Microcontroladores y

Electrónica de Potencia

Trabajo integrador:

Robot clasificador de objetos

Alumno: CEREDA, Mariano - PÉREZ, Ignacio

Legajo: 12254 - 12210

1 - Introducción

En el proceso de clasificación de frutas en un entorno industrial donde se manejan grandes volúmenes al día, es personal humano el que se encarga de desechar o separar aquellas que no se encuentran en óptimas condiciones para su comercialización. En general, las tareas de clasificación en diferentes industrias son bastante automáticas, pero muchas veces realizadas por humanos.

Figura 1 - Tarea que se proyecta automatizar con el uso de nuestro robot

Este proyecto se basa en un robot de dos ejes que se utiliza ,inicialmente, para proporcionar un prototipo automatizado que clasifique por color distintos objetos situados en su área de trabajo. Este robot puede situarse en varias zonas de la industria, ya que se puede programar para que clasifique cualquier tipo de objeto que se desee cambiando algunos sensores. Es decir, aunque en este caso se diferencian objetos por color, cumple con los principios básicos de una clasificación automatizada: reconocer objetos, sujetarlos y poder moverlos entre diferentes zonas según algún criterio, siendo este “criterio” la parte que puede ser más flexible y no necesariamente debe ser un reconocimiento por color.

Figura 2 - Proyecto de idea similar pero con robot ABB de 6 GDL

2 - Esquema tecnológico

En el siguiente diagrama se muestra la interacción entre los distintos módulos del sistema:

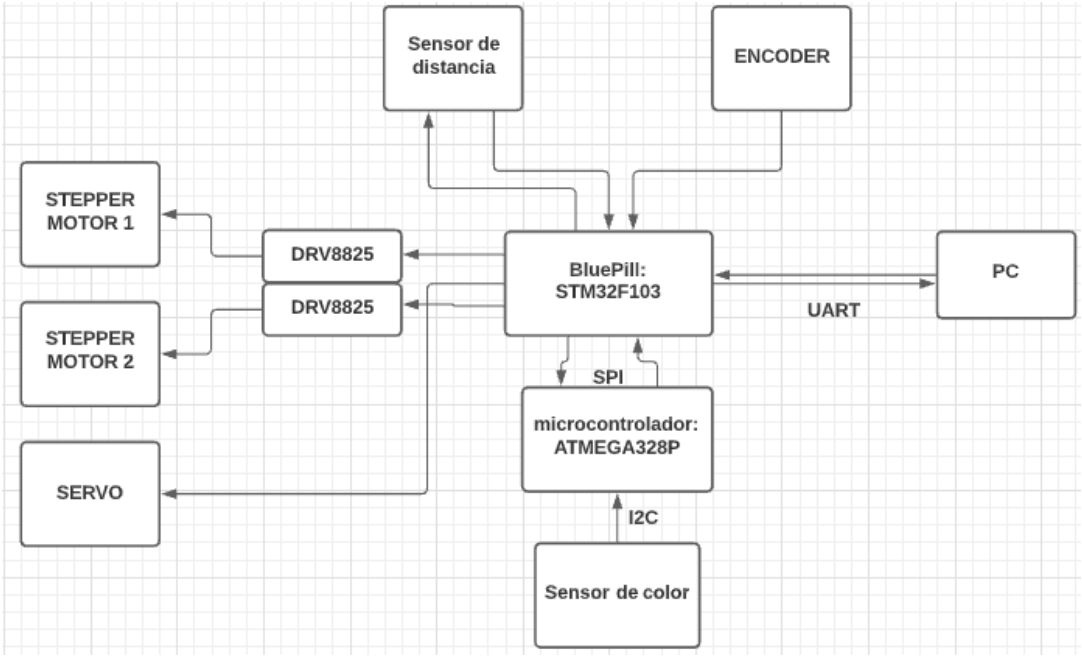

Figura 3 - Esquema tecnológico

3 - Detalle de módulos

Sensor de distancia ultrasónico - HC-SR04

Cantidad : 1

Voltaje de funcionamiento: 5V.

Funcionamiento: Un emisor piezoeléctrico emite 8 pulsos de ultrasonido (40KHz) luego de recibir la orden en el pin TRIG. Las ondas de sonido viajan a través del aire y “rebotan” al encontrar un objeto. Luego, la onda de rebote es detectada por el receptor piezoeléctrico. Posteriormente, el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada. Finalmente, el tiempo del pulso ECHO es medido por el microcontrolador y así se puede calcular la distancia al objeto. La referencia [6] contiene la información técnica del sensor.

Manejo con el microcontrolador: Primero, se genera la onda (de 10 microsegundos de duración) necesaria para que el emisor piezoeléctrico genere los pulsos ultrasónicos . Luego se hace uso de uno de los timers para poder medir el tiempo que dura la onda del pin ECHO en alto. Finalmente el valor obtenido se multiplica por una constante para pasar ese valor a unidades físicas de distancia.

Figura 4- sensor hc-sr04

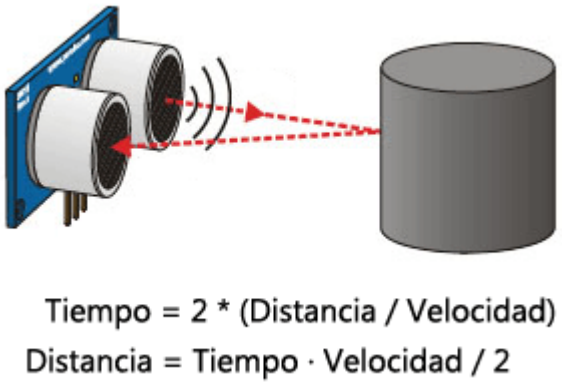

Figura 5 - Funcionamiento de sensor hc-sr04

Sensor de color - tcs34725

Cantidad : 1

Voltaje de funcionamiento: 5V - 3.3V.

Funcionamiento: El TCS34725 es un sensor óptico que incorpora una matriz de 3x4 fotodiodos, junto con 4 ADC de 16 bits de resolución que realizan la medición de los fotodiodos. La matriz de 3x4 está formada por fotodiodos filtrados para rojo, verde, azul, y sin filtro (clear). Los conversores ADC integran la medición de los fotodiodos, que es transferida a los registros internos del TCS34725. Luego, a través de I2C se realiza la comunicación con el microcontrolador para leer los registros que poseen el color medido como valores RGB.

En la referencia [8] se encuentra la información técnica del sensor.

Internamente el tcs34725 se muestra en la figura 6:

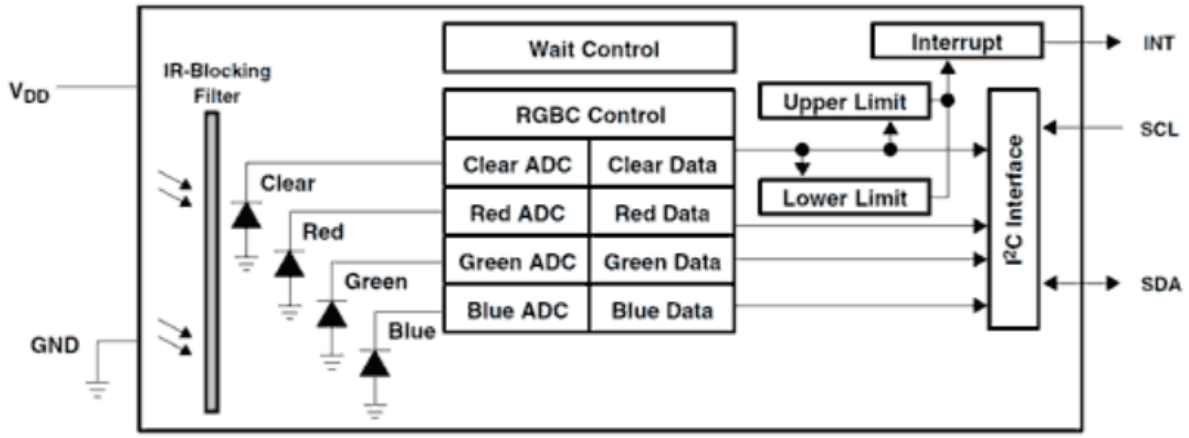

Figura 6 - Construcción interna del sensor de color.

Manejo con el microcontrolador: La complejidad interna del sensor conlleva a que se deban utilizar librerías para el correcto funcionamiento, y dichas librerías están disponibles solo para el microcontrolador ATMEGA328P. Es por eso que se hace uso de este micro para procesar los datos de color y estos luego se transmiten por comuniación SPI hacia el maestro (STM32F103C8T6).

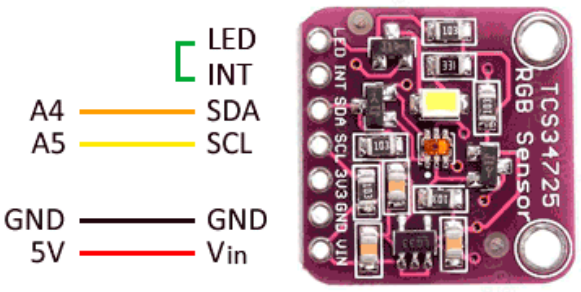

Figura 7 - Sensor de color tcs234725 y sus pines.

Encoder en cuadratura de 20 pulsos

Cantidad : 1

Voltaje de funcionamiento: 5V.

Funcionamiento: Este encoder es un dispositivo encargado de generar 2 canales de pulsos en cuadratura. El dispositivo se acopla al extremo de uno de los ejes y genera 40 pulsos (20 por cada canal) por cada vuelta que realiza. Estos son procesados por el microcontrolador y permiten que se

conozca tanto la posición angular como el sentido de giro del motor. Así, se puede verificar la pérdida de pasos.

En la figura 8, se muestran los 2 canales de pulsos que se generan a la salida del encoder cuando este está girando. Dependiendo de cual canal genera primero un flanco, será el sentido de giro del eje y por ende del motor.

