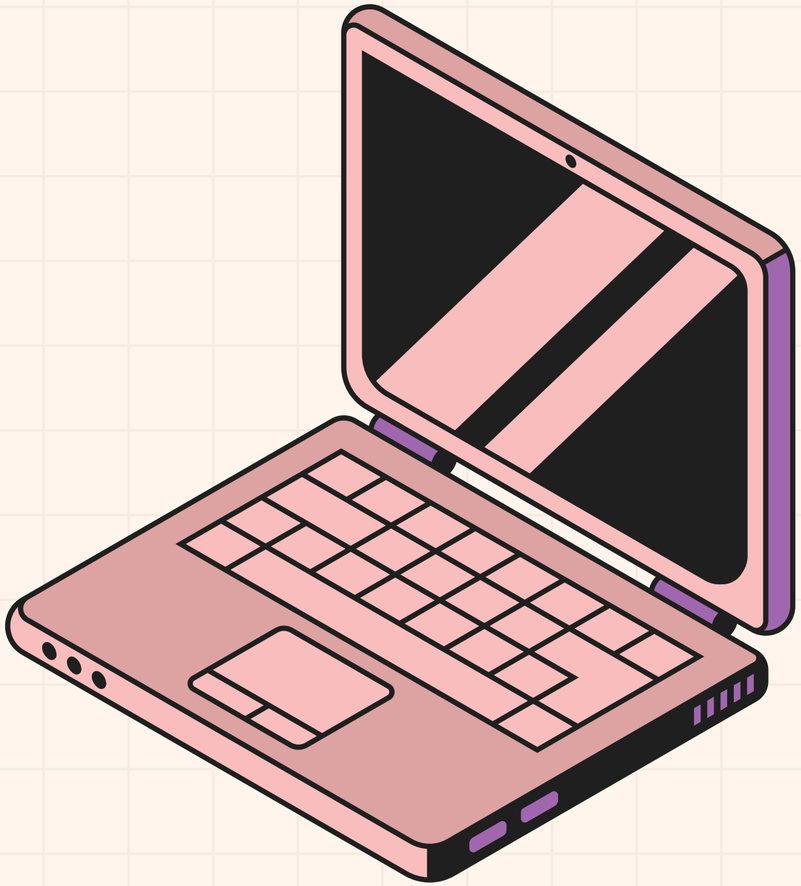
DSI

AUTOPILOT PARA QUADRICÓPTERO

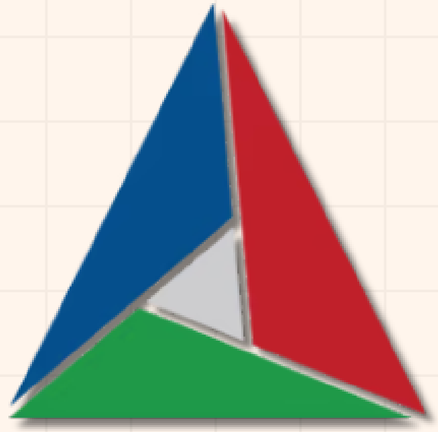
Hugo Blanco Demelo
Renato Bedriñana Cárdenas
Pablo Blanco Pumar
Adrián Cabaleiro Althoff
Aarón Riveiro Vilar

