A Fuzzy Framework with Modeling Language for Type 1 and Type 2 Application Development

Kevin McCarty, Member IEEE, Milos Manic, Sr. Member IEEE, Allan Gagnon [1]

Abstract— **Fuzzy logic, Type-1 and Type-2, are well suited for human systems interactions because they provides a natural way of implementing linguistic terms from human experts. Existing fuzzy frameworks, however, provide limited support for Type-2. They also tend to be fairly complicated and/or have limited portability. This paper introduces a fuzzy framework for building a Type-1 or Type-2 fuzzy controller. A "wizard" application and modeling language are supported to provide an easy-to-use interface for creating a fuzzy inference system. The benefits of this framework are: 1) Increased understanding of fuzzy systems implementation via easy-to-use visual tools; 2) Reduced development time; 3) A standardized and portable codebase; 4) Easy configuration via XML; 5) Support for both Type-1 and Type-2 fuzzy sets and rules. The framework is tested and solves a maze problem using both Type-1 and Type-2 implementations.**

I. Introduction

Complex systems, such as robots, frequently incorporate a multitude of diverse inputs in order to perform a given task. Each input may or may not influence a system's behavior depending upon other inputs and/or system states. For example, a robot that senses a wall nearby will behave differently depending upon its speed, direction of motion, goals, other walls, etc. Due to the potential wide variety of possible conditions and responses, fuzzy logic, in the form of a Fuzzy Inference System (FIS) provides an excellent implementation option. The FIS provides the foundation for Type-1 Fuzzy Logic Controllers (T1-FLCs) or Type-2 Fuzzy Logic Controllers (T2-FLCs), used to control a complex system and is desirable for the following reasons:

1. Fuzzy logic describes both input and output behaviors in human understandable terms.
2. Because real-world factors such as sensor noise, actuator variations and environmental factors add elements of uncertainly to the interpretation, T1-FLCs and T2-FLCs are often the most practical approaches available to describe and deal with the uncertainty.
3. Fuzzy logic often provides a much simpler way to describe and approximate complex behaviors than polynomial functions.

Based upon the concept of Fuzzy Logic introduced by Zadeh [1], T1-FLCs operate on the principal that inputs and outputs can be "fuzzified"; that is, defined not as a specific number or boundary, but instead as a range of values defining varying degrees of membership between 0 and 1 [2]. Type-1 fuzzy logic does have its limitations, such as managing input noise and outliers. Zadeh recognized this and introduced Type-2 fuzzy logic [3, 4] in response.

Type-2 fuzzy logic, while more computationally complex than its Type-1 cousin, has certain advantages in that Type-2 sets employ membership degrees that are themselves fuzzy sets, allowing for an additional degree of computational freedom for modeling uncertainties [4]. T2-FLCs have advantages over T1-FLCs in handling noise and data outliers and are shown to outperform T1-FLCs under certain situations [5]. While T1-FLCs have been in use in numerous applications for over 4 decades, use of T2-FLCs is still relatively rare. However, interest in Type-2 fuzzy logic is growing among engineers and computer scientists, particularly as the growing computational power of recent computers makes Type-2 Fuzzy Logic Controllers (T2 FLCs) more cost-effective [5].

Clearly, practical applications of T1-FLCs and T2-FLCs are everywhere. Despite this, there has been only limited progress in using FLCs for mainstream, commercial applications. Programmers often have little to no appreciation for, or even understanding of, fuzzy techniques and how they might be applied to solve common problems.

One way to address this problem is through the use of a software framework designed to support creation and maintenance of an FIS - a fuzzy framework. Software engineering methods rely on the use of frameworks to improve the quality of software while also reducing complexity and development costs [6-8]. Software frameworks, usually in the form of software class libraries, DLLs or other reusable software, allow developers to focus on solving a specific problem rather than spending time writing generic routines [7]. Fuzzy frameworks do exist and are available as commercial offerings from sources such as Matlab and LabView. These offerings provide very sophisticated functionality as add-on toolkits to their core product line. Open source frameworks such as AForge.Net [9], Sourceforge.net [10] and CodeProject [11] provide another option for developers interested in creating their own custom solutions. Despite the availability of both commercial and non-commercial frameworks, limitations of both continue to inhibit more widespread use and adoption of FLCs in general and T2-FLCs in particular. Among these limitations is the difficulty for novices in trying to understanding the various fuzzy objects and how they relate and how to implementing a working FIS in code [12]. Applications wizards can help with this, along with a suitable modeling language for configuring FLCs dynamically [13].

This paper introduces a novel framework extending open source work done for a Type-1 fuzzy library [9]. This framework is designed to provide an open source alternative to other solutions while addressing some of the existing limitations of both commercial and noncommercial options.

The framework is then used to build a Type-1 based FIS for a T1-FLC and a Type-2 based FIS for a T2-FLC. Each FLC is implemented in order to solve a sample navigation problem.

