*ecse 211 design project*

Testing Document

Version *1.08*

*04/05/2018*

*ECSE 211 TEAM 11*

VERSION HISTORY

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Title** | Testing Document | | | |
| **Description** | Test the built system to make sure it meets all the requirements | | | |
| **Created By** | Tianyi Zou, Testing leader | | | |
| **Date Created** | 1st March 2018 | | | |
| **Version Number** | **Modified By** | **Modifications Made** | **Date Modified** | **Status** |
| 1.00 | Tianyi Zou | Created the Testing Document Template | 1st March | Preliminary version of the document;  added testing template in the appendix |
| 1.01 | Luka Jurisic | Peer reviewed the document. Fixed some small errors and formatted the document. Added the introduction, 2 appendixes, and the test plan document. Created section 1.1-1.3.2 and section 2 | 2nd March | Preliminary template complete |
| 1.02 | Tianyi Zou, Enan Zaman | Completed section 4.2 and 4.4; Light Sensor and Wheels preference tests | 13th March | All other tests remain |
| 1.03 | Tianyi Zou | Completed section 4.1 | 20th March | All other tests remain |
| 1.04 | Volen | Completed 5.2 and 5.3 | 22nd March | All other tests remain |
| 1.05 | Luka Jurisic | Following criticisms in the 3rd weekly meeting, changes regarding test procedures were updated and corrected. Also Added testing process. Table of contents presentation was improved as well as the Testing Template | 26th March | Outstanding tests remain |
| 1.06 | Luka Jurisic,  Tianyi Zou | Continued to improve the document through procedural correction. Tianyi added section 4.3 | 29th March | Software testing has been corrected fully. However, many tests are still required. In hardware, the motor test section has not been completed, but this might be removed following a team discussion. |
| 1.07 | Enan Ashaduzzaman | Enan-Added subparts to section 5 | 4th April | Localization still needs to be tested fully. Software lag is limiting this test. Following that, capture testing needs to tested. Otherwise, testing is on schedule for the final demo. |

TABLE OF CONTENTS

# 1 INTRODUCTION

1.1 Purpose of the Test Plan Document

1.2 Testing Tools ……………………………………………………………………………

1.3 Quality Objective ……………………………………………………………………

1.3.1 Primary Objective ……………………………………………………………

1.3.2 Secondary Objective…………………………………………………………………

1.4 Testing Process   
 1.4.1 Hardware Components  
 1.4.2 Software Components  
 1.4.3 Integration Testing

# 2 TEST DELIVARABLES

# 3 Test report TEMPLATES

# 4 Hardware Testing

4.1 Ultrasonic Sensor ………………………………………………………………………

4.2 Light Sensor …………………………………………………………………………

4.3 Track……………………………………………………………………………

4.4 Wheels…………………………………………………………………………

4.5 Robot Structure…………………………………………………………………………

# 5 Software TESTING

5.1 Localization……………………………………………………………………………

5.2 Back wheel Functionality…………………………………………………………….

5.3 Navigation……………………………………………………………………………

5.4 Obstacle Traversal ………………………………………………………………………

5.5 Search area …………………………………………………………………………

5.6 Capture …………………………………………………………………………

5.7 Obstacle avoidance ………………………………………………………………………

5.8 Wi-Fi Integration…………………………………………………………………………

# 6 INTEGRATION testing

# Introduction

## Purpose of The Test Plan Document

The document documents and tracks the necessary information required to effectively define the approach to be used in the testing of the project’s product. The document is created during the Planning Phase of the project. Its intended audience is the project manager, project team, and testing team. Some portions of this document may on occasion be shared with the client/user.

**1.2 TESTING TOOLS**

The following tools will be used for testing:

|  |  |
| --- | --- |
| PROCESS | TOOLS |
| Test Case Creation | Microsoft Word |
| Test Case Tracking | Microsoft Excel |
| Test Case Execution | Manual |
| Test Case Management | Microsoft Excel |
| Defect Management | Microsoft Excel |

# 1.3 Quality objective

# 

# 1.3.1 Primary Objective

The primary objective of this testing phase is to assure that the system meets the full requirements, including quality requirements, and maintain the metrics for each quality re-equipment of the final design. At the end of the project development, the user should find that the project has met or exceeded all their specifications detailed in the requirements.

# 1.3.2 secondary Objective

The secondary objective of this testing phase is to identify issues and propose solutions to all hardware and/or software issues, and to communicate all this to the project team. This requires careful and methodical testing of the design to ensure all areas of the system are scrutinized appropriately.

# 1.4 TESTING PROCESS

# 1.4.1 Hardware cOMPONENTS

# The hardware components utilized in the final design were separately tested to clearly identify weaknesses and allow a swift evaluation of performance. For example, at our disposal were 6 ultrasonic sensors, and tests were run solely on these sensors to determine which one was most accurate. This procedure was also performed for the light sensors and the motors. Motor testing was especially critical as there were two types of motors available, and a decision had to be made as to which type would be implemented into the final design.

# Similarly, precise values were obtained of both the wheel radius and wheelbase length, which are key for the software department to have and utilize.

# 1.4.2 software components

# The testing of software components follows on from its respective hardware testing, as an initial or even progressive design is required before any meaningful software testing phase can commence. Once again, key classes that governed the robot’s main behaviors were tested in isolation to fully assess their performance and flaws.

# Regarding software, the primary objective is to realize any flaws within the architecture. This requires that the robot be tested in a varying number of unfamiliar and unorthodox situations to ensure full functionality of the code. For example, localization has to be tested many times because the starting orientation of the robot is unknown, and thus as many cases as possible must be accounted for during this testing phase.

# 1.4.3 Integration testing

# Software and hardware testing ensures that each respective individual component performs accordingly. However, when all the components are working in tandem, this gives rise to problems previously unforeseen. The accumulation of small errors within each subsystem could result in a poor working final design. Thus, integration testing is the key final phase of testing that allows us to ensure that all the client’s behavioral specifications are met.

# TEST DELIVARABLES

The testing phase will allow a general progression of the project in terms of both hardware and software. The testing phase will provide key deliverables that fall into 3 basic categories: Documents, Test Cases and Reports. The figure below illustrates the dependencies of these 3 categories.



