MOBILE ROBOT MODELING USING LOCAL MODEL NETWORKS

E. N. Skoundrianos and **S.G. Tzafestas**

Intelligent Robotics and Automation Laboratory
Division of Signals, Control and Robotics
School of Electrical and Computer Engineering
National Technical University of Athens
Zographou 15773, Athens, Greece
e-mail: tzafesta@softlab.ntua.gr

Abstract - This article deals with the modeling issue of plants with unknown description. The modeling approach is based on the Local Model Network structure. Local Models perform local linearization and their structure is quite similar to the Takagi-Sugeno fuzzy models. Local Neural Models (LNM) function as linear filters, giving a satisfying estimation for the plant's output within parts of the operating regime. Their training is performed off-line, which ensures a reliable method. The method is applied on the Robuter™ mobile robot.

Keywords: Local Model Networks, modeling, mobile robot

1. Introduction

The modeling problem of a mobile robot is a well studied issue in the research area of robotics and automation for at least two decades [11], [12], [13]. The difficulty in the use of classical mechanics for the production of the dynamic equations of mobile robots is one of the most significant obstacles that one has to face when dealing with the modeling issue. Even if one can produce these equations for a specified robot, the identification is not an easy task due to the complexity of the equations. For the avoidance of this obstacle one can utilize fuzzy, neural or neuro-fuzzy techniques [1], [5], [9], [14]. The utilization of several simple submodels that all together form a complex model that covers the whole of the operating regime of the plant is the main idea behind these techniques. One structure that achieves the former idea is the Local Model Network structure.

In this paper a training strategy for a Local Model Network used for modeling of mobile robots is proposed. Moreover, the proposed technique has been successfully applied to a mobile robot.

2. Modeling via Local Model Networks

2.1 General issues

The absence of an analytical model in many cases, as well as the uncertainty about the values of the model parameters when such a model is known, leads to the use of modeling techniques. In this paper Local Model Networks (LMN) are used for

system modeling, but have also been used for relative applications like control [2].

To use LMNs for system modeling one has to determine the parameters to be included in the operating point (O.P.) description. In the mobile robot's case, the voltages applied to the two robot motors ($v^L=u_1(t)$, $v^R=u_2(t)$) is satisfying information:

$$O.P. = [u_1(t)\ \ u_2(t)] = [v^L(t)\ \ v^R(t)] \quad \textbf{(1)}$$

The operating regime includes all the operating points where the plant is capable of functioning. The LMNs modeling approach suggests the division of the operating regime in parts, and the correlation of each one with a Local Model that approximates the plant behavior within the respective part of the operating regime. Linear Neural Networks are used in this paper as Local Models.

2.2 Local Model Network structure

This paper deals with Discrete Time MIMO systems. Each Local Model produces a linear estimation of the current value of the system's output, depending on the previous ones as well as on the past input values:

$$\begin{aligned}\hat{\mathbf{y}}(k) = {} & a_{11\ell}\cdot \hat{y}_1(k-1) + a_{21\ell}\cdot \hat{y}_1(k-2) + \dots \\ & + a_{n1\ell}\cdot \hat{y}_1(k-n) \\ & + a_{12\ell}\cdot \hat{y}_2(k-1) + a_{22\ell}\cdot \hat{y}_2(k-2) + \dots \\ & + a_{n2\ell}\cdot \hat{y}_2(k-n) + \dots \\ & + a_{1q\ell}\cdot \hat{y}_q(k-1) + a_{2q\ell}\cdot \hat{y}_q(k-2) + \dots \\ & + a_{nq\ell}\cdot \hat{y}_q(k-n) \qquad \textbf{(2)} \\ & + b_{11\ell}\cdot u_1(k-1) + b_{21\ell}\cdot u_1(k-2) + \dots \\ & + b_{m1\ell}\cdot u_1(k-m) \\ & + b_{12\ell}\cdot u_2(k-1) + b_{22\ell}\cdot u_2(k-2) + \dots \\ & + b_{m2\ell}\cdot u_2(k-m) + \dots \\ & + b_{1p\ell}\cdot u_p(k-1) + b_{2p\ell}\cdot u_p(k-2) + \dots \\ & + b_{mp\ell}\cdot u_p(k-m)\end{aligned}$$

