cs380su21-meta.sg

Grade

Reviewed on Thursday, 29 July 2021, 12:11 AM by Automatic grade
grade: 100.00 / 100.00

Assessment report [-]
[+]**Summary of tests**

Submitted on Thursday, 29 July 2021, 12:11 AM (Download)

functions.cpp

```cpp
/*!*****************************************************************************
\file functions.cpp
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 11
\date 07-26-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/
#include "functions.h"

namespace AI
{


} // end namespace
```

functions.h

```
/*!*****************************************************************************
\file functions.h
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 11
\date 07-26-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <list>
#include <map>
#include <climits>
#include <cfloat> // FLT_EPSILON, ...
#include <memory> // shared_ptr

#include "data.h"

#define UNUSED(x) (void)x;

namespace AI
{
    // Comparison function for floats
    inline bool isEqual(float a, float b)
    {
        float epsilon = 128 * FLT_EPSILON;
        float abs_th = FLT_MIN;

        if (a == b) return true;

        float diff = std::abs(a - b);
        float norm = std::min((std::abs(a) + std::abs(b)),
                              std::numeric_limits<float>::max());

        return diff < std::max(abs_th, epsilon* norm);
    }

    //  Definition of the base fuzzy set class
    class FuzzySet
    {
    protected:
        // Members that define the shape
        float peakPoint;
        float leftOffset;
        float rightOffset;

        // representativeValue - the maximum of the set's membership function.
        //For instance, if the set is triangular then this will be the peak
        //point of the triangular. If the set has a plateau then this value
        //will be the mid point of the plateau. This value is set in creation
        //to avoid run-time calculation of mid-point values.
        float representativeValue;

        // DOM - holds the degree of membership of a given value in this set
        float DOM;

    public:
        FuzzySet(float peakPoint, float leftOffset, float rightOffset,
            float representativeValue)
            : peakPoint{ peakPoint }, leftOffset{ leftOffset },
            rightOffset{ rightOffset },
            representativeValue{ representativeValue }, DOM{ 0.0f }{}

        virtual ~FuzzySet(){}

        /*****************************************************************
        \brief
        Calculates the degree of membership for a particular value. It does not
        set DOM to the value passed as the parameter because the centroid
        defuzzification method also uses this method to determind the degree of
        memberships of the values it uses as its sample points.

        \param val
        Value to calculate for.

        \return
        Returns the degree of membership
        *****************************************************************/
        virtual float calculateDOM(float val) const
        {
            UNUSED(val);

            return 0.0f;
        }

        void clearDOM()
        {
            DOM = 0.0f;
        }

        float getDOM() const
        {
            return DOM;
        }

        void setDOM(float val)
        {
            DOM = val;
        }

        float getRepresentativeValue() const
        {
            return representativeValue;
```

```
109         }
110
111         /*************************************************************************
112         \brief
113         If this fuzzy set is part of a consequent floatiable, and it is fired by
114         a rule then this method sets the DOM (in this context, DOM represents a
115         confidence level) to the maximum of the parameter value or the set's
116         existing member DOM value.
117
118         \param val
119         Value to caculate for.
120         *************************************************************************/
121         void ORwithDOM(float val)
122         {
123             if (val > DOM)
124                 DOM = val;
125         }
126
127         // Fuzzify a value by calculating its degree of membership
128         FuzzySet* fuzzify(float val)
129         {
130             DOM = calculateDOM(val);
131             return this;
132         }
133     };
134
135     //  Definition of a fuzzy set that has a left shoulder shape.
136     class FuzzySet_LeftShoulder : public FuzzySet
137     {
138     public:
139         FuzzySet_LeftShoulder(float peakPoint, float leftOffset,
140             float rightOffset)
141             : FuzzySet(peakPoint, leftOffset, rightOffset,
142                 peakPoint - leftOffset / 2){}
143
144         /*************************************************************************
145         \brief
146         Calculates the degree of membership for a particular value. It does not
147         set DOM to the value passed as the parameter because the centroid
148         defuzzification method also uses this method to determind the degree of
