1 Introduction

Due to the increasing demand for environmental compatibility of vehicles, automotive control systems have to cope with the high degree of non-linearity of combustion engines. Since generic design rules for non-linear control systems are still not available, each system must be designed individually - with enormous engineering efforts involved. Learning approaches, which extract a non-linear process model or a control strategy from measurement data, seem to be an alternative. When on-line learning is performed, these systems are even capable to adapt themselves to varying process characteristics (aging, wear and tear).

Artificial neural networks are known to be universal approximators and provide the capability of learning. Any continuous function can be approximated arbitrarily well by means of a neural network, provided that the number of units (neurons) is sufficiently large. This theoretical result is opposed to the complexity of practically useful networks. With respect to limited computational power and storage capacity in automotive systems, the implementation of neural networks often is impossible.

In this paper, a network architecture for real-time function approximation that can be implemented on low-cost standard hardware is introduced. This approach is based on mathematical methods such as triangulations and scattered data interpolation. The network consists of \( N \) arbitrarily located interpolation nodes. The distribution of these nodes is adjusted to the information content of the modelled function. Subsets of \( n + 1 \) nodes are connected so that they define simplices in the \( n \)-dimensional input space (triangles in the two-dimensional case, polyhedra in higher dimensions). The network output is computed by linear interpolation of attributes (estimated values of the modelled function) which are assigned to the nodes. For any input vector only \( n + 1 \) out of \( N \) nodes are active. Thus, a purely local behaviour is achieved. Section 2 introduces this approach in some detail and illustrates its computational efficiency. The network transfer function is found to be non-linear with respect to the positions of the interpolation nodes and linear with respect to their attributes. These properties are considered when learning algorithms for the proposed network architecture are derived (section 3). Network creation from measurement data (off-line training) combines heuristics for node insertion with linear optimization of node attributes. On-line training of node attributes can be performed by means of least mean square (LMS) algorithms, for which stability and robustness criteria are well known.

As an example of a possible application of the proposed networks in automotive systems, steady-state engine torque modelling is investigated in section 4. Finally, section 5 summarizes the properties (and also restrictions) of these networks and gives an outlook on further research work.

2 Delaunay Networks

The main feature of the proposed network is their capability to use arbitrarily distributed interpolation nodes. It is this flexibility that leads to small problem-dependent and well-generalizing networks with moderate memory consumption.

Each of the \( N \) interpolation nodes consists of a position \( \mathbf{z}_i \) in \( \mathbb{R}^n \) and an attribute \( \mathbf{y}_i \) which is an estimate for the modelled function at the position \( \mathbf{z}_i \). Since the set of nodes do not imply any topological information, the selection of nodes used for output computation (active nodes) at a given input vector \( \mathbf{z}_q \in I \subset \mathbb{R}^n \) (also referred to as query point) can either be based on proximity criteria \([1]\) or requires the a-priori definition of topological relations of the nodes. In this paper, the latter strategy is pursued: a tool from computational geometry, the Delaunay triangulation \([2]\), is used to define an appropriate network topology.
21 The Delaunay triangulation

The Delaunay triangulation of a set of nodes is a graph that connects subsets of \( n + 1 \) nodes. Each subset represents a simplex (triangle in \( \mathbb{R}^2 \), polyhedron in the general case). The convex hulls of these simplices define the areas within which the respective nodes (vertices of the simplex) are used for output computation (figure 1a).

---

**Figure 1:** a) Triangulation of nodes (filled dots) in \( \mathbb{R}^2 \). The depicted query point (marked by a cross) is contained in the convex hull of the shaded triangle and thus activates the nodes which are vertices of this triangle. b) The Delaunay definition.

The maximum number of \( n \)-dimensional simplices that can be constructed from a set of \( N \) nodes, is

\[
S_{\text{max}} = \binom{N}{n+1}.
\]

Those simplices that actually define the network structure, however, have to satisfy the following constraints:

- The union of all \( S \) simplices must cover the complete input space \( I: \bigcup_{i=1}^{S} T_i = I \subset \mathbb{R}^n \).
- The intersection of two simplices, \( T_i \) and \( T_j \), must be empty: \( T_i \cap T_j = \emptyset \), if \( i \neq j \).

The first condition is set up in order to avoid extrapolation, whereas the second constraint assures that a concrete query point \( \mathbf{x}_q \) always activates the same subset of nodes. Non-intersecting simplices are obtained if the set of nodes is in general position and simplices are chosen to comply with the following definition of Delaunay simplices:

**Definition 2.1** A Simplex \( T_k \) consisting of \( n + 1 \) nodes in \( \mathbb{R}^n \) is a Delaunay simplex if and only if its embedding \( n \)-dimensional hypersphere does not contain any other node of the network.