This paper is organized as follows: Section II presents the problem statement. Section III describes the framework in

[1] The authors are with the Department of Computer Science, University of Idaho at Idaho Falls, Idaho Falls, ID 83402. Email: kmccarty@ieee.org, misko@ieee.org, allan.gagnon@nettryx.net

detail and how to use it to implement a working FIS. Section IV applies the framework to solve a simple navigation problem using a T1 FLC and T2 FLC. Section V presents conclusions and discusses future work.

II. Problem Statement

Commercial products, such as Matlab, have limited portability to other languages. Their frameworks require purchase of the core product, often at an extra cost of thousands of dollars, and may involve a steep learning curve for use. Furthermore, portability issues make adding a Matlab Fuzzy Toolkit FIS to a generic web application based upon a language such as Visual Basic.NET a significant undertaking. Open source solutions, on the other hand, provide few, if any, useful tools for configuration and implementation. None of these tools mentioned has yet to provide support for fuzzy Type-2 as part of their framework. Support for Type-2 fuzzy logic in a portable framework does exist [14] but neither source code nor binaries are available for evaluation.

Hence, anyone attempting to use fuzzy logic for applications, training, or just for understanding faces a number of obstacles regardless of whether they opt for a commercial or open source option. In summary, each of the available frameworks mentioned above suffers from one of more of the following limitations: 1) High Cost; 2) High degree of difficulty for implementation; 3) Limited portability; 4) Limited availability; 5) Limited fuzzy type-2 support.

A further difficulty limits the usefulness of many existing frameworks – the inability to easily configure, save and maintain framework configurations. A framework that can't be customized to solve a specific problem is of no use at all while one that requires a substantial learning curve is likely to remain unused outside of highly specialized circles in engineering and academia [5].

Finally, the development and use of a XML-based, Fuzzy Systems Modeling Language (FSXML) would greatly enhance the usability of any fuzzy framework [13]. Any language could be used to develop a Wizard application to serve as a configuration tool and dynamically implement a FIS on the fly.

The framework proposed in this paper attempts to resolve each of these issues. It addresses (1) by providing an open source solution, extending the original open source, type-1 library from [9]. It addresses (2) by providing a Wizard application in C#.NET that provides a visual interface to step a user through the process of creating all of the necessary fuzzy objects for a fully functional FIS. For (3), the framework is a .NET assembly/dll usable by any language capable of calling .NET assemblies. It is written in C#, a variant of C which has wide adoption in the computer science community. The University of Idaho will make the framework, Wizard and samples available as complete source projects on its website addressing (4) and providing educators with simple visual tools as aids for a basic course on fuzzy logic. Finally the framework extends the original open source, type-1 library to support type-2 fuzzy objects and operations with minimal changes to the original design, satisfying issue (5).

Addressing the final point and taking the requirements one step further, the framework implements an XML-based fuzzy modeling language, built and maintained by the Wizard, to allow for an almost code-free implementation of the framework for use by an external application.

III. Framework Architecture

Object-oriented techniques have long been recognized as a way to reduce the complexity of software [7]. The framework consists of a number of objects in support of an FIS, most notably: 1) Membership Functions; 2) Fuzzy Sets; 3) Fuzzy Variables; 4) Fuzzy Rules; 5) Rules Database; 6) Defuzzifier. The following pseudo code the 8 steps necessary to implement the framework in order to create a functioning FIS.

```
function CreateFIS() returns FuzzyInferenceSystem BEGIN
  MembershipFunctions[] mf
  FuzzySets[] fs
  FuzzyVariables[] fv
  FuzzyRules[] fr
  Defuzzifier d
  FuzzyDatabase fd
  FuzzyInferenceSystem fis
  // step 1
  DefineMembershipFunctions(mf)
  // step 2
  DefineFuzzySets(fs)
  // step 3
  DefineFuzzyVariables(fv)
  // step 4
  AssignVariableInputsOutpus(fv)
  // step 5
  DefineFuzzyRules(fr)
  // step 6
  DefineDefuzzifier(d)
  // step 7
  AssignRulesToDB(fr, fd)
  // step 8
  DefineFIS(fd, fis)

  return fis
END
```

Implementing steps 1-8 can be done strictly in code as calls to the framework, but it is not a trivial process. Human systems interactions benefit when the interaction is as simple as possible. To help a user define a fuzzy inference system, with its corresponding fuzzy objects, the framework implements a Fuzzy Modeling language. Previous work [13] demonstrates how useful an XML-based modeling language is in supporting dynamic configuration of FISs and fuzzy objects in general. However, XML is difficult to create by hand so the framework provides the Wizard utility to allow a user to generate and maintain the underlying XML-based model. Wizards of various types provide a mechanism for enhanced human systems interactions with the underlying software and are quite effective at reducing the time required to understand and implement software at both low and high levels.

