|  |
| --- |
| EyeSim-VR Validation Report |
|  |
|  |
| **System Version: 1.0** |
|  |

#### Author: EyeSim-VR Team

Table of Contents

Purpose 2

System Description and Scope 2

Validation Approach and Criteria 3

Documentation Validation 3

Functionalities Validation 6

Non-functional Requirements Validation 19

Review and Approval Signatures 23

Appendices Test Code 24

# Purpose

The purpose of this validation plan is to review the various activities involved in EyeSim-VR project to ensure all necessary documentations are in place, all requirements as specified by our client are met. User training is provided to ensure that it is properly installed and used. Validation of all functionalities is accomplished through the user validation testing that is executed by the users of the application.

# System Description and Scope

EyeSim-VR is a multiple mobile robot simulator with VR functionality based on game engine Unity 3D that allows experiments with the same unchanged EyeBot programs that run on the real robots. EyeSim VR is capable of simulating all major functionalities in RiBIOS-7, including:

* LCD Output/Key Input
* Camera output
* PSD Sensors
* Servos and Motors
* V-Omega Driving
* Radio Communication

User can build 3D customised simulation environment using a world or maze file (in .wld or .maz extension), place any number of different kind of robots (in the provided robot models), and also add different kind of objects (like cans or soccer balls) to the simulation. Due to Unity's excellent physics engine, the simulation of the motion of robots and the interactions between robots and objects/walls can be more realistic and accurate, which considerably improves the quality of simulation. To make simulations more realistic, user can even add errors to the simulation (because real robots aren't perfect) using the simulated error function we provide.

# Validation Approach and Criteria

The validation of the system is divided into documentation validation, functional requirements validation, and non-functional requirements validation. The validation approach and criteria are listed below.

|  |  |  |
| --- | --- | --- |
| Validation Items | Criteria | Validation Approach |
| Documentations | In place; correct | Documentation Review  Interview (client, users) |
| Functional Requirements | Functions can be performed as expected, free from bugs | User validation testing |
| Non-functional Requirements | Non-functional requirements should be met | Interview users  User validation testing |

# Documentation Validation

To ensure that necessary documentations are in place and the content is correct and up-to-date, following detailed validations were performed for each document.

|  |  |  |
| --- | --- | --- |
| Documentation Name | System Requirements Document and  Prototype | |
| Author | EyeSim-VR team | |
| Description | Highlights system scope, requirements (functional and non-functional) and prototype. | |
| Completion Date | 2nd June 2017 | |
| Content | * Client, mentor and team members * Functional and Non-functional requirements * Use cases * Prototype screenshots | |
| Validations | | |
| Documentation Review | Criteria | * Documentation provides the required content * Documentation is nicely formatted and easy to read |
| Validated Method | * Internal peer review |
| Validated by | * EyeSim-VR team |
| Requirements Validation | Criteria | * Requirements are clearly presented and Use cases are correct and easy to understand * Requirements are from Client's real intention |
| Validated Method | * Internal peer review * Interview |
| Validated by | * EyeSim-VR team * Dr Thomas Braunl |

|  |  |  |
| --- | --- | --- |
| Documentation Name | EyeSim-VR User Manual | |
| Author | EyeSim-VR team | |
| Description | Describes in detail on the installation and usage of the simulator | |
| Completion Date | 14th September 2017 | |
| Content | * System requirements * Installation procedures * Procedures to perform various functionalities * Bug reporting | |
| Validations | | |
| Documentation Review | Criteria | * Documentation provides the required content * Documentation is nicely formatted and easy to read |
| Validated Method | * Internal peer review |
| Validated by | * EyeSim-VR team |
| Installation Procedures | Criteria | * Procedures are correct and valid |
| Validated Method | * Internal testing, tested on team member's laptop and lab computers. * Tests by Robotics Students of GENG5508 in their lab sessions either on lab computers or their own laptops. |
| Validated by | * EyeSim-VR team * GENG5508 Students |
| Functionalities | Criteria | * Procedures to perform functionalities should be valid and indicative screenshots and icons are correct. |
| Validated Method | * Internal testing, tested on team member's laptop and lab computers. * Tests by Robotics Students of GENG5508 in their lab sessions either on lab computers or their own laptops. |
| Validated by | * EyeSim-VR team * GENG5508 Students |

|  |  |  |
| --- | --- | --- |
| Documentation Name | EyeSim-VR Website (http://robotics.ee.uwa.edu.au/EyeSim/) | |
| Author | EyeSim-VR team | |
| Description | High level description of the simulator, and provide links for user manual, simulator software and other resources. | |
| Completion Date | 14th September 2017 | |
| Content | * High level introduction * System requirements * Download links * Basic installation and setup instructions | |
| Validations | | |
| Website Review | Criteria | * The website provides the required content * The website is nicely formatted and easy to navigate |
| Validated Method | * Internal peer review |
| Validated by | * EyeSim-VR team |
| Basic Installation procedures | Criteria | * Procedures are correct and valid |
| Validated Method | * Internal testing, tested on team member's laptop and lab computers. * Tests by Robotics Students of GENG5508 in their lab sessions either on lab computers or their own laptops. |
| Validated by | * EyeSim-VR team * GENG5508 Students |
| Download Links | Criteria | * All links should work without "dead link" problems |
| Validated Method | * Internal testing, tested on team member's laptop and lab computers. * Tests by Robotics Students of GENG5508 in their lab sessions either on lab computers or their own laptops. |
| Validated by | * EyeSim-VR team * GENG5508 Students |

Comments: The required documents and website have been created by EyeSim-VR team and validated internally (through peer review) and externally (either by Client or end users) and revised accordingly.