---

22 Selection of active nodes and output computation

The selection of the subset of \( n + 1 \) active nodes for a given query point \( \mathbf{x}_q \) is equivalent to finding the simplex the convex hull of which contains \( \mathbf{x}_q \). Barycentric coordinates \([6]\) prove to be an adequate means for this task.

In the following, \( \mathbf{z}_i, \quad i = 1, \ldots, n+1 \), denotes the position of the \( i \)-th node of the simplex \( T_k \).

By definition, the barycentric coordinates \( b_i(\mathbf{x}_q) \) of a vector \( \mathbf{x}_q \) in relation to a simplex \( T_k \) are the \( n+1 \) weights which move \( T_k \)'s centre of gravity into the point \( \mathbf{x}_q \). Thus, the following equations hold:

\[
\sum_{i=1}^{n+1} b_i(\mathbf{x}_q) \cdot \mathbf{z}_i = \mathbf{x}_q \quad (2)
\]
\[
\sum_{i=1}^{n+1} b_i(\mathbf{x}_q) = 1 \quad (3)
\]

The solution of these equations yields the \( n + 1 \) barycentric coordinates of the query point \( \mathbf{z}_q \) in relation to the simplex \( T_k \). The decision whether the convex hull of \( T_k \) contains \( \mathbf{x}_q \) can be taken on the basis of the signs of these coordinates:

**Given a query point** \( \mathbf{z}_q \in \mathbb{R}^n \) **and a simplex** \( T_k \). **The barycentric coordinates** \( b_i(\mathbf{z}_q) \) **of** \( \mathbf{z}_q \) **in relation to** \( T_k \) **are non-negative if and only if** \( \mathbf{z}_q \) **is contained in the convex hull of** \( T_k \) **[6]**.

An efficient search algorithm should account for the fact, that the temporal sequence of query points
\( \mathbf{x}_i(k) \) is a continuous function\(^1\). Hence, a judicious choice of the starting simplex for the search procedure is that simplex \( T_{\text{prev}} \) that was active in the previous step. At first, the barycentric coordinates of the current query point \( \mathbf{x}_i(k) \) in relation to \( T_{\text{prev}} \) are recomputed. In many cases, no negative barycentric coordinate will occur and the same subset of nodes can again be used for output interpolation. If, however, some of the barycentric coordinates are negative, it is still probable that the query point is located in the vicinity of \( T_{\text{prev}} \). Thus, the search is continued with the neighboring simplices of \( T_{\text{prev}} \) which are accessible from the Lawson data-structure (see above, section 2.1).\(^1\)

Once the subset of active nodes are determined, network output is computed by linear interpolation of the attributes \( \mathbf{y}_i \) assigned to the active nodes. Again, the barycentric coordinates can serve for this task. The network output \( \mathbf{y}(\mathbf{x}_i) \) is the weighted sum of the active nodes' attributes, with the (non-negative) barycentric coordinates of \( \mathbf{x}_i \) in relation to the active simplex chosen as weight coefficients:

\[
y(\mathbf{x}_i) = \sum_{i=1}^{n+1} b_i(\mathbf{x}_i) \cdot \mathbf{y}_i.
\]

(4)

3 Training of Delaunay Networks

The notion of training includes the automatic creation of Delaunay networks from measurement data (off-line learning) and the adaptation of the network in order to compensate for time-varying process behavior (on-line training). Basically, training algorithms can be subdivided in two categories:

1. algorithms which determine/modify the distribution of nodes
2. algorithms which determine/modify the attributes of nodes

To derive appropriate strategies for these two tasks, a mathematical description of the input/output characteristics of Delaunay networks is given first: The network response \( \mathbf{y}(\mathbf{x}_i) \) can be written as the inner product of a weight vector \( \mathbf{w}(\mathbf{x}_i) \) and the vector of attributes \( \mathbf{y} \).

\[
y(\mathbf{x}_i) = \mathbf{w}^T(\mathbf{x}_i) \cdot \mathbf{y}.
\]

(5)

Those elements of \( \mathbf{w} \) that correspond to the active nodes for the given query point \( \mathbf{x}_i \) are equal to the respective barycentric coordinates, while the remaining \( N - n + 1 \) elements are zero. The vector \( \mathbf{y} \) comprises the attributes of all \( N \) nodes.

