**Software document**

**Author :** Maxime Cardinal

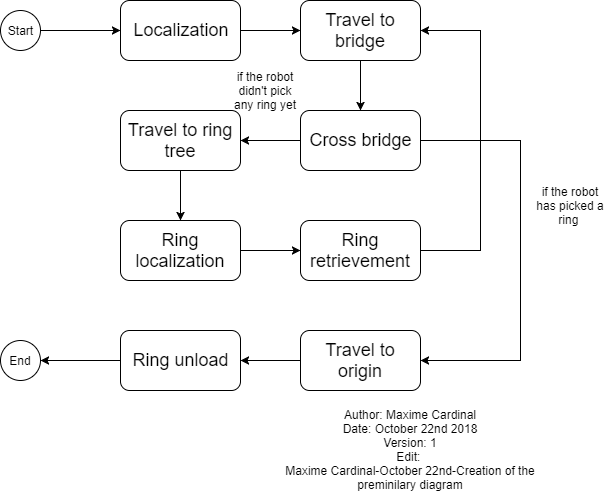
**Date:** October 30th, 2018

**Version:** 1.0

**Edit history:** Maxime Cardinal - October 30th, 2018 - Creation of the document, preliminary content

1. **Functionality**

The robot must execute many independent tasks to reach its goal. First, the robot must localize itself in the grid, then travel to the tunnel, cross that tunnel, reach for the ring tree, localize the rings and retrieve one, then travel back to its original position via the tunnel and unload the ring (see Figure 1).



**Figure 1 - Functionality flowchart**

1. **Classes** 
   1. **DesignProject**

The DesignProject class is the main class of the project. It is responsible for lunching the program, initiating all sensors, all motors and some constants, as well as starting required threads. We decided to initialize all sensors and motors in this class to minimize our time consumption when starting new threads. We make use of the “.join()” method to make sure each thread is executed in the desired order.

**2.2 OdometerData**

The OdometerData class is responsible of keeping track of the robot’s location and orientation. It stores and provides a save access to the odometer data. It contains methods such as “getXYT()” , “update(double dx, double dy, double dtheta)”, “setXYT(double x, double y, double theta)” , “setX(double x)”, ” setY(double y)” and “setTheta(double theta)” which can be used to access odometer data easily. This class has been reused from previous lab.

* 1. **Odometer**

The Odometer class is responsible of updating the odometer data according to the robot’s wheels displacement. This class extends the OdometerData class and has been reused from the previous lab we did. The Odometer class is running as a thread by the main class DesignProject.

* 1. **OdometerExceptions**

The OdometerExceptions class is used to handle errors regarding the singleton pattern used for the odometer and OdometerData. This class has been reused from the previous lab we did.

* 1. **Localization**

The Localization class is responsible of correcting the initial position and orientation of the robot to be (0,0) and 0-degree starting from the y-axis. This class has been implemented using the class UltrasonicLocalizer and LightLocalizer from Lab5. The UltrasonicLocalizer class was responsible of correcting the robot orientation, making use of an ultrasonic sensor and the LightLocalizer class was responsible of correcting the initial position of the robot making use of a light sensor. We decided to merge these two classes into one to minimize the time consumption of the process. By merging these classes, we reduce the number of threads needed by on, thus reducing the time needed to initialize sensors, variables and constants. This class is running as a thread by the main class DesignProject.

* 1. **TravelToTree**

The TravelToTree class will be responsible of travelling the robot from the origin to the tunnel, crossing the tunnel and travelling the robot to the ring tree. This class will be separated in three different tasks and will stop once each of them is executed. To implement this class, we will make use of methods implemented in previous labs such as “travelTo(double x, double y)”. It will require 2 light sensors, which will be used for position correction, and 2 large EV3 motors used for travelling. This class is running as a thread by the main class DesignProject.

* 1. **RingRetrieval**

The RingRetrieval class will be responsible of localizing the rings in the tree, retrieving a ring from the tree and identifying the color of the ring. This class will be separated in three different tasks and will stop once each of them is executed. To implement this class, we will make use of methods and constants used in the ring identification class from Lab5. It will require a light sensor, which will identify the ring color, and a large EV3 motor, which will lift an arm for the ring retrieval. This class is running as a thread by the main class DesignProject.

