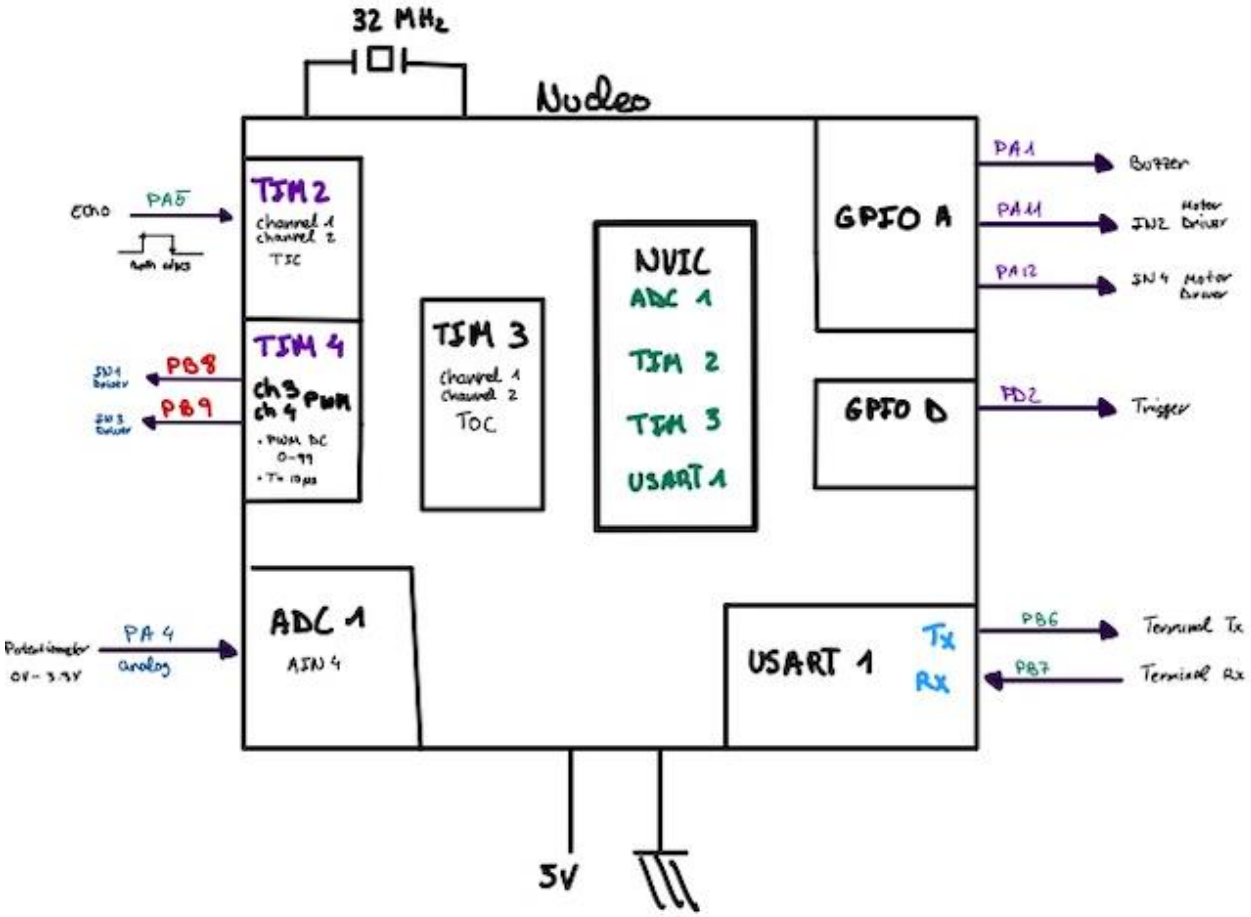
Telecommunications Engineering
Universidad Carlos III de Madrid
Microprocessor based digital systems.

Project Report Laboratory 2022/23

Group	Students(Lab_6x_08)	Signatures
65	Javier Rojas García-Rostan Lucia Rodríguez Treviño Javier Millán Ortiz	

