Design of Advanced Embedded Systems (Gpo 602)

FINAL CHALLENGE REPORT:

WAYPOINT ESTIMATION

Author:

Ximena Lizeth Trejo Lavín A01198557
Dafne Naomy Reyes Pimentel A00834003
Jesús Javier Martínez Hernández A00833296
Valeria Aranza Cerda Ochoa A01236733

Professors:

Dr. Alfonso Avila Ortega

Dr. Raúl Peña

Monterrey, Nuevo León, México. November 30, 2024

Abstract

The rapid advancement of automation and technology in agriculture highlights the need for innovative solutions to enhance productivity and efficiency. This project aims to design and implement an embedded system for a small-scale autonomous tractor capable of estimating and following the most efficient route through predefined waypoints. The main controller is the STM32H755 development board, while incorporating communication protocols such as CAN, SPI, and I2C. The system integrates advanced embedded design principles to achieve seamless navigation. This prototype demonstrates the potential for applications in modern agriculture, addressing key challenges such as labor reduction, error minimization, and productivity enhancement, while advancing the evolution of intelligent agricultural machinery.

Contents

1. Product Description

1.1 Challenge Description

1.1.1 Introduction

John Deere is a globally recognized enterprise, renowned as a leader in agricultural and forestry machinery. The company has established a strong reputation for its commitment to innovation and the development of intelligent solutions that enhance its products and maintain its position as one of the most influential enterprises in the world.

In this project, the task is to design a small-scale tractor capable of autonomously reaching and following a set of four waypoints. The system will leverage multiple communication protocols and utilize a high-performance development board, the STM32H755, known for its advanced capabilities.

1.1.2 Objective

The primary objective of this project is to design an embedded system for a scaled-down tractor that can autonomously estimate and follow the most efficient route to reach predefined waypoints. The project emphasizes the integration of advanced embedded system design, utilizing various communication protocols such as CAN, SPI, and I2C. This prototype aims to present an innovative and efficient solution, potentially scalable for real-world farming applications.

1.1.3 Justification

The agricultural sector is increasingly adopting automation and advanced technologies to enhance productivity and efficiency. The development of an autonomous navigation system capable of following waypoints addresses critical needs in modern farming, including the reduction of human errors, decreased labor demands, and improved operational productivity. This project not only aligns with these technological advancements but also offers a practical and scalable solution for the challenges faced in contemporary agriculture.

1.2 Solution

1.2.1 Product Description

The proposed product is a scaled-down tractor capable of autonomously estimating and following a set of waypoints provided by the user. This system employs the John Deere Localization Positioning System camera for precise waypoint identification and navigation. A

dual-core microcontroller, known for its robustness and efficiency, serves as the central processing unit for the prototype.

The design incorporates several advanced components, including:

Communication Modules: Integration of SPI through an NRF24 module, CAN transceiver, and additional protocols to ensure seamless data transfer.

Actuators: A 12V DC motor to drive the tractor's movement.

Real-Time Operating System (RTOS): Task prioritization and management are achieved through the RTOS, enabling the tractor to handle multiple functions simultaneously and effectively. Tasks are assigned priorities based on their criticality, ensuring optimal performance.

This comprehensive design results in a smart, autonomous tractor that provides a viable solution for the agricultural sector.

1.2.2 Vision

Our vision is to pioneer innovative solutions that bridge the gap between technological advancements and their practical applications across diverse industries. By delivering scalable and efficient solutions, we aim to address the unique challenges faced by specific groups, particularly in the agricultural sector.

1.2.3 Needs

The primary needs addressed by this project include:

- Autonomous Navigation: Developing a system that reduces human intervention in farming tasks, thus minimizing errors and increasing productivity.

- Efficient Resource Utilization: Ensuring optimal power consumption, effective use of communication protocols, and clear communication with the user.

- Error Reduction: Offering precise and reliable operations to mitigate costly mistakes in farming.

1.2.4 Target Group

The target audience for this product includes:

- Farmers and Agricultural Workers: Individuals seeking innovative tools to enhance farming efficiency.

- Agricultural Companies: Enterprises like John Deere that continually strive for groundbreaking solutions in the sector.

- Manufacturers: Organizations focused on integrating cutting-edge technologies into their products to meet industry demands.

1.2.5 How the Product Satisfies Target Group Needs

This prototype meets the needs of the target group in several ways:

- Integration of Advanced Communication Technologies: Ensures seamless and efficient operation through technologies such as SPI, CAN, and RTOS.

- Innovation for the Farming Sector: Provides a groundbreaking solution that minimizes operational errors, reducing potential costs in the long term.

- Autonomous Navigation: Delivers a system capable of navigating independently, significantly enhancing productivity and reducing labor demands.

- Scalability: Demonstrates potential for adaptation and expansion to meet future industry challenges.

2. Development and Teamwork Methodology

2.1 Tasks/Actions List (V-Model)

The following list of tasks has been followed since the start of the project to ensure the successful achievement of the challenge's main objective. It was essential to define and refine these tasks as the project progressed to maintain effective control and ensure a balanced distribution of work among team members.

- ☐ Familiarizing and practicing with the STM32H755, including understanding its configurations and operation.
- ☐ Learning about UART configuration, protocol, testing, and analyzing its functionality using tools such as an oscilloscope.
- ☐ Exploring wireless communication, including its configuration, practice, and application analysis for the challenge.
- ☐ Conducting SPI testing and configuration, and studying its relationship with the challenge using the NRF24 module.
- ☐ Designing, assembling, and analyzing the chassis of the car, along with basic components such as the motors for the front and rear wheels.
- ☐ Learning to control a servo motor using the STM32 configurations and PWM.
- ☐ Learning to control the rear wheel DC motor through PWM configurations, analyzing voltage and current usage, and managing its ROM.
- ☐ Studying the CAN protocol, applying it to the challenge, and practicing with the Toyota cluster, including learning its components such as the transceiver and MCP controller.
- ☐ Integrating and learning about the IMU with I2C communication.
- ☐ Completing the integration of all learned systems and protocols, including programming and testing using the laboratory camera for project finalization.
- ☐ Conducting tests and tracking waypoints.
- ☐ Preparing the final report and presentation.

2.2 List of tasks

Task	Due date	Date delivered	Duration
ExperimentWirelessInterfacing	28/Ago/2024	21/Nov/2024	2 Months 24 Days
Experiment Tractor Direction	05/Sept/2024	10/Sept/2024	5 Days
Challenge Report	08/Sept/2024	10/Sept/2024	3 Days
I2C RTC connection	18/Sept/2024	18/Sept/2024	1 Day
CanBus Connection	30/Sept/2024	1/Nov/2024	1 Month
Interfacing to ESC	02/Oct/2024	22/Nov/2024	1 Month 20 Days
Two MCUS over CAN BUS	07/Oct/2024	21/Nov/2024	1 Month 14 Days
Interfacing to IMU and First Integration	17/Oct/2024	27/Nov/2024	1 Month 10 Days
Discrete Signals using STM32	07/Nov/2024	07/Nov/2024	1 Day
STAGE 1	12/Nov/2024	12/Nov/2024	31 Days
Implement DFT using STM32	19/Nov/2024	19/Nov/2024	9 Days
STAGE 2	19/Nov/2024	19/Nov/2024	10 Days

Table 1. Tasks and their duration

2.3 Gantt Diagram of the work distributed

We decided to buy different STM32H755 modules because at the first moment we saw that we probably will need more than one for having more optimization in the due dates. So at the first moment we started working just with one but in the first week we received our personal STM and that helped us a lot to deliver the work quickly. So we as a team basically decided to split into two and in this way Valeria and Jesus worked in some and Ximena and Dafne worked in some others. The tasks were also distributed depending on availability of each member.

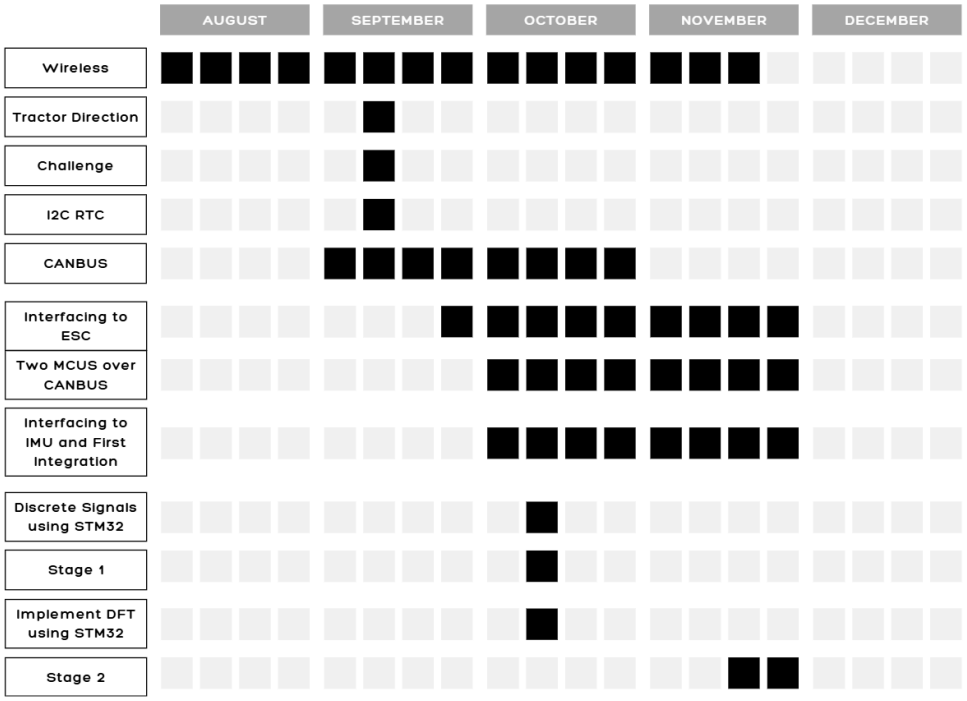


Figure 1. Gantt Chart illustrating the methodology implemented during the activities by section and dates emphasizing the amount of time employed

Teamwork Methodology Activities:

Wireless:	Jesus, Valeria
Tractor Direction:	Ximena
Report Challenge:	Ximena, Jesus, Valeria, Dafne
I2C RTC:	Ximena, Dafne
CAN Bus:	Ximena, Dafne
Interfacing ESC:	Jesus, Valeria
Two MCUs over CAN Bus:	Ximena, Dafne
Interfacing to IMU and first Integration:	Ximena, Dafne
Discrete Signals using STM32:	Valeria, Jesus
Stage 1:	Ximena
Implement DFT using STM32:	Valeria
Stage 2:	Ximena, Jesus, Valeria Dafne

Table 2. Activities and the teammates who did them

2.4 Reflection on Work Distribution

After completing all the activities we were reflecting on the fact that some activities took more time to accomplish than others, also some of them we finished after a lot of time but with all the requirements working. Most of the time spent was also because the modules did not work well since the beginning and we took a lot of time trying to find the problem thinking that it was something of the code implemented or a bad connection and then we just realized that the module given was burnt, and some other problems that with a deeper analysis we found each of them and fixed them correctly. We also suggested that it would be better about defining roles from the beginning and to have some figures in better quality using specialized tools for a better presentation.

2.5 Reflection on Proposed Improvements

We discuss about it and everyone was agreed that probably we should distribute the work better, because we were were having the last week some pending activities and also at the time we had to have the challenge ready for a short period of time and in the challenge we needed to also splitted to have the presentation and the tractor ready with the 100% of the functionality. *But probably we will be done better if we would have roles more defined, tools to use, and better communication.* In the challenge the team also faced some problems with the tractor functionality and that also stressed us more and made us have a final reflection on the work delivered to improve more on the next projects.