hence

$$\begin{aligned}\hat{\mathbf{y}}(k) = {} & A_{1\ell}\cdot \hat{\mathbf{y}}(k-1) + \dots + A_{n\ell}\cdot \hat{\mathbf{y}}(k-n) \\ & + \mathbf{b}_{1\ell}\cdot\mathbf{u}(k-1) + \dots + \mathbf{b}_{m\ell}\cdot\mathbf{u}(k-m) \qquad \textbf{(3)}\end{aligned}$$

$$, \boldsymbol{\ell} = (\ell_1, \ell_2, \dots, \ell_\mu)\ ,\ \ell_{1\dots\mu} = 1,\dots,\delta$$

where:

μ is the number of parameters in the operating point

δ is the number of parts each parameters range is divided into.

n, m are the model classes with respect to the output and input signals, p and q are the numbers of inputs and outputs respectively.

The vector $\boldsymbol{\ell}$ points to a Local Model or the correlated part of the operating regime.

In the special case n=m=1 the model takes the following simple form:

$$\hat{\mathbf{y}}(k) = A\cdot \hat{\mathbf{y}}(k-1) + \mathbf{b}\cdot\mathbf{u}(k-1) \quad \textbf{(4)}$$

Equation (3) gives the description of the next output value estimation each LM performs. This estimator (predictor) is trained within a specific part of the operating regime. The division of the operating regime in parts takes place before training. The range of each variable included in the operating point description is determined and then divided in δ sub-ranges. The desirable operating regime parts are combinations of these sub-ranges for all variables. Since there are μ operating point parameters and each range is divided into δ sub-ranges, there are $r = \delta^\mu$ operating regime parts and respective Local Models.

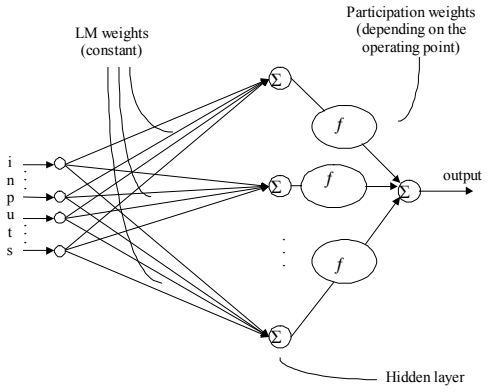
Once there is a LM trained for each part of the operating regime, the training is complete and the model functions as shown in eq (5). Of course instead of having a winner-take-all strategy (which is the case in eq. 5), it is always possible to have a multi-model participation for the model output.

This is accomplished by using participation weights, depending on the operating point. At each time, the winner network is the network correlated to the part of the operating regime where the plant is currently operating.

