



Facultad de Ciencias de la **Alimentación**

Informe trabajo final Microcontroladores

Institución: UNER (Universidad Nacional de Entre Ríos) Facultad de Ciencias de la Alimentación.

Carrera: Ingeniería en Mecatrónica.

Asignatura: Microcontroladores

Docente: German Hachmann

Alumno: Gianluca Lovatto.

Año: 2020

Contenido

Introducción:	I
Planteo del sistema a controlar:	I
Diseño de la mecánica y hardware:	III
Diseño y desarrollo del software:	VII
1.1 Desarrollo del programa en QT.	VII
1.2 Desarrollo del programa en STM32CubeIDE.....	VIII
1.2.1 Configuración del sis clock	IX
1.2.2 Manejo de Timers para eventos	XI
1.2.3 SysTick Timers:	XI
1.2.4 NVIC	XII
1.2.5 Comunicación para transferencias de datos:	XIV
1.2.5.1 USB:	XV
1.2.5.2 Comunicación WIFI (UDP):	XVII
1.2.6 USART:	XIX
1.2.7 FLASH (guardado de datos):	XX
1.2.8 GPIO salidas digitales:	XXI
1.2.9 Conversor analógico-digital (ADC):	XXII
1.2.10 Acceso directo a memoria (DMA):	XXV
1.2.11 PULSE WIDTH MODULATOR (PWM):	XXV
1.2.12 Control proporcional, integral y derivativa PID:	XXVIII
1.2.12.1 Señal de referencia y señal de error	XXVIII
1.2.12.2 Proporcional:	XXIX
1.2.12.3 Derivativa:	XXIX
1.2.12.4 Integral:	XXX
1.2.12.5 Ecuación final de control:	XXXI
1.2.12.6 Sintonización del control PID:	XXXI
1.3 Problemas en el desarrollo y mejoras del sistema:	XXXII
1.3.1 Diseño de la mecánica:	XXXII
1.3.1.1 Diseño del chasis:	XXXII
1.3.1.2 Distribución del peso:	XXXII
1.3.2 Diseño del Hardware:	XXXII
1.3.2.1 Armado de la electrónica:	XXXII
1.3.2.2 Inestabilidad del sistema:	XXXII

1.3.2.3	Armado del circuito de los sensores:	XXXIII
1.3.2.4	Distribución de los sensores:	XXXIII
1.3.2.5	Efecto de la luz solar sobre los sensores	XXXIII
1.3.2.6	Autonomía	XXXIII
1.3.3	Diseño del programa en QT.....	XXXIV
1.3.4	Diseño del PID.....	XXXIV
1.4	Conclusiones:	XXXV

Introducción:

En este informe se desarrollara y describirá el proyecto realizado en la asignatura de Microcontroladores del cuarto año de la Ingeniería en Mecatrónica. Este proyecto se fue desarrollando y concretando de manera conjunta con la cursada de la materia y adquisición de conocimientos como una forma de aplicar los mismos.

El planteo del proyecto consistía en el armado de un sistema (hardware) que va a ser controlado mediante un microcontrolador de 32 bits. A medida de que se adquieren conocimientos sobre arquitectura ARM, funcionamiento y programación de la misma se realizaba la práctica aplicando dicho conocimiento para el control del sistema.

Planteo del sistema a controlar:

El sistema (hardware) consiste en un auto el cual tiene que tener la capacidad de funcionar de manera autónoma (alimentado mediante baterías) y poder andar recto entre dos paredes en una pista rectangular, lo que contempla una curva para cambiar de carril.

Para concretar todos estos requisitos el sistema se tuvo que diseñar y programar siguiendo una serie consideraciones como:

- El control de la velocidad de los motores se debe de realizar mediante un control PID obteniendo datos de la posición con sensores ópticos reflectivos.
- Como la distancia de medición o detección de los sensores no es tan amplia los mismos deben de estar ubicados de manera estratégica en el auto. A su vez las dimensiones de dicho auto deben de ser las apropiadas para conseguir un balance en el giro.
- El peso y tamaño del auto debe de ser tal que permita el avance del mismo con la fuerza suministrada por los motores. Esto lleva a ubicar los componentes en lugares donde nos generen un mayor torque en las ruedas.
- El auto debe de andar de manera autónoma por lo que debe de ser alimentado mediante baterías.
- El auto debe de poder ser controlado mediante una PC de manera inalámbrica o mediante USB
- El auto debe de proveernos a la PC toda la información necesaria para su control. Esta información serán datos que se representaran en gráficas para su posterior interpretación.

Diseño de la pista:





Diseño de la mecánica y hardware:

Como las clases no eran presenciales debido a la pandemia, cada alumno debió de desarrollar el diseño y construcción del auto adaptándose a las herramientas y materiales disponibles.

Lo primero que se realizó fue el diseño del chasis buscando un equilibrio entre tamaño, peso y disponibilidad de espacio en el mismo para ubicar toda la electrónica. Para concretar el diseño se va colocando toda la electrónica necesaria sobre un chasis buscando el equilibrio del sistema. Una vez encontrado el equilibrio y ubicada toda la electrónica se procede a darle las dimensiones finales al chasis.

Los materiales necesarios para la construcción son:

- x1 placa de desarrollo BluePill con el Microcontrolador STM32F103C8T6 basado en la arquitectura ARM CORTEX-M3.





- x2 motores de 600rpm 6v.



- x2 baterías de ION litio 3.2 a 4.2V.



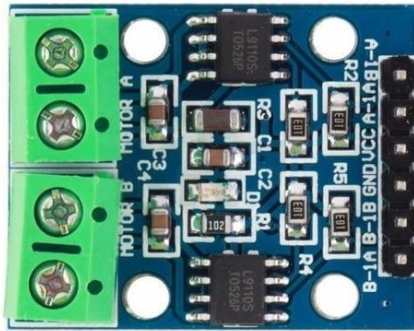
- x2 portabaterías.



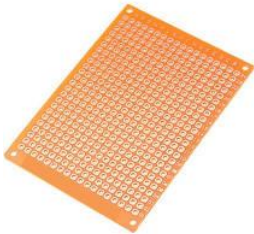
- x1 Fuente Switching Step Down Mp1584 Modulo Dc 4,5-28v 3a Ptec



- x1 puente H de dos canales.



- x1 placa pre perforada.



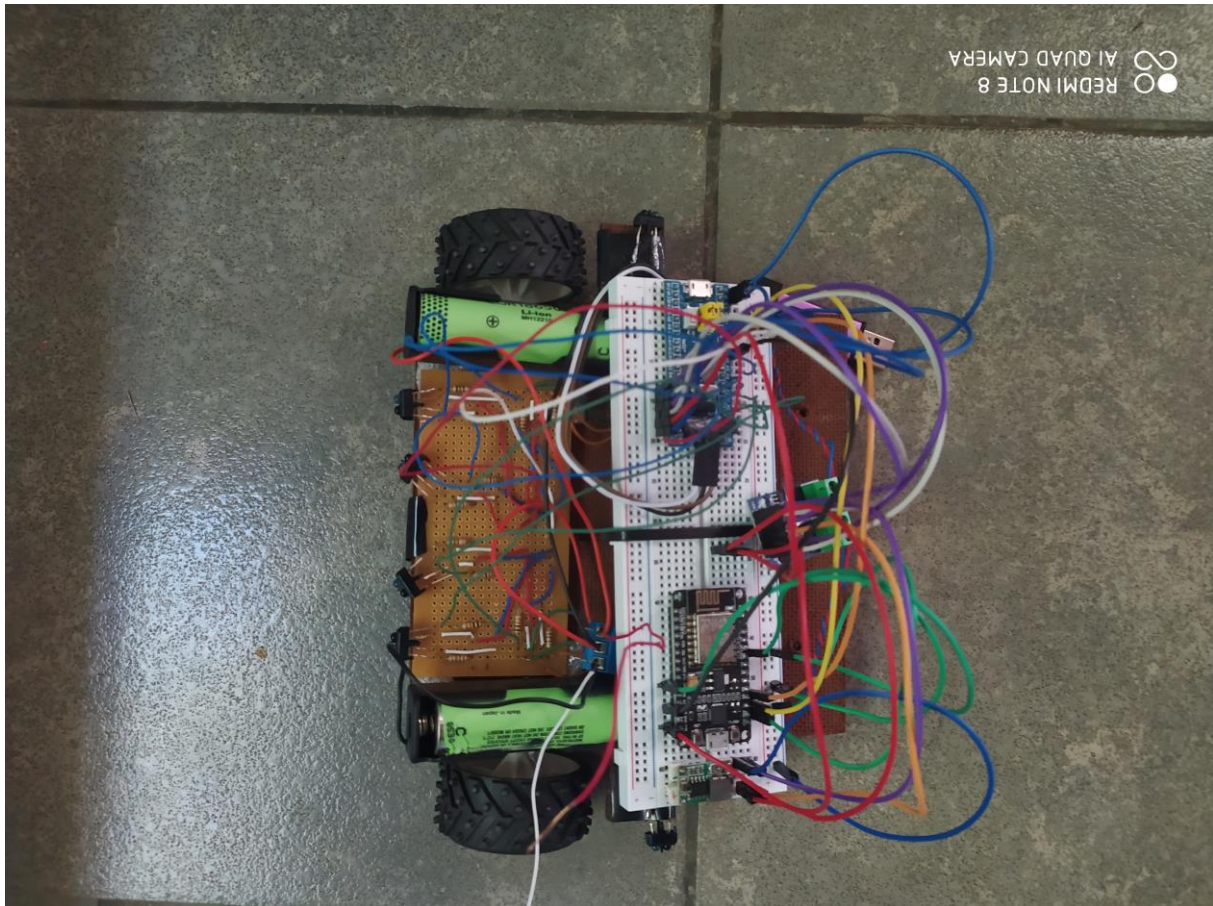
- x1 protoboard.



- x1 chasis diseñado a la necesidad.

Para el armado de la parte electrónica se procede a realizar primero el circuito de los sensores en una placa preperforada. Luego todas las conexiones entre placas, salida de sensores, motores, puente H y alimentación fueron realizadas con cables monofilamento que añade cierto ruido a las entradas de nuestro sistema.

Por ultimo una vez terminada la electrónica se procede a montar la misma sobre el chasis dando por terminado el sistema. Cabe resaltar que es un prototipo muy rudimentario con fines prácticos y no profesionales para volcar todo el conocimiento aprendido en el cursado de la materia. Esto le agrega cierto grado de dificultad al proyecto ya que muchos cambios y balances se deben de ir haciendo sobre la marcha.

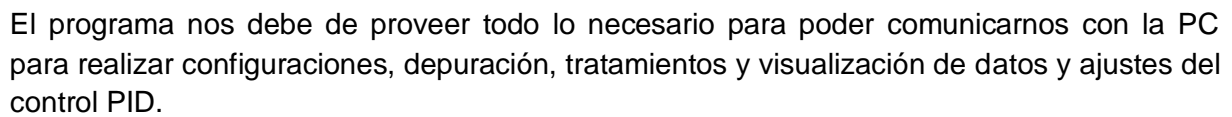


Diseño y desarrollo del software:

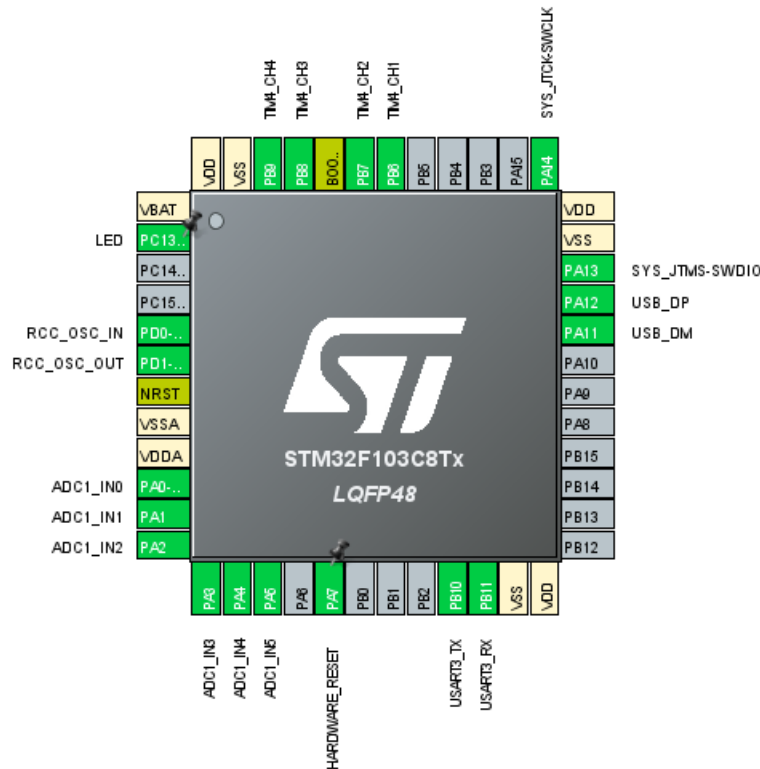
El software para el control del auto consta de dos programas uno para la PC desarrollado en QT y otro programa para la bluePill desarrollado en STM32CubeIDE.

