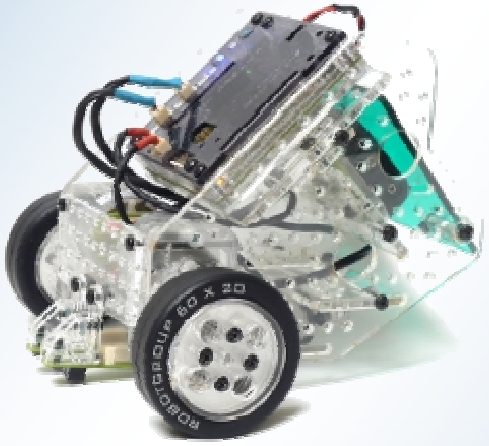


Curso de Robótica

Guía de teoría y actividades

Octavio J. da Silva Gillig

Mónica J. Paves Palacios

Julián U. da Silva Gillig

ROBOTGROUP
robótica para la acción

multiplo®

Autores
Octavio J. da Silva Gillig
Mónica J. Paves Palacios
Julián U. da Silva Gillig

IMPRESO EN ARGENTINA POR ROBOTGROUP®
http://robotgroup.com.ar
http://roboticaeducativa.net

Descargo de responsabilidad / Disclaimer

Contenido

1. Introducción

Introducción

El siguiente curso pretende introducir al alumno en los conocimientos básicos de la Robótica. Para ello, se planificaron una serie de lecciones en las cuales se estudian diversas partes del robot y se enseña a utilizarlas con el fin de entender su función y aplicación finales. El presente curso trata por separado cada una de estas partes pero las integra progresivamente para que el estudiante pueda lograr una mejor comprensión de los temas y poder aplicarlas en sus proyectos personales.

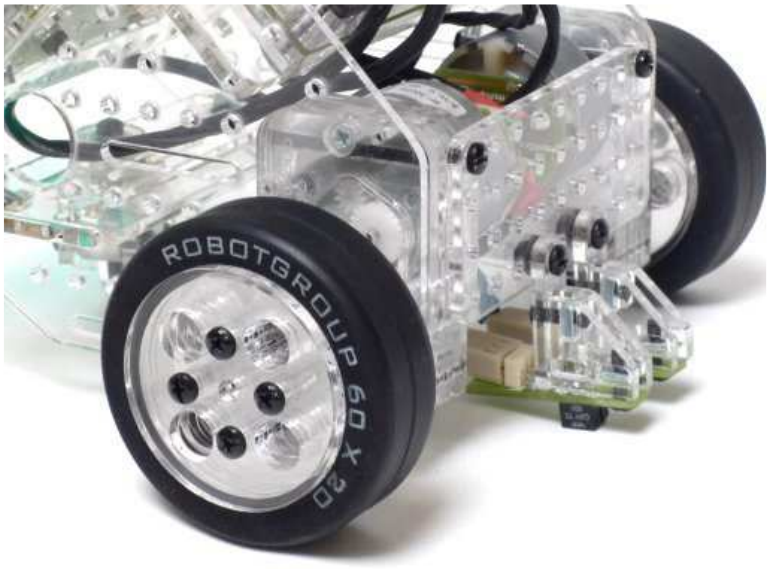
Robótica educativa

La Robótica Educativa, tan de moda en los últimos años, implica la enseñanza con robots en distintas áreas del saber. Debido al carácter interdisciplinario de la Robótica, es cada vez más común que se utilicen robots para enseñar conceptos de computación, electrónica, física, mecánica, matemáticas e incluso diseño. Pero esta disciplina tan compleja también trae al aula discusiones éticas debido a la incertidumbre que provoca en muchas personas el posible alcance de la automatización en la sociedad.
Entre las virtudes que encuentran los docentes al utilizar robots para la enseñanza se destacan el hecho de poder materializar el pensamiento abstracto del alumno en misiones concretas que los robots tendrán que llevar a cabo. También se intenta fomentar el trabajo grupal en el cual se discuten diversas formas de resolver un problema, lo cual ayuda a los estudiantes a trabajar en conjunto con un fin determinado.
Sin embargo, la integración de áreas y el desarrollo del pensamiento lógico parecen ser algunos de los resultados más interesantes de esta disciplina tan nueva pero con tanto impacto en la enseñanza actual.

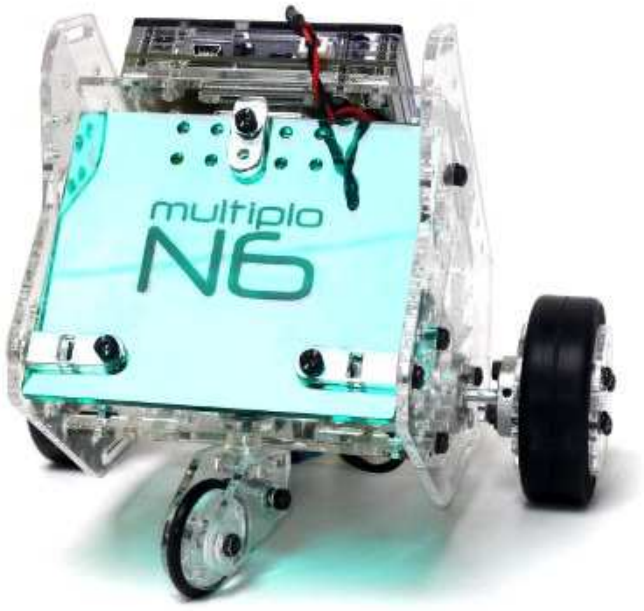
Descripción de la tecnología utilizada en el curso

Este curso de Robótica se realizará utilizando el nuevo robot Multiplo N6 realizado por RobotGroup. Este robot fue pensado especialmente para educación, pudiendo ser adaptado a las necesidades y gustos de cada usuario.

El mismo posee tracción diferencial con dos cajas reductoras con engranajes de alta resistencia y bujes metálicos. Este robot, al igual que el Multiplo N10, son parte del sistema constructivo Multiplo, el cual permite desarmar y reconfigurar el robot a gusto. También cuenta con dos sensores infrarrojos que, colocados en su parte delantera como se muestra en la figura, le permiten seguir una línea negra sobre un fondo blanco. La imagen muestra el detalle de la caja reductora y los sensores montados en su parte frontal.



Además, el robot Multiplo N6 posee 6 entradas para sensores analógicos de 10 bits, con fichas estándar para sensores Multiplo. Estas entradas pueden funcionar también como salidas digitales programables de hasta 40 mA.
La alimentación de la placa está dada por un avanzado sistema que permite entregar hasta 12 V a los motores partiendo de sólo 3 pilas AA. Además, eleva la tensión de alimentación de la lógica, permitiendo la conexión de todo tipo de sensores estándar y otros accesorios Multiplo de 5 V. La siguiente imagen muestra una vista posterior del robot Multiplo N6. En la misma se puede ver la tapa que protege la alimentación del robot.



Datos técnicos de la placa DuinoBot 1.2

Controlador DuinoBot	Procesador	Microcontrolador Atmel AVR ATMega32U4 ~14-16 MIPS @ 16 MHz.
		32 KBytes de memoria Flash (programa) automodificable. 2,5 KBytes de memoria SRAM. 1 KByte de EEPROM (datos no volátiles).
		5 Timers, conversor A/D 10 bits (12 canales multiplexados), comparador analógico, 31 fuentes de interrupción, puertos SPI, 2WI, USART, JTAG.
	Comunicaciones	USB 2.0 / Compatible Arduino.
		Puerto serie (USART) TTL para conexión de módulos de comunicaciones Multiplo.
	Salidas motores	Doble puente H MOSFET de alto rendimiento para motores de 2.5 a 13.5 V, corriente promedio de 1.2 A y pico de 3.2 A.
	I/O	6 entradas para sensores analógicos de 10 bits, con ficha estándar para los sensores Multiplo. Cada entrada es además configurable como salida digital de 40 mA (200 mA como máximo entre todos los pines).
		Más entradas y salidas disponibles en los conectores estándar Arduino.
		El procesador tiene también UART y buses I2C y SPI por hardware.
	Interfaz de usuario	Buzzer que permite la generación de tonos a diferentes frecuencias.
		LED de usuario.
		LED indicador de 5V (alimentación lógica).
		LED indicador de alimentación de motores (6V / 7V).
		4 LEDs indicadores de sentido de giro de los motores.
		Pulsador de Reset.
		Pulsador de Usuario/Run.
	Alimentación	Sistema de alimentación que puede entregar hasta 12 V a partir de 3 pilas AA (o cualquier celda de 3.6 V, ya sea NiMH o Li-Ion). Además eleva la tensión de la lógica permitiendo la conexión de todo tipo de sensores estándar y otros accesorios Multiplo de 5V.
	Software	Compatible con Arduino
		Puede ser programado de forma nativa stand-alone en C/C++, Bitlash y otros lenguajes de alto nivel.
		También puede ser utilizado en modo remoto desde lenguajes que corran en la PC.
	Estructura física	Piezas estructurales Multiplo.
	Ampliación	Anclaje estándar para acoplarle Shields Arduino-compatibles (como los shields WiFi, Ethernet, ZigBee, motores extra, etc.).

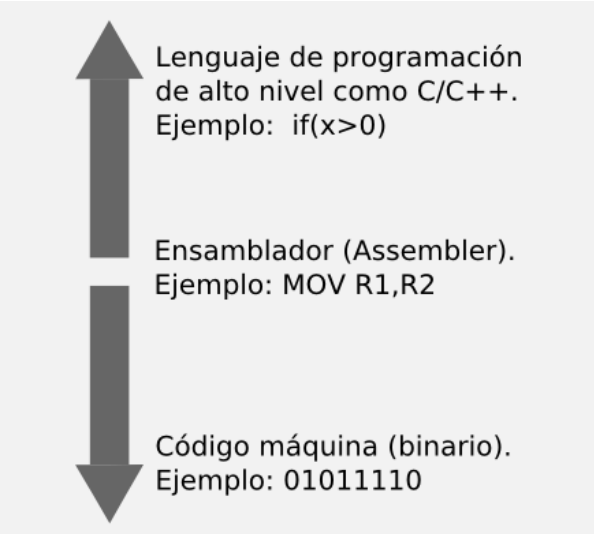
Programación de robots

Como vimos antes, una de las principales partes de un robot es el microprocesador. Este será programado para darle inteligencia al robot. Luego, cualquier decisión que tome, estará basada en el código que hayamos escrito previamente.

En primer lugar, hay que tener en cuenta que, para programar un robot, tendremos que escribir el programa que queremos que este ejecute. Dicho programa lo escribiremos en el entorno Arduino 0022 que viene con cd del robot. Luego, este entorno generará un archivo que, cuando le demos la orden a la computadora y, teniendo el robot conectado y prendido, será descargado al robot para que este pueda ejecutar el código que escribimos. Es importante destacar el hecho de que podremos saber si nuestro programa tiene cierto tipo de errores sin necesidad de conectar el robot a la computadora. En esta primera parte, estudiaremos algunos conceptos indispensables para empezar a programar. Para estos ejemplos, no necesitaremos tener el robot conectado a la pc.

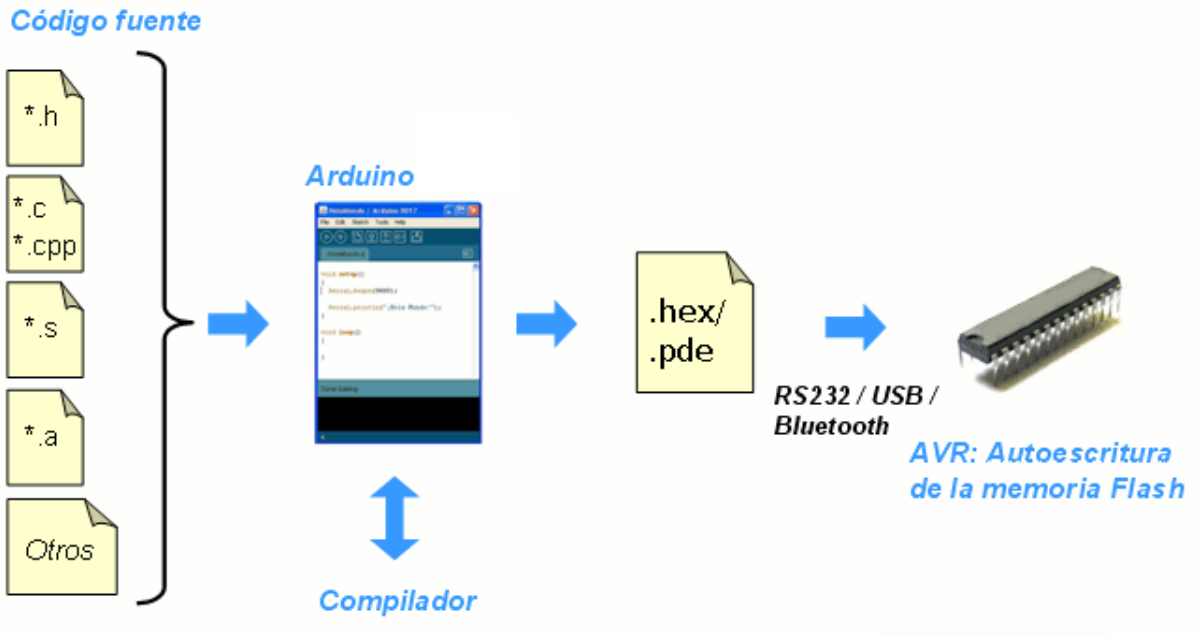
Al igual que sucede con la comunicación entre personas, para comunicarnos con el robot y darle órdenes tenemos que usar un lenguaje específico. En nuestro caso, el lenguaje elegido es C/C++. Esto simplemente significa que cuando programemos tendremos que respetar una serie de reglas semánticas y sintácticas para que el robot entienda lo que queremos que haga. Estas reglas, son las que definen el lenguaje que utilizamos. Para esto, tendremos que instalar un entorno de programación en el que escribiremos las órdenes en el lenguaje escogido. Este entorno de programación es un programa que entiende el lenguaje C/C++ y será el encargado de traducir nuestras órdenes a un lenguaje que el procesador pueda entender.

En este punto conviene parar para repasar algunos conceptos importantes. En primer lugar, no hay que olvidarse que la computadora, el robot y cualquier otro dispositivo que utilice un procesador, solamente entiende cadenas de 1s (unos) y 0s (ceros) a los que llamamos "bits". Esto es lo que se conoce como código binario, ya que solamente utiliza dos tipos de dígitos (0 y 1). Luego, estos unos y ceros se reúnen en grupos de ocho llamados bytes. Debido a que el procesador trabaja con estas secuencias de bits es que necesitamos un entorno de programación que entienda el lenguaje C/C++ con el que vamos a darle las órdenes a la computadora para que lo traduzca al lenguaje con el que trabaja el procesador (código binario).



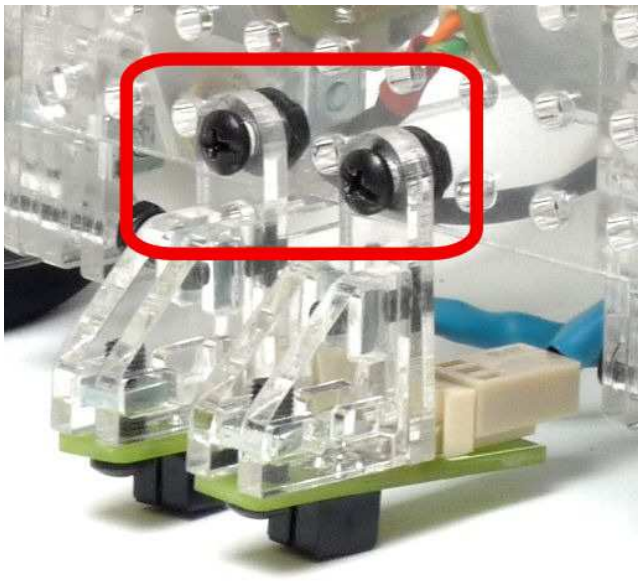
En el caso de los robots de Multiplo se ha decidido utilizar el entorno de programación Arduino 0022 debido a la gran cantidad de documentación que posee y a la gran comunidad de usuarios que lo programan. Esto hace que sea mucho más fácil para el alumno aprender y llevar a cabo distintos proyectos. Por un lado debido a la cantidad de código escrito por otras personas pertenecientes a la comunidad de usuarios y también por la gran cantidad de foros en los cuales se puede participar y hacer consultas sobre distintos temas. La compatibilidad de tecnologías es una característica cada vez más deseada y es por eso que RobotGroup buscó hacer su sistema constructivo de robots compatible con un entorno de programación tan difundido y de tan buena calidad como es Arduino 0022.

Cuando escribamos programas en lenguaje C/C++ estaremos escribiendo o generando archivos cuya extensión será .cpp. Estas son las iniciales de C++ en inglés (C Plus Plus). Es importante no estar cambiando de lugar estos archivos una vez escritos ya que el entorno de programación no solo traducirá lo que escribimos sino que hará algunas otras acciones, como por ejemplo "linkear" los archivos que sean necesarios para ejecutar un determinado programa. Esto se debe a que hay programas que contienen muchos archivos con código necesario para su ejecución y son escritos en distintos lugares para mantener cierto orden a la hora de hacer modificaciones. Por ejemplo, se suelen agrupar en un archivo .cpp todas las órdenes utilizadas para manejar motores y en otro archivo .cpp las órdenes para manejar un determinado tipo de sensores. Luego, nuestro entorno de programación utilizará un compilador para que ninguno de estos archivos sea ignorado. Este compilador viene integrado con el entorno y es el encargado de convertir lo que escribimos en código entendible para el procesador. De la misma manera, es el encargado de linkear los archivos y es quien nos dirá si el programa que escribimos tiene errores o no. A continuación se muestra una imagen que ilustra este hecho. En primer lugar, tenemos archivos con diversas extensiones entre las que se encuentran los .c y .cpp. Estos archivos son editados y modificados por nosotros en el entorno Arduino 0022. Luego, es el entorno el encargado de compilar estos archivos y convertirlos en archivos con extensión .hex o .pde para que el procesador los pueda entender y ejecutar.

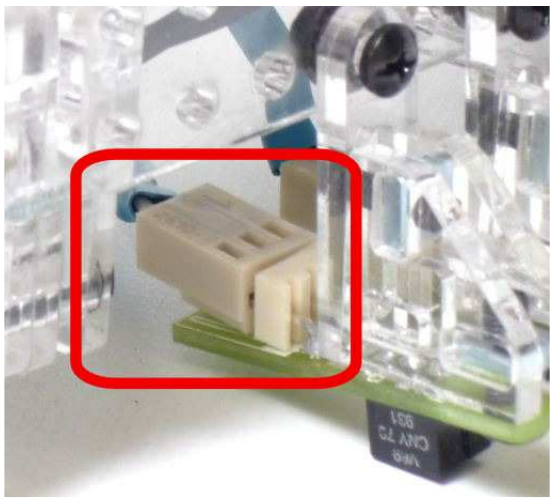


Montaje de los sensores infrarrojos frontales

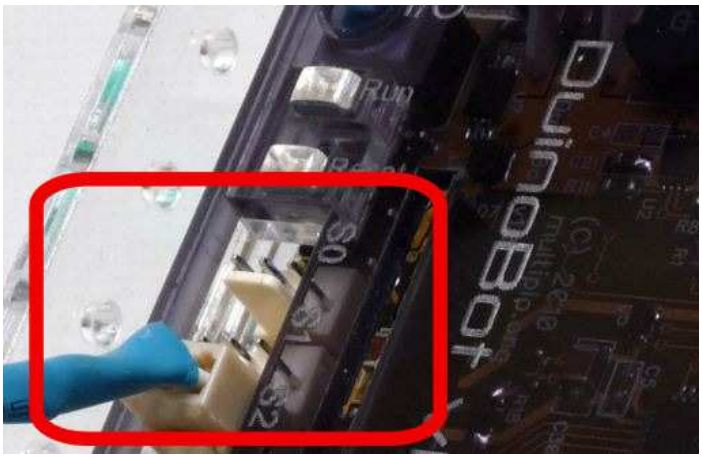
A continuación vamos a ver cómo instalar los sensores infrarrojos frontales en el robot de tal manera que podamos hacer una actividad en la cual el robot pueda seguir una línea negra opaca dibujada sobre un fondo blanco.
Para instalar los sensores, lo primero que hay que hacer es montarlos con dos tornillos cortos en la parte delantera. La posición adecuada es el centro de la pieza frontal dejando solo un espacio entre cada tornillo. La siguiente figura muestra en detalle cómo deberían quedar.



Una vez montados, hay que enchufar los sensores a la placa DuinoBot. Para esto, se debe conectar un extremo del cable al sensor, como se muestra en la siguiente figura:



Luego, se debe conectar el otro extremo del cable a la placa DuinoBot. Uno de los sensores, se conectará en la salida llamada S0 y el otro se conectará en la salida llamada S1. En caso de notar que el robot no se comporta como corresponde, es posible que haya que conectar los sensores al revés, o sea, el que estaba conectado a la salida S0 en la S1 y el S1 en la S0. La siguiente figura muestra el detalle de las conexiones en la placa. El cable que se ve está conectado a la salida S2 dejando libres las salidas S0 y S1 necesarias para esta actividad.



Instalación de drivers de la placa controladora

Antes de empezar a utilizar los robots Multiplo fabricados por RobotGroup, tendremos que instalar los drivers de comunicación de la placa controladora en la computadora que utilicemos. En esta primera sección se describirán los pasos para instalar los drivers utilizando el entorno Minibloq. Luego se describirán los pasos a seguir en caso de utilizar el entorno Arduino (que aquí llamaremos indistintamente como "DuinoPack", que es la versión de Arduino IDE modificada para trabajar con el equipamiento Multiplo). Es importante aclarar que una vez instalados los drivers utilizando cualquiera de los dos entornos, no será necesario volver a instalarlos para el otro. Por ejemplo, si primero se instalan correctamente los drivers utilizando el programa Minibloq, se podrá utilizar DuinoPack sin necesidad de volver a instalar los drivers.

Requerimientos de hardware y software

Requerimientos mínimos de hardware:

Intel Celeron o superior.
1GB Memoria RAM.
200 MB de espacio libre en disco.

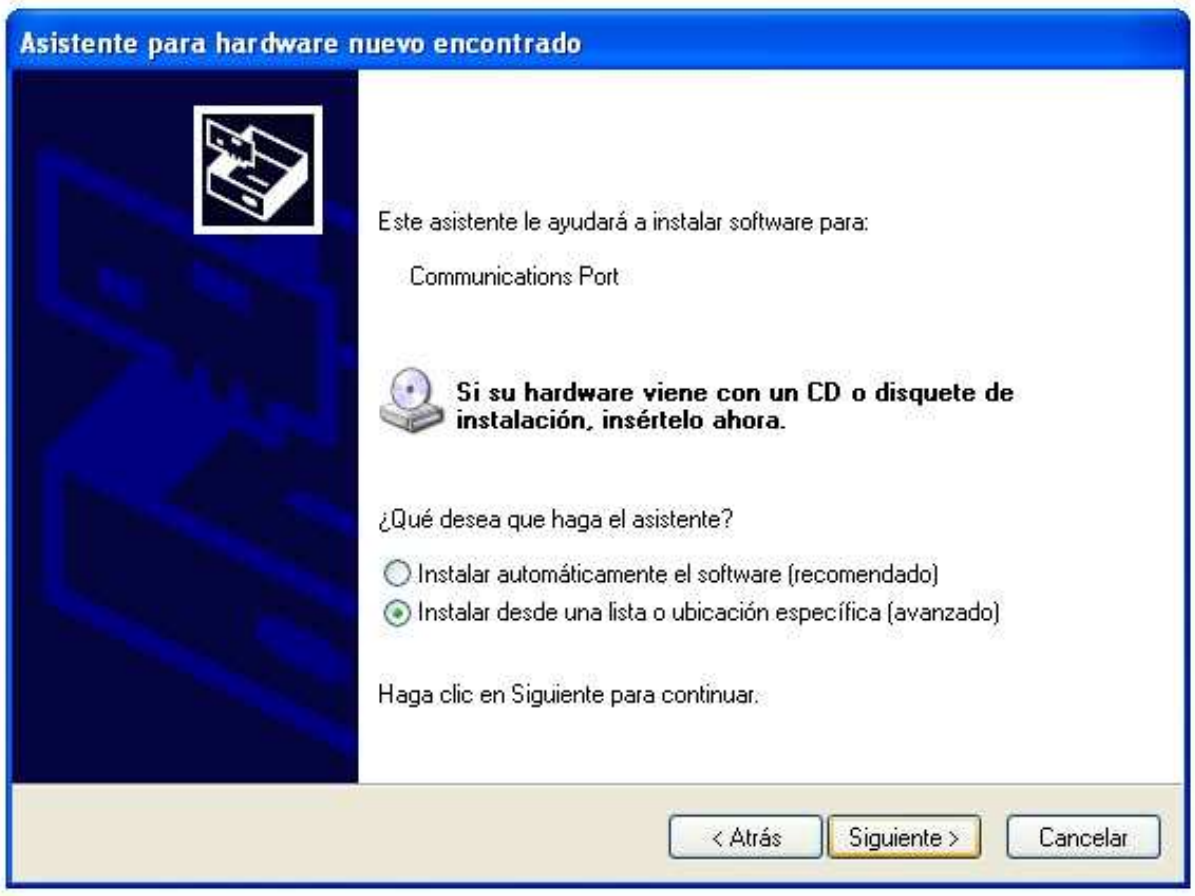
Requerimientos de software:

Windows XP
Windows Vista
Windows 7

Instalación de drivers utilizando Minibloq en Windows XP

Para instalar los drivers de comunicación necesarios para utilizar Minibloq con los robots Multiplo fabricados por RobotGroup debemos conectar la alimentación a la placa. Luego hay que encenderla y conectarla a la computadora utilizando un cable USB. Una vez que conectadas, la computadora nos mostrará la siguiente pantalla. En esta primera pantalla seleccionamos la opción "No por el momento" y presionamos "Siguiente", como se muestra en la figura:

Luego, en la pantalla siguiente, seleccionaremos la opción "Instalar desde una lista o ubicación específica (avanzado)" y presionamos el botón "Siguiente".



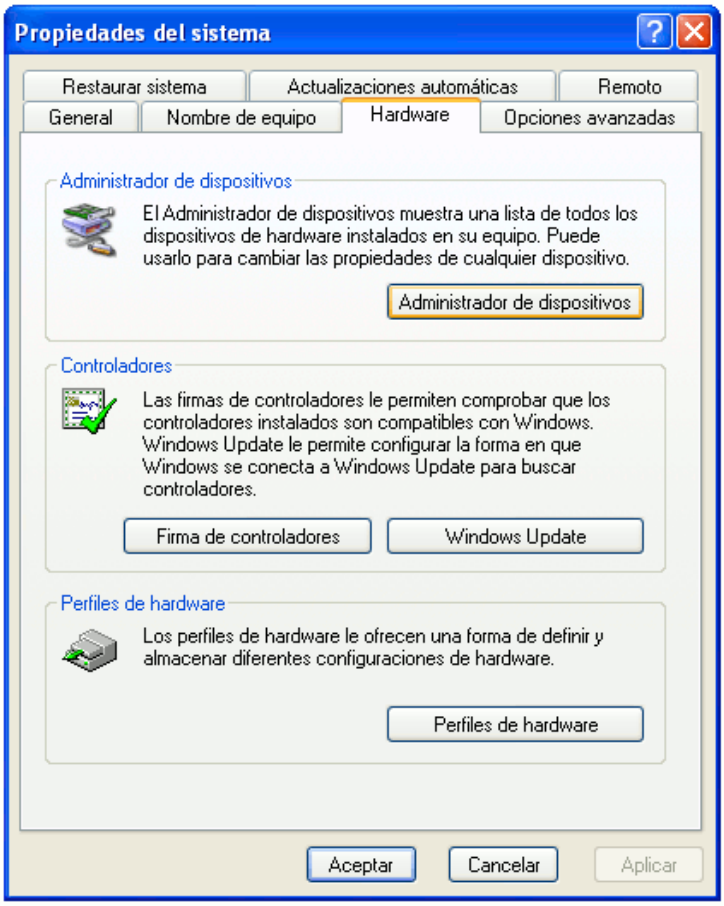
En este punto, tenemos que indicar la ubicación de la carpeta que contiene los drivers de comunicación. Dentro de la carpeta descomprimida de Minibloq, encontraremos la carpeta con los drivers en: "..\Minibloq.v0.8.Beta\Components\Drivers\DuinoBot\v1.x". Ya que lo que está pidiendo el asistente para instalación es la ruta (path) a la carpeta que contiene dichos drivers. El nombre de la carpeta que contiene los drivers puede diferir del mostrado en este caso según la versión que se adquirió. En el caso de poseer la versión 0.8.Beta, la carpeta que contendrá los drivers de comunicación de la placa se encontrará en "..\Minibloq.v0.8.Beta\Components\Drivers\DuinoBot\v1.x". Los dos puntos seguidos que se muestran en la ruta corresponden a la parte de la misma compuesta por las carpetas propias de cada usuario.

Una vez instalado el entorno y los drivers, veremos la siguiente pantalla en la que presionamos el botón "Finalizar" tal como se muestra en la siguiente figura:

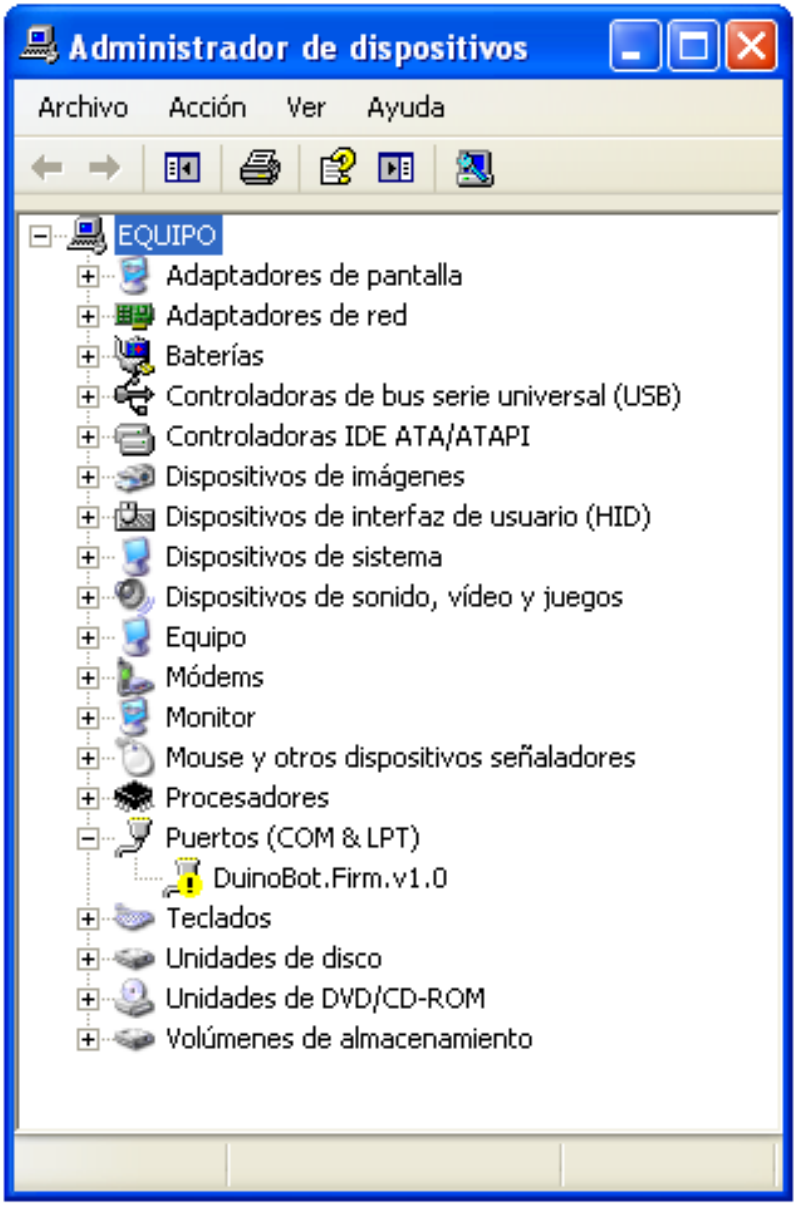
De esta manera, finalizamos la instalación de los drivers de comunicación de Minibloq en una computadora con Windows XP.

¿Problemas? Instalación manual de drivers en Windows XP utilizando Minibloq

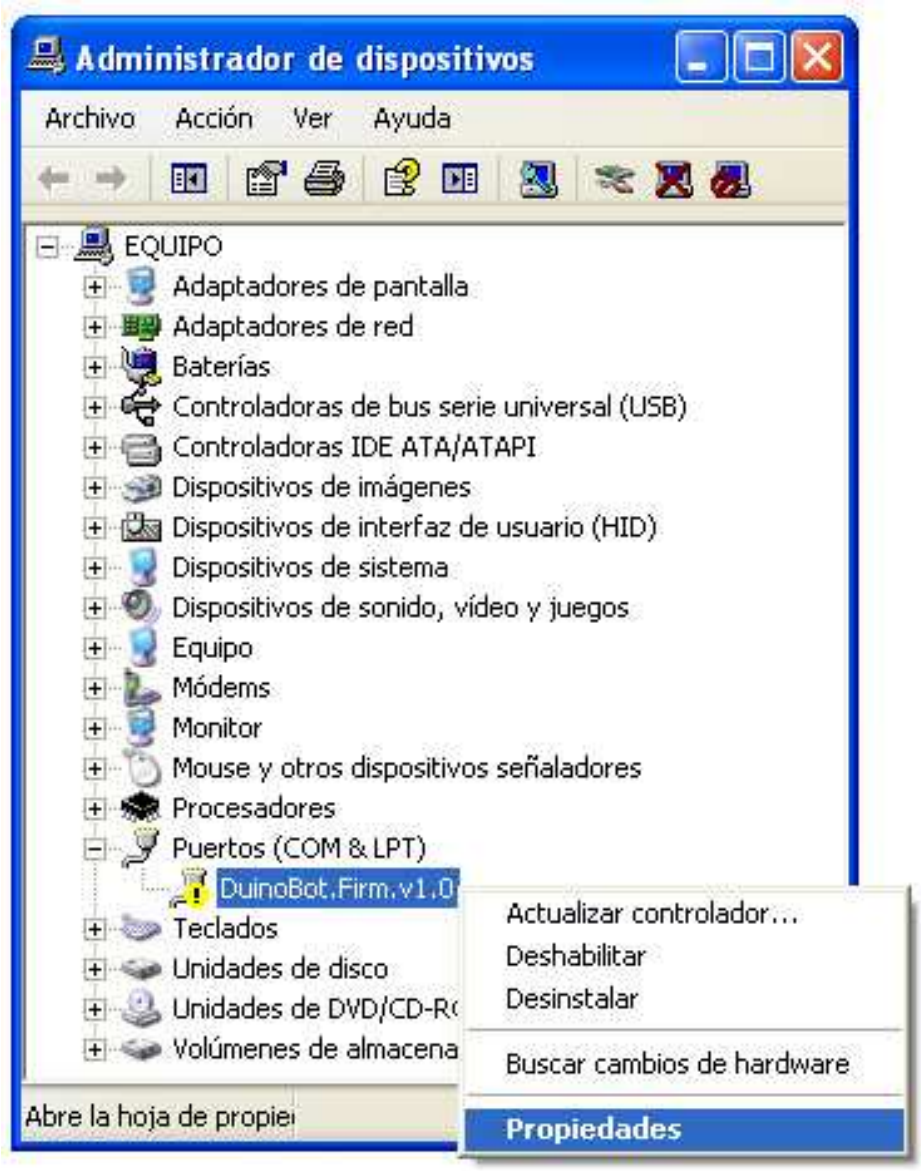
En caso de no haber podido instalar los drivers de comunicación, podemos reintentarlo de la forma descripta a continuación. Primero, hay que presionar el botón derecho del mouse sobre el ícono de "Mi PC". En ese momento, se desplegará un pequeño menú del cual seleccionaremos la opción "Propiedades". Entonces se abrirá la ventana "Propiedades del sistema", en la que tendremos que seleccionar la solapa "Hardware", tal como se muestra en la siguiente figura:



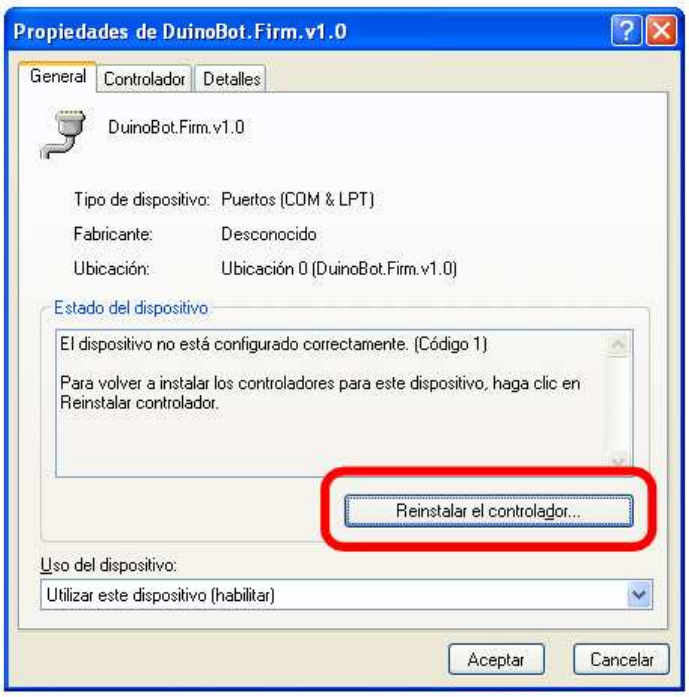
Dentro de la solapa "Hardware", seleccionamos la opción "Administrador de dispositivos". Veremos la siguiente pantalla:



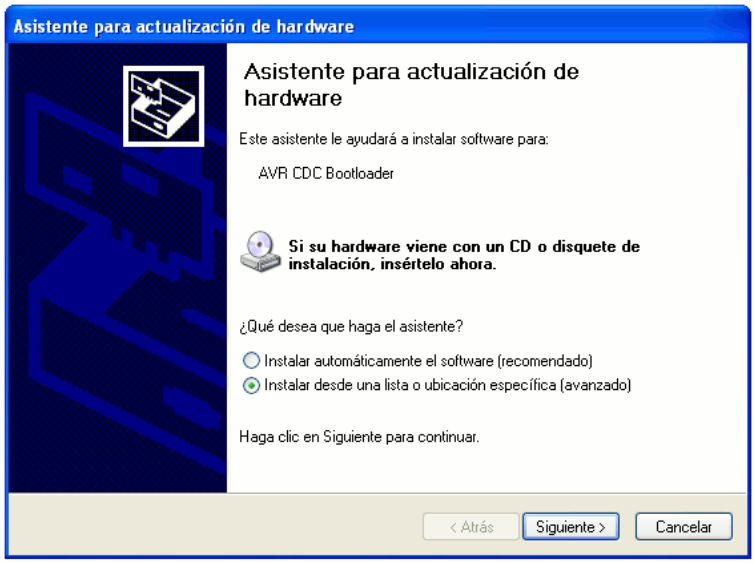
Como lo que nos interesa en este punto es comunicar a través de un puerto USB la placa con la computadora, seleccionamos, dentro "Puertos (COM & LPT)" la opción que corresponde al puerto de comunicaciones de la placa. Una vez seleccionada esta opción, apretamos el botón derecho del mouse y seleccionamos "Propiedades", tal como se muestra en la siguiente figura:



Al seleccionar esta opción, veremos las características del puerto que nos interesa. En este punto, la computadora nos da la opción de volver a instalar el controlador, para lo cual, tendremos que hacer clic en el botón "Reinstalar el controlador...", tal como se muestra en la siguiente figura:



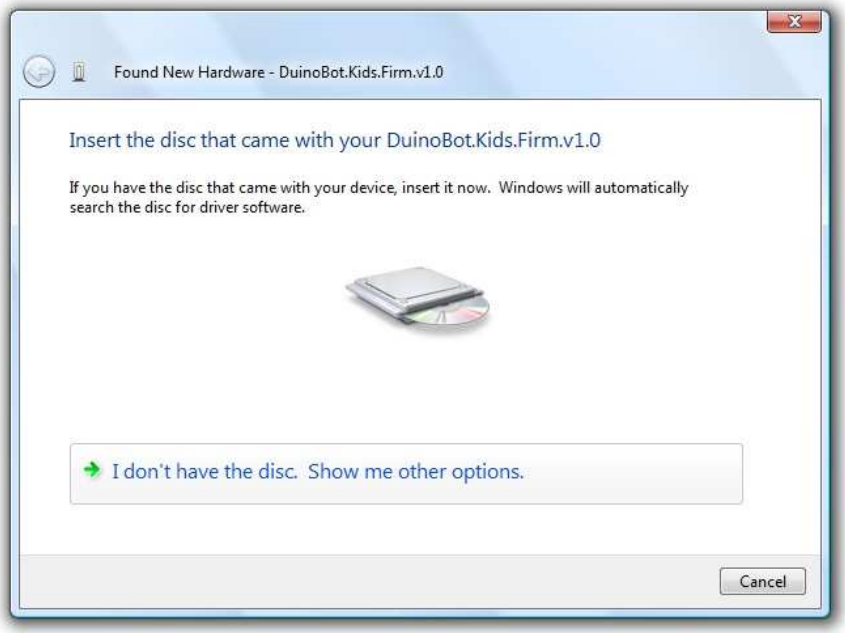
Luego, podremos visualizar el asistente para la instalación de hardware y seguir los pasos descriptos anteriormente acerca de cómo instalar los drivers de comunicación necesarios para utilizar la placa.