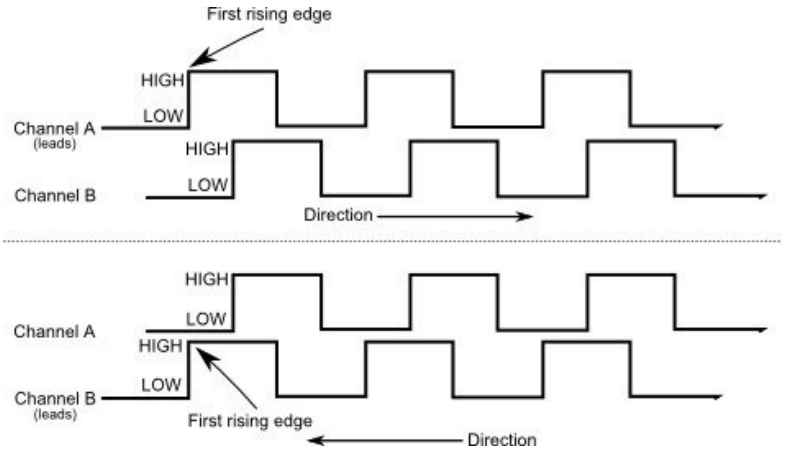

Figura 8 - Pulsos de salida del encoder en cuadratura

Manejo en el microcontrolador: El micro STM32F103, a diferencia de otros , presenta una opción de configuración especial de sus timers para el manejo de encoders en cuadratura. Esta configuración habilita 2 canales del timer correspondiente (vinculados a pines físicos del micro), y cada uno de estos se conecta a las salidas del encoder. En el caso de la aplicación, para aumentar la precisión se configura el microcontrolador no solo para detectar flancos de subida, sino también de bajada. Esto aumenta la cantidad de pulsos por vuelta de 40 (20 por cada canal) a 80 (40 por cada canal).

Figura 9 - Encoder rotativo en cuadratura de 40 pulsos por vuelta.

<u>Sensor fin de carrera tipo switch</u>

Cantidad: 2

Voltaje de funcionamiento: 5V

Funcionamiento: Los sensores finales de carrera tipo switch son simples pulsadores que se activan cuando algún elemento que se mueve a lo largo de un eje, alcanza el límite del mismo. En el caso de aplicación, solo se utilizan estos dispositivos en uno de los ejes.

El switch posee 3 pines, ya que presenta la opción de utilizarlo en modo NC (normalmente cerrado) o NO (normalmente abierto), además de la alimentación. En este caso se lo utiliza en modo NO.

Manejo con el microcontrolador: Se hace uso de las interrupciones externas De esta manera cuando el elemento que se mueve por el eje correspondiente (sería el gripper en este caso) llega al límite final del eje, se genera un flanco de subida gracias al fin de carrera. Esto activa la interrupción y detiene el giro del motor.

Figura 10 - Switch fin de carrera.

Driver - DRV8825

Cantidad : 2

Características: Posee 3 entradas principales: cantidad de pasos,dirección y habilitación o ENABLE. Todas se conectan al microcontrolador. Por otro lado posee 4 salidas que se conectan a cada una de las fases del motor.

Estos drivers nos facilitan el manejo de motores que trabajan con voltajes superiores a los del propio microcontrolador. Además, permiten regular fácilmente la corriente entregada al motor con un pequeño potenciómetro. En la referencia [9] se encuentra el datasheet del driver.

Manejo con el microcontrolador: Utilizamos solo 3 pines digitales de la bluepill para el manejo de los drivers. Uno de los pines, el que conectaremos a “dir”, se mantiene en alto o en bajo dependiendo del sentido de giro que se desee. Por otro lado, el pin de “step” recibe una cantidad de pulsos correspondiente a la cantidad de pasos que el motor debe realizar. Por último, en el caso de la activación de una parada de emergencia, también se deshabilita el funcionamiento de los drivers a través de su pin ENABLE, el cual en 1 deshabilita el funcionamiento y en 0 lo habilita.

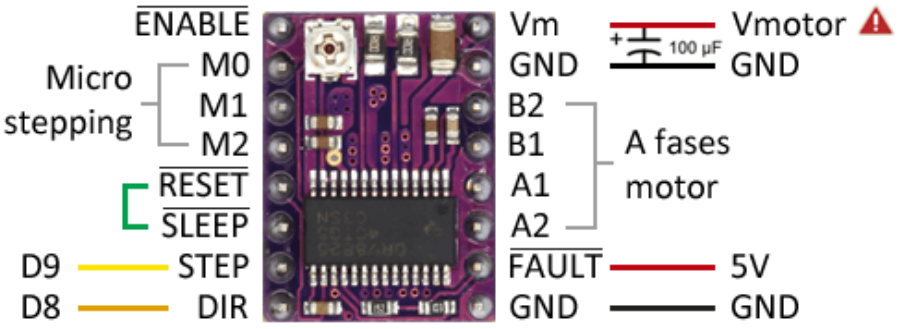

Figura 11 - DRV8825 con su respectivo patillaje.

Servo - micro servo 9g

Cantidad : 1

Voltaje de funcionamiento: 5V.

Funcionamiento: Los servomotores se controlan enviando un pulso eléctrico de ancho variable, o modulación de ancho de pulso (PWM), a través del cable de control. Hay un pulso mínimo, un pulso máximo y una frecuencia de repetición. Por lo general, un servomotor sólo puede girar 90° en cualquier dirección para un movimiento total de 180°. En la referencia [4] se presentan los datos técnicos del servo.

Manejo con el microcontrolador: Para manejar este servo se hace uso de uno de los timers del micro en el modo de funcionamiento “PWM generation” para poder generar la onda PWM. Para este servo específicamente se necesita generar una onda de 20ms de período, y luego variar el ciclo de trabajo entre 0 y 2.5ms para que el motor se mueva de 0 a 180°.

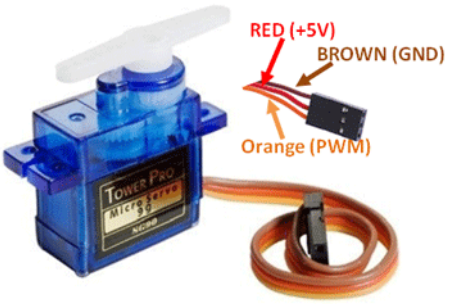

Figura 12 - Micro servo 9g

Motor paso a paso

Cantidad : 2

Corriente: Continua.

Voltaje de funcionamiento: 12V.

Máxima corriente: 1,4 A.

Funcionamiento: El principio de funcionamiento está basado en un estator construido por varios bobinados en un material ferromagnético y un rotor que puede girar libremente en el estator. Estos diferentes bobinados se alimentan uno a continuación del otro y causan un determinado desplazamiento angular que se denomina "paso" y es la principal característica del motor. En nuestro caso, usamos motores bipolares de 4 cables, que internamente son como se muestra en la figura 13. Es importante aclarar que al trabajar con 2 motores reciclados y antiguos fue difícil poder obtener los datasheets correspondientes. Por eso para poder conocer las corrientes máximas de trabajo se debieron buscar proyectos que utilicen los mismos motores y obtener de allí la información técnica.

Manejo con el microcontrolador: El micro se comunica directamente con el driver DRV8825 (que ya se explicó anteriormente) y este envía los pulsos a las bobinas del motor haciendo uso de PWM.

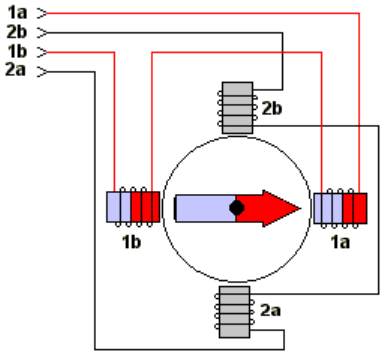

Figura 13 - Construcción interna de motor pap bipolar.

Conversor USB - TTL - Pl2303hx

Se utiliza para convertir de usb a niveles TTL y poder comunicar por UART la PC y el micro principal.

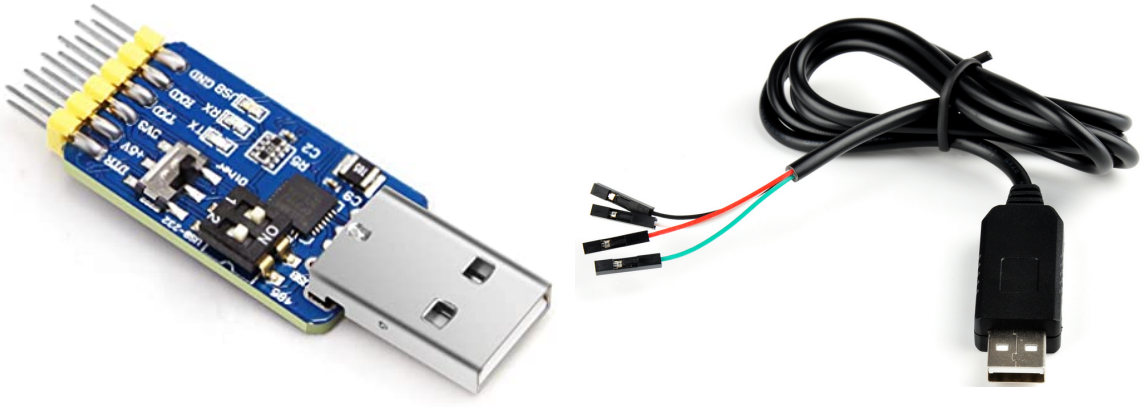

Figura 14 - Conversor usb a ttl

ST Link V2:

Se utiliza para poder comunicar el microcontrolador STM32F103C8T6 (Blue Pill) con la PC para así poder programar y debuggear el micro. Este programador/debugger implementa SWIM (Single Wire Interface Module) y JTAG/SWD (Serial Wire Debugging) para su comunicación.