\(^1\)The query point comprises state and/or control variables of the controlled (or modelled) plant and will therefore not change too rapidly in real-world systems.

Equation (5) shows that the network response depends non-linearly on the positions of nodes. The selection of \( n + 1 \) out of \( N \) nodes and the computation of the respective barycentric coordinates are non-linear operations. Consequently, the creation or modification of the node distribution is a problem of non-linear optimization. Therefore, heuristic strategies are rather tractable than a rigorous mathematical treatment.

However, the network output is linear with respect to the attributes \( \mathbf{y} \) which can thus be determined or modified by well-understood and computationally feasible linear optimization.

Off-line training strategies must include both the heuristic optimization of node positions and the computation of optimal node attributes. On-line training, on the other hand, needs to manipulate the node distribution necessarily but can be restricted to the attributes. If the degree of time-variance of the plant under consideration is not too high, the a-priori distribution will still be adequate. It is then sufficient to modify the attributes only.

3.1 Off-line training

Given a set of \( M \) datapoints \( (\mathbf{x}_j, \mathbf{y}_j^d) \), \( j = 1, \ldots, M \), off-line training means the task of automatic creation of an appropriate Delaunay network. The algorithm described in the following is a constructive strategy, i.e. starting with a small network, nodes are inserted iteratively.

The position of the inserted node is heuristically chosen to be the location of that datapoint \( (\mathbf{x}_j, \mathbf{y}_j^d) \), for which the current network produces the maximum absolute error:

\[
e(j) = \| \mathbf{y}_j^d - \mathbf{y}(\mathbf{x}_j) \| \quad l = \arg \max \{ e(j) \}, \quad j = 1, \ldots, M
\]

(6)

After each insertion of an additional node, the attributes of all nodes currently in the network are optimized. Since, in general, the number of datapoints is significantly higher than the number of nodes \( (M \gg N) \), an over-determined set of equations is defined:

\[
\begin{bmatrix}
\mathbf{y}_1^d \\
\mathbf{y}_2^d \\
\vdots \\
\mathbf{y}_M^d
\end{bmatrix}
= 
\begin{bmatrix}
\mathbf{w}^T(\mathbf{x}_1) \\
\mathbf{w}^T(\mathbf{x}_2) \\
\vdots \\
\mathbf{w}^T(\mathbf{x}_M)
\end{bmatrix}
\begin{bmatrix}
\mathbf{y}_1 \\
\mathbf{y}_2 \\
\vdots \\
\mathbf{y}_N
\end{bmatrix}
\]

(7)

This system can be solved in the least squares sense by pseudo-inversion of the matrix \( \mathbf{W} \):

\[
\mathbf{y}_{\text{opt}} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \mathbf{y}^d
\]

(8)
Node insertion by means of (6) and attribute optimization according to (8) are repeated iteratively until a user-defined criterion, such as a maximum model complexity (total number of nodes) or a minimum model accuracy, is achieved.

### 3.2 On-line training

The attribute vector \( \hat{\mathbf{z}} \) given by equation (8) minimizes the sum of squared errors over the entire dataset:

\[
J = \sum_{j=1}^{M} (y\hat{j} - y(z_j))^2
\]

On-line or instantaneous learning, on the other hand, means that information about the desired input/output behaviour is given only by isolated samples \((z_k, y\hat{k})\). The current squared output error is then used to estimate the cost function \(J\):

\[
\hat{J}(k) = (y\hat{k} - y(z_k))^2, \quad k = 1, 2, \ldots
\]

A gradient descent rule can be used to adapt the attribute vector \(\hat{y}\). Attributes are moved in the direction of the negative gradient of the estimated cost function \(J\):

\[
\hat{y}(k + 1) = \hat{y}(k) - \alpha \cdot \nabla \hat{J}(k)
\]

The parameter \(\alpha\) (the learning rate) determines the size of the attribute movements and thus is crucial for the stability and robustness of the algorithm. In the context of neural networks, this gradient descent method is known as the delta rule, whereas in the field of system identification and control, the term least mean square algorithm (LMS) is preferred. A normalized form (NLMS) of this algorithm leads to the following adaptation rule for the attribute vector of a Delaunay network (see appendix):

\[
\begin{align*}
\hat{y}(k + 1) &= \hat{y}(k) + \beta \cdot \frac{u(z_k) \cdot e(k)}{w(z_k) \cdot w(z_k)} \\
e(k) &= y\hat{k} - y(z_k)
\end{align*}
\]

This adaptation rule has the following properties:

- Due to the linearity with respect to the adapted attributes, the cost function is unimodal. The algorithm will thus converge to the global minimum if the learning rate \(\beta\) is chosen properly, namely if \(0 < \beta < 2\).

