We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

10

Generic Real-Time Motion Controller for Differential Mobile Robots

João Monteiro[1] and Rui Rocha[1]
[1]*ISR - Institute of Systems and Robotics*
Faculty of Science and Technology
University of Coimbra, Portugal

1. Introduction

Trajectory planning and execution are well known tasks to be performed by mobile robots. Its importance aroused in the past from the need to replace humans when performing either complex or dangerous tasks, mainly in industrial environments. Nowadays, the capabilities and applications of mobile robots are far more generic; Complex tasks pose more demanding control requirements within trajectory execution, including high responsiveness and predictability. Robot soccer might be used as a testbed to enact most of these requirements on an amusing environment and evaluate solutions to important robotic problems such as control, perception, intelligent systems, etc. Therefore, the RAC[1] robotic soccer team has been used to study and validate the motion controller described within this chapter. The control project consists of two main parts: 1) the digital controller, and 2) the real-time system. This latter provides high responsiveness to the previous, resulting in a high-performance implementation of the projected algorithm.

Extensive work has been made on controlling differential mobile robots. For instance, [1] presents a generic controller wherein pose estimation is extracted from robot's kinematics, and an adaptive control module is introduced to deal with modelling errors. In [9], a Lyapunov based nonlinear controller is presented, where the influence of the control parameters is studied, without giving emphasis to modelling errors. The approach described herein brings together the simplicity of the Lyapunov mathematical laws, the adaptive control concept to deal with modelling errors, and also proper fusion of two sensory data – vision and odometry –, for robust pose estimation.

Besides addressing the control issue, this chapter also describes a real-time system to complement the developed control module, so that it is able to fulfil pre-defined timing constraints. A considerable number of approaches such as [7] and [5] evidence the need of integrating a real-time system in robot control. Also, the sensory fusion technique using a Kalman filter will be addressed in detail due to its importance.

The modularity of the described system provides easy integration with higher level software modules for intelligent perception and robot actuation.

---

[1] Robótica Académica de Coimbra, http://rac-uc.pt.vu

1.1 Previous Note to the Reader

This chapter aims to clearly explain to the reader an implementation of a generic real-time motion controller for differential mobile robots. To give an insight on how to implement the approach in practice, at some points we refer to specific hardware used to validate the approach. However, this is made in a way that the reader can resemble the mentioned aspects to his own implementations with ease. Practical results and analysis will be present to validate the approach.

2. Control Requirements

In this section the requirements of the controller are defined, as well as the target robot kinematics and trajectory planning. This will serve as base knowledge to understand the subsequent sections.

2.1 Case Study: The Differential Mobile Robot

The developed digital controller performs dynamic control of a mobile robot with two parallel traction wheels which, therefore, possesses differential confguration. Its core goal is to allow the robot to follow a desired trajectory – defined by velocity vectors – to achieve a desired position with high accuracy. This is made by eliminating pose error in every discrete instant of time.

2.1.1 Trajectory Definition and Robot Kinematics

The studied 2D path planner defines a trajectory as a time variant pose vector represented in the world, which has its own global cartesian system defined. The robot's pose vector q possesses three degrees of freedom (DOF): the position (x,y) and its heading h. The latter is assumed to be positive in counter-clockwise direction, beginning at the positive xx axis. The state q_0 is denoted as the zero pose state (0,0,$2n\pi$), where n is an integer value. Since the robot is capable of moving in the world, the pose q is a function of time t. The integer representation of a set of points ($x(t)$,$y(t)$) is defined as trajectory, and if the derivates $\dot{x}$ and $\dot{y}$ exist, $h(t)$ is no longer an independent variable, since

$$h(t) = tan^{-1}\frac{\dot{y}(t)}{\dot{x}(t)} \tag{1}$$

from which one can see that the heading depends on the robot's velocity along each one of the two 2D axis. The robot's movement is controlled using its linear and angular velocities, v and w respectively, which are also functions of time t. The robot kinematics is defined by the Jacobian matrix

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{h}\end{bmatrix} = \dot{q} = Jp = \begin{bmatrix}\cos(h) & 0\\ \sin(h) & 0\\ 0 & 1\end{bmatrix} p, \tag{2}$$

where the velocity vector p is defined by the linear and angular velocities v and w,

$$p = \begin{bmatrix}v\\ w\end{bmatrix}. \tag{3}$$

This kinematics is common for all non-holonomic robots in which the number of controllable DOF's is less than the number of DOF's the robot possesses.

2.1.2 Pose Error

The core goal of any trajectory controller is to reduce the robot's pose error as much as possible, relative to a desired target point. Pose error q_e is defined as the transformation of the reference pose q_d to the local coordinate system of the robot with origin (x, y), wherein the current robot's heading is given by h's amplitude (Fig. 1).

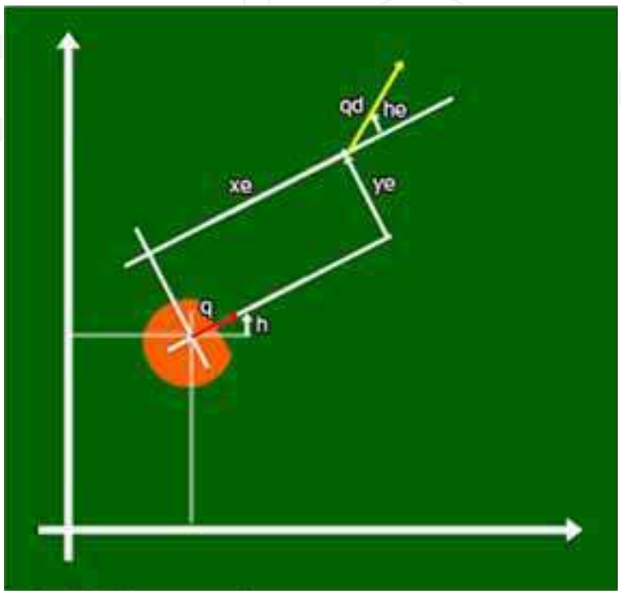

Fig. 1. Representation of robot's pose and pose error vectors.

Such transformation is the difference between q_d and q

$$q_e = \begin{bmatrix} x_e \\ y_e \\ h_e \end{bmatrix} = \begin{bmatrix} \cos(h) & \sin(h) & 0 \\ -\sin(h) & \cos(h) & 0 \\ 0 & 0 & 1 \end{bmatrix} . (q_d - q) \quad . \tag{4}$$

One can easily see that if $q_d = q$, the pose error is null, being this the ideal final state.

2.1.3 Robot Dynamic Model

Based on the Lagrange mathematical modelation of mechanical systems [1], and considering $G(q) = C(q, \ddot{q}) = 0$, the generic dynamic equations of the mobile robot can be written as

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} . \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{h} \end{bmatrix} = \frac{1}{R} . \begin{bmatrix} \cos(h) & \sin(h) \\ \sin(h) & \sin(h) \\ L & -L \end{bmatrix} . \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} + \begin{bmatrix} \sin(h) \\ -\cos(h) \\ 0 \end{bmatrix} \lambda, \tag{5}$$

with τ_1 and τ_2 being the torques of left and right motors respectively, m and I the robot's mass and inertia, R the wheel radius, L the linear distance between the two wheels, and λ the Lagrange multipliers of constrained forces. The non-holonomic restriction is deduced from the above equation, and is given by

$$\dot{x}\sin(h) - \dot{y}\cos(h) = 0, \tag{6}$$

from where it is imposed that a non-holonomic mobile robot can only move in the direction normal to the axis of the driving wheels.

2.1.4 Trajectory Planning

The trajectory planning algorithm, defines intermediate target points, hereafter denoted as virtual points, which are spaced about a configurable distance, representing the intermediate desired poses between the initial instant and the target pose q_d. These poses will be used by the controller to perform pose error elimination during trajectory execution. For the iteratively placed virtual points, the desired pose is defined relative to the robot's local coordinate frame by

$$\begin{aligned} x_{d,p} &= X_{inc}.\cos(\theta_v) - Y_{inc}.\sin(\theta_v) \\ y_{d,p} &= X_{inc}.\sin(\theta_v) + Y_{inc}.\cos(\theta_v) \\ h_{d,p} &= h + \theta_v \end{aligned} \tag{7}$$

To make $x_{d,p}$ and $y_{d,p}$ consistent with a fixed coordinate frame, they need to be referenced to the world's coordinate frame. This is done using the following transformation:

$$\begin{aligned} x_{d,p,w} &= x_{d,p}.\cos(\theta_v) - y_{d,p}.\sin(\theta_v) \\ y_{d,p,w} &= x_{d,p}.\sin(\theta_v) + y_{d,p}.\cos(\theta_v) \\ h_{d,p,w} &= h_{d,p} \end{aligned} \tag{8}$$

2.2 Control Requirements Analysis and Specification

From the above sections, one can easily understand the objective of the controller, pose correction, from where a convergence of the pose error state variables to a null state is desired. To achieve this, the controller needs to deal with inevitable modeling errors which, once implemented, can impose stability problems. Therefore, in the design of the presented controller, the impact of modeling errors was carefully considered so that they can be efficiently minimized. Such errors are cumulative with $t \to \infty$, resulting in evident pose mismatches.

