*ecse 211 design project*

Software Document

Version *2.03*

*04/12/2018*

*ECSE 211 TEAM 11*

VERSION HISTORY

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Title** | Software Document | | | |
| **Description** | Keeps track of all Software related implementation | | | |
| **Created By** | Bryan Jay, Project Manager & Volen Mihaylov, Software Manager | | | |
| **Date Created** | 2st March 2018 | | | |
| **Version Number** | **Modified By** | **Modifications Made** | **Date Modified** | **Status (Refer to Software Document Section 2.0)** |
| 1.00 | Patrick Ghazal | Created the Document. Asserted 3 possible preliminary designs coupled with their respective advantages/disadvantages | 2nd March, 2018 |  |
| 1.01 | Luka Jurisic | Peer reviewed the document. Formatted the Document. | 3rd March, 2018 | Preliminary Week 2 submission Content complete |
| 1.02 | Patrick Ghazal  Volen Mihaylov | Patrick-Added sections 3-Flowchart and 4-Class Diagrams.  Volen-Added Software Progress Report | 22nd March, 2018 | Everything up to milestone 2 is complete. The next step is to complete milestones 3,4 & 5 for the beta demo. |
| 2.00 | Enan Ashaduzzaman  Bryan Jay  Volen Mihaylov | Enan-Formatted the Document  Bryan Jay-Redid sections 2 & 4  Volen-Completed section 3- Flowchart | 29th March, 2018 | Everything up to milestone 6 is complete. The next step is to complete Milestone 7-Search and Localize. |
| 2.01 | Bryan Jay | Updated software flowchart. Updated search and localization description | 5th April 2018 | Milestone 7 is partially completed. Milestones 8-10 still need completed |
| 2.02 | Bryan Jay | Updated Class Diagram. Updated Localization and navigation | 8th April 2018 | Milestone 7 is partially completed. Milestones 8-10 still need completed |
| 2.03 | Volen Mihaylov | Updated Edit History, class explanations, system integration | 12th April 2018 | Milestone 10 is complete having skipped milestone 7 |

TABLE OF CONTENTS

# 1 Table of ContentS ……………………………………………………………….3

# 2 Design Process ……………………………………………………………………

# 

# 3 Flowchart …………………………………………………………………………

# 4 Class Hierarchy

4.1 WiFi Data ………………………………………………………………………………

4.2 Robot ………………………………………………………………………………

4.3 Controller ………………………………………………………………………………

4.4 Localization …………………………………………………………………………

4.5 Odometer …………………………………………………………………

4.6 Navigation ………………………………………………………………………………

4.7 Colour Calibration ……………………………………………………………………

4.8 Search and Localize ……………………………………………………………………

# 5 JAVADOCS ………………………………………………………………………………

# 6 System architecture and integration …………………………………

# 7 edit history …………………………………………………………………………

# 8 Glossary ………………………………………………………………………………

# 2 DESIGN PROCESS

Before jumping into development, it is important to create the skeletons of our design base on the requirements and constraints imposed by the design project at hand. We generated a flow chart as a template for our software, as it visually demonstrated the interactions between our classes with one and another to create a solution. The requirements and specifications drove the development of all the classes and methods illustrated in the original diagram.

The first phase in development was to implement a versioning system in GitHub in order to record all future changes of code. We implemented a README file which outlined the way a proper commit was to be represented, as well as the software milestones. After completing our preliminary code, our software was then pushed forward into the testing phase. As each component was complete such as navigation, localization, etc. the testing team moved in parallel with the software team to test components. Throughout the testing phase our testing team implemented appropriate tests in order to test certain functionalities based on requirements. Based on feedback from the testing team certain improvements were made to the software and hardware.

