

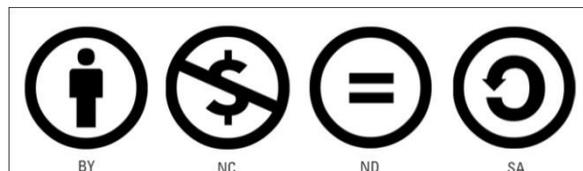
# PRÁCTICAS

## MBOT

Robot con mCore basado en  
Arduino UNO

usando

## MBLOCK



!!!PERO COMPARTAN MUY LOCAMENTE!!!

Javier Fernández Panadero  
[Javierfpanadero@yahoo.com](mailto:Javierfpanadero@yahoo.com)

## ÍNDICE

### Contenido

<b>ÍNDICE .....</b>	<b>2</b>
0. Objetivo.....	3
A. ¿De qué hablamos aquí? .....	3
B. Los niveles de la robótica .....	3
C. mBot .....	4
D. mBlock.....	5
E. Envía tu programa por correo.....	6
1. ¡IT'S ALIIIIIVE! .....	7
2. Viaje.....	8
3. COLORINES .....	11
4. MUSIQUITAS.....	11
5. HÁGASE LA LUZ .....	12
6. ¡DECÍDETE! .....	14
7. QUE NOS CHOCAMOS... ..	16
8. ¡QUE NOS CAEMOS! .....	19
9. SIGUE EL CAMINO DE BALDOSAS NEGRAS... ..	21
10. QUIÉN PUEDE APRETAR EL BOTÓN .....	22
11. EL TIEMPO ES ORO .....	25
12. SUMA Y SIGUE .....	27
13. MANDO A DISTANCIA.....	29
14. mBots CUCHICHEANDO.....	29
15. GIRÓSCOPO .....	31
16. Bloques.....	33
17. ÑACA ÑACA .....	35
18. PROPUESTAS .....	37
<b>FINALMENTE.....</b>	<b>37</b>

## 0. Objetivo

- Prácticas de **robótica desde cero**.
- **Enseñar programación** usando un robot, que resulta más vistoso
- Usaremos **mbot** que es un **robot** comercial basado en **Arduino UNO**

## A. ¿De qué hablamos aquí?

La robótica NO es sólo una automatización (como podría ser la cisterna del váter)

### *TRES ELEMENTOS CLAVE*

#### a) Sensores

Nos darán **información del entorno** (luz-oscuridad, distancias, etc.) o **del usuario** (toca botones, escribe en el teclado, etc.).

#### b) Actuadores

Será **la forma en la que intervengamos en el mundo** (encendiendo luces, produciendo sonidos, enviando información al ordenador, moviendo motores, etc.)

#### c) Programación

Aquí es donde tenemos todo el poder. Con la información de los sensores **haremos que los actuadores se comporten como deseemos**, sin ninguna restricción más que nuestra imaginación.

## B. Los niveles de la robótica

Las cosas que hagamos van a fallar, digo, tienen que funcionar a TRES NIVELES

- **La programación**. Las estructuras de control, el flujo de información, etc.
- **La electrónica**. Nuestros sensores y actuadores trabajan de formas particulares, que si no tenemos en cuenta nos darán sorpresas.
- **El mundo real**. ¿De qué color son las cosas? ¿Cuánto lucen nuestras bombillas? ¿Cómo de rápido es un usuario? ¿Qué tal refleja las ondas una superficie?

Habrá que tener todo esto en cuenta a la hora de escribir el programa. Unas veces será divertido, otras exasperante, pero cuando finalmente el "bicho" haga lo que queremos... nos encantará.

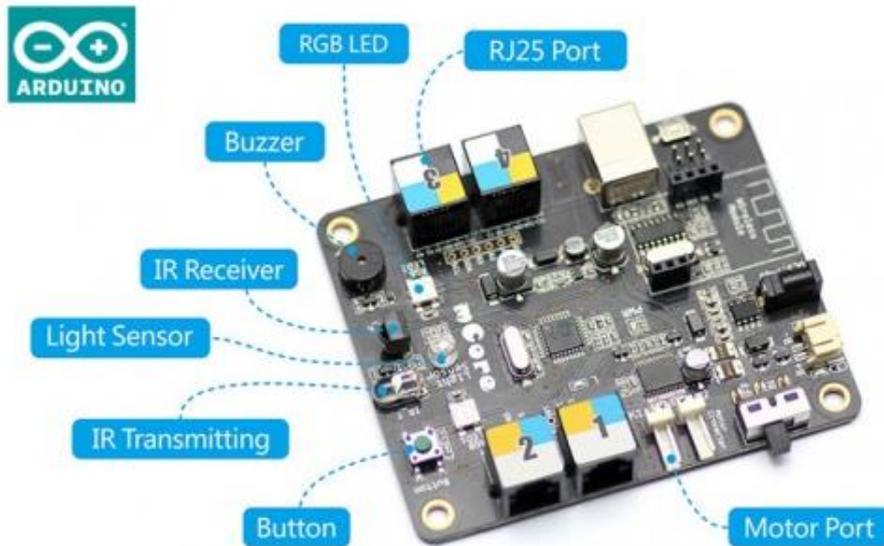
## C. mBot

Robot comercial de MAKEBLOCK



<http://www.makeblock.com/product/mbot-robot-kit>

Está controlado por la tarjeta mCore que es una adaptación de Arduino Uno.

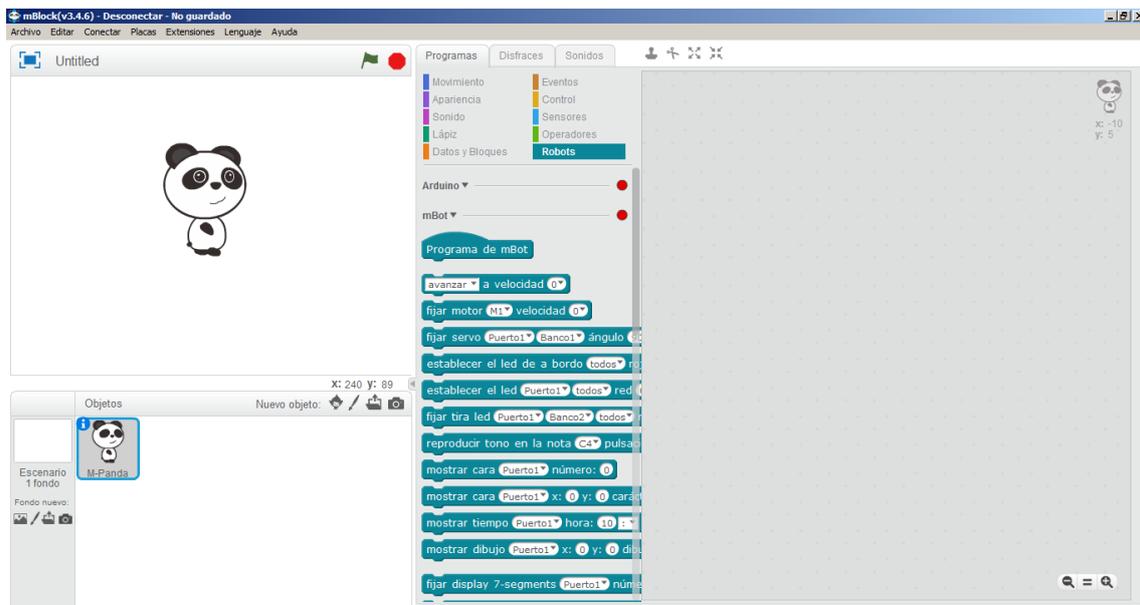


- Tenemos un par de puertos específicos para los dos motores.
- Cuatro puertos con conectores RJ 25
- Un zumbador pasivo (capaz de dar distintos tonos y buen volumen)
- Dos LED RGB
- Un sensor de luz (que mira hacia arriba, desgraciadamente)
- Un receptor de infrarrojos (recibe señales de un mando a distancia –incluido, o de otro mBot)
- Un transmisor de infrarrojos
- Un pulsador
- Un conector para módulo Bluetooth o WiFi

## D. mBlock

mBlock es un programa libre basado en el Scratch del MIT.

- Se basa en bloques
- Muy amigable
- Se puede obtener el código Arduino
  - o Aunque el código tiene instrucciones y llamadas a librerías, con lo que no siempre es sencillo de entender, sobre todo para los chavales.



Podéis hacer programas como con Scratch, con objetos, escenarios, etc. y además tenéis en el menú Robots bloques específicos para trabajar con nuestra tarjeta y robot.

- Lo primero sería **habilitar las extensiones** para tener los bloques necesarios: mBot y Comunicaciones.
- Conecta el robot al ordenador mediante el cable USB (Bluetooth o WiFi)
- En el menú Conectar elige el puerto (tendrás que esperar unos momentos)
- En el menú Conectar actualiza el Firmware
- En el menú Conectar restaura el programa predeterminado

¡Ya estamos listos!

NOTA: El programa predeterminado es un “intérprete” que se carga a la placa y que se comunica constantemente con el ordenador para convertir nuestros programas de bloques en instrucciones Arduino.

Más adelante veremos que también pueden cargarse programas autónomos, pero tendremos ciertas restricciones respecto a los bloques que podremos usar.