*Diagram Source: https://strongqa.com/qa-portal/testing-docs-templates/test-report*

# 3 Test report template

# 

***Test’s Title:*** *Name of Test*

**Tester:** *Concerned Team Members* **Test Date:** *Month/Day/Year*  **Software Version:** *Software version of Robot* **Hardware Version:** *Hardware version of Robot*

**Objective:** *What is the purpose of the test?*

**Background knowledge (if needed):** *Relevant information disclosed here*

**Procedure:** *Detailed description of the steps performed to carry out the test*

**Expected Results:** *What is the predicted outcome of the test?*

**Test Report:** *Provide the relevant data acquired during the test, along with an analysis of the results. The number of times the test was performed must be included. Tables and/or graphs are key in providing visual information. See appendix for table templates.*

**Conclusion:** *Specify the outcome of the test. Were the requirements met and did they meet the expected outcome?***Action:** *Based on the results, provide a list of what needs to done going forward.*

**Distribution:** *Indicate which sections of the team need to be made aware of the results of the test to allow for necessary modifications.*

# Hardware Testing

# 4.1 Ultrasonic Sensor

***Ultrasonic Sensor Consistency Test***

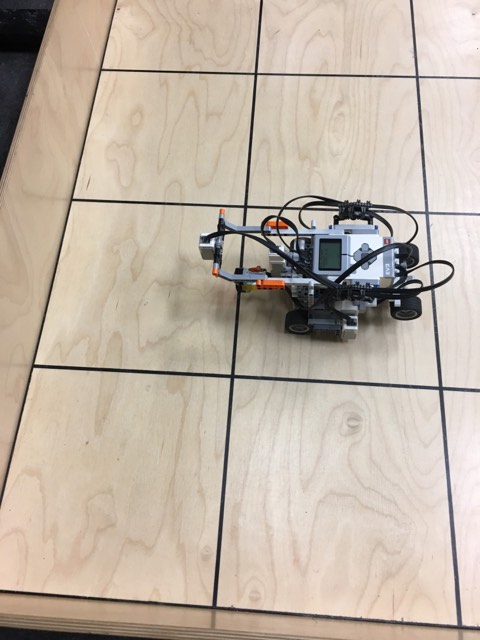
**Tester:** *Tianyi Zou* **Test Date:** *03/15/18* **Software Version:** *N/A\** **Hardware Version:** *N/A\**\**No hardware and software version due to the use of the built-in sensor testing application located in the tools option in the EV3 selection menu*

**Objective:**

Determine the accuracy of ultrasonic sensors available. This test will serve to identify which of the 3 ultrasonic sensors provided to our team have the least incongruities in their distance measurements. Thus, the most accurate and/or consistent ultrasonic sensors will be determined and implemented onto our final design.

**Procedure:**

1. Turn on the EV3 brick. Connect the Port 1 of the brick to the light sensor via a cable.
2. Assemble the ultrasonic sensor vertically in front of the robot. Make sure that the sensor should direct to the front of robot.
3. Use the tools application on the EV3 brick. Select **Tools>Test Sensors >Go> Port 1>EV3 Ultrasonic >Distance**.
4. Record the real distance from each tile and the measured value on the screen of brick. Please refer to the Figure 3.1.1.



*Ultrasonic Sensor*

*Ultrasonic Sensor*

*Figure 3.1.1 Placement of ultrasonic sensor during the test: 2 different views*

5. Calculate the average value and standard deviation of error that between real distance and measured distance.

6. Test other two ultrasonic sensors by using the same procedure. Compare the result from all different ultrasonic sensors.

**Expected result:**

We expect all the sensors would provide small inconsistencies when measuring distances and different ultrasonic sensor would perform differently. Some sensors would perform better than others, which give us a sense of which sensor to choose for object avoidance and localization.

**Test Report**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **US Sensor 1** | | **US Sensor 2** | | **US Sensor 3** | |
| **Real distance (cm)** | **Measured distance (cm)** | **Error (cm)** | **Measured distance**  **(cm)** | **Error (cm)** | **Measured distance**  **(cm)** | **Error (cm)** |
| 29.8 | 30.4 | 0.6 | 30.4 | 0.6 | 30.2 | 0.4 |
| 60.1 | 62.2 | 2.1 | 67.0 | 6.9 | 64.3 | 4.2 |
| 90.4 | 91.6 | 1.2 | 91.7 | 1.3 | 91.7 | 1.3 |
| 120.7 | 121.8 | 1.1 | 122.3 | 1.6 | 121.7 | 1.0 |
| 151.3 | 152.9 | 1.6 | 152.9 | 1.6 | 152.1 | 0.8 |
| 181.6 | 182.9 | 1.3 | 182.9 | 1.3 | 182.4 | 0.8 |
| 211.9 | 213.6 | 1.7 | 213.0 | 1.1 | 213.2 | 1.3 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Sensor 1** | **Sensor 2** | **Sensor 3** |
| **Average Error (cm)** | 1.37 | 2.06 | 1.40 |
| **Standard Deviation (cm)** | 0.48 | 2.16 | 1.27 |

**Conclusion**:

The average error of US sensor 1 is the lowest and it has the lowest standard deviation among three sensors. This means sensor 1 provides more accurate values in distances.

**Action**:

We use ultrasonic sensor 1 as the priority choice. If more than one ultrasonic sensor needs to be used, sensor 3 would be the second-best choice.

**Distribution**: Hardware team

# 4.2 LIGHT Sensor

***Light Sensor Consistency Test***

**Tester’s names:** *Enan Ashaduzzaman, Tianyi Zou* **Test Date:** *03/12/18* **Software Version:** N/A  
**Hardware Version:** N/A

**Objective:**

Determine the best performing light sensors available. This test will serve to identify which of the 3 light sensors provided has the clearest detection of an object from different heights.

**Procedure:**

1. Put a blue paper on the table. Use a ruler to measure the distance by putting a block adjacent to the ruler so that the ruler can be stabilized and placed perpendicular to the table surface.
2. Connect the Port 1 on the brick to the sensor via a cable.
3. Use the tools application on the EV3 brick. Select Tools>Test Sensors >Go> Port 1>EV3 Color >Color ID.
4. Place the light sensor next to the ruler and on the table surface. Make sure the direction of light should be to the blue paper on the table.
5. Move up the light sensor along the ruler slowly.
6. Record the distance *d1* between the light screen of light sensor and the table surface when the value of color ID shown on the screen of EV3 brick becomes 2.0.
7. Repeat step 5 and record the distance *d2* when the value of color ID becomes 7.0.
8. Repeat step 5 and record the distance *d3* when the value of color ID becomes -1.0.
9. Do the same procedure to test other two light sensors.

