Parametric Identification of the Dynamics of Mobile Robots and Its Application to the Tuning of Controllers in ROS

Walter Fetter Lages

Abstract This tutorial chapter explains the identification of dynamic parameters of the dynamic model of wheeled mobile robots. Those parameters depend on the mass and inertia parameters of the parts of the robot and even with the help of modern CAD systems it is difficult to determine them with a precision as the designed robot is not built with 100% accuracy; the actual materials have not exactly the same properties as modeled in the CAD system; there is cabling which density changes over time due to robot motion and many other problems due to differences between the CAD model and the real robot. To overcome these difficulties and still have a good representation of the dynamics of the robot, this work proposes the identification of the parameters of the model. After an introduction to the recursive least-squares identification method, it is shown that the dynamic model of a mobile robot is a cascade between its kinematic model, which considers velocities as inputs, and its dynamics, which considers torques as inputs and then that the dynamics can be written as a set of equations linearly parameterized in the unknown parameters, enabling the use of the recursive least-squares identification. Although the example is a differential-drive robot, the proposed method can be applied to any robot model that can be parameterized as the product of a vector of parameters and a vector of regressors. The proposed parameter identification method is implemented in a ROS package and can be used with actual robots or robots simulated in Gazebo. The package for the Indigo version of ROS is available at http://www.ece.ufrgs.br/twil/indigo-twil.tgz. The chapter concludes with a full example of identification and the presentation of the dynamic model of a mobile robot and its use for the design of a controller. The controller is based on three feedback loops. The first one linearizes the dynamics of the robot by using feedback linearization, the second one uses a set of PI controllers to control the dynamics of the robot, and the last one uses a non-linear controller to control the pose of the robot.

W.F. Lages (✉)
Federal University of Rio Grande do Sul, Av. Osvaldo Aranha, 103,
Porto Alegre RS 90035-190, Brazil
email: fetter@ece.ufrgs.br
URL: http://www.ece.ufrgs.br/∼fetter

A. Koubaa (ed.), *Robot Operating System (ROS)*, Studies in Computational Intelligence 707, DOI 10.1007/978-3-319-54927-9_6

Keywords Parametric identification · Dynamic model · Recursive least-squares · Controller tuning · Feedback linearization · Non-smooth controller

1 Introduction

Like any robot, a wheeled mobile robot is subject to kinematics and dynamics. Also, its kinematic model depends only on geometric parameters, while the dynamic model depends on geometric parameters and mass and inertia moments. However, different from manipulator robots, the kinematic model of a wheeled mobile robot is represented by differential equations. The output of the kinematic model of a mobile robot is its pose (position and orientation) while its inputs are velocities. Depending on how the model is formulated, its inputs may be the velocity on each wheels or angular and linear velocity of the robot, or any other variable which is homogeneous to velocity [9].

Based on those properties of the kinematic model of mobile robots, many controllers for mobile robots command velocities to the robot, under the assumption that those commanded velocities are instantaneously imposed on the robot. Of course, that assumption is only valid if the actuators are powerful enough regarding the mass and inertia moments of the robot. That is indeed the case for small robots or robots using servo motors, which are commanded in velocity, as actuators. In this case, the controller can be designed based on the kinematic model alone, whose parameters are usually well-known. Note that, as the kinematic model of a mobile robot is given by a differential equation, it is often called a dynamic model (because its described by dynamic, i.e. differential equations) although it does not effectively model the dynamics of the robot. In this chapter, that model is referred to as a kinematic model. The term dynamic model is reserved for models that describe the dynamics of the robot.

However, for larger robots, or robots in which actuators are not powerful enough to impose very fast changes in velocities, it is necessary to consider the dynamics of the robot in the design of the controller. Then, a more sophisticated model of the robot, including its kinematics and its dynamics, should be used. The output of this model is the robot pose, as well as in the kinematic model, but its inputs are the torques on the wheels. The parameters of this model depends on the mass and inertia parameters of the parts of the robot and even with the help of modern CAD systems it is difficult to know with good precision some of those values as the designed robot is not built with 100% accuracy; the actual materials have not exactly the same properties as modeled in the CAD system; there is cabling which density changes over time due to motion and many other problems.

To overcome the difficulties in considering all constructive details in a mathematical model and still have a good representation of the dynamics of the robot, it is possible to obtain a model by observing the system output given proper inputs as shown in Fig. 1. This procedure is called system identification [10] or model learning [18].

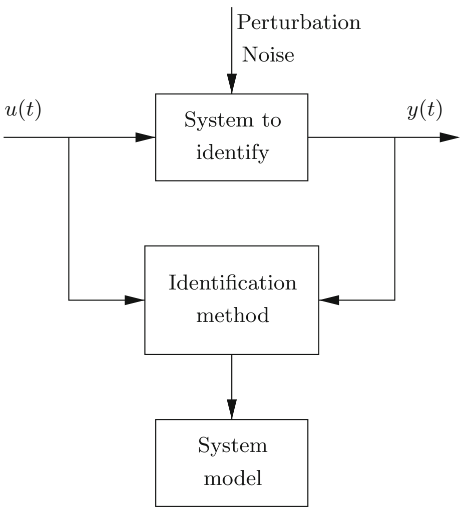


Fig. 1 Basic block diagram for system identification

It is shown that the dynamic model of a mobile robot can be properly parameterized such that the recursive least-squares method [10] can be used for parameter identification. The proposed parameter identification method is implemented in a ROS package and can be used with actual robots or robots simulated in Gazebo. The package for the Indigo version of ROS can be downloaded from http://www.ece.ufrgs.br/twil/indigo-twil.tgz. See Sect. 3 for details on how to install it.

The identified parameters and the respective diagonal of the covariance matrix are published as ROS topics to be used in the off-line design of controllers or even used online to implement adaptive controllers. The diagonal of the covariance matrix is a measure of confidence on the parameter estimation and hence can be used to decide if identified parameters are good enough. In the case of an adaptive controller, it can be used to decide if the adaptation should be shut-off or not.

The chapter concludes with a complete example of identification and controller design. Note that although the example and the general method is developed for differential-drive mobile robots, it can be applied to any robot, as long as it is possible to write the model in a way such that the unknown parameters are linearly related to the measured variables, as shown in Sect. 2.1.

More specifically, the remainder of this chapter will cover the following topics:

- a background on identification
- a background on modeling of mobile robots
- installing the required packages
- testing the installed packages
- description of the package for identification of mobile robots.

2 Background

2.1 Parametric Identification

In order to design a control system it is generally necessary to have a model of the plant (the system to be controlled). In many cases, those models can be obtained by analyzing how the system works and using the laws of Physics to write the set of equations describing it. This is called the white-box approach. However, sometimes it is not possible to obtain the model using this approach, due to complexity of the system or uncertainty about its parameters or operating conditions. In those cases, it might be possible to obtain a model through the observation of the system behavior as shown in Fig. 1, which is known as the black-box approach and formally called system identification.

In this chapter, the focus is on identification methods which can be used online, because they are more convenient for computational implementation and can be readily used for implementing adaptive controllers. When the parameter estimation is performed online, it is necessary to obtain a new updated estimate in the period between two successive samples. Hence, it is highly desirable for the estimation algorithm to be simple and easily implementable. A particularly interesting class of online algorithms are those in which the current estimate $\theta(t)$ is computed as a function of the former estimates, and then it is possible to compute the estimates recursively.

Let a single-input, single-output (SISO) system represented by its ARX[1] model:

$$\begin{aligned} y(t+1) = {} & a_1 y(t) + \cdots + a_p y(t-p+1) \\ & + b_1 u(t) + \cdots + b_q u(t-q+1) + \omega(t+1) \end{aligned} \tag{1}$$

where t is the sampling period index,[2] $y \in \mathbb{R}$ is the system output, $u \in \mathbb{R}$ is the system input, $a_i, i = 1, 2, \ldots p$ and $b_i, j = 1, 2, \ldots q$ are the system parameters and $\omega(t+1)$ is a Gaussian noise representing the uncertainty in the model.

The model (1) can be rewritten as:

$$y(t+1) = \phi^T(t)\theta + \omega(t+1) \tag{2}$$

[1] AutoRegressive with eXogenous inputs.

[2] Note that in system identification theory it is common to use t as the independent variable even though the model is a discrete time one.

with

$$\theta = \begin{bmatrix} a_1 \\ \vdots \\ a_p \\ b_1 \\ \vdots \\ b_1 \end{bmatrix}, \text{ the vector of parameters} \tag{3}$$

and

$$\phi(t) = \begin{bmatrix} y(t) \\ \vdots \\ y(t-p+1) \\ u(t) \\ \vdots \\ u(t-q+1) \end{bmatrix}, \text{ the regression vector.} \tag{4}$$

The identification problem consists in determining θ based on the information (measurements) about $y(t+1)$ and $\phi(t)$ for $t = 0, 1, \ldots, n$. To solve this problem, it can be formulated as an optimization problem with the cost to minimize:

$$J(n, \theta) = \frac{1}{n} \sum_{t=0}^{n-1} \left(y(t+1) - \phi^T(t)\theta \right)^2 \tag{5}$$

where $y(t+1) - \phi^T(t)\theta$ is the prediction error.

More formally:

$$\hat{\theta}(n) = \arg\min_{\theta} J(n, \theta) \tag{6}$$

Figure 2 shows a block diagram of the identification system implementing (6).

In order to solve the minimization (6) it is convenient to write it as:

$$\hat{\theta}(n) = \arg\min_{\theta} \left((Y(n) - \Phi(n)\theta)^T \, (Y(n) - \Phi(n)\theta) \right) \tag{7}$$

with

$$Y(n) = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(n) \end{bmatrix} \tag{8}$$

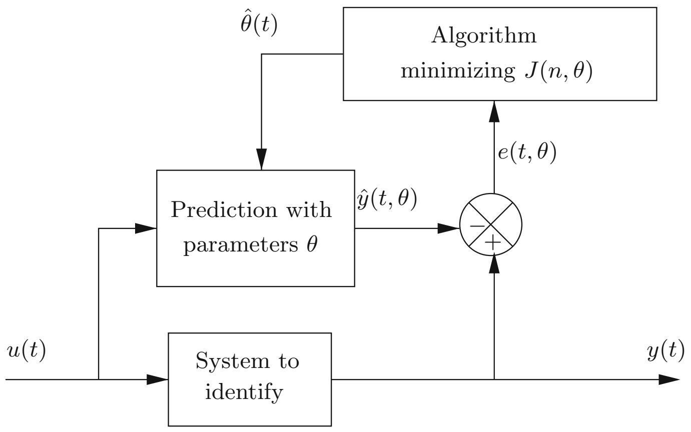


Fig. 2 Block diagram of system identification

and

$$\Phi(n) = \begin{bmatrix} \phi^T(0) \\ \phi^T(1) \\ \vdots \\ \phi^T(n-1) \end{bmatrix} \tag{9}$$

Then, (6) can be solved by making the differential of $J(n,\theta)$ with respect to θ equal to zero:

$$\left.\frac{\partial J(n,\theta)}{\partial\theta}\right|_{\theta=\hat{\theta}(n)} = 0 = -2\Phi^T(n)Y(n) + 2\Phi^T(n)\Phi(n)\hat{\theta}(n) \tag{10}$$

Hence,

$$\hat{\theta}(n) = \left(\Phi^T(n)\Phi(n)\right)^{-1}\Phi^T(n)Y(n) \tag{11}$$

or

$$\hat{\theta}(n) = \left(\sum_{t=0}^{n-1}\phi(t)\phi^T(t)\right)^{-1}\sum_{t=0}^{n-1}\phi(t)y(t+1) \tag{12}$$

Expression (12) is the solution of (6) and can be used to compute an estimate $\hat{\theta}$ for the vector of parameters θ at time instant n. However, this expression is not in a recursive form and is not practical for online computing because it requires the inversion of a matrix of dimension $(n-1)\times(n-1)$ for each update of the estimate. Furthermore, n keeps increasing without bound, thus increasing computation time and memory requirements.

