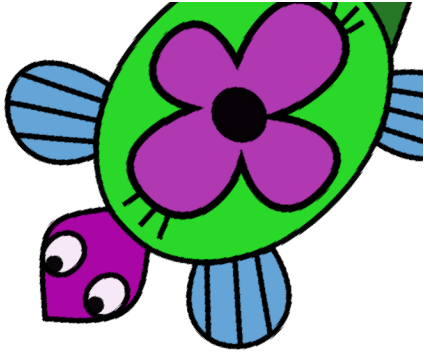


Capítulo 19

Utilización de la red con xLogo

19.1. La red: ¿cómo funciona eso?

En primer lugar es necesario explicar los conceptos básicos de la comunicación en una red para usar correctamente las primitivas de xLogo.

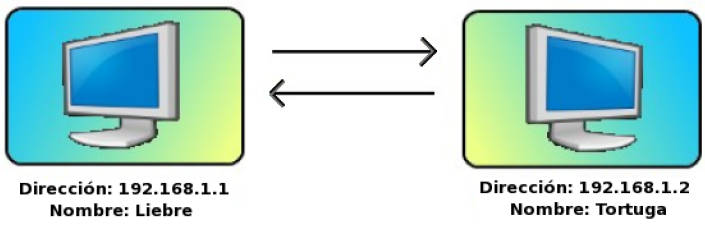


Figura: noción de red

Dos ordenadores pueden comunicarse a través de una red si están equipados con una tarjeta de red (llamada también tarjeta *ethernet*). Cada ordenador se identifica por una dirección personal: su dirección I.P. Esta dirección IP consta de cuatro números enteros comprendidos entre 0 y 255 separados por puntos. Por ejemplo, la dirección IP del primer ordenador del esquema de la figura es `192.168.1.1`

Dado que no es fácil acordarse de este tipo de dirección, también se puede hacer corresponder a cada dirección IP un nombre más fácil de recordar. Sobre el esquema anterior, podemos comunicar con el ordenador de la derecha bien llamándolo por su dirección IP: `192.168.1.2`, o llamándolo por su nombre: `tortuga`.

No nos extendamos más sobre el significado de estos números. Añadamos únicamente una cosa que es interesante saber, el ordenador local en el cual se trabaja también se identifica por una dirección: `127.0.0.1`. El nombre que se asocia con él es habitualmente `localhost`.

19.2. Primitivas orientadas a la red

xLogo dispone de 4 primitivas que permiten comunicarse a través de una red: `escuchatcp`, `ejecutatcp`, `chattcp` y `enviatcp`.

En los ejemplos siguientes, mantendremos el esquema de red de la subsección anterior.

- `escuchatcp`: esta primitiva es la base de cualquier comunicación a través de la red. No espera ningún argumento. Permite poner al ordenador que la ejecuta a la espera de instrucciones dadas por otros ordenadores de la red.

- `ejecutatcp`: esta primitiva permite ejecutar instrucciones sobre otro ordenador de la red.

  Sintaxis: `ejecutatcp palabra lista` → La palabra indica la dirección IP o el nombre del ordenador de destino (el que va a ejecutar las órdenes), la lista contiene las instrucciones que hay que ejecutar.

  Ejemplo: desde el ordenador `liebre`, deseo trazar un cuadrado de lado 100 en el otro ordenador. Por tanto, hace falta que desde el ordenador `tortuga` ejecute la orden `escuchatcp`. Luego, desde el ordenador `liebre`, ejecuto:

  ```
  ejecutatcp "192.168.2.2 [repite 4 [avanza 100 giraderecha 90]]
  ```

  o

  ```
  ejecutatcp "tortuga [repite 4 [avanza 100 giraderecha 90]]
  ```

- `chattcp`: permite chatear entre dos ordenadores de la red, abriendo una ventana en cada uno que permite la conversación.

  Sintaxis: `chattcp palabra lista` → La palabra indica la dirección IP o el nombre del ordenador de destino, la lista contiene la frase que hay que mostrar.

  Ejemplo: `liebre` quiere hablar con `tortuga`.
  `tortuga` ejecuta `escuchatcp` para ponerse en espera de los ordenadores de la red.
  `liebre` ejecuta entonces: `chattcp "192.168.1.2 [buenos días]`.
  Una ventana se abre en cada uno de los ordenadores para permitir la conversación

- `enviatcp`: envía datos hacia un ordenador de la red.

