



GRADO EN INGENIERÍA EN TECNOLOGÍA DE LAS TELECOMUNICACIONES

Curso Académico 2021/2022

Trabajo Fin de Grado

ROBÓTICA EDUCATIVA CON PYTHON Y MBOT

Autor : Eva García Domingo
Tutor : Jose María Cañas Plaza

Índice general

1. Introducción	1
2. Objetivos	2
2.1. Objetivos	2
2.1.1. Programación en Python de un robot basado en Arduino	2
2.1.2. Propuesta educativa completa, basada en Scratch y Python y escalonada según dificultad	2
2.2. Requisitos	3
2.3. Metodología	3
3. Infraestructura	5
3.1. Entorno	5
3.2. Hardware	6
3.2.1. Placa Arduino	6
3.2.2. MBot	6
3.2.2.1. Sensores	8
3.2.2.2. Actuadores	9
3.3. Software	11
3.3.1. Scratch y mBlock	11
3.3.2. Arduino	14
3.3.3. Python	15
4. Plataforma PyBo-Kids-2.0	17
4.1. Diseño	17
4.1.1. Comunicaciones	17
4.1.2. Programa residente	18
4.1.2.1. Comunicación serial	19
4.1.2.2. Uso de sensores y actuadores	20
4.1.3. Programa PC	22
4.2. Resultado: programa PC y programa residente	23
4.2.1. Protocolo de mensajes	24
4.2.2. Bibliotecas	25
5. Aplicación educativa	27
5.1. Contexto	27
5.2. Primera etapa: Scratch	27
5.2.1. Objetivos docentes: metodología	27

<i>ÍNDICE GENERAL</i>	1
5.2.2. Prácticas	27
5.3. Segunda etapa: PyBo-Kids en Python	27
5.3.1. Objetivos docentes: metodología	27
5.3.2. Introducción a Python	27
5.3.3. Prácticas	27
6. Conclusiones	28
Bibliografía	29

Capítulo 1

Introducción

Esto es un ejemplo. Ver en [1] [2]

Capítulo 2

Objetivos

En este explicaremos los objetivos de este Trabajo Fin de Grado, así como la intencionalidad de ellos y la metodología para llevarlos a cabo.

2.1. Objetivos

Como se ha adelantado en la introducción, el carácter, y por tanto el objetivo, principal de este Trabajo Fin de Grado es educacional. Siguiendo este carácter, abordaremos dos caminos complementarios, explicados a continuación.

2.1.1. Programación en Python de un robot basado en Arduino

El objetivo técnico será proporcionar una biblioteca en python que poder utilizar para programar un robot concreto basado en una placa base de Arduino, con el fin de dar una opción de lenguaje más sencilla. Para que esto sea posible, también se trabajará en un programa residente, en Arduino, que grabar en la placa base, que ofrezca una comunicación con la biblioteca de python. Así, se podrán programar los sensores y actuadores del robot con funciones en python, cuya lógica estará en esta biblioteca y de la cual no tendrán que preocuparse los alumnos.

2.1.2. Propuesta educativa completa, basada en Scratch y Python y escalonada según dificultad

Ofreceremos una propuesta educativa, para un curso escolar, orientada según niveles de dificultad y con objetivos docentes. Para ellos, crearemos diferentes ejercicios, o prácticas, de robótica -con el robot educacional Mbot, describiendo los objetivos conceptuales que se persiguen y ordenándolas con la finalidad de un aprendizaje gradual de programación. Este curso estará orientado principalmente a alumnos de Educación Primaria o Secundaria (alumnos sin conocimientos previos de programación). Para esta propuesta, utilizaremos tanto el lenguaje de programación por bloques Scratch, proporcionado por el fabricante, como nuestro *middleware* en Python, más complejo y, por tanto, como segunda parte avanzada.

2.2. Requisitos

Respecto a estos objetivos, hay ciertos requisitos que se han marcado para cumplirlos:

- La plataforma desarrollada, llamada PyBo-Kids-2.0, deberá ejecutarse en el lenguaje de programación Python, y el robot educativo podrá programarse en este lenguaje.
- Esta plataforma contendrá los métodos necesarios, en una biblioteca, para empaquetar la lógica de los sensores y actuadores del robot, haciendo esta lógica invisible al usuario (alumno).
- PyBo-Kids tendrá asociada una biblioteca de Arduino, desarrollada para entenderse con la biblioteca de Python y establecer una comunicación exitosa entre el robot (lado residente) y el PC.
- Esta biblioteca Arduino estará pre-cargada en el robot, para no tener que cambiar el programa del lado residente cada vez que se quiera programar algo nuevo en el Mbot.
- Los ejercicios y prácticas educativas se diseñarán para utilizar el robot Mbot y sus periféricos, así como la plataforma del fabricante y la desarrollada en este Trabajo Fin de Grado (PyBo-Kids-2.0).
- Los contenidos educativos tendrán serán una guía de programación y de robótica, tendrán unos objetivos a corto plazo (por ejercicio), a medio plazo (por bloque de lenguaje) y a largo plazo (por curso). Sin embargo, por edad de los alumnos, podría darse el caso de no poder completar el segundo bloque; en este caso se orientará el curso para avanzar todo lo posible.

2.3. Metodología

Este Trabajo Fin de Grado tiene como premisa un trabajo ya realizado, durante un curso escolar, de clases de robótica a alumnos de Educación Primaria. Con esta base, y este conocimiento adquirido, se crearon los objetivos y sus requisitos, para ampliar la propuesta educativa. Para cumplir estos objetivos, se ha trabajado manteniendo un ciclo semanal de reuniones telemáticas con el tutor, donde se comentaban: avances realizados con respecto a los hitos marcados la semana anterior, problemas encontrados, ideas de trabajo, y nuevos hitos para trabajar durante la semana. La progresión necesaria para el cumplimiento ha sido la siguiente:

1. Familiarización con el entorno robótico de Arduino: uso del robot Mbot con el lenguaje nativo de la placa base, para entender la comunicación *Serial* y el funcionamiento de los actuadores y sensores (valores de entrada y salida), además de familiarización con el lenguaje en sí.
2. Comunicación "robótica" básica entre Python y Arduino, a la que ir añadiendo los periféricos del robot.
3. Diseño de un sistema de mensajes estandarizados para el desarrollo de ambas bibliotecas Python y Arduino, con el que poder establecer una comunicación exitosa entre PC y robot.

4. Desarrollo de ambas bibliotecas con el diseño anterior, y de ejercicios utilizando éstas.
5. Adaptación de los ejercicios realizados durante el curso escolar a la nueva plataforma PyBo-Kids.

Capítulo 3

Infraestructura

En este capítulo describiremos la infraestructura utilizada, tanto software como hardware, detallando los pasos a seguir si se desea emular. En caso de que algún componente haya sido elegido entre otros de igual aplicación, expondremos las razones de la elección.

3.1. Entorno

Teniendo en cuenta el carácter educativo de este Trabajo, y su pretendida aplicación en estudiantes de Educación Primaria, se ha elegido el sistema operativo Windows, un entorno conocido, amigable y fácilmente accesible, para instalar y utilizar las diferentes herramientas software.