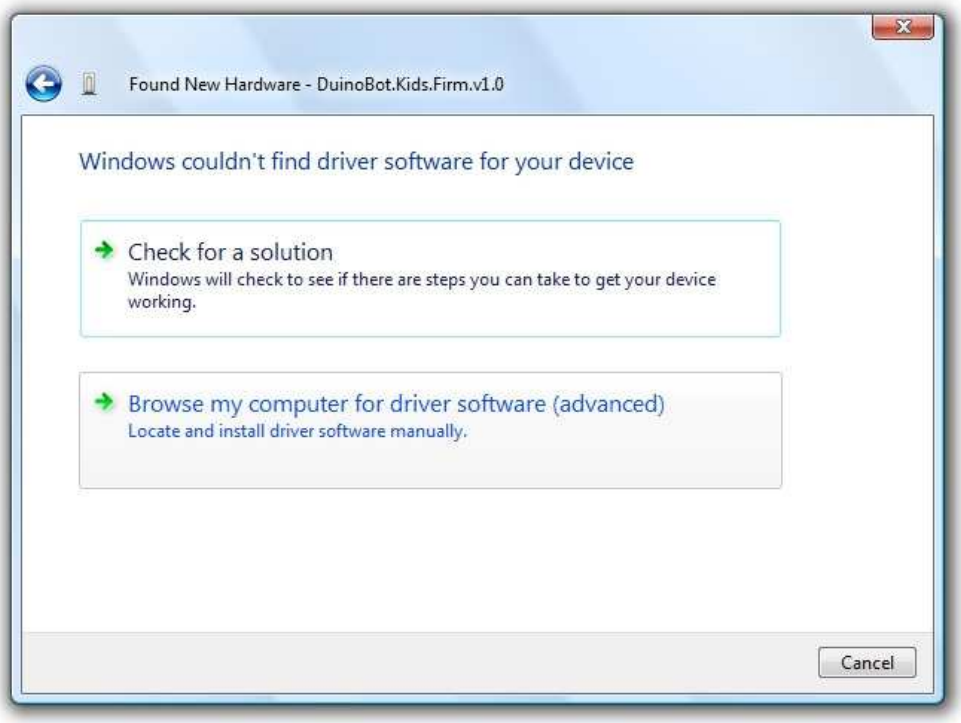
Instalación de drivers utilizando Minibloq en Windows Vista

Al conectar la placa a una pc con Windows Vista, veremos el siguiente mensaje en la pantalla:

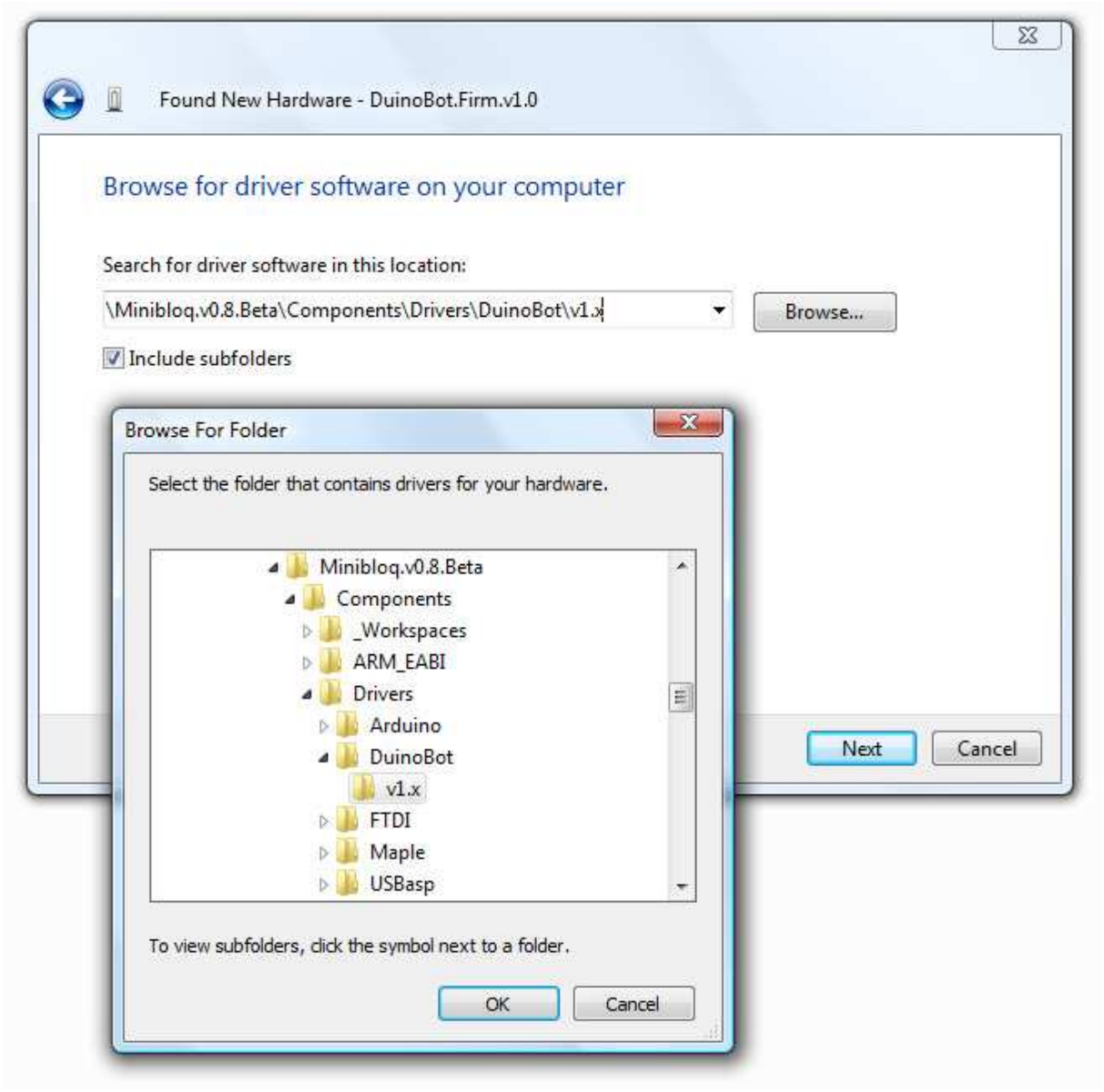
En este caso, seleccionamos la opción "Buscar e instalar el software de controlador (recomendado)". Así podremos elegir la carpeta que contiene los drivers que queremos instalar. Inmediatamente después de seleccionar dicha opción, Windows Vista nos pedirá permiso para continuar con la instalación del nuevo software. Hacemos clic en "Continuar". Luego, la computadora tratará de encontrar los drivers por su cuenta para instalarlos por lo que, después de unos momentos, nos mostrará la siguiente imagen:



Seleccionamos la opción "No tengo el disco. Mostrarme otras opciones.", lo que nos llevará a la siguiente pantalla, en la cual seleccionamos la opción "Buscar software de controlador en el equipo (avanzado)".

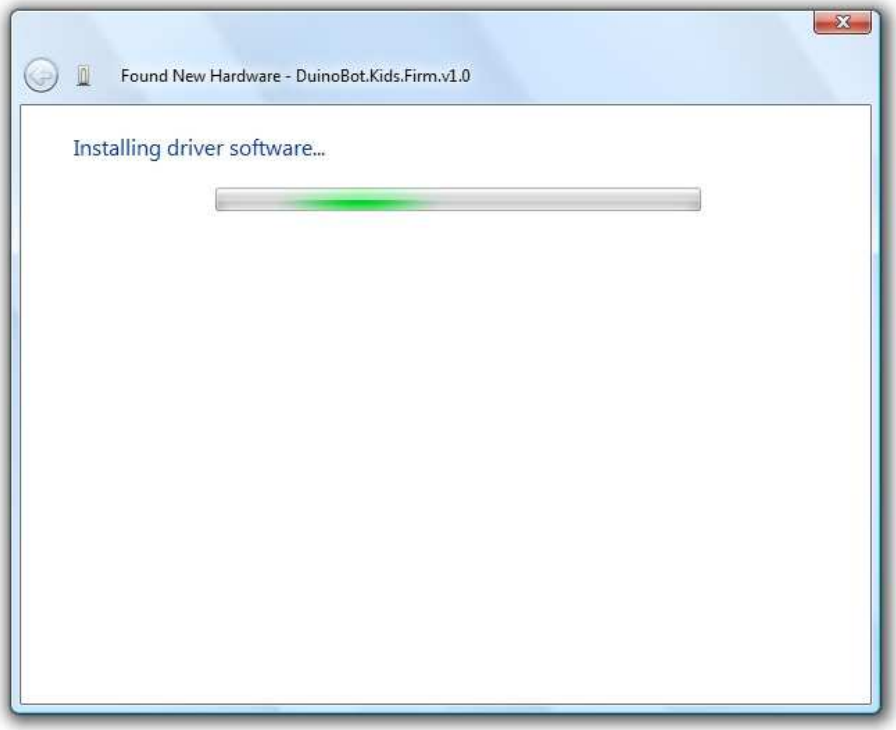


A continuación aparecerá un texto en el que hay que ingresar la ubicación de la carpeta que contiene los drivers de instalación, los cuales se encuentran en "..\Minibloq.v0.8.Beta\Components\Drivers\DuinoBot\v1.x". Una vez seleccionada dicha carpeta, hacemos clic en "Siguiente".

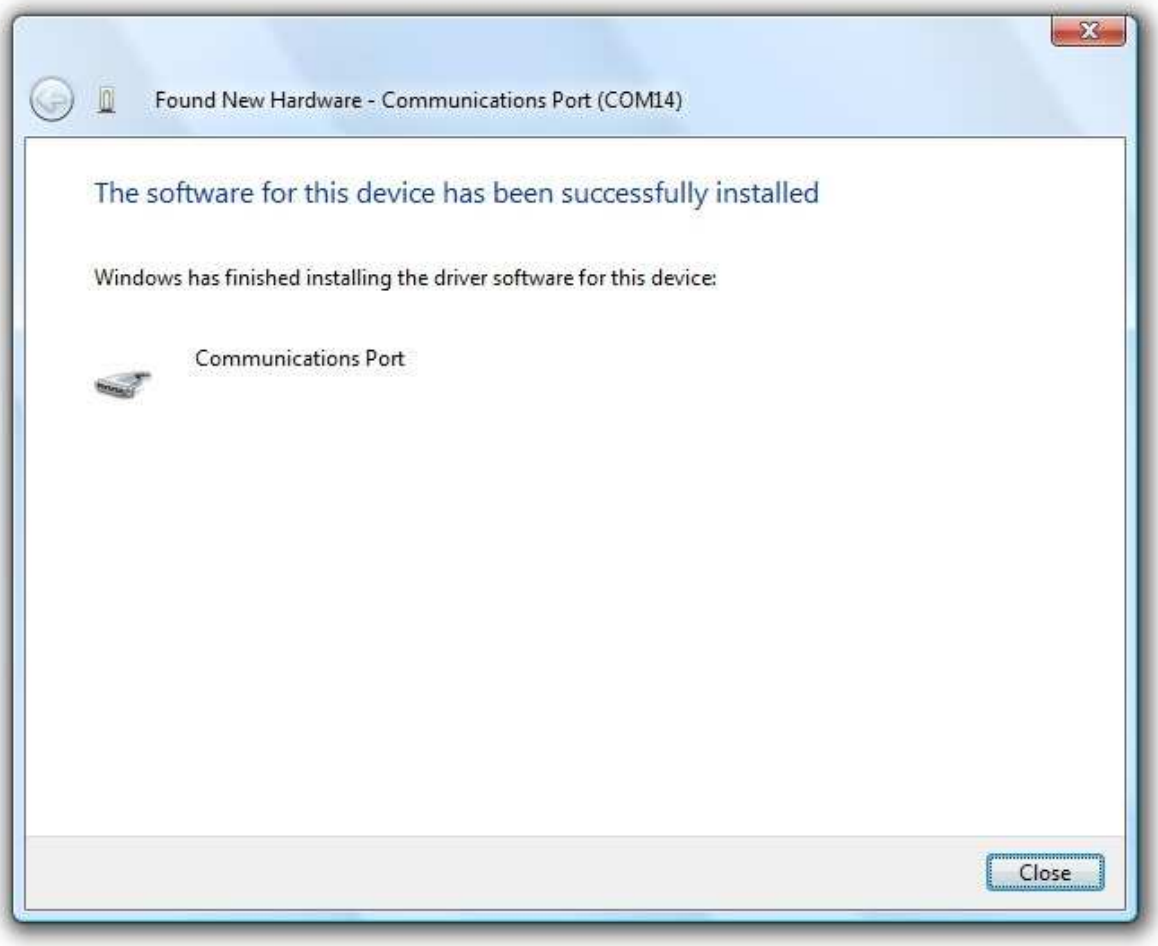


Como se muestra a continuación, Windows Vista nos preguntará si queremos continuar con la instalación, la opción que debemos seleccionar es "Instalar este software de controlador de todas formas" ya que es la que nos permitirá seguir adelante.

A partir de este punto, se realizará la instalación hasta que veamos la siguiente pantalla que indica el final de la misma:

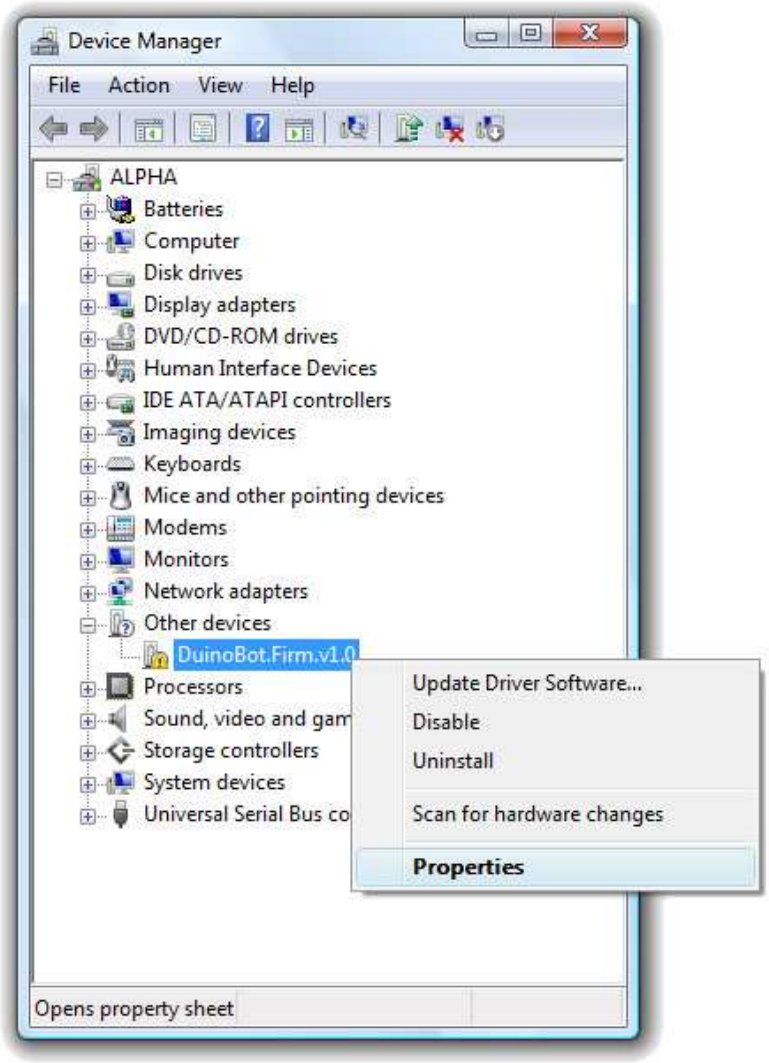


Presionamos el botón "Cerrar" a la derecha abajo de la pantalla y así terminamos con todo el proceso de instalación de drivers de comunicación necesarios para utilizar la placa DuinoBot en una computadora con Windows Vista.

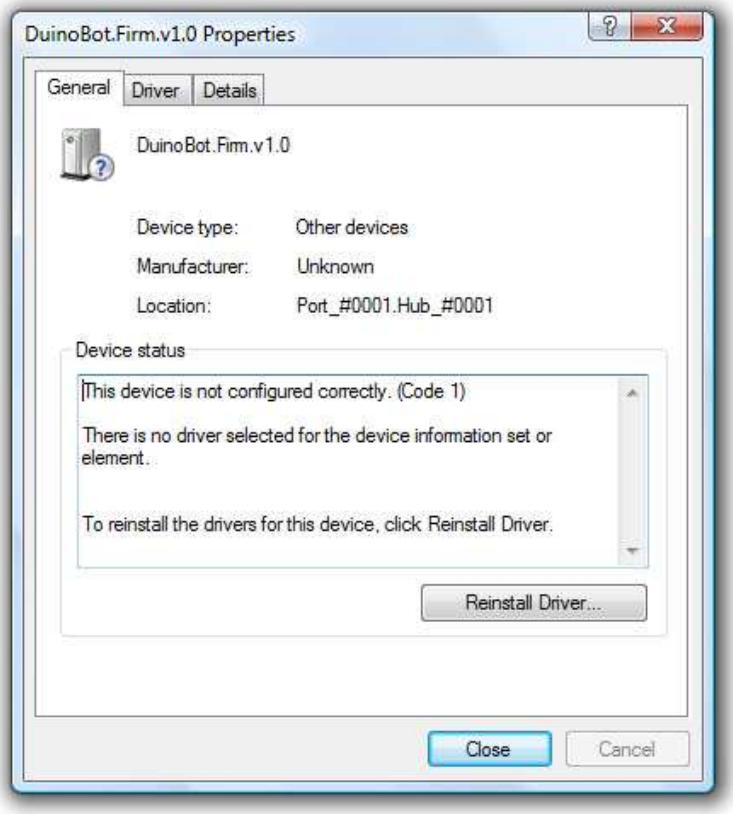


¿Problemas? Instalación manual de drivers en Windows Vista utilizando Minibloq

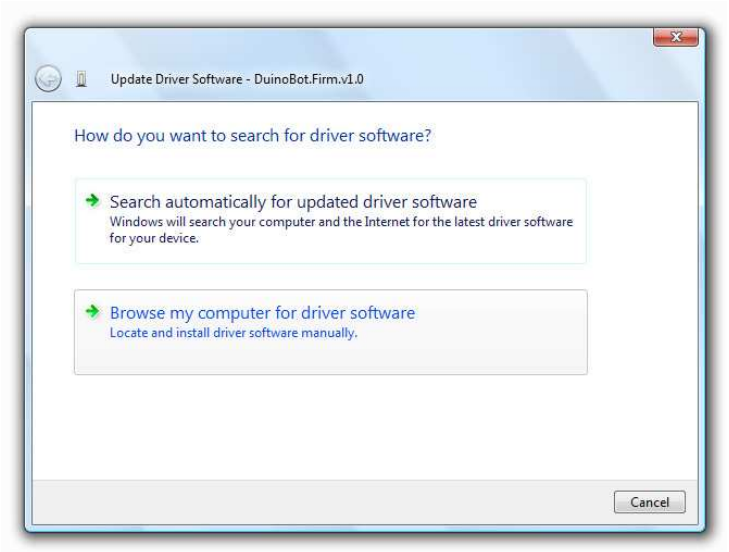
En caso de no haber podido instalar los drivers, podemos reintentarlo de la forma descripta a continuación. Primero, hay que presionar el botón derecho del mouse sobre el ícono de "Equipo" o "Computer". En ese momento, se desplegará un pequeño menú del cual seleccionaremos la opción "Propiedades" o "Properties". Entonces se abrirá la ventana en la que tendremos que seleccionar la opción "Propiedades del sistema" o "Device Manager". Una vez dentro del "Administrador de dispositivos" ("Device Manager" en la versión en inglés) seleccionamos la opción que corresponde a nuestro controlador. En este caso, hacemos clic con el botón derecho del mouse sobre "DuinoBot.Firm.v1.0" y seleccionamos la opción "Properties", como se muestra en la siguiente figura:



Dentro de las propiedades tendremos la opción de "Reinstalar Drivers" ("Reinstall Driver…" en la versión en inglés).



Al seleccionar "Reinstall Driver…", veremos la pantalla que se muestra a continuación en la que seleccionaremos la opción "Browse my computer for driver software" ("buscar en mi computadora el softawe"). Luego podremos realizar el proceso completo de instalación tal como se lo describió en la sección "Instalación de drivers en Windows Vista".



Instalación de Minibloq

Antes de empezar a trabajar con la placa vamos a tener que copiar el entorno de programación Minibloq a nuestra computadora. Para ello, utilizamos el cd que viene incluido con la placa o descargamos el programa desde internet. Luego descomprimimos la carpeta en un directorio local de la computadora que vamos a utilizar.

Ejecución de Minibloq

Dentro de la carpeta que acabamos de grabar en un nuestra computadora tenemos que ejecutar el archivo MinibloqRun.exe. Este abrirá el entorno de programación Minibloq.

Ejemplo de prueba en Minibloq

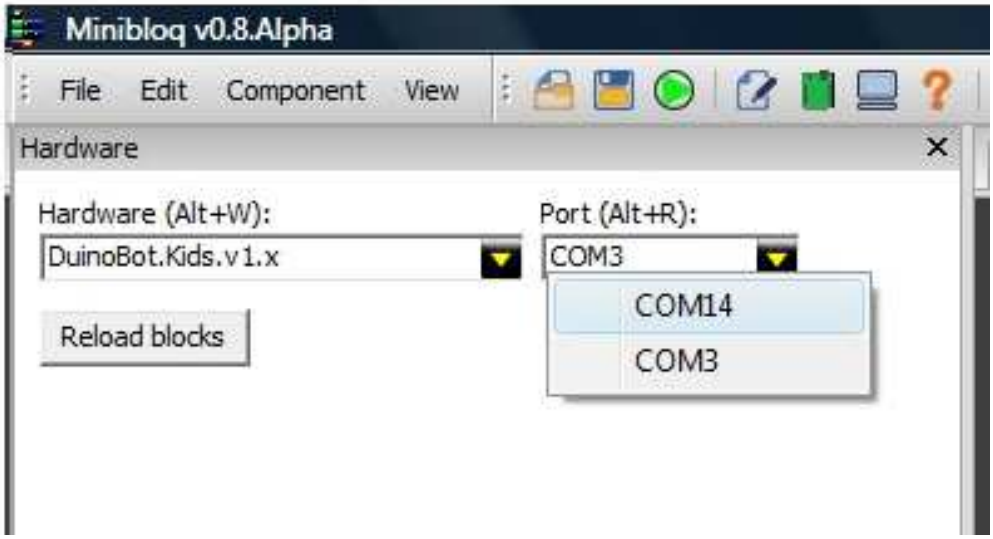
Para probar el entorno Minibloq, vamos a realizar un pequeño programa que haga que la placa reproduzca una melodía sencilla. Para eso, seleccionamos los bloques que se muestran en la siguiente imagen:

Explicación del programa de prueba

Este programa es una melodía compuesta por varios bloques similares. Cada bloque ejecutará una nota durante un tiempo determinado.

Selección del puerto

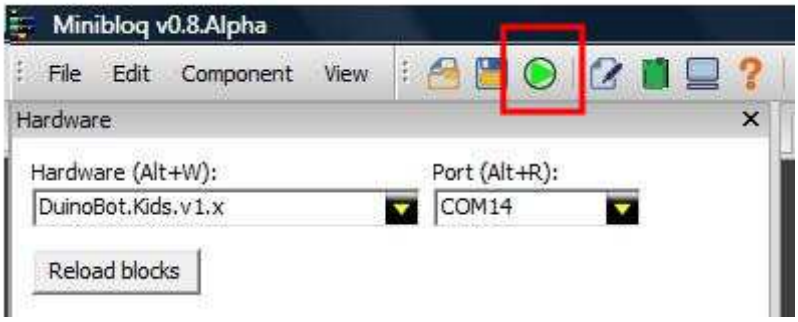
Antes de descargar el programa realizado a la placa, hay que conectar el cable USB a la placa y a la computadora. Luego, hay que prender la placa. Una vez conectada la placa a la computadora hay que seleccionar el puerto adecuado. Para eso se hace "clic" en la sección "Port (Alt+R)", como se muestra en la siguiente figura.



Luego se selecciona el puerto correspondiente. En este caso es el COM14.

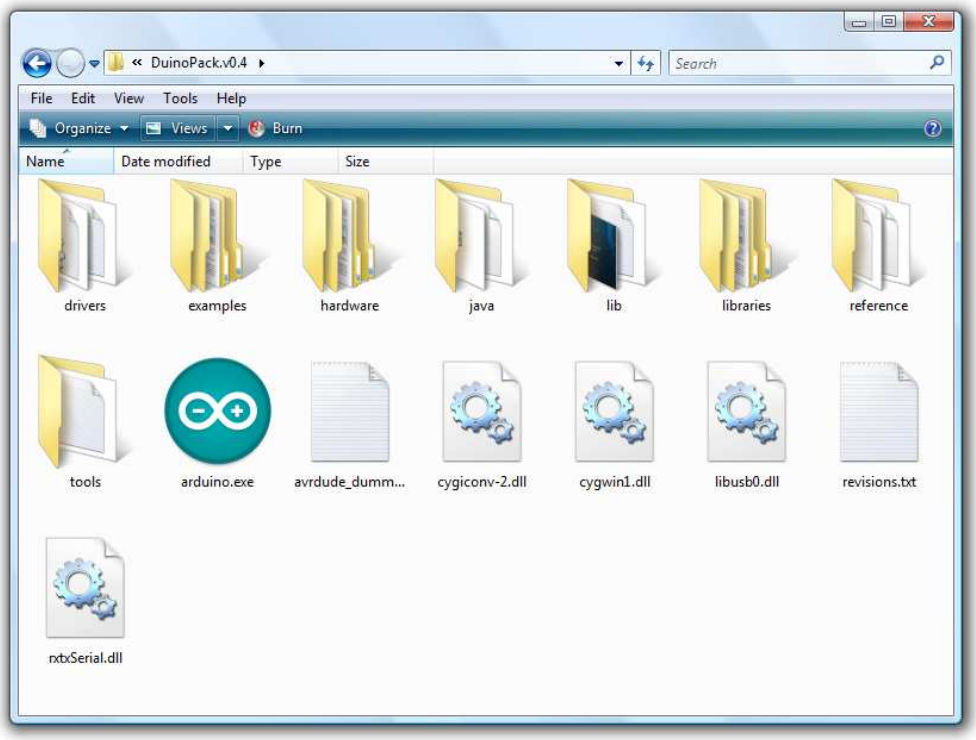
Descarga del programa a la placa

Con la placa conectada a la computadora y puerto correctamente seleccionado, se presiona el botón "Run", el cual hará que el entorno verifique la ausencia de errores y luego descargue el programa desde la computadora a la placa controladora. El botón utilizado es el que se muestra en la siguiente figura:

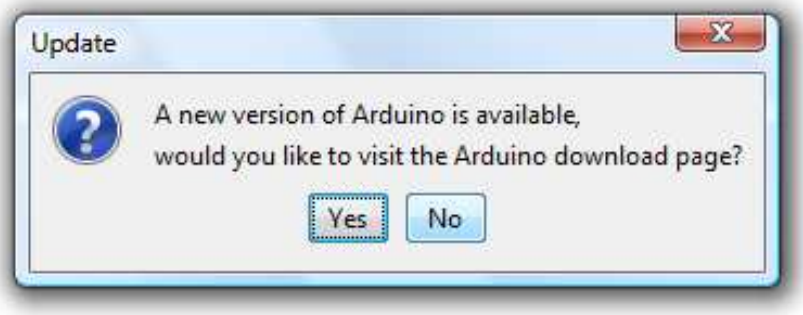


Instalación del entorno de programación Arduino

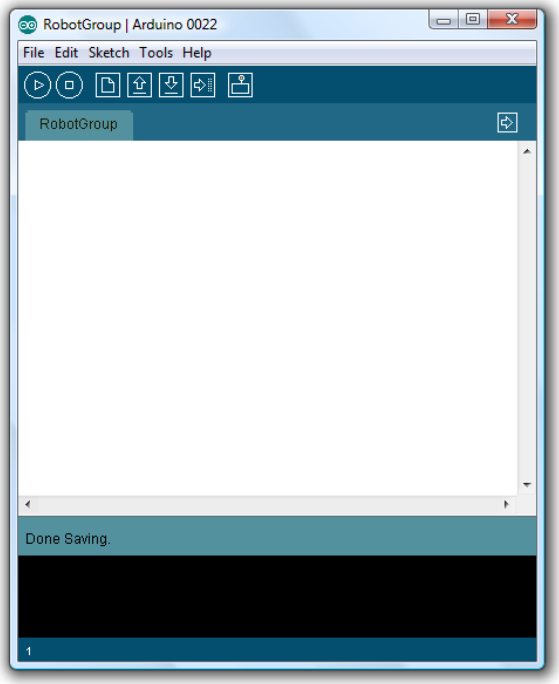
Antes de empezar a trabajar con la placa vamos a tener que copiar el entorno de programación que usaremos en nuestra computadora. Para ello, descargamos el entorno de programación DuinoPack, que es una versión del Arduino IDE pero con todas las librerías y agregados necesarios para trabajar con el hardware Multiplo sin tener que instalar nada adicional. La última versión de DuinoPack se puede siempre descargar del área de Software del sitio de RobotGroup (http://robotgroup.com.ar) Una vez vea que lo tenemos, debemos descomprimirlo en una carpeta local. Luego, lo abrimos haciendo doble click en arduino.exe ya que no requiere instalación.



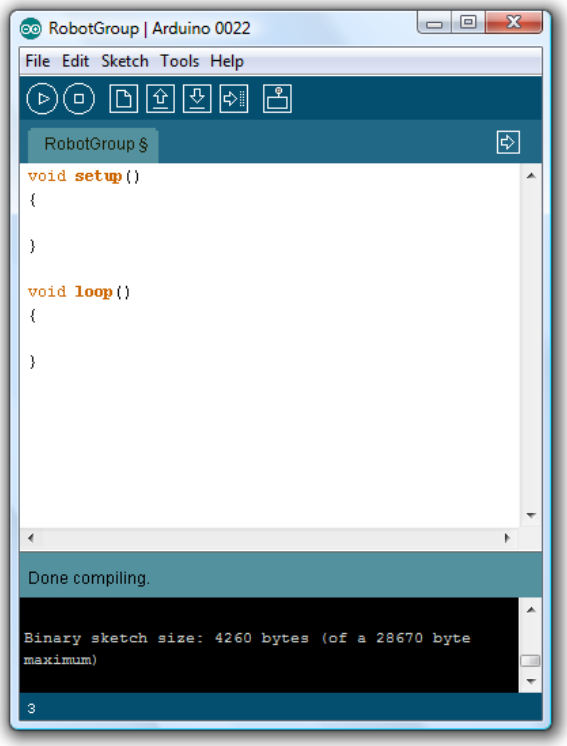
Lo primero que aparecerá al abrir el programa es un mensaje que nos indica que hay una versión más nueva de Arduino para descargar. La opción que hay que seleccionar es "No" ya que la versión que vamos a usar del entorno de programación tiene librerías específicas que sirven para utilizar los robots Multiplo fabricados por RobotGroup. El mensaje que veremos es el que se muestra en la siguiente imagen:



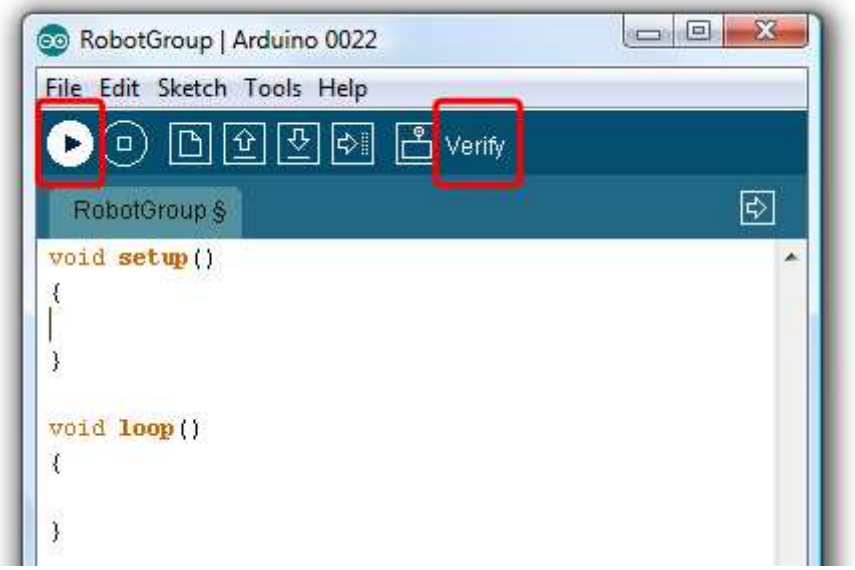
Una vez abierto el entorno de programación Arduino 0022 veremos una pantalla similar a la que se muestra a continuación. La única diferencia es que el nombre no será RobotGroup, sino que será uno que indicará la fecha de creación del archivo.