**NOTA:** Te aconsejo tener el robot siempre metido en una caja mientras cargas programas o pruebas cosas. Puede que el programa le haga moverse o le des a un botón o al mando y se caiga. No me preguntes cómo lo sé.

**E. Envía tu programa por correo.**

No hay pasta para que haya un robot cada dos alumnos, pero pueden programar en sus ordenadores y enviar por correo el archivo. El profe lo carga y lo ejecuta.

- Es rápido
- Les gusta que se ejecute su propio código
- Podemos discutir errores, variantes, refinamientos.

## 1. ¡IT'S ALIIIIIVE!

Empecemos a movernos

Los que estén familiarizados con Scratch ya saben que podemos ejecutar los programas cuando pulsemos a la “banderita verde” del escenario si colgamos las instrucciones bajo este bloque



Pero no es el único iniciador de un programa. En el menú Eventos tienes un montón.

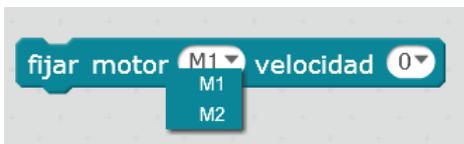
Hagamos un programa con el que podamos mover al robot usando las teclas de los cursores



Con este bloque podemos ejecutar cualquier script disparándolo con la tecla que nos parezca (incluso con cualquiera “any”)

Los movimientos del robot podemos hacerlos combinando movimientos

individuales de los dos motores que van a las ruedas



El número indica la velocidad a la que se moverá el motor, vienen varios valores predeterminados, pero podemos escribir ahí el número que queramos (entre cero y 255).

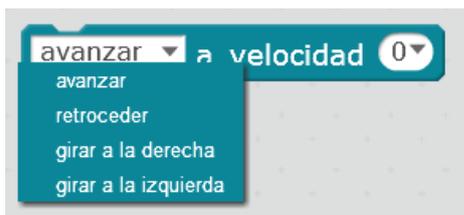
Si son negativos va hacia atrás y por debajo de 50 no tiene fuerza suficiente para mover el coche. De 100 para arriba son valores interesantes.

Si un motor está parado y el otro en marcha girará “pivotando” sobre la rueda parada.

Si un motor se mueve en un sentido y el otro en el contrario, gira alrededor del centro del robot (aprovechando la fuerza de ambos motores)

Si los motores giran a distintas velocidades el coche se mueve dibujando una circunferencia.

Salvo para cosas muy concretas yo prefiero usar este otro bloque



Si pones valores negativos, cambia el sentido (delante-detrás, izquierda-derecha)

Gira en torno al centro, motores en dirección contraria  
Hagamos un coche que se mueva según los cursores (flechas laterales → giros)



¡Fácil!

No hace falta que le des a la “bandera”, mBlock está esperando a que toques las teclas para transformarlas en órdenes Arduino a la placa y que funcione. Pruébalo.

Con este programa vemos dos cosas muy importantes.

A. Este es un programa por **eventos**.

No ejecuta un código y ya, está “esperando” para reaccionar. (Muy robótico!)

B. **Los órdenes persisten** hasta que se recibe otra

Si el coche recibe la orden avanzar, no dejará de hacerlo hasta que no reciba otra como girar o retroceder.

De hecho, fíjate que una vez que tocas alguna de esas teclas NO PUEDES hacerlo parar... Mira en tu zona de programación, ¿está en algún sitio esa orden?

Si quieres puedes completar el programa con esto. Poner cualquiera de las cuatro acciones a velocidad cero es la orden de parar.



Pero esto es un “mando a distancia”, lo que más nos gusta de la robótica es que se muevan por su cuenta...

## 2. Viaje

Haz un programa que recorra el camino que quieras... avance, retroceda, gire... lo que quieras.

Carga distintos programas y hablemos sobre ellos...

Es probable que se mande algo así.



Esto NO funciona, al menos no como lo esperamos. El programa no da un error, hace algo, quizá no lo que tú quieras, pero algo hace.

**NOTA:** Los programas por bloques no te dejan colocarlos donde no se debe, así que es muy difícil que tengan errores sintácticos.

Ya dijimos que las órdenes persisten hasta que se recibe una nueva, por lo que el coche comienza avanzando hasta que llega la orden de girar que es *inmediatamente después*, así que en realidad se queda parando “dudando” hacia donde ir porque le llegan todas las órdenes seguidas.

Si queremos que esté un tiempo avanzando, girando o retrocediendo habrá que añadir una espera después de cada acción. Por ejemplo



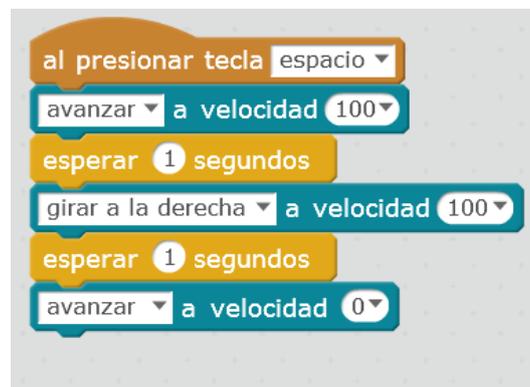
Esto ya avanza durante un segundo y luego gira a la derecha durante un segundo, ¿verdad?

O no... recuerda, las órdenes persisten

Avanza y mira el reloj hasta que se cumple un segundo, gira y mira el reloj hasta que se cumple un segundo... ya no hay más instrucciones. ¡SIGUE GIRANDO!

**NOTA:** Hay que distinguir entre que el robot esté parado y que el programa esté parado o terminado. En nuestro caso el robot está en movimiento, el programa terminado. En el programa anterior (con los cursores) el robot estaba parado “esperando atentamente”, el programa estaba activo.

Si quieres que el robot se pare, habrá que añadir *explícitamente* un bloque con velocidad CERO.



**AYUDAS A LA EDICIÓN**

- Para programar más cómodamente puedes hacer el escenario más pequeño, o incluso ocultarlo totalmente en el menú EDITAR.
- Al hacer doble click en un bloque (o un conjunto de bloques) ejecuta su acción, aunque estén sueltos.
- Con botón derecho pueden duplicarse bloques o grupos de bloques
- Al arrastrar un bloque se separa él y los que cuelguen bajo él
- Arrastrar un bloque o conjunto de bloques a la zona donde están los bloques lo borra
- Los bloques sueltos no se ejecutan al producirse eventos, ni dan errores
- Para ver mejor los valores de las variables en la “pizarra” puedes poner el escenario a pantalla completa
- Para subir un programa como programa autónomo a la placa hay que usar el modo Arduino en el menú Editar
- Para ver el monitor serie hay que seleccionar el modo Arduino en el menú Editar
- Cuando te pongas en modo Arduino no te permitirá hacer multihilo, o disparar ejecución con eventos como tocar teclas. También darán error las variables con nombres de más de una palabra

### 3. COLORINES

El siguiente bloque nos permite poner el color de los dos LEDs de a bordo al tono que queramos bien juntos o por separado.



Los valores en cada color pueden ir de 0 a 255, hay varios valores predeterminados o puedes escribir tú el número que desees, más adelante también podríamos poner una variable...

Si quieres elegir un color concreto puedes ir a un programa como el Paint de Windows o a cualquier web a ver el código RGB (red, green, blue) de cualquier tono. En teoría hay más de 16 millones de colores posibles... pero yo no esperaría esa fiabilidad de estos LEDs nuestros.

Combinando estos bloques con esperas o con bucles de repeticiones podemos liar un buen festival. ¡Adelante!

Por cierto, observa que los LEDs se quedan en el último valor que les hayas dicho (como también hacían los motores), si quieres que al final de tu programa se apaguen, ponlos a cero, pero DÍSELO, no lo van a suponer.

### 4. MUSIQUITAS

Con este bloque podemos hacer que el buzzer (zumbador) pasivo de la placa mCore nos dé un tono a nuestra elección



Predeterminadamente tenemos varias notas en varias octavas (A = La, B = Si... G = Sol), el número sería la octava. A4 corresponde al La central del teclado (440Hz).

La "pulsación" será la duración de la nota, "Medio" equivale a medio segundo, y así sucesivamente.

Si quieres puedes poner en el lugar de la nota un valor de frecuencia y en el de la pulsación una duración en milisegundos.

Si pones dos notas seguidas iguales verás que hay una pequeña "respiración" entre ellas, y si lo pones como Programa de mBot y miras el código Arduino, verás que está indicada con un `delay(20)`. Si quisieras editar el código podrías eliminarla. Para tocar canciones no estorba, pero si quisiéramos hacer una "alarma de proximidad para aparcar" nos estorbaría.

## 5. HÁGASE LA LUZ

Usemos el sensor de luz... pero, ¿cómo funciona? ¿Qué valores da? ¿Sube o baja cuando la luz sube? Necesitamos saber cómo es nuestra electrónica y nuestro entorno... y adaptar la programación a eso.

El sensor le manda el dato a la placa pero, ¿quién nos lo dice a nosotros?

Para esto usaremos nuestra PRIMERA VARIABLE

- Damos a Crear Variable
- Elegimos el nombre de la variable y su alcance
- Nos aparecen distintas operaciones que podemos hacer

**NOTA:** Es bueno que los nombres de las variables sean sencillos y que identifiquen claramente qué son esas variables.

La casilla de verificación hará que se muestre el valor de la variable en el escenario.

