


Rev.	Descripción de la Modificación			Fecha	Firma
00	Creación de documento			ABR'16	NOL
01	Modificaciones de contenido			MAY'9	NOL
02	Modificaciones de contenido y de formato			MAY'30	NOL
-	-				
PROYECTO N°	CANT.	OBSERVACIONES			PEDIDO EN PLANO N° POS.
		NOMBRE / NAME	FIRMA / INITIALS	FECHA / DATE	
	PROYECTADO	Nahuel Olguin nahuelolguin98@gmail.com	NOL	ABR'16	
	REVISADO				
	APROBADO				
<h1>ECOLASER 200</h1>					
<p>TITULO DEL DOCUMENTO:</p> <h2 style="text-align: center;">Máquina de estados del sistema</h2>					
DOCUMENTO N°			REVISIÓN	HOJA	

Contenido

1. Introducción	1
2. Estado 0: inicialización de variables	2
3. Maniobra homing y temporizador 5	5
3.1 Temporizador 5	7
3.2 Final maniobra homing	9
4. Estado 2: esperar tarea	10
4.1 Memoria	10
4.2 Esperar tarea y elección de próxima trayectoria	11
5. Estado 4: trayectoria en curso	14
6. Estado 7: Configuración parámetros y grabado de memoria	16
7. Estado 5: Generar trayectoria	21
8. Sistema de medición de temperatura	25
9. Estado 1: Emergencia	28
10. Anexo 1: Código en C y Bibliotecas	30
10.1 Tabla de variables	30
10.2 Pin Out	31
10.3 Biblioteca LedRGB	34
10.4 Biblioteca Laser	35
10.5 Biblioteca RTC	36
10.6 Biblioteca Trayectorias y biblioteca MY_FLASH	36
11. Anexo 2: Errores solucionados	40

11.1 Ruido en la señal de temperatura	40
11.2 Separación de memoria	40
11.3 Casteo y limitación de consigna.....	42
11.4 Muestro de sensores FC	43
11.5 Configurar bluetooth desde la núcleo	44
11.6 Falla del sensor de temperatura.....	45
11.7 Recuperacion por fallo en sensores FC	46
11.8 Solucion de Watchdog.....	47

1. Introducción

Este documento tiene como objetivo explicar el funcionamiento de una maquina de estados generada en stateflow con el objetivo de simular el programa cargado en el microcontrolador del equipo ecolaser 200.

Para facilitar el entendimiento de lo desarrollado a lo largo del documento es recomendable contar con la documentacion correspondiente al equipo ecolaser 200. Algunos documentos importantes son “Programación de Ecolaser 200”, “Comandos UART”, “Pinout-MapaMemoria_V2”, entre otros.

Otra herramienta fundamental para facilitar el entendimiento de este documento es contar con el software de simulacion y el archivo del modelo de simulink “Maq_Estados.slx” generado en la version de Matlab R2023a (recomendamos trabajar con esta misma version para evitar problemas de compatibilidad).

En la figura 1 se observa una vista general de todos los estados, subestados y transiciones que componen la máquina de estados que representa al sistema ecolaser 200. Cada bloque corresponde a un estado en particular donde se puede apreciar su nombre y su numero asignado, mientras que las flechas correspondan a las transiciones entre los mismos, a lo largo del documento se analizara a profundidad cada uno de ellos.

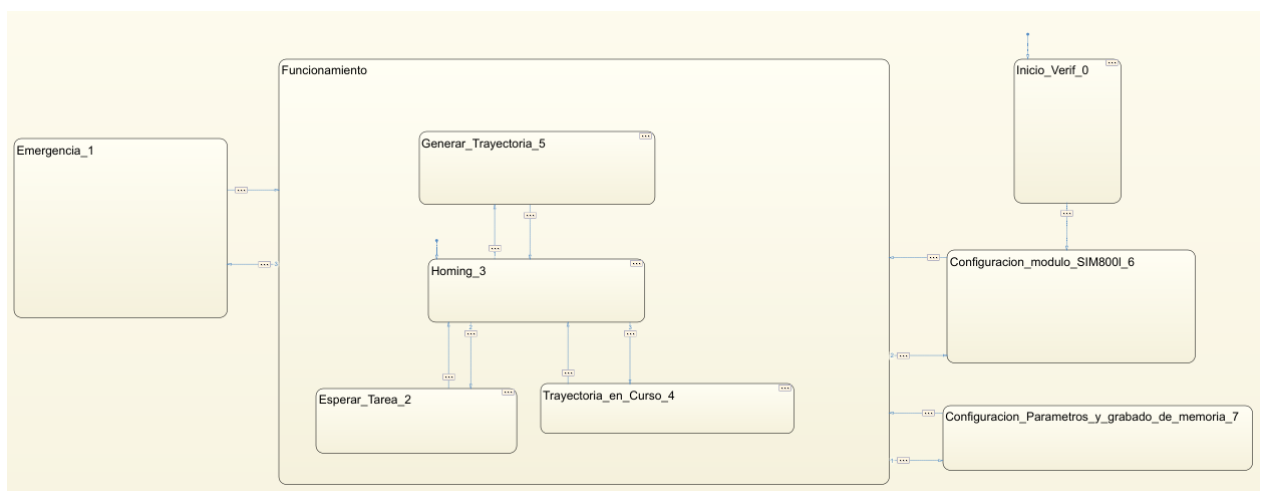


Figura 1: Vista general de máquina de estados

2. Estado 0: inicialización de variables

Se comienza con el estado cero donde se inicializan todas las variables que se utilizan en la máquina de estados a lo largo de todo el desarrollo, como se observa en la figura 2.

Como se dijo en la introducción esta máquina de estados programada en Matlab está desarrollada de forma tal que facilita el entendimiento del código original en lenguaje C sin ser necesariamente igual. Por ejemplo, en el estado cero del código original solo se definen las posiciones de los motores, y las variables “estado” y “estado_anterior”, mientras que el resto se van definiendo en diferentes bibliotecas y funciones; por otro lado en matlab se decide declarar todas las variables juntas para hacer más fácil la lectura de las mismas.

Además, algunas variables e implementación de funciones difieren en ambos lenguajes (para ver la equivalencia entre variables leer anexo seccion 10.1).

```

Inicio_Verif_0
%Parametros Timer5
entry;
Pulso_0 = 0;
Pulso_1 = 0;
dir0=0;
dir1=0;
velocidad=70; %consigna velocidad
TimerPeriod_Min = 200-1; %velocidad maxima
%Parametros motor
MicroStep=1; %Segun el paso que se use se elige el denominador: 1/1 1/2
PosicionesPorPulso = 2/MicroStep;
PasosPorVuelta = 200 * MicroStep;
r=60; %Relacion de transmision, para los dos ejes es la misma
%Homing
ValorInicial=24000;
pos_motor0=ValorInicial;
pos_motor1=ValorInicial;
consigna_0=ValorInicial;
consigna_1=ValorInicial;
%Espera
TimeSlot=0;
trayectoria = 0;
time=zeros(3,1);
time_ant=zeros(3,1);
%Trayectoria
tiempo_uso=zeros(3,2);
tiempo_laser=zeros(3,1);
%Perifericos
led="";
laser="OFF";
FANS="OFF";
%Comunicacion
rx_buffer="";
contador_SIM800 = zeros(2,1);
%Variables para termocupla
Flag_hora = 0;
Temperatura_Emergencia=55;
grados_variable=8; %valor absoluto que puede variar la medicion de temp
contador_mediciones = 0;
contador_mediciones_max=6;
temp_desconetado_max=150; %valor de temperatura que significa sensor desconecta
temp_desconetado_min=-9; %valor de temperatura que significa sensor desconecta
estado_sensor = 0;
mediciones_desc=12; %midiendo cada 5 seg (TIM2) 12 veces da 60 seg
contador_sensor_desc = 0;
Termocupla = lectura_temperatura; %calcula el promedio de las mediciones
Termocupla_ant = Termocupla;
Temperatura_max = Termocupla;
%Variables locales
estado_anterior=0;
estado=6;
    
```

Figura 2: Estado 0 declaración de variables

En la tabla 1 se explica brevemente la finalidad de cada una de las variables que hace uso la máquina de estados.

Nombre	Descripción
Parámetros del Timer 5	
Pulso_0/1	El equipo cuenta con dos motores los cuales serán nombrados como motor 0 y motor 1 a lo largo del desarrollo, estos motores son del tipo PaP (paso a paso) cuyo avance se controla con una señal eléctrica que alterna entre los valores 0 y 1, cuando la señal cambia de 0 a 1 entonces el motor avanzará un paso. La representación de esta señal eléctrica se almacena en estas dos variables según el motor correspondiente, “Pulso_0” para controlar el avance del motor 0 y “Pulso_1” para controlar el avance del motor 1.
dir0/1	Además de las señales para controlar el avance de los motores, también es necesario contar con una señal que determina en qué dirección avanzará el motor. Para esto son necesarias las variables “dir0/1” para cada motor, donde el valor 0 representa el sentido antihorario y el valor 1 indica sentido horario. Para el caso del motor uno y su respectiva variable de dirección “dir1”, el valor 0 baja el láser y el valor 1 sube el láser.
velocidad	Esta variable almacena la consigna deseada de velocidad de los motores que puede ir desde 0% hasta 100% , cuyo valor es utilizado para definir el periodo de tiempo con el que la señal “Pulso_0/1” cambia su valor, y por ende la velocidad a la que se moverán los motores.
TimerPeriod_Min	Es una constante que limita el periodo mínimo del temporizador cinco. El temporizador cinco es el encargado de realizar la medición de tiempo en la que la señal “Pulso_0/1” alterna su valor, por lo que fijar un valor de “TimerPeriod_Min” es equivalente a fijar un valor de velocidad máxima para los motores.
Parámetros motor	
MicroStep	Indica si el motor avanzará un paso completo o medio paso cuando se le envía un pulso.
PosicionesPorPulso	Indica cuantas posiciones se mueve el motor por pulso.
PasosPorVuelta	Indica cuantas posiciones se debe avanzar para realizar una vuelta completa.
r	Relación de transmisión.
Parámetros Homing	
ValorInicial	Almacena la posición inicial que se les asigna a los motores antes de realizar la maniobra de homing. Se toma un valor grande para evitar caer en valores negativos de posición.
pos_motor0/1	Almacena el valor de las posiciones de los motores.
consigna_0/1	Almacena las consignas de posiciones deseadas.
Parámetros Espera	
TimeSlot	El sistema cuenta con 3 intervalos de tiempo a lo largo del día que no se pueden superponer. Esta variable almacena un valor entre 1 y 3 que representa en cuál de los 3 intervalos de tiempo se encuentra el sistema actualmente.
trayectoria	El sistema cuenta con 10 trayectorias disponibles para realizar. Esta variable almacena un valor entre 1 y 10 que representa cual trayectoria está realizando el sistema actualmente.
time	Es un vector de 3 elementos que almacena el tiempo en horas, minutos y segundos (para profundizar más sobre el tema leer sección 10.5 del anexo).

time_ant	Es un vector con la misma estructura que el vector “time”, el cual almacena un tiempo anterior.
Trayectoria	
tiempo_uso	Es un vector de 3x2. En las filas se almacenan los tres datos de tiempo (horas, minutos y segundos). En la primera columna se almacenan los datos de tiempo cuando comienza la trayectoria y en la segunda cuando finaliza la trayectoria.
tiempo_laser	Almacena las horas, minutos y segundos que estuvo encendido el láser.
Periféricos	
led	Es una variable que almacena el color que tomara el led RGB (para profundizar más sobre el tema leer sección 10.3 del anexo).
laser	Es una variable que indica si el láser esta encendido o apagado (para profundizar más sobre el tema leer sección 10.4 del anexo).
FANS	Es una variable que indica si los ventiladores están encendidos o apagados.
Comunicación	
rx_buffer	Variable que almacena el mensaje recibido a través de la UART
contador_SIM800	Es un contador para limitar la espera en la recepción de un mensaje
Variables de medición temperatura	
Flag_hora	Variable que indica si es momento de tomar lectura de la temperatura. El temporizador dos cada 12,5 seg asigna a esta variable el valor 1 para realizar dicha lectura.
Temperatura_Emergencia	Almacena el valor máximo de temperatura que no se debe superar por el sistema, en caso de que se supere el sistema entra en estado de emergencia.
grados_variable	Son los grados máximos que puede variar de una lectura a otra sin considerarse una lectura errónea. Es decir, si la temperatura varía de una lectura a otra una cantidad mayor a “grados_variables”, entonces esa lectura se considera errónea y se descarta.
contador_mediciones	Almacena la cantidad de mediciones descartadas.
contador_mediciones_max	Variable que almacena la cantidad máxima de mediciones sucesivas que pueden ser descartadas debido a que se consideran erróneas.
temp_desconectado_max	Valor de temperatura que significa sensor desconectado
temp_desconectado_min	Valor de temperatura que significa sensor desconectado
estado_sensor	Hace referencia a si el sensor está desconectado o conectado.
contador_sensor_desc	Almacena el tiempo que el sensor permanece desconectado
mediciones_desc	Corresponde al número máximo de mediciones descartadas debido a que el sensor esta desconectado.
Termocupla	Almacena la lectura de temperatura actual
Termocupla_ant	Almacena la última lectura
Temperatura_max	Almacena la temperatura máxima registrada hasta el momento
Variables locales	
estado_anterior	Almacena el estado en el que se estuvo anteriormente.
estado	Almacena el estado en el que se está actualmente.

Tabla 1: Descripción de variables

3. Maniobra homing y temporizador 5

Al ingresar al estado 3 el cual corresponde a la maniobra homing, como se observa en la figura 3 lo primero que se realiza es un ajuste de variables tales como: periféricos, subestados, estados de motores, velocidad y variables locales. Algo importante a destacar es que se resetean las variables que almacenan la posición de los motores “pos_motor0/1”, esto se hace porque queremos calibrar el sistema nuevamente a la posición de homing sin importar la posición en la que estamos actualmente.

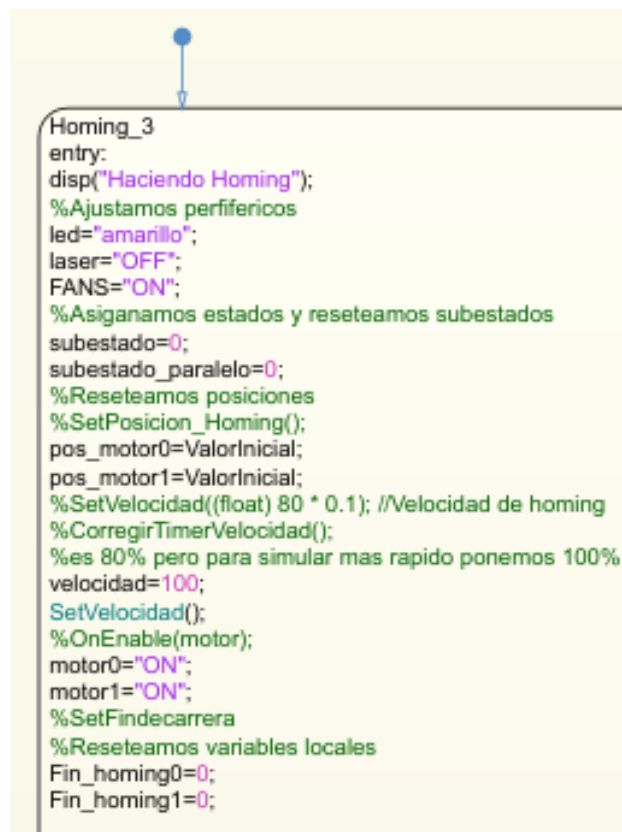


Figura 3: Configuración de variables para realizar homing

Dentro del estado homing tenemos dos subestados que se ejecutan en paralelo, cada uno de estos es la maniobra correspondiente a cada articulación el sistema, como se observa en la figura 4. La maniobra realizada es la misma en ambas articulaciones y se compone de distintos movimientos de retroceso y avance.