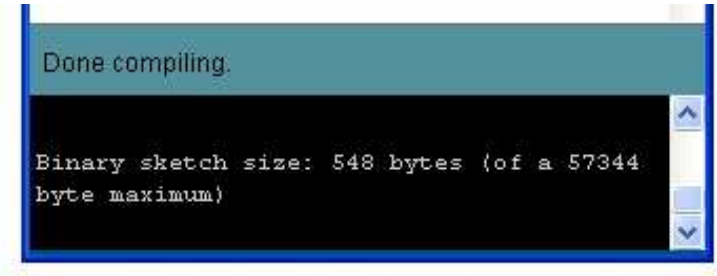
Lo primero que haremos en este punto será escribir la declaración de dos funciones necesarias para poder empezar a escribir nuestro código. Con lo cual, el entorno se verá como muestra la siguiente figura:



Estas dos funciones se llaman setup() y loop(). La primera prueba que podemos hacer para asegurarnos de que todo está funcionando correctamente, es compilar este sencillo programa. Para eso, apretamos el botón "Verify" que se encuentra a la izquierda arriba en la pantalla. Notemos que, al pararnos sobre el botón "Verify", el nombre del mismo aparecerá a la derecha de la pantalla. Lo mismo sucederá con el resto de los botones.



En caso de que hayamos escrito todo correctamente, en la parte inferior de la pantalla nos aparecerá un mensaje en color blanco como el que se muestra a continuación:

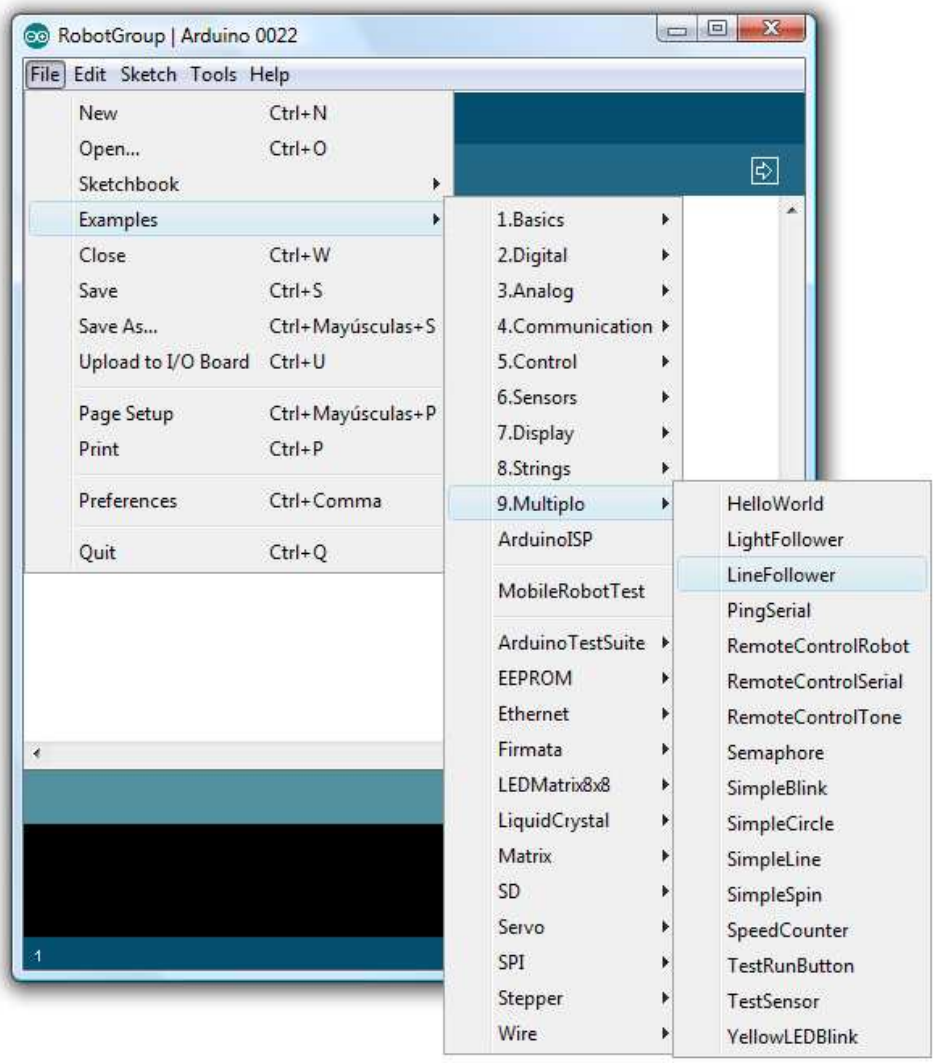


Por un lado, en letras negras y sobre fondo verde, podemos leer que el compilador dice: "Done compiling". Esto significa que terminó de compilar. Por otro lado, en letras blancas y fondo negro nos indica que el tamaño del archivo generado que, en este caso, fue de 548 bytes. En caso de que hayamos escrito algo mal, el entorno nos comunicará el error utilizando esta parte de la pantalla. Por ejemplo, es muy común que cuando una persona empieza a programar, se olvide de cerrar una llave o confunda llaves con paréntesis. También es muy común confundirse y escribir mal una orden o nombre de función. El lenguaje C/C++ es "Case sensitive", lo que significa que hace diferencia entre letras mayúsculas y minúsculas. Con lo cual, si alguien escribe, por ejemplo, Void loop(), esto será contado como un error por el compilador ya que no reconocerá la palabra Void por estar escrita con "V" mayúscula. A continuación se muestra cómo se vería el mensaje de error que nos devolvería el compilador en caso de haber escrito Void loop() en vez de void loop().

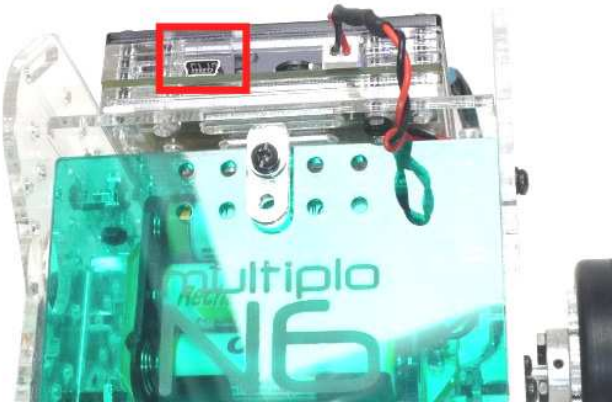
Como podemos ver en la figura anterior, el entorno avisa que encontró un error y lo describe, en este caso, nos dice que no reconoce la palabra "Void", ya que no fue escrita correctamente. Estas pruebas sencillas que acabamos de describir son para empezar a entender cómo será la programación de placa. Por lo tanto, no es necesario tenerla conectada a la computadora para realizar estas pruebas.

Conexión entre la placa DuinoBot y la PC

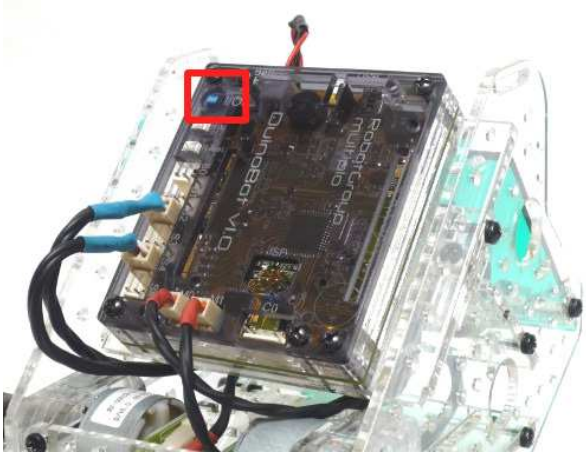
La primera prueba que podemos hacer para asegurarnos de que todo está funcionando correctamente, es abrir un programa de ejemplo que sirva para testear el funcionamiento de los motores. Para lo cual, vamos a seleccionar la opción "File", luego "Examples", "9.DuinoBot" y, por último, seleccionar el ejemplo "LineFollower", como se muestra en la siguiente figura:



Esto nos abrirá el ejemplo correspondiente. Luego, haremos que el entorno compile este archivo para asegurarnos que todo está bien. Para eso, apretamos el botón "Verify". Cuando ya estemos seguros de que nuestro entorno de programación Arduino 0022 funciona correctamente, conectamos el cable USB a la computadora y luego al robot. La conexión en el robot se realiza en la entrada para USB que tiene en la parte superior, tal como lo muestra la siguiente figura:



Una vez conectada la placa DuinoBot a la computadora y, teniendo el entorno instalado, se debe encender la placa desde su parte frontal. Tal como se muestra a continuación:



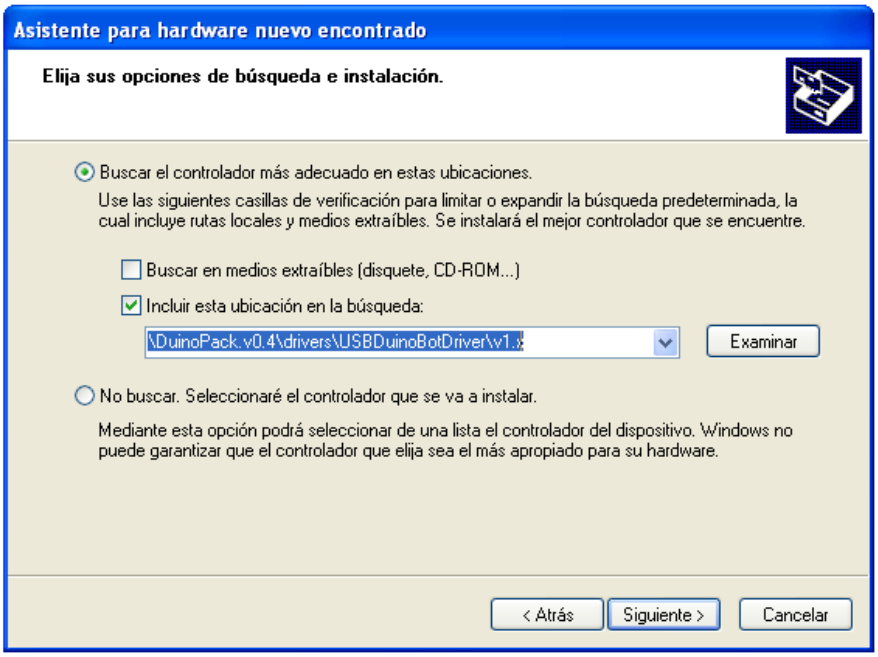
En cuanto hayamos terminado de encender la placa y, estando esta conectada, la computadora nos avisará que detectó nuevo hardware. Es muy importante detenerse en este punto para realizar correctamente la instalación de los drivers de comunicación de la placa DuinoBot. A continuación se muestra cómo realizarla en una computadora con Windows XP y luego en una computadora con Windows Vista.

Instalación de drivers de comunicación utilizando DuinoPack en Windows XP

En primer lugar, la computadora nos mostrará la siguiente pantalla:

En esta primera pantalla seleccionamos la opción "No por el momento" y presionamos "Siguiente", como se muestra en la figura. Luego, en la pantalla que sigue, seleccionaremos la opción "Instalar desde una lista o ubicación específica (avanzado)" y presionamos el botón "Siguiente", como se muestra en la próxima imagen:

En este punto, tenemos que indicar la dirección de la carpeta que contiene los drivers de comunicación. Esta se encuentra en: "..\DuinoPack.v0.4\drivers\USBDuinoBotDriver\v1.x". Ya que lo que está pidiendo el asistente para instalación es la ruta a la carpeta que contiene dichos drivers, debemos asegurarnos de seleccionar correctamente la carpeta cuyo nombre es "v1.x" que se encuentra dentro de "USBDuinoBotDriver".



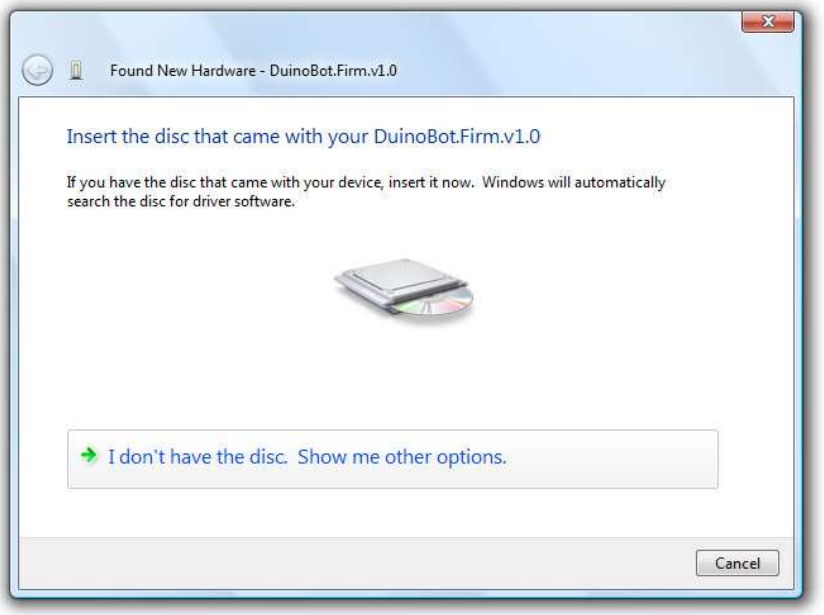
Una vez instalado el entorno y los drivers, presionamos el botón "Upload" tal como se muestra en la siguiente figura:

Al hacer esto, el entorno Arduino 0022 compilará el programa y lo grabará en la memoria del robot.

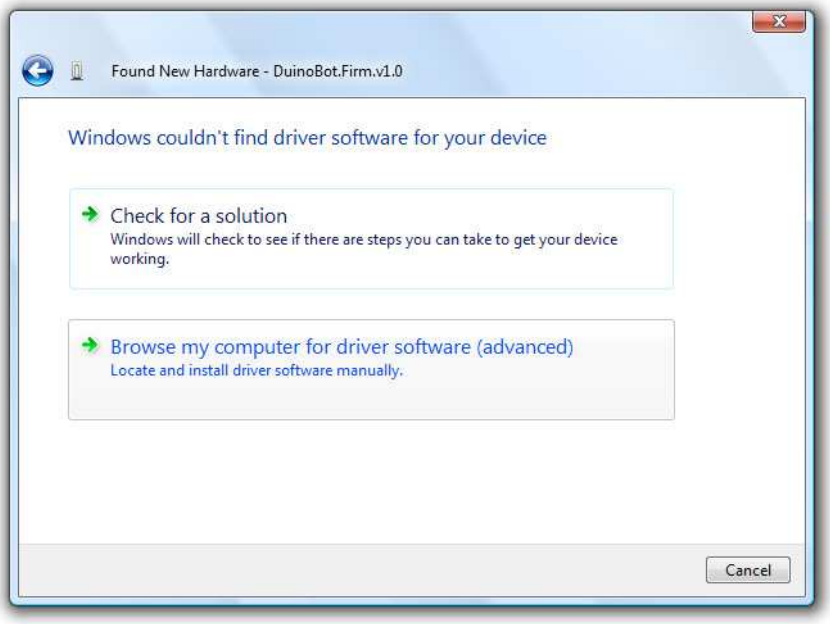
Instalación de drivers de comunicación utilizando DuinoPack en Windows Vista

Al conectar la placa a una pc con Windows Vista, veremos el siguiente mensaje por pantalla:

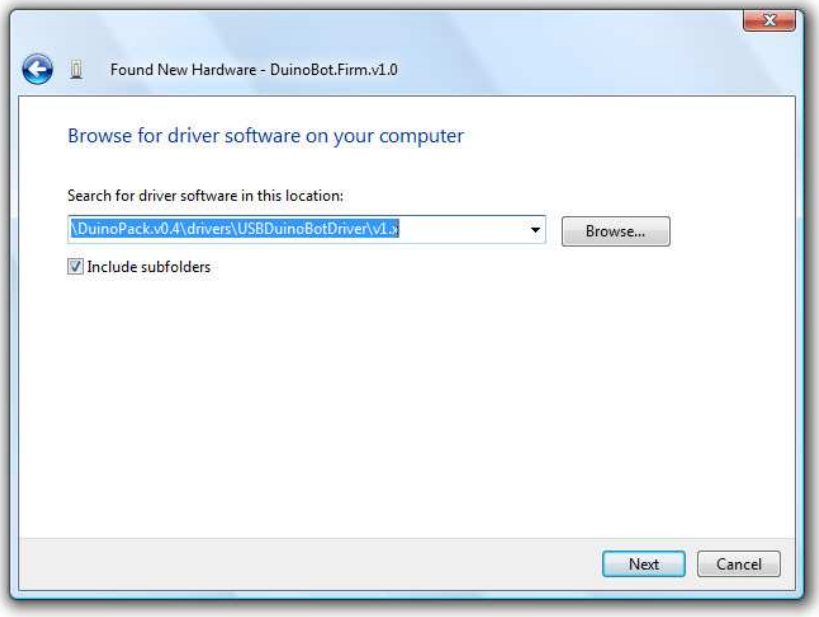
En este caso, seleccionamos la opción "Locate and install driver software (recommended)". La versión en español dirá "Buscar e instalar el software de controlador (recomendado)". Así podremos elegir la carpeta que contiene los drivers que queremos instalar. Inmediatamente después de seleccionar dicha opción, Windows Vista nos dirá "Inserte el disco incluído en AVR CDC Bootloader". En la versión en inglés, mostrará el mensaje "Insert the disc that came with your DuinoBot.Firm.v1.0", tal como lo muestra la siguiente figura:



Seleccionamos la opción "I don´t have the disc. Show me other options" (en la versión en español: "No tengo el disco. Mostrarme otras opciones."). Esto nos llevará a la siguiente pantalla, en la cual seleccionamos la opción "Browse my computer for driver software (advanced)" (en la versión en español: "Buscar software de controlador en el equipo (avanzado))".

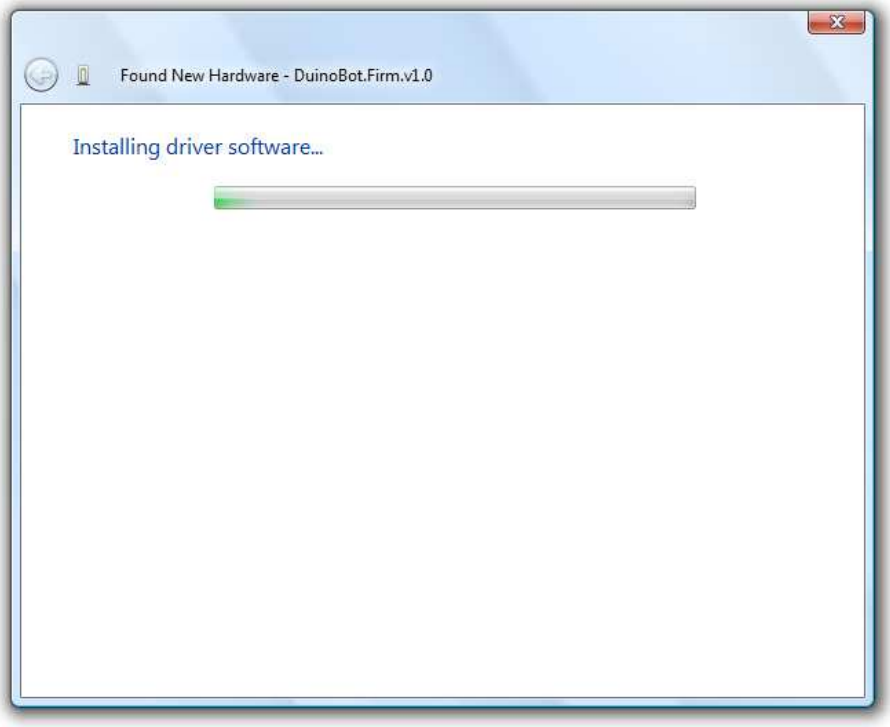


A continuación aparecerá un cuadro de texto en el que hay que ingresar la ubicación de la carpeta que contiene los drivers de instalación, los cuales se encuentran en "..\DuinoPack.v0.4\drivers\USBDuinoBotDriver\v1.x". Una vez seleccionada dicha carpeta, hacemos clic en "Next" ("Siguiente" en la versión en español).

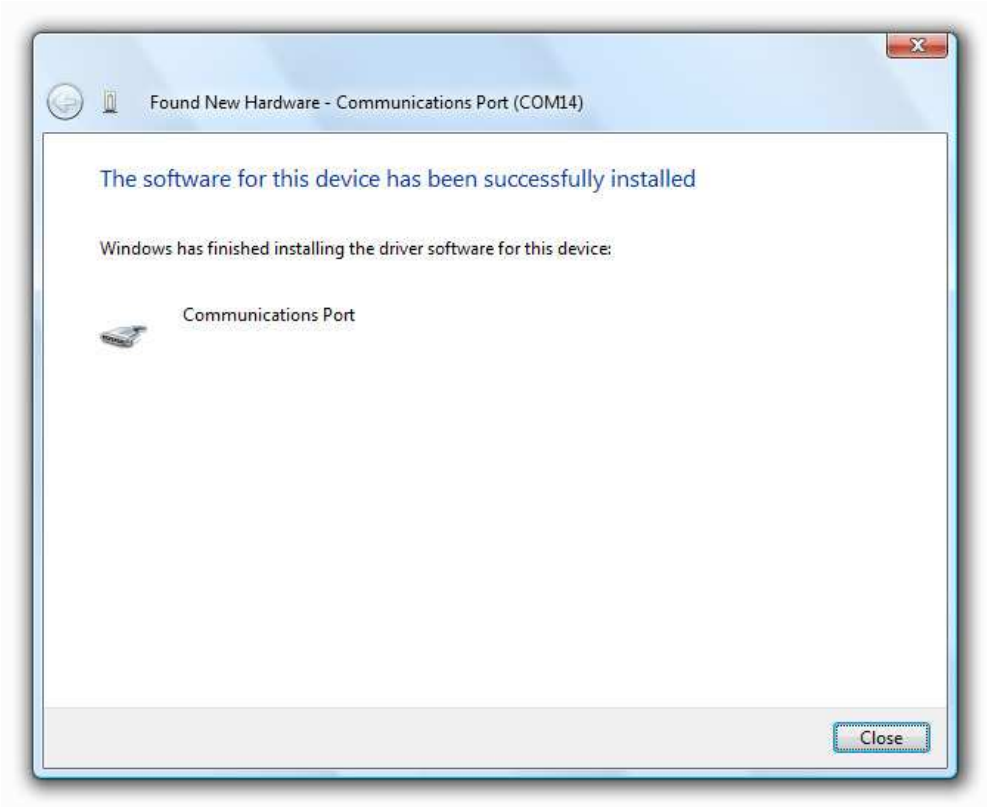


Como se muestra a continuación, Windows Vista nos preguntará si queremos continuar con la instalación, la opción que debemos seleccionar es "Install this driver software anyway" (en la versión en español "Instalar este software de controlador de todas formas") ya que es la que nos permitirá seguir adelante.

A partir de este punto, se realizará la instalación de los drivers. Mientras, veremos la pantalla que se muestra a continuación:

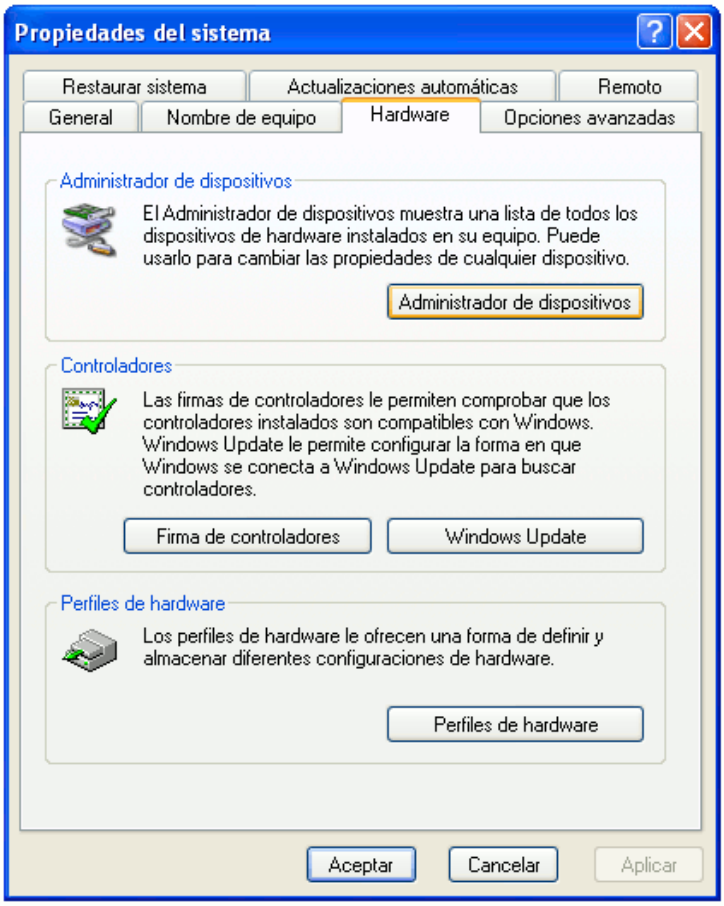


Una vez finalizado el proceso de instalación de los drivers veremos la siguiente pantalla en la que presionaremos el botón "Close" ("Cerrar" en la versión en español):

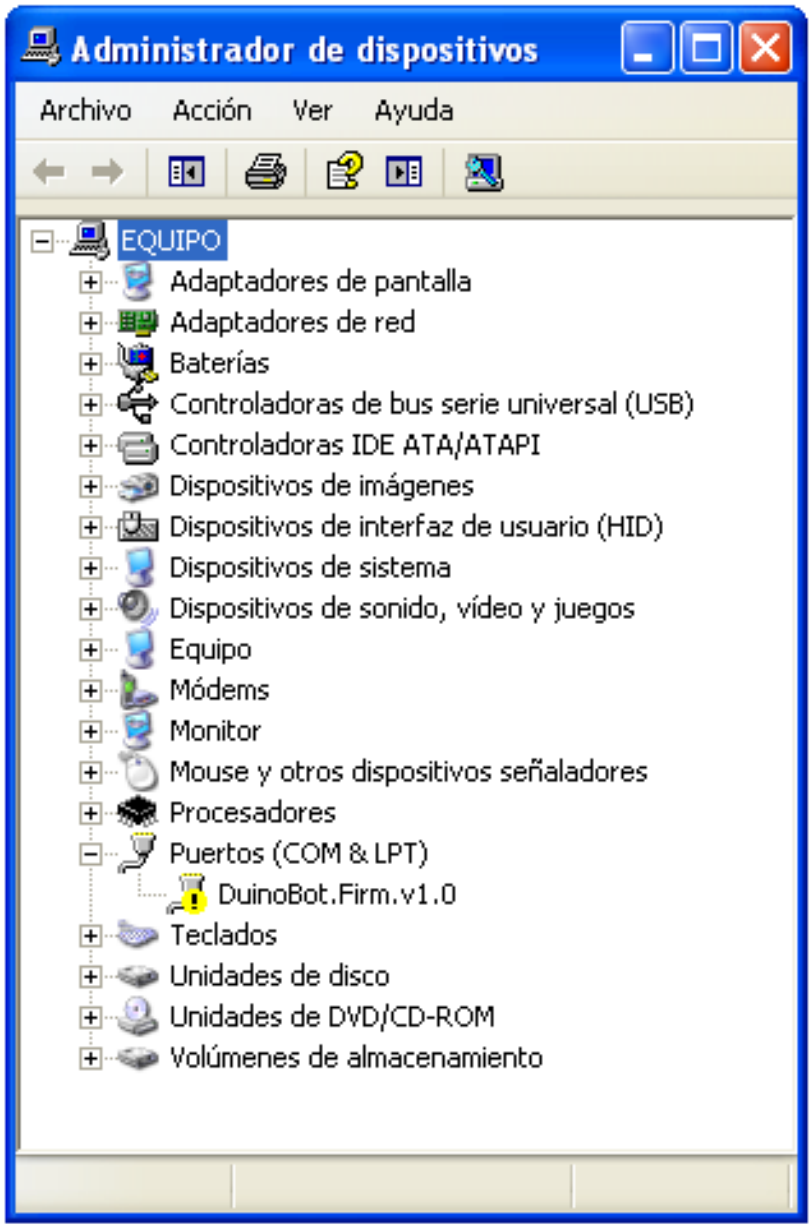


Instalación manual en Windows XP utilizando DuinoPack

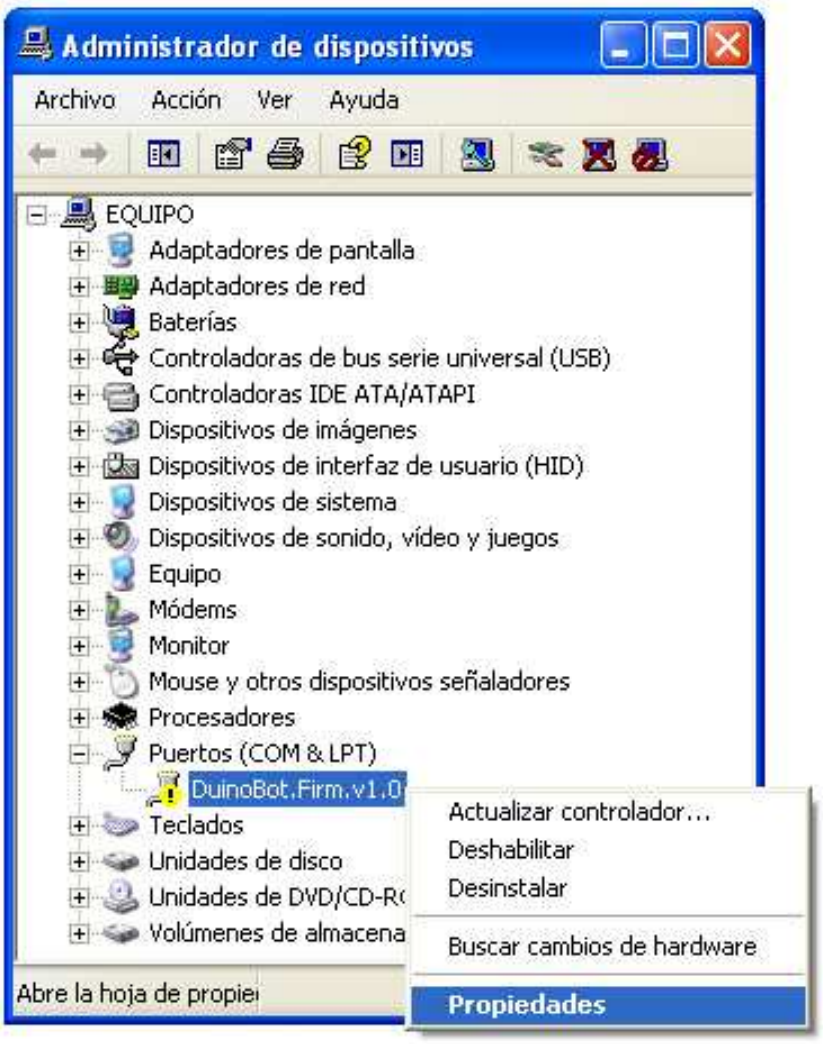
En caso de no haber podido instalar los drivers de comunicación, podemos reintentarlo de la forma descripta a continuación. Primero, hay que presionar el botón derecho del mouse sobre el ícono de "Mi PC". En ese momento, se desplegará un pequeño menú del cual seleccionaremos la opción "Propiedades". Entonces se abrirá la ventana "Propiedades del sistema", en la que tendremos que seleccionar la solapa "Hardware", tal como se muestra en la siguiente figura:



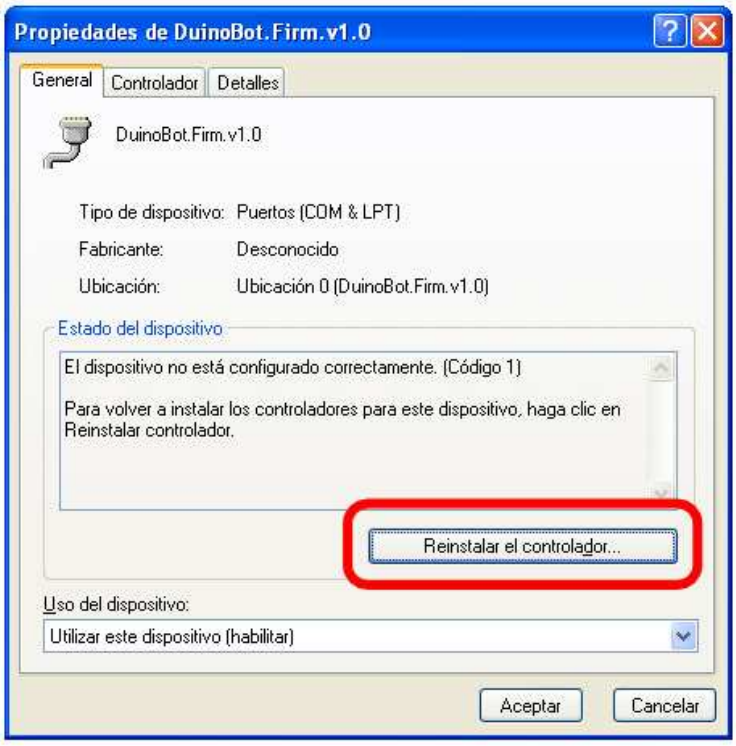
Dentro de la solapa "Hardware", seleccionamos la opción "Administrador de dispositivos" (en la versión en inglés se llama "Device Manager"). Veremos la siguiente pantalla:



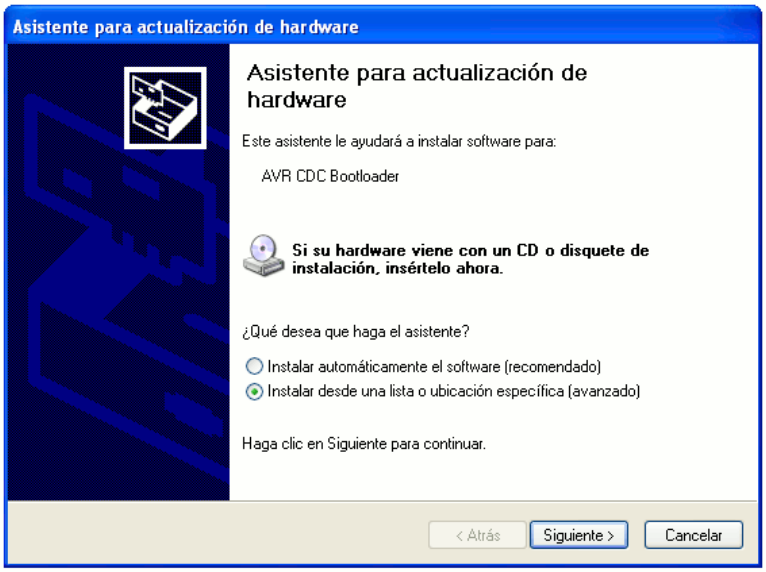
Como lo que nos interesa en este punto es comunicar a través de un puerto USB la placa DuinoBot con la computadora, seleccionamos, dentro de "Puertos (COM & LPT)" la opción "DuinoBot.Firm.v1.0". Esta opción es la que nos muestra que nuestra placa ha sido reconocida. Una vez seleccionada esta opción, apretamos el botón derecho del mouse y seleccionamos "Propiedades", tal como se muestra en la siguiente figura:



Al seleccionar esta opción, veremos las características del puerto que nos interesa. En este punto, la computadora nos da la opción de volver a instalar el controlador, para lo cual, tendremos que hacer click en el botón "Reinstalar el controlador...", tal como se muestra en la siguiente figura:

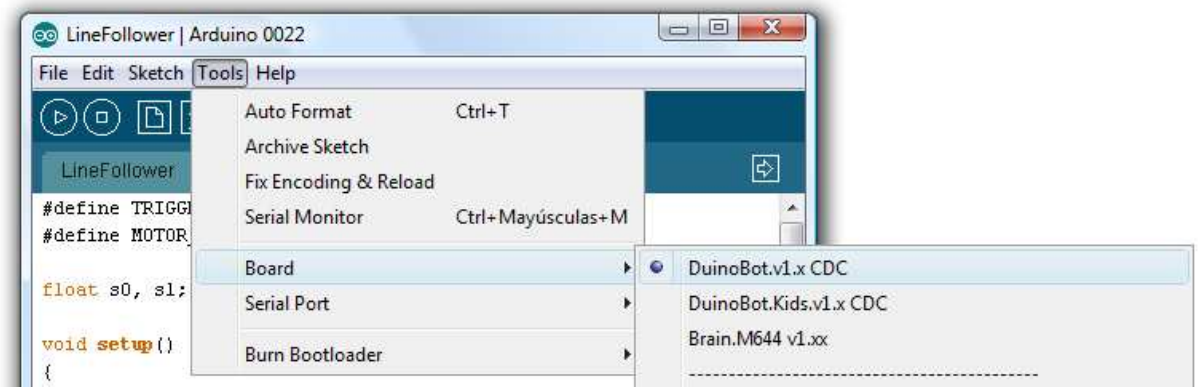


Luego, podremos visualizar el asistente para la instalación de hardware y seguir los pasos descriptos en la sección "Instalación de drivers de comunicación en Windows XP".

