

Inteligencia en Redes de Comunicaciones

# Robots – Resolución de Problemas II

Julio Villena Román, Raquel M. Crespo García,  
José Jesús García Rueda  
[jvillena, rcrespo, rueda}@it.uc3m.es](mailto:{jvillena, rcrespo, rueda}@it.uc3m.es)



Universidad  
Carlos III de Madrid

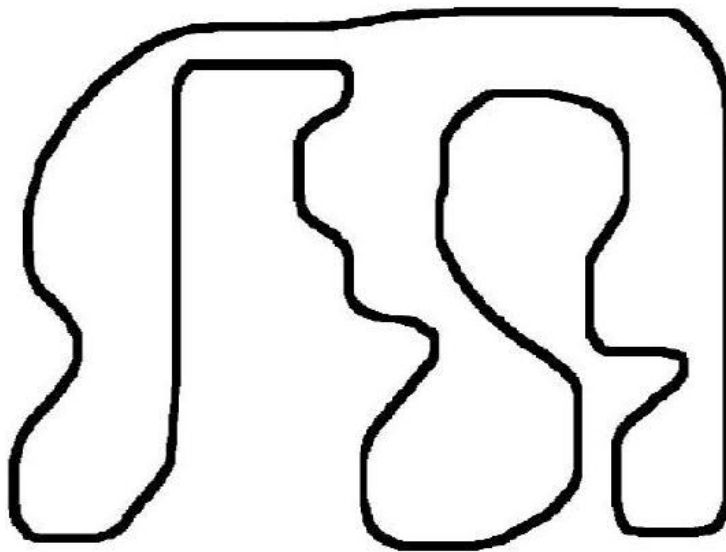
# Robots – Resolución de Problemas II

## Siguelíneas

### Objetivo

El objetivo de la práctica es construir un robot siguelíneas capaz de seguir un circuito (en forma de línea negra en el suelo) lo más rápidamente posible, compitiendo en una carrera contra los demás.

El circuito (no el mostrado en la siguiente figura) es una réplica a escala ☺ del Circuito de Catalunya en Montmeló, sede entre otros del Gran Premio de España de Fórmula 1.



**Figura 1: Ejemplo de circuito**

Puedes ver fotos y vídeos para obtener ideas de robots de otros años en el sitio web de Robot.IT! [en <http://robotit.it.uc3m.es/>]

### Normativa

#### 1. Circuito:

- La pista estará construida sobre un rectángulo blanco de papel o plástico, de aproximadamente 1.5 m de ancho y 3 m de largo, marcada con una línea negra gruesa (aprox. 1 cm de ancho).
- La línea de salida/meta estará claramente indicada.
- El circuito incluirá al menos un túnel, con las dimensiones apropiadas para evitar colisiones con los robots.
- El circuito incluirá al menos una rampa. Las rampas siempre estarán en tramos rectos, no en curvas.

- El circuito incluirá uno o varios obstáculos como baches, que no afectarán a la línea negra.
  - No habrá intersecciones.
2. **Rondas:** Habrá **2 rondas**, una en dirección de las agujas del reloj, y otra en sentido contrario a las agujas del reloj. En cada ronda, los robots tienen que completar **2 vueltas** en el menor tiempo posible.
  3. **Reintentos:** Si un robot se sale de la línea, se permite colocarlo manualmente en el punto donde se salió, hasta un máximo de **3 reintentos** por ronda. Saltarse parte del circuito se considera "salida de línea".
  4. **Tiempo límite:** Se permite un tiempo máximo de **4 minutos** para acabar el circuito. Si un robot no acaba el circuito en ese tiempo, se le asignará como tiempo de la ronda.
  5. **Puntuación:**
    - Los ganadores se decidirán según el tiempo total (la suma) en recorrer ambas rondas..
    - Además, se restarán **15 seconds** cada vez que el robot sea capaz de detectar túneles, rampas u obstáculos, de la manera que sea.
  6. **Tamaño y peso del robot:** Las dimensiones máximas del robot son 20 x 40 cm. El peso no está limitado.
  7. **Restricciones de diseño:**
    - Los robots deben moverse sobre ruedas. No se permiten piezas fijas que arrastren por el suelo.
    - No se permite usar pegamento, cinta aislante o gomas que no sean parte del kit de LEGO.
    - Específicamente, se prohíbe usar sustancias que mejoren la tracción del robot.
  8. **Control autónomo:** Una vez que el robot ha empezado su ronda, debe ser totalmente autónomo. La única intervención humana permitida es colocar el robot en su sitio cuando éste ha perdido la línea.