- If \(\beta\) is chosen to be 1.0, the error at the current sample point \(z_k\) is eliminated in one adaptation step (one shot learning). However, in real-world applications only a noisy version of the desired response \(y\hat{k}\) is available. Choosing a learning rate \(\beta < 1\) then leads to higher robustness.

### 4 Results

#### 4.1 Off-line training

As an example of non-linear function approximation in sophisticated automotive control systems, we investigate the modeling of torque characteristics of a combustion engine. A dataset of 105 datapoints, each of which comprises of throttle valve position \(x_1\), engine speed \(x_2\) and the load provided in steady state \(y\), were used to create a Delaunay network by means of the off-line training algorithm introduced in section 3.1. The model accuracy was prescribed in terms of an average absolute error of 4%. This criterion was met after a total number of 21 nodes had been inserted.

For systematic investigations the actual engine torque characteristic is now replaced by a test function which is assumed to describe torque characteristics of a wide class of engines [8].

\[
y(x_1, x_2) = 1000 \cdot \left[1 + e^{(2 - \frac{x_1}{5} + \frac{x_2}{5})}\right]^{-1}
\]

With the same number of nodes \((N = 21)\) that were used in the actual engine torque model, another Delaunay network that approximates (13) was created. Figure 2 depicts the surface that represents this network's input/output behaviour as well as the distribution of nodes in the two-dimensional input space. Furthermore, the topological relations, i.e., the edges of the Delaunay triangulation, are shown. This illustration reveals the fact that the proposed heuristic strategy for node insertion actually concentrates nodes in areas of high information content. In those regions of the input space, where large second derivatives of the surface occur, a high node density is found, whereas only few nodes are located in flat areas.

The capability of higher-dimensional modelling was then analyzed by extending the test function (13) to a third input signal \(x_3\):

\[
y(x_1, x_2, x_3) = 1000 \cdot \left[1 + e^{(2 - \frac{x_1}{5} + \frac{x_2}{5} + \frac{x_3}{5})}\right]^{-1}
\]

Off-line training of an appropriate Delaunay network was again performed until the average absolute error kept within 4%. \(N = 63\) nodes were required in this case.

\[\text{Note that the weight vector } w(z_k), \text{ which contains zeros expect for the active nodes' weights, appears in the nominator of the modification term in (12).}\]

\[\text{3 All signals are scaled to the interval } [0, 1000].\]
4.2 On-line training

The efficiency of the NLMS algorithm for on-line attribute training was investigated in the following simulations: The attributes of all nodes of the automatically created networks were set to the value 500, which is the average of the normalized signal range. The positions of nodes remained unchanged. Thus, the initial characteristic represented by the network was a plane of constant height 500. The attributes were then trained on-line using signals (throttle valve position and engine speed) acquired in a test vehicle as inputs to the network (query points) and the corresponding values of the test function (13) as desired network responses. The desired response was superposed with Gaussian noise ($\sigma^2_{noise} = 40$).

Figure 2: Illustration of an automatically created Delaunay network with two-dimensional input space. (Approximation of the generalized engine torque model given by (13).)

Figure 3: Results of on-line training by means of the NLMS algorithm.

Figure 3 shows the error signal during on-line training. The simulation comprises of three epochs, each of which consists of approximately 25,000 samples. No training is performed in the first epoch ($\beta = 0.0$). At the beginning of the second epoch (arrow in figure 3) the learning rate $\beta$ is set to 0.8. The diagram illustrates the fast convergence of the NLMS algorithm; within the first 3000 samples after training is enabled, the highest peaks of the error signal are reduced to 50% of the initial values. In the third epoch (samples 50,000 ... 75000) only minor changes of the attributes are still performed. The root mean square value of the error signal in the third epoch is very close to that of the disturbance signal:

$$\text{RMS}(\text{noise}) = \sqrt{40} \approx 6.32, \text{RMS}(\epsilon) = 7.74.$$  

<table>
<thead>
<tr>
<th>Model</th>
<th>n</th>
<th>N</th>
<th>mem. size</th>
<th>$T_{avg}$</th>
<th>$T_{max}$</th>
<th>adapt. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine torque</td>
<td>2</td>
<td>21</td>
<td>188</td>
<td>18</td>
<td>199</td>
<td>64</td>
</tr>
<tr>
<td>eq. (13)</td>
<td>2</td>
<td>21</td>
<td>196</td>
<td>18</td>
<td>237</td>
<td>64</td>
</tr>
<tr>
<td>eq. (14)</td>
<td>3</td>
<td>63</td>
<td>2243</td>
<td>36</td>
<td>706</td>
<td>85</td>
</tr>
</tbody>
</table>

