БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ • BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, **67**
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, **67**

София • 2016 • Sofia

Indoor navigation for service mobile robots using Robot Operating System (ROS)

Denis Chikurtev

Institute of Information and Communication Technologies – BAS

Email: denis@iinf.bas.bg

Abstract: *this paper presents implementation of indoor navigation for service mobile robots. Problems like hardware driver, odometry and sensor data stream identification are solved to set up ROS Navigation stack. The Navigation stack provide all the necessary functions for mobile robots to be autonomous. We applied the navigation on our robot and it is working pretty good. The robot reaches any given position and avoid obstacles, controlled by the navigation.*

Keywords: *mobile robot, service robot, navigation, ROS, odometry, laser sensor.*

1. Introduction

Service robots are very popular and interesting for the people. They are used in different areas for specific applications. Most of the service robots are mobile, because they try to replace humans in their daily work. These robots can be house cleaners, personal assistants, robots for security and inspection and e.t. Most of these jobs requires free and accurate movement by the robots. Because of that we introduce our work of creation navigation for our service mobile robot (fig. 1).

Figure 1. Service mobile robot

1.1. Robot Operating System

Robot Operating System (ROS) is created for controlling robots [1]. ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.

One of the basic goals of ROS is to enable roboticists to design software as a collection of small, mostly independent programs called nodes that all run at the same time. For this to work, those nodes must be able to communicate with one another. The part of ROS that facilitates this communication is called the ROS master. A running instance of a ROS program is called a node. The primary mechanism that ROS nodes use to communicate is to send messages. Messages in ROS are organized into named topics. The idea is that a node that wants to share information will publish messages on the appropriate topic or topics; a node that wants to receive information will subscribe to the topic or topics that it's interested in. The ROS master takes care of ensuring that publishers and subscribers can find each other; the messages themselves are sent directly from publisher to subscriber.

1.2. Navigation stack

The Navigation Stack is fairly simple on a conceptual level. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base. Use of the Navigation Stack on an arbitrary robot, however, is a bit more complicated. As a pre-requisite for navigation stack use, the robot must be running ROS, have a tf transform tree in place, and publish sensor data using the correct ROS Message types. Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level [2].

While the Navigation Stack is designed to be as general purpose as possible, there are three main hardware requirements that restrict its use:

a) It is meant for both differential drive and holonomic wheeled robots only. It assumes that the mobile base is controlled by sending desired velocity commands to achieve in the form of: x velocity, y velocity, theta velocity.

b) It requires a planar laser mounted somewhere on the mobile base. This laser is used for map building and localization.

c) The Navigation Stack was developed on a square robot, so its performance will be best on robots that are nearly square or circular. It does work on robots of arbitrary shapes and sizes, but it may have difficulty with large rectangular robots in narrow spaces like doorways.

tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time. A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. tf keeps track of all these frames over time, and allows you to ask questions like:

- Where was the head frame relative to the world frame, 5 seconds ago?
- What is the pose of the object in my gripper relative to my base?
- What is the current pose of the base frame in the map frame?

tf can operate in a distributed system. This means all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system. There is no central server of transform information [3].

1.3. Our mobile robot specification

Our robot is based on a differential drive platform. This particular design has two motors mounted at fixed positions on the left and the right side of the robot. They

are independently driving one wheel each. Since three points are required to define a plane and our system is operating in a two dimensional plane, this design requires at least one additional passive caster wheel or a slider, depending on the location of the driven wheels. We chose design with two passive wheels or sliders, one placed in the front and one at the back side of the robot. This allows rotation about the center of the robot. However, this design can introduce surface contact problems, because it is using four contact points. The driving control for differential drive type mobile platforms is complex, because it requires coordination and cooperation of two separate driven wheels.

The mobile robot electronics consist of computer, robot control board, battery, camera, Kinect sensor, infrared and ultra-sonic sensors, DC motors, wheel encoders, microphone and speakers [4].

2. Methods

In this section we represent configuration of the Navigation stack and its components according to our mobile robot specifications. We calculated odometry and odometry errors, configured sensor data messages and base controller signals.