The developed controller is focused on the kinematic stabilization of generic mobile robots possessing differential configuration, which imposes a great challenge since no specific dynamic model is considered. Few parameters representative of the physical structure need to extracted; among them we have mass, inertia, wheel radius and distance between wheels. Despite the mentioned effort to avoid modeling errors, measurement errors can be present in the extracted parameters. Having this in mind, the concept of adaptive control is introduced, enhancing the system robustness. The controller was firstly introduced in [13], but it's further refined in this article through a real-time implementation and a more thorough experimental validation.

3. Digital Controller Design

The controller design is comprised of three parts. In the first, kinematic stabilisation is achieved using nonlinear control laws. For the second, the acceleration is used for exponential stabilization of linear and angular velocities. The uncertainty on the robot's structure parameters is compensated using an adaptive control block. Introducing suitable Lyapunov functions, stability of the system state variables is achieved. For the final part, pose estimation is made by fusing odometry and vision for robust pose feedback information by means of a Kalman filter. The latter is designed so that it is independent of the mathematical model of the robot, boosting overall performance and applicability.

3.1 Control Scheme

The developed approach is depicted in Fig. 2.

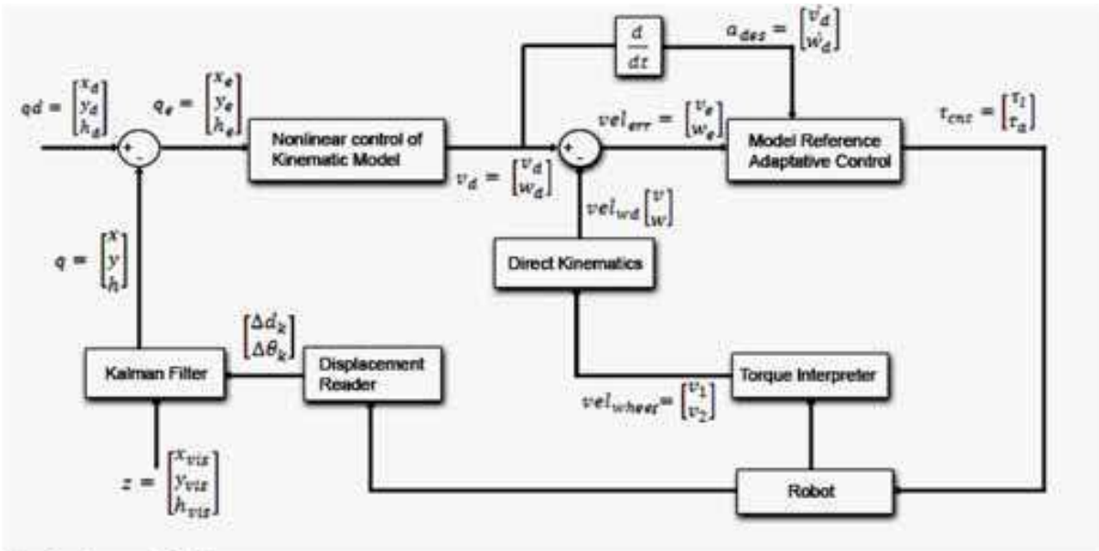

Fig. 2. Control Scheme

It is a feedback controller, in which the input state is the desired robot pose $[x_d\ y_d\ h_d]'$. At its output, proper update of the motor torques for each wheel is done to fulfill the controller's pose error elimination goal. The adaptive control block ensures that a stable condition is achieved independently of the presence of modeling errors in the robotic platform parameters. The estimation error is brought to zero in finite time. The following subsections describe the different control modules in more detail.

3.1.1 Pose Error Generator

The error dynamics is written independently of the inertial (fixed) coordinates frame by Kanayama transformation. Expanding equation 4, the following is given

$$q_e = \begin{bmatrix} x_e \\ y_e \\ h_e \end{bmatrix} = \begin{bmatrix} \cos(h) & \sin(h) & 0 \\ -\sin(h) & \cos(h) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_d - x \\ y_d - y \\ h_d - h \end{bmatrix}, \tag{9}$$

which composes the pose error vector.

3.1.2 Nonlinear Kinematic Controller

Lyapunov based nonlinear controllers are very simple and yet very successful in kinematic stabilization. So, bringing together simplicity and functionality, the Lyapunov stability theorem proved to be of great utility for the controller presented herein. The following Lyapunov candidate function which represents, as stated in [6], the total energy[2] of the robot is considered

[2] The total sum of kinetic and potential energies.

$$V = \frac{1}{2}({x_e}^2 + {y_e}^2) + [1 - \cos(h_e)]. \tag{10}$$

To prove the stability condition, $\dot{V}$ needs to be obtained, and criteria must be applied. Based on [1] and [6], we have

$$\begin{aligned} \dot{V} &= v_r x_e \cos(h_e) - v_d x_e + v_r \sin(h_e)\, y_e + w_r \sin(h_e) - w_d \sin(h_e) \\ \Leftrightarrow \dot{V} &= v_r \cos(h_e - v_d)\, x_e + \sin(h_e)\,(v_r y_e + w_r - w_d), \end{aligned} \tag{11}$$

wherein v_r and w_r are the reference velocities. v_d and w_d are chosen to make $\dot{V}$ become negative semi-definite:

$$\begin{aligned} v_d &= v_r \cos(h_e) - x_e \\ w_d &= w_r + v_r y_e + \sin(h_e). \end{aligned} \tag{12}$$

Replacing the above expressions in (11), we obtain

$$\dot{V} = -{x_e}^2 - v_r sin^2(h_e), \tag{13}$$

from where one can easily see that $\dot{V}$ is always negative semi-definite. Considering a specific instance of the control rule at this block output, the following is given

$$\begin{aligned} v_d &= v_r \cos(h_e) - K_x x_e \\ w_d &= w_r + K_y v_r y_e + K_h \sin(h_e), \end{aligned} \tag{14}$$

where K_x, K_h and $K_h$are positive constants. By La Salle's principle of convergence, and proposition 1 of [6], the null pose state q_0 is always an equilibrium state if the reference velocity is higher than zero ($v_r > 0$). As one can see, there are three weighting constants for the pose error, without interfering in the overall pose stability of the robot. These parameters are responsible for the influence of each component of the pose error vector on the stability process, and need to be carefully tuned.

3.1.3 Model Reference Adaptive Control

The motivation to include this module in the control mesh comes from the need of the controller to cooperate with parameter uncertainties. Based on [1], one can extract the adaptation rules for the linear velocity

$$\begin{aligned} \frac{d\theta_1}{dt} = -\epsilon_1 e \dot{v}_d \Leftrightarrow \theta_1 = \int -\epsilon_1 e \dot{v}_d \, dt \\ \frac{d\theta_2}{dt} = -\epsilon_2 e \dot{v}_d \Leftrightarrow \theta_2 = \int -\epsilon_2 e \dot{v}_d \, dt. \end{aligned} \tag{15}$$

Identically, one can find similar rules for the angular velocity

$$\begin{aligned} \frac{d\theta_3}{dt} = -\epsilon_3 e' \dot{w}_d \Leftrightarrow \theta_3 = \int -\epsilon_3 e' \dot{w}_d \, dt \\ \frac{d\theta_4}{dt} = -\epsilon_4 e' \dot{w}_d \Leftrightarrow \theta_4 = \int -\epsilon_4 e' w_d \, dt \end{aligned} \tag{16}$$

where v_d and w_d are the desired linear and angular velocities that constitute the output of the previous module and e and e' the respective velocity errors. The parameters $\epsilon_i, i = \{1..4\}$ are manually tuned for best performance achievement.

Practical implementation of this block is not simple, since it is necessary to determine the virtual velocity[3] before proceeding with the actual real velocities calculation, which in turn is needed to evaluate the error of the modelled parameters m and I and provide proper

[3] Velocity of the reference model.

acceleration compensation. So, for better organization and comprehension, it was important to create a schematic for the present module (Fig. 3).

