A Matlab/Simulink Toolbox for Inversion of Local Linear Model Trees

Mirko Nentwig * Paolo Mercorelli*

Abstract—**Models in the form of characteristic diagrams or more specifically, in the form of engine operating maps are mostly used in the automobile industry. This yields a large amount of measurements and involves the use of advanced instrumentations. This paper shows a developed software environment, namely a toolbox for the program Matlab/Simulink developed by company Mathworks. The name of the toolbox is "Inversion of the Local Linear Model Trees" and it basically consists of a local inversion of the Local Linear Model Trees (LOLIMOT). The importance of the inversion in control problems is widely known. Neural networks are a very effective and popular tools mostly used for modeling. The inversion of a neural network produces real possibilities to involve the networks in the control problem schemes. The developed toolbox is explained with the help of diagrams and GUI structure from Matlab which tend to clarify the idea of the program and its structure. The presentation is organized as a short tutorial of the toolbox, so that a potential user can directly understand how to access it. Nevertheless, formal mathematical equations, concerning the neural networks and membership functions, need to be explained together with the LOLIMOT structure. To validate and to clarify the explained toolbox, an example from a system used in the automobile industry is briefly shown.**

Keywords: ***Neuro-fuzzy identification, nonlinear systems, local linear model trees (LOLIMOT), inversion, Matlab/Simulink***

1 Introduction and Motivation

The inversion problem has always been an attractive problem for the control area. In fact, through an inversion of the model it is possible to do a feed-forward action which can represent an important part, very often adopted in modern control techniques. The feed-forward action is applied, for instance, in tracking problems, stationary correction of the error, control with two degrees of freedom, estimation of the inputs and in general when some pre-actions are needed in order to prepare the system with regard to energy. In [5] a controller based on inversion using a physical model approach is presented. The model which is adopted suffers from some hard reductions in order to be applied. Nowadays neural networks are often adopted for modeling of complex systems. In the neural network area the inversion problem attracted a lot of researchers and mathematicians. The inversion problem is a very hard problem which involves, in the case of neural networks, the inversion of the membership functions which are non linear functions. In Fig. 1 a schematic structure of a possible control system is represented. From the picture the importance of the inversion of the neural network in particular for tracking problems emerges. Due to the inexact description of the model, its imperfect inversion as well as the presence of not modeled external disturbances imply that the controller must consist of a structured feedback part. This

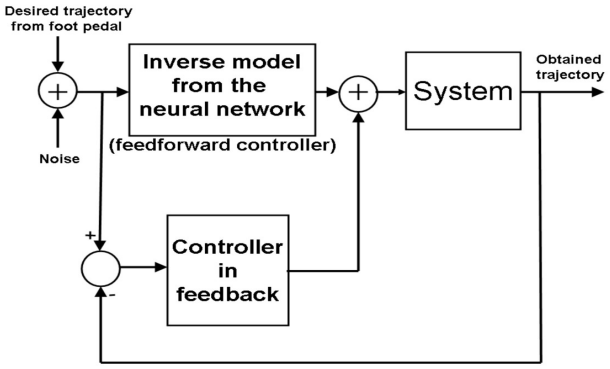

Figure 1: Complete Control Scheme

procedure applies the LOLIMOT algorithm [6], which is based upon Neural-Fuzzy models of Takigi Sugeno type. During the execution of this algorithm, a "divide and conquer strategy" is applied to the modeling problem, so that the major problem is split into smaller ones.

1.1 Structure

The basic network structure of a local linear neuro fuzzy model is depicted in Fig. 2. Every neuron consists of a local linear model, a so called LLM, and a validity function ϕ , which defines the validity of the LLM within the input space.
The local linear model output is defined by:

$$\hat{y}_i = w_{i0} + w_{i1}u_1 + w_{i2}u_2 + ... + w_{ip}u_p$$

*University of Applied Sciences Ostfalia, Department of Automotive Engineering, Robert-Koch-Platz 10 - 14, 38440 Wolfsburg, Germany, Tel. +49-(0)5361-831615 Fax. +49-(0)5361-831602, E-Mail: mail@mnentwig.de, p.mercorelli@ostfalia.de

