Module Interface Specification for MISEG

Ao Dong

November 24, 2019

1 Revision History

Date	**Version**	**Notes**
Nov 25	1.0	Initial Draft

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/Ao99/MIA/blob/master/docs/SRS/CA.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for MISEG which is for medical image segmentation.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/Ao99/MIA.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$. This document has one modification to the original notations: the concatenation notation || can be used to build a new sequence from an existing sequence. For example, s2 := $||(x : \mathbb{N}|x \in s1 \cdot x + 1)$, where s1 = $\langle 1, 2, ..., 10 \rangle$, then s2 = $\langle 2, 3, ..., 11 \rangle$.

The following table summarizes the primitive data types used by MISEG.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
boolean	boolean	a value in $\{true, false\}$

The following table summarizes other data types used by MISEG.

Data Type	**Notation**	**Description**
DICOM file	inputFile	a DICOM image file
DICOM frame	dcmFrame	a frame of image in a DICOM image file
image data	imageData	a data structure containing width, height and a sequence of pixel values
bitmap file	outputFile	an 8-bit 2D grayscale bitmap image file

The specification of MISEG uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, MISEG uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module Output Module Optimal Thresholds Calculation Image Verification Constant Values Control Module
Software Decision Module	Sequence Data Structure Image Data Structure

Table 1: Module Hierarchy

6 MIS of Control Module

6.1 Module

main

6.2 Uses

Input in Section 8, ThresCal in Section 10, Output in Section 11

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	emptyloadedImage, badThresholds, badMethodChoice

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

main():

- transition: use other modules by following these steps

  1. Get (filenameIn: string) and (filenameOut: string) from user
  2. Input.loadInput($filenameIn$)
  3. For $(j : \mathbb{N})$ from 0 to Input.numFrames, repeat the following steps
  4. (Input.isLoaded(j) $=$ true $\implies$ ThresCal.calculation(j) $|$ else $\implies$ emptyloadedImage)

5. (ThresCal.validThresholds = true $\implies$ Output.displayThresholds() | else $\implies$ badThresholds)
6. (ThresCal.methodChoice $\in$ {Constants.CHOICE1, Constants.CHOICE2} $\implies$ Output.writeOutput(filenameOut) | else $\implies$ badMethodChoice)

- output: none
- exception: $exc :=$
  Input.isLoaded[j] = $false$ $\implies$ emptyloadedImage
  ThresCal.validThresholds = $false$ $\implies$ badThresholds
  ThresCal.methodChoice $\notin$ {Constants.CHOICE1, Constants.CHOICE2} $\implies$ badMethodChoice

6.4.5 Local Functions

None

7 MIS of Constant Values

7.1 Module

Constants

7.2 Uses

None

7.3 Syntax

7.3.1 Exported Constants

MINX := 10
MAXX := 1000
MINY := 10
MAXY := 1000
CHOICE1 := 1
CHOICE2 := 2
EMPTY := 0

7.3.2 Access Routine Semantics

N/A

8 MIS of Input Module

8.1 Module

Input

8.2 Uses

Image Data Structure 9, Image Verification in Section 12

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
loadInput	s: string	-	FileError
verifyInput		-	
loadedImages	-	sequence of imageData	
numFrames	-	$\mathbb{N}$	
isLoaded	-	sequence of boolean	

8.4 Semantics

8.4.1 State Variables

loadedImages: sequence of imageData
numFrames: $\mathbb{N}$
isLoaded: sequence of boolean

8.4.2 Environment Variables

inputFile: a .dcm or .dcm30 DICOM medical image file

8.4.3 Assumptions

The data type String has a method parseToNum() to parse a string (such as “1”) to an $\mathbb{N}$ (such as 1).

8.4.4 Access Routine Semantics

Input.loadedImages:

- output: $out :=$ loadedImages

- exception: none

Input.numFrames:

- output: out := numFrames
- exception: none

Input.isLoaded:

- output: out := isLoaded
- exception: none

loadInput(s):

- transition: The filename s is first associated with the file f. inputFile is used to modify the state variables using the following procedural specification:
  1. Read the inputFile.
  2. numFrames := information from inputFile
  3. loadedImages := $||(f$: dcmFrame $| f \in$ inputFile $\cdot$ dcmToImage$(f))$
  4. verifyInput()
- output: none
- exception: exc := a file name s cannot be found or the format of inputFile is incorrect $\implies$ FileError

verifyInput():

- transition: This function modifies the state variables using the following procedural specification:
  1. isLoaded := $||(img$: imageData $| img \in$ loadedImages $\cdot$ ImageVerify.verify1File$(img))$
- output: none
- exception: none

8.4.5 Local Functions

dcmToImage: dcmFrame → imageData
dcmToImage$(f) \equiv$ ImageData$(f.x,\ f.y,$ stringToSequence$(f.s))$
stringToSequence: string → sequence of $\mathbb{N}$
stringToSequence$(str) \equiv ||$ $(pv$: string $| pv \in$ dcmFrame $\cdot$ String.parseToNum$(pv))$
pv is a string containing grayscale value for one pixel.

