Sistema de Control para Cámara de Termovacío

Trabajo Práctico Final
Taller de Sistemas Embebidos

Franco Berni • Manuel Hirsch • Juan Ignacio Giachetti • Lautaro García Vitale

1er Cuatrimestre 2025

Facultad de Ingeniería – UBA

¿Qué es una cámara de termovacío?

Una **cámara de termovacío** es un sistema cerrado que permite **controlar simultáneamente la temperatura y la presión** en un entorno aislado.

Aplicaciones típicas

- Ensayos de componentes electrónicos
- Testing aeroespacial
- Pruebas de materiales

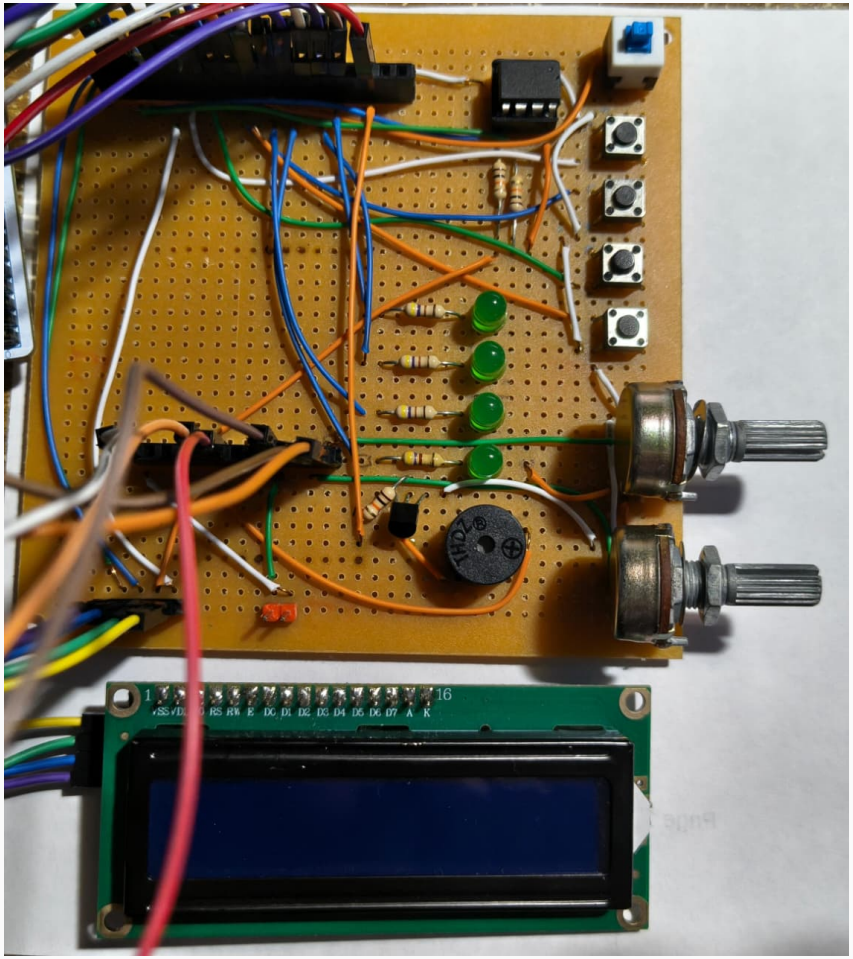
Producto Mínimo Viable (MVP)

El sistema desarrollado corresponde a un **producto mínimo viable** que replica el comportamiento lógico de una cámara de termovacío real, utilizando elementos simples para simular sensores y actuadores.

Sistema real	**Implementación en el MVP**
Sensor de temperatura	Potenciómetro leído por ADC
Sensor de presión	Potenciómetro leído por ADC
Calentador / refrigeración	LED indicador
Bomba de vacío	LED indicador
Sistema de alarma	Buzzer
Panel de control	LCD + botones / switches

Prototipo del sistema (MVP)

Implementación física del MVP utilizando una placa STM32 y periféricos para simular sensores, actuadores e interfaz de usuario.



