Capítulo 4

Integracion Mbot en Kibotics

En este capítulo se aborda el proceso de integración que va a permitir programar al robot físico Mbot sin necesidad de una instalación previa ni descargas adicionales.

4.1. Características del Mbot

Mbot es un robot fácil de montar y con una estructura robusta, orientado para empezar el aprendizaje de la robótica y la programación desde la educación primaria. Está diseñado por la empresa MakeBlock, la cual dispone de una gran variedad de recursos, robots y kits de robotica. El robot pesa 400 gramos y sus dimensiones son de 17x13x9 cm. Respecto a las especificaciones técnicas, posee una placa llamada "mCore" que está basada en ArduinoUno, dispone de una microcontroladora ATmega238, cuatro puertos Rj25, un interruptor de encendido, dos leds RGB, un botón, zumbador, un sensor de infrarrojos y otro de luminosidad. Dispone de una batería de litio de 3,7V, pero también funciona con cuatro pilas de tipo AA. En cuanto sus módulos externos, presenta dos motes, ultrasonido y un seguidor en línea, aunque también pueden utilizarse otras conexiones adicionales. Se comunica por vía puerto serie o por Bluetooth 4.0 (Robótica Educativa con Mbot, 2020).

4.2. Proceso de envío desde la plataforma Kibotics

Desde la parrilla principal de Kibotics se puede entrar a la unidad dedicada al Mbot, pudiendo elegir qué lenguaje de programación usar (Python o Blockly). Dentro de dicha unidad encontramos un apartado dedicado a proporcionar las instrucciones necesarias para realizar un programa y enviárselo al Mbot y otro donde se proporciona el editor que permite escribir el programa.

Una vez que el usuario ha realizado el programa deseado y ha conectado el Mbot por USB a su ordenador, debe pulsar el botón azul que se muestra en la Figura 4.2 y, tras ello, se iniciará el proceso de envío.

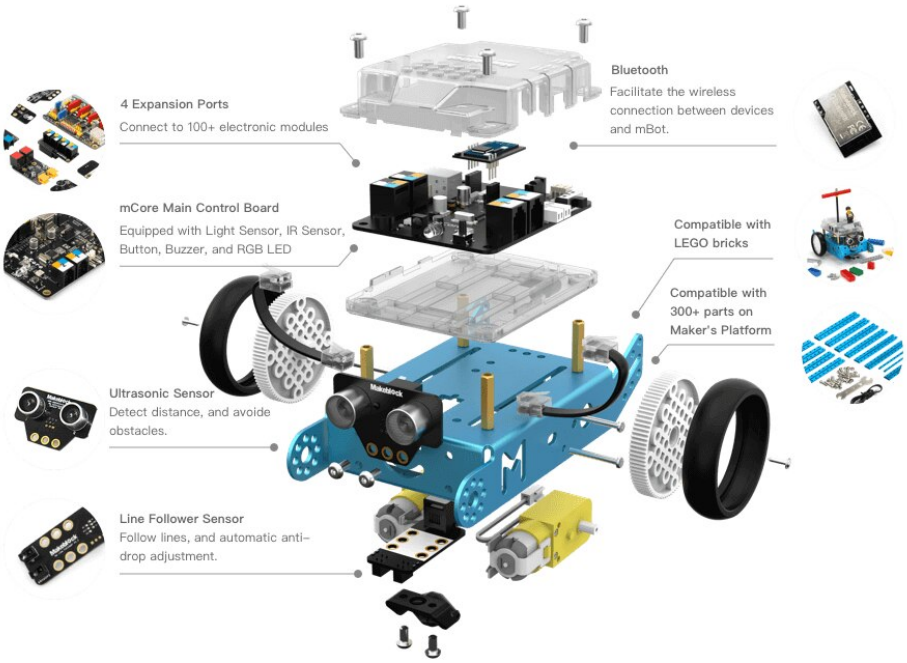

Figura 4.1: Partes robot Mbot

En primer lugar, se abrirá una ventana que muestra los puertos del ordenador disponibles (Figura 4.3). Se debe seleccionar en el que se encuentra conectado el robot, que aparece nombrado como USB2.0Serial.

Tras seleccionar el puerto y pulsar "conectar", el programa se cargará al Mbot. En el caso de haber cometido algún error durante el proceso, aparece un banner dando un aviso.