A traditional "wizard" application, presents a predefined sequence of steps as pages in a program window. With the Wizard, each page presents a user interface that allows a user to "step" through each necessary configuration element in the proper sequence. It is also a way to break down a complex

overall process into a series of simpler steps. As a result, the Wizard application can serve both as a developer's tool and a training tool for the novice learning to use fuzzy logic. Currently, the Wizard application only supports fuzzy Type-1 configuration, although Type-2 support is in development stage. The Wizard application, is shown in fig 1.

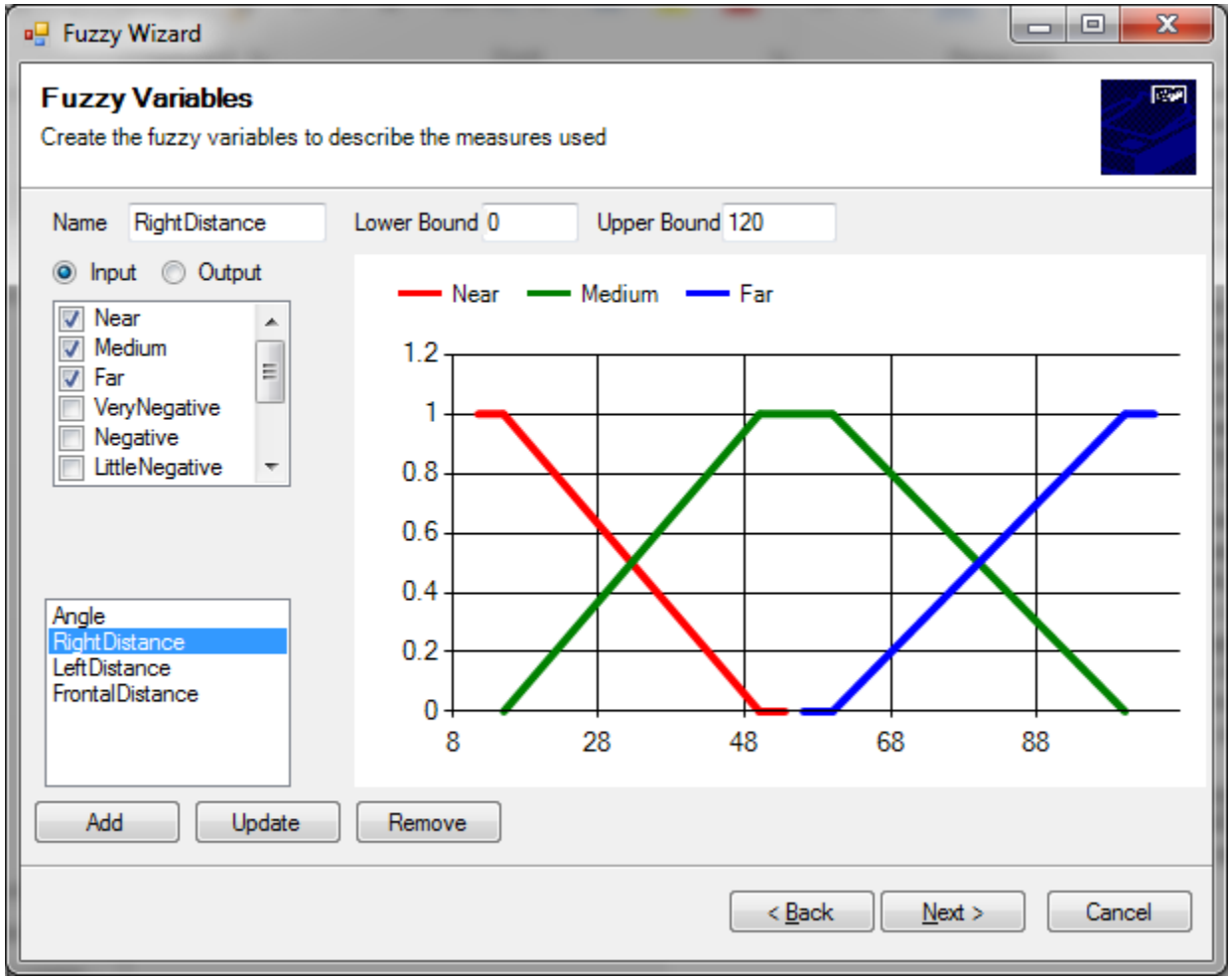

Fig. 1. Fuzzy Wizard – Variables Page

Step 1 implementation requires specifying a convex shape for a membership function μ. μ is a function over a domain *D* such that for each x in *D*, μ(x) is a number between 0 and 1.

$$\mu(x) \rightarrow [0,1], x \in D \qquad (1)$$

Zadeh's rules for fuzzy sets require the functions take the form of a convex shape such as a triangle, trapezoid, Gaussian or other convex curve. The resulting shape serves as the membership function μ. In the framework, each shape, f_{shape} is defined with one or more boundary points P_{bk} and one or more apex points P_{ak}.

$$\mu = f_{shape}(P_{b1}..P_{bn}, P_{a1}..P_{an}) \qquad (2)$$

For each P, the X value specifies a specific value x in the domain while the Y value specifies the corresponding membership value between 0 and 1 at that value, μ(x).

$$f_{shape}(P_{b1},..,P_{bn},P_{a1},..P_{an}) = \mu(x), x \epsilon D \qquad (3)$$

The modeling language, produced by the Wizard provides textual representations of the Fuzzy Function and all other fuzzy objects. The Wizard application handles step 1 as a Membership Functions page. Using the Wizard, the developer specifies the type of fuzzy set and each of the corresponding boundary and apex points. The Wizard interface is demonstrated in fig 2. The resulting FSXML is shown in fig 3.