1.1 Desarrollo del programa en QT.

Para el desarrollo del software en QT se realizó siguiendo un esquema:

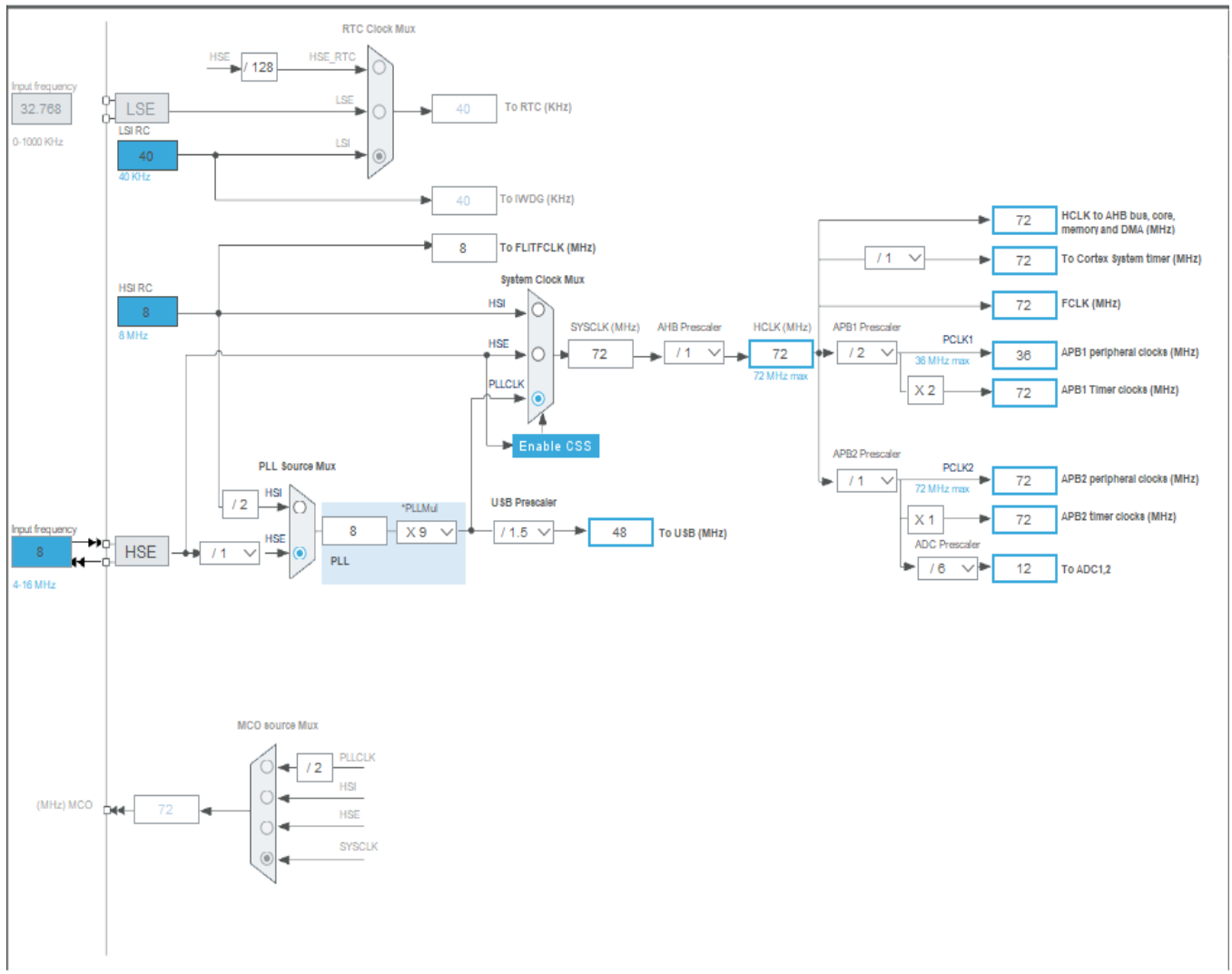


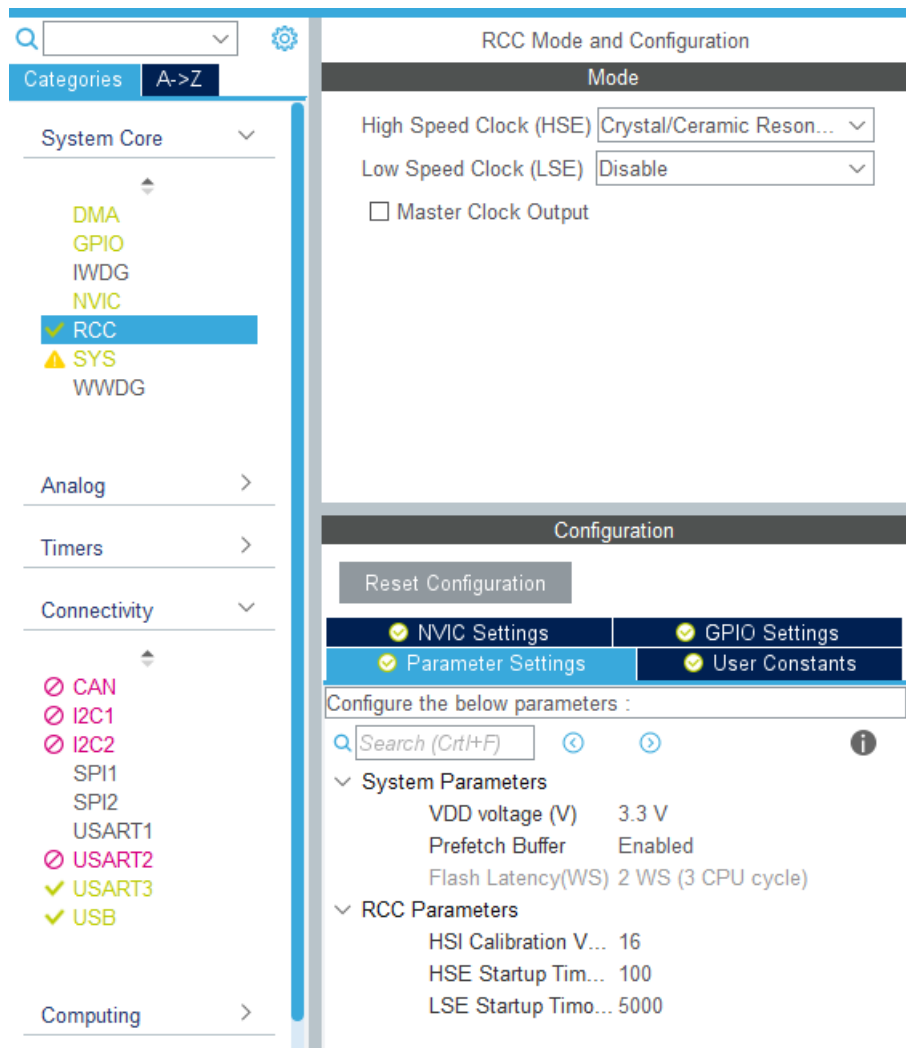
Para el desarrollo del programa se utilizaron diferentes módulos del microcontrolador los cuales se configuraron según los requerimientos.



1.2.1 Configuración del sistema de reloj

Lo primero que se realiza para que el microcontrolador pueda ejecutar instrucciones a una determinada velocidad es la configuración del System Clock, el cual consiste en una serie de prescalers y multiplicadores que configuran las fuentes de reloj de los distintos periféricos para que estos se adapten a nuestros requerimientos.





1.2.2 Manejo de Timers para eventos

Para que se ejecuten varias tareas de forma simultánea cada determinado periodo de tiempo sin tener que esperar una respuesta para que se siga ejecutando el programa se debe de utilizar muy bien los recursos que nos provee el microcontrolador.

Para el manejo de tiempos se utiliza la librería Tickers la cual se basa en la interrupción del SysTick_Handler y un vector tipo puntero a estructura. En la interrupción se incrementa un contador cada 1ms, una vez llegado ese contador al valor seteado se realiza el llamado de la función que se recibe como parámetro de tipo puntero. El llamado de la función se puede realizar dentro de la interrupción si es de alta prioridad o fuera de la función si es de baja prioridad.

1.2.3 Systick Timers:

La Systick Timers se trata de un temporizador de 24 bits de cuenta descendente, que produce una interrupción cuando el registro interno llega a cero desde el valor de recarga inicial.

El flag COUNTFLAG se pone a 1 cuando el contador pasa del valor de cuenta 1a 0. Para que vuelva a 0 se puede leer el registro de control de SysTick, o poner a 0 la cuenta escribiendo cualquier valor en el registro de cuenta.

Su mayor ventaja es su sencillez, dado que no hay que configurar nada excepto el valor de recarga citado anteriormente. El inconveniente es que al no poder configurar preescalas ni otros parámetros las temporizaciones que se pueden realizar son bastante básicas.

Offset	Registro	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	STK_CTRL Valor de reset	Reservado															COUNTFLAG 0	Reservado													CLKSOURCE 1	TICKINT 0	ENABLE 0
0x04	STK_LOAD Valor de reset	Reservado									RELOAD [23:0] 0																						
0x08	STK_VAL Valor de reset	Reservado									CURRENT [23:0] 0																						
0x0C	STK_CALIB Valor de reset	Reservado									TENMS [23:0] 0																						

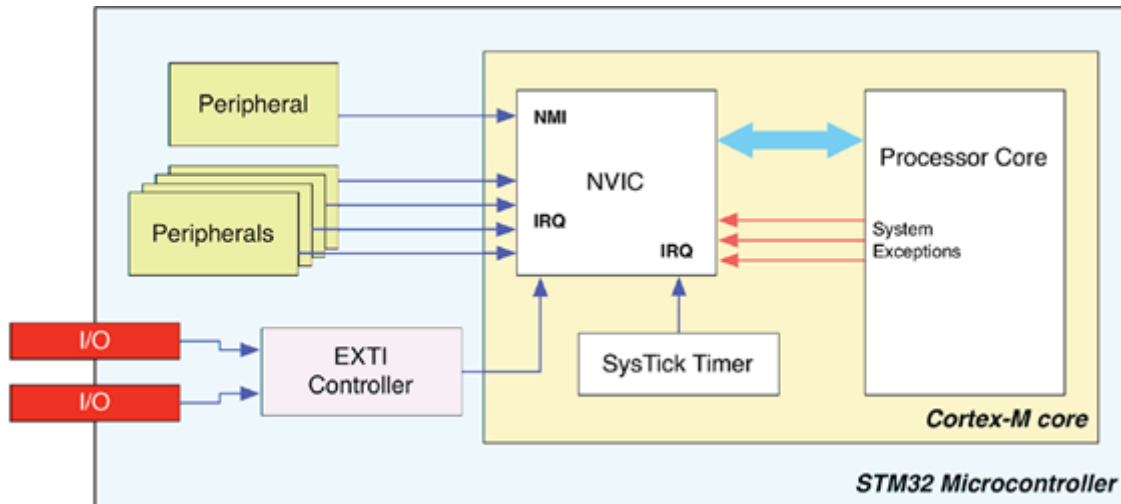
Figura 1. Mapa de registros SysTick

1.2.4 NVIC

Una interrupción es un evento asíncrono que provoca detener la ejecución del código actual en base a una prioridad (cuanto más importante es la interrupción, mayor será su prioridad, lo que hará que una interrupción de menor prioridad se suspenda). La rutina que se ejecuta durante una interrupción se denomina “Rutina de servicio de interrupción” o “Interrupt Service Routine” por sus siglas en ingles ISR.