  Sintaxis: `enviatcp palabra lista` → La palabra indica la dirección IP o el nombre del ordenador de destino, la lista contiene los datos que hay que enviar. Cuando xLogo recibe los datos en el otro ordenador, responderá `Si` , que podrá asignarse a una variable o mostrarse en el **Histórico de comandos**.

Ejemplo: `tortuga` quiere enviar a `liebre` la frase `"3.14159 casi el número pi"`. `liebre` ejecuta `escuchatcp` para ponerse en espera de los ordenadores de la red. Si `tortuga` ejecuta entonces: `enviatcp "liebre [3.14159 casi el número pi]`, `liebre` mostrará la frase, pero en `tortuga` aparecerá el mensaje:

```
Qué hacer con [ Si ] ?
```

Deberíamos escribir:

```
es enviatcp "liebre [3.14159 casi el número pi]
```

o

```
haz "respuesta enviatcp "liebre [3.14159 casi el número pi]
```

En el primer caso, el **Histórico de comandos** mostrará `Si`, y en el segundo `"respuesta` contendrá la lista `[ Si ]`, como podemos comprobar haciendo

```
es lista? :respuesta
cierto
es :respuesta
Si
```

Con esta primitiva se puede establecer comunicación con un robot didáctico a través de su interfaz de red. En este caso, la respuesta del robot puede ser diferente, y se podrán decidir otras acciones en base al contenido de `:respuesta`.

Un pequeño truco: lanzar dos veces XLOGO en un mismo ordenador.

- En la primera ventana, ejecuta `escuchatcp`.
- En la segunda, ejecuta

```
ejecutatcp "127.0.0.1 [repite 4 [avanza 100 giraderecha 90]]
```

¡Así puedes mover a la tortuga en la otra ventana! (¡Ah sí!, esto es así porque `127.0.0.1` indica tu dirección local, es decir, de tu propio ordenador)

19.3. Robótica

Desde 2006, XLOGO es capaz de comandar una interfaz externa para robótica, (y así se presentó en el congreso Cafeconf - Aulas Libres). Las primitivas de comunicación por red que acabamos de ver, además de ser utilizadas para la tareas grupales en red, *chat*, comandar la tortuga en una PC desde otra PC, etc. permiten experimentar en robótica.

Me duele reconocer que mucho del trabajo realizado por Marcelo en este área se nos ha ido con él, pero intentaré explicar las cosas de forma lo bastante clara para que cualquiera pueda seguir el tema sin problemas.

Es aquí más que en ningún otro sitio del curso, donde agradeceré cualquier ayuda y/o corrección que usted, lector o lectora, sea capaz de proporcionar.

19.3.1. Presentación

Prólogo: Prof. María Cristina Moreno.

En los libros de ciencia ficción, el escritor, como un alquimista, le dió vida a los robots que hoy se incorporan a nuestra vida cotidiana; desde una simple cafetera, hasta el Mars Rover.

El objetivo es que tecnología y educación recorran juntas el camino hacia el futuro.
La propuesta es lograr que en casa y en la escuela se construyan y manejen pequeños robots didácticos.

Con estas premisas, Marcelo Duschkin nos dejó el proyecto **Mi Primer Robot**, basado en una interfaz electrónica simple y económica, TortuRob, que todos pueden construir libremente, al ser liberada bajo Licencia Creative Commons.

19.3.2. La electrónica

El currículo de la asignatura de **Tecnología** en la Enseñanza Secundaria nos presenta un bloque de control de procesos y robótica, pero habitualmente no se acompaña de una dotación adecuada en lo referente al material necesario para experimentar en el taller. Lo más habitual es aprovechar las tarjetas controladoras antiguas (tipo Inves, Enconor, CNICE, ladrillos Lego,...), con el problema que ello conlleva: su incompatibilidad con los sistemas Windows actuales. Por otro lado, tampoco es raro el caso de disponer de pocas unidades de los equipos citados, lo que añade un sobrecoste económico a la asignatura.

