

**Instituto Tecnológico y de Estudios Superiores de  
Monterrey**

**Campus Toluca**



**Semana Tec: Herramientas computacionales: el arte de la  
programación**

**Nombre del profesor:**

*Mauricio Paletta Nannarone*

**Evidencia**

Ángel Octavio Mercado Pérez





| A01773013

## ¿Cuál es el objetivo del proyecto?

Crear una librería en la cual se empaquete (procesos y diferentes comandos, funciones, clases, parámetros, etc.) de la clase perteneciente al robot Hexapod. Esto con ayuda de herramientas las cuales permitan una conexión entre el Arduino IDE y Python, la complejidad radica en que el lenguaje usado en Arduino es C++ y el programa para controlar el robot por medio de programación orientada a objetos sería desarrollado en Python. Este programa con la finalidad de ayudar a los alumnos de preparatoria a hacer fácil y simple el llamado de los comandos.

## Desarrollo del proyecto

Comenzamos por una investigación acerca del robot Hexapod, donde encontramos un manual para la instalación de las librerías principales de control del robot, instalándolas en Arduino IDE para el correcto funcionamiento de los procesos a realizar, además de hacer la conexión entre el Arduino incorporado y Arduino IDE.

Nombre	Tipo
 FlexiTimer2	Carpeta comprimida (en z...
 FNHR	Carpeta comprimida (en z...
 RF24	Carpeta comprimida (en z...
 Servo	Carpeta comprimida (en z...

Se realizaron pruebas con los “Examples” que venían dentro de nuestras librerías, presentando un correcto funcionamiento del robot.

Después procedimos a investigar 2 diferentes formas de realizar la conexión entre estos 2 ambientes de desarrollo, una solución fue la librería de PySerial y la alternativa era implementar SWIG para permitir compilar el código de C++, instalando el compilador de g++ en la última versión.

## PySerial

Este código de Arduino hace que el robot se comuniquen con la computadora y recibe los comandos enviados desde Python. Gracias a la librería PySerial, Python puede usar el lenguaje que Arduino usa, para una correcta comunicación.

## ARDUINO

```
#ifndef ARDUINO_AVR_MEGA2560
#error Wrong board. Please choose "Arduino/Genuino Mega or Mega 2560"
#endif

#include <FNHR.h>

FNHR robot;

void setup() {
    Serial.begin(9600); // Inicia la comunicación serial
    robot.Start();      // Inicia el robot
    Serial.println("Robot listo. Esperando comandos...");
}

void loop() {
    if (Serial.available() > 0) {
        String command = Serial.readStringUntil('\n'); // Lee el comando desde Python
        command.trim(); // Elimina espacios y saltos de línea
        Serial.print("Comando recibido: ");
        Serial.println(command);
        executeCommand(command); // Ejecuta el comando
    }
}

void executeCommand(String command) {
    if (command == "CRAWL_FORWARD") {
        Serial.println("Ejecutando CrawlForward()");
        robot.CrawlForward();
    } else if (command == "CRAWL_BACKWARD") {
        Serial.println("Ejecutando CrawlBackward()");
        robot.CrawlBackward();
    } else if (command == "TURN_LEFT") {
        Serial.println("Ejecutando TurnLeft()");
    }
}
```

```

robot.TurnLeft();
} else if (command == "TURN_RIGHT") {
    Serial.println("Ejecutando TurnRight()");
    robot.TurnRight();
} else if (command.startsWith("LEG_MOVE")) {
    int leg = command.substring(9, 10).toInt();
    float x = command.substring(11, 20).toFloat();
    float y = command.substring(21, 30).toFloat();
    float z = command.substring(31, 40).toFloat();
    Serial.println("Ejecutando LegMoveToRelatively()");
    robot.LegMoveToRelatively(leg, x, y, z);
} else if (command.startsWith("CHANGE_HEIGHT")) {
    float height = command.substring(14).toFloat();
    Serial.println("Ejecutando ChangeBodyHeight()");
    robot.ChangeBodyHeight(height);
} else if (command.startsWith("ROTATE_BODY")) {
    float x = command.substring(12, 21).toFloat();
    float y = command.substring(22, 31).toFloat();
    float z = command.substring(32, 41).toFloat();
    Serial.println("Ejecutando RotateBody()");
    robot.RotateBody(x, y, z);
} else if (command.startsWith("TWIST_BODY")) {
    float xMove = command.substring(11, 20).toFloat();
    float yMove = command.substring(21, 30).toFloat();
    float zMove = command.substring(31, 40).toFloat();
    float xRotate = command.substring(41, 50).toFloat();
    float yRotate = command.substring(51, 60).toFloat();
    float zRotate = command.substring(61, 70).toFloat();
    Serial.println("Ejecutando TwistBody()");
    robot.TwistBody(xMove, yMove, zMove, xRotate, yRotate, zRotate);
} else if (command.startsWith("SET_SPEED")) {
    float speed = command.substring(10).toFloat();
    Serial.println("Ejecutando SetActionSpeed()");
    robot.SetActionSpeed(speed);
}

