Tarea 6: Librerías para el Robot Pepper

Paula Sandoval

Abril 2025

Introducción

Este documento presenta una descripción de las principales librerías utilizadas en el desarrollo de aplicaciones para el robot **Pepper**. Se indican una descripción general y los requisitos de software necesarios para su correcto funcionamiento.

Requisitos de Software

- Python 2.7 o 3.x (dependiendo del entorno NAOqi utilizado)
- NAOqi SDK: entorno de desarrollo oficial para Pepper.
- Choregraphe Suite
- Acceso a red local del robot
- Conexión SSH

Librerías

1. `qi`

La librería `qi` es el cliente de Python para conectarse y controlar los módulos que forman parte de **NAOqi**, el framework operativo de los robots Nao y Pepper. Permite la comunicación con los servicios internos para controlar movimientos, voz, visión, sensores y otros componentes.

1.1. Funciones principales de `qi`

1.2. `qi.Session()`

Crea una sesión de comunicación con el robot o servidor NAOqi.

```
session = qi.Session()
```

1.3. `session.connect(url)`

Establece la conexión al robot usando TCP/IP, generalmente en el puerto 9559.

```
session.connect("tcp://192.168.1.100:9559")
```

1.4. `session.service("ServiceName")`

Obtiene una instancia de un servicio NAOqi.

```
motion_service = session.service("ALMotion")
tts_service = session.service("ALTextToSpeech")
```

1.5. `qi.Promise` y `qi.Future`

Mecanismos para programar tareas asíncronas sin bloquear el flujo del programa.

```
def tarea_larga():
    time.sleep(5)
    return "Tarea terminada"

promise = qi.Promise()
future = promise.future()

threading.Thread(target=lambda: promise.setValue(tarea_larga())).start()
print(future.value())
```

1.6. `qi.Application`

Maneja el ciclo de vida de las aplicaciones que interactúan con NAOqi.

```
app = qi.Application()
app.start()
session = app.session
```

1.7. Otros módulos importantes de `qi`

- `qi.Signal`: Emisión y escucha de eventos.
- `qi.PeriodicTask`: Ejecución repetida de tareas en intervalos definidos.
- `qi.ObjectHost`: Publicación de servicios personalizados.
- `qi.Log`: Registro de mensajes de error, advertencia e información.

1.8. Usos de `qi`

- Conexión remota al robot.
- Control de movimiento (brazos, cabeza, locomoción).
- ayuda a generar el proceso de voz
- Acceso a cámaras y sensores.
- Gestión de eventos y comportamientos inteligentes.

1.9. Funciones

Función	Descripción
`Session()`	Crea la conexión al robot
`connect(url)`	Establece la conexión
`service(name)`	Obtiene un servicio NAOqi
`Promise/Future`	Ejecuta tareas en segundo plano
`Application`	Maneja el ciclo de vida de la app
`Signal`	Comunicación por eventos

Ejemplo:

```
import qi
session = qi.Session()
session.connect("tcp://192.168.1.106")
motion_service = session.service("ALMotion")
```

2. `argparse`

`argparse` es una librería estándar de Python utilizada para gestionar argumentos de línea de comandos. Permite a los programas aceptar parámetros dinámicos cuando se ejecutan desde la terminal, facilitando configuraciones flexibles sin necesidad de modificar el código fuente.

2.1. Usos de `argparse`

- Procesar entradas de usuario desde la línea de comandos.
- Definir argumentos opcionales o obligatorios.
- Validar tipos de datos de entrada automáticamente.
- Generar mensajes de ayuda (`--help`) automáticamente.
- Mejorar la interacción de scripts con otros programas o usuarios.

2.2. Funciones principales

2.2.1. `argparse.ArgumentParser()`

Crea un nuevo parser para gestionar argumentos.

```
import argparse
parser = argparse.ArgumentParser(description="Mi programa de ejemplo")
```

2.2.2. `add_argument()`

Define qué argumentos acepta el programa.

```
parser.add_argument('--ip', type=str, required=True, help='Direcci n IP del
parser.add_argument('--port', type=int, default=9559, help='Puerto de conexi
```

2.2.3. `parse_args()`

Analiza los argumentos proporcionados por el usuario y los devuelve como un objeto accesible.

```
args = parser.parse_args()
print(args.ip)
print(args.port)
```

2.3. Otras características:

- Argumentos posicionales: No requieren prefijo (–).
- Subcomandos: Permiten crear comandos secundarios (como git add, git commit).
- Acciones especiales: Se pueden ejecutar acciones automáticas al detectar un argumento.
- Grupos de argumentos: Organización de argumentos relacionados.
- Estandariza la forma en que los scripts manejan parámetros externos.
- Permite ofrecer interfaces de línea de comandos profesionales.
- Simplifica validaciones y manejo de errores.
- Reduce la necesidad de código manual para leer entradas de usuario.