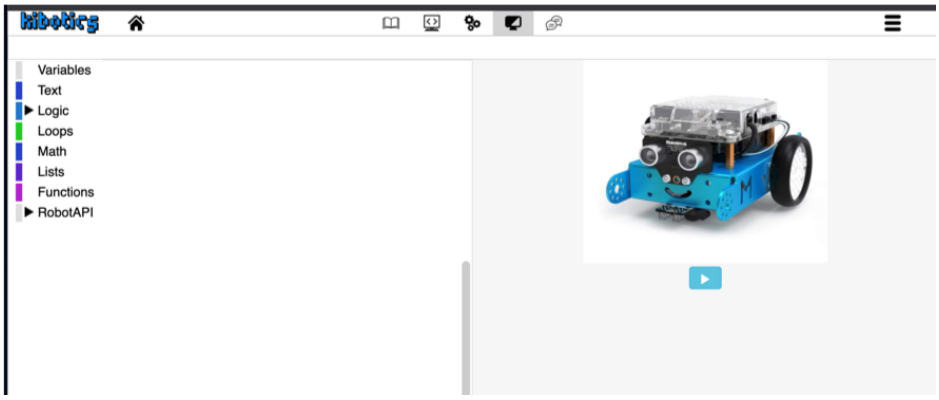

Figura 4.2: Editor Blockly para el Mbot

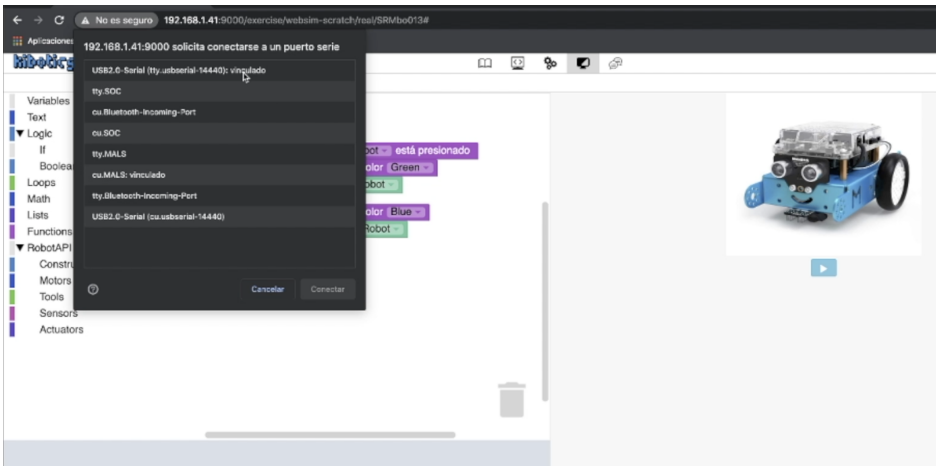

Figura 4.3: Selección puerto donde esta conectado el Mbot

4.3. Desarrollo realizado para la integración

Este proceso descrito en la sección anterior es el que el usuario debería seguir para programar su robot, esto el como funciona a alto nivel, es decir, a ojos del usuario, pero por debajo se llevan a cabo una serie de acciones que permiten toda esta comunicación.

En la Figura 4.4 se muestra la infraestructura y las interacciones necesarias para conseguir cargar el programa al Mbot

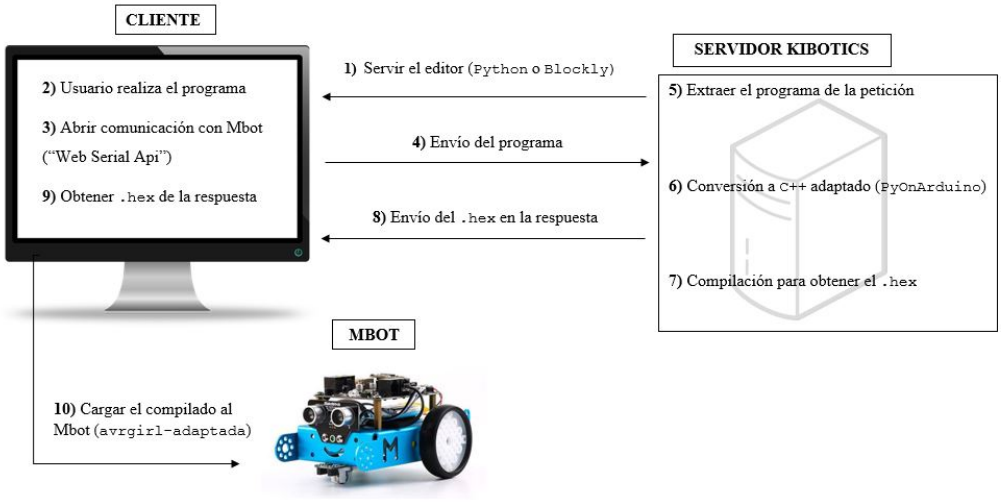

Figura 4.4: Infraestructura seguida para la integración del Mbot

En los siguientes apartados se describen las fases principales que se deben seguir para enviar el programa al Mbot. Todo este proceso se inicia cuando el usuario ha realizado el ejercicio y ha pulsado "enviar".