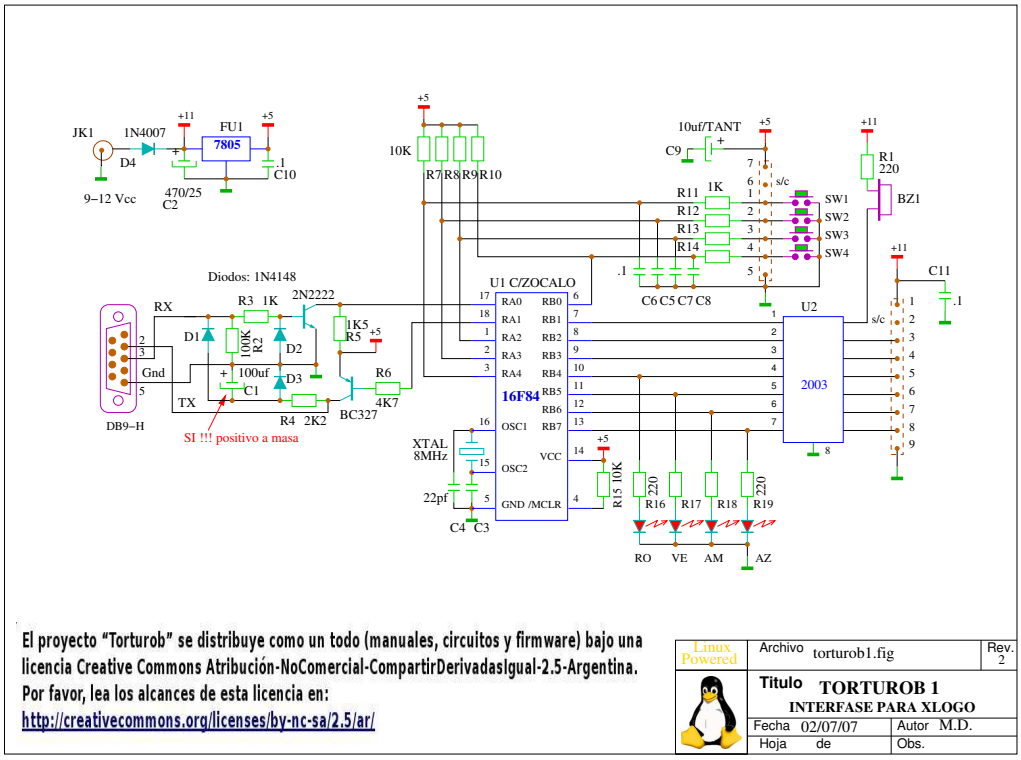
Nuestra propuesta de interfaz electrónica es el proyecto TortuRob.

El proyecto TortuRob

Es un circuito basado en un microprocesador PIC16F628A, que recibe comandos desde el PC, y contiene puertos de entrada y salida para controlar una mecánica de robot. También están incluídos 4 pulsadores y cuatro LEDs, con el objeto de usarlos en la etapa de aprender a usar el sistema. De esta manera, las primeras pruebas no necesitan de una mecánica a controlar.

La placa es de reducidas dimensiones (9,5 x 7 cm) y es muy fácil de armar por el aficionado a la electrónica. Los detalles técnicos para su construcción pueden descargarse de:

`http://downloads.tuxfamily.org/xlogo/robotica/TortuRob.zip`



La placa tiene interfaz serie, por ser el sistema más económico y sencillo, pero como XLOGO utiliza la conexión `ethernet`, es necesario hacer algún tipo de conversión. Lo más simple es una conversión por *software*. En Linux puede utilizarse un *script* Tcl/Tk liberado bajo licencia GPL:

`http://downloads.tuxfamily.org/xlogo/robotica/tcptty-es.zip`

que se ejecuta "*por detrás*" (*background*) de XLOGO.

Por desgracia, la versión en Visual Basic para Windows no es GPL y no puede proporcionarse un enlace al mismo (por razones obvias).

La otra solución es utilizar un módulo de *hardware* externo al PC, es decir, un convertidor `ethernet/rs232` de los hay muchos en el mercado. Esta solución, por supuesto, es más cara, pero tiene una aplicación interesante:

> En un aula con muchos ordenadores en red, se coloca un sólo módulo convertidor con el robot, y cualquier alumno lo comanda desde su puesto.

Finalmente, decir que tiene seis salidas (la séptima dedicada a un *buzzer*) y soporta cuatro sensores digitales, que pueden ser: contactos, pulsadores, fotorresistores u optoacopladores.

19.3.3. El lenguaje de la TortuRob

Evidentemente, para poner ejemplos sobre robótica debemos elegir un lenguaje. De acuerdo a nuestra elección sobre la placa, explicaremos el lenguaje de la TortuRob. Obviamente, entendiendo esta terminología, podremos utilizar los ejemplos en las tarjetas de que disponga el lector en su Centro.

