SOFTWARE DESIGN DOCUMENT

**Project:** L*EV3*RON

**Task:** Construct a robot that can play forward or defense in a game of soccer/basketball

**Document Version Number**: 1.6

**Date:** 09-04-17

**Author:** Alexandre Tessier, Ian Gauthier, Ilana Haddad

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Sections Updated** | **Summary** | **Author** |
| 1.0 | 03-03-17 | All | Preliminary version of the document | Alexandre |
| 1.1 | 19-03-17 | 3, 4 | Updated document to reflect progress in creation of the code | Ian |
| 1.2 | 20-03-17 | 3, 4 | Updated the flowchart/class hierarchy | Ilana |
| 1.3 | 28-03-17 | 3, 4 | Updated class descriptions and flow chart | Ilana, Ian |
| 1.4 | 03-04-17 | 4 | Updated class descriptions | Ian |
| 1.5 | 08-04-17 | 4 | New project name, new information to reflect the progress made | Alexandre |
| 1.6 | 09-04-17 | 3, 4 | Updated the class diagram/flowchart | Ilana |

# 

# 1. TABLE OF CONTENTS

[**1. TABLE OF CONTENTS**](#_vvvz5q9y74vr) **1**

[**2. DESIGN PROCESS**](#_sdaaa9jresbm) **2**

[**3. FLOWCHART**](#_2abkhoogt1co) **2**

[**4. CLASS HIERARCHY**](#_6wduxcjtsdo1) **4**

[4.1 WifiExample](#_b4kzpoyt2dgq) 4

[4.2 Defense](#_y63nec6j2gru) 5

[4.3 Forward](#_lo3ftjiga7e0) 5

[4.4 Launcher](#_t9nh94l3gn6j) 5

[4.5 Navigation](#_ngbno6ti6we5) 6

[4.6 Odometer](#_3o6qnx6a3bt9) 6

[4.7 Obstacle Avoidance](#_ijk5pidwwqfs) 6

[4.8 Localization](#_7rfi8n47myh8) 7

[4.9 Correction](#_5p8xknk1fzew) 7

[**5. JAVADOCS**](#_gzunpcx8cpsn) **8**

[**6. EDIT HISTORY**](#_rwuxpd1ton9p) **8**

[**7. GLOSSARY**](#_u0s9ceg9d1b8) **9**

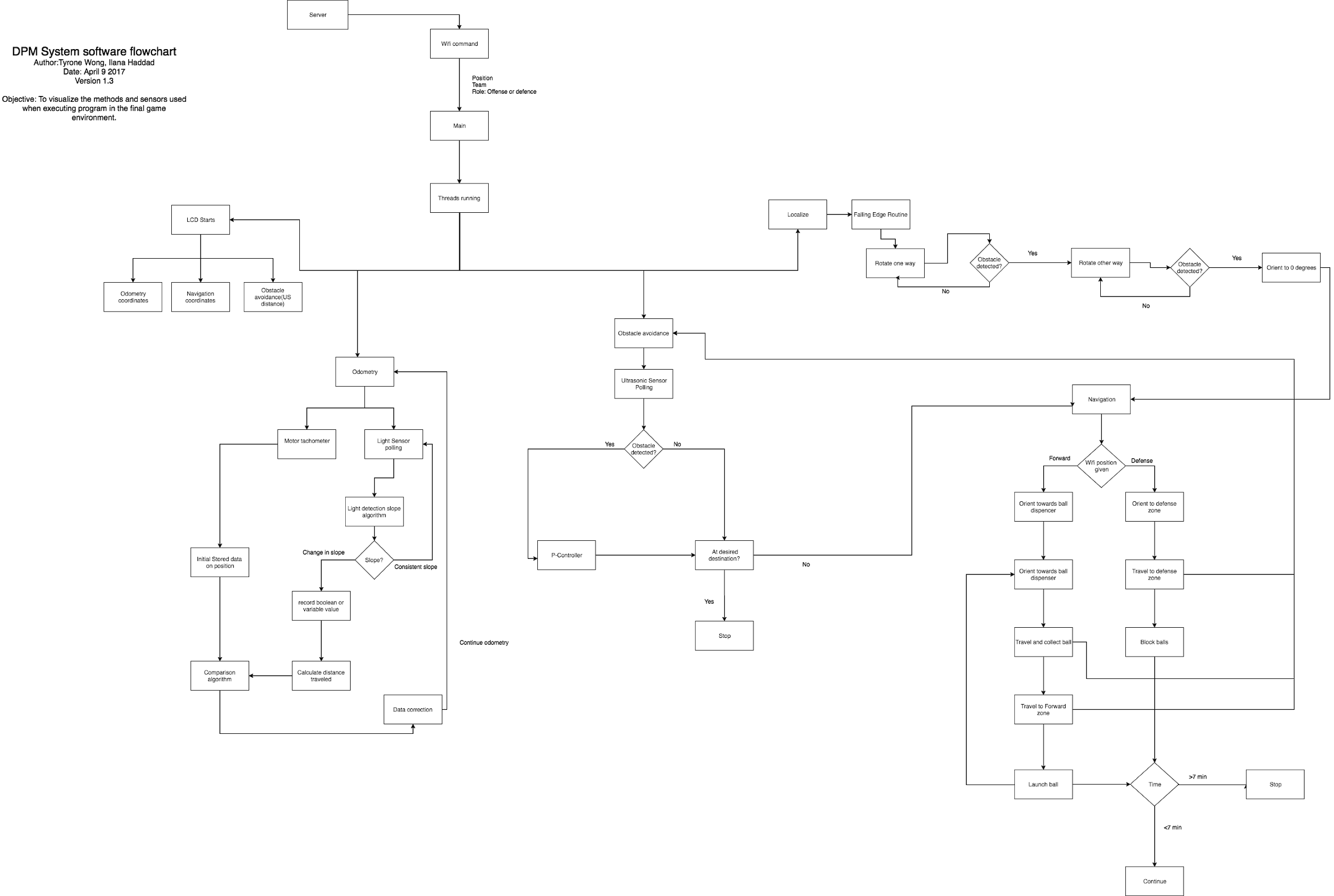
# 2. DESIGN PROCESS

The first step of our design process was to generate a flowchart based on the requirements and the constraints imposed by the design project. This flowchart was then used as a template for our code, as it visually represented the way our classes were supposed to interact with each other in order to achieve our goals. The various classes and methods used all stemmed from a need to meet the requirements laid out in the original diagram.

Once the preliminary version of the code was done, our implemented code entered the testing and debugging phase. We mostly used the Issues feature in GitHub to track and resolve any issues related to the features of our code. Conjointly, as the features such as localization, navigation, etc. were implemented, the testing team was notified in order for them to carry out the appropriate tests to make sure the written code performed as desired. When necessary, crucial feedback was exchanged between the software and testing teams, and the code was improved accordingly.

Throughout the process, a key element was the use of Javadocs (see section 5) as a documentation tool. The descriptions provided along with every class and method ensured that all the team members could easily understand the code. Moreover, documentations played an important role in guaranteeing that someone, in the event that the original author was incapable of finishing a piece of code, would be able to complete what was already started.

# 3. FLOWCHART

Please refer to Section 2 of the Systems Document for a basic flowchart of the overall process. In terms of the software, this translates to: 