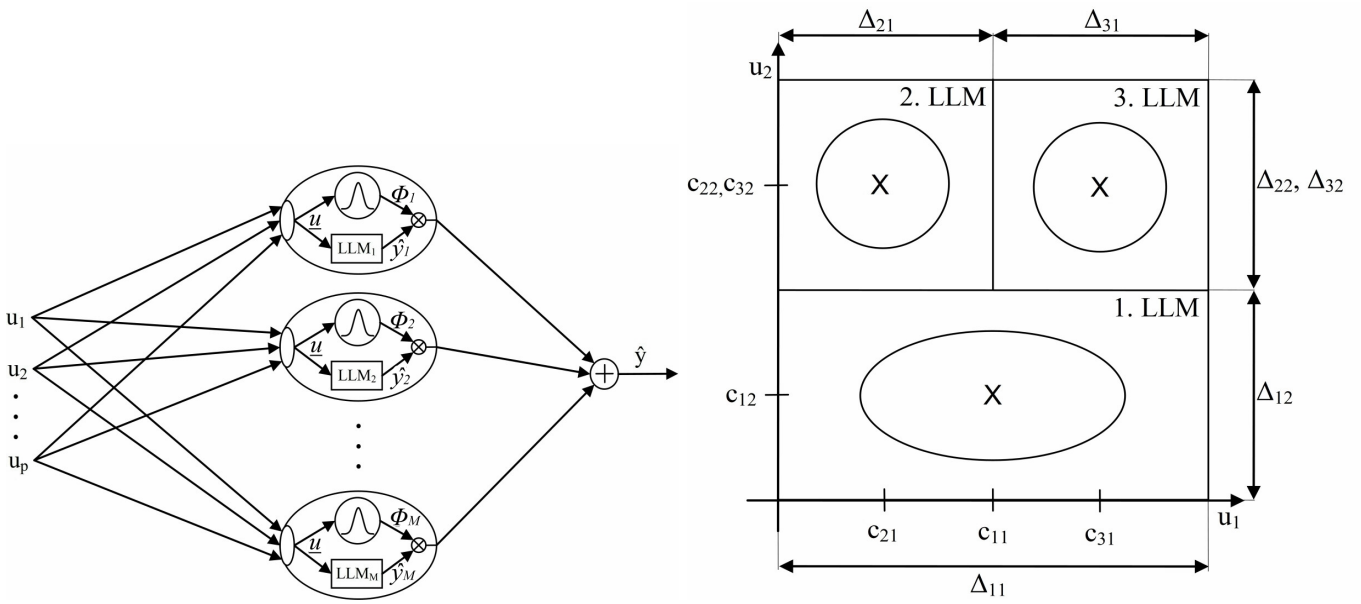

Figure 2: Left: Network Structure of Local Linear Neuro Fuzzy Model. Right: Partitioning of Input Space by Validity Functions

with w_{ij} as parameters of neuron i.
If the validity functions are chosen as normalized Gaussians, then:

$$\sum_{i=1}^{M} \Phi_i(\underline{u}) = 1 \text{ and } \Phi_i = \frac{\mu_i(\underline{u})}{\sum_{j=1}^{M} \mu_j(\underline{u})},$$

where the membership function μ is defined as

$$\mu_i(\underline{u}) = exp\left(-\frac{1}{2}\frac{(u_1 - c_{i1})^2}{\sigma_{i1}^2}\right) + exp\left(-\frac{1}{2}\frac{(u_2 - c_{i2})^2}{\sigma_{i2}^2}\right) + ... + exp\left(-\frac{1}{2}\frac{(u_p - c_{ip})^2}{\sigma_{ip}^2}\right) \quad (1)$$

.

1.2 Estimation of Linear Parameters

The estimation of the linear equation parameters is done through an optimization by local least squares method. This method offers a fast possibility for estimating the rule conclusions. The idea of the local estimation is to treat the optimization problem of every conclusion as an individual one. Instead of estimating all $n = M(p+1)$ parameters parallel, M single local estimations for every $p+1$ parameters of the model are carried out.
The parameter vector for each $i = 1, ..., M$, and the regression matrix are respectively:

$$\underline{w}_i = \begin{bmatrix} w_{i0} \\ w_{i1} \\ \vdots \\ w_{ip} \end{bmatrix} \text{ and}$$

$$\underline{X}_i = \begin{bmatrix} 1 & u_1(1) & u_2(1) & \dots & u_p(1) \\ 1 & u_1(1) & u_2(1) & \dots & u_p(1) \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & u_1(N) & u_2(N) & \dots & u_p(N) \end{bmatrix}.$$

It should be noted that the regression matrices of all linear models $i = 1, ..., M$ are identical, because all the elements of $\underline{X}_i$ do not depend on i. Due to the structure of the local linear models with their limited validity, it's useful to apply a weighted least squares. The validity factors are described by the validity functions. This leads to the following NxN weight matrix

$$\underline{Q}_i = \begin{bmatrix} \phi_i(\underline{u}(1)) & 0 & \dots & 0 \\ 0 & \phi_i(\underline{u}(1)) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \phi_i(\underline{u}(N)) \end{bmatrix}.$$

The weighted least squares solution of the rule conclusion parameters is given by

$$\underline{\hat{w}}_i = \left(\underline{X}_i^T \underline{Q}_i \underline{X}_i\right) \underline{X}_i^T \underline{Q}_i \underline{y}.$$

This estimation has to be carried out for all $i = 1, ..., M$ models.