**Expected result:**

The value of *d1* and *d2* are respectively the closest and farthest distance that the light sensor is able to precisely detect an object, which means light sensor can both detect the object in front of it and identify the color of the object. Value of *d3* is the farthest distance that the light sensor can detect an object, but not able to identify the color.

**Test reports:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sensor # | *d1(cm)* | *d2(cm)* | *d3(cm)* |
| *Light Sensor 1* | 0.3 | 1.8 | 4.5 |
| *Light Sensor 2* | 0.6 | 1.0 | 3.5 |
| *Light Sensor 3* | 0.5 | 1.5 | 3.7 |

**Conclusion:**  
Comparing the average value differences of all 3 sensors for any given height, it can easily be seen that the differences between the sensors are minimal and cannot be detected with tests of this quality. However, there is no need for more precise testing, as this test shows that all 3 color sensors would get the job done equally well as they can each detect the black lines with at least one data point for each black line.

**Action:**   
We must place the sensor at distance between 0.6 cm and 1.0 cm from the block in order to detect the block and identify its color.

**Distribution:**Hardware team

# 4.3 TRACK

# 4.3.1 track 1.0

***Exact Wheelbase Length (Track Value) Test***

**Tester:** *Tianyi Zou* **Test Date:** *03/20/18* **Software Version:** *01.00.08 to 1.01.00*  
**Hardware Version:** *1.0*

**Objective:**

The objective of this test is to achieve a very precise value of the robot’s wheelbase length. This track value is essential in ensure proper navigation and thus a very accurate value is pivotal.

**Background Info:** By initially physically measuring the distance between the two front wheels, the starting test value is found to be 12.70cm.

**Procedure:**

1. Charge up the EV3 brick and make sure the robot works at a battery level of 8.0V.
2. Use the test track code that enables the robot to rotate about itself by 360 °. Set the rotating speed at 150 to avoid slippage of wheels.
3. Put the robot at a cross line on the testing board. Make sure that the axis of wheels and the middle line of robot are respectively right above the two lines of the cross. Please refer to the Fig3.3.1.
4. Run the code. It is expected that the robot will not rotate exactly by 360 °. There is an angle between the black line and the middle line of robot. Measure the angle by using a protractor.
5. If the robot turns more than 360 °, lower the value of track in the code and repeat step 2 and step3. If the robot turns less than 360 °, higher the value of track in the code and repeat step 2 and step 3.
6. Continue the process until a minimum amount of error is found.

**Expected Result**: The Research and Development phase of this project clearly indicated that the precise fine-tuning of an accurate track value is difficult, and it known to be off its true value. We expect that the robot will not turn exactly by 360 °. There would be an error of angle. If the track value we set in the code is greater than the real track, the robot will turn more. Conversely, if the track value is less than the real track, the robot will turn less. Past testing within this phase indicates that the angle errors of the track value is expected to be within the range of 0°± 3°. This error is primarily due to human error and the inexactitude of measurements.

**Test Report:**

|  |  |  |
| --- | --- | --- |
| Test # | Track Value/cm | Error in Angle/° |
| 1 | *12.70* | *+4.0* |
| 2 | *12.70* | *+4.1* |
| 3 | *12.65* | *+3.0* |
| 4 | *12.65* | *+2.9* |
| 5 | *12.64* | *+2.0* |
| 6 | *12.64* | *+2.0* |
| 7 | *12.63* | *+1.5* |
| 8 | *12.63* | *+1.4* |
| 9 | *12.62* | *+1.0* |
| 10 | *12.62* | *+0.9* |
| 11 | *12.61* | *+1.4* |
| 12 | *12.61* | *+1.4* |

**Conclusion:** Through trial and error, it has been determined that the most precise measurement of our track is 12.62cm. This value of track produces the least error when measuring the angle.

**Action**: Change the track to the experimentally determined valued of 12.62cm.

**Distribution:** Software Team

# 4.3.2 track 2.0

***Exact Wheelbase Length (Track Value) Test***

**Tester:** *Tianyi Zou* **Test Date:** *03/20/18* **Software Version:** *01.00.08 to 1.01.00*  
**Hardware Version:** *1.0*

**Objective:**

The objective of this test is to achieve a very precise value of the robot’s wheelbase length. This track value is essential in ensure proper navigation and thus a very accurate value is pivotal.

**Background Info:** By initially physically measuring the distance between the two front wheels, the starting test value is found to be 12.70cm.

**Procedure:**

1. Charge up the EV3 brick and make sure the robot works at a battery level of 8.0V.
2. Use the test track code that enables the robot to rotate about itself by 360 °. Set the rotating speed at 150 to avoid slippage of wheels.
3. Put the robot at a cross line on the testing board. Make sure that the axis of wheels and the middle line of robot are respectively right above the two lines of the cross. Please refer to the Fig3.3.1.
4. Run the code. It is expected that the robot will not rotate exactly by 360 °. There is an angle between the black line and the middle line of robot. Measure the angle by using a protractor.
5. If the robot turns more than 360 °, lower the value of track in the code and repeat step 2 and step3. If the robot turns less than 360 °, higher the value of track in the code and repeat step 2 and step 3.
6. Continue the process until a minimum amount of error is found.

**Expected Result**: The Research and Development phase of this project clearly indicated that the precise fine-tuning of an accurate track value is difficult, and it known to be off its true value. We expect that the robot will not turn exactly by 360 °. There would be an error of angle. If the track value we set in the code is greater than the real track, the robot will turn more. Conversely, if the track value is less than the real track, the robot will turn less. Past testing within this phase indicates that the angle errors of the track value is expected to be within the range of 0°± 3°. This error is primarily due to human error and the inexactitude of measurements.