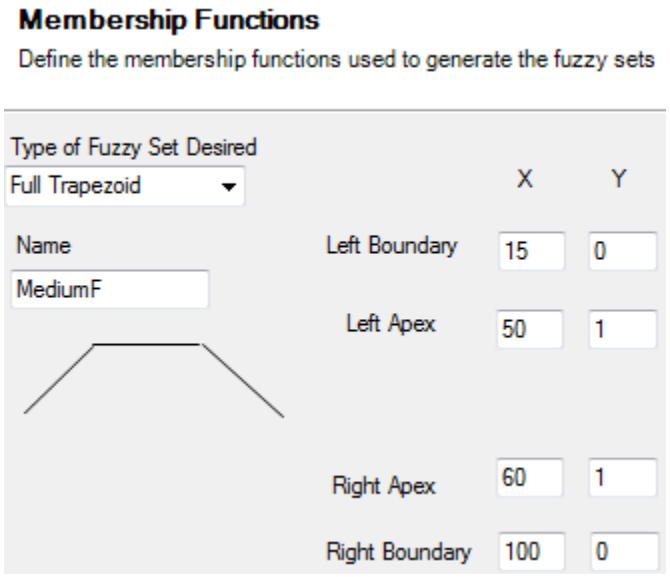

Fig. 2. Defining a Membership Function using the Wizard

```
<MembershipFunction Name="MediumF" FuzzySetType="Full Trapezoid">
  - <Points>
      <Point Id="1" X="15" Y="0"/>
      <Point Id="2" X="50" Y="1"/>
      <Point Id="3" X="60" Y="1"/>
      <Point Id="4" X="100" Y="0"/>
    </Points>
</MembershipFunction>
```

Fig. 3. A Fuzzy Function Definition in XML

In the framework, a fuzzy set f_s is defined by a membership function and a linguistic "term" which describes its purpose in a more easily understandable way. Giving the fuzzy set a "linguistic term" allows use of more intuitive language when describing subsequent rules in the FIS. For instance, a fuzzy set that determines *tallness* might be called "Tall". Later fuzzy rules will reference that particular fuzzy set with the word Tall. This actually adds a certain level of precision to the fuzzy set that is very difficult to emulate using crisp sets or numbers and is one of the major advantages to using fuzzy logic [1, 2].

Step 2 simply requires applying the function defined in step 1 to a linguistic term to generate a fuzzy set f_s.

$$f_s = \mu + \text{linguistic term} \quad (4)$$

The Wizard application provides a Fuzzy Sets page to allow a user to do this procedure. The resulting XML is shown in fig 4. The interface demonstrated in fig 5.

```
<FuzzySets>
    <FuzzySet Name="Near" MembershipFunction="NearF"/>
    <FuzzySet Name="Medium" MembershipFunction="MediumF"/>
    <FuzzySet Name="Far" MembershipFunction="FarF"/>
```

Fig. 4. Combining a Term and Fuzzy Function to Create a Fuzzy Set

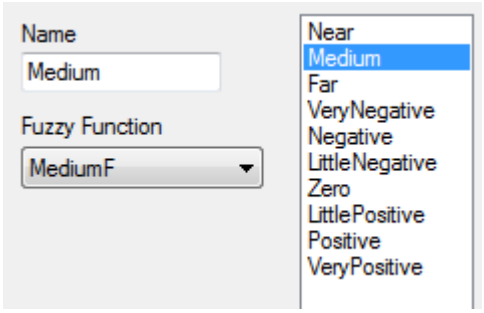

Fig. 5. Using the Wizard to create a Fuzzy Set

In step 3, users create one or more fuzzy variables. Each Fuzzy Variable must have a "name", a linguistic term to appropriately describe the variable's purpose or function. Variable names should make sense within the FIS, describing in easily understood terms what that variable represents. Next, the user defines the boundary of the domain of a particular variable. The domain should be large enough to encompass all of the fuzzy sets to be assigned. Finally, the user designates whether the variable represents input (the antecedent) or output (consequent) when used to build fuzzy rules.

In step 4, once the domain is defined and the variable "termed", the variable then is associated with one or more of the fuzzy sets defined in step 2. These associations are used to determine which fuzzy sets are represented in the fuzzifcation/defuzzification process. The corresponding XML is shown in fig. 6 and the Wizard Fuzzy Variables step detailing this process is demonstrated in fig 7.