```

```
}  
}
```

## PYTHON

```
import serial
```

```
import time
```

```
# Configura el puerto serial (ajusta el puerto COM según tu sistema)
```

```
puerto = 'COM3' # Cambia esto al puerto correcto
```

```
velocidad_baudios = 9600
```

```
try:
```

```
    # Intenta abrir la conexión serial
```

```
    print(f"Conectando al Arduino en el puerto {puerto}...")
```

```
    arduino = serial.Serial(puerto, velocidad_baudios, timeout=1)
```

```
    time.sleep(2) # Espera a que se establezca la conexión
```

```
    print("Conexión establecida con el Arduino.")
```

```
except Exception as e:
```

```
    print(f"Error al conectar con el Arduino: {e}")
```

```
    exit()
```

```
def send_command(command):
```

```
    """
```

```
    Envía un comando al Arduino y muestra un mensaje de depuración.
```

```
    """
```

```
    try:
```

```
        print(f"Enviando comando: {command.strip()}") # Muestra el comando enviado
```

```
        arduino.write(command.encode()) # Envía el comando al Arduino
```

```
    except Exception as e:
```

```
        print(f"Error al enviar el comando: {e}")
```

```
def leer_respuesta():
```

```
    """
```

```
    Lee la respuesta del Arduino y la muestra en la consola.
```

```

"""

try:

    while arduino.in_waiting > 0: # Verifica si hay datos disponibles
        respuesta = arduino.readline().decode().strip() # Lee la respuesta
        print(f"Respuesta del Arduino: {respuesta}")
except Exception as e:
    print(f"Error al leer la respuesta: {e}")


# Funciones de control del robot

def crawl_forward():
    send_command("CRAWL_FORWARD\n")
    leer_respuesta()

def crawl_backward():
    send_command("CRAWL_BACKWARD\n")
    leer_respuesta()

def turn_left():
    send_command("TURN_LEFT\n")
    leer_respuesta()

def turn_right():
    send_command("TURN_RIGHT\n")
    leer_respuesta()

def leg_move_to_relatively(leg, x, y, z):
    send_command(f"LEG_MOVE {leg} {x} {y} {z}\n")
    leer_respuesta()

def change_body_height(height):
    send_command(f"CHANGE_HEIGHT {height}\n")
    leer_respuesta()

# Ejemplo de uso

```

```
print("Iniciando prueba del robot...")
crawl_forward() # El robot avanza
time.sleep(2)
turn_left()    # El robot gira a la izquierda
time.sleep(2)
crawl_backward() # El robot retrocede
time.sleep(2)
turn_right()   # El robot gira a la derecha
time.sleep(2)
print("Prueba completada.")

# Cierra la conexión serial
arduino.close()
print("Conexión serial cerrada.")
```

## **SWIG (Alternativa)**

### **¿Cómo funciona SWIG?**

Se escribe un archivo de interfaz (.i) que describe qué funciones y estructuras de C/C++ se expondrán al lenguaje de scripting.

