Integración con ATmega328P de cinco sistemas: control de plotter, control térmico con LM35, control de motor por referencia analógica, matriz RGB con joystick y cerradura RFID

1° Lucas Elizalde
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
lucas.elizalde@estudiantes.utec.edu.uy

2° Juan Manuel Ferreira
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
juan.ferreira.m@estudiantes.utec.edu.uy

3° Felipe Morrudo
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
felipe.morrudo@estudiantes.utec.edu.uy

***Resumen*—En este informe se presenta el diseño e implementación de cinco sistemas mecatrónicos basados en el microcontrolador ATmega328P: un control de plotter físico con motores paso a paso y sensores de límite para el trazado de figuras; un sistema de control térmico con sensor LM35, actuadores PWM y rangos ajustables por UART; un control de posición de motor DC basado en doble lectura ADC y PWM; un controlador de matriz RGB 8x8 gobernado por joystick analógico; y una cerradura electrónica con lector RFID, pantalla LCD I2C y gestión de ID en memoria EEPROM.**

Las pruebas en simulación (Proteus) y montaje físico demostraron un funcionamiento robusto, validando la capacidad del ATmega328P para integrar adquisición analógica (ADC), control PWM, comunicación serial (UART, SPI, I2C) y gestión de memoria en diversos sistemas embebidos.

***Index Terms*—ATmega328P, plotter, control de temperatura, LM35, PWM, control de motor, potenciómetro, matriz RGB, joystick, RFID, EEPROM, UART, sistemas embebidos, sensores, actuadores**

I. Introducción

El presente informe documenta el diseño, programación e implementación de cinco sistemas mecatrónicos basados en el microcontrolador ATmega328P, con el objetivo de integrar técnicas de sensado, accionamiento, comunicación y control digital en aplicaciones prácticas de laboratorio. Las actividades permiten consolidar conocimientos sobre adquisición de señales analógicas, control mediante PWM, comunicación serial UART y gestión de memoria EEPROM, así como el uso de sensores y actuadores en sistemas embebidos.

En primera instancia se desarrolló un sistema de control para un plóter bidimensional, empleando motores paso a paso, sensores de límite y un solenoide para trazado. El dispositivo ejecuta movimientos controlados en los ejes X–Y y permite seleccionar figuras predeterminadas mediante una interfaz UART, garantizando el respeto de los límites mecánicos y una operación segura.

Posteriormente, se implementó un sistema de regulación térmica basado en la lectura del sensor LM35, que controla un calefactor y un ventilador según rangos de temperatura definidos. El usuario puede ajustar el punto medio de referencia mediante menú serial, y los resultados son representados gráficamente para analizar la respuesta del sistema.

El tercer sistema consiste en un controlador de posición donde el ATmega328P lee dos potenciómetros y ajusta el giro de un motor mediante modulación PWM para igualar ambas referencias. El proceso incluye monitoreo continuo de valores analógicos, dirección de giro y señal de control transmitidos por puerto serial, además de representación gráfica del comportamiento dinámico.

A continuación, se desarrolló un controlador para matriz RGB utilizando un joystick analógico para desplazar un LED activo dentro de la matriz y modificar su color mediante el pulsador integrado. Se emplea lectura por ADC con zonas muertas para evitar ruido y se gestionan límites adecuados para el desplazamiento.

Por último, se implementó una cerradura electrónica con lector RFID, pantalla LCD I2C y almacenamiento de tarjeta autorizada en EEPROM. El sistema permite registrar, validar, actualizar y eliminar la tarjeta, indicando el estado mediante LEDs y comunicación UART, garantizando autenticación segura y persistencia de datos.

Estos desarrollos permiten comprender el diseño integral de sistemas embebidos, desde el análisis funcional hasta pruebas físicas, integrando adquisición de datos, control, interacción con el usuario y persistencia de información.

II. Objetivos

II-A. Objetivo General

- Diseñar e implementar sistemas mecatrónicos basados en el microcontrolador ATmega328P que integren sensado, control digital, comunicación serial y accionamiento para aplicaciones de automatización y control.

II-B. Objetivos Específicos

- Implementar un prototipo de plóter controlado por motores paso a paso y solenoide, con selección de figuras vía

UART y límites de seguridad.

- Desarrollar un sistema de control térmico con lectura mediante sensor LM35, control automático de calefactor y ventilador, ajuste del punto medio y visualización gráfica.
- Programar un sistema de control de posición basado en lectura analógica dual y PWM, con monitoreo serial de referencia, posición y señal de control.
- Controlar el desplazamiento y color de un LED en una matriz RGB mediante joystick analógico con gestión de límites y umbrales.
- Implementar una cerradura electrónica con lector RFID, LCD I2C, indicadores LED y almacenamiento persistente en EEPROM con funciones de registro y validación.

III. Materiales

- ATmega328P
- Fuente de alimentación
- Plotter
- Sensor LM35
- Shield con Calefactor
- Potenciómetros
- Motor DC
- Puente H
- Matriz RGB 8x8
- Joystick analógico
- Módulo RFID RC522
- Pantalla LCD 16x2 con interfaz I2C
- LEDs
- Protoboard
- Cables
- Resistencias
- Multímetro digital

IV. Marco Teórico

IV-A. Puente H

Un puente H es un esquema de conmutación electrónica diseñado para modificar la polaridad de la tensión suministrada a una carga, posibilitando así el control del sentido de giro en motores de corriente continua, tanto hacia adelante como hacia atrás. Este tipo de configuración es ampliamente utilizada en aplicaciones de robótica y en sistemas de conversión de potencia. Si bien existen versiones integradas en circuitos comerciales, también es posible implementar un puente H utilizando componentes electrónicos discretos (Wikipedia, 2025).

IV-B. Potenciometro

Un potenciómetro es un componente electrónico que funciona como una resistencia variable con tres terminales y un cursor móvil. Su operación permite ajustar manualmente el valor resistivo y obtener, entre el terminal central y uno de los extremos, una proporción de la diferencia de potencial aplicada, actuando como un divisor de tensión. Este dispositivo se emplea comúnmente para regular señales eléctricas, controlar niveles de salida y ajustar parámetros en circuitos electrónicos (Wikipedia, 2025).

IV-C. PySerial

PySerial es una biblioteca que brinda soporte para conexiones seriales (RS-232") a través de una variedad de dispositivos diferentes: puertos seriales de estilo antiguo, dongles Bluetooth, puertos infrarrojos, etc. También admite puertos serie remotos a través de RFC 2217 (desde V2.5) (PySerial - Python Wiki, s. f.)

IV-D. Matriz de LEDs RGB 8x8

Este módulo es un panel cuadrado con una interfaz XH2.54 de 3 pines. Este módulo de matriz LED a todo color RGB de 8x8 se basa en LED de control inteligentes WS2812. Cada LED se puede direccionar de forma independiente con píxeles RGB que pueden alcanzar 256 niveles de brillo DFRobot (s. f.).

IV-E. Modulo Joystick Analógico

El módulo joystick permite implementar un control manual bidireccional en los ejes X y Y, incorporando además una función de pulsador que se activa al presionarlo. Está compuesto por dos potenciómetros que generan señales analógicas independientes para cada eje (VRx y VRy) y una salida digital correspondiente al botón (SW). Para interpretar la posición del joystick, es necesario convertir las señales analógicas mediante el conversor analógico–digital (ADC) del microcontrolador (Modulo Joystick Analógico Con Pulsador - PatagoniaTec, s.f.).

IV-F. Motor Paso a Paso

Un motor paso a paso es un tipo de motor eléctrico que avanza en pasos discretos, lo que le permite tener un control preciso de su posición sin la necesidad de un sistema de retroalimentación. Estos motores se componen de un rotor dividido en secciones, y un conjunto de imanes o bobinas que son energizadas secuencialmente para generar un campo magnético que mueve el rotor en pasos definidos. Los motores paso a paso pueden ser de tipo unipolar o bipolar, siendo los motores bipolares más eficientes al requerir mayor control en la dirección de la corriente. Son muy utilizados en aplicaciones que requieren un control de posición preciso, como impresoras, sistemas CNC, robots, y dispositivos de posicionamiento. Ventajas incluyen alta precisión y torque a bajas velocidades, mientras que sus desventajas incluyen menor eficiencia y posibles ruidos y vibraciones debido a su movimiento discreto (Wikipedia, 2025).

IV-G. USBasp

El USBasp es un programador de microcontroladores basado en USB, utilizado principalmente para programar dispositivos AVR, como el ATmega328P, que es muy común en proyectos con Arduino. Este programador se conecta a un puerto USB de una computadora y comunica con el microcontrolador a través del protocolo SPI (Serial Peripheral Interface). El USBasp se utiliza para cargar el código en el microcontrolador, permitiendo la programación en el lugar de destino (in-circuit), sin necesidad de retirar el chip de la placa. Su principal ventaja es su bajo

costo y su capacidad de trabajar con una amplia variedad de microcontroladores AVR (De La Cruz, 2025).