**Test Report:**

|  |  |  |
| --- | --- | --- |
| Test # | Track Value/cm | Error in Angle/° |
| 1 | *16.59* | *+4.0* |
| 2 | *16.59* | *+4.1* |
| 3 | *12.65* | *+3.0* |
| 4 | *12.65* | *+2.9* |
| 5 | *12.64* | *+2.0* |
| 6 | *12.64* | *+2.0* |
| 7 | *12.63* | *+1.5* |
| 8 | *12.63* | *+1.4* |
| 9 | *12.62* | *+1.0* |
| 10 | *12.62* | *+0.9* |
| 11 | *12.61* | *+1.4* |
| 12 | *12.61* | *+1.4* |

**Conclusion:** Through trial and error, it has been determined that the most precise measurement of our track is 12.62cm. This value of track produces the least error when measuring the angle.

**Action**: Change the track to the experimentally determined valued of 12.62cm.

**Distribution:** Software Team

# 4.4 Wheels

***Wheel type: Treads vs Regular Wheels***

**Tester:** *Enan Ashaduzzaman & Tianyi Zou* **Test Date:** *03/12/18* **Software Version:** *N/A* **Hardware Version:** *N/A*

**Objective:**

The objective of this test is to determine whether to use the treads or the regular wheels. Treads can be very useful at overcoming the bumps and avoiding a variable track. Fears include treads not being accurate during navigation. This test serves to provide the necessary *qualitative* data so that the best form of transportation can be implemented on the final robot.

**Background knowledge:**

From past knowledge from the labs, it was seen that the robot completes the navigation of the square accurately when a marble is attached to the rear of the robot. Please note that design 1 and design 2 referred to in the test report below, refer to the initial 2 preliminary designs discussed in the hardware document.

The square driver from lab 3 codes the robot to drive in a square on a 4 by 4 platform.

**Procedure:**

1. Build a dummy robot using the tread
2. Implement the square driver code from lab 3 into the dummy robot. Tweak the track and wheel radius so it meets the standard of the dummy robot.
3. Make the robot complete the square navigation to see the accuracy using the square driver code.
4. Build a dummy robot using the regular wheels. Have a marble attached to the rear end of the robot.
5. Completes steps 2-4 using the second iteration of the robot.

**Expected Results:**

It is expected that the treads will be less accurate than the regular wheels during navigation. These little discrepancies can accumulate at the end of the day.

**Test Report:**

*Treads*

*Design 1:* The treads were loose on the robot since the wheels didn’t span the entire length of the tread. The robot completed the navigation through the bumps. The robot had a lot of accuracy issues during the square navigation.

*Design 2:* The front wheel of the tread was lifted slightly, tightening the tread on the robot. The robot had no difficulty traversing the bumps on the bridge. While the navigation got better from the first implementation, there were still some problems in the navigation of the robot.

Regular Wheels

*Design 1:* Having two wheels on each motor helped with the traction of the robot. The single marble holding the back end of the robot caused many issues when going through the bumps. It caused the robot to never travel straight. At the end, the robot never made it through the bridge during the tests. The navigation of the robot was more accurate than the navigation using treads.

*Design 2:* Having two marbles on the back end instead of one slightly helped the robot when traversing the bridge. It still encountered a lot of problems. It was concluded that the marble was not going to be a viable option for the robot.

**Conclusion:**It was concluded that the final robot would have the regular wheels implemented over the treads. Considering that navigation is a key component during the final project, it is important to use the hardware that completes the navigation the bests. Therefore, the regular wheels were chosen considering they were a ton better at navigation.

**Action:**The next step in the hardware process is to implement a landing gear. The landing gear will only be implemented when the robot is traversing the bridge.

**Distribution:**Hardware team

# SOFTWARE tESTING

# 5.2 LANDING GEAR TEST

***Test’s Title: Back Wheel Functionality***

**Tester’s names:** *Volen Mihaylov* **Test Date:** *03/22/18*

**Software Version:***00.00.00*

**Hardware Version:***1.0*

**Objective:**

The objective of this test is to determine the optimal angle at which to turn the variable back wheels of the robot. This test will serve to highlight the angles at which the back wheels rotate from a stored to lowered position and vice versa that cause no overturning of the motor. Please refer to the procedure for a definition of overturning.

**Background knowledge:** None

**Procedure:**

1. Place the robot parallel to any one black line on the grid.  
2. Ensure the back wheels are lowered.

3.Run the relevant version of the software code.  
4. Observe the robot’s performance, keeping note of how the back wheels overturn. Overturning in this case refers to the back wheels coming too much into contact with other hardware components, producing a “clicking” sound.

**Expected Results:** The robot is not expected to overturn/under turn at all during this test run. The back wheels should lock smoothly back into place.

**Results obtained:**

*Note: A positive angle refers to the back wheels moving from an up position(stored) to a lowered position. A negative angle refers to the back wheels moving from a lowered position to a stored position.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tests** | **Variables** | | | | **Passed?** | **Comments:** |
| Angle down | Angle Up |  |  |  |
| **1** | 270 | -270 |  |  | Yes | Overturning, clicking sound can be heard |
| **2** | 270 | -270 |  |  | Yes | Overturning, clicking sound can be heard |
| **3** | 270 | -270 |  |  | Yes | Overturning, clicking sound can be heard |
| **4** | 250 | -250 |  |  | Yes | No overturning sound, wheels well placed |
| **5** | 250 | -250 |  |  | Yes | No overturning sound, wheels well placed |
| **6** | 250 | -250 |  |  | Yes | No overturning sound, wheels well placed |

**Conclusion:** Utilizing a 250-degree angle within the software proves to be superior to an angle of 270 degrees. This is highlighted in the above test results since at 270 degrees the robot’s back wheels overturned during each test run. However, the issue could also lie within the hardware design of the robot. A

**Action:** As of now, implement 250-degree angle within the software. Following this test, a discussion needs to take place between the software and hardware teams to discuss the possibility of a hardware improvement.

**Distribution:** Hardware/Software Team

***Test’s Title: Back Wheel Functionality - With stopper***

**Tester’s names:** *Volen Mihaylov* **Test Date:** *03/22/18*

**Software Version:** *00.00.00*

**Hardware Version:** *1.01*

**Objective:** The objective of this test is to determine the optimal angle at which to turn the variable back wheels of the robot. This test will serve to highlight the angles at which the back wheels rotate from a stored to lowered position and vice versa that cause no slight overturning/under turning of the motor.