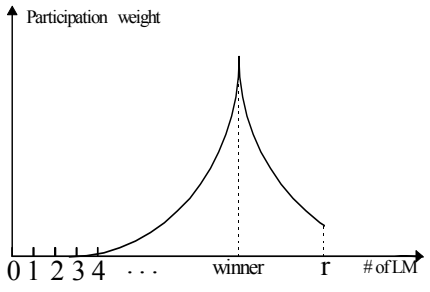
$$\begin{aligned}
\hat{\mathbf{y}}(k) &= A_{1\ell_1}\cdot\hat{\mathbf{y}}(k-1)+\ldots+A_{n\ell_1}\cdot\hat{\mathbf{y}}(k-n)\\
&\quad+\mathbf{b}_{1\ell_1}\cdot\mathbf{u}(k-1)+\ldots+\mathbf{b}_{m\ell_1}\cdot\mathbf{u}(k-m)\ ,\ \text{O.P.}\in P_{\ell_1}\\
\hat{\mathbf{y}}(k) &= A_{1\ell_2}\cdot\hat{\mathbf{y}}(k-1)+\ldots+A_{n\ell_2}\cdot\hat{\mathbf{y}}(k-n)\\
&\quad+\mathbf{b}_{1\ell_2}\cdot\mathbf{u}(k-1)+\ldots+\mathbf{b}_{m\ell_2}\cdot\mathbf{u}(k-m)\ ,\ \text{O.P.}\in P_{\ell_2}\\
&\quad\ldots.\\
\hat{\mathbf{y}}(k) &= A_{1\ell_r}\cdot\hat{\mathbf{y}}(k-1)+\ldots+A_{n\ell_r}\cdot\hat{\mathbf{y}}(k-n)+\\
&\quad+\mathbf{b}_{1\ell_r}\cdot\mathbf{u}(k-1)+\ldots+\mathbf{b}_{m\ell_r}\cdot\mathbf{u}(k-m)\ ,\ \text{O.P.}\in P_{\ell_r}
\end{aligned}\qquad \textbf{(5)}$$

,where r is the number of Local Models and P_{ℓ} ($\ell = \ell_1, \ldots, \ell_r$) are the operating regime parts.

The LMNs structure, along with a possible participation weight function, is shown in Fig. 1.



a) LMN structure



b) participation weights

Fig. 1 Local model network structure

2.3 Training

The training of the LMN must result to matrices $A_{i\ell}$, vectors $\mathbf{b}_{j\ell}$ and parts P_{ℓ}, where i=1,...,n , j=1,...,m and $\ell = \ell_1,\ldots,\ell_r$. The training method proposed in this paper is an *off-line* method. Input and output samples are obtained within each operating part, and then the recursive process of adjusting the weights is applied in order to match model and plant behavior.

Each Local Model is trained using the following strategy:

- Initial values $\mathbf{y}(0)$, $\hat{\mathbf{y}}(0)$ and $\mathbf{u}(0)$ are randomly chosen within the respective operating regime part.
- At each time k we provide a system input $\mathbf{u}(k)$ and measure the output $\mathbf{y}(k)$. The system input must obviously vary within the range specified by the respective part. A constant value equal to the initial one ($\mathbf{u}(k) = \mathbf{u}(0)$ for each k), is an effective choice.
- Model output:
  $\hat{\mathbf{y}}(k) = A_{1\ell}\cdot\mathbf{y}(k-1)+\ldots+A_{n\ell}\cdot\mathbf{y}(k-n) + \mathbf{b}_{1\ell}\cdot\mathbf{u}(k-1)$
  $+\ldots+\mathbf{b}_{m\ell}\cdot\mathbf{u}(k-m)$
  and error: $\mathbf{e}(k) = \mathbf{y}(k) - \hat{\mathbf{y}}(k)$
  are calculated.
- A training rule is used (delta rule, recursive least squares, etc.) to adjust the parameters which function as weights on a linear neural network, until the error drops below a threshold. Here the recursive least squares (RLS) rule is employed [7].

The above process may be repeated, without resetting the weights (parameters) of course, using other initial values of the same operating regime part, in order to ensure that each Local Model captures the whole behavior of the plant and the weights converge.

3. Application to the Robuter™ mobile robot

The proposed approach was applied to the robot shown in figure 2. This robot is named Robuter and is manufactured by the RoboSoft company.

The main purpose of the robot is to move autonomously in indoor environments and perform service tasks. Detailed information about the RoboSoft's structure and figures can be found in [10].

Fig. 2: Picture of the RoboSoft

In the case of the mobile robot, the model describes the relationship between the voltage applied to the wheels and the speed of the wheels. This is a non-linear relationship mostly due to the static friction and the non-linear torque curve of each motor. The combination of linear sub-models with local validity is the way LMNs deal with non-linearity, as described earlier.