For online computation it is convenient to have a recursive form of (12), such that at each update time, the new data can be assimilated without the need to compute everything again. To obtain such a recursive form define:

$$P(n) = \left(\sum_{t=0}^{n} \phi(t)\phi^T(t)\right)^{-1} \tag{13}$$

then, from (12):

$$\hat{\theta}(n+1) = P(n)\sum_{t=0}^{n} \phi(t)y(t+1) \tag{14}$$

On the other hand:

$$P^{-1}(n) = \sum_{t=0}^{n} \phi(t)\phi^T(t) \tag{15}$$

$$= \sum_{t=0}^{n-1} \phi(t)\phi^T(t) + \phi(n)\phi^T(n) \tag{16}$$

$$= P^{-1}(n-1) + \phi(n)\phi^T(n) \tag{17}$$

or

$$P(n) = \left(P^{-1}(n-1) + \phi(n)\phi^T(n)\right)^{-1} \tag{18}$$

$$= P(n-1) - P(n-1)\phi(n)\left(\phi^T(n)P(n-1)\phi(n) + 1\right)^{-1}\phi^T(n)P(n-1) \tag{19}$$

By using the Matrix Inversion Lemma[3] [3] with $A = P^{-1}(n-1)$, $B = \phi(n)$, $C = 1$ e $D = \phi^T(n)$, it is possible to compute:

$$P(n) = P(n-1) - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1 + \phi^T(n)P(n-1)\phi(n)} \tag{20}$$

which, replaced in (14) results:

$$\hat{\theta}(n+1) = P(n)\sum_{t=0}^{n} \phi(t)y(t+1) \tag{21}$$

$$= P(n)\left(\sum_{t=0}^{n-1} \phi(t)y(t+1) + \phi(n)y(n+1)\right) \tag{22}$$

[3]Matrix Inversion Lemma: $(A + BCD)^{-1} = A^{-1} - A^{-1}B\left(C^{-1} + DA^{-1}B\right)^{-1}DA^{-1}$.

$$= \left(P(n-1) - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1+\phi^T(n)P(n-1)\phi(n)} \right) \left(\sum_{t=0}^{n-1} \phi(t)y(t+1) + \phi(n)y(n+1) \right) \tag{23}$$

By expanding the product:

$$\begin{aligned} \hat{\theta}(n+1) = {} & P(n-1)\sum_{t=0}^{n-1} \phi(t)y(t+1) + P(n-1)\phi(n)y(n+1) \\ & - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1+\phi^T(n)P(n-1)\phi(n)} \sum_{t=0}^{n-1} \phi(t)y(t+1) \\ & - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1+\phi^T(n)P(n-1)\phi(n)} \phi(n)y(n+1) \end{aligned} \tag{24}$$

Then by delaying (14) a sampling period and replacing in (24):

$$\begin{aligned} \hat{\theta}(n+1) = {} & \hat{\theta}(n) + P(n-1)\phi(n)y(n+1) \\ & - \frac{P(n-1)\phi(n)\phi^T(n)}{1+\phi^T(n)P(n-1)\phi(n)} \hat{\theta}(n) \\ & - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1+\phi^T(n)P(n-1)\phi(n)} \phi(n)y(n+1) \end{aligned} \tag{25}$$

and by grouping together the terms in $\phi(n)y(n+1)$:

$$\begin{aligned} & \hat{\theta}(n+1) = \hat{\theta}(n) + \\ & + \frac{P(n-1) + P(n-1)\phi^T(n)P(n-1)\phi(n) - P(n-1)\phi(n)\phi^T(n)P(n-1)}{1+\phi^T(n)P(n-1)\phi(n)} \phi(n)y(n+1) \\ & - \frac{P(n-1)\phi(n)\phi^T(n)}{1+\phi^T(n)P(n-1)\phi(n)} \hat{\theta}(n) \end{aligned} \tag{26}$$

or

$$\begin{aligned} \hat{\theta}(n+1) = {} & \hat{\theta}(n) + \frac{P(n-1)}{1+\phi^T(n)P(n-1)\phi(n)} \phi(n)y(n+1) \\ & - \frac{P(n-1)\phi(n)\phi^T(n)}{1+\phi^T(n)P(n-1)\phi(n)} \hat{\theta}(n) \end{aligned} \tag{27}$$

which can be rewritten as:

$$\hat{\theta}(n+1) = \hat{\theta}(n) + \frac{P(n-1)\phi(n)}{1+\phi^T(n)P(n-1)\phi(n)} \left(y(n+1) - \phi^T(n)\hat{\theta}(n) \right) \tag{28}$$

The term multiplying the error can be regarded as the optimal gain of the identification algorithm. Hence, the solution for the problem (6) in a recursive form is given by:

$$\hat{\theta}(n+1) = \hat{\theta}(n) + K(n)\left(y(n+1) - \phi^T(n)\hat{\theta}(n)\right) \tag{29}$$

$$K(n) = \frac{P(n-1)\phi(n)}{1 + \phi^T(n)P(n-1)\phi(n)} \tag{30}$$

$$P(n) = \left(I - K(n)\phi^T(n)\right)P(n-1) \tag{31}$$

Expression (29) is an update of the previous parameter estimate $\hat{\theta}(n)$ by an optimal gain $K(n)$, from (30), multiplied by the prediction error, $y(n+1) - \hat{y}(n+1)$. Note that $\hat{y}(n+1) = \phi^T(n)\hat{\theta}(n)$ is the prediction of the system output. It can be shown that $P(n)$ as computed by (31) is the covariance of the prediction error and hence it is a measure of the confidence in the parameter estimates.

The Algorithm 1 details the procedure for parameter identification:

Algorithm 1 Recursive Least-Squares.

```
Initialize $\phi(0)$, $\hat{\theta}(0)$, e $P(-1) = cI$.
At sampling time $n+1$:
  1. Read system output $y(n+1)$ from sensors
  2. Compute the prediction of system output $\hat{y}(n+1)$:
        $\hat{y}(n+1) = \phi^T(n)\hat{\theta}(n)$    (32)
  3. Compute the gain $K(n)$:
        $K(n) = \frac{P(n-1)\phi(n)}{1 + \phi^T(n)P(n-1)\phi(n)}$    (33)
  4. Update the parameter vector estimate:
        $\hat{\theta}(n+1) = \hat{\theta}(n) + K(n)\left(y(n+1) - \hat{y}(n+1)\right)$    (34)
  5. Store $\hat{\theta}(n+1)$ for use, if necessary
  6. Update the covariance matrix:
        $P(n) = \left(I - K(n)\phi^T(n)\right)P(n-1)$    (35)
  7. Wait for the next sampling time
  8. Increment $n$ and return to step 1
```

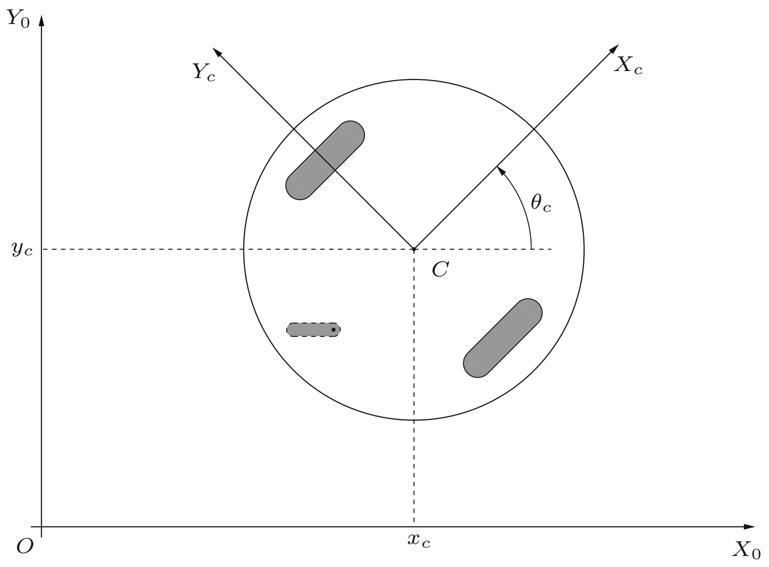


Fig. 3 Coordinate systems

2.2 Mobile Robot Model

The model of the mobile robot used in this chapter is described in this section. Figure 3 shows the coordinate systems used to describe the mobile robot model, where X_c and Y_c are the axes of the coordinate system attached to the robot and X_0 and Y_0 form the inertial coordinate system.

The pose (position and orientation) of the robot is represented by $x = \begin{bmatrix} x_c & y_c & \theta_c \end{bmatrix}^T$. The mobile robot dynamic model can be obtained based on the Lagrange-Euler formulation [9] and is given by:

$$\begin{cases} \dot{x} = B(x)u \\ H(\beta)\dot{u} + f(\beta, u) = F(\beta)\tau \end{cases} \tag{36}$$

where β is the vector of the angles of the caster wheels, $u = \begin{bmatrix} v & \omega \end{bmatrix}^T$ is the vector of the linear and angular velocities of the robot and τ is the vector of input torques on the wheels. $B(x)$ is a matrix whose structure depends on the kinematic (geometric) properties of the robot, while $H(\beta)$, $f(\beta, u)$ and $F(\beta)$ depend on the kinematic and dynamic (mass and inertia) parameters of the robot. Although this chapter is based on a differential-drive mobile robot, the model (36) is valid for any type of wheeled mobile robot. See [9] for details and examples for other types of wheeled mobile robots.

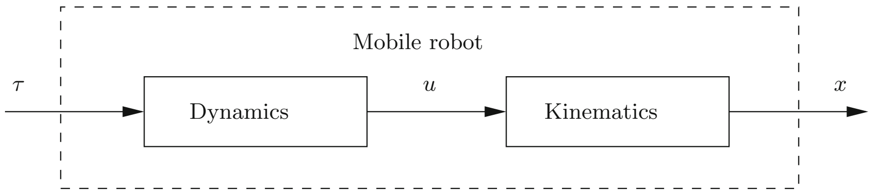
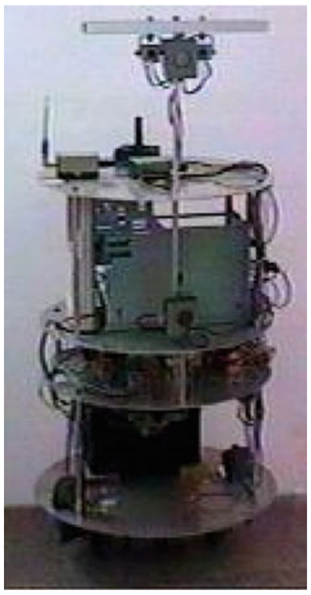


Fig. 4 Cascade between dynamics and the kinematic model

Fig. 5 The Twil mobile robot



Note that the dynamic model of the robot is a cascade between its kinematic model (the first expression of (36), with velocities as inputs) and its dynamics (the second expression of (36), with torques as inputs), as shown in Fig. 4.

This chapter is based on the Twil mobile robot (see Fig. 5), which is a differential-drive mobile robot, but the results and the ROS package for parameter identification can be used directly for any other differential-drive mobile robot, as it does not depends on Twil characteristics. For other types of wheeled mobile robots, the model has the same form as (36) and given its particular characteristics such as the location of the wheels with respect to the robot reference frame, the model (36) can be customized and rewritten in a form similar to the one used here for differential-drive robots. Then, the same procedure, could be used for parameter estimation.

