



**INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE
MONTERREY**

Diseño de sistemas embebidos avanzados

Profesor: Alfonso Ávila Ortega

EVIDENCIA FINAL _ REPORTE

Equipo 5

Integrantes:

Felipe de Jesús García García	A01705893
Alfonso Solís Díaz	A00838034
Jesús René Hernández Galindo	A00837617
Juan José Castillo González	A01750541

30 de noviembre del 2024

Visión

El objetivo principal del sistema es proporcionar una solución automatizada para la agricultura de precisión, capaz de realizar tareas de navegación autónoma y comunicación avanzada entre componentes. Este proyecto busca cerrar la brecha entre las tecnologías actuales y las aplicaciones prácticas en el sector agrícola, fomentando prácticas sostenibles y eficientes.

Necesidades

La agricultura moderna enfrenta desafíos como el uso ineficiente de recursos, la necesidad de reducir la intervención humana y la falta de sistemas que puedan adaptarse a diversas condiciones del terreno. Este sistema pretende resolver estos problemas mediante la integración de tecnologías avanzadas que permitan una navegación precisa, decisiones en tiempo real y optimización de recursos.

Público objetivo

El sistema está diseñado para beneficiar a agricultores, empresas dedicadas a la agricultura de precisión y desarrolladores de tecnologías agrícolas. Proporciona una herramienta que mejora la productividad, reduce costos operativos y facilita la toma de decisiones basada en datos.

Solución Propuesta

El producto consiste en un tractor a escala equipado con tecnologías embebidas avanzadas, incluyendo sensores IMU, módulos de comunicación (NRF24, CAN), y motores controlados mediante PWM. En lugar de utilizar un sistema de tiempo real como FreeRTOS, se implementaron rutinas específicas de control que permiten realizar las siguientes tareas:

- 1. Ruta Hardcoding con Delays:** La programación fija permitió establecer trayectorias predefinidas utilizando delays para sincronizar el movimiento del tractor entre puntos.
- 2. Llegada a un Punto con NRF:** Se integró un módulo NRF24L01 para determinar la llegada precisa a un waypoint mediante comunicación inalámbrica.
- 3. Ruta Hardcoding con Encoder:** El encoder del motor se empleó para calcular distancias recorridas, asegurando una mejor precisión en trayectorias predefinidas.
- 4. Ruta Completa:** Combinando los métodos anteriores, se logró una navegación eficiente por toda la trayectoria preestablecida.

Este enfoque combina técnicas de programación directa y sincronización basada en hardware, eliminando la necesidad de un sistema operativo de tiempo real. La solución sigue siendo efectiva para simular el comportamiento de un tractor inteligente a escala real.

Listado de tareas/acciones realizadas utilizando el modelo V

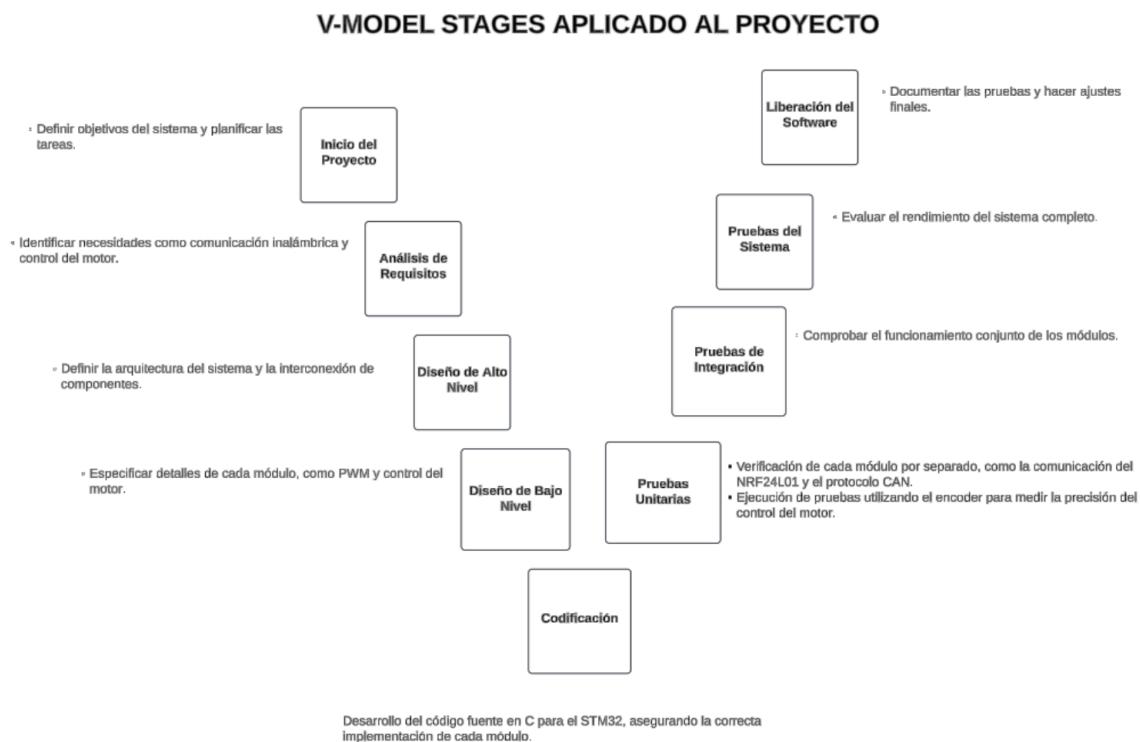


Figura 1. Diagrama modelo V

- **Análisis de Requerimientos:**

- Elaboración de un diagrama de contexto para definir los límites del sistema.
- Creación de una tabla de requerimientos detallada, especificando hardware, software y funcionalidad.

- **Diseño de Alto y Bajo Nivel:**

- Diseño del diagrama de despliegue para identificar los componentes principales del sistema y sus interacciones.
- Desarrollo de diagramas de actividad para representar los flujos de comunicación entre módulos (CAN, UART, I2C, etc.).
-

- **Implementación:**
 - Ensamble del tractor a escala con los componentes electrónicos y mecánicos.
 - Configuración de la interfaz UART para la comunicación entre el STM32 y el buzzer.
 - Integración del sensor IMU con el protocolo I2C y el módulo de comunicación inalámbrica NRF24L01 con SPI.
- **Pruebas:**
 - Ejecución de pruebas unitarias para cada módulo (e.g., transmisión de datos con UART, lectura de sensores con I2C).
 - Pruebas de integración entre subsistemas, como el envío de datos del encoder al STM32 mediante CAN.
 - Validación del sistema completo mediante un recorrido del tractor siguiendo waypoints predefinidos.

Actividades Realizadas

El proyecto abarcó un período de 15 semanas, durante el cual se llevaron a cabo las siguientes acciones clave para garantizar la integración, comunicación y funcionamiento eficiente de los sistemas del tractor a escala:

1. Estudio inicial del STM32H755:

Familiarización con las capacidades del microcontrolador, incluyendo la programación de sus dos núcleos y la verificación mediante LEDs internos.

2. Pruebas iniciales de UART:

Configuración básica del UART para la transmisión y recepción de mensajes entre módulos, validada mediante un osciloscopio para asegurar la integridad de los datos.

3. Integración de NRF24:

Configuración del módulo de comunicación inalámbrica utilizando SPI, ajustando direcciones de comunicación, tamaño de paquetes y tasas de transferencia para asegurar estabilidad.

4. Control de motores con PWM:

Generación de señales PWM precisas para controlar la dirección con un servomotor y la tracción con un motor DC, ajustando la velocidad según las necesidades de la trayectoria.

5. Implementación de CAN:

Configuración del protocolo CAN para permitir la comunicación entre nodos como el Arduino y el STM32, utilizando transceptores TJA1051 y módulos MCP2515.

6. Calibración de IMU:

Configuración y calibración del sensor MPU6050 mediante I2C, logrando lecturas precisas del acelerómetro y giroscopio para la estimación de la posición.

7. Ensambles finales:

Construcción del tractor con integración mecánica y electrónica, asegurando que cada componente esté conectado y configurado de acuerdo a los requisitos del sistema.

8. Pruebas de integración:

Validación conjunta de los módulos (NRF24, IMU, motores, CAN y UART) para asegurar sincronización y funcionalidad en un escenario realista.

Funcionalidades Técnicas

- **UART:**

Configuración para enviar y recibir datos entre el STM32 y periféricos, usando herramientas como el osciloscopio para verificar la transmisión correcta.

- **Comunicación inalámbrica (NRF24):**

Configuración del módulo para asegurar comunicación bidireccional con tasas de datos confiables y direcciones adecuadas.

- **Protocolo SPI:**

Configuración del STM32 para manejar la transferencia de datos con el módulo NRF24, ajustando parámetros de velocidad y sincronización.