Documentación: argparse — Python Docs

3. `sys`

La librería `sys` es un módulo incorporado que proporciona acceso a funciones y objetos del sistema operativo y del intérprete de Python.Permite interactuar con argumentos de línea de comandos, controlar la salida estándar, terminar programas, gestionar módulos y acceder a información del sistema.

3.1. Funciones principales de sys

3.1.1. sys.argv

Contiene una lista de los argumentos pasados desde la línea de comandos. El primer elemento siempre es el nombre del script.

```
import sys
print(sys.argv)
```

Ejemplo:

```
python mi_script.py valor1 valor2
```

Resultado:

```
['mi_script.py', 'valor1', 'valor2']
```

3.1.2. sys.exit([arg])

Termina la ejecución del programa inmediatamente, opcionalmente pasando un código de salida.

```
import sys

if len(sys.argv) < 2:
    print("Error: Faltan argumentos")
    sys.exit(1)

print("Script ejecutado correctamente")
```

3.1.3. sys.path

Lista de directorios que Python busca para importar módulos. Permite añadir rutas personalizadas.

```
import sys
sys.path.append('/ruta/personalizada')
```

3.1.4. sys.version

Devuelve la versión de Python que se está utilizando.

```
import sys
print(sys.version)
```

—

3.1.5. `sys.platform`

Devuelve el nombre del sistema operativo donde se ejecuta Python.

```
import sys
print(sys.platform)
```

—

3.1.6. `sys.stdin`, `sys.stdout` y `sys.stderr`

Permiten interactuar con los flujos estándar de entrada, salida y error.

```
import sys
sys.stdout.write("Mensaje enviado a stdout\n")
```

—

3.2. Usos de `sys`

- Lectura de argumentos dinámicos al ejecutar scripts.
- Terminación controlada de programas.
- Gestión de rutas de módulos personalizados.
- Adaptación del comportamiento según el sistema operativo.
- Redirección de flujos de entrada y salida.

3.3. Resumen general

Función	Descripción
`sys.argv`	Acceso a los argumentos de la línea de comandos
`sys.exit()`	Finalizar el script
`sys.path`	Manipular rutas de importación
`sys.version`	Obtener versión de Python
`sys.platform`	Detectar el sistema operativo
`sys.stdin/stdout/stderr`	Flujos estándar de entrada y salida

Ejemplo:

```
import sys
if len(sys.argv) < 2:
    sys.exit("Falta la IP del robot")
```

Documentación: sys — Python Docs

4. os

La librería `os` proporciona una forma portátil de interactuar con el sistema operativo desde Python. Permite realizar operaciones relacionadas con el sistema de archivos, procesos, variables de entorno y más, de forma sencilla y multiplataforma.

Es un módulo de la biblioteca estándar de Python que ofrece funciones para:

- Navegar y manipular archivos y directorios.
- Ejecutar comandos del sistema operativo.
- Leer y modificar variables de entorno.
- Obtener información sobre el entorno de ejecución.

4.1. Funciones principales

4.1.1. `os.getcwd()`

Devuelve el directorio de trabajo actual.

```
import os
print(os.getcwd())
```

4.1.2. `os.chdir(path)`

Cambia el directorio de trabajo actual a la ruta especificada.

```
import os
os.chdir('/nueva/ruta')
```

4.1.3. `os.listdir(path)`

Devuelve una lista de los archivos y carpetas en el directorio especificado.

```
import os
print(os.listdir('.'))
```

4.1.4. `os.mkdir(path)` y `os.makedirs(path)`

Crea un nuevo directorio.

```
import os
os.mkdir('nueva_carpeta')
os.makedirs('ruta/mas/profunda', exist_ok=True)
```