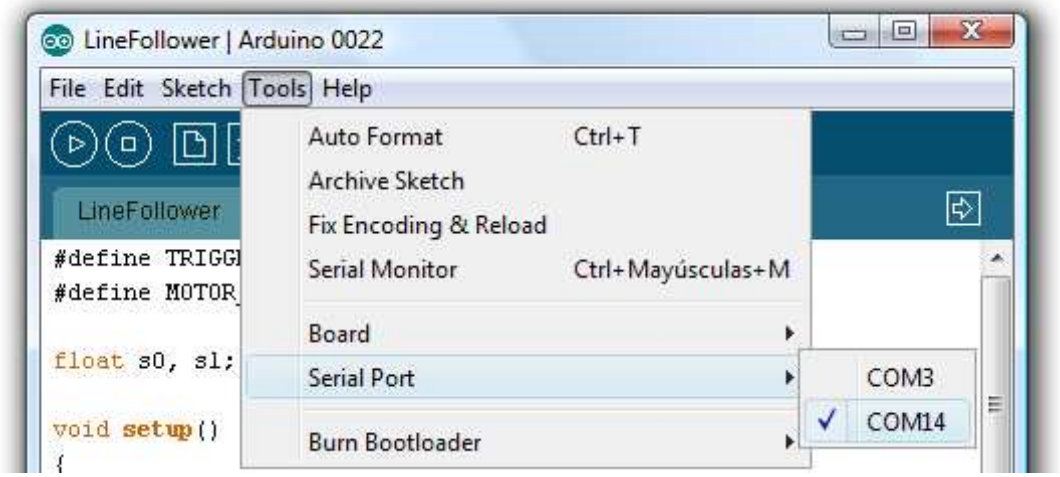


Posibles problemas y sus soluciones

Es probable que el robot no funcione la primera vez que quiera usarlo. A continuación se describen algunas cosas para tener en cuenta en caso que esto suceda. Lo primero que hay que chequear, es que efectivamente esté seleccionada la opción correspondiente en el menú "Tools", "Board". En este caso, hay que seleccionar la opción "DuinoBot.v1.x CDC", tal como se muestra en la siguiente figura:

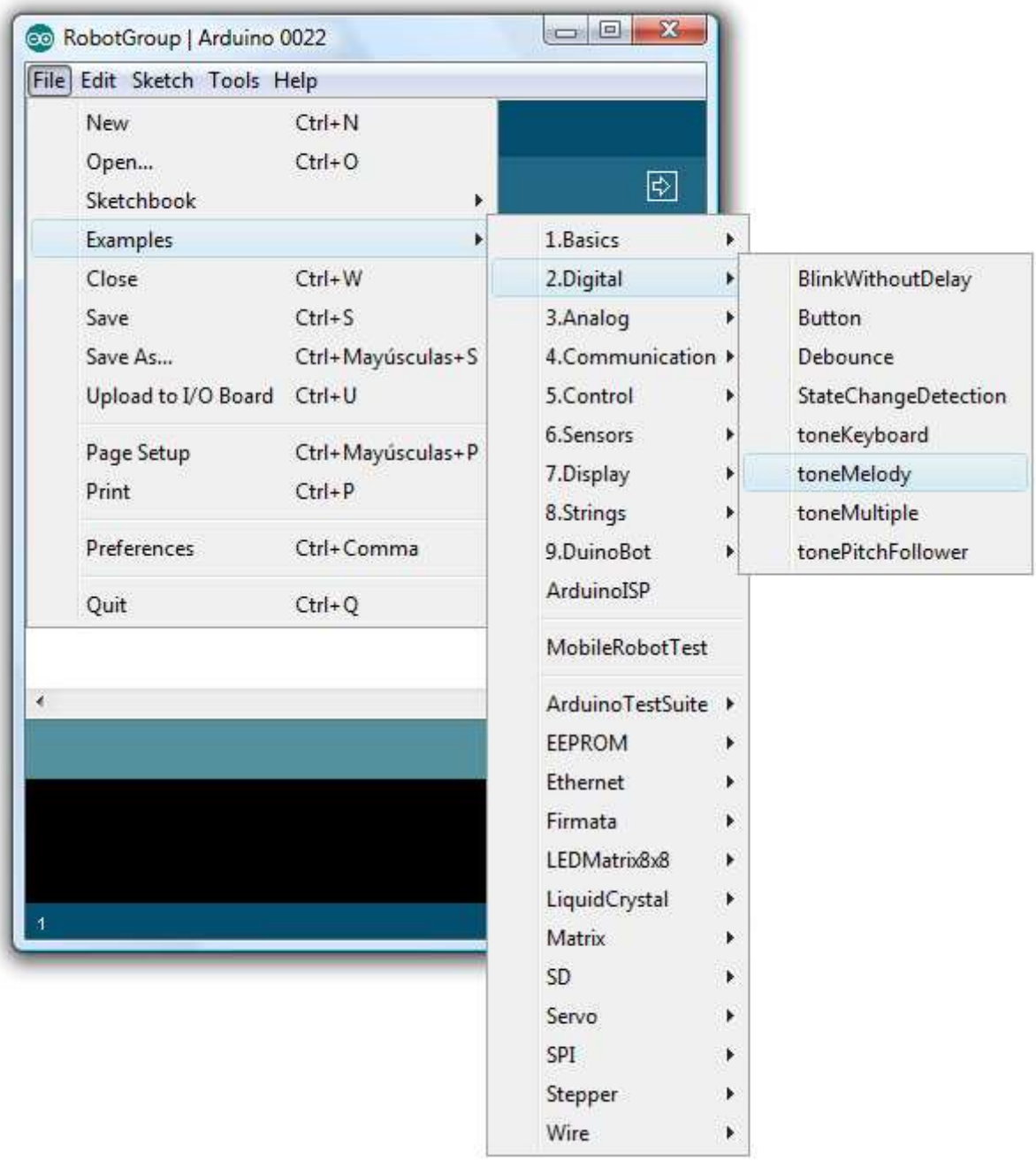


También hay que asegurarse de que el puerto de comunicación seleccionado sea el correcto. Esto hay que chequearlo en "Tools", "Serial Port". Hay que estar seguro de que el puerto COM correspondiente esté tildado, tal como se muestra en la siguiente figura:

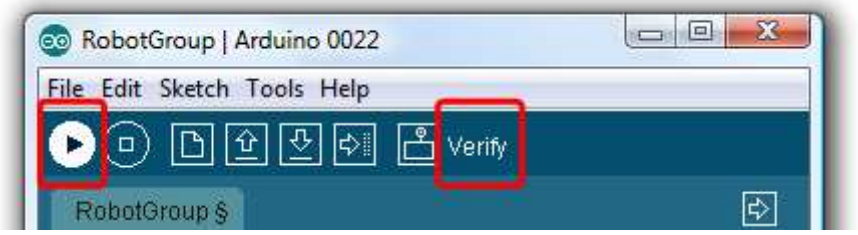


Ejemplo de prueba en Arduino

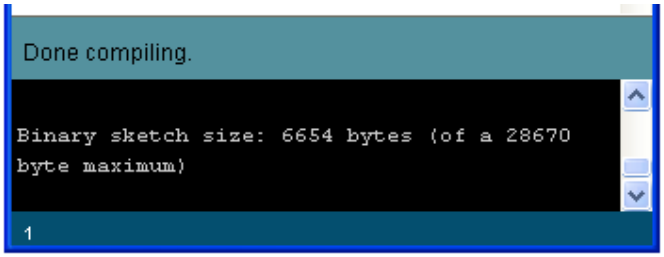
Luego de la instalación del entorno de programación Arduino, estamos en condiciones de ejecutar un programa de prueba. Para eso, luego de abrir el entorno, seleccionamos la opción "File", después "Examples", luego "Digital" y, por último, "toneMelody", como se muestra en la siguiente figura:



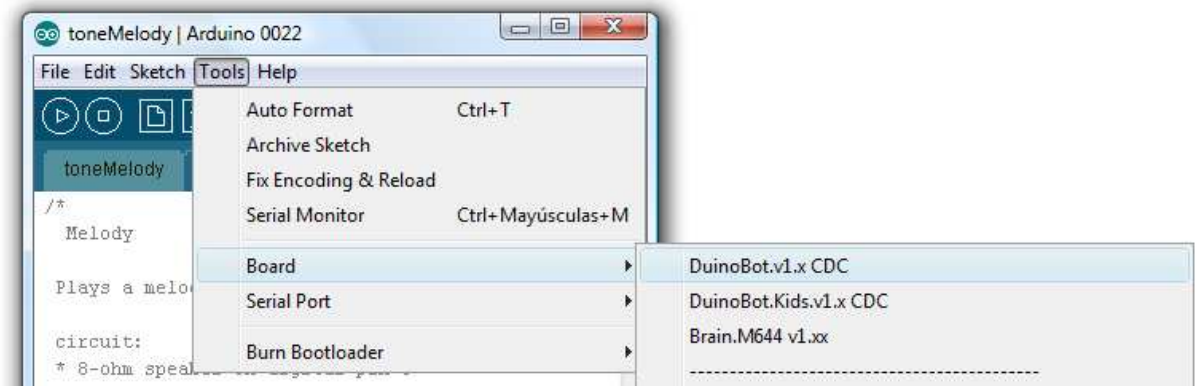
De esta manera, el entorno nos mostrará un archivo listo para ser compilado y ejecutado. Lo primero que haremos con este archivo es verificar que no tenga ningún error. Para esto presionamos el botón "Verify" que se encuentra a la izquierda arriba en el entorno, como se muestra en la siguiente figura:



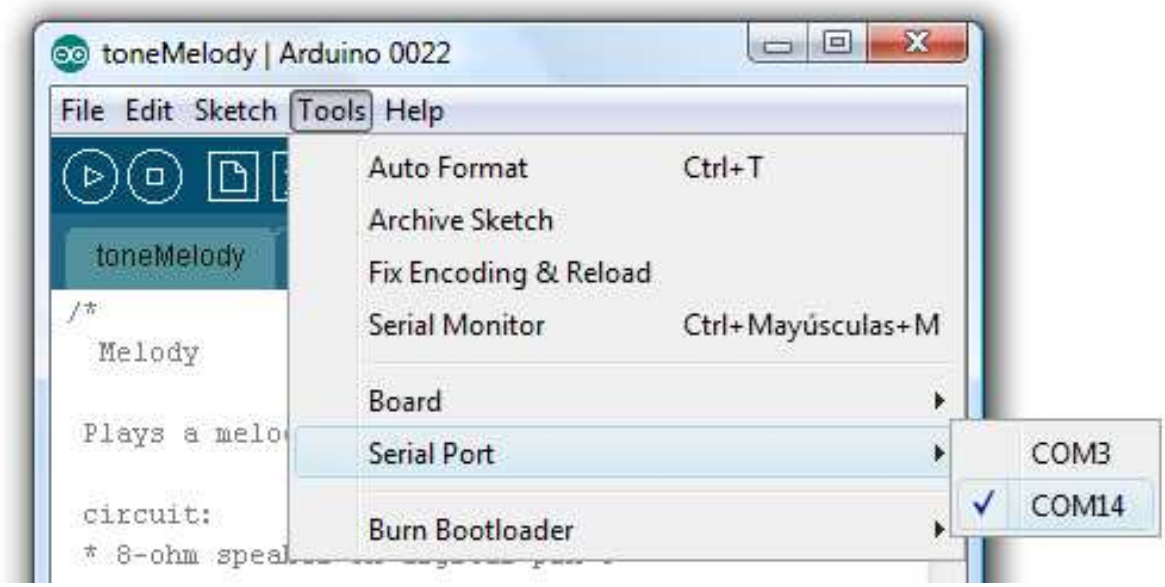
Una vez terminado el proceso de compilación, el entorno nos mostrará en la parte inferior de su pantalla un mensaje como el que se muestra a continuación:



Ahora solo nos falta mandar el programa a la placa para que esta lo ejecute. Para esto, conectamos la placa a la PC utilizando un puerto USB y luego la encendemos. Después configuramos el entorno para que reconozca el destino al que se enviará el programa. Para esto, seleccionamos la opción "Tools" y luego la "DuinoBot.v1.x CDC", como se muestra en la imagen siguiente:



Luego, nos aseguramos de que el puerto de comunicación USB esté correctamente seleccionado. Esto lo podemos chequear viendo la opción "Serial Port" dentro de "Tools", como se muestra en la siguiente figura. Lo más común es que el número de puerto sea el más alto de los disponibles.



Antes de enviar el programa a la placa, tenemos que modificarle un valor dentro del código. La línea que modificaremos es la siguiente:

```
tone(8, notes[thisSensor], 20);
```

La única modificación que hay que hacerle es cambiar el 8 por 23, debido a que este valor representa el número de pin al que está conectado el buzzer de la placa. Con lo cual, después de la modificación, quedará de la siguiente manera:

```
tone(23, notes[thisSensor], 20);
```

Ahora sí, está todo listo para subir el programa a la placa y ejecutarlo. Lo último que nos queda por hacer, es presionar el botón "Upload", con lo cual, el entorno se encargará de mandar el código compilado al controlador DuinoBot.

2. Motores

Nivel: básico/intermedio.

Objetivos

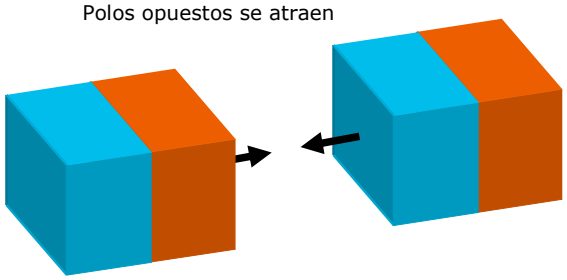
- Que los alumnos conozcan el funcionamiento de los motores eléctricos utilizados por nuestros robots.
- Que se familiaricen con las órdenes básicas usadas para controlar los motores.
- Que apliquen la teoría estudiada en proyectos que involucren a los robots.

Algo te teoría de motores

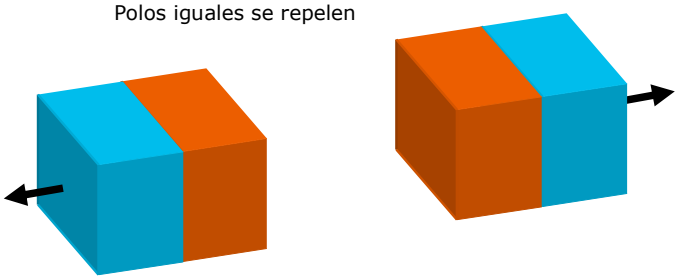
Los robots que usaremos en nuestras actividades están equipados con motores eléctricos que poseen una reducción interna. Esta, está formada por un juego de engranajes que permiten mover con mayor fuerza lo que se conecte al eje del motor. La imagen muestra el motor como lo vemos nosotros. La reducción está dentro de la carcasa del motor lo que hace que no se pueda apreciar sin desarmarlo.

Para entender mejor el funcionamiento de los motores eléctricos vamos a estudiar cómo interactúan sus partes. Para comenzar, es importante conocer qué es un imán y sus características. Los imanes tienen 2 polos: Norte y Sur. Cuando los polos son iguales se repelen o evitan, y cuando son distintos, se atraen. Esta situación se ilustra en la siguiente figura en la cual se representan los polos con distintos colores.

Polos opuestos se atraen

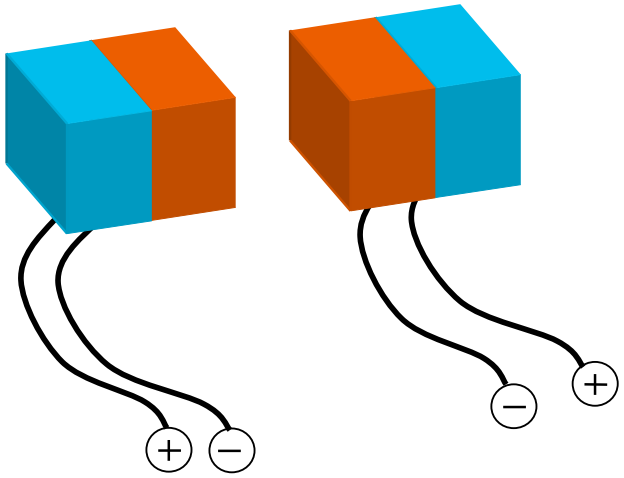


Polos iguales se repelen

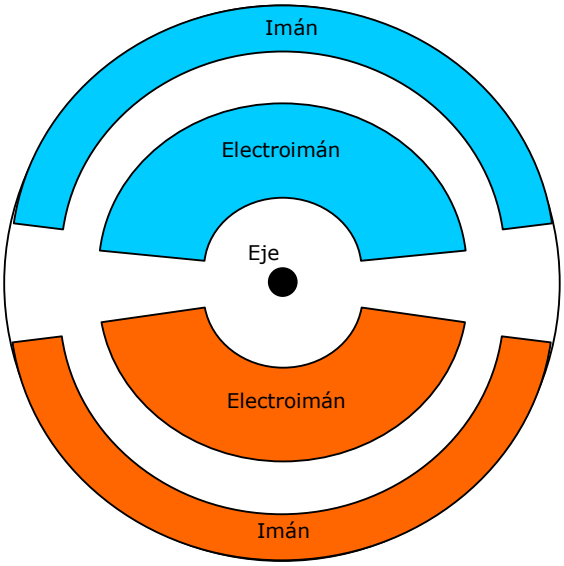


Otro elemento de importancia en el estudio del funcionamiento de un motor eléctrico, es el electroimán. Este posee la particularidad de que si le conectamos una fuente de energía eléctrica (por ejemplo una pila) se comporta como imán. Al comportarse como imán, también aparecen sus 2 polos. Luego, invirtiendo la polaridad entre la fuente y el electroimán, podemos invertir también sus polos.

Electroimanes



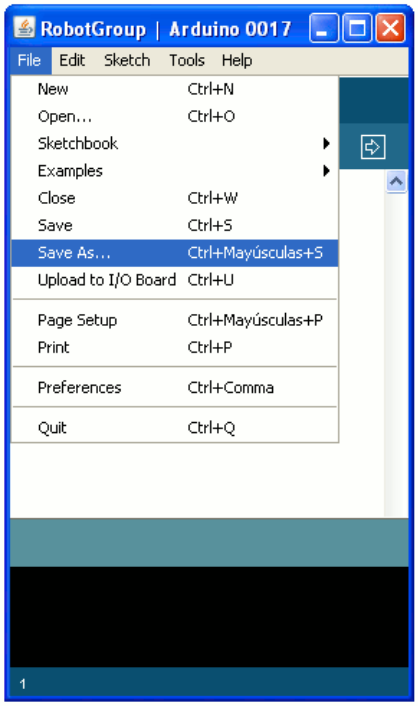
Ahora que sabemos qué son los imanes y los electroimanes, estamos listos para ver cómo interactúan estos junto a una fuente de energía eléctrica. Si cortamos un motor como los de los robots por el medio y lo miramos de frente, veremos algo como esto:



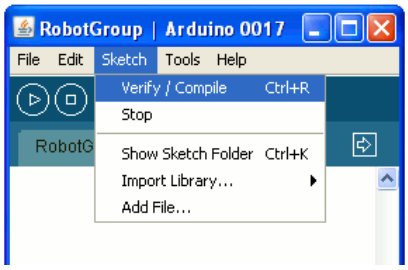
El motor tiene montado en el eje un "conmutador" que tiene el mismo efecto sobre los electroimanes (del rotor) que el que tendría si invirtiéramos las pilas. Cuando el motor gira por la repulsión entre los polos iguales de los imanes fijos y los electroimanes del rotor, el conmutador vuelve a invertir y así el movimiento no se detendrá hasta que desconectemos la fuente. Es interesante notar que, si invertimos la polaridad en la conexión entre la fuente y el motor, se invierte el sentido de giro de éste.

Introducción a la programación

Ya sabemos cómo funcionan los motores eléctricos, pero ¿cómo hacemos para que reciban las órdenes que les queremos dar? Como vimos antes, lo que queramos que haga el robot se lo tendremos que decir utilizando el entorno de programación. Una vez escrito el programa que queremos que ejecute y, luego de asegurarnos que no tiene errores, lo enviaremos al robot. Pero empecemos por las órdenes.
Cuando abrimos el entorno de programación Arduino 0022 nos encontramos con una página en blanco. Lo primero que conviene hacer en estos casos es escribir todo lo básico que tendrá que tener el programa. Luego haremos una carpeta que contenga nuestro programa y lo guardaremos allí para no correr el riesgo de perderlo. Una vez creada la carpeta, vamos a guardar nuestro proyecto en ella con el nombre que consideremos más apropiado. Para guardarlo, vamos a hacer click en el botón "File" y luego "Save As" tal como lo muestra la siguiente figura.

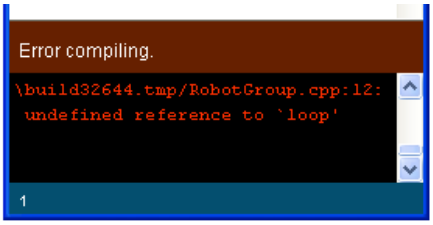


Una vez guardado el archivo vacío, empecemos a escribir nuestro programa. Lo primero que hay que hacer es escribir la declaración de un par de funciones necesarias para que el programa compile sin errores. Con el archivo en blanco podemos hacer la siguiente prueba: clickeamos la opción "Sketch" y luego "Verify/Compile", como se muestra a continuación:

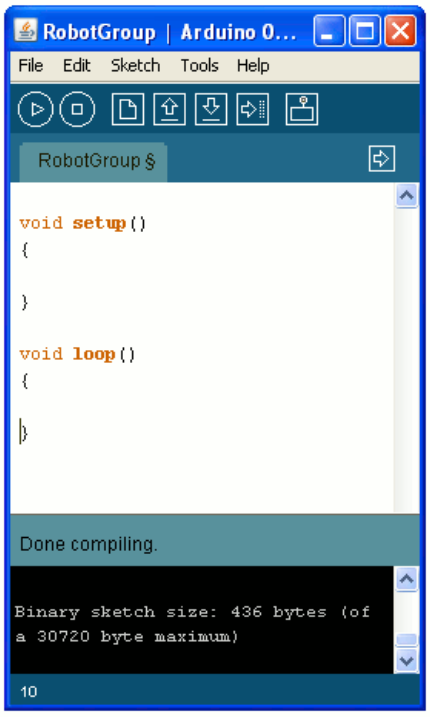


Esto hará que el entorno traduzca lo que hayamos escrito a un lenguaje entendible por el procesador. Notemos cómo el entorno nos informa qué es lo que hace por medio de una barra de estado situada debajo del editor. En este caso, nos informa que está compilando (compiling):

Luego de realizar este proceso, nos mostrará un mensaje como el siguiente:



Este mensaje de error nos indica que no declaramos un par de funciones que son indispensables para que el entorno compile correctamente. Estas funciones se llaman setup() y loop() y se declaran de la siguiente manera:



Luego de escribir estas dos declaraciones, podemos compilar de nuevo el programa y veremos que esta vez no nos devuelve ningún mensaje de error.

Funciones en programación

Una función es un conjunto de líneas de código agrupadas con una funcionalidad específica. Es importante tener en cuenta que muchas de las funciones que veremos en este curso ya fueron escritas por otra persona y solo las usaremos para dar órdenes concretas. Sin embargo, también vamos a escribir funciones desde cero para agregarles funcionalidad a nuestros programas. Volviendo a nuestro programa, hay dos funciones que no nos pueden faltar mientras trabajemos con el entorno Arduino 0022, estas funciones son las siguientes: setup() y loop().

En primer lugar tenemos una función declarada de la siguiente manera:

```
void setup()
{

}
```

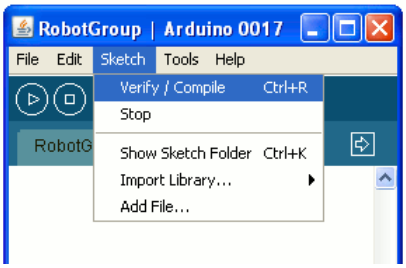
Estas líneas de código contienen la declaración de la función (`void setup()`) y las llaves que encierran al cuerpo de la misma. En cuanto a la declaración, es importante notar algunos puntos:

1. La palabra reservada `void` indica que esta función no retornará ningún valor de salida. Si uno piensa en el concepto matemático de función, inmediatamente le viene a la mente la idea de transformar un conjunto de valores en otro valor. Con las funciones en programación se puede hacer lo mismo, pero en este caso evitaremos devolver un valor numérico debido a que no lo necesitamos.

2. Luego de la palabra reservada `void`, escribimos el nombre de la función. En este caso `setup`. Cuando escribamos nuestras propias funciones podremos utilizar cualquier nombre mientras no sea una palabra reservada del sistema. Es una buena práctica elegir nombres que sean declarativos, esto es, nombres que indiquen lo que va a hacer la función. Esto nos ayudará en el futuro a arreglar problemas que surjan en nuestros programas. Por otro lado, cuando hablamos de "palabras reservadas" nos referimos a expresiones que son utilizadas por el compilador de manera exclusiva. Un ejemplo de palabra reservada es `void`.

3. Por último, la declaración de la función tiene un par de paréntesis. En caso de que la función tenga parámetros de entrada, estos se escribirían entre estos paréntesis. En los casos en que no tienen parámetros de entrada, como pasa con `setup()`, los paréntesis se escriben igual pero se dejan vacíos.

Una vez escrita la declaración de la función, se escribe el cuerpo de la misma entre las llaves. Si este se deja vacío, el programa se podrá compilar sin errores pero no hará nada. Ahora que ya tenemos las declaraciones de las dos funciones principales, probemos compilar el programa para asegurarnos de que no cometimos errores en las declaraciones. No hay que olvidarse que el lenguaje C es "case sensitive", lo que significa que no se pueden intercambiar mayúsculas con minúsculas sin que el compilador lo considere un error. Para compilar, simplemente presionamos "Verify/Compile" como se muestra en la siguiente figura:



Además de estas dos funciones que tenemos que escribir, hay muchas otras que ya están escritas y listas para ser usadas. Este es el caso de las órdenes que usaremos para decirle a un determinado motor que avance hacia delante o hacia atrás con una determinada potencia. También usaremos funciones ya escritas por otras personas para comunicarnos con la computadora y para consultar valores a través de los sensores del robot.

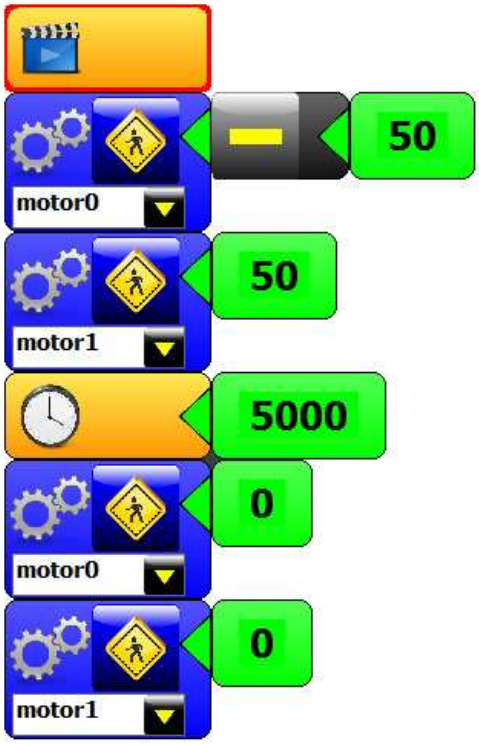
Cuando usamos órdenes para manejar los motores del robot tenemos que tener en cuenta a qué motor nos estamos refiriendo. Luego, le podremos asignar potencia y dirección a ese motor. En el caso de asignarle potencia al motor izquierdo, al cual llamaremos motor 0, la orden que escribamos en la computadora hará que el procesador se encargue de hacerle llegar dicha potencia al motor en forma de energía.

Por ejemplo, si escribimos motor0.setSpeed(100) estaremos asignando la máxima potencia que se puede al motor izquierdo, al cual identificamos con el número 0. Una vez en el procesador, esta orden producirá una salida de energía de la placa desde el pin correspondiente al motor izquierdo. Esta energía llegará al motor a través del cable que lo conecta a la placa y así empezará a funcionar.

Actividad 2.1: avanzar en línea recta

Objetivo: Que el alumno se familiarice con las órdenes básicas para motores.

En esta primera actividad queremos hacer que el robot avance por un tiempo determinado en línea recta y que luego se detenga. Utilizando el entorno de programación Minibloq es muy sencillo. Simplemente hay que agregar los siguientes bloques en el orden que se muestra a continuación:



Explicación de la actividad 2.1 hecha en Minibloq

Lo primero que hacemos es seleccionar el bloque de motor. Dentro de este, seleccionamos el número de motor que queremos que avance. En este caso, el motor 0. En caso de querer que el motor avance en sentido inverso, tendremos que asignar velocidad negativa al motor. Para esto, seleccionamos el bloque que contiene el signo negativo, el cual tiene como símbolo: –(x). Por último, con el símbolo # (numeral), asignamos la velocidad.

Como queremos que el robot avance durante un tiempo determinado y luego frene, asignamos velocidad 50 al motor 1 y luego le asignamos tiempo. Para esto, utilizamos el bloque que contiene un reloj al que le asignamos el tiempo correspondiente.

Por último, le asignamos potencia igual a cero a cada motor para frenar al robot.

Código de la actividad 2.1 en Arduino 0022

Para realizarla, escriba el siguiente código en el entorno Arduino 0022.

Actividad 2.1: hacer que el robot avance en línea recta.

```
void setup()
{
  motor0.setSpeed(-50.0);
  motor1.setSpeed(50.0);
  delay(5000);

  motor0.brake();
  motor1.brake();
}

void loop()
{ }
```

Explicación del código de la actividad 2.1

```
motor0.setSpeed(-50.0);
```

Esta línea de código es la que se usa para asignar potencia al motor izquierdo. Por convención, vamos a llamar al motor izquierdo como Motor 0 y al derecho como Motor 1. Es importante notar que en la primera parte de la orden se indica a qué motor se está haciendo referencia, mientras que en la segunda, se da la orden de asignar el valor que pusimos entre paréntesis como velocidad. Las velocidades que podemos asignar varían entre -100.00 y 100.00, siendo 0 el valor en el que el motor está detenido. Mientras que -100.00 sería la máxima velocidad de reversa. En este caso utilizamos una velocidad negativa porque los motores están en posiciones diferentes, por lo tanto, para uno avanzar hacia adelante significa ir en determinada dirección y para el otro significa ir en la dirección contraria.

```
motor1.setSpeed(50.0);
```

Esta línea de código hace lo mismo que la anterior pero, en este caso, le asigna velocidad 50 al motor derecho (Motor 1).

```
delay(5000);
```

Con esta orden le estamos diciendo al procesador del robot que por 5 segundos no ejecute nuevas órdenes. Esto lo hacemos para que el robot pueda avanzar durante ese tiempo ya que, sin esta línea, el procesador ejecutaría rápidamente las órdenes que siguen y no podríamos ver los resultados. A la función `delay` le pasamos el valor 5000 como parámetro porque esta función trabaja en milisegundos.

```
motor0.brake();
```

Por último, con esta orden le decimos al motor 0 que se detenga. También lo podríamos haber hecho asignando velocidad 0 a cada motor.

Qué es una librería en programación

Para manejar las distintas partes de un robot, como los motores o los sensores, usamos órdenes que son funciones escritas por otras personas. Estas funciones se suelen agrupar según su funcionalidad, por ejemplo, todas las órdenes que se utilizan para manejar los motores, estarán juntas. A este conjunto de funciones lo llamamos librería. Las librerías suelen contener un par de archivos: uno con extensión .h y otro con extensión .cpp. El primero toma la letra de su extensión de la palabra inglesa "header" que significa encabezado. Esto se debe a que en este archivo, estarán solamente los nombres de las funciones y de las variables más importantes del conjunto. El archivo con extensión .cpp será el que contenga el código de las funciones declaradas en el header. En el caso del entorno de programación Arduino 0022, las funciones estarán escritas en lenguaje C++, como lo vimos antes. Estos dos archivos están contenidos en una carpeta cuyo nombre es el de la librería. Por ejemplo, la librería Servo, contiene dos archivos llamados Servo.h y Servo.cpp. Esta librería está lista para que la usemos ya que viene con el entorno Arduino 0022. Lo único que tendremos que hacer en caso de querer manejar servos utilizando esta librería, será agregar la siguiente línea de código antes del código que queremos ejecutar:

```
#include <Servo.h>
```

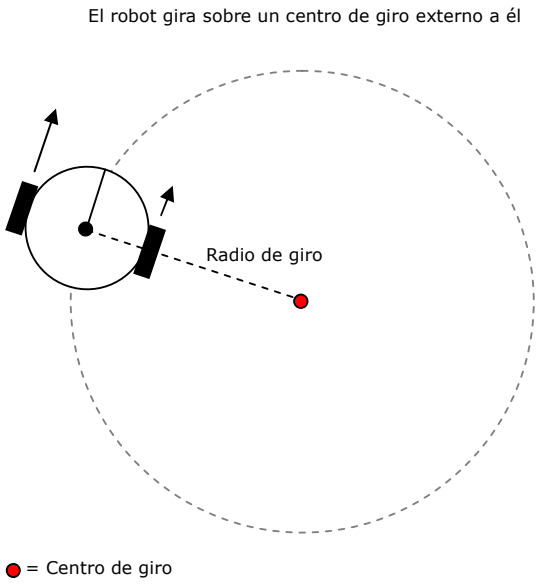
Esto se puede hacer tanto escribiendo la línea de código como haciendo clic en "Sketch", luego "Import Library" y, por último, seleccionando la librería que uno quiere utilizar como se muestra en la siguiente imagen:

Esto simplemente generará la línea de código que escribimos más arriba. Lo bueno de este método, es que nos da la seguridad de que la librería es reconocida por el entorno de programación Arduino 0022.

Actividad 2.2: dibujar una circunferencia

Objetivos: Que el alumno identifique las distintas órdenes posibles para que el robot gire y que entienda el concepto de *centro de giro*. También que sea capaz de pensar cuál es el mejor código para lograr la trayectoria deseada.

En esta actividad queremos hacer que el robot dibuje circunferencias de distinto tamaño. Tal como se muestra en la figura que está a continuación, el centro de giro del robot será el punto central de la circunferencia que este dibuje.



Para esto, escriba el siguiente código en el entorno Arduino 0022:

Actividad 2.2: Hacer que el robot dibuje un círculo.

```
void setup()
{
  motor0.setClockwise(false);
  motor0.setSpeed(50.0);
  motor1.setSpeed(100.0);
  delay(5000);

  motor0.brake();
  motor1.brake();
}

void loop()
{ }
```

Explicación del código de la actividad 2.2

```
motor0.setClockwise(false);
```

Esta línea de código la agregamos para decirle al motor 0 que su dirección será inversa a la que debería ser. Los motores de los robots giran en el sentido de las agujas del reloj, si le decimos a un motor que gire de manera inversa, tendremos a los dos motores avanzando en la misma dirección y no necesitaremos asignarle velocidad negativa a ninguno de ellos.

```
motor0.setSpeed(50.0);
motor1.setSpeed(100.0);
```

Estas son las líneas principales de este programa ya que, dependiendo de la relación que establezcamos entre las velocidades de los motores, el centro de giro estará más cerca o más lejos del robot. Para eso, elegimos la potencia que tendrá cada motor. En este caso, queremos que el motor 1 avance más rápido que el motor 0 para que el robot pueda girar en torno a un punto situado a su izquierda.

```
delay(5000);
```

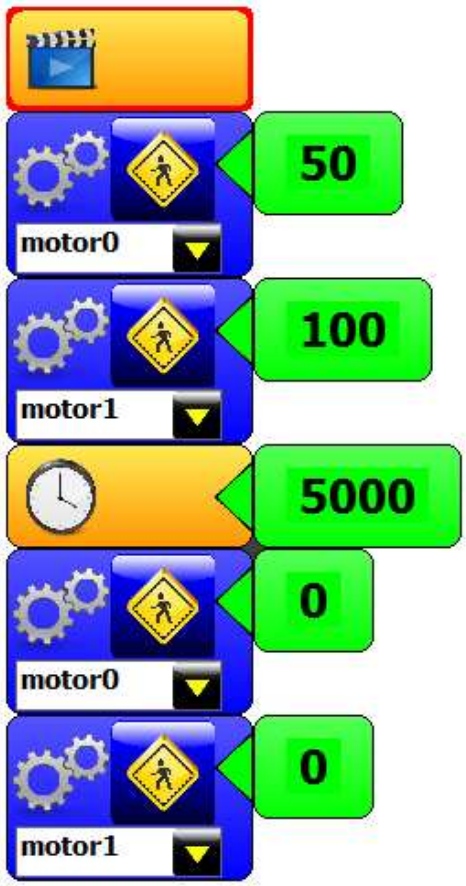
Con esta orden establecemos el tiempo que el robot usará para dibujar el círculo. Es importante aclarar que, si no se escriben las órdenes de frenado que siguen a continuación, el robot seguirá girando ya que no recibió nuevas instrucciones. Es probable que el robot no complete un círculo en el tiempo establecido, en ese caso, tendremos que modificar el tiempo que le pasamos como parámetro a la función `delay()`.

```
motor0.brake();
motor1.brake();
```

Por último, frenamos ambos motores para que el robot se detenga.