# Functionalities Validation

To validate the functionalities of the EyeSim-VR simulator, following test cases have been used, each testing and validating one or multiple functionalities. Each test case has been performed multiple times on both lab computers and laptops.

|  |  |
| --- | --- |
| Test Case Number | Test Case 1 |
| Description | Test world loading, scene viewing, robot placing, moving, rotating and deleting |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Steps | 1. Start EyeSim-VR simulator 2. Load a world file/maze file in the simulator 3. Add a robot in the scene 4. Drag and move the robot to three other locations 5. Rotate the robot 90,180 and 360 degrees using "+" and "-" keys. 6. Inspect the robot parameters by double clicking on the robot 7. Click on the pause button, and edit the (x,y) coordinates of the robot and parameter of rotation, then click resume button. 8. Delete the robot |
| Expected Results | 1. Simulator should be started 2. Simulator can build the environment according to the world/maze file selected. 3. Robot can be added in the scene 4. Robot can be dragged using mouse to other valid locations 5. Robot can rotate with the pressing of "+" and "-" keys 6. Inspector window should pop up showing PSD sensors readings, camera captures and coordinates of the robot. 7. The location and rotation parameters can be edited and the position of the robot will change accordingly 8. Target robot can be deleted |
| Testing Results | As expected |
| Tested Features | * World Loading * Scene Viewing * Robot Placing * Robot Moving & Rotating * Robot Parameter Viewing * Robot Parameter Editing * Robot Deletion |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Test Case Number | Test Case 2 |
| Description | Test object placing, moving, rotating and deleting |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Steps | 1. Start EyeSim-VR simulator 2. Add an Object (can or soccer ball) in the scene 3. Drag and move the object to three other locations 4. Rotate the robot 90,180 and 360 degrees using "+" and "-" keys. 5. Inspect the object parameters by double clicking on the object 6. Click on the pause button, and edit the x,y coordinates of the object and parameter of rotation, then click resume button. 7. Delete the object |
| Expected Results | 1. Simulator should be started 2. The object can be added in the scene 3. Object can be dragged using mouse to other valid locations 4. Object can rotate with the pressing of "+" and "-" keys 5. Inspector window should pop up showing (x,y) coordinates of the object and the rotation degree. 6. The location and rotation parameters can be edited and the position of the object will change accordingly 7. Target object can be deleted |
| Testing Results | As expected |
| Tested Features | * Object Placing * Object Moving & Rotating * Object Deletion |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Test Case Number | Test Case 3 |
| Description | Test the basic driving functions |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * motor.c |
| Steps | 1. Start EyeSim-VR simulator 2. Compile and run the test code in terminal 3. Click on "Go" buttons in simulated LCD window |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. The robot should drive forward at full motor power |
| Testing Results | As expected |
| Tested Features | * Basic Driving |
| Tested Functions | * MOTORDrive |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 4 |
| Description | LCD and image processing functions. |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * rgb.c * img.c |
| Steps | 1. Start EyeSim-VR simulator 2. Compile and run the test code in terminal 3. Click on different buttons in simulated LCD window |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. The robot should run according to the script and pressed button |
| Testing Results | As expected |
| Tested Features | * LCD Window * Image Processing |
| Tested Functions | * LCDImageStart * LCDImage * LCDPrintf * LCDSetPrintf * LCDImageGray * IPSetSize * IPWriteFile * IPReadFile |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 5 |
| Description | LCD functions. |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * graphics.c * fonts.c |
| Steps | 1. Start EyeSim-VR simulator 2. Compile and run the test code in terminal 3. Click on different buttons in simulated LCD window |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. The robot should run according to the script and pressed button |
| Testing Results | As expected |
| Tested Features | * LCD Window |
| Tested Functions | * LCDMenu * LCDSetColor * LCDClear * LCDPixel * LCDPixelInvert * LCDLine * LCDLineInvert * LCDArea * LCDAreaInvert * LCDCircle * LCDCircleInvert * LCDSetFont * LCDSetFontSize |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 6 |
| Description | Test Velocity/Omega Driving functions |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * speed.c * drivedemo.c * drive.c |
