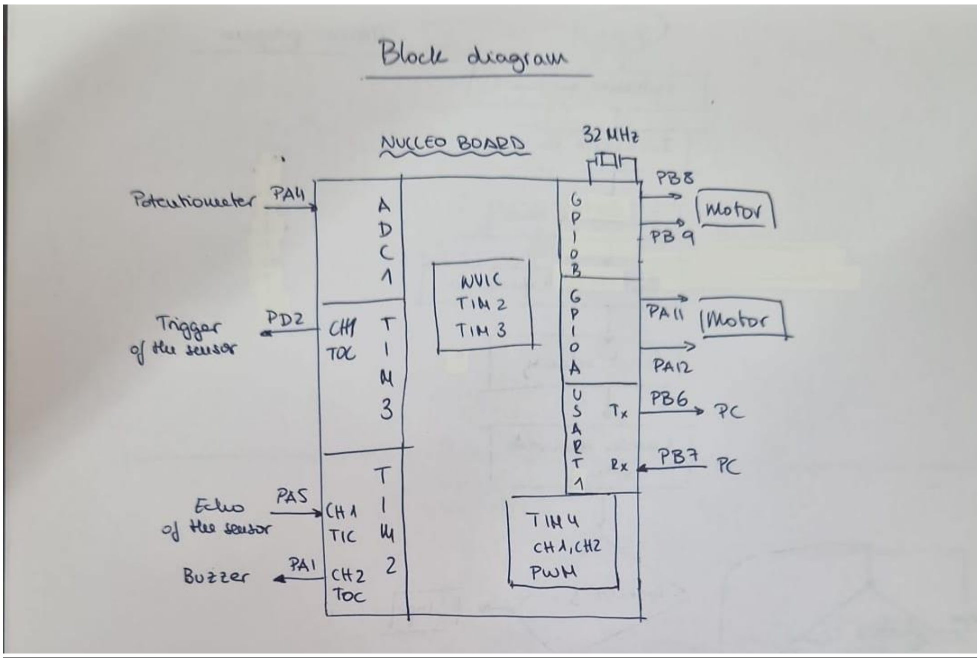
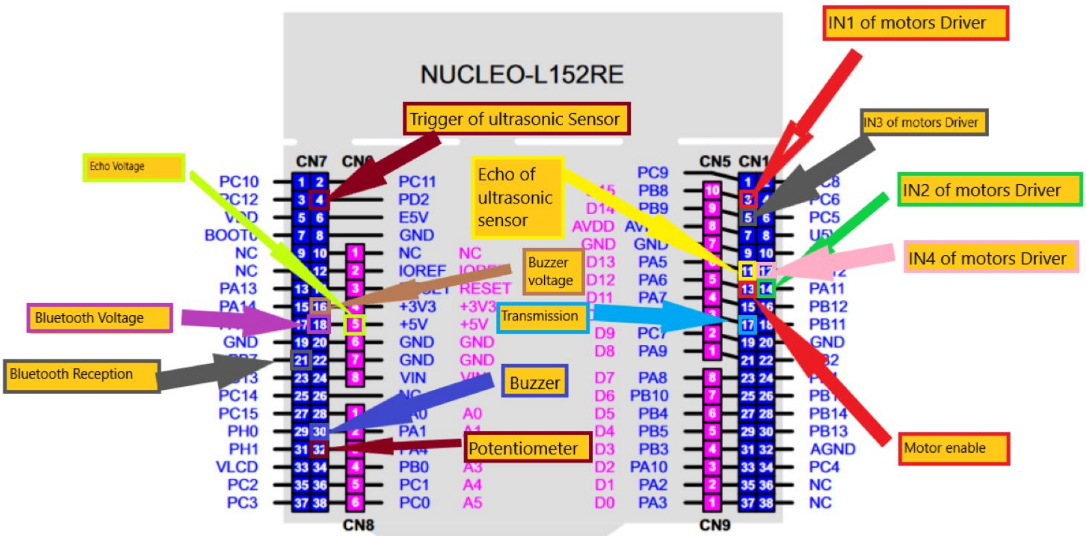
Block Diagram:



Components and Connections:



- We used a NUCLEO-L152RE board.
- A L298N Motor Driver Module
- A HC-SR04 ultrasonic ranging sensor
- A “HALJIA pasiva bajo Nivel gatillo zumbador Alarma Módulo Compatible con Arduino” Buzzer
- A HC05 Wireless Bluetooth Communication Module
- A A10K 16mm Potentiometer.