Código en Minibloq de la actividad 2.2

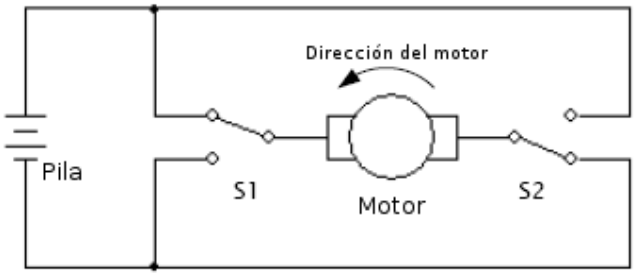
Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los siguientes bloques:



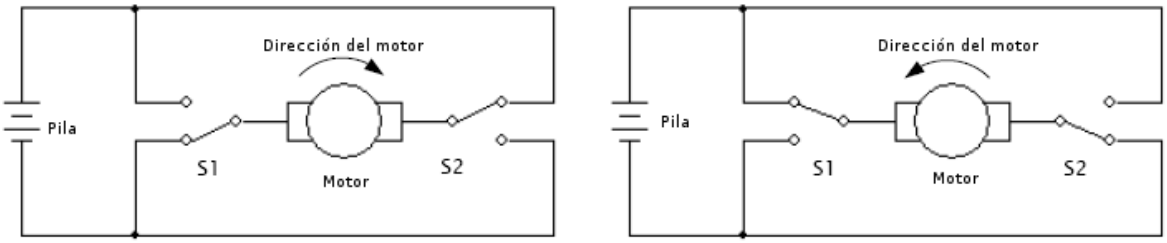
Puente H

Ya hemos visto que al motor le asignamos velocidades comprendidas entre -100 y 100. Siendo las velocidades negativas las que harán que retroceda y las positivas que avance. ¿Pero cómo funciona internamente este cambio de dirección en el motor? Para entender esto y, antes de pasar a la siguiente actividad, estudiaremos el siguiente concepto.

Se llama Puente H o H-Bridge en inglés, al circuito formado por el motor y su fuente de energía cuando se disponen como lo muestra la siguiente figura:



Tal como se muestra en la figura, la corriente que circula a través de las compuertas S1 y S2 podría tomar circuitos distintos dependiendo de si estas compuertas están "hacia arriba" o "hacia abajo". Luego, la configuración que se elija, hará que el robot circule hacia un lado o hacia el otro, como se muestra en la siguiente figura:

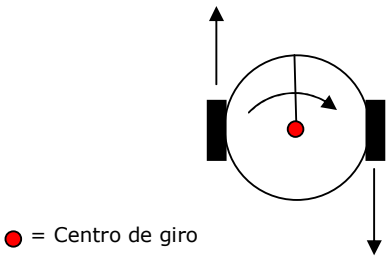


Luego, cuando le decimos a un motor que su velocidad será de 100, las compuertas S1 y S2 arman el circuito de tal manera que el motor gire hacia delante, mientras que, cuando asignamos velocidad -100, las compuertas se cerrarán de manera inversa haciendo que el motor gire con máxima potencia hacia el lado opuesto.

Actividad 2.3: hacer que el robot gire sobre su propio eje

En esta actividad queremos hacer que el robot gire en el lugar donde está sin desplazarse. Tal como lo muestra el esquema que está a continuación, tendremos que hacer que una rueda avance y la otra retroceda, pero que ambas lo hagan a la misma velocidad.

El robot gira sobre su propio centro



Para esto, escriba el siguiente código en el entorno de programación Arduino 0022:

Actividad 2.3: Hacer que el robot gire sobre su propio eje.

```
void setup()
{
  motor0.setClockwise(false);
  motor0.setSpeed(70);
  motor1.setSpeed(-70);
  delay(4000);

  motor0.brake();
  motor1.brake();
}

void loop()
{ }
```

Explicación del código de la actividad 2.3

```
motor0.setClockwise(false);
```

Con esta línea de código seteamos la dirección de giro del motor 0 en la dirección opuesta a las agujas del reloj.

```
motor0.setSpeed(70);
motor1.setSpeed(-70);
```

Para que el robot gire sobre su propio eje, vamos a asignarle la misma potencia a cada motor, pero uno tendrá que retroceder mientras el otro avanza. Por ese motivo, al Motor 1, le asignamos potencia (-70.0).

```
delay(4000);
```

Gracias a esta línea de código, el procesador del robot esperará 4 segundos antes de ejecutar nuevas órdenes.

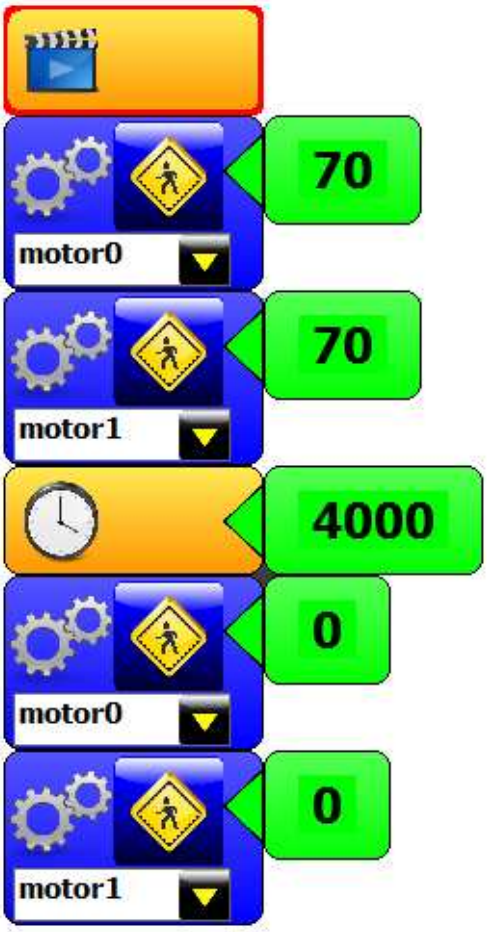
```
motor0.brake();
motor1.brake();
```

Para finalizar, hacemos que el robot se detenga frenando ambos motores.

Ahora, pensemos en la siguiente actividad: se quiere hacer que el robot gire sobre su propio eje como en el último ejercicio, pero esta vez queremos que desacelere desde su velocidad máxima y, una vez que llegue a su cero, frene. Para hacer esto, necesitaríamos poder modificar el valor de la velocidad cada cierta cantidad de tiempo. O sea, lo que querríamos hacer sería usar el código de la última actividad una vez con velocidad 100, luego reutilizarlo con velocidad 90 y así ir decrementando la velocidad a la que gira el robot hasta que llegue a cero. Para esto usaremos variables a las que les modificaremos el valor dentro de un ciclo. Cómo hacer esto, lo explicamos a continuación.

Código en Minibloq de la actividad 2.3

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación. Hay que tener en cuenta que, en el robot Multiplo N6, los motores están en direcciones opuestas por lo que asignarles a ambos la misma velocidad, hará que uno avance mientras que el otro retrocede.



Variables en programación

En programación, una variable es simplemente un espacio de memoria en el que guardamos algo. Para acceder a ese algo, lo hacemos a través del nombre que le asignamos a la variable cuando la declaramos. Luego, podemos modificar el contenido de ese espacio de memoria tantas veces como sea necesario. Por ejemplo, si quisiéramos hacer una variable que represente la velocidad de los motores, podríamos declararla de la siguiente manera:

```
float speed = 100;
```

Esta declaración, al igual que las declaraciones de las funciones, está formada por varias partes:

1. La palabra reservada "float" indica que lo que vamos a guardar en el espacio de memoria que llamamos "speed" será un número de real (float viene de punto flotante, que es como se llama a los números reales en programación debido a la forma en que trabaja el procesador). Es indispensable que, al declarar una variable, se indique su tipo. O sea, siempre hay que decir qué tipo de dato se va a guardar en la memoria. Los tipos de datos que nos interesarán por ahora son los siguientes:

Tipo de dato	Palabra reservada	Ejemplos
Número real	`float`	`10.5, -100.00`
Número entero	`int`	`10, 100, -15`
Caracter	`char`	`a, b, c`
Cadena de caracteres	`string`	`"Hello World"`

2. Luego de declarar el tipo de dato que se guardará, se le asigna un nombre. Este nombre es elegido por el programador y puede ser cualquier palabra menos las que son reservadas por el sistema. Es recomendable que el nombre de la variable se relacione con el uso que se le dará, esto ahorrará mucho trabajo cuando haga falta corregir el código escrito ya que nos resultará más fácil saber lo que esta variable debería hacer. En nuestro ejemplo, elegimos el nombre "speed" porque esta variable será usada para asignar velocidad a los motores del robot.

3. Por último, a la variable real speed, le asignamos un valor inicial que será guardado en el lugar de la memoria designado para esta variable. En nuestro caso, el valor de inicio será 100.

Ahora que tenemos nuestra variable declarada e inicializada, podemos empezar a trabajar con ella. Empecemos cambiándole el valor, esto lo haremos simplemente escribiendo el nombre de la variable e igualándola al nuevo número.

```
speed = 0;
```

Es importante notar que ya no necesitamos decir qué tipo de dato guarda la variable, solo necesitamos su nombre. Y, por último, podemos usar esta variable dentro de una función que necesite números reales, como se muestra a continuación:

```
float speed = 100;

motor0.setClockwise(false);
motor0.setSpeed(speed);
motor1.setSpeed(speed);
delay(1000);
```

También podemos modificar el valor de la variable en función del valor que tiene. Por ejemplo, si queremos que cada cierto tiempo la variable "speed" aumente en 10, lo haremos de la siguiente manera:

```
speed = speed + 10;
```

Al escribir esta línea de código, lo que estamos haciendo es asignando a la variable speed, el valor que tenía pero incrementado en 10. Pero para que realmente tenga sentido usar esta variable, haremos uso de otro concepto importante de la programación: la estructura de control "`while`".
Las órdenes de un programa son ejecutadas una tras otra por el procesador. Si esta secuencialidad no se pudiese romper, no podríamos hacer que el robot tome decisiones. Por ejemplo, piense en el caso de un robot que está en otro planeta avanzando en línea recta y recolectando piedras para su estudio. Si este robot se encontrara con un obstáculo que no le permitiera avanzar, se volvería inútil rápidamente. En cambio, si en su código estuviese especificado que, ante cierto tipo de obstáculo, el robot detenga su avance y la recolección de piedras y luego lo esquive, el robot seguiría cumpliendo su misión. Este comportamiento, debería estar especificado en el código del robot. Las estructuras de control nos permiten romper la secuencialidad de un programa para ejecutar otra parte del código que no tiene por qué ser el que le sigue a la orden que se está ejecutando.

Ahora bien, una estructura de control como "while", se utiliza de la siguiente manera:

```
while(/* CONDICIÓN */)
{
/* Código que se repite mientras la condición sea verdadera */
}
```

Lo que llamamos "condición" puede ser simplemente una ecuación o expresión matemática que podamos evaluar como verdadera o falsa. Por ejemplo, si sabemos que nuestra variable "speed" actualmente almacena el valor 100, y escribimos las siguientes condiciones, tendremos los siguientes resultados:

Condición	Valor de verdad de la ecuación	Resultado
`(speed = 100)`	Verdadero (TRUE)	Se ejecuta el código entre llaves.
`(speed < 50)`	Falso (FALSE)	No se ejecuta el código.
`(speed > 0)`	Verdadero (TRUE)	Se ejecuta el código.
`(speed <= 100)`	Verdadero (TRUE)	Se ejecuta el código.
`(speed < 90)`	Falso (FALSE)	No se ejecuta el código entre llaves.

Volviendo a nuestro ejemplo en el que queremos hacer que el robot gire desacelerando, podríamos escribirlo de la siguiente manera:

```
float speed = 100;

while(speed > 0)
{
  motor0.setSpeed(speed);
  motor1.setSpeed(-speed);
  delay(1000);

  speed = speed - 10;
}
```

Analicemos el código completo línea por línea para entenderlo bien.

```
float speed = 100;
```

Con esta línea de código generamos un espacio en memoria en el que guardaremos números reales. Al número real almacenado lo llamamos speed y lo referenciamos por su nombre. Inicialmente, este espacio de memoria tiene el valor 100.

```
while(speed > 0)
```

Mientras la variable "speed" contenga un valor mayor que cero, se ejecutará el código encerrado entre llaves. Cuando el valor de dicha variable sea igual o menor que cero, el código entre llaves no se ejecutará. Este código será salteado y se ejecutará la línea que lo preceda.

```
  motor0.setSpeed(speed);
```

Al motor izquierdo le asignamos como potencia el valor almacenado en "speed". Si este valor es igual a 100, el motor avanzará a máxima velocidad, si el valor es igual a 0, el motor se frenará.

```
  motor1.setSpeed(-speed);
```

Al motor derecho le asignamos la potencia almacenada en la variable "speed" pero con un signo negativo adelante. O sea, si la variable "speed" vale 100, el motor derecho avanzará con velocidad -100.

```
  delay(1000);
```

Dejamos de ejecutar órdenes nuevas por 1 segundo para que se vean los resulados.

```
  speed = speed - 10;
```

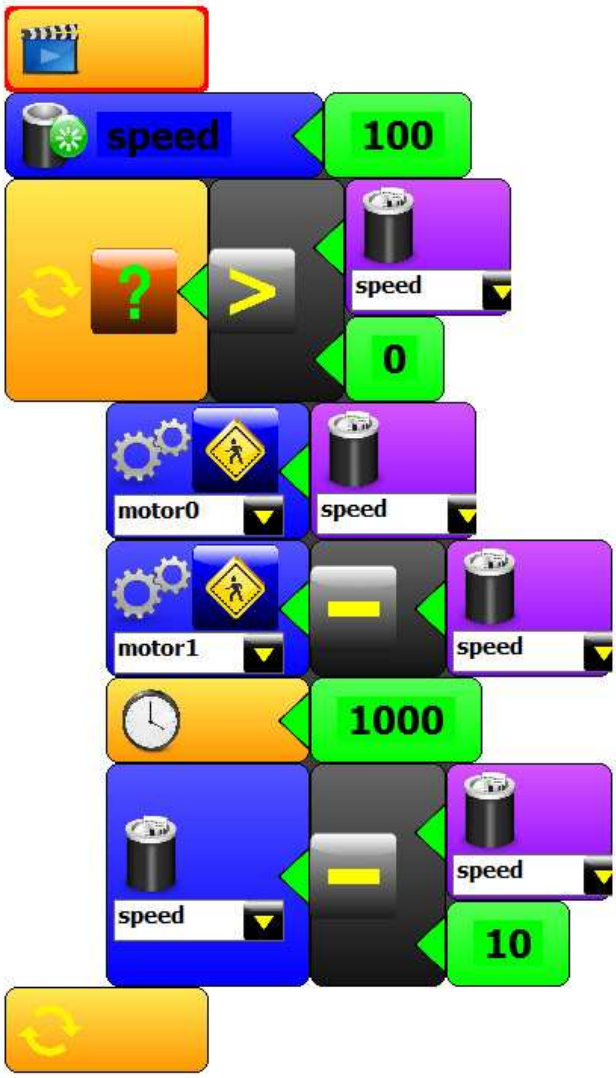
Le restamos 10 al valor almacenado en "speed". Así, tendremos que cada vez que el código ejecute las órdenes entre llaves, la variable se achicará en 10. Llegará un momento en que la variable valdrá 0, entonces, este código no se ejecutará más porque la condición dentro del "while" será falsa.

Ejemplo 2.1: El código completo para que el robot decremente su velocidad.

```
void setup()
{
  motor0.setClockwise(false);

  float speed = 100;

  while(speed > 0)
  {
    motor0.setSpeed(speed);
    motor1.setSpeed(-speed);
    delay(1000);

    speed = speed - 10;
  }
}

void loop()
{}
```

Código en Minibloq del ejemplo 2.1

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Actividad 2.4: giro con distintas velocidades

Esta actividad es parecida a la anterior, pero en este caso, queremos que el robot luego de llegar a su velocidad máxima, empiece a frenar hasta detenerse. Para esto, escribimos el siguiente código:

Actividad 2.4: hacer que el robot gire con distintas velocidades.

```
void setup()
{
  float speed = 0;
  motor0.setClockwise(false);

  while(speed < 100)
  {
    motor0.setSpeed(speed);
    motor1.setSpeed(-speed);
    delay(1000);

    speed = speed + 10;
  }

  while(speed > 0)
  {
    motor0.setSpeed(speed);
    motor1.setSpeed(-speed);
    delay(1000);
    speed = speed - 10;
  }
}

void loop()
{}
```

Explicación del código de la actividad 2.4

Si bien una parte del código es igual a la escrita más arriba, vamos a analizar las partes más importantes y los cambios que realizamos en la lógica del programa.

```
float speed = 0;
```

Con esta línea de código generamos un espacio en memoria en el que guardaremos números reales. Al número real almacenado lo llamamos speed y lo referenciamos por su nombre. Inicialmente, este espacio de memoria tiene el valor 0. Esta línea fue modificada ya que ahora el robot empezará frenado y aumentará su velocidad hasta el máximo. Luego se reducirá la misma hasta llegar a cero nuevamente.

```
while(speed < 100)
```

Mientras la variable "speed" contenga un valor mayor que 100, se ejecutará el código encerrado entre llaves. Cuando el valor de dicha variable sea igual o menor que 100, el código entre llaves no se ejecutará. Este código será salteado y se ejecutará la línea que lo preceda. Este es otro de los cambios realizados ya que el límite que tendrá que alcanzar la variable esta vez será una cota superior y no una inferior.

```
  motor0.setSpeed(speed);
```

Al motor izquierdo le asignamos como potencia el valor almacenado en "speed". Si este valor es igual a 100, el motor avanzará a máxima velocidad, si el valor es igual a 0, el motor se frenará.

```
  motor1.setSpeed(-speed);
```

Al motor derecho le asignamos la potencia almacenada en la variable "speed" pero con un signo negativo adelante. O sea, si la variable "speed" vale 100, el motor derecho avanzará con velocidad -100.

```
  delay(1000);
```

Dejamos ejecutar órdenes nuevas por 1 segundo para que se vean los resulados.

```
  speed = speed + 10;
```

Le sumamos 10 al valor almacenado en "speed". Así, tendremos que cada vez que el código ejecute las órdenes entre llaves, la variable se incrementará en 10. Llegará un momento en que la variable valdrá 100, entonces, este código no se ejecutará más porque la condición dentro del "while" será falsa. Este es otro de los cambios que hay que introducir en el código ya que este determinará en qué dirección crecerá el valor almacenado en la variable.

```
while(speed > 0)
```

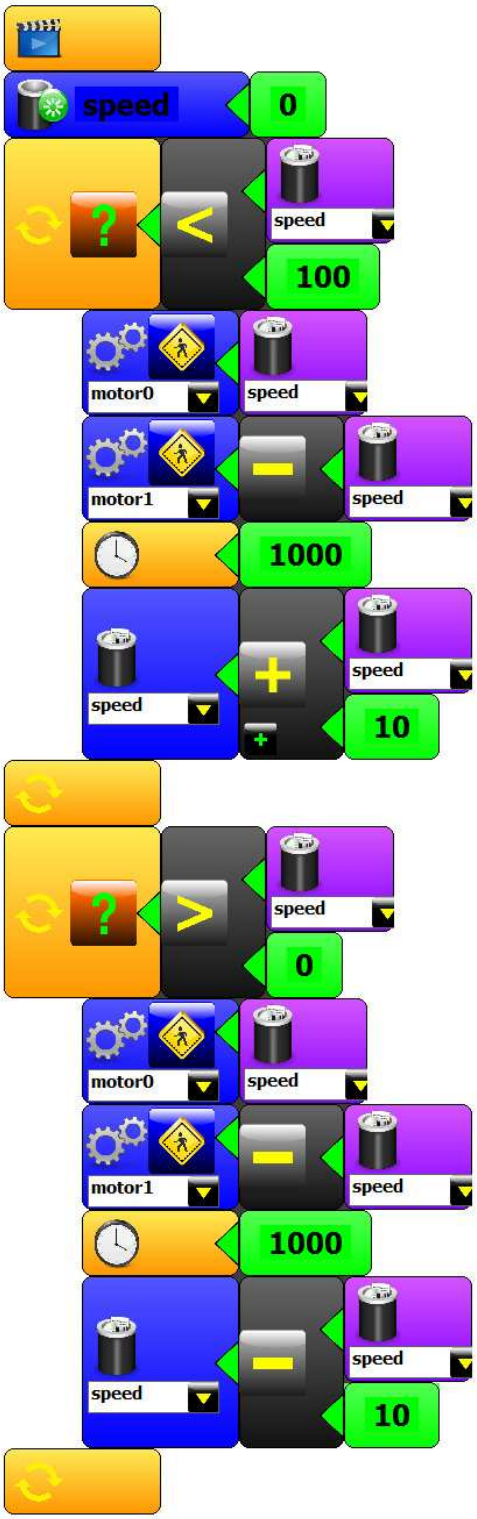
Una vez que haya terminado de ejecutar el primer ciclo, el programa deberá ejecutar el segundo. En este la velocidad empezará a disminuir hasta que el robot se detenga. Es por esta razón que comparamos a la variable que guarda la velocidad con cero. Cuando esta condición sea verdadera, el ciclo se dejará de ejecutar y los motores se detendrán.

```
    motor0.setSpeed(speed);
    motor1.setSpeed(-speed);
    delay(1000);
    speed = speed - 10;
```

Dentro del bucle, asignamos a ambos motores la velocidad representada por la variable speed. En un caso con un signo negativo para que avance en dirección contraria al otro motor. Luego, la variable speed será decrementada en cada ejecución del ciclo hasta llegar a valer 0, en cuyo caso la condición del while será falsa y, por lo tanto, no se ejecutará más este código.

Código en Minibloq de la actividad 2.4

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Variables en Arduino 0022

Como vimos anteriormente, el entorno de programación que utilizan los robots Multiplo es el Arduino 0022. Una de las razones por las que se eligió este entorno es por la gran cantidad de información que hay disponible para sus usuarios. Particularmente, en el sitio de Arduino, podemos encontrar una excelente guía de referencia que nos servirá para consultar distintos temas referidos a la programación con Arduino 0022. A continuación, vamos a ver cómo puede servirnos dicha guía de referencias, para lo cual, vamos a ingresar en el sitio oficial de Arduino:

http://www.arduino.cc/

Una vez en el sitio oficial, vamos a la sección "Reference", tal como se muestra en la figura:

En esta sección encontramos referencias de programación del entorno Arduino 0022. Muchas son propias del entorno, mientras que otras son propias del lenguaje C/C++. Por ejemplo, en primer lugar encontramos explicaciones acerca de las funciones setup() y loop(), las cuales son indispensables a la hora de programar con este entorno, pero no lo son cuando se programa en otros entornos que también usan el lenguaje C/C++. En general, cuando se programa en este lenguaje, la función principal se llama main() y es indispensable escribirla, ya que ella marcará el punto de inicio del programa. A continuación se muestra una figura que indica dónde encontrar información detallada acerca de la función setup():

Una vez dentro de la sección de la función, podremos encontrar una pequeña explicación acerca de la función y un ejemplo de cómo utilizarla. En este caso, puede no ser de tanta ayuda, pero en otros casos, consultar esta guía de referencia podrá ayudarnos en la investigación de funciones que salen del alcance del presente curso y que pueden resultar interesantes para algún proyecto particular que tengamos.

Volviendo a la guía de referencia, vemos que la misma está divida en tres columnas. La primera se llama "Structure", ya que contiene información acerca de temas que son de carácter estructural, como por ejemplo las funciones que acabamos de nombrar. La tercera columna se llama "Functions" y es la que describe distintas funciones que están disponibles en el entorno. Algunas de ellas las usaremos más adelante en este curso. En medio de estas dos columnas, tenemos una llamada "variables" que es la que nos interesa en este momento.

Dentro de esta columna, tenemos tres clases de variables nombradas como "Constants", "Data Types" y "Conversion". La primera de ellas nos será de mucha utilidad más adelante, cuando usemos algunas funciones propias del entorno. La segunda, nos amplía la información acerca de los tipos de datos que utilizamos y, además, nos describe otros que no usamos, como por ejemplo, boolean, unsigned int, long, byte, etc. Es interesante analizar algunos de estos tipos y ver qué diferencias tienen con los vistos hasta aquí. Por ejemplo, el tipo de dato "unsigned int" es igual al tipo de datos "int" con la diferencia de que no soporta números negativos. Por lo tanto, trabajar con este tipo de datos será como trabajar con números naturales. Un dato importante para tener en cuenta es el siguiente, los números enteros son almacenados en secciones de memoria de 2 bytes de tamaño, lo que equivale a 16 bits. Debido a esto, se podrán utilizar números enteros mayores o iguales a -32768 y menores a 32768. Luego, utilizando valores del tipo unsigned int, podremos utilizar números mayores o iguales a 0 y menores a 65536.

Otro tipo de dato interesante es el llamado "boolean". Este puede tener solamente dos valores: true (verdadero) o false (falso). Pensemos en el ejemplo donde utilizamos una condición que podía ser verdadera o falsa. En ese caso, podríamos escribir simplemente "true" dentro de los paréntesis y así haríamos que la condición fuese siempre verdadera, obligando al programa a ejecutar el código encerrado entre llaves. Tal ejemplo quedaría de la siguiente manera:

```
while(true)
{
/* Código del ciclo */
}
```

Mejor aún sería crear una variable booleana a la que le podríamos asignar el valor "true" o "false" dependiendo del momento. Luego, esa variable sería la condición dentro de los paréntesis. En el ejemplo que se muestra a continuación, se crea una variable llamada "ejemplo" y se le da el valor "true". De esta manera el programa ingresará en el ciclo la primera vez pero luego cambiará el valor a "false" con lo cual, no volverá a entrar al ciclo.

```
boolean ejemplo = true;

while (ejemplo)
{
  // Código del ciclo
  ejemplo = false;
}
```

Por último, en la columna de variables de la página de referencias de Arduino, encontramos la sección "Conversion". Esta sección muestra formas de convertir una variable de un tipo en una variable de otro tipo. Esto se logra anteponiendo al nombre de la variable existente la palabra que se utilizaría para declararla en otro caso.

Ejercicios de integración

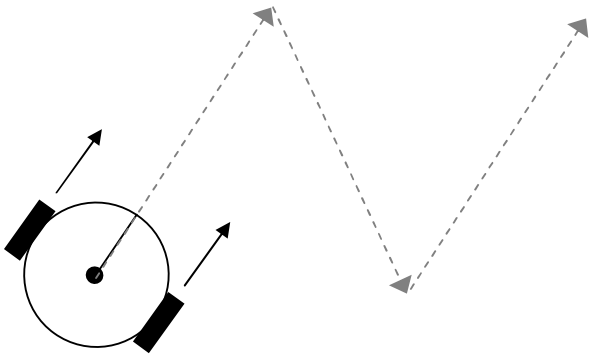
¿Cómo haría para que el robot empiece sin girar e incremente su velocidad hasta llegar a 100 y luego la decremente hasta llegar de nuevo a 0?

Actividades para el alumno

Relación entre velocidades de motores
Completar la siguiente tabla con los valores que le asignaría a cada motor para que el robot realice la trayectoria descripta en la columna "objetivo".

Centro de giro	Motor 0	Motor 1
En la rueda izquierda		
Cerca, a la derecha del robot		
Lejos, a la izquierda del robot		

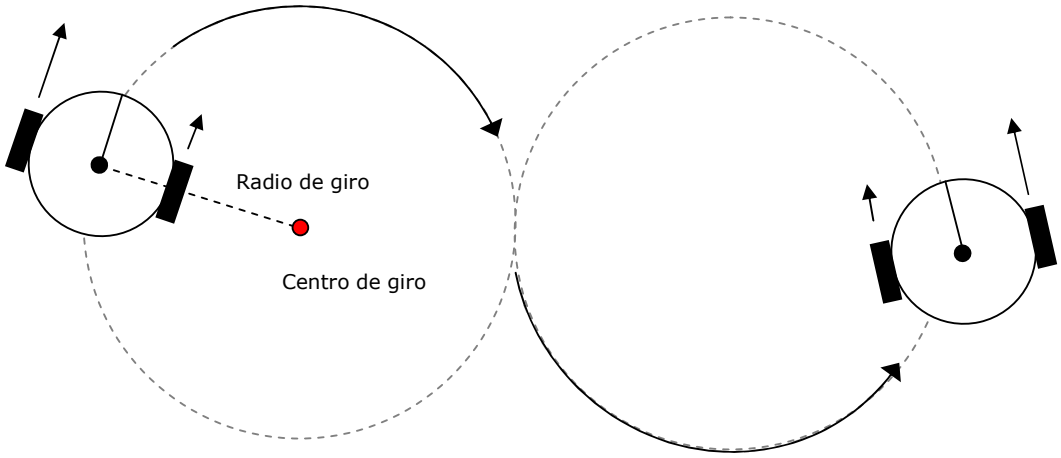
Cómo dibujar la letra "N"
Completar las órdenes escritas a continuación para que el robot dibuje la letra "N" tal como lo muestra la figura. No te olvides de tener en cuenta que el robot tiene que frenar al final del recorrido.



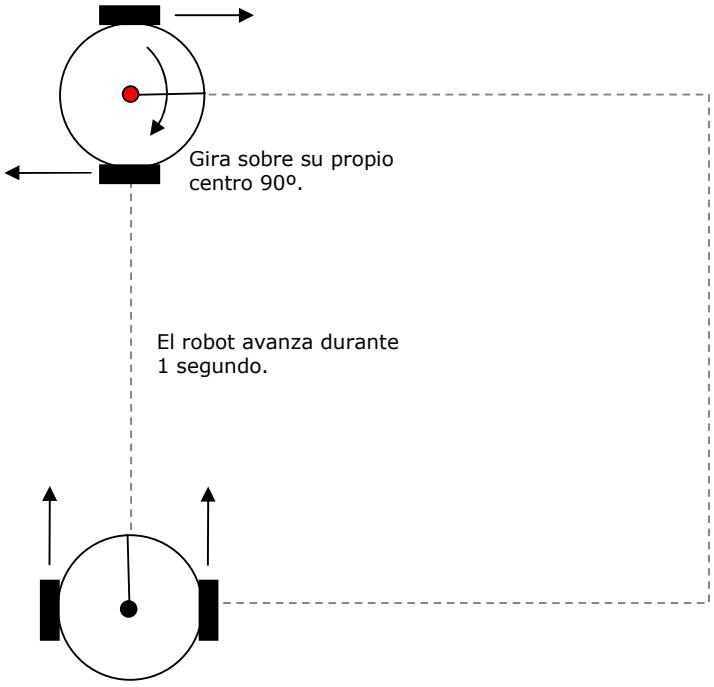
```
motor0.setSpeed(    );
motor1.setSpeed(    );
delay(    );

motor0.setSpeed(    );
motor1.setSpeed(    );
delay(    );

motor0.setSpeed(    );
motor1.setSpeed(    );
delay(    );

motor0.setSpeed(    );
motor1.setSpeed(    );
delay(    );

motor0.setSpeed(    );
motor1.setSpeed(    );
delay(    );

motor0.setSpeed(    );
motor1.setSpeed(    );
delay(    );

motor0.setSpeed(    );
motor1.setSpeed(    );
delay(    );

motor0.setSpeed(    );
motor1.setSpeed(    );
```

Actividades de integración con robots

Dibujar una figura en forma de "8"
Usando las órdenes vistas para el manejo de los motores del robot: ¿Cómo haría para que el robot dibuje un "ocho" como el de la siguiente figura?



Dibujar un cuadrado
Dibujar, usando las órdenes aprendidas para motores, un cuadrado como el que se muestra en la siguiente figura.



Actividad de aplicación a la física

Usando el código de la actividad 2.1, podemos estudiar el concepto de velocidad. Teniendo en cuenta que la misma se mide dividiendo la distancia recorrida por un móvil por el tiempo que tarda, requeriremos que los alumnos asignen una velocidad determinada a ambos motores y midan la distancia recorrida en un lapso de tiempo elegido. Luego, completar la siguiente tabla con los valores obtenidos.

Potencia asignada a los motores	Distancia recorrida	Tiempo de recorrido	Velocidad
40			
60			
80			
100			

Preguntas para investigar

1- Investigar en qué otros dispositivos o juguetes se encuentra este tipo de motores.
2- ¿Dependen estos dispositivos o juguetes de un procesador central para manejar los motores como en el caso de los robots?
3- Investigar qué otros tipos de motores existen y en qué se diferencian de los que usan los robots.

3. Comunicaciones

Nivel: básico/intermedio.

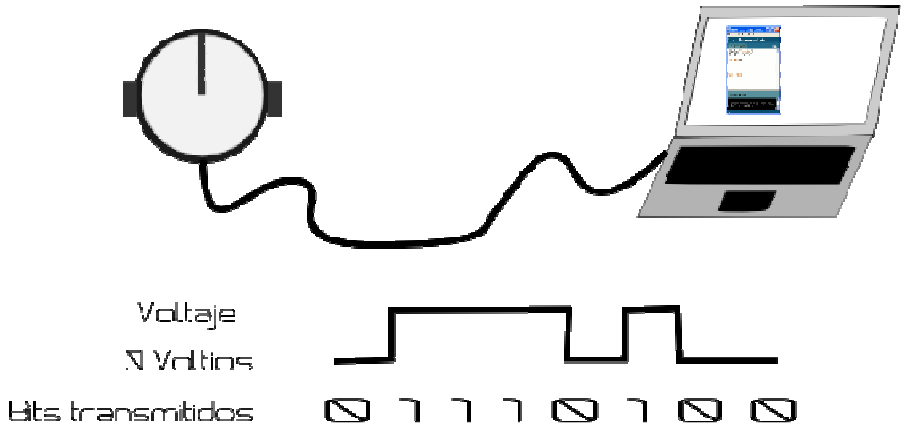
Objetivos

- Que los alumnos comprendan la interacción existente entre la computadora y el robot.
- Que también puedan establecer comunicaciones entre el robot y la computadora a través del software.

Algo de teoría de comunicaciones

Cuando hablamos de comunicaciones tenemos que tener en cuenta que en toda comunicación deberían participar un transmisor, un medio y un receptor. En el caso de nuestros robots, el papel del transmisor y del receptor lo puede tomar tanto la computadora como el robot y el medio será un cable o el mismo aire en caso de estar usando módulos Bluetooth, ya que estos son inalámbricos. Por otro lado, también es importante diferenciar la comunicación analógica de la digital. La analógica, es la que llevamos a cabo cuando hablamos y su principal característica es que se puede representar por medio de funciones matemáticas continuas. La digital, en cambio, se representa por medio de valores discretos. Por ejemplo, con ceros y unos como en el caso de las computadoras.

En el caso de nuestros robots, al comunicarse con la computadora lo hacen usando solo los dígitos 0 y 1 lógicos. Estos dos valores, internamente, son representados por dos voltajes: 0 voltios (ausencia de tensión) para representar un 0 lógico, y 5 voltios para representar un 1 lógico. La siguiente figura ejemplifica la transmisión de información entre la computadora y el robot a través de un cable. Abajo se muestran los bits transmitidos representados por una señal eléctrica que oscila entre los 0 y los 5 voltios.



En resumen, el procesador transmite solo ceros y unos que son traducidos por la computadora a voltajes eléctricos como 0 voltios o 5 voltios. Luego, el robot recibe estos voltajes y los vuelve a transformar en ceros y unos. Internamente, el procesador usará estos dos dígitos para realizar todas sus tareas. Estos ceros y unos son llamados bits y comúnmente son reunidos en grupos de ocho llamados Bytes. Como vimos antes, cuando declaramos un número entero en nuestro entorno, usamos un espacio de memoria de 2 Bytes para almacenarlo. Por lo tanto, cada número entero que utilizamos está formado por 16 bits (2 Bytes x 8).

Cada posición dentro el Byte representa una potencia de 2. La siguiente tabla representa la estructura de un Byte. En la primera fila se detalla el número de posición de cada bit, es importante notar que se empieza a contar por la posición cero. En la segunda fila se muestra la potencia de 2 que corresponde a dicha posición. Por último, en la tercera fila, se muestra el resultado de multiplicar un bit con valor 1 por la potencia de 2 de la posición en la que se encuentra.

Posición 7	Posición 6	Posición 5	Posición 4	Posición 3	Posición 2	Posición 1	Posición 0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Luego, el valor máximo que podemos expresar usando un Byte es:
128 + 64 + 32 + 16 + 8 + 4 +2 + 1 = 255. Este valor es muy común verlo, por ejemplo, cuando se configura una red, ya que las direcciones IP que se utilizan están formadas por 4 números de 8 bits, lo que da una longitud total de 32 bits para cada dirección IP.

Siguiendo con el tema comunicaciones, las podemos clasificar también según el uso del medio: comunicación en paralelo y comunicación serial. En el caso de la comunicación serial, los bits se transmiten "uno atrás del otro", como si fuera una fila de ceros y unos. Este es el tipo de comunicación que usan nuestros robots tanto cuando están conectados con conectores RS-232 o con USB. En la comunicación en paralelo, los bits viajan en ráfagas "uno al lado del otro", como si fuera una autopista con varios carriles por los que circulan autos en la misma dirección.

En el caso particular de nuestros robots, las comunicaciones se pueden realizar a través de Bluetooth o a través de USB. Bluetooth es un protocolo para redes inalámbricas, lo que significa que es un conjunto de reglas que definen la comunicación entre el emisor y el receptor. Luego, cualquier dispositivo que respete estas reglas, podrá comunicarse con un dispositivo Bluetooth. Sin embargo, estas reglas solo se aplican a la transmisión física del mensaje.

Otra posibilidad para realizar la comunicación entre robot y computadora, es que se los comunique a través de un cable con conectores USB (Universal Serial Bus). Esta tecnología fue pensada especialmente para comunicar periféricos con computadoras, como por ejemplo cámaras digitales o discos de almacenamiento externo.

Actividad 3.1: "Hello world!"