La etiqueta con el nombre de la variable nos dará el VALOR de esa variable

FIJAR asigna un valor a la variable

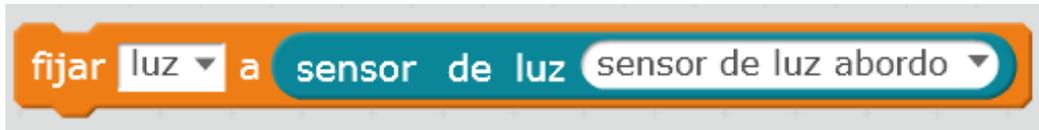
CAMBIAR suma lo que pongamos a la variable. Por ejemplo si la luz vale 300 y ponemos el bloque Cambiar luz por 5, el nuevo valor de luz será 305. (Lo sé es una traducción lamentable)

MOSTRAR Y ESCONDER hará que cuando se esté ejecutando un programa y aparezca ese bloque el valor de la variable aparezca o desaparezca. A nosotros, para este ámbito, nos interesa mucho verlas.

## Volvamos a nuestro problema

### ¿Queremos saber el valor que lee el sensor?

Lo metemos en una variable y listo.



Insisto en que usamos FIJAR y no CAMBIAR

Si miras al escenario y haces click en este bloque verás cómo aparece el valor de la luz en ese instante.

Como seguro que le has dado unas cuantas veces... habrás notado que fluctúa. Prueba a ver los valores con la luz encendida, apagada, iluminándolo con la linterna del móvil...

El rango va de 0 a 1023, aunque en nuestras condiciones normalitas no llegamos a los extremos.

Hagamos que el robot mire de forma continua el valor de la luz



No es demasiado rápido para el sensor, que responde bien, pero nosotros no somos capaces de leerlo, pongamos un retardo.



Ya podemos leerlo tranquilamente. Fíjate de nuevo en los tres niveles:

### Programación-Electrónica-MundoReal

Hagamos un programa que produzca una alarma *si* hay luz o *si no* la hay.

Apaga y enciende la luz para detectar cuáles son los niveles que lee tu detector en esas dos situaciones. Imagina que sean 100 y 600. Para ver dónde ponemos el "corte" podemos hacer la media, nos sale 350. De esa forma, cuando la luz suba de 350 diremos que está encendida y cuando baje de ahí que esté apagada. Esta es una manera sencilla de elegir un umbral, hay otras, dependiendo de si prefieres tener falsos positivos o falsos negativos, por ejemplo.

## 6. ¡DECÍDETE!

Para tomar decisiones seguiremos siempre este esquema:

MIDE → DECIDE

Antes dijimos *si... y si no...*

Para eso tenemos los siguientes bloques



En el hexágono pondremos la **CONDICIÓN** (¿es igual a? ¿es mayor que?...)

Debajo pondremos el código que queremos que se ejecute cuando se cumple y en el siguiente hueco el código a ejecutar cuando no se cumple.

Por ejemplo, hagamos un programa que haga sonar un sonido cuando apagamos la luz.

Lo primero que se nos ocurre es esto



Tiene buena pinta, primero medimos y luego tomamos una decisión.

Ponle un disparador de evento encima (el de la bandera verde, por ejemplo) y ejecútalo. Enciende y apaga la luz y, enhorabuena... no funciona.

**NOTA:** Una forma estupenda de analizar un programa es fijarse en cómo no funciona.

Mira el valor de la variable en el escenario, enciende y apaga la luz de la habitación. ¿Ves que no cambia?

**NOTA:** Otra forma estupenda de analizar un programa es ejecutarlo mentalmente, como si uno fuera el robot.

**NOTA:** No solemos poner "igual a" porque es raro que el sensor dé justo un valor, así que miramos por encima o debajo de un umbral, o en un intervalo alrededor de un valor.

¡Seamos el robot! Paso a escuchar las instrucciones

1. Mira la luz y mete ese valor en la variable luz. ¡A la orden!
2. Piensa si la luz es menor que 350, y si es así, toca la nota C4.
3. FIN

Ya está, ninguna instrucción más. El valor de la luz sólo se mide UNA VEZ, en el momento del inicio, y con esa información se toma la decisión.

Convéncete por ti mismo. Ejecuta el programa con la luz encendida. Para el programa. Ejecútalo con la luz apagada. Verá que en el segundo caso, sí que toca la nota. Una sola vez, por cierto, ¿verdad?

**IMPORTANTE: Lo mejor y lo peor de programar es que el programa sólo hace lo que le dices**

Ya lo vimos con el movimiento, los LEDs...

- ¿Quieres que se pare? DILE que se pare
- ¿quieres que se apaguen? DILES que se apaguen

¿QUIERES QUE MIDA UNA Y OTRA VEZ EL VALOR DE LA LUZ? ¡¡DÍSELO!!



Ahora sí que funciona como queríamos (ya sabes, siempre funciona, siempre hace algo... igual no lo que tú quieres).

MIDE, DECIDE... otra vez... MIDE, DECIDE... otra vez...

De esta forma el programa está preparado para detectar un cambio si ocurre.

Pruébalo y mira cómo ahora sí que funciona, va dando pitidos uno detrás de otro cuando estás a oscuras porque mira una y otra vez y cada vez que ve oscuridad, suena.

**NOTA:** A mí me parece tarda mucho... Piensa que está el retardo del sensor en medir, de que la placa envíe al ordenador el valor, que se ejecute tu código, que se mande el orden de vuelta... Además en estos programas basados en Scratch los bucles y otras estructuras van despacio para que los usuarios lo entiendan mejor, pero a veces nos molesta. Si quieres que vaya más rápido, podrías poner el modo Turbo (en el menú edición) y cargar el programa en la placa... pero a mí me sigue pareciendo lento (frente a un Arduino normal).

Si miras cómo cambia el valor de la variable, verás cómo se mide justo al acabar la nota, así que eso nos impone una limitación a la velocidad de respuesta.

Si quieres que dé más de una nota o que se enciendan los LEDs, o que baile la conga... pues DÍSELO!!

En el hueco donde has puesto ese bloque puedes poner todos los que quieras... pruébalo.

También podría ser que cuando haya más luz quieras que pasen cosas, entonces tendrás que usar este bloque



## 7. QUE NOS CHOCAMOS...

¿Qué tal si hacemos un programa que sea capaz de parar antes de dársela con una pared?

Tenemos un sensor de ultrasonidos que nos da la distancia (te aconsejo enchufarlo en el puerto 3, para ajustarnos a los valores por defecto de las apps)

Parecen dos ojitos, pero uno es un emisor y otro un receptor de ultrasonidos. Mide el tiempo desde que emite un pulso hasta que recibe el eco y, sabiendo la velocidad del sonido, estima la distancia.



Con este bloque tendremos la distancia en centímetros, hasta un máximo de 400 (4 metros).

A distancias cortas no capta bien y, al parecer, podremos fiarnos a partir de 4 cm o así.

No siempre mide con acierto, los objetos pueden ser pequeños o no reflejar bien los ultrasonidos de vuelta... lo irás viendo.

Seguiremos nuestra estructura de siempre

MIDE, DECIDE

A ver qué te parece



Bien, ¿no?

No...

¿Por qué no?

Mira la distancia, si es menor que un valor se para. Estupendo, ¿no?

Nooooo.

Pregunta: ¿En algún momento decimos que AVANCE? Pues si no se lo dice nadie, NO anda

Venga otra propuesta



¿Mejor?

Tampoco

Sé el robot.

- Avanzo
- Miro la distancia (me sale 40)
- Si es menor que 30 paro (no es menor, no hago nada)
- Miro la distancia (me sale 28)
- Si es menor que 30 paro (sí es menor, paro)

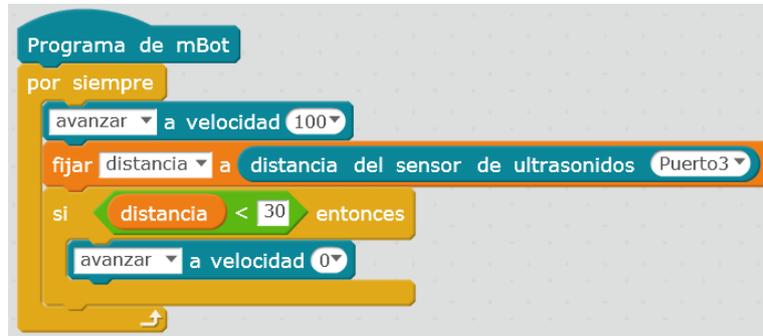
*Aparece alguien y me separa de la pared, yo me sigo ejecutando*

- Miro la distancia (me sale 32)

- Si es menor que 30 paro (no es menor, no mando la orden de parar... pero es que... ya estaba parado)

Este programa está mejor, pero una vez que se pare... no vuelve a arrancar, porque esa orden está fuera del bucle "por siempre".

Bueeeeno, otra propuesta



¿Mejor? Esteeee...

Vamos a ejecutarlo mentalmente.

Avanzo, miro la distancia, si es menor que treinta paro, vuelvo a avanzar, miro la distancia...