El **protocolo de comunicación** con la TortuRob parte de las tres premisas siguientes:

1. La placa TortuRob nunca inicia una comunicación, solo responde.
2. Todo comando enviado recibe una respuesta.
3. Todos los comandos y respuestas tienen la estructura: `$Lx...x%[CR]`, donde:

   `$` → Identificador de inicio de comando.

   `%` → Identificador de fin de comando.

   `L` **y** `x...x` → `L` es una letra que identifica la acción (mayúscula si es comando, minúscula si es respuesta), mientras que `x...x` son los datos asociados a las órdenes/respuestas anteriores y su longitud es variable.

Con esta breve información, veamos unos ejemplos antes de entrar en detalle:

El semáforo

La TortuRob posee colores, concretamente, las salidas 3, 4 y 5 se corresponden con los LED rojo, verde y amarillo, respectivamente. ¿No invita eso a construir un semáforo?

En este programa sólo damos órdenes a la placa, concretamente a los LED y al zumbador. Los códigos asociados a los LED comienzan con `A` seguido del número del LED y un `0` ó `1` según queramos apagarlo o encenderlo, mientras que activamos el *buzzer* con `B` seguido del número de zumbidos que queremos que haga:

```
para semaforo
  haz "estado enviatcp "localhost [$A40%]  # apaga verde
  haz "estado enviatcp "localhost [$A50%]  # apaga amarillo
  haz "estado enviatcp "localhost [$A31%]  # enciende rojo
  haz "estado enviatcp "localhost [$B1%]   # un beep (semaforo para ciegos)
  espera 100
#
  haz "estado enviatcp "localhost [$A30%]  # apaga rojo
  haz "estado enviatcp "localhost [$A50%]  # apaga amarillo
  haz "estado enviatcp "localhost [$A41%]  # enciende verde
  haz "estado enviatcp "localhost [$B2%]   # dos beep (semaforo para ciegos)
```

```
  espera 100
#
  haz "estado enviatcp "localhost [$A30%]  # apaga rojo
  haz "estado enviatcp "localhost [$A40%]  # apaga verde
  haz "estado enviatcp "localhost [$A51%]  # enciende amarillo
  haz "estado enviatcp "localhost [$B3%]   # tres beep (semaforo para ciegos)
  espera 30
#
  semaforo                                 # recursivo, volvemos a empezar
fin
```

Simple, ¿no?

Alarma

Creemos un botón de alarma. Al pulsarlo, aparecerá escrito en pantalla un aviso:

ALARMA!

En este programa vamos a activar la zona de los pulsadores y leer la respuesta que da la controladora. El código asociado a los pulsadores lleva asociada la letra `D`. El programa resulta ser:

```
para alarma
  borrapantalla
  poncolorlapiz 1                          # color lapiz -> rojo
  ponfuente 30
  repite 1000
    [ haz "puls enviatcp "localhost [$D0%]  # lee pulsadores
      si :puls= [$d1000%]                   # respuesta
        [ rotula "ALARMA!
          ocultatortuga
          alto ]
      espera 10 ]
fin
```

La tortuga rotulará en pantalla `ALARMA!` al apretar uno de los pulsadores de la controladora (concretamente, el SW1).

Moviendo a la tortuga

En la misma línea del anterior, vamos a utilizar un programa recursivo con el que controlemos los movimientos de la tortuga mediante los cuatro pulsadores de la TortuRob, de modo similar al programa de ejemplo de la sección 3.1 en la página 20:

- El botón 1 hará que la tortuga avance 10 pasos.
- El botón 2 hará que la tortuga se dé la vuelta.
- El botón 3 hará que la tortuga gire 90 grados a la izquierda.
- El botón 4 hará que la tortuga gire 90 grados a la derecha.

```
para tortu
  haz "puls enviatcp "localhost [$D0%]  # lee pulsadores
  si :puls = [$d1000%]                  # Pulsaron el boton 1
    [ avanza 10 espera 5 ]      # esperamos 5/60 seg para evitar rebotes
  si :puls = [$d0100%]                  # Pulsaron el boton 2
    [ giraderecha 180 espera 5 ]
  si :puls = [$d0010%]                  # Pulsaron el boton 3
    [ giraizquierda 90 espera 5 ]
  si :puls = [$d0001%]                  # Pulsaron el boton 4
    [ giraderecha 90 espera 5 ]
  tortu
fin
```

Vemos que las respuestas de la controladora son muy simples: tras inicializarla con `$D0%`, las respuestas vienen en la posición correspondiente al pulsador, y es un `1` ó un `0` según esté abierto o cerrado.