During the fuzzification process, the fuzzy sets assigned to the fuzzy variable are evaluated using the underlying fuzzy function assigned to the corresponding fuzzy set

```
<FuzzyVariables>
  - <FuzzyVariable Name="Angle" UpperBound="50" LowerBound="-50" Type="Output">
      <FuzzySet Name="VeryNegative"/>
      <FuzzySet Name="Negative"/>
      <FuzzySet Name="LittleNegative"/>
      <FuzzySet Name="Zero"/>
      <FuzzySet Name="LittlePositive"/>
      <FuzzySet Name="Positive"/>
      <FuzzySet Name="VeryPositive"/>
  </FuzzyVariable>
```

Fig. 6. Defining a Fuzzy Variable in XML

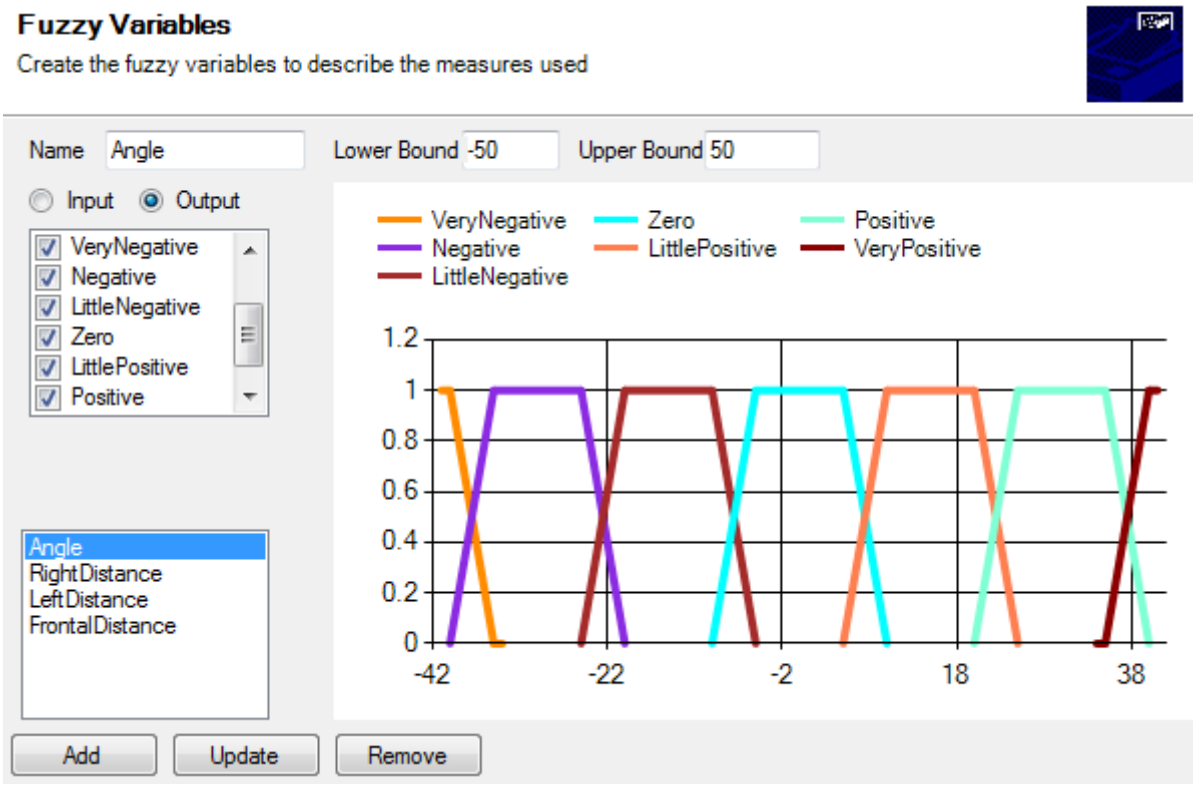

Fig. 7. Defining a Fuzzy Variable using the Wizard

Step 5 specifies the fuzzy rules that constitute the FIS. Each fuzzy rule consists of an antecedent which is a statement of the form:

IF <fuzzy variable> IS <fuzzy variable or fuzzy set>

The antecedent specifies a testable condition similar to a crisp IF statement, except instead of a true or false result, the fuzzy result consists of a value between 0 and 1 inclusive, dependent upon the input value and the various membership or fuzzy functions underlying the corresponding fuzzy sets shown in Eq.3. Antecedents can be combined using AND/OR and parenthetical operators. For example, to test the distance of a barrier to the front of an obstacle the antecedent might take the form of:

IF FrontalDistance IS Far

"FrontalDistance" can consist of multiple fuzzy sets, for example: "Near", "Medium" and "Far". The antecedent "IF FrontalDistance IS Far" looks at the membership function of the fuzzy set "Far" assigned to the fuzzy variable "FrontalDistance". Note again the use of linguistic terms that are easily understandable even to laypersons. A parser within the framework turns the text into its corresponding fuzzy sets and fuzzy variables.