The matrices of the model (36) customized for a differential-drive robot such as Twil are:

$$B(x) = \begin{bmatrix} \cos\theta_c & 0 \\ \sin\theta_c & 0 \\ 0 & 1 \end{bmatrix} \tag{37}$$

$$H(\beta) = I \tag{38}$$

$$f(\beta, u) = -\begin{bmatrix} 0 & K_5 \\ K_6 & 0 \end{bmatrix} \begin{bmatrix} u_1 u_2 \\ u_2^2 \end{bmatrix} = -f(u) \tag{39}$$

$$F(\beta) = \begin{bmatrix} K_7 & K_7 \\ K_8 & -K_8 \end{bmatrix} = F \tag{40}$$

where I is the identity matrix and K_5, K_6, K_7 and K_8 are constants depending only on the geometric and inertia parameters of the robot. Note that for this robot $H(\beta)$, $f(\beta, u)$ and $F(\beta)$ do not actually depend on β.

Note that only the dynamics of the robot depends on mass and inertia parameters, which are difficult to know with precision. Furthermore, u is a vector with linear and angular velocity of the robot, which can be easily measured, while x are the robot pose, which are more difficult to be obtained. However, the parameter of the kinematics depends on the geometry of the robot and can be obtained with good precision by calibration. Therefore, only the part of the model regarding the dynamics is used for parameter estimation, taking u as output and τ as input.

In the following it is shown that the dynamics of the robot can be written as a set of equations in the form of $y(k+1) = \phi^T(k)\theta(k)$, where y is the acceleration (measured or estimated from velocities), ϕ is the vector of regressors (measured velocities and applied torques) and θ is the vector of unknown parameters to be identified. Then, it is possible to obtain an estimate $\hat{\theta}$ for θ by using the recursive least squares algorithm [10] described in Sect. 2.1.

The parameters K_5, K_6, K_7 and K_8 depend on the geometric and mass properties of the robot in a very complex way. Even for a robot simulated in Gazebo, the same problem arises, as the model (36) is more simple than a typical robot described in URDF, which typically include more constructive details, for a realistic and good looking animation. On the other hand, the model described in URDF is not available in a closed form as (36), whose structure can be explored in the design of a controller. Also, it is not trivial to obtain a model in the form of (36) equivalent to an URDF description.

To overcome the difficulties in considering all constructive details in an algebraic model such as (36) and still have a good representation of the dynamics of the robot, the parameters of the model are identified.

In obtaining a model in a form suitable to be identified by the recursive least squares algorithm described in Sect. 2.1 it is important to note that only the second expression of (36) depends on the unknown parameters. Furthermore, u is a vector with linear and angular velocity of the robot, which can be easily measured, while x are the robot pose, which are more difficult to be obtained. Therefore, only the second expression of (36) will be used for parameter estimation, taking u as output and τ as input.

By using (37)–(40), the second expression of (36) for the Twil robot can be written as:

$$\dot{u} = \begin{bmatrix} 0 & K_5 \\ K_6 & 0 \end{bmatrix} \begin{bmatrix} u_1 u_2 \\ u_2^2 \end{bmatrix} + \begin{bmatrix} K_7 & K_7 \\ K_8 & -K_8 \end{bmatrix} \tau \tag{41}$$

Although (41) seems somewhat cryptic, its physical meaning can be understood as $\dot{u} = \begin{bmatrix} \dot{v} & \dot{\omega} \end{bmatrix}^T$ is the vector of linear and angular acceleration of the robot. Hence, the term $K_5 u_2^2$ represents the centrifugal, the term $K_6 u_1 u_2$ represents the Coriolis acceleration. Also, as the linear acceleration of the robot is proportional to the average of the torques applied to the left and right wheels, $1/K_7$ represents the robot mass and as the angular acceleration of the robot is proportional to the difference of torques, $1/K_8$ represents the moment of inertia of the robot.

For the purpose of identifying K_5, K_6, K_7 and K_8 it is convenient to write (41) as two scalar expressions:

$$\dot{u}_1 = K_5 u_2^2 + K_7 (\tau_1 + \tau_2) \tag{42}$$

$$\dot{u}_2 = K_6 u_1 u_2 + K_8 (\tau_1 - \tau_2) \tag{43}$$

Then, by discretizing (42)–(43) it is possible to obtain two recursive models: one linearly parameterized in K_5 and K_7 and another linearly parameterized in K_6 and K_8:

$$y_1(k+1) = \dot{u}_1(k) \simeq \frac{u_1(k+1) - u_1(k)}{T} \tag{44}$$

$$= K_5 u_2^2(k) + K_7(\tau_1(k) + \tau_2(k)) \tag{45}$$

$$y_1(k+1) = \begin{bmatrix} u_2^2(k) \\ \tau_1(k) + \tau_2(k) \end{bmatrix}^T \begin{bmatrix} K_5 \\ K_7 \end{bmatrix} \tag{46}$$

$$y_2(k+1) = \dot{u}_2(k) \simeq \frac{u_2(k+1) - u_2(k)}{T} \tag{47}$$

$$= K_6 u_1(k) u_2(k) + K_8(\tau_1(k) - \tau_2(k)) \tag{48}$$

$$y_2(k+1) = \begin{bmatrix} u_1(k) u_2(k) \\ \tau_1(k) - \tau_2(k) \end{bmatrix}^T \begin{bmatrix} K_6 \\ K_8 \end{bmatrix} \tag{49}$$

Note that it is easier and more convenient to identify two models depending on two parameters each one than to identify a single model depending on four parameters.

Then, by defining:

$$\phi_1(k) = \begin{bmatrix} u_2^2(k) \\ \tau_1(k) + \tau_2(k) \end{bmatrix} \tag{50}$$

$$\theta_1(k) = \begin{bmatrix} K_5 \\ K_7 \end{bmatrix} \tag{51}$$

$$\phi_2(k) = \begin{bmatrix} u_1(k)u_2(k) \\ \tau_1(k) - \tau_2(k) \end{bmatrix} \tag{52}$$

$$\theta_2(k) = \begin{bmatrix} K_6 \\ K_8 \end{bmatrix} \tag{53}$$

it is possible to write (46) and (49) as:

$$y_1(k+1) = \phi_1^T(k)\theta_1(k) \tag{54}$$

$$y_2(k+1) = \phi_2^T(k)\theta_2(k) \tag{55}$$

and then, it is possible to obtain an estimate $\hat{\theta}_i$ for θ_i by using a standard recursive least squares algorithm such as described in Sect. 2.1:

$$\hat{y}_i(n+1) = \phi_i^T(n)\hat{\theta}_i(n) \tag{56}$$

$$K_i(n) = \frac{P_i(n-1)\phi_i(n)}{1 + \phi_i^T(n)P(n-1)\phi_i(n)} \tag{57}$$

$$\hat{\theta}_i(n+1) = \hat{\theta}_i(n) + K_i(n)\left(y_i(n+1) - \hat{y}_i(n-1)\right) \tag{58}$$

$$P_i(n) = \left(I - K_i(n)\phi_i^T(n)\right)P_i(n-1) \tag{59}$$

where $\hat{y}_i(n+1)$ are estimates for $y_i(n+1)$, $K_i(n)$ are the gains and $P_i(n)$ are the covariance matrices.

3 ROS Packages for Identification of Robot Model

This section describes the installation of some packages useful for the implementation of the identification procedure described in Sect. 2. Some of them are not present in a standard installation of ROS and should be installed. Also, some custom packages with our implementation of the identification should be installed.

3.1 Setting up a Catkin Workspace

The packages to be installed for implementing ROS controllers assume an existing catkin workspace. If it does not exist, it can be created with the following commands (assuming a ROS Indigo version):

```
source /opt/ros/indigo/setup.bash
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ~/catkin_ws
catkin_make
source ~/catkin_ws/devel/setup.bash
```

3.2 `ros_control`

The `ros_control` meta-package includes a set of packages to implement generic controllers. It is a rewrite of the `pr2_mechanism` packages to be used with all robots and no just with PR2. This package implements the base architecture of ROS controllers and hence is required for setting up controllers for the robot. In particular, for the identification method proposed here, it is necessary to actuate the robot directly, that is, without any controller, neither an open-loop nor a closed-loop controller. This can be done in ROS by configuring a forward controller, a controller that just replicates its input in its output.

This meta-package includes the following packages:

`control_toolbox`: contains classes that are useful for all controllers, such as PID controllers.
`controller_interface`: implements a base class for interfacing with controllers, the `Controller` class.
`controller_manager`: implements the `ControllerManager` class, which loads, unloads, starts and stops controllers.
`hardware_interface`: base class for implementing the hardware interface, the `RobotHW` and `JointHandle` classes.
`joint_limits_interface`: base class for implementing the joint limits.
`transmission_interface`: base class for implementing the transmission interface.
`realtime_tools`: contains a set of tool that can be used from a hard real-time thread.

The `ros_control` meta-package is not included in the standard ROS desktop installation, hence it should be installed. On Ubuntu, it can be installed from Debian packages with the command:

```
sudo apt-get install ros-indigo-ros-control
```

3.3 `ros_controllers`

This meta-package is necessary to make the `forward_command_controller` and `forward_command_controller` controllers available. The robot is put in open-loop by using the `forward_command_controller` controller, and then the desired input signal can be applied for identification. The `joint_state_controller` controller, which by its name seems an state-space controller in the joint space, but actually it is just a publisher for the values of the position and velocities of the joints. Then, that topic is used to obtain the output of the robot system to be used in the identification.

More specifically, `ros_controllers` includes the following:

`forward_command_controller`: just a bypass from the reference to the control action as they are the same physical variable.

`effort_controllers`: implements effort controllers, that is, SISO controllers in which the control action is the torque (or an equivalent physical variable) applied to the robot joint. There are there types of `effort_controllers`, depending on the type of the reference and controlled variable:

- `effort_controllers/joint_effort_controller`: just a bypass from the reference to the control action as they are the same physical variable.
- `effort_controllers/joint_position_controller`: a controller in which the reference is joint position and the control action is torque. The PID control law is used.
- `effort_controllers/joint_velocity_controller`: a controller in which the reference is joint velocity and the control action is torque. The PID control law is used.

`position_controllers`: implements SISO controllers in which the control action is the position (or an equivalent physical variable) applied to the robot joint. Currently, there is just one type of `position_controllers`:

- `position_controllers/joint_position_controller`: just a bypass from the reference to the control action as they are the same physical variable.

`velocity_controllers`: implements SISO controllers in which the control action is the velocity (or an equivalent physical variable) applied to the robot joint. Currently, there is just one type of `velocity_controllers`:

- `velocity_controllers/joint_velocity_controller`: just a bypass from the reference to the control action as they are the same physical variable.

`joint_state_controller`: implements a sensor which publishes the joint state as a `sensor_msgs/JointState` message, the `JointStateController` class.

The `ros_controllers` meta-package is not included in the standard ROS desktop installation, hence it should be installed. On Ubuntu, it can be installed from Debian packages with the command:

```
sudo apt-get install ros-indigo-ros-controllers
```

3.4 gazebo_ros_pkgs

This is a collection of ROS packages for integrating the ros_control controller architecture with the Gazebo simulator [12], containing the following:

gazebo_ros_control: Gazebo plugin that instantiates the RobotHW class in a DefaultRobotHWSim class, which interfaces with a robot simulated in Gazebo. It also implements the GazeboRosControlPlugin class.