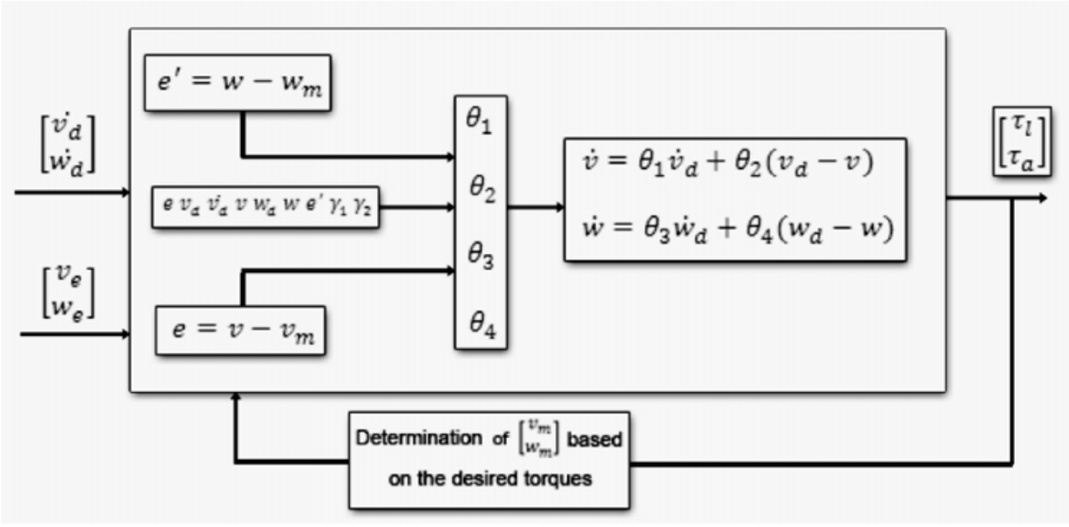

Fig. 3. Adaptive control block scheme.

From this figure, one can easily see the virtual velocity determination before evaluating the velocity error due to uncertainty on modelling parameters. The output are the linear and angular torques, which are used to easily determine the left and right wheel -- τ_1 and τ_2 -- torques,

$$\tau_l = \frac{1}{R}(\tau_1 + \tau_2) \qquad (17)$$
$$\tau_a = \frac{L}{R}(\tau_1 - \tau_2)$$

In practice, this module provides acceleration compensation to the robot. Imagine, for instance, that theoretically the robot should be navigating with a velocity of 1mps, instructed by the nonlinear control module. By theoretically we mean the robot modelled when planning the controller. This 1mps will be the velocity of the reference model, however the platform may not be navigating at that velocity due to badly modeled parameters. The adaptive control block will take notice of this difference and accelerate/decelerate the robot so that its velocity follows the reference model. Referring to Fig. 3, the reader can easily see the linear and angular acceleration equations that use the adaptation rules, which compensate the difference between the actual robot velocity relative to the reference model. The reader can think of this as a virtual (ghost) robot walking with the real platform that must be followed.

3.1.4 DC Motor Actuation Based on the Desired Torques

As we have seen from the controller scheme in Fig. 2, the output of the control mesh consists of the torque vector, but it is not explicit on how the motors are actuated in response to this torque command. The amplitude of the electric current that controls the motor power is regulated by a drive board. Such board implements a H bridge to drive each motor individually by a PWM signal. Note that this is a hardware specific section and has been put

here to give to the reader a better comprehension on how the algorithm can be implemented.
Following the testbed team's specific case, and according to the datasheet of the used DC motors, the torque-current relation is given by

$$I = k_i \tau, \tag{18}$$

where k_i is the electrical current constant, which equals 0.487. For a motor (left or right) to which a rotor torque of τ_d is desired, the necessary current that the motor must consume can be extracted as follows

$$I_d = k_i \tau_d. \tag{19}$$

Referring to the datasheet, the maximum current value that the motor should consume is 0.57A, being the ideal value 0.35A. Therefore, the I_{MAX} reference value should be between 0.35 and 0.57, letting the hardware capabilities be fully used, but avoiding persistent overload. So, a value of 0.4A is admitted to be suitable and the maximum power consumption for the robot's 12VDC motors is calculated by

$$P_{MAX} = VI_{MAX} = 12 * 0.4 = 4.8W \tag{20}$$

By assuming that the motor will not work above a maximum value of its active power of $P_{MAX} = 4.8W$, its safety is ensured and, at the same time, the robot takes advantage of its performance. It is also guaranteed that it will not overheat. Based on such value, the relation *dutyCycle-current* can be determined.

Let V_{ap} be the average voltage value applied to the motor as a consequence of a command dutyCycle of *DtC*, the following is given

$$V_{ap} = \frac{DtC}{100} * 12. \tag{21}$$

Therefore, if the motor consumes a maximum current of 0.4A at 12VDC, we have

$$I_{ap} = \frac{V_{ap}}{12} * 0.4. \tag{22}$$

Bringing together the two above equations, the following is given,

$$DtC = k_{ap} I_{ap}, \tag{23}$$

where $k_{ap} = \frac{1}{0.4}.100$. If the *torque-current* relation is applied, the useful *dutyCycle-torque* relation can be extracted,

$$DtC = k_{ap} k_i \tau_{ap}. \tag{24}$$

Based on this latter equation, it is possible to know by a simple calculation what value should be attributed to the duty cycle for a motor to generate a τ_{ap} torque.

3.1.5 Torque Interpreter

This module makes proper measurement of the actual velocity based on the encoder displacement during a sample time. Its output data will be used to determine the velocity error of the platform at a given time instant to feed the adaptive control module.
This module works as follows. First, a measurement of the encoder count is made and t_{samp} is waited. After this short time (the shortest possible), a new measurement is sampled and the counting is made by finding the difference of the last and actual pulse count value.
Given that the used motors possess a gear box with 1:3.71 ratio, and that the encoder is able to produce 512 pulses in a row, there are 4*412 transactions when the encoder reading is in quadrature mode (defined in the drive board). The angular increment of the encoder is

$\frac{2\pi}{2048} rad$ referred to the motor. Referring to the wheel, this results in $\frac{2\pi}{2048\times3.71} rad$. By this, the robot wheel velocity vector in rpm (rotations per minute) is

$$v_{wheel} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \tag{25}$$

where $v_1 = v_2 = \frac{pulse\_count}{t_{samp}.1000} . \frac{2\pi}{2048\times3.71} . 60.$

3.1.6 Displacement Reader

Due to the fact that the implemented Kalman filter needs a specific input vector of Δd and Δh, linear and angular displacements respectively, it is necessary to determine such values for every control loop as

$$\Delta d = \frac{d_1 + d_2}{2} \; ; \; \Delta h = \frac{d_1 - d_2}{b}, \tag{26}$$

wherein b is the distance between wheels, and d_1, d_2 the distance walked by the left and right wheels respectively. This is done simply using odometry data and counting pulses between sample instants to determine d_1 and d_2. Then, knowing how many pulses it takes for a complete revolution of the wheel, i.e., $2\pi r$, we can easily determine the linear displacement of each wheel.

3.1.7 Kalman Filter for Sensory Fusion

A differential robot possessing an odometer system is equipped with an encoder in each motor. An angular displacement of α radians on the rotor corresponds to a moved distance d on the periphery of the wheel and, subsequently, to an encoder count. The distance is given by $d = k\alpha$, with $k = \frac{1}{r}$, being r the radius of the robot. If the robot's movement is assumed to be linear, the distances d_1 and d_2 walked by the left and right wheels respectively, can be transformed into linear and angular displacements through (26). For a particular sample instant k, the following is given

$$\Delta d_k = \frac{d_{1,k} + d_{2,k}}{2} \; ; \; \Delta h_k = \frac{d_{1,k} - d_{2,k}}{b}. \tag{27}$$

The robot's coordinates referenced on the world's coordinate frame can be determined by these simple relations

$$\begin{aligned} X_{k+1} &= X_k + \Delta d_k \cos\left(h_k + \frac{\Delta h}{2}\right) \\ Y_{k+1} &= Y_k + \Delta d_k \sin\left(h_k + \frac{\Delta h}{2}\right) \\ h_{k+1} &= h_k + \Delta h \end{aligned} \tag{28}$$

These coordinates constitute the state vector, and are observed by the vision's coordinate state vector z.

Note to the reader: It is important at this time to refer that we use a global vision system capable of detecting the position of the robot in the world referred to its global axis. This is composed by two non-stereo cameras placed at the top, parallel to the world, where a color code capable of identifying the robot is used so that the vision system can determine its pose vector. Therefore, we use the Kalman filter to fuse vision and odometry data to retrieve a highly accurate pose estimation for the robot in each loop. The reader should not be concerned, since Kalman filter can be used to fuse any kind of sensory data, as long as they

provide the same state variables. For instance, instead of vision, the reader can fuse inertial sensory data and odometry with this technique. We will keep on mentioning vision as the sensory system used as observation in the Kalman filter. However, the reader can imagine this as a black box that retrieves an estimate of the robot's pose, i.e., a pose vector composed of $[x'\ y'\ h']$.

Such measurements can be described as a nonlinear function c of the robot coordinates, which possesses an independent noise vector v. Denoting the nonlinear equations (28) as the estimated pose vector α, and placing Δd_k and Δh_k as an input vector u_k, with associated process noise vector w_k, the robot can be modelled by the following nonlinear equations

$$x_{k+1} = \alpha(x_k, u_k, w_k, k) \qquad (29)$$
$$z_k = c(x_k, v_k, k)$$

which represent the state transition model and the observation model of the robot. The reader should note the importance of these equations, since they reveal the basis of sensor fusion with a Kalman filter, in this case, the Extended Kalman Filter (EKF), since we are dealing with a nonlinear system.

Note to the reader: Real systems are often nonlinear. This means that its state variables have a nonlinear behaviour in time. In our case, the position of the robot cannot be modelled by a linear equation, since its pose can possess unpredicted values.

