SYSTEM MODEL

**Project:** Obstacle track-racer – A competitive robot design project

**Task:** Design and implement a working model out of EV3 Lego sets that would autonomously navigate to a racetrack on an island shown in the world map and complete as many laps as possible within the 5-minute time limit, eventually returning to its starting point.

**Document Version Number**: 1.2

**Date:** 13/03/2021

**Individual Responsible for the Document:** Angelina Duan

**Edit History:**

|  |  |  |  |
| --- | --- | --- | --- |
| Version Number | Author(s) | Date | Content |
| 1.0 | Angelina Duan | 11/03/2021 | The initial version of a block diagram, a functional diagram, and a mechanical block diagram |
| 1.1 | Lide Cui, Shichang Zhang | 12/03/2021 | The initial version of a class diagram, a state diagram, and a software structure diagram |
| 1.2 | Angelina Duan | 13/03/2021 | Complete the initial version of the system model document |

1.0 TABLE OF CONTENTS

[2.0 SYSTEM MODEL](#_heading=h.gjdgxs) [2](#_heading=h.gjdgxs)

[3.0 STRUCTURES](#_heading=h.30j0zll) [5](#_heading=h.30j0zll)

[4.0 HARDWARE](#_heading=h.1fob9te) [5](#_heading=h.1fob9te)

[5.0 SOFTWARE](#_heading=h.3znysh7) [5](#_heading=h.3znysh7)

[6.0 COMPATIBILITY](#_heading=h.2et92p0) [6](#_heading=h.2et92p0)

