Universidad Nacional de Colombia

Sede Bogotá

Facultad de Ingeniería

Departamento de Ingeniería Mecánica y Mecatrónica

Documentación de la Arquitectura del Firmware del Robot Cheetah

Daniel Esteban Ramirez Chiquillo

6 de marzo de 2025

Índice

1. Introducción

Este documento describe la arquitectura interna del firmware desarrollado para la ESP32 que se usa como control del la pata del robot Cheetah. Se abordan los componentes principales del sistema, su funcionalidad, y la forma en que interactúan. La intención es facilitar a futuros contribuyentes la comprensión del código base.

2. Visión General del Sistema

El firmware se ejecuta en una plataforma ESP32 y está estructurado en múltiples componentes que se encargan de diferentes tareas, tales como la comunicación en bus CAN, la generación de trayectorias de movimiento, el manejo de comandos de motores, la gestión de la conectividad WiFi y el servicio HTTP, entre otros. A grandes rasgos, el sistema se divide en:

- **Comunicación y Control de Motores:** Involucra la comunicación con los motores a través del bus CAN, el procesamiento de comandos y la generación de trayectorias.
- **Interfaz de Usuario y Conectividad:** Proporciona una API y gestiona la conectividad WiFi y el servicio mDNS.
- **Coordinación Global:** Un controlador de robot de alto nivel que coordina la activación, apagado, y manejo de estados críticos (por ejemplo, modos de recuperación).

3. Componentes del Sistema

3.1. CAN Bus

Ubicación: `./components/can_bus/`
Funcionalidad:

- Inicializa el driver TWAI (CAN) de ESP32.
- Proporciona funciones para limpiar el bus (`can_bus_flush`) y enviar mensajes con espera de respuesta (`can_bus_request_response`).
- Define comandos preestablecidos (p.ej., entrar/salir del modo motor, cero del sensor de posición).

Este componente encapsula toda la lógica necesaria para la comunicación CAN, aislando detalles de hardware y facilitando su uso por otros módulos.

3.2. Servidor HTTP

Ubicación: `./components/http_server/`
Funcionalidad:

- Implementa un servidor HTTP basado en la librería `esp_http_server`.
- Sirve contenido estático desde SPIFFS (por ejemplo, `index.html`, `style.css`, `script.js`).

- Registra endpoints para comandos del robot (encendido, apagado, recuperación, y comandos de movimiento individuales y por lote).

Este módulo actúa como la puerta de entrada para la interacción remota, permitiendo a los usuarios controlar el robot mediante solicitudes HTTP.

3.3. Parser de Mensajes

Ubicación: `./components/message_parser/`
Funcionalidad:

- Empaqueta datos de comandos en un formato de 8 bytes para la transmisión CAN (`pack_cmd`).
- Desempaqueta respuestas recibidas de los motores (`unpack_reply`).
- Provee funciones de mapeo entre valores flotantes y enteros, adaptando rangos definidos por el hardware.

El parser de mensajes garantiza la correcta codificación y decodificación de los datos intercambiados entre el firmware y los motores.

3.4. Perfil de Movimiento

Ubicación: `./components/motion_profile/`
Funcionalidad:

- Genera trayectorias de movimiento suaves (S-curve) para los motores, basadas en parámetros como velocidad, aceleración y tiempo de muestreo.
- Calcula el número de puntos y las posiciones, velocidades y aceleraciones correspondientes para realizar un movimiento controlado.

Este módulo es fundamental para evitar movimientos bruscos, permitiendo una transición gradual en la posición de los motores.

3.5. Manejadores de Comandos de Motor

Ubicación: `./components/motor_command_handlers/`
Funcionalidad:

- Recibe un comando de motor y lo despacha al manejador adecuado (por ejemplo, entrar en modo, salir de modo, cero de sensor, o movimiento).
- Valida y procesa parámetros del comando, utilizando internamente el *parser de mensajes* y el *perfil de movimiento*.

Este componente centraliza la lógica de decisión para los diferentes tipos de comandos que pueden ejecutarse en cada motor.

3.6. Núcleo de Control de Motores

Ubicación: `./components/motor_control_core/`
Funcionalidad:

- Inicializa y mantiene el estado de cada motor, incluyendo parámetros como la posición actual, estado de compromiso (engaged) y datos de trayectorias activas.
- Provee funciones para acceder y modificar el estado de los motores (p.ej., `motor_control_get_stat` y `motor_control_move_blocking`).
- Gestiona comandos en lote, permitiendo la ejecución secuencial y paralela según corresponda.

Este módulo es el corazón del sistema de control, conectando los comandos de alto nivel con la ejecución física de los motores.