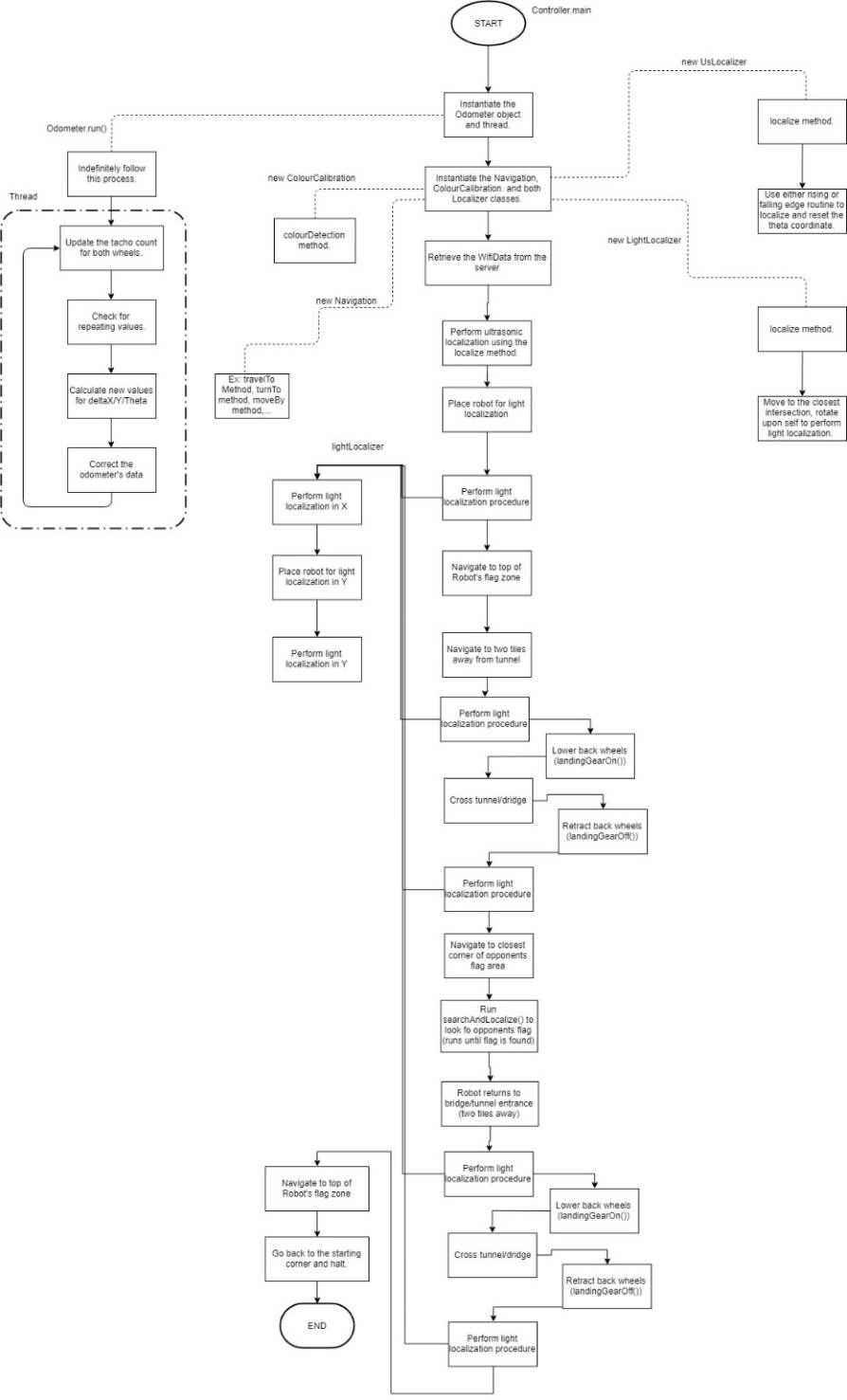
|  |
| --- |
|  |

Figure 1: Excerpt from README

In order for our code to remain accessible and understandable, we used Javadocs for documentation. There are descriptions provided with every method and glass generated which is accessible on an HTML file for the team. The documentation allowed our multiple software engineers to be working on the code simultaneously, due to the fact that it was understandable for any new author working on the software.

# 3 flowchart

Refer to the Systems Document for systems flowchart which is the robots process. In terms of our software representation, the diagram below explains the implemented classses and how they work together.



*Figure 2: Software Flowchart*

The software flowchart roots from the controller class. The main method sits in the controller class and it is the class that connects to all the other in order to co complete the objective.

The first process is to start the odometry thread in order for our robot to keep track of its location relative to the coordinate system on the board from the lines. The odometer capable of keeping track of the location through the calculation of the motors tachocounts and convert these values to coordinates from the track and wheel radius variable. Throughout the travel of the robot error is accumulated in the odometer.

In order to gain access to these other classes, we instantiate the objects for the Navigation, Colour Calibration, and both Localizer classes. The DPM WiFi server is platform that is used for our robot to access the data that required for it to begin execution of the objective.

