Instituto Tecnológico y de Estudios Superiores de Monterrey
Escuela de ingeniería

Materia:

Herramientas computacionales: el arte de la programación (Gpo 101)

Actividad:

Evidencia Documentación

Integrantes:

Valentina Tejeda Fuentes A01771768

Profesor:

Dr. Mauricio Paletta Nannarone

Fecha:

21 de marzo de 2025

Introducción

Una librería en programación es una colección de código desarrollado previamente que los programadores pueden utilizar para desarrollar software de manera más ágil.
Entonces el desarrollador no tiene que construir todo desde cero y puede utilizar funcionalidades de estas para construir su software.

En esta evidencia se documenta el desarrollo de una librería para conectar un robot Hexapod FNK0031 con Python, con el objetivo de facilitar su uso en talleres educativos para estudiantes de preparatoria.

Buscamos simplificar el control del robot usando python de manera que se pueda ejecutar comandos básicos sin preocuparse de la complejidad de la programación arduino.

Desarrollo

Para comenzar realizamos una investigación sobre el robot Hexapod FNK0031 para conocer sus funcionalidades y checar que el robot funcionara correctamente.

Se descargó la documentación oficial en un zip que contiene toda la información del robot para luego verificar su funcionamiento. Luego empezamos a conocer los métodos con los que trabaja en arduino, para después nosotros crear nuestras propias clases y métodos con los que funcionará el código de python.

Es importante recordar que Arduino trabaja con librerías que hacen las conexiones al robot. En Arduino, la clase principal utilizada para el control del robot es *FNHR*. Esta clase interactúa con dos componentes clave:

- *communication*: Maneja la conexión y comunicación con el robot.
- *robotAction*: Controla las acciones físicas del Hexapod.

Para esto se propuso crear una clase Hexápodo que servirá como la interfaz principal, y una clase comunicación que manejara la conexión del robot.

Esta fue la primera versión de lo que teníamos pensado utilizando Programación orientada a objetos (POO).

```
class Hexapod:
    def __init__(self):
        self.comunicacion = Comunicacion()

    def iniciar(self, usar_comunicacion=True):
        self.comunicacion.iniciar(usar_comunicacion)

    def avanzar_adelante(self):
        self.comunicacion.enviar_comando("CRAWL_FORWARD")

    def retroceder(self):
```

```
        self.comunicacion.enviar_comando("CRAWL_BACKWARD")

    def girar_izquierda(self):
        self.comunicacion.enviar_comando("TURN_LEFT")

    def girar_derecha(self):
        self.comunicacion.enviar_comando("TURN_RIGHT")
```

Primera forma

Ahora teníamos que buscar una forma de conectar el arduino con python, se acordó que el programa usará pyserial y time, que son unas bibliotecas que proporcionan funciones que nos van a facilitar la conexión.

- PySerial nos va a ayudar a establecer conexión con el Arduino; para importar la librería se debe tener instalado correctamente.
- time es una biblioteca estándar de Python que nos proporciona funciones para manejar el tiempo y las pausas en la ejecución de programas.

Primeramente se cargo un programa directamente desde arduino, con la finalidad de controlar un robot hexápodo a través de comandos seriales, enviados desde una conexión Python a través de Pyserial . Los comandos cubren movimientos básicos, ajustes de altura y rotaciones, ofreciendo un control completo sobre el robot.

```
#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);
  }
}
```

Después se ejecuta el siguiente código de python, este código tendrá varias funciones principales que envían comando a través del puerto serial como por ejemplo crawl_forward() y turn_left(). Y send_command () y leer_respuesta() maneja la comunicación entre python y el robot.

```
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.")
```

Para eso compilaba el programa pero no guardaba en la memoria del arduino las instrucciones, entonces nunca pudimos visualizar si realizaba las acciones que mandábamos desde python.

Segunda forma

Otra forma que intentamos realizar fue utilizando SWIG (Simplified Wrapper and Interface Generator) que es una herramienta que genera automáticamente enlaces entre programas escritos en C o C++ y varios lenguajes de programación como Python.

En este caso, como teníamos un proyecto en C++ y queremos controlarlo desde Python, SWIG permite hacer esto sin reescribir.

Debemos preparar los archivos necesarios en una carpeta, con las librerías necesarias para la conexión.