El NVIC (Administrador de vector de interrupciones anidadas) es una unidad de hardware dentro de los microcontroladores basados en Cortex-M que es responsable de la administración de interrupciones.

La imagen muestra la relación entre la unidad NVIC, el núcleo del procesador y los periféricos. Aquí tenemos que distinguir dos tipos de periféricos: los externos al núcleo Cortex-M, pero internos a la MCU STM32 (por ejemplo, temporizadores, UARTS, etc.) y los periféricos externos a la MCU. La fuente de las interrupciones procedentes de la última clase de periféricos son puertos GPIO, que pueden configurarse como entrada o salida (por ejemplo, un botón conectado a un pin configurado como entrada) o para excitar un periférico externo avanzado (por ejemplo, un módulo USB). Un controlador programable dedicado, llamado “External Interrupt/Event Controller” (EXTI), es responsable de la interconexión entre las señales de E/S externas y el controlador NVIC.



Las interrupciones se identifican mediante números enteros, asociándose números negativos a las interrupciones proporcionadas por el núcleo ARM Cortex-M y las positivas al diseño particular de cada fabricante.

Las interrupciones con número más bajo tendrán prioridad, lo que significa que si hay dos peticiones simultáneas, se atenderá la que tenga el menor número y, cuando esta termine, se podrá atender a la otra. Si durante la ejecución de una interrupción llega otra de mayor prioridad, entonces la interrupción de baja prioridad es interrumpida a su vez para poder atender a la de mayor prioridad.

Configuración en el ioc:

Categories
A->Z

System Core

DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG

Analog
Timers
Connectivity

CAN
I2C1
I2C2
SPI1
SPI2
USART1
USART2
USART3
USB

Computing
Middleware

Configuration

NVIC
Code generation

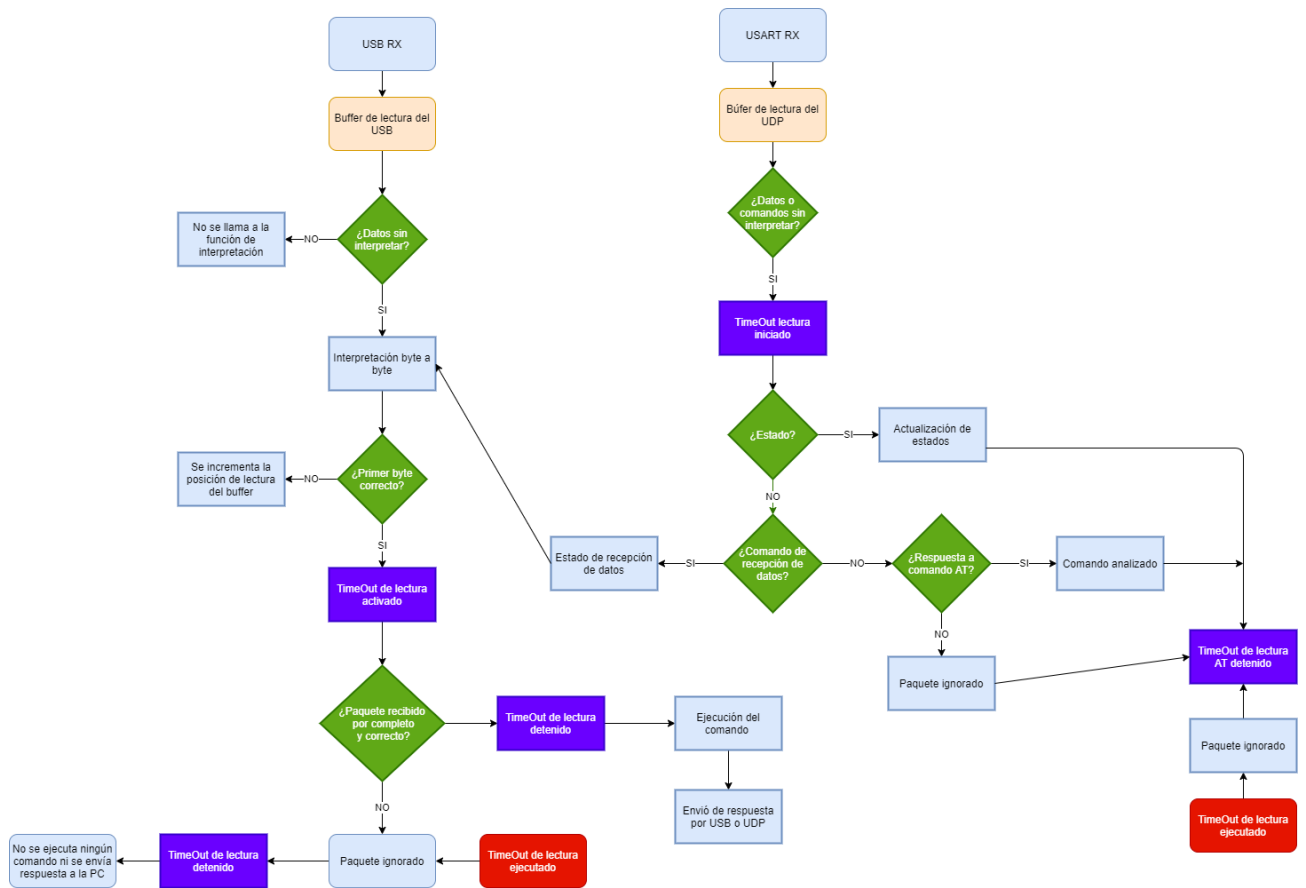
Priority Group
4 bits for pre-emp...
Sort by Preemption Priority and Sub Priority

Search
Search (C...)
Show only enabled interrupts
Force DMA channels Interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
DMA1 channel1 global interrupt	<input checked="" type="checkbox"/>	0	0
ADC1 and ADC2 global interrupts	<input type="checkbox"/>	0	0
USB high priority or CAN TX interrupts	<input type="checkbox"/>	0	0
USB low priority or CAN RX0 interrupts	<input checked="" type="checkbox"/>	0	0
TIM4 global interrupt	<input type="checkbox"/>	0	0
USART3 global interrupt	<input checked="" type="checkbox"/>	0	0

1.2.5 Comunicación para transferencias de datos:

La comunicación se realiza mediante dos vías USB y UDP. Las mismas se realizaron en base al siguiente esquema:



1.2.5.1 USB:

El bus universal en serie, consiste en una norma para bus periférico, desarrollado tanto por industrias de computación como de telecomunicaciones. USB permite adjuntar dispositivos periféricos a la computadora rápidamente, sin necesidad de reiniciar la computadora ni de volver a configurar el sistema. Los dispositivos con USB se configuran automáticamente tan pronto como se han conectado físicamente. Es un bus de comunicaciones que sigue un estándar que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos.



En nuestro caso la comunicación USB es de vital importancia ya que por ella se va a realizar el envío de datos necesarios para las configuraciones, depuración, tratamientos y visualización de los mismos, además de los ajustes del control PID.

Configuración del módulo USB en el ioc:

Primero se configura USB.

Timers >

Connectivity >

CAN

I2C1

I2C2

SPI1

SPI2

USART1

USART2

USART3

USB

Computing >

Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

Speed

Full Speed 12MBit/s

Power Parameters

Low Power

Disabled

Link Power Management

Disabled

Battery Charging

Disabled

Timers >

Connectivity >

CAN

I2C1

I2C2

SPI1

SPI2

USART1

USART2

USART3

USB

Computing >

Configuration

Reset Configuration

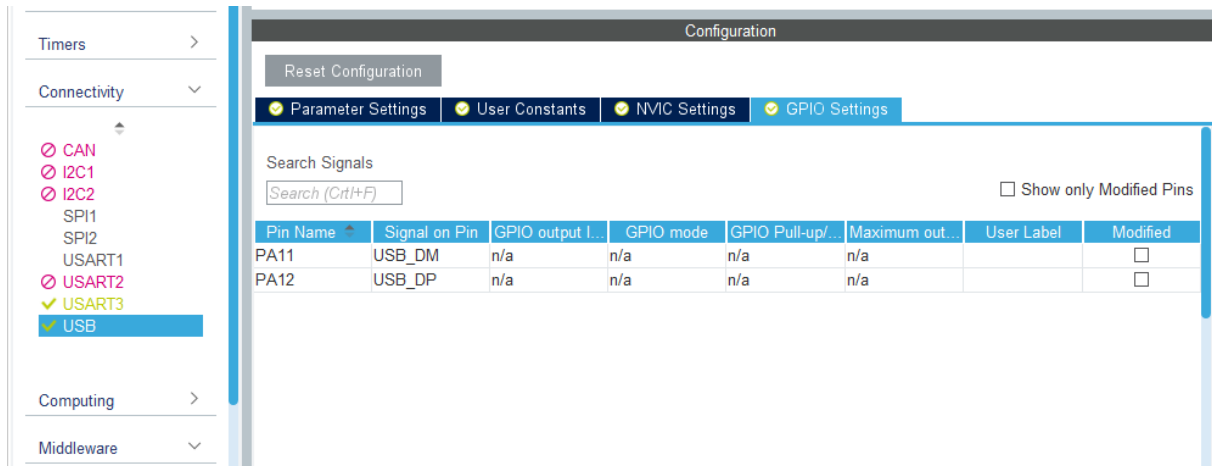
Parameter Settings

User Constants

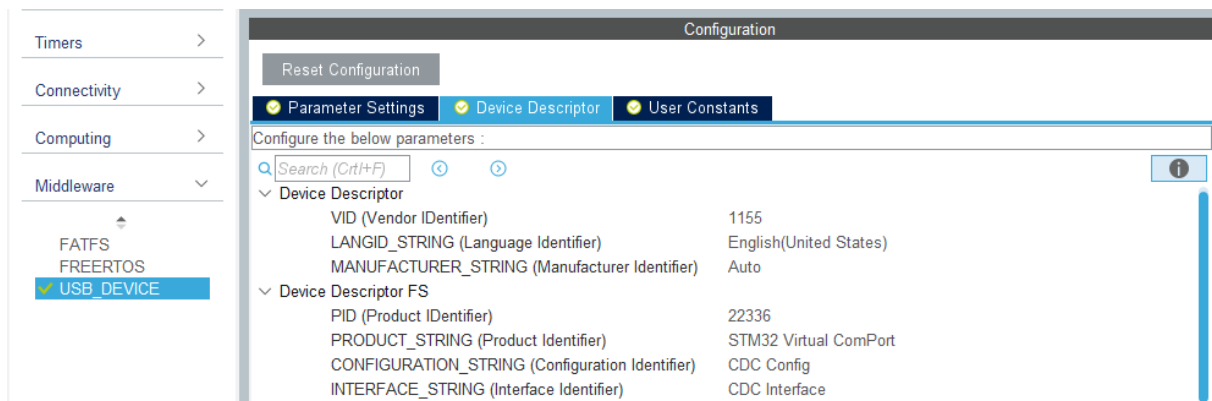
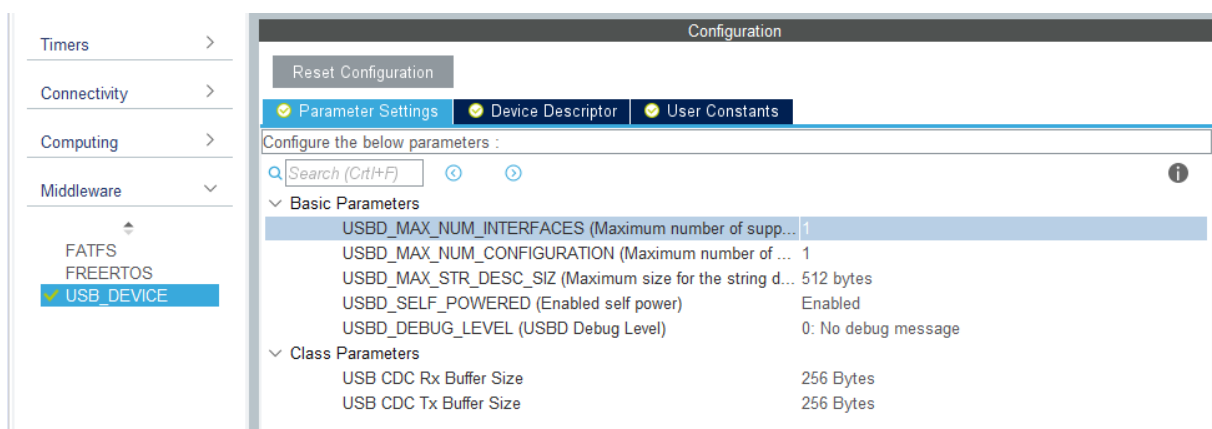
NVIC Settings

GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USB high priority or CAN TX interrupts	<input type="checkbox"/>	0	0
USB low priority or CAN RX0 interrupts	<input checked="" type="checkbox"/>	0	0



Luego se configura USB_DEVICE.