| Steps | 1. Start EyeSim-VR simulator 2. Place a robot in the scene 3. Compile and run each of the test code in terminal 4. If there's command in the script to show the LCD window, click on different buttons in simulated LCD window |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. A simulated LCD should show up with 4 buttons. 4. The robot should run according to the script and pressed button |
| Testing Results | As expected |
| Tested Features | * Velocity/Omega Driving |
| Tested Functions | * VWSetSpeed * VWGetPosition * VWSetPosition * VWStraight * VWTurn * VWWait * VWDone |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 7 |
| Description | Test servo function |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * servo.c |
| Steps | 1. Start EyeSim-VR simulator 2. Place a robot in the scene 3. Compile and run the test code in terminal |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. An LCD window should show up with the image captured by the camera, and the camera will pan and then tilt with the captured image changing accordingly. |
| Testing Results | As expected |
| Tested Features | * Servo Movement |
| Tested Functions | * SERVOSet |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 8 |
| Description | Camera captures |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * graycol.c |
| Steps | 1. Start EyeSim-VR simulator 2. Place a robot in the scene 3. Compile and run the test code in terminal |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. LCD window should show up with the captured image in gray, user can change the image to color by clicking on the LCD button. |
| Testing Results | As expected |
| Tested Features | * Camera Capture |
| Tested Functions | * CAMGet * CAMGetGray |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 9 |
| Description | Camera captures and image processing & LCD functions |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * hsi.c |
| Steps | 1. Start EyeSim-VR simulator 2. Place a robot in the scene 3. Compile and run the test code in terminal |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. An LCD window will show up with 6 different styles of captured image, representing "Color, Gray, Binary - Hue, Sat Intensity". |
| Testing Results | As expected |
| Tested Features | * Camera Capture * Image Processing * LCD Window |
| Tested Functions | * CAMInit * CAMRelease * IPCol2Gray * IPCol2HSI * LCDImageBinary |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 10 |
| Description | Test PSD function |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * psd.c |
| Steps | 1. Start EyeSim-VR simulator 2. Place a robot in the scene 3. Compile and run the test code in terminal |
| Expected Results | 1. Simulator should be started 2. The robot should be placed in the scene 3. The robot should show an LCD window showing the PSD readings of the robot. |
| Testing Results | As expected |
| Tested Features | * PSD infrared Position Sensors |
| Tested Functions | * PSDGet |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 11 |
| Description | Test camera movement, physics, simulation speedup and slowdown, simulation pause and resume. |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * drive-straight.c |
| Steps | 1. Start EyeSim-VR simulator 2. Place a robot in the lower end of the scene 3. Adjust the simulation camera using "w,s,a,d" keys. 4. Click on the Camera-> Birdseye View button to have a birds-eye view 5. Click on the Camera-> Reset Camera button to reset camera 6. Place a can in front of the robot 7. Double click on the robot 8. In the popped up inspector window click on the "select control" button 9. Navigate and select the test code file 10. During the simulation, click on the pause button to pause the simulation 11. Click on the play button to resume the simulation 12. On the robot inspector window, click on the add trace button to add trace. 13. During the simulation, click on the speedup button to speedup the simulation 14. Click on the play button to slowdown the simulation 15. Watch the interaction between robot and the can when the robot hit the can. |
| Expected Results | 1. Simulator should be started 2. A robot should be added to the scene 3. Camera should be adjusted accordingly 4. We can see a birds-eye view of the simulation 5. The camera view point should return to normal 6. A can should be added to the scene 7. A robot inspector window should pop up 8. A file selector should pop up for selection of code file 9. The robot should move forward as commanded in the code. 10. The simulation should pause 11. The simulation should resume 12. A green line will appear indicating the trace of the robot 13. The simulation should run at twice the speed 14. The simulation should run at normal speed 15. The can should be knocked over by the robot representing a physical interaction |
| Testing Results | As expected |
| Tested Features | * Physics * Simulation Speedup * Simulation Slowdown * Simulation Pausing/Resuming * Camera Movement * Client Loading * Add Trace |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