Estas reglas podrán adaptarse de forma flexible a las condiciones del laboratorio y de los participantes, siempre que se mantenga el espíritu del juego.

## **Pistas**

- Para seguir la línea, se pueden usar dos sensores de luz, de tal forma que la línea quede entre los dos. Así se puede saber si hay una curva y girar hacia el lado correspondiente. Lee la sección "Line follower" (siguelíneas) del anexo.
- Es importante construir un robot con un buen control de la dirección.
- Es buena idea que los sensores estén por delante, para anticipar el sentido de la curva.
- Es importante construir un robot con un buen control de la dirección para seguir bien las líneas y no perderse. El robot debe apoyarse en ruedas, no pudiendo usar patines ni otras piezas que arrastren por el suelo.

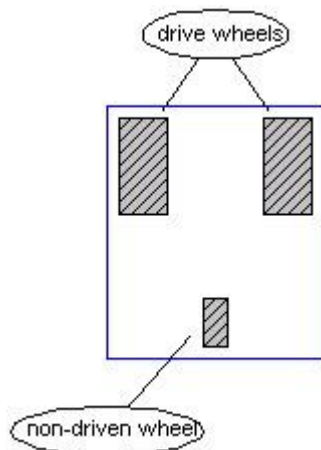
## Anexo: Basic Techniques in Robotics

[en <http://robotit.it.uc3m.es/basics/basics.htm#5>]

This section describes some basic techniques in Robotics that will probably be very useful for solving the challenges.

### *Differential Drive*

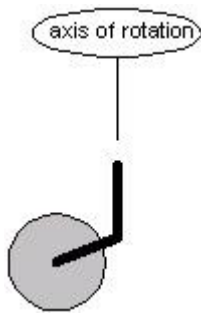
The **differential drive** is a basic locomotion technique that consists of a two-wheeled drive system with independent actuators for each wheel. This is essentially the same arrangement as that used in a wheelchair. The name refers to the fact that the motion vector of the robot is the addition of the independent wheel motions. This is similar to the mechanical differential in vehicles. The drive wheels are usually placed on each side of the robot and toward the front:



In the above diagram, the large gray rectangles are the **drive wheels**. Each drive wheel is connected to a different motor. It is easy to have an intuitive grasp of the way it behaves. If both drive wheels turn in tandem, the robot moves in a **straight line**. If one wheel rotates faster than the other, the robot follows a **curved path** inward toward the slower wheel. If the wheels turn at equal speed, but in opposite directions, the robot **pivots**. Thus, steering the robot is just a matter of varying the speeds of the drive wheels.

There are more or less complex mathematical expressions to calculate the speeds so that the robot will move exactly in a given path, such as make a turn or rounding a corner.

The small gray rectangle is a **non-driven wheel** (or a free wheel) which forms a tripod-like support structure for the body of the robot. Often, the non-driven wheel is a caster wheel, a small swiveled wheel used on office furniture:



Unfortunately, caster wheels can cause problems if the robot reverses its direction. Then the caster wheel must turn 180 degrees and, in the process, the offset swivel can impart an undesired motion vector to the robot. This may result in a translational heading error. However, if the robot always changes direction by moving forward and turning, a caster wheel may be okay.

Another alternative is a captive ball which does not use a swivel mechanism. In fact, a rolling device is not strictly necessary if the floor is smooth. The only negative is the increased friction component as that piece must slide along instead of rolling.

It can be difficult to make a differential drive robot move in a straight line. Since the drive wheels are independent, if they are not turning at exactly the same rate the robot will veer to one side. Making the drive motors turn at the same rate is a challenge due to slight differences in the motors and friction differences between the wheel and the ground. To ensure that the robot is traveling in a straight line, it may be necessary to adjust the motor RPM very often (many times per second). It is also very important to have accurate information on wheel position, usually coming from the odometry sensors. For example, Lego Mindstorms NXT motors include a rotation encoder which returns the position of the shaft with 1° resolution [NXT Motor Internals: <http://www.philohome.com/nxtmotor/nxtmotor.htm>].

