

JdeRobot-Kids: entorno docente de robótica con Arduino y Python

Autores:

Madrid, 2017

Resumen

En este manual no solo se describe cómo programar en cierto lenguaje de programación, también se detallan los recursos, procedimientos y prácticas para aprender a programar desde cero y para, finalmente, aplicarlo al área de Robótica. En concreto a cómo programar un robot Arduino usando el lenguaje Python.

Así como en los idiomas el hecho de conocer un vocabulario y una gramática no equivale a dominarlo, entender un lenguaje de programación implica además saber combinar sus elementos según la situación para producir instrucciones (frases) que expresen lo que uno quiere que haga el robot.

En un primer momento aprenderemos las palabras clave del lenguaje Python, las sentencias básicas y la sintaxis propias del mismo. Esto ayudará a entender cómo funcionan programas ya hechos. Los siguientes pasos serán aprender a programar en Python, como lenguaje de desarrollo de nuestros algoritmos robóticos. Es un lenguaje sencillo, intuitivo, y que no es compilado. Es interpretado, con lo que no necesita de un compilador, lo que ahorrará multitud de problemas.

Índice general

| | |
|--|-----------|
| Índice general | I |
| 1. Introducción | 1 |
| 1.1. Conceptos básicos: robots y programación | 3 |
| 1.2. La memoria: bits, bytes y palabras | 5 |
| 1.3. Identificadores | 5 |
| 1.4. Concepto de programa | 6 |
| 1.5. Lenguaje de pseudocódigo | 8 |
| 1.6. Algoritmo | 8 |
| 1.7. Concepto de función | 8 |
| 1.8. Nombres de variables y palabras claves en Python | 11 |
| 1.9. Sentencias | 13 |
| 1.10. Evaluación de expresiones | 13 |
| 1.11. Operadores y operandos | 14 |
| 2. Robots y su hardware | 15 |
| 2.1. Procesador Arduino | 15 |
| 2.2. Nociones de electrónica | 16 |
| 2.3. Sistemas electrónicos | 19 |
| 2.4. Componentes electrónicos del kit robótico Arduino | 21 |
| 2.5. Mbot | 26 |
| 2.6. PI-Mbot | 26 |
| 2.7. PiBot | 27 |
| 3. Introducción a la programación de robots con Scratch | 29 |
| 3.1. Lenguaje | 29 |
| 3.2. Instalación del entorno | 30 |
| 3.3. Ejercicios de lenguaje | 30 |
| 3.4. Ejercicios de pseudocódigo | 39 |
| 3.5. Mbot: prácticas básicas | 46 |
| 4. Introducción a la programación con lenguaje Python | 52 |
| 4.1. Ejercicios básicos | 52 |
| 4.2. Ejercicios avanzados | 55 |

| | |
|---|-----------|
| 4.3. Juegos básicos | 56 |
| 4.4. Ejercicios de listas | 57 |
| 5. Entorno de programación JdeRobot-Kids | 59 |
| 5.1. Arquitectura | 59 |
| 5.2. Instalación del entorno | 59 |
| 5.3. Interfaz Python para programar robots | 62 |
| 6. Programación en Python sobre Arduino: sensores y actuadores | 67 |
| 6.1. Lectura de valor de pulsador | 67 |
| 6.2. Control de LED mediante pulsador | 68 |
| 6.3. Control de servos gráficamente con librería Tk | 69 |
| 6.4. Uso del zumbador | 70 |
| 6.5. Uso del potenciómetro | 71 |
| 6.6. Motor DC | 72 |
| 6.7. Modulo LED | 73 |
| 6.8. Sensor de luz | 75 |
| 6.9. Infrarrojos | 75 |
| 6.10. Comunicación vía Bluetooth | 76 |
| 6.11. Lectura de ultrasonidos | 77 |
| 6.12. Lectura de intensidad de sonido | 77 |
| 6.13. Matriz de LEDs | 78 |
| 6.14. Garra | 81 |
| 6.15. Garra montada sobre un mini cuello Pan-Tilt | 82 |
| 6.16. Mando a distancia | 83 |
| 6.17. Ejercicios | 84 |
| 7. Programación en Python sobre Arduino: Comportamientos en robots | 85 |
| 7.1. Choca gira | 85 |
| 7.2. Palmadas | 86 |
| 7.3. ¡Huye de la luz! | 88 |
| 7.4. Lucha de sumos | 90 |
| 7.5. Piedra, papel o tijera | 92 |
| 7.6. Ejercicios | 94 |
| 8. Programación en Python sobre PiBot: sensores y actuadores | 96 |
| 8.1. Sensor de contacto con LED | 96 |
| 8.2. Ultrasonidos básico de cuatro pines | 97 |
| 8.3. Ultrasonidos avanzado de tres pines | 98 |
| 8.4. Emisor de infrarrojos | 99 |
| 8.5. Receptor de infrarrojos | 100 |
| 8.6. Control de servo básico | 101 |
| 8.7. Servo avanzado de rotación continua | 102 |

| | |
|---|------------|
| 9. Programación en Python sobre PiBot: Comportamientos en robots | 105 |
| 9.1. Follow Ball | 105 |
| 9.2. Choca-gira usando visión | 110 |
| 9.3. Choca-gira usando ultrasonidos | 112 |
| 9.4. Sigue-líneas usando infrarrojos | 114 |
| 9.5. Sigue-líneas usando visión | 115 |

Capítulo 1

Introducción

Hoy día hay cada vez más aplicaciones robóticas para el gran público. Más allá de las clásicas aplicaciones en entornos industriales y ensamblado de vehículos se utilizan robots, por ejemplo, en el envasado de alimentos o la gestión de almacenes (como los centros logísticos de Amazon). Las aspiradoras robóticas (Roomba como pionera) han supuesto un éxito sin precedentes resolviendo una necesidad del mercado doméstico con robots autónomos. También los coches incorporan cada vez más tecnología robótica, como el aparcamiento automático o asistentes de conducción autónoma (autopiloto de Tesla). Los grandes fabricantes de automoción están empujando estos nuevos avances, tienen prototipos avanzados de coches autónomos y empresas de software como Google o Apple se han posicionado muy bien. Igualmente las aplicaciones de los drones, de los robots aéreos, están en pleno crecimiento.



Figura 1.1: Coche autónomo de Google y aspiradora robótica Roomba

Uno de los factores que permiten a los robots desplegar inteligencia y desenvolverse en situaciones reales ofreciendo robustez similar a la humana es su software, su programación. Debajo de todas las aplicaciones robóticas que están llegando al mercado de masas hay un software en el que reside gran parte de la inteligencia de los robots. Típicamente este software para robots tiene varias capas (drivers, middleware y aplicaciones) y presenta unos requisitos específicos distintos del software para otros campos: funcionamiento en tiempo real, robustez, distribuidos, hardware heterogéneo. Uso del interfaz gráfico para depurar, pero pocas veces en tiempo de ejecución (embarcados).



Figura 1.2: Robots mueven mercancías en un almacén de Amazon y drone empleado para inspeccionar una plantación

La robótica es un campo transversal donde concurren muchas tecnologías: electrónica, mecánica, informática, telecomunicaciones... Actualmente la formación en robótica aparece en secundaria y se realiza fundamentalmente en la universidad, con titulaciones de grado y postgrados específicos.

En secundaria la educación en robótica está cobrando una importancia creciente. Tiene poder de motivación en los estudiantes y eso permite acercar la tecnología a los niños, usando la robótica como herramienta para exponerles a conceptos básicos de ciencias, tecnología, ingeniería y matemáticas (*STEM Science, Technology, Engineering and Math*). Como ejemplo en esta dirección, la Comunidad de Madrid (Decreto 48/2015) ha introducido recientemente la asignatura *Tecnología, programación y robótica* en el currículum oficial de Educación Secundaria Obligatoria.

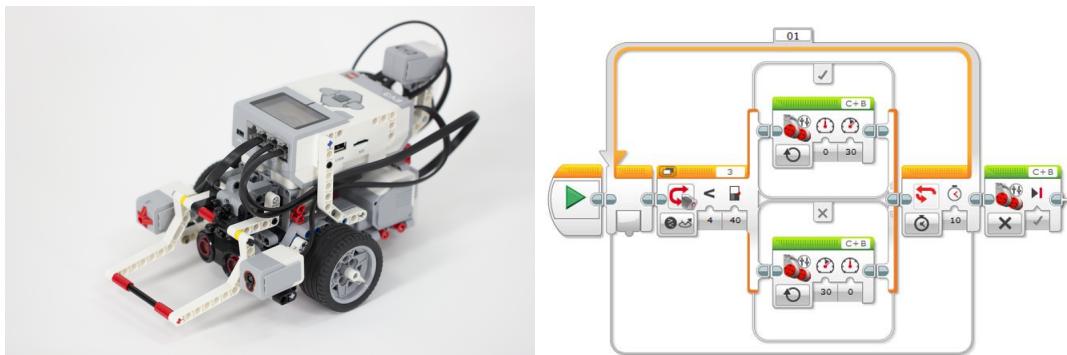


Figura 1.3: Robot LEGO Mindstorm Ev3 y programa para él en lenguaje gráfico

En la práctica de la enseñanza robótica en educación secundaria son frecuentes plataformas como los robots LEGO (Mindstorm, NXT, Ev3, WeDo), placas con procesadores Arduino a los que se conectan sensores de bajo coste y servos. Se enseña el funcionamiento básico de sensores, actuadores y los rudimentos de la programación. Se usan lenguajes sencillos que facilitan su programación por niños, como los lenguajes gráficos RCX-code y recientemente Scratch o Blockly.

Para este curso hemos elegido como plataforma hardware un robot Arduino, al que se le pueden conectar numerosos sensores y actuadores de bajo coste. Es una plataforma muy



Figura 1.4: Programa de ejemplo en el lenguaje gráfico Scratch

usada en la docencia de robótica en secundaria y bachillerato, permite la interacción de los estudiantes con un robot real, sensores reales y actuadores reales a un coste asumible. Ofrece muchas posibilidades didácticas. En vez de programarlo en el lenguaje nativo de Arduino, hemos escogido Python como lenguaje de programación, por su simplicidad, su potencia expresiva y porque se usa mucho en niveles más altos de educación y programación, por ejemplo en la universidad.

Otra muestra de la importancia creciente de la robótica en la educación son los campeonatos robóticos para adolescentes, que motivan el interés por la tecnología. Por ejemplo el campeonato robótico RoboCampeones en la Comunidad de Madrid concitó a más de 1200 estudiantes en la última edición, con pruebas como el sigue líneas, el sumo entre robots, etc.. Y a nivel internacional destaca la RoboCup Junior como pruebas como el rescate o el fútbol robótico 1.5.

1.1. Conceptos básicos: robots y programación

Dado su carácter transversal, la robótica se puede ver desde muchas perspectivas. Un enfoque muy útil es verla como la conjunción de sensores, actuadores conectados a uno o más computadores. Da igual la apariencia final del robot: un robot humanoide, un drone, un coche autónomo, un robot aspiradora... Todos tienen estos componentes.

$$\text{Robot} = \text{computador} + \text{sensores} + \text{actuadores}$$

Los *sensores* son dispositivos que permiten al robot medir alguna característica del entorno o suya. Por ejemplo hay sensores de distancia a obstáculos, de nivel batería, cámaras, sensores de ultrasonido, de luz, de posición, etc.

Los *actuadores* son dispositivos que permiten al robot interactuar con el entorno, hacer algo, ejecutar alguna acción o moverse. Por ejemplo hay actuadores que son motores (asociados a las ruedas del robot o sus articulaciones). Los altavoces, las luces o un zumbador también son ejemplos de actuadores.



Figura 1.5: Campeonato robótico RoboCampeones y campeonato internacional RoboCup Junior

Hoy en día los *computadores* están presentes en prácticamente todas partes: cajeros, teléfonos móviles, comercios, automóviles, electrodomésticos, casas domotizadas, y un largo etcétera. La diferencia principal entre estos sistemas y un ordenador personal radica, sencillamente, en el carácter especializado de los primeros frente al propósito general de éste último. Todos tienen en común la incorporación de un procesador o chip electrónico. Éste hará las veces de cerebro del sistema, siendo nosotros, como programadores, los encargados de aprovecharlo para que intérprete las instrucciones que demos a la hora de conferir un determinado comportamiento en las máquinas.

Los computadores manejan información digital. Una canción, una carta, un mensaje de whatsapp, una película, los datos de los sensores, se digitalizan y entonces los ordenadores pueden manejarlos, transmitirlos y analizarlos. Los computadores incluyen en su hardware el *microprocesador* (CPU), la *memoria* y *periféricos* como el disco duro, la tarjeta de comunicaciones en red –p.e. wifi– o el *bluetooth*. El hardware es la parte tangible, física, de un computador. Su inteligencia no obstante reside en su software, en su parte lógica.

$$\text{Computador} = \text{CPU} + \text{memoria} + \text{perifericos} + \text{software}$$

El software consta básicamente de programas y datos. Los programas se escriben en un lenguaje de programación. Estos lenguajes pueden ser gráficos (con bloques visuales

que conectan) o basados en texto. Los lenguajes textuales suelen ser más potentes. Hay software en muchos campos: software para bases de datos, software de comunicaciones, software de inteligencia artificial, software de ofimática, software de juegos, software para aplicaciones web, para aplicaciones en los teléfonos móviles, etc.. El software es la manera de hacer que las máquinas hagan lo que es útil a las personas. También en la robótica el software juega un papel importante, es donde reside la inteligencia de los robots.

$$\text{Software} = \text{programas} + \text{datos}$$

1.2. La memoria: bits, bytes y palabras

La memoria de un computador está constituida por un gran número de unidades elementales, llamadas bits, que contienen unos ó ceros. Un bit aislado tiene muy escasa utilidad; un conjunto adecuado de bits puede almacenar casi cualquier tipo de información. Para facilitar el acceso y la programación, casi todos los ordenadores agrupan los bits en conjuntos de 8, que se llaman bytes u octetos. La memoria se suele medir en Kbytes (1024 bytes), Mbytes o simplemente megas (1024 Kbytes), Gbytes o gigas (1024 Mbytes) y Tbytes o teras (1024 Gbytes).

Como datos significativos, puede apuntarse que un PC estándar actual, tendrá entre 4 y 16 Gbytes de RAM, y entre 500 y 2000 Gbytes de disco. Siendo los discos duros SSD dispositivos con una nueva técnica de almacenamiento y lectura/escritura, cuya capacidad es notablemente inferior a los ordinarios pero su velocidad, mayor. El que la CPU pudiera acceder por separado a cada uno de los bytes de la memoria resultaría antieconómico. Normalmente se accede a una unidad de memoria superior llamada palabra (word), constituida por varios bytes. En los PCs antiguos la palabra tenía 2 bytes (16 bits); a partir del procesador Intel 386 la palabra tiene 4 bytes (32 bits), y actualmente es normal que tengan palabras de 8 bytes (64 bits).

Hay que señalar que la memoria de un ordenador se utiliza siempre para almacenar diversos tipos de información. Quizás la distinción más importante que ha de hacerse es entre datos y programas. A su vez, los programas pueden corresponder a aplicaciones (programas de usuario, destinados a una tarea concreta), o al propio sistema operativo del ordenador, que tiene como misión el arrancar, coordinar y cerrar las aplicaciones, así como mantener activos y accesibles todos los recursos del ordenador.

1.3. Identificadores

Como se ha dicho, la memoria de un computador consta de un conjunto enorme de palabras, en el que se almacenan datos y programas. Las necesidades de memoria de cada tipo de dato no son homogéneas (por ejemplo, un carácter alfanumérico ocupa un byte,

mientras que un número real con 16 cifras ocupa 8 bytes), y tampoco lo son las de los programas. Además, el uso de la memoria cambia a lo largo del tiempo dentro incluso de una misma sesión de trabajo, ya que el sistema reserva o libera memoria a medida que la va necesitando.

Cada posición de memoria puede identificarse mediante un número o una dirección, y éste es el modo más básico de referirse a una determinada información. No es, sin embargo, un sistema cómodo o práctico, por la nula relación nemotécnica que una dirección de memoria suele tener con el dato contenido, y porque -como se ha dicho antes- la dirección física de un dato cambia de ejecución a ejecución, o incluso en el transcurso de una misma ejecución del programa. Lo mismo ocurre con partes concretas de un programa determinado.

Dadas las citadas dificultades para referirse a un dato por medio de su dirección en memoria, se ha hecho habitual el uso de identificadores. Un *identificador* es un nombre simbólico que se refiere a un dato o programa determinado. Es muy fácil elegir identificadores cuyo nombre guarde estrecha relación con el sentido físico, matemático o real del dato que representan. Así por ejemplo, es lógico utilizar un identificador llamado `salario_bruto` para representar el coste anual de un empleado. El usuario no tiene nunca que preocuparse de direcciones físicas de memoria: el sistema se preocupa por él por medio de una tabla, en la que se relaciona cada identificador con el tipo de dato que representa y la posición de memoria en la que está almacenado.

Python, como todos los demás lenguajes de programación, tiene sus propias reglas para elegir los identificadores. Los usuarios pueden elegir con gran libertad los nombres de sus variables y programas, teniendo siempre cuidado de respetar las reglas del lenguaje y de no utilizar un conjunto de palabras reservadas (*keywords*), que son utilizadas por el propio lenguaje. Más adelante se explicarán las reglas para elegir nombres y cuáles son las palabras reservadas del lenguaje Python. Baste decir por ahora que todos los identificadores que se utilicen han de ser declarados por el usuario, es decir, hay que indicar explícitamente qué nombres se van a utilizar en el programa para datos y funciones, y qué tipo de dato va a representar cada uno de ellos. Más adelante se volverá sobre estos conceptos.

1.4. Concepto de programa

Un programa -en sentido informático- está constituido por un conjunto de instrucciones que se ejecutan de modo secuencial, es decir, cada una a continuación de la anterior. Aunque, desde hace algunos años, ya contamos con procesadores capaces de ejecutar instrucciones de forma paralela para disminuir los tiempos de ejecución de programas

críticos por su tamaño o complejidad. La programación paralela es mucho más complicada que la secuencial y no se hará referencia a ella en este curso.

Análogamente a los datos que maneja, las instrucciones que un procesador digital es capaz de entender están constituidas por conjuntos de unos y ceros. A esto se llama lenguaje de máquina o binario, y es muy difícil de manejar para los humanos. Por ello, desde casi los primeros años de los ordenadores, se comenzaron a desarrollar los llamados lenguajes de alto nivel (tales como C, C++, Java, Arduino, Python, etc.), que están mucho más cerca del lenguaje natural. Estos lenguajes están basados en el uso de identificadores, tanto para los datos como para las componentes elementales del programa, que en algunos lenguajes se llaman rutinas o procedimientos, y que en Python se denominan funciones. Además, cada lenguaje dispone de una sintaxis o conjunto de reglas con las que se indica de modo inequívoco las operaciones que se quiere realizar.

Los lenguajes de alto nivel son más o menos comprensibles para el usuario, pero no para el procesador. Normalmente, para que éste pueda ejecutarlos es necesario traducirlos a su propio lenguaje de máquina. El programa de alto nivel se suele almacenar en uno o más ficheros llamados ficheros fuente, que en casi todos los sistemas operativos se caracterizan por una terminación -también llamada extensión- especial. Así, por ejemplo, todos los ficheros fuente de Arduino deben terminar por (.ino); ejemplos de nombres de estos ficheros son *servos.ino*, *manejoleds.ino*, etc.. Hay lenguajes de alto nivel compilados y otros interpretados.

En los lenguajes compilados la traducción a lenguaje de máquina la realiza un programa especial llamado *compilador*. La primera tarea del compilador es realizar una traducción directa del programa a un lenguaje más próximo al del computador (llamado lenguaje ensamblador), produciendo un fichero objeto con el mismo nombre que el fichero original, pero con la extensión (.obj). En una segunda etapa se realiza el proceso de montaje (enlazado) del programa, consistente en producir un programa ejecutable en lenguaje de máquina, en el que están ya incorporados todos los otros módulos que aporta el sistema sin intervención explícita del programador (funciones de librería, recursos del sistema operativo, etc.). En un PC con sistema operativo Windows el programa ejecutable se guarda en un fichero con extensión .exe. Este fichero es cargado por el sistema operativo en la memoria RAM cuando el programa va a ser ejecutado.

En los lenguajes interpretados existe un programa, llamado *intérprete*, que va traduciendo al vuelo el fichero con el programa en algo nivel, ejecutándolo línea a línea. Python es un lenguaje interpretado, lo cual nos ahorra mucho tiempo durante el desarrollo ya que no es necesario compilar ni enlazar. El intérprete puede usarse interactivamente, lo que facilita experimentar con características del lenguaje, escribir programas descartables, o probar funciones cuando se hace desarrollo de programas de abajo hacia arriba. Python permite escribir programas compactos y legibles; son típicamente más cortos que sus programas equivalentes en C, C++ o Java.

A medida que la complejidad de las tareas que realizaban las computadoras aumentaba,

se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de alto nivel, como el que veremos nosotros más adelante: Python (ver Capítulo 4). Mientras que una tarea tan trivial como multiplicar dos números puede necesitar un conjunto de instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará con sólo una.

Pero antes de eso, nosotros empezaremos a usar el lenguaje visual Scratch (ver Capítulo 3).

1.5. Lenguaje de pseudocódigo

Al contrario que los anteriores lenguajes, el pseudocódigo (o falso lenguaje) es comúnmente utilizado por los programadores para omitir secciones de código o para dar una explicación del paradigma que tomó el mismo programador para hacer sus códigos, esto quiere decir que el pseudocódigo no es programable sino facilita la programación.