Tradicionalmente, cuando se hablaba de programación y especialmente de programación robótica, siempre se ha utilizado el sistema operativo Linux. Éste daba la posibilidad de instalar todos los paquetes de lenguajes de programación y entornos, mientras que Windows era especialmente cerrado en cuanto a lenguajes no nativos (fuera de *bash* o de *visual basic*, se hacía complicado utilizar un lenguaje de programación sin acabar recurriendo a virtualizar una máquina Linux), y los entornos software (aplicaciones) de terceros diseñados para programar habitualmente no tenían una versión instalable para Windows.

Sin embargo, los últimos años el Sistema Operativo se ha abierto a esta operativa, ya que la política popular demandaba poder utilizarlo, siendo el sistema operativo más utilizado por usuarios, como herramienta de desarrollo también a nivel usuario. La nueva línea de comandos de Windows, **Powershell** (también lenguaje de scripting), añadió a la original *Bash* características nativas de Linux, siendo una herramienta de programación además de administración. De hecho, en la última versión, está disponible para el propio SO de Linux (en algunas distribuciones).

Al principio de este Trabajo, se consideró a Linux un entorno menos amigable para usuarios sin experiencia en programación, y de la edad comentada anteriormente, por ser considerado "no de usuario": en el poco probable caso de que un estudiante conociera el sistema, lo consideraba algo muy complejo y no accesible para su nivel. Por tanto, elegir Windows eliminaba ese prejuicio y contaba con la ventaja de predisponer positivamente al alumno y de facilitarle el acceso al entorno.

3.2. Hardware

En esta sección describiremos los componentes hardware utilizados. La razón de la elección de éstos responde a la misma filosofía que en la sección anterior, la facilidad de acceso a los componentes y la facilidad con que se complementa en el entorno.

3.2.1. Placa Arduino

Las placas Arduino son las más extendidas en cuanto a robótica. Arduino nació como una solución barata con el principal objetivo de utilizarlo en Educación. Además, al ser un proyecto liberado al público, su uso está extendido a toda una comunidad, que amplía y comparte sus propios desarrollos.

Estas placas son hardware libre (uno de las principales razones de su bajo coste), y contienen un procesador re-programable y una serie de pines hembra, donde se conectarán los periféricos de entrada/salida necesarios para controlar un robot. Hay diferentes modelos de placas Arduinos, cada una fabricada con un propósito diferente; en este caso, hemos utilizado el modelo mCore, basado en [Arduino Uno](#), ya que es la que lleva por defecto el robot educativo Mbot (que comentaremos en la sección 3.2.2). Además de los pines hembra, o puertos, contiene una serie de actuadores y sensores integrados en la placa.

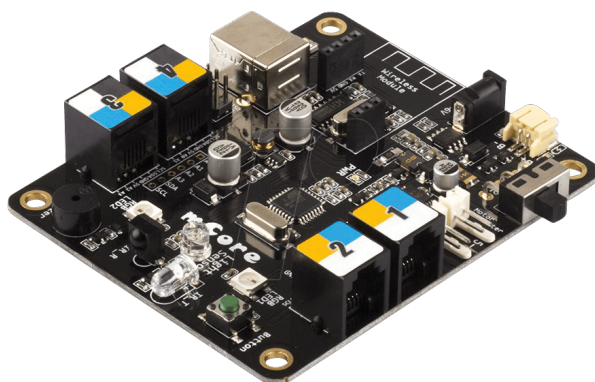


Figura 3.1: Placa mCore

3.2.2. MBot

Actualmente hay una gran variedad de robots educativos, orientados principalmente a los alumnos más jóvenes. El objetivo de la robótica educativa es ofrecer un entorno amigable y divertido, además de *más realista* que la programación tradicional. Es más fácil para un niño o niña sentirse interesado por algo que tiene una reacción *visible*, en algo que puede tocar y manejar, que en la programación normal que, aunque produce una reacción, es mucho más abstracta. Esto convierte la robótica en la herramienta perfecta para enseñar lenguajes, conceptos básicos de programación como funciones, uso de variables, y avanzar hasta la algoritmia se hace un proceso natural al que los propios alumnos llegan ellos mismos.

El principal problema al que nos enfrentamos para enseñar programación a los más jóvenes son los lenguajes de programación. Son poco intuitivos y legibles, sumado a la dificultad de mantener la atención de un alumno o alumna tan joven escribiendo en un ordenador en estos lenguajes. Era necesaria, por tanto, una forma de enseñar estos conceptos de programación

comentados anteriormente sin tener que utilizar lenguajes clásicos de programación, al menos para los alumnos más jóvenes. La solución fue la programación por bloques de *pseudocódigo*. El pseudocódigo es un *framework*, una carcasa, que recubre el verdadero lenguaje de programación de la placa del robot, y lo hace legible y entendible. Además, siendo esto una de las grandes ventajas añadidas, la mayoría de los pseudocódigos están en casi todos los idiomas, permitiendo a los alumnos programar en su propio idioma.

Esto nos lleva al robot Mbot. Está basado en una placa Arduino Uno, y pensado especialmente para la enseñanza: los diferentes sensores y actuadores (los que vienen en el paquete básico, aunque hay muchos más posibles del mismo fabricante) están pensados para ofrecer prácticas entretenidas y, lo más importante, con diferentes niveles de dificultad, por lo que se puede utilizar con alumnos de diferentes edades y, durante un mismo curso escolar, crear prácticas con las que evolucionar en los conceptos.



Figura 3.2: Modelo Mbot utilizado

Este robot Mbot se programa utilizando un lenguaje llamado *Scratch* (3.3.1), un pseudo código muy gráfico y sencillo pero potente, que "recubre" la placa base. Sin embargo, al ser una placa Arduino y por lo tanto, reprogramable, siempre se puede programar en el propio lenguaje Arduino. Esto se comentará en profundidad más adelante, en el punto 4.

Como se puede observar en la imagen, el robot cuenta con dos motores conectados a la placa; contiene además cuatro puertos a los que poder conectar diferentes periféricos con los que trabajar (numerados del uno al cuatro), sin contar con los motores. El modelo mostrado es el básico, sin embargo se pueden cambiar los componentes, añadiendo y/o cambiando la estructura base, y crear "otro" robot (por ejemplo, en las figuras siguientes ??). Aunque en este Trabajo de Fin de Grado solo trabajaremos con la versión básica del Mbot, esta posibilidad de añadirle componentes o cambiarlos es muy interesante para los alumnos, ya que trabajan la mecánica y pueden utilizar más tipos de periféricos.



