ECE 478

Portland State University

Fuzzy Logic for Robo-Magellan

Chris Clark
Bradon Kanyid
Tyler Tricker

December 10, 2012

1 Overview

For homework 3, we implemented a fuzzy obstacle avoidance ruleset for our Robo-Magellan robot. The logic was implemented in C++ using three classes: a fuzzy number class, a fuzzy function class, and a fuzzy set class. The rules were derived from a paper written at Tsinghua University in 1994 entitled "Fuzzy Logic Based Robot Navigation In Uncertain Environments By Multisensor Integration" written by Wei Li[1]. Currently the robot doesn't have a goal or target; it is simply attempting to travel and avoid obstacles.

2 Fuzzy Logic Implementation

2.1 Fuzzy_t Class

The fuzzy number class encapsulates a floating point number that is in the domain of $[0.0, 1.0]$, where 1.0 is completely true and 0.0 is completely false. This class in particular defines the standard set of fuzzy operators.

```
/**
 * Fuzzy Number Implementation
 */
class Fuzzy_t
{
public:

    Fuzzy_t(float value)
    {
        this->value = value;
    }

    Fuzzy_t operator|(const Fuzzy_t& right) const
    {
        return std::max<float>(this->value, right.value);
    }

    Fuzzy_t operator&(const Fuzzy_t& right) const
    {
        return std::min<float>(this->value, right.value);
    }

    Fuzzy_t operator^(const Fuzzy_t& right) const
    {
        return (*this & ~right) | (~*this & right) ;
    }

    Fuzzy_t operator~() const
    {
    return 1.0f - value;
    }

    bool   defuzzy() const
    {
        return (value > 0.5)? true: false;
    }
```

[1] http://www.cs.csubak.edu/ wli/Wei_Li_Pub/IEEE_MFI94_Li.pdf

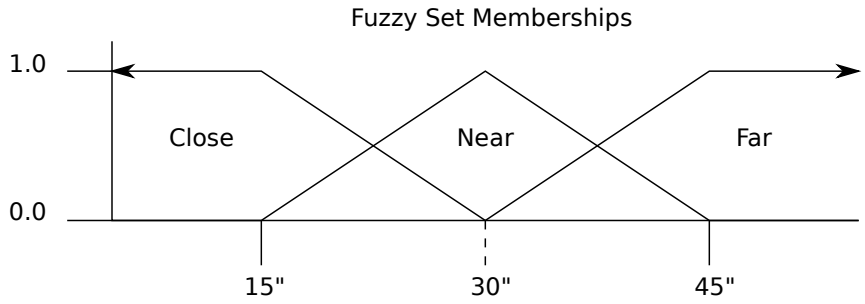

Figure 1: Fuzzy Set Visualization

```
    float getvalue()
    {
        return value;
    }

private:
    float value;
};
```

2.2 Fuzzy Membership Functions

The membership function implemented are standard triangular-shaped membership functions defined by their minimum and maximum range, (min, max). Everything outside these ranges is 0.0 and the midway point between the minimum and maximum is 1.0.

```
/***
 * Fuzzy Membership Function. This is a triangulated function with a
 * 1.0 output directly inbetween the min and max.
 */
class FuzzyMember
{
public:
    FuzzyMember(const float& min, const float& max)
    :begin(min),
     end(max)
    {}
    FuzzyMember(); // default constructor

    virtual ~FuzzyMember(){}

/***
 * This is the triangulation function to determine the membership of a variable.
 * Does not work for infinite values and nans.
 */
    virtual Fuzzy_t membership(const float& value)
    {
    if ((value < begin) | (value > end))
        {
            return 0.0f;
        }
```

```
    else
        {
            float halfspan = (end - begin) / 2.0f;
            float dist     =  halfspan + begin - value;
            dist = fabs(dist);
            return 1.0f - dist / halfspan;
        }
    }
private:
    friend class FuzzySet;

    // Private state
    float begin;
    float end;
};
```

2.3 Fuzzy Set

The fuzzy set class was derived from the C++ STL Vector container class. This allws the set to be extended, changed and manipulated using standard c++ algorithms and syntax. Currently, set operators are not implemented but may be in the future with the use of STL's Map class. The benefits of using the map class is that fuzzy set operators can be done at runtime to allow for fuzzy logic synthesis.

```
/***
 * Fuzzy Set Impletementation
 * Use an enum and pushback to initialize the membership set
 */
class FuzzySet : public std::vector<FuzzyMember>
{
public:
    FuzzySet() : vector<FuzzyMember>::vector()
    {
        input = NULL;
    }
    virtual ~FuzzySet()
    {
        input = NULL;
    }

// Automatically returns the membership of the requested element
    Fuzzy_t operator[](size_type n)
    {
    FuzzyMember& member =
        this->vector<FuzzyMember>::operator[](n);
    if(input)
    {
        if((n == 0) && (*input < member.begin))
        {
            return 1.0f;
        }
        else if (n == (this->size() - 1) && (*input > member.end))
        {
            return 1.0f;
```

```
        }

        return member.membership(*input);
    }
    else return 0.0f;
    }

    void set_input(float* input_variable)
    {
        input = input_variable;
    }
private:
    float* input;
};
```