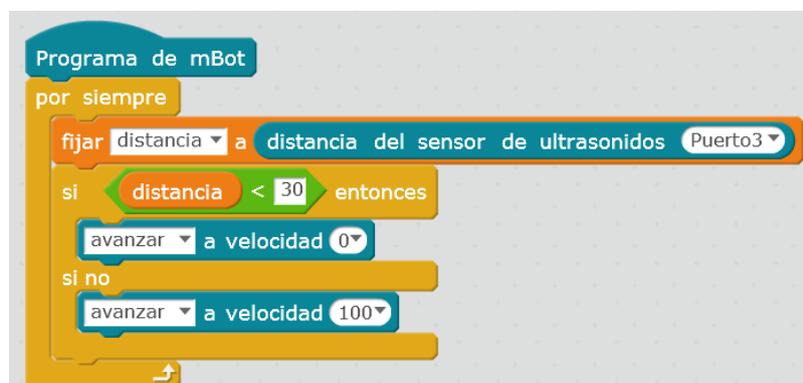
Está casi bien, si la distancia es larga no deja de avanzar, pero es que como antes de ejecutar cada ciclo MIDE, DECIDE le mandamos avanzar, nos da un "pasito". Si le ponemos delante de una pared hace lo siguiente

- Avanza (un pelín)
- Mira la distancia
- Si es menor que 30 manda parar
- Avanza (otro pelín)...

Y va dando pequeños pasitos acercándose a la pared.

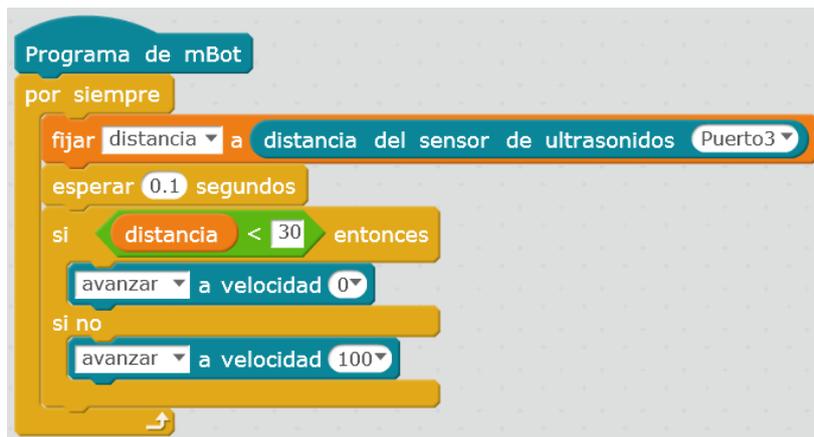
Hay una manera mejor de hacer esto... fíjate si casi nos lo dice el propio lenguaje natural.

*Quiero que si la distancia es menor que treinta se pare y si no avance*



**NOTA:** Los lenguajes de programación y las matemáticas son una formulación del “sentido común”, con la práctica puedes ir aprendiendo a traducir uno a otro con más facilidad.

Sólo nos queda un pequeño refinamiento y es el siguiente: este sensor necesita un momentito para poder tomar bien la medida y hacer los cálculos necesarios, así que justo después de medir y antes de aplicar el condicional pondremos una pequeña espera. Finalmente nos quedará así:



## 8. ¡QUE NOS CAEMOS!

Tenemos los sensores que llaman sigue-líneas. Son dos, cada uno una combinación de emisor y detector de infrarrojos. Si la luz rebota en un suelo reflectante, el sensor da un valor, si no rebota (porque es negro, o porque no hay suelo) entonces da otro valor. Por la parte de arriba se ilumina una luz azul que nos permite saber si el sensor está viendo “blanco” o “negro”.

Os aconsejo ponerlo al puerto 2 para los valores por defecto de las app. Podéis elegir lado derecho o izquierdo y que se evalúe si es blanco o negro.



**Uno esperaría que este bloque fuera un valor Verdadero-Falso pero meted el bloque en una variable y veréis que NO.**

El seguidor de línea tiene “mezclados” los valores de ambos sensores. Ya pongas derecho, izquierdo, blanco o negro. Y da los siguientes valores:

SENSOR IZQUIERDO	SENSOR DERECHO	VALOR
BLANCO	BLANCO	3
BLANCO	NEGRO	2
NEGRO	BLANCO	1
NEGRO	NEGRO	0

Por lo tanto **pongas el bloque que pongas** si estás sobre negro ambos verás cero, y distinto de cero en otro caso.

Que si los evalúas como verdadero o falso, te dará verdadero (>0) salvo en el caso de que NINGUNO esté viendo blanco. **Resulta engañoso... mal.**

Por lo tanto preguntarte si un sensor es negro o blanco es en realidad preguntarte si “alguno de los dos es blanco” o si “alguno de los dos es negro”.

Esto para no caer nos es suficiente, pero no está bien diseñado.

Debería ser un valor booleano (verdadero o falso, 1 o 0) según fuera el sensor que hemos elegido del color que hemos elegido.



Hagamos el programa equivalente al que hicimos para la distancia



Para este sensor no necesitamos esperas, no hace falta meter nada en una variable, se evalúa una y otra vez.

Fíjate que cuando está el robot parado al borde del precipicio el programa sigue ejecutándose una y otra vez (si lo ves en mBlock estará rodeado de un halo). Programa en marcha, robot parado (no es contradictorio).

El programa manda órdenes redundantes tanto si estoy sobre el suelo como si me voy a caer: PARA, PARA, PARA... o AVANZA, AVANZA, AVANZA... pero eso me asegura que, al cambiar la situación, me llegará la otra orden para adaptarme.

## 9. SIGUE EL CAMINO DE BALDOSAS NEGRAS...

No es difícil pensar que el sensor seguidor de línea está pensado para... seguir líneas.

Hay varias formas de seguir líneas dependiendo de cómo sea la línea de gruesa, de las posiciones de los sensores, de si detectan Sí-No, o un nivel de luz, etc.

Por ejemplo, en nuestro caso, se puede usar este bloque que da los siguientes valores



SENSOR IZQUIERDO	SENSOR DERECHO	VALOR
NEGRO	NEGRO	0
NEGRO	BLANCO	1
BLANCO	NEGRO	2
BLANCO	BLANCO	3

Como nosotros tenemos una línea gruesa y unos sensores juntos nosotros querremos

SENSOR IZQUIERDO	SENSOR DERECHO	ACCIÓN
NEGRO	NEGRO	AVANZAR
NEGRO	BLANCO	G. IZQUIERDA
BLANCO	NEGRO	G. DERECHA
BLANCO	BLANCO	X

Ver blanco significa estar saliéndote por ese lado, así que hay que girar para el otro.

En el caso blanco-blanco es que nos hemos salido, podemos elegir pararnos, seguir con la última opción a ver si encontramos la línea de nuevo, o incluso programar un algoritmo de búsqueda. Dejémoslo en blanco a ver qué pasa.



Por lo tanto podemos poner cuatro condicionales

Es curioso ver cómo se “reengancha” en el caso blanco-blanco. Cuando se sale por la izquierda manda la orden de girar a la derecha, puede que la orden llegue tarde y se haya salido también el otro sensor, pero girando a la derecha se corregirá, así que **dejar la última opción en blanco actúa como un algoritmo de búsqueda.**

## 10. QUIÉN PUEDE APRETAR EL BOTÓN

Ya os contaba que en la adaptación de Arduino Uno, mCore, habían puesto un botón en la placa, que no hay que confundir con el botón RESET, que con la tapa de plástico han dejado oculto(!). Si queréis resetear el programa apagáis y encendéis con el botón deslizante.

A ese botón se le pueden asociar acciones tanto al ser pulsado como al liberarse con el siguiente bloque



Ahora, tened cuidado, el programa predeterminado YA TIENE ACCIONES ASOCIADAS, y si pulsáis sale corriendo el bicho... Así que insisto, yo meto el robot en una caja de plástico abierta por arriba y ahí voy probando cosas, así se me cae menos. Cuando quiero, ya lo saco.

Por lo tanto, para asociar acciones al botón, tendréis que generar un Programa de mBot.



