Software Design Document

Project: DPM Final Project

Document Version Number: 3.0

Author: Zakaria Essadaoui

Carl El Khoury

EDIT HISTORY:

[29/10/2018] Zakaria: Created the document + added flow chart.

[30/10/2018] Zakaria: Added UMLs and class hierarchy+ comments.

[05/11/2018] Zakaria: Added the Week 3 subsection on the document.

[10/11/2018] Carl: Added the Week 4 subsection on the document.

[13/11/2018] Zakaria: Added the flow chart for the line detection

[13/11/2018] Carl: Added the flow chart for the ring grabbing routine

Tables of contents:

A. Week 2:

1. Overview:

Our software design will be mainly constructed from the subparts that were developed during the research phase of the project in the labs. We will be using all those pieces (with minor changes) in addition to new classes to perform the tasks required. Since the final project doesn't require searching, we will not be using that part from Lab5, we will only be using the color detection feature from this lab. We will also need the following parts: Odometer, Localization and Navigation.

2. Navigation:

This subsystem is used to go to a desired point in the map. It always interacts with odometer class to get its position on the grid to know by how much it should move. In our final project, this system will be used in a similar fashion in order to navigate to the tree and back to our starting corner.

In order to visualise this system, we included the UML diagram that we had from the Navigation Lab:

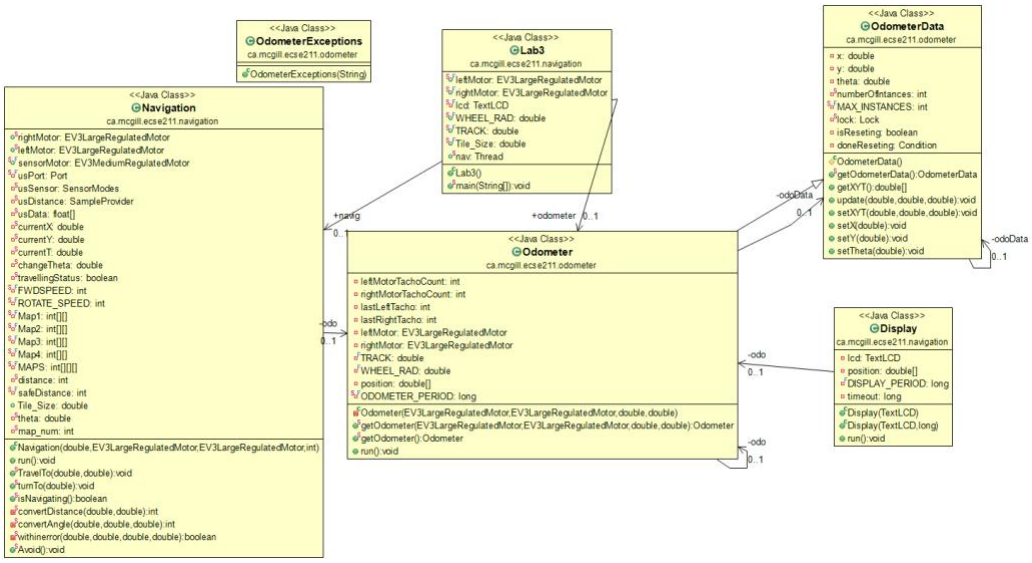

Figure 1: UML diagram from the Navigation Lab

3. Localization:

This system is used to perform the ultrasonic and light localization. In this lab, we had two ways of performing the ultrasonic localization (falling edge and rising edge). Since we don't use both ways and falling edge works better in our design, we decided to delete the rising edge part. For the light localization, we decided to change the design to a use two light sensor instead of one, we noticed that this gave us more precision. We also decided to add poller classes to have

cleaner code structure. By doing so, we would have three pollers for these tasks, “LeftLightPoller”, “RightLightPoller”, “UltrasonicPoller”.

We included the UML diagram from this lab in the following figure:

