V2V Communication
Emulating real car actions

MEMBERS

HOSSAM ELWAHSH

AHMED SAKR

AHMED HISHAM

NADA SALLOUM

SALMA FARAGALLA

NORHAN MOHAMED

جامعة مصر للمعلوماتية
EGYPT UNIVERSITY OF INFORMATICS

Ministry of Communications and Information Technology

eme
egypt makes electronics

SEITECH
SMART EMBEDDED & INDUSTRIAL TECHNOLOGY

V2V Communication
Emulating Real Car Actions

1. Introduction

1.1. Summary

The EME - Egypt Makes Electronics V2V communication project is a system that clones, processes, and sends certain car actions from a Fiat Tipo 2019 to an RC car.

The system uses two STM32 microcontrollers running FreeRTOS and two ESP8266 chips communicating over ESP-NOW protocol to minimize delay. One of the ESP8266 chips has an active web server that can be accessed to monitor the RC car status in real-time.

The system is currently able to clone the following car actions:

- Throttle position
- Steering wheel position
- Current automatic transmission selection (P, R, N, D)
- Selection of car lighting

1.2. Goals

- To develop a system that can clone the actions of a real car on an RC car
- To use ESP-NOW protocol to minimize delay in communication between the different components of the system
- To develop a web server that can be used to monitor the RC car status in real-time
- Demonstrate V2V Communication: Showcase the ability to establish reliable and low-latency communication between two vehicles, emulating a real-world V2V scenario.
- Real-time Interaction: Enable real-time interaction between the Fiat Tipo and the RC car, allowing for instantaneous communication and control.
- IoT Integration: Explore how this automotive IoT system can be integrated with other IoT devices and smart home automation.

1.3.

Application

- Driver Assistance and Training: This system can be used to assist new or inexperienced drivers in practicing their driving skills in a controlled environment. The RC car can mimic the movements of the real car, helping the driver learn and practice maneuvers safely.

- Entertainment and Gaming: Enthusiasts and hobbyists can use this project for remote-controlled car racing or gaming. It can be part of an interactive gaming experience or a unique form of entertainment.
- Real-time Monitoring and Data Collection: The active web server on one of the ESP8266 chips allows for real-time monitoring of the RC car's status. This feature can be useful for collecting data and analyzing vehicle behavior in different scenarios.
- Education and Training: This project can be used for educational purposes in schools and universities to teach students about embedded systems, real-time operating systems (FreeRTOS), and wireless communication protocols. It's a practical way to demonstrate concepts in automotive engineering and IoT

The team is also working on developing new features, such as automatically engaging the brakes if the car is drifting on neutral towards an obstacle, and automatic corner lights with steering.

1.4. Hardware Components

Component	Quantity
STM32F103C8T6 Microcontroller	2
ESP8266	3
Ultrasonic Sensor	2
100nF Capacitor	4
L298N DC Motor Driver	1
DC Motors	4
MCP2551 CAN-BUS Shield	1
SSD1306 OLED	1
RC Car Frame	1
5mm Red LED (Brakes)	2
5mm Yellow LED (Hazard Lights)	2
5mm White LED (Reverse Lights)	2
10 mm Yellow LED (Front Lights)	2
Raspberry Pi 4 ***(optional)****	1
MCP2515 Stand-Alone CAN Controller ***(optional)****	1

* For faster debugging, analysis and testing.
**To be used with the Raspberry Pi as it doesn't contain a CAN controller nor a transceiver.MCP2515 can be used over SPI with the RPi.

1.5. System Components

The system will consist of the following components:

1. A microcontroller to tap into the HS-CAN network and collect vehicle data then analyzes, filters, and map these data to a single byte for faster transfers
2. A wireless communication module (ESP8622) to broadcast the data to the two other wireless modules
3. one of which is set up on the RC Car and the other is a standalone with a web server to monitor the data and show the latest RC state
4. Another microcontroller installed on the RC Car to receive the sent bytes parses them and initiates the required actions to mimic the real car and includes a safety module to monitor the front and back sensors to override throttle response and prevent accidents.

1.6. System Requirements

1.6.1. ECU 1 (Fiat Side)

- Implement the system using STM32F103 MC based on **FreeRTOS Kernel V10.0.1**
- Enable RTOS preemption and give all tasks equal priorities to run on a round robin basis.
- Configure CAN Speed to 500kbps to match HS-CAN on Fiat Tipo Car, enable CAN receive interrupts
- Configure UART baud rate to **115200,** stop bits: **1,** parity: **none**.
- Connect ESP8266 to UART
- Connect MCP2551 to CAN
- Connect transceiver CANH and CANL pins to Fiat HS-CAN.
- Analyze and extract CAN message IDs and bits responsible for the following from Fiat Tipo CAN Bus:
    - Throttle position
    - Steering wheel position
    - Automatic transmission selection (P, R, N, D)
    - Selection of car lighting
        - Brake lights
        - Front lights
        - Hazard lights (left and right indicators)
        - Reverse lights
- Map all the messages to a single byte only with the higher nibble being a standard agreed on header for each option (throttle, steering, …) and the lower nibble holding the data
- Map the throttle values to control the DCM speed of the RC car divided into 5 levels
    - STOP
    - Speed level 1
    - Speed level 2
    - Speed level 3
    - Speed level 4
- Map steering values to 5 levels
    - Sharp left
    - Left
    - Straight
    - Right
    - Sharp right
- Provide an app_config.h file to be able to configure the previous mapping if needed
- Send mapped data over UART to ESP8266

- Handle and prevent duplicate data from being sent e.g. throttle range could be trying to resend speed level 1 due to changes in the throttle position but still in the mapped range of speed level 1

- 1.6.1.1. RTOS Tasks
  - Task 1: processes received can data and maps them to the required values
  - Task 2: sends processed data over UART to ESP8266

1.6.2. ECU 2 (RC Car Side)

- Implement the system using STM32F103 MC based on **FreeRTOS Kernel V10.0.1**
- Configure UART baud rate to **115200,** stop bits: **1,** parity: **none**.
- Connect ESP8266 to UART
- Implement appropriate tasks to handle executing received actions immediately
- Implement an accident avoidance safety feature to override throttle response and stop the vehicle if it is about to hit an object, use front or rear ultrasonic depending on the current transmission selection (drive / reverse), turn off ultrasonics in neutral and park mode to save power.
- Implement a safety feature to turn off the throttle response if the brakes was pressed.
- Handle stopping motors immediately when in neutral or park mode overriding throttle input to mimic real car gears

1.6.3. ESP8266 (Sender - Fiat Side)

- Configure UART to match with the previous ECUs
- Use ESP-NOW protocol for faster transmissions
- Set ESP8266 mode as Master
- Pair with the other 2 ESPs exclusively
- Broadcast data immediately to paired devices when they are received on UART

1.6.4. ESP8266 (Receiver - RC Side)

- Configure UART to match with the connected ECU
- Use ESP-NOW protocol for faster transmission
- Set ESP8266 mode as Slave
- Wait for data sent from the Master
- Transmit data on UART

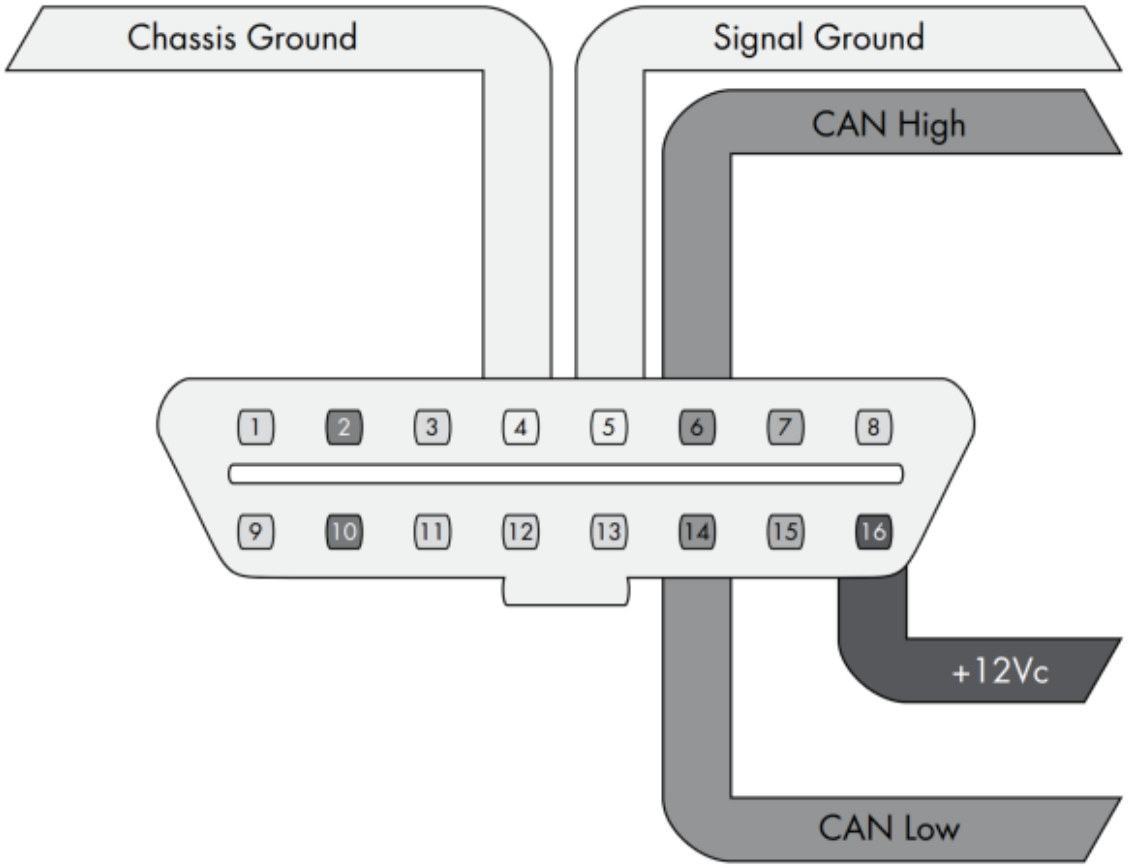
1.6.5. ESP8266 (Receiver - Standalone web server)

- Enable ESP-NOW
- Set ESP8266 mode as Slave

- Define a static SSID and password to connect to upon startup
- Implement a web server GUI using HTML/CSS/JS to show and monitor the current RC Car status

2. Analysis and Findings

- Using RPi 4, with MCP2515 Can Controller/Transceiver module over SPI and with the help of “The Car Hacker Handbook” we were able to correctly tap into the Car HS-CAN bus through pins 6 and 14 of the OBD port.



- After comprehensive monitoring and filtration, we were able to extract the required information as follows:

Module	Description	Fiat Tipo Hex	RC Hex (map)	RC Decimal (map)
Throttle	Throttle STOP	0000 - 000A	0x10	16
Throttle	Throttle 70%	000B - 03EB	0x11	17
Throttle	Throttle 80%	03EC - 0FA0	0x12	18
Throttle	Throttle 90%	0FA1 - 2EE1	0x13	19
Throttle	Throttle 100%	2EE2 - 401F	0x14	20
Transmission	Neutral	0x0F	0x70	112
Transmission	Drive	0x1F	0x71	113
Transmission	Reverse	0x7F	0x77	119
Transmission	Parking	0xFF	0x7F	127
Lighting	Brake lights off	0x00000000	0x80	128

Lighting	Brake lights on	0x00000400	0x81	129
Lighting	Right Indicators off	0x00000000	0x90	144
Lighting	Right Indicators on	0x00002000	0x91	145
Lighting	Left Indicators off	0x00000000	0xA0	160
Lighting	Left Indicators on	0x00800000	0xA1	161
Lighting	Front lights off	0x00000000	0xC0	192
Lighting	Front lights on	0x02000000	0xC1	193
Lighting	Reverse lights off	transmission (R)	0xD0	208
Lighting	Reverse lights on	transmission (R)	0xD1	209
Steering	Straight	0x19 - 0x20	0xE0	224
Steering	Right	0x11 - 0x18	0xE2	226
Steering	Sharp right	0x09 - 0x10	0xE3	227
Steering	Left	0x21 - 0x26	0xE8	232
Steering	Sharp left	0x27 - 0x2C	0xEC	236

3. Testing Notes

3.1. Unit Testing

- Each module was tested alone using test functions simulating the appropriate environment and actions (e.g. releasing a semaphore) to test each module on its own before integrating with other modules, this greatly reduced our debugging time in the integration part as there weren't many issues at the point.

3.2. Integration Testing

- RPi was also useful in integration testing as to simulate the car CAN data that would be sent to the STM32 MC to filter and process, extensive testing was done to ensure that all the above data were mapped correctly and that the actions are taken appropriately by the RC car.
- Cansend library in RPi was used along with some manually written bash scripts to simulate tests for the above table data.
- Tests were verified by debugging and intercepting the RX interrupts, by viewing the monitoring GUI on the ESP webserver and by examining the RC car response itself.

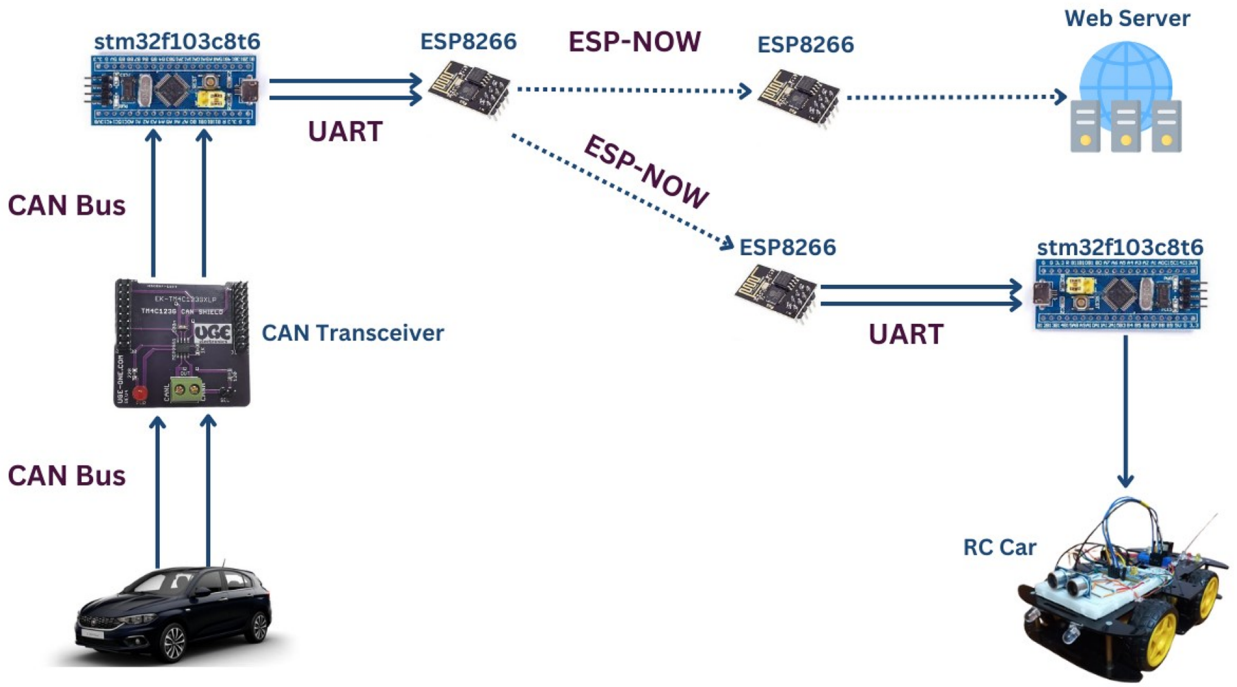
3.3. System Testing

- Final system testing was done on the real fiat hardware to ensure that everything is working correctly, and fortunately simulating the data in the previous testing stages using the raspberry made a huge difference in the system testing stage as we faced minor issues with most if not all of the features were running correctly from the first trial.
- We just had to tweak the throttle mapping ranges a bit and fix a minor issue with the left indicators being switched with the right indicators.