ÍNDICE



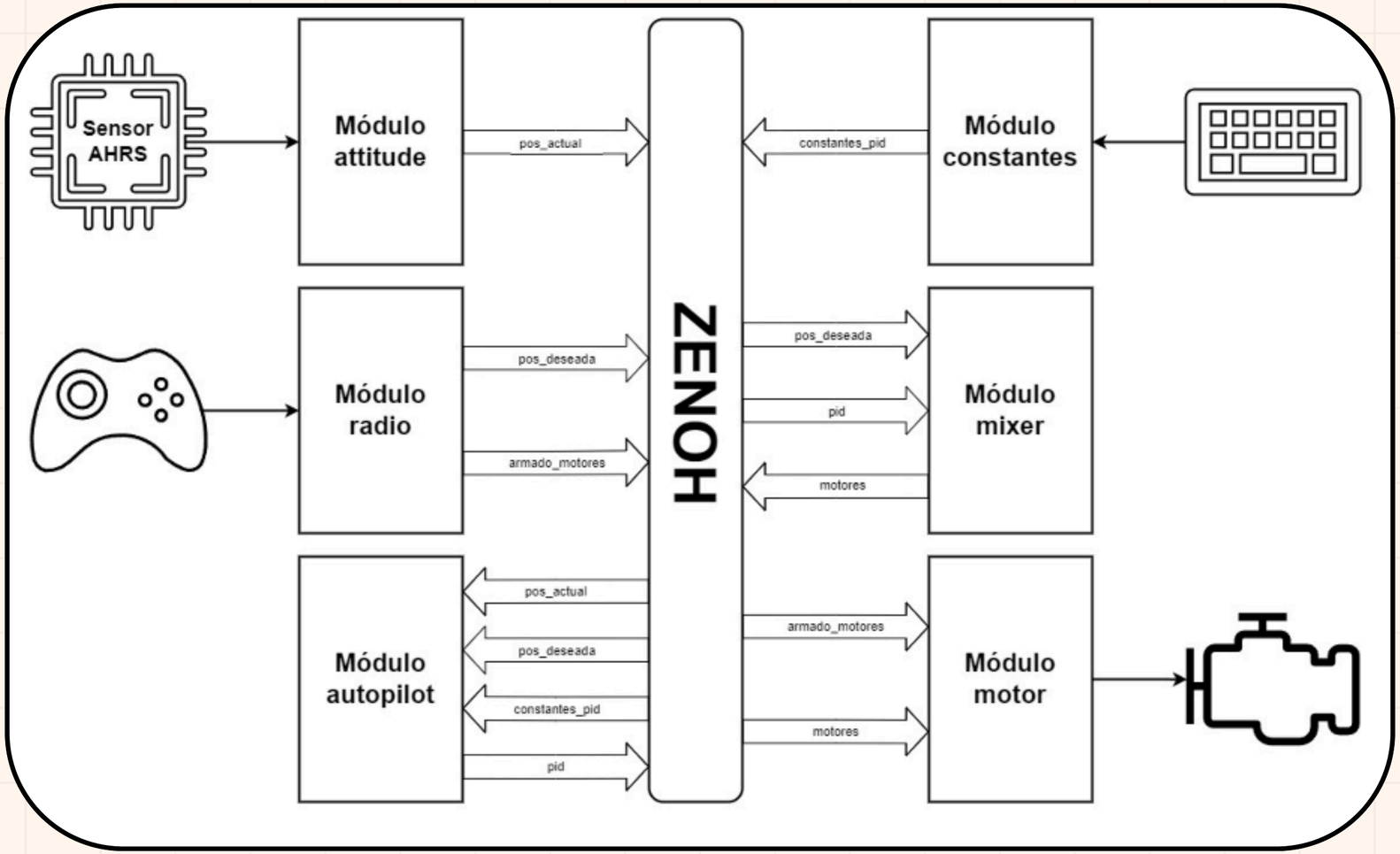
DECISIONES DE DISEÑO

- **Lenguaje de programación:** C++
- **Protocolo de comunicación:** Zenoh
- **Compilación:**
  - Un makefile para cada módulo
  - Un makefile global
- **Ejecución:**
  - Cada módulo individualmente
  - Ejecutable global