*See Software Flowchart\_Week6\_Final.png for a larger version of this flowchart.*

The DPM server is the tool used to trigger the execution of the entire program, providing information that is crucial to the execution of the code. Using the data provided through WiFi, the main class is then responsible of calling the appropriate methods. The flowchart is divided into 4 main categories : odometry, localization, navigation, and obstacle avoidance.

To begin, the robot needs to start an odometry thread in order to know its position relative to where it was originally located. An odometry correction using the light sensor was also implemented, to compensate for any inaccuracies related to factors that cannot be easily compensated for through software.

The first physical task the robot has to perform is to localize in one of the 4 designated corners, regardless of whether it was in offense or defense. To do so, the robot uses the Localization program detailed in section 4.8 of this document. A different type of localization is also performed periodically as the robot moved around the competition field. These differences are, again, detailed in section 4.8 of this document.

Navigation is separated into the defence and offense subsections, each encompassing the tasks associated with its respective role. Although the l*aunch ball* and *block balls* actions fall under the navigation category in this flowchart, they were implemented using separate, independent, classes. Essentially, the navigation class is responsible for getting the robot to a set of coordinates, where it will then perform various tasks based on its role. Depending on whether the robot is in offense or in defense, it will navigate to, respectively, the offense or the defense area.

Finally, whenever the robot is navigating and obstacles are placed on the field (such as in round 3 of the competition), the ultrasonic sensor must continuously be polled to detect any object blocking the robot’s path. The class responsible for obstacle avoidance is required to notice the navigation class that an obstacle has been encountered and that measures need to be taken in order to avoid it. This must be done before the robot can resume navigating towards a set of coordinates.

Overall, the process is as follow:

1. The robot is placed in one of the 4 corners, depending on given instructions.
2. Data is retrieved via wifi and the various objects, motors, and sensors are initialized.
3. Odometry thread is started along with obstacle avoidance.
4. Localization is performed and the robot moves to the origin.
5. Depending on its role, the robot either navigates to the defense zone or proceeds to receive a ball from the dispenser before moving to the forward zone.
6. Periodically localize whenever the robot is moving as well as correct the heading of the robot throughout periods in which it is navigating.
7. Proceed to launch balls or block them depending on the robot’s role.
8. If required, retrieve another ball from the dispenser (only in forward).
9. When time runs out, navigate the robot back to its original location (optional).