19.3.4. Los comandos de la TortuRob

Hemos visto tres ejemplos sencillos de la controladora. Veamos ahora todo lo que puede hacer. Recordemos:

1. La placa TortuRob nunca inicia una comunicación, solo responde.
2. Todo comando enviado recibe una respuesta.
3. Todos los comandos y respuestas tienen la estructura: `$Lx...x%[CR]`, donde:

   `$` → Identificador de inicio de comando.

   `%` → Identificador de fin de comando.

   `L` **y** `x...x` → `L` es una letra que identifica la acción (mayúscula si es comando, minúscula si es respuesta), mientras que `x...x` son los datos asociados a las órdenes/respuestas anteriores y su longitud es variable.

   Los valores que pueden tomar son:

   `$Gsssssss%` → Apaga o activa las salidas correspondientes:

   - `s1`: Apaga/activa la salida 1 (CO1, pin 3).

- `s2`: Apaga/activa la salida 2 (CO2, pin 4).
- `s3`: Apaga/activa la salida 3, LED rojo (CO1, pin 5).
- `s4`: Apaga/activa la salida 4, LED verde (CO1, pin 6).
- `s5`: Apaga/activa la salida 5, LED amarillo (CO1, pin 7).
- `s6`: Apaga/activa la salida 6, LED azul (CO1, pin 8).
- `s7`: Apaga/activa la salida 7, *buzzer*.

`s` puede tomar los valores

`0:` apagar.

`1:` encender permanentemente.

`a-z:` encender durante un tiempo que depende de la letra introducida (desde medio segundo a 8 segundos).

La respuesta a las órdenes anteriores viene en la forma:

- `$0%`: El comando se ejecutó correctamente.
- `$1%`: Error, no se pudo ejecutar el comando.

Ejemplo: `$G00100dy%`:

– Apaga las salidas 1, 2, 4 y 5.

– Activa la salida 3 de forma permanente.

– Activa la salida 6 durante `d` fracciones de segundo.

– Activa la salida 7 durante `y` fracciones de segundo.

`$Asx%` → Apaga o activa la salida `s` durante `x` fracciones de segundo. El valor de `s` puede ser:

- `1`: Salida 1 (CO1, pin 3).
- `2`: Salida 2 (CO2, pin 4).
- `3`: Salida 3, LED rojo (CO1, pin 5).
- `4`: Salida 4, LED verde (CO1, pin 6).
- `5`: Salida 5, LED amarillo (CO1, pin 7).
- `6`: Salida 6, LED azul (CO1, pin 8).
- `7`: Salida 7, *buzzer*.

mientras que `x` puede tomar los valores:

`0:` Apaga la salida.

`1:` Enciende la salida permanentemente.

`a-z:` Enciende la salida un tiempo que oscila entre medio segundo y 8 segundos.

La respuesta a las órdenes anteriores viene, como antes, en la forma:

- `$0%`: El comando se ejecutó correctamente
- `$1%`: Error, no se pudo ejecutar el comando.

Ejemplo: `$A4y%`: Activa el LED verde durante `y` fracciones de segundo.

`$Bn%` → Activa el *buzzer* o zumbador, `n` veces.
La respuesta es, de nuevo:

- `$0%`: El comando se ejecutó correctamente
- `$1%`: Error, no se pudo ejecutar el comando.

Ejemplo: `$B3%`: El *buzzer* emite 3 *beeps*.

`$Dn%` → Lee el estado de entradas y salidas. `n` puede ser:

`0:` Grupo de 4 pulsadores o conector CO2.

`1:` Contenido del contador binario asociado a CO2, pin1 (SW1).

`2:` Estado de salidas activadas.

La respuesta viene precedida de una `E`:

- `$E%`: para cualquier valor de `n` si hubo error.
- `$dwxyz%`: si `n` fue 0. `wxyz` hacen referencia a los pulsadores SW1, SW2, SW3 y SW4 (o a masa CO2, pin1, pin2, pin3 y pin4) respectivamente, y serán `0` si el pulsador está cerrado y `1` si está abierto.
- `$cxyz%`: si `n` fue 1, `xyz` es un valor entre `000` y `255`, representando el valor del contador binario asociado a CO2, pin1 (SW1).
- `$assssssss%`: si `n` fue 2, hace referencia a las 7 salidas ya explicadas, y seá `0` si la salida correspondiente está apagada y `1` si está activa.