The robot first need to localize fully localize in one of the 4 corners and within its team’s colour respective area. To complete the full localization the robot uses the process outlined in section 4.4 of the software document. As previously mentioned the odometer accumulates an error due to certain physical and software phenomenon’s. In order to mitigate the error there is another localization process that is quicker outlined in section 4.4 of the software document that is used to readjust itself throughout the course of its path.

Depending on the received data the robot recognizes its location and its team’s colour. The data is the used to navigate to the respective obstacle. The navigation method is the same in all cases and is used similarly in all cases except for when crossing the bridge. In order to traverse the bridge, it is imperative that the robot lowers its landing gear and then raises it again for it to navigate. Again, localization is performed after navigation calls to limit the error accumulated. Essentially the navigation class is responsible for moving the robot to its given coordinates.

The search and localize procedure is next and this procedure is outlined in section 4.8. The robot’s goal here is to search for the blocks in the search area. After finding a block, it utilize the colour calibration class to determine the block’s colour with the algorithm in section 4.7 in this document.

# 4 class hierarchy

Our software flowchart is then broken down into the following Class Hierarchy:

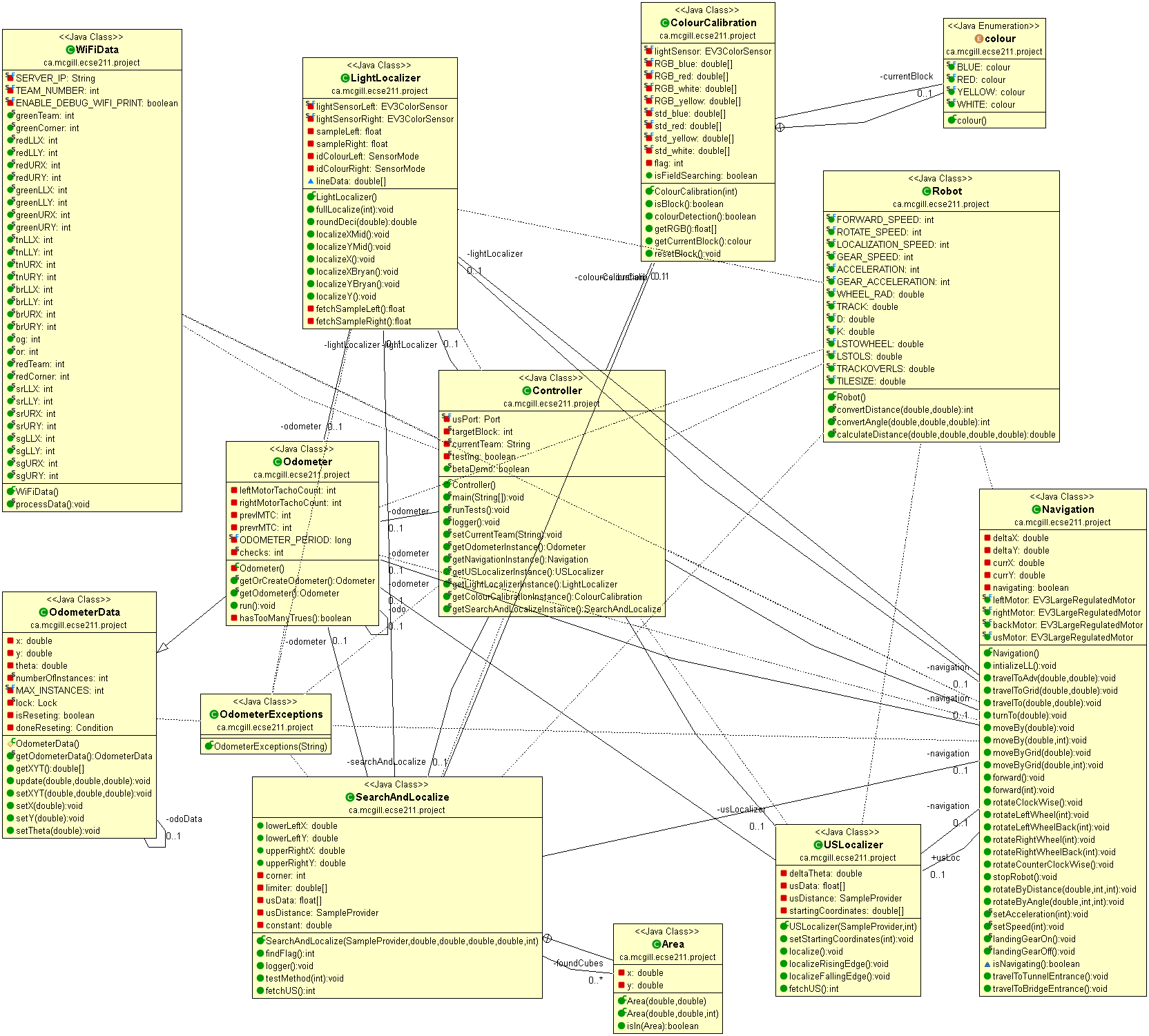


Figure 3: Class Diagram

**4.1 WIFI Data**

The WIFI class is used as link between our robot and the computer. The class receives the data which our robot then uses to determine its team, its position, its target block and the location of the obstacles. The robot will complete the requirements based on the initial data forwarded to it.

**4.2 Robot**

The Robot class’ purpose is to contain and manage all the constants and variables related to the robot’s: motors, sensors, constants such as the track and wheel radius, etc. This allows for less memory to be used as the data will centralized and accessible by all classes limiting the repetition of variable compared to previously. In addition, the Robot class also includes the convertDistance, convertAngle, and calculateDistance methods, that are used namely for Navigation purposes.

**4.3 Controller**

The controller class is the main entry point into the software for the entire project, as it contains the main method relating to other classes. Using the controller class, we instantiate and declare the required threads and objects, respectively. Therefore, the controller class creates the central class which communicates with all other classes to run accordingly. Controller as well contains the test methods to allow for systematic testing of each of the robot’s functions and of its system. This class as well contains the pathing algorithms for the full system run.

* The test method contains multiple tests which are linked to the current iteration of the robot’s version. The tests can be selected via the change of the “test” integer.