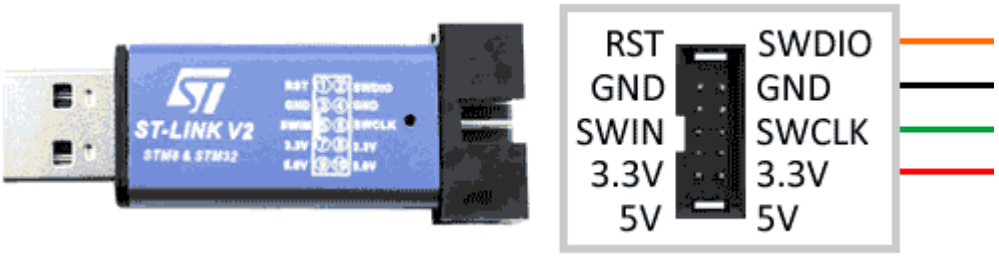

Figura 15 - ST Link

Por último, se muestra el esquemático, con las conexiones físicas entre los diferentes módulos del sistema (en el anexo [1] puede observarse con mayor claridad):

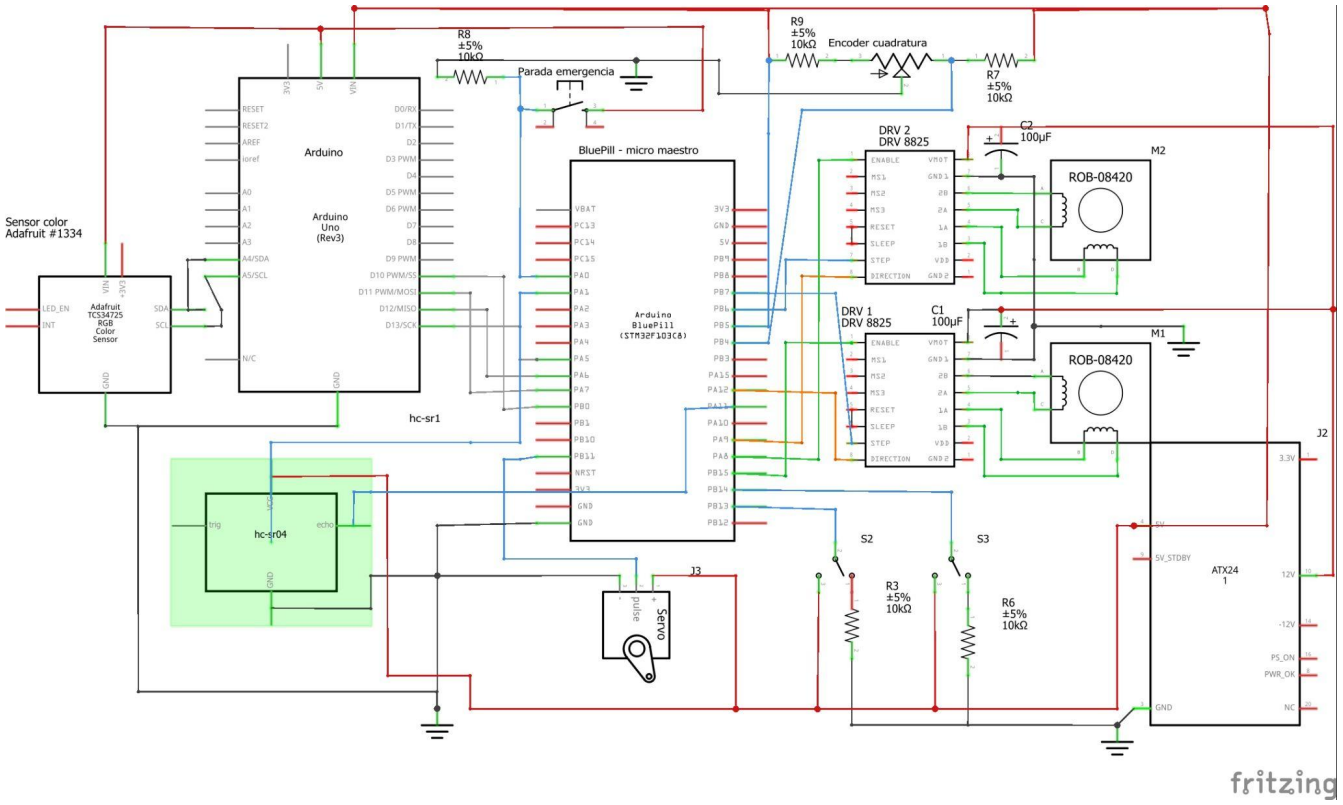

Figura 16 - Esquemático con conexiones.

4 - Funcionamiento general

Antes de comenzar a detallar el funcionamiento se muestra la estructura del prototipo. Para la estructura en general se utilizó madera, excepto para los tornillos sinfín (ejes) y varillas guía, que son de material metálico. Además, el gripper fue impreso en 3D en un material plástico. En la figura 17 se muestra un boceto de la planta del prototipo (no es un plano técnico de estructura, es un boceto para comprender mejor el funcionamiento).

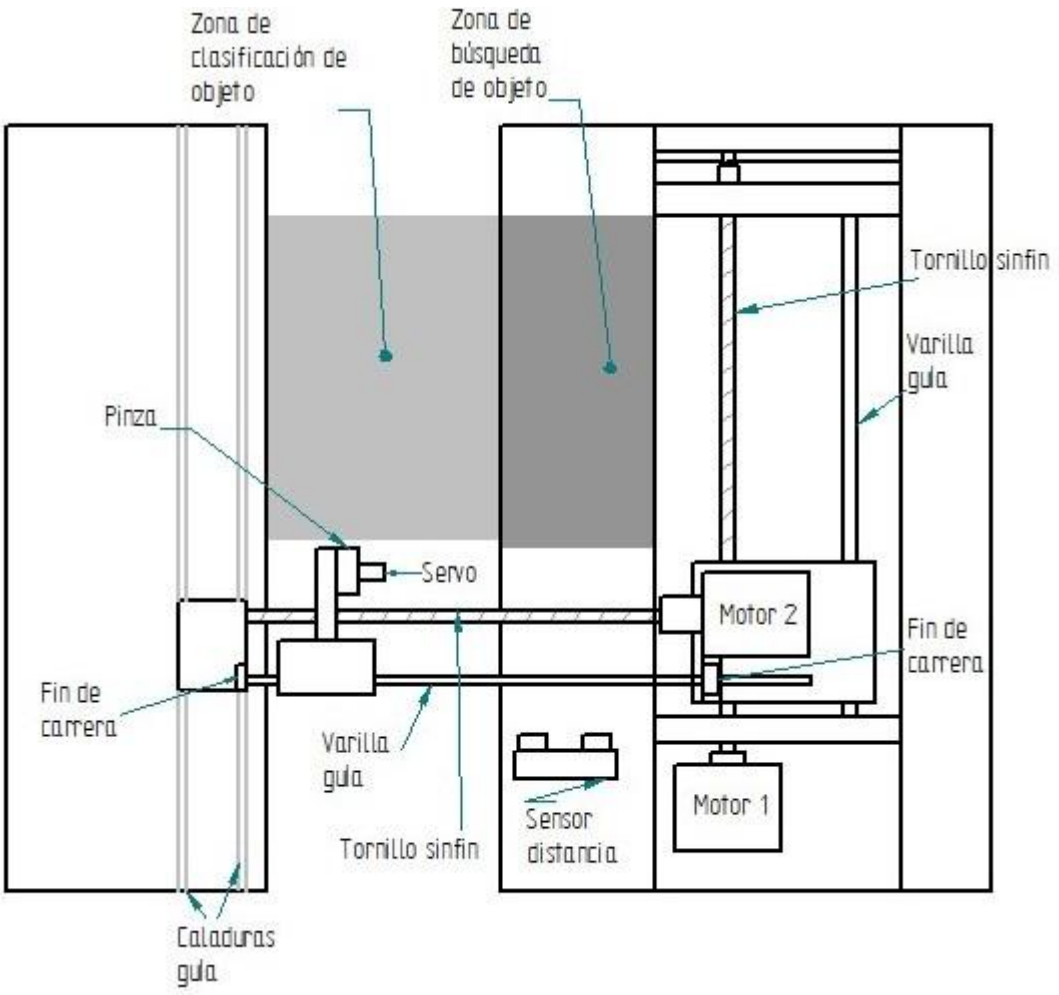

Figura 17 - Boceto del robot en planta.

La idea básica del funcionamiento es la detección, recolección y clasificación de objetos según su color. Es decir, se debe colocar un objeto a clasificar en la región correspondiente (gris oscura) , que está frente al sensor de distancia. El sensor mide la distancia a la cual se encuentra dicho objeto y mueve (con los motores paso a paso 1 y 2) el gripper a dicho lugar para sujetarlo. Tener en cuenta que el objeto a clasificar debe estar ubicado a más de 9 y menos de 26 centímetros del sensor de distancia (por los límites constructivos). Luego, el sensor de color (que se encuentra dentro de uno de los brazos de la pinza) detecta el color correspondiente y el programa le asigna una posición a la cuál se deberá transportar a ese objeto. Una vez transportado al lugar asignado (región gris, que a su vez posee divisiones internas para diferentes colores), la pinza suelta el objeto y queda en espera de una nueva detección del sensor de distancia.

Por otra parte, se cuenta con una parada de emergencia que detiene el funcionamiento del programa en cualquier posición donde se encuentre.

Además, se pueden configurar por UART diferentes funcionalidades:

- El encendido o comienzo del programa con el comando **":on;"**. Si se colocan objetos antes de que este comando haya sido ingresado, no serán reconocidos. Este solo se debe ingresar una vez para el inicio del programa.El reconocimiento y clasificación de objetos es automático y no necesita ningún comando extra.