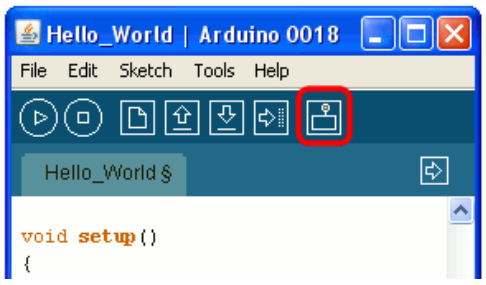
En esta primera actividad de comunicaciones queremos hacer que la placa se comunique con nosotros a través de la computadora. Para esto, tendremos que usar funciones de la librería `Serial`. El porqué del nombre de esta librería tiene que ver con la manera como los bits viajan a través del cable entre la computadora y la placa. Esto es, "de a uno a la vez" o "uno tras otro". Recordemos que la placa envía un pulso eléctrico que puede ser de 0 voltios o 5 voltios. En el primer caso, la computadora interpretará la ausencia de voltaje como un 0 lógico, mientras que en el segundo caso lo interpretará como un 1 lógico. Así podremos formar cualquier secuencia de ceros y unos que queramos. Luego, esas secuencias de números binarios se convertirán en caracteres alfanuméricos que podremos visualizar en la pantalla ya que cada letra de nuestro alfabeto está representada por una secuencia de unos y ceros.

Ahora bien, el código que debemos escribir en Arduino 0022 para esta primera actividad de comunicaciones es el siguiente:

Actividad 3.1: Código para que el robot diga "Hello World!".

```
void setup()
{
  delay(1000);
  Serial.begin(9600);
  Serial.println("Hello World!");
}

void loop()
{

}
```

La siguiente figura muestra cuál será la salida del programa. Esta se obtendrá luego de presionar el botón "Serial Monitor" que se encuentra a la derecha arriba y que está marcado en rojo en la siguiente imagen. Este botón muestra lo que se está enviando por un determinado puerto que, en el caso del ejemplo que estamos viendo, es el puerto COM13. Este puerto debe ser elegido en cada computadora por el programador según la situación.



Explicación del código de la actividad 3.1

```
  delay(1000);
```

La primera línea de código de este pequeño programa demorará la comunicación entre la computadora y la placa. De esta manera nos aseguramos de que la computadora mostrará por pantalla el mensaje que escribiremos a continuación.

```
  Serial.begin(9600);
```

Con esta línea de código establecemos la velocidad de la comunicación entre la placa y la computadora en 9600 baudios. Cuando abramos la pantalla en la que se muestra el mensaje que nos manda la placa, deberemos configurar la misma velocidad a la derecha abajo del cuadro de diálogo, como se ve en la figura que muestra la salida del programa.

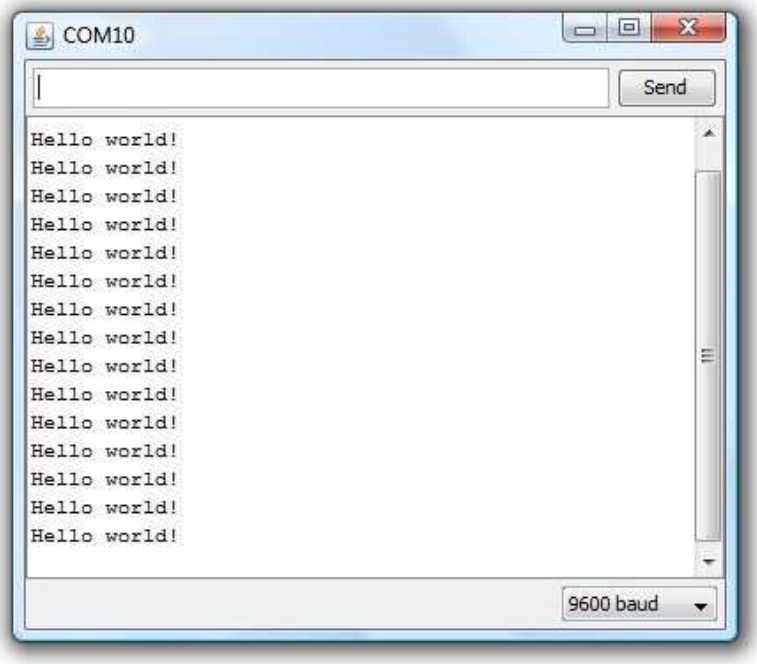
```
  Serial.println("Hello World!");
```

La siguiente línea de código que deberemos escribir es precisamente la que imprime por pantalla el mensaje que deseamos mostrar. Es importante notar que el mensaje en cuestión debe estar encerrado entre comillas y además, debe estar entre paréntesis ya que este es el parámetro que toma la función `println()`. Por último, cabe aclarar que el nombre de la función `println()` significa "imprimir línea", ya que ln es una abreviación de "line" (línea).

Algo a tener en cuenta en esta actividad es que la función `loop()` está completamente vacía. ¿Qué hubiese pasado si, en vez de escribir `Serial.println("Hello World!");` dentro de la función `setup()`, lo hubiésemos escrito dentro de la función `loop()`? Para saberlo, no hay mejor opción que probarlo, por lo tanto, vamos a cambiar de lugar esa línea de código. Luego, nos quedaría el siguiente programa:

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Hello World!");
}
```

Al correr este programa, nos encontramos con la siguiente salida:



Notaremos que el mensaje "Hello World!" se imprime por pantalla a gran velocidad. Para evitar esta situación agregaremos un delay(1000); como se muestra a continuación:

Ejemplo 3.1: Código para que el robot diga "Hello World!" reiteradas veces.

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Hello World!");
  delay(1000);
}
```

Esto lo haremos para que haya 1 segundo de espera entre una línea y otra. Conviene hacerlo de esta manera ya que, si no se pone una pausa entre dos mensajes consecutivos, puede que la comunicación se sature y empiece a funcionar mal por no poder procesar todos los mensajes que recibe y que debe mostrar.

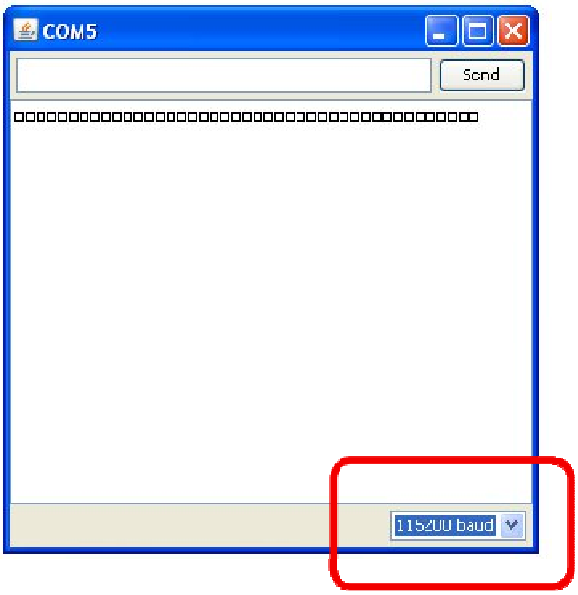
Esta pequeña actividad nos sirve para empezar a comunicar la placa con nuestra computadora. Al hacer programas más complejos, notaremos la necesidad de usar este tipo de comunicación, ya que un mensaje de la placa en un determinado momento, podrá darnos información muy útil acerca de la situación del sistema que estemos desarrollando. Basta pensar en los mensajes de error que devuelve una computadora cuando no puede resolver un determinado problema, para darse cuenta de una posible utilidad de este tipo de comunicación.

Funciones de comunicación del entorno Arduino 0022

Hasta ahora hemos utilizado las funciones println() y begin() de la librería Serial que nos provee el entorno Arduino 0022. Es interesante notar lo que pasaría si no coinciden las velocidades de transferencia configuradas por medio de la función begin() con la del entorno. Para ver lo que sucede, probemos el siguiente código dentro de la función setup():

```
void setup()
{
  Serial.begin(4800);
  Serial.println("Hello World!");
}

void loop()
{}
```

Luego configuramos una velocidad distinta en la salida del entorno. Para esto, apretamos el botón "Monitor" y cambiamos la velocidad de transferencia a la derecha debajo de la pantalla, como se muestra en la siguiente figura:



En este caso, configuramos la velocidad de transferencia del entorno al máximo posible y, en vez de mostrarnos el texto "Hello World!", nos mostró los cuadraditos que se ven en la pantalla. La salida puede variar dependiendo de la computadora y de la relación entre las dos velocidades, incluso puede ser que nos muestre el texto correctamente, pero la mayoría de las veces, muestra un mensaje no entendible y sin sentido.

Es interesante notar que, si vamos a la página de referencias de Arduino (http://arduino.cc/en/Reference/HomePage) y hacemos click en la última función de la columna "Functions" a la derecha abajo de la pantalla, sobre la única referencia de comunicación que hay, podremos ver todas las funciones de comunicación que provee el entorno Arduino 0022. Luego, cuando vemos el listado de funciones de comunicación que provee el entorno, veremos que existe una función end(). Esta función cierra la comunicación abierta por la función begin() que utilizamos. Sin embargo, no es necesario para nosotros cerrarla ya que no compartimos el puerto de comunicaciones.

También resulta interesante ver que hay dos formas de imprimir un mensaje por pantalla: la que usamos hasta ahora, println() y otra muy parecida que se llama print(). Al usarlas vamos a notar que, cuando usamos la función print() dentro de loop(), el texto que queremos mostrar por pantalla sale repetidas veces pero sin lo que se llama retorno de carro. Mientras que la función println() imprime el texto en una línea única, lo que hace que sea más legible la salida del programa en caso de tener bastante texto o datos para mostrar.

Funciones con parámetros

Antes de seguir avanzando con más actividades, es importante que nos detengamos para estudiar en profundidad un concepto fundamental de la programación: funciones. En matemática, una función se define como una relación entre dos conjuntos llamados dominio y codominio. El primero corresponde a los "datos de entrada" que recibirá la función y el segundo a los datos de salida que devolverá la misma después de modificar los datos de entrada. En programación la idea es similar, una función recibirá una serie de parámetros de entrada que serán utilizados y luego devolverá un resultado a partir de lo que haya hecho con los datos de entrada. Ahora bien, ¿cómo sabe la función cuáles son los datos de entrada y cuáles lo de salida? La respuesta es muy sencilla, lo sabe porque el programador lo decide cuando escribe la declaración de la función. Por ejemplo, una declaración ya conocida para nosotros es la siguiente: `void setup()` en esta podemos identificar tres cosas: `void`, que es una palabra reservada del sistema que indica que la función no devolverá ningún resultado; `setup`, que es el nombre de la función que usaremos; y, por último los (), que encierran los parámetros de entrada que recibirá la función. Luego, podemos ver que esta función en particular no recibe ningún parámetro y no devuelve ningún resultado.

Ahora bien, si quisiéramos escribir una función que tomara dos números enteros como parámetros y devolviera la resta de los dos como resultado, ¿cómo la declararíamos? Lo haríamos de la siguiente manera:

```
int substraction (int first_number, int second_number)
```

Esta línea de código es lo que llamamos la declaración de la función. En este caso, tenemos lo siguiente: el primer "`int`" nos indica que la función devolverá un número entero como resultado. Es importante notar que no le damos ningún nombre a este número. Luego escribiremos el nombre de la función, en este caso: `substraction` (resta en inglés). Este nombre lo elegimos nosotros y será el que utilizaremos para llamar a la función que escribamos cuando deseemos usarla. Si quisiéramos que nuestra función se llame `pepito` o `carlitos`, no habría ningún problema, salvo que probablemente podríamos olvidarnos de la funcionalidad de `pepito` o de `carlitos` y eso nos complicaría usarla más adelante. No así, `substraction`, es un nombre bien declarativo en cuanto a la razón por la que escribimos esta función. Por último, entre paréntesis, tenemos los números que servirán de entrada para esta función. Por un lado `int first_number` y, separado por una coma, `int second_number`. Otra vez, la palabra reservada `int` nos indica que trabajaremos con números enteros mientras que, `first_number` y `second_number`, son nombres que elegimos porque nos parecieron declarativos y útiles para entender lo que hará la función.

Una vez declarada, nos queda escribir el código que ejecutará la misma cada vez que sea llamada. Para esto, encerramos entre llaves las líneas de código que formarán el cuerpo de la función, de la siguiente manera:

```
int substraction (int first_number, int second_number)
{
        return first_number - second_number;
}
```

Parece no tener mucho sentido escribir una función que nos devuelva la resta entre dos números, pero lo importante de este ejemplo reside precisamente en su sencillez. Es importante notar dos cosas: `return` es una palabra reservada del sistema que indica que lo que se escriba a su derecha, será lo que devuelva la función que escribimos. Por otro lado, vemos que `first_number` y `second_number` son utilizados como variables dentro de la función de tal manera que podemos hacer operaciones con ellas. En nuestra función de ejemplo, podemos restarlas como si fueran números ya que, de hecho, ambos nombres de parámetros representan números enteros.

Por último, antes de pasar a un ejemplo que sirva para aclarar las dudas que queden con respecto a este tema, nos falta saber cómo usar esta función que escribimos. Esto nos resultará muy fácil ya que solo tenemos que nombrarla pasándole como parámetros los números que deseamos restar. Por ejemplo, si quisiéramos restar los números 10 y 5, tendríamos que llamar a la función `substraction` de la siguiente manera:

```
substraction(10, 5);
```

Es importante notar que ya no usamos las palabras `int` en ningún lugar, ni antes del nombre de la función, ni antes de los parámetros. A continuación hay una tabla para repasar la forma de llamar una función con distintos parámetros.

Valor primer_numero	Valor segundo_numero	Llamada a la función
10	5	`substraction (10, 5);`
20	4	
3	3	
5	10	

Ahora sí, veamos un ejemplo con el cual podamos poner en práctica lo aprendido hasta este punto. Pensemos en una función que calcule el área de una circunferencia. La función, dado el radio del círculo, debería realizar el siguiente cálculo:

Área del círculo = pi x radio^2

Lo primero que haremos será declarar la función. Para esto, tenemos que pensar en tres cosas: (1) qué nombre queremos que tenga la función; (2) cuáles serán sus parámetros de entrada y (3) cuál será la salida que produzca. En cuanto al nombre de la función, podría ser `circle_area`, ya que este nombre es bien declarativo y al leerlo nos da suficiente información acerca de lo que hace la función. Luego, como parámetros de entrada, nos va a alcanzar con uno solo que represente el radio del círculo y que sea un número real (float). Y, por último, la salida será otro número real. Luego, la declaración de la función será la siguiente:

```
float circle_area(float radio)
```

Ahora, pensemos en las líneas de código que se encargarán de realizar el cálculo que nos interesa. Para empezar, necesitaremos una variable que represente al número pi. Esta variable tendrá que ser declarada e inicializada. Esto lo haremos de la siguiente manera:

```
float circle_area(float radio)
{
        float pi = 3.14;
}
```

Luego, usando los números pi y radio, calcularemos el área del círculo en cuestión para devolverlo como resultado de la función que escribimos. Esto lo haremos de la siguiente manera:

```
float circle_area(float radio)
{
        float pi = 3.14;
        float area;
        area = pi * radio * radio;
        return area;
}
```

Ya tenemos escrita una función que calcula el área de un círculo pero, ¿cómo la usamos? Una vez escrita la función, solo basta llamarla. En este caso, dado que la función devuelve un valor numérico, la llamaremos de tal manera que podamos ver el valor calculado. Para eso, pasaremos a nuestra función `circle_area (radio)` como parámetro de la función `Serial.println()` de la siguiente manera:

```
void setup()
{
        Serial.begin(9600);
        Serial.println("Circle area: ");
        Serial.print(circle_area(2));
}
```

Algo importante que hay que tener en cuenta es que el "2" que le pasamos como parámetro a la función `circle_area(radio)`, representa el radio de la circunferencia a la que le queremos calcular el área. Este valor, al ser el parámetro de la función, lo elegimos nosotros.
El código completo que habría que escribir en nuestro entorno de programación quedaría de la siguiente manera:

Ejemplo 3.2: función que devuelve un número real como resultado.

```
float circle_area(float radio)
{
        float pi = 3.14;
        float area;
        area = pi * radio * radio;
        return area;
}

void setup()
{
        delay(1000);
        Serial.begin(9600);
        Serial.println("Circle area: ");
        Serial.print(circle_area(2));
}

void loop ()
{}
```

Por último, veamos otra forma de escribir la misma función que calcula el área de un círculo. En este caso, vamos a hacer que nuestra nueva función imprima ella misma por pantalla el resultado obtenido en vez de simplemente devolvernos un valor. Para esto, vamos a declarar nuevamente la función usando la palabra reservada `void`.

```
void show_area(float radio)
```

Ahora, al escribir el cuerpo de la función, tenemos que hacer los mismos cálculos que antes y luego mostrarlos por pantalla, lo que antes hicimos en la función `setup()`.

```
void show_area (float radio)
{
  float pi = 3.14;
  float area;
  area = pi * radio * radio;
  Serial.begin(9600);
  Serial.println("Circle area: ");
  Serial.print(area);
}
```

Es importante notar las siguientes características de este código:
1- El cálculo del área se realiza de la misma manera que antes.
2- Las funciones que usamos para mostrar por pantalla el resultado, ahora están dentro del cuerpo de esta función.
3- Cuando imprimimos por pantalla el resultado del cálculo realizado, simplemente escribimos `Serial.print(area);` ya que la variable `area` contiene el resultado de lo calculado más arriba.
4- En ningún lugar de este código le decimos a la función cuál es el radio del círculo en cuestión ya que eso lo haremos cuando llamemos a la función. Esto lo haremos desde `setup()` de la siguiente manera:

```
void setup()
{
  show_area(4);
}
```

Con lo cual, el código completo quedará como se muestra a continuación:

Ejemplo 3.3: función que calcula el area de un círculo.

```
void show_area (float radio)
{
  float pi = 3.14;
  float area;
  area = pi * radio * radio;
  Serial.begin(9600);
  Serial.println("Circle area: ");
  Serial.print(area);
}

void setup()
{
  delay(1000);
  show_area(4);
}

void loop ()
{}
```

En este último caso, la salida del programa será como se muestra a continuación:

Actividad 3.2: hacer que el robot diga qué motor está usando

En esta actividad queremos hacer que el robot envíe un mensaje indicando qué motor está usando en un determinado momento. La idea principal es que, cada vez que le asignamos dirección y potencia a un motor, también imprimamos por pantalla los valores que utilizamos.

A continuación, se muestra el código que se deberá escribir dentro en el entorno Arduino:

Actividad 3.2: el robot comunica información acerca de sus motores.

```
void setup()
{
  motor1.setClockwise(false);
  Serial.begin(9600);
  delay(1000);

  motor0.setSpeed(50.0);
  Serial.println("Motor: 0");
  Serial.println("Direction: Forward");
  Serial.println("Speed: 50");
  delay(1000);

  motor1.setSpeed(100.0);
  Serial.println("Motor: 1");
  Serial.println("Direction: Reverse");
  Serial.println("Speed: 100");
  delay(1000);

  motor0.setSpeed(0);
  motor1.setSpeed(0);
}

void loop()
{}
```

Explicación del código de la actividad 3.2

```
motor1.setClockwise(false);
```

Dentro de la función setup(), lo primero que hacemos es setear los parámetros que necesitamos. Con esta línea de código le asignamos al motor 1 una dirección de giro opuesta a la que tiene establecida. Esto lo hacemos porque las posiciones en las que están montados los motores en el robot Multiplo N6 hacen que cada uno avance en una dirección opuesta.

```
Serial.begin(9600);
```

Con esta línea de código establecemos la velocidad de la comunicación entre el robot y la computadora en 9600 baudios. Cuando abramos la pantalla en la que se muestra el mensaje que nos manda la placa, deberemos configurar la misma velocidad a la derecha abajo del cuadro de diálogo, como se ve en la figura que muestra la salida del programa.

```
motor0.setSpeed(50.0);
```

Con esta línea de código le asignamos dirección y potencia al motor 0. Este es el valor que queremos mostrar por pantalla junto a la dirección, la que escribiremos en inglés por convención.

```
Serial.println("Motor: 0");
Serial.println("Direction: Forward");
Serial.println("Speed: 50");
```

Estas líneas son las que le indican al robot qué mensaje va a enviar. Con la primera de ellas mostramos el número de motor que está en movimiento, con la segunda la dirección que le asignamos a dicho motor (Forward) y, con la tercera, la velocidad.

```
delay(1000);
```

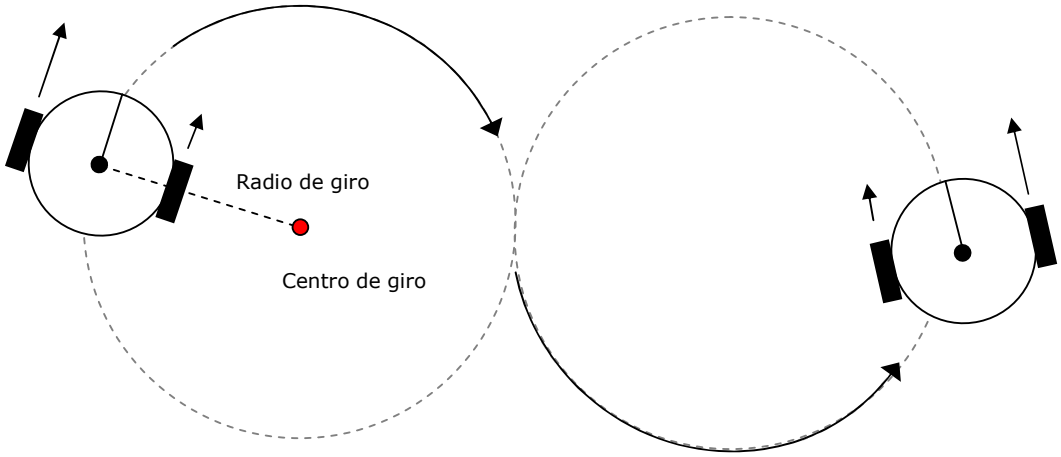
Para que se pueda leer mejor la información que corresponde a cada motor, establecemos un tiempo de un segundo entre cada bloque de código. Así podremos leer con más facilidad los mensajes mostrados en la pantalla.

```
motor0.setSpeed(0);
motor1.setSpeed(0);
```

Por último, asignamos potencia cero a cada motor para que el robot se detenga.

Actividad 3.3: hacer que el robot comunique su trayectoria

En esta actividad queremos que el robot nos avise cada vez que cambie de trayectoria. Para realizarla, vamos a escribir el código necesario para que dibuje un "8" como el que se muestra en la siguiente figura.



Luego, haremos que el robot imprima por pantalla un mensaje en el cual nos indicará hacia qué lado está girando. Los mensajes del robot se verán como se muestra a continuación.

A continuación, se muestra el código que se deberá escribir dentro de la función setup():

Actividad 3.3: Código para que el robot dibuje un ocho comunicando su trayectoria.

```
void setup()
{
  motor1.setClockwise(false);
  Serial.begin(9600);
  delay(1000);

  motor0.setSpeed(80.0);
  motor1.setSpeed(40.0);
  Serial.println("Direction: Right");
  delay(3000);

  motor0.setSpeed(40.0);
  motor1.setSpeed(80.0);
  Serial.println("Direction: Left");
  delay(3000);

  motor0.brake();
  motor1.brake();
  Serial.println("Brake");
}

void loop()
{}
```

Explicación del código de la actividad 3.3

```
motor0.setSpeed(80.0);
motor1.setSpeed(40.0);
```

Con estas 2 líneas de código el robot girará hacia su derecha. Esta parte del programa solo afecta al desplazamiento del robot y no a la comunicación entre el mismo y la computadora.

```
Serial.println("Direction: Right");
```

Esta es la línea principal del programa ya que es la que permite imprimir por pantalla el mensaje que manda el robot. En ella indicamos la dirección en la que se mueve el robot que en este caso es hacia la derecha (Right).

```
delay(3000);
```

Es importante darle tiempo suficiente al robot para que dibuje la trayectoria deseada. Por otro lado, esta orden la escribimos a continuación del mensaje para que este aparezca mientras el robot está girando.

```
motor0.brake();
motor1.brake();
Serial.println("Brake");
```

Por último, frenamos cada motor para que el robot se detenga. Luego, queremos que nos avise que así lo hizo.

Actividades de integración

1- Escribir y testear una función que haga que el robot gire sobre su propio eje desde cero hasta su velocidad máxima y de vuelta hasta cero mostrando por pantalla la velocidad de los motores.
2- Escribir y testear una función que tres cuadrados de distinto tamaño uno dentro del otro y que avise por pantalla qué lado de qué cuadrado está dibujando.

Preguntas para investigar

1- Investigar en qué otros dispositivos se utiliza este tipo de comunicación.
2- ¿Dependen estos dispositivos o juguetes de un procesador central para manejar las comunicaciones como en el caso de los robots?
3- Enumerar y describir otros tipos de comunicación.
4- Clasificar los distintos tipos de comunicación del punto 3.

4. Sensores

Nivel: básico/intermedio.

Objetivos

- Que los alumnos comprendan la interacción existente entre el robot y el medio.
- Que sean capaces de realizar un programa que verifique la interacción.

Necesidad de utilizar sensores

Hasta este momento, los programas que realizamos para nuestro robot, no le proveían la posibilidad de tomar decisiones. Sin embargo, esta característica va a cambiar desde que empecemos a utilizar sensores. Estos le permitirán al robot decidir qué hacer frente a distintos estímulos que reciba del exterior.
La idea principal que hay que tener en cuenta a la hora de utilizar sensores es que estos simplemente guardarán un valor en la memoria. Luego, seremos nosotros los encargados de decirle al robot qué tendrá que hacer con ese valor. Por ejemplo, es muy común que se monte un sensor en la parte frontal del robot. Este tomará valores del medio en el que se encuentra. Estos valores, se traducirán a números reales (float) y serán guardados en la memoria de la misma manera como se guardan las variables. Una vez almacenados en la memoria, estos valores podrán ser consultados para cambiar el curso del programa utilizándolos dentro de estructuras de programación como el while, if, etc..

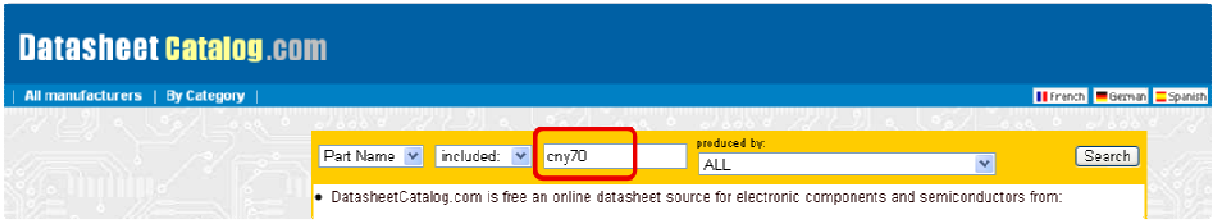
Algo de teoría de sensores infrarrojos (IR)

Los sensores IR funcionan emitiendo un haz de luz infrarrojo invisible. Este, luego de rebotar contra un objeto, vuelve al sensor, el cual capta la intensidad del regreso. Esta intensidad es convertida en un número real con el cual trabajaremos en las actividades que proponemos a continuación.

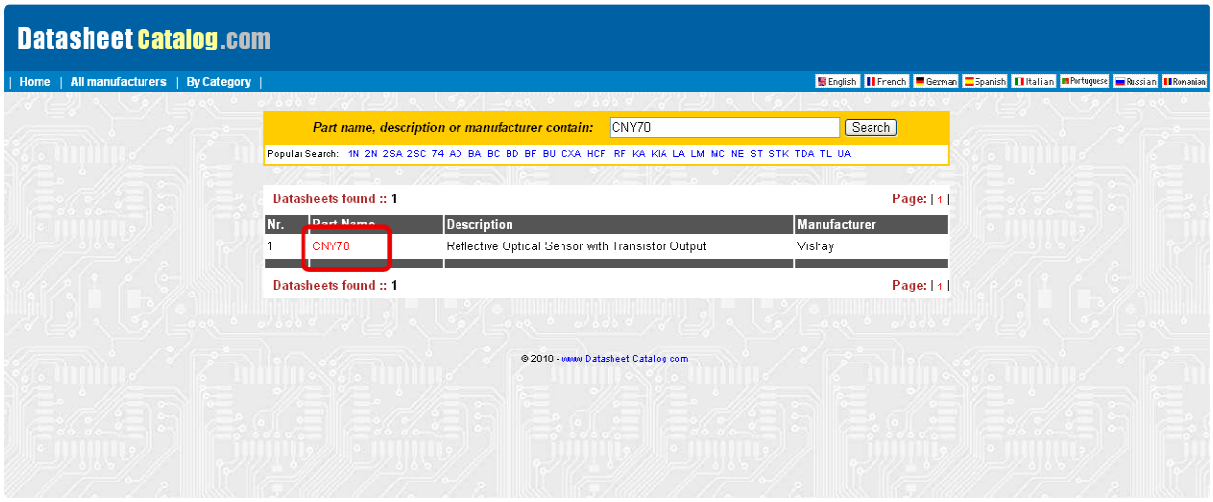
Los sensores IR que utilizan los robots Multiplo N6 y N10 son montados en RobotGroup para que puedan ser usados sin problemas. Estos están compuestos por tres partes: (1) una pequeña placa sobre la que se monta el sensor, (2) el sensor propiamente dicho y (3) la ficha con la cual se lo conecta a través de un cable con la placa. En cuanto al sensor, se pueden ver sus características técnicas en las hojas de datos que provee su fabricante. Para descargar la hoja de datos, podemos hacerlo desde el siguiente sitio web:

http://www.datasheetcatalog.com/

Una vez en el sitio, escribimos el nombre del componente en el cuadro de búsqueda, como se muestra en la siguiente figura. En nuestro caso, escribiremos el nombre del modelo del sensor que es el CNY70.



Luego de presionar el botón "search", el sitio nos provee la posibilidad de descargar dicha hoja de datos. Para eso, seleccionamos la opción que corresponde al sensor que ingresamos, como se muestra en la siguiente figura:



Ahora bien, tomemos el sensor junto con su hoja de datos y veamos un poco qué tenemos. Si uno mira bien de cerca al sensor IR que utilizan nuestros robots, podrá ver que en el centro del cubo negro tiene dos círculos pequeños uno al lado del otro. Estos están separados uno del otro por una distancia de 2,8 mm y ambos apuntan hacia el mismo lugar. Uno de ellos es un diodo que funciona como emisor y el otro es un fototransistor que funciona como receptor. El diodo es el encargado de emitir un haz de luz infrarrojo. Este haz de luz rebotará contra cualquier objeto que se le interponga. Luego, el reflejo producido por el robote será captado por el fototransistor. Cabe aclarar que no todos los sensores IR emiten un haz de luz. Los que no lo hacen son llamados pasivos, mientras que los que utilizan nuestros robots son llamados activos por poseer esta característica.

La salida del fototransistor es analógica y oscila entre 0 y 5 Voltios. Luego, el valor captado será convertido a un valor digital por el conversor del procesador y será almacenado en un registro de 8 bits. Al llamar a la función `analogRead(n)` estamos consultando el valor guardado en dicho registro. En el caso de nuestros robots, la "n" que toma como parámetro la función `analogRead(n)`, representa la entrada a la que está conectado dicho sensor. En el robot Multiplo N6, podemos conectar los cables que unen los sensores con la placa en las entradas nombradas como S0, S1, S2, S3, S4 y S5. Pero tendremos que saber cuáles de estas utilizamos.

Actividad 4.1: testeo de un sensor IR

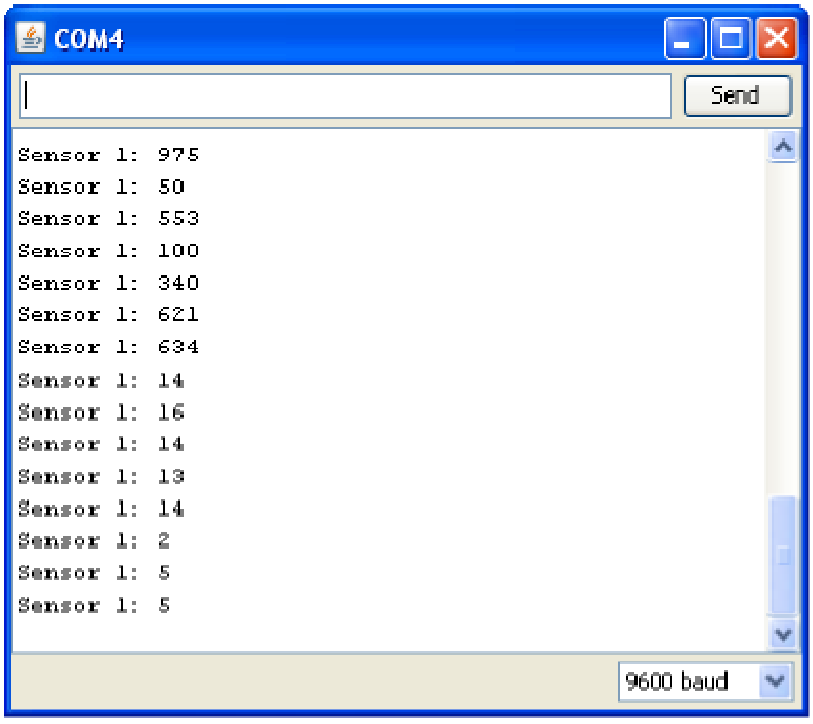
En esta actividad vamos a probar el funcionamiento de un sensor infrarrojo (IR). Para ello, mostraremos en la pantalla los valores que capta el mismo en distintas situaciones. A medida que vaya avanzando en las pruebas, le recomendamos completar la tabla que se encuentra en esta actividad.

A continuación se escribe el código necesario para realizar la actividad:

Actividad 4.1: testeo de un sensor.

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print("Sensor 1: ");
  Serial.println(analogRead(1));

  delay(500);
}
```

Luego de bajar el programa, podremos ver que el robot nos manda mensajes como los que se muestran a continuación. Para verlos, presionamos el botón "Serial Monitor" que se encuentra a la derecha arriba en el entorno Arduino 0022.



Los valores más elevados corresponden a situaciones en las que el sensor detectó algo a corta distancia, mientras que los valores más bajos son los captados cuando no había nada cerca del sensor.

Explicación del código de la actividad 4.1

```
Serial.begin(9600);
```

Con esta línea de código establecemos la velocidad de la comunicación entre el robot y la computadora en 9600 baudios. Cuando abramos la pantalla en la que se muestra el mensaje que nos manda la placa, deberemos configurar la misma velocidad a la derecha abajo del cuadro de diálogo. Es importante notar que esta orden la escribimos dentro de la función `setup()` debido a que solamente la queremos setear una vez ya que luego trabajaremos siempre con la misma velocidad de comunicación.

```
Serial.print("Sensor 1: ");
```

Con esta línea de código simplemente mostramos por pantalla el mensaje "`Sensor 1: `", la cual nos va a ayudar a mostrar por pantalla información más clara

```
Serial.println(analogRead(1));
```

Esta es la línea de código más importante del programa. Sabemos que la función `Serial.println()` nos mostrará por pantalla el mensaje que le pasemos como parámetro entre los paréntesis. Lo que haremos entonces, es decirle que muestre el valor detectado por el sensor número 1. Para eso, usamos la función `analogRead(número_sensor)` la cual devuelve el valor captado por el sensor indicado. Los sensores los llamamos por número como hacemos con los motores y, este número, depende del pin al que esté conectado.