IV-H. LM35

El LM35 es un circuito electrónico sensor que puede medir temperatura. Su salida es analógica, es decir, te proporciona un voltaje proporcional a la temperatura. El sensor tiene un rango desde 55°C a 150°C. (Administrador, 2018)

IV-I. RFID

La tecnología RFID es un tipo de sistema de identificación de productos, y básicamente su significado es identificación por radiofrecuencia. Es un sistema que es similar al de los códigos de barras tradicionales, pero tienen grandes ventajas frente a esta tecnología.

Esta tecnología no requiere identificar una etiqueta de un producto mediante una imagen. Por su parte la tecnología RFID utiliza ondas de radio con las que se comunican con un microchip, el cual se puede montar en muchos soportes tales como tag o etiqueta RFID, una tarjeta o un transpondedor.(Tecnipesa. Soluciones de Marcaje, Etiquetado y Codificación de almacenes., s. f.)

V. Procedimiento

V-A. Control del Plotter (Físico)

El objetivo de este apartado es controlar el trazador gráfico (plotter) con un microcontrolador ATmega328P. Para ello, se retomaron las figuras del trabajo previo, como la cruz, el triángulo y el círculo, además de los nuevos patrones (*Rana* y *Manzana*), que fueron ejecutados **exclusivamente en equipo real** (sin simulación y sin interfaz UART en esta entrega).

El control de los movimientos del plotter se logró mediante drivers *step/dir/enable* para los ejes X e Y, y un solenoide para la herramienta de trazo, permitiendo la subida y bajada de la lapicera. La asignación de señales del ATmega328P se resume en el Cuadro I.

Cuadro I: Asignación de pines del ATmega328P hacia los drivers de ejes y solenoide

Señal	Pin ATmega328P
X_STEP	PB3
X_DIR	PB4
X_EN	PB5
Y_STEP	PC3
Y_DIR	PC4
Y_EN	PC5
Solenoide (lapicera)	PC0
Límite superior (INT0)	PD2
Límite inferior (INT1)	PD3

El firmware se ejecutó con `F_CPU` $= 1$ MHz (RC interno con `CKDIV8` activo). Los pulsos de `STEP` fueron generados por *bit-banging* utilizando retardos en ciclos (`_delay_loop_2`), calibrando los tiempos de `HIGH`/`LOW` para asegurar un avance estable en ambos ejes.

Los finales de carrera en los pines PD2/PD3 se configuraron como entradas con *pull-up* y se habilitaron interrupciones externas `INT0`/`INT1` por flanco descendente. Cuando se dispara un límite, se activa una bandera de aborto que forza el movimiento de la pluma hacia arriba y deshabilita ambos drivers, dejando el sistema en un estado seguro.

La programación del microcontrolador se realizó a través del programador USBasp usando el conector de 6 pines (MISO, MOSI, SCK, RESET, VCC, GND) con alimentación a 5V y masa común. Dado que el micro funciona a `F_CPU` $= 1$ MHz, se activó el modo de reloj lento del USBasp para garantizar una captura estable del ISP. La escritura y verificación del binario se realizó con `avrdude`, manteniendo el `SPIEN` habilitado y sin modificar el `CKDIV8`. Las conexiones se muestran en el Cuadro II.

Cuadro II: Conexiones USBasp ↔ ATmega328P (ISP de 6 pines).

USBasp	ATmega328P
MOSI	PB3 (MOSI)
MISO	PB4 (MISO)
SCK	PB5 (SCK)
RESET	RESET
VCC	VCC (5V)
GND	GND

Se trazaron, en este orden: **Cruz** (dos diagonales a 45°), **Cuadrado** (aunque no se pedía, sirve como referencia para observar la calibración de los motores), **Triángulo**, **Círculo** (aproximado por pequeños arcos cardenales), y las siluetas extendidas de **Rana** y **Manzana**. Las Fig. 1 y 2 muestran el montaje del plotter durante las pruebas.

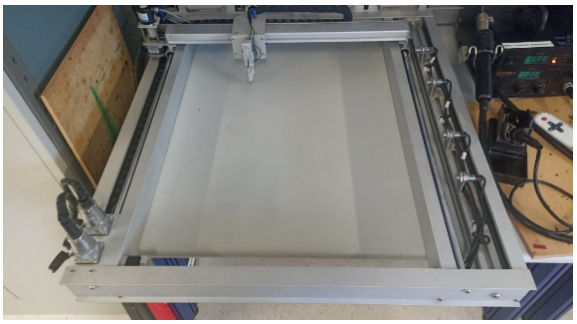

Fig. 1: Montaje del plotter.

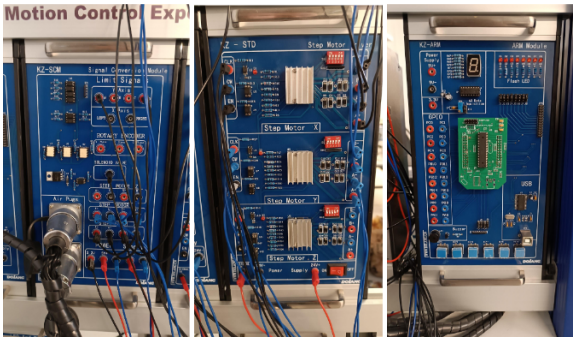

Fig. 2: Conexionado hacia drivers y solenoide.

V-B. RC522

En este apartado se llevo a cabo la implementación de una cerradura electrónica inteligente, se utilizó un lector RFID RC522 el cual lee las ID de las tarjetas de los usuarios para identificar si el acceso debe ser permitido o negado. Las instrucciones y el estado del acceso deben ser visualizados a través de una pantalla LCD con interfaz I2C y comunicación UART, además de dos LEDs(rojo y verde) los cuales indican si la tarjeta leída tiene acceso denegado o permitido. Para la modificación del ID que tiene el acceso permitido, se disponen de dos botones, uno para borrar dicha ID y otro para agregar una nueva ID. En el caso de agregar una ID nueva, ya habiendo una registrada, la nueva reemplaza a la anterior. También la ID de la tarjeta con acceso autorizado debe guardarse en la EEPROM, por lo que si se apaga el sistema, esta ID sigue manteniendo el acceso permitido al encenderse nuevamente.

Una vez se comprendió los requerimientos necesarios para cumplir la solución, se empezó a simular el apartado en el software PROTEUS. Dicho software no cuenta con un lector RFID RC522, por lo que en su defecto se utilizo una segunda terminal que actue como lector, ya que por esta se ingresan las ID de las tarjetas a leer. Además, para poder utilizar la interfaz I2C fue necesario implementar un módulo PCF8574 la cual usa un bus de datos I2c, ya que no es posible configurar la LCD directamente en PROTEUS.

La distribución de pines utilizada para el diseño del circuito se muestra a continuación en el Cuadro.III.

Cuadro III: Conexiones del ATMEGA328P

Componente	Pin ATmega328P
Botón de borrar	D4 (PD4)
Botón de actualizar	D5 (PD5)
LED Rojo	D6 (PD6)
LED Verde	D7 (PD7)
PCF8574(SDA)	A4 (PC4)
PCF8574(SCL)	A5 (PC5)
Terminal "Lector"(TXD)	RXD (PDO)
Terminal Estado (RXD)	TXD (PD1)

Las conexiones entre el modulo PCF8574 y la pantalla LCD se muestran a continuación en el Cuadro.IV.

Cuadro IV: Conexiones del PCF8574 y LCD

PCF8574	Pantalla LCD
P0	RS
P2	E
P4	D4
P5	D5
P6	D6
P7	D7

Una vez finalizado la implementación en simulación, se llevo a cabo el ensamblaje físico, donde se mantuvo la misma distribución de pines, exceptuando que ahora si se puede conectar un lector RFID 522. A continuación, en el Cuadro.V. se observa las conexiones de dicho lector.

Cuadro V: Conexiones del RC522

RC522	Pin ATmega328P
SDA	D10 (PB2)
RST	D9 (PB1)
MOSI	D11 (PB3)
MISO	D12 (PB4)
SCK	D13 (PD5)
3.3V	3.3V
GND	GND

V-C. Sistema de control de temperatura

En este apartado se implenta un sistema de control de temperatura, para esto se hace uso de un shield de Arduino, el cual cuenta con transistores TIP31 los cuales cumplen el rol de calefactores. Utilizando un puente H también se controla un ventilador a diferentes velocidades según el requerimiento del sistema. A continuación, se lista los rangos de temperatura que rige el sistema

- Entre 0 y 22 grados: Encender el calefactor para incrementar la temperatura de manera continua hasta alcanzar el rango deseado.
- Entre 23 y 30 grados (punto medio): Mantener el calefactor apagado y el ventilador apagado, ya que la temperatura se encuentra dentro de un rango adecuado.
- Entre 31 y 40 grados: Encender el ventilador a baja velocidad para comenzar a reducir la temperatura ambiente de forma gradual.
- Entre 41 y 50 grados: Encender el ventilador a velocidad media para un enfriamiento moderado.
- Más de 51 grados: Encender el ventilador a alta velocidad para reducir la temperatura de manera urgente y prevenir sobrecalentamientos.