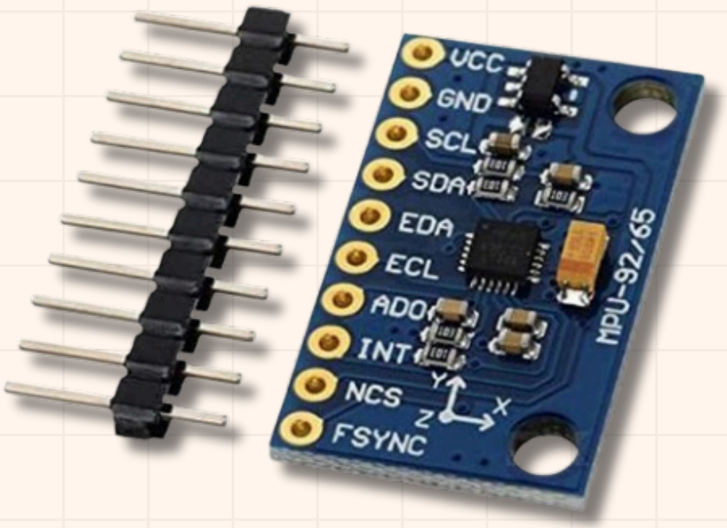


Zenoh

ARQUITECTURA DEL SISTEMA



MÓDULO ATTITUDE



Lee datos del sensor AHRS para conocer la orientación del dron

- Lee el fichero de configuración de RTIMULib
- Inicializa el sensor
- Lee los datos y los publica cada 10 milisegundos

Entrada

- Sensor AHRS
- Fichero de configuración RTIMULib

Salida

Tópico pos_actual:

- Roll, pitch, yaw: [-180, 180]

MÓDULO RADIO

Lee datos del mando para conocer las intenciones del usuario

- Inicializa un puntero al mando
- Lee los datos de los seis canales y los normaliza a ciertos rangos
- Publica los resultados cada 10 milisegundos

Entrada

Mando

Salida

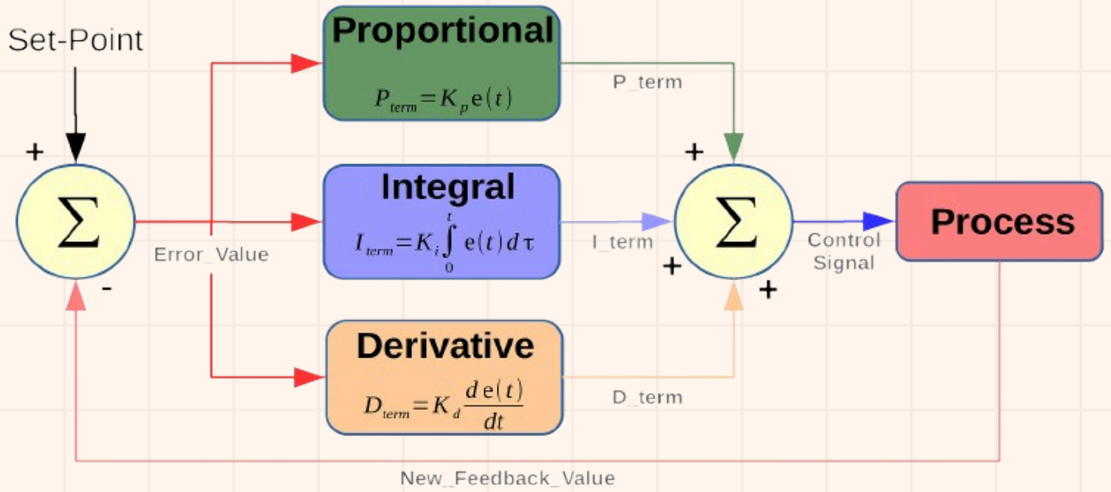
Tópico pos_deseada:

- Throttle: [0, 1]
- Roll, pitch: [−30, 30]
- Yaw: [−1, 1]

Tópico armado_motores

- Throttle: [0, 1]
- SwitchC: {−1, 0, 1}
- SwitchD: {−1, 1}

MÓDULO AUTOPILOT



Calcula las correcciones necesarias para los ángulos roll y pitch usando un algoritmo PID