4.3.1. Abrir una comunicación por el puerto serie

El primer paso consiste en abrir una comunicación entre el navegador y el puerto serie donde está conectado el robot. Esta comunicación es posible gracias a "Web Serial Api", que proporciona una Api al navegador; actualmente está disponible para el navegador Chrome. Una vez que el usuario ha pulsado "enviar", se procede a abrir la comunicación. Se abrirá el panel mencionado en el capítulo anterior (Figura 4.3) donde el usuario podrá conectarse al puerto donde se encuentra el Mbot y así poder iniciar la comunicación.

```
    document.getElementById('uploadMbot').addEventListener('click',
        clickConnect);

});

async function clickConnect() {
    progress.style.width = '0%';
    alertError.style.visibility = 'hidden';
    //-- Si el puerto serie estaba ya abierto, cerrarlo
    if (portPrev) {
```

```
        await disconnect();
    }
    await connect();
    activeProgress.style.visibility = 'visible';
    progress.style.width = '25%';
    if(document.getElementById('uploadMbot').value == 'python'){
        send_code_to_mbot()
    }else{
        convert_and_send_code_to_mbot();
    }

}

async function connect() {
    portPrev = await navigator.serial.requestPort();

    await portPrev.open({ baudrate: 115200 });

}
async function disconnect() {
    // — Cerrar el puerto serie
    await portPrev.close();
    portPrev = null;
}
```

Fragmento 4.1: Abrir conexión con el Mbot por el puerto serie

4.3.2. Enviar el programa para conversión

Una vez abierta la comunicación de forma correcta, se procede a enviar el programa escrito al servidor mediante una petición GET y se introduce el programa como un query parameter en la URL. Para este envío se utiliza la función fetch() que está proporcionada por JavaScript.

```
function send_code_to_mbot() {
    var editor = ace.edit("ace");
    let code = editor.getValue();
    console.log(code);
    var enc = new TextEncoder();
    const message = {
        method: "GET"
     };
     url = '/get_python_to_arduino_binary?python_code=' + JSON.
         stringify(code);
     fetch(url, message)
           . . . . .
```

```
}
```

Fragmento 4.2: Envio del programa desde el navegador al servidor

Hay que tener en cuenta que si el programa está escrito en Blockly, debe convertirse previamente al lenguaje Python antes de enviarse, como puede verse en el fragmento 4.3.

```
var pythoncode = Blockly.Python.workspaceToCode(editor.ui);
```

Fragmento 4.3: Conversión de Blockly a Python

El servidor recibe la petición del navegador y extrae el código del programa.

```
def get_python_to_arduino_code_binary(request):
    ....
    python_code = json.loads(request.GET.get('python_code', None))
    ....
```

Fragmento 4.4: Extracción programa en el servidor

4.3.3. Conversión del programa

El programa extraído en el servidor es lenguaje Python, pero el Mbot, basado en Arduino como se mencionaba antes, no entiende este lenguaje. El lenguaje que usa Arduino es un C++ adaptado, por lo que se debe traducir desde Python. Esto es posible gracias a la librería PyOnArduino, la cual proporciona los mecanismos necesarios para esa traducción y conversión de un fichero.py (extensión para Python) a .ino (extensión para Arduino).

```
...
 parsed_file = ast.parse(code)
 translator.robot = 'MBot'
 translator.robot_architecture = ''
 translator.vars.Variables()
 translator.vars.halduino_directory = py_to_arduino_path + 'HALduino
    /halduino'
 translator.TranslatorVisitor().visit(parsed_file)
 translator.create_setup()
 translator.variables_manager = translator.create_variables_manager(
    file=py_to_arduino_path + 'HALduino/variablesManager.ino')
 translator.create_output('output-file.ino')
 translator.create_makefile('MBot', py_to_arduino_path + 'makefiles
    /')
...
```

Fragmento 4.5: Traducción de lenguaje Python a Arduino

4.3.4. Compilado del programa

Una vez extraído el código, se debe compilar para obtener el fichero .hex que contiene las instrucciones traducidas a hexadecimal entendidas por la microcontroladora. Esta compilación se lleva a cabo utilizando avrdude, una herramienta que permite la programación de chips AVR y que puede ser llamada por la línea de comandos.

```
def create_mbot_binary():
    shutil.copyfile(os.path.join(settings.BASE_DIR, 'kibotics-drivers/
        mbot/avrdude'), './avrdude')
    shutil.copyfile(os.path.join(settings.BASE_DIR, 'kibotics-drivers/
        mbot/libftdi1.so'), './libftdi.so')
    shutil.copyfile(os.path.join(settings.BASE_DIR, 'kibotics-drivers/
        mbot/libreadline.so'), './libreadline.so.6')
    shutil.copyfile(os.path.join(settings.BASE_DIR, 'kibotics-drivers/
        mbot/avrdude.conf'), './avrdude.conf')
    shutil.copytree(os.path.join(settings.BASE_DIR, 'kibotics-drivers/
        mbot/arduino-1.8.10/'), 'arduino/')
    call(['make'])
    exercise_dir = './'
    # Extraemos el binario
    f_binary = open(exercise_dir + 'build-uno/output.hex', 'r')
    binary = f_binary.read()
    f_binary.close()
    return binary
```

