Employing Interpolation to Enable Higher Order Fuzzy Logic Controllers on Resource Constrained Embedded Devices

Christian Wagner
The Computational Intelligence Centre
Department of Computing and Electronic Systems
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ
United Kingdom
chwagn @ essex.ac.uk

Hani Hagras
The Computational Intelligence Centre
Department of Computing and Electronic Systems
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ
United Kingdom
hani @ essex.ac.uk

Abstract

It has been shown recently that higher order Fuzzy Logic Controllers (FLCs) such as type-2 FLCs have the potential to outperform type-1 FLCs. However, the high computational cost associated with such higher order FLCs is still impeding their widespread use in real world applications where the computational and memory resources are limited. In this paper we will present a simple framework that allows to tie together the advantages of using higher order FLCs while keeping computational costs at a minimum. The proposed technique is based on the offline computation of high-quality control surfaces which are sparsely sampled and reproduced through interpolation on the actual embedded device. The proposed technique will be described in detail and we will present some example applications and results achieved using the proposed system.

1 Introduction

The last 30 years have witnessed the wide deployment of Fuzzy Logic Controllers (FLCs) in a vast number of real world applications. The motivation for this is that the FLC is credited with being an adequate methodology for designing robust systems that are able to deliver good performance in the face of uncertainty and imprecision ubiquitous in real world applications. In addition, FLCs facilitate the construction of control algorithms in a user-friendly way closer to human thinking and perception.

The vast majority of fuzzy systems research and applications employ type-1 FLCs. However, type-1 FLCs cannot fully handle or accommodate the high levels of linguistic and numerical uncertainties associated with dynamic real world applications and environments as type-1 FLCs use the crisp and precise type-1 fuzzy sets. However, advances in technology and fuzzy logic theory have made it possible to research and apply more complex, i.e. higher order forms of fuzzy logic such as type-2 fuzzy logic. It has been shown that interval type-2 fuzzy systems can provide superior performance when compared to type-1 fuzzy systems with the same amount of rules as a result of the additional degrees of freedom provided by only the footprint of uncertainty [1], [2].

A major drawback of such higher order FLCs has nevertheless been the significant increase of computational complexity which has considerably narrowed the possibility for real world application of such controllers.

Recently, a series of advances such as [3],[4], [5], [6] have been made in trying to reduce the computational complexity associated with the currently most complex used form of FLCs – general type-2 FLCs, in order to facilitate and enable their practical usage. In [4] a novel form of representing general type-2 fuzzy sets using geometric concepts such as polygons and poly-lines was presented. In [5], [6] new forms of representation were suggested based on the concepts of zSlices and α-cuts. Both representations aim towards devising new representations which enable the implementation of general type-2 FLCs while relying on the existing theory of interval type-2 FLCs. As such, the complexity of general type-2 fuzzy logic should become manageable and its application will be greatly facilitated.

The work presented in this paper aims to enable the utilization of the benefits provided by higher order FLCs while minimizing the computational requirements. In particular we are trying to enable the application of higher order FLCs on the small, resource-constrained embedded systems present in the vast majority of real world applications. This will enable such real world applications to benefit from the various advantages and control performance
improvements gained by the application of higher order FLCs.

In this paper, we will be presenting our framework which allows using higher order FLCs offline to construct a multidimensional control surface which is then sampled and stored in the form of a lookup table on the actual system. We will showcase several techniques to interpolate values which are not included in the lookup table and present an initial evaluation of the trade-offs that are involved in selecting the number of samples for the lookup table and the interpolation technique.

Section 2 gives a high-level overview of the framework while Section 3 details the experimental setup. The experimental results are discussed in section 4, followed by the conclusions in section 5.

2 An Overview of the Proposed Framework

The proposed framework can be divided into a set of steps or processes as follows:

1. Generation of a control surface offline using a high-order FLC using a specified sampling (discretization) frequency along all the input axis.
2. Selection of an interpolation technique.
3. Transfer of the discretized control surface and interpolation component to the embedded system.
4. Generation of the output (control) values on the embedded system during run-time using the lookup table and interpolation.

All of these steps are described in detail below.

2.1 Generation of a Control Surface.

The proposed framework can in fact be employed with any type of controller and is not restricted to FLCs. For the ease of visualization, we will present examples of multiple input, single output controllers. However, of course multiple output FLCs are possible as any multiple output FLC can be regarded as a collection of several single output FLCs.

A control surface is generated by sampling the inputs of the controller at a specified frequency. The choice of this frequency i.e. the number of samples per axis or input dimension is chosen according to two criteria:

- The amount of storage space available on the embedded device. The space required $S$ (in bits) for the lookup table which contains all possible combinations of inputs with their corresponding output can roughly be determined by the following formula:

$$S \cong (D_1 \ast D_2 \ast \ldots \ast D_N) \ast (N + 1) \ast B \ (1)$$

where $N$ represents the number of dimensions or inputs of the controller, $D_1, D_2, \ldots, D_n$ represent the number of discretizations of the different dimensions and $B$ represents the number of bits required per value in the lookup table. The “+1” accounts for the storage of the output values for each combination of inputs. It should be noted that we have used the $\cong$ and not the $=$ sign as the choice of file format and operating system characteristics will have subtle effects on the actual file size.