3. Deployment Diagrams

3.1 UML Diagram

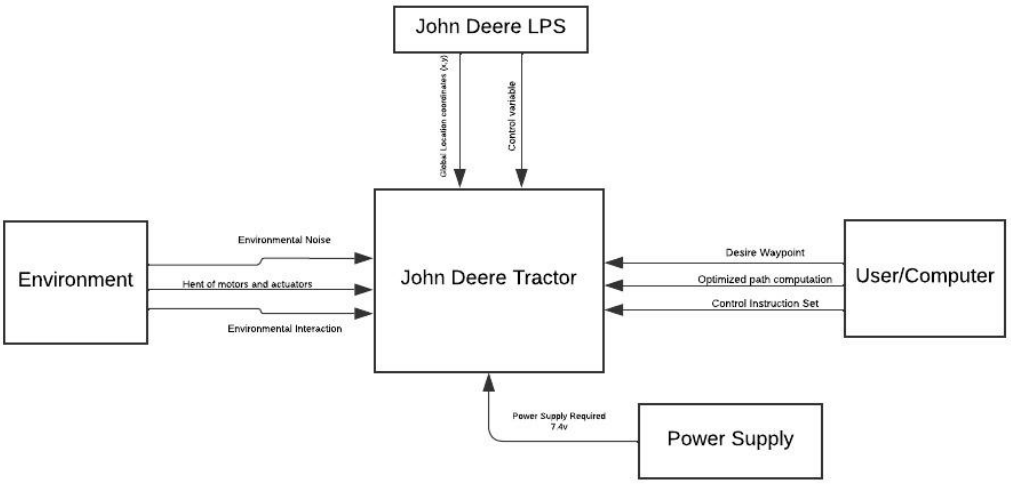


Figure 2. UML context diagram

The diagram represents the ecosystem of a John Deere tractor, highlighting its interactions with the user, the environment, the setup system, and other key elements. The user sends commands, receives position graphs, and controls system notifications, while the tractor monitors its interaction with the environment to follow the waypoint trajectory. The Deere Setup system provides feedback control and location data, while a computer optimizes routes and waypoints for navigation. Finally, the charger ensures the power supply to keep the tractor operational.

3.2 Requirements Table

Name	**John Deere NavTrak**
Purpose	Implement the Navigation System for Autonomous Direction
Inputs	Sensors/ Global Positioning Device
Outputs	Wireless Data of estimation obtained in position by the sensor
Functions	**1** Tractor collect and Store data **1.1** Reception in real time of the positioning device **1.2** Receive data from Encoder (CAN FORMAT) **1.3** Receive data (IMU UART/SPI/CAN FORMAT) **2** Use the sensors to estimate the position **2.1** The John Deere GPS estimates the position **2.2** The encoder analyzes the data from UART **3** The tractor will generate the specific route with the previous analysis of the waypoints **4** The tractor navigates through the selected waypoints **5** The tractor sends back the information of the route navigated collected from the sensors
Performance	Gets the real time position for every second performed.
Manufacturing cost	**Approximate cost:** $150 USD
Power	15-20 kW 6-12 V per motor work
Physical Size / Weight	**Length:** 248 mm **Width:** 146 mm **Height**: 70 mm **Weight**: 680 gr

Table 3. Requirement table including the process of manufacturing cost and sizes established.

Requirement table specific for each component:

Requirements	Task Description	Type
Sensors	Be in constant tracking of the position of the tractor and monitorization by IMU	Functional in tractor
Tractor navigation system	The autonomous system implemented needs to guide the direction of the tractor through the waypoints defined correctly.	Functional in tractor
Motors management	Regulation of the direction and speed of the DC motor and also the servo control when it's in action.	Functional in tractor
Wireless information send/receive	Have the correct communication with the wireless module. Tractor - NRF24	Functional in tractor
Efficiency	All the physical prototype including the software coding needs to be optimized for having better performance, efficiency and low power consumption.	Non Functional in tractor
Feedback	The feedback in all moments is being transmitted in real time by the protocol receiving the errors during the performance, task data, position in the actual moment and general status of the tractor while it is performing the task.	Non Functional in tractor
Tasks prioritization	The task prioritization was implemented with RTOS also to make the system optimized.	Non Functional in tractor
System checks	To execute the waypoints trajectory the tractor needs to perform all the system checks connected with all the components to be sure	Non Functional in Tractor

	everything is ready to perform.	
Scalability	This prototype is coded to perform the waypoint trajectory proposed but also is open to have modifications, extra tasks, components and to perform a different task in order to what the user wants.	Non Functional in Tractor

Table 4. Requirement table from each component functional and non functional of the tractor to realice the complete performance of the waypoints.

3.3 Deployment Diagram

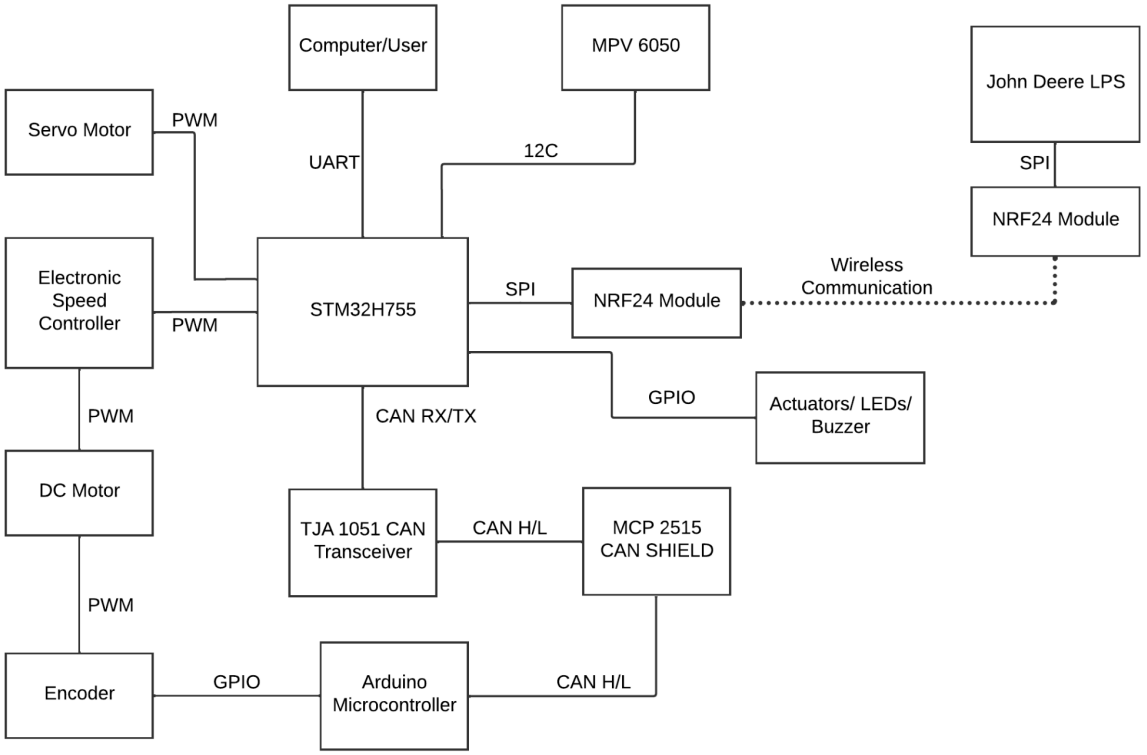


Figure 3. Deployment Diagram

The diagram illustrates our system, primarily based on the "brain," which is the STM32H755 microcontroller as the central core, connected to various peripherals for control, communication, and monitoring. The STM32 controls a servo motor and an Electronic Speed Controller (ESC) for a DC motor using PWM signals, while receiving data from an MPU6050 sensor via I2C to measure acceleration and gyroscope readings. Additionally, it is linked to an NRF24 module via SPI for wireless communication, which connects with other distributed NRF24 modules. The system uses a TJA1051 transceiver to manage CAN

communication, allowing connection to an Arduino module (equipped with an MCP2515 CAN Shield) that, in turn, interacts with an encoder to measure position or speed. On the other hand, a PC (referred to as the "John Deere System") connects to the STM32 via USB for programming or monitoring, with the option to exchange data through the CAN bus. This system integrates sensors, actuators, and distributed nodes through communication protocols such as PWM, SPI, I2C, and CAN.

3.4 UML Activity Diagram

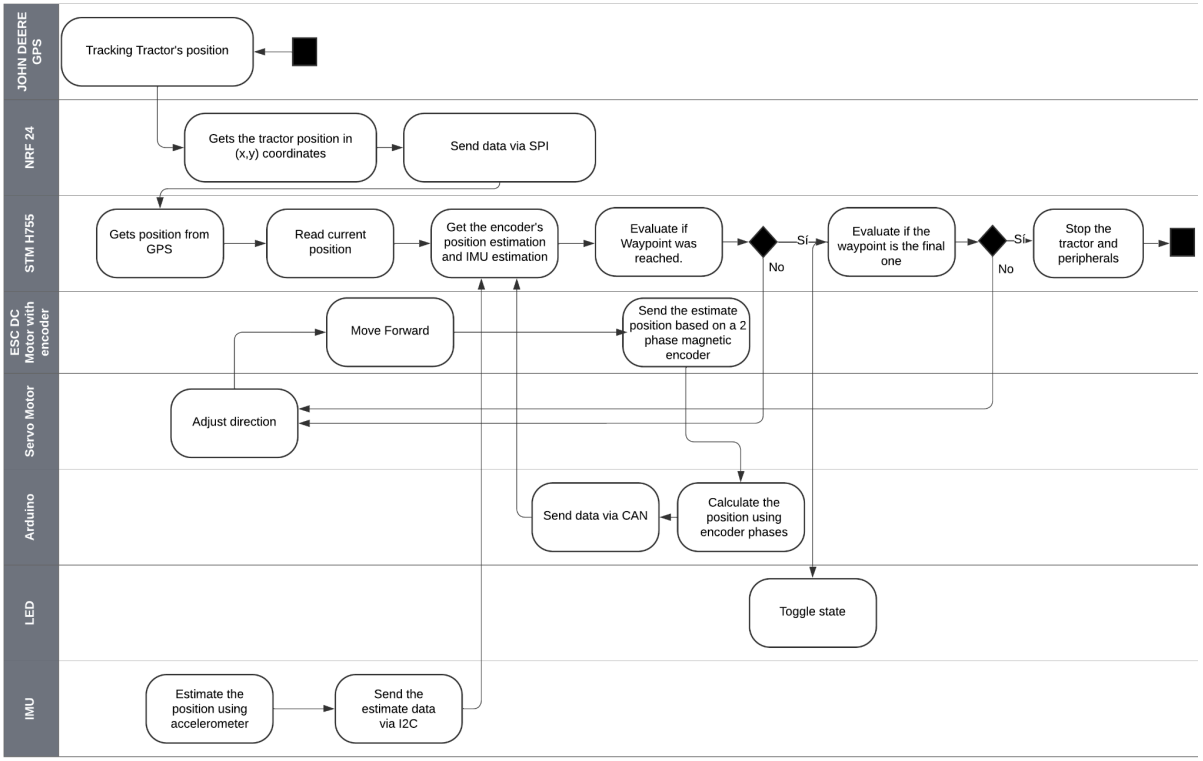


Figure 4. Activity Diagram

The activity diagram provides a detailed representation of the tractor system's workflow, highlighting the sequence of operations and interactions between its components. It showcases key transitions, such as position acquisition through GPS and the NRF24 module, position estimation via a two-phase magnetic encoder, and dynamic adjustments to the tractor's direction and speed using traction and steering motors.