- Lee los datos necesarios de los tópicos
- Ejecuta el algoritmo PID para cada eje
- Normaliza los resultados a cierto rango y los publica

Entrada

- Tópico pos_actual
- Tópico pos_deseada
- Tópico constantes_pid

Salida

Tópico pid:

- pid_roll, pid_pitch: [-1, 1]

MÓDULO CONSTANTES

Pide al usuario que introduzca las constantes que desea usar para el algoritmo del PID

- Pide por teclado las tres constantes para el eje roll
- Pide por teclado las tres constantes para el eje pitch
- Publica las constantes cada segundo

Entrada

Teclado

Salida

Tópico constantes_pid:

- Kp (roll/pitch)
- Kd (roll/pitch)
- Ki (roll/pitch)

MÓDULO MIXER

Mezcla los resultados de los algoritmos PID con los valores de aceleración y yaw

- Lee los datos necesarios de los tópicos
- Calcula el valor PWM resultante para cada motor
- Publica los resultados

Entrada

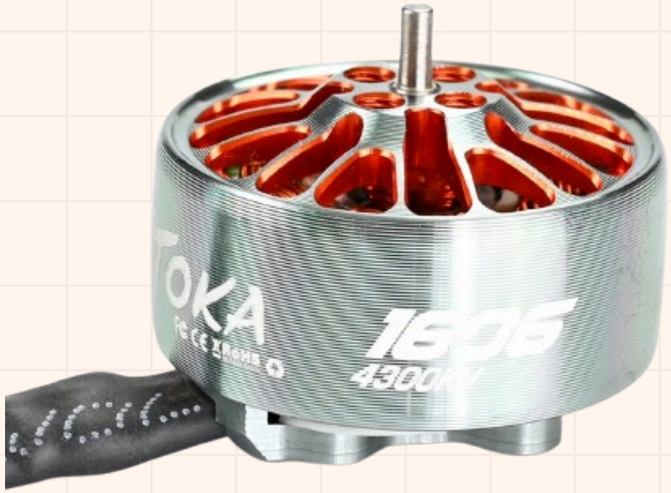
- Tópico pos_deseada
- Tópico pid

Salida

Tópico motores

- PWM_motor_X: [1150, 1924]

MÓDULO MOTOR



Realiza la secuencia de armador de los motores, y gestiona la potencia entregada a cada motor

- Lee los datos necesarios de los tópicos
- Si los motores están desarmados, los arma con la secuencia correcta
- Entrega la potencia correspondiente a cada motor

Entrada

- Tópico armado_motores
- Tópico motores

Salida

- Potencia a los motores

CALIBRADO
RTIMULIB

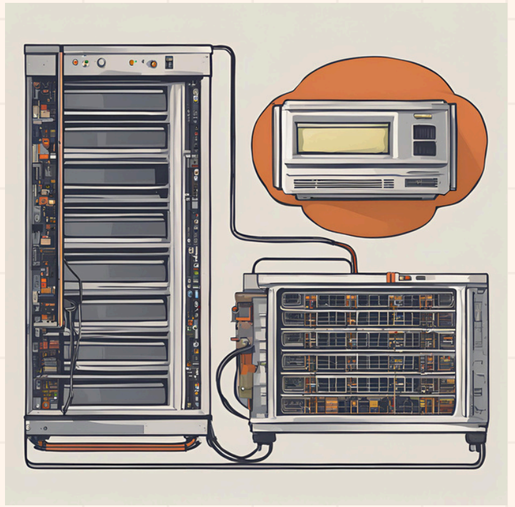
- Descargamos y compilamos la librería RTIMULib
- Ejecutamos el programa de calibrado y seguimos los pasos:
  - Calibrar magnetómetro con min/max
  - Calibrar magnetómetro con elipsoide
  - Calibrar acelerómetros
- Hacemos una modificación al fichero generado (RTIMULib.ini) para habilitar el filtrado de Kalman