Brief Explanation of the microcontroller peripherals used and their basic setup:

- Potentiometer: in Analog functionality with GPIOA->MODER |=(1<<4) and GPIOA->MODER |=(1<<5), which corresponds to PA4. Then we initialize ADC1 CR2 as off, CR1 as 0 in all bits, CR2 with 0x00000400 so that EOCS is activated at the end of each regular conversion, SQR1 to 0 since we do one conversion, SQR5 to 0x00000004, and finally the first bit of CR2 for power on.
- Motor: we use PA6 as an output to enable the motor, and then MODER in GPIOB and GPIOA for each of the IN 1-4, we also associate IN 1 and 3 to TIM 4 so that we can later use PWM to adjust velocity changes (they are used to go forward, hence why we associate them). These IN go in AF 2, and we use AFRH to link them to TIM4.
- For the sensor we first set PD2 as an output using GPIOD MODER, however, this pin is closely related to TIM3 CH1 in TOC functionality. We set PA5 for the ECHO as a TIC of TIM2 CH1, using MODER and AFR.
- For Buzzer we use PA1, which is initialized as MODER and is set as an output.
- For the Bluetooth setup we configure PB6 and PB7 as usart functionality in Asynchronous mode, select the Basic parameters (9600, 8, N, 1), then set the communication as both ways. Our usart connects our program to the app provided by the Lab to allow Bluetooth communication from the app to the program.
- TIM2, it uses CH1 as a TIC, CH2 and 3 as TOC. TIM2 uses a pre-scaler equivalent to 319, so it can measure up to 0.6secs (approx). This decision is maybe not the best one as the CH1 measures much smaller quantities, however, the commodity is worth the loss of precision and the pre-scaler works perfectly for both the other channels, which measure bigger quantities. For CH2, we use TIM2 CCR2 to measure a third of a second, and for CH3, we use TIM2 CCR3 to measure half a second. Furthermore, the TIC of CH1 is activated in both rising and falling edges. All channels use interrupts.
- TIM3 uses CH1 as a TOC, it does not need a pre-scaler, as we are measuring a small quantity of time. We use CCR1 to measure 10 microseconds, which is why we give the value of 320.
- TIM4 CH3 and CH4 are used with PWM to modulate the velocity of the wheels. We use ARR equal to 320 to make the different velocities possible, we used 320 because in the original problem, we thought we needed to use 4 different velocities, so we needed it to be divisible by 4. In the beginning, both channels use the full duty cycle, until the user selects otherwise with the potentiometer.

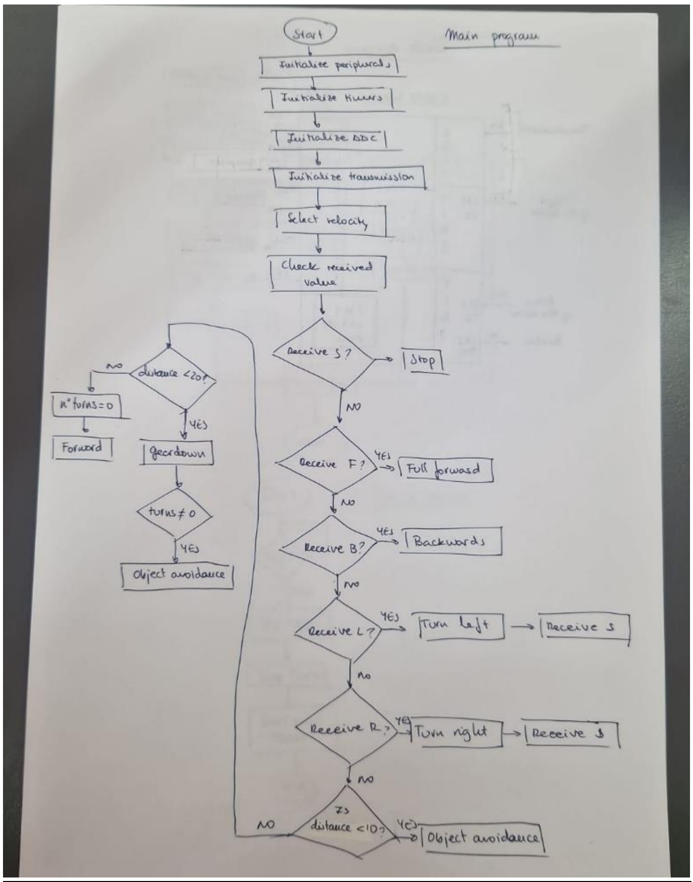
Which IRQ have you used, and the functionality achieved by their ISRs:

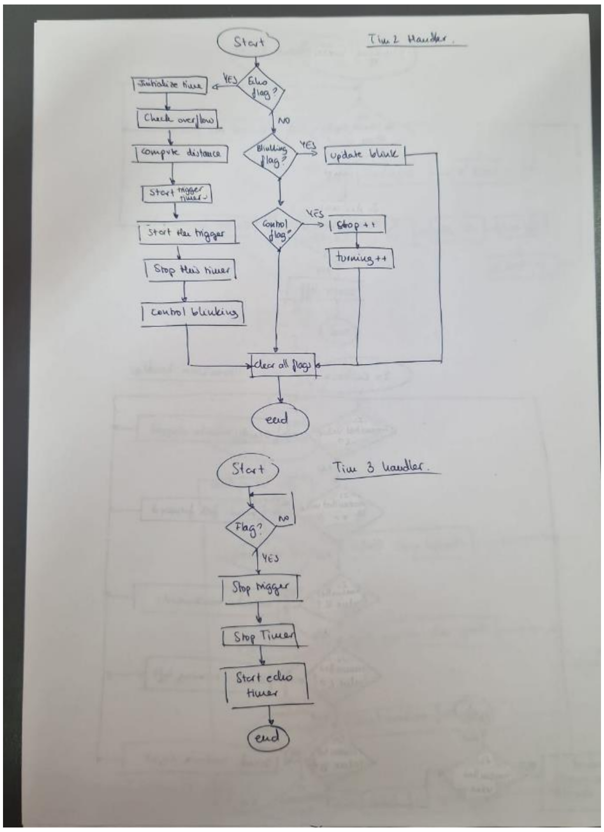
We use different ISR but mainly we use three IRQs, TIM2_IRQHandler: when the TIC in ch1 receives an interrupt, it stores the CCR1 in a variable, and uses it to compute the distance, then it decides what to do with the sound mode of the buzzer and then it starts the trigger and the timer associated to it again. In the second channel, we update the blinking of the buzzer (if needed) and in CH3, we manage the robot’s mobility functions, turning time of the wheels and time of stops.

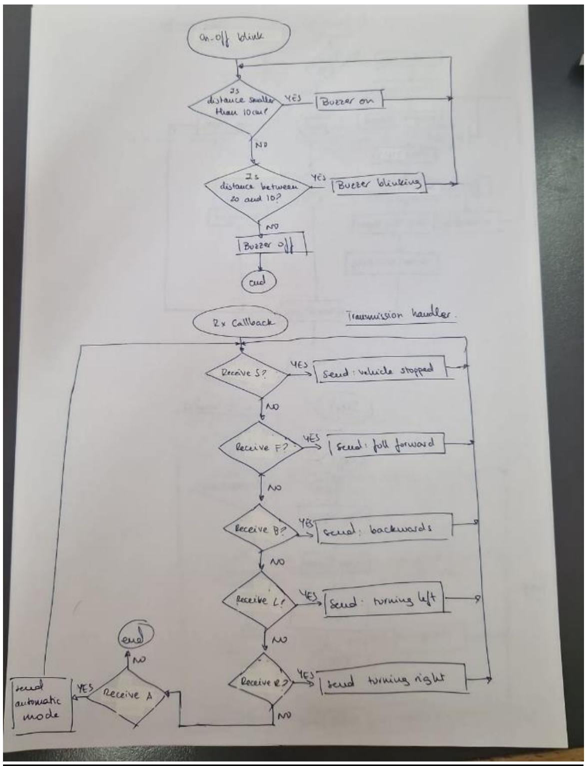
TIM3_IRQHandler: A simple handler that upon receiving an interrupt, it powers off the trigger and starts TIM2, then it stops itself.