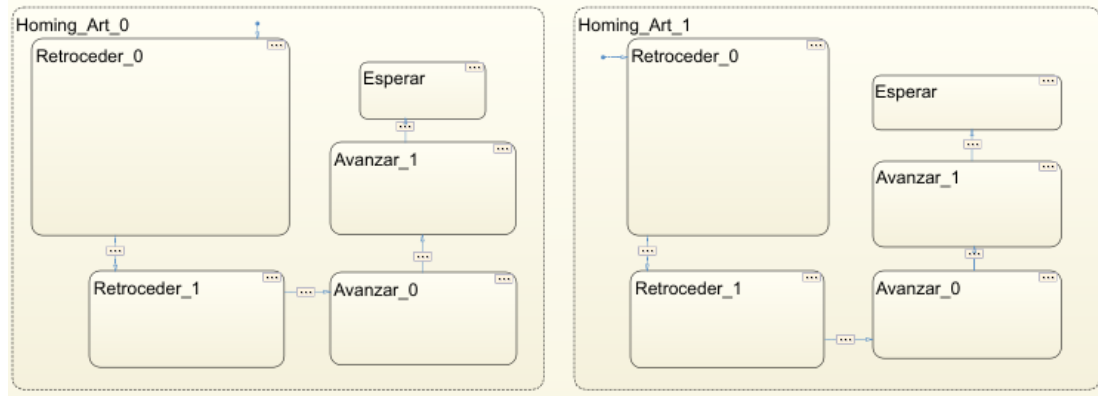


Figura 4: Maniobras de cada articulación

Se procede a explicar detalladamente en qué consisten las maniobras del proceso de Homing observadas en la figura 5:

1. El primer paso consiste en retroceder hasta hacer contacto con el sensor final de carrera, esto se consigue llevando la variable de consigna a valores más negativos en cada iteración. En caso de que el sensor falle, la posición del motor llegará a ser menor que una constante llamada “Posicion_Home_Max”, cuando esto pasa se da por finalizada la maniobra homing y el sistema pasa a un estado de emergencia.
2. El segundo paso consiste en hacer un leve retroceso para asegurar un buen contacto con el final de carrera, se retrocede una distancia que está determinada por un valor constante “OFFSET_FC”.
3. Una vez finalizadas las dos etapas de retroceso se comienza a avanzar hasta dejar de tener contacto con el final de carrera.
4. Cuando se deja de hacer contacto con el final de carrera se sigue avanzando hasta alcanzar la posición programada como homing, la cual está dada por “OFFSET_FC”.
5. Finalmente se espera a que la otra articulación haya finalizado su respectiva maniobra de homing.

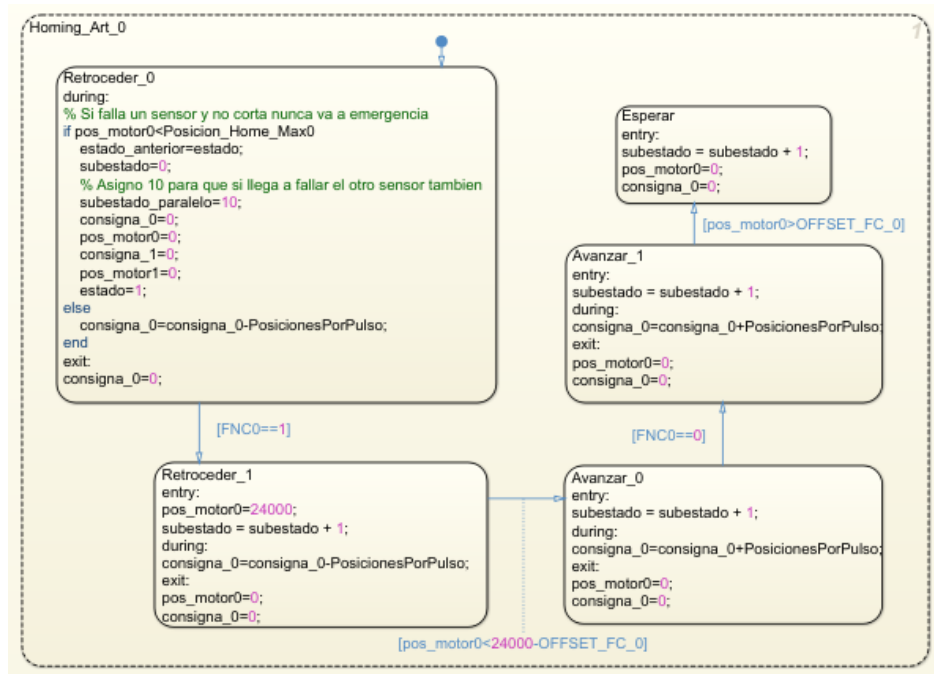


Figura 5: Maniobras de homing en la articulación 0

3.1 Temporizador 5

Se ha dicho que para avanzar o retroceder solo se debe modificar el valor de “consigna_0/1”, el resto del proceso es realizado por las interrupciones del temporizador 5. Se observa en la figura 6 que el timer 5 cuenta con 2 subestados los cuales son “Generado_pulsos” y “Sample_FNC”.

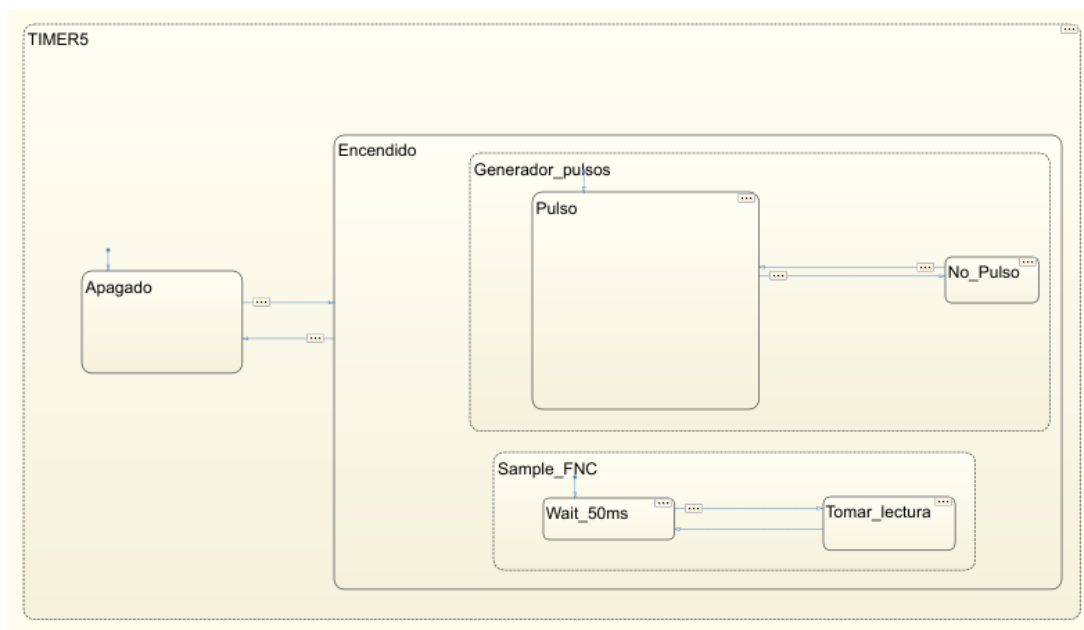


Figura 6: Subestados del Timer5

El generador de pulsos es el encargado de alternar la señal eléctrica entre 0 y 1 para mover los motores paso a paso. Vemos en la figura 7 que cada cierto tiempo “Periodo_tim5” las variables “Pulso_0/1” van alternando su valor entre 0 y 1 cuando la posición de los motores no coincide con la consigna. En caso de que las posiciones coincidan con su respectiva consigna entonces las señales de pulso se mantienen en cero.

Además se observa que dependiendo de que si la consigna es mayor o menor a la posición actual del motor, las variables “dir0/1” tomaran valores diferentes haciendo que la rotación sea en sentido horario o antihorario.

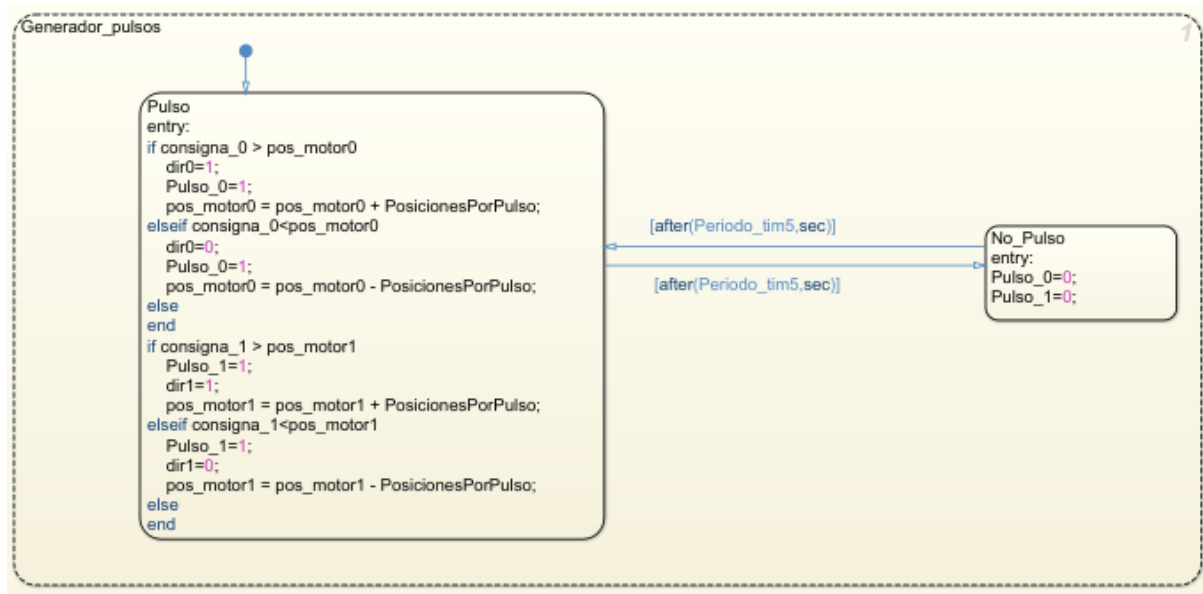


Figura 7: Generación de pulsos

El periodo con el que el pulso alterna su valor es modificado mediante la consigna “velocidad” la cual puede tomar valores entre 0 y 100%. Como se observa en la figura 3, cuando la variable “velocidad” es modificada se procede a llamar a la función SetVelocidad() la cual, como se observa en la figura 8, realiza los cálculos para modificar el valor máximo del conteo del temporizador cinco “TimerPeriod_actual” y posteriormente hace uso del prescaler “htim5_Init_Prescaler=840” y la frecuencia del reloj del timer correspondiente a las placas STM32 “Frecuencia_reloj=84 MHz”, para obtener el periodo de tiempo en el que se generan los pulsos “Periodo_tim5”.

```
function SetVelocidad()
    if velocidad >= 90
        TimerPeriod_actual = TimerPeriod_Min * (100-90)/10;
    else
        TimerPeriod_actual = TimerPeriod_Min * (100-velocidad)/10;
    end
    %Corregir Timer velocidad
    Periodo_tim5=TimerPeriod_actual*(htim5_Init_Prescaler+1)/Frecuencia_reloj;
end
```

Figura 8: Variación en la velocidad de pulsos

Por otro lado, en la figura 9, tenemos el subestado “Sample_FNC”, el cual se encarga de muestrear el estado de los sensores finales de carrera cada 0,05 segundos.

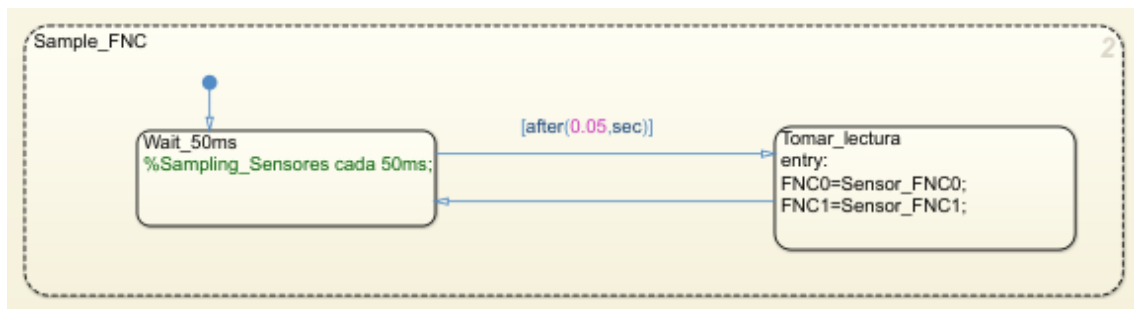


Figura 9: Muestreo de los fines de carrera

3.2 Final maniobra homing

Una vez que la maniobra homing fue completada en las dos articulaciones, se procede a ajustar variables y periféricos de acuerdo al siguiente estado en el cual el sistema entrará, como se observa en la figura 10.

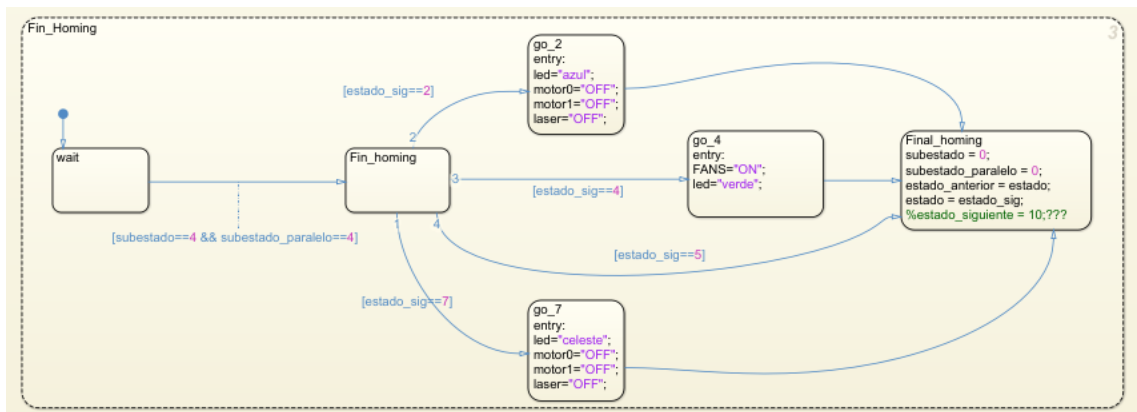


Figura 10: Ajuste de parámetros al finalizar homing

4. Estado 2: esperar tarea

En este estado el objetivo es determinar cuál será la siguiente trayectoria a realizar.

Pero antes se procede a explicar cómo funciona la memoria correspondiente a esta máquina de estados.

4.1 Memoria

Si bien en el sistema real, la memoria puede representarse como un conjunto de posiciones de almacenamiento, donde cada posición puede contener un valor y tiene una dirección única para identificarla (para profundizar más sobre el tema leer anexo sección 10.6). En este sistema optaremos por una representación más simple para facilitar el entendimiento de estados posteriores.