4. High Level Design

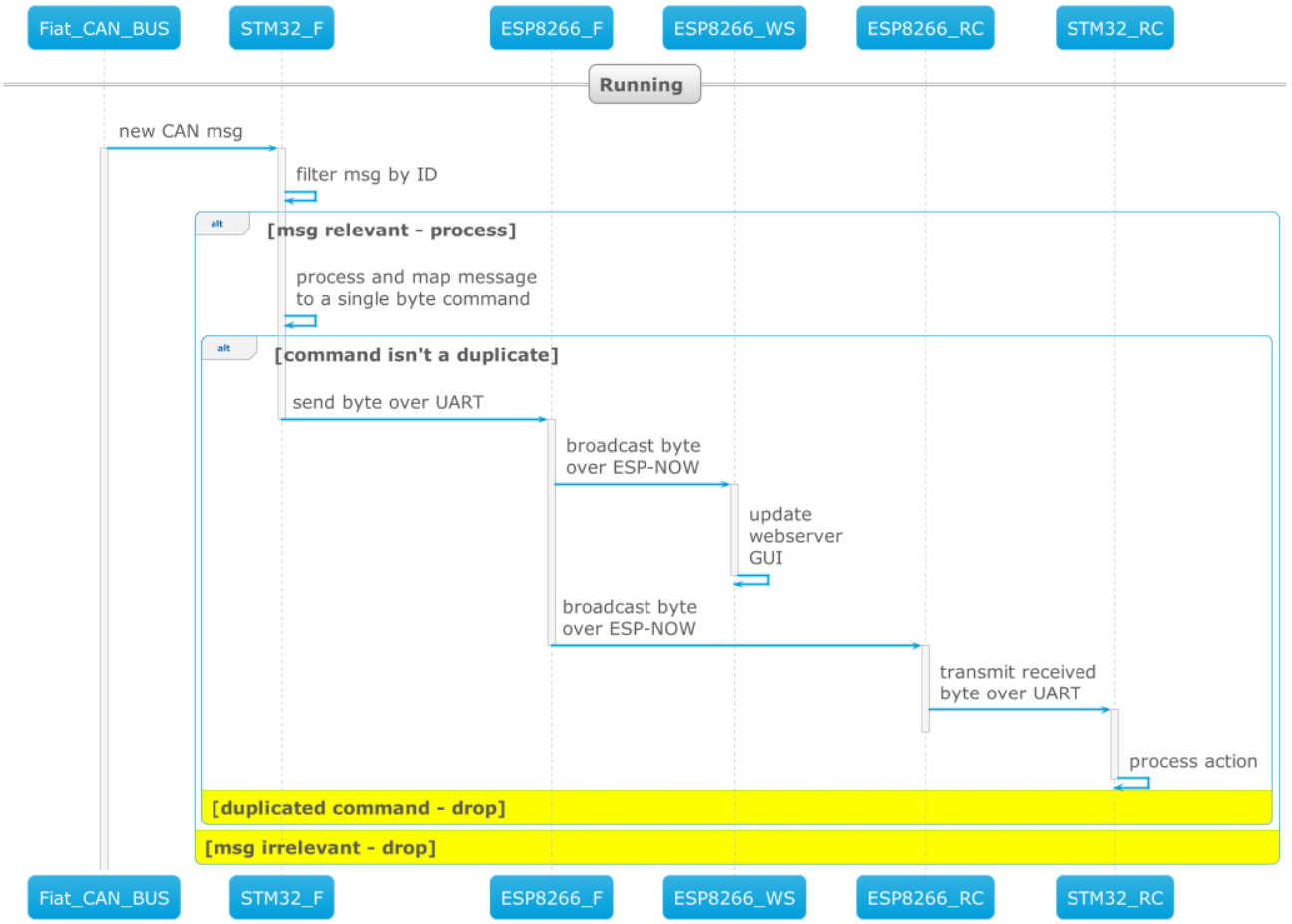
4.1. System Architecture

4.1.1. Overview



1. The CAN Transceiver is connected to the Fiat Tipo CAN bus via pins 6 and 14 in the OBD-II connector , which are the HS-CAN lines (with a speed of 500kbps).
2. The CAN Transceiver forwards the CAN messages sent by different ECUs in the Fiat Tipo to the STM32 microcontroller.
3. The STM32 microcontroller filters these CAN messages by IDs, maps them to a single byte for faster transfer, and sends them to the RC car microcontroller (also STM32) via ESP8266 so that the RC car can mimic the actions of the Fiat Tipo.
4. The mapped messages are also sent to another ESP8266 that hosts a web server to monitor the RC car actions.

4.1.2. Brief Sequence Diagram



4.2. ECU 1 (Fiat Side)

4.2.1. Layered Architecture

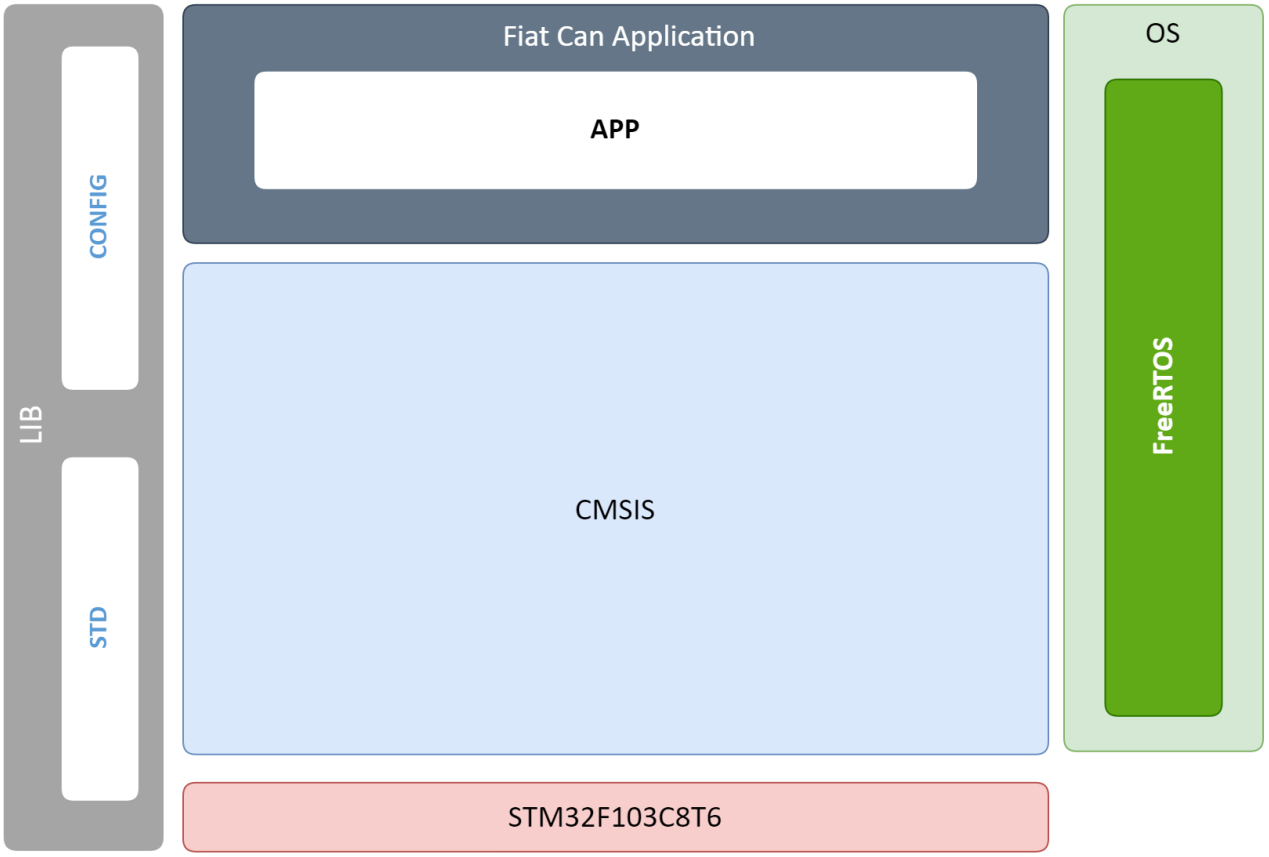


Figure 1. Layered Architecture Design

4.2.2. Circuit Schematic

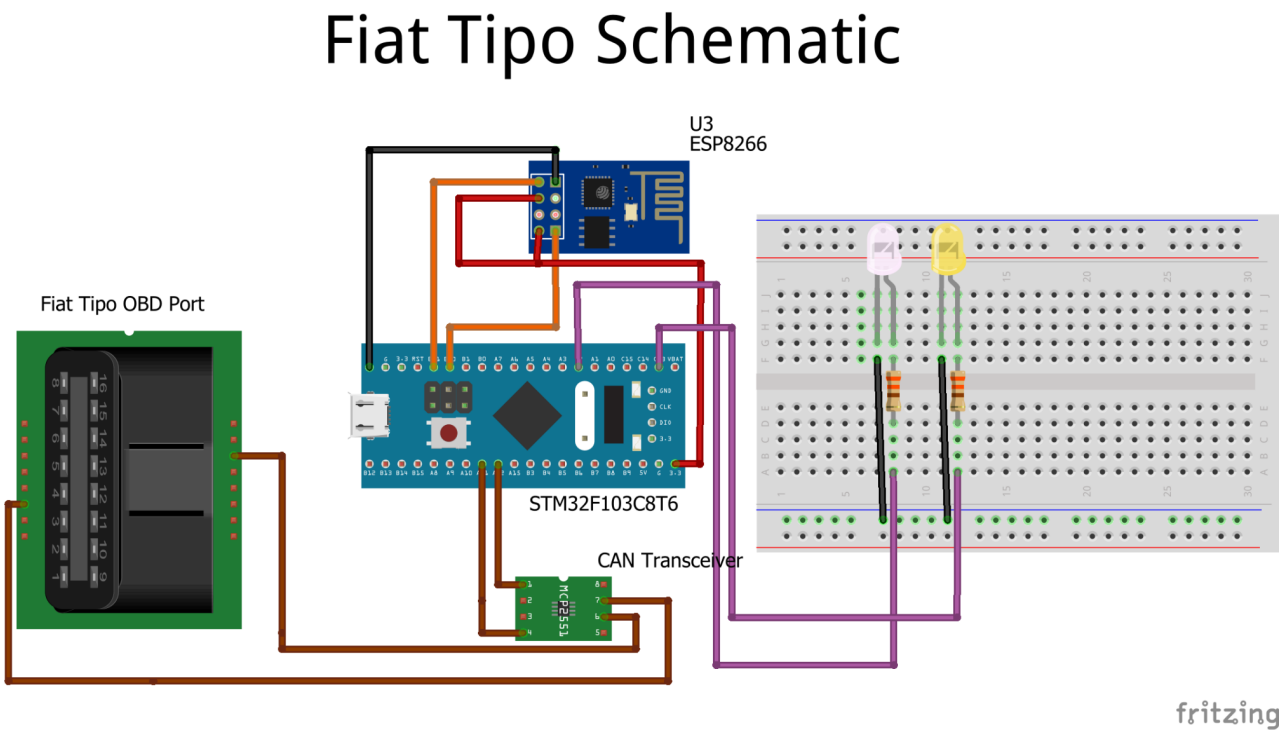
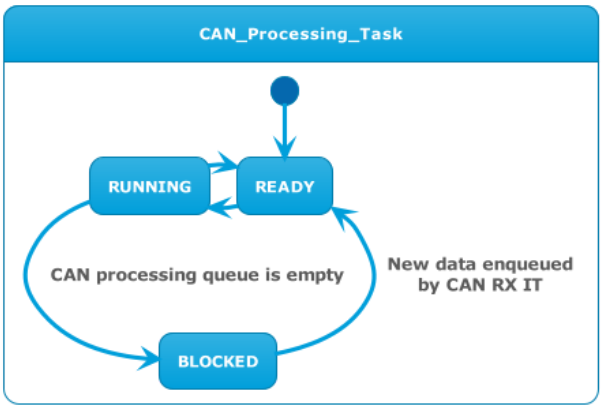
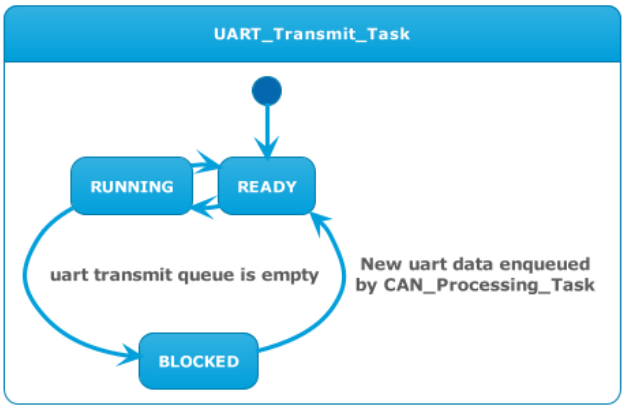


Figure 2. Project Circuit Schematic

4.2.3. Tasks State Diagrams





4.3. ECU 2 (RC Side)

4.3.1. Layered Architecture

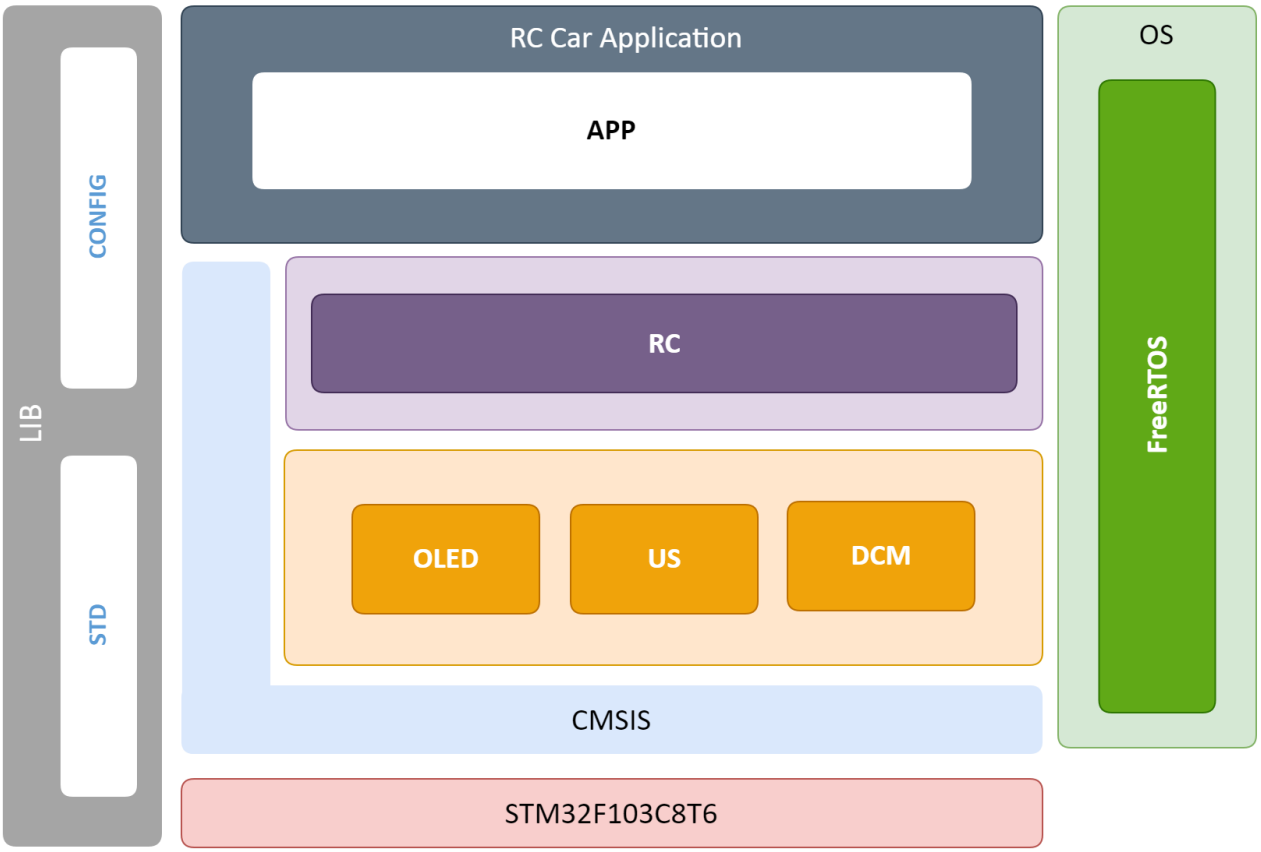


Figure 3. Layered Architecture Design

4.3.2. RC Circuit Schematic

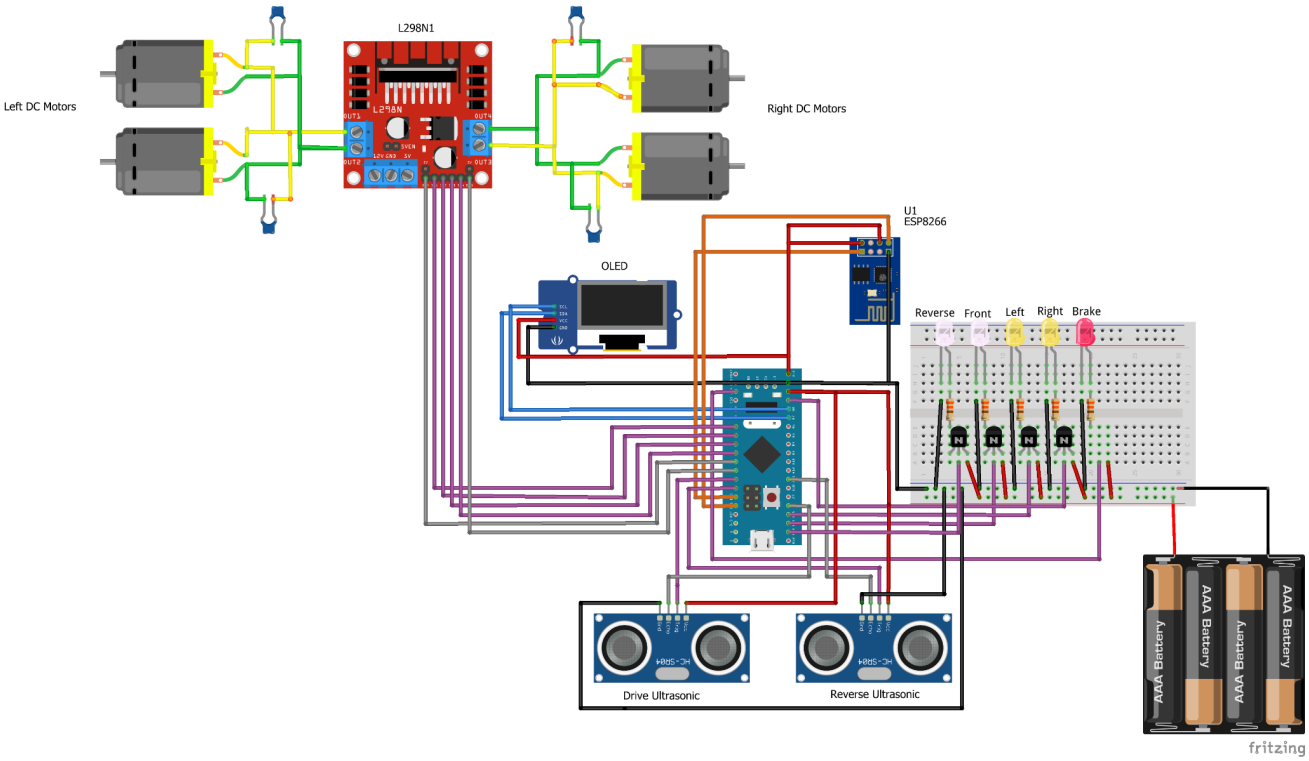


Figure 4. Project Circuit Schematic

Purple	**Output**
Grey	**Timer Channel**
Blue	**I2C**
Orange	**UART**
Red	**Vcc**
Black	**Ground**