9 MIS of Image Data Structure

9.1 Module

ImageData

9.2 Uses

Constant Values 7

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
ImageData	$x : \mathbb{N}$, $y : \mathbb{N}$, p: sequence of $\mathbb{N}$	imageData	badWidthInput, badHeightInput, badPixelValueLength
width	-	$\mathbb{N}$	
height	-	$\mathbb{N}$	
pixelValue	-	sequence of $\mathbb{N}$	

9.4 Semantics

9.4.1 State Variables

width: $\mathbb{N}$
height: $\mathbb{N}$
pixelValue: sequence

9.4.2 Environment Variables

none

9.4.3 Assumptions

We only need images with width and height not less than Constants.MINX or Constants.MINY, and not greater than Constants.MAXX or Constants.MAXY.

9.4.4 Access Routine Semantics

Input.width:

- output: $out :=$ width

- exception: none

Input.height:

- output: *out* := height

- exception: none

Input.isLoaded:

- output: *out* := isLoaded

- exception: none

ImageData(x, y, p):

- transition: The parameters x and y are natural numbers, p is a sequence of natural numbers representing the pixel values from left to right, top to bottom. ImageData() is the conxtructor of this data structure, and it modifies the state variables using the following procedural specification:

  1. height := x
  2. height := y
  3. pixelValue := p

- output: := itself

- exception: $exc :=$
  $x \notin [Constants.MINX, Constants.MAXX] \implies$ badWidthInput
  $y \notin [Constants.MINY, Constants.MAXY] \implies$ badHeightInput
  $p.length \neq x \times y \implies$ badPixelValueLength

9.4.5 Local Functions

None

10 MIS of Optimal Thresholds Calculation

10.1 Module

ThresCal

10.2 Uses

Constant Values 7, Input in Section 8, Image Data Structure 9

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
calculation	$j : \mathbb{N}$	-	badResult1, badResult2
getMethodChoice	$c : \mathbb{N}$	-	badChoiceInput
imageIndex	-	$\mathbb{N}$	
methodChoice	-	$\mathbb{N}$	
validThresholds	-	boolean	
k1	-	$\mathbb{N}$	
k2	-	$\mathbb{N}$	

10.4 Semantics

10.4.1 State Variables

imageIndex: $\mathbb{N}$
methodChoice: string
validThresholds: boolean
k1: $\mathbb{N}$
k2: $\mathbb{N}$

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

ThresCal.imageIndex:

- output: out := imageIndex
- exception: none

ThresCal.methodChoice:

- output: out := methodChoice
- exception: none

ThresCal.validThresholds:

- output: out := validThresholds
- exception: none

ThresCal.k1:

- output: out := k1
- exception: none

ThresCal.k2:

- output: out := k2
- exception: none

calculation(j):
$j \in \mathbb{N}$ is the index of one imageData in the Input.loadedImages sequence.

- transition: This function modifies the state variables using the following procedural specification:

    1. imageIndex := j
    2. getMethodChoice()
    3. According to chosen method, calculate $k1$ or both $k1$ and $k2$:
       if methodChoice = Constants.CHOICE1,
        - $k1 := k1 \in$ [1, 254]. sigma2b1($k1$) = $\max\limits_{0<t1<255}$ sigma2b1($t1$)
        - $k2 := Constants.EMPTY$
        - validThresholds := $(k1 \in [1, 254] \implies true |\ else \implies false)$

       if methodChoice = Constants.CHOICE2,
        - $k1 := k1 \in [1, 254].\ k1 < k2 \wedge sigma2b2(k1, k2) = \max\limits_{0<t1<t2<255} sigma2b2(t1, t2))$

  - $k2 := k2 \in [1, 254].\ k1 < k2 \wedge sigma2b2(k1, k2) = \max_{0<t1<t2<255} sigma2b2(t1, t2)$
  - validThresholds := ($k1 \in$ [1, 254] $\wedge k2 \in$ [1, 254] $\implies$ true | else $\implies$ false)