Se representa a la memoria como un bloque en paralelo a la máquina de estados principal, que almacena variables y vectores necesarios para el correcto funcionamiento del sistema, como se observa en la figura 11.

```

Memoria
entry:
%Espacio donde se almacenan todas las variables de los parametros
%Tres columnas para 3 timeslots. Las filas son hora inicio, minutos inicio, hora fin y minutos fin.
TimeSlots=zeros(4,3);
%Matriz de intervalos activados para cada trayectoria
%filas->trayectorias. columnas->activacion de cada uno de los 3 Timeslots(1 activado, 0 desactivado)
trayectorias_intervalos = zeros(10,3);
%filas->trayectorias (1 a 10 motor0 y 11 a 20 motor1). columnas ->100 puntos consigna de motores
trayectorias_puntos = zeros(20,100);
%filas->velocidad de cada trayectoria
Velocidades=zeros(10,1);
%filas->min de espera de cada trayectorias.
Esperas=zeros(10,1);
%A continuacion se muestra en comentarios como aplicar una configuracion inicial de la memoria
%Conf inicial intervalos de Timers (se pueden modificar para hacer pruebas)
%1
TimeSlots(1,1)=0;
TimeSlots(2,1)=0;
TimeSlots(3,1)=7;
TimeSlots(4,1)=59;
%2
TimeSlots(1,2)=8;
TimeSlots(2,2)=0;
TimeSlots(3,2)=15;
TimeSlots(4,2)=59;
%3
TimeSlots(1,3)=16;
TimeSlots(2,3)=0;
TimeSlots(3,3)=23;
TimeSlots(4,3)=59;
%Parametros trayectoria
%trayectoria 6
trayectorias_intervalos(6,1:3)=1;
Velocidades(6)=100;
Esperas(6)=0;
trayectorias_puntos(6,1)=100;
trayectorias_puntos(6,2)=0;
trayectorias_puntos(6,3)=-100;
trayectorias_puntos(6,4:100)=trayectorias_puntos(6,3);
trayectorias_puntos(16,1)=0;
trayectorias_puntos(16,2)=100;
trayectorias_puntos(16,3)=0;
trayectorias_puntos(16,4:100)=trayectorias_puntos(16,3);
during:
%Reloj: horas, min y segundos
time(3)=horas;
time(2)=minutos;
time(1)=segundos;
    
```

Figura 11: Variables almacenadas en la memoria

Se procede a explicar brevemente el funcionamiento de estos vectores:

“TimeSlots”: es un vector 4x3, donde las 3 columnas corresponden a los tres intervalos de tiempo disponibles y las 4 filas corresponden a la hora de inicio, minutos inicio, hora final y minutos final. Como ejemplo se observa en la figura 11 que se inicializa el intervalo del TimeSlot 1 con hora de inicio 00:00 y hora de finalización 07:59, de forma análoga son inicializados los otros dos intervalos.

“trayectorias_intervalos”: es un vector 10x3, donde las filas corresponden a las diez trayectorias disponibles y las columnas corresponden a la activación de dichas trayectoria en cada uno de los tres intervalos de tiempo. En la figura 11 se observa que se activa la trayectoria 6 en los tres intervalos de tiempo.

“trayectorias_puntos”: es un vector 20x100 cuya función es almacenar todos los puntos o posiciones que conforman las trayectorias de ambos motores, las primeras 10 filas corresponden al motor cero y las otras 10 corresponden al motor uno. Las columnas corresponden a las posiciones que deben ir siguiendo los motores para realizar una determinada trayectoria. En la figura 11, se observa que para la trayectoria seis los puntos que debe alcanzar el motor cero son {100, 0,-100} y los puntos que debe alcanzar el motor uno son {0, 100, 0}.

“Velocidades”: es un vector 10x1 que almacena las velocidades correspondientes a cada una de las diez trayectorias.

“Esperas”: es un vector 10x1 que almacena los tiempos de espera de cada una de las diez trayectorias.

“time”: es un vector que almacena la hora, minutos y segundos actuales.

4.2 Esperar tarea y elección de próxima trayectoria

Lo primero que se realiza es determinar en qué intervalo de tiempo o “TimeSlot” se encuentra el sistema según el horario actual. Para conseguir esto, en la figura 12 se observa

que primero se calculan los minutos totales que han transcurrido durante el día y se almacenan en la variable “minutos_total_act”. Luego se calculan los minutos mínimos y máximos de cada intervalo y evaluamos en cuál de los tres intervalos queda la variable “minutos_total_act”, de esta forma obtenemos en que intervalo de tiempo o TimeSlot nos encontramos actualmente.

```
Esperar_Tarea_2
entry:
disp("Buscando nueva trayectoria");
during:
%Determinar en cual TimeSlot estamos de acuerdo a la hora
minutos_total_act=time(3)*60+time(2);
for j=1:3
    minutos_total_min=TimeSlots(1,j)*60+TimeSlots(2,j);
    minutos_total_max=TimeSlots(3,j)*60+TimeSlots(4,j);
    if minutos_total_act>=minutos_total_min && minutos_total_act<=minutos_total_max
        TimeSlot = j;
    end
end
end
```

Figura 12: determinación del TimeSlot actual

Una vez determinado el “TimeSlot”, en la figura 13 se muestra que lo siguiente a realizar es esperar a que transcurran los minutos de espera de la trayectoria que finalizó previamente. En base a la trayectoria que finalizó se obtienen los minutos de espera de la misma “Min_espera” y se calcula cuantos son los segundos que se deben esperar a partir del momento en que finalizo la última trayectoria, cuyo tiempo fue almacenado en el vector “time_ant”. Una vez que los segundos actuales sean mayores que los segundos que se tenían que esperar, se avanza al siguiente paso.

```
TimeSlots_detectado
%Consulta_Activacion_Trayectoria
%Revisar tiempo de espera para prox trayectoria
entry:
i=trayectoria;
if(trayectoria == 0)
    Min_espera= 0
else
    Min_espera=Esperas(trayectoria)
end
segundos_espera = time_ant(3) * 3600 + time_ant(2) * 60 + time_ant(1) +Min_espera * 60;
during:
segundos_actuales = time(3) * 3600 + time(2) * 60 + time(1);
if (segundos_actuales >= segundos_espera)
```

Figura 13: Control del tiempo de espera

Ahora en la figura 14 se evalúa la siguiente trayectoria (la última trayectoria es la diez, por lo que si se supera este número se repite el proceso desde la primer trayectoria), se analiza el vector “trayectorias_intervalos”, para evaluar si la trayectoria “i” que se está evaluando actualmente está activa para el intervalo de tiempo actual “TimeSlot”.

Si se verifica que la trayectoria “i” está activada para el intervalo de tiempo “TimeSlot”, entonces el elemento “trayectorias_intervalos (i, TimeSlot)” será igual a uno. Por lo tanto ha sido encontrada una trayectoria a realizar y se avanza al siguiente estado, el cual es la realización de la maniobra homing nuevamente. En caso de que no se encuentre una trayectoria disponible, se repetirá el proceso con la siguiente trayectoria hasta encontrar una que esté activada para el “TimeSlot” actual.

```
if (segundos_actuales >= segundos_espera)
    %Se repite hasta obtener alguna trayectoria que se active en este TimeSlot
    %GetTrayectoriaSiguiente(int TimeSlot, int trayect_actual)
    i = i + 1;
    %En caso de que llegue hasta el final sin encontrar trayectoria activa empiezo de nuevo
    if i > 10
        i = 1;
    end
    if(trayectorias_intervalos(i,TimeSlot)==1)
        trayectoria=i %Se ha encontrado una trayectoria activa
        %Salimos de este estado
        estado_sig=4;
        estado=3;
    else
        %No se encuentra trayectoria disponible por el momento y se pasa a la proxima iteracion
        trayectoria= 0;
    end
end
```

Figura 14: Proceso de selección de trayectoria

5. Estado 4: trayectoria en curso

En este estado se lleva a cabo la trayectoria encontrada previamente. Lo primero que se hace es modificar el periodo del “Timer5” enviándole el valor de la consigna de velocidad de la trayectoria en cuestión (figura 15).

Luego se actualiza las consignas de ambos motores en base a la posición del primer punto que se debe alcanzar, teniendo en cuenta no superar los valores límites “Posicion_Max0/1”. Observar que una vez alcanzado el primer punto “ultimo_punto=1” de la trayectoria es donde se da inicio a la trayectoria por lo que se enciende el láser y se guarda el tiempo en el que se encendió.

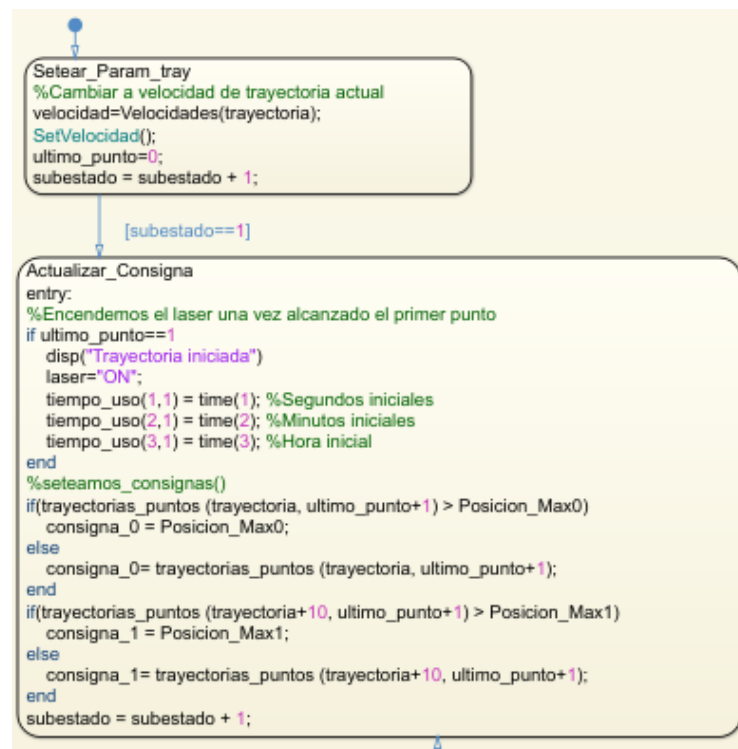


Figura 15: Ajuste de variables para iniciar trayectoria

Una vez actualizada la consigna se debe esperar hasta que el temporizador “Timer5” mueva los motores hasta alcanzar tales consignas (figura 16). Cuando las consignas son alcanzadas se pasa al siguiente punto y si todavía no llegamos al punto cien “ultimo_punto<100”, entonces se vuelve a actualizar la consigna como se hizo en la figura 14.

```

Moviendo
%Se evalúa si ya se alcanzó la consigna
if pos_motor0==consigna_0 && pos_motor1==consigna_1
    %modificación adaptada a matlab
    ultimo_punto = ultimo_punto + 1;
    if ultimo_punto<100
        subestado = subestado - 1;
    end
end
end
    
```

Figura 16: Etapa de movimiento y evaluación de finalización

Cuando se alcanza el punto número cien, el cual es el último punto de las trayectorias, se observa en la figura 17 que se apaga el láser y se guarda el tiempo en el que se apagó el láser. Posteriormente se pasa el tiempo de encendido y apagado del láser a segundos, se obtiene la diferencia y se lo vuelve a pasar a horas, minutos y segundos para obtener el tiempo que estuvo encendido el láser (observar que hay un caso donde se contempla el cambio de día). Finalmente se realiza la maniobra de homing para ir buscar la próxima trayectoria a realizar.

```

Trayectoria_en_Curso_4
entry:
disp("Preparandose para iniciar trayectoria");
during:
if ultimo_punto>99
    laser="OFF";
    %time_ant(2) = time(2); %Hora
    %time_ant(1) = time(1); %Minutos
    %time_ant(0) = time(0); %Segundos
    tiempo_uso(1,2) = time(1); %Segundos finales
    tiempo_uso(2,2) = time(2); %Minutos finales
    tiempo_uso(3,2) = time(3); %Hora finales

    segundos_iniciales = tiempo_uso(3,1)* 3600 + tiempo_uso(2,1) * 60 + tiempo_uso(1,1);
    segundos_finales = tiempo_uso(3,2) * 3600 + tiempo_uso(2,2) * 60 + tiempo_uso(1,2);

    auxiliar_tiempo = segundos_finales - segundos_iniciales;
    %uso_laser();
    if (auxiliar_tiempo >= 0)
        tiempo_laser(3) = auxiliar_tiempo / 3600; %guardo solo las horas y descarto el resto
        tiempo_laser(2) = (auxiliar_tiempo - tiempo_laser(3) * 3600) / 60; %guardo solo los minutos y descarto el resto
        tiempo_laser(1) = (auxiliar_tiempo - tiempo_laser(3) * 3600 - tiempo_laser(2) * 60); %guardo solo los seg
    else %para el caso que la rutina haya comenzado el día anterior justo
        auxiliar_tiempo = 24 * 3600 - segundos_iniciales + segundos_finales;
        tiempo_laser(3) = auxiliar_tiempo / 3600; %guardo solo las horas y descarto el resto
        tiempo_laser(2) = (auxiliar_tiempo - tiempo_laser(3) * 3600) / 60; %guardo solo los minutos y descarto el resto
        tiempo_laser(1) = (auxiliar_tiempo - tiempo_laser(3) * 3600 - tiempo_laser(2) * 60); %guardo solo los seg
    end
    %end

    %GrabarDataLogLaser(tiempo_laser). (Terminar)

    %Reiniciamos el contador de tiempo para el laser
    tiempo_uso=zeros(3,2);

    estado_sig=2;
    disp("Trayectoria finalizada");
    estado=3;
end
    
```

Figura 17: Procedimiento final del estado

6. Estado 7: Configuración parámetros y grabado de memoria

Este estado contiene varios subestados con funciones específicas cuyas transiciones dependen de los comandos recibidos a través de la UART. En la figura 18 se observan los subestados que componen este estado.

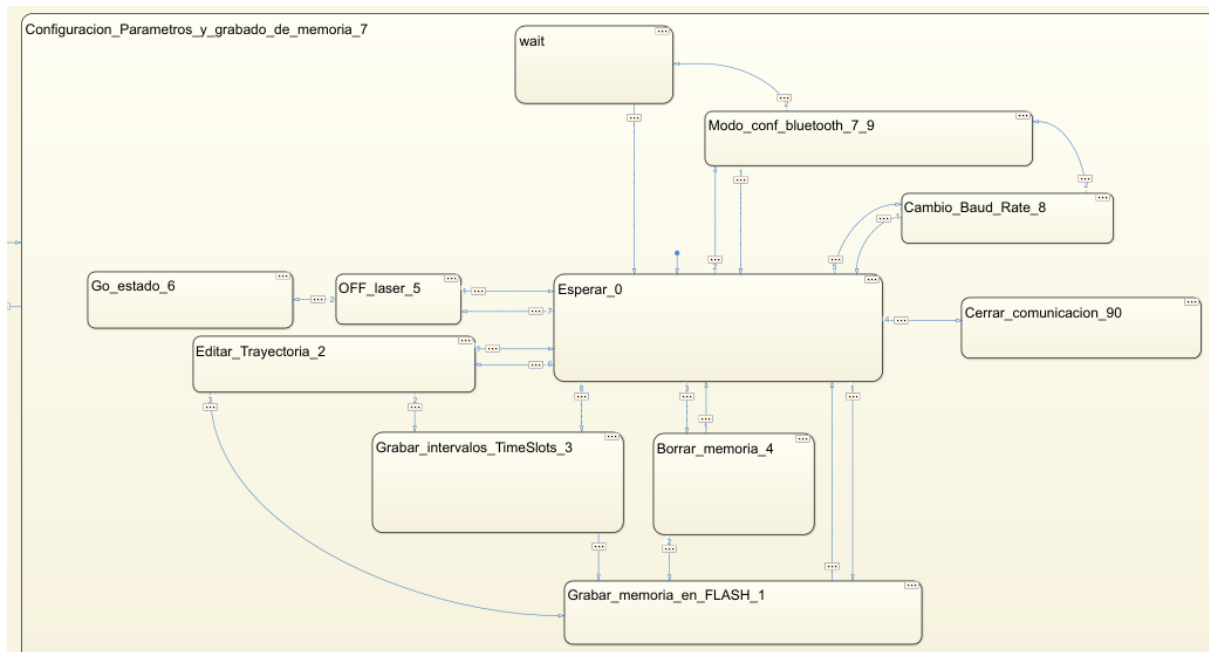


Figura 18: subestados del estado 7

Para entrar al estado 7 se debe recibir a través de la UART el comando “B”, donde al primer subestado que se ingresa se llama “Esperar_0” (figura 19), en el cual solo se esperan otros comandos.