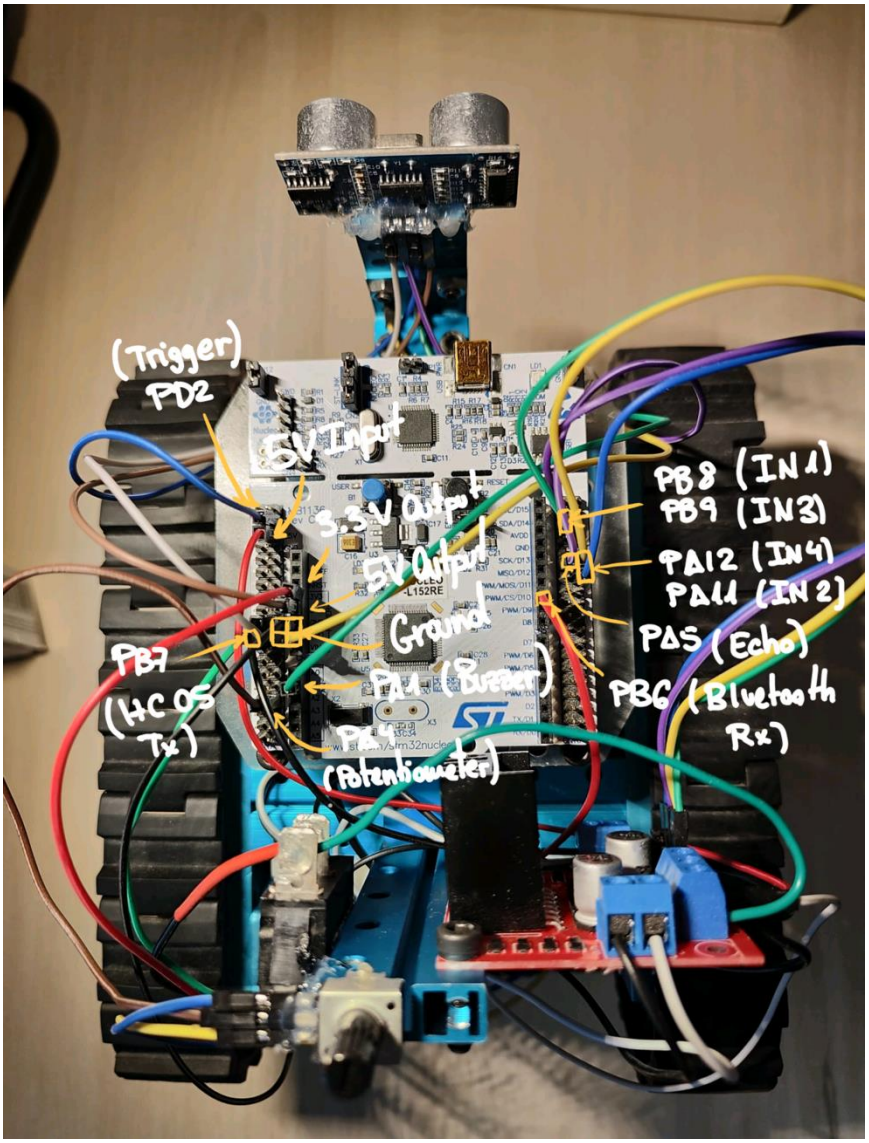
1.Block Diagram


32 MHz
Nucleo
TIM 2
channel 1
channel 2
TIC
TIM 4
ch3 PWM
ch4 PWM
TIM 3
channel 1
channel 2
TOC
NVIC
ADC 1
TIM 2
TIM 3
USART 1
GPIO A
GPIO B
ADC 1
AIN 4
USART 1
Tx
Rx
Echo
PA5
PB8
PB9
PA 4
analog
PA1
Buzzer
PA11
PA12
PB2
Trigger
PB6
Terminal Tx
PB7
Terminal Rx
5V

2. Schematics and description of connections, showing the board used, and the components.

We used the NUCLEO-L152RE Board. The main benefit of this board is that it does not have an LCD, so there are many more available ports without changing any configuration. Moreover, we have USART compatible with the USB connection of the board, so any USB converters were needed during our lab. The components used and the connections are described as the following:

- L298N Motor Driver: This was the first component we needed. Firstly, the Driver needs power, so it is connected to a 9V battery composed of 6 AA batteries in series. We need to connect the battery and board ground to the same ground port in the Driver. The 5V output of the driver is connected to the board to supply power. Then, we have OUT1 and OUT2 which correspond to motor1 positive and negative and OUT3 and OUT4 for motor2. Finally, IN1 and IN2 are connected to board PINs PB8 and PA11 to control motor1, whereas IN3 and IN4 are connected to PB9 and PA12 for motor2.
- Ultrasonic Sensor HC-SR04: There is a PIN for ground and another for 5V. Then, the trigger PIN (connected to PA5) needs a 12us pulse to emit an ultrasonic pulse which will be later received by the echo PIN (PD2). With the pulse length measured by the echo, we can get the distance to some obstacle.
- Active Buzzer: We chose a passive buzzer because there was no need to control the frequency of the intermittent sound. Our buzzer only has two connections: one is for ground and the other one is connected to PA1. When the board turns a 1 in the PIN, there is sound. So, in the implementation, a timer oversees changing the value of the output.
- 10kΩ potentiometer: Potentiometers generally have 3 PINs. One is for ground, another for 3.3V and the other one is connected to PA4 in the board. Configuring that PIN in the board as analog, we receive a signal between 0 and 3.3V and as we have set it up in 12bits mode, the SR goes from 0 to 4095.
- Bluetooth Module HC05: Firstly, the Bluetooth module needed to be configured. For that, an Arduino UNO was used. The baud rate and a personalized name were set. Then, the PINs were connected in the following way: GND to ground and VCC to 5V. EN and STATE kept disconnected because they were not important for our purpose. Finally, it was important to connect HC05 TX PIN with board RX PIN(PB7) and vice versa (PB6 is TX PIN in NUCLEO board).
- Some minor detail, there was also an on-off switch added (welded and glued) to switch off the power to the Driver and the board and not be always on.


(Trigger)
PD2
5V Input
3.3V Output
5V Output
Ground
PB7
(HC 05
Tx)
PA1 (Buzzer)
(Potentiometer)
PB8 (IN1)
PB9 (IN3)
PA12 (IN4)
PA11 (IN2)
PA5 (Echo)
PB6 (Bluetooth
Rx)

3. Brief explanation about the microcontroller peripherals used and their basic setup.

The peripherals considered would be:

GPIO:

1) **Buzzer:** We consider pin PA1 as a digital output (01), and configure it with the GPIOA->MODER, placing a 0 in bit 3 and a 1 in bit 2.
2) **Trigger:** We consider pin PD2 as a digital output (01), and configure it with the GPIOD->MODER, placing a 0 in bit 5 and a 1 in bit 4.
3) **Echo:** We consider pin PA5 as an alternate function (10), specifically the one associated to Timer 2, that is AF1, and configure it first with the GPIOA->MODER, placing a 1 in bit 11 and a 0 in bit 10, and then clear the AF register and place a 1 in bit 20 of the AFR[0] (AF1 = 0001).
4) **Motor 1:** We consider two pins:
   a) PB8 as an alternate function (10), specifically the one associated to Timer 4, that is AF2, and configure it first with the GPIOB->MODER, placing a 1 in bit 17 and a 0 in bit 16, and then clear the AF register and place a 1 in bit 1 of the AFR[1] (AF2 = 0010).
   b) PA11 as a digital output (01), and configure it with the GPIOA->MODER, placing a 1 in bit 23 and a 0 in bit 22.
5) **Motor 2:** We consider two pins:
   c) PB9 as an alternate function (10), specifically the one associated to Timer 4, that is AF2, and configure it first with the GPIOB->MODER, placing a 1 in bit 19 and a 0 in bit 18, and then clear the AF register and place a 1 in bit 1 of the AFR[1] (AF2 = 0010).
   d) PA12 as a digital output (01), and configure it with the GPIOA->MODER, placing a 1 in bit 25 and a 0 in bit 24.
6) **Potentiometer:** We consider pin PA4 as an analog input (11), and configure it with the GPIOA->MODER, placing a 1 in bit 9 and a 1 in bit 8.

ADC:

1) **ADC1:** We consider AIN4 (channel 4). Before the configuration, we need to turn off the converter. Once it is done, in CR1 we configure the ADC with 12-bit resolution (RES = 00), enabling interrupts for end of conversion (EOCIE = 1), and disabling scan mode as it is not a PWM signal. We set to one the end of conversion mode (EOC = 1) and the default right aligning (AIGN = 0) in CR2, state the number of channels as 1 in SQR1 (L = 0000), and the channel number in SQR5, which is 4 (SQ1 = 00100). After the configuration is finished, we turn on again the converter in CR2 with ADON = 1.

TIMERS:

1) **TIMER 2:** This timer will be used for the Echo (channel 1 as TIC), and the Trigger (channel 2 as TOC with no output):

   1.1) **Internal clock:** First we disable the counter for configuration (CEN = 0) and as it is not a PWM, there would be no preload (ARPE = 0).
   1.2) **Counter behavior:** Taking 32 MHz as the frequency of the board and considering that the trigger pulse should be 12 us, we take a time unit of 1 us, so the frequency of unit is 10MHz. Dividing the board frequency by a PSC + 1 should give us the frequency unit, and with this we get that PSC = 31. The number of time units needed to reach 12 us is 12, so the CCR2 = TRIGGERPULSE = 15 (a little more time is given to avoid problems). The counter is initialized (CNT = 0), and the ARR set with the maximum value.