- **Control de dirección y tracción (PWM):**

Uso de temporizadores del STM32 para generar señales PWM que ajusten la dirección del tractor mediante el servomotor y controlen la velocidad del motor de tracción.

- **Protocolo CAN:**

Configuración y prueba de mensajes CAN entre el STM32 y otros nodos, validando el envío de comandos y retroalimentación.

- **I2C para IMU:**
Integración y calibración del sensor MPU6050, obteniendo lecturas precisas de movimiento para guiar al tractor.
- **Pruebas del sistema completo:**
Validación de todas las funcionalidades bajo condiciones simuladas, asegurando que el tractor navegue de forma autónoma entre puntos predefinidos.

Validación del Sistema

Pruebas realizadas:

Prueba	Objetivo	Resultado Esperado	Resultado Obtenido
Prueba de UART	Validar la transmisión de datos entre el STM32 y el buzzer.	Comunicación fluida y sin errores.	Comunicación validada correctamente.
Prueba de NRF24	Asegurar la transmisión/recepción inalámbrica de datos.	Estabilidad en la conexión.	Comunicación inalámbrica exitosa.
Prueba de SPI	Verificar la sincronización del NRF24 con el STM32.	Sincronización sin pérdida de datos.	Comunicación SPI establecida.
Prueba de PWM	Controlar la dirección y velocidad del tractor.	Respuesta precisa a señales PWM.	Control exitoso de motores.
Prueba de IMU	Calibrar y obtener datos precisos del acelerómetro y giroscopio.	Datos consistentes en diferentes pruebas.	Datos precisos y estables.
Prueba de CAN	Comprobar la transmisión de mensajes entre nodos.	Mensajes transmitidos sin interferencias.	Comunicación CAN funcional.
Prueba de Integración	Verificar la interacción entre todos los módulos del sistema.	Sincronización perfecta entre componentes.	Interacción integrada sin fallos.
Prueba del Sistema	Evaluuar la navegación del tractor en condiciones reales.	Trayectoria completa y precisa.	Navegación satisfactoria.

Listado de tareas con responsables

Tarea	Responsable	Clasificación
Elaboración de diagrama de contexto	Alfonso	Análisis de Requerimientos
Creación de tabla de requerimientos	Jesús René	Análisis de Requerimientos
Diseño del diagrama de despliegue	Juan José	Diseño de Alto Nivel
Desarrollo de diagramas de actividad	Juan José	Diseño de Alto Nivel
Ensamble del tractor	Jesús René	Implementación
Configuración del módulo NRF24L01	Alfonso Solis	Implementación
Configuración de IMU con I2C	Alfonso	Implementación
Pruebas unitarias de UART	Alfonso- Juan José	Pruebas
Validación del sistema completo	Todo el equipo	Pruebas

Diagrama de Gantt

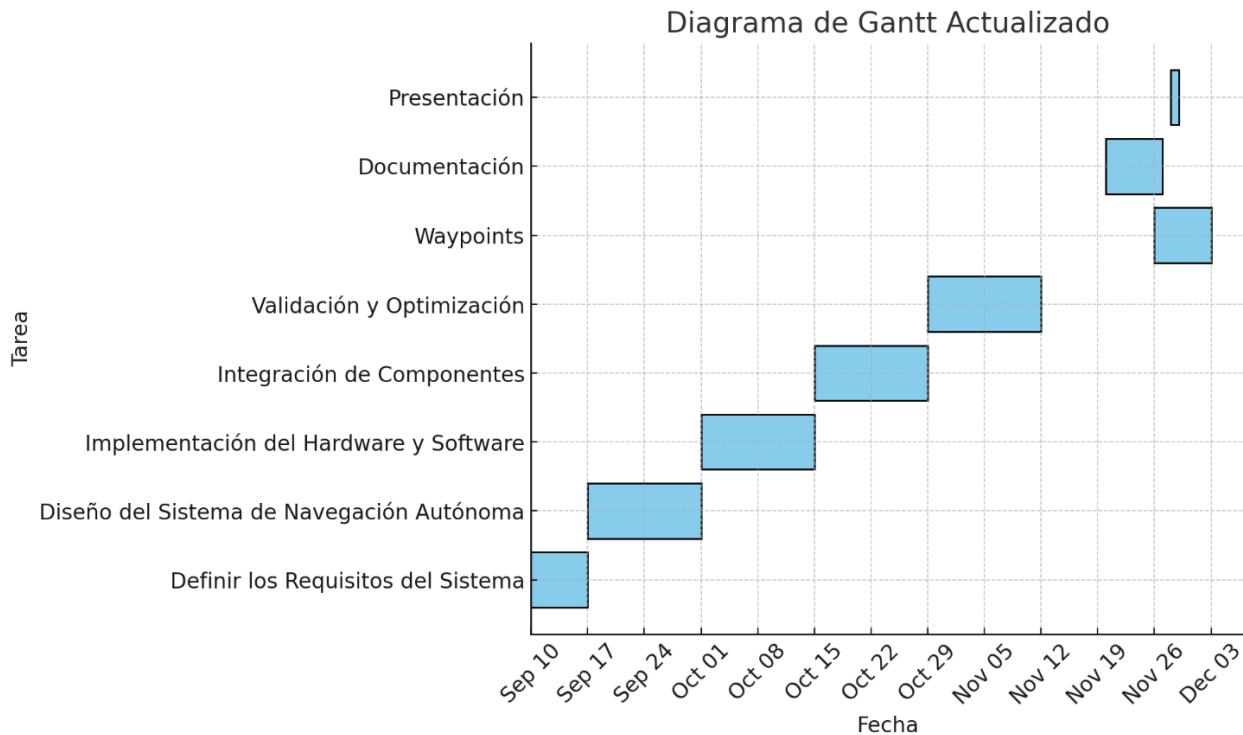


Figura 2. Diagrama de Gantt

Reflexión sobre la distribución del trabajo

La distribución de tareas se realizó considerando las habilidades técnicas y la experiencia previa de cada integrante. Por ejemplo, Felipe asumió la parte de la creación de la carcasa más la base para cada módulo que íbamos ocupar para nuestro proyecto gracias a conocimiento en modelación 3D. Juan se encargó del armado del tractor y de la lógica detrás del recorrido final del tractor, gracias a su experiencia de estar en una escudería y en un equipo de robótica. Jesús y Alfonso asumieron la tarea de los módulos de comunicación porque su perfil se ajustaba a las necesidades del proyecto, desarrollando soluciones basadas en protocolos como CAN, I2C, además de resolver los problemas que se fueron encontrando a lo largo del proyecto en electrónica y mecanismos mecanismos. Esta división aseguró que cada tarea fuera asignada al miembro más capacitado, maximizando la eficiencia del equipo y garantizando la calidad del trabajo.

Reflexión sobre mejoras

Para mejorar la colaboración, el equipo propone implementar reuniones más frecuentes para revisar avances y redistribuir tareas en caso de imprevistos. Además, se sugiere documentar más detalladamente cada etapa del proyecto para facilitar la transición en caso de que un miembro no

pueda continuar con su tarea asignada. Estas acciones fortalecerán la resiliencia del equipo y permitirán una mejor respuesta a contingencias.

Diagrama de contexto

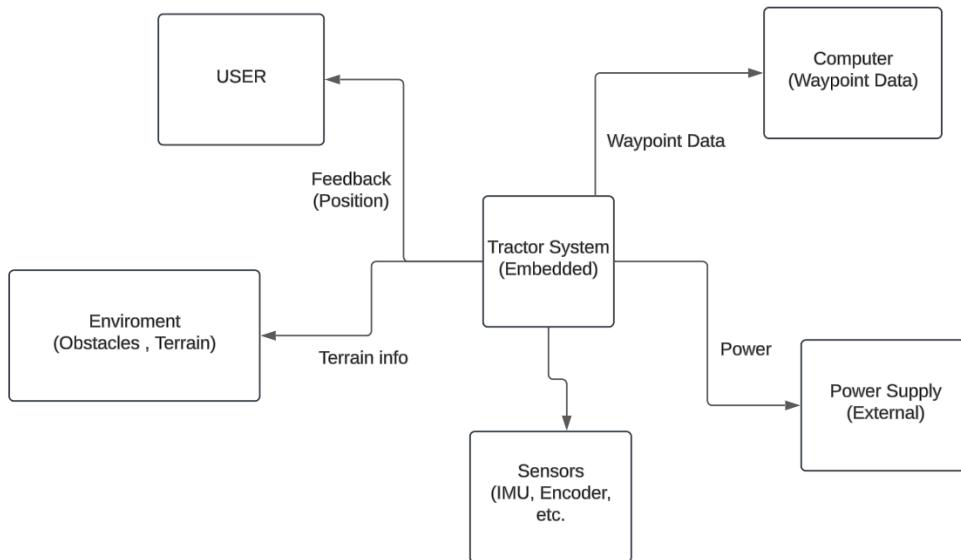


Figura 3. Diagrama de contexto de la Situación problema

Tabla de requerimientos

Nombre	Tractor autónomo con navegación por waypoints
Propósito	Implementar un sistema de tractor autónomo que navegue por una ruta predefinida minimizando la intervención humana y optimizando operaciones agrícolas.
Entradas	GPS (datos de cámara LPS), datos del encoder del motor, datos del acelerómetro y giroscopio (IMU), mensajes CAN (protocolo L1939), comandos UART.
Salidas	Movimientos precisos del tractor, alertas sonoras (buzzer) en waypoints, transmisión de datos inalámbrica (NRF24L01).
Funciones	Las funciones principales funciones que requiere el tractor para tener un desempeño óptimo, son la recepción de datos de la NRF24, el uso del ESC para la potencia de torque y el ángulo del servo motor, las comunicaciones entre dispositivos, mediante CAN y I2S. y el respaldo de trayectoria que se realiza con el encoder.