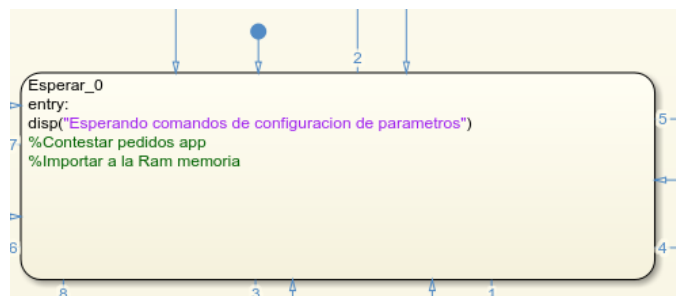


Figura 19: subestado de espera de comandos

Los comandos que permiten transicionar entre los distintos subestados son:

- “b0\n” para cerrar comunicación con la UART y salir del modo espera volviendo al modo normal de funcionamiento (figura 20). Tener en cuenta que El carácter \n es un carácter de control que representa una nueva línea en sistemas operativos y lenguajes de programación, es útil para separar distintos comandos o mensajes.

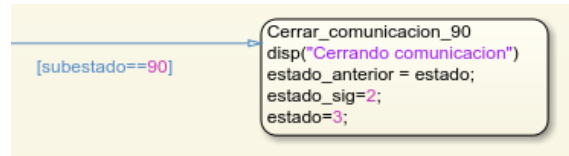


Figura 20: volver al funcionamiento normal

- “b1\n” para cancelar cualquier otra acción y volver al subestado “Esperar_0”.
- “b2\n” para grabar valores en la memoria flash (figura 21).

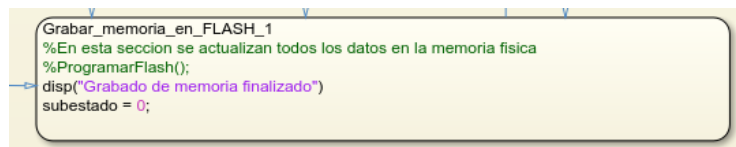


Figura 21: Grabar datos en la memoria

- “b3\n” para resetear toda la memoria flash (figura 22).

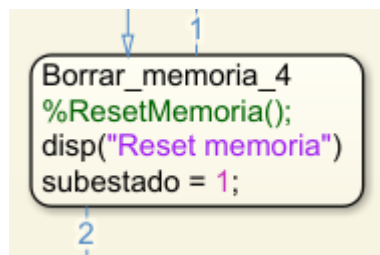


Figura 22: Reset de memoria

- “b4\n” para ir al modo configuración el Bluetooth (figura 23).

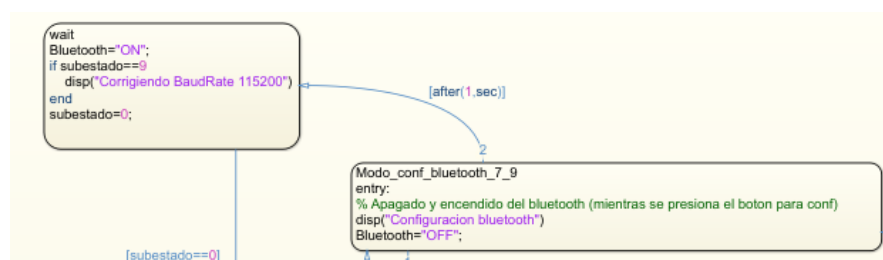


Figura 23: Configuración de Bluetooth

- “b5\n” para configurar Baud rate del Bluetooth (figura 24).

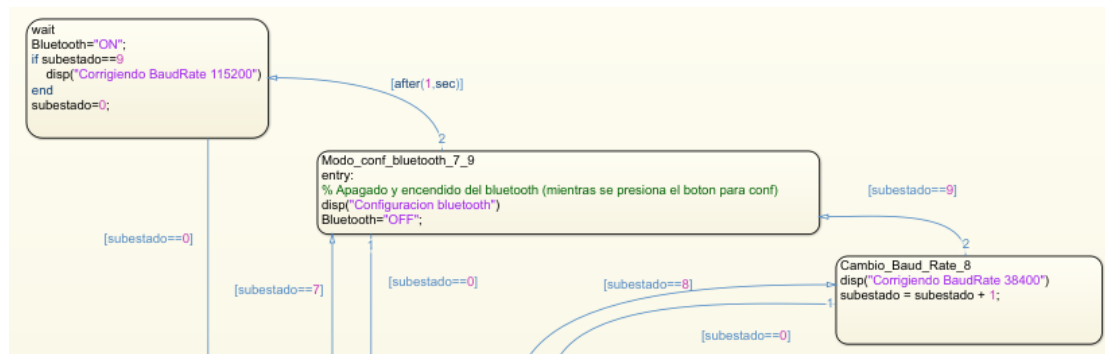


Figura 24: Configurar Baud rate del Bluetooth

Para elegir una trayectoria cuyos parámetros se desean modificar, se debe enviar un comando del tipo “c; Num; \n” a través de la UART (figura 25), donde “Num” es el número correspondiente a la trayectoria que se desea seleccionar. Vemos que este comando hace que se avance al subestado dos el cual corresponde a “Editar_trayectoria_2” de la figura 18.

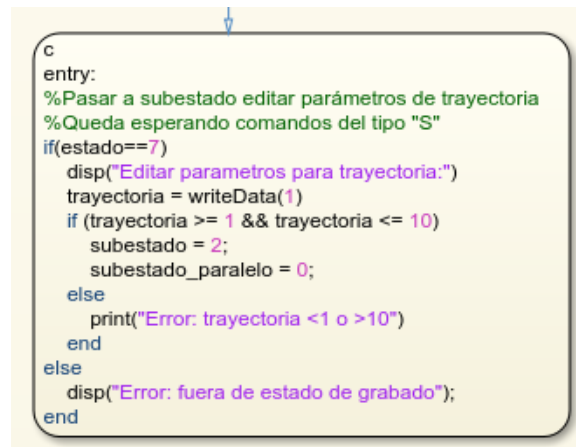


Figura 25: selección de trayectoria para cambiar sus parámetros

Se observa en la figura 26 que en el estado “Editar_Trayectoria_2” no se hace más que esperar otro comando que me permiten ajustar parámetros de la trayectoria.

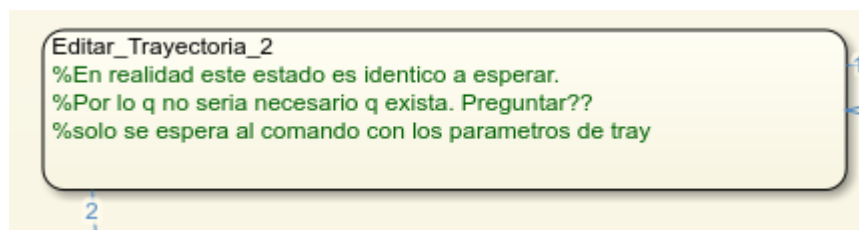


Figura 26: Subestado editar trayectoria

El comando que se espera en el subestado anterior “Editar_Trayectoria_2” es del tipo “S; Num1; Num2; Num3;\n” (figura 27), donde el primer valor indica en qué TimeSlots se activará la trayectoria (va de 1 a 8, ver tabla 2), el valor siguiente indica qué porcentaje de la velocidad máxima usará (va de 1% a 100%) y el tercer valor indica la cantidad de minutos que se esperará al finalizar la trayectoria para empezar otra (de 0 a 60 min). Finalmente se va hacia el subestado uno de grabado en la memoria flash.

```

S
entry:
%writeData(1) configuracion de timerslots
%writeData(2) velocidad
%writeData(3) minutos espera
if (estado == 5 || estado == 7) && writeData(1) < 9 && writeData(1) > 0 && writeData(2) > 0 && writeData(2) <= 100 && writeData(3) >= 0 && writeData(3) < 60
%configuramos la activacion de Timerslots
trayectorias_intervalos(trayectoria,1:3)=0;
switch (writeData(1))
case 1 %Se activa en TimeSlot 1
trayectorias_intervalos(trayectoria,1)=1;
case 2 %Se activa en TimeSlot 1 y 2
trayectorias_intervalos(trayectoria,1:2)=1;
case 3 %Se activa en TimeSlot 1, 2 y 3
trayectorias_intervalos(trayectoria,1:3)=1;
case 4 %Se activa en TimeSlot 2
trayectorias_intervalos(trayectoria,2)=1;
case 5 %Se activa en TimeSlot 2 y 3
trayectorias_intervalos(trayectoria,2:3)=1;
case 6 %Se activa en TimeSlot 3
trayectorias_intervalos(trayectoria,3)=1;
case 7 %Se activa en TimeSlot 1 y 3
trayectorias_intervalos(trayectoria,1)=1;
trayectorias_intervalos(trayectoria,3)=1;
case 8 % No se activa en ninguno
end
trayectorias_intervalos

%conf velocidad
Velocidades(trayectoria)=writeData(2)

%conf minutos de espera
Esperas(trayectoria)=writeData(3)

%Grabamos en la memoria
if (estado == 5)
subestado = 7;
else
subestado = 1;
end
else
disp("Error: fuera de estado de grabado");
end
%iter_memoria = 0;???

```

Figura 27: comando S para modificar parámetros de trayectorias

Num1\TimeSlot	1	2	3
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	0
5	0	1	1
6	0	0	1
7	1	0	1
8	0	0	0

Tabla 2: valores de Num1 en comando S

Otro comando importante es el que me permite cambiar los intervalos de tiempo de los TimeSlots “s” (figura 28), es decir que permite modificar el horario de inicio y fin de un TimeSlot. Este comando se envía de la siguiente forma “s; Num1; Num2; Num3; Num4; Num5; \n”, donde el primer valor indica que TimeSlot es (1, 2 o 3), el segundo indica hora de arranque, tercero minutos de arranque, cuarto hora de fin y quinto minutos de fin. Este comando se puede enviar mientras se graba una trayectoria o mientras se está en estado siete.

```

s
entry:
%writeData(1) N° de TimeSlot
%writeData(2) hora y writeData(3) minutos arranque
%writeData(4) hora y writeData(5) minutos arranque
if (estado == 5 || estado == 7)
if ((writeData(1) > 0) && (writeData(1) <= 3) && (writeData(2) >= 0) && (writeData(2) < 24) && (writeData(3) >= 0) && (writeData(3) < 60) && (writeData(4) >= 0) && (writeData(4) < 24) && (writeData(5) >= 0) && (writeData(5) < 60))
subestado = 8;
else
subestado = 3;
end
else
disp("Error: Valor fuera de rango");
end
else
disp("Error: fuera de estado de grabado");
end

```

Figura 28: comando “s” para recibir intervalos de tiempo para TimeSlots

Una vez definidos los valores de tiempo para el TimeSlot en cuestión se graban estos valores en la variable almacenada en la memoria (figura 29) y se procede a ir al estado de grabado en la memoria flash.

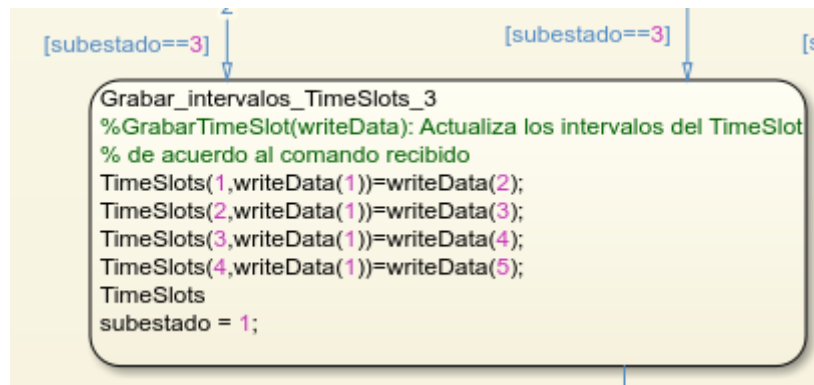


Figura 29: Escritura de los intervalos de tiempos recibidos en la memoria

El comando “P; Num; \n”, permite seleccionar la trayectoria “Num” e ir al estado cinco “Generar trayectoria” (figura 30), donde se generan los puntos que esa trayectoria seguirá de forma automática en el modo normal de funcionamiento.

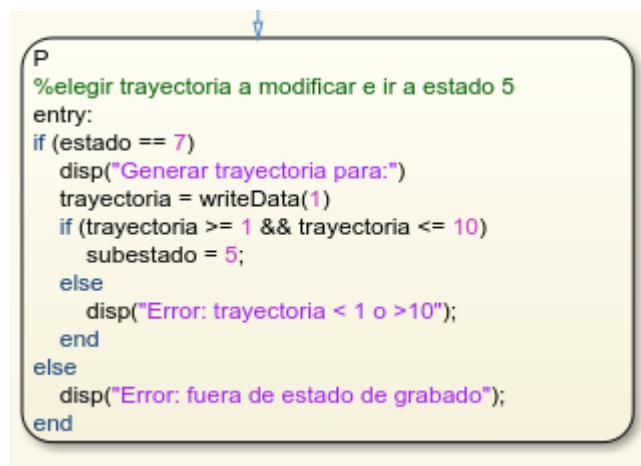


Figura 30: comando para ir al estado "Generar trayectoria"

Vemos en la figura 31 que al avanzar al subestado cinco lo primero que se realiza es el apagado del láser y luego se realiza la maniobra homing (estado 3) antes de ir al generador de trayectorias (estado 5).

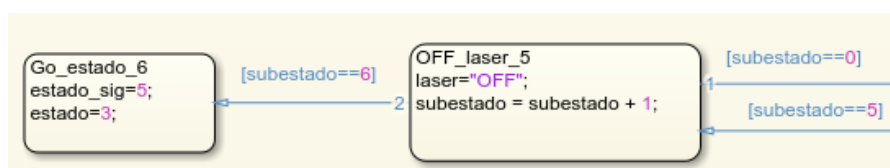


Figura 31: procedimiento previo al estado 5

7. Estado 5: Generar trayectoria

Este estado tiene como objetivo principal generar una trayectoria que los motores deben seguir en base a una serie de puntos (posiciones) almacenados en la memoria. En la figura 32 se observan los subestados que componen este estado.

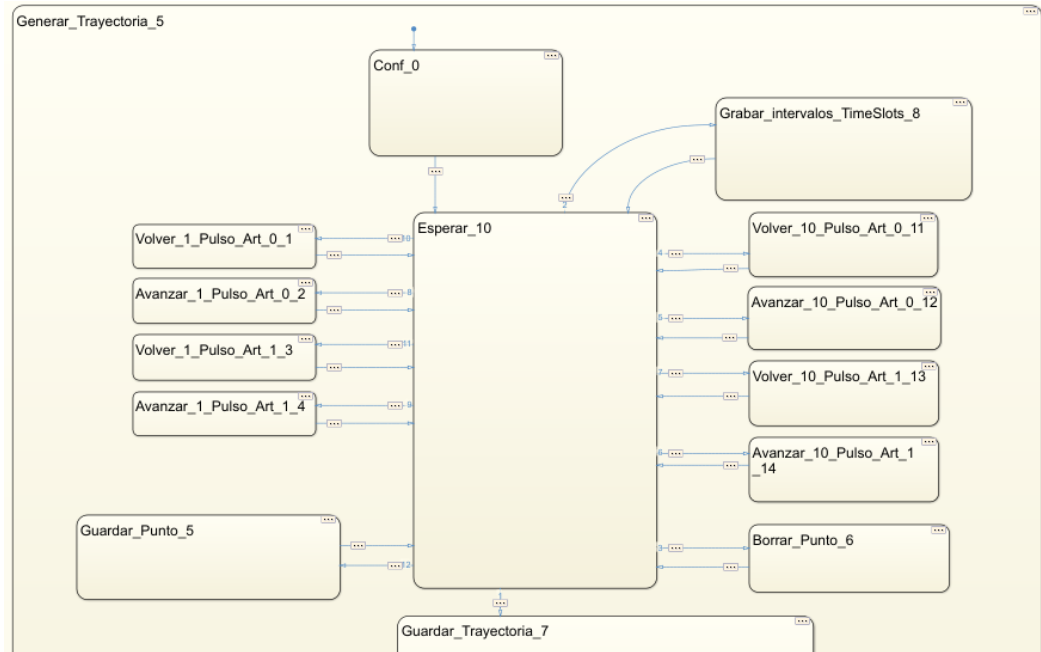


Figura 32: subestados que componen el estado 5

Inicialmente se ajusta el color del led, la velocidad, una variable llamada “ultimo_punto” y se avanza al subestado diez. El subestado 10 solo se utiliza como modo espera para esperar otros comandos (figura 33).

