PAPER • OPEN ACCESS

Trajectory tracking control for a nonholonomic mobile robot under ROS

To cite this article: Khadir Lakhdar Besseghieur *et al* 2018 *J. Phys.: Conf. Ser.* **1016** 012008

View the article online for updates and enhancements.

Trajectory tracking control for a nonholonomic mobile robot under ROS

Khadir Lakhdar Besseghieur[1] , Radosław Trębiński[1], Wojciech Kaczmarek[1], Jarosław Panasiuk[1]

[1] Faculty of Mechatronics and Aerospace, Military University of Technology, Kaliskiego 2, 00-908, Warsaw, Poland

Corresponding author: besseghieurkh@hotmail.fr

Abstract. In this paper, the implementation of the trajectory tracking control strategy on a ROS-based mobile robot is considered. Our test-bench is the nonholonomic mobile robot 'TURTLEBOT'. ROS facilitates considerably setting-up a suitable environment to test the designed controller. Our aim is to develop a framework using ROS concepts so that a trajectory tracking controller can be implemented on any ROS-enabled mobile robot. Practical experiments with 'TURTLEBOT' are conducted to assess the framework reliability.

1. Introduction

Recently, a lot of research have been conducted in order to control mobile robots [1], [2] and [3]. The mobile robots with a steering wheel (unicycle) or two independent drive wheels like 'TURTLEBOT' are examples with considerably important engineering interest. A big part of mobile robots are subjected to non-integrable kinematic constraints which cannot be reduced to holonomic ones, they are called non-holonomic constraints. The problem known as the tracking trajectory problem is when the reference point on the robot is required to follow the desired trajectory. In other words, the distance between the reference point on the robot and the current path reference point must be kept as small as possible. Controlling a mobile robot to track a specific trajectory has several practical applications particularly while operating in known map environments. Robots running on ROS like TURTLEBOT are commonly used to autonomously navigate in known environments. The autonomous navigation algorithm must contain a trajectory planner which generates the appropriate trajectory with the purpose of arriving at a particular location, patrolling through specified area and at the same time avoiding collisions with different kinds of obstacles. Once the trajectory is generated, the robot must actually track it. Therefore, the trajectory tracking control plays a substantial role in the autonomous navigation algorithm. The aim of this work is to develop a ROS-based framework that allows researchers to implement their trajectory tracking control algorithms on ROS-enabled mobile robots.

A number of trajectory tracking control laws for nonholonomic systems can be found in the literature and different approaches have been used to tackle this problem. Among the first approaches, Lyapunov stable time-varying state-tracking control laws were pioneered in [4], [5] and [6]. A model predictive controller is proposed in [7]. To the best of our knowledge, the trajectory tracking control implementation on a ROS-enabled robot is not considered in the literature. In this paper, we propose a new framework based on ROS concepts for all the ROS-enabled robots so that real experiments for the trajectory tracking control problem can be conducted effectively. The reminder of this paper is organized as follows: the next section is dedicated for introducing 'TURTLEBOT' and its kinematic model, then the tracking problem is formulated and it is concluded by the controller design. In Section 3, the proposed ROS-framework is detailed. Experimental results are presented and discussed in section 4. Finally, we conclude this paper and list some future work directions in section 5.

2. Tracking problem formulation for 'TURTLEBOT'

TURTLEBOT is a highly capable autonomous mobile platform for developing robot applications. The hardware structure can be divided into three main parts: A KOBUKI base, a netbook computer, and sensors. The Kobuki base is a differential drive mobile robot base with two passive caster wheels for balancing. It is modeled as a two-wheeled differential drive robot. Our netbook runs Ubuntu 14.04 LTS OS and has ROS Indigo running on it. These robots are considered as a class of computer-controlled vehicles whose motion can be described or transformed into the following model of constrained movement in a plane as in equation (1). With *(x; y)* represent the planar coordinates, θ is the orientation with respect to the X-axis, v and w denote the translational and rotational velocity respectively:

$$\begin{pmatrix} x^{\cdot} \\ y^{\cdot} \\ \theta^{\cdot} \end{pmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} v \\ w \end{pmatrix} \tag{1}$$

It is easy to see that the TURTLEBOT is governed by the following nonholonomic constraint:

$$x^{\cdot}\sin\theta - y^{\cdot}\cos\theta = 0 \tag{2}$$

Now consider a reference trajectory of the form:

$$\begin{pmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{pmatrix} = \begin{bmatrix} \cos\theta_r & 0 \\ \sin\theta_r & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} v_r \\ w_r \end{pmatrix} \tag{3}$$

