Software Requirements &

Functional Specifications

**Main Program**

Prepared by:

<Alex Barrington>

<Jonathan Hjelmgren>

<Sophia Shanahan>

<Version 1.0>

Date Created:

<4.10.18>

Table of Contents

1. **Introduction …………………………………………………………….2**

1.1 Purpose **……………………………………………………………....2**

1.2 Document Conventions **……………………………………………...2**

1.3 Intended Audience and Definitions **………………………………….2**

1.4 Scope **………………………………………………………………...3**

**2. Overall Description ………………………………………………………….4**

2.1: Product Perspective **………………………………………………...4**

2.2: Product Functions **…………………………………………………..4-5**

2.3: User Classes **………………………………………………………..5**

2.4: Operating Environment **…………………………………………….5**

**3. External Interface Requirements ………………………………………….6**

3.1: Software Interfaces **………………………………………………...6**

**4. System Features …………………………………………………………… 7**

4.1: User input data **…………………………………………………….7**

4.2: Read in jpeg image **………………………………………………..7**

4.3: Call other functions/subsystems **…………………………………..8**

4.4: Write output data **………………………………………………….8**

**5.1 Other Nonfunctional Requirements …………………………………….9**

5.1: Performance requirements **………………………………………..9**

5.2: Safety Requirements **……………………………………………...9**

5.3: Software Quality Attributes **………………………………………9**

5.4: Performance Requirements **………………………………………10**

**6. References …………………………………………………………………10**

**7. Appendix A: Design of Main Program Subsystem ……………………...11**

**8. Appendix B: Test Plan ……………………………………………………12-17**

**1.Introduction**

1.1 Purpose

This software requirements document serves two purposes: one, to define both the

functionality and design of the main program subsystem, and two, to provide students shortly entering the professional world of biocomputing with the technical writing experience of defining software specifications. Further, the preparation of this document will ensure that team members are all on the same page with the product they are designing, and will serve as a resource for other teams building subsystems that will eventually be integrated with this subsystem.

Information on the scope of the project, external interfaces, system features, and design can be found within this document.

1.2 Document Conventions

No non-intuitive conventions were used in the writing of this document.

1.3 Intended Audience and Definitions

The intended audience for this document consists of the BIEN 4290 professor and TA for grading purposes, and other development teams to provide guidance how to prepare other subsystems with integration with the current one.

Table 1: Includes definitions for words contained within this report that were deemed as potentially unfamiliar to the reader

|  |  |
| --- | --- |
| Term | Definition in Current Context |
| Decode | To convert the jpeg image into a matrix of pixel values |
| RGB | The red, green, and blue intensities that comprise an image pixel |
| Grayscale | Grayscale images only contain intensity information and not color information [2] |

1.4 Scope

This document relates primarily to functional requirements and associated algorithmic design characteristics for the main program subsystem.

**2. Overall Description**

2.1 Product Perspective

The product perspective defines the main program subsystem’s relationship to the other subsystems in the system.

The GUI subsystem is designed to display program instruction to the user and to read in the user’s input. The desired input for the main program subsystem is two filenames: an input filename and an output filename. The subsystem performs several error-checks to ensure that two filenames have been entered as user input, and that the input file is of the jpeg type. Subsequent to opening the input file but prior to sending the data to the other subsystems for analysis, the main program subsystem decodes the pixel data from the jpeg image, and stores each pixel value into an array of appropriate size. The array is converted to grayscale then sent to the edge-detection subsystem and later the Hough transform subsystem for analysis. The pupil center coordinates and radius are returned to the main program subsystem as output data, which the main program then takes and writes to a pre-specified output file.

All subsystems in the larger system are software-based, and thus no integration with hardware systems was required for this project.

This system is a new, self-contained product that originated as the BIEN4290 final project.