- El apagado del programa con el comando **“:off;”**. Al ingresar este comando, el programa termina la tarea que está realizando y hace un “homing”. Si se quiere volver a usar el programa luego de esto, se deberá reingresar **“:on;”**.

- La configuración de la velocidad máxima a la que pueden llegar los motores con el comando **“:vx.x”**, donde “x.x” debe estar entre los límites [1 , 3]. Este valor ingresado modifica la máxima duración del pulso enviado al motor. Además, se muestra por pantalla el valor correspondiente en rpm que ha sido configurado con esa duración de pulso.

- Cuando se quiere volver a hacer uso del programa luego de haber activado la parada de emergencia, se debe ingresar el comando **“:restart;”** que realiza un homing y re configura las variables correspondientes para volver a utilizar el programa.

- Por último, el comando **“:color;”** muestra por pantalla el último color clasificado y la hora en la que esto se realizó.

En la siguiente figura se muestra el diagrama de flujo del funcionamiento:

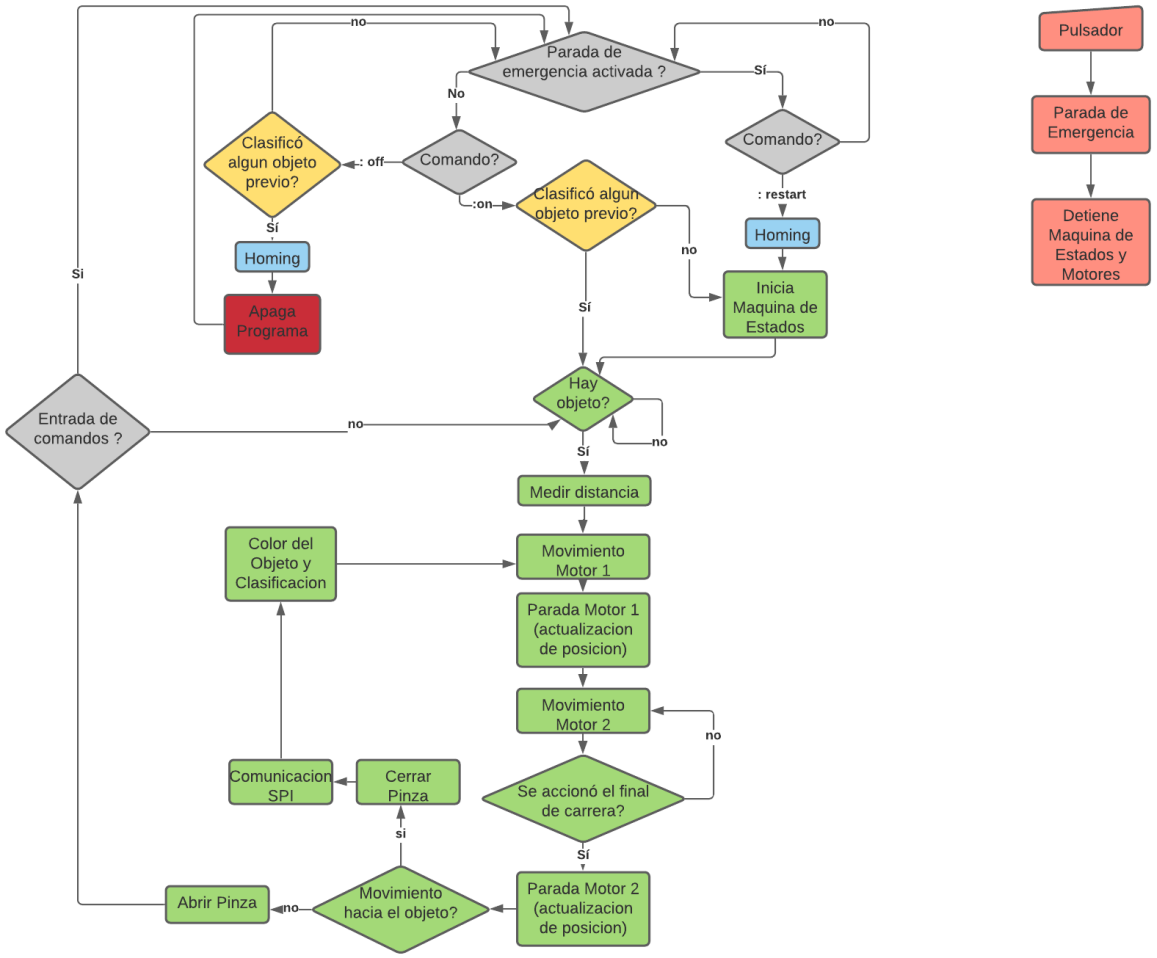

Figura 18 - Diagrama de Flujo del funcionamiento.

Debido a que la lógica del código se implementa como una máquina de estados, otra representación para comprender mejor su funcionamiento puede ser la presentada en la Figura 19:

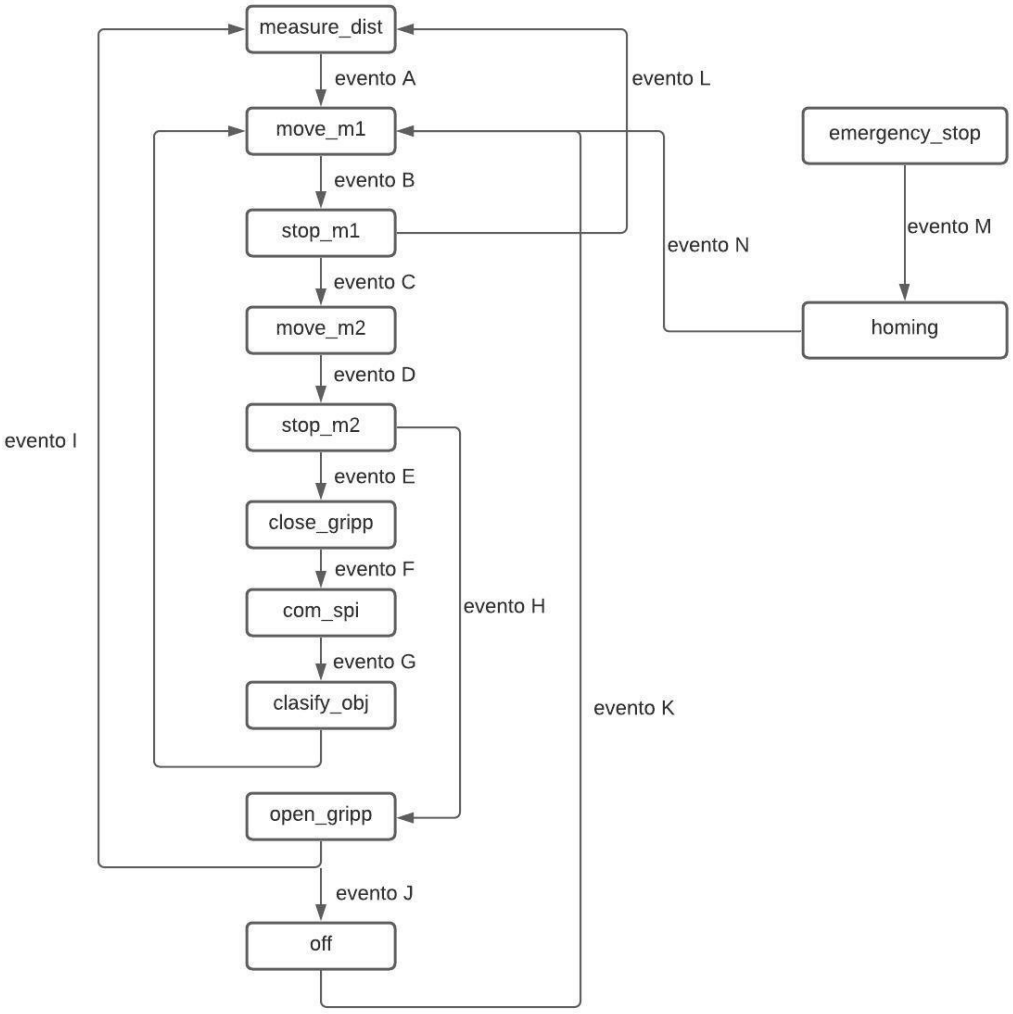

Figura 19 - Funcionamiento representado como máquina de estados.

Cada uno de los eventos mencionados en el diagrama de la figura 19, se describen a continuación. Tener en cuenta que cada evento es una condición que , en caso de cumplirse, permite que se pase al estado siguiente.

A = Detección de objeto dentro de los límites especificados. Deben realizarse 2 mediciones iguales consecutivas para verificar la correcta medición del objeto. Al cumplirse este evento, se pasa al estado **move_m1.**

B = Cálculo de pasos a realizar e inicializar PWM del motor 1. Luego de realizar este cálculo, se pasa al estado **stop_m1.**

C = Coincidencia de pasos medidos por el Encoder y pasos a realizar (previamente calculados) por el motor 1. Además la variable **homing_on** debe valer **0**.

D = Inicializar PWM del motor 2.

E = Detección de algún fin de carrera. La variable **mot2_dir** debe ser igual a 0. Luego de detectado el evento, además, inicializa el PWM correspondiente para el movimiento del servo.

F = Variable **dcycle** llega al valor **MAX_VALUE_Gripper,** que indica que el ciclo de trabajo es máximo para poder cerrar la pinza.

G = Fin de comunicación SPI.

H = Detección de algún fin de carrera. La variable **mot2_dir** debe ser igual a 1. El valor que posee esta variable es fundamental, ya que de esto depende si se pasa al estado **open_gripp** que abre la pinza o al estado **close_gripp** que la cierra.