Comunicación de datos	<ul style="list-style-type: none"> - Interactuar con el módulo SPI para recibir datos del GPS.
	<ul style="list-style-type: none"> - Verificar y procesar los datos del encoder y el IMU mediante I2C y CAN.
	<ul style="list-style-type: none"> - Transmitir datos procesados al usuario a través de UART.
Control del motor	<ul style="list-style-type: none"> - Controlar el motor utilizando señales PWM configuradas en los pines TIM13 y TIM14 del STM32.
	<ul style="list-style-type: none"> - Ajustar la dirección del tractor con el servo mediante el canal PA6.
Señalización	<ul style="list-style-type: none"> - Activar el buzzer conectado al Arduino al alcanzar un waypoint utilizando el Pin 9.
	<ul style="list-style-type: none"> - Enviar alertas de error al usuario por UART en caso de condiciones no previstas.
Rendimiento	
Tiempo de respuesta	<ul style="list-style-type: none"> - Latencia de procesamiento de menos de 100 ms para control de motores y lectura de sensores.
Frecuencia de datos	<ul style="list-style-type: none"> - Envío de datos cada 500 ms mediante NRF24L01 para monitoreo preciso de waypoints.
Conexiones Principales	<p>Las principales conexiones que se realizaron son a los periféricos de la STM32:</p> <p>ESC:</p> <ul style="list-style-type: none"> • Señal (PWM): TIM14_CH1 → PA7 <p>Servo:</p> <ul style="list-style-type: none"> • Señal (PWM): TIM13_CH1 → PA6 <p>NRF24L01:</p> <ul style="list-style-type: none"> • CE: PC6. • CSN: PC7. • SCK: PF7 (SPI5). • MOSI: PF9 (SPI5). • MISO: PF8 (SPI5).

	<p>CAN Bus (con transceptor MCP):</p> <ul style="list-style-type: none"> • CAN_RX: PD0. • CAN_TX: PD1. <p>MPU:</p> <ul style="list-style-type: none"> • SCL: PF14. • SDA: PF15.
Regulador	Entrada: Cable rojo (+), Cable gris (-). Salida: Cable azul, Cable naranja.
ESC	<ul style="list-style-type: none"> - Alimentación: Cable negro (-) al regulador, cable rojo (+) al regulador.
	<ul style="list-style-type: none"> - Control PWM: Cable blanco conectado a TIM14-PA7, jumper entre CN10 y CN7.
Servo	<ul style="list-style-type: none"> - Control PWM: Cable naranja conectado a TIM13-PA6, segundo jumper soldado entre CN10 y CN7.
	<ul style="list-style-type: none"> - Alimentación: Cable rojo (+5V), cable café (GND).
NRF24L01	<ul style="list-style-type: none"> - Alimentación: VCC (3.3V), GND.
	<ul style="list-style-type: none"> - SPI: CE (PC6), CSN (PC7), SCK (PF7), MOSI (PF9), MISO (PF8).
MPU/STM32	<ul style="list-style-type: none"> - Alimentación: VCC (5V), GND.
	<ul style="list-style-type: none"> - Comunicación I2C: SDA (PF15), SCL (PF14).
Encoder	<ul style="list-style-type: none"> - Fase A: Digital Pin 3, Fase B: Digital Pin 8.
	<ul style="list-style-type: none"> - Alimentación: Cable azul (5V), cable negro (GND).
Fabricación	Todos los componentes y elementos del carrito fueron adquiridos, el equipo se encargó de su configuración y ensamblaje. Únicamente la carrocería del tractor se fabricó a partir de madera MDF.
Costos Estimados	<ul style="list-style-type: none"> - STM32H755 Microcontroller: \$1,200 MXN.

	- MPU6050 Sensor: \$64 MXN.
	- NRF24L01 Module: \$350 MXN.
	- CAN Transceiver (TJA1051): 80 MXN.
	- ESC (Electronic Speed Controller): \$110 MXN.
	- Servo Motor: \$350 MXN.
Energía	Alimentación mediante batería recargable Li-Po de 6V para operación autónoma durante 4 horas.
Dimensiones	Longitud: 29.5 cm, Ancho: 14.8 cm, Altura: 16.2 cm, Peso: 1.2 kg.