Fragmento 4.6: Compilado y obtención del .hex

Tras la compilación del programa, obtenemos el .hex, que será enviado como respuesta al navegador.

```
    ...
    binary = create_mbot_binary()

    response = HttpResponse(binary, content_type='text/plain')
    response['Content-Length'] = len(response.content)
    return response
```

Fragmento 4.7: Respuesta a la petición que envió el navegador

4.3.5. Respuesta de la petición

De la respuesta enviada por el navegador con el programa realizado al servidor, se recibe una respuesta que contiene el programa compilado para poder enviárselo a la microcontroladora.

La función fetch(), mencionada anteriormente y la que permitía enviar una petición de lado del cliente, espera hasta recibir la respuesta, pudiendo extraer así el cuerpo de la respuesta, donde se

encuentra la traducción del programa enviado.

```
fetch(url, message)
    .then(function(response) {
        if(response.ok){
            responseOk = true
        }else{
            responseOk = false
        }
          return response.text();
    })
    .then(function(data) {

        var dataBuffer = enc.encode(data);

    })
    .catch(function(err) {

        console.error(err);
     });
```

Fragmento 4.8: Extracción del dato enviado en la respuesta

4.3.6. Cargar compilado al robot

Una vez extraído el código compilado, se procede a cargarlo en el robot. Avrgirl-arduino, una librería nodejs de código abierto, proporciona esa posibilidad de quemar el programa compilado a la microcontroladora, y es compatible con un gran número de placas (entre ellas ArduinoUno). Hemos utilizado esta librería adaptándola a nuestras necesidades para el proceso de carga.

```
. . .

.then(function(data) {

    if(responseOk){
        var dataBuffer = enc.encode(data);
        upload_to_mbot(dataBuffer)
        progress.style.width = '99%';
        alertError.style.visibility = 'hidden';
    }else{
        console.log("Fallo al compilar el programa")
        infoError.innerHTML = "Error"
        alertError.style.visibility = 'visible';
        activeProgress.style.visibility = 'hidden';
    }
```

```
})
 . . .

 function upload_to_mbot(dataBuffer){
    let avrgirl = new AvrgirlArduino({
       board: "uno"
    });

    avrgirl.flash(dataBuffer,(error) => {
       if (error) {
          console.error(error);
          portPrev = null;
          infoError.innerHTML = "Error!"
          alertError.style.visibility = 'visible';
          activeProgress.style.visibility = 'hidden';

       } else {
          console.info('done correctly.');
          portPrev = null
          alertError.style.visibility = 'hidden';
          activeProgress.style.visibility = 'hidden';
       }
   });
}
```

Fragmento 4.9: Carga del compilado al Mbot

4.4. Resolución y experimentación

En la explicación anterior no se ha mencionado en ningún momento para qué sistema operativo era compatible, y es aquí donde recae una de las principales ventajas de este desarrollo, pues al tratarse de una tecnología de lado del navegador proporciona compatibilidad para cualquier sistema operativo que disponga del navegador Chrome. Con ello, se consigue que esta herramienta sea multiplataforma.

Además, al no necesitar una instalación previa de dependencias o programas por parte del usuario, su usabilidad es más sencilla. Esto es muy importante de cara al uso de la plataforma, ya que, al estar orientada a alumnos de baja edad, es necesario que el proceso sea lo más simple posible.

La principal desventaja es que, actualmente, Web Serial Api está en fase de experimentación en Chrome, por lo que necesita ser activada con un proceso previo. Para ello, debemos escribir en el navegador Chrome "chrome://flags" y habilitar el flag "Experimental web Plataform features".

Con el objetivo de verificar el funcionamiento se ha probado en diferentes sistemas operativos

y ordenadores. Entre los que han sido probados se encuentran:

- ***Ubuntu 18.04***.
- ***Ubuntu 16.04***
- ***Windows 10***.
- ***Windows 7***.
- ***MacOs Catalina Versión 10.15.6***.
- ***MacOs Mojave Versión 10.14.6***.