Where v_r and w_r are the linear and angular reference velocity respectively. The tracking errors and the errors dynamics can be defined as in [4] by:

$$\begin{pmatrix} e_x \\ e_y \\ e_z \end{pmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} \dot{e_x} \\ \dot{e_y} \\ \dot{e_\theta} \end{pmatrix} = \begin{bmatrix} e_y w - v + v_r \cos e_\theta \\ -e_x w + v_r \sin e_\theta \\ w_r - w \end{bmatrix} \tag{5}$$

To globally asymptotically stabilize the error dynamics in equation (5), JIANG and NIJMEIJER proposed their approach in [6], where time varying state feedback control laws based on the backstepping technique are proposed. Using the Lyapunov's direct method, global controller is derived for the trajectory tracking problem. Referring to [6], if $\dot{v_r}, \dot{w_r}, v_r$ and w_r are all bounded and with the condition of either v_r or w_r does not converge to zero, the error dynamics in equation (5) are globally asymptotically stabilized using the control inputs in equation (6). The proof of convergence are omitted here, the reader is referred to [6]. With $a=(w_r^2 + b v_r^2)^{1/2}$, $\varepsilon > 0$ and $b>0$, the control laws are:

$$\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} v_r \cos e_\theta + K_x e_x \\ w_r + (K_y e_y \sin(e_\theta)/e_\theta) + K_\theta . e_\theta \end{pmatrix} \; with \; \begin{pmatrix} K_x \\ K_y \\ K_\theta \end{pmatrix} = \begin{pmatrix} 2\varepsilon\, a \\ b\,|v_r| \\ 2\varepsilon\, a \end{pmatrix} \tag{6}$$

3. Tracking controller ROS implementation

The main contribution of this paper, mainly the proposed ROS framework is detailed in this section, the ROS packages and the nodes that are relevant in our framework are also presented. ROS is a Linux-based, an open source software package that provides a software framework to aid in the development of complex robotic applications [8]. The proposed framework needs first an offline-built 2D map of the environment necessary for the robot's localization. The map is generated using the open-source package 'G-Mapping' where the robot is teleoperated around the environment; the system uses the point cloud generated by Kinect and the Odometry information from the encoders to build a 2D map and stores it locally on the robot's laptop. Our framework mainly consists of: the localization packages and the control node. The two parts are connected through the topic named "*/tf*".

3.1. Localization
To track a desired path, it is crucial that the robot should be aware of where it is itself as well as where the rest of the world is in relation to itself. The ROS navigation stack is used to provide the localization of the robot based on the Adaptive Monte-Carlo Localization (AMCL) approach presented in [9]. It is based on a weighted particle system in which each particle represents an estimated pose of the robot and consists of two phases of calculation: the prediction and update phases. AMCL combines the on-board encoders' measurements and the data provided by the Kinect sensor to provide an accurate estimation of the robot position. The framework needs the 'map_server' to upload the offline-built map which is necessary for the AMCL node to estimate the robot's coordinates vis-à-vis the map's origin. AMCL then publishes the transform between the two frames '/map' and '/base_footprint' into the topic 'tf'. The first frame corresponds to the map's origin whereas the latter corresponds to the robot's base center. In other words, the robot's coordinates in the map's coordinate system are published into the topic 'tf'. To read this topic's data, we opted for the tf library [10]. It was designed to provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system. The tf library provides a listener module which allows us to track the robot's coordinates in the absolute frame named '/map'.

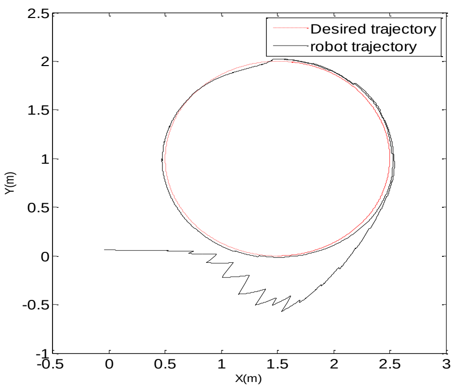
3.2. Control node
The control node code was developed in C++. The programmed control laws need as entries the robot's coordinates provided by the localization nodes. The control loop with period T=10ms starts with a 'tf_listener' from the 'tf' library [10] is used to obtain the frame transform between the frames '/map' and '/base-footprint' published in the topic 'tf'. Having obtained the actual robot's position, the trajectory tracking errors are evaluated and then the input control values are computed. A publisher is programmed to publish the obtained values to the corresponding topics in order to control the robot's actuators. To respect the physical limitations of the robot, the control speeds are limited within the control node program before published. A saturation of the command velocities [1] that preserves the current curvature $k = w/v$ is performed as:

$$\begin{cases} \sigma = max\{|v|/v_{max}, |w|/w_{max}, 1\} \\ v_c = sign(v)v_{max}, w_c = w/\sigma \; if \; \sigma = |v|/v_{max} \\ w_c = sign(w)w_{max}, v_c = v/\sigma \; if \; \sigma = |w|/w_{max} \\ v_c = v, w_c = w \; if \; \sigma = 1 \end{cases} \qquad (7)$$