El principal objetivo del pseudocódigo es el de representar la solución a un algoritmo de la forma más detallada posible, y a su vez lo más parecida posible al lenguaje que posteriormente se utilizará para la codificación del mismo

El pseudocódigo utiliza para representar las acciones sucesivas palabras reservadas en inglés (similares a sus homónimos en los lenguajes de programación), tales como star, begin, end, stop, if-then-else, while, repeat-until, etc. Es un lenguaje de especificación de algoritmos. El uso de tal lenguaje hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil.

1.6. Algoritmo

Un algoritmo es un conjunto pre-escrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución.

1.7. Concepto de función

Las aplicaciones informáticas que habitualmente se utilizan, incluso a nivel de informática personal, suelen contener decenas y aún cientos de miles de líneas de código fuente. A medida que los programas se van desarrollando y aumentan de tamaño, se convertirían rápidamente en sistemas poco manejables si no fuera por la *modularización*, que es el proceso consistente en dividir un programa muy grande en una serie de módulos mucho

más pequeños y manejables.

A estos módulos se les ha solido denominar de distintas formas (subprogramas, subrutinas, procedimientos, funciones, etc.) según los distintos lenguajes. El lenguaje Python hace uso del concepto de función. Sea cual sea la nomenclatura, la idea es sin embargo siempre la misma: dividir un programa grande en un conjunto de subprogramas o funciones más pequeñas que son llamadas por el programa principal; éstas a su vez llaman a otras funciones más específicas y así sucesivamente.

La división de un programa en unidades más pequeñas o funciones presenta -entre otras- las ventajas siguientes:

- Modularización. Cada función tiene una misión muy concreta, de modo que nunca tiene un número de líneas excesivo y siempre se mantiene dentro de un tamaño manejable. Además, una misma función (por ejemplo, un producto de matrices, una resolución de un sistema de ecuaciones lineales, ...) puede ser llamada muchas veces en un mismo programa, e incluso puede ser reutilizada por otros programas. Cada función puede ser desarrollada y comprobada por separado.
- Ahorro de memoria y tiempo de desarrollo. En la medida en que una misma función es utilizada muchas veces, el número total de líneas de código del programa disminuye, y también lo hace la probabilidad de introducir errores en el programa.
- Independencia de datos y ocultamiento de información. Una de las fuentes más comunes de errores en los programas de computador son los efectos colaterales o perturbaciones que se pueden producir entre distintas partes del programa. Es muy frecuente que al hacer una modificación para añadir una funcionalidad o corregir un error, se introduzcan nuevos errores en partes del programa que antes funcionaban correctamente. Una función es capaz de mantener una gran independencia con el resto del programa, manteniendo sus propios datos y definiendo muy claramente la interfaz o comunicación con la función que la ha llamado y con las funciones a las que llama, y no teniendo ninguna posibilidad de acceso a la información que no le compete.

Nombre, valor de retorno y argumentos de una función

Las funciones suelen tener argumentos de entrada y de salida. En la entrada se le pasan los datos con los que se quiere que opere esa vez la función. En la salida se recoge el valor entregado por ella como resultado.

En Python, la palabra reservada *def* se usa para definir funciones. Debe seguirle el nombre de la función y la lista de parámetros formales entre paréntesis. Las sentencias que forman el cuerpo de la función empiezan en la línea siguiente, y deben estar con sangría.

```

def fib(n): # escribe la serie de Fibonacci hasta n
    """Escribe la serie de Fibonacci hasta n."""
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

# Ahora llamamos a la funcion que acabamos de definir:
fib(2000)

# Y la salida sería: 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

```

La primera sentencia del cuerpo de la función puede ser opcionalmente una cadena de texto literal; esta es la cadena de texto de documentación de la función, o **docstring**. Hay herramientas que usan las **docstrings** para producir automáticamente documentación en línea o imprimible, o para permitirle al usuario que navegue el código en forma interactiva; es una buena práctica incluir **docstrings** en el código que uno escribe, por lo que se debe hacer un hábito de esto.

La ejecución de una función introduce una nueva *tabla de símbolos* usada para las variables locales de la función. Más precisamente, todas las asignaciones de variables en la función almacenan el valor en la tabla de símbolos local; así mismo la referencia a variables primero mira la tabla de símbolos local, luego en la tabla de símbolos local de las funciones externas, luego la tabla de símbolos global, y finalmente la tabla de nombres predefinidos. Así, no se les puede asignar directamente un valor a las variables globales dentro de una función (a menos se les nombre en la sentencia global), aunque si pueden ser referenciadas.

Los parámetros reales (argumentos) de una función se introducen en la tabla de símbolos local de la función llamada cuando esta es ejecutada; así, los argumentos son pasados por valor (dónde el valor es siempre una referencia a un objeto, no el valor del objeto). Cuando una función llama a otra función, una nueva tabla de símbolos local es creada para esa llamada.

La definición de una función introduce el nombre de la función en la tabla de símbolos actual. El valor del nombre de la función tiene un tipo que es reconocido por el interprete como una función definida por el usuario. Este valor puede ser asignado a otro nombre que luego puede ser usado como una función. Esto sirve como un mecanismo general para renombrar:

```

fib
<function fib at 10042ed0>
f = fib

```

```
f(100)
# La salida sería: 1 1 2 3 5 8 13 21 34 55 89
```

Las funciones se suelen agrupar en bibliotecas. Hay muchas bibliotecas útiles que ya han construido otros programadores. Tu programa en Python puede importar una cierta biblioteca y así usar las funcionalidades que incluye. No tienes que reinventar la rueda, usa bibliotecas existentes y así podrás avanzar más rápidamente en tu aplicación aprovechando su funcionalidad.

1.8. Nombres de variables y palabras claves en Python

Los nombres de variables válidos en Python debe ajustarse a las siguientes tres simples reglas:

1. Son secuencias arbitrariamente largas de letras y dígitos.
2. La secuencia debe empezar con una letra.
3. Además de a...z, y A...Z, el guión bajo (_) es una letra.

Los programadores generalmente escogen nombres significativos para sus variables, que especifiquen para qué se usa la variable. Aunque está permitido usar letras mayúsculas, por convención no se hace. Las usemos o no, hemos de tener en cuenta que las letras mayúsculas importan; por ejemplo, Julio y julio son variables diferentes.

El carácter subrayado (_) puede aparecer en un nombre. A menudo se usa en nombres con múltiples palabras, tales como *mi_nombre*. Hay algunas situaciones en las que los nombres que comienzan con un guión tienen un significado especial, por lo que una regla segura para los principiantes es empezar todos los nombres con una letra que no sea un guión. Si le damos un nombre inválido a una variable obtendremos un error de sintaxis:

```
31galletas = "gran merienda"
SyntaxError: invalid syntax
class = "Sala de ordenadores"
SyntaxError: invalid syntax
```

31galletas es inválido porque no empieza con una letra. En cuanto a la palabra *class* resulta ser una de las palabras claves de Python. Las palabras claves definen las reglas del lenguaje y su estructura, y no pueden ser usadas como nombres de variables.

Python tiene treinta y tantas palabras clave (y todas son mejoradas de vez en cuando para que Python introduzca o elimine una o dos). Y como Python tiene módulos para

todo, podemos hacer lo siguiente para obtener todas las palabras clave (*keywords*) que tiene:

```
import keyword  
print(keyword.kwlist)
```

Que dará como resultado:

```
['False', 'None', 'True', 'and', 'as', 'assert',  
'break', 'class', 'continue', 'def', 'del', 'elif',  
'else', 'except', 'finally', 'for', 'from', 'global',  
'if', 'import', 'in', 'is', 'lambda', 'nonlocal',  
'not', 'or', 'pass', 'raise', 'return', 'try',  
'while', 'with', 'yield']
```

Asimismo, podemos obtener las funciones integradas (*builtins*) usando:

```
import builtins  
print(dir(builtins))
```

Que dará como resultado:

```
['ArithmetError', 'AssertionError', 'AttributeError',  
'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError',  
'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError',  
'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError',  
'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError',  
'Exception', 'False', 'FileExistsError', 'FileNotFoundException',  
'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError',  
'ImportError', 'ImportWarning', 'IndentationError', 'IndexError',  
'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',  
'LookupError', 'MemoryError', 'NameError', 'None', 'NotADirectoryError',  
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',  
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',  
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',  
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',  
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',  
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',  
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',  
'ValueError', 'Warning', 'ZeroDivisionError', '__build_class__',  
'__debug__', '__doc__', '__import__', '__loader__', '__name__',  
'__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin',  
'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',  
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict',  
'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter',  
'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
```

```
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance',
'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map',
'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord',
'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed',
'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod',
'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

1.9. Sentencias

Una sentencia es una instrucción que el intérprete de Python puede ejecutar. Hemos visto algunas clases de sentencias hasta el momento. Otros tipos de sentencias que se verán en el capítulo 4 son las de `while`, `for`, `if`, y la de `import`, e incluso otros tipos más también.

Cuando escribimos una sentencia en la línea de comandos, Python la ejecuta. Las sentencias de asignación no producen un resultado.

1.10. Evaluación de expresiones

Una expresión es una combinación de valores, variables, operadores, y llamadas a funciones. Si escribimos una expresión en la línea de comandos de Python, el intérprete la evalúa y despliega el resultado:

```
>>> 1 + 1
2
>>> len("hola")
5
```

En este ejemplo `len` es una función integrada de Python que devuelve el número de caracteres de una cadena. La evaluación de una expresión produce un valor; es por ello que las expresiones pueden aparecer en el lado derecho de las sentencias de asignación. Un valor por si mismo se considera como una expresión. De igual modo ocurre para las variables.

```
>>> 17
17
>>> y = 3.14
>>> x = len("hello")
>>> x
5
>>> y
3.14
```

1.11. Operadores y operandos

Los operadores son símbolos especiales que representan cálculos como la suma y la multiplicación. Los valores que el operador usa se denominan operandos. Las siguientes son expresiones válidas en Python:

```
20 + 32    hora - 1    hora * 60 + minuto    minuto / 60    5 ** 2
(5 + 9) * (15 - 7)
```

Los símbolos +, -, *, y los paréntesis para agrupar, significan en Python lo mismo que en matemáticas. El asterisco (*) es el símbolo para la multiplicación, y ** es el símbolo para la potenciación.

```
>>> 2 ** 3
8
>>> 3 ** 2
9
```

Cuando el nombre de una variable aparece en la posición de un operando se reemplaza por su valor antes de realizar la operación. La suma, resta, multiplicación y potenciación realizan lo que esperaríamos. En este ejemplo vamos a convertir 645 minutos en horas:

```
>>> minutos = 645
>>> horas = minutos / 60
>>> horas
10.75
```

Capítulo 2

Robots y su hardware

El propósito de este capítulo es abordar conceptos básicos de electrónica, entender las comunicaciones del puerto serie, introducir la plataforma robótica Arduino, ver el entorno Arduino IDE, y aprender nociones básicas sobre el uso de sensores, actuadores y servos.

2.1. Procesador Arduino

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware libre, flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquier interesado en crear entornos u objetos interactivos.

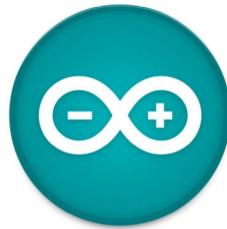


Figura 2.1: Logo de Arduino.

Arduino puede tomar información del entorno a través de sus pines de entrada, para esto se puede usar toda una gama de sensores y puede interactuar con aquello que le rodea mediante luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectarlo a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (p.ej. Flash, Processing, MaxMSP).

Las placas pueden ser hechas a mano o comprarse montadas de fábrica; el software (o Arduino IDE, ver figura 2.2) puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, así pues eres libre de adaptarlos a tus necesidades.

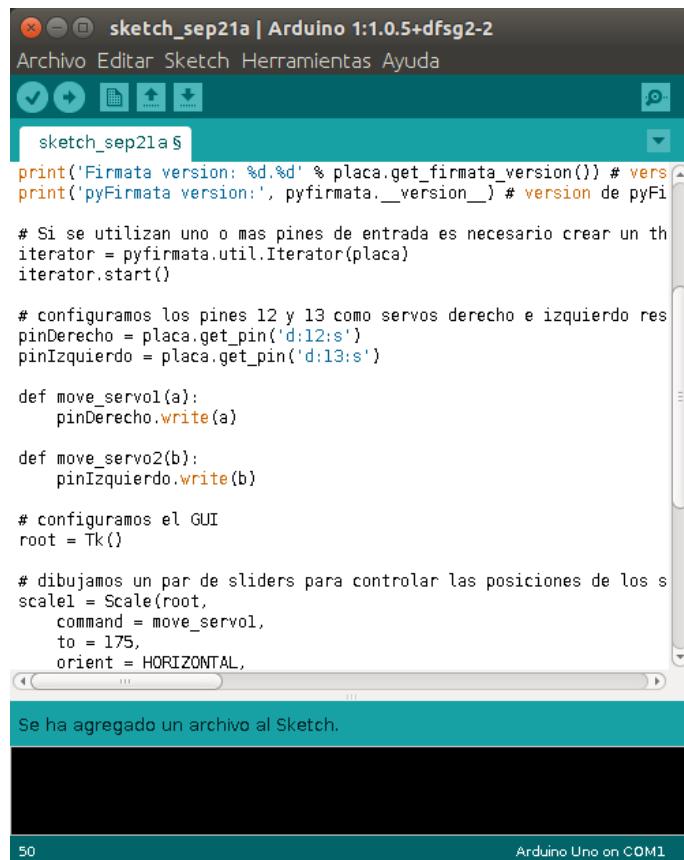


Figura 2.2: Entorno de programación de Arduino IDE.

2.2. Nociónes de electrónica

Desde el inicio de esta sección vamos a introducir los términos técnicos que tiene la electrónica, para así poder manejar con soltura la placa Arduino y sus diversos componentes electrónicos.

2.2.1. Conceptos

La Electrónica es la rama de la Física y especialización de la Ingeniería, que estudia y emplea sistemas cuyo funcionamiento se basa en la conducción y el control del flujo microscópico de los electrones u otras partículas cargadas eléctricamente.

Utiliza una gran variedad de conocimientos, materiales y dispositivos. El diseño y la gran construcción de circuitos electrónicos para resolver problemas prácticos forman parte de la electrónica y de los campos de la ingeniería electrónica, electromecánica y la informática en el diseño de software para su control. El estudio de nuevos dispositivos semiconductores y su tecnología se suele considerar una rama de la física, más concretamente en la rama de ingeniería de materiales.

La electrónica desarrolla en la actualidad una gran variedad de tareas. Los principales usos

de los circuitos electrónicos son el control, el procesado, la distribución de información, la conversión y la distribución de la energía eléctrica. Estos dos usos implican la creación o la detección de campos electromagnéticos y corrientes eléctricas.

2.2.2. Voltaje

Una magnitud física que impulsa a los electrones a lo largo de un conductor en un circuito eléctrico cerrado, provocando el flujo de una corriente eléctrica. Su unidad es el Voltio(V). El instrumento usado para medir el voltaje se conoce como voltímetro.

Voltaje DC

Es el flujo continuo de electrones a través de un conductor entre dos puntos de distinto potencial. En la corriente continua las cargas eléctricas circulan siempre en la misma dirección, es continua toda corriente que mantenga siempre la misma polaridad. En la norma sistemática europea el color negro corresponde al negativo y el rojo al positivo o sencillamente se simboliza para el positivo con VCC, +, VSS y para el negativo con 0V, -, GND.



Figura 2.3: Una pila aplica corriente continua.

Muchos aparatos necesitan corriente continua para funcionar, sobre todos los que llevan electrónica (equipos audiovisuales, computadores, etc.), para ello se utilizan fuentes de alimentación que rectifican y convierten la tensión a una adecuada. Lo puedes encontrar en la baterías, pilas, salida de los cargadores de computador.

Voltaje AC

Es la corriente eléctrica en la que la magnitud y dirección varían cíclicamente. La forma de onda de la corriente alterna más comúnmente utilizada es la de una onda sinusoidal.

Utilizada genéricamente, la AC se refiere a la forma en la cual la electricidad llega a los hogares y a las empresas, es muy común encontrarla en las tomas de corriente donde se conectan nuestros electrodomésticos. Sin embargo, las señales de audio y de radio transmitidas por los cables eléctricos son también ejemplos de corriente alterna. En estos

usos, el fin más importante suele ser la transmisión y recuperación de la información codificada (o modulada) sobre la señal de la AC.

2.2.3. Corriente

Es el flujo de electrones libres a través de un conductor o semiconductor en un sentido. La unidad de medida de ésta es el amperio (A). Una corriente eléctrica, puesto que se trata de un movimiento de cargas, produce un campo magnético, un fenómeno que puede aprovecharse en el electroimán, este es el principio de funcionamiento de un motor.



Figura 2.4: La corriente que pasa por una bombilla.

El instrumento usado para medir la intensidad de la corriente eléctrica es el galvanómetro que, calibrado en amperios, se llama amperímetro, colocado en serie con el conductor cuya intensidad se desea medir.

2.2.4. Resistencia

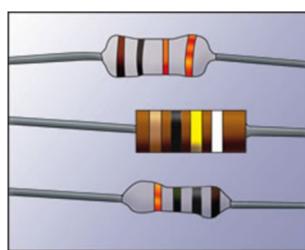


Figura 2.5: Ejemplos de resistencias.

Es la propiedad física mediante la cual todos los materiales tienden a oponerse al flujo de la corriente. La unidad de este parámetro es el Ohmio. Puedes encontrar resistencias en los calefactores eléctricos, tarjetas electrónicas, son muy útiles para limitar el paso de la corriente y el voltaje.

2.2.5. Ley de Ohm

La ley dice que la corriente (I) que circula por un conductor eléctrico es directamente proporcional al voltaje (V) e inversamente proporcional a la resistencia (R).

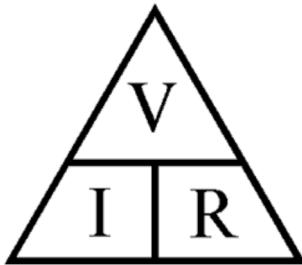


Figura 2.6: Pirámide de Ley de Ohm.

La pirámide que vemos es muy útil para conocer la fórmula a la que es igual la variable que tapes con el dedo, por ejemplo: Tapa con tu dedo la V (voltaje), entonces voltaje va a ser igual a I (corriente) por R (resistencia), una más, tapa I (Corriente), I va ser igual a V dividido R .

2.3. Sistemas electrónicos

Un sistema electrónico es un conjunto de circuitos que interactúan entre sí para obtener un resultado. Una forma de entender los sistemas electrónicos consiste en dividirlos en entradas, salidas y procesamiento de señal.



Figura 2.7: Esquema de un sistema electrónico.

2.3.1. Entradas

Las entradas o Inputs: Son sensores (o transductores) electrónicos o mecánicos que toman las señales (en forma de temperatura, presión, humedad, contacto, luz, movimiento, pH etc.) del mundo físico y las convierten en señales de corriente o voltaje.

Por ejemplo un sensor de temperatura, un pulsador, una fotocelda, un potenciómetro, un sensor de movimiento entre muchos más.

2.3.2. Salidas

Las salidas o Outputs: Son actuadores u otros dispositivos (también transductores) que convierten las señales de corriente o voltaje en señales físicamente útiles como movimiento, luz, sonido, fuerza, rotación entre otros.

Por ejemplo: un display que registre la temperatura, un LED o sistema de luces que se encienda automáticamente cuando esté oscureciendo, un motor, un buzzer que genere diversos tonos.

2.3.3. Procesamiento de señal

Se realiza mediante circuitos de procesamiento de señales generalmente conocidos como microcontroladores. Consisten en piezas electrónicas conectadas juntas para manipular, interpretar y transformar las señales de voltaje y corriente provenientes de los sensores (Entradas) y tomar las respectiva decisiones para generar acciones en las salidas.

2.3.4. Señales electrónicas

Son la representación de un fenómeno físico. Las entradas y salidas de un sistema electrónico serán consideradas como las señales variables. En electrónica se trabaja con variables que se toman en forma de voltaje o corriente, éstas se pueden denominar comúnmente señales.

Las señales primordialmente pueden ser de dos tipos descritos a continuación.

2.3.5. Variable digital

También llamadas variables discretas. Se caracterizan por tener dos estados diferenciados y por lo tanto se pueden llamar binarias. Siendo estas variables más fáciles de tratar (en lógica serían los valores Verdadero (V) y Falso (F) o podrían ser 1 ó 0 respectivamente).

Un ejemplo de una señal digital es el interruptor del timbre de tu casa, por que este interruptor tiene dos estados pulsado y sin pulsar.

2.3.6. Variable analógica

Son aquellas que pueden tomar un número infinito de valores comprendidos entre dos límites. La mayoría de los fenómenos de la vida real son señales de este tipo (sonido, temperatura, voz, video, etc.).

Un ejemplo de sistema electrónico analógico es el altavoz, que se emplea para amplificar el sonido de forma que éste sea oído por una gran audiencia. Las ondas de sonido que son

analógicas en su origen, son capturadas por un micrófono y convertidas en una pequeña variación analógica de tensión denominada señal de audio.

2.3.7. Comunicación serial

Es una interfaz de comunicaciones de datos digitales, frecuentemente utilizado por computadores y periféricos, donde la información es transmitida bit a bit enviando un solo bit a la vez. Uno de sus usos es monitorear a través de la pantalla del computador el estado del periférico conectado, por ejemplo al pulsar la letra A en el teclado se debe accionar un LED conectado de manera remota la computador.

2.4. Componentes electrónicos del kit robótico Arduino

Diversos componentes electrónicos unen sus fuerzas para lograr aplicaciones fantásticas como por ejemplo el televisor de tu casa o el computador, por dentro de ellos vas a encontrar tarjetas con resistencias, condensadores, circuitos integrados, transistores entre otros.

En esta sección vamos a conocer los distintos elementos que conforman el kit de robótica, con placa de Arduino, que vamos a utilizar en este curso.

2.4.1. Microcontrolador

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres unidades funcionales principales: unidad central de procesamiento (CPU), memoria y periféricos de entrada y salida. En la figura 2.8 vemos el microcontrolador de Arduino UNO, que es la que usamos nosotros.

Para que pueda controlar algún proceso es necesario generar o crear y luego grabar en la memoria EEPROM del microcontrolador algún programa, el cual puede ser escrito en lenguaje ensamblador u otro lenguaje para microcontroladores; sin embargo, para que el programa pueda ser grabado en la EEPROM del microcontrolador, debe ser codificado en sistema numérico hexadecimal que es finalmente el sistema que hace trabajar al microcontrolador cuando éste es alimentado con el voltaje adecuado y asociado a dispositivos analógicos y discretos para su funcionamiento.

Los microcontroladores representan la inmensa mayoría de los chips vendidos, sobre un 50 % son controladores simples y el restante corresponde a DSPs más especializados. Mientras se pueden tener uno o dos microprocesadores de propósito general en casa



Figura 2.8: Placa microcontroladora Arduino UNO.

(estás usando uno para esto), tienes distribuidos seguramente entre los electrodomésticos de tu hogar una o dos docenas de microcontroladores. Pueden encontrarse en casi cualquier dispositivo electrónico como automóviles, lavadoras, hornos microondas, teléfonos, Arduino, etc.

Los microcontroladores utilizan la mayoría de su chip para incluir funcionalidad, como los dispositivos de entrada/salida o la memoria que incluye el microcontrolador, con la gran ventaja de que se puede prescindir de cualquier otra circuitería externa.

Los puertos de E/S (entrada/salida) en el microcontrolador, generalmente se agrupan en puertos de 8 bits de longitud, lo cual permite leer datos del exterior o escribir en ellos desde el interior del microcontrolador, el destino habitual es el trabajo con dispositivos simples como relés, LED, motores, photoceldas, pulsadores o cualquier otra cosa que se le ocurra al programador.

2.4.2. Protoboard

Es una placa reutilizable usada para construir prototipos de circuitos electrónicos sin soldadura. Compuestas por bloques de plástico perforados y numerosas láminas delgadas de una aleación de cobre, estaño y fósforo.

2.4.3. Resistencia

Es un material formado por carbón y otros elementos resistivos para disminuir la corriente que pasa. Se opone al paso de la corriente. La corriente máxima en un resistor viene condicionado por la máxima potencia que puede disipar su cuerpo. Esta potencia se

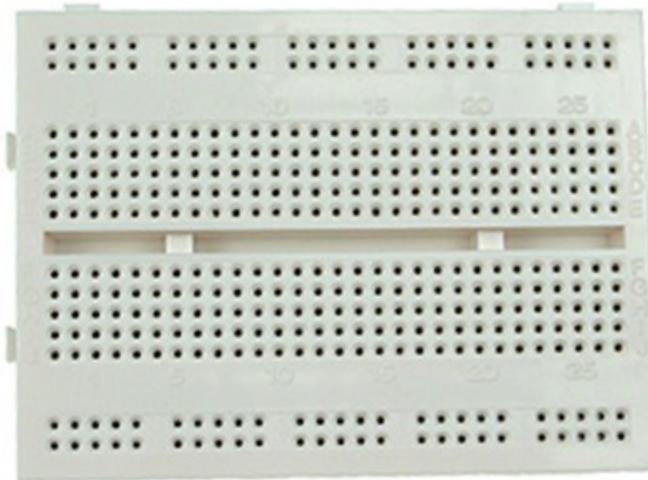


Figura 2.9: Protoboard usada para nuestros circuitos electrónicos.

puede identificar visualmente a partir del diámetro sin que sea necesaria otra indicación. Los valores más comunes son 0,25 W, 0,5 W y 1 W.

El valor de la resistencia eléctrica se obtiene leyendo las cifras como un número de una, dos o tres cifras; se multiplica por el multiplicador y se obtiene el resultado en Ohmios. En la siguiente figura 2.10 podemos ver el valor de una resistencia según sus colores.

2.4.4. Diodo

Un diodo es un componente electrónico de dos terminales que permite la circulación de la corriente eléctrica a través de él en un solo sentido. Tiene dos partes: el cátodo y el ánodo.

2.4.5. Transistor

El transistor es un dispositivo electrónico semiconductor que cumple funciones de amplificador, oscilador, conmutador o rectificador. El término *transistor* es la contracción en inglés de transfer resistor (resistencia de transferencia). Tiene tres partes: la base (B), el emisor (E) y colector (C).

Actualmente se encuentran prácticamente en todos los aparatos domésticos de uso diario: radios, televisores, grabadoras, reproductores de audio y vídeo, microondas, lavadoras, automóviles, relojes de cuarzo, ordenadores, calculadoras, impresoras, tomógrafos, ecógrafos, teléfonos móviles, etc.

| Color de la banda | Valor de la 1ºcifra significativa | Valor de la 2ºcifra significativa | Multiplicador | Tolerancia |
|-------------------|-----------------------------------|-----------------------------------|---------------|--------------|
| Negro | - | 0 | 1 | - |
| Marrón | 1 | 1 | 10 | $\pm 1\%$ |
| Rojo | 2 | 2 | 100 | $\pm 2\%$ |
| Naranja | 3 | 3 | 1 000 | - |
| Amarillo | 4 | 4 | 10 000 | $\pm 4\%$ |
| Verde | 5 | 5 | 100 000 | $\pm 0,5\%$ |
| Azul | 6 | 6 | 1 000 000 | $\pm 0,25\%$ |
| Violeta | 7 | 7 | 10000000 | $\pm 0,1\%$ |
| Gris | 8 | 8 | 100000000 | $\pm 0,05\%$ |
| Blanco | 9 | 9 | 1000000000 | - |
| Dorado | - | - | 0,1 | $\pm 5\%$ |
| Plateado | - | - | 0,01 | $\pm 10\%$ |
| Ninguno | - | - | - | $\pm 20\%$ |

Figura 2.10: Tabla de código de colores de las resistencias.

2.4.6. Condensador

Un condensador o capacitor es un dispositivo pasivo, utilizado en electricidad y electrónica, capaz de almacenar energía sustentando un campo eléctrico. Está formado por un par de superficies conductoras, generalmente en forma de láminas o placas, en situación de influencia total separadas por un material dieléctrico o por el vacío. Las placas, sometidas a una diferencia de potencial, adquieren una determinada carga eléctrica, positiva en una de ellas y negativa en la otra.

2.4.7. LED

Un LED (Diodo emisor de luz) es un diodo semiconductor que emite luz. Se usan como indicadores en muchos dispositivos, y cada vez con mucha más frecuencia en iluminación. Los LEDs presentan muchas ventajas sobre las fuentes de luz incandescente como un consumo de energía mucho menor, mayor tiempo de vida, menor tamaño, gran durabilidad y fiabilidad.

El LED tiene una polaridad, un orden de conexión, y al conectarlo al revés se puede quemar. Hay que tener esto en cuenta y revisar los circuitos para conocer en qué lado corresponde el positivo y el negativo.



Figura 2.11: Diodo LED.

2.4.8. Pulsador



Figura 2.12: Pulsador.

Un botón o pulsador es un dispositivo utilizado para activar alguna función. Los botones son por lo general activados al ser pulsados, normalmente con un dedo. Un botón de un dispositivo electrónico funciona por lo general como un interruptor eléctrico, es decir en su interior tiene dos contactos, si es un dispositivo NA (normalmente abierto) o NC (normalmente cerrado), con lo que al pulsarlo se activará la función inversa de la que en ese momento este realizando.

2.4.9. Potenciómetro

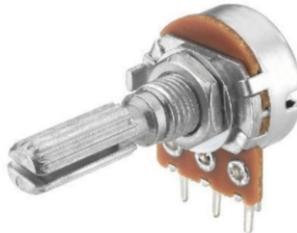


Figura 2.13: Potenciómetro.

Un potenciómetro es una resistencia cuyo valor de resistencia es variable. De esta manera, indirectamente, se puede controlar la intensidad de corriente que fluye por un circuito si se conecta en paralelo, o controlar el voltaje al conectarlo en serie. Son adecuados para su uso como elemento de control en los aparatos electrónicos. El usuario acciona sobre ellos para variar los parámetros normales de funcionamiento. Por ejemplo, el volumen de un radio.

2.4.10. Zumbador

El zumbador, buzzer en inglés, es un transductor electroacústico que produce un sonido o zumbido continuo o intermitente de un mismo tono. Sirve como mecanismo de señalización o aviso, y son utilizados en múltiples sistemas como en automóviles o en electrodomésticos.



Figura 2.14: Zumbador.

Su construcción consta de dos elementos, un electroimán y una lámina metálica de acero. El zumbador puede ser conectado a circuitos integrados especiales para así lograr distintos tonos. Cuando se acciona, la corriente pasa por la bobina del electroimán y produce un campo magnético variable que hace vibrar la lámina de acero sobre la armadura.

2.4.11. Motor DC



Figura 2.15: Motor DC.

El motor de corriente continua (DC) es una máquina que convierte la energía eléctrica en mecánica, provocando un movimiento rotatorio. Esta máquina de corriente continua es una de las más versátiles. Su fácil control de posición, paro y velocidad la han convertido en una de las mejores opciones en aplicaciones de control y automatización de procesos.

2.5. Mbot

2.6. PI-Mbot

Con el fin de tener un robot más potente y autónomo, al que poder añadirle más actuadores y sensores que el software del mbot "normal" no soporta, añadimos una placa

Raspberry-3 (elegimos la 3 pues tiene tarjeta wifi incorporada)



Figura 2.16: Placa base Raspberry3

La Raspberry es una placa base de ordenador, básica y asequible, que se puede adquirir directamente en su web y que es más que suficientemente potente para el robot Mbot (además de no pesar mucho, lo cual es una ventaja). Su software es Linux y, como veremos más adelante, muy fácil de instalar.

Como cualquier placa base, la Raspberry necesita una fuente de alimentación, en este caso elegimos una batería externa de las utilizadas para cargar dispositivos móviles o tablets. Como no tenemos monitor, ni sonido, ni nada de lo que realmente gasta batería en un ordenador, no necesitamos mucha capacidad (no necesitamos una batería muy grande). En cuanto conectemos la placa con la batería, ésta se enciende, sin que necesitemos un monitor o un ratón para ello. Para apagarla, basta con desconectarla de la batería.

Como se ve en la siguiente figura, conectamos y añadimos la placa al robot, y conectamos ésta a la batería.



Figura 2.17: Pi-mBot

2.7. PiBot

Yendo un paso más allá, podemos prescindir completamente de Arduino y mBot, obteniendo una plataforma robótica directa y exclusivamente conectada a la Raspberry Pi 3. Para ello contamos con un chasis, dos ruedas motrices, una rueda loca, la batería que da energía a la Raspberry, y los sensores que necesitamos para conseguir una navegación autónoma con algoritmos de visión: sensor de infrarrojos y PiCam.

Obtenemos así el siguiente prototipo:

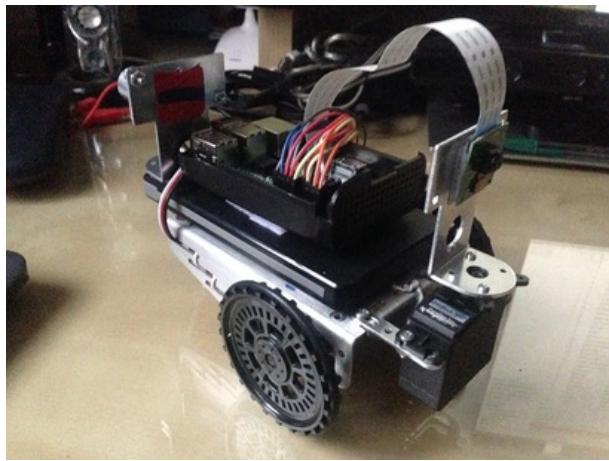


Figura 2.18: PiBot

Capítulo 3

Introducción a la programación de robots con Scratch

En este capítulo vamos a empezar en el mundo de la programación mediante el lenguaje Scratch. Veremos cómo la programación es un gran recurso que nos permite crear diversas secuencias de pasos lógicos que van a satisfacer nuestras necesidades y las de nuestros sistemas. Scratch es un buen punto de partida debido a su carácter visual y a que son "bloques" de instrucciones en vez de programación clásica, facilitando mucho el aprender conceptos, antes de necesitar aprender sintaxis, asunto muy farragoso para niños de primaria o secundaria.

3.1. Lenguaje

Scratch es un lenguaje de programación y una comunidad virtual, diseñado por el MIT, donde niños, jóvenes y educadores pueden crear y compartir sus creaciones. Se pueden elaborar desde animaciones básicas hasta juegos, siendo la base de su programación los bloques de instrucciones. Para empezar a programar en Scratch deberás visitar [la página web](#).



Possiblemente los juegos sean la creación más atractiva y podemos encontrar creaciones artesanales de juegos clásicos como Pacman o Space Invaders hasta juegos más modernos como Candy Crash o Geometry Dash. Pero también podemos crear pequeñas escenas en las que varios personajes hablar, se mueven e interaccionan.

Para encontrar cualquier creación solo tienes ir al recuadro *Buscar* y si pulsas en *Ver Dentro* podrás observar la programación en bloques usada.

Para empezar a usar Scratch puedes pulsar en *Crear*; para mirar tutoriales y guías, en *Ayuda*. Para guardar tus creaciones o comunicarte con otros usuarios debes crearte un usuario y loguearte pulsando *Únete a Scratch* o *Ingresar*, respectivamente.

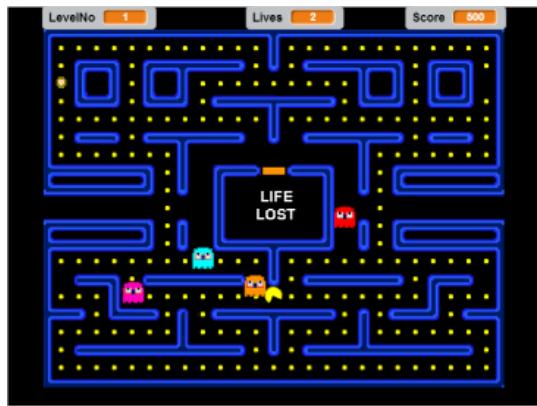


Figura 3.1: Juego del Comecocos en Scratch

Para conocer más características, funciones o ayuda sobre Scratch se puede visitar la sección de Ayuda. Además, existen numerosas guías, ejemplos y videos de cómo hacer multitud de algoritmos en la web <https://scratch.mit.edu/help>. Asimismo, para alumnado menor de edad se puede encontrar más información sobre Scratch en <https://scratch.mit.edu/parents>

3.2. Instalación del entorno

Para empezar a utilizar el entorno de Scratch no necesitamos nada salvo una conexión a Internet. Si vamos a querer trabajar más cómodamente, siempre podemos descargar la aplicación de escritorio.

Es cuando querremos programar los robots de aprendizaje Mbot usando Scratch cuando no podremos usar simplemente la versión web de Scratch y necesitaremos el software del fabricante MakeBlock.

Podemos elegir si usar la versión online del software (aunque la conexión con el robot a veces falla) o descargar la versión aplicación de escritorio (sólo disponible para Windows o Mac), cuya página se muestra en la figura 3.2. Una vez descargado el programa, la instalación es igual a cualquier software de Windows. A diferencia de Scratch "normal", este software tiene una biblioteca más: **Robot** (Figura 3.3), que contiene las instrucciones propias del robot que hayamos elegido previamente; en este caso MBot.

Como podemos ver, Makeblock nos proporciona acceso a todos los sensores y actuadores del MBot.

3.3. Ejercicios de lenguaje

Para comprender los conceptos de la programación, tales como objetos, entornos, etc, es conveniente usar Scratch como lenguaje de programación antes de programar el robot directamente. Cuando los niños ya hayan visto la generalidad del lenguaje, seguiremos

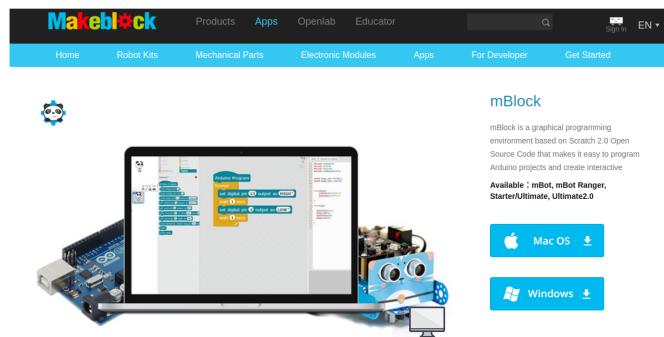


Figura 3.2: Descarga de la aplicación de escritorio de Makeblock

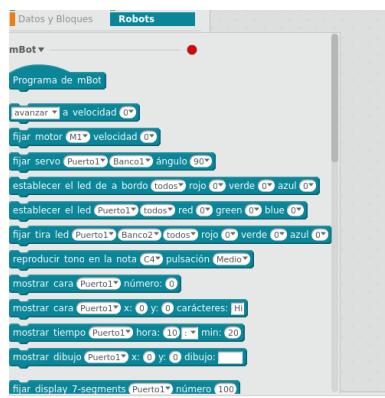


Figura 3.3: Biblioteca de Scratch para el MBot

utilizando Scratch a la vez que la programación de robots, con el fin de continuar el aprendizaje de conceptos y aplicar esos conocimientos a la programación de robots, añadiendo además el hecho de que a los niños les entretienen mucho más los robots. Veamos primero qué necesitamos para programar en Scratch:

3.3.1. Crear, modificar o añadir objetos

Siempre nos sale por defecto ese gato naranja, pero vamos a darle colorido o cambiarle de aspecto. Teniendo seleccionado el objeto del gato hemos pulsado la pestaña de *Disfraces*. Ha cambiado la ventana de la derecha y ahora tiene el aspecto de un *Paint*, con pinceles, cubetas y colores. Hay objetos que tienen varios disfraces para poder hacer una animación cuando se mueva.

Hay dos tipos de imágenes que podemos incluir como objetos: los mapas de bits y las imágenes vectoriales. Hay diferencias entre unas y otras pero podemos transformar una imagen de un tipo a otro pulsando en el botón *Modo vector*, *Convertir a mapa de bits*.

Si pulsamos en la ventana de *Disfraces* aparecen muchas herramientas para cambiar el aspecto del muñeco. Podemos cambiar colores, deformar líneas, escribir texto, hacer



Figura 3.4: Entorno de Disfraces de Scratch

formas geométricas, etc. También podemos incluir más personajes en el juego; para ello, tenemos que pinchar en la ventana de *Objetos* en el botón de *Nuevo Objeto* desde la *Biblioteca*, según vemos en la siguiente figura. Si pulsamos, saldrá una pantalla nueva llena de objetos y a la izquierda hay una biblioteca que me permite elegirlo según el tema que me apetezca.

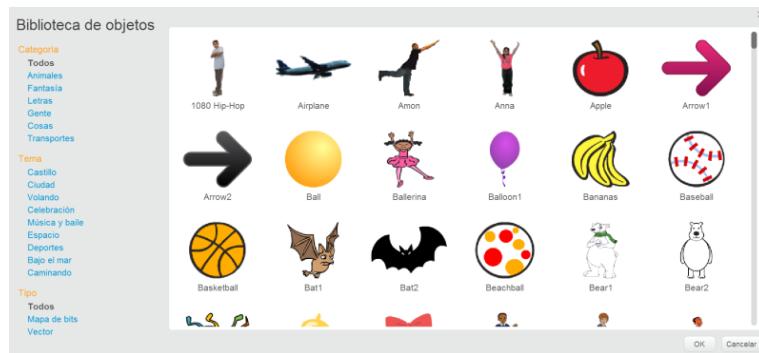


Figura 3.5: Biblioteca de objetos de Scratch

Si lo que quiero es incluir un objeto que tenga en mi ordenador, en cualquier carpeta, lo que tengo que hacer es pulsar en *Cargar objeto desde archivo*. Por contra, si la imagen que quiero incluir en el programa no la tengo en el ordenador, puedo buscarla en Internet y descargarla en mi ordenador.

Todos los objetos pueden hacerse más grandes o más pequeños, o incluso borrarlos o

duplicarlos si pulso en los botones que aparecen en la parte superior, y que se muestran a continuación.



Figura 3.6: Botones para modificar objetos de Scratch

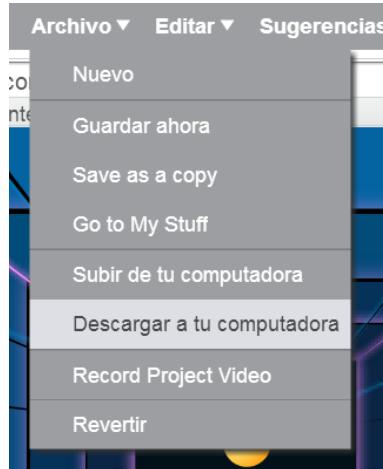


Figura 3.7: Cómo guardar el sketch creado

También tienes la opción de descargar el juego en tu ordenador como un archivo una vez hayas terminado tu creación. Para ello pulsamos en *Archivo* del menú principal y después en *Descargar a tu computadora* (ver la anterior figura). Nos saldrá una ventana que nos indica donde queremos guardarlo y decidiremos dónde (el escritorio suele ser un buen sitio).

3.3.2. Crear, modificar o añadir fondos

Los fondos son elementos visuales que se colocan debajo de todos los objetos y sitúan a los objetos en un entorno. Para añadir un fondo desde la biblioteca, dibujar uno nuevo o subir uno desde un archivo pulsaré en los botones de la esquina inferior izquierda, y que tienen el siguientes aspecto.



Figura 3.8: Botones para modificar fondos de Scratch

Si elijo un fondo y quiero modificarlo, solo tengo que seleccionarlo y pulsar en *Fondos*.

3.3.3. Tipos de bloques de instrucciones

Existen instrucciones de muy diferentes tipos y están diferenciadas con colores diferentes.



Figura 3.9: Tipos de bloques de programación en Scratch

Si vemos detenidamente cada tipo veremos que tienen diferentes formas. Unas instrucciones se deben colocar al principio del bloque, como las de Evento. Otras se acoplan unas a continuación de otras como las de Movimiento, Apariencia o Sonido. Otras tienen un espacio interior para meter dentro de ellas otras instrucciones como las de Control. Y otras como los Sensores o Operadores sirven para modificar otras instrucciones.

3.3.4. Guardar el archivo de un juego

La mejor forma de guardar una creación de Scratch es crearte un usuario, puesto que el guardado es automático. De este modo podrás consultar cualquiera de ellas en cualquier momento. Además podrás compartir tus creaciones para que las vean otros usuarios y te las valoren.

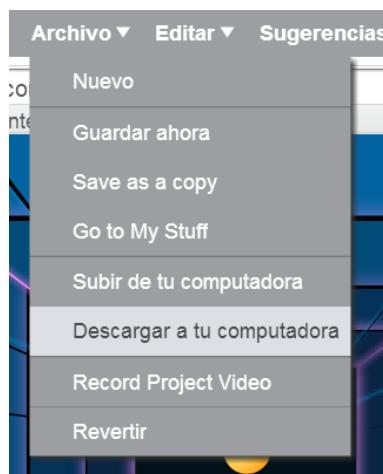


Figura 3.10: Cómo guardar el sketch creado

También tienes la opción de descargar el juego en tu ordenador como un archivo una vez hayas terminado tu creación. Para ello pulsamos en *Archivo* del menú principal y después en *Descargar a tu computadora* (ver la anterior figura). Nos saldrá una ventana que nos indica donde queremos guardarlo y decidiremos dónde (el escritorio suele ser un buen sitio).

3.3.5. Ejemplo práctico

Scratch es un entorno visual muy intuitivo y sencillo de manejar. Con un poco de práctica se pueden hacer creaciones muy atractivas. A continuación, y a modo de recapitulación de todo lo anterior, vamos a ver un ejemplo de cómo realizar una pequeña animación de dos personajes. Para ello, vamos a usar dos personajes y un fondo.

Estos personajes se moverán hasta estar cara a cara y después tendrán una pequeña conversación simulada mediante bocadillos de texto. En nuestro caso hemos elegido al gato y al hipopótamo. Y el fondo es el de la pista de baloncesto. Hemos colocado a ambos en cada esquina de abajo como se puede ver en la siguiente figura.



Figura 3.11: Aspecto del ejemplo práctico, con dos personajes

La dinámica de nuestro juego es muy sencilla: al pulsarse la Bandera de Inicio, el gato y el hipopótamo deben deslizarse durante 1 segundo desde su posición inicial hasta la canasta central, y allí, empezarán la conversación típica de un gato y un hipopótamo con alas... La orden a cumplir es, por tanto, una Instrucción de Movimiento *Deslizar*.



Figura 3.12: Personaje de gato

Pero para hacer bien esta instrucción es necesario indicar dónde deben deslizarse el gato y el hipopótamo. Esa posición se indica con un valor X y otro Y. Colocaremos el gato y el hipopótamo donde queramos que lleguen, y para saber esa situación debemos pulsar en cada personaje en la *i* de información, y nos saldrán los valores de X e Y de la posición a la que debe llegar.

En este caso el gato debe llegar a X=-54 e Y=-11. Por lo tanto, la instrucción *Deslizar* para el gato debe ser la que se ve en la siguiente figura.



Figura 3.13: Información de posición del personaje



Figura 3.14: Bloques para desplazar al gato

Para el hipopótamo debemos hacer lo mismo. Además, podemos cambiar el tiempo en segundos para hacer que el deslizamiento sea más rápido o más lento. La instrucción para el hipopótamo debe ser como se muestra en la siguiente figura.



Figura 3.15: Bloques para desplazar al hipopótamo

Si colocamos cada personaje en su esquina inicial y pulsamos en la Bandera de Inicio, veremos cómo se desplazan los dos en 1 segundo al punto que hemos decidido.

Ahora, debemos hacer que los personajes tengan una pequeña conversación. Las instrucciones a usar son las de Apariencia: *Decir*. Si ponemos las instrucciones como vemos a continuación y pulsamos la Bandera veremos que los dos personajes hablan a la vez.

Para que sea una conversación más realista, uno de los personajes debe empezar hablando mientras el otro espera los mismos segundos; para ello haremos uso de la Instrucción de Control *Esperar*. De este modo, puedo hacer una conversación más amena. Los bloques de instrucciones quedarían como sigue.

Ya para acabar, vamos a usar una Instrucción de Control llamada *Repetir*, que nos permitirá repetir unas instrucciones de forma repetida el número de veces que queramos.

Nosotros vamos a hacer que el hipopótamo de vueltas. Para ello, tras las instrucciones de la conversación del hipopótamo, ponemos que repita 100 veces el movimiento y giro del mismo, quedando el código como sigue.

La instrucción *Repetir* repite las órdenes que hay dentro de ella el número de veces que se le indica. Así, con lo anterior, si le damos a la Bandera, el hipopótamo se pondrá a dar vueltas pero solo unos instantes. También podemos hacer que el mismo de vueltas



Figura 3.16: Conversación a la vez de los dos personajes



Figura 3.17: Conversación final de los dos personajes

constantemente sin parar; para ello debemos usar la instrucción *Por siempre* según la siguiente imagen.

En este caso, las órdenes de dentro del bloque *Por siempre* se repetirán siempre, hasta que no se pulsa el botón rojo que hay al lado de la bandera verde de inicio. Y así ya tendremos acabado nuestro primer *sketch* de Scratch.

3.3.6. Ejercicios de Scratch

A pesar de la jugabilidad y el entorno gráfico que proporciona Scratch, que lo hacen mucho más visual y “atractivo” para los niños que un lenguaje de programación típico, la experiencia nos dice que pronto se aburren de hacer moverse o cambiar de color al muñeco. Dado que nuestro objetivo es enseñarles los conceptos básicos de programación (objetos, entornos, variables, bucles, condiciones, etc), necesitamos ejercicios con los que enseñar esos conceptos, aunque ellos nos se den cuenta, gradualmente y de forma iterativa y retroalimentada.

Con este fin, diseñamos ciertos ejercicios (podemos añadir el aliciente extra de hacerlo competición) para aprender los conceptos antes mencionados de una forma gradual y entretenida.

Snake

De este juego hay varias versiones, siendo la más conocida la pre-grabada en los teléfonos Nokia. Sin embargo, los alumnos conocen versiones más recientes del juego. Como la implementación es la misma, les divertirá hacer un juego que conocen y poder jugar con él al terminarlo les mueve a hacerlo mejor.

En este caso, elegimos la versión del juego en el que, el “animal” (pues los alumnos elegirán el objeto que más les guste, y lo hacen personalizado) aumentará su tamaño cuando



Figura 3.18: Cómo repetir un trozo de código



Figura 3.19: Cómo repetir un trozo de código por siempre

como ciertos objetos y lo disminuirá con otros. Así, utilizamos distintas funcionalidades del lenguaje, con sus correspondientes conceptos de programación:

- Aumento y disminución del tamaño del objeto: los objetos de programación no tienen unas características inamovibles sino que se pueden cambiar durante la ejecución del programa.
- Bloques de condición anidados; la ejecución condicional es uno de los conceptos más básicos y necesarios en la programación.

Brick Breaker

Aunque en realidad valga cualquier otro juego, este es muy conocido por su popularidad en los años 80. Los alumnos tendrán que diseñar en Scratch el juego desde el principio, y el que más se parezca al juego original, gana la competición. Con este juego, conseguimos interiorizar diferentes necesidades de un programador:

- Objetos: como el juego necesita un paddle, y distintos bricks, entendemos el concepto de que cada objeto tiene su propio código y su propio comportamiento.
- Variables: tenemos una puntuación total, y cada brick nos da más o menos puntuación; esto nos sirve para entender el concepto de entorno de cada variable.
- Bucles: obviamente, el juego se mantiene todo el rato, pero tendremos bucles infinitos (como el del paddle) y otros que acaben si pasamos de nivel, etc.
- Condiciones: la misma necesidad de tener puntuaciones (además, distintas), nos obliga a aprender la condición "si...".
- Escenarios: la implementación de diferentes niveles de dificultad podríamos

entenderla como diferentes funciones, cada una con sus variables, mientras mantenemos una global (puntuación total), y también cambiamos los objetos, aunque el objeto 'paddle' se mantenga.

Además, como el juego se programa en distintas iteraciones, aprenderán a subdividir los problemas grandes (como sería intentar hacer el juego todo de una vez) en problemas más pequeños (por ejemplo, necesitamos un paddle que se mueva hacia los lados). Los ejercicios que vayamos viendo, los podremos enfocar como partes del juego, haciéndolo poco a poco y pudiéndolo combinar con ejercicios meramente de lenguaje Scratch.

3.4. Ejercicios de pseudocódigo

En esta sección vamos a plantear algunos ejercicios básicos en pseudocódigo para comenzar con los principios de la programación. Ejemplificaremos algunos de ellos para facilitar el aprendizaje.

3.4.1. Toma de contacto con el entorno

Hacer un pseudocódigo que escriba 'Hola mundo'.

