IEOR 140: Project 3 Milestone 2

Team 6: MoonSoo Choi & Sherman Siu

**Sherman**: coding, task analysis, experiments

**MoonSoo**: coding, task analysis, data, reports

**Time spent:** 6 hours

**Project Description:**

Last time, we programmed the robot to scan any lights nearby by continuously recording the distance and the angle, and approach the nearest light. In addition to such function, this time we programmed the robot to detect obstacles while it is travelling toward the light, and back off 10 centimeters away from the object if obstacles are on the way.

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

Clearly, the robot now has to take charge of more complicated set of responsibilities; scanning light, approaching light, detecting via ultrasonic sensor and touch sensor, and avoiding the obstacle by steering with appropriate angle. Since our coding style generally tends to be a bit too complicated, when we have a sophisticated structure of task analysis and software design scheme, it is highly important for us to keep the overall flow of coding as simple as possible. Meanwhile, we also realized that we also have to recognize the significance of having strong controlling ability overall – we plan on locating robot into numerous situations that robot may encounter in the “battlefield”, and strengthen the robot’s control scheme as it faces challenging obstacles that may cause the robot to feel “lost”.

**Task Analysis**

|  |  |  |
| --- | --- | --- |
| **Go to the light\* (milestone 1)** | Detect Obstacles (milestone 2)  Note: Detecting obstacles while travelling | Avoid (milestone 2)  Note: Aborting “obstacle detection” action once the robot spots an obstacle. |
| 1st Class: *Scanner* Class  (*italicized* name may NOT be the actual name of the class)   * Takes light intensity and angle (measured by tacho count) as parameters * While the robot is travelling, it updates the sensor’s light intensity value, and value of the angle at maximum light intensity. * Several get methods, such as obtaining the value of the light and the angle. * A method that continuously scans to look for appropriate | 1st Class: *Detector* Class   * *Detector* extends *Thread*   Therefore, the method that detects objects is named *run* method, in order to be called within the Racer class.   * If this class detects an object, then it will *“turn on the alarm”* by changing *isDetected* to true, which is a Boolean DetectorListener variable within the class. | 1st Class: *Avoider* Class   * In order to avoid obstacles after one is detected: * Back off about 10 cm away from the object, then steers away with an angle in opposite direction from the angle respect to the obstacle. * In the future, in order to strengthen our control scheme, we also plan on programming the robot to steer away from the obstacle ***without*** having to back away from the obstacle, so that it may save time. |
| 3rd Class: *Milestone1* class   * Merely meant to give “instructions” to the robot. * All the actions are actually implemented by Milestone1 main method.   Above tasks done by **Milestone1** class in the milestone  3rd Class: *Racer* class   * *Racer* class starts the Detector Thread within *goToLight()* method (Thread starts only once)   *Detector* Thread is supposed to run throughout the whole time while the robot is approaching the light at the same time. Then, it gives signals (setting *isDetected()* true or false) based on whether or not it detects any obstacle.   * Once the robot recognizes any obstacles, *Scanner* will set its *isDetected* Boolean variable to be true. Then, the *Racer* class will recognize this piece of information, and will have to call the *Avoider* class in order to tell the Pilot to stop and back off 10 cm away from the object. Once the *Avoider* class is done, the *isDetected* Boolean variable will be changed back to **false**. * Once the method within the *Avoider* class is complete, then *Milestone2* class returns to the beginning of its while loop, in which the robot will repeat searching the light (and *Detector* is still running throughout the whole entire time) | | |

**Overall Control Scheme**

Last time for milestone 1, we programmed the scanner to rotate degrees from the best angle (which is obtained from *getTargetBearing* method from *Scanner* class). For milestone 2, we decided to increase *Scanner*’s scanning angle to degrees for two reasons. First, the pilot will be able to detect obstacles in wider ranges: in the real obstacle race, it is extremely significant to detect **mobile** obstacles ***ahead of time*** so that our robot can take an appropriate avoiding action away from the obstacle. Second, our team occasionally encountered several situations where the robot strayed off from the track “in search of its light”, but once it strayed off too much due to distraction from excessive amount of obstacles, it eventually turns the opposite direction and starts heading towards the other light.

For milestone 2, our team did not undergo too much difficulty or challenges, as we merely needed to implement “detect” and “avoid” function within our robot. For the future, however, it is significant for our team to face with variety of difficult situations in the track, provide solutions for each problem, and assess the effectiveness of each solution. An important step for us for next part of the project, is to program the robot to actually steer away from obstacles when the *DetectorListener* “alarm” is on, rather than running into it every time and backing off 10 cm away from it (so our “backup” avoiding method should be really our backup option, or Plan B, rather than primary option).

Furthermore, in terms of the speed of the robot, we’re continuously looking for a balance between faster travelling speed of a robot and robot’s ability to detect objects quickly, and this responsibility is to be done before the grand race on next Tuesday!

**Experimental work:**

We ran an experiment for this milestone to determine appropriate parameters for the obstacle detection scheme. For control variables, we used physical distance away from the obstacle and various shapes of the objects as our experiment parameters. Our objective in this experiment is to understand the overall function of an ultrasonic sensor, its *getDistance()* method, and limits of its ability.

**\*Results attached on Appendix**

**Programs:**

Racer.java

Milestone2.java

Detector.java

Scanner.java

Avoider.java

**Appendix**

3

1

4

2

**Experimental Data results**

**Results Analysis**

One obvious result that one can observe is that as the robot approaches closer to the robot, the distance obtained by the ultrasonic sensor becomes smaller.

From this experiment, we realized that it is important to assign an appropriate *limit* distance to the robot, regarding when the robot needs to be alarmed via DetectorListener. If this distance is too close, the robot won’t be able to avoid the obstacle until it intimately touches it. If this distance is too far, then the robot would stop way too often on the pathway.

We observed some fluctuations within the data, and

this is due to several external factors, such as extraneous

obstacles laying on the field while conducting the experiment,

various shapes of each object, etc. We also observed that the

ultrasonic sensor reaches distance of 255 quite often.

Furthermore, we also observed that as the object is

wider and bigger, variability in ultrasonic sensor distance

data will be lesser. For example, graph 1 has a lesser

variability than graph 3, because the surface area of a

wooden block is much wider than that of a cone, so it will

be easier for the ultrasonic sensor to detect objects.