- FNHR.cpp: Archivo fuente de C++ con las funciones del hexápodo.
- FNHROrders.h
- FNHRBasic.h y FNHRBasic.cpp
- FNHRRemote.h y FNHRRemote.cpp
- FNHRComm.h y FNHRComm.cpp
- FNHR.h: Archivo de encabezado con las definiciones de las funciones.
- FNHR.i: Archivo de interfaz para SWIG (definirá qué funciones estarán disponibles en Python).
- test.py: Archivo de prueba para ejecutar funciones desde Python.

- Dentro del archivo llamado FNHR.i se debe escribir el siguiente contenido:

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

%include "FNHR.h"
```

- Después se generan archivos con SWIG.

Ejecutando el comando para generar los archivos .py y .cxx

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

Lo que crea un FNHR_wrap.cxx: Código intermedio que conecta C++ con Python.
y el archivo FNHR.py: Archivo Python que importará _FNHR.

- El siguiente paso es compilar el código Fuente y el Wrapper asegurandonos de tener g++ instalado ejecutamos el siguiente comando.

```
g++ -fPIC -c FNHR.cpp FNHR_wrap.cxx
-I"C:/Users/valen/AppData/Local/Programs/Python/Python313/include"
```

Este comando generará:

- FNHR.o
- FNHR_wrap.o

- Por último Creamos la biblioteca compartida .pyd con el siguiente comando:

```
g++ -shared FNHR.o FNHR_wrap.o -o _FNHR.pyd
-L"C:/Users/valen/AppData/Local/Programs/Python/Python313/libs" -lpython3.13
```

- Lo que nos genera

_FNHR.pyd

La estructura final de archivos es la siguiente:

FNHR.cpp → Código fuente principal del robot hexápodo.
FNHR.h → Archivo de encabezado del robot hexápodo.
FNHR.i → Archivo de interfaz para SWIG (debe contener las funciones que deseas exportar).
FNHR_wrap.cxx → Archivo generado por SWIG (contiene el código puente entre C++ y Python).
FNHR.py → Archivo Python generado por SWIG que importa _FNHR.
_FNHR.pyd → Biblioteca compartida compilada (equivalente a .dll en Windows).
test.py → Tu script Python para probar el funcionamiento de la librería.
FNHR.o → Archivo objeto de la compilación (g++ -c FNHR.cpp).
FNHR_wrap.o → Archivo objeto de la compilación del wrapper (g++ -c FNHR_wrap.cxx).
FNHROrders.h
FNHRBasic.h y FNHRBasic.cpp
FNHRRemote.h y FNHRRemote.cpp
FNHRComm.h y FNHRComm.cpp

- Finalmente probamos la librería desde python con este ejemplo de test.py, asegurándonos de que estén en la misma carpeta que _FNHR.pyd y FNHR.py