4.3.3. RTOS Tasks

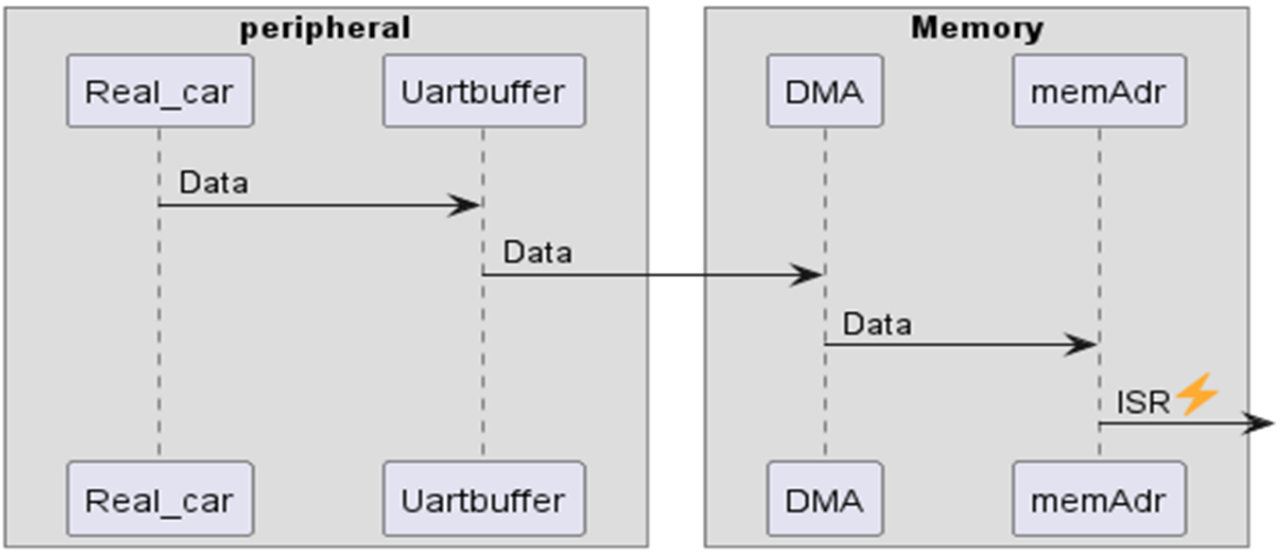
Task	Task Function Name	Priority	Stack Size Used	High Water Mark
UART / ESP	task_uart_processing	4	42	86
Lighting	LightingSystem	1	44	84
Blinking	Blinking	1	33	95
DC Motors	Task_DCM	3	42	86
Ultrasonic	Ultrasonic_Task	2	40	88
OLED	OLED_Function	1	124	4

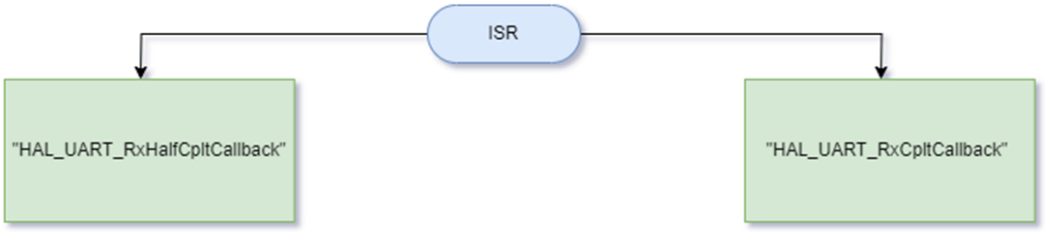
4.3.4. Processing Task

HOW DOES IT WORK?

DMA: Delivers the received data from UART Buffer to Memory.

ISR: Fired once the data is being transferred.





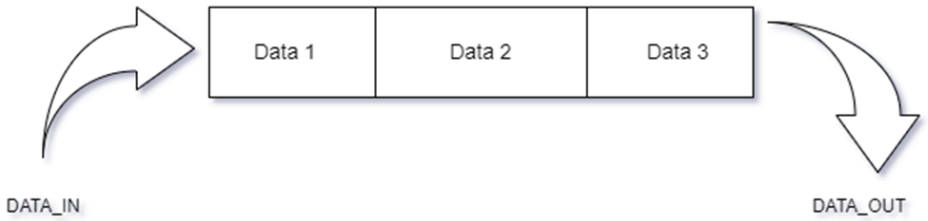
•The interrupt "HAL_UART_RxHalfCpltCallback" fires at half of the buffer size.

•The interrupt "HAL_UART_RxCpltCallback" fires at the end of the buffer size.

Why Using Queues?

•Preventing losing the data when receiving new data while processing previous one.

for example: if data "1" for instance is being processed and data "2" was received in the buffer and followed by data "3" while data "1" is still in the process, so data 2 will be lost, as data 3 will overwrite data 3 in the buffer.



4.4. Modules Description

4.4.1. OLED Module

The OLED module is a display device that uses organic light-emitting diodes to produce images. OLED displays are known for their high contrast, wide viewing angles, and thin form factor.

In the context of our project, the OLED module will be used to display information to the user, such as the current speed, gear position, and navigation (steering) directions.

4.4.2. US Module

The US module is an ultrasonic sensor that is used to measure distance. US sensors are known for their accuracy, low cost, and wide range of operating temperatures.

In the context of our project, the US module will be used to detect obstacles in the path of the vehicle front and back. This information will be used to avoid collisions and to safely navigate the vehicle.

4.4.3. DCM Module

The DCM module is a DC motor driver module that is used to control the speed and direction of a DC motor. DCM modules typically use PWM (pulse-width modulation) to control the motor speed. PWM is a technique where the motor is turned on and off very quickly. The percentage of time that the motor is turned on is called the duty cycle. The higher the duty cycle, the faster the motor will spin.
The DCM module is an essential component of the vehicle's control system, and it plays a key role in the vehicle's ability to safely and efficiently navigate its environment.

4.5. Drivers' Documentation (APIs)

4.5.1. Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

4.5.2. MCAL APIs

We are using STM32Cube CMSIS drivers and CubeMX to quickly configure what we need.

4.5.3. HAL APIs

4.5.3.1. DCM Module

4.5.3.1.1. DCM_MoveForward

Service Name	DCM_MoveForward	
Syntax	void DCM_MoveForward(st_dcm_config_t *motor)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_config	pointer to the motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the motor forward.	
Available via	DCM_interface.h	

4.5.3.1.2. DCM_MoveBackward

Service Name	DCM_MoveBackward	
Syntax	void DCM_MoveBackward (st_dcm_config_t *motor)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_config	pointer to the motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the motor backward.	
Available via	DCM_interface.h	

4.5.3.1.3. DCM_SetSpeed

Service Name	DCM_SetSpeed	
Syntax	void DCM_SetSpeed(st_dcm_config_t *motor_config, uint8_t speed)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_config	pointer to the motor configuration structure.
	speed	desired speed of the motor. This parameter can be one of the following values: ● SPEED_ZERO: speed is zero. ● SPEED_MIN: minimum speed. ● SPEED_MED: medium speed. ● SPEED_HIGH: high speed. ● SPEED_MAX: maximum speed.
Parameters (out)	None	
Return value	None	
Description	Sets the speed of the motor.	
Available via	DCM_interface.h	

4.5.3.1.4. DCM_Stop

Service Name	DCM_Stop	
Syntax	void DCM_Stop(st_dcm_config_t *motor_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_config	pointer to the motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Stops the motor	
Available via	DCM_interface.h	

4.5.4. ULTRASONIC Module

4.5.4.1. set_Ultrasonic_num

Service Name	set_Ultrasonic_num	
Syntax	void set_Ultrasonic_num(ULTRASONIC_NUM Num)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	ULTRASONIC_NUM Num	Enum to Number of ultrasonic system (Ultrasonic1 ,Ultrasonic2)
Parameters (out)	None	
Return value	None	
Description	Set the number of ultrasonic to select which ultrasonic will work.	
Available via	Ultrasonic_interface.h	

4.5.4.2. Ultrasonic_Updatedistance

Service Name	Ultrasonic_Updatedistance
Syntax	void Ultrasonic_Updatedistance(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	None
Return value	None
Description	Update Ultrasonic distance by triggering the next cycle.
Available via	Ultrasonic_interface.h

4.5.4.3. Ultrasonc_getstage

Service Name	Ultrasonc_getstage
Syntax	ULTRASONIC_STAGE Ultrasonc_getstage(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	ULTRASONIC_STAGE | Enum to the stage of operation(Half_way_operation ,Complete_operation)
Description	Get the stage of ultrasonic that indicates whether the timeout happened and the ultrasonic got stuck or it completes its operation
Available via	Ultrasonic_interface.h

4.5.4.4. Ultrasonc_getdistace

Service Name	Ultrasonc_getdistace
Syntax	uint16_t Ultrasonc_getdistace(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	None
Return value	uint16_t | return ultrasonic distance
Description	Return last saved ultrasonic distance from the last cycle
Available via	Ultrasonic_interface.h

4.5.4.5. HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)

Service Name	HAL_TIM_IC_CaptureCallback
Syntax	Void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)

Sync/Async	Asynchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	htim	Pointer to Timer cofiguration
Parameters (out)	None	
Return value	None	
Description	The system fires an interrupt when transmitting the first wave then stores the time of transmitting in variable ,changing the polarity to failing so the next interrupt fires when the wave comes back.Increment an Ultrasonic flag as an indicator for Number of entry times, Update stage to half way stage The second time the system will enter this function will store the time of the return wave, change polarity to the Rising again for the next cycle , update the stage to complete stage then calculate the difference between the 2 times and calculate the distance	
Available via	Ultrasonic_interface.h	

4.5.4.6. void Ultrasonic_Int_Timeout(ULTRASONIC_NUM Number_ultra)

Service Name	Ultrasonic_Int_Timeout
Syntax	void Ultrasonic_Int_Timeout(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	None
Parameters (out)	None
Return value	None
Description	This function get called when timeout period elapses and the stage is still half way that means the wave didn’t return during Max_Time out This function restart the system by returning the polarity to raising for the next cycle ,Returning the ultrasonic flag it’s initial value
Available via	Ultrasonic_interface.h

4.5.5. OLED Module

4.5.5.1. SSD1306_Init

Service Name	SSD1306_Init
Syntax	uint8_t SSD1306_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	None
Parameters (out)	None
Return value	Error Status 0: initialization unsuccessful 1: initialization successful
Description	Initialize the OLED
Available via	ssd1306.h

4.5.5.2. SSD1306_Clear

Service Name	SSD1306_Clear
Syntax	void SSD1306_Clear(void)
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return value	None
Description	Clears OLED display screen
Available via	ssd1306.h

4.5.5.3. SSD1306_GotoXY

Service Name	SSD1306_GotoXY	
Syntax	void SSD1306_GotoXY(uint16_t x, uint16_t y)	
Sync/Async	Synchronous	
Reentrancy	Non-reentrant	
Parameters (in)	X	X coordinate we want to go to
	Y	Y coordinate we want to go to
Parameters (out)	None	
Return value	None	
Description	Set cursor to a specific position	
Available via	ssd1306.h	

4.5.5.4. SSD1306_Puts

Service Name	SSD1306_Puts	
Syntax	char SSD1306_Puts(char* str, FontDef_t* Font, SSD1306_COLOR_t color)	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	str	String we want to display
	Font	Address of the required font
	color	Color of the text
Parameters (out)	None	
Return value	0: if no problem was found Any other value otherwise	
Description	Print a string on the OLED display screen	
Available via	ssd1306.h	

4.5.6. APP APIs

4.5.6.1. RC_MoveStraightForward

Service Name	RC_MoveStraightForward	
Syntax	void RC_MoveStraightForward(st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car straight in the forward direction.	
Available via	DCM_interface.h	

4.5.6.2. RC_MoveRightForward

Service Name	RC_MoveRightForward	
Syntax	void RC_MoveRightForward(st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the right in the forward direction.	
Available via	DCM_interface.h	

4.5.6.3. RC_MoveLeftForward

Service Name	RC_MoveLeftForward	
Syntax	void RC_MoveLeftForward(st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the left in the forward direction.	
Available via	DCM_interface.h	

4.5.6.4. RC_MoveRightSharpForward

Service Name	RC_MoveRightSharpForward	
Syntax	void RC_MoveRightSharpForward (st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the right sharply in the forward direction.	
Available via	DCM_interface.h	

4.5.6.5. RC_MoveLeftSharpForward

Service Name	RC_MoveLeftSharpForward	
Syntax	RC_MoveLeftSharpForward (st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the left sharply in the forward direction.	
Available via	DCM_interface.h	

4.5.6.6. RC_MoveStraightBackward

Service Name	RC_MoveStraightBackward	
Syntax	void RC_MoveStraightBackward(st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car straight in the backward direction.	
Available via	DCM_interface.h	

4.5.6.7. RC_MoveRightBackward

Service Name	RC_MoveRightBackward	
Syntax	voidRC_MoveRightBackward(st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the right in the backward direction.	
Available via	DCM_interface.h	

4.5.6.8. RC_MoveLeftBackward

Service Name	RC_MoveLeftBackward	
Syntax	void DCM_MoveLeftBackward (st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the left in the backward direction.	
Available via	DCM_interface.h	

4.5.6.9. RC_MoveRightSharpBackward

Service Name	RC_MoveRightSharpBackward	
Syntax	void RC_MoveRightSharpBackward (st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the right sharply in the backward direction.	
Available via	DCM_interface.h	

4.5.6.10. RC_MoveLeftSharpBackward

Service Name	RC_MoveLeftSharpBackward	
Syntax	void RC_MoveLeftSharpBackward (st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
Parameters (out)	None	
Return value	None	
Description	Moves the car to the left sharply in the backward direction.	
Available via	DCM_interface.h	

4.5.6.11. RC_SetSpeed

Service Name	RC_SetSpeed	
Syntax	void RC_SetSpeed(st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config, uint8_t speed)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.
	speed	desired speed of the car. This parameter can be one of the following values: ● SPEED_ZERO: speed is zero. ● SPEED_MIN: minimum speed. ● SPEED_MED: medium speed. ● SPEED_HIGH: high speed. ● SPEED_MAX: maximum speed.
Parameters (out)	None	
Return value	None	
Description	Sets the speed of the car.	
Available via	DCM_interface.h	

4.5.6.12. RC_Stop

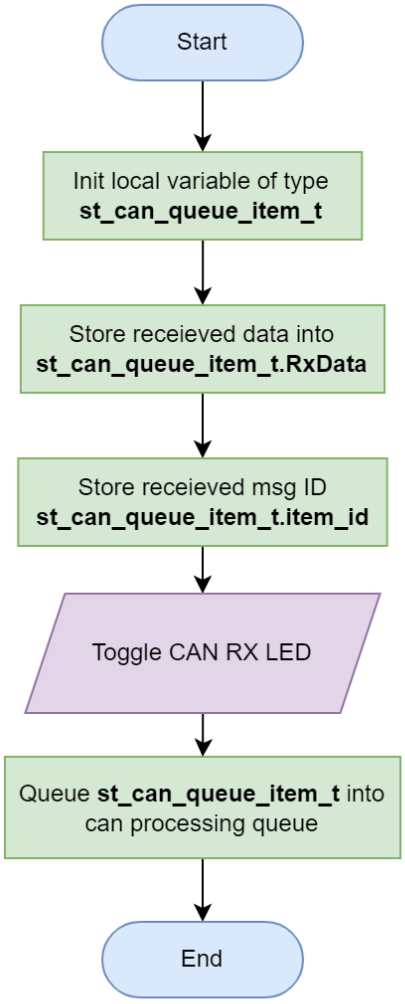
Service Name	RC_Stop	
Syntax	void RC_Stop (st_dcm_config_t *motor_left_config, st_dcm_config_t *motor_right_config)	
Sync/Async	Synchronous	
Reentrancy	Non-Reentrant	
Parameters (in)	motor_left_config	pointer to the left motor configuration structure.
	motor_right_config	pointer to the right motor configuration structure.