The system incorporates waypoint evaluation, distinguishing between intermediate and final waypoints to ensure precise trajectory control. Communication between modules, enabled by SPI and CAN protocols, ensures synchronization, while automated stopping mechanisms with visual and auditory notifications (LEDs and buzzer) guarantee safe operation. This robust, modular design is well-suited for scalable agricultural applications.

3.5 Schematic

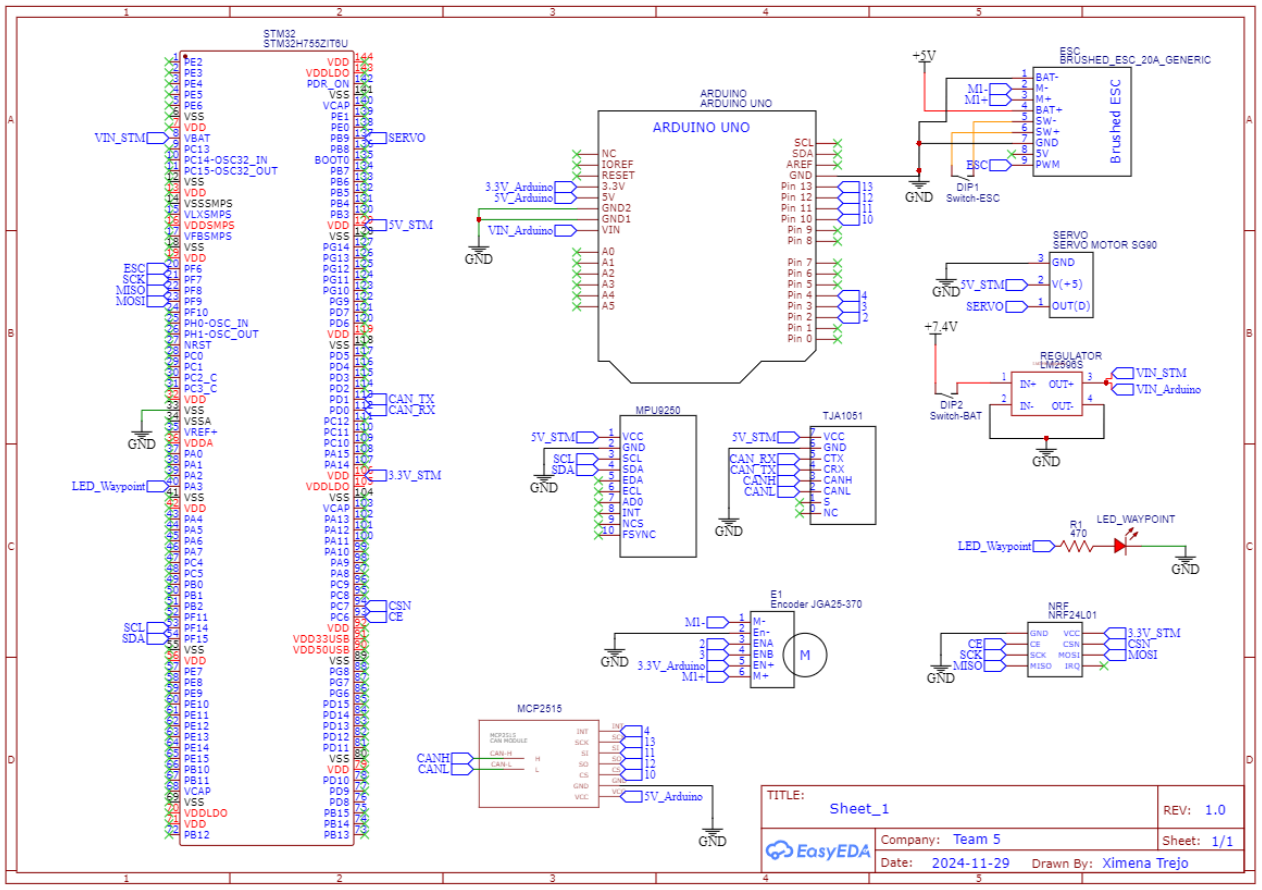


Figure 5. Final schematic for the project

4. Selected Peripheral Devices

4.1 Justification of Selected Components

4.1.1 Microcontrollers:

- ***STM32H755***

Microcontroller with two cores, which is the one that integrated all the subsystems. The NRF24L01, MPU9060, servomotor, ESC and TJA1051 were directly connected to it. Given that its specifications mentioned that it could be powered through the VIN pin, 5V were input into this pin to power part of the system. It was also necessary to have a common ground, so there would not be any sort of electrical issues.

- ***Arduino UNO***

The Arduino sent information from the encoder's two phases to the STM32 through CAN. It was powered to the regulator's output through its VIN pin.

4.1.2 Sensors, Actuators and General Hardware:

- ***LM2596***

It regulated +7.4V from the batteries to +5V, which was what is referred to in this document as the "regulator's output".

- ***NRF24L01***

This component is connected through SPI to the STM32H755, and wirelessly connects to the John Deere camera. In the code, it gets the coordinates and the angle of the arrow on the ground through the NRF's reception buffer. It had to be connected to +3.3V from the STM32.

- ***MPU9250***

Although it was not used in the final implementation, it gave information through I2C with outputs from its accelerometer and gyroscope, translating into coordinates and angles. It is powered with +3.3-5V from the regulator's output.

- ***TJA1051***

The TJA1051 controlled and detected data from and to the CAN bus that was implemented between the Arduino UNO and the STM32. It was powered to the regulator's output.

- ***20A Brushed Electronic Speed Controller***

The ESC helped control the DC motor through pulse widths. Depending on the number of pulse widths, the motor would go forward or backwards, and also slowlier or faster. The battery interface was connected to +7.4V with 2.8A.

- ***MCP2515***

It is a CAN bus controller that can communicate through SPI to microcontrollers, sending and receiving CAN messages. It was powered to the regulator's output.

- ***Voltmeter***

Although this was optional, we decided to add a voltmeter so that we would not have to directly check the battery's voltage to make sure that they had to be recharged.

4.1.3 Motors:

- ***MG996R Servomotor***

It helped change the steering direction of the tractor, since the back motor only gave forward motion. Most servomotors work starting +3.3V, but since this one has more torque, it had to be powered from the regulator's output.

- ***JGA25-370 12V 620 RPM DC Motor with Encoder***

The motor's interfaces were connected according to this model's pinout. The encoder was powered with +3.3V from the Arduino UNO, and both of its phases were attached to interrupt pins 2 and 3 from this same microcontroller.

4.2 Software

4.2.1 List and Justification:

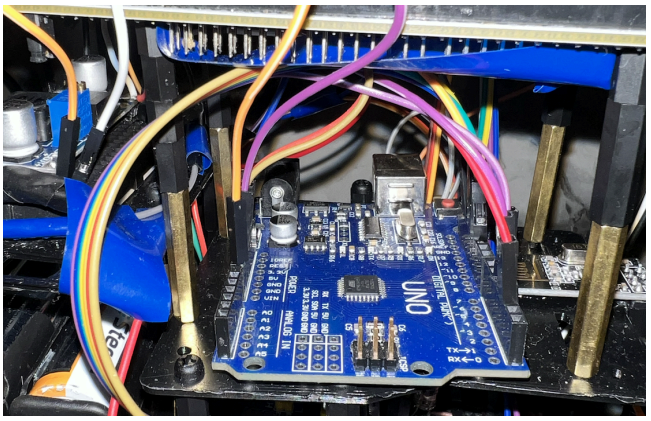
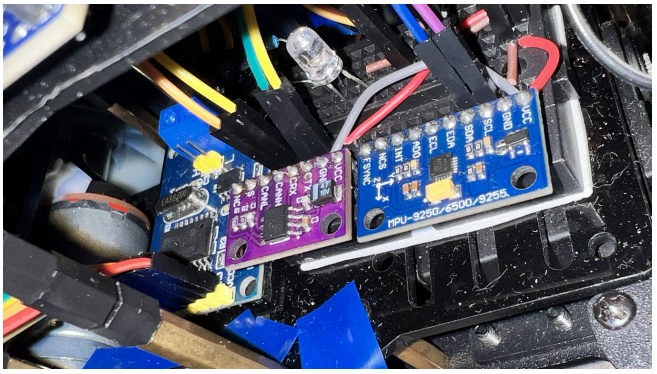
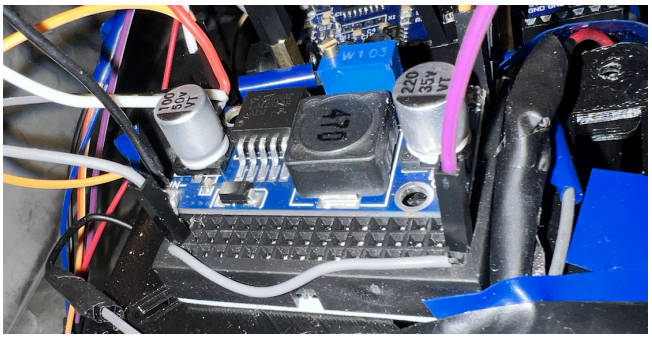
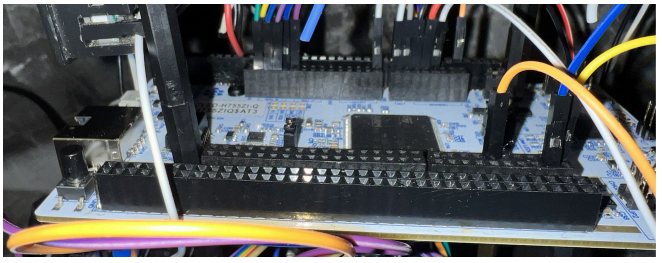
- ***STM32 CubeIDE:***
  Used for programming and integrating each protocol and system required for the project, such as CAN communication, Wireless integration, and systems fusion. It serves as the primary environment for developing and debugging firmware for the STM32 microcontroller.
- ***C Programming:***
  The programming language used within STM32 CubeIDE to write the code for configuring and implementing various protocols and functionalities on the STM32 microcontroller, ensuring precise and efficient system behavior.
- ***Arduino IDE:***
  Utilized to read encoder data from the motor and transmit it via the CAN protocol, providing a streamlined interface for managing and testing motor data acquisition and communication.
- ***SolidWorks:***
  Employed to design additional components added to the car, including structural elements and adjustments for each new floor or element integrated into the system, ensuring mechanical compatibility and robust construction.

These tools collectively address both software and hardware aspects of our project, ensuring seamless integration and operation.

