**Software Engineering I**

Course Number: CS 07321 1

Instructor: Dr. Adrian Rusu

**Fall 2014**

**Team #1**

**December 15th, 2014**

**Robo-Ops Competition**

**Testing Document**

**NASA/Department of Mechanical Engineering**



**Team Members:**

**Daniel Bittner**

**Edward Carter**

**Jun He**

**Matthew Ng**

**Dakota Pollitt**

**Zhentao Zhong**

Testing Summary

This document defines the final implementation testing of Rover software. All the functional and structural testing are included with test cases and short descriptions of their purpose. Any errors found during the testing phase will be corrected through exception handling or the addition of further checks.

The functional testing was designed to cover two of the main functions of the final software: A successful connection between Command and the Rover and the transfer of control specific information to the Rover. Due to the lack of a functional prototype, the modules tested here will need slight updates to operate the final Rover’s Arduino board.

The third main function of the software is the display of video feeds transmitted from the Rover to Command. The success of a video transmission is evident, so instead we are testing the success of switching the active camera displays. Due to the lack of a prototype the command may need to be modified to synchronize with the final Rover’s cameras. The video feeds currently transmits successfully to a webcam.

Summary of functional testing results

The Command and Rover PC’s established a successful connection and began transmitting data from the neutral controller. Each movement of the joystick successfully printed it’s intended data on the Rover’s Console. The Command PC also displayed video feeds from the Rover’s Camera’s. A small number of minor bugs were found in their transmission during testing. However, they have been corrected and will not cause further problems for our software. The software handles controller input well but the camera streams seems to have a short delay when transferring images from the Rover to the Command. The delay is not long enough for the rover to suffer much in efficiency and more images can be sent if the network allows it.

Summary of structural testing results

The code was unable to handle the absence of a camera for display, and exception handling was added to prevent a fatal crash. When only one feed was available, both displays played the same camera feed. When multiple feeds were available, the displays could successfully switch between the available feeds independently.

**Functional Testing for Connection and Joystick Controls**

**Basis:** Verify successful connection

**Description:** Test initial connection between the Rover and Command Communications. All other tests are dependent upon this success.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INPUT** | **CORRECT** | **INVALID** | **EXPECTED OUTPUT** | **NOTES** |
| Rover and Command Client Connect | X |  | “Client Accept” | The console will print a success message upon connection initialization |

**Basis:** Send and receive motor control commands correctly

**Description:** Test that vital control information is successfully transmitted between the Rover and Command PC. This data will be used in the execution of Arduino code in further versions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TYPE** | **INPUT** | **EXPECTED OUTPUT** | **ACTUAL OUTPUT** | **NOTES** |
| Move Forward | Joystick pushed forward | 90 | 90 |  |
| Move Backward | Joystick pulled backward | -90 | -90 |  |
| Move Left | Joystick shifted left | 0 | 0 |  |
| Move Right | Joystick shifted right | 180 | 179..-179 | Joystick doesn’t read exactly 180, cycles from 179 to -179 |
| Set Speed -  Neutral | Throttle set in middle position | 0 | 0 |  |
| Set Speed -  Forward | Throttle pushed completely forward | 45 | 45 |  |
| Set Speed -  Backward | Throttle pulled completely backward | -45 | -45 |  |

**Input Domain**: List<Webcam> camList

**Basis:** Number of cameras

**Description:** Test the GUI ability to display camera feed for different numbers of cameras

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INPUT** | **CORRECT** | **INVALID** | **EXPECTED OUTPUT** | **NOTES** |
| 0 Cameras |  | X | No camera feeds | ArrayIndexOutOfBoundsException |
| 1 Camera |  | X | Same camera feed in each display | ArrayIndexOutOfBoundsExceptionException |
| 2 Camera | X |  | Separate camera feeds in each display |  |
| 3+ Camera | X |  | Separate camera feeds in each display |  |