|  |  |
| --- | --- |
| Test Case Number | Test Case 12 |
| Description | Test camera movement, physics, simulation speedup and slowdown, simulation pause and resume. |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Test Code1 | * test.sim * dirve.c * ping.c |
| Steps | 1. Start EyeSim-VR simulator 2. Select test.sim from simulator Load Sim menu 3. Use the add error menu in simulator to select type of error and add error to simulation 4. Use Oculus VR headset to control camera of robot during the simulation and control the main camera 5. With the robots in the scene, compile and run code ping.c in terminal |
| Expected Results | 1. Simulator should be started 2. 2 robots will be placed at specific spot and facing direction and 3. Error will be added to the simulation according to the selected type 4. We can use VR headset to control and view both the robot camera and main camera 5. The robots should show an LCD windows showing their Ids and receive each other's id |
| Testing Results | As expected |
| Tested Features | * Add Error * Radio Control * VR Functions * Simulation File Batch Script |
| Tested Functions | * RADIOSend * RADIOInit * RADIOGetID * RADIOStatus * RADIOReceive |
| Tested By | EyeSim-VR team |

1 Detailed code can be seen in appendix

Following table shows all the available features of the EyeSim-VR and the corresponding validation test cases.

|  |  |
| --- | --- |
| **Features** | **Validation Test Cases** |
| Simulator Functionality | |
| World Loading | Test Case 1 |
| Robot Placing | Test Case 1 |
| Robot Moving & Rotating | Test Case 1 |
| Robot Deletion | Test Case 1 |
| Client Loading | Test Case 11 |
| Object Placing | Test Case 2 |
| Object Moving& Rotating | Test Case 2 |
| Object Deletion | Test Case 2 |
| Physics | Test Case 11 |
| Simulation Speedup | Test Case 11 |
| Simulation Slowdown | Test Case 11 |
| Simulation Pausing/Resuming | Test Case 11 |
| Add Trace | Test Case 11 |
| Add Error | Test Case 12 |
| Robot Functionality | |
| Basic Driving | Test Case 3 |
| PSD infrared Position Sensors | Test Case 10 |
| Velocity/Omega Driving | Test Case 6 |
| Camera Capture | Test Case 8, Test Case 9 |
| Servo Movement | Test Case 7 |
| Radio Control | Test Case 12 |
| LCD window | Test Case 4, Test Case 5, Test Case 9 |
| Image Processing | Test Case 4, Test Case 9 |
| User Interface | |
| Scene Viewing | Test Case 1 |
| Camera Movement | Test Case 11 |
| Robot Parameter Viewing | Test Case 1 |
| Robot Parameter Editing | Test Case 1 |
| Object Parameter Viewing | Test Case 2 |
| Object Parameter Editing | Test Case 2 |
| Virtual Reality | |
| Robot Camera VR Controlling & Viewing | Test Case 12 |
| Main Camera VR Controlling & Viewing | Test Case 12 |
| External Features | |
| Simulation File Batch Script | Test Case 12 |

# Non-functional Requirements Validation

|  |  |
| --- | --- |
| Requirement | **Security** |
| Description | Providing security to source code |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Requirements/Steps | 1. Source code should be kept in client’s private Git Repository. 2. Client gives EyeSim-VR team access to his private git repository 3. Final EyeSim product will not have any personal or important information. |
| Expected Results | 1. EyeSim-VR team gets source code access 2. Source code can only be changed physically or logically by EyeSim- VR team or client 3. Unauthorised people cannot access the private git repository account. 4. Other students will not be able to make changes to source code or product. |
| Testing Results | As expected |
| Tested Aspects | * Git access * Making changes to code to alter the EyeSim environment |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Requirement | **Compatibility** |
| Description | Checking compatibility of operating systems for all students to have easy access to the EyeSim simulator |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Requirements/Steps | 1. Install Cygwin, X11 and unity application on windows 2. Install Xcode, Xquartz and unity application on Mac OS |
| Expected Results | 1. Students are able to run EyeSim Simulator on Windows 2. Students are able to run simulator on mac. |
| Testing Results | As expected |
| Tested Aspects | * Installation process on both mac and on windows. |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Requirement | **Usability** |
| Description | Checking user interface standards |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Requirements/Steps | 1. EyeSim installation process must be completed in few clicks and should be easy for students who have no coding experience. 2. To be cross compatible for all the operating systems and ease of use of environment, code must be placed in a single file and will be able to run EyeSim simulator through batch commands. 3. Instructions are given at every step of robot movements and functions. |
| Expected Results | 1. Students are able to run simulator environment by clicking on single file. 2. User interface should be standard and easy for non-coders. Instructions or guidelines helps users to install the environment. |
| Testing Results | As expected |
| Tested Aspects | * Installation process on both mac and on windows. * Opening EyeSim simulator through single file |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Requirement | **Performance** |
| Description | Checking product performance |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Requirements/Steps | 1. EyeSim helps to run multiple robots in real time in the same environment by placing code in the real-time environment. 2. Robot has different functions to perform and each is saved in different files. So, each program has many number of threads. EyeSim helps to run different functions at same time without any time lag. 3. EyeSim allows robot to navigate in the environment |
| Expected Results | 1. Students can run multiple robots on simulator environment by clicking on its specific functions. |
| Testing Results | As expected |
| Tested Aspects | * Time lag when adding, deleting and running multiple robots on the simulator. |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Requirement | **Reliability** |
| Description | Checking reliability of the product |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Requirements/Steps | 1. Product is hosted publicly on university website and can be installed on lab computers and on peer laptops as well. 2. We can handle small errors by restarting product without any information loss as we have not provided any important information. 3. In case of bugs rise in the system, they can be immediately emailed or notify EyeSim-VR team through Bugzilla application. |
| Expected Results | 1. Students can easily install EyeSim simulator 2. Students can restart the simulator many time with no information loss 3. Bugs can be notified and rectified easily 4. Product is can be installed on windows, mac and Linux operating systems. It is cross compatible |
| Testing Results | As expected |
| Tested Features | * Cross compatibility and reinstalling product with no information loss |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Requirement | **Upgradability** |
| Description | Checking upgradability of the product |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Requirements/Steps | 1. Source code is written and stored in editable text files in order to help future developers to extend the functionalities of the product. |
| Expected Results | 1. Future developers will be able to add new features to product. |
| Testing Results | As expected |
| Tested Features | * Tested on newly added features of simulator environment (Wall and floor colors of maze file) at the end of project |
| Tested By | EyeSim-VR team |

|  |  |
| --- | --- |
| Requirement | **Implementation** |
| Description | Product implementation |
| System Environment | Windows, Mac OS and Linux |
| Hardware Environment | Computers and laptops |
| Requirements/Steps | 1. Testing is done on each robot functionality. 2. Bugzilla application is provided to student to report bugs 3. Once the testing is done, Product is ready to implement. 4. Need to get client approval of the product |
| Expected Results | 1. Many of the bugs are identified and solved 2. Client approves the final product 3. Product is launched with installation guidelines 4. EyeSim-VR team will be available to fix the bugs in later stages |
| Testing Results | As expected |
| Tested Features | * Simulation environment and each functionality of robot |
| Tested By | EyeSim-VR team |

Comments: The non-functional requirements are also crucial part of the requirements of the EyeSim VR project. During the second half year of the project, major proportion of our efforts is allocated to the validations of these requirements both by infernal testing, and in the student labs, feedbacks from students are acquired from Bugzilla bug reporting system to facilitate our validations and improvements of the product.