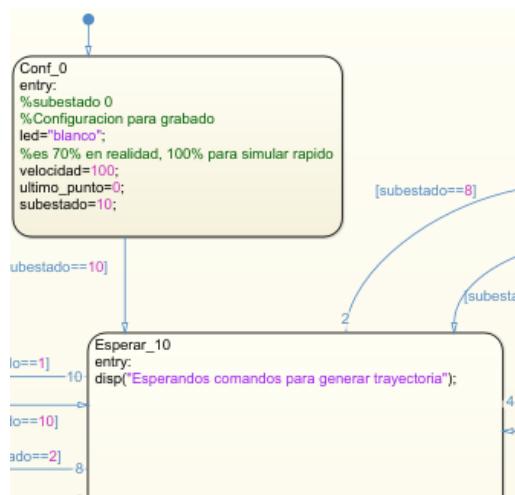


Figura 33: subestado 10 para espera de comandos

Dentro de este estado también podemos aplicar el comando “s” para modificar los intervalos de tiempo de los TimeSlots (figura 34), el funcionamiento es el mismo que se explicó anteriormente para la figura 28.

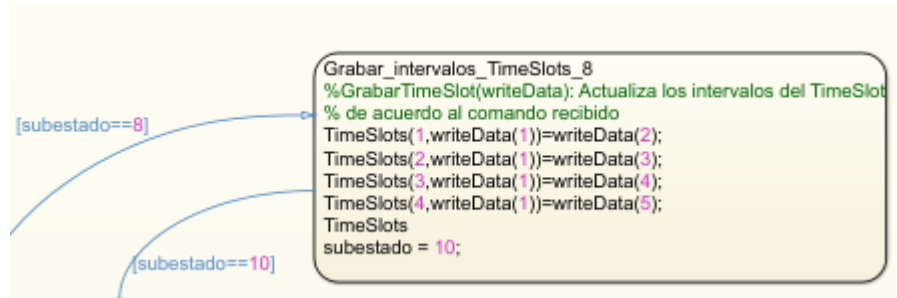


Figura 34: subestado para modificar TimeSlots

Otros comandos importantes en este estado se muestra en la figura 35, los cuales son “Ai”, “Vi”, “AiR” y “ViR”; donde “i” indica el número de motor (motor 0 o motor 1). Se observa que dependiendo de cuál sea el comando enviado se derivara el sistema a distintos subestados, por ejemplo si el comando es “A0” el sistema avanza al subestado dos, si el comando es “V1R” el sistema avanza al subestado trece.

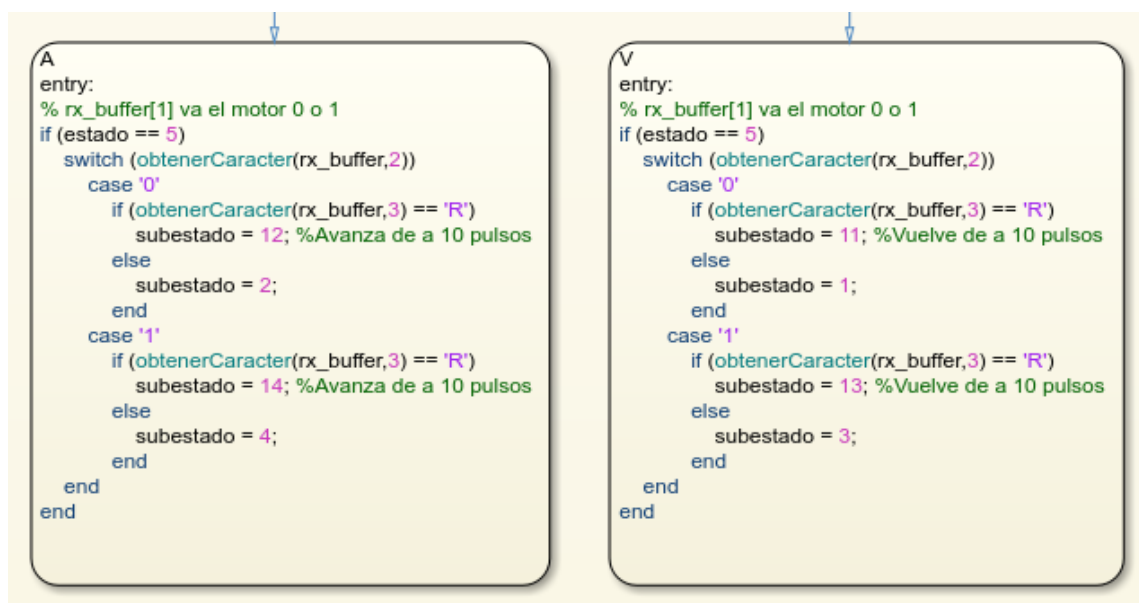


Figura 35: comandos “A” y “V” para activar transiciones

Dependiendo del subestado donde el sistema avanza, se modifica la consigna para realizar distintos movimientos en los motores (figura 36). Los movimientos pueden ser de

avance y retroceso para ambos motores, además de que los avances y retrocesos pueden ser de un solo pulso o de diez pulsos para conseguir un movimiento más rápido.

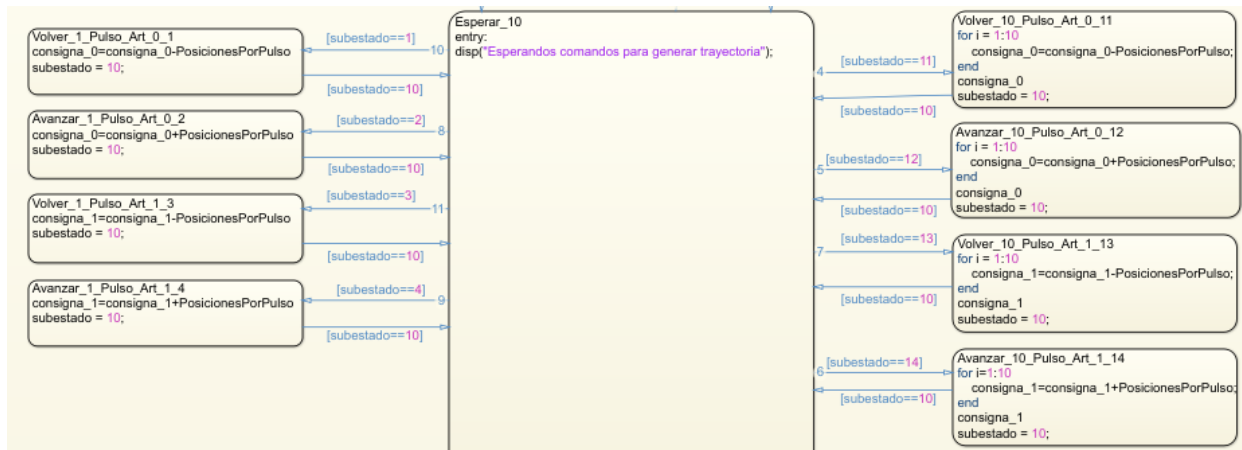


Figura 36: subestados que permiten distintos movimientos

Los últimos tres comandos que se utilizan en este estado son “p0”, “p1” y “p2”. Estos comandos al igual que los anteriores sirven para indicar hacia que subestado debe avanzar el sistema para realizar determinadas funciones. El comando “p0” llama al subestado cinco (siempre y cuando no se superen los 100 puntos almacenados), el “p1” al seis y el “p2” al siete (figura 37).

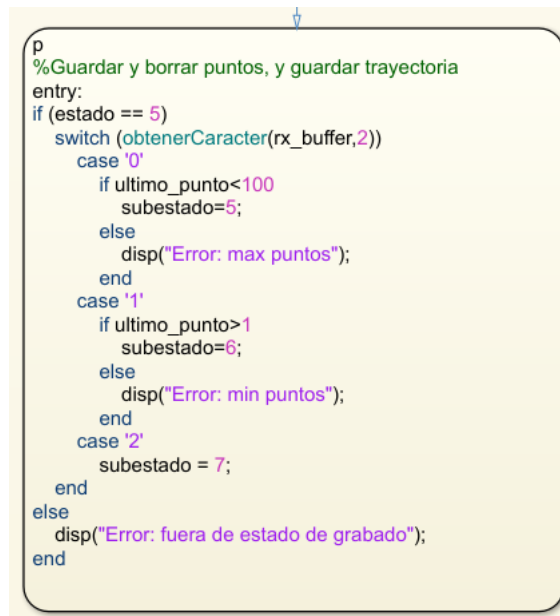


Figura 37: comando "p" para activar transiciones

El subestado cinco almacena en el vector “trayectorias_puntos” las posiciones actuales de los dos motores (figura 38). Por otro lado, el subestado seis elimina las dos últimas posiciones de los dos motores en el vector “trayectoria_puntos”.

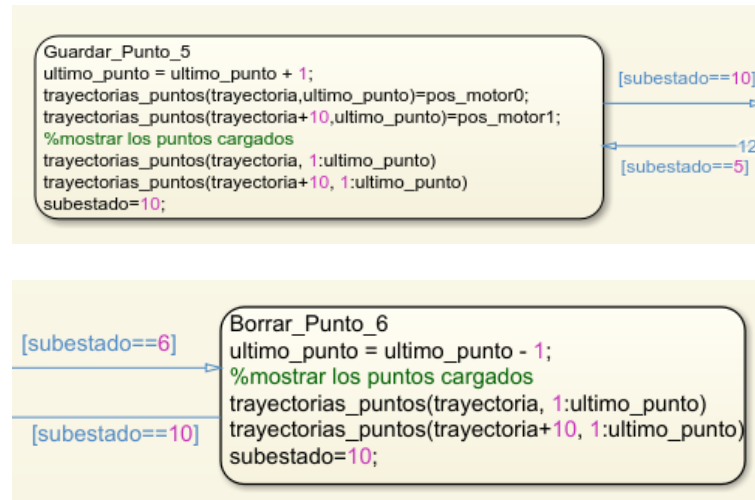


Figura 38: subestados para guardar o borrar puntos

Finalmente en el subestado siete se graban en la memoria la secuencia de puntos que fue almacenada a lo largo del proceso de generación de trayectoria (figura 39). Se resetean algunas variables, se apaga el láser y se procede a realizar la maniobra de homing para volver al estado 7 de configuración de parámetros y grabado de memoria.

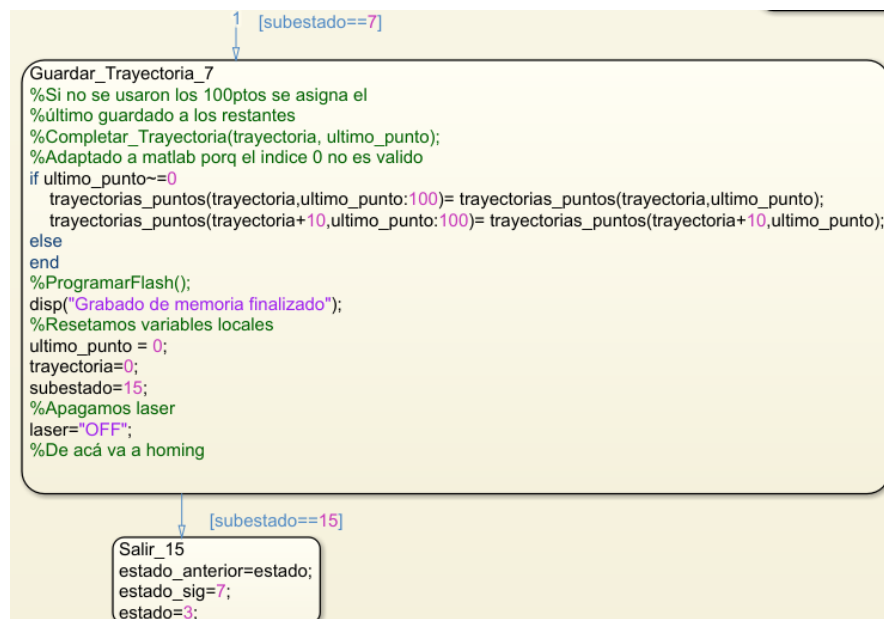


Figura 39: Procedimiento final del estado 5

8. Sistema de medición de temperatura

La medición de temperatura del sistema se realiza cada un cierto periodo de tiempo, el cual está controlado por el temporizador “Timer2” (figura 40). Este temporizador envía una señal “Flag_hora” cada 5 segundos correspondientes al cálculo almacenado en “Periodo_tim2” para tomar una lectura de la temperatura, además también envía una señal llamada “Flag_RGB”.

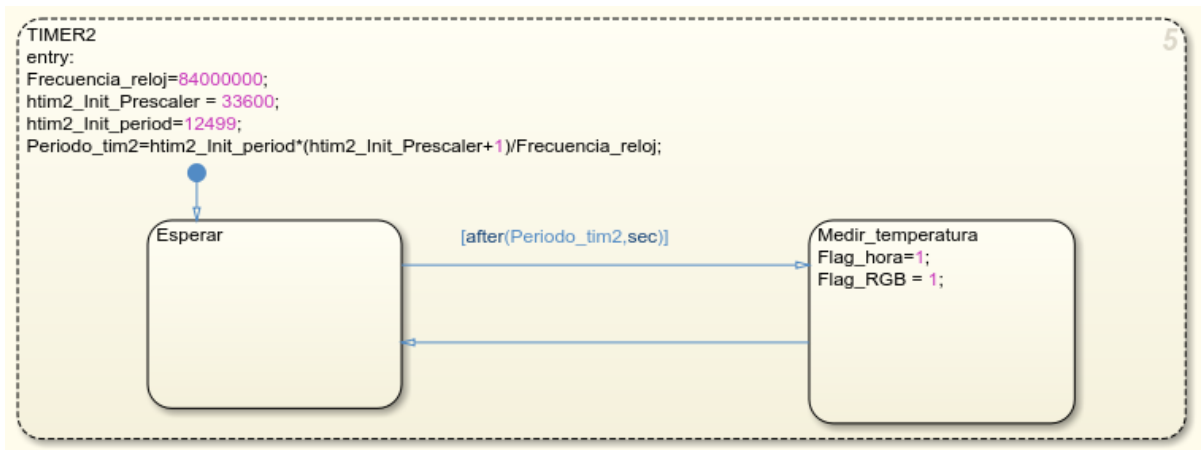


Figura 40: señal para tomar lectura de temperatura

El bloque “Sistema_Seg_temp” es el encargado de realizar todo el procedimiento para realizar la medición de la temperatura (figura 41). Primero se espera la señal “Flag_hora”, luego se resetea la señal y se evalúa si no estamos en el estado seis correspondiente al estado de emergencia. Luego se apaga el temporizador “temp5” que es el encargado de generar los pulsos y se resetean algunas variables.