Diagrama de despliegue

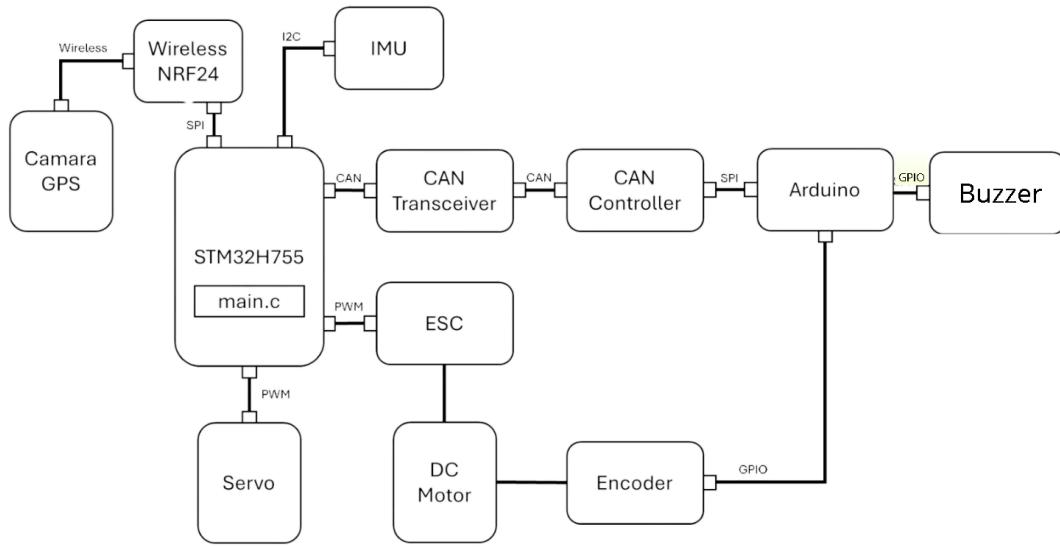


Figura 4. Diagrama de despliegue de actividades

Diagrama de actividad

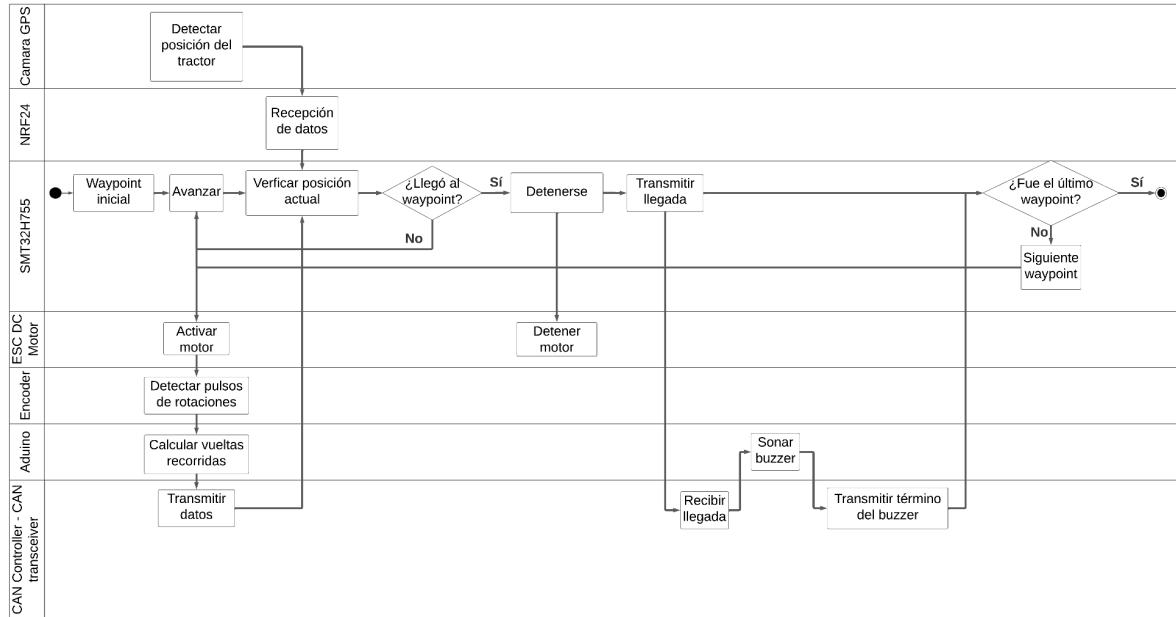


Figura 5. Diagrama de actividad

Selección de periféricos

- **NRF24:** Este era indispensable para poder comunicarse con la cámara, en realidad no había opción. Con él recibimos las coordenadas del tractor para así ajustar la dirección y saber cuando llega a los waypoints.
- **CAN Controller (MCP2515):** Gestiona los mensajes lógicos del bus CAN, recibiendo datos del Arduino (como las vueltas recorridas) y enviándolos al bus a través del CAN Transceiver. También procesaba señales de llegada desde la STM32H755 y las transmitía al bus CAN.
- **ESC (Electronic Speed Controller) :** Era el responsable de una de las tareas más esenciales de este reto, que era controlar la velocidad del motor, esto lo hacía gracias a la recepción de las señales PWM del Timer 14 de la STM32, que ajustan la cantidad de potencia entregada al motor que ajustan la cantidad de potencia entregada al motor.
- **MPU2050:** Se encarga de estimar la orientación, aceleración y velocidad angular del tractor. En la implementación no se tomó en cuenta ya que tenía ruido al estimar y dependía de la velocidad del motor.
- **Servomotor:** Otro de los pilares de la implementación. Tuvimos que escoger un motor con suficiente potencia como para mover ambas llantas del tractor aún cuando este estuviese en movimiento, además de que nos permitiera un rango lo más amplio posible de giro para poder llegar a cualquier punto de la pista.
- **Encoder:** El encoder fue de gran utilidad para tener un respaldo al NRF, con este calculamos la cantidad de vueltas necesarias para llegar a los waypoints y evitar que se pase en caso de no recibir las coordenadas del NRF, lo utilizamos sobre el MPU para esta función debido a su alta consistencia, ya que no se ve afectado por su uso ni su entorno, ya que una rotación siempre será una rotación independientemente de la velocidad o aceleración que lleve.
- **Buzzer:** Optamos por usar un buzzer ya que era la forma más clara y sencilla posible de marcar la llegada a los waypoints, lo preferimos sobre un led ya que este puede ser difícil de percibir en ciertas ocasiones.

Implementación del sistema

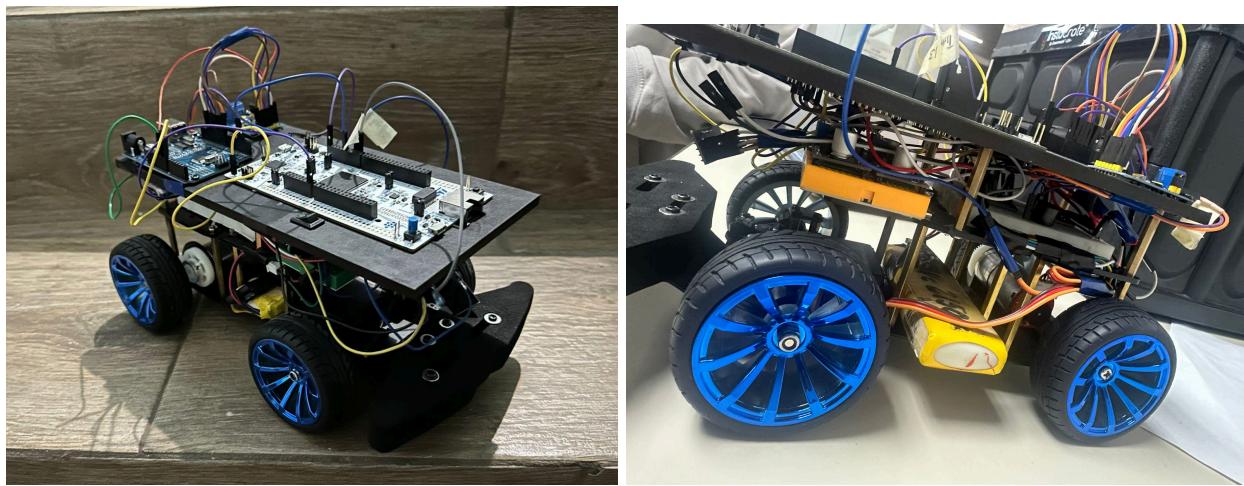


Figura 6. Implementación general

Subsistemas y Periféricos Utilizados

1. PWM (Pulse Width Modulation)

Configuración: Utilizando los canales TIM13_CH1 y TIM14_CH1, configurados en los pines PA6 y PA7 respectivamente. La configuración permite generar señales PWM para controlar tanto el motor principal como el servo de dirección.

- **Frecuencia:** Configurada según las necesidades del motor y el servo.
- **Modos:** PWM1.
- **Aplicación:** Control de velocidad y ángulo del motor y servo.

A screenshot of the STM32CubeMX software interface. At the top, it says "GPIO Mode and Configuration". Below that is a "Configuration" tab. Under "Group By Peripherals", there are checkboxes for GPIO, Single Mapped Signals, DEBUG, FDCAN, I2C, RCC, SPI, TIM (which is checked), and USART. In the "Search Signals" section, there's a search bar with "Search (Ctrl+F)" and a "Show only Modified Pins" checkbox. The main part of the screen is a table with columns: Pin Name, Signal on Pin, Pin Context, GPIO output..., GPIO mode, GPIO Pull-up..., Maximum out., Fast Mode, User Label, and Modified. Two rows are shown: PA6 with Signal on Pin "TIM13_CH1", Pin Context "ARM Cortex...", GPIO mode "Alternate Fun...", and Maximum out. "Low"; PA7 with Signal on Pin "TIM14_CH1", Pin Context "ARM Cortex...", GPIO mode "Alternate Fun...", and Maximum out. "Low". Both rows have a checked "Modified" column.

Figura 7. Configuración PWM

TIM13 Mode and Configuration

Mode

Runtime contexts:

Cortex-M7	Cortex-M4	PowerDomain D2
<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Activated

Channel1 **PWM Generation CH1**

One Pulse Mode

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

- Counter Settings

Prescaler (PSC - 16 bits value)	239
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	19999
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
- Clear Input

Clear Input Source	Disable
--------------------	---------
- PWM Generation Channel 1

Figura 8. Configuración del TIM13

TIM14 Mode and Configuration

Mode

Runtime contexts:

Cortex-M7	Cortex-M4	PowerDomain D2
<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Activated

Channel1 **PWM Generation CH1**

One Pulse Mode

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

- Counter Settings

Prescaler (PSC - 16 bits value)	239
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	19999
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
- Clear Input

Clear Input Source	Disable
--------------------	---------
- PWM Generation Channel 1

Figura 9. Configuración del TIM14

2. I2C (Inter-Integrated Circuit)

Configuración: Utilizando el bus I2C4, con los pines PF14 y PF15 configurados para las señales SCL y SDA.

- **Velocidad:** Configurado en modo estándar a 100 kHz.
- **Uso:** Comunicación con el sensor MPU6050 para obtener datos de acelerómetro y giroscopio.

