IEOR 140: Project 3 Milestone 3

Team 6: MoonSoo Choi & Sherman Siu

Team Members:

**Sherman**, **MoonSoo:** Task Analysis, Class Design, Coding Implementation

**Time spent:** Approximately 20 hours … this milestone was not quite as easy as I expected ☹

**Project Description:**

Before milestone 3, we successfully programmed the robot to seek lights, approach the lights, and detect the obstacles if necessary. In milestone 2, once the robot detects an obstacle, it backs off 10 cm away from the object. In milestone 3, before the Grand Race, our objective was to program the robot to detect and avoid all the obstacles that are blocking the robot on its light-seeking path without any assistance.

**Most interesting/challenging/difficult part of the project:**

This milestone was extremely difficult for us, as it probably was for everybody else in the class. The idea of avoiding obstacles was extremely simple when we first tried our idea with a relatively small obstacle. However, the actual race course that Professor Glassey showed to us on Tuesday was absolutely more sophisticated than any of our classmates has expected. While our robot was on its way to seek its light out on the field, the robot ran into various obstacle situations, and we had to come up with a subtask sequence that could be used in any obstacle situation. The most challenging situation is when robot encounters a relatively long obstacle. Based on our programming code, the *Detector* class notifies the *Racer* class whether the obstacle is located on the left or right. However, when the robot runs into a long rod-shaped obstacle, the *Racer* is going to *listen* to both “is obstacle left?” event and “is obstacle right?” event, confusing the robot which direction exactly to turn. As a result, the robot sometimes steers to the left direction, sometimes to the right direction, confusing the robot and even resulting in the robot running in an endless loop. I will specifically discuss how we approached this problem later in the report.

Coding if statement conditions for the obstacle detection and the light seeking was also another challenging responsibility that we encountered. Based on our programming code, the *Detector* class notifies the Racer if there is an obstacle on the way of light-seeking, whether the obstacle was located on the right or on the left. We needed to determine whether the obstacle detection angle was positive and negative, and whether the robot needed to turn positive angle or negative angle, based on the information that *Racer* class “listened” to. Additionally, we struggled a bit through how we could keep track of *total* angle that the robot turns. Measuring ultrasonic sensor’s *getDistance()* experiment from milestone 2 greatly helped us in determining those parameter values. Nonetheless, being able to see our robot completing 2 round trips of the course brought us a sense of accomplishment!

**Hardware Design**

Throughout working on this milestone, our teammates realized that we could reduce the *touch sensor* functioning time by extending the touch sensor beam out to the front. Moreover, by constructing a stronger and bigger hardware piece for robot’s touch sensor, obstacle crash caused the sensor button to be pushed at a much faster rate, so that robot makes sure that it is alarmed when it crashed into the obstacle.

**Questions regarding the software design of the project**

1. Explain the responsibilities of each class you design for this lab, and how the classes collaborate, and how this relates to you task analysis.

In this project 3, there are 5 classes: *Detector*, *Scanner, Avoider, Racer, and Milestone.* *Milestone* class merely contains a main method, in which we specify how many laps we want the robot to travel. *Racer* class is a *listener* – it listens to any event that happens throughout the travelling, such as distance and angle away from the light (measured by *Scanner* class), distance and angle away from obstacles (measured by *Detector* class), and what appropriate actions the robot needs to execute through *Avoider* class. While the robot scans for light, it also scans for the obstacle at the same time. As the *Racer* class listens to various events throughout the travelling process, it also serves as a “communicator” between the events. For example, *Racer* class listens to the *Detector* class about various Boolean variables, such as whether an obstacle is detected, and whether it is located on the left or right. Then, *Racer* passes on those variables to *Avoider*, for this class to take an appropriate action in order to successfully avoid the obstacle. Overall in our task analysis, *Racer* is an overall body, and *Scanner, Detector,* and *Avoider* can be referred to as various body parts (hands, ears, eyes) of the *Racer*.

        What is the sequence sub-tasks that the robot should execute after an obstacle is detected?

Sequence of sub-tasks:

1. Detect obstacle (either by ultrasonic sensor or touch sensor)
2. Stop and back off about 10cm
3. Determine whether the obstacle is located on left or right side.
   1. If located on the left side, rotate in right direction.
   2. If located on the right side, rotate in left direction.
4. Detect if there are any nearby obstacles in that direction.
5. Travel forward for certain distance, until it detects yet another obstacle.
   1. If no additional obstacle is detected, rotate back to the original position and continue travelling.
   2. If an additional obstacle Is detected, rotate back to the original position first (in order to initialize the total angle rotated), and repeat the procedure from 1.

        What should happen after successful completion of this sequence?

After successful completion of this sequence, the robot shall either scan again to obtain the best light angle (also known as *getTargetBearing()*), or repeat the same procedure if it encounters yet another obstacle.

        Where in the sequence could another obstacle is detected?

As mentioned above in the description, another obstacle may be immediately detected after the robot backs off and rotates in an appropriate direction (case 1), or an obstacle may be detected while travelling along the rotated direction in order to avoid the robot (case 2).

Or even sometimes, if the obstacle located in front of the robot is long enough, the robot would still detect the same obstacle, after it travels along the rotated direction and rotates back to its original position (case 3).

