Composite behaviours
- *Intelligent Robotic Systems* -

Giulia Nardicchia
giulia.nardicchia@studio.unibo.it
Dept. of Computer Science and Engineering (DISI), Alma Mater Studiorum Università di Bologna

March 15, 2024

Composite behaviours

This Laboratory Activity 02 aims at combining simple behaviours into a composite one.

Task

The robot should:

- find a light source and go towards it
  - reach the target as fast as possible (`MAX_VELOCITY` = 15)
  - once reached it, it should stay close to it (either standing or moving)
- avoid collisions with other objects (such as walls, boxes and other robots).

The robot is equipped with both light and proximity sensors.

Solution description

- **utilities.lua** → contains enums and utility functions.
  - `MovementType`: enum representing the robot's movement type (`FORWARD`: moves forward, `BACKWARD`: moves backward, `LEFT`: turns left, `RIGHT`: turns right, `BACK_LEFT`: turns left backward, `BACK_RIGHT`: turns right backward, `STOP`: stops)[1]
  - `OrientationType`: orientation, considering the robot's sensors as arranged vertically and horizontally:
    - `VERTICAL_FRONT` table containing indices of sensors corresponding to the front half
    - `HORIZONTAL_LEFT` table containing indices of sensors corresponding to the left half
  - `move`: function representing the robot's movement based on the specified `MovementType`, setting wheel velocities using `wheels.set_velocity`.
  - `search_highest_value`: function that searches among input sensors for the highest value and returns that value with its corresponding index
  - `read_half_sensors`: calculates the sum of half the sensors based on the specified `OrientationType` (left/right horizontally or front/back vertically).
- **performance.lua** → contains the `euclidean_distance` function to calculate the Euclidean distance between the robot's current position and the light's position.

[1]These movement types have not all been used; I implemented them with future reuse in mind.

- **composite_behaviour.lua** → contains the robot controller logic. I developed a solution using a behavior-based architecture with classic arbitration (sense-think-act) and divided the robot's behavior into four main states, structured like a **deterministic finite state automaton**.
  - Initially, I defined global variables representing threshold values, which were heuristically adjusted based on environmental conditions:
    - `MOVE_STEPS = 15`
    - `MAX_VELOCITY = 15`
    - `LIGHT_THRESHOLD = 1.8`
    - `PROXIMITY_THRESHOLD = 0.1`
    - `NOISE_THRESHOLD = 0.06`
    - `FRONTAL_THRESHOLD = 0.05`
  - The `init` and `reset` functions are identical and establish a random initial movement, set `n_steps` to 0, and set the robot's initial state to `RANDOM_WALK`.
  - In the `destroy` function, there are `logs` of the robot's position, the number of steps, and the calculation of the Euclidean distance to verify how close the robot gets to the goal at the end of the simulation.
  - The `step` function, containing the actual controller logic, is executed every 10 ticks per second. It reads the light and proximity sensor values and performs some sensor pre-processing. It checks which state the robot is in.
  - if the total sum of all light sensors exceeds a certain threshold, the robot is in the state of:
    - `REACHED_TARGET`: it has reached the goal and should stop;
  - if the total sum of all proximity sensors exceeds a certain threshold, the robot is in the state of:
    - `OBSTACLE_AVOIDANCE`: if the sum of right proximity sensors exceeds that of left ones, it turns left, and vice versa, aiming to move away from the obstacle.
  - if the total sum of all light sensors detects nothing and no object is detected in front, the robot is in the state of:
    - `RANDOM_WALK`: it executes a random walk, changing periodically after every `MOVE_STEPS`;
  - if none of the previous conditions succeed, the robot remains in the state of:
    - `SEARCHING_TARGET`: if it detects light in front (greater than 0) and detects nothing behind (below the `FRONTAL_THRESHOLD`), it means it is positioned in front of the light and can move forward at maximum velocity. Otherwise, it performs further checks to recalibrate the robot's position; if the sum of right sensors exceeds that of left ones, it turns left, and vice versa, aiming to approach the light.

Problems and possible solutions for each environment change

For each .argos file, I set a `random_seed` to reproduce the experiment to see how the robot's behavior changes with varying thresholds, then modified this value to observe changes in behavior with varying environments.

- **01_composite_behaviour.argos** → cubic blocks were left in this file with a single robot.
  - Problem: Initially, I had not divided the front and back parts for light sensors, causing the robot to zig-zag even though it reached the goal effortlessly.
  - Modified Solution: I added an extra check to make the robot move straight if it is facing the light.
- **02_composite_behaviour_more_blocks.argos** → the number of blocks was increased in this file, with different dimensions (heights and lengths).
  - Problem: Tall blocks obstructed the light sensor's view, causing the robot to fail to detect light intensity.

  - Modified Solution: I added random movement (after a certain number of steps) if it does not detect light and there are no obstacles in front (which was unnecessary in the first solution, with a simpler and more favorable environment).

- **03_composite_behaviour_more_robots.argos** → the blocks remained unchanged in this file, but the number of robots increased (quantity = 3), to observe behavior with moving objects.
  - Problem: I had set them to keep rotating near the light, causing the robots to "push" each other since they detect each other as obstacles.
  - Modified Solution: When a robot finds the light, it stops.

- **04_composite_behaviour_more_lights.argos** → the number of lights was increased in this file, with one central light of intensity 1 and another shifted close to the edge, lower in height, and with lower intensity.
  - Problem 1: If the intensity is too low, the total detected light intensity never exceeds `LIGHT_THRESHOLD`, causing the robot to rotate in proximity, which is one of the possible behaviors in the solution but not intended according to the logic set.
  - Problem 2: In an unfortunate scenario where the blocks form a cone and the light is behind, the robot keeps rotating because it continually detects the light and the obstacles, sometimes unable to escape the trap.

- **05_composite_behaviour_more_lights_with_noise.argos**→ this file retained all settings from the previous environment with the addition of noise in sensors and actuators set to 0.01, and I changed the value of random_seed.
  - Problem: I had to adjust threshold values due to noise.

To visually inspect the behavior, LEDs are utilized with colors corresponding to the states of the automaton.

- In the searching target state, black LEDs are assigned.
- In the obstacle avoidance state, red LEDs are assigned.
- In the random walk state, green LEDs are assigned.
- In the reached target state, yellow LEDs are assigned.

Observations

The main problems encountered are:

- Threshold values set through empirical testing in one environment may no longer be effective if the conditions of the world change.
- In a specific scenario where blocks form a cone with the light behind them, the robot may become trapped in a loop, unable to escape.