The first, on equation (29) is the state transition model. The prediction stage of the Kalman filter relies only on odometry data to estimate the actual state, based on the walked amount between the actual and last iteration (determined by odometry). u_k is often denoted as the control input, which in our case, is simply the module of the distance the robot walked between two sample instants. It is important at this point to take in account the encoder resolution: it should be higher than the controller's sampling ratio so that between different control instants we can have a measurement of the distance walked by the robot as explained in section 3.1.5. So basically what the state transition model does, is to perform a naive prediction of the actual pose of the robot based only on odometry data. This prediction will next be refined by the correction phase of the filter, where the observation data is used: the vision system in our case. w_k is denoted as the state transition model associated error. Here we define the variance (or associated uncertainty) to the state variables determined by the state transition model. This vector is set up when projecting the stochastic model of the Kalman filter, and the variance may be adjusted using system simulations. If no correlation between the state variables exists, the covariance matrix of the state transition model noise Q_k will be diagonal, having w_k in its diagonal. The same goes for the noise vector associated with the observation model v_k. This vector represents the uncertainty associated to the observation model, (in our case to the vision system). Its covariance matrix is R_k and again, it will be a diagonal matrix with v_k as its diagonal values if the state variables are uncorrelated, which is the case for the mobile robot's pose estimation.

Note to the reader: The reader can easily imagine the noise by visualizing a Gaussian probabilistic density function for each state variable. If more uncertainty is given to a certain state variable, we are disbelieving that the process can provide a good estimation, hence raising the bell shape size of the Gaussian distribution in turn of the mean value, which is normally zero for white noise. We are then allowing values farther than zero for the noise, which will affect our estimated state variables.

With this being said, it is recommended to simulate the system using MATLAB for instance, so that proper tuning of the state variables can be achieved.

So, in our case $w_k \sim N(0, Q_k)$ and $v_k \sim N(0, R_k)$, being both non correlated, i.e., $E[w_l {v_l}^T] = 0$, where l is the number of state variables. The Extended Kalman Filter can then be constructed, using the odometry-based system model [8]

$$\begin{aligned} \hat{x}_{k+1} &= \alpha(x_k, u_k, w_k, k) \\ P_{k+1} &= A_k P_k A_k^T + Q_k, \\ K_k &= P_k C_k^T [C_k P_k C_k^T + R_k]^{-1} \\ \hat{x}_{k+1} &= \hat{x}_k + K_k [z_k - C_k \hat{x}_k] \\ P_{k+1} &= [I - K_k C_k] P_k. \end{aligned} \tag{30}$$

The filter was simulated using different noise vectors for both state transition and observation model. A DC motor transfer function was used to simulate both models (Fig. 4). This naïve simulation allowed to make concrete conclusions.

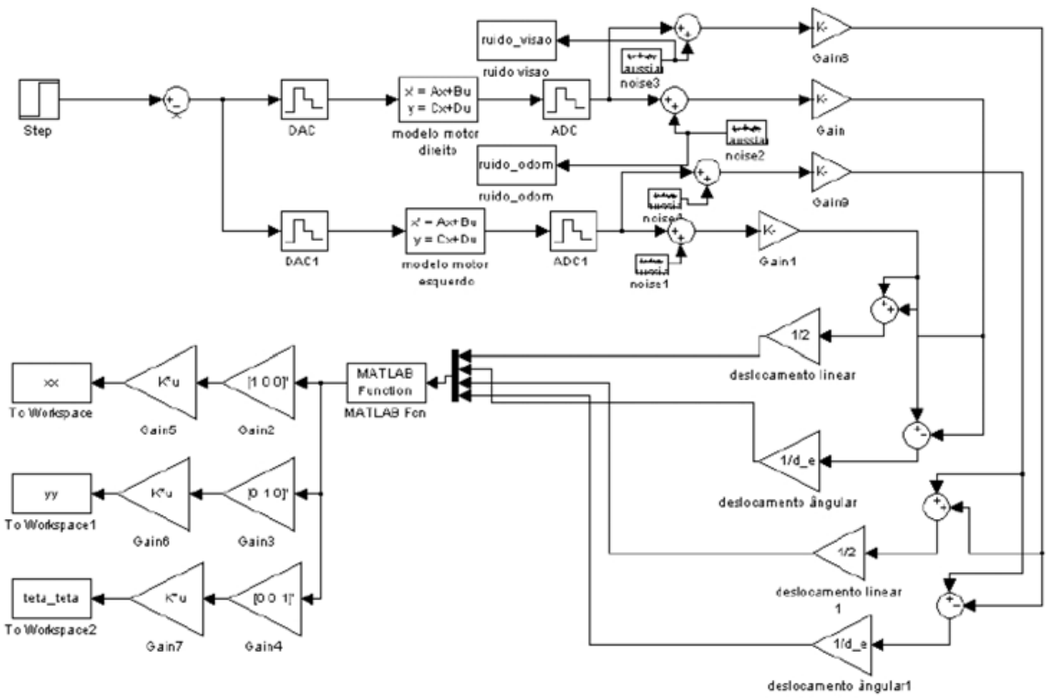

Fig. 4. Kalman filter simulation using same source for both models.

When a voltage is present at the input of each motor transfer function, a displacement is produced at its output, allowing us to simulate the robot movement within our system. In this case, and because there is intentionally no feedback loop (evident by the disconnected entrance placed before the DAC's) as the objective is to simply evaluate the filter's performance under different noise conditions for both the state transition model and for the observation model, the output will rise indefinitely in a linear form, except at the start phase of the motor's rotation movement. Observations of the filter's behavior in the present of both vision and odometry noise, and subsequent analysis of the estimated state variables were made. for this, specific situations were imposed.

3.1.7.1 Simulation 1 – robot in x=0, y=0, h=0, $\sigma_{vis}^2 = 1, \sigma_{odo}^2 = 1$

For this case, the displacement made by the robot is indefinitely linear along the xx axis, being the displacement over yy and the robot's heading equal to zero. Fig. 5 (left) shows this situation, being the blue slope the displacement over xx, the red slope the displacement over yy and the green slope, the robot's heading.

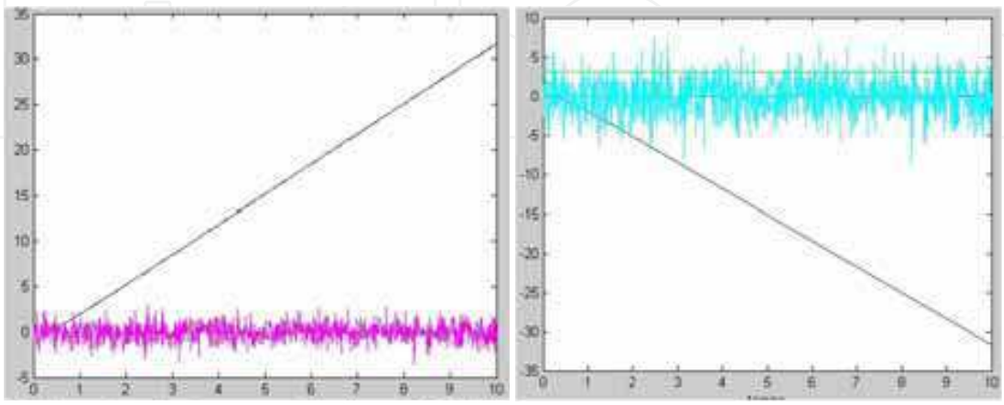

Fig. 5. Filter simulation case 1 (left) and 2 (right).

The magenta slope represents the vision noise. As one can see, the filter possesses sensitivity to the vision noise contrary to what happens with odometry noise as we will further see.

3.1.7.2 Simulation 2 – robot in x=0, y=0, h=3.14, $\sigma_{vis}^2 = 0, \sigma_{odo}^2 = 6$

In this case, the filter's robustness to odometry errors is evidenced. Note that, because the robot is oriented by π radians, letting xx axis be the reference of the angle, the robot will move along the negative side of this axis. Fig. 5 (right) shows this situation, having a cyan slope that represents the odometry noise and its exaggerated variance. High belief is put to the observation model, therefore, s it can be seen from the figure, in every sample instant the noise does not affect the output.

3.1.7.3 Simulation 2 – robot in x=0, y=0, h=3.14/2, $\sigma_{vis}^2 = 10, \sigma_{odo}^2 = 6$

In this case, the vision noise was drastically augmented, revealing some fragility of the filter (Fig. 6). However, this situation will not happen in practice since calibration makes the vision noise drop to very low values. One must however be aware of this fact when choosing the sensory system for observation.