The gazebo_ros_pkgs meta-package is not included in the standard ROS desktop installation, hence it should be installed. On Ubuntu, it can be installed from Debian packages with the command:

```
sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control
```

3.5 twil

This is a meta-package with the package for identification of the Twil robot. It contains an URDF description of the Twil mobile robot [4] and the implementation of the identification and some controllers used for identification and using the identified parameters. More specifically it includes the following packages:

twil_description: URDF description of the Twil mobile robot.
twil_controllers: implementation of a forward controller, a PID controller and a linearizing controller for the Twil mobile robot.
twil_ident: ROS node implementing the recursive least-squares for identification of the parameters of a differential-drive mobile robot.

The twil meta-package can be downloaded and installed in the ROS catkin workspace with the commands:

```
cd ~/catkin_ws/src
wget http://www.ece.ufrgs.br/twil/indigo-twil.tgz
tar -xzf indigo-twil.tgz
cd ~/catkin_ws
catkin_make
source ~/catkin_ws/devel/setup.bash
```

4 Testing the Installed Packages

A simple test for the installation of the packages described in Sect. 3 is performed here.

The installation of the ROS packages can be done by loading the Twil model in Gazebo and launching the computed torque controller with the commands:

```
source /opt/ros/indigo/setup.bash
source ~/catkin_ws/devel/setup.bash
roslaunch twil_controllers joint_effort.launch
```

The robot should appear in Gazebo as shown in Fig. 6.

Then, start the simulation by clicking in the play button in the Gazebo panel, open a new terminal and issue the following commands to move the robot.

```
source /opt/ros/indigo/setup.bash
source ~/catkin_ws/devel/setup.bash
rosrun twil_controllers test_openloop.sh
```

If everything is right, the Twil robot should move for some seconds and then stop, as shown in Fig. 7.

In this simulation, the Twil mobile robot is driven by standard ROS controllers implementing a *bypass* from its reference to its output. This is the equivalent to drive the robot in open-loop, that means, without any controller.

The `effort_controllers/JointEffortController` controller implements just a *bypass* from its input to its output as its input is effort and its output is effort, as well. The example uses one of such controllers in each wheel of the Twil mobile robot, effectively keeping it in open-loop. Hence, the reference applied to the controllers is directly the torque applied to each wheel. Figure 8 shows the computation graph for this example.

The controllers themselves are not shown because they are plugins loaded by the controllers manager and hence they are not individual ROS nodes. The right wheel controller receives its reference through the `/twil/right_wheel_joint _effort_controller_command` topic and the left wheel controller receives its through the `/twil/left_wheel_joint_effort_controller_command`

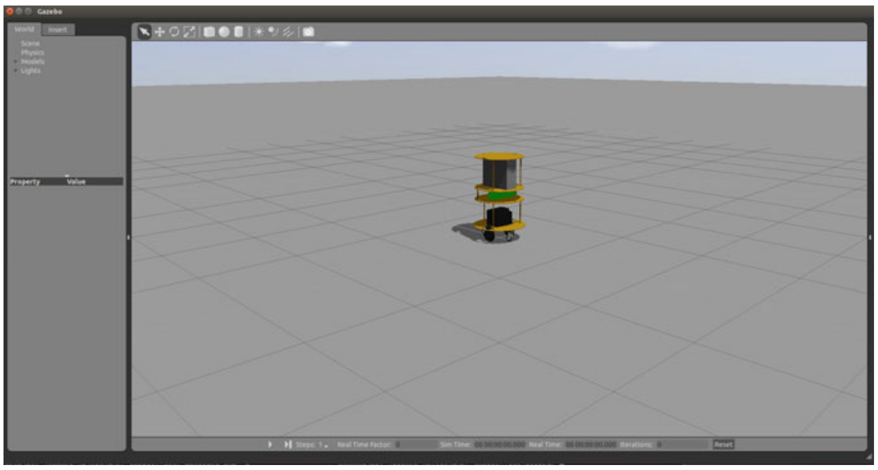


Fig. 6 Twil mobile Robot in Gazebo

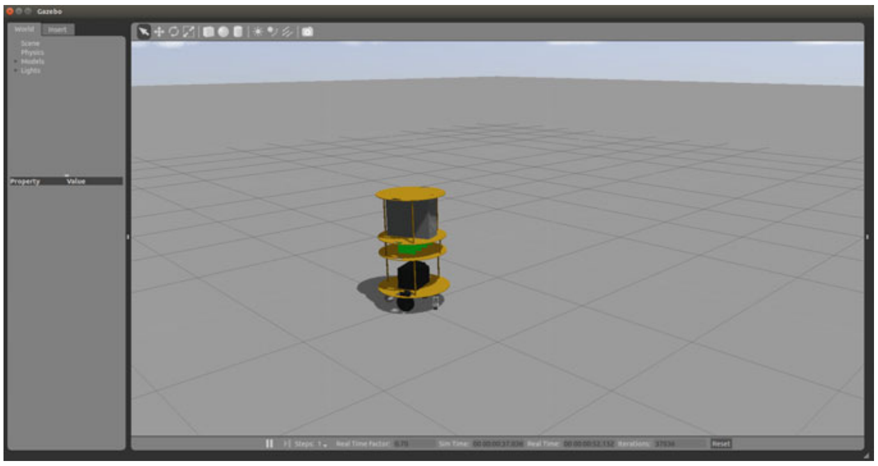


Fig. 7 Gazebo with Twil robot after test motion

topic. The `/joint_states` topic is where the state of the joints (wheels) are published by the `JointStateController` controller. In the next Sections the data published in this topic, will be used to identify the parameters of the Twil mobile robot. For a good identification, adequate signals, as detailed in Sect. 5.3, will be applied to the `/twil/right_wheel_joint_effort_controller_command` and `/twil/left_wheel_joint_effort_controller_command` topics.

The `test_openloop.sh` is a script with an example of how to set the reference for the controllers, in this case, the torque on right and left wheels of the Twil robot. The script just publishes the required values by using the `rostopic` command. In a real application, probably with a more sophisticated controller, those references would be generated by a planning package, such as MoveIt! [24] or a robot navigation package, such as the Navigation Stack [15, 16]. In the case of an identification task, as discussed here, the references for the controllers are generated by a node implementing the identification algorithm.

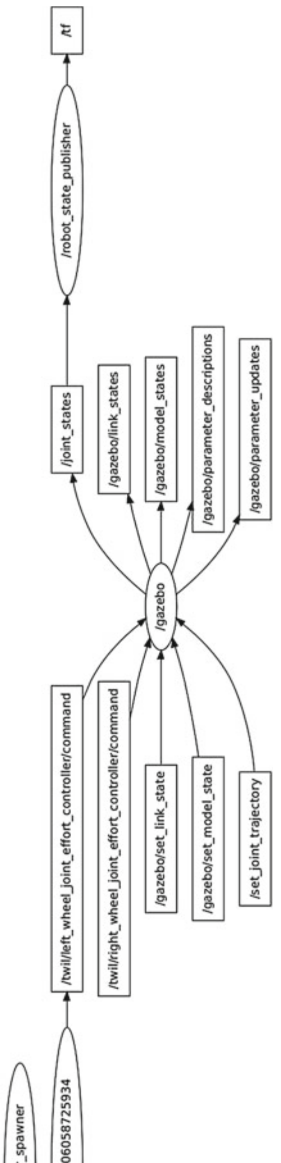

/twil/controller_spawner
/rostopic_5687_1406058725934
/twil/left_wheel_joint_effort_controller/command
/twil/right_wheel_joint_effort_controller/command
/gazebo/set_link_state
/gazebo/set_model_state
/set_joint_trajectory
/gazebo
/joint_states
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/robot_state_publisher
/tf

Fig. 8 Computation graph for Twil in open loop

5 Implementation of Parametric Identification in ROS

In this section, the `twil` ROS meta-package is detailed. This meta-package consists of an URDF description of the Twil mobile robot (`twil_description`), the implementation of some controllers for Twil (`twil_controllers`) and a ROS node for implementing the parametric identification (`twil_ident`). Although the parametric identification launch file is configured for Twil, the source code for the identification is generic and should work directly with any differential-drive mobile robot and with minor modifications for any wheeled mobile robot. Hence, in most cases, the package can be used with any robot by just adapting the launch file.

5.1 `twil_description` Package

The `twil_description` package has the URDF description of the Twil robot. Files in the `xacro` directory contains the files describing the geometric and mass parameters of the many bodies used to compose the Twil robot, while the `meshes` directory holds the STereoLithography (STL) files describing the shapes of the bodies. The files in the `launch` directory are used to load the Twil model in the ROS parameter server. The `twil.launch` file just loads the Twil model in the parameter server and is intended to be used with the actual robot, while the `twil_sim.launch` file loads the Twil model in the parameter server and launches the Gazebo simulator.

It is beyond the scope of this chapter to discuss the modeling of robots in URDF. The reader is directed to the introductory ROS references for learning the details about URDF modeling in general. However, one key point for simulating ROS controllers in Gazebo is to tell it to load the plugin for connecting with `ros_control`. In the `twil_description` package this is done in the top level URDF file, within the `<gazebo>` tag, as shown in Listing 1. See [13] for a detailed description of the plugin configuration.

Listing 1 Plugin description in `twil.urdf.xacro`.

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so" >
    <robotNamespace>/twil</robotNamespace>
    <controlPeriod>0.001</controlPeriod>
  </plugin>
</gazebo>
```

5.2 `twil_controllers` Package

The `twil_controllers` package implements the controllers for Twil. In particular, a Cartesian linearizing controller is implemented as an example of using the results of the parametric identification. The files in the `config` directory specify the parameters for the controllers, such as the joints of the robot associated to the controller, its gains and sampling rate. The `script` directory has some useful scripts for setting the reference for the controllers and can be used for testing them. Note that although only the `CartLinearizingController` is implemented in this package, the Twil can use other controllers, as those implemented in the `ros_controllers` package. In particular, the controller used for identification (`effort_controllers/JointEffortController`) comes from the `ros_controllers` package.

The file in the `src` directory are the implementation of controllers for Twil, in particular `CartLinearizingController`, derived from the `Controller` class, while the `include` directory holds the files with the declarations of those classes. The `twil_controllers_plugins.xml` file specifies that the classes implementing the controllers are plugins for the ROS controller manager. The files in the `launch` directory are used to load and start the controllers with the respective configuration files. The detailed description of the implementation of controllers is not the scope of this chapter. See [13] for a detailed discussion about the implementation of controllers in ROS.

5.3 `twil_ident` Package

The `twil_ident` package contains a ROS node implementing the parameter identification procedure described in Sect. 2.1. Again, the source code is in the `src` directory and there is a launch file in the `launch` directory which is used to load and start the node.

The identification node can be launched through the command:

```
roslaunch twil_ident ident.launch
```

The launch file is shown in Listing 2. Initially, there are remaps of the topic names used as reference for the controllers for the right and left wheels, then, another launch file, which loads Gazebo with the Twil simulation is called. The next step is the loading of controller parameters from the configuration file `effort_control.yaml` in the parameter server and the controller manager node is spawn for loading the controllers for both wheels and the `joint_state_controller` to publish the robot state. Finally the identification node is loaded and the identification procedure starts.

Listing 2 Launch file `ident.launch`.