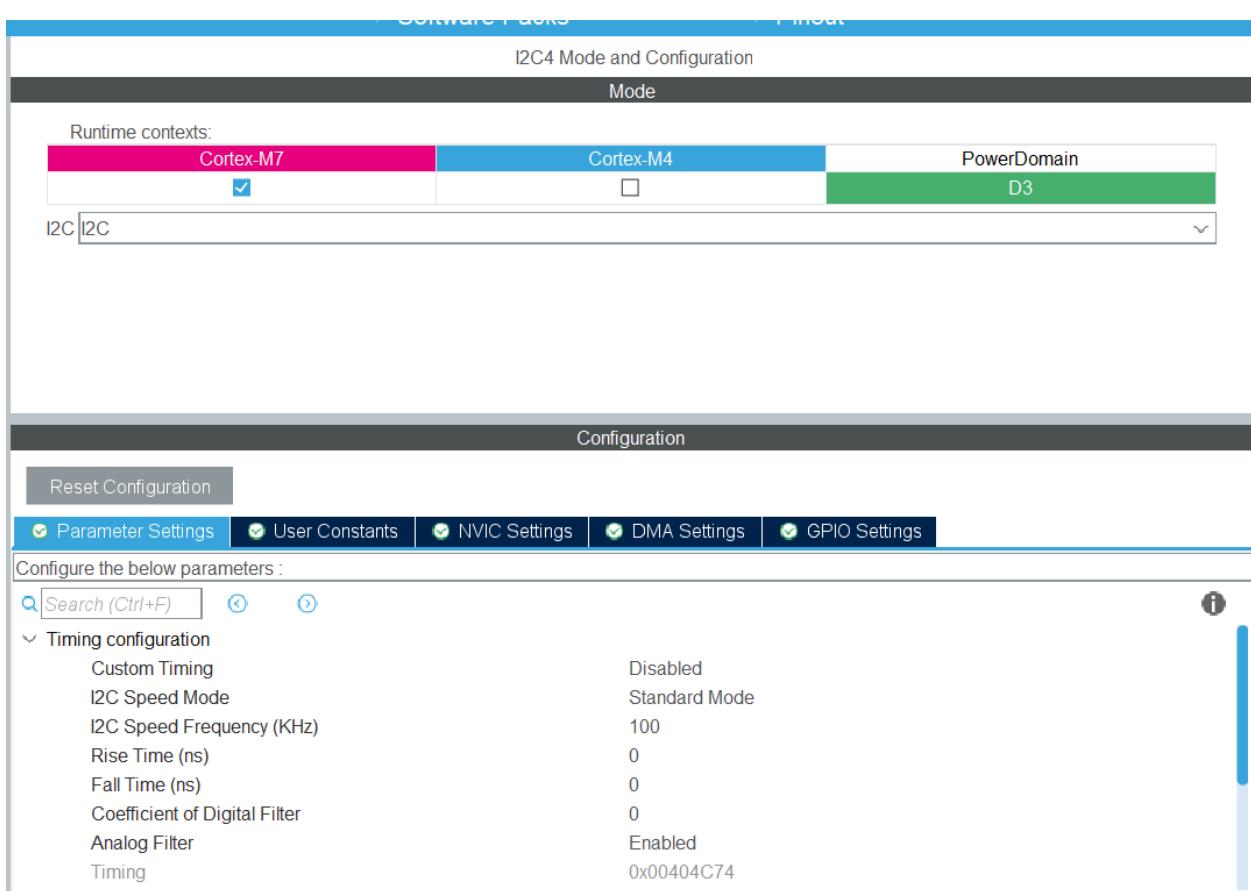


Figura 10. Configuración del I2C

3. SPI (Serial Peripheral Interface)

Configuración: Usando el periférico SPI5 para la comunicación con el módulo NRF24L01.

- **Pines configurados:**
 - SCK: PF7
 - MISO: PF8
 - MOSI: PF9
- **Modo:** Maestro.

- **Uso:** Transmisión y recepción de datos inalámbricos.
- **Velocidad de transferencia:** Configurada según las especificaciones del módulo NRF.

SPI5 Mode and Configuration

Mode

Runtime contexts: Cortex-M7 (checked), Cortex-M4 (unchecked), PowerDomain D2

Mode: Full-Duplex Master

Hardware NSS Signal: Disable

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings GPIO Settings

Configure the below parameters:

Search (Ctrl+F)

Basic Parameters

- Frame Format: Motorola
- Data Size: 8 Bits
- First Bit: MSB First

Clock Parameters

- Prescaler (for Baud Rate): 16
- Baud Rate: 4.0 MBit/s
- Clock Polarity (CPOL): Low
- Clock Phase (CPHA): 1 Edge

SPI5 Mode and Configuration

Mode

Runtime contexts: Cortex-M7 (checked), Cortex-M4 (unchecked), PowerDomain D2

Mode: Full-Duplex Master

Hardware NSS Signal: Disable

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings GPIO Settings

Search Signals

Show only Modified Pins

Pin Name	Signal on Pin	Pin Context	GPIO output	GPIO mode	GPIO Pull-up	Maximum out	Fast Mode	User Label	Modified
PF7	SP15_SCK	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		<input checked="" type="checkbox"/>
PF8	SP15_MISO	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		<input checked="" type="checkbox"/>
PF9	SP15_MOSI	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		<input checked="" type="checkbox"/>

Figura 11. Configuración del SPI

4. UART (Universal Asynchronous Receiver-Transmitter)

Configuración: Usando USART3 configurado con los pines PD8 y PD9 para TX y RX respectivamente.

- **Velocidad en baudios:** 115200.
- **Aplicación:** Debugging y visualización de datos en tiempo real.

USART3 Mode and Configuration

Mode

Runtime contexts: Cortex-M7 (checked), Cortex-M4 (checked), Initializer: Cortex-M7, PowerDomain: D2

Mode: Asynchronous

Hardware Flow Control (RS232): Disable

Hardware Flow Control (RS485):

Slave Select(NSS) Management: Disable

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings GPIO Settings

Configure the below parameters:

Search (Ctrl+F)

Basic Parameters

- Baud Rate: 115200 Bits/s
- Word Length: 8 Bits (including Parity)
- Parity: None
- Stop Bits: 1

Advanced Parameters

- Data Direction: Receive and Transmit
- Over Sampling: 16 Samples
- Single Sample: Disable

Figura 12. Configuración del UART

5. CAN (Controller Area Network)

Configuración: Utilizando FDCAN1 con los pines PD0 y PD1 configurados para RX y TX.

- **Aplicación:** Comunicación entre la STM32 y el Arduino para la sincronización del sistema.
 - **Filtro configurado:** Configurado para aceptar mensajes específicos del Arduino.
 - **Modo:** Normal.

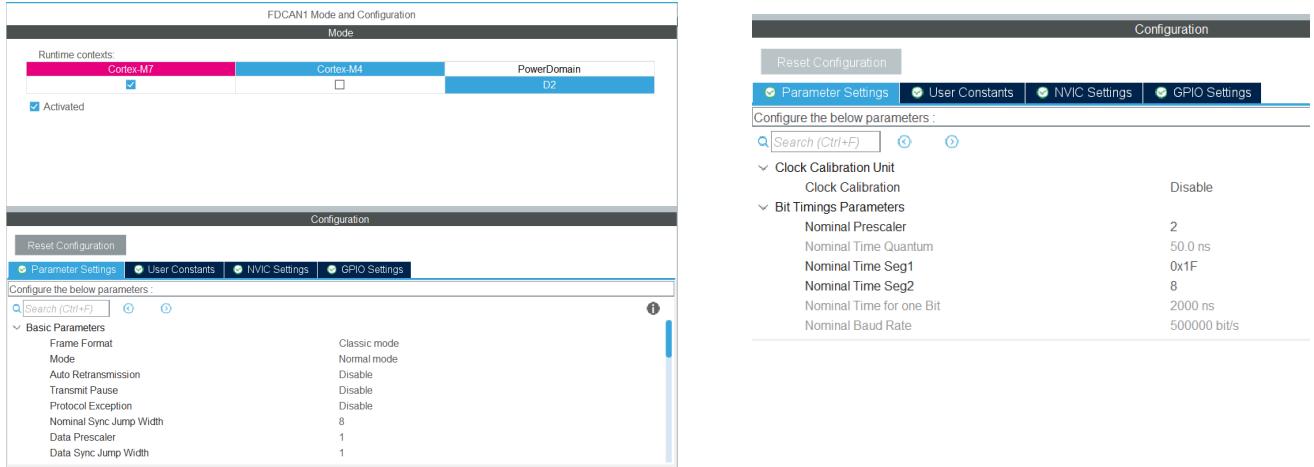


Figura 13. Configuración del CAN.