Por ejemplo, este programa:

Se inicia, apaga los LEDs y queda esperando hasta que el botón de la placa se presiona, en ese momento los pone a ROJO y queda de nuevo esperando hasta que el botón se libera, y ahí los pone a VERDE. Termina en programa y quedan en VERDE (recuerda que no se hace nada más que lo que se manda)

Podéis poner esto como un **“botón de inicio”**. Empezáis cualquier programa con



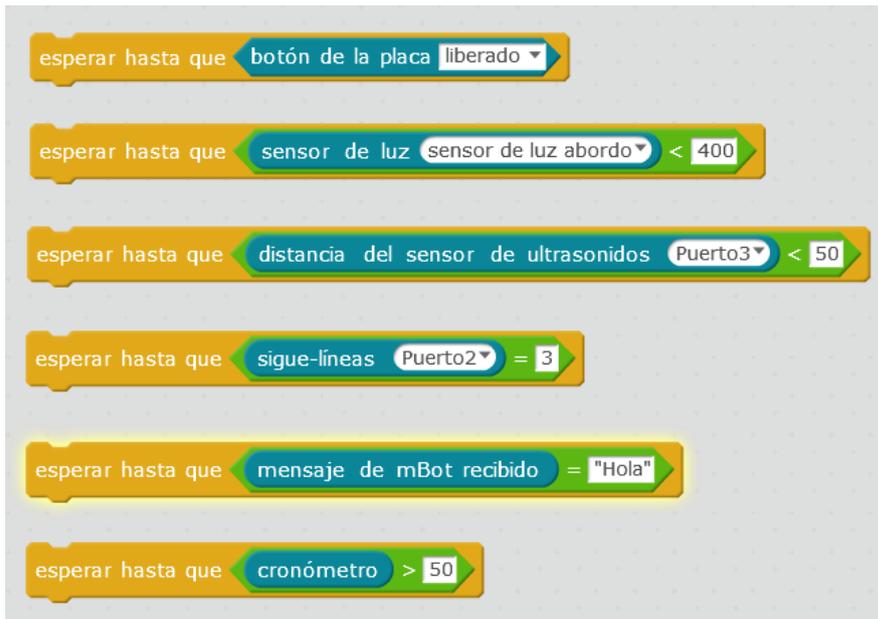
El coche no hará nada hasta que no pulséis el botón y luego os deja un segundito para quitar el dedo. Si queréis lo podéis adornar con una musiquilla y unos parpadeos para mayor teatralidad.

Os he colado este bloque que es muy interesante, pero tiene mucho peligro.



Podemos usarlo sin problemas con sensores porque va a estar esperando y mirando el sensor constantemente, cuando el sensor cambie, nos deja pasar.

Todos estos ejemplos funcionan estupendamente:



En cada uno la placa se encarga de buscar el sensor y tomar el dato.

Te he colado un par de cosas al final que seguro te imaginas como funcionan, luego te las cuento.

Pero en muchos casos no andamos siempre con el sensor, sino que usamos variables, entonces podría pasar algo así



Tiene mucho sentido, ¿no?

Avanza, mide la distancia, espera hasta que sea menor que 30, entonces sigue el programa y se para.

Pues no.

Imagina que cuando empezamos la distancia es de 50.

Así que ese será el valor que medimos y el valor que metemos en la variable distancia. Y esa variable **no vuelve a cambiar de valor**.

Comprobamos una y otra vez que la variable (no la distancia de verdad) no es menor que 30, pero no actualizamos el valor de esa variable. Así que la distancia de verdad va variando, pero no se refleja en la variable, por lo que el programa se queda parado allí.

Estos dos bloques que os pongo a continuación pueden parecer iguales pero son muy diferentes.

En el primero comparo una variable con un número, una variable que **no** estoy actualizando

En el segundo comparo el valor de un sensor (su valor instantáneo) con un número.



Si queremos poder hacer esa espera con la variable hay que hacerlo como hacen los niños cuando van de viaje.

Papá, ¿hemos llegado?

¿Y ahora?

¿Y ahora?

Miden una y otra vez. En nuestro caso sería así



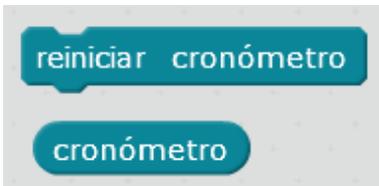
Actualizo el valor de la variable comparo con el umbral, si no, vuelvo a actualizar el valor de la variable... y así la variable sí va cambiando y reflejará el valor real, en lugar de un valor antiguo.

Esto es lo que se llama un bucle con condición de paro "Repíte HASTA que".

**NOTA:** No olvidéis que la variable sobre la que se aplica la condición de paro tiene que cambiar de alguna manera dentro del bucle (leyendo un sensor, con una operación, etc.), porque si no cambia, nunca saldréis del bucle. ¡Os quedaréis "embuclados"! Usad esto como un chequeo para ver que no os quedaréis embuclados.

## 11. EL TIEMPO ES ORO

Existe la posibilidad de usar estos dos bloques, de manera que tenemos una medida precisa del tiempo



Podéis reiniciar el cronómetro al comenzar el programa, y casi me apetece hacerlo sólo por “higiene” pero en general nos interesan los intervalos, así que muchos de nuestros resultados salen de restar valores y sería innecesario.

Un uso muy interesante del cronómetro es para no tener que parar el programa cada vez que queremos hacer una pausa en algún proceso. Por ejemplo

Si quiero hacer un cacharro que no se choque con las paredes ya sabemos que es así



Pero esto queremos repetirlo constantemente y lo pondremos dentro de un “por siempre”

Si a la vez que esto nos apetece que las luces vayan parpadeando y lo tenemos que hacer en un solo hilo para cargarlo como Programa de mBot, tenemos un problema. Mirad



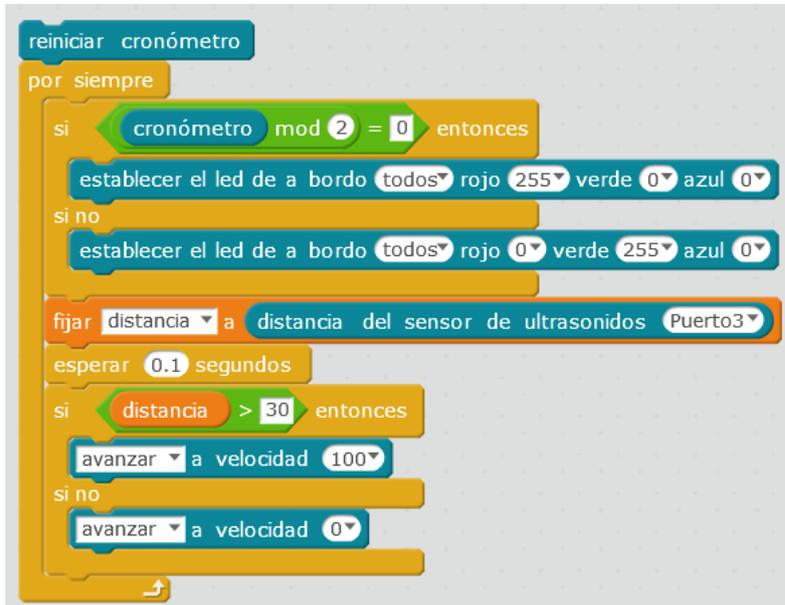
Antes de mirar la distancia y ver si nos la pegamos tenemos que perder dos segundos en el parpadeo.

La solución consiste en usar el cronómetro con un bloque como este, por ejemplo.

Si estamos en un segundo par me pongo a rojo, si el segundo es impar me pongo a verde



Esta función me da el resto de la división. Si es 0 es que es par, si es 1 es que es impar.



Reinicio el cronómetro. Miro si estoy en segundo par o no, tomo una decisión sobre la luz que pongo y bajo a mirar la distancia para no chocarme.

Repito de nuevo, y paso por todo el programa cada vez, sin esperas ni retardos. Será el cronómetro el que me indique el cambio de color en la vuelta que sea.

De la misma manera podéis mezclar hilos con sonidos y cosas así... no te digo que no queda mucho más sucio y más difícil de entender que el multihilo que nos permite mBlock mientras es interpretado por el Programa Predeterminado.

El cronómetro también nos permite apuntar tiempos en los que ocurren distintos sucesos. Por ejemplo, dejar el coche encendido en mi cuarto a oscuras y que me registre cuando suba la luz a qué hora ha ocurrido para saber quién cotillea mis cosas...

También puede limitar tiempos durante los que ocurren distintas fases de un programa. Por ejemplo, podemos hacer un buscador pero que cuando el tiempo pase de cierto valor se detenga.

No sé si mCore le tiene restricciones al cronómetro de Arduino, pero creo recordar que este tenía una duración enorme, así que no será un problema en principio que dejéis el programa corriendo durante días.

## 12. SUMA Y SIGUE

Ya te contaba que el siguiente bloque nos incrementa (o decrementa) la variable en la cantidad que ponemos en el círculo.



Puede ser una opción chula para ir contando cosas muy variadas:

- Las veces que nos hemos quedado a menos distancia que un valor (encuentros)
- O las que casi nos caemos (seguidor de línea)
- Las veces que se ha encendido la luz en una habitación
- Si pitamos líneas negras en el suelo, podemos ir contándolas y que funcionen como variables XY

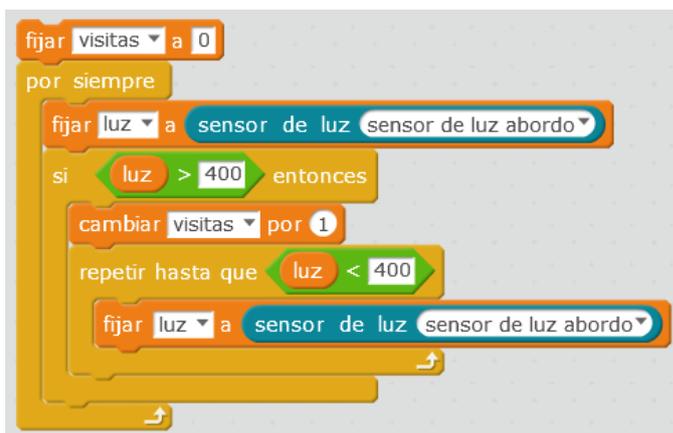
Os voy a hacer de ejemplo un programa que detecte las visitas no deseadas a mi habitación en mi ausencia



Tiene buena pinta... Pone las visitas a cero, mira la luz, si sube suma una visita. Estupendo... bueno, ya sabes que seguro que lo he puesto mal... ¿Ves dónde?