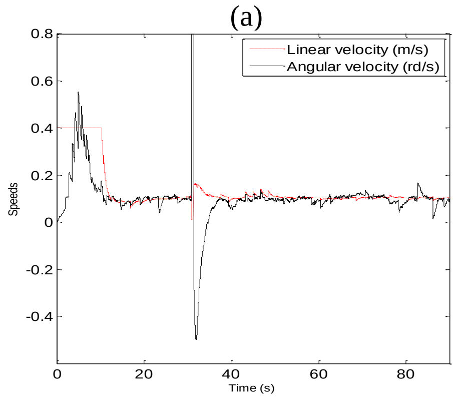
All the aforementioned nodes necessary for our framework are executed within one launch file, where all the parameters needed by the nodes are specified as well. The custom launch file is executed with one line command from a terminal.

4. Experimental results
To test the proposed framework, two reference trajectories are selected to be tracked by the robot. The mathematical model of the desired trajectories: circular and 8-shape trajectory are presented in equation (8.a) and (8.b) respectively. $(\boldsymbol{x_c}, \boldsymbol{y_c})$ represent the center coordinates and R is the radius of the circle, k_1, $k_2>0$, $k_1 \neq k_2$ and $k_3=0, 1$. In experimentation, the robot starts from the map's origin i.e. with an initial state error from the reference trajectory. Note that in both experiments the maximum control velocities were limited, as explained in section 3.2 and the maximum allowed linear and angular velocity in both simulation and experiments were $\boldsymbol{v_{max} = 0.4\, m/s, w_{max} = 0.8\, rd/s}$. The obtained data are recorded and saved on the robot's laptop then drawn using MATLAB.

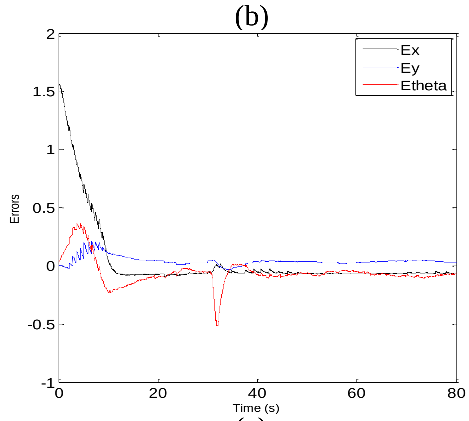
$$\begin{cases} x_r = x_c + R.sin(k.t) \\ y_r = y_c - R.cos(k.t) \\ \theta_r = k.t \\ v_r = k.R \\ w_r = k \end{cases} \quad (a), \qquad \begin{cases} x_r = x_c + R.sin(k_1.t) \\ y_r = y_c + R.sin(k_2.t) \\ \theta_r = Atan2(\dot{y}_r(t), \dot{x}_r(t)) + k_3\pi \\ v_r = \pm\sqrt{\dot{y}_r^{\,2}(t) + \dot{x}_r^{\,2}(t)} \\ w_r = \dfrac{\ddot{y}_r(t)\,\dot{x}_r(t) - \ddot{x}_r(t)\,\dot{y}_r(t)}{\dot{y}_r^{\,2}(t) + \dot{x}_r^{\,2}(t)} \end{cases} \quad (b) \qquad (8)$$



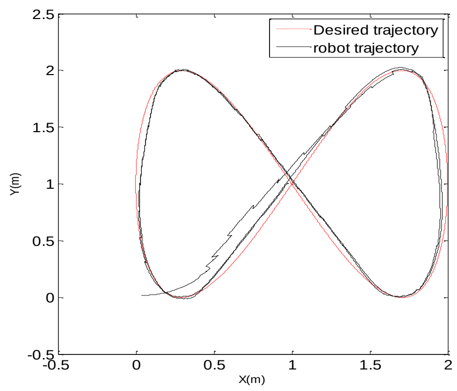
(a)



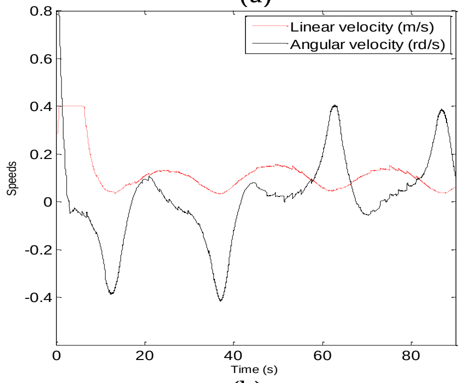
(b)