# Review and Approval Signatures

The signatures below indicate that all the above described tests and validations conform to the requirements as specified in the System Requirements Documentation, and the EyeSim simulation has been validated to met the functional and non-functional requirements of client.

Approved by:

|  |  |  |
| --- | --- | --- |
| Signature | Initials | Date |

# Appendices Test Code

In the sequence of the code showing up in this document:

Motor.c

#include "eyebot.h"

#include <stdio.h>

#include <math.h>

int main ()

{ int k, x,y,phi;

LCDMenu("GO","BACK","CIRCLE","END");

do

{ switch(k = KEYGet())

{ case KEY1: MOTORDrive(1, 100);

MOTORDrive(2, 100);

break;

case KEY2: VWSetSpeed(-300, 0); break;

case KEY3: VWSetSpeed(+300, 90); break;

}

OSWait(2000); // 1 sec

VWSetSpeed(0,0); // stop

VWGetPosition(&x, &y, &phi);

LCDPrintf("x=%d y=%d p=%d\n", x,y,phi);

} while(k != KEY4);

}

**RJB.c**

// EyeBot Demo Program: Image File I/O, T. Bräunl, June 2015

#include "eyebot.h"

int main()

{ BYTE img[QVGA\_SIZE];

int pos;

for (int i=0; i<50; i++)

for (int j=0; j<320; j++)

{ pos = 3\*(320\*i + j);

img[pos]=0; img[pos+1]=0; img[pos+2]=0;

}

for (int i=50; i<100; i++)

for (int j=0; j<320; j++)

{ pos = 3\*(320\*i + j);

img[pos]=255; img[pos+1]=0; img[pos+2]=0;

}

for (int i=100; i<150; i++)

for (int j=0; j<320; j++)

{ pos = 3\*(320\*i + j);

img[pos]=0; img[pos+1]=255; img[pos+2]=0;

}

for (int i=150; i<200; i++)

for (int j=0; j<320; j++)

{ pos = 3\*(320\*i + j);

img[pos]=0; img[pos+1]=0; img[pos+2]=255;

}

for (int i=200; i<240; i++)

for (int j=0; j<320; j++)

{ pos = 3\*(320\*i + j);

img[pos]=255; img[pos+1]=255; img[pos+2]=255;

}

IPSetSize(QVGA);

IPWriteFile("pic/rgb.ppm", img);

LCDImageStart(0,0, 320,240);

LCDImage(img);

LCDPrintf("Black - Red - Green - Blue - White");

OSWait(5000); // 5s

}

Img.c

// EyeBot Demo Program: Image File I/O, T. Bräunl, June 2015

#include "eyebot.h"

int main()

{ BYTE img[QVGA\_SIZE];

IPSetSize(QVGA);

LCDImageStart(0,10, 320,240);

LCDSetPrintf(0,0, "IMAGE 1: Color");

IPReadFile("pic/image1.ppm", img);

LCDImage(img);

OSWait(2000); // 2s

LCDSetPrintf(0,0, "IMAGE 2: Gray");

IPReadFile("pic/image2.pgm", img);

LCDImageGray(img);

OSWait(2000); // 2s

LCDSetPrintf(0,0, "IMAGE 3: Binary");

IPReadFile("pic/image3.pbm", img);

LCDImageGray(img); // same function as Gray pgm

OSWait(2000); // 2s

}

Graphics.c

#include "eyebot.h"

char \*text = "The quick brown fox jumps over the lazy dog. 1234567890";

#define ColorConstNum 17

int TestLCDSetColor(COLOR fg, COLOR bg)

{

static int ColorCount = 0;

printf("LCDClear()\n");

LCDClear();

printf("LCDMenu()\n");

LCDMenu("One", "Two", "Three", "Four");

printf("LCDSetColor(%08x, %08x), %d/%d\n", (int)fg, (int)bg, ++ColorCount, ColorConstNum);

LCDSetColor(fg, bg);

printf("LCDPrintf(text)\n");

LCDPrintf(text);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

return(0);

}

// int TestLCDMenuI(int pos, char\* string, COLOR fg, COLOR bg)

// {

// printf("Testing LCDMenuI() ...\n")

// printf("LCDClear()\n");

// LCDClear();

// printf("LCDMenuI()\n");

// LCDMenuI(pos, string, fg, bg);

// printf("LCDPrintf(text)\n");

// LCDPrintf(text);

// printf("KEYWait(ANYKEY)\n\n");