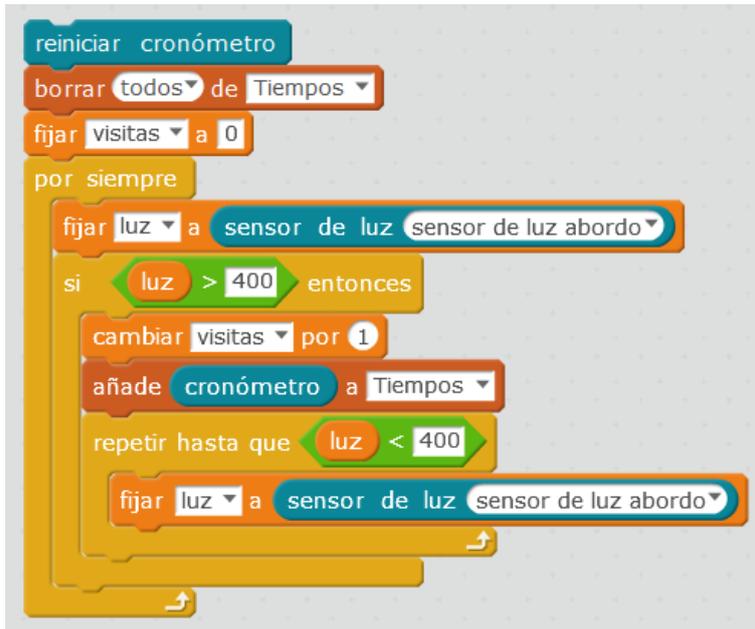
El bucle “por siempre” no deja de dar vueltas y va sumando visitas una y otra vez mientras esté la luz encendida.

Lo suyo es que sume una visita y espere hasta que baje la luz para esperar la siguiente visita



Ahora mejor, me quedo esperando en el “repetir hasta que” a que la luz vuelva a bajar, fíjate que dentro del bucle con condición de paro tengo que actualizar el valor de la variable luz, si no, no salgo nunca de ahí. Pruébalo si quieres y mira los valores de la luz en el escenario.

Si quieres apuntar los tiempos, puedes usar una LISTA. Lo tienes en el menú de variables y bloques, con las funciones esperables: añadir al final, insertar en una posición, borrar...



Empiezo poniendo todo en orden: Lista de tiempos vacía, cronómetro y visitas a cero.

Luego igual que antes, pero cuando sumo una visita, apunto el valor del cronómetro en la lista de tiempos.

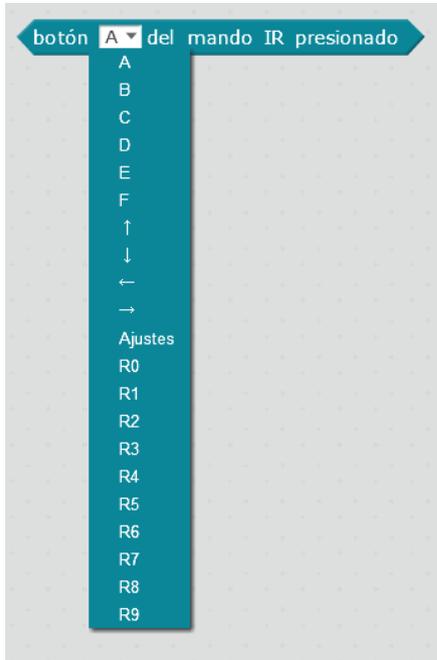
Si quieres que el tiempo aparezca en horas, con las funciones matemáticas lo tienes fácil.

Otro ejemplo podría ser ir sumando “encuentros” con las mismas precauciones. Si tienes un condicional para cuando la distancia sea menor que un valor, cuando se cumpla y apuntes un encuentro, tendrás que esperar un tiempo, o esperar a que la distancia pase de un mínimo, o algo para que no se dispare la variable, contando sin parar.

### 13. MANDO A DISTANCIA

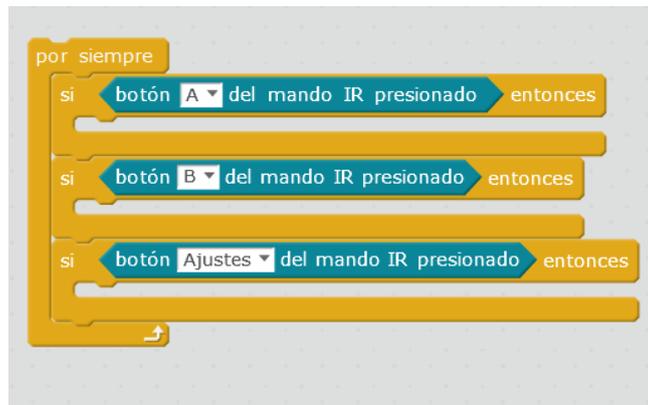
mBot viene equipado con un mando a distancia por infrarrojos.

El programa predeterminado tiene asociadas acciones a algunos de los botones, pero nosotros podemos redefinirlos. Para eso tendremos que hacerlo como Programa de mBot.



Que sean flechas o que ponga ajustes no tiene por qué condicionarnos para asociar las acciones que nos parezcan más convenientes.

De hecho, podríamos asociar a cada tecla un programa entero que llamaríamos a través de condicionales.



El programa está siempre mirando, cuando ve una tecla ejecuta todo el código que lleve ahí.

Recuerda que mientras esté ejecutando ese "subprograma" no estará mirando a ver si pulsas otro botón, no vas a interrumpirlo, la subrutina termina primero y después volverá a dar la vuelta al bucle "por siempre" y mirar a ver si estás pulsando alguna tecla. Si en esos programas tienes "esperas" fijas o condicionales, el coche se quedará en ese punto del código.

### 14. mBots CUCHICHEANDO...

mCore está equipada con un receptor de infrarrojos que le permite recibir órdenes del mando a distancia y de cualquier otro emisor de infrarrojos; quizá tú móvil, quizá el mando de tu tele. Puedes bajarte una aplicación y probar o te puede pasar por casualidad como me ha pasado a mí con el mando de mi tele (vaya susto me llevé... y casi se cae de nuevo. Insisto en la caja).

Resulta que mCore también tiene un emisor de infrarrojos que podéis usar para mandar mensajes que serán recibidos por el resto de mBots que estén por allí.

El receptor está en la parte delantera apuntando hacia delante y el emisor igual. Si hay una visual directa puede salvar un par de metros al menos.

En esta página os lo explican con detalle, os lo resumo.

[http://mrkalmes.weebly.com/uploads/2/2/0/1/22013084/mbot\\_lesson\\_18.pdf](http://mrkalmes.weebly.com/uploads/2/2/0/1/22013084/mbot_lesson_18.pdf)

Este es el bloque para enviar mensajes



enviar mensaje de mBot hola

Notad que es texto y que no va con comillas.

Nos dicen que no podemos usar como mensaje el valor de una variable. No pasa nada se trata de activar una respuesta en el otro, cualquier cosa vale, una letra, por ejemplo.

Dicen que para que se “oiga” bien, hay que mandarlo varias veces, yo he probado con dos veces y me funcionaba bien, en el ejemplo lo mandan como 50 y aconsejan hacer una pequeña espera cada 50. Supongo que dependerá de las condiciones del entorno, visibilidad, etc.

Para leer el mensaje recibido tenemos este otro bloque



mensaje de mBot recibido

Que lo querremos para compararlo y tomar una acción u otra, según el mensaje recibido.



esperar hasta que mensaje de mBot recibido = "Hola"

Notad las comillas. Ahora que me doy cuenta no he probado si distingue mayúsculas de minúsculas... os lo dejo como ejercicio, jeje. Aventuro que no, porque si miráis el código Arduino veréis que recoge carácter a carácter, y son diferentes. Ya me contaréis.

En el ejemplo del enlace hacen una espera hasta que el mensaje es distinto de vacío y luego condicionales sobre los distintos mensajes posibles.



esperar hasta que no mensaje de mBot recibido = ""



si mensaje de mBot recibido = "uno" entonces

Podría ser divertido para luchas o colaboración entre mBots.

Por ejemplo un enjambre de búsqueda que se para y va a algún sitio cuando uno de ellos encuentra y envía mensaje a todos.

## 15. GIRÓSCOPO

El giróscopo no viene con el mBot, pero puede comprarse aparte y tenemos puertos libres para tenerlo conectado de manera continua y completar el robot.

Mi interés es saber el ángulo en el plano XY, aunque también podría saberse si el coche se inclina a los lados o si está subiendo o bajando una cuesta.

Para eso podría haber pensado en el módulo brújula, pero tengo entendido que en funcionamiento y también por interferencias magnéticas puede tener errores de 10 o 15 grados. No sé, pero me pareció mucho.



Este es el bloque que nos da los tres ángulos.

La manera de medirlo es un poco extraña. En el eje Z entre -180 y 180 y en los otros dos ejes entre -90 y 90. Este último dato te debe hacer pensar que no puedes poner el giróscopo de cualquier manera, porque está considerando un ángulo y su opuesto iguales. En el eje Z si tenemos un número diferente para cada ángulo.