4.1.5. `os.remove(path)` y `os.rmdir(path)`

Elimina un archivo o un directorio vacío.

```
import os
os.remove('archivo.txt')  # Eliminar archivo
```

4.1.6. `os.environ`

Permite acceder y modificar las variables de entorno.

```
import os
print(os.environ['HOME'])  # Mostrar variable de entorno
os.environ['NUEVA_VARIABLE'] = 'valor'
```

4.2. Usos comunes:

- Automatización de tareas de archivos y carpetas.
- Configuración de variables para la ejecución de programas.
- Integración con comandos y herramientas del sistema operativo.
- Creación de scripts portables entre diferentes sistemas.

4.3. Resumen general

Función	Descripción
`os.getcwd()`	Obtener el directorio de trabajo actual
`os.chdir(path)`	Cambiar el directorio de trabajo
`os.listdir(path)`	Listar contenido de un directorio
`os.mkdir(path)` / `os.makedirs(path)`	Crear directorios
`os.remove(path)` / `os.rmdir(path)`	Eliminar archivos o carpetas
`os.environ`	Manipular variables de entorno

```
import os
log_path = os.path.join(os.getcwd(), "logs")
os.makedirs(log_path, exist_ok=True)
```

Documentación: os — Python Docs

5. `almath`

La librería `almath` es un módulo proporcionado por SoftBank Robotics en el framework NAOqi. Está diseñado para realizar cálculos matemáticos específicos para robótica, especialmente aquellos relacionados con transformaciones espaciales, cinemática y geometría.

`proporciona funciones y clases para:`

- Manejar matrices de transformación homogénea (Transformaciones 4x4).
- Calcular trayectorias de movimiento.
- Realizar operaciones con ángulos, vectores y posiciones en el espacio 3D.
- Apoyar cálculos de cinemática directa e inversa.

5.1. Funciones principales de `almath`

5.1.1. `Transform()`

Clase que representa una transformación espacial (traslación y rotación) en 3D.

- Se puede inicializar como una matriz identidad o establecer valores manualmente.
- Permite multiplicaciones de transformaciones, inversas, etc.

5.1.2. `Position2D` y `Position3D`

Estructuras ligeras para almacenar posiciones en 2D o 3D.

- `Position2D`: maneja coordenadas (x, y, orientación).
- `Position3D`: maneja coordenadas (x, y, z).

5.1.3. `rotationFromAngleDirection()`

Calcula una matriz de rotación a partir de un ángulo y un vector de dirección.

- Útil para definir rotaciones arbitrarias en el espacio.

5.1.4. `Pose2D` y `Pose3D`

Modelos de poses (posición + orientación) en 2D o 3D.

- Se utilizan para representar la postura de un robot o un objeto.
- Permiten interpolaciones y combinaciones de poses.

5.1.5. `math functions`

Incluye funciones matemáticas generales optimizadas para robótica:

- Cálculo de distancias euclidianas.
- Normalización de ángulos.
- Conversión entre diferentes representaciones de rotación (matrices, ángulos de Euler, cuaterniones).

5.2. Usos de `almath`

- Calcular movimientos y trayectorias de robots humanoides.
- Convertir posiciones entre diferentes sistemas de referencia.
- Definir transformaciones entre partes del robot (base, torso, cabeza, extremidades).
- Apoyar en la resolución de problemas de cinemática inversa.

5.3. Resumen general

Función	Descripción
`Transform()`	Manejo de transformaciones homogéneas en 3D
`Position2D/3D`	Representación de posiciones en 2D o 3D
`rotationFromAngleDirection()`	Generar matrices de rotación
`Pose2D/3D`	Modelado de posiciones y orientaciones
`Funciones matemáticas`	Cálculos geométricos y de trayectorias

```
import almath
transform = almath.Transform()
transform.r1_c4 = 0.5
print(transform.toVector())
```

Documentación: almath — NAOqi SDK

6. `math`

La librería `math` es un módulo estándar de Python que proporciona acceso a funciones matemáticas básicas y avanzadas. Permite realizar cálculos precisos de trigonometría, exponenciales, logaritmos, raíces cuadradas, entre otros.