[7.0 GLOSSARY OF TERMS](#_heading=h.tyjcwt) 6

### 2.0 SYSTEM MODEL

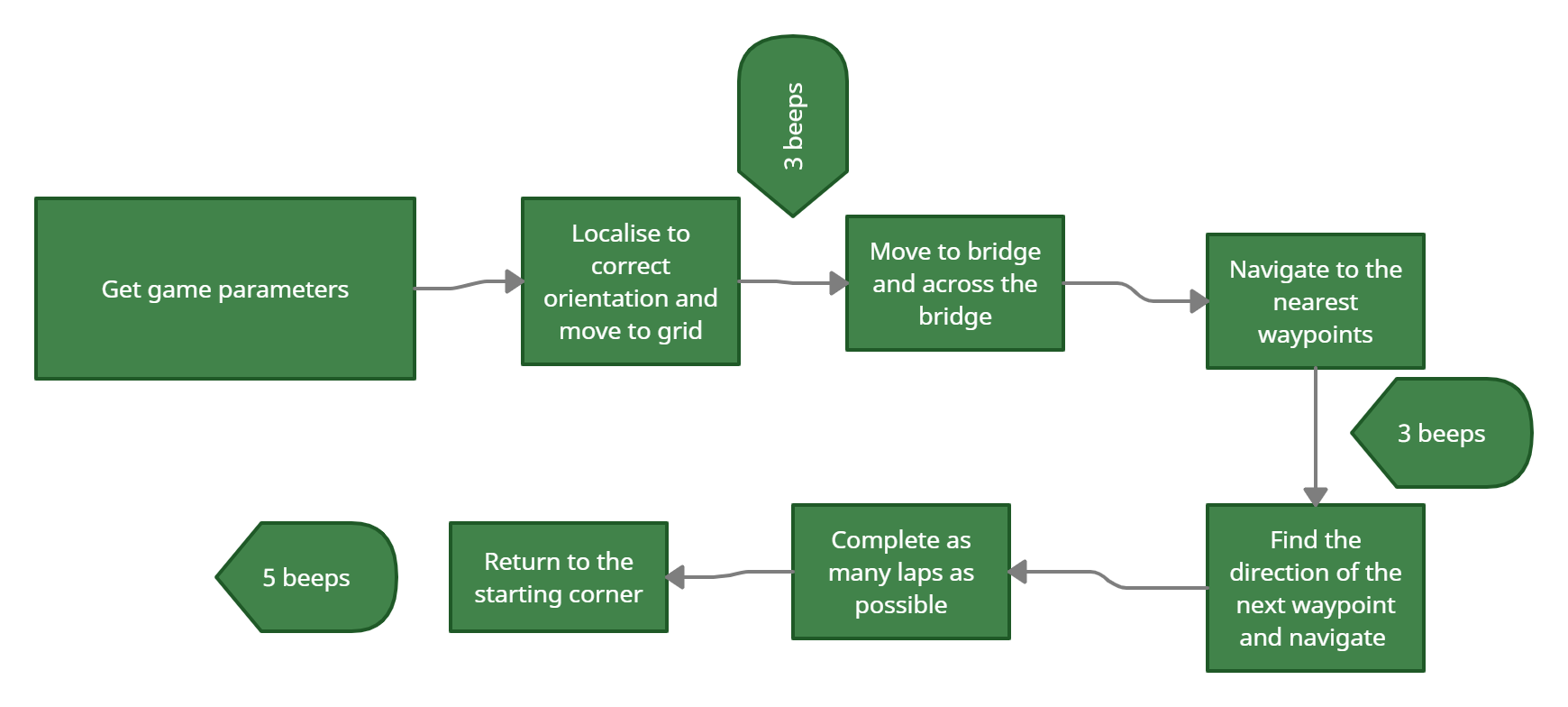


Figure 1. The block diagram

This functional diagram represents a high-level overview of the behavior our robot is going to implement to complete its task.

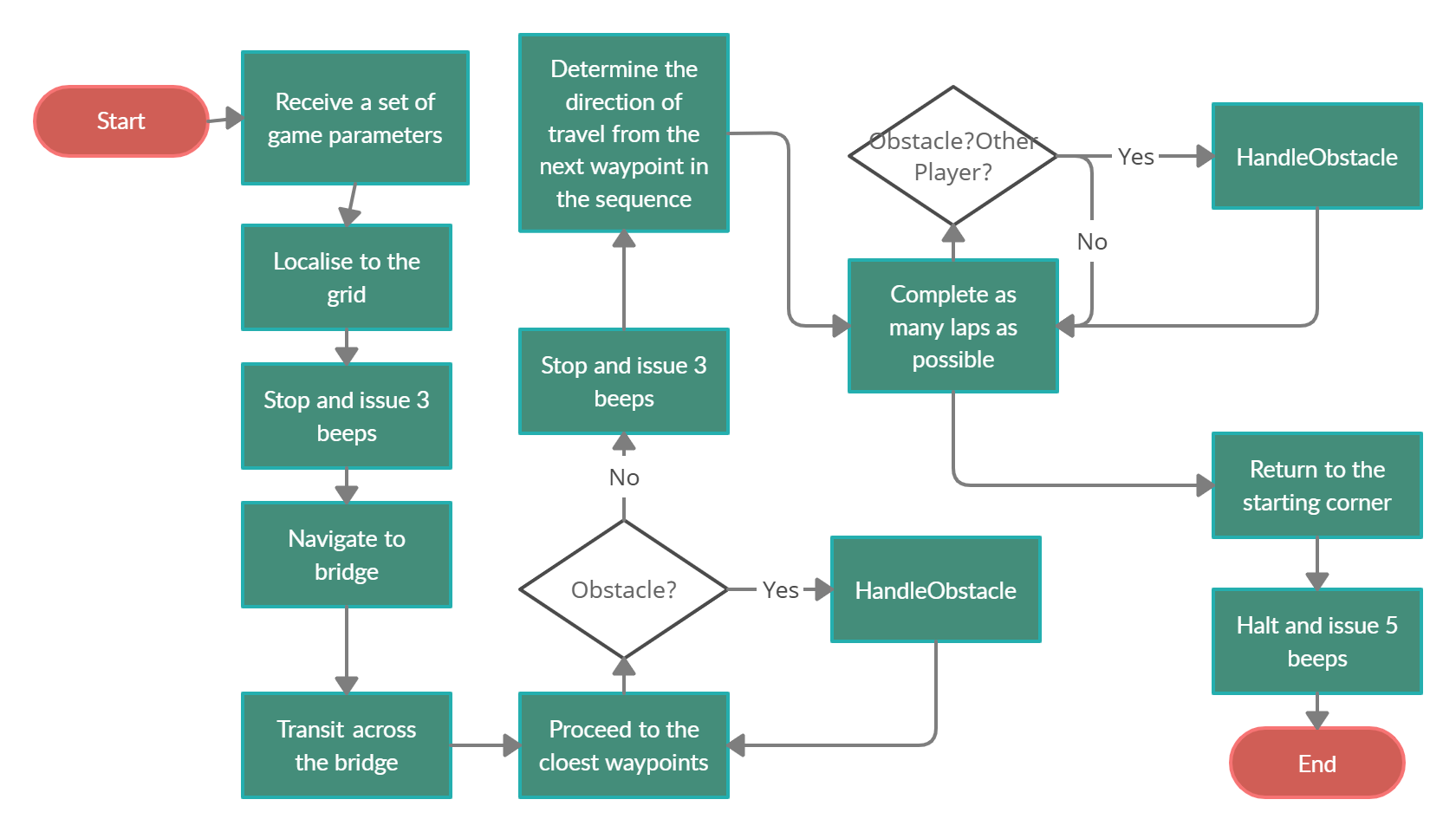


Figure 2. the functional diagram

The following figure shows different states of the machine with the activities that happened in each state.