1.2.5.2 Comunicación WIFI (UDP):

La comunicación WIFI del auto se realiza con la placa de desarrollo Nodemcu WIFI ESP8266 que cuenta con capacidad de conectarse a redes WIFI y establecer comunicación mediante protocolo UDP/IP o TCP/IP.

Una red WIFI se trata básicamente de un sistema que permite que diferentes dispositivos electrónicos se conecten a las redes de comunicación a través de un punto de acceso de red inalámbrica (hotspot).

En nuestro caso se estableció una comunicación UDP/IP. El protocolo UDP se utiliza para transmitir datagramas de forma rápida en redes IP y funciona como una alternativa sencilla y sin retardos del protocolo TCP. Para obtener los servicios deseados en los hosts de destino, se basa en los puertos que están listados como uno de los campos principales en la cabecera UDP. Como muchos otros protocolos de red, UDP pertenece a la familia de protocolos de Internet, por lo que debe clasificarse en el nivel de transporte y, en consecuencia, se encuentra en una capa intermedia entre la capa de red y la capa de aplicación.

El protocolo UDP funciona sin conexión: el protocolo UDP se caracteriza porque permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión entre el emisor y el receptor. Los datagramas respectivos se envían a la dirección IP preferida de la secuencia especificando el puerto de destino, sin que sea necesario que el ordenador asociado al mismo tenga que dar una respuesta.

UDP utiliza puertos: al igual que el TCP, el protocolo UDP utiliza puertos para permitir que los datagramas se transfieran a los protocolos correctos, es decir, a las aplicaciones elegidas del sistema de destino. Los puertos quedan definidos mediante un número conforme a un rango de valores válidos, estando reservado el rango de 0 a 1023 para los servicios fijos.

El protocolo UDP permite una comunicación rápida y sin retardos: el protocolo de transporte es el adecuado para una transmisión de datos rápida debido a que no hay que llevar a cabo una configuración de la conexión. Esto resulta también del hecho de que la pérdida de un paquete individual afecta exclusivamente a la calidad de la transmisión.

El protocolo UDP no ofrece ninguna garantía de seguridad e integridad de los datos: la ausencia de acuse de recibo mutuo entre el emisor y el receptor garantiza que la velocidad de transmisión en el protocolo UDP sea excelente; no obstante, el protocolo no puede garantizar la seguridad ni la integridad de los datagramas. Tampoco puede garantizar el orden de los paquetes enviados. Por ello, los servicios que utilizan UDP deben aplicar sus propias medidas de corrección y protección.

El módulo WiFi es controlado mediante comandos seriales AT.

AT+CWMODE=X. Configura modo de operación.

AT+CWMODE: Indica el modo de operación actual.

AT+CWJAP="ssid","pass". Conecta a una red WiFi.

AT+CIPSTA: Lee la dirección IP, cliente y/o servidor.

AT+CIPSTA=ip. Asigna una IP al modulo cuando se configura como cliente.

AT+CWLAP: Proporciona una lista con las redes disponibles al módulo WiFi.

AT+CWQAP: Desconecta el módulo de la red.

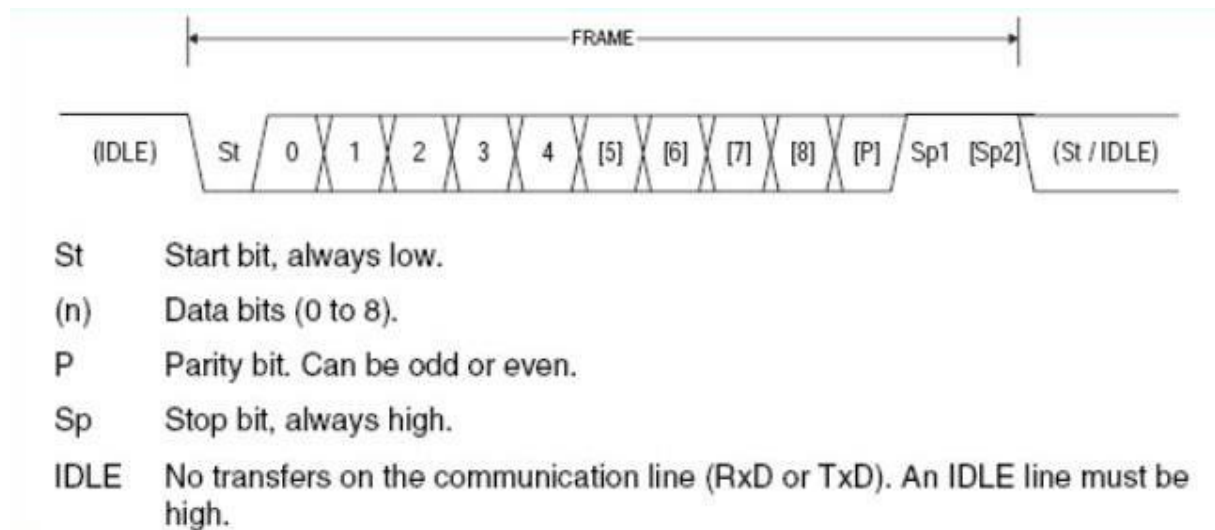
AT+RST: Resetea el modulo, y si ya tenía una red guardada se vuelve a conectar.

Todos estos comando AT están disponibles en el datasheet " ESP8266 AT Instruction Set".

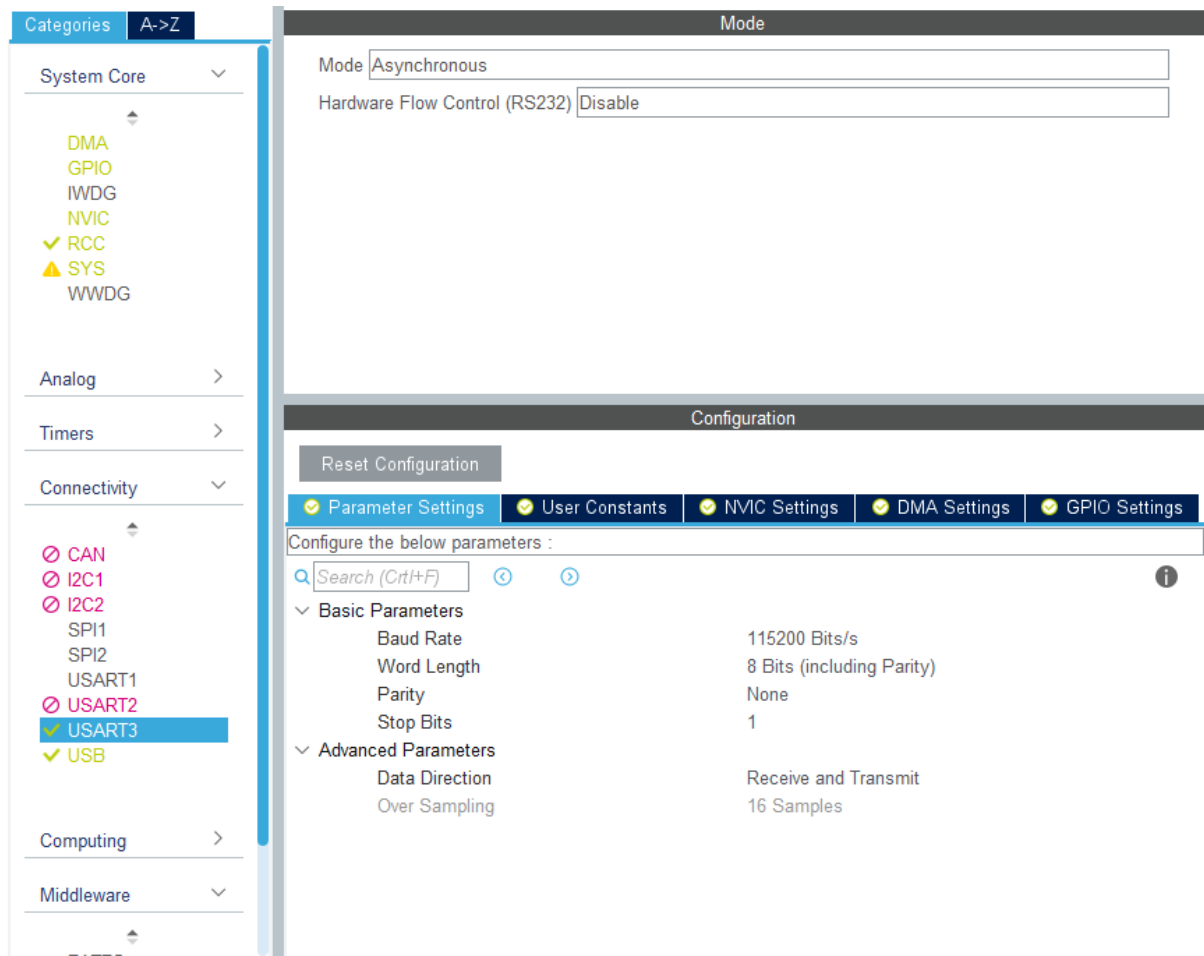
La Bluepill se comunica con la ESP8266 mediante protocolo RS232 por lo que se hace necesario la utilización de un módulo USART.

1.2.6 USART:

USART (Universal Synchronous/Asynchronous Receiver Transmitter ó Transmisor-Receptor Síncrono/Asíncrono Universal) es un protocolo empleado en comunicaciones duales, es decir que está en la capacidad de recibir y transmitir simultáneamente. Los datos son transmitidos de manera serial, lo que significa que sólo un bit es transferido por el canal al tiempo.



Configuración del módulo USART en el ioc:



1.2.7 FLASH (guardado de datos):

La memoria Flash es una memoria no volátil (no pierde la información guardada incluso quitando tensión o apagando el microcontrolador), de bajo consumo, se utiliza para alojar el programa que queramos ejecutar por el microcontrolador. El programa se guarda en la memoria flash, y una vez compilado no puede sobrecribirse de nuevo.

En nuestro programa se reservó la última página de la flash (tamaño 1k) para el guardado de datos necesarios en la conexión WiFi y el control PID. Una vez corriendo el programa se pasan los datos guardados de la flash a la RAM.