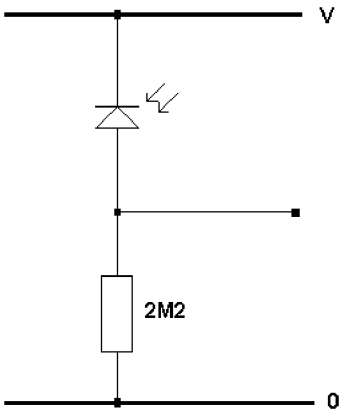
Ejemplos:
`$d0010%`: Pulsadores SW1, SW2 y SW4 abiertos, SW3 cerrado.
`$c035%`: El contador registró 35 cierres de SW1 desde la última lectura.
`$a0010000%`: Solo la salida 3 está activa (*timer* 3 contando).

`[CR]` → Cierre. Lo asigna el sistema de comunicaciones (es decir, el *Enter*)

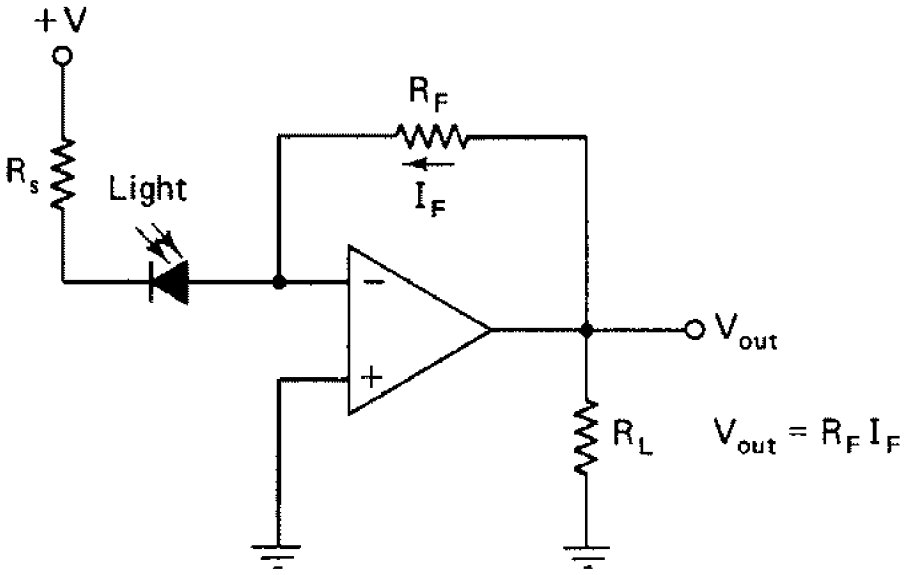
Sigue la luz

Analicemos ahora un *proyecto* con mayor complejidad. Vamos a hacer que nuestra tortuga responda a estímulos luminosos (en una habitación oscura y con una linterna, por ejemplo), sustituyendo los pulsadores por fotodiodos.

El fotodiodo es un diodo que, conectado en inversa, genera una intensidad proporcional al nivel de luz recibida; aunque podríamos intentar disparar directamente los sensores mediante la intensidad del diodo, el esquema sería muy susceptible a la luz ambiente, por lo que es mejor conectarlo a masa mediante una resistencia *pull–down*, de forma que la tensión en la misma será directamente proporcional a la intensidad/luz recibida:



Por supuesto el concepto básico puede refinarse tanto como queramos, ya sea mejorando el circuito de disparo (ver figura) o sustituyendo el tipo de sensor de activación, por ejemplo haciendo que la tortuga reaccione a sonidos en lugar de luces:



La respuesta de la tortuga a los sensores deberá ser:

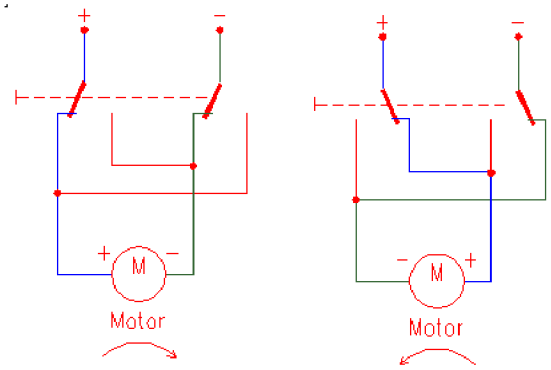
- Si le da la luz por un lateral, haremos que gire 90°.
- Si se ilumina por delante o por detrás, irá en esa dirección.