En la figura 41 se observa un bucle for que se repite una cantidad de veces dadas por “cant_med”. Dentro del bucle se comentan varias funciones cuya finalidad es tomar la lectura del sensor y pasarla a través del ADC (conversor de señal analógica a digital), finalmente se almacena la lectura en la variable “aux1” para posteriormente añadirla a una sumatoria que se utilizara a la salida del bucle para obtener el promedio de las lecturas realizadas y asignarlo a la variable “Termocupla”.

```

Sistema_Seg_temp
during:
if (Flag_hora)
    Flag_hora = 0;
    %HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_0);    %Aca reinicio el pic. porq???
    if (estado == 6)
        %HAL_TIM_Base_Stop_IT(&htim5);
        temp5="OFF";
        %ConsultaHora(&hi2c1, time);
        %ControlCambioDia();
        %MedirTemperatura()
        cant_med = 5;
        prom = 0;
        aux1=0;
        %Se realizan las 5 mediciones
        for i = 1 : cant_med
            %sConfig.Channel = ADC_CHANNEL_1;    %selecciona el canal del ADC que se utilizará para la medición.
            %HAL_ADC_ConfigChannel(&hadc1, &sConfig); %configura el ADC con los ajustes especificados.
            %HAL_ADC_Start(&hadc1); %inicia la conversión del ADC.
            %if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK) { ... } verifica si la conversión del ADC se ha completado correctamente.
            %lectura = HAL_ADC_GetValue(&hadc1); obtiene el valor de la conversión del ADC.
            %Vout = lectura * Vin / 1023; %convierte el valor del ADC a un voltaje de salida.
            %R2 = (uint32_t) R1 * Vout / (Vin - Vout) + offset_Termores; %calcula la resistencia del termistor a partir del voltaje de salida.
            %aux1 = (float) Interpolar(lectura); %interpola el valor de la lectura para convertir el valor de la lectura a una temperatura.
            aux1=lectura_temperatura;
            prom = prom + aux1; %suma el valor interpolado al promedio.
        end
        Termocupa = prom / cant_med; %calcula el promedio de las mediciones
    %End Medir Temperatura();

```

Figura 41: procedimiento de medición de temperatura (Parte 1)

Luego en la figura 42 se observa que se verifica si el sensor de temperatura está conectado, en caso de no estarlo se mide cuánto tiempo lleva así y se lo almacena en la variable “contador_sensor_desc”.

Posteriormente se obtiene la diferencia entre la lectura de temperatura actual y la anterior, esto se hace para comprobar que la variación de temperatura este dentro de un rango de valores que se consideran lógicos, ya que a veces el sensor puede fallar y mostrar picos que no son deseables. Aquellas lecturas fuera del rango de variación se descartan completamente, pero en el caso de que se hayan descartado un número de lecturas mayor a “contador_mediciones_max”, la próxima lectura se tomara sin importar el rango de variación.

Después se almacena el valor máximo de todas las temperaturas registradas en la variables “Temperatura_max”. Luego se controla que el sensor no este apagado por un tiempo mayor a “mediciones_desc”.

```
%Contador de tiempo mientras el sensor permanece desconectado (esos dos valores de temp son los q representan al sensor desconectado)
if(Termocupla > temp_desconetado_max || Termocupla < temp_desconetado_min)
    contador_sensor_desc = contador_sensor_desc + 1;
else
    contador_sensor_desc = 0;
end
%valor absoluto de la diferencia de medicion actual y anterior
Abs_temp = abs(Termocupla - Termocupla_ant);
%si la ultima medicion tomada es muy diferente a la actual la descartamos (siempre y cuando no superemos un N° limite de descartes)
if (Abs_temp > grados_variable && contador_mediciones < contador_mediciones_max)
    Termocupla = Termocupla_ant;
    contador_mediciones = contador_mediciones + 1;
else
    Termocupla_ant = Termocupla;
    contador_mediciones = 0;
end
%Si la lectura actual es la mayor lectura registrada hasta el momento, la colocamos como Temp_max
if (Termocupla > Temperatura_max)
    Temperatura_max = Termocupla;
end

%si el sensor permanece desconectado en mas de 12 mediciones y el sensor esta apagado y no estamos en estado emergencia, lo encendemos
%Preguntar esta parte???
if(contador_sensor_desc > mediciones_desc && estado_sensor == 0 && estado ~= 1)
    estado_sensor = 1;
    led="violeta";
end
```

Figura 42: procedimiento de medición de temperatura (Parte 2)

En la figura 43 se observa que si el láser esta encendido y se supera la temperatura máxima de seguridad “Temperatura_Emergencia”, se avanza al estado uno que corresponde al estado de emergencia. Se observa que se apagan tanto el láser como los motores y se encienden los ventiladores “FANS”, además se almacena la hora en la cual se dispara el estado de emergencia en el vector “time_ant” y el tiempo en el que se apaga el láser en la segunda columna del vector “tiempo_uso”. Finalmente se inicia nuevamente el temporizador “temp5”.

```
%Si el laser esta prendido y la temp supera la temp de emergencia (si la supera demasiado es un error de lectura), entramos al modo emergencia
if(Termocupla >= Temperatura_Emergencia && Termocupla <= (Temperatura_Emergencia + 40) && laser == "ON")
    estado_anterior = estado;
    estado = 1;
    estado_sig = 3;
    subestado = 1;
    led="violeta";
    subestado_paralelo = 0;
    FANS="ON";
    laser="OFF";
    motor0="OFF";
    motor1="OFF";
    time_ant(3) = time(3); %Hora
    time_ant(2) = time(2); %Minutos
    time_ant(1) = time(1); %Segundos
    tiempo_uso(1,2) = time(1); %Segundos finales
    tiempo_uso(2,2) = time(2); %Minutos finales
    tiempo_uso(3,2) = time(3); %Hora finales
    %Uso_laser();
end
%HAL_TIM_Base_Start_IT(&htim5);
temp5="ON";
end
end
```

Figura 43: procedimiento de medición de temperatura (Parte 3)

9. Estado 1: Emergencia

Este estado está orientado a ejecutar las maniobras de emergencia debido a altas temperaturas o debido a alguna falla en los fines de carrera. En la figura 44 se observan los subestados que componen este estado.

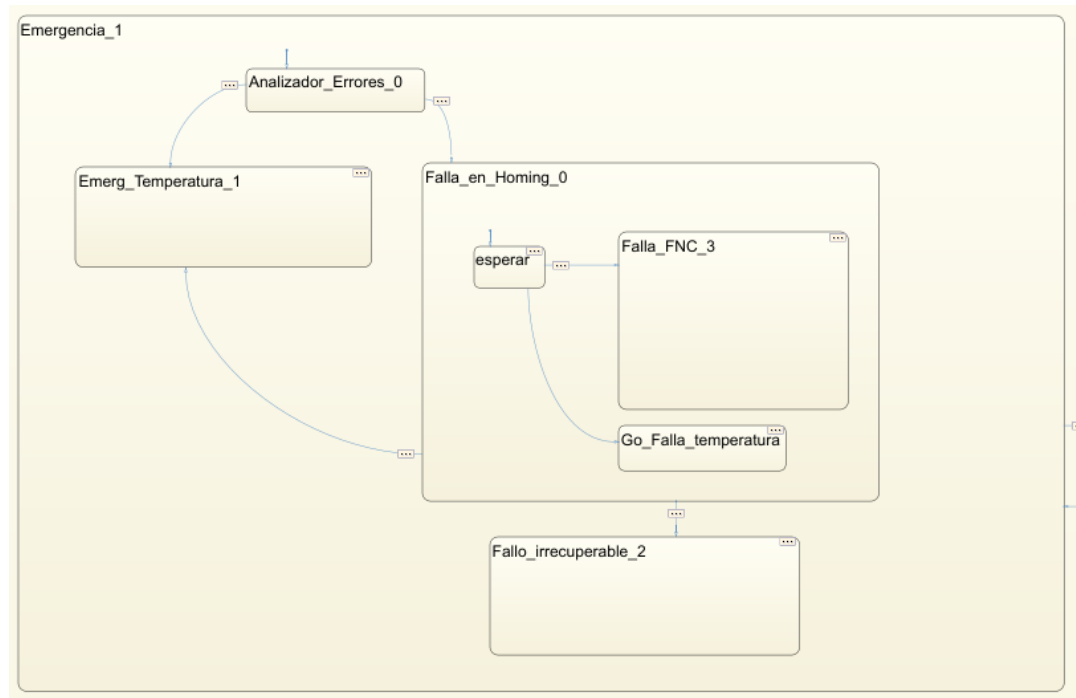


Figura 44: subestados que componen el estado 1

En la figura 45 lo primero que se observa es un análisis del tipo de error que llevo al sistema al estado de emergencia, si el error se debe a la temperatura la variable “subestado” tendrá el valor de uno, mientras que si se debe a fallas en los sensores finales de carrera la variable tomara el valor cero.

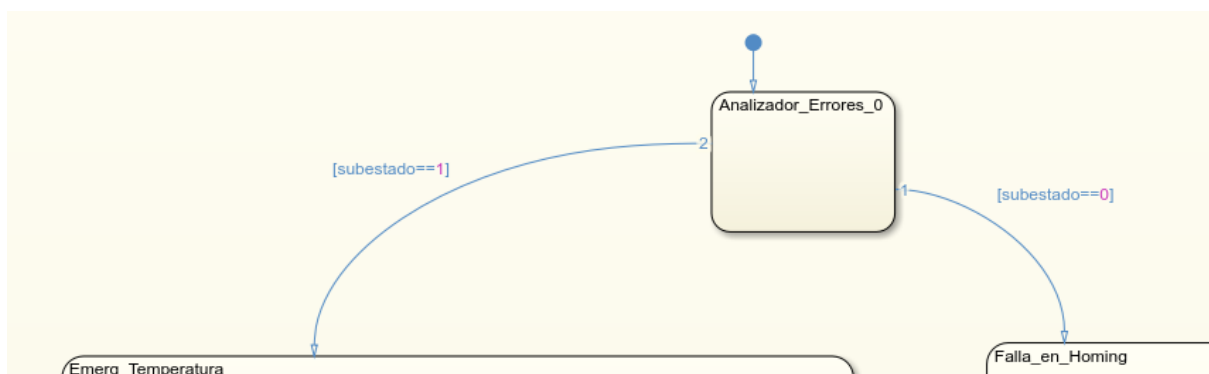


Figura 45: evaluación del tipo de error

El procedimiento de emergencia a causa de la temperatura (figura 46) consiste en dejar enfriar el sistema y cuando la temperatura sea menor a “Temperatura_restaurar” y haya pasado una cantidad de “Min_descenso_temp” minutos, se avanza a la maniobra homing nuevamente para seguir con el funcionamiento normal del sistema.

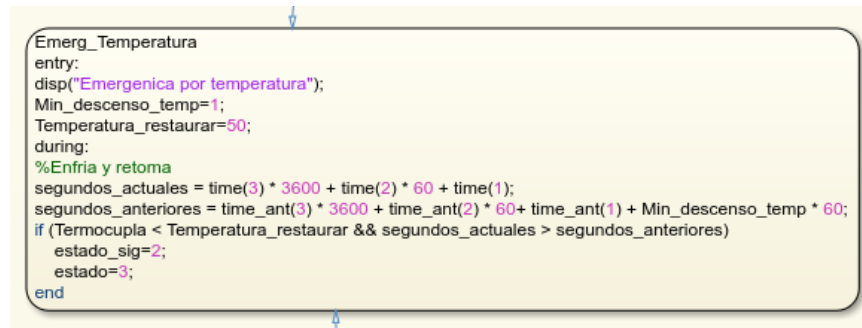


Figura 46: procedimiento a falla por temperatura

Por otro lado, el procedimiento de emergencia debido a una falla en los fines de carrera consiste en llevar el motor a una posición segura, almacenar el tiempo y esperar tres minutos antes de continuar con el funcionamiento normal del sistema (figura 47).

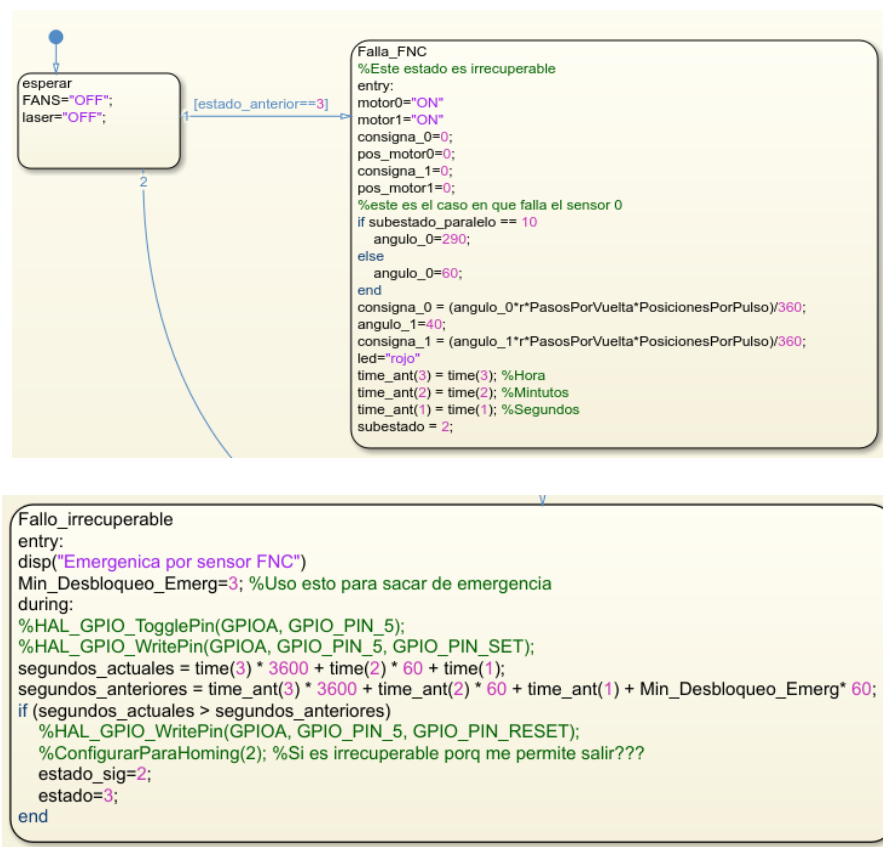


Figura 47: Procedimiento frente a falla por finales de carrera

10. Anexo 1: Código en C y Bibliotecas

10.1 Tabla de variables

Nombre en Matlab	Nombre en C	Tipo de variable	Biblioteca que la define
Pulso_0/1	PUL_0/1	pin GPIO	MovMotor.c
dir0/1	DIR_0/1	pin GPIO	MovMotor.c
velocidad		uint16_t	Trayectorias.c
TimerPeriod_Min		long int	MovMotor.c
MicroStep		int	MovMotor.c
PosicionesPorPulso		int	MovMotor.c
PasosPorVuelta		int	MovMotor.c
r		int	MovMotor.c
ValorInicial		int	MovMotor.c
pos_motor0/1	Posición	long int vector[2]	MovMotor.c
consigna_0/1	Consigna	long int vector[2]	MovMotor.c
TimeSlot		int	main.h
trayectoria		int	main.c
time		long int vector[3]	main.c
time_ant		long int vector[3]	main.c
tiempo_uso		long int matriz [3x2]	main.c
tiempo_laser		long int vector[3]	main.c
led	ROJO, AZUL, VERDE	pin GPIO	LedRGB.c
laser	LASER	pin GPIO	Laser.c
FANS		pin GPIO	Laser.c
rx_buffer		char vector[75]	main.c
contador_SIM800		int vector[2]	main.c
Flag_hora		int	main.h
Temperatura_Emergencia		int	main.c
grados_variable		int	main.c
contador_mediciones		long int	main.c
contador_mediciones_max		int	main.c
temp_desconectado_max		int	main.c
temp_desconectado_min		int	main.c
estado_sensor		int	main.c
contador_sensor_desc		long int	main.c
mediciones_desc		int	main.c
Termocupla		long int	main.c
Termocupla_ant		long int	main.c
Temperatura_max		long int	main.c
estado_anterior		int	main.h
estado		int	main.h

Tabla 3: Tabla de variables en C

10.2 Pin Out

Para entender el funcionamiento de las distintas bibliotecas es necesario primero observar la disposición de los pines en la placa del microcontrolador junto con las funciones y usos asignados a cada uno de ellos. En la figura 48 se observan todos los pines correspondientes a la placa NUCLEO-F411RE.