```
PROGRAMA hola_mundo_1
    ESCRIBIR: "hola mundo!"
FINPROGRAMA
```

Vemos que las líneas aparecen indentadas. Éste es un convencionalismo en Programación, ya que de este modo se aprecia de forma sencilla qué instrucciones pertenecen a qué bloque de instrucciones.

Hacer un pseudocódigo que escriba 'hola mundo' y 'adiós mundo' ahorrando espacio.

```
PROGRAMA hola_mundo_2
    ENTORNO c = "mundo"
    ALGORITMO
        ESCRIBIR: "Hola" y c
        ESCRIBIR: "Adios" y c
    FINPROGRAMA
```

Sumar dos números y escribir el resultado.

```
PROGRAMA sumar
    ENTORNO
```

```
n1 = 3  
n2 = 2  
ALGORITMO  
    nt = n1+n2  
    ESCRIBIR nt  
FINPROGRAMA
```

Hacer un pseudocódigo que multiplique dos números, reste un tercero e imprima el resultado.

Hacer un pseudocódigo que imprima el área de un círculo de radio 4.

3.4.2. Entradas por el teclado

Hacer un pseudocódigo que lea un número del teclado y escribirlo.

```
PROGRAMA leer_imprimir  
    LEER num  
    ESCRIBIR num  
FINPROGRAMA
```

Hacer un pseudocódigo que lea un numero del teclado, lo eleve al cuadrado y lo imprima.

Hacer un pseudocódigo que lea una frase del teclado y la imprima.

3.4.3. Bucles y tomas de decisión

Hacer un pseudocódigo que imprima los números del 1 al 100.

```
PROGRAMA contador1  
ENTORNO:  
    numero = 0  
ALGORITMO:  
    Borrar_pantalla  
    MIENTRAS numero < 101 HACER  
        ESCRIBIR numero  
        numero = numero + 1  
    FINMIENTRAS  
FINPROGRAMA
```

Hacer un pseudocódigo que imprima los números del 100 al 0, en orden decreciente.

Hacer un pseudocódigo que imprima los números pares entre 0 y 100.

Hacer un programa que imprima la suma de los 100 primeros números.

Hacer un pseudocódigo que imprima los números impares hasta el 100 y que imprima cuantos impares hay.

Hacer un pseudocódigo que imprima todos los números naturales que hay desde la unidad hasta un numero que introducimos por teclado.

Hacer un pseudocódigo que no pare hasta que introduzcamos S o N.

```
PROGRAMA sn
ENTORNO:
    res <- " "
ALGORITMO:
    Borrar_pantalla
    MIENTRAS res <> "S" Y res <> "N" HACER
        ESCRIBIR "Introduce S o N"
        LEER res
        res <- Convertir_mayúsculas( res )
    FINMIENTRAS
FINPROGRAMA
```

Hacer un pseudocódigo que dé error si no introducimos S o N

Introducir un número por teclado. Que nos diga si es positivo o negativo.

Imprimir y contar los múltiplos de 3 desde la unidad hasta un número que introducimos por teclado.

Imprimir los números del 1 al 100. Que calcule la suma de todos los números pares por un lado, y por otro, la de todos los impares y que imprima el resultado.

Imprimir el mayor y el menor de una serie de cinco números que vamos introduciendo por teclado.

```
PROGRAMA mayor_menor
ENTORNO:
    con = 0
    n = 0
    maximo = 0
```

```

minimo = 99999
ALGORITMO:
    Borrar_pantalla
    MIENTRAS con <= 5 HACER
        ESCRIBIR "Número: "
        LEER n
        SI n > maximo ENTONCES
            maximo = n
        FINSI
        SI n < minimo ENTONCES
            minimo = n
        FINSI
        con = con + 1
    FINMIENTRAS
    ESCRIBIR "El mayor de los números es: "
    ESCRIBIR maximo
    ESCRIBIR "El menor de los números es: "
    ESCRIBIR minimo
FINPROGRAMA

```

3.4.4. Bucles anidados y subprogramas

Imprimir, contar y sumar los múltiplos de 2 que hay entre una serie de números, tal que el segundo sea mayor o igual que el primero.

Simular el funcionamiento de un reloj digital y que permita ponerlo en hora.

```

PROGRAMA reloj
ENTORNO:
    horas = 0
    minutos = 0
    segundos = 0
    res = "S"
ALGORITMO:
    Borrar_pantalla( )
    ESCRIBIR "Horas: "
    LEER horas
    ESCRIBIR "Minutos: "
    LEER minutos
    ESCRIBIR "Segundos: "
    LEER segundos
    MIENTRAS res = "S" HACER
        MIENTRAS horas < 24 HACER

```

```

MIENTRAS minutos < 60 HACER
    MIENTRAS segundos < 60 HACER
        ESCRIBIR horas
        ESCRIBIR minutos
        ESCRIBIR segundos
        segundos = segundos + 1
    FINMIENTRAS
    minutos = minutos + 1
    segundos = 0
FINMIENTRAS
horas = horas + 1
minutos = 0
FINMIENTRAS
horas = 0
FINMIENTRAS
FINPROGRAMA

```

Introducir un numero menor de 5000 y pasarlo a numero romano.

3.4.5. Switches, números aleatorios y menús

Introducir dos números por teclado y, mediante un menú, calcule su suma, su resta, su multiplicación o su división.

```

PROGRAMA menu1
ENTORNO:
op = 0
ALGORITMO:
EN 10,20 ESCRIBIR "Numero 1: "
EN 10,29 LEER n1
EN 12,20 ESCRIBIR "Numero 2: "
EN 12,29 LEER n2
MIENTRAS op <> 5 HACER
    op = 0
    Borrar_pantalla
    EN 6,20 ESCRIBIR "Menu de opciones"
    EN 10,25 ESCRIBIR "1.- Suma"
    EN 12,25 ESCRIBIR "2.- Resta"
    EN 14,25 ESCRIBIR "3.- Multiplicación"
    EN 16,25 ESCRIBIR "4.- División"
    EN 18,25 ESCRIBIR "5.- Salir del programa"
    EN 22,25 ESCRIBIR "Elija opción: "
    EN 22,39 LEER op

```

```

Borrar_pantalla
HACER CASO
  CASO op == 1
    EN 10,20 ESCRIBIR "Su suma es: "
    EN 10,33 ESCRIBIR n1 + n2
    Pausa
  CASO op == 2
    EN 10,20 ESCRIBIR "Su resta es: "
    EN 10,33 ESCRIBIR n1 - n2
    Pausa
  CASO op == 3
    EN 10,20 ESCRIBIR "Su multiplicación es: "
    EN 10,33 ESCRIBIR n1 * n2
    Pausa
  CASO op == 4
    EN 10,20 ESCRIBIR "Su división es: "
    EN 10,33 ESCRIBIR n1 / n2
    Pausa
FINCASO
FINMIENTRAS
FINPROGRAMA

```

3.4.6. Arrays unidimensionales

Leer las calificaciones de un alumno en 10 asignaturas, las almacene en un vector y calcule e imprima su media.

```

PROGRAMA notamedia
ENTORNO:
  DIMENSIONA notas[ 10 ]
  suma = 0
  media = 0
ALGORITMO:
  Borrar_pantalla
  fi = 7
  PARA i DESDE 1 HASTA 10 HACER
    EN fi,15 ESCRIBIR "Nota "
    EN fi,20 ESCRIBIR i
    EN fi,21 ESCRIBIR ":" "
    EN fi,23 LEER notas[ i ]
    fi = fi + 1
  FINPARA
  PARA i DESDE 1 HASTA 10 HACER

```

```

    suma = suma + notas[ i ]
FINPARA
media = suma / 10
EN 20,20 ESCRIBIR "Nota media: "
EN 20,32 ESCRIBIR media
FINPROGRAMA

```

Usando el ejemplo anterior, hacer que busque una nota en el vector.

3.4.7. Arrays bidimensionales

Generar una matriz de 4 filas y 5 columnas con números aleatorios entre 1 y 100, e imprimirla.