4.3 Final Product Pictures

4.3.1 Each Component Picture

The following pictures give a clearer idea of the final product:

Arduino	
CAN & IMU	
Voltage Regulator	
STM32H755	

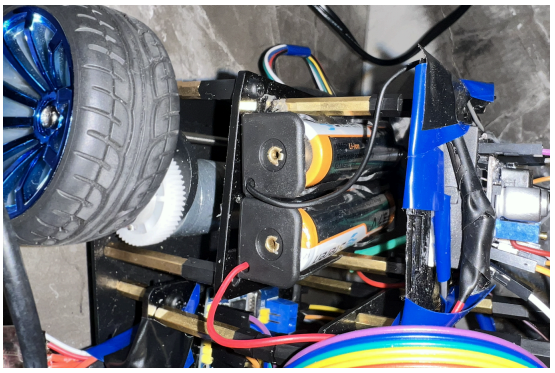
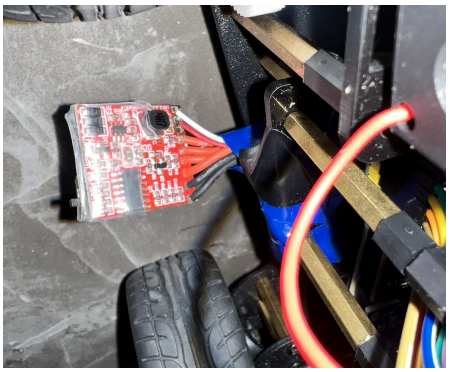
Batteries	
ESC	

Table 5. Components of the final tractor

4.3.2 Final Design:

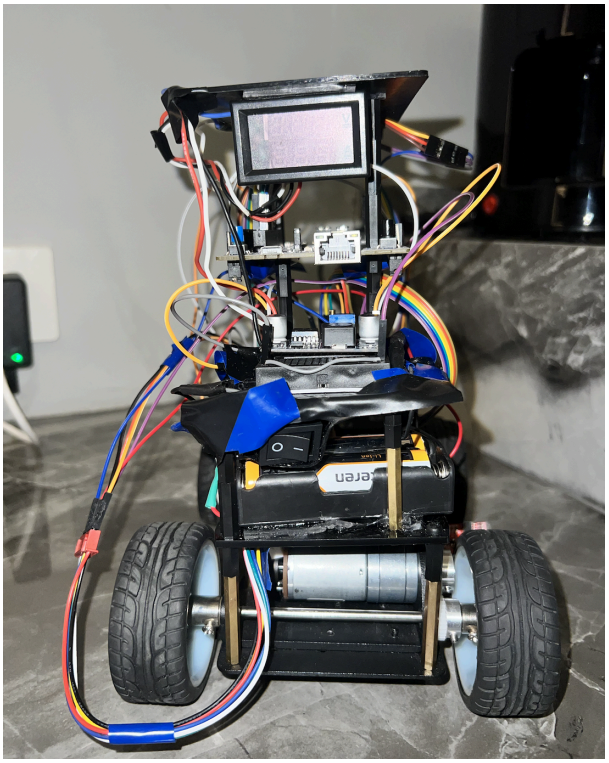


Figure 6. Final Prototype

5. System Implementation

5.1 Complete system

After implementing all this, the physical result was the following:

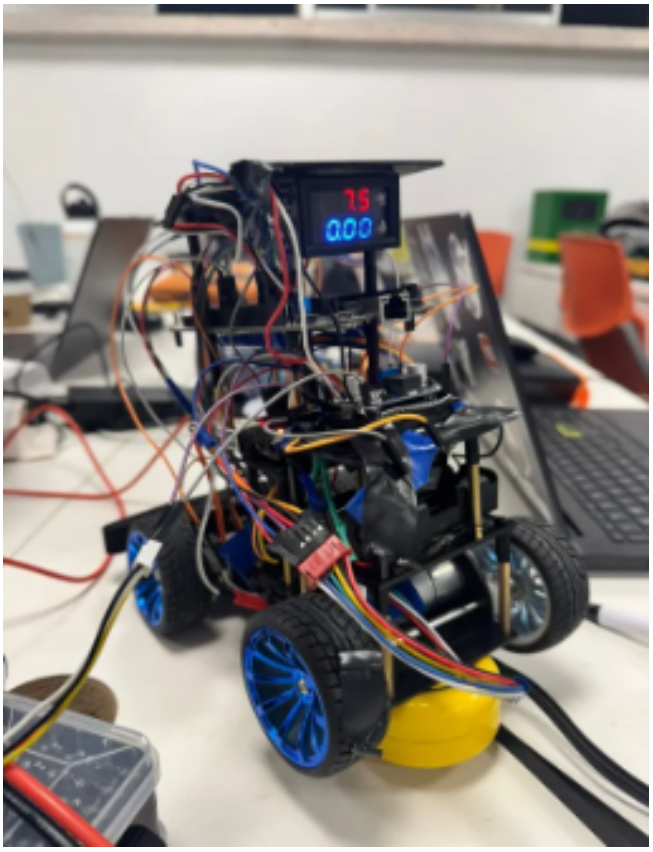


Figure 7. Complete system (physical)

It is important to note that the vehicle's height was deliberately designed to ensure sufficient clearance for the electrical circuits, minimizing the risk of interference or damage during potential reconnection or disconnection procedures.

The Bill of Materials (BOM) was prepared based on the items that needed to be purchased, rather than the components provided by the lab. After this clarification, the BOM is presented in the following table:

Component	**Quantity**	**Price (MXN)**
STM32H755	1	$ 1,400.00
LM2596	1	$ 54.00
NRF24L01	1	$ 43.00
MG996R Servo Motor	1	$ 80.00
JGA25-370 12V 620 ROM DC Motor with Encoder	1	$ 250.00
Total		**$ 1,827.00**

Table 6. Bill of Materials

5.2 Subsystems

PWM

The PWM for the servomotor was configured through TIM17 on the STM32 through PB9. The Prescaler was set to 239, with a counter period of 19999:

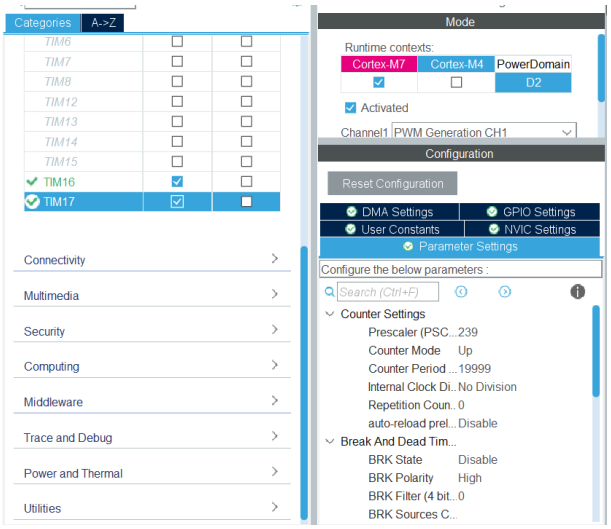


Figure 8. TIM 17 Parameter Settings

Pin Name	Signal on Pin	Pin Context...	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PB9	TIM17_CH1	ARM Corte...	n/a	Alternate F...	No pull-up a...	Low	Disable	PWM_Servo	☑

Figure 9. PB9 pin configuration

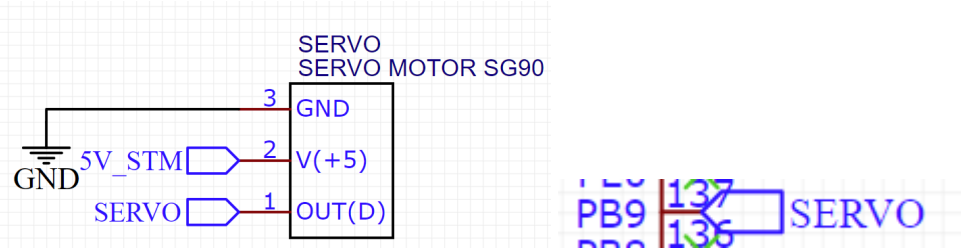


Figure 10. Location in Schematic

As for the ESC, which was assigned to TIM16, in pin PF6, the parameters were the same as for TIM17:

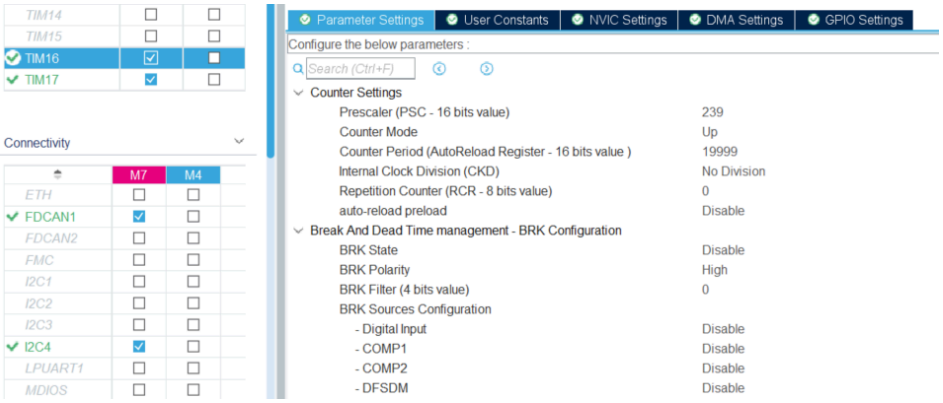


Figure 11. TIM 16 Parameter Settings

Pin Name	Signal on Pin	Pin Context ...	GPIO output l...	GPIO mode	GPIO Pull-up ...	Maximum out...	Fast Mode	User Label	Modified
PF6	TIM16_CH1	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a	PWM_ESC	☑

Figure 11. PF6 pin configuration

Figure 12. Location in Schematic

SPI

Pins PF7, PF8 and PF9 were used for SCK, MISO and MOSI respectively on the STM32. Some of the parameters were 8 bits for data size, with the MSB first, and a prescaler of 16:

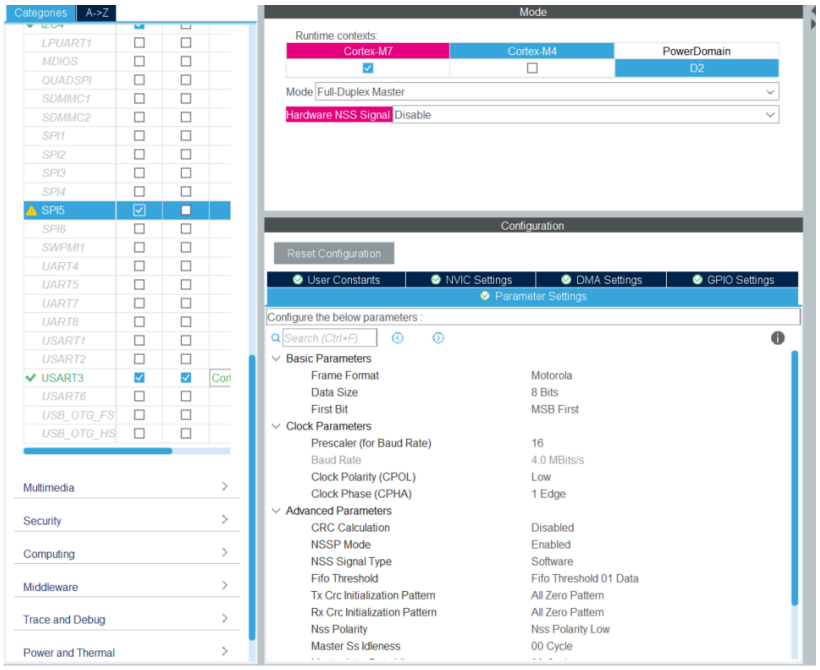


Figure 13. SPI5 Parameter Settings

Pin Name	Signal on Pin	Pin Context...	GPIO output l...	GPIO mode	GPIO Pull-up...	Maximum out...	Fast Mode	User Label	Modified
PF7	SPI5_SCK	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		☑
PF8	SPI5_MISO	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		☑
PF9	SPI5_MOSI	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		☑

Figure 14. SPI Pin configuration

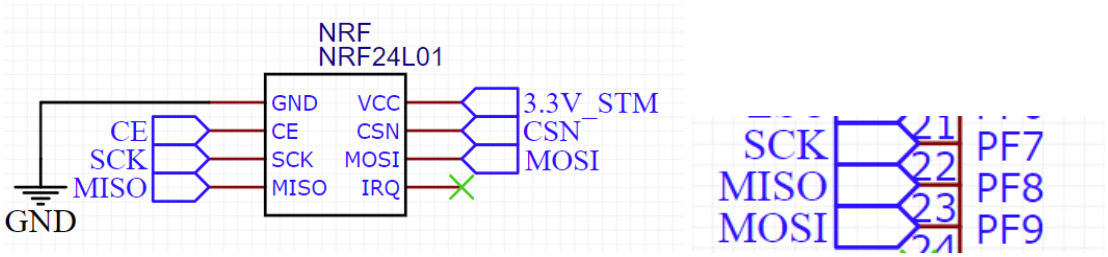


Figure 15. Location in Schematic

I2C

Pins PF14 and PF15 were used for SCL and SDA respectively on the STM32. The I2C was set to a frequency of 100kHz, with standard speed mode:

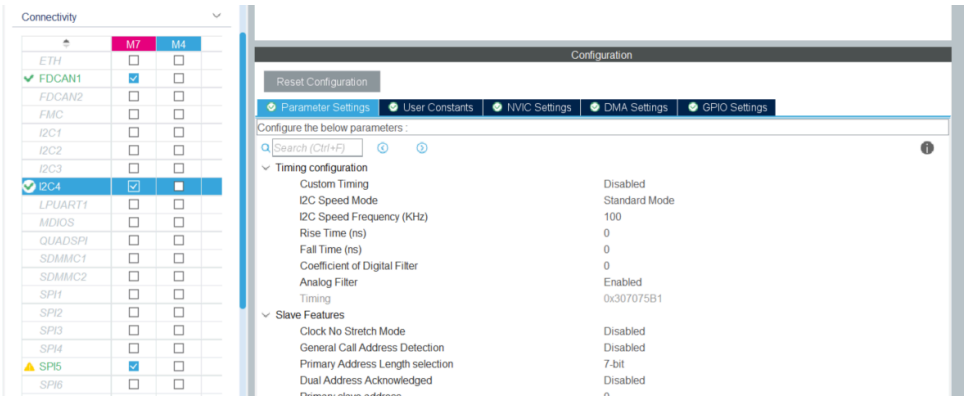


Figure 5.2.10 I2C Parameter Settings

Pin Name	Signal on Pin	Pin Context...	GPIO output l...	GPIO mode	GPIO Pull-up...	Maximum out...	Fast Mode	User Label	Modified
PF14	I2C4_SCL	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		☑
PF15	I2C4_SDA	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		☑

Figure 16. I2C Pin configuration

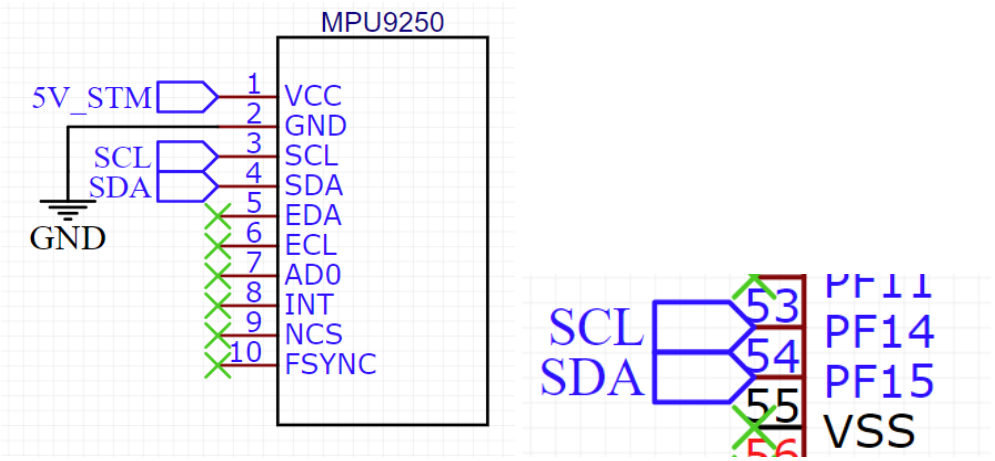


Figure 17. Location in Schematic

UART

This protocol was *only* used to print messages to the terminal. The parameters were a baud rate of 9600 Bits/s, with a word length of 8 bits, no parity and 1 stop bit:

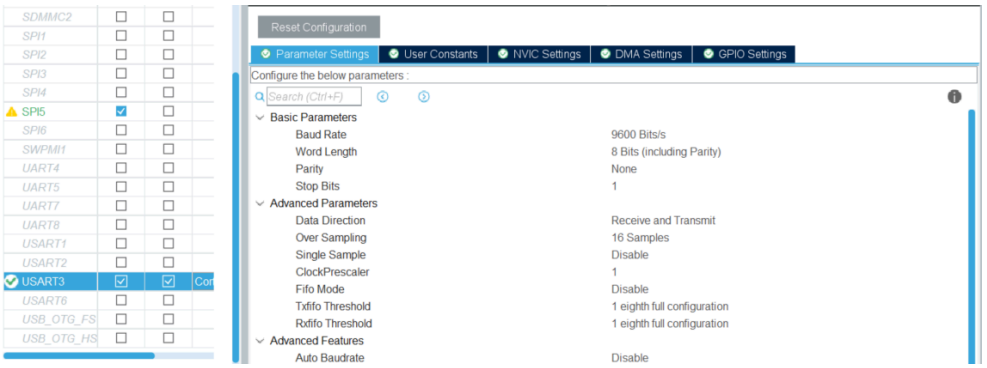


Figure 18. UART Parameter Settings

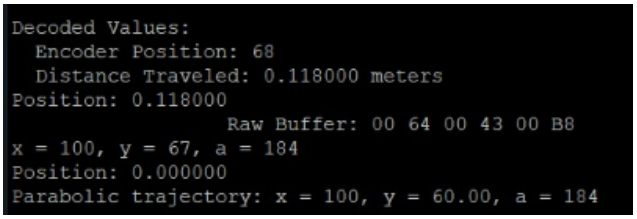


Figure 19. UART Message Received

CAN

Pins PD0 and PD1 were used for CANRX and CANTX respectively on the STM32. Some of the parameters were classic frame format, normal mode, nominal sync jump width of 8, etc.:

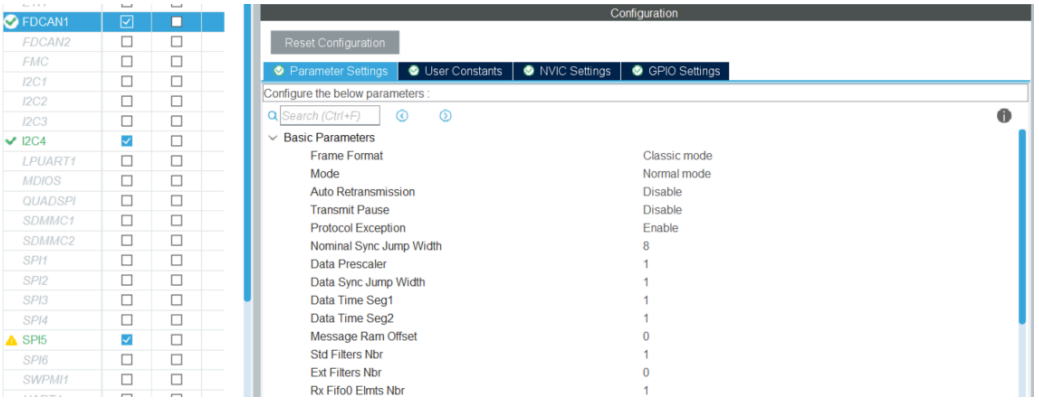


Figure 20. CAN Parameter Settings

Pin Name	Signal on Pin	Pin Context ...	GPIO output l...	GPIO mode	GPIO Pull-up...	Maximum out...	Fast Mode	User Label	Modified
PD0	FDCAN1_RX	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		☑
PD1	FDCAN1_TX	ARM Cortex...	n/a	Alternate Fun...	No pull-up an...	Low	n/a		☑

Figure 21. CAN Pin configuration

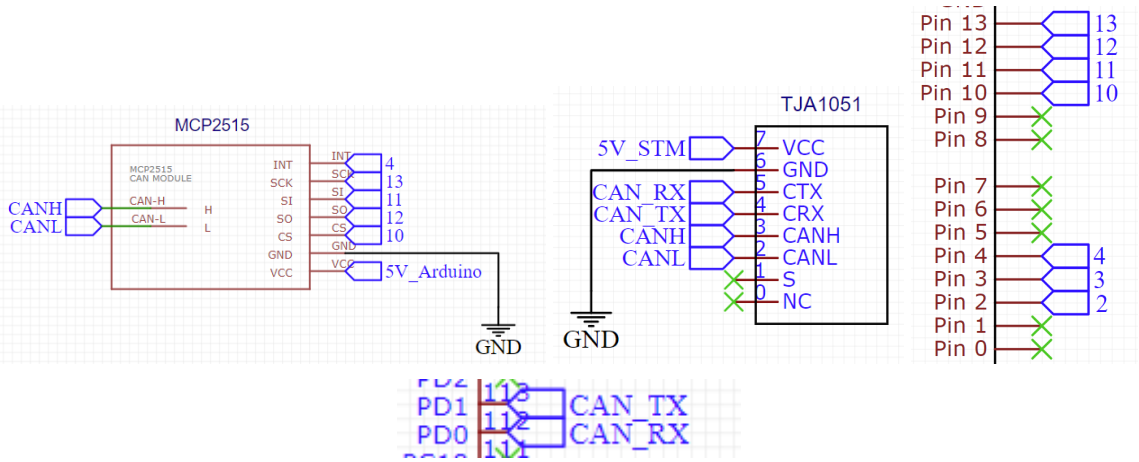


Figure 22. Location in Schematic

6. Testing Process

6.1 Unit tests

<table>
<tr><td>1</td><td colspan="4">Testing serial port operation/22-Aug-2024</td></tr>
<tr><td>Team</td><td colspan="4">5</td></tr>
<tr><td>Responsible</td><td colspan="4">Ximena Trejo</td></tr>
<tr><td>Description</td><td colspan="4">The PC correctly sends a character to the microcontroller using the UART port, and echoes it back.</td></tr>
<tr><td>Function</td><td colspan="4">Monitor distance traveled, coordinates given by camera, and if the tractor is in a waypoint.</td></tr>
<tr><td>Components</td><td colspan="4">UART driver, main program.</td></tr>
<tr><td>Hardware</td><td colspan="4">Serial port, computer with serial interface, extension cable.</td></tr>
<tr><td rowspan="4">Procedure</td><td></td><td>Expected behavior with acceptance criteria</td><td>Actual behavior</td><td>Result passed or not</td></tr>
<tr><td>Initial conditions: Observed using debugger with breakpoint</td><td>Serial buffer initialized to zero</td><td>Serial buffer initialized to zero</td><td>OK</td></tr>
<tr><td>Step 1: Start Putty with the right baud rate and start the program</td><td>The microcontroller program should block and wait for someone to press a key on Putty</td><td>Microcontroller displays nothing on Putty. Press the A key.</td><td>OK</td></tr>
<tr><td>Step 2: Type a key at the terminal and the character is echoed</td><td>The microcontroller program should continue execution, and the rx_buffer should contain the character typed in. The program transmits back the character plus one to the PC.</td><td>The character pressed appears on the screen. The next character in ASCII appeared.</td><td>OK</td></tr>
<tr><td>Comments</td><td colspan="4">None.</td></tr>
</table>

