Capítulo 00 · Introducción al curso

Conceptos clave

Arduino mide y actúa sobre el entorno. Raspberry Pi concentra los datos y presenta el estado del proceso.

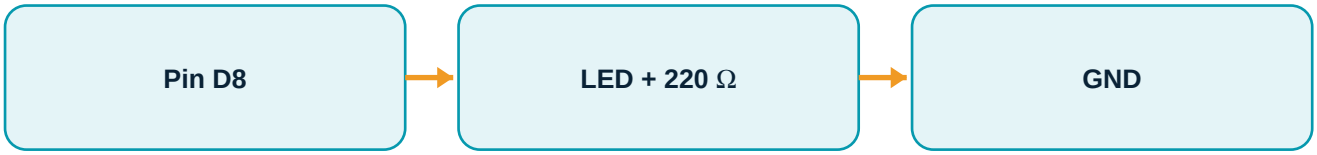


Internet de las Cosas con Arduino y Raspberry Pi

Capítulo 01 · Electricidad, protoboard y primer LED

Conceptos clave

Conecta el ánodo del LED a D8 y el cátodo a tierra a través de 220 Ω. Desconecta el USB antes de cambiar el cableado.



Ejemplo complementario

```
const byte LED = 8;
void setup() { pinMode(LED, OUTPUT); }
void loop() {
  digitalWrite(LED, HIGH);
  delay(1000);
  digitalWrite(LED, LOW);
  delay(1000);
}
```

Guía Paso a Paso de Práctica 01: ¡Hola Mundo Físico! Encendiendo tu Primera Luz LED

[!TIP]

Meta de esta práctica (15 a 25 minutos): Armar tu primer circuito real sin soldar sobre la protoboard, subir tu primer programa desde el computador al Arduino UNO y lograr que una luz LED parpadee al ritmo que tú elijas.

Paso 1: Prepara tus piezas sobre la mesa (2 minutos)

Antes de conectar nada, coloca frente a ti únicamente estos elementos:

1. Tu placa Arduino UNO y su cable USB.

2. Tu Protoboard (la tabla blanca llena de agujeritos).

3. Un LED Rojo (fíjate que tiene una patita más larga y otra más corta).

4. Una Resistencia de 220 Ohmios (colores: Rojo - Rojo - Marrón).

5. Dos cables jumper Macho-Macho (uno de color rojo o naranja, y otro negro o azul).

Paso 2: Pincha el LED y la Resistencia en la Protoboard (5 minutos)

1. Mira el centro de tu protoboard: verás números de fila (1, 2, 3... hasta 30 o 60).

2. Toma el LED Rojo e identifica sus dos patitas:

* Patita Larga (+ Ánodo): Es el lado positivo.

* Patita Corta (- Cátodo): Es el lado negativo.

3. Inserta la patita larga en el agujerito E10 (fila 10) y la patita corta en el agujerito de al lado E11 (fila 11).

* *¡Cuidado!* Nunca pongas las dos patitas del LED en el mismo número de fila, porque los 5 agujeritos de una misma fila están unidos por dentro.

4. Toma la resistencia de 220 Ohmios (no importa de qué lado la pongas, las resistencias no tienen positivo ni negativo):

* Pincha una patita de la resistencia en la fila 11 (por ejemplo, en el agujerito C11, compartiendo fila con la patita corta del LED).

* Pincha la otra patita de la resistencia en la línea larga azul (-) del borde de la protoboard.

Paso 3: Conecta los 2 cables hacia el Arduino UNO (3 minutos)

1. Cable de Energía (Rojo/Naranja):

* Conecta una punta en el Pin Digital 8 del Arduino UNO.

* Conecta la otra punta en la fila 10 de la protoboard (por ejemplo, en el agujerito A10, donde está la patita larga del LED).

2. Cable de Tierra o Regreso (Negro/Azul):

* Conecta una punta en cualquiera de los agujeritos que dicen GND en tu Arduino UNO.

* Conecta la otra punta en cualquier agujerito de la línea larga azul (-) de tu protoboard.

Paso 4: Sube tu primer programa desde Arduino IDE (5 minutos)

1. Conecta el cable USB desde tu Arduino UNO a un puerto USB de tu computador. Verás que se enciende una lucecita verde en la placa.

2. Abre el programa Arduino IDE en tu computador.

3. En la barra superior, selecciona tu placa (Arduino UNO) y el puerto (COM3, COM4, etc.).

4. Abre el archivo 01_mi_primer_led_parpadeante.ino incluido en la carpeta codigo_fuente/.

5. Haz clic en el botón redondo con una flecha hacia la derecha (Subir / Upload).

6. Espera 5 segundos... ¡Felicitaciones! Tu LED rojo ya está encendiendo y apagando cada 1 segundo.

Paso 5: Retos divertidos para experimentar (5 minutos)

Reto · Qué cambiar en el código · Qué ocurrirá en tu protoboard

:--- · :--- · :---

Reto 1: Latido Rápido · Cambia ambos delay(1000); por delay(200); y vuelve a presionar la flecha Subir. · Tu LED parpadeará 5 veces por segundo como una baliza de emergencia.

Reto 2: Destello de Faro · Pon delay(100); después de HIGH y delay(1500); después de LOW. · El LED dará un destello cortito seguido de una pausa larga, igual que un faro marino.

Reto 3: Señal S.O.S. · Abre y sube el archivo 03_faro_destello_doble_sos.ino. · Tu LED enviará la señal internacional de auxilio: 3 cortos, 3 largos y 3 cortos.

Solución Rápida de Problemas (¿No encendió a la primera?)

* El programa subió bien pero el LED no prende: Saca el LED, gíralo media vuelta para intercambiar sus patitas (asegurando que la patita larga esté en la fila del cable que viene del Pin 8) y vuelve a pincharlo.

* Las patitas no hacen buen contacto: Asegúrate de empujar suavemente las patitas del LED, la resistencia y los cables hasta el fondo de los agujeritos de la protoboard.

Listado de Materiales (BOM) — Módulo 01: ¡Hola Mundo Físico!

[!NOTE]

Todos los componentes de este módulo vienen incluidos en cualquier Kit Básico de Iniciación Arduino. No necesitas soldador, ni pinzas especiales, ni fuentes de alimentación externas.

Tabla de Componentes del Módulo 01

Ítem · Componente · Cantidad · ¿Cómo reconocerlo en tu caja? · ¿Para qué sirve en esta práctica?

:---: · :--- · :---: · :--- · :---

1 · Placa Arduino UNO R3 · 1 · Placa azul con un conector USB plateado y dos filas de agujeritos negros a los lados. · Es el cerebro reflejo que enviará los 5 Voltios para encender y apagar nuestra luz.

2 · Cable USB para Arduino (Tipo A a Tipo B) · 1 · Cable grueso (suele ser azul o negro) con punta cuadrada de impresora en un extremo. · Conecta el Arduino al computador para darle energía y enviarle nuestro programa.

3 · Protoboard (Placa de pruebas de 830 o 400 puntos) · 1 · Tabla plástica blanca llena de agujeritos con líneas rojas (+) y azules (-) en los bordes. · Permite unir componentes y cables enchufándolos con los dedos sin soldar.

4 · LED de 5 mm (Color Rojo) · 1 · Pequeña cápsula plástica roja transparente con dos patitas metálicas (una larga y una corta). · Es nuestra primera luz indicadora controlada por código.

5 · Resistencia de 220 Ohmios · 1 · Pequeño cilindrito beige o celeste con alambres a los lados y bandas Rojo - Rojo - Marrón (o Rojo - Rojo - Negro - Negro). · Es el "guardaespaldas" del LED: frena un poquito la corriente para que el LED nunca se queme.

6 · Cables Jumper Macho-Macho · 2 · Cablecitos flexibles de colores con una aguja metálica firme en cada punta. · Unen el Pin Digital 8 y el Pin GND del Arduino con las filas de la protoboard.

Guía Visual para Reconocer tu Resistencia de 220 Ohmios

No necesitas medirla con ningún aparato. Solo mira sus bandas de colores con buena luz:

Tipo de Resistencia en tu Kit · Bandas de Colores en Orden · Valor

:--- · :--- · :---:

Versión de 4 bandas (cuerpo beige) · Rojo — Rojo — Marrón — Dorado · 220 Ohmios

Versión de 5 bandas (cuerpo celeste) · Rojo — Rojo — Negro — Negro — Marrón · 220 Ohmios

[!TIP]

¿Qué pasa si en tu kit viene de 330 Ohmios (Naranja - Naranja - Marrón)? ¡Puedes usarla exactamente igual! Cualquier resistencia entre 220 Ohmios y 1000 Ohmios (1k) protegerá perfectamente a tu LED.

Manual Explicado desde Cero — Módulo 01: Tus Placas, la Protoboard y tu Primera Luz

[!NOTE]

Bienvenido/a: Este manual está escrito para personas que comienzan desde cero absoluto. Aquí explicamos cada concepto con ejemplos de la vida diaria, sin fórmulas matemáticas ni tecnicismos complicados.

1. ¿Qué es el Internet de las Cosas (IoT)?

La sigla IoT viene del inglés *Internet of Things* (Internet de las Cosas). En palabras muy simples, consiste en darle a un objeto cotidiano de tu casa tres habilidades:

1. Sentir: Medir algo del mundo real con un sensor (por ejemplo, si hace calor, si oscureció o si la tierra de una maceta está seca).

2. Pensar y Actuar: Tomar una decisión automática en una placa controladora (por ejemplo, encender un ventilador o una luz).

3. Avisar: Mostrar esa información en la pantalla de tu teléfono celular o enviarte un mensaje.

2. Arduino UNO vs. Raspberry Pi: ¿Quién es quién?

En este curso aprenderás a usar las dos placas más famosas del mundo. Piénsalo con una analogía del cuerpo humano:

Característica · Arduino UNO (Placa Azul) · Raspberry Pi (Placa Verde)

:--- · :--- · :---

Analogía humana · Los reflejos rápidos de tu mano · Un computador completo en miniatura

¿Qué hace mejor? · Conectar cables directamente a luces, botones, sensores y motores, respondiendo al instante. · Conectarse al Wi-Fi, mostrar un escritorio con ventanas, guardar historiales y crear paneles para el celular.

¿Tiene sistema operativo? · No. Ejecuta directamente un único programa tuyo una y otra vez sin distraerse. · Sí. Funciona con su propio sistema con escritorio, navegador web e íconos, igual que un PC.

¿Cuándo la usamos? · Módulos 01 al 04 (y en equipo en los Módulos 06 al 08). · Módulos 05 al 08.

3. La Analogía de la Manguera de Agua (Voltaje, Corriente y Resistencia)

Para entender cómo viaja la electricidad por tus cables sin usar ni una sola ecuación matemática, imagina una manguera de jardín conectada a una canilla (llave de agua):

* El Voltaje (V) es la presión del agua: Nuestro Arduino UNO trabaja con apenas 5 Voltios. Es una presión tan bajita y segura como la de un cargador de celular: puedes tocar los cables y la placa con tus manos con total tranquilidad.

* La Corriente (I) es la cantidad de agua que sale: Si abres una manguera gigante a máxima potencia directo sobre una flor pequeña, el chorro la romperá. De igual modo, si conectas un LED directo a 5 Voltios sin freno, pasará demasiada corriente de golpe y se quemará.

* La Resistencia (R) es un estrechamiento en la manguera: Al poner una resistencia de 220 Ohmios en el camino del LED, pisamos suavemente la manguera para que pase solo el hilito de corriente que el LED necesita para brillar feliz durante años.

4. El Secreto Interior de la Protoboard

La protoboard parece mágica hasta que descubres cómo están unidas las chapitas metálicas escondidas debajo del plástico blanco:

Zona de la Protoboard · ¿Cómo están conectados sus agujeritos por dentro? · ¿Para qué la usamos?

:--- · :--- · :---

Franja larga Roja (+) del borde · Todos los agujeritos de esa línea larga están unidos de punta a punta. · Autopista para llevar los 5 Voltios (5V) a todo el tablero.

Franja larga Azul (-) del borde · Todos los agujeritos de esa línea larga están unidos de punta a punta. · Autopista para llevar la Tierra (GND) a todo el tablero.

Filas cortas del centro (A-B-C-D-E) · Los 5 agujeritos de un mismo número de fila están unidos entre sí. · Para unir la patita de un componente con un cable enchufándolos en el mismo número de fila.

Canal central hundido · Separa el lado izquierdo (A-E) del lado derecho (F-J). · Evita que ambos lados se toquen entre sí.

5. Cómo funcionan los dos bloques de todo programa Arduino (`setup` y `loop`)

Cuando abres Arduino IDE, siempre verás dos bloques:

1. void setup() (Preparación única): Es como poner la mesa antes de almorzar. Se ejecuta una sola vez cuando enchufas el Arduino. Ahí usamos pinMode(8, OUTPUT); para avisarle que el Pin 8 sacará electricidad hacia el LED.

2. void loop() (Bucle infinito): Es como el latido del corazón. Todo lo que pongas dentro de sus llaves { ... } se repetirá de arriba hacia abajo para siempre mientras el Arduino tenga energía:

* digitalWrite(8, HIGH); $\rightarrow$ Sube el interruptor del Pin 8 a 5 Voltios (el LED se enciende).

* delay(1000); $\rightarrow$ Espera 1000 milisegundos (1 segundo).

* digitalWrite(8, LOW); $\rightarrow$ Baja el interruptor del Pin 8 a 0 Voltios (el LED se apaga).

* delay(1000); $\rightarrow$ Espera 1 segundo antes de volver automáticamente a la primera línea.

Plano y Conexiones Cable a Cable — Módulo 01: ¡Hola Mundo Físico!

[!IMPORTANT]

Regla de Oro para conectar tu primer LED: La electricidad siempre sale del Pin Digital 8 del Arduino, entra por la patita larga (+) del LED, sale por la patita corta (-), atraviesa la resistencia protectora de 220 Ohmios y regresa al pin GND (Tierra) del Arduino para cerrar el circuito.

1. Tabla de Conexiones Pin a Pin (Guía Rápida sin Errores)

Paso · De dónde sale el cable / patita · A dónde se conecta en la Protoboard / Arduino · Color de cable sugerido · Explicación en palabras simples

:---: · :--- · :--- · :---: · :---

1 · Pin Digital 8 del Arduino UNO · Agujerito A10 de la protoboard (Fila 10) · Rojo o Naranja · Lleva los 5 Voltios desde el Arduino cuando el código dice HIGH.

2 · Patita Larga (+ Ánodo) del LED Rojo · Agujerito E10 de la protoboard (Fila 10) · *(Patita del LED)* · Al estar en la misma Fila 10 que el cable del Pin 8, recibe la energía positiva.

3 · Patita Corta (- Cátodo) del LED Rojo · Agujerito E11 de la protoboard (Fila 11) · *(Patita del LED)* · Sale hacia la fila vecina (Fila 11) para encontrarse con su resistencia.

4 · Punta 1 de la Resistencia (220 Ohmios) · Agujerito C11 de la protoboard (Fila 11) · *(Resistencia)* · Recibe la corriente que sale de la patita corta del LED y la frena a un nivel seguro.

5 · Punta 2 de la Resistencia (220 Ohmios) · Cualquier agujerito del Riel Azul (-) · *(Resistencia)* · Entrega la corriente ya frenada a la autopista negativa de la protoboard.

6 · Riel Azul (-) de la protoboard · Pin GND del Arduino UNO · Negro o Azul · Cierra el camino de regreso hacia Tierra (GND) en el Arduino.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad

Observa en el siguiente plano exactamente en qué agujeritos va conectado cada cable, el LED rojo y su resistencia protectora de 220 Ohmios:

![Plano 1: Montaje en Protoboard Estilo Tinkercad](planos_circuitos/plano_protoboard_modulo_01.png)

3. Plano 2: Guía Visual de Funcionamiento

En este diagrama visual puedes ver cómo viaja la orden desde las instrucciones setup() y loop() de tu código hasta encender y apagar físicamente la luz LED:

4. Lista de Chequeo Visual Antes de Enchufar el USB

- [x] ¿La patita larga del LED está en la misma fila numerada que el cable que viene del Pin 8?

- [x] ¿Las dos patitas del LED están en filas distintas (por ejemplo, Fila 10 y Fila 11)?

- [x] ¿La resistencia de 220 Ohmios une la fila de la patita corta del LED con la línea azul (-)?

- [x] ¿Hay un cable uniendo la línea azul (-) de la protoboard con un pin GND del Arduino?

Capítulo 02 · Entradas y salidas: semáforo, pulsador y buzzer

Conceptos clave

Con INPUT_PULLUP, el pulsador abierto se lee HIGH y el pulsado se lee LOW. Separa la lectura de la entrada de la secuencia del semáforo.



Ejemplo complementario

```
const byte BOTON = 2;
void setup() {
  pinMode(BOTON, INPUT_PULLUP);
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, digitalRead(BOTON) == LOW);
}
```

Guía Paso a Paso de Práctica 02: Semáforo, Botón Pulsador y Sonidos con Buzzer

[!TIP]

Meta de esta práctica (25 a 35 minutos): Construir un semáforo completo de tres colores (Verde, Amarillo y Rojo), conectarle un botón pulsador usando solo dos cables con el truco INPUT_PULLUP y agregar un zumbador (Buzzer) para crear un cruce peatonal con aviso sonoro.

Paso 1: Prepara el Riel Negativo (`GND`) compartido (3 minutos)

Como ahora conectaremos varios componentes (3 LEDs, 1 botón y 1 buzzer), todos necesitarán regresar a Tierra (GND).

1. Toma un cable jumper negro o azul.

2. Conecta una punta en un pin GND de tu Arduino UNO.

3. Conecta la otra punta en la línea larga azul (-) de tu protoboard.

* *¡Listo!* Ahora toda esa fila azul larga de tu protoboard es Tierra (GND).

Paso 2: Arma las 3 luces del Semáforo (8 minutos)

1. Pincha los 3 LEDs (Verde, Amarillo y Rojo) dejando espacio entre ellos en tu protoboard:

* LED Verde: Patita larga (+) en la fila 5, patita corta (-) en la fila 6.

* LED Amarillo: Patita larga (+) en la fila 10, patita corta (-) en la fila 11.

* LED Rojo: Patita larga (+) en la fila 15, patita corta (-) en la fila 16.

2. Coloca una resistencia de 220 Ohmios para cada LED, desde la fila de su patita corta (6, 11 y 16) hacia la línea larga azul (-).

3. Conecta los 3 cables desde el Arduino hacia las patitas largas:

* Pin Digital 8 $\rightarrow$ Fila 5 (Patita larga del LED Verde).

* Pin Digital 9 $\rightarrow$ Fila 10 (Patita larga del LED Amarillo).

* Pin Digital 10 $\rightarrow$ Fila 15 (Patita larga del LED Rojo).

4. Sube el programa 01_semaforo_tres_colores.ino y observa cómo tu semáforo cobra vida.

Paso 3: Agrega el Botón Pulsador con Solo 2 Cables (7 minutos)

1. Pincha el botón pulsador en el centro de la protoboard (por ejemplo, en las filas 20 y 22, cruzando el canal central).

2. Conecta un cable desde el Pin Digital 2 del Arduino hacia la fila 20 (una patita del botón).

3. Conecta otro cable desde la fila 22 (la otra patita del mismo lado del botón) hacia la línea larga azul (- GND).

4. Sube el programa 02_boton_magico_input_pullup.ino y presiona el botón con el dedo: ¡verás cómo cambia la luz al instante!

Paso 4: Agrega el Buzzer y prueba el Cruce Peatonal (7 minutos)

1. Mira tu Buzzer (cilindro negro): tiene un signo + arriba (o una patita más larga).

2. Pincha la patita positiva (+) en la fila 26 y la patita negativa (-) en la fila 27.

3. Lleva un cable desde el Pin Digital 11 del Arduino hacia la fila 26 (+ del Buzzer).

4. Lleva un cable desde la fila 27 (- del Buzzer) hacia la línea larga azul (- GND).

5. Sube el programa 03_cruce_peatonal_con_buzzer.ino, presiona el botón y escucha la secuencia completa del cruce peatonal inteligente.

Listado de Materiales (BOM) — Módulo 02: Semáforos, Botones y Sonidos

[!NOTE]

En este segundo módulo reutilizamos el Arduino UNO, la protoboard y el LED rojo del Módulo 01, y sumamos dos LEDs más (verde y amarillo), un botón pulsador y un zumbador (Buzzer).

Tabla de Componentes del Módulo 02

Ítem · Componente · Cantidad · ¿Cómo reconocerlo en tu caja? · ¿Para qué sirve en esta práctica?

:---: · :--- · :---: · :--- · :---

1 · Arduino UNO R3 + Cable USB · 1 · Placa controladora azul con su cable USB. · Controla las 3 luces, lee el botón en el Pin 2 y genera el sonido en el Pin 11.

2 · Protoboard de 830 o 400 puntos · 1 · Tabla blanca de conexiones sin soldadura. · Nos permite compartir el riel azul negativo (GND) entre todos los componentes.

3 · LEDs de 5 mm (1 Verde, 1 Amarillo y 1 Rojo) · 3 · Tres lucecitas plásticas de los colores de un semáforo. · Representan el semáforo vehicular de calle (Avanzar, Precaución y Detenerse).

4 · Resistencias de 220 Ohmios · 3 · Bandas Rojo - Rojo - Marrón (una para cada LED). · Protegen a cada uno de los tres LEDs para que brillen parejo y seguros.

5 · Botón Pulsador Táctil (Pushbutton) · 1 · Cuadradito pequeño con un botón redondo en el centro y patitas metálicas. · Permite al usuario tocar con el dedo para pedir el cruce peatonal.

6 · Buzzer (Zumbador Activo o Pasivo de 5V) · 1 · Pequeño tamborcito cilíndrico negro con un orificio en la tapa superior y un signo +. · Emite pitidos audibles de confirmación y aviso de paso peatonal seguro.

7 · Cables Jumper Macho-Macho · 8 · Cablecitos de distintos colores con puntas metálicas. · Unen los pines 2, 8, 9, 10, 11 y GND con la protoboard.

¿Necesito una resistencia extra para el botón pulsador?

¡No! Gracias a que activamos por código el modo interno INPUT_PULLUP del Arduino UNO (pinMode(2, INPUT_PULLUP);), el propio chip del Arduino conecta internamente su resistencia protectora. Así te ahorras piezas y cables en la protoboard.

Manual Explicado desde Cero — Módulo 02: Semáforos, Botones (`INPUT_PULLUP`) y Sonidos

[!NOTE]

En este módulo aprenderás cómo darle capacidad de decisión a tu Arduino para que reaccione cuando presionas un botón y coordine varias luces y sonidos a la vez.

1. Variables con Nombres Claros: Poniéndole Etiquetas a los Pines

Cuando conectamos varios componentes, recordar de memoria qué número de pin corresponde a cada luz es incómodo. Por eso usamos variables:

```
int ledVerde    = 8;
int ledAmarillo = 9;
int ledRojo     = 10;
```

* La palabra int significa *número entero* (sin decimales).

* Piensa en una variable como pegarle una etiqueta con nombre a un cajón: en vez de recordar "el cajón número 8", simplemente escribes ledVerde en todo tu programa y el Arduino reemplaza esa palabra por el número 8 automáticamente.

2. El Secreto de `INPUT_PULLUP`: Cómo Conectar un Botón sin Enredarse

Un botón pulsador es simplemente un puentecito metálico: mientras no lo tocas, sus dos patitas están separadas; cuando lo presionas con el dedo, sus dos patitas se tocan.

En este curso usamos el modo inteligente interno del Arduino llamado INPUT_PULLUP:

Estado de tu Dedo · Qué pasa en el Pin 2 del Arduino · Qué devuelve digitalRead(2)

:--- · :--- · :---:

Botón SUELTO (sin tocar) · La resistencia interna del Arduino mantiene el pin levantado a 5V. · HIGH

Botón PRESIONADO (dedo apretando) · El botón une el Pin 2 directamente con Tierra (GND = 0V). · LOW

Por eso en el código siempre preguntamos: "¿El botón está en LOW?" para saber si alguien lo está presionando.

3. Cómo Toma Decisiones el Arduino: `if` (Si...) y `else` (Si no...)

Para que un proyecto sea inteligente debe poder decidir qué hacer según lo que ocurra afuera. En programación usamos la estructura if / else:

```
if (estadoBoton == LOW) {
  // Lo que está aquí adentro se ejecuta SOLO SI el botón está presionado
  digitalWrite(ledVerde, HIGH);
} else {
  // Lo que está aquí adentro se ejecuta SI NO está presionado (botón suelto)
  digitalWrite(ledVerde, LOW);
}
```

[!IMPORTANT]

Detalle importante (= vs ==):

* Un solo signo igual (=) sirve para guardar un valor: int ledVerde = 8;

* Dos signos iguales juntos (==) sirven para preguntar / comparar: if (estadoBoton == LOW)

4. Cómo Funciona el Sonido con `tone()` en el Buzzer

Para emitir sonidos por el Buzzer conectado al Pin 11, usamos la orden tone(pin, nota, duracion):

```
tone(11, 1000, 250);
```

* 11: Es el número del Pin Digital donde está conectado el positivo (+) del Buzzer.

* 1000: Es el tono del sonido en Hercios (Hz). Un número como 500 suena más grave, y un número como 1500 o 2000 suena más agudo.

* 250: Es cuántos milisegundos durará el pitido antes de apagarse solo (250 ms = un cuarto de segundo).

Plano y Conexiones Cable a Cable — Módulo 02: Semáforo, Botón y Buzzer

[!IMPORTANT]

Consejo de orden en tu protoboard: Conecta primero el cable negro desde GND del Arduino hacia el riel largo azul (-) de la protoboard. Así podrás conectar allí el negativo de los 3 LEDs, del botón y del buzzer sin cruzar cables largos.

1. Tabla de Conexiones Pin a Pin Completa

Componente · Patita del Componente · A dónde va en el Arduino UNO · A dónde va el otro extremo · Función en el Circuito

:--- · :--- · :--- · :--- · :---

Riel Negativo Común · Línea larga azul (-) · Pin GND · *(Recorre toda la protoboard)* · Tierra común para todos los componentes.

LED Verde · Patita Larga (+) / Corta (-) · Pin Digital 8 · Resistencia 220 $\rightarrow$ Riel Azul (-) · Luz de avance libre del semáforo.

LED Amarillo · Patita Larga (+) / Corta (-) · Pin Digital 9 · Resistencia 220 $\rightarrow$ Riel Azul (-) · Luz de precaución antes de cambiar a rojo.

LED Rojo · Patita Larga (+) / Corta (-) · Pin Digital 10 · Resistencia 220 $\rightarrow$ Riel Azul (-) · Luz de detención de autos y cruce peatonal.

Botón Pulsador · Patita 1 / Patita 2 · Pin Digital 2 · Directo al Riel Azul (-) · Detecta cuando presionas con el dedo (INPUT_PULLUP).

Buzzer (Zumbador) · Patita Positiva (+) / Negativa (-) · Pin Digital 11 · Directo al Riel Azul (-) · Emite los pitidos sonoros del cruce peatonal.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad

En el siguiente diagrama de protoboard puedes ver la ubicación exacta de los 3 LEDs con sus resistencias de 220 Ohmios, el botón pulsador en el Pin 2 y el Buzzer en el Pin 11:

3. Plano 2: Guía Visual de Funcionamiento

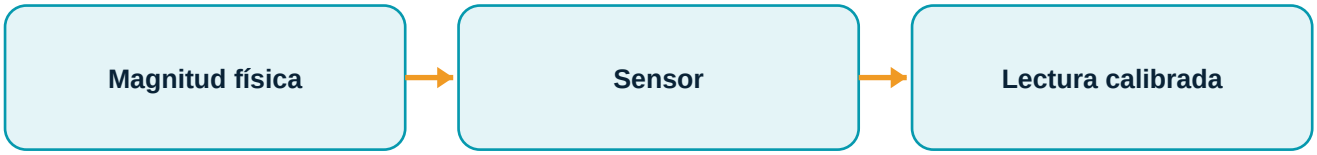
Observa en este esquema el flujo de decisión lógica (if / else) desde que tu dedo presiona el botón hasta que el semáforo cambia de color y emite los pitidos de paso seguro:

Internet de las Cosas con Arduino y Raspberry Pi

Capítulo 03 · Sensores LDR, DHT y ultrasonido

Conceptos clave

Usa un divisor para la LDR y convierte el tiempo de ida y vuelta del ultrasonido a distancia: d = v × t / 2. Registra unidades y lecturas inválidas.



Ejemplo complementario

```
void setup() { Serial.begin(9600); }
void loop() { Serial.println(analogRead(A0)); delay(200); }
```

Guía Paso a Paso de Práctica 03: Dándole Sentidos al Arduino (Luz, Clima y Distancia)

[!TIP]

Meta de esta práctica (30 minutos): Descubrir el Monitor Serie para conversar en vivo con el Arduino en la pantalla de tu PC, leer una perilla giratoria, armar una lámpara nocturna automática y medir la temperatura, humedad y distancia en centímetros.

Paso 1: Abre la Ventana Mágica del Monitor Serie (5 minutos)

1. Conecta las 3 patitas del potenciómetro (perilla) en tu protoboard:

* Patita de un extremo $\rightarrow$ Riel Rojo (5V).

* Patita del centro $\rightarrow$ Pin Analógico A0 del Arduino UNO.

* Patita del otro extremo $\rightarrow$ Riel Azul (GND).

2. Sube el archivo 01_perilla_potenciometro_monitor_serie.ino a tu Arduino.

3. En la esquina superior derecha de Arduino IDE, haz clic en el ícono de la Lupita (Monitor Serie).

4. Asegúrate de que abajo a la derecha de esa ventana diga 9600 baudios.

5. Gira la perilla con los dedos: ¡verás cómo el número sube de 0 a 1023 en vivo en tu pantalla!

Paso 2: Construye tu Lámpara Automática con Sensor de Luz `LDR` (8 minutos)

1. Si tu sensor de luz LDR viene en módulo de 3 pines:

* Conecta VCC a 5V, GND a GND y OUT (o AO) al Pin Analógico A1.

* *(Si tienes el sensor LDR suelto de 2 patitas: conecta una patita a 5V, la otra patita al Pin A1 y pon una resistencia de 10k entre A1 y GND).*

2. Deja tu LED con resistencia de 220 Ohmios conectado en el Pin Digital 8.

3. Sube el programa 02_lampara_nocturna_automatica_ldr.ino.

4. Cubre el sensor LDR con tu mano para hacerle sombra: ¡el LED se encenderá solo al instante! Cuando quites la mano, se apagará.

Paso 3: Instala la Librería `DHT` en 2 Clics (5 minutos)

Para leer el sensor de temperatura y humedad sin complicaciones:

1. En Arduino IDE, haz clic en el ícono de Libros (Gestor de Bibliotecas) en la barra izquierda (o menú *Herramientas $\rightarrow$ Administrar Bibliotecas*).

2. En el buscador escribe: DHT sensor library.

3. Busca la que dice by Adafruit y haz clic en el botón Instalar (si te pregunta si deseas instalar sus componentes compañeros, haz clic en *Instalar todo*).

Paso 4: Mide Temperatura, Humedad y Distancia Ultrasónica (12 minutos)

1. Conecta el Sensor DHT11 o DHT22:

* Pin + (VCC) $\rightarrow$ 5V | Pin - (GND) $\rightarrow$ GND | Pin central (OUT/DATA) $\rightarrow$ Pin Digital 4.

2. Conecta el Sensor Ultrasónico HC-SR04:

* Pin VCC $\rightarrow$ 5V | Pin GND $\rightarrow$ GND | Pin Trig $\rightarrow$ Pin Digital 5 | Pin Echo $\rightarrow$ Pin Digital 6.

3. Sube el archivo 03_clima_dht_y_distancia_ultrasonica.ino y abre el Monitor Serie.

4. Sopla suavemente sobre el sensor DHT y acerca tu mano al sensor HC-SR04: verás cambiar los grados Celsius, el porcentaje de humedad y los centímetros en tiempo real.

Listado de Materiales (BOM) — Módulo 03: Luz, Temperatura y Distancia

[!NOTE]

En este módulo incorporamos los sentidos de nuestro kit de iniciación: una perilla giratoria, un sensor de luz ambiental, un sensor de temperatura/humedad y un medidor de distancia por ultrasonido.

Tabla de Componentes del Módulo 03

Ítem · Componente · Cantidad · ¿Cómo reconocerlo en tu caja? · ¿Para qué sirve en esta práctica?

:---: · :--- · :---: · :--- · :---

1 · Arduino UNO R3 + Protoboard · 1 c/u · Tu placa controladora y tabla de pruebas. · Recibe las señales de todos los sensores y las envía por USB a la pantalla del PC.

2 · Potenciómetro de 10 kOhmios · 1 · Pequeña perilla redonda giratoria con 3 patitas metálicas en la base. · Permite variar manualmente el voltaje de 0V a 5V para ver números de 0 a 1023.

3 · Fotorresistencia (LDR) o Módulo LDR · 1 · Cabecita plana con un dibujo en zigzag rojo/naranja sensible a la luz. · Detecta si es de día (mucha luz) o de noche (oscuridad) para encender la lámpara sola.

4 · Sensor de Clima DHT11 (celeste) o DHT22 (blanco) · 1 · Cajita con rejilla plástica montada en una pequeña placa de 3 pines (+, OUT, -). · Mide la temperatura ambiente en grados Celsius (°C) y la humedad del aire (%).

5 · Sensor Ultrasónico HC-SR04 · 1 · Placa con dos cilindros metálicos con rejilla que parecen dos "ojitos" (T y R). · Mide a cuántos centímetros (cm) de distancia se encuentra un objeto o tu mano.

6 · LED de 5 mm + Resistencia de 220 Ohmios · 1 c/u · Luz indicadora y su resistencia Rojo-Rojo-Marrón. · Funciona como farola automática nocturna y alerta de proximidad.

7 · Cables Jumper Macho-Macho y Macho-Hembra · 12 · Cablecitos de colores para conectar sensores a la protoboard. · Llevan alimentación (5V y GND) y señales hacia los pines A0, A1, 4, 5, 6 y 8.

Manual Explicado desde Cero — Módulo 03: El Monitor Serie y los Sensores

[!NOTE]

Hasta ahora nuestro Arduino solo sabía encender luces o detectar si un botón estaba apretado o suelto. En este módulo descubrirás cómo el Arduino mide cosas que cambian poco a poco (como la luz o el calor) y cómo nos escribe mensajes en la pantalla del computador.

1. La Ventana Mágica: ¿Qué es el Monitor Serie?

El Monitor Serie es una ventana de chat que viene dentro del programa Arduino IDE. Usa el mismo cable USB para que el Arduino te envíe textos y números en vivo:

* Serial.begin(9600); (se pone en setup): Abre la línea telefónica entre el Arduino y tu PC a una velocidad estándar de 9600.

* Serial.print("Hola");: Escribe un texto o número sin bajar de renglón.

* Serial.println(temperatura);: Escribe el dato y hace "Punto y aparte" (Enter) para que el siguiente mensaje salga en el renglón de abajo.

2. Digital vs. Analógico: El Interruptor vs. La Perilla de Volumen

Concepto · Ejemplo de tu Casa · En qué pines del Arduino se conecta · Qué números lee el Arduino

:--- · :--- · :--- · :---

Digital · Interruptor de la luz del cuarto: Solo tiene 2 estados (Encendido o Apagado). · Pines Digitales (0 al 13) con digitalRead(). · Solo dos valores: LOW (0) o HIGH (1).

Analógico · Perilla de volumen de la radio o dimmer: Sube y baja suavemente pasando por muchos puntos intermedios. · Pines Analógicos (A0 al A5) con analogRead(). · Una escala gradual desde 0 (0 Voltios) hasta 1023 (5 Voltios).

3. ¿Cómo funcionan nuestros 3 sensores de hoy?

A. El Sensor de Luz (`LDR` / Fotocelda)

Funciona como una puerta que se abre con la luz:

* Cuando hay mucha luz (Día), deja pasar la electricidad fácilmente y el pin A1 lee un número alto (por ejemplo, 750).

* Cuando lo tapas con la mano (Noche), frena la electricidad y el número baja (por ejemplo, 200).

B. El Sensor de Temperatura y Humedad (`DHT11` / `DHT22`)

Adentro de su cajita con rejilla tiene dos pequeños sensores electrónicos y un mini-chip que traduce los datos y se los entrega listos al Arduino por el Pin Digital 4 gracias a la librería DHT.

C. El Sensor de Distancia Ultrasónico (`HC-SR04`)

Funciona exactamente igual que un murciélago en la oscuridad o el sensor de estacionamiento de un auto:

1. El cilindro Trig (Disparador) emite un "clic" de sonido ultrasónico (tan agudo que las personas ni las mascotas lo oyen).

2. El sonido viaja por el aire, choca contra tu mano y rebota de vuelta.

3. El cilindro Echo (Receptor) escucha el eco. Sabiendo cuánto tardó en ir y volver, dividimos ese tiempo por 58 y obtenemos los centímetros exactos de distancia.

Plano y Conexiones Cable a Cable — Módulo 03: Sensores de Luz, Clima y Distancia

[!IMPORTANT]

Antes de empezar: En este módulo ya usamos los dos rieles largos de tu protoboard: conecta un cable rojo desde 5V del Arduino al riel largo rojo (+), y un cable negro desde GND del Arduino al riel largo azul (-).

1. Tabla de Conexiones Pin a Pin Completa

Componente · Pin del Componente · A dónde se conecta en el Arduino UNO / Protoboard · Función

:--- · :--- · :--- · :---

Alimentación General · Pines 5V y GND · Riel Rojo (+) y Riel Azul (-) de la protoboard · Lleva 5 Voltios y Tierra a todos los sensores.

Potenciómetro (Perilla) · Patita Izq. / Centro / Der. · Riel Rojo (+) / Pin Analógico A0 / Riel Azul (-) · Envía un valor gradual de 0 a 1023 según el giro.

Sensor de Luz (LDR) · VCC / OUT / GND · Riel Rojo (+) / Pin Analógico A1 / Riel Azul (-) · Mide el nivel de iluminación de la habitación.

Sensor de Clima (DHT11/22) · + (VCC) / OUT / - (GND) · Riel Rojo (+) / Pin Digital 4 / Riel Azul (-) · Entrega Temperatura (°C) y Humedad (%).

Sensor Distancia (HC-SR04) · VCC y GND · Riel Rojo (+) y Riel Azul (-) · Alimenta el sonar ultrasónico.

Sensor Distancia (HC-SR04) · Trig y Echo · Pin Digital 5 (Trig) y Pin Digital 6 (Echo) · Dispara el ultrasonido y escucha el eco para medir en cm.

LED Indicador / Farola · Patita Larga (+) / Corta (-) · Pin Digital 8 / Resistencia 220 $\rightarrow$ Riel Azul (-) · Se enciende en la oscuridad o cuando algo se acerca a < 15 cm.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad

En el siguiente plano puedes ver cómo se conectan el potenciómetro en A0, la fotocelda LDR en A1, el sensor de clima DHT en el Pin 4 y el sensor de distancia HC-SR04 en los Pines 5 y 6:

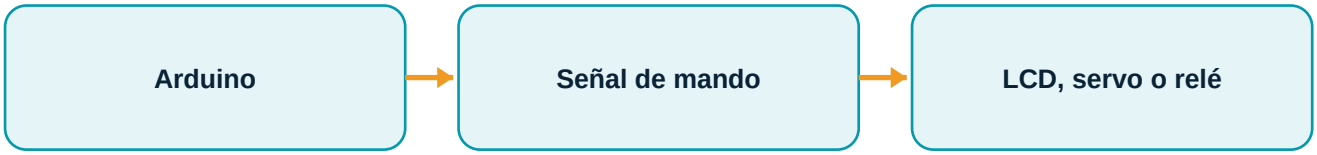
3. Plano 2: Guía Visual de Funcionamiento

En este diagrama visual observarás cómo los sensores transforman la luz, el calor y la distancia del mundo físico en números claros dentro del Monitor Serie de tu computador:

Capítulo 04 · Pantalla LCD I²C, servomotor y módulo relé

Conceptos clave

Comprueba la dirección I²C del LCD. Verifica el consumo del servo, usa una alimentación adecuada y comparte la referencia de tierra para su señal.



Guía Paso a Paso de Práctica 04: Pantalla LCD I2C, Servomotor y Módulo Relé

[!TIP]

Meta de esta práctica (30 minutos): Conectar tu propia pantalla digital LCD 16x2 con solo 4 cables, controlar el giro exacto de un brazo mecánico (Servomotor SG90) y activar un interruptor magnético (Módulo Relé) para encender un ventilador cuando sube la temperatura.

Paso 1: Conecta y Prueba tu Pantalla `LCD 16x2 I2C` (10 minutos)

1. Mira la parte trasera de tu pantalla LCD: verás una plaquita negra soldada con 4 patitas (GND, VCC, SDA, SCL).

2. Conecta 4 cables jumper Macho-Hembra desde esa mochilita hacia tu Arduino UNO:

* GND $\rightarrow$ Riel Azul (GND).

* VCC $\rightarrow$ Riel Rojo (5V).

* SDA $\rightarrow$ Pin Analógico A4 del Arduino UNO.

* SCL $\rightarrow$ Pin Analógico A5 del Arduino UNO.

3. En Arduino IDE, abre el Gestor de Bibliotecas e instala LiquidCrystal I2C (autor: *Frank de Brabander*).

4. Sube el archivo 01_pantalla_lcd_i2c_hola_mundo.ino.

* *Truco clave:* Si la pantalla se ilumina pero no ves las letras (o ves cuadraditos blancos), gira suavemente con un destornillador pequeño el tornillito azul de contraste que está en la espalda de la pantalla hasta que aparezca el texto "Hola Mundo IoT!".

Paso 2: Mueve el Brazo Mecánico del Servomotor `SG90` (10 minutos)

1. Encaja una de las hélices o bracitos plásticos blancos en el eje superior de tu Servomotor SG90.

2. Conecta sus 3 cables a la protoboard y al Arduino:

* Cable Marrón (o Negro) $\rightarrow$ Riel Azul (GND).

* Cable Rojo (del centro) $\rightarrow$ Riel Rojo (5V).

* Cable Naranja (o Amarillo) $\rightarrow$ Pin Digital 9 del Arduino UNO.

3. Deja conectado tu potenciómetro (perilla) en el Pin Analógico A0.

4. Sube el archivo 02_control_servomotor_sg90.ino y gira la perilla: ¡el brazo mecánico se moverá entre 0° y 180° imitando tu mano!

Paso 3: Conecta el Módulo Relé y arma el Termostato Autónomo (10 minutos)

1. Conecta los 3 pines de entrada del Módulo Relé hacia tu Arduino UNO:

* VCC $\rightarrow$ Riel Rojo (5V).

* GND $\rightarrow$ Riel Azul (GND).

* IN (Señal) $\rightarrow$ Pin Digital 7 del Arduino UNO.

2. Mantén tu sensor de temperatura DHT11/DHT22 conectado en el Pin Digital 4.

3. Sube el archivo 03_termostato_lcd_servo_y_rele.ino.

4. Observa la temperatura en la pantalla LCD y sopla aire tibio sobre el sensor DHT: cuando supere los 28 °C, escucharás el "¡clic!" del relé, el servomotor abrirá la compuerta a 90° y la pantalla dirá Ventilador: ON.

Listado de Materiales (BOM) — Módulo 04: Pantallas, Servomotores y Relés

[!NOTE]

Con las piezas de este cuarto módulo tu proyecto deja de depender de la pantalla del computador y adquiere pantalla propia, movimiento mecánico y capacidad de encender un ventilador real.

Tabla de Componentes del Módulo 04

Ítem · Componente · Cantidad · ¿Cómo reconocerlo en tu caja? · ¿Para qué sirve en esta práctica?

:---: · :--- · :---: · :--- · :---

1 · Arduino UNO R3 + Protoboard · 1 c/u · Tu placa controladora azul y tabla blanca. · Coordina el sensor, la pantalla LCD, el servomotor y el relé.

2 · Pantalla LCD 16x2 con adaptador I2C soldado · 1 · Pantalla rectangular verde o azul con una plaquita negra detrás de 4 pines (GND, VCC, SDA, SCL). · Muestra textos y mediciones en 2 renglones de 16 letras cada uno.

3 · Servomotor Micro SG90 (con bracitos plásticos) · 1 · Pequeño motor azul translúcido con engranajes y cable de 3 colores (Marrón, Rojo, Naranja). · Gira a un ángulo exacto entre 0° y 180° para

abrir/cerrar compuertas.

4 · Módulo Relé de 1 Canal (5V) · 1 · Plaquita con un cubito azul (el relé) y una bornera de 3 tornillos de un lado. · Interruptor magnético que hace *"clic"* para encender un ventilador de 5V o 12V.

5 · Sensor de Clima DHT11 o DHT22 · 1 · Sensor de temperatura y humedad de 3 pines (usado en el Módulo 03). · Mide los grados Celsius para que el termostato tome decisiones.

6 · Mini Ventilador DC (5V o 12V) — *Opcional* · 1 · Pequeño ventilador de 2 cables (rojo y negro) como los de PC. · Se conecta a las borneras del relé para ventilar cuando sube el calor.

7 · Cables Jumper Macho-Hembra y Macho-Macho · 12 · Cables con punta hueca (Hembra) para la pantalla LCD y punta aguja (Macho) para el Arduino. · Permiten conectar la pantalla I2C, el relé y el servo sin soldar.

Manual Explicado desde Cero — Módulo 04: Pantalla LCD I2C, Servomotor y Relé

[!NOTE]

En este módulo aprenderás cómo funcionan por dentro los tres componentes favoritos de todo creador de proyectos: la pantalla LCD I2C, el servomotor de precisión y el módulo relé.

1. ¿Por qué nuestra Pantalla LCD lleva una "Mochilita" `I2C`?

* Una pantalla LCD 16x2 significa que puede mostrar 16 letras por renglón en 2 renglones (Fila 0 arriba y Fila 1 abajo).

* Antiguamente había que conectar 12 cables entre la pantalla y el Arduino, ocupando casi todos los pines.

* La plaquita negra soldada detrás se llama Adaptador I2C: recibe las palabras del Arduino por solo 2 cables de comunicación (SDA en A4 y SCL en A5) más los 2 cables de energía (5V y GND), dejando libres todos los pines digitales.

Las 4 órdenes que necesitas saber para usar la pantalla:

1. lcd.init(); $\rightarrow$ Despierta la pantalla al encender el Arduino.

2. lcd.backlight(); $\rightarrow$ Enciende la luz de fondo para leer en la oscuridad.

3. lcd.setCursor(columna, fila); $\rightarrow$ Elige dónde empezar a escribir ((0, 0) es arriba a la izquierda; (0, 1) es abajo a la izquierda).

4. lcd.print("Texto"); $\rightarrow$ Escribe palabras o números en la pantalla.

2. Motor Común vs. Servomotor `SG90`

Característica · Motor Eléctrico Común · Servomotor (SG90)

:--- · :--- · :---

¿Cómo se mueve? · Gira dando vueltas completas sin parar mientras tenga corriente. · Se mueve únicamente al ángulo exacto que le pidas entre 0° y 180° y mantiene esa posición.

¿Para qué se usa? · Ventiladores, hélices o ruedas. · Abrir una ventana de invernadero, mover una cerradura, girar una cámara o dispensar alimento.

¿Cómo se ordena en código? · Encendido / Apagado. · miServo.write(90); (lo lleva exactamente a 90 grados).

3. ¿Qué es un Módulo Relé y por qué hace *"Clic"*?

El Arduino UNO es un "cerebro", no un "músculo de fuerza": sus pines entregan muy poquita corriente (apenas para encender un LED). Si quieres encender un ventilador, una tira de luces o una bomba de agua de maceta, usas un Módulo Relé:

1. Del lado del Arduino (3 pines: VCC, GND, IN): El Arduino envía una señal pequeñita y sin esfuerzo por el pin IN (Pin Digital 7).

2. Adentro del cubito azul: Un pequeño electroimán se activa y mueve una palanquita metálica haciendo un sonido audible de *"¡clic!"*.

3. Del lado de los tornillos (Borneras COM y NO): Esa palanquita une los dos cables del ventilador para que funcione con su propia energía, totalmente separado y aislado del Arduino.

Plano y Conexiones Cable a Cable — Módulo 04: Pantalla LCD I2C, Servo y Relé

[!IMPORTANT]

Regla fácil para no olvidar nunca los pines I2C en Arduino UNO:

* SDA (Datos) va siempre al Pin Analógico A4.

* SCL (Reloj) va siempre al Pin Analógico A5.

1. Tabla de Conexiones Pin a Pin Completa

Componente · Pin / Cable del Componente · A dónde se conecta en el Arduino UNO / Protoboard · Explicación Simple

:--- · :--- · :--- · :---

Pantalla LCD 16x2 I2C · GND · Riel Azul (- GND) · Polo negativo de la pantalla.

Pantalla LCD 16x2 I2C · VCC · Riel Rojo (+ 5V) · Alimentación de 5 Voltios para la pantalla.

Pantalla LCD 16x2 I2C · SDA · Pin Analógico A4 · Cable por donde viajan las letras hacia la pantalla.

Pantalla LCD 16x2 I2C · SCL · Pin Analógico A5 · Cable que marca el ritmo de comunicación I2C.

Servomotor SG90 · Cable Marrón / Rojo / Naranja · Riel Azul (-) / Riel Rojo (+) / Pin Digital 9 · Mueve el brazo mecánico entre 0° y 180°.

Módulo Relé (1 Canal) · VCC / GND / IN · Riel Rojo (+) / Riel Azul (-) / Pin Digital 7 · Hace *"clic"* para encender el ventilador cuando el Pin 7 se activa.

Sensor Clima DHT11/22 · + (VCC) / OUT / - (GND) · Riel Rojo (+) / Pin Digital 4 / Riel Azul (-) · Mide la temperatura para mostrarla en el LCD.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad

En el siguiente plano puedes ver cómo se conectan los 4 cables de la pantalla LCD I2C hacia A4 y A5, el servomotor en el Pin 9, el módulo relé en el Pin 7 y el sensor DHT en el Pin 4:

3. Plano 2: Guía Visual de Funcionamiento

Observa en este esquema cómo el Arduino lee la temperatura del sensor, la escribe en los dos renglones de la pantalla LCD y decide automáticamente cuándo mover el servomotor y activar el relé:

Capítulo 05 · Raspberry Pi desde cero: Linux, Python y Thonny

Conceptos clave

Ejecuta primero un programa pequeño con salida visible. Conserva los datos y dependencias del proyecto separados de la instalación del sistema.



Ejemplo complementario

```
from platform import node

print(f"Servidor IoT: {node()}")
```

Guía Paso a Paso de Práctica 05: Despertando a tu Raspberry Pi y Programando en Thonny

[!TIP]

Meta de esta práctica (30 minutos): Preparar la memoria microSD de tu Raspberry Pi sin usar comandos difíciles, entrar a su escritorio gráfico con ventanas y ejecutar tus primeros 3 programas en Python con el botón verde de Thonny.

Paso 1: Prepara la tarjeta microSD con `Raspberry Pi Imager` (10 minutos)

1. Inserta tu tarjeta microSD (de 16 GB o 32 GB) en el lector de tarjetas de tu computador habitual.

2. Descarga y abre el programa gratuito oficial Raspberry Pi Imager (raspberrypi.com/software).

3. Haz clic en los 3 botones en orden:

* Elegir Dispositivo: Selecciona tu modelo (Raspberry Pi 4 o Raspberry Pi 5).

* Elegir Sistema Operativo: Selecciona la primera opción recomendada: Raspberry Pi OS (64-bit) (trae escritorio gráfico completo).

* Elegir Almacenamiento: Selecciona tu tarjeta microSD.

4. Haz clic en Siguiente y cuando pregunte si deseas aplicar ajustes personalizados, elige Editar Ajustes:

* Nombre de host: Escribe mi-raspberry.

* Usuario y contraseña: Escribe por ejemplo usuario pi y una contraseña fácil de recordar.

* Wi-Fi (LAN inalámbrica): Escribe el nombre exacto del Wi-Fi de tu casa y su clave.

* En la pestaña Servicios, activa la casilla Habilitar SSH.

5. Haz clic en Guardar y luego en Sí para grabar la tarjeta. Cuando diga * Grabación completada*, retira la microSD.

Paso 2: Enciende tu Raspberry Pi y Explora el Escritorio (8 minutos)

1. Con la Raspberry Pi desenchufada, inserta la tarjeta microSD en la ranura que está debajo de la placa.

2. Elige cómo quieres ver su pantalla:

* Con monitor o TV: Conecta el cable HDMI y un teclado/mouse USB a la Raspberry Pi.

* Desde la pantalla de tu PC: Usa Raspberry Pi Connect o VNC Viewer conectándote a mi-raspberry.local.

3. Enchufa el cargador USB-C de la Raspberry Pi.

4. En menos de 1 minuto verás aparecer el Escritorio Gráfico con el botón de la Frambuesa Roja arriba a la izquierda (igual al menú Inicio de Windows) y el ícono del Wi-Fi conectado arriba a la derecha.

Paso 3: Abre Thonny y Ejecuta tu Primer Script en Python (12 minutos)

1. Haz clic en la Frambuesa Roja (Menú) $\rightarrow$ Programación $\rightarrow$ Thonny.

2. Abre el archivo 01_hola_mundo_thonny.py y presiona el botón redondo verde Ejecutar (Play).

3. Mira la ventana inferior (Consola / Shell): verás aparecer el saludo y el cálculo de temperatura al instante.

4. Ahora abre y ejecuta 02_reloj_y_repeticion_en_vivo.py para ver cómo Python muestra la hora cada 2 segundos (presiona el botón rojo STOP para detenerlo).

5. Finalmente, ejecuta 03_mi_primer_historial_en_archivo.py y verifica cómo se crea automáticamente el archivo de texto historial_mi_habitacion.txt en tu carpeta.

Listado de Materiales (BOM) — Módulo 05: Tu Mini-Computador Raspberry Pi

[!NOTE]

En este quinto módulo dejamos descansar por un momento la protoboard y conocemos a la segunda protagonista de nuestro curso: la Raspberry Pi.

Tabla de Componentes del Módulo 05

Ítem · Componente · Cantidad · ¿Cómo reconocerlo? · ¿Para qué sirve en esta práctica?

:---: · :--- · :---: · :--- · :---

1 · Placa Raspberry Pi 4 o Raspberry Pi 5 · 1 · Placa verde del tamaño de una tarjeta de crédito con 4 puertos USB y conector de red. · Es nuestro mini-computador con Wi-Fi, escritorio gráfico y Python preinstalado.

2 · Fuente de Alimentación Oficial USB-C (5V) · 1 · Cargador con conector USB-C (5V 3A para Pi 4, o 5V 5A para Pi 5). · Alimenta de forma estable a la Raspberry Pi y a los dispositivos USB que le conectemos.

3 · Tarjeta de Memoria microSD (16 GB o 32 GB Clase 10 / A1) · 1 · Pequeña tarjeta de memoria como la que usan las cámaras o celulares, con su adaptador USB. · Actúa como el "disco duro" donde se instala el sistema operativo y se guardan tus programas.

4 · Cable micro-HDMI a HDMI (Opcional si usas TV/Monitor) · 1 · Cable con punta pequeñita (micro-HDMI) en un extremo y HDMI normal en el otro. · Permite ver el escritorio de la Raspberry Pi en cualquier televisor o monitor.

5 · Teclado y Mouse USB (Opcionales si usas TV/Monitor) · 1 set · Cualquier teclado y mouse normales de computador. · Se enchufan en los puertos USB de la Raspberry Pi (no los necesitas si entras desde tu laptop).

Manual Explicado desde Cero — Módulo 05: Tu Raspberry Pi y Python con Thonny

[!NOTE]

Si sabes usar Windows o Mac para abrir una carpeta y navegar por Internet, ¡ya sabes usar el 90% de una Raspberry Pi! En este manual te explicamos todo sin jerga técnica.

1. ¿Qué tiene adentro una Raspberry Pi?

Aunque cabe en el bolsillo de una camisa, la placa Raspberry Pi 4 o 5 es un computador completo y silencioso:

* Procesador y Memoria RAM: El chip cuadrado plateado del centro que ejecuta el escritorio, el navegador web y tus programas.

* Wi-Fi y Bluetooth integrados: Se conecta inalámbricamente al router de tu casa igual que tu teléfono.

* 4 Puertos USB normales: Donde enchufaremos en el próximo módulo el cable USB de nuestro Arduino UNO.

* Ranura para tarjeta microSD (debajo de la placa): Como no tiene un disco duro grande, todo su sistema operativo vive en la tarjeta microSD.

2. Comparación Rápida: Cómo se dice en Arduino vs. Cómo se dice en Python

En la Raspberry Pi programamos en Python usando el editor Thonny, que ya viene instalado en el menú de la Frambuesa. Mira qué fácil y parecido es a lo que ya aprendiste en Arduino:

Qué queremos hacer · Cómo lo escribíamos en Arduino (C++) · Cómo se escribe en Python (Thonny)

:--- · :--- · :---

Mostrar un texto en pantalla · Serial.println("Hola"); · print("Hola") *(¡sin punto y coma al final!)*

Crear una variable · int temperatura = 25; · temperatura = 25

Tomar una decisión (Si...) · if (temperatura > 28) { ... } · if temperatura > 28: *(usando dos puntos : y sangría)*

Esperar 2 segundos · delay(2000); · time.sleep(2)

Repetir sin parar (Bucle infinito) · void loop() { ... } · while True:

3. El Único Secreto de Python: La Sangría (Espacios a la Izquierda)

En Arduino usábamos llaves { y } para encerrar lo que iba adentro de un if o de un loop.

En Python no existen las llaves: en su lugar, dejamos 4 espacios hacia la derecha (Sangría / Tabulador) debajo de los dos puntos (:):

```
if temperatura_actual > 28:
    # Esta línea tiene sangría, así que está ADENTRO del if
    print("Hace calor, encendiendo ventilador")
```

Thonny pone esos espacios automáticamente por ti cada vez que presionas la tecla Enter después de escribir :.

Plano y Conexiones Cable a Cable — Módulo 05: Mapa de la Raspberry Pi

[!IMPORTANT]

Regla de Oro al encender tu Raspberry Pi: Primero inserta siempre la tarjeta microSD grabada en la ranura inferior y conecta el cable HDMI (si usas monitor), y por último enchufa el cargador de energía USB-C.

1. Tabla de Puertos y Conexiones de tu Raspberry Pi

Puerto / Ranura en la Raspberry Pi · Qué se conecta ahí · ¿Cuándo conectarlo? · Función en palabras simples

:--- · :--- · :--- · :---

Ranura inferior microSD · Tarjeta microSD de 16/32 GB grabada con *Raspberry Pi Imager* · Antes de enchufar la corriente · Es el cerebro y memoria donde vive el sistema operativo y tus archivos.

Puerto micro-HDMI 0 (junto al USB-C) · Cable micro-HDMI hacia tu televisor o monitor · Antes de enchufar la corriente *(Opcional si usas VNC/Connect)* · Envía la imagen del escritorio gráfico y el sonido hacia la pantalla.

Puertos USB 2.0 / 3.0 (Azules y Negros) · Teclado, Mouse y (desde el Módulo 06) el cable USB del Arduino · En cualquier momento (*Plug-and-Play*) · Permiten conectar periféricos y comunicarnos con el Arduino UNO.

Conector USB-C Power · Fuente de alimentación oficial de 5V · Al final, para encender la placa · Da energía eléctrica segura a toda la Raspberry Pi.

Chip Wi-Fi Interno · *(Sin cables)* Se conecta al router de tu casa · Automático al encender · Conecta la Raspberry Pi a tu red local y a Internet.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad (Mapa Visual de Puertos de la Raspberry Pi)

En la siguiente lámina visual puedes identificar cada puerto de tu Raspberry Pi, la ubicación de la tarjeta microSD, la fuente USB-C y su conexión con el entorno de trabajo:

![Plano 1: Montaje en Protoboard Estilo Tinkercad](planos_circuitos/plano_protoboard_modulo_05.png)

3. Plano 2: Guía Visual de Funcionamiento (De Raspberry Pi Imager a Thonny Python)

En este diagrama visual verás los 3 pasos completos del módulo: grabar la tarjeta microSD, abrir el escritorio gráfico y ejecutar tus scripts en el editor Thonny con el botón verde *Play*:

![Plano 2: Guía Visual de Funcionamiento](planos_circuitos/diagrama_visual_flujo_modulo_05.png)

Capítulo 06 · Puente USB bidireccional entre Arduino y Raspberry Pi

Conceptos clave

Usa la misma velocidad serie en ambos extremos. Delimita cada mensaje con un salto de línea y valida su contenido antes de actuar.



Ejemplo complementario

```
void setup() { Serial.begin(9600); }
void loop() {
  if (Serial.available() > 0) { Serial.write(Serial.read()); }
}
```

Guía Paso a Paso de Práctica 06: El Puente Mágico por Cable USB (Arduino + Raspberry Pi)

[!TIP]

Meta de esta práctica (30 minutos): Unir físicamente tu Arduino UNO con tu Raspberry Pi usando únicamente el cable USB normal del Arduino, recibir las mediciones de los sensores en Thonny Python y enviarle órdenes de regreso ('E' y 'A') para encender o apagar el ventilador.

Paso 1: Sube el programa al Arduino UNO (8 minutos)

1. Verifica que en tu protoboard estén conectados:

* Sensor DHT11/22 en el Pin Digital 4.

* Sensor de Luz LDR en el Pin Analógico A0.

* Módulo Relé en el Pin Digital 7 y LED con resistencia en el Pin Digital 8.

2. Abre el archivo 01_arduino_envia_y_recibe_usb.ino y súbelo a tu placa Arduino UNO.

Paso 2: Enchufa el Cable USB del Arduino en tu Raspberry Pi (5 minutos)

1. Si subiste el programa desde tu computador habitual, desconecta el cable USB de tu PC y enchúfalo directamente en cualquiera de los 4 puertos USB de tu Raspberry Pi.

2. ¡Mira qué cómodo! El Arduino se encenderá de inmediato tomando energía de la Raspberry Pi y ambos quedarán comunicados sin usar ni un solo cable suelto entre pines.

Paso 3: Recibe los datos del Arduino en Thonny Python (8 minutos)

1. En el escritorio de tu Raspberry Pi, abre Thonny Python.

2. Abre el archivo 02_raspberry_recibe_datos_usb.py.

3. Presiona el botón verde Ejecutar (Play).

4. Mira la consola inferior de Thonny: cada 2 segundos verás llegar el informe en vivo desde el Arduino:

[16:40:12] El Arduino dice: TEMP:24.5,HUM:55,LUZ:710

Paso 4: Haz que la Raspberry Pi le dé órdenes al Arduino (9 minutos)

1. Detén el programa anterior con el botón rojo STOP de Thonny.

2. Abre el archivo 03_raspberry_manda_ordenes_usb.py y presiona Ejecutar (Play).

3. Acerca aire tibio al sensor DHT para que supere los 28 °C: verás cómo Python envía automáticamente la letra 'E' por el cable USB y en tu protoboard se encienden el LED y el relé del ventilador.

Listado de Materiales (BOM) — Módulo 06: Puente USB entre Arduino y Raspberry Pi

[!NOTE]

¡La mejor noticia de este módulo es que no necesitas comprar ningún adaptador ni conversor de voltaje! El propio cable USB azul/negro que vino con tu Arduino UNO hace todo el trabajo de forma 100% segura.

Tabla de Componentes del Módulo 06

Ítem · Componente · Cantidad · ¿Cómo reconocerlo? · ¿Para qué sirve en esta práctica?

:---: · :--- · :---: · :--- · :---

1 · Raspberry Pi 4 o 5 (encendida con microSD) · 1 · Tu mini-computador preparado en el Módulo 05. · Recibe los datos en Python, guarda el historial y toma decisiones.

2 · Arduino UNO R3 · 1 · Tu placa controladora azul. · Lee los sensores de la protoboard y obedece las órdenes 'E' y 'A'.

3 · Cable USB estándar del Arduino · 1 · El mismo cable USB que usas para programar el Arduino. · El Puente Mágico: alimenta al Arduino y transporta los mensajes de ida y vuelta.

4 · Sensor DHT11/22 + Sensor de Luz LDR · 1 c/u · Sensores de clima y luz montados en la protoboard. · Miden temperatura, humedad y luz cada 2 segundos.

5 · Módulo Relé + LED con Resistencia de 220 Ohmios · 1 c/u · Conectados en los Pines Digitales 7 y 8 del Arduino. · Se encienden cuando la Raspberry Pi envía la letra 'E' y se apagan con 'A'.

Manual Explicado desde Cero — Módulo 06: Cómo Conversan Arduino y Raspberry Pi por USB

[!NOTE]

En este manual entenderás por qué conectar ambas placas mediante el cable USB es el método favorito para aprender sin riesgos y cómo funciona el diálogo en ambos sentidos.

1. ¿Por qué conectamos por Cable USB en lugar de unir pines sueltos?

Criterio · Unir pines sueltos (GPIO) · Conectar por Cable USB (Nuestro Método)

:--- · :--- · :---

Riesgo de quemar la placa · Alto para principiantes (el Arduino usa 5V y los pines de la Raspberry Pi solo toleran 3.3V). · Cero riesgo (100% seguro): los puertos USB están diseñados para trabajar a 5V con protección integrada.

Cantidad de cables · Varios cables sueltos + divisores de resistencias. · Un solo cable USB con ficha que no se puede enchufar al revés.

Energía del Arduino · Requiere alimentar al Arduino aparte. · La propia Raspberry Pi alimenta eléctricamente al Arduino por el mismo cable USB.

2. Cómo viajan los mensajes de Ida (Arduino $\rightarrow$ Raspberry Pi)

1. En el Arduino: Cada 2 segundos arma una frase corta y prolija separada por comas y la envía con Serial.println():

text

TEMP:24.5,HUM:55,LUZ:720

2. En la Raspberry Pi (Python): Lee ese renglón con tres palabras mágicas encadenadas:

python

linea = puerto.readline().decode('utf-8').strip()

* readline(): Espera hasta que llega un renglón completo por el cable USB.

* decode('utf-8'): Convierte los impulsos eléctricos del cable en letras normales.

* strip(): Limpia el salto de renglón final para que el texto quede prolijo.

3. Cómo viajan las órdenes de Vuelta (Raspberry Pi $\rightarrow$ Arduino)

Para que la Raspberry Pi controle las salidas del Arduino usamos órdenes de una sola letra:

Letra enviada desde Python · Instrucción en Python · Qué hace el Arduino al recibirla

:---: · :--- · :---

'E' *(Encender)* · puerto.write(b'E') · Activa el Pin 7 (Relé) y el Pin 8 (LED).

'A' *(Apagar)* · puerto.write(b'A') · Apaga el Pin 7 (Relé) y el Pin 8 (LED).

[!TIP]

¿Por qué se escribe b'E' con una b adelante en Python? Esa letra b pequeña significa *bytes* y simplemente le indica a Python que envíe la letra 'E' lista para viajar por el cable USB.

Plano y Conexiones Cable a Cable — Módulo 06: El Puente USB Arduino y Raspberry Pi

[!IMPORTANT]

Importante al usar el puerto USB: Solo un programa a la vez puede usar el cable USB. Si tienes abierto el *Monitor Serie* de Arduino IDE, ciérralo antes de darle *Play* a tu programa en Thonny Python.

1. Tabla de Conexiones Pin a Pin y Enlace USB

Dispositivo Origen · Pin / Puerto Origen · A dónde se conecta · Función en el Sistema

:--- · :--- · :--- · :---

Arduino UNO · Conector USB Plateado · Puerto USB de la Raspberry Pi · Alimenta al Arduino y lleva datos y órdenes (9600 baudios).

Arduino UNO · Pines 5V y GND · Riel Rojo (+) y Riel Azul (-) · Energiza la protoboard.

Sensor Clima DHT11/22 · + / OUT / - · Riel Rojo (+) / Pin Digital 4 / Riel Azul (-) · Mide temperatura y humedad para enviarlas a la Raspberry Pi.

Sensor de Luz LDR · VCC / OUT / GND · Riel Rojo (+) / Pin Analógico A0 / Riel Azul (-) · Mide el nivel de iluminación ambiental (0 a 1023).

Módulo Relé (Ventilador) · VCC / GND / IN · Riel Rojo (+) / Riel Azul (-) / Pin Digital 7 · Se enciende al recibir 'E' y se apaga al recibir 'A'.

LED Indicador · Patita Larga (+) / Corta (-) · Pin Digital 8 / Resistencia 220 $\rightarrow$ Riel Azul (-) · Muestra visualmente si la orden actual es Encendido o Apagado.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad (Arduino + Protoboard + Enlace USB a Raspberry Pi)

En el siguiente plano puedes ver el montaje completo de sensores y relé en el Arduino UNO y cómo su cable USB se conecta directamente al puerto USB de la Raspberry Pi:

3. Plano 2: Guía Visual de Funcionamiento (Conversación Bidireccional por Cable USB)

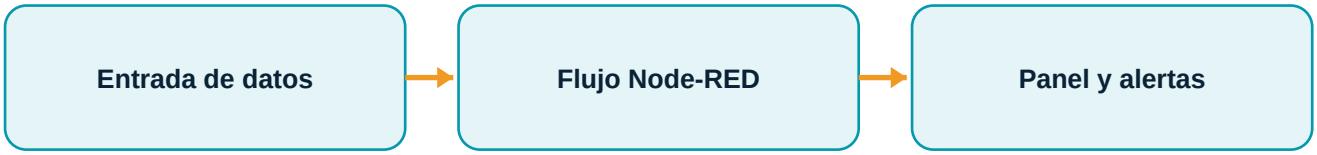
Observa en este esquema cómo viajan las mediciones TEMP, HUM, LUZ hacia Python cada 2 segundos y cómo regresan las órdenes 'E' y 'A' por el mismo cable:



Capítulo 07 · Node-RED: flujos, panel móvil y alertas

Conceptos clave

Normaliza la entrada antes de visualizarla. Usa histéresis o un intervalo mínimo para evitar alertas repetidas al oscilar cerca del umbral.



Guía Paso a Paso de Práctica 07: Tu Primer Panel en el Celular (`Node-RED`) y Alertas en Telegram

[!TIP]

Meta de esta práctica (30 minutos): Crear un panel con velocímetros, gráficas e interruptores accesible desde el navegador de tu teléfono celular sin programar páginas web, y configurar un robot asistente en Telegram para recibir alertas en tu bolsillo.

Paso 1: Abre `Node-RED` en tu Raspberry Pi (5 minutos)

1. Abre el navegador web en tu Raspberry Pi (o desde tu PC conectado al mismo Wi-Fi) y escribe en la barra de direcciones:

http://mi-raspberry.local:1880

2. Verás la pantalla de Node-RED: bloques de colores a la izquierda, una hoja cuadriculada en el centro y el botón rojo Deploy arriba a la derecha.

Paso 2: Importa el Panel Listo con 2 Clics (10 minutos)

1. En Node-RED, haz clic en el menú de las 3 rayitas (■) arriba a la derecha $\rightarrow$ Import (Importar).

2. Haz clic en select a file to import y elige el archivo 01_flujo_nodered_panel_celular.json de la carpeta codigo_fuente/.

3. Conecta un bloque serial in (apuntando a /dev/ttyACM0 a 9600) a la entrada del bloque separador, y conecta la salida del interruptor a un bloque serial out.

4. Haz clic en el botón rojo Deploy arriba a la derecha.

Paso 3: Abre el Panel en tu Teléfono Celular (5 minutos)

1. Asegúrate de que tu celular esté conectado al mismo Wi-Fi de tu casa que la Raspberry Pi.

2. Abre el navegador de tu teléfono (Chrome o Safari) y escribe:

http://mi-raspberry.local:1880/ui

3. ¡Mira la pantalla de tu teléfono! Verás la aguja del velocímetro de temperatura, los medidores de humedad y luz, la gráfica en vivo y el botón interruptor para encender y apagar el ventilador con tu dedo.

Paso 4: Crea tu Bot de Telegram y Recibe tu Primera Alerta (10 minutos)

1. Abre Telegram en tu celular, busca a @BotFather (con tilde azul de verificado) y envíale el mensaje: /newbot.

2. Escribe un nombre para tu asistente (por ejemplo, Mi Casa IoT) y luego un apodo que termine en bot (por ejemplo, micasa_iot_123_bot).

3. Copia el código TOKEN que te entrega @BotFather y abre un chat con tu nuevo bot presionando Iniciar.

4. Busca a @userinfobot en Telegram y presiona Iniciar para ver tu número de Id (CHAT_ID).

5. Abre en Thonny el archivo 02_enviar_alerta_telegram.py, pega tu TOKEN y tu CHAT_ID, y presiona el botón verde Play: ¡tu teléfono sonará al instante con el mensaje de tu Raspberry Pi!

Listado de Materiales (BOM) — Módulo 07: Panel en el Celular y Telegram

[!NOTE]

En este séptimo módulo mantenemos exactamente el mismo circuito del Módulo 06 (Arduino + Sensores + Relé + Cable USB hacia la Raspberry Pi) y sumamos tu teléfono celular o tablet conectado al Wi-Fi de casa.

Tabla de Elementos del Módulo 07

Ítem · Elemento · Cantidad · ¿Qué función cumple en este módulo?

:---: · :--- · :---: · :---

1 · Raspberry Pi 4/5 + Arduino UNO por Cable USB · 1 set · El Arduino mide y acciona el relé; la Raspberry Pi corre Node-RED y envía mensajes a Telegram.

2 · Sensores (DHT11/22 y LDR) + Módulo Relé y LED · 1 set · Envían datos en vivo hacia los velocímetros del celular y obedecen al interruptor táctil.

3 · Teléfono Celular o Tablet (Android o iPhone/iPad) · 1 · Para abrir el panel web http://mi-raspberry.local:1880/ui y recibir avisos en Telegram.

4 · Herramienta Visual Node-RED (Gratuita) · Software · Permite crear velocímetros, gráficas y botones arrastrando cajitas con el mouse.

5 · Aplicación Telegram (Gratuita) · App · Nos permite crear nuestro propio robot asistente (@BotFather) gratis en 3 minutos.

Manual Explicado desde Cero — Módulo 07: `Node-RED` y Bots de Telegram

[!NOTE]

Crear páginas web desde cero suele llevar meses de estudio. En este módulo aprenderás cómo Node-RED te permite armar paneles profesionales para tu celular en minutos como si fueran piezas de LEGO.

1. ¿Qué es `Node-RED` y cómo funcionan sus cajitas (*Nodos*)?

Node-RED es un programa visual que funciona dentro de tu Raspberry Pi. En lugar de escribir código para dibujar botones o agujas, arrastras cajitas de colores y las unes con líneas:

Cajita en Node-RED · De qué lado va · ¿Qué hace en palabras simples?

:--- · :--- · :---

serial in (Entrada USB) · Izquierda (Entrada) · Escucha los mensajes que llegan por el cable USB desde el Arduino.

function (Separador) · Centro (Proceso) · Corta el texto TEMP:24.5,HUM:55,LUZ:720 en tres números separados.

gauge (Velocímetro) · Derecha (Pantalla) · Dibuja una aguja circular en el celular que cambia de verde a amarillo y rojo.

chart (Gráfica) · Derecha (Pantalla) · Dibuja una línea en el tiempo para ver cómo subió o bajó la temperatura en la última hora.

switch (Interruptor) · Izquierda (Botón táctil) · Dibuja un botón en el celular que envía 'E' o 'A' hacia el bloque serial out.

Las 2 direcciones web que debes recordar:

* Para armar y editar las cajitas: http://mi-raspberry.local:1880

* Para ver el panel terminado en tu celular: http://mi-raspberry.local:1880/ui *(termina en /ui)*.

2. ¿Cómo funciona un Bot de Telegram y por qué usamos un "Cerrojo" de Alerta?

Para que la Raspberry Pi te escriba al celular solo necesita dos datos:

1. TOKEN: Es la contraseña de tu robot asistente (te la entrega @BotFather).

2. CHAT_ID: Es el número de tu casilla personal dentro de Telegram (te lo muestra @userinfobot).

El truco del "Cerrojo" (`alerta_ya_enviada`)

Como el Arduino mide la temperatura cada 2 segundos, si hace 29 °C durante media hora y no pusiéramos un control, ¡tu teléfono recibiría 900 mensajes seguidos!

Por eso usamos una variable alerta_ya_enviada:

* Cuando sube a 28 °C por primera vez, envía 1 solo mensaje y pone alerta_ya_enviada = True.

* Solo cuando la temperatura vuelve a enfriarse por debajo de 27 °C, envía un mensaje de *"Clima normalizado"* y vuelve a abrir el cerrojo (False).

Plano y Conexiones Cable a Cable — Módulo 07: Panel Celular `Node-RED` y Telegram

[!IMPORTANT]

Recuerda antes de presionar Deploy en Node-RED: Asegúrate de que Thonny Python y el Monitor Serie estén cerrados para que Node-RED tenga acceso exclusivo al cable USB del Arduino (/dev/ttyACM0).

1. Tabla de Conexiones Físicas y Bloques Visuales

Parte del Sistema · Origen · Destino · Qué viaja por esa conexión

:--- · :--- · :--- · :---

Enlace Físico USB · Arduino UNO (Sensores y Relé) · Puerto USB de la Raspberry Pi · Datos TEMP,HUM,LUZ y órdenes 'E' / 'A' a 9600 baudios.

Bloque serial in en Node-RED · Puerto /dev/ttyACM0 · Bloque Separar Datos · Renglón de texto que llega cada 2 segundos.

Salidas del Separador · Salidas 1, 2 y 3 · Bloques Gauge y Chart · Temperatura (°C), Humedad (%) y Luz (0-1023).

Interruptor en el Celular · Bloque ui_switch · Bloque serial out (/dev/ttyACM0) · Letra 'E' al encender y letra 'A' al apagar con el dedo.

Enlace Wi-Fi Doméstico · Raspberry Pi · Tu Teléfono Celular · Panel web en vivo (:1880/ui) y alertas a Telegram.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad

En el siguiente plano se muestra el circuito de sensores y relé conectado por cable USB a la Raspberry Pi, que sirve el panel inalámbrico hacia tu teléfono celular:

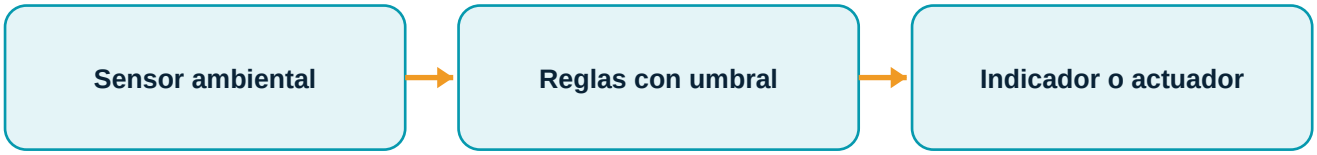
3. Plano 2: Guía Visual de Funcionamiento (Flujo de Cajitas en Node-RED y Bot de Telegram)

Observa en este esquema cómo se unen las cajitas dentro de Node-RED para transformar las mediciones del cable USB en velocímetros, gráficas, botones táctiles y notificaciones de Telegram:

Capítulo 08 · Proyecto: habitación e invernadero IoT

Conceptos clave

Comprueba el sensor, el tablero y el actuador por separado. Define un estado de salida seguro si se pierde la lectura ambiental.



Guía Paso a Paso de Práctica 08: Proyecto Final — Tu Habitación o Invernadero Inteligente

[!TIP]

Meta del Proyecto Final (40 minutos): Armar paso a paso tu estación inteligente completa (Clima + Luz + Pantalla LCD + LEDs + Buzzer + Ventilador con Relé + Puente USB a Raspberry Pi + Panel en el Celular y Alerta a Telegram) y realizar las 4 pruebas de graduación.

Etapa 1: Conecta los Sensores en la Protoboard (10 minutos)

1. Conecta primero tus dos autopistas de energía desde el Arduino UNO hacia la protoboard:

* Pin 5V $\rightarrow$ Riel Largo Rojo (+).

* Pin GND $\rightarrow$ Riel Largo Azul (-).

2. Sensor de Temperatura y Humedad DHT11/22:

* + (VCC) al Riel Rojo (+), - (GND) al Riel Azul (-) y OUT al Pin Digital 4.

3. Sensor de Luz LDR:

* Alimentación al Riel Rojo/Azul y salida de medición al Pin Analógico A0.

Etapa 2: Conecta la Pantalla LCD, los LEDs y el Buzzer (12 minutos)

1. Pantalla LCD 16x2 I2C (4 cables):

* GND al Riel Azul (-), VCC al Riel Rojo (+), SDA al Pin A4 y SCL al Pin A5.

2. Luces Indicadoras (con sus resistencias de 220 Ohmios hacia el Riel Azul):

* LED Verde (Clima Normal): Patita larga (+) al Pin Digital 8.

* LED Rojo (Alerta de Calor): Patita larga (+) al Pin Digital 9.

3. Buzzer de Aviso:

* Patita positiva (+) al Pin Digital 11 y patita negativa (-) al Riel Azul (-).

Etapa 3: Conecta el Módulo Relé, sube el Código y une el Cable USB (8 minutos)

1. Módulo Relé (Ventilador):

* VCC al Riel Rojo (+), GND al Riel Azul (-) y señal IN al Pin Digital 7.

2. Sube al Arduino UNO el programa maestro 01_proyecto_final_arduino_habitacion_inteligente.ino.

3. Escucharás los dos pitidos alegres de bienvenida y verás encenderse la pantalla LCD y el LED verde.

4. Enchufa el cable USB del Arduino en un puerto USB de tu Raspberry Pi.

Etapa 4: Las 4 Pruebas Reales de Graduación (10 minutos)

Prueba · Qué hacer con tus manos · Qué debes observar como resultado

:--- · :--- · :---

Prueba 1: Reposo · Deja los sensores a temperatura ambiente y con luz normal. · El LED Verde está encendido, el ventilador apagado y la pantalla LCD muestra T, H, L y Vent:OFF.

Prueba 2: Sombra / Noche · Cubre con la palma de tu mano el sensor de luz LDR. · En la pantalla LCD y en el velocímetro del celular el nivel L: baja inmediatamente por debajo de 300.

Prueba 3: Calor Automático · Sopla suavemente aire tibio sobre el sensor DHT hasta pasar de 28 °C. · Suena el Buzzer, cambia del LED Verde al LED Rojo, el Relé hace *"clic"* encendiendo el ventilador y llega la alerta a tu Telegram.

Prueba 4: Control desde el Celular · Toca los botones ENCENDER, APAGAR y MODO AUTOMATICO en el panel de tu teléfono. · El Arduino emite un pitido de confirmación y obedece tu orden táctil en menos de 1 segundo.

Listado de Materiales (BOM) — Módulo 08: Proyecto Final Completo

[!NOTE]

Este proyecto integrador reúne las piezas de tu Kit de Iniciación que ya aprendiste a usar a lo largo de los Módulos 01 al 07.

Tabla Completa de Materiales del Proyecto Final

Ítem · Componente · Cantidad · Pin en el Arduino UNO / Conexión · Rol dentro del Proyecto Final

:---: · :--- · :---: · :--- · :---

1 · Arduino UNO R3 + Cable USB · 1 · Enchufado por USB a la Raspberry Pi · Lee los sensores, actualiza el LCD y controla el relé, LEDs y buzzer.

2 · Raspberry Pi 4 o 5 (con microSD y fuente USB-C) · 1 · Conectada al Wi-Fi de casa · Guarda el historial CSV, muestra el panel en el celular y envía avisos a Telegram.

3 · Protoboard de 830 puntos + Cables Jumper · 1 set · Rieles Rojo (5V) y Azul (GND) · Tablero donde se unen todos los componentes sin soldadura.

4 · Sensor de Temperatura y Humedad DHT11 o DHT22 · 1 · Pin Digital 4 (5V y GND) · Mide los grados Celsius (°C) y el porcentaje de humedad (%).

5 · Sensor de Luz LDR (Módulo o con resistencia 10k) · 1 · Pin Analógico A0 (5V y GND) · Mide la iluminación del cuarto o invernadero (0 a 1023).

6 · Pantalla LCD 16x2 con adaptador I2C · 1 · SDA $\rightarrow$ Pin A4 y SCL $\rightarrow$ Pin A5 · Muestra en el escritorio la temperatura, humedad, luz y estado del ventilador.

7 · Módulo Relé de 1 Canal + Mini Ventilador 5V/12V · 1 · Pin Digital 7 (5V y GND) · Enciende y apaga el ventilador real haciendo *"clic"*.

8 · LED Verde + LED Rojo + 2 Resistencias de 220 Ohmios · 1 par · Pin Digital 8 (Verde) y Pin 9 (Rojo) · Semáforo de estado: Verde = clima normal; Rojo = calor / ventilador activo.

9 · Buzzer (Zumbador) · 1 · Pin Digital 11 y GND · Emite pitidos audibles al iniciar, al detectar calor o al recibir órdenes del celular.

Manual Explicado desde Cero — Módulo 08: Cómo Funciona tu Proyecto Final de Punta a Punta

[!NOTE]

¡Felicitaciones por llegar al módulo final! En este documento repasamos cómo trabajan juntas todas las partes de tu Habitación o Invernadero Inteligente y cómo alterna entre el Modo Automático y el Modo Manual desde tu celular.

1. El Trabajo en Equipo de Todo tu Sistema IoT

Tu proyecto final combina tres niveles que conversan en armonía cada 2 segundos:

Nivel · Protagonista · ¿Qué tareas realiza cada 2 segundos?

:--- · :--- · :---

1. Mundo Físico (Mesa / Maceta) · Sensores + Actuadores · El DHT siente el calor y la humedad; el LDR siente la luz; la pantalla LCD muestra los datos; los LEDs, el Buzzer y el Relé actúan.

2. Cerebro de Reflejos Rápidos · Arduino UNO R3 · Recoge todas las lecturas, actualiza la pantalla LCD, decide si encender el ventilador y envía el renglón TEMP,HUM,LUZ,VENT por el cable USB.

3. Cerebro Conectado a Internet · Raspberry Pi 4/5 · Recibe el renglón por USB, lo guarda con fecha y hora en el archivo .csv, mueve las agujas del panel Node-RED en tu celular y te escribe por Telegram.

2. Cómo Funciona el Doble Modo: Automático vs. Manual

Una de las mejores funciones de nuestro programa final es que no tienes que cambiar el código para elegir entre control automático o control manual desde el teléfono:

Orden desde el Celular · Modo Activo · Cómo se comporta tu Estación Inteligente

:---: · :--- · :---

(Al encender o al enviar 'M') · Modo Automático Inteligente · Si la temperatura es < 28 °C, el ventilador está apagado y brilla el LED Verde. Si sube a >= 28 °C, enciende solo el ventilador y el LED Rojo.

Botón 'E' *(Encender)* · Modo Manual Forzado ON · Enciende inmediatamente el ventilador sin importar cuántos grados haga (por ejemplo, si quieres ventilar el cuarto ahora mismo).

Botón 'A' *(Apagar)* · Modo Manual Forzado OFF · Apaga inmediatamente el ventilador hasta que decidas volver a presionar el botón de Modo Automático ('M').

3. ¡Lo Lograste! Qué Sabes Hacer Ahora

Al completar este curso desde cero absoluto, ya dominas:

1. Armar circuitos limpios y seguros en protoboard sin soldar.

2. Programar un Arduino UNO con luces, botones INPUT_PULLUP, sonidos, sensores analógicos/digitales, pantallas LCD I2C, servomotores y relés.

3. Preparar y usar una Raspberry Pi con su escritorio gráfico y programar en Thonny Python.

4. Comunicar Arduino y Raspberry Pi de ida y vuelta por cable USB sin riesgos de voltaje.

5. Crear paneles visuales para tu teléfono con Node-RED y recibir alertas en vivo en Telegram.

Plano y Conexiones Cable a Cable — Módulo 08: Proyecto Final Completo

[!IMPORTANT]

Guía Maestra de Cableado: Sigue esta tabla fila por fila para armar tu Habitación o Invernadero Inteligente sin olvidar ningún cable. Todos los pines 5V van al Riel Rojo (+) de la protoboard y todos los pines GND van al Riel Azul (-).

1. Tabla Maestra de Conexiones Pin a Pin del Proyecto Final

Componente · Pin del Componente · A dónde se conecta en el Arduino UNO / Protoboard · Función en el Proyecto Final

:--- · :--- · :--- · :---

Alimentación Protoboard · 5V y GND del Arduino · Riel Rojo (+) y Riel Azul (-) · Distribuye 5 Voltios y Tierra a todo el tablero.

Sensor DHT11/DHT22 · + (VCC) / OUT / - (GND) · Riel Rojo (+) / Pin Digital 4 / Riel Azul (-) · Mide Temperatura (°C) y Humedad (%).

Sensor de Luz LDR · VCC / OUT / GND · Riel Rojo (+) / Pin Analógico A0 / Riel Azul (-) · Mide la luz ambiental (0 a 1023).

Pantalla LCD 16x2 I2C · GND / VCC / SDA / SCL · Riel Azul (-) / Riel Rojo (+) / Pin A4 / Pin A5 · Muestra en vivo Temp, Hum, Luz y estado del ventilador.

Módulo Relé (Ventilador) · VCC / GND / IN · Riel Rojo (+) / Riel Azul (-) / Pin Digital 7 · Activa el ventilador cuando Temp >= 28 °C o por orden 'E'.

LED Verde (Normal) · Patita Larga (+) / Corta (-) · Pin Digital 8 / Resistencia 220 $\rightarrow$ Riel Azul (-) · Encendido cuando la temperatura es agradable y normal.

LED Rojo (Alerta Calor) · Patita Larga (+) / Corta (-) · Pin Digital 9 / Resistencia 220 $\rightarrow$ Riel Azul (-) · Encendido cuando el ventilador está activo.

Buzzer de Aviso · Positivo (+) / Negativo (-) · Pin Digital 11 / Riel Azul (-) · Emite tonos de inicio, alerta térmica y confirmación táctil.

Puente de Datos y Energía · Cable USB del Arduino UNO · Puerto USB de la Raspberry Pi · Comunica ambas placas de ida y vuelta a 9600 baudios.

2. Plano 1: Montaje en Protoboard Estilo Tinkercad (Proyecto Final Completo)

En el siguiente plano puedes ver el montaje completo en protoboard con los sensores DHT y LDR, la pantalla LCD I2C, los LEDs verde y rojo, el buzzer, el módulo relé y el enlace USB hacia la Raspberry Pi:

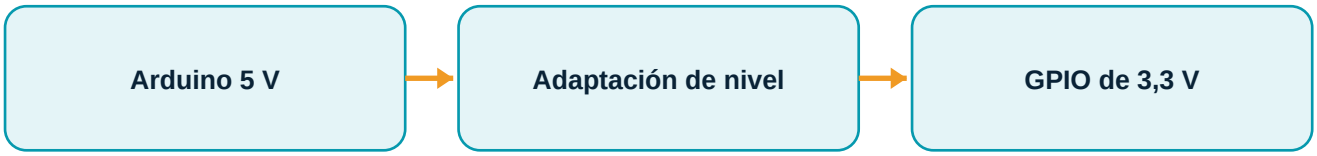
3. Plano 2: Guía Visual de Funcionamiento (Arquitectura Completa del Proyecto Final)

Observa en este diagrama el recorrido completo desde los sensores y la pantalla en tu escritorio, pasando por el puente USB hacia la Raspberry Pi, hasta el panel interactivo y las alertas de Telegram en tu celular:

Capítulo 09 · Arquitectura ciberfísica y niveles de 5 V y 3,3 V

Conceptos clave

Compara las hojas técnicas antes de unir placas. Evita aplicar 5 V directamente a un GPIO de Raspberry Pi.



Guía Paso a Paso y Listado de Materiales (BOM) — Módulo 01 (Video 01)

De los Sistemas Embebidos al IoT: Arquitectura de 4 Capas y Regla 5V vs. 3.3V

[!IMPORTANT]

Material Complementario del Video 01: Incluye el listado de materiales de la celda híbrida, los esquemas de protección lógica de $5.0\text{ V} \rightarrow 3.3\text{ V}$ (planos_circuitos/) y el script en Python [codigo_fuente/calculadora_presupuesto_iot.py](file:///D:/_Mecatrobotica/Video_01_Modulo_01_De_Sistemas_Embebidos_a_IoT/material_complementario/codigo_fuente/calculadora_presupuesto_iot.py).

1. Listado de Materiales (BOM) — Módulo 01

Componente · Especificación · Cant. · Rol en el Módulo 01

:--- · :--- · :---: · :---

Arduino UNO R3 / Nano · MCU ATmega328P (5.0 V, 16 MHz, ADC 10 bits) · 1 · Controlador determinista de Capa 1 ($< 5\text{ ms}$).

Raspberry Pi 4B / 5 · SBC ARM64, 2GB-8GB RAM, GPIO a 3.3 V · 1 · Pasarela Edge Computing de Capa 3 (Linux + Docker + MQTT).

Resistencia R_1 · $1.8\text{ k}\Omega$ (o $2.2\text{ k}\Omega$), $1/4\text{ W}$, $\pm 1\%$ · 1 · Brazo superior del divisor resistivo TX Arduino (5V) $\rightarrow$ RX RPi (3.24V).

Resistencia R_2 · $3.3\text{ k}\Omega$, $1/4\text{ W}$, $\pm 1\%$ · 1 · Brazo inferior a GND del divisor resistivo.

Conversor BSS138 · Módulo bidireccional 4 canales ($3.3\text{ V} \leftrightarrow 5.0\text{ V}$) · 1 · Adaptación segura para bus bidireccional I2C (SDA/SCL).

Transceptor MAX485 · Módulo RS-485 con terminador de $120\text{ }\Omega$ · 2 · Enlace diferencial industrial hasta 1200 m (Capa 2).

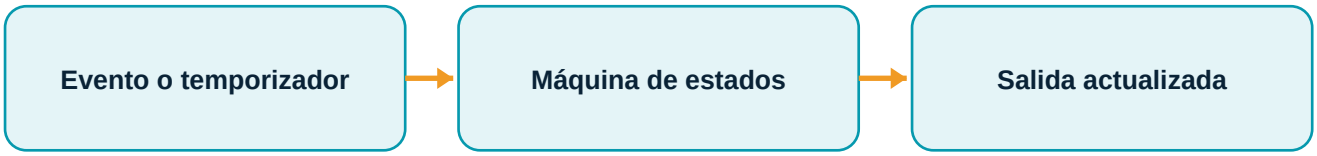
2. Ejecución de la Calculadora de Presupuesto IoT

```
python codigo_fuente/calculadora_presupuesto_iot.py
```

Capítulo 10 · Firmware no bloqueante con millis() y máquinas de estados

Conceptos clave

Calcula el tiempo transcurrido mediante resta de enteros sin signo. Modela las transiciones con condiciones explícitas.



Ejemplo complementario

```
unsigned long anterior = 0;
bool encendido = false;
void setup() { pinMode(LED_BUILTIN, OUTPUT); }
void loop() {
  const unsigned long ahora = millis();
  if (ahora - anterior >= 500UL) {
    anterior = ahora;
    encendido = !encendido;
    digitalWrite(LED_BUILTIN, encendido);
  }
}
```

Guía Paso a Paso y Listado de Materiales (BOM) — Módulo 02 (Video 02)

Firmware No Bloqueante (`millis()`), Filtro Digital EMA, NTC y Etapa MOSFET en Arduino

[!IMPORTANT]

Material Complementario del Video 02: Incluye el firmware completo de producción en C++11 (codigo_fuente/nodo_adquisicion_no_bloqueante.ino), los planos esquemáticos del divisor ratiométrico con termistor NTC y etapa de potencia con MOSFET IRLZ44N (planos_circuitos/) y el listado de componentes.

1. Listado de Materiales (BOM) — Módulo 02

Designador · Componente · Especificación · Cant. · Pin en Arduino UNO

:---: · :--- · :--- · :---: · :---

U1 · Arduino UNO R3 / Nano · ATmega328P, 16 MHz, lógica 5.0 V · 1 · —

RT1 · Termistor NTC · $10\text{ k}\Omega$ a $25\text{ }^\circ\text{C}$, $\beta = 3950\text{ K}$ · 1 · Divisor hacia A0

R_SER · Resistencia de Referencia · $10.0\text{ k}\Omega$, película metálica $\pm 1\%$, $1/4\text{ W}$ · 1 · Entre +5V y A0

Q1 · MOSFET Canal N Nivel Lógico · IRLZ44N ($V_{GS(th)} < 2.0\text{ V}$, satura a 5 V) · 1 · Gate desde Pin D9 (PWM)

R_G · Resistencia de Compuerta · $220\text{ }\Omega$, $1/4\text{ W}$ · 1 · En serie entre D9 y Gate

R_PD · Resistencia Pull-Down · $10\text{ k}\Omega$, $1/4\text{ W}$ · 1 · Entre Gate y GND

D1 · Diodo Flyback · 1N4007 o Schottky 1N5819 · 1 · En antiparalelo con ventilador 12 V

SW1 · Pulsador de Rearme · Normalmente abierto (N.A.) · 1 · Entre Pin D2 (INPUT_PULLUP) y GND

2. Procedimiento de Compilación y Prueba del Firmware

1. Abre [codigo_fuente/nodo_adquisicion_no_bloqueante.ino](file:///D:/_Mecatrobotica/Video_02_Modulo_02_Firmware_No_Bloqueante_y_Adquisicion_Arduino/material_complementario/codigo_fuente/nodo_adquisicion_no_bloqueante.ino) en Arduino IDE 2.3+ o PlatformIO.

2. Compila y carga el firmware en tu placa Arduino UNO/Nano.

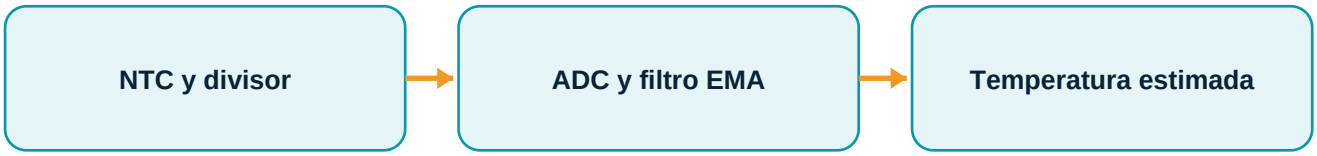
3. Abre el Monitor Serie a 115200 baudios y verifica la recepción cada 1000 ms (1 Hz) de la trama JSON con checksum XOR (*XX).

4. Envía los comandos no bloqueantes SET:SP=30.0, CMD:STOP y CMD:RESET para comprobar las transiciones de la máquina de estados finitos (FSM) y la supervisión del *Watchdog Timer* (WDTO_2S).

Capítulo 11 · Instrumentación: NTC, filtro EMA y buses de sensores

Conceptos clave

Reconstruye la resistencia a partir de la topología del divisor. Ajusta el filtro EMA considerando el equilibrio entre ruido y retraso.



Ejemplo complementario

```
def ema(muestras, alfa=0.25):
    filtrado = None
    for valor in muestras:
        filtrado = valor if filtrado is None else filtrado + alfa * (valor -
filtrado)
        yield filtrado

if __name__ == "__main__":
    print(list(ema([21, 22, 27, 22, 23])))
```

Guía Paso a Paso y Listado de Materiales (BOM) — Módulo 02 (Video 02)

Firmware No Bloqueante (`millis()`), Filtro Digital EMA, NTC y Etapa MOSFET en Arduino

[!IMPORTANT]

Material Complementario del Video 02: Incluye el firmware completo de producción en C++11 (codigo_fuente/nodo_adquisicion_no_bloqueante.ino), los planos esquemáticos del divisor ratiométrico con termistor NTC y etapa de potencia con MOSFET IRLZ44N (planos_circuitos/) y el listado de componentes.

1. Listado de Materiales (BOM) — Módulo 02

Designador · Componente · Especificación · Cant. · Pin en Arduino UNO

:---: · :--- · :--- · :---: · :---

U1 · Arduino UNO R3 / Nano · ATmega328P, 16 MHz, lógica 5.0 V · 1 · —

RT1 · Termistor NTC · $10\text{ k}\Omega$ a $25\text{ }^\circ\text{C}$, $\beta = 3950\text{ K}$ · 1 · Divisor hacia A0

R_SER · Resistencia de Referencia · $10.0\text{ k}\Omega$, película metálica $\pm 1\%$, $1/4\text{ W}$ · 1 · Entre +5V y A0

Q1 · MOSFET Canal N Nivel Lógico · IRLZ44N ($V_{GS(th)} < 2.0\text{ V}$, satura a 5 V) · 1 · Gate desde Pin D9 (PWM)

R_G · Resistencia de Compuerta · $220\text{ }\Omega$, $1/4\text{ W}$ · 1 · En serie entre D9 y Gate

R_PD · Resistencia Pull-Down · $10\text{ k}\Omega$, $1/4\text{ W}$ · 1 · Entre Gate y GND

D1 · Diodo Flyback · 1N4007 o Schottky 1N5819 · 1 · En antiparalelo con ventilador 12 V

SW1 · Pulsador de Rearme · Normalmente abierto (N.A.) · 1 · Entre Pin D2 (INPUT_PULLUP) y GND

2. Procedimiento de Compilación y Prueba del Firmware

1. Abre [codigo_fuente/nodo_adquisicion_no_bloqueante.ino](file:///D:/_Mecatrobotica/Video_02_Modulo_02_Firmware_No_Bloqueante_y_Adquisicion_Arduino/material_complementario/codigo_fuente/nodo_adquisicion_no_bloqueante.ino) en Arduino IDE 2.3+ o PlatformIO.

2. Compila y carga el firmware en tu placa Arduino UNO/Nano.

3. Abre el Monitor Serie a 115200 baudios y verifica la recepción cada 1000 ms (1 Hz) de la trama JSON con checksum XOR (*XX).

4. Envía los comandos no bloqueantes SET:SP=30.0, CMD:STOP y CMD:RESET para comprobar las transiciones de la máquina de estados finitos (FSM) y la supervisión del *Watchdog Timer* (WDTO_2S).

Capítulo 12 · Proyecto: estación climática con SSH, JSON y SQLite

Conceptos clave

Guarda la marca de tiempo y las unidades de cada lectura. Verifica el formato antes de insertar y consulta el historial con SQL parametrizado.



Ejemplo complementario

```
import json
import sqlite3

lectura = json.loads('{"t_c": 21.5, "rh_pct": 54}')
with sqlite3.connect("clima.db") as db:
    db.execute("CREATE TABLE IF NOT EXISTS clima (hora TEXT, t_c REAL, rh_pct
REAL)")
    db.execute("INSERT INTO clima VALUES (CURRENT_TIMESTAMP, ?, ?)",
               (float(lectura["t_c"]), float(lectura["rh_pct"])))
```

Capítulo 13 · Servidor Edge Linux con udev, GPIO y systemd

Conceptos clave

Identifica el equipo por atributos estables. Ejecuta el servicio con los permisos indispensables y comprueba su recuperación tras reiniciar.



Guía Paso a Paso del Laboratorio 03 — Módulo 03 (Video 03)

Despliegue Headless, Reglas `udev`, Control GPIO Moderno y Servicio Resiliente `systemd`

> [!NOTE]

> **Archivo Complementario Referenciado en el Video 03:** Esta guía contiene el procedimiento completo paso a paso para reproducir en laboratorio o en casa la configuración íntegra del servidor Edge Linux explicada en el **Video 03**. Todos los scripts listos para usar se encuentran en la carpeta [`codigo_fuente/`] (file:///D:/_Mecatrobotica/Video_03_Modulo_03_Servidor_Edge_Linux_Raspberry_Pi/material_complementario/codigo_fuente/).

Paso 1: Grabación *Headless* de Raspberry Pi OS Lite (64-bit) y Llaves SSH `Ed25519`

- En tu computador de ingeniería (Windows, Linux o macOS), abre una terminal y genera tu par de llaves criptográficas `Ed25519` si aún no dispones de una:

  ```bash
  ssh-keygen -t ed25519 -C "ingenieria-iot@mecatrobotica"
  ```

- Abre **Raspberry Pi Imager**, selecciona tu modelo de placa (**Raspberry Pi 4** o **Raspberry Pi 5**), elige **Raspberry Pi OS (other) ➔ Raspberry Pi OS Lite (64-bit)** (basado en Debian 12 *Bookworm*) y selecciona tu tarjeta microSD Clase A2.
- Haz clic en **Edit Settings (Personalización del SO)**:
- **Hostname:** `edge-gateway-01` (accesible en red local mediante `mDNS` como `edge-gateway-01.local`).
- **Username:** `iotadmin` y una contraseña robusta.
- **Wireless LAN / Ethernet:** Configura el SSID si usarás Wi-Fi o conecta cable Ethernet Cat 6.
- **Services (Pestaña Servicios):** Activa **Enable SSH**, selecciona **Allow public-key authentication only** y pega el contenido de tu archivo `~/.ssh/id_ed25519.pub`.
- Inserta la microSD en la Raspberry Pi, energízala con la fuente oficial de $5.1\ \text{V}$ y conéctate remotamente tras $40\ \text{segundos}$:

  ```bash
  ssh iotadmin@edge-gateway-01.local
  ```

Paso 2: Habilitación de Buses de Hardware (`I2C`, `SPI`, `UART`, `1-Wire`)

Una vez dentro de la sesión SSH en la Raspberry Pi, habilita los periféricos del kernel y libera el puerto serie de la consola de login ejecutando:

```
# 1. Habilitar bus I2C (/dev/i2c-1) y bus SPI (/dev/spidev0.0)
sudo raspi-config nonint do_i2c 0
sudo raspi-config nonint do_spi 0

# 2. Desactivar consola de login en UART (1 = desactivar consola) y activar hardware UART (0 = activar)
sudo raspi-config nonint do_serial_cons 1
sudo raspi-config nonint do_serial_hw 0

# 3. Habilitar bus 1-Wire en GPIO 4 (Pin físico 7)
echo "dtoverlay=w1-gpio,gpiopin=4" | sudo tee -a /boot/firmware/config.txt
```

Paso 3: Grupos de Privilegio Mínimo y Reglas `udev` Deterministas

- Otorga acceso directo a los dispositivos de hardware (`/dev/ttyACM`, `/dev/i2c-1`, `/dev/spidev`, `/dev/gpiochip*`) al usuario `iotadmin` sin necesidad de ejecutar tus programas como `root`:

  ```bash
  sudo usermod -aG dialout,i2c,spi,gpio iotadmin
  ```

- Copia el archivo complementario [`codigo_fuente/99-iot-arduinos.rules`] (file:///D:/_Mecatrobotica/Video_03_Modulo_03_Servidor_Edge_Linux_Raspberry_Pi/material_complementario/codigo_fuente/99-iot-arduinos.rules) a `/etc/udev/rules.d/` y recarga el gestor `udev`:

  ```bash
  sudo cp 99-iot-arduinos.rules /etc/udev/rules.d/99-iot-arduinos.rules
  ```

sudo udevadm control --reload-rules

sudo udevadm trigger

```

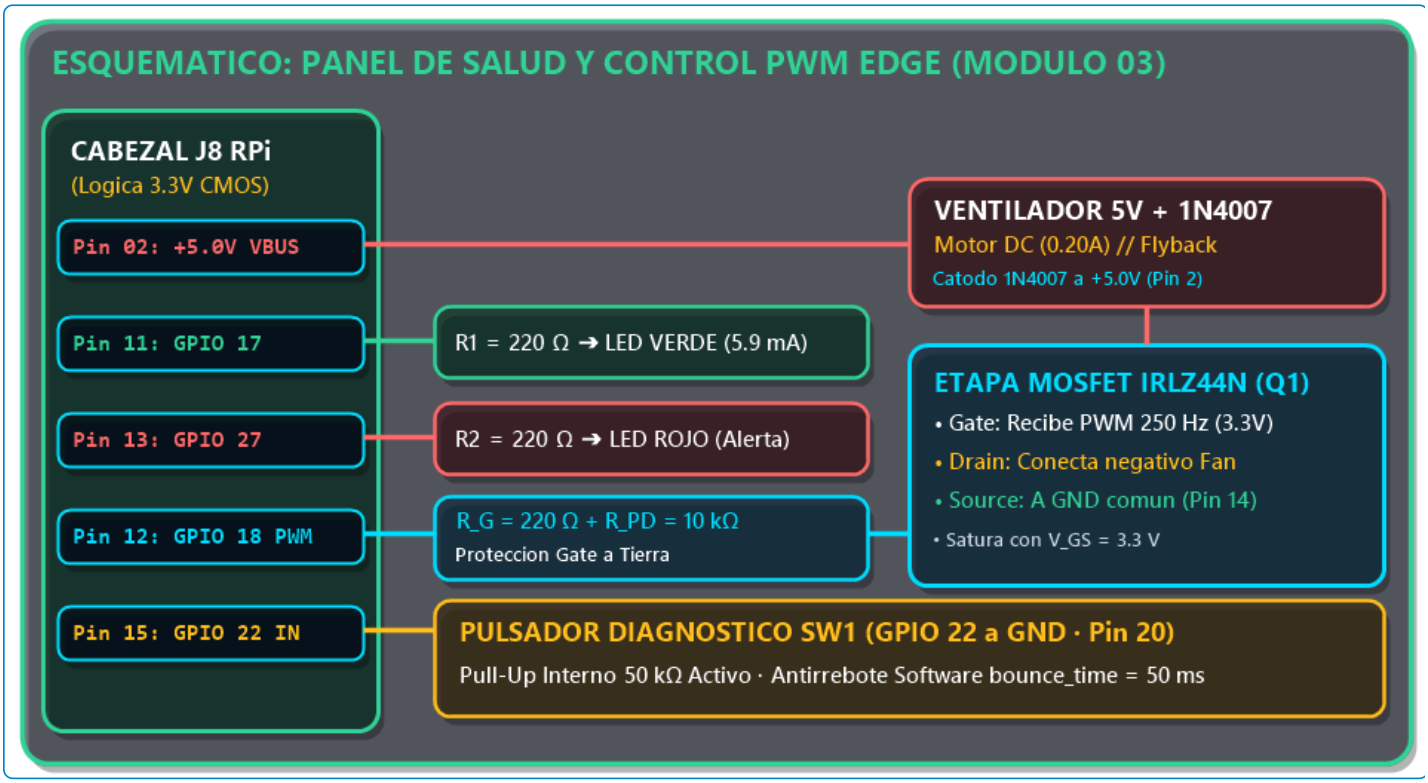
- Verifica que al conectar un Arduino UNO R3 aparezca el enlace simbólico fijo `/dev/arduino_clima`:

```bash

ls -l /dev/arduino_*

```

Paso 4: Armado del Circuito de Supervisión GPIO en Protoboard



- Conecta `GPIO 17` (**Pin físico 11**) ➜ Resistencia $220\ \Omega$ ➜ Ánodo LED Verde ➜ Cátodo a `GND` (**Pin físico 9**).
- Conecta `GPIO 27` (**Pin físico 13**) ➜ Resistencia $220\ \Omega$ ➜ Ánodo LED Rojo ➜ Cátodo a `GND` (**Pin físico 14**).
- Conecta `GPIO 18` (**Pin físico 12**) ➜ Resistencia $220\ \Omega$ ➜ Compuerta (*Gate*) del MOSFET `IRLZ44N`, añadiendo la resistencia *pull-down* de $10\ \text{k}\Omega$ entre *Gate* y `GND`.
- Conecta el ventilador de $5\ \text{V}$ entre el **Pin físico 2 ($+5.0\ \text{V}$)** y el Drenador (*Drain*) del MOSFET, colocando el diodo `1N4007` en antiparalelo (cátodo hacia $+5.0\ \text{V}$).
- Conecta el pulsador normalmente abierto `SW1` entre `GPIO 22` (**Pin físico 15**) y `GND` (**Pin físico 20**).

Paso 5: Despliegue del Demonio en Python y Servicio `systemd`

Puedes ejecutar el aprovisionamiento automático con [`codigo_fuente/provision_edge_rpi.sh`] (file:///D:/_Mecatrobotica/Video_03_Modulo_03_Servidor_Edge_Linux_Raspberry_Pi/material_complementario/codigo_fuente/provision_edge_rpi.sh) o realizar los pasos manualmente:
```

```
# 1. Instalar dependencias del sistema operativo (backend lgpio para RPi 4 y RPi 5)
sudo apt-get update
sudo apt-get install -y python3-venv python3-gpiozero python3-lgpio gpiod i2c-tools

# 2. Crear directorio de producción y entorno virtual heredando librerías C del sistema
sudo mkdir -p /opt/mecatrobotica/edge
sudo chown -R iotadmin:iotadmin /opt/mecatrobotica
python3 -m venv --system-site-packages /opt/mecatrobotica/edge/.venv

# 3. Copiar el script del demonio y otorgar permisos de ejecución
cp edge_health_daemon.py /opt/mecatrobotica/edge/edge_health_daemon.py
chmod +x /opt/mecatrobotica/edge/edge_health_daemon.py

# 4. Instalar y habilitar la unidad de servicio systemd
sudo cp edge-health.service /etc/systemd/system/edge-health.service
sudo systemctl daemon-reload
sudo systemctl enable --now edge-health.service
```

Paso 6: Pruebas de Auditoría y Resiliencia (Inyección de Fallas)

- **Inspección en Tiempo Real de `journald`:**

  ```bash
  journalctl -u edge-health.service -f
  ```

  Presiona el botón físico conectado a `GPIO 22` (Pin 15) y verifica que aparezca instantáneamente la línea `DIAGNÓSTICO MANUAL [GPIO 22] -> Temp SoC: ...`.

- **Prueba de Auto-Recuperación ante Fallo Crítico (`Crash Test`):**

  Obtén el `PID` del proceso y envíale una señal letal `SIGKILL (-9)` simulando un cuelgue abrupto:

  ```bash
  sudo kill -9 $(systemctl show --property MainPID --value edge-health.service)
  ```

  Observa en `journalctl -u edge-health.service -f` cómo `systemd` detecta la caída, espera exactamente `RestartSec=5` segundos y vuelve a levantar el servicio automáticamente restaurando el latido del LED verde.

Listado de Materiales (BOM) y Especificaciones Eléctricas — Módulo 03 (Video 03)

Preparación y Administración de Raspberry Pi como Servidor Edge Linux

> [!IMPORTANT]

> **Archivo Complementario Referenciado en el Video 03:** Este documento detalla todos los componentes electrónicos, herramientas e insumos requeridos para construir el panel local de supervisión y enfriamiento activo del **Servidor Edge Linux (** `edge-health.service` **)** explicado en la **Diapositiva 08, 09, 10 y 13** del **Video 03**.

---

1. Tabla Maestra de Materiales (Bill of Materials — BOM)

Ítem	Código / Designador	Componente	Especificación Técnica Recomendada	Cant.	Función en el Módulo 03
01	`SBC1`	**Raspberry Pi 4 Model B** o **Raspberry Pi 5**	2 GB / 4 GB / 8 GB RAM, SoC ARM64 (`BCM2711` o `BCM2712` + `RP1`), cabezal GPIO de 40 pines a $3.3\ V$	1	Computador monoplaca que actúa como Pasarela Edge Linux *headless*.
02	`PSU1`	**Fuente de Alimentación Oficial USB-C**	$5.1\ V\ DC$ estabilizados / $3.0\ A$ ($15.3\ W$ para RPi 4) o $5.0\ A$ ($27\ W$ PD para RPi 5)	1	Evita caídas de tensión ($< 4.63\ V$) y alertas de *undervoltage* (`0x50005`).
03	`SD1`	**Tarjeta microSD Industrial / Clase A2**	$32\ GB$ o $64\ GB$, estándar **Application Performance Class A2** (mínimo $4000\ IOPS$ lectura aleatoria)	1	Aloja `Raspberry Pi OS Lite (64-bit)` (Debian 12 *Bookworm*) y los diarios de `systemd-journald` .
04	`D_OK`	**LED Verde Difuso de 5 mm**	Tensión directa $V_F \approx 2.0\ V$, corriente de operación segura $I_F \approx 6\ mA$	1	Indicador de latido (*Heartbeat* a $0.5\ Hz$, $200\ ms$ encendido / $1800\ ms$ apagado) en `GPIO 17` (Pin físico 11).
05	`D_ALM`	**LED Rojo Difuso de 5 mm**	Tensión directa $V_F \approx 1.9\ V$, corriente de operación segura $I_F \approx 6.4\ mA$	1	Indicador de alerta por sobretemperatura del SoC ($T_{CPU} \geq 65\ ^\circ C$) en `GPIO 27` (Pin físico 13).
06	`R1, R2`	**Resistencias Limitadoras para LEDs**	$220\ \Omega$ (o $330\ \Omega$), película metálica $1/4\ W$, tolerancia $\pm 1\%$	2	Limitan la corriente extraída de los pines `GPIO 17` y `GPIO 27` muy por debajo del límite absoluto de $16\ mA$.
07	`Q1`	**Transistor MOSFET Canal N de Nivel Lógico**	`IRLZ44N` o `2N7000` / `BS170` (Alt. BJT NPN `2N2222A`), saturación plena con $V_{GS} = 3.3\ V$	1	Conmuta por modulación de ancho de pulso (`PWM` a $250\ Hz$) la corriente del ventilador desde `GPIO 18` (Pin físico 12).
08	`R_GATE`	**Resistencia de Compuerta / Base**	$220\ \Omega$ (para MOSFET) o $1\ k\Omega$ (para BJT `2N2222A`), $1/4\ W$, $\pm 1\%$	1	Protege el pin `GPIO 18` frente a transitorios de carga capacitiva durante la conmutación PWM.
09	`R_PD`	**Resistencia* Pull-Down *de Seguridad**	$10\ k\Omega$, $1/4\ W$, $\pm 1\%$	1	Mantiene la compuerta de `Q1` a $0\ V$ (`GND`) mientras la Raspberry Pi arranca el kernel Linux.
10	`D_FLY`	**Diodo Rectificador* Flyback***	`1N4007` ($1000\ V$, $1\ A$) o Schottky `1N5819`	1	Conectado en antiparalelo con el ventilador para absorber el pico inductivo ($V = -L\ (di)\ /\ (dt)$).
11	`FAN1`	**Ventilador Axial Brushless DC**	$5\ V\ DC$ ($0.15\ A$ - $0.25\ A$), $30\times 30\ mm$ o $40\times 40\ mm$ para gabinete Edge	1	Disipación térmica activa del procesador ARM gobernada por la curva proporcional en `edge_health_daemon.py` .
12	`SW1`	**Pulsador Táctil Normalmente Abierto (N.A.)**	$6\times 6\ mm$ o pulsador de panel industrial	1	Disparo manual de reporte de diagnóstico hacia `journald` conectado entre `GPIO 22` (Pin físico 15) y `GND` .
13	`C_DEB`	**Condensador Cerámico**	$100\ nF$ ($0.1\ \mu F$, código `104`), $50\ V$	1	Desacoplo de ruido y filtrado físico complementario al antirrebote por software (`bounce_time=0.05`).

Ítem	Código / Designador	Componente	Especificación Técnica Recomendada	Cant.	Función en el Módulo 03
		Multicapa (Opcional)			
14	`PB1`	**Protoboard y Cables DuPont**	Protoboard de 400 u 830 puntos + cables DuPont Hembra-Macho de $20\ \text{cm}$	1 kit	Interconexión limpia entre el cabezal de 40 pines de la Raspberry Pi y la etapa de supervisión.
15	`INST1`	**Multímetro Digital True-RMS**	Medición de continuidad, tensión DC y resistencia	1	Verificación obligatoria de ausencia de cortocircuitos entre los rieles de $5.0\ \text{V}$ (Pines 2/4) y $3.3\ \text{V}$ (Pin 1).

---

2. Cálculos de Dimensionamiento Eléctrico del Cabezal GPIO ($3.3\ \text{V}$)

2.1. Límite de Corriente del SoC Broadcom / Chip `RP1`

A diferencia del Arduino UNO (cuyos pines a $5.0\ \text{V}$ toleran hasta $20\ \text{mA}$ nominales), los buffers CMOS de $3.3\ \text{V}$ de la Raspberry Pi tienen restricciones estrictas:

- **Corriente máxima por pin individual:** $\text{I}_{\text{pin(máx)}} \leq 16\ \text{mA}$ (recomendado diseñar para $\text{I}_{\text{pin}} \leq 8\ \text{mA}$).
- **Corriente total combinada en todo el banco GPIO de $3.3\ \text{V}$:** $\text{I}_{\text{banco(máx)}} \leq 50\ \text{mA}$.

2.2. Cálculo de las Resistencias Limitadoras `R1` y `R2` ($220\ \Omega$)

Aplicando la Ley de Tensiones de Kirchhoff al lazo del pin `GPIO 17` ($\text{V}_{\text{GPIO}} = 3.3\ \text{V}$) con el LED verde ($\text{V_F} = 2.0\ \text{V}$):

$$
I_{LED} = \frac{V_{GPIO} - V_F}{R_1} = (3.3\ V - 2.0\ V) / (220\ \Omega) = (1.3\ V) / (220\ \Omega) \approx 5.91\ mA < 16\ mA
$$

La potencia disipada en la resistencia R_1 es:

$$
P_{R1} = I_{LED}^2 \cdot R_1 = (0.00591\ A)^2 \cdot 220\ \Omega \approx 7.68\ mW \ll 250\ mW\ (1/4\ W)
$$

Plano de Circuito y Mapa del Cabezal GPIO de 40 Pines — Módulo 03 (Video 03)

Interconexión Eléctrica del Demonio de Salud Edge (`edge_health_daemon.py`)

> [!IMPORTANT]



---

1. Mapa de Pines Críticos en el Cabezal J8 de 40 Pines (Numeración Física vs. BCM)

> [!CAUTION]

> **REGLA DE ORO DEL CABEZAL DE 40 PINES:** Los **Pines Físicos 2 y 4** entregan $5.0\ V\ DC$ **directos de la fuente USB-C** sin pasar por ningún limitador lógico. Jamás conectes los Pines 2 o 4 a ningún pin `GPIO`, ya que destruirás instantáneamente el SoC o el controlador `RP1` de la Raspberry Pi.

Pin Físico (Impar - Izquierda)	Función / Nombre BCM (Impar)	Función / Nombre BCM (Par)	Pin Físico (Par - Derecha)
Pin 1	`3.3V DC Power` (Referencia Lógica)	`5.0V DC Power` (Riel de Potencia para Ventilador)	**Pin 2**
Pin 3	`GPIO 2` (`I2C1 SDA` — Pull-up 1.8 kΩ interno)	`5.0V DC Power`	**Pin 4**
Pin 5	`GPIO 3` (`I2C1 SCL` — Pull-up 1.8 kΩ interno)	`GND` (Tierra Común Principal)	**Pin 6**
Pin 7	`GPIO 4` (`1-Wire` para sondas `DS18B20`)	`GPIO 14` (`UART0 TXD` @ 3.3 V)	**Pin 8**
Pin 9	`GND` (Retorno para LEDs de Estado)	`GPIO 15` (`UART0 RXD` @ 3.3 V)	**Pin 10**
Pin 11	`GPIO 17` **→ Salida LED Verde (** `Heartbeat` **)**	`GPIO 18` (`PWM0`) **→ Gate MOSFET Ventilador**	**Pin 12**
Pin 13	`GPIO 27` **→ Salida LED Rojo (** `Alerta Temp` **)**	`GND` (Retorno para Etapa MOSFET)	**Pin 14**
Pin 15	`GPIO 22` **→ Entrada Botón Diagnóstico (** `Pull-Up` **)**	`GPIO 23` (Libre)	**Pin 16**
Pin 17	`3.3V DC Power`	`GPIO 24` (Libre)	**Pin 18**
Pin 19	`GPIO 10` (`SPI0 MOSI`)	`GND` (Retorno para Botón `SW1`)	**Pin 20**

2. Diagrama Esquemático de Conexiones del Laboratorio 03

```
flowchart LR
    subgraph RPi["Cabezal J8 de 40 Pines — Raspberry Pi 4B / 5 (Lógica 3.3V)"]
        P2["Pin Físico 2<br/>+5.0V DC (V_BUS)"]
        P11["Pin Físico 11<br/>GPIO 17 (BCM)"]
        P13["Pin Físico 13<br/>GPIO 27 (BCM)"]
        P12["Pin Físico 12<br/>GPIO 18 (PWM)"]
        P15["Pin Físico 15<br/>GPIO 22 (Pull-Up 50kΩ)"]
        GND["Pines Físicos 6, 9, 14, 20<br/>GND (Tierra Común 0V)"]
    end

    subgraph StageLEDs["1. Indicadores de Estado (I_pin ≈ 6 mA)"]
        P11 --> R1["R1 = 220 Ω (1/4W)"] --> LED_G["LED Verde (Heartbeat 0.5 Hz)<br/>Ánodo (+) ➔ Cátodo (-)"] --> GND
        P13 --> R2["R2 = 220 Ω (1/4W)"] --> LED_R["LED Rojo (Alerta ≥ 65 °C)<br/>Ánodo (+) ➔ Cátodo (-)"] --> GND
    end

    subgraph StageFan["2. Etapa de Potencia PWM para Ventilador (250 Hz)"]
        P12 --> RG["R_GATE = 220 Ω"] --> GATE["Compuerta (Gate)<br/>MOSFET IRLZ44N (Q1)"]
        GATE --> RPD["R_PD = 10 kΩ (Pull-Down)"] --> GND
        P2 --> FAN["Ventilador DC 5V (+)"]
        P2 --> D_CATH["Cátodo Diodo 1N4007 (Flyback)"]
        FAN --> DRAIN["Drenador (Drain) Q1"]
        DRAIN --> D_ANOD["Ánodo Diodo 1N4007"]
        DRAIN --> SOURCE["Fuente (Source) Q1"] --> GND
    end

    subgraph StageBtn["3. Entrada de Diagnóstico con Antirrebote"]
        P15 --> SW1["Pulsador N.A. (SW1)<br/>bounce_time = 50 ms"] --> GND
    end
```

---

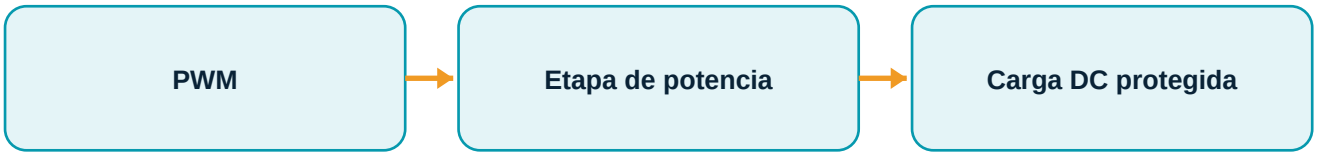
3. Tabla de Verificación Punto a Punto con Multímetro (Antes de Energizar)

- **Prueba de Aislamiento de Rieles ($5.0\ \text{V}$ vs. $3.3\ \text{V}$):** Medir resistencia entre el **Pin Físico 1 ($3.3\ \text{V}$)** y el **Pin Físico 2 ($5.0\ \text{V}$)** con la Raspberry Pi apagada. Debe marcar alta impedancia ($> 10\ \text{k}\Omega$).
- *Prueba de Polaridad del Diodo* Flyback *(`1N4007`): **Verificar que la** banda plateada (cátodo) **del diodo** `1N4007` **esté conectada al positivo de** $+5.0\ \text{V}$ (Pin 2) **y el ánodo al** Drenador (*Drain)* del MOSFET. Si se invierte, producirá un cortocircuito directo cuando el MOSFET conduzca.
- *Prueba de Resistencia* Pull-Down *de Compuerta: Medir entre la compuerta (*Gate*) del MOSFET y `GND` (Pin 14); el multímetro debe registrar $\approx 10.0\ \text{k}\Omega$.