149         memberships of the values it uses as its sample points.
150
151         \param val
152         Value to calculate for.
153
154         \return
155         Returns the degree of membership
156         *************************************************************************/
157         float calculateDOM(float val) const
158         {
159             // Test for the case where the left or right offsets are zero
160             if (isEqual(this->peakPoint, val) &&
161                 (isEqual(this->leftOffset, 0.0f) ||
162                 isEqual(this->rightOffset, 0.0f)))
163                 return 1.0f;
164
165             // Find DOM if right of center
166             if ((val >= this->peakPoint) &&
167                 (val < (this->peakPoint + this->rightOffset)))
168                 return (1.0f / -this->rightOffset) *
169                 (val - (this->peakPoint)) + 1.0f;
170
171             // Find DOM if left of center
172             if ((val < this->peakPoint) &&
173                 (val >= this->peakPoint - this->leftOffset))
174                 return 1.0f;
175
176             // Out of range
177             return 0.0f;
178         }
179
180     };
181
182     // Definition of a fuzzy set that has a right shoulder shape.
183     class FuzzySet_RightShoulder : public FuzzySet
184     {
185     public:
186         FuzzySet_RightShoulder(float peakPoint, float leftOffset,
187             float rightOffset)
188             : FuzzySet(peakPoint, leftOffset, rightOffset,
189                 peakPoint + rightOffset / 2){}
190
191         /*************************************************************************
192         \brief
193         Calculates the degree of membership for a particular value. It does not
194         set DOM to the value passed as the parameter because the centroid
195         defuzzification method also uses this method to determind the degree of
196         memberships of the values it uses as its sample points.
197
198         \param val
199         Value to calculate for.
200
201         \return
202         Returns the degree of membership
203         *************************************************************************/
204         float calculateDOM(float val) const
205         {
206             // Test for the case where the left or right offsets are zero
207             // (to prevent divide by zero errors below)
208             if (isEqual(this->peakPoint, val) &&
209                 (isEqual(this->leftOffset, 0.0f) ||
210                 isEqual(this->rightOffset, 0.0f)))
211                 return 1.0f;
212
213             // Find DOM if left of center
214             if ((val <= this->peakPoint) &&
215                 (val > (this->peakPoint - this->leftOffset)))
216                 return (1.0f / this->leftOffset) *
```

```
            (val - (this->peakPoint - this->leftOffset));

        // Find DOM if right of center and less than center + right offset
        if ((val > this->peakPoint) &&
            (val <= this->peakPoint + this->rightOffset))
            return 1.0f;

        // Out of range
        return 0.0f;
    }
};

// This defines a fuzzy set that is a singleton (a range
// over which the DOM is always 1.0f)
class FuzzySet_Singleton : public FuzzySet
{
public:
    FuzzySet_Singleton(float peakPoint, float leftOffset, float rightOffset)
        : FuzzySet(peakPoint, leftOffset, rightOffset, peakPoint){}

    /*************************************************************************
    \brief
    Calculates the degree of membership for a particular value. It does not
    set DOM to the value passed as the parameter because the centroid
    defuzzification method also uses this method to determind the degree of
    memberships of the values it uses as its sample points.

    \param val
    Value to calculate for.

    \return
    Returns the degree of membership
    *************************************************************************/
    float calculateDOM(float val) const
    {
        if ((val >= this->peakPoint - this->leftOffset) &&
            (val <= this->peakPoint + this->rightOffset))
            return 1.0f;

        return 0.0f;
    }
};

// This is a simple class to define fuzzy sets that have a triangular
// shape and can be defined by a mid point, a left displacement and a
// right displacement.
class FuzzySet_Triangle : public FuzzySet
{
public:
    FuzzySet_Triangle(float peakPoint, float leftOffset, float rightOffset)
        : FuzzySet(peakPoint, leftOffset, rightOffset, peakPoint){}

    /*************************************************************************
    \brief
    Calculates the degree of membership for a particular value. It does not
    set DOM to the value passed as the parameter because the centroid
    defuzzification method also uses this method to determind the degree of
    memberships of the values it uses as its sample points.

    \param val
    Value to calculate for.

    \return
    Returns the degree of membership
    *************************************************************************/
    float calculateDOM(float val) const