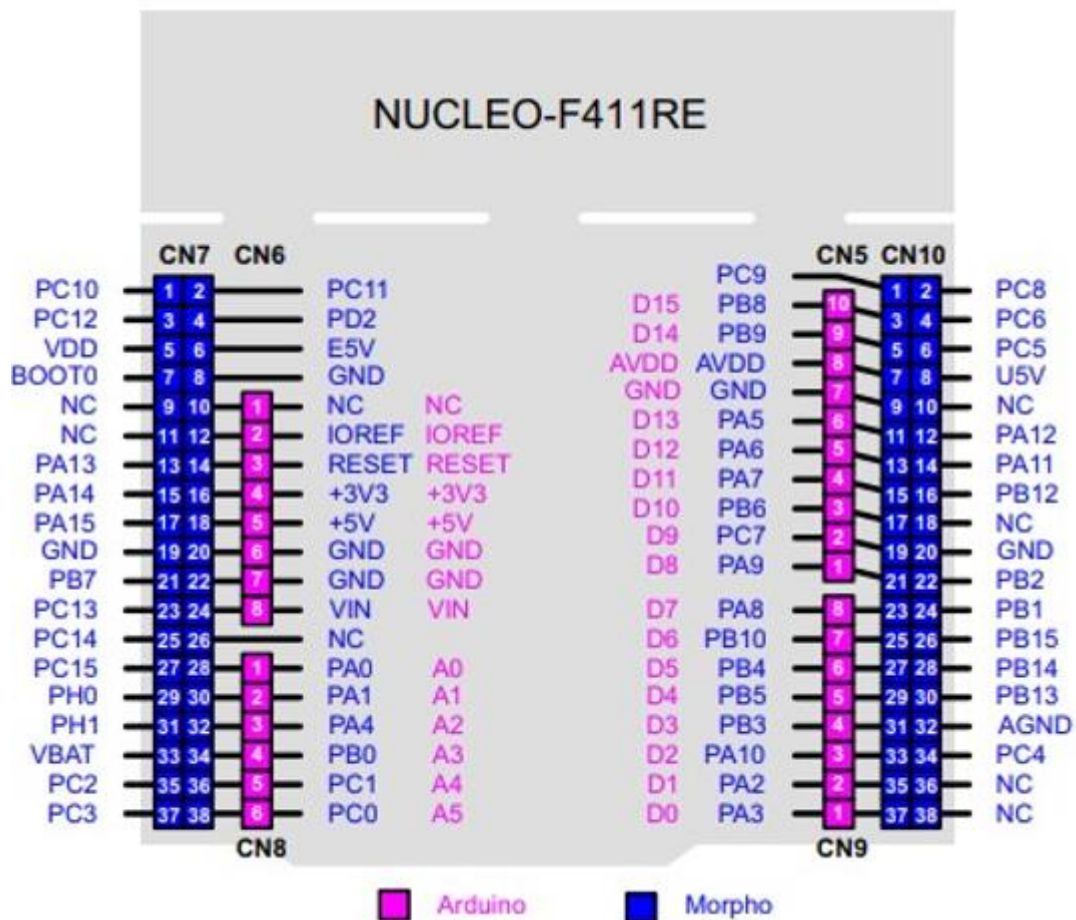


Figura 48: Pines NUCLEO-F411RE

En las tablas 4 y 5 se observan las funciones y usos asignados a cada uno de los pines. Se observa que los usos son variados, como por ejemplo control de los motores, ventiladores, laser, finales de carrera, LED RGB, sensores, transmisión, recepción, etc (cada una de estas funciones serán desarrolladas en las próximas secciones).

CN7				
NUCLEO	uC	FUNCIÓN	USO	
1	PC10			
2	PC11			
3	PC12			
4	PD2			
5	VDD			
6	E5V	Alimentación	5V externo	
7	BOOT0			
8	GND			
9	NC			
10	NC			
11	NC			
12	IOREF			
13	PA13	TMS		
14	RESET	RESET	BRESET	
15	PA14	TCK		
16	3V3			
17	PA15	GPIO_INPUT	FINAL DE CARRERA 0	pull up
18	5V			
19	GND			error 1
20	GND			
21	PB7	GPIO_INPUT	FINAL DE CARRERA 1	pull up
22	GND			
23	PC13	GPIO_EXTI13		Blue botton
24	VIN			
25	PC14			
26	NC			
27	PC15			
28	PA0	GPIO_OUTPUT	Pulsos pic	wake up
29	PH0			
30	PA1	ADC_Reader	Reserva sensor	
31	PH1			
32	PA4	GPIO_OUTPUT	FANS	
33	VLCD			
34	PB0	GPIO_OUTPUT	Rojo RGB	
35	PC2	GPIO_OUTPUT	Láser	
36	PC1			
37	PC3	ADC_Reader	Sensor temp	
38	PC0	GPIO_OUTPUT	ALIMENTACIÓN SIM800	

Tabla 4: Función y uso de cada pin (Canal 7)

CN10				
NUCLEO	uC	FUNCIÓN	USO	Observ
1	PC9			
2	PC8			
3	PB8	I2C SCL	SCL reloj	
4	PC6	USART6_TX	RX Bluetooth	
5	PB9	I2C SDA	SDA Reloj	
6	PC5			
7	AVDD			
8	U5V			
9	GND			
10	NC			
11	PA5	LD2 (led verde)	LD2	
12	PA12	Pin de Debug		
13	PA6			
14	PA11	Pin de Debug		
15	PA7			
16	PB12	GPIO_OUTPUT	Pulsos motor 0	
17	PB6			
18	PB11			
19	PC7	USART6_RX	TX Bluetooth	
20	GND			
21	PA9	USART1_TX	RX? SIM800	
22	PB2	GPIO_OUTPUT	Dirección motor 0	
23	PA8	GPIO_OUTPUT	Reset Bluetooth	
24	PB1			
25	PB10	GPIO_OUTPUT	Encendido Bluetooth	
26	PB15	GPIO_OUTPUT	Enable motor 0	
27	PB4	GPIO_OUTPUT	Azul RGB	
28	PB14	GPIO_OUTPUT	Pulsos motor 1	
29	PB5	GPIO_OUTPUT	Verde RGB	
30	PB13	GPIO_OUTPUT	Dirección motor 1	
31	PB3	SWO		
32	AGND			
33	PA10	USART1_RX	TX? SIM800	
34	PC4	GPIO_OUTPUT	Enable motor 1	
35	PA2	USART2_TX	UART2_TX	
36	NC			
37	PA3	USART2_RX	UART2_RX	
38	NC			

Tabla 5: Función y uso de cada pin (Canal 10)

10.3 Biblioteca LedRGB

El objetivo de la biblioteca LedRGB es facilitar el control del led RGB que indica el estado actual del sistema. En la tabla 6 se observa la descripción del estado actual dependiendo del color RGB que presenta el led.

Descripción	Color RGB
Luz Estable: Temperatura máxima alcanzada Parpadeo: Sensor de temperatura desconectado	Violeta
Fallo Irrecuperable (Atascamiento, sensor o motor fallando)	Rojo
Estado de espera y selección de trayectoria	Azul
Calibración o Homing	Amarillo
Trayectoria en curso	Verde
Grabado de Trayectoria	Blanco
Configuración de Parámetros y grabado de memoria	Celeste
El robot está sin corriente o la electrónica está trabada/quemada	Apagado

Tabla 6: estados según Led RGB

Para conseguir los distintos colores se deben enviar las señales adecuadas en cada uno de los pines destinados al control del led RGB. Si bien en la tabla 5 se especifica que pin corresponde a cada uno de los tres colores (disposición estándar), en la práctica podrían intercambiarse los pines y el código contempla esta posibilidad. Se define una constante llamada LED_WIRES, esta constante se utiliza para determinar qué pines se asignarán a cada color del LED, como se observa en la tabla 7.

LED_WIRES	ROJO	AZUL	VERDE
0 (estándar)	PB0	PB4	PB5
1	PB4	PB0	PB5
2	PB4	PB0	PB5

Tabla 7: disposiciones de pines RGB

Cada LED RGB tiene tres diodos emisores de luz, uno para cada color: rojo, verde y azul. Dependiendo de la combinación de intensidades de cada color primario, se obtiene un color diferente. Por ejemplo la función “ColorAmarillo ()”, para conseguir que la luz del led tome color amarillo pone las señales del pin rojo y verde en alto (PB0 y PB5 en alto de acuerdo a la disposición estándar de pines). El resto de funciones se observan en la tabla 8.

Función	Descripción
OffROJO	Apagar rojo
OnROJO	Encender rojo
OffAZUL	Apagar azul
OnAZUL	Encender azul
OffVERDE	Apagar verde
OnVERDE	Encender verde
OffRGB	Apagar todos
ColorBlanco	Encender todos
ColorRojo	Encender solo rojo
ColorAzul	Encender solo azul
ColorVerde	Encender solo verde
ColorVioleta	Encender solo azul y rojo
ColorAmarillo	Encender solo rojo y verde
ColorCeleste	Encender solo azul y verde
GetLedWires	Obtener la disposición de pines

Tabla 8: Funciones para el Led RGB

10.4 Biblioteca Laser

El objetivo de la biblioteca Laser es ofrecer las funciones que permitan facilitar el control del láser y los ventiladores. Como se observa en la tabla 5 el pin PC2 controla la activación del láser y el pin PA4 controla la activación de los ventiladores, por lo que solo es necesario poner en alto o bajo esos pines para activar o desactivar esos periféricos. En la tabla 9 se observan las funciones definidas y su descripción.

Función	Descripción
OnLASER()	Pone en 1 el pin PC2
OffLASER()	Pone en 0 el pin PC2
OnFANS()	Pone en 1 el pin PA4
OffFANS()	Pone en 0 el pin PA4
GetEstadoLaser()	Obtener el estado del laser
GetEstadoFAN()	Obtener el estado de los ventiladores

Tabla 9: Funciones para laser y ventiladores

En esta biblioteca también se encuentra una variable llamada “pinLevel2”, la cual es una constante de 32 bits (uint32_t) con un valor binario. El objetivo de esta variable es adaptar el código de acuerdo a la polaridad de los periféricos, sin la necesidad de modificar operaciones lógicas para obtener el resultado deseado en la activación o desactivación de pines. Es decir, que los bits de esta variable tomaran valores ceros o unos dependiendo de la polaridad del periférico en cuestión.

10.5 Biblioteca RTC

La biblioteca RTC contiene las funciones necesarias para obtener la información del módulo reloj en tiempo real (real time clock o RTC) a través del protocolo I2C. Esta biblioteca contiene cuatro funciones en las cuales se ven involucradas variables de tiempo.

La primera función se llama “ConsultarHora” en la cual se consultan a través del protocolo I2C las direcciones de los registros de segundos, minutos y horas en el RTC; las cuales son 0x00, 0x01 y 0x02 respectivamente. Tener en cuenta que 0xD0 es la dirección I2C del RTC. Finalmente se realizan conversiones de las lecturas a valores decimales y se almacenan en la posición de memoria correspondiente al array “hora”.

Otra función importante es la de “CambiarHora”, que se utiliza para cambiar la hora en el módulo de reloj en tiempo real (RTC) a través del protocolo I2C. Esta función recibe como argumentos las horas y minutos que se desean cargar en los registros del RTC, primero los transforma de formato decimal a BCD (Binary-Coded Decimal) por cuestiones de compatibilidad y posteriormente, en base a la dirección I2C del RTC y la dirección de los registros de las horas y minutos, actualiza los valores.

10.6 Biblioteca Trayectorias y biblioteca MY_FLASH

Esta biblioteca facilita modificar todo lo relacionado con los parámetros y las posiciones de los diversos puntos que conforman una trayectoria. Para que estos datos se presenten de forma ordenada y puedan almacenarse sin importar que el equipo se apague, es necesario hacer uso de una memoria.

Una memoria contiene posiciones de memoria que se conocen como direcciones de memoria. Cada posición en la memoria tiene una dirección única que permite a la computadora localizar y acceder a los datos almacenados en esa posición específica.

En la tabla 10 se observa cómo se distribuyen las posiciones de memoria en el equipo ecolaser 200. Se observa que las primeras 10 posiciones corresponden a la activación o

desactivación de las diez trayectorias, posteriormente siguen las horas y minutos de cuando empiezan y terminan cada uno de los 3 TimeSlots disponibles, a continuación se almacenan todas los parámetros y puntos que conforman cada una de las 10 trayectorias, y finalmente se almacena el tiempo que estuvo encendido el láser en horas, minutos y segundos (Data Log Láser)

Función	Inicio	Fin	Diferencia
On/Off trayectorias	0	9	10
Hora y minutos encendido y apagado c/Timeslot	10	21	12
Trayectoria 1	22	228	207
Trayectoria 2	229	435	207
Trayectoria 3	436	642	207
Trayectoria 4	643	849	207
Trayectoria 5	850	1056	207
Trayectoria 6	1057	1263	207
Trayectoria 7	1264	1470	207
Trayectoria 8	1471	1677	207
Trayectoria 9	1678	1884	207
Trayectoria 10	1885	2091	207
Data Log Láser	2092	2094	3

Tabla 10: Posiciones de memoria

En la tabla 10 se observa que 207 espacios de memoria son utilizados por cada una de las trayectorias. Sabemos que cada trayectoria almacena como máximo 100 puntos de posiciones en el espacio, por lo que es necesario contar con 200 posiciones de memoria para almacenar todas las posiciones que deben tomar los dos motores para seguir la trayectoria. Tres posiciones de memoria corresponden a la activación o desactivación de la trayectoria en cuestión para cada uno de los 3 TimeSlots, otro es para almacenar el valor de la velocidad, otro es para almacenar el tiempo de espera y finalmente las otras 2 posiciones corresponden a identificadores. De esta forma se tienen los 207 espacios de memoria que corresponden a una trayectoria, en la tabla 11 se observa lo mencionado en este párrafo.

Parámetros por trayectoria	Posiciones usadas
Identificador	1
Puntos de ambas articulaciones	200
Activación por Timeslot	3
Velocidad	1
Minutos de espera	1
Identificador	1

Tabla 11: Espacios de memoria utilizados por cada trayectoria

La biblioteca “Trayectorias” ofrece las variables y funciones que permiten modificar los parámetros de los espacios de memoria correspondiente a cada una de las trayectorias.

Para representar a la memoria física se crea un vector de 2095 elementos llamado “MemoriaCompleta”.

Las funciones correspondientes a esta biblioteca toman los argumentos que reciben y los escriben en el índice del vector “MemoriaCompleta” que corresponda. Por ejemplo, la función “GuardarParametrosTrayectoria” algunos de los argumentos que recibe son la trayectoria “tray” y la velocidad “velocidad” que será asignada a la trayectoria. Si suponemos que “tray=1” y “velocidad=20” entonces el valor de velocidad, de acuerdo a la tabla 10, debería colocarse en la posición 226 del vector “MemoriaCompleta”, es decir “MemoriaCompleta [226]=20”.

Para facilitar el manejo de las posiciones de la memoria, en el programa se han definido varias constantes que actúan como índices en el vector “MemoriaCompleta”, todas estas constantes se observan en la tabla 12. Esto permite expresar el ejemplo del párrafo anterior como “MemoriaCompleta [i_tray1+i_velocidad]=20”.