Configuración en el linkerscript:

```
MEMORY
{
  RAM      (xrw)      : ORIGIN = 0x20000000,   LENGTH = 20K
  FLASH    (rx)       : ORIGIN = 0x80000000,   LENGTH = 63K
  FLASHDATA (rx)      : ORIGIN = 0x800FC00,    LENGTH = 1K
}
```

```

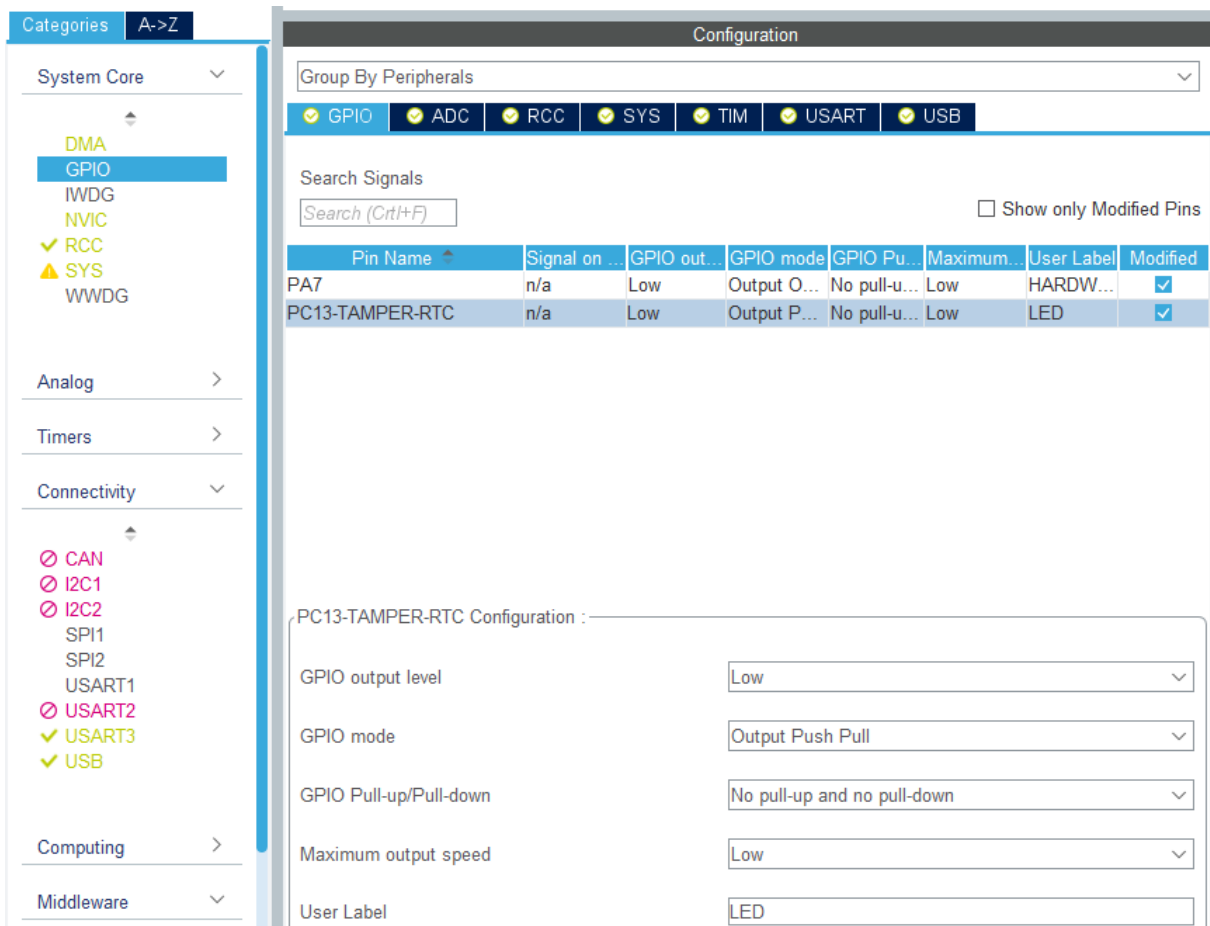
/* Datos almacenados en la flash */
.configdata :
{
    . = ALIGN(4);
    KEEP(*(.configdata))
    KEEP(*(.configdata*))
    . = ALIGN(4);
} >FLASHDATA

```

1.2.8 GPIO salidas digitales:

GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General) es un pin genérico en un microcontrolador, cuyo comportamiento (incluyendo si es un pin de entrada o salida) se puede controlar (programar) por el usuario en tiempo de ejecución.

Para nuestro caso se configuran dos pines como salidas digitales. Uno para el led de estado (PC13) de manera de tener un indicador visual cuando nuestro programa se encuentra corriendo. Y el otro (PA7) para la realización de un reset por HARDWARE de la ESP8266.



The screenshot shows the STM32CubeMX Configuration window. On the left, the 'Categories' pane is set to 'A->Z'. Under 'System Core', 'GPIO' is selected. The 'Configuration' pane shows 'Group By Peripherals' and tabs for various peripherals, with 'GPIO' selected. A table lists the configured pins:

Pin Name	Signal on	GPIO out	GPIO mode	GPIO Pu	Maximum	User Label	Modified
PA7	n/a	Low	Output O...	No pull-u...	Low	HARDW...	✓
PC13-TAMPER-RTC	n/a	Low	Output P...	No pull-u...	Low	LED	✓

Below the table, the 'PC13-TAMPER-RTC Configuration' is detailed:

- GPIO output level: Low
- GPIO mode: Output Push Pull
- GPIO Pull-up/Pull-down: No pull-up and no pull-down
- Maximum output speed: Low
- User Label: LED

Categories A->Z

System Core

- DMA
- GPIO**
- IWDG
- NVIC
- ✓ RCC
- ⚠ SYS
- WWDG

Analog >

Timers >

Connectivity

- ⊗ CAN
- ⊗ I2C1
- ⊗ I2C2
- SPI1
- SPI2
- USART1
- ⊗ USART2
- ✓ USART3
- ✓ USB

Computing >

Middleware

Configuration

Group By Peripherals

☒ GPIO
 ☒ ADC
 ☒ RCC
 ☒ SYS
 ☒ TIM
 ☒ USART
 ☒ USB

Search Signals

Search (Ctrl+F)

☐ Show only Modified Pins

Pin Name	Signal on ...	GPIO out...	GPIO mode	GPIO Pu...	Maximum...	User Label	Modified
PA7	n/a	Low	Output O...	No pull-u...	Low	HARDW...	<input checked="" type="checkbox"/>
PC13-TAMPER-RTC	n/a	Low	Output P...	No pull-u...	Low	LED	<input checked="" type="checkbox"/>

PA7 Configuration :

GPIO output level: Low

GPIO mode: Output Open Drain

GPIO Pull-up/Pull-down: No pull-up and no pull-down

Maximum output speed: Low

User Label: HARDWARE_RESET

1.2.9 Conversor analógico-digital (ADC):

Un conversor o convertidor de señal analógica a digital es un dispositivo electrónico capaz de convertir una señal analógica, ya sea de tensión o corriente, en una señal digital mediante un cuantificador y codificándose en muchos casos en un código binario en particular. Donde un código es la representación unívoca de los elementos, en este caso, cada valor numérico binario hace corresponder a un solo valor de tensión o corriente. Permitiendo una comunicación entre Hardware y Software.

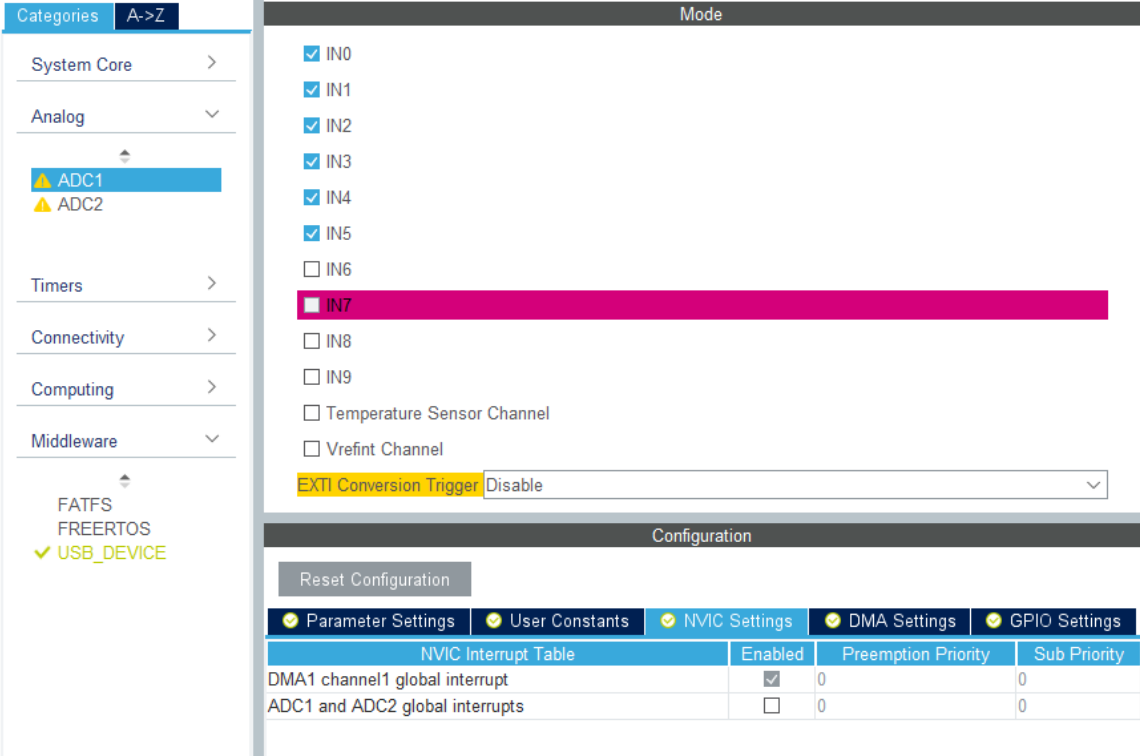
Como nuestro auto cuenta con 6 sensores infrarrojos (ópticos reflectivos) para poder posicionarse en la pista y realizar el recorrido de la misma bajo las condiciones antes mencionadas es indispensable el uso del ADC. A cada sensor le va a corresponder un canal del ADC. Algunos de los parámetros a configurar del mismo son:

- Clock Prescaler: la frecuencia a la que se capturarán datos del ADC.
- Scan Conversion Mode: este modo se activa cuando queremos usar varios canales del ADC. Este método se utiliza para convertir todos los canales de un mismo grupo. Una conversión es realizada por cada canal del grupo. Después de cada conversión, el siguiente canal del grupo se convierte automáticamente.
- Continuous Conversion Mode: al activarlo el ADC se pone a leer muestras de forma continua.

- Discontinuous Conversion Mode: se utiliza para convertir un grupo cerrado de conversiones.
- DMA Continuous Requests: se habilita cuando queremos utilizar el DMA.
- End Of Conversion Selection: este flag se utiliza para determinar cuando se ha realizado una conversión.
- Number of Conversion: determina el número de canales que van a realizar conversiones.

En nuestra aplicación utilizamos un Scan Conversion Mode con un Number of Conversion de 6 y habilitamos el DMA Continuous Requests. La conversión se activa por software cada 2ms.

Configuración en el ioc:



The screenshot shows the Ioc configuration interface. On the left is a sidebar with categories: System Core, Analog, Timers, Connectivity, Computing, and Middleware. Under Analog, ADC1 and ADC2 are listed. Under Computing, USB_DEVICE is checked. The main area is titled 'Mode' and shows a list of input channels (IN0 to IN9) with checkboxes. IN0 through IN5 are checked, IN6 is unchecked, and IN7 is highlighted with a pink bar. Below the channels, there are options for Temperature Sensor Channel and Vrefint Channel, both unchecked. A dropdown menu for 'EXTI Conversion Trigger' is set to 'Disable'. Below the Mode section is the 'Configuration' section, which includes a 'Reset Configuration' button and a table for NVIC Interrupt Table settings.