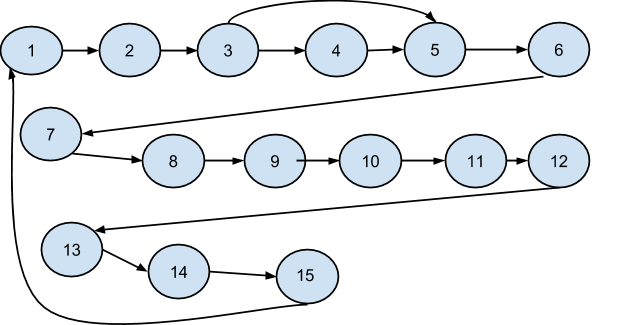
**Structural Testing for User Client**

Code To Connect A Socket To The Input IP Address

**Description:** The User client attempts to create a socket based on the input IP address.

Server Address String Test Cases : {“” , “150.250.221.2” , “1.1.1.1” }

|  |
| --- |
| 1. while (!done) {  2. try {  3. if (serverAddress.equals("")) {  4. throw new UnknownHostException();  5. }  6. s = new Socket(serverAddress, 9090);  7. done = true;  8. } catch (ConnectException e) {  9. gui.display("Waiting for server...");  10. } catch (UnknownHostException ex) {  11. gui.display("An Unknown Host Exception has occurred...");  12. } catch (IOException ex) {  13. Logger.getLogger(UserClient.class.getName()).log(Level.SEVERE, null, ex);  14. }  15. } |



Test Case {“”} : [1,2,3,4,10,11] - Program reports that an UnknownHostException has occured

Test Case {“150.250.221.2”} : [1,2,3,5,6,7,14,15] - Program connects to the Rover Client

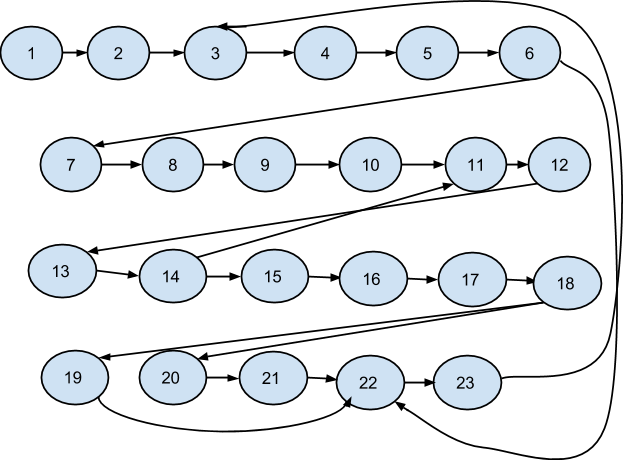
Test Case {“1.1.1.1”} : [1,2,3,5,6,8,9 ] - Program continuously cycles through a ConnectException until the program is terminated manually

Code To Read Camera Pictures From The Socket Stream

**Description**: The program continuously reads the input stream for data until it is disconnected

Test Cases : {Empty stream, Bytes in Stream for first camera, Bytes in Stream for second camera}

|  |
| --- |
| 1. DataInputStream dataIn = new DataInputStream(s.getInputStream());  2. InputStream inStream = s.getInputStream();  3. while(!disconnect)  4. {  5. int available = inStream.available();  6. if(available>0)  7. {  8. short cameraNum = dataIn.readShort();  9. int len = dataIn.readInt();  10. byte[] bytes = new byte[len];  11. for(int i=0;i<len;i++)  12. {  13. bytes[i] = dataIn.readByte();  14. }  15. BufferedImage bImage;  16. ByteArrayInputStream in = new ByteArrayInputStream(bytes);  17. bImage = ImageIO.read(in);  18. if(cameraNum==0)  19. { gui.setIcons(bImage,null);}  20. else  21. { gui.setIcons(null,bImage);}  22. }  23. } |



Test Case : {Empty stream} : [1,2,3,4,5,6, 21 ] : The program continually cycles through checking for available bytes until disconnect

Test Case : {Bytes in Stream First Camera} : [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,22] : The program processes all the bytes into an image and then sets the gui image for the first camera

Test Case : {Bytes in Stream Second Camera} : [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,20,21,22] : The program processes all the bytes into an image and then sets the gui image for the second camera