* The full system run method contains the 16 main possibilities of course setups i.e. the pathing the robot would take if it was either the red or green team in whichever of the 4 corners may the course be upwards (bridge and tunnel pointing in the Y direction) or may it be sideways. The differentiation between an upwards and sideways course is done by verifying in which direction the bridge has a dimension of two tiles, or if it is 1 by 1 it verifies in which direction are both team zones spaced by 1 tile.

**4.4 USLocalization and LightLocalization**

The light localizer class works jointly with the ultrasonic localization in order to determine the robot’s initial starting point and establish its heading and coordinates. The localization is used to set the robot’s X and Y coordinates and the heading based on the playing field. After completing localization, the robot is facing away from the wall in the Y direction correspondingly to the corner it is placed. So it is facing at the angle of 0˚ at corners 0 and 1 and 180˚ at corners 2 and 3 with the appropriate X and Y coordinates. The entire light localization process takes approximately 25 seconds to localize.

The initial US localization is used to set the heading of the robot using either a falling edge or rising edge method based on the distance of the US sensor. The position is also estimated to be three quarter tiles away from walls. The falling edge is executed when the US sensor is faced away from the wall and vice versa. This routine rotates the robot on its axis recording the angle, α, at which it reaches a falling edge for the falling edge case and rising edge for the rising edge case and then it rotates the other direction and records β. The heading is then calculated from the recorded angles α, β using either one of the following equations:

The robot then positions itself on what it believes to be the closest grid intersection. The position and heading are then reset to their exact locations on the playing field due to the two methods of light localization localizeXBryan() and localizeYBryan(). Everything is tied in using the startLocalization(int corner) method in controller.

The light localization methods are used at the beginning to calibrate its position and as well throughout the competition to correct the robots position. The robot utilizes the two light sensors equipped at each wheel in order to determine the robots offset when traversing a line. The robot will turn until both light sensors are aligned on the line and have the same amount of red light reflected onto them. If the robot happens to turn for more than 35 degrees and has yet to find the line it will start rotating the other way as it will assume it has missed it.

Throughout the course of the game we repeat localization at every grid line as integrated in the advTravelToGrid() method using the aforementioned localization methods localizeXBryan() and localizeYBryan(). The robot calculates the next grid lines its light sensors should encounter using the following formulas:

For localizeXBryan()::

For localizeYBryan():

When localization procedures repeat, the ultrasonic localization method is not performed as the robot will be capable of re-localizing using the lines on the playing field in order to skip that procedure and perform a variant of the light localization procedure.