```
<launch>
  <remap from="/twil/left_wheel_joint_effort_controller/command"
    to="/twil/left_wheel_command"/>
  <remap from="/twil/right_wheel_joint_effort_controller/command"
    to="/twil/right_wheel_command"/>

  <include file="$(find twil_description)/launch/twil_sim.launch"/>
```

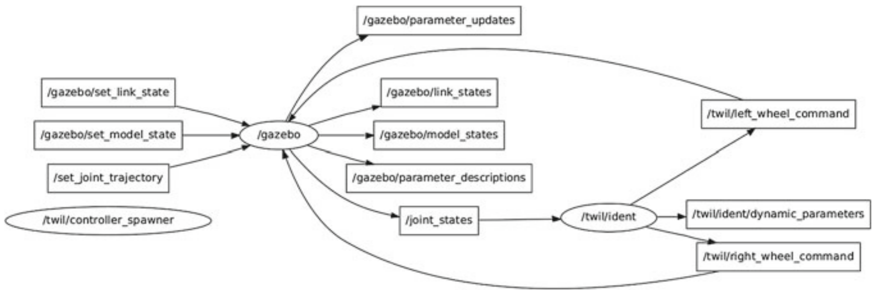


Fig. 9 Computation graph for parameter identification

```
  <rosparam file="$(find twil_controllers)/config/effort_control.yaml" command="load"/>

  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
    output="screen" ns="/twil"
    args="joint_state_controller left_wheel_joint_effort_controller \
      right_wheel_joint_effort_controller"/>

  <node name="ident" ns="/twil" pkg="twil_ident" type="ident" output="screen">
    <remap from="ident/left_wheel_command" to="left_wheel_command"/>
    <remap from="ident/right_wheel_command" to="right_wheel_command"/>
  </node>
</launch>
```

Figure 9 shows the computation graph used for the identification. The difference with respect to the computation graph in Fig. 8 is that torques to be applied to the wheels are published on the `/twil/right_wheel_command` and `/twil/left_wheel_command` by the `ident` node. These topics are the same described in Sect. 4, but had their names changed to force the connection between the `/gazebo` and the `/twil/ident` nodes.

The `/twil/ident` node implements the parameter identification and subscribes to the `/joint_states` topic in order to obtain the joint (wheel) velocities necessary for the identification algorithm and publishes the torque commands for the wheels in the `/twil/right_wheel_command` and `/twil/left_wheel_command` topics. The torques follow a Pseudo Random Binary Sequence (PRBS) pattern [21] in order to ensure the persistent excitation for a good identification of the parameters.

The identified parameters and the respective diagonal of the covariance matrix are published on the `/twil/ident/dynamic_parameters` topic. Those values can be used offline by controllers, for the implementation of non-adaptive controllers or can be used online to implement adaptive controllers. In this case, the controller can subscribe to the `/twil/ident/dynamic_parameters` topic to receive updates of the identified parameters and the respective diagonal of the covariance matrix, which is a measure of confidence on the parameter estimation and hence can be used to decide if the adaptation should be shut-off or not. This way, the mass and inertia parameters of the robot would be adjusted on-line for variations due to changes in workload, for example. In order to ensure the reliability of the identified values, parameters would be changed only if the associated covariance is small enough.

The `Ident` class is shown in Listing 3. The private variable members of the `Ident` class are:

`node_`: the ROS nodeidentifier
`jointStateSubscriber`: ROS topic subscriber to receive the joint state
`dynParamPublisher`: ROS topic to publish the identified parameters
`leftWheelCommandPublisher`: ROS topic to publish the reference for the left wheel controller
`dynParamPublisher`: ROS topic to publish the reference for the right wheel controller
`nJoints`: the number of joints (wheels) of the robot
`u`: vector of joint velocities
`thetaEst1`: vector of estimated parameters $\hat{\theta}_1$
`thetaEst2`: vector of estimated parameter $\hat{\theta}_2$
`P1`: covariance of the error in estimates $\hat{\theta}_1$
`P2`: covariance of the error in estimates $\hat{\theta}_2$
`prbs`: vector of PRBS sequences used as input to the robot for identification
`lastTime`: time for the last identification iteration.

Listing 3 `Ident` class.

```
class Ident
{
        public:
                Ident(ros::NodeHandle node);
                ~Ident(void);
                void setCommand(void);

        private:
                ros::NodeHandle node_;

                ros::Subscriber jointStatesSubscriber;
                ros::Publisher dynParamPublisher;
                ros::Publisher leftWheelCommandPublisher;
                ros::Publisher rightWheelCommandPublisher;

                const int nJoints;

                Eigen::VectorXd u;
                Eigen::VectorXd thetaEst1;
                Eigen::VectorXd thetaEst2;
                Eigen::MatrixXd P1;
                Eigen::MatrixXd P2;

                std::vector<Prbs> prbs;

                ros::Time lastTime;

                void jointStatesCB(const sensor_msgs::JointState::ConstPtr &jointStates);
                void resetCovariance(void);
};
```

The `jointStatesCB()` function is the callback for receiving the state of the robot and running the identifier iteration, as shown in Listing 4.

Listing 4 `JointStatesCB()` function.

```
void Ident::jointStatesCB(const sensor_msgs::JointState::ConstPtr &jointStates)
{
        ros::Duration dt=jointStates->header.stamp-lastTime;
        lastTime=jointStates->header.stamp;

        Eigen::VectorXd y=-u;   //y(k+1)=(u(k+1)-u(k))/dt

        Eigen::VectorXd Phi1(nJoints);
        Eigen::VectorXd Phi2(nJoints);
        Phi1[0]=u[1]*u[1];      // u2^2(k)
        Phi2[0]=u[0]*u[1];      // u1(k)*u2(k)

        Eigen::VectorXd torque(nJoints);
        for(int i=0;i < nJoints;i++)
        {
                u[i]=jointStates->velocity[i];          // u(k+1)
                torque[i]=jointStates->effort[i];       // torque(k)
        }

        y+=u;
        y/=dt.toSec();

        Phi1[1]=torque[0]+torque[1];
        Phi2[1]=torque[0]-torque[1];

        double yEst1=Phi1.transpose()*thetaEst1;
        Eigen::VectorXd K1=P1*Phi1/(1+Phi1.transpose()*P1*Phi1);
        thetaEst1+=K1*(y[0]-yEst1);
        P1-=K1*Phi1.transpose()*P1;

        double yEst2=Phi2.transpose()*thetaEst2;
        Eigen::VectorXd K2=P2*Phi2/(1+Phi2.transpose()*P2*Phi2);
        thetaEst2+=K2*(y[1]-yEst2);
        P2-=K2*Phi2.transpose()*P2;

        std_msgs::Float64MultiArray dynParam;
        for(int i=0;i < nJoints;i++)
        {
                dynParam.data.push_back(thetaEst1[i]);
                dynParam.data.push_back(thetaEst2[i]);
        }
        for(int i=0;i < nJoints;i++)
        {
                dynParam.data.push_back(P1(i,i));
                dynParam.data.push_back(P2(i,i));
        }
        dynParamPublisher.publish(dynParam);
}
```

In the callback, first of all the time interval since last call is computed (`dt`), then the $\phi_1(t)$ and $\phi_2(t)$ vectors are assembled in variables `Phi1` and `Phi2` and the system output $y(t+1)$ is assembled. Then, the parameter estimates and their covariances are computed from (56)–(59) and finally, the parameter estimates and the covariance matrix diagonal are published in `dynParamPublisher`.

The `main()` function of the `ident` node is shown in Listing 5. It is just a loop running at 100 Hz publishing torques for the robot wheels through the `setCommand()` function.

Listing 5 `ident` node `main()` function.

```
int main(int argc,char* argv[])
{
        ros::init(argc,argv,"twil_ident");
        ros::NodeHandle node;

        Ident ident(node);

        ros::Rate loop(100);
        while(ros::ok())
        {
                ident.setCommand();

                ros::spinOnce();
                loop.sleep();
        }
        return 0;
}
```

Torques to be applied to the robot wheels are published by the `setCommand()` function shown in Fig. 6. A PRBS signal with amplitude switching between -5 and 5 Nm is applied to each wheel.

Listing 6 `setCommand()` function.

```
void Ident::setCommand(void)
{
        std_msgs::Float64 leftCommand;
        std_msgs::Float64 rightCommand;
        leftCommand.data=10.0*prbs[0]-5.0;
        rightCommand.data=10.0*prbs[1]-5.0;
        leftWheelCommandPublisher.publish(leftCommand);
        rightWheelCommandPublisher.publish(rightCommand);
}
```

While the identification procedure is running, the estimates of parameters K_5, K_6, K_7 and K_8 and their associated covariances are published as a vector in the `/twil/ident/dynamic_parameters` topic. Hence, the results of estimation can be observed by monitoring this topic with the command:

```
rostopic echo /twil/ident/dynamic_parameters
```

The results for the estimates of parameters K_5, K_6, K_7 and K_8 can be viewed in Figs. 10, 11, 12 and 13, respectively. Note that for a better visualization the time horizon for Figs. 10 and 11 is not the same in all figures.

Figures 14, 15, 16 and 17 show the evolution of the diagonal of the covariance matrix related to the K_5, K_6, K_7 and K_8 parameters, respectively.

Although the identified values remain changing over time due to noise, it is possible to consider that they converge to an average value and stop the identification algorithm. The resulting values are shown in Table 1, with the respective diagonal of the covariance matrix. Those values were used for the implementation of the feedback linearization controller.

Given the results in Table 1 and recalling the model (36) and (37)–(40), the identified model of the Twil mobile robot is:

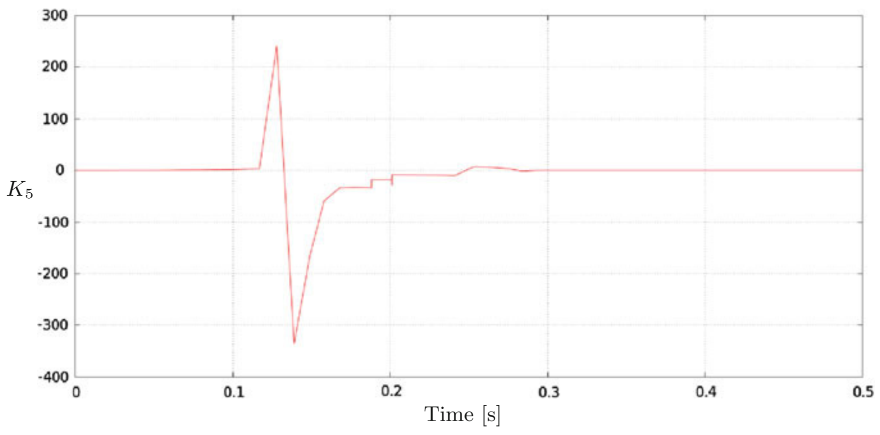


Fig. 10 Evolution of the estimate of the K_5 parameter

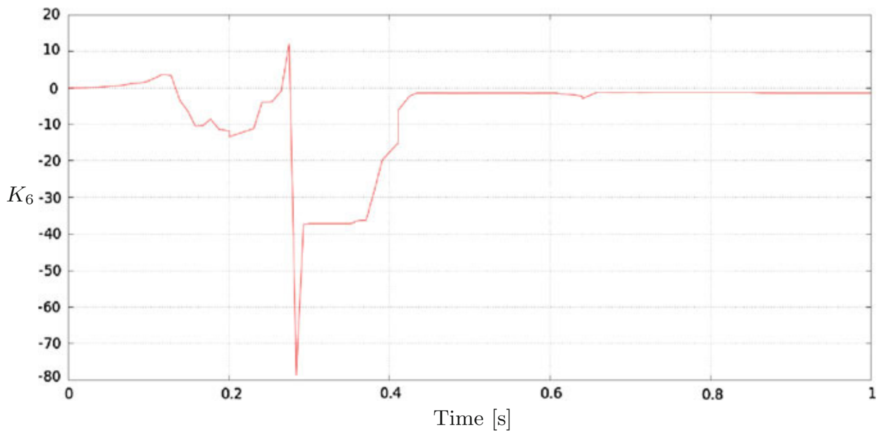


Fig. 11 Evolution of the estimate of the K_6 parameter

$$\begin{cases} \dot{x} = \begin{bmatrix} \cos\theta_c & 0 \\ \sin\theta_c & 0 \\ 0 & 1 \end{bmatrix} u \\ \dot{u} = \begin{bmatrix} 0 & 0.00431 \\ 0.18510 & 0 \end{bmatrix} \begin{bmatrix} u_1 u_2 \\ u_2^2 \end{bmatrix} + \begin{bmatrix} 18.7807 & 18.7807 \\ -14.3839 & 14.3839 \end{bmatrix} \tau \end{cases} \tag{60}$$

In principle, it sounds pointless to identify the parameters of a simulated robot. However, the simulation performed by Gazebo is based on the Open Dynamics Engine (ODE) library [23] with parameters derived from a URDF description of the robot, which is more detailed than the model used for identification and control. Due to the model non-linearities and the richness of details of the URDF description

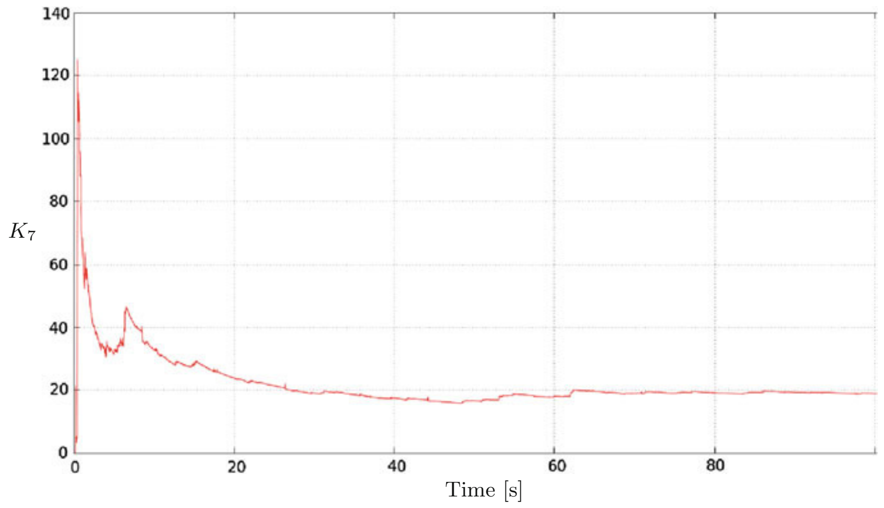


Fig. 12 Evolution of the estimate of the K_7 parameter

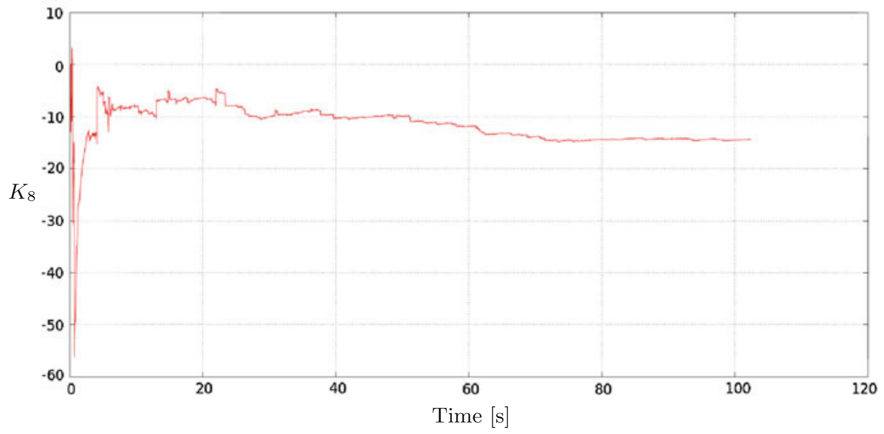


Fig. 13 Evolution of the estimate of the K_8 parameter

it is not easy to compute the equivalent parameters to be used in a closed form model useful for control design. Hence, those parameters are identified. Note that this situation is analogous to a real robot, where the actual parameters are not the same as the theoretical ones due to many details not being modeled.

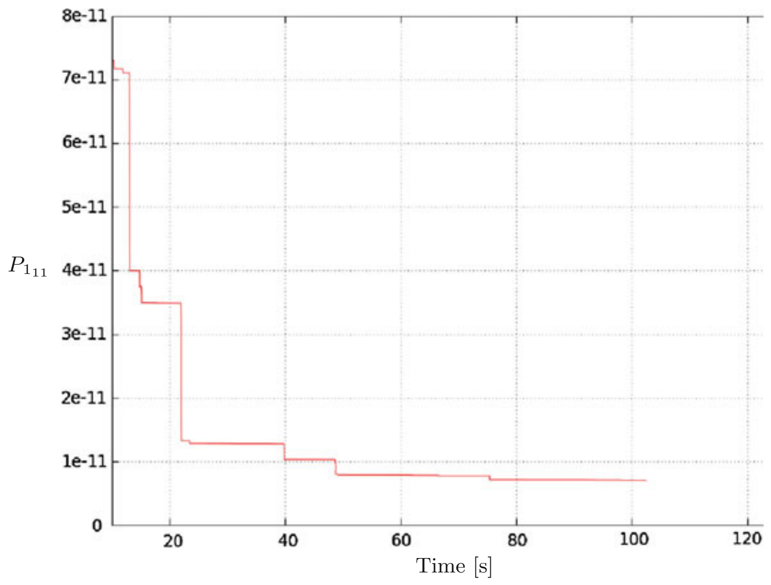


Fig. 14 Diagonal of the covariance matrix related to the K_5 parameter

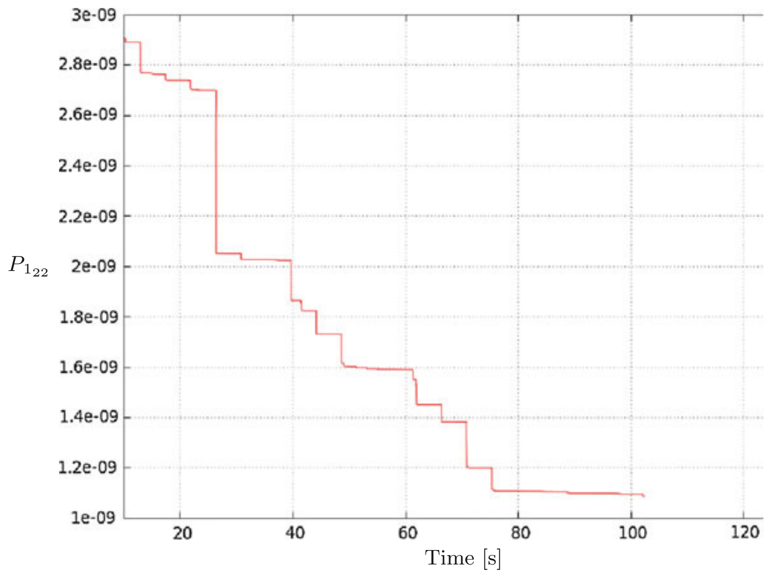


Fig. 15 Diagonal of the covariance matrix related to the K_6 parameter

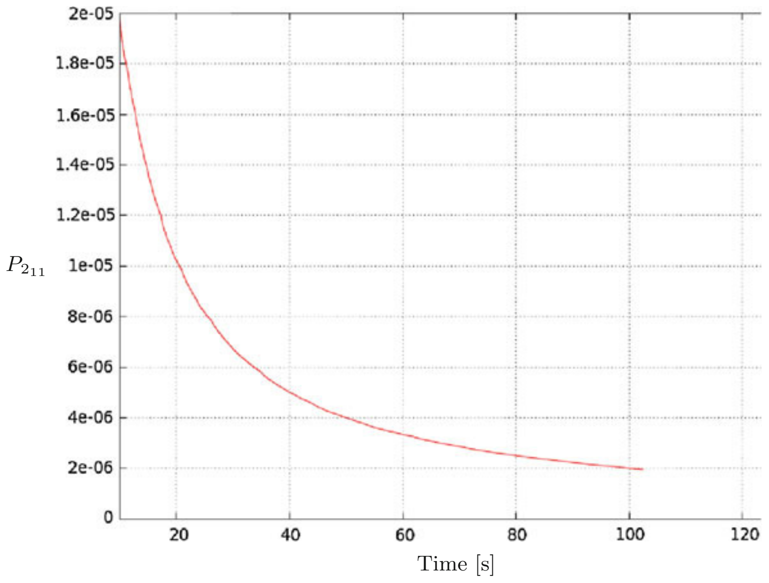


Fig. 16 Diagonal of the covariance matrix related to the K_7 parameter

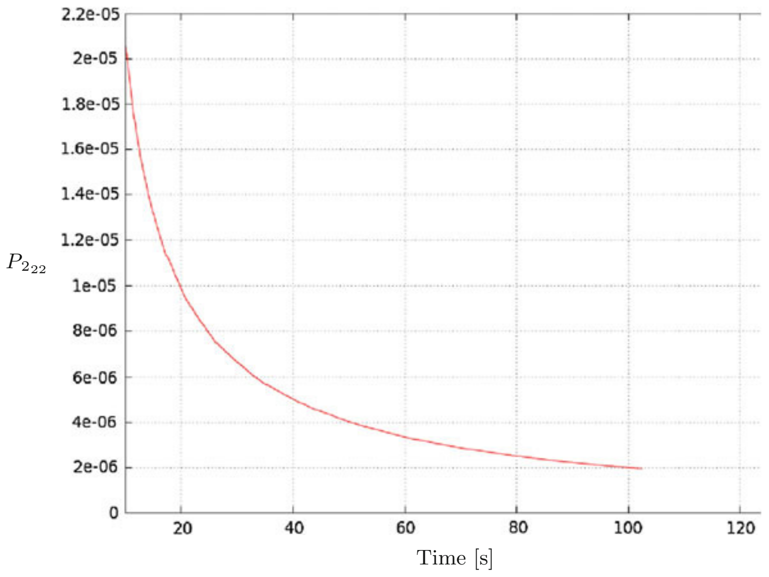


Fig. 17 Diagonal of the covariance matrix related to the K_8 parameter

Table 1 Twil parameters obtained by identification

Parameter	Value	Covariance diagonal
K_5	0.00431	7.0428×10^{-12}
K_6	0.18510	1.0870×10^{-09}
K_7	18.7807	1.9617×10^{-06}
K_8	-14.3839	1.9497×10^{-06}

6 Controller Design

The model (60) can be used for the design of controllers. Although now all parameters are known, it is still a non-linear model and a cascade of the dynamics and the kinematics as shown in Fig. 4. Also, as discussed in Sect. 1 there are in the literature many publications dealing with the control of mobile robots using only the kinematic model. Furthermore, the non-holonomic constraints associated to mobile robots, with exception of the ominidirectional ones, are associated to the first expression of (60), while the second expression is a holonomic system. For this reason, most difficulties in designing a controller for a mobile robot are related to its kinematic model and not to its dynamic model.

In order to build-up on the many methods developed to control mobile robots based on the kinematic model alone, the control strategy proposed here considers the kinematics and the dynamics of the robot in two independent steps. See [6, 7] for a control approach dealing with the complete model of the robot in a single step.

The dynamics of the robot is described by the second expression of (60) and has the form:

$$\dot{u} = f(u) + F\tau \tag{61}$$

and a state feedback linearization [11, 13] with the control law:

$$\tau = F^{-1}(\nu - f(u)) \tag{62}$$

where ν is a new control input, leads to:

$$\dot{u} = \nu \tag{63}$$

which is a linear, decoupled system. That means that each element of u is driven by a single element of ν or $\dot{u}_i = \nu_i$.

For differential-drive mobile robot such as Twil the elements of $u = \begin{bmatrix} v & \omega \end{bmatrix}^T$, are the linear and angular velocities of the robot. For other types of wheeled mobile robots, the number and meaning of the elements of u would not be the same, but (63) would still have the same form, eventually with larger vectors. Anyway, the transfer function for each element of (63) is:

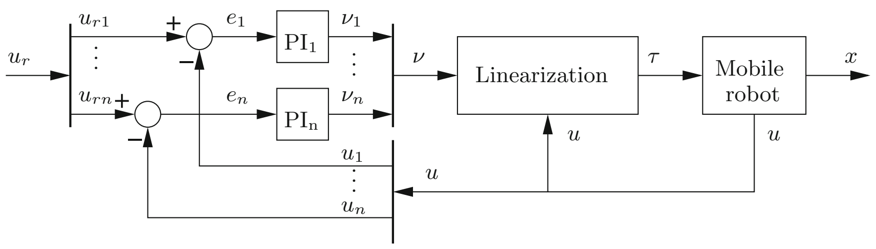


Fig. 18 Block diagram of the controller for the dynamics of the mobile robot

$$G_i(s) = \frac{U_i(s)}{V_i(s)} = \frac{1}{s} \tag{64}$$

In other words, by using the feedback (62), the system (61) is transformed in a set of independent systems, each one with a transfer function equal to $G_i(s)$. Each one of these systems can be controlled by a PI controller, then:

$$\nu_i = K_{pi} e_i + K_{ii} \int e_i dt \tag{65}$$

where $e_i = u_{ri} - u_i$, u_{ri} is the reference for the i-th element of u and K_{pi} and K_{ii} are the proportional and integral gains, respectively.

The transfer function of PI controller (65) is:

$$C_i(s) = \frac{V_i(s)}{E_i(s)} = \frac{K_{pi} s + K_{ii}}{s} \tag{66}$$

Then, by remembering that $E_i(s) = U_{ri}(s) - U_i(s)$ and using (64) and (66), it is possible to write the closed-loop transfer function as:

$$H_i(s) = \frac{U_i(s)}{U_{ri}(s)} = \frac{C_i(s)G_i(s)}{1 + C_i(s)G_i(s)} = \frac{K_{pi} s + K_{ii}}{s^2 + K_{pi} s + K_{ii}} \tag{67}$$

Figure 18 shows the block diagram of the proposed controller, which is implemented by using (62) and (65). Note that (65) can be implemented by using the `Pid` class already implemented in the `control_toolbox` ROS package, by just making the derivative gain equal to zero.

The performance of the controller is determined by the characteristic polynomial (the denominator) of (67). For canonical second order systems, the characteristic polynomial is given by:

$$s^2 + 2\xi\omega_n s + \omega_n^2 \tag{68}$$

where ξ is the damping ratio and ω_n is the natural frequency.

Hence, it is easy to see that $K_{pi} = 2\xi\omega_n$ and $K_{ii} = \omega_n^2$. Furthermore, the time it takes to the control system to settle to a precision of 1% is given by [19]:

$$T_s = \frac{-\ln 0.01}{\xi\omega_n} = \frac{4.6}{\xi\omega_n} \tag{69}$$

Therefore, by choosing the damping ration and the settling time required for each PI controller it is possible to compute K_{pi} and K_{ii}.

Again, for the Twil mobile robot and all differential-drive robots, $u = \begin{bmatrix} v \ \omega \end{bmatrix}^T$, hence, for this type of robot there are two PI controllers: one for controlling the linear velocity and other for controlling the angular velocity. In most cases it is convenient to tune both controllers for the same performance, therefore, as the system model is the same, the controller gains would be the same. In robotics, it is usual to set $\xi = 1.0$, to avoid overshoot, and T_s is the time required for the controlled variable to converge to within 1% of error of the reference. In a first moment one may think that T_s should be set to a very small value. However, the trade-off here is the control effort. A very small T_s would require a very large ν and hence very large torques on the motors, probably above what they are able to provide. Therefore, T_s should be set to a physically sensible value. By making $T_s = 50\,\text{ms}$, from (69):