- output: none
- exception: $exc :=$
  methodChoice = Constants.CHOICE1 $\wedge k1 \notin [1, 254] \implies$ badResult1
  methodChoice = Constants.CHOICE2 $\wedge (k1 \notin [1, k2) \vee k2 \notin (k1, 254] \implies$ badResult2

getMethodChoice(c):

- transition: The parameter c is a natural number representing user's choice. This function modifies the state variables using the following procedural specification:
  1. Use hardware to display a message, asking for user's input "1" or "2 " for method choice.
  2. methodChoice := ($c \in$ {Constants.CHOICE1, Constants.CHOICE2} $\implies c \mid else \implies$ Constants.EMPTY)
- output: none
- exception: $exc := c \notin$ {Constants.CHOICE1, Constants.CHOICE2} $\implies$ badChoiceInput

10.4.5 Local Functions

n: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$
n$(i, j) \equiv +(pv.\mathbb{N}|pv \in Input.loadedImages[j].pixelValue \cdot (pv = i \implies 1|else \implies 0))$
p: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
p$(i, j) \equiv n(i, j)/(+(i.\mathbb{N}|i \in [0, 255] \cdot n(i, j)))$
prb1: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
prb1$(t1, j) \equiv +(i.\mathbb{N}|i \in [0, t1] \cdot p(i, j))$
prb2: $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$
prb2$(t1, t2, j) \equiv +(i.\mathbb{N}|i \in [t1 + 1, t2] \cdot p(i, j))$
prb3: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
prb3$(t2, j) \equiv +(i.\mathbb{N}|i \in [t2 + 1, 255] \cdot p(i, j))$
m1: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
m1$(t1, j) \equiv (+(i.\mathbb{N}|i \in [0, t1] \cdot i \times p(i, j)))/prb1(t1, j)$
m2: $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$
m2$(t1, t2, j) \equiv (+(i.\mathbb{N}|i \in [t1 + 1, t2] \cdot i \times p(i, j)))/prb2(t1, t2, j)$
m3: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
m3$(t2, j) \equiv (+(i.\mathbb{N}|i \in [t2 + 1, 255] \cdot i \times p(i, j)))/prb3(t2, j)$
mg: $\mathbb{N} \to \mathbb{R}$
mg$(j) \equiv +(i.\mathbb{N}|i \in [0, 255] \cdot i \times p(i, j))$
sigma2b1: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$

sigma2b1$(t1, j) \equiv prb1(t1, j) \times (m1(t1, j) - mg(j))^2 + prb2(t1, 255, j) \times (m2(t1, 255, j) - mg(j))^2$
sigma2b2: $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$
sigma2b2$(t1, t2, j) \equiv prb1(t1, j) \times (m1(t1, j) - mg(j))^2 + prb2(t1, t2, j) \times (m2(t1, t2, j) - mg(j))^2 + prb3(t2, j) \times (m3(t2, j) - mg(j))^2$

11 MIS of Output Module

11.1 Module

Output

11.2 Uses

Constant Values 7, Image Data Structure 9, ThresCal in Section 10, Image Verification in Section 12

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayThresholds	-	-	
writeOutput	s: string	-	noAccess
createSegmentation	-	-	
segImage	-	imageData	

11.4 Semantics

11.4.1 State Variables

segImage: imageData

11.4.2 Environment Variables

outputFile: a bitmap file

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

Output.segImage:

- output: $out :=$ segImage
- exception: none

displayThresholds():

- transition: This function has the following procedural specification:

  1. If methodChoice = Constants.CHOICE1, use Hardware-Hiding Module to display the following message:
     "Single-global-threshold method selected, the threshold value k= " + k1 + "."
  2. If methodChoice = Constants.CHOICE2, use Hardware-Hiding Module to display the following message:
     "Multiple-global-threshold method selected, the threshold values k1= " + k1 + ", k2= " + k2 + "."