ARMADO
MOTORES

- La información necesaria para realizar el armado/desarmado de los motores viaja en el tópico armado_motores:
  - Throttle
  - Switch C
  - Switch D
- El módulo motor está constantemente monitoreando este tópico

- Si los motores están desarmados:
  - Cuando ambos switches estén en posición hacia atrás, y el acelerador a cero, se realiza el armado
- Si los motores están armados:
  - Cuando ambos switches estén en posición hacia delante, se desarman los motores

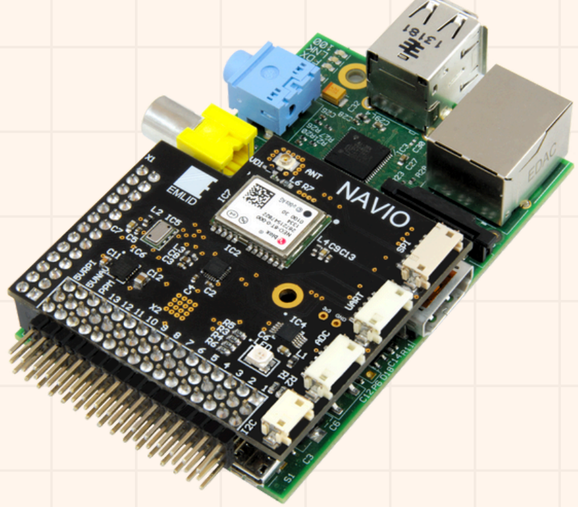
INTERFAZ WEB
FRONTEND



- El frontend está construido con HTML, CSS y JavaScript, y utiliza Flask para servir las páginas web y manejar las solicitudes HTTP. Las principales funcionalidades del frontend incluyen:
  - Interfaz de Usuario: **index.html** y **terminal.html**
  - Estilos y Scripts: **styles.css** y **scripts.js**
  - Rutas de Flask: **app.py** y **ssh_terminal.py**

- **app.py**: Maneja la interfaz web principal y las solicitudes para establecer la IP y obtener datos de los tópicos.
  - Librerías: Flask, Requests, ssh_terminal.
- **ssh_terminal.py**: Maneja la conexión SSH, ejecución de comandos y lectura de salida.
  - Librerías: Flask, Paramiko, Threading, Time, Re.

