Universidad Nacional de Colombia

Sede Bogotá

Facultad de Ingeniería

Departamento de Ingeniería Mecánica y Mecatrónica

Referencia de la API de la pata del robot Cheetah

Daniel Esteban Ramírez Chiquillo

6 de marzo de 2025

Índice

1. Introducción

Este documento proporciona una referencia completa para la API HTTP implementada en el firmware del robot. La API permite el control remoto del robot a través de un conjunto de endpoints HTTP. Todas las solicitudes y respuestas (excepto el contenido estático) utilizan el formato JSON. Este documento está dirigido a desarrolladores que integren o mantengan el sistema de control del robot.

1.1. Visión General

- **URL Base:** El robot es accesible mediante `http://cheetah.local)`
- **Versión de la API:** 1.0
- **Formato de Respuesta:** JSON (salvo el contenido estático)
- **Seguridad:** Las operaciones críticas verifican el estado de *recovery*. Si se requiere un procedimiento de recuperación, los comandos se rechazan e indican al usuario las acciones seguras a realizar.

2. Endpoints de la API

2.1. Endpoints para Contenido Estático

GET / y GET /*

Propósito: Servir archivos estáticos almacenados en el sistema de archivos SPIFFS (por ejemplo, `index.html`, archivos CSS, JS).

Detalles:

- `GET /` retorna la página principal (`index.html`).
- `GET /*` actúa como comodín para servir otros archivos. El tipo MIME se establece según la extensión:
  - `.css` → `text/css`
  - `.js` → `application/javascript`
  - `.html` → `text/html`
  - Otros archivos se sirven con `text/plain`.

Manejo de Errores: Si el archivo no se encuentra, se retorna un error 404.

2.2. Endpoints de Control de Energía del Robot

POST /api/robot/on

Descripción: Este endpoint enciende el robot. Inicia la secuencia para activar motores, sensores y relés, verificando previamente que el sistema no se encuentre en estado de *recovery*.

Solicitud:

- **Método:** POST
- **Cuerpo:** Ninguno.

Proceso:

- Se verifica que el sistema no esté en estado de *recovery*.
- Se invoca la función interna `robot_controller_turn_on()`.

Respuesta de Éxito:

```
{
  "status": "ok",
  "message": "Robot turned ON successfully"
}
```

Respuesta de Error: Si el sistema requiere recovery o ocurre otro fallo:

```
{
  "status": "error",
  "message": "Recovery needed. Please physically power
              off each motor..."
}
```

POST `/api/robot/off`

Descripción: Apaga el robot, desactivando los motores y relés, siempre que el sistema no se encuentre en estado de *recovery*.

Solicitud:

- **Método:** POST
- **Cuerpo:** Ninguno.

Proceso:

- Se comprueba que el sistema no esté en estado de *recovery*.
- Se invoca la función `robot_controller_turn_off()`.

Respuesta de Éxito:

```
{
  "status": "ok",
  "message": "Robot turned OFF successfully"
}
```

Respuesta de Error:

```
{
  "status": "error",
  "message": "Failed to turn off"
}
```

2.3. Endpoint de Gestión de Recovery

POST `/api/recovery/clear`

Descripción: Limpia la bandera de *recovery* del sistema. Este endpoint debe llamarse únicamente después de haber realizado el siguiente procedimiento:

1. Apagar físicamente cada motor (usando el switch o botón de emergencia).
2. Mover manualmente todos los motores a la posición `home`.
3. Volver a encender los motores.

Solicitud:

- **Método:** POST
- **Cuerpo:** Ninguno.

Respuesta de Éxito (Recovery Limpiado):

```
{
  "status": "ok",
  "message": "Recovery cleared. Motors are now considered safe.
             You may now use other commands."
}
```

Respuesta Alternativa (Sin Recovery Necesario):

```
{
  "status": "ok",
  "message": "No recovery is needed. System is normal."
}
```

Respuesta de Error:

```
{
  "status": "error",
  "message": "Failed to clear recovery state"
}
```

2.4. Endpoints de Comandos para Motores

POST `/api/command/move`

Descripción: Envía un comando de movimiento a un motor individual, estableciendo la posición objetivo y la velocidad deseada.