Figura 3.3: Posibles cambios en el Mbot

3.2.2.1. Sensores

En robótica un *sensor* es un periférico de entrada que, conectado a una placa base, recoge información del medio (cantidad de ruido, temperatura, distancia frontal, etc) y la envía a la placa, dejándola disponible para toma de decisiones. La cantidad de información que se pueda recibir sólo depende de la cantidad de sensores que se pueda conectar a la vez a la placa (cuatro, en este caso, si solo se trabajara con sensores y ningún actuador). En el robot básico (de la imagen ??) los sensores son:

Sensor de ultrasonidos Está colocado en el frente, y recoge información de **distancia** hasta un objeto, en *cm* (el valor máximo es 400)

Sensor infrarrojo Sigue Líneas Está ubicado de tal forma que lea, del "suelo", si el sensor está tapado o no, es decir: está sobre blanco o negro (de fondo, es un sensor binario). Se compone en realidad de dos sensores, izquierdo y derecho, por lo que habrá cuatro posibilidades, cada una codificada con un valor numérico (el valor que devuelve el sensor a la placa Arduino):

- Ningún sensor tapado: 0
- Sensor izquierdo tapado y derecho no: 1
- Sensor derecho tapado e izquierdo no: 2
- Los dos sensores tapados: 3

Sensor de Luz En este caso, está integrado en la placa, por lo que no es necesario utilizar un puerto para él. Nos da un valor de cantidad de luz en el ambiente, pudiéndolo utilizar para saber si hay más o menos luminosidad de la deseada. Por supuesto, para este valor "deseado" será necesario obtener un valor inicial de la habitación en la que nos encontremos, para poder establecer ese valor barrera (*threshold*)

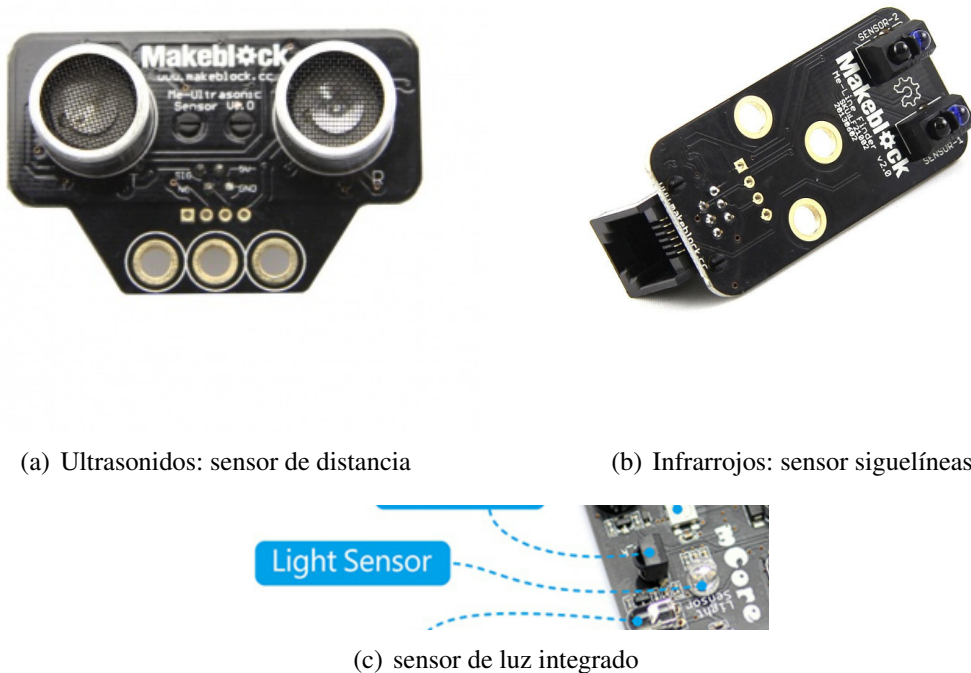


Figura 3.4: Sensores

3.2.2.2. Actuadores

La definición de *actuador* es un periférico de salida al que la placa envía datos con los que éste realiza una acción de una forma u otra. Por ejemplo, teniendo una velocidad v_0 , este valor es enviado a los motores, que se moverán a esa velocidad y no a otra.

Los actuadores, los que no están integrados en la placa directamente, se conectan a la placa a través de los mismos puertos que los sensores. Para poder usarlos, habrá que especificarle a la placa en qué puerto están conectados (igual que con los sensores). En nuestro robot básico, tenemos los siguientes actuadores:

Leds . Están integrados en la placa, en la parte superior, y compuesto por dos led individuales que poder combinar dependiendo de qué valor codificado se envíe a la placa:

- Los dos led: 0
- Sólo led derecho: 1
- Sólo led izquierdo: 2

Los led son RGB (*[red, green, blue]*), y el color de los dos led se codifica con un valor entre 0 y 255 para cada uno de los colores rojo, verde y azul. Así, por ejemplo, el rojo completo sería [0,255,0], el morado sería [255,0,255], el negro [0,0,0] o el blanco [255,255,255]. Estos led están codificados con valores decimales, en vez de hexadecimales como sería una codificación RGB tradicional, para facilitar la programación a los alumnos, que no conocerían el sistema hexadecimal.

americana. La equivalencia, que los alumnos necesitan conocer, está en la tabla siguiente. Para utilizar diferentes notas más agudas o más graves, se utilizan números a continuación de las letras (**C0** es más grave que **C5**). Internamente, el zumbador entiende valores enteros, correspondientes en frecuencia con cada nota (en la tabla, sólo se pone el valor para los valores *1*, a modo de ejemplo):

Europea	Americana	Valor entero
Do	C	33
RE	D	37
MI	E	41
FA	F	44
SOL	G	49
LA	A	55
SI	B	62



Figura 3.7: Zumbador integrado

3.3. Software

En esta sección describiremos el diferente software utilizado y el propósito de éste en el marco de este Trabajo Fin de Grado.

3.3.1. Scratch y mBlock

Como se ha comentado anteriormente, el robot Mbot es programable con Scratch, un lenguaje de **programación por bloques**. Un *bloque de código* consiste en codificar en un "paquete" una sentencia completa de lenguaje (del lenguaje correspondiente, Arduino en este caso). Así, el estudiante que utilice Scratch, será capaz de utilizar una sentencia *if..else* o un bucle *for* de forma muy fácil y entendible, sin necesitar aprenderse todas las reglas de sintaxis del lenguaje real. A continuación, se muestran algunos ejemplos de bloques en Scratch correspondientes a los conceptos de programación más utilizados:

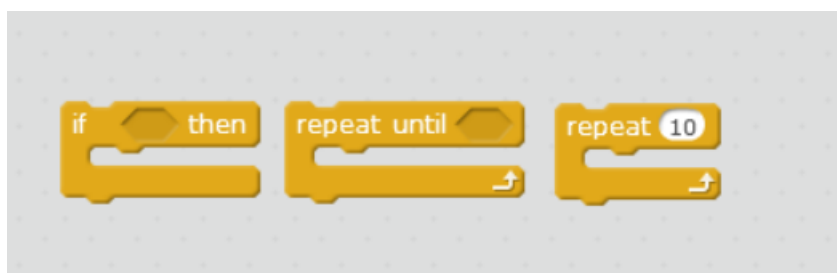


Figura 3.8: Algunos ejemplos de bloques en Scratch

Como puede observarse, las sentencias que se quieran repetir, o las variables de condición, tienen un lugar muy intuitivo donde colocarse. Además, pueden crearse variables, en las que guardar los valores de los sensores y poder utilizar como entrada de actuadores, o como valores de referencia.

