Intelligent Robotic Systems
-
Third Lab Activity

Simone Ceredi

22/03/2024

Solution implemented

Task definition

The solution implemented contains four different behaviours that are exposed in the levels interface. The behaviours are: *stay_still*, *go_towards_light*, *random_walk* and *collision_avoidance*. Before combining the final behavior from the four listed above, each one was individually tested.

- The *stay_still* behaviour stops the robot from wandering when it gets close to the light source, it was tested in order to find an appropriate light intensity threshold that allowed the robot to stop when close enough to the light source. When the level is active, all the LED's of the robot get coloured yellow.
- The *collision_avoidance* behavior enables the robot to steer away from obstacles. When close to an obstacle, the robot rotates on its axis until the obstacle is behind it, allowing the robot to proceed normally. When the level is active, the robot will turn all its LED's red.
- The *random_walk* behaviour makes the robot wander randomly, "exploring" the arena. When the level is active, the robot will turn all its LED's green.
- The *go_towards_light* behaviour moves the robot towards the light source by rotating it on its axis until it faces the light and then going straight. When the level is active, a LED will turn yellow indicating the approximate position where the robot senses the strongest light intensity.

Subsumption architecture implementation

The implementation of the subsumption architecture is created as follows:

```
M.list = {
    {
        callback = levels.random_walk,
    },
    {
        callback = levels.go_towards_light,
    },
    {
        callback = levels.collisions_avoidance,
    },
```

```
        {
            callback = levels.stay_still,
        },
    }
```

Making it very easy to extend the functionality of the implemented collective behaviour. Each level has to define if it should run or not. If it runs, it must return the speed values that need to be assigned to the wheels; otherwise it must return nil instead. At each step, every level will be called, and if executed, it will overwrite the speed values the previous level had returned.

Code structure

The modelling of the solution using the subsumption architecture allowed for a better structure in the codebase. All the thresholds and other variables that need to be fine-tuned can be found in the *src/globals.lua* file, while *src/levels.lua* and *src/levels-stack.lua* contain respectively, the level definition and the structure of the subsumption architecture.

Evaluation

After running the simulation 1000 times with different arenas, the results were the following:

Count	**Mean**	**Std**	**Min**	**25%**	**50%**	**75%**	**Max**
1000.0	0.686324	1.168522	0.00007	0.138454	0.167848	0.239291	5.462819

Table 1: Summary Statistics

And the box plot of the results:

The results look quite good as shown by both the 75 percentile and the box plot. Showing that the robot most of the times arrives at it's destination without issues. As seen in the box plot the times it does not arrive there are "outliers", but the problem the robot usually faces is getting stuck between obstacle with a funnel-like or "L"-like structure. As shown in the histogram in fig. 2 the times the robot does not arrive to the destination are uniformly distributed, this could mean that the reason it does not get to the destination is because it gets stuck in the vertex of the previously described obstacles.