El programa se detendrá cuando se pulse la tecla "Escape".

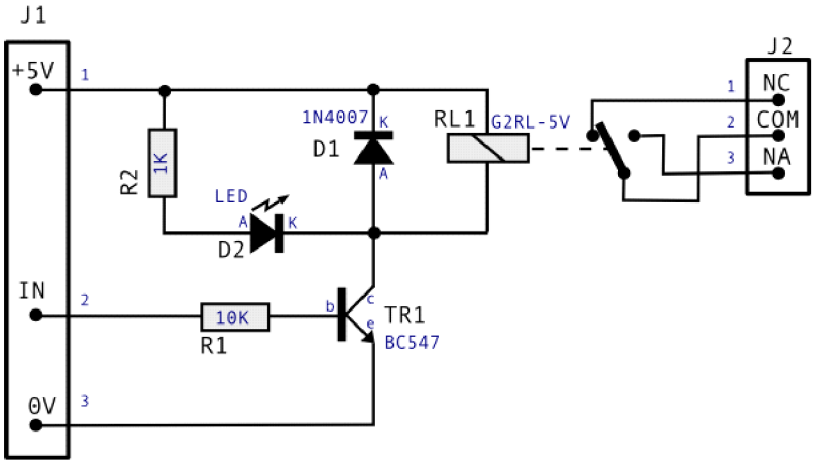
Necesitamos montar la placa en un robot que tenga una base con dos motores y cuatro sensores, uno a cada lado, otro en el frontal y el cuarto en la parte posterior.

Los motores se montarán a ambos lados de la base, de modo que:

- Activando los dos, se moverá hacia delante.
- Activando solo el derecho, girará a la izquierda y viceversa.
- Para lograr la marcha atrás, conectamos un inversor de giro en `CO1`. Invertiremos la fase de los motores con los dos contactos del relé (ver figura).



Como tenemos que controlar dos motores, y el inversor de giro afecta a ambos a la vez, los pondremos en paralelo. Además, conectaremos en serie con cada motor el contacto del relé que regula su puesta en marcha, permitiendo que trabajen independientemente.
Por último, y dado que no es conveniente intentar activar los relés directamente, deberemos agregar un transistor en modo interruptor a cada puerto de disparo, si es que el entrenador no está instalado.



Ha llegado el momento de realizar el programa. Elegimos que el LED "rojo" mueva el motor izquierdo y "LED verde" el derecho; encender ambos a la vez hace que el robot avance, y activando `CO1` llamamos al inversor de giro. Por tanto:

- el fotodiodo asociado al pulsador 1 irá detrás (marcha atrás),
- el fotodiodo 2 en el costado derecho (enciende rojo → motor izquierdo en solitario),

- el fotodiodo 3 en el lado izquierdo (enciende verde → motor derecho en solitario)
- el cuarto fotodiodo en el frontal (ambos encendidos → hacia delante).

Traducido a lenguaje TortuRob:

`[$d1000%]` → `[$Ga0aa000%]`: Pulsador 1, activa CO1, rojo y verde

`[$d0100%]` → `[$G00a0000%]`: Pulsador 2, activa rojo

`[$d0010%]` → `[$G000a000%]`: Pulsador 3: activa verde

`[$d0001%]` → `[$G00aa000%]`: Pulsador 4: activa rojo y verde

El programa resulta::

```
para sigueluz
  haz "puls enviatcp "localhost [$D0%]
  si :puls = [$d1000%]
    [ es enviatcp "localhost [$Ga0aa000%] ]      # Activar inversion
  si :puls = [$d0100%]
    [ es enviatcp "localhost [$G00a0000%] ]      # Activar izquierdo
  si :puls = [$d0010%]
    [ es enviatcp "localhost [$G000a000%] ]      # Activar derecho
  si :puls = [$d0001%]
    [ es enviatcp "localhost [$G00aa000%] ]      # Activar ambos
  si tecla? [ si leetecla = 27 [alto] ]
  espera 10
  sigueluz
fin
```