ECSE 211

Lab 3:

Group 21
Theodore Janson (260868223)
Marwan Khan (260762251)

February 7th, 2019

1. <u>Design Evaluation</u>

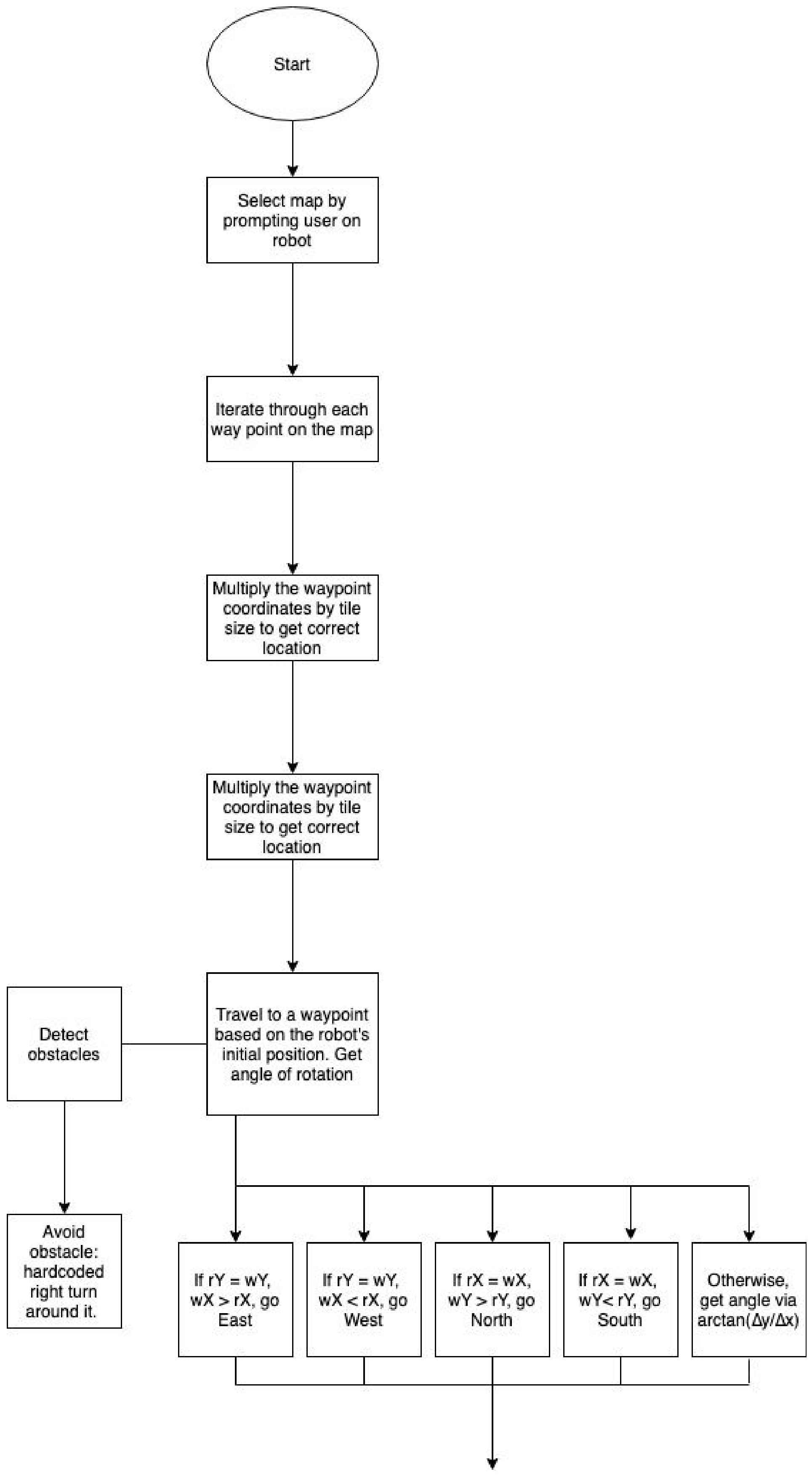
a. Hardware

For this lab, we wanted to change our design. A wider design would be more stable, and likely be more precise in navigating to a specific points he motors used were the Large EV3 motors, placed on either side of the brick, and supported at many points for stability. A marble support was placed at the back of the robot to support the rear half. The ultrasonic sensor was placed far out front of the robot. It is mounted on a EV3 medium motor, as our designed initially used a rotating sensor when objects were detected, but this part of the software was omitted, while the motor remained. To keep all the wires from taking up to much room, which could cause issue when avoiding obstacles, we used a clamp to reduce wire volume and attached this to the side of the robot.

b. Software

The main entry point to the code is the lab 3 class. It contains the initialization of motors, sensors and ports, as well as the main instance variables of the project. Its main method prompts the user with a selection of maps and allows the user to select a map using a do-while loop. The class also contains the creation and starting of odometer, display and navigation classes. The navigation class contains other variables such as the two-dimensional arrays for each map, an instance of the ultrasonic sensor. It also specifies the speed of forward and rotational motion of the motors. In the main run method, the map is selected and its waypoints are iterated through. During each iteration, the TravelTo method is called, which instructs the robot to travel to a location given