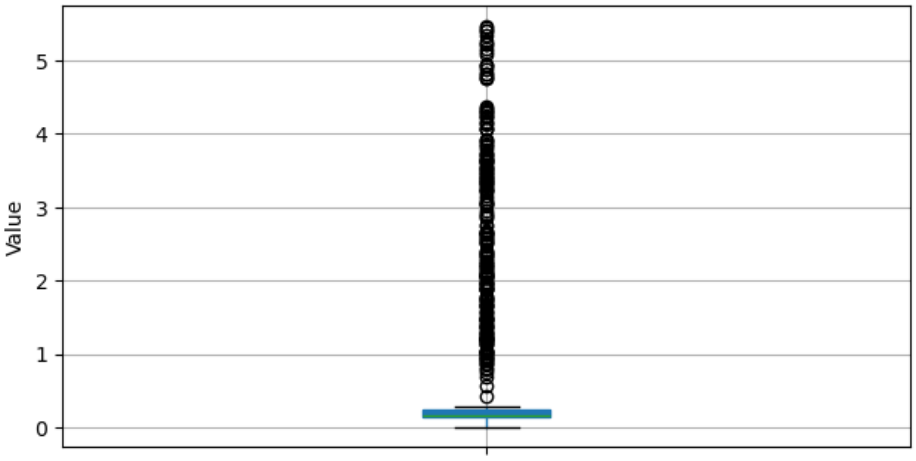

Figure 1: Box plot of the results

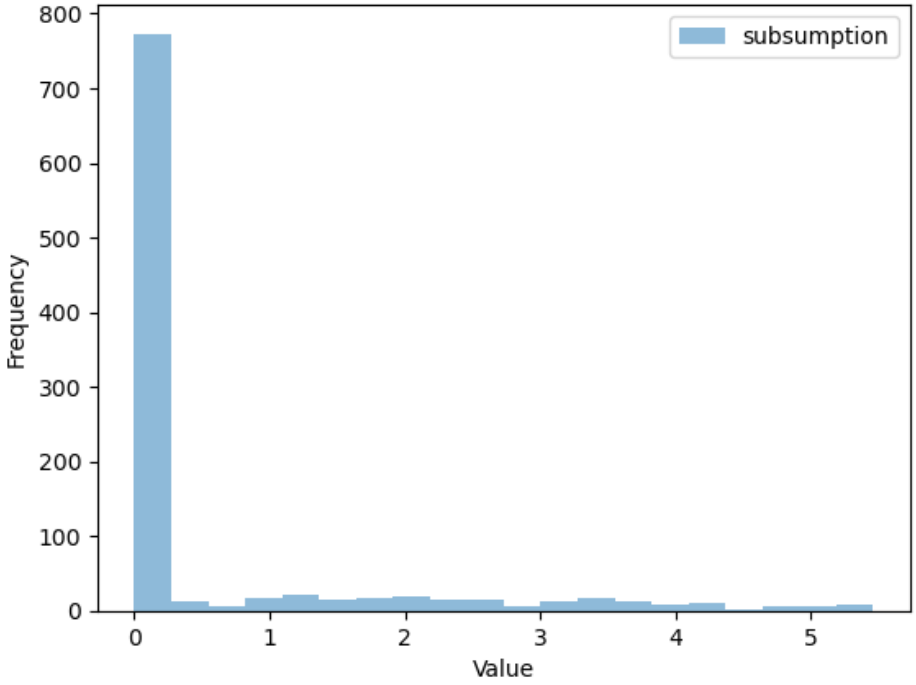

Figure 2: Histogram of the results

Problems faced

Some problems emerged during the execution of the composite behaviors that did not appear during the testing of the individual ones. Some problem were solved, while others remains unresolved.

Solved problems

A problem that occurred regarded the *obstacle_avoidance* behaviour in combination with the *go_towards_light* one. When avoiding an obstacle if the light was positioned behind the edge of the obstacle the robot was trying to avoid, the robot would often get stuck, as it was switching between the 2 behaviours described above. The problem was mitigated by introducing some randomness in the *go_towards_light* behavior by extracting the speed values assigned to both wheels from a uniform distribution.
Another problem encountered has to do with "L" shaped obstacles, when the robot approached the vertex of the "L", since the *collision_avoidance* level used to only consider the closest obstacle, the robot would go deeper and deeper in the obstacle by wiggling left and right. The problem was mitigated by adding a check that spins the robot to the left if the 2 closest obstacles are in front of the robot, and their distance to the robot is fairly close. Unfortunately this solution only mitigates the problem as the robot can get stuck in obstacle with a gap in the middle.

Unsolved problems

The biggest and still unsolved problem faced regards a specific configuration of the arena. When the robot encounters a funnel-like structure, if it gets trapped inside, it is unable to escape. The funnel structure is composed by 2 obstacles positioned in a "V" shape, creating the funnel mouth, while the funnel spout is the light source. In this particular case the robot will get stuck between the *obstacle_avoidance* and *go_towards_light* behaviours and never be able to exit or circumnavigate the funnel.