# 4. CLASS HIERARCHY

## 

*See Class Diagram\_Week6\_Final.gif for a larger version of this flowchart.*

## 4.1 WifiExample

The wifi class is used to provide the robot with the information that it needs to choose what mode of gameplay (offense or defense) it will be playing in addition to its starting position on the playing field. The code has largely been provided by the clients; the team being required to solely add code that causes the robot to enter one of the two modes depending on the chosen input.

## 4.2 Defense

One of the two roles that the robot could be told to take on during gameplay. These roles can be assumed directly after the initial localization of the robot takes place. The main goal of this portion of gameplay is to navigate to the defensive zone and then attempt to block any balls launched by the opposing robot from entering the target. These actions require the likely simultaneous use of the odometer, navigator and defensive arm movement. Initially, the tactic was to deploy a net in front of the opponent. However, due to time constraints, this idea was discarded. In the end, the robot was programmed to navigate right in front of the target, in the defensive zone, and to raise its catapult arm. This tactic was implemented based on the fact that most teams programmed their robot to navigate right in front of the target before launching in order to minimize the distance the ball had to travel.

## 4.3 Forward

The second of the two roles that can be performed by the robot. This is also performed after the initial localization procedure has been completed. The goal of this procedure is for the robot to navigate to the ball dispenser, receive the ball and place it in the launching mechanism, navigate again to a place from which it can shoot and then shoot the ball through the target. In order to complete these tasks, the robot needs to use the navigation procedure, the odometer and the ball launcher. In addition, the robot needs to be able to avoid obstacles on the playing field. For this, the robot must use the obstacle avoidance class.

## 4.4 Launcher

The procedure by which the ball is received by the robot, moved to the shooter and then launched toward the goal. The robot has to be able to use its position on the field when it is shooting the ball (which can vary due to the lack of knowledge on where the line for the end of the defensive zone is positioned) to determine the speed at which the motor should rotate to propel the ball out of the launcher. Care was taken to ensure that the speeds used were within the constraints of the motor and that variation of motor performance due to the current battery level of the robot were taken into consideration.

The speeds at which the motor used for the launcher had to rotate were determined by trial and error. For all the possible distances, multiple trials were performed, each at different speeds. Whenever the robot successfully launched the ball through the target, the speed was noted, and then a different distance was tested. Based on the current forward line position, the robot chooses the speed at which the motor needs to operate from a list of predetermined values.

## 4.5 Navigation

The procedure that is used to move the robot from one position on the playing field to another. The navigation takes into account the initial positioning of the robot and the rotational motion of the wheels to figure out how far the robot needs to move to reach its destination. The navigation procedure is thus closely related to the odometer. While the robot is moving, it continuously updates the odometer and uses the information provided by the latter to determine the point at which the robot should stop. In addition, the navigation is required to work in tandem with the chosen obstacle avoidance methods that are running continuously to ensure that the robot does not hit an obstacle. When one is detected, the navigation stops while the robot moves around the obstacle and then resumes when the obstacle has been successfully avoided. It must be noted that we decided to restrain the robot’s motion to straight lines. Therefore, the robot cannot travel in diagonals; it must thus travel in x and then in y, or vice versa. This was done in order to minimize the impact of rotational errors on the navigation and to allow for the implementation of odometry correction.

## 4.6 Odometer

The odometer class is used to determine the estimated position of the robot by counting motor rotations. Certain assumptions were made to implement the odometer such as ignoring slip and having a constant wheel radius and wheelbase. In practice, these assumptions do not hold and slowly introduce errors in the odometer. For instance, an issue encountered was that the wheelbase would change once the robot lowered its launching mechanism (See Section 10.4 of the Hardware document for an extensive description of this issue). This would therefore influence the robot’s availability to rotate accurately. We thus had to set a different wheelbase value to be used when the robot would be traveling with its arm lowered. As mentioned above, localization was performed multiple times during navigation around the playing field in order to update the odometer and reduce the effect of these errors as we went along.

## 4.7 Obstacle Avoidance

This method can be used by the roles of defense and forward. As the robot navigates the playing field before reaching their destination, it has to use this method to avoid any blocks standing in its path. The use of the ultrasonic sensor allows the robot to detect obstacles at a certain distance and to, therefore, navigate around them accordingly.

