Behaviour-based control in ARGoS: Motor schemas

– Intelligent Robotic Systems –

Andrea Roli
andrea.roli@unibo.it
Dept. of Computer Science and Engineering (DISI)
Alma Mater Studiorum Università di Bologna

Motor schemas

Motor schemas are among the most used models for implementing behaviour-based control. In this lab session you will be asked to program a robot for a given task according to this architecture.

Task

The robot is expected to be able to find a light source and go towards it, while avoiding collisions with other objects, such as walls, boxes and other robots. The robot should reach the target as fast as possible and, once reached it, it should stay close to it (either standing or moving). For your convenience, an `argos` file defining the arena is provided. This defines the distribution of typical arenas for which you have to devise your controller. For physical constraints the wheel velocity cannot exceed the value 15 (i.e., 15^{-2} m/s). The robot (a *footbot*) is equipped with light and proximity sensors. In addition, the robot is also equipped with the positioning sensor, which however can only be used for testing and evaluation purposes.

Exercise

Implement the robot control program in ARGoS on the basis of the motor schemas architecture. Before starting to code, define the basic behaviours and the corresponding potential fields.

Some suggestions:

- For a given behaviour, decide first what kind of field is suitable. Remember that the goal is to define a force that will be exerted on the robot (i.e. don't care about the movement: this comes for free when you translate length and angle of the resulting vector into wheel velocities). Then focus on the *perceptual schema* (one PS for one sensor? One PS for a set of sensors?) and consequently define the vector (length and angle).
- In case, set the light and proximity sensor rays on, so as to be able to visually inspect whether the robot is perceiving a stimulus or not.
- Try the controller also with more than one robot in the arena (just place n robots randomly by using `distribute`). This will be useful to test the collision avoidance module in presence of moving objects.
- As the position of the light bulb is fixed and known in advance, you can compute the distance between the robot and the target by means of the positioning sensor. Therefore, you can evaluate the performance of the robot by estimating the time needed to reach the light, e.g. in terms of number of steps. Alternatively, you can either count how many times the robot can reach the light in a given time (just set the simulation duration in the ARGoS configuration file) or take the distance between the robot and the light at the end of the simulation. Question: what is a sound way for statistically estimating the performance of the controller?

For vector operations, you can use the Lua module `vector.lua`, which you can load from your main program (the controller) via this instruction:

```
local vector = require "vector"
```

Functions can be called with dot notation.

Remember that light and proximity sensor readings in ARGoS–Lua return two fields, namely *value* and *angle*. Sensors are numbered from 1 to 24, counterclockwise. Angles of sensors from 1 to 12 are from 0 to (almost) π, whilst they have negative values for sensors 13 to 24 (from $-\pi$ to 0).

In usual implementations of this architecture, you just need to create and operate on vectors in polar coordinates, which must contain two fields (`length` and `angle`) which are initialized and assigned as follows:

```
v = {length = 0.0, angle = 0.0}
....
v.length = 2.1
v.angle = 1.2
```

Then you sum the vectors (in polar coordinates) two at a time by using the function `vector.vec2_polar_sum(v1, v2)`.

To convert motor commands that are expressed in the form of translational and angular velocities into differential actions (wheel velocities) you may want to use the formulas in the following.

Variables:

- v: translational velocity
- ω: angular velocity (expressed coeherently with ARGoS convention: positive angles towards the left, negative otherwise)
- v_l, v_r: linear left and right wheel velocity
- L: distance between the two wheels

From (v, ω) to linear velocities v_l and v_r:

$$\begin{bmatrix} v_l \\ v_r \end{bmatrix} = \begin{bmatrix} 1 & -L/2 \\ 1 & L/2 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

The distance between the two wheels in the footbot can be retrieved by:

1. adding the *differential steering* among the sensors:

   ```
   <differential_steering implementation="default" noise_level="0.0" />
   ```

2. and using the following piece of code in your Lua controller:

   ```
   L = robot.wheels.axis_length
   ```

   A possibility is to set a global variable `L` in the `init()` function.

Food for thought

- What are advantages and disadvantages of the MS architecture?
- Have you observed situations in which the robot gets stuck or is trapped in a cycle?
- Try to elucidate the main differences between the subsumption architecture and motor schemas.
  - Which is the easiest to implement?
  - How can one assess the performance of the two implementations?
  - Which is the best from the viewpoint of extendibility and composition of behaviours?
  - Is there a way to design a hybrid architecture composed of both the subsumption architecture and motor schemas? Would it be a good idea to do it?[1]

[1] If your answer is "no", please provide a good reason for this.