (ICCSCE)*. 2021. https://doi.org/10.1109/ICCSCE52189.2021.9530991.