The program uses a single ultrasonic sensor mounted on the front of the robot. When a block or robot is detected in front of the sensor, the robot turns right and left by ninety degrees to find a direction in which there is no other obstacle. At this point, the robot can move one block in that direction to avoid the obstacle before returning to the original heading and, again, checking for any obstacle. If there appears to be one, the robot navigates another tile off of the original line and check again. This process can be repeated until no obstacle is detected in the robot’s path. The robot then moves forward a single tile in the original heading to move past the obstacle before checking to see if it can move back to the original line it was traveling on. It then does so when possible. Moving a single, full tile during each move ensures that the robot remains on top of the grid lines and, thus that the correction algorithm of the robot can be used.

Once this algorithm is completed and the robot is back on its original path, the navigation class is called using the original coordinates the robot was travelling to before avoiding an obstacle. Navigation is thus resumed and the robot travels to its given destination.

## 4.8 Localization

The procedure by which the program can reset its exact heading and position to an exact known point on the playing field. This program is used both at the start to define an initial position of the robot and throughout the match to recalibrate the robot’s position. This therefore minimizes the error in the position of the robot that is accrued through navigating across the playing field.

During the initial running of the robot, the localization program first uses the ultrasonic sensor to determine an overall heading and move the robot to the nearest gridline intersection. It then performs a second localization program using the light sensor and moves the robot to sit with the midpoint between its wheels directly on top of the gridline intersection. After light localization, the robot’s heading is pointed along one of the gridlines. This first localization program needed to be performed within 30 seconds at the start of the round as detailed in section 2.2.2 of the Requirements document. On average, it would take 24 seconds for the robot to localize.

For localization procedures occurring later in the course of the game, the ultrasonic localization program is not performed as the robot should already be close enough to an intersection of gridlines to skip that procedure and perform a variant of the light localization procedure right away. The exact method for localization in this form can be found in section 4.9.

## 4.9 Correction

Correction is a separate class that works in parallel with navigation. The first method called LightCorrection uses the two light sensors at the back of the robot to continuously correct the angle of the robot’s heading. Whenever moving in a straight line, the robot, using light sensors, checks for gridlines. If the light sensor to the right of the robot detects a line before the sensor on the left does, the robot corrects its heading by rotating slightly until the left sensor detects the other line as well, and vice versa.

The correction algorithm also has a localization program which can not only correct the robot’s heading but also its position. After the robot has been calculated to have crossed over 6 lines since the last localization, it uses its two rear sensors to locate grid lines moving in both the x and y dimensions and return the robot to a placement directly on top of the intersection of the grid lines.

# 5. JAVADOCS

Javadocs is the primary tool used to generate our API documentation. Throughout the coding process, comments following the Javadocs format were added to our methods. Our API documentation was then automatically generated, using the specified comments. A roadmap to our code, it includes brief descriptions of our classes and methods, along with how to use them. It is an essential tool used to develop an understanding of how our various classes work together in order to get our robot to operate. It also includes the name of the team members who worked on every method, along with its corresponding version.

# 6. EDIT HISTORY

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Summary** | **Author(s)** |
| 1.0 | 04-03-17 | Imported given Wifi Code | Michael Smith |
| 1.1 | 04-03-17 | Added commands to main in WiFiExample to test program with wifi.  Created classes Defense and Forward | Ilana Haddad |
| 1.2 | 05-03-17 | Imported LightLocalizer method from Lab 4 | Ian Gauthier |
| 1.3 | 11-03-17 | Developed a working Falling Edge Localization | Ilana, Ian |
| 2.0 | 13-03-17 | Developed a working Light Localization followed by a Falling Edge Routine for robot placed in any of 4 corners | Ilana, Tristan, Alexandre |
| 2.1 | 19-03-17 | Developed a complete working Localization program and corrected input values in Navigation program | Ilana, Tyrone, Alexandre, Ian |
| 3.0 | 20-03-17 | Implemented Demo Instructions to navigate to forward line position after localizing, and shoot ball into target | Ilana, Tristan |
| 3.1 | 26-03-17 | Added an odometer correction using an extra 2 light sensors at the back of the robot | Alexandre, Tyrone |
| 3.2 | 27-03-17 | Implementation of localization after navigating a certain distance. | Alexandre, Ilana |
| 4.0 | 01-04-17 | Implementation of obstacle avoidance | Tyrone, Ian, Ilana |
| 4.1 | 03-04-17 | Changes in mechanism of ball launcher leading to changes in when to lower arm, locking it, and firing | Tristan, Ilana, Ian |
| 4.2 | 04-04-17 | Final small changes to avoid intersection of tiles once playing field was set up | Ilana, Alexandre, Tristan |

# 7. GLOSSARY

API : Application Programmer Interface, involves providing descriptions of methods, classes, etc. in order to paint a general portrait of how the different components of the code work together.

Wheelbase: Distance separating the center of the two wheels. It is used mainly in the odometer and in navigation, where it plays a key role in the calculation of the amount of degrees each motor needs to turn in order for the robot to rotate by a certain angle.