<table>
<tr><td>2</td><td colspan="4">Testing wireless connection through NRF/28-Aug-2024</td></tr>
<tr><td>Team</td><td colspan="4">5</td></tr>
<tr><td>Responsible</td><td colspan="4">Valeria Aranza and Jesús Martínez</td></tr>
<tr><td>Description</td><td colspan="4">The Arduino UNO correctly sends a message wirelessly, the STM32 gets it and displays it on the serial terminal.</td></tr>
<tr><td>Function</td><td colspan="4">Get camera coordinates wirelessly.</td></tr>
<tr><td>Components</td><td colspan="4">UART driver, NRF24L01, Arduino UNO, main program, SPI protocol.</td></tr>
<tr><td>Hardware</td><td colspan="4">Serial port, computer with serial interface, extension cable, wireless interface.</td></tr>
<tr><td rowspan="3">Procedure</td><td></td><td>Expected behavior with acceptance criteria</td><td>Actual behavior</td><td>Result passed or not</td></tr>
<tr><td>Initial conditions: Observed using debugger with breakpoint</td><td>NRF buffer initialized to zero</td><td>NRF buffer initialized to zero</td><td>OK</td></tr>
<tr><td>Step 1: Start Putty with the right baud rate and start the program</td><td>The microcontroller program should block and wait for the Arduino to send a message</td><td>The microcontroller only displayed a message once it got it from the Arduino</td><td>OK</td></tr>
<tr><td>Comments</td><td colspan="4">None</td></tr>
</table>

3	Testing tractor direction/05-Sep-2024			
Team	5			
Responsible	Ximena Trejo			
Description	The servo moves in different angles.			
Function	Change the tractor's direction.			
Components	UART driver, PWM, main program.			
Hardware	Serial port, computer with serial interface, extension cable, SPI interface, wireless interface.			
Procedure		**Expected behavior with acceptance criteria**	**Actual behavior**	**Result passed or not**
	Initial conditions: Observed using debugger with breakpoint	NRF buffer initialized to zero	NRF buffer initialized to zero	OK
	Step 1: PWM sends pulse widths to the servomotor, translating into angles	The servo goes from 0-180°	The servo goes from 0-180°	OK
Comments	None			

4	Testing ESC/02-Oct-2024			
Team	5			
Responsible	Valeria Aranza and Jesus Martínez			
Description	The ESC correctly controls the DC motor through pulse widths sent by the STM32.			
Function	Control DC motor's direction and speed.			
Components	Speed controller, main program, PWMJ, GA25-370 DC motor with Encoder, ESC.			
Hardware	Extension cable.			
Procedure		**Expected behavior with acceptance criteria**	**Actual behavior**	**Result passed or not**
	Initial conditions: Observed using debugger with breakpoint	TIM16 is correctly initialized for ESC	TIM16 is initialized for ESC	OK
	Step 1: ESC sends pulse widths to the DC motor.	The DC motor goes forward, and then backwards.	The DC motor goes forward, and then backwards.	OK
Comments	None			

<table>
<tr><td>5</td><td colspan="4">Testing CAN protocol/26-Nov-2024</td></tr>
<tr><td>Team</td><td colspan="4">5</td></tr>
<tr><td>Responsible</td><td colspan="4">Ximena Trejo and Dafne Reyes</td></tr>
<tr><td>Description</td><td colspan="4">A CAN message is sent from the Arduino UNO, and the STM32 correctly gets it and displays it on the serial monitor.</td></tr>
<tr><td>Function</td><td colspan="4">Get encoder values from the Arduino UNO.</td></tr>
<tr><td>Components</td><td colspan="4">UART driver, CAN protocol, main program, TJA1051, MCP2515, Arduino UNO.</td></tr>
<tr><td>Hardware</td><td colspan="4">Serial port, computer with serial interface, extension cable.</td></tr>
<tr><td rowspan="3">Procedure</td><td></td><td>Expected behavior with acceptance criteria</td><td>Actual behavior</td><td>Result passed or not</td></tr>
<tr><td>Initial conditions: Observed using debugger with breakpoint</td><td>The RX buffer is initialized to zero.</td><td>The RX buffer is initialized to zero.</td><td>OK</td></tr>
<tr><td>Step 1: Start Putty with the right baud rate and start the program</td><td>The microcontroller program should block and wait for the Arduino to send a message</td><td>The microcontroller only displayed a message once it got it from the Arduino</td><td>OK</td></tr>
<tr><td>Comments</td><td colspan="4">None</td></tr>
</table>

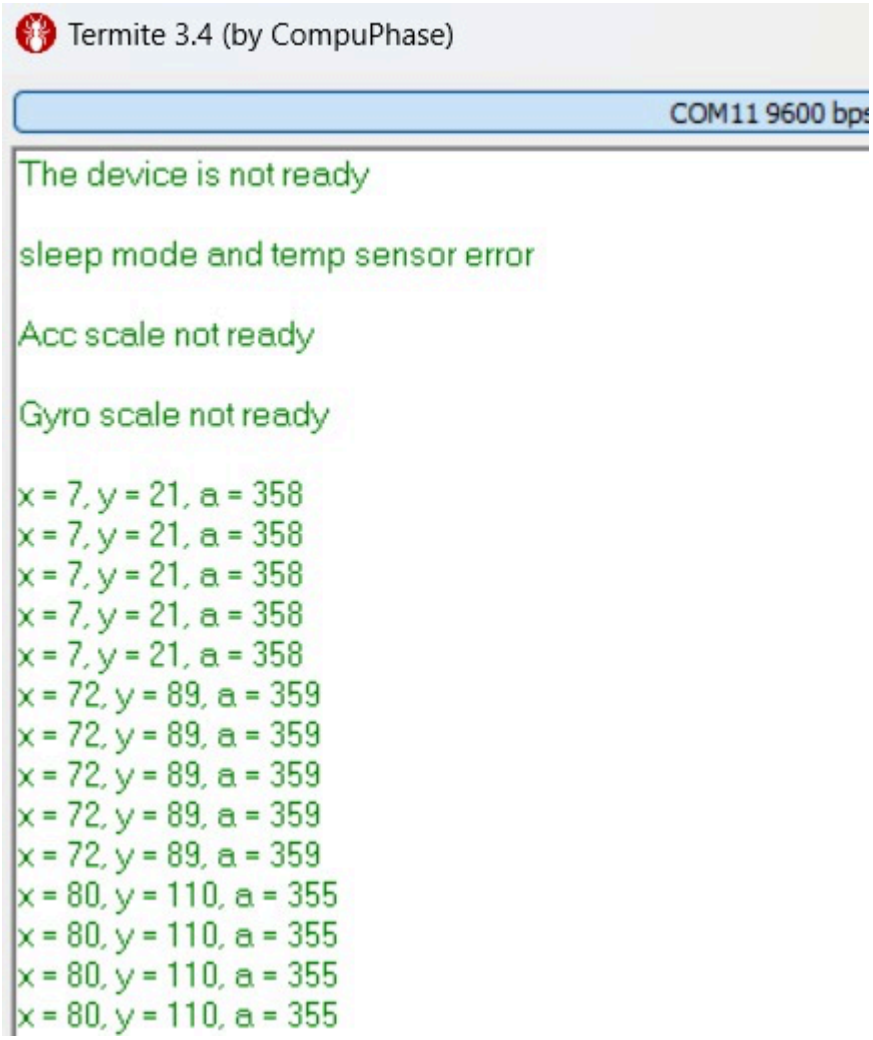
<table>
<tr><td>6</td><td colspan="4">Testing IMU/26-Nov-2024</td></tr>
<tr><td>Team</td><td colspan="4">5</td></tr>
<tr><td>Responsible</td><td colspan="4">Ximena Trejo and Dafne Reyes</td></tr>
<tr><td>Description</td><td colspan="4">The IMU calculates values from its accelerometer and gyroscope, and the program translates those values into coordinates and angles.</td></tr>
<tr><td>Function</td><td colspan="4">A better orientation for the tractor.</td></tr>
<tr><td>Components</td><td colspan="4">UART driver, main program, I2C protocol.</td></tr>
<tr><td>Hardware</td><td colspan="4">Serial port, computer with serial interface, extension cable.</td></tr>
<tr><td rowspan="3">Procedure</td><td></td><td>Expected behavior with acceptance criteria</td><td>Actual behavior</td><td>Result passed or not</td></tr>
<tr><td>Initial conditions: Observed using debugger with breakpoint</td><td>The coordinates and angles buffers are initialized to zero</td><td>The coordinates and angles buffers are initialized to zero</td><td>OK</td></tr>
<tr><td>Step 1: Start Putty with the right baud rate and start the program</td><td>The microcontroller program should block and wait for the IMU to calculate the data, to then display it on the serial monitor</td><td>The microcontroller only displayed data once it got it from the IMU</td><td>OK</td></tr>
<tr><td>Comments</td><td colspan="4">None</td></tr>
</table>

6.2 Integration Tests

<table>
<tr><td>1</td><td colspan="4">Arduino reads the encoder and sends the data to the STM32 through CAN/26-Nov-2024</td></tr>
<tr><td>Team</td><td colspan="4">5</td></tr>
<tr><td>Responsible</td><td colspan="4">Valeria Aranza</td></tr>
<tr><td>Description</td><td colspan="4">The Arduino UNO reads the encoder’s phases, to then send that information through CAN to the STM32, which will display it on the serial monitor. This also includes the ESC functionality.</td></tr>
<tr><td>Function</td><td colspan="4">Have better feedback of the motor’s traveled distance.</td></tr>
<tr><td>Components</td><td colspan="4">UART driver, CAN protocol, main program, PWM, TJA1051, MCP2515, Arduino UNO, JGA25-370 DC motor with Encoder, ESC.</td></tr>
<tr><td>Hardware</td><td colspan="4">Serial port, computer with serial interface, extension cable.</td></tr>
<tr><td rowspan="4">Procedure</td><td></td><td>Expected behavior with acceptance criteria</td><td>Actual behavior</td><td>Result passed or not</td></tr>
<tr><td>Initial conditions: Observed using debugger with breakpoint</td><td>The RX buffer is initialized to zero.</td><td>The RX buffer is initialized to zero.</td><td>OK</td></tr>
<tr><td>Step 1: Start Putty with the right baud rate and start the program</td><td>The microcontroller program should block and wait for the DC motor to move.</td><td>DC motor moved.</td><td>OK</td></tr>
<tr><td>Step 2: The Arduino UNO sends the traveled distance to the STM32.</td><td>The serial monitor only displays something once it has gotten the encoder’s data from the Arduino</td><td>The serial monitor displays the encoder’s data from the Arduino.</td><td>OK</td></tr>
<tr><td>Comments</td><td colspan="4">None</td></tr>
<tr><td>Evidence</td><td colspan="4">https://drive.google.com/file/d/1GODhKM4UAcY_W1MdQdS2STHCZmqREcYf/view?usp=drivesdk</td></tr>
</table>

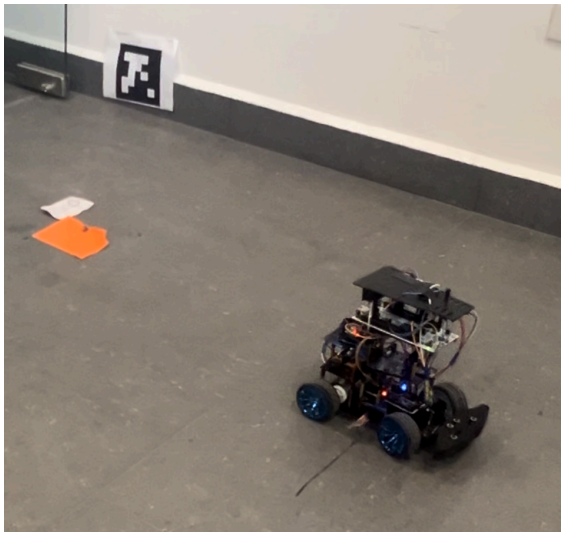

Termite 3.4 (by CompuPhase)
COM11 9600 bps, 8N1, no handshake
Received from Arduino - ID: 0x0, Data: 0x0 0x0 0x0 0x37 0x0 0x0 0x0 0x1F
Decoded Values:
Encoder Position: 55
Distance Traveled: 0.031 meters
Received from Arduino - ID: 0x0, Data: 0x0 0x0 0x0 0x69 0x0 0x0 0x0 0x3B
Decoded Values:
Encoder Position: 105
Distance Traveled: 0.059 meters
Received from Arduino - ID: 0x0, Data: 0x0 0x0 0x0 0x93 0x0 0x0 0x0 0x53
Decoded Values:
Encoder Position: 147
Distance Traveled: 0.083 meters
Received from Arduino - ID: 0x0, Data: 0x0 0x0 0x0 0xAE 0x0 0x0 0x0 0x62
Decoded Values:
Encoder Position: 174
Distance Traveled: 0.098 meters