**Background knowledge:** This test is identical to the “Back Wheel Functionality test” except that in this case a **“stopper”** has been implemented within the hardware. The stopper serves to fully ensure that the motors don’t overturn. Also, the stopper acts as a support so that the robot can stand up on its back wheels.

**Procedure:**

1. Place the robot parallel to any one black line on the grid.  
2. Ensure the back wheels are lowered.

3.Run the relevant version of the software code.  
4. Observe the robot’s performance, keeping note of how the back wheels overturn. Overturning in this case refers to the back wheels coming too much into contact with other hardware components, producing a “clicking” sound.

**Expected Results:** The robot is not expected to overturn/under turn at all during this test run. The back wheels should lock smoothly back into place.

**Results obtained:***Note: A positive angle refers to the back wheels moving from an up position(stored) to a lowered position. A negative angle refers to the back wheels moving from a lowered position to a stored position.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tests** | **Variables** | | | | **Passed?** | **Comments:** |
| Angle down | Angle Up |  |  |  |
| **1** | 200 | -200 |  |  | No | Too much Rotation |
| **2** | 195 | -195 |  |  | No | Too little rotation, does not go all the way |
| **3** | 192 | -192 |  |  | No | Too little rotation, almost goes all the way but is missing a couple of degrees |
| **4** | 190 | -190 |  |  | No | Not enough rotation |
| **5** | 185 | -185 |  |  | Yes | Looks good |
| **6** | 180 | -180 |  |  | No | Too little rotation, almost goes all the way but is missing a couple of degrees |
| **7** | 180 | -180 |  |  | No | Too little rotation, almost goes all the way but is missing a couple of degrees |
| **8** | 180 | -180 |  |  | Yes | Perfect |

**Conclusion:** When implementing a stopper, a wheel rotation of 195 degrees proves to be the most smooth and well-done action. The rotation is qualitatively “perfect”. The test also shows that the addition of a stopper to deal with overturning/underturning and to provide support is proving effective.  
**Action:** Implement 195 degree of the wheel rotation within the software.   
**Distribution:** Software Team

# 5.3 Navigation

# 5.3.1 Navigation 1.0

**Test’s Title: Navigation**

**Tester:** *Volen Mihaylov* **Test Date:** 03/22/18

**Software Version:** *01.00.07 to 1.01.00*

**Hardware Version:** *2.0*

**Objective:** The objective of this test is to test the accuracy of navigation code by checking whether the robot can move to a certain coordinate within an error tolerance of 2 cm using the Euclidean distance measure.

.

**Background knowledge:** None

**Procedure:**

1. Place the robot at its original position; that is, (0,0) as defined by your co-ordinate orientation.
2. Make the robot turn at certain wanted degrees. These are: (±45, ±90, ±270). Verify the offset, if there is any.
3. Make the robot move by a set wanted distance and verify the offset (1 tile, 2 tiles, 6 tiles).
4. Finally, make the robot move to a certain co-ordinate and verify the distance covered. In this case, our starting co-ordinate is (0,0) and our end co-ordinate is (5,3).

**Expected Results:**

After conducting many tests runs to correct for small error offsets in angle and travelling, it expected that the Navigation be fully functioning as well as precise.

**Results obtained:**

**Note:** *The test data has been sub-split into 6 test sets. Please refer to the notes in the table breaks.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tests** | **Variables** | | | | **Passed?** | **Comments:** |
| Degrees tested at | Distance tested at | Coordinates tested at | Code Constant changed: |  |
| **1** | 45 |  |  | 0 | No | Left turn is off by 2 degrees to the left |
| **1.1** | -45 |  |  | 0 | No | Right turn is perfect |
| **1.2** | 90 |  |  | 0 | No | Left turn is off by 2 degrees to the left |
| **1.3** | -90 |  |  | 0 | No | Right turn is perfect |
| **1.4** | +270 |  |  | 0 | No | Right turn is perfect |
| **1.5** | -270 |  |  | 0 | No | Left turn is off by 2 degrees to the left |
| *Following these initial 6 tests from test set #1, it is observed that the left turn is off by a couple of degrees. Test set #2 below adds a 4-degree correction to the left turn.* | | | | | | |
| **2.0** | 45 |  |  | Left: +4 | No | Left turn is off by 2 degrees to the right. |
| **2.1** | -45 |  |  | Left: +4 | No | Right is perfect |
| **2.2** | 90 |  |  | Left: +4 | No | Left turn is off by 2 degrees to the right. |
| **2.3** | -90 |  |  | Left: +4 | No | Right is perfect |
| **2.4** | +270 |  |  | Left: +4 | No | Right is perfect. |
| **2.5** | -270 |  |  | Left: +4 | No | Left turn is off by 2 degrees to the right |
| *Following test set #2 it is observed that 4 degrees is an overcorrection. Test set #3 below adds a 2-degree correction to the left turn instead.* | | | | | | |
| **3.0** | 45 |  |  | Left: +2 | No | Left Turn is perfect. |
| **3.1** | -45 |  |  | Left: +2 | No | Right Turn is perfect |
| **3.2** | 90 |  |  | Left: +2 | No | Left Turn is perfect. |
| **3.3** | -90 |  |  | Left: +2 | No | Right Turn is perfect |
| **3.4** | +270 |  |  | Left: +2 | No | Right Turn is off by 2 degrees not enough. |
| **3.5** | -270 |  |  | Left: +2 | No | Left is perfect |
| *Following test set #3, it is observed that both right and left turns are perfect. Test set #4 below now records the accuracy of the odometer.* | | | | | | |
| **4.0** |  | 1 tile |  |  | No | 0.5cm too short |
| **4.1** |  | 1 tile |  |  | No | 0.5cm too short |
| **4.2** |  | 2 tiles |  |  | No | 0.9cm too short |
| **4.3** |  | 2 tiles |  |  | No | 0.9cm too short |
| **4.4** |  | 6 tiles |  |  | No | 2cm too short. |
| **4.5** |  | 6 tiles |  |  | No | 2cm too short |
| *Following test set #4, it is observed that the robot does not travel the full desired tile length, falling slight short. Test set #5 below adjusts this error by applying: Rotation multiplied by: (TILESIZE+0.4)/TILESIZE.* | | | | | | |
| **5.0** |  | 1 tile |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.1** |  | 1 tile |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.2** |  | 2 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.3** |  | 2 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.4** |  | 6 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Satisfactory | Perfect |
| **5.5** |  | 6 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Satisfactory | Perfect |
| *Following test set #5, it is observed that the correctional formula applied corrects the offset error. Test set #6 now implements both angle turning as well as travelling. This serves to test the integration and overall accuracy of the navigation. Start corner: (0,0) -> End Corner:(5,3)* | | | | | | |
| **6.0** |  |  | (5,3) |  |  | Off by 3cm |
| **6.1** |  |  | (5,3) |  |  | Perfect |
| **6.2** |  |  | (5,3) |  |  | Perfect |

**Conclusion:** This extensive test procedure has concluded by ensuring that Navigation is fully functional and precise.

**Action:** Left turning is corrected by applying a +2-degree constant and travelling across a tile is corrected by rotateByDistance constant of (*TILESIZE*+0.4)/*TILESIZE* within the Navigation class.