El sistema debe actuar de forma automática para poder mantener la temperatura en el rango ideal, activando el calefactor al estar debajo de este rango, y el ventilador al superarlo. La temperatura y los elementos activados deben ser mostrados cada un segundo a través de UART, para que el usuario pueda monitorear a tiempo real.

Además, el usuario debe tener libertad de modificar el rango del punto medio a su gusto, lo que implica modificar el rango de los demas para no solaparse. Por último, se debe hacer una representación visual del estado del sistema, por lo que se usara Phyton para gráficar, dicha gráfica debe contener lo siguiente:

- Evolución de la temperatura.
- Acciones del sistema (encendido/apagado del calefactor y velocidad del ventilador).
- Rango de temperatura ideal según el "punto medio"definido.

Una vez comprendidos los requerimientos necesarios para cumplir la consigna, se empezó a diseñar la simulación en

el software PROTEUS. Al no tener el shield de componente, simplemente se colocaron los componentes independientemente. A continuación, en el Cuadro VI se observa la distribución de pines

Cuadro VI: Conexiones del ATMEGA328P

Componente	Pin ATmega328P
Calefactor	D4 (PD3)
Ventilador	D5 (PD5)
Sensor	A0 (PC0)
Serial(RX)	D0 (PD0)
Serial(TX)	D1 (PD1)

Por más que en simulación funcione el ventilador sin necesidad de un puente H, se coloca uno para que sea mas acorde a los componentes utilizados en fisico. Al no poder representarse el calor en PROTEUS, la temperatura se debe modificar manualmente desde el sensor, y para indicar que el calefactor esta calentando, se le pone un LED al transistor. Este transistor TIP 31 es un transistor NPN por lo que actua como interruptor, cuando no esta activado funciona como un interruptor abierto, y al momento de activarse es un interruptor cerrado, por lo que enciende el LED representando que el calefactor está funcionando.

Una vez corroborada la solución en simulación, se paso a la etapa de ensamblaje, donde fue necesario alimentar el shield con una fuente externa para poder lograr que el calefactor caliente a las temperaturas deseadas. Se mantuvieron la misma distribución de pines que en el entorno simulado.

Por último se implemento un código en phyton para el pyserial, que sea capaz de gráficar a tiempo real la temperatura, las acciones del sistema y la temperatura ideal segun el rango de punto medio establecido.

V-D. Control de Motor (Simulado y Físico)

El objetivo de este apartado es programar el microcontrolador ATmega328P para implementar un sistema de control de posición para un motor de corriente continua. El sistema deberá leer un valor de referencia desde un primer potenciómetro y ajustar la posición de un motor, mediante modulación por ancho de pulso (PWM), hasta que el valor de un segundo potenciómetro, acoplado físicamente al eje del motor, coincida con dicha referencia.

El microcontrolador comparará continuamente el valor objetivo (potenciómetro 1) con el valor actual (potenciómetro 2). Basado en la diferencia entre estas dos señales, el sistema calculará la señal de control PWM necesaria para variar la velocidad de giro del motor y determinará el sentido de giro (horario o antihorario) requerido para aproximar el valor actual al valor de referencia.

Durante todo el proceso, el sistema informará constantemente, a través de la comunicación serial, los siguientes parámetros clave para permitir un monitoreo en tiempo real: el valor del primer potenciómetro (referencia), el valor del segundo potenciómetro (posición actual del motor), el valor del ciclo de trabajo del PWM aplicado al motor y el sentido de giro seleccionado.

Adicionalmente, como parte de la validación del sistema, se requiere la captura de estos datos seriales para su posterior graficación utilizando Python o Matlab, permitiendo analizar visualmente el comportamiento del sistema de control (valor de referencia vs. valor actual) y la señal de control (PWM) en el tiempo.

Una vez comprendido el funcionamiento del sistema, se procedió a realizar la programación mientras, de manera simultánea, se efectuaba la simulación del circuito en el software Proteus. Para organizar el desarrollo, la implementación se dividió en tres etapas principales:

En primer lugar, se configuraron las entradas analógico-digitales (ADC) para la correcta lectura de ambos potenciómetros, obteniendo los valores de referencia (desde el canal 3) y de posición actual (desde el canal 4).

En segundo lugar, se implementó la lógica de control principal. Esta rutina calcula el error (diferencia) entre la referencia y la posición. Si el error absoluto se encuentra dentro de una `TOLERANCIA` definida, el motor se detiene (`motor_parar()`). Si está fuera de la tolerancia, el sistema determina el sentido de giro (manejado por los pines `PD2` y `PD3`) y calcula una señal PWM. La velocidad se fija a un valor mínimo (`PWM_MINIMO`) para errores pequeños (pero fuera de la tolerancia) y se satura en un valor máximo (`PWM_MAXIMO`) para errores grandes, asegurando un movimiento controlado.

Finalmente, se configuró el Timer1 para generar la señal PWM en el pin `PB1` (D9) y se habilitó la comunicación serial (USART) para transmitir de forma continua los datos de monitoreo (ambos potenciómetros, PWM y dirección) hacia la computadora.

Para la programación y simulación se utilizó la siguiente distribución de pines entre el circuito y el microcontrolador ATmega328P. En el Cuadro VII se muestran las conexiones analógicas y de control del motor.

Cuadro VII: Conexión de Potenciómetros y Motor al ATmega328P

Componente	Pin ATmega328P
Potenciómetro de Referencia (Señal)	A3 (PC3)
Potenciómetro del Motor (Señal)	A4 (PC4)
Salida PWM (Habilitador Motor)	D9 (PB1)
Control Dirección Motor 1 (IN1)	D2 (PD2)
Control Dirección Motor 2 (IN2)	D3 (PD3)

Una vez concluida la simulación en el software y verificado el correcto funcionamiento del lazo de control, se implementó el circuito en físico utilizando la misma distribución de pines definida en la etapa de prueba.

V-E. Matriz RGB 8×8 con ATmega328P y Joystick (Físico)

En este apartado se controló una matriz RGB 8×8 basada en LEDs `WS2812B` con un ATmega328P ejecutando firmware bare-metal en `C`. El desplazamiento del píxel activo se realizó

con un joystick analógico (ejes X/Y a ADC) y su pulsador integrado para cambiar el color del píxel de forma aleatoria.

El pin de datos de la matriz WS2812 se conectó a PD2 (D2). El joystick se conectó a ADC0/ADC1 (A0/A1) y el pulsador a PD3 (D3) con *pull-up*. La matriz se alimentó a 5V desde fuente externa con masa común. El Cuadro VIII resume el conexionado.

Cuadro VIII: Conexiones del sistema (ATmega328P ↔ Matriz/Joystick)

Señal / Componente	**Pin ATmega328P**
WS2812 *Data In*	PD2 (D2)
Pulsador (color aleatorio)	PD3 (D3), entrada con *pull-up*
Joystick X	ADC0 (PC0 / A0)
Joystick Y	ADC1 (PC1 / A1)
VCC Lógica	5V (fuente externa)
GND Común	GND compartida

El protocolo WS2812 opera a 800kHz con codificación temporal: para '1' ($T_H \approx 0{,}7\ \mu s$, $T_L \approx 0{,}55\ \mu s$) y para '0' ($T_H \approx 0{,}35\ \mu s$, $T_L \approx 0{,}9\ \mu s$). Se transmitió en orden **GRB** y se aplicó un *reset/latch* final de $\sim 300\ \mu s$.

El búfer se almacenó como arreglo lineal GRB; `index_xy()` convierte $(x, y) \rightarrow$ índice, con opción de mapeo lineal o serpentina. Se definió una **zona muerta** (*deadzone*) alrededor del centro del joystick y un **tiempo mínimo entre pasos** (*cooldown*) para evitar desplazamientos múltiples por ruido.

VI. Resultados

VI-A. Control del Plotter con ATmega328P

Una vez configurado el **USBasp** y cargado el código en el ATmega328P, se validó el trazado **en equipo real** para todas las figuras.

Inicialmente, el plotter ejecuta un patrón de movimiento alrededor de la hoja para verificar los márgenes. Durante este recorrido, el sistema detecta si la herramienta se sale de los márgenes predefinidos mediante los sensores de límite ubicados en los extremos de los ejes Y. Si el movimiento supera los márgenes, se activa una señal de abortar que detiene el sistema. En ese caso, el usuario debe reacomodar la hoja y el sistema reinicia el proceso de trazado desde el principio, asegurando que no haya trazos fuera de los límites de la hoja.

La Fig. 3 muestra los dibujos obtenidos (cruz, cuadrado, triángulo, círculo y patrones extendidos).