2	NRF reads the camera's data and sends it to the STM32 through SPI/26-Nov-2024			
Team	5			
Responsible	Jesús Martínez			
Description	The IMU calculates values from its accelerometer and gyroscope, and the program translates those values into coordinates and angles.			
Function	A better orientation for the tractor.			
Components	UART driver, main program, NRF24L01, I2C protocol.			
Hardware	Serial port, computer with serial interface, extension cable.			
Procedure		**Expected behavior with acceptance criteria**	**Actual behavior**	**Result passed or not**
	Initial conditions: Observed using debugger with breakpoint	The coordinates and angles buffers are initialized to zero	The coordinates and angles buffers are initialized to zero	OK
	Step 1: Start Putty with the right baud rate and start the program	The microcontroller program should block and wait for the IMU to calculate the data, to then display it on the serial monitor	The microcontroller only displayed data once it got it from the IMU	OK
Comments	None			
Evidence	https://drive.google.com/file/d/1ee4ukxIpTT7yu3big3McXErIgzd5FjUC/view			


Termite 3.4 (by CompuPhase)
COM11 9600 bps
The device is not ready
sleep mode and temp sensor error
Acc scale not ready
Gyro scale not ready
x = 7, y = 21, a = 358
x = 7, y = 21, a = 358
x = 7, y = 21, a = 358
x = 7, y = 21, a = 358
x = 7, y = 21, a = 358
x = 72, y = 89, a = 359
x = 72, y = 89, a = 359
x = 72, y = 89, a = 359
x = 72, y = 89, a = 359
x = 72, y = 89, a = 359
x = 80, y = 110, a = 355
x = 80, y = 110, a = 355
x = 80, y = 110, a = 355
x = 80, y = 110, a = 355

6.3 System Tests

1	The tractor stops when there is a waypoint/26-Nov-2024			
Team	5			
Responsible	Jesús Martínez			
Description	Based on the coordinates and the data given by the encoder, the tractor stops when it is in a waypoint's range. An LED is also turned on when it reaches the waypoint.			
Function	Signal all the waypoints.			
Components	UART driver, main program, I2C protocol, NRF24L01, PWM, CAN protocol, main program, PWM, TJA1051, MCP2515, Arduino UNO, JGA25-370 DC motor with Encoder, ESC, SPI protocol.			
Hardware	Serial port, computer with serial interface, extension cable.			
Procedure		**Expected behavior with acceptance criteria**	**Actual behavior**	**Result passed or not**
	Initial conditions: Observed using debugger with breakpoint	The coordinates, RX and angles buffers are initialized to zero	The coordinates, RX and angle angles buffers are initialized to zero	OK
	Step 1: Start Putty with the right baud rate and start the program	The microcontroller program should block and wait for the camera to send the coordinates, and the NRF should display them	The NRF displayed the coordinates once it got them from the camera	OK
	Step 2: Wait for the Arduino UNO to read and send the encoder data	The STM32 will display the traveled distance from the encoder.	The STM32 displayed the traveled distance.	OK
	Step 3: The tractor is aware of the waypoints, and it waits until it reaches one.	Once it detects a waypoint, it stops and turns on an LED.	The tractor stopped at the waypoint, and turned on the LED.	OK

Comments	None
Evidence	https://drive.google.com/file/d/1a9hiq_vrZ5bTBFgntaizkSFiR-nAFY37/view?usp=drivesdk https://drive.google.com/file/d/11fOlvXYLqtKNRiSTX3lO2g9bTGwXKXFz/view?usp=drivesdk 

6.4 Testing Area

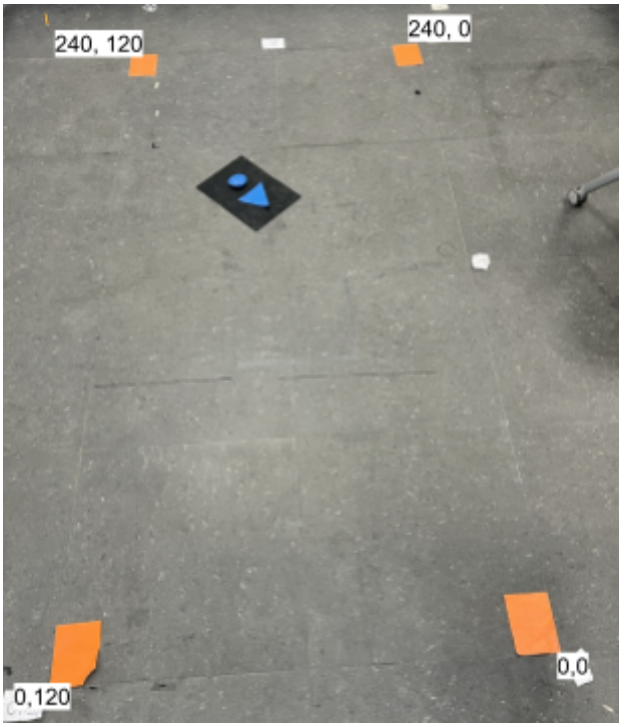


Figure 23. Testing area and its coordinates in centimeters

6.5 Code Testing

6.5.1 Function Waypoint Path Encoder

```
void waypointPathEncoder(float position, uint16_t x, uint16_t y, uint16_t a) {
    // Detener el coche inicialmente
    //stopCar();
    // Variables para rango de posición
    float rangeStart = 0.0f;  // Inicio del rango
    float rangeEnd = 0.8f;    // Fin del rango

    // Variables de control
    bool isWithinStartZone = (x >= 0 && x < 20) && (y > 115 && y <= 120);
    bool isParabolicZone = (x >= 80 && x < 90);
    //setEscSpeed(1300);   // Velocidad hacia adelante

    // 1. Forward movement in start zone
    if (isWithinStartZone) {
        if (position >= rangeStart && position <= rangeEnd) {
            setServoAngle(70);    // Ángulo neutral del servo
            setEscSpeed(1300);   // Velocidad hacia adelante
            printf("Camera coordinates T3: x=%u, y=%u, a=%u\n\n", x, y, a);
            printf("Distance Traveled: %.3f\n", position);
            osDelay(500);
        }
        else if (position > rangeEnd) {
            stopCar();  // Detener el coche si excede el rango
            printf("Out of range! Position exceeded: %.3f\n", position);
        }
    }
    // 2. Parabolic trajectory in specified zone
    else if (isParabolicZone) {
        HAL_GPIO_TogglePin(LEDWay_GPIO_Port, LEDWay_Pin);
        osDelay(200);


       // Parámetros de la parábola
       float a_coeff = 0.005f;  // Mayor curvatura
       float b_coeff = 0.5f;    // Ajusta pendiente según sea necesario
       float c_coeff = 0.0f;    // Offset inicial

       // Cálculo de la trayectoria parabólica
       float parabolicY = a_coeff * x * x + b_coeff * x + c_coeff;

       // Ajuste del servo y velocidad según el valor calculado
       if (parabolicY > 20) {
           setServoAngle(50);   // Gira a la derecha
           setEscSpeed(1350);  // Reducir velocidad
       } else if (parabolicY < 10) {
           setServoAngle(130);  // Gira a la izquierda
           setEscSpeed(1350);  // Ajuste de velocidad
       } else {
           setServoAngle(90);   // Recto
           setEscSpeed(1350);   // Velocidad estándar
       }

       printf("Parabolic trajectory: x = %u, y = %.2f, a = %u\n", x, parabolicY, a);
       osDelay(500); // Delay para suavidad de movimiento
    }
    // 3. Handling other zones or fallback behavior
    else {
        osDelay(500); // Ajustar posición si es necesario
    }
    stopCar();
    // Delay final para permitir estabilidad
    osDelay(500);
}
```

6.5.2 Function Task 3: Receiving the Queue

```
void startTask3(void const *argument){
    osEvent evt;
    for(;;)
    {
        evt = osMessageGet(msgQueueHandle, osWaitForever);
        if(evt.status == osEventMessage){
            if (osMutexWait(Mutex1Handle, osWaitForever) == osOK){
                positionMessage* receivedMsg = (positionMessage*) evt.value.p;

                //testingServoLimits();
                //testingESCLimits();
                printf("Position: %f\n\r", distanceTraveled);
                waypointPathEncoder(distanceTraveled);
                osMutexRelease(Mutex1Handle); // Liberar el mutex después de usarlo


            }
        }
        osDelay(50);
    }
}
```

```
    }
    // Zone 3: Handling other positions
    else {
        printf("Outside Defined Zones: Position = %.3f\n", position);
        stopCar();
        osDelay(500); // Pause to stabilize
    }

    // Stop the car after each movement or decision
    stopCar();
    osDelay(500); // Allow time for decision-making before the next action
}
```

6.5.3 Testing Waypoints

```
void waypointPathEncoder(float position) {
    // Define position ranges for different zones
    float rangeStartZoneEnd = 0.8f;  // End of start zone
    float rangeParabolicZoneStart = 0.81f;  // Start of parabolic zone
    float rangeParabolicZoneEnd = 0.3f;    // End of parabolic zone

    // Zone 1: Forward movement in the start zone
    if (position >= 0 && position <= rangeStartZoneEnd) {
        setServoAngle(90);    // Straight movement
        setEscSpeed(1300);    // Moderate forward speed
        printf("Start Zone: Position = %.3f\n", position);
        osDelay(500);         // Delay for smooth movement
    }
    // Zone 2: Parabolic trajectory
    else if (position > rangeParabolicZoneStart && position <= rangeParabolicZoneEnd) {
        // Adjust servo and speed based on position in the parabolic zone
        if (position < (rangeParabolicZoneStart + rangeParabolicZoneEnd) / 2) {
            setServoAngle(50);    // Turn right
            setEscSpeed(1350);    // Slightly reduced speed
            printf("Parabolic Zone: Turning Right, Position = %.3f\n", position);
        } else {
            setServoAngle(130);   // Turn left
            setEscSpeed(1350);    // Slightly reduced speed
            printf("Parabolic Zone: Turning Left, Position = %.3f\n", position);
        }
        osDelay(500); // Delay to stabilize trajectory
    }
    // Zone 3: Handling other positions
    else {
        printf("Outside Defined Zones: Position = %.3f\n", position);
        stopCar();
        osDelay(500); // Pause to stabilize
    }
```

6.5.4 Testing NRF24 Wireless Connection

```
void cameraCoord(){
    if (NRF24_available())
    {
        // Leer los datos recibidos en el buffer
        NRF24_read(buffer, 6);

        // Depuración: Imprimir el contenido del buffer
        snprintf(myRxData, sizeof(myRxData), "Raw Buffer: %02X %02X %02X %02X %02X %02X \r\n",
               buffer[0], buffer[1], buffer[2], buffer[3], buffer[4], buffer[5]);
        HAL_UART_Transmit(&huart3, (uint8_t *)myRxData, strlen(myRxData), HAL_MAX_DELAY);

        // Ensamblar valores x, y, y a
        x = (buffer[0] << 8) | buffer[1];  // Big-endian
        y = (buffer[2] << 8) | buffer[3];
        a = (buffer[4] << 8) | buffer[5];

        // Depuración: Imprimir los valores calculados
        snprintf(myRxData, sizeof(myRxData), "x = %u, y = %u, a = %u \r\n", x, y, a);
        HAL_UART_Transmit(&huart3, (uint8_t *)myRxData, strlen(myRxData), HAL_MAX_DELAY);

        HAL_Delay(50);
    }
}
```