Descripción de pos	Constantes en C	Índice
Inicios de trayectorias	indices[10]	i_tray1
		22
		i_tray2
		229
		i_tray3
		436
		i_tray4
		643
		i_tray5
		850
		i_tray6
		1057
		i_tray7
		1264
		i_tray8
		1471
		i_tray9
		1678
		i_tray10
		1885
Parámetros de trayectorias	i_puntos	1
	i_timeslots	201
	i_velocidad	204
	i_mintutos_intervalo	205
Datos láser	i_DataLogLaser	2092

Tabla 12: Variables que almacenan índices para "MemoriaCompleta"

Finalmente para grabar los datos de “MemoriaCompleta” en la memoria física se llama a la función “MY_FLASH_WriteN” de la biblioteca “MY_FLASH” la cual hace uso de funciones proporcionadas por la biblioteca de abstracción de hardware (HAL) del fabricante del microcontrolador STM32. Algunas de estas funciones son: HAL_FLASH_Unlock (), HAL_FLASH_Program (), y HAL_FLASH_Lock ().

11. Anexo 2: Errores solucionados

11.1 Ruido en la señal de temperatura

Como se mencionó en la sección 8, la lectura que realiza el sensor de la temperatura externa suele presentar ruido lo que genera picos que no son reales y dificultan la toma de decisiones basadas en la temperatura del sistema. Como se observa en la figura 49, para solucionar este inconveniente se implementó un filtro en la señal leída “Termocupla” donde se descartan lecturas cuya variación “Abs_temp” sobrepase la variación máxima admitida “grados_variable”.

Sin embargo, había momentos donde la variación de temperatura era considerable (sobre todo en el enfriamiento durante el estado de emergencia) y el filtro descartaba estas lecturas sin importar que sean reales, entrando en un estado de bloqueo de la señal leída. Para solucionar este inconveniente se implementó una condición de desbloqueo del filtro, en donde si el numero lecturas descartadas “contador_mediciones” superaba un cierto valor “contador_mediciones_max”, se desactiva el filtro y se toma la última lectura de la señal de temperatura.

```
if (Abs_temp > grados_variable
    && contador_mediciones < contador_mediciones_max) {
    Termocupla = Termocupla_ant;
    contador_mediciones++;
} else {
    Termocupla_ant = Termocupla;
    contador_mediciones = 0;
}
```

Figura 49: Filtro de ruido en sensor temperatura

11.2 Separación de memoria

Se separó en dos lugares la memoria del equipo: una parte para trayectorias, y otra para datalog de eventos y uso del láser. Esto se debe a que las trayectorias a veces se

corrompían cuando se grababan nuevamente cada día en la memoria. Se aplican las mismas funciones para una nueva variable “Memoria_Data_Log” (figura 50)

```
HAL_StatusTypeDef ProgramarFlashDataLog() {  
    HAL_StatusTypeDef status = HAL_ERROR;  
    status = MY_FLASH_WriteN(0, Memoria_Data_Log, Data_Log_Size, DATA_TYPE_16);  
    return status;  
}
```

Figura 50: Separación de memorias

Se implementó un control en el grabado de información en la memoria, se espera que el sistema entregue un “OK” para corroborar que el grabado fue exitoso y que el código se siga ejecutando (figura 51).

```
//TODO: Flash  
  
void GrabarTrayectoriasEnFlash() {  
    HAL_TIM_Base_Stop_IT(&htim5);  
    HAL_TIM_Base_Stop_IT(&htim2);  
  
    HAL_StatusTypeDef aux_status = HAL_ERROR;  
    MY_FLASH_SetSectorAddrs(7, ADDRESS);  
    aux_status = ProgramarFlash();  
    int contador_progam = 0;  
    while(aux_status != HAL_OK || contador_progam > 5) {  
        aux_status = ProgramarFlash();  
        contador_progam++;  
    }  
    HAL_TIM_Base_Start_IT(&htim5);  
    HAL_TIM_Base_Start_IT(&htim2);  
}
```

Figura 51: control en escritura de memoria

También al leer la memoria de las trayectorias se hace una rutina de verificación de errores. La rutina consiste en leer la memoria tres veces y comprobar si las tres lecturas son iguales para considerar que la lectura fue exitosa (figura xx). Si la lectura no es exitosa en cinco intentos, no se actualizan las variables que almacenan el valor que se esta leyendo.

```

void inicializarMemoria(){
    uint16_t readData[1]={0};
    for(uint32_t i=0; i<MemoriaUsada;i++){
        MY_FLASH_ReadN(i,readData,1,DATA_TYPE_16);
        MemoriaCompleta[i]=readData[0];
        MY_FLASH_ReadN(i,readData,1,DATA_TYPE_16);
        MemoriaCompleta_1[i]=readData[0];
        MY_FLASH_ReadN(i,readData,1,DATA_TYPE_16);
        MemoriaCompleta_2[i]=readData[0];
        int contador_lectura = 0;
        while (MemoriaCompleta[i] != MemoriaCompleta_1[i]
            && MemoriaCompleta[i] != MemoriaCompleta_2[i] && contador_lectura < 5){
            contador_lectura++;
            MY_FLASH_ReadN(i,readData,1,DATA_TYPE_16);
            MemoriaCompleta[i]=readData[0];
            MY_FLASH_ReadN(i,readData,1,DATA_TYPE_16);
            MemoriaCompleta_1[i]=readData[0];
            MY_FLASH_ReadN(i,readData,1,DATA_TYPE_16);
            MemoriaCompleta_2[i]=readData[0];
        }
    }
}

```

Figura 52: Control en lectura de memoria

11.3 Casteo y limitación de consigna

Recordamos que los motores PaP pueden configurarse para moverse con micropasos. En el caso de este sistema en cuestión solo se considera utilizar el paso completo “1” o medio paso “1/2”, ya que micropasos más pequeños presentan problemas de precisión. Si consideramos como referencia de pasos mínima al medio paso, lo anterior equivale a decir que con medio paso se avanza una posición y con paso completo se avanzan dos posiciones.

Supongamos que en la memoria se almacenó una trayectoria que fue generada con medio paso, y posteriormente se cambia al modo paso completo y se busca que el sistema siga esta trayectoria. En esta situación es muy probable que se presente un problema, al momento de generar la trayectoria con medio paso se pueden tomar posiciones impares, como por ejemplo una posición de “15”, las cuales no pueden ser alcanzadas si el sistema se pasa al modo paso completo, ya que el sistema va avanzando cada dos posiciones, es decir que oscilara entre los valores de posición “14” y “16”.

Para solucionar este inconveniente se implementa un casteo implícito para transformar un valor impar a par. Como se observa en la figura 53, primero se divide la consigna por las posiciones que se avanzan en un pulso “PosicionesPorPulso” y posteriormente se multiplica por este mismo valor.

Para entender mejor lo que sucede se tomará el siguiente ejemplo: Supongamos que estamos en modo paso completo para el cual se dijo que con un paso se avanzan dos posiciones “PosicionesPorPulso=2” y que la consigna es “consigna=15”. Al ser la variable “consigna” una variable tipo entero “int”, cuando se hace la división se realiza un redondeo y se obtiene el siguiente resultado “consigna=consigna/PosicionesPorPulso=15/2=7”; posteriormente se realiza la multiplicación “consigna=consigna*PosicionesPorPulso=14”, siendo el resultado final un valor par que puede ser alcanzado por el modo paso completo.

Otra solución que se observa en la figura 53, es cuando se presenta un valor de consigna mayor a la posición máxima que puede ocupar el motor en cuestión “Posicion_Max[motor]”. Para solucionar esto simplemente se asigna a la consigna el valor de la posición máxima del motor

Es importante resaltar que en la articulación cero se acotó el valor de “Posicion_Max[0]” de forma tal que el giro máximo sea 270°, porque al haber colocado un solo sensor en la articulación cero se enredaban los cables.

```
void SetConsigna(int motor, long int consigna){
    //Divido y multiplico por PosicionesPorPulso por si queda mal grabado en un valor impar
    //al que el equipo no puede llegar cuando el MicroStep es 2 (es un casteo)
    consigna = consigna / PosicionesPorPulso;
    consigna = consigna * PosicionesPorPulso;
    //Asigno valor max en caso de que se halla grabado mal y consigna supere el valor del max
    if(consigna > Posicion_Max[motor]){
        Consigna[motor]= Posicion_Max[motor];
    }else{
        Consigna[motor]= consigna;
    }
}
```

Figura 53: casteo y limitación de consigna

11.4 Muestro de sensores FC

Inicialmente se optó por tomar las muestras de los sensores finales de carrera haciendo uso simplemente de las interrupciones, pero se presentaban fallas donde en ciertos momentos se dejaba de atender a dichas lecturas.

Para solucionar este problema se optó crear una función “sampling_sensores()” (figura 54) donde se utiliza un contador “contador_sampling_FC” y un limitador de este contador “lim_contador_sampling_FC” cuyos valores permiten que se pueda tomar muestras de los sensores FC cada 50 ms.

```
void Sampling_Sensores(){
    if(contador_sampling_FC >= lim_contador_sampling_FC){
        SetFinCarrera(0);
        SetFinCarrera(1);
        contador_sampling_FC = 0;
    }else{
        contador_sampling_FC ++;
    }
}
```

Figura 54: Muestreo de sensores

11.5 Configurar bluetooth desde la núcleo

Se agregaron comandos para poder configurar al bluetooth desde la núcleo, para evitar inconvenientes en los casos que no se haya configurado previamente. En la figura 55 se observa que para entrar en el modo configuración de bluetooth se debe enviar el comando “b4\n” o “b5\n”

```
case 'b':
    if (estado == 7) {
        switch (rx_buffer[1]) {
            case '0': // cerrar comunicaci+n
                subestado = 90;
                RepetirMensaje(huart);
                break;
            case '1': // para cancelar cualquier acci+ny volver a esperar en estado 7
                subestado = 0;
                RepetirMensaje(huart);
                break;
            case '2': // para mandar a grabar la Flash
                subestado = 1;
                RepetirMensaje(huart);
                break;
            case '3': //para borrar y poner en 0 toda la Flash
                subestado = 4;
                RepetirMensaje(huart);
                break;
            case '4': //para poner en modo configuracion el bluetooth
                subestado = 7;
                RepetirMensaje(huart);
                break;
            case '5':
                subestado = 8;
                RepetirMensaje(huart);
                break;
            default:
                subestado_paralelo = 0;
                RepetirMensaje(huart);
                break;
        }
    }
    break;
```

Figura 55: comando configuración de bluetooth

El comando nos envía al subestado siete u ocho del estado siete, donde se observa en la figura 56 que se realizan acciones como reiniciar el bluetooth o cambiar el BaudRate.

```
case 7: // Apgado y encendido del bluetooth (mientras se presiona el boton para conf)
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
    subestado = 0;
    break;
case 8: //Cambio el baudrate
    CorregirBaudRateUart6(38400);
    sprintf(tx_buffer, "AT+UART=115200,0,0");
    Transmitir_UART6();
    HAL_Delay(1000);
    subestado++;
    break;
case 9: //Reinicio bluetooth y vuelvo a estado 7
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
    CorregirBaudRateUart6(115200);
    subestado = 0;
    break;
```

Figura 56: Configuración bluetooth desde la núcleo

11.6 Falla del sensor de temperatura

Se realizó un sistema de detección de falla del sensor de temperatura porque a veces estaban mal soldados los terminales y no hacían contacto, entregando resultados erróneos al sistema.

Cuando el sensor de temperatura esta desconectado, la señal medida de temperatura almacenada en la variable “Termocupla” toma un valor extremadamente alto o bajo. Para detectar este evento (figura 57) se crean dos variables que almacenan un valor de temperatura máximo “temp_desconectado_max” y mínimo “temp_desonectado_min”, si “Termocupla” está fuera de este rango se considera que el sensor esta desconectado y se aumenta un contador “contador_sensor_desc” en cada iteración.

```
#define temp_desconetado_max 150 //valor de temperatura que significa sensor desconectado
#define temp_desconetado_min -9 //valor de temperatura que significa sensor desconectado

if(Termocupla > temp_desconetado_max ||
    Termocupla < temp_desconetado_min){
    contador_sensor_desc++;
}else{
    contador_sensor_desc = 0;
}
```

Figura 57: detección de falla en sensor de temperatura

En la figura 58 se observa que cuando el contador del sensor desconectado supera el valor de la variable “mediciones_desc”, se considera que el sensor está fallando “estado_sensor=1” y se pone la luz Led color violeta en modo parpadeo.

```
#define mediciones_desc 12 //midiendo cada 5 seg (TIM2) 12 veces da 60 seg

if(contador_sensor_desc > mediciones_desc && estado_sensor == 0
    && estado != 1){
    estado_sensor = 1;
    ColorVioleta();
}

//Sistema error med temperatura
switch (estado_sensor) {
case 0:
    estado_sensor = 0; //solo para hacer algo
    break;
case 1: //Apago RGB
    if(Flag_RGB == 1 && estado != 1){
        OffRGB();
        Flag_RGB = 0;
        estado_sensor++;
    }
    break;
case 2: //Enciendo RGB
    if(Flag_RGB == 1 && estado != 1){
        ColorVioleta();
        Flag_RGB = 0;
        estado_sensor--;
    }
    break;
}
```

Figura 58: Procedimiento frente a falla de sensor de temperatura

11.7 Recuperacion por fallo en sensores FC

Como se observa en la figura 59, al entrar en fallo por sensores de final de carrera, el equipo avanza a lazo abierto una cantidad de grados determinados para cada articulacion, espera 3 min y vuelve a intentar la maniobra homing. Esto también es para que si los equipos fallan vuelvan a ponerse en funcionamiento autónomamente.

```
//Estado 1: Emergencia////////////////////////////////////
case 1:
    switch (subestado) {
    case 0:
        OffFANS();
        OffLASER();
        switch (estado_anterior) {
        case 3:
            OnEnable(1);
            OnEnable(0);
            ReiniciarMotor(0, 0);
            ReiniciarMotor(1, 0);
            if(subestado_paralelo == 10){//este es el caso en que falla el sensor 0
                SetConsignaAngular(0,290);
            }else{
                SetConsignaAngular(0,60);
            }
            SetConsignaAngular(1,40); //Avanza una cierta cantidad de pasos
            //OffEnable(0);
            //OffEnable(1);
            ColorRojo();
            time_ant[2] = time[2]; //Hora
            time_ant[1] = time[1]; //Minutos
            time_ant[0] = time[0]; //Segundos
            subestado = 2;
            break;

#define Min_Desbloqueo_Emerg 3 //Uso esto para sacar de emergencia

        case 2: //Fallo Irrecuperable
            //HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
            segundos_actuales = time[2] * 3600 + time[1] * 60 + time[0];
            segundos_anteriores = time_ant[2] * 3600 + time_ant[1] * 60
                + time_ant[0] + Min_Desbloqueo_Emerg* 60;
            if (segundos_actuales > segundos_anteriores) {
                ConfigurarParaHoming(2);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
            }
            //HAL_Delay(10); //Espera 3 minutos y realiza homing nuevamente
            break;
        }
    }
    break;
```

Figura 59: Procedimiento frente a falla de sensor FC

11.8 Solucion de Watchdog

Inicialmente se optó por utilizar el watchdog interno de la nucleo, pero presentaba el inconveniente que a partir de los dos o tres días comenzaba a fallar. Por lo que finalmente se optó por utilizar un watchdog externo donde se manda un pulso al pic en cuestión cada vez que se se enciende el flag para medir temperatura (figura 60).

```
//Sistema Seg temp
if (Flag_hora) {
    Flag_hora = 0;
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_0); //Aca reinicio el pic
```

Figura 60: implementación de watchdog externo