1.3 Structure Optimization of Rule Premises

The local least squares method mentioned for estimating the rule conclusions is only applicable if the validity functions have been estimated first.
The number of the validity functions and their parameters define the partitioning of the input space. The validity functions are depicted in Fig. 2. These divide the

input space in rectangle areas shown in Fig. 2. If normalized Gaussians are used, then c_{ij} describe the centers of the rectangles and the standard deviations σ_{ij} describe the extension in each dimension. The proportional factor k_σ relates the standard deviation of the validity functions to the extension of the rectangle by $\sigma_{ij} = k_\sigma \cdot \Delta_{ij}$.

1.4 Local Linear Model Tree Algorithm (LOLIMOT)

The LOLIMOT is an incremental tree construction algorithm, which divides the input space axes in an orthogonal way. By iterations, one new local linear model is added to the model. To do that, validity functions are calculated and the local linear models are adapted with the least squares method. The only fiddle parameter is k_σ. This is a proportional factor between the size of the rectangles and the standard deviation. The optimal value depends on the application, but generally the following delivers good results: $k_\sigma = \frac{1}{3}$. The Standard deviations are calculated as follows: $\sigma_{ij} = k_\sigma \cdot \Delta_{ij}$, where Δ_{ij} describes the extension of the hyper rectangles of model I in dimension u_j depicted in Fig. 2. The LOLIMOT algorithm consists of an *external loop* (external algorithm) in which the structure of the model is determined and an *inner loop* (internal algorithm) in which the parameters of the model are estimated by local least squares.

Short Description of the Algorithm

1. Start with an initial model
   Construct the validity functions for the beginning input space portioning and estimate the LLM parameters by local least squares. Set M to starting number of LLM. If no starting input space division is available set $M = 1$ and start with one LLM, which covers the whole input space.

2. Find the worst LLM
   Calculate the loss function for each of the $I = 1, ..., M$ local linear models. A local loss function is used, with a squared model error:

   $$I_i = \sum_{j=1}^{N} e^2(j)\phi_i(\underline{u}(j)).$$

   Find the worst LLM, where $\max_i(I_i)$. i defines the index of the worst LLM.

3. Check all possible divisions
   The worst LLM i will be considered for further optimizations. Each hyper rectangle of the local linear models will be divided orthogonally along the axes in two pieces. Divisions in every of p dimensions are carried out. By each division the following steps are carried out
   - 3a: Construction of the n-dimensional membership functions for every hyper rectangle
   - 3b: Construction of the validity functions
   - 3c: Local estimation of the parameters which belong to the newly created local linear models
   - 3d: Calculation of the loss functions for the complete model

4. Find the best division
   The best among p alternatives of step 3 is chosen. The validity functions which were constructed in step 3a and the optimized local linear models in step 3c, will be added to the model. The number of LLM are incremented $M \rightarrow M + 1$.

5. Check for convergence
   If the final criteria is met, then stop the algorithm, otherwise return to step 2
   More than one possibility exists for the final criteria, e.g. maximal model complexity that means maximal model count, statistical judges or information reliability.

In Fig. 3 the LOLIMOT algorithm for the first five iteration steps with a two dimensional input space is depicted. Especially two features make the algorithm extremely fast. First, during every iteration not all local linear models are considered for the division. Step 2 selects only the worst local linear model which may gain the biggest performance improvement. For example in iteration step four only the LLM 4-4 is considered for further optimization. All other LLMs remain unchanged.
Secondly, in step 3c the local least squares method makes it possible that only the parameter of the two newly created LLM needs to be estimated. For Example, if in the fourth iteration step the LLM 4-4 is divided in LLM 5-4 and 5-5 the local linear models 4-1, 4-2 and 4-3 can be directly taken to the next iteration loop 5-1, 5-2 and 5-3 - without any changes.

2 LOLIMOT Inversion

In this paragraph the inversion of the local linear model tree is described. The idea is to develop an algorithm which returns the required model input u_r depending on the desired model output y and the other model inputs $\underline{u}$. Fink and Toepfer in [1] offer some strategies in their analysis of the inversion of nonlinear models:

- Inverse access by numerical inversion
  Only one model of the nonlinear function is created. This model is used for standard and inverse access.