```
import FNHR

# Ejemplo de llamada a una función
robot = FNHR.FNHR()
robot.Start(False)
robot.CrawlForward()
print("El robot está avanzando.")
```

- Tuvimos un par de errores que se listan a continuación:

Esto nos marcaba errores como:
ModuleNotFoundError: No module named '_FNHR':

Para esto tenemos una lista de posibles soluciones:
Veificar que _FNHR.pyd esté en el mismo directorio que FNHR.py.
Verificar que se usa la versión correcta de Python (3.13.2).
Para el error cannot find -lpython3.13:

Confirma que python3.13.lib existe en tu carpeta de App Data C:/Users/valen/AppData/Local/Programs/Python/Python313/libs/.

Si no existe, instala Python con la opción de "Development Libraries".

Proyección a futuro del proyecto y solución individual.

Una vez que el proyecto actual funcione correctamente, se crearía la libreria en python llamada Hexapod, mi propuesta de funciones y la clase principal se muestra a continuación:

Esta librería tendrá funciones predeterminadas en español como:

Comando	Descripción	Parámetros
.conectar()	Conecta el robot hexápodo al puerto serial.	Ninguno.
.mover_adelante()	Mueve el hexápodo hacia adelante.	Ninguno.
.mover_atras()	Mueve el hexápodo hacia atrás.	Ninguno.
.mover_izquierda()	Gira el hexápodo hacia la izquierda.	Ninguno.
.mover_derecha()	Gira el hexápodo hacia la derecha.	Ninguno.
.Modo_Activo()	Activa el modo del robot (modo de funcionamiento).	Ninguno.
.Cambiar_Altura()	Cambia la altura del cuerpo del hexápodo.	altura (un valor numérico para la altura). Ej: 10.
.Mover_cuerpo()	Mueve el cuerpo del hexápodo en las tres dimensiones (x, y, z).	x, y, z (valores numéricos de las coordenadas). Ej: 5, 0, 2.
.Mover_pierna()	Mueve una pierna del hexápodo a una posición relativa.	pierna (la pierna que se moverá, ej: "F1" para la pierna delantera izquierda), x, y, z (valores numéricos de la nueva posición). Ej: "F1", 2, 3, 1.
.Establecer_velocidad()	Establece la velocidad de movimiento del robot.	velocidad (un valor entre 1, 2 o 3, donde 1 es baja, 2 es media y 3 es alta). Ej: 2.
.Rotar_cuerpo()	Rota el cuerpo del hexápodo.	Ninguno.
.girar_izquierda()	Gira el robot a la izquierda.	Ninguno.

.cambiar_velocidad()	Cambia la velocidad del robot durante el movimiento.	velocidad (un valor numérico, ej: 50 para una velocidad del 50%).
.desconectar()	Desconecta el robot y cierra la conexión serial.	Ninguno.

Ejemplo del código python para declarar las funciones y luego usarlo como librería:

```
import serial  # Importa la librería serial para la comunicación con el hardware
import time    # Importa la librería time para poder usar delays

class Hexapod:
    """
    Clase que representa un robot hexápodo controlado a través de comandos seriales.
    """

    def __init__(self, puerto='COM3', velocidad=9600):
        """
        Inicializa la conexión con el hexápodo a través del puerto serial.

        Args:
            puerto (str): El puerto serial al que se conecta el hexápodo (por defecto 'COM3').
            velocidad (int): La velocidad de comunicación en baudios (por defecto 9600).
        """
        try:
            # Establece la conexión serial con el hexápodo
            self.conexion = serial.Serial(puerto, velocidad, timeout=1)
            time.sleep(2)  # Espera 2 segundos para garantizar que la conexión esté establecida
            print("Conexión establecida con el hexápodo.")
        except Exception as e:
            # Maneja cualquier excepción en la conexión
            print(f"Error al conectar: {e}")

    def enviar_comando(self, comando):
        """
        Envía un comando al hexápodo.

        Args:
            comando (str): El comando que se enviará al hexápodo.
        """
        try:
            # Envia el comando codificado en bytes
            self.conexion.write(f"{comando}\n".encode())
            print(f"Comando enviado: {comando}")
```

```
        except Exception as e:
            # Maneja cualquier error al enviar el comando
            print(f"Error al enviar comando: {e}")

    def conectar(self):
        """
        Inicia la conexión y envía el comando de inicio al hexápodo.
        """
        print("Conectando al hexápodo...")
        self.enviar_comando("START")  # Envia el comando para iniciar la conexión

    def mover_adelante(self):
        """
        Envía el comando para mover el hexápodo hacia adelante.
        """
        self.enviar_comando("CRAWL_FORWARD")

    def mover_atras(self):
        """
        Envía el comando para mover el hexápodo hacia atrás.
        """
        self.enviar_comando("CRAWL_BACKWARD")

    def mover_izquierda(self):
        """
        Envía el comando para girar el hexápodo hacia la izquierda.
        """
        self.enviar_comando("TURN_LEFT")

    def mover_derecha(self):
        """
        Envía el comando para girar el hexápodo hacia la derecha.
        """
        self.enviar_comando("TURN_RIGHT")

    def modo_activo(self):
        """
        Activa el modo de funcionamiento del hexápodo.
        """
        self.enviar_comando("ACTIVATE_MODE")

    def cambiar_altura(self, altura):
        """
        Cambia la altura del cuerpo del hexápodo.

        Args:
            altura (int): El valor de la altura a la que se ajustará el cuerpo.
```

```
    """
    self.enviar_comando(f"CHANGE_BODY_HEIGHT {altura}")

def mover_cuerpo(self, x, y, z):
    """
    Mueve el cuerpo del hexápodo en las direcciones x, y, z.