The US localization is only called at the beginning for the initial localization. As the light localization method is integrated into navigation the robot can decrease its error in position which is accrued through many factors such as friction, motor imperfections, sensor imperfections, etc.

**4.5 Odometer**

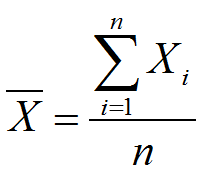
The odometer class is used to determine the estimated position of the robot by counting motor rotations. The odometer ignores physical experiences such a slip, friction, a constant wheel radius and wheelbase. Therefore, the omittance of such errors and flaws accumulated throughout the robot’s navigation and the error increases as the motor rotations increase. Localization must be performed multiple times throughout navigation around the playing field in order to update the odometer and reduce the effect of these error. The odometer class is an extension of the OdometerData class which handles limit the amount of instantiations and value setting and getting. OdometerException is used to throw errors.

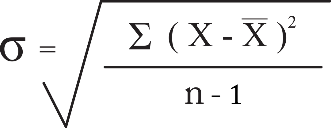
**4.6 Navigation**

The navigation class implements methods which moves the robot around the playing field and rotates its heading. The navigation procedure works very closely with the odometer as it relies on the odometer to move to its given locations. The navigation method works in tandem with odometry as the odometer is continuously being updated throughout its course. As well, in order to mitigate the accumulating error from traversing the game board, the robot is continuously re-localizing as it crosses every line. This localization method is outlined in this document’s section 4.4.

**4.7 Colour Calibration**

The colour detection class utilizes the Gaussian distribution approach to determine the colour of the blocks. In order to do this, we recorded the RGB readings from the light sensor on the different coloured blocks at close distances and using this data we calculated the mean and standard of deviation using the following formulas:



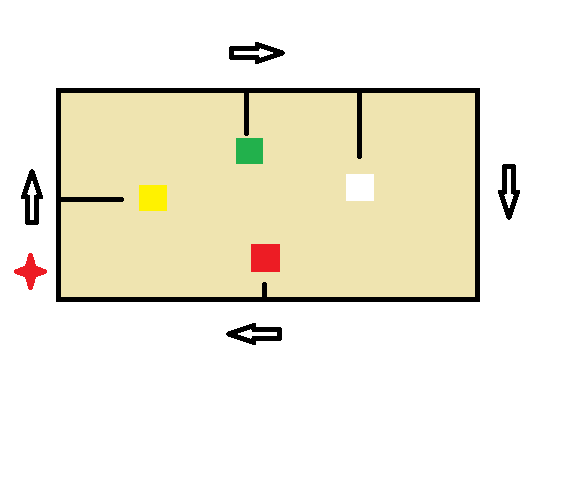


The color calibration determines whether or not the block’s readings are within the standard deviation of all three RGB values which deduces its color. This method is effective ad distances of 0-2 cm from the block.

**4.8 Search and Localize**

The search and localize method uses a searching algorithm which allows our robot to stay at the perimeter of the search area and find the blocks without an obstacle avoidance. The robot begins in any given corner and patrols the edge of the search zone and traverses the area clockwise as it stays on the grid lines. The robot’s ultrasonic sensor turns 90˚ facing right to allow for block detection. The robot stops at every third of a tile to routinely check for a block that is at a distance of less then half the search area dimension the ultrasonic sensor is scanning. If the sensor detects the block the following steps will be executed:

1. Navigation will stop.
2. The cubes coordinates will be verified to see if it hasn’t been already detected
3. Two options
   1. If it has already detected the cube as verified by the Area class, robot will continue on its path,
   2. If it hasn’t the following steps will follow
4. The ultrasonic sensor will turn to face forward.
5. The robot will turn 90˚ to face the block.
6. It will navigate to the block until the floating light sensor is above the block.
7. The colour calibration methods will be called to determine its colour.
   1. If the target block is found the robot will beep three times signaling a successful search and return to its position on the path
   2. If it is not the target block the robot will return to its initial position along the path by simply backing up and continue its navigation.

  
The area class stores the coordinates of the detected blocks. The block’s center point is estimated to be 5cm away in the direction that it has been detected. When it is checked if a detected block has already been tested it is verified that the detected estimated center coordinates are at a distance of half a tile length within every other stored coordinate which is the minimum distance between two cube sides. If the currently coordinates are within such a rang of all other stored coordinates then the cube has already been checked. If not, the robot proceeds to go to it.