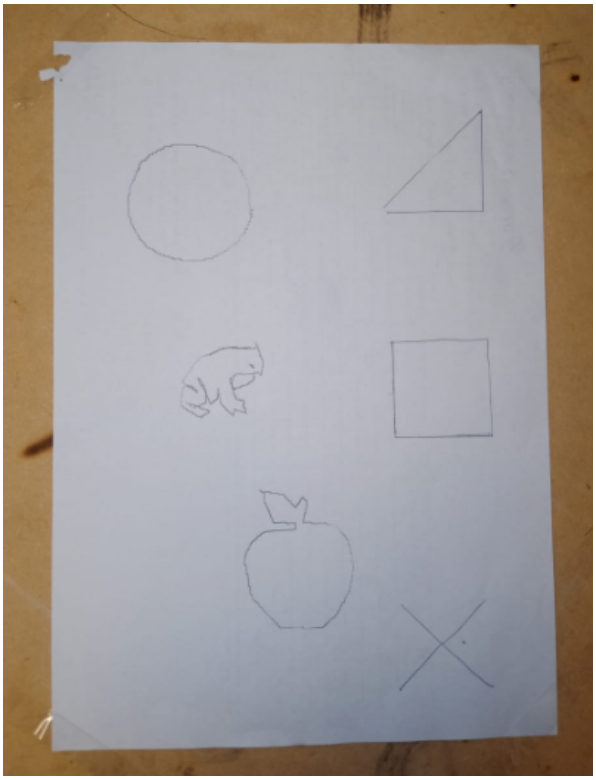

Fig. 3: Resultados de trazado en papel: figuras básicas y siluetas extendidas.

VI-A0a. Comportamiento observado.: (i) La **cruz** mostró convergencia limpia en el centro al ejecutar diagonales 45°. (ii) El **cuadrado** cerró relativamente bien. (iii) El **triángulo** mantuvo rectitud en catetos y cierre sin desalineo. (iv) El **círculo** presentó contorno uniforme y simétrico. (v) En *Rana* y *Manzana* se logró continuidad de contorno con cambios de dirección cortos y conmutaciones nítidas de *pluma arriba/abajo*. No se detectaron pérdidas de paso a la cadencia utilizada.

VI-A0b. Explicación del firmware (fragmentos relevantes).: A continuación se documentan los bloques clave del código:

(1) Generación de pasos por eje: Cada eje se modela con su puerto de `DIR` y `STEP`; el pulso se emite con retardos calibrados para garantizar tiempo en `HIGH`/`LOW`.

Listing 1: Ejecución de pasos en un eje.

```
static inline void run_steps(const eje_t *e, uint8_t
     dir, uint16_t n){
  abort_if_needed();
  if (dir) poner_bit(e->port_dir, e->bit_dir);
  else     limpiar_bit(e->port_dir, e->bit_dir);
  ciclo_delay(T_SETUP_DIR);
  while(n--){
    abort_if_needed();
    poner_bit(e->port_step, e->bit_step);
    ciclo_delay(T_PULSO_ALTO);
    limpiar_bit(e->port_step, e->bit_step);
    ciclo_delay(T_PULSO_BAJO);
  }
}
```

(2) Diagonales a 45°: Se fijan sentidos en X/Y y se emiten pulsos casi simultáneos para lograr pendiente ± 1.

Listing 2: Movimiento diagonal 45°.

```
void d45(uint8_t modo, uint16_t pasos){
  /* fija DIR en X/Y segn modo y luego emite STEP en
     paralelo */
  ...
  while (pasos--){
    abort_if_needed();
    poner_bit(EJE_X.port_step, EJE_X.bit_step);
    poner_bit(EJE_Y.port_step, EJE_Y.bit_step);
    ciclo_delay(T_PULSO_ALTO);
    limpiar_bit(EJE_X.port_step, EJE_X.bit_step);
    limpiar_bit(EJE_Y.port_step, EJE_Y.bit_step);
    ciclo_delay(T_PULSO_BAJO);
  }
}
```

(3) Movimiento con pendiente arbitraria (DDA/Bresenham): Se usan acumuladores para decidir en qué iteraciones se emite `STEP` en cada eje, reproduciendo razones $\Delta X : \Delta Y$.

Listing 3: Interpolación de pasos con DDA.

```
void mover_xy(uint8_t dirx, uint8_t diry, uint16_t
    px, uint16_t py){
  uint16_t maxp = (px>py)?px:py; uint32_t ax=0, ay=0
    ;
  ...
  for(uint16_t i=0;i<maxp;i++){
    abort_if_needed();
    uint8_t sx=0, sy=0;
    ax += px; if(ax>=maxp){ ax-=maxp; sx=1; }
    ay += py; if(ay>=maxp){ ay-=maxp; sy=1; }
    if(sx) { /* pulso X */ }
    if(sy) { /* pulso Y */ }
    /* temporizacin HIGH/LOW */
  }
}
```

(4) Gestión de pluma *y seguridad:* El solenoide se controla desde PC0; ante bandera de aborto se garantiza *pluma arriba* y deshabilitación de drivers.

Listing 4: Pluma y rutina de aborto.

```
static inline void pluma_abajo(void){ limpiar_bit(&
    PORTC, SOLENOIDE); _delay_ms(100); }
static inline void pluma_arriba(void){ poner_bit(&
    PORTC, SOLENOIDE);  _delay_ms(100); }

ISR(INT0_vect){ abort_flag = 1; }
ISR(INT1_vect){ abort_flag = 1; }
static inline void abort_if_needed(void){
  if(!abort_flag) return;
  pluma_arriba();
  limpiar_bit(&PORTB, HABIL_X);
  limpiar_bit(&PORTC, HABIL_Y);
  cli(); while(1){}  /* estado seguro */
}
```

VI-A0c. Síntesis.: Con reloj interno a 1MHz, pulsos `STEP` cronometrados por ciclo y protección por `INT0/1`, el sistema ejecutó las figuras previstas sin pérdidas de paso. La carga del firmware vía **USBasp** (modo *slow SCK*) fue estable y permitió iterar rápidamente los tiempos de pulso hasta obtener el trazo mostrado en la Fig. 3.

VI-B. RC522

VI-B1. Simulación y montaje: El apartado fue emulado con éxito, siendo capaz de guardar el ID de la tarjeta correcta en la EEPROM, accediendo correctamente al momento de mandar dicha ID por la terminal donde se reciben las IDs, y negando el acceso a claves diferentes. Ambos botones cumplen con su funcionalidad, borrando y actualizando la ID almacenada sin inconvenientes. La pantalla LCD, los LEDs y la comunicación UART indican en cada momento el estado del acceso. A continuación, en la Fig.4 se observa el circuito emulado en el software PROTEUS.

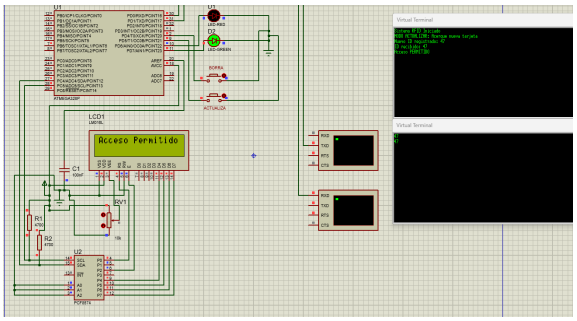

Fig. 4: Simulación en PROTEUS Cerradura RFID.

Después de ser corroborado la correcta funcionalidad del circuito en simulación, se llevo a cabo el montaje en físico. A continuación, en la Fig.5 se observa el circuito montado en físico.

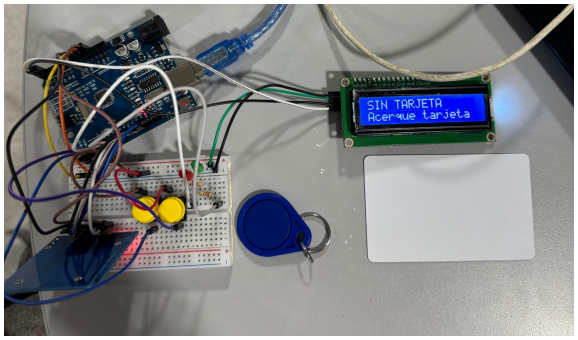

Fig. 5: Ensamblaje en físico Cerradura RFID.

VI-B2. Código: Para el diseño del código, fue necesario incluir las librerias RC522, SPI y UART. A continuación, se mostraran bloques del código fundamentales para su correcto funcionamiento.