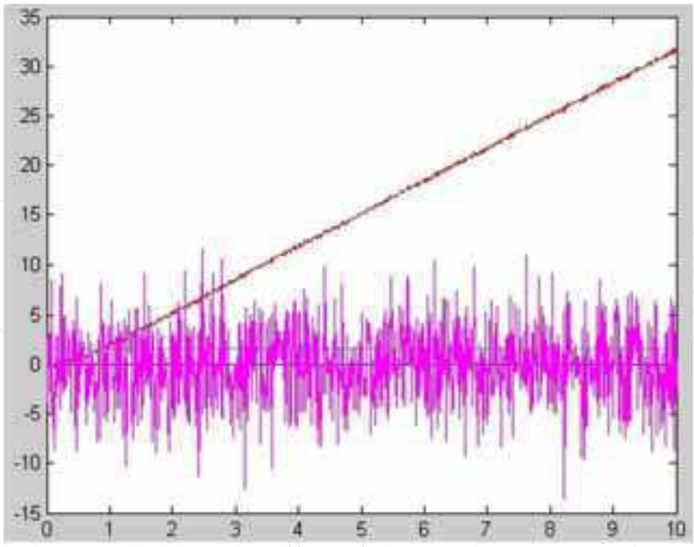

Fig. 6. Filter simulation case 3. Note the effect of observation error on the pose estimation.

3.2 Preliminary Results

The previously described control algorithm was implemented in its preliminary version 1 (without the real-time system for now), and promising results were achieved. It accepts linear velocity vectors as input commands (`setVelocities`) composed of a module and angle$[v_f \ \theta_f]$ in [m/s rad], as well as angular velocity commands in rad/s. A simple test application that sends these commands was used to test the controller, and a visualizer application was made for one to see and record what is really happening in the world. It is basically a virtual representation of the reality happening in real-time. This latter registers all the robot movements as white dots, represents the virtual point trajectory (calculated as shown in section 2.1.4), and also the initial state of the robot. The goal is that the robot autonomously follows the predefined trajectory of virtual points. All the data is sent from the robot to the visualizer using a TCP/IP wireless network.

The test tool is also able to send target desired poses, which we will call from now on as `setPoints`. This command is interpreted by the control module in its input stage, and a velocity vector is composed that allows the robot reach the desired point.

Note to the reader: Recall that we are focused on the control module, so no obstacle avoidance is taken in account here. Our experiments occurred in a world free of objects. This task should be performed by higher level software modules capable of perceiving the environment and sending velocity commands to the robot so it can then be instructed with movements that will result in obstacle avoidance. Our work is focused on the motion control only.

3.2.1 `setPoint` command to (0,0), the center of the world

For this test, we send a `setPosition` command to (0, 0). In figure 7 (left), a screenshot of the previously referred visualizer tool developed for the controller module is shown for this particular case. The robot accurately goes to the defined `setPoint`, possessing a yy axis precision error of 1 to 2 centimeters maximum. This precision error exists because of two

main causes: the first is due to the backward force exerted by the energy cable that feeds the robot in test environment; the second – and main – reason is the defined tuning parameters for the influence of the robot's error over the yy coordinateK_y. Tuning for near-zero error is possible but leads to a very hard control scheme in the presence of physical disturbances, making the controller produce high overshoots for compensation. Since our robot is destined to walk on a world where collisions with other robots may be present, the revealed accuracy perfectly suits for the team's needs. For collision-free applications, where minimum physical errors exist, and depending on the world's physical available space, the controller can be made harder by rising not only K_y but also K_x and K_h (please recall section 3.1.2).

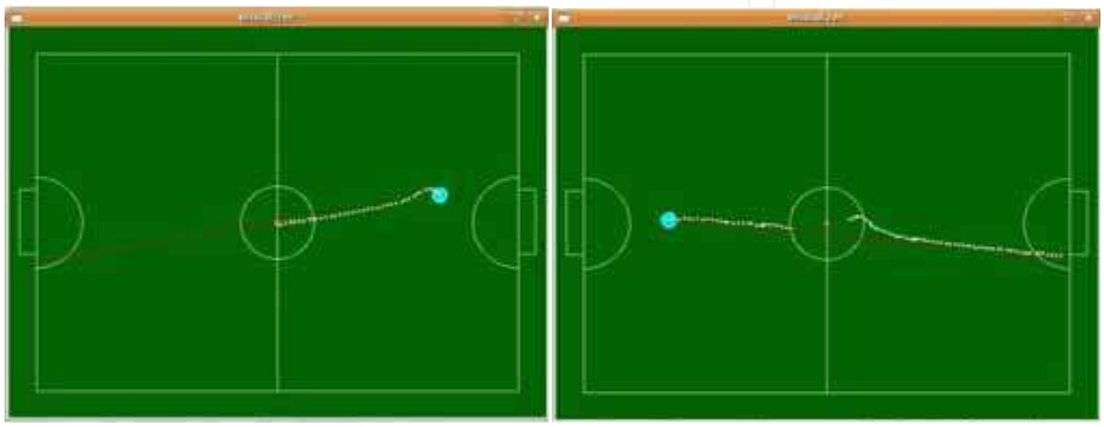

Fig. 7. `setpoint` to (0,0) (left) and `setVelocity` with v=0.3mps and desired angle = 0 (right).

3.2.2 `setVelocity` command with v = 0.3mps and desired velocity angle = 0 rad

For this test, the robot was subjected to extreme noise conditions. Referring to fig. 7 (right), it can be seen that the robot is subjected to two disturbances. One was accomplished by blocking its left wheel, evident by the multiple white dots in the same place. Another was made by blocking the color code of the robot used by the vision module to identify it, so that no vision data was being received by the controller for it to estimate the actual pose during this period (odometry-only based pose estimation: Kalman bypass). Controller's robustness is then proved, since the robot follows the desired trajectory as it starts receiving vision feedback, despite its erroneous position at that time because of the -- widely already known -- odometry alone based pose estimation cumulative error for the feedback loop of the controller. Note that a bypass to the Kalman is made when no vision data is available.

3.2.3 Sequence of `setVelocity` commands with v=0.3mps and velocity angle = 1.3 rad

In this final test (Fig. 8), a sequence of `setVelocity` instructions were sent to evaluate the control module's response in the presence of new velocity instructions. This test approaches the real application target environment, the soccer game, where high dynamic handling is required for the robot. Referring to Fig. 8, a velocity vector with zero desired angle is first sent, followed by two `setVelocity`'s with the same module and 1.3rad for the desired

velocity angle. Finally, the robot is halted with a halt instruction. As it can be seen, the robot accurately executes the performed commands, evidencing the software module's robustness. More tests will be shown in further sections regarding the final version of the software implementation which is capable, for instance, of raising the acceleration so that the robot can overcome obstacles. It will also possess the real-time system for greater responsiveness achievement and more predictability.

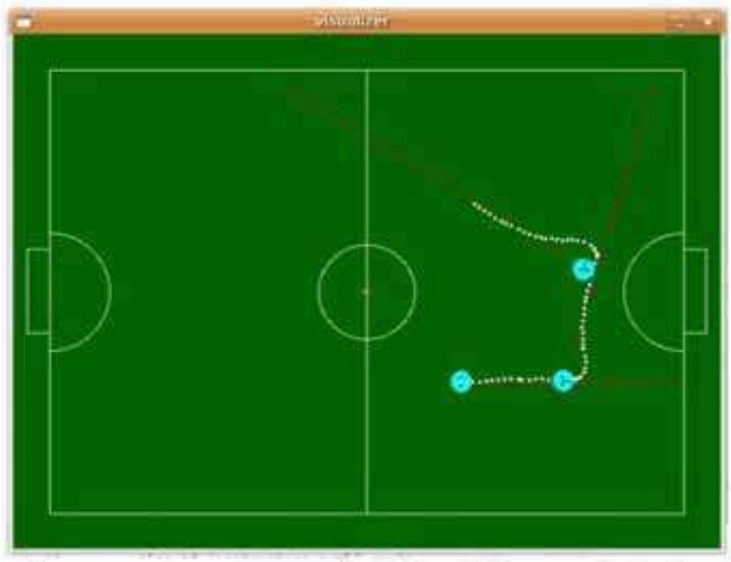

Fig. 8. Sequence of `setVelocity` commands with v = 0.3mps and velocity angle = 1.3rad.

4. Real-Time System

The modularity of the implemented controller described in the previous sections made possible the project of a suitable real-time RTAI (Real Time Application Interface) [12] based system.

Note to the reader: During the course of designing and implementing the real-time system, two articles were written [10] and [11] that may help the reader to implement an RTAI based real-time system for his own applications.

During all the control module software run-time, there are concurrent threads scheduled with the fair non-preemptive Linux scheduler which in turn is implemented in its kernel space. This imposes critical problems, since it is desired to have tasks scheduled in a premeditated fashion so that the control system can be predictable, and Linux is not able to guarantee response times for its processes. For instance, a critical situation comes when the `setTrajectory`[4] instruction is executing: the thread that calculates the distance to the `setPoint` is superimposed by the `setVelocity` thread, making the robot not realize when it is near the target. So, it is desired to have a better solution than the one implemented, which was presented above, so that full robustness can be achieved. Therefore, the desired system needs to have the following requisites,

[4] Instruction that defines various `setPoints`, i.e., target positions that the robot must achieve.