$$\omega_n = \frac{4.6}{\xi T_s} = \frac{4.6}{50 \times 10^{-3}} = 92\,\text{rad/s} \tag{70}$$

and

$$K_{p1} = K_{p2} = 2\xi\omega_n = 184 \tag{71}$$
$$K_{i1} = K_{i2} = \omega_n^2 = 8464 \tag{72}$$

By using the above gains, the controls system shown in Fig. 18 ensures that u will converge to u_r in a time T_s. Then, by commanding u_r it is possible to steer the mobile robot to the desired pose. To do this in a robust way, it is necessary to have another control loop using the robot pose as feedback signal. By supposing that T_s is selected to be much faster than the pose control loop (at least five times faster), the dynamics (67) can be neglected and the resulting system (equivalent to see the system in Fig. 18 as a single block) model can be written as:

$$\dot{x} = \begin{bmatrix} \cos\theta_c & 0 \\ \sin\theta_c & 0 \\ 0 & 1 \end{bmatrix} u_r \tag{73}$$

It is important to note that using (73) for the design of the pose controller is not the same as using only the kinematic model (first expression of (36)). Although the equations are the same, and then, the same control methods can be used, now there is an internal control loop (Fig. 18) that forces the commanded u_r to be effectively applied to the robot despite the dynamics of the robot.

The pose controller used here follows the one proposed in [1] and is non-linear controller based on the Lyapunov theory and a change of the robot model to polar coordinates. Also, as most controllers based on Lyapunov theory, it is assumed that the system should converge to its origin. However, as it is interesting to be able to stabilize the robot at any pose $x_r = \left[x_{cr}\ y_{cr}\ \theta_{cr}\right]^T$, the following coordinate change [2] is used to move the origin of the new system to the reference pose::

$$\bar{x} = \begin{bmatrix} \bar{x}_c \\ \bar{y}_c \\ \bar{\theta}_c \end{bmatrix} = \begin{bmatrix} \cos\theta_{cr} & \sin\theta_{cr} & 0 \\ -\sin\theta_{cr} & \cos\theta_{cr} & 0 \\ 0 & 0 & 1 \end{bmatrix} (\mathbf{x} - \mathbf{x_r}) \tag{74}$$

By using a change to polar coordinates [5] given by:

$$e = \sqrt{\bar{x}_c^2 + \bar{y}_c^2} \tag{75}$$

$$\psi = \text{atan2}(\bar{y}_c,\ \bar{x}_c) \tag{76}$$

$$\alpha = \bar{\theta}_c - \psi \tag{77}$$

the model (73) can be rewritten as:

$$\begin{cases} \dot{e} = \cos\alpha u_{r1} \\ \dot{\psi} = \dfrac{\sin\alpha}{e} u_{r1} \\ \dot{\alpha} = -\dfrac{\sin\alpha}{e} u_{r1} + u_{r2}. \end{cases} \tag{78}$$

which is only valid for differential-drive mobile robots. For a similar procedure for other configurations of wheeled mobile robots see [14].

Then, given a candidate to Lyapunov function:

$$V = \frac{1}{2}\left(\lambda_1 e^2 + \lambda_2 \alpha^2 + \lambda_3 \psi^2\right), \tag{79}$$

with $\lambda_i > 0$. Its time derivative is:

$$\dot{V} = \lambda_1 e\dot{e} + \lambda_2 \alpha\dot{\alpha} + \lambda_3 \psi\dot{\psi} \tag{80}$$

By replacing $\dot{e}$, $\dot{\alpha}$ and $\dot{\psi}$ from (78):

$$\dot{V} = \lambda_1 e \cos\alpha u_{r1} - \lambda_2 \alpha \frac{\sin\alpha}{e} u_{r1} + \lambda_2 \alpha u_{r2} + \lambda_3 \psi \frac{\sin\alpha}{e} u_{r1} \tag{81}$$

and, it can be shown that the input signal:

$$u_{r1} = -\gamma_1 e \cos\alpha \tag{82}$$

$$u_{r2} = -\gamma_2\alpha - \gamma_1 \cos\alpha \sin\alpha + \gamma_1 \frac{\lambda_3}{\lambda_2} \cos\alpha \frac{\sin\alpha}{\alpha}\psi \tag{83}$$

leads to:

$$\dot{V} = -\gamma_1\lambda_i e^2 \cos^2\alpha - \gamma_2\lambda_2\alpha^2 \leq 0 \tag{84}$$

which, along with the continuity of V, assures the system stability. However, the convergence of system state to the origin still needs to be proved. See [22] for other choices of u_r leading to $\dot{V} \leq 0$.

Given that V is lower bounded, that V is non-increasing, as $\dot{V} \leq 0$ and that $\dot{V}$ is uniformly continuous, as $\ddot{V} < \infty$, the Barbalat lemma [20] assures that $\dot{V} \to 0$ which implies $\alpha \to 0$ and $e \to 0$. It remains to be shown that ψ also converges to zero.

To prove that $\psi \to 0$, consider the closed loop system obtained by applying (82)–(83) to (78), given by:

$$\dot{e} = -\gamma_1 e \cos^2\alpha \tag{85}$$

$$\dot{\psi} = -\gamma_1 \sin\alpha \cos\alpha \tag{86}$$

$$\dot{\alpha} = -\gamma_2\alpha + \gamma_1 \frac{\lambda_3}{\lambda_2}\psi \frac{\sin\alpha}{\alpha} \cos\alpha \tag{87}$$

Given that ψ is bounded and from (86) it can be concluded that $\dot{\psi}$ is also bounded, it follows that ψ is uniformly continuous, which implies that $\dot{\alpha}$ is uniformly continuous as well, since $\ddot{\alpha} < \infty$. Then, it follows from the Barbalat lemma that $\alpha \to 0$ implies $\dot{\alpha} \to 0$. Hence, from (87) it follows that $\psi \to 0$. Therefore, (82)–(83) stabilize the system (78) at its origin.

Note that although the open loop system described by (78) has a mathematical indetermination due to the e in denominator, the closed-loop system (85)–(87) is not undetermined and hence can converge to zero. The indetermination in (78) is not due to a fundamental physical constraint as it not present in the original model (73), but was artificially created by the coordinate change (75)–(77). It is a well-known result from [8] that a non-holonomic mobile robot can not be stabilized to a pose by using a smooth, time-invariant feedback. Here, those limitations are overcame by using a discontinuous coordinate change. Also, the input signals (82)–(83) can be always computed as $\frac{\sin(\alpha)}{\alpha}$ converges to 1 as α converges to 0. Furthermore, (78) is just an intermediate theoretical step to obtain the expressions for the input signals (82)–(83). There is no need to actually compute it. If using the real robot, there is no need to use the model for simulation and if using a simulated robot, it can be simulated by the Cartesian model (73), which does not present any indetermination.

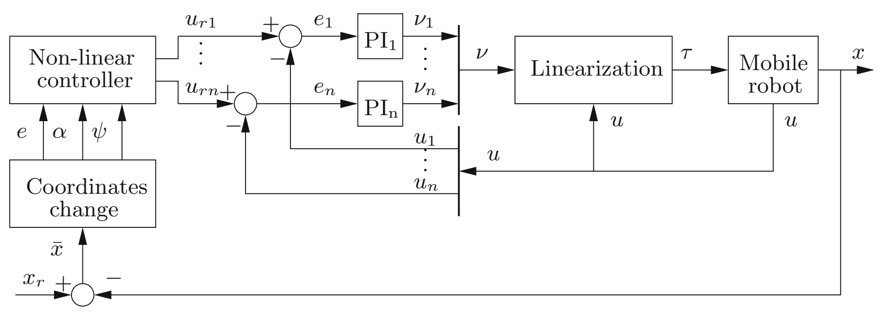


Fig. 19 Block diagram of the pose controller considering the kinematics and the dynamics of the mobile robot

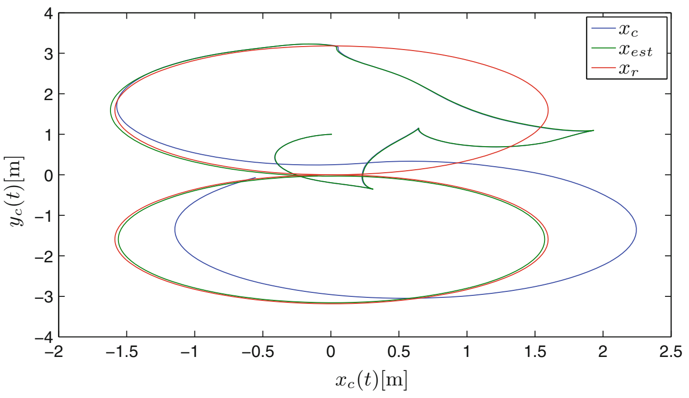


Fig. 20 Controller performance in the Cartesian plane

Figure 19 shows a block diagram of the whole pose control system, considering the kinematics and the dynamics of the robot. Note that although this control system can theoretically make the robot converge to any x_r pose departing from any pose, without a given trajectory (it is generated implicitly by the controller), in practice it is not a good idea to force x_r too far from the current robot position, as this could lead to large torque signals which can saturate the actuators. Hence, in practice x_r should be a somewhat smooth reference trajectory and the controller would force the robot to follow it.

Figure 20 shows the performance of the controller with the proposed controller while performing an 8 path. The red line is the reference path, starting at $x_r = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$, the blue line is the actual robot position, starting at $x_{est} = \begin{bmatrix}0 & 1 & 0\end{bmatrix}^T$, and the read line is the robot position estimated by odometry, starting at $x_{est} = \begin{bmatrix}0 & 1 & 0\end{bmatrix}^T$. Note that the starting position of the robot is outside the reference path and that the controller forces the convergence to the reference path and then the reference path is followed. Also note that the odometry error increases over time, but that is a pose estimation problem, which is not addressed in this chapter. The controller forces the estimated robot position to follow the reference. Unfortunately, using just odometry, the pose estimation is not good and after some time it does not reflect the actual robot pose.