Configuration				
Reset Configuration				
Parameter Settings	User Constants	NVIC Settings	DMA Settings	GPIO Settings
NVIC Interrupt Table		Enabled	Preemption Priority	Sub Priority
DMA1 channel1 global interrupt		<input checked="" type="checkbox"/>	0	0
ADC1 and ADC2 global interrupts		<input type="checkbox"/>	0	0

Categories
A-Z

System Core >

Analog v

ADC1

ADC2

Timers >

Connectivity >

Computing >

Middleware v

FATFS

FREERTOS

USB_DEVICE

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings GPIO Settings

Search (Ctrl+F)

ADCs_Common_Settings

Mode Independent mode

ADC_Settings

Data Alignment Right alignment

Scan Conversion Mode Enabled

Continuous Conversion Mode Disabled

Discontinuous Conversion Mode Disabled

ADC_Regular_ConversionMode

Enable Regular Conversions Enable

Number Of Conversion 6

External Trigger Conversion Source Regular Conversion launched by software

Rank

Channel Channel 0

Sampling Time 239.5 Cycles

Rank 2

Rank 3

Rank 4

Rank 5

Rank 6

ADC_Injected_ConversionMode

Enable Injected Conversions Disable

WatchDog

Enable Analog WatchDog Mode

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings GPIO Settings

DMA Request	Channel	Direction	Priority
ADC1	DMA1 Channel 1	Peripheral To Memory	Low

Add
Delete

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA1 channel1 global interrupt	<input checked="" type="checkbox"/>	0	0
ADC1 and ADC2 global interrupts	<input type="checkbox"/>	0	0

1.2.10 Acceso directo a memoria (DMA):

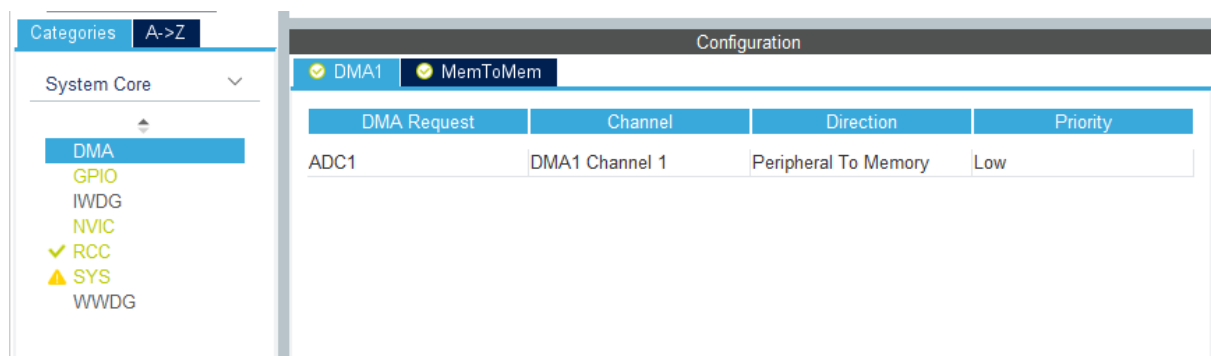
El DMA (Direct Memory Access) es una forma de leer o escribir en memoria sin utilizar la CPU de la que disponen ciertos elementos como el ADC. Esto significa que no estamos perdiendo tiempo de CPU en procesar las muestras, por lo que éstas se procesan en background y mientras podemos estar haciendo otra cosa. Además, esto permite que las muestras de una señal no se pierdan por tiempo de procesamiento de la CPU.

Una transferencia DMA consiste principalmente en copiar un bloque de memoria de un dispositivo a otro. En lugar de que la CPU inicie la transferencia, esta se lleva a cabo por el controlador DMA. Un ejemplo típico es mover un bloque de memoria desde una memoria externa a una interna más rápida. Tal operación no ocupa al procesador y, por ende, este puede efectuar otras tareas. Las transferencias DMA son esenciales para aumentar el rendimiento de aplicaciones que requieran muchos recursos.

Cabe destacar que aunque no se necesite a la CPU para la transacción de datos, sí se necesita el bus del sistema (tanto bus de datos como bus de direcciones), por lo que existen diferentes estrategias para regular su uso, permitiendo así que no quede totalmente acaparado por el controlador DMA.

En nuestro caso se utiliza el DMA para transferir los datos de los 6 canales de conversión que se encuentran en los registros del ADC, a la memoria RAM para luego tratarlos y analizarlos.

Configuración del DAM en el ioc:

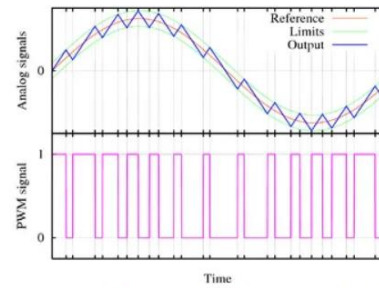


1.2.11 PULSE WIDTH MODULATOR (PWM):

La Regulación por Ancho de Pulso de un motor de CC está basada en el hecho de que si se recorta la CC de alimentación en forma de una onda cuadrada, la energía que recibe el motor disminuirá de manera proporcional a la relación entre la parte alta (habilita corriente) y baja (cero corriente) del ciclo de la onda cuadrada. Controlando esta relación se logra variar la velocidad del motor de una manera bastante aceptable.

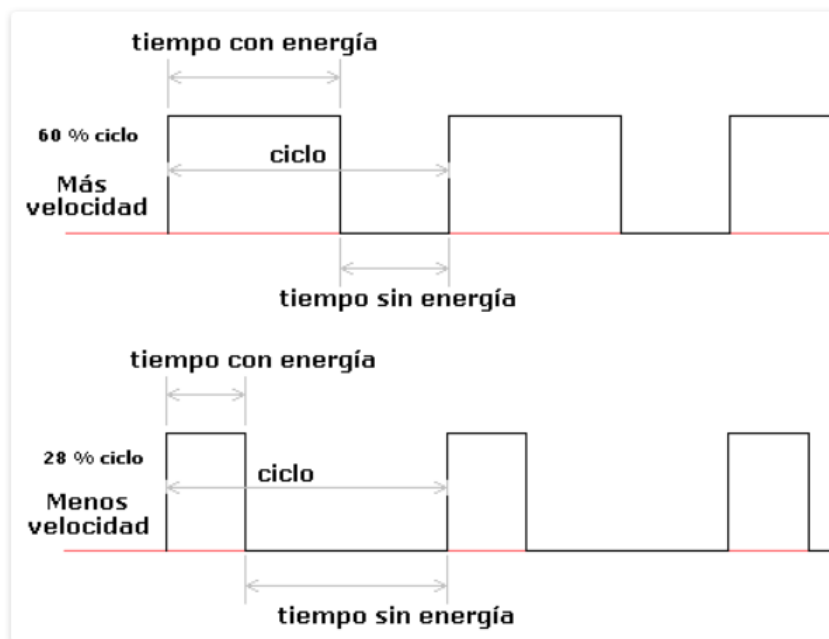
El PWM usa una onda de pulso rectangular, cuyo ancho de pulso es modulado, dando como resultado la variación del valor medio de la forma de onda. Si consideramos una forma de onda de pulso $f(t)$ con un valor bajo y_{min} , un valor alto y_{max} y un ciclo de trabajo, el valor medio de la forma de onda será.

$$\bar{y} = \frac{1}{T} \int_0^T f(t) dt.$$



Comparación de las señales analógicas y señales PWM

El duty cycle o ciclo de trabajo se expresa en tanto por ciento, representado que porcentaje de tiempo estará el interruptor encendido permitiendo el paso de energía.



La señal promedio es el producto de la tensión máxima y el valor Duty Cycle. La expresión para el cálculo es:

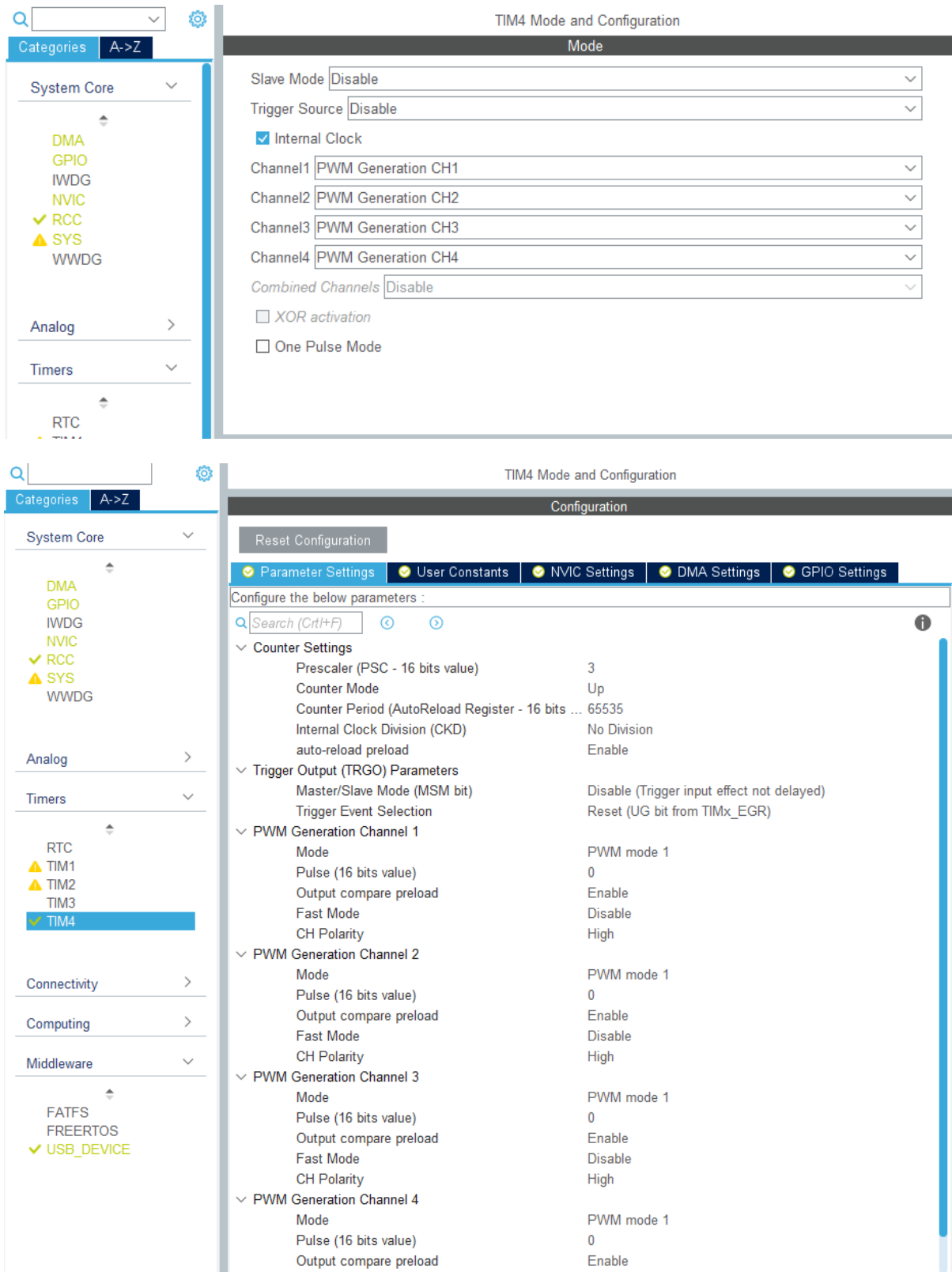
$$V_{medio} = (V_{cc+} - V_{cc-}) \cdot \frac{DutyCycle}{100}$$

De forma similar, tenemos que

$$DutyCycle = 100 \cdot \frac{V_{medio}}{(V_{cc+} - V_{cc-})}$$

Como nuestro auto posee dos motores de CC que giran en dos sentidos horario y antihorario, además de que cada uno gira a una velocidad determinada es indispensable el uso un PWM de la mano de un puente H. Todos los valores para la regulación van a ser calculados mediante el PID. El PWM cuenta con 4 canales dos por giro de cada motor.