```

PROGRAMA matriz
ENTORNO:
  DIMENSIONAR A[ 4, 5 ]
  i = 1
  fi = 10
  co = 15
ALGORITMO:
  Borrar_pantalla
  EN 6,25 ESCRIBIR "Elementos de la matriz"
  MIENTRAS i <= 4 HACER
    j = 1
    MIENTRAS j <= 5 HACER
      A[ i, j ] = Int( Rnd * 100 ) + 1
      EN fi,co ESCRIBIR A[ i, j ]
      co = co + 5
      j = j + 1
    FINMIENTRAS
    co = 15
    fi = fi + 2
    i = i + 1
  FINMIENTRAS
FINPROGRAMA

```

Cargar en una matriz las notas de los alumnos de un colegio en función del número de cursos (filas) y del número de alumnos por curso (columnas).

PROYECTO FINAL. Una empresa guarda en una tabla de $3 \times 12 \times 4$ las ventas realizadas por sus tres representantes a lo largo de doce meses de sus cuatro productos, $\text{VENTAS}[\text{representante}, \text{mes}, \text{producto}]$. Queremos proyectar el array tridimensional sobre uno de dos dimensiones que represente el total de ventas, $\text{TOTAL}[\text{mes}, \text{producto}]$, para lo cual sumamos las ventas de cada producto de cada mes de todos los representantes. Imprimir ambos arrays.

3.5. Mbot: prácticas básicas

3.5.1. Cómo ejecutar un programa de Scratch en Mbot.

Lo primero que deberemos hacer, para poder probar cada pequeña cosa que hagamos directamente en el robot, es conectarle al ordenador. Podemos conectarle por cable USB, por bluetooth (si tenemos la versión del mBot con él) o con el USB a frecuencia de Wifi (Figura 3.20).

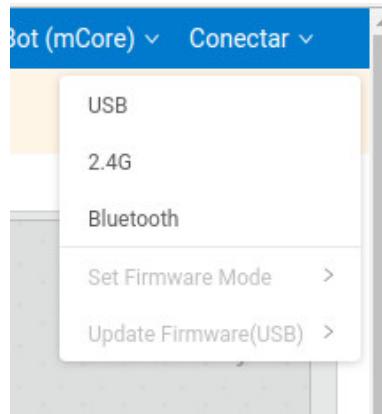


Figura 3.20: Conectar el robot

Una vez conectado el robot, la ejecución del programa será como la de Scratch online normal, y siempre que hayamos usado los bloques de robot, junto con los bloques de control, el robot se moverá sin ningún otro paso por nuestra parte.

3.5.2. Mover el robot

Lo primero y más fácil de hacer es mover el robot en las diferentes direcciones. El código del ejemplo para los primeros movimientos del robot se ven en la Figura 3.21

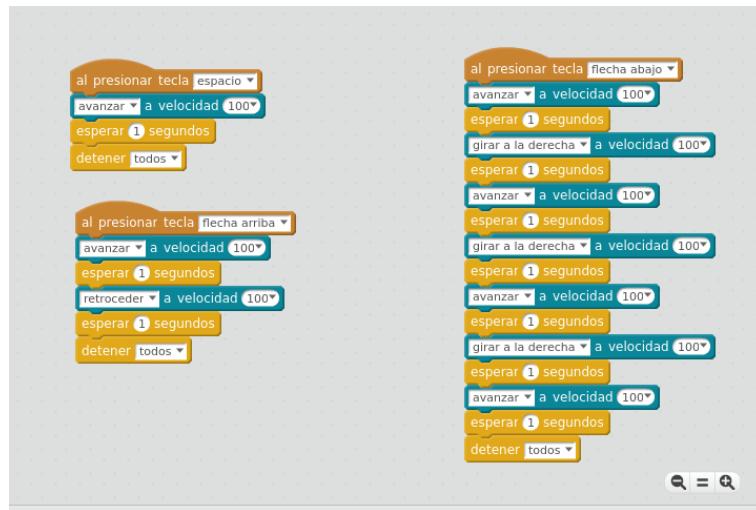


Figura 3.21: Mover el robot

3.5.3. Bucle

Hacer moverse al robot varias veces igual tiene diferentes formas de hacerse, como vemos en este ejemplo de un bucle muy sencillo. Observamos que, sin el bloque de "Parar todos",



Figura 3.22: Ejemplo de bucle

el robot no parará, pues el motor se queda con la última instrucción mandada. Aunque el bucle termine, y la secuencia de instrucciones no se repetirá, el motor es "tonto" se queda con lo último que estaba haciendo.

3.5.4. Ejercicios

Camión

Un camión real avanza normalmente y, sólo cuando da marcha atrás, activa una sirena y las luces. El objetivo es que hagan del Mbot un camión, para que aprendan la

temporalidad y las condiciones: sólo cuando esté retrocediendo debe pitir y encender luces.

Cumpleaños feliz

Con este ejercicio aprendemos lo siguiente:

- Utilizar más actuadores
- Bucles: hay estrofas que se repiten y otras que no
- Orden temporal de las instrucciones

El ejercicio es tan simple como que el robot entone la melodía de Cumpleaños Feliz, jugando con las distintas notas del zumbador del Mbot.

Cumpleaños feliz con botón de Start

Después de haber entonado el cumpleaños feliz, se les plantea la cuestión de controlar el comienzo.

Hasta ahora, según encienden el robot, el programa arranca. Les enseñaremos a controlar el comienzo con el botón pulsador de la placa del Mbot. Usaremos el bloque **Esperar hasta que =** el botón esté pulsado. El resto del código, será el mismo.

S.O.S

Hacemos que el Mbot pida ayuda en morse. Lo haremos en distintas dificultades, primero sólo con luces (los LED del mbot), luego con sonidos y luces, que lo hagan a la vez.

Panel LED

Con el objetivo de usar más actuadores y ver que no todos se programan igual ni tienen los mismos parámetros, cambiamos el ultrasonido por el panel Led y "jugamos" con él: al ser mucho más visual, es mucho más entretenido para los niños. Podemos hacer varias cosas:

- Ojos abriéndose y cerrándose
- Repetir S.O.S pero visual
- Cuenta atrás
- Cualquier dibujo que ellos quieran diseñar

Huye de la luz

Utilizando el sensor de luz de la placa del Mbot, hacemos que el robot huya de la luz.

El escondite inglés

El diseño del típico ejercicio de reaccionar al ruido también lo plantearemos como un juego: el tradicional *Un, dos, tres, al escondite inglés*. Lo cambiamos un poco, en vez de cuando un robot se de la vuelta, los robots tendrán que avanzar cuando el nivel de ruido supere un umbral, que tendrán que definir los mismos alumnos contando con el ruido ambiente (probando los distintos niveles). El robot que antes llegue a la pared, gana.

Choca-gira

El objetivo del ejercicio es que, usando el sensor de infrarrojos en la parte delantera, el robot nunca se choque. Si se encuentra una pared, deberá girar un poco para intentar rodearla.

Persiguiendo al robot

Como ejercicio posterior al choca gira, y con el objetivo de hacerlo juego, y que los alumnos no se aburran de hacer todos los ejercicios de forma teórica -no olvidemos que la finalidad de este curso es interesar a los jóvenes en la robótica- diseñamos el ejercicio con la especificación de que el robot nunca puede chocarse, mientras que le perseguimos para intentar que se choque.

Esto requiere la única modificación de meter la funcionalidad *no chocar* dentro de un bucle. Como añadido, podemos introducir el concepto de función de programación, metiendo los bloques correspondientes al *no chocar* dentro de una función y ésta dentro del bucle.

Sigue-líneas

Utilizando el sensor de sigue líneas del kit del mbot y el circuito que también viene con él, hacemos que el robot siempre siga la línea negra.

Utilizando varios circuitos (o cinta aislante negra, siempre teniendo en cuenta que el suelo tendrá que ser blanco), podemos crear circuitos más complicados. Uno de los últimos ejercicios sería crear una ^{en}crucijada^{en} la que el robot deberá quedarse parado hasta que, con el control IR remoto, el estudiante le diga dónde ir.

Fútbol

Mezclando los dos últimos ejercicios, haremos que los robot jueguen un Uno contra Uno al fútbol: no se podrá salir del campo (con el sensor de sigue líneas) y tendrá que perseguir la pelota, dirigiéndoles con el mando IR. Los mismos estudiantes deberán darse cuenta que, si utilizan los mismos controles del mando IR para dirigir al robot, las señales de los robot se interferirán entre ellas.

Lucha de Sumo

Con el sensor de sigue líneas para no salirse del ring y el de ultrasonido delantero, deberemos perseguir al contrincante y empujarlo hasta echarle fuera del ring.

Detección de muros: sistema de seguridad

En este ejercicio introduciremos el sensor de ultrasonido par controlar la proximidad de objetos delanteros. Con el objetivo de que los alumnos vean que la robótica no es algo abstracto sino que la usan en su día a día -con sensores no tan diferentes de los que ellos usan-, les proponemos que diseñen un sistema de seguridad como el que los coches actuales (los mismos a los que ellos están acostumbrados). Como especificaciones del sistema de seguridad, tendremos los mismos que en un coche normal: cuando se acerque a 'x' distancia de un muro, pitará un poco; si se acerca más, pitará más rápido y, eventualmente, cuando esté muy cerca, se parará.

Tendremos que utilizar varios sensores y actuadores a la vez, con lo que la complejidad va subiendo poco a poco, aparte de que las condiciones no son siempre las mismas.

Diseñamos el ejercicio en dos iteraciones, para que primero aprendan como funciona el nuevo sensor:

1. Sólo la detección del muro, y que el robot pare cuando esté a la distancia adecuada.
2. Añadimos la funcionalidad "de coche": los pitidos y las diferentes distancias

Laberinto

Poniendo dos sensores de ultrasonido, el delantero y otro a la derecha (o a la izquierda, dependerá de ello cómo vayamos a resolver el laberinto), el objetivo es que el robot sea capaz de resolver y salir de cualquier laberinto de forma completamente autónoma.

Truco: Seguiremos la teoría de que, en laberintos de hasta cierta dificultad, siempre se puede salir de ellos siguiendo todo el tiempo una pared hacia el mismo lado (derecha o izquierda). Aunque se den, aparentemente, más vueltas, la salida está asegurada.

Aparcamiento autónomo

El ejercicio consiste en emular el sistema de aparcamiento autónomo de los coches, que los estudiantes conocen.

Lo haremos en dos niveles: primero sólo el hecho de aparcar en un hueco y luego incluiremos la búsqueda de un hueco válido donde el *coche* quepa.

En este ejercicio aprovechamos e introducimos el concepto de "función" de programación. En Scratch, **bloques**. En este ejemplo, la algoritmo de *hueco_valido* y el de *aparcar* son ejemplos perfectos de bloque, ya que los tenemos que repetir o condicionar, y así el programa se hará más legible.

Capítulo 4

Introducción a la programación con lenguaje Python

Ya vimos en el capítulo 1 una introducción al lenguaje Python, su sintaxis, operadores y palabras reservadas. En el presente capítulo empezamos ya a trabajar en este lenguaje para hacer pequeños programas básicos, como los ya tratados en el anterior capítulo 3.

Se recomienda realizar estos ejercicios directamente en el ordenador, bajo el entorno NinjaIDE. Éste nos facilita la escritura en este lenguaje, e incluye asimismo un terminal de salida, para que veamos el resultado de nuestro programa sin necesitar de acudir a un terminal externo.

4.1. Ejercicios básicos

4.1.1. Ejemplo inicial hello-world y operaciones básicas.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
print ("Hola mundo")
# raw_input() si quiero hacer una pausa de teclado

# esto es una cadena
c = "Hola Mundo"
# y esto es un entero
e = 23
# podemos comprobarlo
print (type(c))
print (type(e))

a = "uno"
b = "dos"
c = a + b # c es unodos
print (c)
```

```

c = a * 3 # c es unounounouno
print (c)

l = [22, True, "una lista", [1, 2]]
mi_var = l[3][1] # mi_var vale 2
print (mi_var)

```

Si nos fijamos, hemos de poner la línea inicial para que nuestro programa sea un auto-ejecutable. Indicamos al sistema dónde debe buscar el programa python, que será quien ejecute nuestro código. En los sucesivos ejemplos vamos a obviar esta línea para no resultar redundantes, pero recordemos que tenemos que ponerla siempre.

La segunda línea nos permite poner tildes y caracteres propios del castellano, como la ñ. Se recomienda también su uso siempre.

Los primeros ejercicios vendrán con la solución dada, para facilitar el aprendizaje. No obstante, se recomienda intentar encontrar una solución, y una vez hecho, comparar con la solución que se da.

4.1.2. Definir una función max() que tome como argumento dos números y devuelva el mayor de ellos.

(Python ya tiene una función max() incorporada, pero hacerla nosotros mismos nos servirá para iniciarnos.)

```

def max (n1, n2):
    if n1 < n2:
        print n2
    elif n2 < n1:
        print n1
    else:
        print "Son iguales"

```

(Uso el print para llamar a la función de la forma: max(8, 5). También se puede usar return.)

4.1.3. Escribir una función que tome un carácter y devuelva True si es una vocal, de lo contrario devuelve False.

```

def es_vocal (x):
    if x == "a" or x == "e" or x == "i" or x == "o" or x == "u":
        return True
    elif x == "A" or x == "E" or x == "I" or x == "O" or x == "U":
        return True

```

```
    else:  
        return False
```

4.1.4. Definir una función inversa() que calcule la inversión de una cadena.

(Por ejemplo la cadena *estoy programando* debería devolver la cadena *odnamargorp yotse*.

```
def inversa (cadena):  
    invertida = ""  
    cont = len(cadena)  
    indice = -1  
    while cont >= 1:  
        invertida += cadena[indice]  
        indice = indice + (-1)  
        cont -= 1  
    return invertida
```

4.1.5. Definir una función superposicion() que tome dos listas y devuelva True si tienen al menos 1 miembro en común o devuelva False de lo contrario.

```
def superposicion (lista1, lista2):  
    for i in lista1:  
        for x in lista2:  
            if i == x:  
                return True  
    return False
```

4.1.6. Definir un histograma procedimiento() que tome una lista de números enteros e imprima un histograma en la pantalla.

Ejemplo: procedimiento([4, 9, 7]) debería imprimir lo siguiente: **** ***** ***

```
def procedimiento (lista):  
    for i in lista:  
        print i * "x"
```

4.2. Ejercicios avanzados

- 4.2.1. Escribir una función maslarga() que tome una lista de palabras y devuelva la mas larga.

```
def mas_larga(lista):
    mas_larga = ""
    for i in lista:
        if len(i) > len(mas_larga):
            mas_larga = i
    return mas_larga
```

- 4.2.2. Escribir un programa que le diga al usuario que ingrese una cadena. El programa tiene que evaluar la cadena y decir cuantas letras mayúsculas tiene.

```
def c_mayusculas (cadena):
    cont = 0
    for i in cadena:
        if i != i.lower(): #Recordar que lower() convierte una cadena en minúsculas
            cont += 1
    print "La cadena tiene", cont, "mayuscula/s"
```

- 4.2.3. Definir una tupla con 10 edades de personas. Imprimir la cantidad de personas con edades superiores a 20.

```
def mayora20 (tup):
    cont = 0
    for i in tup:
        if i > 20:
            cont += 1
    print "Hay", cont, "numeros mayores a 20"
```

- 4.2.4. Crear una función contarvocales(), que reciba una palabra y cuente cuantas letras a tiene, cuantas letras e tiene y así hasta completar todas las vocales.

```
def contar_vocales(cadena):
    cadena = cadena.lower()
    vocales = "aeiou"

    for x in vocales:
```

```

contador = 0
for i in cadena:
    if i == x:
        contador += 1
print "Hay %d %s." % (contador, x)

```

4.3. Juegos básicos

4.3.1. Master mind

El juego consistirá en adivinar una cadena de números distintos. Al principio, el programa debe pedir la longitud de la cadena (de 2 a 9 cifras). Después el programa debe ir pidiendo que intentes adivinar la cadena de números. En cada intento, el programa informará de cuántos números han sido acertados (el programa considerará que se ha acertado un número si coincide el valor y la posición).

Un ejemplo podría ser:

1. Dime la longitud de la cadena: 4
2. Intenta adivinar la cadena: 1234
3. Con 1234 has adivinado 1 valores. Intenta adivinar la cadena: 1243
4. Con 1243 has adivinado 0 valores. Intenta adivinar la cadena: 1432
5. Con 1432 has adivinado 2 valores. Intenta adivinar la cadena: 2431
6. Con 2431 has adivinado 4 valores. Felicidades

4.3.2. Rimas

Escribe un programa que pida dos palabras y diga si riman o no. Si coinciden las tres últimas letras tiene que decir que riman. Si coinciden sólo las dos últimas tiene que decir que riman un poco y si no, que no riman.

4.3.3. Hipoteca

Haz un programa que pida al usuario una cantidad de dinero, una tasa de interés y un numero de años. Muestra por pantalla en cuánto se habrá convertido el capital inicial transcurridos esos años si cada año se aplica la tasa de interés introducida. Recordar que un capital C a un interés del x por cien durante n años se convierte en $C * (1 + x/100)$ elevado a n (años).

Ejemplo: Con una cantidad de 10000 euros al 4.5% de interés anual se convierte en 24117.14 euros al cabo de 20 años.

4.3.4. Descuento por compras

Este programa pide inicialmente la cantidad de dinero gastado en el total de compras de una persona. Si la cantidad es inferior a 100.00 euros, el programa dirá que el cliente tiene derecho a la promoción. Pero si la persona ingresa una cantidad en compras igual o superior a 100.00 euros, el programa genera de forma aleatoria un número entero del cero al cinco. Cada número corresponderá a un color diferente de cinco colores de bolas que hay para determinar el descuento que el cliente recibirá como premio. Si la bola aleatoria es color blanco, no hay descuento, pero si es uno de los otros cuatro colores, sí se aplicará un descuento determinado según la tabla que aparecerá, y ese descuento se aplicará sobre el total de compra que introdujo inicialmente el usuario, de manera que el programa mostrará un nuevo valor a pagar tras haber aplicado el descuento.

4.4. Ejercicios de listas

Las listas son un tipo de colección ordenada; también se les conoce como vectores.

En Python estas listas pueden contener enteros, booleanos, cadenas de texto, flotantes o incluso listas. Los elementos de una lista van encerrados entre corchetes ([]) y separados por comas cada uno de ellos. Veamos un ejemplo de lista en Python

```
lista = [1, 'dos', False, [45, 'cien']]
```

Esta lista tiene contiene: un entero (1), una cadena de texto (*dos*), un booleano (False) y otra lista con dos elementos (un entero, 45, y una cadena de texto, *cien*).

Cada elemento de una lista cuenta con un índice. Este índice comienza con el numero 0 para el elemento número 1. En nuestra lista (lista) el índice 0 corresponde al elemento (1), la cadena de texto que contiene la palabra *dos* sería el elemento número dos de la lista pero contiene el índice 1 y así sucesivamente con los demás elementos de la lista.

Así para asignar a una variable *num* el elemento con índice uno de la lista (lista) hacemos lo siguiente:

```
num = lista[1]
```

4.4.1. Suma

Escribe una función que reciba como parámetro una lista de números y devuelva la suma de todos ellos

4.4.2. Longitud

Escribe una función que reciba una lista como parámetro y devuelva la longitud (número de elementos) de la lista

4.4.3. Suma acumulada

Escribe una función que tome una lista de números y devuelva la suma acumulada, es decir, una nueva lista donde el primer elemento es el mismo, el segundo elemento es la suma del primero con el segundo, el tercer elemento es la suma del resultado anterior con el siguiente elemento y así sucesivamente. Por ejemplo, la suma acumulada de [1,2,3] es [1, 3, 6].

4.4.4. Media

Haz una función *media* que tome una lista y devuelva una nueva lista que contenga todos los elementos de la lista anterior menos el primero y el último.

4.4.5. Duplicados

Escribe una función llamada *elimina_duplicados* que tome una lista y devuelva una nueva lista con los elementos únicos de la lista original. No tienen porque estar en el mismo orden.

4.4.6. Ordenación

Escribe una función *ordenada* que tome una lista de números enteros y/o flotantes y devuelva una nueva lista con los valores colocados en orden ascendente.

4.4.7. Longitud strings

Escribe una función que reciba como parámetro una lista de strings, que imprima por pantalla por cada string:

”La longitud de /¡contenido del string¡/ es: ¡longitud¡”

por ejemplo: ”La longitud de /hola/ es: 4”

Capítulo 5

Entorno de programación JdeRobot-Kids

Ahora que estamos familiarizados con las diferentes sentencias básicas que se emplean en Programación, e igualmente con el lenguaje Python, podemos empezar a programar los distintos componentes robóticos incluidos en nuestro kit hardware robótico de Arduino.

5.1. Arquitectura

JdeRobot-arduino, JdeRobot. en PI-mbot. RaspberryPI. en Mbot.

5.2. Instalación del entorno

Ya explicamos en el primer capítulo de introducción (ver 1) que aunque por defecto el hardware de Arduino está preparado para ser programado bajo el IDE de Arduino, nosotros vamos a trabajar con Python. Así, hemos de notar que hay una serie de instrucciones iniciales que tenemos que incluir al comienzo del programa para que, una vez cargada la librería *PyFirmata* en la placa, podamos ejecutar el código Python como hacemos normalmente y éste se comunique con dicha placa.

5.2.1. Plataforma Arduino estándar

Lo primero que necesitamos es el entorno de programación *Arduino-IDE*. Para ello, recomendamos la instalación manual de la última versión que ofrezca Arduino en su web oficial ¹, en vez de utilizar la que aparece en los repositorios.

Este entorno lo vamos a necesitar, fundamentalmente, para cargar los distintos *firmwares* que vayamos necesitando en la placa Arduino. Así, por ejemplo, para poder usar código Python en el robot, hemos de cargar el *firmware PyFirmata* en éste. Para ello, basta con

¹<https://www.arduino.cc/en/main/software>

abrir el IDE de Arduino, ir a *Archivo - Ejemplos - Firmata - StandardFirmata* y cargar este *firmware* en la placa con el botón *Cargar*.

También debemos tener instalada las siguientes librerías de Python en nuestro PC:

```
sudo apt-get install python-pip python-serial  
sudo pip install pyfirmata
```

5.2.2. Plataforma mBot

En caso de que usemos concretamente el robot *mBot*, los pasos a seguir son ligeramente diferentes:

1. Descargar Arduino IDE de aquí ²
2. Descargar librería oficial make-block de aquí ³
3. Descargar el firmware para mBot de aquí ⁴.
4. Instalar librerías:

```
sudo aptitude install python3-all python3-pip
```

5. Instalar Complementos de python3:

```
sudo pip3 install cython  
sudo aptitude install libusb-1.0-0-dev libudev-dev  
sudo pip3 install hidapi  
sudo pip3 install pyserial  
sudo pip3 install pymata
```

(Si tenemos problemas con *hidapi*, podemos instalarla mediante la orden `sudo aptitude install python3-hidapi`).

6. Copiar carpeta (*GitLab JdeRobot-kids*)/*infraestructura/ArduinoJderobot* a `/usr/lib/python3.5`.

7. Para asegurarse de usar "python3", conviene reenlazar "python", aunque esta operación puede perjudicar a otras aplicaciones que usen python2.7.

```
rm /usr/bin/python (eliminamos actual enlace a python2.7 o la que sea)  
ln -s /usr/bin/python3.5 /usr/bin/python (así python apunta a python3.5)
```

8. Ir a la carpeta descargada de Arduino-IDE y desde ahí lanzar Arduino (*arduino*). Vamos al menú *.Abrir*, seleccionamos el firmware (descargado en paso 3) y cargamos en robot mediante botón *Cargar*.

²<https://www.arduino.cc/en/main/software>

³<https://github.com/Makeblock-official/Makeblock-Libraries>

⁴(*GitLab JdeRobot-kids*)/*infraestructura/mbot_jderobot/mbot_jderobot.ino*

9. Una vez hecho este último paso, ya podemos usar en el robot mBot ejemplos de `(GitLab JdeRobot-kids)/exercises/python/capitulo5/mbot` simplemente lanzándolos desde el PC con `python3.5 nombre.py`

Si tenemos problemas al intentar conectar con el robot, es probable que se deba a un tema de permisos sobre el puerto serial (en Ubuntu registrados como `/dev/ttyUSB0` y similares). Para solucionarlos podemos optar por lanzar los programas Python como superusuarios (`sudo python3.5 nombre.py`) o, de forma más segura, lanzarlos como habitualmente con nuestro usuario, habiéndolo añadido al grupo `dialout` de Ubuntu (grupo al que están suscritos los puertos seriales) mediante la orden: `sudo adduser tu-usuario dialout`.