- output: none
- exception: none

writeOutput(s): This method use segImage to write a outputFile to the environment using the following procedural specification:

1. createSegmentation()
2. (ImageVerify.verify1File($segImage$) = true $\Longrightarrow$ continue | else $\Longrightarrow$ stop)
3. Use local references j for state variable in ThresCal: j = ThresCal.imageIndex
4. (ImageVerify.compare2Files($Input.loadedImages[j], segImage$) = true $\Longrightarrow$ continue | else $\Longrightarrow$ stop)
5. write a outputFile $s.bmp$ to the environment

- transition: none
- output: none
- exception: exc := no access to write a file to the output directory $\Longrightarrow$ noAccess

createSegmentation():

- transition: This function modifies the state variables using the following procedural specification:

  1. Use local references $j, c, k1, k2$ for state variables in ThresCal.
     - j = ThresCal.imageIndex
     - c = ThresCal.methodChoice
     - $k1$ = ThresCal.ThresCal.k1
     - $k2$ = ThresCal.ThresCal.k2
     - img := Input.loadedImages[j]
  2. Initiate a sequence pixelValue[$img.x \times img.y$]

3. pixelValue := $||(pv : \mathbb{N} | pv \in img.pixelValue \cdot (c =$Constants.CHOICE1 $\implies (pv > k1 \implies 255 |\ else \implies 0) |\ c =$Constants.CHOICE2 $\implies (pv > k2 \implies 255 |\ k2 \geq pv > k1 \implies 128 |\ else \implies 0))$
4. segImage := ImageData$(img.x, img.y, pixelValue)$

- output: none
- exception: none

11.4.5 Local Functions

None

12 MIS of Image Verification

12.1 Module

ImageVerify

12.2 Uses

Constant Values 7

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
verify1File	imageData	boolean	emptyImage, badWidth, badHeight, badPixelData
compare2Files	imageData, imageData	boolean	emptyImage1, emptyImage2, badWidth2, badHeight2

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

We only need images with width and height not less than Constants.MINX and Constants.MINY, and not greater than Constants.MAXX and Constants.MAXY.

compare2Files(imageData, imageData) does not check if these two inputs are valid, it assumes that during the previous steps, the software has called verify1File(imageData) to verify these two inputs individually.

12.4.4 Access Routine Semantics

verify1File(*img*): The parameter *img* is an instance of Image Data Structure.

- transition: none

- output: := $(img.x \in [Constants.MINX, Constants.MAXX] \wedge img.y \in [Constants.MINY, Constants.MAXY] \wedge (\forall pv \in img.pixelValue.\ pv \in [0, 255] \implies true |\ else \implies false)$

- exception: $exc :=$
  img is an empty instance of imageData type $\implies$ emptyImage
  $x \notin [Constants.MINX, Constants.MAXX] \implies$ badWidth
  $y \notin [Constants.MINY, Constants.MAXY] \implies$ badHeight
  $\forall pv \in img.pixelValue.\ pv \notin [0, 255] \implies$ badPixelData

compare2Files($img1, img2$): The parameters $img1$ and $img2$ are instances of Image Data Structure.

- transition: none

- output: := $((img1.x = img2.x) \wedge (img1.y = img2.y) \implies true |\ else \implies false)$

- exception: $exc :=$
  $img1$ is an empty instance of imageData type $\implies$ emptyImage1
  $img2$ is an empty instance of imageData type $\implies$ emptyImage2
  $img1.x \neq img2.x \implies$ badWidth2
  $img1.y \neq img2.y \implies$ badHeight2

12.4.5 Local Functions

None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL https://pdfs.semanticscholar.org/2d2f/609de3c6d694b88b5b987b05bd5ec53be372.pdf.

13 Appendix

Table 2: Possible Exceptions

Message ID	**Error Message**
emptyloadedImage	Error: The image of frame j is not loaded
badThresholds	Error: No correct thresholds have been calculated
badMethodChoice	Error: No correct segmentation method has been chosen
badWidthInput	Error: Image width must be $\in$ [Constants.MINX, Constants.MAXX]
badHeightInput	Error: Image height must be $\in$ [Constants.MINX, Constants.MAXX]
badPixelValueLength	Error: The length of image pixel value sequence must equal to Constants.MINX × Constants.MAX
badResult1	Error: $k1$ must be $\in$ [1, 254]
badResult2	Error: $k1$ and $k2$ must follow this rule: $1 \leq k1 < k2 \leq 254$
badChoiceInput	Error: Must choose input from the set {Constants.CHOICE1, Constants.CHOICE2}
emptyImage	Error: Cannot verify an empty image
badWidth	Error: Image width $\notin$ [Constants.MINX, Constants.MAXX]
badHeight	Error: Image height $\notin$ [Constants.MINX, Constants.MAXX]
badPixelData	Error: One or more pixel values $\notin$ [0, 255]
emptyImage1	Error: Cannot verify an empty original image
emptyImage2	Error: Cannot verify an empty segmentation image
badWidth2	Error: The original image and the segmentation image do not have the same width
badHeight2	Error: The original image and the segmentation image do not have the same height