// KEYWait(ANYKEY);

// return(0);

// }

int main(void)

{

int x=0, y=0;

printf("LCDMenu()\n");

LCDMenu("One", "Two", "Three", "Four");

printf("LCDPrintf(text)\n");

LCDPrintf(text);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("LCDClear()\n");

LCDClear();

printf("LCDMenu()\n");

LCDMenu("One", "Two", "Three", "Four");

printf("LCDSetPos(3,5)\n");

LCDSetPos(3,5);

printf("LCDPrintf(text)\n");

LCDPrintf(text);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

// printf("LCDClear()\n");

// LCDClear();

// printf("LCDSetColor(RED, GREEN)\n");

// LCDSetPos(RED, GREEN);

// printf("LCDPrintf(text)\n");

// LCDPrintf(text);

// printf("KEYWait(ANYKEY)\n\n");

// KEYWait(ANYKEY);

// LCD

// TestLCDSetColor(RED, GREEN);

// TestLCDSetColor(GREEN, BLUE);

// TestLCDSetColor(BLUE, WHITE);

// TestLCDSetColor(WHITE, GRAY);

// TestLCDSetColor(GRAY, BLACK);

// TestLCDSetColor(BLACK, ORANGE);

// TestLCDSetColor(ORANGE, SILVER);

// TestLCDSetColor(SILVER, LIGHTGRAY);

// TestLCDSetColor(LIGHTGRAY, DARKGRAY);

// TestLCDSetColor(DARKGRAY, NAVY);

// TestLCDSetColor(NAVY, CYAN);

// TestLCDSetColor(CYAN, TEAL);

// TestLCDSetColor(TEAL, MAGENTA);

// TestLCDSetColor(MAGENTA, PURPLE);

// TestLCDSetColor(PURPLE, MAROON);

// TestLCDSetColor(MAROON, YELLOW);

// TestLCDSetColor(YELLOW, OLIVE);

// TestLCDSetColor(OLIVE, RED);

// LCDSetMode() should work properly, skipping

printf("LCDClear()\n"); // Whether LCDClear() resets the Pos and Color or not?

LCDClear();

printf("LCDMenu()\n");

LCDMenu("One", "Two", "Three", "Four");

printf("LCDPrintf(text)\n");

LCDPrintf(text);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

// LCDMenuI

printf("Testing LCDMenuI() ...\n");

printf("LCDClear()\n");

LCDClear();

printf("LCDMenuI()\n");

LCDMenuI(0, "GREEN One", GREEN, BLUE);

printf("LCDMenuI()\n");

LCDMenuI(1, "BLUE Two", BLUE, WHITE);

printf("LCDMenuI()\n");

LCDMenuI(2, "WHITE Three", WHITE, RED);

printf("LCDMenuI()\n");

LCDMenuI(3, "RED Four", RED, GREEN);

printf("LCDPrintf(text)\n");

LCDPrintf(text);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

// // LCDGetSize

// printf("LCDClear()\n");

// LCDClear();

// printf("LCDMenu()\n");

// LCDMenu("One", "Two", "Three", "Four");

// printf("LCDGetSize(&x, &y)\n");

// LCDGetSize(&x, &y);

// printf("Result: x=%d, y=%d\n", x, y);

// printf("KEYWait(ANYKEY)\n\n");

// KEYWait(ANYKEY);

// LCDPixel

printf("LCDClear()\n");

LCDClear();

printf("LCDMenu()\n");

LCDMenu("One", "Two", "Three", "Four");

printf("LCDPixel(50, 50, WHITE)\n");

LCDPixel(50, 50, WHITE);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

// printf("LCDClear()\n");

// LCDClear();

// printf("LCDMenu()\n");

// LCDMenu("One", "Two", "Three", "Four");

printf("LCDPixelInvert(50, 50)\n");

LCDPixelInvert(50, 50);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("LCDLine(0, 0, 200, 100, GREEN)\n");

LCDLine(0, 0, 200, 100, GREEN);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("LCDLineInvert(0, 100, 200, 0)\n");

LCDLineInvert(0, 100, 200, 0);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

// int fill

printf("LCDArea(50, 50, 100, 120, BLUE, 0)\n");

LCDArea(50, 50, 100, 120, BLUE, 0);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("LCDArea(50, 50, 100, 120, BLUE, 1)\n");

LCDArea(50, 50, 100, 120, BLUE, 1);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("LCDAreaInvert(60, 60, 120, 100)\n");

LCDAreaInvert(60, 60, 120, 100);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

// LCDCircle

printf("LCDCircle(140, 140, 50, ORANGE, 0)\n");

LCDCircle(140, 140, 50, ORANGE, 0);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("LCDCircle(140, 140, 50, ORANGE, 1)\n");

LCDCircle(140, 140, 50, ORANGE, 1);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("LCDCircleInvert(150, 150, 20)\n");

LCDCircleInvert(150, 150, 20);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

printf("KEYXY(&x, &y)\n");

KEYGetXY(&x, &y);

printf("Result: x=%d, y=%d\n", x, y);

printf("KEYWait(ANYKEY)\n\n");

KEYWait(ANYKEY);