Cuando inicias la placa toma la posición como cero para los tres ángulos, pero esos números fluctúan y derivan... así que no puedes tomarlos como referencia absoluta. No puedes andar dando pirulos un rato y luego querer volver a la dirección 150 grados.

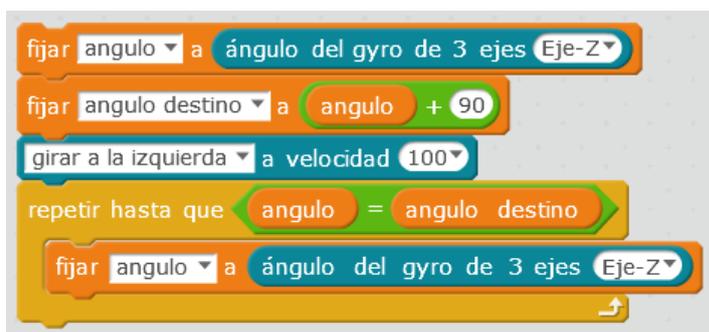
Pruébalo, crea tres variables para esos ángulos, ponle un “por siempre” y ponte a girar el robot para que veas lo que te cuento.

Puede que no se midan los ángulos, si te ocurre, vuelve a cargar el Firmware y el Programa Predeterminado. Si sigue sin funcionar, otra vez. A veces toma varias veces que se ponga como debe.

Entonces no lo usaremos para referencia absoluta, pero sí para una referencia relativa en un giro concreto. Lo haríamos así:

- Medir ángulo actual
- Calcular ángulo destino = ángulo actual + ángulo a girar
- Girar hasta llegar al ángulo destino

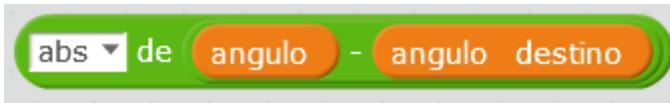
Por ejemplo, para girar 90 grados



Qué bonito sería todo si os pusiera bien desde el principio el código, ¿verdad?

Esto no funciona, no deja de girar... porque nunca nos va a coincidir el ángulo medido con todos los decimales exactamente con el ángulo objetivo.

Si en lugar de que sean iguales haces que se quede **a menos de un grado** (valor absoluto)



Pues tampoco...

Con un grado y medio de margen... unas veces sí, y otras no.

Con dos grados de margen sí que se para, pero ya tenemos un error ahí. No te preocupes que hay más.



Si ejecutáis esto veréis que hace claramente más de 90 grados, claramente.

Bueno, es culpa de que la señal de paro no es inmediata, hay una inercia y se pasa en torno a 5 grados. Total que entre unas cosas y otras tienes bastante error en el ángulo de giro.

La inercia la puedes calcular tomando una nueva medida del ángulo después de girar, verás que aunque hayas dado la orden cuando la resta era menor de dos grados, el ángulo final real y el ángulo destino se llevan unos grados.

Si le echas un rato puedes ajustar tu deriva, que no es constante, y tenerla en cuenta para que, cuando quieras girar 90, pues que ordenes un giro de ochenta y pico.

Hay un pequeño ajuste que hay que hacer al buscar el ángulo objetivo. Como te decía los ángulos van de -180 a 180, así que si estás en 150 y quieres girar 40 grados más tu objetivo no lo va a reconocer el giróscopo como 190, sino como -170. Te toca hacer la conversión cuando los ángulos sean menores que el mínimo o mayores que el máximo (simplemente restar o sumar 360)

Sí que es cierto que el giróscopo nos puede servir para hacer ángulos rectos medio respetables aunque las ruedas patinen un poco, pero no para mucho más, porque lo que yo quería era posicionarme en el plano.

Algo así como:

Tomar ángulo, por trigonometría calcular el incremento de X e Y (en unidades de segundos de avance) e ir actualizando esas variables para poder volver al origen después de una búsqueda aleatoria, por ejemplo. En mis ensayos, a los pocos trayectos acumulo un error tremendo... fíjate además que cuando las ruedas patinan tengo desplazamientos puros en el plano que no se tienen en cuenta.

Es posible que se pueda hacer mejor la cuenta subiendo y bajando cuestas y deducir la distancia avanzada, no la recorrida.

El modelo Ranger superior (como el doble de precio) incluye una configuración que se mantiene en equilibrio sobre dos ruedas. Estuve pensando cómo hacerlo con mBot pero el chasis va muy bajo para poder ponerse casi vertical y poder estar equilibrado. Supongo que con algunas piezas e imaginación pueda hacerse. Si lo hacéis, me contáis.

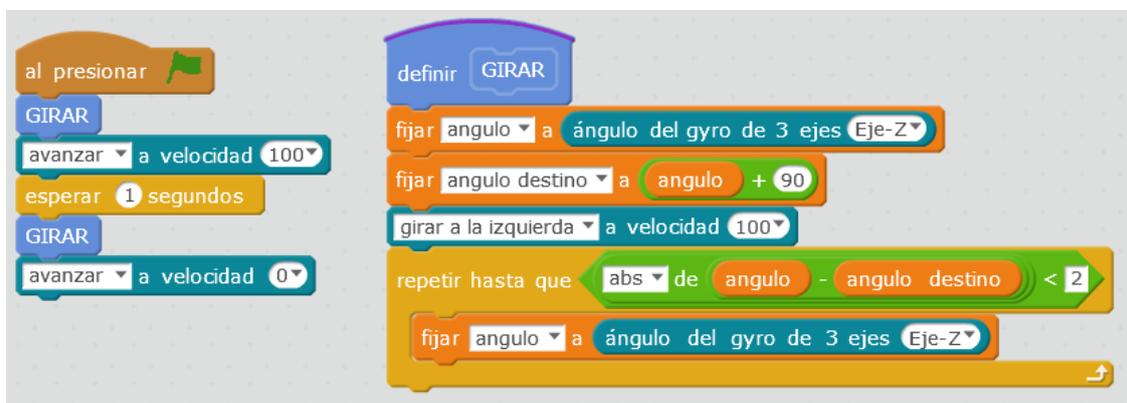
Para posicionarse en el plano quizá podría hacerse pintando una cuadrícula en el suelo y tomando el seguidor de línea como un contador de líneas que actuarían como unidades de X e Y. Aquí el problema sería la orientación y no cruzar líneas a la vez. Es posible que si nos moviésemos siempre siguiendo las líneas horizontales o verticales se pueda hacer con tanta precisión como líneas pongamos. Es posible que hiciera falta un segundo seguidor (para guiarse con uno y contar con otro) o a través de una LDR, o vigilando el ángulo respecto al eje Z para ir lo más recto posible.

## 16. Bloques

En cuanto un programa se sofisticaba un poco se hace poco legible y manejable, incluso a este nivel.

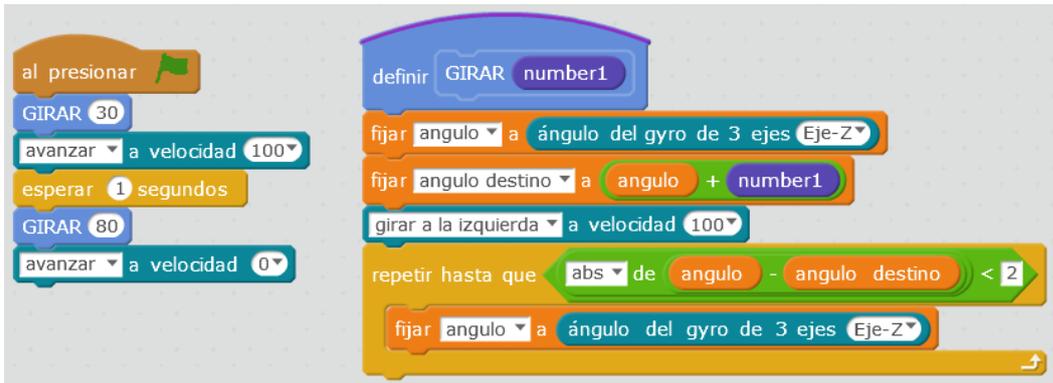
Ahí es donde entran las funciones, en este entorno de bloques, se trata de crear un bloque personalizado.

Agrupamos un poco de código y le damos un nombre que se entienda. Por ejemplo, lo que acabamos de contar sobre cómo girar...