INTERFAZ WEB
BACKEND

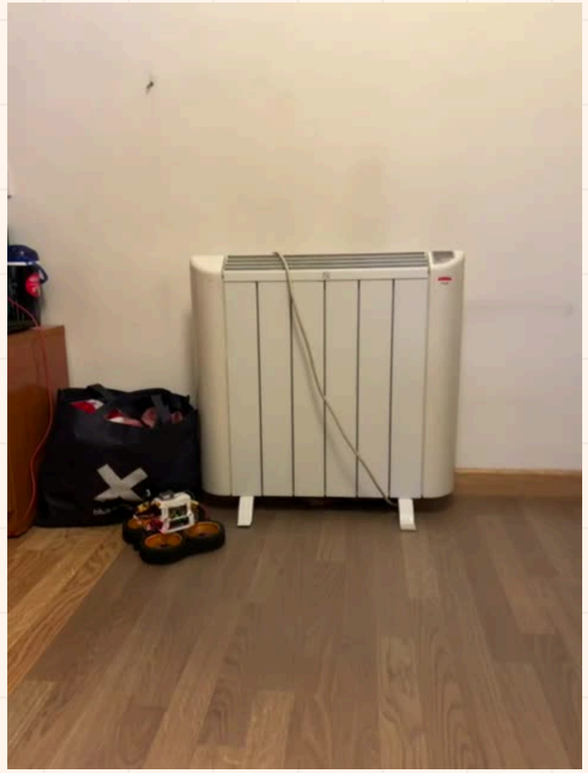


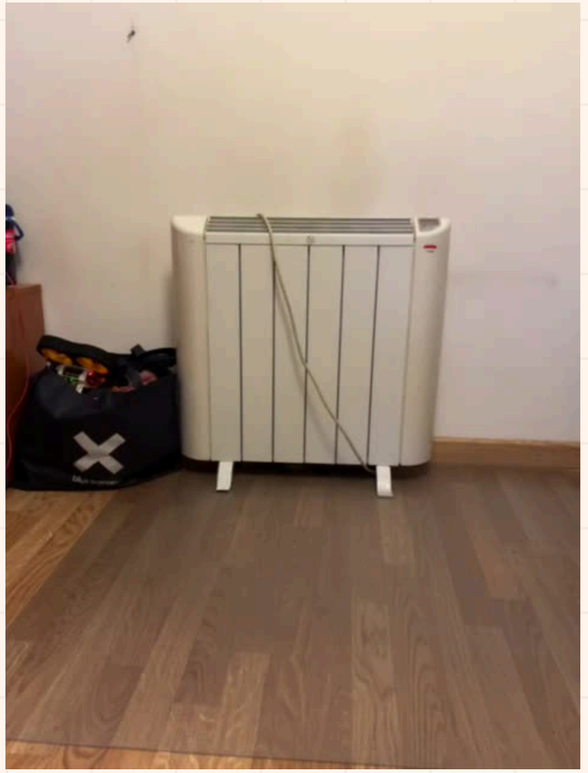
- El backend está construido con Flask y utiliza Zenoh para suscribirse a varios tópicos y recibir datos en tiempo real. Las principales funcionalidades del backend incluyen:
  - Aplicación Flask: **backend.py**
  - Suscripciones Zenoh: Se crean **hilos** para suscribirse a varios tópicos y actualizar las variables globales con los datos recibidos.
  - Manejo de Datos: Funciones de **callback** para cada tópico que decodifican el payload recibido y actualizan las variables globales correspondientes.

- **Suscripciones Zenoh:** Se declara un suscriptor en el tópico especificado. La función de callback correspondiente se ejecuta cada vez que se recibe un mensaje en el tópico suscrito.

PRUEBAS
CONSTANTES PID

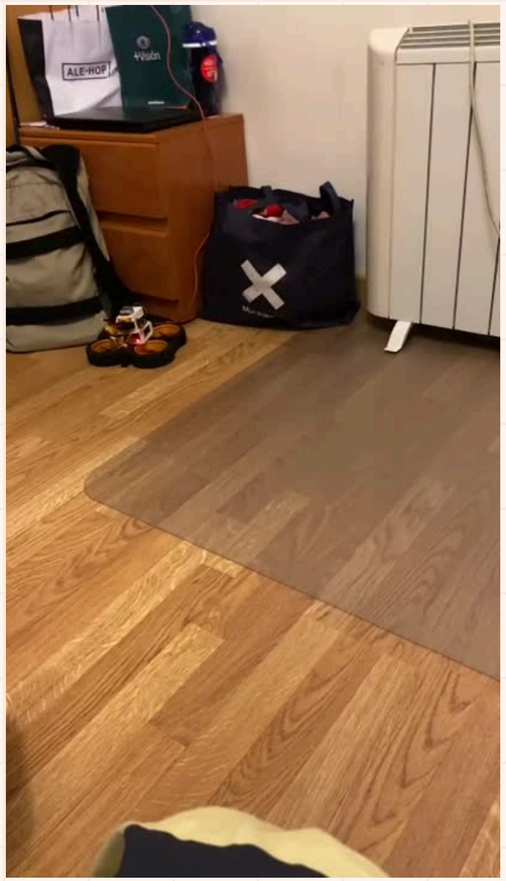
- Tras hacer diferentes pruebas variando las constantes del PID, hemos decidido que los mejores valores son :
    - kp = 1.5
    - ki = 0.125
    - kd = 0.15