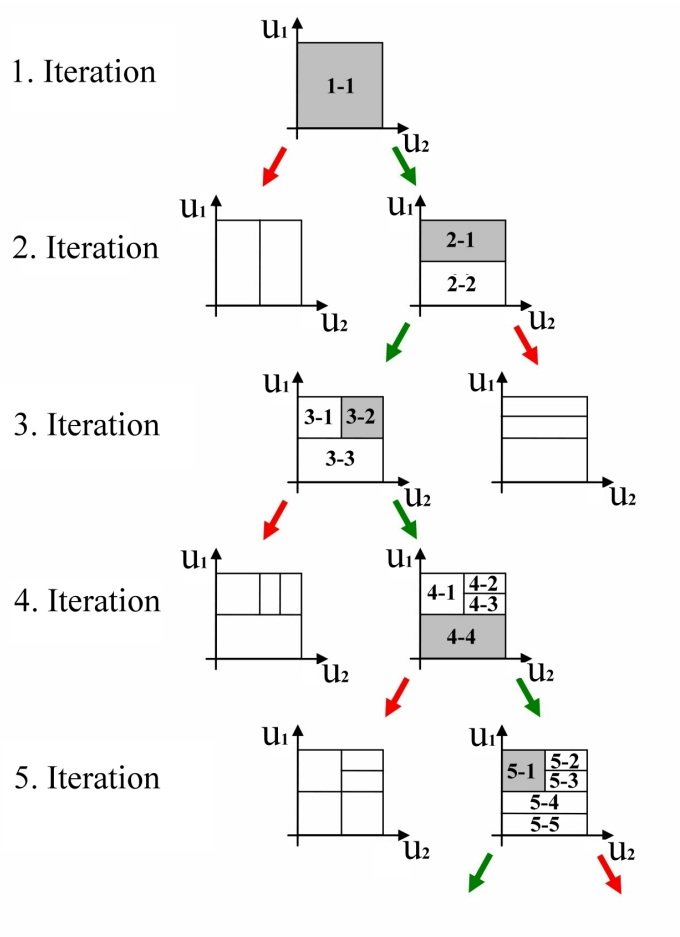

Figure 3: LOLIMOT Algorihm

The inverse access equals a numerical inversion and requires the application of optimization methods to find the required input for the requested output.

- Data driven generation of an inverse model
  As well as the model for feed forward access, a model for inverse access will be created.
- Analytical inversion of models
  This method applies a direct inversion of the forward trained model. Hence it's an advantage to use model architectures which allow the direct calculation of the inverse model by its own parameters.

The developed algorithm applies an analytical/numerical inversion of a given local linear model structure.
To achieve this goal, the following constraints are needed:

1. An existing forward trained local linear model tree of the process is available
2. An expected model output y is known
3. The input values of the other inputs on which y is depending are known

2.1 The Validity Functions Issue

Consider the model output function $\hat{y}$, with M local linear model $\underline{u} = [u_1, ..., u_p]$ inputs

$$\hat{y} = \sum_{i=1}^{M} (w_{i0} + w_{i1}u_1 + + w_{i2}u_2 + ... + w_{ip}u_p)\Phi_i(\underline{u}), \quad (2)$$

the validity function

$$\Phi_i = \frac{\mu_i(\underline{u})}{\sum_{i=1}^{M} \mu_j(\underline{u})}$$

and the membership function

$$\mu_i(\underline{u}) = exp\left(-\frac{1}{2}\left(\frac{(u_1 - c_{i1})^2}{\sigma_{i1}^2}\right)\right) + exp\left(-\frac{1}{2}\left(\frac{(u_2 - c_{i2})^2}{\sigma_{i2}^2}\right)\right) + ... + exp\left(-\frac{1}{2}\left(\frac{(u_p - c_{ip})^2}{\sigma_{ip}^2}\right)\right). \quad (3)$$

A difficulty is that, due to the exponential quadratic nonlinearity, the model is not invertible. Hence, it is necessary to convert the functions into a linear type, as shown in Fig. 4. The membership function is split into a spline function that consists of the linear functions.

$$\mu_{ir} = \begin{cases} \frac{k_\sigma}{1.6 \cdot \sigma_r}(u_r - c) + 1 & -\frac{1.6\sigma_r}{k_\sigma} + c \le u_r \le c \\ -\frac{k_\sigma}{1.6 \cdot \sigma_r}(u_r + c) + 1 & c \le u_r \le \frac{1.6\sigma_r}{k_\sigma} + c. \end{cases} \quad (4)$$

Using the function defined in equation (4), it is now possible to invert the local linear model tree.

2.2 Algorithm for the Inversion of the LLM

We apply the following algorithm to invert the model:

1. Calculate the LLMs represented in equation (2), omitting the required input variable u_r. That is, calculate the LLMs with the available input data until there only remains a linear equation depending on one input, e.g., $y_i = w_r \cdot u_r + u_{calc}$. To be more comprehensible, if $\hat{y}_i = w_{i0} + w_{i1}u_{i1} + w_{i2}u_{i2} + w_{i3}u_{i3}$ and u_{i1} is required, then $y_i = w_{i1}u_{i1} + u_{calc}$, with $w_r u_r = w_{i1}u_{i1}$ and $u_{calc} = w_{i0} + w_{i2}u_{i2} + w_{i3}u_{i3}$.
2. Calculate the membership functions represented in equation (3), omitting the required input variable u_r. The membership function of the LLMs is calculated with the available input data to the extent possible. Since the nonlinear term depending on u_r is omitted, the membership function is a constant number. To be more precise,

$$\mu_c(\underline{u}) = exp\left(-\frac{1}{2}\left(\frac{(u_2 - c_{i2})^2}{\sigma_{i2}^2}\right)\right) + ... + exp\left(-\frac{1}{2}\left(\frac{(u_p - c_{ip})^2}{\sigma_{ip}^2}\right)\right). \quad (5)$$