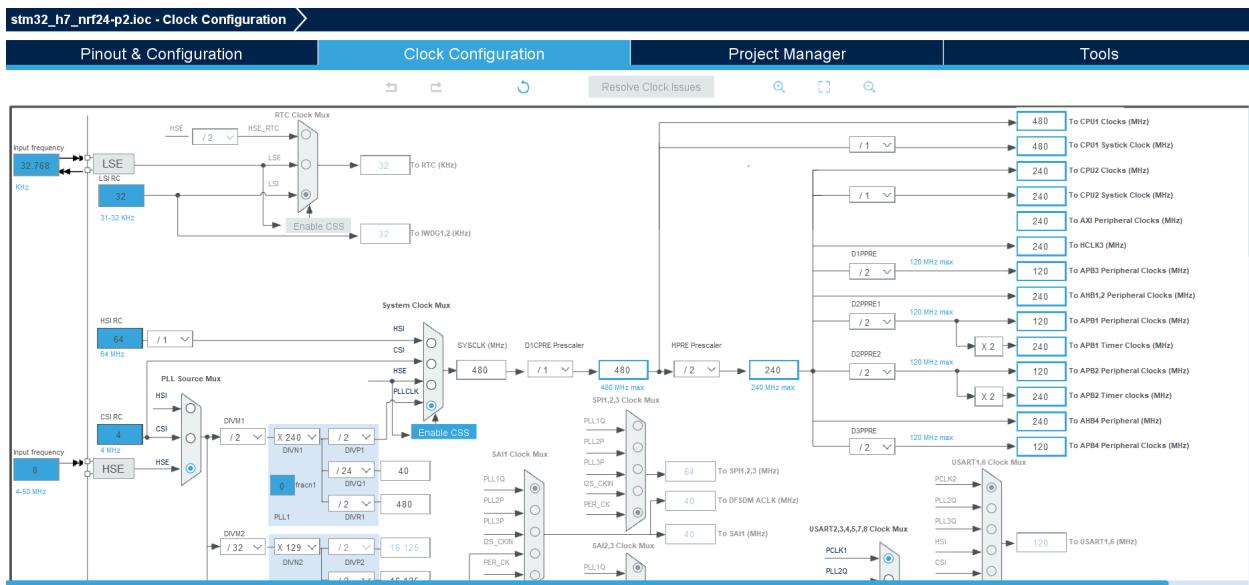


Figura 14. Configuración CLOCK 1

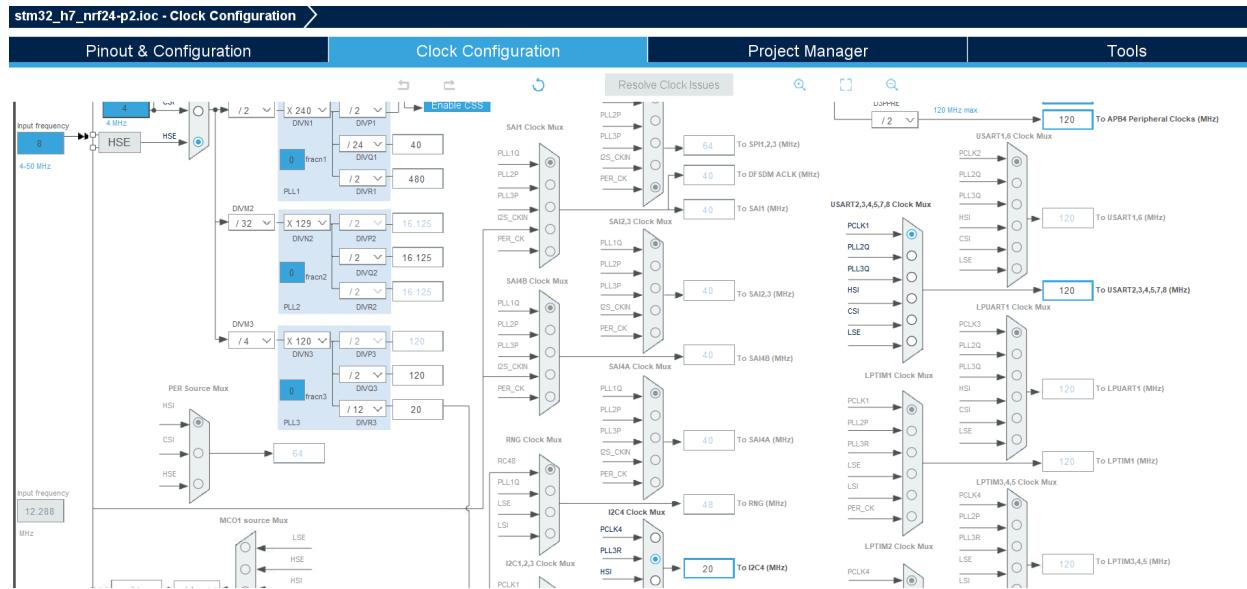


Figura 15. Configuración CLOCK 2

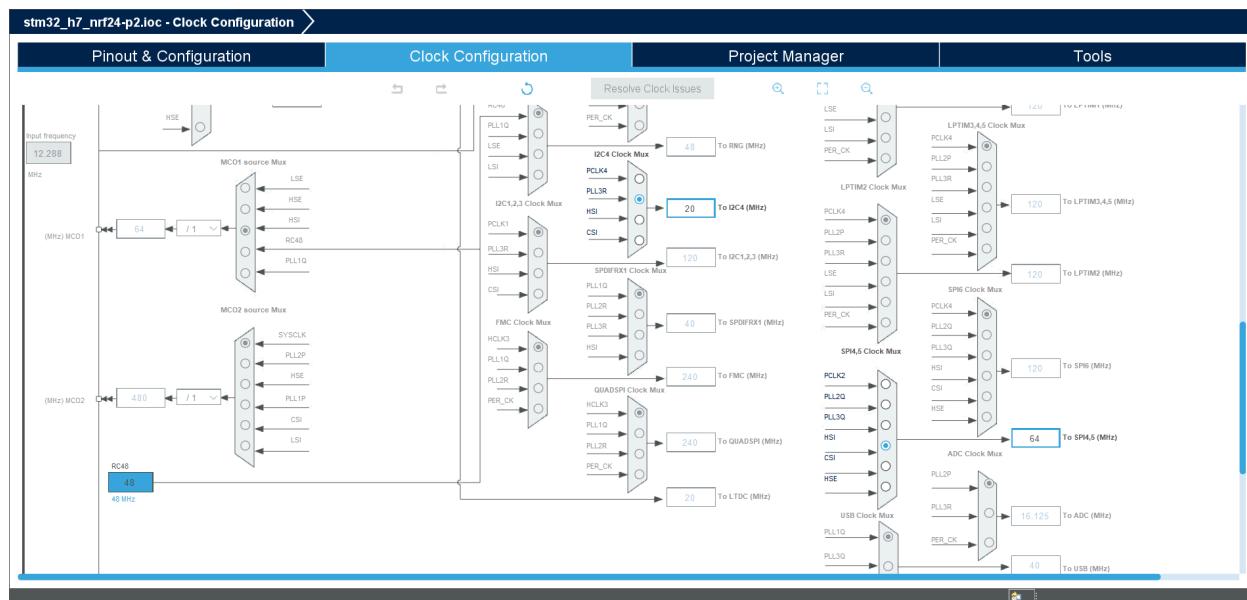


Figura 16. Configuración CLOCK 3

# Ítem	Nombre del Componente	Número de Parte	Cantidad	Precio Unitario (MXN)	Precio Total (MXN)	Descripción	Proveedor
1	Microcontrolador STM32H755	STM32H755	1	876.0	876.0	Microcontrolador de alto rendimiento con doble núcleo	STMicroelectronics
2	Motor DC con encoder de 12V	GA25-370	1	250.0	250.0	Motor DC de 12V con encoder integrado	Electrónica para estudiantes
3	Regulador Step-Down	LM2596	2	54.0	108.0	Convertidor buck para regulación de voltaje	Latch Electronics
4	Módulo de Radio	NRF24L01	1	59.0	59.0	Módulo de comunicación inalámbrica	Electrónica para estudiantes
5	Transceptor CAN	TJA1051	1	0	0	Transceptor para bus CAN	Tec de Monterrey
6	Controlador CAN	MCP2515	1	0	0	Controlador independiente para bus CAN	Tec de Monterrey
7	Arduino	Arduino Uno	1	450.0	450.0	Placa de microcontrolador	Latch Electronics
8	Buzzer	Buzzer Genérico	1	12.0	12.0	Pequeño zumbador piezoelectrónico	Latch Electronics
9	ESC	ESC-10A	1	103.0	103.0	Controlador electrónico de velocidad	Amazon
10	Servomotor de 360°	DS04-NFC	1	0.0	0.0	Servomotor de rotación continua de 360 grados	Tec de Monterrey
11	Batería LiPo	LiPo-7.4V-2150m Ah	1	0	0	Batería recargable LiPo	Tec de Monterrey
12	Interruptor Simple	SPST-Switch	1	27.0	27.0	Interruptor de un solo polo, una sola posición	Electrónica para Estudiantes
13	Cortes de MDF	MDF-CUSTOM	5	70.0	70.0	Cortes de MDF personalizados para montaje y carcasa	Tec de Monterrey
14	Kit de Robot de Dirección con Servo	Servo Steering Robot	1	0.0	0.0	Kit del tractor	Tec de Monterrey

15	Sensor IMU	MPU-6050	1	0.0	0.0	Sensor IMU para detección de movimiento	Tec de Monterrey
16	Cables Jumper	JUMPERS	Surtidor	90.0	90.0	Cables jumper surtidos para conexiones	Latch Electronics
17	Cámara GPS	Cámara GPS	1	0	0	Cámara que detecta la posición del tractor	Tec de Monterrey