Parameters (out)	None
Return value	None
Description	Stops the car.
Available via	DCM_interface.h

5. Low Level Design

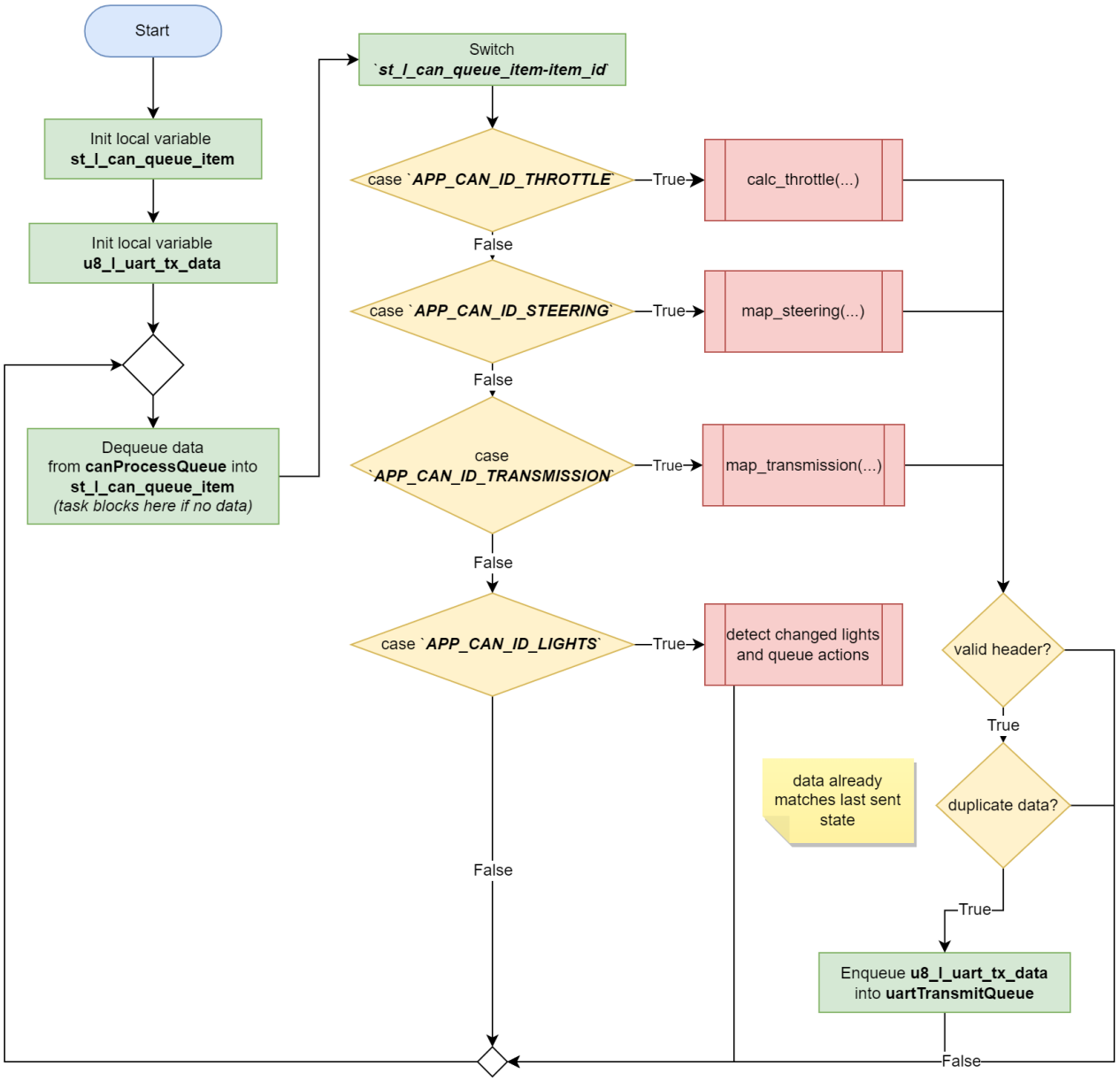
5.1. ECU 1 (Fiat Side)

5.1.1. CAN RX Interrupt

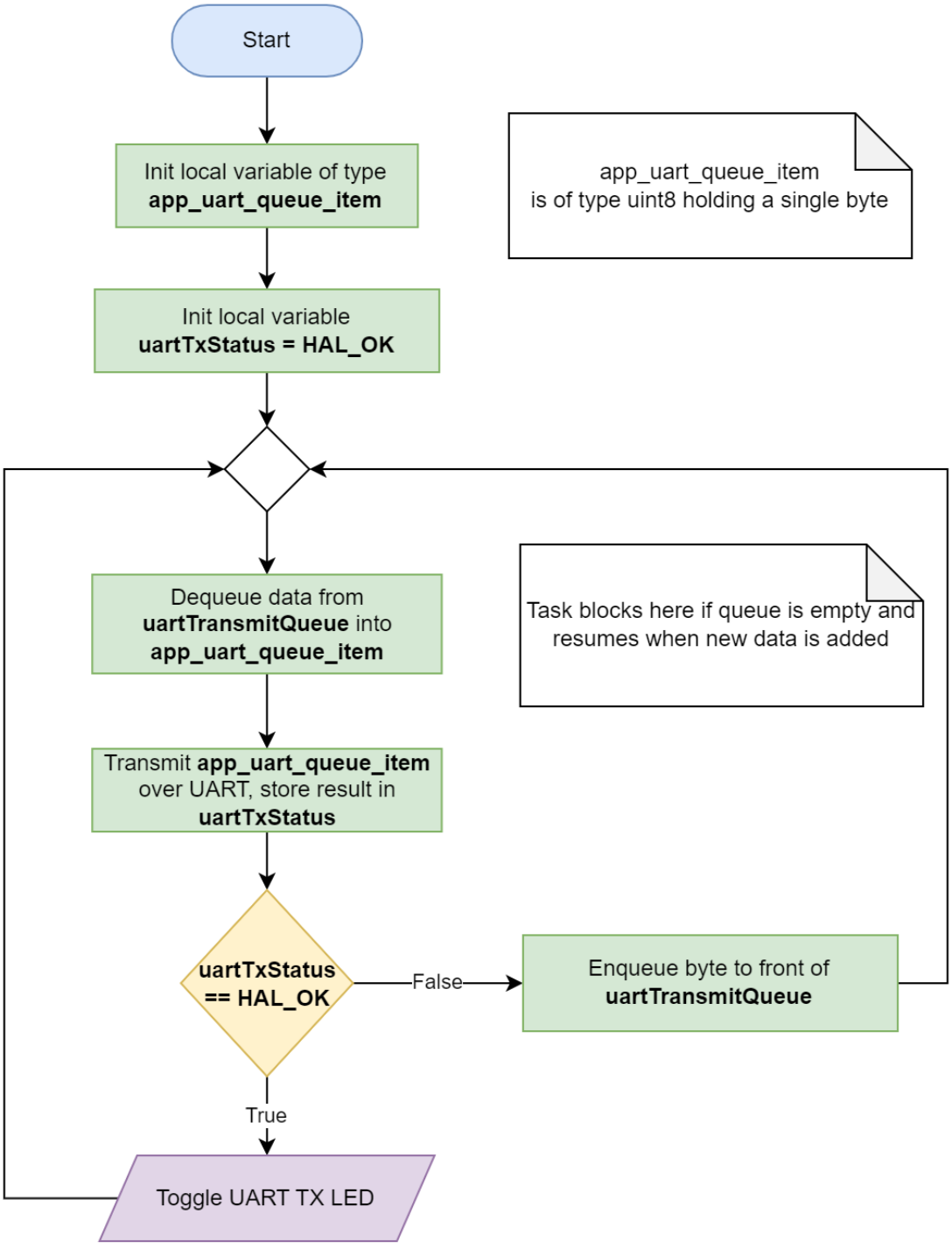



```
/* struct for CAN queued item to be processed */
typedef struct
{
    uint16_t          item_id             ;
    un_data_converter_t un_data_converter   ;
}st_can_queued_item_t;
```

5.1.2. CAN Processing Task



5.1.3. UART Processing Task



5.2. ECU 2 (RC Side)

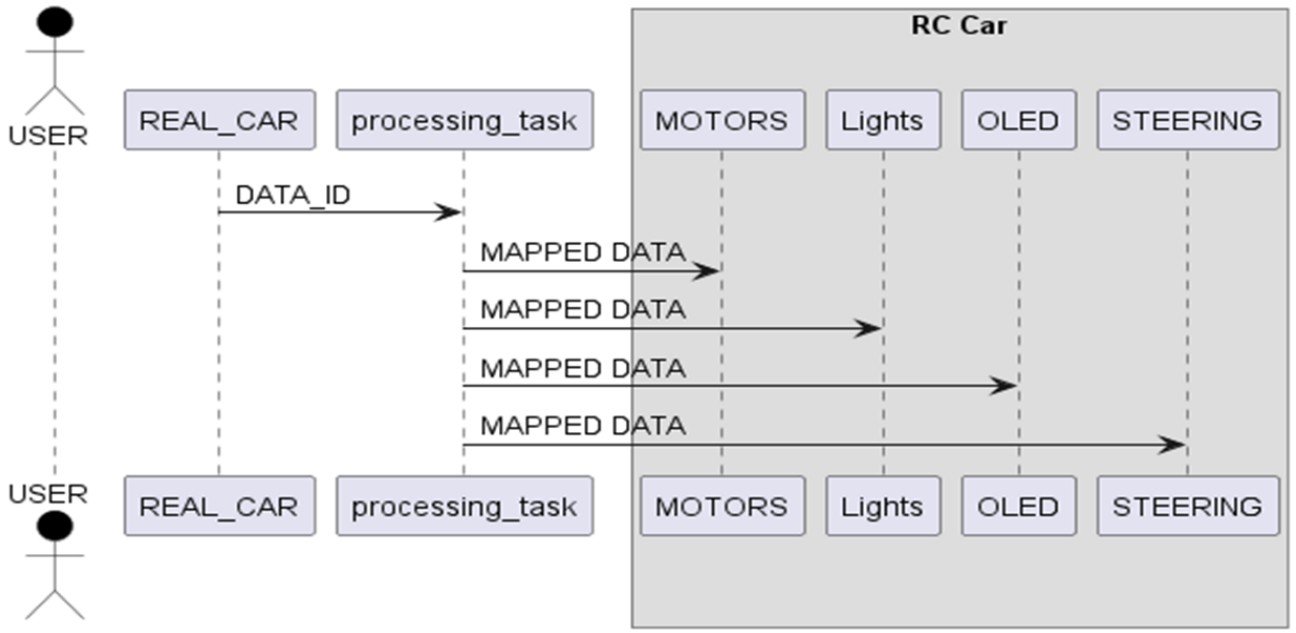
5.2.1. Receive interrupt

Processing Task

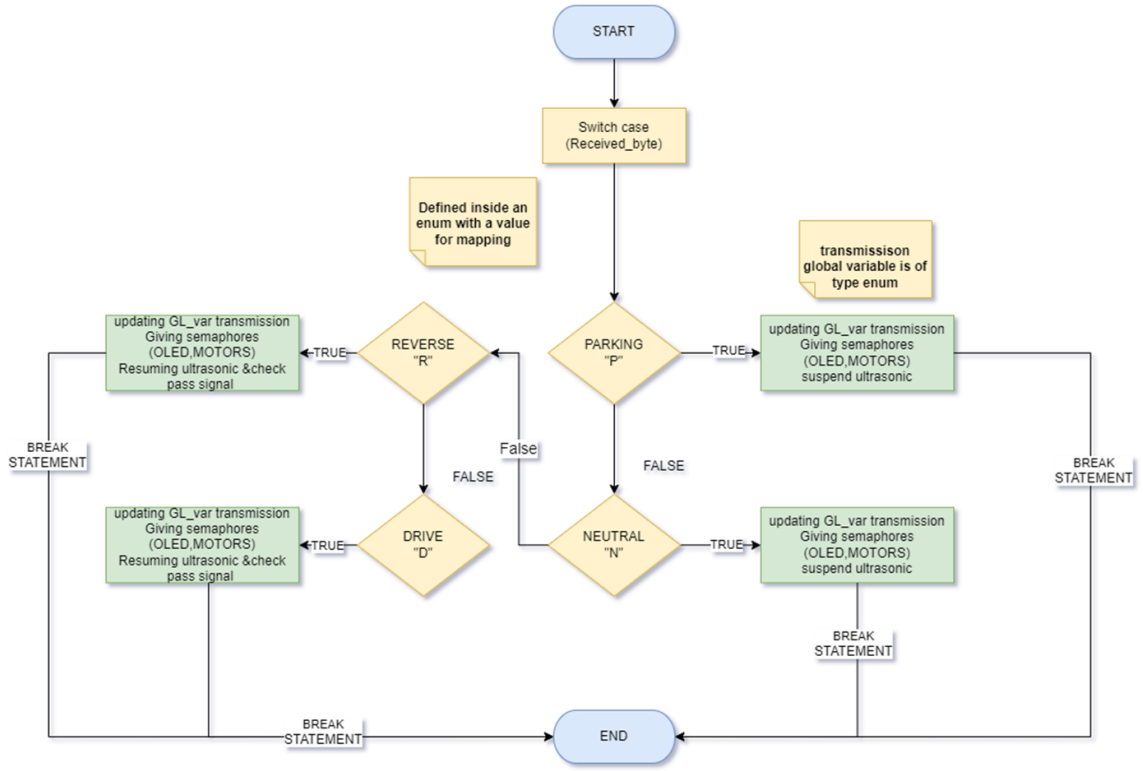
Responsible for mapping all the data coming from the real car through esp8266.

this task is responsible for mapping the data sent by the car and controlling other tasks by giving semaphores

needs to other task that will execute these data on real manner,such as motors, lights and sensors.

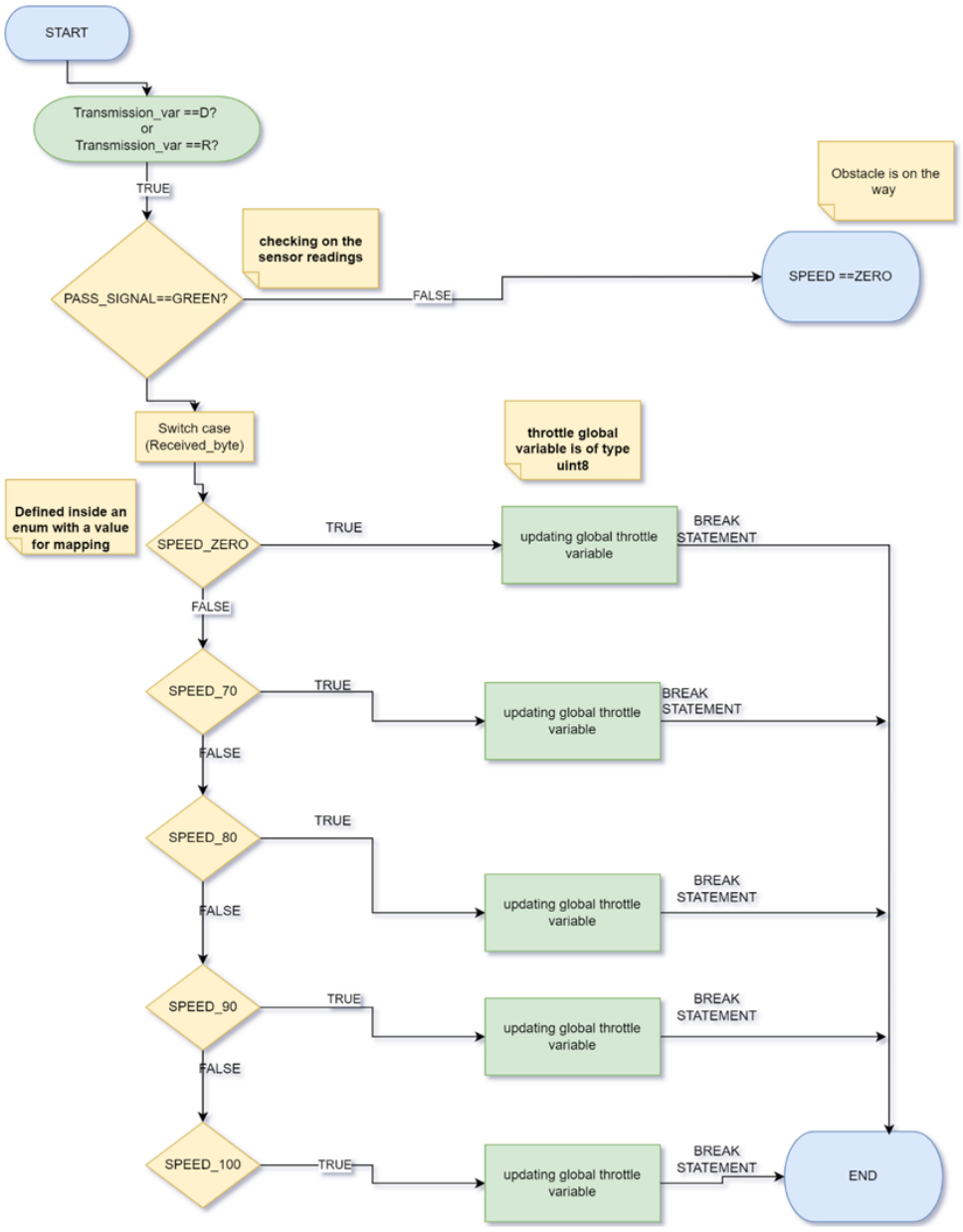


Processing Transmission



switch case on the states and updating it through a global variable that can be checked on later for an action,and giving semaphores for both the OLED MOTORS.
Suspension of the Ultrasonic sensors in all states except Drive and reverse states.
Resuming the Ultrasonic sensors task in Drive and Reverse state and making sure that pass signal coming from the Ultrasonic sensors is green so it indicates that it's clear to move.