• Predictability;
• Quick response to instructions;
• Ability to choose the highest priority task within a concurrent set of tasks;
• Minimize the error of the current robot pose received from the vision system and/or odometry;
• Ensure fixed periodicity for the controller;
• Preemptivity;
• Ensure precise cycle times for periodic tasks.

Every system that deals with a high number of tasks at the same time must be predictable, i.e. such systems must always exhibit a specified behavior (e.g. a task executed with a precise periodicity and no interruptions). Or if a critical instruction is sent to be executed on the fly, the system responds to the desired order as expected, when expected. This type of behavior is completely desired for the system.

As for the pose data received from odometry and/or from the vision module, it is important to have an intelligent planned scheduling for both tasks, in order to receive the precise actual pose of the robot at any time without information delays caused by thread overlapping.

The feedback cycle needs to be periodic so that the controller executes properly. This can only be ensured by scheduling the controller task with a pre-defined period and setting a high priority to it in the preemptive real-time system. This task is classified as a hard real-time task, since its activity must be started at every period with no delays. The integration of a real-time system in the control module aims at fulfilling these requirements.

4.1 Designing the system: time line example

A real-time system must be carefully planned. We used UML (Unified Modeling Language) to aid on this complex task. This is, for sure, the most important and hard task on the process of developing a real-time system for an application. All the evolved tasks must be pointed out, and their dataflow must be clearly exposed. The importance of each task must also to be taken in account when assigning priorities to the tasks. Then, the system architect must also to evaluate if the tasks are *soft* or *hard* real-time, allowing (or not) the task to possess a delay for its activation.

We projected a system for our case, which will not be referred here due to loss of generality. However, an example of a time-line case of our system is depicted to serve as example to the reader on what is pretended. Fig. 9 shows various tasks of the controller software module and its states. The Two final are instructions which handler is implemented in the control software module. The `Controller`, `Command Interpreter` (parsing) and `System Status` are main tasks of the control software module. The sequence of instructions executed upon receiving a `setVelocity` instruction can be seen in the figure. For the tasks to run as expected their priorities, activity mode (one shot or periodic) and activation type (hard or soft) were carefully planned. For instance, the `System Status` task possesses high periodicity over all other tasks to listen for new instructions. This is visible upon receiving a HALT instruction, which makes the controller stop while running.

Note to the reader: Recall that projecting a real-time system is all about knowing the application to implement. Modularity helps a lot on this, since then we can clearly define the real-time requirements for the system. Implementing the system is in fact really easy when compared to the real-time project phase.

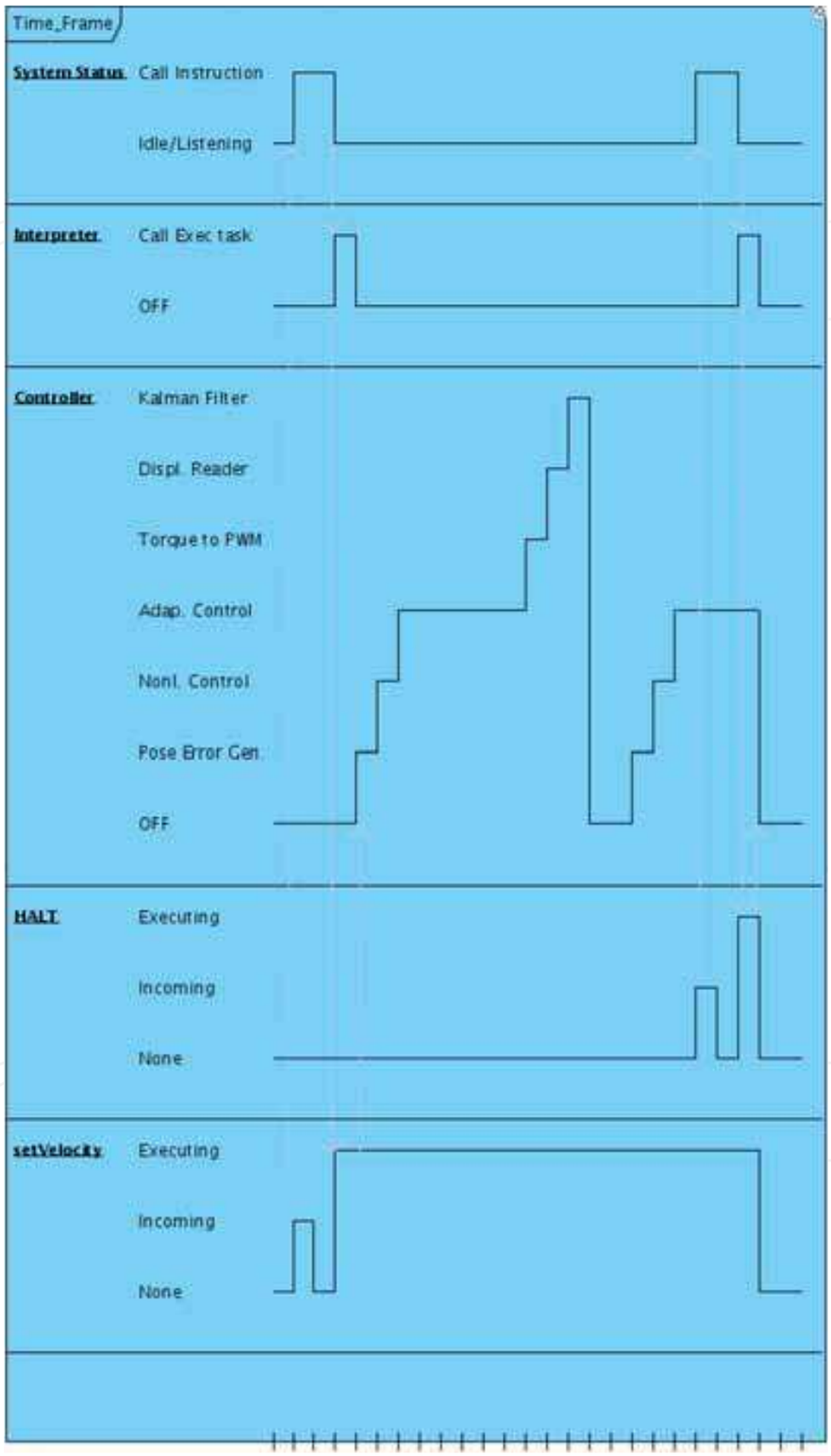

Fig. 9. setVelocity instruction receive case timeline.

4.2 Timing Performance Analysis

Various tests were made in order to validate the implemented real-time system. These tests allowed to investigate whether the tasks were meeting their timing requirements.
One test was made by extracting the cycle time of the controller task, which was set as a hard real-time task. This period was set to 52ms.

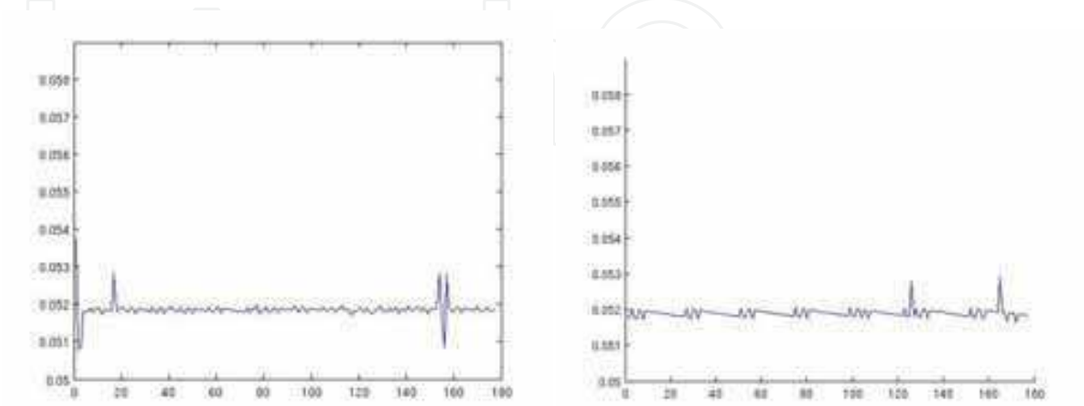

Fig. 10. Measured period of a hard real-time period task of the controller (left) and Soft real-time controller task times between control loops (right)

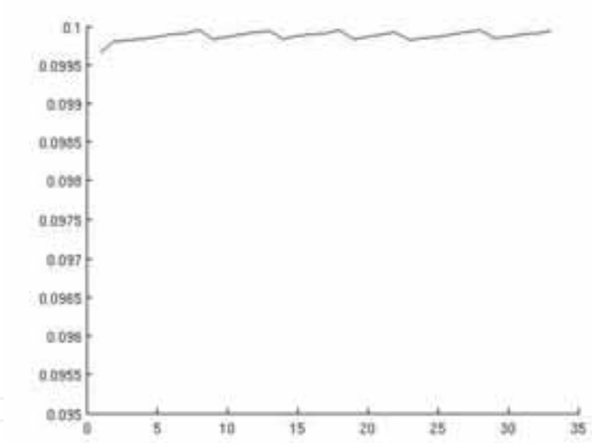

Fig. 11. Cycle time of `setTrajectory` handler instruction, implemented in the control module.