SWIG procesa este archivo y genera:

Código fuente en C/C++ que actúa como puente entre el lenguaje de scripting y la implementación nativa.

Código específico para el lenguaje de destino, como módulos en Python (.py), archivos de clase en Java (.java), etc.

Se compila el código generado junto con la biblioteca original para crear el módulo de extensión.

El módulo resultante se carga en el lenguaje de scripting y permite acceder a la funcionalidad de C/C++ como si fuera código nativo.

### **¿Cómo implementaríamos SWIG?**

Aquí SWIG pide compilar con g++, por lo que de preferencia se tiene que instalar la última versión, con esto SWIG podrá enlazar nuestro código de C++ de Arduino con nuestro código de Python.

## ✓ Objetivo

Convertir este archivo `.cpp` ( `FNHR.cpp` ) en una **librería dinámica** ( `.dll` o `.so` ) que puedas importar y usar desde Python utilizando **SWIG** o **ctypes**.

## 📦 Paso 1: Organizar la Librería

Asegúrate de tener estos archivos:

- `FNHR.h` → Archivo de cabecera con las declaraciones.
- `FNHR.cpp` → Implementación de las funciones (el que mostraste).
- `communication.h` y `communication.cpp` → Parece que manejan la comunicación del hexápodo.
- `robotAction.h` y `robotAction.cpp` → Controlan los movimientos del robot.

## ⚙️ Paso 2: Crear una Interfaz para Python

Ahora, puedes usar **SWIG** para crear una interfaz que permita usar esta librería desde Python.

### 📄 Crear el Archivo de Interfaz ( `FNHR.i` )

```
swig                                Copiar  Editar

%module FNHR
%{
#include "FNHR.h"
%}

#include "FNHR.h"
```

Este archivo indica a SWIG que cree los enlaces entre Python y C++.



## Paso 4: Usar la Librería en Python

Crea un archivo `test.py`:

python

 Copiar

 Editar

```
import FNHR

# Crear instancia de la clase FNHR
hexapod = FNHR.FNHR()

# Ejecutar comandos
hexapod.Start(True)
hexapod.CrawlForward()
hexapod.TurnLeft()
hexapod.SleepMode()
```

Asegúrate de que:

- `_FNHR.so` (Linux/macOS) o `FNHR.dll` (Windows) esté en la misma carpeta.
- `FNHR.py` (el archivo wrapper de SWIG) también esté junto a `test.py`.



## Paso 3: Compilar con SWIG

1. **Generar los Wrappers** Ejecuta el siguiente comando para generar el código de envoltura (`.cxx`):

bash

 Copiar

 Editar

```
swig -c++ -python FNHR.i
```

2. **Compilar el Código** Asegúrate de que tienes `g++` o `MinGW` instalado si usas Windows:

bash

 Copiar

 Editar

```
g++ -fPIC -c FNHR.cpp FNHR_wrap.cxx -I/usr/include/python3.x
g++ -shared FNHR.o FNHR_wrap.o -o _FNHR.so
```

- Si usas Windows:

bash

 Copiar

 Editar

```
g++ -shared FNHR.cpp FNHR_wrap.cxx -o FNHR.dll -I"C:/Python311/include"
```