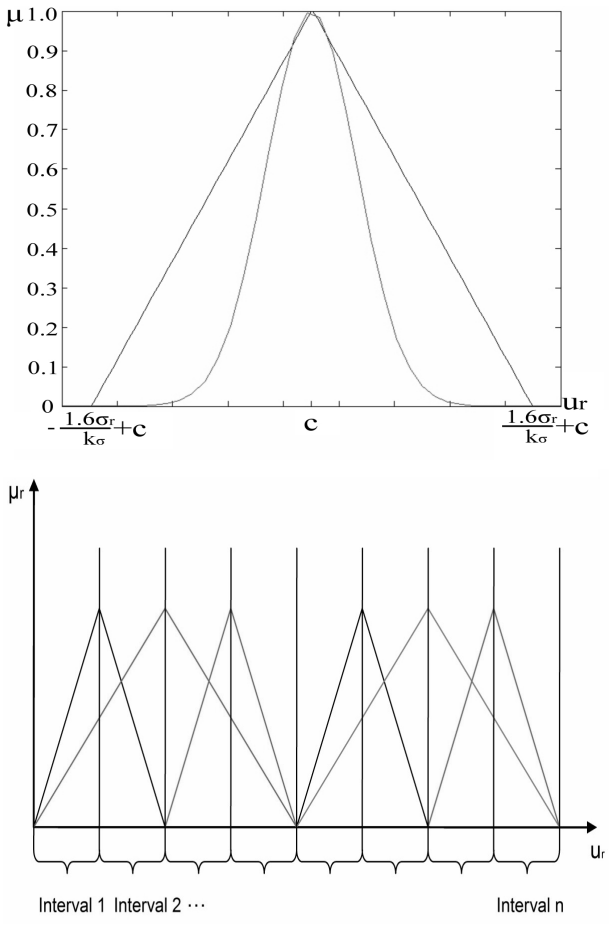

Figure 4: Top: Linear and nonlinear validity function. Bottom: Linear functions on the intervals.

Then, the input u_r is reconsidered in the final membership function and the following expression is obtained,

$$\mu_i(\underline{u}) = exp\left(-\frac{1}{2}\left(\frac{(u_r - c_{ir})^2}{\sigma_{ir}^2})\right)\right) + \mu_c. \qquad (6)$$

3. Create the linear membership function for the required input as from equation (4).

4. Partition the input space u_r.
The input space of u_r is partitioned into q search intervals, which are used in the later estimation of u_r. Every interval describes the validity of half of the local linear model. Thus, the input space of every LLM is divided into two intervals. This is necessary due to the structure of the new linear membership functions, because they consist of two linear functions as mentioned above. For every interval, a "left function" and a "right function" are considered (Fig. 3). For the sake of brevity, equation (4) is represented as the following:

$$\mu(u_r)_{i,r} = \begin{cases} \mu(u_r)_{i,r,1} \\ \mu(u_r)_{i,r,2}. \end{cases} \qquad (7)$$

5. In the following loop, consider every interval for a possible solution of u_r.

- Determine which of i membership functions $\mu(u_r)_{i,r}$ are valid for the currently considered interval, by checking every spline. $\mu(u_r)_{i,r,1}$ is valid if

$$\mu(interval\_left)_{i,r,1} \geq 0 \wedge \mu(interval\_right)_{i,r,1} \leq 1 \qquad (8)$$

$\mu(u_r)_{i,r,2}$ is valid if

$$\mu(interval\_left)_{i,r,2} \leq 1 \wedge \mu(interval\_right)_{i,r,2} \geq 0, \qquad (9)$$

where $\wedge$ indicates the "and" logical function.

- Use the valid membership spline functions to create validity functions for each local linear model. Taking the previously calculated part of the membership function i, $\mu_{i,calc}$ as in (6), and sum it with the valid linear spline membership function i $\mu(u_r)_{i,r,\{1,2\}}$, where $\mu(u_r)_{i,r,\{1,2\}}$ represents a valid spline membership function within the range of functions. This yields:

$$\mu(u_r)_i = \mu(u_r)_{i,r,\{1,2\}} + \mu_{i,calc} \text{ and}$$
$$\Phi(u_r)_i = \frac{\mu(u_r)_i}{\sum_{j=1}^{M} \mu(u_r)_j}.$$

If there are no valid linear membership functions for a local linear model, then the model will not be considered for further actions.