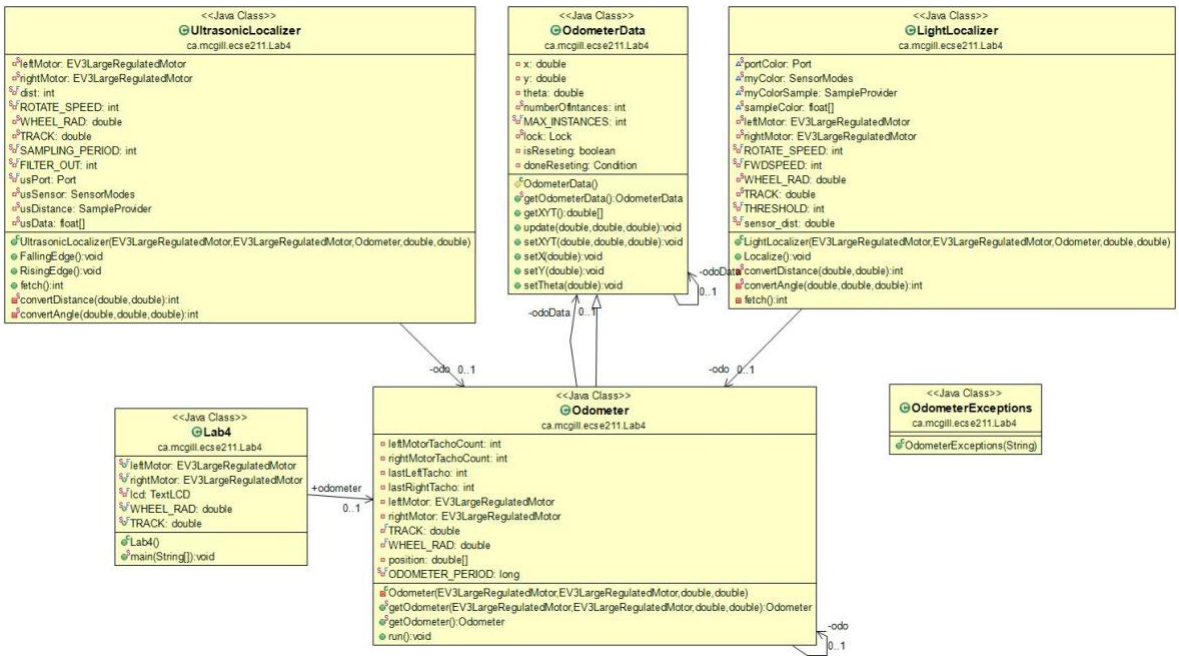

Figure 2: UML diagram from the localization lab

4. Search and Localize:

The main part that will be used from Lab 5 isthe color detection. Basically, we will create an instance of this class and call the detect method to get the ring color. We can see all the fields and classes from this class in the following figure:

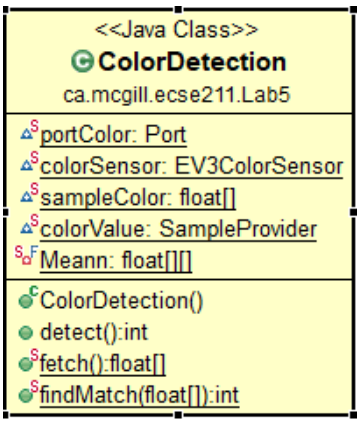

Figure 3: Summary of the Color Detection class

5. Project design:

For now, our software is still simple since we are still writing the initial code and building an effective prototype and thus we don't know what kind of issues we would have to fix with software.