1.3) Setting the IRQ: First we clear all the dier bits, enable IRQ for channel 1 (ECHO), and disabling IRQ for channel 2 (TRIGGER).

1.4) Output mode: With the CCMR1, it is specified the timer mode, for channel 1 = TIC (CC1S = 01), for channel 2 = TOC (CC2S = 00). For channel 2, the preload is disabled and there is no output (0C2M = 000). Then with the CCER, in TOC we disable the hardware output (CC2E = 0) and the capture for the TIC is enabled (CC1E = 1) considering both edges (CC1P = 11).

1.5) Counter enabling: Start the counter (CEN = 1), update the registers (UG = 1) and clear the flags.

2) **TIMER 3:** We use this timer as a "principal loop", which would control the time for different functionalities (channel 1 as TOC no output).

2.1) Internal clock: First we disable the counter for configuration (CEN = 0) and as it is not a PWM, there would be no preload (ARPE = 0).

2.2)Counter behavior: Taking 32 MHz as the frequency of the board and considering that the time we want to set in the counter is 0.5 s, we take a time unit of 1 ms, so the frequency of unit is 1000Hz. Dividing the board frequency by a PSC + 1 should give us the frequency unit, and with this we get that PSC = 31999. The number of time units needed to reach 500 ms is 500, so the CCR1 = HALFSECONDPERIOD = 500. The counter is initialized (CNT = 0), and the ARR set with the maximum value.

2.3)Setting the IRQ: First we clear all the dier bits, enable IRQ for channel 1, and disabling IRQ for channel 2.

2.4)Output mode: With the CCMR1, it is specified the timer mode, for channel 1 = TOC (CC1S = 00), the preload is disabled and there is no output (0C1M = 000). Finally, with the CCER, we disable the hardware output (CC1E = 0).

2.5)Counter enabling: Start the counter (CEN = 1), update the registers (UG = 1) and clear the flags.

3) **TIMER 4:** We use this timer for the PWM generation, which will use channels 3 and 4.

3.1) Internal clock: First we disable the counter for configuration (CEN = 0) and as it is a PWM, there would be a preload (ARPE = 1).

3.2)**Counter behavior:** Taking 32 MHz as the frequency of the board and considering that the time we want to set in the counter is 1 ms, we take a time unit of 10 us, so the frequency of unit is 100000Hz. Dividing the board frequency by a PSC + 1 should give us the frequency unit, and with this we get that PSC = 319. The number of time steps needed to reach 1 ms is 100, so the ARR = 100 -1 = 99. The counter is initialized (CNT = 0), and the CCR3 and CCR4 are set with the maximum speed value.

3.3)**Setting the IRQ:** No IRQ are needed when the counting is finished, all dier bits set to zero.

3.4)**Output mode:** In this case, the CCMR1 is set to zero because channels 1 and 2 are not used. With CCMR2, it is specified the timer mode, for both channels = TOC -PWM, (CCyS = 00), the preload is enabled (OCyPE = 1) and the PWM starts with a 1 (0CyM= 110). Finally, with the CCER, we enable the hardware output (CC1E = 1) and set active high (CCyP = 0).

3.5)Counter enabling: Start the counter (CEN = 1), update the registers (UG = 1) and clear the flags.

NVIC:

1) **ADC1_IRQ:** Enabling IRQ for ADC1 with `NVIC->ISER[0] |= (1 << 18)` (bit for ADC = 18)
2) **TIM2_IRQ:** Enabling IRQ for TIM2 with `NVIC->ISER[0] |= (1 << 28)` (bit for TIM2 = 28)
3) **TIM3_IRQ:** Enabling IRQ for TIM3 with `NVIC->ISER[0] |= (1 << 29)` (bit for TIM3 = 29)

4. Which IRQ have you used, and the functionality archived by the ISRs

For this project we have used the following ISRs, for their respective peripherals, ADC1, TIM3, TIM2, and USART1, we are going to describe them and explain their respective functionalities.

1. ADC (ADC1_IRQHandler(void))

The Interrupt Service Routine associated with the ADC1, continuously checks whether the ADC has finished the conversion, once this condition is met, and the respective flag has been turned on, it will assign the value of the conversion, the value register in the maxvalue is value between 0-4095 and we mapped it to be between 99-50, to the variable max speed, once after that, it will clear the flag in the SR register, for future conversions. If the ISR is called and the flag is not active, then it will clear it without doing anything else.