2.2 Product Functions

1. Take, as user input, the filename of the jpeg image in which the pupil will be detected and the output filename to which the data will be written to
2. Read in the pixel data from the image using a publicly available decoder library
3. Error check to determine that two filenames have been entered as user input, and that the input file name has a .jpeg extension
4. Send data to the edge-detection algorithm and Hough transform algorithm for further processing
5. Receive the output data from the two algorithms listed in step 4, and write this data to the output file specified by the user

The main program is responsible for decoding a .jpeg file representing an image, storing the associated pixel values into pointer arrays, and passing the correct pointers to the algorithm functions written by other development teams. First, the main program must if check the number of arguments the function received when called (not counting the main being called) from the GUI is two, as the program requires both input and output filenames. If not, ‘-1’ is returned to the GUI. If two arguments were received, the next error-checking mechanism is to determine if the input filename has a .jpeg extension. If the file cannot be opened or is not a jpeg, the main program returns ‘-1’ to the GUI. If a jpeg file has been successfully opened, the jpeg is then decoded and its size in bytes is recorded, which is then translated to height and width in pixels. As the pixel elements of the image are decoded, they are also stored in an array of appropriate size. The RGB array is then converted to a grayscale image. The decoded array is subsequently sent to the edge detection algorithm and then to the Hough transform algorithm. After the image is processed, the output file is opened and the pupil center coordinates and radius returned to the main function are written. Upon termination of the program, the opened files are closed and the space allocated for the arrays is deallocated. The program then returns ‘0’ to the GUI.

2.3 User Classes

User classes for the main program subsystem include the development team for testing purposes and the BIEN4290 professor and TA for grading purposes.

2.4 Operating Environment

The subsystem and associated system is designed to run on a Windows computer running the Linux operating system.

**3. External Interface Requirements**

3.1 Software Interfaces

The main program subsystem interacts with three other software-based subsystems: the GUI subsystem, the Hough transform subsystem, and the edge-detection subsystem. The interaction occurs via calling the subsystems and transferring input/output data between them. All algorithms, with the exception of the GUI, will be written in the C++ programming language, and thus the system integration will come through the inclusion of header files into the different subsystems. As specified prior in the document, the input and output file names will be transferred to the main subsystem from the GUI subsystem, and the array containing the grayscale pixel values will be sent to the Hough-transform and edge-detection algorithms. The pupil center coordinates and radius returned to the main program subsystem will be written to the output filename that the user entered into the GUI. See appendix A below to see the flow of data between the main program subsystems and the other subsystems.

**4. System Features**

Please note that all system features for the main program subsystem are defined as “high priority” because the main program is what directs the entire system.

4.1: User input data

4.1.1: Description and priority

The main program will take as user input the filename of the jpeg image in which the pupil will be detected and the output filename to which the data will be written to. This information will come from the GUI subsystem. This feature is defined as a high priority.

4.1.2: Stimulus/Response Sequences

GUI triggers the main program -> main program reads in the data passed by the GUI subsystem

4.1.3: Functional Requirements:

REQ-1: subsystem is executable and able to be called by the GUI subsystem

REQ-2: subsystem can accept parameters and successfully read in information passed by the GUI subsystem

REQ-3: subsystem can open files provided a filename

4.2: Decode jpeg image

4.2.1: Description and priority

The main program will use a publicly available decoder library to decode the jpeg image into associated pixel values. This feature is considered a high priority.

4.2.2: Stimulus/Response Sequences

Input filename successfully opened -> decoder function is called and resulting pixel values are stored into an array

4.2.3: Functional Requirements:

REQ-4: the decoder has proper functionality

4.3: Call other functions/subsystems

4.3.1: Description and priority

The main program calls the other two subsystems (edge-detection and Hough transform) and passes them the array of pixel data. This feature is considered a high priority.

4.3.2: Stimulus/Response Sequences

EOF is reached in the decoded jpeg file (i.e, all pixel values have been stored into array) -> edge-detection and Hough transform subsystems are called and passed the pixel data array

4.3.3: Functional Requirements:

REQ-5: subsystem is able to call other functions and pass parameters to those functions

4.4: Write output data

4.4.1: Description and priority

The main program writes the pupil center coordinates and radius value to the output filename specified by the user. This feature is defined as a high priority.