## Código para futuro empaquetamiento

```
import serial
```

```
import time
```

```
class Hexapodo:
```

```
    def __init__(self, puerto='COM3', velocidad_baudios=9600):
```

```
        self.puerto = puerto
```

```
        self.velocidad_baudios = velocidad_baudios
```

```
        self.arduino = None
```

```
        self.conectar()
```

```
    def conectar(self):
```

```
        try:
```

```
            print(f"Conectando al Arduino en el puerto {self.puerto}...")
```

```
            self.arduino = serial.Serial(self.puerto, self.velocidad_baudios,  
timeout=1)
```

```
            time.sleep(2)
```

```
            print("Conexión establecida con el Arduino.")
```

```
        except Exception as e:
```

```
            print(f"Error al conectar con el Arduino: {e}")
```

```
            exit()
```

```
    def enviar_comando(self, comando):
```

```
        try:
```

```
            print(f"Enviando comando: {comando.strip()}")
```

```
            self.arduino.write(comando.encode())
```

```
        except Exception as e:
```

```
            print(f"Error al enviar el comando: {e}")
```

```
    def leer_respuesta(self):
```

```
        try:
```

```

        while self.arduino.in_waiting > 0:
            respuesta = self.arduino.readline().decode().strip()
            print(f"Respuesta del Arduino: {respuesta}")
        except Exception as e:
            print(f"Error al leer la respuesta: {e}")

    def cerrar_conexion(self):
        if self.arduino:
            self.arduino.close()
            print("Conexión serial cerrada.")

class RobotHexapodo(Hexapodo):
    def avanzar(self):
        self.enviar_comando("CRAWL_FORWARD\n")
        self.leer_respuesta()

    def retroceder(self):
        self.enviar_comando("CRAWL_BACKWARD\n")
        self.leer_respuesta()

    def girar_izquierda(self):
        self.enviar_comando("TURN_LEFT\n")
        self.leer_respuesta()

    def girar_derecha(self):
        self.enviar_comando("TURN_RIGHT\n")
        self.leer_respuesta()

    def mover_pata (self, pata, x, y, z):
        self.enviar_comando(f"LEG_MOVE {pata} {x} {y} {z}\n")

```

```
self.leer_respuesta()
```

```
def cambiar_altura_cuerpo(self, altura):  
    self.enviar_comando(f"CHANGE_HEIGHT {altura}\n")  
    self.leer_respuesta()
```

Con este código en Python definimos los métodos de la clase Hexapodo para el futuro uso de este programa. En este caso después de empaquetar, el siguiente paso debería ser crear un programa donde importemos el código de arriba con nombre (por ejemplo) "hexaespanol.py" para tener nuestra librería en el código a realizar, seguidamente los alumnos de preparatoria tendrían que utilizar los métodos dispuestos en la librería para el correcto funcionamiento del robot.

#Ejemplo de uso

```
import hexaespanol.py
```

```
# Conectar al Arduino
```

```
Hexapodo.conectar()
```

```
print("Iniciando prueba del robot...")
```

```
# Moverse hacia adelante
```

```
Hexapodo.avanzar()
```

```
time.sleep(2)
```

```
# Girar a la izquierda
```

```
Hexapodo.girar_izquierda()
```

```
time.sleep(2)
```

```
# Moverse hacia atrás
```

```
Hexapodo.retroceder()
```

```
time.sleep(2)
```

```
# Girar a la derecha
```

```
Hexapodo.girar_derecha()
```

```
time.sleep(2)
```

```
# Mover una pata en coordenadas relativas (ejemplo: pata 1, desplazamiento  
en x=10, y=5, z=3)
```

```
Hexapodo.mover_pata_ (1, 10, 5, 3)
```

```
time.sleep(2)
```

```
# Cambiar la altura del cuerpo a 15 cm
```

```
Hexapodo.cambiar_altura_cuerpo(15)
```

```
time.sleep(2)
```

```
print("Prueba completada.")
```

```
# Cerrar la conexión serial
```

```
Hexapodo.cerrar_conexion()
```

Cabe recalcar que este proyecto no fue concluido debido a un malfuncionamiento del robot, por lo que no fue posible hacer mas pruebas para comprobar el correcto funcionamiento de todos los programas, aunque este documento es mi propuesta de lo que se debería de realizar después del punto en el que el equipo se quedó.

## **Referencias:**

*Freenove – Tutorial for V3.* (n.d.). <https://freenove.com/tutorial>