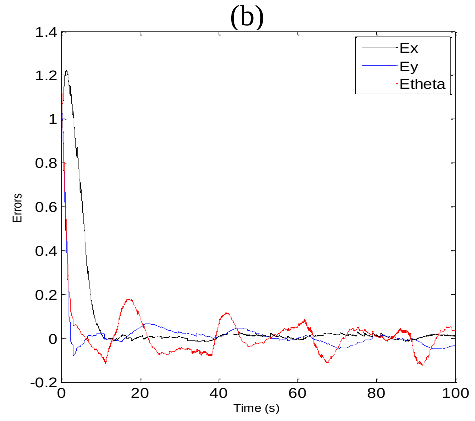
(c)



(a)



(b)



(c)

Figure 1. circular trajectory tracking. **Figure 2.** 8-shape trajectory tracking.

Before implementing the controller on the real robot, a simulation was conducted first to assess the effectiveness of the trajectory tracking controller in [6] before evaluating the framework itself. The environment 'GAZEBO' was used for simulating a virtual 'TURTLEBOT' seeking to track a circular trajectory. For the real tests on Turtlebot, the robot is initially at the origin (0,0) and required first to drive along a circle centered in *(x_c, y_c)=(1.5,1)* with *k*=0.1 and *R=1*. Figure 1.a shows that the robot tracks the desired trajectory with a negligible error. The tracking errors in figure 1.c converge after approximately 15 seconds to some negligible values around zero. Control inputs for the real robot in figure 1.b finish by converging around the desired trajectory velocities (v_r=0.1 m/s, w_r=0.1 m/s) as obtained by (8). In figure 2, the robot is required to track the 8-shape trajectory where $(x_c,y_c)=(1,1)$, $k_1=2л/100$ and $k_2=2*k_1$. This a more challenging task since the desired trajectory velocities are time variant in this case. Figures 2.a and 2.c show that the robot tracks the desired trajectory with negligible errors. The control inputs are illustrated in figure 2.b and are time variant but they lie within the limited velocity ranges. We can observe in all graphs that the errors do not converge to the absolute zero. This is due only to the localization error where the position estimate delivered by the AMCL node is not very accurate with using only Kinect. This can be improved with incorporating a laser scanner.

5. Conclusion

In this work, a ROS-framework is proposed to implement a trajectory tracking controller on a ROS-enabled mobile robot. Experiments were conducted on a 'TURTLEBOT' by implementing the controller proposed in [6]. Two different trajectories were successfully tracked by the robot which confirms the reliability of the proposed framework. Future works lie on implementing under ROS more complicated and sophisticated control laws as well as improving the robot's self localization. A more developed framework for controlling several mobile robots simultaneously under ROS is under study.

References

[1] Klančar G and Škrjanc I 2007 Tracking-error model-based predictive control for mobile robots in real time *Robotics and Autonomous Systems* 460-469
[2] Z. Li, J. Deng and R. Lu 2016 Trajectory-Tracking Control of Mobile Robot Systems Incorporating Neural-Dynamic Optimized Model Predictive Approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 740-749
[3] Mobayen S 2015 Fast terminal sliding mode tracking of nonholonomic systems with exponential decay rate *IET Control Theory & Applications* 1294–1301
[4] Kanayama Y., Kimura Y., Miyazaki F, Noguchi T 1990 A stable tracking control method for an autonomous mobile robot. *IEEE International Conf. on Robotics and Automation* 384-389
[5] Samson C., Ait-abderrahim K 1991 Feedback control of a nonholonomic wheeled cart in Cartesian space. *1991 IEEE International Conference on Robotics and Automation* 1136–1141
[6] Z-P JIANGdagger, H.NIJMEIJER 1997 Tracking Control of Mobile Robots: A Case Study in Backstepping. *Automatica* Volume 33, Issue 7 1393-1399
[7] Ollero A. and Amidi O 1991 Predictive path tracking of mobile robots. *Proceedings of 5th Int.Conf. on Advanced Robotics, Robots in Unstructured Environments* 1081 – 1086
[8] Quigley M., Gerky B., and Smart W. D 2015 Programming Robots with ROS, a Practical Introduction to the Robot Operating System *First. O'Reilly Media, Inc*
[9] Hennes D, Claes D, Meeussen W, 2012 Multi-robot collision avoidance with localization uncertainty. *11th Int. Conf. on Autonomous Agents and Multiagent Systems* 147-154.
[10] T. Foote 2013 tf: The transform library *In Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, 1–6.