The following flowchart explains the biggest steps of the

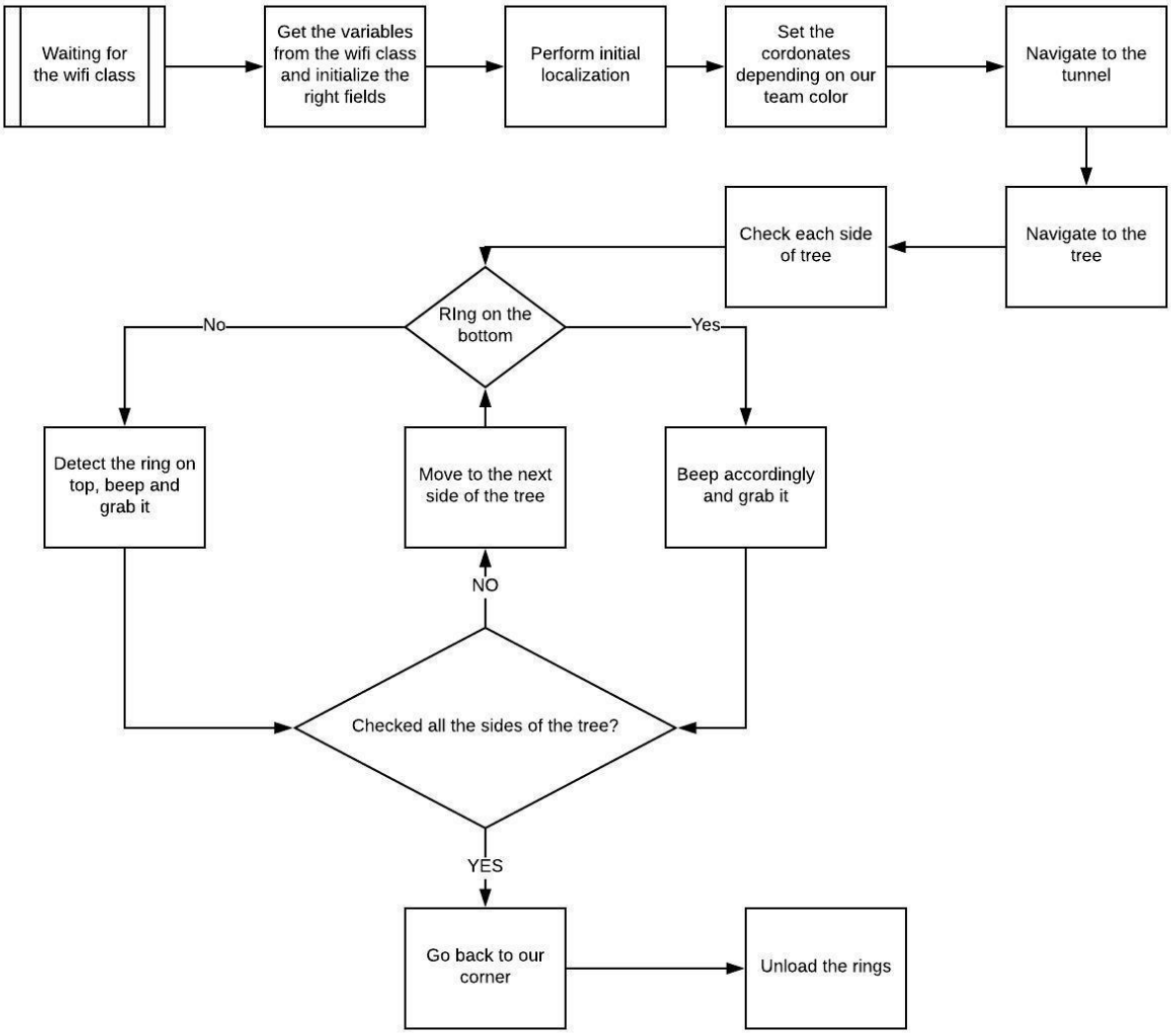

Figure 4: Preliminary flowchart for the project

We also included a preliminary class hierarchy that shows the basic interactions between the class. There are a few changes to note compared to the other labs:

- We are now using poller classes for the sensors instead of implementing them directly in the classes.

- We will be using two light pollers for light localization.
- The navigation class will call the light localization class from time to time

Preliminary class hierarchy:

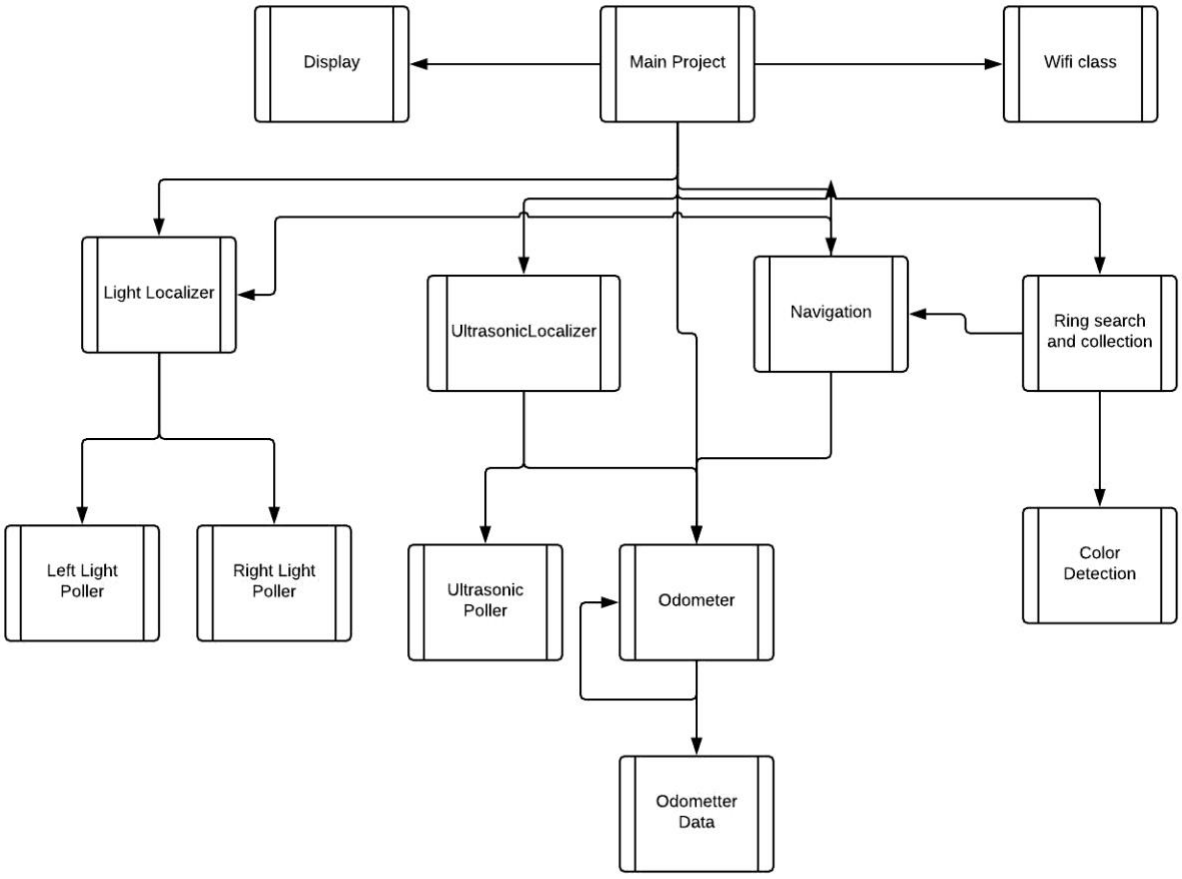

Figure 5: Preliminary class hierarchy for the project

6. Objectives for this week:

- Developing an accurate light localization routine that accounts for all the particular cases where the sensors are on the wrong line since we will be using it while travelling.
- Developing the first prototype of the ring search and collection class when the hardware will be available.
- Dealing with false positives from the color detection class coming from seeing the tree stand as being a yellow ring.

B. Week 3:

1. Overview and progress after week 2:

As was mentioned in the week 2 part, we focused this week on redesigning the software and our classes to get a better structure. We also created the first algorithm for grabbing the rings and ran some tests using it. Some issues with the color detection were discovered (mainly due to the distance of the sensor from the rings and will be dealt with). We will discuss our current software design and all the major changes.

2. Software design as of week 3:

A part of redesigning our software was the creation of two polling classes:

- One for the ultrasonic sensor which takes care of acquiring data from the sensor and also deals with false negative through our filter.
- The second one is for the light sensor, it takes care of the acquiring data from the sensor and detecting the lines using a differential filter.

We also developed the full algorithm for the game play (depending on which team we are); However, some changes must be made to that algorithm since the tree will be put in an intersection instead of the center of a tile.

We also started working on a new algorithm for line detection using the light poller class. This algorithm is still under developed and we are also still discussing whether we should detect black lines using the ratio of readings or the difference of the readings.

The algorithm used for grabbing rings was developed under the tester package. This preliminary code worked well and it was possible to retrieve a ring using. It is not yet implemented in the RingGrabbing class; However, this will be done shortly after the hardware is finalized.

A preliminary API has also been developed and have been generated using Doxygen. It can be accessed in the following folder: Week 3\ API_week3\html. Please select the file named: "_final_project_8java" to access. You will then be able to navigate through all the classes.

3. UML diagram:

The following figure shows the UML diagram of our design:

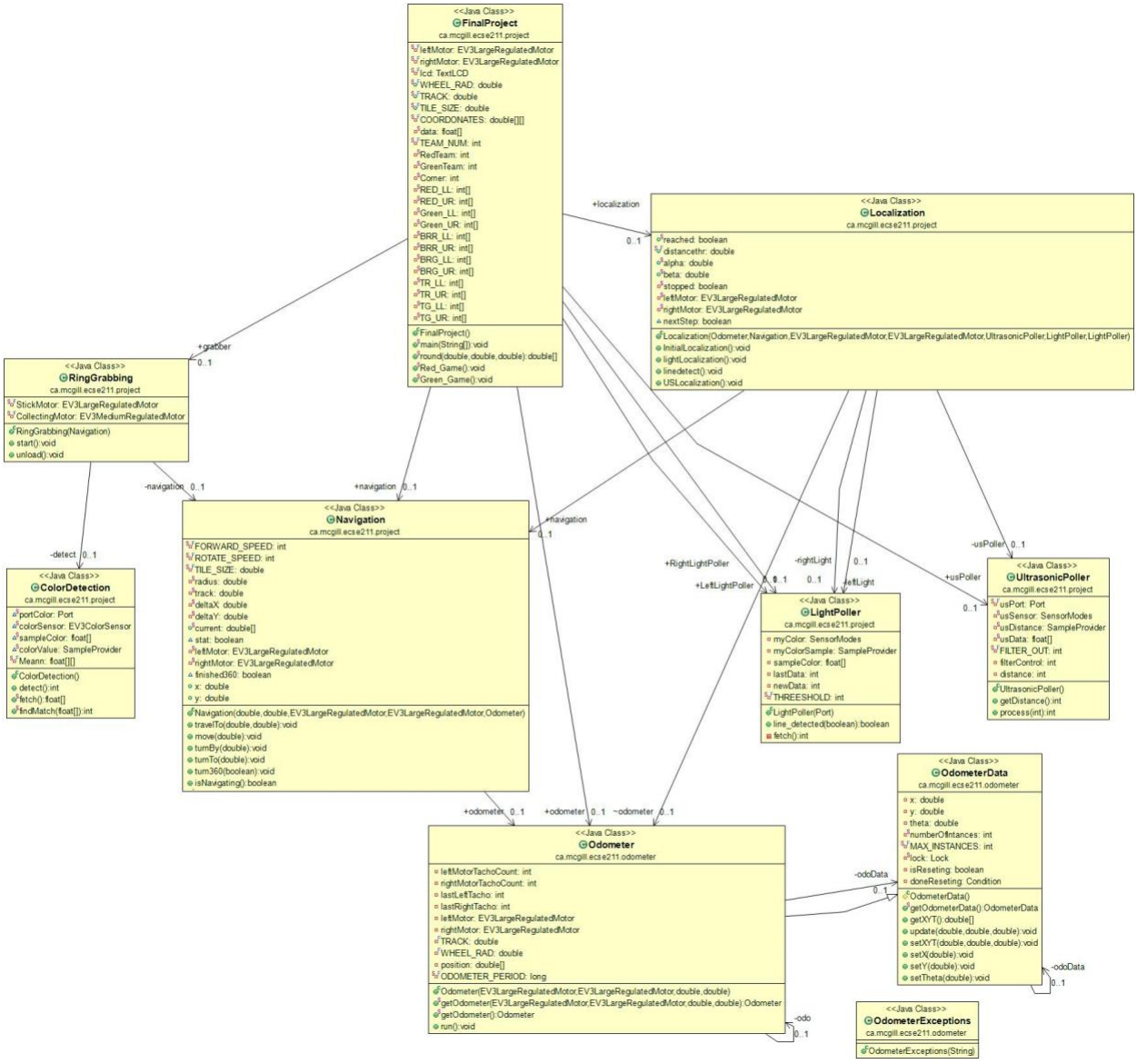

Figure 6: UML diagram of our project as of week 3

This structure is very similar to the class hierarchy proposed in week 2. The Wifi and Display classes will be added later on. The figure above clearly shows all the interactions between the different classes and class hierarchy. One thing to note here is that up to now only the odometer class is running as its own thread. All the other classes are called from the main method. However, this might change if we decide to use the light pollers to detect and correct the odometer while navigation. This design decision will be made after the line detection algorithm is finalized and that more intensive tests are ran on it to know its limitations.

4. Objective for week 4:

For week 4, our main focus will be to prepare a reliable software for the beta demo. To do so, we will be doing the following tasks:

- Finalizing the line detection method as well as a localization algorithm that would be used when navigating.
- Working with the hardware leader to develop a design that will detect the rings without any false readings.
- Building a faster localization algorithm that must finish under 30 seconds.
- Making changes to our initial gameplay algorithm to take into account the new changes in the setup of the competition that have been introduced in the version 2.0 of the project document.

C.Week 4:

1.Method summary:

-Odometer class:

Description: This class is the skeleton of the odometer class.

getOdometer:

Type: public synchronized static
Parameters: EV3LargeRegulatedMotor leftMotor, EV3LargeRegulatedMotor rightMotor, final double TRACK, final double WHEEL_RAD
Throws: OdometerException
Return: Odometer

Description: This method is meant to ensure only one instance of the odometer is used throughout the code.

getOdometer:

Type: public synchronized static
Parameters: None
Return: Odometer
Throws: OdometerException (if no previous odometer exists)

Description: This class is meant to return the existing Odometer Object. It is meant to be used only if an odometer object has been created

run:

Type: public
Parameters: None
Return: None

Description: This method calculates the change in X, Y and Theta and updates them. This is the same method used in the slides with some changes to the angles units.