4.4.2: Stimulus/Response Sequences

All functions have been called -> output data passed back from those functions is written to the output file specified by the user

4.4.3: Functional Requirements:

REQ-6: subsystem needs to be able to receive parameters passed by other functions

REQ-7: subsystem needs to be able to open an output file

**5. Other Nonfunctional Requirements**

5.1 Safety Requirements

There are not real definable risks involved with the creation of the main program software.

**5.2 Security Requirements**

As the system will be completely offline and the information being processed is not confidential or personal, the system does not need to have any security requirements at this time. If the system were to ever be adapted for a real-time, online system, security measures would then need to be put in place to protect the users.

**5.3 Software Quality Attributes**

The following attributes relate to the quality of the software developed in this stage.

* **Code Comments.** To ensure complete understanding of code between subgroups and future developers, comments will be documented along with the code. Comments will be included at the beginning of any program naming the authors, dates and brief description of the program’s functionality. Furthermore, at the start of each class, function and/or when deemed necessary, comments will be added to describe, in more detail, the functionality to follow.
* **Readability.** The readability of the code will also determine how easily the project can be understood by the differing subgroups and by potential future developers. To ensure readability, well-supported programming languages (C++) will be used to code the system.

**5.4 Performance Requirements**

REQ-8: The analysis objective of the current project requires that the pixel values decoded as part of system feature 4.2 be converted from the RGB values they were decoded as into grayscale values