by the coordinates of the waypoint multiplied by the tile size. Before moving to a point, methods for getting the proper heading to the next waypoint, and rotating to this heading are called. Heading is determined by trigonometry, and the distance needed to travel is determined by the Euclidean distance formula. The motor’s travel methods are called to travel to the waypoint. On the way, if the ultrasonic sensor detects an object in its way, a method of hardcode is called which makes the robot round the object via the right. Other classes include the display class which displays the odometer values on the lcd. The odometer class is used to

get the current location of the robot, which is determined by wheel rotation. The displacement of each wheel is calculated by:
Displacement = (wheel radius)(π)(current tachometer count - last tachometer count) /180.
The total displacement of the robot is the obtained by taking the average of these two values. The location for x and y is then obtained by multiplying this average by the cosine and sine of the heading respectively. The heading is obtained by summing the displacements of the left and right wheels and dividing this by the track (distance between wheels).


Start
Select map by prompting user on robot
Iterate through each way point on the map
Multiply the waypoint coordinates by tile size to get correct location
Multiply the waypoint coordinates by tile size to get correct location
Travel to a waypoint based on the robot's initial position. Get angle of rotation
Detect obstacles
Avoid obstacle: hardcoded right turn around it.
If rY = wY, wX > rX, go East
If rY = wY, wX < rX, go West
If rX = wX, wY > rY, go North
If rX = wX, wY< rY, go South
Otherwise, get angle via arctan(Δy/Δx)