Processing Throttle



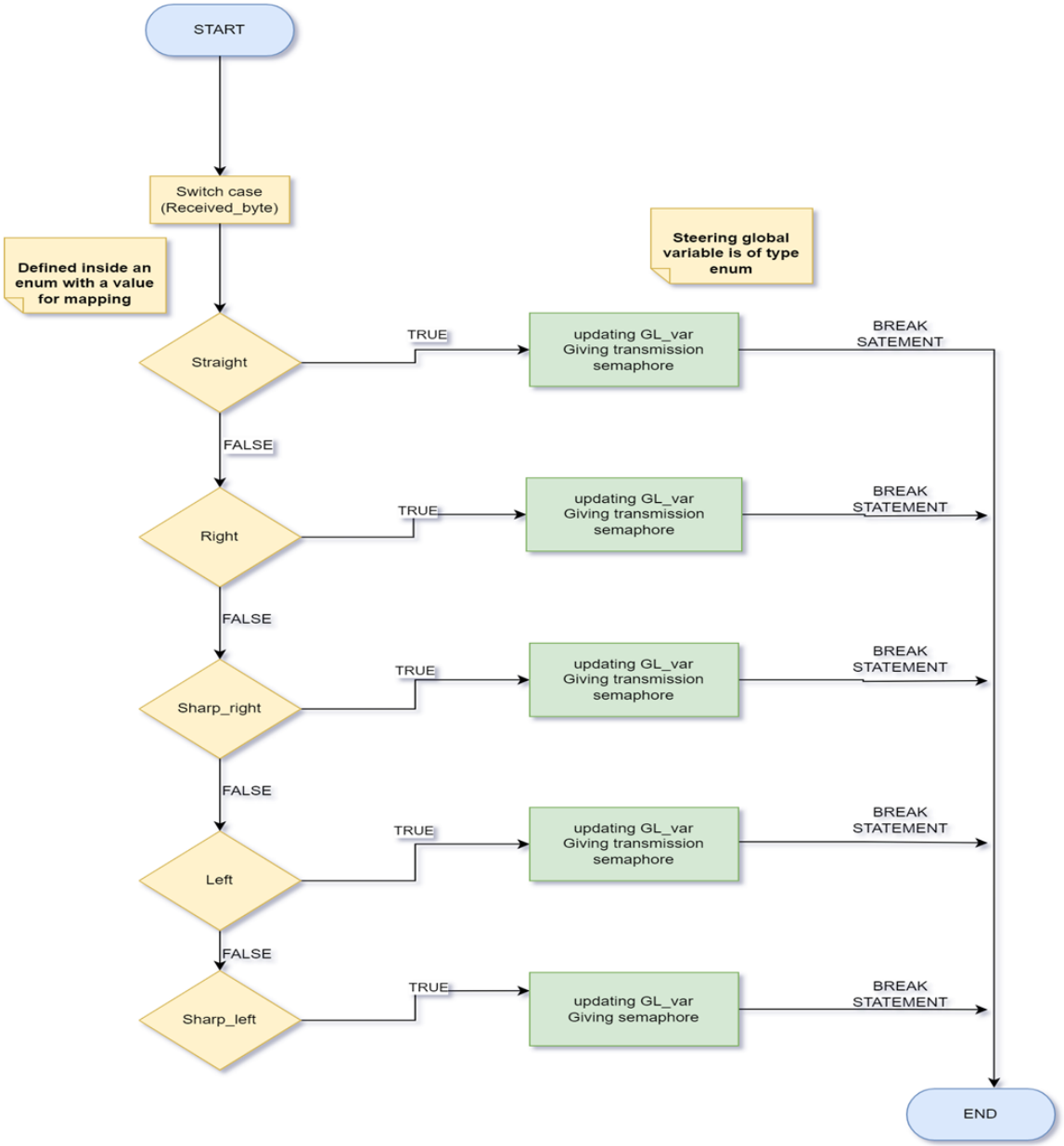
We have 5 speeds and they doesn't execute unless the green signal comes from the Ultrasonic sensors is green

throttle_0_percent = 0x10,
throttle_70_percent = 0x11,
throttle_80_percent = 0x12,
throttle_90_percent = 0x13,
throttle_100_percent = 0x14

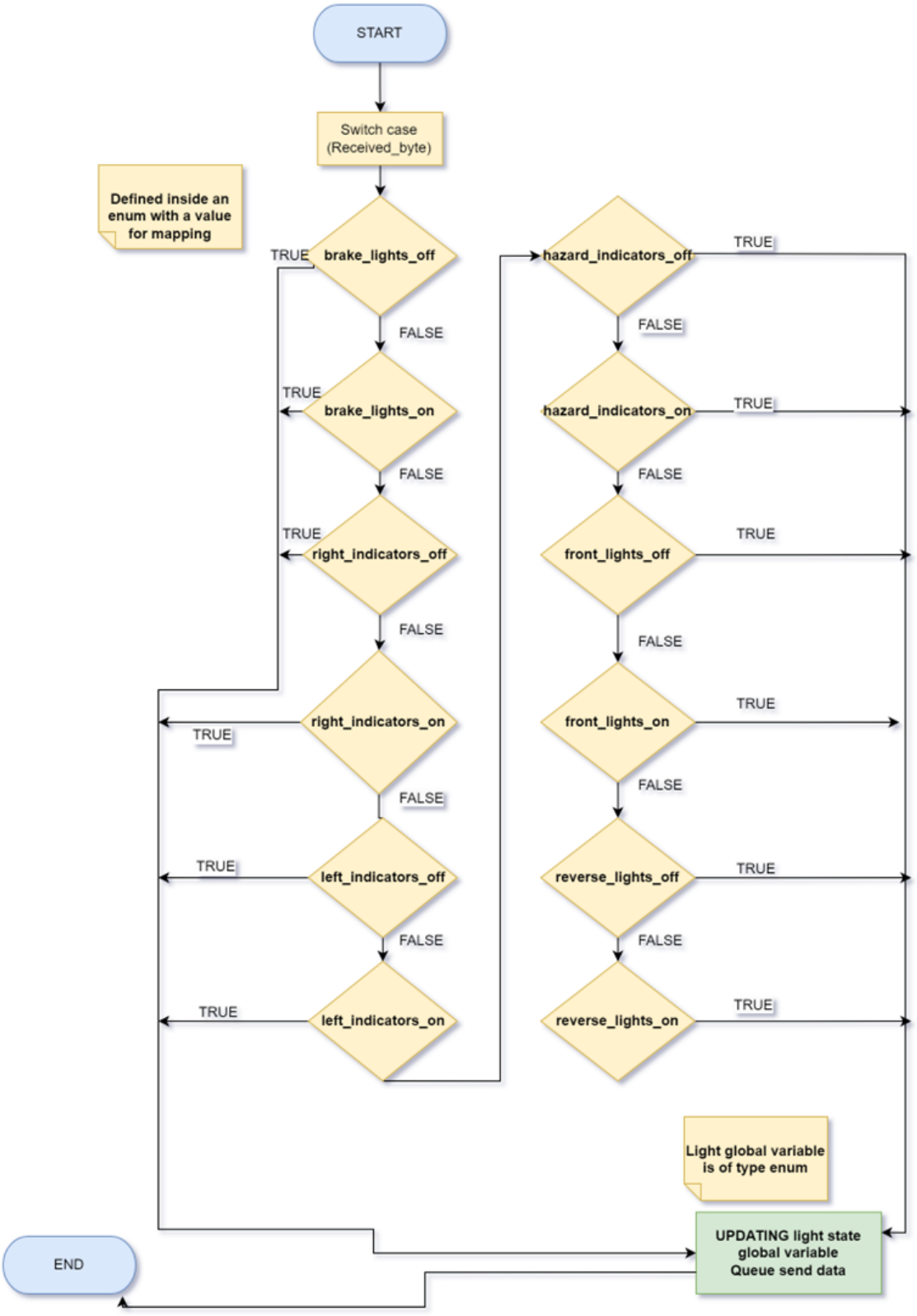
Those are the IDs for each speed of them.

Speed zero is the default speed when the signal comes from the Ultrasonic sensors is Red.