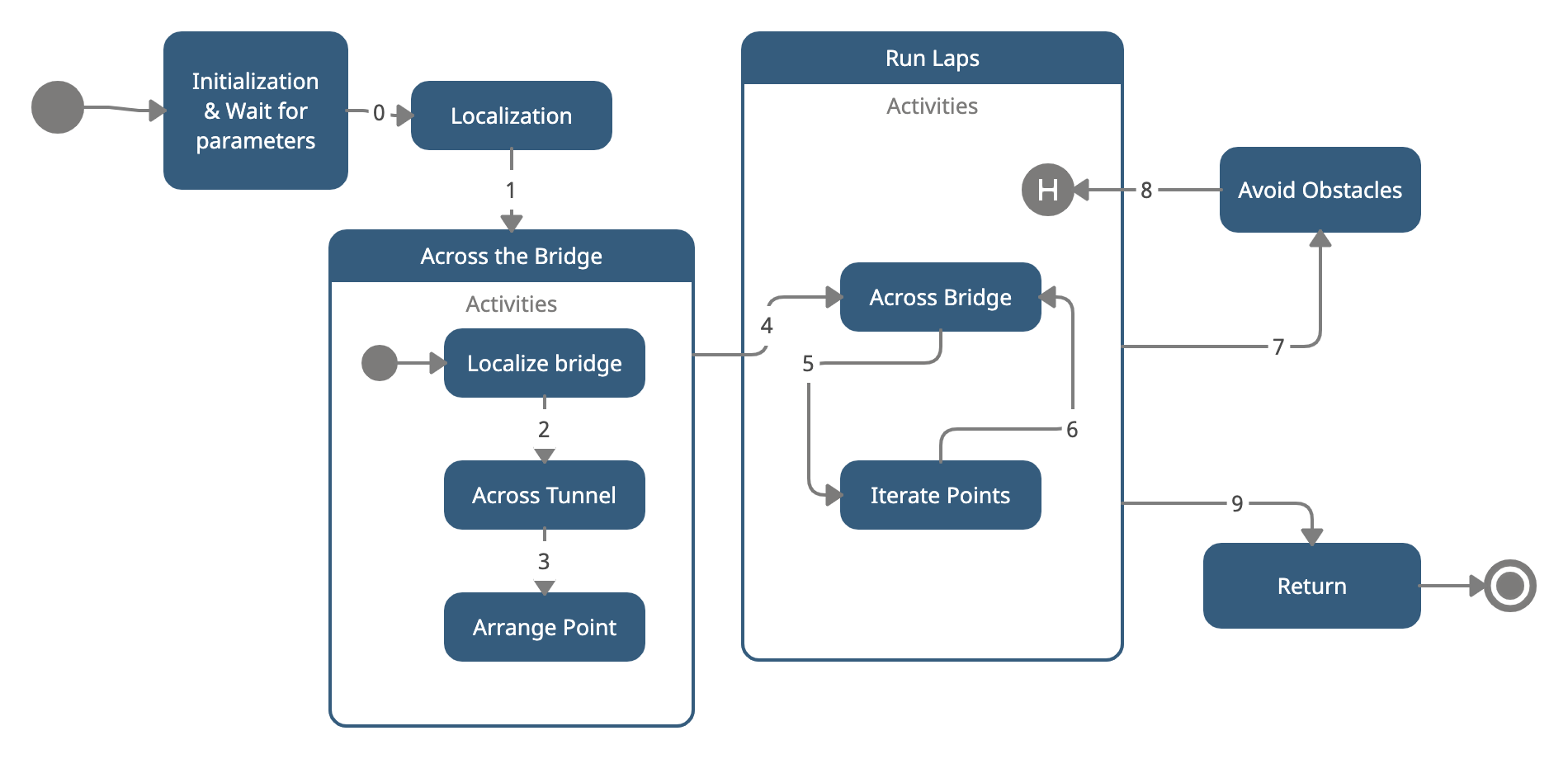


Figure 3. The state diagram

To construct the mechanical block diagram. We need to analyze what Hardware components we need, the following is the list:

* EV3 brick
* Two large motors and two wheels
* Ball Caster (One or Two)
* Two Color Sensors

Detect the grid lines and race track

* One Ultrasonic sensor

Use for Localization and detecting obstacles and walls

We then build up the following diagram showing the components we need and what are they used for.