The ℓ-th subnetwork of the LMN structure should have the following general form:

$$\begin{aligned}\omega^L(k) = {} & a^L_{0\ell}\cdot \omega^L(k) + a^L_{1\ell}\cdot \omega^L(k\text{-}1) + \dots \\ & + a^L_{n\ell}\cdot \omega^L(k\text{-}n) \\ & + b^L_{0\ell}\cdot v^L(k) + b^L_{1\ell}\cdot v^L(k\text{-}1) + \dots \\ & + b^L_{m\ell}\cdot v^L(k\text{-}m)\end{aligned} \qquad \textbf{(6)}$$

$, \boldsymbol{\ell} = (\ell_1, \ell_2)\ ,\ \ell_1, \ell_2 = 1,\dots,\delta$

(δ : the number of parts the range of each parameter –within the operating point- is to be divided).

Naturally, there is an identical equation for the right wheel.

The model described is an equation of differences (since the method deals with discrete-time systems) between the input $v_L(k)$ which represents the amount of voltage being sent to the left motor, and the speed $\omega_L(k)$.

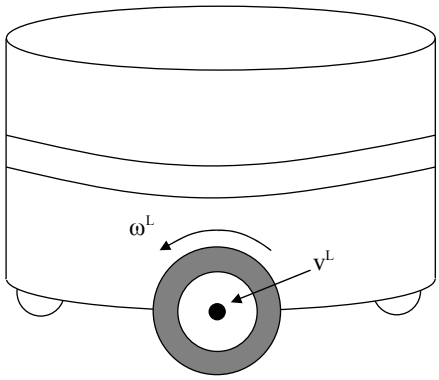


Fig. 3 Input and output signals (left wheel)

A priori knowledge of the system can be used for the determination of the orders m and n. The order n can be zero if the transitory state is unimportant, which is the case here. The order m is chosen to be 1 for better interpretation of the results.

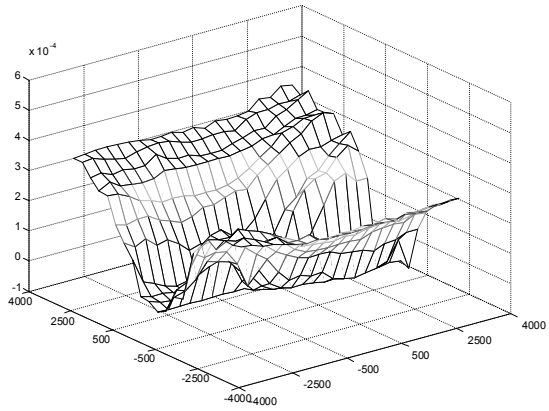
Hence, we have:

$$\omega^L(k) = b^L_{0\ell}\cdot v^L(k)\ ,\ \boldsymbol{\ell} = (\ell_1, \ell_2) \qquad \textbf{(7)}$$

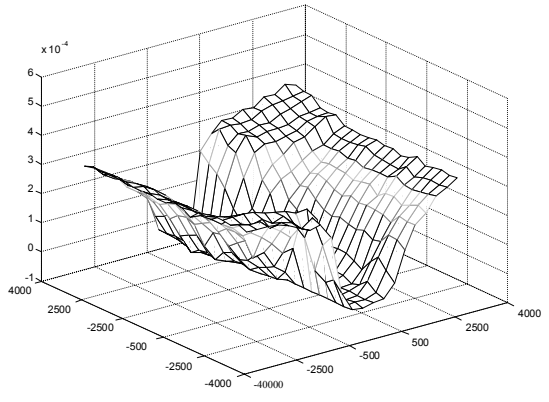
The cross-dependence between the two wheels is another major issue and must also be included in the model. If, for instance, there is zero voltage on one wheel, the motor will act like a brake, and the static friction produced by the unmovable wheel will also affect the other wheel. Since the coefficients of eq. (6) and (7) depend on the operating point which includes both voltages v^L , v^R the model takes the cross-dependence between the two wheels into account.

