Behaviour-based control in ARGoS: Motor schemas
- *Intelligent Robotic Systems* -

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

April 12, 2024

Motor schemas

The task required for the Laboratory Activity 04 is to implement the robot control program in ARGoS on the basis of the motor schemas architecture.

Perceptual schemas

I have identified two perceptual schemas. Each perceptual schema relies on a specific sensor: the first schema identifies the sensor detecting the highest light intensity, while the second schema does the same with the proximity sensor.

Behaviors

The task remains consistent with the previous laboratory. I defined the primary behaviors I wanted the robot to exhibit:

- **Phototaxis**: The robot should find a light source and move towards it. The foot-bot should reach the target as quickly as possible and, once there, should stay close to it (either standing or moving).
- **Obstacle Avoidance**: The robot should avoid collisions with objects such as walls, boxes, and other robots.
- **Tangential Movement**: This behavior involves guiding the robot around an obstacle encountered in its path, enabling it to navigate smoothly without colliding.
- **Uniform Movement**: The robot should move in a given direction.
- **Random Movement**: The robot should move randomly if it doesn't detect any light or obstacles in the proximity.

Implementation description

Potential Fields

In `potential_field.lua`, I established corresponding potential fields for each behavior:

- `attractive_field`:
  - If the *distance* to the light source is less than or equal to the threshold distance D, the function calculates a length for the attractive force vector. The *length* of the force vector is determined by the ratio of the difference between D and the actual *distance* to D. This ratio is used to scale the attractive force, with the force being stronger when the robot is closer to the light source.

  - The *angle* of the force vector is set to the input angle parameter, indicating the desired direction of movement.
  - This setup ensures that the robot is attracted to the light source and moves towards it.

- `repulsive_field`:
  - The *length* of the force vector is calculated similarly for both the attractive and repulsive forces.
  - The negative sign is applied to the *angle* parameter for the repulsive force to ensure the robot is pushed away from obstacles.

- `tangential_field`:
  - The *length* of the force vector is set to the distance parameter.
  - The *angle* of the force vector is adjusted by adding $\pi/2$ (90 degrees) to the input angle, enabling clockwise rotation. This adjustment ensures that the force vector is perpendicular to the obstacle, directing the robot tangentially around it.

- `uniform_field`:
  - Used for both uniform movement and random movement.
  - The *length* of the force vector is set to the distance parameter, ensuring that the strength of the force remains consistent regardless of distance.
  - The *angle* of the force vector is set to the input angle parameter, indicating the desired direction of movement.

Environment

The environment, `test-motor_schemas.argos`, is set up to test the laboratory. Light and proximity sensor rays are activated to visually inspect whether the robot perceives stimuli or not. Initially, I tested the controller with a single robot in the arena, then scaled up to 10 to ensure functionality in the presence of mobile objects.

Utilities

`utilities.lua` contains utility functions, and in this lab, I specifically utilized `search_highest_value`, which returns the index corresponding to the maximum value among all sensors.

`performance.lua` contains a function for computing the Euclidean distance between the robot's position and the target light source. I employed this information, based on known positions, to establish the distance threshold D.

Within `vector.lua`, there are several utility functions provided by the Professor. Primarily, I utilized the function `vector.vec2_polar_sum(v1,v2)` to add vectors (in polar coordinates) pairwise.

Controller

`controller-motor_schemas.lua` encapsulates the robot's controller logic.

Some constants used in the code are defined, such as the maximum number of movement steps (`MOVE_STEPS`), maximum velocity (`MAX_VELOCITY`), thresholds for light (`LIGHT_THRESHOLD`), proximity (`PROXIMITY_THRESHOLD`), and noise (`NOISE_THRESHOLD`), as well as the wheel axis length (`L`) and distance (D).

The `init` and `reset` functions are equivalent, where the global constant `L` is defined, whose value is obtained with `robot.wheels.axis_length`, and `n_steps` is set to 0.

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 performance at the end of the simulation.

In the `step` function:

- First, the values of the light and proximity sensors are read, and the corresponding indices for the maximum detected value for each sensor are determined.