Costo Total: \$2045.00 MXN

Implementación (codificación) y pruebas

1. Sensor MPU6050 (IMU)

Conexiones:

- SDA -> PF15
- SCL -> PF14
- VCC -> 5V
- GND -> GND

Configuración de Inicialización:

```
C/C++
void SetupMPU(void) {
    mpu6050_init(&hi2c4, AD0_LOW, AFSR_2G, GFSR_250DPS, 0.98, 0.004);
    // Calibración opcional
    // MPU_calibrateAccel(&hi2c4, 1000);
    // MPU_calibrateGyro(&hi2c4, 1000);
}
```

2. Módulo NRF24L01 (Comunicación Inalámbrica)

Conexiones:

- CE -> PC6
- CSN -> PC7
- SCK -> PF7 (SPI5)
- MOSI -> PF9 (SPI5)
- MISO -> PF8 (SPI5)
- VCC -> 3.3V
- GND -> GND

C/C++

```
void SetupNRF24(void) {
    NRF24_begin(GPIOC, Nrf_CSn_Pin, Nrf_CE_Pin, hspi5);
    NRF24_setChannel(60);
    NRF24_setDataRate(RF24_2MBPS);
    NRF24_startListening();
    printf("NRF24L01 Configurado.\n");
}
```

3. Actuadores (Motor DC y Servo)

3.1. Servo Motor

Conexiones:

- Control -> PA6 (TIM13)
- VCC -> 5V
- GND -> GND

C/C++

```
void Turning_SetAngle(uint16_t angle) {
    uint32_t pulseWidth = 1000 + (angle * 1000) / 180;
    __HAL_TIM_SET_COMPARE(&htim13, TIM_CHANNEL_1, pulseWidth);
}
```

3.2. Motor DC

Conexiones:

- Control -> PA7 (TIM14)
- Alimentación: 12V a través del ESC.

C/C++

```
void SetMotorSpeed(uint16_t pulseWidth) {
    __HAL_TIM_SET_COMPARE(&htim14, TIM_CHANNEL_1, pulseWidth);
}
```

4. Red CAN

Conexiones:

- CRX -> PD0 (FDCAN1_RX)
- CTX -> PD1 (FDCAN1_TX)
- CANH -> Línea alta del bus CAN
- CANL -> Línea baja del bus CAN

C/C++

```
void MX_FDCAN1_Init(void) {
    hfdcan1.Init.FrameFormat = FDCAN_FRAME_CLASSIC;
    hfdcan1.Init.Mode = FDCAN_MODE_NORMAL;
    if (HAL_FDCAN_Init(&hfdcan1) != HAL_OK) {
        Error_Handler();
    }
}
```

1. SetupMPU(void)

Propósito: Configura el sensor IMU (MPU6050) para recolectar datos de aceleración y giroscopio.

Explicación:

- Inicializa el MPU6050 con parámetros predeterminados para el rango de aceleración y giro (2G y 250DPS).

- Opcionalmente, incluye calibraciones del acelerómetro y giroscopio para eliminar sesgos en las lecturas.

2. SetupNRF24(void)

Propósito: Configura el módulo NRF24L01 para comunicación inalámbrica.

Explicación:

- Inicializa el módulo con el SPI y los pines GPIO configurados.
- Ajusta parámetros como el canal, la velocidad de transmisión (2Mbps) y el tamaño de los paquetes (32 bytes).
- Activa el modo de lectura y habilita payload dinámico y ACK (Acknowledgment) automático.

3. ProcessNRF24Data(void)

Propósito: Procesa los datos recibidos por el módulo NRF24L01 y los interpreta como coordenadas y ángulo.

Explicación:

- Verifica si hay datos disponibles en el módulo NRF24L01.
- Extrae las coordenadas X, Y y el ángulo del paquete recibido.
- Convierte los datos crudos en valores legibles.
- Envía un payload de confirmación y los datos procesados a través de UART.

4. ProcessCANMessage(void)

Propósito: Procesa los mensajes CAN recibidos por el microcontrolador.

Explicación:

- Verifica si hay un mensaje disponible en el FIFO del bus CAN.
- Extrae datos del mensaje y los interpreta (por ejemplo, el estado del buzzer y las vueltas del encoder).
- Actualiza las variables globales correspondientes y genera retroalimentación visual (LED).

5. SendCANMessage(uint8_t state)

Propósito: Envía un mensaje a través del bus CAN.

Explicación:

- Configura el mensaje con el identificador predeterminado.
- Establece los datos del mensaje en el buffer de transmisión.
- Transmite el mensaje al bus CAN para interactuar con otros nodos (por ejemplo, encender o apagar el buzzer).

6. SetupPWM(void)

Propósito: Configura los periféricos PWM (TIM13 y TIM14) para controlar el servo y el motor DC.

Explicación:

- Inicia los temporizadores PWM y los configura con un ciclo de trabajo inicial (1500 µs) correspondiente a una posición o velocidad neutral.
- Permite el control del motor y el servo a través de señales PWM.

7. SetMotorSpeed(uint16_t pulseWidth)

Propósito: Controla la velocidad del motor DC ajustando el ciclo de trabajo del PWM.

Explicación:

- Modifica el ancho del pulso PWM en el canal correspondiente (TIM14) para aumentar o reducir la velocidad del motor.

8. BrakeMotor(void)

Propósito: Frena el motor DC estableciendo su velocidad en el valor neutral.

Explicación:

- Ajusta el ancho del pulso PWM a un valor predefinido (1500 µs), lo que detiene el motor.

9. Turning_SetAngle(uint16_t angle)

Propósito: Controla la dirección del vehículo ajustando el ángulo del servo.

Explicación:

- Calcula el ancho del pulso PWM necesario para el ángulo deseado.
- Establece el ancho del pulso PWM en el canal del servo (TIM13) para ajustar su posición.

10. RutaHardCoding(void)

Propósito: Ejecución de una rutina de prueba predefinida con segmentos de ruta.

Explicación:

- Controla el motor y el servo para recorrer una ruta predefinida utilizando tiempos de espera (delays).
- Incluye detenciones intermedias para procesar mensajes CAN.
- Es útil para pruebas iniciales y validación de movimiento.

11. NRFMotorCAN(void)

Propósito: Controla el motor y el servo utilizando datos recibidos por el NRF24L01 y el bus CAN.

Explicación:

- Calcula el ángulo objetivo en función de las coordenadas actuales y el punto de destino.
- Ajusta el motor y el servo para dirigir el vehículo hacia el punto de destino.
- Envía y procesa mensajes CAN para coordinar con otros nodos.

12. NRFMotorEncoderCAN(void)

Propósito: Implementa una rutina de navegación que utiliza el encoder, NRF24L01 y CAN para alcanzar múltiples segmentos de una ruta.

Explicación:

- Divide la ruta en segmentos definidos por coordenadas o posiciones del encoder.
- Ajusta la dirección y la velocidad del vehículo en función de los datos recibidos.
- Envía mensajes CAN para indicar eventos importantes, como alcanzar un "waypoint".

13. MotorEncoderCAN(void)

Propósito: Controla el motor y el servo utilizando exclusivamente datos del encoder, el NRF y el bus de CAN para la comunicación entre el Arduino que recibe el encoder y la STM que procesa la señal para determinar la dirección del servo y activar el motor.

Explicación:

- Divide la ruta en segmentos definidos por las vueltas del motor.
- Ajusta la dirección función del segmento actual, teniendo en cuenta el waypoint y la ubicación del tractor.
- Usa mensajes CAN para coordinar eventos entre nodos, Encoder-Arduino y STM-ESC.
- Activa servo y motor.

C/C++

```
//RUTA FINAL
void NRFMotorEncoderCAN(){
    uint8_t segmento = 0;
    uint8_t state = 0;
    float posiciones[] = {
        1,    // Primer avance
        5,    // Primera vuelta
        8.5,  // Segunda vuelta
        11   // Segundo avance
    };

    uint8_t targetsX[] = {230,155,125,125};
    uint8_t targetsY[] = {75,25,75,120};

    while (segmento < 5){
        ProcessNRF24Data();
        ProcessCANMessage();
        if((abs(coordX - targetsX[segmento]) <= 10 && abs(coordY - targetsY[segmento]) <= 10) || vueltas >= posiciones[segmento]){
            BrakeMotor();
            if (state == 0){
                state = 1;
                SendCANMessage(state);
            }
            while(buzzer == 0){
                ProcessCANMessage();
            }
            HAL_Delay(1000);
            ProcessNRF24Data();
            HAL_Delay(1000);
            segmento++;
            if (state == 1){
```

```

        state = 0;
        SendCANMessage(state);
    }
}else{
    if (state == 1){
        state = 0;
        SendCANMessage(state);
    }
    double targetAngle = atan2(targetsY[segmento] - coordY,
targetsX[segmento] - coordX);
    targetAngle = round(targetAngle * 180.0 / M_PI); // Convertimos radianes a grados

    double angleError = targetAngle - (double)(angle);

    targetAngle = 90 - angleError;
    if(targetAngle > 160) targetAngle = 160;
    if(targetAngle < 20) targetAngle = 20;

    Turning_SetAngle(targetAngle);
    if (targetAngle <= 60 || targetAngle >= 120 ){
        SetMotorSpeed(1100);
    }else{
        SetMotorSpeed(1300);
    }
}
}

```

Documentación de Pruebas

Prueba de Integración:

Se conectaron todos los subsistemas (IMU, NRF24L01, Actuadores y Red CAN) al STM32 para ejecutar la rutina NRFMotorEncoderCAN.