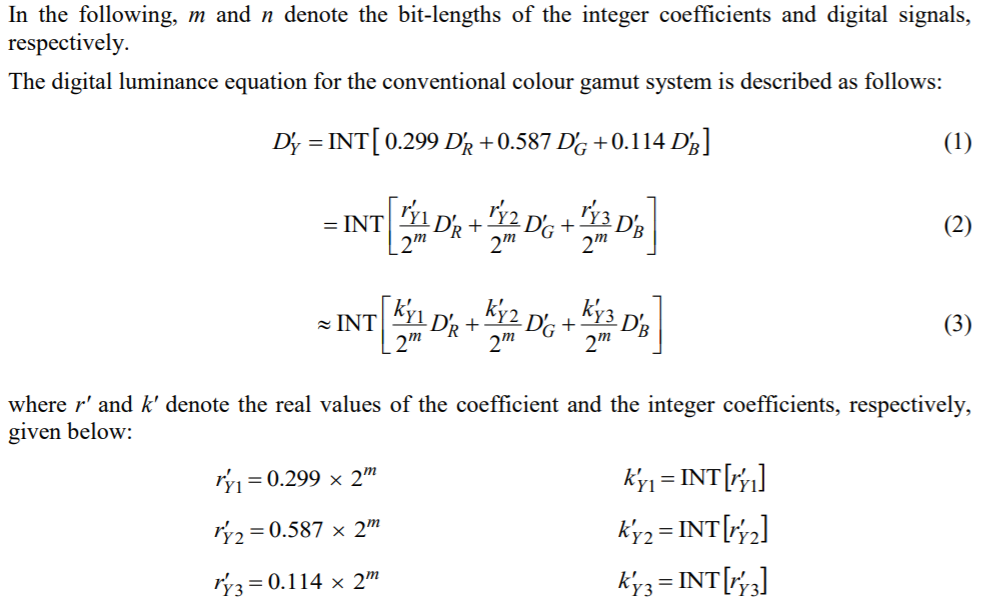
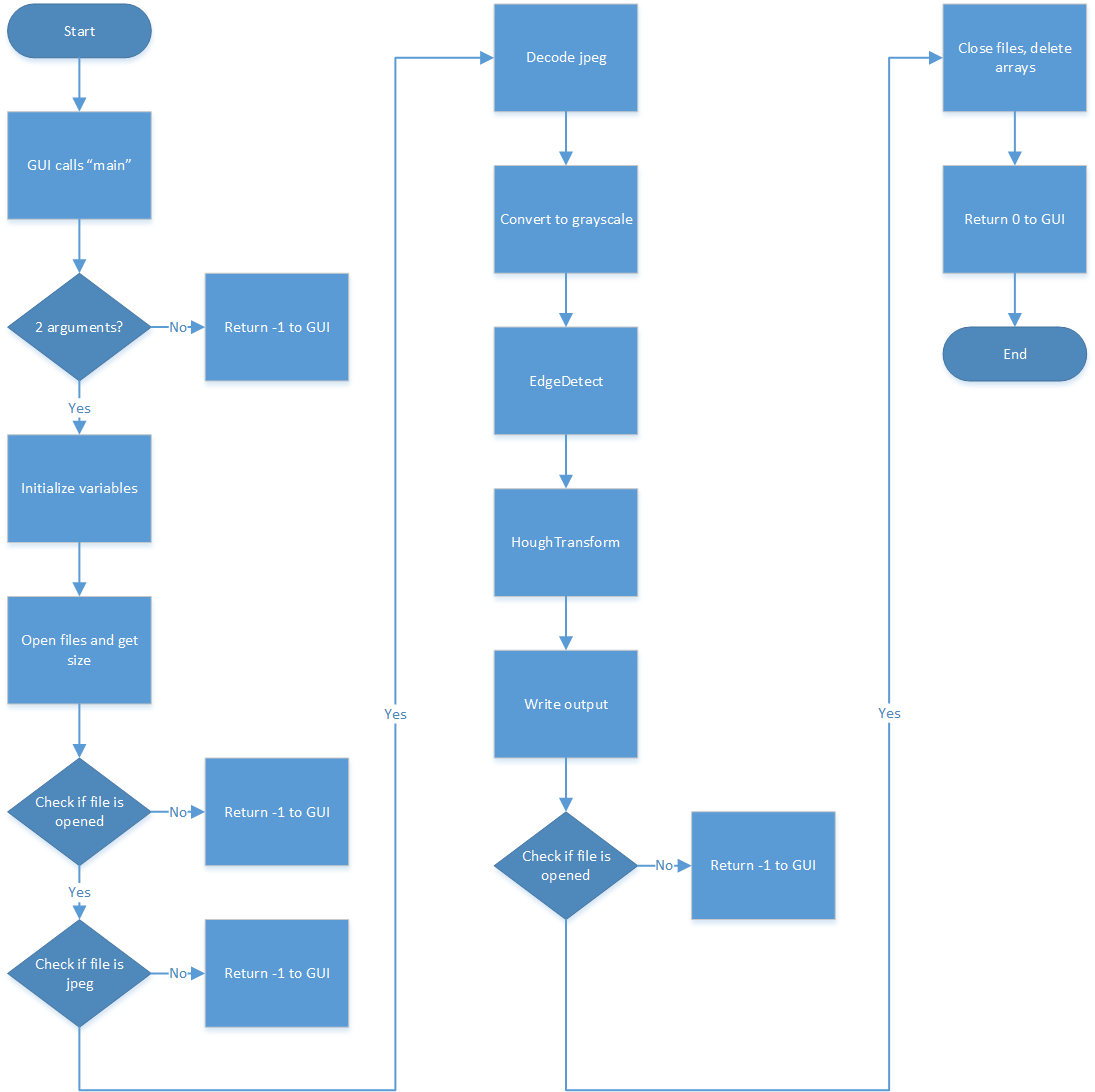
****

Figure 1: Luminance equations from the International Telecommunications Union Rec. ITU-R BT.601-7. The D’Y is the grayscale pixel value, while the D’R, D’G, and D’B are the RGB values. [3]

**6. References**

1. The BIEN4290 D2l webpage
   1. The Project Manual
   2. Main\_Program\_Instructions document
   3. Srs\_template-ieee
2. <https://www.techopedia.com/definition/7468/grayscale>
3. <https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf>