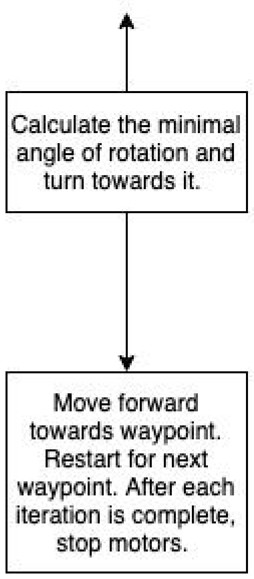
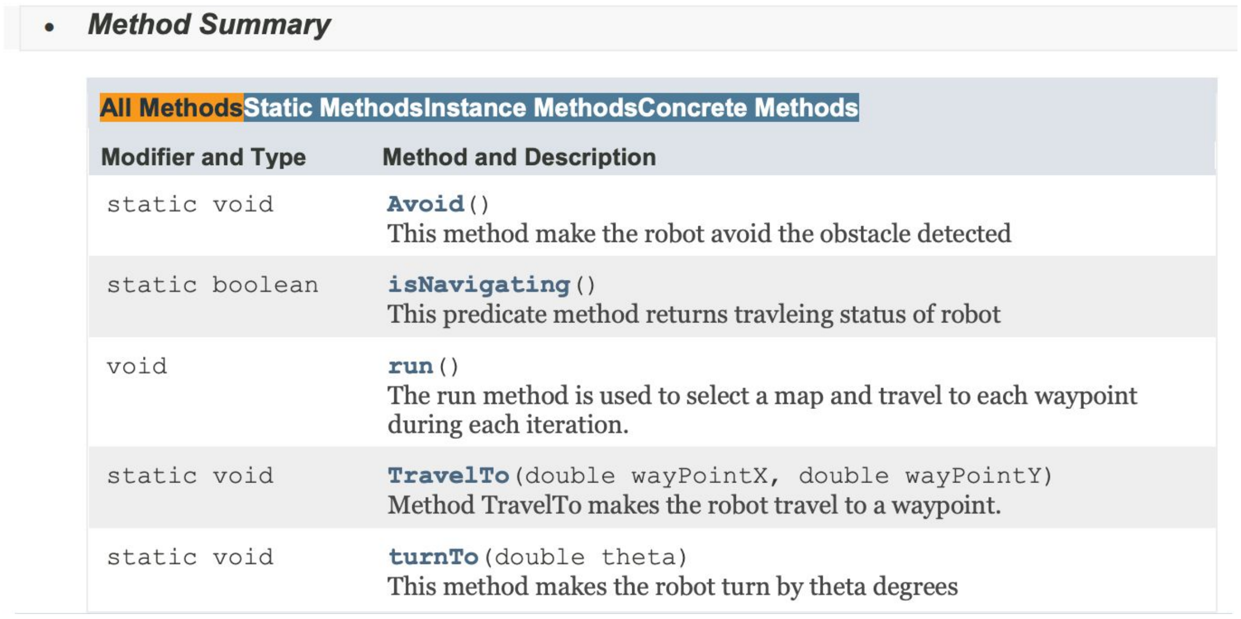

Calculate the minimal angle of rotation and turn towards it.
Move forward towards waypoint. Restart for next waypoint. After each iteration is complete, stop motors.

Figure 2: Navigation class diagram



- ***Method Summary***

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
`static void`	`Avoid()` This method make the robot avoid the obstacle detected
`static boolean`	`isNavigating()` This predicate method returns travleing status of robot
`void`	`run()` The run method is used to select a map and travel to each waypoint during each iteration.
`static void`	`TravelTo(double wayPointX, double wayPointY)` Method TravelTo makes the robot travel to a waypoint.
`static void`	`turnTo(double theta)` This method makes the robot turn by theta degrees

Figure 3: Navigation Java docs

3. Test Data

Trial Number	Final Position - X_F (cm)	Final Position - Y_F (cm)	Destination Waypoint - X_D (cm)	Destination Waypoint - Y_D (cm)	Error - X (cm)	Error - Y (cm)
1	68.20	-1.50	67.49	-0.72	0.71	-0.78
2	59.80	-1.10	61.11	-0.88	-1.31	-0.22
3	62.50	0.50	61.01	-0.89	1.49	1.39
4	61.50	-4.20	64.78	-1.23	-3.28	-2.97
5	66.00	-0.90	61.08	-1.97	4.92	1.07
6	57.30	0.80	60.34	0.34	-3.04	0.46
7	62.50	-2.30	60.12	-1.51	2.38	-0.79
8	63.20	-2.70	62.24	-1.62	0.96	-1.08
9	64.40	0.40	63.79	-1.71	0.61	2.11
10	63.30	-1.50	61.56	-0.76	1.74	-0.74

2. Test Analysis

Trial Number	Euclidean Error - $\in N$
1	1.0548
2	1.3283
3	2.0377
4	4.4249
5	5.0350
6	3.0746
7	2.5077
8	1.4450
9	2.1964
10	1.8908

Mean -x: 2.4995

Standard Deviation - σ: 1.321

Sample Calculations:

Euclidean Error:

$$\in = \sqrt{(X_D - X_F)^2 + (Y_D - Y_F)^2}$$

$$\in_1 = \sqrt{(60.12 - 62.50)^2 + ((-1.51) - (-2.30))^2}$$

Euclidean Mean:

$$\bar{x} = \frac{\sum_1^N \in_N}{N}$$

$$\bar{x} = \frac{2.5077 + 1.3283 + 5.0350 + 1.4450 + 3.0746 + 1.8908 + 4.4249 + 2.1964 + 1.0548 + 2.0377}{10}$$

Euclidean Standard Deviation:

$$\sigma = \sqrt{\frac{\sum_1^N (\epsilon_N - \bar{x})^2}{N-1}}$$

$$\sigma = \sqrt{\frac{\begin{array}{c}(2.5077 - 2.4995)^2 + (1.3283 - 2.4995)^2 + (5.0350 - 2.4995)^2 + (1.4450 - 2.4995)^2 + (3.0746 - 2.4995)^2 + \\ (1.8908 - 2.4995)^2 + (4.4249 - 2.4995)^2 + (2.1964 - 2.4995)^2 + (1.0548 - 2.4995)^2 + (2.0377 - 2.4995)^2\end{array}}{9}}$$

3. Observations and Conclusions

a) Are the errors you observed due to the odometer or navigator? What are the main sources?

Both the odometer and the navigator are sources for the errors observed. Readings in the odometer can be falsified by errors such as motor timing issues or tire slippage. Over large distances, this can be add up to considerable errors in the robot's perceived location. This yields a false value for the robot's current position variable used in the navigator class, which can in turn yield incorrect decisions regarding heading and distance to the next waypoint. The formulas for these calculations both depend on the robot's current position and the waypoint position: heading = arctan(y/x) and Euclidean distance formula.

b) How accurately does the navigation controller move the robot to its destination?

More or less accurately. The navigator controller as an mean Euclidean error of 2.5, which is not too bad, considering error less than 2cm is considered negligible . Generally, the error in y is less than the error in x.

c) How quickly does it settle (i.e. stop oscillating) on its destination? The robot stops oscillating right away when it reaches its destination. This means that the odometer worked pretty accurately and that the wheel rotation was pretty reliable.

d) How would increasing the speed of the robot affect the accuracy of your navigation? Generally, increasing the speed of the robot would decrease the accuracy of navigation because of slip. However, if it were possible to guarantee zero slip, increasing the speed of the motors would be preferable because there would be less error in the timing of the motors. The odometer would therefore be more accurate. However, slip causes a greater error, so it is preferable to move a bit slower. Also, the acceleration and deceleration of the robot decreases the performance of the odometer since the increase in displacement is not linear. Since the robot has a greater top speed, in needs a greater acceleration, which would result in inaccuracies.

4. <u>Further Improvements</u>

What steps can be taken to reduce the errors you discussed above? Identify at least one hardware and one software solution. Provide explanations as to why they would work.

a. Hardware

The use of a light sensor to correct the robot’s location according to the odometer, so that the robot navigates to to the specific point based on the difference between its corrected current location and the waypoint location. This would work because a light sensor used on a grid of lines can provide a source of correction to the odometer.

b. Software

The use of a light sensor and an odometer correction class would allow for the robot’s navigation to be more precise. This can be done similarly to in lab 2. If the robot is moving in Y exclusively, the robot’s Y position can be updated every time that the robot crosses a black line. This would work similarly in the X direction. If the robot is moving in a direction 45 degrees off of a cardinal heading, X and Y can be both increased or decreased based on which of the 4 quadrants the robot is heading towards. The robot could then use these corrected odometer values to get a proper distance between itself and the waypoint, as well as a proper heading. This would work because, even if the robot is not completely on one of the middle waypoints, its initial position could be made more accurate, and therefore the distance travelled between waypoints, as well as the heading would be more accurate, causing the final location of the robot to be also more accurate.