        If it happens what should the robot do then?

In case 1, when the robot detects another obstacle immediately upon rotation, *isDetected* Boolean variable will be true, and it will return to its original position, and it will rotate to an opposite direction from initial one, and travel along that direction.

In case 2, if the robot detects another obstacle before travelling (in rotated direction) is finished, it will stop, and rotate to the original position (by rotating –totalAngle degrees). If there are no obstacle in the front, the robot will continue scanning and approaching the light.

Following case 2, if there still *is* an obstacle in the front even after travelling a certain distance in its rotated direction and rotates back to its original position, (this happens often when an obstacle is long) then the robot would rotate again to an appropriate direction, and repeat the same procedures when the robot detects an obstacle.

        Under what circumstances should the Detector object be monitoring its sensors?  All the time?

Ideally, all the time, with the light bulb being the *only exception*. When the robot detects an object (whether it’s either a light or an obstacle) we programmed the robot to do this simply by *conditioning* before calling the *avoider* method, by saying that the robot only needs to call the avoider method if the object detected is actually an obstacle (specifically, the condition will be: *if the distance away from the light is more than one and a half feet,* then call the avoid the obstacle). In other cases, the robot is obviously not as intelligent as human beings are, so in order to avoid frequently crashing into objects, the *Detector* object should always be monitoring (which the robot does by extending *Detector Thread*)

        What is the flow of information and control among the Seeker, Detector, and Avoider objects?

Seeker provides the *Racer* class the distance and the angle away from the light. While the robot is approaching the light, the *Detector* object will be run simultaneously as the *Seeker*. Once the *Detector* object recognizes an obstacle and informs the *Racer*, the robot will abort the light-scanning action for a moment, and focus on avoiding the obstacle (via *Avoider* class). Once an obstacle is successfully avoided, the robot can start running the *Scanner* and the *Detector* again through the *Racer* (here, the *Racer* class is regarded as the main communicator of this project, as it listens to other classes and carry the information onto other classes).

**Modifications on the software design?**

For the avoider, whenever the *Detector* object detects obstacles, the pilot would rotate 45 degrees either to the left or to the right. However, I eventually realized that there are quite a few long obstacles, so robot needed to repeat the *detection* and the *avoider* procedure several times before it could successfully avoid obstacles, and that took our robot quite a bit of time before it could successfully avoid the object and move onto the light. As a result, we decided to rotate the robot 90 degrees (in appropriate directions based on the relative location of the obstacle to the robot) when it ran into an obstacle. It brought us two big advantages. First, it was easier for us and for the robot to keep directions, since robot only rotates either to the right or to the left. Second, as mentioned above, the robot did not have to rotate multiple times before it finds an obstacle-free pathway, particularly when the robot runs into a long obstacle.

We also made modifications to the touch sensor. Originally, if the touch sensor *touched* any objects, it would simply back off and rely on ultrasonic sensor to determine whether an obstacle was on the right or on the left. However, we observed several situations in which the robot would *barely passes by and touches the obstacle located on the side* and it would back off after the touch sensor is pressed. Nonetheless, because the ultrasonic detector originally thinks that the obstacle was out of the angle range for the robot, it would not perceive such obstacle as an important obstacle. As a result, we split up the touch sensor into two situations: one when the left sensor is pressed and other when the right sensor is pressed. This way, the robot could turn immediately to an appropriate direction after its backup.

Finally, in the avoiding process, while originally we programmed the robot merely to rotate and travel in appropriate direction whenever it detects an obstacle, we programmed the robot to check whether there is an obstacle on its way while travelling in its avoiding process. This was absolutely a crucial key to the robot to avoid the obstacle, and helped us strengthen the overall control scheme.

Detector

(Running simultaneously with the scanner)

* If ultrasonic sensor or touch sensor detects the obstacle, let the *Racer* know, and give relevant information about the distance and the angle of the obstacle.

**Top Level Design**

Racer

* Steer to the light, also while detecting for obstacles.
* If an obstacle is detected, call avoider method.

Scanner

* Obtain distance and the angle of the light

Avoider

* Receive information about obstacle detection from the *Racer* class, and take an appropriate action.

Milestone 3

* Turn 180 degrees in front of the light

Calibration

*Scanner* informs the *Racer* the best light angle and the distance away from the light.

**(BEGINNING POINT)**

Turn 180 degrees in front of the light (**END POINT)**

If the light is within 1.5 feet distance

*Racer* steers towards the light, while *Detector* object is scanning for obstacles.

If no obstacle detected

If obstacle detected

If no obstacle detected

If obstacle detected

Stop

*Racer* listens to information collected by *Detector* object:

* Is obstacle on the left or the right?
* How far is the obstacle away from the robot?
* Angle of the obstacle respect to the robot?
* Is touch sensor pressed?

Rotate back to the original position

If obstacle detected while travelling

Travel in order to avoid the obstacle

*Rotate* right if obstacle on the left

Then, the *Racer* class carries this set of information to *Avoider* class, so the *Avoider* class can take an appropriate action in order to avoid the obstacle.

*Rotate* left if obstacle on the right

*Back off* 10cm when robot crashes

**Programs:**

Racer.java

Milestone3.java

Scanner.java

Detector.java

Avoider.java

JavaDocs located in the doc folder.