    {
        // Test for the case where the left or right offsets are zero
        // (to prevent divide by zero errors below)
        if (isEqual(this->peakPoint, val) &&
            (isEqual(this->leftOffset, 0.0f) ||
            isEqual(this->rightOffset, 0.0f)))
            return 1.0f;

        // Find DOM if left of center
        if ((val <= this->peakPoint) &&
            (val >= (this->peakPoint - this->leftOffset)))
            return (1.0f / this->leftOffset) *
                    (val - (this->peakPoint - this->leftOffset));

        // Find DOM if right of center
        if ((val > this->peakPoint) &&
            (val < (this->peakPoint + this->rightOffset)))
            return (1.0f / -this->rightOffset) *
                    (val - (this->peakPoint)) + 1.0f;

        // Out of range
        return 0.0f;
    }
};

// Fuzzy logic works with membership values in a way that mimics Boolean
//logic. Replacements for basic logic operators AND and OR are defined here.

// Definition of the base operator class
class FuzzyOperator
{
protected:
    std::list<std::shared_ptr<FuzzySet>> sets;

public:
    FuzzyOperator(std::initializer_list<std::shared_ptr<FuzzySet>> sets={})
        : sets{ sets }{}

    virtual ~FuzzyOperator(){}

    // Returns the minimum DOM of the sets it is operating on
    virtual float getDOM()
```

```
        {
            return 0;
        }

        void clearDOM()
        {
            for (std::shared_ptr<FuzzySet> set : sets)
                set->clearDOM();
        }

        void ORwithDOM(float val)
        {
            for (std::shared_ptr<FuzzySet> set : sets)
                set->ORwithDOM(val);
        }
    };

    //  Definition of the AND operator class
    class FuzzyAND : public FuzzyOperator
    {
    public:
        FuzzyAND(std::initializer_list<std::shared_ptr<FuzzySet>> sets = {})
            : FuzzyOperator{ sets }{}

        /*****************************************************************
        \brief
        Retrieve the smallest DOM value within the sets.

        \return
        Returns the smallest DOM value.
        *****************************************************************/
        float getDOM()
        {
            float smallest = static_cast<float>(INT_MAX);

            for (auto& set : sets)
            {
                float cur = set->getDOM();
                if (cur < smallest)
                    smallest = cur;
            }
            return smallest == static_cast<float>(INT_MAX) ? 0.0f : smallest;
        }
    };

    // Definition of the OR operator class
    class FuzzyOR : public FuzzyOperator
    {
    public:
        FuzzyOR(std::initializer_list<std::shared_ptr<FuzzySet>> sets = {})
            : FuzzyOperator{ sets }{}

        /*****************************************************************
        \brief
        Retrieve the largest DOM value within the sets.

        \return
        Returns the largest DOM value.
        *****************************************************************/
        float getDOM()
        {
            float largest = static_cast<float>(INT_MIN);

            for (auto& set : sets)
            {
                float cur = set->getDOM();
                if (cur > largest)
                    largest = cur;
            }
            return largest == static_cast<float>(INT_MIN) ? 0.0f : largest;
        }
    };

    // Definition of the fuzzy variable class
    class FuzzyVariable
    {
    protected:
        // A map of the fuzzy sets that comprise this variable
        std::map<std::string, std::shared_ptr<FuzzySet>> sets;

        // The minimum and maximum value of the range of this variable
        float minRange;
        float maxRange;

    public:
        FuzzyVariable(): sets{ }, minRange{ 0.0f }, maxRange{ 0.0f }{}

        virtual ~FuzzyVariable(){}

        std::shared_ptr<FuzzySet> getSet(const std::string& name)
        {
            return sets[name];
        }

        // This method is called with the upper and lower bound of a set
        //each time a new set is added to adjust the upper and lower range
        //values accordingly
        void adjustRangeToFit(float minBound, float maxBound)
        {
            if (minBound < minRange)
                minRange = minBound;
            if (maxBound > maxRange)
                maxRange = maxBound;
        }

        // The following methods create instances of the sets named in the
        //method name and add them to the member set map. Each time a set of
        //any type is added the minRange and maxRange are adjusted accordingly
```

```
433
434         // Adds a left shoulder type set
435         FuzzyVariable& addLeftShoulderSet(const std::string& name,
436             float minBound, float peak, float maxBound)
437         {
438             sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
439                 std::shared_ptr<FuzzySet>(new FuzzySet_LeftShoulder(peak,
440                     peak - minBound, maxBound - peak))));
441
442             adjustRangeToFit(minBound, maxBound);