- Initially create the output function for every local linear model by multiplying its validity function with the local linear model function:

$$\hat{y}(u_r)_{LLM,i} = \hat{y}(u_r)_i \cdot \Phi(u_r)_i.$$

Next, sum the output functions to create the model output:

$$\hat{y}(u_r) = \sum_{i=1}^{M} \hat{y}(u_r)_{LLM,i}.$$

Finally, equate the model output function to the desired model output value, and solve the resulting equation with respect to the variable u_r:

$$y = \hat{y}(u_r).$$

- Verify that the calculated u_r is inside the currently considered interval.

$$interval\_left \leq u_r \leq interval\_right.$$

If so, accept it as one possible solution. If not, disregard it.

Due to the structure of the validity functions, an inversion of the model is possible only within the input space of the required variable. Beyond these borders, the model input will drift to zero, which is comparable to the normal LOLIMOT behavior. That is, once a *work nominal point* is chosen, the inversion is possible within the input domain. The worst case is if the working nominal point is close to the border of the domain. If so, a more suitable division of the input space is needed.

3 Toolbox Description

3.1 Setup the Local Linear Model Tree Structure

To apply the inversion tool to a LOLIMOT it's necessary to transform the model into the following data structure as depicted in fig. 5 on page 7. The structures should be named "Model" so it is possible to use them also along with the Simulink block set. The structure "Networkname" consists of neurons which are named Networkname(1) Networkname(M). Each neuron includes the parameters for its membership function Φ and the local linear model g. The membership function consists of the central and standard deviation parameters for every dimension of the input space. The LLM parameter vector g owns the weight values of the local liner model $[w_0 \quad w_1 \quad ... \quad w_n]$.

3.2 Usage of the Matlab Command Line Inversion Tool

The usage of the command line inversion tool is intuitive. To call the inversion function you have to type "inver" on the Matlab command line. Before executing the function you have to submit some parameters.

`Matlab Command:`
$[possible\_solutions] =$
$inver(Lolimot, u\_seek, u\_input, y\_output, k\_sigma)$

Input Parameters:

- *Lolimot* - Local linear model tree structure
- *u_seek* - Number of the desired input e.g. $1 = u_1$
- *u_input* - Model input data as row vector $[u_1 \quad u_2 \quad ... \quad u_n]$, where the desired input should be set to 0, but it is a "not caring" value, e.g. $[u_1 \quad u_2(required) \quad u_3]$, i.e. [12.4 0 34.2]
- *y_output* - Value of the desired model output, e.g. y_output = 123.456
- *k_sigma* - The k_sigma used at model creation, e.g. k_sigma = 0.33 refer LOLIMOT introduction.

Output Parameters:

- *possible_solutions* - Row vector of possible values for the desired input$[sol._1 \quad sol._2 \quad ... \quad sol._N]$ e.g. $[1, 4, 5]$.

3.3 An Application

The considered system is a throttle valve shown in Fig. 6, the parts of the internal combustion engine are depicted. Particularly difficult is the model of the air path which consists of an air filter, a throttle valve, a collecting tank and an intake valve. A complete model with chemical reaction is not considered. The command

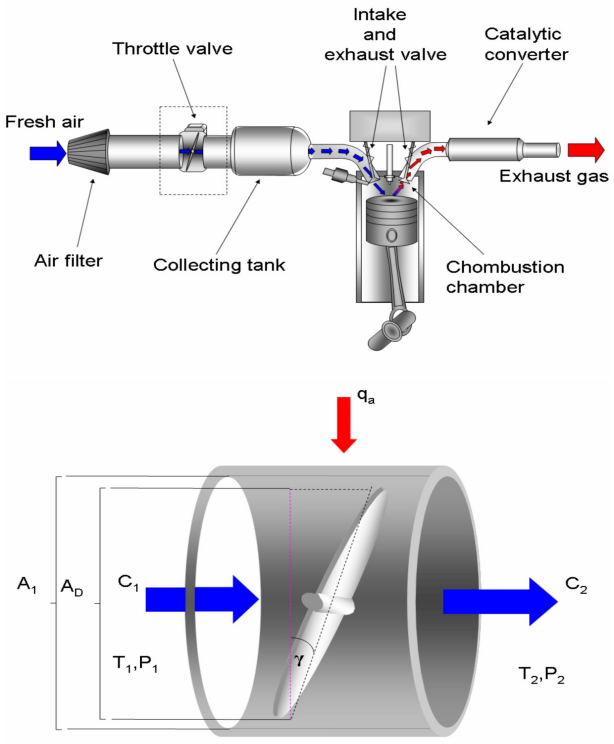

Figure 6: Top: Overview. Bottom: Schematic Structure of Throttle Valve

line function will be applied to a manifold air mass flow model with a size of 3 neurons, which depends on four inputs, where u_1 represents the throttle angle in degrees, u_2 is the ambient air pressure in mbar, u_3 represents the manifold air pressure in mbar and u_4 the ambient temperature in Celsius, see the lower part of Fig. 6. The model output is manifold air mass flow in kg/h. The