VI-B3. Funciones del código: Se declararon las funciones necesarias para cumplir con lo requerido en la propuesta

Listing 5: Funciones del codigo

```
// funciones I2C
void I2C_Inicializar(void) {
    TWSR = 0x00;
    TWBR = 0x48;
    TWCR = (1 << TWEN);
}

void I2C_Iniciar(void) {
```

```
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN)
    ;
    while (!(TWCR & (1 << TWINT)));
}

void I2C_Detener(void) {
    TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN)
    ;
    _delay_us(100);
}

void I2C_Escribir(uint8_t dato) {
    TWDR = dato;
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));
}

// LCD
void LCD_EnviarNibble(uint8_t nibble, uint8_t rs) {
    uint8_t dato = nibble | LCD_LUZ_FONDO;
    if (rs) dato |= LCD_RS;

    I2C_Iniciar();
    I2C_Escribir(LCD_DIRECCION << 1);
    I2C_Escribir(dato | LCD_HABILITAR);
    _delay_us(1);
    I2C_Escribir(dato);
    _delay_us(50);
    I2C_Detener();
}

void LCD_EnviarByte(uint8_t dato, uint8_t rs) {
    LCD_EnviarNibble(dato & 0xF0, rs);
    LCD_EnviarNibble((dato << 4) & 0xF0, rs);
}

void LCD_Comando(uint8_t cmd) {
    LCD_EnviarByte(cmd, 0);
    _delay_ms(2);
}

void LCD_Dato(uint8_t dato) {
    LCD_EnviarByte(dato, 1);
}

void LCD_Inicializar(void) {
    _delay_ms(50);
    LCD_EnviarNibble(0x30, 0);
    _delay_ms(5);
    LCD_EnviarNibble(0x30, 0);
    _delay_us(150);
    LCD_EnviarNibble(0x30, 0);
    LCD_EnviarNibble(0x20, 0);

    LCD_Comando(0x28);
    LCD_Comando(0x0C);
    LCD_Comando(0x06);
    LCD_Comando(0x01);
    _delay_ms(2);
}


// funciones de la EEPROM
void EEPROM_GuardarTarjeta(volatile uint8_t *
    id_tarjeta) {
    eeprom_update_byte((uint8_t*)
    EEPROM_DIR_TARJETA_VALIDA, 0xAA);
    for (uint8_t i = 0; i < 4; i++) {
        eeprom_update_byte((uint8_t*)(
    EEPROM_DIR_TARJETA_ID + i), id_tarjeta[i]);
    }
}
```

```
uint8_t EEPROM_CargarTarjeta(volatile uint8_t *
    id_tarjeta) {
    uint8_t valida = eeprom_read_byte((uint8_t*)
    EEPROM_DIR_TARJETA_VALIDA);
    if (valida == 0xAA) {
        for (uint8_t i = 0; i < 4; i++) {
            id_tarjeta[i] = eeprom_read_byte((uint8
    _t*)(EEPROM_DIR_TARJETA_ID + i));
        }
        return 1;
    }
    return 0;
}

void EEPROM_BorrarTarjeta(void) {
    eeprom_update_byte((uint8_t*)
    EEPROM_DIR_TARJETA_VALIDA, 0xFF);
}
```

VI-C. Sistema de control de temperatura

La simulación cumplió con todos los requerimientos correctamente, siendo capaz de registrar la temperatura y el estado del sistema en cada momento, informando al usuario a través de UART. También actua de forma automática al entrar en rangos que no sean el rango medio. También el usuario puede modificar a su gusto el rango del punto medio. A continuación, en la Fig.6 se observa el diagrama del circuito en simulación.

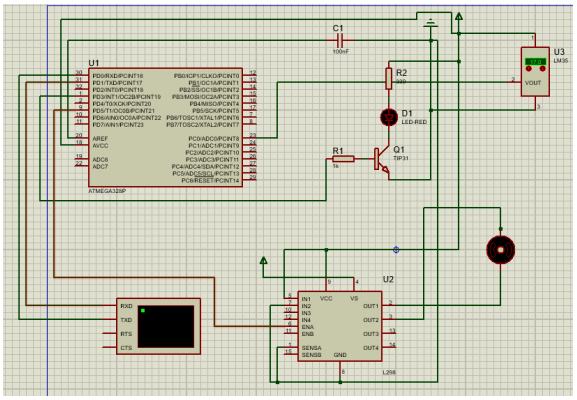

Fig. 6: Circuito simulado del control de temperatura.

Una vez finalizada la simulación, se procedió a montar el circuito en físico. A continuación, en la Fig. se observa el circuito montado

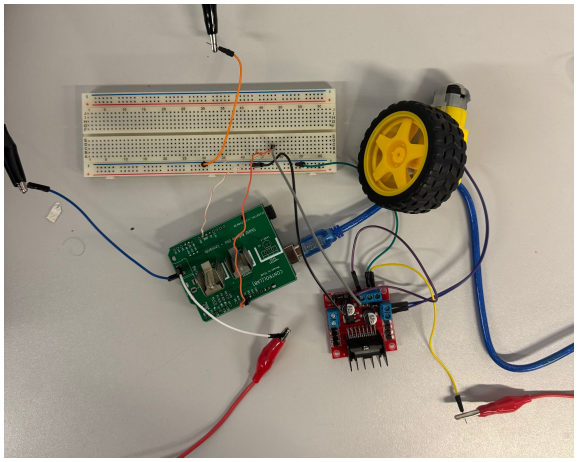

Fig. 7: Circuito montado del control de temperatura.

Los valores de PWM usados en el circuito se muestran a continuación en el Cuadro.IX

Cuadro IX: PWM

Componente	PWM
Calefactor encendido	250
Ventilador bajo	85
Ventilador medio	170
Ventilador alto	255

Después de corroborar que la implementación en físico funcionaba correctamente, se programo un código en Phyton que use Pyserial para hacer una gráfica que represente de forma visual las acciones del sistema y la temperatura del mismo a tiempo real. A continuación, en la Fig. 8 se muestra la gráfica resultante.

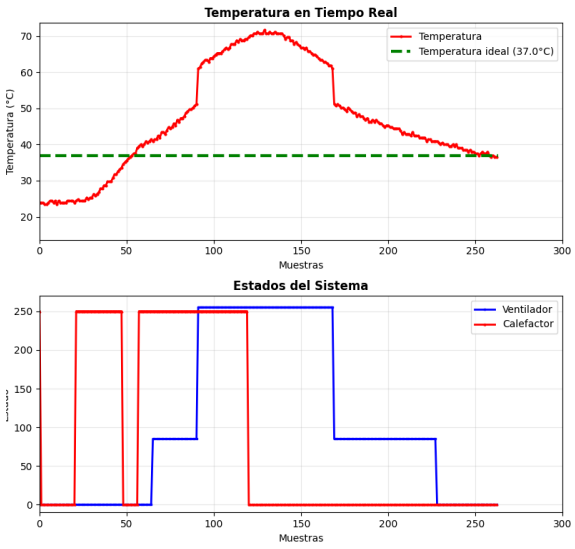

Fig. 8: Gráfica de temperatura y actuadores con Pyseriañ

La linea verde representa la temperatura ideal, que es un promedio entre los dos rangos del punto medio. Como se observa en la gráfica, este punto medio fue modificado a un rango con valores superiores a la temperatura ambiente, para poder observar el primer estado donde el calefactor se enciende automáticamente. Se puede ver en la gráfica de abajo que el calefactor se activa a un PWM automaticamente, y cuando en la gráfica de arriba la temperatura llega al punto medio, este se apaga automaticamente. Después, se enciende manualmente para elevar la temperatura con el fin de observar todos los estados del ventilador. se observa que, a medida que la temperatura aumenta progresivamente, el ventilador incrementa su PWM al entrar en rangos de temperatura mas altos. Una vez que se apaga el calefactor, la temperatura empieza a descender, y el PWM del ventilador baja mientras se acerca al rango del punto medio, apagandose completamente al llegar la temperatura a este rango deseado.

A continuación, se mostraran bloques de código fundamentales para el correcto funcionamiento del sistema de control automatico.

VI-C1. Funciones ADC y PWM: Se declaran funciones para leer la temperatura, y controlar el PWM del calefactor y el ventilador

Listing 6: Funciones del codigo

```
// ADC y PWM
static void adc_inicializar(void){
ADMUX  = (1<<REFS0) | (CANAL_ADC_LM35 & 0x0F);
ADCSRA = (1<<ADEN) | (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0
    );
        }
        static uint16_t adc_leer_canal(uint8_t canal
    ){
                ADMUX = (ADMUX & 0xF0) | (canal & 0x
    0F);
                ADCSRA |= (1<<ADSC);
                while(ADCSRA & (1<<ADSC));
                return ADC;
        }

        static void calefactor_pwm_inicializar(void)
    {
                CALEFACTOR_DDR |= CALEFACTOR_PIN_BM;
                TCCR0A = (1<<COM0B1) | (1<<WGM01) |
    (1<<WGM00);
                TCCR0B = (1<<CS01) | (1<<CS00);
                OCR0B  = 0;
        }
        static inline void calefactor_fijar_pwm(uint
    8_t ciclo_trabajo){ OCR0B = ciclo_trabajo; }

        static void ventilador_pwm_inicializar(void)
    {
                VENTILADOR_DDR |= VENTILADOR_PIN_BM;
                TCCR1A = (1<<WGM10);
                TCCR1B = (1<<WGM12);
        ifVENTILADOR_INVERTIDO
                TCCR1A |= (1<<COM1B1) | (1<<COM1B0);
        else
                TCCR1A|= (1<<COM1B1);
        endif
                TCCR1B|= (1<<CS11) | (1<<CS10);
                OCR1B = 0;
        }
        static inline void ventilador_fijar_pwm(uint
    8_t ciclo_trabajo){
        ifVENTILADOR_INVERTIDO
```

```
                OCR1B = (uint8_t)(255 -
    ciclo_trabajo);
        else
                OCR1B= ciclo_trabajo;
        endif
        }
    // Control ventilador
        static accion_ventilador_t
    aplicar_control_ventilador(uint16_t temp_celsius
    ){
                if (temp_celsius <= T_CALOR_MAX){
    ventilador_fijar_pwm(0); return VENT_APAGADO; }
                else if (temp_celsius <= T_MEDIO_MAX
    ){ ventilador_fijar_pwm(0); return VENT_APAGADO;
     }
                else if (temp_celsius <= T_BAJO_MAX)
    { ventilador_fijar_pwm(VENTILADOR_CICLO_BAJO);
    return VENT_BAJO;  }
                else if (temp_celsius <= T_MED_MAX){
     ventilador_fijar_pwm(VENTILADOR_CICLO_MEDIO);
    return VENT_MEDIO; }
                else {
      ventilador_fijar_pwm(VENTILADOR_CICLO_ALTO);
    return VENT_ALTO;  }
        }
```

VI-C2. Configuración de rangos: También se definieron los rangos iniciales, y las funciones necesarias para que el usuario pueda modificar a voluntad el rango deseado del punto medio.

