Behaviour-based control in ARGoS: The Subsumption Architecture

Observations

– Intelligent Robotic Systems –

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

Common solutions proposed by the class:

1. Competencies implemented as functions that return either velocities or a "subsumption message" (e.g. inhibition flag for a level) or both.
   - Note: it is important that the functions set to `true` the "subsumption message" on the basis of the readings from the sensors, so that each function has its own mechanism for deciding whether to take control. This is in general the most appropriate implementation for the subsumption architecture (see example code provided).
   - These functions can be either nested or simply called sequentially with final selection of the signals to be sent to the wheels.
2. Functions directly send signals to the wheels but are called sequentially, from 0 to $n-$th level, so that higher levels can override the signals.
   - Simple yet effective solution in simulation, but to be tested on real robots. Safe solution: levels override two variables that keep the velocity values; use them to control the motors at the end of the sequential call of the functions.
3. In function `step()` a selector reads sensor values and select the behaviour to activate.
   - Neat and effective solution, but not completely in the spirit of Brook's idea. More similar to FSMs. Care needed to implement it so as to have a code that can be easily extended.

In the solutions presented, usually obstacle avoidance is the uppermost level (i.e. it is more important to avoid collisions, so this level takes control in these cases), but it also happens that phototaxis is the uppermost (in this case, the decision has to be taken also with respect to to proximity sensor values).

Observations

- The implementation on the robot model in ARGoS requires some adjustments, as the parallelism of the architecture can not be achieved. However, we can encapsulate all the computation related to a competence in one function, and set a priority among these levels; then we can either call them either from the highest priority function (and we use locks or similar mechanisms to suppress outputs of lower levels) or from the lowest one (implementing an overriding of the variables used to control the actuators).
- The temptation of programming the overall behaviour simply by means of nested IF-THEN-ELSE structures is natural, but this makes the code less extendible (not in the spirit of the subsumption architecture) and error prone.
- It is not trivial to decide the hierarchy of the levels and often this depends upon the personal viewpoint of the task (remember that the segmentation of behaviour depends on the observer).
- The issue of setting thresholds has to be faced.

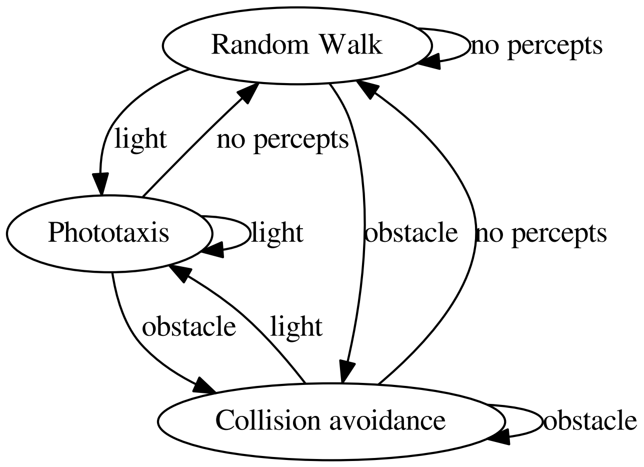
Questions

- What is the difference between this architecture and a FSM? (see appendix)
- Does the robot move smoothly? If not, why?

Take-home messages

- The architecture forces designers to neatly structure their code.
- The simpler the implementation of competences the easier to combine them. It is preferable to have more simple competences, than just a few but rather complicated.
- If the architecture is clean then it is easy to add new competences.

Appendix: An example of FSM implementing phototaxis with collision avoidance



Priorities between percepts: $obstacle \prec light \prec no\ percepts$

This FSM just implements an arbitration based on the priorities of the signals coming from the sensors.