6.5.5 Testing Encoder Position

```
void encoderPos(void){
    if (HAL_FDCAN_GetRxMessage(&hfdcan1, FDCAN_RX_FIFO0, &RxHeader, RxData) == HAL_OK)
    {
        printf("\n\rReceived from Arduino - ID: 0x%lx, Data: ", RxHeader.Identifier);

        for (int i = 0; i < 8; i++) {
          printf("0x%X ", RxData[i]);
        }

        // Decodificar los datos recibidos
        // Los primeros 4 bytes son la posición del encoder (int32_t)
        encoderPosition = (int32_t)(
          ((int32_t)RxData[0] << 24) |
          ((int32_t)RxData[1] << 16) |
          ((int32_t)RxData[2] << 8)  |
          ((int32_t)RxData[3])
        );

        // Los siguientes 4 bytes son la distancia recorrida (float, en milímetros)
        uint32_t distanceRaw = (uint32_t)(
          ((uint32_t)RxData[4] << 24) |
          ((uint32_t)RxData[5] << 16) |
          ((uint32_t)RxData[6] << 8)  |
          ((uint32_t)RxData[7])
        );

        distanceTraveled = distanceRaw / 1000.0f; // Convertir milímetros a metros

        // Imprimir los valores decodificados
        printf("\n\rDecoded Values:\n\r");
        printf("  Encoder Position: %ld\n\r", encoderPosition);
        printf("  Distance Traveled: %.3f meters\n\r", distanceTraveled);
    }
```

7. Conclusions

Throughout the semester, we encountered many topics that would later become part of the challenge. Protocols such as SPI, CAN and I2C were relevant, since they were a key part for obtaining data from both the camera and other sensors. Integrating them into the final product led to an autonomous device that is able to perform the task that John Deere aspired to see, which is stopping at certain waypoints defined by the developers.

Some challenges that were also present during the development of the challenge were the delay that the camera had for sending data and scheduling the tasks in FreeRTOS, since both of them were a key part for the tractor to function. These difficulties were solved by applying delays to the system, among other considerations.

This project can be applied on a large scale, demonstrating its scalability and the achievement of the competences that were clarified at the beginning of the course. The integration gave a great idea of what an embedded systems project is, and both new and old knowledge were applied to solve the challenge.

8. Bibliography

DigiKey. (w.d.). JGA25-370 Geared Motor. Retrieved from https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/114090046_Web.pdf

Electrónicos Caldas. (w.d.). MG996R High Torque Metal Gear Dual Ball Bearing Servo. Retrieved from https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf

Indiamart. (w.d.). 20A Brush ESC 20A Brushed Electronic Speed Controller w/Brake for RC Car Boat Tank - RS3005. Indiamart. Retrieved from https://www.indiamart.com/proddetail/20a-brush-esc-20a-brushed-electronic-speed-controller-w-brake-for-rc-car-boat-tank-rs3005-2853674232755.html

InvenSense. (w.d.). MPU-9250 Product Specification Revision 1.1. Retrieved from https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

Microchip. (w.d.). Stand-Alone CAN Controller with SPI Interface. Retrieved from https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf

Nordic Semiconductor. (w.d.). nRF24L01+ Single Chip 2.4GHz Transceiver. Retrieved from https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

NXP. (w.d). TJA1051 High-speed CAN transceiver. Retrieved from https://www.nxp.com/docs/en/data-sheet/TJA1051.pdf

STMicroelectronics. (w.d.). UM2408 User Manual. Retrieved from https://www.st.com/resource/en/user_manual/um2408-stm32h7-nucleo144-boards-mb1363-stmicroelectronics.pdf

Texas Instruments. (w.d.). LM2596 SIMPLE SWITCHER Power Converter 15’-kHz 3-A Step-Down Voltage Regulator. Retrieved from https://www.ti.com/lit/ds/symlink/lm2596.pdf

9. Appendixes

Appendix 1: Task 1 (Encoder's data)

```
void startTask1(void const *argument) { // Read Arduino CAN Message
    //positionMessage msg;
    for (;;) {
        if (HAL_FDCAN_GetRxMessage(&hfdcan1, FDCAN_RX_FIFO0, &RxHeader, RxData) == HAL_OK) {
            if (osMutexWait(Mutex1Handle, osWaitForever) == osOK) {
                // Decodificar los datos recibidos para encoderPosition (uint32_t)
                uint32_t encoderPosition = (uint32_t)(
                    ((uint32_t)RxData[0] << 24) |
                    ((uint32_t)RxData[1] << 16) |
                    ((uint32_t)RxData[2] << 8) |
                    ((uint32_t)RxData[3])
                );

                // Decodificar los datos recibidos para distanceTraveled (uint32_t)
                uint32_t distanceRaw = (uint32_t)(
                    ((uint32_t)RxData[4] << 24) |
                    ((uint32_t)RxData[5] << 16) |
                    ((uint32_t)RxData[6] << 8) |
                    ((uint32_t)RxData[7])
                );

                // Convertir distanceRaw de milímetros a metros
                distanceTraveled = distanceRaw / 1000.0f;
                //msg.position = distanceTraveled;

                // Verificar los datos decodificados
                printf("\n\rDecoded Values:\n\r");
                //printf("  Encoder Position: %ld\n\r", encoderPosition); // Signed integer
                //printf("  Distance Traveled: %f meters\n\r", msg.position); // Unsigned float
                //osMessagePut(msgQueueHandle, (uint32_t)&msg, 1);

                osMutexRelease(Mutex1Handle); // Liberar el mutex después de usarlo
            } else {
                // Agregar un mensaje de error si no se puede obtener el mutex
               printf("Task1: Failed to acquire mutex\n");
            }
        }
        osDelay(50); // Pausa de 10 ms para evitar sobrecargar la CPU
    }
}
```

Appendix 2: Task 2 (Camera's coordinates and angle)

```
void startTask2(void const *argument) {
    //HAL_GPIO_TogglePin(LEDWay_GPIO_Port,LEDWay_Pin);
    //HAL_Delay(100);
    // Buffer de recepción del NRF24
    uint8_t buffer[6];
    // Buffer de salida con un tamaño adecuado
    char myRxData[64]; // Ajustado para manejar toda la salida sin truncamiento
    positionMessage msg;

    for (;;) {
        if (NRF24_available()) {
            if (osMutexWait(Mutex1Handle, osWaitForever) == osOK){
                // Leer los datos recibidos en el buffer
                NRF24_read(buffer, sizeof(buffer));

                // Depuración: Imprimir el contenido del buffer
                int len = snprintf(myRxData, sizeof(myRxData),
                                   "Raw Buffer: %02X %02X %02X %02X %02X %02X \r\n",
                                   buffer[0], buffer[1], buffer[2], buffer[3], buffer[4], buffer[5]);

                // Verificar si el mensaje fue truncado
                if (len >= sizeof(myRxData)) {
                    printf("Advertencia: Truncamiento de datos en myRxData\n");
                }

                HAL_UART_Transmit(&huart3, (uint8_t *)myRxData, strlen(myRxData), HAL_MAX_DELAY);

                // Ensamblar valores x, y, y a
                x = (buffer[0] << 8) | buffer[1]; // Big-endian
                y = (buffer[2] << 8) | buffer[3];
                a = (buffer[4] << 8) | buffer[5];

                // Depuración: Imprimir los valores calculados
                len = snprintf(myRxData, sizeof(myRxData),
                               "x = %u, y = %u, a = %u \r\n", x, y, a);
                 msg.x = x;
                 msg.y = y;
                 msg.a = a;

                 if (len >= sizeof(myRxData)) {
                     printf("Advertencia: Truncamiento de datos en myRxData\n");
                 }

                 HAL_UART_Transmit(&huart3, (uint8_t *)myRxData, strlen(myRxData), HAL_MAX_DELAY);
                 osMessagePut(msgQueueHandle, (uint32_t)&msg, 1);
                 osMutexRelease(Mutex1Handle); // Liberar el mutex después de usarlo
            }

            /*
            else
            {
                    osDelay(10); // Pausa de 10 ms cuando no hay datos
            }
            */
        }
        osDelay(500);
    }
}
```

Appendix 3: Task 3 (Tractor's movement and waypoint identification)

```
void startTask3(void const *argument){
    osEvent evt;
    for(;;)
    {
        evt = osMessageGet(msgQueueHandle, osWaitForever);
        if(evt.status == osEventMessage){
            if (osMutexWait(Mutex1Handle, osWaitForever) == osOK){
                positionMessage* receivedMsg = (positionMessage*) evt.value.p;

                //testingServoLimits();
                //testingESCLimits();
                printf("Position: %f\n\r", distanceTraveled);
                waypointPathEncoder(distanceTraveled, receivedMsg->x, receivedMsg->y, receivedMsg->a);
                osMutexRelease(Mutex1Handle); // Liberar el mutex después de usarlo


            }
        }
        osDelay(50);
    }
}
```

Appendix 4: Routine for waypoint identification

```
void waypointPathEncoder(float position, uint16_t x, uint16_t y, uint16_t a) {
    // Detener el coche inicialmente
    //stopCar();
    // Variables para rango de posición
    float rangeStart = 0.0f;  // Inicio del rango
    float rangeEnd = 0.8f;    // Fin del rango

    // Variables de control
    bool isWithinStartZone = (x >= 0 && x < 20) && (y > 115 && y <= 120);
    bool isParabolicZone = (x >= 80 && x < 90);
    //setEscSpeed(1300);   // Velocidad hacia adelante

    // 1. Forward movement in start zone
    if (isWithinStartZone) {
        if (position >= rangeStart && position <= rangeEnd) {
            setServoAngle(70);    // Ángulo neutral del servo
            setEscSpeed(1300);   // Velocidad hacia adelante
            printf("Camera coordinates T3: x=%u, y=%u, a=%u\n", x, y, a);
            printf("Distance Traveled: %.3f\n", position);
            osDelay(500);
        }
        else if (position > rangeEnd) {
            stopCar();  // Detener el coche si excede el rango
            printf("Out of range! Position exceeded: %.3f\n", position);
        }
    }
    // 2. Parabolic trajectory in specified zone
    else if (isParabolicZone) {
        HAL_GPIO_TogglePin(LEDWay_GPIO_Port, LEDWay_Pin);
        osDelay(200);

        // Parámetros de la parábola
        float a_coeff = 0.005f;  // Mayor curvatura
        float b_coeff = 0.5f;    // Ajusta pendiente según sea necesario
```

```
        float c_coeff = 0.0f;    // Offset inicial

        // Cálculo de la trayectoria parabólica
        float parabolicY = a_coeff * x * x + b_coeff * x + c_coeff;

        // Ajuste del servo y velocidad según el valor calculado
        if (parabolicY > 20) {
            setServoAngle(50);   // Gira a la derecha
            setEscSpeed(1350);  // Reducir velocidad
        } else if (parabolicY < 10) {
            setServoAngle(130);  // Gira a la izquierda
            setEscSpeed(1350);  // Ajuste de velocidad
        } else {
            setServoAngle(90);   // Recto
            setEscSpeed(1350);   // Velocidad estándar
        }

        printf("Parabolic trajectory: x = %u, y = %.2f, a = %u\n", x, parabolicY, a);
        osDelay(500); // Delay para suavidad de movimiento
    }
    // 3. Handling other zones or fallback behavior
    else {
        osDelay(500); // Ajustar posición si es necesario
    }
    stopCar();
    // Delay final para permitir estabilidad
    osDelay(500);
}
```