I = Variable **dcycle** igual a **MIN_VALUE_Gripper**, que indica que el ciclo de trabajo es mínimo para poder abrir la pinza. Al cumplirse este evento, se pasa al estado inicial denominado **measure_dist.**

J = Variable **dcycle** igual a **MIN_VALUE_Gripper** y además**, v_off** debe haber modificado su valor a 1, lo cual se hace por uart con el comando **":off;".**

K = Actualización de variable **posx = 9 y mot2_dir = 1.**

L = Coincidencia de los pasos medidos por el Encoder y de los pasos a realizar por el motor 1 y variable **homing_on** debe valer **1.**

M = Ingreso de comando ":**restart;"** por uart que modifica las variables correspondientes.

N = Actualización de variables **start** = **1** y habilitación de motores.

Nota: **B,D, K y N** no son eventos en sí mismo, sino que actualizan variables solamente, pero para un mejor ordenamiento del código se los consideró como estados dentro del switch.

5 - Programación

5.1 - Configuración de periféricos,timers y puertos:

Al haber utilizado el IDE STM32CUBE IDE para programar la bluepill, la configuración de los periféricos se hizo mediante la interfaz gráfica que este IDE posee. A continuación, se muestran las características con las que fueron configurados los diferentes periféricos, interfaces de comunicación y puertos.

1- Configuración del clock del sistema:

Clock	Interno
Frecuencia clk (Mhz)	8
Frec. APB1 - Timers(Mhz)	2
Frec. APB1 - Periféricos(Mhz)	2
Frec.APB2 - Timers (Mhz)	1
Frec. APB2 - Periféricos (Mhz)	0.5

Esta configuración del clock se elige por las frecuencias que se debían manejar tanto en los periféricos como en los timers en general.

La frecuencia de timer más elevada que se utiliza es de 1 MHz (en el caso del timer 1 - input capture). Por eso, se optó por configurar en ese valor el bus "APB2 - timer clock", encargado del manejo de clock del timer 1. Como esa es la mayor frecuencia a utilizar, para el resto de los timers se configura el prescaler según el propio uso de cada temporizador.

Por otro lado, al tener como referencia básica la frecuencia de APB2, en el bus APB1 no se puede elegir cualquier configuración sino que se modifica el prescaler del propio bus pero siempre respetando los demás valores configurados para cumplir con la frecuencia en APB2.

2- UART:

Número de USART	2
Uso	Comunicación con PC
Baud Rate (Bits/s)	9600
Stop bits	1
Tamaño de trama (bits)	8
Bit de paridad	No
Interrupción	Si
Tipo de interrupción	Por recepción
TX	PA2
RX	PA3

3- TIMERS

Características	**Timer 1**	**Timer 2**	**Timer 3**	**Timer 4**
Uso	Sensor distancia	Servo	Encoder	Motores PAP
Modo	Input capture	PWM Generation	Encoder	PWM Generation
Canal	4	4	3 y 4	1 y 2
Prescaler	0	19	0	3
Período de conteo	65535	1000	65535	999
Interrupción	Si	No	No	Si

Tipo de interrupción	Por input capture	-	-	Pulso finalizado de PWM
Puerto - Pin	PA - 11	PB - 11	PB - 4 / PB - 5	PB - 6 / PB - 7

4- SPI:

Número de SPI	1
Uso	Comunicación con ATMEGA328P
Primer bit	MSB (Bit más significativo)
Prescaler	2
Baud Rate(KBits/s)	250
CPOL	Low
CPHA	1 Edge
SCLK	PA5
MISO	PA6
MOSI	PA7
SS	PB0

5- INTERRUPCIONES EXTERNAS:

Número de interrupción	**1**	**2**	**3**
Uso	Fin de carrera 1	Fin de carrera 2	Parada de emergencia
Puerto - Pin	PB-13	PB-14	PA-0
Grupo	EXTI_LINE[10:15]	EXTI_LINE[10:15]	EXTI0
Pull - down	Externo*	Externo*	Externo*

*Para los pull-down se utilizaron resistencias de 10kΩ.

6- PINES DIGITALES:

PA1	**trig**	Output
PA8	**enable motor 2**	Output
PA9	**direccion motor 2**	Output

PA12	**direccion motor 1**	Output
PB15	**enable motor 1**	Output

7-ORDEN DE PRIORIDADES DE INTERRUPCIONES:

Orden de prioridad	Tipo de interrupción
0	EXTI 0 - Parada de emergencia
1	EXTI [15:10] - Fines de carrera
2	TIM4 global int - PWM motores PaP
3	TIM1 input capture
4	SPI1 global int - Por recepción de datos
5	USART 2 global int - Por recepción de datos

Tener en cuenta que la interrupción con menor orden tiene mayor prioridad. Es decir, la interrupción con orden de prioridad 0 es la más importante, por ende siempre se tratará esta interrupción primero en situaciones que se activen varias al mismo tiempo. A continuación, se explica el orden de prioridad elegido para cada caso:

- La interrupción con mayor prioridad es la parada de emergencia porque claramente se activa en situaciones extremas que no pueden esperar.
- Por otro lado, los fines de carrera también son fundamentales ya que dan la señal de detención de uno de los motores, indicando que el gripper ha llegado al límite del eje y se necesita detener el movimiento. Si no se detiene, se puede dañar la estructura. Por esta razón se le da prioridad luego de la parada de emergencia.
- La interrupción global del timer 4 es fundamental, ya que es la que genera una interrupción cada vez que se envía un pulso al motor y en esta rutina de interrupción se cambia la velocidad de los motores instante a instante.
- Las interrupciones por SPI e input capture son situaciones de menor importancia respecto de las anteriores, por eso se les colocó este orden de prioridad.
- Por último, la interrupción por recepción de datos por UART se consideró la menos importante. Esto es debido a que los comandos no son de una importancia extrema: el

reinicio del programa, cambio de velocidad o petición del color clasificado tienen una importancia baja respecto a las situaciones nombradas anteriormente.

A continuación, se muestra la interfaz gráfica del IDE con los pines configurados:

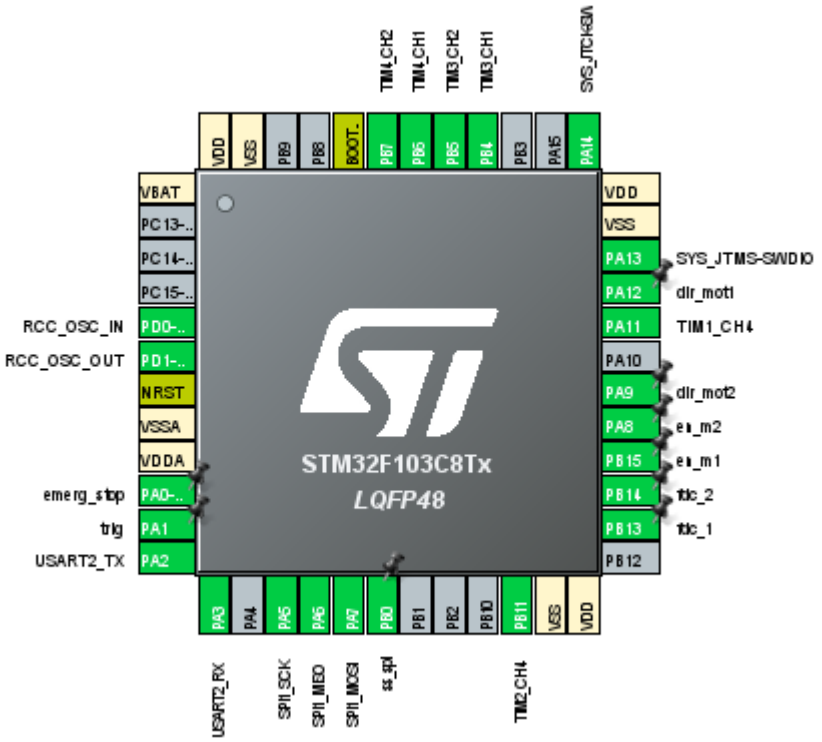

Figura 20 - Conexiones blue pill.

5.2 - Descripción de las Funciones:

Módulos

Además del archivo principal main.c , se tienen los archivos de tim.c, usart.c, gpio.c y spi.c donde se encuentran las funciones de configuración de los periféricos, timers , puertos y algunas de las funciones de elaboración propia.

Funciones de inicialización de periféricos, clock y librería HAL generadas por el IDE:

- **void SystemClock_Config(void)**
- **HAL_Init()**
- **void MX_GPIO_Init()**
- **void MX_TIM1_Init()**
- **void MX_TIM2_Init()**
- **void MX_SPI1_Init()**
- **void MX_USART2_UART_Init()**
- **void MX_TIM3_Init()**
- **void MX_TIM4_Init()**

Estas funciones contienen la información configurada en el entorno gráfico anteriormente.

CÓDIGO PRINCIPAL - BLUE PILL:

Funciones definidas en “main.c”:

- **Generate_trig_pulses():**

  Tarea: Se encarga de generar los pulsos de 10µs de duración, que se envían al pin “trig” del sensor de distancia. Así, éste puede emitir la señal ultrasónica correspondiente. Además, al final de la misma se habilita la interrupción por *input capture* del TIMER 1, de manera que cuando la señal ultrasónica regrese al sensor, este genere una interrupción en el timer.

- v**oid HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim):**

  Parámetro: *htim -> Puntero a timer que tiene habilitada la interrupción , que sería el 1 en este caso.

  Tarea: Mide la distancia a la cual se encuentra el objeto a clasificar relativa al sensor de distancia. Esta distancia es proporcional al tiempo que tarda en ir y volver la señal ultrasónica del sensor HC-SR04. Esta función lee los valores de conteo del TIMER 1 cuando se emite la señal y cuando se recibe la señal de vuelta. Restando ambos, y conociendo la velocidad de la señal, se puede calcular la distancia especificada. Si bien esta función no tiene retorno, modifica una variable global de vital importancia (posx).

- **uint16_t Calculate_steps(float** pos**):**

  Parámetro: pos -> Posición a la que se debe mover el motor 1. Punto flotante, con un dígito en la parte decimal.