6.1. Funciones principales de `math`

6.1.1. `math.sin(x)`, `math.cos(x)`, `math.tan(x)`

Funciones trigonométricas que reciben ángulos en radianes.

```
import math
print(math.sin(math.radians(90)))  # 1.0
```

6.1.2. `math.sqrt(x)`

6.1.3. `math.pow(x, y)`

6.1.4. `math.log(x, base)`

6.1.5. `math.degrees(x)` y `math.radians(x)`

Convierte entre grados y radianes.

```
import math
print(math.degrees(math.pi))   # 180.0
print(math.radians(180))       # 3.14159...
```

6.2. Usos de `math`

- Cálculos de trayectorias y rotaciones en simulaciones robóticas.
- Conversión entre unidades de ángulo (grados y radianes).
- Resolución de problemas geométricos y físicos.
- Desarrollo de aplicaciones científicas y de ingeniería.

```
import math
print(math.sin(math.radians(90)))
```

Documentación: math — Python Docs

7. `ALMotion`

La librería `ALMotion` es un servicio fundamental en el framework NAOqi de SoftBank Robotics. Proporciona una interfaz para controlar todos los movimientos relacionados con el cuerpo del robot, como caminar, manipular articulaciones, y ejecutar movimientos. Es uno de los servicios más importantes para interactuar físicamente con el robot.

`ALMotion` permite:

- Mover articulaciones individuales o múltiples.
- Controlar el equilibrio y la postura del robot.
- Ejecutar trayectorias y movimientos cartesianos.
- Coordinar locomoción (caminar hacia adelante, girar, desplazarse lateralmente).

7.1. Funciones principales:

7.1.1. `setAngles(names, angles, fractionMaxSpeed)`

Mueve una o varias articulaciones hacia posiciones específicas.

- `names`: nombre(s) de las articulaciones.
- `angles`: valor(es) objetivo en radianes.
- `fractionMaxSpeed`: porcentaje de la velocidad máxima.

7.1.2. `moveTo(x, y, theta)`

Realiza un movimiento de traslación y rotación en el plano.

- `x`: desplazamiento hacia adelante (metros).
- `y`: desplazamiento lateral (metros).
- `theta`: rotación en radianes.

7.1.3. `positionInterpolation()`

Mueve el robot siguiendo una serie de posiciones y orientaciones definidas en el espacio cartesiano.

7.1.4. `getPosition(name, space)`

Obtiene la posición actual de un enlace del robot en un sistema de referencia dado.

- `name`: nombre del enlace (por ejemplo, `RHand`).
- `space`: marco de referencia (por ejemplo, `FRAME_TORSO`).

7.1.5. `setStiffnesses(names, stiffnesses)`

Configura la rigidez de una o más articulaciones, controlando qué tanto resisten el movimiento externo.

- `names`: nombre(s) de las articulaciones.
- `stiffnesses`: valor(es) entre 0.0 (suave) y 1.0 (firme).

7.2. Usos de `ALMotion`

- Programar movimientos personalizados de brazos, piernas, cabeza y torso.
- Realizar secuencias de caminata y navegación.
- Ajustar automáticamente la postura para mantener el equilibrio.
- Capturar la posición exacta de partes del cuerpo durante tareas dinámicas.

7.3. Resumen general

Función	Descripción
`setAngles`	Control de articulaciones individuales o múltiples
`moveTo`	Movimientos de locomoción (x, y, rotación)
`positionInterpolation`	Movimiento entre posiciones definidas
`getPosition`	Obtener posición de un enlace
`setStiffnesses`	Ajustar la rigidez de articulaciones

```
motion = session.service("ALMotion")
motion.moveTo(0.5, 0, 0)
```

Documentación: ALMotion — NAOqi SDK

8. `httplib / http.client`

Las librerías `httplib` (en Python 2) y `http.client` (en Python 3) proporcionan las funciones necesarias para realizar conexiones HTTP desde un script de Python. Permiten enviar solicitudes HTTP de bajo nivel y recibir respuestas desde servidores web.

- `httplib` fue introducido en Python 2 para gestionar conexiones HTTP y HTTPS manualmente.
- `http.client` es la versión actualizada de `httplib` en Python 3, manteniendo la funcionalidad principal pero mejorando la estructura del módulo.