There are other locomotion types such as synchro drive and car-type drive [Robot-Rats Locomotion Page: <http://groups.csail.mit.edu/drl/courses/cs54-2001s/locomotion.html>].

### *Line follower*

A **line follower** is a robot that can follow a path. The path can be visible like a black line on a white surface (or viceversa) or can be invisible like a magnetic field. The main idea is to continuously sense the line and maneuver the robot to stay on course, constantly correcting wrong moves using feedback mechanism. This is called a closed loop control system.

The simplest strategy is to use just **one light sensor** to detect the line and implement a simple "zigzag" method of line following where the robot is constantly turning left and right as it sees either side of the color boundary (i.e., detects white). The robot will be faster if a somewhat more complex method of following the line with the goal of going straighter and smoother is

developed. When the robot gets aligned with the color boundary that it is trying to follow, the robot will go straight as far as it can and make small corrections left and right to try to stay on the boundary without zigzagging as much. [nxtprograms.com examples: <http://www.nxtprograms.com/projects.html>].

However, if you have **2 light sensors**, then it is possible to make a line following strategy that is faster, smoother, and more reliable than with one sensor. Place both sensors on either side of the line so that they both see white when the robot is centered over the line. The **basic strategy** would be:

- When both sensors see white (e.g. > 75%), **go straight**.
- When the left sensor sees some black (e.g. 25-75%), make a **slight left turn**.
- When the left sensor sees all black (e.g. < 25%), make a **hard left turn**.
- When the right sensor sees some black, make a **slight right turn**.
- When the right sensor sees all black, make a **hard right turn**.

Obviously, the programming will be trickier because you must test both sensors at the same time, and you will also have to calibrate both sensors at the beginning.

#### *Dynamic light sensor calibration*

Sensors are one of the most important parts of a robot, so it is extremely important that they return a very accurate information. For certain types of sensors, just after setting up the type of sensor, a calibration process is needed. The **calibration** is the validation of the values that the sensor will return, specifically adjusted to the changing conditions of the environment.

When light sensors are used to detect lines on a surface, they must be calibrated to the actual colors and lighting conditions that will be present in the environment. This is important because the actual brightness seen by the light sensor will depend on many things, including the colors of the surface, how shiny the surface is, the exact height of the light sensor off of the surface, and the amount of other ("ambient") light in the room. Light sensor calibration is essential for accurate line following.

The basic idea is to read the light values for BLACK and WHITE and use the intermediate value as a **threshold** for decision, if the value is greater or lower than the threshold.

This threshold could be stored in a constant in the code, but this will force to recompile the program if the robot is operated in a different environment.

Instead, a **dynamic light sensor calibration** should be used. At the start of the program, the user is asked to place the sensor over the line (black) and then over the surface (white) and press the orange button after each, in order to calibrate the light sensor to the actual light and dark colors that will be used for the line following.

You may use this sample code:

```

// declare and init the values
int black = 0;
int white = 0;
int threshold = 0;

// sensor setup
SetSensorLight(IN_1);

// ask the user to place the sensor over WHITE and press left button
TextOut(0, LCD_LINE1, "White");
while(!ButtonPressed(BTNLEFT, true))
    ;
// wait for 1 second (to avoid button rebound)
Wait(1000);

// read 10 values of white (to reduce noise)
for(i=0; i<10; i++) {
    white += Sensor(IN_1);
}

// ask the user to place the sensor over BLACK and press left button
TextOut(0, LCD_LINE1, "Black");
while(!ButtonPressed(BTNLEFT, true))
    ;
// wait for 1 second (to avoid button rebound)
Wait(1000);

// read 10 values of black (to reduce noise)
for(i=0; i<10; i++) {
    black += Sensor(IN_1);
}

// calculate threshold
threshold = (white+black)/20;

// press left button when ready to start your program!!
TextOut(0, LCD_LINE1, "READY!");
while(!ButtonPressed(BTNLEFT, true))
    ;
// wait for 1 second (to avoid button rebound)
Wait(1000);

// go!

```