443             return *this;
444         }
445
446         // Adds a left shoulder type set
447         FuzzyVariable& addRightShoulderSet(const std::string& name,
448             float minBound, float peak, float maxBound)
449         {
450             sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
451                 std::shared_ptr<FuzzySet>(new FuzzySet_RightShoulder(peak,
452                     peak - minBound, maxBound - peak))));
453
454             adjustRangeToFit(minBound, maxBound);
455             return *this;
456         }
457
458         // Adds a triangular shaped fuzzy set to the variable
459         FuzzyVariable& addTriangularSet(const std::string& name, float minBound,
460             float peak, float maxBound)
461         {
462             sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
463                 std::shared_ptr<FuzzySet>(new FuzzySet_Triangle(peak,
464                     peak - minBound, maxBound - peak))));
465
466             adjustRangeToFit(minBound, maxBound);
467             return *this;
468         }
469
470         // Adds a singleton to the variable
471         FuzzyVariable& addSingletonSet(const std::string& name, float minBound,
472             float peak, float maxBound)
473         {
474             sets.insert(std::pair<std::string, std::shared_ptr<FuzzySet>>(name,
475                 std::shared_ptr<FuzzySet>(new FuzzySet_Singleton(peak,
476                     peak - minBound, maxBound - peak))));
477
478             adjustRangeToFit(minBound, maxBound);
479             return *this;
480         }
481
482         // Fuzzify a value by calculating its degree of membership in each of
483         //this variable's subsets. Takes a crisp value and calculates its
484         //degree of membership for each set in the variable.
485         FuzzyVariable* fuzzify(float val)
486         {
487             //for each set in the flv calculate the degree of membership for
488             //the given value
489             for (std::pair<std::string, std::shared_ptr<FuzzySet>> set : sets)
490                 set.second->setDOM(set.second->calculateDOM(val));
491
492             return this;
493         }
494
495         // Method for updating the DOM of a consequent when a rule fires
496         void ORwithDOM(float val)
497         {
498             for (std::pair<std::string, std::shared_ptr<FuzzySet>> set : sets)
499                 set.second->ORwithDOM(val);
500         }
501
502         /*************************************************************************
503         \brief
504         Defuzzifies the value by averaging the maxima of the sets that have
505         fired.
506
507         \return
508         Returns sum of (maxima * degree of membership) / sum (degree of
509         memberships)
510         *************************************************************************/
511         float deFuzzifyMaxAv()
512         {
513             float bottom = 0.0f;
514             float top = 0.0f;
515
516             for (std::pair<std::string, std::shared_ptr<FuzzySet>> set : sets)
517             {
518                 bottom += set.second->getDOM();
519
520                 top += set.second->getRepresentativeValue() *
521                        set.second->getDOM();
522             }
523
524             // Make sure bottom is not equal to zero
525             if (isEqual(0.0f, bottom))
526                 return 0.0f;
527
528             return top / bottom;
529         }
530
531         /*************************************************************************
532         \brief
533         Defuzzify the variable using the centroid method.
534
535         \param numSamples
536         Number of Samples.
537
538         \return
539         Returns the centroid by dividing total area by the sum of moments, like
540         calculating the center of mass of an object
```

```
541         ************************************************************************/
542         float deFuzzifyCentroid(int numSamples)
543         {
544             // Calculate the step size
545             float stepSize = (this->maxRange - this->minRange) /
546                              static_cast<float>(numSamples);
547
548             float totalArea = 0.0f;
549             float sumOfMoments = 0.0f;
550
551             // Step through the range of this variable in increments equal to
552             // stepSize adding up the contribution (lower of calculateDOM or the
553             // actual degree of membership of this variable's fuzzified value)
554             // for each subset. This gives an approximation of the total area of
555             // the fuzzy manifold.(This is similar to how the area under a curve
556             // is calculated using calculus... the heights of lots of 'slices'
557             // are summed to give the total area.)
558             // In addition the moment of each slice is calculated and summed.
559             for (int samp = 1; samp <= numSamples; ++samp)
560             {