Capítulo 14 · Proyecto de potencia: MOSFET, PWM, relés y flyback

Conceptos clave

Dimensiona el transistor para tensión, corriente y disipación. En cargas inductivas de corriente continua se necesita una trayectoria de descarga adecuada.



Internet de las Cosas con Arduino y Raspberry Pi

Capítulo 15 · UART, I²C, SPI y tramas CRC-16

Conceptos clave

Elige el bus según distancia, velocidad y cantidad de dispositivos. Comprueba orden de bytes, variante de CRC y encuadre con vectores de prueba.



Ejemplo complementario

```
def crc16_modbus(datos):
    crc = 0xFFFF
    for byte in datos:
        crc ^= byte
        for _ in range(8):
            crc = (crc >> 1) ^ 0xA001 if crc & 1 else crc >> 1
    return crc

assert crc16_modbus(b"123456789") == 0x4B37
```

Guía Paso a Paso — Laboratorio 04: Comunicación Directa Arduino ↔ Raspberry Pi (UART, I²C, SPI y Bus Industrial RS-485 / Modbus RTU)

Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 1, 3, 5, 9, 10, 12, 13 y 14

---

1. Objetivo de Ingeniería del Laboratorio

Integrar el nodo controlador de tiempo real basado en **Arduino UNO (`5.0 V`)** con la computadora de borde **Raspberry Pi (`3.3 V`)** analizando cuatro tecnologías de enlace físico y lógico:

- **Enlace Serie Asíncrono (`UART`)**: Comprensión del fenómeno de reinicio automático por línea `DTR` a través del condensador de acoplamiento de $100\ \text{nF}$ y aseguramiento de integridad de trama mediante **verificación de redundancia cíclica (`CRC-16/MODBUS`)**.
- **Buses Síncronos de Corto Alcance (`I²C` y `SPI`)**: Traslación segura de nivel lógico bidireccional con transistores MOSFET de canal N `BSS138` y desactivación por software de las resistencias internas de elevación a `5.0 V` del microcontrolador AVR.
- **Capa Física Diferencial Industrial (`RS-485`)**: Cableado en cadena (*Daisy-Chain*) con par trenzado blindado, cálculo e instalación de resistencias terminadoras de $120\ \Omega$ únicamente en los dos extremos físicos de la línea troncal ($R_{eq} = 60\ \Omega$) y control de dirección semidúplex (`DE` / `/RE`) en el transceptor `MAX485`.
- **Protocolo Industrial `Modbus RTU`**: Exposición del nodo de invernadero como Esclavo `ID = 1` a `19200 bps` en formato `8E1` (8 bits de datos, paridad Par, 1 bit de parada), implementando las funciones `0x03` (Lectura de Registros de Retención) y `0x06` (Escritura de Registro Individual) sobre 6 registros de $16\ \text{bits}$ con escalado en punto fijo.

---

2. Etapa 1: Enlace UART y Prevención del Auto-Reset por Línea DTR

2.1 Diagnóstico del Fenómeno de Auto-Reset (`100 nF`)

Al conectar el Arduino UNO por cable USB a la Raspberry Pi (`/dev/ttyACM0`), el chip conversor USB-Serie controla la línea `DTR` (*Data Terminal Ready*), la cual está acoplada al pin `RESET` del ATmega328P mediante un condensador cerámico en serie de $C_{DTR} = 100\ \text{nF}$:

- Cada vez que un proceso en Linux abre el descriptor del puerto serie (`serial.Serial()`), la señal `DTR` cae de $5.0\ \text{V}$ a $0\ \text{V}$.
- El condensador de $100\ \text{nF}$ transmite ese escalón negativo al pin `RESET`, reiniciando el microcontrolador y activando el cargador de arranque (*Bootloader*) durante aproximadamente $1.5\ \text{s}$ a $1.8\ \text{s}$.
- **Regla de Oro de Software**: Mantener el puerto serie abierto de forma persistente durante toda la vida del servicio, aguardar `time.sleep(2.0)` tras su apertura inicial y vaciar las colas de recepción (`reset_input_buffer()`) antes de emitir la primera consulta.

2.2 Alternativa de Enlace UART Directo en Cabezal GPIO (`/dev/serial0`)

Si se enlazan directamente los pines `TX` / `RX` sin pasar por USB:

- Conectar el pin `D1 (TX, 5.0 V)` del Arduino hacia una resistencia en serie $R_1 = 1.8\ \text{k}\Omega$, y derivar desde el nodo central una resistencia $R_2 = 3.3\ \text{k}\Omega$ hacia `GND`.
- La tensión resultante entregada al **Pin Físico 10 (`GPIO 15 / RXD`)** de la Raspberry Pi es:

> $V_{RX,RPi} = 5.0\ V \cdot (3.3\ k\Omega) / (1.8\ k\Omega + 3.3\ k\Omega) = 3.235\ V < 3.30\ V$

---

3. Etapa 2: Coprocesador en Bus I²C con Conversor Bidireccional BSS138

Cuando el Arduino UNO y la Raspberry Pi residen dentro del mismo gabinete eléctrico a menos de $20\ cm$:

- **Cableado del Adaptador `BSS138`**:
- Lado de Baja Tensión (`LV`): Conectar `LV` al **Pin 01 (`+3.3V`)** de la Raspberry Pi, `A1` a **`GPIO 2 (SDA, Pin 03)`**, `A2` a **`GPIO 3 (SCL, Pin 05)`** y `GND` al **Pin 06 (`GND`)**.
- Lado de Alta Tensión (`HV`): Conectar `HV` al pin **`5V`** del Arduino UNO, `B1` al pin **`A4 (SDA)`**, `B2` al pin **`A5 (SCL)`** y `GND` a tierra común.
- **Protección del Dominio de `3.3 V` en el Firmware del Arduino (`coprocesador_i2c_esclavo.ino`)**:
- Inmediatamente después de invocar `Wire.begin(0x08)`, la biblioteca estándar de Arduino activa por defecto las resistencias internas de elevación a $+5.0\ V$ en los pines `A4` y `A5`.
- **Acción obligatoria**: Ejecutar `digitalWrite(SDA, LOW)` y `digitalWrite(SCL, LOW)` justo después de `Wire.begin(0x08)` para desactivar los *pull-ups* internos a $5\ V$.
- **Verificación en la Raspberry Pi**:
- Habilitar el bus `I²C` con `sudo raspi-config nonint do_i2c 0`.
- Ejecutar `i2cdetect -y 1` y comprobar que aparece la dirección hexadecimal **`08`** en la matriz de escrutinio.

---

4. Etapa 3: Montaje del Bus Diferencial Industrial RS-485 (`MAX485`)

Para distancias de $10\ m$ a $1200\ m$ en entornos con motores, variadores de frecuencia y contactores:

4.1 Conexión entre Arduino UNO y Transceptor `MAX485`

- **Alimentación**: Conectar `VCC` del módulo `MAX485` a `+5.0 V` del Arduino y `GND` a tierra común (con condensador de desacoplo de $100\ nF$ entre `VCC` y `GND`).
- **Señales UART**:
- Pin **`RO`** (*Receiver Output*) del `MAX485` → Pin **`D0 (RX)`** del Arduino UNO.
- Pin **`DI`** (*Driver Input*) del `MAX485` → Pin **`D1 (TX)`** del Arduino UNO.
- **Control de Dirección Semidúplex (`DE` + `/RE`)**:
- Unir físicamente los pines **`DE`** (*Driver Enable*, activo en alto) y **`/RE`** (*Receiver Enable*, activo en bajo) del módulo `MAX485`.
- Conectar ambos pines unidos al **Pin Digital `D4`** del Arduino UNO, añadiendo una resistencia *pull-down* de $10\ k\Omega$ hacia `GND` para garantizar que el nodo permanezca en modo escucha (`LOW`) mientras el microcontrolador arranca.

4.2 Cableado de la Línea Troncal Diferencial (`A+` , `B-` y `GND`)

- Utilizar **cable de par trenzado blindado** de impedancia característica $Z_0 = 120\ \Omega$.
- Unir el borne **`A (+)`** del adaptador USB-RS485 de la Raspberry Pi con el borne **`A (+)`** del módulo `MAX485` del Arduino.
- Unir el borne **`B (-)`** del adaptador USB-RS485 con el borne **`B (-)`** del módulo `MAX485`.
- Unir la referencia común **`GND`** entre todos los nodos para mantener la tensión de modo común dentro del margen operativo del `MAX485` ($-7\ V$ a $+12\ V$).

- **Instalación de Resistencias Terminadoras ($120\ \Omega$)**:
- Instalar una resistencia $R_{term,1} = 120\ \Omega\ (1/4\ W)$ entre `A(+)` y `B(-)` en el **Extremo Inicial** (adaptador de la Raspberry Pi).
- Instalar una segunda resistencia $R_{term,2} = 120\ \Omega\ (1/4\ W)$ entre `A(+)` y `B(-)` en el **Extremo Final** del cable troncal.
- **Verificación con óhmetro (con el sistema apagado)**: Medir la resistencia equivalente entre `A(+)` y `B(-)`; debe leerse $R_{eq} = 120\ \Omega\ ||\ 120\ \Omega = 60\ \Omega\ \pm 5\%$. **Jamás** colocar resistencias de $120\ \Omega$ en nodos intermedios.

---

5. Etapa 4: Puesta en Marcha del Enlace Maestro-Esclavo Modbus RTU

5.1 Carga del Firmware Esclavo en Arduino (`esclavo_modbus_rtu_arduino.ino`)

- Desconectar temporalmente el pin `D0 (RX)` del módulo `MAX485` mientras se sube el programa por USB al Arduino UNO.
- Compilar y cargar `esclavo_modbus_rtu_arduino.ino`. El firmware configura el puerto serie a **`19200 bps`** con formato **`SERIAL_8E1`** (8 bits de datos, paridad Par, 1 bit de parada) y asigna la dirección **`SLAVE_ID = 1`**.
- Reconectar el pin `D0 (RX)` al pin `RO` del módulo `MAX485`.

5.2 Ejecución del Maestro en Raspberry Pi (`maestro_modbus_rtu_raspberry.py`)

- Activar el entorno virtual Python en la Raspberry Pi e instalar las bibliotecas requeridas:

  ```bash
  source ~/iot_edge_node/venv/bin/activate
  pip install minimalmodbus pyserial
  ```

- Ejecutar el programa maestro:

  ```bash
  python3 maestro_modbus_rtu_raspberry.py --port /dev/ttyUSB0 --slave-id 1
  ```

- Verificar en consola la lectura cíclica de los 6 registros de retención (`0x0000` a `0x0005`) y probar la escritura remota de una nueva consigna térmica con la función **`0x06`**:

  ```bash
  python3 maestro_modbus_rtu_raspberry.py --port /dev/ttyUSB0 --setpoint 26.50
  ```

---

6. Lista de Verificación de Diagnóstico de Fallos

Síntoma Observado	Causa Raíz Probable	Solución de Ingeniería
El último byte de respuesta del Arduino llega corrupto (error de `CRC-16`)	El pin `D4` (`DE` / `/RE`) pasa a `LOW` antes de que el registro de desplazamiento UART termine de emitir los bits físicos	Invocar obligatoriamente `Serial.flush()` más una guarda de `delayMicroseconds(1820)` antes de `digitalWrite(4, LOW)`
Error `No communication with the instrument` en `minimalmodbus`	Líneas diferenciales `A(+)` y `B(-)` invertidas o formato de paridad distinto entre Maestro y Esclavo	Verificar polaridad `A(+)` con `A(+)`, `B(-)` con `B(-)` y confirmar `19200 bps` en formato **`8E1`** en ambos extremos
Sobrecalentamiento de los transceptores `MAX485` y caída de tensión diferencial	Múltiples nodos intermedios tienen instalada su resistencia SMD de $120\ \Omega$ de fábrica ($R_{eq} \ll 60\ \Omega$)	Desoldar la resistencia `R7` ($120\ \Omega$) de todos los módulos `MAX485` ubicados en posiciones intermedias del bus
El Arduino se reinicia cada vez que se ejecuta el script en Python por USB	Pulso negativo en la línea `DTR` a través del condensador de $100\ nF$	Mantener la instancia de puerto abierta dentro de un bucle continuo y aguardar $2.0\ s$ tras abrir el descriptor

Listado de Materiales (BOM) — Módulo 04: Comunicación Directa UART, I²C, SPI, RS-485 y Modbus RTU

Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 3, 5, 6, 8, 9, 10 y 14

---

1. Tabla Principal de Componentes de Hardware y Comunicaciones

Ítem	Cant.	Designador	Componente / Descripción Técnica	Especificación Eléctrica / Mecánica	Función en el Módulo 04
01	1	`SBC1`	**Raspberry Pi 4 Model B** (o RPi 5)	SoC ARM64 (`3.3 V` CMOS), puertos USB y cabezal GPIO de 40 pines	Nodo Maestro `Modbus RTU` / Maestro `I²C` y `SPI`
02	1	`U1`	**Arduino UNO R3** (`ATmega328P`)	Lógica TTL de `5.0 V`, reloj de $16\ \text{MHz}$, `UART` hardware (`D0` / `D1`), `I²C` (`A4` / `A5`)	Nodo Esclavo `Modbus RTU` (`ID = 1`) / Coprocesador `I²C` (`0x08`)
03	1	`U2`	**Adaptador Industrial USB a RS-485**	Chip `FT232RL` + `MAX485` (o `CH340E`), protección `TVS` contra transitorios y bornes `A(+)` , `B(-)` , `GND`	Interfaz física `RS-485` en la Raspberry Pi (`/dev/ttyUSB0`)
04	2	`U3, U4`	**Módulo Transceptor RS-485 (`MAX485CSA`)**	Alimentación `5.0 V` , semidúplex hasta $2.5\ \text{Mbps}$, hasta 32 cargas unitarias (`1 UL = 12 kΩ`)	Conversión entre `UART` TTL (`RO` , `DI` , `DE` / `/RE`) y par diferencial `RS-485`
05	1	`U5`	**Conversor Bidireccional de Nivel Lógico `BSS138` (4 Canales)**	4 transistores MOSFET Canal N `BSS138` con resistencias *pull-up* de $10\ \text{k}\Omega$ en `LV` (`3.3 V`) y `HV` (`5.0 V`)	Adaptación bidireccional de colector abierto para buses `I²C` (`SDA` / `SCL`) y `SPI`
06	2	`R_TERM1, R_TERM2`	**Resistencias Terminadoras de Línea RS-485**	$120\ \Omega\ \pm\ 1\%$, película metálica, potencia de $1/4\ \text{W}$ ($0.25\ \text{W}$)	Adaptación de impedancia característica ($Z_0 = 120\ \Omega$) en los 2 extremos físicos del bus
07	1	`R_PD`	**Resistencia Pull-Down de Habilitación TX**	$10\ \text{k}\Omega\ \pm\ 1\%$, $1/4\ \text{W}$	Mantiene los pines `DE` / `/RE` (`Pin D4`) en modo escucha (`0 V`) durante el arranque del AVR
08	2	`R_PU1, R_PU2`	**Resistencias Pull-Up Externas para Bus I²C**	$4.7\ \text{k}\Omega\ \pm\ 1\%$, $1/4\ \text{W}$	Ajuste del tiempo de subida ($\tau = R_p \cdot C_{bus}$) hacia el riel de `3.3 V` si hay múltiples esclavos
09	1	`R1, R2`	**Par de Resistencias para Divisor UART Directo**	$R_1 = 1.8\ \text{k}\Omega$ y $R_2 = 3.3\ \text{k}\Omega$ ($\pm\ 1\%$, $1/4\ \text{W}$)	Atenuación de la línea `TX` de `5.0 V` del Arduino a $3.24\ \text{V}$ hacia `GPIO 15 (RXD)`
10	2	`C_DEC1, C_DEC2`	**Condensadores de Desacoplo Cerámicos**	$100\ \text{nF}$ ($0.1\ \mu\text{F}$), dieléctrico `X7R` , $50\ \text{V}$	Filtrado de picos de corriente en el pin `VCC` de cada transceptor `MAX485`
11	5 m	`CAB1`	**Cable de Par Trenzado Blindado (`STP` / `RS-485`)**	Impedancia característica $Z_0 = 120\ \Omega$, calibre `24 AWG` con malla de drenaje	Línea troncal diferencial para interconexión en cadena (*Daisy-Chain*)

2. Cálculos de Verificación Eléctrica

2.1 Impedancia Equivalente y Corriente en el Bus RS-485

Con únicamente las dos resistencias terminadoras de $120\ \Omega$ instaladas en los extremos inicial y final de la línea troncal:

$$R_{eq} = \frac{R_{term,1} \cdot R_{term,2}}{R_{term,1} + R_{term,2}} = (120\ \Omega \cdot 120\ \Omega) / (120\ \Omega + 120\ \Omega) = 60\ \Omega$$

Para una tensión diferencial típica de salida $V_{OD} = 2.0\ V$ entregada por el transmisor `MAX485`:

- **Corriente en el bucle diferencial**: $I_{bus} = (2.0\ V) / (60\ \Omega) = 33.3\ mA$ (dentro del límite nominal de $60\ mA$ del `MAX485`).
- **Potencia disipada en cada terminador**: $P_{term} = ((2.0\ V)^2) / (120\ \Omega) = 33.3\ mW \ll 250\ mW$ ($1/4\ W$).

2.2 Tiempo de Carácter y Silencio de Trama Modbus RTU a 19200 bps (`8E1`)

Cada carácter en formato `8E1` consta de $11\ bits$ ($1\ Start + 8\ Datos + 1\ Paridad\ Par + 1\ Stop$):

$$T_{char} = (11\ bits) / (19200\ bits/s) = 572.92\ \mu s \quad\Longrightarrow\quad T_{3.5\ char} = 3.5 \times 572.92\ \mu s \approx 2005\ \mu s \approx 2.0\ ms$$

Manual Técnico del Protocolo Modbus RTU y Verificación CRC-16 (Módulo 04)

Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 4, 11, 12, 13 y 14

---

1. Fundamentos Matemáticos del Código de Redundancia Cíclica `CRC-16/MODBUS`

En enlaces serie asíncronos (`UART` y `RS-485`) expuestos a ruido electromagnético industrial, un simple bit de paridad o una suma aritmética de bytes (*Checksum*) no detecta inversiones simétricas de bits ni transposiciones de octetos. El estándar `Modbus RTU` emplea el polinomio generador algebraico de grado 16:

$$G(x) = x^{16} + x^{15} + x^2 + 1$$

1.1 Parámetros Canónicos del Algoritmo `CRC-16/MODBUS`

- **Representación normal del polinomio**: `0x8005` ($1000\,0000\,0000\,0101_2$).
- **Representación reflejada (procesamiento LSB primero)**: `0xA001` ($1010\,0000\,0000\,0001_2$).
- **Valor semilla inicial (`Init`)**: `0xFFFF` .
- **Orden de transmisión en la línea**: Primero se envía el byte menos significativo (`CRC_Lo`) y en último lugar el byte más significativo (`CRC_Hi`).
- **Capacidad de detección**: Detecta el 100% de los errores de 1, 2 y número impar de bits, todas las ráfagas de error de hasta 16 bits de longitud, y el 99.9985% ($1 - 2^{-16}$) de ráfagas mayores.

1.2 Algoritmo Iterativo Paso a Paso (C++ / Python)

Para una trama de N bytes de carga útil $B_0, B_1, \dots, B_{N-1}$:

- Inicializar el acumulador de 16 bits: $CRC \leftarrow 0xFFFF$.
- Para cada byte B_k ($k = 0 \dots N-1$):
- Operar $CRC \leftarrow CRC \oplus B_k$.
- Repetir 8 veces (para cada bit $i = 0 \dots 7$):
- Si el bit menos significativo $(CRC \,\&\, 0x0001) = 1$, entonces: $CRC \leftarrow (CRC >> 1) \oplus 0xA001$.
- En caso contrario: $CRC \leftarrow (CRC >> 1)$.
- Al finalizar, adjuntar $CRC_{Lo} = CRC \,\&\, 0xFF$ seguido de $CRC_{Hi} = (CRC >> 8) \,\&\, 0xFF$.

---

2. Estructura de Trama Binaria y Temporización `Modbus RTU`

A diferencia de formatos textuales como JSON o CSV, `Modbus RTU` transmite octetos binarios puros delimitados exclusivamente por **intervalos de silencio en la línea**:

Campo de la Trama	Tamaño	Codificación	Descripción Técnica
Silencio Inicial	≥ 3.5 char	Línea en reposo (`Idle`)	A `19200 bps` (`8E1`), equivale a ≥ 2.0 ms sin actividad
Dirección de Esclavo (`ID`)	1 Byte	`0x01` a `0xF7` (1\dots 247)	Identifica al nodo destinatario (`0x00` reservado para difusión *Broadcast*)
Código de Función (`FC`)	1 Byte	`0x03` o `0x06`	`0x03` : Leer Registros de Retención · `0x06` : Escribir Registro Individual
Campo de Datos (`Payload`)	N Bytes	**Big-Endian** (`MSB` primero)	Direcciones de registro de 16 bits, cantidad de palabras y valores escalados
Verificación `CRC-16`	2 Bytes	**Little-Endian** (`LSB` primero)	Primero se transmite `CRC_Lo` y finalmente `CRC_Hi`
Silencio Final	≥ 3.5 char	Línea en reposo (`Idle`)	Cierra la trama actual y habilita el bus para la siguiente transacción

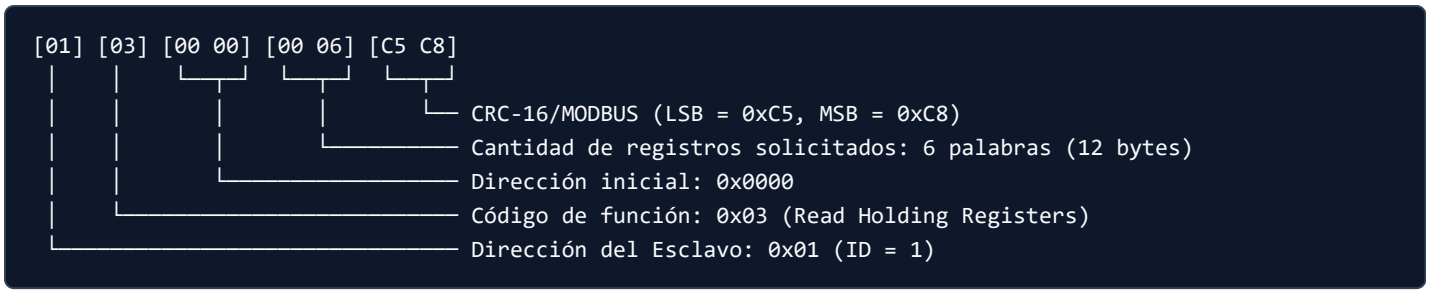
3. Mapa Oficial de Registros de Retención del Nodo de Invernadero (`ID = 1`)

Para evitar el envío de números en coma flotante (`float` IEEE-754 de 32 bits) y problemas de orden de palabras, todas las magnitudes continuas se transmiten como **enteros de 16 bits con escalado en punto fijo**:

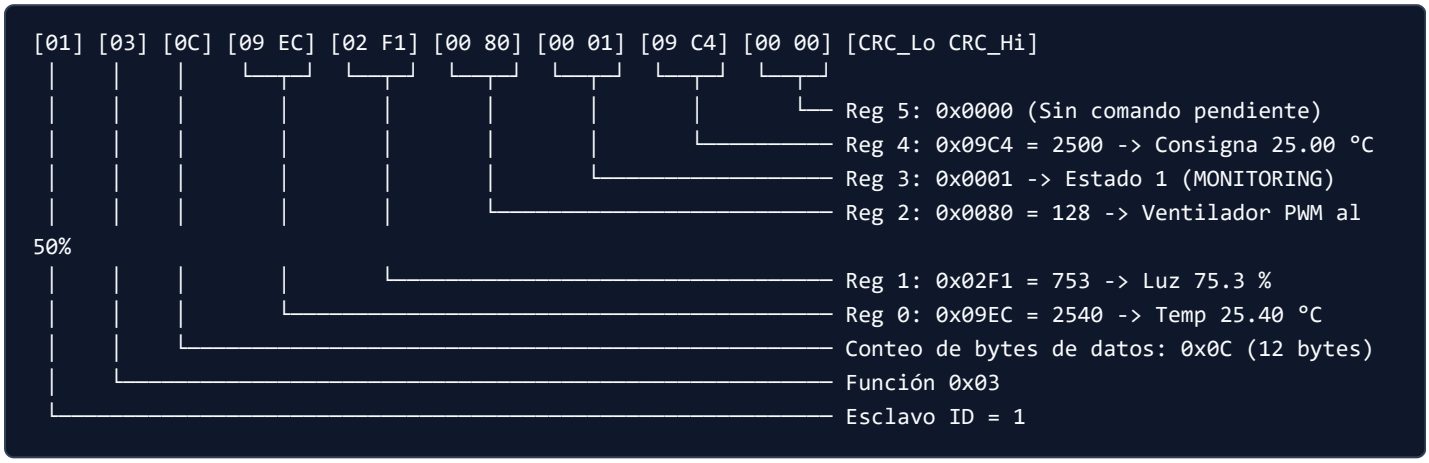
Dirección Hex (Dec)	Nombre Simbólico	Tipo de Dato	Factor de Escala	Acceso	Descripción y Conversión en la Raspberry Pi
`0x0000` (`0`)	`REG_TEMP_C_X100`	`int16_t`	$\times 100$	Solo Lectura (`0x03`)	Temperatura filtrada EMA en centésimas de °C ($T = Valor / 100.0$, ej. `2540` $\rightarrow 25.40$ ^\circC)
`0x0001` (`1`)	`REG_LIGHT_PCT_X10`	`uint16_t`	$\times 10$	Solo Lectura (`0x03`)	Luminosidad relativa en décimas de porcentaje ($L = Valor / 10.0$, ej. `753` $\rightarrow 75.3\%$)
`0x0002` (`2`)	`REG_FAN_PWM`	`uint16_t`	$1:1$ (0\dots 255)	Solo Lectura (`0x03`)	Ciclo de trabajo PWM actual aplicado al ventilador en el pin `D9`
`0x0003` (`3`)	`REG_STATE_FLAGS`	`uint16_t`	Mapa de Bits	Solo Lectura (`0x03`)	Bits `0..3` : Estado FSM (`0..4`) · Bit `4` : Calefactor activo · Bits `8..15` : Código de fallo
`0x0004` (`4`)	`REG_SETPOINT_X100`	`int16_t`	$\times 100$	Lectura / Escritura (`0x03` , `0x06`)	Consigna térmica de control en centésimas de °C (rango válido: `1500` a `4000` , es decir, 15.00 ^\circC a 40.00 ^\circC)
`0x0005` (`5`)	`REG_CMD_CONTROL`	`uint16_t`	Comando	Lectura / Escritura (`0x03` , `0x06`)	`1` : Rearmar alarma de fallo (`FAULT_SENSOR` $\rightarrow$ `IDLE`) · `2` : Forzar parada segura

4. Ejemplo de Trama Hexadecimal Real en el Bus RS-485

4.1 Petición del Maestro (Lectura de los 6 Registros desde `0x0000` , Función `0x03`)



4.2 Respuesta del Esclavo Arduino (`17 Bytes` en total)



Planos y Esquemáticos de Circuito — Módulo 04: Enlaces UART, I²C (`BSS138`), RS-485 (`MAX485`) y Trama Modbus RTU

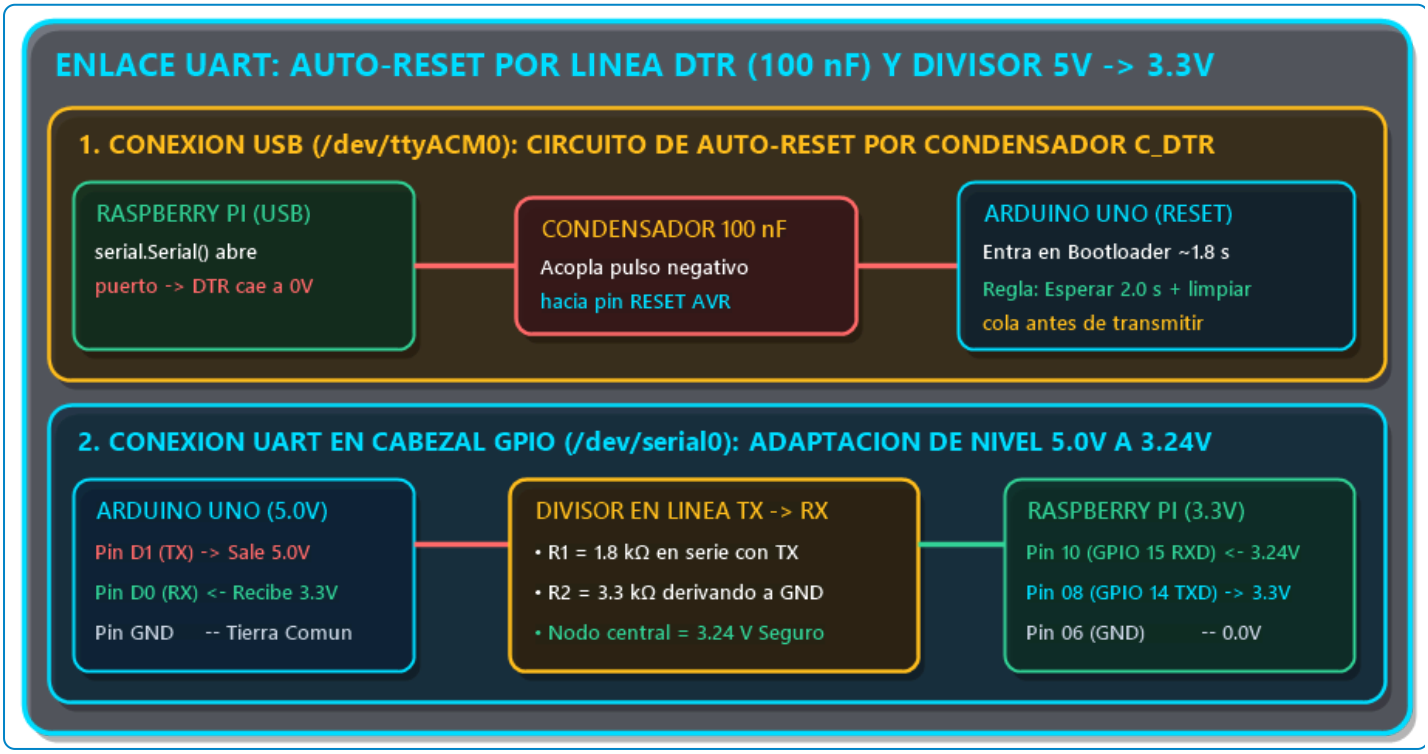
Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 3, 5, 8, 9, 10 y 12

---

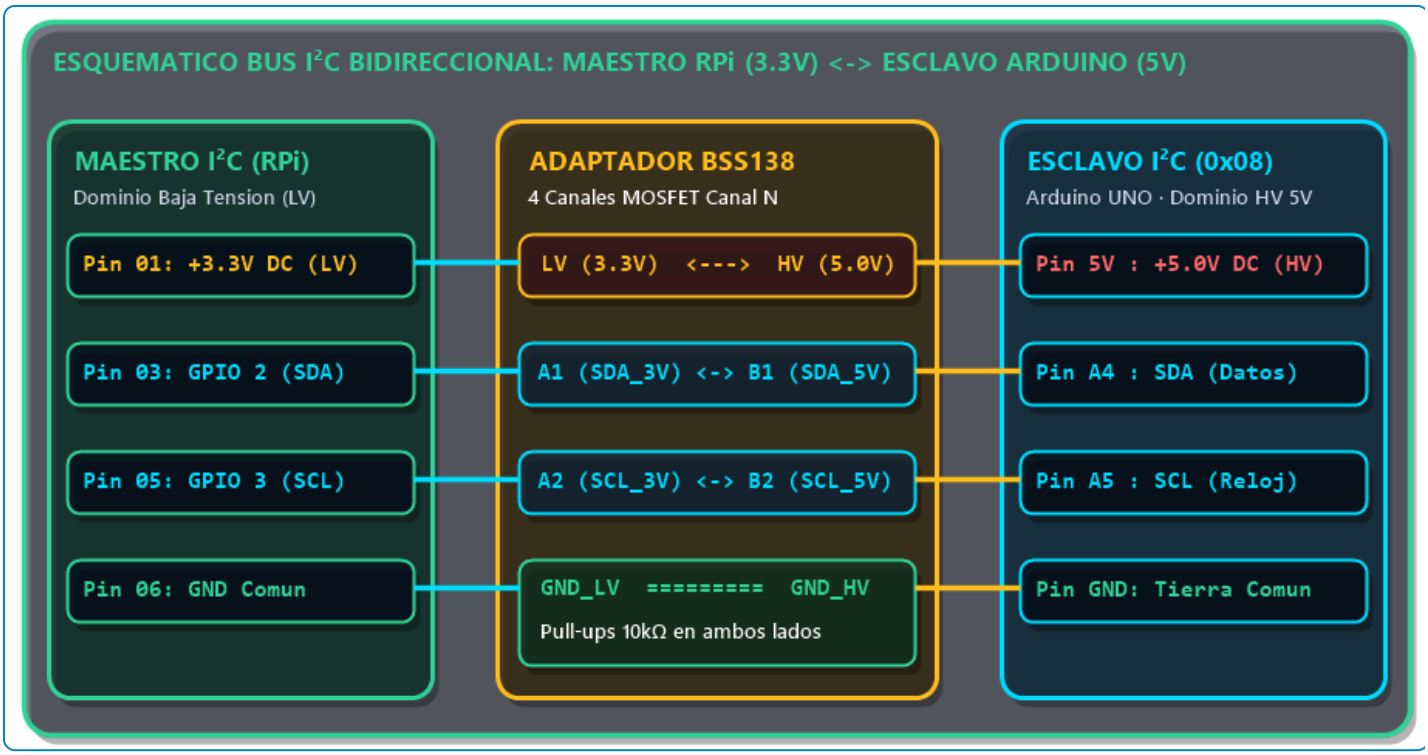
1. Plano 1: Enlace UART, Circuito de Auto-Reset por Línea DTR ($100\ \text{nF}$) y Divisor Resistivo $5\ \text{V} \rightarrow 3.3\ \text{V}$



Origen (Arduino UNO `5.0 V`)	Etapa Intermedia de Acondicionamiento	Destino (Raspberry Pi `3.3 V`)	Nivel Eléctrico Resultante
Puerto USB (`ATmega16U2` / `CH340`, línea `DTR`)	Condensador cerámico en serie $C_{DTR} = 100\ \text{nF}$ hacia pin `RESET`	Puerto USB Host (`/dev/ttyACM0`)	Pulso bajo al abrir el puerto → espera obligatoria de $2.0\ \text{s}$
Pin `D1 (TX)` (`5.0 V` TTL)	Divisor resistivo: $R_1 = 1.8\ \text{k}\Omega$ en serie + $R_2 = 3.3\ \text{k}\Omega$ a `GND`	Pin Físico 10 (`GPIO 15 / RXD0`)	$V_{RX} = 5.0\ \text{V} \cdot (3.3) / (1.8 + 3.3) = 3.24\ \text{V}$ (Seguro)
Pin `D0 (RX)` ($V_{IH,mín} = 3.0\ \text{V}$)	Conexión directa (o vía `BSS138`)	Pin Físico 08 (`GPIO 14 / TXD0`)	$3.30\ \text{V} > 3.00\ \text{V}$ (Reconocido como `'1'` lógico)
Pin `GND`	Conductor directo de retorno	Pin Físico 06 (`GND`)	Referencia equipotencial de $0.0\ \text{V}$

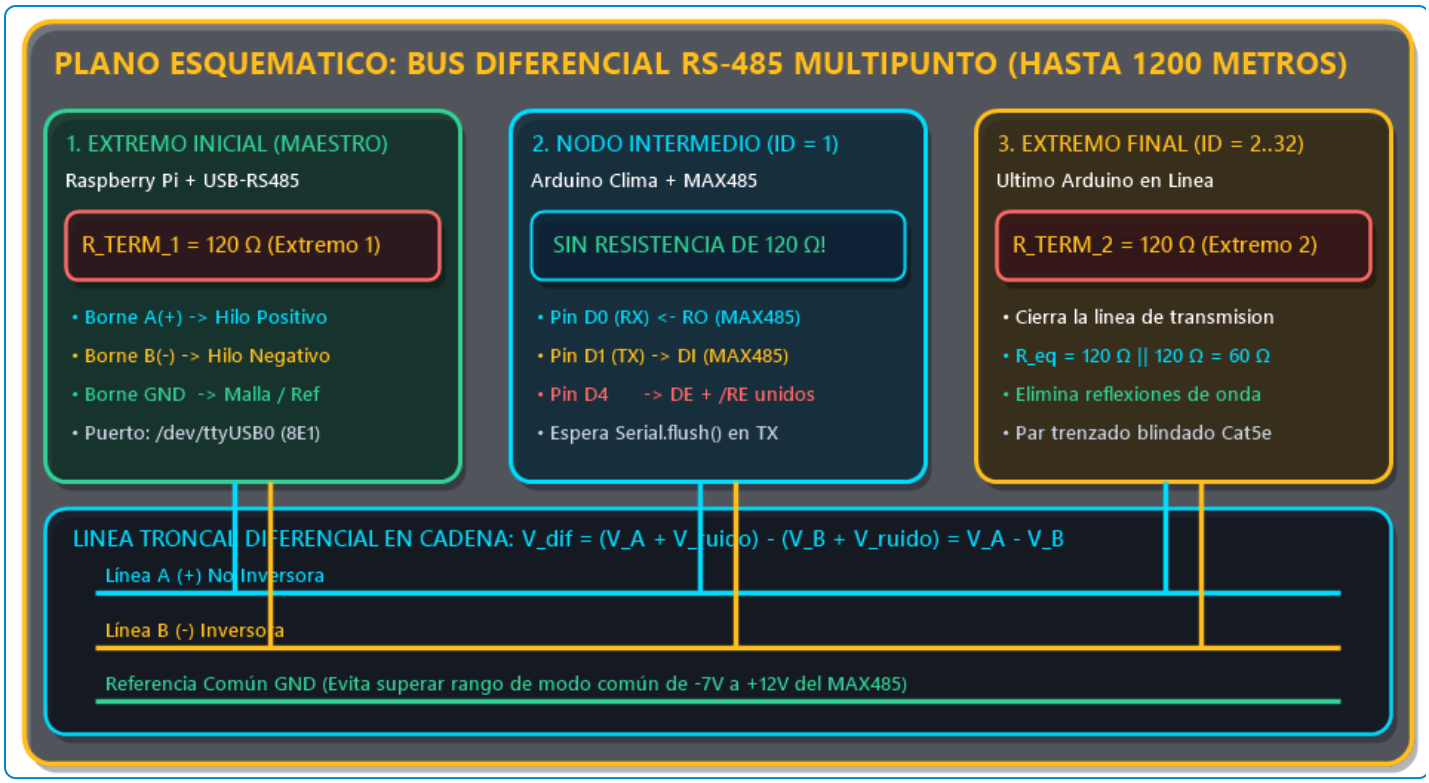
---

2. Plano 2: Esquemático del Bus I²C Bidireccional con Adaptador MOSFET `BSS138`



Señal del Bus	Maestro Raspberry Pi (`3.3 V`)	Conversor `BSS138` (4 Canales)	Esclavo Arduino UNO (`0x08`, `5.0 V`)	Nota de Configuración
Referencia Baja (`LV`)	Pin Físico 01 (`+3.3V DC`)	Pin `LV` (Surtidor + Compuerta MOSFET)	—	Polariza las resistencias *pull-up* de $10\ \text{k}\Omega$ del lado `3.3 V`
Referencia Alta (`HV`)	—	Pin `HV` (Drenador vía *pull-up* $10\ \text{k}\Omega$)	Pin `5V` (`+5.0V DC`)	Polariza el dominio de `5.0 V`
Datos Serie (`SDA`)	Pin Físico 03 (`GPIO 2 / SDA1`)	Canal 1: `A1` (`3.3 V`) $\leftrightarrow$ `B1` (`5.0 V`)	Pin `A4 (SDA)`	Desactivar *pull-up* interno en AVR con `digitalWrite(SDA, LOW)`
Reloj Serie (`SCL`)	Pin Físico 05 (`GPIO 3 / SCL1`)	Canal 2: `A2` (`3.3 V`) $\leftrightarrow$ `B2` (`5.0 V`)	Pin `A5 (SCL)`	Desactivar *pull-up* interno en AVR con `digitalWrite(SCL, LOW)`
Tierra Común (`GND`)	Pin Físico 06 (`GND`)	Pines `GND` (`LV` y `HV` unidos)	Pin `GND`	Indispensable para referencia de tensión compuerta-surtidor (V_{GS})

3. Plano 3: Bus Diferencial Industrial RS-485 Multipunto (`MAX485`) hasta $1200\ \text{m}$



Pin del Transceptor `MAX485`	Nombre Técnico	Conexión en el Nodo Esclavo (Arduino UNO)	Conexión en la Línea Troncal RS-485
Pin 1 (`RO` **)**	*Receiver Output* (Salida del Receptor)	Pin `D0 (RX)` del Arduino UNO	—
Pin 2 (`/RE` **)**	*Receiver Enable* (Activo en Bajo)	Unido a `DE` y conectado al **Pin** `D4` (con *pull-down* de $10\ \text{k}\Omega$ a `GND`)	—
Pin 3 (`DE` **)**	*Driver Enable* (Activo en Alto)	Unido a `/RE` y conectado al **Pin** `D4` (`HIGH` = Transmitir, `LOW` = Escuchar)	—
Pin 4 (`DI` **)**	*Driver Input* (Entrada del Transmisor)	Pin `D1 (TX)` del Arduino UNO	—
Pin 5 (`GND` **)**	Tierra de Referencia	Pin `GND` del Arduino UNO	Hilo de referencia común / malla del cable blindado
Pin 6 (`A` **)**	Línea Diferencial No Inversora `A (+)`	—	Hilo `A (+)` del par trenzado ($R_{term} = 120\ \Omega$ entre `A` y `B` solo en los 2 extremos)
Pin 7 (`B` **)**	Línea Diferencial Inversora `B (-)`	—	Hilo `B (-)` del par trenzado
Pin 8 (`VCC` **)**	Alimentación Positiva (`+5.0 V`)	Pin `5V` del Arduino UNO + condensador cerámico de $100\ \text{nF}$ a `GND`	—

4. Plano 4: Estructura de Trama Binaria y Mapa de Registros `Modbus RTU`

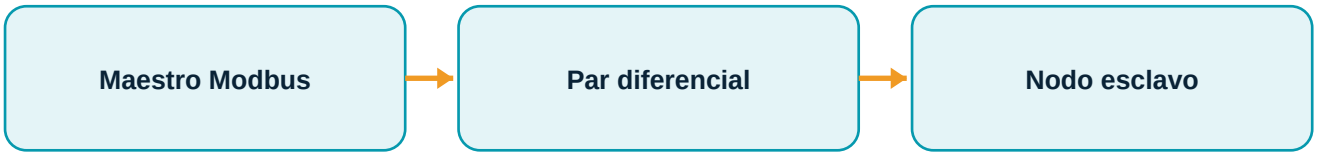


Consulte el documento `MANUAL_PROTOCOLO_MODBUS_RTU_Y_CRC16.pdf` para el detalle bit a bit de los 6 registros de retención (`0x0000` a `0x0005`) y las secuencias de bytes de las funciones `0x03` y `0x06` .

Capítulo 16 · Proyecto: bus industrial RS-485 y Modbus RTU

Conceptos clave

Termina el bus en los extremos físicos si corresponde y evita direcciones duplicadas. Consulta el mapa de registros antes de leer o escribir.



Guía Paso a Paso — Laboratorio 04: Comunicación Directa Arduino ↔ Raspberry Pi (UART, I²C, SPI y Bus Industrial RS-485 / Modbus RTU)

Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 1, 3, 5, 9, 10, 12, 13 y 14

---

1. Objetivo de Ingeniería del Laboratorio

Integrar el nodo controlador de tiempo real basado en **Arduino UNO (`5.0 V`)** con la computadora de borde **Raspberry Pi (`3.3 V`)** analizando cuatro tecnologías de enlace físico y lógico:

- **Enlace Serie Asíncrono (`UART`)**: Comprensión del fenómeno de reinicio automático por línea `DTR` a través del condensador de acoplamiento de $100\ \text{nF}$ y aseguramiento de integridad de trama mediante **verificación de redundancia cíclica (`CRC-16/MODBUS`)**.
- **Buses Síncronos de Corto Alcance (`I²C` y `SPI`)**: Traslación segura de nivel lógico bidireccional con transistores MOSFET de canal N `BSS138` y desactivación por software de las resistencias internas de elevación a `5.0 V` del microcontrolador AVR.
- **Capa Física Diferencial Industrial (`RS-485`)**: Cableado en cadena (*Daisy-Chain*) con par trenzado blindado, cálculo e instalación de resistencias terminadoras de $120\ \Omega$ únicamente en los dos extremos físicos de la línea troncal ($R_{eq} = 60\ \Omega$) y control de dirección semidúplex (`DE` / `/RE`) en el transceptor `MAX485`.
- **Protocolo Industrial `Modbus RTU`**: Exposición del nodo de invernadero como Esclavo `ID = 1` a `19200 bps` en formato `8E1` (8 bits de datos, paridad Par, 1 bit de parada), implementando las funciones `0x03` (Lectura de Registros de Retención) y `0x06` (Escritura de Registro Individual) sobre 6 registros de $16\ \text{bits}$ con escalado en punto fijo.

---

2. Etapa 1: Enlace UART y Prevención del Auto-Reset por Línea DTR

2.1 Diagnóstico del Fenómeno de Auto-Reset (`100 nF`)

Al conectar el Arduino UNO por cable USB a la Raspberry Pi (`/dev/ttyACM0`), el chip conversor USB-Serie controla la línea `DTR` (*Data Terminal Ready*), la cual está acoplada al pin `RESET` del ATmega328P mediante un condensador cerámico en serie de $C_{DTR} = 100\ \text{nF}$:

- Cada vez que un proceso en Linux abre el descriptor del puerto serie (`serial.Serial()`), la señal `DTR` cae de $5.0\ \text{V}$ a $0\ \text{V}$.
- El condensador de $100\ \text{nF}$ transmite ese escalón negativo al pin `RESET`, reiniciando el microcontrolador y activando el cargador de arranque (*Bootloader*) durante aproximadamente $1.5\ \text{s}$ a $1.8\ \text{s}$.
- **Regla de Oro de Software**: Mantener el puerto serie abierto de forma persistente durante toda la vida del servicio, aguardar `time.sleep(2.0)` tras su apertura inicial y vaciar las colas de recepción (`reset_input_buffer()`) antes de emitir la primera consulta.

2.2 Alternativa de Enlace UART Directo en Cabezal GPIO (`/dev/serial0`)

Si se enlazan directamente los pines `TX` / `RX` sin pasar por USB:

- Conectar el pin `D1 (TX, 5.0 V)` del Arduino hacia una resistencia en serie $R_1 = 1.8\ \text{k}\Omega$, y derivar desde el nodo central una resistencia $R_2 = 3.3\ \text{k}\Omega$ hacia `GND`.
- La tensión resultante entregada al **Pin Físico 10 (`GPIO 15 / RXD`)** de la Raspberry Pi es:

> $V_{RX,RPi} = 5.0\ V \cdot (3.3\ k\Omega) / (1.8\ k\Omega + 3.3\ k\Omega) = 3.235\ V < 3.30\ V$

---

3. Etapa 2: Coprocesador en Bus I²C con Conversor Bidireccional BSS138

Cuando el Arduino UNO y la Raspberry Pi residen dentro del mismo gabinete eléctrico a menos de $20\ cm$:

- **Cableado del Adaptador `BSS138`**:
- Lado de Baja Tensión (`LV`): Conectar `LV` al **Pin 01 (`+3.3V`)** de la Raspberry Pi, `A1` a **`GPIO 2 (SDA, Pin 03)`**, `A2` a **`GPIO 3 (SCL, Pin 05)`** y `GND` al **Pin 06 (`GND`)**.
- Lado de Alta Tensión (`HV`): Conectar `HV` al pin **`5V`** del Arduino UNO, `B1` al pin **`A4 (SDA)`**, `B2` al pin **`A5 (SCL)`** y `GND` a tierra común.
- **Protección del Dominio de `3.3 V` en el Firmware del Arduino (`coprocesador_i2c_esclavo.ino`)**:
- Inmediatamente después de invocar `Wire.begin(0x08)`, la biblioteca estándar de Arduino activa por defecto las resistencias internas de elevación a $+5.0\ V$ en los pines `A4` y `A5`.
- **Acción obligatoria**: Ejecutar `digitalWrite(SDA, LOW)` y `digitalWrite(SCL, LOW)` justo después de `Wire.begin(0x08)` para desactivar los *pull-ups* internos a $5\ V$.
- **Verificación en la Raspberry Pi**:
- Habilitar el bus `I²C` con `sudo raspi-config nonint do_i2c 0`.
- Ejecutar `i2cdetect -y 1` y comprobar que aparece la dirección hexadecimal **`08`** en la matriz de escrutinio.

---

4. Etapa 3: Montaje del Bus Diferencial Industrial RS-485 (`MAX485`)

Para distancias de $10\ m$ a $1200\ m$ en entornos con motores, variadores de frecuencia y contactores:

4.1 Conexión entre Arduino UNO y Transceptor `MAX485`

- **Alimentación**: Conectar `VCC` del módulo `MAX485` a `+5.0 V` del Arduino y `GND` a tierra común (con condensador de desacoplo de $100\ nF$ entre `VCC` y `GND`).
- **Señales UART**:
- Pin **`RO`** (*Receiver Output*) del `MAX485` → Pin **`D0 (RX)`** del Arduino UNO.
- Pin **`DI`** (*Driver Input*) del `MAX485` → Pin **`D1 (TX)`** del Arduino UNO.
- **Control de Dirección Semidúplex (`DE` + `/RE`)**:
- Unir físicamente los pines **`DE`** (*Driver Enable*, activo en alto) y **`/RE`** (*Receiver Enable*, activo en bajo) del módulo `MAX485`.
- Conectar ambos pines unidos al **Pin Digital `D4`** del Arduino UNO, añadiendo una resistencia *pull-down* de $10\ k\Omega$ hacia `GND` para garantizar que el nodo permanezca en modo escucha (`LOW`) mientras el microcontrolador arranca.

4.2 Cableado de la Línea Troncal Diferencial (`A+` , `B-` y `GND`)

- Utilizar **cable de par trenzado blindado** de impedancia característica $Z_0 = 120\ \Omega$.
- Unir el borne **`A (+)`** del adaptador USB-RS485 de la Raspberry Pi con el borne **`A (+)`** del módulo `MAX485` del Arduino.
- Unir el borne **`B (-)`** del adaptador USB-RS485 con el borne **`B (-)`** del módulo `MAX485`.
- Unir la referencia común **`GND`** entre todos los nodos para mantener la tensión de modo común dentro del margen operativo del `MAX485` ($-7\ V$ a $+12\ V$).

- **Instalación de Resistencias Terminadoras ($120\ \Omega$)**:
- Instalar una resistencia $R_{term,1} = 120\ \Omega\ (1/4\ W)$ entre `A(+)` y `B(-)` en el **Extremo Inicial** (adaptador de la Raspberry Pi).
- Instalar una segunda resistencia $R_{term,2} = 120\ \Omega\ (1/4\ W)$ entre `A(+)` y `B(-)` en el **Extremo Final** del cable troncal.
- **Verificación con óhmetro (con el sistema apagado)**: Medir la resistencia equivalente entre `A(+)` y `B(-)`; debe leerse $R_{eq} = 120\ \Omega\ ||\ 120\ \Omega = 60\ \Omega\ \pm 5\%$. **Jamás** colocar resistencias de $120\ \Omega$ en nodos intermedios.

---

5. Etapa 4: Puesta en Marcha del Enlace Maestro-Esclavo Modbus RTU

5.1 Carga del Firmware Esclavo en Arduino (`esclavo_modbus_rtu_arduino.ino`)

- Desconectar temporalmente el pin `D0 (RX)` del módulo `MAX485` mientras se sube el programa por USB al Arduino UNO.
- Compilar y cargar `esclavo_modbus_rtu_arduino.ino`. El firmware configura el puerto serie a **`19200 bps`** con formato **`SERIAL_8E1`** (8 bits de datos, paridad Par, 1 bit de parada) y asigna la dirección **`SLAVE_ID = 1`**.
- Reconectar el pin `D0 (RX)` al pin `RO` del módulo `MAX485`.

5.2 Ejecución del Maestro en Raspberry Pi (`maestro_modbus_rtu_raspberry.py`)

- Activar el entorno virtual Python en la Raspberry Pi e instalar las bibliotecas requeridas:

  ```bash
  source ~/iot_edge_node/venv/bin/activate
  pip install minimalmodbus pyserial
  ```

- Ejecutar el programa maestro:

  ```bash
  python3 maestro_modbus_rtu_raspberry.py --port /dev/ttyUSB0 --slave-id 1
  ```

- Verificar en consola la lectura cíclica de los 6 registros de retención (`0x0000` a `0x0005`) y probar la escritura remota de una nueva consigna térmica con la función **`0x06`**:

  ```bash
  python3 maestro_modbus_rtu_raspberry.py --port /dev/ttyUSB0 --setpoint 26.50
  ```

---

6. Lista de Verificación de Diagnóstico de Fallos

Síntoma Observado	Causa Raíz Probable	Solución de Ingeniería
El último byte de respuesta del Arduino llega corrupto (error de `CRC-16`)	El pin `D4` (`DE` / `/RE`) pasa a `LOW` antes de que el registro de desplazamiento UART termine de emitir los bits físicos	Invocar obligatoriamente `Serial.flush()` más una guarda de `delayMicroseconds(1820)` antes de `digitalWrite(4, LOW)`
Error `No communication with the instrument` en `minimalmodbus`	Líneas diferenciales `A(+)` y `B(-)` invertidas o formato de paridad distinto entre Maestro y Esclavo	Verificar polaridad `A(+)` con `A(+)`, `B(-)` con `B(-)` y confirmar `19200 bps` en formato **`8E1`** en ambos extremos
Sobrecalentamiento de los transceptores `MAX485` y caída de tensión diferencial	Múltiples nodos intermedios tienen instalada su resistencia SMD de $120\ \Omega$ de fábrica ($R_{eq} \ll 60\ \Omega$)	Desoldar la resistencia `R7` ($120\ \Omega$) de todos los módulos `MAX485` ubicados en posiciones intermedias del bus
El Arduino se reinicia cada vez que se ejecuta el script en Python por USB	Pulso negativo en la línea `DTR` a través del condensador de $100\ nF$	Mantener la instancia de puerto abierta dentro de un bucle continuo y aguardar $2.0\ s$ tras abrir el descriptor

Listado de Materiales (BOM) — Módulo 04: Comunicación Directa UART, I²C, SPI, RS-485 y Modbus RTU

Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 3, 5, 6, 8, 9, 10 y 14

---

1. Tabla Principal de Componentes de Hardware y Comunicaciones

Ítem	Cant.	Designador	Componente / Descripción Técnica	Especificación Eléctrica / Mecánica	Función en el Módulo 04
01	1	`SBC1`	**Raspberry Pi 4 Model B** (o RPi 5)	SoC ARM64 (`3.3 V` CMOS), puertos USB y cabezal GPIO de 40 pines	Nodo Maestro `Modbus RTU` / Maestro `I²C` y `SPI`
02	1	`U1`	**Arduino UNO R3** (`ATmega328P`)	Lógica TTL de `5.0 V`, reloj de $16\ \text{MHz}$, `UART` hardware (`D0` / `D1`), `I²C` (`A4` / `A5`)	Nodo Esclavo `Modbus RTU` (`ID = 1`) / Coprocesador `I²C` (`0x08`)
03	1	`U2`	**Adaptador Industrial USB a RS-485**	Chip `FT232RL` + `MAX485` (o `CH340E`), protección `TVS` contra transitorios y bornes `A(+)` , `B(-)` , `GND`	Interfaz física `RS-485` en la Raspberry Pi (`/dev/ttyUSB0`)
04	2	`U3, U4`	**Módulo Transceptor RS-485 (`MAX485CSA`)**	Alimentación `5.0 V` , semidúplex hasta $2.5\ \text{Mbps}$, hasta 32 cargas unitarias (`1 UL = 12 kΩ`)	Conversión entre `UART` TTL (`RO` , `DI` , `DE` / `/RE`) y par diferencial `RS-485`
05	1	`U5`	**Conversor Bidireccional de Nivel Lógico `BSS138` (4 Canales)**	4 transistores MOSFET Canal N `BSS138` con resistencias *pull-up* de $10\ \text{k}\Omega$ en `LV` (`3.3 V`) y `HV` (`5.0 V`)	Adaptación bidireccional de colector abierto para buses `I²C` (`SDA` / `SCL`) y `SPI`
06	2	`R_TERM1, R_TERM2`	**Resistencias Terminadoras de Línea RS-485**	$120\ \Omega\ \pm\ 1\%$, película metálica, potencia de $1/4\ \text{W}$ ($0.25\ \text{W}$)	Adaptación de impedancia característica ($Z_0 = 120\ \Omega$) en los 2 extremos físicos del bus
07	1	`R_PD`	**Resistencia Pull-Down de Habilitación TX**	$10\ \text{k}\Omega\ \pm\ 1\%$, $1/4\ \text{W}$	Mantiene los pines `DE` / `/RE` (`Pin D4`) en modo escucha (`0 V`) durante el arranque del AVR
08	2	`R_PU1, R_PU2`	**Resistencias Pull-Up Externas para Bus I²C**	$4.7\ \text{k}\Omega\ \pm\ 1\%$, $1/4\ \text{W}$	Ajuste del tiempo de subida ($\tau = R_p \cdot C_{bus}$) hacia el riel de `3.3 V` si hay múltiples esclavos
09	1	`R1, R2`	**Par de Resistencias para Divisor UART Directo**	$R_1 = 1.8\ \text{k}\Omega$ y $R_2 = 3.3\ \text{k}\Omega$ ($\pm\ 1\%$, $1/4\ \text{W}$)	Atenuación de la línea `TX` de `5.0 V` del Arduino a $3.24\ \text{V}$ hacia `GPIO 15 (RXD)`
10	2	`C_DEC1, C_DEC2`	**Condensadores de Desacoplo Cerámicos**	$100\ \text{nF}$ ($0.1\ \mu\text{F}$), dieléctrico `X7R` , $50\ \text{V}$	Filtrado de picos de corriente en el pin `VCC` de cada transceptor `MAX485`
11	5 m	`CAB1`	**Cable de Par Trenzado Blindado (`STP` / `RS-485`)**	Impedancia característica $Z_0 = 120\ \Omega$, calibre `24 AWG` con malla de drenaje	Línea troncal diferencial para interconexión en cadena (*Daisy-Chain*)

2. Cálculos de Verificación Eléctrica

2.1 Impedancia Equivalente y Corriente en el Bus RS-485

Con únicamente las dos resistencias terminadoras de $120\ \Omega$ instaladas en los extremos inicial y final de la línea troncal:

$$R_{eq} = \frac{R_{term,1} \cdot R_{term,2}}{R_{term,1} + R_{term,2}} = (120\ \Omega \cdot 120\ \Omega) / (120\ \Omega + 120\ \Omega) = 60\ \Omega$$

Para una tensión diferencial típica de salida $V_{OD} = 2.0\ V$ entregada por el transmisor `MAX485`:

- **Corriente en el bucle diferencial**: $I_{bus} = (2.0\ V) / (60\ \Omega) = 33.3\ mA$ (dentro del límite nominal de $60\ mA$ del `MAX485`).
- **Potencia disipada en cada terminador**: $P_{term} = ((2.0\ V)^2) / (120\ \Omega) = 33.3\ mW \ll 250\ mW$ ($1/4\ W$).

2.2 Tiempo de Carácter y Silencio de Trama Modbus RTU a 19200 bps (`8E1`)

Cada carácter en formato `8E1` consta de $11\ bits$ ($1\ Start + 8\ Datos + 1\ Paridad\ Par + 1\ Stop$):

$$T_{char} = (11\ bits) / (19200\ bits/s) = 572.92\ \mu s \quad\Longrightarrow\quad T_{3.5\ char} = 3.5 \times 572.92\ \mu s \approx 2005\ \mu s \approx 2.0\ ms$$

Manual Técnico del Protocolo Modbus RTU y Verificación CRC-16 (Módulo 04)

Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 4, 11, 12, 13 y 14

---

1. Fundamentos Matemáticos del Código de Redundancia Cíclica `CRC-16/MODBUS`

En enlaces serie asíncronos (`UART` y `RS-485`) expuestos a ruido electromagnético industrial, un simple bit de paridad o una suma aritmética de bytes (*Checksum*) no detecta inversiones simétricas de bits ni transposiciones de octetos. El estándar `Modbus RTU` emplea el polinomio generador algebraico de grado 16:

$$G(x) = x^{16} + x^{15} + x^2 + 1$$

1.1 Parámetros Canónicos del Algoritmo `CRC-16/MODBUS`

- **Representación normal del polinomio**: `0x8005` ($1000\,0000\,0000\,0101_2$).
- **Representación reflejada (procesamiento LSB primero)**: `0xA001` ($1010\,0000\,0000\,0001_2$).
- **Valor semilla inicial (`Init`)**: `0xFFFF` .
- **Orden de transmisión en la línea**: Primero se envía el byte menos significativo (`CRC_Lo`) y en último lugar el byte más significativo (`CRC_Hi`).
- **Capacidad de detección**: Detecta el 100% de los errores de 1, 2 y número impar de bits, todas las ráfagas de error de hasta 16 bits de longitud, y el 99.9985% ($1 - 2^{-16}$) de ráfagas mayores.

1.2 Algoritmo Iterativo Paso a Paso (C++ / Python)

Para una trama de N bytes de carga útil $B_0, B_1, \dots, B_{N-1}$:

- Inicializar el acumulador de 16 bits: $CRC \leftarrow 0xFFFF$.
- Para cada byte B_k ($k = 0 \dots N-1$):
- Operar $CRC \leftarrow CRC \oplus B_k$.
- Repetir 8 veces (para cada bit $i = 0 \dots 7$):
- Si el bit menos significativo $(CRC \,\&\, 0x0001) = 1$, entonces: $CRC \leftarrow (CRC >> 1) \oplus 0xA001$.
- En caso contrario: $CRC \leftarrow (CRC >> 1)$.
- Al finalizar, adjuntar $CRC_{Lo} = CRC \,\&\, 0xFF$ seguido de $CRC_{Hi} = (CRC >> 8) \,\&\, 0xFF$.

---

2. Estructura de Trama Binaria y Temporización `Modbus RTU`

A diferencia de formatos textuales como JSON o CSV, `Modbus RTU` transmite octetos binarios puros delimitados exclusivamente por **intervalos de silencio en la línea**:

Campo de la Trama	Tamaño	Codificación	Descripción Técnica
Silencio Inicial	≥ 3.5 char	Línea en reposo (`Idle`)	A `19200 bps` (`8E1`), equivale a ≥ 2.0 ms sin actividad
Dirección de Esclavo (`ID`)	1 Byte	`0x01` a `0xF7` (1\dots 247)	Identifica al nodo destinatario (`0x00` reservado para difusión *Broadcast*)
Código de Función (`FC`)	1 Byte	`0x03` o `0x06`	`0x03` : Leer Registros de Retención · `0x06` : Escribir Registro Individual
Campo de Datos (`Payload`)	N Bytes	**Big-Endian** (`MSB` primero)	Direcciones de registro de 16 bits, cantidad de palabras y valores escalados
Verificación `CRC-16`	2 Bytes	**Little-Endian** (`LSB` primero)	Primero se transmite `CRC_Lo` y finalmente `CRC_Hi`
Silencio Final	≥ 3.5 char	Línea en reposo (`Idle`)	Cierra la trama actual y habilita el bus para la siguiente transacción

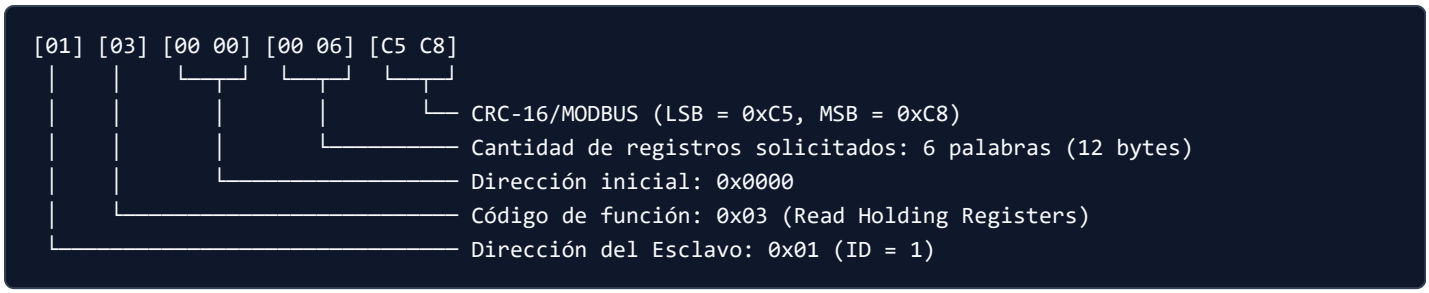
3. Mapa Oficial de Registros de Retención del Nodo de Invernadero (`ID = 1`)

Para evitar el envío de números en coma flotante (`float` IEEE-754 de 32 bits) y problemas de orden de palabras, todas las magnitudes continuas se transmiten como **enteros de 16 bits con escalado en punto fijo**:

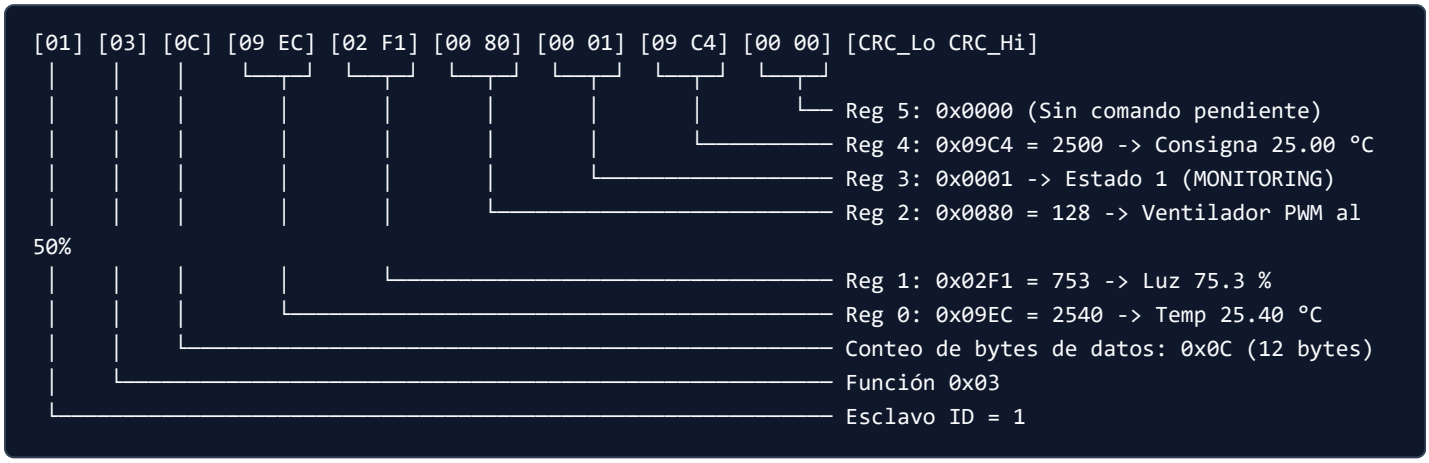
Dirección Hex (Dec)	Nombre Simbólico	Tipo de Dato	Factor de Escala	Acceso	Descripción y Conversión en la Raspberry Pi
`0x0000` (`0`)	`REG_TEMP_C_X100`	`int16_t`	$\times 100$	Solo Lectura (`0x03`)	Temperatura filtrada EMA en centésimas de °C ($T = Valor / 100.0$, ej. `2540` $\rightarrow 25.40$ ^\circC)
`0x0001` (`1`)	`REG_LIGHT_PCT_X10`	`uint16_t`	$\times 10$	Solo Lectura (`0x03`)	Luminosidad relativa en décimas de porcentaje ($L = Valor / 10.0$, ej. `753` $\rightarrow 75.3\%$)
`0x0002` (`2`)	`REG_FAN_PWM`	`uint16_t`	$1:1$ (0\dots 255)	Solo Lectura (`0x03`)	Ciclo de trabajo PWM actual aplicado al ventilador en el pin `D9`
`0x0003` (`3`)	`REG_STATE_FLAGS`	`uint16_t`	Mapa de Bits	Solo Lectura (`0x03`)	Bits `0..3` : Estado FSM (`0..4`) · Bit `4` : Calefactor activo · Bits `8..15` : Código de fallo
`0x0004` (`4`)	`REG_SETPOINT_X100`	`int16_t`	$\times 100$	Lectura / Escritura (`0x03` , `0x06`)	Consigna térmica de control en centésimas de °C (rango válido: `1500` a `4000` , es decir, 15.00 ^\circC a 40.00 ^\circC)
`0x0005` (`5`)	`REG_CMD_CONTROL`	`uint16_t`	Comando	Lectura / Escritura (`0x03` , `0x06`)	`1` : Rearmar alarma de fallo (`FAULT_SENSOR` $\rightarrow$ `IDLE`) · `2` : Forzar parada segura

4. Ejemplo de Trama Hexadecimal Real en el Bus RS-485

4.1 Petición del Maestro (Lectura de los 6 Registros desde `0x0000` , Función `0x03`)



4.2 Respuesta del Esclavo Arduino (`17 Bytes` en total)



Planos y Esquemáticos de Circuito — Módulo 04: Enlaces UART, I²C (`BSS138`), RS-485 (`MAX485`) y Trama Modbus RTU

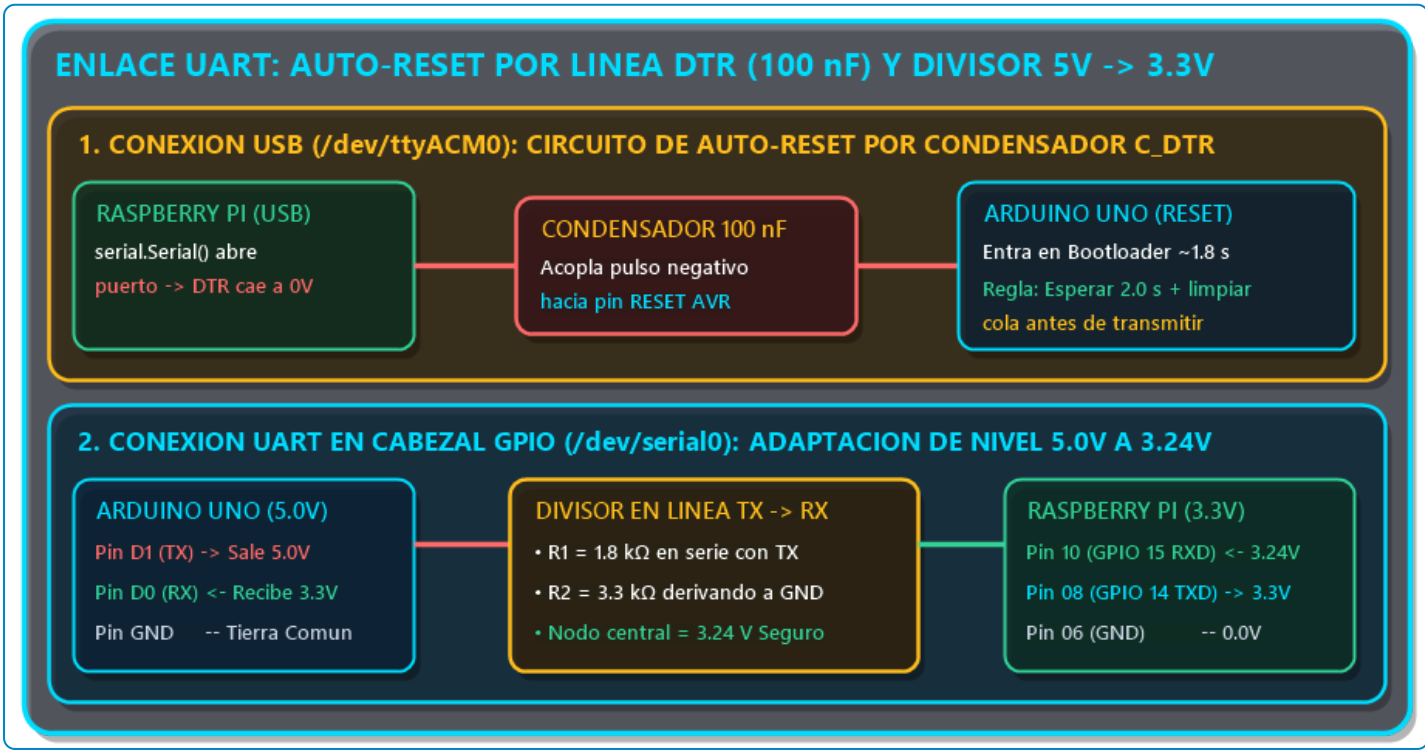
Curso: De Sistemas Embebidos a IoT — Integración Práctica con Arduino y Raspberry Pi

Módulo 04: Comunicación Directa Arduino ↔ Raspberry Pi

Referencia en Video 04: Diapositivas 3, 5, 8, 9, 10 y 12

---

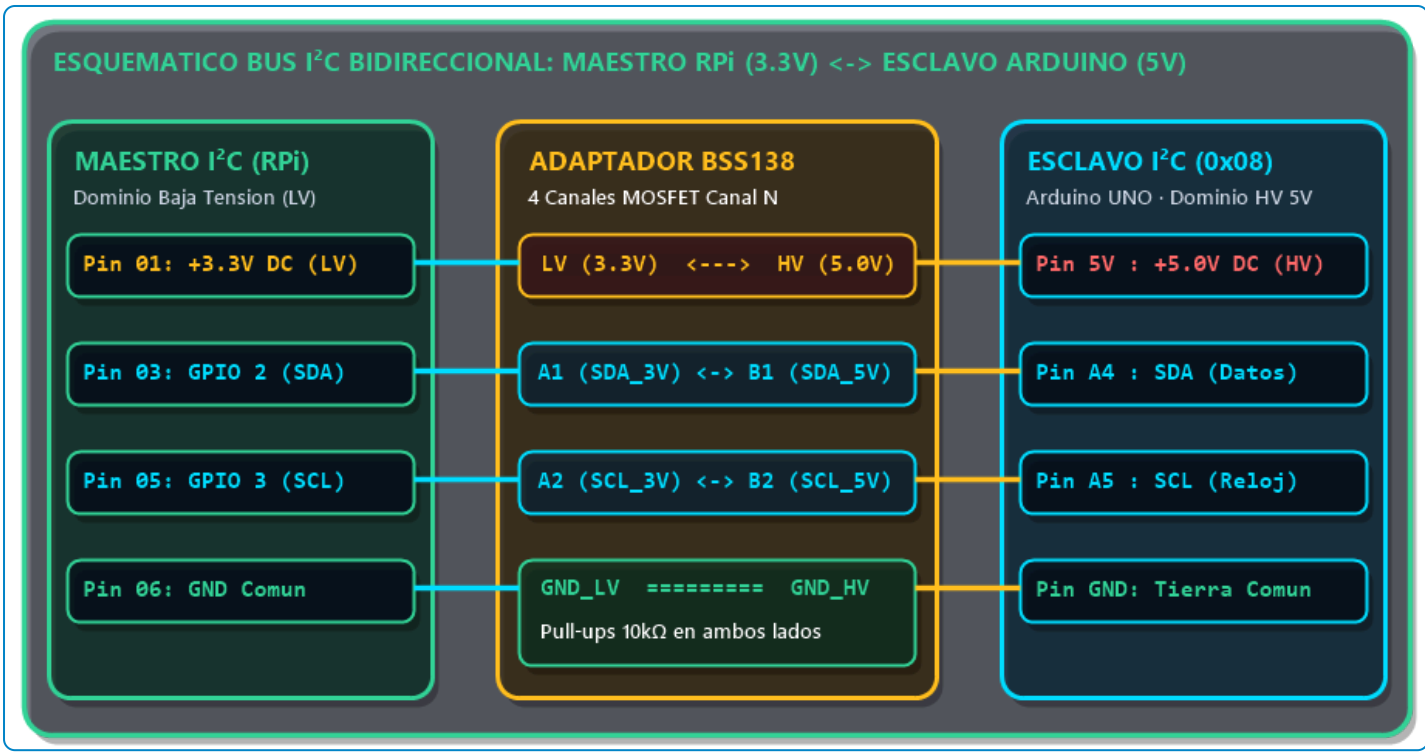
1. Plano 1: Enlace UART, Circuito de Auto-Reset por Línea DTR ($100\ \text{nF}$) y Divisor Resistivo $5\ \text{V} \rightarrow 3.3\ \text{V}$



Origen (Arduino UNO `5.0 V`)	Etapa Intermedia de Acondicionamiento	Destino (Raspberry Pi `3.3 V`)	Nivel Eléctrico Resultante
Puerto USB (`ATmega16U2` / `CH340`, línea `DTR`)	Condensador cerámico en serie $C_{DTR} = 100\ \text{nF}$ hacia pin `RESET`	Puerto USB Host (`/dev/ttyACM0`)	Pulso bajo al abrir el puerto → espera obligatoria de $2.0\ \text{s}$
Pin `D1 (TX)` (`5.0 V` TTL)	Divisor resistivo: $R_1 = 1.8\ \text{k}\Omega$ en serie + $R_2 = 3.3\ \text{k}\Omega$ a `GND`	Pin Físico 10 (`GPIO 15 / RXD0`)	$V_{RX} = 5.0\ \text{V} \cdot (3.3) / (1.8 + 3.3) = 3.24\ \text{V}$ (Seguro)
Pin `D0 (RX)` ($V_{IH,mín} = 3.0\ \text{V}$)	Conexión directa (o vía `BSS138`)	Pin Físico 08 (`GPIO 14 / TXD0`)	$3.30\ \text{V} > 3.00\ \text{V}$ (Reconocido como `'1'` lógico)
Pin `GND`	Conductor directo de retorno	Pin Físico 06 (`GND`)	Referencia equipotencial de $0.0\ \text{V}$

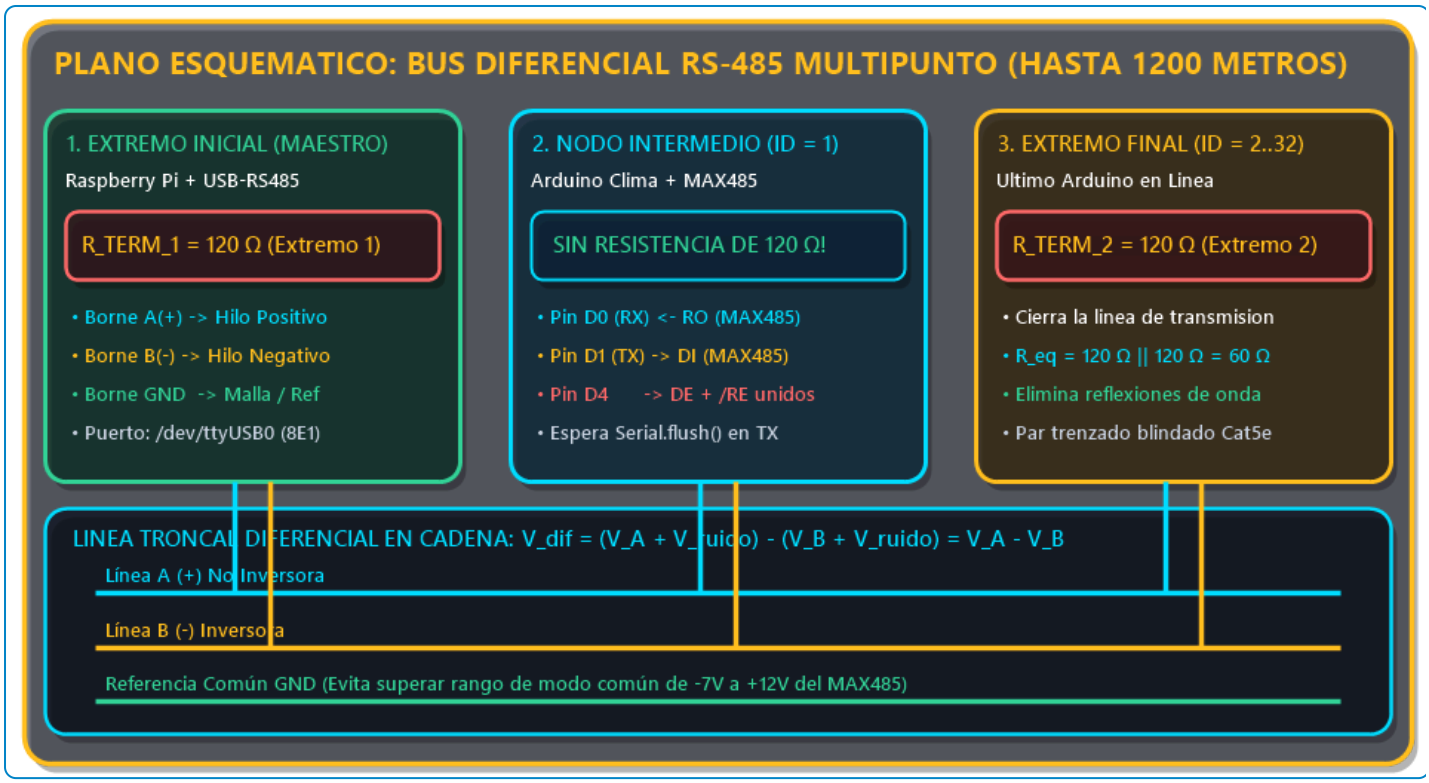
---

2. Plano 2: Esquemático del Bus I²C Bidireccional con Adaptador MOSFET `BSS138`



Señal del Bus	Maestro Raspberry Pi (`3.3 V`)	Conversor `BSS138` (4 Canales)	Esclavo Arduino UNO (`0x08`, `5.0 V`)	Nota de Configuración
Referencia Baja (`LV`)	Pin Físico 01 (`+3.3V DC`)	Pin `LV` (Surtidor + Compuerta MOSFET)	—	Polariza las resistencias *pull-up* de $10\ \text{k}\Omega$ del lado `3.3 V`
Referencia Alta (`HV`)	—	Pin `HV` (Drenador vía *pull-up* $10\ \text{k}\Omega$)	Pin `5V` (`+5.0V DC`)	Polariza el dominio de `5.0 V`
Datos Serie (`SDA`)	Pin Físico 03 (`GPIO 2 / SDA1`)	Canal 1: `A1` (`3.3 V`) $\leftrightarrow$ `B1` (`5.0 V`)	Pin `A4 (SDA)`	Desactivar *pull-up* interno en AVR con `digitalWrite(SDA, LOW)`
Reloj Serie (`SCL`)	Pin Físico 05 (`GPIO 3 / SCL1`)	Canal 2: `A2` (`3.3 V`) $\leftrightarrow$ `B2` (`5.0 V`)	Pin `A5 (SCL)`	Desactivar *pull-up* interno en AVR con `digitalWrite(SCL, LOW)`
Tierra Común (`GND`)	Pin Físico 06 (`GND`)	Pines `GND` (`LV` y `HV` unidos)	Pin `GND`	Indispensable para referencia de tensión compuerta-surtidor (V_{GS})

3. Plano 3: Bus Diferencial Industrial RS-485 Multipunto (`MAX485`) hasta $1200\ \text{m}$



Pin del Transceptor `MAX485`	Nombre Técnico	Conexión en el Nodo Esclavo (Arduino UNO)	Conexión en la Línea Troncal RS-485
Pin 1 (`RO` **)**	*Receiver Output* (Salida del Receptor)	Pin `D0 (RX)` del Arduino UNO	—
Pin 2 (`/RE` **)**	*Receiver Enable* (Activo en Bajo)	Unido a `DE` y conectado al **Pin** `D4` (con *pull-down* de $10\ \text{k}\Omega$ a `GND`)	—
Pin 3 (`DE` **)**	*Driver Enable* (Activo en Alto)	Unido a `/RE` y conectado al **Pin** `D4` (`HIGH` = Transmitir, `LOW` = Escuchar)	—
Pin 4 (`DI` **)**	*Driver Input* (Entrada del Transmisor)	Pin `D1 (TX)` del Arduino UNO	—
Pin 5 (`GND` **)**	Tierra de Referencia	Pin `GND` del Arduino UNO	Hilo de referencia común / malla del cable blindado
Pin 6 (`A` **)**	Línea Diferencial No Inversora `A (+)`	—	Hilo `A (+)` del par trenzado ($R_{term} = 120\ \Omega$ entre `A` y `B` solo en los 2 extremos)
Pin 7 (`B` **)**	Línea Diferencial Inversora `B (-)`	—	Hilo `B (-)` del par trenzado
Pin 8 (`VCC` **)**	Alimentación Positiva (`+5.0 V`)	Pin `5V` del Arduino UNO + condensador cerámico de $100\ \text{nF}$ a `GND`	—

4. Plano 4: Estructura de Trama Binaria y Mapa de Registros `Modbus RTU`



Consulte el documento `MANUAL_PROTOCOLO_MODBUS_RTU_Y_CRC16.pdf` para el detalle bit a bit de los 6 registros de retención (`0x0000` a `0x0005`) y las secuencias de bytes de las funciones `0x03` y `0x06` .

Capítulo 17 · MQTT: jerarquía de temas, QoS, LWT y Mosquitto

Conceptos clave

Define temas por activo y variable. Usa LWT para anunciar desconexiones y adapta el QoS al efecto que tendría recibir duplicados.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 05

> **Despliegue de Broker Eclipse Mosquitto en Raspberry Pi, Endurecimiento de Seguridad y Puente Bidireccional Modbus RTU ↔ MQTT**

1. Objetivo General del Laboratorio

Implementar una arquitectura completa de mensajería industrial **Publicador/Suscriptor (MQTT v3.1.1 / v5.0)** sobre una **Raspberry Pi 4 Model B / 5**, integrando el nodo de campo **Arduino UNO R3 (Esclavo Modbus RTU ID=1)** construido en el Módulo 04 hacia un **Broker Eclipse Mosquitto 2.x** asegurado con autenticación por contraseña cifrada (`SHA-512`), listas de control de acceso por tópico (**ACL**) y un servicio demonio en Python 3 (`puente_modbus_a_mqtt_edge.py`) con exclusión mutua (`threading.Lock`), mensajes retenidos (**Retained**) y testamento de última voluntad (**LWT**).

2. Arquitectura de Red y Mapa de Tópicos del Laboratorio

Parámetro de Red / Serial	Valor de Producción en Laboratorio
Interfaz Serial RS-485 en Raspberry Pi	`/dev/ttyUSB0` (9600 bps, formato `8N1` , Half-Duplex)
Dirección Esclavo Modbus RTU (Arduino UNO)	`ID = 1` (Registros Holding `40001` a `40005`)
Dirección IP del Broker Mosquitto (RPi)	`127.0.0.1` (local) / `192.168.1.100` (LAN Planta)
Puertos TCP del Broker	`1883` (LAN autenticada) / `8883` (TLS 1.3 con X.509)
Raíz Jerárquica de Tópicos (ISA-95)	`invernadero/zona_norte/nodo_01/`
Tópico de Estado y Testamento (LWT)	`invernadero/zona_norte/nodo_01/estado/conexion` (`QoS 1, Retain=True`)
Tópico de Telemetría Agregada JSON	`invernadero/zona_norte/nodo_01/estado/resumen` (`QoS 1, Retain=False`)
Tópico de Mando Descendente (Actuadores)	`invernadero/zona_norte/nodo_01/actuadores/comando` (`QoS 1, Retain=False`)
Tópico de Confirmación de Escritura `0x06`	`invernadero/zona_norte/nodo_01/actuadores/confirmacion` (`QoS 1, Retain=False`)

3. Fase 1: Instalación del Broker Eclipse Mosquitto 2.x en Raspberry Pi OS

Conéctese por SSH a la Raspberry Pi y ejecute la actualización de repositorios e instalación tanto del servidor demonizado (`mosquitto`) como de las herramientas de diagnóstico por línea de comandos (`mosquitto-clients`):

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y mosquitto mosquitto-clients python3-pip python3-venv
sudo systemctl enable mosquitto
```

Verifique que el binario del servidor corresponde a la serie **Eclipse Mosquitto 2.x**, la cual por diseño de ciberseguridad **restringe el tráfico no autenticado a la interfaz `loopback` (`127.0.0.1`)** hasta que se configure explícitamente el archivo de seguridad:

```
mosquitto -h | head -n 3
sudo systemctl status mosquitto
```

---

4. Fase 2: Creación de Cuentas Cifradas (`mosquitto_passwd`) y Listas ACL

Nunca opere un broker industrial con `allow_anonymous true`. Crearemos tres identidades con privilegios segregados según el principio de mínimo privilegio:

- `puente_edge`: Cuenta exclusiva del demonio Python en la Raspberry Pi.
- `scada_central`: Cuenta del sistema supervisor / Node-RED con permiso de mando.
- `monitor_operador`: Cuenta de solo lectura para paneles de visualización.

```
# 1. Crear archivo de contraseñas cifradas con hash SHA-512 + Salt (-c crea el archivo)
sudo mosquitto_passwd -c /etc/mosquitto/passwd puente_edge
sudo mosquitto_passwd /etc/mosquitto/passwd scada_central
sudo mosquitto_passwd /etc/mosquitto/passwd monitor_operador

# 2. Copiar los archivos de configuración industrial y ACL del material complementario
sudo cp mosquitto_industrial.conf /etc/mosquitto/conf.d/mosquitto_industrial.conf
sudo cp acl_invernadero.conf /etc/mosquitto/acl_invernadero.conf

# 3. Asegurar permisos estrictos de propietario para que Mosquitto pueda leerlos
sudo chown mosquitto:mosquitto /etc/mosquitto/passwd /etc/mosquitto/acl_invernadero.conf
sudo chmod 0600 /etc/mosquitto/passwd /etc/mosquitto/acl_invernadero.conf

# 4. Reiniciar el broker y verificar que escucha en 0.0.0.0:1883
sudo systemctl restart mosquitto
sudo ss -tulpn | grep 1883
```

---

5. Fase 3: Verificación del Árbol de Tópicos, QoS y Retención por CLI

Abra **dos terminales SSH** en la Raspberry Pi para comprobar el enrutamiento y los permisos ACL antes de conectar el bus físico:

Terminal 1 (Suscriptor Supervisor con comodín multinivel `#`):

```
mosquitto_sub -h localhost -p 1883 \
  -u "scada_central" -P "TuClaveSeguraSCADA" \
  -t "invernadero/zona_norte/nodo_01/#" -q 1 -v
```

Terminal 2 (Prueba de publicación retenida y prueba de bloqueo ACL):

```
# Publicar configuración retenida (-r) con QoS 1 (-q 1)
mosquitto_pub -h localhost -p 1883 \
  -u "puente_edge" -P "ClaveSeguraIndustrial2026" \
  -t "invernadero/zona_norte/nodo_01/configuracion/limites" \
  -m '{"consigna_temp_c": 27.5, "histeresis_c": 1.0}' -q 1 -r

# Verificar que monitor_operador NO puede publicar comandos (ACL deniega escritura)
mosquitto_pub -h localhost -p 1883 \
  -u "monitor_operador" -P "ClaveOperador" \
  -t "invernadero/zona_norte/nodo_01/actuadores/comando" \
  -m '{"ventilador": 1}' -q 1
```

---

6. Fase 4: Despliegue del Puente Bidireccional Python (`Modbus RTU ⇔ MQTT`)

- Agregue el usuario `pi` al grupo `dialout` para acceso directo al conversor USB-RS485 (`/dev/ttyUSB0`):

  ```bash
  sudo usermod -a -G dialout pi
  ```

- Instale las bibliotecas de comunicación industrial en un entorno virtual o del sistema:

  ```bash
  pip3 install minimalmodbus pyserial paho-mqtt
  ```

- Copie `puente_modbus_a_mqtt_edge.py` a `/opt/iot_invernadero/` y ejecútelo inicialmente en primer plano para observar el ciclo de sondeo cada $2.0\ s$ (`Función 0x03`) y el mensaje de nacimiento (**Birth Message**):

  ```bash
  sudo mkdir -p /opt/iot_invernadero
  sudo cp puente_modbus_a_mqtt_edge.py /opt/iot_invernadero/
  sudo chown -R pi:pi /opt/iot_invernadero
  python3 /opt/iot_invernadero/puente_modbus_a_mqtt_edge.py
  ```

---

7. Fase 5: Prueba de Mando Bidireccional (`MQTT → Mutex → Modbus 0x06 → Relé`)

Con el puente en ejecución y el suscriptor abierto en la Terminal 1, envíe desde otra consola una orden JSON para encender el extractor forzado y actualizar la consigna térmica a $26.0\ ^\circ C$:

```
mosquitto_pub -h localhost -p 1883 \
  -u "scada_central" -P "TuClaveSeguraSCADA" \
  -t "invernadero/zona_norte/nodo_01/actuadores/comando" \
  -m '{"ventilador": 1, "bomba_riego": 0, "consigna_c": 26.0}' -q 1
```

Comportamiento esperado:

- El callback asíncrono `al_recibir_mensaje` adquiere `candado_serial` (`threading.Lock()`), esperando a que finalice cualquier lectura `0x03` en curso (~$32\ ms$).
- Envía la trama **Modbus** `0x06` **(** `Write Single Register` **)** al registro `40003` (valor `1`) y al registro `40005` (valor `260` , coma fija $\times\ 10$).
- El Arduino UNO R3 conmuta físicamente el **Relé K1** en la protoboard, encendiendo el **motor DC del ventilador extractor**, y devuelve el eco Modbus `0x06` .
- El puente libera `candado_serial` y publica en `.../actuadores/confirmacion` el objeto JSON con `"resultado": "OK_MODBUS_0x06"` .

---

8. Fase 6: Demonización 24/7 con `systemd` y Prueba de Testamento (LWT)

Instale la unidad `modbus-mqtt-bridge.service` para arranque automático y recuperación ante fallos:

```
sudo cp modbus-mqtt-bridge.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable modbus-mqtt-bridge.service
sudo systemctl start modbus-mqtt-bridge.service
sudo journalctl -u modbus-mqtt-bridge.service -f
```

Para validar el **Testamento de Última Voluntad (LWT)**, simule una caída abrupta del proceso enviando la señal `SIGKILL` (`kill -9`), la cual impide enviar el paquete `DISCONNECT` :

```
sudo kill -9 $(pgrep -f puente_modbus_a_mqtt_edge.py)
```

Tras $T_{LWT} = 1.5 \times T_{keepalive}$ (o inmediatamente al cerrarse el descriptor TCP del kernel), el Broker Mosquitto publicará automáticamente `"DESCONECTADO_INESPERADO"` con bandera retenida en `invernadero/zona_norte/nodo_01/estado/conexion`, y `systemd` relanzará el puente a los $5\ s$.

LISTADO DE MATERIALES (BOM) — MÓDULO 05

Hardware de Campo, Pasarela Edge Raspberry Pi, Red RS-485 y Software Broker MQTT

1. Unidad Central de Borde (Gateway IIoT & Broker MQTT)

Ítem	Cant.	Componente / Especificación Técnica	Función en la Arquitectura del Módulo 05
01	1	**Raspberry Pi 4 Model B (4 GB / 8 GB RAM)** o **Raspberry Pi 5**	Ejecuta en simultáneo el Broker **Eclipse Mosquitto 2.x** y el demonio **Puente Python Modbus ↔ MQTT**.
02	1	**Fuente Oficial USB-C 5.1 V / 3.0 A (15.3 W)** o Riel DIN 5 V / 4 A	Alimentación regulada libre de caídas de tensión durante ráfagas de red y escritura en disco.
03	1	**Tarjeta microSD Industrial A2 / U3 (32 GB)** o SSD NVMe por USB 3.0	Almacenamiento de Raspberry Pi OS (64-bit), base de persistencia `mosquitto.db` y registros `systemd`.
04	1	**Carcasa de Aluminio Pasiva o Riel DIN** + Cable Ethernet Cat 6 RJ45	Disipación térmica continua 24/7 y enlace cableado determinista hacia el switch de planta.

2. Interfaz de Acoplamiento Físico RS-485 (Borde ↔ Campo)

Ítem	Cant.	Componente / Especificación Técnica	Función en la Arquitectura del Módulo 05
05	1	**Conversor Industrial USB a RS-485** (Chipset FT232RL o CH340G con TVS)	Expone el puerto `/dev/ttyUSB0` en Linux con bornera de tornillo (`A+` , `B-` , `GND`).
06	1	**Módulo Transceptor MAX485CPA** (DIP-8 en protoboard o PCB con bornera)	Convierte niveles UART TTL ($0–5\ V$) del Arduino UNO a señal diferencial RS-485 ($\pm 1.5\ V$ a $\pm 5\ V$).
07	2	**Resistencias de Terminación** $R_T = 120\ \Omega$ **(**$1/4\ W$**, 1%)**	Adaptación de impedancia característica ($Z_0 = 120\ \Omega$) en ambos extremos físicos del bus.
08	2	**Resistencias de Polarización Fail-Safe (**$680\ \Omega$**,** $1/4\ W$**)**	Mantienen $V_{AB} > +200\ mV$ cuando todos los nodos están en alta impedancia (`DE=0`).
09	5 m	**Cable de Par Trenzado Apantallado (STP Belden 9841 / Cat 5e)**	Transporte diferencial de las líneas `A+` , `B-` y drenaje de referencia `GND` .

3. Nodo Esclavo de Campo en Protoboard (Arduino UNO ID=1 + Sensores y Actuadores)

Ítem	Cant.	Componente / Especificación Técnica	Pin Arduino / Registro Modbus / Tópico MQTT
10	1	**Arduino UNO R3 (ATmega328P, 16 MHz)** con cable USB-B	Esclavo Modbus RTU (`ID=1`): pines `D0 (RX)` , `D1 (TX)` y `D2 (DE/RE)` hacia el MAX485.
11	2	**Protoboards (Solderless Breadboards)** de 830 y 400 puntos	Montaje físico de relés, transceptor, divisores resistivos, pulsadores y diodos LED testigo.
12	2	**Relés de 5 V DC (`K52B-M-DC5` / Módulo 2 Canales Optoacoplado)** + `2N2222` + `1N4007`	Conmutación de potencia controlada por `D8` (**Reg 40003**, Extractor) y `D4` (**Reg 40004**, Bomba).
13	2	**Motores DC de 12 V con Engranaje Reductor / Ventilador Brushless**	Actuadores electromecánicos de extracción forzada y bombeo gobernados desde MQTT (`0x06`).
14	1	**Sensor de Temperatura NTC** $10\ k\Omega$ **/ Sonda Impermeable DS18B20**	Entrada `A0` : temperatura en décimas de grado (**Reg 40001** → `.../sensores/temperatura`).
15	1	**Fotorresistor LDR GL5528** + Resistencia $10\ k\Omega$ (Divisor)	Entrada `A1` : porcentaje de radiación lumínica (**Reg 40002** → `.../sensores/iluminacion`).
16	1	**Sensor Ultrasónico HC-SR04** (Nivel de estanque de riego)	Pines `D3 (TRIG)` y `A2 (ECHO)` : nivel de agua en cm (**Reg 40005** → `.../sensores/nivel_tanque`).
17	1	**Micro Servomotor SG90 (PWM** $50\ Hz$**)**	Pin `D5` : apertura proporcional de escotilla cenital (**Reg 40006**).
18	4	**Diodos LED 5 mm** (Azul, Verde, Ámbar, Rojo) + 4 Resistencias $220\ \Omega$	Pines `D9..D12` : indicadores visuales de tráfico MQTT, estado Online, trama Modbus y Alarma.
19	2	**Pulsadores Táctiles** $6\times 6\ mm$ + 2 Resistencias Pull-Down $10\ k\Omega$	Pines `D6` y `D7` : parada local de emergencia y rearme manual en protoboard.
20	1	**Fuente de Banco / Auxiliar DC (**$12\ V$ **/** $5\ V$**)** con masa común (`GND`)	Alimentación dedicada para bobinas de relés, servomotor y motores DC sin inducir caída en el MCU.

4. Pila de Software e Instrumentación Lógica

Paquete / Herramienta	Versión Recomendada	Propósito en el Laboratorio
Eclipse Mosquitto	`>= 2.0.18`	Broker MQTT ligero en C con soporte TLS 1.3, autenticación `SHA-512` y listas ACL.
mosquitto-clients	`>= 2.0.18`	Herramientas CLI `mosquitto_pub` , `mosquitto_sub` y `mosquitto_passwd` .
Python 3 + MinimalModbus	`Python 3.11+` / `2.1.1`	Maestro Modbus RTU sobre `/dev/ttyUSB0` con cálculo y verificación automática de CRC-16.
Eclipse Paho-MQTT Python	`>= 1.6.1 / 2.0`	Cliente asíncrono MQTT con soporte nativo de `will_set` (LWT), reconexión y niveles QoS 0/1/2.

MANUAL DE ARQUITECTURA DE TÓPICOS, CALIDAD DE SERVICIO (QoS) Y SEGURIDAD MQTT

Diseño de Espacios de Nombres ISA-95, Dimensionamiento de Tráfico, Retención, LWT y Endurecimiento de Mosquitto

---

1. Paradigma Publicador/Suscriptor frente a Sondeo Maestro-Esclavo

En el nivel de campo (**Tecnología de Operación — OT**), protocolos como **Modbus RTU sobre RS-485** utilizan un esquema *Maestro-Esclavo por sondeo secuencial (*polling*) **altamente determinista, pero con acoplamiento directo y una capacidad limitada a un único maestro físico en el bus. Cuando múltiples sistemas de nivel superior (**SCADA, Historiador de Base de Datos, Analítica en la Nube y Paneles Móviles**) necesitan consumir las variables del proceso simultáneamente, interrogar de forma directa el bus serial resulta inviable.

El protocolo *MQTT (*Message Queuing Telemetry Transport*, estándar OASIS / ISO/IEC 20922) **resuelve esta frontera mediante un** Broker centralizado **que desacopla a los productores de datos (**Publicadores**) de los consumidores (**Suscriptores**) en tres dimensiones fundamentales:

- **Desacoplamiento de Espacio:** El nodo de borde (Raspberry Pi) no necesita conocer las direcciones IP ni la cantidad de clientes SCADA o bases de datos existentes; únicamente publica en la IP del Broker.
- **Desacoplamiento de Tiempo:** Gracias a las sesiones persistentes (`clean_session=False` / `Session Expiry Interval`) y a los **mensajes retenidos (`retain=True`)**, un suscriptor puede conectarse horas después de la emisión y recibir instantáneamente el último estado conocido.
- **Desacoplamiento de Sincronización:** Las publicaciones y recepciones operan de manera asíncrona mediante eventos sin bloquear el bucle de adquisición de control.

---

2. Modelado Matemático de Sobrecarga y Caudal de Red en MQTT

Una trama MQTT posee una cabecera fija ultracompacta de tan solo $2\ \text{bytes}$ (1 byte de tipo de control y banderas + 1 a 4 bytes de longitud restante codificada en base 128). Para un mensaje de publicación (`PUBLISH`), el tamaño total en bytes de aplicación L_{MQTT} viene dado por:

$$L_{MQTT} = L_{fija} + \leq ft(2 + L_{tópico} \right) + L_{PacketID} + L_{payload}$$

Donde:

- $L_{fija} \in [2,\ 5]\ \text{bytes}$ es la cabecera fija.
- $2 + L_{tópico}$ corresponde a los 2 bytes de longitud UTF-8 más la cadena del tópico (por ejemplo, `invernadero/zona_norte/nodo_01/estado/resumen` tiene $L_{tópico} = 45\ \text{bytes}$).
- $L_{PacketID} = 0\ \text{bytes}$ para **QoS 0**, y $L_{PacketID} = 2\ \text{bytes}$ para **QoS 1** y **QoS 2**.
- $L_{payload}$ es la longitud útil del cuerpo JSON (por ejemplo, $160\ \text{bytes}$).

Si el sistema cuenta con N nodos publicando cada T_{pub} segundos hacia M suscriptores con un nivel de calidad de servicio $q \in \{0,\ 1,\ 2\}$, el número total de paquetes MQTT intercambiados por ciclo de publicación es:

$$N_{paquetes}(q,\ M) = 2^q \cdot (1 + M)$$

- **En QoS 0 ($q=0$):** $1\ \text{paquete}$ del emisor al broker (`PUBLISH`) + $M\ \text{paquetes}$ del broker a los suscriptores $= 1 + M$.

- **En QoS 1 ($q=1$):** 2 paquetes por salto (`PUBLISH` + `PUBACK`) $= 2 \cdot (1 + M)$.
- **En QoS 2 ($q=2$):** 4 paquetes por salto (`PUBLISH` → `PUBREC` → `PUBREL` → `PUBCOMP`) $= 4 \cdot (1 + M)$.

---

3. Reglas de Ingeniería para Jerarquía de Tópicos (Alineación ISA-95)

- **Estructura de General a Específico:**
  `sitio/area/unidad_borde/categoria/variable`
  Ejemplo: `invernadero/zona_norte/nodo_01/sensores/temperatura`
- **Prohibición de Barra Inicial (`/`):** Un tópico que inicia con `/invernadero/...` introduce un nivel vacío `""` innecesario en el árbol de enrutamiento del broker.
- **Caracteres ASCII Minúsculos y Guion Bajo (`snake_case`):** Evite espacios en blanco, tildes o caracteres especiales dentro del nombre del tópico para garantizar compatibilidad total con puentes SQL, InfluxDB y Kafka.
- **Uso Correcto de Comodines (*Wildcards — *Exclusivos para `SUBSCRIBE`):**
- **Comodín de Un Nivel (`+`):** Sustituye exactamente un nivel entre barras. Ejemplo: `invernadero/+/+/sensores/temperatura` suscribe la temperatura de todos los nodos en todas las zonas.
- **Comodín Multinivel (`#`):** Sustituye cero o más niveles y **debe ubicarse obligatoriamente al final del tópico**. Ejemplo: `invernadero/zona_norte/nodo_01/#` .

---

4. Semántica de Mensajes Retenidos (`Retain`) y Testamento (`LWT`)

Mecanismo	Configuración Recomendada	Propósito Operativo Industrial
Telemetría Periódica (`.../estado/resumen`)	`QoS = 1` , `Retain = False`	Evita que un cliente reciba una lectura histórica como si acabara de ocurrir en el milisegundo actual.
Parámetros y Límites (`.../configuracion/limites`)	`QoS = 1` , `Retain = True`	Permite que cualquier panel HMI recién abierto conozca de inmediato la consigna vigente sin esperar cambios.
Estado de Salud + LWT (`.../estado/conexion`)	`QoS = 1` , `Retain = True`	Combina el **Birth Message** (`"CONECTADO"`) con el **LWT** (`"DESCONECTADO_INESPERADO"`) almacenado en el Broker.
Comandos de Actuación (`.../actuadores/comando`)	`QoS = 1` , `Retain = False`	**Crítico:** Jamás usar `Retain = True` en tópicos de comando directo para evitar re-ejecuciones accidentales al reconectar el puente.

El tiempo máximo de detección de caída abrupta por parte del Broker Mosquitto obedece la especificación OASIS MQTT:

> $$T_{\text{detección LWT}} = 1.5 \times T_{\text{keepalive}}$$

Para $T_{\text{keepalive}} = 60\ \text{s}$, si el puente sufre un corte eléctrico o pérdida de enlace físico, el Broker espera exactamente $90\ \text{s}$ sin recibir paquetes de control ni `PINGREQ` antes de despachar el mensaje LWT a todos los clientes supervisores.

PLANO Y ESQUEMÁTICO DEL CIRCUITO — MÓDULO 05

> ## Montaje en Protoboard (Estilo Tinkercad / Fritzing), Placa de Circuito Impreso Soldada (PCB FR-4), Interconexión RS-485 y Concurrencia del Puente

1. Descripción General del Sistema Físico

El sistema físico del Módulo 05 integra dos dominios de hardware interconectados mediante un bus diferencial **RS-485 Half-Duplex**:

- **Dominio de Campo OT (Arduino UNO R3 en Protoboard o Placa PCB FR-4 Soldada):** Ejecuta el firmware esclavo Modbus RTU (`ID = 1`), adquiriendo en tiempo real la temperatura (`A0`), iluminación (`A1`) y nivel de estanque ultrasónico (`D3/A2`), y gobernando las etapas de potencia mediante relés (`K52B-M-DC5` / `SRD-05VDC-SL-C`) para el motor extractor (`D8`), la bomba de riego (`D4`), el servomotor de escotilla (`D5`) y la botonera/indicadores LED locales (`D6..D12`).
- **Dominio de Borde IT/OT (Conversor USB-RS485 + Raspberry Pi 4B):** Acopla el par trenzado diferencial (`A+` , `B-` , `GND`) hacia el puerto `/dev/ttyUSB0` , donde el servicio `puente_modbus_a_mqtt_edge.py` arbitra el puerto serie con un cerrojo de exclusión mutua (`threading.Lock`) y publica/suscribe hacia el **Broker Eclipse Mosquitto 2.x**.

2. Plano 1: Montaje Completo en Protoboard del Nodo Esclavo (Estilo Tinkercad / Fritzing)

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad / Fritzing

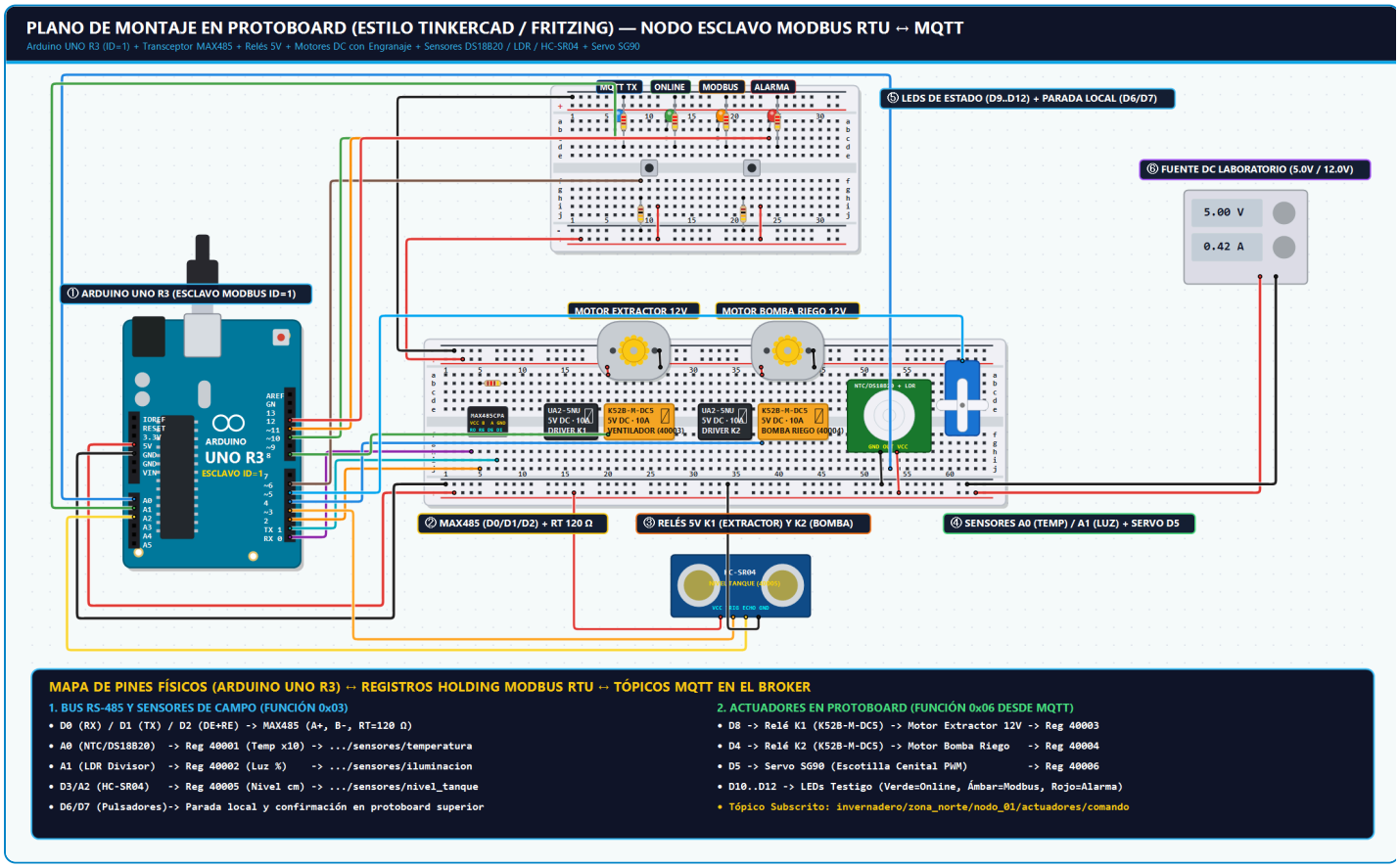


Tabla de Asignación de Pines del Arduino UNO R3 y Registros Modbus

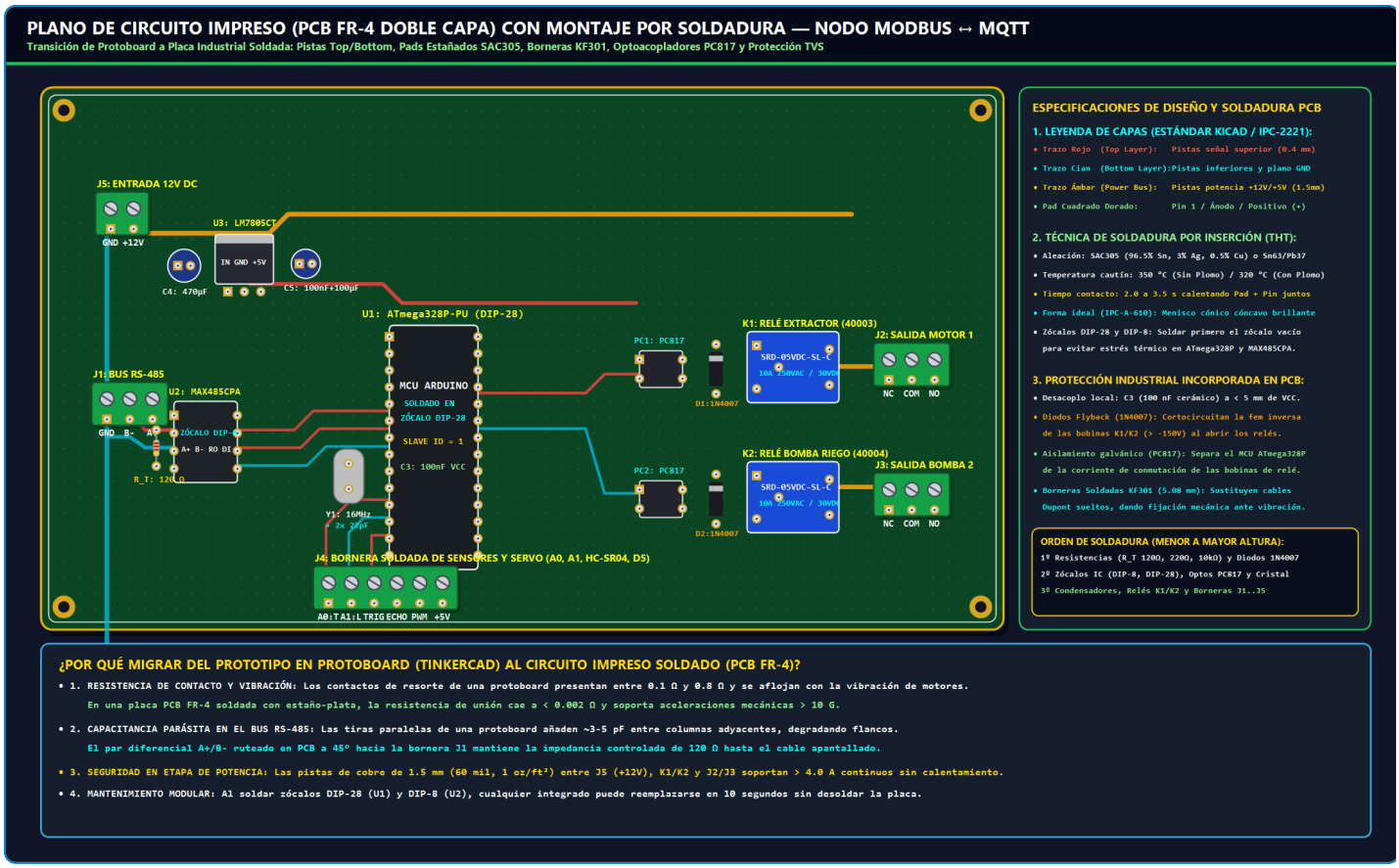
Pin Arduino UNO	Dispositivo en Protoboard / PCB	Registro Holding Modbus	Dirección de Datos y Tópico MQTT Asociado
`D0 (RX)` / `D1 (TX)`	Transceptor **MAX485** (`RO` / `DI`)	—	Tramas serie Modbus RTU a $9600\ bps$ (`8N1`).
`D2`	Transceptor **MAX485** (`DE + /RE` puenteados)	—	Control de dirección Half-Duplex (`HIGH` =TX, `LOW` =RX).
`A0`	Sensor Térmico **NTC / DS18B20**	`40001` (Índice 0)	Lectura `0x03` ($\times\ 10$) → `.../sensores/temperatura`
`A1`	Divisor Fotorresistivo **LDR +** $10\ k\Omega$	`40002` (Índice 1)	Lectura `0x03` ($0-100\ \%$) → `.../sensores/iluminacion`
`D8`	**Relé K1 (`K52B-M-DC5`)** → Motor Extractor 12V	`40003` (Índice 2)	Lectura `0x03` / Escritura `0x06` $\leq ftarrow$ `.../actuadores/comando`
`D4`	**Relé K2 (`K52B-M-DC5`)** → Motor Bomba Riego	`40004` (Índice 3)	Lectura `0x03` / Escritura `0x06` $\leq ftarrow$ `.../actuadores/comando`
`D3 (TRIG)` / `A2 (ECHO)`	Sensor Ultrasónico **HC-SR04** / Consigna	`40005` (Índice 4)	Nivel de estanque / Consigna térmica ($\times\ 10$) vía `0x06` .
`D5 (PWM)`	Micro Servomotor **SG90** (Escotilla)	`40006` (Índice 5)	Apertura proporcional de ventilación cenital ($0^\circ-90^\circ$).
`D6` , `D7`	Pulsadores Táctiles con Pull-Down $10\ k\Omega$	—	Parada local de emergencia y confirmación manual en planta.
`D9..D12`	Diodos LED 5 mm con resistencias de $220\ \Omega$	—	Indicadores luminosos: `MQTT TX` , `ONLINE` , `MODBUS` , `ALARMA` .

---

3. Plano 2: Circuito Impreso Industrial (PCB FR-4 Doble Capa) con Montaje por Soldadura

Para despliegues definitivos en invernaderos o naves industriales sometidas a vibraciones mecánicas, humedad relativa elevada y ciclos térmicos, el prototipo en protoboard se transfiere a una **placa de circuito impreso (PCB FR-4 de doble capa) con componentes soldados por inserción (`Through-Hole Technology - THT`) y bornes de tornillo (`KF301` de $5.08\ mm$)**:

📐 Plano 2: Circuito Impreso PCB FR-4 con Montaje por Soldadura



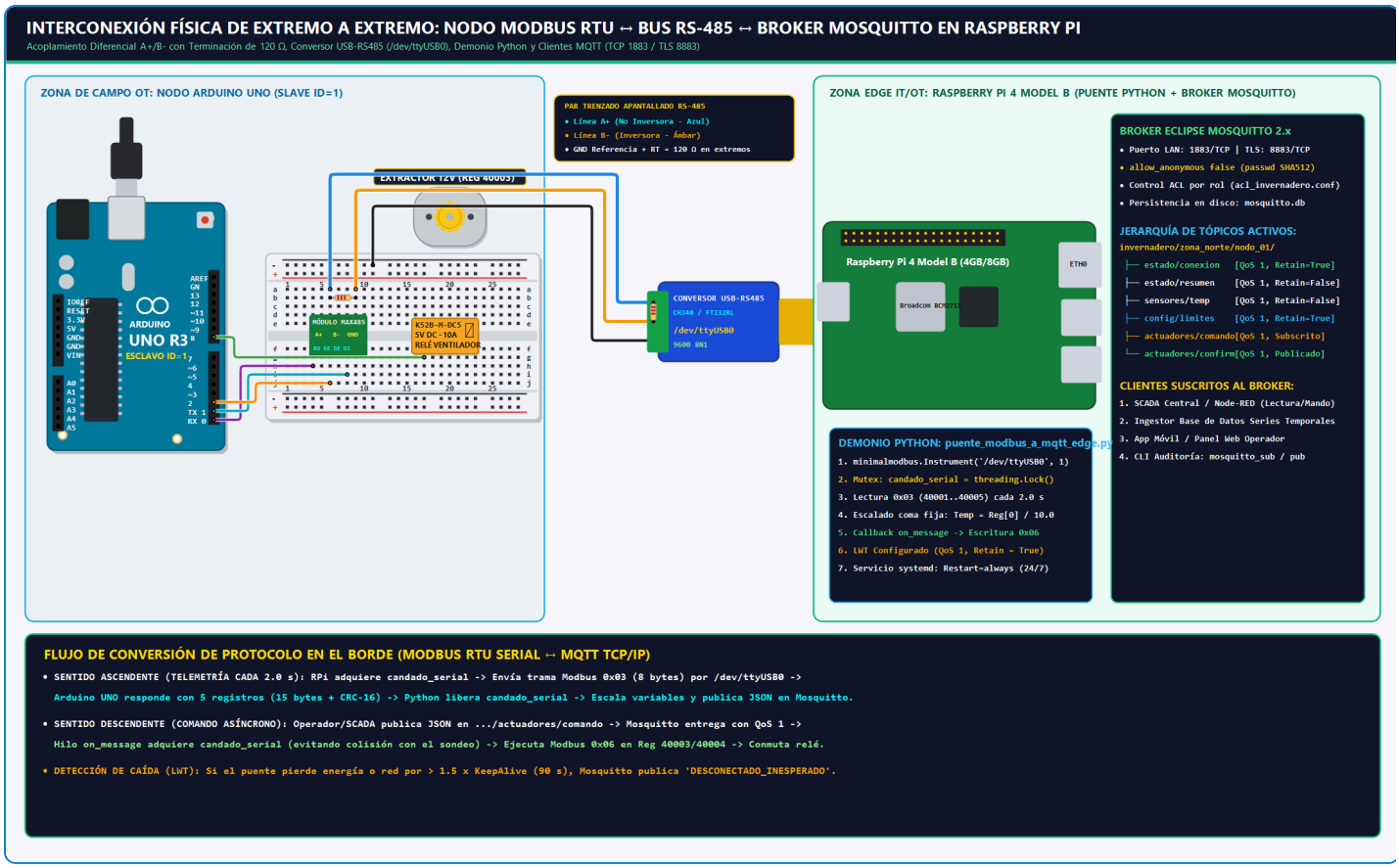
Especificaciones de Fabricación PCB y Proceso de Soldadura (`IPC-2221` / `IPC-A-610`)

Subsistema en PCB	Referencia Serigrafía	Especificación de Ruteo y Soldadura	Función de Ingeniería
Microcontrolador	`U1` (`ATmega328P-PU`) + `Y1` (16 MHz)	Soldar primero zócalo `DIP-28` sin el chip; insertar `U1` en frío. Condensador `C3` (100 nF) a < 5 mm de `VCC`.	Evita daño térmico al silicio durante la soldadura a $350\ ^\circ C$ y permite reemplazo rápido en campo.
Transceptor RS-485	`U2` (`MAX485CPA`) + `R_T` ($120\ \Omega$) + `J1`	Pistas diferenciales `A+` / `B-` de 0.6 mm ruteadas con codos a 45° hacia bornera soldada `J1`.	Elimina reflexiones de impedancia y capacitancias parásitas propias de las protoboards (3–5 pF).
Etapa de Potencia Aislada	`PC1/PC2` (`PC817`) + `D1/D2` (`1N4007`) + `K1/K2`	Pistas de potencia en cobre de 1.5 mm (60 mil, $1\ oz/ft^2$) entre relés `SRD-05VDC-SL-C` y borneras `J2` / `J3`.	Aísla galvánicamente el microcontrolador y suprime picos inductivos inversos (> -150 V) al desconectar motores.
Regulación de Energía	`U3` (`LM7805CT`) + `C4` ($470\ \mu F$) + `J5`	Pad cuadrado dorado identifica el terminal positivo (`+12V` / Ánodo). Aleación recomendada: `SAC305` o `Sn63/Pb37`.	Convierte la línea industrial de 12 V DC en 5.0 V estabilizados con resistencia de contacto $< 0.002\ \Omega$.

---

4. Plano 3: Acoplamiento Físico RS-485 hacia Raspberry Pi y Broker Mosquitto

📐 Plano 3: Acoplamiento RS-485 a Broker Mosquitto en Raspberry Pi



Reglas de Cableado e Integridad de Señal en el Bus RS-485

- **Polaridad Homóloga Directa (Sin Cruce):** A diferencia de UART punto a punto (donde `TX` se cruza con `RX`), en el bus diferencial RS-485 el borne `A+` **del módulo MAX485 se conecta directamente al borne** `A+` **del conversor USB-RS485**, y el borne `B-` **se conecta directamente al borne** `B-`.
- **Doble Terminación Resistiva ($R_T = 120\ \Omega$):** Instale una resistencia de $120\ \Omega$ entre `A+` y `B-` en el nodo Arduino UNO (final de línea) y otra de $120\ \Omega$ en la bornera del conversor USB-RS485 (cabecera de línea en la Raspberry Pi).
- **Referencia Común de Masa (`GND`):** Conecte siempre un tercer conductor de referencia `GND` entre la masa lógica del nodo de campo y el terminal `GND` del conversor USB-RS485 para mantener el voltaje de modo común dentro del rango operativo del transceptor ($-7\ V$ a $+12\ V$).

---

5. Plano 4: Jerarquía de Tópicos ISA-95, Niveles QoS y Máquina de Estados LWT

📐 Plano 4: Mapa de Jerarquía de Tópicos y Flujo QoS MQTT

ARQUITECTURA DE TÓPICOS JERÁRQUICOS ISA-95, COMODINES (+ / #), NIVELES QoS Y CICLO LWT

Diseño de Espacio de Nombres MQTT, Handshakes de Entrega (QoS 0, QoS 1, QoS 2), Mensajes Retenidos y Testamento de Última Voluntad

1. ÁRBOL JERÁRQUICO DE TÓPICOS (PLANTA → ZONA → NODO → RECURSO)