2.1. Robot setup

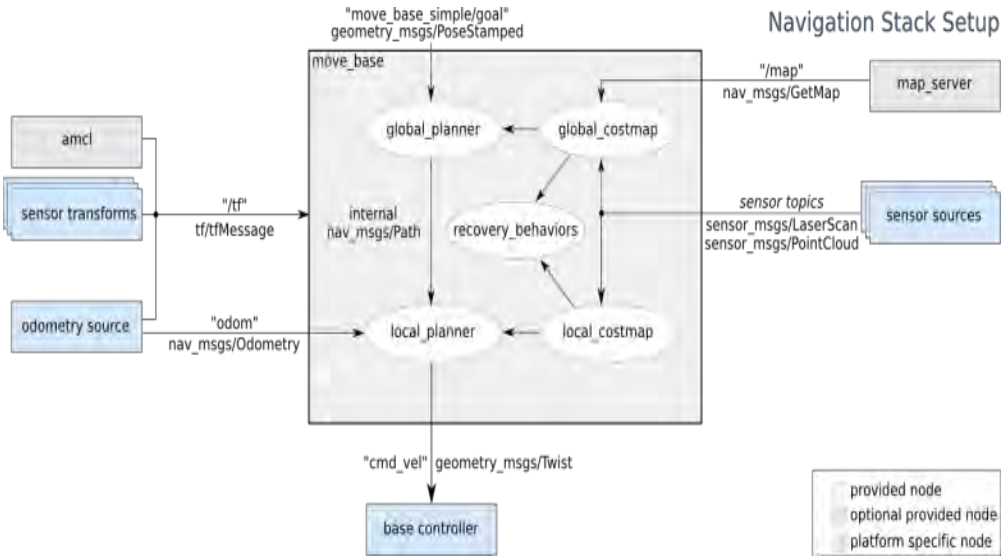

Figure 2. Navigation stack configuration

The navigation stack assumes that the robot is configured in a particular manner in order to run. The diagram above shows an overview of this configuration. The white components are required components that are already implemented, the gray components are optional components that are already implemented, and the blue components must be created for each robot platform.

2.2. Odometry data calculation

Odometry computes the robot's relative horizontal displacement and change in orientation as a function of the incremental horizontal displacement of the drive wheels [5]. The latter is found from incremental wheel encoders as follows: Suppose that at sampling interval I the left and right wheel encoders show a pulse increment of NL and NR, respectively. Suppose further that

cm = Dn/nCe

where
cm - Conversion factor that translates encoder pulses into linear wheel displacement.
Dn - Nominal wheel diameter (in mm).
Ce - Encoder resolution (in pulses per revolution).
n - Gear ratio of the reduction gear between the motor (where the encoder is attached) and the drive wheel.
One can then compute the incremental travel distance for the left and right wheel, UL,i and UR,i according to

UL/R, i = cm NL/R, i

Our robot has the following parameters:

- Wheel radius – 76,2 mm
- Distance between wheels – 390 mm
- Encoders resolution – 144 ticks / turn

We calculate the distance travelled by each wheel in meters, given the current readings of the encoders [6]:

left_distance: = (current_left_encoder_ticks – last_left_encoder_ticks)/ LEFT_TICKS_PER_METER

last_left_encoder_ticks: = current_left_encoder_ticks

Next, we estimate the total distance between the current and previous positions in meters:

distance: = (left_distance + right_distance) / 2.0

We calculate the heading of the robot:

theta: = theta + (left_distance – right_distance) / DISTANCE_BETWEEN_WHEELS

Finally, we can estimate the current position (x, y) of our robot:

x: = x + distance * cos(theta)
y: = y + distance * sin(theta)

Odometry is based on simple equations that are easily implemented and that utilize data from inexpensive incremental wheel encoders. However, odometry is based on the assumption that wheel revolutions can be translated into linear displacement relative to the floor. This assumption is only of limited validity. One extreme example is wheel slippage: If one wheel was to slip on, say, an oil spill, then the associated encoder would register wheel revolutions even though these revolutions would not correspond to a linear displacement of the wheel [7].

We publishing the odometry information over nav_msgs/Odometry message. The pose in this message corresponds to the estimated position of the robot in the odometric frame along with an optional covariance for the certainty of that pose estimate. The twist in this message corresponds to the robot's velocity in the child frame, normally the coordinate frame of the mobile base, along with an optional covariance for the certainty of that velocity estimate [8].

2.3. Configuring sensor sources

Publishing data correctly from sensors over ROS is important for the Navigation Stack to operate safely. If the Navigation Stack receives no information from a robot's sensors, then it will be driving blind and, most likely, hit things. There are many sensors that can be used to provide information to the Navigation Stack: lasers, cameras, sonar, infrared, bump sensors, etc. However, the current navigation stack only accepts sensor data published using either the sensor_msgs/LaserScan Message type or the sensor_msgs/PointCloud Message type.