Listing 7: Manejo de rangos

```
static void mostrar_rango_actual(uint16_t
    temp_celsius){
                if (temp_celsius <= T_CALOR_MAX){
                        uart_tx_cadena("Rango: 0-");
     uart_tx_entero16(T_CALOR_MAX);
                } else if (temp_celsius <=
    T_MEDIO_MAX){
                        uart_tx_cadena("Rango: ");
    uart_tx_entero16(T_MEDIO_MIN); uart_tx_caracter(
    '-'); uart_tx_entero16(T_MEDIO_MAX);
                } else if (temp_celsius <=
    T_BAJO_MAX){
                        uart_tx_cadena("Rango: ");
    uart_tx_entero16((uint16_t)(T_MEDIO_MAX+1));
    uart_tx_caracter('-'); uart_tx_entero16(
    T_BAJO_MAX);
                } else if (temp_celsius <= T_MED_MAX
    ){
                        uart_tx_cadena("Rango: ");
    uart_tx_entero16((uint16_t)(T_BAJO_MAX+1));
    uart_tx_caracter('-'); uart_tx_entero16(
    T_MED_MAX);
                } else {
                        uart_tx_cadena("Rango: >= ")
    ; uart_tx_entero16((uint16_t)(T_MED_MAX+1));
                }
        }

// comando para reconfigurar el punto medio
        static void comando_reconfigurar_rango(void)
    {
                uart_tx_cadena("\r\n Reconfigurar
    Rango Medio \r\n");
                uart_rx_limpiar_buffer();
                uart_tx_cadena("Ingrese minimo del
    nuevo rango: ");
                uint16_t min_medio =
    uart_leer_entero_flexible();
                uart_tx_cadena("Ingrese maximo del
    nuevo rango: ");
                uint16_t max_medio =
    uart_leer_entero_flexible();
    recalcular_desde_medio(min_medio, max_medio);
                uart_tx_cadena("Nuevo rango aplicado
    .\r\n");
        }
```

VI-D. Control de Motor

El programa, desarrollado en lenguaje C para el ATmega328P, implementa un control proporcional simple para un motor de CC. El sistema utiliza dos canales ADC para leer la referencia y la posición, el Timer1 para generar una señal PWM de 8 bits (Modo 5, Fast PWM) y dos pines digitales para controlar la dirección mediante un driver (Puente-H).

El código se estructuró en funciones de inicialización para cada periférico (`iniciar_adc`, `iniciar_pwm`, `iniciar_motor`, `iniciar_uart`) y funciones de control (`motor_derecha`, `motor_izquierda`, `motor_parar`). Esto permite un bucle `main` limpio que se centra exclusivamente en la lógica de control.

En cuanto al código, este incluye las librerías estándar de AVR: `<avr/io.h>` (registros), `<util/delay.h>` (retardos), `<stdint.h>` (tipos de enteros) y `<stdio.h>` (entrada/salida estándar). La inclusión de `stdio.h` es clave, ya que se utiliza para redirigir `stdout` al puerto serial, permitiendo el uso de `printf` para la telemetría.

A continuación, se muestra la configuración de los periféricos. La UART se configura a 9600 baudios y se redirige `stdout` a ella. El ADC se configura con prescaler de 128 (125KHz) y referencia en AVCC. El PWM (Timer1) se configura en modo Fast PWM de 8 bits, con prescaler de 8, resultando en una frecuencia de $\approx 7{,}8$ KHz.

```
/* Redireccion de printf a UART */
static int uart_putchar(char c, FILE *stream){
    if (c == '\n'){ while(!(UCSR0A & (1<<UDRE0
        ))); UDR0 = '\r'; }
    while(!(UCSR0A & (1<<UDRE0)));
    UDR0 = c;
    return 0;
}
static FILE uart_stdout = FDEV_SETUP_STREAM(
    uart_putchar, NULL, _FDEV_SETUP_WRITE);

void iniciar_uart(uint32_t baud){
    uint16_t ubrr = (F_CPU/16/baud) - 1;
    UBRR0H = (uint8_t)(ubrr>>8);
    UBRR0L = (uint8_t)(ubrr);
    UCSR0B = (1<<TXEN0);
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
    stdout = &uart_stdout;
}

void iniciar_pwm(void){
    DDRB |= (1 << PB1);
    TCCR1A = (1 << COM1A1) | (1 << WGM10);
    TCCR1B = (1 << WGM12) | (1 << CS11);
}

void iniciar_motor(void){
    DDRD |= (1 << IN1) | (1 << IN2);
    PORTD &= ~((1 << IN1) | (1 << IN2));
```

```
}

void iniciar_adc(void){
    ADMUX  = (1 << REFS0);
    ADCSRA = (1 << ADEN) | (1 << ADPS2)|(1 <<
        ADPS1)|(1 << ADPS0);
    DIDR0  = (1 << ADC3D) | (1 << ADC4D);
    ADCSRA |= (1 << ADSC);
    while(ADCSRA & (1 << ADSC));
}
```

Listing 8: Inicialización de periféricos (ADC, PWM, Motor y UART)

La lógica principal, mostrada a continuación, reside en el bucle `while(1)`. Primero, lee ambos potenciómetros y calcula el error. Se implementa una "zona muerta"(definida por `TOLERANCIA`) donde el error es lo suficientemente pequeño como para detener el motor, evitando oscilaciones y vibraciones cerca del punto objetivo. Si el error está fuera de esta zona, se determina la dirección y se aplica un PWM. Este PWM es çlampeado"(limitado) entre `PWM_MINIMO` y `PWM_MAXIMO`, asegurando que el motor tenga suficiente torque para arrancar, pero sin exceder una velocidad máxima segura. Finalmente, los datos se envían por serial.

```
int main(void){
    iniciar_uart(9600);
    iniciar_adc();
    iniciar_motor();
    iniciar_pwm();

    printf("ref,eje,pwm,sentido\n");

    while(1){
        uint16_t referencia = leer_adc(3);
        uint16_t posicion   = leer_adc(4);
        int16_t  error      = (int16_t)
            referencia - (int16_t)posicion;

        uint8_t pwm_salida = 0;
        const char *direccion = "STOP";

        if (error > -((int16_t)TOLERANCIA) &&
            error < (int16_t)TOLERANCIA){
            motor_parar();
            pwm_salida = 0;
            direccion = "STOP";
        } else {
            if (error > 0){
                motor_izquierda();
                direccion = "IZQ";
            } else {
                motor_derecha();
                direccion = "DER";
            }

            uint16_t magnitud = (error < 0) ?
                (uint16_t)(-error) : (uint16_t
                )error;
            uint16_t duty = magnitud;

            if (duty < PWM_MINIMO) duty =
                PWM_MINIMO;
            if (duty > PWM_MAXIMO) duty =
                PWM_MAXIMO;
            pwm_salida = (uint8_t)duty;
            ajustar_pwm(pwm_salida);
        }

        printf("%u,%u,%u,%s\n", referencia,
            posicion, pwm_salida, direccion);
        _delay_ms(50);
    }
}
```

Listing 9: Bucle principal y lógica de control

Luego de terminado el código y documentado, se pasó a la etapa de simulación en el software Proteus para comprobar el funcionamiento de la lógica de control, como se observa en la Fig. 9.

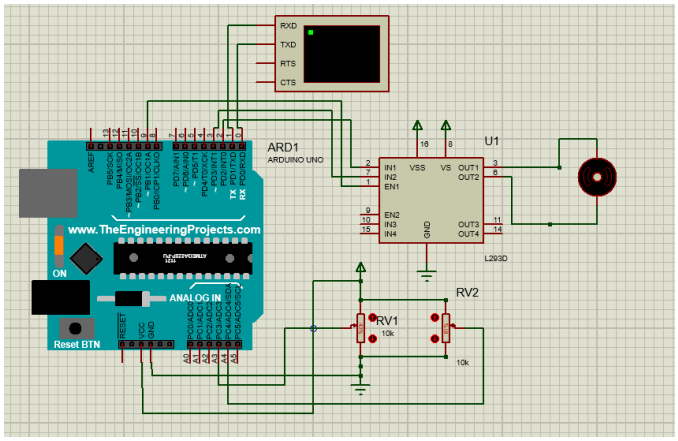

Fig. 9: Circuito de control de motor simulado en Proteus

En cuanto a la simulación, se presentó una limitación fundamental: el software Proteus no permite simular el acoplamiento físico de un motor de CC moviendo el eje de un potenciómetro para cerrar el lazo de control de forma automática. Sin embargo, al mover manualmente el potenciómetro de "posición"(conectado a A4), se verificó que la lógica de control responde correctamente: el sistema lee la diferencia de voltaje, ajusta la señal PWM en el pin D9 y activa los pines de dirección (`IN1`, `IN2`) tal como se esperaba, validando la lógica del programa. La salida serial también funcionó correctamente.