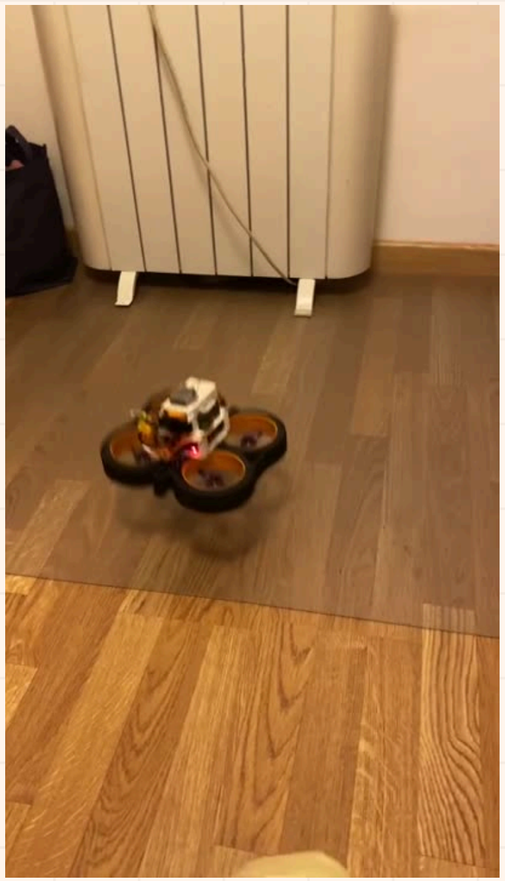


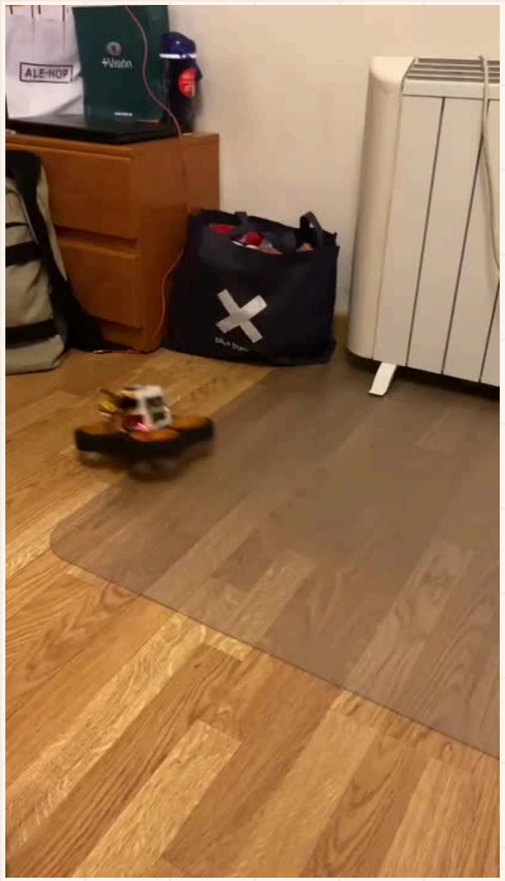
PRUEBAS
CONSTANTES PID

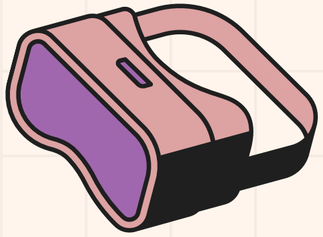
- kp = 2
- ki = 0.1
- kd = 0.15

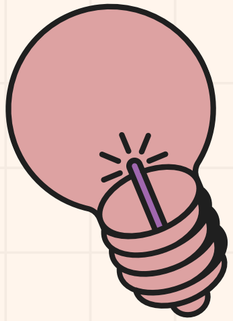


- kp = 2
- ki = 0.05
- kd = 0.15









DEMO