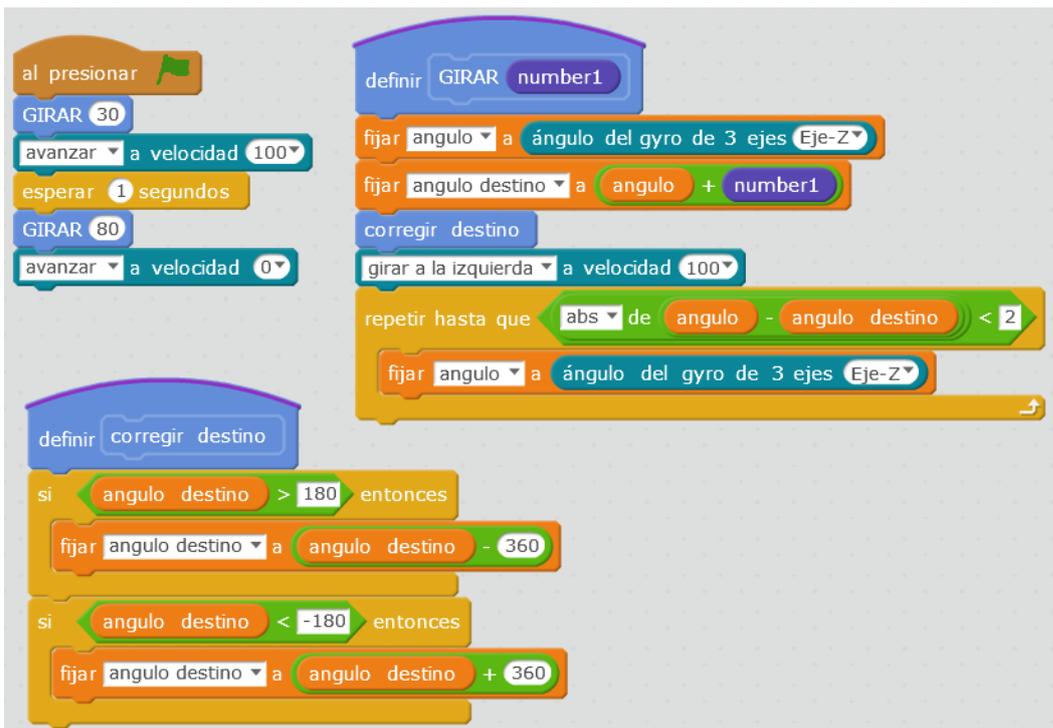
En el código principal se “invoca” dos veces la función. Se entiende mejor el código y es más fácil de programar.

Las funciones pueden tener parámetros, por ejemplo, hagamos un bloque genérico para girar cualquier ángulo, no sólo 90.



En el modo edición del bloque (o activándolo después con botón derecho sobre su título) tenemos las opciones. Podemos meterle varios parámetros de distintos tipos (numéricos, texto, etc.)

Podríamos llamar a unas funciones dentro de otras sin problema, por ejemplo para esa corrección que decíamos en los valores del ángulo de destino fuera del rango -180,180



Acostúmbrate a usar funciones y quizá así puedas entender tú mismo tus programas más allá de dos días después...

Imagina en el sencillo caso de arriba, meter todo en el código principal.

Hay que dejar claras varias cosas respecto de las funciones.

1. Las funciones no se ejecutan automáticamente, tienen que ser llamadas desde el código.

Me gusta decir a los chavales que una función es lo mismo que el hecho de que yo sepa fregar, o hacer torrijas, o bailar la conga. No lo estoy haciendo todo el rato, lo hago sólo cuando me lo piden... de buenas maneras.

Aprovechando la metáfora podríamos decir que una librería es un conjunto de habilidades, por ejemplo: Tareas del hogar. Si me voy a vivir con alguien, cargo la librería "Tareas del hogar" para cuando haya que invocar a alguna de sus funciones que estén allí.

2. Tener funciones no es lo mismo que la programación multihilo.

Precisamente porque no se ejecutan por sí solas, sólo cuando se llamen desde un hilo y, por lo tanto, forman parte de ese hilo. Así que se pueden usar sin problema incluso para cargar los programas autónomos como Programa de mBot.

**NOTA:** Desde Scratch, todos sus "hijos" adolecen de **no tener funciones con retorno**, que te devuelvan un valor. Funciones como esta que me invento "cambio\_unidades()" que podría llamarse así:

```
distancia_metros=cambio_unidades(distancia_millas)
```

La aplicación de la función devuelve un valor que se recoge en una variable, aquí en distancia\_metros.

No se puede.... Mal.

Para hacer algo parecido aquí hay que hacerlo en dos pasos.

Primero ejecutas tu bloque y metes el valor que quieres devolver en una variable temporal

Ahora pones un bloque de asignación para darle a tu variable el valor de la variable temporal.

Me da corajillo, porque creo que no sería tan difícil y no estorba. Otros lenguajes de bloques lo tienen.

Aprovecho para también echar de menos la función mapear.

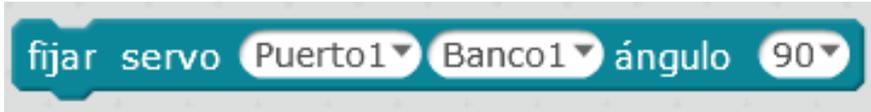
## 17. ÑACA ÑACA

También he comprado un extra que es una minipinza que se mueve con un servo.

Se puede poner en vertical con cierta facilidad en los salientes que tiene a los lados por delante sujetando con dos tornillos y tuerca.

Hay que comprar un módulo más para poder enchufar el servo allí y luego ir con un cable RJ a la placa. En ese módulo se podría enchufar otro servo (si quisierais hacer una “cabeza” para los ultrasonidos, por ejemplo).

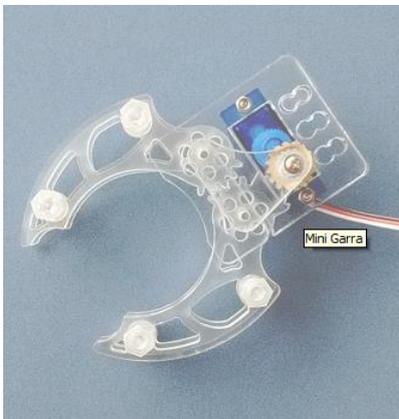
Se abre y cierra con facilidad y combinando con la información de distancia, se pueden hacer programas en los que se recoja algo. La verdad es que es vistoso y así se puede contar lo que es un servo y ver cómo funciona y cómo se programa.



El puerto es la conexión RJ y el banco es cada uno de las conexiones que puedes hacer en el módulo que necesitas para conectar el servo.



Delante, en blanco, los dos “bancos” (slots) para conectar hasta dos servos



Aunque el rango de un servo completo es de 0 a 180, por la forma de la garra abre y cierra entre 70 y 130 más o menos.

A veces por su propio mecanismo o por la carga que lleve, puede quedarse “zumbando” intentando alcanzar su ángulo objetivo sin lograrlo. Esto es un problema porque al funcionar por PWM ocupa el Timer y... bueno que no funcionarán cosas que necesitan el reloj, como la medida de ultrasonidos. Si le das un toquecito puedes pararla. Una manera de hacerlo por código, para los más avanzados sería

usar detach() después de darle su ángulo objetivo y esperar un tiempo prudencial.

Este mal funcionamiento del servo, o mi falta de habilidad, o mis ñapas por no comprar más piezas, o la rigidez del cable RJ... ha sido lo que me ha hecho desistir de poner los ultrasonidos sobre un servo y tener una “cabeza giratoria”. Al quedarse el servo “enganchado” me inutilizaba los ultrasonidos. Después de muchas pruebas he optado por girar el coche completo cuando quiero medir distancias.

## 18. PROPUESTAS

Aunque se han descrito y hecho algunos programas concretos me interesaba mucho más contar los elementos con los que se pueden construir, los errores más comunes, la forma de pensar. A partir de aquí las propuestas son millones, las que se os ocurran, las que veáis por ahí... esa es la potencia de conocer los elementos básicos.

Pongo algunas, por dar ideas.

1. Movimiento aleatorio
2. Movimiento aleatorio sobre una mesa, sin caerse  
Si ve que no se cae avanza o gira un tiempo aleatorio, si ve que está en el borde, retrocede gira un ángulo aleatorio y vuelve a empezar
3. Alarma de proximidad con pitidos (aparcamiento)
4. Ve un obstáculo y lo esquiva
5. Ve un obstáculo y lo esquiva de cualquier anchura, no conocida
6. Ve un obstáculo, lo esquiva y después recupera la línea original
7. Busca aleatoriamente (un patrón espiral, sobre una mesa sin caerse, etc.) hasta que encuentra algo (distancia corta)
8. Busca la distancia mínima y "ataca", sin salir de un círculo (SUMO)
9. Radar (pintando líneas en el escenario) Hay ejemplos en Internet
10. Salir del laberinto (girar siempre a un lado e ir siguiendo la pared a distancia fija)
11. Ir a un punto dado por sus coordenadas XY (pasar a distancia y ángulo y dirigirse allí)
12. Variantes de buscar cosas recojiéndolas con la garra

Hay otros módulos y otros robots que pueden comprarse, yo creo que con este modelo más el giróscopo y la garra tenemos un buen equipo para hacer prácticas variadas.

### FINALMENTE

Este es el fruto de unas cuantas horas de darse de cabezazos con hardware, software y textos... que han llegado a buen puerto gracias también a la ayuda de mucha gente y de toda la comunidad. La ciencia es una labor conjunta.

Quedan cosas en el tintero, pero no es este un documento que pretenda ser exhaustivo.

**La intención de estas prácticas es que puedan ser usadas con facilidad por personas sin conocimientos previos y, sobre todo, por profesores de secundaria.**

Espero que os sirvan y se agradecen comentarios que puedan mejorarlas. Iré actualizando el documento y lo tendréis a vuestra disposición en mi blog [Lacienciaparatodos.wordpress.com](http://Lacienciaparatodos.wordpress.com).

Javier Fernández Panadero