2. TIM3 (TIM3_IRQHandler(void))

For TIM3, which is configured as TOC, when it gets to the desired time the ISR is launched. We have set up 2 channels, so the ISR, will handle different things for the different channels if we recall each timer could handle up to 4 channels.
If the SR register has launched a flag for Channel 1, then it will update the CCR1 so the counter of the TIM will count to another half second. After that, the ISR will start the conversion of the ADC that we have previously explained. It will also lunch the trigger pulse changing the BSRR register to 1, it will update the CCR2 of TIM2 to be the CNT added to the time needed for the pulse, in our case, it will be 15 us, so the CCR2 saves the moment the Trigger starts sending the pulse, in TIM2 it will stop we will explain that later. We also enable the DIER for channel 2 of the TIM2.
The ISR will also check the variable GlobalMode, which it's received in the USART, it will have different modes, if the value of GlobalMode is 1, then the ISR will call the function Stop, which will stop the car completely, if it equals to 2, then it will call the function goForward if the variable is equal to 3, then the function called it will be goBackwards when it's 4 then it will call goRight function when the variable is equal to 5, the function called it will be goRight. Then if the variable is 6 it will check if the variable cautionMode is equal to 0, which will be configured in TIM2 depending if the distance is less than 10 or not, in this case, if it's equal to 0 it will check for turnMode, that's another variable it will start the BSRR for the pin PA1 to start the Buzzer peripheral, it will call the function stop, disable the DIER associated to the echo as we know that there is an object less than 10 cm away to stop receiving any interruptions for the echo. We will set up the TIM2 channel 2 DIER that will be used for the sequence of turns the robot must follow when an object is detected. It will also update the CNT for the CCR2 of the TIM3 so it checks every half second, and change turnMode to 1 so the sequence configured in the main can start. In the case that turnMode is different than 0 it will also turn on the buzzer and stop the car. In the case that cautionMode is equal to 1, that would mean that the robot has an object between 10 and 20 cm of distance, then the forwardBrake function will be called, which uses a mathematic formula to calculate the distance. It will start turning on and the Buzzer as an alert measure, if the

cautionMode is else, then it means that the distance is greater than 20cm so it will turn off the buzzer, and call the function to go forward, in this function the TIM4 CCR3 and CCR4 will be the maximum speed send to the motors by a PWM, between 1 and 0. The ISR will also initialize the array received for the USART to 0 and clear the flags for channel 1 of TIM3.
If the SR register has a flag active for channel 2, which is used for the sequence when an object is detected, it will check the previousMode variable and then update the variable turnMode, to choose the next movement of the robot, once the turnMode variable is updated, then also the PrevTurnMode is updated as it will be equal to the actual turnMode variable, we disable the DIER for channel 2, and clear the flag.
This explains in a resumed way, what are the functionalities of the TIM3.

3. TIM2 (TIM2_IRQHandler(void))

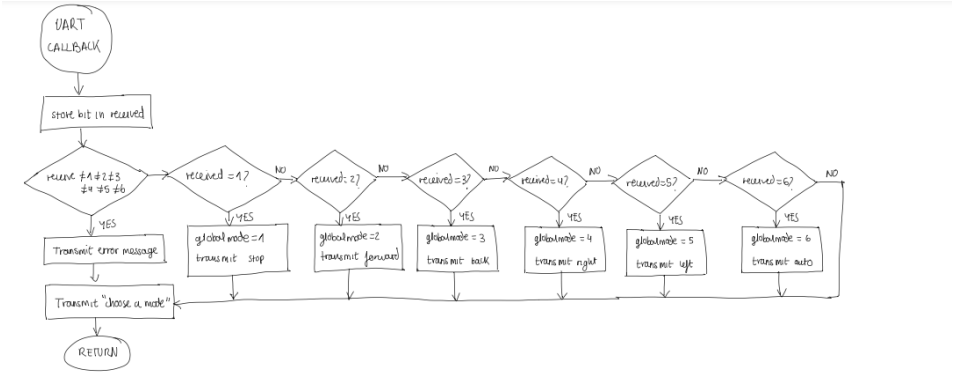
In TIM2 we also have 2 different channels, set up channel 2 and channel 1, this TIM is configured as a TIC which is used to measure the time lapse between external events.
When the ISR is launched, first it will check whether there is a flag in the SR register of TIM2 corresponding to channel 2, if there is one, it will stop the trigger, disable the DIER for this channel, and clear the flag, for this channel.
After that, it will check if there is a flag in the SR register, corresponding to channel 1, the one associated with the echo. We now check in the IDR in the PIN 5 if there is a 1, meaning that the echo has been received, if that is the case then we save in a variable the current CCR1, when we got the sequence 1-0 we know that echo has been up and down and we calculated the sequence and the echoStartTime, we handle the overflow, and calculate the distance, as the total time divided by 90. Then we decide depending on the distance, which is the mode the robot is in, if its less than 10, the robot has an obstacle less than 10 cm away and should set cautionMode to 0, and turnMode to 0, we also transmit via the USART "< 10" as is requested that any change to the trajectory is reported in the terminal. We also call a defined function that transmits to the terminal, a line break.
After that, we check if the distance is between 10 and 20 if it is, cautionMode will be 1, which will affect TIM3 as we have explained earlier, prevTurnMode will be 4, and we will transmit, that an obstacle is between 10 and 20 cm if the distance is bigger than 20, cautionMode will be 2, and we clear the flags for the channel 1, as a conclusion, this timer handles part of the trigger and the echo and some calculations, also it includes transmission to the terminal so it's easier for the user to know that is happening.

