Assignment 01 - Behavior Trees

Peter Nadel

June 6, 2025

1 Instructions

To run the cleaning script, follow these steps:

- Make sure your working directory is the "bt_py_2.0.1" directory with the `pwd` command.
- Once you are in that directory, simply run the main script with `python main.py`.
- You will be asked whether you would like the robot to engage in spot cleaning or general cleaning. A description of how I interpreted and implemented these actions is below.
- The script will end after 10 general cleanings or 10 spot cleanings. The script will also end if the robot hits 0% battery

2 Design, Implementation and User interface

Along with the code and the general description, we were supplied with a tree diagram which strongly shaped my design of the cleaning robot. First, I began by determining what was supplied to me in the `bt` and `bt_library` libraries and what I needed to fill in. Included in these sources was:

- The selector composite;
- The battery-less-than-30 condition;
- The timer decorator;
- The blackboard and some blackboard variables, including battery level and home path.

I quickly realized, however, that these too would need to modified and updated, especially once I got to my extra experiment.

This would leave several composites, tasks, conditions, and decorators for me to implement. I started by understanding the left-most branch of the given tree: how the robot charges. As the battery is initialized to 29, this step would be the first to happen, no matter if the robot would conduct spot cleaning or general cleaning. The first step was to implement the sequence composite, which should only return `SUCCEEDED` if all of the branches return `SUCCEEDED` and otherwise should return `FAILED`. My sequence composite takes in a list of children nodes (tasks, conditions, etc...) and loops through them. If any of them return `FAILED` then the whole sequence return `FAILED`. If, instead, it gets to the end of the loop, it will return `SUCCEEDED`.

With this composite, I was able to move to the tasks. The battery-less-than-30 condition and the find home task were already implemented for me, so I turned to the go home and doc tasks. These involved me recalling the home path from the blackboard and then charging the robot in its dock. All charging and cleaning tasks I simulated with a loop for a certain amount of iterations, using the `time.sleep` function to mock actual charging or cleaning.

Once this left branch was complete, I needed to work on the next composite, the priority. This was implemented similarly to how the selection was implemented, except I added a new initialization parameter, priorities, which could be any iterable whose values corresponded to the list of child nodes in the first parameter. In the `run` method, I zip together the children and the priorities and iterate through them. The priority returns `SUCCEEDED` if any child returns `SUCCEEDED` and `FAILED` otherwise.

With the priority composite implemented, I followed a similar structure to implementing the other branches as I did for the docking and charging branch above. The spot cleaning condition simply check the state of the blackboard to see if the user has indicated they want to initiate spot cleaning. If so, the robot begins spot cleaning for 20 seconds by way of the timer decorator. Each time the robot completes a run of the clean spot task, 5 battery points are deducted. Finally, the done spot task reports the conclusion of the task to the blackboard.

For the general cleaning branch of the tree, I again followed a similar pattern. The condition general clean checks the blackboard for the response of the user, while the dusty spot condition checks the blackboard for the random variable of dusty spot. Whether or not the dusty spot variable is `True` or not, if general is selected, the robot will always do a general cleaning. Conducting a genera cleaning costs the robot 45 battery points. Finally, once the general cleaning is complete, the robot reports this too the blackboard and switch the general cleaning keyword to `False`.

The last task is the do nothing task, which simply does nothing. It simulates the robot in an idle position and awaiting a command for either spot or general cleaning from the user. This user interface is conducted through the built-in `input` method in Python. It accepts only two options: ”spot” and ”general” and will alert the user if they input any unsupported keywords.

3 Extra experiment

For my additional experiment, rather than expanding the tree, I sought to add a spatial component to the assignment through the use of the `turtle` package in the Python Standard Library. Usually, `turtle` is used to trace complex figures and shapes. However, in this case, I used it to track the position of the robot, in a top-down, cartesian coordinate system.

To implement such a system. I first needed to introduce a new blackboard variable: `CURRENT_COORDINATES` which kept the state of the robot after any movement and is initialized to be at (0,0). For the sake of simplicity, this variable is updated once the robot has reached its destination. Too, I added a `HOME_COORDINATES` variable which would keep track of where the docking station is. This variable is never updated as the location of the docking station should never change.

I added the `turtle` behavior to several of the tasks. For all actions of the robot, I used simple vertical (y) and horizontal (x) movement. To that end, the robot cannot move diagonally. This choice was somewhat arbitrary, but I like the nearly retro look of it. First, because the robot is initialized at 29% battery, it begins by going home. I used `turtle` to draw its path from its starting coordinates (0,0) to the home coordinates, arbitrarily set to (-150, 150). In `turtle`, we can use the `setx` or `sety` methods to instruct the robot to move to a particular location. For this task, I then use a red colored line to trace the robot's path. Next, for the spot cleaning task, I generate a random x,y point using the `randint` method and have the robot move there using a blue line to differentiate it from other movements. The operation occurs in the case that the dusty spot sensor is switched the `True` in the general cleaning branch. The general cleaning task exploits similar functionality to the other `turtle` operations, but rather than going to a particular coordinate to clean, the robot goes to the origin and follows a spiral path to simulate a general cleaning of the room. The robot goes through this spiral pattern three times at an offset of 120 degrees so that it is able to explore new areas while also making sure that the areas it has already explored are completely clean. This operation traces its path in green for further differentiation.

A sample image of the path of the robot for a few actions in included below.

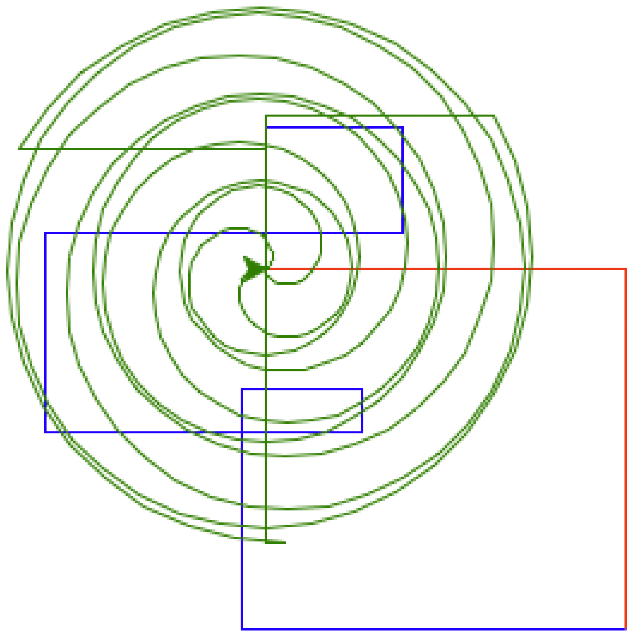

Figure 1: Sample path of the robot rendered with `turtle`