For robots with laser scanners, ROS provides a special Message type in the sensor_msgs package called LaserScan to hold information about a given scan. LaserScan Messages make it easy for code to work with virtually any laser as long as the data coming back from the scanner can be formatted to fit into the message. We are using the Kinect sensor as a laser scanner [9]. The package that provides ROS to interface with depth sensors is openni_camera. The topics that we need from that node are for the depth sensor of the Kinect:

Depth camera [10]

- Published only when ~depth_registration is false (OpenNI registration disabled).

depth/camera_info (sensor_msgs/CameraInfo) – Camera calibration and metadata.
depth/image_raw (sensor_msgs/Image) – Raw image from device. Contains uint16 depths in mm – Registered depth camera (aligned with RGB camera)

- Published directly by driver only when ~depth_registration is true (OpenNI registration enabled).

depth_registered/camera_info (sensor_msgs/CameraInfo) – Camera calibration and metadata. Same as rgb/camera_info but time-synced to depth_registered/image_raw.

depth_registered/image_raw (sensor_msgs/Image) – Raw image from device. Contains uint16 depths in mm.

2.4. Controlling the robots base

The navigation stack assumes that it can send velocity commands using a geometry_msgs/Twist message assumed to be in the base coordinate frame of the robot on the "cmd_vel" topic. This means there must be a node subscribing to the "cmd_vel" topic that is capable of taking (vx, vy, vtheta) <==> (cmd_vel. linear.x, cmd_vel. linear.y, cmd_vel.angular.z) velocities and converting them into motor commands to send to a mobile base. The move_base node provides a ROS interface for configuring, running, and interacting with the navigation stack on a robot [11].

Next thing we did to achieve robust control was using regulators based on, the aforementioned, PID technique. This type of control was integrated into the firmware of the used Eddie Control Board in our robot. We used it, to regulate the speed of each driving wheel. The feedback was given by the mounted on the wheel axis quadrature encoder. There were two separate controllers on each wheel working on the following principle figure 3.

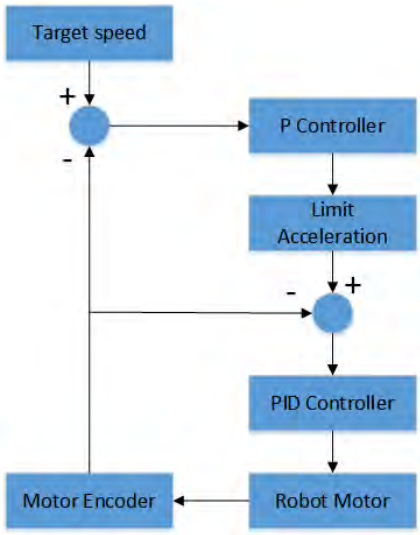

Figure 3. PID control sheme

The encoder ticks measure the travelled by the robot distance. By differentiating it, the current speed of the wheel can be estimated. Then, the difference between the desired speed and the current one is computed. This difference is the given on the input of a simple P controller whose task is to convert the desired speed rate into a desired acceleration. This acceleration is then limited and feeded into PID controller. On its output is the necessary motor power for achievement our goal speed. This additional PID control improve the movement of the robot [12].

3. Results

We created map of the floor (fig. 4), then we send commands to the robot. These commands consist the desired position and orientation that it must reach. The robot reached every position that we have given as a destination point. In figure 5 is shown how the robot is moving from one room to another. Every movement is controlled by the ROS Navigation stack.