Ciertamente, y aunque el lenguaje Scratch puede utilizarse como lenguaje de programación "tradicional", utilizando un escenario virtual con un personaje para observar el resultado del programa (sin robot físico), es mucho más completo al añadirle el módulo del robot deseado. Este módulo contiene bloques para recoger valores de los sensores o enviar valores a los actuadores, ya sean "on board" (integrados en la placa), o teniendo que especificar el puerto al que están conectados:

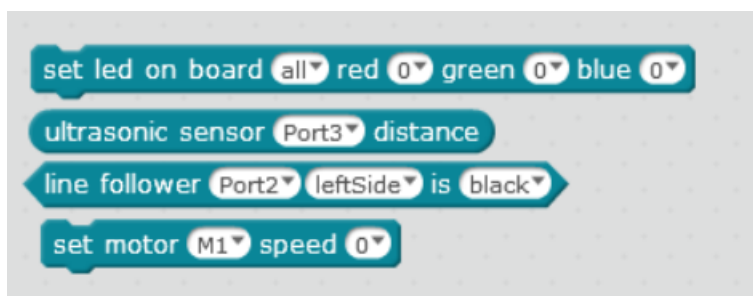


Figura 3.9: Ejemplos de bloques de Mbot en Scratch

Para utilizar Scratch y programar el robot, es necesario instalar un programa, que es el que contiene el compilador y el que permite conectar el robot al PC para enviarle el programa, llamado *mBlock*. El ejecutable se descarga de la página oficial del fabricante, [Makeblock](https://makeblock.com/), disponible para PC (también existe una versión simplificada para dispositivos móviles).

Una vez instalado el software, el proceso para poder empezar a usar el mBot es muy simple:

1. Conectar el robot al pc con el cable USB, y conectarlo con el programa (el robot debe estar encendido).

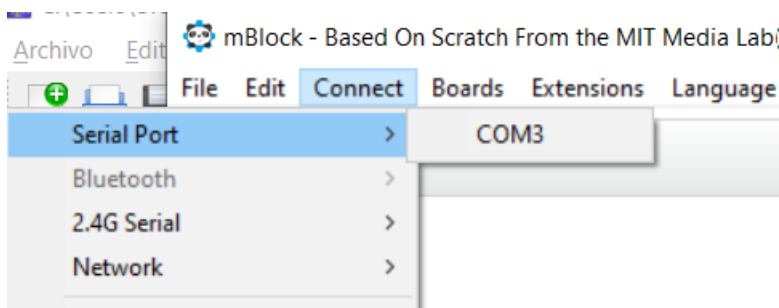


Figura 3.10: Conectar el mBot

2. Asegurarse de que la placa corresponde con el robot que tenemos

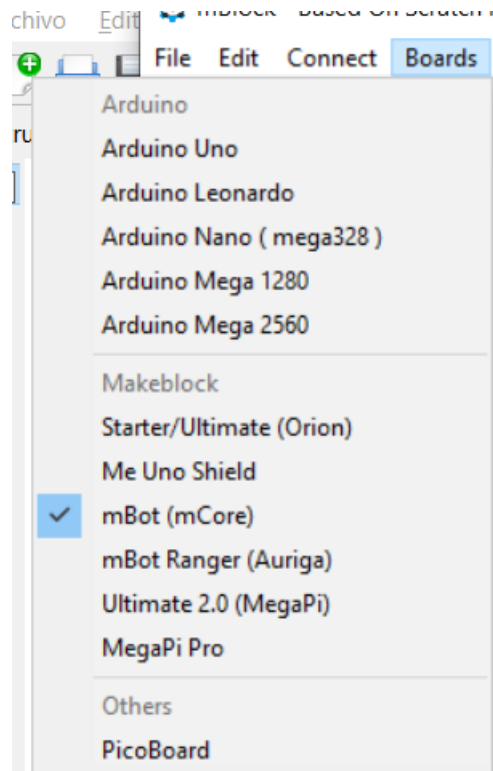


Figura 3.11: Placa mBot

3. Actualizar el programa Arduino subido a la placa base, para que funcione con Scratch (en caso de haber utilizado el robot con otro programa)

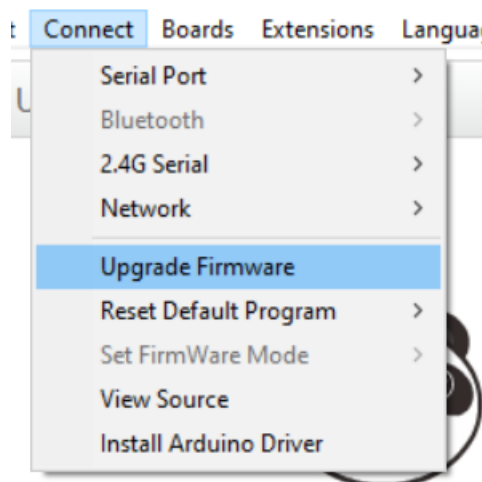


Figura 3.12: Actualizar firmware de la placa

4. Una vez programado algo de código, para subirlo al robot: click derecho sobre el código, 'upload to arduino', para compilar el programa, y otra vez a 'upload to arduino' cuando el compilador aparezca en la parte derecha de la pantalla, para subirlo a la placa

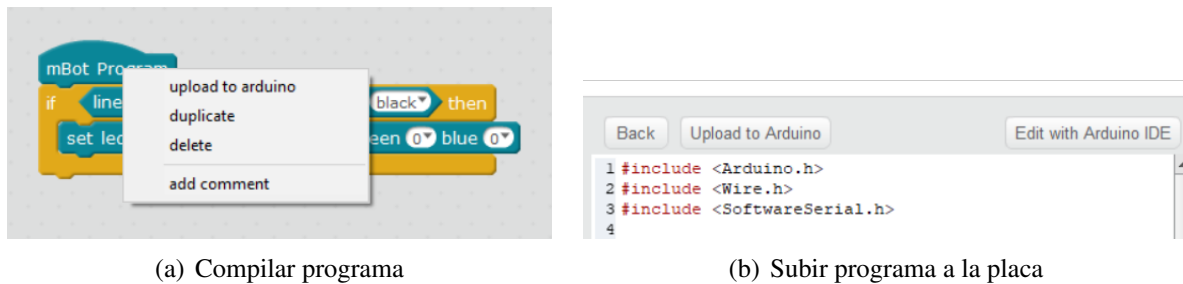


Figura 3.13: Subir programa a la placa del mBot

5. Una vez subido el programa, el robot comenzará a funcionar tal y como lo hayamos programado.

Este lenguaje, junto a la interfaz gráfica, es una perfecta primera aproximación a la programación y a la robótica, dando más importancia al aprendizaje conceptual que a la sintaxis y al lenguaje. Obviamente, los alumnos necesitarán que se les guíe, principalmente al comienzo del curso, en comprender las necesidades de incluir los diferentes bloques y las ventajas que produce en el código. El objetivo no será explicarles para que usar, por ejemplo, un bloque condicional, sino proponerles un ejercicio en el que, para llegar a la solución, necesitarán de forma intuitiva programar un condicional, y lleguen naturalmente a la necesidad de ello.

3.3.2. Arduino

Como se explicó cuando hablamos de la placa base en la sección 3.2.1, ésta se programa en **Arduino**. Este lenguaje de programación está basado en C++, y pensado para interactuar con objetos electrónicos.

Teniendo en cuenta el objetivo de este Trabajo Fin de Grado, ofrecer una posibilidad de programar el robot Mbot en Python, es necesario utilizar el lenguaje nativo de la placa base, con el fin de que el robot *entienda* las órdenes programadas en Python. Por lo tanto, para crear este "framework", es necesario instalar el entorno Arduino. A continuación se explican las instrucciones para instalar el entorno en Windows, y para configurarlo para el robot.

