Programming Exercise: Multiple Behaviors.

As we have discussed, behaviors for behavior-based autonomous robots are usually built up incrementally. This assignment will generate robot control code with multiple behaviors. The resulting robot will have 4 behaviors: avoid_obstacles, wander, detect_beacon, and go_to_beacon. The robot control will be based upon motor schemas, with outputs of individual behaviors being vectors that are cooperatively summed to generate the resultant output control vector for the robot. The behaviors that are active at each point in time are determined by a sequencer that invokes the proper behaviors during the task. For example, the go_to_beacon behavior will only be activated after the detect_beacon has actually found a beacon.

This assignment guides you through a step-by-step development of this robot control code. You don’t necessarily have to develop the code in exactly the specified order. If you prefer to develop it in some other order, you can. The final robot behavior should cause the robot to wander around avoiding obstacles (similar to sobounce.c, except that you will break apart the “wander” behavior from the “avoid obstacles” behavior) until it detects a beacon (which is like a “goal” in your Assignment #2), then going to the location of the beacon using the go-to-beacon behavior. Once the beacon is reached, the Sequencer should cause the robot control code to exit. The following diagram gives the overall architecture of this robot control code.

The results you are to turn in for this assignment are listed at the end of this document.
1. **Simulation of beacon.** Often, robot simulators do not provide built-in capabilities to test all functionalities of a physical robot. In these cases, we have to build in new functions to allow us to simulate other robot capabilities. This can be done by changing the simulator itself (which can be rather involved), by mimicking additional capabilities using existing simulator functions, or through simple function calls. In this exercise, we want to simulate a robot sensor that can detect the position of a beacon whenever the robot is within a specified range of the beacon. Here, we will add functionality to the robot control code to simulate this capability.

To maintain some realism, in this exercise we will assume that obstacles in the environment are “short” (e.g., short walls), and that a beacon can be detected over the obstacles (e.g., a beacon on a pole). This simplifies the sensor simulation by eliminating the need to compute visibility from the robot to the beacon due to intervening obstacles.

a. Write a function called “initialize_beacon” that accepts the (x,y) coordinates of a beacon and, in order to aid the human user’s visualization of the beacon, draws a small circle in the simulated environment to visually mark the location of the beacon. Use the Nomad200 function “draw_arc(x,y,100,100,0,3600,1)” to mark the beacon position in the robot window. This initialize_beacon function will be included as an initialization function in your robot control code (i.e., no need to make changes to the Nomad200 simulator itself).

b. Write a function called “detect_beacon(range)” that provides the (x,y) position of the beacon if it is within a distance of “range” from the robot’s current position. If the beacon is not within range, then the function returns 0. For this exercise, set the detection range to 2000 units.

These functions will be used later.

2. **Avoid_obstacles.** Write an “avoid_obstacles” robot behavior. The perceptual schema for this behavior is the detection of a nearby obstacle in the direction of motion of the robot. The motor schema for this behavior is the generation of an output vector that turns the robot away from the obstacle. You can use capabilities similar to those provided within the “sobounce” code for detecting and avoiding obstacles, except that instead of generating “vm” commands, your code now generates an output vector giving the desired direction and magnitude of motion.

3. **Wander.** Write a “wander” robot behavior. The perceptual schema for this behavior is a potential field that occasionally changes the robot motion by some random direction and velocity amount, within pre-specified turning and velocity ranges. The motor schema for this behavior is the generation of an output vector that turns the robot in the specified random way.

4. **Combination of output vectors.** Write a function called “vector_combine” that accepts output vectors from multiple behaviors and generates a summed output vector by multiplying the individual behavior vectors by a given gain matrix, and then using vector addition.

5. **Conversion of output vector to motion commands.** Write a function called “vector_conversion” that accepts a resultant output vector from the behavior combination function and converts it to a “vm” command to control the robot’s motion.

6. **Test wander + avoid_obstacles + vector combination + output vector conversion.** Test your code from parts 2 through 5 to ensure that the robot successfully wanders about its environment avoiding obstacles. Generate your own test environments to debug your code.

7. **Go_to_beacon.** Re-use parts of your Assignment #2 code for “go_to_goal” to generate a behavior for go_to_beacon. Here, the beacon position returned by the “detect_beacon” behavior is the robot’s perceptual schema (same as the goal from Assignment #2), and the go_to_beacon motor schema generates an output vector that turns the robot toward the goal.
8. **Add Sequencer module.** To your debugged code from part 6, add a Sequencer module that activates the behaviors as follows:

- Initially, activate “wander”, “avoid_obstacles” and “detect_beacon”.
- When “detect_beacon” has found the beacon, deactivate “detect_beacon”, deactivate “wander”, and activate “go_to_beacon”.
- When the beacon position is reached, turn off all behaviors and exit.

9. **Test complete system.** Test your complete system using the environment described in file map4.txt, downloadable from the course web site. Experiment with various beacon locations and their consequences for your robot control code. In particular, test out beacon locations with the following characteristics:

   I. Beacon within range of the starting robot position.
   II. Beacon out of range of starting robot position, but with no obstacles between the beacon and the robot when the beacon is first detected.
   III. Beacon out of range of starting robot position, and with “short, simple” obstacles between the beacon and the robot when the beacon is first detected.
   IV. Beacon out of range of starting robot position, and with “long wall” obstacles between the beacon and the robot when the beacon is first detected.

**NOTE:** You DO NOT have to add additional functionality to your code to address beacons of type IV. However, your code should be able to handle beacons of types I through III.

**TURN IN THE FOLLOWING:**

a) A description of the functions you used to generate output vectors for avoid obstacles, wander, and go_to_beacon, including a definition of the magnitude and direction of each output vector. Also discuss any issues and challenges you had to overcome in defining these output vector functions.

b) A description of the gain matrix you used to weight the output vectors from the various behaviors. Also discuss any issues and challenges you had to overcome in defining the gain matrix.

c) A description of the function you used to convert the resultant output vector to a robot “vm” command. Also discuss any issues and challenges you had to overcome in converting this output vector to a suitable vm command.

d) Four separate screen dumps of your code running using the environment in file map4.txt, showing a trace of the robot motions from its starting point to the point where the robot has reached the four beacons described in part 9 above. Make sure you clearly label each screen dump according to which category of beacon the screen dump depicts.

e) A description of any other issues and challenges you experienced and had to resolve in developing this robot control code.

f) A hard copy of your complete code, fully documented.

g) Email an electronic version of your code to parker@cs.utk.edu, naming your code yourlastname-3.c (or .zip, etc.)