Figure 21 shows the robot orientation. Again, the red line is the reference orientation, the blue line is the actual robot orientation and the red line is the robot orientation.

An adaptive version of the controllers shown in Fig. 19 can be built, by using simultaneously the proposed controller and the identification module, as shown in Fig. 22. Then, the mass and inertia parameters of the robot would be adjusted on-line for variations due to changes in workload, for example.

Note that in this case, a PRBS pattern of torques is not necessary, as the control input generated by the controller is used. The noise and external perturbations should provide enough richness in the signal for a good identification. In extreme cases, the

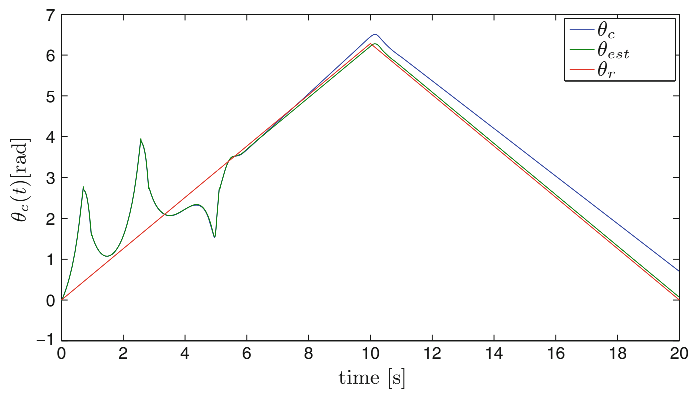


Fig. 21 Controller performance. Orientation in time

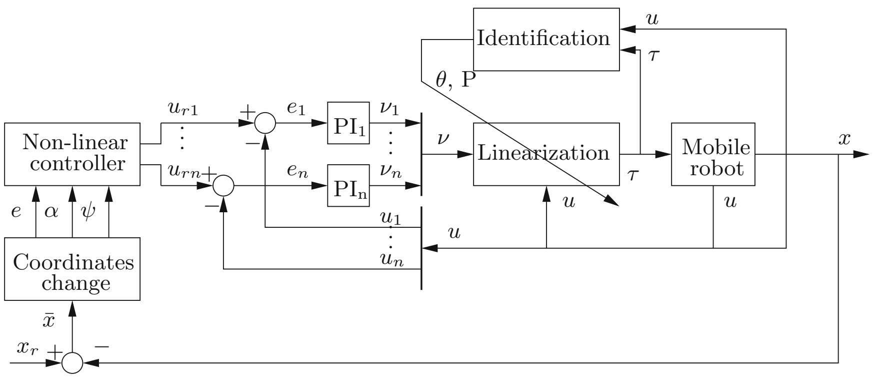


Fig. 22 Block diagram of the adaptive controller

persistence of excitation of the control signal could be tested for and the identification turned off while it is not rich enough for a good identification.

7 Conclusion

This chapter presented the identification of the dynamic model of a mobile robot in ROS. This model is the departure point for the design of advanced controllers for mobile robots. While for small robots, it is possible to neglect the dynamics and design a controller based only on the kinematic model, for larger or faster robots the controller should consider the dynamic effects. However, the theoretical determination of the parameters of the dynamic model is not easy due to the many parts of the robot and uncertainty in the assembly of the robot. Then, the online identification of those parameters enables the overcame of those difficulties.

The packages used for such identification were described and a complete example, from modeling, parameterizing of the model until the computation the of numerical values for the unknown parameters and the write down of the model with all its numerical values was shown.

The identification method was implemented as an online recursive algorithm, which enable its use in an adaptive controller, where new estimates of the parameters of the model are used to update the parameters of the controller, in a strategy known as indirect adaptive control [17].

The results of the identification procedure were used to design a controller based on the dynamics and the kinematics of the mobile robot Twil and adaptive version of that controller was proposed.

References

1. Aicardi, M., G. Casalino, A. Bicchi, and A. Balestrino. 1995. Closed loop steering of unicycle-like vehicles via lyapunov techniques. *IEEE Robotics and Automation Magazine* 2 (1): 27–35.
2. Alves, J.A.V., and W.F. Lages. 2012. Mobile robot control using a cloud of particles. In *Proceedings of 10th International IFAC Symposium on Robot Control*. pp. 417–422. International Federation of Automatic Control, Dubrovnik, Croatia. doi:10.3182/20120905-3-HR-2030.00096.
3. Åström, K.J., and B. Wittenmark. 2011. *Computer-Controlled Systems: Theory and Design*, 3rd ed., Dover Books on Electrical Engineering, Dover Publications.
4. Barrett Technology Inc. 2011. *Cambridge*. MA: WAM User Manual.
5. Barros, T.T.T., and W.F. Lages. 2012. Development of a firefighting robot for educational competitions. In *Proceedings of the 3rd Intenational Conference on Robotics in Education*. Prague, Czech Republic.
6. Barros, T.T.T., and W.F. Lages. 2014. A backstepping non-linear controller for a mobile manipulator implemented in the ros. In *Proceedings of the 12th IEEE International Conference on Industrial Informatics*. IEEE Press, Porto Alegre, RS, Brazil.
7. Barros, T.T.T. and W.F. Lages. 2014. A mobile manipulator controller implemented in the robot operating system. In *Proceedings for the Joint Conference of 45th International Symposium*

on Robotics and 8th German Conference on Robotics. pp. 121–128. VDE Verlag, Munich, Germany, iSBN 978-3-8007-3601-0.

8. Brockett, R.W. 1982. *New Directions in Applied Mathematics*. New York: Springer.
9. Campion, G., G. Bastin, and B. D'Andréa-Novel. 1996. Structural properties and classification of kinematic and dynamical models of wheeled mobile robots. *IEEE Transactions on Robotics and Automation* 12 (1): 47–62. Feb.
10. Goodwin, G.C., and K.S. Sin. 1984. *Adaptive Filtering, Prediction and Control*. Prentice-Hall Information and System Sciences Series. Englewood Cliffs, NJ: Prentice-Hall Inc.
11. Isidori, A. 1995. *Nonlinear Control Systems*, 3rd ed. Berlin: Springer.
12. Koenig, N., and A. Howard. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*. vol. 3, pp. 2149–2154. IEEE Press, Sendai, Japan.
13. Lages, W.F. 2016. Implementation of real-time joint controllers. In *Robot Operating System (ROS): The Complete Reference (Volume 1), Studies in Computational Intelligence*, vol. 625, ed. A. Koubaa, 671–702. Switzerland: Springer International Publishing.
14. Lages, W.F., and E.M. Hemerly. 1998. Smooth time-invariant control of wheeled mobile robots. In *Proceedings of The XIII International Conference on Systems Science*. Technical University of Wrocław, Wrocław, Poland.
15. Marder-Eppstein, E. 2016. Navigation Stack. http://wiki.ros.org/navigation.
16. Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and K. Konolige. 2010. The office marathon: Robust navigation in an indoor office environment. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 300–307. IEEE Press, Anchorage, AK.
17. Narendra, K.S., and A.M. Annaswamy. 1989. *Stable Adaptive Systems*. Englewood Cliffs, NJ: Prentice-Hall Inc.
18. Nguyen-Tuong, D., and J. Peters. 2011. Model learning for robot control: a survey. *Cognitive Processing* 12(4), 319–340 (2011). http://dx.doi.org/10.1007/s10339-011-0404-1.
19. Ogata, K. 1970. *Modern Control Engineering*. Englewood Cliffs, NJ, USA: Prentice-Hall.
20. Popov, V.M. 1973. *Hyperstability of Control Systems, Die Grundlehren der matematischen Wissenshaften*, vol. 204. Berlin: Springer.
21. Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. 1992. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge: Cambridge University Press.
22. Secchi, H., Carelli, R., and V. Mut. 2003. An experience on stable control of mobile robots. *Latin American Applied Research* 33(4):379–385 (10 2003). http://www.scielo.org.ar/scielo.php?script=sci_arttext&pid=S0327-07932003000400003&nrm=iso.
23. Smith, R. 2005. Open dynamics engine. http://www.ode.org.
24. Sucan, I.A., and S. Chitta. 2015. MoveIt! http://moveit.ros.org.

Author Biography

Walter Fetter Lages graduated in Electrical Engineering at Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) in 1989 and received the M.Sc. and D.Sc. degrees in Electronics and Computer Engineering from Instituto Tecnológico de Aeronáutica (ITA) in 1993 and 1998, respectively. From 1993 to 1997 he was an assistant professor at Universidade do Vale do Paraíba (UNIVAP), from 1997 to 1999 he was an adjoint professor at Fundação Universidade Federal do Rio Grande (FURG). In 2000 he moved to the Universidade Federal do Rio Grande do Sul (UFRGS) where he is currently a full professor. In 2012/2013 he held an PostDoc position at Universität Hamburg. Dr. Lages is a member of IEEE, ACM, the Brazilian Automation Society (SBA) and the Brazilian Computer Society (SBC).