8.1. Funciones de `http.client`

8.1.1. `HTTPConnection(host, port)`

Establece una conexión HTTP con un servidor.

```
import http.client
conn = http.client.HTTPConnection("example.com", 80)
```

8.1.2. `request(method, url, body=None, headers={})`

Envía una solicitud HTTP (como GET o POST) al servidor conectado.

```
conn.request("GET", "/")
```

8.1.3. `getresponse()`

Recupera la respuesta del servidor después de una solicitud.

```
response = conn.getresponse()
print(response.status, response.reason)
```

8.1.4. `read()`

Lee el contenido del cuerpo de la respuesta HTTP.

```
data = response.read()
print(data)
```

8.2. Usos de `http.client`

- Realizar solicitudes HTTP a servidores web de manera manual.
- Construir clientes HTTP personalizados.
- Analizar respuestas HTTP de baja capa.
- Conectar sistemas embebidos o robóticos a servicios web.

8.3. Ventajas de `http.client`

- Proporciona control de bajo nivel sobre las conexiones HTTP.
- Es parte de la biblioteca estándar de Python (sin dependencias externas).
- Útil para aplicaciones ligeras que no requieren bibliotecas pesadas como `requests`.

8.4. Resumen general

Función	Descripción
`HTTPConnection(host, port)`	Crear una conexión HTTP
`request(method, url)`	Enviar una solicitud al servidor
`getresponse()`	Obtener la respuesta del servidor
`read()`	Leer el cuerpo de la respuesta

```
import http.client
conn = http.client.HTTPConnection("example.com")
conn.request("GET", "/")
resp = conn.getresponse()
print(resp.status)
```

Documentación: http.client — Python Docs

9. `json`

La librería `json` es un módulo estándar en Python que permite codificar y decodificar datos en formato JSON (JavaScript Object Notation). JSON es un formato ligero de intercambio de datos, fácil de leer y escribir tanto por humanos como por máquinas.

Proporciona funciones para:

- Convertir estructuras de datos de Python (diccionarios, listas) en cadenas JSON.
- Convertir cadenas JSON en estructuras de datos de Python.
- Leer archivos JSON y escribir archivos en formato JSON.

9.1. Funciones de `json`

9.1.1. `json.dumps(obj)`

Convierte un objeto de Python en una cadena JSON.

```
import json
data = {"nombre": "Pepper", "edad": 3}
json_string = json.dumps(data)
```

9.1.2. `json.loads(s)`

Convierte una cadena JSON en un objeto de Python.

```
import json
s = '{"nombre":"Pepper","edad":3}'
data = json.loads(s)
```

9.1.3. `json.dump(obj, file)`

Escribe un objeto de Python en un archivo como JSON.

```
import json
data = {"nombre": "Pepper", "edad": 3}
with open('data.json', 'w') as f:
    json.dump(data, f)
```

9.1.4. `json.load(file)`

Lee datos JSON desde un archivo y los convierte en un objeto de Python.

```
import json
with open('data.json', 'r') as f:
    data = json.load(f)
```

9.2. Usos de `json`

- Guardar configuraciones de aplicaciones en archivos JSON.
- Intercambiar datos entre aplicaciones web y servidores.
- Serializar datos para almacenamiento o transmisión en red.
- Procesar respuestas de APIs web que devuelven datos en JSON.

9.3. Resumen general

Función	Descripción
`json.dumps(obj)`	Serializar un objeto Python a una cadena JSON
`json.loads(s)`	Deserializar una cadena JSON a objeto Python
`json.dump(obj, file)`	Escribir JSON en un archivo
`json.load(file)`	Leer JSON desde un archivo

```
import json
data = {"nombre": "Pepper", "edad": 3}
print(json.dumps(data))
```

Documentación: json — Python Docs

Estas librerías forman parte del ecosistema de desarrollo para Pepper. Se recomienda explorar sus respectivas documentaciones oficiales para un mayor dominio de su uso.

Bibliografía

- SoftBank Robotics. (2023). NAOqi Framework. Recuperado de: `http://doc.aldebaran.com/2-5/index.html`
- Python Software Foundation. (2023). Python Documentation. `https://docs.python.org/3/`
- SoftBank Robotics. (2023). Motion API. `http://doc.aldebaran.com/2-5/naoqi/motion/almotion.html`
- Aldebaran Robotics. (2018). Choregraphe SDK. `http://doc.aldebaran.com/2-5/software/choregraphe/index.html`