**Distribution:** Software Team

# 5.3.2 Navigation 2.0

**Test’s Title: Navigation**

**Tester:** *Bryan Jay* **Test Date: 04/05/18**

**Software Version:** *06.01.00*

**Hardware Version:** *2.2*

**Objective:** The objective of this test is to test the accuracy of navigation code by checking whether the robot can move to a certain coordinate within an error tolerance of 2 cm using the Euclidean distance measure. Due to one of the motors breaking down, this test had to be redone.

**Background knowledge:** None

**Procedure:**

1. Place the robot at its original position; that is, (0,0) as defined by your co-ordinate orientation.
2. Make the robot turn at certain wanted degrees. These are: (±45, ±90, ±270). Verify the offset, if there is any.
3. Make the robot move by a set wanted distance and verify the offset (1 tile, 2 tiles, 6 tiles).
4. Finally, make the robot move to a certain co-ordinate and verify the distance covered. In this case, our starting co-ordinate is (0,0) and our end co-ordinate is (5,3).

**Expected Results:**

After conducting many tests runs to correct for small error offsets in angle and travelling, it expected that the Navigation be fully functioning as well as precise.

**Results obtained:**

**Note:** *The test data has been sub-split into 6 test sets. Please refer to the notes in the table breaks.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tests** | **Variables** | | | | **Passed?** | **Comments:** |
| Degrees tested at | Distance tested at | Coordinates tested at | Code Constant changed: |  |
| **1** | 45 |  |  | 0 | No | Left turn is off by 2 degrees to the left |
| **1.1** | -45 |  |  | 0 | No | Right turn is perfect |
| **1.2** | 90 |  |  | 0 | No | Left turn is off by 2 degrees to the left |
| **1.3** | -90 |  |  | 0 | No | Right turn is perfect |
| **1.4** | +270 |  |  | 0 | No | Right turn is perfect |
| **1.5** | -270 |  |  | 0 | No | Left turn is off by 2 degrees to the left |
| *Following these initial 6 tests from test set #1, it is observed that the left turn is off by a couple of degrees. Test set #2 below adds a 4-degree correction to the left turn.* | | | | | | |
| **2.0** | 45 |  |  | 0 | No | Left turn is off by 2 degrees to the right. |
| **2.1** | -45 |  |  | 0 | No | Right is perfect |
| **2.2** | 90 |  |  | 0 | No | Left turn is off by 2 degrees to the right. |
| **2.3** | -90 |  |  | 0 | No | Right is perfect |
| **2.4** | +270 |  |  | 0 | No | Right is perfect. |
| **2.5** | -270 |  |  | 0 | No | Left turn is off by 2 degrees to the right |
| *Following test set #2 it is observed that 4 degrees is an overcorrection. Test set #3 below adds a 2-degree correction to the left turn instead.* | | | | | | |
| **3.0** | 45 |  |  | 0 | No | Left Turn is perfect. |
| **3.1** | -45 |  |  | 0 | No | Right Turn is perfect |
| **3.2** | 90 |  |  | 0 | No | Left Turn is perfect. |
| **3.3** | -90 |  |  | 0 | No | Right Turn is perfect |
| **3.4** | +270 |  |  | 0 | No | Right Turn is off by 2 degrees not enough. |
| **3.5** | -270 |  |  | 0 | No | Left is perfect |
| *Following test set #3, it is observed that both right and left turns are perfect. Test set #4 below now records the accuracy of the odometer.* | | | | | | |
| **4.0** |  | 1 tile |  |  | No | 0.5cm too short |
| **4.1** |  | 1 tile |  |  | No | 0.5cm too short |
| **4.2** |  | 2 tiles |  |  | No | 0.9cm too short |
| **4.3** |  | 2 tiles |  |  | No | 0.9cm too short |
| **4.4** |  | 6 tiles |  |  | No | 2cm too short. |
| **4.5** |  | 6 tiles |  |  | No | 2cm too short |
| *Following test set #4, it is observed that the robot does not travel the full desired tile length, falling slight short. Test set #5 below adjusts this error by applying: Rotation multiplied by: (TILESIZE+0.4)/TILESIZE.* | | | | | | |
| **5.0** |  | 1 tile |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.1** |  | 1 tile |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.2** |  | 2 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.3** |  | 2 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Yes | Perfect |
| **5.4** |  | 6 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Satisfactory | Perfect |
| **5.5** |  | 6 tiles |  | *(TILESIZE+0.4)/TILESIZE* | Satisfactory | Perfect |
| *Following test set #5, it is observed that the correctional formula applied corrects the offset error. Test set #6 now implements both angle turning as well as travelling. This serves to test the integration and overall accuracy of the navigation. Start corner: (0,0) -> End Corner:(5,3)* | | | | | | |
| **6.0** |  |  | (5,3) |  |  | Off by 3cm |
| **6.1** |  |  | (5,3) |  |  | Perfect |
| **6.2** |  |  | (5,3) |  |  | Perfect |

**Conclusion:** This extensive test procedure has concluded by ensuring that Navigation is fully functional and precise.

**Action:** Left turning is corrected by applying a +2-degree constant and travelling across a tile is corrected by rotateByDistance constant of (*TILESIZE*+0.4)/*TILESIZE* within the Navigation class.