Table:  
Table 1 summarizes the results. The memory expense includes the nodes (positions and attributes) as well as the triangulation (Lawson's datastructure). The response time of a Delaunay network depends on the velocity of the query point in input space. In steady-state operation, the search procedure described in section 2.2 succeeds after a few steps. Abrupt changes of one input signal (the throttle valve position in the simulation discussed here), however, lead to a larger number of steps (computations of barycentric coordinates in relation to two or more simplices) and thus to the maximum response times $T_{max}$. These abrupt changes occur very seldomly, so that the average response times $T_{avg}$ are considerably lower. In the simulations described above, 99% of all queries were answered in less than 70 usec and

---

4 The sampling time of typical automotive applications is 20 msec, one epoch thus takes 8.3 min.
5 The TS85 processor runs at 30 MHz. Automotive control units are likely to be equipped with such devices in the next years.
5 Conclusion

The proposed Delaunay networks are based on conventional mathematical principles. They are especially suited for real-time applications with limited computer power, e.g., for automotive control systems. Due to the linearity with respect to the node attributes, well-understood linear optimization methods are applicable. An off-line training strategy that combines linear attribute optimization with heuristics for node insertion was developed. On-line training of node attributes can be performed by means of the NLMS algorithm which is known for its robustness and simple stability criteria.

Delaunay networks are, on the other hand, restricted to low-dimensional input spaces. From a mathematical point of view, the definition of Delaunay simplices and that of the barycentric coordinates are valid in any dimension $n$. However, the computational effort, as well as the memory consumption grow drastically with the input space dimension. (Compare two- and three-dimensional examples in table 1.) Furthermore, problems of numerical exactness, i.e., round-off errors when hyperspheres and barycentric coordinates are computed, get more serious in higher-dimensional spaces. Our software implementation is thus tailored for $n = 2, 3$. A four-dimensional version is currently under development.

Further research work on Delaunay networks will cover two areas: Firstly, the extension of on-line training strategies is investigated. Apart from attribute adaptation by means of the NLMS algorithm, on-line insertion and deletion of nodes shall increase the network's ability to adapt to varying characteristics. Secondly, the utilization of Delaunay networks for modelling of dynamical non-linear processes is studied.

Appendix

LMS algorithm for adaptation of node attributes

The gradient of the cost function $\mathcal{J}$ is given by the formula

$$\nabla \mathcal{J}(k) = \frac{\partial \epsilon^2(k)}{\partial \mathbf{w}} = -2 \cdot \mathbf{w}(z_k) \cdot \epsilon(k).$$

The gradient descent rule (11) thus can be written:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\alpha \cdot \mathbf{w}(z_k) \cdot \epsilon(k).$$

(16)

If each adaptation step shall result in a constant amount of error-reduction, the learning rate $\alpha$ must be chosen as a function of the current weight vector. The a-posteriori error at the query point $z_k$ is given by

$$\epsilon(k)|_{\mathbf{w}(k+1)} = y_k^d - \mathbf{w}^T(z_k) \cdot \mathbf{w}(k+1).$$

(17)

If a reduction factor of $(1 - \beta)$ is prescribed, we obtain

$$\epsilon(k)|_{\mathbf{w}(k+1)} = (1 - \beta) \cdot \epsilon(k)|_{\mathbf{w}(k)} \quad y_k^d - \mathbf{w}^T(z_k) \cdot \mathbf{w}(k+1) = (1 - \beta) \cdot \epsilon(k)|_{\mathbf{w}(k)} \quad \mathbf{w}^T(z_k) \cdot 2\alpha \cdot \mathbf{w}(z_k) = \beta \quad \alpha = \frac{\beta}{2 \cdot \mathbf{w}^T(z_k) \cdot \mathbf{w}(z_k)}.$$ (18)

Substituting (18) into (16) finally yields the normalized least squares algorithm given by equation (12).

Acknowledgement

This work is sponsored by the German National Science Foundation (DFG) under grant 75/22-1. Furthermore, the author is grateful to Dr. T. Roos, Federal Institute of Technology at Zurich, for the helpful discussions on Delaunay triangulations. Simulation data used in this paper were provided by R. Bosch GmbH.

References