  Retorno: steps -> Cantidad de pasos a realizar por el motor.

  Tarea: Transforma las posiciones “x” (en centímetros) en pasos a realizar por el motor. Es decir, si se quiere que el motor se mueva a una posición en “x” de 10 cm, la función transforma esta distancia a pasos . Para realizar este cálculo se tienen en cuenta la cantidad de pasos por vuelta de motor (200 en este caso) y la distancia de avance del tornillo sinfín por cada vuelta que realiza.

- **void Move_motor1(uint_16** steps**):**

  Parámetros: steps -> Pasos a realizar por el motor 1 (eje x)

  Tarea: Inicializar el PWM del motor 1, realizar el cálculo de pasos (es decir pasa de unidades de distancia a pasos a realizar por el motor) e inicializa la velocidad del mismo.

  Los valores de pasos que recibe la función están en valores absolutos, es decir relativos a la posición “0”, no a la posición anterior donde se encontraba el motor. Por eso, la función debe realizar la resta entre el valor de los pasos ya realizados anteriormente y los calculados actualmente, para que el motor se mueva a una distancia correcta. Es decir, si en el movimiento anterior se realizaron 500 pasos y el cálculo de pasos a realizar ahora es de 100, entonces el motor debe moverse 400 pasos (y el signo positivo o negativo determinará la dirección del movimiento).

- **void Move_motor2 (uint8_t** dir**):**

  Parámetro: dir -> Dirección de giro del motor (1 o 0).

Tarea: Inicializar el PWM del motor 2 y setear la velocidad con la que se moverá dicho motor. Para este segundo caso, no se tiene rampa de velocidad.

- **void HAL_GPIO_EXTI_Callback(uint16_t** GPIO_Pin):

  Parámetros: GPIO_Pin -> Número de pin donde se generó la interrupción.

  Tarea: Rutina de interrupción. Se ingresa a esta función cuando se detecta una interrupción en cualquiera de los fines de carrera y se modifica la variable fdc, para detener el motor 2 y pasar al estado siguiente. Además, se ingresa también cuando se detecta la interrupción de la parada de emergencia y cambia el "estado" de la máquina de estados.

- **void Clasify_object():**

  Tarea: Toma el valor del color recibido por SPI (contenido en un vector global) y , dependiendo del contenido de este, modifica el valor de la posición a la cual se debe mover el motor 1 (posx) para clasificar dicho objeto.

- **void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim):**

  Parámetro: *htim -> Puntero a timer que tiene habilitada la interrupción , que sería el 4.

  Tarea: Se ingresa a dicha función cada vez que el PWM del TIMER 4 (Channel 1) genera un pulso. Cada vez que se ingresa, se incrementa una variable para seguir el conteo de pulsos generados (y por ende , pasos del motor). Además, dentro de esta función se realiza la rampa de velocidad del motor, cambiando la frecuencia del PWM cada vez que se genera un pulso.

- **void Print_color():**

  Tarea: Sencillamente se encarga de imprimir por pantalla el último color detectado.

- **void Execute_emergency_stop()**

  Tarea: Se encarga de detener los tres PWM que se utilizan en el programa: los 2 correspondientes a los motores 1 y 2, y además el correspondiente al servo. Además, se deshabilitan los drivers.

- **int main(void):**

  Tarea: Ejecutar el loop principal de código e inicializar variables. Se implementó como una máquina de estados, por lo que solo se tiene un loop en todo el código . Dicho bucle contiene un switch donde cada "case" es un posible estado del programa. De esta forma se tiene un solo bucle en toda la codificación, haciendo más sencilla la implementación de situaciones especiales, como por ejemplo la parada de emergencia. Para comprender las tareas realizadas en esta función, se debe observar el gráfico de la máquina de estados con los eventos correspondientes.

Funciones definidas en "usart.c":

- **void HAL_UART_RxCpltCallback(UART_HandleTypeDef** *huart):

  Parámetros: *huart -> Puntero a la UART 2

Tarea: Rutina de interrupción de la UART 2, cada vez que se detecta un byte en el buffer de la UART, se ingresa a esta función y se rellena el vector "command" con dicho byte (excepto los caracteres de inicio y fin de trama). Posteriormente (fuera de esta función) se analiza el contenido y se actúa dependiendo del mismo. Tener en cuenta que para cada mensaje se debe cumplir con una trama que comienza con **":"** y termina con **";"**. Si no se cumple con esta trama, el mensaje no se interpreta.

- **void Analyze_command():**

  Tarea: Analiza el valor contenido dentro del vector "command" que fue previamente rellenado con el mensaje recibido por transmisión serie. Dependiendo del mensaje que se detecte, puede realizar 3 tareas: devolver mensajes por pantalla (mediante uso de "printf") o modificar variables globales para iniciar/finalizar el código o modificar la velocidad de los motores.

Funciones definidas en "spi.c":

- **void SPI_com():**

  Tarea: Se encarga de transmitir y recibir los bytes para solicitar la información del color detectado y el horario de la detección al atmega328P. Envía de a uno diferentes caracteres contenidos en un vector. Cada carácter realiza la petición de diferentes datos al micro esclavo:

  - "?", realiza la petición del color medido.
  - "h", realiza la petición de la hora.
  - "m", realiza la petición de los minutos.
  - "s", realiza la petición de los segundos.

  Dentro del bucle *for* que realiza la transmisión y recepción de datos hay un pequeño delay de 50 ms. Esto fue necesario para asegurar la correcta recepción.

- **void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef * hspi):**

  Parámetros -> Puntero a la spi correspondiente.

  Tarea : Cada vez que se recibe un dato por spi, se ingresa a esta función y se va rellenando un vector con los datos de color detectado ,hora,minuto y segundo en el que se realizó la transacción. Cuando ya no hay más transacciones, se guarda cada valor contenido en el vector en diferentes variables globales (color,hour,minute,second)

Funciones específicas de la librería HAL utilizadas:

- **HAL_TIM_PWM_Start (TIM_HandleTypeDef *htim, uint32_t Channel)**

  Parámetros: timer y canal del timer donde se genera el PWM.

  Tarea: Inicializa la generación de pulsos por PWM.

- **HAL_TIM_PWM_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel):**

  Parámetros: timer y canal del timer donde se genera el PWM con interrupción.

Tarea: Inicializa la generación de pulsos por PWM, con interrupción cada vez que se genere un pulso.

- **HAL_TIM_IC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel)**

  Parámetros: timer y canal el cual tiene configurado esta interrupción.

  Tarea: Inicializa la interrupción por *input capture*

- **__HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_4,0)**

  Parámetros: timer (htim), canal(TIM_CHANNEL_4) donde se genera el pwm. Valor "pulse" que modificará el duty cycle, de acuerdo a las siguientes ecuaciones:

  - FreqPWM = FreqTimer / (CounterPeriod + 1)
  - Period = (FreqTimer / FreqPWM) – 1 -> periodo del PWM
  - Pulse = ((Period + 1) * DutyCicle) / 100 – 1 -> duty cycle del pwm

  Tarea: Cambiar el duty cycle del PWM correspondiente.

- **HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)**

  Parámetros: uart, buffer donde se guardarán los datos recibidos (pData) y cantidad de bytes que se deben recibir (Size) para que se genere la interrupción.

  Tarea: Habilitar la recepción por interrupción de la UART correspondiente.

- **HAL_TIM_Encoder_Start(TIM_HandleTypeDef *htim, uint32_t Channel)**

  Parámetros: timer (htim) y canal (Channel).

  Tarea: Inicializar el modo encoder del TIMER correspondiente:

- **HAL_SPI_Receive_IT(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size)**

  Parámetros: spi (hspi), búffer donde se guardarán los datos recibidos (pData) y cantidad de bytes que se deben recibir (Size) para que se genere la interrupción.

  Tarea: Inicializar la recepción por interrupción de la SPI correspondiente.

- **HAL_NVIC_ClearPendingIRQ(IRQn_Type IRQn)**

  Parámetros: Grupo de interrupciones externas (**IRQn**)

  Tarea: Limpia los flags generados por las interrupciones externas correspondientes.

- **HAL_NVIC_EnableIRQ(IRQn_Type IRQn):**

  Parámetro: Grupo de interrupciones (**IRQn**)

  Tarea: Habilitar las interrupciones externas correspondientes.

Librerías utilizadas:

- **stm32f1xx_hal.h:** Es fundamental para el desarrollo de todo el programa, ya que contiene todas las funciones de la librería HAL. Éstas permiten que se pueda programar el microcontrolador con un nivel mayor de abstracción, ocultando la complejidad electrónica que puede tener el micro. En IDEs que generan código como *STM32-Cube* (que es el caso de

uso) estas funciones de la librería se generan automáticamente habiendo previamente realizado las configuraciones en una interfaz gráfica.

- **math.h** : Permite hacer uso de funciones específicas de matemática, como por ejemplo: obtención de valor absoluto, raíces,etc.
- **string.h :** Contiene varias funciones para aplicarlas sobre cadenas de caracteres. En el caso de aplicación, se hizo uso de la función “strcmp” , que recibe 2 cadenas de caracteres y compara si son iguales o no.
- **stdio.h:** Incluye las funciones, constantes y macros desde la librería de Entrada / Salida estándar.