561                 // for each set get the contribution to the area. This is the
562                 // lower of the value returned from calculateDOM or the actual
563                 // degree of membership of the fuzzified value itself
564                 for (std::pair<std::string,
565                     std::shared_ptr<FuzzySet>> set : sets)
566                 {
567                     float contribution = std::min(
568                     set.second->calculateDOM(this->minRange + samp * stepSize),
569                     set.second->getDOM());
570
571                     totalArea += contribution;
572                     sumOfMoments += (this->minRange + samp * stepSize) *
573                                     contribution;
574                 }
575             }
576
577             // Make sure total area is not equal to zero
578             if (isEqual(0.0f, totalArea))
579                 return 0.0f;
580
581             return sumOfMoments / totalArea;
582         }
583     };
584
585     // Definition of the fuzzy rule class of form
586     // IF antecedent THEN consequence
587     class FuzzyRule
588     {
589     protected:
590         // Antecedent (usually a composite of several fuzzy sets and operators)
591         std::shared_ptr<FuzzyOperator> antecedent;
592
593         // Consequence (usually a single fuzzy set, but can be several ANDed
594         //together)
595         std::shared_ptr<FuzzySet> consequence;
596
597     public:
598         FuzzyRule(std::shared_ptr<FuzzyOperator> antecedent,
599             std::shared_ptr<FuzzySet> consequence)
600             : antecedent{ antecedent }, consequence{ consequence }{}
601
602         // Updates the DOM (the confidence) of the consequent set with
603         // the DOM of the antecedent set.
604         std::shared_ptr<FuzzySet> calculate()
605         {
606             consequence->ORwithDOM(antecedent->getDOM());
607             return consequence;
608         }
609
610         void setConfidenceOfConsequentToZero()
611         {
612             consequence->clearDOM();
613         }
614     };
615
616     // Definition of the fuzzy module class
617     class FuzzyModule
618     {
619     public:
620         // Defuzzify methods supported by this module.
621         enum DefuzzifyMethod { max_av = 0, centroid = 1 };
622
623     protected:
624
625         // When calculating the centroid of the fuzzy manifold this value is
626         // used to determine how many cross-sections should be sampled
627         int numSamples;
628
629         // A map of all the fuzzy variables this module uses
630         std::map<std::string, FuzzyVariable> variables;
631
632         // An array containing all fuzzy rules
633         std::list<FuzzyRule> rules;
634
635     public:
636         FuzzyModule(): numSamples{ 15 }, variables{ }, rules{ }{}
637
638         FuzzyVariable& getVariable(const std::string& name)
639         {
640             return variables[name];
641         }
642
643         // Zeros the DOMs of the consequents of each rule. Used by Defuzzify()
644         void setConfidencesOfConsequentsToZero()
645         {
646             for (FuzzyRule rule : rules)
647                 rule.setConfidenceOfConsequentToZero();
```

```
    }

    // Creates and return a new 'empty' fuzzy variable.
    FuzzyVariable& createVariable(std::string varName)
    {
        variables.insert(std::pair<std::string, FuzzyVariable>
            (varName, FuzzyVariable()));

        return variables[varName];
    }

    // Adds a rule to the module
    void addRule(std::shared_ptr<FuzzyOperator> antecedent,
        std::shared_ptr<FuzzySet> consequence)
    {
        rules.push_back(FuzzyRule(antecedent, consequence));
    }

    // Calls the Fuzzify method of the variable with the same name
    void fuzzify(const std::string& varName, float val)
    {
        variables[varName].fuzzify(val);
    }

    /*************************************************************************
    \brief
    Defuzzify the variable given a fuzzy variable and a deffuzification
    method.

    \param varName
    String of variable names.

    \param method
    Deffuzifaction method.

    \return
    Returns a crsip value.
    *************************************************************************/
    float deFuzzify(const std::string& varName, DefuzzifyMethod method)
    {
        // Clear the DOMs of all the consequents of all the rules
        this->setConfidencesOfConsequentsToZero();
        // Process the rules
        for (FuzzyRule rule : rules)
            rule.calculate();

        // Defuzzify the resultant conclusion using the specified method
        switch (method)
        {
            case FuzzyModule::DefuzzifyMethod::centroid:
                return this->variables[varName].deFuzzifyCentroid
                        (this->numSamples);

            case FuzzyModule::DefuzzifyMethod::max_av:
                return this->variables[varName].deFuzzifyMaxAv();
        }

        return 0.0f;
    }
};

} // end namespace

#endif
```

VPL

◄ Showcase: Weapon

Jump to...

Slides. Part 1 ►