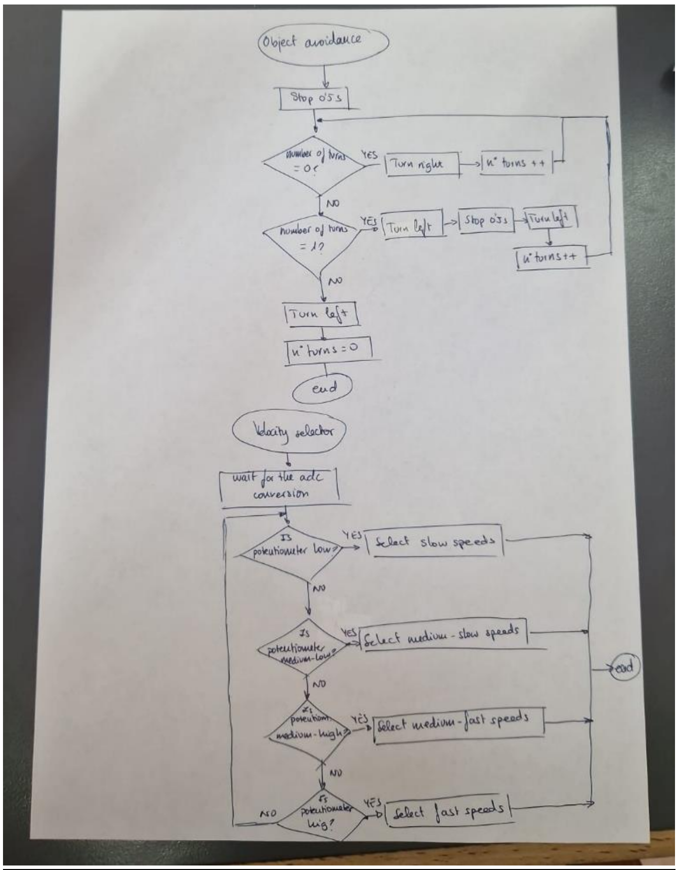
HAL_UART_RxCpltCallback: Manages the reception of the USART when the interrupt occurs, it stores the transmitted value that is in the reception buffer into a global variable, which is used afterwards to command the robot, it also gives feedback to the user by sending a simple command explaining the actions occurring.

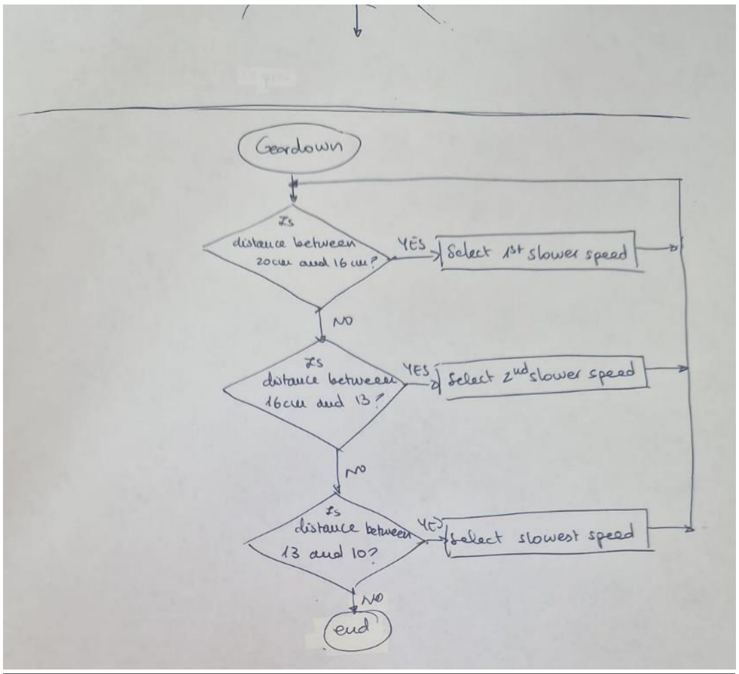
Flowcharts:




Start
Tim 2 Handler.
Echo flag?
YES
NO
Check overflow
compute distance
Start the trigger
Stop this timer
control blinking
Blinking flag?
Update blink
Control flag?
Stop ++
turning ++
clear all flags
end
Start
Tim 3 handler.
Flag?
Stop trigger
Stop Timer
Start echo timer
end


on-off blink
Is distance smaller than 10cm?
YES
NO
Buzzer on
Is distance between 20 and 10?
Buzzer blinking
Buzzer off
end
Rx Callback
Transmission handler.
Receive S?
Send: vehicle stopped
Receive F?
Send: full forward
Receive B?
Send: backwards
Receive L?
Send: turning left
Receive R?
send turning right
Receive A
send automatic mode
end





Conclusions:

We have managed to create a program that manages to solve most of the problems posed in the original lab assignment, that is controlled perfectly through an app and can turn, go, stop, and avoid objects, even adjust velocity, and produce sound. The only thing we didn't fully achieve was restructuring our TIM2 and TIM3 so that they would measure with more precision, due to the high value of the PSC in TIM2, this was because changing them would force a complete re-do of most of our functions, and this simply was not worth the time it would've taken. We completed the project except for the final extra-point part. In the future we would like to use the optocouplers to measure the velocity and re-structure the timers so that it works even better.