```
invernadero/                          [Nivel 1: Planta / Sitio]
└── zona_norte/                       [Nivel 2: Área / Célula]
    └── nodo_01/                      [Nivel 3: Unidad Edge / ID=1]
        ├── estado/
        │   ├── conexion              -> QoS 1 | Retain=True  (Online / LWT)
        │   └── resumen               -> QoS 1 | Retain=False (JSON Completo)
        ├── sensores/
        │   ├── temperatura           -> QoS 1 | Retain=False (24.5 °C)
        │   ├── iluminacion           -> QoS 1 | Retain=False (68 %)
        │   └── nivel_tanque          -> QoS 0 | Retain=False (142 cm)
        ├── configuracion/
        │   └── limites               -> QoS 1 | Retain=True  (Consigna 28.0 °C)
        └── actuadores/
            ├── comando               -> QoS 1 | Retain=False (Hacia Puente)
            └── confirmacion          -> QoS 1 | Retain=False (Eco Modbus 0x06)
```

REGLAS DE SUSCRIPCIÓN CON COMODINES (SOLO EN SUBSCRIBE):

- **Comodín Un Nivel (+):** `invernadero/+/+/sensores/temperatura`
- **Comodín Multinivel (#):** `invernadero/zona_norte/nodo_01/#`
- Antipatrones Prohibidos: Nunca iniciar con '/' ni usar espacios o '#'

2. NIVELES DE CALIDAD DE SERVICIO (QoS 0, QoS 1 Y QoS 2)

QoS 0 — A LO SUMO UNA VEZ (AT MOST ONCE / FIRE AND FORGET)

```
Emisor  ──────[ PUBLISH (QoS=0) ]──────►  Broker Mosquitto   (1 paquete MQTT, sin acuse)
```

Uso recomendado: Variables de alta frecuencia no críticas donde el siguiente muestreo reemplaza al anterior.

QoS 1 — AL MENOS UNA VEZ (AT LEAST ONCE — ESTÁNDAR INDUSTRIAL IIoT)

```
Emisor  ──────[ 1. PUBLISH (PacketId=K, DUP=0) ]──────►  Broker Mosquitto
Emisor  ◄──────[ 2. PUBACK  (PacketId=K)        ]──────  Broker Mosquitto   (2 paquetes MQTT)
```

Uso recomendado: Telemetría agregada, alarmas, LWT y comandos de actuación con idempotencia.

QoS 2 — EXACTAMENTE UNA VEZ (EXACTLY ONCE — HANDSHAKE DE 4 FASES)

```
Emisor ──[1. PUBLISH]──► Broker ──[2. PUBREC]──► Emisor ──[3. PUBREL]──► Broker ──[4. PUBCOMP]
```

Sobrecarga: 4 paquetes por mensaje. Reservado para facturación fiscal o dosificación no repetible.

Fórmula Paquetes Totales (1 Emisor + M Suscriptores en QoS q): $N_{paq} = (2^q) * (1 + M)$

3. MENSAJES RETENIDOS (RETAIN=TRUE) Y MÁQUINA DE ESTADOS DEL TESTAMENTO (LWT)

- FASE 1 (CONEXIÓN INICIAL): Puente envía paquete CONNECT registrando el LWT en RAM del Broker:
  Tópico LWT: .../estado/conexion | Payload: 'DESCONECTADO_INESPERADO' | QoS=1 | Retain=True
- FASE 2 (MENSAJE DE NACIMIENTO / BIRTH): Inmediatamente tras recibir CONNACK, el puente publica:
  Tópico: .../estado/conexion | Payload: 'CONECTADO' | QoS=1 | Retain=True
- FASE 3 (SUPERVISIÓN KEEP-ALIVE): El cliente envía PINGREQ si no hay tráfico en T_keepalive = 60 s.
- FASE 4 (DISPARO AUTOMÁTICO LWT): Si transcurre T_LWT = 1.5 * T_keepalive = 90 s sin PINGREQ ni DISCONNECT,
  el Broker Mosquitto cierra el socket y publica automáticamente el LWT retenido a todos los SCADA suscritos.

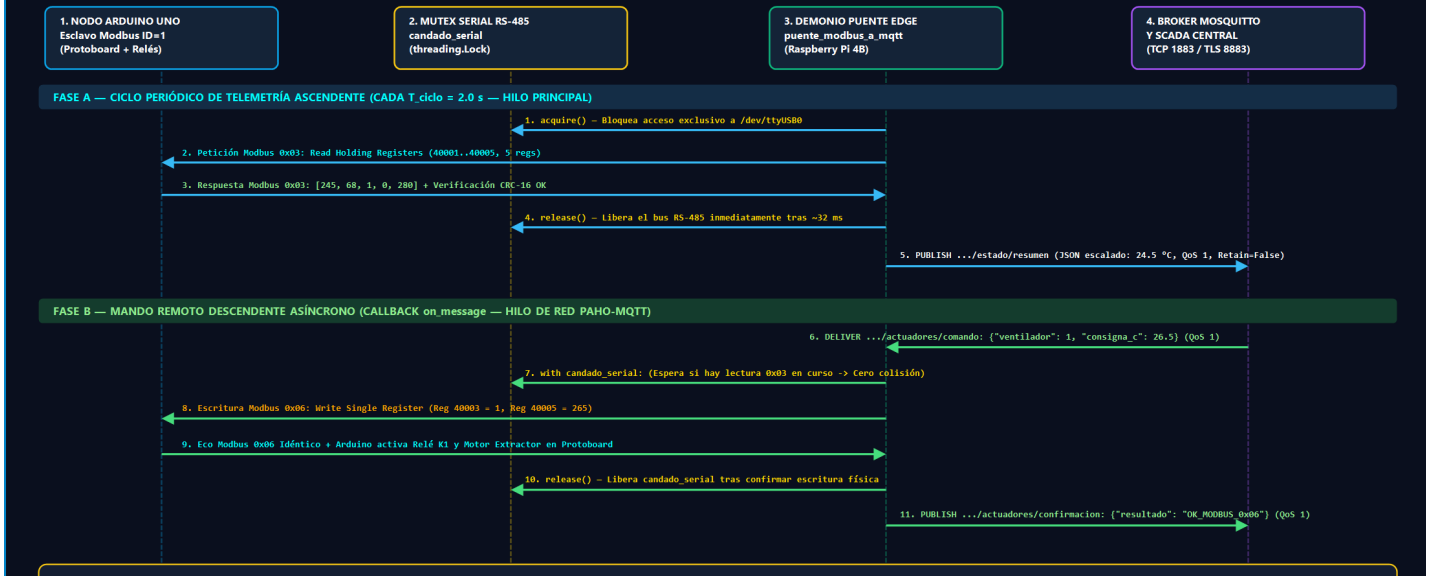
---

6. Plano 5: Secuencia Bidireccional y Exclusión Mutua (`threading.Lock`)

📐 Plano 5: Diagrama de Secuencia del Puente Bidireccional Modbus a MQTT

DIAGRAMA DE SECUENCIA BIDIRECCIONAL Y EXCLUSIÓN MUTUA (MODBUS RTU ↔ PUENTE PYTHON ↔ MOSQUITTO)

Arbitraje del Bus RS-485 con threading.Lock() entre el Bucle de Sondeo (0x03) y el Hilo Asíncrono de Comandos MQTT (0x06)



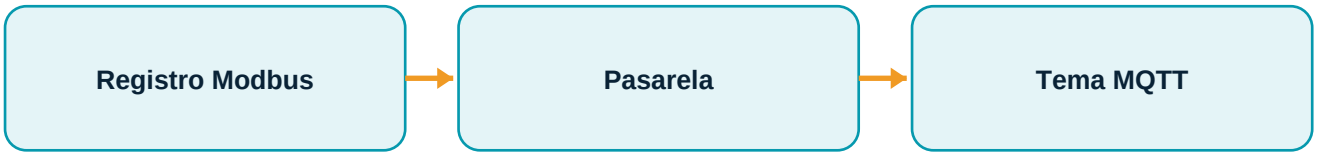
¿POR QUÉ ES OBLIGATORIO EL MUTEX (threading.Lock) EN EL PUENTE MODBUS ↔ MQTT?

- En paho-mqtt, client.loop_start() ejecuta las recepciones de red (on_message) en un hilo secundario asíncrono, mientras que el bucle principal while True ejecuta el sondeo periódico de los sensores cada 2.0 segundos.
- El bus RS-485 es estrictamente Half-Duplex: si el hilo on_message intenta enviar una trama 0x06 por /dev/ttyUSB0 en el mismo milisegundo en que el bucle principal espera los 15 bytes de respuesta de una lectura 0x03, los bytes de ambas tramas se entrelazan en el UART, corrompiendo el CRC-16 y provocando una excepción de Timeout.
- Con 'with candado_serial:', toda transacción Petición-Respuesta + Silencio de 3.5 caracteres es 100% ATÓMICA.
- Latencia máxima de encolamiento del comando: T_espera_max <= T_transaccion_0x03 (~32 ms a 9600 bps).

Capítulo 18 · Proyecto: pasarela bidireccional Modbus a MQTT

Conceptos clave

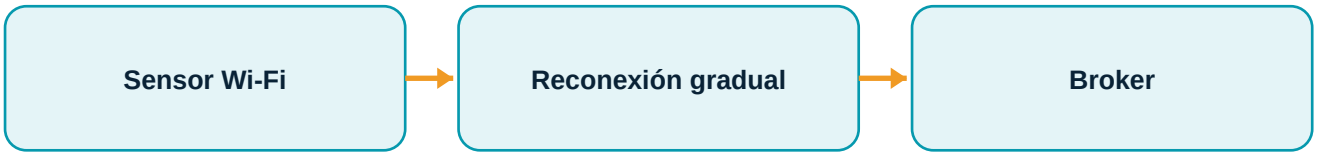
Documenta el mapeo de direcciones, escala y unidades. Limita los comandos de escritura a parámetros permitidos y devuelve confirmación.



Capítulo 19 · Proyecto: nodos Wi-Fi con ESP32 y Arduino R4

Conceptos clave

Reintenta con espera creciente y un límite. Distingue la conectividad de red de la disponibilidad del broker.



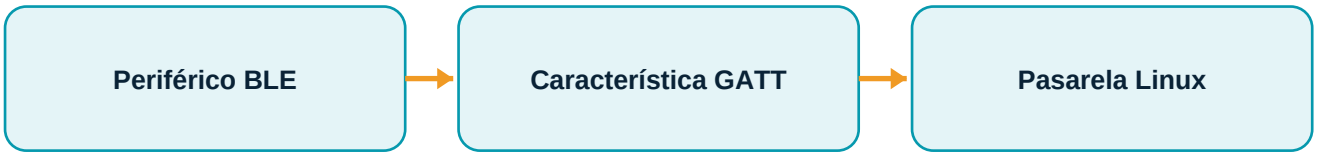
Ejemplo complementario

```
def intervalos_reintento(base=1, maximo=60):
    pausa = base
    while True:
        yield pausa
        pausa = min(pausa * 2, maximo)
```

Capítulo 20 · Bluetooth Low Energy: GATT y pasarela BlueZ

Conceptos clave

Distingue anuncios, lecturas y notificaciones. Conserva las unidades y la escala al decodificar bytes.



GUÍA PASO A PASO DE LABORATORIO 06: NODOS INALÁMBRICOS E INTEGRACIÓN HÍBRIDA (WI-FI, BLE, LORA Y DEEP SLEEP)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Nodos Inalámbricos Autónomos, Servidor BLE GATT (`0x181A`), Radio de Largo Alcance LoRa (`SX1278`) y Sueño Profundo (`Deep Sleep`) con Sensor Ambiental Bosch `BME280`

Plataformas: Arduino Nano ESP32 (`3.3 V` Nativo) + Raspberry Pi 4 Model B / 5 (`64-bit`)

---

1. Objetivo de Ingeniería del Laboratorio

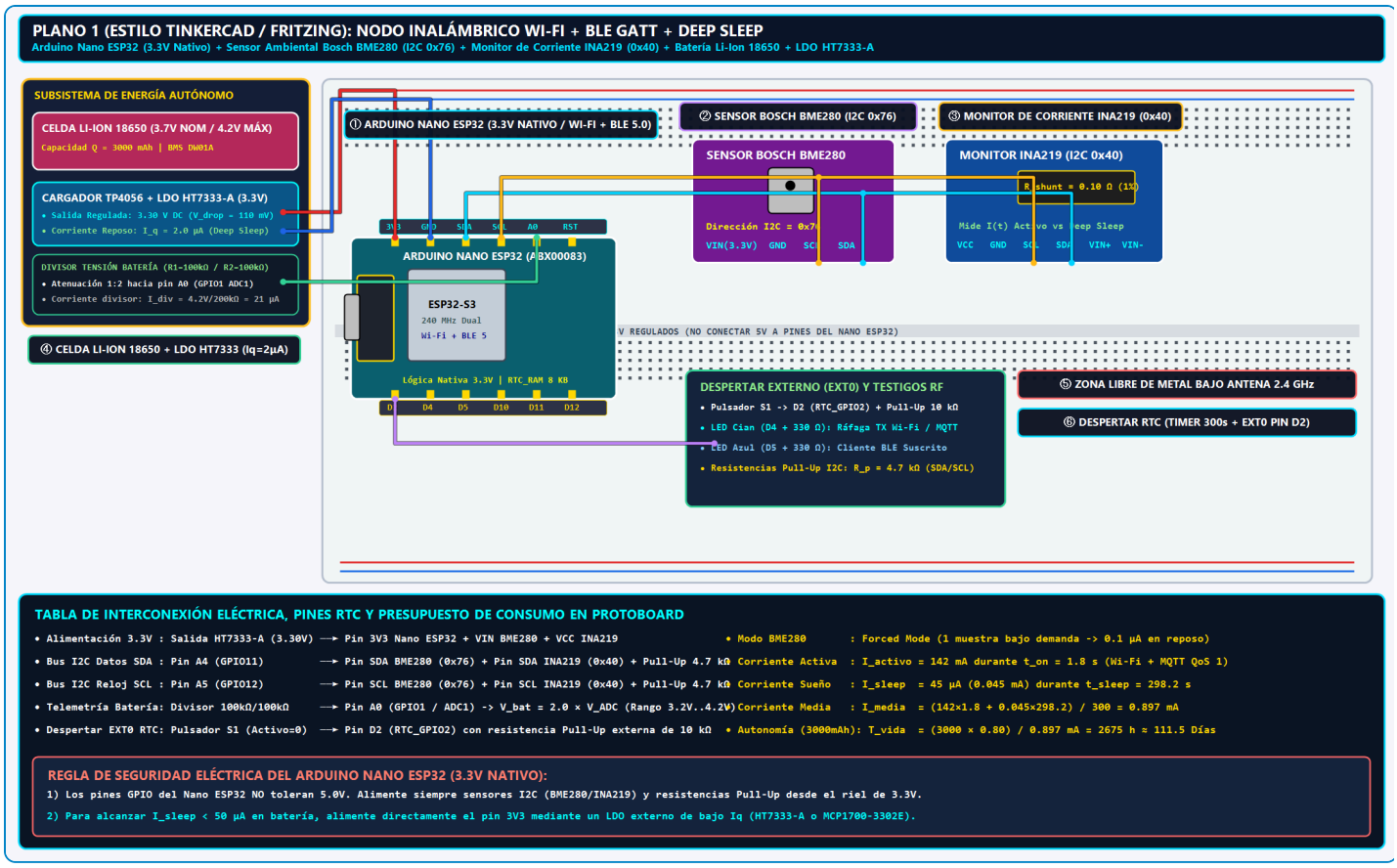
Diseñar, ensamblar en protoboard (estilo Tinkercad / Fritzing) y en placa de circuito impreso soldada (`PCB FR-4`), programar y auditar una **arquitectura inalámbrica híbrida de tres bandas/protocolos (`Wi-Fi` , `BLE 5.0` y `LoRa Sub-GHz`)** convergente hacia el Broker Eclipse Mosquitto en la Raspberry Pi:

- **Nodo Wi-Fi + MQTT de Ultrabajo Consumo (`Deep Sleep`)**: Basado en Arduino Nano ESP32 alimentado por celda `Li-Ion 18650 (3.7 V / 3000 mAh)` y regulador LDO `HT7333-A (I_q = 2.0 µA)` , leyendo temperatura, humedad y presión barométrica mediante el sensor Bosch `BME280` en **Modo Forzado (`Forced Mode`)** y reconectando con **Backoff Exponencial + Jitter**.
- **Nodo Periférico Bluetooth Low Energy (`BLE GATT 0x181A`)**: Exponiendo las características estándar `0x2A6E` (Temperatura), `0x2A6F` (Humedad) y `0x2A6D` (Presión) con notificaciones asíncronas (`CCCD 0x2902`) hacia el demonio Python `bleak` en la Raspberry Pi.
- **Concentrador de Largo Alcance** `LoRa SX1278 (433 / 915 MHz)` : Acoplado por bus `SPI0` e interrupción física `DIO0 (GPIO 25)` a la Raspberry Pi para enlaces rurales de `2 km` a `15 km` .

---

2. Montaje Físico en Protoboard (Estilo Tinkercad) y en Placa PCB Soldada

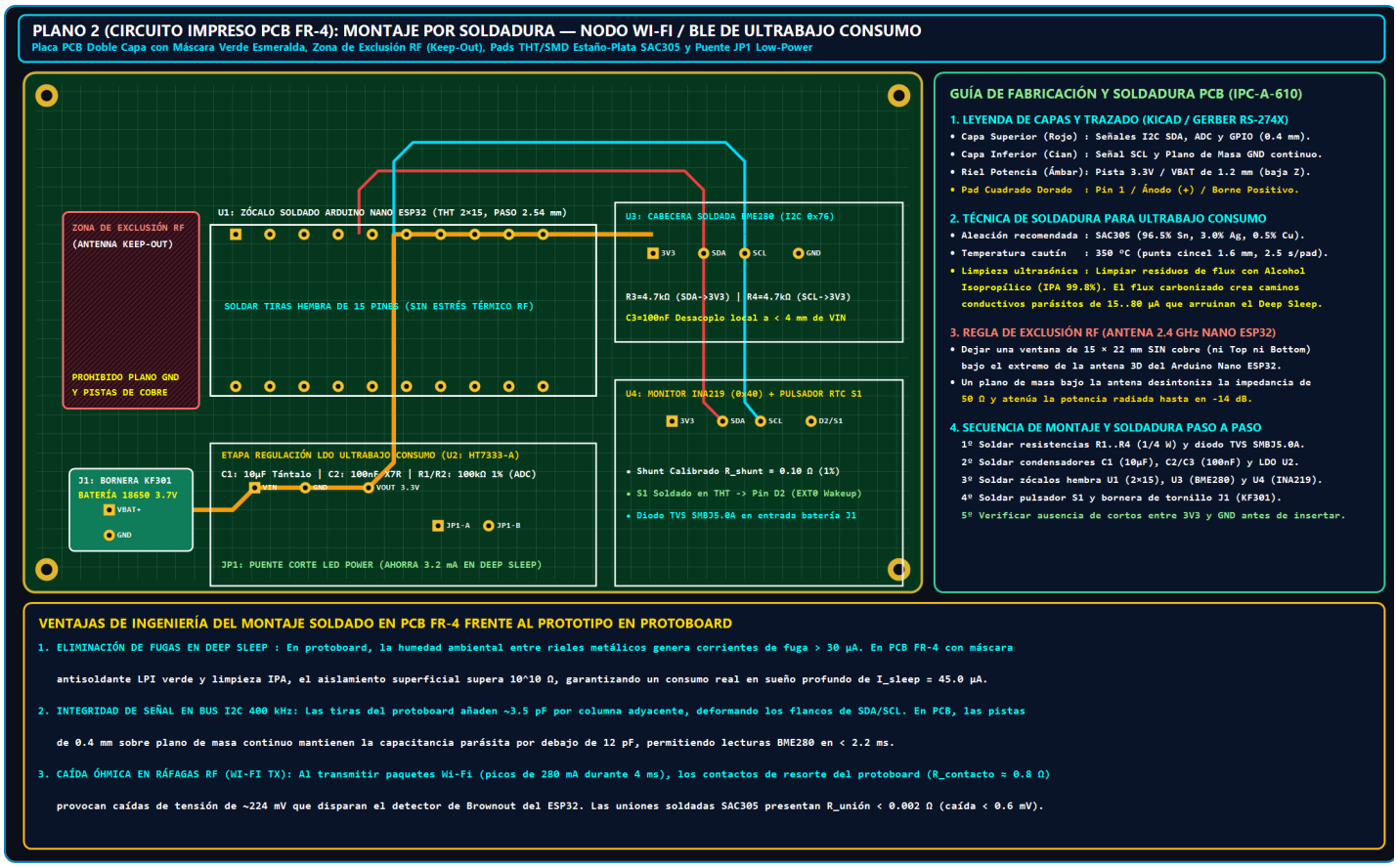
📐 **Plano 1: Montaje en Protoboard Estilo Tinkercad del Nodo Wi-Fi / BLE con BME280, INA219 y Batería 18650**



2.1 Tabla Maestra de Conexiones del Nodo Arduino Nano ESP32 (`Lógica Estricta 3.3 V`)

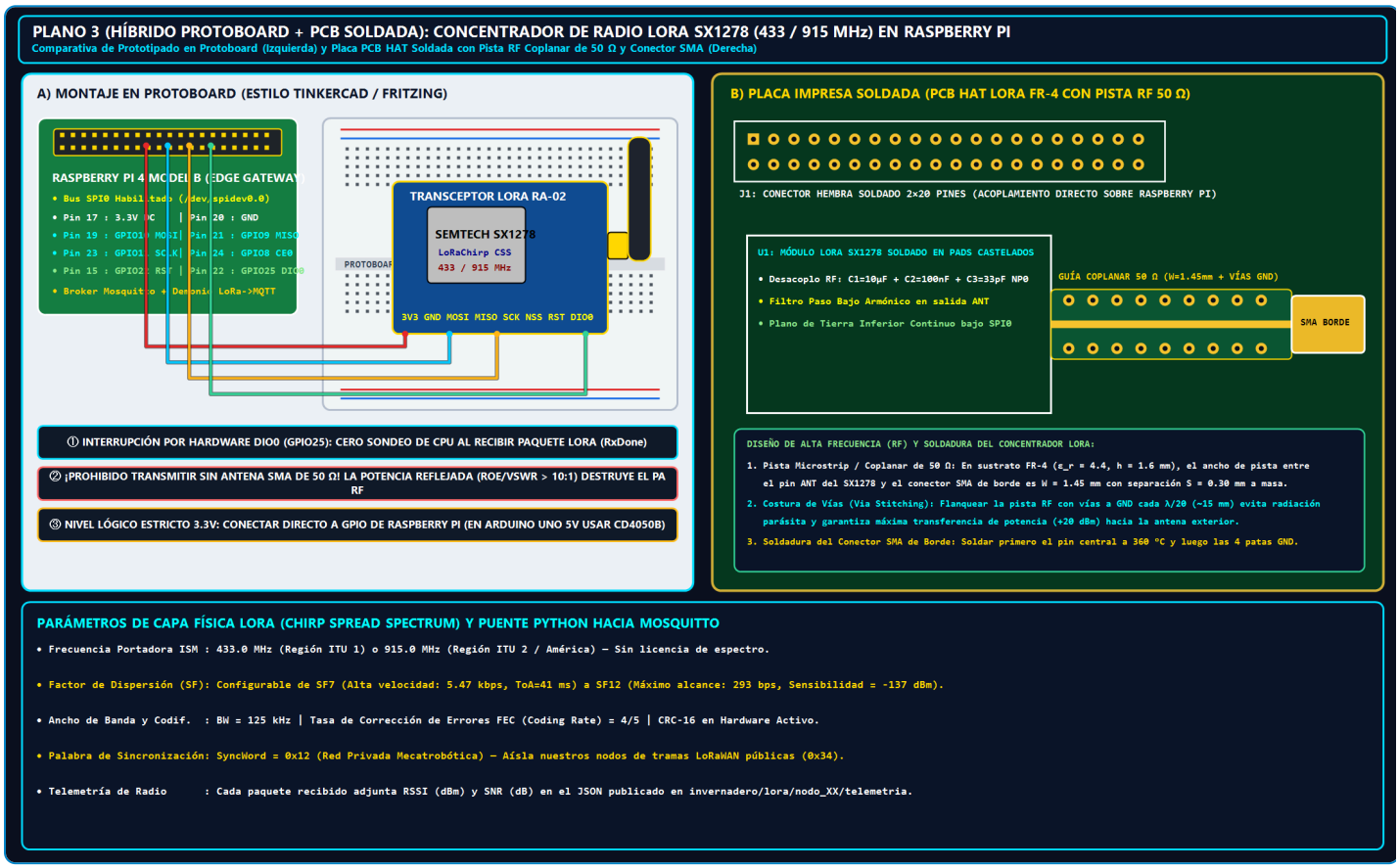
Subsistema / Componente	Pin del Componente	Pin en Arduino Nano ESP32	Función Eléctrica y Observaciones de Ingeniería
Regulador LDO HT7333-A	`VOUT (3.30 V)`	`Pin 3V3`	Alimentación directa regulada con corriente de reposo $I_q = 2.0\ \mu A$
Sensor Bosch BME280 (`0x76`)	`VIN / GND / SDA / SCL`	`3V3 / GND / A4 / A5`	Bus I^2C a $400\ kHz$ con resistencias Pull-Up de $4.7\ k\Omega$ a $3.3\ V$
Monitor Corriente INA219 (`0x40`)	`VCC / GND / SDA / SCL`	`3V3 / GND / A4 / A5`	Shunt de precisión $R_{shunt} = 0.10\ \Omega\ (1\%)$ en serie con el riel de $3.3\ V$
Divisor Tensión Batería	Punto medio `100kΩ/100kΩ`	`Pin A0 (GPIO1 / ADC1)`	Lectura calibrada con `analogReadMilliVolts(A0)` ($V_{bat} = 2.0 \times V_{ADC}$)
Pulsador Despertar RTC (`S1`)	Contacto NA a `GND`	`Pin D2 (RTC_GPIO2)`	Interrupción externa `EXT0` con resistencia Pull-Up externa de $10\ k\Omega$ a $3.3\ V$
Indicadores LED RF	Ánodo vía $330\ \Omega$	`Pin D4 (Wi-Fi)` / `D5 (BLE)`	Testigos luminosos activos únicamente durante la ventana despierta

📐 Plano 2: Placa de Circuito Impreso PCB FR-4 con Montaje por Soldadura del Nodo Inalámbrico



3. Montaje del Concentrador LoRa `SX1278` en Raspberry Pi (`SPI0` + `IRQ DIO0`)

📐 Plano 3: Interconexión en Protoboard y Placa PCB HAT Soldada del Transceptor LoRa SX1278



> **Regla Crítica de Radiofrecuencia (ROE / VSWR):** Jamás alimente ni ejecute el script de transmisión/recepción del módulo `SX1278` sin haber enroscado previamente su **antena helicoidal o dipolo de $50\ \Omega$ en el conector SMA**. Transmitir a $+20\ \text{dBm}$ con el puerto RF abierto refleja el 100% de la energía hacia el transistor de salida del amplificador de potencia (PA), destruyendo el chip en milisegundos.

Pin del Módulo LoRa `SX1278 (Ra-02)`	Pin Físico en Raspberry Pi (40 Pines)	GPIO BCM	Función en el Bus `SPI0` y Control de Radio
`VCC (3.3V)`	**Pin 17**	Riel `3.3V DC`	Alimentación limpia capaz de entregar picos de $120\ \text{mA}$
`GND`	**Pin 20**	Riel `GND`	Masa común de referencia digital y blindaje RF
`MOSI`	**Pin 19**	`GPIO 10 (SPI0_MOSI)`	Escritura de registros y carga de FIFO hacia el SX1278
`MISO`	**Pin 21**	`GPIO 9 (SPI0_MISO)`	Lectura de registros, RSSI, SNR y paquete recibido
`SCK`	**Pin 23**	`GPIO 11 (SPI0_SCLK)`	Reloj síncrono SPI a $4.0\ \text{MHz}$ (Modo `0,0`)
`NSS (CS)`	**Pin 24**	`GPIO 8 (SPI0_CE0_N)`	Selección de esclavo activa en nivel bajo
`RST`	**Pin 15**	`GPIO 22`	Reinicio físico por hardware del transceptor
`DIO0`	**Pin 22**	`GPIO 25`	Interrupción por flanco de subida (`RxDone` / `TxDone`)

---

4. Despliegue de Software y Verificación en Laboratorio

Paso 1: Habilitar `SPI0` y `Bluetooth BlueZ` en Raspberry Pi e instalar dependencias

```
sudo raspi-config nonint do_spi 0
sudo systemctl enable --now bluetooth
sudo apt update && sudo apt install -y python3-pip python3-spidev python3-rpi.gpio bluez
pip3 install --break-system-packages bleak paho-mqtt
```

Paso 2: Compilar y cargar los firmwares en Arduino Nano ESP32

- En Arduino IDE 2.x, instalar el paquete de placas **Arduino ESP32 Boards** y las librerías `PubSubClient`, `Adafruit BME280 Library` y `Adafruit INA219`.
- Para ensayar el nodo **Wi-Fi + MQTT con Deep Sleep**, abrir `codigo_fuente/nodo_wifi_mqtt_bme280_deepsleep.ino`, ajustar `WIFI_SSID`, `WIFI_PASSWORD` y `MQTT_BROKER_IP`, y subir al Nano ESP32.
- Para ensayar el nodo **Periférico BLE GATT (`0x181A`)**, subir `codigo_fuente/periferico_ble_gatt_ambiental.ino`.

Paso 3: Lanzar las pasarelas de borde en la Raspberry Pi y auditar en Mosquitto

```
# Terminal 1: Iniciar pasarela asíncrona BLE GATT -> MQTT
python3 codigo_fuente/pasarela_ble_a_mqtt_bleak.py

# Terminal 2: Iniciar concentrador LoRa SX1278 SPI0 -> MQTT
python3 codigo_fuente/transceptor_lora_sx1278_mqtt_bridge.py

# Terminal 3: Auditar en tiempo real la convergencia de las 3 tecnologías RF en el Broker
mosquitto_sub -h localhost -u scada_central -P Clave -t 'invernadero/#' -q 1 -v
```

LISTADO DE MATERIALES (BOM) — MÓDULO 06: NODOS INALÁMBRICOS E INTEGRACIÓN HÍBRIDA

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Nodos Inalámbricos (`Wi-Fi` , `BLE 5.0` , `LoRa SX1278`), Gestión Energética `Li-Ion 18650` y Montaje en Protoboard / PCB Soldada

---

1. Microcontroladores, Transceptores RF y Computadora de Borde

Ítem	Cant.	Designación PCB	Componente / Referencia Industrial	Especificación Técnica Clave	Función en el Módulo 06
01	2	`U1`	**Arduino Nano ESP32** (`ABX00083`)	SoC `ESP32-S3` Dual-Core Xtensa LX7 @ $240\ \text{MHz}$, $8\ \text{KB}$ RTC SRAM, Lógica nativa $3.3\ \text{V}$, Wi-Fi `802.11b/g/n` + `BLE 5.0`	Nodo 1: Wi-Fi + Deep Sleep; Nodo 2: Servidor Periférico BLE GATT (`0x181A`)
02	1	`GW1`	**Raspberry Pi 4 Model B (4 GB) o RPi 5**	Broadcom Quad-Core 64-bit, Bluetooth `BlueZ 5.x` (`/dev/hci0`), Bus `SPI0` (`/dev/spidev0.0`), Broker Mosquitto	Pasarela de Borde Híbrida (`BLE` + `LoRa` + `Wi-Fi` → `MQTT`)
03	2	`U_LORA`	**Módulo Transceptor LoRa** `Ra-02 (SX1278)`	Banda ISM Sub-GHz ($433\ \text{MHz}$ / $915\ \text{MHz}$), Modulación CSS (`SF7..SF12`), Sensibilidad $-137\ \text{dBm}$, SPI $3.3\ \text{V}$	Enlace de largo alcance rural ($2\ \text{km}$ a $15\ \text{km}$) con CRC-16 en hardware
04	2	`ANT1`	**Antena Dipolo / Helicoidal SMA** $50\ \Omega$	Impedancia característica $Z_0 = 50\ \Omega$, Ganancia $G = +3.0\ \text{dBi}$, ROE (VSWR) $< 1.5:1$, Cable pigtail IPEX a SMA	Radiación eficiente y protección del amplificador de potencia del `SX1278`

2. Sensórica Ambiental I^2C e Instrumentación de Consumo Energético

Ítem	Cant.	Designación PCB	Componente / Referencia Industrial	Especificación Técnica Clave	Función en el Módulo 06
05	2	`U3`	**Sensor Ambiental Bosch** `BME280`	Interfaz I^2C (`0x76`) / SPI $3.3\ \text{V}$, Temp ($\pm 0.5\ ^\circ\text{C}$), Humedad ($\pm 3\%\ \text{RH}$), Presión ($\pm 1.0\ \text{hPa}$), $I_{sleep} = 0.1\ \mu\text{A}$	Adquisición meteorológica en Modo Forzado (`Forced Mode`) bajo demanda
06	1	`U4`	**Monitor High-Side Texas Instruments** `INA219`	Interfaz I^2C (`0x40`), ADC de $12\ \text{bits}$, resistencia Shunt $R_{shunt} = 0.10\ \Omega$ (1%), rango $\pm 3.2\ \text{A}$	Medición de perfil de corriente $I(t)$ en ráfaga RF frente a sueño profundo

3. Subsistema de Alimentación por Batería de Ultrabajo Consumo (`Deep Sleep`)

Ítem	Cant.	Designación PCB	Componente / Referencia Industrial	Especificación Técnica Clave	Función en el Módulo 06
07	1	`BAT1`	**Celda Recargable Li-Ion** `18650`	Tensión nominal $3.7\ V$ (carga plena $4.20\ V$, corte $3.0\ V$), Capacidad nominal $Q = 3000\ mAh$, química INR/NCR	Fuente primaria de energía para operación autónoma $> 110\ días$
08	1	`CHG1`	**Módulo Cargador Li-Ion** `TP4056` **con BMS**	Corriente de carga CC/CV hasta $1.0\ A$ más integrado de protección `DW01A` + doble MOSFET `FS8205A`	Carga segura por USB-C / panel solar de $5\ V$ y corte por sobredescarga
09	2	`U2`	**Regulador LDO** `HT7333-A` **o** `MCP1700-3302E`	Salida fija $3.30\ V$ ($I_{\max} = 250\ mA$), tensión de caída $V_{drop} = 110\ mV$, **corriente de reposo** $I_q = 2.0\ \mu A$	Sustituye reguladores `AMS1117` ($I_q > 5\ mA$) para lograr $I_{sleep} = 45\ \mu A$
10	2	`R1, R2`	**Resistencias de Película Metálica** $100\ k\Omega$	Tolerancia $\pm 1\%$, potencia $1/4\ W$, bajo coeficiente térmico ($\pm 50\ ppm/^\circ C$)	Divisor resistivo $1:2$ hacia el pin `A0 (ADC1)` para telemetría de batería
11	2	`C1, C2`	**Condensadores de Desacoplo** $10\ \mu F$ **y** $100\ nF$	$10\ \mu F$ Tántalo ($16\ V$, baja ESR) + $100\ nF$ Cerámico multicapa `X7R`	Absorción de picos transitorios de corriente ($240\ mA$) al transmitir en Wi-Fi

4. Insumos de Prototipado en Protoboard (Tinkercad) y Soldadura en Placa PCB FR-4

Ítem	Cant.	Designación PCB	Componente / Herramienta de Montaje	Especificación Técnica Clave	Función en el Módulo 06
12	2	`PROTO`	**Protoboard de 830 Puntos (** `MB-102` **)**	Contactos de bronce fosforoso niquelado, doble riel de alimentación lateral	Ensamble rápido sin soldadura (estilo Tinkercad / Fritzing)
13	2	`PCB1`	**Placa Virgen o Fabricada** `PCB FR-4` **Doble Capa**	Fibra de vidrio epoxi `FR-4` ($h = 1.6\ mm$, cobre $1\ oz/ft^2 = 35\ \mu m$), máscara antisoldante verde esmeralda	Montaje definitivo por soldadura con zona de exclusión RF y pista de $50\ \Omega$
14	4	`U1, U3, U4`	**Tiras de Pines Hembra Paso** $2.54\ mm$ **(** `THT` **)**	Contactos dorados de $15\ pines$ (para Nano ESP32) y $4/6\ pines$ (para `BME280` / `INA219`)	Permite insertar y retirar los módulos sobre la PCB sin aplicar calor directo
15	2	`J1, J2`	**Borneras de Tornillo Soldables** `KF301-2P`	Paso $5.08\ mm$, capacidad $300\ V\ /\ 10\ A$, fijación mecánica por jaula de latón	Fijación firme de los cables de la celda `18650` sobre la placa PCB
16	1	`SOLD`	**Soldadura** `SAC305` **+ Alcohol Isopropílico** `IPA`	Aleación sin plomo ($96.5\% Sn, 3.0\% Ag, 0.5\% Cu$) + `IPA 99.8%` para limpieza de flux	Eliminación de corrientes de fuga superficiales entre pads durante el Deep Sleep

MANUAL DE INGENIERÍA DE RADIOFRECUENCIA: WI-FI, BLE GATT, LORA Y SUEÑO PROFUNDO (DEEP SLEEP)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Fundamentos Matemáticos de Presupuesto de Enlace RF, Arquitectura GATT Bluetooth SIG, Modulación Chirp Spread Spectrum (`LoRa`) y Modelado Energético de Baterías

---

1. El Trilema de las Comunicaciones Inalámbricas en IoT

En ingeniería de telecomunicaciones industriales, ningún protocolo de radiofrecuencia puede maximizar simultáneamente **tres variables físicas contrapuestas**:

- **Alcance Geográfico (d)**
- **Caudal de Datos o Ancho de Banda (R_b)**
- **Autonomía Energética en Batería (T_{vida})**

Parámetro de Ingeniería	Wi-Fi (`IEEE 802.11n`)	Bluetooth Low Energy (`BLE 5.0`)	Radio `LoRa Sub-GHz (SX1278)`
Banda de Frecuencia ISM	$2.4\ GHz$ / $5.0\ GHz$	$2.4\ GHz$ (40 canales de $2\ MHz$)	$433\ MHz$ / $868\ MHz$ / $915\ MHz$
Velocidad Efectiva (R_b)	$20\ Mbps$ a $72\ Mbps$	$125\ kbps$ a $2.0\ Mbps$	$293\ bps$ (`SF12`) a $5.47\ kbps$ (`SF7`)
Alcance Típico en Campo	$30\ m$ a $90\ m$	$15\ m$ a $60\ m$	$2.0\ km$ **a** $15.0\ km$
Sensibilidad del Receptor ($S_{\min}$)	$-75\ dBm$	$-95\ dBm$	$-123\ dBm$ **(** `SF7` **)** **a** $-137\ dBm$ **(** `SF12` **)**
Corriente Pico en Transmisión	$180\ mA$ a $260\ mA$	$12\ mA$ a $18\ mA$	$90\ mA$ a $120\ mA$ ($+17..20\ dBm$)
Pila TCP/IP Nativa	Sí (Directo a Broker MQTT)	No (Requiere Pasarela GATT → IP)	No (Requiere Concentrador SPI → IP)

---

2. Presupuesto de Enlace RF (Link Budget) y Ecuación de Friis

La potencia electromagnética recibida P_{rx} (expresada en dBm, decibelios referidos a $1\ mW$) en la antena del servidor de borde viene determinada por el **balance logarítmico de potencias, ganancias y pérdidas**:

$$P_{rx}\ (dBm) = P_{tx}\ (dBm) + G_{tx}\ (dBi) + G_{rx}\ (dBi) - FSPL(d, f)\ (dB) - L_{misc}\ (dB)$$

Donde la **Pérdida de Propagación en el Espacio Libre de Friis ($FSPL$)** en función de la distancia $d\ (km)$ y la frecuencia portadora $f\ (MHz)$ es:

$$FSPL\ (dB) = 20 \cdot \log_{10}(d_{km}) + 20 \cdot \log_{10}(f_{MHz}) + 32.44$$

2.1 ¿Por qué la banda Sub-GHz ($433\ MHz$) penetra mucho mejor que $2.4\ GHz$?

A igual distancia d, la diferencia de atenuación en espacio libre entre $2400\ MHz$ (`Wi-Fi` / `BLE`) y $433\ MHz$ (`LoRa`) es:

$$\DeltaFSPL = 20 \cdot \log_{10}\left((2400) / (433)\right) = 20 \cdot \log_{10}(5.5427) \approx +14.87 \text{ dB}$$

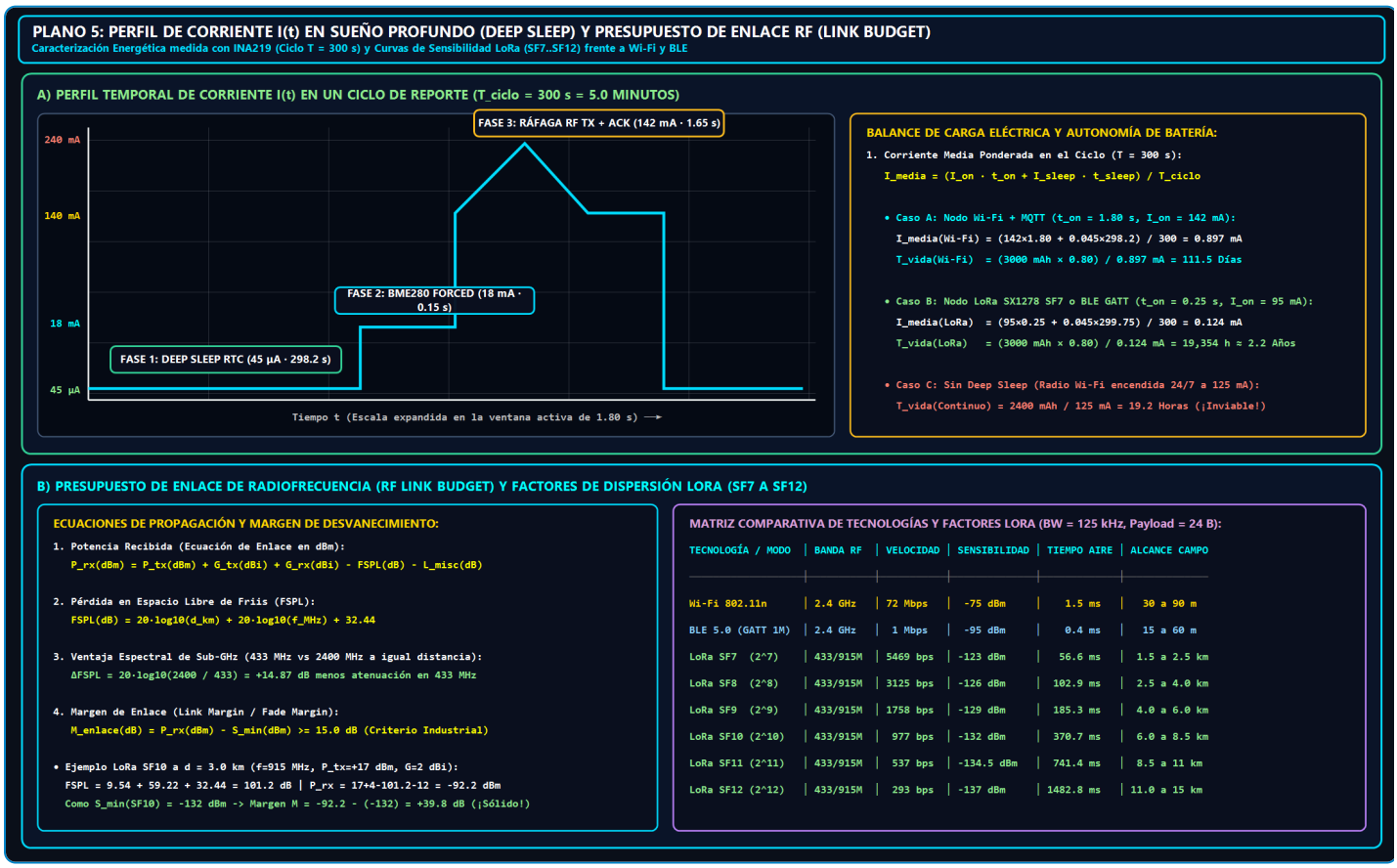
Una ventaja de $14.87\ \text{dB}$ equivale a multiplicar el alcance lineal en espacio libre por un factor de $5.54\times$ con la misma potencia de transmisión, además de sufrir menor absorción dieléctrica al atravesar follaje húmedo en invernaderos o muros de hormigón.

2.2 Margen de Desvanecimiento (Link Margin)

Para que un enlace industrial sea confiable ante lluvia, niebla o interferencia multitrayecto, exigimos un margen mínimo de seguridad de $15\ \text{dB}$ sobre la sensibilidad límite $S_{\min}$ del demodulador:

$$M_{\text{enlace}}\ (\text{dB}) = P_{\text{rx}}\ (\text{dBm}) - S_{\min}\ (\text{dBm}) \geq 15.0\ \text{dB}$$

📐 Plano 5: Perfil Temporal de Corriente en Deep Sleep y Presupuesto de Enlace RF



---

3. Reconexión Wi-Fi y MQTT con Backoff Exponencial y Jitter Estocástico

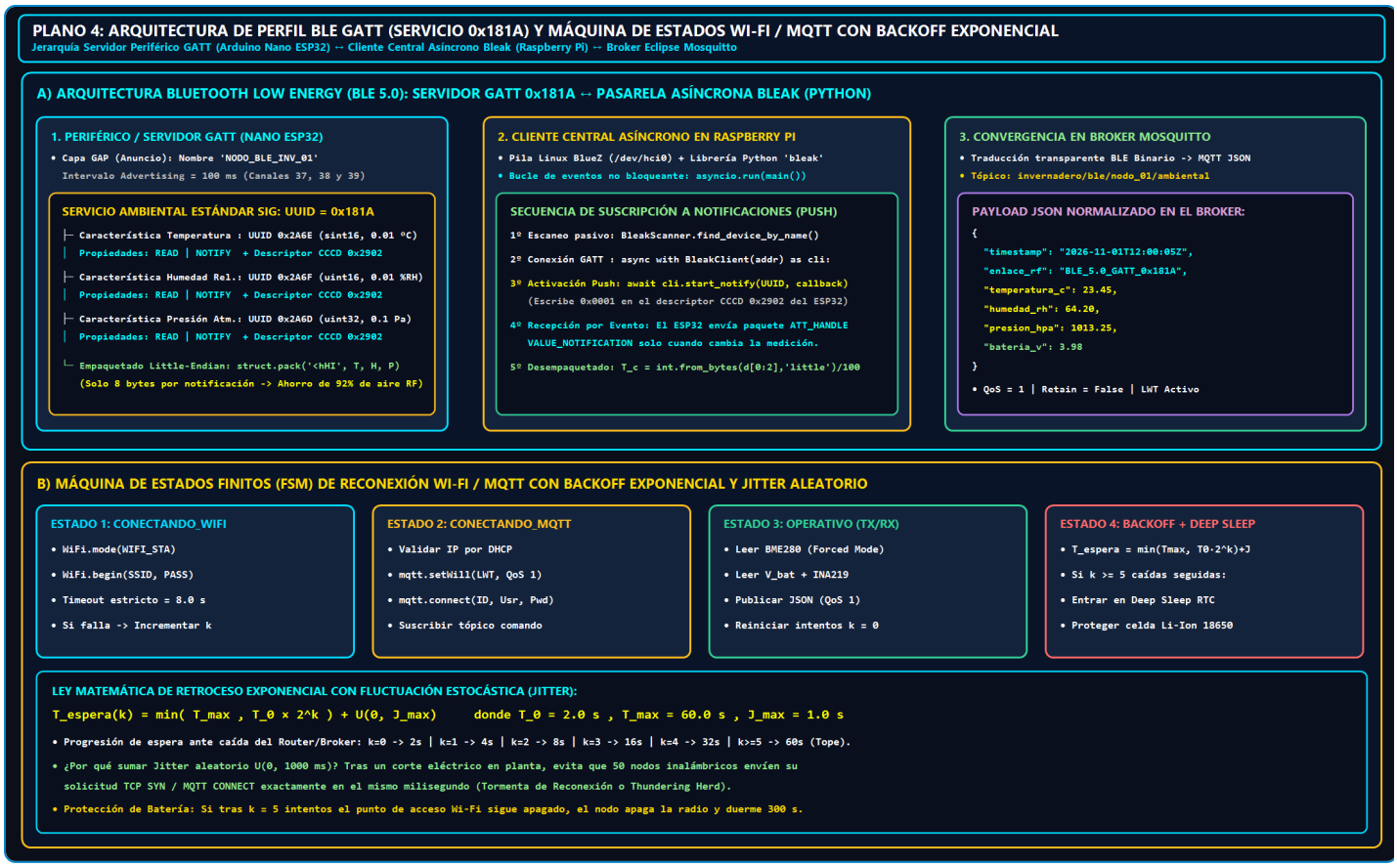
Cuando se restaura la energía eléctrica tras un apagón en planta, los microcontroladores arrancan en menos de $200\ \text{ms}$, mientras que el router Wi-Fi industrial y el servidor Linux tardan entre $30\ \text{s}$ y $60\ \text{s}$ en levantar sus servicios. Si N nodos reintentan conectarse en un bucle fijo cada $500\ \text{ms}$, provocan una *Tormenta de Reconexión (*Thundering Herd*)* y agotan sus baterías.

La solución de grado industrial implementa *Retroceso Exponencial Truncado con Fluctuación Aleatoria (*Jitter*)*:

$$T_{\text{espera}}(k) = \min\left(T_{\max},\ T_0 \cdot 2^k\right) + U(0, J_{\max})$$

Con $T_0 = 2.0\ \text{s}$, $T_{\max} = 60.0\ \text{s}$ y $J_{\max} = 1.0\ \text{s}$, la espera crece geométricamente ($2\ \text{s}, 4\ \text{s}, 8\ \text{s}, 16\ \text{s}, 32\ \text{s}, 60\ \text{s}$) y el término aleatorio uniforme $U(0,\ 1000\ \text{ms})$ desincroniza las ráfagas TCP SYN entre nodos.

📐 Plano 4: Arquitectura de Perfil BLE GATT 0x181A y Máquina de Estados de Reconexión con Backoff



---

4. Modelado Energético en Sueño Profundo (`Deep Sleep`) y Vida Útil de Batería

En un nodo cíclico con periodo total $T_{ciclo} = t_{activo} + t_{sleep}$, la **corriente media ponderada** drenada de la batería es:

$$I_{media} = \frac{I_{activo} \cdot t_{activo} + I_{sleep} \cdot t_{sleep}}{t_{activo} + t_{sleep}}$$

Aplicando un factor de descarga segura $\eta_{bat} = 0.80$ para preservar la química de la celda `Li-Ion 18650` ($Q_{bat} = 3000\ mAh$), la autonomía en días es:

$$T_{vida\ (días)} = \frac{Q_{bat\ (mAh)} \cdot \eta_{bat}}{24 \cdot I_{media\ (mA)}}$$

- **Sin Deep Sleep ($I_{media} = 125\ mA$ continuo):** $T_{vida} = (3000 \times 0.80) / (24 \times 125) = 0.80$ días $= 19.2$ horas.
- **Con Deep Sleep en Wi-Fi ($t_{activo} = 1.8$ s a $142\ mA$, $t_{sleep} = 298.2$ s a $45\ \mu A$):** $I_{media} = 0.897\ mA \implies T_{vida} = \mathbf{111.5\ días}$.
- **Con Deep Sleep en LoRa `SF7` o `BLE` ($t_{activo} = 0.25$ s a $95\ mA$, $t_{sleep} = 299.75$ s a $45\ \mu A$):** $I_{media} = 0.124\ mA \implies T_{vida} = \mathbf{806\ días\ (\approx 2.2\ años)}$.

PLANO Y ESQUEMÁTICO DEL CIRCUITO — MÓDULO 06: NODOS INALÁMBRICOS Y PLACAS PCB SOLDADAS

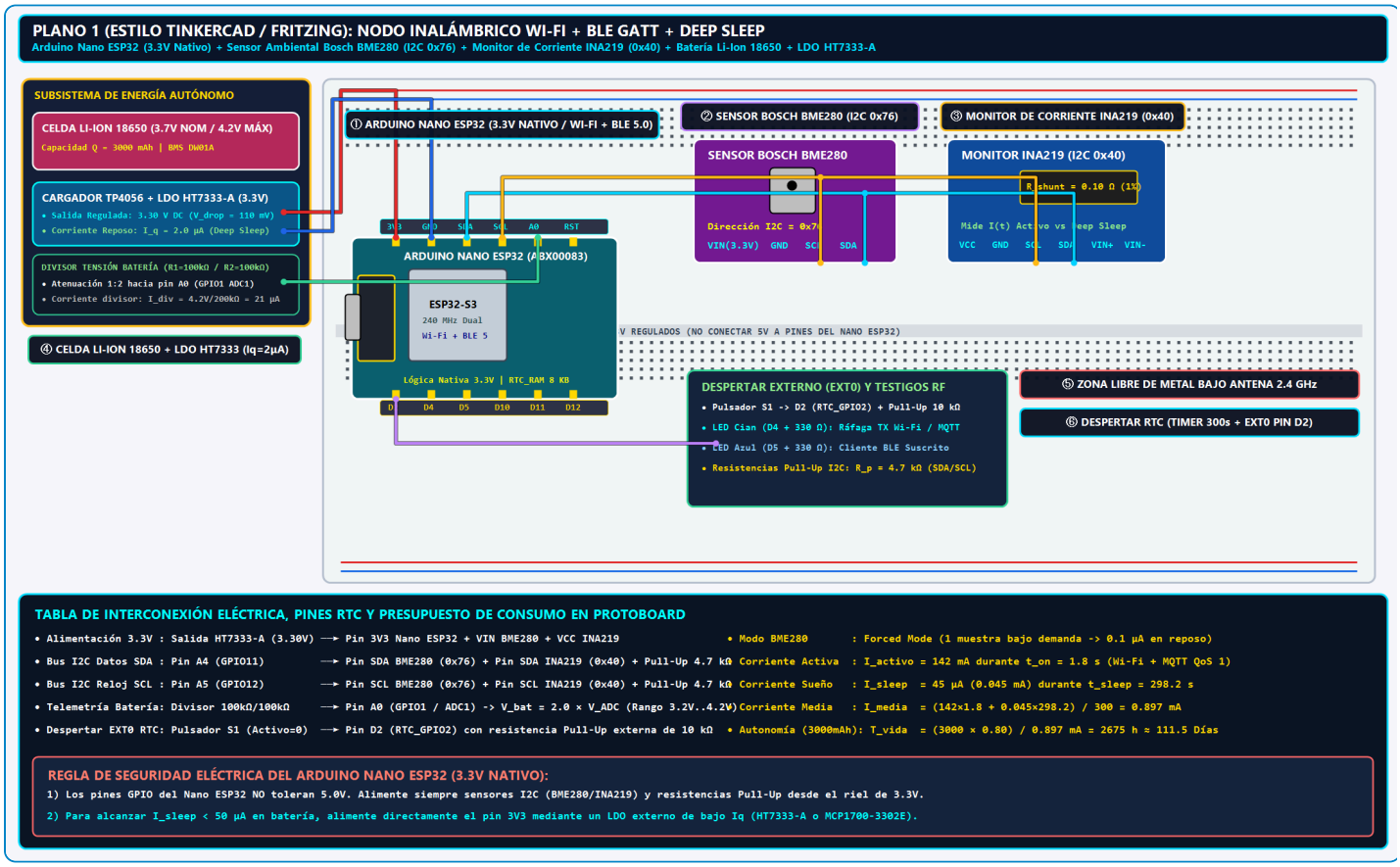
Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Planos de Montaje en Protoboard (Estilo Tinkercad / Fritzing) de Alta Legibilidad y Planos de Circuito Impreso (`PCB FR-4`) con Montaje por Soldadura

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Nodo Inalámbrico Wi-Fi + BLE + Deep Sleep en Protoboard

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad del Nodo Wi-Fi y BLE con Sensor BME280, Monitor INA219 y Batería Li-Ion 18650

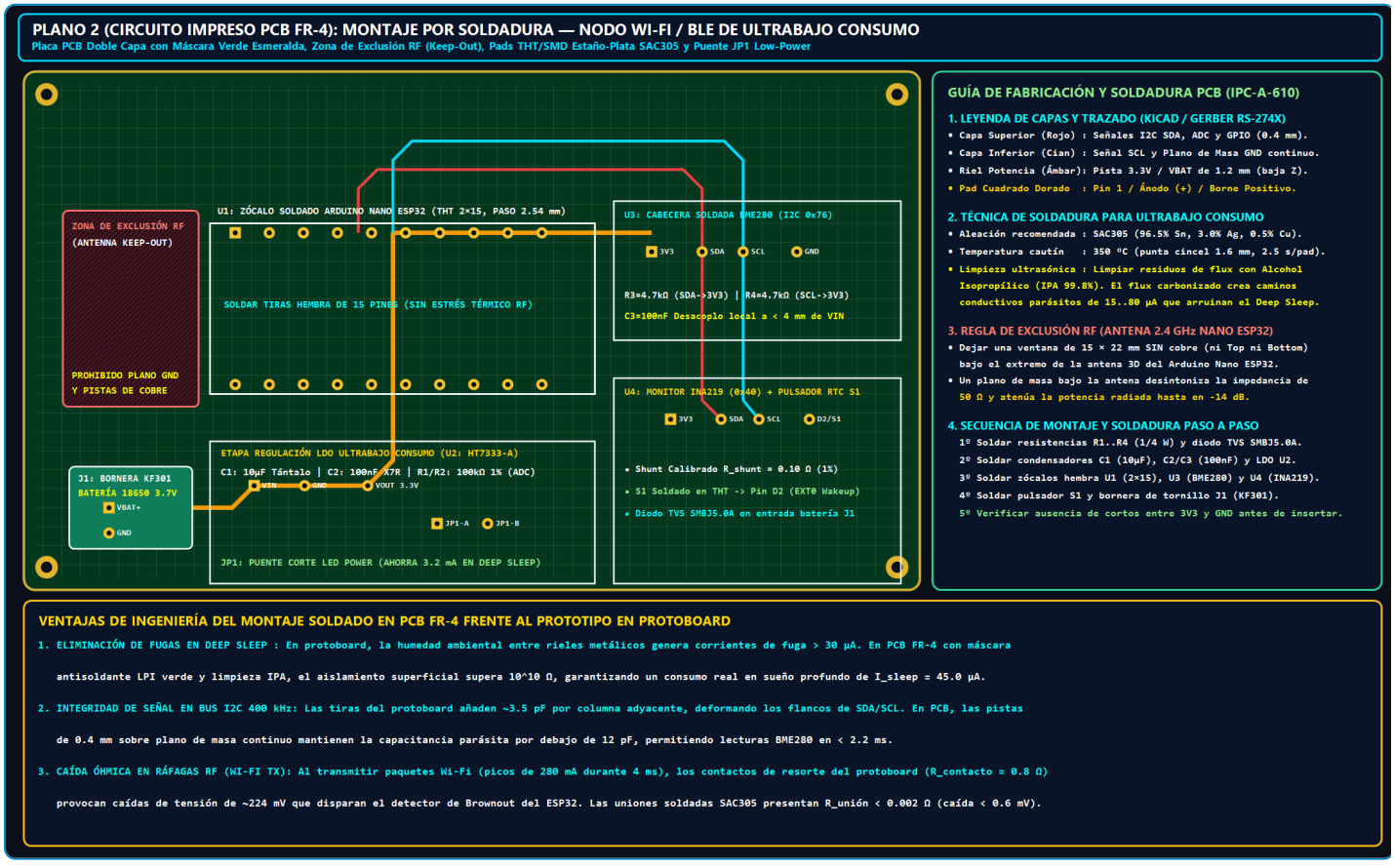


1.1 Resumen de Asignación de Pines del Arduino Nano ESP32 (`3.3 V` Nativo)

Callout en Plano	Subsistema / Módulo	Pines en Protoboard	Parámetros Eléctricos y de Firmware
①	**Arduino Nano ESP32 (** `ABX00083` **)**	Riel `3V3` y `GND`	SoC `ESP32-S3` con $8\ KB$ de memoria RTC (`RTC_DATA_ATTR`)
②	**Sensor Ambiental Bosch** `BME280`	`SDA -> A4` / `SCL -> A5`	Dirección I^2C `0x76` , operado en `MODE_FORCED` ($0.1\ \mu A$ en reposo)
③	**Monitor de Corriente** `INA219`	`SDA -> A4` / `SCL -> A5`	Dirección I^2C `0x40` , resistencia Shunt $R_{shunt} = 0.10\ \Omega\ (1\%)$
④	**Batería** `18650` **+ LDO** `HT7333-A`	Salida `3.30 V` → `Pin 3V3`	Corriente de reposo $I_q = 2.0\ \mu A$ y divisor $100\ k\Omega/100\ k\Omega$ hacia `A0`
⑤	**Antena Cerámica / 3D** `2.4 GHz`	Extremo izquierdo del módulo	Libre de cables puente y planos metálicos a $15\ mm$ a la redonda
⑥	**Despertar RTC (** `Timer` **+** `EXT0` **)**	Pulsador `S1` → `Pin D2`	`esp_sleep_enable_timer_wakeup(300e6)` + `esp_sleep_enable_ext0_wakeup(GPIO_NUM_2, 0)`

2. Plano 2 (Circuito Impreso `PCB FR-4`): Montaje por Soldadura del Nodo de Ultrabajo Consumo

📐 Plano 2: Placa de Circuito Impreso PCB FR-4 Doble Capa con Montaje por Soldadura del Nodo Inalámbrico

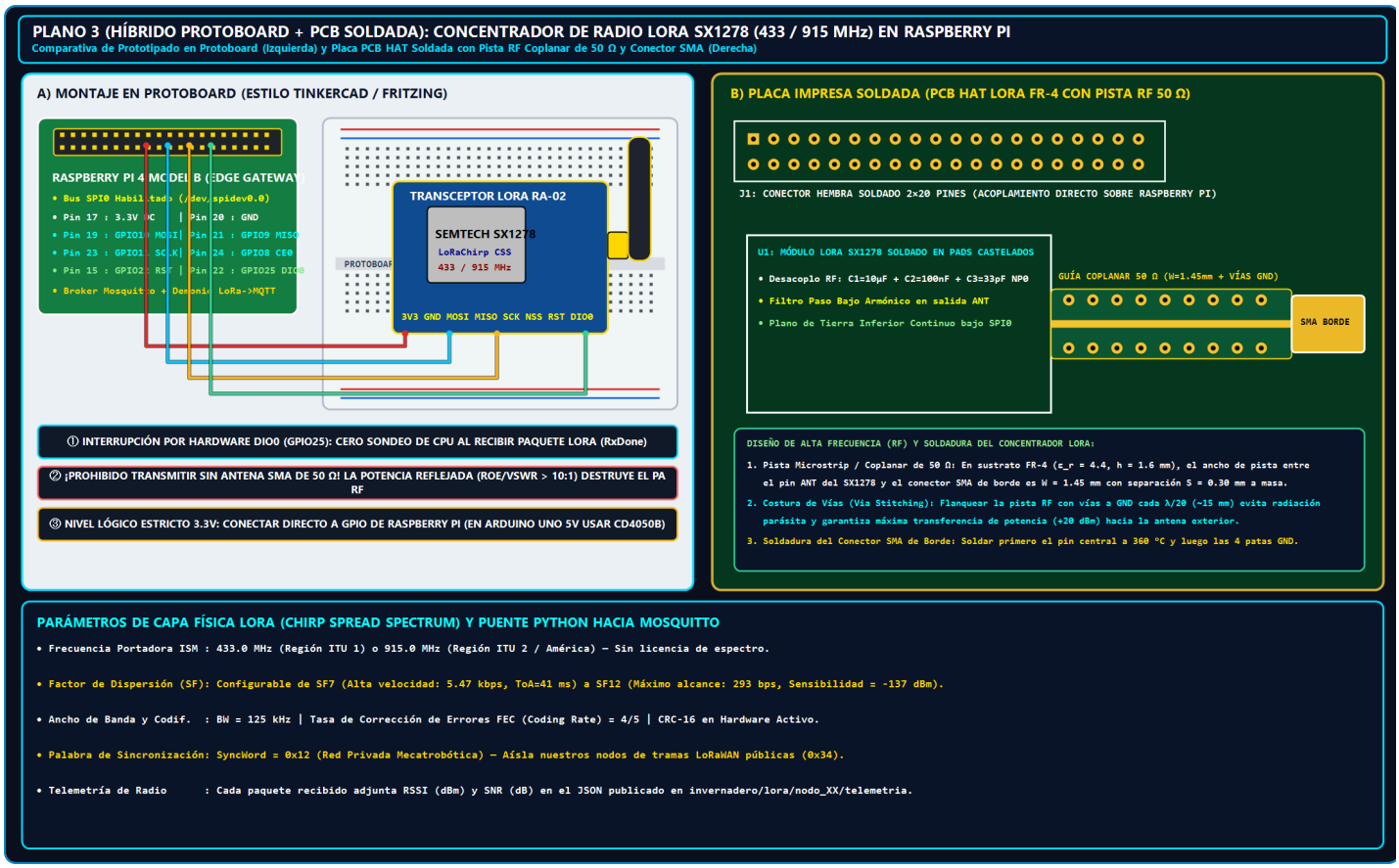


2.1 Especificaciones de Fabricación PCB (`KiCad` / `IPC-2221`) y Soldadura (`IPC-A-610`)

Elemento en la Placa PCB	Especificación Geométrica y Metalúrgica	Razón de Ingeniería (RF y Ultrabajo Consumo)
Zona de Exclusión RF (`Keep-Out`)	Ventana de 15×22 mm **sin cobre** en `Top` ni `Bottom` bajo la antena del Nano ESP32	Evita desintonizar la antena de 2.4 GHz (un plano GND debajo atenúa hasta -14 dB)
Puente Soldable `JP1` (Corte LED)	Dos pads dorados de 2.0×2.0 mm en serie con el LED indicador de encendido	Abrir `JP1` en despliegue final elimina el consumo fijo de 3.2 mA del LED Power
Limpieza de Flux con `IPA 99.8%`	Lavado con cepillo antiestático tras soldar con aleación `SAC305` a $350\ ^\circ$C	Los residuos de flux forman resistencias parásitas superficiales que drenan $15..80\ \mu$A
Zócalos Hembra `THT` (`U1, U3, U4`)	Agujeros metalizados $\varnothing 1.0$ mm con anillo anular dorado $\varnothing 1.8$ mm	Permite sustituir el Nano ESP32 o el `BME280` en campo en 5 segundos sin desoldar

3. Plano 3 (Híbrido Protoboard + `PCB HAT Soldada`): Concentrador `LoRa SX1278` en Raspberry Pi

📐 Plano 3: Montaje en Protoboard y Placa PCB HAT Soldada con Guía de Onda Coplanar de 50 Ohmios del Concentrador LoRa SX1278



4. Plano 4: Arquitectura de Perfil BLE GATT (`0x181A`) y Máquina de Estados con Backoff Exponencial

📐 Plano 4: Jerarquía del Servidor BLE GATT 0x181A y Máquina de Estados de Reconexión Wi-Fi/MQTT

PLANO 4: ARQUITECTURA DE PERFIL BLE GATT (SERVICIO 0x181A) Y MÁQUINA DE ESTADOS WI-FI / MQTT CON BACKOFF EXPONENCIAL

Jerarquía Servidor Periférico GATT (Arduino Nano ESP32) ↔ Cliente Central Asíncrono Bleak (Raspberry Pi) ↔ Broker Eclipse Mosquitto

A) ARQUITECTURA BLUETOOTH LOW ENERGY (BLE 5.0): SERVIDOR GATT 0x181A ↔ PASARELA ASÍNCRONA BLEAK (PYTHON)

1. PERIFÉRICO / SERVIDOR GATT (NANO ESP32)

- Capa GAP (Anuncio): Nombre 'NODO_BLE_INV_01'
  Intervalo Advertising = 100 ms (Canales 37, 38 y 39)

SERVICIO AMBIENTAL ESTÁNDAR SIG: UUID = 0x181A

- ├ Característica Temperatura : UUID 0x2A6E (sint16, 0.01 °C)
  | Propiedades: READ | NOTIFY + Descriptor CCCD 0x2902
- ├ Característica Humedad Rel.: UUID 0x2A6F (uint16, 0.01 %RH)
  | Propiedades: READ | NOTIFY + Descriptor CCCD 0x2902
- ├ Característica Presión Atm.: UUID 0x2A6D (uint32, 0.1 Pa)
  | Propiedades: READ | NOTIFY + Descriptor CCCD 0x2902
- └ Empaquetado Little-Endian: struct.pack('<hHI', T, H, P)
  (Solo 8 bytes por notificación -> Ahorro de 92% de aire RF)

2. CLIENTE CENTRAL ASÍNCRONO EN RASPBERRY PI

- Pila Linux BlueZ (/dev/hci0) + Librería Python 'bleak'
- Bucle de eventos no bloqueante: asyncio.run(main())

SECUENCIA DE SUSCRIPCIÓN A NOTIFICACIONES (PUSH)

1º Escaneo pasivo: BleakScanner.find_device_by_name()

2º Conexión GATT : async with BleakClient(addr) as cli:

3º Activación Push: await cli.start_notify(UUID, callback)
(Escribe 0x0001 en el descriptor CCCD 0x2902 del ESP32)

4º Recepción por Evento: El ESP32 envía paquete ATT_HANDLE VALUE_NOTIFICATION solo cuando cambia la medición.

5º Desempaquetado: T_c = int.from_bytes(d[0:2],'little')/100

3. CONVERGENCIA EN BROKER MOSQUITTO

- Traducción transparente BLE Binario -> MQTT JSON
- Tópico: invernadero/ble/nodo_01/ambiental

PAYLOAD JSON NORMALIZADO EN EL BROKER:

```
{
  "timestamp": "2026-11-01T12:00:05Z",
  "enlace_rf": "BLE_5.0_GATT_0x181A",
  "temperatura_c": 23.45,
  "humedad_rh": 64.20,
  "presion_hpa": 1013.25,
  "bateria_v": 3.98
}
```

- QoS = 1 | Retain = False | LWT Activo

B) MÁQUINA DE ESTADOS FINITOS (FSM) DE RECONEXIÓN WI-FI / MQTT CON BACKOFF EXPONENCIAL Y JITTER ALEATORIO

ESTADO 1: CONECTANDO_WIFI

- WiFi.mode(WIFI_STA)
- WiFi.begin(SSID, PASS)
- Timeout estricto = 8.0 s
- Si falla -> Incrementar k

ESTADO 2: CONECTANDO_MQTT

- Validar IP por DHCP
- mqtt.setWill(LWT, QoS 1)
- mqtt.connect(ID, Usr, Pwd)
- Suscribir tópico comando

ESTADO 3: OPERATIVO (TX/RX)

- Leer BME280 (Forced Mode)
- Leer V_bat + INA219
- Publicar JSON (QoS 1)
- Reiniciar intentos k = 0

ESTADO 4: BACKOFF + DEEP SLEEP

- T_espera = min(Tmax, T0·2^k)+J
- Si k >= 5 caídas seguidas:
- Entrar en Deep Sleep RTC
- Proteger celda Li-Ion 18650

LEY MATEMÁTICA DE RETROCESO EXPONENCIAL CON FLUCTUACIÓN ESTOCÁSTICA (JITTER):

$T_{espera}(k) = \min(T_{max} , T_0 \times 2^k) + U(0, J_{max})$ donde $T_0 = 2.0$ s , $T_{max} = 60.0$ s , $J_{max} = 1.0$ s

- Progresión de espera ante caída del Router/Broker: k=0 -> 2s | k=1 -> 4s | k=2 -> 8s | k=3 -> 16s | k=4 -> 32s | k>=5 -> 60s (Tope).
- ¿Por qué sumar Jitter aleatorio U(0, 1000 ms)? Tras un corte eléctrico en planta, evita que 50 nodos inalámbricos envíen su solicitud TCP SYN / MQTT CONNECT exactamente en el mismo milisegundo (Tormenta de Reconexión o Thundering Herd).
- Protección de Batería: Si tras k = 5 intentos el punto de acceso Wi-Fi sigue apagado, el nodo apaga la radio y duerme 300 s.

---

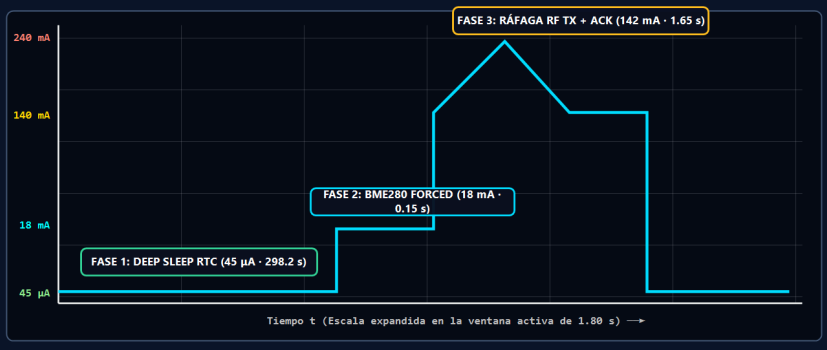
5. Plano 5: Perfil de Corriente $I(t)$ en `Deep Sleep` y Presupuesto de Enlace RF (`SF7..SF12`)

📐 Plano 5: Perfil Temporal de Corriente medido con INA219 y Presupuesto de Enlace RF

PLANO 5: PERFIL DE CORRIENTE I(t) EN SUEÑO PROFUNDO (DEEP SLEEP) Y PRESUPUESTO DE ENLACE RF (LINK BUDGET)

Caracterización Energética medida con INA219 (Ciclo T = 300 s) y Curvas de Sensibilidad LoRa (SF7..SF12) frente a Wi-Fi y BLE

A) PERFIL TEMPORAL DE CORRIENTE I(t) EN UN CICLO DE REPORTE (T_ciclo = 300 s = 5.0 MINUTOS)



BALANCE DE CARGA ELÉCTRICA Y AUTONOMÍA DE BATERÍA:

1. Corriente Media Ponderada en el Ciclo (T = 300 s):
   $I_{media} = (I_{on} \cdot t_{on} + I_{sleep} \cdot t_{sleep}) / T_{ciclo}$

- Caso A: Nodo Wi-Fi + MQTT (t_{on} = 1.80 s, I_{on} = 142 mA):
  $I_{media(Wi-Fi)} = (142 \times 1.80 + 0.045 \times 298.2) / 300 = 0.897$ mA
  $T_{vida(Wi-Fi)} = (3000$ mAh × 0.80) / 0.897 mA = 111.5 Días
- Caso B: Nodo LoRa SX1278 SF7 o BLE GATT (t_{on} = 0.25 s, I_{on} = 95 mA):
  $I_{media(LoRa)} = (95 \times 0.25 + 0.045 \times 299.75) / 300 = 0.124$ mA
  $T_{vida(LoRa)} = (3000$ mAh × 0.80) / 0.124 mA = 19,354 h ≈ 2.2 Años
- Caso C: Sin Deep Sleep (Radio Wi-Fi encendida 24/7 a 125 mA):
  $T_{vida(Continuo)}$ = 2400 mAh / 125 mA = 19.2 Horas (¡Inviable!)

B) PRESUPUESTO DE ENLACE DE RADIOFRECUENCIA (RF LINK BUDGET) Y FACTORES DE DISPERSIÓN LORA (SF7 A SF12)

ECUACIONES DE PROPAGACIÓN Y MARGEN DE DESVANECIMIENTO:

1. Potencia Recibida (Ecuación de Enlace en dBm):
   $P_{rx(dBm)} = P_{tx(dBm)} + G_{tx(dBi)} + G_{rx(dBi)} - FSPL(dB) - L_{misc(dB)}$

2. Pérdida en Espacio Libre de Friis (FSPL):
   $FSPL(dB) = 20 \cdot \log10(d_{km}) + 20 \cdot \log10(f_{MHz}) + 32.44$

3. Ventaja Espectral de Sub-GHz (433 MHz vs 2400 MHz a igual distancia):
   $\Delta FSPL = 20 \cdot \log10(2400 / 433) = +14.87$ dB menos atenuación en 433 MHz

4. Margen de Enlace (Link Margin / Fade Margin):
   $M_{enlace(dB)} = P_{rx(dBm)} - S_{min(dBm)} >= 15.0$ dB (Criterio Industrial)

- Ejemplo LoRa SF10 a d = 3.0 km (f=915 MHz, P_{tx}=+17 dBm, G=2 dBi):
  FSPL = 9.54 + 59.22 + 32.44 = 101.2 dB | P_{rx} = 17+4-101.2-12 = -92.2 dBm
  Como $S_{min(SF10)}$ = -132 dBm -> Margen M = -92.2 - (-132) = +39.8 dB (¡Sólido!)

MATRIZ COMPARATIVA DE TECNOLOGÍAS Y FACTORES LORA (BW = 125 kHz, Payload = 24 B):

TECNOLOGÍA / MODO	BANDA RF	VELOCIDAD	SENSIBILIDAD	TIEMPO AIRE	ALCANCE CAMPO
Wi-Fi 802.11n	2.4 GHz	72 Mbps	-75 dBm	1.5 ms	30 a 90 m
BLE 5.0 (GATT 1M)	2.4 GHz	1 Mbps	-95 dBm	0.4 ms	15 a 60 m
LoRa SF7 (2^7)	433/915M	5469 bps	-123 dBm	56.6 ms	1.5 a 2.5 km
LoRa SF8 (2^8)	433/915M	3125 bps	-126 dBm	102.9 ms	2.5 a 4.0 km
LoRa SF9 (2^9)	433/915M	1758 bps	-129 dBm	185.3 ms	4.0 a 6.0 km
LoRa SF10 (2^10)	433/915M	977 bps	-132 dBm	370.7 ms	6.0 a 8.5 km
LoRa SF11 (2^11)	433/915M	537 bps	-134.5 dBm	741.4 ms	8.5 a 11 km
LoRa SF12 (2^12)	433/915M	293 bps	-137 dBm	1482.8 ms	11.0 a 15 km

Capítulo 21 · Radio LoRa SX1278, presupuesto de enlace y Deep Sleep

Conceptos clave

Verifica banda, potencia y antena para la región. Calcula el margen de enlace y mide el consumo durante transmisión y reposo.



GUÍA PASO A PASO DE LABORATORIO 06: NODOS INALÁMBRICOS E INTEGRACIÓN HÍBRIDA (WI-FI, BLE, LORA Y DEEP SLEEP)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Nodos Inalámbricos Autónomos, Servidor BLE GATT (`0x181A`), Radio de Largo Alcance LoRa (`SX1278`) y Sueño Profundo (`Deep Sleep`) con Sensor Ambiental Bosch `BME280`

Plataformas: Arduino Nano ESP32 (`3.3 V` Nativo) + Raspberry Pi 4 Model B / 5 (`64-bit`)

---

1. Objetivo de Ingeniería del Laboratorio

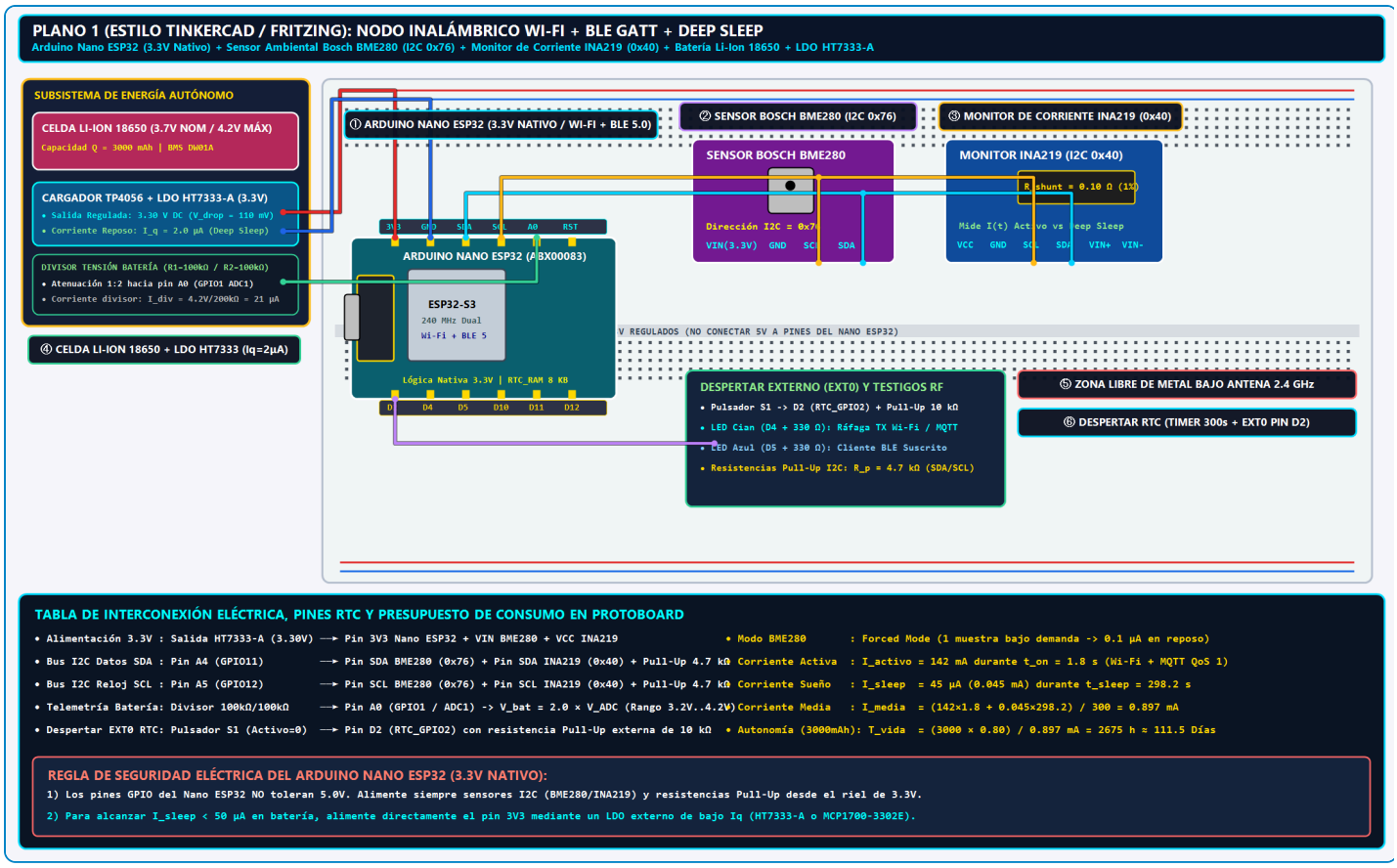
Diseñar, ensamblar en protoboard (estilo Tinkercad / Fritzing) y en placa de circuito impreso soldada (`PCB FR-4`), programar y auditar una **arquitectura inalámbrica híbrida de tres bandas/protocolos (`Wi-Fi` , `BLE 5.0` y `LoRa Sub-GHz`)** convergente hacia el Broker Eclipse Mosquitto en la Raspberry Pi:

- **Nodo Wi-Fi + MQTT de Ultrabajo Consumo (`Deep Sleep`)**: Basado en Arduino Nano ESP32 alimentado por celda `Li-Ion 18650 (3.7 V / 3000 mAh)` y regulador LDO `HT7333-A (I_q = 2.0 µA)` , leyendo temperatura, humedad y presión barométrica mediante el sensor Bosch `BME280` en **Modo Forzado (`Forced Mode`)** y reconectando con **Backoff Exponencial + Jitter**.
- **Nodo Periférico Bluetooth Low Energy (`BLE GATT 0x181A`)**: Exponiendo las características estándar `0x2A6E` (Temperatura), `0x2A6F` (Humedad) y `0x2A6D` (Presión) con notificaciones asíncronas (`CCCD 0x2902`) hacia el demonio Python `bleak` en la Raspberry Pi.
- **Concentrador de Largo Alcance** `LoRa SX1278 (433 / 915 MHz)` : Acoplado por bus `SPI0` e interrupción física `DIO0 (GPIO 25)` a la Raspberry Pi para enlaces rurales de `2 km` a `15 km` .

---

2. Montaje Físico en Protoboard (Estilo Tinkercad) y en Placa PCB Soldada

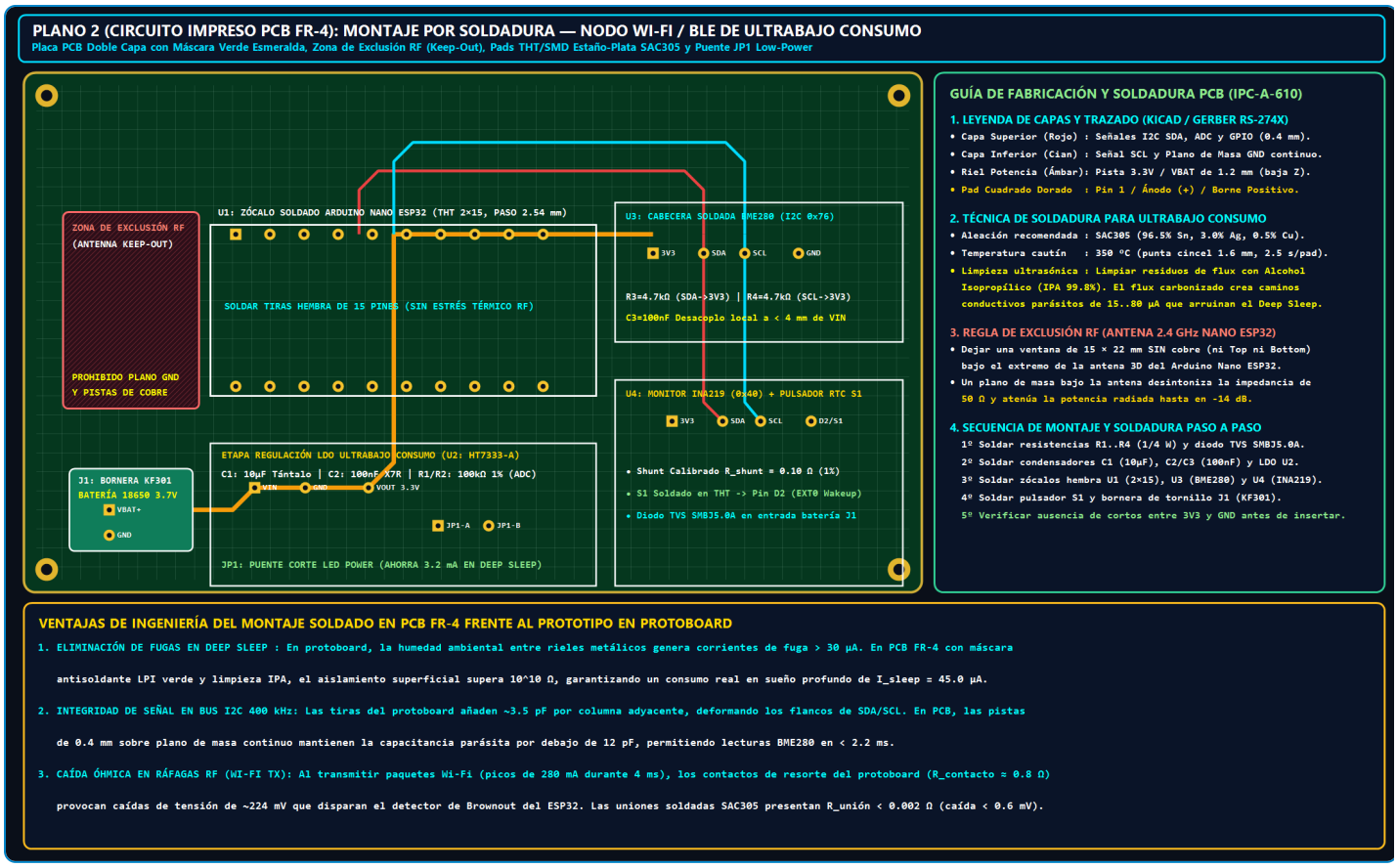
📐 **Plano 1: Montaje en Protoboard Estilo Tinkercad del Nodo Wi-Fi / BLE con BME280, INA219 y Batería 18650**



2.1 Tabla Maestra de Conexiones del Nodo Arduino Nano ESP32 (`Lógica Estricta 3.3 V`)

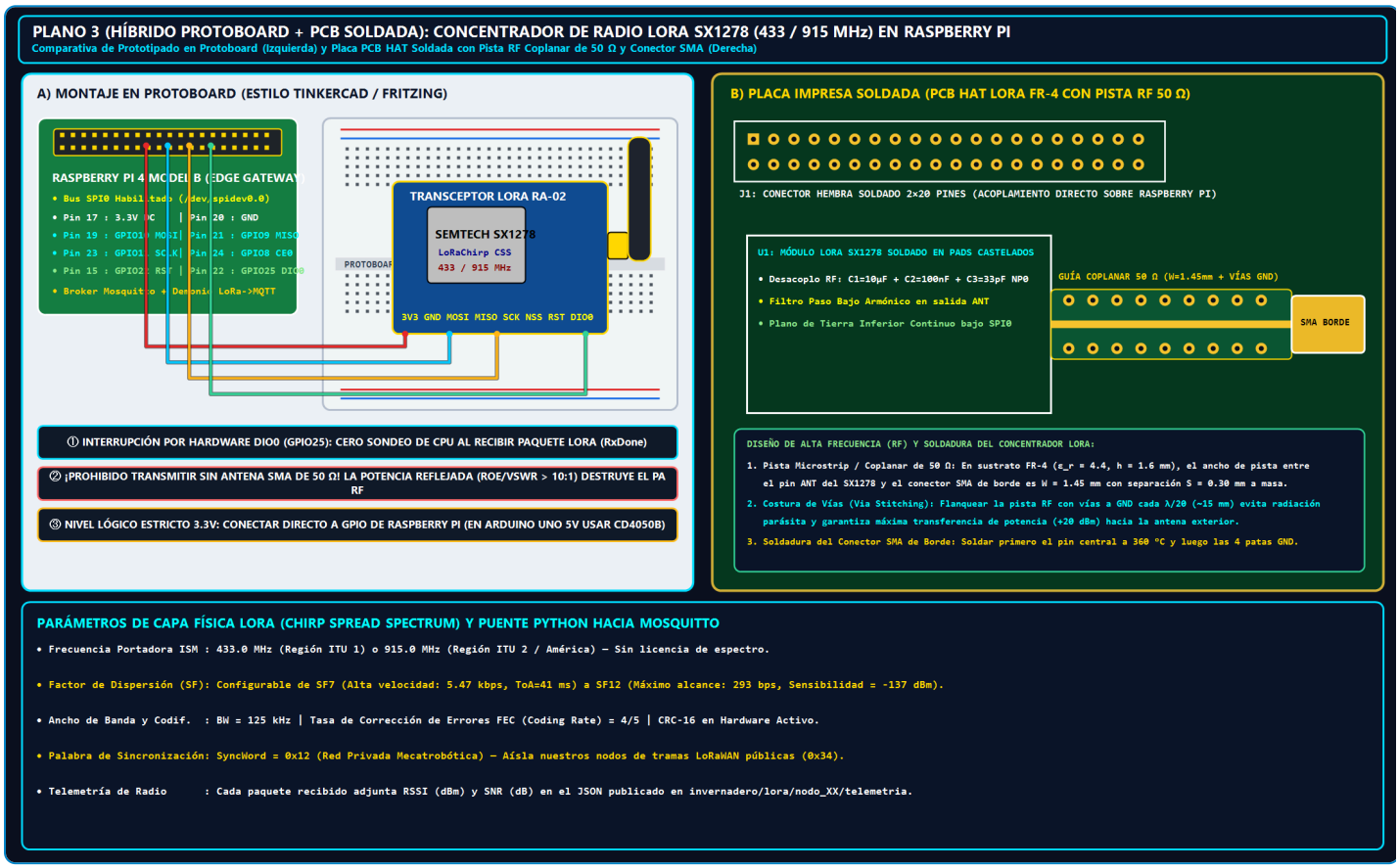
Subsistema / Componente	Pin del Componente	Pin en Arduino Nano ESP32	Función Eléctrica y Observaciones de Ingeniería
Regulador LDO HT7333-A	`VOUT (3.30 V)`	`Pin 3V3`	Alimentación directa regulada con corriente de reposo $I_q = 2.0\ \mu A$
Sensor Bosch BME280 (`0x76`)	`VIN / GND / SDA / SCL`	`3V3 / GND / A4 / A5`	Bus I^2C a $400\ kHz$ con resistencias Pull-Up de $4.7\ k\Omega$ a $3.3\ V$
Monitor Corriente INA219 (`0x40`)	`VCC / GND / SDA / SCL`	`3V3 / GND / A4 / A5`	Shunt de precisión $R_{shunt} = 0.10\ \Omega\ (1\%)$ en serie con el riel de $3.3\ V$
Divisor Tensión Batería	Punto medio `100kΩ/100kΩ`	`Pin A0 (GPIO1 / ADC1)`	Lectura calibrada con `analogReadMilliVolts(A0)` ($V_{bat} = 2.0 \times V_{ADC}$)
Pulsador Despertar RTC (`S1`)	Contacto NA a `GND`	`Pin D2 (RTC_GPIO2)`	Interrupción externa `EXT0` con resistencia Pull-Up externa de $10\ k\Omega$ a $3.3\ V$
Indicadores LED RF	Ánodo vía $330\ \Omega$	`Pin D4 (Wi-Fi)` / `D5 (BLE)`	Testigos luminosos activos únicamente durante la ventana despierta

📐 Plano 2: Placa de Circuito Impreso PCB FR-4 con Montaje por Soldadura del Nodo Inalámbrico



3. Montaje del Concentrador LoRa `SX1278` en Raspberry Pi (`SPI0` + `IRQ DIO0`)

📐 Plano 3: Interconexión en Protoboard y Placa PCB HAT Soldada del Transceptor LoRa SX1278



> **Regla Crítica de Radiofrecuencia (ROE / VSWR):** Jamás alimente ni ejecute el script de transmisión/recepción del módulo `SX1278` sin haber enroscado previamente su **antena helicoidal o dipolo de $50\ \Omega$ en el conector SMA**. Transmitir a $+20\ \text{dBm}$ con el puerto RF abierto refleja el 100% de la energía hacia el transistor de salida del amplificador de potencia (PA), destruyendo el chip en milisegundos.

Pin del Módulo LoRa `SX1278 (Ra-02)`	Pin Físico en Raspberry Pi (40 Pines)	GPIO BCM	Función en el Bus `SPI0` y Control de Radio
`VCC (3.3V)`	**Pin 17**	Riel `3.3V DC`	Alimentación limpia capaz de entregar picos de $120\ \text{mA}$
`GND`	**Pin 20**	Riel `GND`	Masa común de referencia digital y blindaje RF
`MOSI`	**Pin 19**	`GPIO 10 (SPI0_MOSI)`	Escritura de registros y carga de FIFO hacia el SX1278
`MISO`	**Pin 21**	`GPIO 9 (SPI0_MISO)`	Lectura de registros, RSSI, SNR y paquete recibido
`SCK`	**Pin 23**	`GPIO 11 (SPI0_SCLK)`	Reloj síncrono SPI a $4.0\ \text{MHz}$ (Modo `0,0`)
`NSS (CS)`	**Pin 24**	`GPIO 8 (SPI0_CE0_N)`	Selección de esclavo activa en nivel bajo
`RST`	**Pin 15**	`GPIO 22`	Reinicio físico por hardware del transceptor
`DIO0`	**Pin 22**	`GPIO 25`	Interrupción por flanco de subida (`RxDone` / `TxDone`)

---

4. Despliegue de Software y Verificación en Laboratorio

Paso 1: Habilitar `SPI0` y `Bluetooth BlueZ` en Raspberry Pi e instalar dependencias

```
sudo raspi-config nonint do_spi 0
sudo systemctl enable --now bluetooth
sudo apt update && sudo apt install -y python3-pip python3-spidev python3-rpi.gpio bluez
pip3 install --break-system-packages bleak paho-mqtt
```

Paso 2: Compilar y cargar los firmwares en Arduino Nano ESP32

- En Arduino IDE 2.x, instalar el paquete de placas **Arduino ESP32 Boards** y las librerías `PubSubClient`, `Adafruit BME280 Library` y `Adafruit INA219`.
- Para ensayar el nodo **Wi-Fi + MQTT con Deep Sleep**, abrir `codigo_fuente/nodo_wifi_mqtt_bme280_deepsleep.ino`, ajustar `WIFI_SSID`, `WIFI_PASSWORD` y `MQTT_BROKER_IP`, y subir al Nano ESP32.
- Para ensayar el nodo **Periférico BLE GATT (`0x181A`)**, subir `codigo_fuente/periferico_ble_gatt_ambiental.ino`.

Paso 3: Lanzar las pasarelas de borde en la Raspberry Pi y auditar en Mosquitto

```
# Terminal 1: Iniciar pasarela asíncrona BLE GATT -> MQTT
python3 codigo_fuente/pasarela_ble_a_mqtt_bleak.py

# Terminal 2: Iniciar concentrador LoRa SX1278 SPI0 -> MQTT
python3 codigo_fuente/transceptor_lora_sx1278_mqtt_bridge.py

# Terminal 3: Auditar en tiempo real la convergencia de las 3 tecnologías RF en el Broker
mosquitto_sub -h localhost -u scada_central -P Clave -t 'invernadero/#' -q 1 -v
```

LISTADO DE MATERIALES (BOM) — MÓDULO 06: NODOS INALÁMBRICOS E INTEGRACIÓN HÍBRIDA

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Nodos Inalámbricos (`Wi-Fi` , `BLE 5.0` , `LoRa SX1278`), Gestión Energética `Li-Ion 18650` y Montaje en Protoboard / PCB Soldada

---

1. Microcontroladores, Transceptores RF y Computadora de Borde

Ítem	Cant.	Designación PCB	Componente / Referencia Industrial	Especificación Técnica Clave	Función en el Módulo 06
01	2	`U1`	**Arduino Nano ESP32** (`ABX00083`)	SoC `ESP32-S3` Dual-Core Xtensa LX7 @ $240\ \text{MHz}$, $8\ \text{KB}$ RTC SRAM, Lógica nativa $3.3\ \text{V}$, Wi-Fi `802.11b/g/n` + `BLE 5.0`	Nodo 1: Wi-Fi + Deep Sleep; Nodo 2: Servidor Periférico BLE GATT (`0x181A`)
02	1	`GW1`	**Raspberry Pi 4 Model B (4 GB) o RPi 5**	Broadcom Quad-Core 64-bit, Bluetooth `BlueZ 5.x` (`/dev/hci0`), Bus `SPI0` (`/dev/spidev0.0`), Broker Mosquitto	Pasarela de Borde Híbrida (`BLE` + `LoRa` + `Wi-Fi` → `MQTT`)
03	2	`U_LORA`	**Módulo Transceptor LoRa** `Ra-02 (SX1278)`	Banda ISM Sub-GHz ($433\ \text{MHz}$ / $915\ \text{MHz}$), Modulación CSS (`SF7..SF12`), Sensibilidad $-137\ \text{dBm}$, SPI $3.3\ \text{V}$	Enlace de largo alcance rural ($2\ \text{km}$ a $15\ \text{km}$) con CRC-16 en hardware
04	2	`ANT1`	**Antena Dipolo / Helicoidal SMA** $50\ \Omega$	Impedancia característica $Z_0 = 50\ \Omega$, Ganancia $G = +3.0\ \text{dBi}$, ROE (VSWR) $< 1.5:1$, Cable pigtail IPEX a SMA	Radiación eficiente y protección del amplificador de potencia del `SX1278`

2. Sensórica Ambiental I^2C e Instrumentación de Consumo Energético

Ítem	Cant.	Designación PCB	Componente / Referencia Industrial	Especificación Técnica Clave	Función en el Módulo 06
05	2	`U3`	**Sensor Ambiental Bosch** `BME280`	Interfaz I^2C (`0x76`) / SPI $3.3\ \text{V}$, Temp ($\pm 0.5\ ^\circ\text{C}$), Humedad ($\pm 3\%\ \text{RH}$), Presión ($\pm 1.0\ \text{hPa}$), $I_{sleep} = 0.1\ \mu\text{A}$	Adquisición meteorológica en Modo Forzado (`Forced Mode`) bajo demanda
06	1	`U4`	**Monitor High-Side Texas Instruments** `INA219`	Interfaz I^2C (`0x40`), ADC de $12\ \text{bits}$, resistencia Shunt $R_{shunt} = 0.10\ \Omega$ (1%), rango $\pm 3.2\ \text{A}$	Medición de perfil de corriente $I(t)$ en ráfaga RF frente a sueño profundo

3. Subsistema de Alimentación por Batería de Ultrabajo Consumo (`Deep Sleep`)

Ítem	Cant.	Designación PCB	Componente / Referencia Industrial	Especificación Técnica Clave	Función en el Módulo 06
07	1	`BAT1`	**Celda Recargable Li-Ion** `18650`	Tensión nominal $3.7\ V$ (carga plena $4.20\ V$, corte $3.0\ V$), Capacidad nominal $Q = 3000\ mAh$, química INR/NCR	Fuente primaria de energía para operación autónoma $> 110\ días$
08	1	`CHG1`	**Módulo Cargador Li-Ion** `TP4056` **con BMS**	Corriente de carga CC/CV hasta $1.0\ A$ más integrado de protección `DW01A` + doble MOSFET `FS8205A`	Carga segura por USB-C / panel solar de $5\ V$ y corte por sobredescarga
09	2	`U2`	**Regulador LDO** `HT7333-A` **o** `MCP1700-3302E`	Salida fija $3.30\ V$ ($I_{\max} = 250\ mA$), tensión de caída $V_{drop} = 110\ mV$, **corriente de reposo** $I_q = 2.0\ \mu A$	Sustituye reguladores `AMS1117` ($I_q > 5\ mA$) para lograr $I_{sleep} = 45\ \mu A$
10	2	`R1, R2`	**Resistencias de Película Metálica** $100\ k\Omega$	Tolerancia $\pm 1\%$, potencia $1/4\ W$, bajo coeficiente térmico ($\pm 50\ ppm/^\circ C$)	Divisor resistivo $1:2$ hacia el pin `A0 (ADC1)` para telemetría de batería
11	2	`C1, C2`	**Condensadores de Desacoplo** $10\ \mu F$ **y** $100\ nF$	$10\ \mu F$ Tántalo ($16\ V$, baja ESR) + $100\ nF$ Cerámico multicapa `X7R`	Absorción de picos transitorios de corriente ($240\ mA$) al transmitir en Wi-Fi

4. Insumos de Prototipado en Protoboard (Tinkercad) y Soldadura en Placa PCB FR-4

Ítem	Cant.	Designación PCB	Componente / Herramienta de Montaje	Especificación Técnica Clave	Función en el Módulo 06
12	2	`PROTO`	**Protoboard de 830 Puntos (** `MB-102` **)**	Contactos de bronce fosforoso niquelado, doble riel de alimentación lateral	Ensamble rápido sin soldadura (estilo Tinkercad / Fritzing)
13	2	`PCB1`	**Placa Virgen o Fabricada** `PCB FR-4` **Doble Capa**	Fibra de vidrio epoxi `FR-4` ($h = 1.6\ mm$, cobre $1\ oz/ft^2 = 35\ \mu m$), máscara antisoldante verde esmeralda	Montaje definitivo por soldadura con zona de exclusión RF y pista de $50\ \Omega$
14	4	`U1, U3, U4`	**Tiras de Pines Hembra Paso** $2.54\ mm$ **(** `THT` **)**	Contactos dorados de $15\ pines$ (para Nano ESP32) y $4/6\ pines$ (para `BME280` / `INA219`)	Permite insertar y retirar los módulos sobre la PCB sin aplicar calor directo
15	2	`J1, J2`	**Borneras de Tornillo Soldables** `KF301-2P`	Paso $5.08\ mm$, capacidad $300\ V\ /\ 10\ A$, fijación mecánica por jaula de latón	Fijación firme de los cables de la celda `18650` sobre la placa PCB
16	1	`SOLD`	**Soldadura** `SAC305` **+ Alcohol Isopropílico** `IPA`	Aleación sin plomo ($96.5\% Sn, 3.0\% Ag, 0.5\% Cu$) + `IPA 99.8%` para limpieza de flux	Eliminación de corrientes de fuga superficiales entre pads durante el Deep Sleep

MANUAL DE INGENIERÍA DE RADIOFRECUENCIA: WI-FI, BLE GATT, LORA Y SUEÑO PROFUNDO (DEEP SLEEP)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Fundamentos Matemáticos de Presupuesto de Enlace RF, Arquitectura GATT Bluetooth SIG, Modulación Chirp Spread Spectrum (`LoRa`) y Modelado Energético de Baterías

---

1. El Trilema de las Comunicaciones Inalámbricas en IoT

En ingeniería de telecomunicaciones industriales, ningún protocolo de radiofrecuencia puede maximizar simultáneamente **tres variables físicas contrapuestas**:

- **Alcance Geográfico (d)**
- **Caudal de Datos o Ancho de Banda (R_b)**
- **Autonomía Energética en Batería (T_{vida})**

Parámetro de Ingeniería	Wi-Fi (`IEEE 802.11n`)	Bluetooth Low Energy (`BLE 5.0`)	Radio `LoRa Sub-GHz (SX1278)`
Banda de Frecuencia ISM	$2.4\ GHz$ / $5.0\ GHz$	$2.4\ GHz$ (40 canales de $2\ MHz$)	$433\ MHz$ / $868\ MHz$ / $915\ MHz$
Velocidad Efectiva (R_b)	$20\ Mbps$ a $72\ Mbps$	$125\ kbps$ a $2.0\ Mbps$	$293\ bps$ (`SF12`) a $5.47\ kbps$ (`SF7`)
Alcance Típico en Campo	$30\ m$ a $90\ m$	$15\ m$ a $60\ m$	$2.0\ km$ **a** $15.0\ km$
Sensibilidad del Receptor ($S_{\min}$)	$-75\ dBm$	$-95\ dBm$	$-123\ dBm$ **(** `SF7` **)** **a** $-137\ dBm$ **(** `SF12` **)**
Corriente Pico en Transmisión	$180\ mA$ a $260\ mA$	$12\ mA$ a $18\ mA$	$90\ mA$ a $120\ mA$ ($+17..20\ dBm$)
Pila TCP/IP Nativa	Sí (Directo a Broker MQTT)	No (Requiere Pasarela GATT → IP)	No (Requiere Concentrador SPI → IP)

---

2. Presupuesto de Enlace RF (Link Budget) y Ecuación de Friis

La potencia electromagnética recibida P_{rx} (expresada en dBm, decibelios referidos a $1\ mW$) en la antena del servidor de borde viene determinada por el **balance logarítmico de potencias, ganancias y pérdidas**:

$$P_{rx}\ (dBm) = P_{tx}\ (dBm) + G_{tx}\ (dBi) + G_{rx}\ (dBi) - FSPL(d, f)\ (dB) - L_{misc}\ (dB)$$

Donde la **Pérdida de Propagación en el Espacio Libre de Friis ($FSPL$)** en función de la distancia $d\ (km)$ y la frecuencia portadora $f\ (MHz)$ es:

$$FSPL\ (dB) = 20 \cdot \log_{10}(d_{km}) + 20 \cdot \log_{10}(f_{MHz}) + 32.44$$

2.1 ¿Por qué la banda Sub-GHz ($433\ MHz$) penetra mucho mejor que $2.4\ GHz$?

A igual distancia d, la diferencia de atenuación en espacio libre entre $2400\ MHz$ (`Wi-Fi` / `BLE`) y $433\ MHz$ (`LoRa`) es:

$$\DeltaFSPL = 20 \cdot \log_{10}\left((2400) / (433)\right) = 20 \cdot \log_{10}(5.5427) \approx +14.87 \text{ dB}$$

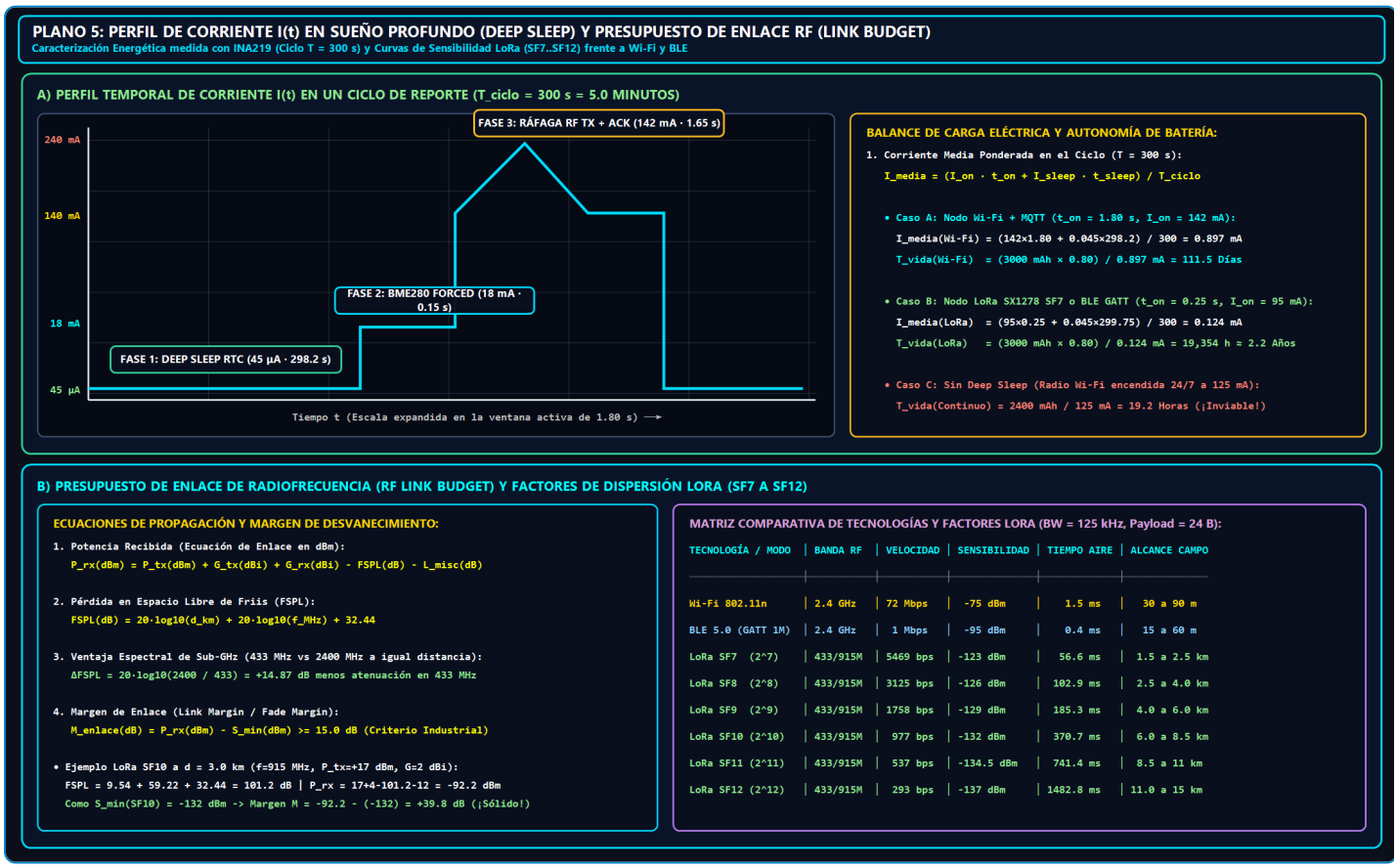
Una ventaja de $14.87\ \text{dB}$ equivale a multiplicar el alcance lineal en espacio libre por un factor de $5.54\times$ con la misma potencia de transmisión, además de sufrir menor absorción dieléctrica al atravesar follaje húmedo en invernaderos o muros de hormigón.

2.2 Margen de Desvanecimiento (Link Margin)

Para que un enlace industrial sea confiable ante lluvia, niebla o interferencia multitrayecto, exigimos un margen mínimo de seguridad de $15\ \text{dB}$ sobre la sensibilidad límite $S_{\min}$ del demodulador:

$$M_{\text{enlace}}\ (\text{dB}) = P_{\text{rx}}\ (\text{dBm}) - S_{\min}\ (\text{dBm}) \geq 15.0\ \text{dB}$$

📐 Plano 5: Perfil Temporal de Corriente en Deep Sleep y Presupuesto de Enlace RF



---

3. Reconexión Wi-Fi y MQTT con Backoff Exponencial y Jitter Estocástico

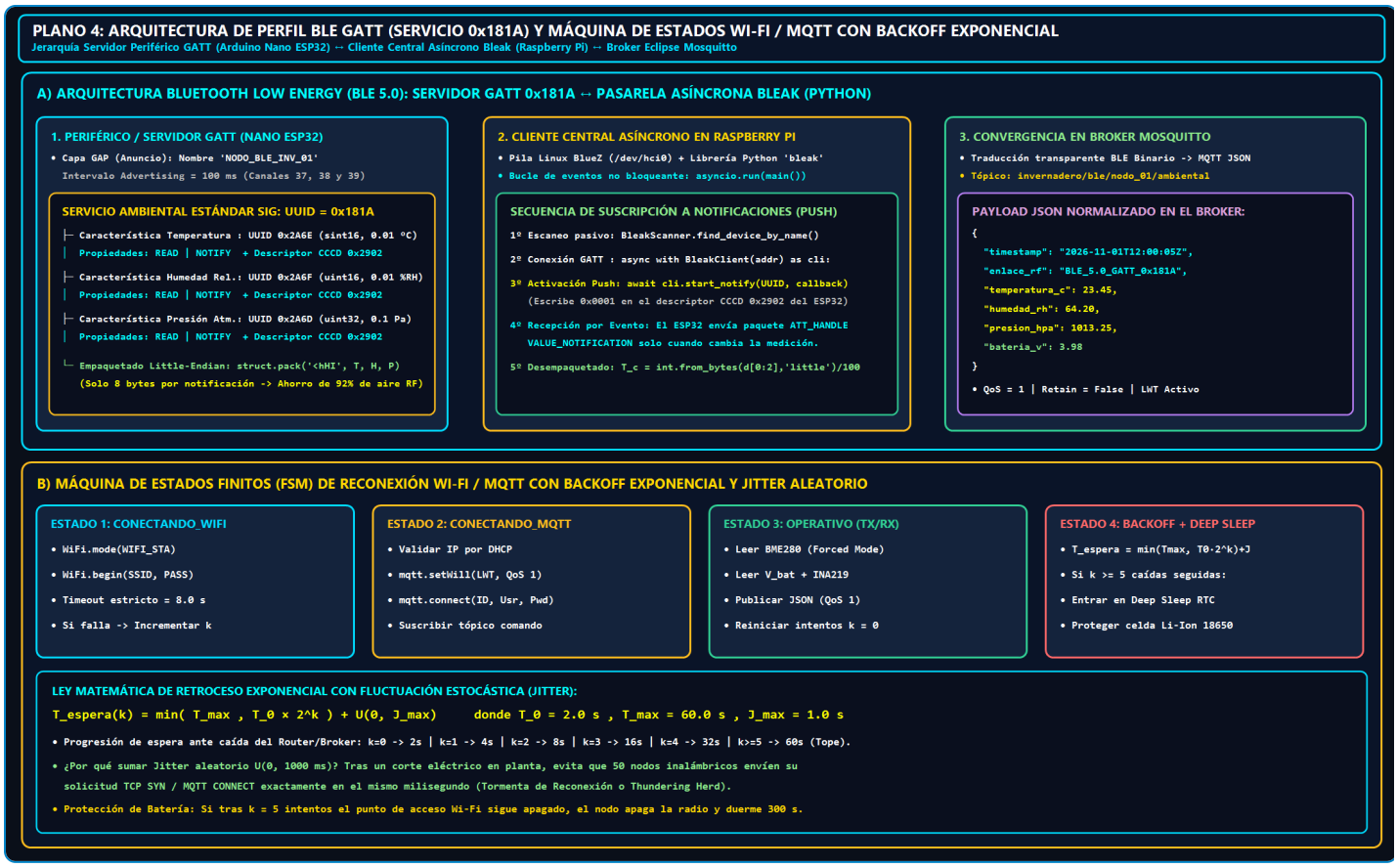
Cuando se restaura la energía eléctrica tras un apagón en planta, los microcontroladores arrancan en menos de $200\ \text{ms}$, mientras que el router Wi-Fi industrial y el servidor Linux tardan entre $30\ \text{s}$ y $60\ \text{s}$ en levantar sus servicios. Si N nodos reintentan conectarse en un bucle fijo cada $500\ \text{ms}$, provocan una *Tormenta de Reconexión (*Thundering Herd*)* y agotan sus baterías.

La solución de grado industrial implementa *Retroceso Exponencial Truncado con Fluctuación Aleatoria (*Jitter*)*:

$$T_{\text{espera}}(k) = \min\left(T_{\max},\ T_0 \cdot 2^k\right) + U(0, J_{\max})$$

Con $T_0 = 2.0\ \text{s}$, $T_{\max} = 60.0\ \text{s}$ y $J_{\max} = 1.0\ \text{s}$, la espera crece geométricamente ($2\ \text{s}, 4\ \text{s}, 8\ \text{s}, 16\ \text{s}, 32\ \text{s}, 60\ \text{s}$) y el término aleatorio uniforme $U(0,\ 1000\ \text{ms})$ desincroniza las ráfagas TCP SYN entre nodos.

📐 Plano 4: Arquitectura de Perfil BLE GATT 0x181A y Máquina de Estados de Reconexión con Backoff



---

4. Modelado Energético en Sueño Profundo (`Deep Sleep`) y Vida Útil de Batería

En un nodo cíclico con periodo total $T_{ciclo} = t_{activo} + t_{sleep}$, la **corriente media ponderada** drenada de la batería es:

$$I_{media} = \frac{I_{activo} \cdot t_{activo} + I_{sleep} \cdot t_{sleep}}{t_{activo} + t_{sleep}}$$

Aplicando un factor de descarga segura $\eta_{bat} = 0.80$ para preservar la química de la celda `Li-Ion 18650` ($Q_{bat} = 3000\ mAh$), la autonomía en días es:

$$T_{vida\ (días)} = \frac{Q_{bat\ (mAh)} \cdot \eta_{bat}}{24 \cdot I_{media\ (mA)}}$$

- **Sin Deep Sleep ($I_{media} = 125\ mA$ continuo):** $T_{vida} = (3000 \times 0.80) / (24 \times 125) = 0.80$ días $= 19.2$ horas.
- **Con Deep Sleep en Wi-Fi ($t_{activo} = 1.8$ s a $142\ mA$, $t_{sleep} = 298.2$ s a $45\ \mu A$):** $I_{media} = 0.897\ mA \implies T_{vida} = \mathbf{111.5\ días}$.
- **Con Deep Sleep en LoRa `SF7` o `BLE` ($t_{activo} = 0.25$ s a $95\ mA$, $t_{sleep} = 299.75$ s a $45\ \mu A$):** $I_{media} = 0.124\ mA \implies T_{vida} = \mathbf{806\ días\ (\approx 2.2\ años)}$.

PLANO Y ESQUEMÁTICO DEL CIRCUITO — MÓDULO 06: NODOS INALÁMBRICOS Y PLACAS PCB SOLDADAS

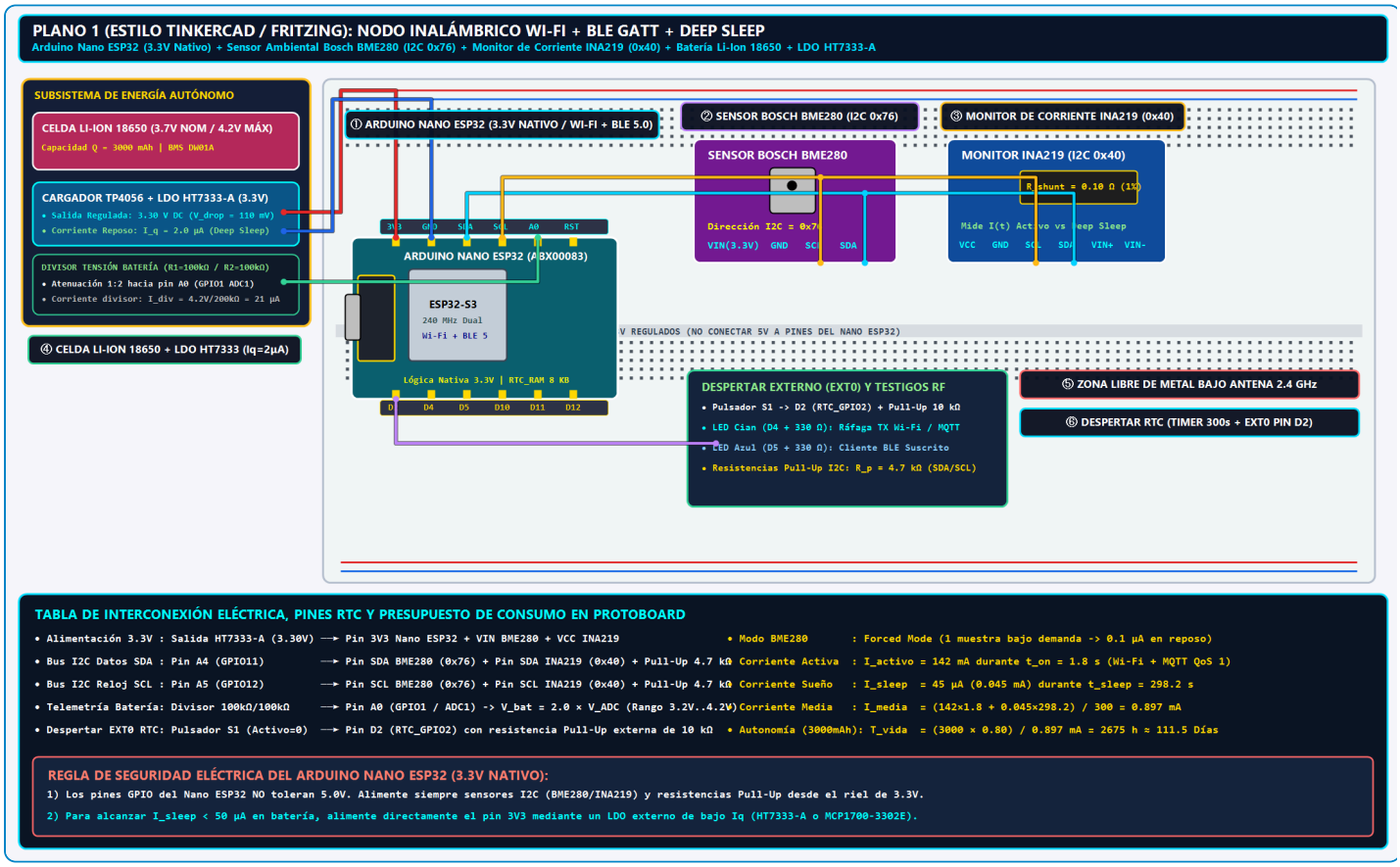
Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo 06 / Video 06: Planos de Montaje en Protoboard (Estilo Tinkercad / Fritzing) de Alta Legibilidad y Planos de Circuito Impreso (`PCB FR-4`) con Montaje por Soldadura

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Nodo Inalámbrico Wi-Fi + BLE + Deep Sleep en Protoboard

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad del Nodo Wi-Fi y BLE con Sensor BME280, Monitor INA219 y Batería Li-Ion 18650

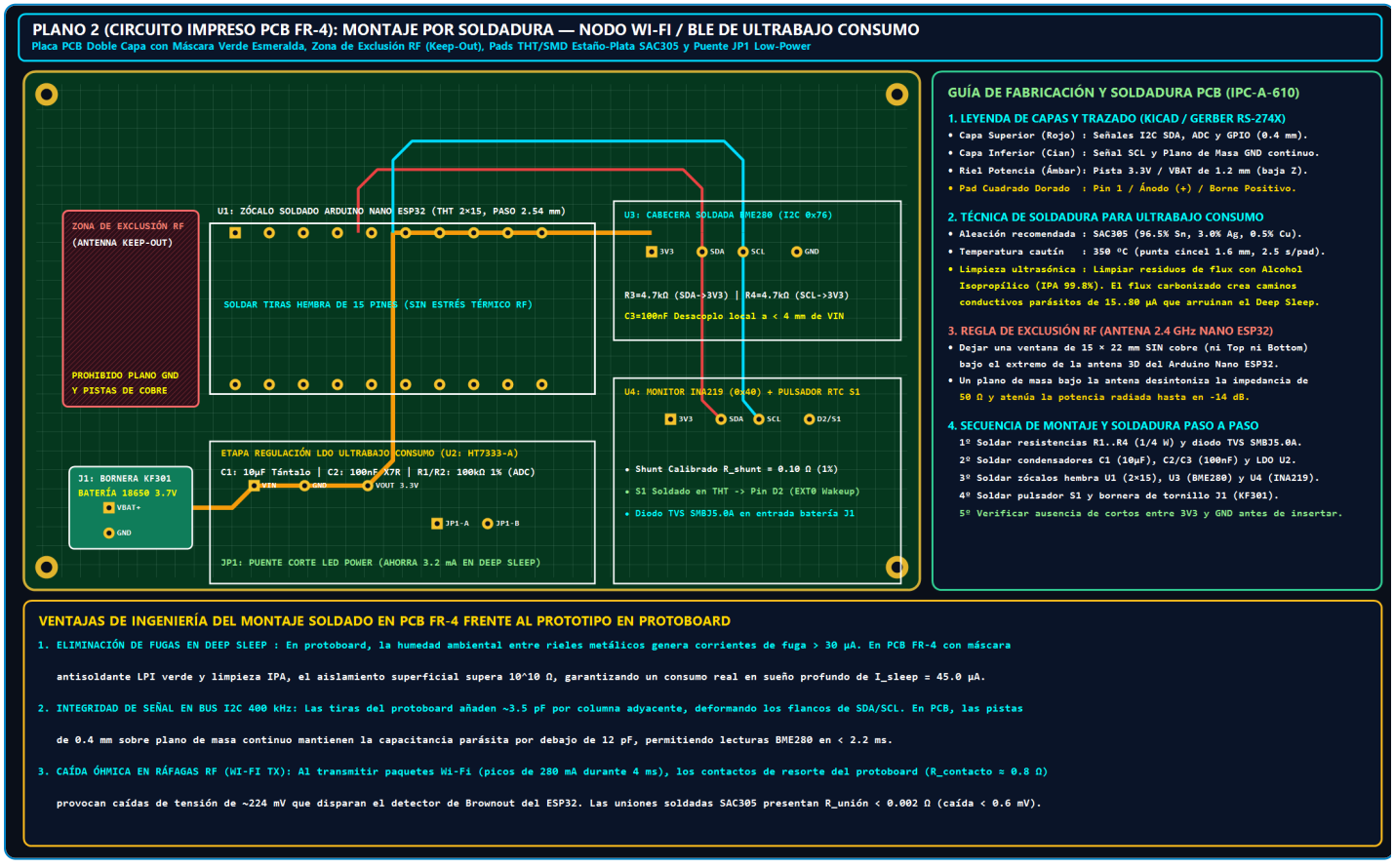


1.1 Resumen de Asignación de Pines del Arduino Nano ESP32 (`3.3 V` Nativo)

Callout en Plano	Subsistema / Módulo	Pines en Protoboard	Parámetros Eléctricos y de Firmware
①	**Arduino Nano ESP32 (** `ABX00083` **)**	Riel `3V3` y `GND`	SoC `ESP32-S3` con $8\ KB$ de memoria RTC (`RTC_DATA_ATTR`)
②	**Sensor Ambiental Bosch** `BME280`	`SDA -> A4` / `SCL -> A5`	Dirección I^2C `0x76` , operado en `MODE_FORCED` ($0.1\ \mu A$ en reposo)
③	**Monitor de Corriente** `INA219`	`SDA -> A4` / `SCL -> A5`	Dirección I^2C `0x40` , resistencia Shunt $R_{shunt} = 0.10\ \Omega\ (1\%)$
④	**Batería** `18650` **+ LDO** `HT7333-A`	Salida `3.30 V` → `Pin 3V3`	Corriente de reposo $I_q = 2.0\ \mu A$ y divisor $100\ k\Omega/100\ k\Omega$ hacia `A0`
⑤	**Antena Cerámica / 3D** `2.4 GHz`	Extremo izquierdo del módulo	Libre de cables puente y planos metálicos a $15\ mm$ a la redonda
⑥	**Despertar RTC (** `Timer` **+** `EXT0` **)**	Pulsador `S1` → `Pin D2`	`esp_sleep_enable_timer_wakeup(300e6)` + `esp_sleep_enable_ext0_wakeup(GPIO_NUM_2, 0)`

2. Plano 2 (Circuito Impreso `PCB FR-4`): Montaje por Soldadura del Nodo de Ultrabajo Consumo

📐 Plano 2: Placa de Circuito Impreso PCB FR-4 Doble Capa con Montaje por Soldadura del Nodo Inalámbrico

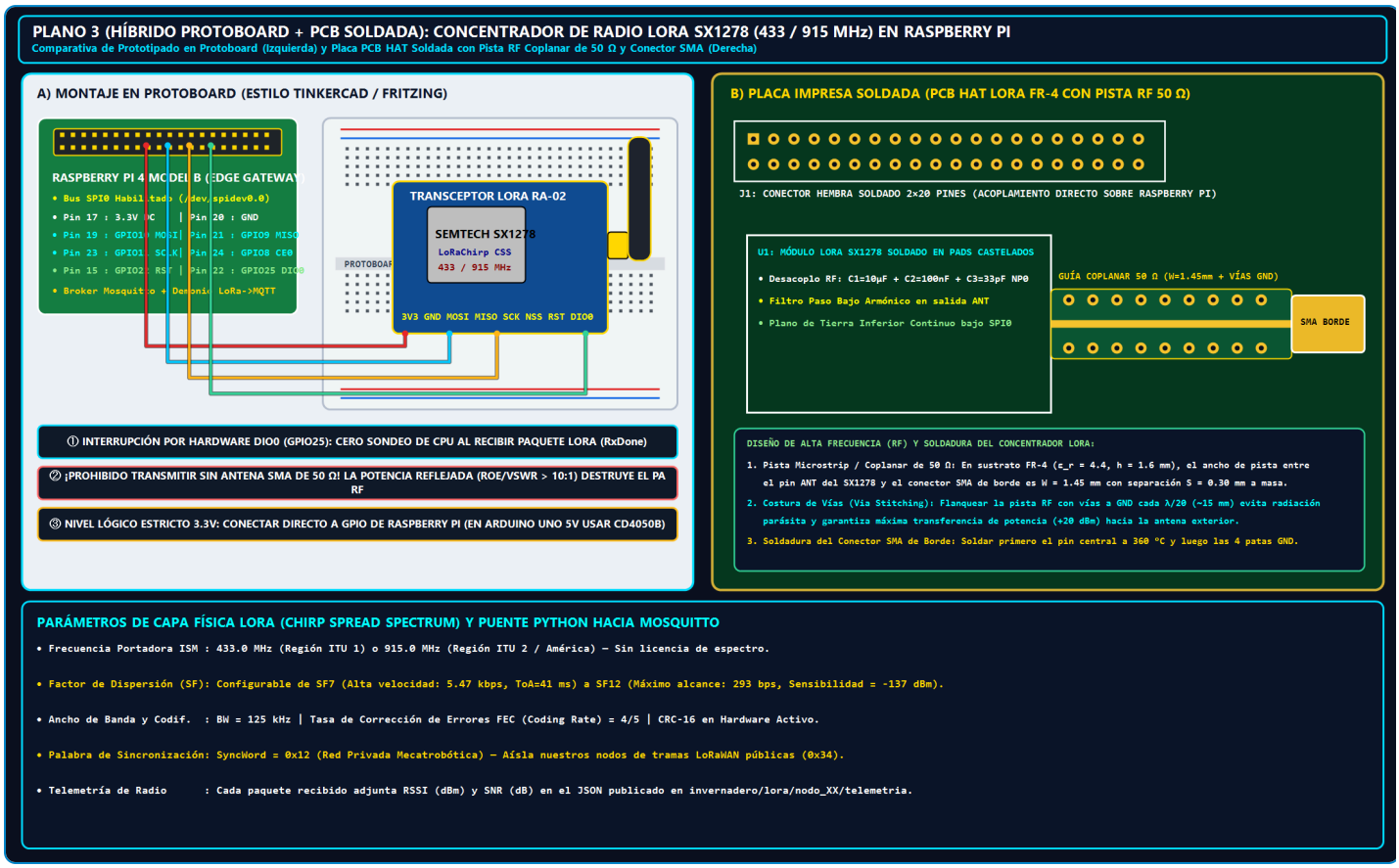


2.1 Especificaciones de Fabricación PCB (`KiCad` / `IPC-2221`) y Soldadura (`IPC-A-610`)

Elemento en la Placa PCB	Especificación Geométrica y Metalúrgica	Razón de Ingeniería (RF y Ultrabajo Consumo)
Zona de Exclusión RF (`Keep-Out`)	Ventana de 15×22 mm **sin cobre** en `Top` ni `Bottom` bajo la antena del Nano ESP32	Evita desintonizar la antena de 2.4 GHz (un plano GND debajo atenúa hasta -14 dB)
Puente Soldable `JP1` (Corte LED)	Dos pads dorados de 2.0×2.0 mm en serie con el LED indicador de encendido	Abrir `JP1` en despliegue final elimina el consumo fijo de 3.2 mA del LED Power
Limpieza de Flux con `IPA 99.8%`	Lavado con cepillo antiestático tras soldar con aleación `SAC305` a $350\ ^\circ$C	Los residuos de flux forman resistencias parásitas superficiales que drenan $15..80\ \mu$A
Zócalos Hembra `THT` (`U1, U3, U4`)	Agujeros metalizados $\varnothing 1.0$ mm con anillo anular dorado $\varnothing 1.8$ mm	Permite sustituir el Nano ESP32 o el `BME280` en campo en 5 segundos sin desoldar

3. Plano 3 (Híbrido Protoboard + `PCB HAT Soldada`): Concentrador `LoRa SX1278` en Raspberry Pi

📐 Plano 3: Montaje en Protoboard y Placa PCB HAT Soldada con Guía de Onda Coplanar de 50 Ohmios del Concentrador LoRa SX1278



4. Plano 4: Arquitectura de Perfil BLE GATT (`0x181A`) y Máquina de Estados con Backoff Exponencial

📐 Plano 4: Jerarquía del Servidor BLE GATT 0x181A y Máquina de Estados de Reconexión Wi-Fi/MQTT

PLANO 4: ARQUITECTURA DE PERFIL BLE GATT (SERVICIO 0x181A) Y MÁQUINA DE ESTADOS WI-FI / MQTT CON BACKOFF EXPONENCIAL

Jerarquía Servidor Periférico GATT (Arduino Nano ESP32) ↔ Cliente Central Asíncrono Bleak (Raspberry Pi) ↔ Broker Eclipse Mosquitto

A) ARQUITECTURA BLUETOOTH LOW ENERGY (BLE 5.0): SERVIDOR GATT 0x181A ↔ PASARELA ASÍNCRONA BLEAK (PYTHON)

1. PERIFÉRICO / SERVIDOR GATT (NANO ESP32)

- Capa GAP (Anuncio): Nombre 'NODO_BLE_INV_01'
  Intervalo Advertising = 100 ms (Canales 37, 38 y 39)

SERVICIO AMBIENTAL ESTÁNDAR SIG: UUID = 0x181A

- ├ Característica Temperatura : UUID 0x2A6E (sint16, 0.01 °C)
  | Propiedades: READ | NOTIFY + Descriptor CCCD 0x2902
- ├ Característica Humedad Rel.: UUID 0x2A6F (uint16, 0.01 %RH)
  | Propiedades: READ | NOTIFY + Descriptor CCCD 0x2902
- ├ Característica Presión Atm.: UUID 0x2A6D (uint32, 0.1 Pa)
  | Propiedades: READ | NOTIFY + Descriptor CCCD 0x2902
- └ Empaquetado Little-Endian: struct.pack('<hHI', T, H, P)
  (Solo 8 bytes por notificación -> Ahorro de 92% de aire RF)

2. CLIENTE CENTRAL ASÍNCRONO EN RASPBERRY PI

- Pila Linux BlueZ (/dev/hci0) + Librería Python 'bleak'
- Bucle de eventos no bloqueante: asyncio.run(main())

SECUENCIA DE SUSCRIPCIÓN A NOTIFICACIONES (PUSH)

1º Escaneo pasivo: BleakScanner.find_device_by_name()

2º Conexión GATT : async with BleakClient(addr) as cli:

3º Activación Push: await cli.start_notify(UUID, callback)
(Escribe 0x0001 en el descriptor CCCD 0x2902 del ESP32)

4º Recepción por Evento: El ESP32 envía paquete ATT_HANDLE VALUE_NOTIFICATION solo cuando cambia la medición.

5º Desempaquetado: T_c = int.from_bytes(d[0:2],'little')/100

3. CONVERGENCIA EN BROKER MOSQUITTO

- Traducción transparente BLE Binario -> MQTT JSON
- Tópico: invernadero/ble/nodo_01/ambiental

PAYLOAD JSON NORMALIZADO EN EL BROKER:

```
{
  "timestamp": "2026-11-01T12:00:05Z",
  "enlace_rf": "BLE_5.0_GATT_0x181A",
  "temperatura_c": 23.45,
  "humedad_rh": 64.20,
  "presion_hpa": 1013.25,
  "bateria_v": 3.98
}
```

- QoS = 1 | Retain = False | LWT Activo

B) MÁQUINA DE ESTADOS FINITOS (FSM) DE RECONEXIÓN WI-FI / MQTT CON BACKOFF EXPONENCIAL Y JITTER ALEATORIO

ESTADO 1: CONECTANDO_WIFI

- WiFi.mode(WIFI_STA)
- WiFi.begin(SSID, PASS)
- Timeout estricto = 8.0 s
- Si falla -> Incrementar k

ESTADO 2: CONECTANDO_MQTT

- Validar IP por DHCP
- mqtt.setWill(LWT, QoS 1)
- mqtt.connect(ID, Usr, Pwd)
- Suscribir tópico comando

ESTADO 3: OPERATIVO (TX/RX)

- Leer BME280 (Forced Mode)
- Leer V_bat + INA219
- Publicar JSON (QoS 1)
- Reiniciar intentos k = 0

ESTADO 4: BACKOFF + DEEP SLEEP

- T_espera = min(Tmax, T0·2^k)+J
- Si k >= 5 caídas seguidas:
- Entrar en Deep Sleep RTC
- Proteger celda Li-Ion 18650

LEY MATEMÁTICA DE RETROCESO EXPONENCIAL CON FLUCTUACIÓN ESTOCÁSTICA (JITTER):

$T_{espera}(k) = \min(T_{max} , T_0 \times 2^k) + U(0, J_{max})$ donde $T_0 = 2.0$ s , $T_{max} = 60.0$ s , $J_{max} = 1.0$ s

- Progresión de espera ante caída del Router/Broker: k=0 -> 2s | k=1 -> 4s | k=2 -> 8s | k=3 -> 16s | k=4 -> 32s | k>=5 -> 60s (Tope).
- ¿Por qué sumar Jitter aleatorio U(0, 1000 ms)? Tras un corte eléctrico en planta, evita que 50 nodos inalámbricos envíen su solicitud TCP SYN / MQTT CONNECT exactamente en el mismo milisegundo (Tormenta de Reconexión o Thundering Herd).
- Protección de Batería: Si tras k = 5 intentos el punto de acceso Wi-Fi sigue apagado, el nodo apaga la radio y duerme 300 s.

---

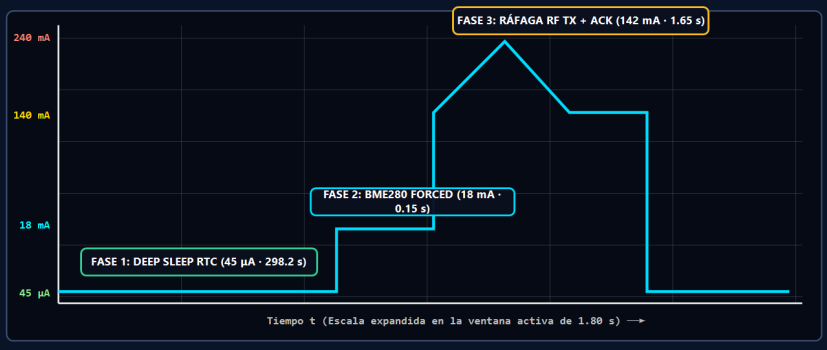
5. Plano 5: Perfil de Corriente $I(t)$ en `Deep Sleep` y Presupuesto de Enlace RF (`SF7..SF12`)

📐 Plano 5: Perfil Temporal de Corriente medido con INA219 y Presupuesto de Enlace RF

PLANO 5: PERFIL DE CORRIENTE I(t) EN SUEÑO PROFUNDO (DEEP SLEEP) Y PRESUPUESTO DE ENLACE RF (LINK BUDGET)

Caracterización Energética medida con INA219 (Ciclo T = 300 s) y Curvas de Sensibilidad LoRa (SF7..SF12) frente a Wi-Fi y BLE

A) PERFIL TEMPORAL DE CORRIENTE I(t) EN UN CICLO DE REPORTE (T_ciclo = 300 s = 5.0 MINUTOS)



BALANCE DE CARGA ELÉCTRICA Y AUTONOMÍA DE BATERÍA:

1. Corriente Media Ponderada en el Ciclo (T = 300 s):
   $I_{media} = (I_{on} \cdot t_{on} + I_{sleep} \cdot t_{sleep}) / T_{ciclo}$

- Caso A: Nodo Wi-Fi + MQTT (t_{on} = 1.80 s, I_{on} = 142 mA):
  $I_{media(Wi-Fi)} = (142 \times 1.80 + 0.045 \times 298.2) / 300 = 0.897$ mA
  $T_{vida(Wi-Fi)} = (3000$ mAh × 0.80) / 0.897 mA = 111.5 Días
- Caso B: Nodo LoRa SX1278 SF7 o BLE GATT (t_{on} = 0.25 s, I_{on} = 95 mA):
  $I_{media(LoRa)} = (95 \times 0.25 + 0.045 \times 299.75) / 300 = 0.124$ mA
  $T_{vida(LoRa)} = (3000$ mAh × 0.80) / 0.124 mA = 19,354 h ≈ 2.2 Años
- Caso C: Sin Deep Sleep (Radio Wi-Fi encendida 24/7 a 125 mA):
  $T_{vida(Continuo)}$ = 2400 mAh / 125 mA = 19.2 Horas (¡Inviable!)

B) PRESUPUESTO DE ENLACE DE RADIOFRECUENCIA (RF LINK BUDGET) Y FACTORES DE DISPERSIÓN LORA (SF7 A SF12)

ECUACIONES DE PROPAGACIÓN Y MARGEN DE DESVANECIMIENTO:

1. Potencia Recibida (Ecuación de Enlace en dBm):
   $P_{rx(dBm)} = P_{tx(dBm)} + G_{tx(dBi)} + G_{rx(dBi)} - FSPL(dB) - L_{misc(dB)}$

2. Pérdida en Espacio Libre de Friis (FSPL):
   $FSPL(dB) = 20 \cdot \log10(d_{km}) + 20 \cdot \log10(f_{MHz}) + 32.44$

3. Ventaja Espectral de Sub-GHz (433 MHz vs 2400 MHz a igual distancia):
   $\Delta FSPL = 20 \cdot \log10(2400 / 433) = +14.87$ dB menos atenuación en 433 MHz

4. Margen de Enlace (Link Margin / Fade Margin):
   $M_{enlace(dB)} = P_{rx(dBm)} - S_{min(dBm)} >= 15.0$ dB (Criterio Industrial)

- Ejemplo LoRa SF10 a d = 3.0 km (f=915 MHz, P_{tx}=+17 dBm, G=2 dBi):
  FSPL = 9.54 + 59.22 + 32.44 = 101.2 dB | P_{rx} = 17+4-101.2-12 = -92.2 dBm
  Como $S_{min(SF10)}$ = -132 dBm -> Margen M = -92.2 - (-132) = +39.8 dB (¡Sólido!)

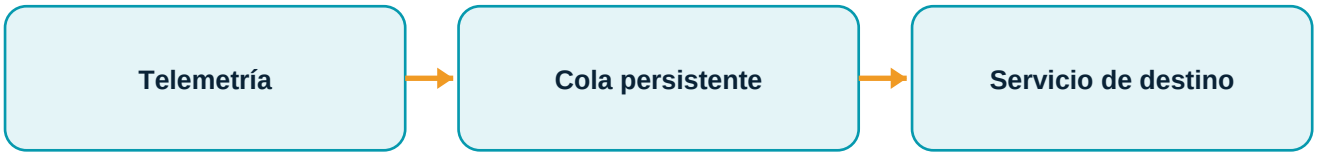
MATRIZ COMPARATIVA DE TECNOLOGÍAS Y FACTORES LORA (BW = 125 kHz, Payload = 24 B):

TECNOLOGÍA / MODO	BANDA RF	VELOCIDAD	SENSIBILIDAD	TIEMPO AIRE	ALCANCE CAMPO
Wi-Fi 802.11n	2.4 GHz	72 Mbps	-75 dBm	1.5 ms	30 a 90 m
BLE 5.0 (GATT 1M)	2.4 GHz	1 Mbps	-95 dBm	0.4 ms	15 a 60 m
LoRa SF7 (2^7)	433/915M	5469 bps	-123 dBm	56.6 ms	1.5 a 2.5 km
LoRa SF8 (2^8)	433/915M	3125 bps	-126 dBm	102.9 ms	2.5 a 4.0 km
LoRa SF9 (2^9)	433/915M	1758 bps	-129 dBm	185.3 ms	4.0 a 6.0 km
LoRa SF10 (2^10)	433/915M	977 bps	-132 dBm	370.7 ms	6.0 a 8.5 km
LoRa SF11 (2^11)	433/915M	537 bps	-134.5 dBm	741.4 ms	8.5 a 11 km
LoRa SF12 (2^12)	433/915M	293 bps	-137 dBm	1482.8 ms	11.0 a 15 km

Capítulo 22 · Docker, almacenamiento SSD y búfer SQLite WAL

Conceptos clave

Usa almacenamiento persistente y conserva cada registro hasta recibir confirmación. Prueba el reinicio con registros pendientes.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 07

Persistencia en Series Temporales, Orquestación con Docker Compose, Node-RED, InfluxDB y Grafana

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 07 — Capa de Almacenamiento Histórico, Contenedores ARM64 y Observabilidad Industrial

---

1. Objetivos de Ingeniería del Laboratorio

- Montar una unidad de estado sólido externa (`SSD USB 3.0`) en la Raspberry Pi y configurar el reloj de tiempo real por hardware (`DS3231` en el bus `I2C1` dirección `0x68`) junto con el detector optoacoplado de corte eléctrico (`PC817` en `GPIO 17`).
- Desplegar mediante `Docker Compose` una arquitectura multicontenedor aislada sobre Linux ARM64 compuesta por `Eclipse Mosquitto 2` , `InfluxDB 2.7` , `Node-RED` y `Grafana OSS` .
- Programar y auditar el demonio en Python de ingesta resiliente **Store-and-Forward** (`ingestor_store_and_forward_sqlite_influxdb.py`) que respalda muestras en una cola `SQLite3` en modo `WAL` cuando la base de datos principal se encuentra en mantenimiento o desconectada.
- Configurar políticas de retención de datos (`30 días` a `1 Hz`) y tareas automáticas de agregación o *Downsampling* en lenguaje `Flux` (`300:1` hacia ventanas de `5 minutos` con retención de `365 días`).
- Construir paneles de supervisión industrial en `Grafana` cumpliendo la norma de ergonomía visual **ANSI/ISA-101** y flujos de alerta con histéresis en **Node-RED**.

---

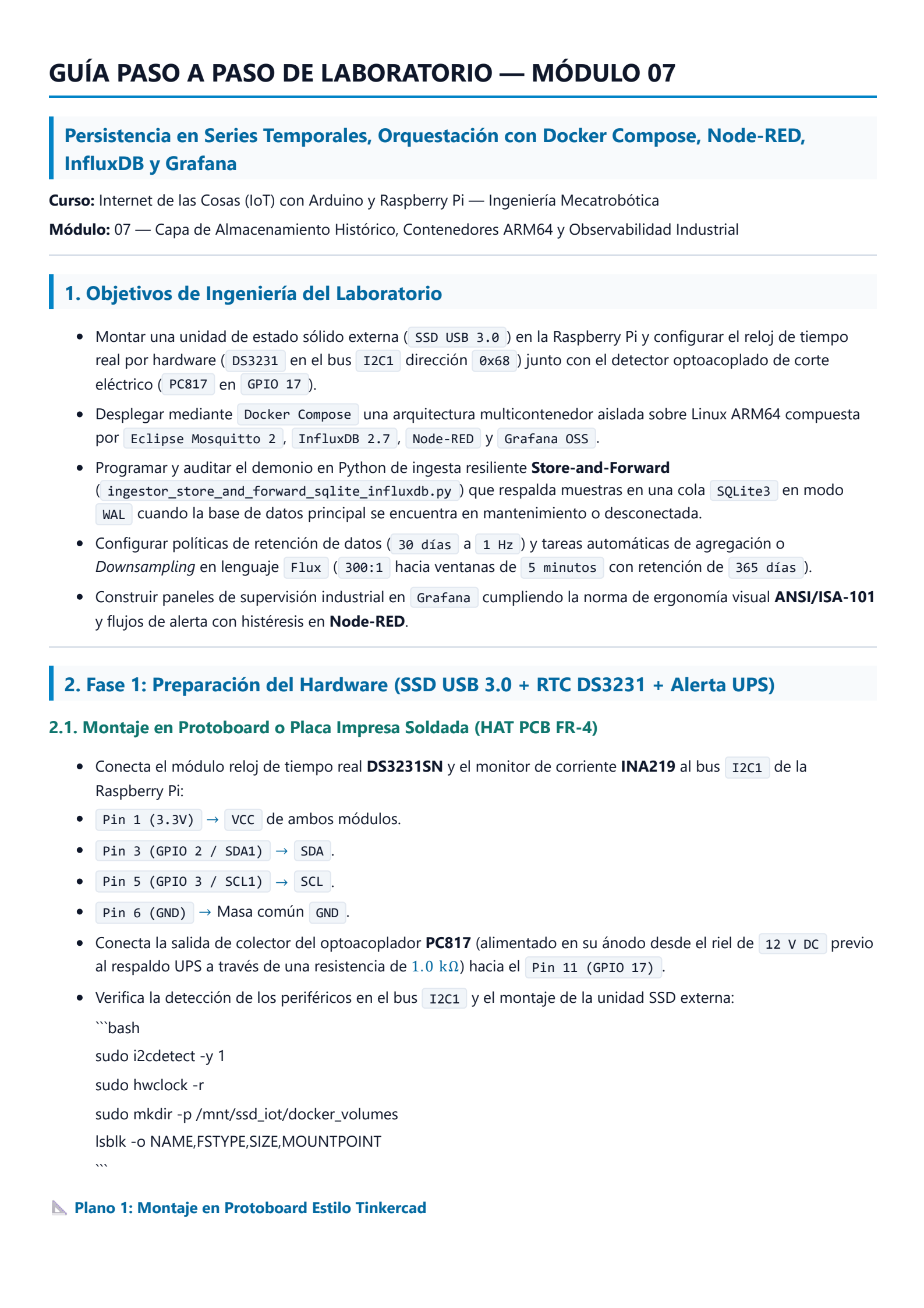
2. Fase 1: Preparación del Hardware (SSD USB 3.0 + RTC DS3231 + Alerta UPS)

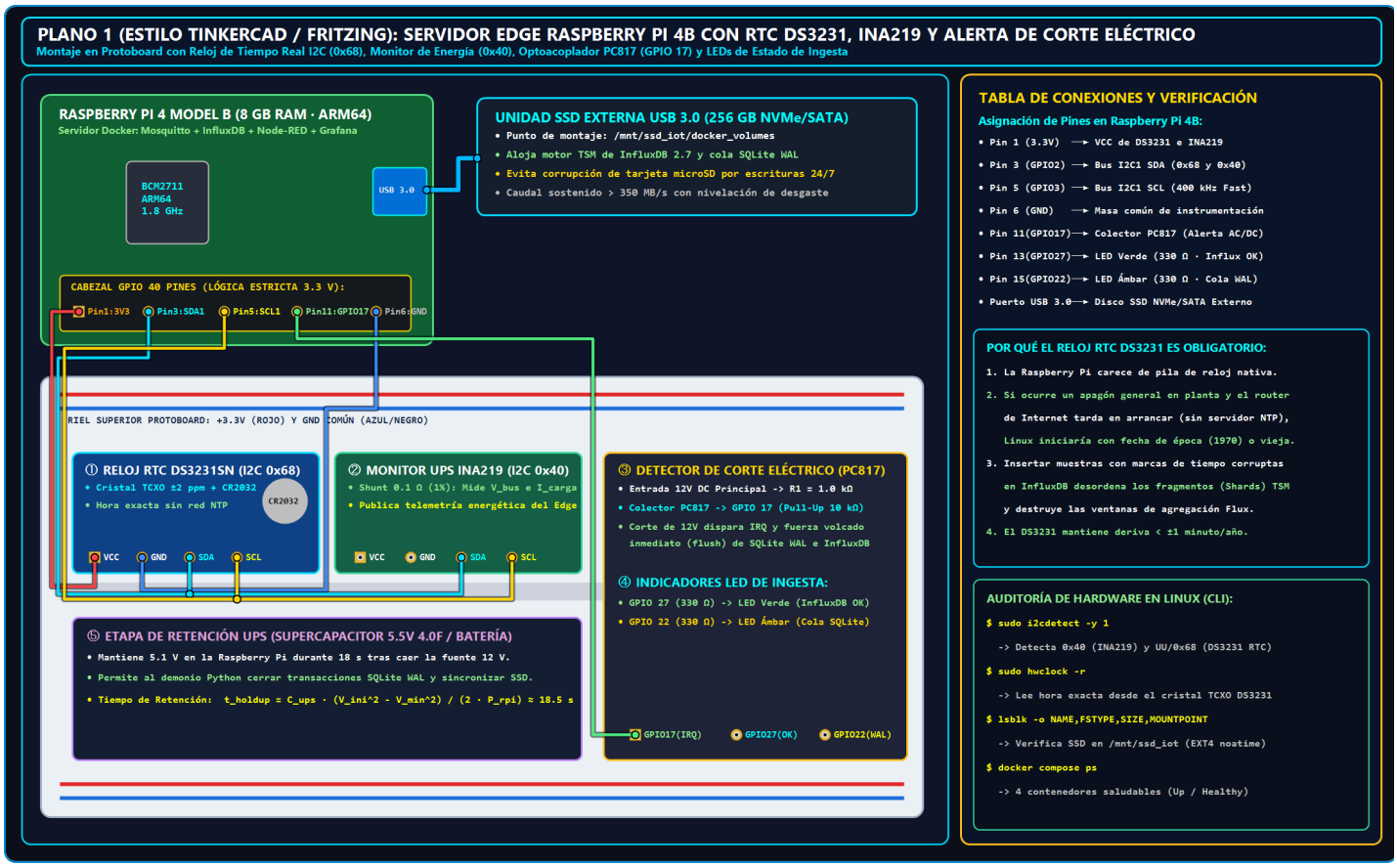
2.1. Montaje en Protoboard o Placa Impresa Soldada (HAT PCB FR-4)

- Conecta el módulo reloj de tiempo real **DS3231SN** y el monitor de corriente **INA219** al bus `I2C1` de la Raspberry Pi:
- `Pin 1 (3.3V)` → `VCC` de ambos módulos.
- `Pin 3 (GPIO 2 / SDA1)` → `SDA` .
- `Pin 5 (GPIO 3 / SCL1)` → `SCL` .
- `Pin 6 (GND)` → Masa común `GND` .
- Conecta la salida de colector del optoacoplador **PC817** (alimentado en su ánodo desde el riel de `12 V DC` previo al respaldo UPS a través de una resistencia de $1.0\ \text{k}\Omega$) hacia el `Pin 11 (GPIO 17)` .
- Verifica la detección de los periféricos en el bus `I2C1` y el montaje de la unidad SSD externa:

  ```bash
  sudo i2cdetect -y 1
  sudo hwclock -r
  sudo mkdir -p /mnt/ssd_iot/docker_volumes
  lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT
  ```

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad





---

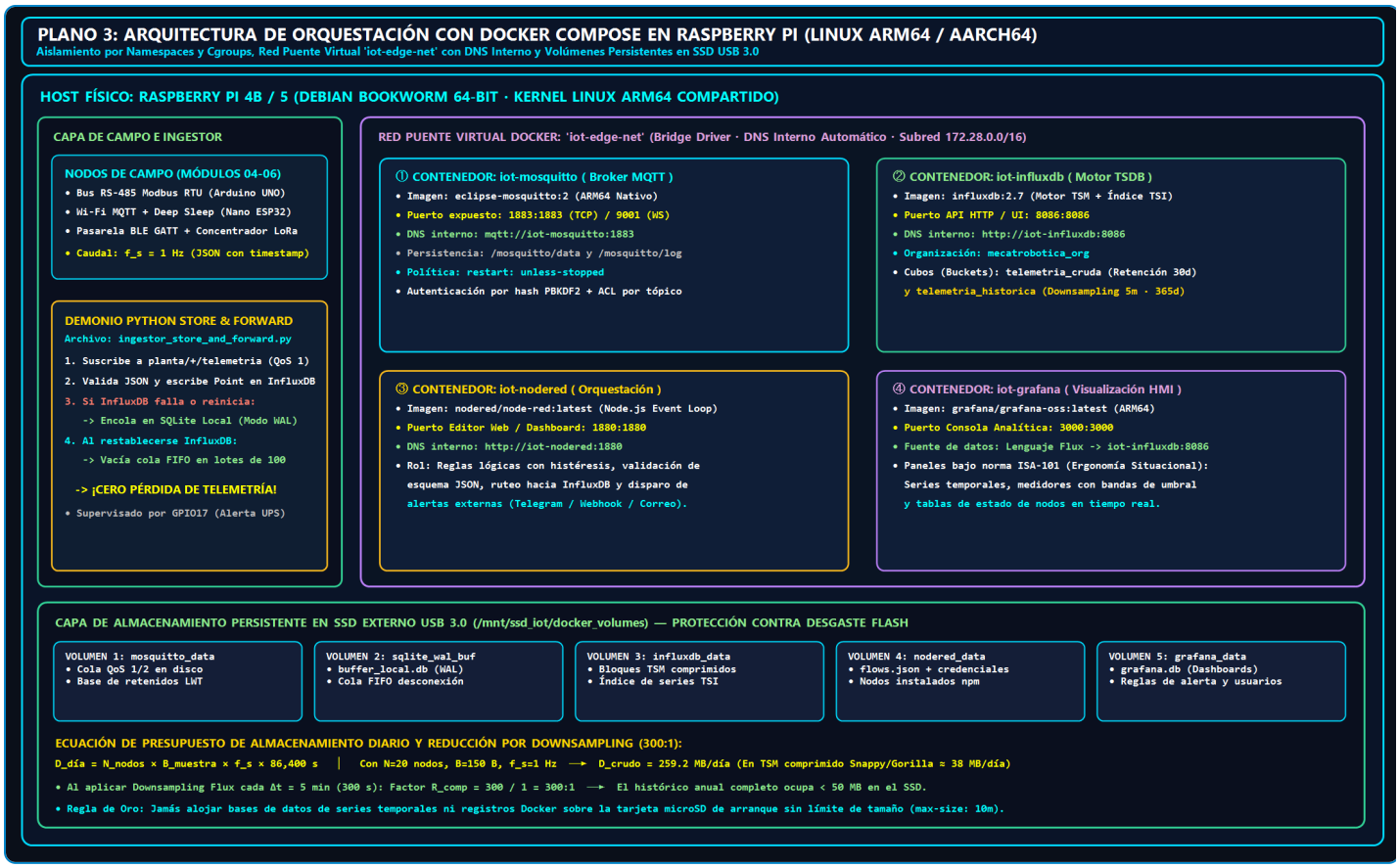
3. Fase 2: Despliegue de la Pila con Docker Compose en ARM64

- Copia el archivo `docker-compose.yml` incluido en la carpeta `codigo_fuente/` hacia `/opt/mecatrobotica/stack_v07/docker-compose.yml`.
- Levanta los cuatro servicios en segundo plano y verifica su estado de salud:

  ```bash
  cd /opt/mecatrobotica/stack_v07
  docker compose up -d
  docker compose ps
  ```

- Comprueba que los cuatro contenedores (`iot-mosquitto`, `iot-influxdb`, `iot-nodered` e `iot-grafana`) reporten estado `Up` y que `iot-influxdb` alcance el estado `healthy` en el puerto `8086`.

📐 Plano 3: Arquitectura Docker Compose en Raspberry Pi ARM64



---

4. Fase 3: Prueba de Fuego del Ingestor Store-and-Forward (SQLite WAL + InfluxDB)

- Instala las bibliotecas cliente en el entorno virtual de la Raspberry Pi:

  ```bash
  pip install paho-mqtt influxdb-client
  ```

- Ejecuta el demonio `ingestor_store_and_forward_sqlite_influxdb.py`.
- **Simulación de caída de base de datos**: Detén temporalmente el contenedor de InfluxDB mientras los nodos continúan publicando telemetría:

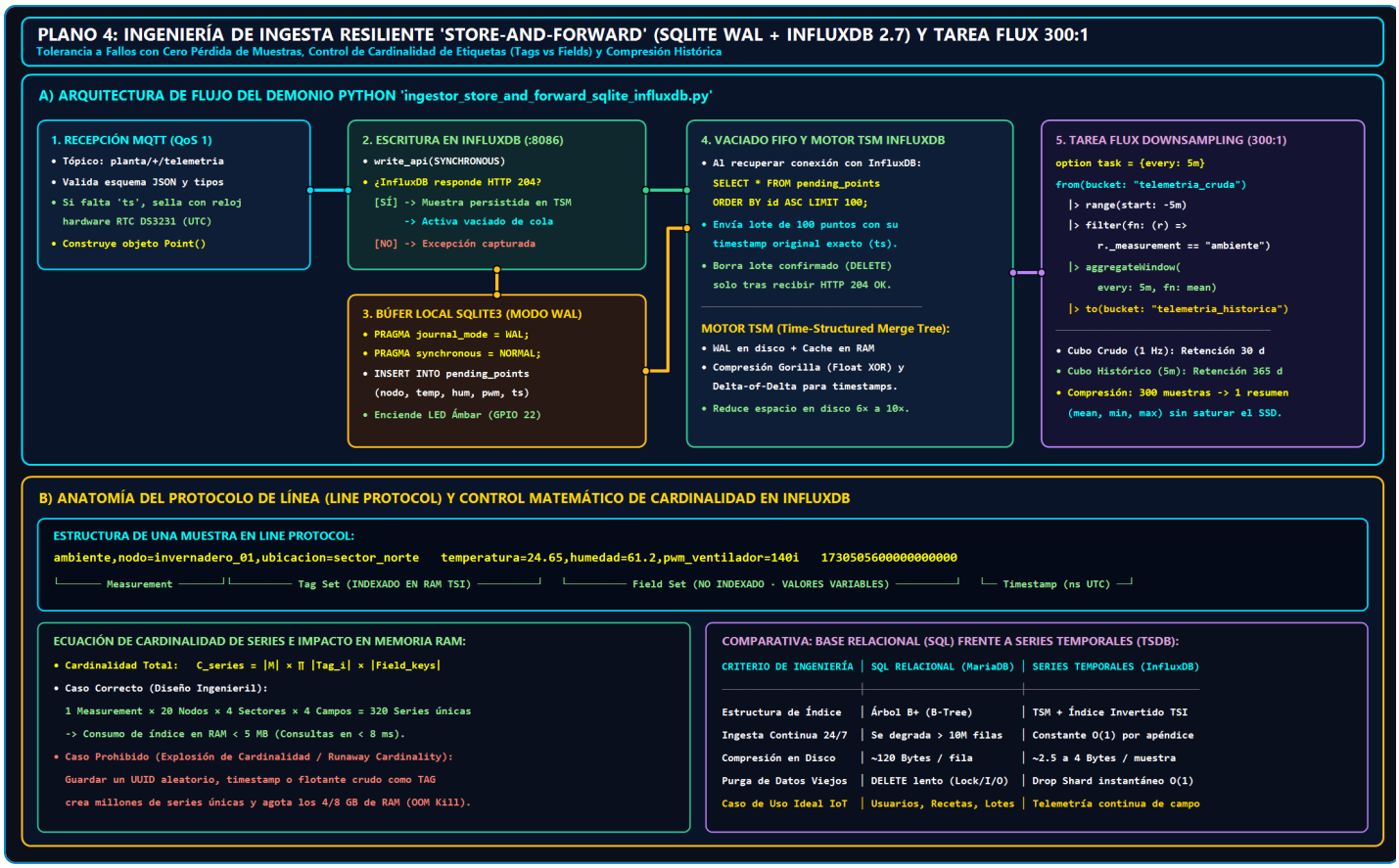
  ```bash
  docker stop iot-influxdb
  ```

- Observa en la consola cómo el demonio captura la excepción de conexión y desvía inmediatamente cada muestra entrante hacia la tabla `pending_points` de `buffer_local_wal.db` (`PRAGMA journal_mode=WAL`).
- **Restauración y vaciado por lotes**: Vuelve a iniciar el contenedor:

  ```bash
  docker start iot-influxdb
  ```

- Verifica cómo en el siguiente ciclo de recepción el demonio vacía automáticamente la cola FIFO en lotes de `100 muestras` (`ORDER BY id ASC LIMIT 100`), insertando toda la serie histórica con sus marcas de tiempo originales intactas.

📐 Plano 4: Arquitectura Store-and-Forward y Tarea Flux 300:1



5. Fase 4: Tarea de Downsampling en Flux y Paneles ISA-101 en Grafana

- Accede a la interfaz web de InfluxDB (`http://<IP_RASPBERRY>:8086`), crea el segundo cubo llamado `telemetria_historica` con retención de `365 días` y registra el script `tarea_downsampling_flux_300a1.flux` en la sección **Tasks**.
- Accede a **Node-RED** (`http://<IP_RASPBERRY>:1880`), importa el archivo `flujo_orquestacion_alertas_nodered.json` y despliega el flujo con validación de esquema JSON e histéresis térmica ($T_{on} = 30.0\ ^\circ C$, $T_{off} = 28.5\ ^\circ C$).
- Accede a **Grafana** (`http://<IP_RASPBERRY>:3000`), configura la fuente de datos **InfluxDB** seleccionando el lenguaje **Flux**, la URL interna `http://iot-influxdb:8086` y el token de administración.
- Diseña el panel de supervisión aplicando la jerarquía de tres niveles y la semántica de colores de la norma **ANSI/ISA-101**.

📐 Plano 5: Ergonomía Visual ISA-101 en Grafana y Flujo Node-RED

PLANO 5: DISEÑO DE INTERFACES HMI BAJO NORMA ISA-101 EN GRAFANA Y FLUJO DE REGLAS EN NODE-RED

Conciencia Situacional para Operadores de Planta, Jerarquía de 3 Niveles, Histéresis de Alerta y Disponibilidad Operativa (SLA)

A) PANEL DE SUPERVISIÓN INDUSTRIAL GRAFANA (:3000) DISEÑADO BAJO ESTÁNDAR ANSI/ISA-101.01

TEMPERATURA PROCESO (NODO 01)

24.6 °C

Banda Nominal: [18.0 .. 28.0 °C] (OK)

HUMEDAD RELATIVA Y PRESIÓN

61.2 %RH | 1013 hPa

Sensor Bosch BME280 (I2C 0x76)

PRE-ALARMA PREVENTIVA (ADVERTENCIA)

29.4 °C (Sector Sur)

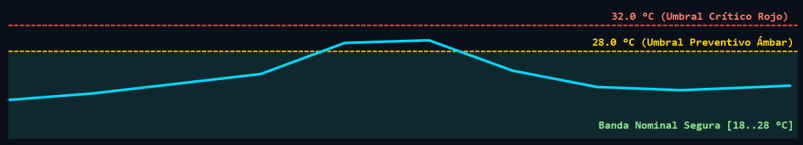
Supera umbral preventivo > 28.0 °C

ALARMA CRÍTICA ISA-18.2 (PRIORIDAD 1)

0 ALARMAS ACTIVAS

Disponibilidad SLA: A = 99.982% (Online)

CURVA DE SERIE TEMPORAL CON BANDA OPERATIVA SOMBREADA (CONSULTA FLUX aggregateWindow)



REGLAS DE ORO DE ERGONOMÍA VISUAL ISA-101:

1. Prohibido el efecto 'Árbol de Navidad': Si todo brilla en rojo, verde y amarillo saturado, el operador sufre fatiga visual y no detecta fallos.
2. Fondos oscuros mate (#0A101C / #121E36) y rejillas de bajo contraste para descanso ocular en turno 24/7.
3. Reserva cromática estricta:
   - Cian / Esmeralda -> Operación dentro de consigna
   - Ámbar (#FFBE1E) -> Desviación preventiva
   - Rojo (#FF5050) -> Alarma crítica que exige acción
4. Incluir siempre unidades físicas (°C, %RH, mA, dBm).

B) FLUJO DE ORQUESTACIÓN EN NODE-RED (:1880) CON HISTÉRESIS DE ALARMA Y CÁLCULO DE DISPONIBILIDAD (SLA)

① MQTT In
planta/+/telemetria
(QoS 1 · JSON)

② JSON Parser &
Schema Validator
(Verifica llaves)

③ Function Node
Filtro Histéresis
(T_on=30, T_off=28.5)

④ RBE / Filter
Report by Exception
(Bloquea spam)

⑤ Salida Dual:
• InfluxDB Batch Out
• Alerta Telegram/MQTT

LÓGICA JAVASCRIPT CON MEMORIA DE CONTEXTO E HISTÉRESIS EN NODE-RED:

```
const T_ALARMA_ON  = 30.0; // Umbral de disparo (°C)
const T_ALARMA_OFF = 28.5; // Umbral de reposición (Banda ΔT = 1.5 °C)
let enAlarma = context.get('enAlarma') || false;
if (!enAlarma && msg.payload.temp >= T_ALARMA_ON) {
    enAlarma = true;  context.set('enAlarma', true);
    return [msg, { payload: { alerta: 'ALTA_TEMP', valor: msg.payload.temp } }];
} else if (enAlarma && msg.payload.temp <= T_ALARMA_OFF) {
    enAlarma = false; context.set('enAlarma', false);
}
```

INGENIERÍA DE CONFIABILIDAD Y DISPONIBILIDAD DEL SISTEMA (MTBF / MTTR):

1. Ecuación de Disponibilidad Estacionaria (Availability):
   A = MTBF / (MTBF + MTTR) = T_operativo / T_total

2. Efecto de Docker 'restart: unless-stopped' + Healthchecks:
   Reduce el Tiempo Medio de Reparación (MTTR) de horas (intervención humana manual) a menos de 3.5 segundos (reinicio automático).

3. Combinado con el búfer SQLite WAL (Store-and-Forward):
   La disponibilidad efectiva de datos alcanza el 100.0% frente a microcortes de red o reinicios de mantenimiento de contenedores.

LISTADO DE MATERIALES (BOM) Y PRESUPUESTO DE ALMACENAMIENTO — MÓDULO 07

Servidor Edge de Series Temporales, Respaldo UPS, Reloj RTC DS3231 y Placa HAT Soldada

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 07 — Persistencia en Series Temporales, Docker, Node-RED, InfluxDB y Grafana

---

1. Tabla Maestra de Componentes de Hardware (BOM)

Ítem	Designador	Componente / Especificación Técnica	Cantidad	Función en la Arquitectura del Módulo 07
1	`SBC1`	**Raspberry Pi 4 Model B (4 GB / 8 GB RAM)** o Raspberry Pi 5 con disipador de aluminio anodizado	1	Servidor Linux ARM64 para orquestación de los 4 contenedores Docker (`Mosquitto` , `InfluxDB` , `Node-RED` , `Grafana`).
2	`SSD1`	**Unidad de Estado Sólido Externa USB 3.0 (256 GB NVMe/SATA)** con soporte UASP y TRIM	1	Almacenamiento de alta durabilidad para bloques TSM de InfluxDB y cola SQLite WAL (evita destruir la tarjeta microSD).
3	`U1 / BT1`	**Módulo Reloj en Tiempo Real DS3231SN (`I2C 0x68`)** con oscilador TCXO compensado (± 2 ppm) y pila `CR2032`	1	Garantiza marcas de tiempo exactas en nanosegundos UTC tras cortes eléctricos aunque aún no exista sincronización NTP.
4	`U2`	**Sensor de Corriente y Tensión High-Side INA219 (`I2C 0x40`)** con resistencia Shunt Kelvin de $0.100\ \Omega$ (1%)	1	Monitorea en tiempo real la tensión del riel de $5.1\ V$ y el consumo de corriente del servidor Edge y del SSD.
5	`U3`	**Optoacoplador Fototransistor PC817C (DIP-4)** con aislamiento galvánico de $2500\ V_{rms}$	1	Detecta instantáneamente la caída del riel primario de $12\ V\ DC$ y dispara la interrupción en `GPIO 17` para volcado limpio (`flush`).
6	`C_UPS`	**Banco Supercapacitor de Respaldo** $5.5\ V$ $4.0\ F$ (o módulo UPS de celdas `18650` de $5.1\ V$ $3\ A$)	1	Sostiene la tensión de $5.1\ V$ durante 15 a $30\ s$ tras un apagón para cerrar archivos SQLite WAL e InfluxDB sin corrupción.
7	`SW1`	**Switch Ethernet Industrial de 5 Puertos (10/100/1000 Mbps)** para riel DIN con cables Cat6 blindados	1	Interconexión de baja latencia entre la pasarela Raspberry Pi, los puntos de acceso inalámbricos y las consolas de operador.
8	`PCB1`	**Placa de Circuito Impreso FR-4 Doble Capa (`MECATROBOTICA-EDGE-HAT-V07`)** con cabezal hembra de 2×20 pines y bornera `KF301`	1	Montaje definitivo por soldadura libre de falsos contactos para operación continua $24/7$.
9	`R1..R5, D1..D2`	Resistencias de película metálica ($1\ k\Omega$, $10\ k\Omega$, $2\times\ 330\ \Omega$) + Diodos LED de $3\ mm$ Verde y Ámbar + Capacitor `100 nF X7R`	1 kit	Polarización del optoacoplador `PC817` , indicadores visuales de estado de ingesta (`GPIO 27` y `GPIO 22`) y desacoplo local.

---

2. Cálculo de Ingeniería: Presupuesto de Almacenamiento y Factor de Compresión

Para dimensionar científicamente la unidad SSD externa, calculamos el volumen bruto diario de telemetría $D_{día}$ generado por una planta con $N_{nodos} = 20$ nodos enviando tramas de $B_{muestra} = 150\ Bytes$ a una frecuencia de muestreo $f_s = 1.0\ Hz$:

$$D_{día} = N_{nodos} \cdot B_{muestra} \cdot f_s \cdot 86\,400\ s/día = 20 \times 150 \times 1.0 \times 86\,400 = 259.2\ MB/día$$

Gracias al motor **TSM (Time-Structured Merge Tree)** de InfluxDB 2.7 (que aplica codificación *Delta-of-Delta* en marcas de tiempo y compresión XOR *Gorilla* en punto flotante), el volumen físico en disco se reduce aproximadamente un factor de $6.8\times$, ocupando solo $\approx 38\ MB/día$ ($\approx 1.14\ GB$ para la ventana de retención cruda de $30\ días$).

Al ejecutar nuestra tarea automática de **Downsampling** en lenguaje `Flux` hacia ventanas de $\Delta t_{agregado} = 5\ minutos = 300\ s$, el factor de reducción temporal es:

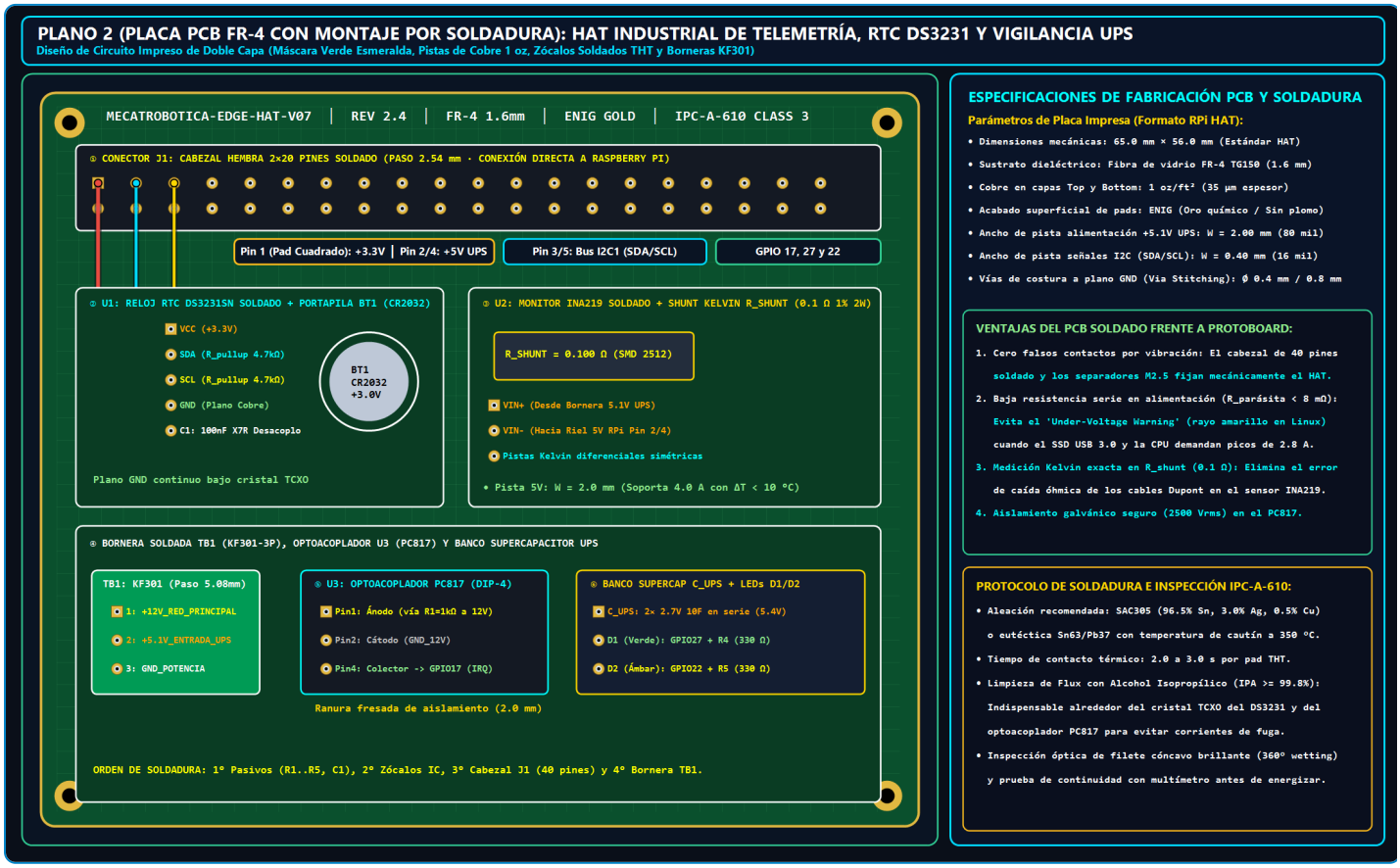
$$R_{comp} = \frac{\Delta t_{agregado}}{\Delta t_{crudo}} = (300\ s) / (1\ s) = 300 : 1$$

Por consiguiente, un año completo ($365\ días$) de historial analítico resumido (media, mínimo y máximo cada $5\ minutos$) requiere menos de $50\ MB$ de espacio en disco, garantizando consultas gráficas instantáneas en **Grafana**.

---

3. Comparativa de Montaje: Protoboard (Tinkercad) vs. Placa Impresa Soldada (PCB HAT)

📐 Plano 2: Placa PCB FR-4 con Montaje por Soldadura



MANUAL TÉCNICO DE INGENIERÍA — MÓDULO 07

Arquitectura de Contenedores Docker en ARM64, Series Temporales en InfluxDB 2.7, Resiliencia Store-and-Forward y Ergonomía Visual ISA-101

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

---

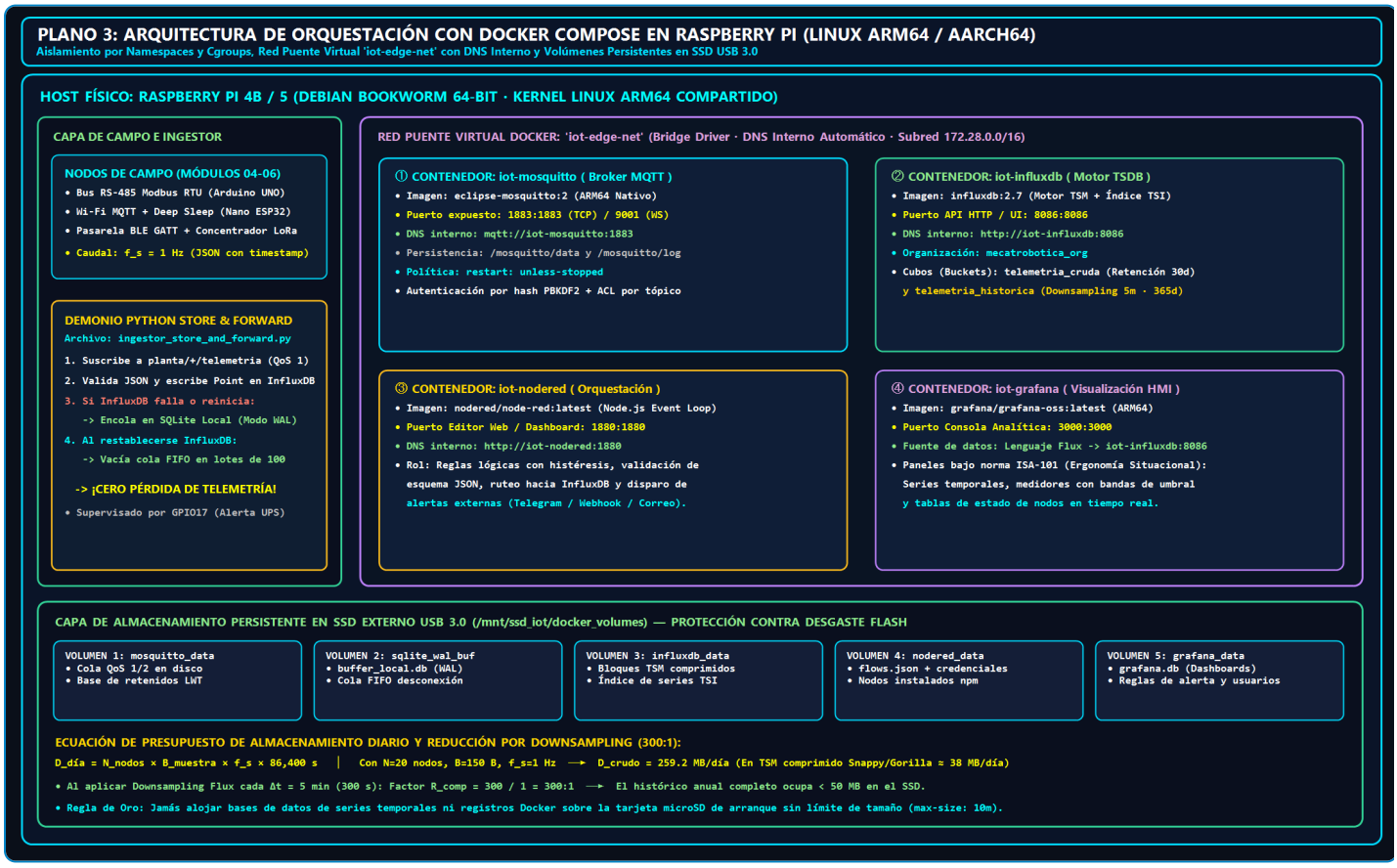
1. Fundamentos de Contenedorización con Docker en Linux ARM64

En despliegues industriales de borde (*Edge Computing*), instalar manualmente demonios de bases de datos, intérpretes de Node.js y servidores web directamente sobre el sistema operativo anfitrión conduce a conflictos de bibliotecas dinámicas y dificulta la recuperación ante desastres.

Mediante **Docker Engine** y **Docker Compose** sobre arquitecturas `ARM64 (aarch64)`, cada servicio opera aislado mediante dos primitivas nativas del kernel de Linux (sin la sobrecarga de emular hardware de una máquina virtual):

- **Espacios de nombres (`Namespaces`)**: Aíslan el árbol de procesos (`PID`), la pila de red (`NET`) y los puntos de montaje (`MNT`) de cada contenedor.
- **Grupos de control (`Cgroups v2`)**: Limitan y auditan el uso de memoria RAM y ciclos de CPU por servicio.
- **Red puente definida por el usuario (`iot-edge-net`)**: Provee resolución DNS interna automática por nombre de servicio (`iot-mosquitto`, `iot-influxdb`, `iot-nodered`, `iot-grafana`), evitando depender de direcciones IP estáticas frágiles.

📐 Plano 3: Arquitectura Docker Compose ARM64



---

2. Bases de Datos Relacionales (SQL) frente a Series Temporales (TSDB)

Una señal de telemetría mecatrónica es una sucesión discreta ordenada cronológicamente:

$$S = \{(t_1, \mathbf{v}_1), (t_2, \mathbf{v}_2), \dots, (t_k, \mathbf{v}_k)\} \quad \text{con } t_1 < t_2 < \dots < t_k$$

2.1. Anatomía del Protocolo de Línea (*Line Protocol*) en InfluxDB

Cada registro en InfluxDB se compone de cuatro elementos:

```
ambiente,nodo=invernadero_01,ubicacion=sector_norte temperatura=24.65,humedad=61.2,pwm_ventilador=140i
1730505600000000000
```

- **Medición (`Measurement`)**: Contenedor lógico de la magnitud (`ambiente`).
- **Conjunto de Etiquetas (`Tag Set`)**: Pares clave-valor de metadatos **indexados en memoria RAM** (`nodo`, `ubicacion`). Se utilizan en cláusulas de filtrado rápido.
- **Conjunto de Campos (`Field Set`)**: Valores numéricos variables **no indexados** (`temperatura`, `humedad`, `pwm_ventilador`).
- **Marca de Tiempo (`Timestamp`)**: Entero de 64 bits en nanosegundos UTC desde la época Unix.

2.2. Control Matemático de la Cardinalidad de Series

La cardinalidad total de series C_{series} en un cubo de InfluxDB viene dada por el producto cartesiano de los valores únicos de cada etiqueta:

$$C_{series} = |M| \times \prod_{i=1}^{m} |Tag_i| \times |Field_{keys}|$$

> **Regla Crítica de Diseño**: Jamás almacenes valores continuos (temperaturas con decimales), marcas de tiempo ni identificadores únicos por mensaje (`UUID`) dentro de un **Tag**. Hacerlo provoca una explosión de cardinalidad (*Runaway Cardinality*) que agota la memoria RAM de la Raspberry Pi (`OOM Kill`). Los valores variables van siempre en **Fields**.

---

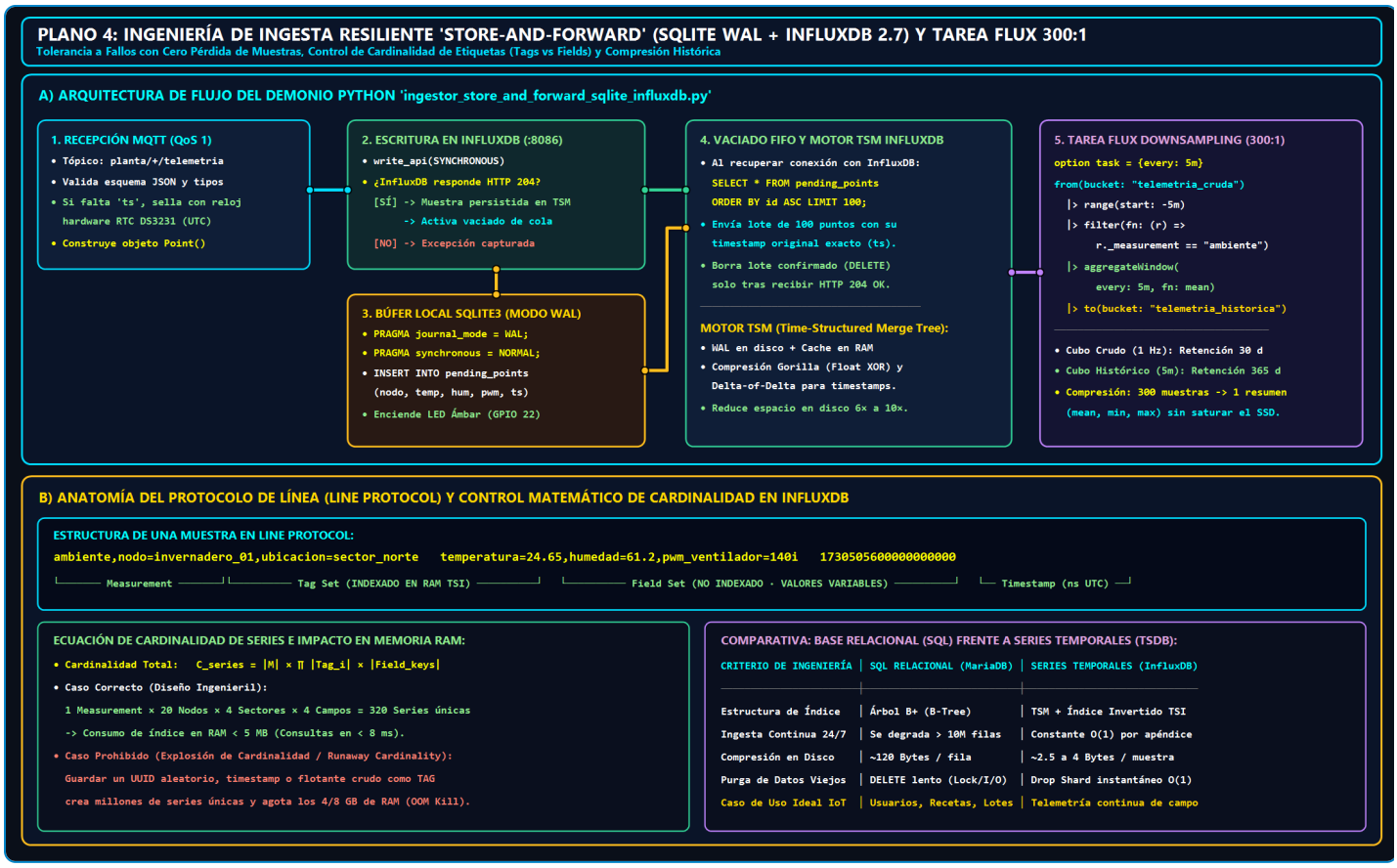
3. Patrón de Resiliencia *Store-and-Forward* con SQLite3 en Modo WAL

Para alcanzar una pérdida de datos igual a cero ($\text{Data Loss} = 0\%$) cuando el contenedor de InfluxDB se reinicia por mantenimiento o el enlace hacia un servidor central cae, nuestro demonio en Python implementa un búfer local en **SQLite3** activando la directiva:

```
PRAGMA journal_mode = WAL;
PRAGMA synchronous = NORMAL;
```

El modo **Write-Ahead Logging (`WAL`)** escribe las transacciones entrantes en un archivo secuencial de apéndice (`buffer_local_wal.db-wal`) sin bloquear las lecturas concurrentes y reduciendo drásticamente las operaciones de escritura aleatoria sobre las celdas Flash del SSD.

📐 **Plano 4: Flujo Store-and-Forward y Tarea Flux 300:1**



---

4. Ergonomía Visual bajo Norma ANSI/ISA-101 en Grafana

Un panel de control industrial no se evalúa por cuántos colores brillantes exhibe, sino por la velocidad con la que un operador humano detecta una condición anormal a las tres de la madrugada:

- **Contexto neutro de bajo contraste**: Fondos oscuros mate (`#0A101C` y `#121E36`) que reducen el deslumbramiento ocular.
- **Reserva cromática para anomalías**: Las curvas normales utilizan tonos cian o verde esmeralda moderados; el color **ámbar (`#FFBE1E`)** indica advertencia preventiva y el color **rojo (`#FF5050`)** se reserva exclusivamente para alarmas críticas activas.
- **Jerarquía de tres niveles**: Nivel 1 (KPIs globales y estado de salud), Nivel 2 (tendencias por subsistema con bandas sombreadas) y Nivel 3 (diagnóstico detallado de tramas, RSSI y latencias).

📐 Plano 5: Ergonomía Visual ISA-101 y Flujo Node-RED

PLANO 5: DISEÑO DE INTERFACES HMI BAJO NORMA ISA-101 EN GRAFANA Y FLUJO DE REGLAS EN NODE-RED

Conciencia Situacional para Operadores de Planta, Jerarquía de 3 Niveles, Histéresis de Alerta y Disponibilidad Operativa (SLA)

A) PANEL DE SUPERVISIÓN INDUSTRIAL GRAFANA (:3000) DISEÑADO BAJO ESTÁNDAR ANSI/ISA-101.01

TEMPERATURA PROCESO (NODO 01)	HUMEDAD RELATIVA Y PRESIÓN	PRE-ALARMA PREVENTIVA (ADVERTENCIA)	ALARMA CRÍTICA ISA-18.2 (PRIORIDAD 1)
24.6 °C	61.2 %RH | 1013 hPa	29.4 °C (Sector Sur)	0 ALARMAS ACTIVAS
Banda Nominal: [18.0 .. 28.0 °C] (OK)	Sensor Bosch BME280 (I2C 0x76)	Supera umbral preventivo > 28.0 °C	Disponibilidad SLA: A = 99.982% (Online)

CURVA DE SERIE TEMPORAL CON BANDA OPERATIVA SOMBREADA (CONSULTA FLUX aggregateWindow)



REGLAS DE ORO DE ERGONOMÍA VISUAL ISA-101:

1. Prohibido el efecto 'Árbol de Navidad': Si todo brilla en rojo, verde y amarillo saturado, el operador sufre fatiga visual y no detecta fallos.
2. Fondos oscuros mate (#0A101C / #121E36) y rejillas de bajo contraste para descanso ocular en turno 24/7.
3. Reserva cromática estricta:
   - Cian / Esmeralda -> Operación dentro de consigna
   - Ámbar (#FFBE1E) -> Desviación preventiva
   - Rojo (#FF5050) -> Alarma crítica que exige acción
4. Incluir siempre unidades físicas (°C, %RH, mA, dBm).

B) FLUJO DE ORQUESTACIÓN EN NODE-RED (:1880) CON HISTÉRESIS DE ALARMA Y CÁLCULO DE DISPONIBILIDAD (SLA)

① MQTT In
planta/+/telemetria
(QoS 1 · JSON)

② JSON Parser &
Schema Validator
(Verifica llaves)

③ Function Node
Filtro Histéresis
(T_on=30, T_off=28.5)

④ RBE / Filter
Report by Exception
(Bloquea spam)

⑤ Salida Dual:
• InfluxDB Batch Out
• Alerta Telegram/MQTT

LÓGICA JAVASCRIPT CON MEMORIA DE CONTEXTO E HISTÉRESIS EN NODE-RED:

```
const T_ALARMA_ON  = 30.0; // Umbral de disparo (°C)
const T_ALARMA_OFF = 28.5; // Umbral de reposición (Banda ΔT = 1.5 °C)
let enAlarma = context.get('enAlarma') || false;
if (!enAlarma && msg.payload.temp >= T_ALARMA_ON) {
    enAlarma = true;  context.set('enAlarma', true);
    return [msg, { payload: { alerta: 'ALTA_TEMP', valor: msg.payload.temp } }];
} else if (enAlarma && msg.payload.temp <= T_ALARMA_OFF) {
    enAlarma = false; context.set('enAlarma', false);
}
```

INGENIERÍA DE CONFIABILIDAD Y DISPONIBILIDAD DEL SISTEMA (MTBF / MTTR):

1. Ecuación de Disponibilidad Estacionaria (Availability):
   A = MTBF / (MTBF + MTTR) = T_operativo / T_total

2. Efecto de Docker 'restart: unless-stopped' + Healthchecks:
   Reduce el Tiempo Medio de Reparación (MTTR) de horas (intervención humana manual) a menos de 3.5 segundos (reinicio automático).

3. Combinado con el búfer SQLite WAL (Store-and-Forward):
   La disponibilidad efectiva de datos alcanza el 100.0% frente a microcortes de red o reinicios de mantenimiento de contenedores.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 07

Planos en Protoboard (Estilo Tinkercad), Placa PCB FR-4 con Montaje por Soldadura y Diagramas de Arquitectura

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

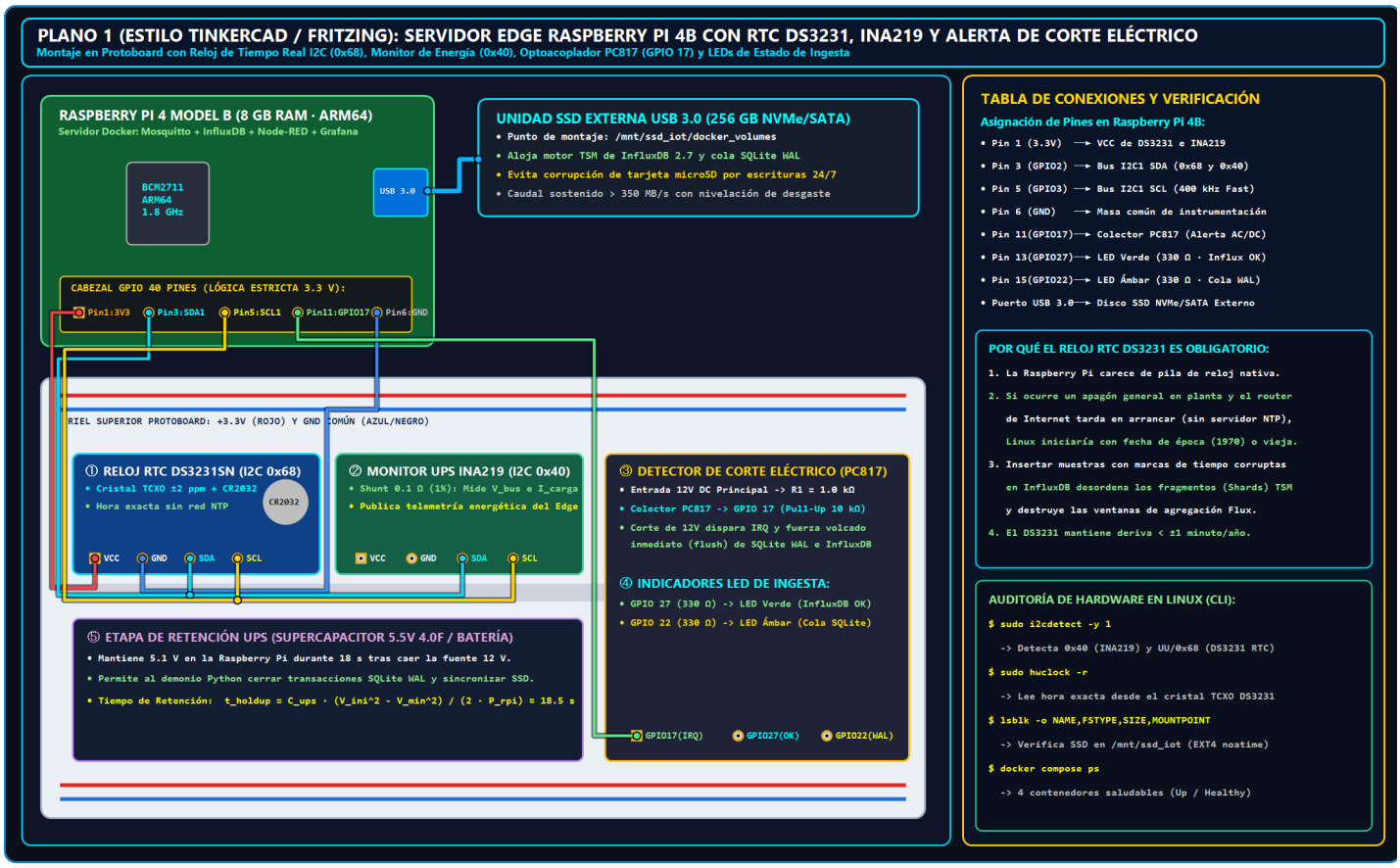
Módulo: 07 — Persistencia en Series Temporales, Docker, Node-RED, InfluxDB y Grafana

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje en Protoboard con RTC DS3231, INA219 y Alerta de Corte Eléctrico

En este primer plano de alta legibilidad se detalla el conexionado sobre protoboard del servidor Edge **Raspberry Pi 4B** acoplado a una unidad **SSD USB 3.0**, un reloj de tiempo real de alta precisión **DS3231SN (** `0x68` **)**, un monitor de potencia **INA219 (** `0x40` **)** y un detector optoacoplado de caída de alimentación principal **PC817** conectado al pin `GPIO 17`.

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad



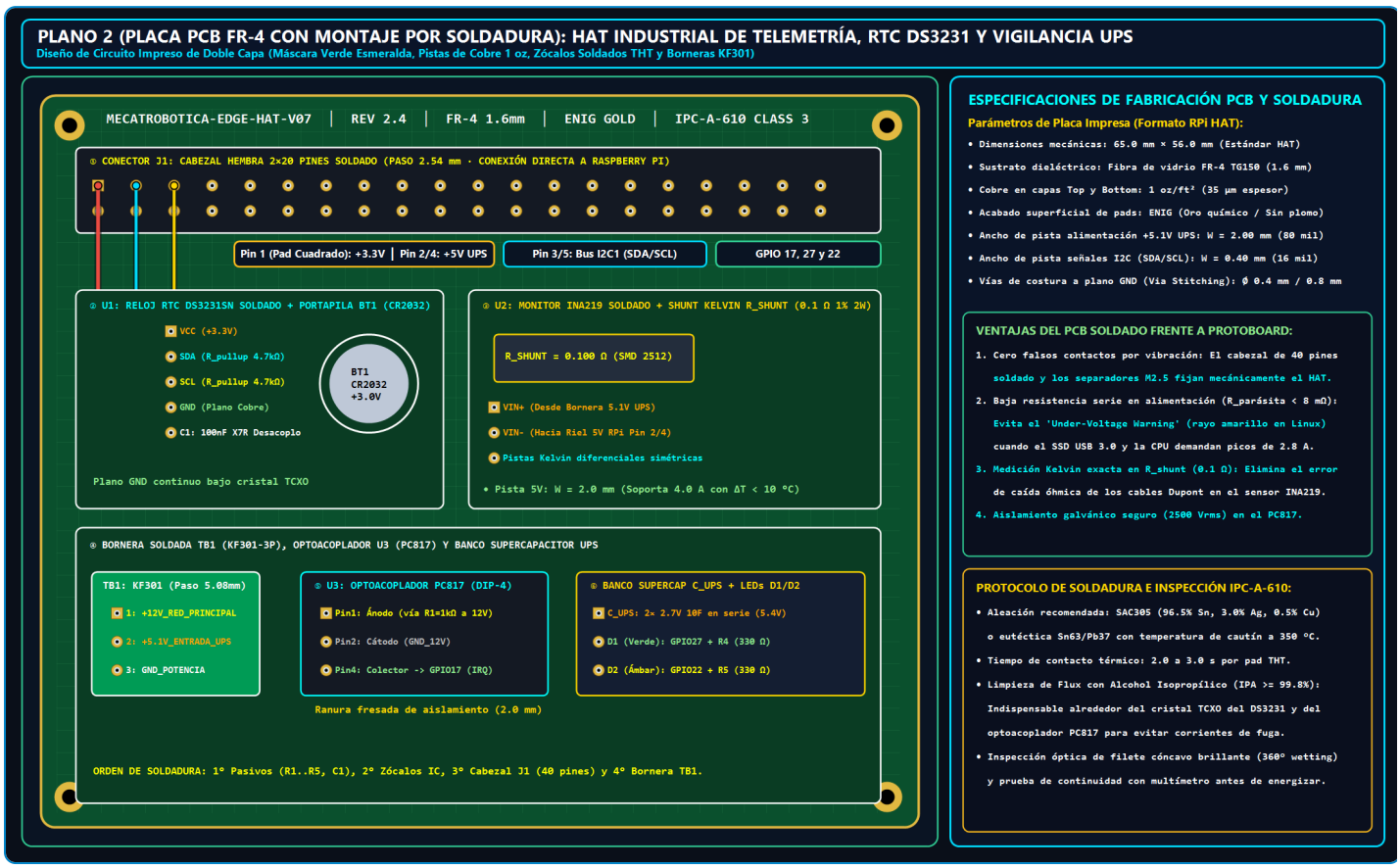
---

2. Plano 2 (Placa de Circuito Impreso PCB FR-4): Montaje por Soldadura de HAT Industrial para Raspberry Pi

Para servidores que operan las 24 horas de los 365 días del año en gabinetes sometidos a vibraciones de maquinaria, los cables puente Dupont sobre protoboard presentan riesgo de desconexión y caída óhmica en el riel de $5.1\ V$. Este segundo plano presenta el diseño de una **placa de circuito impreso de doble capa en fibra de vidrio FR-4 (** `MECATROBOTICA-EDGE-HAT-V07` **)** con montaje por soldadura (`IPC-A-610 Clase 3`):

- **Conector `J1` (2×20 pines hembra paso $2.54\ mm$ soldado)**: Acopla rígidamente el HAT sobre la Raspberry Pi junto con cuatro separadores mecánicos `M2.5`.
- **Pistas de potencia de $2.0\ mm$ de ancho y medición Kelvin en `R_SHUNT` ($0.100\ \Omega$)**: Soportan picos de corriente de hasta $4.0\ A$ con una caída resistiva despreciable.
- **Bornera de tornillo `TB1 (KF301)` soldada y ranura de aislamiento bajo el optoacoplador `PC817`**: Separan físicamente el riel industrial de $12\ V\ DC$ de la lógica de $3.3\ V$ del procesador Broadcom.

📐 Plano 2: Placa PCB FR-4 con Montaje por Soldadura

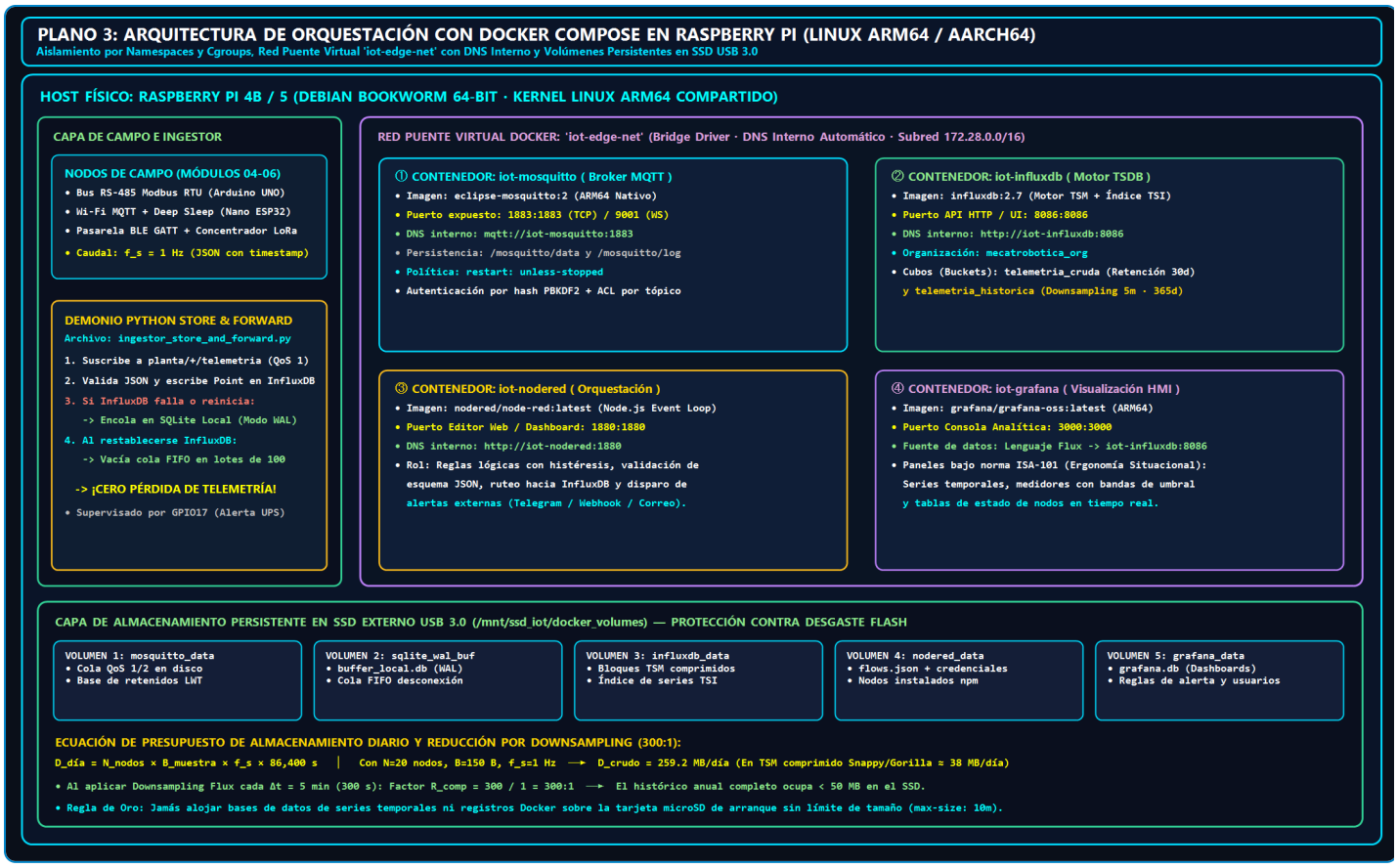


---

3. Plano 3: Arquitectura de Orquestación con Docker Compose en Linux ARM64

Representación detallada de la red puente aislada `iot-edge-net`, el mapeo de puertos de red (`1883`, `8086`, `1880` y `3000`) y la vinculación de los cinco volúmenes persistentes sobre la unidad externa **SSD USB 3.0** (`/mnt/ssd_iot/docker_volumes`).

📐 Plano 3: Arquitectura Docker Compose ARM64

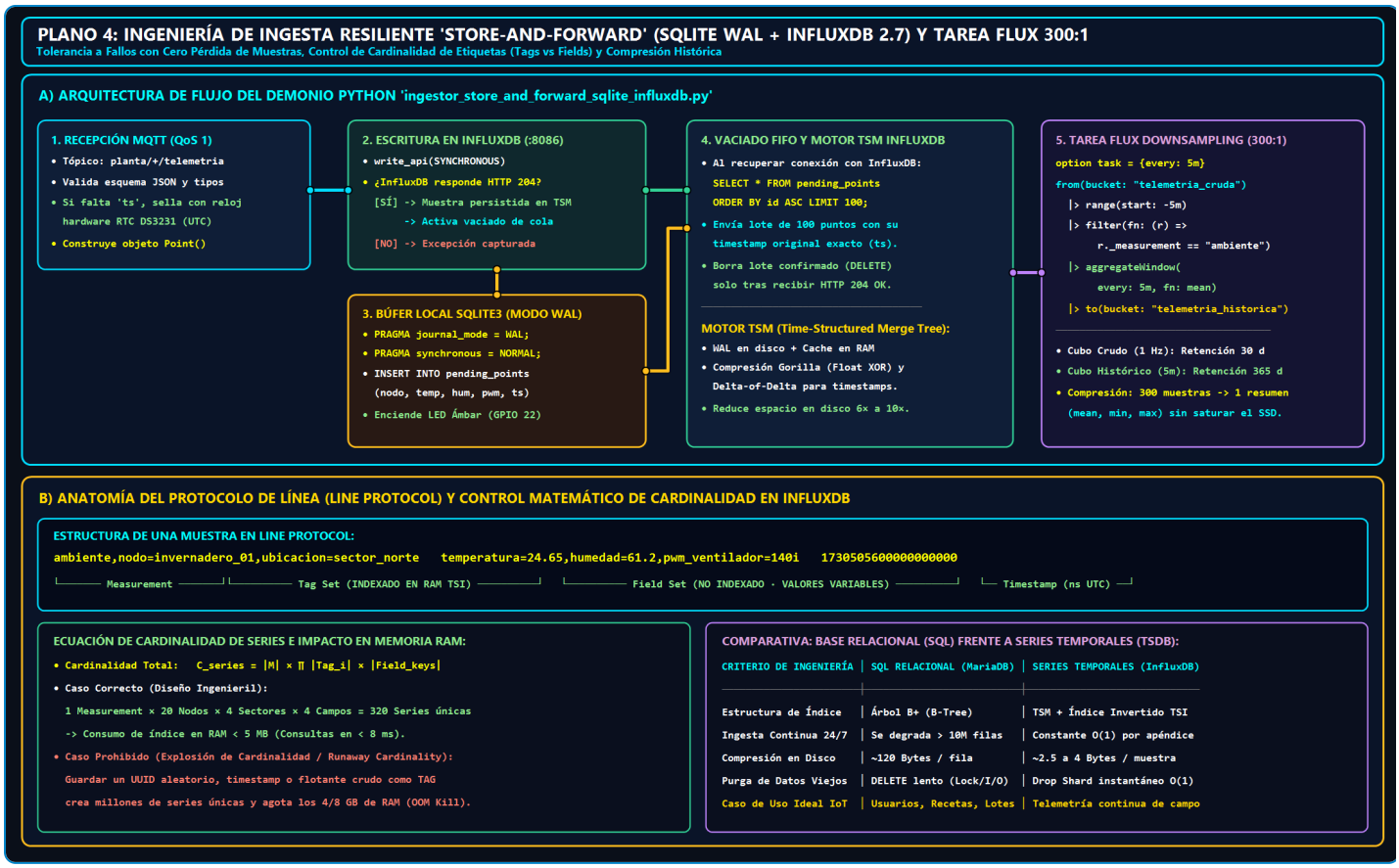


---

4. Plano 4: Flujo Store-and-Forward (SQLite WAL + InfluxDB 2.7) y Tarea Flux 300:1

Diagrama de ingeniería del demonio suscriptor en Python con conmutación automática a cola `SQLite3 WAL` ante caídas de red, recuperación por lotes FIFO (`LIMIT 100`), anatomía del *Line Protocol* y control de cardinalidad.

📐 Plano 4: Flujo Store-and-Forward y Tarea Flux 300:1



---

5. Plano 5: Ergonomía Visual ISA-101 en Grafana y Flujo de Reglas con Histéresis en Node-RED

Diseño de paneles de alto desempeño con banda operativa sombreada, semántica cromática estricta (cian/esmeralda nominal, ámbar preventivo y rojo crítico) y máquina de estados con histéresis en Node-RED.

📐 Plano 5: Ergonomía Visual ISA-101 y Flujo Node-RED

PLANO 5: DISEÑO DE INTERFACES HMI BAJO NORMA ISA-101 EN GRAFANA Y FLUJO DE REGLAS EN NODE-RED

Conciencia Situacional para Operadores de Planta, Jerarquía de 3 Niveles, Histéresis de Alerta y Disponibilidad Operativa (SLA)

A) PANEL DE SUPERVISIÓN INDUSTRIAL GRAFANA (:3000) DISEÑADO BAJO ESTÁNDAR ANSI/ISA-101.01

TEMPERATURA PROCESO (NODO 01)

24.6 °C

Banda Nominal: [18.0 .. 28.0 °C] (OK)

HUMEDAD RELATIVA Y PRESIÓN

61.2 %RH | 1013 hPa

Sensor Bosch BME280 (I2C 0x76)

PRE-ALARMA PREVENTIVA (ADVERTENCIA)

29.4 °C (Sector Sur)

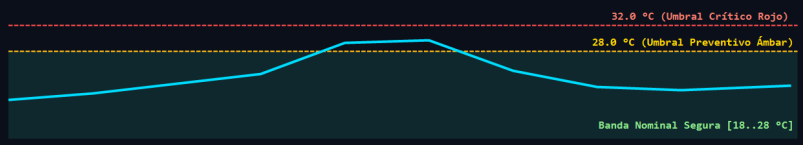
Supera umbral preventivo > 28.0 °C

ALARMA CRÍTICA ISA-18.2 (PRIORIDAD 1)

0 ALARMAS ACTIVAS

Disponibilidad SLA: A = 99.982% (Online)

CURVA DE SERIE TEMPORAL CON BANDA OPERATIVA SOMBREADA (CONSULTA FLUX aggregateWindow)



REGLAS DE ORO DE ERGONOMÍA VISUAL ISA-101:

1. Prohibido el efecto 'Árbol de Navidad': Si todo brilla en rojo, verde y amarillo saturado, el operador sufre fatiga visual y no detecta fallos.
2. Fondos oscuros mate (#0A101C / #121E36) y rejillas de bajo contraste para descanso ocular en turno 24/7.
3. Reserva cromática estricta:
   - Cian / Esmeralda -> Operación dentro de consigna
   - Ámbar (#FFBE1E) -> Desviación preventiva
   - Rojo (#FF5050) -> Alarma crítica que exige acción
4. Incluir siempre unidades físicas (°C, %RH, mA, dBm).

B) FLUJO DE ORQUESTACIÓN EN NODE-RED (:1880) CON HISTÉRESIS DE ALARMA Y CÁLCULO DE DISPONIBILIDAD (SLA)

① MQTT In
planta/+/telemetria
(QoS 1 · JSON)

② JSON Parser &
Schema Validator
(Verifica llaves)

③ Function Node
Filtro Histéresis
(T_on=30, T_off=28.5)

④ RBE / Filter
Report by Exception
(Bloquea spam)

⑤ Salida Dual:
• InfluxDB Batch Out
• Alerta Telegram/MQTT

LÓGICA JAVASCRIPT CON MEMORIA DE CONTEXTO E HISTÉRESIS EN NODE-RED:

```
const T_ALARMA_ON  = 30.0; // Umbral de disparo (°C)
const T_ALARMA_OFF = 28.5; // Umbral de reposición (Banda ΔT = 1.5 °C)
let enAlarma = context.get('enAlarma') || false;
if (!enAlarma && msg.payload.temp >= T_ALARMA_ON) {
    enAlarma = true;  context.set('enAlarma', true);
    return [msg, { payload: { alerta: 'ALTA_TEMP', valor: msg.payload.temp } }];
} else if (enAlarma && msg.payload.temp <= T_ALARMA_OFF) {
    enAlarma = false; context.set('enAlarma', false);
}
```

INGENIERÍA DE CONFIABILIDAD Y DISPONIBILIDAD DEL SISTEMA (MTBF / MTTR):

1. Ecuación de Disponibilidad Estacionaria (Availability):
   A = MTBF / (MTBF + MTTR) = T_operativo / T_total

2. Efecto de Docker 'restart: unless-stopped' + Healthchecks:
   Reduce el Tiempo Medio de Reparación (MTTR) de horas (intervención humana manual) a menos de 3.5 segundos (reinicio automático).

3. Combinado con el búfer SQLite WAL (Store-and-Forward):
   La disponibilidad efectiva de datos alcanza el 100.0% frente a microcortes de red o reinicios de mantenimiento de contenedores.

Capítulo 23 · Series temporales: InfluxDB, Flux y Grafana

Conceptos clave

Define campos numéricos, etiquetas de cardinalidad controlada y retención. Verifica el intervalo y la agregación del panel con datos conocidos.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 07

Persistencia en Series Temporales, Orquestación con Docker Compose, Node-RED, InfluxDB y Grafana

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 07 — Capa de Almacenamiento Histórico, Contenedores ARM64 y Observabilidad Industrial

---

1. Objetivos de Ingeniería del Laboratorio

- Montar una unidad de estado sólido externa (`SSD USB 3.0`) en la Raspberry Pi y configurar el reloj de tiempo real por hardware (`DS3231` en el bus `I2C1` dirección `0x68`) junto con el detector optoacoplado de corte eléctrico (`PC817` en `GPIO 17`).
- Desplegar mediante `Docker Compose` una arquitectura multicontenedor aislada sobre Linux ARM64 compuesta por `Eclipse Mosquitto 2` , `InfluxDB 2.7` , `Node-RED` y `Grafana OSS` .
- Programar y auditar el demonio en Python de ingesta resiliente **Store-and-Forward** (`ingestor_store_and_forward_sqlite_influxdb.py`) que respalda muestras en una cola `SQLite3` en modo `WAL` cuando la base de datos principal se encuentra en mantenimiento o desconectada.
- Configurar políticas de retención de datos (`30 días` a `1 Hz`) y tareas automáticas de agregación o *Downsampling* en lenguaje `Flux` (`300:1` hacia ventanas de `5 minutos` con retención de `365 días`).
- Construir paneles de supervisión industrial en `Grafana` cumpliendo la norma de ergonomía visual **ANSI/ISA-101** y flujos de alerta con histéresis en **Node-RED**.

---

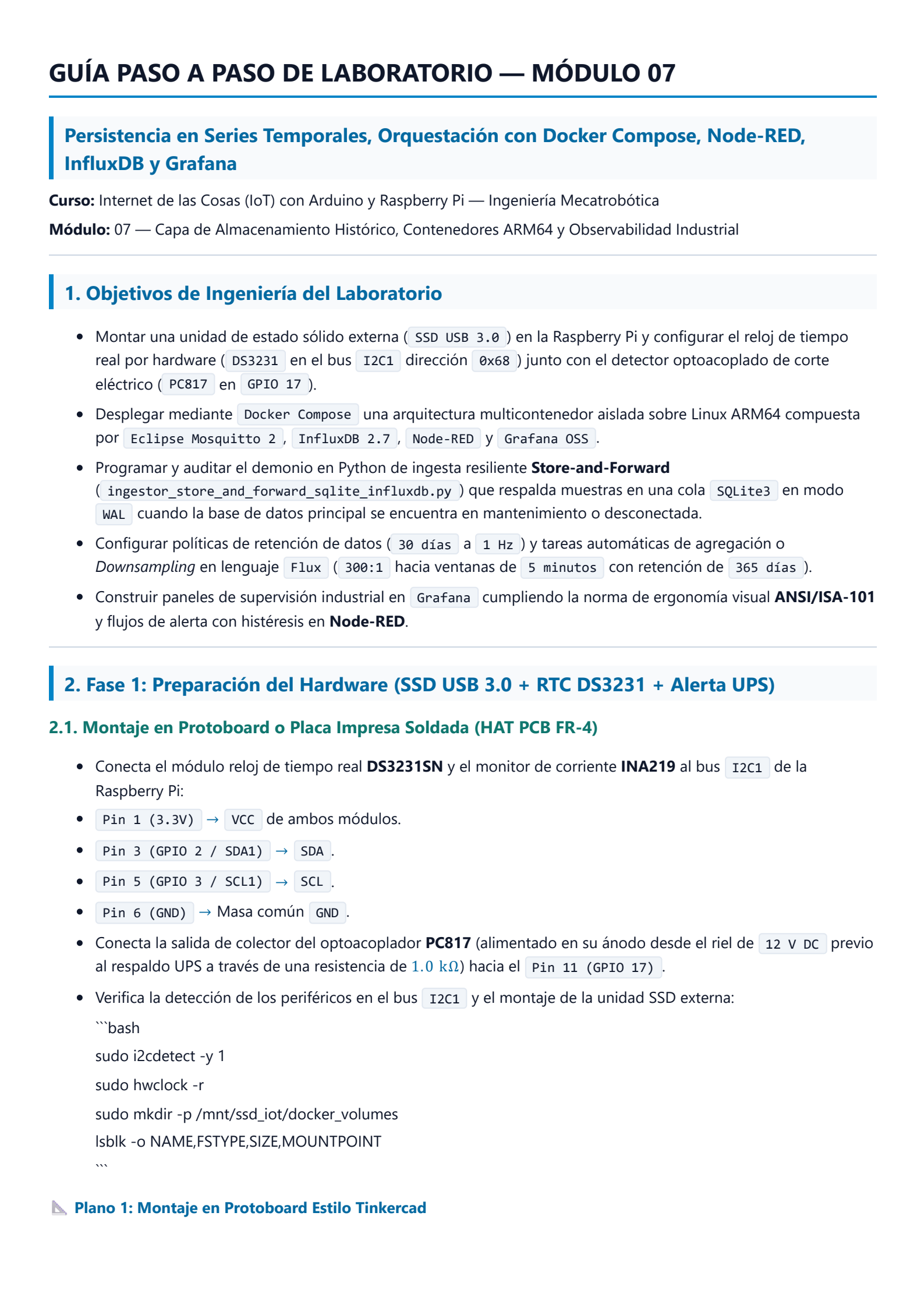
2. Fase 1: Preparación del Hardware (SSD USB 3.0 + RTC DS3231 + Alerta UPS)

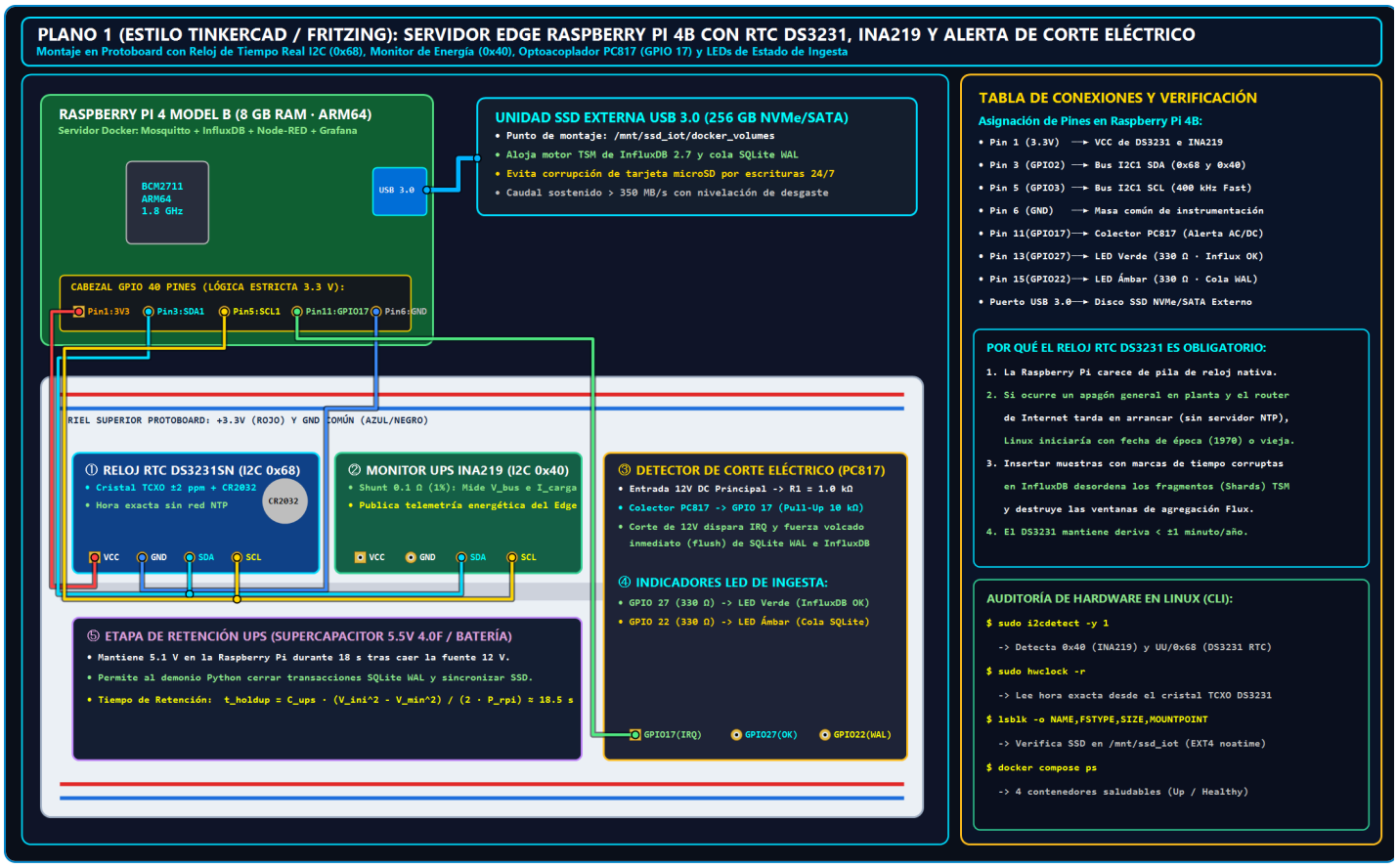
2.1. Montaje en Protoboard o Placa Impresa Soldada (HAT PCB FR-4)

- Conecta el módulo reloj de tiempo real **DS3231SN** y el monitor de corriente **INA219** al bus `I2C1` de la Raspberry Pi:
- `Pin 1 (3.3V)` → `VCC` de ambos módulos.
- `Pin 3 (GPIO 2 / SDA1)` → `SDA` .
- `Pin 5 (GPIO 3 / SCL1)` → `SCL` .
- `Pin 6 (GND)` → Masa común `GND` .
- Conecta la salida de colector del optoacoplador **PC817** (alimentado en su ánodo desde el riel de `12 V DC` previo al respaldo UPS a través de una resistencia de $1.0\ \text{k}\Omega$) hacia el `Pin 11 (GPIO 17)` .
- Verifica la detección de los periféricos en el bus `I2C1` y el montaje de la unidad SSD externa:

  ```bash
  sudo i2cdetect -y 1
  sudo hwclock -r
  sudo mkdir -p /mnt/ssd_iot/docker_volumes
  lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT
  ```

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad





---

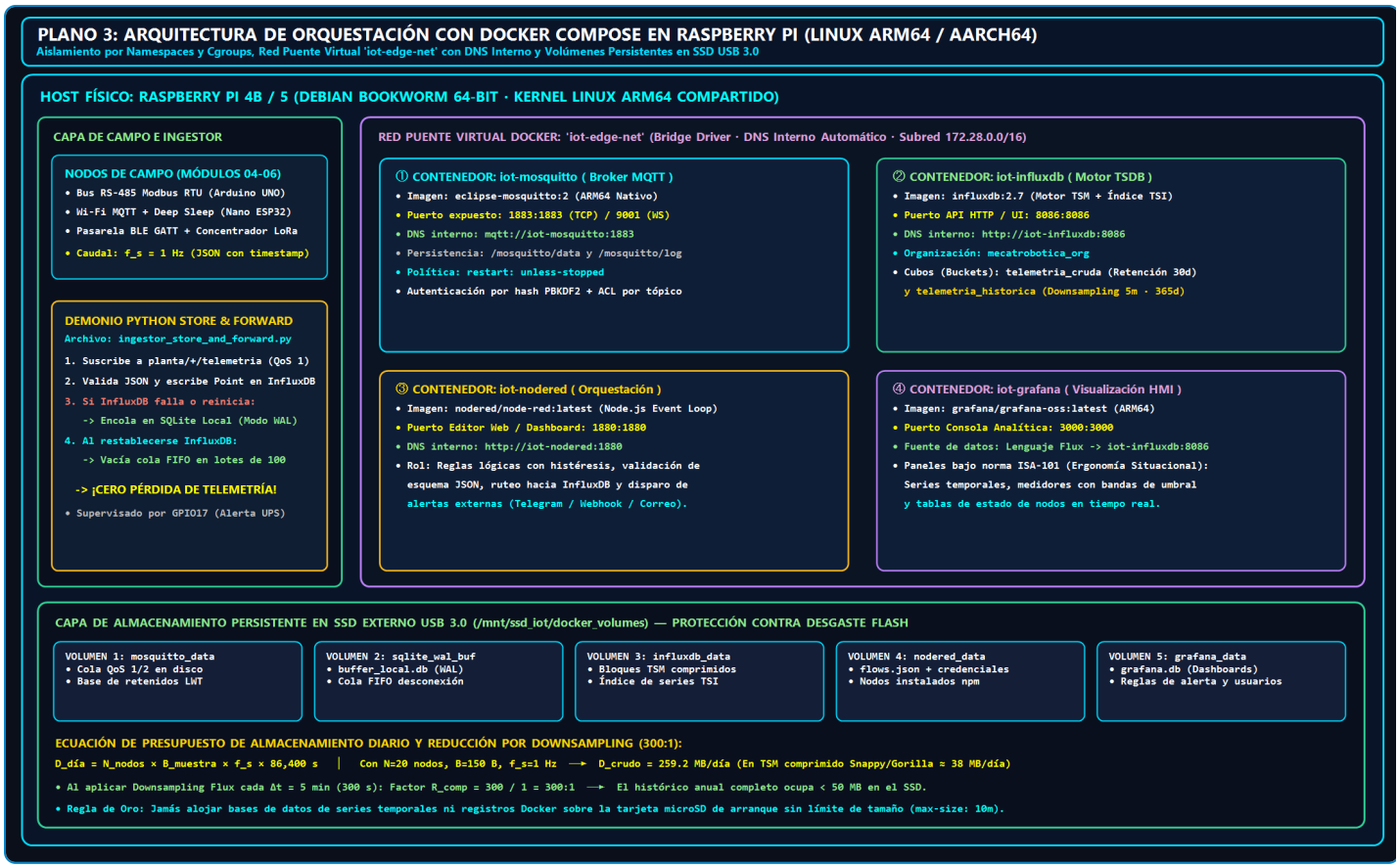
3. Fase 2: Despliegue de la Pila con Docker Compose en ARM64

- Copia el archivo `docker-compose.yml` incluido en la carpeta `codigo_fuente/` hacia `/opt/mecatrobotica/stack_v07/docker-compose.yml`.
- Levanta los cuatro servicios en segundo plano y verifica su estado de salud:

  ```bash
  cd /opt/mecatrobotica/stack_v07
  docker compose up -d
  docker compose ps
  ```

- Comprueba que los cuatro contenedores (`iot-mosquitto`, `iot-influxdb`, `iot-nodered` e `iot-grafana`) reporten estado `Up` y que `iot-influxdb` alcance el estado `healthy` en el puerto `8086`.

📐 Plano 3: Arquitectura Docker Compose en Raspberry Pi ARM64



---

4. Fase 3: Prueba de Fuego del Ingestor Store-and-Forward (SQLite WAL + InfluxDB)

- Instala las bibliotecas cliente en el entorno virtual de la Raspberry Pi:

  ```bash
  pip install paho-mqtt influxdb-client
  ```

- Ejecuta el demonio `ingestor_store_and_forward_sqlite_influxdb.py`.
- **Simulación de caída de base de datos**: Detén temporalmente el contenedor de InfluxDB mientras los nodos continúan publicando telemetría:

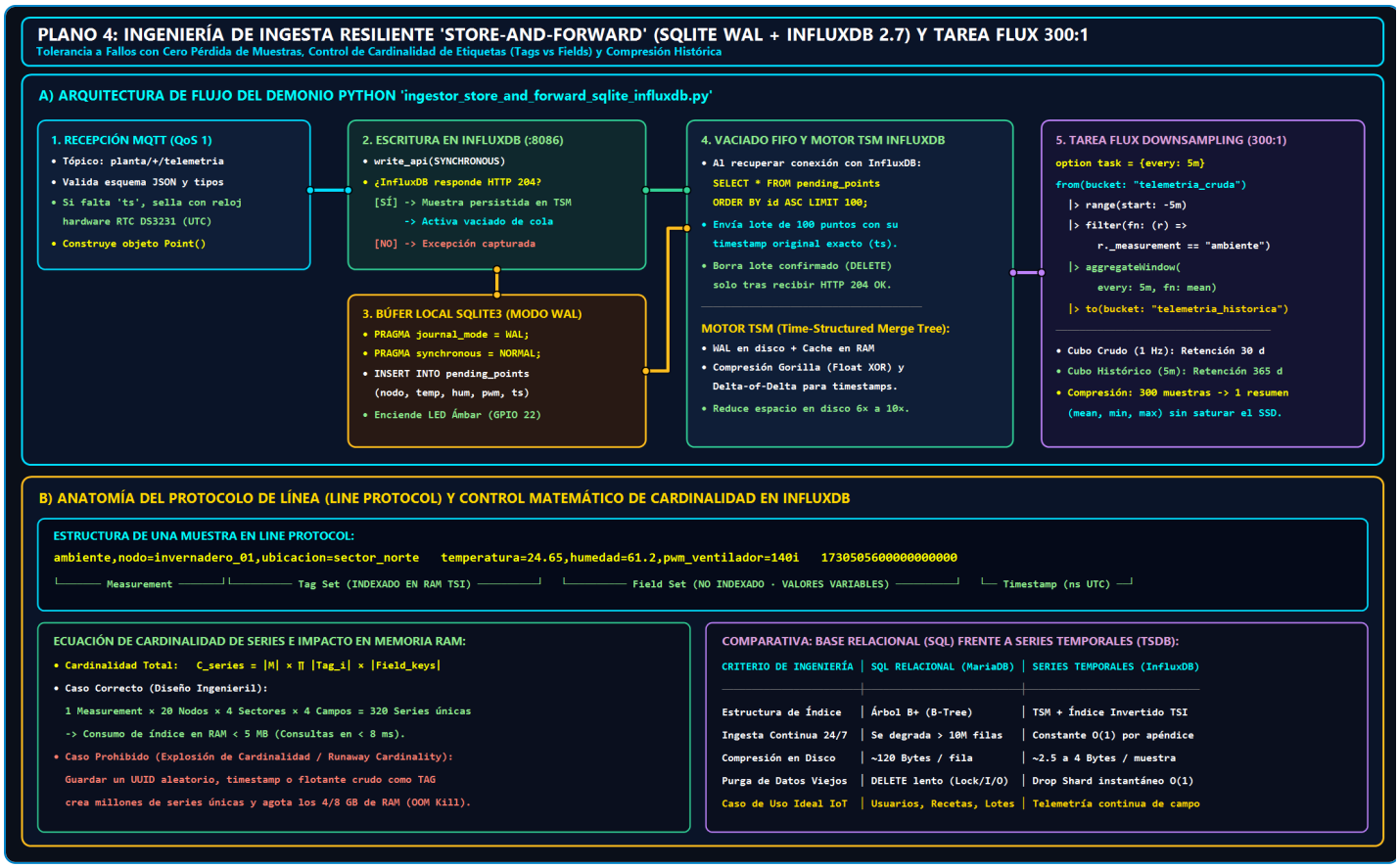
  ```bash
  docker stop iot-influxdb
  ```

- Observa en la consola cómo el demonio captura la excepción de conexión y desvía inmediatamente cada muestra entrante hacia la tabla `pending_points` de `buffer_local_wal.db` (`PRAGMA journal_mode=WAL`).
- **Restauración y vaciado por lotes**: Vuelve a iniciar el contenedor:

  ```bash
  docker start iot-influxdb
  ```

- Verifica cómo en el siguiente ciclo de recepción el demonio vacía automáticamente la cola FIFO en lotes de `100 muestras` (`ORDER BY id ASC LIMIT 100`), insertando toda la serie histórica con sus marcas de tiempo originales intactas.

📐 Plano 4: Arquitectura Store-and-Forward y Tarea Flux 300:1



5. Fase 4: Tarea de Downsampling en Flux y Paneles ISA-101 en Grafana

- Accede a la interfaz web de InfluxDB (`http://<IP_RASPBERRY>:8086`), crea el segundo cubo llamado `telemetria_historica` con retención de `365 días` y registra el script `tarea_downsampling_flux_300a1.flux` en la sección **Tasks**.
- Accede a **Node-RED** (`http://<IP_RASPBERRY>:1880`), importa el archivo `flujo_orquestacion_alertas_nodered.json` y despliega el flujo con validación de esquema JSON e histéresis térmica ($T_{on} = 30.0\ ^\circ C$, $T_{off} = 28.5\ ^\circ C$).
- Accede a **Grafana** (`http://<IP_RASPBERRY>:3000`), configura la fuente de datos **InfluxDB** seleccionando el lenguaje **Flux**, la URL interna `http://iot-influxdb:8086` y el token de administración.
- Diseña el panel de supervisión aplicando la jerarquía de tres niveles y la semántica de colores de la norma **ANSI/ISA-101**.

📐 Plano 5: Ergonomía Visual ISA-101 en Grafana y Flujo Node-RED

PLANO 5: DISEÑO DE INTERFACES HMI BAJO NORMA ISA-101 EN GRAFANA Y FLUJO DE REGLAS EN NODE-RED

Conciencia Situacional para Operadores de Planta, Jerarquía de 3 Niveles, Histéresis de Alerta y Disponibilidad Operativa (SLA)

A) PANEL DE SUPERVISIÓN INDUSTRIAL GRAFANA (:3000) DISEÑADO BAJO ESTÁNDAR ANSI/ISA-101.01

TEMPERATURA PROCESO (NODO 01)

24.6 °C

Banda Nominal: [18.0 .. 28.0 °C] (OK)

HUMEDAD RELATIVA Y PRESIÓN

61.2 %RH | 1013 hPa

Sensor Bosch BME280 (I2C 0x76)

PRE-ALARMA PREVENTIVA (ADVERTENCIA)

29.4 °C (Sector Sur)

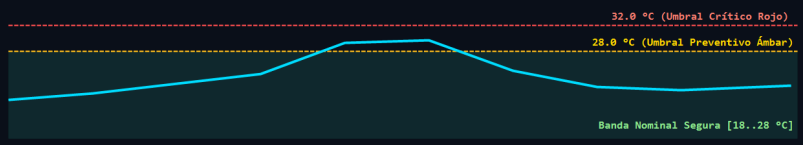
Supera umbral preventivo > 28.0 °C

ALARMA CRÍTICA ISA-18.2 (PRIORIDAD 1)

0 ALARMAS ACTIVAS

Disponibilidad SLA: A = 99.982% (Online)

CURVA DE SERIE TEMPORAL CON BANDA OPERATIVA SOMBREADA (CONSULTA FLUX aggregateWindow)



REGLAS DE ORO DE ERGONOMÍA VISUAL ISA-101:

1. Prohibido el efecto 'Árbol de Navidad': Si todo brilla en rojo, verde y amarillo saturado, el operador sufre fatiga visual y no detecta fallos.
2. Fondos oscuros mate (#0A101C / #121E36) y rejillas de bajo contraste para descanso ocular en turno 24/7.
3. Reserva cromática estricta:
   - Cian / Esmeralda -> Operación dentro de consigna
   - Ámbar (#FFBE1E) -> Desviación preventiva
   - Rojo (#FF5050) -> Alarma crítica que exige acción
4. Incluir siempre unidades físicas (°C, %RH, mA, dBm).

B) FLUJO DE ORQUESTACIÓN EN NODE-RED (:1880) CON HISTÉRESIS DE ALARMA Y CÁLCULO DE DISPONIBILIDAD (SLA)

① MQTT In
planta/+/telemetria
(QoS 1 · JSON)

② JSON Parser &
Schema Validator
(Verifica llaves)

③ Function Node
Filtro Histéresis
(T_on=30, T_off=28.5)

④ RBE / Filter
Report by Exception
(Bloquea spam)

⑤ Salida Dual:
• InfluxDB Batch Out
• Alerta Telegram/MQTT

LÓGICA JAVASCRIPT CON MEMORIA DE CONTEXTO E HISTÉRESIS EN NODE-RED:

```
const T_ALARMA_ON  = 30.0; // Umbral de disparo (°C)
const T_ALARMA_OFF = 28.5; // Umbral de reposición (Banda ΔT = 1.5 °C)
let enAlarma = context.get('enAlarma') || false;
if (!enAlarma && msg.payload.temp >= T_ALARMA_ON) {
    enAlarma = true;  context.set('enAlarma', true);
    return [msg, { payload: { alerta: 'ALTA_TEMP', valor: msg.payload.temp } }];
} else if (enAlarma && msg.payload.temp <= T_ALARMA_OFF) {
    enAlarma = false; context.set('enAlarma', false);
}
```

INGENIERÍA DE CONFIABILIDAD Y DISPONIBILIDAD DEL SISTEMA (MTBF / MTTR):

1. Ecuación de Disponibilidad Estacionaria (Availability):
   A = MTBF / (MTBF + MTTR) = T_operativo / T_total

2. Efecto de Docker 'restart: unless-stopped' + Healthchecks:
   Reduce el Tiempo Medio de Reparación (MTTR) de horas (intervención humana manual) a menos de 3.5 segundos (reinicio automático).

3. Combinado con el búfer SQLite WAL (Store-and-Forward):
   La disponibilidad efectiva de datos alcanza el 100.0% frente a microcortes de red o reinicios de mantenimiento de contenedores.

LISTADO DE MATERIALES (BOM) Y PRESUPUESTO DE ALMACENAMIENTO — MÓDULO 07

Servidor Edge de Series Temporales, Respaldo UPS, Reloj RTC DS3231 y Placa HAT Soldada

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 07 — Persistencia en Series Temporales, Docker, Node-RED, InfluxDB y Grafana

---

1. Tabla Maestra de Componentes de Hardware (BOM)

Ítem	Designador	Componente / Especificación Técnica	Cantidad	Función en la Arquitectura del Módulo 07
1	`SBC1`	**Raspberry Pi 4 Model B (4 GB / 8 GB RAM)** o Raspberry Pi 5 con disipador de aluminio anodizado	1	Servidor Linux ARM64 para orquestación de los 4 contenedores Docker (`Mosquitto` , `InfluxDB` , `Node-RED` , `Grafana`).
2	`SSD1`	**Unidad de Estado Sólido Externa USB 3.0 (256 GB NVMe/SATA)** con soporte UASP y TRIM	1	Almacenamiento de alta durabilidad para bloques TSM de InfluxDB y cola SQLite WAL (evita destruir la tarjeta microSD).
3	`U1 / BT1`	**Módulo Reloj en Tiempo Real DS3231SN (`I2C 0x68`)** con oscilador TCXO compensado (± 2 ppm) y pila `CR2032`	1	Garantiza marcas de tiempo exactas en nanosegundos UTC tras cortes eléctricos aunque aún no exista sincronización NTP.
4	`U2`	**Sensor de Corriente y Tensión High-Side INA219 (`I2C 0x40`)** con resistencia Shunt Kelvin de $0.100\ \Omega$ (1%)	1	Monitorea en tiempo real la tensión del riel de $5.1\ V$ y el consumo de corriente del servidor Edge y del SSD.
5	`U3`	**Optoacoplador Fototransistor PC817C (DIP-4)** con aislamiento galvánico de $2500\ V_{rms}$	1	Detecta instantáneamente la caída del riel primario de $12\ V\ DC$ y dispara la interrupción en `GPIO 17` para volcado limpio (`flush`).
6	`C_UPS`	**Banco Supercapacitor de Respaldo** $5.5\ V$ $4.0\ F$ (o módulo UPS de celdas `18650` de $5.1\ V$ $3\ A$)	1	Sostiene la tensión de $5.1\ V$ durante 15 a $30\ s$ tras un apagón para cerrar archivos SQLite WAL e InfluxDB sin corrupción.
7	`SW1`	**Switch Ethernet Industrial de 5 Puertos (10/100/1000 Mbps)** para riel DIN con cables Cat6 blindados	1	Interconexión de baja latencia entre la pasarela Raspberry Pi, los puntos de acceso inalámbricos y las consolas de operador.
8	`PCB1`	**Placa de Circuito Impreso FR-4 Doble Capa (`MECATROBOTICA-EDGE-HAT-V07`)** con cabezal hembra de 2×20 pines y bornera `KF301`	1	Montaje definitivo por soldadura libre de falsos contactos para operación continua $24/7$.
9	`R1..R5, D1..D2`	Resistencias de película metálica ($1\ k\Omega$, $10\ k\Omega$, $2\times\ 330\ \Omega$) + Diodos LED de $3\ mm$ Verde y Ámbar + Capacitor `100 nF X7R`	1 kit	Polarización del optoacoplador `PC817` , indicadores visuales de estado de ingesta (`GPIO 27` y `GPIO 22`) y desacoplo local.

---

2. Cálculo de Ingeniería: Presupuesto de Almacenamiento y Factor de Compresión

Para dimensionar científicamente la unidad SSD externa, calculamos el volumen bruto diario de telemetría $D_{día}$ generado por una planta con $N_{nodos} = 20$ nodos enviando tramas de $B_{muestra} = 150\ Bytes$ a una frecuencia de muestreo $f_s = 1.0\ Hz$:

$$D_{día} = N_{nodos} \cdot B_{muestra} \cdot f_s \cdot 86\,400\ s/día = 20 \times 150 \times 1.0 \times 86\,400 = 259.2\ MB/día$$

Gracias al motor **TSM (Time-Structured Merge Tree)** de InfluxDB 2.7 (que aplica codificación *Delta-of-Delta* en marcas de tiempo y compresión XOR *Gorilla* en punto flotante), el volumen físico en disco se reduce aproximadamente un factor de $6.8\times$, ocupando solo $\approx 38\ MB/día$ ($\approx 1.14\ GB$ para la ventana de retención cruda de $30\ días$).

Al ejecutar nuestra tarea automática de **Downsampling** en lenguaje `Flux` hacia ventanas de $\Delta t_{agregado} = 5\ minutos = 300\ s$, el factor de reducción temporal es:

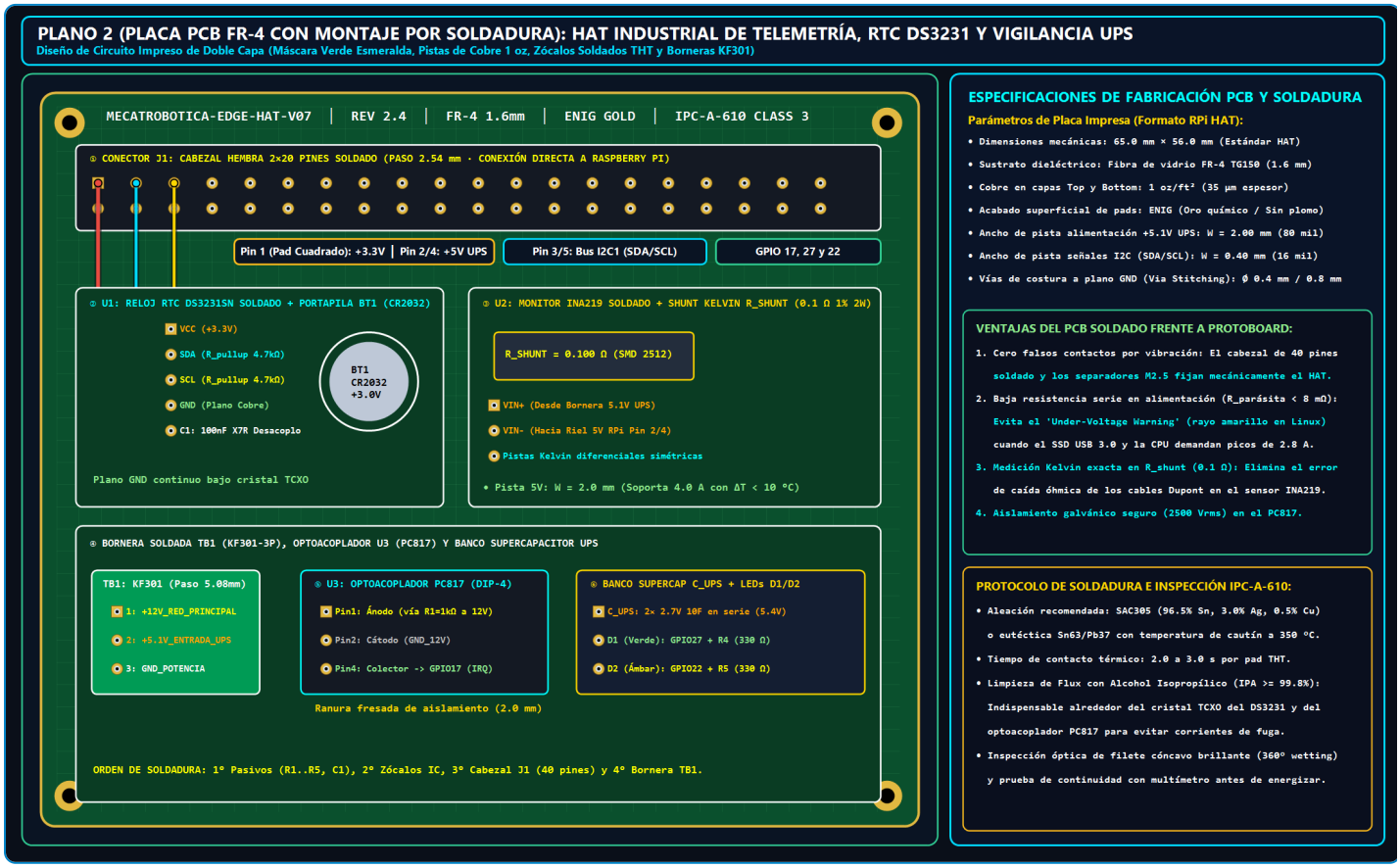
$$R_{comp} = \frac{\Delta t_{agregado}}{\Delta t_{crudo}} = (300\ s) / (1\ s) = 300 : 1$$

Por consiguiente, un año completo ($365\ días$) de historial analítico resumido (media, mínimo y máximo cada $5\ minutos$) requiere menos de $50\ MB$ de espacio en disco, garantizando consultas gráficas instantáneas en **Grafana**.

---

3. Comparativa de Montaje: Protoboard (Tinkercad) vs. Placa Impresa Soldada (PCB HAT)

📐 Plano 2: Placa PCB FR-4 con Montaje por Soldadura



MANUAL TÉCNICO DE INGENIERÍA — MÓDULO 07

Arquitectura de Contenedores Docker en ARM64, Series Temporales en InfluxDB 2.7, Resiliencia Store-and-Forward y Ergonomía Visual ISA-101

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

---

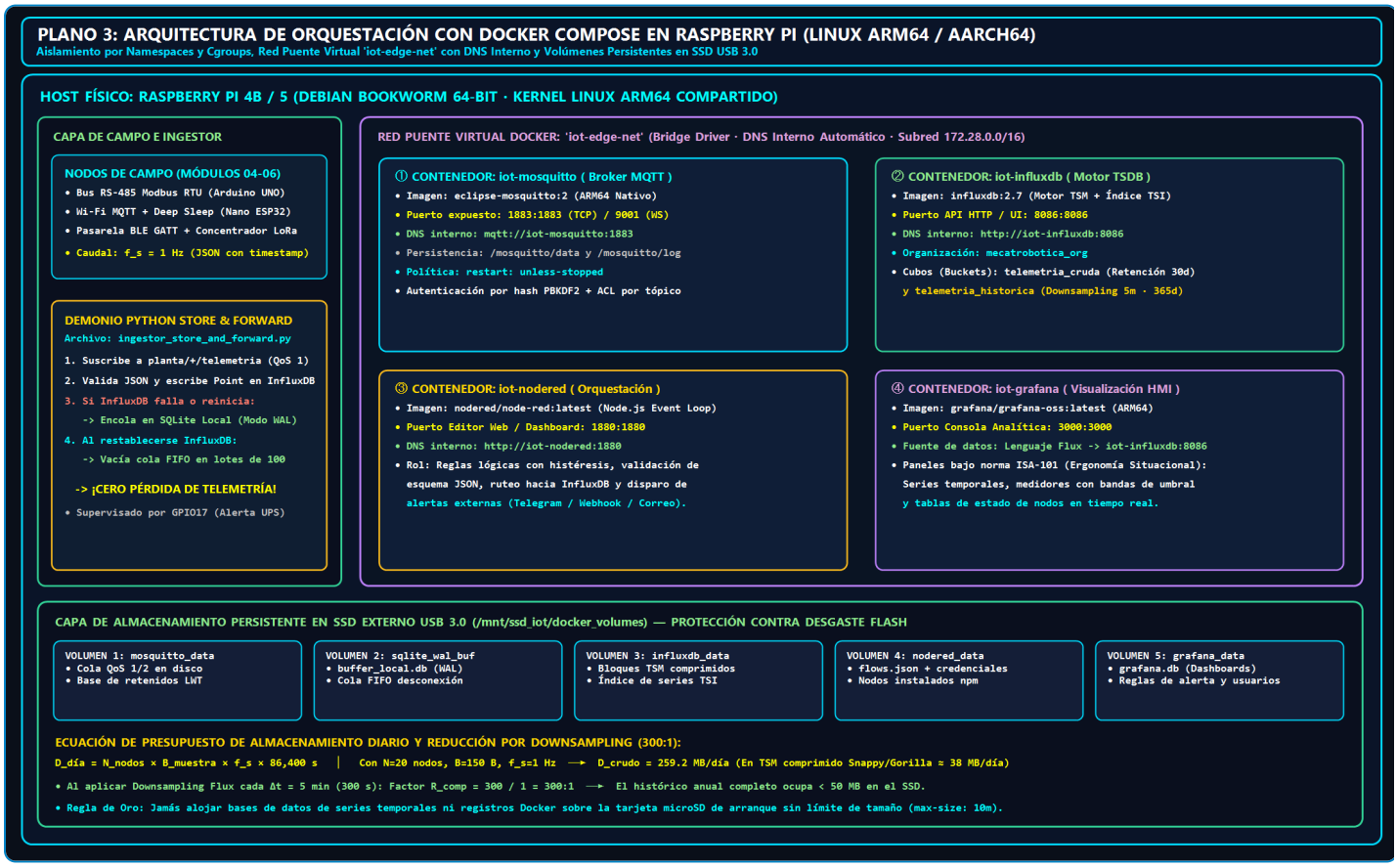
1. Fundamentos de Contenedorización con Docker en Linux ARM64

En despliegues industriales de borde (*Edge Computing*), instalar manualmente demonios de bases de datos, intérpretes de Node.js y servidores web directamente sobre el sistema operativo anfitrión conduce a conflictos de bibliotecas dinámicas y dificulta la recuperación ante desastres.

Mediante **Docker Engine** y **Docker Compose** sobre arquitecturas `ARM64 (aarch64)`, cada servicio opera aislado mediante dos primitivas nativas del kernel de Linux (sin la sobrecarga de emular hardware de una máquina virtual):

- **Espacios de nombres (`Namespaces`)**: Aíslan el árbol de procesos (`PID`), la pila de red (`NET`) y los puntos de montaje (`MNT`) de cada contenedor.
- **Grupos de control (`Cgroups v2`)**: Limitan y auditan el uso de memoria RAM y ciclos de CPU por servicio.
- **Red puente definida por el usuario (`iot-edge-net`)**: Provee resolución DNS interna automática por nombre de servicio (`iot-mosquitto`, `iot-influxdb`, `iot-nodered`, `iot-grafana`), evitando depender de direcciones IP estáticas frágiles.

📐 Plano 3: Arquitectura Docker Compose ARM64



---

2. Bases de Datos Relacionales (SQL) frente a Series Temporales (TSDB)

Una señal de telemetría mecatrónica es una sucesión discreta ordenada cronológicamente:

$$S = \{(t_1, \mathbf{v}_1), (t_2, \mathbf{v}_2), \dots, (t_k, \mathbf{v}_k)\} \quad \text{con } t_1 < t_2 < \dots < t_k$$

2.1. Anatomía del Protocolo de Línea (*Line Protocol*) en InfluxDB

Cada registro en InfluxDB se compone de cuatro elementos:

```
ambiente,nodo=invernadero_01,ubicacion=sector_norte temperatura=24.65,humedad=61.2,pwm_ventilador=140i
1730505600000000000
```

- **Medición (`Measurement`)**: Contenedor lógico de la magnitud (`ambiente`).
- **Conjunto de Etiquetas (`Tag Set`)**: Pares clave-valor de metadatos **indexados en memoria RAM** (`nodo`, `ubicacion`). Se utilizan en cláusulas de filtrado rápido.
- **Conjunto de Campos (`Field Set`)**: Valores numéricos variables **no indexados** (`temperatura`, `humedad`, `pwm_ventilador`).
- **Marca de Tiempo (`Timestamp`)**: Entero de 64 bits en nanosegundos UTC desde la época Unix.

2.2. Control Matemático de la Cardinalidad de Series

La cardinalidad total de series C_{series} en un cubo de InfluxDB viene dada por el producto cartesiano de los valores únicos de cada etiqueta:

$$C_{series} = |M| \times \prod_{i=1}^{m} |Tag_i| \times |Field_{keys}|$$

> **Regla Crítica de Diseño**: Jamás almacenes valores continuos (temperaturas con decimales), marcas de tiempo ni identificadores únicos por mensaje (`UUID`) dentro de un **Tag**. Hacerlo provoca una explosión de cardinalidad (*Runaway Cardinality*) que agota la memoria RAM de la Raspberry Pi (`OOM Kill`). Los valores variables van siempre en **Fields**.

---

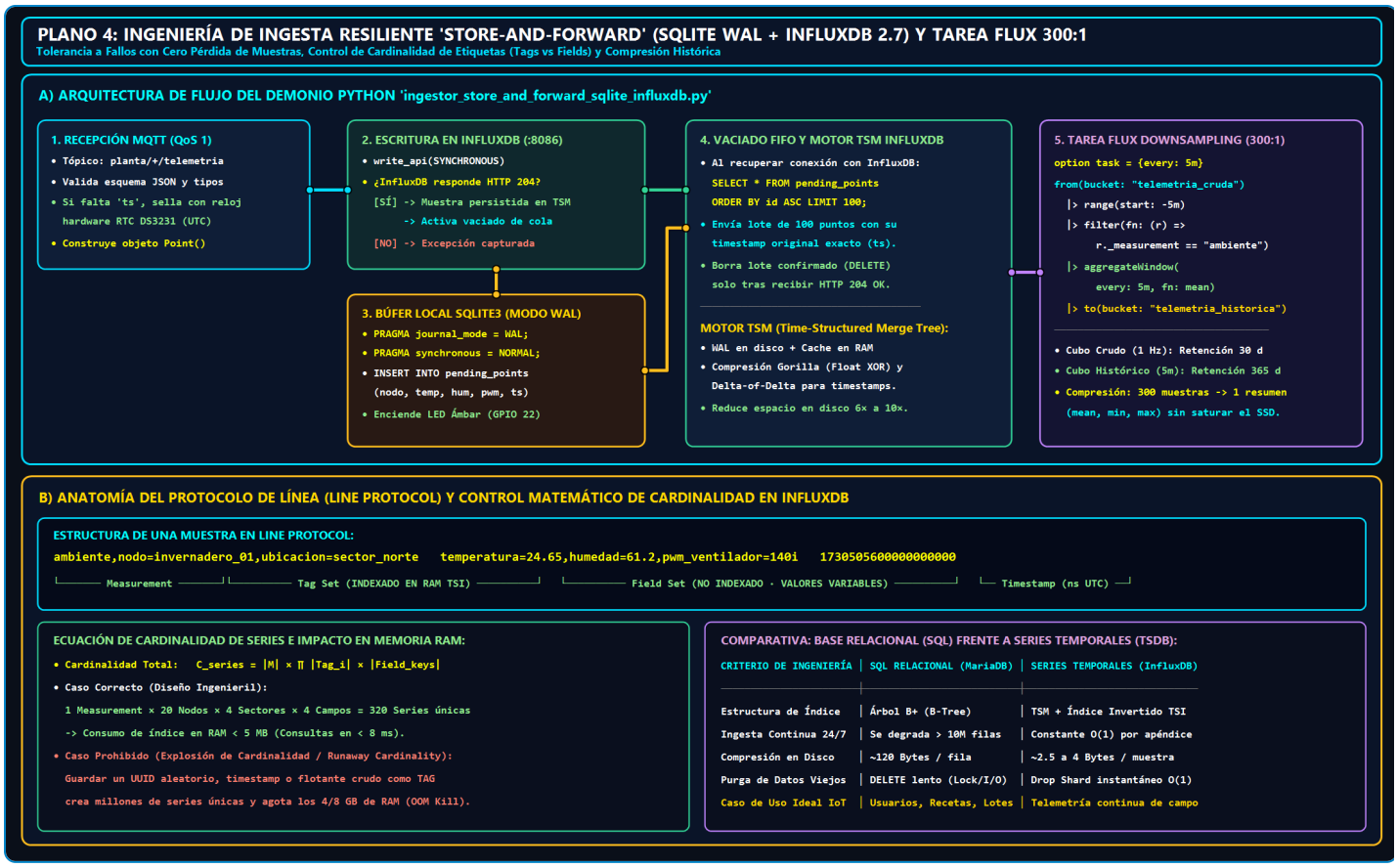
3. Patrón de Resiliencia *Store-and-Forward* con SQLite3 en Modo WAL

Para alcanzar una pérdida de datos igual a cero ($\text{Data Loss} = 0\%$) cuando el contenedor de InfluxDB se reinicia por mantenimiento o el enlace hacia un servidor central cae, nuestro demonio en Python implementa un búfer local en **SQLite3** activando la directiva:

```
PRAGMA journal_mode = WAL;
PRAGMA synchronous = NORMAL;
```

El modo **Write-Ahead Logging (`WAL`)** escribe las transacciones entrantes en un archivo secuencial de apéndice (`buffer_local_wal.db-wal`) sin bloquear las lecturas concurrentes y reduciendo drásticamente las operaciones de escritura aleatoria sobre las celdas Flash del SSD.

📐 **Plano 4: Flujo Store-and-Forward y Tarea Flux 300:1**



---

4. Ergonomía Visual bajo Norma ANSI/ISA-101 en Grafana

Un panel de control industrial no se evalúa por cuántos colores brillantes exhibe, sino por la velocidad con la que un operador humano detecta una condición anormal a las tres de la madrugada:

- **Contexto neutro de bajo contraste**: Fondos oscuros mate (`#0A101C` y `#121E36`) que reducen el deslumbramiento ocular.
- **Reserva cromática para anomalías**: Las curvas normales utilizan tonos cian o verde esmeralda moderados; el color **ámbar (`#FFBE1E`)** indica advertencia preventiva y el color **rojo (`#FF5050`)** se reserva exclusivamente para alarmas críticas activas.
- **Jerarquía de tres niveles**: Nivel 1 (KPIs globales y estado de salud), Nivel 2 (tendencias por subsistema con bandas sombreadas) y Nivel 3 (diagnóstico detallado de tramas, RSSI y latencias).

📐 Plano 5: Ergonomía Visual ISA-101 y Flujo Node-RED

PLANO 5: DISEÑO DE INTERFACES HMI BAJO NORMA ISA-101 EN GRAFANA Y FLUJO DE REGLAS EN NODE-RED

Conciencia Situacional para Operadores de Planta, Jerarquía de 3 Niveles, Histéresis de Alerta y Disponibilidad Operativa (SLA)

A) PANEL DE SUPERVISIÓN INDUSTRIAL GRAFANA (:3000) DISEÑADO BAJO ESTÁNDAR ANSI/ISA-101.01

TEMPERATURA PROCESO (NODO 01)	HUMEDAD RELATIVA Y PRESIÓN	PRE-ALARMA PREVENTIVA (ADVERTENCIA)	ALARMA CRÍTICA ISA-18.2 (PRIORIDAD 1)
24.6 °C	61.2 %RH | 1013 hPa	29.4 °C (Sector Sur)	0 ALARMAS ACTIVAS
Banda Nominal: [18.0 .. 28.0 °C] (OK)	Sensor Bosch BME280 (I2C 0x76)	Supera umbral preventivo > 28.0 °C	Disponibilidad SLA: A = 99.982% (Online)

CURVA DE SERIE TEMPORAL CON BANDA OPERATIVA SOMBREADA (CONSULTA FLUX aggregateWindow)



REGLAS DE ORO DE ERGONOMÍA VISUAL ISA-101:

1. Prohibido el efecto 'Árbol de Navidad': Si todo brilla en rojo, verde y amarillo saturado, el operador sufre fatiga visual y no detecta fallos.
2. Fondos oscuros mate (#0A101C / #121E36) y rejillas de bajo contraste para descanso ocular en turno 24/7.
3. Reserva cromática estricta:
   - Cian / Esmeralda -> Operación dentro de consigna
   - Ámbar (#FFBE1E) -> Desviación preventiva
   - Rojo (#FF5050) -> Alarma crítica que exige acción
4. Incluir siempre unidades físicas (°C, %RH, mA, dBm).

B) FLUJO DE ORQUESTACIÓN EN NODE-RED (:1880) CON HISTÉRESIS DE ALARMA Y CÁLCULO DE DISPONIBILIDAD (SLA)

① MQTT In
planta/+/telemetria
(QoS 1 · JSON)

② JSON Parser &
Schema Validator
(Verifica llaves)

③ Function Node
Filtro Histéresis
(T_on=30, T_off=28.5)

④ RBE / Filter
Report by Exception
(Bloquea spam)

⑤ Salida Dual:
• InfluxDB Batch Out
• Alerta Telegram/MQTT

LÓGICA JAVASCRIPT CON MEMORIA DE CONTEXTO E HISTÉRESIS EN NODE-RED:

```
const T_ALARMA_ON  = 30.0; // Umbral de disparo (°C)
const T_ALARMA_OFF = 28.5; // Umbral de reposición (Banda ΔT = 1.5 °C)
let enAlarma = context.get('enAlarma') || false;
if (!enAlarma && msg.payload.temp >= T_ALARMA_ON) {
    enAlarma = true;  context.set('enAlarma', true);
    return [msg, { payload: { alerta: 'ALTA_TEMP', valor: msg.payload.temp } }];
} else if (enAlarma && msg.payload.temp <= T_ALARMA_OFF) {
    enAlarma = false; context.set('enAlarma', false);
}
```

INGENIERÍA DE CONFIABILIDAD Y DISPONIBILIDAD DEL SISTEMA (MTBF / MTTR):

1. Ecuación de Disponibilidad Estacionaria (Availability):
   A = MTBF / (MTBF + MTTR) = T_operativo / T_total

2. Efecto de Docker 'restart: unless-stopped' + Healthchecks:
   Reduce el Tiempo Medio de Reparación (MTTR) de horas (intervención humana manual) a menos de 3.5 segundos (reinicio automático).

3. Combinado con el búfer SQLite WAL (Store-and-Forward):
   La disponibilidad efectiva de datos alcanza el 100.0% frente a microcortes de red o reinicios de mantenimiento de contenedores.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 07

Planos en Protoboard (Estilo Tinkercad), Placa PCB FR-4 con Montaje por Soldadura y Diagramas de Arquitectura

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

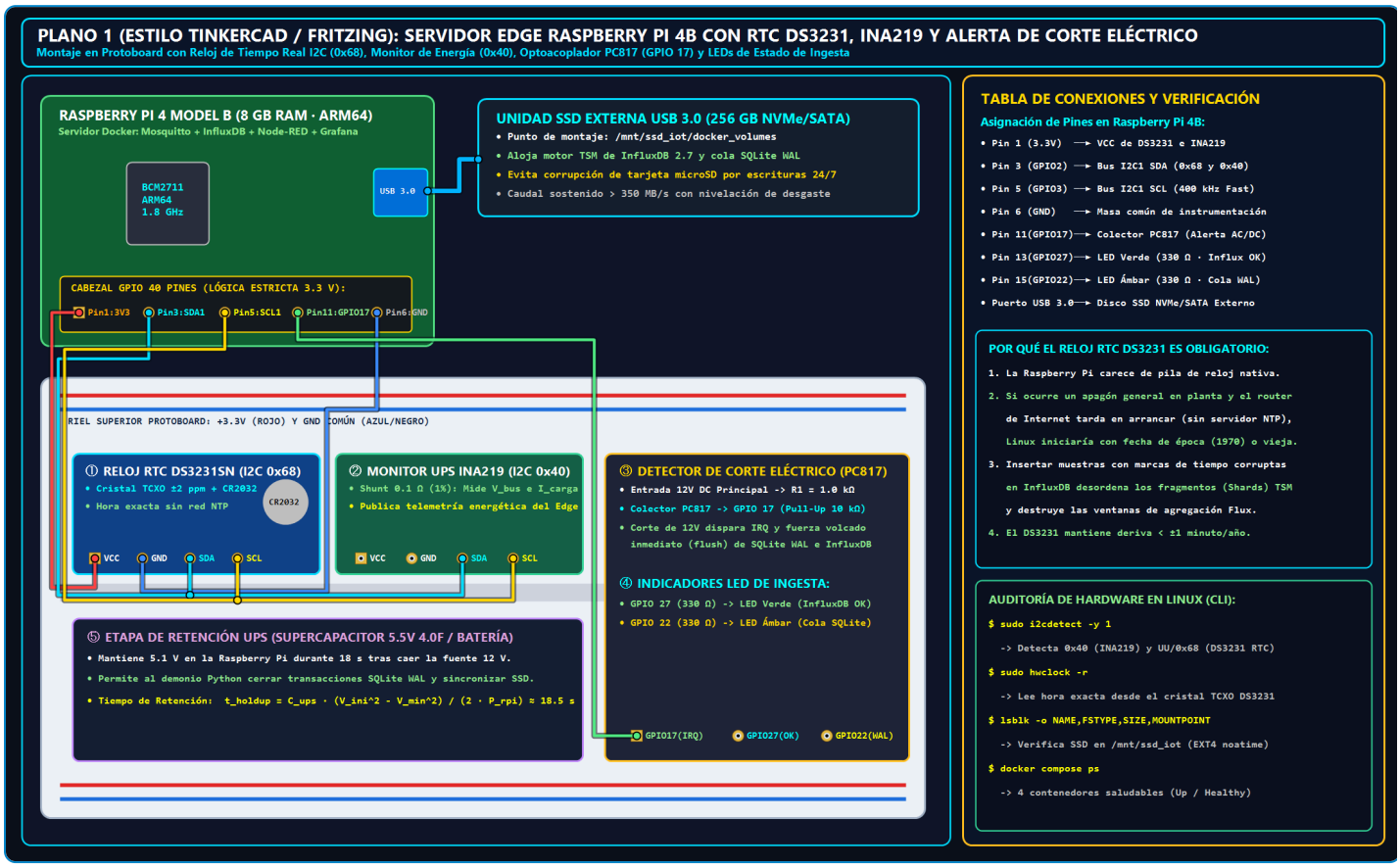
Módulo: 07 — Persistencia en Series Temporales, Docker, Node-RED, InfluxDB y Grafana

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje en Protoboard con RTC DS3231, INA219 y Alerta de Corte Eléctrico

En este primer plano de alta legibilidad se detalla el conexionado sobre protoboard del servidor Edge **Raspberry Pi 4B** acoplado a una unidad **SSD USB 3.0**, un reloj de tiempo real de alta precisión **DS3231SN (** `0x68` **)**, un monitor de potencia **INA219 (** `0x40` **)** y un detector optoacoplado de caída de alimentación principal **PC817** conectado al pin `GPIO 17`.

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad



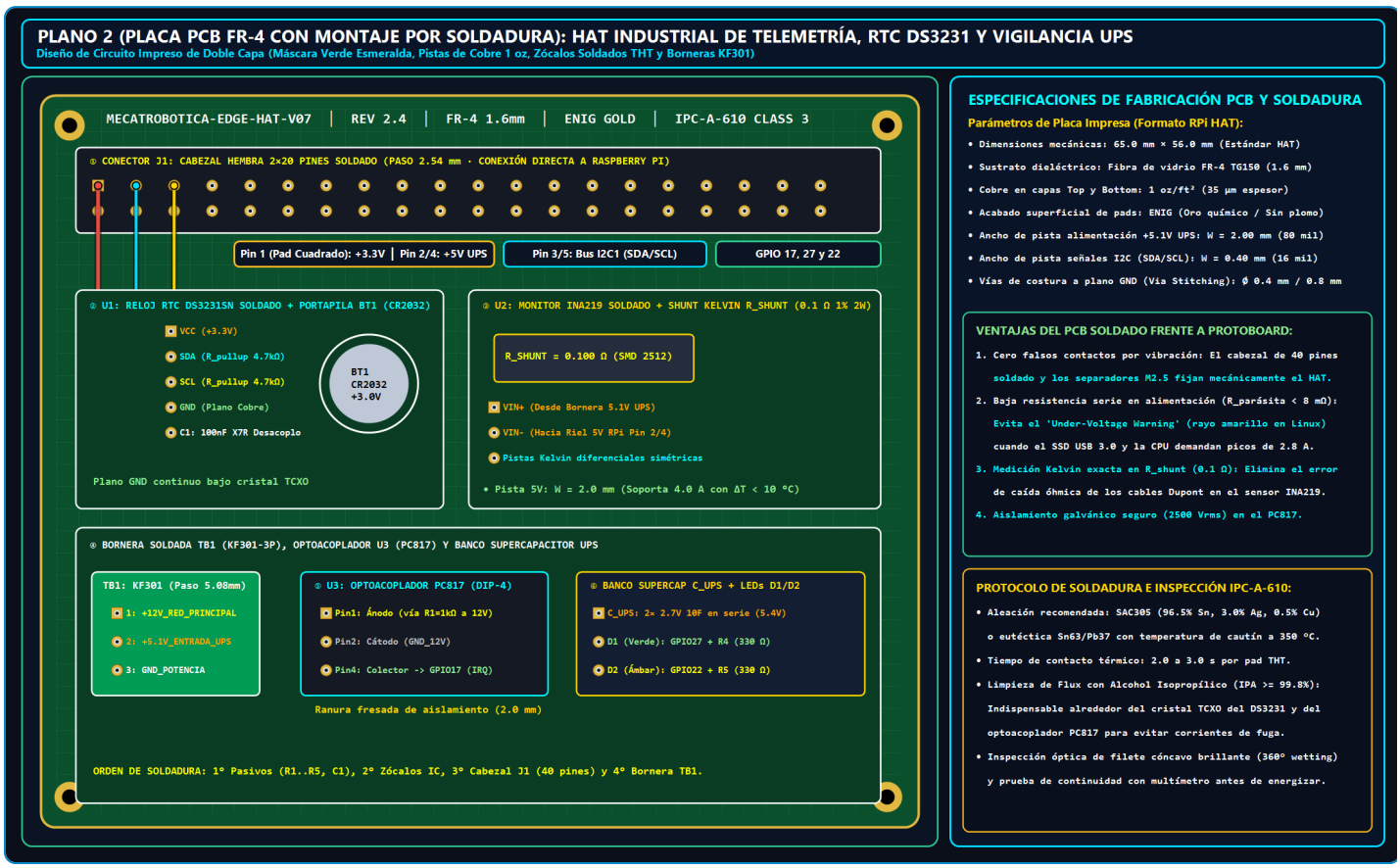
---

2. Plano 2 (Placa de Circuito Impreso PCB FR-4): Montaje por Soldadura de HAT Industrial para Raspberry Pi

Para servidores que operan las 24 horas de los 365 días del año en gabinetes sometidos a vibraciones de maquinaria, los cables puente Dupont sobre protoboard presentan riesgo de desconexión y caída óhmica en el riel de $5.1\ V$. Este segundo plano presenta el diseño de una **placa de circuito impreso de doble capa en fibra de vidrio FR-4 (** `MECATROBOTICA-EDGE-HAT-V07` **)** con montaje por soldadura (`IPC-A-610 Clase 3`):

- **Conector `J1` (2×20 pines hembra paso $2.54\ mm$ soldado)**: Acopla rígidamente el HAT sobre la Raspberry Pi junto con cuatro separadores mecánicos `M2.5`.
- **Pistas de potencia de $2.0\ mm$ de ancho y medición Kelvin en `R_SHUNT` ($0.100\ \Omega$)**: Soportan picos de corriente de hasta $4.0\ A$ con una caída resistiva despreciable.
- **Bornera de tornillo `TB1 (KF301)` soldada y ranura de aislamiento bajo el optoacoplador `PC817`**: Separan físicamente el riel industrial de $12\ V\ DC$ de la lógica de $3.3\ V$ del procesador Broadcom.

📐 Plano 2: Placa PCB FR-4 con Montaje por Soldadura

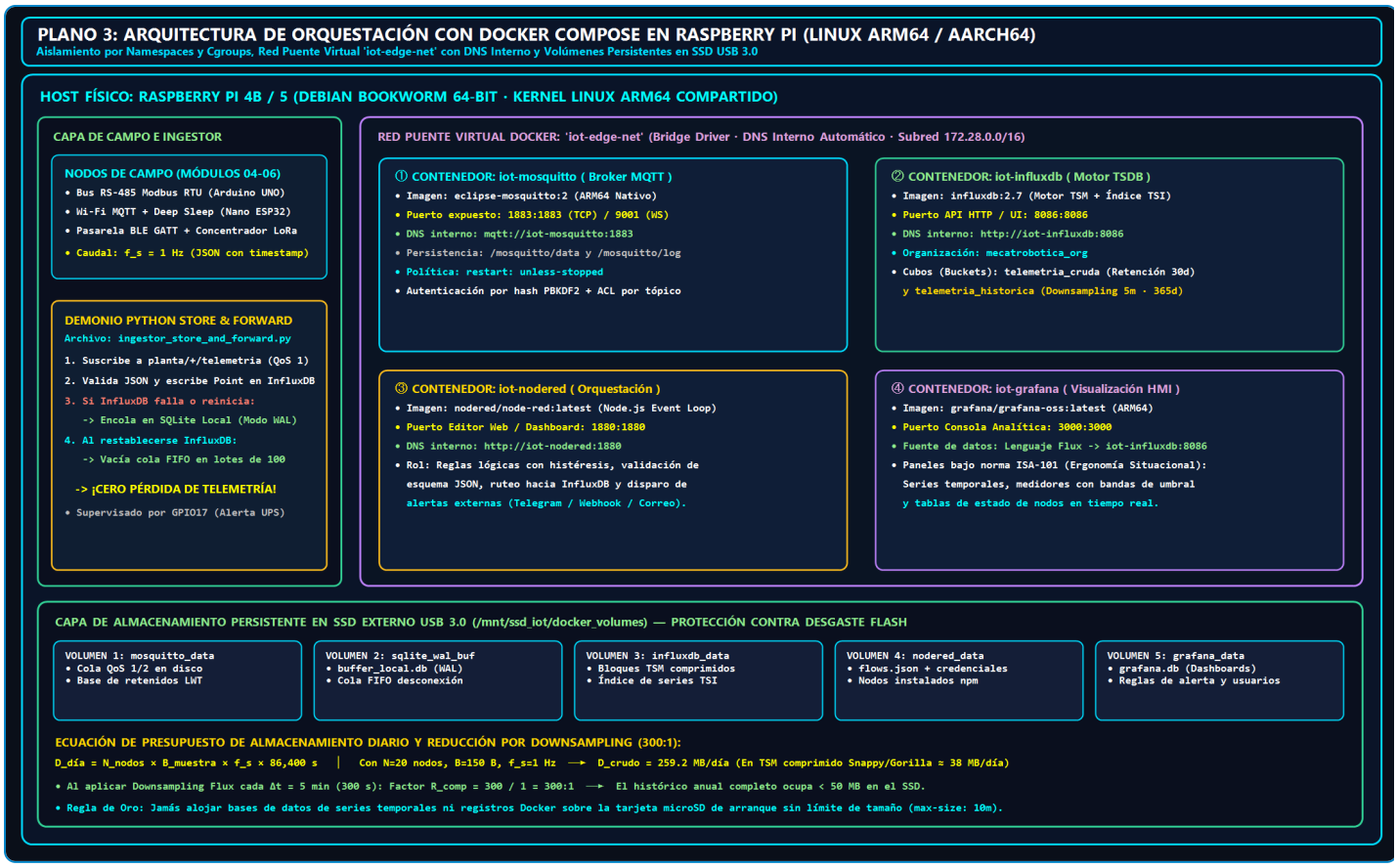


---

3. Plano 3: Arquitectura de Orquestación con Docker Compose en Linux ARM64

Representación detallada de la red puente aislada `iot-edge-net`, el mapeo de puertos de red (`1883`, `8086`, `1880` y `3000`) y la vinculación de los cinco volúmenes persistentes sobre la unidad externa **SSD USB 3.0** (`/mnt/ssd_iot/docker_volumes`).

📐 Plano 3: Arquitectura Docker Compose ARM64

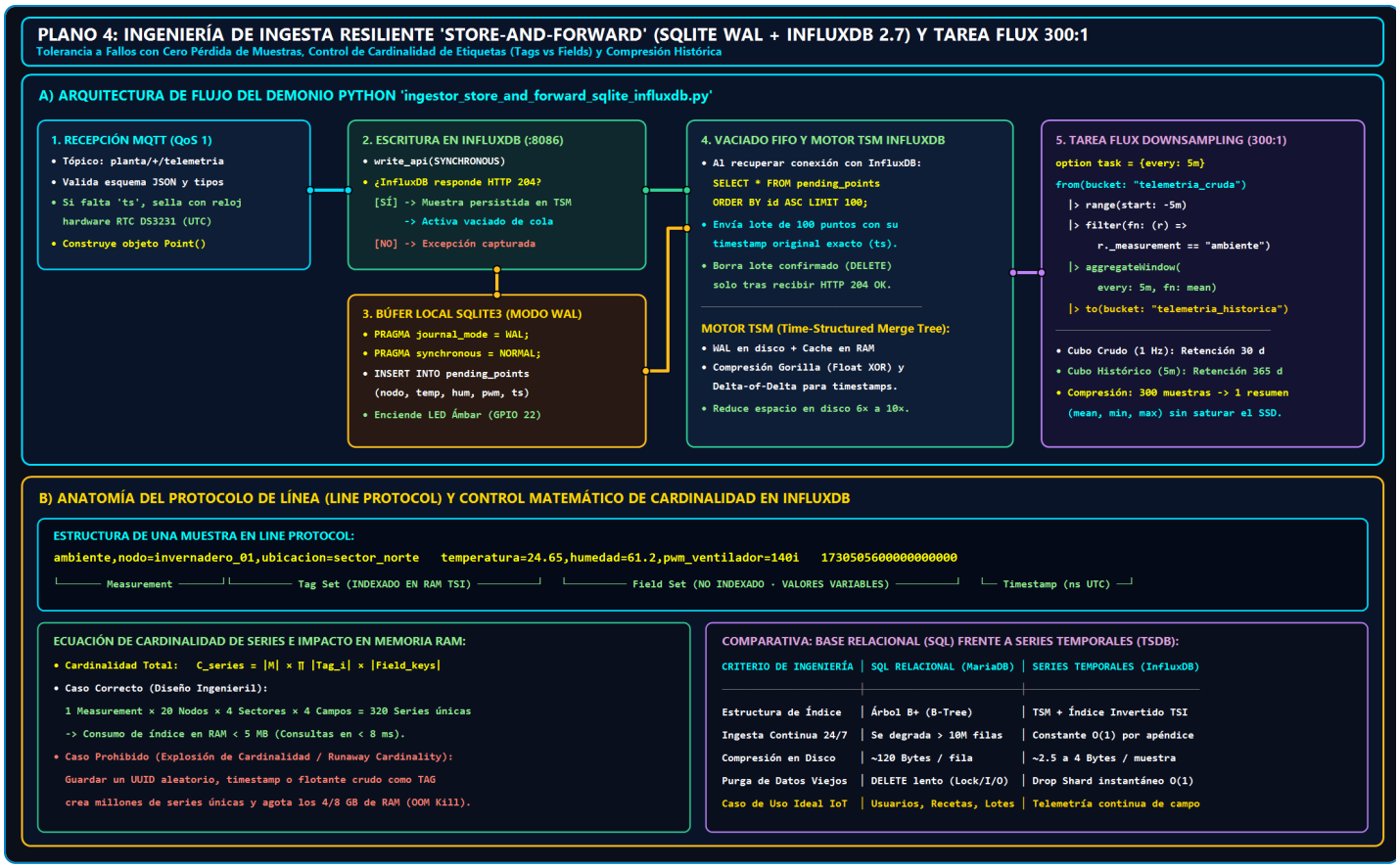


---

4. Plano 4: Flujo Store-and-Forward (SQLite WAL + InfluxDB 2.7) y Tarea Flux 300:1

Diagrama de ingeniería del demonio suscriptor en Python con conmutación automática a cola `SQLite3 WAL` ante caídas de red, recuperación por lotes FIFO (`LIMIT 100`), anatomía del *Line Protocol* y control de cardinalidad.

📐 Plano 4: Flujo Store-and-Forward y Tarea Flux 300:1



---

5. Plano 5: Ergonomía Visual ISA-101 en Grafana y Flujo de Reglas con Histéresis en Node-RED

Diseño de paneles de alto desempeño con banda operativa sombreada, semántica cromática estricta (cian/esmeralda nominal, ámbar preventivo y rojo crítico) y máquina de estados con histéresis en Node-RED.

📐 Plano 5: Ergonomía Visual ISA-101 y Flujo Node-RED

PLANO 5: DISEÑO DE INTERFACES HMI BAJO NORMA ISA-101 EN GRAFANA Y FLUJO DE REGLAS EN NODE-RED

Conciencia Situacional para Operadores de Planta, Jerarquía de 3 Niveles, Histéresis de Alerta y Disponibilidad Operativa (SLA)

A) PANEL DE SUPERVISIÓN INDUSTRIAL GRAFANA (:3000) DISEÑADO BAJO ESTÁNDAR ANSI/ISA-101.01

TEMPERATURA PROCESO (NODO 01)

24.6 °C

Banda Nominal: [18.0 .. 28.0 °C] (OK)

HUMEDAD RELATIVA Y PRESIÓN

61.2 %RH | 1013 hPa

Sensor Bosch BME280 (I2C 0x76)

PRE-ALARMA PREVENTIVA (ADVERTENCIA)

29.4 °C (Sector Sur)

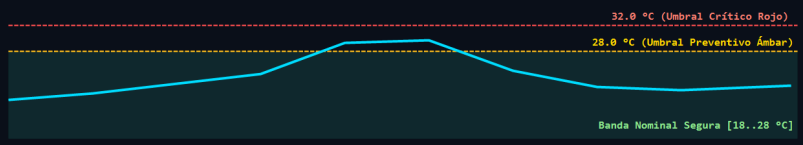
Supera umbral preventivo > 28.0 °C

ALARMA CRÍTICA ISA-18.2 (PRIORIDAD 1)

0 ALARMAS ACTIVAS

Disponibilidad SLA: A = 99.982% (Online)

CURVA DE SERIE TEMPORAL CON BANDA OPERATIVA SOMBREADA (CONSULTA FLUX aggregateWindow)



REGLAS DE ORO DE ERGONOMÍA VISUAL ISA-101:

1. Prohibido el efecto 'Árbol de Navidad': Si todo brilla en rojo, verde y amarillo saturado, el operador sufre fatiga visual y no detecta fallos.
2. Fondos oscuros mate (#0A101C / #121E36) y rejillas de bajo contraste para descanso ocular en turno 24/7.
3. Reserva cromática estricta:
   - Cian / Esmeralda -> Operación dentro de consigna
   - Ámbar (#FFBE1E) -> Desviación preventiva
   - Rojo (#FF5050) -> Alarma crítica que exige acción
4. Incluir siempre unidades físicas (°C, %RH, mA, dBm).

B) FLUJO DE ORQUESTACIÓN EN NODE-RED (:1880) CON HISTÉRESIS DE ALARMA Y CÁLCULO DE DISPONIBILIDAD (SLA)

① MQTT In
planta/+/telemetria
(QoS 1 · JSON)

② JSON Parser &
Schema Validator
(Verifica llaves)

③ Function Node
Filtro Histéresis
(T_on=30, T_off=28.5)

④ RBE / Filter
Report by Exception
(Bloquea spam)

⑤ Salida Dual:
• InfluxDB Batch Out
• Alerta Telegram/MQTT

LÓGICA JAVASCRIPT CON MEMORIA DE CONTEXTO E HISTÉRESIS EN NODE-RED:

```
const T_ALARMA_ON  = 30.0; // Umbral de disparo (°C)
const T_ALARMA_OFF = 28.5; // Umbral de reposición (Banda ΔT = 1.5 °C)
let enAlarma = context.get('enAlarma') || false;
if (!enAlarma && msg.payload.temp >= T_ALARMA_ON) {
    enAlarma = true;  context.set('enAlarma', true);
    return [msg, { payload: { alerta: 'ALTA_TEMP', valor: msg.payload.temp } }];
} else if (enAlarma && msg.payload.temp <= T_ALARMA_OFF) {
    enAlarma = false; context.set('enAlarma', false);
}
```

INGENIERÍA DE CONFIABILIDAD Y DISPONIBILIDAD DEL SISTEMA (MTBF / MTTR):

1. Ecuación de Disponibilidad Estacionaria (Availability):
   A = MTBF / (MTBF + MTTR) = T_operativo / T_total

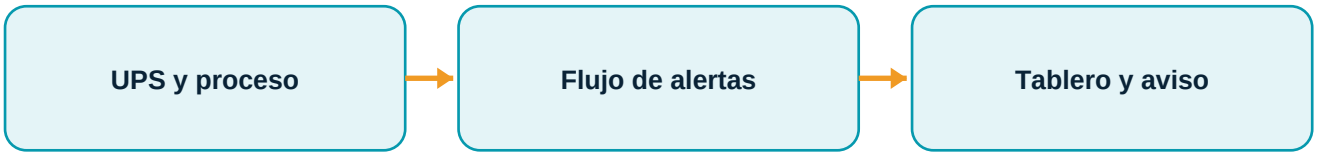
2. Efecto de Docker 'restart: unless-stopped' + Healthchecks:
   Reduce el Tiempo Medio de Reparación (MTTR) de horas (intervención humana manual) a menos de 3.5 segundos (reinicio automático).

3. Combinado con el búfer SQLite WAL (Store-and-Forward):
   La disponibilidad efectiva de datos alcanza el 100.0% frente a microcortes de red o reinicios de mantenimiento de contenedores.

Capítulo 24 · Proyecto: tablero Node-RED, bot de Telegram y UPS

Conceptos clave

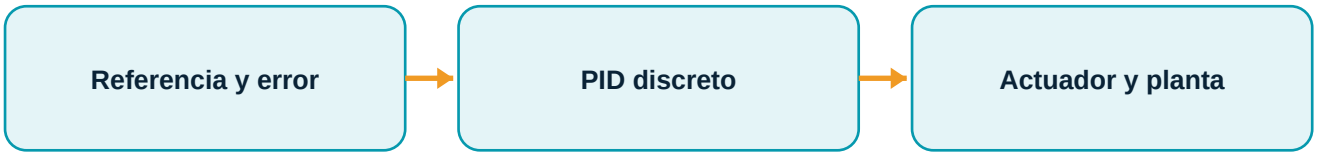
Separa la visualización local de la entrega de avisos. Prueba alertas de pérdida y retorno de energía sin exponer tokens.



Capítulo 25 · Control en lazo cerrado: PID discreto y anti-windup

Conceptos clave

Muestrea con un periodo definido. Aplica los límites del actuador y comprueba cómo se recupera la integral al salir de saturación.



Ejemplo complementario

```
def paso_pid(error, previo, integral, dt, kp, ki, kd):
    if dt <= 0:
        raise ValueError("dt debe ser positivo")
    candidata = integral + error * dt
    base = kp * error + kd * (error - previo) / dt
    propuesta = base + ki * candidata
    if 0 <= propuesta <= 100 or (propuesta > 100 and error < 0) or (propuesta < 0
and error > 0):
        integral = candidata
    return max(0, min(100, base + ki * integral)), integral
```

GUÍA PASO A PASO DE LABORATORIO — MÓDULO 08

Control en Lazo Cerrado (`PID` Discreto), Llamadas a Procedimiento Remoto (`RPC` sobre `MQTT`) e Inteligencia Artificial en el Borde (`Edge AI`)

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Regulación `PID` en Tiempo Discreto, Protocolo `RPC` Correlacionado, `Isolation Forest` y Visión por Eventos

Duración estimada de laboratorio: 4 horas prácticas

---

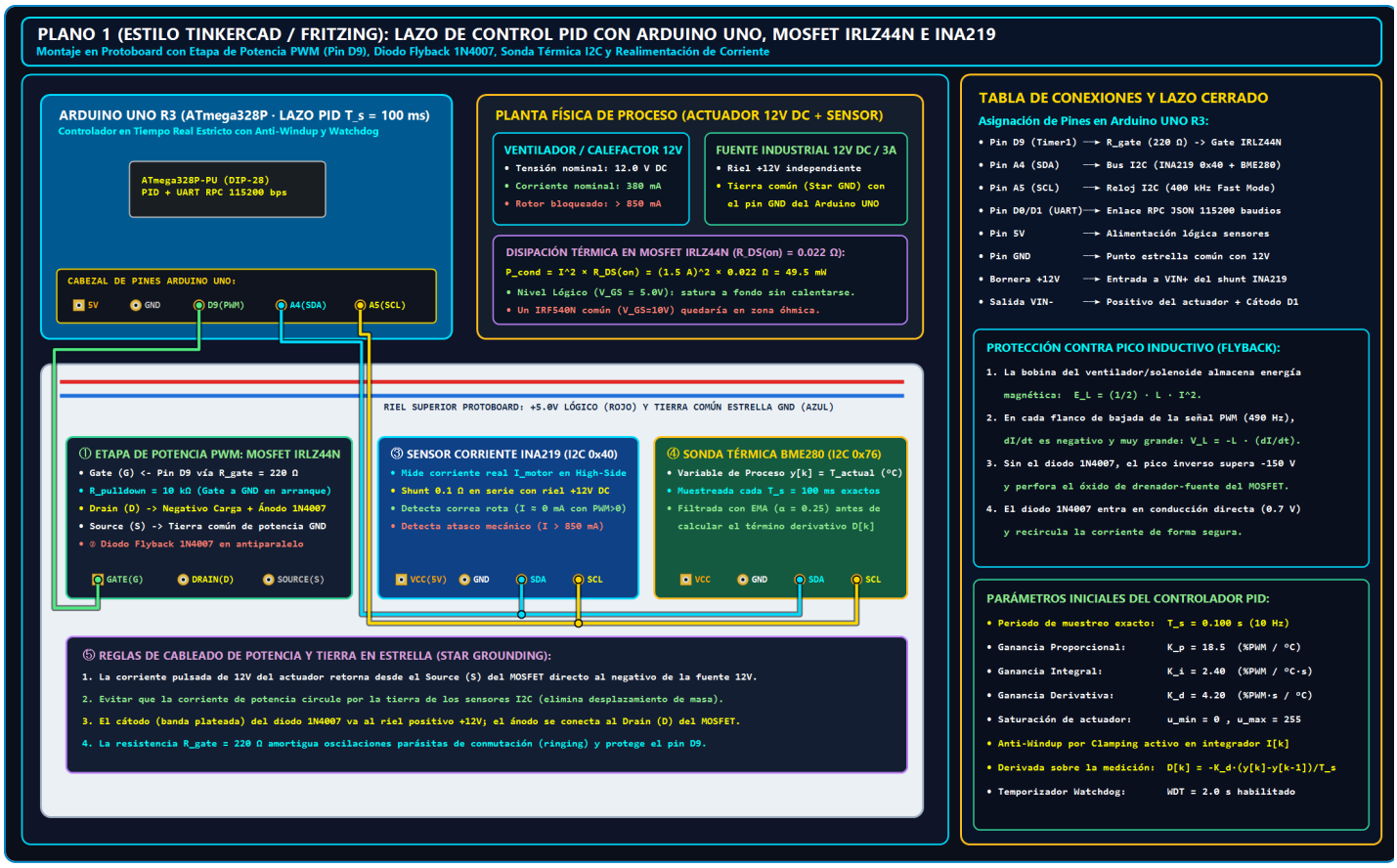
1. Objetivos de Ingeniería del Laboratorio

- **Separar la jerarquía de control en tiempo real estricto (`Hard Real-Time`) de la supervisión analítica en el borde (`Soft Real-Time`):** ejecutar el lazo cerrado proporcional-integral-derivativo (`PID`) cada $T_s = 100\ ms$ en el microcontrolador **Arduino UNO R3 (`ATmega328P`)**, delegando el ajuste de consigna (`Setpoint`), la inferencia de anomalías y la visualización a la **Raspberry Pi 4 / 5**.
- **Construir la etapa de potencia en modulación por ancho de pulso (`PWM`) con transistor `MOSFET` de nivel lógico (`IRLZ44N`):** calcular la disipación térmica por conducción, instalar la resistencia limitadora de compuerta ($220\ \Omega$), la resistencia de anclaje a tierra ($10\ k\Omega$) y el diodo volante de rueda libre (`1N4007`) en antiparalelo con la carga inductiva de $12\ V\ DC$.
- **Implementar ambos planos de construcción electrónica:** el prototipo experimental en **protoboard (estilo Tinkercad / Fritzing)** con tipografía ampliada de alta legibilidad y la **placa de circuito impreso (`PCB FR-4` de doble capa) con montaje por soldadura** y pistas de potencia de $2.50\ mm$.
- **Programar un protocolo de Llamada a Procedimiento Remoto (`RPC`) correlacionado sobre `MQTT` (`QoS 1`):** emparejar solicitudes y respuestas mediante un identificador único universal (`req_id`), sincronizar hilos con `threading.Event()` , establecer un temporizador de expiración de $1500\ ms$ y aplicar reversión segura (`Rollback`) en la interfaz gráfica si el hardware no confirma la orden.
- **Desplegar Inteligencia Artificial en el Borde (`Edge AI`) y captura visual dirigida por eventos:** ejecutar detección estadística por puntaje Z móvil (`Rolling Z-Score` 3σ) e inferencia multivariada con **Bosque de Aislamiento (`Isolation Forest`)** sobre el vector $\mathbf{x} = [T_{actual}, PWM, I_{motor}]^T$, disparando la cámara `CSI` (`Picamera2` + `OpenCV`) en el milisegundo exacto de una anomalía.

---

2. Arquitectura Física del Lazo Cerrado y Etapa de Potencia (`Plano 1`)

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad del Lazo de Control PID con MOSFET IRLZ44N e INA219



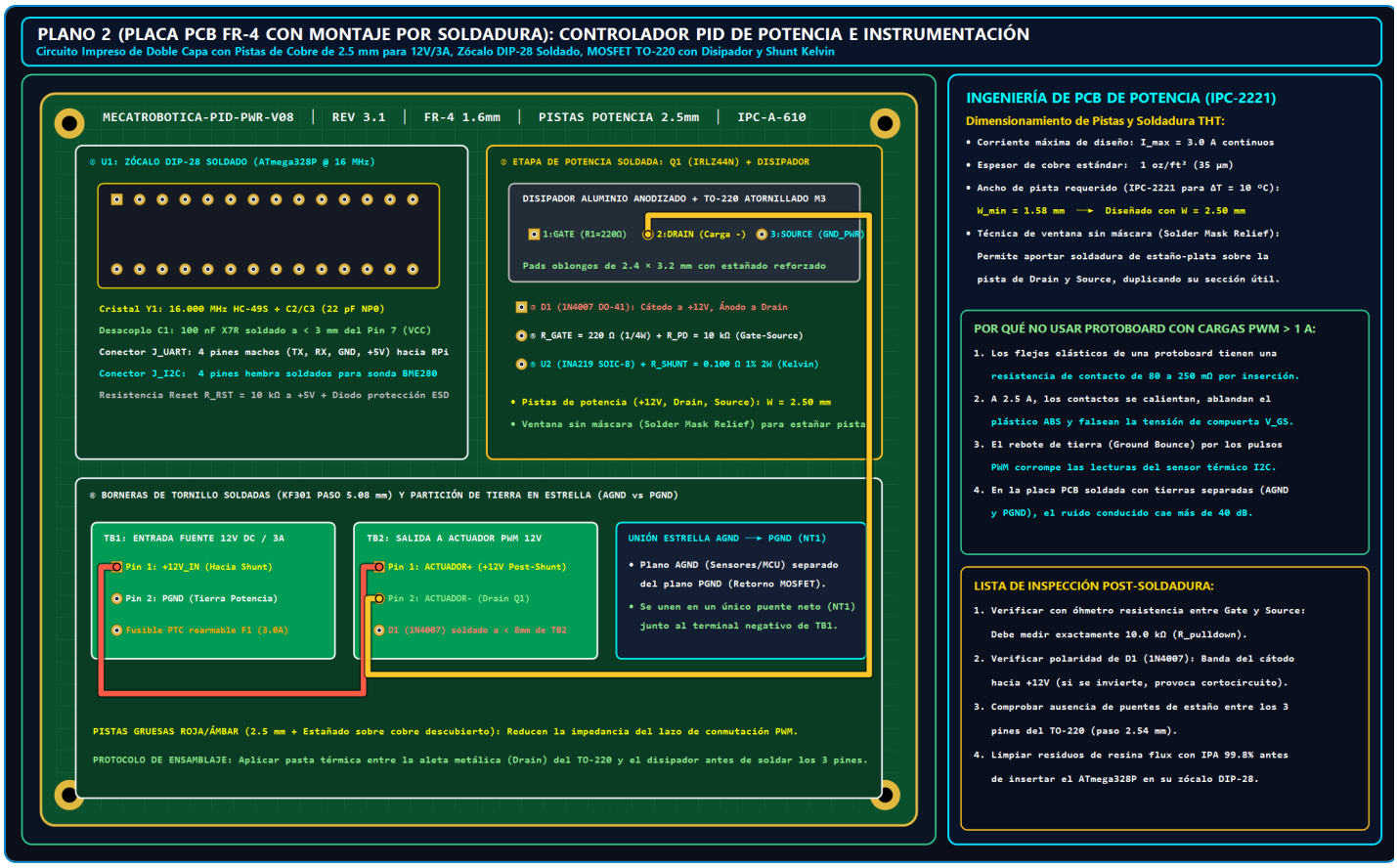
Paso 2.1 — Reglas Críticas de Cableado de Potencia y Tierra en Estrella (`Star Grounding`)

- **Por qué es obligatorio un `MOSFET` de nivel lógico (`IRLZ44N`):** el pin `D9` del Arduino UNO entrega $5.0\ \text{V}$ (y las placas de $3.3\ \text{V}$ entregan aún menos). Un transistor `MOSFET` estándar como el `IRF540N` requiere $V_{GS} = 10\ \text{V}$ para saturar completamente; si se excita con $5\ \text{V}$, permanece en su región lineal/óhmica y se destruye por sobrecalentamiento. El `IRLZ44N` posee tensión umbral $V_{GS(th)} \in [1.0, 2.0]\ \text{V}$ y alcanza $R_{DS(on)} = 0.022\ \Omega$ con $V_{GS} = 5.0\ \text{V}$.
- **Diodo de rueda libre (`Flyback 1N4007`):** conéctelo directamente entre los terminales del actuador inductivo (ventilador o válvula solenoide de $12\ \text{V}$), con el **cátodo (banda plateada)** hacia el riel positivo de $+12\ \text{V}$ y el **ánodo** hacia el drenador (`Drain`) del `IRLZ44N`.
- **Tierra en estrella (`Star Ground`):** el surtidor (`Source`) del `IRLZ44N` debe retornar directamente por un cable grueso al borne negativo de la fuente de $12\ \text{V DC}$, evitando que los pulsos de corriente de conmutación `PWM` circulen por la pista de tierra de los sensores `I2C` (`BME280` e `INA219`).

---

3. Montaje Definitivo en Placa de Circuito Impreso Soldada (`Plano 2`)

📐 Plano 2: Placa PCB FR-4 de Doble Capa con Montaje por Soldadura para Controlador PID de Potencia



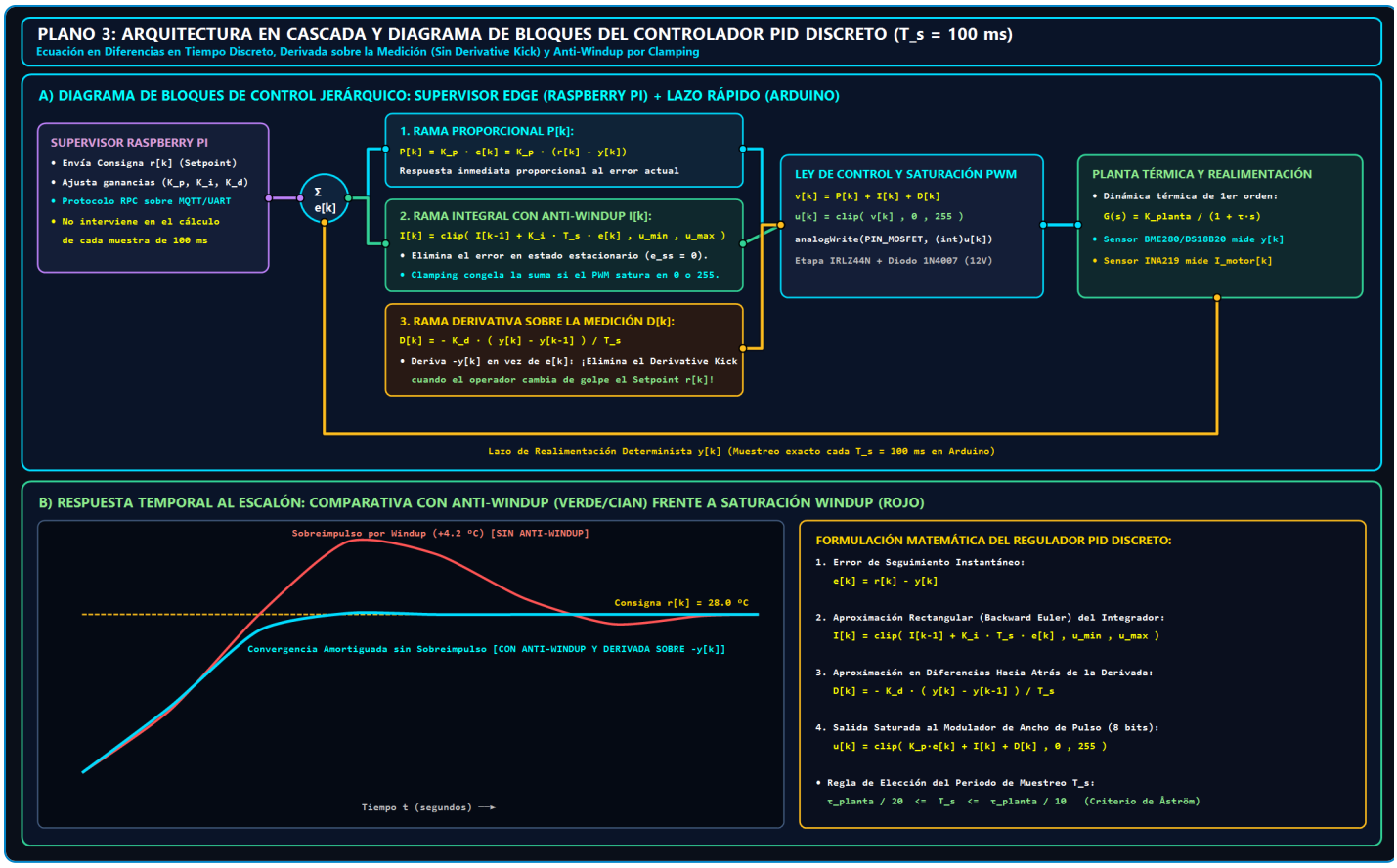
Paso 3.1 — Procedimiento de Soldadura de Potencia según `IPC-2221` e `IPC-A-610`

- Suelde primero los componentes de bajo perfil: resistencias de película metálica ($220\ \Omega$, $10\ k\Omega$), condensadores cerámicos `X7R` de $100\ nF$, el cristal de $16.000\ MHz$ y el sensor de corriente `INA219` junto a su resistencia `Shunt` Kelvin de $0.100\ \Omega$ (1%, $2\ W$).
- Suelde el **zócalo torneado** `DIP-28` para el microcontrolador `ATmega328P-PU` sin el chip insertado.
- Atornille la aleta metálica del `MOSFET IRLZ44N` (`TO-220`) al disipador de aluminio anodizado aplicando pasta térmica y suelde sus tres terminales en los pads oblongos reforzados.
- Suelde las borneras de tornillo `TB1` (entrada $12\ V\ DC$) y `TB2` (salida al actuador) y aporte una capa uniforme de aleación `SAC305` sobre las ventanas descubiertas de máscara (*Solder Mask Relief*) en las pistas de $2.50\ mm$.

---

4. Discretización del Controlador `PID` y Anti-Windup (`Plano 3`)

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup



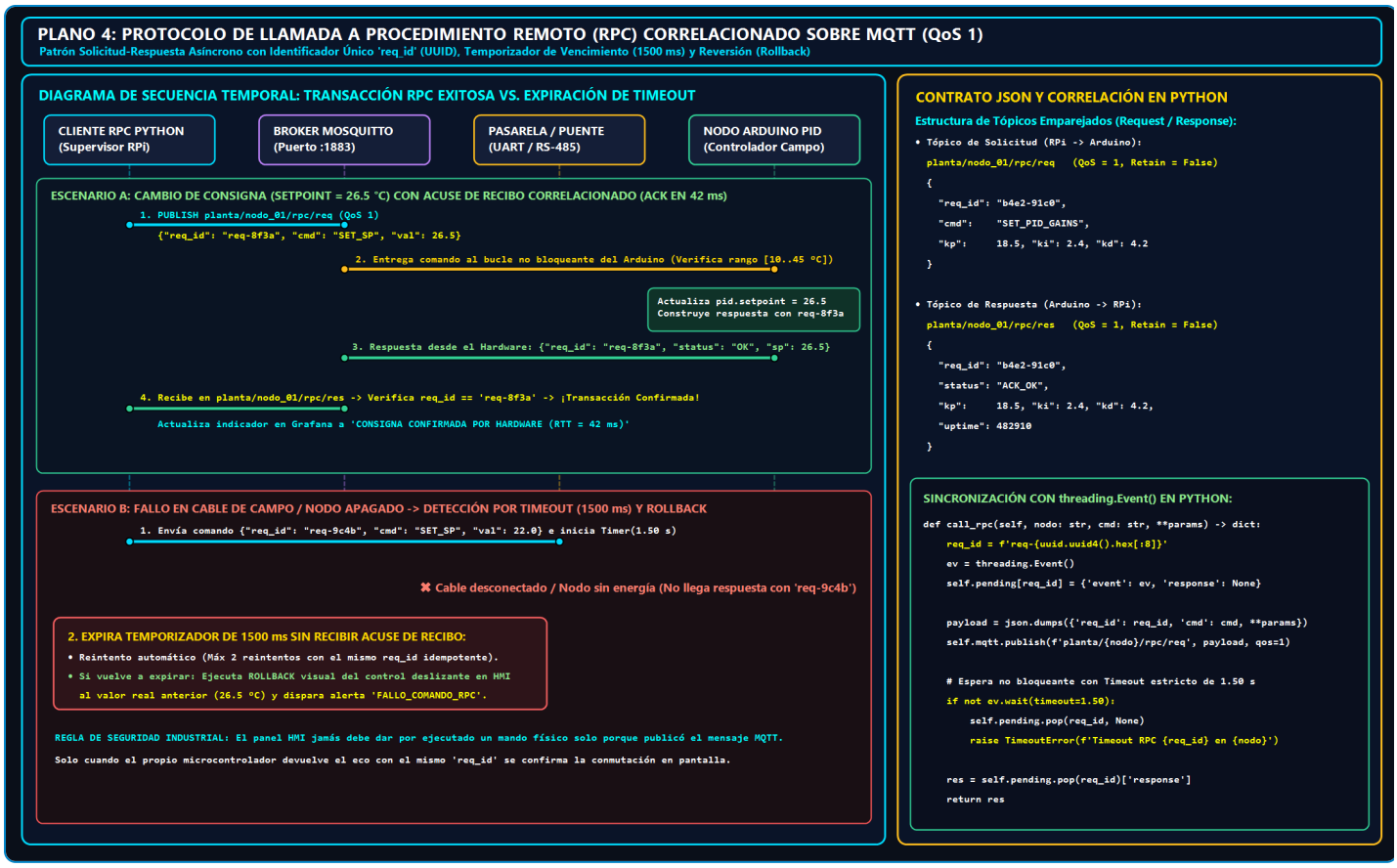
Paso 4.1 — Compilación y Carga del Firmware Determinista en Arduino UNO

- Abra el archivo `codigo_fuente/controlador_pid_discreto_rpc_arduino.ino` en Arduino IDE o PlatformIO.
- Verifique las tres mejoras industriales implementadas en la función `computeDiscretePID()`:
- **Muestreo exacto por temporizador no bloqueante:** $T_s = 100\ ms$ $(10\ Hz)$.
- **Derivada sobre la medición (**$-y[k]$**):** $D[k] = -K_d \cdot (y[k] - y[k-1]) / T_s$, eliminando el golpe derivativo (*Derivative Kick*) cuando el operador modifica de golpe el `Setpoint` desde Grafana.
- **Anti-Windup por recorte condicional (*Clamping*):** congela la acumulación del integrador $I[k]$ cuando la salida `PWM` alcanza 0 o 255.

---

5. Protocolo `RPC` Correlacionado sobre `MQTT` (`Plano 4`)

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



Paso 5.1 — Prueba de Transacción Confirmada y Prueba de Fallo con `Rollback`

- Ejecute el cliente supervisor en la Raspberry Pi:

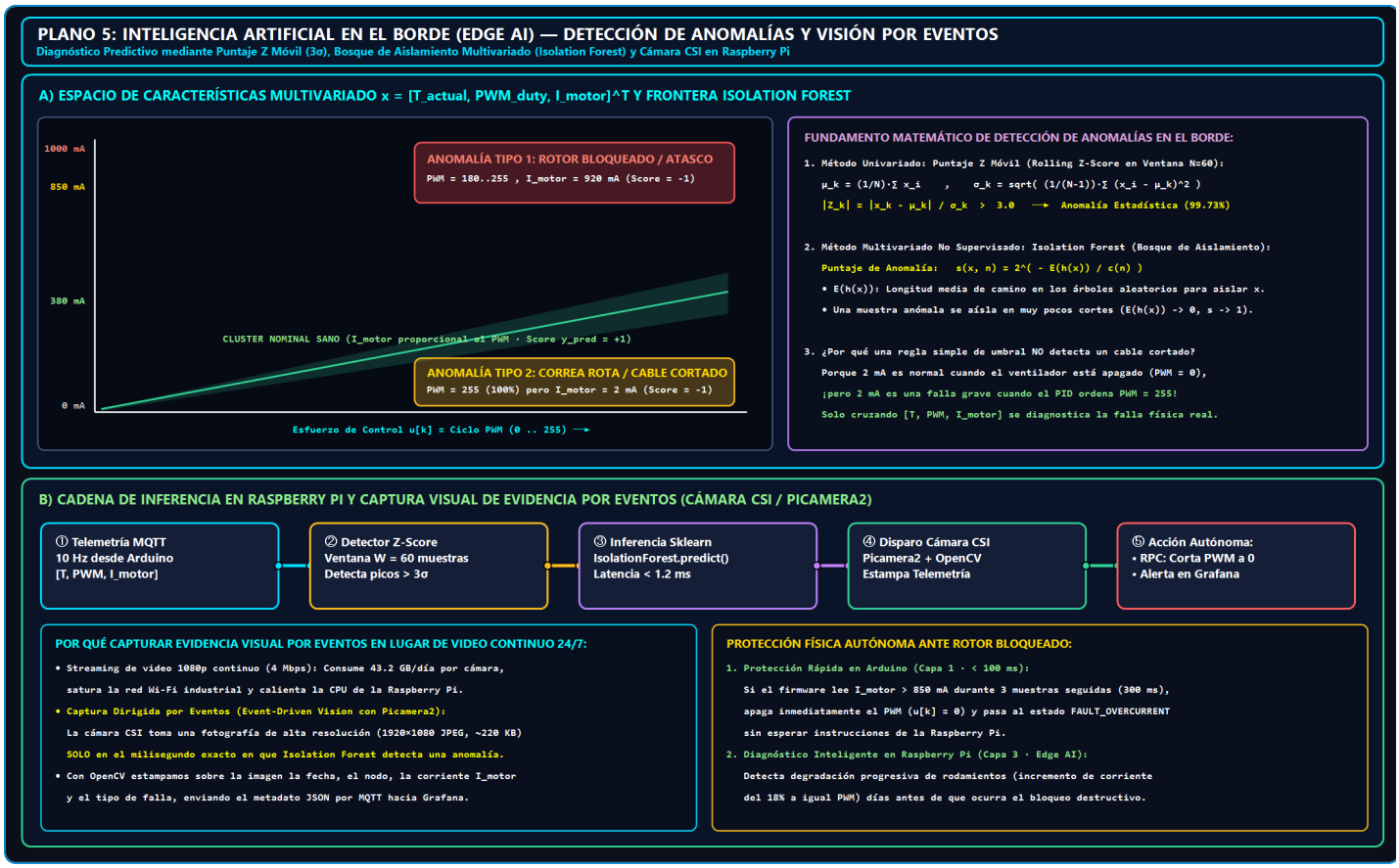
  ```bash
  python3 codigo_fuente/cliente_rpc_correlacionado_mqtt.py
  ```

- Verifique en consola que cada comando enviado en `planta/nodo_01/rpc/req` incluye un `req_id` único (por ejemplo, `req-8f3a19c2`) y que el hilo espera en `threading.Event.wait(timeout=1.50)` hasta recibir exactamente ese mismo `req_id` en `planta/nodo_01/rpc/res`.
- Desconecte momentáneamente el cable serial del Arduino y envíe una nueva consigna: compruebe cómo tras 1.50 s y 2 reintentos idempotentes, el servicio publica el evento de `Rollback` restaurando el indicador de la interfaz al valor real anterior.

---

6. Detección de Anomalías con `Edge AI` y Cámara `CSI` (`Plano 5`)

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos



Paso 6.1 — Ejecución del Detector `Isolation Forest` y Captura Forense por Eventos

- Inicie en terminales separadas el motor de inferencia y el servicio de cámara `CSI`:

  ```bash
  python3 codigo_fuente/detector_anomalias_edge_ai_isolation_forest.py
  python3 codigo_fuente/captura_visual_eventos_camara_csi.py
  ```

- **Prueba de Anomalía Tipo 1 (Rotor Bloqueado):** frene suavemente el eje del ventilador durante $400\ ms$. Observe cómo la corriente I_{motor} supera los $850\ mA$, el Arduino corta autónomamente el `PWM` a 0 en menos de $300\ ms$ y el motor `Edge AI` clasifica `ROTOR_BLOQUEADO_SOBRECORRIENTE`, disparando una fotografía `1080p` con marca de agua en `/var/lib/mecatrobotica/evidencia_visual/`.
- **Prueba de Anomalía Tipo 2 (Correa Rota / Cable Abierto):** desconecte el hilo positivo del ventilador mientras el controlador `PID` ordena $\text{PWM} = 255$. Aunque $I_{motor} \approx 2\ mA$ no viola ningún límite de sobrecorriente, el modelo multivariado **`Isolation Forest`** detecta inmediatamente que la combinación $[\text{PWM}=255,\ I_{motor}=2\ mA]$ se ubica fuera de la variedad nominal ($\text{Score} = -1$) y reporta `CORREA_ROTA_O_CABLE_ACTUADOR_ABIERTO`.

LISTADO DE MATERIALES E INSTRUMENTACIÓN (`BOM`) — MÓDULO 08

Hardware de Control en Lazo Cerrado `PID` , Etapa de Potencia `MOSFET PWM` , Sensórica `I2C` y Visión en el Borde (`CSI`)

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado (`PID`), Protocolo `RPC` , `Edge AI` y Captura Visual por Eventos

---

1. Computación de Supervisión (`Edge AI`) y Controlador de Tiempo Real (`PID`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
01	1	`SBC1`	**Raspberry Pi 4 Modelo B / Raspberry Pi 5 (`4 GB` o `8 GB` RAM)**	Supervisor de planta, cliente `RPC` correlacionado, motor `Isolation Forest` y captura de cámara `CSI` .
02	1	`U1`	**Arduino UNO R3 (`ATmega328P-PU` @ `16 MHz`)**	Ejecución determinista del lazo cerrado `PID` discreto cada $T_s = 100\ ms$ y protección de sobrecorriente.
03	1	`CAM1`	**Cámara Oficial Raspberry Pi `CSI` (`IMX219 8MP` o `IMX708 12MP`) con cinta `FFC` de 15/22 pines**	Captura fotográfica `1920x1080` dirigida por eventos ante detección de anomalías mecánicas o térmicas.

---

2. Etapa de Potencia `PWM` de $12\ V\ DC$ y Protección contra Transitorios Inductivos

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
04	1	`Q1`	**Transistor** `MOSFET` **Canal N de Nivel Lógico** `IRLZ44N` **(** `TO-220` **,** $V_{DSS}=55\ V$**,** $I_D=47\ A$**,** $R_{DS(on)}=0.022\ \Omega$ **@** $V_{GS}=5\ V$**)**	Conmutador de potencia en lado bajo (*Low-Side Switch*) gobernado por la señal `PWM` del pin `D9` ($490\ Hz$).
05	1	`HS1`	**Disipador de Aluminio Anodizado para cápsula** `TO-220` **(**$19 \times 15 \times 25\ mm$**) con tornillo** `M3` **y pasta térmica**	Evacuación térmica de pérdidas por conducción y conmutación del transistor `Q1`.
06	1	`D1`	**Diodo Rectificador de Rueda Libre** `1N4007` **(** `DO-41` **,** $1000\ V$**,** $1.0\ A$**) o** `1N5819` **(** `Schottky` $40\ V$**)**	Supresión del pico inductivo inverso ($V_L = -L \cdot dI/dt$) en cada corte del ciclo `PWM`.
07	1	`R_GATE`	**Resistencia de Película Metálica de** $220\ \Omega$ **(**$1/4\ W$**,** $\pm 1\%$**)**	Limita el pico de corriente de carga de la capacitancia de compuerta (C_{iss}) y amortigua oscilaciones parásitas (*ringing*).
08	1	`R_PD`	**Resistencia de Anclaje (** `Pull-Down` **) de** $10\ k\Omega$ **(**$1/4\ W$**,** $\pm 1\%$**)**	Mantiene la compuerta (`Gate`) firmemente en $0\ V$ durante el reinicio o alta impedancia del microcontrolador.
09	1	`ACT1`	**Planta Térmica / Ventilador Axial Industrial de** $12\ V\ DC$ **(**$380\ mA$ **nominal,** $> 850\ mA$ **rotor bloqueado)**	Actuador físico del lazo de regulación térmica y banco de pruebas de anomalías mecánicas.
10	1	`PSU1`	**Fuente Conmutada Industrial de** $12\ V\ DC\ /\ 3.0\ A$ **(**$36\ W$**) con riel negativo común en estrella**	Alimentación dedicada para la etapa de potencia de $12\ V$, desacoplada del riel lógico de $5\ V$.

3. Sensórica de Realimentación de Estado (`T_actual` e `I_motor`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
11	1	`U2`	**Módulo Sensor de Corriente y Tensión de Lado Alto** `INA219` **(** `I2C 0x40` **) con Shunt Kelvin de** $0.100\ \Omega$ **(**1%**,** $2\ W$**)**	Medición en tiempo real de $I_{motor}[k]$ y $V_{bus}[k]$ para protección de sobrecorriente y vector de `Edge AI`.
12	1	`U3`	**Sonda Ambiental Digital** `Bosch BME280` **(** `I2C 0x76` **) o** `DS18B20` **impermeable**	Medición de la variable controlada $y[k] = T_{actual}$ con resolución de $0.01\ ^\circ C$.

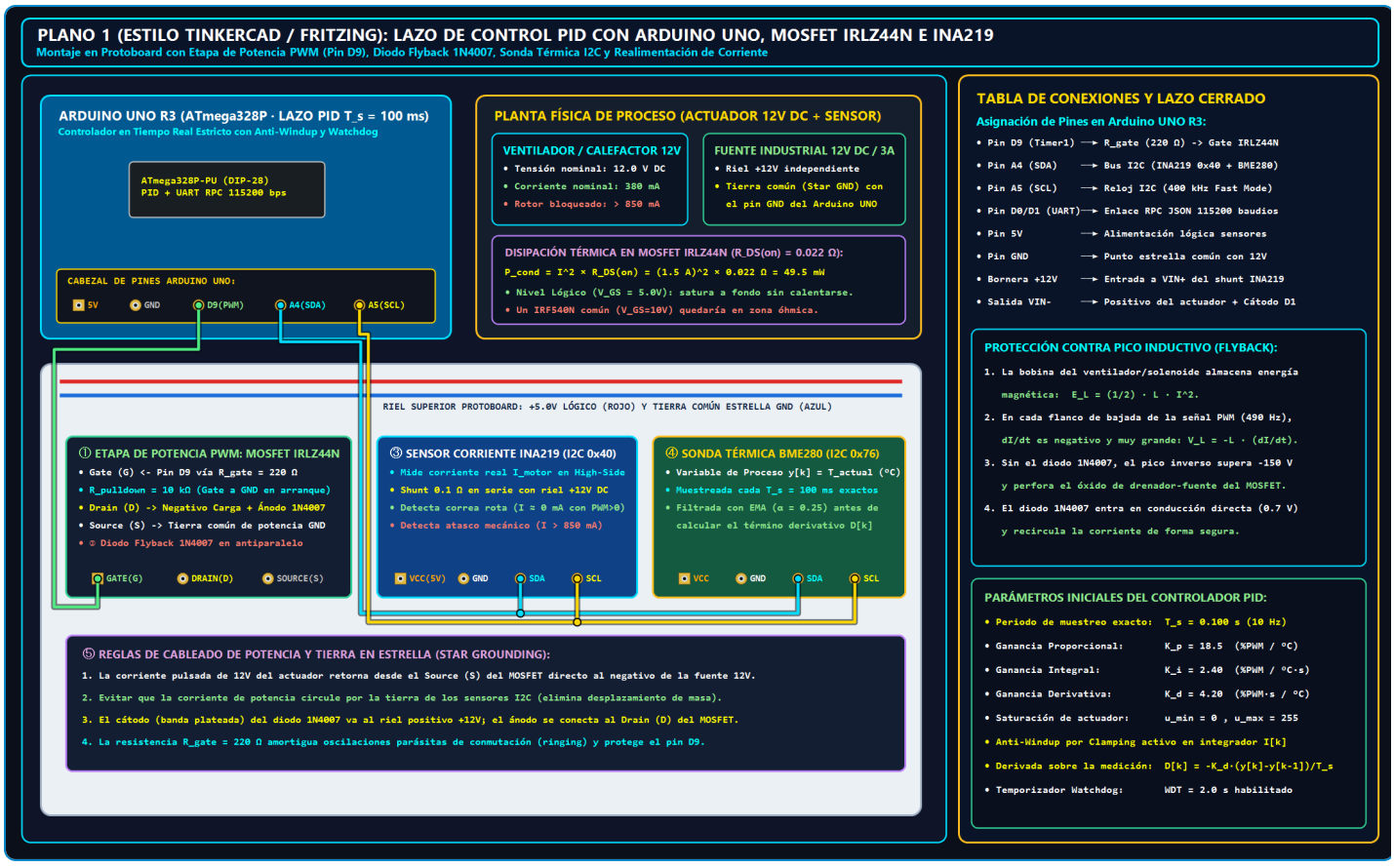
4. Insumos de Construcción para Placa Impresa Soldada (`PCB FR-4`) y Prototipo (`Tinkercad`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
13	1	`PCB1`	**Placa Virgen o Fabricada `FR-4` de Doble Capa (1.6 mm, cobre 1 oz/ft^2, pistas de potencia de 2.50 mm)**	Montaje definitivo por soldadura libre de falsos contactos y con partición de tierra `AGND` / `PGND`.
14	1	`SOCK1`	**Zócalo Torneado `DIP-28` de paso 2.54 mm + Cristal `HC-49S` 16.000 MHz + 2× 22 pF NP0**	Alojamiento seguro del microcontrolador `ATmega328P-PU` sin exponerlo al calor directo del cautín.
15	2	`TB1`, `TB2`	**Borneras de Tornillo para Circuito Impreso `KF301-2P` (paso 5.08 mm, 300 V / 15 A)**	Fijación mecánica firme de los cables de la fuente de 12 V DC y del actuador.
16	1	`F1`	**Fusible Polimérico Rearmable `PTC` de 3.0 A (`PPTC` radial)**	Protección contra cortocircuitos francos en la línea de +12 V.

5. Planos de Referencia para Inspección de Materiales y Montaje

5.1 Plano de Prototipado en Protoboard (`Plano 1`)

📐 Plano 1: Montaje en Protoboard del Lazo Cerrado PID

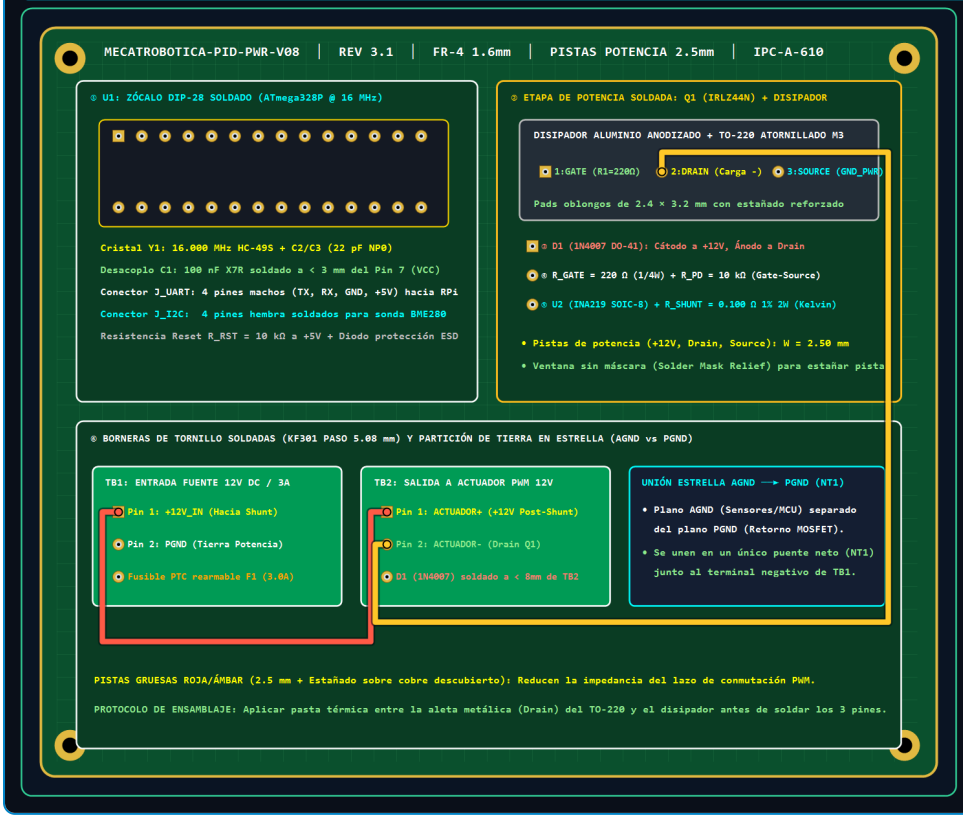


5.2 Plano de Circuito Impreso `FR-4` con Montaje por Soldadura (`Plano 2`)

📐 Plano 2: Placa PCB Soldada del Controlador PID de Potencia

PLANO 2 (PLACA PCB FR-4 CON MONTAJE POR SOLDADURA): CONTROLADOR PID DE POTENCIA E INSTRUMENTACIÓN

Circuito Impreso de Doble Capa con Pistas de Cobre de 2.5 mm para 12V/3A, Zócalo DIP-28 Soldado, MOSFET TO-220 con Disipador y Shunt Kelvin



INGENIERÍA DE PCB DE POTENCIA (IPC-2221)

Dimensionamiento de Pistas y Soldadura THT:

- Corriente máxima de diseño: $I_{max} = 3.0$ A continuos
- Espesor de cobre estándar: 1 oz/ft² (35 µm)
- Ancho de pista requerido (IPC-2221 para $\Delta T = 10$ °C):
  $W_{min} = 1.58$ mm ⟶ Diseñado con $W = 2.50$ mm
- Técnica de ventana sin máscara (Solder Mask Relief): Permite aportar soldadura de estaño-plata sobre la pista de Drain y Source, duplicando su sección útil.

POR QUÉ NO USAR PROTOBOARD CON CARGAS PWM > 1 A:

1. Los flejes elásticos de una protoboard tienen una resistencia de contacto de 80 a 250 mΩ por inserción.
2. A 2.5 A, los contactos se calientan, ablandan el plástico ABS y falsean la tensión de compuerta V_{GS}.
3. El rebote de tierra (Ground Bounce) por los pulsos PWM corrompe las lecturas del sensor térmico I2C.
4. En la placa PCB soldada con tierras separadas (AGND y PGND), el ruido conducido cae más de 40 dB.

LISTA DE INSPECCIÓN POST-SOLDADURA:

1. Verificar con óhmetro resistencia entre Gate y Source: Debe medir exactamente 10.0 kΩ ($R_{pulldown}$).
2. Verificar polaridad de D1 (1N4007): Banda del cátodo hacia +12V (si se invierte, provoca cortocircuito).
3. Comprobar ausencia de puentes de estaño entre los 3 pines del TO-220 (paso 2.54 mm).
4. Limpiar residuos de resina flux con IPA 99.8% antes de insertar el ATmega328P en su zócalo DIP-28.

MANUAL DE INGENIERÍA: CONTROL `PID` DISCRETO, PROTOCOLO `RPC` Y `EDGE AI` — MÓDULO 08

Fundamentos Matemáticos de Control en Tiempo Discreto, Conmutación `MOSFET PWM`, Correlación Asíncrona `MQTT` y Detección Multivariada con `Isolation Forest`

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado, Llamadas a Procedimiento Remoto (`RPC`) e Inteligencia Artificial en el Borde

---

1. Arquitectura de Control Jerárquico: ¿Por qué nunca cerrar un lazo `PID` rápido a través de `Wi-Fi` o `MQTT`?

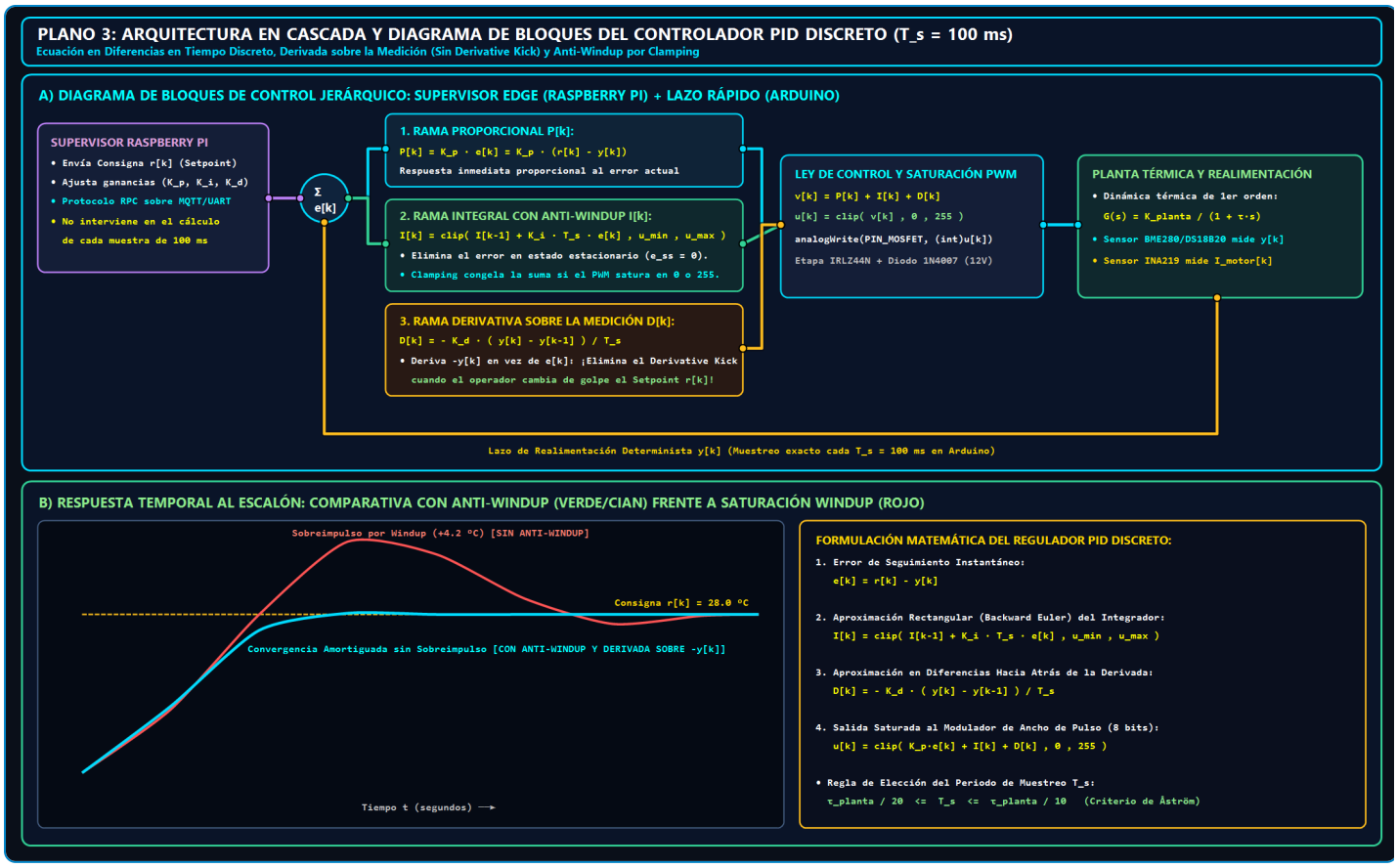
En la ingeniería de sistemas ciberfísicos e industriales, la primera regla de diseño consiste en **separar la capa de control regulatorio en tiempo real estricto (`Hard Real-Time`) de la capa de supervisión y analítica en tiempo real blando (`Soft Real-Time`)**:

- *Incertidumbre temporal (Jitter) de la red:* un enlace `Wi-Fi` o una cola de mensajes `MQTT` sobre `TCP/IP` presenta latencias variables que oscilan entre $3\ ms$ en condiciones ideales y más de $800\ ms$ cuando ocurre una retransmisión `TCP` o una recolección de basura en el sistema operativo.
- **Degradación del margen de fase:** en un controlador proporcional-integral-derivativo (`PID`), un retardo de transporte variable $\tau_d(t)$ introduce un desfase adicional $\Delta\phi(\Omega) = -\Omega \cdot \tau_d(t)$ que reduce el margen de fase del sistema hasta volverlo inestable u oscilatorio.
- **Solución arquitectónica en cascada:**
- **Nivel 1 (`Arduino UNO / ATmega328P`):** ejecuta el algoritmo `PID` discreto con un periodo de muestreo determinista $T_s = 100\ ms$ ($10\ Hz$), lee directamente los sensores `I2C` (`BME280` e `INA219`), comanda el modulador `PWM` del transistor `MOSFET IRLZ44N` y aplica enclavamiento autónomo de seguridad si la corriente supera $850\ mA$.
- **Nivel 2 (`Raspberry Pi 4 / 5`):** envía cambios de consigna $r[k]$ (`Setpoint`) o ajustes de ganancias (K_p, K_i, K_d) mediante transacciones `RPC` correlacionadas, ejecuta el motor de aprendizaje automático `Isolation Forest` y captura evidencia visual por cámara `CSI` .

---

2. Formulación Matemática del Controlador `PID` en Tiempo Discreto (`Plano 3`)

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup



En el dominio continuo, la ley de control `PID` ideal relaciona la señal de mando $u(t)$ con el error de seguimiento $e(t) = r(t) - y(t)$:

$$u(t) = K_p \, e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \, \frac{de(t)}{dt}$$

Al discretizar para un microcontrolador con periodo de muestreo constante $T_s = t_k - t_{k-1} = 0.100\ s$:

2.1 Término Proporcional $P[k]$

$$P[k] = K_p \cdot e[k] = K_p \cdot \left(r[k] - y[k] \right)$$

2.2 Término Integral $I[k]$ con Anti-Windup por Recorte Condicional (*Clamping*)

Aplicando la aproximación rectangular hacia atrás (*Backward Euler*) y acotando el acumulador dentro de los límites físicos del modulador `PWM` de 8 bits ($[u_{min}, u_{max}] = [0, 255]$):

$$I[k] = \text{clip}\left(I[k-1] + K_i \cdot T_s \cdot e[k] \;;\; u_{min} \;;\; u_{max} \right)$$

Además, si la suma total pre-saturación $v[k] = P[k] + I[k] + D[k]$ excede 255 y el error $e[k] > 0$ (o es menor que 0 con $e[k] < 0$), el integrador se congela para evitar el fenómeno de **arrollamiento integral (*Integral Windup)**, eliminando el sobreimpulso térmico.

2.3 Término Derivativo sobre la Medición $D[k]$ (Eliminación del *Derivative Kick*)

Si derivamos el error $e[k] = r[k] - y[k]$, cualquier cambio escalón en la consigna $r[k]$ enviado por el operador desde Grafana produce una pendiente instantánea $\Delta r / T_s$ enorme que satura de golpe el actuador (*Derivative Kick*). Como $r[k]$ es constante por tramos, $\frac{dr}{dt} = 0$ salvo en el instante del cambio de consigna, por lo que reemplazamos $\frac{de}{dt}$ por $-\frac{dy}{dt}$:

$$D[k] = -K_d \cdot (y[k] - y[k-1]) / (T_s)$$

Previamente filtramos la temperatura cruda con un **filtro de media móvil exponencial (`EMA`)** con factor $\alpha = 0.25$:

$$y_{filt}[k] = \alpha \cdot y_{raw}[k] + (1 - \alpha) \cdot y_{filt}[k-1]$$

---

3. Física de Conmutación `MOSFET PWM` y Supresión de Pico Inductivo

- **Pérdidas por conducción en el `MOSFET` de nivel lógico `IRLZ44N`:**

  $$P_{cond} = I_{carga}^2 \cdot R_{DS(on)} = (1.5\ A)^2 \times 0.022\ \Omega = 0.0495\ W \approx 49.5\ mW$$

  Con una resistencia térmica unión-ambiente $R_{\theta JA} = 62\ ^\circ C/W$, el incremento de temperatura sin siquiera disipador para $1.5\ A$ es de apenas $\Delta T = 0.0495 \times 62 \approx 3.1\ ^\circ C$.

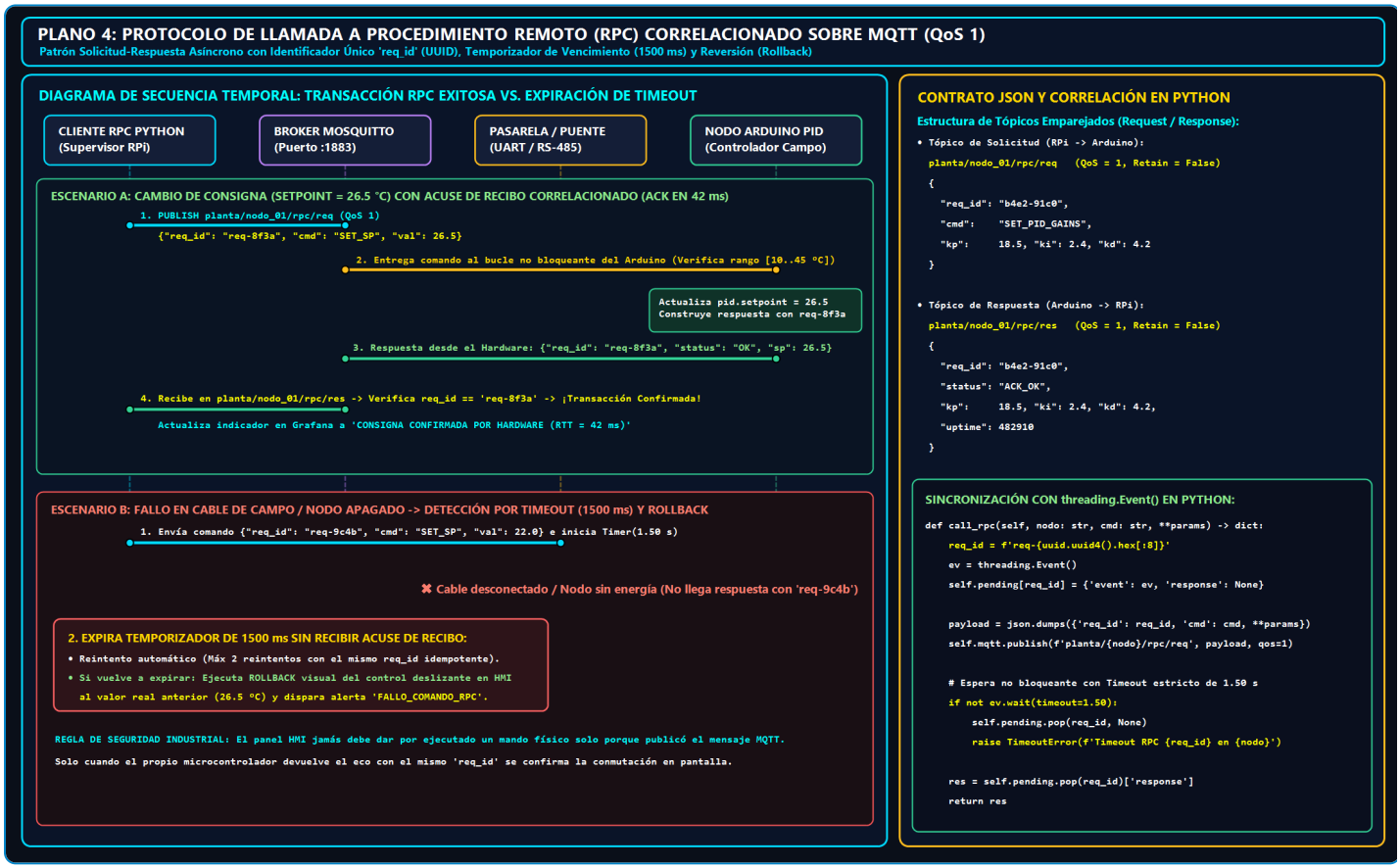
- **Física del diodo volante (`Flyback 1N4007`):**

  El devanado de un ventilador o electroválvula almacena energía magnética $E_L = (1)/(2)\ L\ I^2$. Cuando el transistor `IRLZ44N` pasa de conducción a corte en pocos nanosegundos, la derivada $(dI)/(dt)$ es negativa y de gran magnitud, induciendo una fuerza contraelectromotriz $V_L = -L\ (dI)/(dt)$ que puede superar $+150\ V$ en el drenador y destruir el transistor por avalancha. El diodo `1N4007` en antiparalelo entra en conducción directa a $0.7\ V$, ofreciendo un camino seguro de recirculación para la corriente inductiva.

---

4. Protocolo `RPC` Correlacionado sobre `MQTT` (`Plano 4`)

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



Dado que `MQTT` es un protocolo intrínsecamente asíncrono de publicación/suscripción, publicar un comando en un tópico no garantiza que el microcontrolador de campo lo haya recibido ni ejecutado. Para construir una **Llamada a**

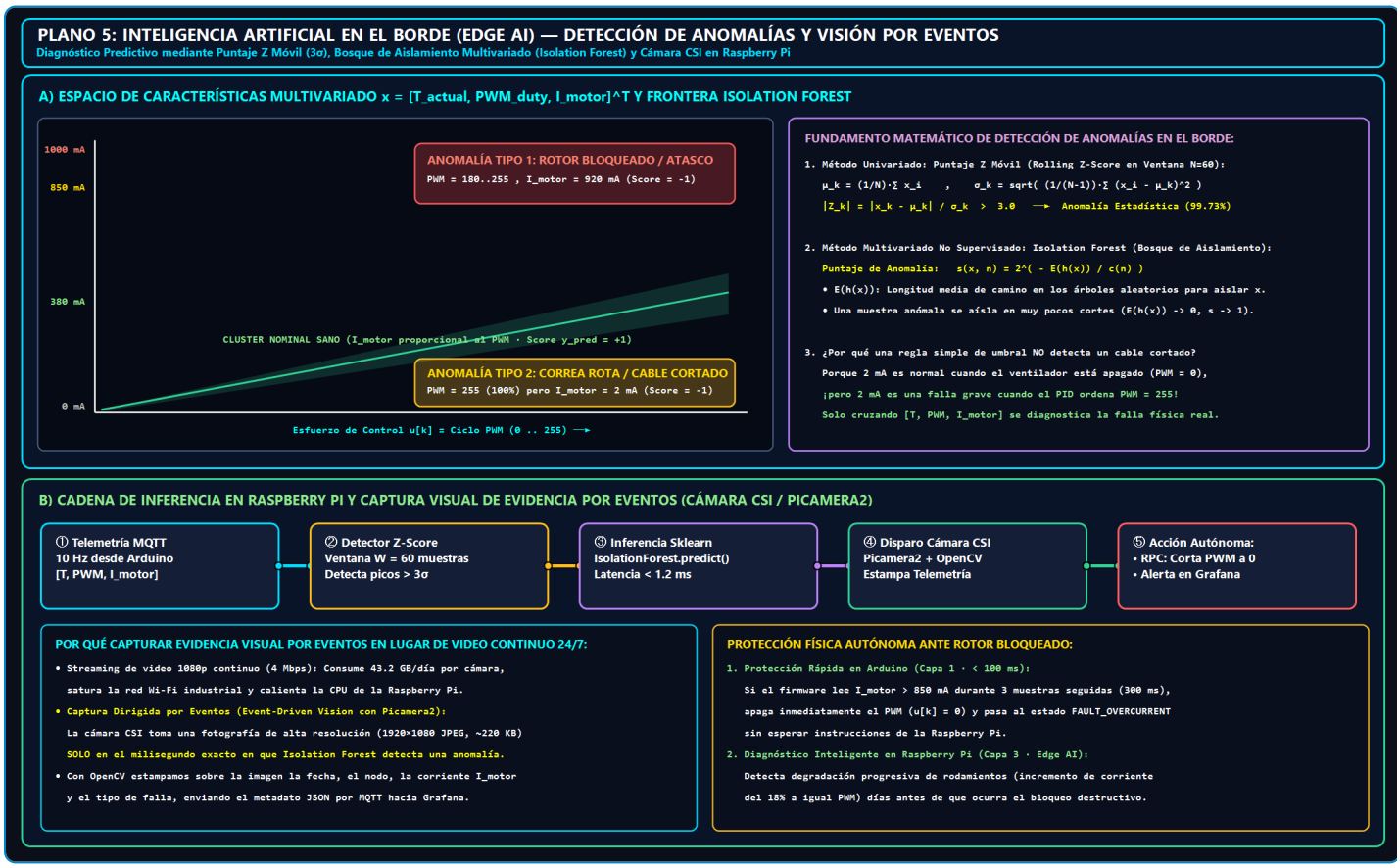
Procedimiento Remoto (`RPC`) de grado industrial:

- **Tópicos desacoplados de solicitud y respuesta:** `planta/nodo_01/rpc/req` y `planta/nodo_01/rpc/res` con `QoS = 1` y `Retain = False`.
- **Correlación estricta por `req_id` (`UUID`):** cada solicitud inyecta un identificador único (ej. `"req_id": "req-8f3a"`). El hilo invocador registra un objeto `threading.Event()` en un diccionario protegido por cerrojo (`Lock`) y se suspende con `event.wait(timeout=1.50)`.
- **Idempotencia y Reversión (`Rollback`):** si el temporizador de $1500\ ms$ expira tras los reintentos configurados, el cliente supervisor revierte el control deslizante en la interfaz HMI al último valor confirmado por el hardware y emite una alerta de fallo de comunicación.

---

5. Inteligencia Artificial en el Borde (`Edge AI`) y Visión por Eventos (`Plano 5`)

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos



5.1 Puntaje Z Móvil (*Rolling Z-Score*) sobre Ventana Deslizante ($N = 60$)

Para detectar picos abruptos en una serie temporal univariada, calculamos en tiempo real la media μ_k y la desviación estándar muestral σ_k:

$$\mu_k = \frac{1}{N}\sum_{i=k-N+1}^{k} x_i \quad , \quad \sigma_k = \sqrt{\frac{1}{N-1}\sum_{i=k-N+1}^{k}\left(x_i - \mu_k\right)^2} \quad , \quad Z_k = \frac{|x_k - \mu_k|}{\sigma_k}$$

Toda lectura con $Z_k > 3.0$ se desvía más de tres desviaciones estándar de su comportamiento reciente ($p < 0.0027$).

5.2 Detección Multivariada No Supervisada con `Isolation Forest`

Una regla simple de umbral sobre la corriente I_{motor} **no puede detectar una correa rota ni un cable cortado**, porque $I_{motor} = 2\ mA$ es completamente normal cuando el ventilador está apagado ($PWM = 0$). Sin embargo, $I_{motor} = 2\ mA$

cuando el regulador `PID` ordena $\mathrm{PWM} = 255$ constituye una falla crítica de lazo abierto.

Al alimentar el algoritmo **Bosque de Aislamiento (`Isolation Forest`)** con el vector de estado tridimensional:

$$\mathbf{x}_k = \begin{bmatrix} T_{actual}[k] \\ PWM_{duty}[k] \\ I_{motor}[k] \end{bmatrix}$$

el conjunto de 100 árboles aleatorios particiona el espacio de características y calcula el puntaje de anomalía:

$$s(\mathbf{x}, n) = 2^{-\frac{E(h(\mathbf{x}))}{c(n)}}$$

donde $E(h(\mathbf{x}))$ es la longitud media de camino necesaria para aislar la muestra $\mathbf{x}$. Las muestras anómalas (tanto el **rotor bloqueado** con $I_{motor} > 850\ mA$ como la **correa rota** con $PWM{=}255$ e $I_{motor} \approx 0\ mA$) quedan aisladas cerca de la raíz de los árboles ($E(h(\mathbf{x})) \to 0 \implies s \to 1$), disparando en menos de $1.2\ ms$ la captura de una fotografía forense `1080p` con la cámara `CSI` y el apagado preventivo del actuador.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 08

Planos de Alta Legibilidad en Protoboard (`Tinkercad`), Circuito Impreso Soldado (`PCB FR-4`), Lazo `PID` , Secuencia `RPC` y `Edge AI`

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado (`PID`), Protocolo `RPC` , `Edge AI` y Visión por Eventos

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje en Protoboard del Lazo Cerrado `PID`

Este primer plano presenta el montaje experimental en protoboard con tipografía ampliada de alta legibilidad y llamadas numeradas (① a ⑤), interconectando el **Arduino UNO R3**, la etapa de potencia `MOSFET IRLZ44N` con diodo volante `1N4007` , el sensor de corriente de lado alto `INA219 (0x40)` y la sonda térmica `BME280 (0x76)` .

📐 Plano 1: Montaje en Protoboard del Lazo de Control PID con MOSFET IRLZ44N e INA219

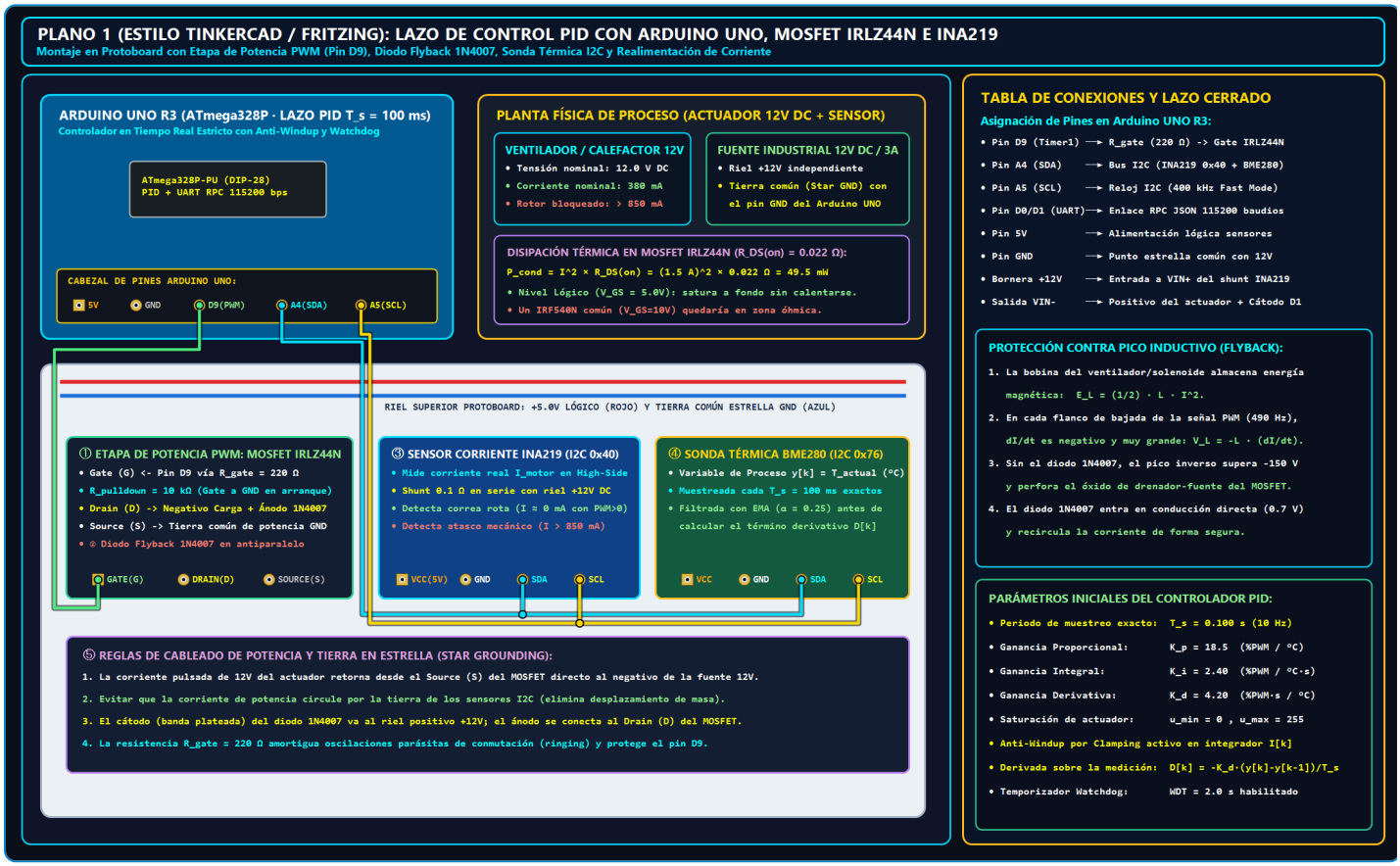


Tabla de Asignación de Terminales y Funciones Físicas

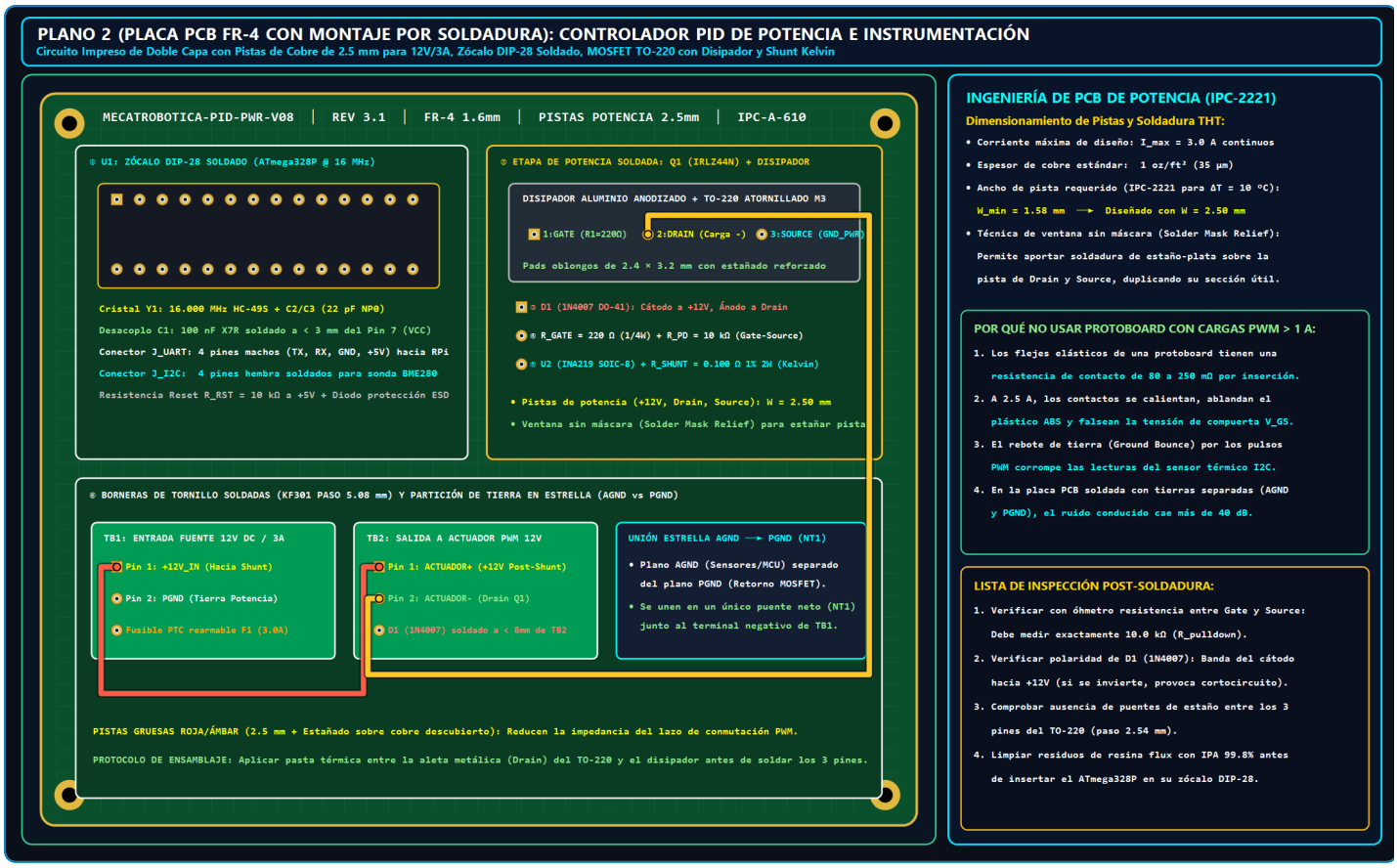
Origen (`Arduino / Fuente`)	Destino en Etapa de Campo	Color de Cable	Función Eléctrica y Regla de Diseño
`Pin D9` (`Timer1 PWM`)	Compuerta (`Gate`) del `IRLZ44N` vía $R_{gate} = 220\ \Omega$	Verde	Señal `PWM` de $490\ Hz$ (0 a 255). Incluye $R_{pulldown} = 10\ k\Omega$ entre `Gate` y `GND`.
`Pin A4` (`SDA`)	Pines `SDA` de `INA219 (0x40)` y `BME280 (0x76)`	Cian	Línea de datos serie `I2C` operando a $400\ kHz$ (*Fast Mode*).
`Pin A5` (`SCL`)	Pines `SCL` de `INA219 (0x40)` y `BME280 (0x76)`	Amarillo	Línea de reloj serie `I2C` sincronizada por el `ATmega328P`.
`Fuente +12V DC`	Borne `VIN+` del `Shunt` de $0.1\ \Omega$ (`INA219`)	Rojo	Entrada de potencia de lado alto (*High-Side*) protegida por fusible `PTC` de $3\ A$.
Borne `VIN-` (`INA219`)	Positivo del Actuador $12\ V$ + Cátodo de `1N4007`	Ámbar	Alimentación medida hacia el actuador y punto de sujeción del diodo de rueda libre.
Negativo del Actuador $12\ V$	Drenador (`Drain`) del `IRLZ44N` + Ánodo de `1N4007`	Amarillo	Conmutación de lado bajo (*Low-Side*) y absorción del pico inductivo inverso.
Surtidor (`Source`) `IRLZ44N`	Punto Estrella `GND` de la Fuente de $12\ V\ DC$	Azul / Negro	Retorno directo de corriente pulsada sin contaminar la tierra analógica de los sensores.

---

2. Plano 2 (Placa `PCB FR-4` con Montaje por Soldadura): Controlador `PID` de Potencia

Para aplicaciones permanentes donde las corrientes pulsadas `PWM` de 1 a $3\ A$ degradarían los contactos elásticos de un protoboard, este segundo plano detalla el diseño de una **placa de circuito impreso (`PCB FR-4`) de doble capa con montaje por soldadura (`THT` / `SMD`)**, pistas de potencia de $2.50\ mm$ con ventana de estañado (*Solder Mask Relief*) y partición de tierras en estrella (`AGND` vs. `PGND`).

📐 Plano 2: Placa PCB FR-4 de Doble Capa con Montaje por Soldadura para Controlador PID

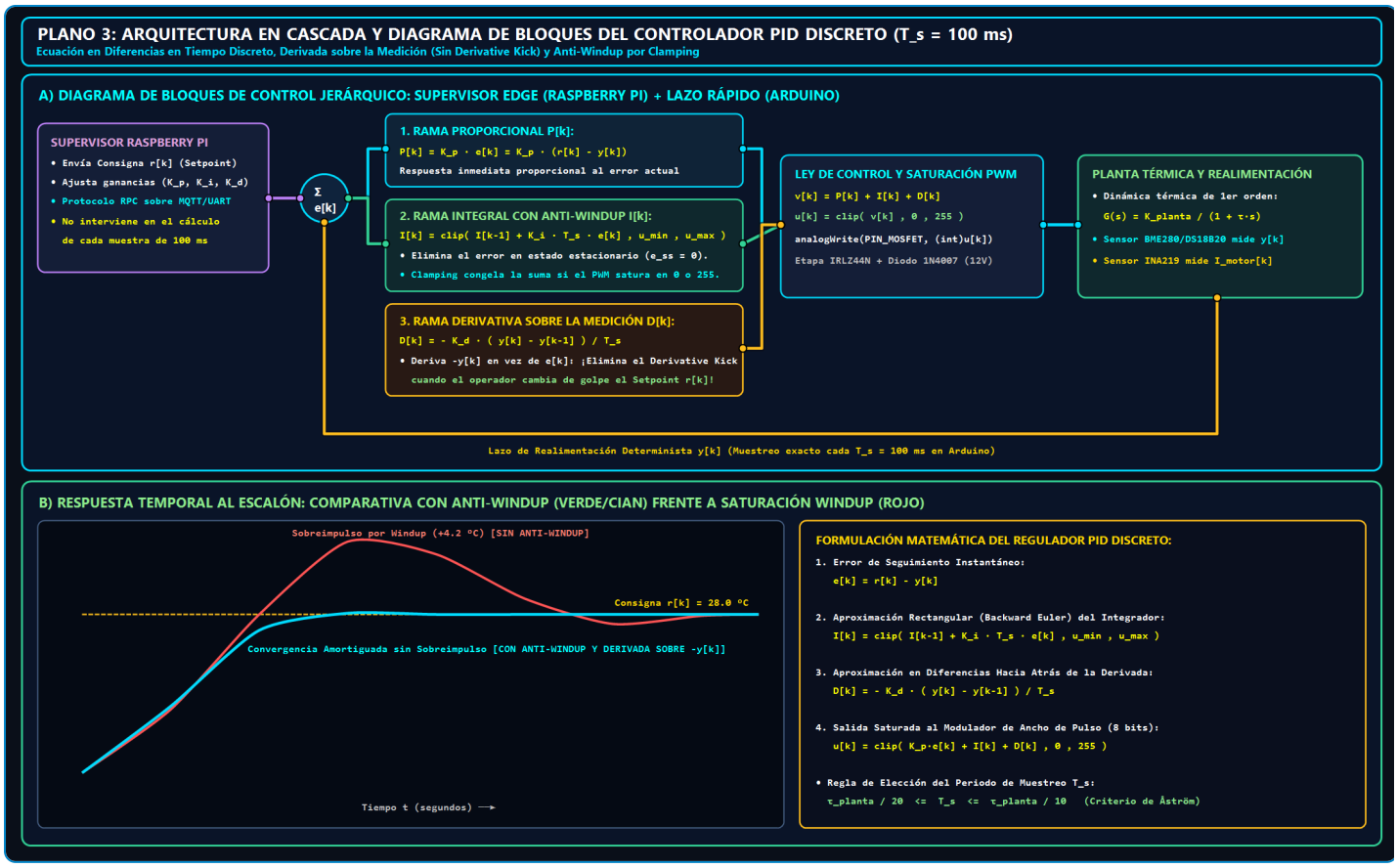


---

3. Plano 3: Arquitectura en Cascada y Diagrama de Bloques del Regulador `PID` Discreto

Detalla la estructura matemática de las tres ramas paralelas ($P[k]$, $I[k]$ con *Clamping Anti-Windup*, y $D[k]$ calculada sobre $-y[k]$) junto con las curvas comparativas de respuesta temporal al escalón.

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup

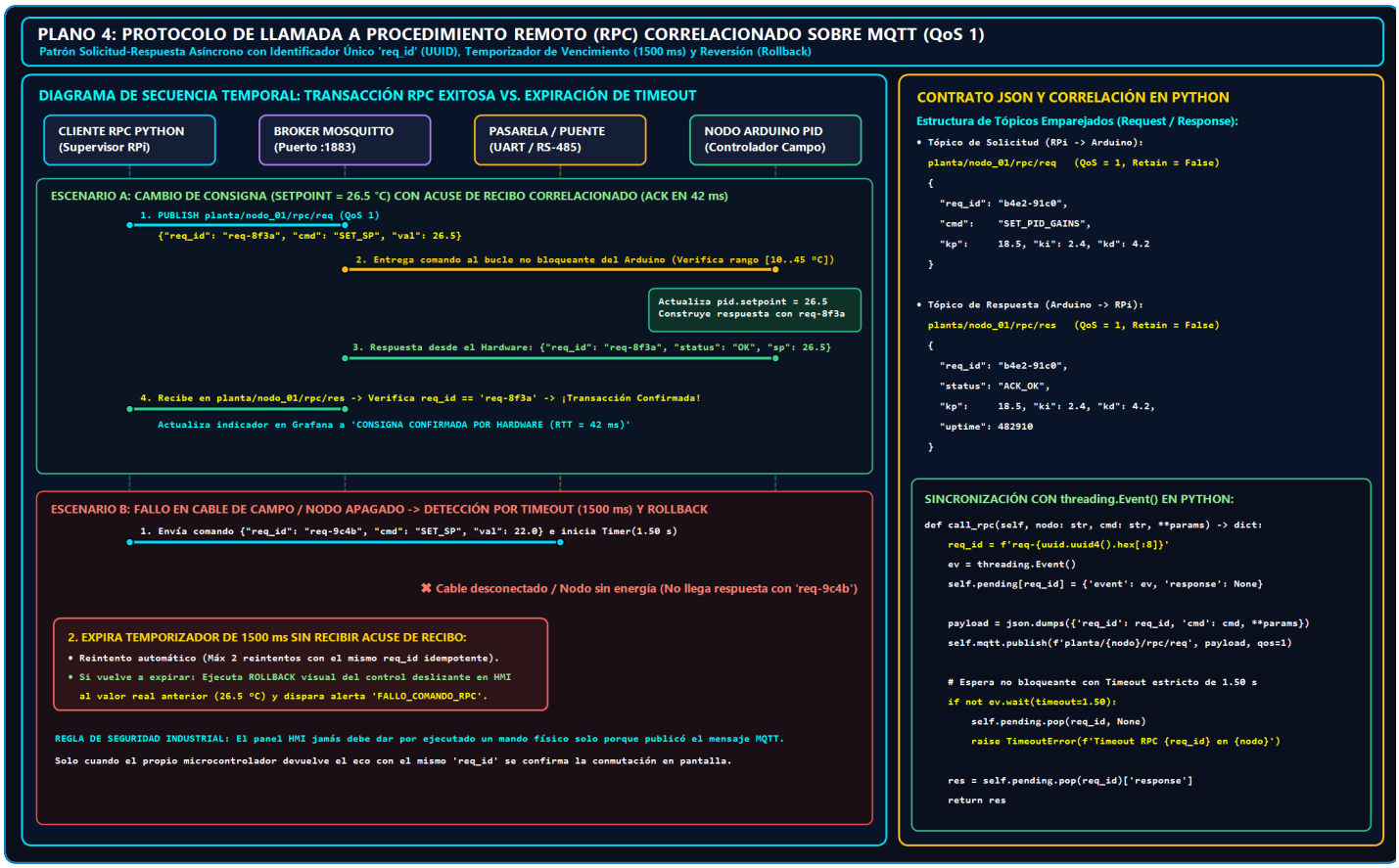


---

4. Plano 4: Diagrama de Secuencia del Protocolo `RPC` Correlacionado sobre `MQTT`

Representa el flujo temporal completo de una transacción exitosa con acuse de recibo (`ACK`) correlacionado en $42\ ms$ frente a un escenario de corte de cable detectado por expiración de temporizador ($1500\ ms$) con reversión automática (`Rollback`).

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



---

5. Plano 5: Espacio de Características `Isolation Forest` y Visión por Eventos (`CSI`)

Ilustra la separación geométrica en el espacio tridimensional $[T_{actual}, PWM, I_{motor}]$ entre el corredor nominal sano, la región de rotor bloqueado y la región de correa rota/cable abierto, así como la cadena de captura fotográfica `1080p` con `Picamera2` y `OpenCV`.

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos

PLANO 5: INTELIGENCIA ARTIFICIAL EN EL BORDE (EDGE AI) — DETECCIÓN DE ANOMALÍAS Y VISIÓN POR EVENTOS

Diagnóstico Predictivo mediante Puntaje Z Móvil (3σ), Bosque de Aislamiento Multivariado (Isolation Forest) y Cámara CSI en Raspberry Pi

A) ESPACIO DE CARACTERÍSTICAS MULTIVARIADO x = [T_actual, PWM_duty, I_motor]^T Y FRONTERA ISOLATION FOREST



FUNDAMENTO MATEMÁTICO DE DETECCIÓN DE ANOMALÍAS EN EL BORDE:

1. Método Univariado: Puntaje Z Móvil (Rolling Z-Score en Ventana N=60):

   $\mu_k = (1/N)\cdot\Sigma\ x_i$, $\sigma_k = \text{sqrt}((1/(N-1))\cdot\Sigma\ (x_i - \mu_k)^2)$

   $|Z_k| = |x_k - \mu_k| / \sigma_k > 3.0 \longrightarrow$ Anomalía Estadística (99.73%)

2. Método Multivariado No Supervisado: Isolation Forest (Bosque de Aislamiento):

   Puntaje de Anomalía: $s(x, n) = 2^{(- E(h(x)) / c(n))}$

   • E(h(x)): Longitud media de camino en los árboles aleatorios para aislar x.

   • Una muestra anómala se aísla en muy pocos cortes (E(h(x)) -> 0, s -> 1).

3. ¿Por qué una regla simple de umbral NO detecta un cable cortado?

   Porque 2 mA es normal cuando el ventilador está apagado (PWM = 0),

   ¡pero 2 mA es una falla grave cuando el PID ordena PWM = 255!

   Solo cruzando [T, PWM, I_motor] se diagnostica la falla física real.

B) CADENA DE INFERENCIA EN RASPBERRY PI Y CAPTURA VISUAL DE EVIDENCIA POR EVENTOS (CÁMARA CSI / PICAMERA2)

① Telemetría MQTT
10 Hz desde Arduino
[T, PWM, I_motor]

② Detector Z-Score
Ventana W = 60 muestras
Detecta picos > 3σ

③ Inferencia Sklearn
IsolationForest.predict()
Latencia < 1.2 ms

④ Disparo Cámara CSI
Picamera2 + OpenCV
Estampa Telemetría

⑤ Acción Autónoma:
• RPC: Corta PWM a 0
• Alerta en Grafana

POR QUÉ CAPTURAR EVIDENCIA VISUAL POR EVENTOS EN LUGAR DE VIDEO CONTINUO 24/7:

• Streaming de video 1080p continuo (4 Mbps): Consume 43.2 GB/día por cámara, satura la red Wi-Fi industrial y calienta la CPU de la Raspberry Pi.

• Captura Dirigida por Eventos (Event-Driven Vision con Picamera2):
  La cámara CSI toma una fotografía de alta resolución (1920×1080 JPEG, ~220 KB) SOLO en el milisegundo exacto en que Isolation Forest detecta una anomalía.

• Con OpenCV estampamos sobre la imagen la fecha, el nodo, la corriente I_motor y el tipo de falla, enviando el metadato JSON por MQTT hacia Grafana.

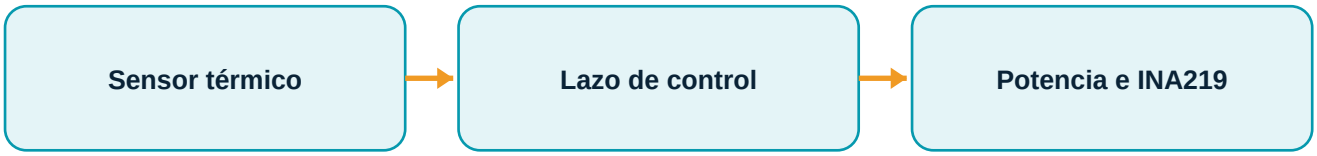
PROTECCIÓN FÍSICA AUTÓNOMA ANTE ROTOR BLOQUEADO:

1. Protección Rápida en Arduino (Capa 1 · < 100 ms):
   Si el firmware lee I_motor > 850 mA durante 3 muestras seguidas (300 ms), apaga inmediatamente el PWM (u[k] = 0) y pasa al estado FAULT_OVERCURRENT sin esperar instrucciones de la Raspberry Pi.
2. Diagnóstico Inteligente en Raspberry Pi (Capa 3 · Edge AI):
   Detecta degradación progresiva de rodamientos (incremento de corriente del 18% a igual PWM) días antes de que ocurra el bloqueo destructivo.

Capítulo 26 · Proyecto: control térmico, INA219 y comandos RPC

Conceptos clave

INA219 mide corriente y tensión, no temperatura. Valida el rango de la referencia recibida por RPC y conserva límites de seguridad locales.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 08

Control en Lazo Cerrado (`PID` Discreto), Llamadas a Procedimiento Remoto (`RPC` sobre `MQTT`) e Inteligencia Artificial en el Borde (`Edge AI`)

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Regulación `PID` en Tiempo Discreto, Protocolo `RPC` Correlacionado, `Isolation Forest` y Visión por Eventos

Duración estimada de laboratorio: 4 horas prácticas

---

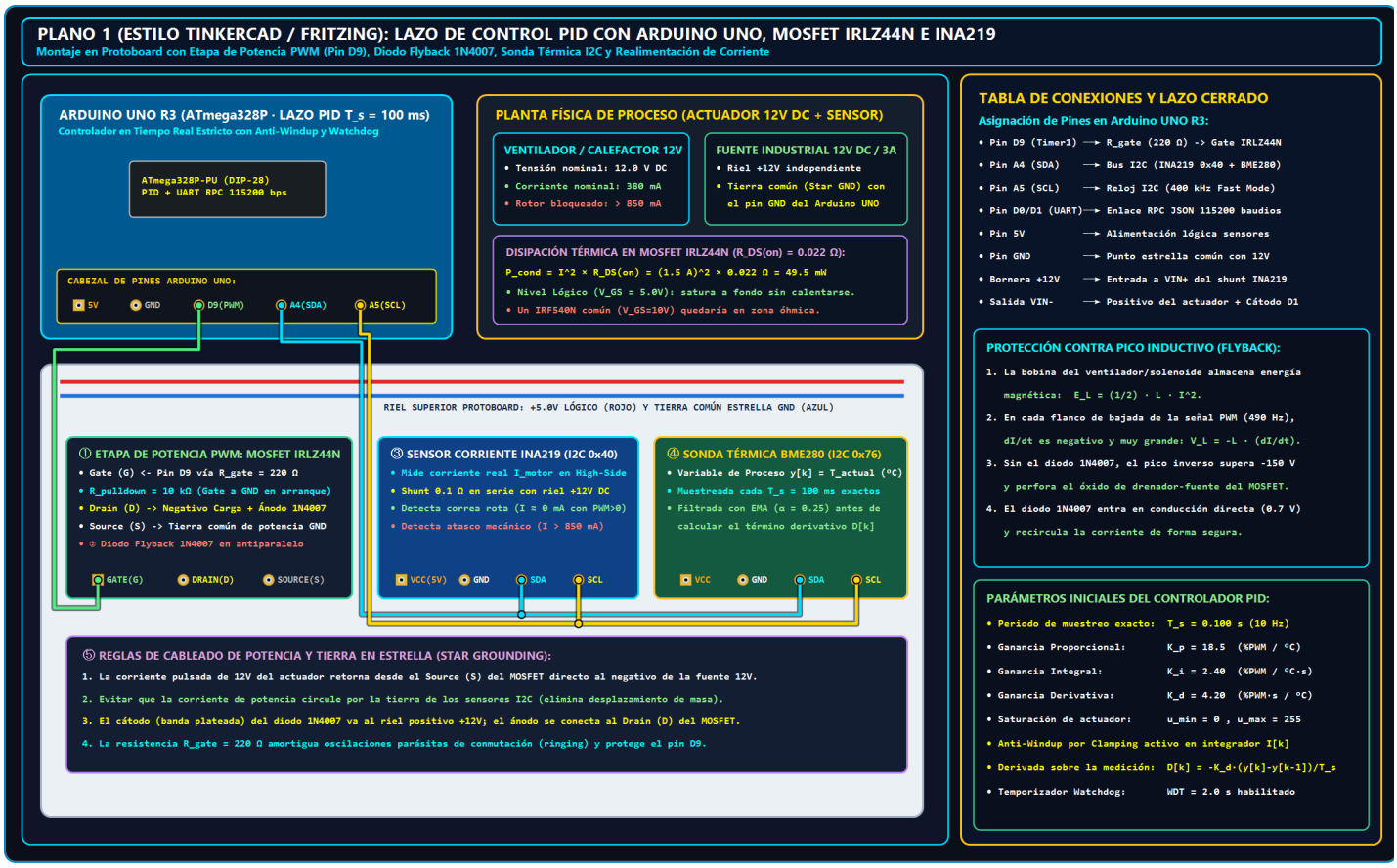
1. Objetivos de Ingeniería del Laboratorio

- **Separar la jerarquía de control en tiempo real estricto (`Hard Real-Time`) de la supervisión analítica en el borde (`Soft Real-Time`):** ejecutar el lazo cerrado proporcional-integral-derivativo (`PID`) cada $T_s = 100\ ms$ en el microcontrolador **Arduino UNO R3 (`ATmega328P`)**, delegando el ajuste de consigna (`Setpoint`), la inferencia de anomalías y la visualización a la **Raspberry Pi 4 / 5**.
- **Construir la etapa de potencia en modulación por ancho de pulso (`PWM`) con transistor `MOSFET` de nivel lógico (`IRLZ44N`):** calcular la disipación térmica por conducción, instalar la resistencia limitadora de compuerta ($220\ \Omega$), la resistencia de anclaje a tierra ($10\ k\Omega$) y el diodo volante de rueda libre (`1N4007`) en antiparalelo con la carga inductiva de $12\ V\ DC$.
- **Implementar ambos planos de construcción electrónica:** el prototipo experimental en **protoboard (estilo Tinkercad / Fritzing)** con tipografía ampliada de alta legibilidad y la **placa de circuito impreso (`PCB FR-4` de doble capa) con montaje por soldadura** y pistas de potencia de $2.50\ mm$.
- **Programar un protocolo de Llamada a Procedimiento Remoto (`RPC`) correlacionado sobre `MQTT` (`QoS 1`):** emparejar solicitudes y respuestas mediante un identificador único universal (`req_id`), sincronizar hilos con `threading.Event()` , establecer un temporizador de expiración de $1500\ ms$ y aplicar reversión segura (`Rollback`) en la interfaz gráfica si el hardware no confirma la orden.
- **Desplegar Inteligencia Artificial en el Borde (`Edge AI`) y captura visual dirigida por eventos:** ejecutar detección estadística por puntaje Z móvil (`Rolling Z-Score` 3σ) e inferencia multivariada con **Bosque de Aislamiento (`Isolation Forest`)** sobre el vector $\mathbf{x} = [T_{actual}, PWM, I_{motor}]^T$, disparando la cámara `CSI` (`Picamera2` + `OpenCV`) en el milisegundo exacto de una anomalía.

---

2. Arquitectura Física del Lazo Cerrado y Etapa de Potencia (`Plano 1`)

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad del Lazo de Control PID con MOSFET IRLZ44N e INA219



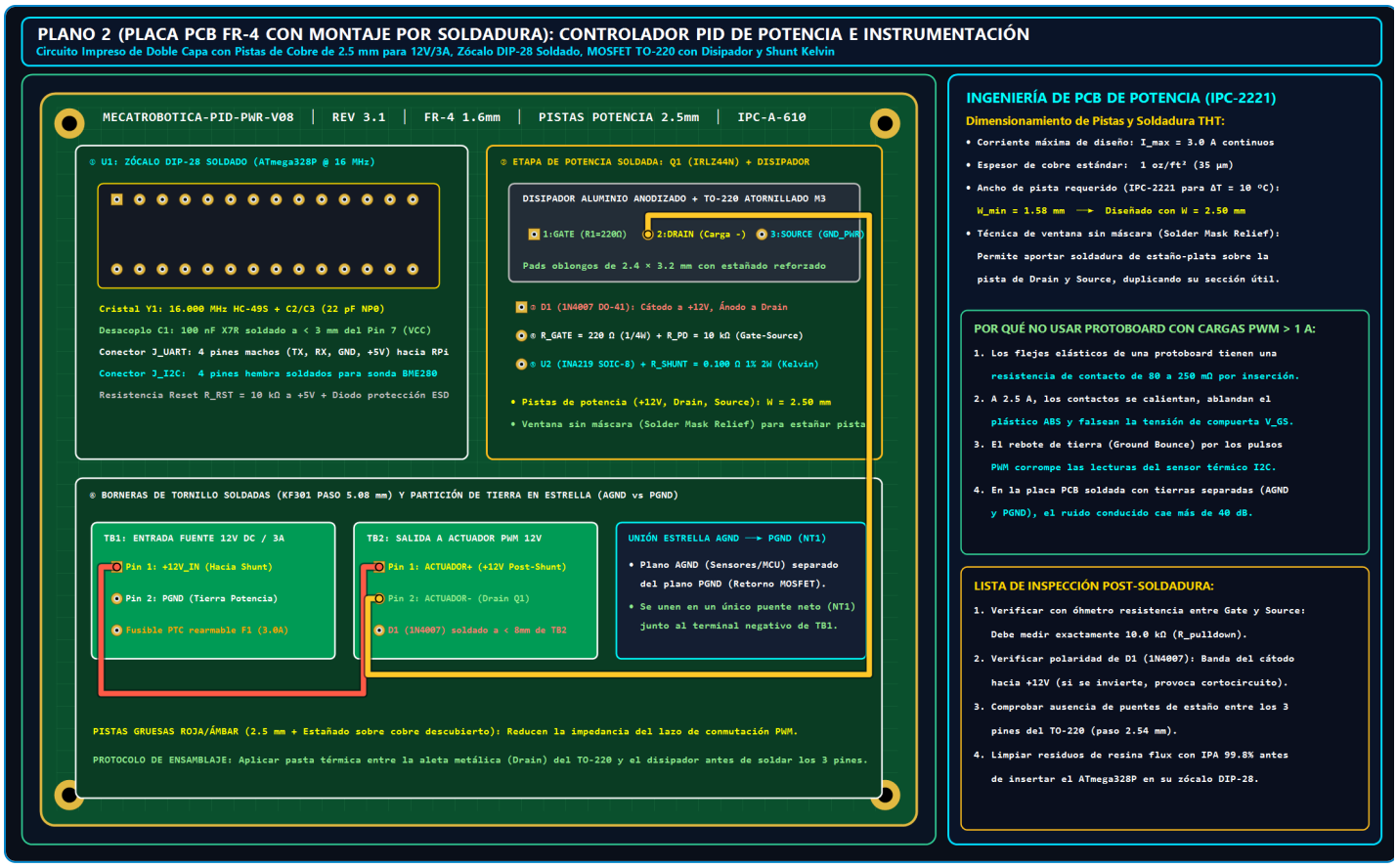
Paso 2.1 — Reglas Críticas de Cableado de Potencia y Tierra en Estrella (`Star Grounding`)

- **Por qué es obligatorio un `MOSFET` de nivel lógico (`IRLZ44N`):** el pin `D9` del Arduino UNO entrega $5.0\ \text{V}$ (y las placas de $3.3\ \text{V}$ entregan aún menos). Un transistor `MOSFET` estándar como el `IRF540N` requiere $V_{GS} = 10\ \text{V}$ para saturar completamente; si se excita con $5\ \text{V}$, permanece en su región lineal/óhmica y se destruye por sobrecalentamiento. El `IRLZ44N` posee tensión umbral $V_{GS(th)} \in [1.0, 2.0]\ \text{V}$ y alcanza $R_{DS(on)} = 0.022\ \Omega$ con $V_{GS} = 5.0\ \text{V}$.
- **Diodo de rueda libre (`Flyback 1N4007`):** conéctelo directamente entre los terminales del actuador inductivo (ventilador o válvula solenoide de $12\ \text{V}$), con el **cátodo (banda plateada)** hacia el riel positivo de $+12\ \text{V}$ y el **ánodo** hacia el drenador (`Drain`) del `IRLZ44N`.
- **Tierra en estrella (`Star Ground`):** el surtidor (`Source`) del `IRLZ44N` debe retornar directamente por un cable grueso al borne negativo de la fuente de $12\ \text{V DC}$, evitando que los pulsos de corriente de conmutación `PWM` circulen por la pista de tierra de los sensores `I2C` (`BME280` e `INA219`).

---

3. Montaje Definitivo en Placa de Circuito Impreso Soldada (`Plano 2`)

📐 Plano 2: Placa PCB FR-4 de Doble Capa con Montaje por Soldadura para Controlador PID de Potencia



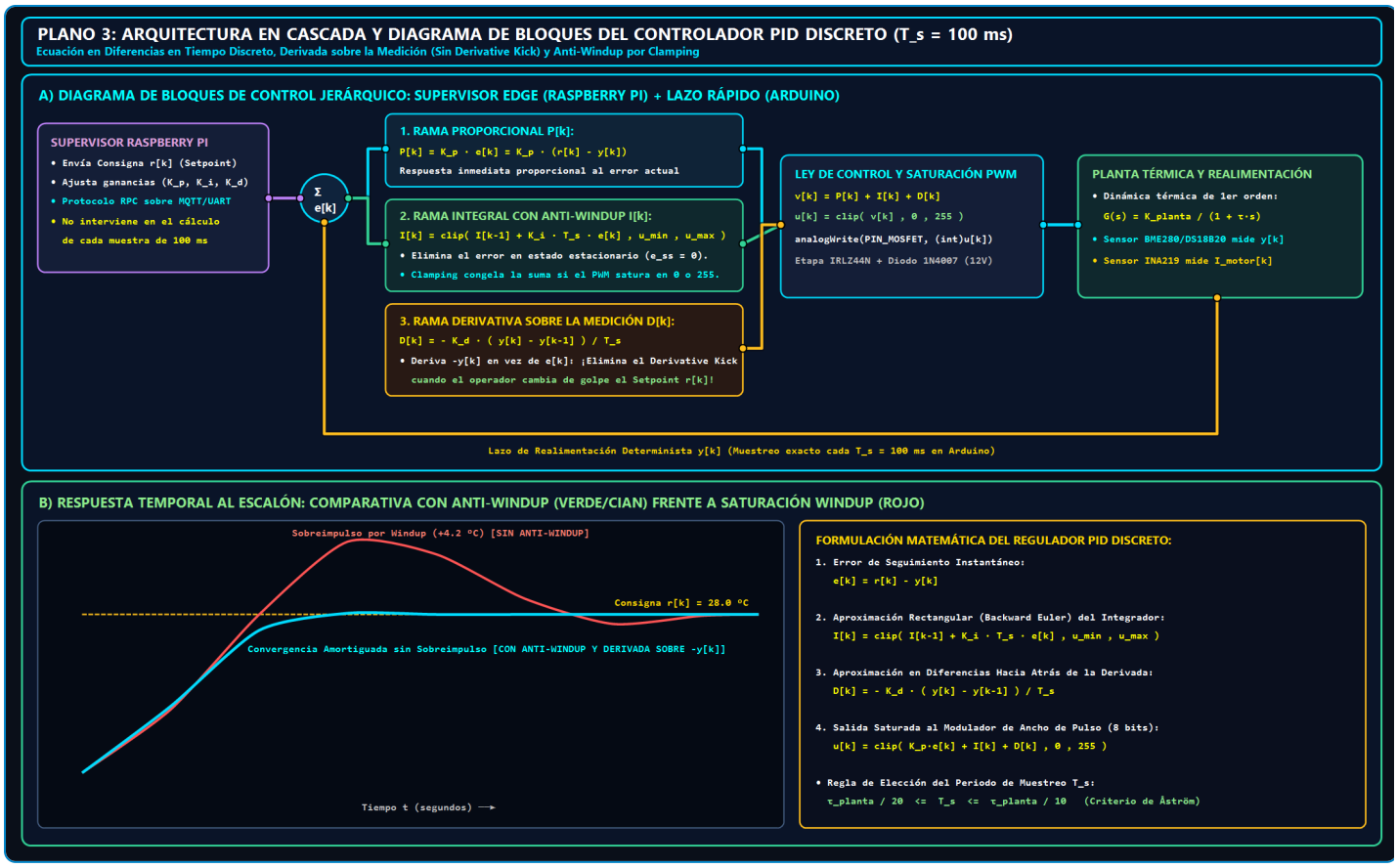
Paso 3.1 — Procedimiento de Soldadura de Potencia según `IPC-2221` e `IPC-A-610`

- Suelde primero los componentes de bajo perfil: resistencias de película metálica ($220\ \Omega$, $10\ k\Omega$), condensadores cerámicos `X7R` de $100\ nF$, el cristal de $16.000\ MHz$ y el sensor de corriente `INA219` junto a su resistencia `Shunt` Kelvin de $0.100\ \Omega$ (1%, $2\ W$).
- Suelde el **zócalo torneado** `DIP-28` para el microcontrolador `ATmega328P-PU` sin el chip insertado.
- Atornille la aleta metálica del `MOSFET IRLZ44N` (`TO-220`) al disipador de aluminio anodizado aplicando pasta térmica y suelde sus tres terminales en los pads oblongos reforzados.
- Suelde las borneras de tornillo `TB1` (entrada $12\ V\ DC$) y `TB2` (salida al actuador) y aporte una capa uniforme de aleación `SAC305` sobre las ventanas descubiertas de máscara (*Solder Mask Relief*) en las pistas de $2.50\ mm$.

---

4. Discretización del Controlador `PID` y Anti-Windup (`Plano 3`)

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup



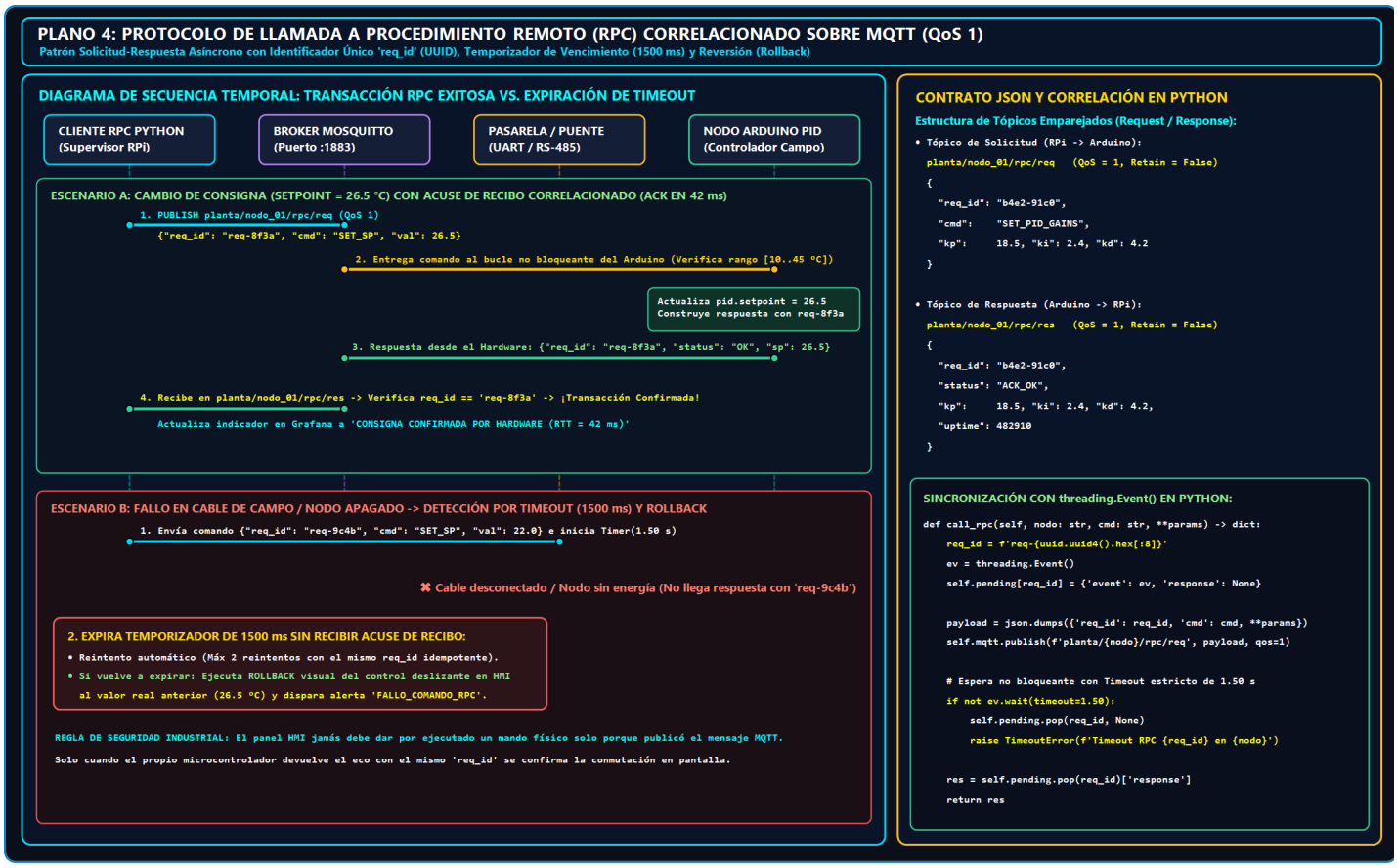
Paso 4.1 — Compilación y Carga del Firmware Determinista en Arduino UNO

- Abra el archivo `codigo_fuente/controlador_pid_discreto_rpc_arduino.ino` en Arduino IDE o PlatformIO.
- Verifique las tres mejoras industriales implementadas en la función `computeDiscretePID()`:
- **Muestreo exacto por temporizador no bloqueante:** $T_s = 100\ ms$ $(10\ Hz)$.
- **Derivada sobre la medición (**$-y[k]$**):** $D[k] = -K_d \cdot (y[k] - y[k-1]) / T_s$, eliminando el golpe derivativo (*Derivative Kick*) cuando el operador modifica de golpe el `Setpoint` desde Grafana.
- **Anti-Windup por recorte condicional (*Clamping*):** congela la acumulación del integrador $I[k]$ cuando la salida `PWM` alcanza 0 o 255.

---

5. Protocolo `RPC` Correlacionado sobre `MQTT` (`Plano 4`)

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



Paso 5.1 — Prueba de Transacción Confirmada y Prueba de Fallo con `Rollback`

- Ejecute el cliente supervisor en la Raspberry Pi:

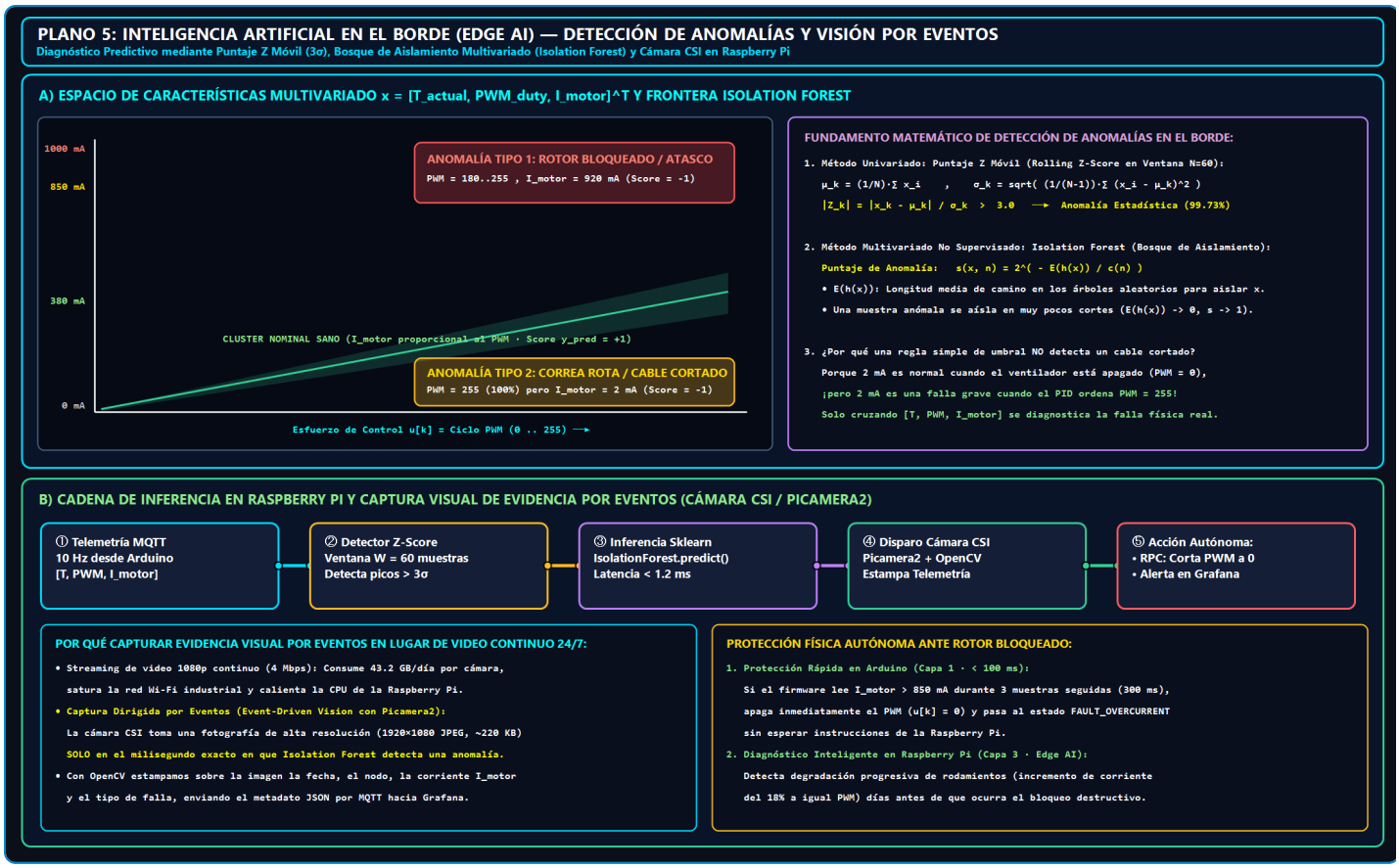
  ```bash
  python3 codigo_fuente/cliente_rpc_correlacionado_mqtt.py
  ```

- Verifique en consola que cada comando enviado en `planta/nodo_01/rpc/req` incluye un `req_id` único (por ejemplo, `req-8f3a19c2`) y que el hilo espera en `threading.Event.wait(timeout=1.50)` hasta recibir exactamente ese mismo `req_id` en `planta/nodo_01/rpc/res`.
- Desconecte momentáneamente el cable serial del Arduino y envíe una nueva consigna: compruebe cómo tras 1.50 s y 2 reintentos idempotentes, el servicio publica el evento de `Rollback` restaurando el indicador de la interfaz al valor real anterior.

---

6. Detección de Anomalías con `Edge AI` y Cámara `CSI` (`Plano 5`)

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos



Paso 6.1 — Ejecución del Detector `Isolation Forest` y Captura Forense por Eventos

- Inicie en terminales separadas el motor de inferencia y el servicio de cámara `CSI`:

  ```bash
  python3 codigo_fuente/detector_anomalias_edge_ai_isolation_forest.py
  python3 codigo_fuente/captura_visual_eventos_camara_csi.py
  ```

- **Prueba de Anomalía Tipo 1 (Rotor Bloqueado):** frene suavemente el eje del ventilador durante $400\ ms$. Observe cómo la corriente I_{motor} supera los $850\ mA$, el Arduino corta autónomamente el `PWM` a 0 en menos de $300\ ms$ y el motor `Edge AI` clasifica `ROTOR_BLOQUEADO_SOBRECORRIENTE`, disparando una fotografía `1080p` con marca de agua en `/var/lib/mecatrobotica/evidencia_visual/`.
- **Prueba de Anomalía Tipo 2 (Correa Rota / Cable Abierto):** desconecte el hilo positivo del ventilador mientras el controlador `PID` ordena $\text{PWM} = 255$. Aunque $I_{motor} \approx 2\ mA$ no viola ningún límite de sobrecorriente, el modelo multivariado **`Isolation Forest`** detecta inmediatamente que la combinación $[\text{PWM}=255,\ I_{motor}=2\ mA]$ se ubica fuera de la variedad nominal ($\text{Score} = -1$) y reporta `CORREA_ROTA_O_CABLE_ACTUADOR_ABIERTO`.

LISTADO DE MATERIALES E INSTRUMENTACIÓN (`BOM`) — MÓDULO 08

Hardware de Control en Lazo Cerrado `PID` , Etapa de Potencia `MOSFET PWM` , Sensórica `I2C` y Visión en el Borde (`CSI`)

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado (`PID`), Protocolo `RPC` , `Edge AI` y Captura Visual por Eventos

---

1. Computación de Supervisión (`Edge AI`) y Controlador de Tiempo Real (`PID`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
01	1	`SBC1`	**Raspberry Pi 4 Modelo B / Raspberry Pi 5 (`4 GB` o `8 GB` RAM)**	Supervisor de planta, cliente `RPC` correlacionado, motor `Isolation Forest` y captura de cámara `CSI` .
02	1	`U1`	**Arduino UNO R3 (`ATmega328P-PU` @ `16 MHz`)**	Ejecución determinista del lazo cerrado `PID` discreto cada $T_s = 100\ ms$ y protección de sobrecorriente.
03	1	`CAM1`	**Cámara Oficial Raspberry Pi `CSI` (`IMX219 8MP` o `IMX708 12MP`) con cinta `FFC` de 15/22 pines**	Captura fotográfica `1920x1080` dirigida por eventos ante detección de anomalías mecánicas o térmicas.

---

2. Etapa de Potencia `PWM` de $12\ V\ DC$ y Protección contra Transitorios Inductivos

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
04	1	`Q1`	**Transistor** `MOSFET` **Canal N de Nivel Lógico** `IRLZ44N` **(** `TO-220` **,** $V_{DSS}=55\ V$**,** $I_D=47\ A$**,** $R_{DS(on)}=0.022\ \Omega$ **@** $V_{GS}=5\ V$**)**	Conmutador de potencia en lado bajo (*Low-Side Switch*) gobernado por la señal `PWM` del pin `D9` ($490\ Hz$).
05	1	`HS1`	**Disipador de Aluminio Anodizado para cápsula** `TO-220` **(**$19 \times 15 \times 25\ mm$**) con tornillo** `M3` **y pasta térmica**	Evacuación térmica de pérdidas por conducción y conmutación del transistor `Q1`.
06	1	`D1`	**Diodo Rectificador de Rueda Libre** `1N4007` **(** `DO-41` **,** $1000\ V$**,** $1.0\ A$**) o** `1N5819` **(** `Schottky` $40\ V$**)**	Supresión del pico inductivo inverso ($V_L = -L \cdot dI/dt$) en cada corte del ciclo `PWM`.
07	1	`R_GATE`	**Resistencia de Película Metálica de** $220\ \Omega$ **(**$1/4\ W$**,** $\pm 1\%$**)**	Limita el pico de corriente de carga de la capacitancia de compuerta (C_{iss}) y amortigua oscilaciones parásitas (*ringing*).
08	1	`R_PD`	**Resistencia de Anclaje (** `Pull-Down` **) de** $10\ k\Omega$ **(**$1/4\ W$**,** $\pm 1\%$**)**	Mantiene la compuerta (`Gate`) firmemente en $0\ V$ durante el reinicio o alta impedancia del microcontrolador.
09	1	`ACT1`	**Planta Térmica / Ventilador Axial Industrial de** $12\ V\ DC$ **(**$380\ mA$ **nominal,** $> 850\ mA$ **rotor bloqueado)**	Actuador físico del lazo de regulación térmica y banco de pruebas de anomalías mecánicas.
10	1	`PSU1`	**Fuente Conmutada Industrial de** $12\ V\ DC\ /\ 3.0\ A$ **(**$36\ W$**) con riel negativo común en estrella**	Alimentación dedicada para la etapa de potencia de $12\ V$, desacoplada del riel lógico de $5\ V$.

3. Sensórica de Realimentación de Estado (`T_actual` e `I_motor`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
11	1	`U2`	**Módulo Sensor de Corriente y Tensión de Lado Alto** `INA219` **(** `I2C 0x40` **) con Shunt Kelvin de** $0.100\ \Omega$ **(**1%**,** $2\ W$**)**	Medición en tiempo real de $I_{motor}[k]$ y $V_{bus}[k]$ para protección de sobrecorriente y vector de `Edge AI`.
12	1	`U3`	**Sonda Ambiental Digital** `Bosch BME280` **(** `I2C 0x76` **) o** `DS18B20` **impermeable**	Medición de la variable controlada $y[k] = T_{actual}$ con resolución de $0.01\ ^\circ C$.

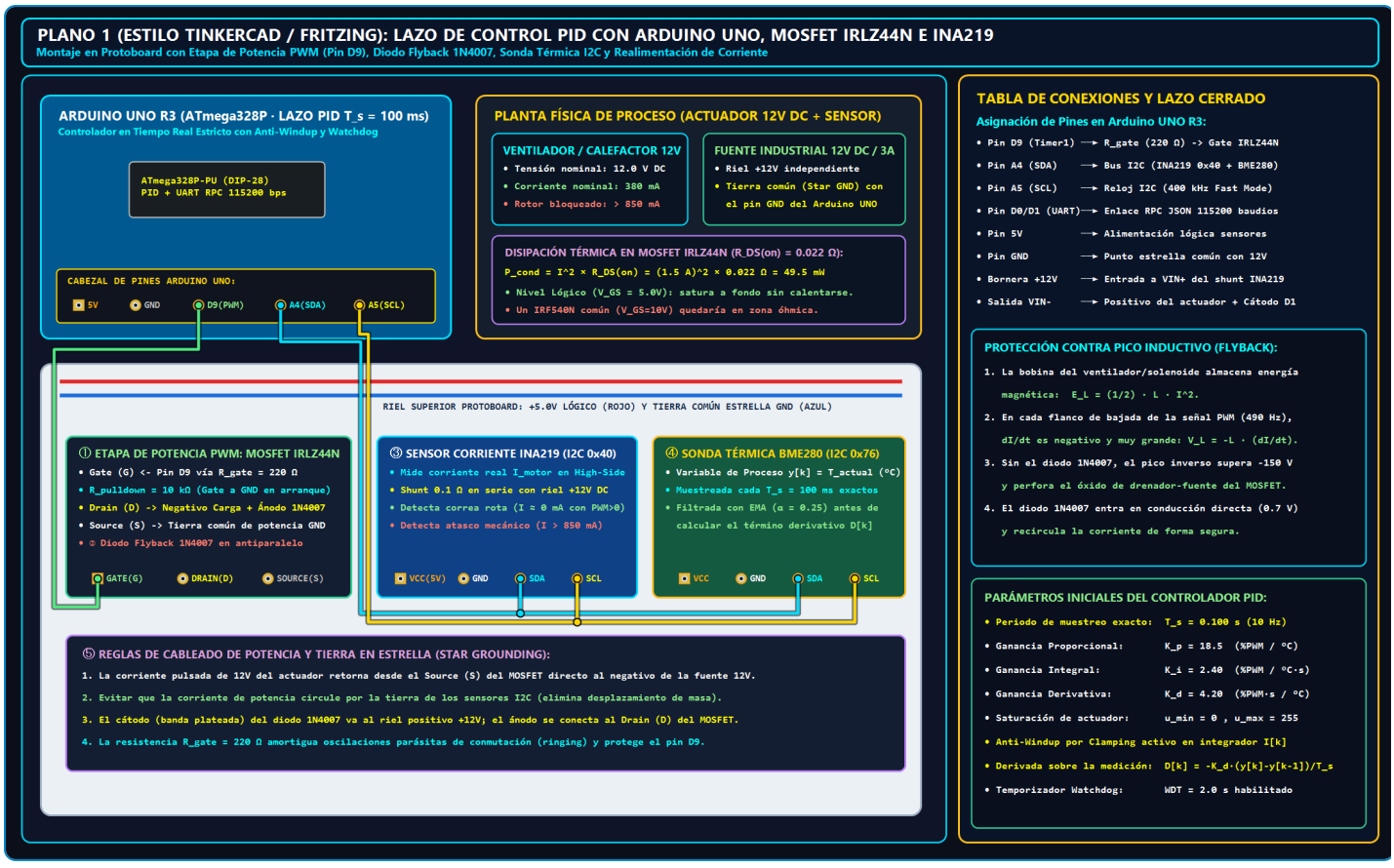
4. Insumos de Construcción para Placa Impresa Soldada (`PCB FR-4`) y Prototipo (`Tinkercad`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
13	1	`PCB1`	**Placa Virgen o Fabricada `FR-4` de Doble Capa (1.6 mm, cobre 1 oz/ft^2, pistas de potencia de 2.50 mm)**	Montaje definitivo por soldadura libre de falsos contactos y con partición de tierra `AGND` / `PGND`.
14	1	`SOCK1`	**Zócalo Torneado `DIP-28` de paso 2.54 mm + Cristal `HC-49S` 16.000 MHz + 2× 22 pF NP0**	Alojamiento seguro del microcontrolador `ATmega328P-PU` sin exponerlo al calor directo del cautín.
15	2	`TB1`, `TB2`	**Borneras de Tornillo para Circuito Impreso `KF301-2P` (paso 5.08 mm, 300 V / 15 A)**	Fijación mecánica firme de los cables de la fuente de 12 V DC y del actuador.
16	1	`F1`	**Fusible Polimérico Rearmable `PTC` de 3.0 A (`PPTC` radial)**	Protección contra cortocircuitos francos en la línea de +12 V.

5. Planos de Referencia para Inspección de Materiales y Montaje

5.1 Plano de Prototipado en Protoboard (`Plano 1`)

📐 Plano 1: Montaje en Protoboard del Lazo Cerrado PID

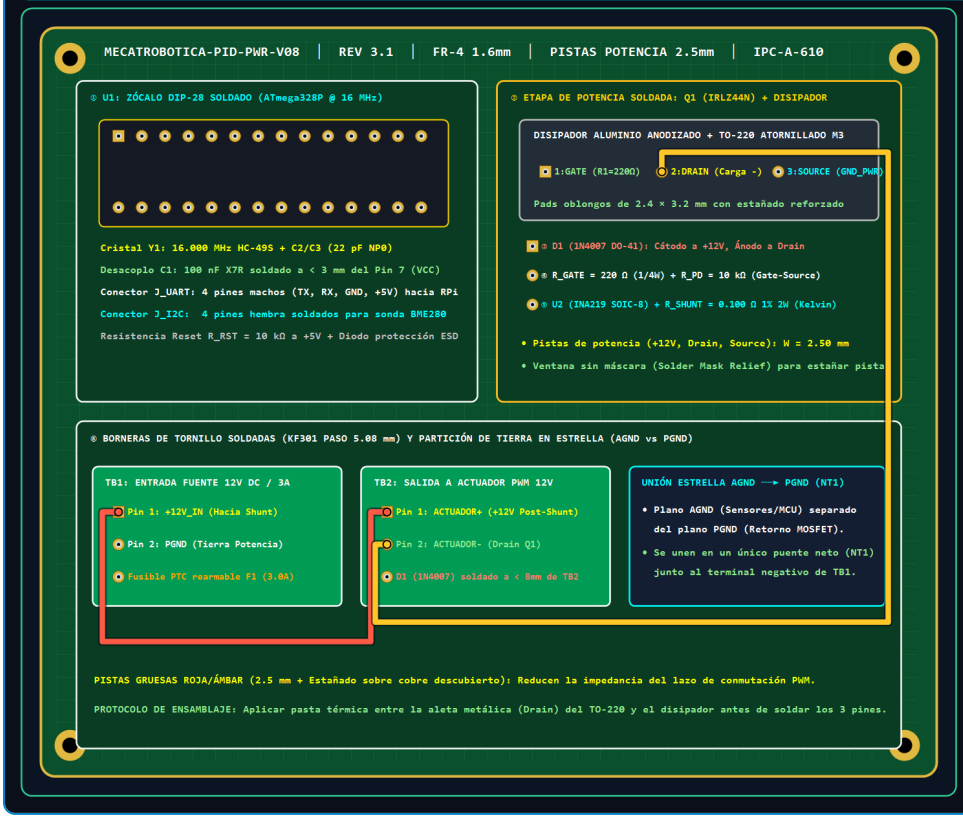


5.2 Plano de Circuito Impreso `FR-4` con Montaje por Soldadura (`Plano 2`)

📐 Plano 2: Placa PCB Soldada del Controlador PID de Potencia

PLANO 2 (PLACA PCB FR-4 CON MONTAJE POR SOLDADURA): CONTROLADOR PID DE POTENCIA E INSTRUMENTACIÓN

Circuito Impreso de Doble Capa con Pistas de Cobre de 2.5 mm para 12V/3A, Zócalo DIP-28 Soldado, MOSFET TO-220 con Disipador y Shunt Kelvin



INGENIERÍA DE PCB DE POTENCIA (IPC-2221)

Dimensionamiento de Pistas y Soldadura THT:

- Corriente máxima de diseño: $I_{max} = 3.0$ A continuos
- Espesor de cobre estándar: 1 oz/ft² (35 µm)
- Ancho de pista requerido (IPC-2221 para $\Delta T = 10$ °C):
  $W_{min} = 1.58$ mm ⟶ Diseñado con $W = 2.50$ mm
- Técnica de ventana sin máscara (Solder Mask Relief): Permite aportar soldadura de estaño-plata sobre la pista de Drain y Source, duplicando su sección útil.

POR QUÉ NO USAR PROTOBOARD CON CARGAS PWM > 1 A:

1. Los flejes elásticos de una protoboard tienen una resistencia de contacto de 80 a 250 mΩ por inserción.
2. A 2.5 A, los contactos se calientan, ablandan el plástico ABS y falsean la tensión de compuerta V_{GS}.
3. El rebote de tierra (Ground Bounce) por los pulsos PWM corrompe las lecturas del sensor térmico I2C.
4. En la placa PCB soldada con tierras separadas (AGND y PGND), el ruido conducido cae más de 40 dB.

LISTA DE INSPECCIÓN POST-SOLDADURA:

1. Verificar con óhmetro resistencia entre Gate y Source: Debe medir exactamente 10.0 kΩ ($R_{pulldown}$).
2. Verificar polaridad de D1 (1N4007): Banda del cátodo hacia +12V (si se invierte, provoca cortocircuito).
3. Comprobar ausencia de puentes de estaño entre los 3 pines del TO-220 (paso 2.54 mm).
4. Limpiar residuos de resina flux con IPA 99.8% antes de insertar el ATmega328P en su zócalo DIP-28.

MANUAL DE INGENIERÍA: CONTROL `PID` DISCRETO, PROTOCOLO `RPC` Y `EDGE AI` — MÓDULO 08

Fundamentos Matemáticos de Control en Tiempo Discreto, Conmutación `MOSFET PWM`, Correlación Asíncrona `MQTT` y Detección Multivariada con `Isolation Forest`

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado, Llamadas a Procedimiento Remoto (`RPC`) e Inteligencia Artificial en el Borde

---

1. Arquitectura de Control Jerárquico: ¿Por qué nunca cerrar un lazo `PID` rápido a través de `Wi-Fi` o `MQTT`?

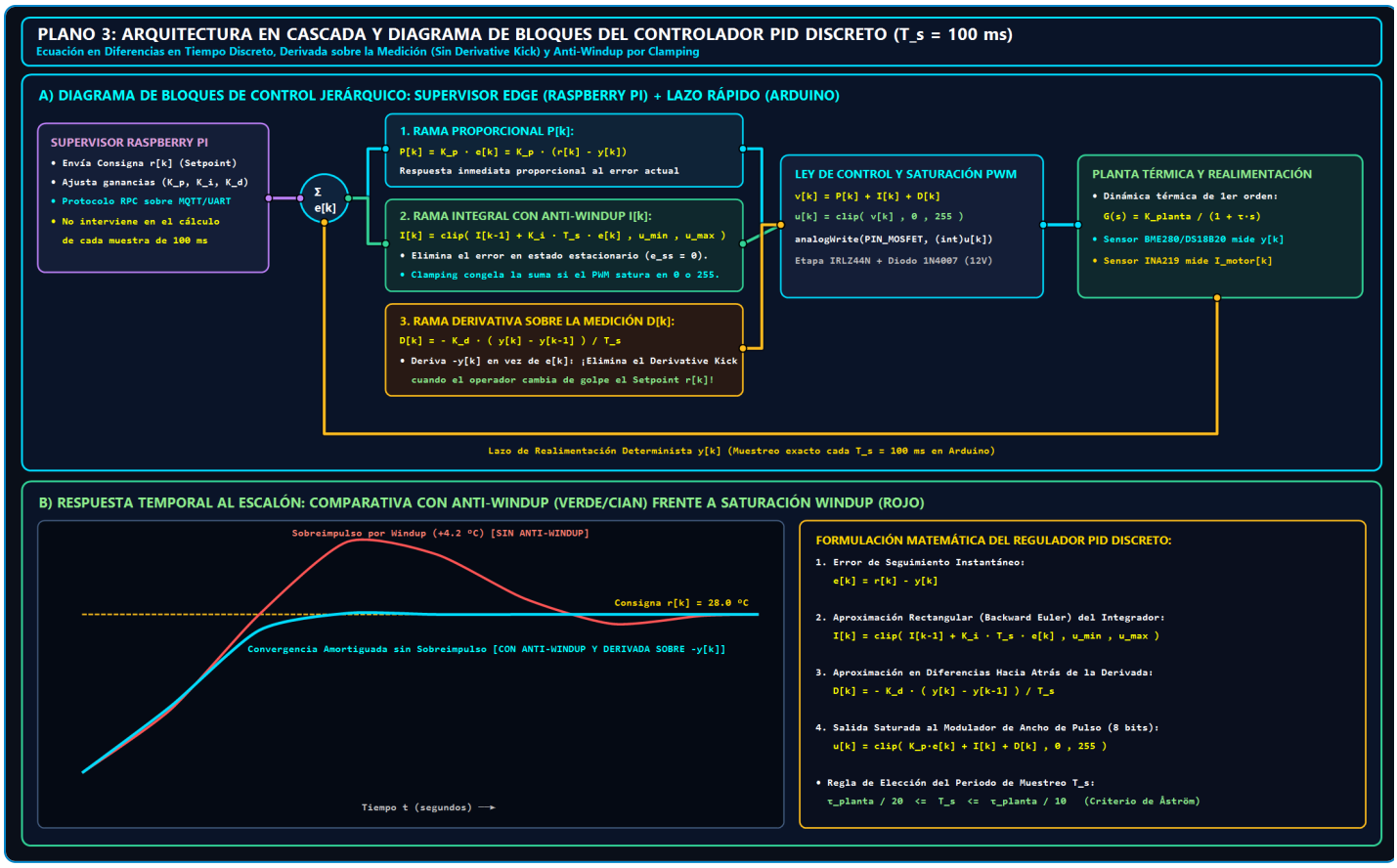
En la ingeniería de sistemas ciberfísicos e industriales, la primera regla de diseño consiste en **separar la capa de control regulatorio en tiempo real estricto (`Hard Real-Time`) de la capa de supervisión y analítica en tiempo real blando (`Soft Real-Time`)**:

- *Incertidumbre temporal (Jitter) de la red:* un enlace `Wi-Fi` o una cola de mensajes `MQTT` sobre `TCP/IP` presenta latencias variables que oscilan entre $3\ ms$ en condiciones ideales y más de $800\ ms$ cuando ocurre una retransmisión `TCP` o una recolección de basura en el sistema operativo.
- **Degradación del margen de fase:** en un controlador proporcional-integral-derivativo (`PID`), un retardo de transporte variable $\tau_d(t)$ introduce un desfase adicional $\Delta\phi(\Omega) = -\Omega \cdot \tau_d(t)$ que reduce el margen de fase del sistema hasta volverlo inestable u oscilatorio.
- **Solución arquitectónica en cascada:**
- **Nivel 1 (`Arduino UNO / ATmega328P`):** ejecuta el algoritmo `PID` discreto con un periodo de muestreo determinista $T_s = 100\ ms$ ($10\ Hz$), lee directamente los sensores `I2C` (`BME280` e `INA219`), comanda el modulador `PWM` del transistor `MOSFET IRLZ44N` y aplica enclavamiento autónomo de seguridad si la corriente supera $850\ mA$.
- **Nivel 2 (`Raspberry Pi 4 / 5`):** envía cambios de consigna $r[k]$ (`Setpoint`) o ajustes de ganancias (K_p, K_i, K_d) mediante transacciones `RPC` correlacionadas, ejecuta el motor de aprendizaje automático `Isolation Forest` y captura evidencia visual por cámara `CSI` .

---

2. Formulación Matemática del Controlador `PID` en Tiempo Discreto (`Plano 3`)

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup



En el dominio continuo, la ley de control `PID` ideal relaciona la señal de mando $u(t)$ con el error de seguimiento $e(t) = r(t) - y(t)$:

$$u(t) = K_p \, e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \, \frac{de(t)}{dt}$$

Al discretizar para un microcontrolador con periodo de muestreo constante $T_s = t_k - t_{k-1} = 0.100\ s$:

2.1 Término Proporcional $P[k]$

$$P[k] = K_p \cdot e[k] = K_p \cdot \left(r[k] - y[k] \right)$$

2.2 Término Integral $I[k]$ con Anti-Windup por Recorte Condicional (*Clamping*)

Aplicando la aproximación rectangular hacia atrás (*Backward Euler*) y acotando el acumulador dentro de los límites físicos del modulador `PWM` de 8 bits ($[u_{min}, u_{max}] = [0, 255]$):

$$I[k] = \text{clip}\left(I[k-1] + K_i \cdot T_s \cdot e[k] \;;\; u_{min} \;;\; u_{max} \right)$$

Además, si la suma total pre-saturación $v[k] = P[k] + I[k] + D[k]$ excede 255 y el error $e[k] > 0$ (o es menor que 0 con $e[k] < 0$), el integrador se congela para evitar el fenómeno de **arrollamiento integral (*Integral Windup)**, eliminando el sobreimpulso térmico.

2.3 Término Derivativo sobre la Medición $D[k]$ (Eliminación del *Derivative Kick*)

Si derivamos el error $e[k] = r[k] - y[k]$, cualquier cambio escalón en la consigna $r[k]$ enviado por el operador desde Grafana produce una pendiente instantánea $\Delta r / T_s$ enorme que satura de golpe el actuador (*Derivative Kick*). Como $r[k]$ es constante por tramos, $\frac{dr}{dt} = 0$ salvo en el instante del cambio de consigna, por lo que reemplazamos $\frac{de}{dt}$ por $-\frac{dy}{dt}$:

$$D[k] = -K_d \cdot (y[k] - y[k-1]) / (T_s)$$

Previamente filtramos la temperatura cruda con un **filtro de media móvil exponencial (`EMA`)** con factor $\alpha = 0.25$:

$$y_{filt}[k] = \alpha \cdot y_{raw}[k] + (1 - \alpha) \cdot y_{filt}[k-1]$$

---

3. Física de Conmutación `MOSFET PWM` y Supresión de Pico Inductivo

- **Pérdidas por conducción en el `MOSFET` de nivel lógico `IRLZ44N`:**

  $$P_{cond} = I_{carga}^2 \cdot R_{DS(on)} = (1.5\ A)^2 \times 0.022\ \Omega = 0.0495\ W \approx 49.5\ mW$$

  Con una resistencia térmica unión-ambiente $R_{\theta JA} = 62\ ^\circ C/W$, el incremento de temperatura sin siquiera disipador para $1.5\ A$ es de apenas $\Delta T = 0.0495 \times 62 \approx 3.1\ ^\circ C$.

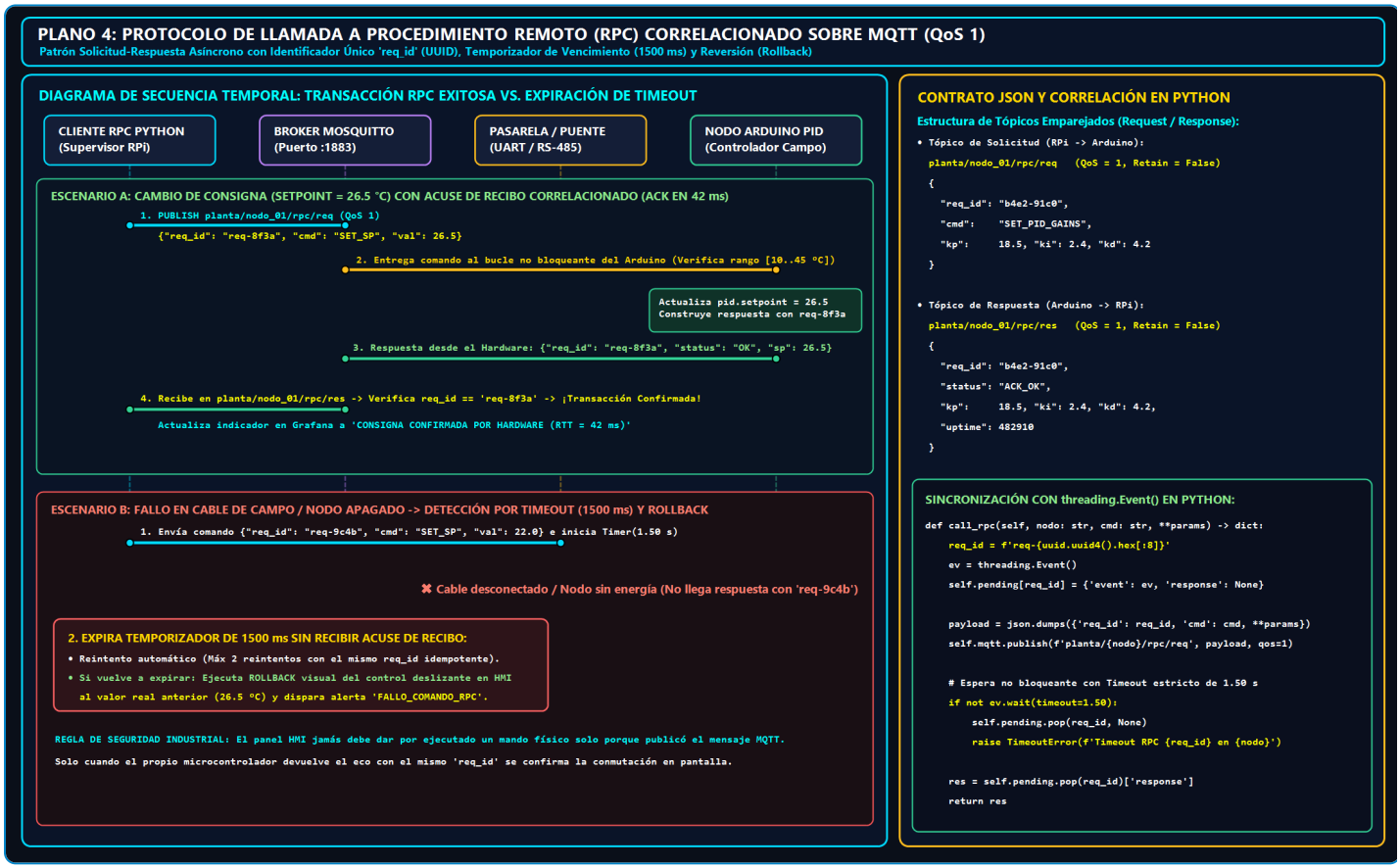
- **Física del diodo volante (`Flyback 1N4007`):**

  El devanado de un ventilador o electroválvula almacena energía magnética $E_L = (1)/(2)\ L\ I^2$. Cuando el transistor `IRLZ44N` pasa de conducción a corte en pocos nanosegundos, la derivada $(dI)/(dt)$ es negativa y de gran magnitud, induciendo una fuerza contraelectromotriz $V_L = -L\ (dI)/(dt)$ que puede superar $+150\ V$ en el drenador y destruir el transistor por avalancha. El diodo `1N4007` en antiparalelo entra en conducción directa a $0.7\ V$, ofreciendo un camino seguro de recirculación para la corriente inductiva.

---

4. Protocolo `RPC` Correlacionado sobre `MQTT` (`Plano 4`)

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



Dado que `MQTT` es un protocolo intrínsecamente asíncrono de publicación/suscripción, publicar un comando en un tópico no garantiza que el microcontrolador de campo lo haya recibido ni ejecutado. Para construir una **Llamada a**

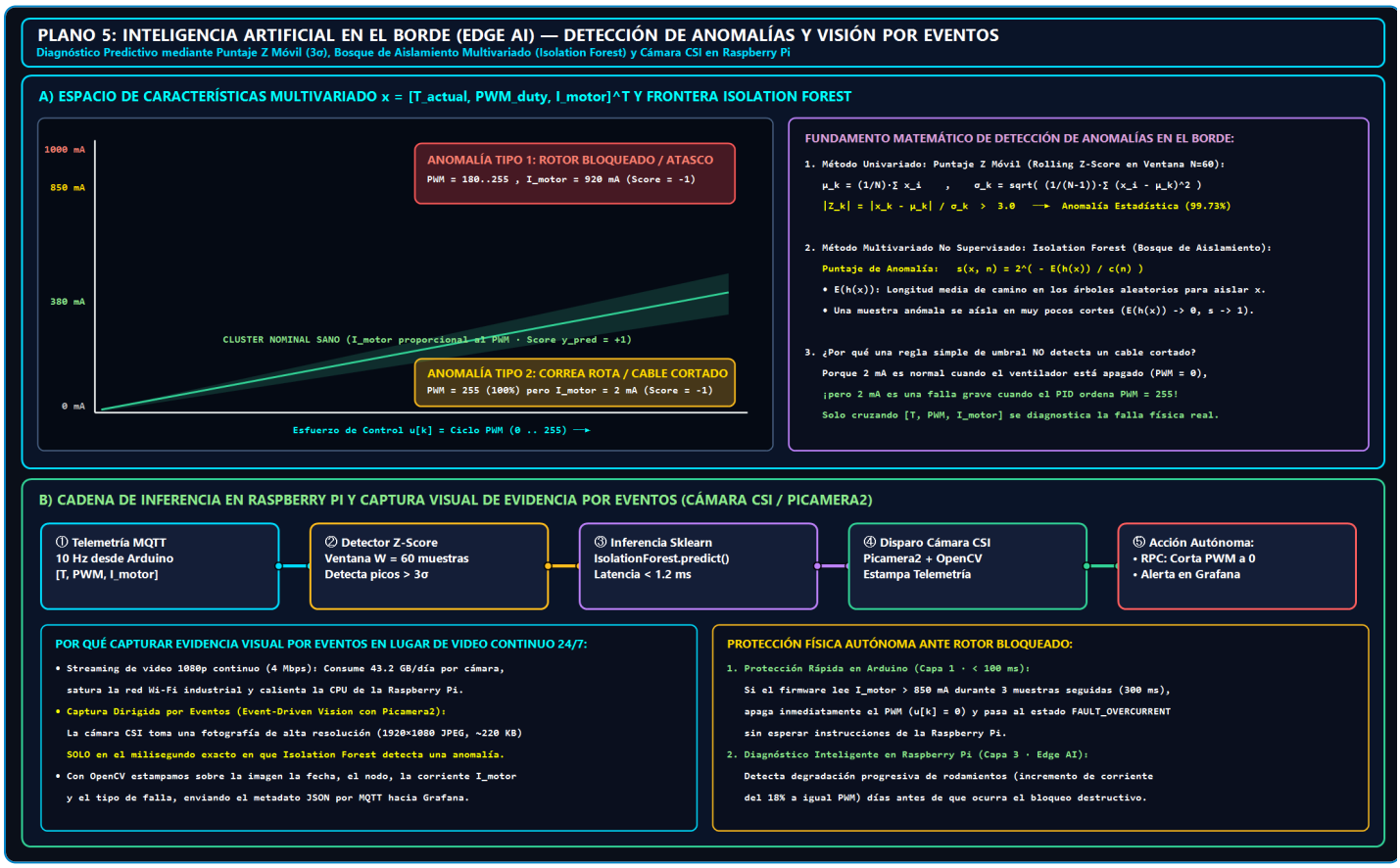
Procedimiento Remoto (`RPC`) de grado industrial:

- **Tópicos desacoplados de solicitud y respuesta:** `planta/nodo_01/rpc/req` y `planta/nodo_01/rpc/res` con `QoS = 1` y `Retain = False`.
- **Correlación estricta por `req_id` (`UUID`):** cada solicitud inyecta un identificador único (ej. `"req_id": "req-8f3a"`). El hilo invocador registra un objeto `threading.Event()` en un diccionario protegido por cerrojo (`Lock`) y se suspende con `event.wait(timeout=1.50)`.
- **Idempotencia y Reversión (`Rollback`):** si el temporizador de $1500\ ms$ expira tras los reintentos configurados, el cliente supervisor revierte el control deslizante en la interfaz HMI al último valor confirmado por el hardware y emite una alerta de fallo de comunicación.

---

5. Inteligencia Artificial en el Borde (`Edge AI`) y Visión por Eventos (`Plano 5`)

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos



5.1 Puntaje Z Móvil (*Rolling Z-Score*) sobre Ventana Deslizante ($N = 60$)

Para detectar picos abruptos en una serie temporal univariada, calculamos en tiempo real la media μ_k y la desviación estándar muestral σ_k:

$$\mu_k = \frac{1}{N}\sum_{i=k-N+1}^{k} x_i \quad , \quad \sigma_k = \sqrt{\frac{1}{N-1}\sum_{i=k-N+1}^{k}\left(x_i - \mu_k\right)^2} \quad , \quad Z_k = \frac{|x_k - \mu_k|}{\sigma_k}$$

Toda lectura con $Z_k > 3.0$ se desvía más de tres desviaciones estándar de su comportamiento reciente ($p < 0.0027$).

5.2 Detección Multivariada No Supervisada con `Isolation Forest`

Una regla simple de umbral sobre la corriente I_{motor} **no puede detectar una correa rota ni un cable cortado**, porque $I_{motor} = 2\ mA$ es completamente normal cuando el ventilador está apagado ($PWM = 0$). Sin embargo, $I_{motor} = 2\ mA$

cuando el regulador `PID` ordena $\mathrm{PWM} = 255$ constituye una falla crítica de lazo abierto.

Al alimentar el algoritmo **Bosque de Aislamiento (`Isolation Forest`)** con el vector de estado tridimensional:

$$\mathbf{x}_k = \begin{bmatrix} T_{actual}[k] \\ PWM_{duty}[k] \\ I_{motor}[k] \end{bmatrix}$$

el conjunto de 100 árboles aleatorios particiona el espacio de características y calcula el puntaje de anomalía:

$$s(\mathbf{x}, n) = 2^{-\frac{E(h(\mathbf{x}))}{c(n)}}$$

donde $E(h(\mathbf{x}))$ es la longitud media de camino necesaria para aislar la muestra $\mathbf{x}$. Las muestras anómalas (tanto el **rotor bloqueado** con $I_{motor} > 850\ mA$ como la **correa rota** con $PWM{=}255$ e $I_{motor} \approx 0\ mA$) quedan aisladas cerca de la raíz de los árboles ($E(h(\mathbf{x})) \to 0 \implies s \to 1$), disparando en menos de $1.2\ ms$ la captura de una fotografía forense `1080p` con la cámara `CSI` y el apagado preventivo del actuador.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 08

Planos de Alta Legibilidad en Protoboard (`Tinkercad`), Circuito Impreso Soldado (`PCB FR-4`), Lazo `PID` , Secuencia `RPC` y `Edge AI`

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado (`PID`), Protocolo `RPC` , `Edge AI` y Visión por Eventos

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje en Protoboard del Lazo Cerrado `PID`

Este primer plano presenta el montaje experimental en protoboard con tipografía ampliada de alta legibilidad y llamadas numeradas (① a ⑤), interconectando el **Arduino UNO R3**, la etapa de potencia `MOSFET IRLZ44N` con diodo volante `1N4007` , el sensor de corriente de lado alto `INA219 (0x40)` y la sonda térmica `BME280 (0x76)` .

📐 Plano 1: Montaje en Protoboard del Lazo de Control PID con MOSFET IRLZ44N e INA219

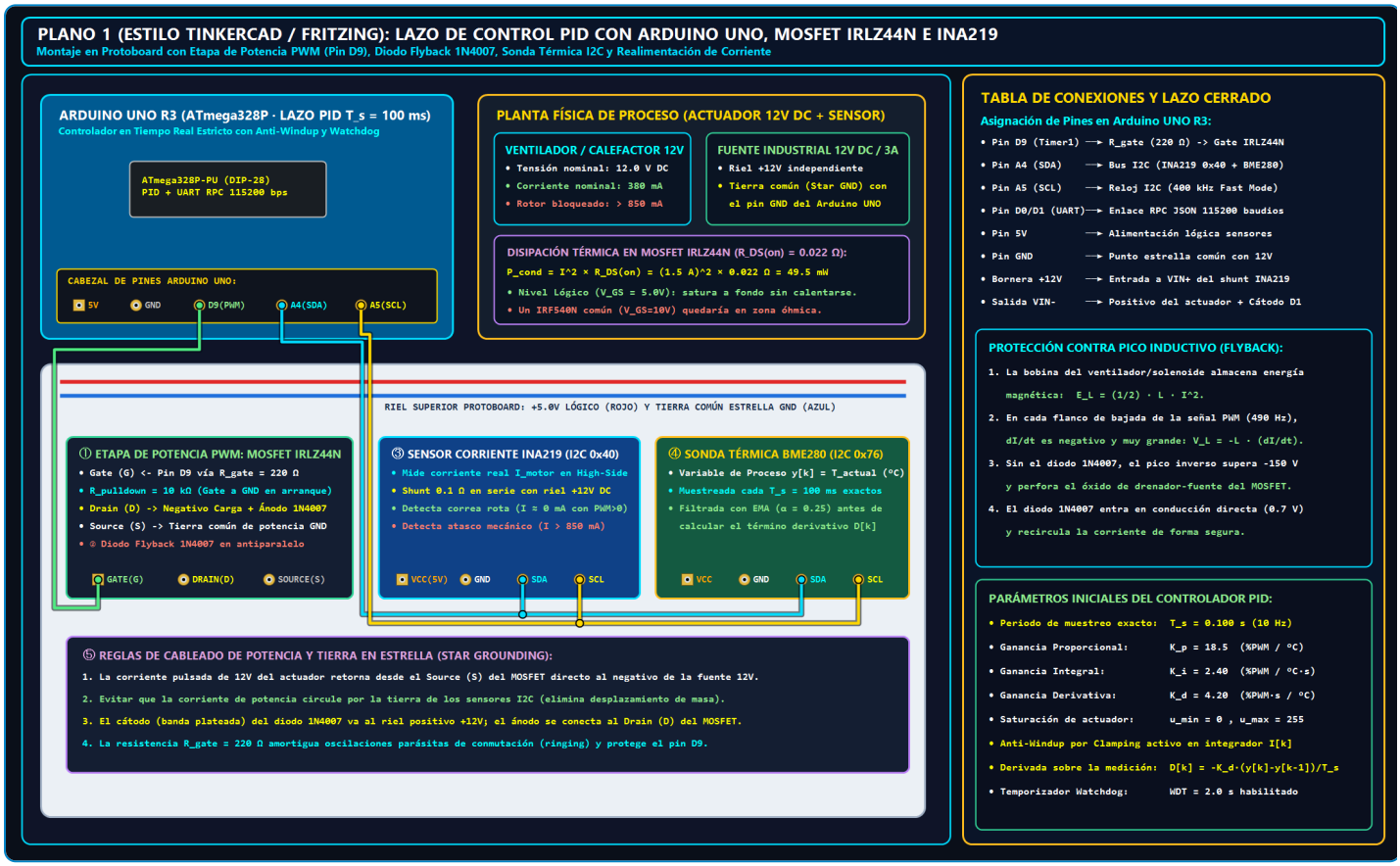


Tabla de Asignación de Terminales y Funciones Físicas

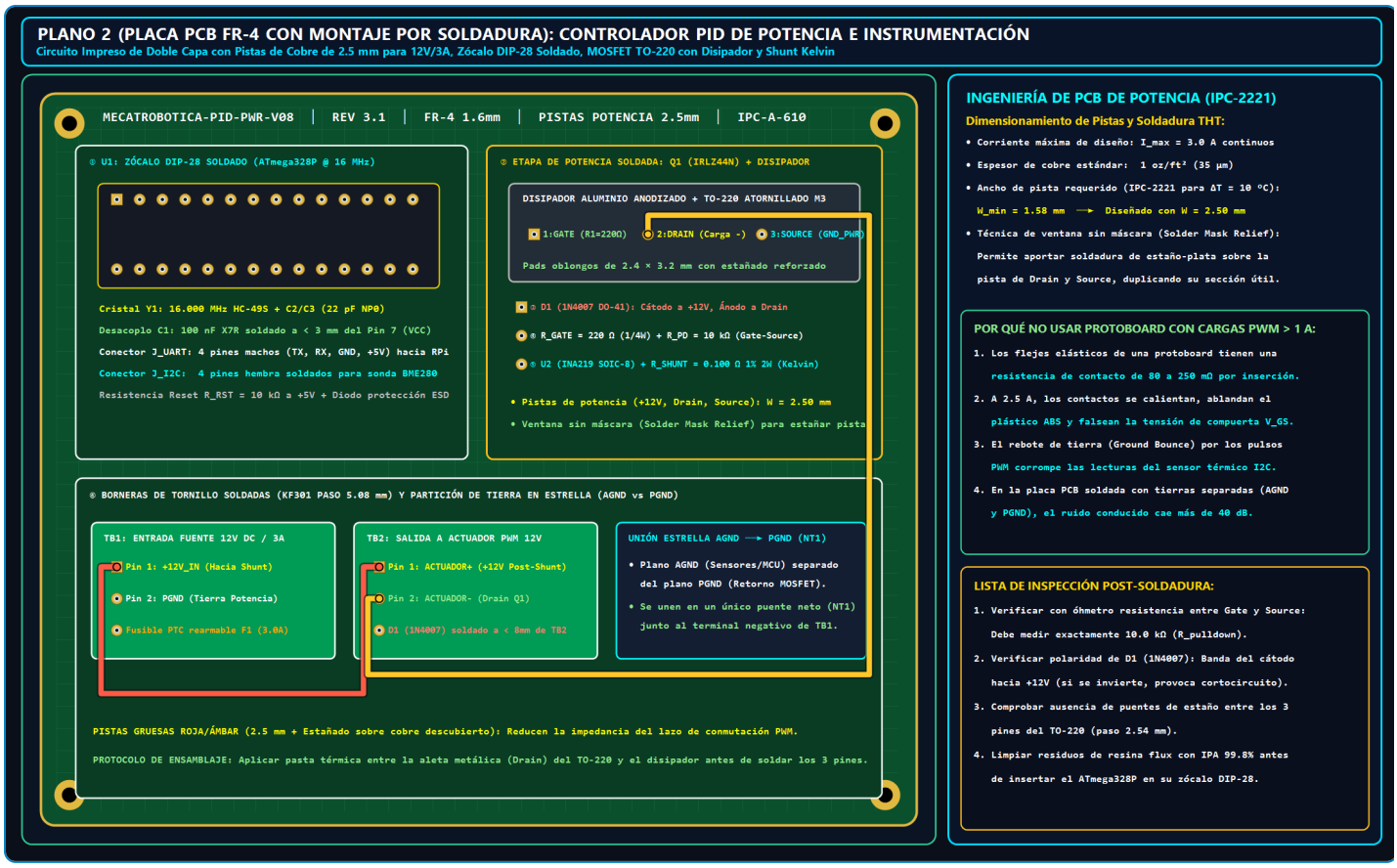
Origen (`Arduino / Fuente`)	Destino en Etapa de Campo	Color de Cable	Función Eléctrica y Regla de Diseño
`Pin D9` (`Timer1 PWM`)	Compuerta (`Gate`) del `IRLZ44N` vía $R_{gate} = 220\ \Omega$	Verde	Señal `PWM` de $490\ Hz$ (0 a 255). Incluye $R_{pulldown} = 10\ k\Omega$ entre `Gate` y `GND`.
`Pin A4` (`SDA`)	Pines `SDA` de `INA219 (0x40)` y `BME280 (0x76)`	Cian	Línea de datos serie `I2C` operando a $400\ kHz$ (*Fast Mode*).
`Pin A5` (`SCL`)	Pines `SCL` de `INA219 (0x40)` y `BME280 (0x76)`	Amarillo	Línea de reloj serie `I2C` sincronizada por el `ATmega328P`.
`Fuente +12V DC`	Borne `VIN+` del `Shunt` de $0.1\ \Omega$ (`INA219`)	Rojo	Entrada de potencia de lado alto (*High-Side*) protegida por fusible `PTC` de $3\ A$.
Borne `VIN-` (`INA219`)	Positivo del Actuador $12\ V$ + Cátodo de `1N4007`	Ámbar	Alimentación medida hacia el actuador y punto de sujeción del diodo de rueda libre.
Negativo del Actuador $12\ V$	Drenador (`Drain`) del `IRLZ44N` + Ánodo de `1N4007`	Amarillo	Conmutación de lado bajo (*Low-Side*) y absorción del pico inductivo inverso.
Surtidor (`Source`) `IRLZ44N`	Punto Estrella `GND` de la Fuente de $12\ V\ DC$	Azul / Negro	Retorno directo de corriente pulsada sin contaminar la tierra analógica de los sensores.

---

2. Plano 2 (Placa `PCB FR-4` con Montaje por Soldadura): Controlador `PID` de Potencia

Para aplicaciones permanentes donde las corrientes pulsadas `PWM` de 1 a $3\ A$ degradarían los contactos elásticos de un protoboard, este segundo plano detalla el diseño de una **placa de circuito impreso (`PCB FR-4`) de doble capa con montaje por soldadura (`THT` / `SMD`)**, pistas de potencia de $2.50\ mm$ con ventana de estañado (*Solder Mask Relief*) y partición de tierras en estrella (`AGND` vs. `PGND`).

📐 Plano 2: Placa PCB FR-4 de Doble Capa con Montaje por Soldadura para Controlador PID

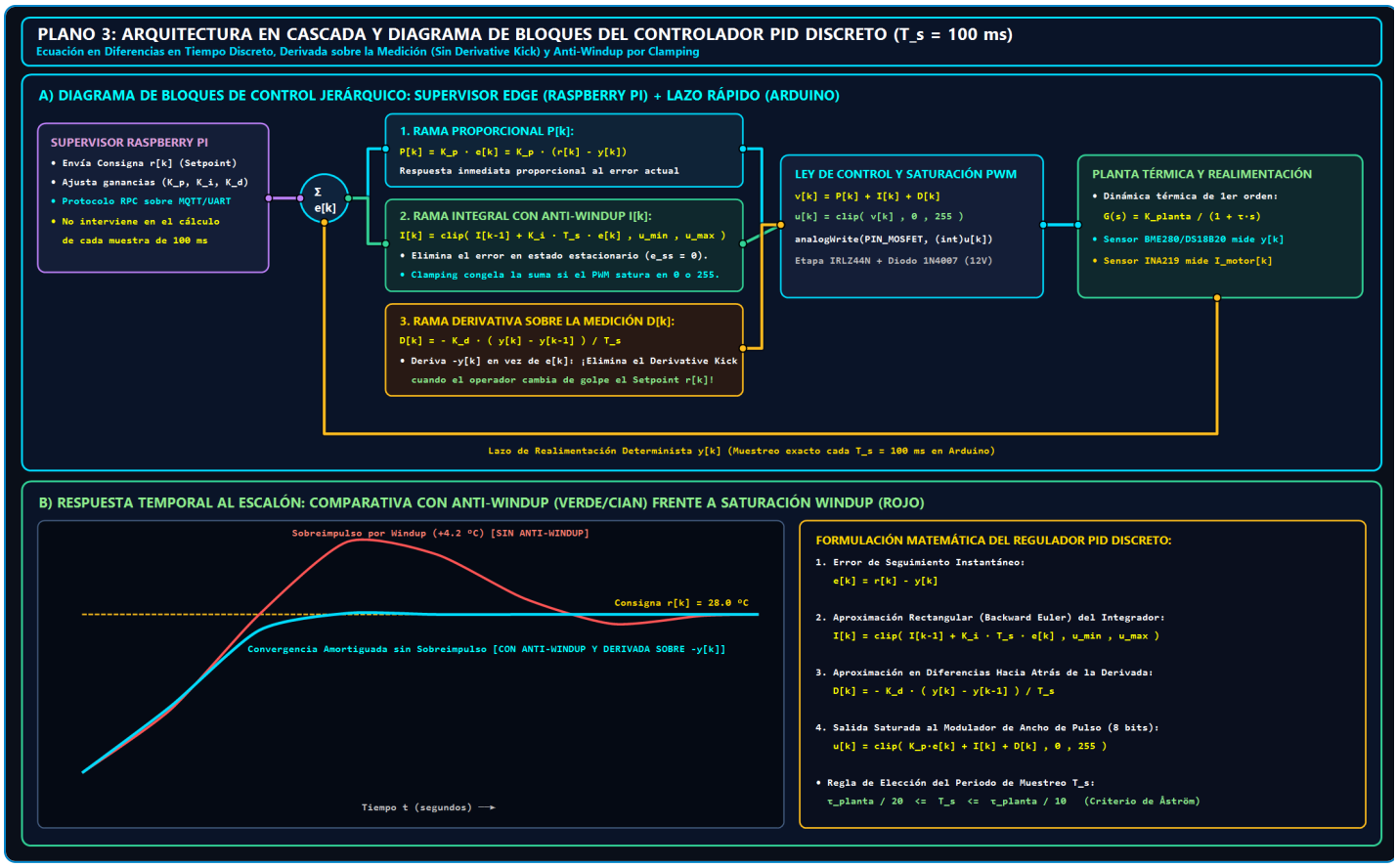


---

3. Plano 3: Arquitectura en Cascada y Diagrama de Bloques del Regulador `PID` Discreto

Detalla la estructura matemática de las tres ramas paralelas ($P[k]$, $I[k]$ con *Clamping Anti-Windup*, y $D[k]$ calculada sobre $-y[k]$) junto con las curvas comparativas de respuesta temporal al escalón.

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup

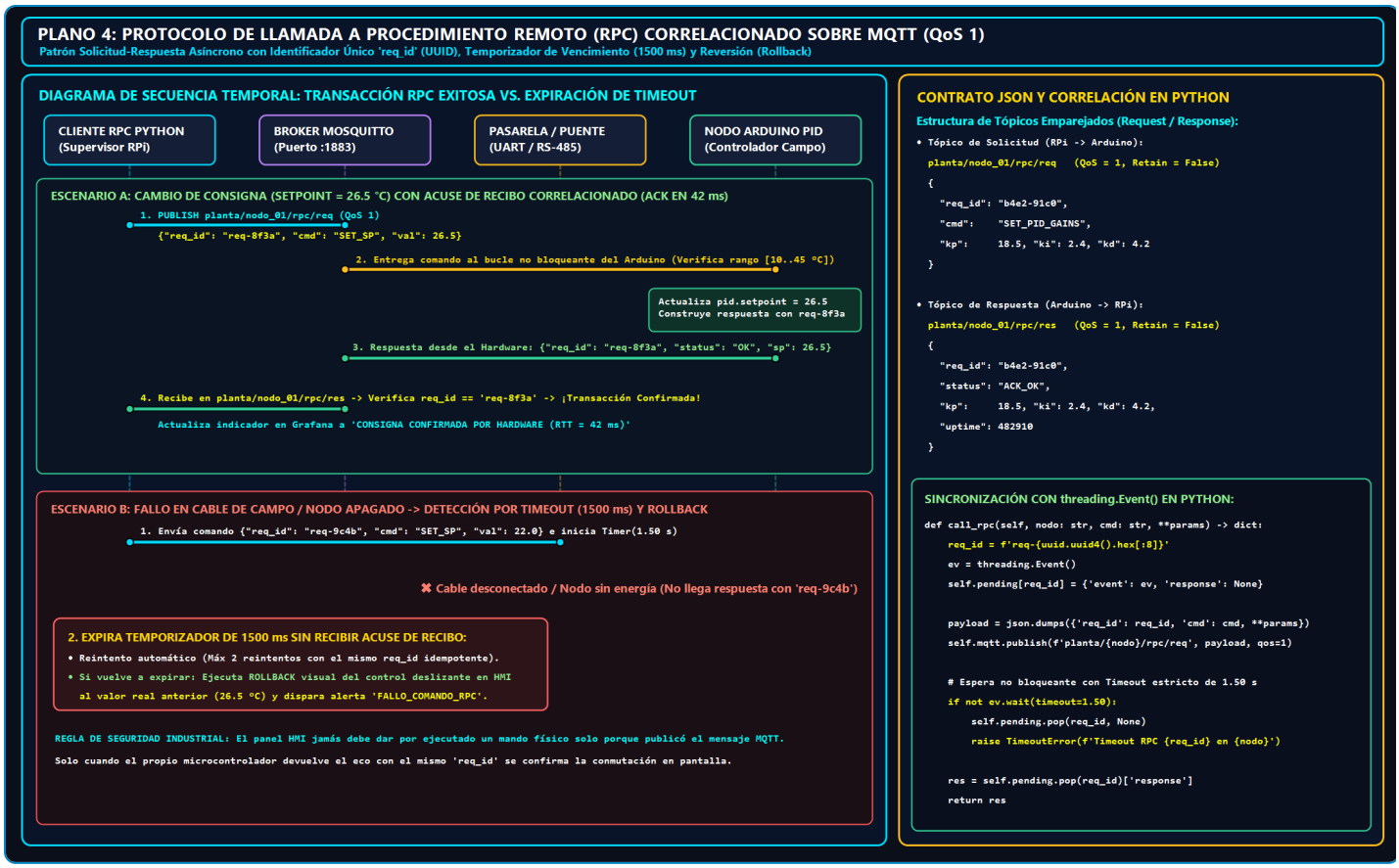


---

4. Plano 4: Diagrama de Secuencia del Protocolo `RPC` Correlacionado sobre `MQTT`

Representa el flujo temporal completo de una transacción exitosa con acuse de recibo (`ACK`) correlacionado en $42\ ms$ frente a un escenario de corte de cable detectado por expiración de temporizador ($1500\ ms$) con reversión automática (`Rollback`).

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



---

5. Plano 5: Espacio de Características `Isolation Forest` y Visión por Eventos (`CSI`)

Ilustra la separación geométrica en el espacio tridimensional $[T_{actual}, PWM, I_{motor}]$ entre el corredor nominal sano, la región de rotor bloqueado y la región de correa rota/cable abierto, así como la cadena de captura fotográfica `1080p` con `Picamera2` y `OpenCV`.

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos

PLANO 5: INTELIGENCIA ARTIFICIAL EN EL BORDE (EDGE AI) — DETECCIÓN DE ANOMALÍAS Y VISIÓN POR EVENTOS

Diagnóstico Predictivo mediante Puntaje Z Móvil (3σ), Bosque de Aislamiento Multivariado (Isolation Forest) y Cámara CSI en Raspberry Pi

A) ESPACIO DE CARACTERÍSTICAS MULTIVARIADO x = [T_actual, PWM_duty, I_motor]^T Y FRONTERA ISOLATION FOREST



FUNDAMENTO MATEMÁTICO DE DETECCIÓN DE ANOMALÍAS EN EL BORDE:

1. Método Univariado: Puntaje Z Móvil (Rolling Z-Score en Ventana N=60):

   $\mu_k = (1/N)\cdot\Sigma\ x_i$, $\sigma_k = \text{sqrt}((1/(N-1))\cdot\Sigma\ (x_i - \mu_k)^2)$

   $|Z_k| = |x_k - \mu_k| / \sigma_k > 3.0 \longrightarrow$ Anomalía Estadística (99.73%)

2. Método Multivariado No Supervisado: Isolation Forest (Bosque de Aislamiento):

   Puntaje de Anomalía: $s(x, n) = 2^{(- E(h(x)) / c(n))}$

   • E(h(x)): Longitud media de camino en los árboles aleatorios para aislar x.

   • Una muestra anómala se aísla en muy pocos cortes (E(h(x)) -> 0, s -> 1).

3. ¿Por qué una regla simple de umbral NO detecta un cable cortado?

   Porque 2 mA es normal cuando el ventilador está apagado (PWM = 0),

   ¡pero 2 mA es una falla grave cuando el PID ordena PWM = 255!

   Solo cruzando [T, PWM, I_motor] se diagnostica la falla física real.

B) CADENA DE INFERENCIA EN RASPBERRY PI Y CAPTURA VISUAL DE EVIDENCIA POR EVENTOS (CÁMARA CSI / PICAMERA2)

① Telemetría MQTT
10 Hz desde Arduino
[T, PWM, I_motor]

② Detector Z-Score
Ventana W = 60 muestras
Detecta picos > 3σ

③ Inferencia Sklearn
IsolationForest.predict()
Latencia < 1.2 ms

④ Disparo Cámara CSI
Picamera2 + OpenCV
Estampa Telemetría

⑤ Acción Autónoma:
• RPC: Corta PWM a 0
• Alerta en Grafana

POR QUÉ CAPTURAR EVIDENCIA VISUAL POR EVENTOS EN LUGAR DE VIDEO CONTINUO 24/7:

• Streaming de video 1080p continuo (4 Mbps): Consume 43.2 GB/día por cámara, satura la red Wi-Fi industrial y calienta la CPU de la Raspberry Pi.

• Captura Dirigida por Eventos (Event-Driven Vision con Picamera2):
  La cámara CSI toma una fotografía de alta resolución (1920×1080 JPEG, ~220 KB) SOLO en el milisegundo exacto en que Isolation Forest detecta una anomalía.

• Con OpenCV estampamos sobre la imagen la fecha, el nodo, la corriente I_motor y el tipo de falla, enviando el metadato JSON por MQTT hacia Grafana.

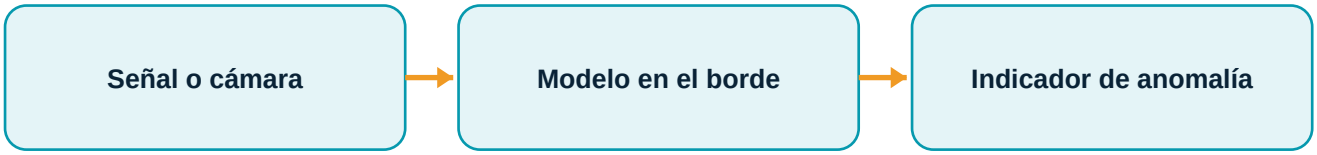
PROTECCIÓN FÍSICA AUTÓNOMA ANTE ROTOR BLOQUEADO:

1. Protección Rápida en Arduino (Capa 1 · < 100 ms):
   Si el firmware lee I_motor > 850 mA durante 3 muestras seguidas (300 ms), apaga inmediatamente el PWM (u[k] = 0) y pasa al estado FAULT_OVERCURRENT sin esperar instrucciones de la Raspberry Pi.
2. Diagnóstico Inteligente en Raspberry Pi (Capa 3 · Edge AI):
   Detecta degradación progresiva de rodamientos (incremento de corriente del 18% a igual PWM) días antes de que ocurra el bloqueo destructivo.

Capítulo 27 · IA en el borde: Isolation Forest y visión CSI

Conceptos clave

Separa las muestras de entrenamiento y validación. Verifica falsas alarmas, tiempos de inferencia y comportamiento ante datos ausentes.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 08

Control en Lazo Cerrado (`PID` Discreto), Llamadas a Procedimiento Remoto (`RPC` sobre `MQTT`) e Inteligencia Artificial en el Borde (`Edge AI`)

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Regulación `PID` en Tiempo Discreto, Protocolo `RPC` Correlacionado, `Isolation Forest` y Visión por Eventos

Duración estimada de laboratorio: 4 horas prácticas

---

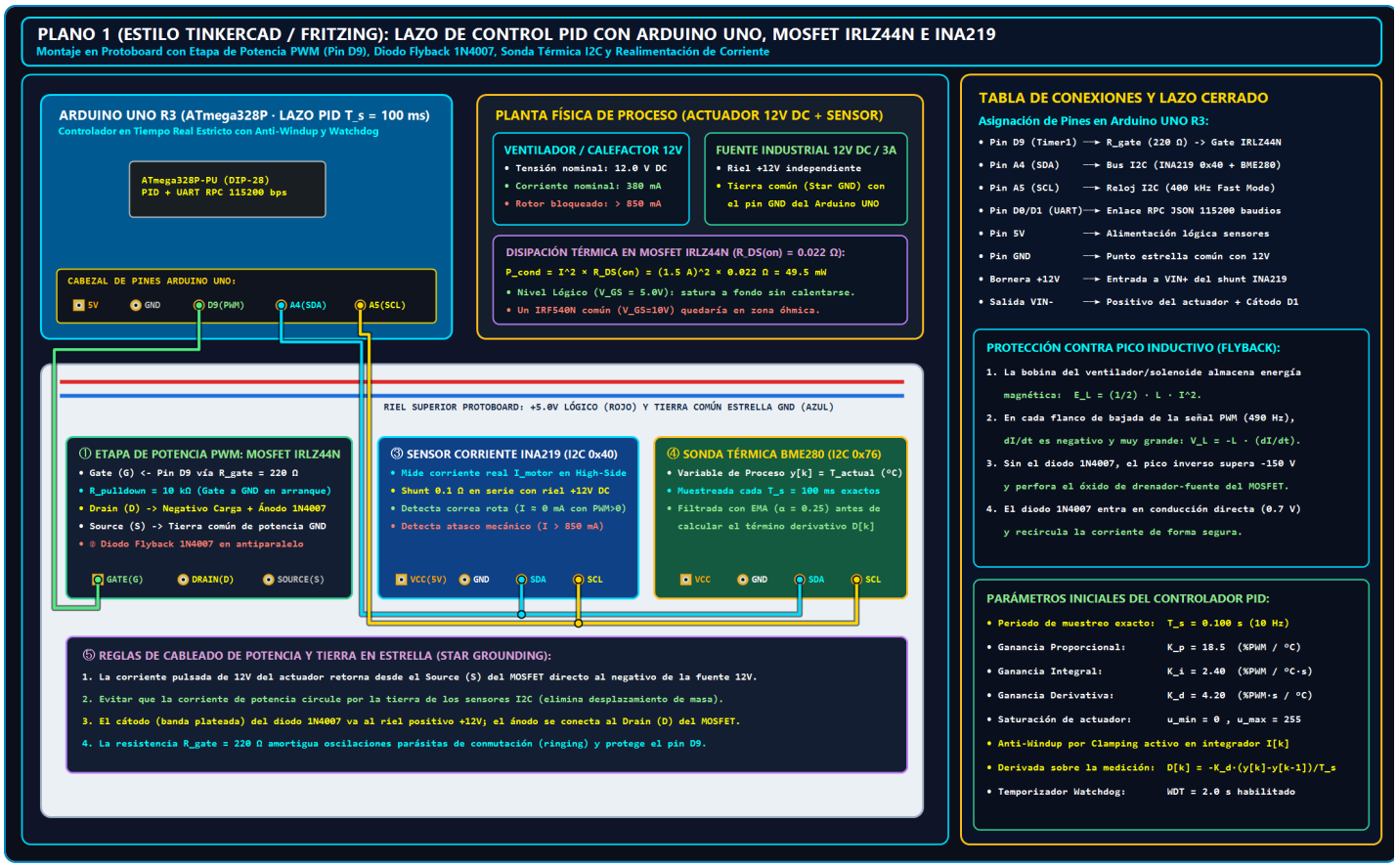
1. Objetivos de Ingeniería del Laboratorio

- **Separar la jerarquía de control en tiempo real estricto (`Hard Real-Time`) de la supervisión analítica en el borde (`Soft Real-Time`):** ejecutar el lazo cerrado proporcional-integral-derivativo (`PID`) cada $T_s = 100\ ms$ en el microcontrolador **Arduino UNO R3 (`ATmega328P`)**, delegando el ajuste de consigna (`Setpoint`), la inferencia de anomalías y la visualización a la **Raspberry Pi 4 / 5**.
- **Construir la etapa de potencia en modulación por ancho de pulso (`PWM`) con transistor `MOSFET` de nivel lógico (`IRLZ44N`):** calcular la disipación térmica por conducción, instalar la resistencia limitadora de compuerta ($220\ \Omega$), la resistencia de anclaje a tierra ($10\ k\Omega$) y el diodo volante de rueda libre (`1N4007`) en antiparalelo con la carga inductiva de $12\ V\ DC$.
- **Implementar ambos planos de construcción electrónica:** el prototipo experimental en **protoboard (estilo Tinkercad / Fritzing)** con tipografía ampliada de alta legibilidad y la **placa de circuito impreso (`PCB FR-4` de doble capa) con montaje por soldadura** y pistas de potencia de $2.50\ mm$.
- **Programar un protocolo de Llamada a Procedimiento Remoto (`RPC`) correlacionado sobre `MQTT` (`QoS 1`):** emparejar solicitudes y respuestas mediante un identificador único universal (`req_id`), sincronizar hilos con `threading.Event()` , establecer un temporizador de expiración de $1500\ ms$ y aplicar reversión segura (`Rollback`) en la interfaz gráfica si el hardware no confirma la orden.
- **Desplegar Inteligencia Artificial en el Borde (`Edge AI`) y captura visual dirigida por eventos:** ejecutar detección estadística por puntaje Z móvil (`Rolling Z-Score` 3σ) e inferencia multivariada con **Bosque de Aislamiento (`Isolation Forest`)** sobre el vector $\mathbf{x} = [T_{actual}, PWM, I_{motor}]^T$, disparando la cámara `CSI` (`Picamera2` + `OpenCV`) en el milisegundo exacto de una anomalía.

---

2. Arquitectura Física del Lazo Cerrado y Etapa de Potencia (`Plano 1`)

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad del Lazo de Control PID con MOSFET IRLZ44N e INA219



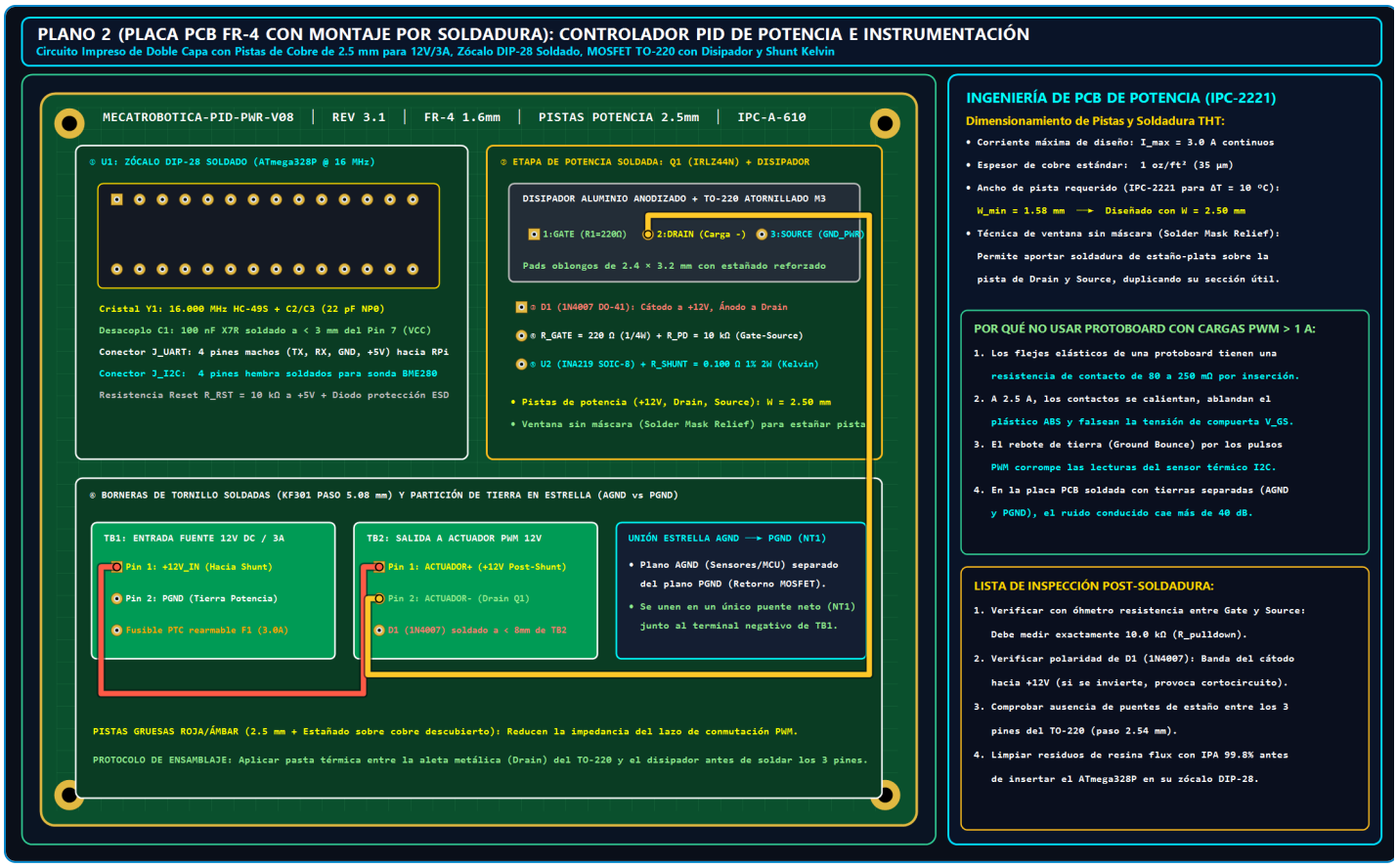
Paso 2.1 — Reglas Críticas de Cableado de Potencia y Tierra en Estrella (`Star Grounding`)

- **Por qué es obligatorio un `MOSFET` de nivel lógico (`IRLZ44N`):** el pin `D9` del Arduino UNO entrega $5.0\ \text{V}$ (y las placas de $3.3\ \text{V}$ entregan aún menos). Un transistor `MOSFET` estándar como el `IRF540N` requiere $V_{GS} = 10\ \text{V}$ para saturar completamente; si se excita con $5\ \text{V}$, permanece en su región lineal/óhmica y se destruye por sobrecalentamiento. El `IRLZ44N` posee tensión umbral $V_{GS(th)} \in [1.0, 2.0]\ \text{V}$ y alcanza $R_{DS(on)} = 0.022\ \Omega$ con $V_{GS} = 5.0\ \text{V}$.
- **Diodo de rueda libre (`Flyback 1N4007`):** conéctelo directamente entre los terminales del actuador inductivo (ventilador o válvula solenoide de $12\ \text{V}$), con el **cátodo (banda plateada)** hacia el riel positivo de $+12\ \text{V}$ y el **ánodo** hacia el drenador (`Drain`) del `IRLZ44N`.
- **Tierra en estrella (`Star Ground`):** el surtidor (`Source`) del `IRLZ44N` debe retornar directamente por un cable grueso al borne negativo de la fuente de $12\ \text{V DC}$, evitando que los pulsos de corriente de conmutación `PWM` circulen por la pista de tierra de los sensores `I2C` (`BME280` e `INA219`).

---

3. Montaje Definitivo en Placa de Circuito Impreso Soldada (`Plano 2`)

📐 Plano 2: Placa PCB FR-4 de Doble Capa con Montaje por Soldadura para Controlador PID de Potencia



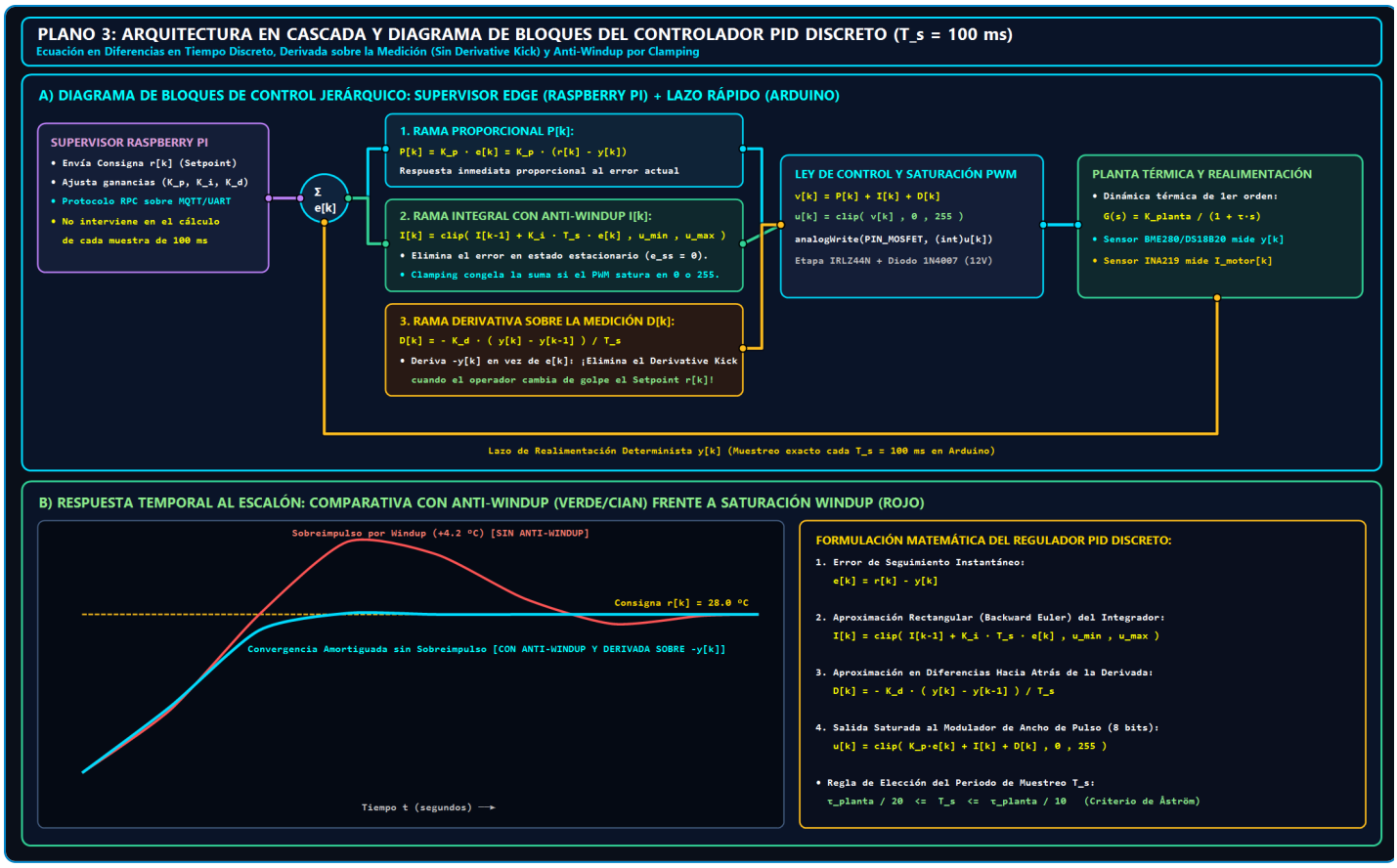
Paso 3.1 — Procedimiento de Soldadura de Potencia según `IPC-2221` e `IPC-A-610`

- Suelde primero los componentes de bajo perfil: resistencias de película metálica ($220\ \Omega$, $10\ k\Omega$), condensadores cerámicos `X7R` de $100\ nF$, el cristal de $16.000\ MHz$ y el sensor de corriente `INA219` junto a su resistencia `Shunt` Kelvin de $0.100\ \Omega$ (1%, $2\ W$).
- Suelde el **zócalo torneado** `DIP-28` para el microcontrolador `ATmega328P-PU` sin el chip insertado.
- Atornille la aleta metálica del `MOSFET IRLZ44N` (`TO-220`) al disipador de aluminio anodizado aplicando pasta térmica y suelde sus tres terminales en los pads oblongos reforzados.
- Suelde las borneras de tornillo `TB1` (entrada $12\ V\ DC$) y `TB2` (salida al actuador) y aporte una capa uniforme de aleación `SAC305` sobre las ventanas descubiertas de máscara (*Solder Mask Relief*) en las pistas de $2.50\ mm$.

---

4. Discretización del Controlador `PID` y Anti-Windup (`Plano 3`)

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup



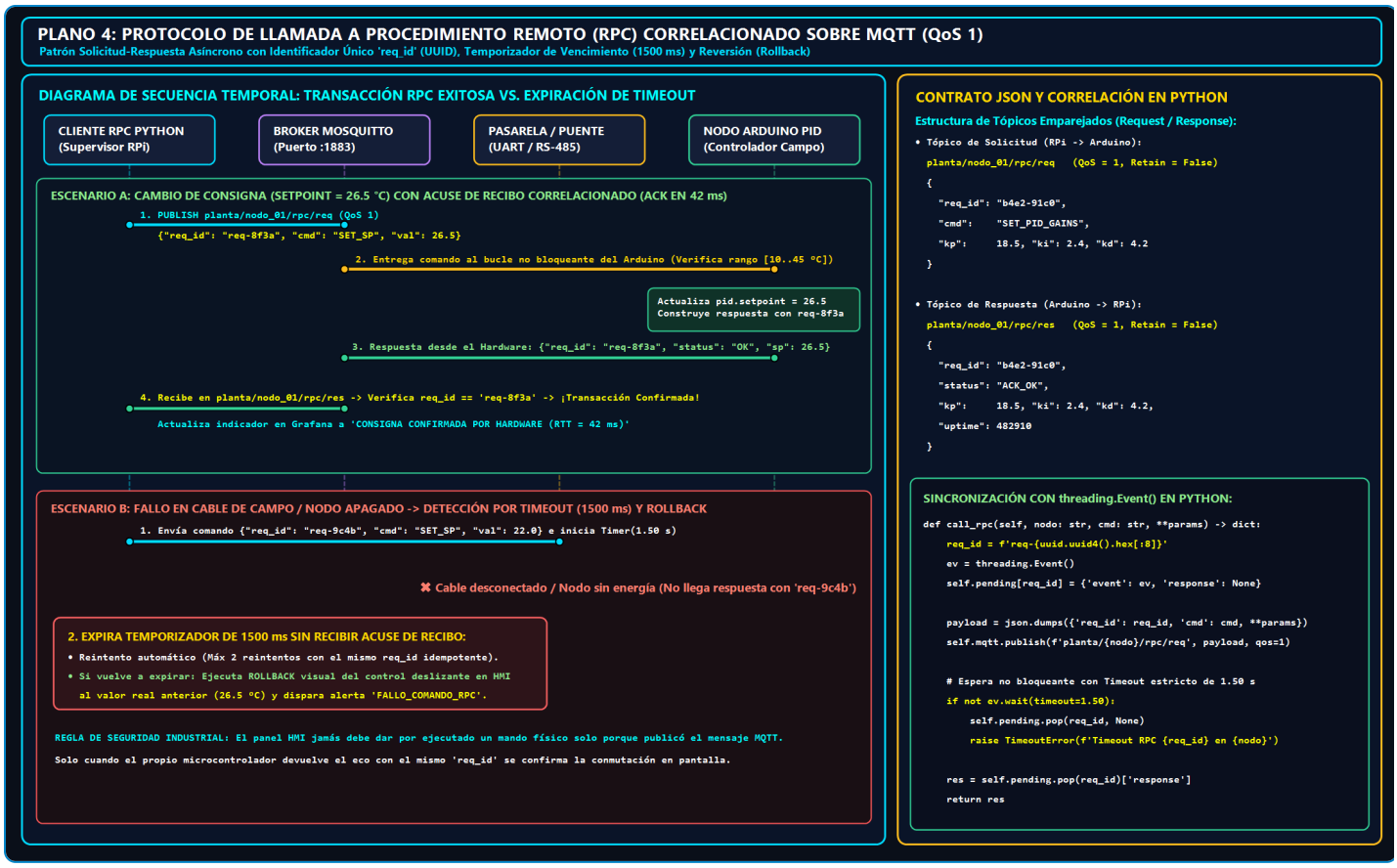
Paso 4.1 — Compilación y Carga del Firmware Determinista en Arduino UNO

- Abra el archivo `codigo_fuente/controlador_pid_discreto_rpc_arduino.ino` en Arduino IDE o PlatformIO.
- Verifique las tres mejoras industriales implementadas en la función `computeDiscretePID()`:
- **Muestreo exacto por temporizador no bloqueante:** $T_s = 100\ ms$ $(10\ Hz)$.
- **Derivada sobre la medición (**$-y[k]$**):** $D[k] = -K_d \cdot (y[k] - y[k-1]) / T_s$, eliminando el golpe derivativo (*Derivative Kick*) cuando el operador modifica de golpe el `Setpoint` desde Grafana.
- **Anti-Windup por recorte condicional (*Clamping*):** congela la acumulación del integrador $I[k]$ cuando la salida `PWM` alcanza 0 o 255.

---

5. Protocolo `RPC` Correlacionado sobre `MQTT` (`Plano 4`)

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



Paso 5.1 — Prueba de Transacción Confirmada y Prueba de Fallo con `Rollback`

- Ejecute el cliente supervisor en la Raspberry Pi:

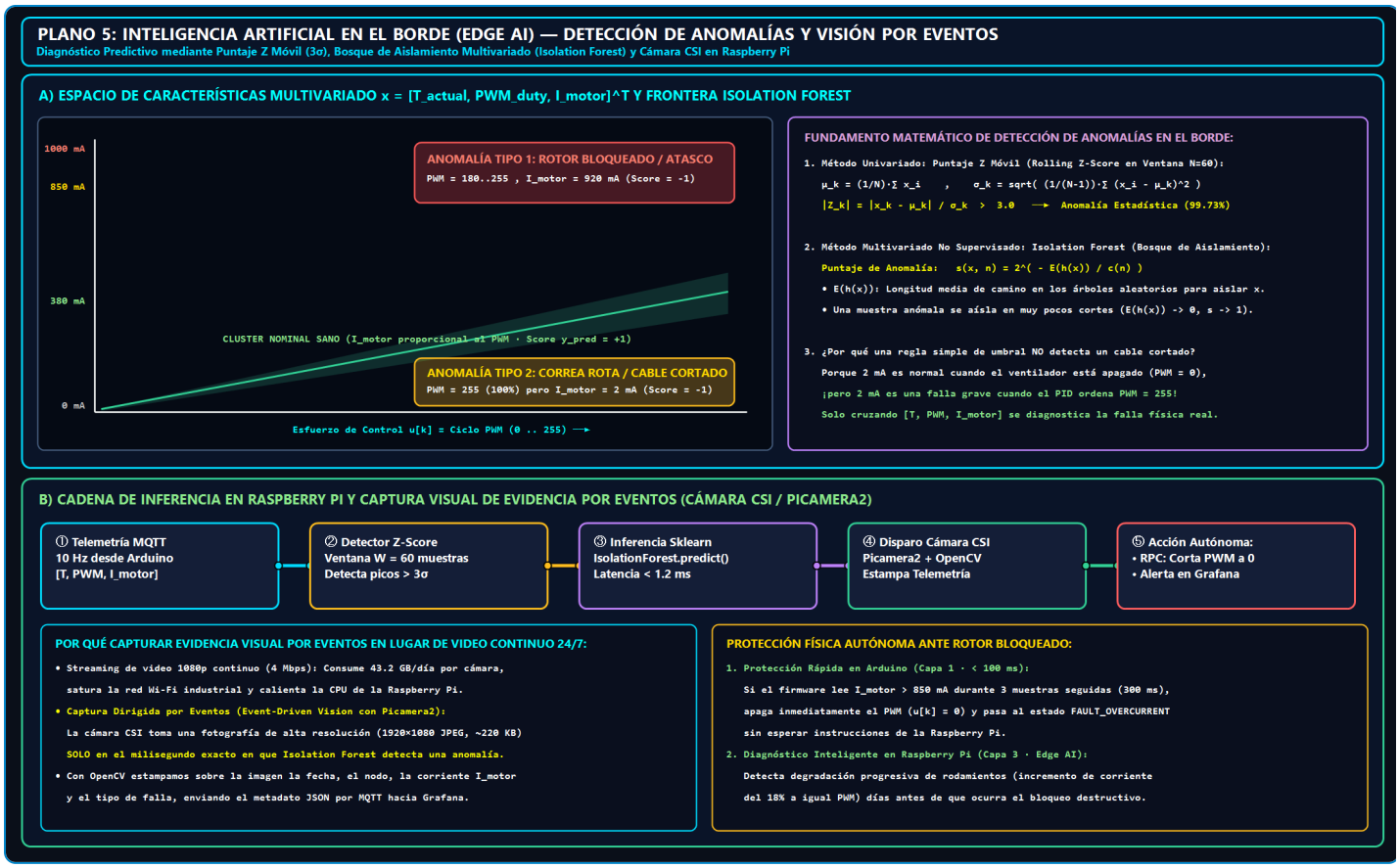
  ```bash
  python3 codigo_fuente/cliente_rpc_correlacionado_mqtt.py
  ```

- Verifique en consola que cada comando enviado en `planta/nodo_01/rpc/req` incluye un `req_id` único (por ejemplo, `req-8f3a19c2`) y que el hilo espera en `threading.Event.wait(timeout=1.50)` hasta recibir exactamente ese mismo `req_id` en `planta/nodo_01/rpc/res`.
- Desconecte momentáneamente el cable serial del Arduino y envíe una nueva consigna: compruebe cómo tras 1.50 s y 2 reintentos idempotentes, el servicio publica el evento de `Rollback` restaurando el indicador de la interfaz al valor real anterior.

---

6. Detección de Anomalías con `Edge AI` y Cámara `CSI` (`Plano 5`)

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos



Paso 6.1 — Ejecución del Detector `Isolation Forest` y Captura Forense por Eventos

- Inicie en terminales separadas el motor de inferencia y el servicio de cámara `CSI`:

  ```bash
  python3 codigo_fuente/detector_anomalias_edge_ai_isolation_forest.py
  python3 codigo_fuente/captura_visual_eventos_camara_csi.py
  ```

- **Prueba de Anomalía Tipo 1 (Rotor Bloqueado):** frene suavemente el eje del ventilador durante $400\ ms$. Observe cómo la corriente I_{motor} supera los $850\ mA$, el Arduino corta autónomamente el `PWM` a 0 en menos de $300\ ms$ y el motor `Edge AI` clasifica `ROTOR_BLOQUEADO_SOBRECORRIENTE`, disparando una fotografía `1080p` con marca de agua en `/var/lib/mecatrobotica/evidencia_visual/`.
- **Prueba de Anomalía Tipo 2 (Correa Rota / Cable Abierto):** desconecte el hilo positivo del ventilador mientras el controlador `PID` ordena $\text{PWM} = 255$. Aunque $I_{motor} \approx 2\ mA$ no viola ningún límite de sobrecorriente, el modelo multivariado **`Isolation Forest`** detecta inmediatamente que la combinación $[\text{PWM}=255,\ I_{motor}=2\ mA]$ se ubica fuera de la variedad nominal ($\text{Score} = -1$) y reporta `CORREA_ROTA_O_CABLE_ACTUADOR_ABIERTO`.

LISTADO DE MATERIALES E INSTRUMENTACIÓN (`BOM`) — MÓDULO 08

Hardware de Control en Lazo Cerrado `PID` , Etapa de Potencia `MOSFET PWM` , Sensórica `I2C` y Visión en el Borde (`CSI`)

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado (`PID`), Protocolo `RPC` , `Edge AI` y Captura Visual por Eventos

---

1. Computación de Supervisión (`Edge AI`) y Controlador de Tiempo Real (`PID`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
01	1	`SBC1`	**Raspberry Pi 4 Modelo B / Raspberry Pi 5 (`4 GB` o `8 GB` RAM)**	Supervisor de planta, cliente `RPC` correlacionado, motor `Isolation Forest` y captura de cámara `CSI` .
02	1	`U1`	**Arduino UNO R3 (`ATmega328P-PU` @ `16 MHz`)**	Ejecución determinista del lazo cerrado `PID` discreto cada $T_s = 100\ ms$ y protección de sobrecorriente.
03	1	`CAM1`	**Cámara Oficial Raspberry Pi `CSI` (`IMX219 8MP` o `IMX708 12MP`) con cinta `FFC` de 15/22 pines**	Captura fotográfica `1920x1080` dirigida por eventos ante detección de anomalías mecánicas o térmicas.

---

2. Etapa de Potencia `PWM` de $12\ V\ DC$ y Protección contra Transitorios Inductivos

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
04	1	`Q1`	**Transistor** `MOSFET` **Canal N de Nivel Lógico** `IRLZ44N` **(** `TO-220` **,** $V_{DSS}=55\ V$**,** $I_D=47\ A$**,** $R_{DS(on)}=0.022\ \Omega$ **@** $V_{GS}=5\ V$**)**	Conmutador de potencia en lado bajo (*Low-Side Switch*) gobernado por la señal `PWM` del pin `D9` ($490\ Hz$).
05	1	`HS1`	**Disipador de Aluminio Anodizado para cápsula** `TO-220` **(**$19 \times 15 \times 25\ mm$**) con tornillo** `M3` **y pasta térmica**	Evacuación térmica de pérdidas por conducción y conmutación del transistor `Q1`.
06	1	`D1`	**Diodo Rectificador de Rueda Libre** `1N4007` **(** `DO-41` **,** $1000\ V$**,** $1.0\ A$**) o** `1N5819` **(** `Schottky` $40\ V$**)**	Supresión del pico inductivo inverso ($V_L = -L \cdot dI/dt$) en cada corte del ciclo `PWM`.
07	1	`R_GATE`	**Resistencia de Película Metálica de** $220\ \Omega$ **(**$1/4\ W$**,** $\pm 1\%$**)**	Limita el pico de corriente de carga de la capacitancia de compuerta (C_{iss}) y amortigua oscilaciones parásitas (*ringing*).
08	1	`R_PD`	**Resistencia de Anclaje (** `Pull-Down` **) de** $10\ k\Omega$ **(**$1/4\ W$**,** $\pm 1\%$**)**	Mantiene la compuerta (`Gate`) firmemente en $0\ V$ durante el reinicio o alta impedancia del microcontrolador.
09	1	`ACT1`	**Planta Térmica / Ventilador Axial Industrial de** $12\ V\ DC$ **(**$380\ mA$ **nominal,** $> 850\ mA$ **rotor bloqueado)**	Actuador físico del lazo de regulación térmica y banco de pruebas de anomalías mecánicas.
10	1	`PSU1`	**Fuente Conmutada Industrial de** $12\ V\ DC\ /\ 3.0\ A$ **(**$36\ W$**) con riel negativo común en estrella**	Alimentación dedicada para la etapa de potencia de $12\ V$, desacoplada del riel lógico de $5\ V$.

3. Sensórica de Realimentación de Estado (`T_actual` e `I_motor`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
11	1	`U2`	**Módulo Sensor de Corriente y Tensión de Lado Alto** `INA219` **(** `I2C 0x40` **) con Shunt Kelvin de** $0.100\ \Omega$ **(**1%**,** $2\ W$**)**	Medición en tiempo real de $I_{motor}[k]$ y $V_{bus}[k]$ para protección de sobrecorriente y vector de `Edge AI`.
12	1	`U3`	**Sonda Ambiental Digital** `Bosch BME280` **(** `I2C 0x76` **) o** `DS18B20` **impermeable**	Medición de la variable controlada $y[k] = T_{actual}$ con resolución de $0.01\ ^\circ C$.

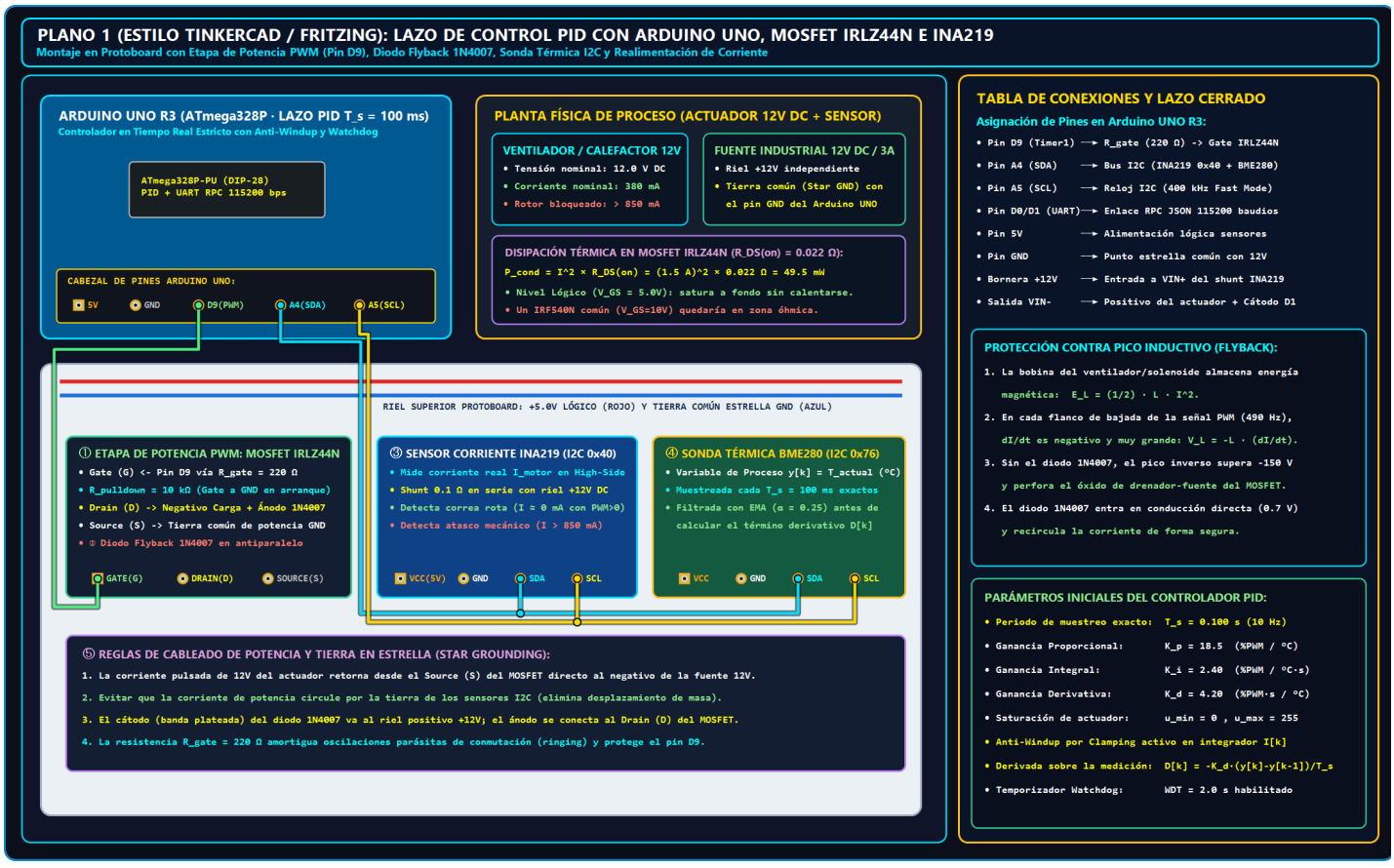
4. Insumos de Construcción para Placa Impresa Soldada (`PCB FR-4`) y Prototipo (`Tinkercad`)

Ítem	Cant.	Designador	Componente / Especificación Técnica	Función en el Sistema
13	1	`PCB1`	**Placa Virgen o Fabricada `FR-4` de Doble Capa (1.6 mm, cobre 1 oz/ft^2, pistas de potencia de 2.50 mm)**	Montaje definitivo por soldadura libre de falsos contactos y con partición de tierra `AGND` / `PGND`.
14	1	`SOCK1`	**Zócalo Torneado `DIP-28` de paso 2.54 mm + Cristal `HC-49S` 16.000 MHz + 2× 22 pF NP0**	Alojamiento seguro del microcontrolador `ATmega328P-PU` sin exponerlo al calor directo del cautín.
15	2	`TB1`, `TB2`	**Borneras de Tornillo para Circuito Impreso `KF301-2P` (paso 5.08 mm, 300 V / 15 A)**	Fijación mecánica firme de los cables de la fuente de 12 V DC y del actuador.
16	1	`F1`	**Fusible Polimérico Rearmable `PTC` de 3.0 A (`PPTC` radial)**	Protección contra cortocircuitos francos en la línea de +12 V.

5. Planos de Referencia para Inspección de Materiales y Montaje

5.1 Plano de Prototipado en Protoboard (`Plano 1`)

📐 Plano 1: Montaje en Protoboard del Lazo Cerrado PID

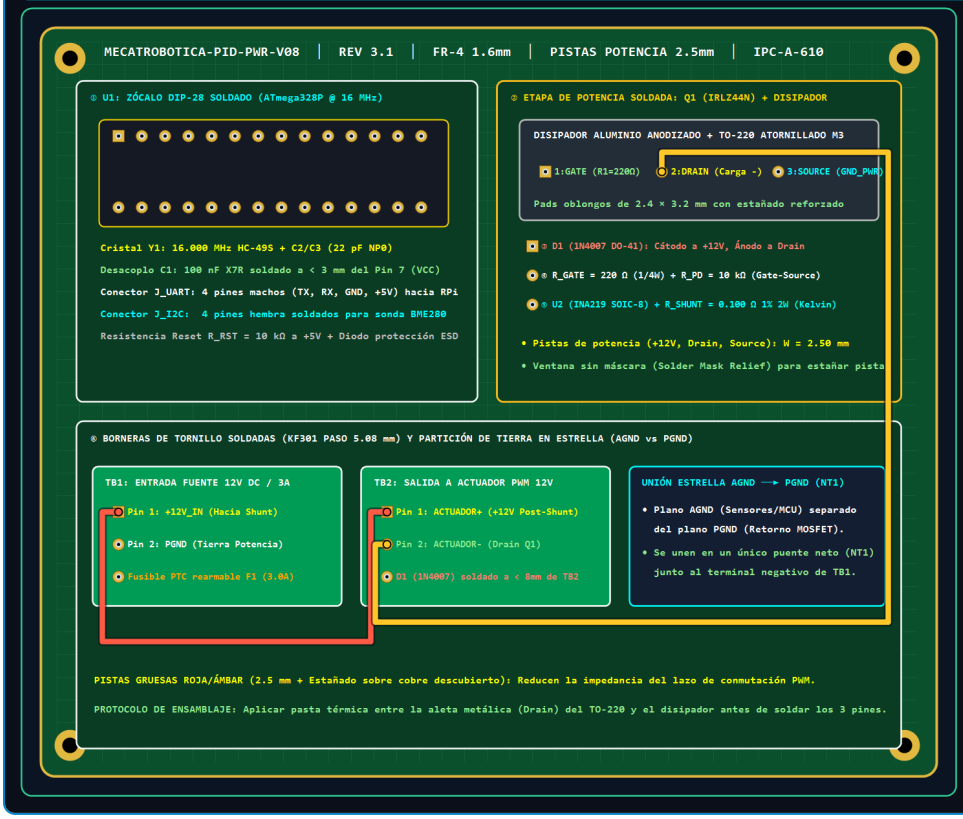


5.2 Plano de Circuito Impreso `FR-4` con Montaje por Soldadura (`Plano 2`)

📐 Plano 2: Placa PCB Soldada del Controlador PID de Potencia

PLANO 2 (PLACA PCB FR-4 CON MONTAJE POR SOLDADURA): CONTROLADOR PID DE POTENCIA E INSTRUMENTACIÓN

Circuito Impreso de Doble Capa con Pistas de Cobre de 2.5 mm para 12V/3A, Zócalo DIP-28 Soldado, MOSFET TO-220 con Disipador y Shunt Kelvin



INGENIERÍA DE PCB DE POTENCIA (IPC-2221)

Dimensionamiento de Pistas y Soldadura THT:

- Corriente máxima de diseño: $I_{max} = 3.0$ A continuos
- Espesor de cobre estándar: 1 oz/ft² (35 µm)
- Ancho de pista requerido (IPC-2221 para $\Delta T = 10$ °C):
  $W_{min} = 1.58$ mm ⟶ Diseñado con $W = 2.50$ mm
- Técnica de ventana sin máscara (Solder Mask Relief): Permite aportar soldadura de estaño-plata sobre la pista de Drain y Source, duplicando su sección útil.

POR QUÉ NO USAR PROTOBOARD CON CARGAS PWM > 1 A:

1. Los flejes elásticos de una protoboard tienen una resistencia de contacto de 80 a 250 mΩ por inserción.
2. A 2.5 A, los contactos se calientan, ablandan el plástico ABS y falsean la tensión de compuerta V_{GS}.
3. El rebote de tierra (Ground Bounce) por los pulsos PWM corrompe las lecturas del sensor térmico I2C.
4. En la placa PCB soldada con tierras separadas (AGND y PGND), el ruido conducido cae más de 40 dB.

LISTA DE INSPECCIÓN POST-SOLDADURA:

1. Verificar con óhmetro resistencia entre Gate y Source: Debe medir exactamente 10.0 kΩ ($R_{pulldown}$).
2. Verificar polaridad de D1 (1N4007): Banda del cátodo hacia +12V (si se invierte, provoca cortocircuito).
3. Comprobar ausencia de puentes de estaño entre los 3 pines del TO-220 (paso 2.54 mm).
4. Limpiar residuos de resina flux con IPA 99.8% antes de insertar el ATmega328P en su zócalo DIP-28.

MANUAL DE INGENIERÍA: CONTROL `PID` DISCRETO, PROTOCOLO `RPC` Y `EDGE AI` — MÓDULO 08

Fundamentos Matemáticos de Control en Tiempo Discreto, Conmutación `MOSFET PWM`, Correlación Asíncrona `MQTT` y Detección Multivariada con `Isolation Forest`

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado, Llamadas a Procedimiento Remoto (`RPC`) e Inteligencia Artificial en el Borde

---

1. Arquitectura de Control Jerárquico: ¿Por qué nunca cerrar un lazo `PID` rápido a través de `Wi-Fi` o `MQTT`?

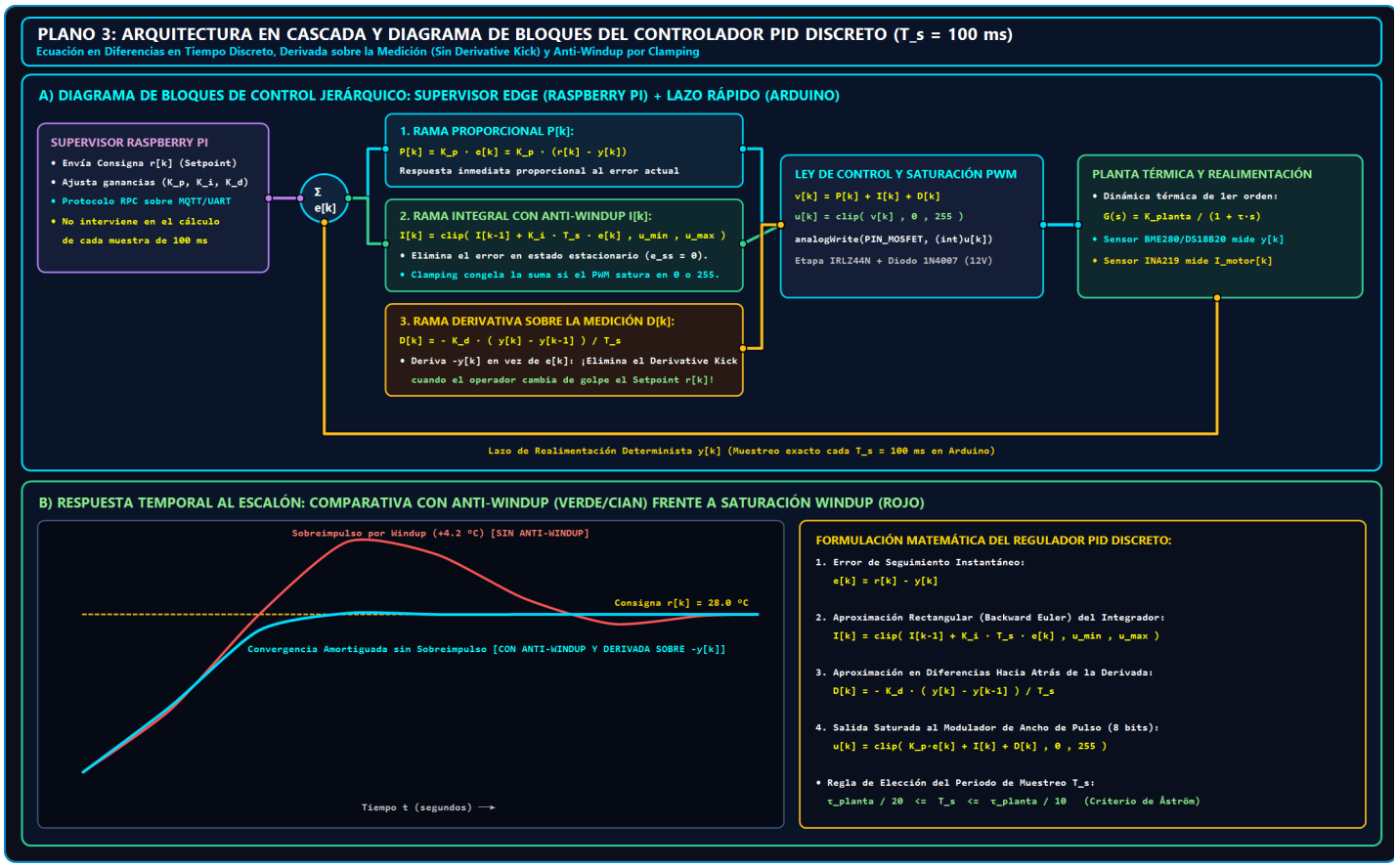
En la ingeniería de sistemas ciberfísicos e industriales, la primera regla de diseño consiste en **separar la capa de control regulatorio en tiempo real estricto (`Hard Real-Time`) de la capa de supervisión y analítica en tiempo real blando (`Soft Real-Time`)**:

- *Incertidumbre temporal (Jitter) de la red:* un enlace `Wi-Fi` o una cola de mensajes `MQTT` sobre `TCP/IP` presenta latencias variables que oscilan entre $3\ ms$ en condiciones ideales y más de $800\ ms$ cuando ocurre una retransmisión `TCP` o una recolección de basura en el sistema operativo.
- **Degradación del margen de fase:** en un controlador proporcional-integral-derivativo (`PID`), un retardo de transporte variable $\tau_d(t)$ introduce un desfase adicional $\Delta\phi(\Omega) = -\Omega \cdot \tau_d(t)$ que reduce el margen de fase del sistema hasta volverlo inestable u oscilatorio.
- **Solución arquitectónica en cascada:**
- **Nivel 1 (`Arduino UNO / ATmega328P`):** ejecuta el algoritmo `PID` discreto con un periodo de muestreo determinista $T_s = 100\ ms$ ($10\ Hz$), lee directamente los sensores `I2C` (`BME280` e `INA219`), comanda el modulador `PWM` del transistor `MOSFET IRLZ44N` y aplica enclavamiento autónomo de seguridad si la corriente supera $850\ mA$.
- **Nivel 2 (`Raspberry Pi 4 / 5`):** envía cambios de consigna $r[k]$ (`Setpoint`) o ajustes de ganancias (K_p, K_i, K_d) mediante transacciones `RPC` correlacionadas, ejecuta el motor de aprendizaje automático `Isolation Forest` y captura evidencia visual por cámara `CSI` .

---

2. Formulación Matemática del Controlador `PID` en Tiempo Discreto (`Plano 3`)

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup



En el dominio continuo, la ley de control `PID` ideal relaciona la señal de mando $u(t)$ con el error de seguimiento $e(t) = r(t) - y(t)$:

$$u(t) = K_p \, e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \, \frac{de(t)}{dt}$$

Al discretizar para un microcontrolador con periodo de muestreo constante $T_s = t_k - t_{k-1} = 0.100\ s$:

2.1 Término Proporcional $P[k]$

$$P[k] = K_p \cdot e[k] = K_p \cdot \left(r[k] - y[k] \right)$$

2.2 Término Integral $I[k]$ con Anti-Windup por Recorte Condicional (*Clamping*)

Aplicando la aproximación rectangular hacia atrás (*Backward Euler*) y acotando el acumulador dentro de los límites físicos del modulador `PWM` de 8 bits ($[u_{min}, u_{max}] = [0, 255]$):

$$I[k] = \text{clip}\left(I[k-1] + K_i \cdot T_s \cdot e[k] \;;\; u_{min} \;;\; u_{max} \right)$$

Además, si la suma total pre-saturación $v[k] = P[k] + I[k] + D[k]$ excede 255 y el error $e[k] > 0$ (o es menor que 0 con $e[k] < 0$), el integrador se congela para evitar el fenómeno de **arrollamiento integral (*Integral Windup)**, eliminando el sobreimpulso térmico.

2.3 Término Derivativo sobre la Medición $D[k]$ (Eliminación del *Derivative Kick*)

Si derivamos el error $e[k] = r[k] - y[k]$, cualquier cambio escalón en la consigna $r[k]$ enviado por el operador desde Grafana produce una pendiente instantánea $\Delta r / T_s$ enorme que satura de golpe el actuador (*Derivative Kick*). Como $r[k]$ es constante por tramos, $\frac{dr}{dt} = 0$ salvo en el instante del cambio de consigna, por lo que reemplazamos $\frac{de}{dt}$ por $-\frac{dy}{dt}$:

$$D[k] = -K_d \cdot (y[k] - y[k-1]) / (T_s)$$

Previamente filtramos la temperatura cruda con un **filtro de media móvil exponencial (`EMA`)** con factor $\alpha = 0.25$:

$$y_{filt}[k] = \alpha \cdot y_{raw}[k] + (1 - \alpha) \cdot y_{filt}[k-1]$$

---

3. Física de Conmutación `MOSFET PWM` y Supresión de Pico Inductivo

- **Pérdidas por conducción en el `MOSFET` de nivel lógico `IRLZ44N`:**

  $$P_{cond} = I_{carga}^2 \cdot R_{DS(on)} = (1.5\ A)^2 \times 0.022\ \Omega = 0.0495\ W \approx 49.5\ mW$$

  Con una resistencia térmica unión-ambiente $R_{\theta JA} = 62\ ^\circ C/W$, el incremento de temperatura sin siquiera disipador para $1.5\ A$ es de apenas $\Delta T = 0.0495 \times 62 \approx 3.1\ ^\circ C$.

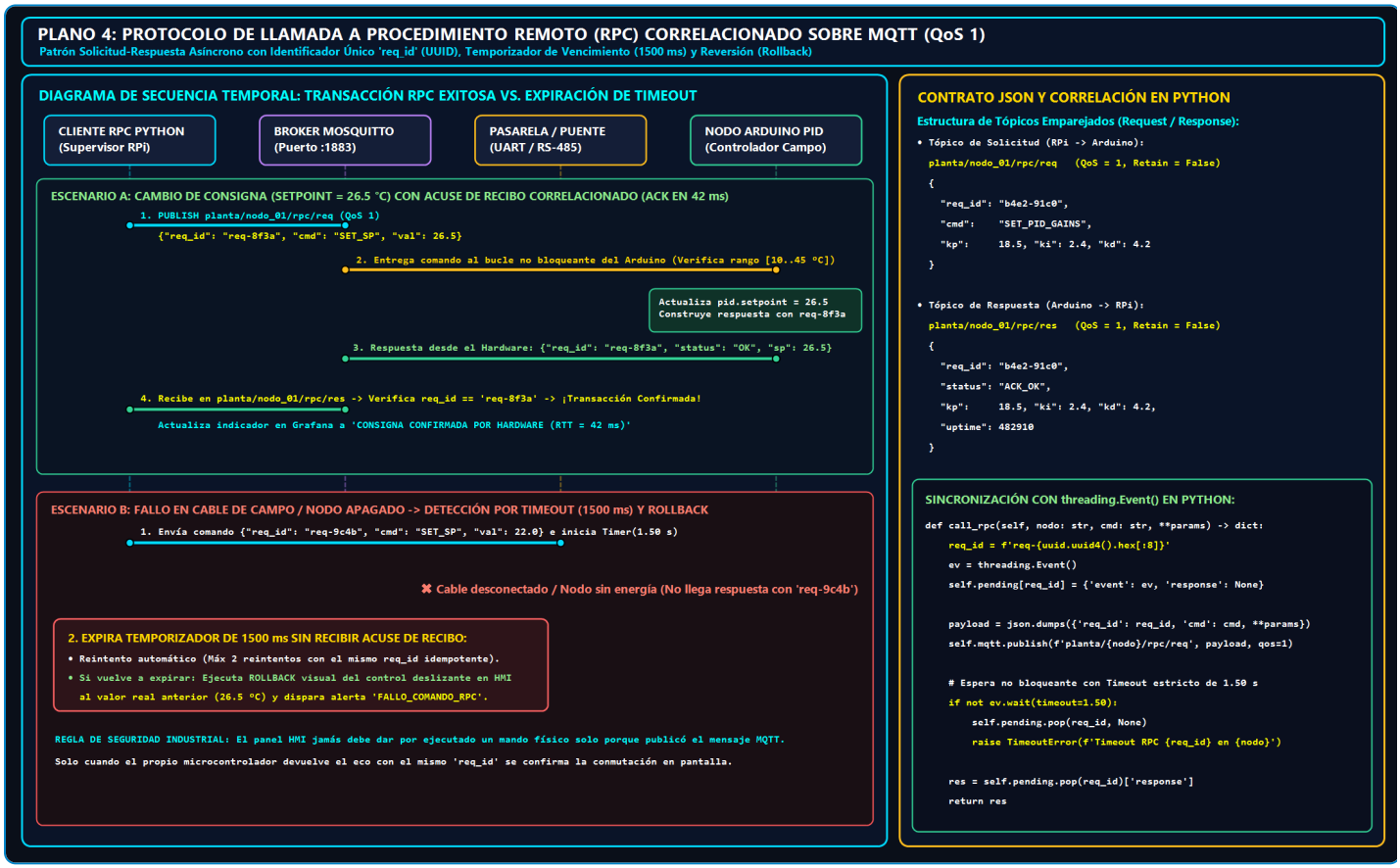
- **Física del diodo volante (`Flyback 1N4007`):**

  El devanado de un ventilador o electroválvula almacena energía magnética $E_L = (1)/(2)\ L\ I^2$. Cuando el transistor `IRLZ44N` pasa de conducción a corte en pocos nanosegundos, la derivada $(dI)/(dt)$ es negativa y de gran magnitud, induciendo una fuerza contraelectromotriz $V_L = -L\ (dI)/(dt)$ que puede superar $+150\ V$ en el drenador y destruir el transistor por avalancha. El diodo `1N4007` en antiparalelo entra en conducción directa a $0.7\ V$, ofreciendo un camino seguro de recirculación para la corriente inductiva.

---

4. Protocolo `RPC` Correlacionado sobre `MQTT` (`Plano 4`)

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



Dado que `MQTT` es un protocolo intrínsecamente asíncrono de publicación/suscripción, publicar un comando en un tópico no garantiza que el microcontrolador de campo lo haya recibido ni ejecutado. Para construir una **Llamada a**

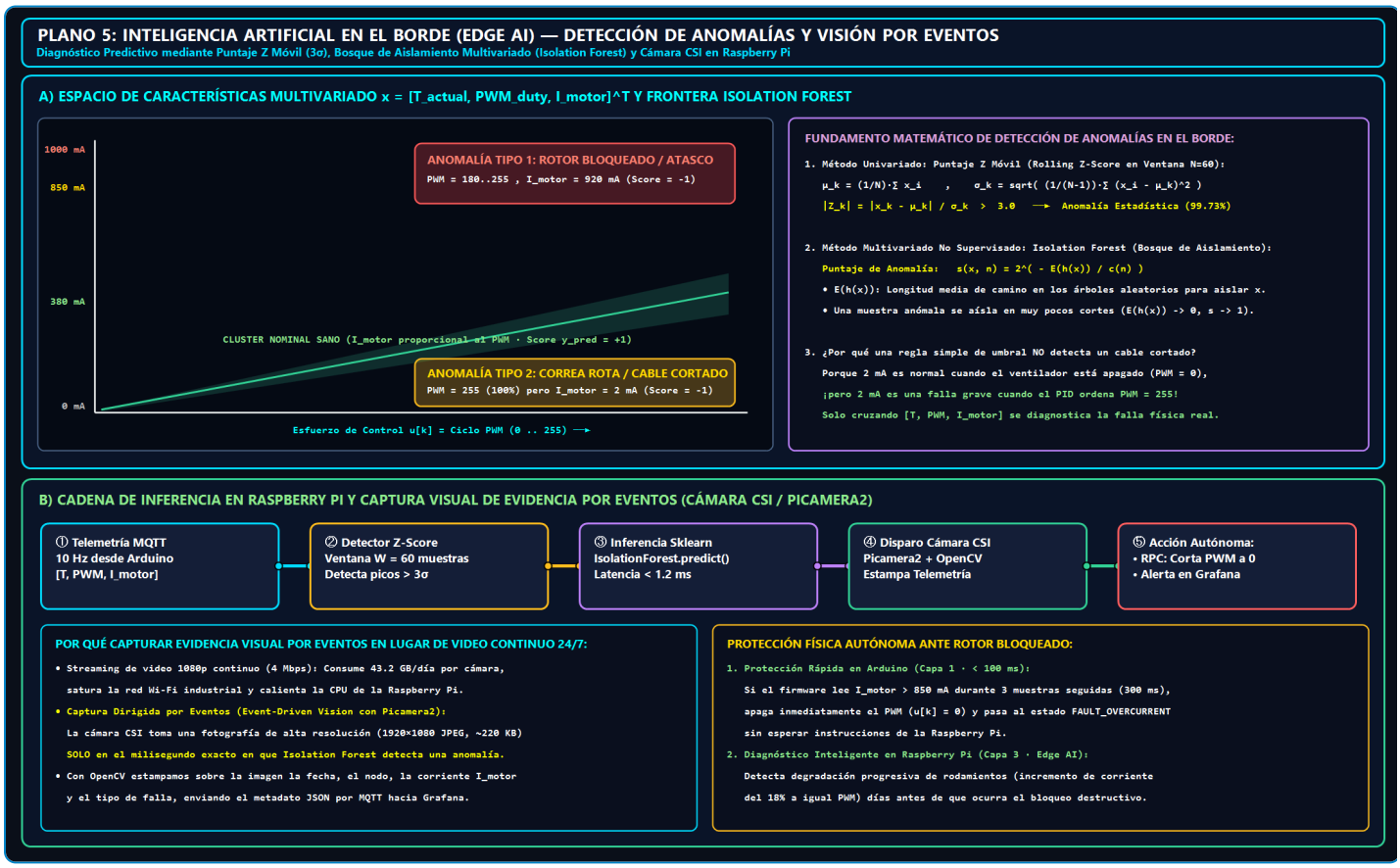
Procedimiento Remoto (`RPC`) de grado industrial:

- **Tópicos desacoplados de solicitud y respuesta:** `planta/nodo_01/rpc/req` y `planta/nodo_01/rpc/res` con `QoS = 1` y `Retain = False`.
- **Correlación estricta por `req_id` (`UUID`):** cada solicitud inyecta un identificador único (ej. `"req_id": "req-8f3a"`). El hilo invocador registra un objeto `threading.Event()` en un diccionario protegido por cerrojo (`Lock`) y se suspende con `event.wait(timeout=1.50)`.
- **Idempotencia y Reversión (`Rollback`):** si el temporizador de $1500\ ms$ expira tras los reintentos configurados, el cliente supervisor revierte el control deslizante en la interfaz HMI al último valor confirmado por el hardware y emite una alerta de fallo de comunicación.

---

5. Inteligencia Artificial en el Borde (`Edge AI`) y Visión por Eventos (`Plano 5`)

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos



5.1 Puntaje Z Móvil (*Rolling Z-Score*) sobre Ventana Deslizante ($N = 60$)

Para detectar picos abruptos en una serie temporal univariada, calculamos en tiempo real la media μ_k y la desviación estándar muestral σ_k:

$$\mu_k = \frac{1}{N}\sum_{i=k-N+1}^{k} x_i \quad , \quad \sigma_k = \sqrt{\frac{1}{N-1}\sum_{i=k-N+1}^{k}\left(x_i - \mu_k\right)^2} \quad , \quad Z_k = \frac{|x_k - \mu_k|}{\sigma_k}$$

Toda lectura con $Z_k > 3.0$ se desvía más de tres desviaciones estándar de su comportamiento reciente ($p < 0.0027$).

5.2 Detección Multivariada No Supervisada con `Isolation Forest`

Una regla simple de umbral sobre la corriente I_{motor} **no puede detectar una correa rota ni un cable cortado**, porque $I_{motor} = 2\ mA$ es completamente normal cuando el ventilador está apagado ($PWM = 0$). Sin embargo, $I_{motor} = 2\ mA$

cuando el regulador `PID` ordena $\mathrm{PWM} = 255$ constituye una falla crítica de lazo abierto.

Al alimentar el algoritmo **Bosque de Aislamiento (`Isolation Forest`)** con el vector de estado tridimensional:

$$\mathbf{x}_k = \begin{bmatrix} T_{actual}[k] \\ PWM_{duty}[k] \\ I_{motor}[k] \end{bmatrix}$$

el conjunto de 100 árboles aleatorios particiona el espacio de características y calcula el puntaje de anomalía:

$$s(\mathbf{x}, n) = 2^{-\frac{E(h(\mathbf{x}))}{c(n)}}$$

donde $E(h(\mathbf{x}))$ es la longitud media de camino necesaria para aislar la muestra $\mathbf{x}$. Las muestras anómalas (tanto el **rotor bloqueado** con $I_{motor} > 850\ mA$ como la **correa rota** con $PWM{=}255$ e $I_{motor} \approx 0\ mA$) quedan aisladas cerca de la raíz de los árboles ($E(h(\mathbf{x})) \to 0 \implies s \to 1$), disparando en menos de $1.2\ ms$ la captura de una fotografía forense `1080p` con la cámara `CSI` y el apagado preventivo del actuador.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 08

Planos de Alta Legibilidad en Protoboard (`Tinkercad`), Circuito Impreso Soldado (`PCB FR-4`), Lazo `PID` , Secuencia `RPC` y `Edge AI`

Curso: Ingeniería en Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi

Módulo: 08 — Control en Lazo Cerrado (`PID`), Protocolo `RPC` , `Edge AI` y Visión por Eventos

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje en Protoboard del Lazo Cerrado `PID`

Este primer plano presenta el montaje experimental en protoboard con tipografía ampliada de alta legibilidad y llamadas numeradas (① a ⑤), interconectando el **Arduino UNO R3**, la etapa de potencia `MOSFET IRLZ44N` con diodo volante `1N4007` , el sensor de corriente de lado alto `INA219 (0x40)` y la sonda térmica `BME280 (0x76)` .

📐 Plano 1: Montaje en Protoboard del Lazo de Control PID con MOSFET IRLZ44N e INA219

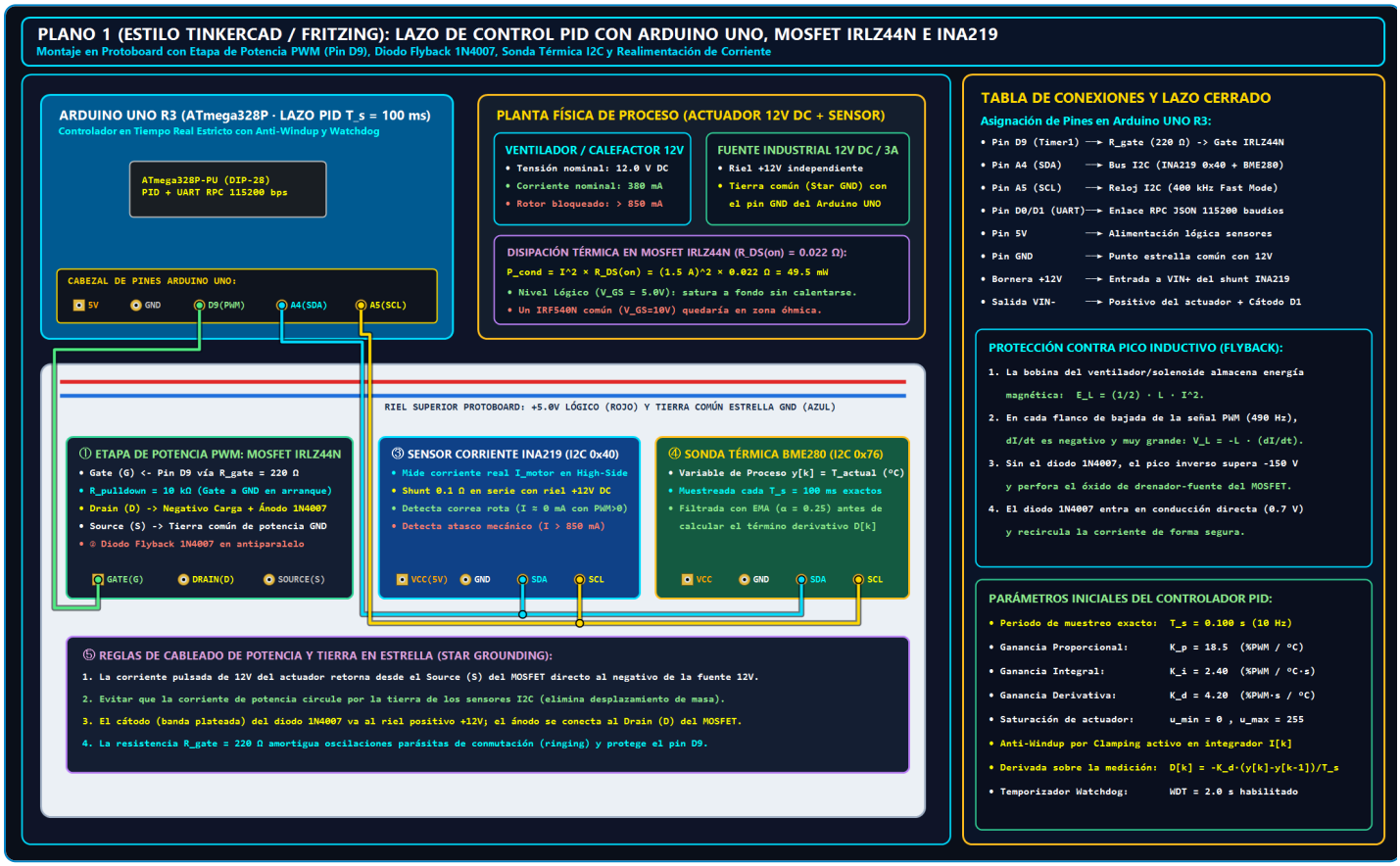


Tabla de Asignación de Terminales y Funciones Físicas

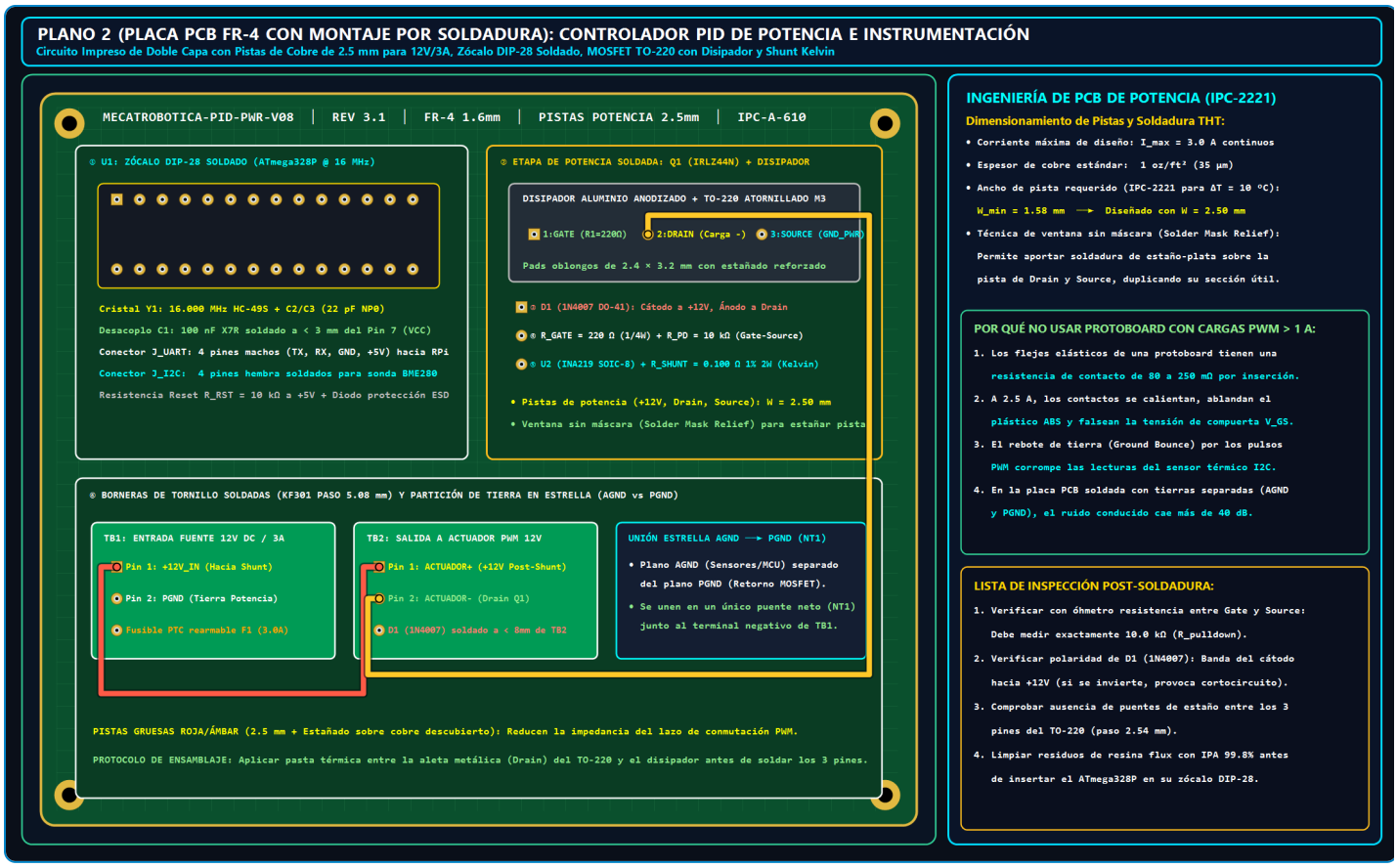
Origen (`Arduino / Fuente`)	Destino en Etapa de Campo	Color de Cable	Función Eléctrica y Regla de Diseño
`Pin D9` (`Timer1 PWM`)	Compuerta (`Gate`) del `IRLZ44N` vía $R_{gate} = 220\ \Omega$	Verde	Señal `PWM` de $490\ Hz$ (0 a 255). Incluye $R_{pulldown} = 10\ k\Omega$ entre `Gate` y `GND`.
`Pin A4` (`SDA`)	Pines `SDA` de `INA219 (0x40)` y `BME280 (0x76)`	Cian	Línea de datos serie `I2C` operando a $400\ kHz$ (*Fast Mode*).
`Pin A5` (`SCL`)	Pines `SCL` de `INA219 (0x40)` y `BME280 (0x76)`	Amarillo	Línea de reloj serie `I2C` sincronizada por el `ATmega328P`.
`Fuente +12V DC`	Borne `VIN+` del `Shunt` de $0.1\ \Omega$ (`INA219`)	Rojo	Entrada de potencia de lado alto (*High-Side*) protegida por fusible `PTC` de $3\ A$.
Borne `VIN-` (`INA219`)	Positivo del Actuador $12\ V$ + Cátodo de `1N4007`	Ámbar	Alimentación medida hacia el actuador y punto de sujeción del diodo de rueda libre.
Negativo del Actuador $12\ V$	Drenador (`Drain`) del `IRLZ44N` + Ánodo de `1N4007`	Amarillo	Conmutación de lado bajo (*Low-Side*) y absorción del pico inductivo inverso.
Surtidor (`Source`) `IRLZ44N`	Punto Estrella `GND` de la Fuente de $12\ V\ DC$	Azul / Negro	Retorno directo de corriente pulsada sin contaminar la tierra analógica de los sensores.

---

2. Plano 2 (Placa `PCB FR-4` con Montaje por Soldadura): Controlador `PID` de Potencia

Para aplicaciones permanentes donde las corrientes pulsadas `PWM` de 1 a $3\ A$ degradarían los contactos elásticos de un protoboard, este segundo plano detalla el diseño de una **placa de circuito impreso (`PCB FR-4`) de doble capa con montaje por soldadura (`THT` / `SMD`)**, pistas de potencia de $2.50\ mm$ con ventana de estañado (*Solder Mask Relief*) y partición de tierras en estrella (`AGND` vs. `PGND`).

📐 Plano 2: Placa PCB FR-4 de Doble Capa con Montaje por Soldadura para Controlador PID

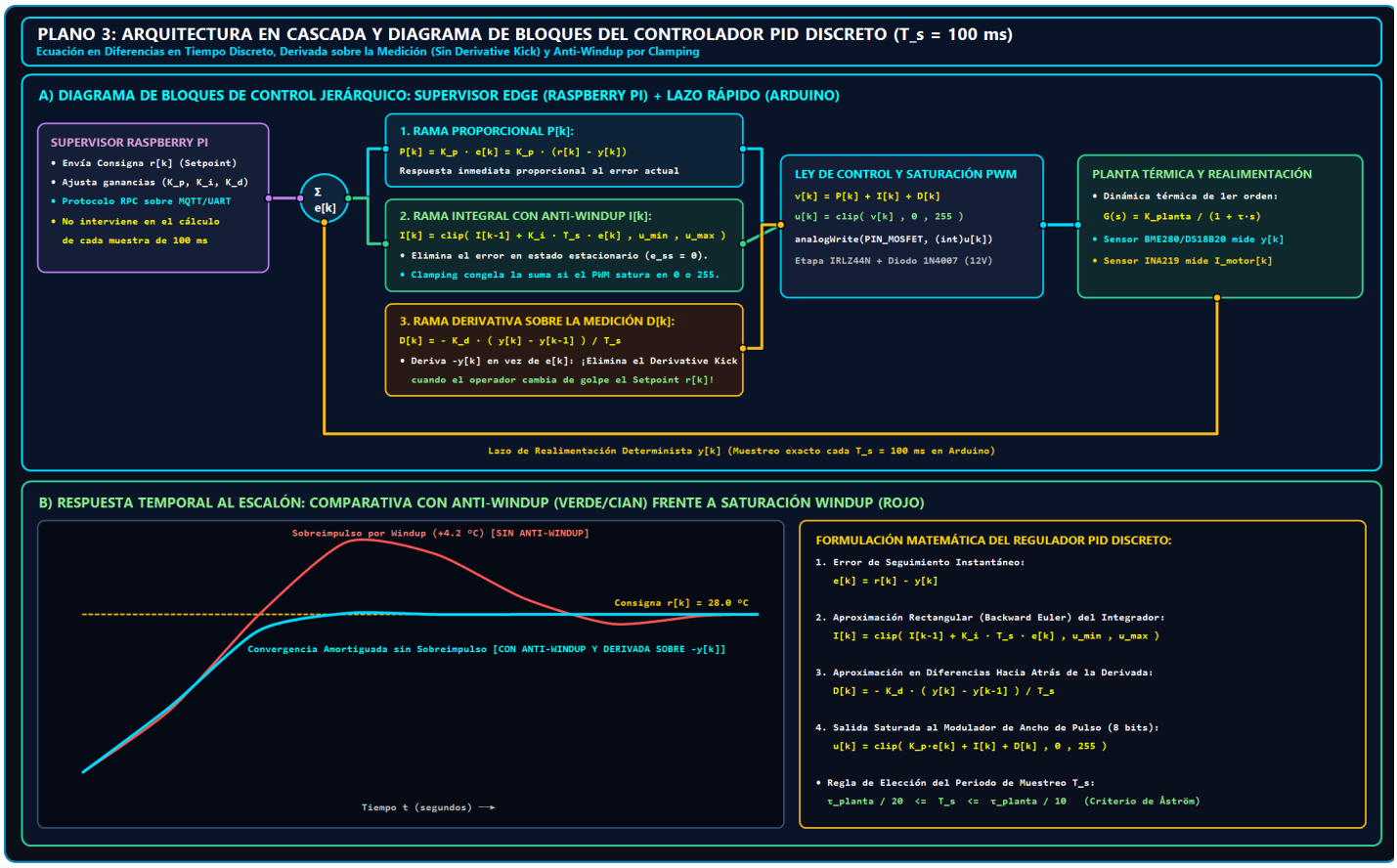


---

3. Plano 3: Arquitectura en Cascada y Diagrama de Bloques del Regulador `PID` Discreto

Detalla la estructura matemática de las tres ramas paralelas ($P[k]$, $I[k]$ con *Clamping Anti-Windup*, y $D[k]$ calculada sobre $-y[k]$) junto con las curvas comparativas de respuesta temporal al escalón.

📐 Plano 3: Diagrama de Bloques del Controlador PID Discreto y Respuesta con Anti-Windup

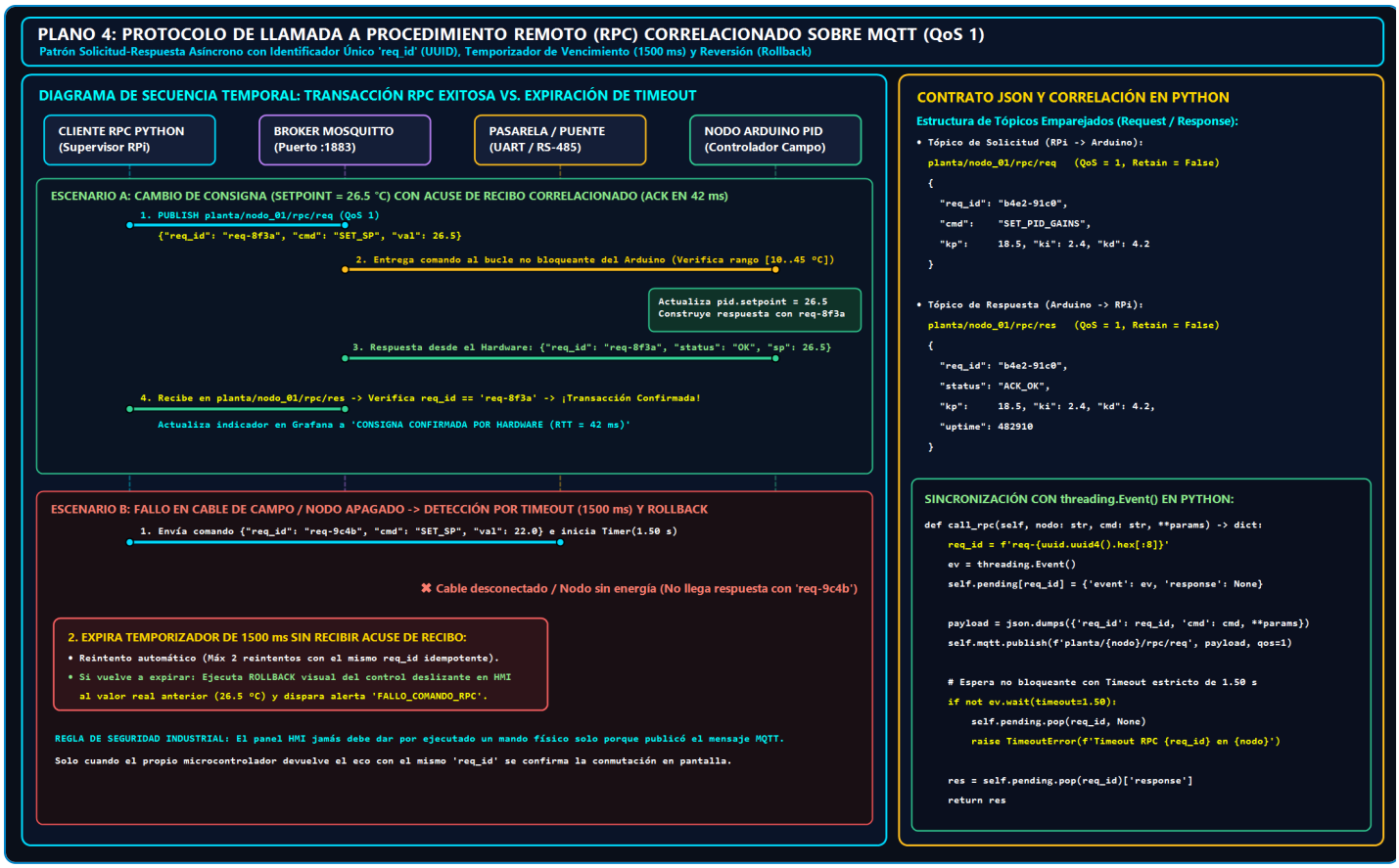


---

4. Plano 4: Diagrama de Secuencia del Protocolo `RPC` Correlacionado sobre `MQTT`

Representa el flujo temporal completo de una transacción exitosa con acuse de recibo (`ACK`) correlacionado en $42\ ms$ frente a un escenario de corte de cable detectado por expiración de temporizador ($1500\ ms$) con reversión automática (`Rollback`).

📐 Plano 4: Diagrama de Secuencia del Protocolo RPC Correlacionado con Timeout y Rollback



---

5. Plano 5: Espacio de Características `Isolation Forest` y Visión por Eventos (`CSI`)

Ilustra la separación geométrica en el espacio tridimensional $[T_{actual}, PWM, I_{motor}]$ entre el corredor nominal sano, la región de rotor bloqueado y la región de correa rota/cable abierto, así como la cadena de captura fotográfica `1080p` con `Picamera2` y `OpenCV`.

📐 Plano 5: Arquitectura Edge AI con Puntaje Z Móvil, Isolation Forest y Cámara CSI por Eventos

PLANO 5: INTELIGENCIA ARTIFICIAL EN EL BORDE (EDGE AI) — DETECCIÓN DE ANOMALÍAS Y VISIÓN POR EVENTOS

Diagnóstico Predictivo mediante Puntaje Z Móvil (3σ), Bosque de Aislamiento Multivariado (Isolation Forest) y Cámara CSI en Raspberry Pi

A) ESPACIO DE CARACTERÍSTICAS MULTIVARIADO x = [T_actual, PWM_duty, I_motor]^T Y FRONTERA ISOLATION FOREST



FUNDAMENTO MATEMÁTICO DE DETECCIÓN DE ANOMALÍAS EN EL BORDE:

1. Método Univariado: Puntaje Z Móvil (Rolling Z-Score en Ventana N=60):

   $\mu_k = (1/N)\cdot\Sigma\ x_i$, $\sigma_k = \text{sqrt}((1/(N-1))\cdot\Sigma\ (x_i - \mu_k)^2)$

   $|Z_k| = |x_k - \mu_k| / \sigma_k > 3.0 \longrightarrow$ Anomalía Estadística (99.73%)

2. Método Multivariado No Supervisado: Isolation Forest (Bosque de Aislamiento):

   Puntaje de Anomalía: $s(x, n) = 2^{(- E(h(x)) / c(n))}$

   • E(h(x)): Longitud media de camino en los árboles aleatorios para aislar x.

   • Una muestra anómala se aísla en muy pocos cortes (E(h(x)) -> 0, s -> 1).

3. ¿Por qué una regla simple de umbral NO detecta un cable cortado?

   Porque 2 mA es normal cuando el ventilador está apagado (PWM = 0),

   ¡pero 2 mA es una falla grave cuando el PID ordena PWM = 255!

   Solo cruzando [T, PWM, I_motor] se diagnostica la falla física real.

B) CADENA DE INFERENCIA EN RASPBERRY PI Y CAPTURA VISUAL DE EVIDENCIA POR EVENTOS (CÁMARA CSI / PICAMERA2)

① Telemetría MQTT
10 Hz desde Arduino
[T, PWM, I_motor]

② Detector Z-Score
Ventana W = 60 muestras
Detecta picos > 3σ

③ Inferencia Sklearn
IsolationForest.predict()
Latencia < 1.2 ms

④ Disparo Cámara CSI
Picamera2 + OpenCV
Estampa Telemetría

⑤ Acción Autónoma:
• RPC: Corta PWM a 0
• Alerta en Grafana

POR QUÉ CAPTURAR EVIDENCIA VISUAL POR EVENTOS EN LUGAR DE VIDEO CONTINUO 24/7:

• Streaming de video 1080p continuo (4 Mbps): Consume 43.2 GB/día por cámara, satura la red Wi-Fi industrial y calienta la CPU de la Raspberry Pi.

• Captura Dirigida por Eventos (Event-Driven Vision con Picamera2):
  La cámara CSI toma una fotografía de alta resolución (1920×1080 JPEG, ~220 KB) SOLO en el milisegundo exacto en que Isolation Forest detecta una anomalía.

• Con OpenCV estampamos sobre la imagen la fecha, el nodo, la corriente I_motor y el tipo de falla, enviando el metadato JSON por MQTT hacia Grafana.

PROTECCIÓN FÍSICA AUTÓNOMA ANTE ROTOR BLOQUEADO:

1. Protección Rápida en Arduino (Capa 1 · < 100 ms):
   Si el firmware lee I_motor > 850 mA durante 3 muestras seguidas (300 ms), apaga inmediatamente el PWM (u[k] = 0) y pasa al estado FAULT_OVERCURRENT sin esperar instrucciones de la Raspberry Pi.
2. Diagnóstico Inteligente en Raspberry Pi (Capa 3 · Edge AI):
   Detecta degradación progresiva de rodamientos (incremento de corriente del 18% a igual PWM) días antes de que ocurra el bloqueo destructivo.

Capítulo 28 · Seguridad industrial: IEC 62443, PKI, TLS y ACL

Conceptos clave

Verifica la cadena de confianza y la identidad del servidor. Restringe la lectura y escritura de cada cliente a sus propios temas.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 09

Ciberseguridad IoT Industrial, Cifrado MQTTS (`TLS 1.3 :8883`), PKI X.509 (`OpenSSL SAN`), Cortafuegos `UFW` , `WireGuard` y Supervivencia 24/7 (`log2ram` + `Watchdog`)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 09 — Seguridad Ciberfísica, Endurecimiento de Servidores Edge y Alta Disponibilidad

---

1. Objetivos de Ingeniería del Laboratorio

- Construir y cablear el **circuito de seguridad física de gabinete y supervisión de hardware** (tanto en protoboard estilo Tinkercad como en placa impresa PCB FR-4 soldada), integrando un microinterruptor de apertura de puerta (*Tamper Switch* `NC`), un selector de llave física de mantenimiento (*Keylock*), aislamiento por optoacopladores `PC817` y el circuito de latido (*Heartbeat*) del procesador.
- Desplegar una **Infraestructura de Clave Pública (`PKI X.509`)** privada de planta con `OpenSSL` (`aprovisionar_pki_x509_openssl_san.sh`), generando una Autoridad Certificadora Raíz (`RSA 4096 bits`) y certificados de servidor/cliente con extensión **Subject Alternative Name (`SAN` - RFC 9525)**.
- Reconfigurar el broker **Eclipse Mosquitto 2.x** para operar exclusivamente con cifrado `TLS v1.3` **en el puerto** `8883` hacia la red local y aplicar una **Lista de Control de Acceso (`ACL`)** bajo el Principio de Mínimo Privilegio.
- Endurecer el sistema operativo Linux de la Raspberry Pi mediante el cortafuegos de estado `UFW` , protección contra fuerza bruta `Fail2ban` y acceso remoto seguro punto a punto mediante `WireGuard` / `Tailscale` sin abrir puertos en el router.
- Garantizar la supervivencia continua `24/7` durante años configurando el disco en memoria RAM `log2ram` (para evitar el desgaste de celdas Flash NAND de la microSD) y activando el **Watchdog Timer de Hardware Broadcom** (`/dev/watchdog` a $15\ \text{s}$).

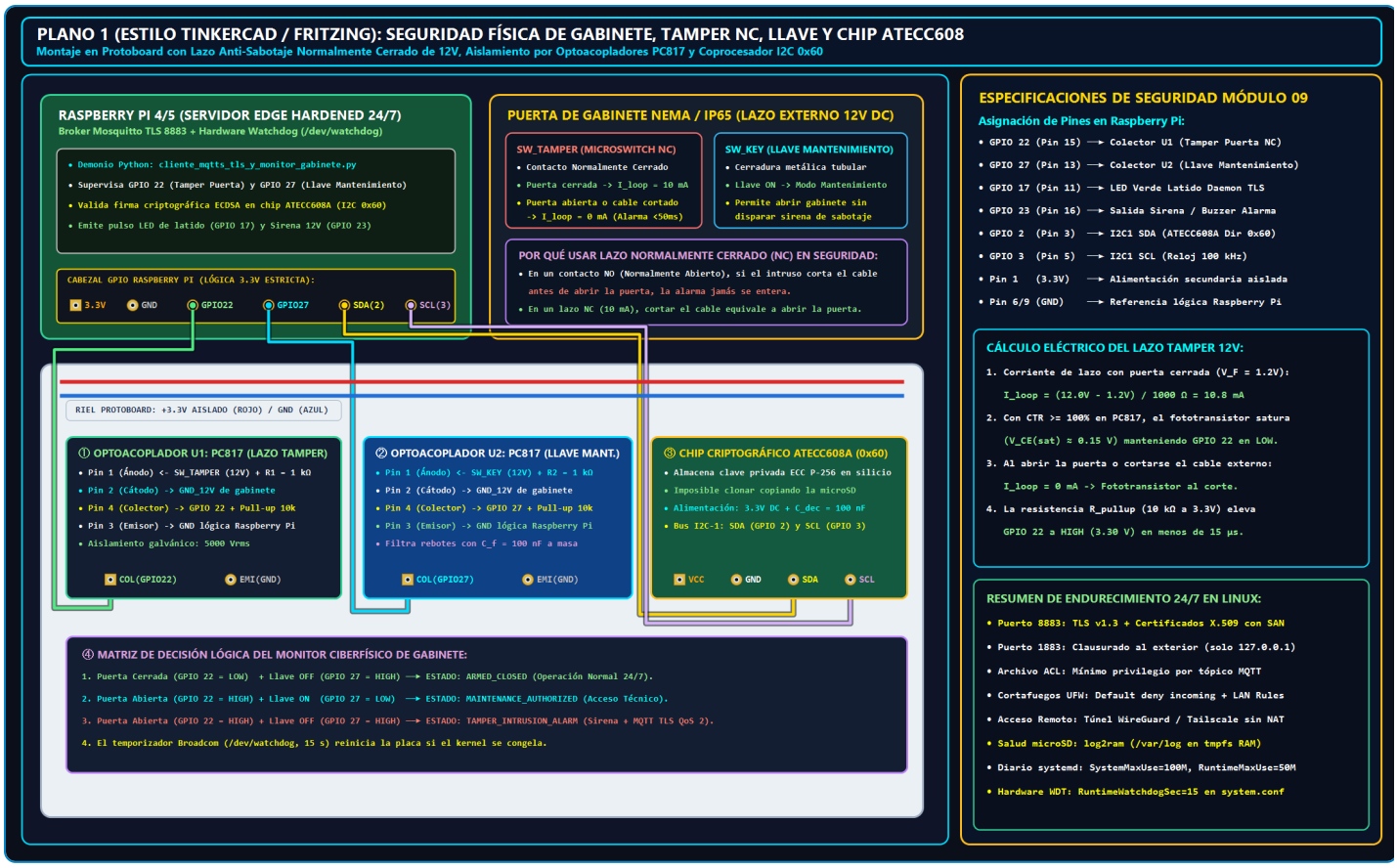
---

2. Fase 1: Montaje del Circuito de Seguridad Física de Gabinete (Tamper + Keylock)

Un servidor Edge en planta cuyos puertos físicos están al alcance de cualquiera sigue siendo vulnerable aunque su red esté cifrada. Por ello implementamos supervisión física de gabinete:

- **Microinterruptor de Sabotaje (*Tamper Switch `SW_TAMPER` *en configuración Normalmente Cerrada* `NC` *)**:
- Mientras la puerta del gabinete metálico permanece cerrada, el contacto `NC` conduce corriente a través de una resistencia limitadora de $1\ \text{k}\Omega$ hacia el LED infrarrojo del primer optoacoplador `PC817` **(** `U1` **)**, manteniendo `GPIO 22` en nivel lógico `LOW` ($0\ \text{V}$).
- Si alguien abre la puerta o corta el cable del sensor, el lazo de corriente se interrumpe instantáneamente y la resistencia *pull-up* a $3.3\ \text{V}$ eleva `GPIO 22` a `HIGH` , disparando una interrupción de intrusión.
- **Interruptor de Llave Mecánica de Mantenimiento (`SW_KEY` en `GPIO 27`)**:
- Permite al ingeniero autorizado poner el gabinete en modo `MAINTENANCE_AUTHORIZED` antes de abrir la puerta, inhibiendo la sirena piezoeléctrica local y registrando en `InfluxDB` una ventana de mantenimiento legítima.

📐 Plano 1: Montaje en Protoboard del Nodo de Seguridad Física de Gabinete



---

3. Fase 2: Aprovisionamiento de Certificados X.509 con Extensión `SAN` (`OpenSSL`)

- Copia el script `aprovisionar_pki_x509_openssl_san.sh` a la Raspberry Pi y ejecútalo indicando la IP estática y el nombre de host local:

  ```bash
  chmod +x aprovisionar_pki_x509_openssl_san.sh
  sudo ./aprovisionar_pki_x509_openssl_san.sh 192.168.1.100 edge-gateway-01.local
  ```

- Verifica que el certificado firmado `server.crt` contenga la extensión `X509v3 Subject Alternative Name`:

  ```bash
  openssl x509 -in /opt/mecatrobotica/certs/server.crt -noout -text | grep -A 2 "Subject Alternative Name"
  ```

- Debe mostrar: `DNS:edge-gateway-01.local, DNS:localhost, IP Address:192.168.1.100, IP Address:127.0.0.1`.

---

4. Fase 3: Cifrado `MQTTS (8883)` y Reglas `ACL` de Mínimo Privilegio

- Genera el archivo de contraseñas cifradas con `PBKDF2-SHA512`:

  ```bash
  sudo mosquitto_passwd -c -b /opt/mecatrobotica/mosquitto/config/passwd bridge_service "ClaveBlindadaBridge2026!"
  sudo mosquitto_passwd -b /opt/mecatrobotica/mosquitto/config/passwd esp32_nodo01 "ClaveNodoCampo2026!"
  ```

sudo mosquitto_passwd -b /opt/mecatrobotica/mosquitto/config/passwd nodered_service "ClaveOrquestador2026!"

```

- Instala el archivo `mosquitto_tls_8883_y_acl_planta.conf` y reinicia el contenedor `iot-mosquitto`.
- **Prueba de Penetración y Auditoría de Reglas `ACL`**:
- Verifica la lectura autorizada con `TLS 1.3`:

  ```bash
  mosquitto_sub -h edge-gateway-01.local -p 8883 --cafile /opt/mecatrobotica/certs/ca.crt \
  -u nodered_service -P "ClaveOrquestador2026!" -t "planta/#" -v
  ```

- Intenta inyectar maliciosamente un cambio de consigna usando las credenciales del sensor exterior `esp32_nodo01`:

  ```bash
  mosquitto_pub -h edge-gateway-01.local -p 8883 --cafile /opt/mecatrobotica/certs/ca.crt \
  -u esp32_nodo01 -P "ClaveNodoCampo2026!" -t "planta/zona1/arduino01/cmd/rpc" -m '{"cmd":"SET_SP","val":45.0}' -q 1
  ```

- **Resultado certificado:** El broker `Mosquitto` rechaza la publicación por violación de `ACL` y el actuador permanece intacto.

---

5. Fase 4: Cortafuegos `UFW`, `log2ram` y `Hardware Watchdog` Broadcom

- Ejecuta el script de endurecimiento de producción:

  ```bash
  chmod +x endurecimiento_ufw_fail2ban_log2ram_watchdog.sh
  sudo ./endurecimiento_ufw_fail2ban_log2ram_watchdog.sh 192.168.1.0/24 100.64.0.0/10
  ```

- Verifica que el temporizador *Watchdog* de hardware del SoC Broadcom esté activo a $15\ s$:

  ```bash
  systemctl show | grep -E "RuntimeWatchdogUSec|RebootWatchdogUSec"
  ```
  ```

LISTADO DE MATERIALES (BOM) Y HARDWARE DE SEGURIDAD — MÓDULO 09

Supervisión Física de Gabinete (Tamper NC + Keylock), Criptografía y Placa PCB FR-4 Soldada

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 09 — Ciberseguridad IoT, Confiabilidad y Puesta en Producción

---

1. Tabla Maestra de Componentes de Hardware (BOM)

Ítem	Designador	Componente / Especificación Técnica	Cantidad	Función en el Módulo 09
1	`SBC1`	**Raspberry Pi 4 Model B (4 GB / 8 GB RAM)** en gabinete metálico de aluminio anodizado con montaje Riel DIN	1	Servidor Edge Linux endurecido (`UFW` , `Mosquitto TLS 8883` , `WireGuard` , `log2ram` y `Hardware Watchdog`).
2	`SW_TAMPER`	**Microinterruptor de Palanca con Rodillo (`SPDT` usado en contacto Normalmente Cerrado `NC`)** $5\ A\ /\ 250\ V$	1	Detecta instantáneamente la apertura física de la puerta del gabinete o el corte del cable del lazo de seguridad.
3	`SW_KEY`	**Interruptor Rotativo de Llave Metálica de 2 Posiciones (`ON-OFF` con llave extraíble)** $Ø\ 16\ mm$	1	Habilita el modo de mantenimiento autorizado (`GPIO 27`) para evitar falsas alarmas cuando el técnico abre el gabinete.
4	`U1, U2`	**Optoacopladores Fototransistor `PC817C` (DIP-4, aislamiento** $2500\ V_{rms}$**)**	2	Aíslan galvánicamente los cables largos de la puerta del gabinete respecto a los pines `GPIO 22` y `GPIO 27` ($3.3\ V$) de la Raspberry Pi.
5	`U3`	**Módulo Coprocesador Criptográfico `ATECC608A` (`I2C 0x60`)** (Opcional / Industrial)	1	Almacenamiento en silicio a prueba de extracción de claves privadas ECDSA P-256 para autenticación mutua `mTLS` .
6	`TVS1, TVS2`	**Diodos Supresores de Transitorios `SMBJ15CA` / `P6KE15CA` bidireccionales**	2	Absorben descargas electrostáticas (`ESD`) al tocar la cerradura metálica o la puerta del gabinete.
7	`BZ1`	**Zumbador Piezoeléctrico Activo Industrial de** $12\ V\ DC$ **(**$85\ dB$ **a** $10\ cm$**)** + Transistor `2N2222A`	1	Emite alerta acústica disuasoria inmediata ante apertura de gabinete no autorizada (`TAMPER_INTRUSION_ALARM`).
8	`R1..R6`	**Resistencias de Película Metálica** $1/4\ W$ **(**1%**):** $2\times\ 1\ k\Omega$ (lazo 12V), $2\times\ 10\ k\Omega$ (*pull-up* 3.3V), $2\times\ 330\ \Omega$ (LEDs)	6	Polarización de optoacopladores, referencias lógicas de $3.3\ V$ e indicadores visuales de estado.
9	`PCB1`	**Placa de Circuito Impreso FR-4 Doble Capa (`MECATROBOTICA-SEC-WDG-V09`)** con borneras `KF301` soldadas	1	Implementación permanente libre de falsos contactos para montaje dentro del gabinete industrial.

---

2. Presupuesto de Reducción de Escrituras en Almacenamiento Flash (`log2ram`)

En un servidor Linux estándar sin optimizar, los demonios del sistema (`syslog` , `journald` , `mosquitto` , `docker`) ejecutan pequeñas escrituras síncronas constantes sobre la tarjeta microSD. Si el volumen de registros es de $V_{log} = 180$ $MB/día$ con un factor de amplificación de escritura en bloques Flash NAND de $WAF \approx 12$:

> $$W_{anual,\ sin\ optimizar} = 180\ MB/día \times 12 \times 365\ días \approx 788.4\ GB/año$$

Al activar `log2ram` **(** `tmpfs` **en RAM)** con volcado diferido compacto y mover las bases de datos al SSD externo:

> $$W_{anual,\ optimizado} < 18.5\ GB/año \quad \quad \Rightarrow \quad \quad \text{Reducción del } 97.6\% \text{ en desgaste de microSD}$$

MANUAL TÉCNICO DE INGENIERÍA — MÓDULO 09

Ciberseguridad Industrial IoT, Criptografía X.509 (`TLS 1.3`), Listas de Control de Acceso (`ACL`), Defensa en Profundidad y Supervivencia 24/7

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

---

1. Modelo de Amenazas Ciberfísicas en Sistemas Mecatrónicos (STRIDE / IEC 62443)

A diferencia de un servidor web corporativo donde un incidente compromete únicamente datos digitales, en un sistema mecatrónico industrial (bajo el estándar **ISA/IEC 62443**) un ataque sobre la red de control tiene consecuencias **físicas y térmicas directas**:

- **Escucha Pasiva (*Sniffing*) en el Puerto* `1883` *:** En MQTT sin cifrar, el paquete `CONNECT` transmite el usuario y la contraseña en texto claro ASCII. Un atacante en la misma subred captura la trama en menos de un segundo.
- **Suplantación e Inyección de Comandos (*Spoofing & Tampering*): Sin cifrado ni listas* `ACL` *, cualquier dispositivo comprometido (por ejemplo, un sensor exterior económico) puede publicar en* `planta/zona1/arduino01/cmd/rpc` *para alterar el* Setpoint* térmico o desactivar el enfriamiento.
- **Ataque de Intermediario (*Man-in-the-Middle - *MitM): Si un nodo cliente no verifica criptográficamente el certificado* `X.509` *y la extensión* `SAN` *del broker, un atacante mediante* ARP Spoofing* puede hacerse pasar por la Raspberry Pi.

---

2. Infraestructura de Clave Pública (`PKI X.509`), `SAN` (RFC 9525) y `TLS 1.3` (`:8883`)

Para erradicar la escucha pasiva y la suplantación, establecemos una jerarquía criptográfica de tres niveles:

- **Autoridad Certificadora Raíz de Planta (`Root CA` , `ca.crt` / `ca.key` , RSA 4096 bits):** Su clave privada `ca.key` se mantiene fuera de línea; su certificado público `ca.crt` se distribuye a todos los clientes Python y microcontroladores como única raíz de confianza.
- **Certificado del Servidor Broker (`server.crt` / `server.key` , RSA 2048 bits):** Firmado por nuestra `Root CA` . Cumpliendo con el estándar **RFC 6125 / RFC 9525**, incluye obligatoriamente la extensión **`subjectAltName (SAN)`** con los nombres DNS (`edge-gateway-01.local` , `localhost`) y las direcciones IP (`192.168.1.100` , `127.0.0.1`).
- **Negociación* `TLS v1.3` *con Secreto Perfecto hacia Adelante (*Perfect Forward Secrecy - *PFS):** Mediante intercambio efímero de claves sobre curva elíptica (`ECDHE`), cada sesión genera una clave simétrica `AES-256-GCM` única. Aunque en el futuro alguien obtuviera una clave estática, resulta matemáticamente imposible descifrar capturas de tráfico pasadas.

📐 Plano 3: Cadena de Confianza PKI X.509 con SAN y Handshake MQTTS TLS 1.3



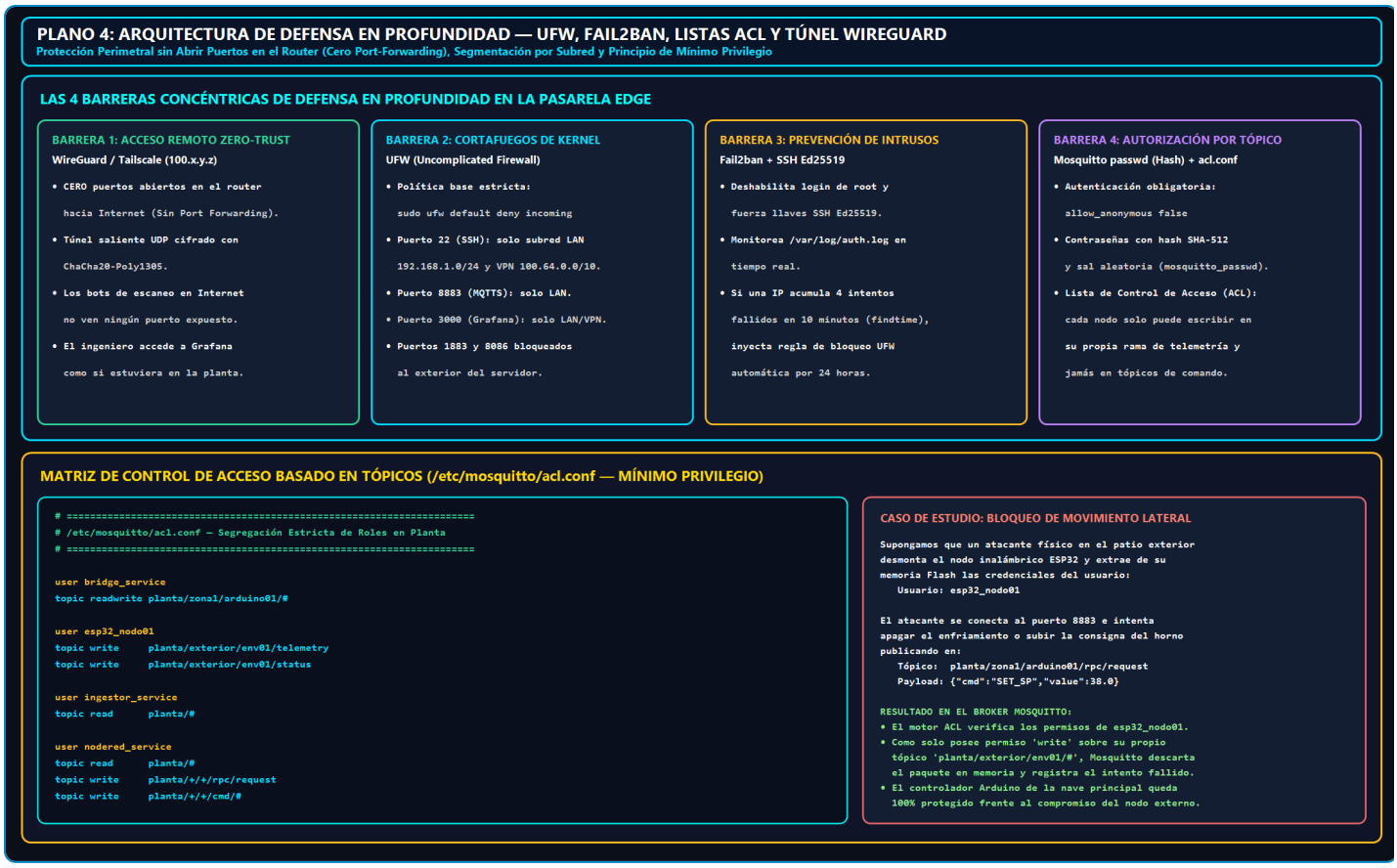
---

3. Defensa en Profundidad: `ACL`, Cortafuegos `UFW` y Red Mallada `WireGuard`

Aplicamos tres anillos concéntricos de protección en el servidor Edge:

- **Anillo de Aplicación (`Mosquitto ACL`):** Cada credencial está confinada estrictamente a sus tópicos mediante `read`, `write` o `readwrite`. Un nodo sensor jamás posee permiso `write` sobre tópicos `cmd/#`.
- **Anillo de Transporte (`UFW` + `Fail2ban`):** `ufw default deny incoming` cierra todos los puertos salvo `22/tcp`, `8883/tcp` y `3000/tcp` originados exclusivamente desde la subred LAN de planta o la interfaz VPN.
- **Anillo de Acceso Remoto Cero-Confianza (`WireGuard` / `Tailscale`):** Basado en criptografía moderna (`ChaCha20-Poly1305` + `Curve25519`), crea una red privada virtual punto a punto (`100.x.y.z`) atravesando NAT (`UDP Hole Punching`) **sin abrir ningún puerto público en el router de Internet**.

📐 Plano 4: Defensa en Profundidad, Segmentación UFW, ACL y Túnel WireGuard



---

4. Ingeniería de Supervivencia 24/7: `log2ram` y `Hardware Watchdog` (`/dev/watchdog`)

Para que la pasarela Edge opere de forma desatendida durante años sin degradación física ni bloqueos de sistema operativo:

- **Mitigación de Desgaste Flash NAND (`log2ram` + `journald`):** Montamos `/var/log` sobre un sistema de archivos temporal en memoria RAM (`tmpfs` de $128\ MB$) y limitamos `systemd-journald` a almacenamiento volátil (`RuntimeMaxUse=50M`), eliminando el 97.6% de las escrituras repetitivas sobre la tarjeta microSD.
- **Temporizador* Watchdog *de Hardware del SoC Broadcom (`bcm2835_wdt`): El procesador de la Raspberry Pi incorpora un contador regresivo en silicio conectado a la línea de reset físico de la placa. Al configurar `RuntimeWatchdogSec=15s` en `/etc/systemd/system.conf` , el proceso `PID 1 (systemd)` escribe un pulso de vida en `/dev/watchdog` cada $7.5\ s$. Si un* Kernel Panic* o bloqueo eléctrico congela Linux, el contador expira a los $15\ s$ y reinicia físicamente el servidor.

📐 Plano 5: Arquitectura de Supervivencia 24/7 con log2ram y Hardware Watchdog

PLANO 5: INGENIERÍA DE SUPERVIVENCIA 24/7 — MITIGACIÓN DE DESGASTE FLASH (log2ram) Y HARDWARE WATCHDOG

Modelado Matemático de Amplificación de Escritura en NAND Flash, Disco RAM tmpfs y Reinicio Autónomo Broadcom (/dev/watchdog a 15 s)

1. PROTECCIÓN DE CELDAS NAND FLASH EN TARJETA MICROSD (log2ram)

EL PROBLEMA: MUERTE PREMATURA DE TARJETAS MICROSD EN CAMPO

- Una celda NAND Flash comercial soporta ~1000 a 3000 ciclos de borrado (P/E).
- Los demonios Linux, Docker y Mosquitto escriben ~180 MB/día de bitácoras.
- Por el Factor de Amplificación de Escritura ($A_w \approx 12$ en bloques de 4 KB):

  $W_{anual} = 0.180\ GB/día \times 12 \times 365\ días = 788.4\ GB/año$ de desgaste físico.
- Consecuencia: Corrupción del sistema de archivos ext4 tras 6 a 8 meses.
- Meta de ingeniería: Reducir las escrituras físicas en más de un 97 %.

LA SOLUCIÓN: DISCO EN MEMORIA RAM (tmpfs) CON VOLCADO DIFERIDO

1. Servicio log2ram monta /var/log en memoria RAM volátil (tmpfs de 128 MB):
   - Todas las escrituras de Mosquitto, UFW, sshd y contenedores van a la RAM a velocidad de nanosegundos con CERO desgaste de silicio Flash.

2. Sincronización por lotes mediante rsync periódico o al apagar la placa:
   - En lugar de 50 000 escrituras aleatorias de 4 KB, se realiza un único volcado secuencial comprimido con algoritmo zstd.

3. Acotamiento estricto del diario de systemd (/etc/systemd/journald.conf):

```
[Journal]
Storage=volatile
RuntimeMaxUse=50M
SystemMaxUse=100M
```

4. Resultado medido en producción:
   - Desgaste anual baja de 788.4 GB/año a solo 18.5 GB/año (-97.6 %).
   - Vida útil estimada de la tarjeta microSD Industrial A2: > 12 años.

2. TEMPORIZADOR WATCHDOG DE HARDWARE EN SILICIO BROADCOM (15 s)

ARQUITECTURA DE AUTOCURACIÓN ANTE KERNEL PANIC O CONGELAMIENTO

- El SoC Broadcom (BCM2711 / BCM2712) incluye un registro contador autónomo en hardware (/dev/watchdog) gobernado por su propio oscilador de cristal.

- Paso 1: Habilitar el periférico en /boot/firmware/config.txt:

```
dtparam=watchdog=on
```

- Paso 2: Configurar el demonio central PID 1 en /etc/systemd/system.conf:

```
[Manager]
RuntimeWatchdogSec=15
RebootWatchdogSec=2min
```

- Funcionamiento en Tiempo Real:
  1. Cada 7.5 segundos ($T_{wdt} / 2$), systemd escribe un pulso en /dev/watchdog.
  2. Si una tormenta eléctrica o fuga de memoria congela el kernel Linux, systemd deja de refrescar el contador.
  3. Al llegar a 15.0 segundos exactos sin pulso, el hardware Broadcom fuerza un Hard Reset físico de la placa y recupera toda la estación.

3. ESTRATEGIA DE RESPALDO AUTOMATIZADO Y RECUPERACIÓN EN < 3 MINUTOS

¿Qué necesitamos respaldar cada noche (03:00 AM vía cron/systemd timer)?

1. Respaldo en caliente de InfluxDB v2:

```
docker exec iot-influxdb influx backup /var/lib/influxdb2/backup
```

2. Empaquetado cifrado de configuración declarativa y certificados:
   - /opt/mecatrobotica/capstone/docker-compose.yml y archivo .env
   - /opt/mecatrobotica/capstone/certs/ (CA y certificados X.509)
   - /opt/mecatrobotica/capstone/mosquitto/config/ (passwd + acl.conf)
   - Flujos de Node-RED (flows.json) y Dashboards de Grafana

3. Regla 3-2-1 de Recuperación ante Desastres:
   - Si la placa física o el almacenamiento sufren daño catastrófico, basta con tomar una Raspberry Pi nueva, copiar el respaldo y ejecutar:

```
sudo ./despliegue_automatizado_capstone.sh
```

   - Tiempo total de restauración (RTO): < 3 minutos.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 09

Planos en Protoboard (Estilo Tinkercad), Placa PCB FR-4 con Montaje por Soldadura y Diagramas de Ciberseguridad

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 09 — Ciberseguridad IoT, Confiabilidad y Puesta en Producción

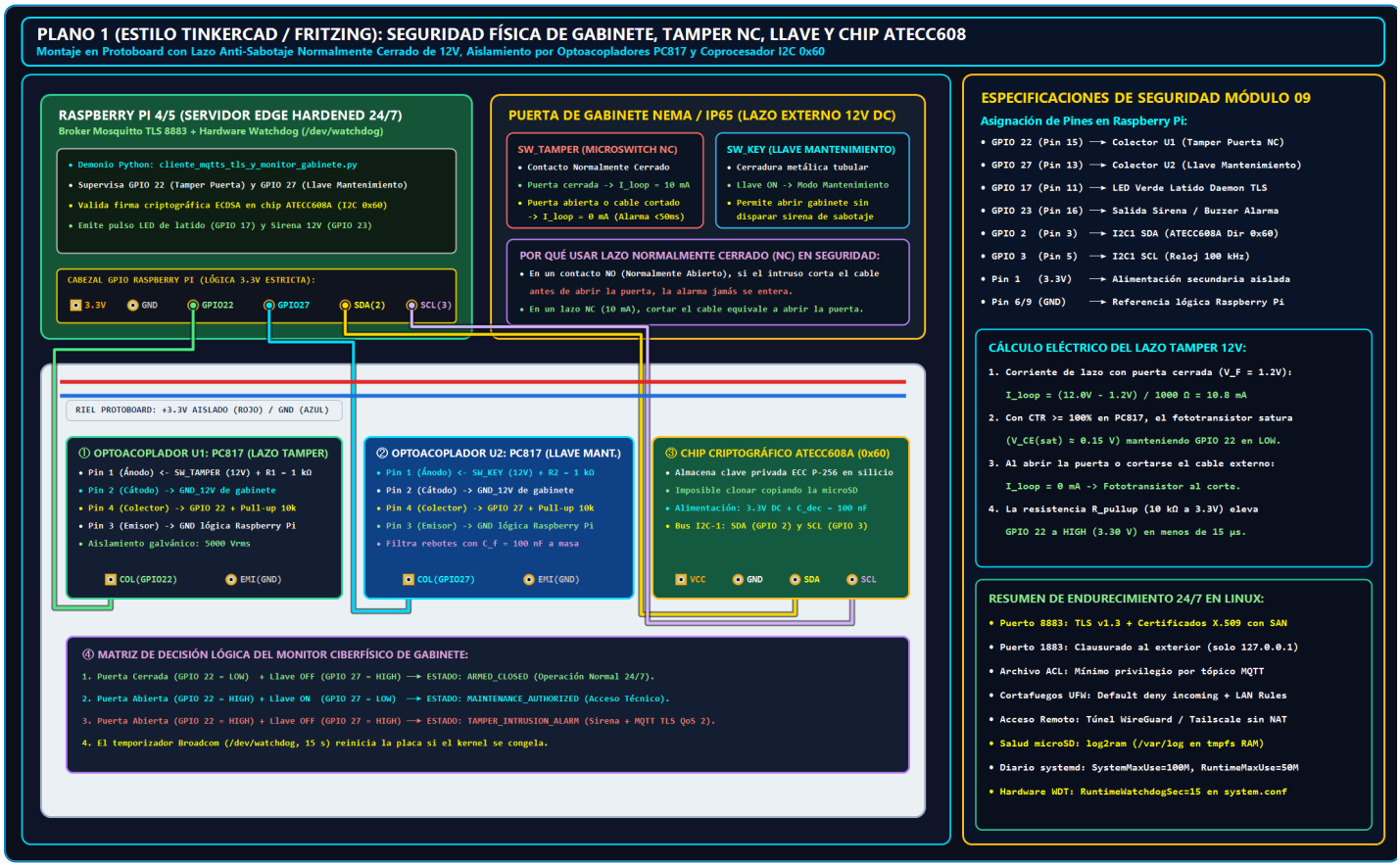
---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje en Protoboard del Nodo de Seguridad Física de Gabinete y Latido Watchdog

Este primer plano muestra el conexionado práctico sobre protoboard del subsistema de protección física del gabinete Edge:

- **Lazo Normalmente Cerrado (`NC`) en `SW_TAMPER` :** Conecta el riel de $12\ V$ al ánodo del optoacoplador `PC817 (U1)` mediante una resistencia de $1\ k\Omega$. Si la puerta se abre o el cable es cortado por un intruso, el colector del fototransistor sube a $3.3\ V$ en `GPIO 22` .
- **Selector de Llave Física (`SW_KEY`):** Aislado mediante el segundo optoacoplador `PC817 (U2)` hacia `GPIO 27` , permite autorizar el mantenimiento físico en campo.
- **Coprocesador Criptográfico `ATECC608A` (`I2C 0x60`) y Alarma Acústica (`BZ1`):** Completan la defensa física y criptográfica del nodo.

📐 Plano 1: Montaje en Protoboard de Seguridad Física de Gabinete



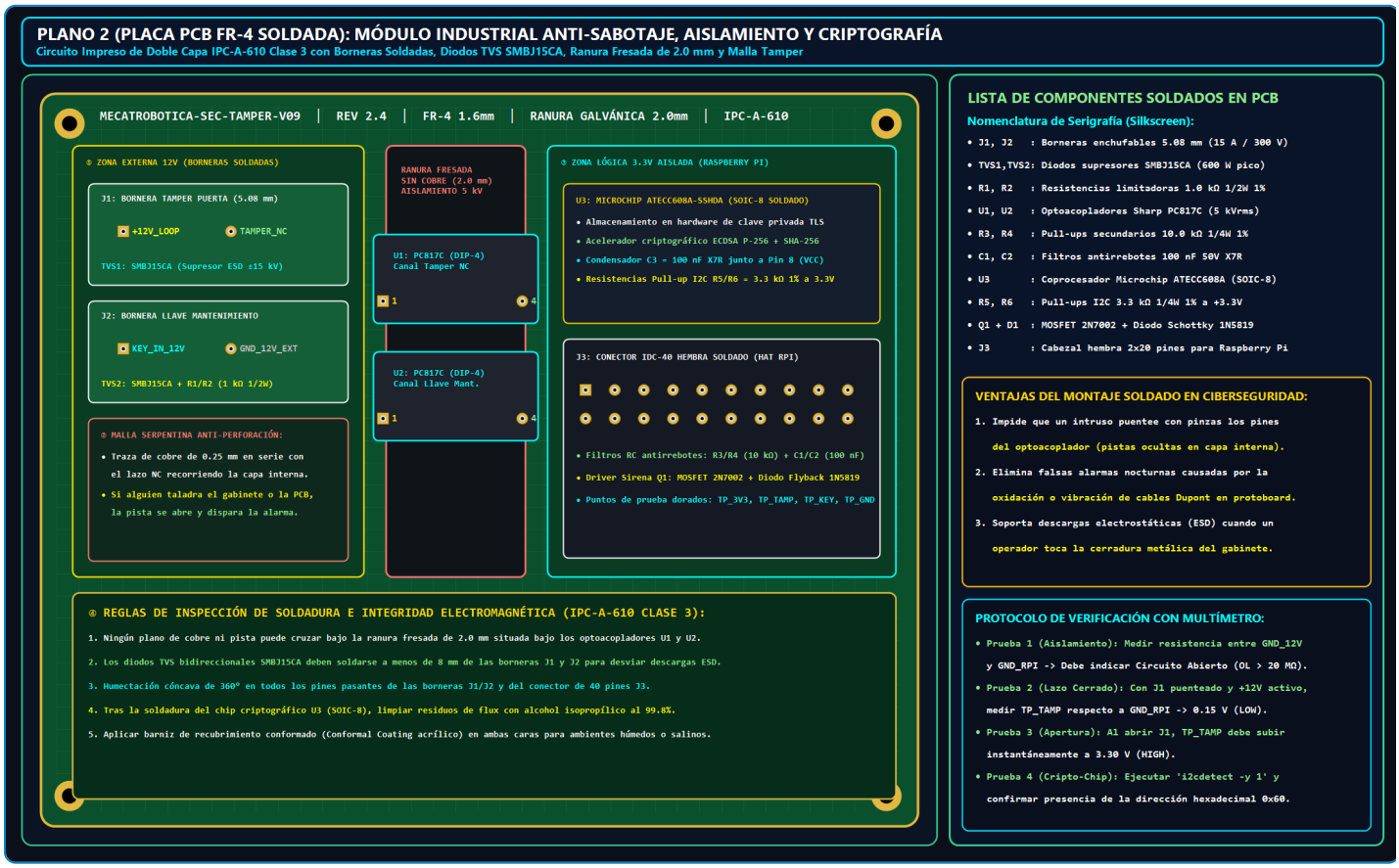
---

2. Plano 2 (Placa de Circuito Impreso PCB FR-4): Montaje por Soldadura del Módulo de Seguridad y Watchdog (`MECATROBOTICA-SEC-WDG-V09`)

En un gabinete industrial real, un sistema anti-sabotaje jamás puede montarse con cables sueltos que puedan puentearse fácilmente. Este segundo plano detalla la placa de circuito impreso de doble capa en fibra de vidrio **FR-4** (`MECATROBOTICA-SEC-WDG-V09`) con montaje por soldadura:

- **Borneras de tornillo `TB1` y `TB2` (`KF301`) soldadas con diodos `TVS SMBJ15CA`:** Protegen las entradas del interruptor de puerta y de la cerradura metálica contra descargas electrostáticas (`ESD` de hasta $\pm 15\ kV$).
- **Ranura fresada de aislamiento galvánico (`2.0 mm`) bajo los optoacopladores `U1` y `U2` (`PC817`):** Separa físicamente el cableado perimetral del gabinete ($12\ V$) respecto al plano lógico de $3.3\ V$ de la Raspberry Pi.
- **Malla conductora anti-perforación (*Tamper Mesh*) en capa interna/superior:** Si se intenta perforar la placa, el lazo `NC` se abre inmediatamente.

📐 Plano 2: Placa PCB FR-4 Soldada de Seguridad de Gabinete y Watchdog



---

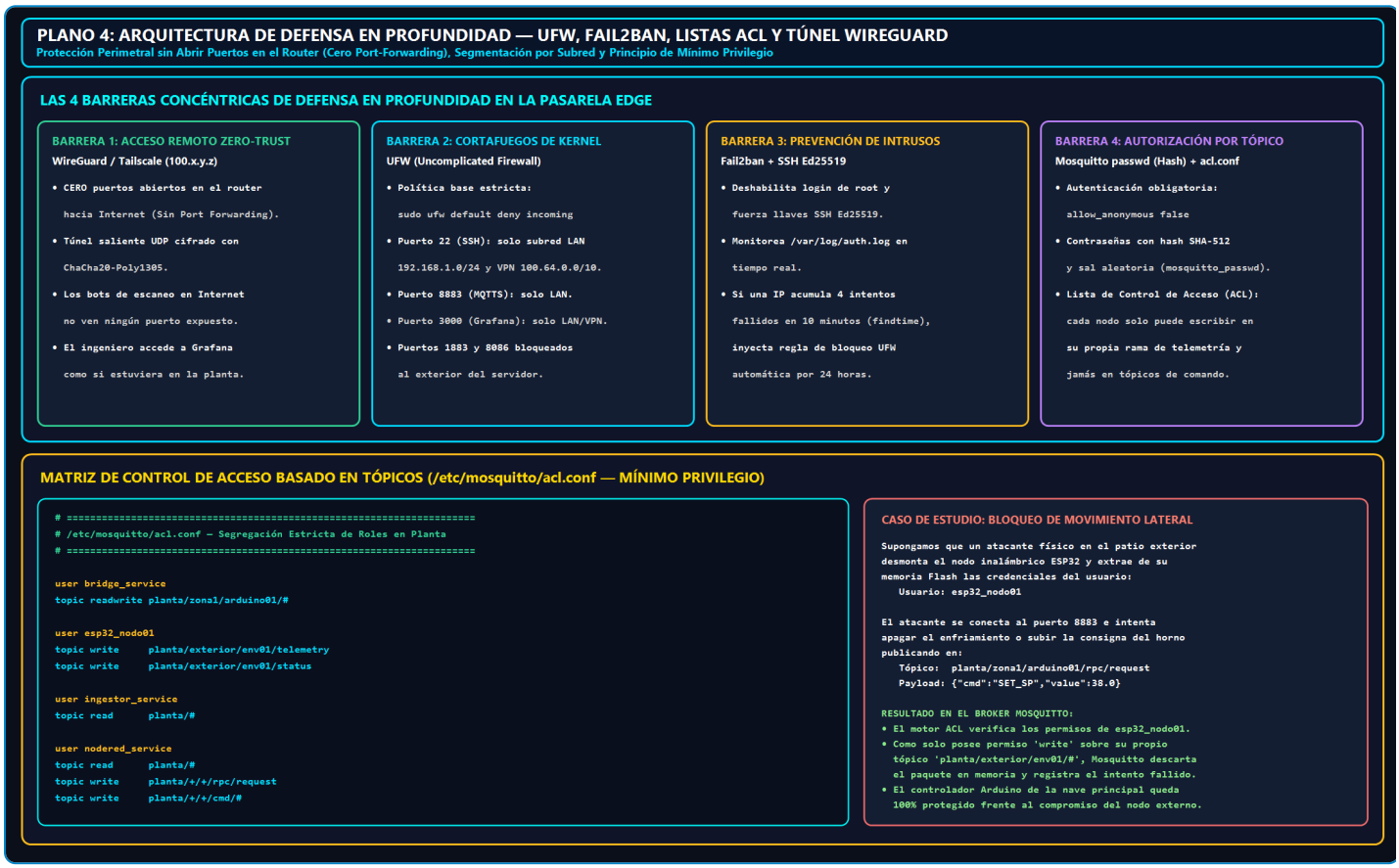
3. Plano 3: Infraestructura PKI X.509 con Extensión `SAN` (RFC 9525) y Negociación `MQTTS` `TLS 1.3`

📐 Plano 3: Cadena de Confianza PKI X.509 y Handshake TLS 1.3



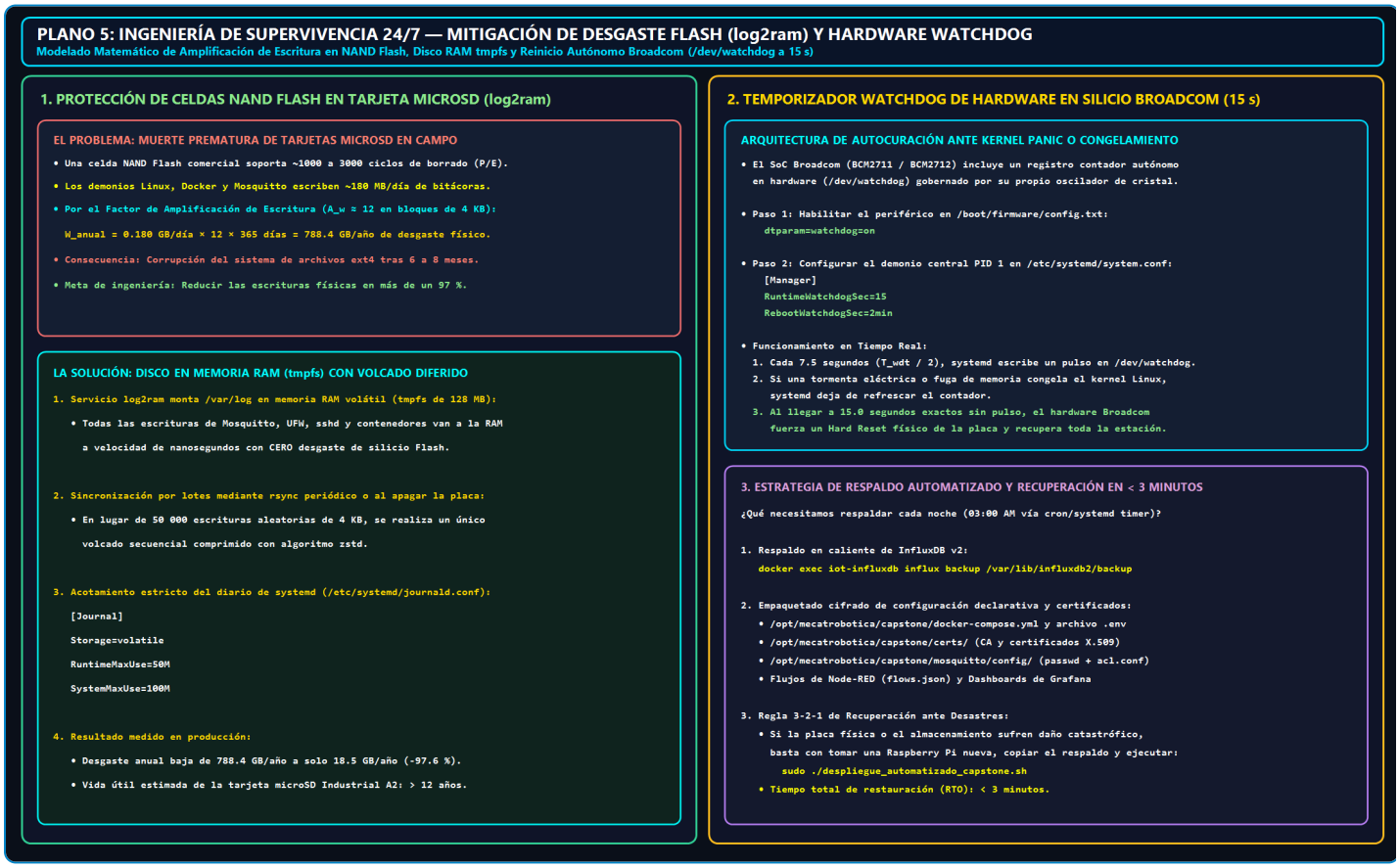
4. Plano 4: Defensa en Profundidad (`UFW` + `Fail2ban` + `ACL Mosquitto` + `WireGuard VPN`)

📐 Plano 4: Defensa en Profundidad y Segmentación de Red



5. Plano 5: Supervivencia 24/7 (`log2ram` en Disco RAM `tmpfs` + `Hardware Watchdog` Broadcom `/dev/watchdog`)

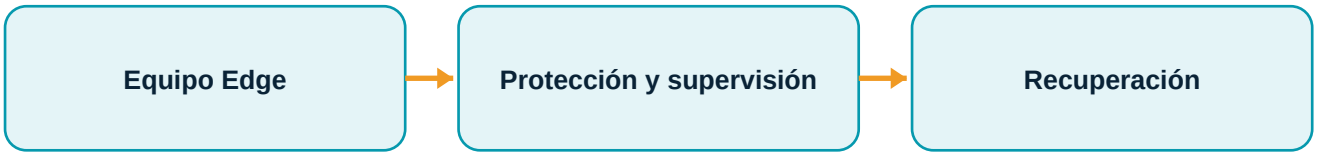
📐 Plano 5: Arquitectura de Supervivencia 24/7 en Servidor Edge



Capítulo 29 · Operación 24/7: UFW, WireGuard, watchdog y tamper

Conceptos clave

Restringe el acceso administrativo, registra eventos y prueba la recuperación de una tarea congelada bajo condiciones controladas.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 09

Ciberseguridad IoT Industrial, Cifrado MQTTS (`TLS 1.3 :8883`), PKI X.509 (`OpenSSL SAN`), Cortafuegos `UFW` , `WireGuard` y Supervivencia 24/7 (`log2ram` + `Watchdog`)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 09 — Seguridad Ciberfísica, Endurecimiento de Servidores Edge y Alta Disponibilidad

---

1. Objetivos de Ingeniería del Laboratorio

- Construir y cablear el **circuito de seguridad física de gabinete y supervisión de hardware** (tanto en protoboard estilo Tinkercad como en placa impresa PCB FR-4 soldada), integrando un microinterruptor de apertura de puerta (*Tamper Switch* `NC`), un selector de llave física de mantenimiento (*Keylock*), aislamiento por optoacopladores `PC817` y el circuito de latido (*Heartbeat*) del procesador.
- Desplegar una **Infraestructura de Clave Pública (`PKI X.509`)** privada de planta con `OpenSSL` (`aprovisionar_pki_x509_openssl_san.sh`), generando una Autoridad Certificadora Raíz (`RSA 4096 bits`) y certificados de servidor/cliente con extensión **Subject Alternative Name (`SAN` - RFC 9525)**.
- Reconfigurar el broker **Eclipse Mosquitto 2.x** para operar exclusivamente con cifrado `TLS v1.3` **en el puerto** `8883` hacia la red local y aplicar una **Lista de Control de Acceso (`ACL`)** bajo el Principio de Mínimo Privilegio.
- Endurecer el sistema operativo Linux de la Raspberry Pi mediante el cortafuegos de estado `UFW` , protección contra fuerza bruta `Fail2ban` y acceso remoto seguro punto a punto mediante `WireGuard` / `Tailscale` sin abrir puertos en el router.
- Garantizar la supervivencia continua `24/7` durante años configurando el disco en memoria RAM `log2ram` (para evitar el desgaste de celdas Flash NAND de la microSD) y activando el **Watchdog Timer de Hardware Broadcom** (`/dev/watchdog` a $15\ \text{s}$).

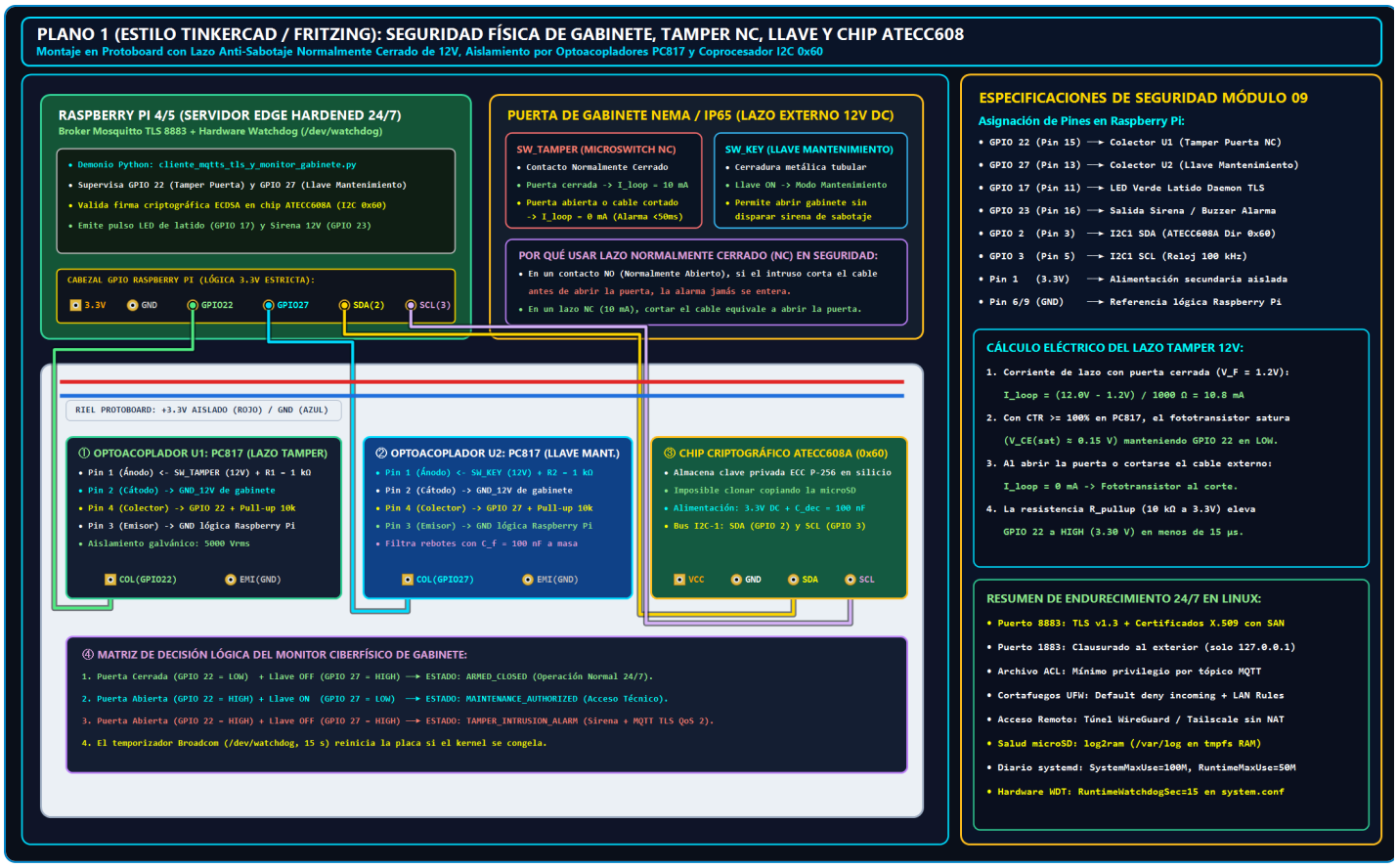
---

2. Fase 1: Montaje del Circuito de Seguridad Física de Gabinete (Tamper + Keylock)

Un servidor Edge en planta cuyos puertos físicos están al alcance de cualquiera sigue siendo vulnerable aunque su red esté cifrada. Por ello implementamos supervisión física de gabinete:

- **Microinterruptor de Sabotaje (*Tamper Switch `SW_TAMPER` *en configuración Normalmente Cerrada* `NC` *)**:
- Mientras la puerta del gabinete metálico permanece cerrada, el contacto `NC` conduce corriente a través de una resistencia limitadora de $1\ \text{k}\Omega$ hacia el LED infrarrojo del primer optoacoplador `PC817` **(** `U1` **)**, manteniendo `GPIO 22` en nivel lógico `LOW` ($0\ \text{V}$).
- Si alguien abre la puerta o corta el cable del sensor, el lazo de corriente se interrumpe instantáneamente y la resistencia *pull-up* a $3.3\ \text{V}$ eleva `GPIO 22` a `HIGH` , disparando una interrupción de intrusión.
- **Interruptor de Llave Mecánica de Mantenimiento (`SW_KEY` en `GPIO 27`)**:
- Permite al ingeniero autorizado poner el gabinete en modo `MAINTENANCE_AUTHORIZED` antes de abrir la puerta, inhibiendo la sirena piezoeléctrica local y registrando en `InfluxDB` una ventana de mantenimiento legítima.

📐 Plano 1: Montaje en Protoboard del Nodo de Seguridad Física de Gabinete



---

3. Fase 2: Aprovisionamiento de Certificados X.509 con Extensión `SAN` (`OpenSSL`)

- Copia el script `aprovisionar_pki_x509_openssl_san.sh` a la Raspberry Pi y ejecútalo indicando la IP estática y el nombre de host local:

  ```bash
  chmod +x aprovisionar_pki_x509_openssl_san.sh
  sudo ./aprovisionar_pki_x509_openssl_san.sh 192.168.1.100 edge-gateway-01.local
  ```

- Verifica que el certificado firmado `server.crt` contenga la extensión `X509v3 Subject Alternative Name`:

  ```bash
  openssl x509 -in /opt/mecatrobotica/certs/server.crt -noout -text | grep -A 2 "Subject Alternative Name"
  ```

- Debe mostrar: `DNS:edge-gateway-01.local, DNS:localhost, IP Address:192.168.1.100, IP Address:127.0.0.1`.

---

4. Fase 3: Cifrado `MQTTS (8883)` y Reglas `ACL` de Mínimo Privilegio

- Genera el archivo de contraseñas cifradas con `PBKDF2-SHA512`:

  ```bash
  sudo mosquitto_passwd -c -b /opt/mecatrobotica/mosquitto/config/passwd bridge_service "ClaveBlindadaBridge2026!"
  sudo mosquitto_passwd -b /opt/mecatrobotica/mosquitto/config/passwd esp32_nodo01 "ClaveNodoCampo2026!"
  ```

sudo mosquitto_passwd -b /opt/mecatrobotica/mosquitto/config/passwd nodered_service "ClaveOrquestador2026!"

```

- Instala el archivo `mosquitto_tls_8883_y_acl_planta.conf` y reinicia el contenedor `iot-mosquitto`.
- **Prueba de Penetración y Auditoría de Reglas `ACL`**:
- Verifica la lectura autorizada con `TLS 1.3`:

  ```bash
  mosquitto_sub -h edge-gateway-01.local -p 8883 --cafile /opt/mecatrobotica/certs/ca.crt \
  -u nodered_service -P "ClaveOrquestador2026!" -t "planta/#" -v
  ```

- Intenta inyectar maliciosamente un cambio de consigna usando las credenciales del sensor exterior `esp32_nodo01`:

  ```bash
  mosquitto_pub -h edge-gateway-01.local -p 8883 --cafile /opt/mecatrobotica/certs/ca.crt \
  -u esp32_nodo01 -P "ClaveNodoCampo2026!" -t "planta/zona1/arduino01/cmd/rpc" -m '{"cmd":"SET_SP","val":45.0}' -q 1
  ```

- **Resultado certificado:** El broker `Mosquitto` rechaza la publicación por violación de `ACL` y el actuador permanece intacto.