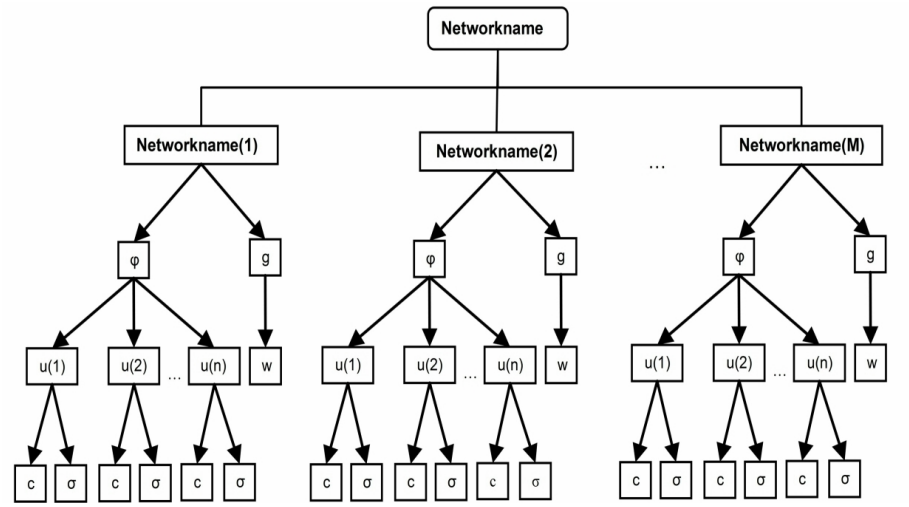

Figure 5: Local Linear Model Structure

model was trained with a k_sigma equal to 0.33.
With the following data set, all four required input values for the requested model output are estimated. They are depending on the remaining three inputs and the model output.
$u = [40.32 \quad 1012.5 \quad 977.3 \quad 21.21]$;
$y = 147.18$.
The known but omitted input value for the required input will be used for the validation of the estimated value.

Estimation of Input u_1. First the estimation of throttle angle u_1 is carried out. The input parameters are: the model structure named "Model", the required input $u_1 = 1$, the measured input data, the desired model output value and the k_sigma which was used for model training. As Output value, u_1 is selected.

`Matlab Command:`
$u_1 = inver(Model, 1, [0 \quad 1012.5 \quad 977.3 \quad 21.21], 147.18, 0.33)$
$u_1 = 39.1160$

The estimated throttle angle is 39.116° the reference angle is 40.32°, which results in a difference of 3.08%.

Estimation of Input u_2**.** The u_2 input represents the ambient air pressure. The input parameters are those of the Model again but the required input is set to $u_2 = 2$ and adapted to the input data. All other parameters remain the same.

`Matlab Command:`
$u_2 = inver(Model, 2, [40.32 \quad 0 \quad 977.3 \quad 21.21], 147.18, 0.33)$
$u_2 = 1.0e+003 * [1.116 \quad 1.0116]$

The inversion tool estimated two possible solutions for the ambient pressure of 1011.6 mbar or 1166 mbar. The real value is 1015.5 mbar. Due to the difference of the results 1011.6 mbar is assumed to be the right result. In this case the difference is 0.39 %.

Estimation of Input u_3**.** For the estimation of the manifold air pressure u_3 the required input must be changed and the input vector is set to [40.32 1012.5 0 21.21] the other values must not be changed.

`Matlab Command:`
$u_3 = inver(Model, 3, [40.32 \quad 1012.5 \quad 0 \quad 21.21], 147.18, 0.33)$
$u_3 = 982.9629$

The reference pressure for the requested model output is 977.3 mbar, as a result an air pressure of 982.96 mbar is obtained. The difference is 0.58%.

Estimation of Input u_4**.** At last the ambient temperature u_4 must be estimated. Now the required input must be the 4^{th} and the parameter vector must be adapted.

`Matlab Command:`
$u_4 = inver(Model, 4, [40.32 \quad 1012.5 \quad 977.3 \quad 0], 147.18, 0.33)$
$u_4 = 21.2064$

The function estimated an ambient temperature of 21.2064°C. The reference temperature is 21.21°C. In this case the error seems to be caused by the rounding error. Therefore the inversion tool matches the real input value with no difference.

As demonstrated, the command line tool offers a strong possibility to inverse local linear model trees with a high accuracy and less time effort. If a more suitable input value is calculated, an automatic selection criterion for the possible solution is useful. This can be introduced by taking the previous process input value $u_r(k-1)$ and calculating the difference with the estimated values, if the process is not an *impulsive process*. The value with the smallest difference should be the right one.