# 5 javadocs

In order to document our software, the Javadocs tool is used to generate our API documentation on a HTML file. Throughout the software development process, there are comments following each respective method to explain their usage and behaviours for easy readability and reusability. The API documentation is automatically generated, and it represents a roadmap to our implemented software. The Javadocs are essential for understanding our classes and methods, especially for the multiple people working our software at the same time.

# 6 system architecture and integration

The Controller contains the system integration and links all the classes together. There are a total of 16 possible course set-ups as illustrated in the behaviour trees below. The pathing of the robot itself allows for full obstacle avoidance.

|  |
| --- |
| C:\Users\Volen Mihaylov\AppData\Local\Microsoft\Windows\INetCache\Content.Word\FieldUpwardsSystemFlowchart.png  System Integration Part 1: Cases if Field is Upwards (Bridge and Tunnel must be crossed in the Y-Direction) |
| C:\Users\Volen Mihaylov\AppData\Local\Microsoft\Windows\INetCache\Content.Word\FieldSidewaysSystemFlowchart.png  System Integration Part 2: Cases if Field is Sideways (Bridge and Tunnel must be crossed in the X-Direction) |

# 7 MAJOR edit History

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | Keeps track of modification of the final software code. | | |
| **Created By** | Volen Mihaylov, Software Manager | | |
| **Date Created** | 12th March, 2018 | | |
| **Version** | **Engineer** | **Summary** | **Date Modified** |
| 1.00 | Volen Mihaylov | Imported All Class Shells from Lab 5. Created Controller and Robot class | 03/16/18 |
| 1.01 | Patrick Ghazal | Imported & Implemented WiFi code, Connected WiFi Data class to the controller class | 03/17/18 |
| 1.02 | Volen Mihaylov | Completed Robot class and implemented Landing Gear Functionality | 03/18/18 |
| 2.00 | Volen Mihaylov | Navigation Development Complete | 03/22/18 |
| 3.00 | Volen Mihaylov | Imported old odometer from Lab 5 | 03/23/18 |
| 4.00 | Volen Mihaylov, Bryan Jay | Implemented US Localizer | 03/24/18 |
| 5.00 | Volen Mihaylov,  Bryan Jay | Developed a working localization algorithm, Updated navigation values | 03/26/18 |
| 5.01 | Volen Mihaylov | Removed Useless functions, Updated adjustments for beta demo | 03/27/18 |
| 6.00 | Patrick Ghazal | Improved WiFi data retrival and Elaborated Javadocs, Updates Constants | 03/28/18 |
| 6.01 | Volen Mihaylov | Fix WiFi data error | 03/29/18 |
| 6.02 | Bryan Jay | Developed Colour Calibration to work without threading | 03/31/18 |
| 6.03 | Volen Mihaylov | Developed a Search & Localize | 04/01/18 |
| 6.04 | Patrick Ghazal | Updated Javadocs & removed useless class instances | 04/02/18 |
| 6.05 | Bryan Jay | Updated all constants and methods from new documents | 04/03/18 |
| 7.00 | Volen Mihaylov, Bryan Jay | Updated localization algorithm (new algorithm implementation: localizeXBryan(), localizeYBryan() ). Search and localize was left to be completed at a later date | 04/07/18 |
| 7.01 | Volen Mihaylov | Implemented advanced travel to method which localizes at every grid line using localizeXBryan(), localizeYBryan() ) | 04/08/18 |
| 7.10 | Volen Mihaylov | Advanced travel to method fully functional with choice of traveling on the X or Y axis first | 04/08/18 |
| 8.00 | Volen Mihaylov | Integrated full system run. Pathing should ascertain obstacles avoidace | 04/09/18 |
| 9.00 | Volen Mihaylov | Completed and tested full system run. Code is ready for demonstration | 04/10/18 |
| 9.01 | Volen Mihaylov | Added boolean in controller to decide if Search and Localize is to be attempted. Boolean is defaulted at false. | 04/10/18 |

# 8 Glossary

The Application Programmer Interface was designed using JavaDoc comments on Eclipse and generated through it as well. The needed HTML files are included with the weekly reports as demanded. The API is updated as the code changes.