Resultados Esperados:

- Comunicación estable entre el Arduino y el STM32 a través del protocolo CAN.
 - El motor DC responde adecuadamente a los valores de PWM generados por las rutinas de control.
 - El servo ajusta el ángulo en función de los datos recibidos por el NRF24L01.

Resultados Obtenidos:

- Los datos transmitidos por NRF24L01 permitieron la navegación precisa entre los "waypoints".
- La activación del buzzer/LED ocurrió al alcanzar cada "waypoint" con éxito.
- Las posiciones medidas por el encoder coincidieron con los valores esperados.

Conclusión: El sistema integrado cumple con los requisitos de funcionalidad, proporcionando un control preciso y estable en las pruebas realizadas.

Unit Test: Sensor MPU6050

Team	Equipo 5
Responsible	Alfonso Solis
Description	Test the MPU6050 sensor for accurate acceleration and gyroscopic readings.
Function	Read sensor data and calculate orientation angles.
Components	MPU6050, STM32 I2C driver.
Hardware Used	STM32H745, MPU6050, logic analyzer.
Procedure	
Initial Conditions	MPU6050 configured with I2C, calibration complete.
Step 1	Request acceleration and gyro data.
Step 2	Verify received data with expected ranges.

Step 3	Calculate angles using sensor data and validate.
Expected Behavior with Acceptance Criteria	Sensor provides stable data within defined limits.
Actual Behavior	Sensor data was stable and accurate.
Result	OK.

Unit Test: Actuator - Servo Motor

Team	Equipo 5
Responsible	Alfonso Solis
Description	Verify servo motor response to PWM signals for angular positioning.
Function	Adjust the servo angle using TIM13 PWM output.
Components	Servo motor, STM32 TIM13.
Hardware Used	STM32H745, servo motor, oscilloscope.
Procedure	
Initial Conditions	TIM13 configured for PWM with 1-2 ms pulse width.

Step 1	Set servo angle to 0° and verify position.
Step 2	Set servo angle to 90° and verify position.
Step 3	Set servo angle to 180° and verify position.
Expected Behavior with Acceptance Criteria	Servo moves to the exact angles with no jitter.
Actual Behavior	Servo responded accurately to PWM signals.
Result	OK.

Integration Test: Communication - NRF24L01

Team	Equipo 5
Responsible	Alfonso Solis
Description	Test communication between two NRF24L01 modules.
Function	Send and receive data packets via SPI.
Components	NRF24L01, STM32 SPI driver.
Hardware Used	STM32H745, NRF24L01 modules.

Procedure	
Initial Conditions	NRF24L01 initialized in RX and TX modes.
Step 1	Transmit test data packet from module 1.
Step 2	Verify reception on module 2.
Step 3	Transmit acknowledgment packet from module 2.
Expected Behavior with Acceptance Criteria	Data packets transmitted and acknowledged without errors.
Actual Behavior	Data packets transmitted and acknowledged successfully.
Result	OK.

Integration Test: Communication - CAN Network

Team	Equipo 5
Responsible	Alfonso Solis
Description	Validate CAN communication between STM32 and Arduino node.
Function	Exchange encoded messages over the CAN bus.

Components	STM32 CAN driver, MCP2515 CAN module, Arduino.
Hardware Used	STM32H745, MCP2515, CAN transceivers, Arduino.
Procedure	
Initial Conditions	CAN filters and IDs configured.
Step 1	Transmit test frame from STM32.
Step 2	Verify data reception on Arduino.
Step 3	Send acknowledgment frame from Arduino.
Expected Behavior with Acceptance Criteria	Messages transmitted and received correctly within 1 ms latency.
Actual Behavior	Communication was reliable and within acceptable latency.
Result	OK.

Unit Test: Actuator - ESC for DC Motor

Team	Equipo 5
Responsible	Alfonso Solis

Description	Test ESC functionality for motor speed control using PWM.
Function	Generate PWM signals to control motor speed.
Components	ESC, STM32 TIM14.
Hardware Used	STM32H745, ESC, DC motor, multimeter.
Procedure	
Initial Conditions	TIM14 configured for PWM.
Step 1	Set PWM to 50% and verify motor speed.
Step 2	Increase PWM to 75% and observe speed change.
Step 3	Decrease PWM to 25% and confirm motor slows down.
Expected Behavior with Acceptance Criteria	Motor speed changes proportionally to PWM duty cycle.
Actual Behavior	Motor speed adjusted correctly with PWM signals.
Result	OK.

System Test: Overall System Navigation

Team	Equipo 5
Responsible	Alfonso Solis
Description	Test the full system's ability to navigate between waypoints using sensors, actuators, and communication modules.
Function	Integrate sensor readings, communication, and motor control for autonomous navigation.
Components	STM32, MPU6050, NRF24L01, CAN, DC motor(with encoder), Servo.
Hardware Used	STM32H745, NRF24L01, MPU6050, motor, servo.
Procedure	
Initial Conditions	All modules initialized and ready.
Step 1	Define waypoints and start the navigation routine.
Step 2	Use Encoder and NRF24L01 to monitor position and orientation.
Step 3	Adjust motor and servo based on feedback.
Step 4	Verify successful waypoint navigation and log errors.

Expected Behavior with Acceptance Criteria	Robot reaches all waypoints with less than 5% positional error.
Actual Behavior	Navigation successful with minor adjustments required.
Result	OK.

Comportamiento inteligente

El tractor toma 2 parámetros importantes a la hora de detenerse en los waypoints, el primero y principal son las coordenadas por el NRF24, hace la resta de las coordenadas recibidas menos las objetivo y si el absoluto en ambas da menor a un rango de 7 cm, se detiene, se le puso este rango para amortiguar el tiempo lento tiempo de envío de datos de la cámara. En caso de no recibir el valor a tiempo, usa la cantidad de vueltas detectadas por el encoder para tomar la decisión de si detenerse o seguir avanzando. Igualmente cada que recibe su posición, ajusta mediante la arcotangente de sus coordenadas menos la posición objetivo y su ángulo actual el error en el ángulo objetivo, además, si detecta que la curva a tomar es cerrada, aumenta la velocidad de movimiento para no quedarse atorado mientras gira

Video de demostración:

<https://drive.google.com/file/d/1VipUzo-c-BnRVilcmz3iSH7LOjSFaA0G/view?usp=sharing>

Conclusiones

En el desarrollo del reto se alcanzaron importantes logros, destacando la integración eficiente de sistemas embebidos avanzados como la IMU, el protocolo CAN, el módulo NRF24L01 y el control de motores mediante PWM. Además, se logró implementar un sistema robusto de navegación autónoma que cumplió con los objetivos planteados, como la correcta interpretación de datos en tiempo real, la sincronización de periféricos y el seguimiento preciso de trayectorias predefinidas.

Reflexiones y agradecimiento

Durante este proyecto, el equipo desarrolló competencias clave en ingeniería de sistemas embebidos, como la programación eficiente de microcontroladores, el manejo avanzado de protocolos de comunicación (SPI, CAN e I2C) y la validación de sistemas mediante pruebas unitarias e integradas. Este reto permitió también fortalecer habilidades de trabajo en equipo, resolución de problemas y comunicación técnica.

Queremos expresar nuestro más sincero agradecimiento a John Deere, por brindarnos la oportunidad de aplicar y expandir nuestros conocimientos en un entorno realista, fomentando nuestra preparación para desafíos futuros en la ingeniería de precisión y tecnología agrícola.

Participación

Integrante	Sección Asignada
Alfonso	Implementación de la codificación básica de cada módulo y configuración del clock en STM32CubeIDE.
René	Construcción e implementación del armado del carro en conjunto.
Juan	Implementación del código para la RutaHardCoding y Waypoints.
Felipe	Creación de la carcasa y base para cada componente para el tractor en conjunto.