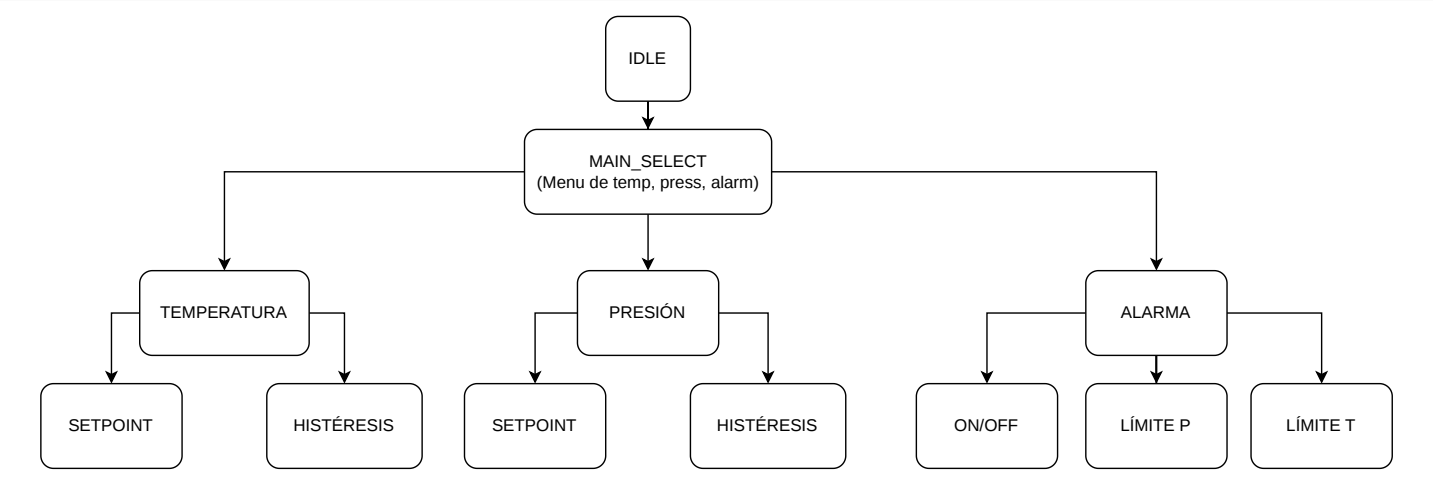
Statechart del sistema

Funcionamiento general

- En modo **NORMAL**, si llega un evento de encendido, avisa a los sistemas de presión y temperatura que controlen.
- En modo **MENÚ** se configuran los parámetros.
- Estando en modo **NORMAL**, si algún valor excede los límites de alarma, se entra en modo **ALARMA**.
- Se navega con el botón de Enter.

Implementación: `code/app/src/task_system.c`

Statechart del menú



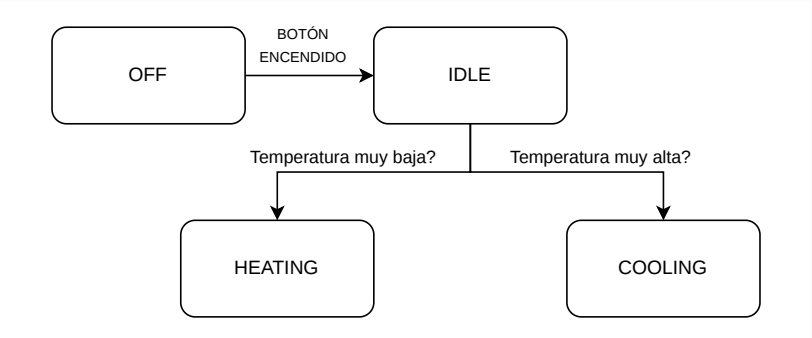
Funcionamiento del menú

- El usuario accede al **MENÚ** mediante los botones del sistema.
- Se pueden modificar los **setpoints de temperatura y presión** y las **bandas de histéresis**.
- Se puede configurar el uso de **alarmas**.
- Una vez confirmados los cambios, los parámetros se **guardan en memoria**.