The fuzzy rule also requires a consequent, which is constructed similarly to the antecedent but uses output variables and sets. The Wizard application contains a simple text builder a user can employ to construct the both the antecedent and consequent.

In step 6, the user defines the number of fuzzy intervals used for defuzzification. The framework currently supports traditional Zadeh rules for fuzzification of fuzzy Type-1 where membership of a fuzzy variable is equal to the minimum membership of the corresponding fuzzy sets. This is also referred to as a fuzzy intersection of fuzzy sets.

$$\cap \mu_{C_i} = \min(\mu_{C_1}, \mu_{C_2}, .., \mu_{C_n}) \quad (5)$$

Defuzzication is achieved by then taking a fuzzy union across all intervals in the domain. This is accomplished by taking the maximum across the sets of intervals and their corresponding memberships.

$$\cup \mu_{C_i} = \max(\mu_{C_1}, \mu_{C_2}, .., \mu_{C_n}) \quad (6)$$

A centroid, of center of gravity is then calculated by determining a weighted mean across the fuzzy region [2]. The fuzzy solution region A is calculated by the following:

$$\Re = \frac{\sum_{i=0}^{n} d_i \mu_A(d_i)}{\sum_{i=0}^{n} \mu_A(d_i)} \quad (7)$$

where d is the i^{th} domain value and $\mu(d)$ is the membership value returned the corresponding fuzzy function defined in step 1. Fuzzy Type-2 defuzzification uses the Karnik-Mendel Interval Technique described in [15] which is a variation on the

centroid technique involving a calculation of multiple centroids over the footprint of uncertainty.

In step 7, the user creates a database of the fuzzy rules defined to use in the FLC. Depending upon which rules are input and which are output, the FIS will attempt to evaluate all relevant rules during the fuzzification/defuzzification process.

Step 8 is performed by the framework. The resulting FIS then consists of the database repository of all the relevant fuzzy objects as well the domain space and fuzzy operators used. The XML defining the final FIS is listed in fig 8

```
<FuzzyInferenceSystem Name="MySystem" OutputToDB="false">
    <FuzzyDB Name="FuzzyDB"/>
    <Defuzzifier Name="Defuzzifier" Intervals="1000"/>
  - <Rules>
        <FuzzyRule Name="Rule 1"/>
        <FuzzyRule Name="Rule 2"/>
        <FuzzyRule Name="Rule 3"/>
        <FuzzyRule Name="Rule 4"/>
        <FuzzyRule Name="Rule 5"/>
        <FuzzyRule Name="Rule 6"/>
        <FuzzyRule Name="Rule 7"/>
    </Rules>
    <SQLCn Name="SQLCn"/>
</FuzzyInferenceSystem>
```

Fig. 8. Defining an FIS Using the Fuzzy Modeling Language

Note that the reference to SQL is currently not supported but detailed in future work. Pseudocode for constructing the FIS from the model language is as follows:

```
Procedure Init(XMLFile filename) returns FIS
  FIS fis = GetInfModel(filename)
  FuzzyDatabase database
  FuzzyVariable[] fuzzyVariables = GetVariables(filename)
  For Each variable in fuzzyVariables
    FuzzySets[] fuzzySets = GetSets(variable, filename)
    For Each set in fuzzySets
      FuzzyFunction ff = GetFunction(set, filename)
      set.Function = ff
      variable.Add(set)
    Next
    database.Add(variable)
  Next
  fis.Add(database)

  fuzzyRules rules = GetRules(filename)
  For Each rule in rules
    IF fis.ruleUsed = rule THEN
      fis.AddRule(rule)
    END IF
  Next

  return fis
END
```

IV. Test Examples

An application validates both the configuration and underlying framework. Consider a simple software robot pictured in the maze in fig 9. In order to navigate successfully, the robot must adjust its angle of motion the appropriate amount at the appropriate time under the appropriate conditions. Fuzzy logic has been demonstrated to be very effective for navigation [16] and as such is widely used for directing robots, hence, robot navigation serves as a very useful test of the framework.

Suppose the task is simply to be able to navigate through the maze without encountering any walls. To meet its goal the robot needs to perform the following functions:

1. Measure the distance to each of the possible barriers (front, right, left).
2. Determine a direction that will allow the robot to continue forward motion without hitting any of the barriers of the maze.