The input voltage on each wheel is a parameter within the range -4000 , … , 4000 (techni-

cally this range can be increased up to -10000, … , 10000). The bi-dimensional operating regime, formed by the two parameters (v^L, v^R), must be divided into parts. Each local model is then trained within the respective part of the operating regime, as described earlier. This procedure results to the attainment of the coefficients $b^L{}_{0\ell}$ and $b^R{}_{0\ell}$ which are presented in figure 4.



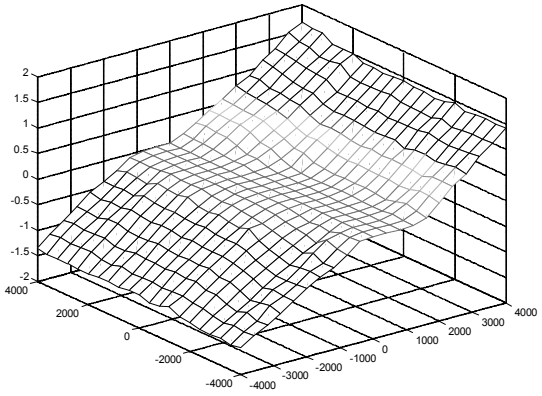
(a)



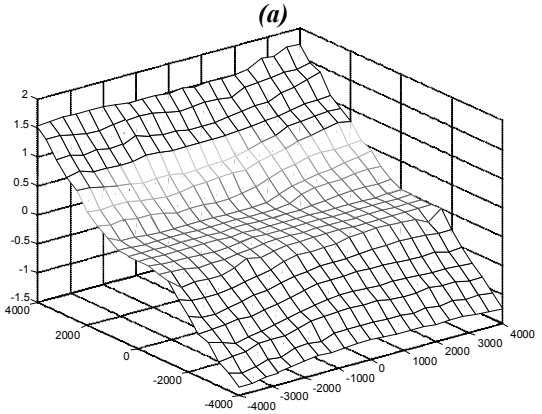
(b)

Fig. 4: The coefficients (a) $b^L{}_{0\ell}$ and (b) $b^R{}_{0\ell}$

Due to the simplicity of eq. (7), one could easily produce the relationship between the two input voltages v^L, v^R and the speeds of both wheels ω^L, ω^R. This is shown on fig. 5. The resulting relationship has the form of a dead-zone function. This is a typical behavior in electrical motors and is related to a number of phenomena (i.e. static friction).



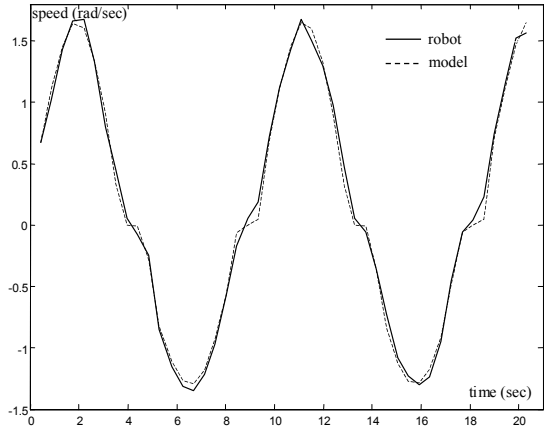
(a)



(b)

Fig.5: relationship between (a) input v^L and output ω^L, (b) input v^R and output ω^R.

The model was tested on the robot and some of the produced results are presented in figures 6-7.