1. Descargar e instalar, Arduino IDE, el cual instala el entorno completo de Arduino. Para las versiones 8 y 10 de Windows, está disponible directamente en el Microsoft Store. Para otras versiones de Windows, está disponible en la [página web oficial](#).



Figura 3.14: Arduino IDE en el Microsoft Store

2. Descargar las librerías de [Makeblock](#) para añadirlas a Arduino y poder utilizarlas en nuestro entorno.

3. Incluir el archivo comprimido descargado desde el Arduino IDE:
Programa - Incluir librería - Añadir biblioteca .ZIP - Seleccionar el fichero - Abrir
4. Seleccionar la placa básica que se va a conectar al IDE:
Herramientas - Placa - Seleccionar Arduino Uno
5. Especificar el modelo exacto de placa al principio del programa de Arduino; en este caso la placa que estábamos utilizando es la mCore. Así se cargan todos los métodos correspondientes al modelo de placa; debe incluirse en cada programa de Arduino que se escriba.

```
#include "MeMCore.h"
```

6. Igual que con Scratch, primero se debe conectar el robot al software (con el robot enchufado al PC y encendido):
Herramientas - Puerto - Seleccionar el puerto en el que está el robot
Los puertos USB en Windows son *COM1*, *COM2*, *COM3*, etc.
7. Una vez tengamos un programa en Arduino listo para subir a la placa y probar con el robot:
Programa - Subir o el botón rápido de "subir".
Este proceso primero compilar el programa y, si está correcto, lo sube a la placa. Sin embargo, se puede compilar primero como comprobación (*Programa - Verificar*).

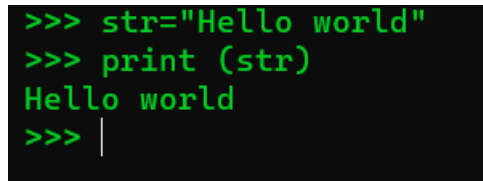
3.3.3. Python

Por último, utilizaremos **python** para programar la lógica de los ejercicios. El objetivo es que se programe en python sólo los ejercicios, habiendo paquetizado todo lo relativo al robot en Arduino, y sólo utilizando la conexión con éste para obtener datos de los sensores y enviar datos a los actuadores. La descripción de este proceso y de la lógica de ambos lados (PC y robot) se describirá en profundidad en el capítulo 4.

La razón de utilizar Python como lenguaje de programación es la misma que la del resto de componentes: la facilidad para los alumnos de acceder a ello. Es uno de los lenguajes con sintaxis más simple (por comparación, Arduino / C++ es muy cerrado y complejo), además de ser las palabras reservadas significativamente entendibles (aunque es cierto que en inglés: *while*, *print*, *read*, etc), así como la declaración de variables es más flexible. Además, y aparte de la cuestión educacional, el módulo *serial* para conexión con periféricos, está con la instalación base, y es mucho más sencillo que en otros lenguajes, haciéndolo perfecto para la electrónica.

La instalación de python en Windows también es muy sencilla:

1. Descargar el paquete de la [web oficial](#) y ejecutar el instalable descargado. También es posible, para Windows 10, obtenerlo desde el Microsoft Store, tal y como se hizo para Arduino.
2. Una vez descargado, comprobar que se puede ejecutar abriendo una consola -Powershell- y escribiendo *python*, se ejecutará la consola de python, y podremos probar que tenemos python instalado en nuestro entorno (*exit()* para salir). También es útil comprobar que hemos instalado la versión correcta (3.10): *python --version* en una consola de Powershell (no teniendo abierta la consola de python)

A screenshot of a Python shell window with a black background and green text. The text shows a sequence of commands and their output: first, a string is assigned to a variable, then the variable is printed, resulting in the output 'Hello world', and finally, a new prompt line is shown with a vertical cursor.

```
>>> str="Hello world"
>>> print (str)
Hello world
>>> |
```

Figura 3.15: Comprobar python

3. Para ejecutar un programa escrito en python desde la consola de Windows:

```
py HelloWorld.py
```

Para escribir código en python, se puede utilizar cualquier editor de texto. El mismo instalador de python instala uno propio, *IDLE shell*, suficientemente simple y preparado particularmente para su sintaxis.

Capítulo 4

Plataforma PyBo-Kids-2.0

En este capítulo explicaremos el proceso seguido para desarrollar las bibliotecas Arduino y Python, explicando el código y las distintas necesidades que han surgido durante el desarrollo. Como se explicó en el Capítulo 3, se ha utilizado Arduino y su IDE nativo para esta programación, y Python 3 y un editor de texto estándar (en este caso, Visual Studio Code, de Microsoft) para la programación en Python.

4.1. Diseño

Lo primero en el diseño de la plataforma ha sido el modelo "PC - Residente". Partiendo de la premisa dada de programación del robot en Python, era necesaria una forma para que, dado que la placa base iba a funcionar en Arduino, la comunicación Serial¹ funcionara entre los dos lenguajes.

4.1.1. Comunicaciones

La comunicación con el robot, como se ha comentado en el Capítulo 3 de Infraestructura, debe establecerse entre el entorno y el robot (con la placa base), cada vez que se encienda éste. Es, por tanto, el primer problema a solventar en ambos lenguajes.

El protocolo *Serial* abre una vía de comunicación a través de un canal electrónico, en este caso un cable USB, entre el entorno y el robot, a una velocidad en baudios². Este canal para el traspaso de información es necesario para enviar datos a la placa (por ejemplo, los colores a los que encender los LED integrados) o recibir datos de ésta (los valores de lectura de los sensores) y poder utilizarlos en toma de decisiones.

Al abrir la comunicación en ambas partes, placa base y PC, cualquiera de ellas es capaz de leer del canal la información que necesite, y de enviar a través de él (para que esto sea así, ambas partes deben haber abierto la comunicación a la misma velocidad). Por tanto, si la placa Arduino envía a través del canal Serial el dato que recoge del sensor de infrarrojos, el lado PC, que estaría leyendo de ese canal, obtendría este dato.

En la parte Arduino, al grabar el programa residente completo en la placa, la comunicación se abre a la vez que se enciende el robot, puesto que el programa arranca con él. En la parte PC, de

¹Comunicación secuencial de información a través de un canal, electrónico en este caso

²Velocidad, utilizada en electrónica, medida en número de símbolos por segundo

Python, la comunicación se abre cuando ejecutamos el programa que queremos que ejecute el robot. El flujo, entonces, podría dibujarse como el diagrama que aparece a continuación. Como