2.4 Writing Rules

Because the fuzzy set class is a derived class from C++'s STL Vector class, membership functions can be indexed like integers. Creating an enum variable to index into the fuzzy set makes code more readable and avoids the overhead of string comparisons.

```
enum kettle_members
{
        Cold,
        Warm,
        Hot,
        Max
};

void example(void)
{
        FuzzySet kettle;
        float temp;
        kettle.setinput(&temp);
        kettle.resize(Max - 1);
        //FuzzySet[0] is always [-inf, max)
        kettle[Cold]  = Fuzzy_Member(0.0f, 30.0f);
        sonar1[Warm]  = Fuzzy_Member(15.0f, 45.0f);
        sonar1[Hot]   = Fuzzy_Member(30.0f, 60.0f);

        //Fuzzy Rules
        if((kettle[Hot]).defuzzy())
        {
                // turn kettle off
                setHeat(0.0f);
        }
        else if((kettle[Warm]).defuzzy())
        {
                // turn heat on low
                setHeat(0.5f);
        }
        else if((kettle[Cold]).defuzzy())
        {
                // turn heat on high
```

```
                setHeat(1.0f);
        }

        // alternative method
    // 0.0f -> heat off 1.0f -> heat on full
        setHeat(~kettle[Hot].getvalue());

}
```

3 Object Avoidance with Fuzzy Logic

The 2012 Portland State Magellan Bot has access to four long range sonar sensors and one GPS unit to attempt to navigate a course. We integrated the long range sonar to demonstrate that obstacle avoidance can be done using only four fuzzy rules.

3.1 Fuzzy Sets

```
    sonar[VERYNEAR] = FuzzyMember(00.0f, 30.0f);
    sonar[NEAR] = FuzzyMember(15.0f, 45.0f);
    sonar[FAR] = FuzzyMember(30.0f, 60.0f);
```

3.2 Demo Rule Set and Motor Controller

Our simple implementation of four rules allow the robot to avoid most obstacles. Each rule checks the distance on a few of the sonar sensors, determines which direction it should move to avoid the object and then updates the motor controller appropriately. The motor controller uses a degree-based system to control the left and right motors. $sin(direction \cdot speed + 45°)$ controls the left motor's speed while $sin(45° - direction \cdot speed)$ controls the right. This allows a single variable to control the motors where: 0 is straight forward, 90 is only right wheels, 180 is spin around to the right (left forward, right backwards), 360 is back up.

```
if((frontLeft[VERYNEAR] | frontRight[VERYNEAR]).defuzzy())
{
    int direction = (s2.distance - s3.distance) < 0 ? 1 : -1;
    motor->setMovement(direction * 180, 0.5);
    Thread::wait(500); //turn most of the way around
}
else if (((frontLeft[NEAR] | frontRight[NEAR]).defuzzy()))
{
    int direction = (s2.distance - s3.distance) < 0 ? 1 : -1;
    motor->setMovement(direction * 60, 0.5);
}
else if (((~sideLeft[FAR] | ~sideRight[FAR]).defuzzy()))
{
    int direction = (s1.distance - s4.distance) < 0 ? 1 : -1;
    motor->setMovement(direction * 60, 0.5);
}
else if (((frontLeft[FAR] | frontRight[FAR]).defuzzy()))
{
    motor->setMovement(0, 0.5);
}
```

4 Conclusion

The fuzzy logic rules implemented currently are sufficient for avoiding obstacles, but do not lead the robot towards any goal or waypoint as described in the Robo-Magellan competition.

Integrating the GPS sensor information for heading should modify the rule set to be similar to those rules found in Wei Li's paper.

We also will need to use the GPS sensor to determine when to exchange rule sets depenending on the mode that the robot is in. When far from a waypoint, the robot should use the described rule set for obstacle avoidance, augmented with the GPS information for general heading. When close to an obstacle, we will need to switch the rule set to an obstacle-seeking mode, where it will seek out the marker at the waypoint and touch it.

In the following quarter, for this seeking mode, it may make sense to use a simple vision system, as the waypoint marker is always a road cone, and therefore, we should be able to look for the color orange. This could again be implemented in our fuzzy rule set, giving a single integrated sensor network to base our rules from.