CÓDIGO EN ARDUINO:

Librerías utilizadas:

- **"Adafruit_TCS34725.h":** Contiene todas las funciones y objetos necesarios para la comunicación mediante i2c entre el arduino y el sensor de color.
- **"ColorConverterLib.h":** Contiene las funciones necesarias para convertir los valores de color obtenidos del sensor de rgb a hsv, que es una forma mas adecuada para representar los colores ya que tiene en cuenta saturación y luminosidad, mas adecuado para el mundo real.
- **<TimeLib.h>:** Librería utilizada para obtener los valores de horario del día.
- **<SPI.h> :** Librería para realizar la comunicación SPI de forma más sencilla, utilizando funciones propias de arduino.
- **<Wire.h>:** Librería utilizada para la comunicación I2C con el propio sensor. Esta librería es utilizada por la clase **Adafruit_TCS34725.**

Funciones:

- **void setup():**

Utilizada para la inicialización de módulos y periféricos. Con la línea de **tcs.begin()** se verifica que la comuniación con el sensor de color se inicie de forma correcta. En cuanto a la SPI, **SPCR |= (1 << SPE)** coloca el bus SPI en modo esclavo, **SPCR |= (1 << SPIE**) inicializa la interrupción por recepción de spi y **pinMode(MISO, OUTPUT)** coloca al pin MISO (master IN, slave OUT) como salida. Por último, la función **setTime()** inicializa la fecha y hora.

- **void loop():**

Se encarga de la medición del color en sí y de la transformación de RGB a HSV. Primero con la función **getRawData()** propia de la clase **Adafruit_TCS34725** se obtienen los valores de color rgb y clear leídos desde el sensor. Luego, se dividen los valores de r,g y b por el valor obtenido del fotodiodo correspondiente a “clear” que no contiene ningún filtrado previo, es decir no filtra por r,g ni b, simplemente mide la luz recibida del objeto tal cual. Posteriormente, se pasan estos valores a la función “RgbToHsv” de la librería “ColorConverter” para obtener valores de color que sean mas “reales” o “legibles” por el ser humano (es decir, en formato HSV).

- **ISR(SPI_STC_vect):**

Es la parte principal de la comunicación con la blue pill. Cada vez que se detecta la llegada de un dato por el bus SPI ingresa a esta función. Dependiendo del carácter que se haya detectado en el búffer, escribe el registro **SPDR** con el dato correspondiente para luego responder al micro maestro: color,hora,minuto o segundo actual.

6 - Etapas de montaje y ensayos realizados

Se comenzó ideando el prototipo del proyecto, teniendo en cuenta que sería el movimiento de una pinza en un plano y que las consideraciones de peso y precisión en relación a paralelismo de ejes serían de gran importancia para el correcto funcionamiento del mismo.Luego, se plantearon bocetos del primer prototipo, el cual constaba de dos ejes formando un plano paralelo a la superficie, pero al no estar tan conformes con la primer idea se planteó hacer que la pinza se moviera en un plano perpendicular a la superficie (“x” y “z”). Esta idea fue cautivante en un principio, pero a la hora de construirlo se verificó que la estabilidad vertical estaba muy comprometida y que se necesitaría una estructura mucho más rígida de la que se podía realizar. Por lo que se volvió a la idea inicial pero con otro enfoque, lo que permitió llegar a un punto medio entre lo que se pensó en un principio y lo que realmente se podría hacer.

Primera prueba:

La primera prueba que se realizó fue en base al funcionamiento individual de cada periférico, con el microprocesador STM32F103C8T6 y los diferentes componentes. En el caso de los motores PAP Bipolares, se utilizó un código simple con delay y un pin set/reset para el step para comprobar el correcto funcionamiento de la conexión y los parámetros seleccionados. Para ello indagamos sobre los valores como la cantidad de pulsos sobre el pin step que se necesitan para dar una vuelta de 360° (200 en este caso) y se hicieron los cálculos correspondientes para poder determinar cuántos pasos le corresponden al motor dependiendo de la distancia medida por el sensor de proximidad.

Luego, como se mencionó anteriormente, se comprobó de forma tanto individual como en conjunto el correcto funcionamiento del sensor de proximidad HC-SR04. Este tenía la particularidad de que toda la información que se encontraba era sobre su manejo con micros AVR. Pero al analizar su datasheet, resultó ser un periférico fácil de configurar por lo que se pudo corroborar con rapidez el funcionamiento en conjunto de los motores y dicho sensor.

Segunda prueba:

Una vez que se comprobó el correcto funcionamiento, se colocaron los motores y el sensor en posición dentro del prototipo para corroborar que verdaderamente los valores arrojados y medidos eran correctos. A su vez, se comenzó con la puesta a punto del sensor de color la cual no fue tarea facil. La particularidad que presentaba este sensor es que todas sus librerías se encontraban para micros AVR. Al principio se pensó en crear funciones propias emulando las que aparecen en las librerías de arduino, pero al fallar en el intento, fuimos por otro camino: utilizar las librerías de arduino, con un arduino UNO, y hacer una conexión SPI de este con nuestro micro principal.

Tercera prueba:

La tercera prueba que se realizó fue en torno a la pinza y en la rutina completa que debería respetar. Esto quedó relegado a la tercera prueba porque se debió esperar un tiempo para fabricar la pinza con impresora 3D. Luego, se montó el servo y se comprobó su funcionamiento aislado mediante un código de prueba como con los demás periféricos. Pero el verdadero problema fue el lugar en el que se encontraria el sensor de color debido a que, para detectar de manera fiel el color, el objeto debía

estar muy cerca del mismo. Por lo que una posibilidad era dejar el sensor fijo en el espacio y que la pinza acercara el objeto al sensor para que detectara el color, luego de algunas pruebas llegamos a la conclusión que era difícil determinar de manera precisa cómo debía ubicarse la pinza para que a pesar de la geometría y dimensiones del objeto capturado, pueda medir su color correctamente. La otra prueba se hizo con el sensor dentro del agarre de la pinza, esto fue posible porque el servo no ejercía tanta presión y permitía que el sensor y el objeto siempre estén en contacto. Además, en robótica industrial, generalmente los sensores se encuentran cerca o incluso dentro del efector final (pinza).

Cuarta prueba:

En esta prueba se introdujeron tres elementos: encoder en cuadratura y dos switches mecánicos de fin de carrera. La implementación del encoder pasó por muchos procesos de errores propios donde por ejemplo al principio como el movimiento de los motores se hacía simplemente por un bucle *for* en vez de hacerlo mediante un PWM (modulación por ancho de pulso) el código que se implementó comenzaba con un movimiento según consigna del sensor de distancia y luego se comparaba lo que midió el encoder con la consigna real para después corregir si se habían perdido pasos. Lo que se implementó finalmente fue un movimiento mediante PWM donde la consigna de parada la asignaba directamente el encoder, por lo cual, cuando la consigna de movimiento era igual a lo medido por el encoder, el motor se detenía.

En referencia a los fines de carrera, lo que se realizó fueron dos implementaciones a raíz de los problemas que generaban las interrupciones por presencia de ruido. En primer lugar, los fines de carrera funcionaban en el eje del motor secundario para que de esta forma estas fueran la condición de detención del motor, sin embargo el criterio de parada se hacía de forma más rudimentaria (para fines prácticos, descartar errores de otras interrupciones o de hardware) la cual se basaba en bucle *for* que corroboraba si el pin pasaba de un estado bajo a uno alto marcando así un pulso de subida. Lo anterior descrito se utilizó para corroborar el correcto funcionamiento, luego se procedió a habilitar la interrupción por flanco de subida en ambos pines.

Quinta prueba:

En esta última prueba se realizaron cambios en el código para aplicar una parada de emergencia. Se implementó nuevamente el código en base a una máquina de estados, donde es mucho más sencillo comprender el flujo del programa, su funcionamiento en general y desde luego la puesta en marcha de una parada de emergencia. Para esto último, se utilizó un botón pulsador el cual mediante interrupción cambia el estado de la máquina y lo detiene completamente sea cual sea su estado actual, por lo que puede interrumpir cualquier proceso que se esté llevando a cabo.

Según la implementación anterior a la máquina de estados, donde se tenían varios bucles fuera del bucle principal, hubiese sido un problema la implementación de un botón de emergencia. Esto es debido a que hubiese sido necesaria la implementación de condiciones de detención en cada uno de estos bucles secundarios fuera de la función principal.

7 - Resultados, especificaciones finales

Para comprender mejor el funcionamiento del prototipo se explican con mayor profundidad algunas consideraciones de diseño e implementación:

- Rampa de velocidad de motor 1: Para evitar la pérdida de pasos del motor en el instante de movimiento inicial donde se debe vencer la inercia, se decidió aplicar una rampa de velocidad de manera tal que se comience con una velocidad baja y que ésta aumente conforme se va realizando el movimiento. Se impuso un valor de velocidad máxima a la cual debe llegar el motor, de manera tal que no se sobrepase. Los valores de velocidad al comienzo del movimiento y al final quedan definidos cuando el usuario ingresa por UART el valor del periodo del PWM que se encarga del manejo de los motores PaP.

  Por otro lado, la ecuación de la rampa de velocidad a elegir debía ser de cálculo rápido, de manera que se pueda realizar durante la generación de un pulso. Por eso luego de investigar un poco, se eligió la fórmula:

  ARR =ARR - (2*ARR)/(4*(steps_mot1-measured_steps)+1)

  Siendo "ARR" un valor que se guarda en el registro "AutoReload" que modifica el valor de la frecuencia del PWM (contiene el valor hasta el cual debe contar el timer en cada pulso). Por otra parte "**steps_mot1**" es la cantidad de pasos a realizar por el motor durante el movimiento actual y **"measured_steps**" se va actualizando instante a instante a medida que se van realizando los pasos. Como se puede observar, esta ecuación posee operaciones sencillas (sumas y divisiones) las cuales el microcontrolador puede procesar rápidamente. Cabe aclarar esto ya que, la mayoría de fórmulas que se suelen aplicar en estos casos contienen raíces, que implican mayor complejidad y tiempos de cálculo mayores.

  Debido a que solo se debe prever la pérdida de pasos en el motor que se mantiene fijo (motor 1), dicha rampa solo se tiene en cuenta para este y no para el motor 2. Este último siempre se mueve entre 2 puntos fijos y la pérdida de pasos no es una preocupación.

  En la referencia [1] hay mayor información sobre el desarrollo de las ecuaciones para dicha rampa de velocidad.

