Intelligent Systems
Lab 2: Programming a Reactive Agent

Eduardo Larios Fernández (A00569364)
Manuel Alejandro Lopez Perez (A01208598)
José Ramón Romero Chávez (A01700318)

Tecnológico de Monterrey, Campus Querétaro
9th February 2019

Introduction

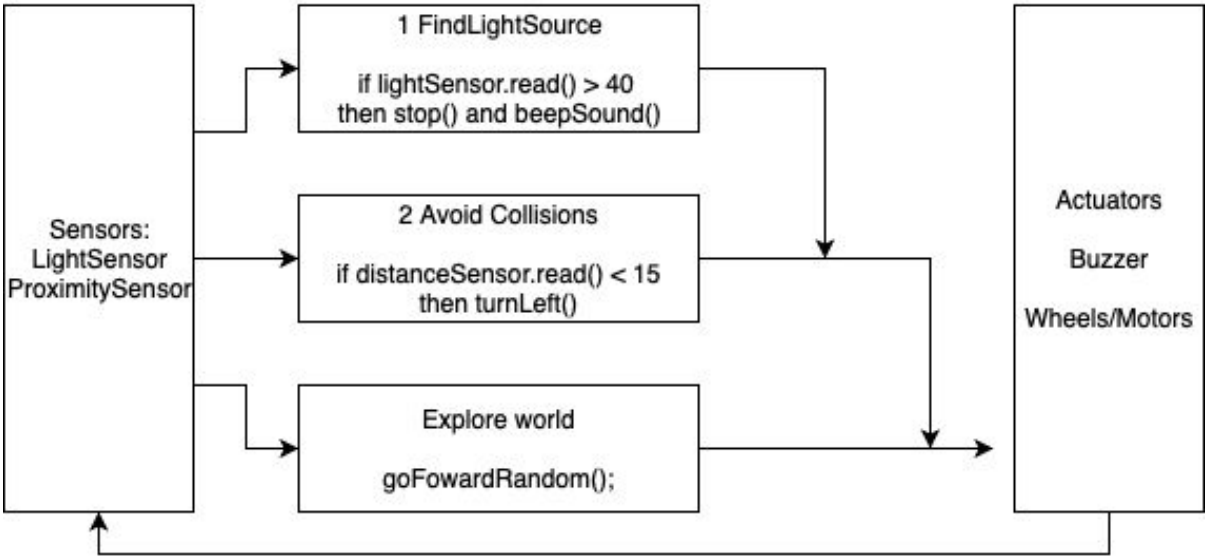
Our current robot architecture and the implemented changes were chosen because of the limitations of our previous laboratory. Ignoring the hardware limitations, learning about the subsumption architecture helped us understand how simple conditions can be used to create a reactive agent capable of responding to events occurring in its environment.

In our previous attempt to include more autonomy into our solution we found that working with a very large number of conditions quickly became complicated to write and debug. This is now solved by a set of priorities that decide which things should be executed by the agent.

This change of paradigm of how events should be processed makes it easier to understand what is the real objective of the robot in its environment. Previously we had an idea of what the robot did, but not “why”, now it is more easily understood.

In our case the re-structure had to be thorough as our previous solution had no idea about how to treat obstacles, and was only capable of drawing shapes, as well as very simple navigation. However, now our robot is capable of adequately navigating a small space without colliding with objects.

Architecture



Priority #1: If it finds a light source which measures more than 40 lumens (in a 0 to 255 scale) it will beep and remain in its current position until something interrupts the light readings.
Priority #2: If it finds an object closer than 15 cm, it will perform 90 degree turn to the left to avoid a collision with the object.
Priority #3: If neither of the previously described situations occur, it will continue to explore the world using randomly generated movement patterns.

Advantages and Disadvantages

A fundamental advantage of the reactive agent architecture is the speed of the agent's response, as the logic needed to respond to any situation is considerably simple and easy to follow.

All actions rely on the measures provided by the sensors and not any internal state, so it's hard for unexpected behavior to happen, unless the sensors malfunction. This kind of architecture is fundamental in embedded systems, where also memory limitations become an important consideration to take in account.

This kind of agents are specially suited for simple and atomic tasks which can be accomplished by using simple conditions about how to react to different events and how their priorities interact in the real world, as such, it is suited for autonomous work which requires high reliability.

An example can be seen in the following link: https://youtu.be/JWikCSmY_pM

Source Code

```
// Declara tus brikos
distancebk sensordistancia (PORT7);
temperaturebk sensortemperatura (PORT5);
lightbk sensorluz (PORT3);
buzzerbk bocina (PORT4);
motorbk motor1 (PORT2);
motorbk motor2 (PORT6);
int degree90 = 918;
int degree60 = 1224;

void firefight() {
  if (sensorluz.read() > 40) {
    motor1.set(STOP);
    motor2.set(STOP);
    bocina.set(ON);
  }
  else {
    bocina.set(OFF);
    if(sensordistancia.read(CM) < 15 && sensordistancia.read(CM) >= 0) {
      motor1.set(LEFT);
      motor2.set(LEFT);
      delay(918);
    }
    else {
      motor1.set(LEFT);
      motor2.set(RIGHT);

      delay(500);
      motor1.set(LEFT);
      motor2.set(LEFT);
      int d = random(0,918);
      delay(d);
    }
  }
}
/////Escribe tu código
code(){
  firefight();
}
/////Termina loop
```