Processing Steering



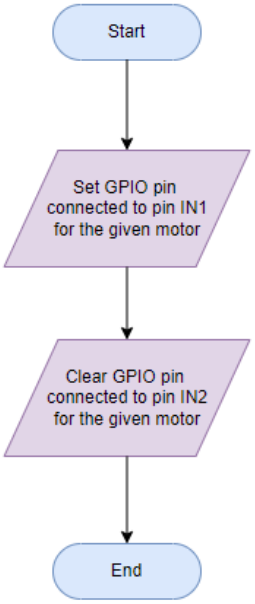
Processing Lights



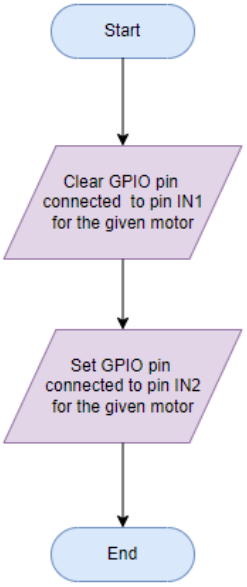
5.2.2. HAL Layer

5.2.2.1. DCM Module

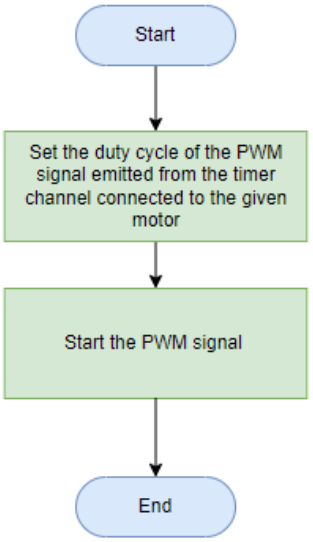
5.2.2.1.1. DCM_MoveForward



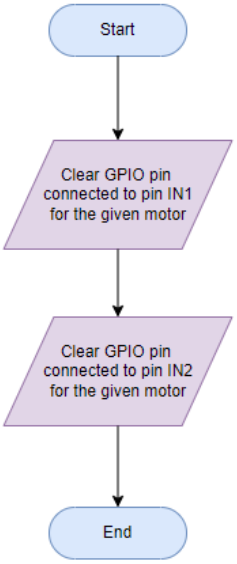
5.2.2.1.2. DCM_MoveBackward



5.2.2.1.3. DCM_SetSpeed

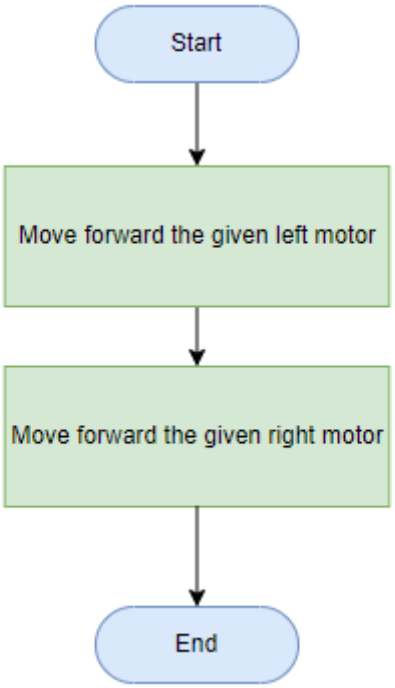


5.2.2.1.4. DCM_Stop

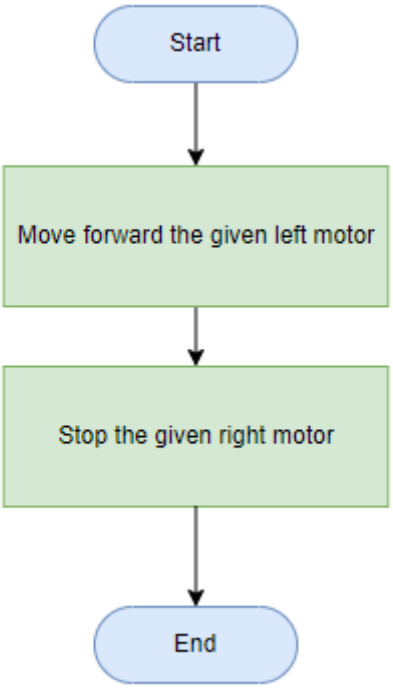


5.3. APP Layer

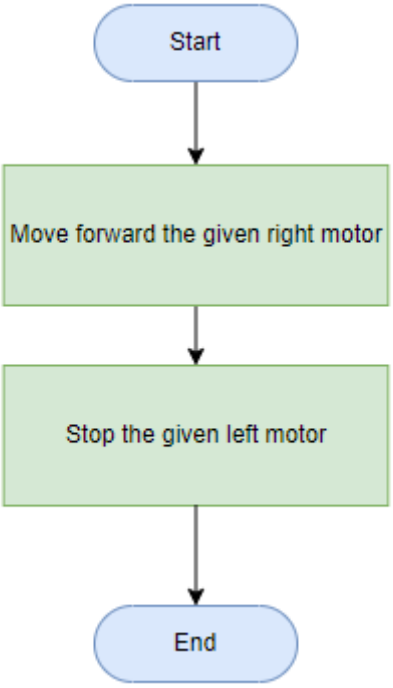
5.3.1. RC_MoveStraightForward



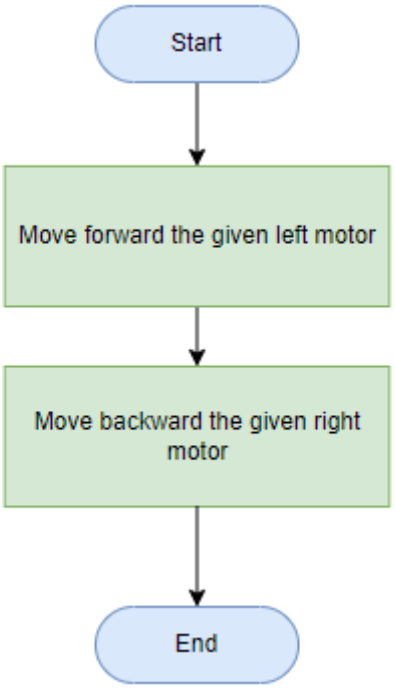
5.3.2. RC_MoveRightForward



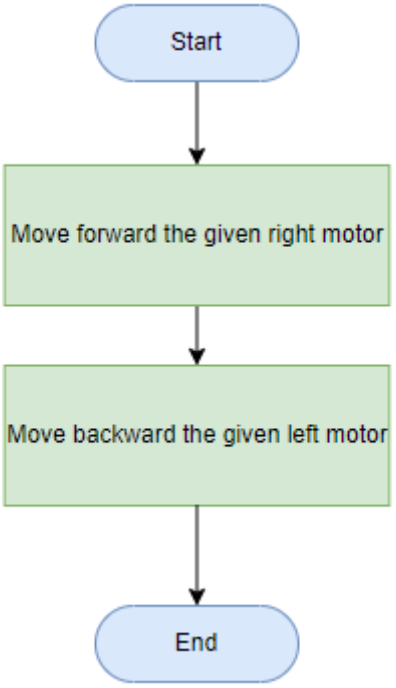
5.3.3. RC_MoveLeftForward



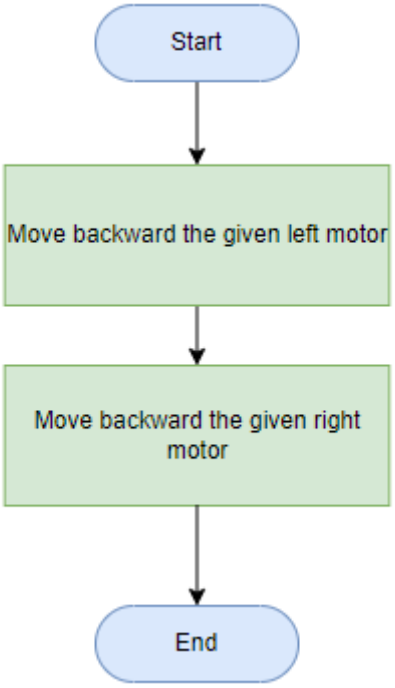
5.3.4. RC_MoveRightSharpForward



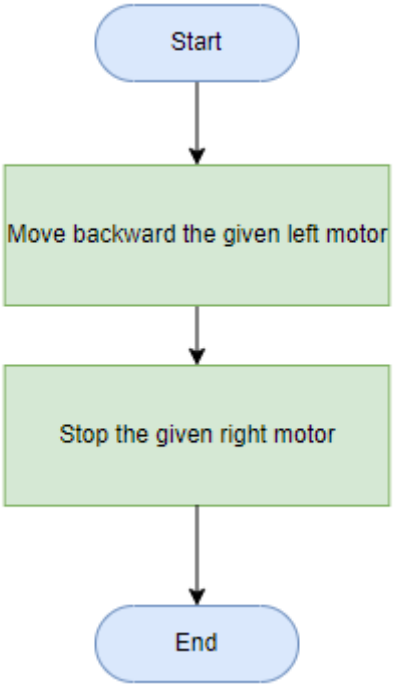
5.3.5. RC_MoveLeftSharpForward



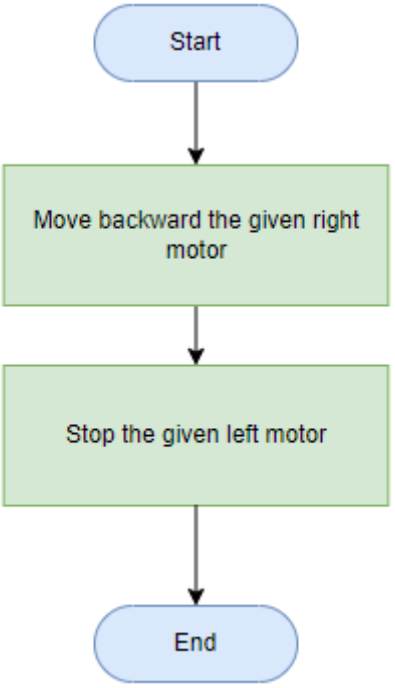
5.3.6. RC_MoveStraightBackward



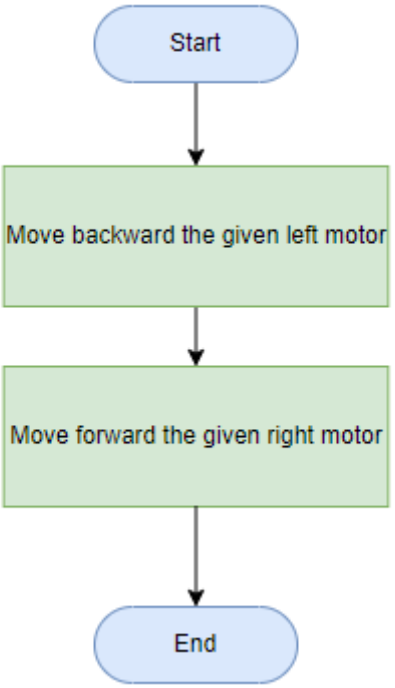
5.3.7. RC_MoveRightBackward



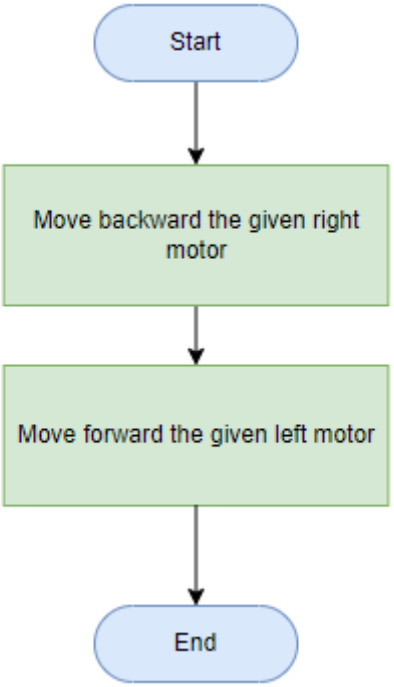
5.3.8. RC_MoveLeftBackward



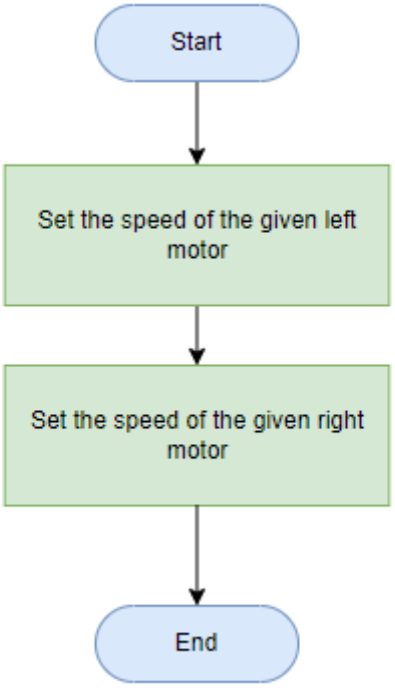
5.3.9. RC_MoveRightSharpBackward



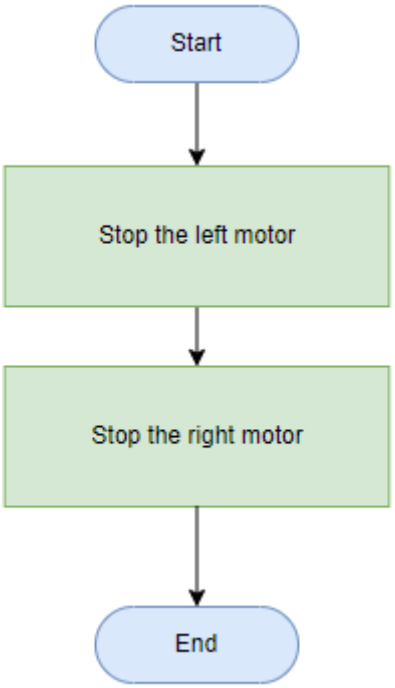
5.3.10. RC_MoveLeftSharpBackward