4. USART (HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)

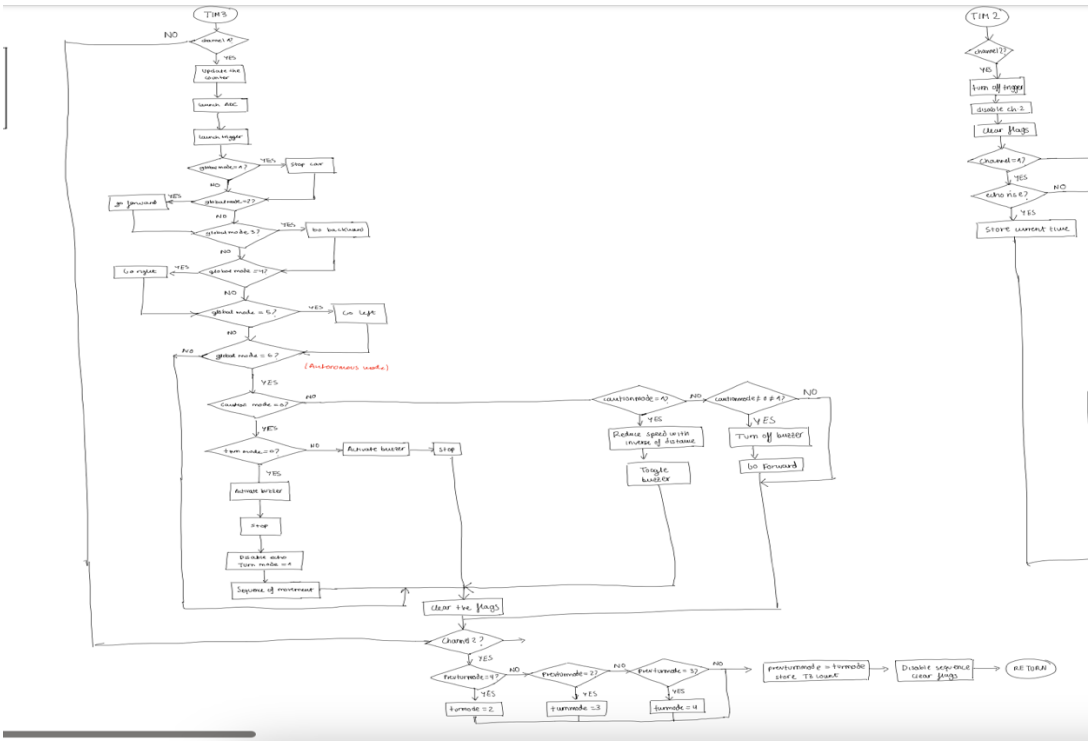
This function uses the HAL and allows us to communicate with the robot and send instructions, first, we start, then we check if, in the received variable, we have a 1, 2,3,4,5, or 6, which are the only values that we accept in our program, it will transmit, "Error" as they are not allowed.
Once one of the allowed values is received, depending on which is the value it will do different things, if the value received is 1, the program will select globalMode variable to 1, which will have an effect on TIM3 as we have seen before, also stop will be transmitted to inform of the changes in the trajectory, if the value received is a 2, then globalMode will be changed to 2, and forward, if it's 3, we will transmit back, and if it's 4 it will transmit right, 5 will be transmitting left and 6 will transmit automatic, also modifying globalMode, respectively, so then in TIM3, their respective functions for the selected movement or mode are realized.