```
delay(500);
```

Es importante darle un tiempo al procesador entre una ejecución y otra. De lo contrario, es muy probable que no pueda mostrar por pantalla correctamente los datos que queremos debido a la velocidad a la que trabaja. Para eso, usando la orden delay(500) hacemos que el procesador espere medio segundo (500 milisegundos) antes de volver a ejecutar una orden.

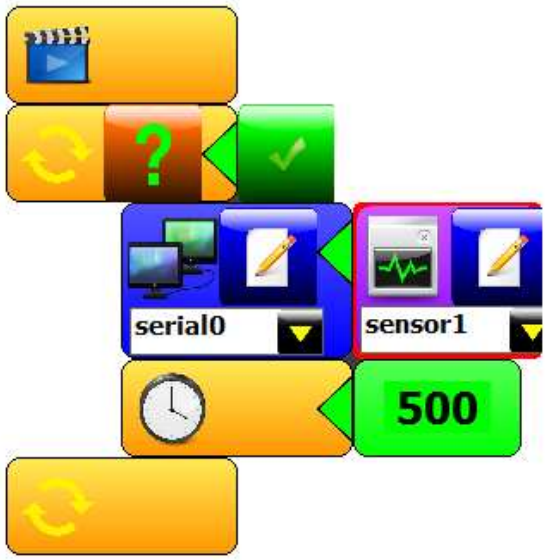
Tabla de valores

Completar la siguiente tabla con los valores captados por el sensor seleccionado.

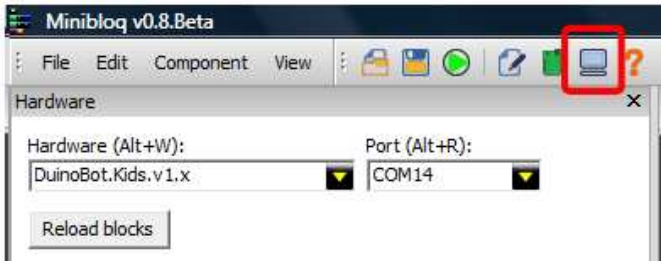
Situación del sensor:	Valor medido:
No hay nada frente al sensor.	
Hay un objeto a 10 cm. de distancia.	
Hay un objeto opaco pegado al sensor.	
Hay un objeto brillante pegado al sensor.	

Código en Minibloq de la actividad 4.1

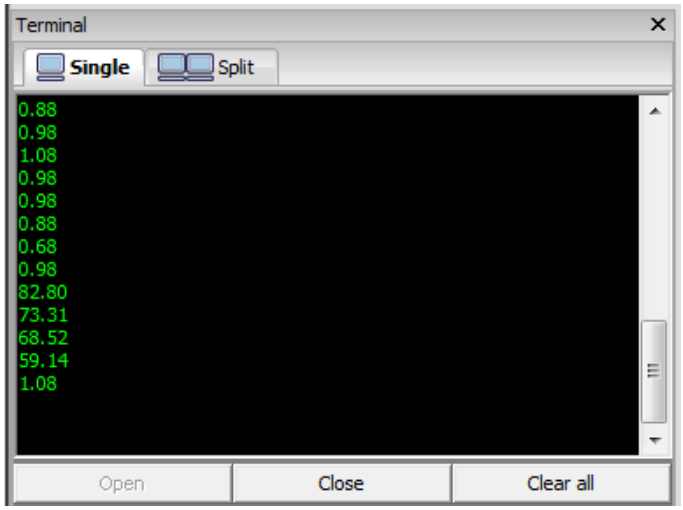
Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Luego, presionamos el botón "Terminal" en el entorno lo que nos mostrará la pantalla de comunicación entre el robot y la computadora.



Una vez abierta la pantalla del terminal, hacemos clic en el botón "Open" para que empiece la comunicación.



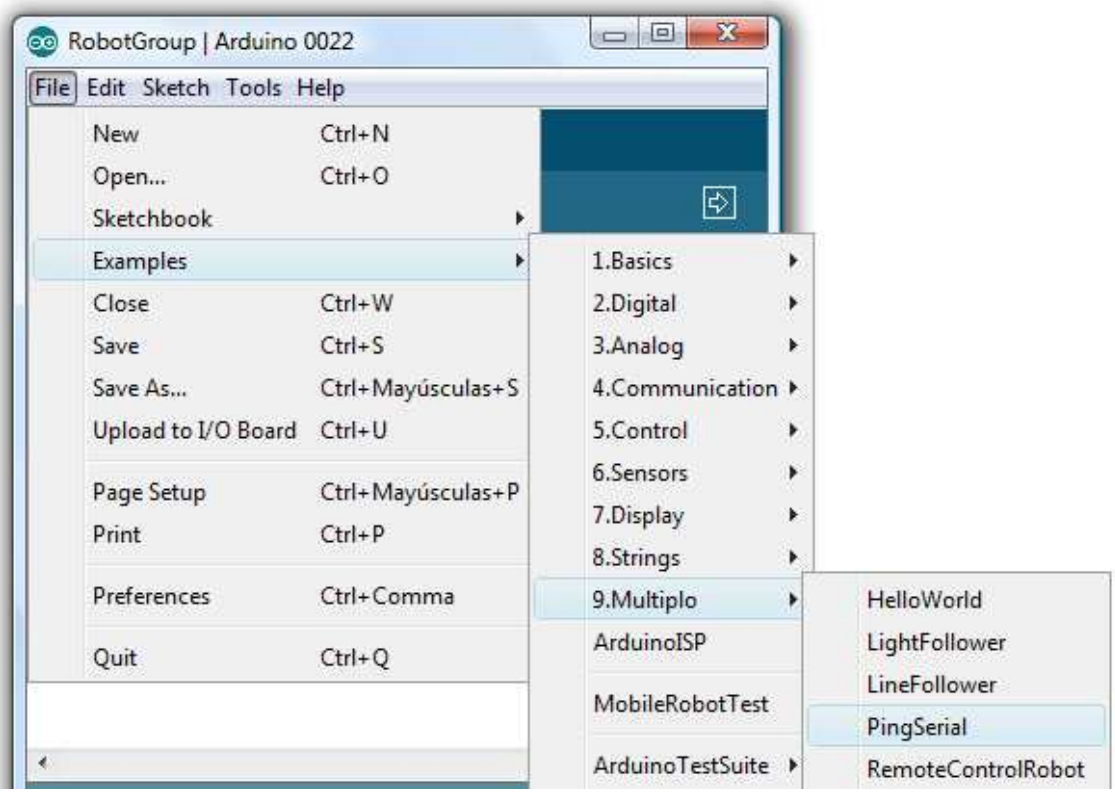
Algo de teoría de sensores ultrasónicos

Los sensores ultrasónicos funcionan emitiendo pulsos de sonido por medio de un emisor y recibiendo el rebote del sonido emitido en un receptor. Luego de recibir el rebote del sonido emitido, se mide el tiempo que transcurrió entre la emisión y la recepción. De esta manera se calcula la distancia hacia el objeto en el que rebotó la señal emitida.

Este tipo de sensor es muy utilizado en Robótica especialmente cuando se utilizan robots para competencias infantiles. Entre las ventajas que posee sobre otros sensores, podemos nombrar que interferencias lumínicas (como el flash de una cámara de fotos) no lo afectan. Además, suele ser un sensor muy preciso en sus mediciones.
También hay que aclarar que, en los robots Multiplo fabricados por RobotGroup, este sensor no puede usarse junto a los sensores IR de 38KHz, estos últimos son los que se utilizan, por ejemplo, para enviar señales al robot a través de un control remoto de televisión.

Ejemplo 4.1: medición de distancias usando un sensor ultrasónico

En caso de querer utilizar un sensor ultrasónico, el entorno de programación provee un ejemplo con el que, conectando el sensor a la entrada S1, puede medirse la distancia hacia distintos objetos. Para abrir el código del ejemplo hay que presionar el botón "File", luego "Examples", "9.Multiplo" y, por último, seleccionar "PingSerial", como se muestra en la siguiente figura:



A continuación se escribe el código necesario para realizar la actividad:

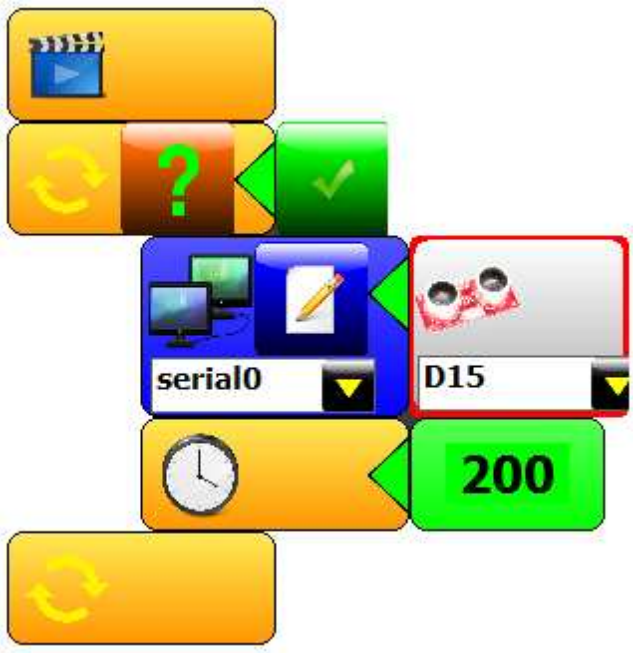
Ejemplo 4.1: medición de distancias usando un sensor ultrasónico.

```
#include <Ping.h>

PingSensor ping(15); //S1 input.

void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.println(ping.measureCM());
  delay(200);
}
```

Para ejecutar este código hay que presionar el botón "Serial Monitor" que muestra por pantalla la comunicación entre el robot y la computadora. Hay que asegurarse de que la velocidad de transferencia sea de 115200 baudios, como lo indica el código.

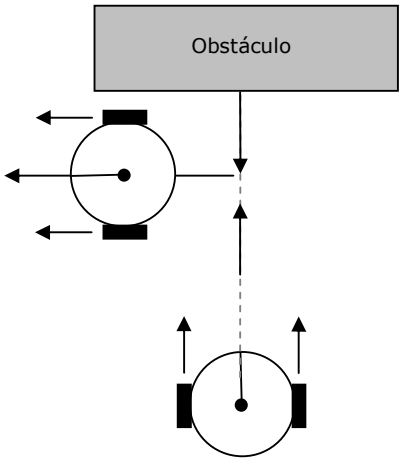
Código en Minibloq del ejemplo 4.1

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Actividad 4.2: avanzar hasta detectar un obstáculo

¿Cómo haría un robot para esquivar un obstáculo usando un único sensor montado en su parte frontal tal como lo muestra la siguiente figura? Para realizar esta actividad, habrá que montar uno de los sensores en la parte delantera del robot y el mismo tendrá que estar apuntando hacia delante.



A continuación, se muestra el código que se deberá escribir en el entorno de programación para realizar esta actividad:

Actividad 4.2: hacer que el robot avance hasta detectar un obstáculo.

```
void setup()
{
  motor0.setClockwise(false);
}

void loop()
{
    motor0.setSpeed(40);
    motor1.setSpeed(40);

  while(analogRead(1) > 500)
  {
    motor0.setSpeed(-40);
    motor1.setSpeed(-40);
    delay(250);

    motor0.setSpeed(40);
    motor1.setSpeed(-40);
    delay(250);
  }
}
```

Explicación del código de la actividad 4.2

```
void setup()
{
  motor0.setClockwise(false);
}
```

En la función `setup()` asignamos la dirección del motor 0 como contrario a las agujas del reloj.

```
    motor0.setSpeed(40);
    motor1.setSpeed(40);
```

Para empezar, le damos al robot las órdenes necesarias para que avance en línea recta. Este será su comportamiento natural, el cual modificará en caso de encontrar un obstáculo.

```
  while(analogRead(1) > 500)
```

Esta línea de código es la que definirá el comportamiento del robot. Mientras el sensor 1 detecte valores menores a 500, el robot no ejecutará las órdenes encerradas entre las llaves de este ciclo (while) y simplemente avanzará en línea recta debido a que es lo que establecimos en las líneas anteriores. En cambio, cuando el sensor detecte un obstáculo, el valor que devolverá la función analogRead() será mayor a 500 y esto hará que la condición dentro del while sea verdadera, con lo cual, se ejecutará todo lo que esté entre llaves a continuación de la condición.

```
    motor0.setSpeed(-40);
    motor1.setSpeed(-40);
    delay(250);
```

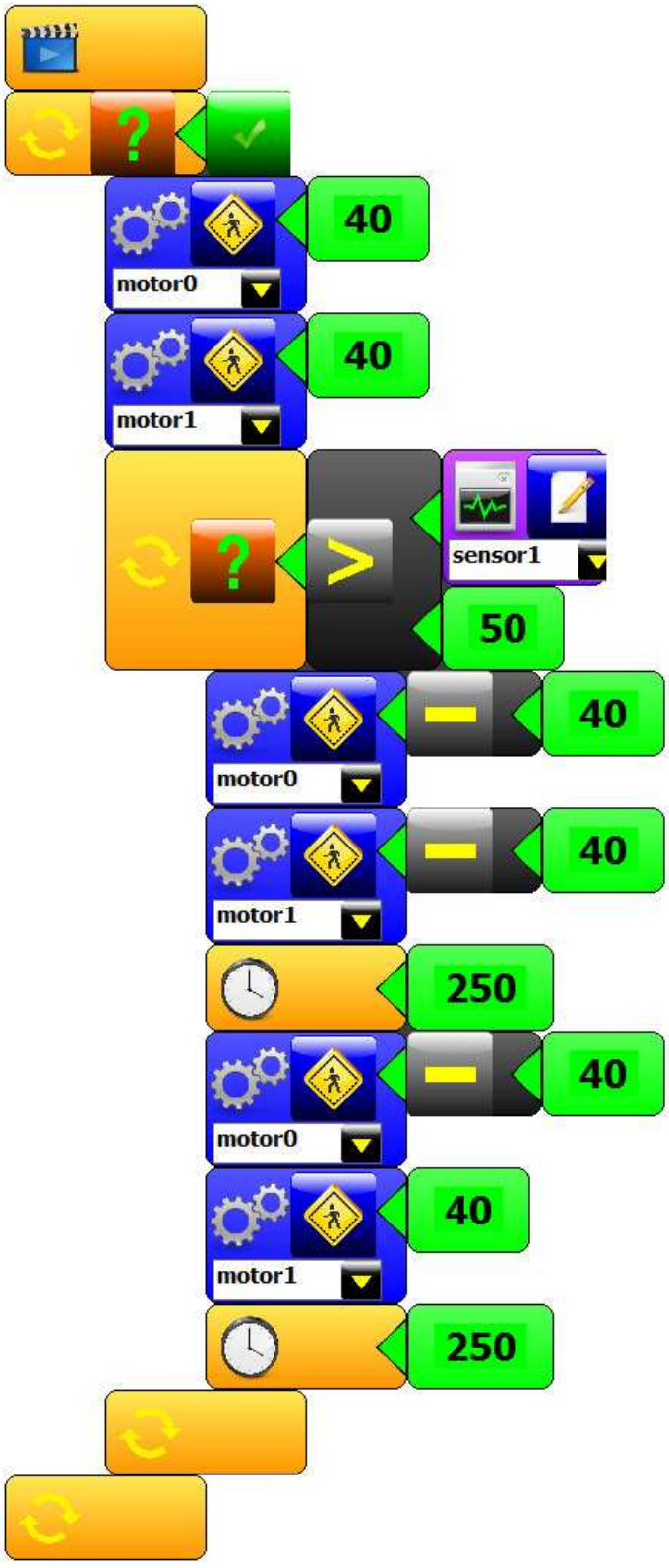
Con estas tres líneas hacemos que el robot retroceda por un cuarto de segundo. Tanto el tiempo como la velocidad pueden ser modificados según la superficie en la que estemos utilizando el robot como así también la carga de las pilas. En caso de realizar esta actividad sobre una superficie en la que no traccione bien es probable que haya que asignarles mayor potencia a los motores o mayor tiempo.

```
    motor0.setSpeed(40);
    motor1.setSpeed(-40);
    delay(250);
```

Por último, hacemos que el robot haga un pequeño giro sobre su eje para que luego siga avanzando en una dirección distinta de la que tenía.

Código en Minibloq de la actividad 4.2

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Actividad 4.3: avanzar hasta detectar un obstáculo, código alternativo

El código que acabo de escribir realiza la actividad correctamente, pero es solo una de las posibilidades que tenemos. A continuación, describimos otra forma de hacer que el robot avance hasta detectar un obstáculo que tal vez resulte más natural de pensar.

A continuación, se muestra el código que se deberá escribir en el entorno de programación para realizar esta actividad:

Actividad 4.3: hacer que el robot avance hasta detectar un obstáculo, código alternativo.

```
void setup()
{
  motor0.setClockwise(false);

  motor0.setSpeed(40);
  motor1.setSpeed(40);
}

void loop()
{
  if(analogRead(1) > 500)
  {
    motor0.setSpeed(-40);
    motor1.setSpeed(-40);
    delay(250);

    motor0.setSpeed(40);
    motor1.setSpeed(-40);
    delay(250);
  }
  else
  {
    motor0.setSpeed(40);
    motor1.setSpeed(40);
  }
}
```

Explicación del código de la actividad 4.3

```
void setup()
{
  motor0.setClockwise(false);

  motor0.setSpeed(40);
  motor1.setSpeed(40);
}
```

En la función `setup()` asignamos la dirección del motor 0 como contrario a las agujas del reloj. Pero además, seteamos la dirección inicial que tendrán los motores.

```
  if(analogRead(1) > 500)
```

Los motores empiezan avanzando hacia adelante con la misma velocidad. Pero al entrar en el ciclo loop(), lo primero que hace el programa es preguntar por un posible obstáculo. Esta vez, lo hace utilizando la sentencia if en vez de while.

```
    motor0.setSpeed(-40);
    motor1.setSpeed(-40);
    delay(250);

    motor0.setSpeed(40);
    motor1.setSpeed(-40);
    delay(250);
```

Si la condición de la sentencia if es verdadera, el robot retrocederá y girará.

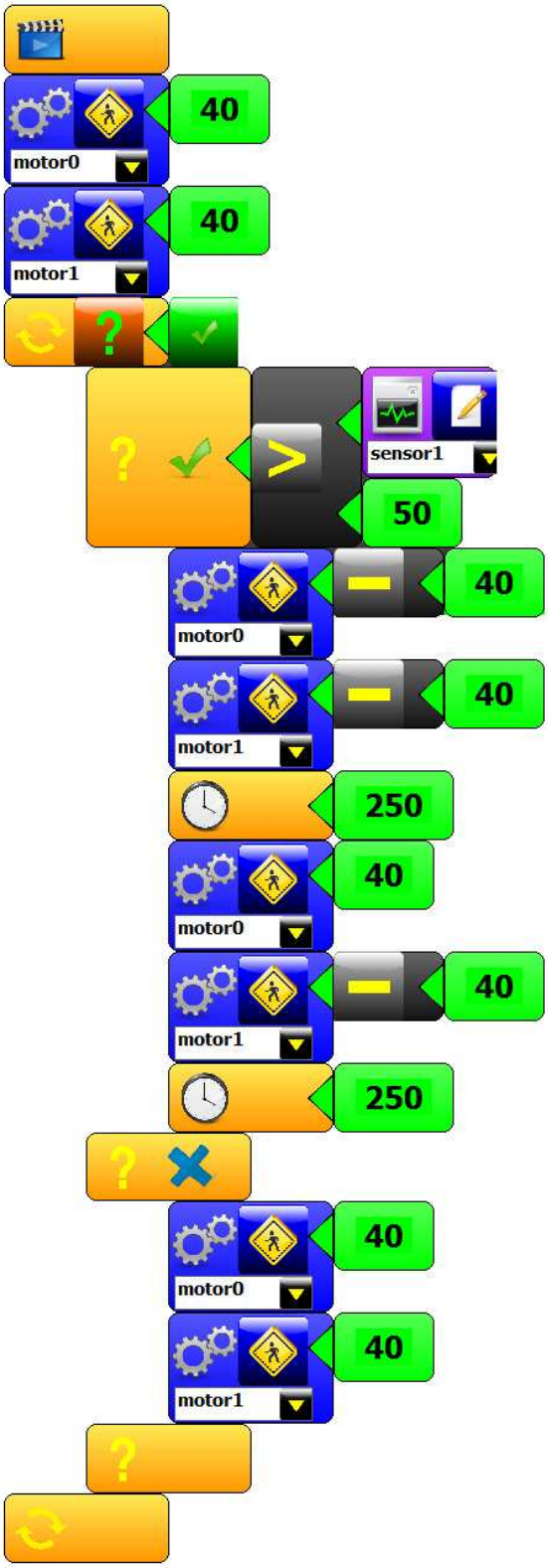
```
  else
  {
    motor0.setSpeed(40);
    motor1.setSpeed(40);
  }
```

Pero si la condición es falsa, el robot seteará sus motores con la misma velocidad para seguir avanzando en línea recta. Sin este "else", una vez que el robot detecte un obstáculo, no podrá volver a avanzar hacia adelante, ya que no volvería a asignarle velocidad a los motores.

Pensar el código de esta manera, utilizando el condicional if en vez del while, puede resultar más natural ya que, al escribir la alternativa de comportamiento dentro de las llaves del "else" nos permite pensar el ciclo loop como un conjunto de operaciones disjuntas. Esto es, el robot retrocede cuando el sensor no detecta un obstáculo o avanza si pasa lo opuesto. Cuando utilizamos la sentencia while tuvimos que agregar el bloque de código para que el robot avance en línea recta en cada ejecución de la función loop().

Código en Minibloq de la actividad 4.3

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Operadores lógicos en programación

Los operados lógicos los utilizamos cuando queremos combinar varias condiciones dentro de una misma estructura de control. Por ejemplo, en el caso en que queramos que se ejecute un determinado bloque de código solo si pasan dos cosas al mismo tiempo, utilizaremos un operador lógico que signifique "y". En al caso que queramos que se ejecute un bloque al suceder una de dos condiciones, utilizaremos un operador lógico que signifique "o". También podremos pedirle al procesador que ejecute ciertas órdenes cuando "no" sea cierta una condición. A continuación se muestra una tabla con algunos operadores lógicos y su significado.

Operador	Significado	Uso
&&	Y lógico	Deberán ser verdaderas todas las condiciones separadas por este operador.
||	O lógico	Deberá ser cierta sólo una de las condiciones separadas por este operador.
!	Negación	Deberá ser falsa la condición que se encuentra a continuación de este operador.

Veamos un ejemplo utilizando el código que hace que el robot retroceda y gire al encontrar un obstáculo:

```
while(analogRead(1) > 500)
{
  motor0.setSpeed(-40);
  motor1.setSpeed(-40);
  delay(250);

  motor0.setSpeed(40);
  motor1.setSpeed(-40);
  delay(250);
}
```

Cambiemos la condición para utilizar el operador lógico que la niega.

```
while(!analogRead(1) < 500)
```

En este caso, además de poner la negación delante de la condición, también cambiamos el signo mayor por uno que signifique menos. De esta manera, estaríamos diciendo: "Mientras (while) el sensor 1 NO detecte un valor menor a 100, ejecutá el código que está entre llaves".

Otra posibilidad es escribir algo como lo que sigue:

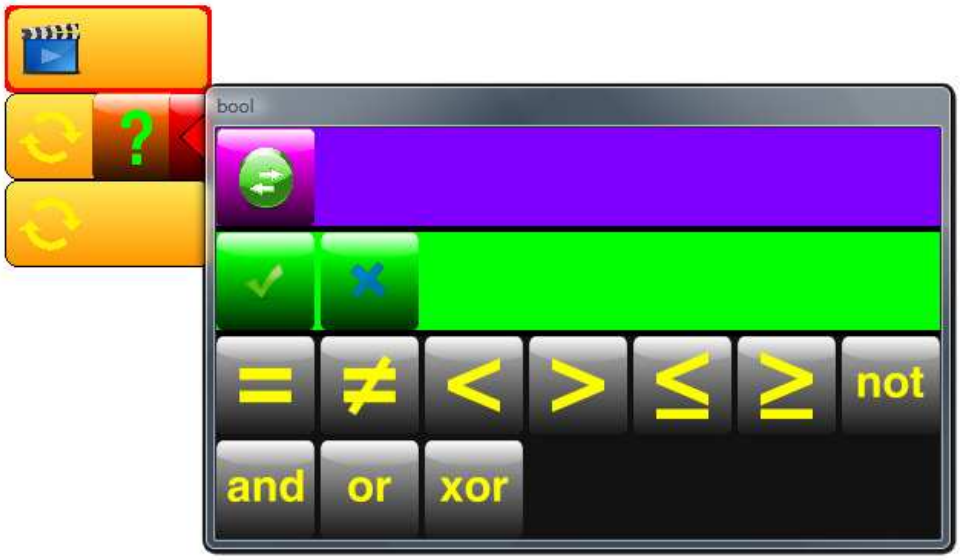
```
while((analogRead(1) > 500) && (analogRead(0) > 500))
```

En este caso le estaríamos diciendo al procesador algo del estilo: "Mientras el sensor 1 detecte un valor mayor a 100 Y el sensor 0 también detecte un valor mayor a 100, ejecutá el código que está a continuación".

Más información acerca de los operadores lógicos soportados por el entorno Arduino 0022 podremos encontrarla en la página de referencias bajo el título "Boolean Operators" en el sitio www.arduino.cc. Ahora, hagamos un ejemplo donde podamos poner en práctica algunos de estos operadores lógicos.

Operadores lógicos en Minibloq

En el entorno Minibloq también disponemos de operadores lógicos. Estos se usan dentro de estructuras de control al igual que otros operadores como los matemáticos o los booleanos. En la imagen que se muestra a continuación, podemos ver la lista de operadores que se despliega junto a un ciclo como el while. Los operadores lógicos están al final y son "not", "and", "or" y "xor".



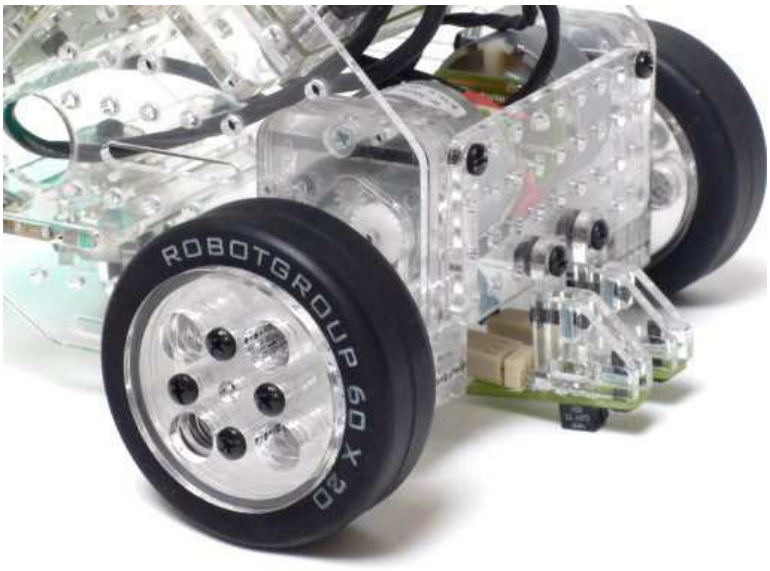
Actividad 4.4: hacer que el robot siga una línea

Objetivo: Que el alumno aplique distintas configuraciones para que el robot resuelva problemas de distintas clases utilizando los sensores IR.

En esta actividad vamos a programar el robot para que, usando dos sensores IR, pueda seguir una línea negra pintada sobre un fondo blanco. Recomendamos imprimir la pista Multiplo.PAD.Basic9x7__v1_0.png con la cual se podrán testear los resultados.

1- Lo primero que hay que hacer es adaptar el robot para que un par de sensores IR apunten hacia el suelo. Hay que tener en cuenta que los IR seleccionados tendrán que estar apoyados sobre el piso ya que, si se los pusiera un poco separados, podrían no darse los resultados esperados. La adaptación recomendada consiste en reemplazar la rueda de apoyo delantera por una pequeña estructura que sostenga los dos sensores. Luego, conectar estos a dos salidas para sensores (para más información acerca de los conectores, ver la Guía rápida del robot Multiplo N10). Recomendamos testear el funcionamiento de ambos sensores antes de continuar (para más información acerca de testeo de sensores IR, ver la Guía de actividades 1.3.1).

La siguiente figura muestra cómo quedarían los sensores IR en la parte frontal del robot.



2- Escriba el siguiente código en el entorno de programación Arduino 0022. La idea del código escrito a continuación es la siguiente: si los dos sensores detectan color negro, lo cual significa que están sobre la línea, el robot avanza en línea recta. Si el sensor derecho detecta blanco y el izquierdo negro, el robot avanzará hacia la izquierda para volver a la línea negra. Por último, si el sensor derecho detecta negro y el izquierdo blanco, el robot girará hacia la derecha. Detectar blanco significa que el haz de luz rebotó con mucha intensidad, mientras que detectar negro significa que el sensor captó un valor bajo de luz. Por esta razón, es importante que la línea negra sea opaca ya que, de ser brillante, rebotaría el haz de luz y sería captado con más intensidad de la que deseamos.

Actividad 4.4: hacer que el robot siga una línea negra dibujada sobre un fondo blanco.

```
#define TRIGGER        600
#define MOTOR_SPEED    70.0

float s0, s1;

void setup()
{
  Serial.begin(115200);
  motor1.setClockwise(false);
  motor0.setSpeed(MOTOR_SPEED);
  motor1.setSpeed(MOTOR_SPEED);
}

void loop()
{
  //Read sensors:
  s0 = analogRead(0);
  s1 = analogRead(1);

  /*
  //Debug:
  Serial.print("s0 = ");
  Serial.print(s0);
  Serial.print("/ s1 = ");
  Serial.println(s1);
  */

  if ( (s0<TRIGGER) && (s1<TRIGGER) )  //00
  {
    motor0.setSpeed(MOTOR_SPEED);
    motor1.setSpeed(MOTOR_SPEED);
  }
  else if ( (s0<TRIGGER) && (s1>TRIGGER) ) //01
  {
    motor0.setSpeed(0.1*MOTOR_SPEED);
    motor1.setSpeed(MOTOR_SPEED);
  }
  else if( (s0>TRIGGER) && (s1<TRIGGER) ) //10
  {
    motor0.setSpeed(MOTOR_SPEED);
    motor1.setSpeed(0.1*MOTOR_SPEED);
  }
  //11: Keeps last state.
}
```

Explicación del código de la actividad 4.4

```
#define TRIGGER        600
```

Esta línea de código crea una constante. Una constante es un valor cuyo valor no puede ser modificado durante la ejecución del programa. Por lo tanto, no le podremos volver a asignar un valor. En nuestro caso, la constante se llama TRIGGER y la vamos a utilizar para determinar cuando el sensor está detectando negro y cuando blanco.

```
#define MOTOR_SPEED    70.0
```

Esta nueva constante que declaramos con el nombre MOTOR_SPEED la usaremos para asignarle velocidad a los motores del robot.

```
float s0, s1;
```

A diferencia de las constantes declaradas más arriba, esta línea de código declara dos variables llamadas s0 y s1 las cuales almacenarán un número de punto flotante cada una. Estos valores sí podrán ser modificados durante la ejecución del programa por lo tanto podremos asignarles valores más adelante.

```
  Serial.begin(115200);
  motor1.setClockwise(false);
```

Dentro de la función setup(), seteamos la velocidad de comunicación en 115200 baudios y también la dirección del motor 1.

```
  motor0.setSpeed(MOTOR_SPEED);
  motor1.setSpeed(MOTOR_SPEED);
```

También dentro de la función setup() asignamos la velocidad que elegimos al principio del código utilizando la constante MOTOR_SPEED. De esta manera, los motores avanzarán hacia adelante con la velocidad elegida en la segunda línea de este programa.

```
  //Read sensors:
  s0 = analogRead(0);
  s1 = analogRead(1);
```

Dentro de la función loop(), empezamos asignando a las variables s0 y s1 los valores captados por los sensores 0 y 1 respectivamente. Al realizar esta acción dentro del ciclo principal de la aplicación, podremos tener actualizado el valor guardado en s0 y s1 con respecto a lo detectado por los sensores IR.

```
/*
//Debug:
Serial.print("s0 = ");
Serial.print(s0);
Serial.print("/ s1 = ");
Serial.println(s1);
*/
```

Este pequeño bloque de código, al estar comentado, no será ejecutado por el procesador. En caso de querer probar el funcionamiento de los sensores se podrá usar este código descomentándolo. Para eso hay que borrar la primera línea que contiene una barra y un asterisco (/*) y la última que contiene un asterisco y una barra (*/).

```
if ( (s0<TRIGGER) && (s1<TRIGGER) )   //00
```

Luego, escribimos la estructura de control if con dos condiciones. La primera compara el valor detectado por el sensor 0 con la constante TRIGGER. Si el valor almacenado en s0 es menor que el valor asignado a la constante, entonces el sensor 0 está recibiendo poca luz de rebote por lo que suponemos que está sobre la línea negra. La misma comparación la hacemos con el sensor 1. Las dos comparaciones las relacionamos con el operador lógico && porque tienen que ser verdaderas ambas al mismo tiempo. En caso que los dos sensores detecten negro, entonces el robot va a avanzar en línea recta hacia adelante. Al final de la línea encontramos dos barras y dos ceros (//00). Esto es simplemente un comentario que escribimos para recordar el estado de los sensores. Las dos barras hacen que el compilador no lea lo que se escriba a continuación de ellas y, los ceros indican que cada sensor detectó un valor inferior al TRIGGER.

```
  motor0.setSpeed(MOTOR_SPEED);
  motor1.setSpeed(MOTOR_SPEED);
```

Como los dos sensores detectaron “negro”, asignamos la misma velocidad a ambos motores. Esto lo hacemos utilizando la constante MOTOR_SPEED que definimos al principio.

```
else if ( (s0<TRIGGER) && (s1>TRIGGER) ) //01
{
  motor0.setSpeed(0.1*MOTOR_SPEED);
  motor1.setSpeed(MOTOR_SPEED);
}
```

En este segundo if comparamos el valor captado por cada sensor de manera diferente. En el caso del sensor 0 preguntamos si es menor que el umbral que almacenamos bajo el nombre TRIGGER. Mientras que, con respecto al sensor 1, preguntamos si el valor que detectó es mayor que el umbral. En caso de que las dos condiciones sean verdaderas simultáneamente (por usamos el operador lógico “&&”), vamos acelerar un motor más que el otro. Teniendo el sensor 0 a la izquierda del sensor 1 y el motor 0 a la izquierda del robot, cuando la condición de este if es verdadera, estaríamos detectando la línea negra a la izquierda del robot y blanco a la derecha. Por lo tanto deberíamos desacelerar al motor izquierdo. Para reducir la velocidad de este motor multiplicamos la velocidad que utilizamos por 0.1. De esta manera, la velocidad del motor 0 será del 10% de la velocidad del motor 1.

```
else if( (s0>TRIGGER) && (s1<TRIGGER) ) //10
{
  motor0.setSpeed(MOTOR_SPEED);
  motor1.setSpeed(0.1*MOTOR_SPEED);
}
```

Por último, tenemos la situación en la que el sensor derecho detecta blanco y el sensor izquierdo detecta negro. En este caso, reduciremos la velocidad del motor derecho multiplicándola por 0.1. Por otro lado, a continuación de la condición, escribimos un comentario que indica la situación de los sensores siendo el primer valor el que representa lo detectado por el sensor izquierdo (1 para `s0>TRIGGER`) y el segundo lo detectado por el derecho (0 para `s1<TRIGGER`).

Actividad 4.5: hacer que el robot siga una luz

Objetivo: Que el alumno aumente la complejidad de sus programas logrando una mayor interacción del robot con el medio que lo rodea.

En esta actividad queremos que el robot avance siguiendo una luz. La idea es que alguien con una linterna le marque el camino que desee, apuntándoles a los sensores IR que están montados en la parte frontal del robot. Para realizar esta actividad habrá que montar dos sensores IR en la parte frontal del robot, los cuales tendrán que estar apuntados hacia delante. Para realizarla, escriba el siguiente código:

Actividad 4.5: hacer que el robot siga una luz.

```
void setup()
{
  motor0.setClockwise(false);
}

void loop()
{
  if(analogRead(0)<500 && analogRead(1)<500)
  {
    //Avanza hacia adelante
    motor0.setSpeed(50);
    motor1.setSpeed(50);
  }
  else if(analogRead(0)>500)
  {
    //Gira a la derecha
    motor0.setSpeed(50);
    motor1.setSpeed(0);
  }
  else if(analogRead(1)>500)
  {
    //Gira a la izquierda
    motor0.setSpeed(0);
    motor1.setSpeed(50);
  }
  else
  {
    //Se detiene
    motor0.setSpeed(0);
    motor1.setSpeed(0);
  }
}
```

Explicación del código de la actividad 4.5

```
if(analogRead(0)<500 && analogRead(1)<500)
```

La estructura de control if permite ejecutar un bloque de código cuando la condición encerrada entre paréntesis es verdadera. En este caso, la condición que nos interesa, toma el valor captado por los sensores 0 y 1 y los compara con un valor. Si resulta ser cierto, ejecutará el código encerrado entre llaves que le sigue. Dentro de los paréntesis separamos las dos condiciones que nos importan utilizando el operador lógico &&. Este hará que se ejecute el código que está a continuación únicamente si las dos condiciones son verdaderas.

```
  //Avanza hacia adelante
  motor0.setSpeed(50);
  motor1.setSpeed(50);
```

Dentro de las llaves del primer if, escribiremos el código necesario para que el robot avance en línea recta hacia adelante. Podemos acompañar el código con comentarios que sirvan para seguir mejor el programa en caso de que tengamos que modificarlo. Estos comentarios no son tenidos en cuenta por el compilador, pero nos pueden ayudar a entender lo que estamos haciendo. Los comentarios simples se escriben utilizando // al comienzo de la línea.

```
else if(analogRead(0)>500)
{
  //Gira a la derecha
  motor0.setSpeed(50);
  motor1.setSpeed(0);
}
```

En caso de que sea el sensor derecho el que reciba la luz de la linterna, el robot deberá doblar hacia dicho lado. Para lograrlo, hacemos la misma comparación que en la primera parte del código pero esta vez utilizando el sensor número 0. Por otro lado, no hay que olvidarse de asignar a los motores potencias distintas para que el robot gire hacia el lado del que viene la luz. Es importante destacar el hecho de que este fragmento de código empieza con la palabra reservada "else" porque se presenta como una alternativa al primer "if" que se escribió. Esto es, si (if) la primera condición es verdadera, se ejecuta el bloque de código que le sigue, si no (else) se ejecuta el bloque de código que sigue al else.

```
else if(analogRead(1)>500)
{
  //Gira a la izquierda
  motor0.setSpeed(0);
  motor1.setSpeed(50);
}
```

Por último, hacemos la misma comparación con el sensor izquierdo que en este caso será el número 1 y asignamos potencia a los motores de tal manera que el robot gire a su izquierda. De nuevo nos encontramos con el hecho de que el bloque empieza con la palabra reservada "else", esto se debe a que ninguno de los tres bloques que escribimos se puede ejecutar en simultáneo con el otro. Solamente uno de los tres será verdadero y será el que se ejecute. En caso de que alguien apunte al robot con dos linternas, se ejecutará el primero de los bloques.

Actividad 4.6: hacer que el robot siga una luz utilizando tres sensores IR

Objetivo: Que el alumno aumente la complejidad de sus programas logrando una mayor interacción del robot con el medio que lo rodea.

En esta actividad queremos que el robot avance siguiendo una luz. La idea es que alguien con una linterna le marque el camino que desee, apuntándoles a los sensores IR que están montados en la parte frontal del robot. Para esta actividad será necesario tener montados tres sensores IR en la parte frontal del robot. Para realizarla, escriba el siguiente código:

Actividad 4.6: hacer que el robot siga una luz, utilizando 3 sensores IR.