Una vez realizada la simulación lógica, se procedió con el montaje físico, como se muestra en la Figura 10.

En el montaje físico, el sistema funcionó correctamente en todos sus aspectos. Al variar el potenciómetro de referencia, el motor reaccionaba inmediatamente, girando en la dirección necesaria hasta que el potenciómetro acoplado a su eje alcanzaba una lectura de ADC similar a la de referencia (dentro del rango de `TOLERANCIA`), momento en el cual el motor se detenía exitosamente.

Para validar el comportamiento dinámico del sistema y analizar los datos enviados por el ATmega328P, se capturaron los datos seriales (referencia, posición y PWM) durante una prueba física. Estos datos se procesaron utilizando un script de Python (cuyo código se adjunta en el repositorio de GitHub del

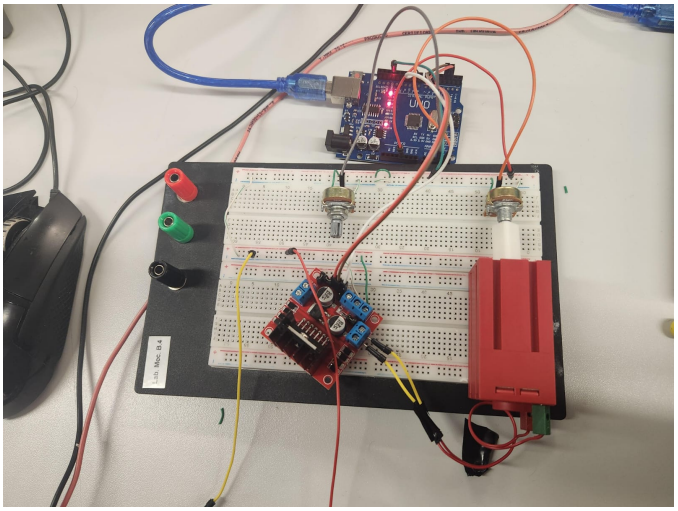

Fig. 10: Circuito de control de motor en Físico

proyecto) para generar la gráfica que se muestra en la Fig. 11. En ella, se observa cómo la señal "Potenciómetro Motor"(línea verde) sigue exitosamente a la señal "Potenciómetro"(línea azul), que representa la referencia. La señal "PWM"(línea roja) muestra la actividad del control: se activa (con valores limitados por `PWM_MINIMO` y `PWM_MAXIMO`) cuando existe un error entre la referencia y la posición, y se desactiva (valor 0) una vez que el motor alcanza la zona de tolerancia, demostrando el correcto funcionamiento del lazo de control.

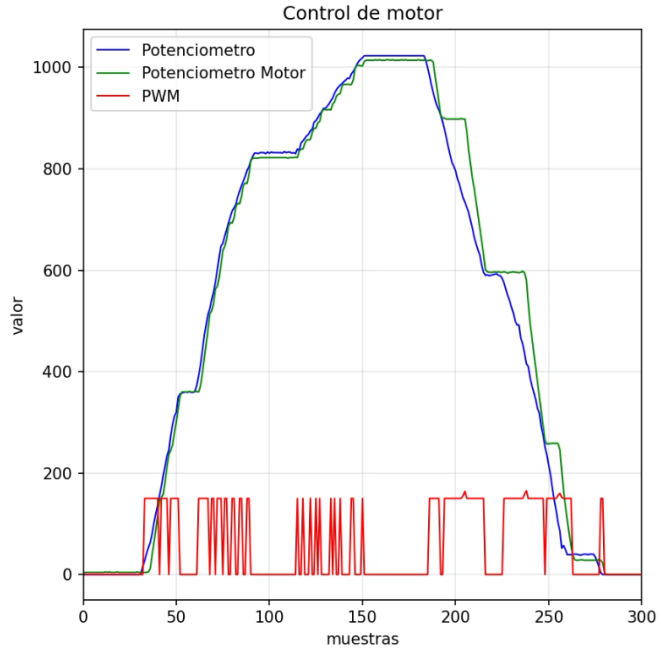

Fig. 11: Gráfica de telemetría del control de motor (Generada con Python).

VI-E. Matriz RGB 8×8 con ATmega328P y Joystick — Resultados

Con el código cargado en el microcontrolador, el sistema funcionó en equipo real: el píxel activo inicia cerca del centro y se desplaza según los ejes X/Y del joystick; al presionar el pulsador integrado, el color cambia aleatoriamente. La Fig. 12 muestra el montaje en protoboard, con inyección de 5V y masa común para la cadena WS2812. Y la Fig. 13 ilustra desde donde inícia el pixel.

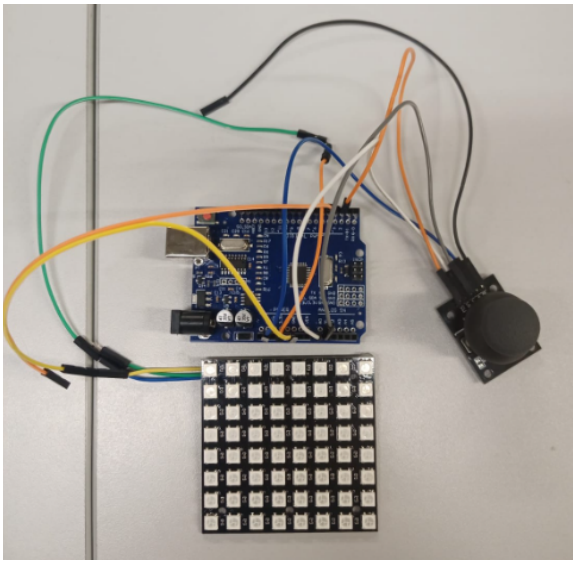

Fig. 12: Montaje físico: matriz WS2812 8×8, joystick, pulsador y ATmega328P (carga por bootloader USB–serial).

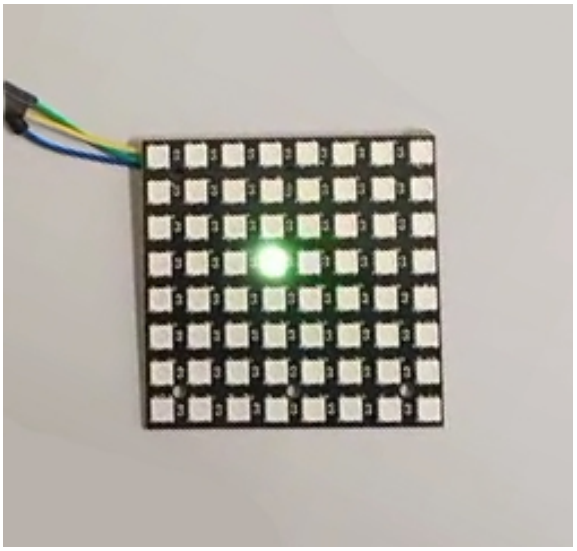

Fig. 13: Demostración: desplazamiento por joystick y cambio de color por pulsador.

VI-E0a. Comportamiento observado.: (i) La **deadzone** evitó "vibración" cerca del centro. (ii) El **cooldown** generó pasos discretos y controlados. (iii) El **debounce** por *software* suprimió disparos múltiples del pulsador. (iv) El **latch** de

$\sim$ 300 μs estabilizó la cadena WS2812 sin *glitches*. (v) Se evitó el negro absoluto en el arranque para asegurar visibilidad inmediata.