Configuración del PWM en el ioc:



The image displays two screenshots of the STM32CubeMX software interface, showing the configuration of the TIM4 timer for PWM generation.

Top Screenshot: TIM4 Mode and Configuration - Mode

The left sidebar shows the project structure with the following components:

- System Core
 - DMA
 - GPIO
 - IWDG
 - NVIC
 - ✓ RCC
 - ⚠ SYS
 - WWDG
- Analog
- Timers
 - RTC

The main configuration area shows the TIM4 Mode and Configuration settings:

- Slave Mode: Disable
- Trigger Source: Disable
- ☒ Internal Clock
- Channel1: PWM Generation CH1
- Channel2: PWM Generation CH2
- Channel3: PWM Generation CH3
- Channel4: PWM Generation CH4
- Combined Channels: Disable
- ☐ XOR activation
- ☐ One Pulse Mode

Bottom Screenshot: TIM4 Mode and Configuration - Configuration

The left sidebar shows the project structure with the following components:

- System Core
 - DMA
 - GPIO
 - IWDG
 - NVIC
 - ✓ RCC
 - ⚠ SYS
 - WWDG
- Analog
- Timers
 - RTC
 - ⚠ TIM1
 - ⚠ TIM2
 - TIM3
 - ✓ TIM4
- Connectivity
- Computing
- Middleware
 - FATFS
 - FREERTOS
 - ✓ USB_DEVICE

The main configuration area shows the TIM4 Configuration settings:

- Reset Configuration
- Parameter Settings (selected)
- User Constants
- NVIC Settings
- DMA Settings
- GPIO Settings

Configure the below parameters :

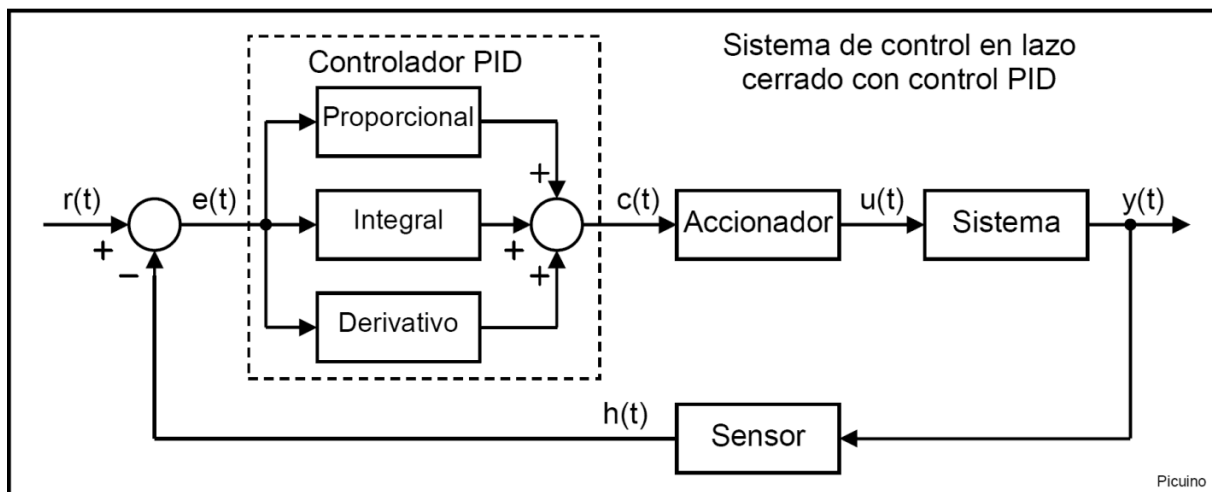
Search (Ctrl+F)

- Counter Settings
 - Prescaler (PSC - 16 bits value): 3
 - Counter Mode: Up
 - Counter Period (AutoReload Register - 16 bits ...): 65535
 - Internal Clock Division (CKD): No Division
 - auto-reload preload: Enable
- Trigger Output (TRGO) Parameters
 - Master/Slave Mode (MSM bit): Disable (Trigger input effect not delayed)
 - Trigger Event Selection: Reset (UG bit from TIMx_EGR)
- PWM Generation Channel 1
 - Mode: PWM mode 1
 - Pulse (16 bits value): 0
 - Output compare preload: Enable
 - Fast Mode: Disable
 - CH Polarity: High
- PWM Generation Channel 2
 - Mode: PWM mode 1
 - Pulse (16 bits value): 0
 - Output compare preload: Enable
 - Fast Mode: Disable
 - CH Polarity: High
- PWM Generation Channel 3
 - Mode: PWM mode 1
 - Pulse (16 bits value): 0
 - Output compare preload: Enable
 - Fast Mode: Disable
 - CH Polarity: High
- PWM Generation Channel 4
 - Mode: PWM mode 1
 - Pulse (16 bits value): 0
 - Output compare preload: Enable

TIM4 Mode and Configuration							
Configuration							
Reset Configuration							
<input checked="" type="checkbox"/> Parameter Settings <input checked="" type="checkbox"/> User Constants <input checked="" type="checkbox"/> NVIC Settings <input checked="" type="checkbox"/> DMA Settings <input checked="" type="checkbox"/> GPIO Settings							
Search Signals <input type="text" value="Search (Ctrl+F)"/> <input type="checkbox"/> Show only Modified Pins							
Pin Name	Signal on Pin	GPIO output...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PB6	TIM4_CH1	n/a	Alternate Fu...	n/a	Low		<input type="checkbox"/>
PB7	TIM4_CH2	n/a	Alternate Fu...	n/a	Low		<input type="checkbox"/>
PB8	TIM4_CH3	n/a	Alternate Fu...	n/a	Low		<input type="checkbox"/>
PB9	TIM4_CH4	n/a	Alternate Fu...	n/a	Low		<input type="checkbox"/>

1.2.12 Control proporcional, integral y derivativa PID:

Un controlador PID (controlador proporcional, integral y derivativo) es un mecanismo de control simultáneo por realimentación ampliamente usado en sistemas de control industrial. Este calcula la desviación o error entre un valor medido y un valor deseado.



El algoritmo del control PID consta de tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor proporcional depende del error actual, el integral depende de los errores pasados y el derivativo es una predicción de los errores futuros. La suma de estas tres acciones es usada para ajustar el proceso por medio de un elemento de control, como la posición de una válvula de control o la potencia suministrada a un calentador, en nuestro caso se regula la velocidad de giro de los motores.

1.2.12.1 Señal de referencia y señal de error

La señal $r(t)$ se denomina referencia e indica el estado que se desea conseguir en la salida del sistema $y(t)$ en nuestro caso la referencia es un valor de la suma ponderada de los valores de nuestros sensores cuando el auto se encuentra posicionado en el centro de la pista.

La entrada al controlador PID es la señal de error $e(t)$. Esta señal indica al controlador la diferencia que existe entre el estado que se quiere conseguir o referencia $r(t)$ y el estado real

del sistema medido por el sensor, señal $h(t)$. En nuestro caso la señal de error va a ser la diferencia entre el valor de la referencia de la suma ponderada calculado mediante un comando cuando nuestro auto se encuentra en el centro y el valor de dicha suma cuando el algoritmo de control PID comienza a correr.

Si la señal de error es grande, significa que el estado del sistema se encuentra lejos del estado de referencia deseado. Si por el contrario el error es pequeño, significa que el sistema ha alcanzado el estado deseado.

1.2.12.2 Proporcional:

La parte proporcional consiste en el producto entre la señal de error y la constante proporcional para lograr que el error en estado estacionario se aproxime a cero, pero en la mayoría de los casos, estos valores solo serán óptimos en una determinada porción del rango total de control, siendo distintos los valores óptimos para cada porción del rango. La parte proporcional no considera el tiempo, por lo tanto, la mejor manera de solucionar el error permanente y hacer que el sistema contenga alguna componente que tenga en cuenta la variación respecto al tiempo, es incluyendo y configurando las acciones integral y derivativa.

La fórmula del proporcional está dada por: $P_{sal} = K_p e(t)$

Donde:

K_p es la ganancia proporcional, valor de sintonización.

$e(t) = SP - PV(t)$ es el error (SP es el punto de establecimiento, y $PV(t)$ es la variable de proceso)

t es el tiempo o tiempo instantáneo (el actual),

Aumentar la acción proporcional K_p tiene los siguientes efectos:

Aumenta la velocidad de respuesta del sistema.

Disminuye el error del sistema en régimen permanente.

Aumenta la inestabilidad del sistema.

Los dos primeros efectos son positivos y deseables. El último efecto es negativo y hay que intentar minimizarlo. Por lo tanto al aumentar la acción proporcional existe un punto de equilibrio en el que se consigue suficiente rapidez de respuesta del sistema y reducción del error, sin que el sistema sea demasiado inestable. Aumentar la acción proporcional más allá de este punto producirá una inestabilidad indeseable. Reducir la acción proporcional, reducirá la velocidad de respuesta del sistema y aumentará su error permanente.

1.2.12.3 Derivativa:

La acción derivativa se manifiesta cuando hay un cambio en el valor absoluto del error; (si el error es constante, solamente actúan los modos proporcional e integral). El error es la desviación existente entre el punto de medida y el valor consigna, o "Set Point". La función de la acción derivativa es mantener el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce; de esta manera evita que el error se incremente. Se deriva con respecto al tiempo y se multiplica por una constante K_d y luego se suma a las señales anteriores (P+I). Es importante adaptar la respuesta de control a los cambios en el

sistema ya que una mayor derivativa corresponde a un cambio más rápido y el controlador puede responder acordemente.

La fórmula del derivativo está dada por: $D_{sal} = K_d \frac{de}{dt}$

Donde:

K_d es la ganancia derivativa, un parámetro de ajuste,
 t es el tiempo o tiempo instantáneo (el presente)

Aumentar la constante de control derivativa K_d tiene los siguientes efectos:

Aumenta la estabilidad del sistema controlado.

Disminuye un poco la velocidad del sistema.

El error en régimen permanente permanecerá igual.

1.2.12.4 Integral:

Esta acción de control como su nombre indica, calcula la integral de la señal de error $e(t)$. El modo de control Integral tiene como propósito disminuir y eliminar el error en estado estacionario, provocado por perturbaciones exteriores y los cuales no pueden ser corregidos por el control proporcional. El control integral actúa cuando hay una desviación entre la variable y el punto de consigna, integrando esta desviación en el tiempo y sumándola a la acción proporcional. El error es integrado, lo cual tiene la función de promediarlo o sumarlo por un período determinado; Luego es multiplicado por una constante K_i . Posteriormente, la respuesta integral es adicionada al modo Proporcional para formar el control P + I con el propósito de obtener una respuesta estable del sistema sin error estacionario.

La fórmula del integral está dada por: $I_{sal} = K_i \int_0^t e(\tau) d\tau$

Donde:

K_i es la ganancia integral, un parámetro de ajuste,
 τ es la variable de integración (toma en cuenta el valor desde el instante 0 hasta el instante actual t).

Aumentar la acción integral K_i tiene los siguientes efectos:

Disminuye el error del sistema en régimen permanente.

Aumenta la inestabilidad del sistema.

Aumenta un poco la velocidad del sistema.

1.2.12.5 Ecuación final de control:

Ecuación del controlador

La ecuación del control PID es la siguiente:

$$c(t) = K_p \cdot e(t) + K_i \cdot \int e(t)dt + K_d \cdot \frac{\partial e(t)}{\partial t}$$

Para:

- $c(t)$ = señal de control
- $e(t)$ = señal de error
- K_p, K_i, K_d = parámetros del controlador PID

1.2.12.6 Sintonización del control PID:

Si bien existen muchos modos de sintonización del PID como el método de Ziegler-Nichols nos faltan datos como para poder aplicar algún método que no sea el manual.