**Distribution:** Software Team

# 5.4 Obstacal travaersal

# 5.4.1 obsatcle traversal – Applying different rear wheels

***Obstacle Traversal (Applying Different Rear Wheels)***

**Tester:** *Enan Ashaduzzaman*  **Test Date:** *03/12/18* **Software Version:** *N/A* **Hardware Version:** *1.0*

**Objective:**

The objective of this test is to determine whether or not the robot is able to drive through the bridge and tunnel successfully. Not being able to traverse either the bridge or tunnel will not allow the robot to move in-between the red and green zones of the playing field.

**Background knowledge:**

**Procedure:**

1. Make sure the Hardware Version 1.0 is not equipped with the landing gear. Instead run this test using a single marble attached to the rear of the robot.
2. Implement a test code that allows the robot to drive straight.
3. Place the robot in front of the bridge.
4. Execute the drive-forward code.
5. Analyze if the robot successfully traverses the bridge.
6. Make sure the Hardware Version 1.0 is not equipped with the landing gear. Instead run this test using two marbles attached to the rear of the robot, one on each corner.
7. Place the robot in front of the tunnel.
8. Execute the drive-forward code.
9. Analyze if the robot successfully traverses the bridge.
10. Use the exact Hardware Version 1.0. Make sure the landing gear is pulled is lowered when the robot is completing this test.
11. Place the robot in front of the tunnel.
12. Execute the drive-forward code.
13. Analyze if the robot successfully traverses the bridge.

**Expected Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **tEST #** | obstacle | **Rear Wheel type?** | **Passed?** | **Comments** |
| **1.1** | BRIDGE | single marble | no | the marble from the rear end changes the direction of the robot, causing it to crash into the wall. |
| **1.2** | BRIDGE | single marble | no |
| **2.1** | tunnel | single marble | yes | The robot has difficulty in the beginning when the marble encoutners the surface of the bridge. It causes the robot to buffer for a couple of seconds and it chabges the direction of the robot as it enters the tunnel. The robot is able to traverse the bridge. |
| **2.2** | tunnel | single marble | yes |
| Two marbles were attached in the rear end instead of one. They were placed on the corners. | | | | |
| **3.1** | bridge | two marbles | no | tHE TWO MARBLES PERFORMED BETTER THAN THE SINGLE MARBLE. tHE ROBOT STILL HAD DIFFICULTIES TRAVERSING THE BRIDGE. tHE MARBLES IN THE REAR END CONTINUED TO CHANGE THE DIRECTION OF THE ROBOT WHEN TRAVERSING THE BRIDGE. tHIS MADE IT CRASH INTO THE WALL. |
| **3.2** | bridge | two marbles | no |
| **4.1** | TUNNEL | two marbles | yes | tHE ROBOT HAD A SLIGHT SECOND OF HESITATION WHEN THE MARBLES ENCOUNTERED THE SURFACE OF THE BRIDGE. tHE TWO MARBLES HELPED THE ROBOT TO ALIGN ITSELF BEFORE ENTERING THE TUNNEL. |
| **4.2** | TUNNEL | two marbles | yes |
| **A landing gear was attached to the robot. tHE TWO MARBLES WERE TAKEN OFF THE ROBOT. a SINGLE MARBLE IS ATTACHED INSTEAD RIGHT BEFORE THR LANDING GEAR (MARBLE USED DURING NAVIGATION AND LOCALIZATION).** | | | | |
| **5.1** | bridge | Landing Gear | yes | wHEN THE LANDING GEAR WAS USED, THE ROBOT TRAVERSED THE BRIDGE WITHOUT ANY ISSUES. |
| **5.2** | bridge | landing gear | yes |
| **6.1** | tunnel | landing gear | yes | using THE LANDING GEARS, THE ROBOT DID NOT HAVE A SECOND OF HESITATION WHEN TRAVERSING THE TUNNEL. mOREOVER, THE LANDING GEARS DID NOT CHANGE THE DIRECTION OF THE ROBOT WHILE ENTERING THE TUNNEL. |
| **6.2** | tunnel | landing gear | yes |

**Conclusion:**

The robot performed best when landing gears were utilized while it traversed both obstacles. The landing gears did not change the direction of the robot while it was traversing either obstacles.

**Action:**The next step is to implement a landing gear on the robot which will only be used when the robot is traversing obstacles.

**Distribution:**Hardware Team and Software Team

# 5.4.2 obsatcle traversal - Encoutnering obsatcles at an angle

***Obstacle Traversal (When Encountered at Different Angles)***

**Tester:** *Enan Ashaduzzaman, Bryan Jay***,** *Tianyi Zou* **Test Date:** *04/04/18* **Software Version:** *N/A* **Hardware Version:** *2.0 - 2.1*

**Objective:**

The objective of this test is to determine whether or not the robot is able to drive through the bridge and tunnel successfully when encountering the obstacles at an angle.

**Background knowledge:**

Test 4.4 concluded that the marbles were not a feasible option when traversing the bridge. Moreover, it was known that the threads were not as precise as the regular wheels when completing navigation.

**Procedure:**

1. Implement a test code that allows the robot to drive straight on Hardware Version 2.0.
2. Place the robot in front of the bridge
3. Execute the drive-forward code.
4. Analyze if the robot successfully traverses the bridge.
5. Redo steps 2-4 placing the robot at an angle (to check if the robot is able to traverse the bridge when it doesn’t start straight)
6. Place the robot in front of the tunnel.
7. Execute the drive-forward code.
8. Analyze if the robot successfully traverses the bridge.
9. Redo steps 6-8 placing the robot at an angle (to check if the robot is able to traverse the tunnel when it doesn’t start straight).
10. Implement guard rails on the robot near the corners (Hardware Version 2.1)
11. Redo steps 2-9.

**Expected Results:**

It is expected that the robot will be able to successfully traverse both obstacles using Hardware Version 2.0. The implementation of guard rails will not be necessary.