    Args:
        x (int): Desplazamiento en el eje x.
        y (int): Desplazamiento en el eje y.
        z (int): Desplazamiento en el eje z.
    """
    self.enviar_comando(f"MOVE_BODY {x} {y} {z}")

def mover_pierna(self, pierna, x, y, z):
    """
    Mueve una pierna específica del hexápodo.

    Args:
        pierna (str): La pierna a mover (ej. "F1" para la pierna delantera izquierda).
        x (int): Desplazamiento en el eje x.
        y (int): Desplazamiento en el eje y.
        z (int): Desplazamiento en el eje z.
    """
    self.enviar_comando(f"LEG_MOVE_TO_RELATIVELY {pierna} {x} {y} {z}")

def establecer_velocidad(self, velocidad):
    """
    Establece la velocidad del robot.

    Args:
        velocidad (int): Velocidad que se quiere establecer (ej. 1, 2 o 3).
    """
    self.enviar_comando(f"SET_SPEED {velocidad}")

def rotar_cuerpo(self):
    """
    Rota el cuerpo del hexápodo.
    """
    self.enviar_comando("ROTATE_BODY")

def girar_izquierda(self):
    """
    Gira el robot hacia la izquierda.
    """
    self.enviar_comando("TURN_LEFT")
```

```
    def cambiar_velocidad(self, velocidad):
        """
        Cambia la velocidad de las acciones del hexápodo.

        Args:
            velocidad (int): Velocidad de la acción (un valor numérico).
        """
        self.enviar_comando(f"SET_ACTION_SPEED {velocidad}")

    def desconectar(self):
        """
        Cierra la conexión serial con el hexápodo.
        """
        if self.conexion.is_open:
            self.conexion.close()  # Cierra la conexión si está abierta
            print("Conexión cerrada.")
```

Prueba:

```
from ControladorHexapod import Hexapod
import time

# Crear una instancia del robot
robot = Hexapod()

# Conectar al robot
robot.conectar()

# Mover hacia adelante y esperar 2 segundos
robot.mover_adelante()
time.sleep(2)

# Mover hacia atrás y esperar 2 segundos
robot.mover_atras()
time.sleep(2)

# Girar a la izquierda y esperar 2 segundos
robot.mover_izquierda()
time.sleep(2)

# Girar a la derecha y esperar 2 segundos
robot.mover_derecha()
time.sleep(2)

# Activar el modo del robot
robot.modo_activo()
```

```
time.sleep(2)

# Cambiar la altura del cuerpo
robot.cambiar_altura(10)
time.sleep(2)

# Mover el cuerpo en 3 dimensiones
robot.mover_cuerpo(5, 0, 2)
time.sleep(2)

# Mover una pierna
robot.mover_pierna("F1", 2, 3, 1)
time.sleep(2)

# Establecer velocidad 3
robot.establecer_velocidad(3)
time.sleep(2)

# Rotar el cuerpo
robot.rotar_cuerpo()
time.sleep(2)

# Cambiar la velocidad a 50
robot.cambiar_velocidad(50)
time.sleep(2)

# Desconectar el robot
robot.desconectar()
```

Con el archivo que se importa como libreria para usar la clase hexapod, los estudiantes podrán realizar talleres prácticos enfocados en:

- Programación de movimientos básicos y avanzados.
- Creación de coreografías y competencias de robots.
- Experimentación con conceptos de cinemática y programación en Python.

Conclusiones:

Tuvimos algunos incovenientes a lo largo del proyecto, pero al final pudimos realizar cada quien su enfoque para simplificar considerablemente los códigos para el uso del robot Hexapod FNK0031 para los estudiantes lo cual les va a permitir usar Python, un lenguaje más accesible para la mayoría de los estudiantes de preparatoria, sin la necesidad de aprender a fondo la programación en Arduino.

El uso de diferentes librerías de python y la programación orientada a objetos en Python

proporciona una estructura clara y modular, que facilita el mantenimiento y la expansión del código en el futuro.

Referencias:

de, E. (2023, November 20). *¿Qué son las librerías en programación y para qué sirven?* UNIR; Universidad Internacional de La Rioja. https://www.unir.net/revista/ingenieria/librerias-programacion/