- Control de PWM de servomotor: Para el control del PWM se deben generar pulsos de 50Hz de frecuencia , la cual no se modifica en ningún momento. Lo que se modifica es el ciclo de trabajo, que varía entre 0 y 12.5%, haciendo variar el ángulo adecuado para abrir la pinza o cerrarla. Para lograr que se abra la pinza, el duty cycle va desde 0 hasta el 12.5% y para cerrar la pinza va desde 12.5% hasta 0.

Por otro lado, es importante aclarar detalles observados en cuanto a la precisión de los diferentes sensores:

- El *sensor de color* algunas veces podría considerar dos colores que son similares, como iguales, por ejemplo naranja y amarillo. De esta manera, los clasificaba como iguales pero en realidad eran diferentes. Si bien el sensor cuenta con una linterna propia para iluminar la zona donde se detectará el color y así se produzca el reflejo, si la luz del medio tampoco es adecuada la detección puede ser errónea.

- Por otro lado, el *sensor de distancia*. La precisión del sensor ultrasónico no es la mejor pero para el rango de trabajo funcionó correctamente. El error se podría considerar entre 1 y 50 milímetros aproximadamente. En un caso real, donde se podría trabajar con mediciones mayores a los 30 centímetros dicho sensor no sería recomendable, o bien podría serlo pero se debería agregar alguna otra condición de detección del objeto.

- Por último, *la precisión del encoder*. En el prototipo empleado se hizo uso de un encoder de 20 pulsos por vuelta (por razones de costos) pero lo ideal hubiese sido uno con no menos de 200 pulsos por vuelta. De esta forma se podría asegurar tener uno o más pulsos por cada paso del motor y realizar así una mejor corrección.

Finalmente, también se detallan los consumos en operación de algunos módulos del prototipo:

- Con respecto a los motores PaP, se los hizo trabajar con corrientes no mayores a 0.98A (que implica el 70% de su corriente máxima aproximadamente). Con estos valores de corriente se logró conseguir el torque adecuado para el funcionamiento. Además, si bien se hizo uso de un encoder para la verificación de la pérdida de pasos, se pudo corroborar durante el funcionamiento que incluso en situaciones de velocidades elevadas no hubo tal pérdida. Esto también se logró gracias a la rampa de velocidad que evita la pérdida de pasos al momento del inicio del movimiento, cuando se debe vencer la inercia inicial.

- Teniendo en cuenta el sensor de color, su consumo de corriente no superó los 100mA en situaciones de trabajo y 2µA en situaciones de modo de espera. Esto permitió que el sensor se pueda alimentar con la fuente que dispone Arduino, con una corriente máxima de 150mA.

- Debido a que se trabajó con una fuente con capacidad de hasta 10A de corriente, no se tuvo inconveniente alguno a la hora de alimentar los sensores y el micro Atmega328P con la misma fuente.

Conclusiones. Ensayo de ingeniería de producto.

Se logró que el prototipo cumpla con la tarea para la cual fue pensado, pero se deberían realizar mejoras para un caso de aplicación real. Uno de los objetivos básicos y más importantes era que el programa se comporte de forma autónoma, con la mínima intervención del usuario para su funcionamiento. Dicha tarea se cumplió correctamente y se lograron los objetivos.

El prototipo fue pensado a partir de la base de un robot clasificador donde en los casos industriales son robots con más GDL donde el movimiento puede ser más preciso y los sensores son mucho más sofisticados, pero en el caso de nuestro proyecto, se lograron las metas adaptando la tarea no solo en complejidad sino también en costos, logrando una aplicación adecuada de las enseñanzas adquiridas en la materia de Microcontroladores y Máquinas Eléctricas.

Se tiene en cuenta que el prototipo desarrollado está lejos de ser aplicado así como se presenta en una industria, por eso se proponen algunas mejoras para que pueda ser utilizado bajo dichas circunstancias:

- El agregado de un grado de libertad adicional a la pinza (sobre el eje vertical z). En la realidad, donde se pueden manejar objetos de diferentes formas y tamaños sería fundamental que la pinza no solo tome el objeto, sino también que pueda elevarlo del suelo una distancia considerable mientras este se encuentra en movimiento. Esto evitaría colisiones entre el mismo y el suelo o la superficie sobre la cual se está moviendo.

- El sensor de distancia debería ser de uso industrial, con mayor precisión y alcance. El HC-SR04 que se utilizó en este prototipo es de tipo educativo, lejos de cumplir con condiciones industriales. El rango de detección de un sensor industrial puede ser de hasta 1.7 metros, pero la precisión y robustez aumentan considerablemente. En cuanto a

especificaciones técnicas, la mayoría trabajan con valores de alimentación de 24V, y salidas de corriente no mayores a 100mA.

El precio puede encontrarse entre los $20.000 y $40.000 dependiendo de las características del mismo. La temperatura de trabajo puede elevarse hasta los 70°C.

- Se deberían mejorar las conexiones utilizadas y reemplazar los cables dupont por un conexionado de mayor calidad soldado al patillaje de los periféricos con algún nivel de aislamiento para evitar posibles entradas de ruido electrónico . Además, se debería reemplazar la protoboard por un circuito embebido.

- Algo fundamental en el ámbito industrial es poseer una interfaz hombre-máquina adecuada. En el prototipo desarrollado, algunas funciones como el inicio del programa o el homing se realizaron por comandos provenientes de UART , pero lo ideal y correcto sería que se activen por medio de pulsadores industriales. Las configuraciones de velocidad o peticiones del color clasificado podrían obtenerse haciendo uso de un sistema de tipo SCADA, permitiendo la comunicación entre el operario y el sistema programable de forma remota.

- La forma en la que se programaron las tareas de homing o apagado del programa es debido a la disponibilidad de materiales y presupuesto para el prototipo. En un caso industrial lo correcto sería tener fines de carrera en los 2 ejes. Esto haría que se cambie un poco la lógica de programación pero sería fundamental. Uno de los inconvenientes que tiene la lógica actual es que el homing se realiza al final del ciclo trabajo actual con el comando **“off”**, donde se apaga el programa y queda listo para ser utilizado luego. Es decir, el sistema queda en la posición de homing para luego comenzar otro ciclo de trabajo desde dicha posición. Pero habría problemas en una situación en la que el programa se apague por un corte inesperado de corriente por ejemplo. En dicho caso el sistema quedaría ubicado en una posición aleatoria y no se podría hacer el homing. Por eso, al agregar fines de carrera adicionales al eje número 1 esta situación se resolvería haciendo mover al motor hasta que se detecte el fin de carrera, quedando en la posición de homing listo para volver a trabajar.

- Los drivers deberían ser de uso industrial, adaptados para manejar motores de más elevada potencia. Obviamente, el tamaño de estos dependerá de la aplicación en sí. Como se aclaró en la introducción de este trabajo, el sistema de clasificación puede aplicarse no solo por color sino también usando otro tipo de criterio. Es decir que se puede estar hablando de motores que deban mover frutas o cajas con productos de mayor peso, debiendo aumentar la potencia de los motores.

- Si se desea aplicar el caso efectivamente a la clasificación de frutas, lo correcto debería ser utilizar un sistema de visión artificial. En esta situación no sólo se podría diferenciar por el color en sí, sino también por forma y textura. Si bien este prototipo se aleja bastante de haber implementado algo así,lo que se puede rescatar es la realización del procesamiento del color en un micro diferente del encargado de la lógica principal. De esta forma, se podría utilizar un sistema embebido más potente para procesamiento de las imágenes que se comunique con el maestro.

- El uso de un encoder con una mayor precisión sería fundamental. En el prototipo desarrollado se utiliza un encoder de 20 pulsos por vuelta por cuestiones de costo. Lo correcto sería utilizar un encoder de 200 pulsos por vuelta por lo menos, generando por así

un pulso (o más) por cada paso del motor, permitiendo aumentar la precisión para la corrección de la posición si fuese necesario.

Tener en cuenta que estos encoders de tipo industrial con mayor cantidad de pulsos por vuelta pueden tener un costo entre $10000 y $30000 dependiendo de la resolución, llegando a generar hasta 2000 pulsos en ciertos casos.

- Por último, el agregado de una cinta transportadora que posicione los objetos. En el prototipo utilizado, los objetos a clasificar son colocados en la zona gris oscura (ver figura 17) por una persona. En un caso real, estos objetos deberían ser transportados a dicha región por una cinta transportadora o medio similar.

Referencias

- [1] https://www.embedded.com/generate-stepper-motor-speed-profiles-in-real-time (rampa de velocidad)
- [2] https://www.luisllamas.es/motores-paso-paso-arduino-driver-a4988-drv8825/ (uso de drv8825)
- [3] https://www.ti.com/lit/ds/symlink/drv8825.pdf?ts=1625174345771 (hoja de datos drv 8825)
- [4] https://datasheetspdf.com/pdf/791970/TowerPro/SG90/1 (datos técnicos de servomotor)
- [5] https://controllerstech.com/hc-sr04-ultrasonic-sensor-and-stm32/ (uso de sensor de distancia)
- [6] https://datasheetspdf.com/pdf/1380136/ETC/HC-SR04/1 (información técnica sensor de distancia)
- [7] https://www.luisllamas.es/arduino-sensor-color-rgb-tcs34725/ (funcionamiento del sensor de color)
- [8] https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf (datasheet sensor de color)
- [9] https://www.ti.com/lit/ds/symlink/drv8825.pdf?ts=1627200613852&ref_url=https%253A%252F%252Fwww.google.com%252F (datasheet driver drv8825)
- [10] Apuntes cátedra microcontroladores y electrónica de potencia año 2020 - Facultad de ingeniería - Universidad Nacional de Cuyo.
- [11] STM32F10X Reference Manual. (Manual de referencia blue pill)

Anexos

En la carpeta adicional que se adjunta junto con el informe se presentan los anexos.