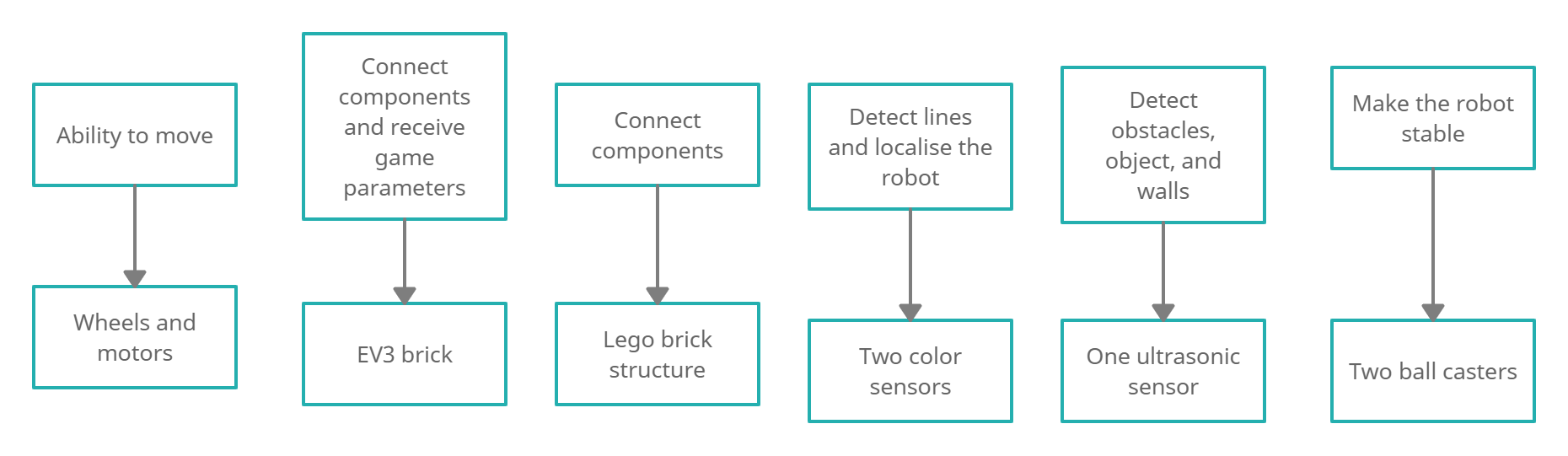


Figure 4. The mechanical block diagram

To construct the software structure diagram, we need to find the classes we need.

Classes that directly inherited and modified: Odometer, Navigation, UltrasonicLocalizer, LightLocalizer, Resources.

Classes that we need to add: GameParameter, HandleObstacle (see if we need this)

Then we build up the following structure diagram which is related to the workflow.

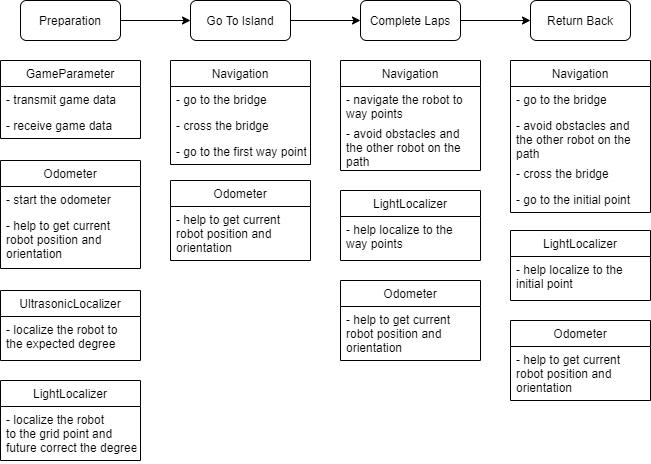


Figure 5. software structure related to the workflow

From the software structure diagram, we see the relationship between each class and add the methods to form this initial version of the class diagram.