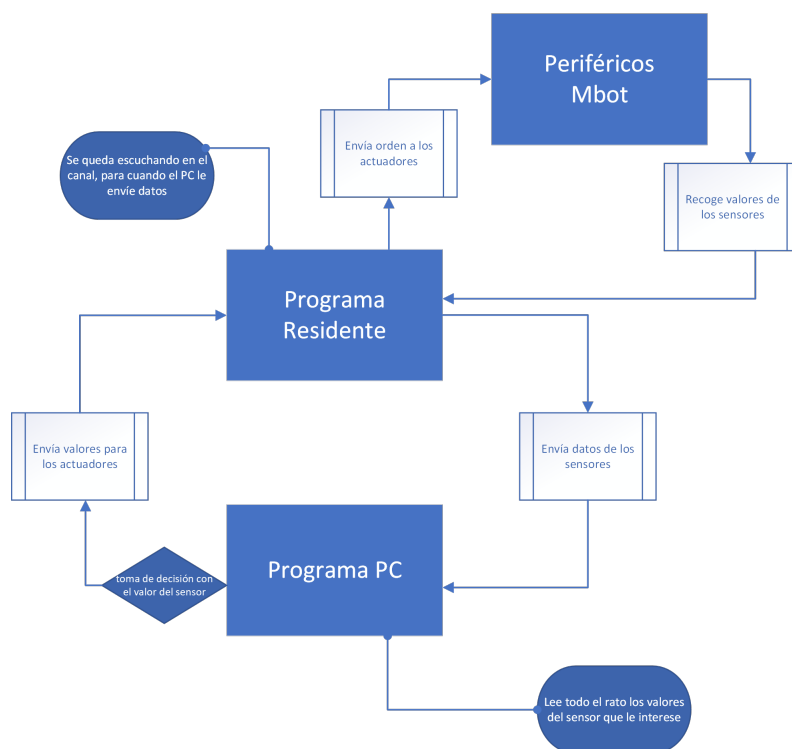


Figura 4.1: Diagrama de comunicaciones

podemos observar, ocurren varias cosas de forma paralela:

- El programa residente está recogiendo los valores de los sensores, conectados a su puerto correspondiente, y los envía por el canal.
- El programa PC recoge los valores del sensor que le interese (dependerá del programa que queramos uno u otro).
- En función del valor del sensor (con respecto a un valor umbral o *threshold*), el programa PC envía por el canal unos valores concretos para un actuador concreto. Lee otra vez el valor -nuevo- del sensor, por si tuviera que cambiar de decisión.
- El programa residente lee del canal si tiene mensajes para un actuador y, en caso afirmativo, recoge los valores y los envía al actuador correcto.

A continuación, detallaremos la forma técnica en la que se han realizado estas dos partes de la comunicación.

4.1.2. Programa residente

La biblioteca residente en Arduino se ha realizado de forma progresiva, encontrando diferentes requerimientos y necesidades a lo largo del proceso.

4.1.2.1. Comunicación serial

En Arduino, para utilizar el protocolo Serial no es necesario cargar ningún módulo añadido, al ser un lenguaje pensado para las comunicaciones electrónicas. La inicialización de la comunicación Serial debe hacerse en la función *setup* de Arduino, donde se coloca el código que debe ejecutarse sólo una vez, al comienzo de la ejecución del programa, con la velocidad en baudios deseada como parámetro.

```
void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
}
```

Listing 4.1: Inicialización del protocolo Serial

Durante el resto del código, cada vez que se quiera escribir o leer del canal, deberá llamarse al 'Serial' que hemos iniciado. Es importante comentar que, antes de llamar a una función que lea del canal, debemos asegurarnos que hay datos que poder leer, o el programa generará un error. Por ejemplo:

```
char mensaje;
void loop() {
    if (Serial.available() > 0) {
        mensaje = Serial.read();
        Serial.println(mensaje);
    }
}
```

Listing 4.2: eco en Arduino: lee del canal Serial y lo escribe

A la hora de leer del canal Serial, en este caso en Arduino pero igual con cualquier herramienta, hay que tener en cuenta la cantidad de bytes que se leen cada vez. En este caso, estábamos considerando, a modo de test, un sólo carácter (inicializando la variable como *char*)³, por lo que leemos del canal un solo byte (método *read()*). Sin embargo, si quisiéramos leer un *String*⁴, como va a ser necesario para leer los mensajes que envíe el programa PC, tendríamos que inicializar la variable como *String* y leer del canal un String entero, esto es, hasta leer el valor nulo en el que éste termina.

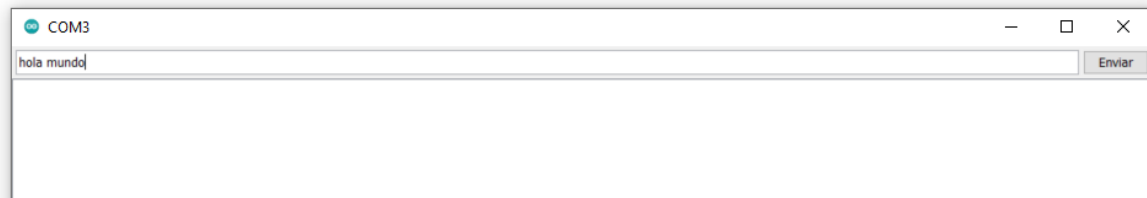
```
String mensaje;
void loop() {
    if (Serial.available() > 0) {
        mensaje = Serial.readString();
        Serial.println(mensaje);
    }
}
```

Listing 4.3: eco en Arduino con un Strin

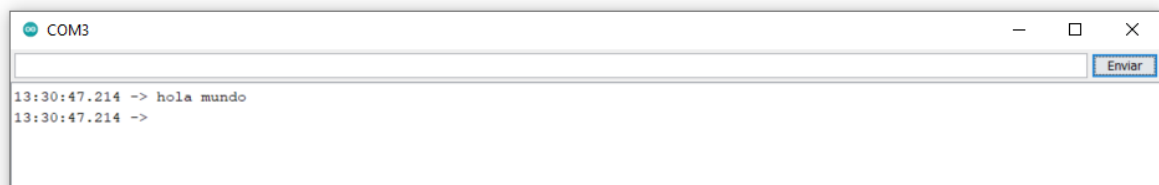
³Variable que almacena un valor de carácter, que ocupa un sólo byte de memoria, y que debe escribirse entre comillas simples: *char mensaje = 'a';*

⁴Array de caracteres, escrito entre comillas dobles, y que termina en valor nulo ('0' en código ASCII): *String mensaje = "Hello String";*

Dado que primero estamos considerando solamente el entorno Arduino, para poder ver el resultado de esta comunicación, usaremos el Monitor Serie del propio IDE de Arduino, que permite escribir por el canal (lo que luego leamos con *serial.readString()*), y muestra lo que Arduino escribe (*Serial.println()*). El código anterior produciría la siguiente ejecución:



(a) Escribir el String que Arduino lee del canal



(b) Muestra el String que Arduino ha escrito en el canal

Figura 4.2: Visualización del eco de Arduino en el Monitor Serie de Arduino IDE

4.1.2.2. Uso de sensores y actuadores

Una vez comprobadas las comunicaciones, continuamos con los actuadores y sensores del robot. Como se ha explicado en el apartado 3.3.2 de Arduino, es necesario cargar los módulos correspondientes a la placa mCore. Una vez incluido el paquete, podremos utilizar los métodos de inicialización, lectura de sensores, envío de órdenes a los actuadores, etc. Este módulo contiene toda la información que necesitan los diferentes componentes (tipos de valores de entrada o de retorno, diferentes métodos, etc), incluidos ejemplos en los que poder apoyarnos. A continuación describiremos como se utilizan los diferentes actuadores y sensores en Arduino, siendo este paso necesario para crear la biblioteca "Residente".