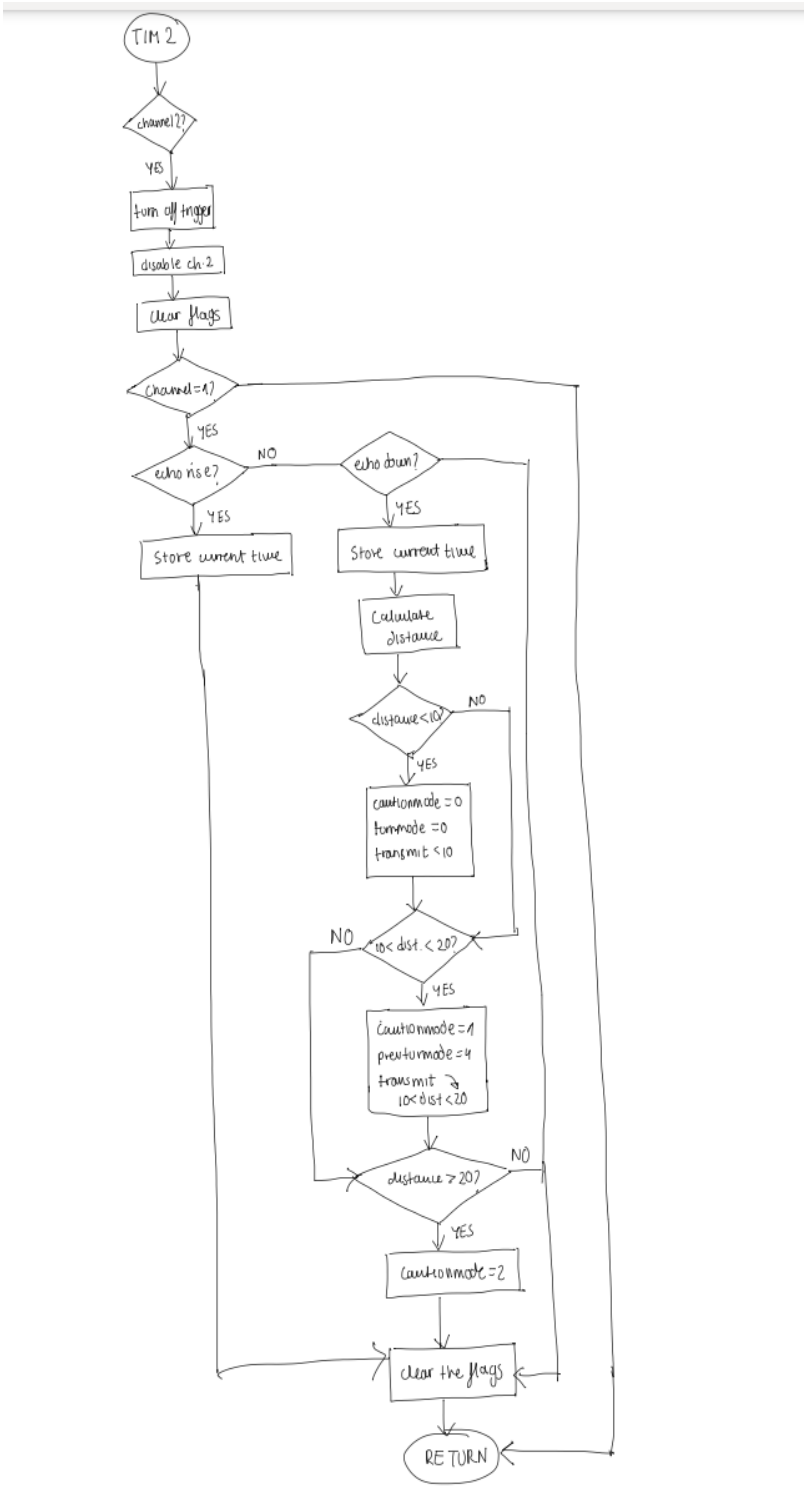
5. Flowcharts of the ISRs developed.