```
void setup()
{
  motor0.setClockwise(false);
}

void loop()
{
  if(analogRead(1)>500)
  {
    //Avanza hacia adelante
    motor0.setSpeed(40);
    motor1.setSpeed(40);
  }
  else if(analogRead(2)>500)
  {
    //Avanza girando a la derecha
    motor0.setSpeed(70);
    motor1.setSpeed(20);
  }
  else if(analogRead(3)>500)
  {
    //Avanza girando a la izquierda
    motor0.setSpeed(20);
    motor1.setSpeed(70);
  }
  else
  {
    //Frena
    motor0.setSpeed(0);
    motor1.setSpeed(0);
  }
}
```

Explicación del código de la actividad 4.6

```
if(analogRead(1)>500)
```

La estructura de control if permite ejecutar un bloque de código cuando la condición encerrada entre paréntesis es verdadera. En este caso, la condición que nos interesa, toma el valor captado por el sensor 1 y lo compara con otro valor. Si resulta ser verdadera la comparación, ejecutará el código encerrado entre las llaves que le siguen.

```
   //Avanza hacia adelante
   motor0.setSpeed(40);
   motor1.setSpeed(40);
```

Dentro de las llaves del primer if, escribiremos el código necesario para que el robot avance en línea recta hacia adelante. Podemos acompañar el código con comentarios que sirvan para seguir mejor el programa en caso de que tengamos que modificarlo.

```
else if(analogRead(2)>500)
{
  //Avanza girando a la derecha
  motor0.setSpeed(70);
  motor1.setSpeed(20);
}
```

En caso de que sea el sensor derecho el que reciba la luz de la linterna, el robot deberá doblar hacia dicho lado. Para lograrlo, hacemos la misma comparación que en la primera parte del código pero esta vez utilizando al sensor número 2. Por otro lado, no hay que olvidarse de asignar a los motores potencias distintas para que el robot gire hacia el lado del que viene la luz. Es importante destacar el hecho de que este fragmento de código empieza con la palabra reservada "else" porque se presenta como una alternativa al primer "if" que se escribió. Esto es, si (if) la primera condición es verdadera, se ejecuta el bloque de código que le sigue, si no (else) se ejecuta el bloque de código que sigue al else.

```
else if(analogRead(3)>500)
{
  //Avanza girando a la izquierda
  motor0.setSpeed(20);
  motor1.setSpeed(70);
}
```

Por último, hacemos la misma comparación con el sensor izquierdo que en este caso será el número 3 y asignamos potencia a los motores de tal manera que el robot gire a su izquierda. De nuevo nos encontramos con el hecho de que el bloque empieza con la palabra reservada "else", esto se debe a que ninguno de los tres bloques que escribimos se puede ejecutar en simultáneo con el otro. Solamente uno de los tres será verdadero y será el que se ejecute. En caso de que alguien apunte al robot con dos linternas, se ejecutará el primero de los bloques.

```
else
{
  //Frena
  motor0.setSpeed(0);
  motor1.setSpeed(0);
}
```

Por último, si ningún sensor detecta que lo están iluminando, el robot se frenará. De esta manera, el robot solo avanza cuando se le apunta con la linterna a alguno de sus sensores frontales.

Preguntas para investigar

1- Averiguar qué otros dispositivos o juguetes utilizan este tipo de sensores.
2- ¿Dependen estos dispositivos o juguetes de un procesador central para manejar la información recolectada por el sensor como en el caso de los robots?
3- Averiguar qué industrias utilizan este tipo sensores y qué funciones cumplen estos.

Actividades para el alumno

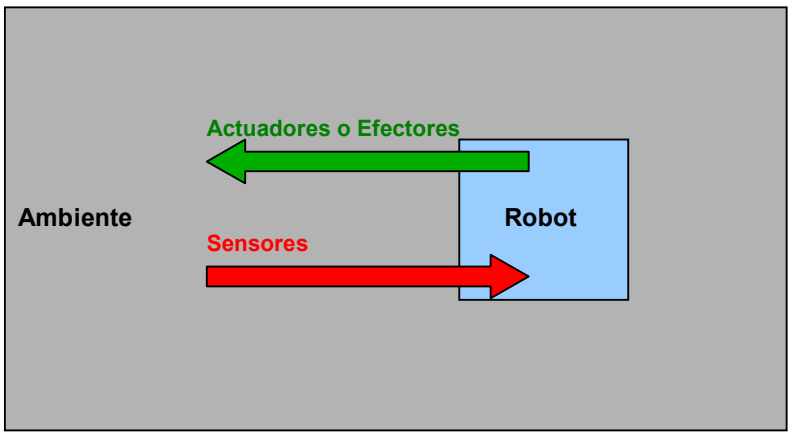
1- Escribir el código de una función que sirva para testear varios sensores al mismo tiempo mostrando por pantalla los valores captados por cada uno de ellos.
2- Escribir el código de una función que haga que el robot, utilizando un solo sensor IR montado en su parte frontal, avance mientras sea alumbrado por una linterna.

5. Actuadores

Algo de teoría sobre emisores de luz

Los robots utilizados poseen dos tipos de emisores de los cuales uno emite luz visible y el otro luz infrarroja. Para utilizarlos se les asigna potencia de la misma manera como se hace con los motores.

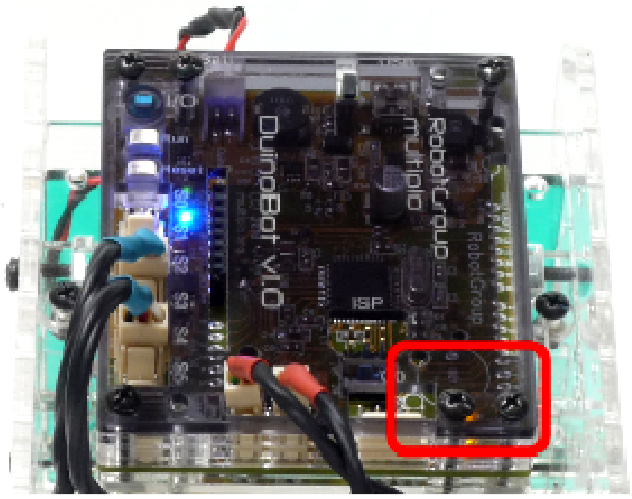
A los emisores de luz de los robots se los suele clasificar como actuadores o efectores. Esto se debe a que, a diferencia de los sensores que detectan características del ambiente, pueden modificar el comportamiento del robot, los emisores modifican el ambiente en el que están.



Los emisores infrarrojos emiten un haz de luz invisible para el ojo humano pero que puede ser muy útil en robótica para darles más alcance a los sensores IR. Esto se debe a que, al emitir junto con el emisor del propio sensor, el alcance y la intensidad de la luz será mayor, por lo tanto, también será mayor la intensidad detectada por el sensor. Lo que le permitirá detectar obstáculos a una distancia mayor.
Por otro lado, contamos con LEDs. Estos emisores de luz reciben su nombre de las iniciales en inglés de las palabras *Light Emitting Diode*, que significan diodo emisor de luz. Estos diodos semiconductores emiten luz cuando una corriente eléctrica circula por ellos. De esta manera, cuando a nuestro robot se le de la orden de asignar potencia a una salida de motor conectada con un emisor de luz, este alumbrará según la potencia asignada. Sin embargo, es importante aclarar que, en el caso de nuestros robots, los emisores de un solo LED se quemarían si se los conectara directamente a los 5V que utilizan los motores, por eso utilizan una resistencia que baja la tensión de 5V a 2V. Por otro lado, los emisores compuestos por 2 LEDs son más potentes y poseen un transistor para que les llegue más corriente.

Actuadores en el robot Multiplo N6

Ya vimos cómo utilizar sensores IR con los robots N6. Ahora vamos a aprender otras formas de interacción entre el robot y el ambiente que lo rodea. Una forma es utilizando LEDs para que el robot emita algún tipo de señal.
El robot Multiplo N6 posee un emisor de luz amarillo la parte frontal de la placa DuinoBot. Tal como se muestra en la siguiente figura:



Para utilizarlo tendremos que configurar algunas cosas desde el entorno de programación Arduino 0022. En primer lugar, hay que tener en cuenta que cada entrada o salida que posee la placa está asociada a un número. Este número corresponde al pin en el que está conectada la entrada. Por ejemplo, cuando utilizamos el sensor 1, estamos utilizando la salida analógica número 1. En el caso de los sensores, no es necesario configurar dicho pin como salida, pero en los ejemplos que vamos a ver a continuación, será necesario decirle al robot cómo será utilizado el pin en cuestión.

En el caso del LED amarillo que posee el robot Multiplo N6, el número de pin correspondiente es el 13. Ahora, sabiendo que el pin que queremos utilizar es el 13, tendremos que decirle al robot si el pin estará encendido o apagado. Para eso, el entorno Arduino 0022 nos provee de la función digitalWrite(pin, value). Esta función toma como parámetros de entrada el número de pin, que en nuestro caso será el 13, y el valor que se le asignará. Este valor será "HIGH" o "LOW" según si queremos prenderlo o apagarlo. Estos dos valores son utilizados por el entorno como constantes, lo que significa que siempre que utilicemos alguno de ellos, el entorno sabrá a qué nos referimos. Para más información sobre esta función, se puede consultar la guía de referencia de Arduino:

http://www.arduino.cc/en/Reference/DigitalWrite

Una constante en programación es un valor que no puede ser modificado mientras se ejecuta el programa al que pertenece. El entorno Arduino 0022 tiene constantes y las mismas se pueden consultar en la página de referencias, en la columna central "Variables":

http://arduino.cc/en/Reference/HomePage

Las dos constantes, HIGH y LOW, representan los valores de voltaje que le llegarán al LED. Si seteamos al pin 13 como HIGH, la placa controladora le mandará 5 voltios, consiguiendo como resultado que el LED se encienda. En el caso que usemos LOW, la placa emitirá 0 voltios a través del pin en cuestión, lo que resultará en un led apagado. Probemos todo esto con un ejemplo sencillo.

Actividad 5.1: hacer que el LED se prenda intermitentemente

Objetivo: Que el alumno aumente la complejidad de sus programas utilizando las salidas que posee el robot.

En esta actividad queremos que el LED frontal del robot esté prendido durante un segundo y luego esté apagado por el siguiente y que esto se repita continuamente. Para realizarla, escribiremos la siguiente función y la llamaremos desde loop():

Actividad 5.1: encender y apagar un LED intermitentemente.

```
void yellowLED()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}

void setup()
{}

void loop()
{
  yellowLED();
}
```

Explicación del código de la actividad 5.1

```
void yellowLED()
```

Esta es la declaración de la función que queremos escribir. Como no nos devuelve ningún valor, la declaramos como void y no escribimos nada entre los paréntesis porque tampoco toma valores de entrada.

```
  digitalWrite(13, HIGH);
```

Como vimos antes, lo primero que tenemos que hacer es decirle a la placa que el pin 13 está prendido. Internamente, el controlador envía 5 voltios al componente conectado al pin, lo que da como resultado que el LED se enciende. Si no escribimos nada más a continuación, el pin quedará encendido hasta que el robot deje de ejecutar el programa.

```
  delay(1000);
```

Con esta orden le decimos al procesador que no ejecute nada nuevo durante un segundo. Este será el tiempo que el LED permanecerá encendido.

```
  digitalWrite(13, LOW);
```

Con esta orden apagamos el pin 13. Como vimos antes, lo que sucede internamente es que el controlador deja de enviar los 5 voltios que enviaba antes debido a que ahora está seteado en 0 voltios. Si no escribimos esta orden, aunque esté el delay(1000) anterior, el LED seguirá encedidol.

```
  delay(1000);
```

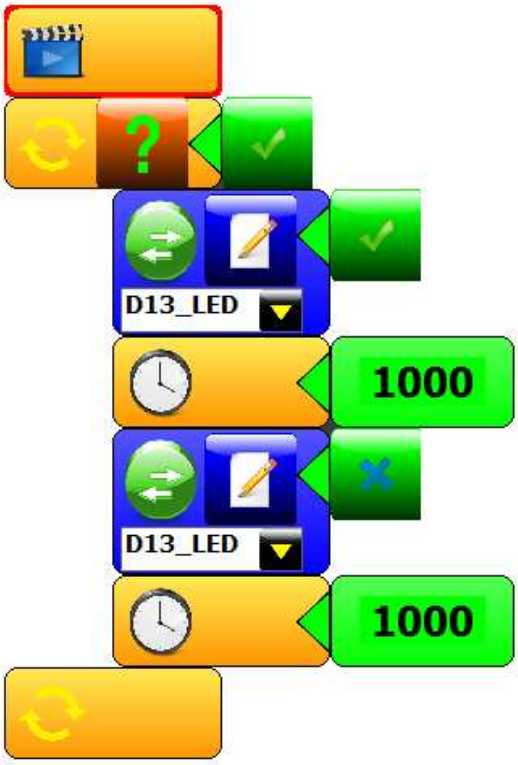
Con esta orden le decimos al procesador que no ejecute nada nuevo durante un segundo. Este será el tiempo que el LED permanecerá apagado.

Luego de escribirla, a esta función la llamamos desde loop() ya que queremos que esto lo realice siempre. Para llamarla, simplemente escribimos el nombre de la función como se muestra a continuación:

```
void loop()
{
  yellowLED();
}
```

Código en Minibloq de la actividad 5.1

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Actividad 5.2: hacer que un LED emita luz intermitentemente

Para la actividad que está a continuación, utilizaremos las salidas digitales que provee la placa DuinoBot pero que no poseen anclaje Multiplo. Estas son similares a las que proveen las placas Arduino. Estas salidas, que se encuentran en el frente de la placa y que están ubicadas contra el borde superior, están numeradas de la siguiente manera:

AREF	GND	13	12	11	10	9	8	7	6	5	4	3	2	1	0

GND indica tierra y los números indican a qué pin corresponde dicha salida. En esta sencilla actividad haremos que un LED se encienda y se apague cada un segundo. Para realizarla, hay que conectar un LED de la siguiente manera: el ánodo que es el contacto más largo se conecta a GND (tierra) mientras que el cátodo se conecta al pin numerado con el número 12, como se muestra en la siguiente figura.

A continuación, escribiremos el siguiente código en el entorno Arduino.

Actividad 5.2: hacer que un LED emita luz intermitentemente

```
void setup()
{
  pinMode(12, OUTPUT);
}

void loop()
{
  digitalWrite(12, HIGH);
  delay(1000);
  digitalWrite(12, LOW);
  delay(1000);
}
```

Explicación del código de la actividad 5.2

A continuación se explica el código escrito para realizar la actividad.

```
pinMode(12, OUTPUT);
```

Con esta línea de código establecemos que el pin numerado como 12 se utilizará como salida. La función pinMode necesita dos parámetros, el primero para el número de pin que queremos utilizar y el segundo para establecer si este se utilizará como salida (OUTPUT) o como entrada (INPUT). Esta línea la escribimos dentro de la función setup() porque solo tenemos que configurar el modo del pin una única vez.

```
digitalWrite(12, HIGH);
```

Esta función la usamos para decirle a la placa que escriba un valor dado por la variable HIGH en el pin 12. En este caso, el pin recibirá 3,3 voltios lo cual es suficiente para que se encienda el pin que estamos utilizando.

```
delay(1000);
```

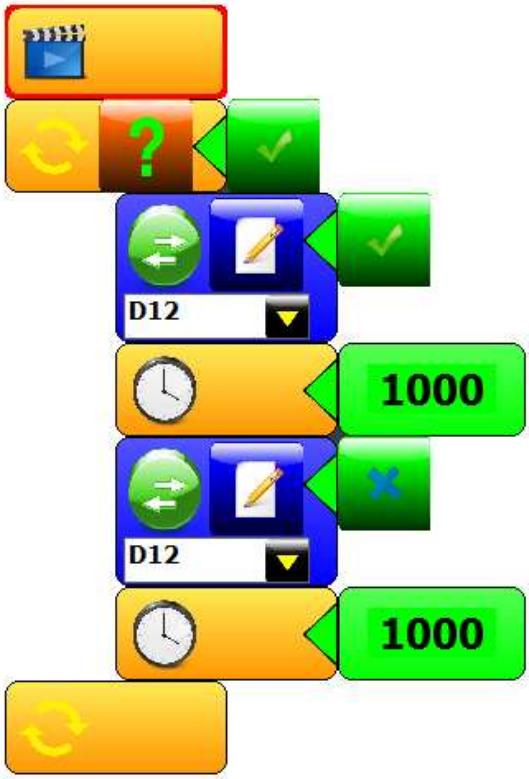
Utilizamos delay() para hacer que el procesador no reciba órdenes por el tiempo especificado entre paréntesis. En este caso, durante 1000 microsegundos (1 segundo), el procesador no recibirá nuevas órdenes, por lo tanto el LED quedará encendido durante este tiempo ya que a continuación le daremos la orden de que se apague.

```
digitalWrite(12, LOW);
```

En este caso utilizamos la constante LOW para que el pin 12 reciba 0 voltios, o sea, no tendrá energía para encenderse.

Código en Minibloq de la actividad 5.2

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



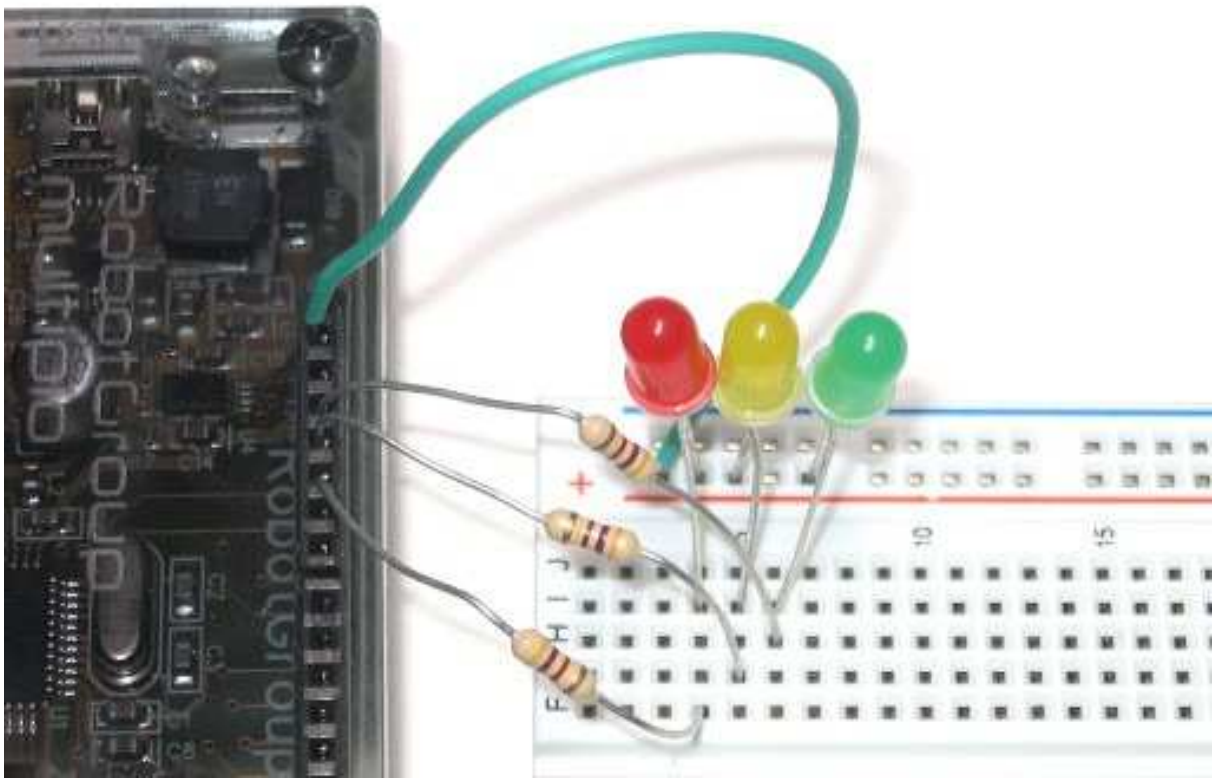
Actividad 5.3: hacer un semáforo con tres LEDs

En esta actividad haremos un semáforo con un LED rojo, uno amarillo y otro verde. Para realizarla, necesitaremos algunos componentes que podremos conseguir en una casa de electrónica. Además de los LEDs, también utilizaremos algunas resistencias, un poco de cable y un protoboard.

Las resistencias son pequeños componentes que nos servirán para evitar que se quemen los LEDs. Su nombre nos indica que su función es resistir la corriente que circula a través de ellas. La resitencia que oponen se mide en Ohmios y las que utilicemos tendrán que ser de 400 a 1500 Ohmios.

Por otro lado, se llama protoboard a una placa que se utiliza para pruebas. Estas permiten conectar componentes y cables sin necesidad de soldarlos. Sobre esta placa de pruebas, montaremos los LEDs y las resistencias de nuestra actividad.

En este caso, en vez de conectar el LED directamente a la salida de la placa como lo hicimos antes, conectaremos tres resistencias a los pines 10, 11 y 12. Estas a su vez se conectarán al protoboard. La idea es que la corriente circule desde el pin y a través de la resistencia para luego alimentar al LED y terminar el recorrido en el pin GND. El circuito quedará como se muestra en la siguiente figura:



Una vez que esté listo este circuito, escribimos el siguiente código en el entorno Arduino:

Actividad 5.3: hacer un semáforo con dos LEDs

```
int green = 12;
int yellow = 11;
int red = 10;

void setup()
{
  pinMode(red, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(green, OUTPUT);
}

void loop()
{
  digitalWrite(red, HIGH);
  digitalWrite(yellow, LOW);
  digitalWrite(green, LOW);
  delay(2000);

  digitalWrite(red, HIGH);
  digitalWrite(yellow, HIGH);
  digitalWrite(green, LOW);
  delay(2000);

  digitalWrite(red, LOW);
  digitalWrite(yellow, LOW);
  digitalWrite(green, HIGH);
  delay(2000);

  digitalWrite(red, LOW);
  digitalWrite(yellow, HIGH);
  digitalWrite(green, LOW);
  delay(1000);
}
```

Explicación del código de la actividad 5.3

A continuación se explica el código escrito para realizar la actividad.

```
int green = 12;
int yellow = 11;
int red = 10;
```

Lo primero que haremos será declarar tres variables enteras que representarán el número de pin al que conectaremos cada LED. En nuestro caso, el LED verde (green) será conectado al pin 12, el amarillo (yellow) al 11 y el rojo (red) al 10.

```
void setup()
{
  pinMode(red, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(green, OUTPUT);
}
```

Dentro de la función setup() seteamos a los tres pines como salidas.

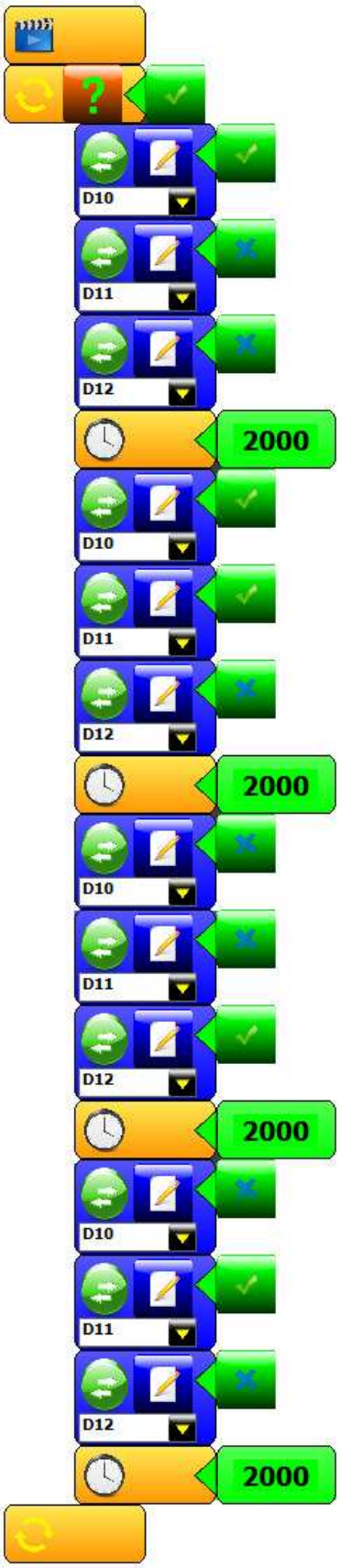
```
  digitalWrite(red, HIGH);
  digitalWrite(yellow, LOW);
  digitalWrite(green, LOW);
  delay(2000);
```

Dentro de la función loop() encendemos y apagamos los LEDs utilizando los pines a los que los conectamos. Al principio encendemos el LED rojo y dejamos apagados el amarillo y el verde. Esto lo hacemos por 2 segundos.

```
  digitalWrite(red, HIGH);
  digitalWrite(yellow, HIGH);
  digitalWrite(green, LOW);
  delay(2000);
```

Con este bloque de código dejamos encendido el LED rojo y encendemos el amarillo mientras que el verde sigue apagado. En los bloques siguientes usaremos el mismo código pero cambiaremos los LEDs que se encienden o se apagan.

Código en Minibloq de la actividad 5.3



El botón RUN como entrada

Una posibilidad que nos dan los robots Multiplo, es la de utilizar el botón RUN como una entrada. Esto significa que podremos darle órdenes al robot a través de este botón. Por ejemplo, podríamos hacer un programa que haga que el robot encienda un led, frene o doble cuando presionamos dicho botón.

Lo primero que tendremos que hacer es decirle al robot que dicho botón será utilizado como entrada y, luego, utilizar la constante HIGH para decirle a la placa que el pin en cuestión está activado. Por lo tanto, tendremos que utilizar una nueva orden que nos provee el entorno Arduino 0022: pinMode(pin, mode). La misma nos sirve para configurar como entrada o como salida el pin en cuestión. Toma como parámetros el número de pin y el modo como se utilizará el mismo. Para decirle el modo de uso, utilizaremos dos nuevas constantes que nos provee el entorno: OUTPUT o INPUT. Con la primera lo podremos configurar como salida y con la segunda como entrada. Para más información acerca de estas constantes, se puede consultar la página:

http://arduino.cc/en/Reference/Constants

Actividad 5.4: testeo del botón RUN

Objetivo: Que el alumno aumente la complejidad de sus programas utilizando las salidas que posee el robot.

En esta actividad queremos probar el funcionamiento del botón RUN para poder utilizarlo en otras actividades. Para realizarla, escribiremos el siguiente código en el entorno de programación Arduino 0022:

Actividad 5.4: utilizar el botón RUN como entrada.

```
void setup()
{
  Serial.begin(9600);
  pinMode(25, INPUT);
  digitalWrite(25, HIGH);
}

void loop()
{
  Serial.print("Value: ");
  Serial.println(digitalRead(25));
  delay(500);
}
```

Explicación del código de la actividad 5.4

```
Serial.begin(9600);
```

Lo primero que tendremos que hacer es establecer la velocidad de comunicación, ya que queremos mostrar por pantalla los resultados de utilizar el botón RUN.

```
pinMode(25, INPUT);
```

Con esta línea de código configuramos la salida 25 como entrada. El pin 25 es el que identifica el botón RUN y, queremos configurarlo como entrada (INPUT) porque nos servirá para tomar valores captados por el robot. En este caso los valores que captará son simplemente 1 ó 0 según el botón esté presionado o no.

```
digitalWrite(25, HIGH);
```

Con esta línea de código le estamos diciendo a la placa que la salida digital 25 está encendida.

```
Serial.print("Value: ");
```

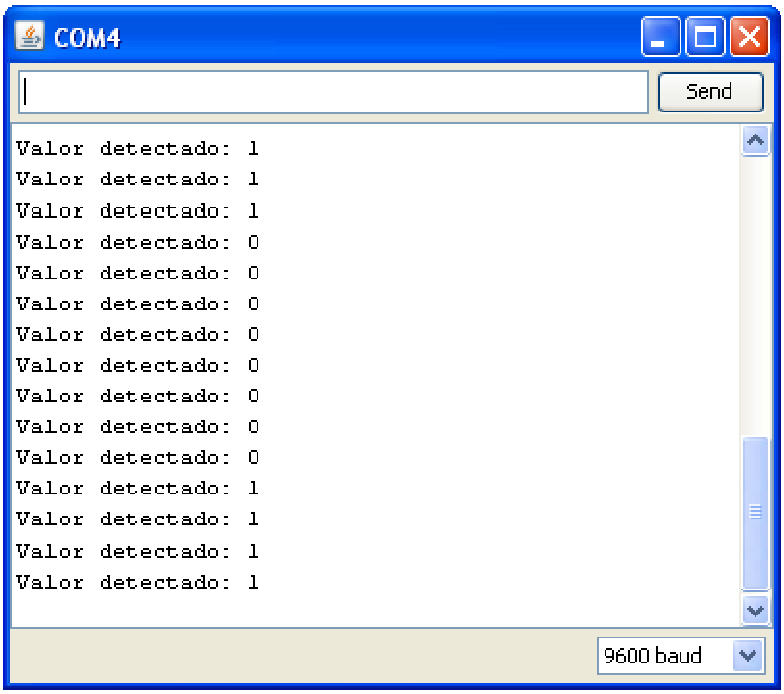
Ya dentro de la función loop(), escribimos esta línea para hacer más legible la salida del programa.

```
Serial.println(digitalRead(25));
```

Esta es la principal línea del programa. Por un lado vamos a imprimir por pantalla el mensaje encerrado entre paréntesis. Ahora bien, dentro de los paréntesis tenemos la llamada a la función digitalRead(25), con lo cual, lo que veremos es el valor detectado por el pin 25. Como con el pin 25 identificamos el botón RUN, esta línea de código nos mostrará un valor cuando el botón no está presionado y otro cuando sí lo está. Es importante destacar el hecho de que estamos usando la función digitalRead(pin) ya que ahora queremos leer (Read) el valor detectado.

Una aclaración importante es que escribimos una parte del código dentro de la función setup() porque no queremos repetir esas líneas ya que las mismas son para configuración. Luego, escribimos el resto del código dentro de la función loop() ya que queremos que este se repita indefinidas veces para poder probar el funcionamiento de la entrada utilizada.

A continuación se muestra la salida del programa. El valor 1 corresponde a las veces que el botón RUN no estaba presionado mientras que el valor 0 a las veces que sí lo estaba:



Actividad 5.5: hacer que el robot avance hasta que se presione el botón RUN

En la siguiente actividad queremos hacer que el robot avance hasta que se apriete el botón RUN, utilizando el mismo como interruptor. Para realizarla, escribimos el siguiente código en el entorno de programación Arduino 0022:

Actividad 5.5: hacer que el robot avance hasta que se presione el botón RUN.

```
void setup()
{
  pinMode(25, INPUT);
  digitalWrite(25, HIGH);
  motor0.setClockwise(false);
}

void loop()
{
 if(digitalRead(25)==1)
 {
   motor0.setSpeed(50);
   motor1.setSpeed(50);
 }
 else
 {
   motor0.setSpeed(0);
   motor1.setSpeed(0);
 }
}
```

Explicación del código de la actividad 5.5

```
pinMode(25, INPUT);
digitalWrite(25, HIGH);
motor0.setClockwise(false);
```

Dentro de la función setup() seteamos los valores correspondientes para que el botón RUN sea utilizado como una entrada. Recordemos que al botón RUN lo identificamos con el pin 25. También cambiamos la dirección del motor 0.

```
if(digitalRead(25)==1)
```

Con esta línea de código haremos que el robot decida según el valor detectado. En el caso de que el valor sea 1, el botón RUN no estará presionado. En este caso usamos un doble signo = ya que estamos comparando dos valores: el captado por la función digitalRead(pin) y el que queremos utilizar como referencia.

```
  motor0.setSpeed(50);
  motor1.setSpeed(50);
```

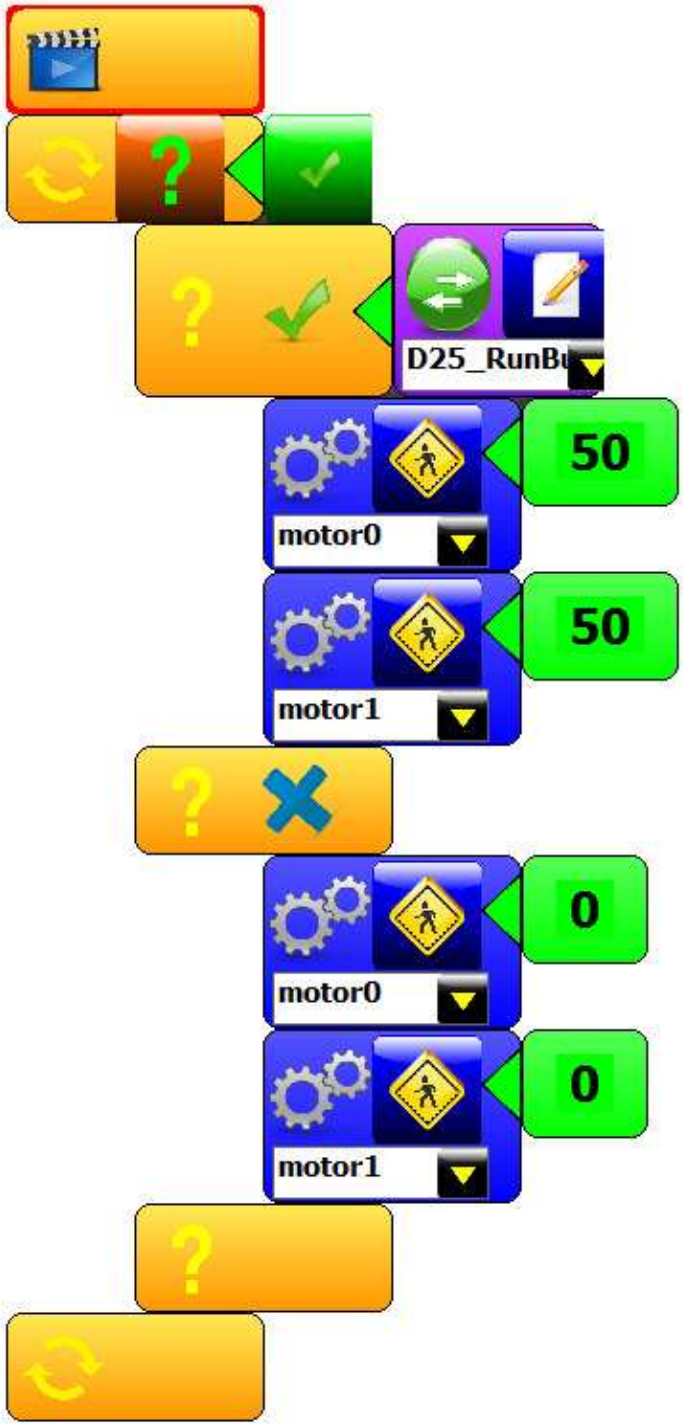
En caso de que la condición sea verdadera, el botón no será presionado, por lo tanto, vamos a decirle al robot que avance en línea recta o que dibuje un círculo.

```
else
{
  motor0.setSpeed(0);
  motor1.setSpeed(0);
}
```

En caso de que la condición dentro del if sea falsa, queremos que el robot se detenga. Para esto, asignamos velocidad igual a cero en ambos motores.

Código en Minibloq de la actividad 5.5

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.



Actividad 5.6: hacer que se encienda el LED frontal al presionar el botón RUN

En esta actividad queremos hacer que, al apretar el botón RUN, el robot encienda el LED amarillo que se encuentra en la parte frontal. Para eso, escribimos el siguiente código:

Actividad 5.6: hacer que el robot encienda un LED cuando se presiona el botón RUN.

```
void setup()
{
  pinMode(25, INPUT);
  digitalWrite(25, HIGH);
}

void loop()
{
 if(digitalRead(25)==1)
 {
  digitalWrite(13, LOW);
 }
 else
 {
  digitalWrite(13, HIGH);
 }
}
```

Explicación del código de la actividad 5.6

```
void setup()
{
  pinMode(25, INPUT);
  digitalWrite(25, HIGH);
}
```

Dentro de la función setup() seteamos los valores necesarios para que el controlador reconozca como entrada el botón RUN.

```
 if(digitalRead(25)==1)
```

Dentro de la función loop(), comparamos el valor de entrada, al cual consultamos a través de la función digitalRead(pin), con 1. De esta manera, sabremos si el botón RUN está presionado o no.

```
  digitalWrite(13, LOW);
```

Si el botón no está presionado, seteamos el pin 13, correspondiente al LED amarillo, como apagado.

```
 else
 {
  digitalWrite(13, HIGH);
 }
```

Si la condición es falsa, significa que el botón RUN está siendo presionado, con lo cual el código que ejecutará el procesador del robot será el que esté dentro de las llaves que siguen a la sentencia else. En este caso, seteamos el valor del pin 13 como encendido. De esta manera el controlador enviará 5 voltios por dicho pin hacia el LED, lo que resultará en que este se encenderá.

Código en Minibloq de la actividad 5.6

Para realizar esta misma actividad utilizando el entorno Minibloq, tenemos que agregar los bloques que se muestran a continuación.

Preguntas para investigar

1- Averiguar qué otros dispositivos o juguetes utilizan este tipo de actuadores.
2- ¿Dependen estos dispositivos o juguetes de un procesador central para manejar las comunicaciones como en el caso de los robots?
3- Investigar acerca de las características que debería poseer un dispositivo para que le resulte útil usar este tipo de sensor.

Actividades para el alumno

1- Un potenciómetro es una resistencia que variable. Esto es, un componente que puede cambiar la resistencia que le ofrece a la corriente. Investigue cómo funciona un potenciómetro y haga un pequeño circuito en el que este se encuentre conectado a un pin de salida y a un LED. Luego, haga variar el brillo del LED utilizando el potenciómetro.