- The control quality required. As the number of samples in the lookup table increases and correspondingly the distance between points decreases, the quality of interpolated output values will increase.

2.2 Selection of an interpolation technique.

The problem of interpolating multi-dimensional data has been intensively researched and a countless variety of algorithms with varying interpolation-accuracy vs. computational cost ratios have been proposed some of which are described in [7], [8]. In this paper we will focus on two common interpolation techniques, specifically linear interpolation (which is an example of fast-but-low-accuracy interpolation technique) and spline interpolation (which is an example of slow-but-high accuracy interpolation technique). Any other interpolation methods can be used in accordance with the system requirements and interpolation methods characteristics which mainly include:

- Interpolation speed (controller sampling speed) required.
- Amount of accurate samples available (refer to Section 2.1).
- Dimensionality of input data, i.e. not all interpolation methods are necessarily easily scalable to many dimensions.

2.3 Transfer of Discretized Control Surface and Interpolation Component to the Embedded System.

This step involves copying the created discretized control surface along with the interpolation method (software) to the embedded system in order for the system to function as a controller. With the constraints set out in Sections 2.1 and 2.2 above, this step can be considered straightforward.
2.4 Execution of Interpolation Method in Conjunction with the Provided Control Surface.

The final step encompasses feeding the controller inputs to the interpolation method which then utilizes the provided discretized control surface to compute an output (control) value for the system. As such, the interpolation method effectively replaces the initial controller when deployed onto the embedded system.

3 Experimental setup

3.1 The Sample Controller.

Due to the lack of space, in this paper we will focus only on the results related to the control surface of an interval type-2 FLC implementing a robot edge following behaviour. The controller uses two inputs (sonar sensors measuring the distance to a wall) and one output (steering direction). Further details on the actual controller can be found in [9]. We have chosen a two-input, one-output controller as it greatly facilitates the visualization but as previously mentioned the framework can be employed with any number of inputs and outputs.

3.2 The interpolation methods.

As mentioned above, we have focused on two interpolation strategies which are the linear interpolation strategy (which is an example of a fast-but-low-accuracy interpolation technique) and the spline interpolation (which is an example of a slow-but-high accuracy interpolation technique).

However, any interpolation technique could be used according to the requirements of any given application. It should be mentioned that we are not advocating any of the interpolation strategies as the best one as all interpolation methods have specific strengths and weaknesses. As such our selection of algorithms should not be considered as a complete overview of the available methods.

The following subsections will briefly introduce both the linear and the spline interpolation methods.

3.2.1 Linear interpolation.

Linear interpolation is a very simple method of interpolation based on linear polynomials. While linear interpolation generally refers to the interpolation of uni-dimensional data, extensions such as bilinear and trilinear interpolation have been devised for higher dimensional data.

Applying basic linear interpolation between two points \(A \) and \(B\) at coordinates \((x_A, y_A)\) and \((x_B, y_B)\) gives the coordinates of all the points forming a straight line between \(A\) and \(B\).

In this paper we are using bilinear interpolation which is a direct extension of linear interpolation for functions with two variables. It is achieved by computing the linear interpolant first in one dimension and then in the second dimension.

Information on linear interpolation in general and bilinear interpolation in particular is widely available in the literature, for example in [8],[10].

3.2.2 Spline interpolation.

Spline interpolation is an interpolation technique relying on an interpolant referred to as a “spline”. A “spline” is a piecewise function defined by polynomials. It has repeatedly been shown to provide a high level of interpolation accuracy and “smoothness”.

The natural cubic spline used in this paper is constructed by piecewise third-order polynomials. It is referred to as “natural” as the second derivative of each polynomial is set to zero at the endpoints.

Describing the exact details of spline interpolation in full detail is beyond the scope of this paper but detailed information on spline interpolation in general and BiCubic natural spline interpolation which is used in this paper is widely available and can for example be found here [7],[10].

3.3 Experiment details.

In order to test the validity of the approach, the existing interval type-2 FLC mentioned above was used to generate three control surfaces at different discretization levels as a benchmark. The coarsest discretization level of 21 steps across both axis was utilized as a basis for the interpolation algorithm, i.e. it was used as data for the lookup table. Applying Equation (1), while assuming 64 bits per value, which is the standard for the datatype double, results in the following size \(S\) for the lookup table:

\[
S \equiv (21 \times 21) \times (2 + 1) \times 64 \equiv 84672b \equiv 10kB
\]

Both the linear and the spline interpolation methods were subsequently used to generate control surfaces at the same discretization levels as the interval type-2 FLC to allow for a direct comparison (Fig. 1).

All the control surfaces were plotted (Fig. 2, 3 and 4) and a similarity measure based on the comparisons shown schematically in Fig. 1 was calculated (Table 1). Additionally, the performance in terms of average speed of each controller was determined (Table 2).