//

// printf("LCDClear()\n");

// LCDClear();

// printf("LCDMenu()\n");

// LCDMenu("One", "Two", "Three", "Four");

// printf("KEYWait(ANYKEY)\n\n");

// KEYWait(ANYKEY);

//

printf("Exiting normally ...\n");

return(0);

}

Fonts.c

#include "eyebot.h"

#include <unistd.h>

int main() {

LCDPrintf("Standard default font\n");

LCDSetFont(HELVETICA, NORMAL);

LCDPrintf("Printing in Helvetica Normal\n");

LCDSetFont(HELVETICA, BOLD);

LCDPrintf("Printing in Helvetica Bold\n");

LCDSetFont(TIMES, NORMAL);

LCDPrintf("Printing in Times Normal\n");

LCDSetFont(TIMES, BOLD);

LCDPrintf("Printfing in Times Bold\n");

LCDSetFont(COURIER, NORMAL);

LCDPrintf("Printing in Courier Normal\n");

LCDSetFont(COURIER, BOLD);

LCDPrintf("Printing in Courier Bold\n");

LCDSetFont(HELVETICA, NORMAL);

LCDSetFontSize(14);

LCDPrintf("Increasing font size to 14\n");

sleep(2);

}

Speed.c

/\*------------------------------------------------------------------------

| Filename: speed.c

| Author: Thomas Braunl UWA 2017

| Description: Drive using SetSpeed functions

-------------------------------------------------------------------------- \*/

#include "eyebot.h"

#include <stdio.h>

#include <math.h>

int main ()

{ int k, x,y,phi;

LCDMenu("GO","BACK","CIRCLE","END");

do

{ switch(k = KEYGet())

{ case KEY1: VWSetSpeed(+300, 0); break;

case KEY2: VWSetSpeed(-300, 0); break;

case KEY3: VWSetSpeed(+300, 90); break;

}

OSWait(1000); // 1 sec

VWSetSpeed(0,0); // stop

VWGetPosition(&x, &y, &phi);

LCDPrintf("x=%d y=%d p=%d\n", x,y,phi);

} while(k != KEY4);

}

Drive-demo.c

#include "eyebot.h"

#include <stdio.h>

#include <math.h>

#define functions 9

char fname[functions][32]=

{"Forward", "Backward", "Rotate Left", "Rotate Right",

"Curve Left\n(FORWARD)", "Curve Right\n(FORWARD)", "Curve Left\n(BACKWARD)",

"Curve Right\n(BACKWARD)", "SetPos [0,0,0]"};

//velocities

int vel[functions][2] =

{ { 300, 0}, {-300, 0}, { 0, 30}, {0, -30},

{ 300, 30}, { 300, -30}, {-300, 30},

{-300,-30}, { 0, 0} };

int main (){

int x, y, phi, v, w;

int fnum = 0, done = 0;

v = 0; w = 0;

do {

LCDClear();

LCDMenu("+", "-", "GO", "END");

LCDPrintf("%s\n", fname[fnum]);

VWGetPosition(&x, &y, &phi);

LCDPrintf("x = %d \n", x);

LCDPrintf("y = %d \n", y);

LCDPrintf("p = %d \n", phi);

v = vel[fnum][0];

w = vel[fnum][1];

LCDPrintf("v=%d, w=%d\n", v, w);

switch(KEYGet()) {

case KEY1: fnum = (fnum+1) % functions;

break;

case KEY2: fnum = (fnum-1 +functions) % functions;

break;

case KEY3: if (fnum<8)

{ VWSetSpeed(v,w);

LCDMenu(" ", " ", "STOP", " ");

KEYWait(KEY3); // continue until key pressed

}

else if (fnum==8)

VWSetPosition(0,0,0);

break;

case KEY4: done = 1;

break;

}

VWSetSpeed(0,0); // stop

} while (!done);

return 0;

}

Drive.c

// EyeBot Demo Program: Drive, T. Bräunl, Nov. 2015

#include "eyebot.h"

int main()

{ for (int i=0; i<2; i++) // run twice

{ VWStraight(500, 10); // 0.5m in ca. 5s

//{ VWStraight(500, 100); // 0.5m in ca. 5s

while (!VWDone())

if (PSDGet(2) < 100) VWSetSpeed(0,0); // STOP if obstacle in front

VWTurn(180, 60); // half turn (180 deg) in ca. 3s

VWWait(); // wait until completed

}

}

Servo.c

#include "eyebot.h"

void checkpos(int pan, int tilt)

{ BYTE img[QVGA\_SIZE];

SERVOSet(1, pan);

SERVOSet(2, tilt);

LCDSetPrintf(0,0,"PanTlt: %3d %3d\n", pan, tilt);

CAMGet(img);

LCDImage(img);

}

int main()

{ int pos;

CAMInit(QVGA);

for (pos=128; pos<255; pos++) checkpos(pos, 128);

for (pos=255; pos>0; pos--) checkpos(pos, 128);

for (pos=0; pos<128; pos++) checkpos(pos, 128);

for (pos=128; pos<255; pos++) checkpos(128, pos);

for (pos=255; pos>0; pos--) checkpos(128, pos);

for (pos=0; pos<128; pos++) checkpos(128, pos);

return 0;