-OdometerData

Description: This class stores and provides thread safe access to the odometer data.

getOdometerData:

Type: public synchronized static
Parameters: None
Return: OdometerData
Throws OdometerException

Description: OdometerData factory. Returns an OdometerData instance and makes sure that only one instance is ever created. If the user tries to instantiate multiple objects, the method throws a MultipleOdometerDataException.

getXYT:

Type: public
Parameters: None
Return: double[]

Description: Return the Odomometer data. Writes the current position and orientation of the robot onto the odoData array. odoData[0] =x odoData[1] = y odoData[2] = theta

Update:

Type: public
Parameters: double dx, double dy, double dtheta
Return: None

Description: Adds dx, dy and dtheta to the current values of x, y and theta, respectively. Useful for odometry.

setXYT:

Type: public
Parameters: double dx, double dy, double dtheta

Return: None

Description : Overrides the values of x, y and theta. Used for odometry correction.

setX:

Type: public
Parameters: double x
Return: None

Description: Overrides x. Use for odometry correction.

setY:

Type: public
Parameters: double y
Return: None

Description: Overrides y. Use for odometry correction.

SetTheta:

Type: public
Parameters: double theta
Return: None

Description: Overrides theta. Use for odometry correction.

-ColorDetection:

Description: This is our color detection that is used when we are searching for rings

Detect:

Type: public
Parameters: None
Return: int

Description: This method returns the color detected.

Fetch:

Type: public static
Parameters: None
Return: float[]

Description: This method gets the data from the sensor

findMatch:

Type: public static
Parameters: float[]
Return: int

Description: This method return the corresponding color for the RGB values that were passed to it If the RGB doesn't correspond to any known colors, it return 5

-LightPoller:

Description: This class is used to acquire values from the light sensor and detect lines

Line_detected:

Type: public
Parameters: boolean
Return: boolean

Description: This method detects if a line was crossed

fetch:

Type: private
Parameters: None
Return: int

Description: This method is used to get data from the light sensor

-Localization:

Description: This class is responsible for the ultrasonic and light Localizations

InitialLocalization:

Type: public
Parameters: None
Return: None

Description: This method performs the initial localization. It first calls the ultrasonic localization method, then it calls the light localization method and returns.

lightLocalization:

Type: public
Parameters: None
Return: None

Description: This method is used to perform our light localization Make the robot move forward until both sensors detect the line turn by 90degress and does the same in that direction.

linedetect:

Type: public
Parameters: None
Return: None

Description: This method makes the robot move forward until one sensor detect a black line. At that point, only the wheel from the other side keep turning until the it detect a line. It also include mechanisms to deal with extreme cases.

linedetect_move:

Type: public
Parameters: None
Return: None

Description: runs linedetect then backtracks afterwards

USLocalization:

Type: public
Parameters: None
Return: None

Description: This method reads distance values from the ultrasonic sensor and detects rapid changes in the distances during movement. Depending on the direction of the change in value, it is considered a rising or falling edge. Two edges are detected during the sequence and the robots angle at the moment of each detection is used to calculate the angle needed to accurately orient the robot at 0 degrees.

-Navigation

Description: This class is responsible for the calculations of distance and angle to reach desired coordinate by taking in current position information and running set calculations on the info to solve for needed variables.

travelTo:

Type: public
Parameters: double x, double y
Return: None

Description: This method makes the robot travel to a (x,y).It first measures get the position of the robot from the odometer. Then it calculates the angle it should be at and the distance it needs to move by.

move:

Type: public
Parameters: double distance
Return: None

Description: This method makes the robot move by the distance that was passed to it

turnBy:

Type: public
Parameters: double theta
Return: None

Descrition: This method makes the robot turn (clockwise) by the angle that was passed to it

turnTo:

Type: public
Parameters: double theta
Return: None

Description: This method make the robot turn to a certain angle. It also make sure that it uses the minimal angle to turn.

turn360:

Type: public
Parameters: boolean clockwise
Return: None

Description: Instructs the robot to spin 360 degrees in a direction specified by the boolean argument. A boolean true argument would result in clockwise 360 degree turn.

isNavigating:

Type: public

Parameters: None
Return: boolean

Description: This method return whether the robot is navigating or not

convertDistance:

Type: public static
Parameters: double radius, double distance
Return: int

Description: This method calculate the angle that must be passed to the motor using the radius of the wheel and the distance we want the robot to cross.

convertAngle:

Type: public static
Parameters: double radius, double width, double angle
Return: int

Description: This method calculate the angle that the motor must turn in order for the robot to turn by an certain angle.