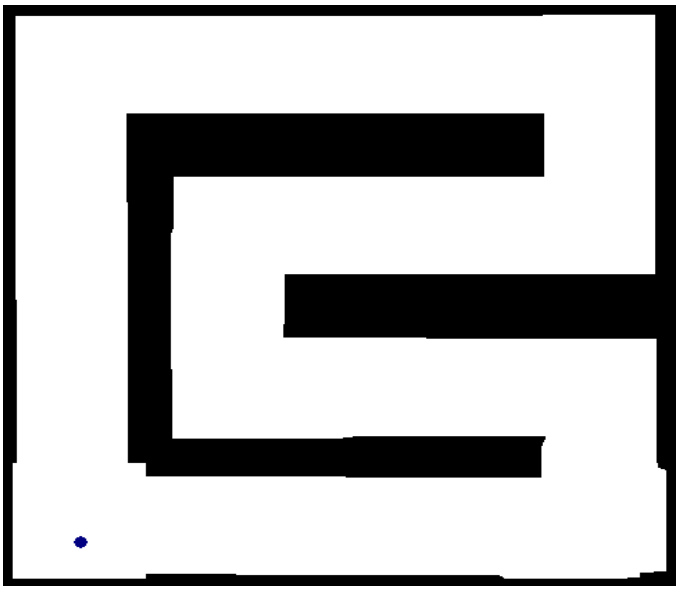

Fig. 9. Robot in a Maze

Ideally, the robot should try to position itself as far away from each barrier as possible while maintaining forward motion. In order to do this, it must constantly reevaluate its position as it moves around the maze and adjust its angle of motion in order to maintain maximum distance from each wall. Thus, the following factors have to be accounted for:

1. Frontal Distance
2. Distance to the Right Wall
3. Distance to the Left Wall
4. Angle of forward movement

The first 3 factors constitute the inputs and the last one is the resulting output. Both inputs and outputs are necessary for the fuzzy inference system (FIS).

Example1: Robot uses T1 FLC to navigate the maze.

A routine to turn the completed XML into a working T1 FLC was incorporated into the framework and used to help a software robot navigate the maze described in the problem statement. Pseudo code to describe the final process is as follows

```
Procedure StartNavigation
        ConfigureFIS(XmlFile)
        While(true)
                Navigate()
End

Procedure Navigate
```

```
        FIS.Input(GetFrontDistance())
        FIS.Input(GetLeftDistance())
        FIS.Input(GetRightDistance())
        FIS.Fuzzify(Inputs)
        FIS.Defuzzify(OutputAngleDelta)
        Robot.Angle = Robot.Angle + OutputAngleDelta
        MoveRobot()
End
```

Using the T1 FLC, the robot was able to successfully navigate the maze without hitting any barriers or halting its forward progress. Its path was tracked and shown in fig 10.

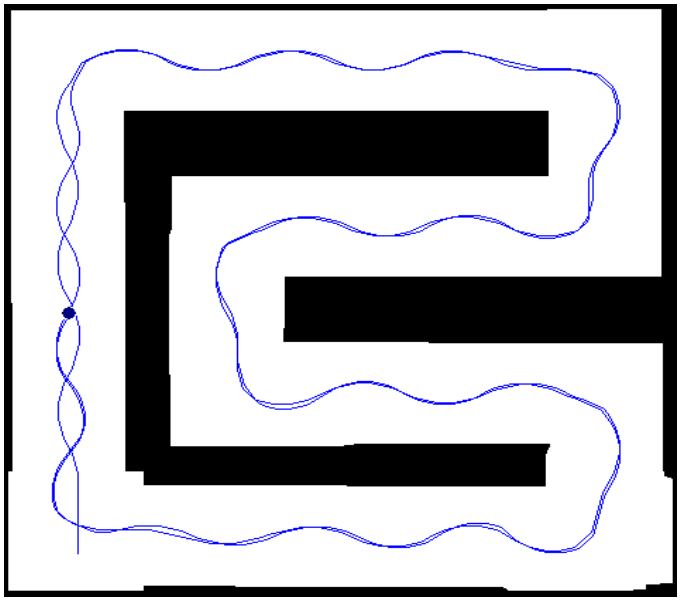

Fig. 10. Robot Navigating Maze Using Framework-Based T1 FLC

Example 2: Robot uses T2 FLC to navigate the maze.

In the second case, the framework was used to create a Type-2 Fuzzy Logic Controller. Without the use of XML, this requires simply coding the various fuzzy objects and FIS. With the Type-2 based FIS, the software robot T2-FLC was able to successfully navigate the same maze. Its path was tracked and shown in fig 11.