3.7. Tarea de Control de Motores

Ubicación: `./components/motor_control_task/`
Funcionalidad:

- Implementa una tarea de FreeRTOS que se ejecuta continuamente.
- Envía puntos de trayectorias a los motores, procesa comandos en lote y monitorea fallos en la comunicación CAN.
- Realiza abortos globales en caso de errores críticos, asegurando una operación segura.

Esta tarea opera en segundo plano, orquestando la ejecución de las trayectorias y asegurando la continuidad del control de los motores.

3.8. Controlador del Robot

Ubicación: `./components/robot_controller/`
Funcionalidad:

- Coordina las operaciones globales del robot, como encender y apagar, y la gestión de estados críticos (por ejemplo, el modo de *recovery*).
- Se encarga de iniciar secuencias de activación que involucran la puesta a cero de sensores, el cambio de modos de los motores y la actualización de banderas en NVS.

El controlador del robot actúa como interfaz entre los comandos de usuario y la operación a nivel de motor, asegurando que las acciones críticas se ejecuten en el orden correcto.

3.9. Gestor de WiFi

Ubicación: `./components/wifi_manager/`
Funcionalidad:

- Inicializa la conectividad WiFi en modo estación, configurando la conexión al SSID especificado.
- Configura el servicio mDNS para facilitar la detección del dispositivo en la red.
- Gestiona eventos de conexión, reconexión y asignación de direcciones IP.

Este componente permite la interacción remota con el robot mediante la red, integrándose con el servidor HTTP.

3.10. Programa Principal

Ubicación: `./main/main.c`
Funcionalidad:

- Es el punto de entrada del firmware, encargado de la inicialización de NVS, montaje de SPIFFS y configuración del hardware (por ejemplo, relés y pines GPIO).
- Inicia la conectividad WiFi, el servidor HTTP y lanza la tarea de control de motores.
- Verifica el estado previo del sistema (por ejemplo, motores comprometidos de ejecuciones anteriores) para activar el modo de *recovery* si es necesario.

Este archivo coordina el arranque de todos los componentes, asegurando que el sistema se encuentre en un estado coherente antes de comenzar a operar.

3.11. Contenido Estático (SPIFFS)

Ubicación: `./spiffs/`
Funcionalidad:

- Contiene archivos web (`index.html`, `script.js`, `style.css`) que constituyen la interfaz gráfica para el control remoto del robot.
- Estos archivos son servidos por el módulo `http_server` y permiten al usuario enviar comandos y visualizar notificaciones.

4. Interconexión y Flujo de Datos

Los componentes descritos interactúan de la siguiente manera:

1. La interfaz web (contenido en SPIFFS) es servida por el **servidor HTTP** y permite al usuario enviar solicitudes (por ejemplo, encender/apagar el robot o mover un motor). Esta interfaz realiza solicitudes HTTP al servidor
2. Las solicitudes HTTP son recibidas por el servidor, el cual invoca funciones del **controlador del robot**. Este controlador valida el estado (por ejemplo, chequeando si se requiere *recovery*) y dirige la operación.

3. Las operaciones de movimiento se envían al **núcleo de control de motores**, que actualiza el estado interno de cada motor y, mediante el **manejador de comandos de motor**, empaqueta y transmite las instrucciones a través del **CAN Bus**.

4. El **parser de mensajes** se encarga de convertir los parámetros de los comandos a un formato adecuado para la transmisión y, de forma inversa, interpreta las respuestas de los motores.

5. La **tarea de control de motores** se ejecuta periódicamente para enviar los puntos de la trayectoria a los motores, gestionar comandos en lote y monitorear la comunicación. En caso de errores, se coordinan abortos globales.

6. El **gestor de WiFi** y la configuración de mDNS aseguran la conectividad de red, haciendo accesible la interfaz web.

7. Finalmente, el **programa principal** orquesta la inicialización de todos estos componentes, asegurando que el sistema comience a operar en un estado consistente.

5. Consideraciones para Contribuidores

- **Modularidad:** Cada componente está diseñado para ser lo más autónomo posible. Se recomienda mantener esta separación al agregar nuevas funcionalidades.
- **Comunicación CAN:** Cualquier modificación en el formato o en la gestión del bus CAN debe realizarse preferiblemente en el módulo `can_bus` y ser propagada a través del `parser de mensajes`.
- **Generación de Trayectorias:** El módulo `motion_profile` puede extenderse para soportar otros tipos de trayectorias, pero se debe garantizar la compatibilidad con el `motor_control_core`.
- **Gestión de Estados:** El `robot_controller` es responsable de los estados críticos (como el modo de *recovery*). Se debe tener especial cuidado al modificar esta lógica, ya que impacta en la seguridad de la operación.
- **Interfaz Web:** Las modificaciones en la interfaz de usuario deben ser coherentes con los endpoints definidos en el `http_server` para asegurar una comunicación bidireccional correcta.

6. Apéndice

6.1. Historial de Revisiones

- **v1.0** - Documentación inicial de la arquitectura.