TEAM - 8

Software Strategy

Introduction

To finish software part of the project successfully it is important to recognize what functionality software should carry to be simple to implement, and at the same time allow the robot to score as many points in the game as possible. The main goal from the beginning though the end of the software programming was to make sure that each behavior is predictable and testable on its own. However, once two behaviors are completed it was essential to test them together and then proceed with coding another behavior.

Software Design

The software design has been decided by the software team. We have looked at different possibilities for our design and found that the subsumption architecture introduced by Brooks is a more preferable way to code. In part, this decision was made due to the confidence that it is possible to break the software design in a several independent modules, or behaviors that could be implemented by two people. After rigorous discussions we broke the design in four behaviors as follows (from highest to lowest priority):

1. **Escape**
2. **Avoid**
3. **Follow Light**
4. **Wander**

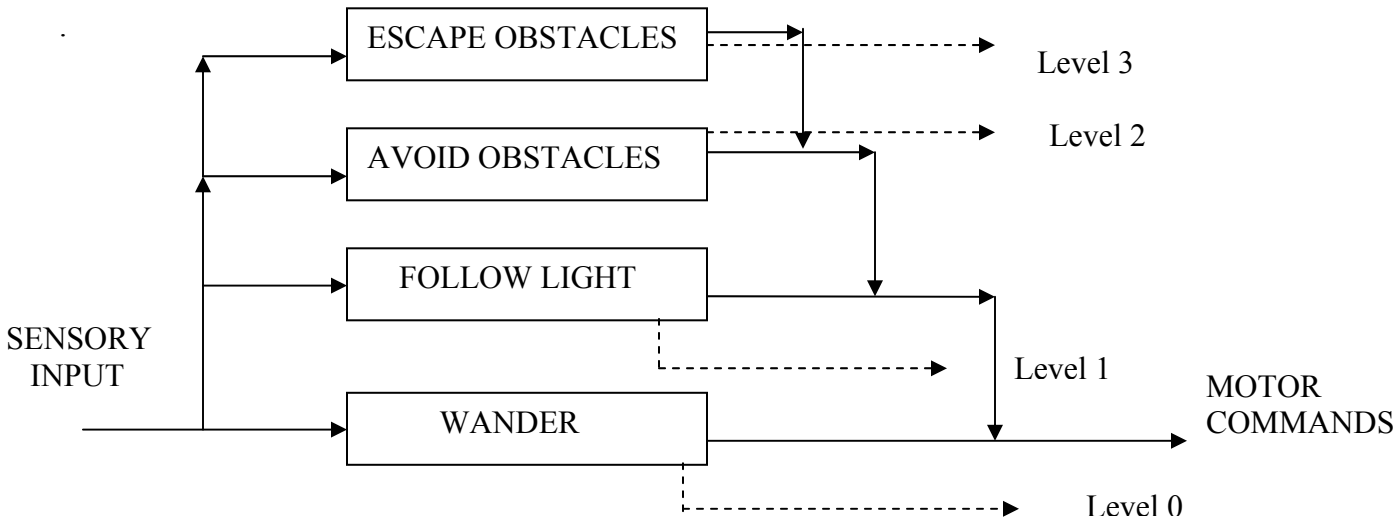

Figure - 1

In this design the Escape behavior would always subsume all behaviors. The **Avoid** module, however, can only subsume **Follow light** and **Wander**. The **Follow Light** can only subsume the **Wander** behavior. This is depicted in figure-1.

Escape behavior

This behavior gets the readings of the E.T sensors (range finders) as input and basing on them it approximates the distance and takes an evasive action. We took 55 as our threshold value using calibration. So, if the reading is greater than this value some obstacle is near by. For an example, if only the left sensor detects the obstacle it takes right turn first quickly until the two sensors do not detect anything in their way. If only right sensor detects the obstacle, it takes quick left turn until the two sensors do not find anything in their way. If both sensors detect an obstacle then the robot goes back for some time and takes either left or right turn quickly. This random nature is necessary as taking always turn to one side may put the robot in infinite loop in some cases.

The approach that we took for getting the E.T sensors readings has given us some benefit. Our sensors used to detect the light from the bulb when it is approaching the light and think that some obstacle is there. So, instead of taking instantaneous readings we took an average of 10 readings as one reading *(get_average_reading(port))*.

Avoid behavior

This behavior avoids rocks, walls and light stands in the arena using the readings given out by the bumpers arranged in the lower front portion of the robot (Refer hardware documentation). If the left bumper is hit, then robot thinks that some obstacle is there on the left side. So, it goes back slightly and then takes a right turn and similarly for the obstacles on the right side. If the robot hits the obstacles in the center, i.e., when at least one of the middle bumpers were pressed, then also, just like in the escape behavior robot goes back and take a left or right turn randomly.

Follow Light behavior

The **Follow Light** behavior in our design has been engineered to track the light source and guide the robot to the light bulb. The extensive testing provided useful to have four light sensors for the input to achieve robust behavior for detecting the light from the sides. The inputs to this behavior were the readings from four light sensors, two on the front and two on the back. The primary algorithm for this behavior was taken from the homework # 2. The algorithm was adjusted so that if the sum of the readings of the front sensors is lower than (or equal to) the reading of the two back light sensors then the robot should rely on the front sensors, otherwise the robot should rely on the readings of the back sensors. Once this decision has been made the basic algorithm from the homework # 2 was used to detect the strongest sensor in a chosen pair of sensors and decide whether to go forward, backward, or to turn left or right. To find the most suitable value for the threshold, “DEAD_ZONE”, a dynamic formula was derived. The testing

showed that a threshold of 17 % of the sum for front pair, and a threshold of 17 % of the sum of back pair is ideal to avoid condition where the **Follow Light** behavior is so sensitive to the light that it issues numerous commands to correct the path of the robot in a very small period of time. This makes the robot not to go anywhere. So it was very desirable to avoid this condition as much as possible.

Wander behavior

The wonder behavior is simply a default behavior, to go forward if robot cannot find light. Because the Follow Light module was so well done that it was able to sense the light almost from any angle and from far away distance this behavior was rarely dominating the movement of the robot.

Summary

With the approach described above we succeeded in touching maximum of two light bulbs in our demonstrations.