* 1. **TravelBack**

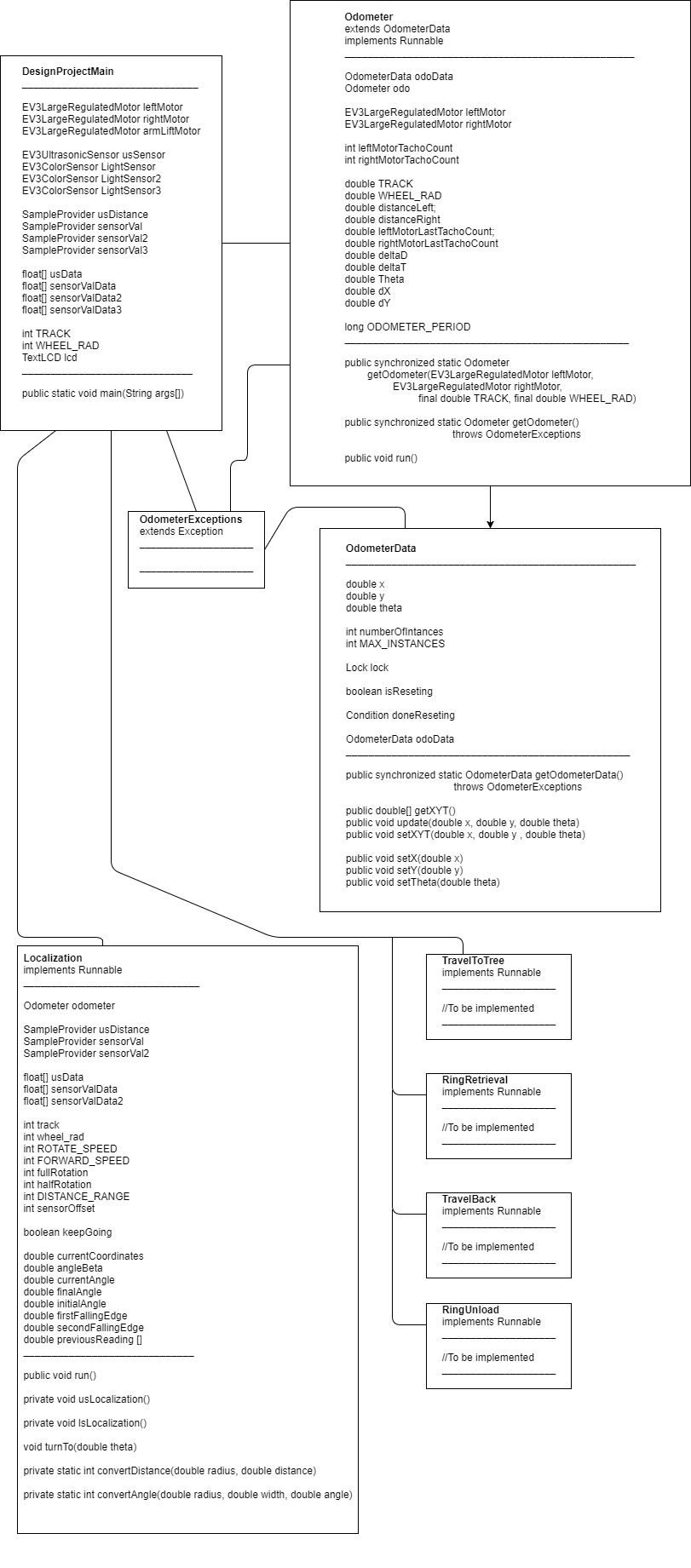
The TravelBack class will be responsible of travelling the robot back to the point (0,0) of the grid. It will execute the opposite function as TravelToTree. To implement this class, we will make use of methods implemented in previous labs such as “travelTo(double x, double y)”. It will require 2 light sensors, which will be used for position correction, and 2 large EV3 motors used for travelling. This class is running as a thread by the main class DesignProject.