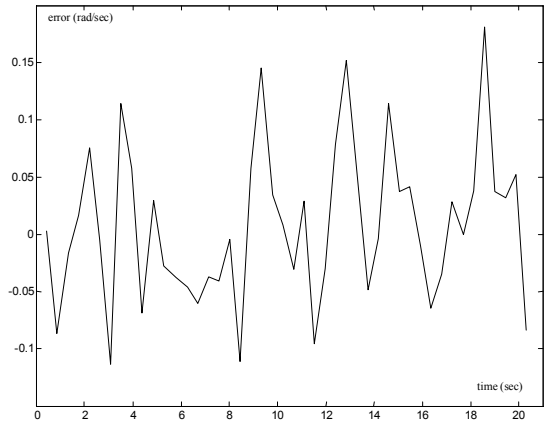
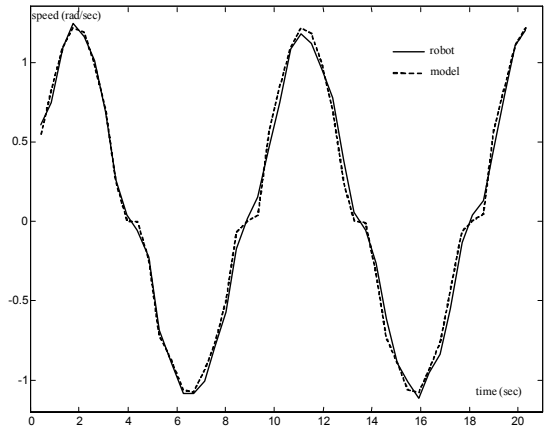


Fig. 6a: model performance and error (sinus input) for the left wheel



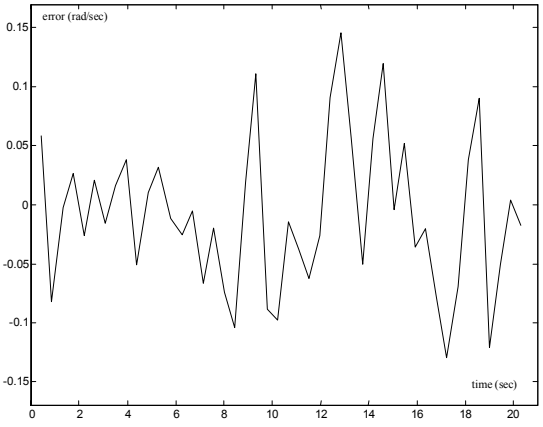
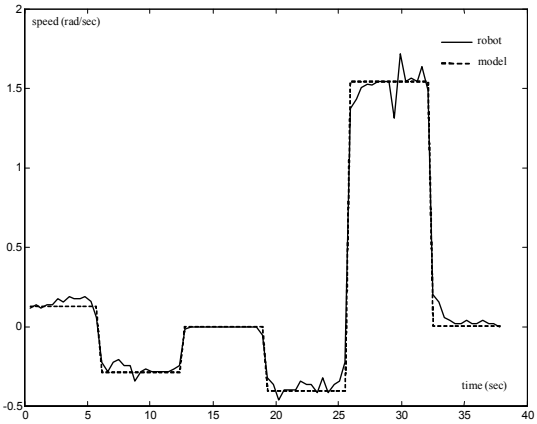