Solicitud:

- **Método:** POST
- **Cuerpo:** Un objeto JSON con la siguiente estructura:

```
{
  "motor_id": 1,          // Valores permitidos: 1, 2 o 3
  "position": 45.0,       // Posicion deseada en grados
  "speed": 180.0          // Velocidad en grados por segundo
}
```

Proceso:

- Se valida el formato JSON y se comprueba que los valores sean numéricos.
- Se verifica que el robot esté encendido y que no se encuentre en estado de *recovery*.
- Se convierten la posición y la velocidad de grados a radianes (multiplicando por $\pi/180$).
- Se asegura que la velocidad no exceda el límite máximo permitido para el motor (definido en constantes como `MOTOR1_MAX_SPEED_DPS`, etc.).

Respuesta de Éxito:

```
{
  "status": "ok",
  "message": "Move command accepted"
}
```

Respuestas de Error:

- Si el JSON es inválido o faltan parámetros:

```
{
  "status": "error",
  "message": "Missing or invalid 'position'"
}
```

- Si la velocidad excede el límite permitido:

```
{
  "status": "error",
  "message": "Speed exceeds maximum allowed (XXX deg/s)"
}
```

- Si el robot está apagado:

```
{
  "status": "error",
  "message": "Robot is off. Please turn on the robot first."
}
```

POST `/api/command/batch`

Descripción: Permite enviar un lote de comandos de movimiento para uno o más motores. Los comandos dirigidos a motores diferentes se ejecutan en paralelo, mientras que los comandos para un mismo motor se procesan de forma secuencial.

Solicitud:

- **Método:** POST
- **Cuerpo:** Un objeto JSON que contenga un arreglo `batch` con cada comando:

```
{
  "batch": [
    {"motor_id": 1, "position": 45,  "speed": 100},
    {"motor_id": 2, "position": 90,  "speed": 50},
    {"motor_id": 3, "position": 135, "speed": 25},
    {"motor_id": 1, "position": 0,   "speed": 50},
    {"motor_id": 2, "position": 0,   "speed": 75},
    {"motor_id": 3, "position": 0,   "speed": 100}
  ]
}
```

Proceso:

1. Se verifica que no exista otro lote en ejecución.
2. Se validan individualmente todos los comandos (tipos numéricos, rangos válidos y motor_id correcto).
3. Se convierten las posiciones y velocidades de grados a radianes.
4. Se asignan y encolan los comandos para cada motor.
5. Se espera a que todos los comandos se ejecuten o se produzca un error.

Respuesta de Éxito: Se retorna un resumen para cada motor:

```
{
  "global_status": "ok",
  "motors": [
    {
      "motor_id": 1,
      "status": "ok",
      "commands_executed": 2
    },
    {
      "motor_id": 2,
      "status": "ok",
      "commands_executed": 2
    },
    {
      "motor_id": 3,
      "status": "ok",
      "commands_executed": 2
    }
  ]
}
```

Respuestas de Error:

- Si algún comando del lote tiene un formato incorrecto:

```
{
  "global_status": "error",
  "message": "Invalid command format in batch"
}
```

- Si ya existe un lote en ejecución:

```
{
  "global_status": "error",
  "message": "Batch already in progress"
}
```

- Si un motor específico encuentra un error durante la ejecución, su resumen incluirá:

```
{
  "motor_id": X,
  "status": "error",
  "error_message": "Detailed error description",
  "commands_executed": Y
}
```

3. Manejo General de Errores

- Todas las respuestas de la API incluyen un campo `status` con los valores «ok» o «error»
- Las respuestas de error incluyen un campo `message` con una explicación legible para el usuario.
- Se utilizan códigos de estado HTTP estándar (por ejemplo, 200 para éxito, 404 para no encontrado, 500 para errores internos).

4. Consideraciones de Seguridad

- **Verificación del Estado de Recovery:** Antes de ejecutar comandos que modifiquen el comportamiento de los motores, la API verifica si el sistema se encuentra en estado de *recovery*. En tal caso, se rechaza la operación y se notifica al cliente.
- **Validación de Entradas:** Todos los endpoints que aceptan payloads JSON realizan una validación rigurosa de los datos para prevenir operaciones inseguras (por ejemplo, movimientos que excedan los límites físicos).
- **Procesamiento Secuencial en Batch:** Para las operaciones en lote, los comandos dirigidos al mismo motor se procesan de forma secuencial.

5. Apéndice

5.1. Historial de Revisiones

- **v1.0** - Versión inicial de la referencia de la API.