Fuzzy mice
IT3709

Nordmoen, Jørgen H.
Tørresen, Håvard

November 11, 2012

Abstract

This paper is an introduction to an implementation of fuzzy logic coupled with a Qt based simulation. In this paper we will try to explain how our fuzzy logic work and how that is coupled with a mouse simulation to test out fuzzy rules.

Contents

List of Figures

1 Introduction

Fuzzy logic is a type of logic where each value can take on a range of truth values. This type of logic deals with degrees of truth in the rules it forms, each expression in the language can take on many values and all rules might fire to some degree.

Fuzzy logic then is well suited for situations where we want to use expressions that a human might use. For example, in first order logic we can express things like "The car is traveling at a speed over 50km/h", this is all well and good, but in most regular conversations humans would not express them self like this. Instead humans would use a sentence like "The car is traveling quite fast". This sentence is hard to represent in first order logic since we have a couple of variables which does not have a single value. "Fast" is not something a computer can work with straight out of the box since "Fast" might be a range of values. This is where fuzzy logic comes in. In fuzzy logic we can explain "Fast" as a range of values and when asked if the car is traveling "fast" we can evaluate that to a degree of truth. The car might not be traveling that fast and so our rules might say that it is fast to a degree of "0.2".

In this project we have implemented fuzzy logic with the aim of controlling mice which are running around a designated area. We read in rules written in a special format (see A) and then use those rules to control the mice with fuzzy logic.

We will first start talking a bit about our fuzzy logic implementation. Then we will move over to the Qt simulation. Finally we will wrap up with some thought and results.

2 Fuzzy logic

In this section we will describe how our fuzzy logic was implemented and try to give some arguments for some of the decisions that we made to arrive at this point.

Our fuzzy logic is divided into three parts. We have the rule parsing which as the name implies parses a specific set of rules and converts them into a runtime structure. The runtime structure is tasked with representing the fuzzy logic and letting other parts of our code interact to get results back. The runtime structure consists of three parts, the "if statement", the "conditional statement" and the "fuzzy expression". The fuzzy expression consists internally of fuzzy sets which contains the knowledge about how rules should be divided into sets of truth values. The last part of the fuzzy logic is the "reasoner" which contains two methods for inference, Mamdani and Sugeno.

2.1 Runtime structure

As stated the runtime structure consists of three parts, each corresponding to a different statement in our rule parsing(see A). To be able to parse as diverse rules as possible we designed the runtime structure around the notion of generality. Each rule is broken down into separate pieces and for each such piece we create a separate runtime structure to keep track of what is going on. When we evaluate

a rule we propagate the necessary information down trough all the structures before we are left with nothing but fuzzy expressions.

The top layer of the structure is the if statements, these represent individual rules which lead to a result. The if statement is what the outside world sees of our structure and it is the only place to interact with. The if statement consists of conditional expressions which hold either a fuzzy expression or a concatenation of conditional expressions. The bottom layer of the structure is the fuzzy expressions which holds the variables and the fuzzy sets to evaluate those variables against. The fuzzy sets is a structure which can evaluate whether a value is within its range and if so to what degree. For this project we support sets representing triangles, trapezoids and simple gradients.

In appendix B we have an example of what the runtime structure has to support. The "define fuzzyset" lines defines fuzzy sets as described above. The lines beginning with "if" represents a complete if statement. An expression surrounded by pretenses represents either a fuzzy expression or if there is an and between to such expressions a conditional statement.

2.2 Fuzzy Inference

To make use of the runtime structure we have to implement some sort of inference which can use the rules created to give output in some form. This is done by doing inference on the input. In our cases we get input from the simulator telling us about the variables in the system. Then we use either Mamdani or Sugeno inference to interpret those variables and what they mean. Our inference returns the action most associated with the rules or if no rules fires at all we throw an exception which must be dealt with other places.

For Mamdani inference we go over a certain number of steps in the output space and sum up all the values and their degree from all the rules. To find out whether or not a certain value is within a certain set we decided that we would try all of them and select the one which has the highest degree of "ownership" of that value. This is slow, but it is easy to think about and easy to implement.

Since Sugeno inference is slightly easier all we have to do is just for each result set that we get back we take the middle value and sum up all these values. This is much faster, but it does lack the resolution of Mamdani. With the simple rules that we have tried for our mice, Mamdani works fine, but if this would become a problem we could easily switch to Sugeno.

A Rule parsing

Below is a description of our fuzzy rule format, each rule described in this format should be possible to parse with our current implementation.

Listing 1: BNF of our rules

```
IF          = if COND_ST then action is ACTION
COND_ST     = (COND_ST [and|or] COND_ST) | (
    FUZZY_EXPR)
FUZZY_EXPR  = LINGVAR [is|not] VAR_NAME
ACTION      = VAR_NAME
LINGVAR     = [a-z]+
```

```
VAR_NAME        = [a-z]+
```

B Example rules

Listing 2: Example rules representing project risks

```
define lingvar funding: inadequate, marginal, adequate
define lingvar staffing: small, large
define lingvar risk: low, normal, high

define fuzzyset funding.inadequate: grade = [(28, 1),
    (43, 0)]
define fuzzyset funding.marginal: triangle = [(28, 0),
    (70, 1), (112, 0)]
define fuzzyset funding.adequate: grade = [(84, 0), (112,
     1)]

define fuzzyset staffing.small: grade = [(33, 1), (64, 0)
    ]
define fuzzyset staffing.large: grade = [(44, 0), (66, 1)
    ]

define fuzzyset risk.low: grade = [(20, 1), (40, 0)]
define fuzzyset risk.normal: triangle = [(20, 0), (50, 1)
    , (80, 0)]
define fuzzyset risk.high: grade = [(60, 0), (80, 1)]

if ((funding is adequate) or (staffing is small)) then
    risk is low
if ((funding is marginal) and (staffing is large)) then
    risk is normal
if (funding is inadequate) then risk is high
```