It should be noted that due to the nature of linear interpolation equations not all values can be calculated strictly according to the formulas as division by 0 would occur (for example at the edges of the data where no neighbouring points exist). The
respective points were eliminated before performing the numerical comparison for the BiLinear interpolation.

![Diagram showing interpolation methods]

**Fig. 1** Schematic view of the generated control surfaces and comparisons. Note that ONLY the 21 Discretization steps version from the FLC was used as input for the interpolation techniques.

### 4 Experimental Results

The experimental results indicate the differences in performance of both interpolation methods.

At 21 discretization steps both methods perform equally well and provide the same results as the interval FLC which is expected: as no interpolation is taking place the actual values stored in the lookup table are returned (Fig. 2a, 2b, 2c).

As the discretization levels increase one can even graphically detect that the BiCubic Spline interpolation (Fig. 2c, 3c and 4c) matches the corresponding FLC control surface (Fig. 2a, 3a and 4a) more accurately than the BiLinear interpolation (Fig. 2b, 3b and 4b).

This intuitive result is confirmed with the numeric calculation of the Root Mean Squared Error of both interpolation methods in respect to the interval type-2 FLC (see Table 1). Nevertheless, the computational cost which is reflected in the average controller iteration time shown in Table 2 indicates that the BiCubic Spline interpolation method is in fact not useful in reducing computational cost when compared against a simple interval type-2 FLC. The Spline interpolation method requires more than 5 times the amount of time required to run the actual FLC. This interpolation could still be useful to approximate future, more complex controllers such as general type-2 FLCs. These controllers will have a much higher iteration time for the same controller and as such Spline interpolation might become attractive.

BiLinear interpolation on the other hand, while performing less well on accuracy as shown in Table 1, nearly outperforms the interval type-2 FLC by four to one in terms of computational time required (see Table 2).

![Graphs showing control surfaces]

**Fig. 2** (a) shows the control surface as generated directly by the interval type-2 FLC using 21 discretization steps; (b) and (c) display the control surface generated when applying BiLinear interpolation and BiCubic spline interpolation respectively to the lookup table generated using 21 discretization steps of the interval type-2 FLC.
Fig. 3 (a) shows the control surface as generated directly by the interval type-2 FLC using 41 discretization steps; (b) and (c) display the control surface generated when applying BiLinear interpolation and BiCubic spline interpolation respectively to the lookup table generated using 21 discretization steps of the interval type-2 FLC.

Fig. 4 (a) shows the control surface as generated directly by the interval type-2 FLC using 101 discretization steps; (b) and (c) display the control surface generated when applying BiLinear interpolation and BiCubic spline interpolation respectively to the lookup table generated using 21 discretization steps of the interval type-2 FLC.
The results above reinforce the need to choose the interpolation algorithm according to the accuracy and execution speed requirements which are application dependent.

Table 1 Root Mean Square Errors (RMSEs) of interpolation methods when compared to interval type-2 FLC at same discretization level.

<table>
<thead>
<tr>
<th>Number of discretization steps</th>
<th>BiLinear Interpolation RMSE</th>
<th>BiCubic Spline Interpolation RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>0.788635515</td>
<td>0.434172168</td>
</tr>
<tr>
<td>101</td>
<td>0.814480134</td>
<td>0.448689506</td>
</tr>
</tbody>
</table>

Table 2 Average time taken per controller iteration. (Each controller was executed over 1000 control cycles and the average execution time was taken.)

<table>
<thead>
<tr>
<th>Interval Type 2 FLC</th>
<th>BiLinear Interpolation</th>
<th>BiCubic Spline Interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.5μs</td>
<td>5.5μs</td>
<td>106μs</td>
</tr>
</tbody>
</table>

5 Conclusions

In this paper we have shown that it is possible to represent the information obtained from complex controllers such as higher order FLCs in the form of a lookup table which can then, in combination with interpolation techniques, be used as an independent controller.

The method allows small, resource constrained devices such as embedded systems to benefit from the advantages provided by complex control algorithms while reducing the computational cost.

It is clear that there is a direct tradeoff between control surface reproduction accuracy (i.e. control performance) and computational cost and we leave it to the interested reader to determine the best combination of sampling rate and interpolation algorithm for his specific application.

In the future we are looking into refining and expanding the framework while applying it to various embedded systems with a focus on the fields of intelligent agents and robotics.

We are particularly interested in investigating the suitability of the framework in exploiting the advantages in control performance provided by the application of general type-2 FLCs. Furthermore we are currently working on other tools aiming to facilitate the implementation and application of higher order FLCs.

The software tools to implement the methods presented in this paper are freely available for non-commercial purposes at www.christian.wagnerweb.net as long as the author is acknowledged.

Acknowledgements

We would like to thankfully acknowledge Michael Thomas Flanagans support in granting us access to his Java software implementing the CubicSpline algorithm(s). It is freely available for non-commercial use at http://www.ee.ucl.ac.uk/~mflanaga/java/ provided that the author is acknowledged.

This work was kindly supported by the Département Recherche scientifique et Recherche appliquée, part of the Ministère de la Culture, de l'Enseignement supérieur et de la Recherche, Luxembourg.

References