5.2.3. Plataforma Pi-Mbot

Como se ha comentado en el capítulo 2, para darle más autonomía y potencia al mbot, le añadimos un ordenador propio: la placa Raspberry-Pi3. Hemos elegido esta versión de Raspberry por la tarjeta Wifi integrada, y por su base Linux. Para tener un robot autónomo, debemos primero preparar la raspberry:

1. Descargar el sistema Raspbian de la web oficial
2. Formatear la tarjeta microSD en la que instalaremos Raspbian.
3. Grabar la imagen Raspbian en la SD, por ejemplo con el programa Etcher.

Una vez tengamos la placa operativa, le instalamos JdeRobot-kids 5.2.2 igual que lo hicimos para el mbot. Lo único "difícil" de usar la raspberry en vez de un ordenador normal, es que, al ser sólo una placa, para acceder a ella por primera vez e instalar JdeRobot, deberemos conectarla a un monitor, teclado y ratón. Además de instalar la plataforma, tendremos que añadirle una red wifi, editando los archivos del sistema `/etc/network/interfaces` y `/etc/wpa_supplicant/wpa_supplicant.conf`.

Una vez tengamos una red, podemos desconectar monitor y periféricos y conectarnos, para ejecutar cualquier práctica de JdeRobot o propia en el nuevo Mbot, por SSH (a la IP que tenga en la red).

5.3. Interfaz Python para programar robots

encenderLedPlaca(self, indice, r=255, g=255, b=255)

Función que enciende un led específico de la placa del robot.

Parameters

indice: número de LED que se desea encender (0 – enciende ambos, 1 – enciende el derecho, 2 – enciende el izquierdo)
 (*type=entero*)
 r: intensidad de color rojo [0-255]
 (*type=entero*)
 g: intensidad de color verde [0-255]
 (*type=entero*)
 b: intensidad de color azul [0-255]
 (*type=entero*)

encenderLed(self, puerto, indice, r=255, g=255, b=255)

Función que enciende un led específico externo a la placa del robot.

Parameters

puerto: puerto al que están enchufados los leds
 (*type=entero*)
 indice: número de LED que se desea encender
 (*type=entero*)
 r: intensidad de color rojo [0-255]
 (*type=entero*)
 g: intensidad de color verde [0-255]
 (*type=entero*)
 b: intensidad de color azul [0-255]
 (*type=entero*)

apagarLedPlaca(self, indice)

Función que apaga un led específico de la placa del robot.

Parameters

indice: número de LED que se desea encender (0 – apaga ambos, 1 – apaga el derecho, 2 – apaga el izquierdo)
 (*type=entero*)

apagarLed(self, puerto, indice)

Función que apaga un led específico de la placa del robot.

Parameters

puerto: puerto al que están enchufados los leds

(type=entero)

indice: número de LED que se desea apagar

(type=entero)

escribirTexto(self, puerto, texto, dx=0, dy=7, brillo=3)

Función que escribe texto en el panel led.

Parameters

puerto: Puerto del panel Led

(type=entero)

texto: texto a escribir

(type=text)

dy: fila inferior donde empezar a escribir

(type=entero)

dx: columna inzquierda donde empezar a escribir

(type=entero)

brillo: brillo de los leds

(type=entero)

escribirFrase(self, puerto, texto, brillo=3)

Función que escribe texto en el panel led con desplazamiento para ver la frase entera.

Parameters

puerto: Puerto del panel Led

(type=entero)

texto: texto a escribir

(type=text)

brillo: brillo de los leds

(type=entero)

dibujosPosibles(self)

Función que Devuelve una lista con los nombres de los dibujos predefinios.

@return lista con posibles dibujos

pintarDibujo(self, puerto, nombreDibujo, brillo=4)

Función que Permite pintar en el panel led Dibujos predefinidos.

Parameters

puerto: Puerto del panel Led

(*type=entero*)

nombreDibujo: Nombre del dibujo predefinido

(*type=text*)

brillo: brillo de los leds

(*type=entero*)

mover(self, velW, velW)

Función que permite mover al robot, con una velocidad lineal y una velocidad de giro.

Parameters

velV: Velocidad de avance del robot. Si queremos que retrocesa, ésta será negativa. En unidades del SI, *m/s*

(*type=entero*)

velW: Velocidad de giro del robot. Si queremos girar a la izquierda, la velocidad será positiva y, para girar a la derecha, la velocidad será negativa. En unidades del SI, *rad/s* (*type=entero*)

dibujar(self, puerto, dibujo, posicionX=0, posicionY=0, brillo=4, ancho=16)

Función que permite dibujar en el panel led. Para dibujar se le pasa una lista que representa el panel led. Cada columna esta representada por un entero en binario Empezando por abajo. por ejemplo, si encendemos el primero, el cuarto y el octavo: $1 + 8 + 128 = 137$

Parameters

puerto: Puerto del panel Led

(*type=entero*)

dibujo: lista que representa el dibujo (leds que se encienden)

(*type=lista*)

posicionX: columna donde empieza el dibujo

(*type=entero*)

posicionY: fila donde empieza el dibujo

(*type=entero*)

brillo: brillo de los leds

(*type=entero*)

ancho: ancho en columnas del dibujo

(*type=entero*)

escribirReloj(self, puerto, hora, minuto, brillo=3)

Función que escribe un reloj digital en el panel led.

Parameters

puerto: Puerto del panel Led
 $(type=entero)$

hora: horas
 $(type=entero)$

minuto: minutos
 $(type=entero)$

brillo: brillo de los leds
 $(type=entero)$

borrarMatriz(self, puerto)

Función que borra el panel led.

Parameters

puerto: Puerto del panel Led
 $(type=entero)$

leerBoton(self)

Función que retorna si el botón ha sido pulsado o no en formato numérico.
 Devuelve 0 si el botón no se ha pulsado y 1 en caso contrario.

playTono(self, tono, tiempo)

Función que reproduce un tono musical utilizando el zumbador integrado en el robot.

Parameters

tono: tono que se quiere reproducir como texto.
 C3":131,"D3":147,"E3":165,"F3":175,"G3":196,"A3":220,"B3":247,
 C4":262,"D4":294,"E4":330,"F4":349,"G4":392,"A4":440,"B4":494,
 C5":523,"D5":587,"E5":659,"F5":698,"G5":784,"A5":880,"B5":988
 $(type=string)$

tiempo: tiempo que dura el tono en milisegundos.
 $(type=entero)$

leerUltrasonido(self)

Función que devuelve el valor leído por el sensor de ultrasonidos del robot.

leerIntensidadLuz(self)

Función que devuelve el valor leído por el sensor de intensidad de luz del robot.

leerIRSiguelineas(self)

Función que devuelve el valor leído por el sensor de IR del siguelíneas.

leerIntensidadSonido(*self*)

Función que devuelve el valor leído por el sensor de ultrasonidos del robot.

leerPotenciómetro(*self*)

Función que devuelve el valor leído por el sensor de ultrasonidos del robot.

avanzar(*self, vel*)

Función que hace avanzar al robot en línea recta a una velocidad dada como parámetro.

Parameters

vel: velocidad de avance del robot (máximo 255)

(*type=entero*)

retroceder(*self, vel*)

Función que hace retroceder al robot en línea recta a una velocidad dada como parámetro.

Parameters

vel: velocidad de retroceso del robot (máximo 255)

(*type=entero*)

parar(*self*)

Función que hace detenerse al robot.

girarIzquierda(*self, vel*)

Función que hace rotar al robot sobre sí mismo hacia la izquierda a una velocidad dada como parámetro.

Parameters

vel: velocidad de giro del robot (máximo 255)

(*type=entero*)

girarDerecha(*self, vel*)

Función que hace rotar al robot sobre sí mismo hacia la derecha a una velocidad dada como parámetro.

Parameters

vel: velocidad de giro del robot (máximo 255)

(*type=entero*)

Capítulo 6

Programación en Python sobre Arduino: sensores y actuadores

Ahora que estamos familiarizados con las diferentes sentencias básicas que se emplean en Programación, e igualmente con el lenguaje Python, podemos empezar a programar los distintos componentes robóticos incluidos en nuestro kit hardware robótico de Arduino.

6.1. Lectura de valor de pulsador

En este ejemplo vamos a leer el valor según el pulsador de la placa del mBot que se haya pulsado.

```
# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexion

robot = MBot("/dev/ttyUSB0") # Conectamos con el robot, hay que indicar el puerto del
                             # robot

print("Empieza a pulsar")

while True:
    but = robot.leerBoton() # leemos el valor de los botones
    # comparamos con cada uno de los botones para saber cual es
```

```

if (but == 1):
    print("Boton Pulsado")

time.sleep(0.2) # esperamos 0.2 segundos para no saturar al robot

```

6.2. Control de LED mediante pulsador

Exponemos el código detalladamente comentado de este ejemplo para que el usuario se familiarice con las instrucciones adicionales que mencionábamos, así como con la lectura/escritura de/en placa.

```

import pyfirmata

# placa representa a nuestro objeto de Arduino
# puerto al que tenemos conectado la placa Arduino
placa = pyfirmata.Arduino('/dev/ttyACM0')

# version de Firmata que esta usando la placa
print('Firmata version: %d.%d' % placa.get_firmata_version())
# version de pyFirmata que tenemos en el ordenador
print('pyFirmata version:', pyfirmata.__version__)

# Si se utilizan uno o mas pines de entrada es necesario crear
# un thread iterador, de lo contrario la placa puede seguir
# enviando datos por la conexion serial hasta producir un
# desbordamiento. Asi que lanzamos un thread:
pyfirmata.util.Iterator(placa).start()

# (d=digital,a=analog):pin 8:input(o=output,p=pwm)
entrada = placa.get_pin('d:8:i')
# para poder leer las senales recibidas a traves del pin 8
entrada.enable_reporting()
salida = placa.get_pin('d:12:o')

try:
    encendido = False
    while True: # bucle infinito
        if entrada.read():
            # este metodo devuelve 0 si el voltaje de entrada
            # es menor a 2.5V, o 1 si es mayor a 2.5V.
            encendido = not encendido
            # escribe a un pin digital, enviando como

```

```

# argumento False (0V) o True (5V)
salida.write(encendido)
# para pausar el programa durante 0.2 segundos,
# si no se pone el programa puede no sensar
# nuestra pulsacion de boton
placa.pass_time(0.2)

finally:
    salida.write(False)
    placa.exit()

```

6.3. Control de servos gráficamente con librería Tk

En este ejemplo vamos a controlar los dos servos de una placa Arduino mediante un slider, gracias a la librería Tk.

```

import pyfirmata
from Tkinter import *

# placa representa a nuestro objeto de Arduino
# puerto al que tenemos conectado la placa Arduino
placa = pyfirmata.Arduino('/dev/ttyACM0')

# version de Firmata que esta usando la placa
print('Firmata version: %d.%d' % placa.get_firmata_version())

# version de pyFirmata que tenemos en el ordenador
print('pyFirmata version:', pyfirmata.__version__)

# Si se utilizan uno o mas pines de entrada es necesario crear un thread
# iterador, de lo contrario la placa puede seguir enviando datos por la
# conexion serial hasta producir un desbordamiento. Lanzamos un thread:
iterator = pyfirmata.util.Iterator(placa)
iterator.start()

# configuramos los pines 12 y 13 como servos derecho e izquierdo:
pinDerecho = placa.get_pin('d:12:s')
pinIzquierdo = placa.get_pin('d:13:s')

def move_servo1(a):
    pinDerecho.write(a)

```

```

def move_servo2(b):
    pinIzquierdo.write(b)

# configuramos el GUI
root = Tk()

# dibujamos un par de sliders para controlar las posiciones de los servos
scale1 = Scale(root,
    command = move_servo1,
    to = 175,
    orient = HORIZONTAL,
    length = 400,
    label = 'Servo Derecho (95 = centro/parado)')
scale1.pack(anchor = CENTER)

scale2 = Scale(root,
    command = move_servo2,
    to = 175,
    orient = HORIZONTAL,
    length = 400,
    label = 'Servo Izquierdo (95 = centro/parado)')
scale2.pack(anchor = CENTER)

# lanzamos el bucle del GUI Tk
root.mainloop()

```

6.4. Uso del zumbador

En este ejemplo programamos una función que recibe como entrada el pin de conexión del zumbador, el patrón que queremos que siga y el número de repeticiones del mismo. Hay dos patrones.

```

import pyfirmata
from time import sleep

def patronZumbador(pin, repeticion, patron):
    patron1 = [0.8, 0.2, 0.8]
    patron2 = [0.1, 0.3, 0.5]
    flag = True

    for i in range(repeticion):

```

```

if patron == 1:
    p = patron1
elif patron == 2:
    p = patron2

else:
    print "Introduzca un patron valido: 1 o 2."
    exit

for delay in p:
    if flag is True:
        placa.digital[pin].write(1)
        flag = False
        sleep(delay)
    else:
        placa.digital[pin].write(0)
        flag = True
        sleep(delay)

placa.digital[pin].write(0)
placa.exit()

# placa representa a nuestro objeto de Arduino
placa = pyfirmata.Arduino('/dev/ttyACM0')

print('Firmata version: %d.%d' % placa.get_firmata_version())
print('pyFirmata version:', pyfirmata.__version__)

sleep(5) # esperamos para que la conexion sea estable

patronZumbador(4, 10, 2) # llamamos a funcion que ejecuta un patron de sonidos

```

6.5. Uso del potenciómetro

En el siguiente ejemplo usamos un potenciómetro y mostramos su salida.

```

import pyfirmata
import os
from pyfirmata import Arduino, util
from time import sleep

# placa representa a nuestro objeto de Arduino

```

```

placa = pyfirmata.Arduino('/dev/ttyACM0')

print('Firmata version: %d.%d' % placa.get_firmata_version())
print('pyFirmata version:', pyfirmata.__version__)

sleep(5) # esperamos para que la conexion sea estable

it = util.Iterator(placa)

it.start()

a0 = placa.get_pin('a:0:i') # a0 pin analogico usado como pin de entrada

try:
    while True: # monitorizamos continuamente la salida del potenciómetro
        p = a0.read()
        print p

except KeyboardInterrupt: # evita que al cerrar programa no se cierre bien
    board.exit()
    os._exit()

```

6.6. Motor DC

En este ejemplo modulamos la velocidad y el tiempo de ejecución en un motor de corriente alterna.

```

import pyfirmata
import os
from pyfirmata import Arduino, util
from time import sleep

# placa representa a nuestro objeto de Arduino
placa = pyfirmata.Arduino('/dev/ttyACM0')

print('Firmata version: %d.%d' % placa.get_firmata_version())
print('pyFirmata version:', pyfirmata.__version__)

sleep(5) # esperamos para que la conexion sea estable

motor = placa.get_pin('d:3:p') # motor conectado en pin 3 en modo PWM

```

```

def motorDCControl(s, t):
    motor.write(s/100.00)
    sleep(t)
    motor.write(0)

try:
    while True: # monitorizamos continuamente el estado del motor
        s = input("Por favor, introduzca un valor de velocidad al motor (1-100): ")

        if (s > 100) or (s <= 0):
            print "Por favor, introduzca un valor adecuado."
            placa.exit()
            break

        t = input("¿Durante cuánto tiempo? (segundos)")
        motorDCControl(s, t)

except KeyboardInterrupt: # evita que al cerrar programa no se cierre bien
    placa.exit()
    os._exit

```

6.7. Modulo LED

En este ejemplo mostramos un uso básico del módulo LED que incluye el pack del mBot Plus. Se trata de varios ejercicios de formas de encendido de los distintos LEDs.

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("COM4") # Conectamos con el robot
    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexión

```

```

while True:
    robot.apagarLed(0,4)
    modo = input("Selecciona modo de encendido \n a) Carrusel rojo (por d")
    if modo == "a":
        r = 255
        g = 0
        b = 0
    elif modo == "b":
        r = 0
        g = 255
        b = 0
    elif modo == "c":
        r = 0
        g = 0
        b = 255
    elif modo == "d":
        r = int(input("Introduce el valor de rojo "))
        g = int(input("Introduce el valor de verde "))
        b = int(input("Introduce el valor de azul "))
        if r > 255 and r < 0:
            r = 0
        if g > 255 and g < 0:
            g = 0
        if b > 255 and b < 0:
            b = 0
    elif modo == "e":
        break
    else:
        r = 255
        g = 0
        b = 0

    vueltas = int(input("Introduce numero de vueltas "))
    for v in range(vueltas):
        robot.apagarLed(0,4)
        for i in range (1,5):
            robot.encenderLed(i,r,g,b,4) # encendemos el LED izquierdo
            time.sleep(0.2) # esperamos 0.2 segundos para no saturar

```

6.8. Sensor de luz

En el siguiente ejemplo vamos a hacer uso de una nueva librería: *PyMata*. Para instalarla, y suponiendo que ya tenemos instalada la herramienta *pip*, lanzamos el siguiente comando en terminal:

```
sudo pip install pymata
```

Y en la placa de Arduino basta con cargar la librería PyFirmata como lo veníamos haciendo previamente. La librería PyFirmata es muy útil como interfaz del protocolo Firmata. Soporta, como hemos visto, modos varios: analógico, digital, PWM, y Servo. Pero se queda corta en cuanto al protocolo I2C, que es el que usaremos en este ejemplo de comunicación inalámbrica. Veamos.

```
import time
from PyMata.pymata import PyMata

port = PyMata("COM5")

port.i2c_config(0, port.ANALOG, 4, 5)

port.i2c_read(0x23, 0, 2, port.I2C_READ) # pedimos al dispositivo enviar 2 bytes

time.sleep(3) # esperamos al mismo

data = port.i2c_get_read_data(0x23) # leemos informacion

LuxSum = (data[1] << 8 | data[2]) >> 4 # obtenemos valores de luz de lo recibido

lux = LuxSum/1.2

print str(lux) + ' lux'

firmata.close()
```

6.9. Infrarrojos

En este ejemplo vamos a ver cómo leer el valor leído por el sensor de infrarrojos situado en la parte inferior de nuestro mBot, cuyo propósito no es otro que seguir una línea, propuesto como ejercicio.

```
# Importando modulos necesarios
import sys
import signal
```

```

import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot, hay que indicar el puerto del robot

    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexion

    # introduce tu código aquí
    while True:
        val = robot.leerIRSiguiLineas()
        print(val)
        time.sleep(0.1)

```

6.10. Comunicación vía Bluetooth

Así como en el anterior ejemplo hacíamos uso de la librería PyMata, en este caso vamos a usar la librería *serial* y la librería *pygame*. La primera de ellas nos permitirá vía comunicación serial enviar/recibir datos a/de la placa Arduino mediante el dispositivo Bluetooth HC-06, que es el que utilizamos. Por su parte, la librería *pygame* nos permite dibujar una ventana, así como leer por teclado la tecla que pulse el usuario.

```

import serial
import pygame

ser = serial.Serial('/dev/rfcomm0', 9600) # inicia comunicación serial

# Iniciamos pygame con parámetros para dibujar una ventana y mostrar el ratón:
pygame.init()
screen = pygame.display.set_mode((640, 480))
pygame.display.set_caption('Robot!')
pygame.mouse.set_visible(1)

val = '--'

while val != 'stop':
    events = pygame.event.get()

```

```

for event in events:
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_UP:
            ser.write('a')
        elif event.key == pygame.K_LEFT:
            ser.write('i')
        elif event.key == pygame.K_RIGHT:
            ser.write('d')
        elif event.key == pygame.K_DOWN:
            ser.write('r')
        elif event.key == pygame.K_ESCAPE:
            val = 'stop'
    if event.type == pygame.KEYUP:
        ser.write('s')

```

6.11. Lectura de ultrasonidos

En el siguiente ejemplo hacemos uso de la librería serial para leer los valores que sensa el ultrasonidos.

```

import serial # Hacemos uso de la libreria serial

puerto = serial.Serial('/dev/ttyACM0', 9600) # puerto serial de comunicacion

while (1==1):
    if (puerto.inWaiting()>0):
        myData = puerto.readline()
        print myData # mostramos los valores leidos del sensor de Ultrasonidos

```

6.12. Lectura de intensidad de sonido

En este ejemplo vemos cómo podemos obtener el valor de intensidad de sonido dado por el micrófono incorporado en el mBot Plus.

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot
#from ArduinoJdeRobot import BOTON_CEN, BOTON_DER, BOTON_IQZ, BOTON_SUP, BOTON_INF

# Señal para poder desconectar del robot bien

```

```

def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot, hay que indicar el pu
    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexió
    # introduce tu código aquí
    while True:
        val = robot.leerIntensidadSonido()
        print(val)
        time.sleep(0.1)

```

6.13. Matriz de LEDs

En los ejemplos que vamos a ver a continuación, hacemos uso de la matriz de LED de 8x16 que nos ofrece MakeBlock para mBot. Esta matriz de LEDs nos permite poner caras, caracteres o animaciones al robot. Veamos algunos ejemplos.