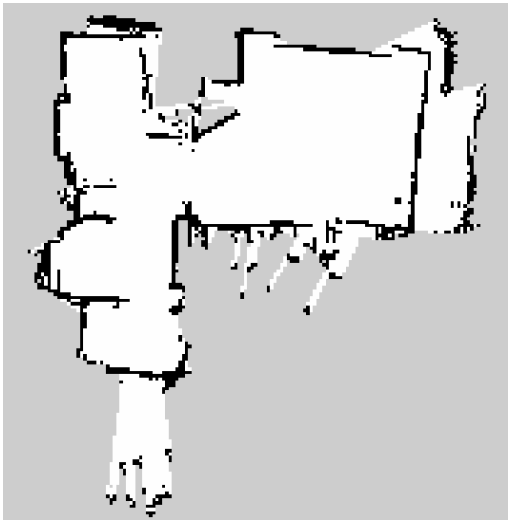

Figure 4. Generated map

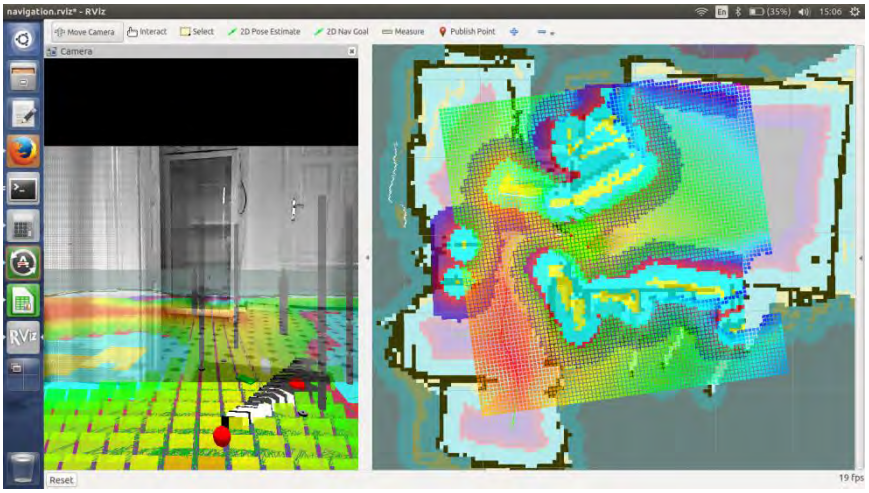

Figure 5. Navigation in real time

The robot handles every given destination and it is moving smooth and fast enough. The accuracy of positioning and orientation when it reaches the given destination is satisfying.

4. Conclusion

Robot operating system is easily applicable and functional. It can control different types of robots. The Navigation stack provide good autonomous navigation, when it's parameters are well configured. To improve the navigation, we can change the Kinect sensor with laser scanner, or add other type of sensors for more observations of the area.

Navigation provide safety and good positioning to the robots and is very applicable in inspection. Using navigation, we don't have to be worry about how to control the robot by joystick or phone, we just set desire point and the navigation do its job.

5. References

[1] J. M. O'Kane, A Gentle Introduction to ROS, 2013.

[2] ROS, "Setup and Configuration of the Navigation Stack on a Robot," ROS.org, [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup.

[3] T. Foote, "tf: The Transform Library," in *TePRA*, Woburn, 2013.

[4] Nayden Chivarov, Denis Chikurtev, Kaloyan Yovchev, Stefan Shivarov, "Cost-Oriented Mobile Robot Assistant for Disabled Care," in *IFAC*, Sozopol, 2015.

[5] Borenstein, Johann and Liqiang Feng, "Correction of Systematic Odometry Errors in Mobile Robots," in *IROS '95*, Pittsburgh, 1995.

[6] Nayden Chivarov, Stefan Shivarov, Kaloyan Yovchev, Denis Chikurtev, Nedko Shivarov, "Intelligent modular service mobile robot ROBCO 12 for elderly and disabled persons care," in *RAAD*, Smolenice, 2014.

[7] Borenstein, Johann and Liqiang Feng, "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," in *Transactions on Robotics and Automation*, 1996.

[8] I. Saito, "Publishing Odometry Information over ROS," ROS, 11 2015. [Online]. Available: http://wiki.ros.org/ navigation/Tutorials/RobotSetup/Odom

[9] Microsoft, "Kinect for Windows," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges. [Accessed 10 10 2016].

[10] M. Guenther, "openni_camera," ROS, 04 2016. [Online]. Available: http://wiki.ros.org/openni_camera. [Accessed 11 2016].

[11] "move_base," ROS, 3 2016. [Online]. Available: http://wiki.ros.org/move_base. [Accessed 10 2016].

[12] D. Chikurtev, "Improving and Optimizing the Navigation for Mobile Robot for Inspection by Using Robot Operating System," *Problems of Engineering Cybernetics and Robotics,* pp. 63-74, 2015.

Acknowledgments
The research work reported in the paper is partly supported by the projects funded by the Young Scientists Grants, reg. No 102/2016.

Навигация в помещениях для обслуживания мобильных роботов с использованием Robot Operating System (ROS)

Денис Чикуртев

Institute of Information and Communication Technologies – BAS

Email: denis@iinf.bas.bg

Аннотация: *В настоящей работе представлены реализации крытого навигации для обслуживания мобильных роботов. Такие проблемы, как аппаратного драйвера, одометрии и датчика идентификации потока данных решаются создать РОС навигации стек. Стек навигации обеспечивают все необходимые функции для мобильных роботов быть автономными. Мы применили навигацию по нашим роботом, и это работает довольно хорошо. Робот достигает любого заданного положения и избегать препятствий, контролируемой навигации.*

Ключевые слова: *мобильный робот, робот сервис, навигация, ROS, одометрии, лазерный датчик.*