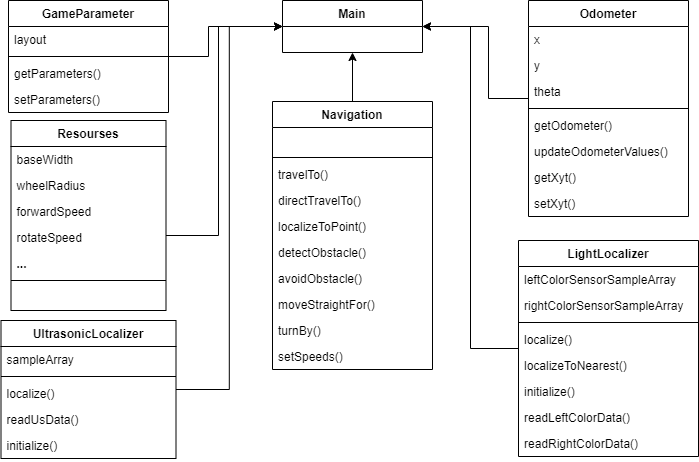


Figure 6. detailed class diagram

### 3.0 STRUCTURES

Hardware structure: The main structure of the robot is an EV3 brick that connects two wheels, two motors with two color sensors, and one ultrasonic sensor. Two ball casters are also added to the structure. The reasons why choosing these components are shown in Figure 3- The mechanical block diagram. The number of these components is controlled by the capability of hardware design which is shown in part 4.0 (refer to HARDWARE).

Software structure: The structure of the software is shown in Figure 5. We have the following classes that can be inherited from Lab 5: Odometer, UltasonicLocalizer, LightLocalizer, Navigation. We just need to modify them to fit the needs of the client. Furthermore, two other classes should be added, are GameParameter for getting and setting parameters and HandleObstacle if there are so many cases of obstacles then we will add this class in the future. The capability of software components is shown in part 5.0 (refer to SOFTWARE).

### 4.0 HARDWARE

As we use LegoCAD to construct the hardware design and export the bricks to Webots, components are limited to those available in the Lego Mindstorms EV3 kit (refer to REQUIREMENT 2.2). In Webots, one EV3 brick has only 8 ports such that we can use up to 4 motors and 4 sensors for one robot that we assume one port is used to detect data of one motor use one sensor. The robot needs to be stable to avoid any obstacles that we need to keep the center of the robot equal to its center of rotation. To handle this requirement, we need to take into account the mass and the size of our robot so there will be limitations.

The speed of motors could not be too fast such that the robot will move at an imaginary speed which is the electromechanical limitation.

As what is shown in the software structure (refer to STRUCTURES), we only need two light sensors(for localization and relocalization) and one ultrasonic sensor (for localization, wall detection, and object detection). The functionality of the EV3 brick also has limitations like the sampling rate and the speed of receiving game parameters.

### 5.0 SOFTWARE

The software components that we use in this project are UltrasonicLocalizer, LightLocalizer, Odometer, Navigation, GameParameter, and Resources. Their detailed usages are described in Figures 5 and 6 (refer to 2.0 SYSTEM MODEL). They do not have capabilities that we need to consider. Instead, we should consider the capabilities of the software tools and languages we use.

The tools that we use are LeoCAD, Webots, and Eclipse. We use LeoCAD to create the base of the robot and form the initial design. However, compared to the real-world Lego kits, we need to use a converter to make the file usable in Webots. Webots is a simulation software that creates virtual setups to replace the real-world environment. Its capabilities depend on what is available in the “protos” folder. Furthermore, only the environment that could be constructed by the tools in Webots is available. The language we use is Java and the IDE we use right now is Eclipse. Ecplise IDE could be replaced by some other IDEs like Visual Studio Code and IntelliJ. However, Java could not be replaced by some other low-level languages like C. If we use C to write code for the robot, we need to understand deeper into the construction and detailed functionality of each part which will cost us more time to finish a project. Compared to Java, C programs tend to be faster and lighter for some functionalities but it is not a good fit for our project.

### 6.0 COMPATIBILITY

The components that can be re-used and the compatibility with third-party products are written in REQUIREMENT DOCUMENT 4.1 and 4.2 respectively. The following is a list of classes and their functionalities that can be re-used in detail:

**UltrasonicLocalizer:** This localization class could be reused since we need it to find the correct orientation of the bridge and the direction to the next waypoints. It also could be used to avoid obstacles.

**LightLocalizer:** This localization class could be reused for the initial localization to the grid and the localization to the waypoints.

**Odometer:** This class is directly connected to the resources class which could be used as a support. It helps to get the current position and orientation of the robot.

**Navigation:** This class is used for all navigation processes as the robot needs to navigate through the racetrack. Also, this class should contain methods of avoiding obstacles and another robot.

### 7.0 GLOSSARY OF TERMS

*LeoCAD:* The CAD software used in ECSE211 to construct the Lego robots.

*Webots:* The simulation software used in ECSE211 to simulate the playing field and the robot.

*proto:* The folder that contains the electrical components that we need to use in Webots like the EV3 brick, the sensor, etc.