USART CALLBACK

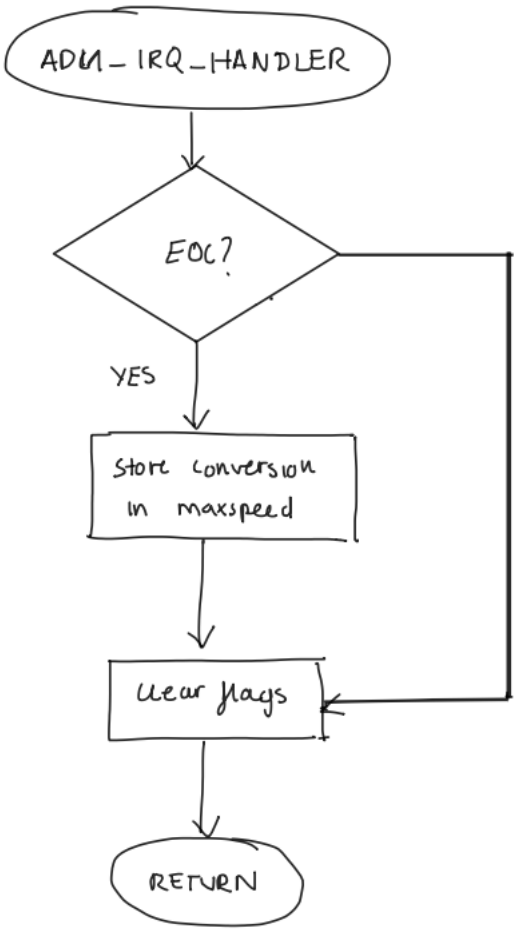


TIM3

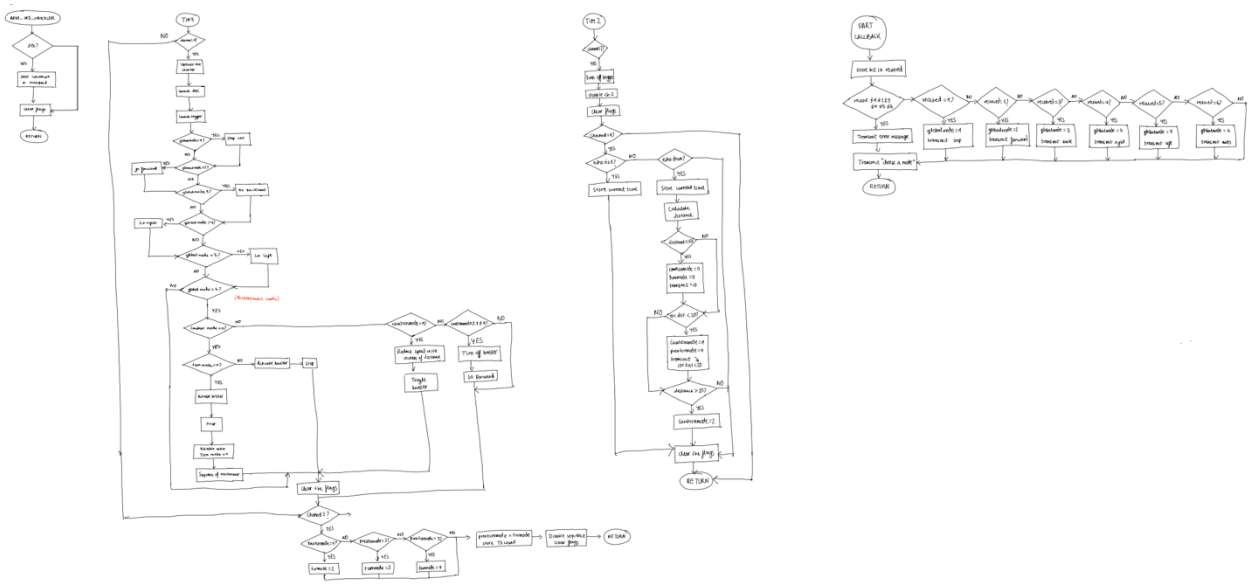



TIM 2
channel?
YES
turn off trigger
disable ch 2
clear flags
channel=1?
YES
echo rise?
NO
echo down?
YES
YES
Store current time
Store current time
Calculate distance
distance<10?
NO
YES
cautionmode = 0
fanmode = 0
transmit < 10
NO
10< dist. < 20?
YES
cautionmode = 1
preventmode = 4
transmit 10< dist <20
distance > 20?
NO
YES
cautionmode = 2
clear the flags
RETURN

TIM2



ADC



In the case the pictures can't be seen correctly we attach here a link to the pdf where all is in full resolution
[Link]

6. Conclusions

We have accomplished the correct configuration of the GPIO, the ADC, the IRQ, TIM, and the USART, and the proper use of them, although we have faced some challenges for example in LAB2 regarding the configuration and understanding of the echo, and the implementation with the TIMs and IRQ. In session 2 we faced different types of problems in the range of hardware, software, and even mathematical issues. In the end, we manage to come across the difficulties, and we accomplished the milestones required in the session.
The most fulfilling accomplishment was the proper Bluetooth functionality.
We as a group have learned how to overcome different difficulties and failures and how to learn from them.