- From the indices, I obtain the *value* and *angle* for both sensors, and by multiplying *value* by D, I obtain the *distance* from the light.
- Force vectors are calculated for attraction, repulsion, and tangential movement behaviors using the `attractive_field`, `repulsive_field`, and `tangential_field` functions.
- The resulting vectors are summed pairwise using polar coordinates.
- I've added two controls:
  - If the light is detected and there are no obstacles in proximity, then an additional vector for uniform movement is added to the previous result vector. I used the `uniform_field` function, passing constant length and an angle proportional to that of the detected light.
  - If the light is not detected, then an additional vector for random movement is added to the previous result vector. I used the `uniform_field` function again, passing constant length and an angle obtained with `robot.random.uniform(`$-\pi/5$`, `$\pi/5$`)`.
- Motor commands expressed as translational and angular velocities are converted into differential actions (wheel velocities) using the `converter_from_rototransational_to_differential` function.
- Afterwards, they are scaled to the maximum velocity using the `multiplier` function.
- Finally, the robot's wheel velocities are set based on the obtained results.

Performance Evaluation

I have run the simulation 1000 times with different arena instances, and the results obtained by calculating the Euclidean distance are as follows, as demonstrated by the statistics in the table and depicted through histograms and box plots:

Motor Schemas							
Count	Mean	Std	Min	25%	50%	75%	Max
1000.0	1.003964	0.995048	0.140637	0.489678	0.512185	0.892344	4.802051

Table 1: Summary Statistics for lab_activity-04

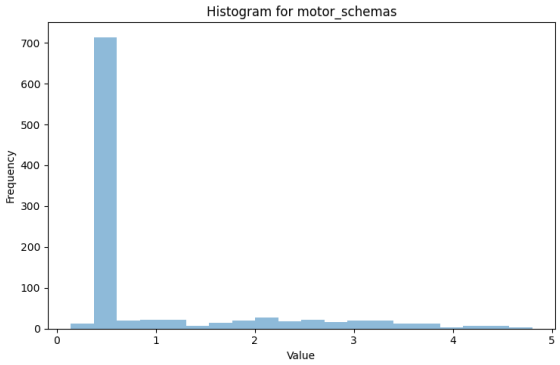

Figure 1: Histogram

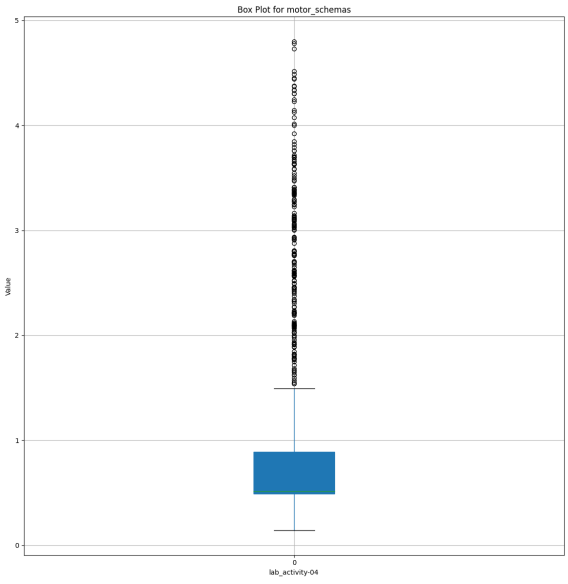

Figure 2: Box Plot

Observations

- Since I set the tangential movement only in a clockwise direction, the robot will always turn to the right, and it's possible that encountering an obstacle placed in a certain position may force the robot to turn back. However, I decided not to allow the possibility of turning left because it got stuck more often in cases where obstacles were positioned in front of the target in a cone shape.
- Personally, I haven't encountered the **Local Minima** problem, probably because there was noise in the light and proximity sensors, and perhaps because I also added a constant uniform and random field in some circumstances.
- The performance evaluation results are slightly different, but the motor schemas architecture appears to be superior to the subsunction one.