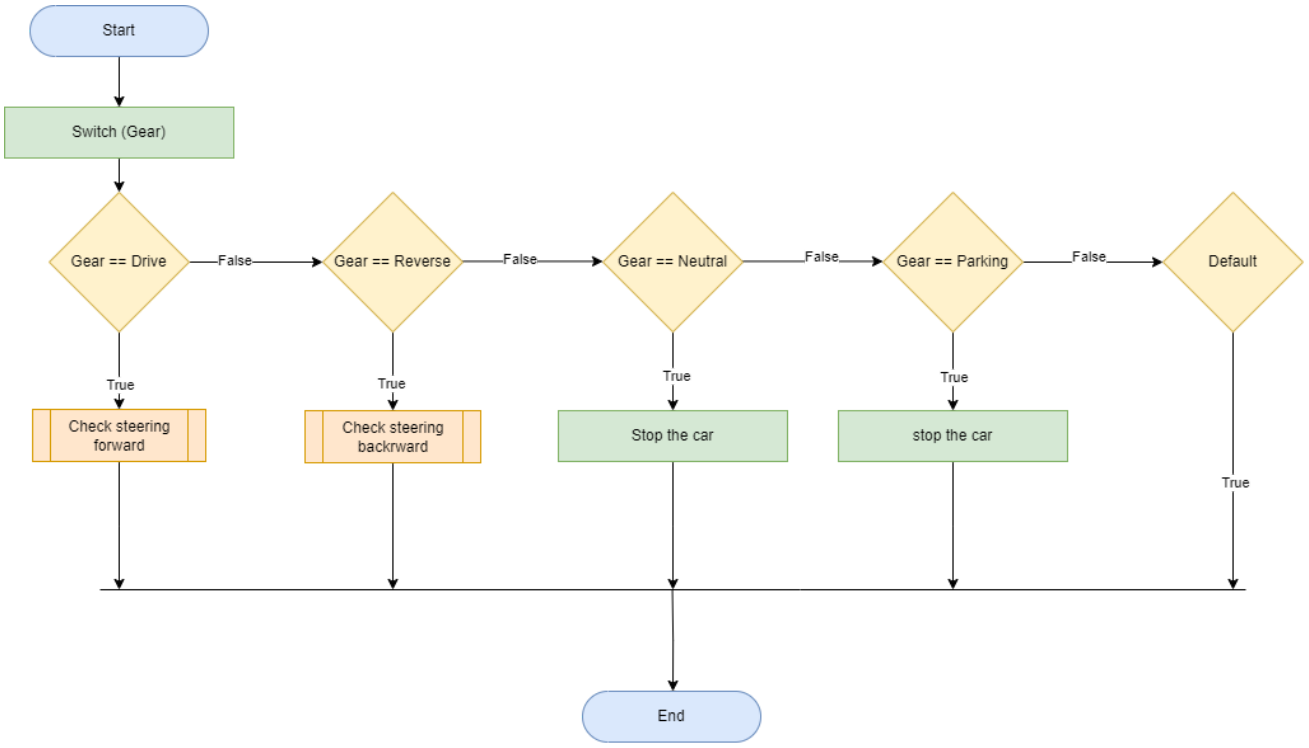
5.3.11. RC_SetSpeed

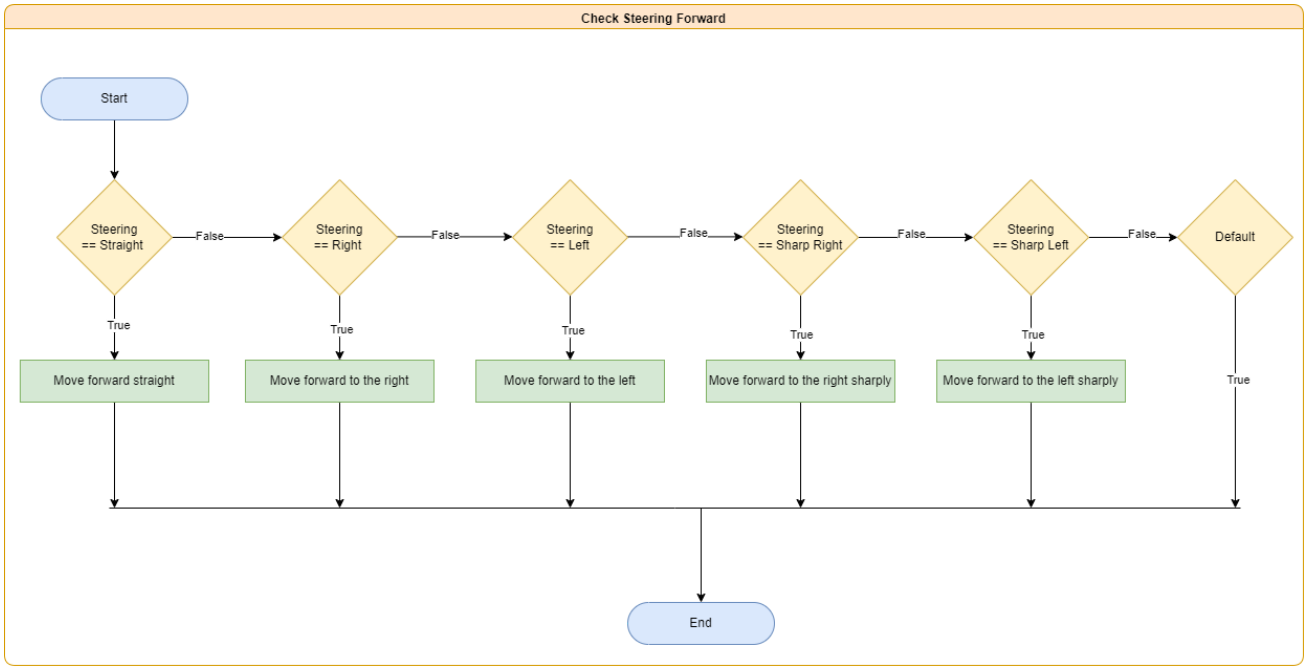


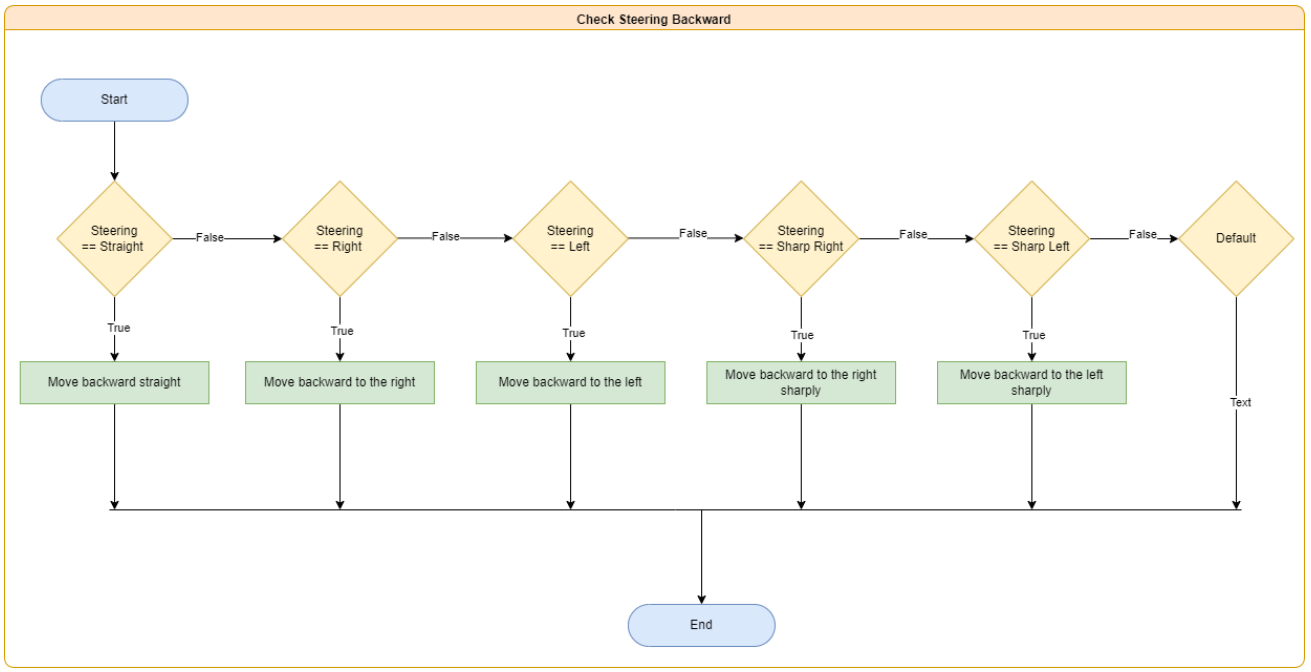
5.3.12. RC_Stop



5.3.13. Task_DCM

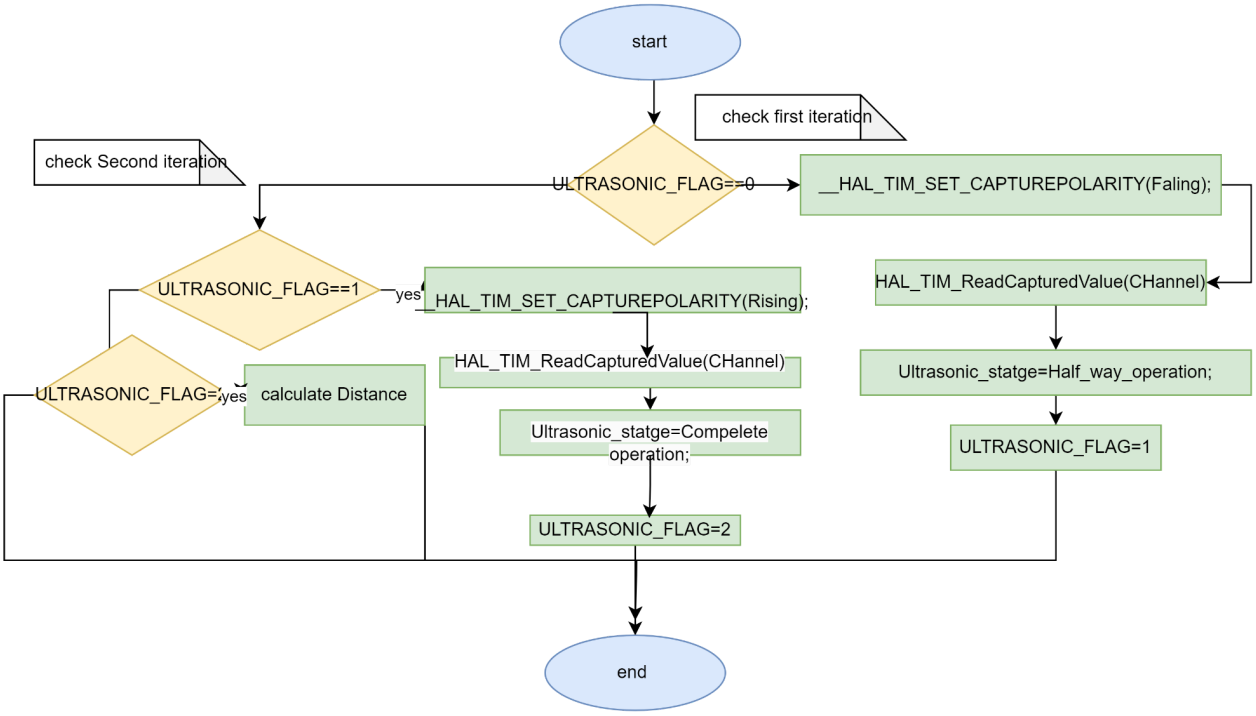




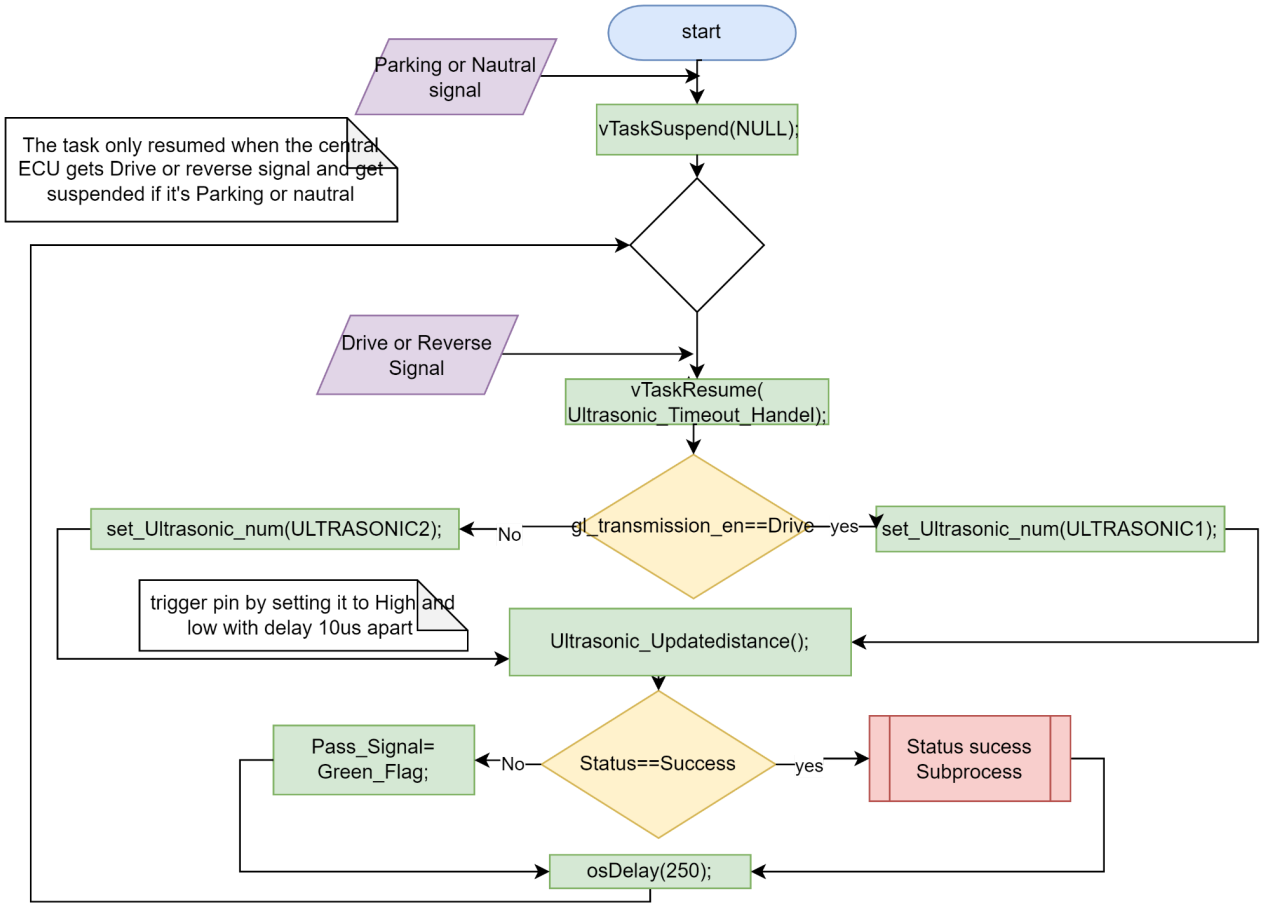

Check Steering Backward
Start
Steering == Straight
False
Steering == Right
False
Steering == Left
False
Steering == Sharp Right
False
Steering == Sharp Left
False
Default
True
True
True
True
True
Text
Move backward straight
Move backward to the right
Move backward to the left
Move backward to the right sharply
Move backward to the left sharply
End

5.3.14. Ultrasonic Module

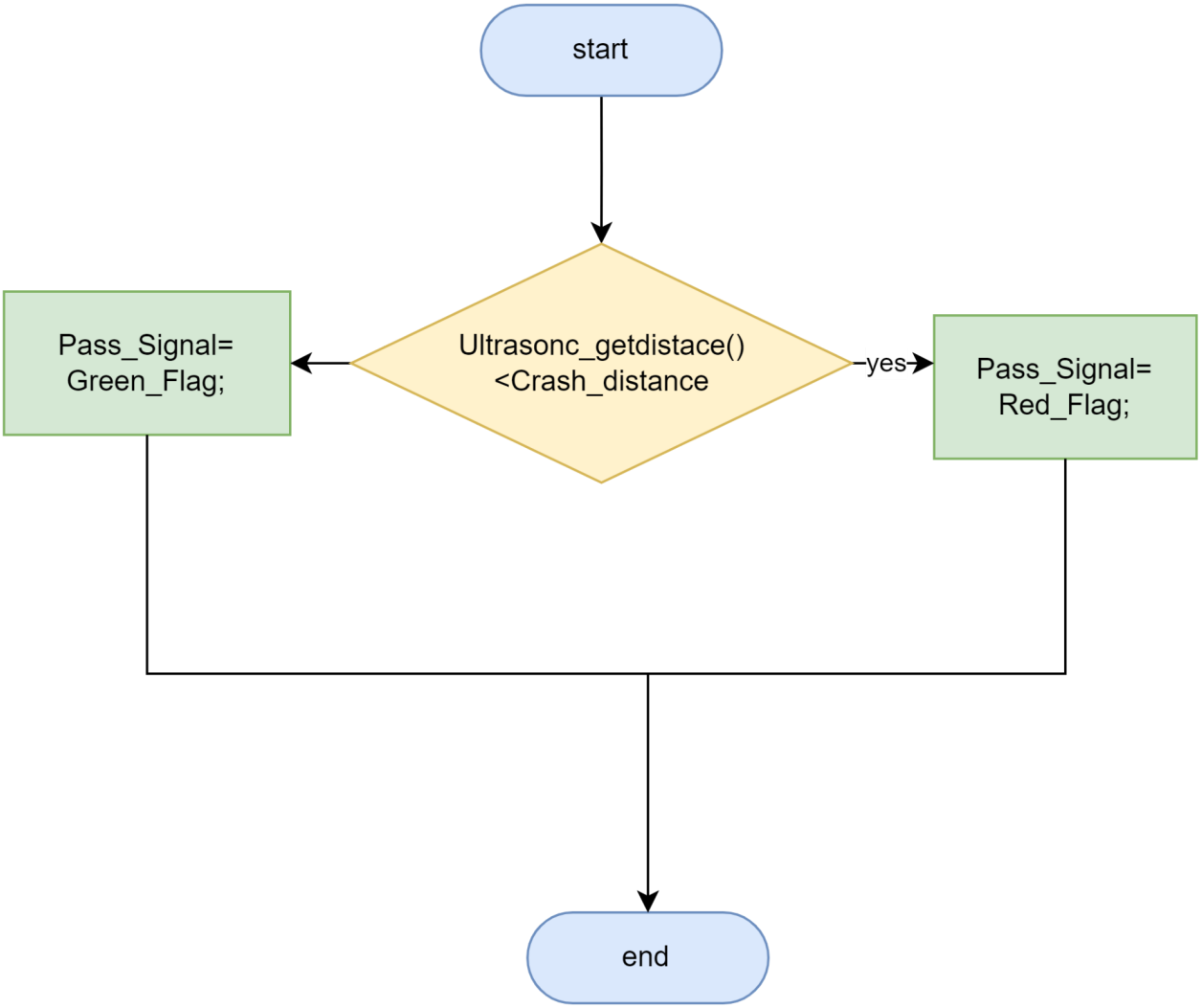
5.3.14.1. Timer ICU interrupt

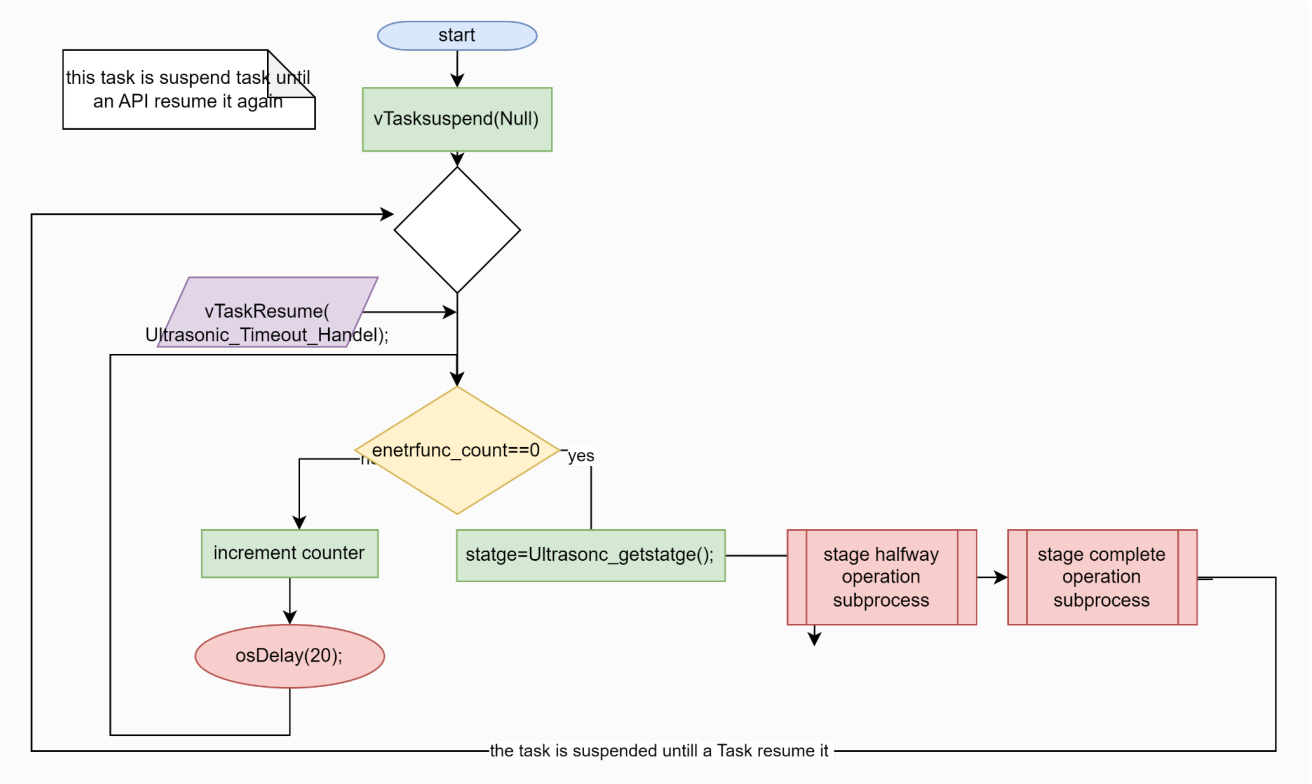


5.3.14.2. Ultrasonic Task

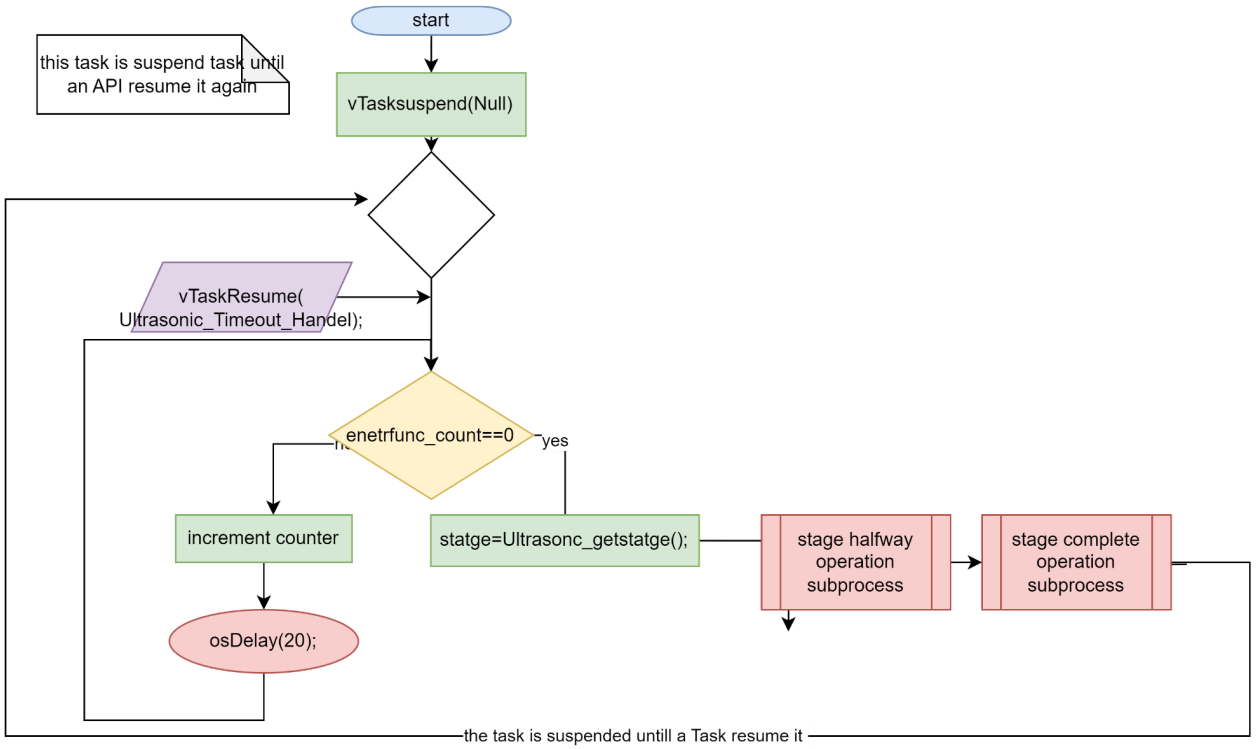


Status Processing

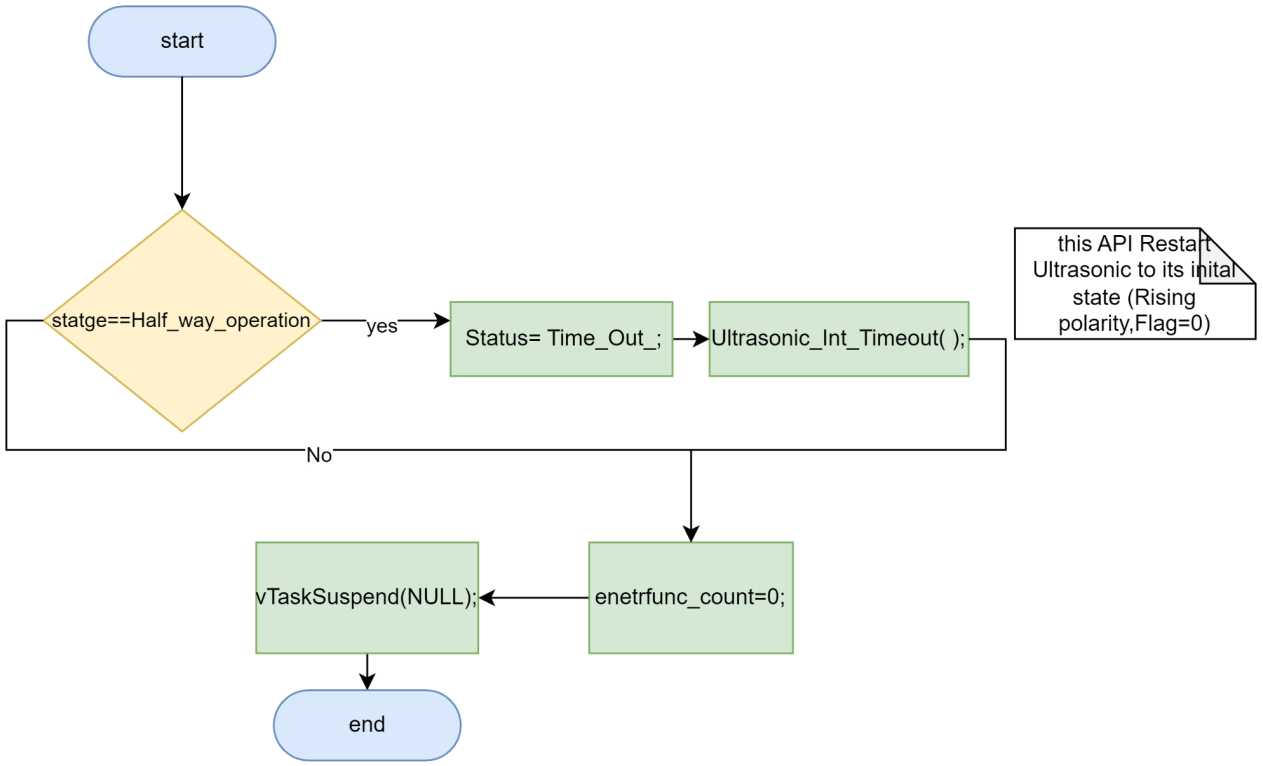



start
this task is suspend task until an API resume it again
vTasksuspend(Null)
vTaskResume(Ultrasonic_Timeout_Handel);
enetrfunc_count==0
yes
increment counter
statge=Ultrasonc_getstatge();
stage halfway operation subprocess
stage complete operation subprocess
osDelay(20);
the task is suspended untill a Task resume it