**Appendix A: Design of Main Program Subsystem**

****

**Appendix B: Test Plan**

**Preliminary Validation Test Plan**

Below is the preliminary validation test plan for the main program of the final project. The Test Table includes nine different aspects of the main program, also highlighted in the flowchart above, that will need to be tested and deemed PASS in order for the program to meet functionality requirements. As the tests are run, the Tester Name/Date and Actual Result columns will be populated with appropriate data. Upon completion of all tests located in the Test Table, and anymore deemed necessary during the validation process, the Validation Test Summary table will be updated. This table provides a concise look into the Validation Test Plan allowing for surface level determination as to if the main program appears functional or not. More in depth detail can then be found within the, yet to be filled in, contents of the Test Table. Tests 1.1, 1.3, 1.5, 1.6, based upon the inputs will be easier to discern functionality as it is a test of will the program run, provide proper error message to the user, or crash when given these inputs. As the tests are attempted and errors are encountered the plan will have ADDED sections with new types of tests needed to prove validity. Test 1.2 will utilize print statements and manual checking to determine accurate sizing. Test 1.4 will utilize the newly created pixel element array and either manually compare to expected results or run the input file through a separate, simpler JPEG decoding call outside of our program to check the validity of the output. Test 1.7 will use the processed output file and again, with help from the Hough Transformation and Edge Detection teams, ensure that our output matches what is appropriate output. Test 1.8 will likely utilize a third-party tool such as Valgrind to properly monitor the deallocation of memory within the main program. Test 1.9 will be run by printing the return value, ensuring it is a 0, and proceeding back to the GUI. Finally, a Revision History section is provided to track any revisions made to the Preliminary Validation Test Plan.

**Validation Test Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Total Number of Test Cases** | **Total Number of Test Cases Executed** | **Total Number of Test Cases That Passed** | **Total Number of Test Cases That Failed** |
| 9 | 0 | 0 | 0 |

**Test Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID/Description** | **Input/Expected Output** | **Tester Name/Date** | **Actual Result** |
| 1.1 Check that the number of variables is correct (2) | **Call main() with two arguments**  OK:   * Program accepts arguments and runs accordingly   ERROR:   * Program does not accept arguments   **Call main() with less or more than 2 arguments**  OK:   * Program refuses to accept main() call   ERROR:   * Program accepts main() call | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.2 Check that the input file is correctly opened and the file size is determined. | **Input file of size X**  OK:   * File is properly opened and determined to be of size X   ERROR:   * File is not properly opened * File is properly opened but the size is determined incorrectly or not at all | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.3 Checks the program accurately discerns between acceptable (.jpeg) files and non acceptable (.\*) files | **Input file of type .jpeg**  OK:   * Program appropriately accepts the .jpeg file   ERROR:   * Program does not accept the .jpeg file   **Input file of type .\***  OK:   * Program does not accept the .\* file   ERROR:   * Program accepts the .\* file | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.4 Check that the program has successfully decoded the .jpeg file | **Input file of type .jpeg**  OK:   * Correctly decoded pixel elements are stored in array   ERROR:   * Pixel elements not properly decoded * Pixel elements not stored properly in array | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.5 Check that the edgeDetection() is called | **edgeDetection()**  OK:   * Function is run and returns to main() after completion   ERROR:   * Function isn't run * Function is run but does not return to main() after completion | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.6 Check that the houghTransform() is called | **houghTransform()**  OK:   * Function is run and returns to main() after completion   ERROR:   * Function isn't run * Function is run but does not return to main() after completion | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.7 Check that the processed image data is properly written to an output file | **Processed image array**  OK:   * output file is opened and the pupil center coordinates and radius returned to the main function are written   ERROR:   * Output file is not opened * Output file is opened but the center coordinates and radius are not returned * Output file is opened and the center coordinates and radius are returned incorrectly | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.8 Check that after necessary data is extracted that the file is closed and the memory is deallocated | **Processed image array**  OK:   * File is closed and memory is freed   ERROR:   * File is not closed and memory is not freed * File is closed but memory is not freed * File is not closed and memory is freed | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |
| 1.9 Checks main() then returns to the GUI after completion of data processing, analysis, and freeing of memory | **Return 0**  OK:   * main() then returns to the GUI   ERROR:   * main() does not return to the GUI   **Return \*other\***  OK:   * main() does not return to the GUI   ERROR:   * main() does return to the GUI | <Name>  <MM/DD/YYYY> | PASS:  FAIL: |

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Date** | **Name** | **Revision** |
| <MM/DD/YYYY> | <Name> | <Description> |