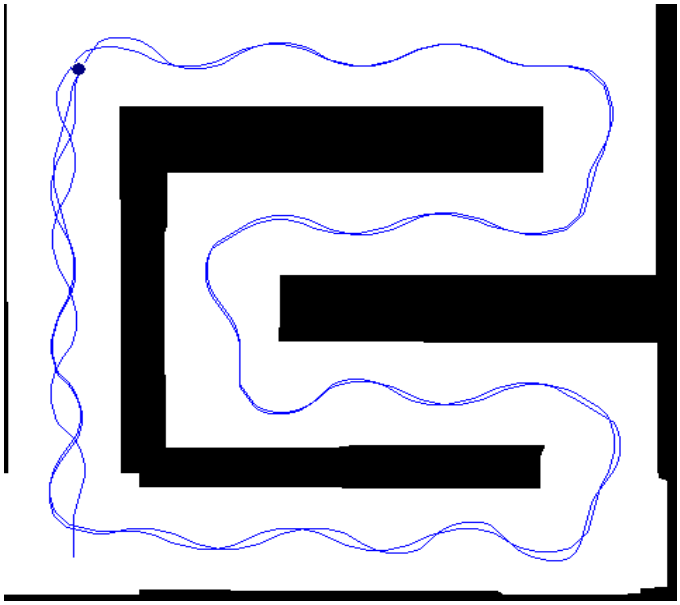

Fig. 11. Robot Navigating Maze Using Framework-Based T2 FLC

V. Conclusion and Future Work

The fuzzy framework is a necessary first step in the creation of an easy-to-use, flexible, standalone utility able to create usable, sophisticated T1-FLCs and T2-FLCs. A simple software robot, relying upon the framework, is able to cleanly and correctly navigate a maze under both a Type-1 and Type-2 controller.

The fuzzy Wizard application generated a standard XML file used to successfully configure the T1-FLC for the robot on-the-fly without additional code. The Wizard allowed for complete configuration of the T1-FLC to solve the maze problem in a few minutes and dynamic configuration of the framework in a separate application. Changing the behavior of the T1-FLC is now simply a matter of modifying the configuration file without the need for recompilation any of the related binaries. As an added benefit, the Wizard application provides a nice visual interface able to serve as a training aid in a classroom environment. As a series of standalone .NET assemblies combined with the Wizard application the framework demonstrates an easier and/or cost-effective way to implement a working, callable FLC when compared other frameworks and with fuzzy Type-2 support.

Future work is ongoing in order to broaden both the functionality and applicability of the framework. Among the planned extensions are a Wizard for configuring a FIS based upon fuzzy Type-2 objects, additional unary and binary fuzzy operators, discrete fuzzy sets and fuzzy numbers, complex rules and database extensions as well as improvements to the Wizard interface in general. Also in progress are the addition of genetic, memetic and local search algorithms and data mining techniques into the framework. We believe these additions will allow the creation of a framework that is “trainable” and self-optimizing. Finally, extending the framework to support contextual fuzzy constructs provides a comprehensive tool to support an even wider range of uses. Upon completion the University of Idaho hopes to make this framework available as an open source project and further collaborative effort.

VI REFERENCES

[1] Zadeh, L. A. (1965) Fuzzy Sets, Information and Control 8(3), 338-353

[2] Cox (1994) The fuzzy systems handbook; Academic Press

[3] Masaharu Mizumoto, Kokichi Tanaka, Some properties of fuzzy sets of type 2, Information and Control, Volume 31, Issue 4, August 1976, Pages 312-340

[4] Mendel, J.M.; John, R.I.B.; , "Type-2 fuzzy sets made simple," Fuzzy Systems, IEEE Transactions on , vol.10, no.2, pp.117-127, Apr 2002

[5] Hagras, H.; Wagner, C.; , "Towards the Wide Spread Use of Type-2 Fuzzy Logic Systems in Real World Applications," Computational Intelligence Magazine, IEEE , vol.7, no.3, pp.14-24, Aug. 2012

[6] Limbu, D.K.; Yeow Kee Tan; Ridong Jiang; Tran Ang Dung; , "A software architecture framework for service robots," Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on , vol., no., pp.1736-1741, 7-11 Dec. 2011

[7] Pressman RS (2005) Software engineering, a practitioner's approach, 6th Ed., McGraw Hill

[8] Buschmann, F.; , "Introducing the Pragmatic Architect," Software, IEEE , vol.26, no.5, pp.10-11, Sept.-Oct. 2009

[9] AForge.NET Framework (n.d.); URL from Jan.2013: http://www.aforgenet.com/aforge/framework/

[10] Sourceforge.NET Framework (n.d.); URL from Jan.2013: http://www.sourceforgenet.com/projects/octave-fuzzy/

[11] CodeProject Fuzzy Framework (n.d.); URL from Jan.2013: http://www.codeproject.com/Articles/151161/Fuzzy-Framework/

[12] Wyne, M.F.; , "SOOD: A simulation tool for OODB," Global Engineering Education Conference (EDUCON), 2012 IEEE , vol., no., pp.1-9, 17-20 April 2012

[13] Moreno-Velo, F.J.; Barriga, A.; Sanchez-Solano, S.; Baturone, I.; , "XFSML: An XML-based modeling language for fuzzy systems," Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on , vol., no., pp.1-8, 10-15 June 2012

[14] Trivedi JA, Sajja PS (2011) Framework for automatic development of type-2 fuzzy, neuro and neuro-fuzzy systems, International Journal of Advanced Computer Science and Applications, vol. 2, No. 1, Jan 2011

[15] Nilesh N. Karnik, Jerry M. Mendel, Centroid of a type-2 fuzzy set, Information Sciences, Volume 132, Issues 1–4, February 2001, Pages 195-220

[16] McCarty, K.; Manic, M.; , "Line-of-sight tracking based upon Manic, M.; , "Line-of-sight tracking based upon modern heuristics approach," Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on , vol., no., pp.40-45, 3-5 June 2008