Después de ver las diferentes acciones proporcional, integral y derivativa de un control PID, se pueden aplicar unas reglas sencillas para sintonizar este controlador de forma manual.

1º - Acción Proporcional.

Se aumenta poco a poco la acción proporcional para disminuir el error (diferencia entre el estado deseado y el estado conseguido) y para aumentar la velocidad de respuesta.

Si se alcanza la respuesta deseada en velocidad y error, el PID ya está sintonizado.

Si el sistema se vuelve inestable antes de conseguir la respuesta deseada, se debe aumentar la acción derivativa.

2º - Acción Derivativa.

Si el sistema es demasiado inestable, se aumentará poco a poco la constante derivativa K_d para conseguir de nuevo estabilidad en la respuesta.

3º - Acción Integral.

En el caso de que el error del sistema sea mayor que el deseado, se aumentará la constante integral K_i hasta que el error se minimice con la rapidez deseada.

Si el sistema se vuelve inestable antes de conseguir la respuesta deseada, se debe aumentar la acción derivativa.

Con estas sencillas reglas es sencillo afinar poco a poco el controlador PID hasta conseguir la respuesta deseada.

1.3 Problemas en el desarrollo y mejoras del sistema:

1.3.1 Diseño de la mecánica:

1.3.1.1 Diseño del chasis:

Partiendo de la escases de recursos ya que las clases no son presenciales debido a la pandemia, lo que nos lleva a desarrollar el proyecto con materiales disponibles en nuestro hogar.

El primer problema que se presentó a la hora del diseño del chasis fue encontrar un balance entre las dimensiones del mismo y la disponibilidad de espacio para el montaje de la electrónica. Esto derivó en la realización de modificaciones sobre el chasis a medida que se arma la electrónica y se intenta montar sobre el mismo, dando como resultado un prototipo muy rudimentario, con unas dimensiones de 15cm*20cm.

La modificación a realizar sería armar una electrónica con menores dimensiones sin el uso de una protoboard. Luego de tener armada la electrónica diseñar un chasis de un material liviano en base a la misma.

1.3.1.2 Distribución del peso:

Una vez montada la electrónica en el chasis se presentó el problema de falta de torque en las ruedas debido a la mala distribución del peso. Como las dimensiones del auto eran amplias y el mismo cuenta con solo dos ruedas ubicadas en la parte delantera, quedando 10cm de chasis sin soporte, sumado a esto la mala ubicación de las baterías que se encontraban en los últimos 2cm de chasis provoca que el auto no tenga agarre al suelo. Para solucionar este inconveniente se ubicó las baterías cerca de las ruedas lo que aumentó de manera considerable el torque.

La mejora sugerida sería realizar la ubicación de la electrónica en base al peso de la misma y las dimensiones del auto, realizando todos los cálculos necesarios para obtener el mayor torque y rendimiento de los motores.

1.3.2 Diseño del Hardware:

1.3.2.1 Armado de la electrónica:

Por la escases de recursos el montaje y armado de la electrónica para los microcontroladores se realizó en una protoboard con conexión manual mediante cables monofilamento de longitudes considerables. Y el montaje del circuito de los sensores se realizó en una placa preperforada con salidas a la bluepill con cables monofilamento. Todo esto hace que el cableado sea excesivo añadiendo una gran cantidad de ruido a nuestro circuito. Además de requiere una revisión continua de las conexiones y riesgos de cortocircuitos si se desconecta algún cable.

La mejora sugerida sería realizar un diseño en una PCB.

1.3.2.2 Inestabilidad del sistema:

Debido a la inestabilidad de las conexiones por cables y el ruido que introducen las mismas a nuestro sistema provoca que el mismo no funcione de manera correcta y estable. Esto trae consigo la pérdida de la conexión wifi lo que requiere un reinicio del programa. Otro inconveniente es la falta de uniformidad en las lecturas de los sensores.

La mejora sugerida sería realizar un diseño en una PCB.

1.3.2.3 Armado del circuito de los sensores:

Nuevamente debido a la escasez de recursos el armado del circuito de los sensores se tuvo que realizar en base a los valores de resistencias disponibles tratando de buscar la mayor precisión y alcance de los mismos según la combinación de las resistencias para fototransistor y led emisor. La combinación que mejor se adaptó fue la de 100Ω en led emisor y 10kΩ en el fototransistor, logrando una distancia de censado precisa de 6cm y una aproximada de 10cm.

La mejora planteada sería la de encontrar una combinación que mejor se adapte a nuestros requerimientos una vez que se cuenten con las dimensiones finales del chasis. O buscar otra alternativa de sensores.

1.3.2.4 Distribución de los sensores:

Uno de los problemas más importantes fue la distribución de los sensores debido a que se realizó el montaje del circuito y ubicación de estos sin considerar las dimensiones del auto con respecto a la distancia de censado. Esto llevó a que los sensores delanteros que están ubicados a 45° den variaciones muy pequeñas en comparación con los sensores laterales cuando el auto se inclina respecto a una pared trayendo consigo grandes problemas a la hora de lograr el avance recto. La solución fue incrementar la ponderación de los sensores a 45° para tratar de igualar la respuesta de sensores laterales.

Otro punto a resalta es la ubicación de los sensores laterales que no tienen mucha influencia en la posición del auto solo tienen efecto cuando estamos llegando a las paredes frontales pero tampoco es de gran utilidad esa información.

La solución sería ubicar los sensores una vez este armado el chasis y se hallan hecho pruebas individuales de cada uno, buscando que los cambios de posición se vean reflejados con la misma intensidad en los sensores laterales como en los de 45°.

1.3.2.5 Efecto de la luz solar sobre los sensores

Otro problema que se presentó es el efecto de la luz solar sobre la medición de los sensores incrementado su valor considerablemente si nos encontramos en presencia de la misma. Esto trae el inconveniente de configurar variables del PID es decir, sintonizar el PID de acuerdo al ambiente donde lo vallamos a utilizar al auto o simplemente usarlo en un ambiente sin presencia de luz solar.

La mejora sería la utilización de otro tipo de sensores.

1.3.2.6 Autonomía

Otro problema que se presentó es el de la falta de información sobre la carga de las baterías. Como los motores poseen un consumo elevado el tiempo de funcionamiento de nuestro auto depende de la carga de las baterías y al no tener información sobre la tensión que están suministrando las mismas las podemos forzar y sobredescargar disminuyendo su vida útil además de provocar un funcionamiento incorrecto de nuestro sistema.

Además como el circuito requiere una conexión en serie de las baterías para luego bajar su tensión a 5V con una fuente regulable, de lo contrario el circuito de carga de estas requiere

una conexión en paralelo esto lleva a tener que cambiar el conexionado cada vez que se quiere cargar las mismas.

La solución sería incluir en nuestro programa un medidor de tensión para poder ver reflejado el valor que están suministrando nuestras baterías y hacer saltar una alarma cuando se llegue a un valor crítico. Además se podría incluir un circuito de carga para no tener que cambiar el conexionado de forma manual.

1.3.3 Diseño del programa en QT

En cuanto al diseño del programa de la PC en Qt no hubo mayores inconvenientes en cuanto a la programación, pero si en lo que respecta a la presentación de la interfaz grafica para que interactúe algún operario. Debido a la cantidad de información que debemos de manejar y presentar el espacio de los widgets nos quedan corto por lo que se requiere una buena gestión del espacio además de presentarle al operario una interfaz entendible y de fácil utilización.

Para mejorarlo se debería de dar el programa a un operario para que lo intente utilizar y evaluar los cambios en base a su opinión.

1.3.4 Diseño del PID

Como el PID depende de todos los factores antes mencionados se presentaron varias dificultades en el desarrollo que para eliminarlas de manera óptima requieren cambios en el diseño de la mecánica y electrónica del auto, pero aun con todos estos problemas se consiguió un control dentro de lo esperado.

El primer inconveniente fue la falta de torque en las ruedas lo que requería velocidades demasiado altas en el giro de las ruedas que no se podían lograr o disparaban los valores fuera de los limites. Esto se solucionó con el cambio en la ubicación de las baterías antes mencionado.

Otro inconveniente fue la falta de uniformidad de las medidas de los sensores ya sea por la presencia de la luz solar o por mala ubicación de los mismos. Esto se solucionó con la aplicación de una suma ponderada cambiando el valor de las ponderaciones hasta que el PID se ajuste.

Con estas dos fallas reconocidas se procedió a realizar la sintonización del PID que fue la parte de mayor dificultad. Se siguió el procedimiento de sintonización manual antes descripto realizando una serie de pruebas sin poder llegar a buen puerto. En un principio se cayó en el error de pensar que los sensores laterales eran los que nos iban a determinar el avance lineal que necesitábamos ya que nos media la distancia a la que se encuentran las paredes laterales por lo que se les dio mayor ponderación que a los demás sensores. Este pensamiento erróneo era el que hacía que el control PID fuera inservible ya que le estábamos dando mayor ponderación al sensor cuyo valor de medida era el de más influencia haciendo que los valores medidos por los sensores a 45° sean despreciables. Además un aspecto no considerado en las primeras pruebas era que los sensores laterales nos dan información sobre qué tan lejos están las paredes laterales pero no nos dicen nada de qué tan inclinado está el auto, es decir, no me dan información del ángulo de inclinación, información que si te suministran los sensores a 45°. Con estos aspectos considerados se fue alternando con diferentes valores de ponderación buscando que la influencia de los

sensores a 45° sobre la suma ponderada sea la misma que la de los sensores laterales. Estos valores de ponderación tienen que ser demasiado elevados debido a la mala ubicación de los sensores a 45° que se encuentran muy en el centro del auto.

Por último la imposibilidad de trabajar con operaciones en punto flotante en el microcontrolador hace que el control PID no sea tan preciso y se tenga que buscar trabajar con números enteros muy grandes para tener mayor precisión.

Como el control todavía está en fase de desarrollo y prueba pueden seguir surgiendo problemas que pudieron haber sido pasados por alto.

1.4 Conclusiones:

Son muchas las conclusiones a las que se puede llegar con un proyecto como el desarrollado ya que integra muchos conocimientos al tratarse del armado de un sistema mecatronico. Al tratar de hacer una conclusión sobre todo lo trabajado y aprendido seguramente algunos puntos se me pasen por alto. Uno de los aspectos importantes a resaltar antes de empezar la conclusión es la importancia de volcar todo lo aprendido durante el cursado a un proyecto practico que valla de la mano con el dictado de las clases, ya que esto hace que no te quedes solo con un concepto teórico sino que busques de aplicar eso a la realidad, aspecto fundamental que tenemos que desarrollar como futuros ingenieros en Mecatrónica.

Sin más que añadir como conclusión puedo decir que además de aprender sobre el funcionamiento y arquitectura de los microcontroladores. También pude aprender mucho sobre algo que considero fundamental a la hora de trabajar con microcontroladores que es de donde obtener información de los mismos y como interpretarla. Los primeros intentos de lectura y tratar de entender el datasheet de un microcontrolador me parecía muy complejo y prácticamente imposible, pero a medida que transcurrieron las clases y profundizábamos mas sobre el funcionamiento y la organización de la arquitectura de los microcontroladores esa interpretación fue costando menos hasta llegar al punto de entender el funcionamiento de los módulos y como adaptarlos a nuestros requerimientos mediante la programación de los mismos. Además sumado a estos aprender formas de programar que mejoran nuestro código.

Para cerrar la conclusión las clases me parecieron muy buenas y entretenidas cada una con un planteo diferente que además de aprender te motivan a investigar y ver cómo aplicar ese conocimiento al problema planteado. La ultima conclusión que creo que es la más importante como futuros ingenieros en mecatrónica es la de poder ver y experimentar la influencia que tiene la electrónica y mecánica a la hora de realizar un control por software mediante un algoritmo o programa, si bien se pueden obtener controles y llegar a resultados buenos lo ideal sería buscar un equilibrio entre la mecánica, electrónica y el desarrollo del software para no dejar todos los problemas en manos del programador.