VI-E0b. Bloques de firmware (fragmentos en estilo del informe anterior).:

Listing 10: Inicialización y lectura ADC (AVcc, /128).

```
static void adc_init(void){
  ADMUX  = (1<<REFS0);
  ADCSRA = (1<<ADEN) | (1<<ADPS2)|(1<<ADPS1)|(1<<
    ADPS0);
}
static uint16_t adc_read(uint8_t ch){
  ADMUX = (ADMUX & 0xF0) | (ch & 0x0F);
  ADCSRA |= (1<<ADSC);
  while (ADCSRA & (1<<ADSC));
  return ADC; // 0..1023
}
```

Listing 11: Rutina de envío WS2812 y latch.

```
if(F_CPU == 8000000UL)
  defineCYCLES_1H 6
  defineCYCLES_1L 4
  defineCYCLES_0H 3
  defineCYCLES_0L 7
endif

staticinline void ws_send_byte(uint8_t b){
  for(uint8_t i=0;i<8;i++){
    if (b & 0x80){ /* '1' */ ... } else { /* '0' */
    ... }
    b <<= 1;
  }
}
static void ws_show(const uint8_t *grb, uint16_t n){
  uint8_t s=SREG; cli();
  for(uint16_t i=0;i<n*3;i++) ws_send_byte(grb[i]);
  SREG=s; _delay_us(300); // reset/latch
}
```

Listing 12: (x,y)→índice y escritura GRB.

```
static inline uint16_t index_xy(uint8_t x,uint8_t y)
    {
ifSERPENTINE
  return (y & 1) ? (y*LEDS_W + (LEDS_W-1 - x)) : (y*
    LEDS_W + x);
else
 return y*LEDS_W + x;
endif
}
staticinline void set_pixel(uint8_t x,uint8_t y,uint
    8_t r,uint8_t g,uint8_t b){
  uint16_t idx = index_xy(x,y)*3;
  leds[idx+0]=g; leds[idx+1]=r; leds[idx+2]=b;
}
```

Listing 13: Debounce por software y generador xorshift32.

```
static bool btn_pressed(void){
  if (!(PIND & (1<<PD3))){ _delay_ms(25); return !(
    PIND & (1<<PD3)); }
  return false;
}
static uint32_t rng_state = 0xC0FFEEu;
static inline uint32_t xorshift32(void){
  uint32_t x=rng_state; x^=x<<13; x^=x>>17; x^=x<<5;
    return rng_state=x;
}
```

Listing 14: Filtrado de movimiento y refresco de cuadro.

```
defineDEADZONE          120
defineMOVE_COOLDOWN_MS 120
...
uint8_t x=3,y=3; /* color inicial no-negro */
for(;;){
  uint16_t ax=adc_read(0), ay=adc_read(1);
  int16_t dx=(int16_t)ax-512, dy=(int16_t)ay-512;
  if (ms - last_move_ms >= MOVE_COOLDOWN_MS){
    if (dx >  DEADZONE && x<LEDS_W-1) x++;
    else if (dx < -DEADZONE && x>0)   x--;
    if (dy < -DEADZONE && y>0)        y--;
    else if (dy >  DEADZONE && y<LEDS_H-1) y++;
    /* refresco si hubo movimiento */
  }
  if (btn_pressed()){ /* nuevo color aleatorio y
    refresco */ }
}
```

VII. Conclusiones

En cuanto al sistema de control del plotter, se logró un rendimiento exitoso al realizar los trazados de las figuras predeterminadas (cruz, triángulo, círculo) y las nuevas siluetas (*Rana* y *Manzana*). El sistema de control de movimiento, utilizando los drivers *step/dir/enable* para los ejes X e Y y el solenoide para la herramienta de trazo, permitió un funcionamiento preciso sin pérdidas de paso, con un control efectivo de los límites de carrera mediante las interrupciones `INT0` e `INT1`.

El firmware, desarrollado con un reloj interno de 1 MHz, mantuvo la precisión necesaria para ejecutar las figuras sin errores, gracias a la correcta temporización de los pulsos `STEP` y a la protección por interrupciones externas. La implementación en el equipo real cumplió las expectativas, garantizando un trazo claro y sin saltos. En resumen, el sistema de control del plotter funcionó de manera estable y precisa, validando tanto la teoría como la implementación práctica.

Para el apartado de control de motor DC, se concluye que el sistema sobre el ATmega328P cumple con los objetivos principales de control. Se implementó exitosamente la lectura de velocidad mediante el potenciómetro (ADC), la generación de una señal `Fast PWM` correspondiente para gobernar el motor vía el L293D, y la gestión del arranque, detención y cambio de sentido de giro mediante pulsadores. La interfaz LCD informa adecuadamente el estado actual (porcentaje de velocidad y dirección).

Respecto a la simulación, el comportamiento observado en Proteus fue coherente con el montaje físico: el ciclo de trabajo del `PWM` respondió correctamente a la variación del potenciómetro y los controles de estado (arranque/parada/giro) operaron según lo esperado, validando el lazo de control principal ADC-PWM.

En síntesis, el prototipo valida la solución de control, demostrando que cumple sus funciones centrales. No obstante, se identificó que la lógica para establecer una `velocidad_máxima` y `velocidad_mínima` personalizadas no funciona correctamente y no limita el motor como se esperaba.

Para el sistema de control de la matriz RGB, el control del píxel activo mediante el joystick analógico funcionó correctamente, permitiendo que el píxel se desplace a lo largo de la matriz de forma fluida. La implementación del pulsador para cambiar el color de manera aleatoria también fue exitosa. El uso del protocolo WS2812, operando a 800kHz, garantizó una actualización rápida y precisa de los colores en la matriz sin causar problemas de flickering o distorsión.

El manejo de la deadzone evitó movimientos erráticos cerca del centro del joystick, y el cooldown controló adecuadamente la velocidad de los desplazamientos, lo que resultó en un comportamiento suave. El debounce del pulsador también se gestionó eficientemente, asegurando que no se registraran pulsaciones múltiples no deseadas. En resumen, el sistema de matriz RGB 8×8 con joystick funcionó como se esperaba, con un control fluido del píxel y una actualización precisa del color, validando el uso del protocolo WS2812 y la interacción con el joystick.

En lo que respecta al apartado de la cerradura electronica inteligente con RFID (RC522), se puede concluir que el montaje sobre el ATmega328P satisface los requerimientos de la propuesta. Se implementó de forma robusta la lectura de las tarjetas, el almacenamiento seguro del ID maestro en la memoria EEPROM y la lógica de comparación para conceder o denegar el acceso.

Además, se validó la funcionalidad de los botones dedicados, permitiendo tanto la actualización como el borrado de la tarjeta autorizada en la memoria.Los distintos elementos de retroalimentación (pantalla LCD, indicadores LED y comunicación UART) operan de forma sincronizada, informando al usuario el estado del acceso en todo momento.

Respecto a la simulación, el comportamiento observado en Proteus validó la lógica de control principal (Lector - EEPROM - Lógica de Acceso), mostrando una total coherencia con el funcionamiento del prototipo físico.

En cuanto al sistema de control de temperatura, se concluye que el sistema implementado sobre el ATmega328P satisface plenamente los requerimientos de regulación ambiental. Se logró implementar de forma efectiva la adquisición analógica de la temperatura (ADC), la generación de señales PWM para gobernar los actuadores (calefactor y ventilador) y una lógica de control automático robusta basada en rangos de temperatura.

Adicionalmente, el sistema gestiona una comunicación UART bidireccional, permitiendo no solo el monitoreo en tiempo real, sino también la reconfiguración dinámica por parte del usuario del rango medio o "punto medio"deseado.

Respecto a la validación del sistema físico, el comportamiento observado en la gráfica de Pyserial (Fig. 8) demostró la correcta operación del lazo de control completo: se observó la activación automática del calefactor para alcanzar el rango deseado, seguida de la respuesta escalonada del ventilador (BAJO, MEDIO, ALTO) al incremento de temperatura. Finalmente, se verificó la desactivación progresiva del ventilador conforme la temperatura descendía de nuevo al rango medio establecido.

En conjunto, todos los sistemas implementados y probados validaron el correcto uso del ATmega328P en aplicaciones mecatrónicas prácticas. Los sistemas de control de motor, plotter y matriz RGB proporcionaron una base sólida para aplicaciones de automatización y control digital, integrando efectivamente sensores, actuadores, comunicación serial y gestión de memoria en un entorno de sistemas embebidos.

VIII. Anexos

1. Link al Repositorio de GITHUB: https://github.com/juanferreiram-cell/Tec.Microprocesamiento
2. Link a la Carpeta de Drive con los videos: https://drive.google.com/drive/folders/1dfyBWJg07bqZ1Sa87PI9gOdNfCK04ClB?usp=sharing

IX. Bibliografía

- Administrador. (2018, 17 enero). LM35 - El sensor de temperatura más popular. HeTPro-Tutoriales. https://hetpro-store.com/TUTORIALES/lm35/
- colaboradores de Wikipedia. (2025, November 5). Puente H (electrónica). Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Puente_H_(electr%C3%B3nica)
- colaboradores de Wikipedia. (2025, May 15). Motor paso a paso. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Motor_paso_a_paso
- colaboradores de Wikipedia. (2025, September 5). Potenciómetro. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Potenci%C3%B3metro
- De La Cruz, W. C. (2025, October 20). El programador AVR Atmel/Microchip USBASP. Apuntes De Walther Curo. https://blog.walthercuro.com/el-programador-avr-atmel-microchip-usbasp/
- Modulo Joystick analógico con pulsador - PatagoniaTec. (s.f.). PatagoniaTec. https://tienda.patagoniatec.com/productos/modulo-joystick-analogico-con-pulsador/
- PySerial - Python Wiki. (s. f.). https://wiki.python.org/moin/PySerial
- Tecnipesa. Soluciones de Marcaje, Etiquetado y Codificación de almacenes. (s. f.). Qué es y cómo funciona la tecnología RFID. TECNIPESA. https://www.tecnipesa.com/blog/69-tecnologia-rfid-que-ventajas-tiene