Fuzzy mice
IT3709

Nordmoen, Jørgen H.
Tørresen, Håvard

November 13, 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

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 thoughts 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 are structures which can evaluate whether a value is within their 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 parentheses 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.

3 Qt Simulation

The Qt[1] simulation is not necessarily the meat of our project, but it's the part that glues it all together and helps us visualize how the inference is working.

Most of the simulation is based on the colliding mice example created by Qt for its C++ UI framework. We have used the PySide[2] bindings and used the colliding mice example[3] to base most of our simulation.

[1]http://qt-project.org/
[2]http://qt-project.org/wiki/PySide
[3]http://qt.gitorious.org/pyside/pyside-examples/blobs/bc97d0b794dfd153462ac409569d73dd991b4e1f/examples/graphicsview/collidingmice/collidingmice.py

The example features several mice running around on a small field, turning around when they reach the edge. We have extended it with fuzzy logic and given each mouse health, strength and speed, to control different aspects of the mice and the fuzzy logic. Whenever two or more mice collide with each other, a fight will occur, and damages are dealt semi-randomly based on the strength of each mouse. In order to restrict the number of fights per second, we added a counter that only allows a mouse to fight every fifth timer-tick.

From the code that we reused from Qt, each mouse will perform an action 30 times per second (specifically, we use a timer on 1000/33ms intervals). In this time, each mouse tries to stay within a certain distance to the middle of the field, but are controlled by our fuzzy logic if any of the rules fire. Each mouse has a "fuzzy reasoner" which deals with the inference. This reasoner has been created with the interpreted fuzzy rules and takes a certain number of arguments. We then identify the two strongest mice within view and feed the reasoner the values associated with the other mice and our own health. The reasoner then uses the fuzzy rules and evaluates which action should be taken. This gives us one action per mouse, which are then prioritized by type: fleeing is more important than attacking, and attacking is more important than ignoring. Knowing which mouse to react to, we then steer the mouse towards, or away from the most important mouse we identified. If we decide to ignore both mice, we just continue straight ahead. In the case where nothing is returned from the reasoner, we default to a fake ignore-action, as it is usually caused by having no mice in view.

To easier visualize what each mouse is doing, we made them change ear-color according to the current action: dark red for attacking, gray for fleeing, and dark yellow when ignoring. When two mice collide, their ears turn bright red.

4 Reflections and Results

In this last section we will briefly reflect on the project and what we have done, before we move on to the results of our simulations and what can possibly interpreted from that.

4.1 Reflection

In this project we have used fuzzy logic to implements fuzzy rules which have controlled simulated mice. The challenge with this project has been how to represent the fuzzy logic in a runtime structure. With this in place most of the pieces was done, the inference mechanisms then just had to do their job and get the final output done. When creating the runtime structure we had to think about how to get different rules to work together, but when we managed to create conditional statements which could be weaved together, it all fell into place.

4.2 Results

From what we have seen in a number of different runs, the rules work quite well. The challenge is to create "intelligent" fuzzy rules. In some sense this is where our project is lacking, we have devoted much time to get the fuzzy logic up and running ,and we have used some time to get the simulation to work. We can easily see that the rules we have work, and we can see that the inference and everything work together, but to call it intelligent is a stretch.

In the different runs we can see that our rules fire, by seeing the mice hunt and flee from enemies. We can see that sometimes no action is selected and mice start fighting even though that might not have been the brightest thing to do. Since the area is quite limited, much fighting is going on at the start because no mice really cares about what is happening, but when the dust settles and there are only a few mice left, we can see that the strong start hunting the weak. Sometimes we see the weak try to flee, and sometimes they try to fight. When this is happening most of the behavior is down to the minimalistic rules that we have used. With more advanced rules, more "intelligent" behavior should be observed.

We have laid the foundations for a very good fuzzy logic implementation, which has proven itself through our simulations.

4.3 Future work

There are many aspects of our fuzzy logic which can be improved. Chief among them is improved parsing. What our implementation lacks is the power to parse arbitrarily created if statements, in its current form we can do what has been required of us, but it is not as good as it can be.

We would also like to improve how rules are parsed using Mamdani inference. As it stands now, we have some restrictions that can hold the inference back compared to what it should be able to do. The Sugeno inference works very well, much because it's easier to implement, but also because our technique was a bit more elegant.

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.

All rules have to be on a single line for the parsing to work. This is a flaw with the parsing mechanisms used and not the runtime structures created.

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
```