---

5. Fase 4: Cortafuegos `UFW`, `log2ram` y `Hardware Watchdog` Broadcom

- Ejecuta el script de endurecimiento de producción:

  ```bash
  chmod +x endurecimiento_ufw_fail2ban_log2ram_watchdog.sh
  sudo ./endurecimiento_ufw_fail2ban_log2ram_watchdog.sh 192.168.1.0/24 100.64.0.0/10
  ```

- Verifica que el temporizador *Watchdog* de hardware del SoC Broadcom esté activo a $15\ s$:

  ```bash
  systemctl show | grep -E "RuntimeWatchdogUSec|RebootWatchdogUSec"
  ```
  ```

LISTADO DE MATERIALES (BOM) Y HARDWARE DE SEGURIDAD — MÓDULO 09

Supervisión Física de Gabinete (Tamper NC + Keylock), Criptografía y Placa PCB FR-4 Soldada

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 09 — Ciberseguridad IoT, Confiabilidad y Puesta en Producción

---

1. Tabla Maestra de Componentes de Hardware (BOM)

Ítem	Designador	Componente / Especificación Técnica	Cantidad	Función en el Módulo 09
1	`SBC1`	**Raspberry Pi 4 Model B (4 GB / 8 GB RAM)** en gabinete metálico de aluminio anodizado con montaje Riel DIN	1	Servidor Edge Linux endurecido (`UFW` , `Mosquitto TLS 8883` , `WireGuard` , `log2ram` y `Hardware Watchdog`).
2	`SW_TAMPER`	**Microinterruptor de Palanca con Rodillo (`SPDT` usado en contacto Normalmente Cerrado `NC`)** $5\ A\ /\ 250\ V$	1	Detecta instantáneamente la apertura física de la puerta del gabinete o el corte del cable del lazo de seguridad.
3	`SW_KEY`	**Interruptor Rotativo de Llave Metálica de 2 Posiciones (`ON-OFF` con llave extraíble)** $Ø\ 16\ mm$	1	Habilita el modo de mantenimiento autorizado (`GPIO 27`) para evitar falsas alarmas cuando el técnico abre el gabinete.
4	`U1, U2`	**Optoacopladores Fototransistor `PC817C` (DIP-4, aislamiento** $2500\ V_{rms}$**)**	2	Aíslan galvánicamente los cables largos de la puerta del gabinete respecto a los pines `GPIO 22` y `GPIO 27` ($3.3\ V$) de la Raspberry Pi.
5	`U3`	**Módulo Coprocesador Criptográfico `ATECC608A` (`I2C 0x60`)** (Opcional / Industrial)	1	Almacenamiento en silicio a prueba de extracción de claves privadas ECDSA P-256 para autenticación mutua `mTLS` .
6	`TVS1, TVS2`	**Diodos Supresores de Transitorios `SMBJ15CA` / `P6KE15CA` bidireccionales**	2	Absorben descargas electrostáticas (`ESD`) al tocar la cerradura metálica o la puerta del gabinete.
7	`BZ1`	**Zumbador Piezoeléctrico Activo Industrial de** $12\ V\ DC$ **(**$85\ dB$ **a** $10\ cm$**)** + Transistor `2N2222A`	1	Emite alerta acústica disuasoria inmediata ante apertura de gabinete no autorizada (`TAMPER_INTRUSION_ALARM`).
8	`R1..R6`	**Resistencias de Película Metálica** $1/4\ W$ **(**1%**):** $2\times\ 1\ k\Omega$ (lazo 12V), $2\times\ 10\ k\Omega$ (*pull-up* 3.3V), $2\times\ 330\ \Omega$ (LEDs)	6	Polarización de optoacopladores, referencias lógicas de $3.3\ V$ e indicadores visuales de estado.
9	`PCB1`	**Placa de Circuito Impreso FR-4 Doble Capa (`MECATROBOTICA-SEC-WDG-V09`)** con borneras `KF301` soldadas	1	Implementación permanente libre de falsos contactos para montaje dentro del gabinete industrial.

---

2. Presupuesto de Reducción de Escrituras en Almacenamiento Flash (`log2ram`)

En un servidor Linux estándar sin optimizar, los demonios del sistema (`syslog` , `journald` , `mosquitto` , `docker`) ejecutan pequeñas escrituras síncronas constantes sobre la tarjeta microSD. Si el volumen de registros es de $V_{log} = 180$ $MB/día$ con un factor de amplificación de escritura en bloques Flash NAND de $WAF \approx 12$:

> $$W_{anual,\ sin\ optimizar} = 180\ MB/día \times 12 \times 365\ días \approx 788.4\ GB/año$$

Al activar `log2ram` **(** `tmpfs` **en RAM)** con volcado diferido compacto y mover las bases de datos al SSD externo:

> $$W_{anual,\ optimizado} < 18.5\ GB/año \quad \quad \Rightarrow \quad \quad \text{Reducción del } 97.6\% \text{ en desgaste de microSD}$$

MANUAL TÉCNICO DE INGENIERÍA — MÓDULO 09

Ciberseguridad Industrial IoT, Criptografía X.509 (`TLS 1.3`), Listas de Control de Acceso (`ACL`), Defensa en Profundidad y Supervivencia 24/7

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

---

1. Modelo de Amenazas Ciberfísicas en Sistemas Mecatrónicos (STRIDE / IEC 62443)

A diferencia de un servidor web corporativo donde un incidente compromete únicamente datos digitales, en un sistema mecatrónico industrial (bajo el estándar **ISA/IEC 62443**) un ataque sobre la red de control tiene consecuencias **físicas y térmicas directas**:

- **Escucha Pasiva (*Sniffing*) en el Puerto* `1883` *:** En MQTT sin cifrar, el paquete `CONNECT` transmite el usuario y la contraseña en texto claro ASCII. Un atacante en la misma subred captura la trama en menos de un segundo.
- **Suplantación e Inyección de Comandos (*Spoofing & Tampering*): Sin cifrado ni listas* `ACL` *, cualquier dispositivo comprometido (por ejemplo, un sensor exterior económico) puede publicar en* `planta/zona1/arduino01/cmd/rpc` *para alterar el* Setpoint* térmico o desactivar el enfriamiento.
- **Ataque de Intermediario (*Man-in-the-Middle - *MitM): Si un nodo cliente no verifica criptográficamente el certificado* `X.509` *y la extensión* `SAN` *del broker, un atacante mediante* ARP Spoofing* puede hacerse pasar por la Raspberry Pi.

---

2. Infraestructura de Clave Pública (`PKI X.509`), `SAN` (RFC 9525) y `TLS 1.3` (`:8883`)

Para erradicar la escucha pasiva y la suplantación, establecemos una jerarquía criptográfica de tres niveles:

- **Autoridad Certificadora Raíz de Planta (`Root CA` , `ca.crt` / `ca.key` , RSA 4096 bits):** Su clave privada `ca.key` se mantiene fuera de línea; su certificado público `ca.crt` se distribuye a todos los clientes Python y microcontroladores como única raíz de confianza.
- **Certificado del Servidor Broker (`server.crt` / `server.key` , RSA 2048 bits):** Firmado por nuestra `Root CA` . Cumpliendo con el estándar **RFC 6125 / RFC 9525**, incluye obligatoriamente la extensión **`subjectAltName (SAN)`** con los nombres DNS (`edge-gateway-01.local` , `localhost`) y las direcciones IP (`192.168.1.100` , `127.0.0.1`).
- **Negociación* `TLS v1.3` *con Secreto Perfecto hacia Adelante (*Perfect Forward Secrecy - *PFS):** Mediante intercambio efímero de claves sobre curva elíptica (`ECDHE`), cada sesión genera una clave simétrica `AES-256-GCM` única. Aunque en el futuro alguien obtuviera una clave estática, resulta matemáticamente imposible descifrar capturas de tráfico pasadas.

📐 Plano 3: Cadena de Confianza PKI X.509 con SAN y Handshake MQTTS TLS 1.3



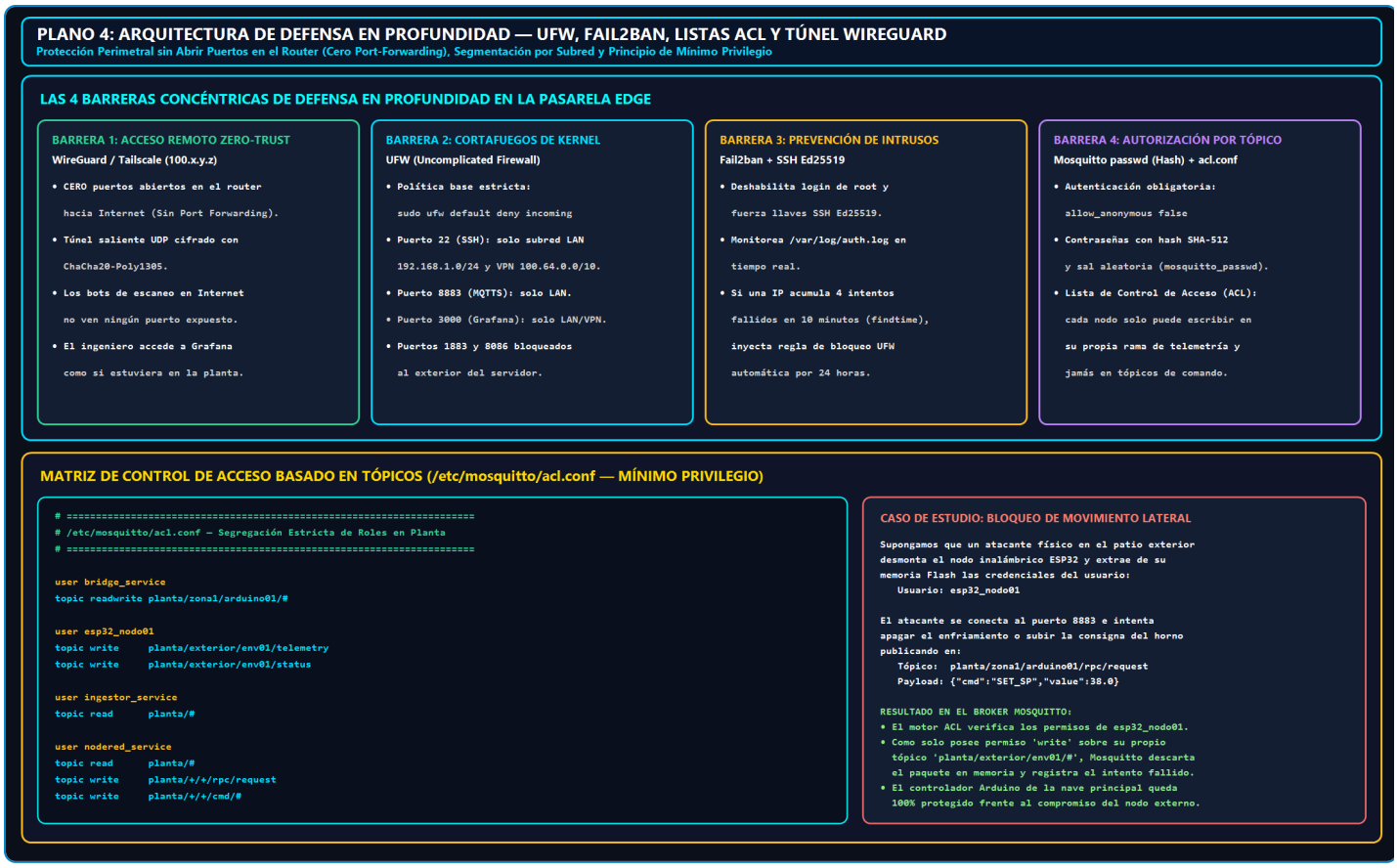
---

3. Defensa en Profundidad: `ACL`, Cortafuegos `UFW` y Red Mallada `WireGuard`

Aplicamos tres anillos concéntricos de protección en el servidor Edge:

- **Anillo de Aplicación (`Mosquitto ACL`):** Cada credencial está confinada estrictamente a sus tópicos mediante `read`, `write` o `readwrite`. Un nodo sensor jamás posee permiso `write` sobre tópicos `cmd/#`.
- **Anillo de Transporte (`UFW` + `Fail2ban`):** `ufw default deny incoming` cierra todos los puertos salvo `22/tcp`, `8883/tcp` y `3000/tcp` originados exclusivamente desde la subred LAN de planta o la interfaz VPN.
- **Anillo de Acceso Remoto Cero-Confianza (`WireGuard` / `Tailscale`):** Basado en criptografía moderna (`ChaCha20-Poly1305` + `Curve25519`), crea una red privada virtual punto a punto (`100.x.y.z`) atravesando NAT (`UDP Hole Punching`) **sin abrir ningún puerto público en el router de Internet**.

📐 Plano 4: Defensa en Profundidad, Segmentación UFW, ACL y Túnel WireGuard



---

4. Ingeniería de Supervivencia 24/7: `log2ram` y `Hardware Watchdog` (`/dev/watchdog`)

Para que la pasarela Edge opere de forma desatendida durante años sin degradación física ni bloqueos de sistema operativo:

- **Mitigación de Desgaste Flash NAND (`log2ram` + `journald`):** Montamos `/var/log` sobre un sistema de archivos temporal en memoria RAM (`tmpfs` de $128\ MB$) y limitamos `systemd-journald` a almacenamiento volátil (`RuntimeMaxUse=50M`), eliminando el 97.6% de las escrituras repetitivas sobre la tarjeta microSD.
- **Temporizador* Watchdog *de Hardware del SoC Broadcom (`bcm2835_wdt`): El procesador de la Raspberry Pi incorpora un contador regresivo en silicio conectado a la línea de reset físico de la placa. Al configurar `RuntimeWatchdogSec=15s` en `/etc/systemd/system.conf` , el proceso `PID 1 (systemd)` escribe un pulso de vida en `/dev/watchdog` cada $7.5\ s$. Si un* Kernel Panic* o bloqueo eléctrico congela Linux, el contador expira a los $15\ s$ y reinicia físicamente el servidor.

📐 Plano 5: Arquitectura de Supervivencia 24/7 con log2ram y Hardware Watchdog

PLANO 5: INGENIERÍA DE SUPERVIVENCIA 24/7 — MITIGACIÓN DE DESGASTE FLASH (log2ram) Y HARDWARE WATCHDOG

Modelado Matemático de Amplificación de Escritura en NAND Flash, Disco RAM tmpfs y Reinicio Autónomo Broadcom (/dev/watchdog a 15 s)

1. PROTECCIÓN DE CELDAS NAND FLASH EN TARJETA MICROSD (log2ram)

EL PROBLEMA: MUERTE PREMATURA DE TARJETAS MICROSD EN CAMPO

- Una celda NAND Flash comercial soporta ~1000 a 3000 ciclos de borrado (P/E).
- Los demonios Linux, Docker y Mosquitto escriben ~180 MB/día de bitácoras.
- Por el Factor de Amplificación de Escritura ($A_w \approx 12$ en bloques de 4 KB):

  $W_{anual} = 0.180\ GB/día \times 12 \times 365\ días = 788.4\ GB/año$ de desgaste físico.
- Consecuencia: Corrupción del sistema de archivos ext4 tras 6 a 8 meses.
- Meta de ingeniería: Reducir las escrituras físicas en más de un 97 %.

LA SOLUCIÓN: DISCO EN MEMORIA RAM (tmpfs) CON VOLCADO DIFERIDO

1. Servicio log2ram monta /var/log en memoria RAM volátil (tmpfs de 128 MB):
   - Todas las escrituras de Mosquitto, UFW, sshd y contenedores van a la RAM a velocidad de nanosegundos con CERO desgaste de silicio Flash.

2. Sincronización por lotes mediante rsync periódico o al apagar la placa:
   - En lugar de 50 000 escrituras aleatorias de 4 KB, se realiza un único volcado secuencial comprimido con algoritmo zstd.

3. Acotamiento estricto del diario de systemd (/etc/systemd/journald.conf):

```
[Journal]
Storage=volatile
RuntimeMaxUse=50M
SystemMaxUse=100M
```

4. Resultado medido en producción:
   - Desgaste anual baja de 788.4 GB/año a solo 18.5 GB/año (-97.6 %).
   - Vida útil estimada de la tarjeta microSD Industrial A2: > 12 años.

2. TEMPORIZADOR WATCHDOG DE HARDWARE EN SILICIO BROADCOM (15 s)

ARQUITECTURA DE AUTOCURACIÓN ANTE KERNEL PANIC O CONGELAMIENTO

- El SoC Broadcom (BCM2711 / BCM2712) incluye un registro contador autónomo en hardware (/dev/watchdog) gobernado por su propio oscilador de cristal.

- Paso 1: Habilitar el periférico en /boot/firmware/config.txt:

```
dtparam=watchdog=on
```

- Paso 2: Configurar el demonio central PID 1 en /etc/systemd/system.conf:

```
[Manager]
RuntimeWatchdogSec=15
RebootWatchdogSec=2min
```

- Funcionamiento en Tiempo Real:
  1. Cada 7.5 segundos ($T_{wdt} / 2$), systemd escribe un pulso en /dev/watchdog.
  2. Si una tormenta eléctrica o fuga de memoria congela el kernel Linux, systemd deja de refrescar el contador.
  3. Al llegar a 15.0 segundos exactos sin pulso, el hardware Broadcom fuerza un Hard Reset físico de la placa y recupera toda la estación.

3. ESTRATEGIA DE RESPALDO AUTOMATIZADO Y RECUPERACIÓN EN < 3 MINUTOS

¿Qué necesitamos respaldar cada noche (03:00 AM vía cron/systemd timer)?

1. Respaldo en caliente de InfluxDB v2:

```
docker exec iot-influxdb influx backup /var/lib/influxdb2/backup
```

2. Empaquetado cifrado de configuración declarativa y certificados:
   - /opt/mecatrobotica/capstone/docker-compose.yml y archivo .env
   - /opt/mecatrobotica/capstone/certs/ (CA y certificados X.509)
   - /opt/mecatrobotica/capstone/mosquitto/config/ (passwd + acl.conf)
   - Flujos de Node-RED (flows.json) y Dashboards de Grafana

3. Regla 3-2-1 de Recuperación ante Desastres:
   - Si la placa física o el almacenamiento sufren daño catastrófico, basta con tomar una Raspberry Pi nueva, copiar el respaldo y ejecutar:

```
sudo ./despliegue_automatizado_capstone.sh
```

   - Tiempo total de restauración (RTO): < 3 minutos.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 09

Planos en Protoboard (Estilo Tinkercad), Placa PCB FR-4 con Montaje por Soldadura y Diagramas de Ciberseguridad

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 09 — Ciberseguridad IoT, Confiabilidad y Puesta en Producción

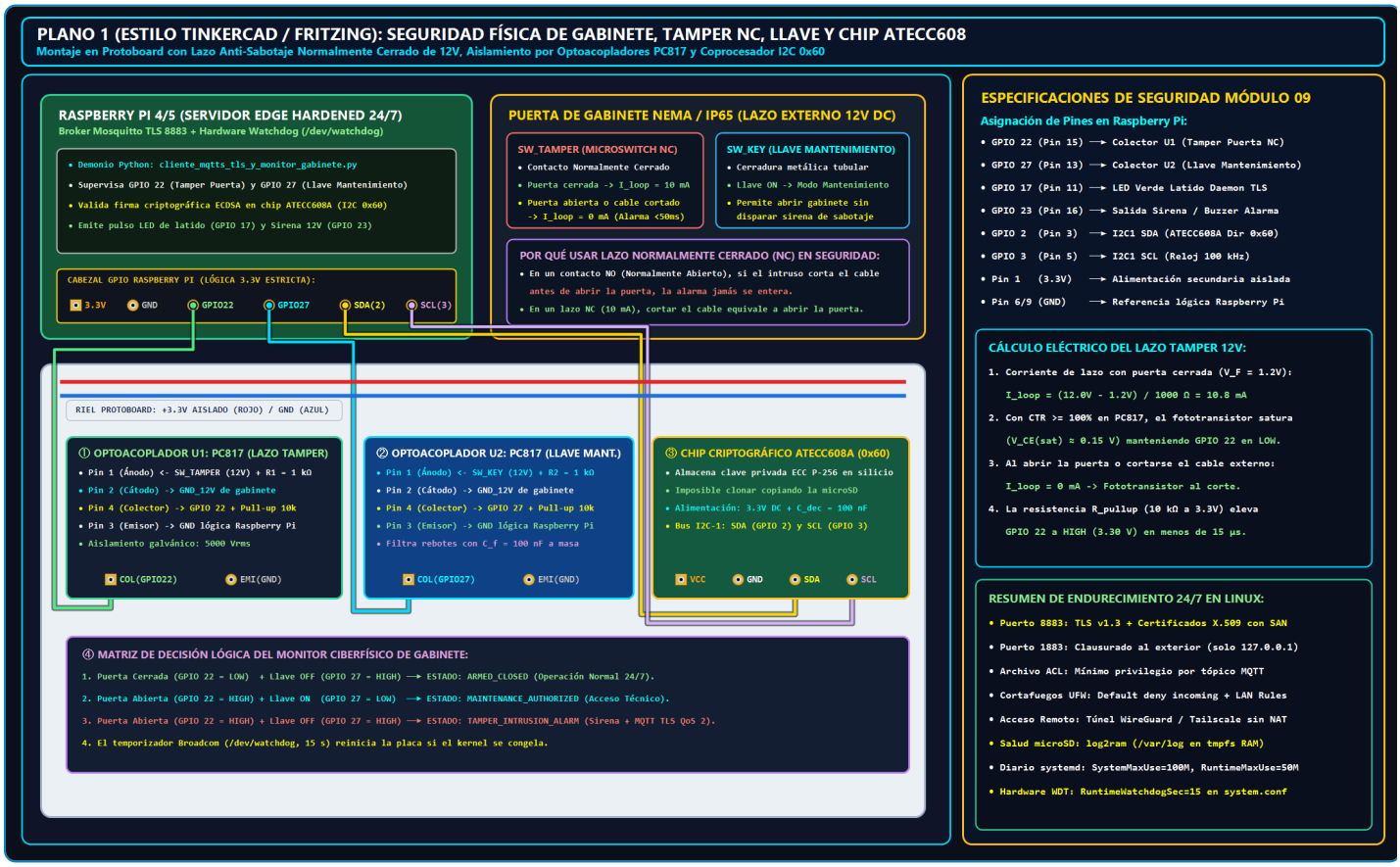
---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje en Protoboard del Nodo de Seguridad Física de Gabinete y Latido Watchdog

Este primer plano muestra el conexionado práctico sobre protoboard del subsistema de protección física del gabinete Edge:

- **Lazo Normalmente Cerrado (`NC`) en `SW_TAMPER` :** Conecta el riel de $12\ V$ al ánodo del optoacoplador `PC817 (U1)` mediante una resistencia de $1\ k\Omega$. Si la puerta se abre o el cable es cortado por un intruso, el colector del fototransistor sube a $3.3\ V$ en `GPIO 22` .
- **Selector de Llave Física (`SW_KEY`):** Aislado mediante el segundo optoacoplador `PC817 (U2)` hacia `GPIO 27` , permite autorizar el mantenimiento físico en campo.
- **Coprocesador Criptográfico `ATECC608A` (`I2C 0x60`) y Alarma Acústica (`BZ1`):** Completan la defensa física y criptográfica del nodo.

📐 Plano 1: Montaje en Protoboard de Seguridad Física de Gabinete



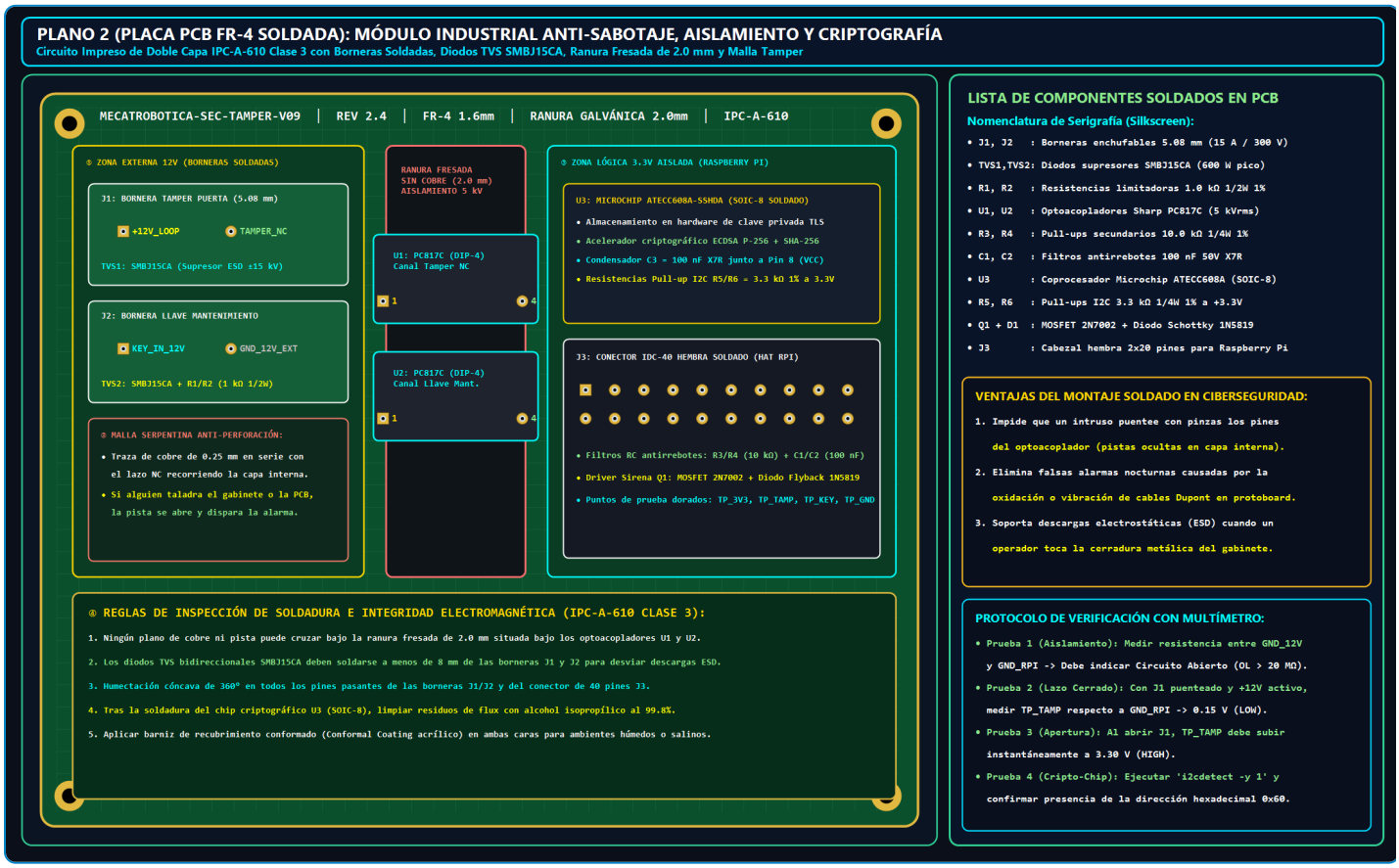
---

2. Plano 2 (Placa de Circuito Impreso PCB FR-4): Montaje por Soldadura del Módulo de Seguridad y Watchdog (`MECATROBOTICA-SEC-WDG-V09`)

En un gabinete industrial real, un sistema anti-sabotaje jamás puede montarse con cables sueltos que puedan puentearse fácilmente. Este segundo plano detalla la placa de circuito impreso de doble capa en fibra de vidrio **FR-4** (`MECATROBOTICA-SEC-WDG-V09`) con montaje por soldadura:

- **Borneras de tornillo `TB1` y `TB2` (`KF301`) soldadas con diodos `TVS SMBJ15CA`:** Protegen las entradas del interruptor de puerta y de la cerradura metálica contra descargas electrostáticas (`ESD` de hasta $\pm 15\ kV$).
- **Ranura fresada de aislamiento galvánico (`2.0 mm`) bajo los optoacopladores `U1` y `U2` (`PC817`):** Separa físicamente el cableado perimetral del gabinete ($12\ V$) respecto al plano lógico de $3.3\ V$ de la Raspberry Pi.
- **Malla conductora anti-perforación (*Tamper Mesh*) en capa interna/superior:** Si se intenta perforar la placa, el lazo `NC` se abre inmediatamente.

📐 Plano 2: Placa PCB FR-4 Soldada de Seguridad de Gabinete y Watchdog



---

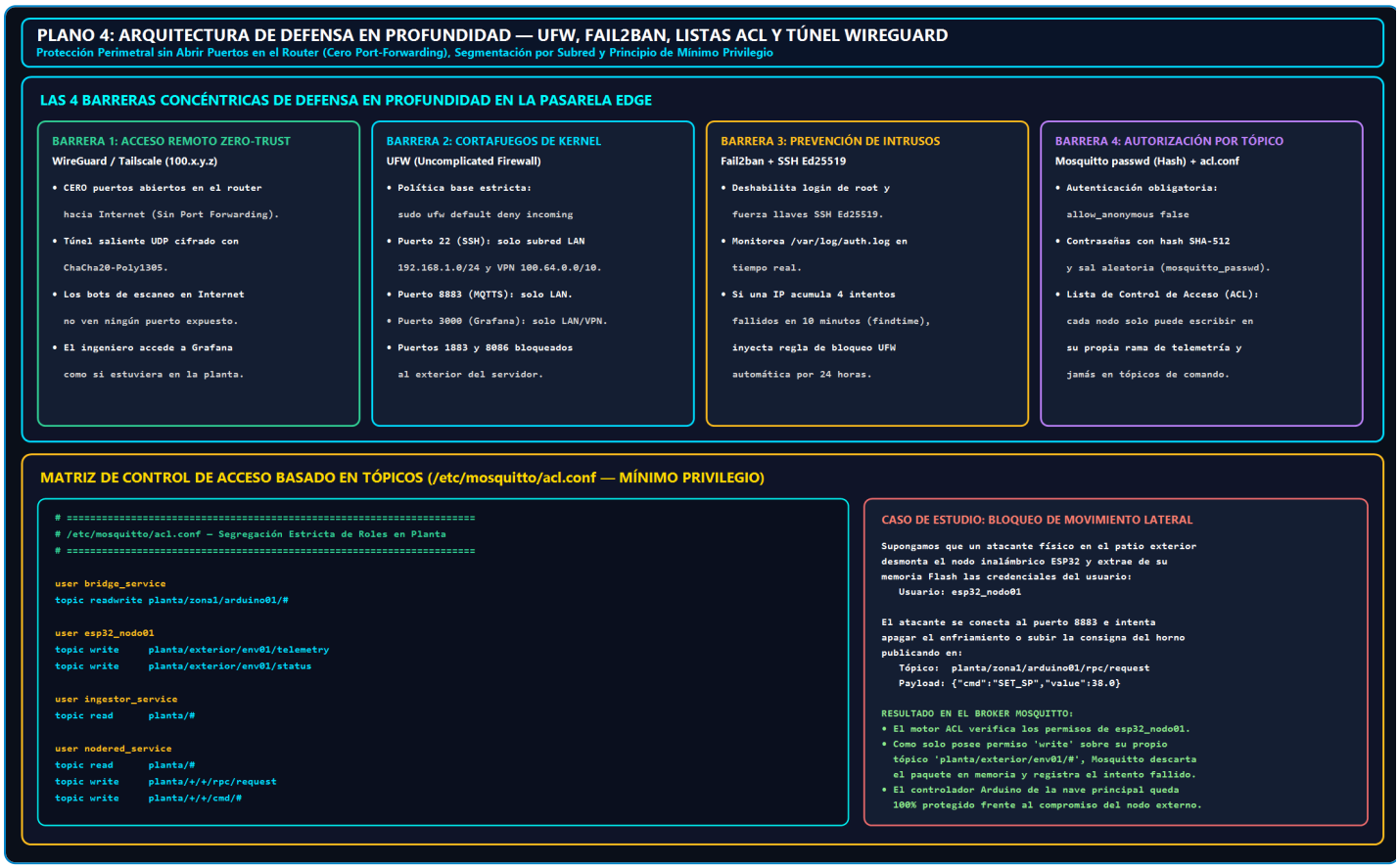
3. Plano 3: Infraestructura PKI X.509 con Extensión `SAN` (RFC 9525) y Negociación `MQTTS` `TLS 1.3`

📐 Plano 3: Cadena de Confianza PKI X.509 y Handshake TLS 1.3



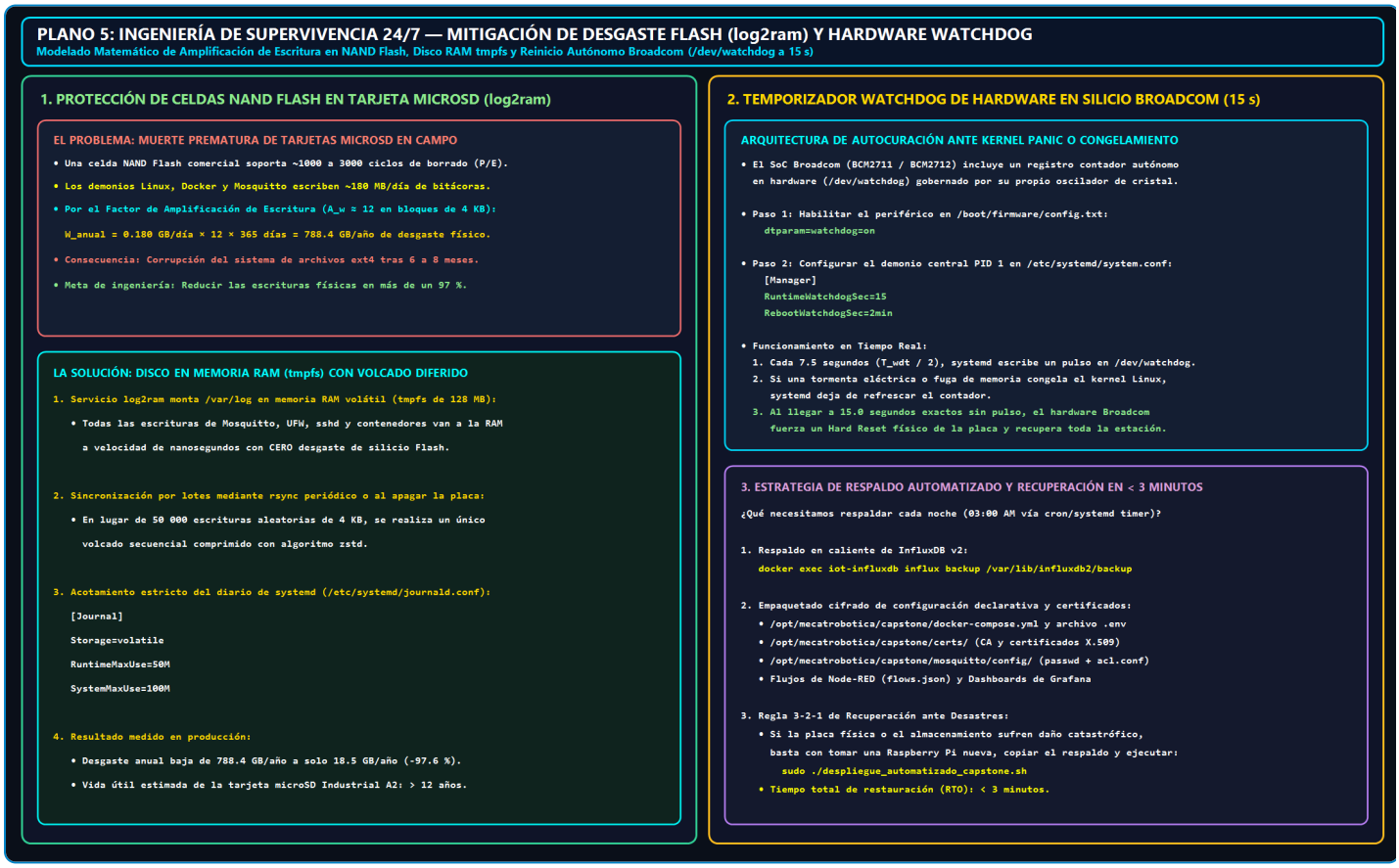
4. Plano 4: Defensa en Profundidad (`UFW` + `Fail2ban` + `ACL Mosquitto` + `WireGuard VPN`)

📐 Plano 4: Defensa en Profundidad y Segmentación de Red



5. Plano 5: Supervivencia 24/7 (`log2ram` en Disco RAM `tmpfs` + `Hardware Watchdog` Broadcom `/dev/watchdog`)

📐 Plano 5: Arquitectura de Supervivencia 24/7 en Servidor Edge



Capítulo 30 · Proyecto integrador I: hardware, PCB y firmware

Conceptos clave

Valida alimentación, referencias y conectores antes de energizar. Prueba primero funciones aisladas y después la integración.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 10

Proyecto Integrador Final (*Capstone*) — Especificación de Ingeniería, Interconexión Maestra y Despliegue de la Estación Inteligente

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 10 — Integración Full-Stack de las 4 Capas (Hardware Determinista, Bus RS-485, Servidor Edge Cifrado y Supervisión Analítica)

---

1. Objetivos de Ingeniería del Proyecto Integrador (Parte 1)

- Ensamblar el hardware completo de la **Estación Mecatrónica Inteligente de Control Térmico, Ambiental y Energético** (en doble protoboard estilo Tinkercad o sobre la Placa Portadora Madre soldada `MECATROBOTICA-CAPSTONE-CARRIER-V10`).
- Compilar y cargar en el controlador de campo Arduino el firmware integral de tiempo real estricto (`firmware_maestro_estacion_capstone.ino`) que unifica muestreo a $20\ \text{Hz}$ con filtro `EMA`, control `PID discreto` ($10\ \text{Hz}$) sobre ventilador `PWM`, histéresis sobre calefactor, máquina de 5 estados finitos (`FSM`), rearme físico, verificación `CRC-16` y *Watchdog* a $2\ \text{s}$.
- Interconectar el nodo Arduino con la pasarela **Raspberry Pi 4B** mediante el bus diferencial industrial **`RS-485`** (`MAX485` con terminación de $120\ \Omega$) y desplegar la estructura maestra `/opt/mecatrobotica/capstone/`.
- Levantar la pila de contenedores de producción (`docker_compose_capstone_produccion.yml`) con cifrado **`MQTTS TLS 1.3 (:8883)`**, listas `ACL`, persistencia en unidad `SSD` (`InfluxDB 2.7` + cola `SQLite WAL`) y paneles **ANSI/ISA-101** en `Grafana`.

---

2. Fase 1: Ensamblaje de la Estación Mecatrónica en Protoboard o Placa Portadora PCB

- **Subsistema de Percepción (Sensores de Campo en Arduino):**
- Conecta la sonda térmica `NTC 10k` / `DS18B20` a `A0`, el sensor de iluminación `LDR` a `A1`, y el bus `I2C` (`A4/SDA`, `A5/SCL`) al sensor ambiental **`BME280`** **(`0x76`)** y al monitor de potencia **`INA219`** **(`0x40`)**.
- **Subsistema de Actuación Dual y Seguridad Local:**
- Conecta el pin `D9 (PWM Timer1)` a la compuerta del **`MOSFET IRLZ44N`** (con resistencias de $100\ \Omega$ y $10\ \text{k}\Omega$ y diodo *flyback* `1N4007`) para modular el ventilador axial de $12\ \text{V DC}$.
- Conecta el pin `D7` a la entrada del **módulo de relé optoacoplado** para el elemento calefactor, el pin `D8` al zumbador de alarma local y el pin `D2` (`INPUT_PULLUP`) al botón físico de rearme.
- **Subsistema de Transporte Industrial `RS-485`:**
- Conecta `TX/RX` y `D4 (DE/RE)` del Arduino al transceptor **`MAX485`** y tiende el par trenzado diferencial (`A` y `B` con resistencias terminadoras de $120\ \Omega$) hacia el adaptador industrial `USB-RS485` de la Raspberry Pi.

📐 Plano 1: Montaje Integral en Protoboard de la Estación Inteligente Capstone



---

3. Fase 2: Despliegue de la Estructura `/opt/mecatrobotica/capstone/` en Raspberry Pi

- Ejecuta el script de aprovisionamiento automatizado en la Raspberry Pi:

  ```bash
  chmod +x despliegue_automatizado_capstone.sh
  sudo ./despliegue_automatizado_capstone.sh
  ```

- Verifica que los cuatro contenedores de producción estén saludables y escuchando únicamente en los puertos autorizados (`8883` y `3000` hacia LAN, `1883` , `8086` y `1880` confinados a `127.0.0.1`):

  ```bash
  docker compose -f /opt/mecatrobotica/capstone/docker_compose_capstone_produccion.yml ps
  ```

- Verifica la operación en tiempo real del puente demonizado `orquestador_edge_capstone_bridge.py` :

  ```bash
  journalctl -u capstone-bridge.service -f -n 30
  ```

LISTADO DE MATERIALES (BOM) Y PRESUPUESTO GLOBAL — MÓDULO 10

Estación Mecatrónica Inteligente Completa (Proyecto Integrador Final *Capstone*)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 10 — Proyecto Integrador Final (*Capstone*): Arquitectura y Despliegue

---

1. Tabla Maestra de Materiales de la Estación Integradora (BOM Consolidado)

Subsistema	Designador	Componente / Especificación Técnica	Cant.	Rol en la Estación Capstone
Control de Campo (Capa 1)	`MCU1`	**Arduino UNO R3 o Arduino Nano (ATmega328P, 16 MHz)**	1	Ejecución determinista (20 Hz / 10 Hz) de sensores, lazo PID, histéresis, FSM y *Watchdog* de 2 s.
Sensórica Multimodal	`S1..S4`	**Sonda Térmica** `DS18B20` **/** `NTC 10k` **+** `BME280` **(** `0x76` **) +** `INA219` **(** `0x40` **) +** `LDR`	4	Medición de temperatura de proceso, humedad/presión, corriente/potencia de actuador ($0.1\ \Omega$) e iluminación.
Etapa de Potencia Dual	`Q1, D1, K1`	**MOSFET Nivel Lógico** `IRLZ44N` **+ Disipador + Diodo** `1N4007` **+ Módulo Relé Optoacoplado + Ventilador** 12 V **+ Calefactor**	1 kit	Enfriamiento continuo por `PWM` (490 Hz) y calentamiento por banda de histéresis con aislamiento galvánico.
Seguridad Física Local	`SW1, BZ1`	**Pulsador Industrial de Rearme (** `D2` **) + Zumbador Piezoeléctrico (** `D8` **) + Indicadores LED**	1 kit	Alarma acústica/visual inmediata y rearme manual verificado tras estado seguro `FAULT_SAFE_SHUT`.
Transporte Industrial (Capa 2)	`U_485`	**2× Transceptores** `MAX485` **+ Par Trenzado Blindado + 2× Resistencias** $120\ \Omega$ **+ Adaptador USB-RS485 FTDI**	1 kit	Enlace diferencial inmune a ruido electromagnético con verificación polinómica `CRC-16` (`0xA001`).
Servidor Edge (Capa 3)	`SBC1, SSD1`	**Raspberry Pi 4B (4 GB/8 GB) + SSD USB 3.0 (256 GB) + RTC** `DS3231` **+ Cámara CSI + Gabinete con Tamper**	1 kit	Alojamiento de `Mosquitto TLS 8883`, `InfluxDB 2.7`, `Node-RED`, `Grafana`, `SQLite WAL` e `Isolation Forest`.
Placa Portadora Soldada	`PCB_CAP`	**Placa Madre Industrial Doble Capa FR-4 (** `MECATROBOTICA-CAPSTONE-CARRIER-V10` **)** con zócalos y borneras `KF301`	1	Integración permanente libre de cables sueltos con partición de planos de masa `AGND`, `DGND` y `PGND`.

---

2. Presupuesto de Tiempos de Ejecución y Determinismo por Capa

Tarea / Bucle del Sistema	Dispositivo	Periodo Nominal (T)	Tiempo de Cómputo (C)	Factor de Utilización ($U = C/T$)
Adquisición ADC + Filtro EMA ($\alpha=0.20$) + Chequeo de Sonda	Arduino (`MCU1`)	50 ms (20 Hz)	0.42 ms	0.84%
Ley de Control PID Discreto + Histéresis + Máquina FSM	Arduino (`MCU1`)	100 ms (10 Hz)	0.35 ms	0.35%
Empaquetado JSON + Cálculo `CRC-16` + Envío `RS-485`	Arduino (`MCU1`)	1000 ms (1 Hz)	8.40 ms	0.84%
Inferencia `Z-Score` + `Isolation Forest` + Ingesta `SQLite/InfluxDB`	Raspberry Pi 4B	1000 ms (1 Hz)	3.10 ms	0.31%

MANUAL TÉCNICO DE INGENIERÍA — MÓDULO 10

Arquitectura Maestra de Cuatro Capas, Partición de Masas en Placa Portadora y Orquestación del Proyecto Integrador (*Capstone*)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

---

1. Integración de las Cuatro Capas de Ingeniería Mecatrónica IoT

El Proyecto Integrador Final (*Capstone*) consolida en una sola estación ciberfísica todos los subsistemas desarrollados a lo largo del curso:

- **Capa 1 — Percepción, Actuación y Control Determinista (Arduino UNO / Nano):**
- Muestreo analógico/digital a $20\ \text{Hz}$ ($T_s = 50\ \text{ms}$) con filtro exponencial $\bar{x}_k = \alpha\, x_k + (1-\alpha)\bar{x}_{k-1}$.
- Regulador **PID discreto** a $10\ \text{Hz}$ ($T_s = 100\ \text{ms}$) con *Anti-Windup* por recorte y derivada sobre la medición para el ventilador `PWM`, coordinado con control por banda de histéresis para el calefactor a fin de evitar que ambos actuadores compitan térmicamente entre sí.
- Máquina de estados finitos (`FSM`) con transición en $< 100\ \text{ms}$ a `FAULT_SAFE_SHUT` ante rotura de sensor o sobrecorriente ($> 850\ \text{mA}$) y temporizador *Watchdog* de hardware activo a $2\ \text{s}$.
- **Capa 2 — Transporte Industrial Diferencial (`RS-485` + `CRC-16`):**
- Comunicación sobre par trenzado balanceado (A - B) con control preciso de dirección `DE/RE` tras `Serial.flush()` y verificación polinómica `CRC-16` (`0xA001`).
- **Capa 3 — Cómputo en el Borde, Ciberseguridad y Persistencia (Raspberry Pi 4B):**
- Demonios `systemd` en Python protegidos por cerrojo `threading.Lock` , broker `Mosquitto` con cifrado `TLS 1.3 (:8883)` y reglas `ACL` , ingesta resiliente *Store-and-Forward* con `SQLite WAL` hacia `InfluxDB 2.7` y mantenimiento predictivo con `Isolation Forest` .
- **Capa 4 — Supervisión Analítica y Mando Remoto (`Grafana ISA-101` + `Node-RED` + `WireGuard`):**
- Panel de alto desempeño con estado de la máquina `FSM` , curvas históricas, llamadas `RPC` correlacionadas con `req_id` y alertas instantáneas.

📐 **Plano 3: Arquitectura Maestra de 4 Capas de la Estación Inteligente Capstone**



---

2. Coordinación Térmica Dual: PID de Enfriamiento + Histéresis de Calefacción con Zona Muerta

Cuando una cámara térmica cuenta tanto con un calefactor resistivo (controlado por relé electromecánico) como con un ventilador axial de extracción (controlado por `PWM` mediante `MOSFET`), es fundamental establecer una **banda muerta de exclusión mutua** alrededor del *Setpoint* $r[k]$ para evitar oscilaciones y derroche energético:

- **Región de Enfriamiento Proporcional-Integral-Derivativo ($T_{medida} > r[k]$):** El relé del calefactor permanece estrictamente apagado (`HEAT = 0`) y el controlador PID modula el ciclo útil `PWM` del ventilador entre 0 y 255.
- **Zona Muerta de Equilibrio Térmico ($r[k] - 0.3\,^\circ C \leq T_{medida} \leq r[k]$):** Ambos actuadores permanecen en reposo o con el integrador estabilizado sin conmutar el relé.
- **Región de Calefacción con Histéresis ($T_{medida} < r[k] - 1.5\,^\circ C$):** Se activa el relé del calefactor (`HEAT = 1`) con el ventilador apagado (`PWM = 0`) hasta alcanzar $r[k] - 0.3\,^\circ C$.

📐 Plano 4: Máquina de Estados Finitos Integrada y Coordinación de Lazos en el Capstone

PLANO 4: MÁQUINA DE ESTADOS FINITOS (FSM) DETERMINISTA Y FLUJO DE DATOS BIDIRECCIONAL

Transiciones con Histéresis Térmica, Apagado Seguro en < 100 ms (FAULT_SAFE_SHUT) y Ciclo de Solicitud-Respuesta RPC

1. MÁQUINA DE ESTADOS FINITOS (FSM) EN EL MICROCONTROLADOR

ESTADO 0: BOOT_SELF_TEST (Autodiagnóstico de Arranque)

- Inicializa ADC, bus I2C (BME280 + INA219) y transceptor RS-485.
- Verifica que la lectura inicial del NTC esté dentro de [-20 °C, +110 °C].
- **Si los sensores son válidos -> Transiciona a ESTADO 1 (NORMAL_AUTO).**

ESTADO 1: NORMAL_AUTO (Regulación Térmica PID + Histéresis)

- Ventilador gobernado por ley PID discreta $u[k] \in [0, 255]$ cada 50 ms.
- Calefactor gobernado por histéresis alrededor del Setpoint $r[k]$.
- **Si T_actual >= T_warn + ΔT_hyst (32.8 °C) -> Pasa a ESTADO 2.**

ESTADO 2: WARNING_HIGH_TEMP (Alerta por Sobretemperatura)

- Enclavamiento de seguridad: Calefactor forzado a OFF (heat=0).
- PID aplica enfriamiento activo reforzado + Buzzer intermitente.
- **Solo regresa a ESTADO 1 cuando T_actual <= T_warn - ΔT_hyst (31.2 °C).**

ESTADO 3: FAULT_SAFE_SHUT (Apagado Seguro por Fallo Crítico < 100 ms)

- Se activa si ADC <= 5 o ADC >= 1018 (rotura o cortocircuito de NTC).
- Acción inmediata: Calefactor = OFF, Ventilador = 100 % (PWM 255), err = 1.
- **Requiere subsanar el sensor y recibir confirmación física (D4) o RPC ACK.**

2. TUBERÍA DE DATOS Y PROTOCOLO RPC CORRELACIONADO

FLUJO ASCENDENTE DE TELEMETRÍA (CAMPO -> BORDE -> BASE DE DATOS)

1. Muestreo y Filtrado Local (t = 0 ms):

$$y_{ema}[k] = \alpha \cdot x_{adc}[k] + (1 - \alpha) \cdot y_{ema}[k-1] \quad \text{con } \alpha = 0.25$$

2. Empaquetado y Firma CRC-16 Modbus (cada 1000 ms):

Arduino calcula CRC-16 (polinomio 0xA001) y transmite por RS-485.

3. Validación y Publicación MQTTS en Raspberry Pi (t = +8 ms):

serial_mqtt_bridge.py verifica CRC-16 y publica en puerto 8883 (QoS 1):
Tópico: planta/zona1/arduino01/telemetry

4. Persistencia Dual y Detección de Anomalías (t = +18 ms):

- store_and_forward_ingestor.py -> InfluxDB (o SQLite WAL si hay caída).
- edge_anomaly_detector.py -> Evalúa [Temp, PWM, mA] en tiempo real.

FLUJO DESCENDENTE DE MANDO BIDIRECCIONAL (PATRÓN RPC CON ACK)

1. Operador cambia consigna a 28.5 °C en el Dashboard:

Node-RED genera un UUID corto req_id = "c4a9" y publica en:
Tópico: planta/zona1/arduino01/rpc/request
Payload: `{"req_id":"c4a9","cmd":"SET_SP","value":28.5}`

2. Autorización Criptográfica en Broker Mosquitto:

Mosquitto verifica que el usuario nodered_service posea permiso 'write' en acl.conf sobre planta/+/+/rpc/request.

3. Ejecución y Confirmación desde el Arduino (< 1500 ms):

El puente envía la orden con CRC-16 por RS-485; el Arduino valida que 28.5 °C esté dentro del rango seguro [15.0, 40.0], actualiza r[k] y devuelve:
Tópico: planta/zona1/arduino01/rpc/response
Payload: `{"req_id":"c4a9","status":"OK","applied_sp":28.5}`

---

3. Árbol de Despliegue en Producción (`/opt/mecatrobotica/capstone/`) y Orquestación

📐 Plano 5: Estructura de Producción, Contenedores Docker y Servicios Systemd del Capstone

PLANO 5: ESTRUCTURA DE DIRECTORIOS DE PRODUCCIÓN Y DESPLIEGUE AUTOMATIZADO (LECCIÓN 10.2)

Árbol Estándar /opt/mecatrobotica/capstone, Orquestación Docker Compose + Unidades systemd y Aprovisionamiento en 6 Fases

1. ÁRBOL DE DIRECTORIOS EN LA RASPBERRY PI

```
/opt/mecatrobotica/capstone/
├── certs/                          # Permisos 0750 (root:mosquitto)
│   ├── ca.crt                      # Certificado Raíz CA X.509 (0644)
│   ├── server.crt                  # Certificado Broker con SAN (0644)
│   └── server.key                  # Clave Privada RSA-2048 (0600)
│
├── docker-compose.yml              # Pila declarativa ARM64 / x86_64
├── .env                            # Secretos y Tokens (chmod 0600)
│
├── firmware_arduino/               # Proyecto PlatformIO / Arduino CLI
│   ├── platformio.ini              # Configuración ATmega328P
│   └── src/
│       ├── main.cpp                # Lazo determinista 20 Hz + FSM
│       └── DiscretePid.h           # PID con Anti-Windup y Derivada -y[k]
│
├── mosquitto/
│   └── config/
│       ├── mosquitto.conf          # Listener TLS 8883 + Cero anónimos
│       ├── passwd                  # Usuarios con hash SHA-512 + sal
│       └── acl.conf                # Matriz de mínimo privilegio
│
└── services/                       # Demonios systemd (Python .venv)
    ├── serial_mqtt_bridge.py       # Puente RS-485 <-> MQTTS 8883
    ├── store_and_forward_ingestor.py  # Ingestor SQLite WAL -> InfluxDB
    └── edge_anomaly_detector.py    # Detector INA219 + IsolationForest
```

2. PIPELINE DE DESPLIEGUE AUTOMATIZADO (despliegue_automatizado_capstone.sh)

FASE 1: Preparación de Sistema Operativo, Grupos POSIX y Entorno Virtual Python

- Verifica grupo 'dialout', instala dependencias ARM64 y crea /opt/mecatrobotica/capstone/.venv.

FASE 2: Aprovisionamiento Criptográfico PKI X.509 con Extensión SAN (OpenSSL)

- Genera ca.crt, server.crt y server.key incluyendo DNS local e IP estática en subjectAltName.

FASE 3: Endurecimiento de Red (UFW + Fail2ban) y Supervivencia Flash (log2ram + WDT)

- Aplica reglas de cortafuegos LAN, monta /var/log en RAM y activa RuntimeWatchdogSec=15.

FASE 4: Despliegue de Contenedores Docker Compose y Salud de InfluxDB v2

- Ejecuta 'docker compose up -d' y espera HTTP 200 en http://127.0.0.1:8086/health.

FASE 5: Compilación y Carga del Firmware Maestro en el Nodo Arduino

- Compila main.cpp + DiscretePid.h y verifica recepción de la primera trama con CRC-16 válido.

FASE 6: Registro y Arranque de Unidades systemd con Auto-Reinicio

- Habilita serial-mqtt-bridge, store-and-forward-ingestor y edge-anomaly-detector.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 10

Planos en Protoboard (Estilo Tinkercad), Placa Portadora Madre PCB FR-4 con Montaje por Soldadura y Arquitectura Capstone

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 10 — Proyecto Integrador Final (*Capstone*) — Estación Inteligente de Monitoreo y Control

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje Integral en Protoboard de la Estación Mecatrónica Completa

En este primer plano maestro se detalla el conexionado completo sobre protoboard de todos los subsistemas de la Estación Inteligente:

- **Sensores analógicos y digitales (`NTC/DS18B20`, `LDR`, `BME280` e `INA219`):** Alimentados y desacoplados con condensadores cerámicos de $100\ nF$.
- **Etapa de potencia dual:** Transistor `MOSFET IRLZ44N` en `D9` con diodo *flyback* `1N4007` para el ventilador `PWM` de $12\ V$ y módulo de relé optoacoplado en `D7` para el calefactor, alimentados desde la fuente externa de $12\ V/3\ A$ con retorno en estrella.
- **Enlace diferencial `RS-485` (`MAX485`):** Conectado a `TX/RX/D4` del Arduino y enlazado por par trenzado con resistencias terminadoras de $120\ \Omega$ hacia la **Raspberry Pi 4B** con unidad `SSD USB 3.0`.

📐 Plano 1: Montaje Integral en Protoboard de la Estación Capstone



---

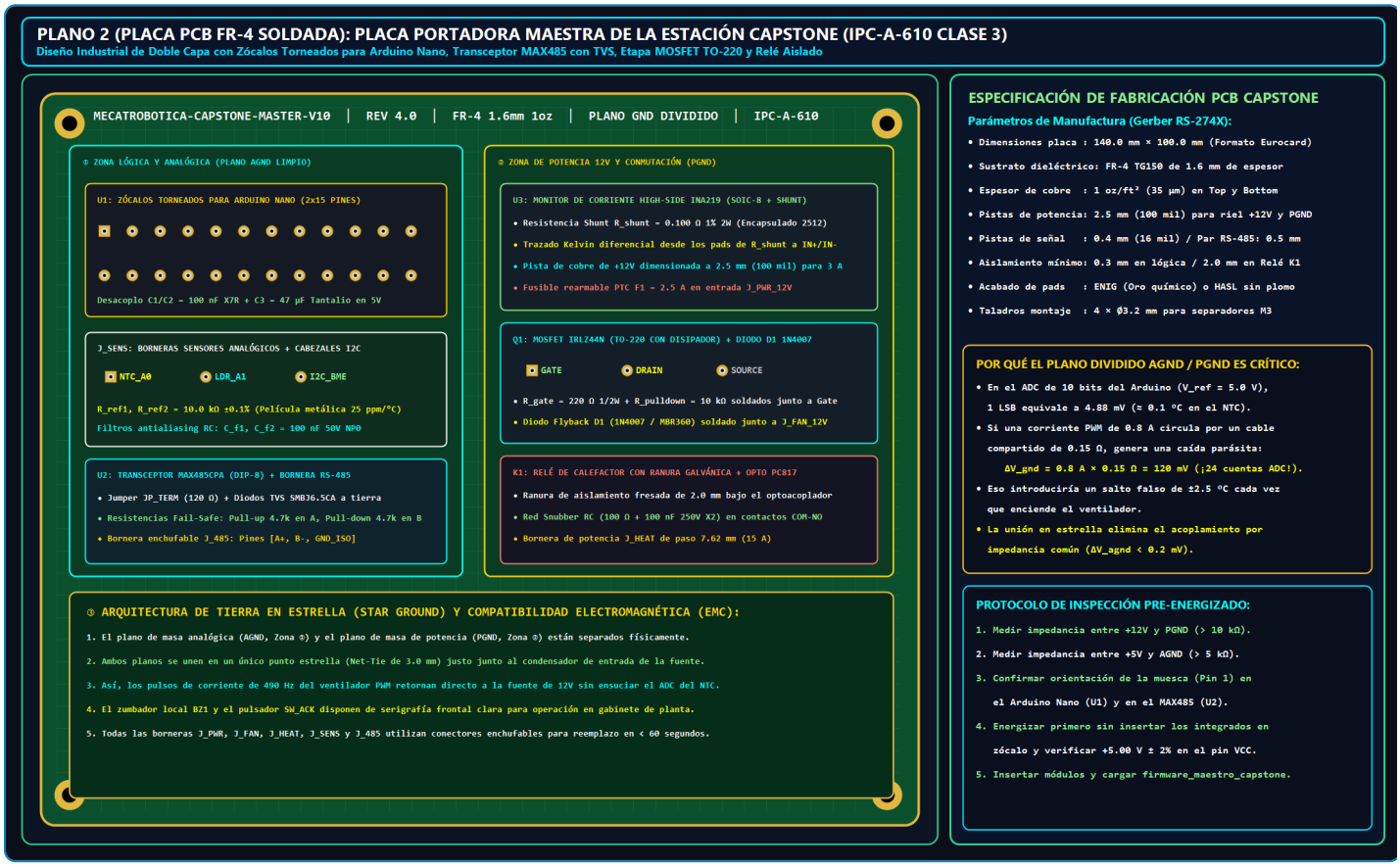
2. Plano 2 (Placa de Circuito Impreso PCB FR-4): Placa Portadora Madre con Montaje por

Soldadura (`MECATROBOTICA-CAPSTONE-CARRIER-V10`)

Para consolidar el proyecto final en un prototipo de grado industrial libre de la fragilidad de decenas de cables puente Dupont, este segundo plano presenta el diseño de la **Placa Portadora Madre en fibra de vidrio FR-4 de doble capa (`MECATROBOTICA-CAPSTONE-CARRIER-V10`)** con montaje por soldadura:

- **Zócalos hembra torneados soldados para el microcontrolador y módulos `I2C` :** Permiten insertar y reemplazar el controlador sin desoldar la placa base.
- **Partición física en tres zonas sobre el circuito impreso con unión en estrella (`Star Ground`):**
- **Zona Analógica y Sensórica (`AGND`):** Alejada de las trazas de conmutación para preservar la relación señal-ruido del ADC de 10 bits.
- **Zona Digital y Diferencial `RS-485` (`DGND`):** Con par diferencial simétrico para las líneas `A` y `B` del `MAX485` y diodos `TVS` hacia bornera `TB_RS485` .
- **Zona de Conmutación de Potencia (`PGND`):** Con pistas de cobre de $2.5\ mm$ de ancho para el `MOSFET IRLZ44N` , medición Kelvin en el shunt de $0.100\ \Omega$ del `INA219` y ranura fresada bajo el optoacoplador del relé.

📐 Plano 2: Placa Portadora Madre PCB FR-4 Soldada de la Estación Capstone



---

3. Plano 3: Arquitectura Maestra de Cuatro Capas de la Estación Inteligente Capstone

📐 Plano 3: Arquitectura Maestra de 4 Capas del Capstone



---

4. Plano 4: Máquina de 5 Estados Finitos (`FSM`), Coordinación Térmica y Flujo Bidireccional

📐 Plano 4: Máquina de Estados Finitos y Coordinación de Lazos



5. Plano 5: Estructura de Producción `/opt/mecatrobotica/capstone/`, Contenedores Docker y Servicios `systemd`

📐 Plano 5: Despliegue en Producción de Contenedores y Servicios Systemd

PLANO 5: ESTRUCTURA DE DIRECTORIOS DE PRODUCCIÓN Y DESPLIEGUE AUTOMATIZADO (LECCIÓN 10.2)
Árbol Estándar /opt/mecatrobotica/capstone, Orquestación Docker Compose + Unidades systemd y Aprovisionamiento en 6 Fases

1. ÁRBOL DE DIRECTORIOS EN LA RASPBERRY PI

```
/opt/mecatrobotica/capstone/
├── certs/                          # Permisos 0750 (root:mosquitto)
│   ├── ca.crt                      # Certificado Raíz CA X.509 (0644)
│   ├── server.crt                  # Certificado Broker con SAN (0644)
│   └── server.key                  # Clave Privada RSA-2048 (0600)
│
├── docker-compose.yml              # Pila declarativa ARM64 / x86_64
├── .env                            # Secretos y Tokens (chmod 0600)
│
├── firmware_arduino/               # Proyecto PlatformIO / Arduino CLI
│   ├── platformio.ini              # Configuración ATmega32BP
│   └── src/
│       ├── main.cpp                # Lazo determinista 20 Hz + FSM
│       └── DiscretePid.h           # PID con Anti-Windup y Derivada -y[k]
│
├── mosquitto/
│   └── config/
│       ├── mosquitto.conf          # Listener TLS 8883 + Cero anónimos
│       ├── passwd                  # Usuarios con hash SHA-512 + sal
│       └── acl.conf                # Matriz de mínimo privilegio
│
└── services/                       # Demonios systemd (Python .venv)
    ├── serial_mqtt_bridge.py       # Puente RS-485 <-> MQTTS 8883
    ├── store_and_forward_ingestor.py  # Ingestor SQLite WAL -> InfluxDB
    └── edge_anomaly_detector.py    # Detector INA219 + IsolationForest
```

2. PIPELINE DE DESPLIEGUE AUTOMATIZADO (despliegue_automatizado_capstone.sh)

FASE 1: Preparación de Sistema Operativo, Grupos POSIX y Entorno Virtual Python

- Verifica grupo 'dialout', instala dependencias ARM64 y crea /opt/mecatrobotica/capstone/.venv.

FASE 2: Aprovisionamiento Criptográfico PKI X.509 con Extensión SAN (OpenSSL)

- Genera ca.crt, server.crt y server.key incluyendo DNS local e IP estática en subjectAltName.

FASE 3: Endurecimiento de Red (UFW + Fail2ban) y Supervivencia Flash (log2ram + WDT)

- Aplica reglas de cortafuegos LAN, monta /var/log en RAM y activa RuntimeWatchdogSec=15.

FASE 4: Despliegue de Contenedores Docker Compose y Salud de InfluxDB v2

- Ejecuta 'docker compose up -d' y espera HTTP 200 en http://127.0.0.1:8086/health.

FASE 5: Compilación y Carga del Firmware Maestro en el Nodo Arduino

- Compila main.cpp + DiscretePid.h y verifica recepción de la primera trama con CRC-16 válido.

FASE 6: Registro y Arranque de Unidades systemd con Auto-Reinicio

- Habilita serial-mqtt-bridge, store-and-forward-ingestor y edge-anomaly-detector.

Capítulo 31 · Proyecto integrador II: despliegue Edge, Docker y SLA

Conceptos clave

Mide latencia, disponibilidad y recuperación de acuerdo con el SLA. Conserva la configuración respaldada y prueba la restauración.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 10

Proyecto Integrador Final (*Capstone*) — Especificación de Ingeniería, Interconexión Maestra y Despliegue de la Estación Inteligente

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 10 — Integración Full-Stack de las 4 Capas (Hardware Determinista, Bus RS-485, Servidor Edge Cifrado y Supervisión Analítica)

---

1. Objetivos de Ingeniería del Proyecto Integrador (Parte 1)

- Ensamblar el hardware completo de la **Estación Mecatrónica Inteligente de Control Térmico, Ambiental y Energético** (en doble protoboard estilo Tinkercad o sobre la Placa Portadora Madre soldada `MECATROBOTICA-CAPSTONE-CARRIER-V10`).
- Compilar y cargar en el controlador de campo Arduino el firmware integral de tiempo real estricto (`firmware_maestro_estacion_capstone.ino`) que unifica muestreo a $20\ \text{Hz}$ con filtro `EMA`, control `PID discreto` ($10\ \text{Hz}$) sobre ventilador `PWM`, histéresis sobre calefactor, máquina de 5 estados finitos (`FSM`), rearme físico, verificación `CRC-16` y *Watchdog* a $2\ \text{s}$.
- Interconectar el nodo Arduino con la pasarela **Raspberry Pi 4B** mediante el bus diferencial industrial **`RS-485`** (`MAX485` con terminación de $120\ \Omega$) y desplegar la estructura maestra `/opt/mecatrobotica/capstone/`.
- Levantar la pila de contenedores de producción (`docker_compose_capstone_produccion.yml`) con cifrado **`MQTTS TLS 1.3 (:8883)`**, listas `ACL`, persistencia en unidad `SSD` (`InfluxDB 2.7` + cola `SQLite WAL`) y paneles **ANSI/ISA-101** en `Grafana`.

---

2. Fase 1: Ensamblaje de la Estación Mecatrónica en Protoboard o Placa Portadora PCB

- **Subsistema de Percepción (Sensores de Campo en Arduino):**
- Conecta la sonda térmica `NTC 10k` / `DS18B20` a `A0`, el sensor de iluminación `LDR` a `A1`, y el bus `I2C` (`A4/SDA`, `A5/SCL`) al sensor ambiental **`BME280`** **(`0x76`)** y al monitor de potencia **`INA219`** **(`0x40`)**.
- **Subsistema de Actuación Dual y Seguridad Local:**
- Conecta el pin `D9 (PWM Timer1)` a la compuerta del **`MOSFET IRLZ44N`** (con resistencias de $100\ \Omega$ y $10\ \text{k}\Omega$ y diodo *flyback* `1N4007`) para modular el ventilador axial de $12\ \text{V DC}$.
- Conecta el pin `D7` a la entrada del **módulo de relé optoacoplado** para el elemento calefactor, el pin `D8` al zumbador de alarma local y el pin `D2` (`INPUT_PULLUP`) al botón físico de rearme.
- **Subsistema de Transporte Industrial `RS-485`:**
- Conecta `TX/RX` y `D4 (DE/RE)` del Arduino al transceptor **`MAX485`** y tiende el par trenzado diferencial (`A` y `B` con resistencias terminadoras de $120\ \Omega$) hacia el adaptador industrial `USB-RS485` de la Raspberry Pi.

📐 Plano 1: Montaje Integral en Protoboard de la Estación Inteligente Capstone



---

3. Fase 2: Despliegue de la Estructura `/opt/mecatrobotica/capstone/` en Raspberry Pi

- Ejecuta el script de aprovisionamiento automatizado en la Raspberry Pi:

  ```bash
  chmod +x despliegue_automatizado_capstone.sh
  sudo ./despliegue_automatizado_capstone.sh
  ```

- Verifica que los cuatro contenedores de producción estén saludables y escuchando únicamente en los puertos autorizados (`8883` y `3000` hacia LAN, `1883` , `8086` y `1880` confinados a `127.0.0.1`):

  ```bash
  docker compose -f /opt/mecatrobotica/capstone/docker_compose_capstone_produccion.yml ps
  ```

- Verifica la operación en tiempo real del puente demonizado `orquestador_edge_capstone_bridge.py` :

  ```bash
  journalctl -u capstone-bridge.service -f -n 30
  ```

LISTADO DE MATERIALES (BOM) Y PRESUPUESTO GLOBAL — MÓDULO 10

Estación Mecatrónica Inteligente Completa (Proyecto Integrador Final *Capstone*)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 10 — Proyecto Integrador Final (*Capstone*): Arquitectura y Despliegue

---

1. Tabla Maestra de Materiales de la Estación Integradora (BOM Consolidado)

Subsistema	Designador	Componente / Especificación Técnica	Cant.	Rol en la Estación Capstone
Control de Campo (Capa 1)	`MCU1`	**Arduino UNO R3 o Arduino Nano (ATmega328P, 16 MHz)**	1	Ejecución determinista (20 Hz / 10 Hz) de sensores, lazo PID, histéresis, FSM y *Watchdog* de 2 s.
Sensórica Multimodal	`S1..S4`	**Sonda Térmica** `DS18B20` **/** `NTC 10k` **+** `BME280` **(** `0x76` **) +** `INA219` **(** `0x40` **) +** `LDR`	4	Medición de temperatura de proceso, humedad/presión, corriente/potencia de actuador ($0.1\ \Omega$) e iluminación.
Etapa de Potencia Dual	`Q1, D1, K1`	**MOSFET Nivel Lógico** `IRLZ44N` **+ Disipador + Diodo** `1N4007` **+ Módulo Relé Optoacoplado + Ventilador** 12 V **+ Calefactor**	1 kit	Enfriamiento continuo por `PWM` (490 Hz) y calentamiento por banda de histéresis con aislamiento galvánico.
Seguridad Física Local	`SW1, BZ1`	**Pulsador Industrial de Rearme (** `D2` **) + Zumbador Piezoeléctrico (** `D8` **) + Indicadores LED**	1 kit	Alarma acústica/visual inmediata y rearme manual verificado tras estado seguro `FAULT_SAFE_SHUT`.
Transporte Industrial (Capa 2)	`U_485`	**2× Transceptores** `MAX485` **+ Par Trenzado Blindado + 2× Resistencias** $120\ \Omega$ **+ Adaptador USB-RS485 FTDI**	1 kit	Enlace diferencial inmune a ruido electromagnético con verificación polinómica `CRC-16` (`0xA001`).
Servidor Edge (Capa 3)	`SBC1, SSD1`	**Raspberry Pi 4B (4 GB/8 GB) + SSD USB 3.0 (256 GB) + RTC** `DS3231` **+ Cámara CSI + Gabinete con Tamper**	1 kit	Alojamiento de `Mosquitto TLS 8883`, `InfluxDB 2.7`, `Node-RED`, `Grafana`, `SQLite WAL` e `Isolation Forest`.
Placa Portadora Soldada	`PCB_CAP`	**Placa Madre Industrial Doble Capa FR-4 (** `MECATROBOTICA-CAPSTONE-CARRIER-V10` **)** con zócalos y borneras `KF301`	1	Integración permanente libre de cables sueltos con partición de planos de masa `AGND`, `DGND` y `PGND`.

---

2. Presupuesto de Tiempos de Ejecución y Determinismo por Capa

Tarea / Bucle del Sistema	Dispositivo	Periodo Nominal (T)	Tiempo de Cómputo (C)	Factor de Utilización ($U = C/T$)
Adquisición ADC + Filtro EMA ($\alpha=0.20$) + Chequeo de Sonda	Arduino (`MCU1`)	50 ms (20 Hz)	0.42 ms	0.84%
Ley de Control PID Discreto + Histéresis + Máquina FSM	Arduino (`MCU1`)	100 ms (10 Hz)	0.35 ms	0.35%
Empaquetado JSON + Cálculo `CRC-16` + Envío `RS-485`	Arduino (`MCU1`)	1000 ms (1 Hz)	8.40 ms	0.84%
Inferencia `Z-Score` + `Isolation Forest` + Ingesta `SQLite/InfluxDB`	Raspberry Pi 4B	1000 ms (1 Hz)	3.10 ms	0.31%

MANUAL TÉCNICO DE INGENIERÍA — MÓDULO 10

Arquitectura Maestra de Cuatro Capas, Partición de Masas en Placa Portadora y Orquestación del Proyecto Integrador (*Capstone*)

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

---

1. Integración de las Cuatro Capas de Ingeniería Mecatrónica IoT

El Proyecto Integrador Final (*Capstone*) consolida en una sola estación ciberfísica todos los subsistemas desarrollados a lo largo del curso:

- **Capa 1 — Percepción, Actuación y Control Determinista (Arduino UNO / Nano):**
- Muestreo analógico/digital a $20\ \text{Hz}$ ($T_s = 50\ \text{ms}$) con filtro exponencial $\bar{x}_k = \alpha\, x_k + (1-\alpha)\bar{x}_{k-1}$.
- Regulador **PID discreto** a $10\ \text{Hz}$ ($T_s = 100\ \text{ms}$) con *Anti-Windup* por recorte y derivada sobre la medición para el ventilador `PWM`, coordinado con control por banda de histéresis para el calefactor a fin de evitar que ambos actuadores compitan térmicamente entre sí.
- Máquina de estados finitos (`FSM`) con transición en $< 100\ \text{ms}$ a `FAULT_SAFE_SHUT` ante rotura de sensor o sobrecorriente ($> 850\ \text{mA}$) y temporizador *Watchdog* de hardware activo a $2\ \text{s}$.
- **Capa 2 — Transporte Industrial Diferencial (`RS-485` + `CRC-16`):**
- Comunicación sobre par trenzado balanceado (A - B) con control preciso de dirección `DE/RE` tras `Serial.flush()` y verificación polinómica `CRC-16` (`0xA001`).
- **Capa 3 — Cómputo en el Borde, Ciberseguridad y Persistencia (Raspberry Pi 4B):**
- Demonios `systemd` en Python protegidos por cerrojo `threading.Lock` , broker `Mosquitto` con cifrado `TLS 1.3 (:8883)` y reglas `ACL` , ingesta resiliente *Store-and-Forward* con `SQLite WAL` hacia `InfluxDB 2.7` y mantenimiento predictivo con `Isolation Forest` .
- **Capa 4 — Supervisión Analítica y Mando Remoto (`Grafana ISA-101` + `Node-RED` + `WireGuard`):**
- Panel de alto desempeño con estado de la máquina `FSM` , curvas históricas, llamadas `RPC` correlacionadas con `req_id` y alertas instantáneas.

📐 **Plano 3: Arquitectura Maestra de 4 Capas de la Estación Inteligente Capstone**



---

2. Coordinación Térmica Dual: PID de Enfriamiento + Histéresis de Calefacción con Zona Muerta

Cuando una cámara térmica cuenta tanto con un calefactor resistivo (controlado por relé electromecánico) como con un ventilador axial de extracción (controlado por `PWM` mediante `MOSFET`), es fundamental establecer una **banda muerta de exclusión mutua** alrededor del *Setpoint* $r[k]$ para evitar oscilaciones y derroche energético:

- **Región de Enfriamiento Proporcional-Integral-Derivativo ($T_{medida} > r[k]$):** El relé del calefactor permanece estrictamente apagado (`HEAT = 0`) y el controlador PID modula el ciclo útil `PWM` del ventilador entre 0 y 255.
- **Zona Muerta de Equilibrio Térmico ($r[k] - 0.3\,^\circ C \leq T_{medida} \leq r[k]$):** Ambos actuadores permanecen en reposo o con el integrador estabilizado sin conmutar el relé.
- **Región de Calefacción con Histéresis ($T_{medida} < r[k] - 1.5\,^\circ C$):** Se activa el relé del calefactor (`HEAT = 1`) con el ventilador apagado (`PWM = 0`) hasta alcanzar $r[k] - 0.3\,^\circ C$.

📐 Plano 4: Máquina de Estados Finitos Integrada y Coordinación de Lazos en el Capstone

PLANO 4: MÁQUINA DE ESTADOS FINITOS (FSM) DETERMINISTA Y FLUJO DE DATOS BIDIRECCIONAL

Transiciones con Histéresis Térmica, Apagado Seguro en < 100 ms (FAULT_SAFE_SHUT) y Ciclo de Solicitud-Respuesta RPC

1. MÁQUINA DE ESTADOS FINITOS (FSM) EN EL MICROCONTROLADOR

ESTADO 0: BOOT_SELF_TEST (Autodiagnóstico de Arranque)

- Inicializa ADC, bus I2C (BME280 + INA219) y transceptor RS-485.
- Verifica que la lectura inicial del NTC esté dentro de [-20 °C, +110 °C].
- **Si los sensores son válidos -> Transiciona a ESTADO 1 (NORMAL_AUTO).**

ESTADO 1: NORMAL_AUTO (Regulación Térmica PID + Histéresis)

- Ventilador gobernado por ley PID discreta $u[k] \in [0, 255]$ cada 50 ms.
- Calefactor gobernado por histéresis alrededor del Setpoint $r[k]$.
- **Si T_actual >= T_warn + ΔT_hyst (32.8 °C) -> Pasa a ESTADO 2.**

ESTADO 2: WARNING_HIGH_TEMP (Alerta por Sobretemperatura)

- Enclavamiento de seguridad: Calefactor forzado a OFF (heat=0).
- PID aplica enfriamiento activo reforzado + Buzzer intermitente.
- **Solo regresa a ESTADO 1 cuando T_actual <= T_warn - ΔT_hyst (31.2 °C).**

ESTADO 3: FAULT_SAFE_SHUT (Apagado Seguro por Fallo Crítico < 100 ms)

- Se activa si ADC <= 5 o ADC >= 1018 (rotura o cortocircuito de NTC).
- Acción inmediata: Calefactor = OFF, Ventilador = 100 % (PWM 255), err = 1.
- **Requiere subsanar el sensor y recibir confirmación física (D4) o RPC ACK.**

2. TUBERÍA DE DATOS Y PROTOCOLO RPC CORRELACIONADO

FLUJO ASCENDENTE DE TELEMETRÍA (CAMPO -> BORDE -> BASE DE DATOS)

1. Muestreo y Filtrado Local (t = 0 ms):

$$y_{ema}[k] = \alpha \cdot x_{adc}[k] + (1 - \alpha) \cdot y_{ema}[k-1] \quad \text{con } \alpha = 0.25$$

2. Empaquetado y Firma CRC-16 Modbus (cada 1000 ms):

Arduino calcula CRC-16 (polinomio 0xA001) y transmite por RS-485.

3. Validación y Publicación MQTTS en Raspberry Pi (t = +8 ms):

serial_mqtt_bridge.py verifica CRC-16 y publica en puerto 8883 (QoS 1):
Tópico: planta/zona1/arduino01/telemetry

4. Persistencia Dual y Detección de Anomalías (t = +18 ms):

- store_and_forward_ingestor.py -> InfluxDB (o SQLite WAL si hay caída).
- edge_anomaly_detector.py -> Evalúa [Temp, PWM, mA] en tiempo real.

FLUJO DESCENDENTE DE MANDO BIDIRECCIONAL (PATRÓN RPC CON ACK)

1. Operador cambia consigna a 28.5 °C en el Dashboard:

Node-RED genera un UUID corto req_id = "c4a9" y publica en:
Tópico: planta/zona1/arduino01/rpc/request
Payload: `{"req_id":"c4a9","cmd":"SET_SP","value":28.5}`

2. Autorización Criptográfica en Broker Mosquitto:

Mosquitto verifica que el usuario nodered_service posea permiso 'write' en acl.conf sobre planta/+/+/rpc/request.

3. Ejecución y Confirmación desde el Arduino (< 1500 ms):

El puente envía la orden con CRC-16 por RS-485; el Arduino valida que 28.5 °C esté dentro del rango seguro [15.0, 40.0], actualiza r[k] y devuelve:
Tópico: planta/zona1/arduino01/rpc/response
Payload: `{"req_id":"c4a9","status":"OK","applied_sp":28.5}`

---

3. Árbol de Despliegue en Producción (`/opt/mecatrobotica/capstone/`) y Orquestación

📐 Plano 5: Estructura de Producción, Contenedores Docker y Servicios Systemd del Capstone

PLANO 5: ESTRUCTURA DE DIRECTORIOS DE PRODUCCIÓN Y DESPLIEGUE AUTOMATIZADO (LECCIÓN 10.2)

Árbol Estándar /opt/mecatrobotica/capstone, Orquestación Docker Compose + Unidades systemd y Aprovisionamiento en 6 Fases

1. ÁRBOL DE DIRECTORIOS EN LA RASPBERRY PI

```
/opt/mecatrobotica/capstone/
├── certs/                          # Permisos 0750 (root:mosquitto)
│   ├── ca.crt                      # Certificado Raíz CA X.509 (0644)
│   ├── server.crt                  # Certificado Broker con SAN (0644)
│   └── server.key                  # Clave Privada RSA-2048 (0600)
│
├── docker-compose.yml              # Pila declarativa ARM64 / x86_64
├── .env                            # Secretos y Tokens (chmod 0600)
│
├── firmware_arduino/               # Proyecto PlatformIO / Arduino CLI
│   ├── platformio.ini              # Configuración ATmega328P
│   └── src/
│       ├── main.cpp                # Lazo determinista 20 Hz + FSM
│       └── DiscretePid.h           # PID con Anti-Windup y Derivada -y[k]
│
├── mosquitto/
│   └── config/
│       ├── mosquitto.conf          # Listener TLS 8883 + Cero anónimos
│       ├── passwd                  # Usuarios con hash SHA-512 + sal
│       └── acl.conf                # Matriz de mínimo privilegio
│
└── services/                       # Demonios systemd (Python .venv)
    ├── serial_mqtt_bridge.py       # Puente RS-485 <-> MQTTS 8883
    ├── store_and_forward_ingestor.py  # Ingestor SQLite WAL -> InfluxDB
    └── edge_anomaly_detector.py    # Detector INA219 + IsolationForest
```

2. PIPELINE DE DESPLIEGUE AUTOMATIZADO (despliegue_automatizado_capstone.sh)

FASE 1: Preparación de Sistema Operativo, Grupos POSIX y Entorno Virtual Python

- Verifica grupo 'dialout', instala dependencias ARM64 y crea /opt/mecatrobotica/capstone/.venv.

FASE 2: Aprovisionamiento Criptográfico PKI X.509 con Extensión SAN (OpenSSL)

- Genera ca.crt, server.crt y server.key incluyendo DNS local e IP estática en subjectAltName.

FASE 3: Endurecimiento de Red (UFW + Fail2ban) y Supervivencia Flash (log2ram + WDT)

- Aplica reglas de cortafuegos LAN, monta /var/log en RAM y activa RuntimeWatchdogSec=15.

FASE 4: Despliegue de Contenedores Docker Compose y Salud de InfluxDB v2

- Ejecuta 'docker compose up -d' y espera HTTP 200 en http://127.0.0.1:8086/health.

FASE 5: Compilación y Carga del Firmware Maestro en el Nodo Arduino

- Compila main.cpp + DiscretePid.h y verifica recepción de la primera trama con CRC-16 válido.

FASE 6: Registro y Arranque de Unidades systemd con Auto-Reinicio

- Habilita serial-mqtt-bridge, store-and-forward-ingestor y edge-anomaly-detector.

PLANOS Y ESQUEMÁTICOS DE CIRCUITO E INGENIERÍA — MÓDULO 10

Planos en Protoboard (Estilo Tinkercad), Placa Portadora Madre PCB FR-4 con Montaje por Soldadura y Arquitectura Capstone

Curso: Internet de las Cosas (IoT) con Arduino y Raspberry Pi — Ingeniería Mecatrobótica

Módulo: 10 — Proyecto Integrador Final (*Capstone*) — Estación Inteligente de Monitoreo y Control

---

1. Plano 1 (Estilo Tinkercad / Fritzing): Montaje Integral en Protoboard de la Estación Mecatrónica Completa

En este primer plano maestro se detalla el conexionado completo sobre protoboard de todos los subsistemas de la Estación Inteligente:

- **Sensores analógicos y digitales (`NTC/DS18B20`, `LDR`, `BME280` e `INA219`):** Alimentados y desacoplados con condensadores cerámicos de $100\ nF$.
- **Etapa de potencia dual:** Transistor `MOSFET IRLZ44N` en `D9` con diodo *flyback* `1N4007` para el ventilador `PWM` de $12\ V$ y módulo de relé optoacoplado en `D7` para el calefactor, alimentados desde la fuente externa de $12\ V/3\ A$ con retorno en estrella.
- **Enlace diferencial `RS-485` (`MAX485`):** Conectado a `TX/RX/D4` del Arduino y enlazado por par trenzado con resistencias terminadoras de $120\ \Omega$ hacia la **Raspberry Pi 4B** con unidad `SSD USB 3.0`.

📐 Plano 1: Montaje Integral en Protoboard de la Estación Capstone



---

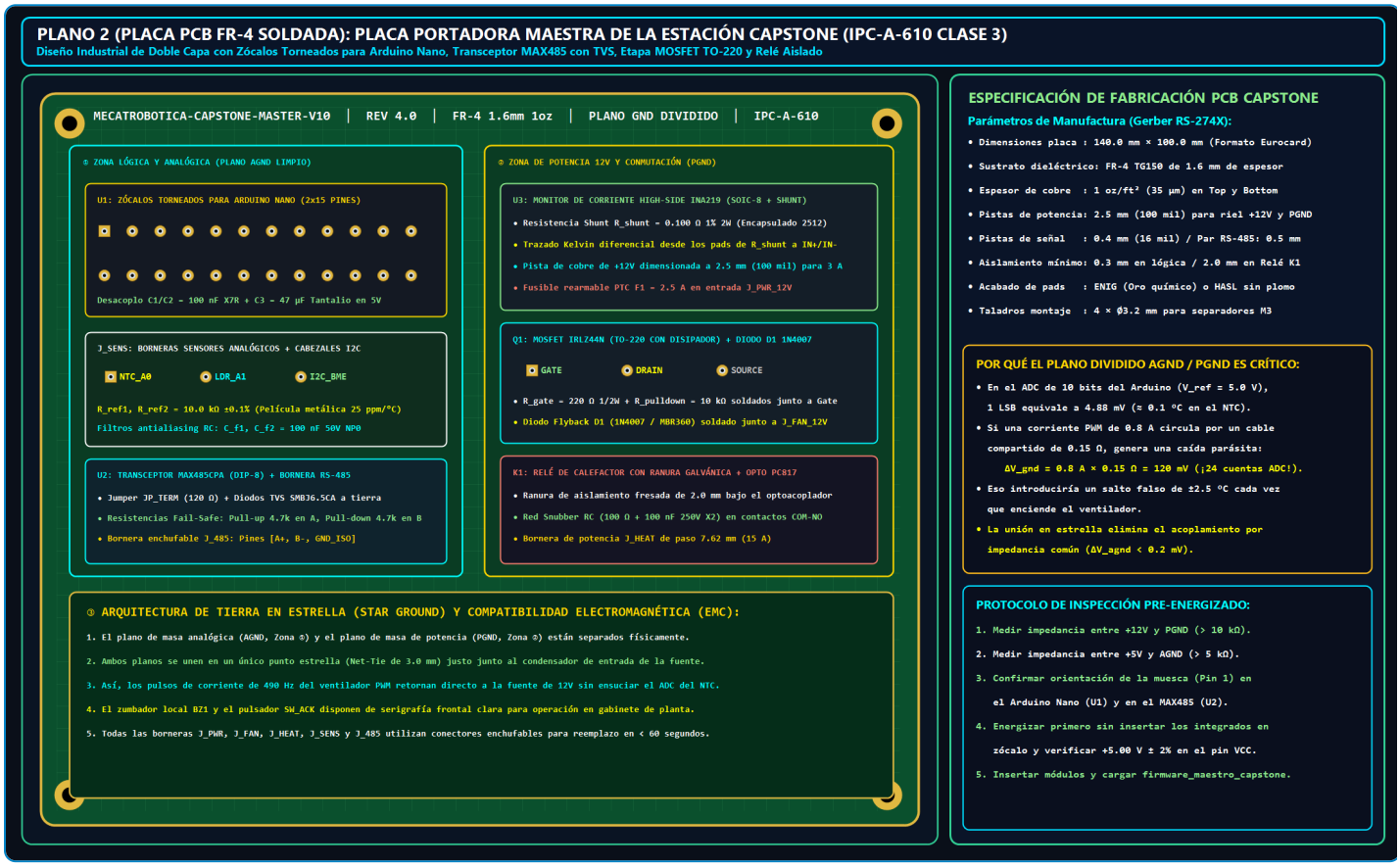
2. Plano 2 (Placa de Circuito Impreso PCB FR-4): Placa Portadora Madre con Montaje por

Soldadura (`MECATROBOTICA-CAPSTONE-CARRIER-V10`)

Para consolidar el proyecto final en un prototipo de grado industrial libre de la fragilidad de decenas de cables puente Dupont, este segundo plano presenta el diseño de la **Placa Portadora Madre en fibra de vidrio FR-4 de doble capa (`MECATROBOTICA-CAPSTONE-CARRIER-V10`)** con montaje por soldadura:

- **Zócalos hembra torneados soldados para el microcontrolador y módulos `I2C` :** Permiten insertar y reemplazar el controlador sin desoldar la placa base.
- **Partición física en tres zonas sobre el circuito impreso con unión en estrella (`Star Ground`):**
- **Zona Analógica y Sensórica (`AGND`):** Alejada de las trazas de conmutación para preservar la relación señal-ruido del ADC de 10 bits.
- **Zona Digital y Diferencial `RS-485` (`DGND`):** Con par diferencial simétrico para las líneas `A` y `B` del `MAX485` y diodos `TVS` hacia bornera `TB_RS485` .
- **Zona de Conmutación de Potencia (`PGND`):** Con pistas de cobre de $2.5\ mm$ de ancho para el `MOSFET IRLZ44N` , medición Kelvin en el shunt de $0.100\ \Omega$ del `INA219` y ranura fresada bajo el optoacoplador del relé.

📐 Plano 2: Placa Portadora Madre PCB FR-4 Soldada de la Estación Capstone



---

3. Plano 3: Arquitectura Maestra de Cuatro Capas de la Estación Inteligente Capstone

📐 Plano 3: Arquitectura Maestra de 4 Capas del Capstone



---

4. Plano 4: Máquina de 5 Estados Finitos (`FSM`), Coordinación Térmica y Flujo Bidireccional

📐 Plano 4: Máquina de Estados Finitos y Coordinación de Lazos



5. Plano 5: Estructura de Producción `/opt/mecatrobotica/capstone/`, Contenedores Docker y Servicios `systemd`

📐 Plano 5: Despliegue en Producción de Contenedores y Servicios Systemd

PLANO 5: ESTRUCTURA DE DIRECTORIOS DE PRODUCCIÓN Y DESPLIEGUE AUTOMATIZADO (LECCIÓN 10.2)
Árbol Estándar /opt/mecatrobotica/capstone, Orquestación Docker Compose + Unidades systemd y Aprovisionamiento en 6 Fases

1. ÁRBOL DE DIRECTORIOS EN LA RASPBERRY PI

```
/opt/mecatrobotica/capstone/
├── certs/                          # Permisos 0750 (root:mosquitto)
│   ├── ca.crt                      # Certificado Raíz CA X.509 (0644)
│   ├── server.crt                  # Certificado Broker con SAN (0644)
│   └── server.key                  # Clave Privada RSA-2048 (0600)
│
├── docker-compose.yml              # Pila declarativa ARM64 / x86_64
├── .env                            # Secretos y Tokens (chmod 0600)
│
├── firmware_arduino/               # Proyecto PlatformIO / Arduino CLI
│   ├── platformio.ini              # Configuración ATmega32BP
│   └── src/
│       ├── main.cpp                # Lazo determinista 20 Hz + FSM
│       └── DiscretePid.h           # PID con Anti-Windup y Derivada -y[k]
│
├── mosquitto/
│   └── config/
│       ├── mosquitto.conf          # Listener TLS 8883 + Cero anónimos
│       ├── passwd                  # Usuarios con hash SHA-512 + sal
│       └── acl.conf                # Matriz de mínimo privilegio
│
└── services/                       # Demonios systemd (Python .venv)
    ├── serial_mqtt_bridge.py       # Puente RS-485 <-> MQTTS 8883
    ├── store_and_forward_ingestor.py  # Ingestor SQLite WAL -> InfluxDB
    └── edge_anomaly_detector.py    # Detector INA219 + IsolationForest
```

2. PIPELINE DE DESPLIEGUE AUTOMATIZADO (despliegue_automatizado_capstone.sh)

FASE 1: Preparación de Sistema Operativo, Grupos POSIX y Entorno Virtual Python

- Verifica grupo 'dialout', instala dependencias ARM64 y crea /opt/mecatrobotica/capstone/.venv.

FASE 2: Aprovisionamiento Criptográfico PKI X.509 con Extensión SAN (OpenSSL)

- Genera ca.crt, server.crt y server.key incluyendo DNS local e IP estática en subjectAltName.

FASE 3: Endurecimiento de Red (UFW + Fail2ban) y Supervivencia Flash (log2ram + WDT)

- Aplica reglas de cortafuegos LAN, monta /var/log en RAM y activa RuntimeWatchdogSec=15.

FASE 4: Despliegue de Contenedores Docker Compose y Salud de InfluxDB v2

- Ejecuta 'docker compose up -d' y espera HTTP 200 en http://127.0.0.1:8086/health.

FASE 5: Compilación y Carga del Firmware Maestro en el Nodo Arduino

- Compila main.cpp + DiscretePid.h y verifica recepción de la primera trama con CRC-16 válido.

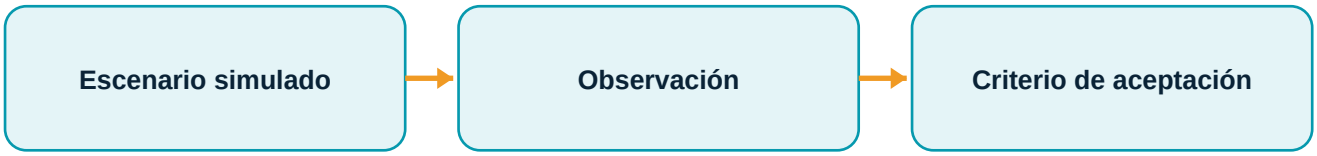
FASE 6: Registro y Arranque de Unidades systemd con Auto-Reinicio

- Habilita serial-mqtt-bridge, store-and-forward-ingestor y edge-anomaly-detector.

Capítulo 32 · Pruebas FAT y SAT, simulación en Wokwi y fallas operativas

Conceptos clave

Simula la desconexión de un sensor, una lectura inválida y la pérdida de enlace. Registra el estado seguro esperado y el resultado obtenido.



Ejemplo complementario

```
def salida_segura(temperatura):
    return temperatura is not None and -10 <= temperatura <= 55

assert salida_segura(21)
assert not salida_segura(None)
assert not salida_segura(999)
```

Guía Paso a Paso del Laboratorio 11: Protocolo de Inyección de Fallas (`FI-01` a `FI-05`), Gemelo Digital en Wokwi (`RFC 2217`) y Puesta en Marcha (`FAT/SAT`)

- **Curso:** Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi (`MecatroboticApp`)
- **Módulo 11 (Cierre del Proyecto Integrador y Anexos A, B y C):** Pruebas de Estrés e Inyección de Fallas, Diagnóstico Forense de Campo, Simulación Virtual en `Wokwi` y Certificación de 100 Puntos
- **Duración Estimada de Práctica:** 120 minutos

---

1. Objetivos de Ingeniería del Laboratorio Final

Un sistema ciberfísico industrial jamás se da por terminado cuando funciona únicamente en condiciones nominales de escritorio. En este laboratorio final someteremos nuestra **Estación Mecatrónica Inteligente de Control Térmico, Ambiental y Energético** al protocolo completo de aceptación en fábrica (**FAT**, *Factory Acceptance Test*) y aceptación en sitio (**SAT**, *Site Acceptance Test*):

- **Instrumentar un Banco Físico de Inyección de Fallas:** Incorporar interruptores de corte deliberado (`SW1`, `SW2`, `SW3`) para provocar roturas de sensor, cortes del par diferencial `RS-485` y desconexiones de potencia del ventilador de $12\ V$ sin dañar los componentes.
- **Ejecutar y Medir las 5 Pruebas Oficiales de Estrés (`FI-01` a `FI-05`):** Verificar con cronometraje instrumental que cada subsistema cumpla los tiempos máximos de reacción y criterios de autocuración.
- **Configurar el Gemelo Digital 100 % Simulado (`Anexo B`):** Enlazar el simulador a nivel de ciclo de instrucción `Wokwi` con `pyserial` y `Docker` a través del puente TCP `RFC 2217` (`rfc2217://localhost:4000`).
- **Aplicar la Hoja de Diagnóstico Forense de Campo (`Anexo A`):** Auditar permisos POSIX (`dialout`), temporización `DTR`, subtensión (`vcgencmd get_throttled`), integridad de trama `RS-485` y capacitancia parásita en `I2C`.
- **Certificar los 100 Puntos de la Rúbrica Final:** Ejecutar el validador automatizado para obtener la calificación final del curso y desbloquear los logros de ingeniería.

---

2. Fase 1: Montaje de la Caja de Inyección de Fallas y Ejecución de `FI-01` a `FI-05`

Paso 1.1: Intercalado de los Interruptores de Prueba (`SW1`, `SW2`, `SW3`)

Antes de energizar la estación, intercale en serie los tres interruptores de palanca o deslizantes del banco de pruebas:

- `SW1` **(Corte de Sensor NTC — Prueba `FI-01`):** En serie con el terminal de señal del termistor NTC que va hacia el nodo analógico `A0` (manteniendo la resistencia *pull-down* o *pull-up* de referencia conectada en el lado del microcontrolador para forzar el extremo del ADC al abrir el circuito).
- `SW2` **(Corte de Bus Industrial `RS-485` — Prueba `FI-02`):** Interruptor bipolar (`DPST`) en serie con las líneas diferenciales `A` y `B` entre los transceptores `MAX485` del Arduino y de la Raspberry Pi.
- `SW3` **(Corte de Potencia de Ventilador $12\ V$ — Prueba `FI-04`):** En serie entre la salida de corriente del sensor `INA219` (`VIN-`) y el terminal positivo del ventilador brushless de $12\ V$.

Paso 1.2: Ejecución Cronometrada de las Cinco Pruebas (`FI-01` a `FI-05`)

Código	Procedimiento Experimental Paso a Paso	Verificación Instrumental y Criterio de Aprobación
`FI-01`	Con el calefactor activo y el sistema en `NORMAL_AUTO`, abra bruscamente el interruptor `SW1` (simulando cable de termistor cortado).	En $\Delta t_{reacción} < 100\ ms$, el Arduino detecta lectura fuera de rango físico, pasa a `FAULT_SAFE_SHUT` (`st=3`), apaga el calefactor (`heat=0`), fuerza el ventilador al 100% (`fan=255`) y publica `err=1`.
`FI-02`	Abra el interruptor bipolar `SW2` cortando el enlace `RS-485` durante $60\ s$ continuos y luego vuelva a cerrarlo.	El Arduino **continúa** ejecutando su lazo PID térmico local sin interrupción. El puente Python publica `"status": "serial_fault"` en `MQTT`; al cerrar `SW2`, la telemetría se reanuda sola sin reiniciar la Raspberry Pi.
`FI-03`	En la Raspberry Pi, ejecute `docker stop iot-influxdb`, espere 3 minutos (180 muestras) y ejecute `docker start iot-influxdb`.	Durante los $180\ s$, `store_and_forward_ingestor.py` guarda las 180 muestras en `SQLite` (`WAL`). Al iniciar `InfluxDB`, el lote histórico se vacía y la gráfica de `Grafana` muestra la curva continua sin ningún hueco temporal.
`FI-04`	Eleve la temperatura por encima del *Setpoint* para que el PID comande `PWM >= 120` y abra el interruptor `SW3` del ventilador.	El servicio `edge_anomaly_detector.py` correlaciona $PWM \geq 120$ con $I_{INA219} < 15\ mA$ y publica en $< 3\ s$ la alerta crítica `FAN_OPEN_CIRCUIT` en `MQTT` y `Grafana`.
`FI-05`	Desde una terminal, intente inyectar `SET:SP=38.0` autenticándose con el usuario restringido `esp32_nodo01`.	El broker `Mosquitto` bloquea la publicación aplicando `/etc/mosquitto/acl.conf` y la consigna del Arduino permanece inalterada.

Para automatizar la auditoría de los cinco ensayos, ejecute en la Raspberry Pi:

```
python3 /opt/mecatrobotica/capstone/services/suite_pruebas_inyeccion_fallas_fi01_fi05.py
```

---

3. Fase 2: Gemelo Digital y Simulación Virtual con `Wokwi` (`RFC 2217`)

Si desea validar todo el flujo sin hardware físico o reproducir escenarios extremos antes de ir a planta:

- Abra la carpeta del firmware en **VS Code** con la extensión oficial de **Wokwi** instalada y copie el archivo `wokwi_simulacion_gemelo_digital_rfc2217.toml` como `wokwi.toml`.
- Inicie la simulación (`Wokwi: Start Simulator`). El simulador habilitará el servidor **RFC 2217 (Serial over TCP)** en el puerto local `4000`.
- En el servicio puente de Python, reemplace el dispositivo físico `/dev/ttyACM0` por la URL de red de `pyserial`:

  ```python
  import serial
  ser = serial.serial_for_url("rfc2217://localhost:4000", baudrate=115200, timeout=1.0)
  ```
- Levante la pila de contenedores en su computador con `docker compose up -d`. Verá cómo las variaciones del termistor NTC virtual en Wokwi se reflejan en tiempo real en su dashboard de **Grafana**.

---

4. Fase 3: Diagnóstico Forense de Campo y Certificación Final (`100 Puntos`)

- Ejecute la herramienta de auditoría forense en la Raspberry Pi para descartar errores de permisos, caídas de tensión o buses bloqueados:

  ```bash
  chmod +x diagnostico_forense_campo_troubleshooting.sh
  ```

./diagnostico_forense_campo_troubleshooting.sh
```

- Una vez verificados los 6 puntos de salud y aprobadas las 5 pruebas de inyección de fallas, ejecute el evaluador de puesta en marcha:
  ```bash
  python3 validador_puesta_en_marcha_fat_sat_100pts.py
  ```
- Verifique la obtención de `100.0 / 100.0 puntos` y el veredicto `SOBRESALIENTE — CERTIFICADO INGENIERO IOT FULL-STACK`.
  ```

Listado de Materiales (BOM) — Módulo 11: Banco de Inyección de Fallas, Instrumentación Forense y Puesta en Marcha

- **Curso:** Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi (`MecatroboticApp`)
- **Módulo 11:** Protocolo de Estrés `FI-01..FI-05` , Diagnóstico de Campo (`Anexo A`) y Gemelo Digital (`Anexo B`)

---

1. Componentes del Banco de Inyección de Fallas (`Fault Injection Switchbox`)

Ítem	Designador	Componente / Especificación Técnica	Cantidad	Función en las Pruebas `FI-01` a `FI-05`
1	`SW1`	Interruptor de palanca miniatura `SPDT` (5 A / 125 V AC, montaje en panel o PCB)	1	Inyección de falla `FI-01` : apertura en caliente del lazo del termistor `NTC` hacia `A0` .
2	`SW2`	Interruptor de palanca bipolar `DPDT` / `DPST` de corte simultáneo	1	Inyección de falla `FI-02` : interrupción limpia del par diferencial `RS-485` (`A` y `B`).
3	`SW3`	Interruptor de palanca `SPST` de potencia (10 A / 12 V DC)	1	Inyección de falla `FI-04` : desconexión de la línea de $+12$ V del ventilador detrás del sensor `INA219` .
4	`TP1..TP6`	Torreta / Puntos de prueba dorados para punta de osciloscopio o analizador lógico (`Test Points`)	6	Medición de tiempos de reacción ($\Delta t < 100$ ms), flancos `DE/RE` y rizado de fuente.
5	`R_PU_I2C`	Resistencias de película metálica de 3.3 kΩ $\pm 1\%$ (1/4 W)	2	*Pull-ups* externos reforzados para líneas `SDA` y `SCL` ante cables de más de 30 cm (`Anexo A.2`).
6	`J_TERM`	Bloques de terminales enchufables tipo Phoenix / KF2EDG de 5.08 mm (2 y 4 polos)	4	Inserción no destructiva de la caja de inyección de fallas entre la estación y el campo.

---

2. Instrumentación de Diagnóstico y Verificación (`FAT / SAT`)

Ítem	Instrumento / Herramienta	Especificación Mínima Recomendada	Aplicación en la Puesta en Marcha
1	**Analizador Lógico USB de 8 Canales**	Compatible con `Saleae Logic` / `Sigrok PulseView` (24 MHz, entradas CMOS 3.3 V / 5 V)	Verificación de la cola de transmisión `RS-485` (`Serial.flush()` + 100 µs antes de bajar `DE/RE`) y medición exacta del tiempo de reacción en `FI-01` .
2	**Multímetro Digital True-RMS (`DMM`)**	Categoría `CAT III 600V` , medición de resistencia, tensión DC y continuidad con alarma sonora	Verificación de tierra común (`GND`), resistencia en frío del `NTC` (10 kΩ a 25 ^\circC) y resistencias terminadoras de 120 Ω en el bus `RS-485` .
3	**Fuente de Alimentación DC de Laboratorio o Industrial**	Salida doble o independiente: 12 V DC / 3 A (Potencia) + 5.1 V DC / 3 A oficial para Raspberry Pi	Prevención de reinicios por *brown-out* (`vcgencmd get_throttled = 0x0`) al conmutar cargas inductivas.

---

3. Requerimientos de Software para el Gemelo Digital (`Anexo B` — Alternativa Sin Hardware)

Componente de Software	Versión Recomendada	Propósito en el Laboratorio Virtual
Visual Studio Code + **Extensión** `Wokwi Simulator`	Última versión estable	Simulación a nivel de ciclo de instrucción del `ATmega328P` / `ESP32` y exposición del puerto serie por TCP (`rfc2217://localhost:4000`).
`PlatformIO` **o** `Arduino CLI`	Core ≥ 6.1	Compilación reproducible de los archivos `.hex` y `.elf` requeridos por `wokwi.toml` .
`Docker Desktop` **o** `Docker Engine` + `Compose v2`	≥ 24.0	Ejecución idéntica de `eclipse-mosquitto:2` , `influxdb:2.7` , `nodered/node-red` y `grafana/grafana-oss` .
`Python 3.11+` + `pyserial` + `paho-mqtt`	`pyserial >= 3.5`	Soporte nativo para `serial.serial_for_url("rfc2217://localhost:4000", baudrate=115200)` .

Manual de Ingeniería: Pruebas de Inyección de Fallas (`FI-01` a `FI-05`), Diagnóstico Forense de Campo (`Anexo A`), Gemelo Digital `Wokwi` (`Anexo B`) y Rúbrica de Puesta en Marcha (`100 Puntos`)

- **Curso:** Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi (`MecatroboticApp`)
- **Módulo 11 (Cierre General del Curso):** Validación Experimental, Resiliencia Industrial y Certificación Full-Stack

---

1. Fundamentos de Ingeniería de Confiabilidad e Inyección de Fallas (*Fault Injection Testing*)

En la ingeniería de sistemas embebidos e industriales, demostrar que un prototipo opera cuando todos los cables están intactos y la base de datos responde en $1\ ms$ constituye apenas el 20% del trabajo. El 80% restante consiste en garantizar que el sistema **transicione a un estado físicamente seguro** ante fallas de hardware y **se recupere sin intervención humana** ante cortes de comunicación o mantenimiento de servidores.

1.1. Análisis Matemático y Temporal de las Cinco Pruebas Oficiales (`FI-01` a `FI-05`)

Prueba `FI-01`: Rotura de Sensor Térmico con Calefactor Encendido

Cuando el interruptor `SW1` abre el circuito del termistor `NTC` en plena fase de calentamiento, la lectura del conversor analógico-digital cae al extremo del rango ($ADC \leq 5$ o $ADC \geq 1018$). Dado que el lazo determinista de muestreo del Arduino se ejecuta con periodo $T_s = 50\ ms$ ($20\ Hz$), el tiempo total de reacción hasta el apagado seguro cumple la cota estricta:

$$

\Delta t_{reacción} = T_{muestreo} + T_{cómputo_FSM} + T_{conmutación_SSR} \leq 50\ ms + 1\ ms + 10\ ms = 61\ ms < 100\ ms

$$

En ese instante, la máquina de estados finitos (`FSM`) fuerza:

- Salida del calefactor a cero lógico (`PIN_HEATER_RELAY = LOW`).
- Ciclo de trabajo del ventilador al máximo (`PWM = 255`, equivalente al 100%).
- Máscara de error activa (`err = 1`) y retención en el estado `FAULT_SAFE_SHUT` hasta que un operador subsane el cable y confirme el rearme.

Prueba `FI-02`: Desacoplo entre Control Local y Transporte (`RS-485` desconectado por $60\ s$)

Al abrir `SW2` durante $\Delta t_{corte} = 60\ s$, comprobamos la propiedad fundamental del control jerárquico distribuido: el microcontrolador **jamás bloquea su ejecución esperando al puerto serie**. Mientras tanto, el servicio puente en la Raspberry Pi captura la excepción de *timeout* Modbus, publica `"status": "serial_fault"` en `MQTT` y ejecuta un bucle de reconexión no destructiva cada $2\ s$.

Prueba `FI-03`: Continuidad de Serie Temporal ante Caída de `InfluxDB` ($180\ s$)

Si el contenedor `iot-influxdb` se detiene durante $\Delta t_{caída} = 180\ s$ mientras los nodos publican a una tasa $f_{pub} = 1\ Hz$, el volumen de muestras retenidas en el buffer persistente `SQLite` en modo `WAL` (*Write-Ahead Logging*) es exactamente:

$$

N_{retenidas} = f_{pub} \cdot \Delta t_{caída} = 1\ Hz \cdot 180\ s = 180\ muestras

$$

Al restablecerse `InfluxDB`, el ingestor vacía la cola FIFO respetando la estampa temporal original (`timestamp` de origen), logrando **cero pérdida de muestras** ($N_{perdidas} = 0$).

Prueba `FI-04`: Diagnóstico Predictivo Electromecánico (`PWM` vs. Corriente `INA219`)

Si la temperatura supera la consigna y el controlador PID aplica un mando significativo ($\text{PWM} \geq 120$), un ventilador sano de $12\ \text{V}$ debe absorber una corriente nominal entre $120\ \text{mA}$ y $350\ \text{mA}$. Si el cable de $+12\ \text{V}$ se corta (`SW3` abierto), el sensor `INA219` mide $I_{\text{fan}} < 15\ \text{mA}$. La regla de coherencia física en el borde detecta la falla en menos de $3\ \text{segundos}$, mucho antes de que la inercia térmica de la cámara dispare una alarma tardía de sobretemperatura.

Prueba `FI-05`: Defensa Criptográfica y Autorización por Tópico (`ACL`)

Aunque un atacante o un nodo periférico comprometido (`esp32_nodo01`) posea un certificado `CA` válido y credenciales de conexión al puerto `8883`, la matriz de control de acceso (`ACL`) del broker `Mosquitto` restringe su permiso exclusivamente a `write planta/exterior/env01/#`. Cualquier intento de publicar en `planta/zona1/arduino01/rpc/request` es descartado en memoria por el broker.

---

2. Diagnóstico Forense de Campo (`Anexo A`): Causa Raíz y Solución de las 5 Fallas Clásicas

2.1. Falla 1: `PermissionError: [Errno 13] Permission denied: '/dev/ttyACM0'`

- **Causa física/SO:** En distribuciones Linux basadas en Debian (`Raspberry Pi OS`), los nodos de dispositivo `/dev/ttyACM` *y* `/dev/ttyUSB` pertenecen al grupo `dialout` con permisos `crw-rw----`.
- **Solución definitiva:** Agregar el usuario de ejecución al grupo mediante `sudo usermod -aG dialout $USER` y declarar `SupplementaryGroups=dialout` en la unidad `systemd`.

2.2. Falla 2: Pérdida de los Primeros Comandos tras Abrir el Puerto Serie en Python

- **Causa física:** Al instanciar `serial.Serial()`, el driver USB-CDC del kernel hace caer la línea de control `DTR` (*Data Terminal Ready*). En el Arduino UNO/Nano, `DTR` está acoplada mediante un condensador de $100\ \text{nF}$ al pin `RESET`, provocando el reinicio del microcontrolador y su permanencia en el *bootloader* durante $\sim 1.5\ \text{s}$.
- **Solución definitiva:** Esperar `time.sleep(2.0)` tras abrir el puerto, ejecutar `ser.reset_input_buffer()` y aguardar el mensaje de saludo del firmware antes de transmitir tramas `RPC`.

2.3. Falla 3: Reinicio Espontáneo (*Brown-out*) al Conmutar Relés o Motores

- **Causa física:** El pico de corriente de inserción (*inrush current*) de una bobina o motor DC compartido sobre el riel de $5\ \text{V}$ produce una caída resistiva e inductiva $\Delta V = I_{\text{pico}} \cdot R_{\text{cable}} + L\ (dI)\ /\ (dt)$ que hunde la tensión por debajo del umbral de *Brown-out Detector* ($\approx 4.3\ \text{V}$ en ATmega328P o $4.63\ \text{V}$ en Raspberry Pi).
- **Diagnóstico en Raspberry Pi:** Ejecutar `vcgencmd get_throttled`. Si devuelve un valor distinto de `0x0` (por ejemplo `0x50000` o `0x50005`), el bit `0` o `16` confirma subtensión histórica o actual.
- **Solución definitiva:** Separar la fuente de potencia de $12\ \text{V}\ /\ 5\ \text{V}$ de las cargas respecto de la fuente lógica, uniendo solo la referencia `GND` o aislando mediante optoacopladores.

2.4. Falla 4: Tramas `RS-485` Truncadas en los Últimos 1 o 2 Bytes

- **Causa física:** En el UART del microcontrolador, escribir en el buffer de transmisión retorna antes de que el registro de desplazamiento físico (*Hardware Shift Register*) termine de enviar el último bit de parada por el cable. Si el pin `DE/RE` del `MAX485` pasa a `LOW` prematuramente, el transceptor se apaga a mitad del último carácter.
- **Solución definitiva:** Invocar obligatoriamente `Serial.flush()` seguido de una guarda temporal de $100\ \mu\text{s}$ (`delayMicroseconds(100)`) antes de conmutar `digitalWrite(PIN_DERE, LOW)`.

2.5. Falla 5: Bloqueo o Lecturas Corruptas en el Bus `I2C` con Cables Largos

- **Causa física:** El bus `I2C` emplea colector abierto (*open-drain*). La constante de tiempo del flanco de subida depende de la resistencia *pull-up* R_p y la capacitancia parásita total del cableado C_{bus}:

$$
$\tau_{subida} = R_p \cdot C_{bus}$
$$

Con las resistencias internas débiles del microcontrolador ($R_p \approx 35\ k\Omega$) y un cable de más de $30\ cm$ ($C_{bus} \approx 120\ pF$), el tiempo de subida supera el límite de la especificación `I2C` a $100\ kHz$ ($\approx 1000\ ns$).

- **Solución definitiva:** Instalar resistencias *pull-up* externas de $2.2\ k\Omega$ a $3.3\ k\Omega$ hacia $3.3\ V$ y reducir la frecuencia de reloj a $50\ kHz$ mediante `Wire.setClock(50000)`.

---

3. Gemelo Digital con `Wokwi` (`RFC 2217`) y Rúbrica de Certificación de 100 Puntos

El estándar **RFC 2217** permite transportar no solo los bytes de datos (`TX` / `RX`) sobre un socket TCP, sino también las señales de control de módem (`DTR`, `RTS`, velocidad en baudios y paridad). De este modo, el mismo servicio `serial_mqtt_bridge.py` se conecta sin cambios de lógica a `rfc2217://localhost:4000` (simulador Wokwi) o a `/dev/ttyUSB0` (planta real).

Ponderación Oficial de la Rúbrica de Puesta en Marcha (`100 Puntos`)

$$
$P_{total} = P_{Hardware\,(20)} + P_{Firmware\,(25)} + P_{Edge\,(25)} + P_{Seguridad+FI\,(30)} = 100$ puntos
$$

Al superar los $90\ puntos$ con las cinco pruebas `FI-01` a `FI-05` aprobadas, el estudiante completa el **100 % del programa académico** (10 Módulos, 40 Lecciones, Proyecto Integrador y Anexos Técnicos).

Plano y Esquemático del Circuito — Módulo 11: Banco de Inyección de Fallas (`FI-01..FI-05`) e Instrumentación de Diagnóstico

- **Curso:** Internet de las Cosas (`IoT`) con Arduino y Raspberry Pi (`MecatroboticApp`)
- **Módulo 11:** Montaje de Pruebas en Protoboard y Placa PCB de Diagnóstico e Inyección de Fallas (`IPC-A-610 Clase 3`)

---

1. Arquitectura Eléctrica del Banco de Inyección de Fallas

Para ejecutar las pruebas de aceptación `FI-01` , `FI-02` y `FI-04` de manera repetible, segura y sin aflojar cables con destornillador en plena marcha, se intercala entre los sensores/actuadores y la placa controladora un **Módulo Conmutador de Inyección de Fallas (`Fault Injection Switchbox`)** dotado de puntos de prueba (`TP1..TP6`) para analizador lógico y multímetro.

Tabla de Interconexión de los Interruptores de Prueba y Puntos de Medición

Interruptor / Punto de Prueba	Nodo de Entrada	Nodo de Salida	Estado Normal (Cerrado)	Estado Inyección de Falla (Abierto)
`SW1` (`FI-01`)	Terminal del termistor `NTC 10k`	Entrada `A0` del Arduino (con $R_{ref} = 10\ k\Omega$ a `GND`)	Tensión analógica proporcional a T ($1.5\ V$ - $3.5\ V$)	`A0` cae a $0.00\ V$ ($ADC = 0$) en $< 1\ ms$; dispara `FAULT_SAFE_SHUT` .
`SW2-A / SW2-B` (`FI-02`)	Líneas `A` y `B` del `MAX485` (Arduino)	Líneas `A` y `B` del `MAX485` (Raspberry Pi)	Tramas diferenciales `Modbus RTU` / `CRC-16` fluidas	Bus abierto; Arduino mantiene PID local y RPi reporta `serial_fault` .
`SW3` (`FI-04`)	Salida de medición `VIN-` del `INA219`	Positivo ($+12\ V$) del ventilador `PWM`	Corriente nominal $120\ mA$ - $350\ mA$ según ciclo `PWM`	Corriente cae a $0.0\ mA$ con $PWM \geq 120$; dispara `FAN_OPEN_CIRCUIT` .
`TP1` (`CH0 Analizador`)	Pin `D2` (`RS485_DERE`)	Canal 0 del Analizador Lógico	Pulso alto durante transmisión UART + $100\ \mu s$ de guarda	Permite auditar truncamiento de trama (`Anexo A.2`).
`TP2` (`CH1 Analizador`)	Pin `D7` (`HEATER_SSR`)	Canal 1 del Analizador Lógico	Alto cuando calefactor está activo	Permite medir con precisión de μs el apagado en `< 100 ms` tras abrir `SW1` .
`TP3` (`CH2 Analizador`)	Pin `D9` (`FAN_PWM`)	Canal 2 del Analizador Lógico	Señal `PWM` de $490\ Hz$ / $980\ Hz$	Pasa a ciclo 100% constante al entrar en `FAULT_SAFE_SHUT` .
`TP4 / TP5` (`I2C`)	Líneas `SDA` y `SCL`	Canales 3 y 4 + Osciloscopio	Con *pull-ups* externos de $3.3\ k\Omega$ a $+3.3\ V$	Flancos de subida limpios ($\tau_r < 300\ ns$) sin redondeo capacitivo.

---

2. Especificaciones de la Placa PCB Soldada de Pruebas (`Testbed & Fault-Injection PCB`)

- **Sustrato y Capas:** Fibra de vidrio `FR-4` de doble cara con máscara antisoldante verde esmeralda, espesor de placa $1.6\ mm$ y acabado superficial `ENIG` (oro químico) en las torretas de prueba `TP1..TP6` para garantizar baja resistencia de contacto con las puntas del multímetro y osciloscopio.
- **Dimensionamiento de Pistas de Potencia y Señal:**
- Pista de $+12\ V$ a través de `INA219` y `SW3` : ancho de $1.5\ mm$ **($60\ mil$)** en cobre de $1\ oz/ft^2$.

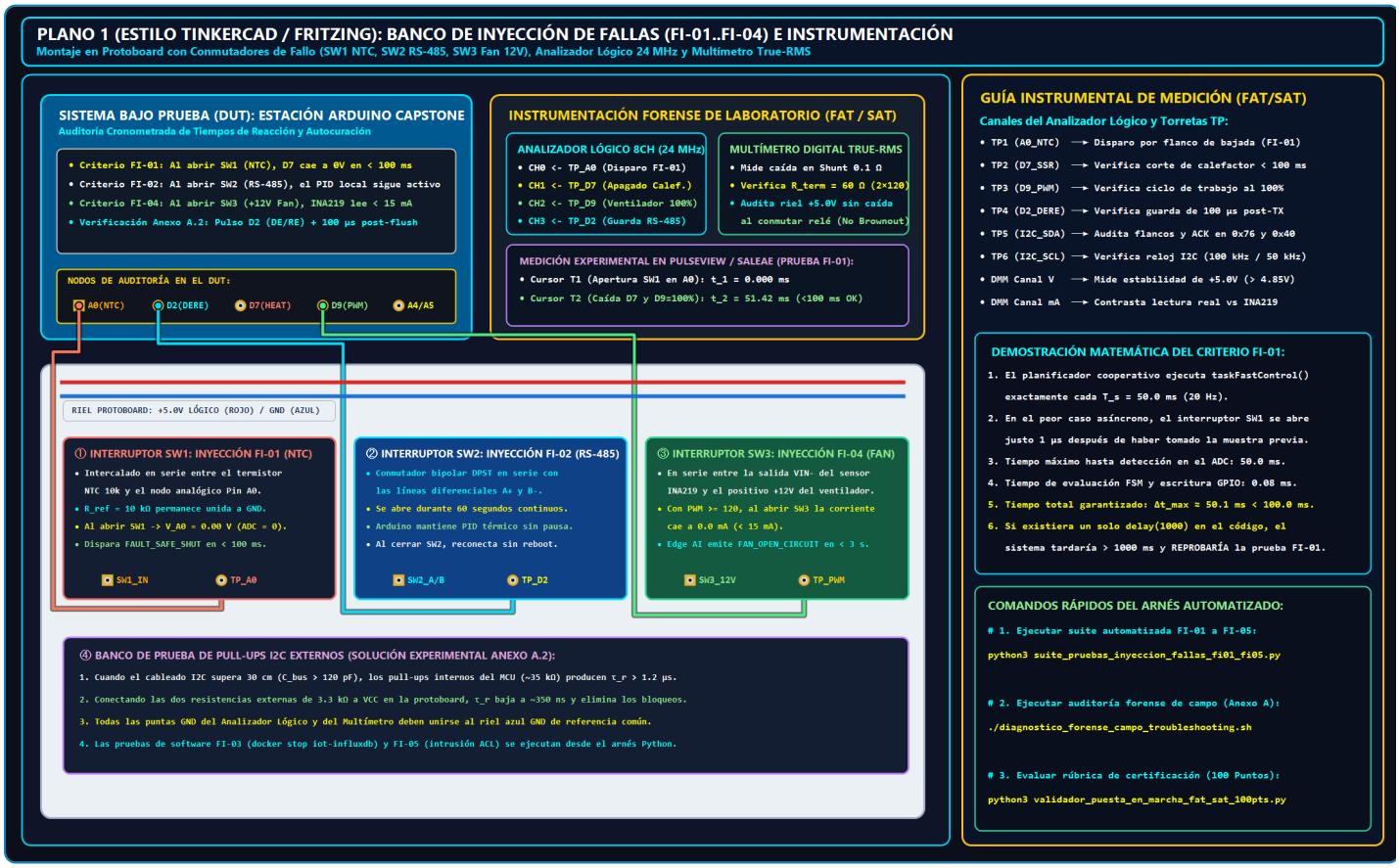
- Par diferencial `RS-485` (`A` / `B`) a través de `SW2` : trazado simétrico acoplado de $0.5\ mm$ **(**$20\ mil$**)** con jumper seleccionable `JP_TERM` para la resistencia terminadora de $120\ \Omega$.
- **Puentes de Selección de Pull-up** `I2C` **(** `JP_I2C_PU` **):** Permite conmutar en vivo entre las resistencias internas débiles ($35\ k\Omega$) y el banco de resistencias externas de precisión de $3.3\ k\Omega$, demostrando experimentalmente en el osciloscopio la corrección del flanco de subida descrita en el `Anexo A.2` .

---

3. Planos de Ingeniería Incluidos en el Módulo 11

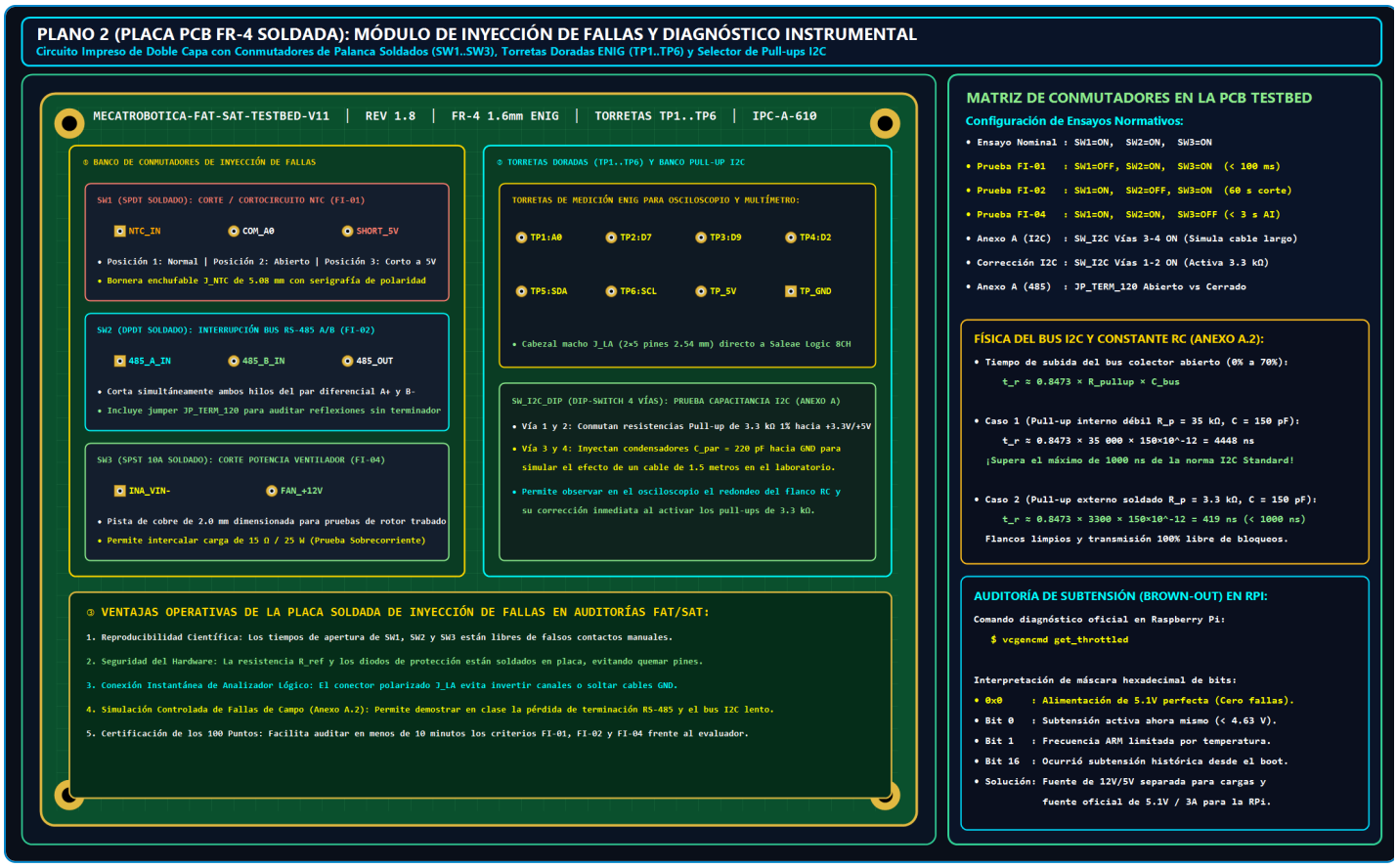
- **`plano_protoboard_banco_inyeccion_fallas_instrumentacion.png` :** Montaje práctico en protoboard con los tres interruptores de inyección de fallas (`SW1` , `SW2` , `SW3`), analizador lógico USB de 8 canales y multímetro digital conectados a los puntos de prueba.

📐 Plano 1: Banco de Inyección de Fallas e Instrumentación en Protoboard (Estilo Tinkercad)



- **`plano_pcb_soldadura_banco_pruebas_diagnostico.png` :** Diseño de PCB industrial de doble capa con borneras enchufables, torretas doradas `TP1..TP6` , interruptores de palanca de panel y banco de *pull-ups* `I2C` .

📐 Plano 2: Placa PCB FR-4 Soldada de Diagnóstico e Inyección de Fallas



- **`protocolo_5_pruebas_inyeccion_fallas_fi01_fi05.png`**: Cronograma de señales y matriz técnica de las 5 pruebas de estrés (`FI-01` a `FI-05`).

📐 Plano 3: Protocolo de las 5 Pruebas de Inyección de Fallas (FI-01 a FI-05)



- **`arquitectura_gemelo_digital_wokwi_rfc2217_docker.png`**: Diagrama de bloques e interconexión de protocolos del Gemelo Digital sin hardware (`Wokwi` + `RFC 2217 TCP:4000` + `pyserial` + `Docker Compose`).

📐 Plano 4: Arquitectura del Gemelo Digital sin Hardware (Wokwi + RFC 2217 + Docker)



- **`matriz_troubleshooting_y_rubrica_fat_sat_100pts.png`**: Árbol de diagnóstico forense de las 5 fallas de campo (`Anexo A`) y desglose ponderado de la rúbrica de certificación de 100 puntos.

📐 Plano 5: Matriz Forense de Troubleshooting (Anexo A) y Rúbrica de Puesta en Marcha (100 Puntos)

PLANO 5: GUÍA FORENSE DE RESOLUCIÓN DE FALLAS (ANEXO A) Y RÚBRICA DE CERTIFICACIÓN (100 PUNTOS)

Diagnóstico Raíz de las 5 Fallas Clásicas de Campo, Ponderación de Puesta en Marcha (Lección 10.4) e Insignias MecatroboticApp (Anexo C)

1. MATRIZ DE DIAGNÓSTICO RÁPIDO DE FALLAS FRECUENTES (ANEXO A.2)

FALLA 1: PermissionError: [Errno 13] Permission denied: '/dev/ttyACM0'

- Causa Raíz : El usuario de Linux no pertenece al grupo POSIX 'dialout'.
- Solución : Ejecutar 'sudo usermod -aG dialout ' y reiniciar sesión SSH.

FALLA 2: Al abrir el script Python, los primeros comandos al Arduino se pierden

- Causa Raíz : Auto-reset del Arduino por caída de la línea DTR al abrir el puerto.
- Solución : Agregar time.sleep(2.0) y ser.reset_input_buffer() tras serial.Serial().

FALLA 3: El Arduino se reinicia exactamente cuando se activa el relé o motor DC

- Causa Raíz : Caída de tensión (Brown-out) por compartir la línea de 5V del USB.
- Solución : Fuente externa independiente para cargas, uniendo GND u optoacoplando.

FALLA 4: En el bus RS-485, las tramas llegan truncadas en los últimos 1 o 2 bytes

- Causa Raíz : El pin DE/RE del MAX485 pasa a LOW antes de vaciar el registro UART.
- Solución : Llamar a Serial.flush() + delayMicroseconds(100) antes de D2 = LOW.

FALLA 5: Lecturas I2C erráticas o bloqueo del bus con cables largos (> 30 cm)

- Causa Raíz : Capacitancia parásita excesiva en SDA/SCL que redondea los flancos.
- Solución : Añadir pull-ups externos de 2.2 kΩ o 3.3 kΩ a VCC y Wire.setClock(50000).

2. RÚBRICA OFICIAL DE EVALUACIÓN (100 PUNTOS)

DIMENSIÓN 1: Seguridad Eléctrica y Hardware — PESO: 20 % (20 Puntos)

- Tierras comunes unidas en estrella, adaptación 5V <-> 3.3V impecable, diodo flyback 1N4007 instalado y condensadores de desacoplo de 100 nF.

DIMENSIÓN 2: Calidad del Firmware Arduino — PESO: 25 % (25 Puntos)

- Cero uso de delay(), planificación con millis(), filtro EMA, FSM con histéresis y FAULT_SAFE_SHUT, PID con anti-windup, CRC-16 y Watchdog 2s.

DIMENSIÓN 3: Robustez del Servicio Edge (RPi) — PESO: 25 % (25 Puntos)

- Servicios systemd con reconexión automática, testamento LWT en MQTT, cola Store-and-Forward en SQLite WAL, log2ram y Hardware Watchdog 15s.

DIMENSIÓN 4: Supervisión, Seguridad y Pruebas FI — PESO: 30 % (30 Puntos)

- Stack Docker operativo, MQTT con TLS 8883 + SAN + ACL, dashboard claro en Grafana (ISA-101) y aprobación de las 5 pruebas FI-01 a FI-05.

3. GRADUACIÓN E INSIGNIAS DEL CURSO (ANEXO C — 1000 XP)

- Curso Completo : 10 Módulos | 40 Lecciones | 2000 Minutos | 11 Videos HD
- Logro 1 (+75 XP) : "Arquitecto del Borde (Edge Architect)"
  Completa los módulos de Linux y Docker en Raspberry Pi.
- Logro 2 (+100 XP): "Maestro del Bus Industrial"
  Completa el laboratorio RS-485 y Modbus RTU sin errores CRC.
- Logro 3 (+250 XP): "Ingeniero IoT Full-Stack"
  Aprueba el Proyecto Integrador Final y las pruebas FI-01..FI-05.

Capítulo 33 · Resumen y cierre

Conceptos clave

Repasa la cadena completa y documenta cómo se valida cada interfaz antes de integrar hardware, red y aplicación.