As it can be seen from Fig. 10 (left), very accurate and consistent periodic times are achieved being, in most of the cases, virtually equal to 52ms possessing very little variation. It was observed that in some few iterations, the time between consecutive cycles shifts 52ms, which was caused by delays of the vision data gather task having higher priority, which needed to call a recv() instruction which, in turn, is scheduled by the Linux kernel layer. Nevertheless, such delays introduced only a negligible impact on the implemented real-time controller. If the hard real-time requirement is not imposed for this task, few differences were surprisingly observed in run-time (Fig. 10 (left)). This leads to the conclusion that the

schedulability of the system provided by RTAI is accurate and also that the system was robustly projected since there aren't observable scheduling issues during the controller execution.
Another test was made to the *setTrajectory* handler task cycle times, which in turn periodically evaluates the distance left for the robot to reach a desired target point allowing the robot to know when to stop the controller since it reached the desired target point. A period time of 100ms was set and no hard real-time imposition was made. As it can be seen from Fig. 11, the periodicity varies within less than 0.1s which confirms the robustness of the implemented real-time controller. These remarkable results allow the robot to precisely reach a desired target point and proceed to the next one with high precision. As for task activation times, the observed values possessed an average of 10ms. These times are perfectly acceptable for the application. These practical results demonstrate that the system's timing requirements are successfully met.

5. Experimental Evaluation

This section is dedicated to the analysis of the experimental results obtained by the implemented
control system with real time system totally incorporated. These tests were aimed at demonstrating about the system's integration on dynamic environments and assess its robustness. Instructions such as *setVelocity* and *setTrajectory* are sent to the robot and the pose error results obtained are analyzed.
This section also intends to analyze the influence of the controller variables K_x, K_y and K_h which need to be tuned so that a good performance is achieved. During the following set of tests, we consider that the robot is walking on a obstacle-free world, where the (x, y) pose measurement units over each axis is mm.

5.1 Analysis of the controller parameters influence: Tuning the controller

As it was introduced in section 3.1.2, there are three important weighting variables which define the controller's correction strength over each DOF of the robot, which are K_x, K_y and K_h. From a practical overview, the bigger these variables are, the more action is put to correct the trajectory (minimize the pose error) of its actuating state variable.
The following analysis will concentrate on the K_x and K_y parameter tuning, letting the correction over the heading be always the best as possible. It is important to note that this section also explains how to make proper tuning of the controller based on the pose error analysis. For the next three sections, a *setVelocity* is sent with module 0.2 mps
and angle 0 rad.

5.1.1 Very soft controller: Kx = 0.0001 ; Ky = 0.02 ; Kh = 2.4

In this case, the parameters were set for the controller to make very soft corrections over each axis. The stop condition is 16mm error over xx, 10mm error over yy and 0.5 radians for the heading. This means that a virtual point is considered to be achieved if the pose error stays bellow these values at the same iteration. Referring to figure 12 (left), it can be seen that a slightly inaccurate control is made over yy. The error oscillates above zero and tends to increase as time goes by. The same happens with the error over xx however, in this case, it

can be seen that the error drops quickly every time a virtual point is achieved (at every pitch value of the blue line). As one can easily see, a more accurate tuning over yy is absolutely needed. Note that the blue line represents the error over xx, the green line, the error over yy, and the red line, the heading error all towards the settled virtual points (red dots in the visualizer screenshots).

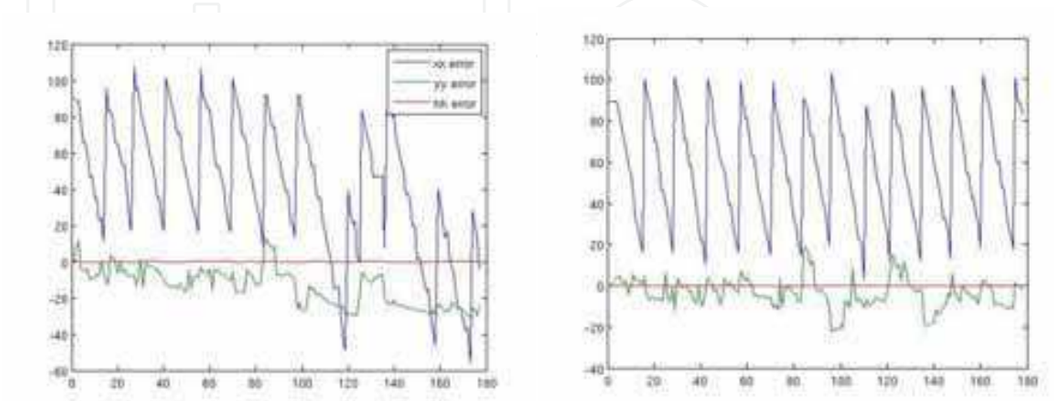

Fig. 12. `setVelocity` error for the very soft controller case (left) and `setVelocity` error for the soft controller case (right)

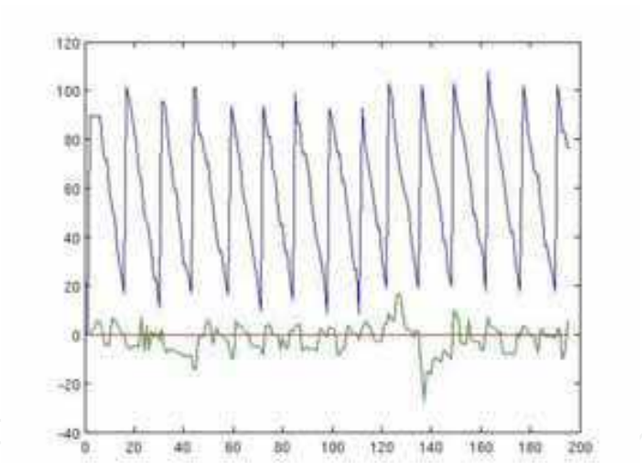

Fig. 13. *setVelocity* error for the hard controller case.

5.1.2 Soft controller: Kx = 0.0001 ; Ky = 0.09 ; Kh = 2.4

It is expected that by raising the value of K_y, the pose error results be more accurate. As it can be seen from Fig. 12 (right), the error over yy diminishes, rounding zero. Now, the robot follows his trajectory but still not with the desired accuracy.

5.1.3 Hard controller: Kx = 0.0001 ; Ky = 0.12 ; Kh = 2.4

Raising K_y a little more, better results are achieved. It can be seen that the robot possesses very little error during the trajectory execution, possessing a pitch of 2 cm at most. These

results are remarkable, and fairly validate the implemented control system as well as its architecture, since all the projected system was conceived for the controller to possess the best performance as possible.
It is important to note that the results could be improved even more by making the controller even harder by raising the weighting variables. The drawback, however, which is high overshoot in the presence of physical disturbances, must be taken in account. By this, it must be comprehended that while tuning the controller, the overshoot of the response to physical disturbances needs to be seriously taken in account. During a soccer-game, for example, the controller can't be made too hard by any means due to the high risk of collision with other robots.

5.2 Field Experiments

A high number of experimental results with the robot on the field (see Fig. 12) were carried out in order to validate the proposed real-time controller.

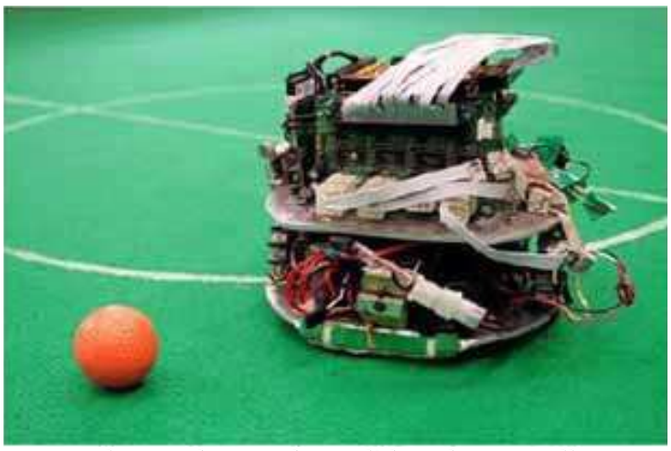

Fig. 11. RAC robot soccer player robot used to validate the controller.

The following results were achieved by setting target points on the field which the mobile robot had to reach. The images represent screenshots of the developed (and already referred in section 3.2) visualization tool that depicts an on-line virtual representation of what is happening on the (real) field. It merely receives data from the robot to update the images. Recall that red dots represent the virtual points associated with the planned trajectory, white dots are the actual (real) trajectory performed by the robot (online) and the blue mark represents the real robot when starting a *setVelocity* command. The final position is not represented. As it can be seen from both Fig. 12 and 13, the robot accurately follows the self-planned virtual point paths, which validates not only the developed controller approach but also the trajectory planning capability. It is important to underline that the virtual point path plan is made accordingly with a velocity vector with no stop point defined (visible by the red dots that go farther than the desired point to achieve. The stop point is detected by the *setTrajectory* task, which is periodically checking the distance to the desired point.
The system is very accurate even in the presence of sensor noise. It is capable of reconfiguring the trajectory if it is placed elsewhere on the environment during the motion execution, due to the effective pose error elimination of the controller. The real-time system

allows to attain dynamic responsiveness and task scheduling. For instance, due to the robust projected task priority system, the controller runs in parallel with the *setTrajectory* task which, in turn, checks periodically the distance left to the target point. The robot accurately switches the movement to reach the next target point as soon as the actual target point accomplishment is detected.