* 1. **RingUnload**

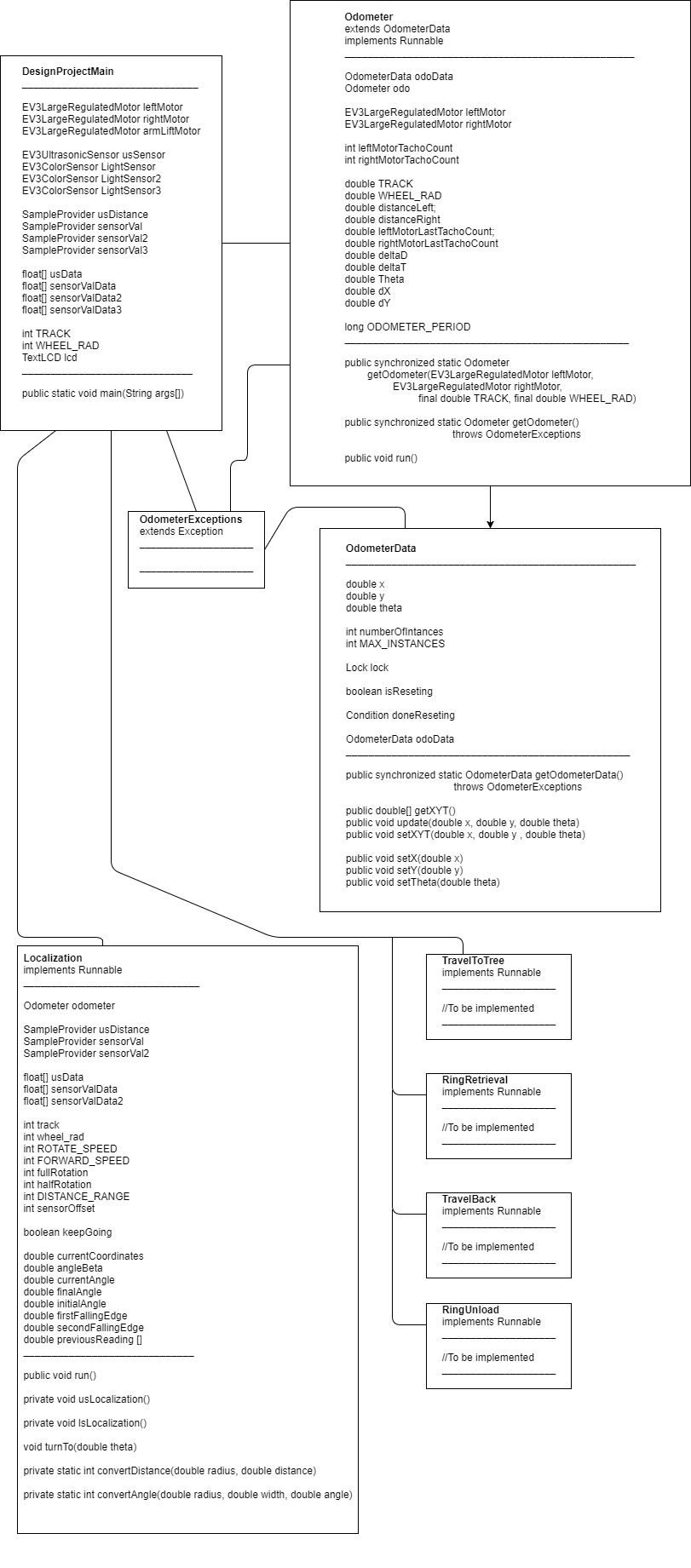
The RingUnload class will be responsible of unloading the retrieved ring in the starting corner of the grid. It will require a large EV3 motor to control the arm and drop the ring. This class is running as a thread by the main class DesignProject.

1. **Class diagram**

The following figure (Figure 2) represents how the classes will interact with each other and shows their content.



**Figure 2 - Class diagram (Part a)**



**Figure 3 - Class diagram (Part b)**