}

Graycol.c

// EyeBot Demo Program: Camera Interactive, T. Bräunl, Nov 2015

#include "eyebot.h"

int main()

{ BYTE img[QVGA\_SIZE];

int k, size = 0, gray = 1;

LCDMenu("SIZE", "COL", "", "END");

CAMInit(QQVGA); // automatically sets LCDIMageSize and IPSize

do

{ k = KEYRead();

if (k==KEY1)

{ size = !size;

LCDClear();

if (size) CAMInit(QVGA);

else CAMInit(QQVGA);

}

else if (k==KEY2) {

gray = !gray;

}

if (gray)

{ CAMGetGray(img);

LCDImageGray(img);

LCDMenu("SIZE", "COL", "", "END");

}

else // color

{ CAMGet(img);

LCDImage(img);

LCDMenu("SIZE", "COL", "", "END");

}

} while (k != KEY4);

return 0;

}

Hsi.c

// EyeBot Demo Program: Display color, gray, binary -- hue, sat, intensity

#include "eyebot.h"

#define SIZE QVGA\_SIZE

#define PIX (SIZE/3)

#define XS 160

#define YS 120

#define X 80

#define Y 60

#define D 5

#define MID (3\*(Y\*XS + X))

int main()

{ BYTE image[SIZE];

BYTE h[PIX], s[PIX], i[PIX], g[PIX], b[PIX];

CAMInit(QQVGA);

LCDMenu(" ", " ", " ", "END");

LCDSetPrintf(20,0, "Color, Gray, Binary - Hue, Sat Intensity");

do

{ CAMGet(image);

LCDImageStart( 0, 0, 160,120);

LCDImage(image);

IPCol2Gray(image, g); // make gray

LCDImageStart(160, 0, 160,120);

LCDImageGray(g);

for (int i=0; i<PIX; i++) b[i] = (g[i]<127); // make bin

LCDImageStart(320, 0, 160,120);

LCDImageBinary(b);

IPCol2HSI(image, h, s, i); // disect HSI

LCDImageStart( 0,120, 160,120);

LCDImageGray(h);

LCDImageStart(160,120, 160,120);

LCDImageGray(s);

LCDImageStart(320,120, 160,120);

LCDImageGray(i);

} while (KEYRead() != KEY4);

CAMRelease();

return 0;

}

Psd.c

#include "eyebot.h"

#define PSD\_LEFT 1

#define PSD\_FRONT 2

#define PSD\_RIGHT 3

int main()

{

int left, front, right;

do{

LCDRefresh();

left = PSDGet(PSD\_LEFT);

front = PSDGet(PSD\_FRONT);

right = PSDGet(PSD\_RIGHT);

VWSetSpeed(200,0);

OSWait(200);

LCDPrintf("Left:%d,Front:%d,Right:%d",left, front,right);

} while(front>200);

VWSetSpeed(0,0); //stop

}

Drive-straight.c

// EyeBot Demo Program: Drive, T. Bräunl, Nov. 2015

#include "eyebot.h"

int main()

{ LCDPrintf("Drive straight\n");

VWStraight(1000, 200); // 1m in ca. 5s

VWWait(); // wait until completed

LCDPrintf("Rotate\n");

VWTurn(180, 60); // half turn in ca. 3s

VWWait(); // wait until completed

LCDPrintf("Drive straight\n");

VWStraight(1000, 200); // 1m in ca. 5s

VWWait(); // wait until completed

LCDPrintf("Rotate\n");

VWTurn(180, 60); // half turn in ca. 3s

VWWait(); // wait until completed

}

Test.sim

#comment

world "/Users/tomzhangle/Desktop/RoBotVR/RobotVR/SavedWorld.wld"

# botname x y phi

LabBot 200 200 0 "/Users/tomzhangle/Desktop/drive.x"

LabBot 500 200 90 "/Users/tomzhangle/Desktop/drive.x"

Ping.c

// Ping-Pong radio communication program

// T. Braunl, May 2017

#include "eyebot.h"

#define MAX 10

int main ()

{ int k, i, num, ret;

int id[256];

int myid, partnerid;

BYTE buf[MAX] = "00000";

LCDMenu("MASTER", "SLAVE", "", "END");

RADIOInit();

myid = RADIOGetID();

LCDPrintf("my id %d\n", myid);

k = KEYGet();

if (k==KEY4) return 0; // exit

if (k==KEY1) // master only

{ LCDPrintf("scanning (takes time ...)\n");

ret = RADIOStatus(id);

if (ret<0) LCDPrintf("error RADIOStatus\n");

partnerid = id[0];

LCDPrintf("partner is %d\n", partnerid);

LCDPrintf("I will start\n");

RADIOSend(partnerid, buf);

}

LCDPrintf("I am waiting for partner\n");

for (i=0; i<10; i++)

{ RADIOReceive(&partnerid, buf, MAX);

LCDPrintf("received from %d text %s\n", partnerid, buf);

sscanf (buf, "%d", &num);

num++;

sprintf(buf, "%05d", num);

RADIOSend(partnerid, buf);

}

KEYWait(KEY4);

}