-RingGrabbing:

Description: This class controls the grabbing mechanism and perform the operations needed for grabbing. It checks for the rings on all the sides of the tree and attempt to grab them

start:

Type: public
Parameters: None
Return None

Description: This method start the scanning and grabbing the rings from the tree. It will check for rings on every side of the tree and grab it if a ring was detected.
One assumption it makes is that it is located in an intersection in front of the tree.

unload:

Type: public
Parameters: None
Return: None

Description: This method is used to unload the rings from the arms. It makes the motor controlling the ring holders rotate forward to make the rings fall.

-UltrasonicPoller

Description: This class is used to get data from the ultrasonic sensor.

getDistance:

Type: public
Parameters: None
Return: int

Description: This method return the distance from the sensor.

Process:

Type: public
Parameters: int distance
Return: int

Description: This is our filter for making sure the distance detected is bigger than 255, i.e. no object in front of the sensor.

-BetaDemo

Description: Main class for the lab demo that contains our main and our method calls

getWifiData:

Type: public static
Parameters: None
Return: None

Description: This method gets data from the wifi class and initiate all the appropriate fields

round:

Type: public static
Parameters: double x, double y, double theta
Return: double[]

Description: This rounding method get the values from the odometer and return the closest grid intersection and orientation This is done after performing light localization

Red_Game:

Type: public static
Parameters: None

Return: None

Description: This method defines the game play in the case where our we are the red team. It controls all the robots movements and operations in order to complete all the tasks required in the project manual.

Green_Game:

Type: public static
Parameters: None
Return: None

Description: This method defines the game play in the case where our we are the green team. It controls all the robots movements and operations in order to complete all the tasks required in the project manual.

3. Flow charts for the main methods:

- Line detection: our line detection routine implements the algorithm that is summarized in the following flow chart:

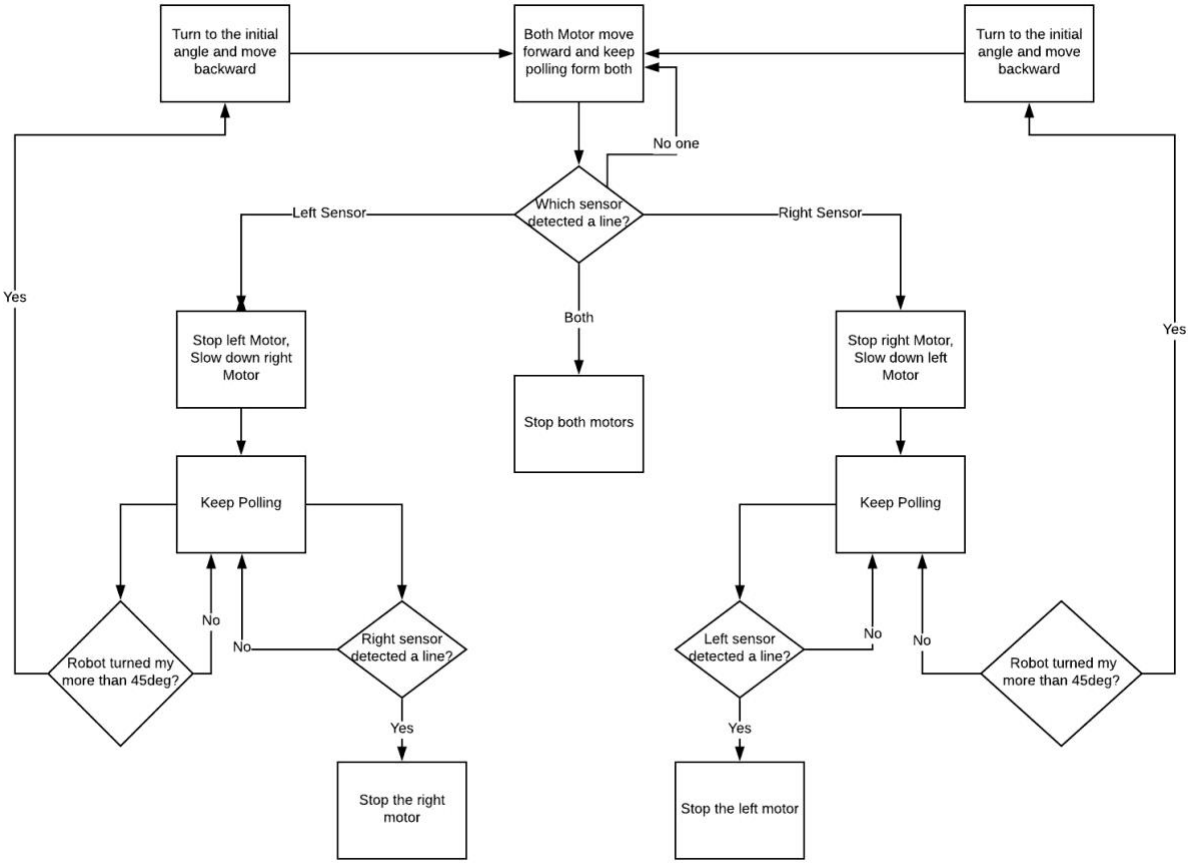

Figure 7: Flow chart for the line detection method

Comments: We added the angle change detection feature to deal with the cases where the poller missed to detect a line.

- Ring grabbing: the algorithm that this method follows can be summarized in the following flow chart:

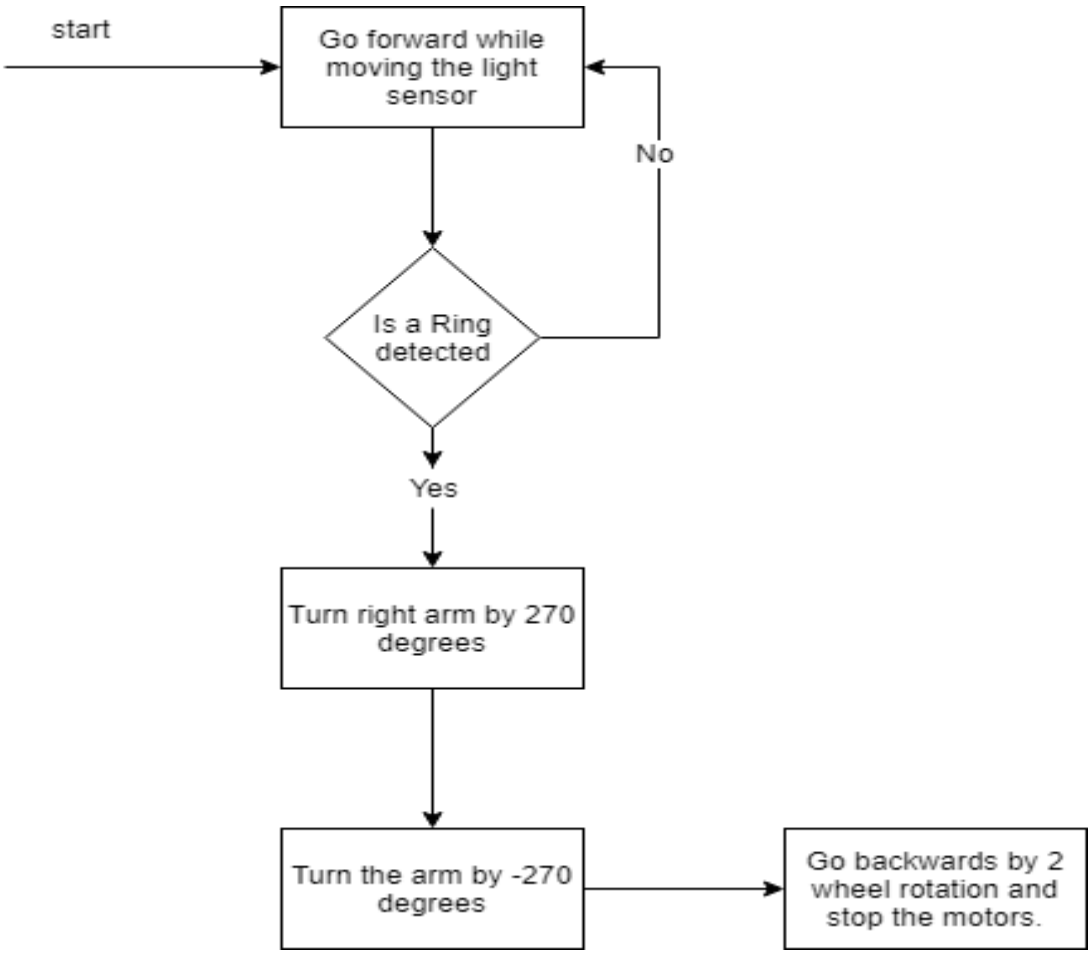

Figure 8: Flow chart for the ring grabbing method for the beta demo