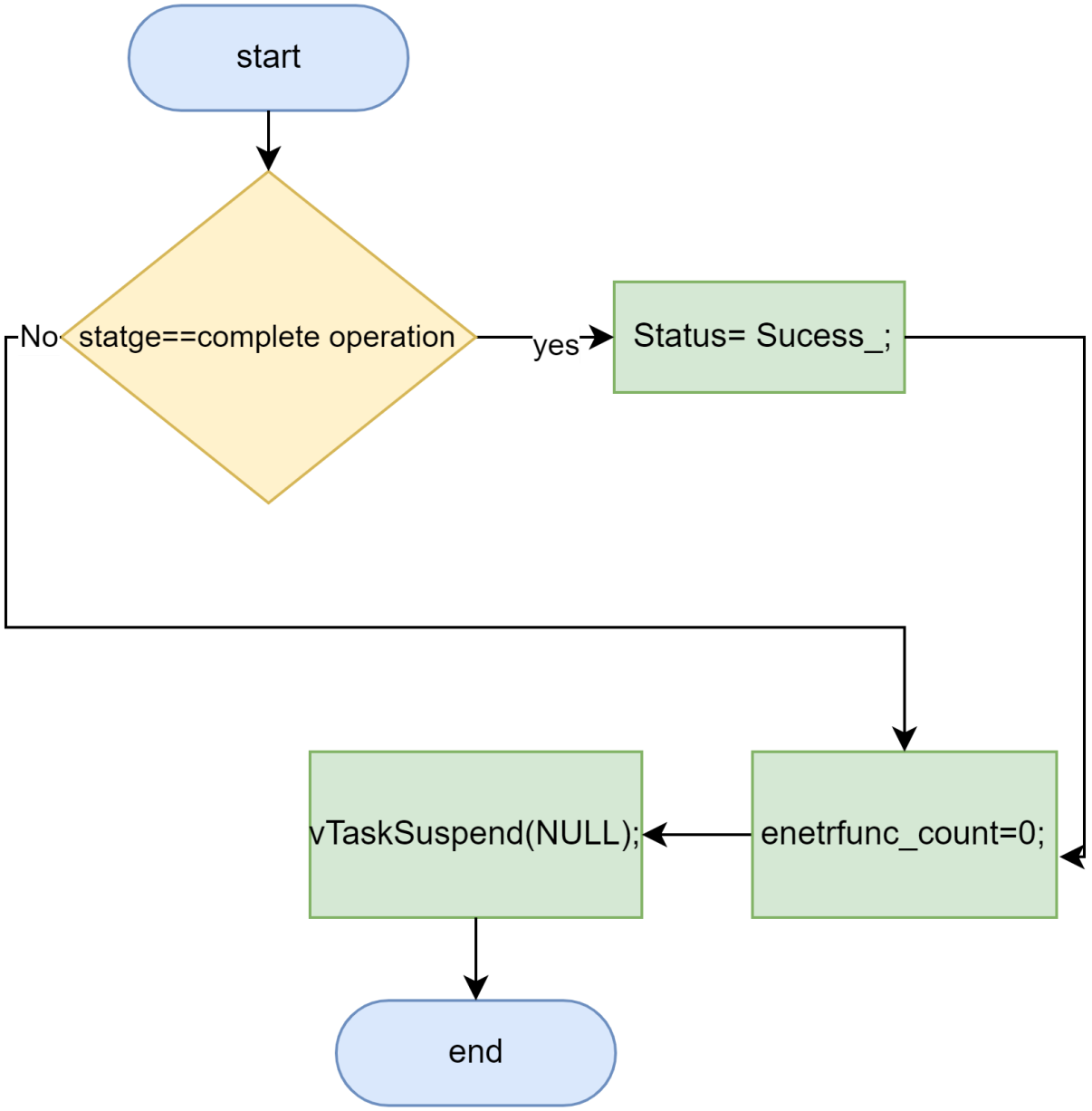
5.3.14.3. Ultrasonic Time out task



5.3.14.4. Stage halfway operation



5.3.14.5. Stage complete operation



6. Pre-configurations

6.1. ECU 1 (Fiat Side)

6.1.1. app_config.h

APP_CAN_ID_THROTTLE	0x1F0	The CAN ID for the throttle message.
APP_CAN_ID_LIGHTS	0x73E	The CAN ID for the lights message.
APP_CAN_ID_TRANSMISSION	0x2EF	The CAN ID for the transmission message.
APP_CAN_ID_STEERING	0x0DE	The CAN ID for the steering message.
APP_CAR_TRANSMISSION_PARK	0xFF	The CAN value for the transmission in park.
APP_CAR_TRANSMISSION_REVERSE	0x7F	The CAN value for the transmission in reverse.
APP_CAR_TRANSMISSION_NEUTRAL	0x0F	The CAN value for the transmission in neutral.
APP_CAR_TRANSMISSION_DRIVE	0x1F	The CAN value for the transmission in drive.
APP_CAR_STEERING_THRESHOLD_SHARP_LEFT_MAX	0x2C	The CAN value for the steering wheel at the maximum sharp left position.
APP_CAR_STEERING_THRESHOLD_SHARP_LEFT_MIN	0x27	The CAN value for the steering wheel at the minimum sharp left position.
APP_CAR_STEERING_THRESHOLD_LEFT_MAX	0x26	The CAN value for the steering wheel at the maximum left position.

APP_CAR_STEERING_THRESHOLD_LEFT_MIN	0x21	The CAN value for the steering wheel at the minimum left position.
APP_CAR_STEERING_THRESHOLD_STRAIGHT_MAX	0x20	The CAN value for the steering wheel at the maximum straight position.
APP_CAR_STEERING_THRESHOLD_STRAIGHT_MIN	0x19	CAN value for the steering wheel at the minimum straight position.
APP_CAR_STEERING_THRESHOLD_RIGHT_MAX	0x18	CAN value for the steering wheel at the maximum right position.
APP_CAR_STEERING_THRESHOLD_RIGHT_MIN	0x11	CAN value for the steering wheel at the minimum right position.
APP_CAR_STEERING_THRESHOLD_SHARP_RIGHT_MAX	0x10	CAN value for the steering wheel at the maximum sharp right position.
APP_CAR_STEERING_THRESHOLD_SHARP_RIGHT_MIN	0x09	CAN value for the steering wheel at the minimum sharp right position.
APP_ESP_HEADER_TRANSMISSION	0x07	ESP frame header for the transmission message.
APP_ESP_HEADER_LIGHT_BRAKES	0x08	ESP frame header for the lights and brakes message.
APP_ESP_HEADER_THROTTLE	0x01	ESP frame header for the throttle message.
APP_ESP_HEADER_LIGHT_R_INDICATORS	0x09	ESP frame header for the right indicator light message.
APP_ESP_HEADER_LIGHT_L_INDICATORS	0x0A	ESP frame header for the left indicator light message.

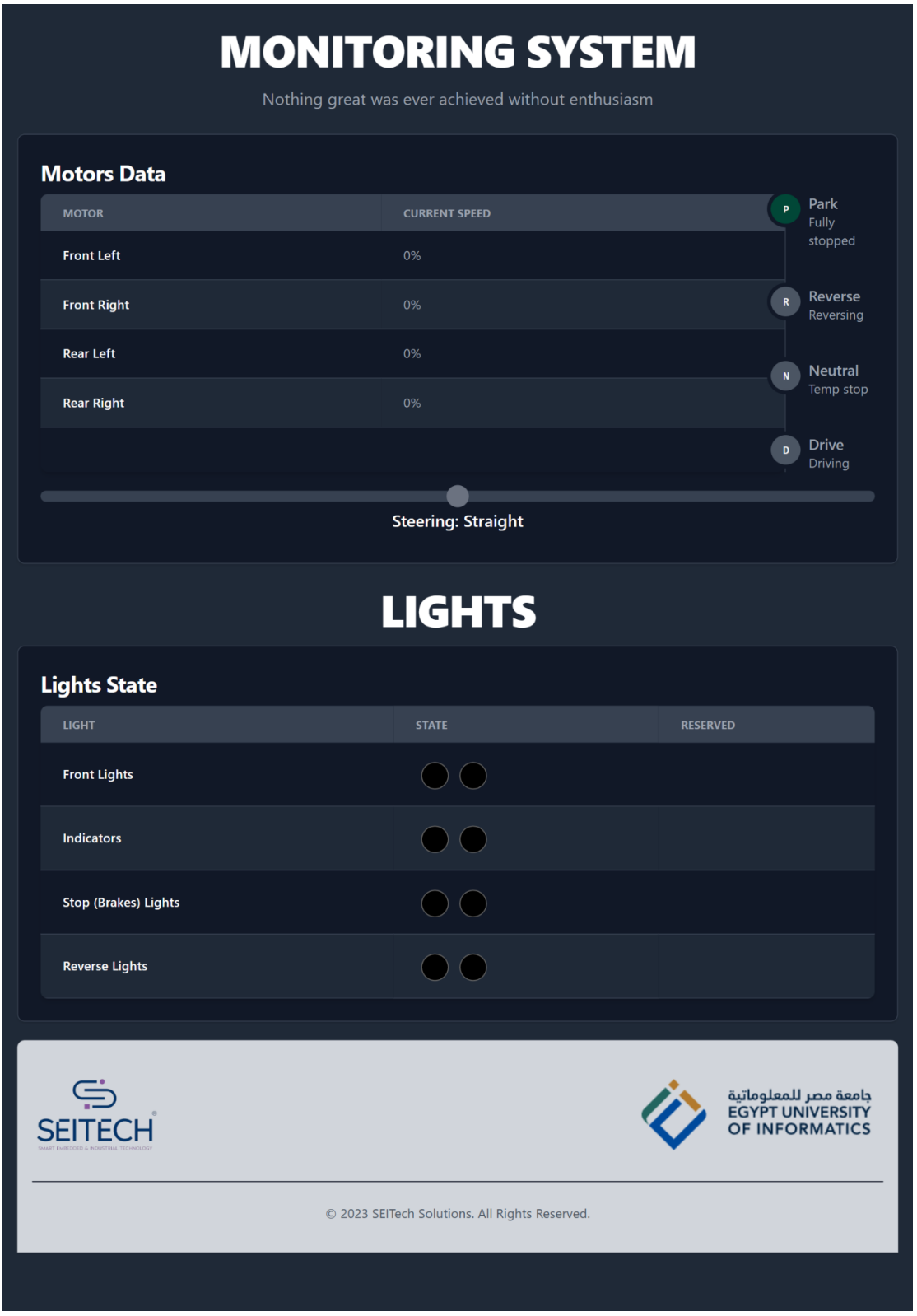
APP_ESP_HEADER_LIGHT_FRONT	0x0C	ESP frame header for the front light message.
APP_ESP_HEADER_LIGHT_REVERSE	0x0D	ESP frame header for the reverse light message.
APP_ESP_HEADER_STEERING	0x0E	ESP frame header for the steering message.
APP_ESP_DATA_STEERING_NONE	0xFF	ESP frame data for the steering wheel in the neutral position.
APP_ESP_DATA_STEERING_STRAIGHT	0x00	ESP frame data for the steering wheel in the straight position.
APP_ESP_DATA_STEERING_SHARP_LEFT	0x0C	ESP frame data for the steering wheel in the sharp left position.
APP_ESP_DATA_STEERING_LEFT	0x08	ESP frame data for the steering wheel in the left position.
APP_ESP_DATA_STEERING_SHARP_RIGHT	0x03	ESP frame data for the steering wheel in the sharp right position.
APP_ESP_DATA_STEERING_RIGHT	0x02	ESP frame data for the steering wheel in the right position.
APP_ESP_DATA_THROTTLE_STOP_MIN	0x0000	Min range for the car throttle reading mapping to stopping the rc car.
APP_ESP_DATA_THROTTLE_STOP_MAX	0x000A	Max range for the car throttle reading mapping to stopping the rc car.
APP_ESP_DATA_THROTTLE_LEVEL_1_MIN	0x000B	Min range for the car throttle reading mapping to the first rc car speed level.

APP_ESP_DATA_THROTTLE_LEVEL_1_MAX	0x03EB	Max range for the car throttle reading mapping to the first rc car speed level.
APP_ESP_DATA_THROTTLE_LEVEL_2_MIN	0x03EC	Min range for the car throttle reading mapping to the second rc car speed level.
APP_ESP_DATA_THROTTLE_LEVEL_2_MAX	0x0FA0	Max range for the car throttle reading mapping to the second rc car speed level.
APP_ESP_DATA_THROTTLE_LEVEL_3_MIN	0x0FA1	Min range for the car throttle reading mapping to the third rc car speed level.
APP_ESP_DATA_THROTTLE_LEVEL_3_MAX	0x2EE1	Max range for the car throttle reading mapping to the third rc car speed level.
APP_ESP_DATA_THROTTLE_LEVEL_4_MIN	0x2EE2	Min range for the car throttle reading mapping to the fourth rc car speed level.
APP_ESP_DATA_THROTTLE_LEVEL_4_MAX	0x401F	Max range for the car throttle reading mapping to the fourth rc car speed level.
APP_ESP_DATA_THROTTLE_STOP_MAP_VAL	0x00	ESP frame data map value for the throttle in the stop position.
APP_ESP_DATA_THROTTLE_LEVEL_1_MAP_VAL	0x01	ESP frame data map value for the throttle at the first level.
APP_ESP_DATA_THROTTLE_LEVEL_2_MAP_VAL	0x02	ESP frame data map value for the throttle at the second level.
APP_ESP_DATA_THROTTLE_LEVEL_3_MAP_VAL	0x03	ESP frame data map value for the throttle at the third level.
APP_ESP_DATA_THROTTLE_LEVEL_4_MAP_VAL	0x04	ESP frame data map value for the throttle at the fourth level.
APP_CAN_PROCESS_QUEUE_LENGTH	50	Length of the CAN process queue.

APP_UART_TX_QUEUE_LENGTH	50	Length of the UART transmit queue.
APP_UART_TX_TIMEOUT_MS	500	UART transmit timeout in milliseconds.
APP_RTOS_TASK_STACK_SIZE	200	RTOS tasks stack size in bytes.
THROTTLE_READING_REDUCTION_FACTOR	250 * 64	Throttle reading reduction factor.

7. Screenshots

7.1. Web server GUI



8. Issues and Challenges

8.1. Undefined behavior when motors are running with speed over 80% or their direction is changed suddenly

- This was clearly a hardware issue, at first we thought there isn't enough amperage to support all the motors on full speed however after measuring the motors max current and using 2A DC adapter the issue still persisted.
- Solution: purchasing 100nF capacitors and soldering one on each motor as a bypass capacitance dealt with the reverse-conducting current and overcame the issue with energy flowing back in the circuit due to the sudden reverse in polarity of the motors and supporting high current being drawn by them.

9. Future Enhancements

- Engage brakes automatically if the car is drifting (on neutral) towards an obstacle.
- Automatic corner lights with steering.
- Replace ESP8266 with ESP32 to be able to add security features such as encrypting messages being transmitted.

10. Links

1. GitHub Repository
   https://github.com/HossamElwahsh/CAN-ESP-Car-Clone-Emulating-Real-Car-Actions
2. Notion - Contains details on mapped data and system HL diagrams
   https://cultured-chauffeur-242.notion.site/CAN-ESP-Car-Clone-Emulating-Real-Car-Actions-81b1da8b31a24f9394ab7fe4c9a54690

11. References

1. The car hacker's handbook : a guide for the penetration tester : Smith, Craig (Reverse engineer)
   https://archive.org/details/thecarhackershandbook
2. Stack Size of FreeRTOS tasks
   https://www.freertos.org/uxTaskGetStackHighWaterMark.html
3. OLED Driver
   https://controllerstech.com/oled-display-using-i2c-stm32/
4. ESP-NOW with ESP8266
   https://randomnerdtutorials.com/esp-now-one-to-many-esp8266-nodemcu/
5. ESP-NOW Web Server
   https://randomnerdtutorials.com/esp8266-esp-now-wi-fi-web-server/