- **Actuadores:** Como cualquier variable, los actuadores requieren de una inicialización; esta inicialización de variables corresponden al principio del programa, para poder utilizar la variable en las funciones principales. Luego, depende de qué actuador sea, requerirá de un tipo u otro de valor de entrada:

Motores Los motores de la placa mCore deben inicializarse cada uno por separado; al darles nombres diferentes podremos enviar la orden al motor correcto (por ejemplo, para girar el robot, no debe enviarse la misma orden al motor derecho que al izquierdo, sino invertir el sentido de uno de ellos, dependiendo de en qué dirección se quiera girar).

```
MeDCMotor motorIzdo (M1);
```

```
MeDCMotor motorDcho (M2);
```

Listing 4.4: Inicializar motores Mbot

El valor de entrada para la velocidad es un valor entero (tipo *int*), entre [-255,255], siendo los valores negativos para una velocidad de retroceso. En este caso, para que los motores se paren, no se les enviaría un valor de 0 sino que tiene un método propio.

```
motorIzdo.run(100);
motorDcho.run(100);
delay(100);
motorIzdo.stop();
motorDcho.stop();
```

Listing 4.5: Uso de motores Mbot

Leds integrados En este caso, la inicialización del led requiere el puerto (integrado de la placa) y un slot (de número de leds). Por tanto:

```
const int PORT = 7;
const int SLOT = 2;
MeRGBLed led(PORT, SLOT);
```

Listing 4.6: Inicializar leds

Los valores necesarios para los leds se han descrito en la sección de Actuadores 3.2.2.2; Arduino requiere primero enviar la configuración de colores, y después mostrar esa configuración. En este caso, para apagarlos, sí se envía un valor de 0 para los tres valores RGB.

```
led.setColor(ledsInt, redInt, greenInt, blueInt);
led.show();
```

Listing 4.7: Uso de los leds

Zumbador Como sólo hay un zumbador en la placa, y está integrado en ella, no es necesario ningún puerto. Para que emita la nota deseada, es necesario un valor entero para la frecuencia y otro para la duración (en milisegundos):

```
MeBuzzer buzzer;
void loop() {
  buzzer.tone(87,3000);
  delay(100);
  // stop the tone playing:
  buzzer.noTone();
}
```

Listing 4.8: Uso del zumbador

- **Sensores:** Todos los sensores en Arduino tienen un método de lectura, además del de inicialización. Si queremos almacenar el valor recogido en una variable, Arduino requiere que ésta sea declarada también.

Sensor de ultrasonidos Al no ser un sensor integrado a la placa, es necesario que se especifique en qué puerto se ha conectado. Éste devuelve la distancia, en centímetros (como valor entero), a la que se encuentra un obstáculo.

```
MeUltrasonicSensor ultraSensor(PORT_3);
void loop() {
    int DistanceValue = ultraSensor.distanceCm();
    Serial.println(DistanceValue);
}
```

Listing 4.9: Sensor de distancia

Sensor de luz El puerto especificado en la llamada al método, aunque integrado en la placa, especifica el pin interno al que está conectado. Devuelve un valor entero, de cantidad de luz. En este caso, para poder utilizar un valor de *threshold* necesitaremos saber qué valor de luminosidad aproximada tiene la habitación (no tiene por qué ser siempre la misma)

```
MeLightSensor lightSensor(PORT_6);
void loop() {
    int LigthValue = lightSensor.read();
    Serial.println(LigthValue);
}
```

Listing 4.10: Sensor de luz

Sensor infrarrojo Requiere también especificar a qué puerto se le ha conectado, devolviendo de la llamada de lectura un valor entero correspondiente a qué combinación de sensores sigue líneas están tapados o no (explicados en la sección de Actuadores (3.2.2.2))

```
MeLineFollower SigueLineas(PORT_1);
void loop() {
    int LineFollowerValue = SigueLineas.readSensors();
    Serial.println(LineFollowerValue);
}
```

Listing 4.11: Sensor siguelíneas

Una vez conocido el funcionamiento de los componentes del mBot, tenemos la capacidad de abstraer este conocimiento y hacer con ello las funciones que compondrán esta biblioteca Arduino, para estructurar el programa residente de forma que funcione independientemente de qué datos se le envíen desde el programa PC. Esta estructura de biblioteca se explicará en la sección 4.2, una vez conocida también la parte de programación en Python.

4.1.3. Programa PC

La finalidad de la "Parte PC" es construir una biblioteca de funciones que contengan la lógica de conexión, lectura, escritura, etc, y que la escondan a los alumnos, teniendo ellos que preocuparse solamente de llamar a una función con un nombre amigable. A continuación describiremos los puntos importantes que se han necesitado en la preparación de esta biblioteca:

Protocolo Serial En este caso, es necesario incluir el módulo Serial al principio del programa de Python. Para iniciar una comunicación Serial, al igual que con Arduino, es necesaria la velocidad en baudios a la que conectarse (como dijimos, debe ser la misma a la que se ha abierto la comunicación en la parte de Arduino); además es necesario el puerto al que está conectado el robot (de forma parecida a la que se especificaba en el Arduino IDE) y un tiempo de *timeout* para el que, si no se ha establecido la comunicación, se eleva una excepción (que se deberá recoger con un bloque de *try..catch*)

```
import serial
serial = serial.Serial('com3', 9600, timeout=1)
serial.close()
```

Para leer del canal, deberemos tener en cuenta igualmente la cantidad de información que queramos leer. En general, como estaremos leyendo Strings de mensajes, python contiene (como Arduino) la función de lectura de una línea completa (hasta fin de línea).

```
serial.readline() #leer hasta EOL
```

Sin embargo, se puede leer un solo byte (como en el caso de leer un solo carácter), o una cantidad de bytes especificada.

Leer del canal Dado que el String leído con el valor de un sensor, lo ha enviado Arduino y ha atravesado el canal, es necesario decodificarlo para obtener un String sin los caracteres de retorno de carro, end of line, etc. Si no, no podríamos utilizar como número ese valor, ya que al intentar convertirlo desde String, no debe tener ningún otro carácter.

```
Data = serial.readline()
decoded = Data.decode()
sensorValue = float(decoded)
```

Escribir en el canal Al igual que para leer, para escribir en el canal y asegurarnos que en el programa residente le llegan datos que sea capaz de interpretar, el envío de datos (texto) debe hacerse forzando la codificación en UTF-8.

```
Data = "Hello _World"
serial.write(bytes(Data, 'utf-8'))
```

Estas decisiones de codificaciones, tanto para escribir como para leer del canal, son el resultado de pruebas entre uno y otro lado (Arduino - Python), con varios tipos de datos y de formas de leer, con la finalidad de asegurar que ambos lados pueden establecer una comunicación con éxito y que las opciones necesarias están recogidas.

4.2. Resultado: programa PC y programa residente

En esta sección explicaremos cómo se juntan los dos programas, Residente y PC, y cómo se utilizan las dos bibliotecas.

Como se ha adelantado anteriormente, para que la comunicación sea posible es necesario un protocolo de mensajes; es necesario asegurarse que ambas partes recojan la información correcta y sepan qué deben hacer con ella. Poniendo un ejemplo: el Programa Residente debe estar

preparado para recibir datos que enviar a los actuadores. Sin embargo, cada actuador requiere datos de entrada diferentes, por tanto, debe estar preparado también para saber para qué actuador le están enviando los datos. Igualmente, el Programa PC debe estar preparado para enviar la información de forma que sea inequívoca.