**Results Obtained:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **tEST #** | Obstacle | **Angle OF APproach (°)** | **Passed?** | **Comments** |
| **1.1** | BRIDGE | -20 | no | tHE ROBOT ENCOUNTERED THE walls OF THE BRIDGE CAUSING IT TO GET STUCK. |
| **1.2** | BRIDGE | -10 | yes | tHE ROBOT ENCOUNTERED THE walLS OF THE BRIDGE BUT, MAKES IT OUT SUCCESSFULLY. |
| **1.3** | BRIDGE | 0 | yes | tHE ROBOT SUCCESSFULLY MAKES IT OUT OF THE BRIDGE WITHOUT ENCOUTNERING THE walLS. |
| **1.4** | BRIDGE | 10 | yes | tHE ROBOT ENCOUNTERED THE RAILS OF THE BRIDGE BUT, MAKES IT OUT SUCCESSFULLY. |
| **1.5** | BRIDGE | 20 | no | tHE ROBOT ENCOUNTERED THE RAILS OF THE BRIDGE CAUSING IT TO GET STUCK. |
| **2.1** | TUNNEL | -20 | no | tHE ROBOT ENCOUNTERED THE WALLS OF THE TUNNEL AND IMMIDIETLY GETS STUCK. |
| **2.2** | TUNNEL | -10 | no | tHE ROBOT ENCOUNTERED THE WALLS OF THE TUNNEL AND IMMIDIETLY GETS STUCK. |
| **2.3** | TUNNEL | 0 | yes | tHE ROBOT SUCCESSFULLY MAKES IT OUT OF THE TUNNEL WITHOUT ENCOUTNERING THE WALLS. |
| **2.4** | TUNNEL | 10 | no | tHE ROBOT ENCOUNTERED THE WALLS OF THE TUNNEL AND IMMIDIETLY GETS STUCK. |
| **2.5** | TUNNEL | 20 | no | tHE ROBOT ENCOUNTERED THE WALLS OF THE TUNNEL AND IMMIDIETLY GETS STUCK. |
| **Guard rails were implemented on the robot (hARDWARE version 2.1) with the intention of having the robot to adjust itself using the rails when it encounters the walls of either obstacles.** | | | | |
| **3.1** | bridge | -20 | yes | The guard rails of the robot encoutners the walls of the bridge and adjusts itself to successfully traverse the obstacle. |
| **3.2** | bridge | -10 | yes | The guard rails of the robot encoutners the walls of the bridge and adjusts itself to successfully traverse the obstacle. |
| **3.3** | bridge | 0 | yes | tHE ROBOT SUCCESSFULLY MAKES IT OUT OF THE bridge WITHOUT ENCOUTNERING THE WALLS. |
| **3.4** | bridge | 10 | yes | The guard rails of the robot encoutners the walls of the bridge and adjusts itself to successfully traverse the obstacle. |
| **3.5** | bridge | 20 | yes | The guard rails of the robot encoutners the walls of the bridge and adjusts itself to successfully traverse the obstacle. |
| **4.1** | tunnel | -20 | yes | The guard rails of the robot encoutners the walls of the tunnel and adjusts itself to successfully traverse the obstacle. |
| **4.2** | tunnel | -10 | yes | The guard rails of the robot encoutners the walls of the tunnel and adjusts itself to successfully traverse the obstacle. |
| **4.3** | tunnel | 0 | yes | tHE ROBOT SUCCESSFULLY MAKES IT OUT OF THE TUNNEL WITHOUT ENCOUTNERING THE WALLS. |
| **4.4** | tunnel | 10 | yes | The guard rails of the robot encoutners the walls of the tunnel and adjusts itself to successfully traverse the obstacle. |
| **4.5** | tunnel | 20 | yes | The guard rails of the robot encoutners the walls of the tunnel and adjusts itself to successfully traverse the obstacle. |

**Conclusion:**

When the robot did not have any guard rails (Hardware Version 2.0), the robot got stuck whenever the wheels encountered the walls of either obstacle. The robot is able to successfully traverse both obstacles without any issues only when there are guard rails implemented on the robot (Hardware Version 2.1). The guard rails were able to adjust the direction of the robot so that it slid against the walls.

**Action:**The next step is to add guard rails to the robot.

**Distribution:**Hardware Team

# 5.8 WI-FI integration

***Wi-Fi Integration***

**Tester:** *Bryan Jay* **Test Date:** *03/31/18* **Software Version:** *N/A* **Hardware Version:** *N/A*

**Objective:**

The objective of this test is to determine whether or not the Wi-Fi class functions when it needs to receive data from the Wi-Fi server. Without the data from the Wi-Fi server, the robot would not function since it would not be able to implement data points inside the controller.

**Background knowledge:**

None

**Procedure:**

1. Setup Wi-Fi in the brick by connecting it to the DPM server using the wireless USB adapter.
2. Connect your computer to the DPM server and run the DPM server jar file.
3. Using the jar file, fill in the data boxes with the xml data.
4. Adjust the data according to the picture at the bottom. (Testing for Green Team)



1. Run the robot’s code and wait until the robot connects to the DPM server jar
2. Click “Start” in the jar application
3. Watch to see if the robot executed the code
4. Repeat steps 4-7 using the following adjustments in the jar application. (Testing for Red Team)



**Expected Results:**

It is expected that the Wi-Fi class functions and is able to receive the data from the Wi-Fi server using the wireless USB adapter. Given that the Wi-Fi class functions, the robot should proceed to localization and navigation.

**Results Obtained:**

The test was completed 5 times. During each test run, every parameter in the jar application was changed other than the “Team Number.” During every test run, the robot was able to execute the code after receiving data from the Wi-Fi server.

**Conclusion:**The Wi-Fi class worked perfectly. Data was received from the Wi-Fi server which was then implemented inside the controller. After receiving the data, the robot executed it’s localization and navigation class.

**Action:**The next step is to complete the Search and Capture class, the final step in the software process.

**Distribution:**Software Team

# Approval

The undersigned acknowledge they have reviewed the **Test Plan** document and agree with the approach it presents. Any changes to this document will be coordinated with and approved by the undersigned.

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

Appendix A: References

1. The following table summarizes the documents referenced in this document.

|  |  |  |
| --- | --- | --- |
| **Document Name and Version** | **Description** | **Location** |
|  |  |  |

1. The following is a table template for many of the tests that have been conducted throughout this document.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tests** | **Variables** | | | | **Passed?** | **Comments:** |
|  |  |  |  |  |
| **1** | v |  |  |  |  |  |
| **2** |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |

Appendix B: Key Terms

The following table provides definitions for terms relevant to this document.

|  |  |
| --- | --- |
| **Term** | **Definition** |
|  |  |
|  |  |
|  |  |