6.13.1. Mostrar un texto

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot
    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexió

```

```

robot.escribirTexto(4, "Hola")

time.sleep(3)
robot.borrarMatriz(4)

```

6.13.2. Mostrar la hora

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot
    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexión

    robot.escribirReloj(4, 12, 11)

    time.sleep(3)
    robot.borrarMatriz(4)

```

6.13.3. Mostrar ojos cerrándose

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')

```

```

    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot

    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexion

    ojos1 = [0,24,36,66,66,36,24,0,0,24,36,66,66,36,24,0]
    ojos2 = [0,24,36,34,34,36,24,0,0,24,36,34,34,36,24,0]
    ojos3 = [0,24,20,18,18,20,24,0,0,24,20,18,18,20,24,0]
    ojos4 = [0,8,12,10,10,12,8,0,0,8,12,10,10,12,8,0]
    ojos5 = [0,8,4,6,6,4,8,0,0,8,4,6,6,4,8,0]
    ojos6 = [0,8,4,2,2,4,8,0,0,8,4,2,2,4,8,0]

    lista = [ojos1, ojos2, ojos3, ojos4, ojos5, ojos6, ojos5, ojos4, ojos3, ojos2]

    while True:
        for i in lista:
            robot.dibujar(4, i)
            time.sleep(0.2)

    robot.borrarMatriz(4)

```

6.13.4. Mostrar un símbolo

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot

    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexi

```

```

robot.pintarDibujo(4, "FLECHA_ABAJO")

time.sleep(3)
robot.borrarMatriz(4)

```

6.13.5. Mostrar en bucle dibujos predefinidos

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot
    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexión

    dibujos = robot.dibujosPosibles()

    for i in dibujos:
        robot.pintarDibujo(4, i)
        time.sleep(3)

    robot.borrarMatriz(4)

```

6.14. Garra

En este ejemplo hacemos uso de otro componente extra que nos ofrece MakeBlock denominado Gripper. Este complemento consta de un servo que le permite tener posiciones de abrir y cerrar. Veamos cómo se maneja un servo:

```

from lib.mBot import *
bot = mBot()
bot.startWithSerial("/dev/ttyUSB0")

```

```

while(1):
try:

bot.moverServo(2, 1, 90)
print( "[2,1] Garra Abriendo...")
sleep(2)
bot.moverServo(2, 1, 0)
print( "[2,1] Garra Cerrando...")
sleep(2)

except Exception:
print (str(Exception.msg))

```

6.15. Garra montada sobre un mini cuello Pan-Tilt

Una vez dominado el manejo de la garra, vamos a montar ésta sobre otro complemento: un mini cuello Pan-Tilt. Éste nos va a permitir tener más libertad de movimiento de la garra. Ahora, además de poder abrir y cerrar la misma, podremos rotarla sobre sí misma, así como moverla a derecha e izquierda.

Estos dos nuevos grados de libertad lo conseguimos gracias a los dos servos que incorpora este cuello Pan-Tilt. Veamos.

```

from lib.mBot import *
bot = mBot()
bot.startWithSerial("/dev/ttyUSB0")

while(1):
try:

bot.moverServo(2, 1, 90)
print( "[2,1] Garra Abriendo...")
sleep(2)
bot.moverServo(2, 1, 0)
print( "[2,1] Garra Cerrando...")
sleep(2)

bot.moverServo(1, 1, 190)
print( "[1,1] Garra Rotando a Derecha...")
sleep(2)
bot.moverServo(1, 1, 0)

```

```

print( "[1,1] Garra Rotando a Izquierda...")
sleep(2)

bot.moverServo(1, 2, 45)
print( "[1,2] Garra a Derecha...")
sleep(2)
bot.moverServo(1, 1, 135)
print( "[1,2] Garra a Izquierda...")
sleep(2)

except Exception:
    print (str(Exception.msg))

```

6.16. Mando a distancia

En esta sección usaremos el mando a distancia del Mbot, comprobando qué devuelve el mando (tipo string).

```

from mBotReal import *
if __name__ == '__main__':
robot = MBotReal("/dev/ttyUSB0")
while(1):
valor_mando = robot.leerMandoIR()
if valor_mando != "null":
print (valor_mando)
time.sleep(0.2)

```

Los valores posibles para el mando están integrados en un diccionario para que sean más reconocibles, puesto que los valores reales del sensor son números, no la tecla del mando que nosotros pulsamos. Por lo tanto, tendremos los siguientes valores -en tipo string- para las diferentes teclas del mando.

- A
- B
- C
- D
- E
- F

- ARRIBA
- IZQUIERDA
- DERECHA
- ABAJO
- AJUSTES
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

Es importante recordar que, en la programación -en este caso con Python- se diferencia entre mayúsculas y minúsculas, por lo que será importante escribir **exactamente** el valor correspondiente a cada botón si queremos usarlo en nuestro programa.

6.17. Ejercicios

1. Leer el botón pulsado e imprimir por pantalla cual se está pulsando.
2. Leer el potenciómetro e imprimir por pantalla el valor.
3. Encender un led con un pulsador y apagarlo con otro.
4. Encender 3 leds con 3 pulsadores diferentes y apagarlos con el central.
5. Encender y apagar un led pulsado en el mismo botón.
6. Poner una frase en la matriz led del mBot.
7. Mostrar un cronómetro desde 1 minuto a 0 en la matriz led del mBot.
8. Pintar todos los dibujos predefinidos en la matriz led del mBot.
9. Pintar unos ojos abriéndose y cerrándose en la matriz led del mBot.
10. Hacer un control remoto (coche teledirigido) con el mando IR a distancia.

Capítulo 7

Programación en Python sobre Arduino: Comportamientos en robots

7.1. Choca gira

La idea detrás de el ejercicio choca-gira es implementar un algoritmo que permita que el robot se mueva por un entorno cualquiera sin llegar a colisionar con ningún obstáculo en su marcha y sin necesidad de supervisión humana, esto es: que sea autónomo.

Para lograr nuestro objetivo, tendremos que hacer uso tanto del sensor de ultrasonidos, como de los motores del robot. El algoritmo choca-gira consiste en que el robot, circulando por su cuenta por el mundo, detecte los obstáculos a su paso y trate de evitarlos sin llegar a colisionar.

Con ayuda del sensor de ultrasonidos, que se utiliza para medir distancias entre el sensor y los obstáculos en su dirección, podremos saber si un objeto está demasiado cerca como para que haya riesgo de colisión, y así evitarlo.

El funcionamiento del ejercicio deberá ser el siguiente:

1. El robot estará avanzando por su entorno de forma autónoma, ya sea en línea recta o con una trayectoria aleatoria.
2. En todo momento, el sensor de ultrasonidos estará recogiendo lecturas de su entorno mientras el robot se mueve.
3. Cuando la distancia que reciba el sensor de ultrasonidos sea menor que una distancia de seguridad elegida por el programador significará que el robot tiene un obstáculo delante.
4. Una vez detectado el obstáculo el robot deberá detener su marcha y evitarlo. Típicamente girando a izquierda o derecha una cantidad de tiempo aleatoria.

A continuación se propone una solución sencilla para el problema propuesto. Nótese que el algoritmo propuesto está diseñado para funcionar sobre el robot arduino MBot:

```

# Importando modulos necesarios
import time
from ArduinoJdeRobot import MBot

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot, hay que indicar el pu
    # introduce tu codigo aqui
    vel = 100
    while True:

        val = robot.leerUltrasonido()
        print(val)
        if (val < 10):
            robot.retroceder(vel)
            time.sleep(0.5)
            robot.girarDerecha(vel)
            time.sleep(0.3)
            robot.avanzar(vel)
            time.sleep(0.3)

```

7.2. Palmadas

A continuación se propone otro ejercicio orientado a los comportamientos autónomos del robot, en este caso lo hemos llamado algoritmo de las palmadas.

El ejercicio es muy sencillo, se trata de hacer que el robot reaccione a nuestros estímulos sonoros; más concretamente, que reaccione a las palmadas que demos. Para lograr este comportamiento será necesario hacer uso del sensor de sonido, del cual recogeremos los valores de ruido del entorno del robot. La idea básica de este ejercicio es que el robot estando parado o en movimiento, pare o retome la marcha respectivamente siempre que .escucheüna palmada dada por nosotros.

El funcionamiento del ejercicio deberá ser el siguiente:

1. El robot comienza estando parado.
2. Cuando el sensor de sonido detecta un ruido más alto que el ruido ambiente, significará que se ha dado una palmada.
3. Cuando el robot detecte la palmada deberá emprender su marcha en línea recta o con una trayectoria aleatoria.

4. El robot seguirá avanzando hasta que detecte que se ha dado otra palmada, en cuyo caso detendrá su marcha.
5. Este comportamiento se repetirá en bucle, con lo que el robot avanzará/parará en función de las palmadas que .“escuche”.

A continuación se propone una solución sencilla para el problema propuesto. En este caso, se completa el algoritmo con la implementación de un calibrador de sonido que mide el ruido ambiente para calcular por encima de qué umbral de intensidad de sonido se habrá dado una palmada. Nótese que el algoritmo propuesto está diseñado para funcionar sobre el robot arduino MBot:

```
# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

def calibracion(robot):
    ambiente = 0
    for i in range(100):
        ambiente += robot.leerIntensidadSonido()
    ambiente = ambiente / 100

    print("el ruido ambiente es:",ambiente)

    calibrado = 0
    lvl = 0
    while lvl < 5 or lvl > 10:
        lvl = input("Vamos a calibrar el sensor. Elige nivel de calibrado (5-")
        lvl = int(lvl)

    print("Da una palmada")
    for i in range(lvl):
        while True:
            calibrar = robot.leerIntensidadSonido()
            if calibrar > ambiente + 150:
                time.sleep(1)
                calibrado += calibrar
```

```

        break
    print("Da otra")
calibrado /= lvl
print("el nivel de calibrado es", calibrado)
return calibrado

if __name__ == "__main__":
    robot = MBot("COM4") # Conectamos con el robot, hay que indicar el puerto del
    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexión

    threshold = calibracion(robot)
    avanzando = True

    # introduce tu código aquí
    while True:

        val = robot.leerIntensidadSonido()
        if val > threshold:
            time.sleep(1)
            avanzando = not avanzando
            if avanzando:
                print("Para!")
                robot.parar(100)
            else:
                print("Avanza!")
                robot.avanzar(100)

```

7.3. ¡Huye de la luz!

El siguiente ejercicio pondrá a prueba el sensor de luz del robot. Este ejercicio, lo hemos titulado ¡huye de la luz! lo que induce a intuir el comportamiento que deberá llevar a cabo el robot.

Se pide elaborar un algoritmo que permita que el robot, avanzando en una trayectoria recta o aleatoria, al detectar una fuente de luz, la evite rápidamente. Para la resolución de la práctica se deberá hacer uso del sensor de luz del robot que recogerá los valores de intensidad lumínica del entorno del robot, de tal modo que el sensor devolverá un valor con la intensidad de luz ambiente, y otro valor mayor si se le apunta con alguna fuente de luz (linterna) directamente al sensor. Con estos valores, se puede calcular si el robot

está bajo un foco de luz (no ambiente) o no.

El funcionamiento del ejercicio deberá ser el siguiente:

1. El robot estará avanzando por su entorno de forma autónoma, ya sea en línea recta o con una trayectoria aleatoria.
2. En todo momento, el sensor de luz estará recogiendo lecturas de su entorno mientras el robot se mueve.
3. Cuando la intensidad de luz que reciba el sensor sea mucho mayor que la intensidad de la luz ambiente se sabrá que el robot está bajo una fuente de luz a evitar.
4. Una vez detectada la fuente de luz a evitar, el robot deberá detener su marcha y evitarlo. Típicamente girando a izquierda o derecha una cantidad de tiempo aleatoria.

A continuación se propone una solución sencilla para el problema propuesto. Nótese que el algoritmo propuesto está diseñado para funcionar sobre el robot arduino MBot:

```
# Importando modulos necesarios
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot, hay que indicar el pu
    ambiente = 0
    for i in range(10000):
        ambiente += robot.leerIntensidadLuz()
    ambiente = ambiente / 10000
    print("La intensidad de luz ambiente es ambiente es:",ambiente)

    # introduce tu codigo aqui
    vel = 100
    while True:
        val = robot.leerIntensidadLuz()
        print(val)
        if (val > ambiente+100):
```

```

robot.retroceder(vel)
time.sleep(0.5)
robot.girarDerecha(vel)
time.sleep(0.3)
robot.avanzar(vel)
time.sleep(0.3)

```

7.4. Lucha de sumos

El ejercicio siguiente es bastante divertido, si se realiza en grupos y con muchos contendientes. Se trata de una lucha robótica (sin violencia) entre dos MBot. Este ejercicio pondrá a prueba múltiples sensores y actuadores.

Se pide elaborar un algoritmo que permita que dos robots puedan llevar a cabo una pelea en la que las reglas son las siguientes:

- Ambos robots comenzarán la pelea de espaldas uno a otro. De esta manera estarán obligados a buscarse.
- Se puede hacer uso de cualquier sensor/actuador que se desee para la pelea.
- Ganará aquel robot que logre sacar del ring al otro.
- Está prohibido tocar a los robots durante la pelea, a menos que se hayan quedado bloqueados sin poder moverse, en cuyo caso se volverá a la situación inicial, con ambos robots dándose la espalda.

Como acabamos de exponer, se puede hacer uso de cualquier sensor/actuador que se desee para realizar la pelea aunque los mínimos obligatorios son: sensor de ultrasonidos, sensor siguelineas y motores; en nuestro caso de ejemplo haremos uso de:

- Sensor de ultrasonidos, para detectar al contrincante y medir distancias a él.
- Sensor de siguelineas, para no salirse del ring.
- Motores del motor, para locomoción del robot.
- ServoMotor, a modo de palanca, para inhabilitar al contrincante.

Lo que se pide en este ejercicio es:

1. Que el robot pueda navegar por dentro del ring sin llegar a salirse si no está siendo empujado por el contrincante. Dado que el ring consiste en un cerco de color negro rodeando un área (cuadrada o circular) del suelo, podemos hacer uso del sensor de siguelineas para determinar si estamos encima de la línea o no. En caso de estar encima debemos retroceder, girar y seguir nuestra marcha.
2. Que el robot sea capaz de localizar a su contrincante dentro del ring. Dado que se trata de una pelea de sumos, el objetivo es echar al oponente del ring empujándolo; para ello debemos localizarlo e ir a por él. Para

esto podemos valernos del sensor de ultrasonidos, ya que al medir distancias podemos saber si el contrincante está a nuestro alcance o no. 3. Buscar una estrategia de combate. Una vez localizado al oponente, tenemos varias vías de aproximación como la fuerza bruta (empujar a máxima velocidad y de frente), control (empujar poco a poco yéndonos cada vez), con más actuadores, una combinación de varias, etc. Lo más sencillo es utilizar la fuerza bruta, pero para este ejemplo vamos a combinarla con el uso de un servomotor que nos ayudará a levantar al oponente.

El funcionamiento del ejercicio deberá ser el siguiente:

1. El robot empezará de espaldas a su oponente dentro del ring.
2. Al comenzar el combate, deberá buscar a su oponente dentro del ring haciendo uso del sensor de ultrasonidos. Una buena estrategia es girar sobre sí mismo hasta localizarlo, ¡jojo con la velocidad de giro! podemos no verlo si giramos muy rápido.
3. Una vez localizado el oponente, debemos intentar sacarlo del ring empujándolo. Como hemos expuesto más arriba, podemos hacer uso de cualquier sensor/actuador extra que nos facilite la labor.
4. Mientras se desarrolla la pelea, no debemos salir en ningún momento del ring, o perderemos.

A continuación se propone una solución sencilla para el problema propuesto. Nótese que el algoritmo propuesto está diseñado para funcionar sobre el robot arduino MBot:

```
# Importando modulos necesarios
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot, hay que indicar el pu
    while True:
        distancia = robot.leerUltrasonidos()
        linea = robot.leerIRSigueLineas()

        if (linea == 3 or linea == 2 or linea == 1): #estamos al borde del ring!
            robot.parar()
            robot.retroceder(100)
```

```

        time.sleep(1)
        robot.girarIzquierda(100) # podemos hacer un giro aleatorio.
        time.sleep(1)
    else:
        if distancia < 50: #oponente localizado!
            robot.avanzar(255)
            if distancia < 10: #estamos pegados, usemos el servo!
                robot.moverServo(4,1,170)
                time.sleep(0.5)
            else:
                robot.moverServo(4,1,60)
                time.sleep(0.5)
        else: #buscamos a nuestro oponente
            robot.girarDerecha(90)

```

7.5. Piedra, papel o tijera

En este clásico juego vamos a necesitar la matriz de LEDs, un zumbador y un pulsador. La idea es obtener un número aleatorio en cada tirada del juego, para que juguemos contra el ordenador. El símbolo que saque éste se representará en la matriz de LEDs, emitiendo un pitido según sea piedra, papel o tijera.

```

# Importando modulos necesarios
import sys
import signal
import time
from ArduinoJdeRobot import MBot

# Señal para poder desconectar del robot bien
def signal_handler(sig, frame):
    print('Has pulsado Ctrl+C')
    sys.exit(0)

if __name__ == "__main__":
    robot = MBot("/dev/ttyUSB0") # Conectamos con el robot
    signal.signal(signal.SIGINT, signal_handler) #activamos la señal de desconexion

    # Simbolo P I E
    piedra = [0,0,0,126,74,74,48,0,62,0,62,42,42,0,0,0,0]
    # Simbolo P A

```

```

papel = [0,0,0,126,74,74,48,0,30,40,40,30,0,0,0,0]
# Simbolo T I J
tijera = [0,0,64,64,126,64,64,0,62,0,2,2,60,0,0,0]

lista = [piedra, papel, tijera]

patron1 = [0.8, 0.2, 0.8]
patron2 = [0.1, 0.3, 0.5]
patron3 = [0.5, 0.1, 0.8]

while True:
    for i in range(lista):
        if i == 0:
            robot.dibujar(4, piedra)
            p = patron1

        if i == 1:
            robot.dibujar(4, papel)
            p = patron2

        if i == 2:
            robot.dibujar(4, tijera)
            p = patron3

    but = robot.leerBoton() # leemos el valor de los botones
    # comparamos con cada uno de los botones para saber cual es
    if (but == 1):
        print("Boton Pulsado")
        time.sleep(2.0) # esperamos 2 seg. para que usuario compare con lo que ha
    else:
        time.sleep(0.2)

    for delay in p:
        robot.playTono("C3", delay)

robot.borrarMatriz(4)

```

7.6. Ejercicios

7.6.1. Semáforo acústico

Combinando la matriz de leds y el sensor de sonido vamos a crear un programa que haga que nuestro robot reaccione al sonido mostrando su estado en la matriz de leds. Cuando la intensidad de la música esté por debajo del límite que establezcamos, el robot dormirá (ojos cerrados y sin moverse); si elevamos su intensidad, el robot despertará (abre los ojos y se mueve, baila).

7.6.2. Los bolos

En este programa, el robot partirá de una posición que elijamos y avanzará hasta una línea negra que detectará gracias al sensor de infrarrojos. En esta línea habrá colocada una pelota que será golpeada por el robot. Al desplazarse la pelota derribará unos bolos situados a una distancia que estimemos adecuada.

7.6.3. Golf

Basándonos en la idea del ejercicio anterior, podemos hacer que cuando el robot llegue a la línea negra, accione un servo-motor que hará las veces de palo de golf (que hay que construir). Éste golpeará una pelota que intentará meter en un hoyo (que también hay que idear) situado en una distancia que consideremos adecuada.

7.6.4. Carrera de velocidad

Diseñaremos un circuito con cinta negra sobre una superficie blanca no deslizante. Mediante el sensor de infrarrojos el robot irá siguiendo la línea negra e intentará hacer el circuito en el menor tiempo posible. Habrá que jugar con las velocidades convenientemente para que el robot consiga hacer bien el circuito en tiempo record, pero sin salirse.

7.6.5. Esgrima

Si partimos del diseño realizado para la práctica del juego del Golf y añadimos un sensor de contacto, podemos tener un luchador de esgrima, que competirá contra otro robot. El robot, al ser tocado, enciende los leds. Así, el primero en encender los leds es el perdedor.

7.6.6. Levantamiento de pesas

En este programa haremos uso de la pinza para levantar un objeto. La idea es acercarse a la zona donde estará colocado el objeto, pararse y cogerlo para posteriormente retroceder

una distancia que estimemos

Capítulo 8

Programación en Python sobre PiBot: sensores y actuadores

Ahora que estamos familiarizados con las diferentes sentencias básicas que se emplean en Programación, e igualmente con el lenguaje Python, podemos empezar a programar distintos componentes robóticos sobre la placa de Raspberry Pi 3.

8.1. Sensor de contacto con LED

En este ejemplo vemos el uso de sensor de contacto en Raspberry Pi 3, con led on/off.

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
LedPin = 10
TouchPin = 11

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
    GPIO.setup(TouchPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

def loop():
    while True:
        if GPIO.input(TouchPin) == GPIO.LOW:
            print '...led on'
            GPIO.output(LedPin, GPIO.LOW) # led on
        else:
            print 'led off...'
            GPIO.output(LedPin, GPIO.HIGH) # led off

def destroy():


```

```

GPIO.output(LedPin, GPIO.HIGH)      # led off
GPIO.cleanup()                      # Release resource

if __name__ == '__main__':          # Program start from here
setup()
try:
loop()
except KeyboardInterrupt:          # When 'Ctrl+C' is pressed, the child program destroy() wi
destroy()

```

8.2. Ultrasonidos básico de cuatro pines

En este ejemplo controlamos el sensor de ultrasonidos básico, que cuenta con cuatro pines, usando dos de ellos de salida y entrada del ECHO respectivamente.

```

import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
TRIG = 23 # Puerto por el que disparamos la emision del ultrasonido
ECHO = 24 # Puerto por el que recibimos la senal de 5V (reducido a 3,3V) del ultrasonido

#set GPIO direction (IN / OUT)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(TRIG, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(ECHO) == 0:

```

```

StartTime = time.time()

# save time of arrival
while GPIO.input(ECHO) == 1:
    StopTime = time.time()

# time difference between start and arrival
TimeElapsed = StopTime - StartTime
# multiply with the sonic speed (34300 cm/s)
# and divide by 2, because there and back
distance = (TimeElapsed * 34300) / 2

return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Distancia estimada = %.1f cm" % dist)
            time.sleep(1)

    # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Medicion interrumpida por usuario")
        GPIO.cleanup()

```

8.3. Ultrasonidos avanzado de tres pines

En este ejemplo usamos un sensor de ultrasonidos avanzado, con tres pines, que usa uno de ellos de entrada y salida del ECHO.

```

import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

PUERTO = 22 # Puerto por el que recibimos y emitimos la señal de 3,3V del ultrasonido

def distance():
    GPIO.setup(PUERTO, GPIO.OUT)
    # set Trigger to HIGH

```

```

GPIO.output(PUERTO, True)

# set Trigger after 0.01ms to LOW
time.sleep(0.00001)
GPIO.output(PUERTO, False)

StartTime = time.time()
StopTime = time.time()

GPIO.setup(PUERTO, GPIO.IN)
# save StartTime
while GPIO.input(PUERTO) == 0:
    StartTime = time.time()

# save time of arrival
while GPIO.input(PUERTO) == 1:
    StopTime = time.time()

# time difference between start and arrival
TimeElapsed = StopTime - StartTime
# multiply with the sonic speed (34300 cm/s)
# and divide by 2, because there and back
distance = (TimeElapsed * 34300) / 2

return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Distancia estimada = %.1f cm" % dist)
            time.sleep(1)

    # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Medicion interrumpida por usuario")
        GPIO.cleanup()

```

8.4. Emisor de infrarrojos

En este ejemplo controlamos la emisión de infrarrojos que, junto con el ejemplo siguiente, hacen la combinación perfecta de sistema de emisión/recepción de infrarrojos.

```

#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

LedPin = 10      # pin11

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
    GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

def loop():
    while True:
        print '...led on'
        GPIO.output(LedPin, GPIO.HIGH) # led on
        time.sleep(0.5)
        print 'led off...'
        GPIO.output(LedPin, GPIO.LOW) # led off
        time.sleep(0.5)

def destroy():
    GPIO.output(LedPin, GPIO.LOW)    # led off
    GPIO.cleanup()                  # Release resource

if __name__ == '__main__':
    # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy() will be executed
        destroy()

```

8.5. Receptor de infrarrojos

En este ejemplo controlamos la recepción de infrarrojos que, junto con el ejemplo anterior, hacen la combinación perfecta de sistema de emisión/recepción de infrarrojos.

```

#!/usr/bin/env python
import RPi.GPIO as GPIO

IrPin  = 11
LedPin = 10

```

```

Led_status = 1

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
    GPIO.setup(IrPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.output(LedPin, GPIO.LOW) # Set LedPin high(+3.3V) to off led

def swLed(ev=None):
    global Led_status
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status) # switch led status(on-->off; off-->on)
    if Led_status == 1:
        print 'led on...'
    else:
        print '...led off'

def loop():
    GPIO.add_event_detect(IrPin, GPIO.FALLING, callback=swLed) # wait for falling
    while True:
        pass # Don't do anything

def destroy():
    GPIO.output(LedPin, GPIO.LOW)      # led off
    GPIO.cleanup()                   # Release resource

if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy() will
        destroy()

```

8.6. Control de servo básico

En el siguiente ejemplo vemos cómo controlar un servo básico (no continuo).

```

import RPi.GPIO as GPIO      #Importamos la libreria RPi.GPIO
import time                  #Importamos time para poder usar time.sleep

GPIO.setmode(GPIO.BOARD)     #Ponemos la Raspberry en modo BOARD
GPIO.setup(24,GPIO.OUT)      #Ponemos el pin 21 como salida

```

```

p = GPIO.PWM(24,50)          #Ponemos el pin 21 en modo PWM y enviamos 50 pulsos por se
p.start(7.5)                 #Enviamos un pulso del 7.5% para centrar el servo

try:
    while True:             #iniciamos un loop infinito

        p.ChangeDutyCycle(4.5)  #Enviamos un pulso del 4.5% para girar el servo hac
        time.sleep(0.5)         #pausa de medio segundo
        p.ChangeDutyCycle(10.5)  #Enviamos un pulso del 10.5% para girar el servo ha
        time.sleep(0.5)         #pausa de medio segundo
        p.ChangeDutyCycle(7.5)  #Enviamos un pulso del 7.5% para centrar el servo d
        time.sleep(0.5)         #pausa de medio segundo

except KeyboardInterrupt:      #Si el usuario pulsa CONTROL+C entonces...
    p.stop()                  #Detenemos el servo
    GPIO.cleanup()             #Limpiamos los pines GPIO de la Raspberry y cerramo

```

8.7. Servo avanzado de rotación continua

A continuación vemos cómo controlar un servo de rotación continua.

```

#!/usr/bin/env python

import sys, tty, termios, time, pigpio

servos = [4,18] # De momento solo usamos el servo conectado en el pin 24

dit = pigpio.pi()

def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def motor1_forward():
    dit.set_servo_pulsewidth(servos[0], 1700)

```

```

def motor1_reverse():
    dit.set_servo_pulsewidth(servos[0], 1300)

def motor1_stop():
    dit.set_servo_pulsewidth(servos[0], 1550)

def motor2_forward():
    dit.set_servo_pulsewidth(servos[1], 1300)

def motor2_reverse():
    dit.set_servo_pulsewidth(servos[1], 1700)

def motor2_stop():
    dit.set_servo_pulsewidth(servos[1], 1490)

def forward():
    print "Avanzando"
    motor1_forward()
    motor2_forward()

def reverse():
    print "Retrocediendo"
    motor1_reverse()
    motor2_reverse()

def turn_left():
    print "Girando izquierda"
    dit.set_servo_pulsewidth(servos[1], 1400)
    dit.set_servo_pulsewidth(servos[0], 1490)

def turn_right():
    print "Girando derecha"
    dit.set_servo_pulsewidth(servos[0], 1600)
    dit.set_servo_pulsewidth(servos[1], 1550)

def stop():
    print("DETENIENDO")
    motor1_stop()
    motor2_stop()
    time.sleep(1)
    for s in servos: # stop servo pulses
        dit.set_servo_pulsewidth(s, 0)

```

```
dit.stop()

while True:
    char = getch()

    print(" " + char)

    if char == "w":
        forward()

    elif char == "s":
        reverse()

    elif char == "a":
        turn_left()

    elif char == "d":
        turn_right()

    elif char == "x":
        stop()
        break

dit.stop()
```

Capítulo 9

Programación en Python sobre PiBot: Comportamientos en robots

9.1. Follow Ball

En este ejemplo vemos cómo usando la cámara propia de la Raspberry, la PiCam, podemos obtener imágenes en las cuales detectamos un objeto de color (en este ejemplo, azul) y, calculando continuamente la diferencia entre la posición del objeto en la imagen respecto al centro, podemos comandar las órdenes oportunas a los motores para que siempre el objeto esté situado en el centro, a la vez que avanza hacia él.

```
# IMPORTANTE: antes de lanzarlo hay que ejecutar el demonio de pigpio con "sudo pigpi
# import the necessary packages
import imutils
import cv2
import argparse
import sys, traceback, Ice
import easyiceconfig as EasyIce
import jderobot
import numpy as np
import threading
import sys, tty, termios, time, pigpio

servos = [4,18] # Usamos los servos conectados a los pines 4 y 24
dit = pigpio.pi()

# Establecemos los limites inferior y superior en HSV de algunos colores basicos
GREEN_RANGE = ((29, 86, 6), (64, 255, 255))
RED_RANGE = ((139, 0, 0), (255, 160, 122))
BLUE_RANGE = ((110,50,50), (130,255,255))
TENNIS_BALL = ((30, 60, 165), (40, 120, 255))

def motor1_forward():
```

```

dit.set_servo_pulsewidth(servos[0], 1600)

def motor1_reverse():
    dit.set_servo_pulsewidth(servos[0], 1300)

def motor1_stop():
    dit.set_servo_pulsewidth(servos[0], 1525)

def motor2_forward():
    dit.set_servo_pulsewidth(servos[1], 1400)

def motor2_reverse():
    dit.set_servo_pulsewidth(servos[1], 1700)

def motor2_stop():
    dit.set_servo_pulsewidth(servos[1], 1510)

def forward (tiempo):
    print "Avanzando"
        motor1_forward()
        motor2_forward()
        time.sleep(tiempo)

def reverse (tiempo):
    print "Retrocediendo"
        motor1_reverse()
        motor2_reverse()
        time.sleep(tiempo)

def turn_left (tiempo):
    print "Avanzando izquierda"
    dit.set_servo_pulsewidth(servos[0], 1540)
    dit.set_servo_pulsewidth(servos[1], 1490)
        time.sleep(tiempo)

def turn_right (tiempo):
    print "Avanzando derecha"
    dit.set_servo_pulsewidth(servos[0], 1550)
    dit.set_servo_pulsewidth(servos[1], 1500)
        time.sleep(tiempo)

def stop (tiempo):
    print("DETENIENDO")

```

```

motor1_stop()
motor2_stop()
time.sleep(tiempo)
#for s in servos: # stop servo pulses
# dit.set_servo_pulsewidth(s, 0)

#dit.stop()

def seguirPelota (image, color):
    # resize the frame
    image = imutils.resize(image, width=600)

    # convert to the HSV color space
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # construct a mask for the color specified
    # then perform a series of dilations and erosions
    # to remove any small blobs left in the mask
    mask = cv2.inRange(hsv, color[0], color[1])
    mask = cv2.erode(mask, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)

    moments = cv2.moments(mask)
    # find contours in the mask and
    # initialize the current center
    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
    center = None

    # only proceed if at least one contour was found
    if len(cnts) > 0:
        # find the largest contour in the mask, then use
        # it to compute the minimum enclosing circle and
        # centroid
        c = max(cnts, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)
        area = moments['m00']

        # only proceed if the radius meets a minimum size
        if radius > 10: # if(area > 100000):
            # draw the circle border and the centroid
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
            cv2.circle(image, (int(x), int(y)), int(radius), (0, 255, 0), 2)

```

```

cv2.circle(image, center, 5, (0, 255, 255), -1)

#determine the x and y coordinates of the center of the object
#we are tracking by dividing the 1, 0 and 0, 1 moments by the area
x = moments['m10'] / area
y = moments['m01'] / area

print 'x: ' + str(int(x)) + ' y: ' + str(int(y)) + ' area: ' + str(area)
moverPiBot (round(x, -1),y,round(area, -5))
time.sleep(0.15)
'''

if area > 3000000:
    stop (1); # paramos los motores
    for s in servos: # stop servo pulses
        dit.set_servo_pulsewidth(s, 0)

        dit.stop()
'''

# show the image to our screen
cv2.imshow("Imagen en curso", image)
# key = cv2.waitKey(1) & 0xFF
if cv2.waitKey(1) & 0xFF == ord('q'):
    stop (1); # paramos los motores
    for s in servos: # stop servo pulses
        dit.set_servo_pulsewidth(s, 0)

        dit.stop()
        clean_up()
        sys.exit()

return center

def moverPiBot (x,y,area):
    if x > 330:
        turn_right((x-320)/(320*12))
    elif x < 310:
        turn_left((320-x)/(320*12))
    elif ((310 <= x <= 330) and area<1500000):
        forward(0.05)
    elif ((310 <= x <= 330) and area>4000000):
        reverse(0.05)

def damePosicionDeObjetoDeColor(image, color):

```

```

# resize the image
image = imutils.resize(image, width=600)

# convert to the HSV color space
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# construct a mask for the color specified
# then perform a series of dilations and erosions
# to remove any small blobs left in the mask
mask = cv2.inRange(hsv, color[0], color[1])
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)

# find contours in the mask and
# initialize the current center
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
center = None

# only proceed if at least one contour was found
if len(cnts) > 0:
    # find the largest contour in the mask, then use
    # it to compute the minimum enclosing circle and
    # centroid
    c = max(cnts, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

    # only proceed if the radius meets a minimum size
    if radius > 10:
        # draw the circle border
        cv2.circle(image, (int(x), int(y)), int(radius), (0, 255, 0), 2)

        # and the centroid
        cv2.circle(image, center, 5, (0, 255, 255), -1)

# show the image to our screen
cv2.imshow("Imagen en curso", image)
key = cv2.waitKey(1) & 0xFF

return center

```

```

if __name__ == "__main__":
    ic = EasyIce.initialize(sys.argv)
    properties = ic.getProperties()
    basecameraL = ic.propertyToProxy("Camera.Proxy")
    cameraProxy = jderobot.CameraPrx.checkedCast(basecameraL)
    key=-1

    color = BLUE_RANGE
#    color = TENNIS_BALL

    while key != 1048689:
        # grab the current image
        imageData = cameraProxy.getImageData("RGB8")
        imageData_h = imageData.description.height
        imageData_w = imageData.description.width
        image = np.zeros((imageData_h, imageData_w, 3), np.uint8)
        image = np.frombuffer(imageData.pixelData, dtype=np.uint8)
        image.shape = imageData_h, imageData_w, 3
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    rotated = imutils.rotate_bound(image, 180) # Because we have inverted piCamera on PiB

        center = seguirPelota (rotated, color)

        if center != None:
            print center
        else:
            print "NO OBJECT"

    # cleanup the camera and close any open windows
    camera.release()
    cv2.destroyAllWindows()

```

9.2. Choca-gira usando visión

En este ejemplo vemos cómo usando la cámara propia de la Raspberry, la PiCam, y usando la librería de Geometría proyectiva así como la el modelo de cámara Pin-Hole, podemos estimar distancias a los objetos como si de un Sonar visual se tratara y montar encima, por ejemplo, un comportamiento de choca-gira.

Aquí sólo se muestra la parte correspondiente a la función dameSonarVisual, que hace uso de la biblioteca de PiBot y la librería ProGeo del proyecto JdeRobot-Kids.

```
def dameSonarVisual ():
```

```

    , , ,
Función que devuelve el array de puntos [X,Y] (Z=0) correspondiente al obstáculo detectado
    , , ,

cameraModel = loadPiCamCameraModel ()
puntosFrontera = 0
fronteraArray = numpy.zeros((ANCHO_IMAGEN,2), dtype = "float64")

image = dameImagen ()
hsvImg = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
bnImg = cv2.inRange(hsvImg, GREEN_MIN, GREEN_MAX)

pixel = Punto2D()
pixelOnGround3D = Punto3D()
tmp2d = Punto2D()
puntosFrontera = 0
j = 0

# Ground image: recorremos la imagen de abajo a arriba (i=filas) y de izquierda a derecha (j=columnas)
while (j < ANCHO_IMAGEN): # recorrido en columnas
    i = LARGO_IMAGEN-1
    esFrontera = None
    while ((i>=0) and (esFrontera == None)): # recorrido en filas
        pos = i*ANCHO_IMAGEN+j # posicion actual

        pix = bnImg[i, j] # value 0 or 255 (frontera)

        if (pix != 0): # si no soy negro
            esFrontera = True # lo damos por frontera en un principio, luego veremos
            # Calculo los demás vecinos, para ver de qué color son...
            c = j - 1
            row = i
            v1 = row*ANCHO_IMAGEN+c

            if (esFrontera == True): # si NO SOY COLOR CAMPO y alguno de los vecinitos ES color campo
                pixel.x = j
                pixel.y = i
                pixel.h = 1
                fronteraImg[i,j] = 255

            # obtenemos su backproject e intersección con plano Z en 3D
            pixelOnGround3D = getIntersectionZ (pixel)

```

```

# vamos guardando estos puntos frontera 3D para luego dibujarlos con PyGame
fronteraArray[puntosFrontera] [0] = pixelOnGround3D.x
fronteraArray[puntosFrontera] [1] = pixelOnGround3D.y
puntosFrontera = puntosFrontera + 1
#print "Hay frontera en pixel [",i," , ",j," ] que intersecta al suelo en [",pixelOnGround3D.x," , ",pixelOnGround3D.y," ]"

i = i - 1
j = j + 5
return fronteraArray

```

9.3. Choca-gira usando ultrasonidos

Al igual que en el ejercicio anterior, pero en este caso usando el sensor de ultrasonidos.

```

#!/usr/bin/env python
import RPi.GPIO as GPIO
import sys, tty, termios, time, pigpio

servos = [4,18] # De momento solo usamos el servo conectado en el pin 24

''' servo[1] is clockwise ==> faster to slower [1280...1480]
    servo[0] is counterclockwise ==> slower to faster [1520...1720]
'''

dit = pigpio.pi()
GPIO.setmode(GPIO.BCM)
PUERTO = 22 # Puerto por el que recibimos y emitimos la señal de 3,3V del ultrasonido

def motor1_forward():
    dit.set_servo_pulsewidth(servos[0], 1720)

def motor1_reverse():
    dit.set_servo_pulsewidth(servos[0], 1280)

def motor1_stop():
    dit.set_servo_pulsewidth(servos[0], 1500)

def motor2_forward():
    dit.set_servo_pulsewidth(servos[1], 1280)

def motor2_reverse():
    dit.set_servo_pulsewidth(servos[1], 1720)

```

```

def motor2_stop():
    dit.set_servo_pulsewidth(servos[1], 1480)

def forward():
    print "Avanzando"
    motor1_forward()
    motor2_forward()

def reverse():
    print "Retrocediendo"
    motor1_reverse()
    motor2_reverse()

def turn_left():
    print "Girando izquierda"
    dit.set_servo_pulsewidth(servos[0], 1530) # + 20 over min
    dit.set_servo_pulsewidth(servos[1], 1380) # mid speed

def turn_right():
    print "Girando derecha"
    dit.set_servo_pulsewidth(servos[0], 1620) # mid speed
    dit.set_servo_pulsewidth(servos[1], 1460) # -20 over min

def stop():
    print("DETENIENDO")
    motor1_stop()
    motor2_stop()
    time.sleep(1)
    for s in servos: # stop servo pulses
        dit.set_servo_pulsewidth(s, 0)

    dit.stop()

def distance():
    GPIO.setup(PUERTO, GPIO.OUT)
    # set Trigger to HIGH
    GPIO.output(PUERTO, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(PUERTO, False)

```

```

StartTime = time.time()
StopTime = time.time()

GPIO.setup(PUERTO, GPIO.IN)
# save StartTime
while GPIO.input(PUERTO) == 0:
    StartTime = time.time()

# save time of arrival
while GPIO.input(PUERTO) == 1:
    StopTime = time.time()

# time difference between start and arrival
TimeElapsed = StopTime - StartTime
# multiply with the sonic speed (34300 cm/s)
# and divide by 2, because there and back
distance = (TimeElapsed * 34300) / 2

return distance

def loop():
dist = distance ()
print(dist)
if (dist < 10):
reverse ()
time.sleep(0.5)
turn_left ()
time.sleep(0.3)
forward ()
time.sleep(0.3)

```

9.4. Sigue-líneas usando infrarrojos

En este ejercicio, el robot sigue una línea que contrasta claramente con el fondo. Para ello, usa sensor de infrarrojos. Aquí se expone el fragmento de código correspondiente al uso de los sensores.

```

def loop():
irleft = GPIO.input(irRecLeft)
irright = GPIO.input(irRecRight)

if irleft == 0 and irright == 0:

```

```

forward()
elif irleft == 1 and irright == 0:
turn_left()
elif irleft == 0 and irright == 0:
forward()
elif irleft == 1 and irleft == 1:
stop()
elif irleft == 0 and irright == 0:
forward()
elif irleft == 0 and irright == 1:
turn_right()
elif irleft == 0 and irright == 0:
forward()
elif irleft == 1 and irleft == 1:
stop()
else:
stop()

if __name__ == '__main__':
setup()
try:
loop()
except KeyboardInterrupt:
GPIO.cleanup()

```

9.5. Sigue-líneas usando visión

En este ejercicio, el robot sigue una línea que contrasta claramente con el fondo. Para ello, en lugar del sensor de infrarrojos, como en el ejercicio anterior, emplea la cámara PiCamera. Aquí se expone el fragmento de código correspondiente al bucle de recepción y tratamiento de la imagen vertida por la cámara.

```

def loop():
originalImg = vs.read()
originalImg = imutils.resize(originalImg, width=320)
rgb_im = imutils.rotate_bound(originalImg, 180) # Because we have inverted piCamera on
width, height = rgb_im.size
pix=im.load()
print width, height
r,g,b=pix[0,0]
initialcolor=r+g+b

```

```

linedetected=0
linestartxvalue=0
lineendxvalue=0

y= int (height/8)

for x in range (0,width):
r,g,b= pix[x,y]
currentcolor=r+g+b

if (currentcolor > initialcolor +10 ) or (currentcolor < initialcolor -10): # cambio
linedetected=1
linestartxvalue=x
print x, y
break

y= height - int(height/8)

for x in range (0,width):
r,g,b= pix[x,y]
currentcolor=r+g+b

if (currentcolor > initialcolor +10 ) or (currentcolor <initialcolor -10):
linedetected=1
lineendxvalue=x
print x, y
break

if (linestartxvalue > lineendxvalue +20):
turn_right()
print "turn right"

if (linestartxvalue < lineendxvalue -20):
turn_left()
print "turn left"

if (lineendxvalue + 19 > linestartxvalue > lineendxvalue - 20):
forward()
print "forward"

```