Fig. 6b: model performance and error (sinus input) for the right wheel



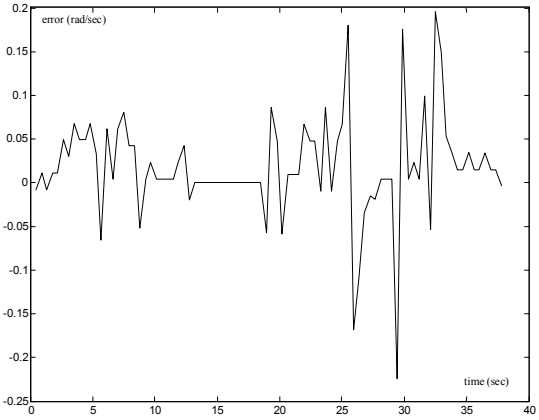


Fig. 7a: model performance and error (varying step input) for the left wheel

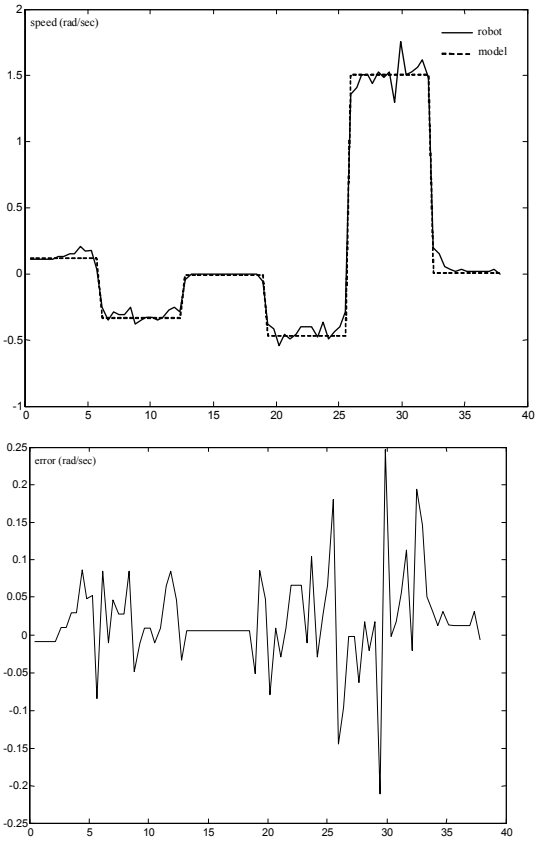


Fig. 7b: model performance and error (varying step input) for the right wheel

4. Discussion Issues

In the following we provide a discussion of some points which need some further clarification:

1. The architecture of the robot includes a PC and the subsystem of the robot's base that controls the motors and sensors of the robot. The base is operated by the Albatros™ operating system that runs under a Motorolla 68000 processor. The commands are sent to the base from the PC via the serial port. This setup results to a very convenient data acquisition subsystem.

2. The angular velocity is not measured directly since no relative option is provided by the manufacturer. The command "pos" uses the proprioceptive sensors (internal sensors, for instance optical encoders or potentiometers) to calculate the position of the wheels (P_L, P_R). The command loop time ΔT is also measured by the PC application (a C++ program), and the velocity is calculated as:

$$V_L(k) = \frac{P_L(k) - P_L(k-1)}{\Delta T}.$$

3. The velocity commands are issued directly to the motors (open loop – not regulated by a low-level PID control process). This is adjusted by the Albatros "serv" command. The command "move" is applying the voltage to the wheels.

4. Concerning the selection of the number of the local models and the criteria to establish the bounds, we would like to point out the following. The method could be enhanced in a way that could allow automatic detection of the bounds of the local models. This could be implemented by compressing or expanding the model bounds during training in order to keep the training error within some predefined limits. This adjustment however, would dramatically increase the computational complexity without analogous increase of the method's efficiency. The use of some a priori knowledge of the system to be modelled (usually available) is a much more efficient way to deal with this issue. One could increase the number of local models (and use relatively compressed bounds) in the areas where the non-linearities of the system are heavier. However, if this knowledge is unavailable, one could start with an even division of the operating regime and later, if necessary, repeat the training by adding local models in areas where some mismatch is observed.

5. Some mismatches can be noticed, i.e. in figure 6a and 6b near the low velocities range (where the presence of friction and motor "dead-zone" is in fact vivid). The model is built in this sense in order to deal with all non-linearities mentioned and, ob-

viously, includes friction and the "dead-zone" of the motor actuators. If these non-linearities were unimportant then the operating regime division would be unnecessary and a linear model would work satisfactorily. However, the training was performed under the assumption of an even distribution of models throughout the operating regime. Also, the number of the models was selected relatively small (about 400) in order to achieve a training time of less than an hour. In order to improve the performance of the model in a neighbourhood of the operating region, one could select a higher number of models to cover the specific area. Therefore this is not really a handicap of the method, but more of a result of the selection of the training parameters.