Implementación: `code/app/src/task_menu.c`

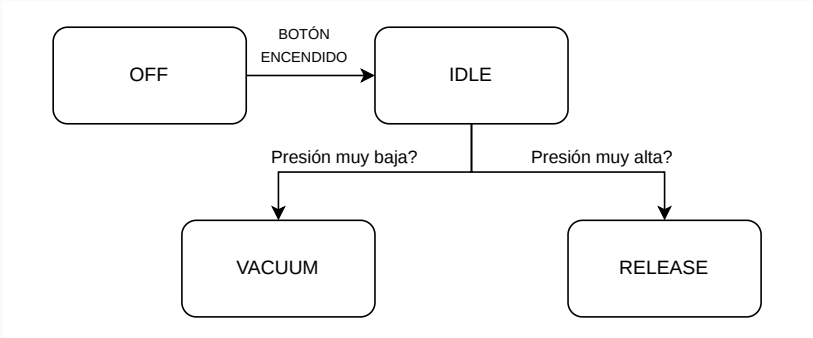
Control de temperatura y presión

Control de temperatura



Implementación: `code/app/src/task_temp.c`

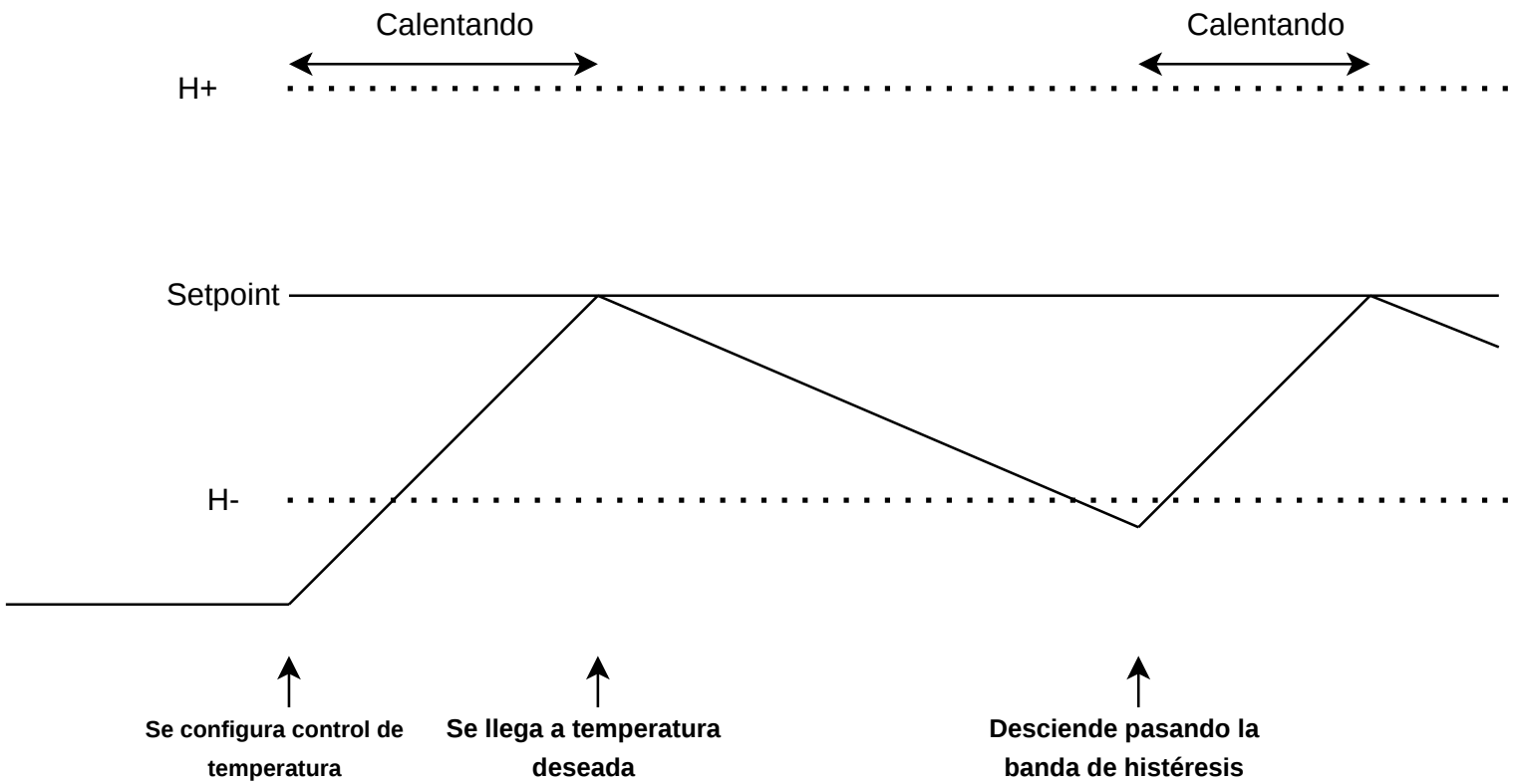
Control de presión



Implementación: `code/app/src/task_press.c`

- Ambos controles utilizan una lógica basada en **setpoints** y **bandas de histéresis**.
- El sistema monitorea continuamente los valores de temperatura y presión.
- Cuando una variable sale de la banda permitida, se activa el actuador correspondiente.

Ejemplo: línea de tiempo



Adquisición de señales analógicas

- El sistema mide las variables de **temperatura** y **presión** mediante dos entradas analógicas del microcontrolador.
- Se simulan utilizando **potenciómetros**, cuyas tensiones son convertidas por el **ADC del STM32**.
- Se convierten ambos canales en cada ciclo
- Los resultados de las conversiones se transfieren automáticamente a memoria mediante **DMA (Direct Memory Access)**.
- El uso de DMA permite almacenar las mediciones en un `adc_buffer` **sin intervención del CPU**, reduciendo la carga del procesador y permitiendo que se sigan ejecutando las tareas.
- `task_adc` lee este buffer y copia los valores a la estructura compartida `shared_data` para que el resto del sistema utilice las mediciones.

Implementación: `code/app/src/task_adc.c`

Escritura no bloqueante en EEPROM

Problema

- Se busca que cada ciclo dure menos de **1 ms**.
- Una escritura en EEPROM tarda aproximadamente **5 ms**.
- Una implementación bloqueante detendría el sistema durante varias iteraciones.

Solución

- Se utiliza **I2C con interrupciones** (`HAL_I2C_Mem_Write_IT`).
- La transmisión se realiza en segundo plano.
- Un **callback** indica cuándo finaliza la operación.

Implementación: `code/app/src/eeprom.c`

Actualización no bloqueante del display

Problema

- La comunicación con el display LCD se realiza por **I2C**.
- Escribir múltiples caracteres puede tomar mucho tiempo.
- Se tiene un problema similar al anterior.

Solución

- Se implementó una **cola de comandos** para el display.
- Las distintas tareas envían comandos a la cola.
- La tarea de display procesa los comandos **gradualmente en cada iteración**.

Implementación: `code/app/src/task_display.c`