3.4 Usage of the Simulink Inversion Block

The developed inversion block (*inver*) for Matlab Simulink implements the features of the inverse local linear model trees into a Simulink Model. This is very important for the application in a control loop.
The necessary settings are similar to the command line tool settings. The Matlab workspace variable of the LOLIMOT model has to be named "Model".
In the picture below, the Simulink block is depicted. It consists of three inputs: u_prev, u_input and y_desired and one output u_crtl. The u_prev input is used for the $k-1$ value of the required input which is taken for the selection of the valid required value, if more than one solution is possible. The multiplexed model input data is used for u_input. The required input should be left open as at the command line tool in Matlab, hence it's a "not caring" input. For input y_desired should be provided the desired value for the model/process output. Finally, after the inversion of the model, block output u_crtl is the required (desired) input value for the desired output at the output port of the block.
Similar to the command line version the user must specify in the dialog box depicted in Fig. 7 the required input u_seek which has to be the number of the inputs and k_sigma which ware used at the training of the local linear model tree. In Fig. 8 a possible application of the

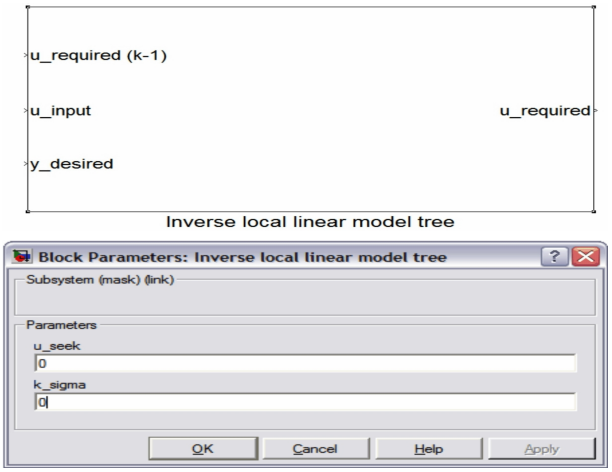

Figure 7: Top: Simulink block. Bottom: Dialog Simulink structure

toolbox is shown. In the scheme a feedback controller can also be considered. A sinus function is tried to tracked.

4 Conclusions and Future Work

This paper deals with local inversion of the Local Linear Model Trees. An analytical solution is presented and implemented in Matlab. Further the theoretical structures of the neural network and also a useful Matlab tool for any researcher who utilizes LOLIMOT are explained. The importance of the paper in the control area consists of the possibility to use the LOLIMOT through its inversion as a forward part of a controller. The inversion of a throttle valve feed forward model is demonstrated and very good results are achieved. Future projects involve the testing of the feed forward control in a feedback control structure.

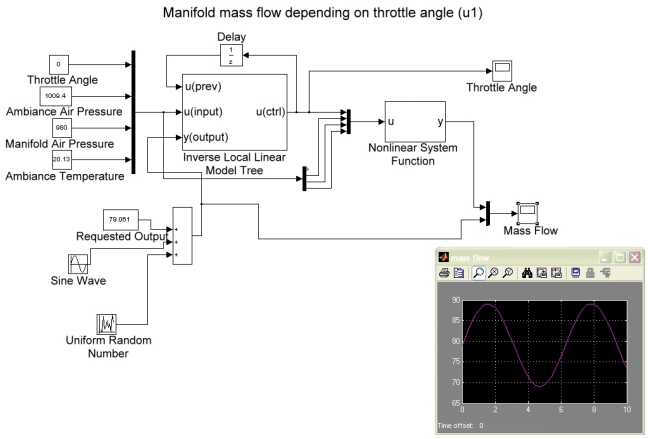

Figure 8: Simulink Structure of a Feed Forward Control Scheme Based on the Inversion of a Neural Network Model

References

[1] A. Toepfer A. Fink. Technical report - on the inversion of nonlinear models. 2003.

[2] R. Isermann A. Fink, S. Toepfer. Neuro and neuro-fuzzy identification for model-based control. *IFAC Workshop on Advanced Fuzzy/Neural Control, Valencia, Spain*, Volume Q:111–116, 2001.

[3] R. Isermann A. Fink, S. Toepfer. Nonlinear model-based control with local linear neuro-fuzzy models. *Archive of Applied Mechanics*, Volume 72(11-12):911–922, 2003.

[4] A. Fink and O. Nelles. Nonlinear internal model control based on local linear neural networks. *IEEE Systems, Man, and Cybernetics, Tucson, USA*, 2001.

[5] P. Mercorelli. An optimal minimum phase approximating pd regulator for robust control of a throttle plate. *45th IEEE Conference on Decision and Control (CDC2006), San Diego (USA), 13th-15th December*, 2006.

[6] O. Nelles. *Nonlinear System Identification with Local Linear Neuro-Fuzzy Models*. Shaker Verlag, 1999.

[7] R. Isermann O. Nelles, A. Fink. Local linear model trees (lolimot) toolbox for nonlinear system identification. *12th IFAC Symposium on System Identification (SYSID), Santa Barbara, USA*, 2000.