5. Conclusions

In this paper we have shown how the Local Model Network structure can be used for mobile robot modeling. The proposed method is based on the general idea of local linearization. Linear models are trained locally and then are combined together to produce the overall model. The paper proposes a generic, yet very suitable for the mobile robot case, training strategy. The results shown in this paper are very satisfactory since, using the proposed method a model for the Robuter™ mobile robot is created. The model achieves sufficiently small modeling errors, and has been used in a series of applications i.e. control and fault diagnosis, where the comparison with other methodologies [3], [4], [6], [8] has shown better performance in many cases.

6. References

1. Babuska R., *Fuzzy Modeling for Control*. Kluwer Academic Publishers, Boston, 1998.
2. Brown M.D., Lightbody G., Irwin G.W. *Nonlinear internal model control using local model networks*, IEE Proc.-Control Theory Appl., Vol. 144, No. 6, November 1997
3. Desai J.P., Ostrowski J.P. and Kumar V., *Modeling and Control of Formations of Nonholonomic Mobile Robots*, IEEE trans. on Robotics and Automation, Vol.17, No. 6, 2001, pp. 905-907.
4. Dixon W.E., Walker I.D., Dawson D.M. and Hartranft J.P., *Fault Detection for Robot Manipulators with Parametric Uncertainty: A prediction-Error Based Approach*, IEEE trans. on Robotics and Automation, Vol.16, No. 6, 2000, pp. 689-699.
5. Driankov D., Eklund P.W. and Ralescu A.L., *Fuzzy Logic and Fuzzy Control*, Springer Verlag, 12/1994.
6. Frank P.M., Alcora Garcia E. and Koppen-Seliger B., *Modelling for Fault Detection and Isolation versus Modelling for Control*, Mathematical and Computer Modelling of Dynamical Systems, Vol. 7, No. 1, 2001, pp. 1-46.
7. Haykin S., *Adaptive Filter Theory*, Prentice-Hall Inc. (ISBN: 0-13-322760-X, Englewood Cliffs, N.J.), 1996.
8. Kim Y.H. and Lewis F.L., *Neural Network Output Feedback Control of Robot Manipulators*, IEEE trans. On Robotics and Automation, Vol.15, No. 2, 1999, pp. 301-309.
9. Tzafestas C.S. and Tzafestas S.G., *Fuzzy and Neurofuzzy Approaches to Mobile Robot Path and Motion Planning Under Uncertainty*, In: S.G. Tzafestas (ed.), Soft Computing in systems and Control Technology, World Scientific, Singapore/London, 1999, pp. 193-220.

10. Tzafestas C.S. *Teleplanning by Human Demonstration for VR-based Teleoperation of a Mobile Robotic Assistant*. Proc of the 10th IEEE International Workshop on Robot-Human Interactive Communication (ROMAN'2001) 2001, pp. 462-467.
11. Tzafestas S.G. and Tzafestas E.S., *Learning, Reasoning, and problem solving in Robotics*, In: S.Y. Nof (ed.), Handbook of Industrial Robotics (ch.20), John Wiley & Sons, 1999, pp. 373-392.
12. Tzafestas S.G., Tzamitzi M.P. and Rigatos G.G., *Autonomous Robot Motion Planning and control in Uncertain Environments: Overview and a New Algorithm Based on Sliding-Mode Control*, In: S.G. Tzafestas (ed.) Advances in Intelligent Autonomous Systems, Kluwer, Dordrecht/Boston, 1999, pp. 267-288.
13. Watanabe K. Shiraishi Y., Tzafestas S.G., Tank J. and Fukuda T., *Feedback control of an omnidirectional autonomous platform for mobile service robots*, J. Intel and Robotic Syst., Vol. 22, Nos. 3-4, 1998, pp. 315-330.
14. Zavlangas P.G. and Tzafestas S.G., *Industrial Robot Navigation and Obstacle Avoidance Employing Fuzzy Logic*, J. Intell. & Robotic Systems, Vol. 27, Nos. 1-2, 2000, pp. 85-97.