El mismo caso se da para los sensores. El Residente tiene que enviar el dato de forma que el PC pueda saber que el dato que lee es el que necesita (no vale para lo mismo si lee el sensor de luz que el del Sigue Líneas)

4.2.1. Protocolo de mensajes

Este sistema de codificación de los mensajes, funciona de la siguiente forma:

■ Sensores

- Dado que la forma más simple, efectiva en esa simpleza, de enviar datos es un string, será así como se enviará la información. Para ello, cuando se lea el dato del sensor, habrá que convertir el valor entero en un String.
- A cada sensor se le asignará un número, único (un identificador), que le representará sólo a él. Así, al sensor de ultrasonidos le corresponderá un 0, al Sigue Líneas un 1, etc.
- El Programa Residente, enviará la información en un único mensaje, como String, concatenando el identificador de sensor con el valor de este sensor recogido del robot, separando los dos valores con punto y coma (para diferenciarlo de una posible coma decimal).
- El Programa PC, cuando está leyendo, separará el mensaje por ';' y , dependiendo del primer *substring*, devolverá al programa principal el tipo de sensor en texto, para que sea amigable para un alumno.

■ Actuadores

- De forma análoga, a cada actuador se le asignará un identificador numérico.
- El Programa PC concatenará, en un mismo mensaje, el identificador del actuador, y los datos necesarios para ese actuador (en caso de los LED, por ejemplo, el valor qué leds encender y los tres valores RGB). Igualmente, todos estos valores irán separados entre ';'.
- Al leer del canal, el Residente separará el primer valor por el ';' y dependiendo de qué identificador sea, leerá una cantidad de valores u otra, y enviará esos valores a un actuador u otro (convirtiéndolo primero a valor numérico).

Como ejemplo, el siguiente código correspondería al envío de información para encender los led:

```
mensaje = "0;2;255;0;0"
send_Message(mensaje, serial)
```

Listing 4.12: Envío de un mensaje desde el Programa PC

Y el siguiente código, a la lectura de ese mensaje en el lado Residente:

```
void loop () {
  String mensaje = Serial.readString();
  int indexActuator = mensaje.indexOf(';');
  String Actuator = mensaje.substring(0,indexActuator);
  String mensajeActuador = mensaje.substring(indexActuator+1);
  if (Actuator == "0") {
    int IndexLeds = mensaje.indexOf(';');
    String leds = mensaje.substring(0, IndexLeds);
    int IndexRed = mensaje.indexOf('; ', IndexLeds+1);
    String red = mensaje.substring(IndexLeds+1, IndexRed+1);
    int IndexGreen = mensaje.indexOf('; ', IndexRed+1);
    String green = mensaje.substring(IndexRed+1, IndexGreen+1);
    String blue = mensaje.substring(IndexGreen+1,-1);
    int ledsInt = leds.toInt();
    int redInt = red.toInt();
    int greenInt = green.toInt();
    int blueInt = blue.toInt();
    led.setColor(ledsInt , redInt , greenInt , blueInt);
    led.show();
  }
}
```

Listing 4.13: Lectura del mensaje en el lado Arduino

Como ventaja añadida, con este sistema de mensajes la funcionalidad de la plataforma PyBo-Kids2.0 es fácilmente ampliable a más sensores o actuadores, puesto que los identificadores son números enteros.

4.2.2. Bibliotecas

Obviamente, esta codificación de los mensajes, junto con toda la funcionalidad correspondiente al uso del canal Serial, es la que estará "escondida" para que los alumnos utilicen simplemente funciones con nombres autoexplicativos, tales como 'turnOnLeds' o 'readSensor'. Toda la lógica de funciones, de parseo de mensajes, de lectura o escritura en el canal, está integrada en la biblioteca, que deberá cargarse como cualquier otra en python.

Así, si utilizamos como ejemplo el mismo que en 4.12, la funcionalidad en la biblioteca sería la siguiente:

```
def send_Message (message , serial):
  serial.write(bytes(message , 'utf-8'))
def create_Message_Led (list):
  mensaje = f"0;{list[0]};{list[1]};{list[2]};{list[3]}"
  return mensaje
def turnOn_Leds(list , serial):
  mensaje = create_Message_Led(list)
  send_Message(mensaje , serial)
```

Mientras que, en el programa principal, si un alumnos quisiera encender los dos led a rojo:

```

from library_Mbot_v1 import *
import sys
import serial
turnOn_Leds([0,255,0,0], serial)

```

En la parte de Arduino, por otro lado, tenemos toda la biblioteca en un solo archivo, que será el que carguemos en la placa base del robot. Con la finalidad de que pueda ser fácilmente ampliada, abstraeremos todo lo posible en funciones. El ejemplo anterior, quedaría entonces:

```

void read_LedsMessage (String mensaje) {
    int IndexLeds = mensaje.indexOf(';');
    String leds = mensaje.substring(0, IndexLeds);
    int IndexRed = mensaje.indexOf(';', IndexLeds+1);
    String red = mensaje.substring(IndexLeds+1, IndexRed+1);
    int IndexGreen = mensaje.indexOf(';', IndexRed+1);
    String green = mensaje.substring(IndexRed+1, IndexGreen+1);
    String blue = mensaje.substring(IndexGreen+1, -1);
    int ledsInt = leds.toInt();
    int redInt = red.toInt();
    int greenInt = green.toInt();
    int blueInt = blue.toInt();
    led.setColor(ledsInt, redInt, greenInt, blueInt);
    led.show();
}

void loop() {
if (Serial.available()>0){
    String mensaje = Serial.readString();
    int indexActuator = mensaje.indexOf(';');
    String Actuator = mensaje.substring(0, indexActuator);
    String mensajeActuador = mensaje.substring(indexActuator+1);
    if (Actuator == "0") {
        read_LedsMessage(mensajeActuador);
    } else if (Actuator == "1") {
        read_MotorsMessage(mensajeActuador);
    } else if (Actuator == "2") {
        read_BuzzerMessage(mensajeActuador);
    }
}
}

```

Tenemos, por tanto, una abstracción de la sintaxis y la complejidad de las herramientas usadas, mientras que seguimos pudiendo utilizar un lenguaje de programación real, como es Python, en vez de Scratch, para ampliar los conocimientos que se habrían iniciado con éste último. Tendremos la oportunidad de ampliar conocimientos con conceptos como listas, diferentes módulos, entrada y salida estándar, etc.

Capítulo 5

Aplicación educativa

5.1. Contexto

5.2. Primera etapa: Scratch

5.2.1. Objetivos docentes: metodología

5.2.2. Prácticas

5.3. Segunda etapa: PyBo-Kids en Python

5.3.1. Objetivos docentes: metodología

5.3.2. Introducción a Python

5.3.3. Prácticas

Esto es un ejemplo. Ver en [1] [2]

Capítulo 6

Conclusiones

Esto es un ejemplo. Ver en [1] [2]

Bibliografía

- [1] DARWIN, C. *El origen de las especies por medio de la selección natural*. Editorial CSIC-CSIC Press, 2009.
- [2] DEL VALLE-INCLÁN, R. *Luces de bohemia*. Dirección General de Música y Teatro, 1984.