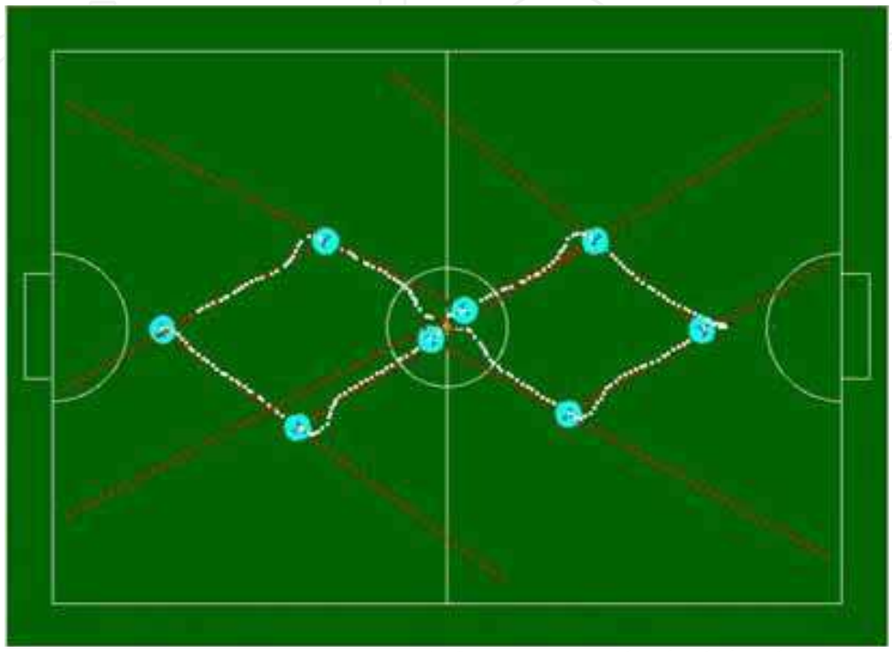

Fig. 12. Instruction `setTrajectory` - case 1: Start point is the far left robot representation and the final one is the same as the start point.

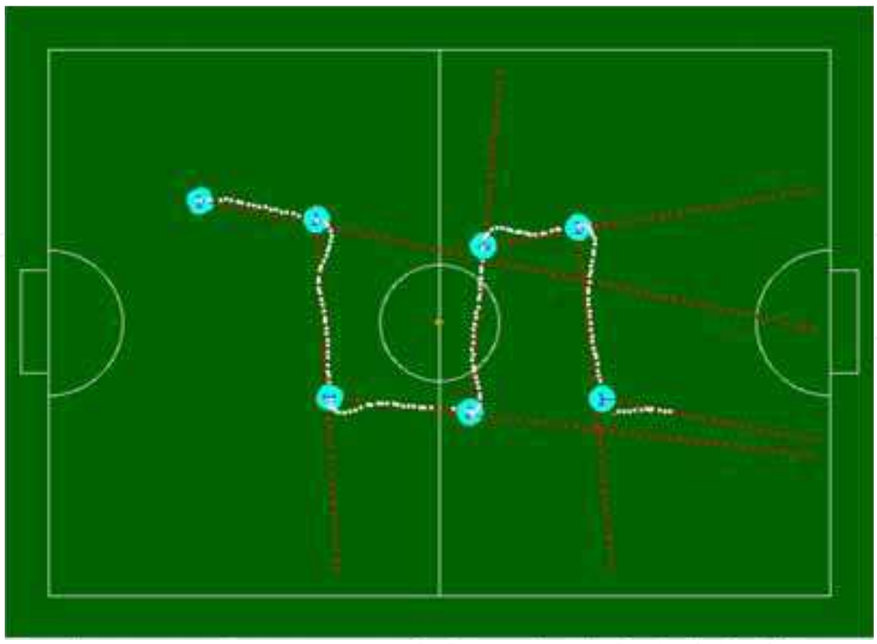

Fig. 13. Instruction `setTrajectory` - case 2: Start point is the far left robot representation and the final position is the far right white dot.

6. Conclusion

A real-time controller for pose error elimination of differential mobile robots was developed, which includes a trajectory planning algorithm. In the controller's theoretical formulation, particular emphasis was given to the design of a generic control scheme, so as to be robust against errors in the estimation of the robot's dynamic parameters. A real-time controller was designed so that specific timing constraints were also met. Experimental results obtained on a real mobile robot reveal that the presented approach is not only valid but also robust. It allows a generic differential mobile robot to correct its trajectory, leading it to converge to the desired pose.

7. References

[1] A. Gholipour, M. J. Yazdanpanah (2000). *Dynamic Tracking Control of Nonholonomic mobile robot with model reference adaptation for uncertain parameters*, University of Tehran.

[2] A. M. Bloch, M. McClamroch (1991). *Control and Stabilization of Nonholonomic Caplygin Dynamic Systems*, England

[3] A. Martinelli, R. Siegwart (1996), *Estimating the Odometry Error of a Mobile Robot During Navigation*, Autonomous Systems Lab, Swiss, Federal Institute of Technology Lausanne (EPFL)

[4] D. Wang, Guangyan Xu (2000). *Full State Tracking and Internal Dynamics of NonHolonomic Wheeled Mobile Robots*, Proceedings of the American control Conference

[5] H. Burkhard (2002). *Real time control for autonomous mobile robots*, IOS Press Amsterdam

[6] M. Vidygascar (1993). *Nonlinear Systems Analysis*, Prentice Hall, New Jersey, NJ

[7] R. Pissard-Gibollet1, K. Kapellos1, P. Rives1 Contact, J. J. Borrelly1 (1997).*Real-time programming of mobile robot actions using advanced control techniques*, Springer Berlin, Heidelberg

[8] T. Larsen, M. Bak., N. A. Andersen O. Ravn (2000).*Location Estimation for an Autonomously Guided Vehicle using an Augmented Kalman Filter to Autocalibrate the Odometry*, Technical University of Denmark

[9] Y. Kanayama, Y. Kimura, F. Miyazaki, Tetsuo Nogushi (1990). *A Stable Tracking Control Method For An Autonomous Mobile Robot*, 1990 IEEE.

[10] J. Monteiro (2008). RTAI *Installation complete guide*, https://www.rtai.org/RTAICONTRIB/RTAI_Installation_Guide.pdf

[11] J. Monteiro (2008). *Building your way through RTAI*, http://isharemyknowledge.blogspot.com/2008/03/building-your-way-through-rtai.html

[12] Real Time Application Interface, http://www.rtai.org

[13] J. Monteiro and R. Rocha (2008). *RACbot-RT: Robust Digital Control for Differential Soccer-Player Robots. In Proc. of 5th Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO'08)*, Funchal, Portugal, pages 225-228.

Contemporary Robotics - Challenges and Solutions

Edited by A D Rodi

ISBN 978-953-307-038-4
Hard cover, 392 pages
Publisher InTech
Published online 01, December, 2009
Published in print edition December, 2009

This book book is a collection of 18 chapters written by internationally recognized experts and well-known professionals of the field. Chapters contribute to diverse facets of contemporary robotics and autonomous systems. The volume is organized in four thematic parts according to the main subjects, regarding the recent advances in the contemporary robotics. The first thematic topics of the book are devoted to the theoretical issues. This includes development of algorithms for automatic trajectory generation using redudancy resolution scheme, intelligent algorithms for robotic grasping, modelling approach for reactive mode handling of flexible manufacturing and design of an advanced controller for robot manipulators. The second part of the book deals with different aspects of robot calibration and sensing. This includes a geometric and treshold calibration of a multiple robotic line-vision system, robot-based inline 2D/3D quality monitoring using picture-giving and laser triangulation, and a study on prospective polymer composite materials for flexible tactile sensors. The third part addresses issues of mobile robots and multi-agent systems, including SLAM of mobile robots based on fusion of odometry and visual data, configuration of a localization system by a team of mobile robots, development of generic real-time motion controller for differential mobile robots, control of fuel cells of mobile robots, modelling of omni-directional wheeled-based robots, building of hunter- hybrid tracking environment, as well as design of a cooperative control in distributed population-based multi-agent approach. The fourth part presents recent approaches and results in humanoid and bioinspirative robotics. It deals with design of adaptive control of anthropomorphic biped gait, building of dynamic-based simulation for humanoid robot walking, building controller for perceptual motor control dynamics of humans and biomimetic approach to control mechatronic structure using smart materials.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China

51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen