UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO:

Desarrollo de software de bajo nivel para un brazo robot portátil

Autora: Sara Marqués Villarroya

Tutor: Dr. Ranko Zotovic Stanisic

Cotutor: Dr. Ángel Francisco Perles Ivars

ETSID (UPV), Valéncia, Julio del 2016

ÍNDICE

1 OBJETO DEL PROYECTO

El proyecto es el Trabajo Final de Grado de la titulación de "Grado de Ingeniería electrónica y automática Industrial" de la Escuela Técnica Superior de Ingeniería del Diseño de la Universidad Politécnica de Valencia.

Este producto forma parte del desarrollo del proyecto EXO-FA (EXOsqueleton – Flail Arm1) es un proyecto de colaboración de la Universidad Politécnica de Valencia y el Hospital Universitario La Fe. Tiene como objetivo el diseño de un brazo robótico que permita realizar tareas determinadas a personas con discapacidad motriz, provocadas por causas degenerativas o accidente.

Dentro de este proyecto se abordarán los siguientes objetivos:

- Adaptación del código existente a las nuevas librerías HAL (*Hardware Abstraction Layer*), para conseguir la actualización a la nueva placa STMF32F429I.
- Diseño e implementación de control por voz, tanto de la velocidad como del movimiento, de un robot de asistencia a minusválidos, con la tarjeta Speak-UP.
- Implementación de una etapa de conversión Analógico-Digital para digitalizar el valor de la aceleración medida con un sensor en cada articulación.

Para conseguir tales objetos se empleará lenguaje de programación C, en el entorno Keil5 (plataforma para la programación de microcontroladores de la familia ARM-Cortex) y el programa de generación de código STM32 CubeMX. También se utilizarán programas de edición de audio (Audacity) para realizar el estudio biométrico de los comandos necesarios para el control por voz, consiguiendo así un menor margen de error.

2 ESTADO DEL ARTE

2.1 Robots para la asistencia de discapacitados

Se calcula que en actualmente en España hay aproximadamente 2.5 millones de personas con movilidad reducida en las extremidades superiores. La preocupación del ser humano por mejorar la calidad de vida de las personas que han perdido la movilidad ha llevado a desarrollar a lo largo de la historia diferentes dispositivos que pueden aliviar en parte distintos tipo de discapacidades, entre ellos, los robots para la asistencia de minusválidos.

En las últimas décadas el desarrollo de este tipo de brazos ha tenido una gran expansión. Se han implementado diferentes tipos:

- Brazos estáticos: están fijados, normalmente a una mesa. Pueden ayudar al paciente a comer, afeitarse…Su principal inconveniente es que no son transportables y por tanto solo pueden asistir al paciente en un lugar concreto.
- Brazos robóticos colocados en una silla de ruedas: no tienen el problema de la movilidad limitada, sin embargo, su principal inconveniente es precisamente que están ligados a una silla de ruedas. Los pacientes que pueden caminar y no necesitan silla de ruedas no pueden utilizar este tipo de brazos.
- Brazos robóticos fijados en los hombros: este tipo de brazos pueden tener diferentes aplicaciones:
    - Para usuarios sanos: provienen a los pacientes con un par de brazos adicionales. Llegan a tener 10 grados de libertad en las manos y tienen interfaz para la electromiografía y para el cerebro humano. Su principal problema es la autonomía y el peso.
    - Exosqueleto: el paciente controla mejor el robot porque siente la posición y la velocidad del brazo de forma más natural, además ayuda a la rehabilitación al mismo tiempo que se utiliza para cualquier tarea. Como principal inconveniente encontramos que los mecanismos para automatizarlo son más complicados, está limitado a los grados de libertad del brazo y al rango de éste y si hubiera algún problema es más probable dañar al usuario. Uno de los últimos ejemplos de este tipo de robots de asistencia es el

exoesqueleto pediátrico desarrollado por la Universidad Carlos III de Madrid el pasado mes de junio.

Ilustración 1 - Exoesqueleto pediátrico Universidad Carlos III

2.2 Control por reconocimiento del habla

La voz es una de las características que nos identifica como particulares y nos permite reconocernos con facilidad.

Se denomina control por reconocimiento del habla al proceso de extraer información lingüística de una señal de voz. Este proceso que el ser humano es capaz de llevar a cabo de forma automática es extremadamente complejo ya que intervienen procesos cerebrales relacionados con el lenguaje.

En los últimos 10 años el campo del reconocimiento de voz ha dado grandes avances gracias al auge en la investigación en el campo del aprendizaje de máquinas, a una renovada atención a métodos estadísticos y a la gran potencia de los ordenadores actuales.

Algunas de las ventajas que presentan estos sistemas son:

- Alta facilidad de medida ya que el coste del hardware necesario es bajo y la adquisición sencilla y cómoda para el usuario.
- Posibilidad de controlar un robot sin necesidad de cables ni un ordenador.

Por otra parte, los inconvenientes que presentan este tipo de sistemas son:

- La imprecisión, ya que el sistema puede no reconocer algunos vocablos y realizar una acción equivocada.
- Sensibilidad ante factores ambientales. El ambiente ideal para un programa de reconocimiento de voz es uno tranquilo, si el es ambiente ruidoso el programa podría fallar al reconocer las palabras.

- Necesidad de entrenar el sistema antes de utilizarlo, lo cual puede ser un proceso largo si se necesitan un número elevado de palabras.

El reconocimiento del habla es aplicable a una gran variedad de situaciones donde se requiera una comunicación hombre-máquina. Las aplicaciones más comunes que podemos ver son los agentes telefónicos que interactúan con el usuario mediante voz (teléfonos inteligentes), en los cuales el programa no sólo reconoce el habla, sino que interactúan con el usuario de forma oral y natural.

Uno de los avances más significativo en los últimos años ha sido el desarrollo de los estudios sobre la biometría de la voz. Estos estudios reconocen la voz mediante comparaciones de patrones únicos de la voz tales como características fisiológicas y hábitos lingüísticos, por ejemplo teniendo en cuenta la pronunciación, énfasis, velocidad al hablar o acento.

3 ACTUALIZACIÓN DE SOFTWARE A LA PLACA STM32F429 Y A LAS LIBRERÍA HAL

3.1 Introducción.

Los sistemas embebidos pueden ser programados de forma tradicional, o mediante el uso de sistemas operativos de tiempo real, diseñados específicamente para atender a los más exigentes requerimientos de velocidad de respuesta y eficiencia.

Tradicionalmente, un programa en C consta de una inicialización (Setup) y un bucle infinito (While). Finalmente, se programan rutinas para atender a interrupciones externas, temporizadores, etc.

Este esquema de funcionamiento secuencial es sencillo y totalmente adecuado cuando todas las tareas a realizar tienen una prioridad parecida.

Sin embargo, en sistemas embebidos, es habitual que el mismo micro-controlador tenga que atender entradas y rutinas de control, con prioridad muy diferente. Por ejemplo, siempre es prioritario supervisar el movimiento de la máquina controlada por el microcontrolador, pero no lo es tanto el atender a la señalización (encender las luces de marcha y paro, avería, etc).

En estos casos, es más eficiente emplear un sistema operativo en tiempo real, conocido por sus siglas RTOS (Real Time Operating System), ya que permite crear hilos de ejecución con diferentes prioridades.

La utilización de estos sistemas operativos es sencilla, ya que solo hay que incluir su código fuente en C en el proyecto, y compilar el conjunto. Ocupan unas decenas de KB, y por tanto no suponen un problema de espacio.

3.2 Adaptación a estándares internacionales y modernización del código fuente

Durante la realización del proyecto se ha estado trabajando en el código ya existente a los nuevos estándares que van implantándose en la industria. Los trabajos se han centrado en:

- Utilización de una capa de abstracción de alto nivel HAL, como middleware de acceso a los periféricos del microcontrolador ARM® Cortex®-3.

3.3 Capa de software CMSIS-RTOS

Con la aparición de numerosas empresas desarrolladoras de RTOS para sistemas embebidos, cada vez se hacía más trabajoso migrar una aplicación de un microcontrolador a otro, o simplemente de un RTOS a otro.

El problema llegó a tal punto, que muchas veces los propios fabricantes de chips definían ya sus propios RTOS. Todo ello conllevaba un elevado coste para las empresas.

En 2012, los desarrolladores de sistemas operativos de tiempo real y fabricantes de microcontroladores ARM®Cortex®-M como los STM32F2 y STM32F4 utilizados en este proyecto, acordaron utilizar un nuevo estándar "de facto" liderado por la empresa ARM®. Este estándar se llama CMSIS-RTOS.

CMSIS-RTOS es, por tanto, un estándar RTOS definido para mejorar la portabilidad entre las aplicaciones de microcontroladores. El cumplimiento de esta norma se puede lograr por diseño (en el caso de nuevos RTOS) o mediante una capa de adaptación en la capa superior de un RTOS existente, esta última es la que se adoptará en el proyecto.

En este proyecto para adaptarnos al estándar CMSIS-RTOS utilizaremos una capa superior desarrollada por STMicroelectronics, que ofrece una interfaz de tipo CMSIS-RTOS hacia la aplicación de control.

3.4 Drivers de alto nivel para el hardware periférico del ARM. Capa "hardware abstraction layer" HAL.

Todos los fabricantes de microcontroladores ARM® Cortex®-M ofrecen sus propias librerías de acceso a los periféricos del microcontrolador. El hardware al que nos estamos refiriendo incluye los buses SPI, I^2C, USART, USB, Ethernet, etc. Este hardware no está definido dentro del estándar ARM, y por tanto su diseño y forma de uso es totalmente dependiente de cada fabricante. De la misma forma, las librerías que ofrecen a los desarrolladores son diferentes, dependiendo del fabricante.

En el caso de los microcontroladores STM32F2 y STM32F4 del fabricante ST Microelectronics utilizados en este proyecto, el fabricante ha pretendido evolucionar sus librerías *Standard Peripheral Libraries*, a otras que ofrezcan una interfaz de más alto nivel.

Con ello se quiere conseguir que pasar de usar un microcontrolador ARM del fabricante ST a otro del mismo fabricante sea una tarea trivial. El fabricante ST Microelectronics se compromete a desarrollar estas librerías de alto nivel (HAL) para todo sus microcontroladores ARM, de forma que su capa superior sea siempre la misma y que todas las particularidades de cada microcontrolador ARM queden tratadas dentro de cada librería HAL.

El esquema general sería:

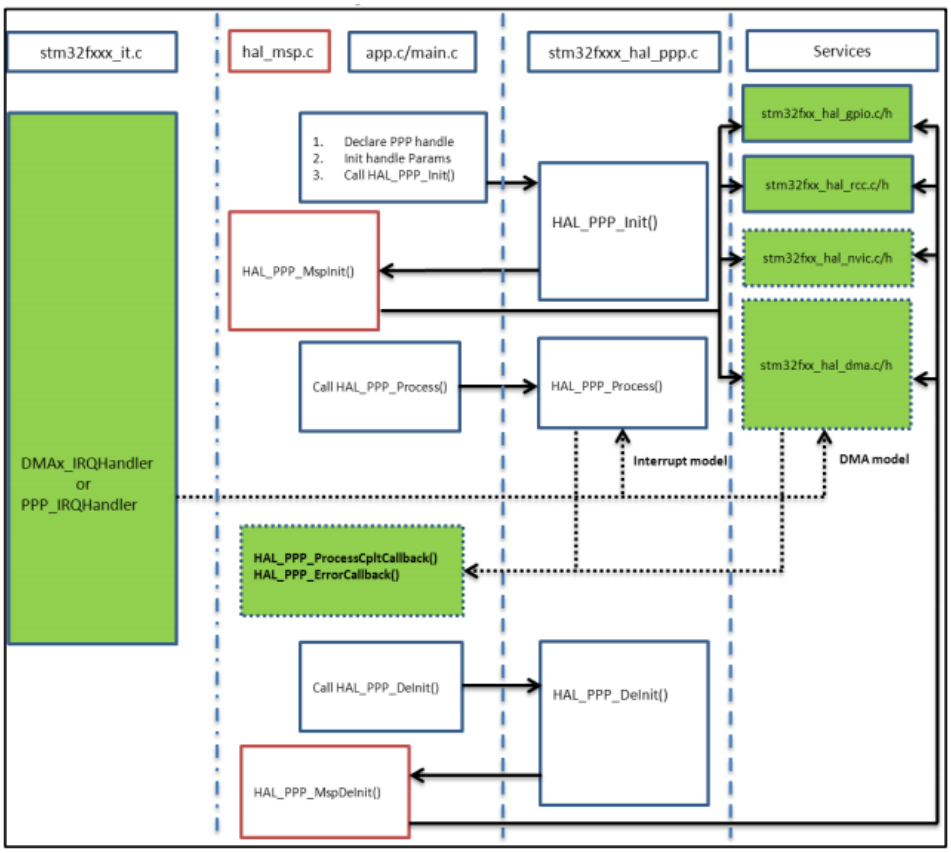


Ilustración 2 - Esquema general de las librerías HAL

La adaptación del código a estos nuevos estándares de la empresa STMicroelectronics ha sido uno de los objetivos de este proyecto.

3.5 Estudio de las librerías existentes del fabricante ST.

Actualmente encontramos 3 grandes conjuntos de librerías en la familia STM: las antiguas *Standard Peripheral Libreries*, las *Hardware Abstraction Layer* (HAL) y las *Low Layer APIs* (LL-APIs).

En la Tabla 1 vemos una comparación entre las prestaciones de cada una de las librerías:

		Portabilidad	Optimización	Facilidad de uso	Preparación	Nivel de hardware que cubre
Standard peripherical library		++	++	+	++	+++
STM32 CUBE	**HAL APIs**	+++	+	++	+++	+++
	LL APIs		+++			++

Tabla 1 - Comparación entre las prestaciones de las librerías

Algunas de las ventajas de utilizar las librerías HAL frente a la antiguas *Standard Peripheral Libraries* son:

- Facilidad de uso, se pueden utilizar los controladores de HAL para configurar y controlar cualquier periférico conectado con su STM32 MCU sin un previo conocimiento profundo del producto.
- Las librerías HAL proporcionan una API intuitiva y lista para su uso con soporte de *polling* para configurar los periféricos, Soporta tanto el modelo de programación orientada a interrupción como orientada a DMA (acceso directo a memoria) para dar cabida a todos los requisitos de la aplicación , lo que permite al usuario final construir una aplicación completa llamando a unas pocas API's
- La transferencia de datos puede usar el modo por bloques (polling) o sin modo por bloques (modo interrupción o modo DMA).
- Cuenta con manejador de errores a través de mecanismos de detección de fallos en periféricos y de timeout.
- Tiene una arquitectura genérica, con una inicialización muy rápida lo que permite a los clientes concentrarse en la innovación.
- Conjunto genérico de API's con total compatibilidad a lo largo de todas las series/líneas STM32, se elimina los trabajos de portabilidad entre MCUs de los STM32.
- Las API proporcionadas dentro de las librerías HAL están orientadas a funciones y no requiere un profundo conocimiento de la operación periférica.
- Las API proporcionadas son modulares. Incluyen la inicialización, la operación IO y funciones de control. El usuario final solo tiene que llamar a la función *init* y a continuación, iniciar el proceso llamando a una de las funciones de operación IO (escribir, leer, transmitir, recibir...).

- La mayoría de los periféricos tienen la misma arquitectura.
- El número de funciones que se requieren para construir una aplicación completa y útil es muy reducida
- A modo de ejemplo, para construir un proceso de comunicación UART, el usuario sólo tiene que llamar HAL_UART_Init ( ) y ya se puede utilizar HAL_UART_Transmit ( ) o HAL_UART_Receive ( ) .

Finalmente el otro gran conjunto de librerías de bajo nivel previstas por STM para la serie F4 son los LL-APIs (Low Layer APIs) cuyas características previstas serían:

- Alta optimización
- Acceso a nivel de registro.
- Código muy corto
- Seguirá muy de cerca el manual de referencia.
- Debug a nivel de registro.

En cambio cuentan con algunas limitaciones:

- Específico para cada uno de los dispositivos, no serán compatibles entre diferentes series
- No se implementará para periféricos complejos (como USB)
- La falta de abstracción significa que los desarrolladores deberán entender el funcionamiento de los periféricos a nivel de registro.
- Hoy en día solo es accesible para la serie STM32 L4.

Para la serie F4 está previsto que aparezca en enero de 2017. Actualmente no está en servicio.

Otro de los puntos a tener en cuenta a la hora de elegir las librerías de trabajo es el desarrollo de actualizaciones por parte del fabricante a los nuevos hardwares en desarrollo. En la Tabla 2 se muestra las actualizaciones previstas por el fabricante STMicroelectronics en cada uno de los conjuntos de librerías.

Offer	STM32F4	STM32F7							
	STM32 F0	STM32 F1	STM32 F3	STM32 F2	STM32 F4	STM32 F7	STM32 L0	STM32 L1	STM32 L4
STM32Snippets	Now	N.A.	N.A.	N.A.	N.A.	N.A.	Now	N.A.	N.A.
Standard Peripheral Library	Now	Now	Now	Now	Now	N.A.	N.A.	Now	N.A.
STM32Cube HAL	Now	Now	Now	Now	Now	Now	Now	Now	Now
STM32Cube LL	May 2016	Feb 2017	June 2016	Jan 2017	Jan 2017	Dec 2016	April 2016	June 2016	Now

Tabla 2 - Actualizaciones previstas para cada conjunto de librerías de ST

Tal y como se ve en esta tabla las librerías Standard no tienen previsto el desarrollo a los nuevos componentes STM32F7, por tanto su uso limitaría el proyecto al hardware actual. En cambio las librerías LL no estarán disponibles hasta enero de 2017, por tanto el proyecto se vería retrasado a esa fecha.

Tras estudiar las principales características de los tres conjuntos de librerías se ha decidido trabajar con las librerías HAL, ya que nos permiten cambiar de microcontrolador sin tener que reprogramar la inicialización de los componentes, cuenta con la ayuda del software STM32 Cube MX que nos facilitará el trabajo de programación y está previsto que cuente con las actualizaciones para el nuevo hardware en desarrollo de ST.

3.6 Cambio de placa del STM32F4-Discovery a la STM32F429I

Las primeras pruebas del sistema se realizaron con la tarjeta STM32F407-Discovery. Al analizar los resultados obtenidos se vio que la velocidad del procesador, así como la memoria de esta placa eran insuficientes para el tratamiento de los datos. Como consecuencia de ésto se decidió cambiar a la tarjeta STM32F429I que contaba con mejores prestaciones y evitar así la pérdida de datos como había ocurrido con la anterior.

Veamos las ventajas que ofrece la 429 frente a la Discovery.

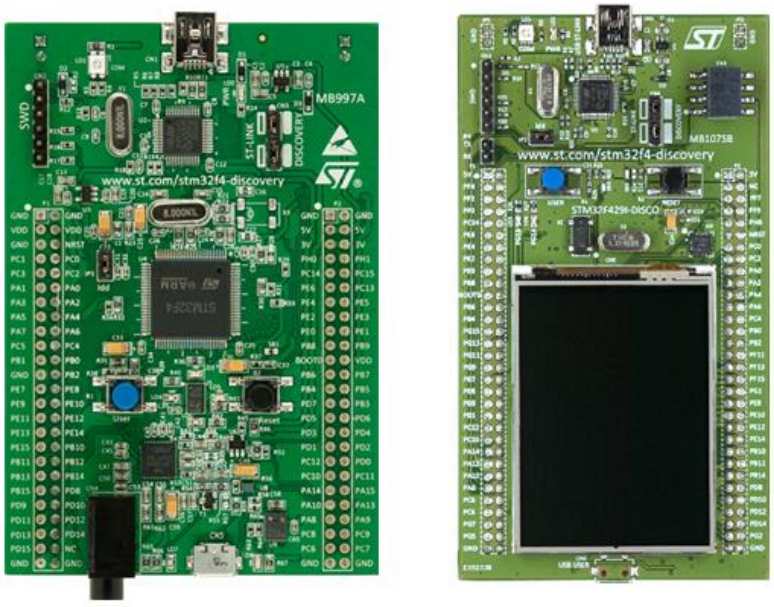


Ilustración 3 - STM32F407 Discovery y STM32F429

Si comparamos las características de las dos placas, vemos que los aspectos más importantes de cada una serían:

STM32F407 DISCOVERY	STM32F429I
• Procesador a 168 MHz • Memoria: 1Mb de memoria Flash. • 192 KB de RAM en el integrado LQFP100 • Integrado en la placa el ST-LINK/V2 que se puede activar como un kit independiente con conector SWD para programar en modo Debug. • Potencia de la placa a través del bus USB o desde una fuente externa de 5 V • Se puede proporcionar a aplicaciones externa una potencia de 3 o 5 V • Sensor del movimiento con salida digital en 3 ejes de acelerómetro LIS30DSH, ST MEMS	• Procesador a 180 MHz • Memoria: 2Mb de memoria Flash • 296 KB de RAM en el integrado LQFP144 • Integrado en la placa el ST-LINK/V2 que se puede activar como un kit independiente con conector SWD para programar en modo Debug. Se puede activar un puerto Debug virtual. • Potencia de la placa a través del bus USB o desde una fuente externa de 5 V • 2.4" QVGA TFT LCD • 64-Mbit SDRAM • Sensor del movimiento con salida digital en 3 ejes de giróscopo L3GD20, ST MEMS.

• MP45DT02, ST MEMS sensor de audio con micrófono monodireccional digital. • CS43L22 , audio DAC con driver de voz clase D integrado. • Ocho Leds: LD1 (rojo/verde) para comunicaciones USB. LD2 (rojo) para cuando la potencia de 3.3 V esté activada. 4 Leds de uso para el usuario: LD3 (naranja) LD4 (verde), LD5 (rojo) LD6 (azul). 2 Leds USB OTG: LD7 (verde) VBus y LD8 (rojo) indica sobrecorrriene. • 2 Botones (usuario y reset). • Conector USB • Cabecera de extensión para LQFP100 I/Os para una rápida conexión a la placa.	• Seis Leds: LD1 (rojo/verde) para comunicaciones USB. LD2 (rojo) para cuando la potencia de 3.3 V esté activada 2 Leds de uso para el usuario: LD3 (naranja) LD4 (verde), 2 Leds USB OTG: LD5 (verde) VBus y LD6(rojo) indica sobrecorrriene • 2 Botones (usuario y reset). • Conector USB • Cabecera de extensión para LQFP100 I/Os para una rápida conexión a la placa. • Software libre integral que incluye una variedad de ejemplos, parte del paquete STM32CubeF4 o STSW-STM32138 para el uso de las bibliotecas estándar.

Tabla 3 - Características STM32F4-Discovery y STM32F429I

En resumen la tarjeta STM32F429/439 trabaja a 180 MHz CPU/225 DMIPS, tiene hasta 2 Mbytes de memoria flash, con interfaz SDRAM, pantalla LCD, Chrom-ART Accelerator, interfaz de audio, ofrece un mayor rendimiento y menor consumo de energía estático en comparación al STM32F4x7 / F4x5

3.7 Trabajo realizado

El presente proyecto es una extensión de los trabajos realizados el curso pasado *diseño, implementación y control de un actuador elástico en serie* y *Diseño e implementación de la arquitectura software, funciones primitivas del movimiento control de un brazo robótico.* El objeto de estos trabajos consistía en la implementación de la arquitectura software genérica de los diferentes elementos del brazo robótico (sensores, actuadores…) y el desarrollo de un actuador elástico.

El trabajo dio buenos resultados pero debido a las limitaciones de memoria y velocidad de la placa utilizada nos hemos visto obligados a realizar un cambio de placa lo que ha obligado a recodificar parte del código.

Al cambiar de tarjeta se ha aprovechado para migrar a las nuevas librerías HAL desarrolladas por STMicrolelectrics.

El proyecto se ha centrado en la migración a las nuevas librerías y a la nueva tarjeta de STMicroelectronics del código de inicialización referente al Encorder y a los puertos (GPIO).

Para ello se han estudiado las nuevas estructuras de datos afectadas así como la inicialización de dichas estructuras en las nuevas librerías, el trabajo con los Timers, llamadas callback, rutinas ISR, nuevas macros etc.

Veamos con detalle las estructuras y macros que han sido actualizados a las nuevas librerías:

3.7.1 Modificación del fichero main.h

Con tal de definir las estructuras de las librerías HAL, se introduce el fichero de cabera #include “stm32f4xx_hal.h”.

3.7.2 Modificación del fichero main.c

Inicializamos la librería HAL mediante la siguiente instrucción:

```
/* STM32F4xx HAL library initialization:
     - Configure the Flash prefetch, Flash preread and Buffer caches
     - Systick timer is configured by default as source of time base, but user
           can eventually implement his proper time base source (a general purpose
           timer for example or other time source), keeping in mind that Time base
           duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and
           handled in milliseconds basis.
     - Low Level Initialization
   */
HAL_Init();

/* Configure the System clock to 180 MHz */
SystemClock_Config();
```

Función de Inicialización libreria HAL

La siguiente función modificada para poder utilizar las nuevas librerías es precisamente SystemClock_Config que aparece en la función main.c, veamos las modificaciones realizadas:

```
/* Enable Power Control clock */
__HAL_RCC_PWR_CLK_ENABLE();

/* The voltage scaling allows optimizing the power consumption when the device is
   clocked below the maximum system frequency, to update the voltage scaling value
   regarding system frequency refer to product datasheet.  */
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

/* Enable HSE Oscillator and activate PLL with HSE as source */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 8;
RCC_OscInitStruct.PLL.PLLN = 360;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
if(HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
  /* Initialization Error */
  Error_Handler();
}

if(HAL_PWREx_EnableOverDrive() != HAL_OK)
{
```

```
/* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
   clocks dividers */
RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
if(HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
{
  /* Initialization Error */
  Error_Handler();
```

Vemos como se rellenan las estructuras *RCC_OscInitStruct* y *RCC_clkInitStruct*, definidas en el fichero *stm32f4xx_hal.h*. Estas configuraciones se vuelcan en las funciones *HAL_RCC_oscConfig* y *HAL_RCC_ClockConfig* para configurar correctamente el sistema de reloj. En este fichero también se observa la utilización de diversas constante como HAL_OK, que indica que todo a funcionado correctamente.

3.7.3 Modificación del fichero _cds_daqquadraturein.c

La primera función modificada en este fichero ha sido *cds_DAQInitQuadratureIn( )* donde se realizan las inicializaciones. El resultado tras las modificaciones es:

```
TIM_Encoder_InitTypeDef sEncoderConfig;

__HAL_RCC_TIM1_CLK_ENABLE();

EncoderHandle.Instance=TIM1;

  /* Prepare timer */
//  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1,ENABLE);

//  TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
  HAL_TIM_Base_Init(&EncoderHandle);
 EncoderHandle.Init.Period = 0xffff;
  EncoderHandle.Init.Prescaler = 1;
  EncoderHandle.Init.CounterMode = TIM_COUNTERMODE_UP;

sEncoderConfig.EncoderMode=TIM_ENCODERMODE_TI12;
sEncoderConfig.IC1Polarity=TIM_ICPOLARITY_RISING;
sEncoderConfig.IC1Selection=TIM_ICSELECTION_DIRECTTI;
sEncoderConfig.IC1Prescaler=TIM_ICPSC_DIV8;
sEncoderConfig.IC1Filter=0x0;

if (HAL_TIM_Encoder_Init(&EncoderHandle,&sEncoderConfig)!=HAL_OK)
{
  Error_Handler();
}

HAL_NVIC_SetPriority(TIM1_CC_IRQn,1,1);
HAL_NVIC_EnableIRQ(TIM1_CC_IRQn);
```

En primer lugar activamos el reloj del Timer1 mediante la función *__HAL_RCC_TIM1_CLK_ENABLE( ).* Ésta sustituye a la función *RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1,ENABLE)* de la librería estándar.

A continuación se rellena la estructura *sEncoderConfig* que pertenece a la estructura *TIM_Encoder_InitTypeDef* propia de las librería HAL, con esta estructura inicializamos el Encoder. Con tal de conseguir que el Encoder genere una interrupción cada 8 pulsos, inicializamos las interrupciones y el servicio NVIC, con las funciones correspondientes a esta nueva librería.

Para terminar de inicializar el Encoder, se limpian los flags utilizando la función *HAL_NVIC_ClearPendingIRQ(TIM1_CC_IRQn).*

Para relacionar el Encoder con el Timer y que éste empiece su conteo se utilizarán las funciones:

HAL_NVIC_EnableIRQ(TIM1_CC_IRQn)

HAL_TIM_Encoder_Start_IT(&EncoderHandle,TIM_CHANNEL_1)

A continuación se ha implementado la función que se llama cada vez que vence la interrupción

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){

  control_pulse_interrupt();

}
```

Esta función tan solo se limita a llamar a la verdadera función que realizara el trabajo, en este aspecto la forma de trabajar de este tipo de librerías es poco intuitiva ya que se van llamando a funciones sin aparente razón para ello.

Si estudiamos el fichero *stm32f4xx_hal_tim.c* donde se tratan todas las funciones relacionadas con el Timer y el encoder, vemos como se realizan muchas comprobaciones y verificaciones, resultando difícil seguir la línea de programación. En este fichero en la línea 2813 donde aparece la llamada a esta función, que es la que deberá implementar el usuario para dar el servicio del Timer asociado al enconder.

3.7.4 Modificación del fichero _cds_time.c

En este fichero se realiza todo el tratamiento sobre el tiempo.

En primer lugar definimos la estructura *TIM_HandleTypeDef TimHandle*; estructura que nos sirve para identificar el TIMER2 utilizado en el proyecto, indicando la inicialización de este.

En la primera función *cds_TimeInit( )* inicializamos el Timer, en la función que se indica más abajo, vemos como inicializamos el reloj correspondiente mediante la función HAL, para a continuación asignar la estructura al TIMER2, el siguiente código se limita a rellenar la estructura correspondiente, prescaler, periodo etc. Finalmente se llama a la función HAL correspondiente de inicialización *HAL_TIM_Base_Init(&TimHandle)* verificando el valor retornado por si hubiere algún error. Finalmente *HAL_TIM_Base_Start_IT(&TimHandle)* inicializa las interrupciones.

```
void cds_TimeInit(void) {

   uint16_t prescaler;


   /* Compute the prescaler value */
   prescaler = (uint16_t) ((SystemCoreClock/2) / CLOCK) - 1;

  __HAL_RCC_TIM2_CLK_ENABLE();

    TimHandle.Instance=TIM2;
```

```
 /* TIM2 clock enable
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);  */

 /* Time base configuration */
TimHandle.Init.Period =0xFFFFFFFF;              // maximum autoreload-value
TimHandle.Init.Prescaler=prescaler;
TimHandle.Init.ClockDivision =0;
TimHandle.Init.CounterMode=TIM_COUNTERMODE_UP;


if( HAL_TIM_Base_Init(&TimHandle)!=HAL_OK)
{
       Error_Handler2();
}

if(HAL_TIM_Base_Start_IT(&TimHandle)!=HAL_OK)
{
       Error_Handler2();
}
```

.

Con las antiguas librerías el código utilizado sería:

```
TIM_TimeBaseStructure.TIM_Prescaler = prescaler;
TIM_TimeBaseStructure.TIM_Period = 0xFFFFFFFF;              // maximum autoreload-value
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

 //Use internal clock

TIM_InternalClockConfig (TIM2);

TIM_SetCounter(TIM2,0);

TIM_GenerateEvent(TIM2, TIM_EventSource_Update);

TIM_Cmd(TIM2, ENABLE);  // start counting
```

También se modifican las funciones que calculan el tiempo entre dos sucesos quedando como sigue:

```
cd_Error cds_TimeElapsedGet(uint8_t subdevice, uint32_t *microseconds) {

  static uint32_t last_time = 0;
  uint32_t t;


  if (subdevice != SUBDEVICE_TIMEELAPSED_1) {
       return(CD_Error_DAQBadSubdevice);  /* bad subdevice */
  }

 // t = TIM_GetCounter(TIM2);
  t=__HAL_TIM_GET_COUNTER(&TimHandle);

  if (last_time > t )
    // t = ((0xFFFFFFFF - t) + 1) + t;
    t = (0xFFFFFFFF - last_time)  + t;
  else
     t=t-last_time;

  *microseconds = t;

  return(CD_Error_NoError);
```

Vemos como se utilizan diferentes funciones propias de estas librerías, *__HAL_TIME_GET_COUNTER(&TimHandle)*

3.7.5 Modificación del fichero _cds_config.c

En este fichero se inicializan los puertos de I/O (GPIO).

La función encargada es *cds_ConfigPinout(void)* que inicializa todos los puertos de la tarjeta (del puerto A al puerto G) aquí se muestra parte del código utilizado para inicializar el puerto PD13 como salida digital para activar el led verde de la placa.

```
  GPIO_InitTypeDef  GPIO_InitStructure;
  __HAL_RCC_GPIOD_CLK_ENABLE();

//Led Verde
 GPIO_InitStructure.Pin   = GPIO_PIN_13;
 GPIO_InitStructure.Mode  = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
 GPIO_InitStructure.Pull  = GPIO_NOPULL;
 HAL_GPIO_Init(GPIOD, &GPIO_InitStructure);
```

La estructura de inicialización para el resto de puertos será similar, solo cambiarán los parámetros de configuración dependiendo del fin de cada uno de ellos.

Para la inicialización del Encoder configuramos los puertos PE9 (canal 1 del Timer 1) y PE11 (canal 2 del Timer 1).

```
/* PORT E ****************************************/
 __HAL_RCC_TIM1_CLK_ENABLE();
 __HAL_RCC_GPIOE_CLK_ENABLE();
// PE.9 (TIM1_CH1) ,PE.11 (TIM1_CH2) for quadrature encoder
GPIO_InitStructure.Pin = GPIO_PIN_9 | GPIO_PIN_11;
GPIO_InitStructure.Pull = GPIO_PULLDOWN;
GPIO_InitStructure.Mode = GPIO_MODE_AF_PP;
GPIO_InitStructure.Alternate=GPIO_AF1_TIM1;
HAL_GPIO_Init(GPIOE,&GPIO_InitStructure);
```

Vemos como se utilizan las nuevas librerías HAL para inicializar los diferentes puertos según su función.

3.8 Trabajo Pendiente

Queda pendiente de actualización a las nuevas librerías todo lo relacionado con la comunicación, protocolo SPI, comunicación a través de UART, transferencia de datos en modo DMA etc.

4 CONVERSORES ANALÓGICO-DIGITALES

4.1 Introducción

Los puntos críticos de un robot de asistencia a minusválidos son todos aquellos que puedan suponer un peligro para el mismo robot o para el usuario, es decir, todas aquellas variables que en algún momento puedan provocar un problema mecánico o una lesión al paciente.

Una de las variables a controlar es la aceleración. Durante la generación de trayectorias que deberá realizar el robot portátil habrá que tener un control en todo momento del valor de ésta.

Para conseguir esta medida se va a utilizar un sensor acelerómetro. La salida de este tipo de sensores es analógica, por tanto para poder analizarla será necesaria una etapa de conversión Analógico-Digital que nos permita su discretización.

Existen diferentes arquitecturas capaces de realizar esta tarea, cada una de ellas optimiza una característica del conversor (precio, velocidad, error…). Las más usadas son: ADC de Rampa, convertidores paralelos y de aproximaciones sucesivas.

4.1.1 ADC tipo Rampa.

Este tipo de convertidores tienen 3 partes principales: un integrador, un contador y una unidad de control. El integrador empieza con valor nulo y se va incrementando, cuando este valor supera en un bit al valor de entrada se dispara el contador. La siguiente ecuación relaciona el valor de salida del convertidor con el resto de parámetros del circuito:

$$n = \frac{RC}{V_{REF}T} * V_{IN}$$

El principal problema de estos conversores es que la salida depende de muchos factores, los cuales deben ser estables en el tiempo para que la conversión sea correcta. Por ejemplo, la presencia del periodo del reloj (T), obliga a que las conversiones sean fieles al periodo y por tanto éste debería variar muy poco.

La principal ventaja es el bajo coste, ya que resulta bastante barato gracias a su simplicidad y por tanto es óptimo para aplicaciones donde la precisión no es crítica y de bajo coste.

4.1.2 Conversor Paralelos.

La estructura de este tipo de convertidores está dividida en dos partes. La primera es un bando de comparadores, donde uno de los pines está conectado a la entrada analógica y el otro al múltiplo conveniente del voltaje de referencia ($V_{REF}*2^n$). La segunda etapa está constituida por un codificador, el cual toma la salida de los comparadores y crea un número binario de n bits.

El principal problema que presentan este tipo de conversores es que el circuito crece exponencialmente según el número de bits de resolución, lo que provoca que sean convertidores muy costosos y por lo general de sólo 8 bits.

Como principal ventaja encontramos es la velocidad de conversión. Estos dispositivos tienen una conversión prácticamente instantánea, ya que el único retardo que tienen es el de propagación de la señal eléctrica de la señal de entrada a la salida. Por esta característica se utilizan principalmente en aplicaciones que requieren altas velocidades de adquisición.

4.1.3 Convertidores ADC por aproximaciones sucesivas.

El principio de funcionamiento se basa en la realización de comparaciones sucesivas de manera descendente o ascendente, hasta encontrar la combinación que iguala la tensión entregada por el D/A y la entrada. Para ello, se busca determinar el valor de un bit en cada ciclo de reloj. En primer lugar se determina el valor del bit más significativo MSB, después el de D_{n-2} y así sucesivamente.

En el esquema siguiente se muestra el diagrama de transiciones para 3 bits donde se indica el proceso de búsqueda de la combinación digital. El proceso se repetirá n veces (siendo n el número de bits del registro de aproximaciones sucesivas).

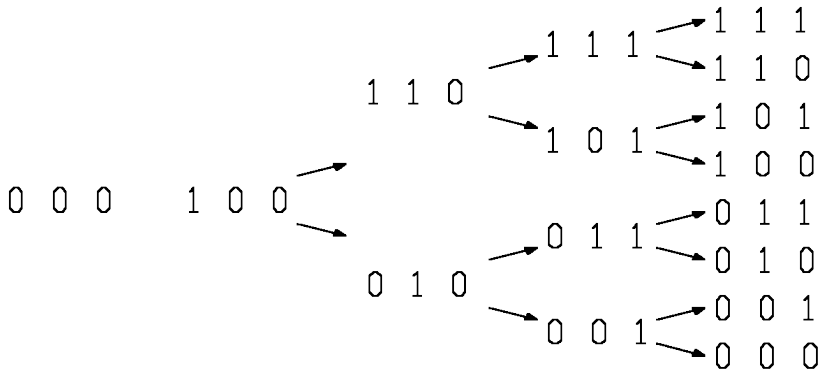


Ilustración 4 - Esquema Conversión por aproximaciones sucesivas

Para realizar las comparaciones con los valores digitales adecuados, el circuito se basa en el Registro de Aproximaciones Sucesivas. Debido a su forma de operación, donde en cada ciclo de reloj se determina un bit, la salida de estos conversores será una cadena de bits, normalmente en formato SPI.

La mayor ventaja de este tipo de convertidores es su versatilidad, ya que tienen un bajo tiempo de conversión (del orden de microsegundos) en un sistema no muy complejo y su salida es inmune a las variaciones de los componentes y la frecuencia de reloj.

4.2 Convertidor de la aplicación

Para la aplicación que nos ocupa el convertidor se utilizará para adaptar la señal enviada por el acelerómetro (señal analógica) y conseguir un control sobre funcionamiento del robot durante la generación de trayectorias. Por esto se necesita un convertidor con una velocidad de conversión alta, ya que si no es así se perderían datos y se podría llegar a valores críticos de aceleración.

Teniendo en cuenta los convertidores estudiados los de tipo rampa quedan descartados, ya que su velocidad de conversión es mayor y es dependiente de muchos factores que deben ser constantes en el tiempo para que las conversiones sean fiables.

Se han considerado los convertidores de aproximaciones sucesivas y convertidores paralelos ya que su tiempo de conversión es menor.

Otro factor importante en la aplicación estudiada es la resolución de la conversión. Se decidió que la resolución mínima que se podía permitir era de 16 bits, ya que con un número menor se perdería exactitud en la medida.

4.3 Convertidor STM32F429I

El microcontrolador utilizado (STM32F429I) dispone de un convertidor ADC. Para realizar las primeras pruebas se decidió utilizarlo para reducir el coste del proyecto.

Las características principales de este convertidor son:

- Resolución configurable: 6, 8, 10 o 12 bits
- Convertidor de aproximaciones sucesivas.
- Tiempo de muestreo programable.
- Tensión de alimentación: 2.4 V a 3.6 V para poder operar a velocidad alta y hasta 1.8 V para velocidad baja.

- Modo de conversión configurable (modo continuo, discontinuo…)
- Generación de interrupción al finalizar la conversión.

Como se ha explicado anteriormente, el tiempo de conversión es un factor crítico. En las características de este conversor encontramos que el tiempo de muestreo es configurable. La velocidad máxima a la que puede trabajar es de 45 MHz. Trabajando a esta frecuencia el tiempo de conversión de este convertidor sería de 688.82 ns.

Uno de los problemas que nos presenta este convertidor es su resolución, ya que el número máximo de bits permitidos es de 12 bits, algo escaso para esta aplicación. Sin embargo, tal y como se ha dicho anteriormente, para no aumentar el coste del proyecto, en las primeras pruebas se utilizará este dispositivo.

Tal y como se ha explicado en el apartado 3.5, en este proyecto se han utilizado las librerías HAL.

Hay que tener en cuenta que para inicializar el convertidor hay que inicializar los puertos GPIO, los canales DMA y el conversor ADC, además hay que configurar la interrupción que se generará tras cada conversión.

Para esta aplicación se utilizará el puerto PA4 como entrada del conversor, el canal DMA2 y se configurará el convertidor ADC 1 (cabe recordar que la placa STM32F429I tiene 3 convertidores ADC).

En primer lugar se activarán los Timers correspondientes al puerto GPIO, al DMA y del convertidor ADC, utilizando las funciones correspondientes:

```
// Enable peripherals and GPIO Clocks
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_ADC1_CLK_ENABLE();
__HAL_RCC_DMA2_CLK_ENABLE();
```

A continuación se configura el puerto GPIO. Hay que tener en cuenta que este puerto será la entrada al conversor, por tanto será inicializado como puerto analógico. La estructura utilizada para este puerto ha sido:

```
//Configure peripheral GPIO
GPIO_InitStruct.Pin=GPIO_PIN_4;
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

La siguiente configuración ha sido la del DMA2. Se ha configurado este DMA porque este acceso directo a memoria comunica con el convertidor que utilizaremos tal y como se ve en la siguiente tabla.

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A	ADC1	SAI1_B	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B	SPI6_TX	SPI6_RX	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX	SPI5_TX	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	-	SPI1_TX	-	-
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDIO	-	USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX	SPI5_TX	TIM8_CH4 TIM8_TRIG TIM8_COM

Tabla 4 - Comunicación del DMA

La estructura utilizada para inicializar el DMA ha sido:

```
//Configure the DMA streams
hdma_adc.Init.Channel= DMA_CHANNEL_2;
hdma_adc.Init.Direction=DMA_PERIPH_TO_MEMORY;
hdma_adc.Init.PeriphInc=DMA_PINC_DISABLE;
hdma_adc.Init.MemInc=DMA_MINC_ENABLE;
hdma_adc.Init.PeriphDataAlignment=DMA_PDATAALIGN_WORD;
hdma_adc.Init.MemDataAlignment=DMA_MDATAALIGN_WORD;
hdma_adc.Init.Mode=DMA_CIRCULAR;
hdma_adc.Init.Priority=DMA_PRIORITY_HIGH;
hdma_adc.Init.FIFOMode=DMA_FIFOMODE_DISABLE;
hdma_adc.Init.FIFOThreshold=DMA_FIFO_THRESHOLD_HALFFULL;
hdma_adc.Init.MemBurst=DMA_MBURST_SINGLE;
hdma_adc.Init.PeriphBurst=DMA_PBURST_SINGLE;

HAL_DMA_Init(&hdma_adc);
```

Para terminar con la inicialización se han configurado las características del conversor ADC. Tal y como se ha explicado anteriormente las especificaciones críticas son la resolución y la velocidad de conversión, por tanto se ha utilizado la máxima resolución permitida por este dispositivo en ambos casos.

Para conseguir la inicialización completa de este periférico primero se configura el dispositivo (prescaler, resolución, modo de conversión…) y para finalizar se acaba de configurar el canal de entrada (offset, periodo de muestreo…). Además tras cada uno de las inicializaciones se comprobará que no haya ningún error.

El código utilizado ha sido:

```
    // Variable used to get converted Valeu

__IO uint16_t ADCConvertedValue=0;

  ADC_ChannelConfTypeDef sConfig;

  //Configure the ADC peripheral

  AdcHandle.Init.ClockPrescaler=ADC_CLOCKPRESCALER_PCLK_DIV2;
  AdcHandle.Init.Resolution=ADC_RESOLUTION_12B;
  AdcHandle.Init.ScanConvMode=DISABLE;
  AdcHandle.Init.DiscontinuousConvMode=DISABLE;
  AdcHandle.Init.ContinuousConvMode=ENABLE;
  AdcHandle.Init.NbrOfDiscConversion=0;
  AdcHandle.Init.ExternalTrigConvEdge=ADC_EXTERNALTRIGCONVEDGE_NONE;
  AdcHandle.Init.ExternalTrigConv=ADC_EXTERNALTRIGCONV_T1_CC1;
  AdcHandle.Init.DataAlign=ADC_DATAALIGN_RIGHT;
  AdcHandle.Init.NbrOfConversion=1;
  AdcHandle.Init.DMAContinuousRequests=ENABLE;
  AdcHandle.Init.EOCSelection=DISABLE;


  if(HAL_ADC_Init(&AdcHandle) != HAL_OK)
  {
    Error_Handler();
  }

  // Configure ADC regular Channel

  sConfig.Channel=GPIO_PIN_4;
  sConfig.Rank=1;
  sConfig.SamplingTime=ADC_SAMPLETIME_3CYCLES;
  sConfig.Offset=0;

  if(HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

Tal y como se aprecia en la configuración, el periférico se ha configurado con 12 bits de resolución, una velocidad de conversión de 45 MHz (velocidad máxima permitida) y modo de conversión continuo.

Para terminar la función de inicialización se ha asociado la inicialización del canal DMA con la inicialización del convertidor ADC utilizando la función *__HAL_LINKDMA(&AdcHandle,DMA_Handle,hdma_adc)* y se ha configurado la interrupción para el DMA utilizando las funciones:

```
HAL_NVIC_SetPriority(DMA2_Stream0_IRQn,0,0);
HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
```

A continuación se ha programado la interrupción, para conseguir guardar en una variable el valor de la conversión.

La función utilizada para conseguir este valor ha sido:

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle){
  ADCConvertedValue=HAL_ADC_GetValue(AdcHandle);
}
```

Tal y como ocurría en la interrupción del Encoder (apartado 3.7), esta función se limita a llamar a la que realizará el verdadero trabajo. En este caso, si nos fijamos en el archivo *stm32f4xx_hal_adc.c*, que es el encargado de tratar las funciones del convertidor ADC, en la línea 984 nos encontramos esta función.

4.4 Convertidor externo

Una de las especificaciones limitantes de la aplicación es la resolución del convertidor. La resolución mínima es de 16 bits, por tanto la segunda opción que se ha utilizado en el desarrollo es un convertidor externo con esta característica. Para implementar esta opción se ha diseñado una placa de circuito impreso (PCB) con una fuente externa, un sensor acelerómetro y el convertidor ADC.

4.4.1 Diseño de la placa PCB

4.4.1.1 Fuente de alimentación

En primer lugar se estudió la utilización de la fuente de alimentación de la misma STM32F429I para alimentar tanto el sensor utilizado (acelerómetro) como el convertidor, sin embargo el error de ésta es del 25%. Este error provocaría que los datos a la salida del convertidor fueran erróneos y por tanto inutilizables para la aplicación. Como solución se buscó una fuente de alimentación con un error menor y cuya salida fuera de 3.3 V. Tras evaluar todas las opciones se decidió comprar la fuente externa LT1461AC8.3.3 ya que presentaba la mejor relación calidad-precio, con un error menor y un precio asequible (3€ aprox.).

4.4.1.2 Convertidor Analógico-Digital

A la hora de buscar un convertidor que cumpliera con todos los requisitos también se estudiaron distintos modelos. Finalmente se escogió el modelo LTC2380-16 por su velocidad de conversión (300 ns aproximadamente) y por permitir la comunicación SPI para la transferencia de datos a la STM32F429I.

4.4.1.3 Acelerómetro

El acelerómetro que se ha escogido ha sido el LIS344ALH. Este sensor es un acelerómetro de bajo consumo de 3 ejes con una sensibilidad de ±2g. Se decidió utilizar este sensor ya que era de la familia ST, con lo que se aseguraba que no hubiera ningún problema de compatibilidad en caso de utilizar el convertidor ADC de la tarjeta directamente.

Además su precio era bajo y por tanto el grupo de trabajo podía conseguirlo sin problemas.

4.4.1.4 Circuito a implementar

En la placa de circuito impreso se va a conectar la fuente de alimentación, el sensor y el convertidor para conseguir como salida los datos digitales y que éstos sean procesador por la tarjeta. En el Diagrama de Bloques 1 se muestra el esquema que debe seguir el circuito:

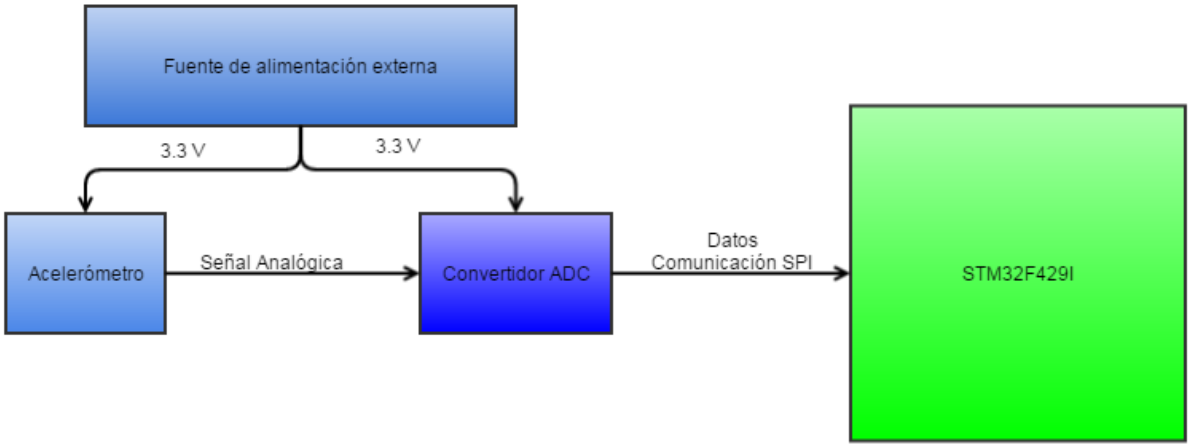


Diagrama de Bloques 1 - Circuito a implementar PCB

Tal y como se aprecia en la imagen, la fuente alimentará tanto el sensor como el convertidor, consiguiendo así que ambos componentes tengan una alimentación estable y con un error mínimo.

La señal analógica que será convertida por el conversor ADC será la aceleración medida por el eje X en el sensor. Se ha escogido esta variable ya que las primeras pruebas que se van a realizar serán sobre una plataforma que se mueve horizontalmente.

La transferencia de datos con la tarjeta se realizará mediante comunicación SPI, ya que es la forma de comunicación que permite el convertidor.

4.4.1.5 Diagrama de conexiones

El circuito completo se puede encontrar en el Plano 1: Plano de conexiones PCB.

En él se puede observar cómo se ha seguido la estructura del Diagrama de Bloques 1 tal y como se ha explicado en el apartado anterior.

La fuente de tensión LT1461AC8-3.3 está alimentada por una fuente externa. Ésta tendrá un valor entre 0.3 y 20 V y puede tener un valor de rizado elevado.

A partir de este componente la tensión será estable en un valor de 3.3 V. En el circuito se puede ver que a la salida del chip se ha colocado condensadores para evitar errores y corrientes de fuga.

La fuente LT1461ACS8-3.3 alimenta tanto al convertidor ADC (LTC2380-16) como al acelerómetro (LIS44ALH), con ello conseguimos no tener errores debidos al rizado en la tensión de alimentación.

Tal y como se ha comentado en el apartado anterior en el acelerómetro solamente utilizamos la señal producida por el eje X del acelerómetro, ya que las pruebas se realizarán sobre una plataforma que se mueve horizontalmente. Como ocurría en la fuente de alimentación, se han colocado condensadores de desacoplo tanto en el pin donde medimos la aceleración como en el los pines de alimentación.

Por último el convertidor ADC se ha dejado preparado para poder realizar de forma fácil y rápida la conexión necesaria para la comunicación SPI. Las salidas SDO, BUSY, SCK y CNV se han unido a dos conectores de dos bornes para poder conectarlos a la tarjeta STM32F429. En este caso también se han utilizado condensadores de desacoplo en los pines de alimentación.

4.4.1.6 Diseño de la Placa de circuito impreso

Uno de los objetivos de la placa PCB era que ésta tuviera el menor tamaño posible, tras el diseño el tamaño final ha sido de 24x21 mm.

Durante el diseño se tuvieron en cuenta las limitaciones de fabricación de las placas, las más destacadas eran:

- Tamaño mínimo de las pistas: 0.2 mm.
- Distancia mínima entre pistas: 0.3 mm.
- Diámetro interior mínimo de las vías: 0.4 mm.
- Diámetro exterior mínimo de las vías: 1.2 mm.

Teniendo en cuenta estas limitaciones se escogieron vías de 0.4 mm. de diámetro interior y 1.2 mm. de diámetro exterior, pistas de 0.3 mm., distancia mínima entre pistas de 0.35 mm. y distancia al borde de la placa de 0.635 mm.

Otro factor que se tuvo en cuenta fue poner los conectores lo más al borde posible para facilitar la posterior conexión entre la PCB y la STM32F429.

El resultado final de este diseño se puede encontrar en los anexos 11.4.2, 11.4.3, 11.4.4 y 11.4.5

Tras la fabricación de la placa el resultado final ha sido el siguiente:

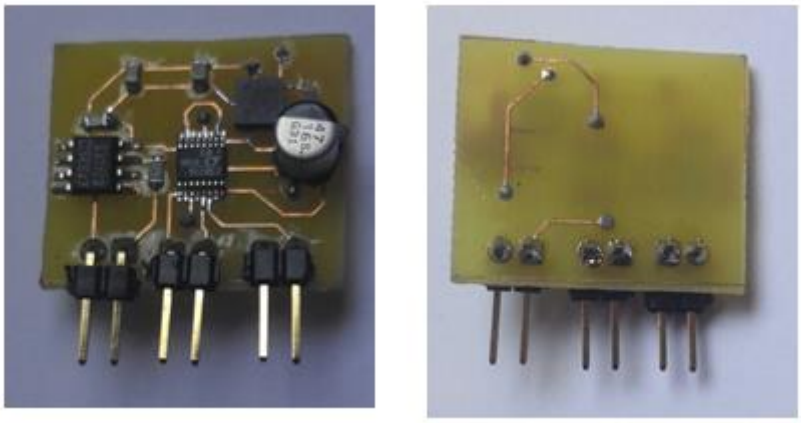


Ilustración 5 - Placa PCB fabricada (Izquierda: cara superior, Derecha: cara inferior)

5 SISTEMA DE CONTROL POR DETECCIÓN DE VOZ

5.1 Introducción

El reconocimiento de voz es un proceso mediante el cual identificamos el lenguaje humano a partir de conversión de una señal analógica (onda acústica) en señales eléctricas que pueden ser interpretadas. Son sistemas probabilísticos la tarea se lleva acabo atendiendo al algoritmo "dadas n opciones para una entrada de datos ambigua, elegir la más probable" (Jurafsky, y otros, 2009).

Hay muchos desafíos técnicos a tener en cuenta al utilizar estas tecnologías:

- Elevado nivel de exactitud**.** La tecnología debe ser percibida por el usuario como muy exacta, robusta y fiable. En este sentido, los principales retos ante los que deben enfrentarse un sistema de reconocimiento de voz son:
  - Variabilidad lingüística: Fonética
  - Variabilidad del usuario: Pronunciación, fatiga, estrés, ronquera...
  - Variabilidad del Canal: Ruido, cambios en el medio de transmisión...
- Facilidad de utilización: La voz es una de las modalidades de entrada / salida de información entre un humano y una máquina, por tanto, debe tener como objetivo la naturalidad y facilidad en la interacción hombre-máquina.

Para superar estos problemas, los sistemas de reconocimiento de voz incluyen tres procesos:

- Extracción de índices acústicos de la señal hablada: comporta el análisis espectral de las señales que produce la voz.
- Estimación de la probabilidad de que la cadena índice fue originada por un hipotético segmento de pronunciación: compara la señal con patrones previamente aprendidos y almacenados que se utilizan como modelo.
- Determinación de la pronunciación reconocida a través de una búsqueda entre hipotéticas alternativas: se utilizan principalmente modelos estocásticos simples, modelos ocultos de Marcov y tecnologías como redes neuronales y alineamiento dinámico en el tiempo.

5.1.1 Procesadores para el reconocimiento por voz

Se clasifican según el tipo de reconocimiento y tenemos básicamente 2 tipos

- Dependiente del hablante: puede reconocer palabras de la persona que lo entrena, su precisión es alta y se utilizan en sistemas de seguridad y en aplicaciones en las que sólo un usuario debe accionar un sistema determinado.
- Independiente del hablante: reconoce palabras independientemente de la persona que las pronuncie, en este caso disminuye la precisión del reconocimiento, se utiliza ampliamente en sistemas de telefonía móvil.

5.1.2 Tarjeta SpeakUp Click

La tarjeta que se va a utilizar es el SpeakUp Click de Mikroelektronica. SpeakUp es un sistema de reconocimiento de voz dependiente del usuario que puede reconocer más de 200 comandos procesándolos mediante el STM32F415RG MCU.

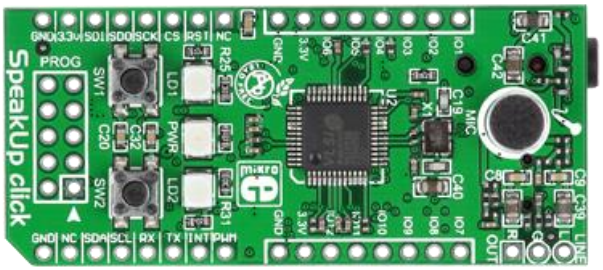


Ilustración 6 - Tarjeta SpeakUp

Funciona comparando los sonidos que capta a través de un micrófono que lleva incorporado con comandos pregrabados. El sonido se procesa mediante el IC VS1053 que contiene internamente un decodificador de audio estéreo. La tarjeta proporciona además un software específico que facilita su utilización. En la placa dispondremos de 12 pines de E/S programables.

5.2 Especificación del producto a desarrollar

5.2.1 Definición del producto

El producto a desarrollar consiste en una interfaz de reconocimiento de voz dedicada al control de motores asociados a aplicaciones en robots de asistencia a minusvalidos. Los comandos a utilizar contendrán órdenes sencillas de movimiento de las diferentes articulaciones, por ejemplo, cierra_mano, sube_hombro gira_muñeca.

Permitirá además codificar diferentes estados de énfasis, ya que el sistema no debe reaccionar de igual forma ante un cierra_mano suave o el mismo comando expresado con mayor fuerza. Las aplicaciones de este producto se realizarán en el campo de la Biometría y la Medicina.

Para ello en este apartado se obtendrá:

- Programación y codificación de los comandos para su posterior utilización
- Programación de la placa STM32F407-Discovery para el control de motores.

La comunicación entre el módulo de reconocimiento de voz y el sistema a controlar se realizará a través de la placa STM32F407-Discovery. En este proyecto se realizará una emulación sencilla controlando pequeños motores de corriente continua.

5.2.2 Requisitos específicos

5.2.2.1 Especificaciones del Hardware

- Tarjeta de reconocimiento de voz elegida, SpeakUp de Mikroelektronica
  - Permite más de 200 comandos
  - Los comandos son dependientes del usuario y precisan ser entrenados
  - Comunicación con el PC puerto USB
  - Comunicación con la placa STM32F407-Discovery mediante pines de E/S digitales Programables
  - Micrófono incorporado y entrada para micrófono externo
- Placa interface STM32F407-Discovery
  - Comunicación con el PC puerto USB
  - Comunicación de secuencias de control motores, pines E/S

5.2.2.2 Especificaciones del Software

- Tarjeta de reconocimiento de voz: software propio de fácil configuración
- Placa STM32F407-Discovery entorno de programación en C Keil5

5.2.2.3 Operaciones a realizar

- Fase de entrenamiento de los diferentes comandos.
- El sistema debe permitir el control de los motores responsables del movimiento de una determinada articulación, mediante control de voz.

5.3 Tarjeta de control por voz

5.3.1 Descripción de la placa

Como ya se ha comentado anteriormente se ha elegido la placa SpeakUp de Mikroelektronica

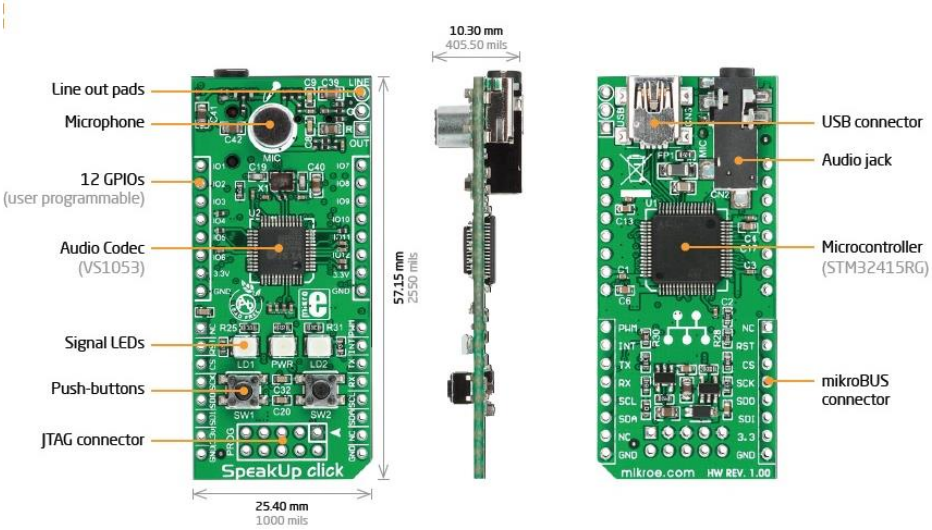


Ilustración 7 - Placa SpeakUp detallada

La placa dispone de micrófono, 12 pin de E/S programables, leds de señalización, feedback de grabación e indicador de alimentación, pulsadores para funcionamiento manual de grabación, conexión USB para PC y conexión Jack para micrófono externo.

La interacción con otros módulos se simplifica a través del conector mikroBUS™

Cada zócalo mikroBUS ™ consta de dos líneas que contienen 1x8 pines hembra. Soporta protocolos de comunicación SPI, UART y la comunicación I^2C. También presenta conectores para PWM, interrupción, entrada analógica, Reset y Selección de Chip. Se puede trabajar con 2 tipos de alimentación: + 5V y GND en una línea y 3,3 V y GND en la otra. El zócalo mikroBUS™ encaja en placas universales estándar.

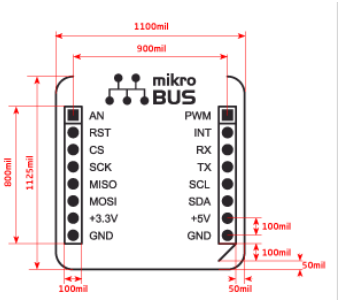


Ilustración 8 - Zócalo MikroBUS

En la parte inferior tenemos un conector JTAG (Joint Test Action Group**).** Diseñado originalmente para circuitos impresos, actualmente se utiliza para la prueba de submódulos de circuitos integrados. Es muy útil para los programadores, por ejemplo cuando se utiliza como herramienta de depuración, en un circuito que usa JTAG como mecanismo de transporte, permite al programador acceder al módulo de depuración que se encuentra integrado dentro de la CPU. El módulo de depuración permite al programador corregir sus errores de código y lógica de sus sistemas.

En la placa también podemos encontrar el decodificador de audio VS1053.

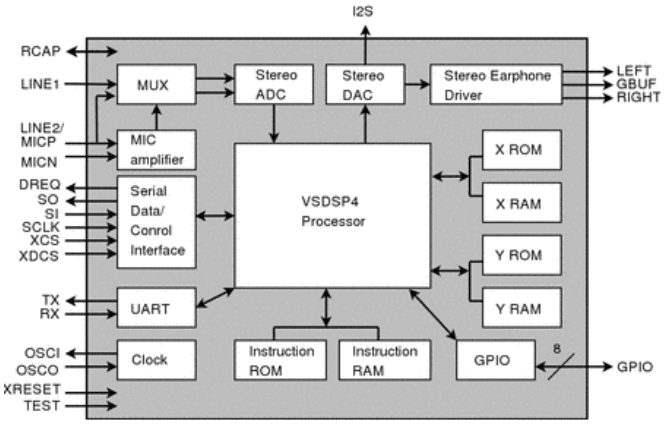


Ilustración 9 - Decodificador de audio VS1053

Es un chip decodificador de MP3 capaz de decodificar los formatos de audio más comunes, además es capaz de grabar en tres diferentes formatos de audio, sin pérdida de PCM (Pulse Code Modulation) de 16 bits (técnica de muestreo para digitalizar señales análogas, especialmente de audio). El PCM muestrea las señales 8000 veces por segundo; cada uno de estos muestreos se representa por 8/16 bits).

El microcontrolador STM32F415RG gestionará las señales recibidas y nos permitirá comunicarnos con la Placa STM32F407-Discovery que programaremos para realizar nuestro proyecto

El funcionamiento de la placa es el siguiente:

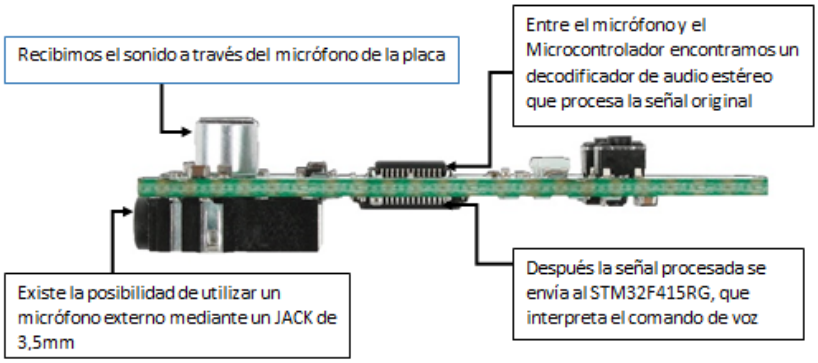


Ilustración 10 - Funcionamiento SpeakUp

Sobre la placa podemos utilizar de forma directa, 12 pines programables de E/S

Su esquema interno se muestra en la siguiente figura:

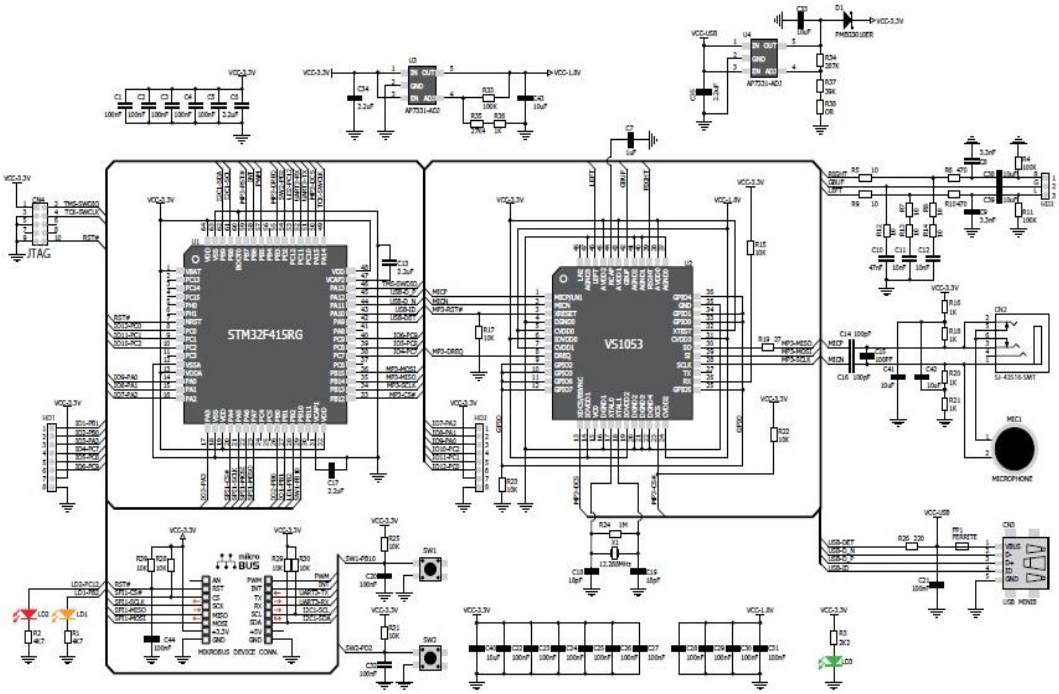


Ilustración 11 - Esquema interno de la placa SpeakUp

5.3.2 Firmware de SpeakUp

El objetivo principal de un sistema de reconocimiento de voz es sustituir el reconocimiento humano. Debemos tener en cuenta que para un sistema artificial es muy difícil lograr la flexibilidad ofrecida por el oído y el cerebro humano. El principio de trabajo de los sistemas de reconocimiento de voz está basado en la comparación de los datos de entrada a los patrones pregrabados. Estos patrones se pueden disponer en forma de fonema o palabra. El sistema asigna a los datos de entrada, el patrón que es más similar siempre que supere un mínimo de semejanza.

La amplitud de las señales de voz puede variar significativamente, por ello se necesita un procesamiento previo de las señales en el cual se extraen las características de la señal. En primer lugar, se obtienen los vectores de características a partir de los datos de voz de entrada, y a continuación, estos vectores se comparan con los patrones guardados. Los vectores que se extraen de la señal de voz deben representar los datos de forma eficiente y deben tener el tamaño y las características distintivas suficientes.

El Firmware que usa SpeakUp es el algoritmo DTW (Dynamic Time Warping o Alineamiento Temporal dinámico), basado en palabra, con reconocimiento de palabra aislada, dependiente del hablante, Template Matching (algoritmos de reconocimiento de plantillas).

- Reconocimiento de voz basado en la palabra: la unidad más pequeña de reconocimiento es una palabra.
- Reconocimiento de palabras aisladas: se reconocen las palabras que se pronuncian con pausas cortas.
- Dependiente del hablante: están construidos para un solo interlocutor
- Template matching: es una forma de reconocimiento de patrones. Representa los datos de voz como conjuntos de vectores de característica/ parámetro llamados plantillas. Cada palabra o frase en una aplicación se almacena como una plantilla separada. La señal de entrada se compara entonces con la plantilla almacenada y la plantilla almacenada más similar al patrón de la voz de entrada se identifica como la palabra o frase de entrada.

El algoritmo DTW merece especial atención y se detalla en el siguiente apartado.

5.3.3 Algoritmo de reconocimiento de voz DTW.

La tarjeta de voz utiliza el algoritmo DTW, para el reconocimiento de los patrones de voz. Este método tiene en cuenta la variación en la escala del tiempo de dos palabras a comparar.

El problema que se presenta cuando se pronuncia una palabra es que ésta no siempre se realiza a la misma velocidad, lo que produce importantes distorsiones temporales. Estas distorsiones afectan no solo a la palabra considerada sino también a sus componentes acústicos.

Las variaciones temporales no son generalmente proporcionales a la velocidad de locución y podrán variar de locutor a locutor. Es por esto que se hace necesario un procedimiento que permita comparar dos palabras, sin considerar las distorsiones temporales.

Los métodos que se usan para realizar lo expuesto se basan en algoritmos de programación dinámica.

El Alineamiento Temporal Dinámico (Dynamic Time Warping, DTW), es una técnica surgida de la problemática inherente a diferentes realizaciones de una misma locución, en las que se observa una variabilidad interna en la duración de los grupos fónicos que la forman, de modo que no existe una sincronización temporal (alineamiento temporal).

Además, esta falta de alineamiento no obedece a una ley fija (p.e. un retardo constante), sino que se da de forma heterogénea, produciéndose así variaciones localizadas que aumentan o disminuyen la duración del tramo de análisis.

La problemática asociada hace referencia a la dificultad añadida en el proceso de medida de distancia entre patrones, puesto que se estarán comparando tramos que pueden corresponder a unidades fónicas distintas. Será necesario alinear temporalmente la locución para proceder a realizar una medida de distancia entre patrones cuyo nuevo eje temporal haya homogeneizado las variaciones iniciales. La Ilustración 12 muestra dos realizaciones de la misma locución (contorno de energía localizada) antes y después de ser alineadas:

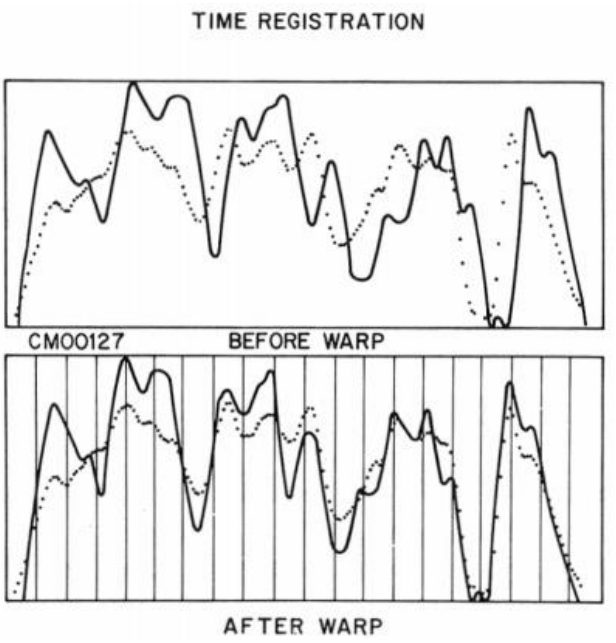


Ilustración 12 - Realizaciones de la misma locución antes y después del alineamiento

Alineamiento temporal (temporal warping): en este proceso, el eje temporal de la señal de test se comprime y expande de forma no lineal para alinear los vectores de características entre patrón y test.

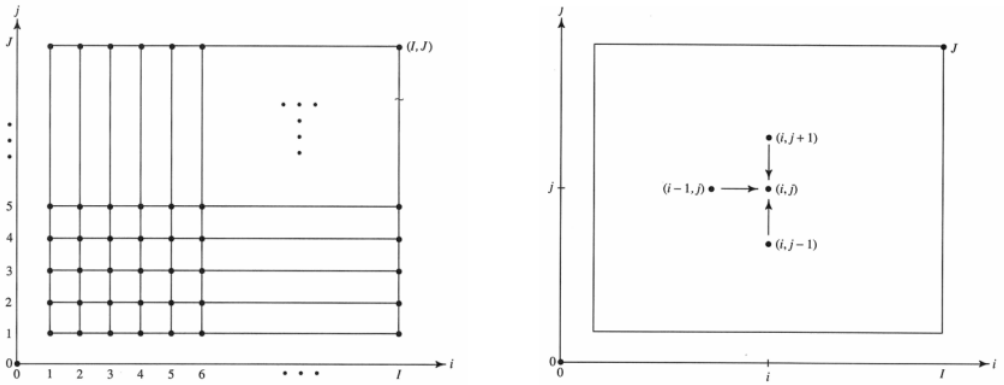


Ilustración 13 - Alineamiento temporal

Fruto del proceso de alineamiento, surge lo que se conoce como camino de alineamiento. La Ilustración 14 muestra de forma simplificada (izqda.) y de forma real (drcha.) dos posibles caminos de alineamiento entre realizaciones

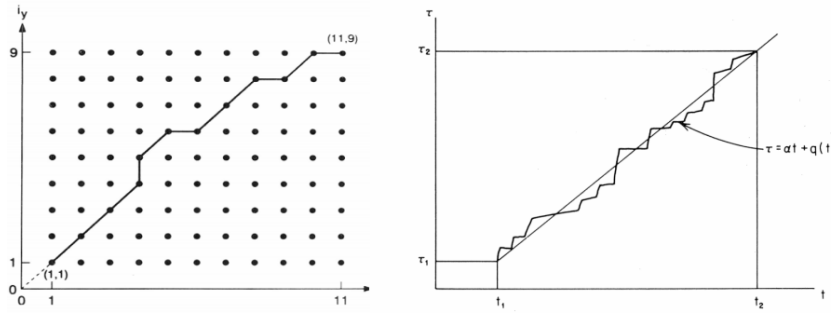


Ilustración 14 - Ejemplos de camino de alineamiento

De forma genérica, podemos presentar la función de alineamiento a través de una representación del tipo siguiente:

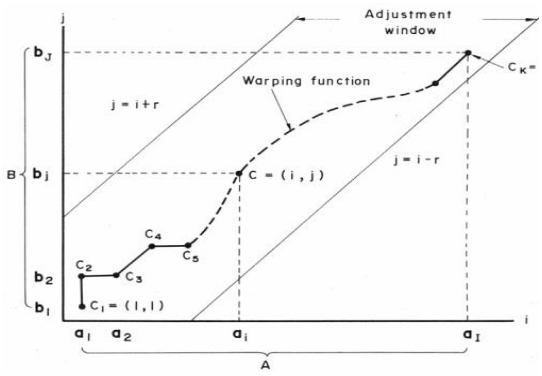


Ilustración 15 - Función de alineamiento genérica

La Ilustración 16 muestra dos realizaciones desalineadas, la función de alineamiento correspondiente y el efecto de alineamiento sobre las locuciones anteriores:

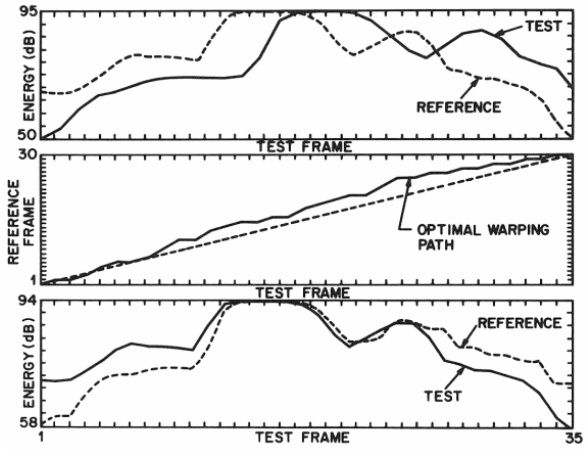


Ilustración 16 - Realizaciones desalineadas, función de alineamiento y efecto de alineamiento

En la Ilustración 17, se observa con detalle el alineamiento entre dos locuciones, haciendo hincapié en la correspondencia precisa entre los dos máximos relativos encontrados en las mismas:

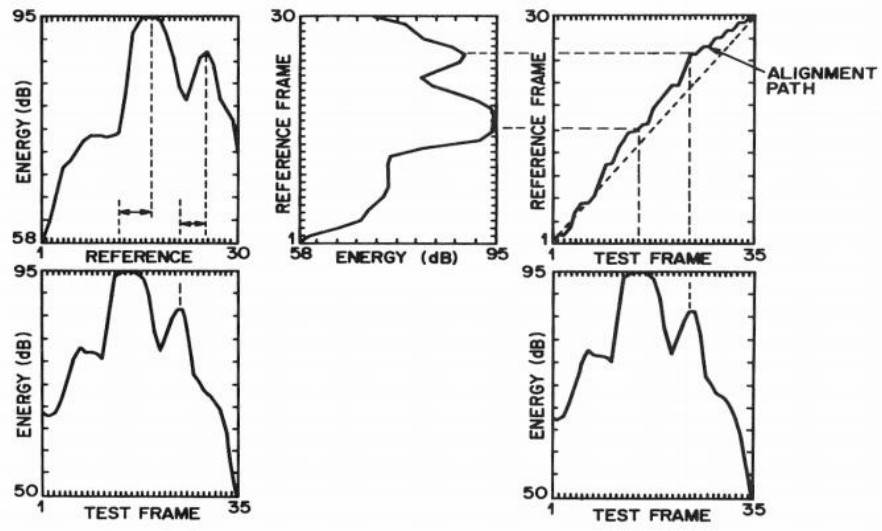


Ilustración 17 - Alineamiento entre dos locuciones

Si observamos la función de alineamiento en detalle, tendremos que si la función de alineamiento pasa por (i,j), comparamos el vector 'i' de A con la 'j' de B.

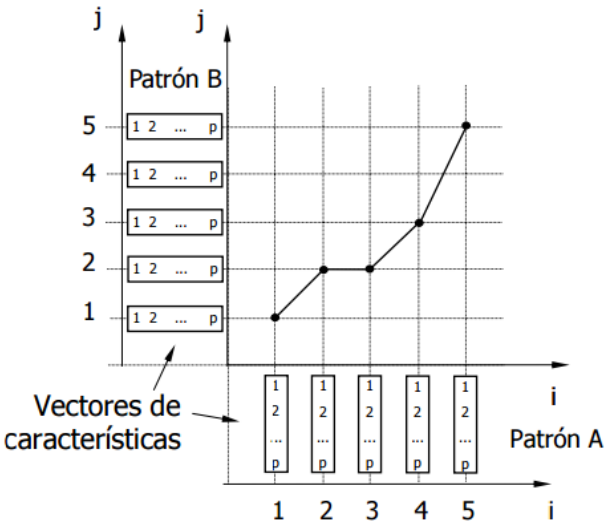


Ilustración 18 - Función de alineamiento en detalle

5.3.3.1 Cálculo de la función de alineamiento

El problema consistirá en encontrar la función de alineamiento a partir de los dos vectores de características (patrones) 'A' y 'B'.

Si tenemos:

$$A = \{a_1, a_2, \dots, a_i, \dots, a_M\}$$

$$B = \{b_1, b_2, \dots, b_j, \dots, b_N\}$$

Llamando 'C' a la función de alineamiento:

$$C = \{c(1), c(2), \dots, c(k), \dots, c(K)\}$$

Donde c(k) es un par de punteros a los elementos a comparar:

$$c(k) = [i(k), j(k)]$$

Para cada c(k) tenemos una función de coste:

$$d\{c(k)\} = \delta\left(a_{i(k)}, b_{j(k)}\right)$$

Ésta refleja la discrepancia entre los elementos comparados. Una función de coste típica es la distancia euclídea:

$$d\{c(k)\} = \left(a_{i(k)} - b_{j(k)}\right)^2$$

La función de alineamiento será aquella que minimice la función de coste total:

$$D(C) = \sum_{k=1}^{K} d\{c(k)\}$$

Esta función de coste deberá cumplir las siguientes limitaciones:

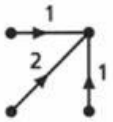
- La función de alineamiento debe ser monótona creciente:

$$i(k) \geq i(k-1) \quad y \quad j(k) \geq j(k-1)$$

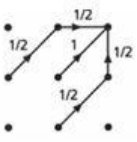
- La función debe alinear los puntos finales de A y B:

$$i(1) = j(1) = 1 \quad y \quad i(K) = M \quad y \quad j(K) = N$$

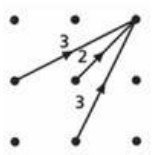
- La función debe cumplir unos límites 'locales', como los siguientes:



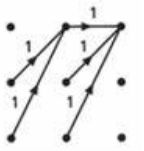
$$\min \begin{Bmatrix} D(i_x - 1, i_y) + d(i_x, i_y) \\ D(i_x - 1, i_y - 1) + 2d(i_x, i_y) \\ D(i_x, i_y - 1) + d(i_x, i_y) \end{Bmatrix}$$



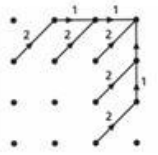
$$\min \begin{Bmatrix} D(i_x - 2, i_y - 1) + \frac{1}{2}d(i_x - 1, i_y) + d(i_x, i_y) \\ D(i_x - 1, i_y - 1) + d(i_x, i_y) \\ D(i_x - 1, i_y - 2) + \frac{1}{2}d(i_x, i_y - 1) + d(i_x, i_y) \end{Bmatrix}$$



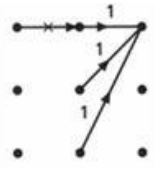
$$\min\begin{Bmatrix} D(i_x-2,i_y-1)+3d(i_x,i_y) \\ D(i_x-1,i_y-1)+2d(i_x,i_y) \\ D(i_x-1,i_y-2)+3d(i_x,i_y) \end{Bmatrix}$$



$$\min\begin{Bmatrix} D(i_x-2,i_y-1)+d(i_x-1,i_y)+d(i_x,i_y) \\ D(i_x-2,i_y-2)+d(i_x-1,i_y)+d(i_x,i_y) \\ D(i_x-1,i_y-1)+d(i_x,i_y) \\ D(i_x-1,i_y-2)+d(i_x,i_y) \end{Bmatrix}$$



$$\min\begin{Bmatrix} (i_x-3,i_y-1)+2d(i_x-2,i_y)+d(i_x-1,i_y)+d(i_x,i_y) \\ D(i_x-1,i_y-1)+2d(i_x-1,i_y)+d(i_x,i_y) \\ D(i_x-1,i_y-1)+2d(i_x,i_y) \\ D(i_x-1,i_y-2)+2d(i_x,i_y-1)+d(i_x,i_y) \\ D(i_x-1,i_y-3)+2d(i_x,i_y-2)+d(i_x,i_y-1)+d(i_x,i_y) \end{Bmatrix}$$



$$\min\begin{Bmatrix} D(i_x-1,i_y)g(k)+d(i_x,i_y) \\ D(i_x-1,i_y-1)+d(i_x,i_y) \\ D(i_x-1,i_y-2)+d(i_x,i_y) \end{Bmatrix}$$

Estos límites locales dan lugar a la denominada 'restricción de pendiente'. A continuación, se muestra un caso particular:

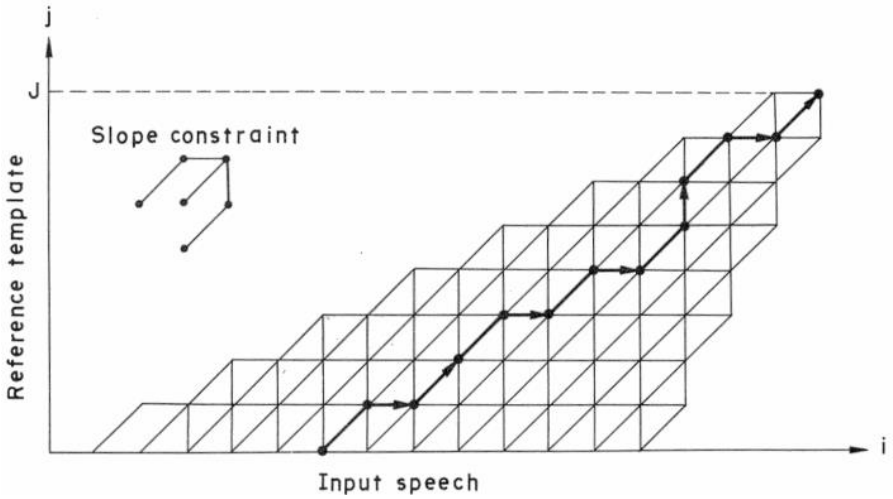


Ilustración 19 - Ejemplo de restricción de pendiente

Generalmente existe algún tipo de límite global en la máxima cantidad de alineamiento. Esto se puede expresar como |i(k) - j(k)|<Q, lo que da lugar a una restricción denominada de 'ventana' o rombo, siendo Q el ancho de la ventana:

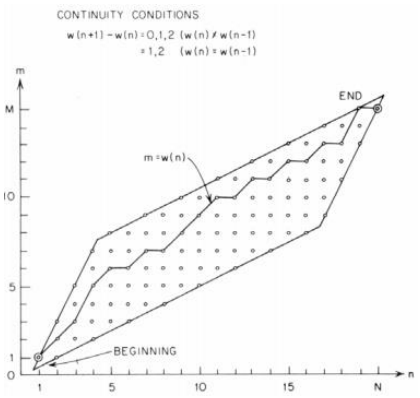


Ilustración 20 - Restricción "ventana"

5.3.3.2 El algoritmo de Programación Dinámica

El problema de calcular la función de alineamiento se resuelve mediante técnicas de programación dinámica, motivo por el que se conoce a esta técnica como alineamiento temporal dinámico.

Una posible idea para la resolución del problema estaría en el cálculo de D(C) (función de coste total) por todos los caminos posibles, eligiendo posteriormente el de coste mínimo.

La Ilustración 21 muestra la multiplicidad posible para la función de alineamiento para unos determinados límites locales y una restricción de ventana dada:

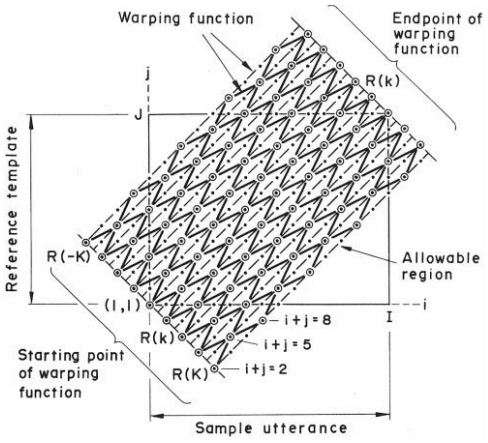


Ilustración 21 - Multiplicidad posible para una función de alineamiento

La programación dinámica nos dice que el mejor camino entre (1,1) y cualquier punto (i,j) es independiente de lo que suceda más allá de este punto. Por tanto, el coste total del punto [i(k),j(k)] es el propio de este punto más el camino más económico hasta él:

$$D(C_k) = d\{\, c(k)\} + \min\{D\ (C_{k-1})\}$$
$$legal\ c(k-1)$$

Donde *legal c(k-1)* representa todos los predecesores permitidos de *c(k)*. Sin embargo, por la limitación local tenemos un conjunto muy limitado de predecesores posibles.

Podemos resumir el algoritmo de programación dinámica como se muestra en:

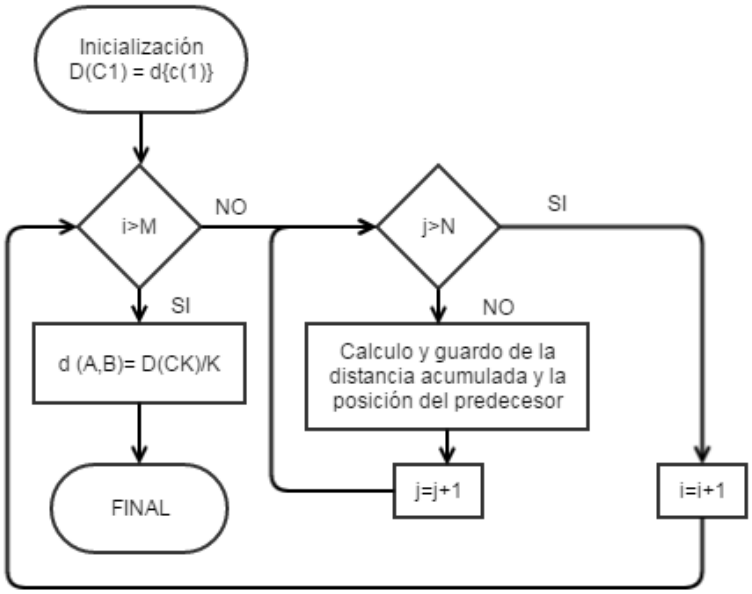


Diagrama de Bloques 2 - Algoritmo de programación dinámica

Para implementar el algoritmo sólo se necesita disponer de dos vectores adicionales:

- Uno de tamaño M x N para guardar el predecesor de cada punto
- Otro de tamaño 2 x N para guardar los costes acumulados de la columna anterior y los de la presente, actualizándolos en cada paso de i.

Si representamos gráficamente el predecesor de cada punto y recorremos hacia atrás (‘backtracking’) desde el punto final (M, N) hasta el inicial (1,1) obtenemos el camino de alineamiento.

5.3.3.3 Extensiones del DTW

El hecho de exigir que el camino de alineamiento comience en (1,1) y termine en (M, N) hace muy dependiente al sistema de la detección de voz del sistema de reconocimiento, por lo que podremos disminuir las restricciones de punto inicial y final. La figura muestra una restricción inicial de 5 tramas y una final de 9:

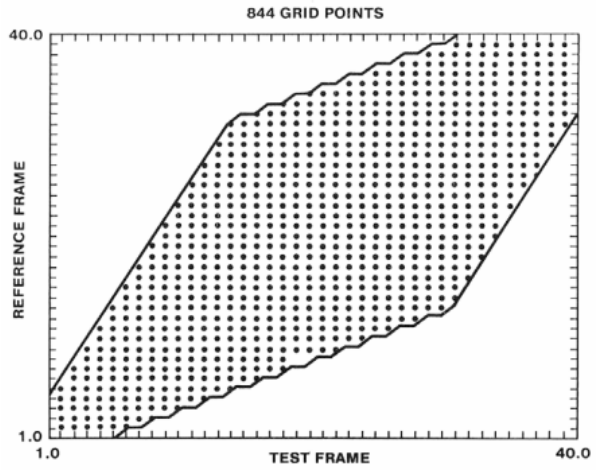


Ilustración 22 - Ejemplo extensión del DTW

5.3.4 Configuración de la placa mediante software

La configuración de la placa se realiza mediante software libre que puede obtenerse en:

http://www.mikroe.com/downloads/get/2077/speakup_app.zip

Mediante el programa podemos configurar la placa para que reconozca 200 comandos de voz y asignar a cada uno de ellos una función.

El diagrama de funcionamiento del programa es el siguiente:

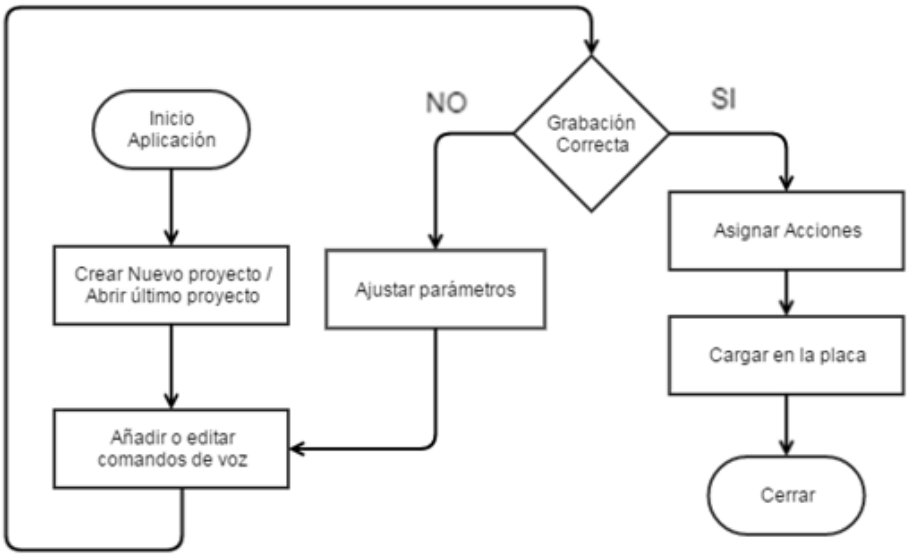


Diagrama de Bloques 3 - Funcionamiento del programa de SpeakUp

5.3.4.1 Configuración inicial del software

En primer lugar, conectamos la placa SpeakUp al PC mediante el puerto USB

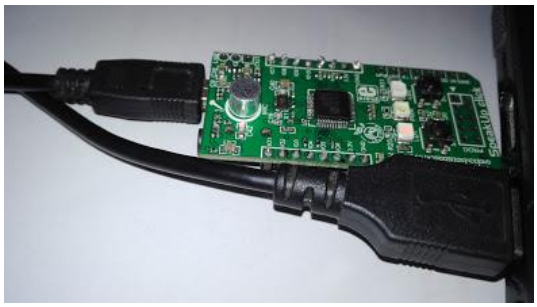


Ilustración 23 - Conexión USB SpeakUp

A continuación, el programa calibra el nivel de ruido del ambiente durante unos 10 seg. Al finalizar este proceso se apaga el led rojo de la placa.

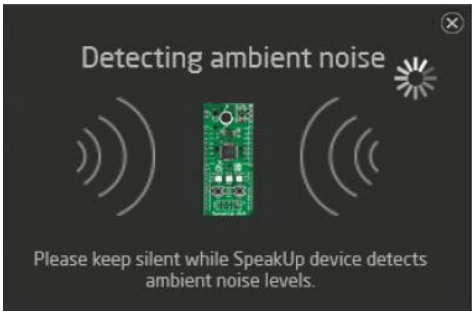


Ilustración 24 - Detección del ruido

Para crear un nuevo proyecto accederemos al botón del menú, asignaremos un nombre al nuevo programa y le indicaremos la ruta para guardarlo

Ilustración 25 - Creación de un nuevo proyecto

El siguiente paso consistirá en crear los comandos de voz. Para grabar un nuevo comando pulsamos en el botón de añadir un nuevo comando de voz y para grabarlo se pulsa el botón *Record*.

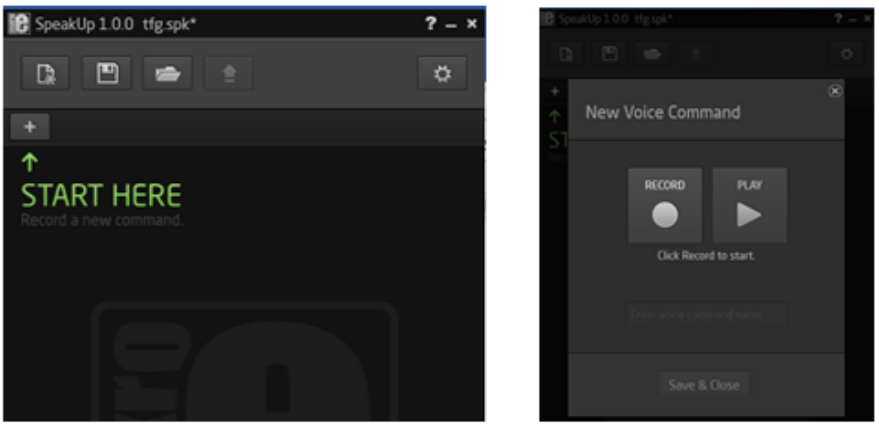


Ilustración 26 - Grabación de un nuevo comando

La duración de la grabación se puede modificar en la opción *Project Settings*. Tras la grabación podemos verificar que el comando grabado se escucha correctamente y tal y como deseamos.

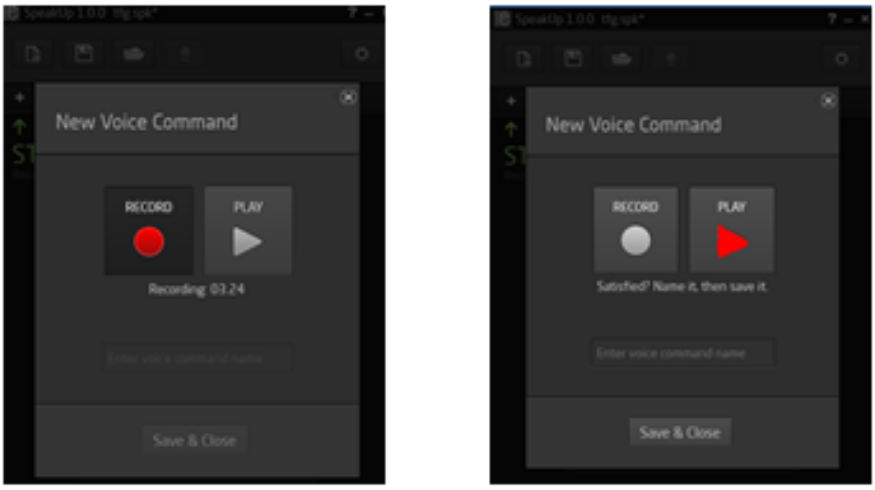


Ilustración 27 - Grabación y Reproducción de los comandos

En el caso de que la grabación no resulte satisfactoria permite volver a grabar el comando. Si la grabación falla al detectar los comandos posiblemente tengamos un entorno con demasiado nivel de ruido ambiente, podemos intentar repetir el comando con voz más fuerte y si no funciona modificar nivel de umbral de ruido *Noise treshhold***.**

En cualquier caso, los comandos pueden ser editados, eliminados o regrabados.

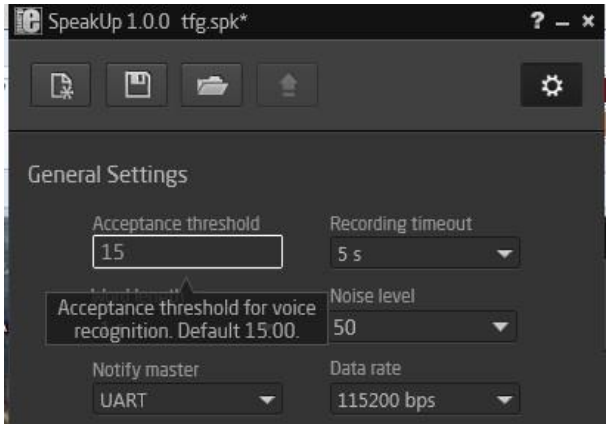


Ilustración 28 - Project Settings

5.3.4.2 *Project settings*

Nos permitirá ajustar diferentes parámetros del programa. Podemos acceder al menú haciendo pulsando en el botón o mediante CTRL- T

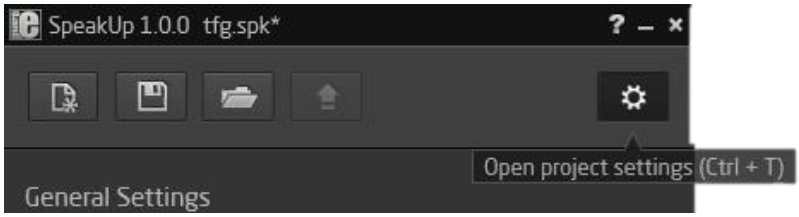


Ilustración 29 - Acceso a Project Setting

En la parte superior de la pantalla nos muestra los ajustes generales

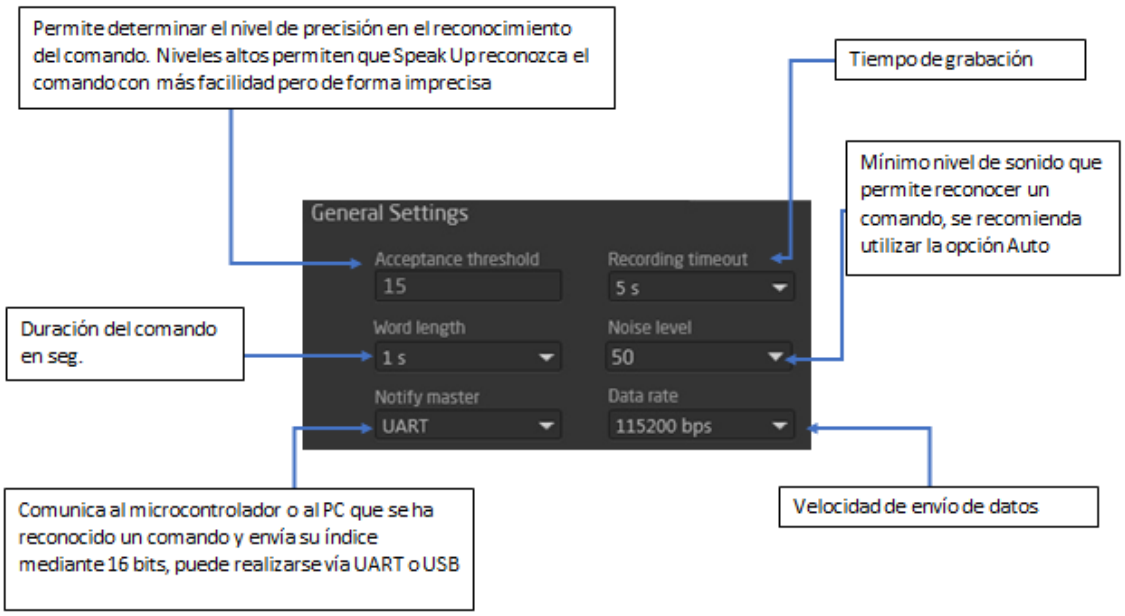


Ilustración 30 - Ajustes generales

En la parte inferior de la pantalla podemos renombrar los pines y fijar su valor inicial

Ilustración 31 - Ajustes de los pines

5.3.4.3 Asignación de acciones a los pines

Al grabar un comando asignamos a los pines elegidos una acción a realizar. Existen 5 acciones posibles

- NONE: no realiza ninguna acción
- ON : el pin correspondiente se coloca en estado alto
- OFF: el pin correspondiente se coloca en estado bajo
- TOGGLE: el pin indicado cambia del estado actual pasa de HIGH a LOW o viceversa
- PULSE: se envía al pin un tren de pulsos según los siguientes parámetros

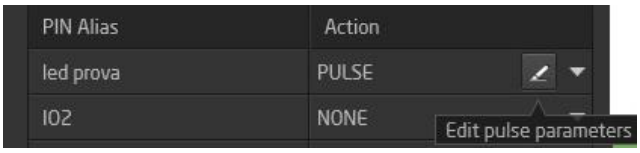


Ilustración 32 - Acceso a los parámetros de las salidas tipo PULSE

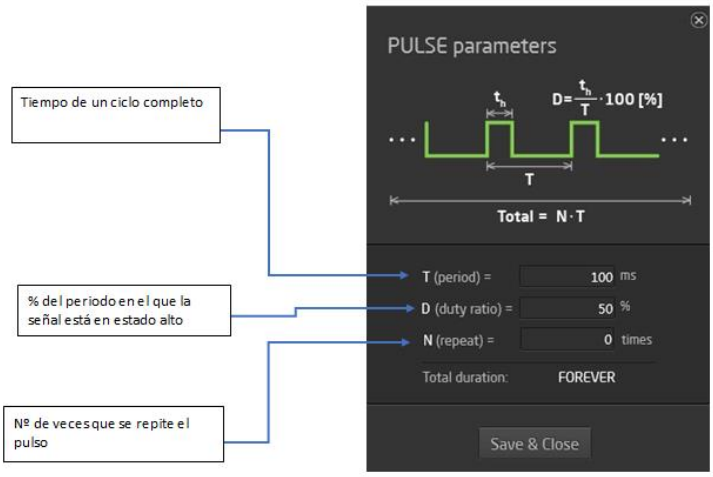


Ilustración 33 - Ajustes de las salidas tipo PULSE

5.3.4.4 Cargar el programa

Una vez determinados los parámetros, grabados los comandos y asignadas las funciones a cada pin sólo queda pasar el programa a la placa de voz

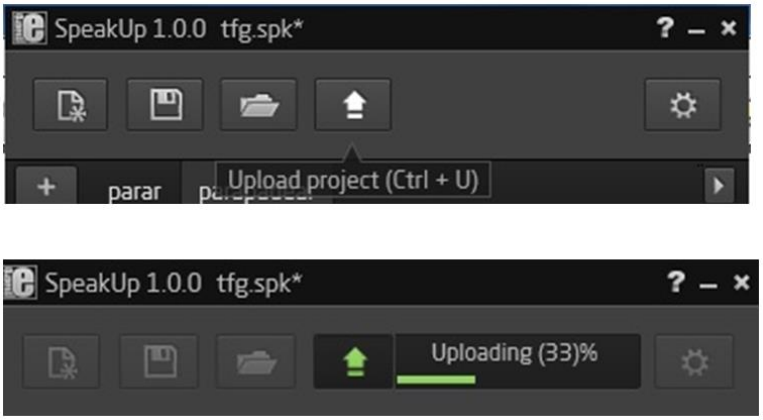


Ilustración 34 - Proceso de carga del programa

Con este proceso tenemos la tarjeta lista para funcionar.

5.4 Resultados del sistema de control por voz

5.4.1 Análisis de señales de voz

Cuando se graba un comando en la tarjeta ésta debe interpretar una señal analógica que varía sus características dependiendo de la persona que realice el entrenamiento, la intensidad de la voz al pronunciar el comando o el ruido ambiente, entre otros.

Por ejemplo, la orden sube_hombro nos proporciona la señal de la Ilustración 35

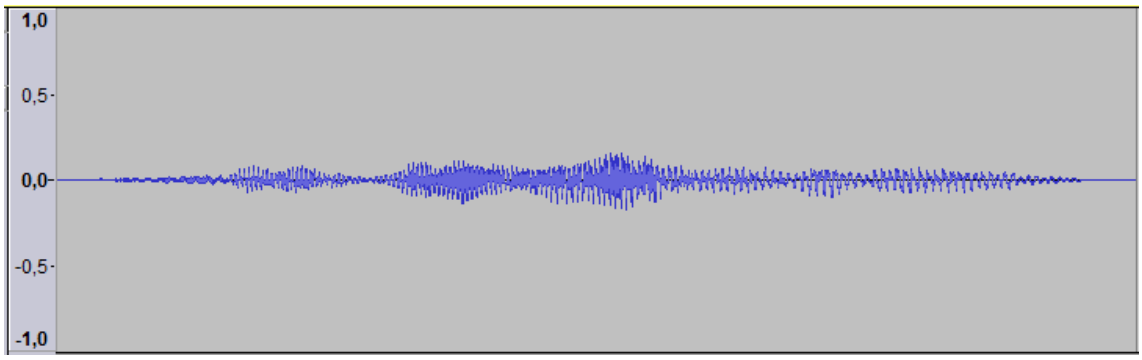


Ilustración 35 - Comando con énfasis suave

Este mismo comando con un énfasis mayor, presenta cambios:

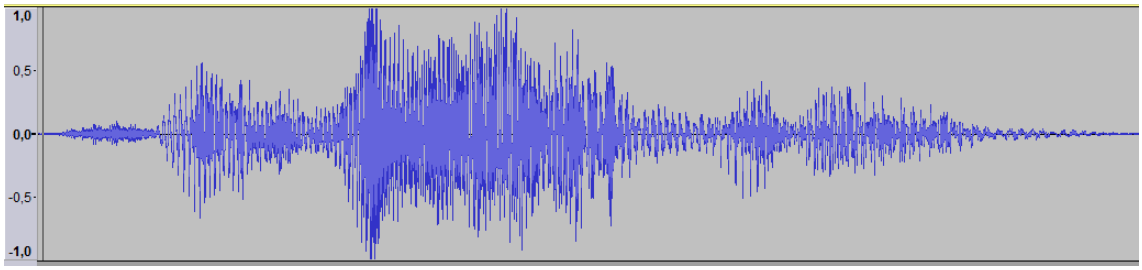


Ilustración 36 - Comando con énfasis fuerte

La amplitud de las señales es mayor e incluso presenta variaciones en la forma de la onda en los fonemas fuertes "br"

Si ahora añadimos ruido a la grabación del comando

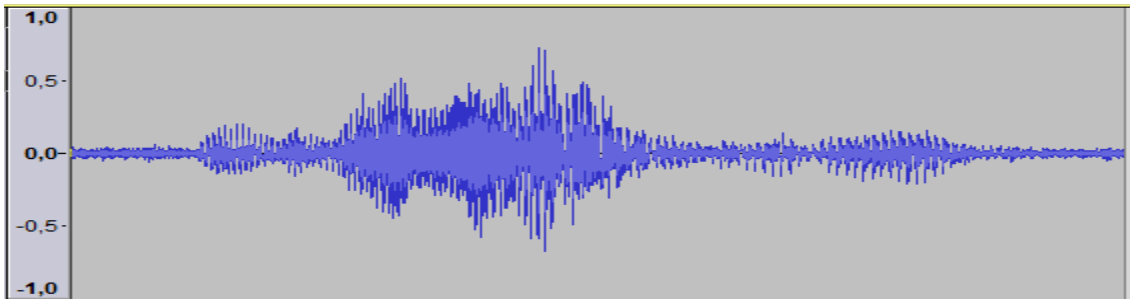


Ilustración 37 - Comando grabado con ruido

Observamos que en este caso la forma de la onda se superpone al ruido introducido.

Esto lleva a pensar que se deberá tener en cuenta estas variaciones para obtener el mayor grado de precisión y fiabilidad posible, pero que también se podrán utilizar para conseguir aplicaciones con mayor variedad de posibilidades.

En este apartado se va a configurar el software de la SpeakUp para que permita reconocer comandos con la mayor precisión posibles, identificando diferentes niveles de énfasis.

5.4.2 Pruebas del sistema de control de voz

En primer lugar, se analizó si el sistema es capaz de reconocer comandos de una palabra y comandos de dos palabras que utilizan fonemas diferentes. Los comandos se entrenarán con 2 niveles de énfasis diferentes que serán denominados *suave* y *fuerte* y debe ser capaz de distinguir a la persona que ha grabado el comando, en este caso se ha utilizado para la muestra voz masculina y femenina

La configuración de la tarjeta utilizada es:

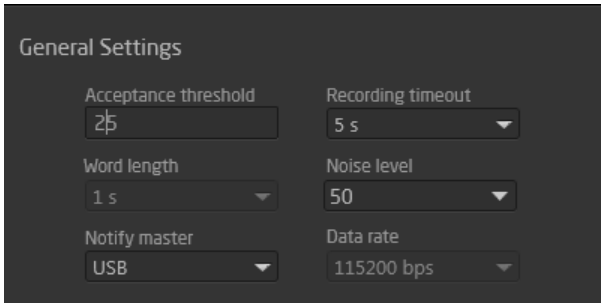


Ilustración 38 - Ajustes para la aplicación

En ambos casos, se ha utilizado 3 comandos que se codificarán con 2 bits y se señalizarán con leds de color verde. El bit más significativo nos indica el énfasis (0 con énfasis suave y 1 con énfasis fuerte) y se muestra mediante un led rojo. Y por último el bit menos significativo indica el sexo (0 si es mujer y 1 si es hombre), y se señaliza mediante un led amarillo. En total la prueba queda codificada con 4 bits.

En el caso de entrenamiento con una sola palabra se han utilizado los comandos: *abrir, para, inicializa.*

La Ilustración 39 muestra la codificación de la palabra 1 (abrir) con énfasis suave y voz femenina y la codificación de la palabra 3 (Inicializa) con énfasis fuerte y voz masculina.

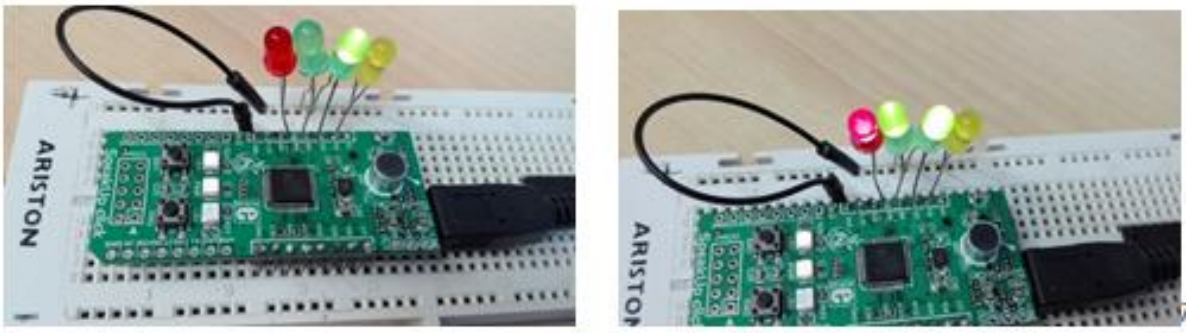


Ilustración 39 - Ejemplos de codificación (Izquierda: voz femenina, énfasis suave Derecha: voz masculina, énfasis fuerte)

Los resultados en cuanto a porcentaje de aciertos en la detección del comando se muestran en la siguiente tabla:

	FEMENINA		MASCULINA	
COMANDO	**SUAVE**	**FUERTE**	**SUAVE**	**FUERTE**
abrir 01	90%	100%	100%	70%
para 10	100%	90%	20%	100%
inicializa 11	70%	70%	20%	100%

Tabla 5 - Comparación entrenamiento una sola palabra

En esta experimentación se ha observado que la tarjeta detecta con el 100% de aciertos el sexo del usuario, es decir, si el entrenamiento se realiza con voz masculina o femenina.

En la Tabla 5 se ve que con voz femenina reconoce con precisión comandos con fonemas oclusivos tanto sonoros como sordos *p* y *b*, acompañados de fonemas vibrantes *r*. En este caso, no se presentan diferencias significativas entre el porcentaje de acierto cuando se varía el énfasis en la pronunciación.

En cambio, cuando los comandos se entrenan con voz masculina el sistema detecta con precisión palabras con fonemas oclusivos sonoros y vibrantes, en cambio identifica como énfasis fuerte comandos entrenados con énfasis suave, esto puede atribuirse a la diferencia de tono y timbre que presenta la voz masculina.

Para realizar la experiencia con comandos formados por 2 palabras se han utilizado *arranca_motor, posición_inicial, mover_base.* Los resultados que se obtienen son los siguientes:

	MUJER		**HOMBRE**	
COMANDO	**SUAVE**	**FUERTE**	**SUAVE**	**FUERTE**
arranca_motor 01	80%	100%	20%	70%
Posición_inicial 10	80%	90%	100%	100%
Mover_base 11	100%	100%	30%	100%

Tabla 6 - Comparación entrenamiento dos palabras

La tarjeta distingue con un 100% de acierto la diferencia entre voz masculina y femenina.

Las diferencias en este caso son más significativas, en el caso de voz femenina el sistema detecta con suficiente precisión comandos entrenados con énfasis suave y con énfasis fuerte, aunque en el segundo caso la detección es aún más precisa. En cambio, con voz masculina, si queremos utilizar niveles de énfasis, deberemos utilizar comandos que no contengan fonemas vibrantes, porque en caso contrario, la mayoría de las detecciones se realizan con énfasis fuerte.

5.4.3 Aplicación a un pequeño brazo de robot

Para la realización de pruebas en motores de corriente continua se ha utilizado el Kit Brazo robótico con mando de la marca *Cebek*. Consta de 5 motores (base, hombro, codo, muñeca y pinza), de los cuales sólo utilizaremos 4 durante las pruebas.

Ilustración 40 - Brazo robot utilizado para las pruebas

5.4.3.1 Identificación de comandos

Para la aplicación sobre un pequeño brazo robot se utilizarán los siguientes comandos:

Movimiento de la pinza: *Abre_pinza*, *Cierra_pinza*

Movimiento del codo: *Arriba, Abajo.*

Movimiento del hombro: *Avanza*, *Retrocede*

Movimiento de la base: *Derecha*, *Izquierda*

Detener motor que esté en movimiento: *Detente*

En total se necesitarán 9 comandos, para codificarlos serán necesarios 4 bits pero se utilizará uno más que permita grabar los comandos con 2 modos de énfasis que se traducirán en 2 niveles de velocidad de los motores. Por tanto, cada uno de estos comandos estará representado por 5 bits y el bit más significativo, indicará la velocidad (0 si la velocidad es baja y 1 si la velocidad es alta).

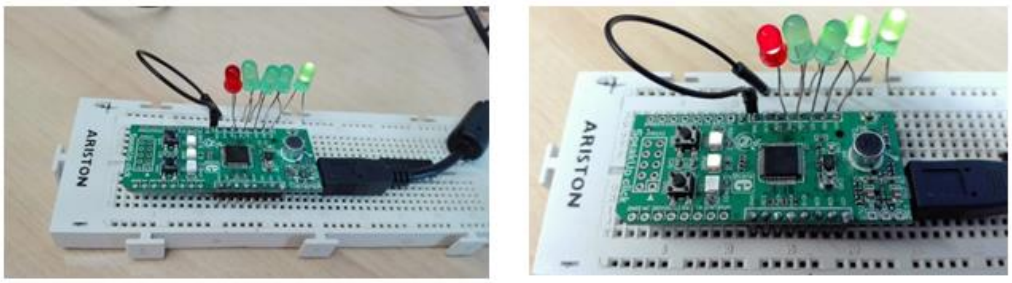


Ilustración 41 - Ejemplo de codificación (izquierda: comando con énfasis suave - derecha: comando con énfasis fuerte)

5.4.3.2 Codificación de los comandos

La codificación se realizó asignado los siguientes valores a los distintos comandos.

COMANDO	CODIFICACIÓN ÉNFASIS SUAVE	CODIFICACIÓN ÉNFASIS FUERTE
ABRE	00001	10001
CIERRA	00010	10010
ARRIBA	00011	10011
ABAJO	00100	10100
AVANZA	00101	10101
RETROCEDE	00110	10110
DERECHA	00111	10111
IZQUIERDA	01000	11000
DETENTE	11111	11111

Tabla 7 - Codificación de los comandos

Para poder visualizar la codificación mediante leds, se ha evitado utilizar la codificación 00000, además el comando *detente* no presenta dos modos de utilización por tanto tiene una única codificación. En la experimentación se ha decidido eliminar el énfasis fuerte en el movimiento de la pinza debido al pequeño recorrido de la articulación.

En la Tabla 7 - Codificación de los comandos se observa como para un mismo comando la codificación de los 4 primeros bits es la misma independientemente del énfasis, en cambio el MSB cambia dependiendo de éste. Por ejemplo para la orden *retrocede* el valor de los 4 primeros bits es para ambos casos 0110, en cambio dependiendo del énfasis su valor varía entre 00110 (énfasis suave) y 10110 (énfasis fuerte).

Para analizar el efecto del ruido ambiente en la fiabilidad de la detección se realizaron dos experiencias. En primer lugar, se forzó a la tarjeta a detectar el sonido ambiente en un entorno ruidoso y en segundo lugar se entrenaron los comandos introduciendo el ruido en las grabaciones. Es necesario mencionar que el ruido ambiente en esta experiencia es un factor importante debido a la proximidad del sistema de detección con los motores.

En el primer caso, los resultados no fueron favorables, en la situación en la que los motores están en marcha no era posible detectar los comandos con suficiente precisión. Se optó por realizar grabaciones de los comandos en diferentes ambientes, por tanto cada uno de los comandos se ha grabado con los dos niveles de énfasis y con y sin ruido.

El porcentaje de acierto en las pruebas realizadas con los comandos descritos se muestra en la siguiente tabla:

COMANDO	ÉNFASIS SUAVE %ACIERTOS	ÉNFASIS FUERTE %ACIERTOS
ABRE	100%	
CIERRA	100%	
ARRIBA	90%	90%
ABAJO	100%	100%
AVANZA	100%	70%
RETROCEDE	90%	90%
DERECHA	90%	90%
IZQUIERDA	80%	70%
DETENTE	100%	100%

Tabla 8 - Porcentaje de acierto de los comandos utilizados sin ruido

En la siguiente tabla se puede ver a que elemento del robot y a que movimiento están asociados cada uno de los comandos:

COMANDO	ELEMENTO DEL ROBOT	MOVIMIENTO
1 - ABRE	Pinza	Abrir pinza
2 - CIERRA	Pinza	Cerrar pinza
3 - ARRIBA	Codo	Subir codo
4 - ABAJO	Codo	Bajar codo
5 - AVANZA	Hombro	Movimiento hacia adelante
6 - RETROCEDE	Hombro	Movimiento hacia atrás
7 - DERECHA	Base	Movimiento a la derecha
8 - IZQUIERDA	Base	Movimiento a la izquierda
15 - DETENTE	Todos los motores	Parar

Tabla 9 - Movimiento y motor asociado a cada comando

5.4.3.3 Control de motores de DC.

Los motores que utilizaremos en la prueba del control por voz son de corriente continua. Para controlar estos motores desde la placa, se usará un driver para motores (L293D) para proporcionarle más corriente al motor ya que las salidas de la tarjeta sólo suministran 40mA. De esta manera, con el driver podemos alimentar el motor con una fuente de alimentación externa. El L293D tiene dos puentes H y proporciona 600mA al motor y soporta un voltaje entre 4,5V y 36V

Ilustración 42 - Driver L293D

Mediante el driver podemos realizar la conexión de dos motores conectados a respectivamente a los pines Y y controlados mediante los pines A conectados a las salidas digitales, la velocidad se controla mediante salidas PWM de la placa conectadas a los pines EN, además el driver permite alimentación externa para los motores en V_{CC2}.

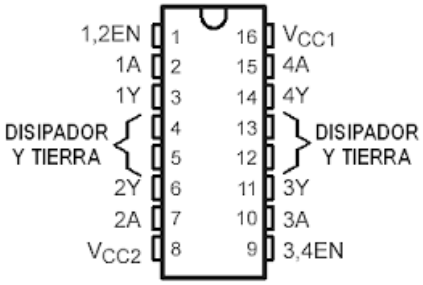


Ilustración 43 - Conexiones Driver L293D

El sentido de giro de los motores queda determinado por la combinación que aparece en los pines de entrada así tenemos:

	1 A	2 A	ESTADO MOTOR
Motor1 1Y- 2Y (conectado a 3,6) *ENABLE 1 -2 ACTIVADO*	0	0	PARADO
	0	1	GIRO
	1	0	GIRO INVERSO
	1	1	PARADO

	3 A	4 A	ESTADO MOTOR
Motor 2 3Y-4Y (conectado a 11,14) *ENABLE 3-4 ACTIVADO*	0	0	PARADO
	0	1	GIRO
	1	0	GIRO INVERSO
	1	1	PARADO

Tabla 10 - Combinaciones Driver L293D y sentidos de giro

5.4.3.4 Circuito de control

El circuito de control necesario para controlar 4 motores de corriente continua consta de 2 drivers L293D, la tarjeta STM32F407 Discovery y la tarjeta de voz SpeakUp. El circuito completo puede encontrarse en Plano 6: Plano de conexiones control por voz.

Las salidas de la placa SpeakUp IO1, IO2, IO3, IO4 e IO5 se conectan a los pines PE0, PE1, PE2, PE3 y PE4 de la tarjeta STM32F407 respectivamente. En estos pines se transmiten los comandos de voz, así como el énfasis con el que son pronunciados por el usuario (énfasis suave- énfasis fuerte). Los pines de la tarjeta STM32F407 están configurados como pines de entrada digitales.

En el circuito se observa como en cada una de las conexiones entre la tarjeta de voz y el microcontrolador ST hay un led. Éstos se han utilizado para verificar el correcto funcionamiento durante las pruebas con la SpeakUp, siendo los 4 primeros la codificación del comando y el quinto (MSB) el marcador del énfasis.

La tarjeta de voz está alimentada a 3.3 V. Para conseguir este voltaje se ha utilizado la salida de 3.3 V con la que cuenta la STM32F407. También se han unificado las masas de ambas tarjetas para evitar problemas de desacoplo.

Por otro lado, y tal como se explicará en el punto 5.4.3.6 las entradas de activación de los drivers (pines *Enable*) están conectados a los pines PD12, PD13, PD14 y PD15. Estos pines están configurados como salidas PWM, y se utilizarán para controlar la velocidad a la que debe moverse el motor dependiendo del énfasis. Además estos pines están conectados a los Leds de la placa STM32F407, con lo que durante las pruebas pudimos comprobar el correcto funcionamiento de estos puertos antes de conectar los motores.

Las 8 entradas de los drivers (4 para el driver 1 y 4 para el driver 2) están conectadas a los pines PB0, PB1, PB5, PB6, PB7, PB8, PB9 y PB10. Dentro del puerto B se descartó el uso de los pines PB2, PB3 y PB4 ya que éstos cada vez que se presiona el botón de Reset, tienen una función reservada por la misma placa (por ejemplo, el puerto PB3 y PB4 configuran el TRACESWO, y si se usaban estos pines como pines digitales de carácter general nos era imposible utilizar la herramienta de depuración). En la siguiente tabla se muestra como estos pines tienen funciones alternativas:

Pin number						Pin name (function after reset)(1)	Pin type	I / O structure	Notes	Alternate functions	Additional functions
LQFP64	WLCSP90	LQFP100	LQFP144	UFBGA176	LQFP176						
55	B6	89	133	A10	161	PB3 (JTDO/ TRACESWO)	I/O	FT	-	JTDO/ TRACESWO/ SPI3_SCK / I2S3_CK / TIM2_CH2 / SPI1_SCK/ EVENTOUT	-
56	A6	90	134	A9	162	PB4 (NJTRST)	I/O	FT	-	NJTRST/ SPI3_MISO / TIM3_CH1 / SPI1_MISO / I2S3ext_SD/ EVENTOUT	-
27	H7	36	47	R4	57	PB1	I/O	FT	(4)	TIM3_CH4 / TIM8_CH3N/ OTG_HS_ULPI_D2/ ETH_MII_RXD3 / TIM1_CH3N/ EVENTOUT	ADC12_IN9
28	J7	37	48	M6	58	PB2/BOOT1 (PB2)	I/O	FT	-	EVENTOUT	-

Tabla 11 - Funciones alternativas puerto B

En la Tabla 11 también observamos como por el ejemplo el pin PB1, no tiene ninguna función alternativa asociada y por tanto no habrá problema en utilizarlo.

Todos los pines de entrada a los drivers en la aplicación están configurados como pines de salida digitales de carácter general.

Las salidas de los Drivers están conectadas a cada una de los conectores de los motores de corriente continua.

La alimentación externa del Driver a 9V se ha realizado con una pila de este voltaje, ya que con ella podemos conseguir tanto la tensión como la corriente deseada para que los motores puedan moverse. Esta alimentación se conecta al pin Vcc_2 del driver. Tal y como se ha hecho anteriormente con las masas de la tarjeta de voz y la STM32F407, en este caso también se han unificado las puestas a tierra para evitar problemas.

Por último cabe mencionar que la salida Vcc_1 de ambos drivers está conectado al pin PB15 de la tarjeta STM32F407. Esta conexión se ha realizado para controlar la alimentación de los drivers, hasta que dicho pin no está a nivel alto los chips no están alimentados y por tanto no están en funcionamiento. Esta conexión se hizo ya que durante la realización de las pruebas, las corrientes de fuga provenientes de los motores y de las tarjetas alimentaban de forma indeseada los drivers de los motores, consiguiendo activarlos y moviendo de forma aleatoria los motores. Este pin se ha configurado como salida digital.

5.4.3.5 Tarjeta STM32f407VG Discovery

Esta es una tarjeta de carácter general que utilizaremos a fin de controlar la ejecución de las órdenes transmitidas a través de la tarjeta de voz SpeakUp.

La placa STM32F4DISCOVERY ofrece las siguientes características:

- Microcontrolador STM32F407VGT6 con 1 MB de memoria flash, 192 KB de RAM, encapsulado LQFP100.
- ST-LINK/V2 incorporado con selector usar el kit como un ST-LINK/V2 independiente (con conector SWD para programación y depuración).
- Fuente de alimentación: a través del bus USB o desde una fuente de alimentación externa de 5V.
- Sensor de movimiento ST MEMS LIS302DL, acelerómetro con salida digital de 3 ejes
- Audio DAC CS43L22 con controlador integrado de altavoz clase D
- Ocho LEDs: - LD1 (rojo / verde) para la comunicación USB - LD2 (rojo) alimentación 3,3 V - Cuatro LEDs de usuario, LD3 (naranja), LD4 (verde), LD5 (rojo) y LD6 (azul) - 2 LEDs USB OTG LD7 (verde), VBus y LD8 (rojo)
- Dos pulsadores (usuario y Reset)
- USB OTG con conector micro-AB

Y el microcontrolador STM32F407VGT6 incorporado a la tarjeta se caracteriza por:

- Procesador ARM Cortex-M4 32-bit
- FPU tiene 210 DMIPS,
- Memoria de 1 MB Flash, 196 KB RAM,
- Conexiones USB OTG HS/FS, Ethernet,
- Timers 17 TIMs
- 3 ADCs.

De todas estas características la más significativa para la aplicación es la posibilidad de generar pulsos PWM en algunos Timers con el fin de controlar la velocidad de los motores de DC.

Las características completas esta tarjeta se encuentran en el punto 0

5.4.3.6 Configuración de pines PWM

La modulación de ancho de pulso, PWM, de una señal es una técnica que logra producir el efecto de una señal analógica sobre una carga a partir de la variación de la frecuencia y ciclo de trabajo de una señal digital. El ciclo de trabajo describe la cantidad de tiempo que la señal está en un estado lógico alto como un porcentaje del tiempo total que éste toma para completar un ciclo completo. Se utiliza para enviar información o para modificar la cantidad de energía que se envía a una carga. Este tipo de señales es muy utilizada en circuitos digitales que necesitan emular una señal analógica, por ejemplo para variar la luminosidad de un led o controlar la velocidad de un motor DC.

La técnica de PWM consiste en producir un pulso rectangular con un ciclo de trabajo determinado, este ciclo de trabajo puede variar de 0 a 100%. Un ciclo de trabajo del 0% significa que la señal siempre está nivel bajo; y un ciclo de trabajo del 100% significa la señal siempre en nivel alto.

Para emular una señal analógica se cambia el ciclo de trabajo (Duty cicle) de tal manera que el valor promedio de la señal sea el voltaje aproximado que se desea obtener, pudiendo entonces enviar voltajes entre 0[V] y el máximo que soporte el dispositivo PWM

D = ciclo de trabajo

t = tiempo en que la señal es positiva

T = Período $D = \frac{t}{T} * 100$

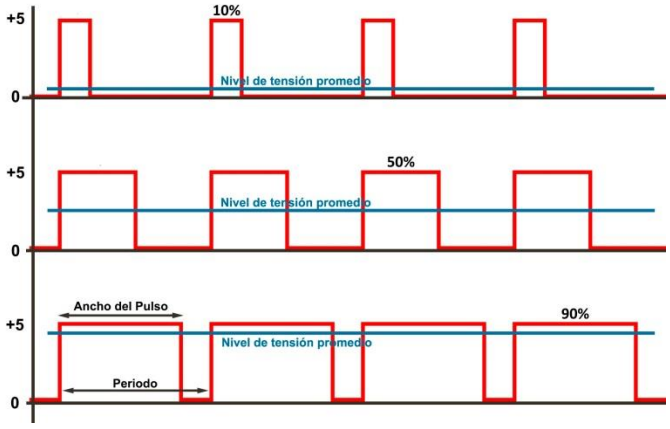


Ilustración 44- Ejemplo de PWM

La tarjeta STM32F407VG Discovery dispone de Timers que permiten definir la frecuencia de trabajo. A partir del ancho de pulso indicado, que queda almacenado en un registro, se compara con un contador para activar la salida a estado alto. Algunos Timers puede activar hasta 4 salidas PWM una por cada canal, controlando el ciclo de trabajo de forma independiente.

No todos los Timers pueden activar salidas PWM, según el datasheet de la tarjeta dos de los Timers que pueden generar salidas PWM son el Timer 3 y el Timer 4, (posteriormente al utilizar únicamente 4 motores se utilizó el Timer 4). En la Tabla 12 se muestra los pines que están conectados a cada uno de los canales de este Timer:

Timer	Canal 1	Canal 2	Canal 3	Canal 4
Timer 4	PB6, PD12	PB7, PD13	PB8, PD14	PB9, PD15

Tabla 12 - Pines conectados al Timer 4

Ta y como se ha explicado anteriormente se han escogido para el proyecto los pines PD12, PD13, PD14 y PD15 para nuestras salidas PWM, de esta forma se puede observar las salidas con los leds de la placa a fin de verificar fácilmente su funcionamiento.

Para configurar el Timer se ha elegido un prescaler con un valor de 24 (este valor se utiliza para ajustar la frecuencia de trabajo) y un periodo de 255, de esta forma el ciclo útil podrá variar desde 0 a 255 (con 255 tenemos el pin de salida continuamente a nivel alto). Con estos valores se puede distinguir claramente la variación en la velocidad de los motores al variar el ciclo de trabajo.

Con estos datos el periodo de la señal PWM se calcularía de la siguiente forma: teniendo en cuenta que trabajamos a una frecuencia de 168MHz que en el bus utilizado se divide por 2.

Frecuencia Timer 4: 84 MHz ajustamos mediante el valor elegido de prescaler y la señal de entrada al Timer sería:

$$f_{TIMER4} = \frac{84MHZ}{24} = 3.5\ MHz$$

Como en cada periodo tenemos 255 divisiones, la duración de éstas será de 285.71 ns.

La frecuencia de la señal PWM se calcula

$$T_{PWM} = 285.71\ ns * 255 = 72.86\ \mu s \rightarrow f_{PWM} = 13.7\ MHz$$

El valor elegido de prescaler se ha tomado de forma aleatoria ya que el objetivo es distinguir claramente las salidas PWM sin preocuparnos del valor exacto de la frecuencia de la señal. No obstante en otros proyectos si podría ser relevante el valor de la frecuencia PWM con lo que se debería calcular el prescaler y el periodo de forma más precisa.

Con todo ello el proceso realizado se detalla a continuación:

El Hardware utilizado para poder llevar a cabo las órdenes dadas por la tarjeta de voz será la tarjeta STM32F407-Discovery ampliamente utilizada, sus características más relevantes para nuestro proyecto se han comentado anteriormente.

Respecto al software, se ha realizado un proyecto con Keil v 5.0, utilizado la herramienta STM32F4 CubeMX.

Tras seleccionar la placa adecuada, hemos seleccionado los Timer 3 y 4 para su inicialización con PWM (Pulse Width Module), cada Timer dispone de 4 canales, y cada canal podrá controlar la velocidad de un motor, según el valor de la propiedad *pulse* del Timer. Veremos estos detalles posteriormente.

En principio se utilizaron dos Timers a fin de controlar 5 motores: base, hombro, codo, muñeca y pinza, aunque finalmente tan solo ha sido necesario utilizar 4 ya que la articulación de la muñeca no se utiliza.

5.4.3.7 Utilización del STM32CubeMX

En primer lugar y tras seleccionar nuevo proyecto, seleccionamos la placa que utilizaremos y la línea adecuada.

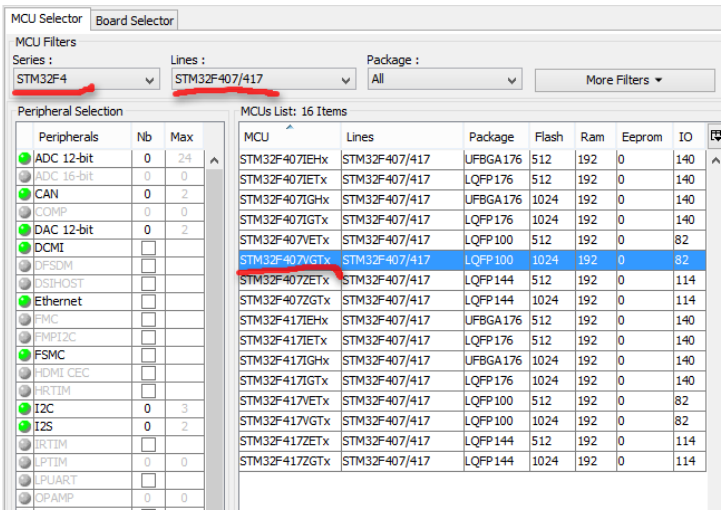


Ilustración 45 - Selección del dispositivo en el STM32 CubeMX

Como vemos se selecciona la tarjeta STM32F407VGT que es la placa que se va a utilizar en esta ocasión.

Tras hacer doble click sobre la placa, aparece el esquema del micro. Se seleccionará el Timer 4 y los canales que se van a usar para controlar la velocidad de los motores (definiremos los canales como salida PWM). También se seleccionarán los pines que se utilizarán para las conexiones con la tarjeta de voz y con los drivers que controlan los motores, detallados en el apartado 5.4.3.6.

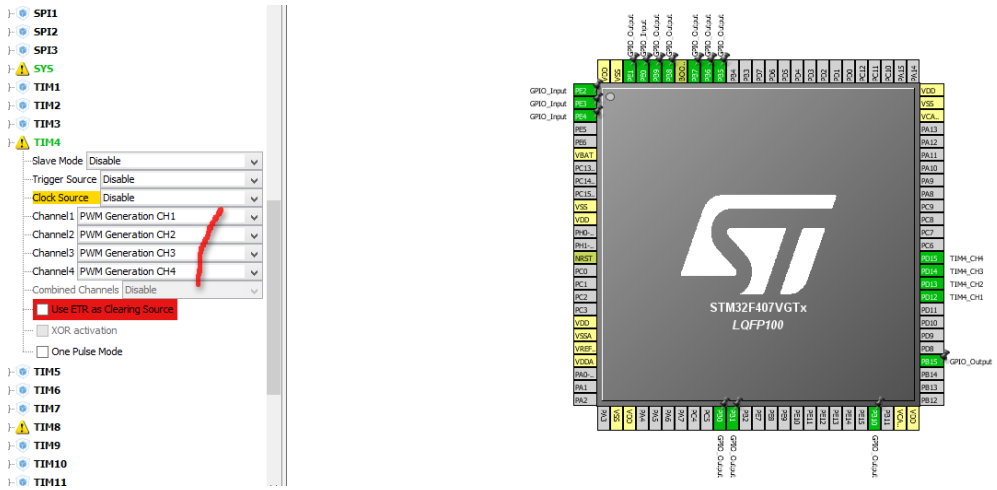


Ilustración 46 - Selección de los canales en el STM32 CubeMX

A continuación se configura el Timer, tras entrar en el apartado de configuración:

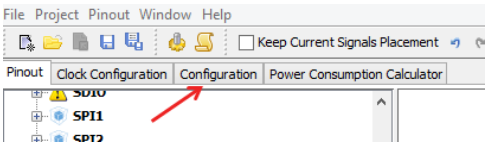


Ilustración 47 - Selección de la configuración de los canales seleccionados

Seleccionamos la configuración para el Timer 4

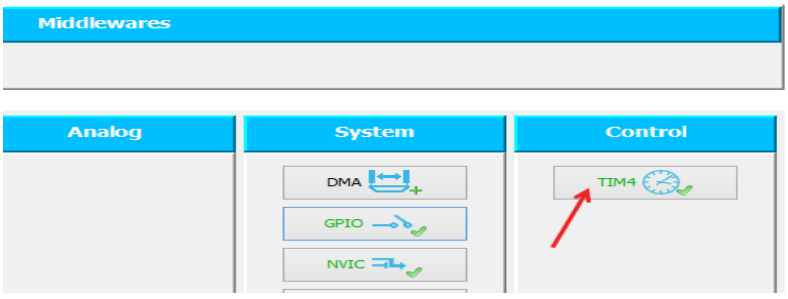


Ilustración 48 - Selección de la configuración del TIimer

Aparece la pantalla de configuración del Timer 4 y de los cuatro canales, introducimos un valor de 24 en el prescaler y 255 en el apartado de periodo, (el valor de PWM podrá variar entre 0 y 255).

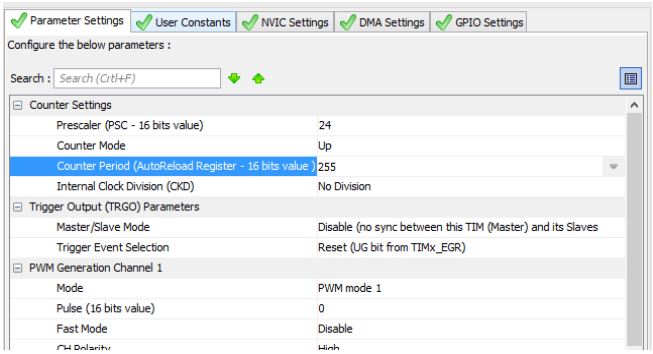


Ilustración 49 - Valores configurados para la aplicación (prescaler, periodo...)

Tras la configuración grabamos los cambios e indicamos que el proyecto es para MDR Keil5

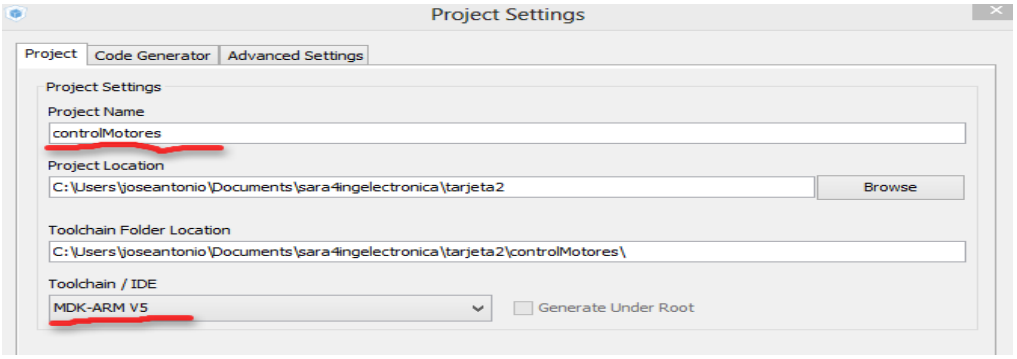


Ilustración 50 - Configuración para conseguir el código en Keil5

En la pestaña *Code Generator* seleccionamos la opción de incluir tan solo las librerías necesarias.

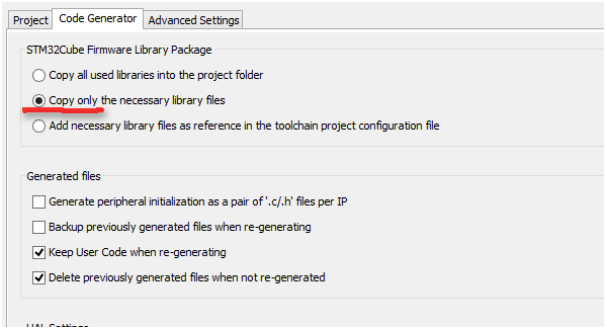


Ilustración 51 - Configuración para incluir únicamente las librerías necesarias

Tras completar la inicialización generamos el código, para ello pulsamos la pestaña de generación de código.

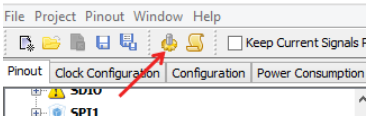


Ilustración 52 - Opción de generación de código

Con esto genera el proyecto con los periféricos inicializados. Ahora sólo habrá que implementar el algoritmo diseñado para el control de los motores, a partir de la codificación recibida desde la tarjeta de voz. Es importante revisar el código generado a fin de verificar que la inicialización se ha realizado correctamente.

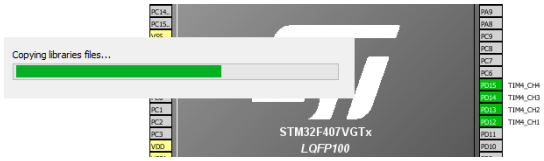


Ilustración 53 - Generación del código

El código completo se puede encontrar en el anexo 11.2 Código modulo control por voz.

5.4.3.8 Algoritmo utilizado.

En primer lugar se codifican las órdenes utilizadas por la tarjeta de voz.

Tal y como se ha explicado en el apartado 5.4.3.2 se utilizan 4 bits de codificación y un quinto que indicará la énfasis del comando.

A fin de facilitar la conexión se renombraron los pines de la tarjeta de voz con el nombre de los pines a los que se conectarán en la STM32F407, empezando por el PE0 para la salida 1 de la tarjeta y terminando por la salida 5 renombrada como énfasis

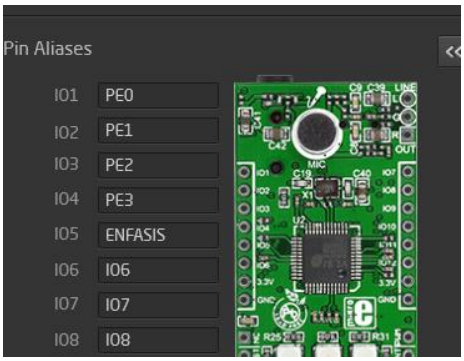


Ilustración 54 - Pines renombrados

Por tanto la tarjeta de voz transmitirá a través de los pines PE0 A PE3 la orden transmitida y a través del pin PE4 el énfasis empleado.

El algoritmo seguido sería el siguiente:

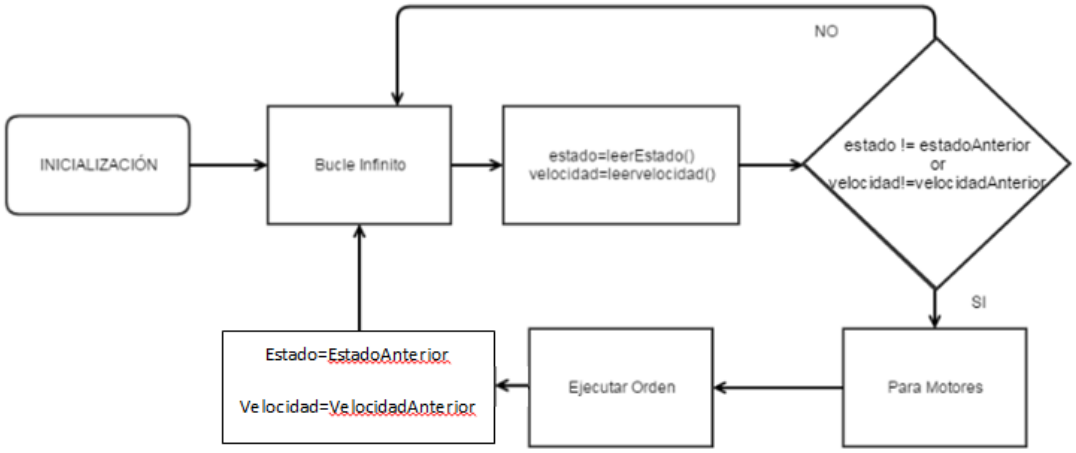


Diagrama de Bloques 4 - Algoritmo control por voz

Veamos ahora como se ha implementado cada parte del algoritmo anterior.

5.4.3.8.1 Inicialización

La mayor parte de este primer código lo ha generado el programa STM32F4 CubeMx. Tan solo se ha tenido que inicializar explícitamente la parte del PWM para que empieza a general pulso el Timer4 (funciones HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_x)).

```
HAL_Init();

/* Configure the system clock */
SystemClock_Config();

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM4_Init();
MX_TIM3_Init();

//activamos el pin de alimentacion del chhip de los motores

HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);

  /* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4);   // ACTIVAMOS  EL PWM
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_3);   // ACTIVAMOS  EL PWM
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_2);   // ACTIVAMOS  EL PWM
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_1);   // ACTIVAMOS  EL PWM
HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_4);   // ACTIVAMOS  EL PWM

   printf("empezamos\n");
```

Inicializado por el generador de codigo

Inicializado por el usuario

Todas las funciones fueron implementadas automáticamente mediante el generador de código, a excepción de las marcadas como inicializadas por el usuario, que deben ser específicamente introducidas.

Se observa que hay que utilizar el pin 15 del puerto B. Tal y como ha visto en el apartado 5.4.3.4, este pin está conectado al pin de alimentación de los drivers de los motores (Vcc_1). La función de este pin es activar los controladores únicamente cuando el programa esté cargado y todos los pines inicializados de lo contrario, mientras se cargaba el programa se ponían en marcha los motores de una forma aleatoria e indeseada.

Destacar la gran ayuda que ha supuesto estas inicializaciones de los periféricos que han permitido centrar el esfuerzo en la programación de la parte del algoritmo que atañe exclusivamente a la activación de los motores según el estado y la velocidad leída.

5.4.3.8.2 Bucle infinito

Tras la inicialización entramos en un bucle infinito que controlará las funciones de la tarjeta una vez inicializada. Simplemente el bucle infinito se implementa mediante un While(1)

```
  while (leerBoton()!=1){

  }

    HAL_Delay(1000);  // pa

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CODE END WHILE */
```

Medida de seguridad

En el código se observa que se ha añadido una medida de seguridad con el fin de que ningún motor se activara por un descuido al inicio del programa. Por tanto, mientras no se pulse el botón de usuario (botón azul de la tarjeta, conectado al pin PA0) el cuerpo del programa no se ejecutará

5.4.3.8.3 Cuerpo del programa

```
estado=leerEstado();
velocidad=leerVelocidad();

  if (estado!=estadoAnterior || velocidad!=velocidadAnterior){
    estadoAnterior=estado;
    velocidadAnterior=velocidad;

  paraMotores();      // paramos el motor que teníamos en marcha (paramos todos)

  switch(estado){

    case 1:
```

Una vez dentro del bucle infinito las primeras funciones que se encuentran son *leerEstado* y *leerVelocidad*.

En la primera se leen los pines de la STM32F4 PE3, PE2, PE1 y PE0 conectados a la tarjeta de voz con el fin de almacenar el comando enviado. Se pasa la lectura a decimal, se almacena en la variable estado y este valor es devuelto por la función.

En esta función nos aseguramos de que el valor leído es estable. Solo se devuelve el estado si el valor leído desde los pines (PE3... PE0) es el mismo durante 5 lecturas consecutivas. Nos encontramos que al ser la velocidad de la tarjeta de voz muy inferior a la velocidad de la STM32F407, esta última detectaba la transición de un estado a otro moviendo el motor incorrecto durante un corto instante de tiempo. El algoritmo seguido ha sido el siguiente:

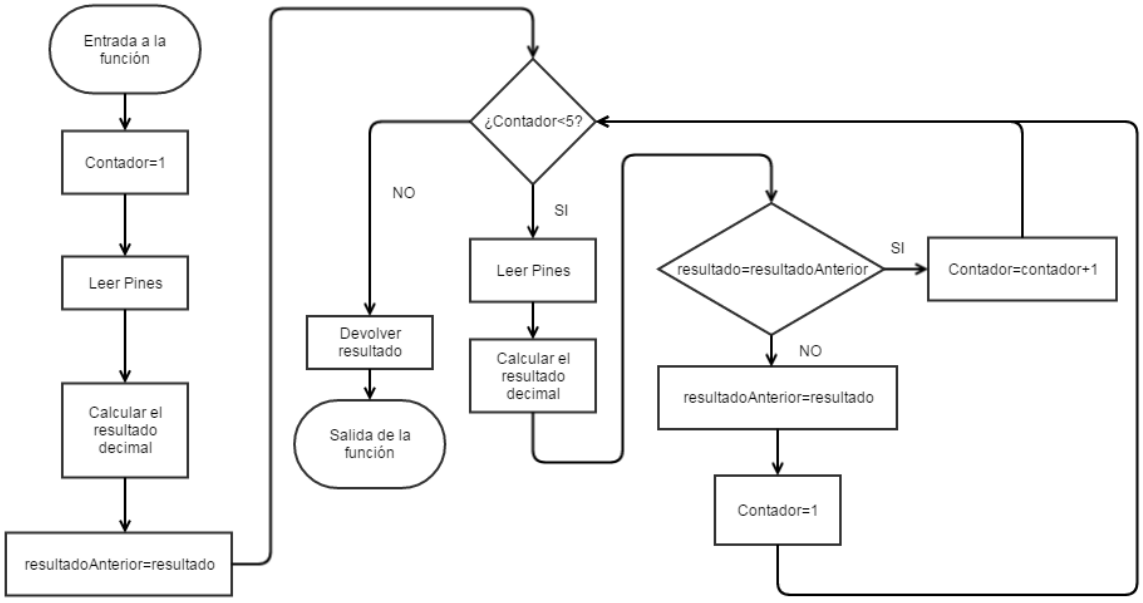


Diagrama de bloques 5 - Función leerEstado

La segunda función se usa para leer la velocidad a la que debe ser ejecutado el comando. Se lee el pin PE4 conectado también la tarjeta de voz, si ese pin está activo (a 1) la orden debe ser ejecutada a velocidad alta, si está nivel bajo (a 0) significaría que debe ser ejecutada a velocidad baja, este valor (0 o 1) se almacena en la variable *velocidad*, si utilizáramos más de 1 bit se podría trabajar con un conjunto de varias velocidades para un mismo movimiento.

A continuación se comprueba si ha cambiado el estado o la velocidad respecto de la medida anterior, si no ha cambiado el sistema no hace ningún cambio. Cuando ha cambiado el estado o la velocidad el sistema cambia bien el motor en movimiento, o bien la velocidad y pasa a ejecutar la orden correspondiente.

Ambas funciones se pueden ver en el código completo del anexo 11.2.

Antes de estudiar cómo se implementa cada estado, nos fijamos en el código, que llamamos a la función *paramotores( ).* Esto impedirá que se muevan varios motores a la vez, se ha implementado de esa forma por sencillez.

Finalmente veamos cómo se ha llevado a cabo la ejecución para cada estado, esto lo podemos ver en el interior del switch.

```
switch(estado){

   case 1:
   case 2: moverPinza(estado,255);
           HAL_Delay(50);
           if(velocidad==1)
                moverPinza(estado,VELOCIDADALTA1+10);
           else
                moverPinza(estado,VELOCIDADBAJA1+10);
           break;

   case 3:    //arriba
           moverCodo(estado,255);
           HAL_Delay(50);
           if(velocidad==1)
                moverCodo(estado,VELOCIDADALTA2);
           else
                moverCodo(estado,VELOCIDADBAJA2);

           break;
```

Como hemos comentado en el apartado 5.4.3.2, vemos como los estados están asociados a los diferentes elementos del robot, aquí vemos como el estado 1 y 2 están asociados al movimiento de la pinza, el estado 1 abrirá la pinza y el estado 2 la cerrará.

Durante la realización de diferentes pruebas se detectó que algunos motores como el hombro y el codo, que soportan mucho peso, no podían comenzar el movimiento cuando la potencia era baja, por tanto se decidió empezar el movimiento con máxima potencia durante un breve instante de tiempo (50 ms) para evitar este problema.

También se ha ajustado la velocidad para cada motor (cada estado) a fin de que se pudiese estimar claramente la diferencia de velocidades. Se han definido dos juegos de velocidades alta y baja, según qué motor se debe accionar se utiliza un juego o el otro.

Dentro de cada estado vemos como se acciona el motor a velocidad alta o baja según el valor leído por la variable *velocidad*.

Se han implementado una serie de funciones que nos ejecutarán la orden del movimiento de los motores. Veamos el código para la pinza

```
void moverPinza(int sentido,int velocidad)
){
  // primero activamos el PWM asociado a la pinza canal2 del timer4  corresponde al pin: PD13
   __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_2,velocidad);
      HAL_Delay(10);
  // ponemos el sentido
```

```
]   switch(sentido){
      case 1: // abrimos pinza

         HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN1,GPIO_PIN_RESET);
         HAL_Delay(10);
         HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN2,GPIO_PIN_SET);
         HAL_Delay(10);
           break;
      case 2: //cerramos pinza
        HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN1,GPIO_PIN_SET);
         HAL_Delay(10);
         HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN2,GPIO_PIN_RESET);
         HAL_Delay(10);
           break;

      case 15: // paramos pinza
        HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN1,GPIO_PIN_SET);
        HAL_Delay(10);
        HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN2,GPIO_PIN_SET);
        HAL_Delay(10);
           break;

   }     // fin del switch


}  // fin de mover pinza
```

En primer lugar se inicializa la salida PWM utilizando como ciclo de trabajo el valor de la variable local *velocidad*. Hay que tener en cuenta que para inicializar esta salida, habrá que configurar el canal del Timer 4 asociado a este motor (en el caso de la pinza utilizamos el canal 2).

Cada motor tiene un canal asignado, por lo que podrían moverse cada uno de ellos a una velocidad diferente según la inicialización del canal correspondiente.

Cada motor se controla desde 2 pines digitales, tal como se ha comentado anteriormente en el apartado 5.4.3.4. Según el valor de estas entradas a los drivers que controlan el motor, éste se mueve en un sentido o en otro.

Se ha definido un estado 15, a fin de que se parara este motor. Actualmente el estado 15 detiene todos los motores, pero se podría, de forma sencilla, asignar un estado de parada para cada motor por separado.

Tras cada escritura en los pines de la tarjeta se ejecuta un *delay* a fin de asegurarnos que se ejecute la orden correctamente.

El resto de motores se implementan de forma idéntica. Se puede consultar el código en el anexo 11.2.

Finalmente la función *paraMotores()* es implementada de tal forma que fija una potencia de 0 a cada salida PWM y asigna un valor alto a las salidas digitales que están conectadas al driver.

6 CONCLUSIONES Y TRABAJO FUTURO

6.1 Conclusiones

La detección por voz ofrece soporte a una grandísima cantidad de aplicaciones. El auge en la investigación en el campo del aprendizaje de máquinas, una renovada atención a métodos estadísticos y la gran potencia de los ordenadores actuales han potenciado este hecho. La tarjeta de voz SpeakUp Click, utilizada en este proyecto es un buen ejemplo de ello.

En este trabajo se ha propuesto la manera de utilizar el reconocimiento por voz como forma de control de movimiento de un robot de asistencia a minusválidos. La base de desarrollo del proyecto con la utilización de las librerías HAL, permite con pequeñas modificaciones que el sistema pueda ser adaptado a diferentes hardwares de la familia ST.

El kit de desarrollo STM32F4-Discovery permite implementar una solución eficaz a esta propuesta a través de la generación de salidas PWM. Estas salidas nos permiten poder controlar la velocidad de forma independiente de cada uno de los motores del robot.

Tanto el microcontrolador STM32F407VGT como el STM32F429I encargados de dirigir la comunicación con la tarjeta de voz para posteriormente enviar las medidas captadas al ordenador presentan características que los convierten en los dispositivos idóneos para futuros desarrollos sobre este trabajo. Ambos microcontroladores son circuitos integrados de núcleos ARM Cortex-M4. La familia ARM se caracteriza por utilizar arquitectura RISC que permite una buena velocidad de procesamiento y un bajo consumo debido a su registro de instrucciones sencillas capaz de ser ejecutadas generalmente en un solo ciclo de reloj, este hecho da como resultado un circuito de reducidas dimensiones, que utiliza un número bajo de transistores y que por tanto produce un consumo relativamente bajo.

En cuanto al estudio del valor de la aceleración de las articulaciones del robot los componentes utilizados son una buena línea de comienzo para un futuro desarrollo que implique la lectura de éste valor como se verá en el siguiente apartado.

6.2 Trabajo futuro

En este trabajo se ha visto algunas opciones para la obtención de la conversión ADC de la aceleración. Este proyecto ha dejado abierto el desarrollo sobre la parte de comunicación entre un convertidor ADC externo y la tarjeta STM32F429I por protocolo SPI y la verificación del funcionamiento de la placa PCB diseñada. Por lo que una de las ampliaciones tendrá que ver con este apartado.

Respecto al apartado de reconocimiento por voz lo más adecuado es elegir un sistema que permita la programación del sistema de reconocimiento con el fin de mejorar sus atribuciones y conseguir un sistema más seguro.

También debería adaptarse el circuito de control del reconocimiento de voz a diferentes tipos de motores (paso a paso, motores AC…), así como realizar los cambios necesarios en el código para utilizar el microcontrolador STM32F429I.

7 BIBLIOGRAFIA

[1] STMicroelectronics, «Discovery Reference Manual,» RM0090.

[2] STMicroelectronics, «Datasheet-Production data STM32F429xx» DM00071990.

[3] STMicroelectronics, «User Manual UM1725» DM00105879.

[4] MikroElektronika, «User Manual SpeakUp» speakup-click-manual-v100.

[5] A. Albert, «Diseño, Implementación y Control de un actuador elástico en serie,» ETSID (UPV), Valencia, 2014.

[6] T. Majerle. [En línea]. Available: http://stm32f4-discovery.com/.

[7] A. Aliques, «Disseny i implementació de l'arquitectura software, funcions prmitives del movimient I control d'un braç robòtic» ETSID (UPV), Valencia, 2015.

[8] J. Panta, «Control domótico por voz», ETSINF (UPV), Valencia, 2012.

[9] Jurasfky, Daniel y Martin, James H. «Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistincs, and Speech Recognition.» Pearson Prentice Hall, 2009.

[10] Hugh, H. «Exoskeletons and orthoses: classification, design challenges and future directions» *Journal of NeuroEngineering and Rehabilitation*, 2009.

[11] Daniel P Ferris. «The exoskeletons are here» *Journal of NeuroEngineering and Rehabilitation*, 2009.

[12] Aaron M.Dollar y Hugh Her (2008). «Lower Extremity Exoskeletons and Active Orthoses: Challenges and State-of-the-Art»,*IEEE Transactions on robotics*, vol.23, nº 1.

[13] Centro de Investigación Biomecánica en Rehabilitación y robótica. [En línea]. Available «http://www.ciberr-m317.com/»

[14] Lumevox [En línea]. Available «http://www.lumenvox.com/espanol/resources/ »

[15] Artículos Informativos USA [En línea]. Available «http://www.articulosinformativos.com/Reconocimiento_de_Voz-a963743.html»

[16] STMicroelectronics, «Datasheet-Production data STM32F415xx», DM00035129.

[17] STMicroelectronics, «Datasheet-Production data LIS344ALH», CD00182781.

[18] Ligarser Roset, J. [En línea]. Available «http://xtec.cat/~jlagares/»

[19] Barrios Cárdenas, J. (2012) «Sistema de control de un manipulador mecánico por detección de voz» *Redes de ingeniería*, vol. 3 nº 2.

[20] Villaluenga Morán J.L. «Uso de acelerómetros para control de dispositivos mediante captura de movimiento», Universitat Ramon Llull (2015)

8 ILUSTRACIONES

9 TABLAS

10 DIAGRAMAS DE BLOQUES

11 ANEXOS

11.1 Código módulo convertidor STM32F429

```
1
2   #include "ConvertidorADC.h"
3   #include "stm32f429i_discovery.h"
4   #include "stm32f4xx_hal.h"
5
6
7   // Inicialización Error
8
9   static void Error_Handler (void)
10  {
11    //BSP_LED_On(1);
12    while(1)
13    {
14    }
15  }
16
17  IO uint16_t ADCConvertedValue=0;
18
19  void ADC_init(void){
20    ADC_HandleTypeDef AdcHandle;
21    GPIO_InitTypeDef GPIO_InitStruct;
22    static DMA_HandleTypeDef hdma_adc;
23
24    // Enable peripherals and GPIO Clocks
25    __HAL_RCC_GPIOA_CLK_ENABLE();
26    __HAL_RCC_ADC1_CLK_ENABLE();
27    __HAL_RCC_DMA2_CLK_ENABLE();
28
29    //Configure peripheral GPIO
30    GPIO_InitStruct.Pin=GPIO_PIN_4;
31    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
32    GPIO_InitStruct.Pull = GPIO_NOPULL;
33    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
34
35    //Configure the DMA streams
36    hdma_adc.Init.Channel= DMA_CHANNEL_2;
37    hdma_adc.Init.Direction=DMA_PERIPH_TO_MEMORY;
38    hdma_adc.Init.PeriphInc=DMA_PINC_DISABLE;
39    hdma_adc.Init.MemInc=DMA_MINC_ENABLE;
40    hdma_adc.Init.PeriphDataAlignment=DMA_PDATAALIGN_WORD;
41    hdma_adc.Init.MemDataAlignment=DMA_MDATAALIGN_WORD;
42    hdma_adc.Init.Mode=DMA_CIRCULAR;
43    hdma_adc.Init.Priority=DMA_PRIORITY_HIGH;
44    hdma_adc.Init.FIFOMode=DMA_FIFOMODE_DISABLE;
45    hdma_adc.Init.FIFOThreshold=DMA_FIFO_THRESHOLD_HALFFULL;
46    hdma_adc.Init.MemBurst=DMA_MBURST_SINGLE;
47    hdma_adc.Init.PeriphBurst=DMA_PBURST_SINGLE;
48
49    HAL_DMA_Init(&hdma_adc);
50
51
52
53
54    ADC_ChannelConfTypeDef sConfig;
55
56    //Configure the ADC peripheral
57
58    AdcHandle.Init.ClockPrescaler=ADC_CLOCKPRESCALER_PCLK_DIV2;
59    AdcHandle.Init.Resolution=ADC_RESOLUTION_12B;
60    AdcHandle.Init.ScanConvMode=DISABLE;
61    AdcHandle.Init.DiscontinuousConvMode=DISABLE;
62    AdcHandle.Init.ContinuousConvMode=ENABLE;
63    AdcHandle.Init.NbrOfDiscConversion=0;
64    AdcHandle.Init.ExternalTrigConvEdge=ADC_EXTERNALTRIGCONVEDGE_NONE;
65    AdcHandle.Init.ExternalTrigConv=ADC_EXTERNALTRIGCONV_T1_CC1;
66    AdcHandle.Init.DataAlign=ADC_DATAALIGN_RIGHT;
67    AdcHandle.Init.NbrOfConversion=1;
68    AdcHandle.Init.DMAContinuousRequests=ENABLE;
69    AdcHandle.Init.EOCSelection=DISABLE;
70
71
```

```
72      if(HAL_ADC_Init(&AdcHandle) != HAL_OK)
73      {
74        Error_Handler();
75      }
77
78      // Configure ADC regular Channel
79
80      sConfig.Channel=GPIO_PIN_4;
81      sConfig.Rank=1;
82      sConfig.SamplingTime=ADC_SAMPLETIME_3CYCLES;
83      sConfig.Offset=0;
84
85      if(HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
86      {
87        Error_Handler();
88      }
89
90
91      //Associate the initialized DMA handle to the ADC handle
92
93    HAL_LINKDMA(&AdcHandle,DMA_Handle,hdma_adc); 94
95
96    //Configure the NVIC for DMA
97
98      HAL_NVIC_SetPriority(DMA2_Stream0_IRQn,0,0);
99      HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
100
101
102
103   }
104
105     void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle){
106     ADCConvertedValue=HAL_ADC_GetValue(AdcHandle);
107     }
```

11.2 Código modulo control por voz

```
/**
  ******************************************************************************
*@file pruebaPWMBoton.uvprojx
@brief prueba activación de hasta 4 motores con control PWM de velocidad,) se utilizan los timers 4
   (los 4 canales) y el timer3 el canal 4 ) a traves de 5 entradas controlados por una tarjeta de voz
   @date junio 2016
   @author Sara Marqués

   @note Parte del TFG
  *
  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */


#define PARAMOTORES 0
#define ABRIRPINZA  1
#define CERRARPINZA 2
#define PARARPINZA  3

#define HOMBROARRIBA  4
#define HOMBROABAJO 5
#define PARARHOMBRO 6

#define CODOARRIBA  7
#define CODOABAJO 8
#define PARARCODO 9

#define MUNECAARRIBA  10
#define MUNECAABAJO 11
#define PARARMUNECA 12

#define BASEDERECHA 13
#define BASEIZQUIERDA 14
#define PARARBASE 15

#define PINZAPIN1 GPIO_PIN_0
#define PINZAPIN2 GPIO_PIN_1
#define PINZAPWM  GPIO_PIN_9

#define HOMBROPIN1  GPIO_PIN_5
#define HOMBROPIN2  GPIO_PIN_6
#define HOMBROPWM GPIO_PIN_15

#define CODOPIN1  GPIO_PIN_7
#define CODOPIN2  GPIO_PIN_8
#define CODOPWM GPIO_PIN_14

#define MUNECAPIN1  GPIO_PIN_11
#define MUNECAPIN2  GPIO_PIN_12
#define MUNECAPWM GPIO_PIN_13

#define BASEPIN1  GPIO_PIN_9
#define BASEPIN2  GPIO_PIN_10

#define BASEPWM GPIO_PIN_12

#define PUERTOPINES GPIOB

#define VELOCIDADALTA1 100
#define VELOCIDADBAJA1  50

#define VELOCIDADALTA2 140
#define VELOCIDADBAJA2  90


```

```
71  int leerVelocidad(void);
72  int leerBoton(void);
73  int leerEstado(void);
74
75  void moverPinza(int ,int );
76  void moverHombro(int ,int );
77  void moverCodo(int ,int );
78  void moverMuneca(int ,int );
79  void moverBase(int ,int );
80  void paraMotores(void );
81
82  /* USER CODE END Includes */
83
84  /* Private variables ---------------------------------------------------------*/
85  TIM_HandleTypeDef htim4;
86  TIM_HandleTypeDef htim3;
87
88  /* USER CODE BEGIN PV */
89  /* Private variables ---------------------------------------------------------*/
90  int pwm=0;  //variable de control de PWM
91
92  /* USER CODE END PV */
93
94  /* Private function prototypes -----------------------------------------------*/
95  void SystemClock_Config(void);
96  void Error_Handler(void);
97  static void MX_GPIO_Init(void);
98  static void MX_TIM4_Init(void);
99  static void MX_TIM3_Init(void);
100
101 void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
102
103
104 /* USER CODE BEGIN PFP */
105 /* Private function prototypes -----------------------------------------------*/
106
107 /* USER CODE END PFP */
108
109 /* USER CODE BEGIN 0 */
110
111 /* USER CODE END 0 */
112
113 int main(void)
114 {
115
116   /* USER CODE BEGIN 1 */
117   // variables utilizadas en el control de los motores,
118     int estado,estadoAnterior=0;
119     int velocidad,velocidadAnterior=-1;
120
121   /* USER CODE END 1 */
122
123   /* MCU Configuration--------------------------------------------------------*/
124
125   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
126   HAL_Init();
127
128   /* Configure the system clock */
129   SystemClock_Config();
130
131   /* Initialize all configured peripherals */
132   MX_GPIO_Init();
133   MX_TIM4_Init();
134   MX_TIM3_Init();
135
136
137   //activamos el pin de alimentacion del chip de los motores
138
139   HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);
140
141   /* USER CODE BEGIN 2 */
142   // Empezamos a general señales de PWM en el timer4, los 4 canales y 1 del timer 3 (solo el canal4
143 HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4);   // ACTIVAMOS  EL PWM
144 HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_3);   // ACTIVAMOS  EL PWM
```

```
145   HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_2);   // ACTIVAMOS  EL PWM
146   HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_1);   // ACTIVAMOS  EL PWM
147   HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_4);   // ACTIVAMOS  EL PWM
148
149       printf("empezamos\n");
151       while (leerBoton()!=1){  // hasta que no apretamos el boton no empezamos por seguridad
153       }
155         HAL_Delay(1000);  // paramos 1 segundo para que el boton vuelva bien a 0
156
157     /* USER CODE END 2 */
158
159     /* Infinite loop */
160     /* USER CODE BEGIN WHILE */
161     while (1)
162     {
163     /* USER CODE END WHILE */
164
165
166       //seguridad:; leemos el estado del boton, si está pulsado paramos todos los motores y no seguimos
167       //sería mejor una implementación por interrupcion externa.
168
169       if(leerBoton()==1){
170         paraMotores();
171         while(1){}      // aqui nos quedamos
172         }
173
174       // leemos el estado y la velocidad que nos está comunicando la tarjeta de voz
175
176       estado=leerEstado();
177       velocidad=leerVelocidad();
178
179         //miramos si ha cambiado o sigue todo igual, si no ha cambiado nada, no hacemos nada y
    volvemos a leer
180
181         if (estado!=estadoAnterior || velocidad!=velocidadAnterior){
182
183           // ha cambiado estado o ha cambiado la velocidad
184
185           // almacenamos estado anterior
186
187           estadoAnterior=estado;
188           velocidadAnterior=velocidad;
189
190         paraMotores();      // paramos el motor que teníamos en marcha (paramos todos)
191
192           // ejecutamos la orden
193
194         switch(estado){
195
196           case 1:
197           case 2: moverPinza(estado,255);  //arrancamos
198                   HAL_Delay(50);
199                   if(velocidad==1)          // pasamos a velocidad adecuada
200                       moverPinza(estado,VELOCIDADALTA1+10);  //ajustamos velocidad
201                   else
202                       moverPinza(estado,VELOCIDADBAJA1+10);
203                   break;
204
205           case 3:   //arriba
206                   moverCodo(estado,255);   //arrancamos
207                   HAL_Delay(50);
208                   if(velocidad==1)
209                       moverCodo(estado,VELOCIDADALTA2);   //ajustamos velocidad
210                   else
211                       moverCodo(estado,VELOCIDADBAJA2);
212
213                   break;
214           case 4:   //abajo
215                  moverCodo(estado,255);   //arrancamos
216                   HAL_Delay(50);
217
218               if(velocidad==1)                      //ajustamos velocidad
219                       moverCodo(estado,VELOCIDADALTA1);
220                   else
221                       moverCodo(estado,VELOCIDADBAJA1);
222                   break;
223
```

```
224           case 5:  //avanza
225                   moverHombro(estado,255);    //arrancamos
226                   HAL_Delay(50);
227                   if(velocidad==1)
228                       moverHombro(estado,VELOCIDADALTA1);   //ajustamos velocidad
229                   else
230                       moverHombro(estado,VELOCIDADBAJA1);
231
232                   break;
233           case 6: // retrocede
234                   moverHombro(estado,255);               //arrancamos
235                   HAL_Delay(100);
236
237               if(velocidad==1)                    //ajustamos velocidad
238                       moverHombro(estado,VELOCIDADALTA2+20);
239                   else
240                       moverHombro(estado,VELOCIDADBAJA2+20);
241                   break;
242   //          case 10:
243   //          case 11:
244   //          case 12: moverMuneca(estado,velocidad);
245   //                   break;
246
247           case 7:
248           case 8: moverBase(estado,255);              //arrancamos
249                   HAL_Delay(50);
250
251               if(velocidad==1)                     //ajustamos velocidad
252                       moverBase(estado,VELOCIDADALTA2);
253                   else
254                       moverBase(estado,VELOCIDADBAJA2);
255                   break;
256
257           case 15: paraMotores();                     //paramos motores
258
259           }  // fin del switch:
260
261         }  // fin de si estado!=estadoAnterior o velocidad!=velocidadAnterior
262
263     /* USER CODE BEGIN 3 */
264
265     }
266     /* USER CODE END 3 */
267
268   }
269
270   /******************moverPinza************
271     funcion: Abre, cierra y para la pinza
272     argumentos: sentido:1-> abre la pinza; 2-> cierra la pinza   3-> para el motor de la pinza
273                 velocidad: velocidda que hay que comunicar a los pines PWM
274      salida: nada
275
276   */
277   void moverPinza(int sentido,int velocidad)
278   {
279     // primero activamos el PWM asociado a la pinza canal2 del timer4  corresponde al pin: PD13
280      __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_2,velocidad);
281         HAL_Delay(10);
282     // ponemos el sentido
283      switch(sentido){
284        case 1: // abrimos pinza
285
286           HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN1,GPIO_PIN_RESET);
287           HAL_Delay(10);
288           HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN2,GPIO_PIN_SET);
289           HAL_Delay(10);
290             break;
291        case 2: //cerramos pinza
292          HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN1,GPIO_PIN_SET);
293           HAL_Delay(10);
294           HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN2,GPIO_PIN_RESET);
295           HAL_Delay(10);
296             break;
```

```
297
298       case 15: // paramos pinza
299         HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN1,GPIO_PIN_SET);
300         HAL_Delay(10);
301         HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN2,GPIO_PIN_SET);
302         HAL_Delay(10);
303            break;
304
305     }     // fin del switch
306
307
308   }  // fin de mover pinza
309
310   /******************moverHombro************
311     funcion: Sube, Baja y para el hombro
312     argumentos: sentido:1-> sube el hombro; 2-> baja el hombro   3-> para el motor del hombro
313                 velocidad: velocidda que hay que comunicar a los pines PWM
314      salida: nada
315
316   */
317   void moverHombro(int sentido,int velocidad)
318   {
319     // primero activamos el PWM asociado a la pinza canal4 del timer4  corresponde al pin: PD15
320
321   //  printf("velocidad=%d",velocidad);
322      __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,velocidad);
323         HAL_Delay(10);
324     // ponemos el sentido
325      switch(sentido){
326        case 5: // SUBIMOS HOMBRO
327
328           HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN1,GPIO_PIN_RESET);
329           HAL_Delay(10);
330           HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN2,GPIO_PIN_SET);
331           HAL_Delay(10);
332             break;
333        case 6: //BAJAMOS HOMBRO
334          HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN1,GPIO_PIN_SET);
335           HAL_Delay(10);
336           HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN2,GPIO_PIN_RESET);
337           HAL_Delay(10);
338             break;
339
340        case 15: // paramos hombro
341          HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN1,GPIO_PIN_RESET);
342          HAL_Delay(10);
343          HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN2,GPIO_PIN_RESET);
344          HAL_Delay(10);
345             break;
346
347     }     // fin del switch
348
349
350   }  // fin de mover hombro
351
352   /******************moverCodo************
353     funcion: Sube, Baja y para el codo
354     argumentos: sentido:1-> sube el codo; 2-> baja el codo   3-> para el motor del codo
355                 velocidad: velocidda que hay que comunicar a los pines PWM
356      salida: nada
357
358   */
359   void moverCodo(int sentido,int velocidad)
360   {
361     // primero activamos el PWM asociado a la pinza canal3 del timer4  corresponde al pin: PD14
362      __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,velocidad);
363         HAL_Delay(10);
364     // ponemos el sentido
365      switch(sentido){
366        case 3: // subimos codo
367
368           HAL_GPIO_WritePin(PUERTOPINES,CODOPIN1,GPIO_PIN_RESET);
369           HAL_Delay(10);
370           HAL_GPIO_WritePin(PUERTOPINES,CODOPIN2,GPIO_PIN_SET);
371           HAL_Delay(10);
```

```
372           break;
373       case 4: //bajamos codo
374         HAL_GPIO_WritePin(PUERTOPINES,CODOPIN1,GPIO_PIN_SET);
375          HAL_Delay(10);
376          HAL_GPIO_WritePin(PUERTOPINES,CODOPIN2,GPIO_PIN_RESET);
377          HAL_Delay(10);
378           break;
379
380       case 15: // paramos codo
381         HAL_GPIO_WritePin(PUERTOPINES,CODOPIN1,GPIO_PIN_SET);
382         HAL_Delay(10);
383         HAL_GPIO_WritePin(PUERTOPINES,CODOPIN2,GPIO_PIN_SET);
384         HAL_Delay(10);
385           break;
386
387     }     // fin del switch
388
389
390   }  // fin de mover codo
391
392   /******************moverMuneca************
393     funcion: Sube, Baja y para la muñeca
394     argumentos: sentido:1-> sube el muñeca; 2-> baja la muñeca   3-> para el motor de la muñeca
395                 velocidad: velocidda que hay que comunicar a los pines PWM
396      salida: nada
397
398   */
399   void moverMuneca(int sentido,int velocidad)
400   {
401     // primero activamos el PWM asociado a la pinza canal4 del timer3  corresponde al pin: PC9
402      __HAL_TIM_SetCompare(&htim3,TIM_CHANNEL_4,velocidad);
403         HAL_Delay(10);
404     // ponemos el sentido
405      switch(sentido){
406       case 1: // arriba muneca
407
408           HAL_GPIO_WritePin(PUERTOPINES,MUNECAPIN1,GPIO_PIN_RESET);
409           HAL_Delay(10);
410           HAL_GPIO_WritePin(PUERTOPINES,MUNECAPIN2,GPIO_PIN_SET);
411           HAL_Delay(10);
412             break;
413       case 2: //abajo muneca
414         HAL_GPIO_WritePin(PUERTOPINES,MUNECAPIN1,GPIO_PIN_SET);
415          HAL_Delay(10);
416          HAL_GPIO_WritePin(PUERTOPINES,MUNECAPIN2,GPIO_PIN_RESET);
417          HAL_Delay(10);
418            break;
419
420       case 15: // paramos muneca
421         HAL_GPIO_WritePin(PUERTOPINES,MUNECAPIN1,GPIO_PIN_SET);
422         HAL_Delay(10);
423         HAL_GPIO_WritePin(PUERTOPINES,MUNECAPIN2,GPIO_PIN_SET);
424         HAL_Delay(10);
425           break;
426
427     }     // fin del switch
428
429
430   }  // fin de mover muneca
431
432   /******************moverBase************
433     funcion: derecha, izquierda para la base
434     argumentos: sentido:1-> gira derecha base; 2-> gira izquierda la base   3-> para el motor de la base
435                 velocidad: velocidda que hay que comunicar a los pines PWM
436      salida: nada
437
438   */
439   void moverBase(int sentido,int velocidad)
440   {
441
442     // primero activamos el PWM asociado a la pinza canal3 del timer4  corresponde al pin: PD12
443      __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_1,velocidad);
444         HAL_Delay(10);
445     // ponemos el sentido
446      switch(sentido){
```

```
447       case 7: // abrimos base
448
449          HAL_GPIO_WritePin(PUERTOPINES,BASEPIN1,GPIO_PIN_RESET);
450          HAL_Delay(10);
451          HAL_GPIO_WritePin(PUERTOPINES,BASEPIN2,GPIO_PIN_SET);
452          HAL_Delay(10);
453            break;
454       case 8: //cerramos base
455         HAL_GPIO_WritePin(PUERTOPINES,BASEPIN1,GPIO_PIN_SET);
456          HAL_Delay(10);
457          HAL_GPIO_WritePin(PUERTOPINES,BASEPIN2,GPIO_PIN_RESET);
458          HAL_Delay(10);
459            break;
460
461       case 15: // paramos base
462         HAL_GPIO_WritePin(PUERTOPINES,BASEPIN1,GPIO_PIN_SET);
463         HAL_Delay(10);
464         HAL_GPIO_WritePin(PUERTOPINES,BASEPIN2,GPIO_PIN_SET);
465         HAL_Delay(10);
466            break;
467
468     }     // fin del switch
469
470
471  }  // fin de mover base
472
473
474  //el pin PE4 nos indica si la velocidad será lento (pin=0) o rápido(pin=1)
475
476  int leerVelocidad(void)
477  {
478    int estadoBotonPE4;
479    estadoBotonPE4=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_4);
480    HAL_Delay(10);
481
482    if (estadoBotonPE4==1)
483
484            return 1;
485    else
486            return 0;
487
488  }
489  // leerBoton
490  // Mira si el boton de usuario esta pulsado o no,
491  // devuelve el estado del boton de usuario ( 0--> no pulsado    1---> pulsado)
492
493  int leerBoton(void)
494  {
495    int estadoBotonPA0;
496    estadoBotonPA0=HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0);
497    HAL_Delay(10);
498
499    return estadoBotonPA0;
500
501  }
502
503  // **************leer estado
504  //pines del PE0 A PE3 CUATRO BITS NOS INDICA EL TIPO DE MOVIMIENTO
505  // no vuelve mientras no se haya leído 5 lecturas idénticas consecutivas
506  // devuelve en decimal el estado leído
507
508  int leerEstado(void)
509  {
510    int estadoBotonPE0;
511    int estadoBotonPE1;
512    int estadoBotonPE2;
513    int estadoBotonPE3;
514    int cont;
515
516    int resultado=-2, resultadoAnterior=-1;
517
518      HAL_Delay(50);
519
520
521
```

```
522
523       estadoBotonPE0=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_0);
524     HAL_Delay(10);
525       estadoBotonPE1=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_1);
526     HAL_Delay(10);
527       estadoBotonPE2=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_2);
528     HAL_Delay(10);
529       estadoBotonPE3=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_3);
530     HAL_Delay(10);
531
532
533     resultado=estadoBotonPE0+2*estadoBotonPE1+4*estadoBotonPE2+8*estadoBotonPE3;
534       resultadoAnterior=resultado;
535
536       cont=1;
537     while(cont<5){
538      estadoBotonPE0=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_0);
539     HAL_Delay(10);
540       estadoBotonPE1=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_1);
541     HAL_Delay(10);
542       estadoBotonPE2=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_2);
543     HAL_Delay(10);
544     estadoBotonPE3=HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_3);
545     HAL_Delay(10);
546       resultado=estadoBotonPE0+2*estadoBotonPE1+4*estadoBotonPE2+8*estadoBotonPE3; // pasamos a decimal
547       if(resultado==resultadoAnterior)
548                   cont++;
549       else{
550         resultadoAnterior=resultado;
551         cont=1;
552       }
553
554     }
555
556
557
558
559
560     return resultado;
561
562   }
563
564   /*************** paraMotores
565   funcion: para todos los motores poniendo a 0 los PWM y desactivando el driver
566   argumentos: nada
567   salida: nada
568
569   */
570   void paraMotores(void)
571   {
572
573     // moverPinza(15,0);
574       __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_2,0);
575         HAL_Delay(10);
576      HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN1,GPIO_PIN_SET);
577          HAL_Delay(10);
578          HAL_GPIO_WritePin(PUERTOPINES,PINZAPIN2,GPIO_PIN_SET);
579          HAL_Delay(10);
580     //moverHombro(15,0);
581       __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,0);
582         HAL_Delay(10);
583      HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN1,GPIO_PIN_SET);
584          HAL_Delay(10);
585          HAL_GPIO_WritePin(PUERTOPINES,HOMBROPIN2,GPIO_PIN_SET);
586          HAL_Delay(10);
587     //moverCodo(15,0);
588       __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_3,0);
589         HAL_Delay(10);
590      HAL_GPIO_WritePin(PUERTOPINES,CODOPIN1,GPIO_PIN_SET);
591          HAL_Delay(10);
592          HAL_GPIO_WritePin(PUERTOPINES,CODOPIN2,GPIO_PIN_SET);
593          HAL_Delay(10);
594     //moverMuneca(15,0);
595     //moverBase(15,0);
596       __HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_1,0);
```

```
597         HAL_Delay(10);
598       HAL_GPIO_WritePin(PUERTOPINES,BASEPIN1,GPIO_PIN_SET);
599           HAL_Delay(10);
600           HAL_GPIO_WritePin(PUERTOPINES,BASEPIN2,GPIO_PIN_SET);
601           HAL_Delay(10);
602 
603   }
604 
605 
606   /** System Clock Configuration
607   */
608   void SystemClock_Config(void)
609   {
610 
611     RCC_OscInitTypeDef RCC_OscInitStruct;
612     RCC_ClkInitTypeDef RCC_ClkInitStruct;
613 
614     HAL_RCC_PWR_CLK_ENABLE();
615 
616     HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
617 
618     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
619     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
620     RCC_OscInitStruct.HSICalibrationValue = 16;
621     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
622     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
623     {
624       Error_Handler();
625     }
626 
627     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
628                                   |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
629     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
630     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
631     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
632     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
633     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
634     {
635       Error_Handler();
636     }
637 
638     HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
639 
640     HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
641 
642     /* SysTick_IRQn interrupt configuration */
643     HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
644   }
645 
646   /* TIM4 init function */
647   static void MX_TIM4_Init(void)
648   {
649 
650     TIM_MasterConfigTypeDef sMasterConfig;
651     TIM_OC_InitTypeDef sConfigOC;
652 
653     htim4.Instance = TIM4;
654     htim4.Init.Prescaler = 24;
655     htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
656     htim4.Init.Period = 255;
657     htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
658     if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
659     {
660       Error_Handler();
661     }
662 
663     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
664     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
665     if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
666     {
667       Error_Handler();
668     }
669 
670     sConfigOC.OCMode = TIM_OCMODE_PWM1;
671     sConfigOC.Pulse = 0;
```

```
672     sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
673     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
674     if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) != HAL_OK) // CONEC. AL PD15
675     {
676       Error_Handler();
677     }
678     if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_3) != HAL_OK) // CONEC. AL PD14
679     {
680       Error_Handler();
681     }
682     if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_2) != HAL_OK) // CONEC. AL PD13
683     {
684       Error_Handler();
685     }
686     if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK) // CONEC. AL PD12
687     {
688       Error_Handler();
689     }
690
691     HAL_TIM_MspPostInit(&htim4);
692
693   }
694
695   /* TIM3 init function */
696   static void MX_TIM3_Init(void)
697   {
698
699     TIM_MasterConfigTypeDef sMasterConfig;
700     TIM_OC_InitTypeDef sConfigOC;
701
702     htim3.Instance = TIM3;
703     htim3.Init.Prescaler = 24;
704     htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
705     htim3.Init.Period = 255;
706     htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
707     if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
708     {
709       Error_Handler();
710     }
711
712     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
713     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
714     if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
715     {
716       Error_Handler();
717     }
718
719     sConfigOC.OCMode = TIM_OCMODE_PWM1;
720     sConfigOC.Pulse = 0;
721     sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
722     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
723     if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_4) != HAL_OK) // conect. al PC9
724     {
725       Error_Handler();
726     }
727
728     HAL_TIM_MspPostInit(&htim3);
729
730   }
731
732   /** Configure pins as
733           * Analog
734           * Input
735           * Output
736           * EVENT_OUT
737           * EXTI
738   */
739   static void MX_GPIO_Init(void)
740   {
741
742     GPIO_InitTypeDef GPIO_InitStruct;
743
744     /* GPIO Ports Clock Enable */
745     HAL_RCC_GPIOA_CLK_ENABLE();
746     HAL_RCC_GPIOD_CLK_ENABLE();
```

```
747     HAL_RCC_GPIOB_CLK_ENABLE();
748     HAL_RCC_GPIOC_CLK_ENABLE();
749     HAL_RCC_GPIOE_CLK_ENABLE();
750
751     /*Configure GPIO pin : PA0  El boton del usuario  PINES DE ENTRADA DESDE LA TARJETA DE VOZ*/
752       // PIN DE SEGURIDAD SI SE PULSA EL BOTON PARAN TODOS LOS MOTORES.
753
754     GPIO_InitStruct.Pin = GPIO_PIN_0 ;
755     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
756     GPIO_InitStruct.Pull = GPIO_NOPULL;
757     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
758
759     /*Configure GPIO pin : PE0  a  PE4 PINES DE ENTRADA DESDE LA TARJETA DE VOZ*/
760       // TOMAMOS 5 ENTRADAS ( del PE0 AL PE4) HASTA 32 COMANDOS AQUI CONECTRIAMOS LAS SALIDAS DEL
    TARJ. VOZ
761     GPIO_InitStruct.Pin = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 |GPIO_PIN_4;
762     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
763     GPIO_InitStruct.Pull = GPIO_NOPULL;
764     HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
765
766     /*Configure GPIO pin : PD15  LED AZUL PD14 LED ROJO PD13 LED NARANJA PD12 LED VERDE SALIDAS PWM
    TIMER 4*/
767     GPIO_InitStruct.Pin = GPIO_PIN_15 | GPIO_PIN_14 | GPIO_PIN_13 | GPIO_PIN_12;
768     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
769     GPIO_InitStruct.Pull = GPIO_NOPULL;
770     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
771     GPIO_InitStruct.Alternate = GPIO_AF2_TIM4;
772     HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
773
774     /*Configure GPIO pin : PC9 SALIDA PWM TIMER 3*/
775     GPIO_InitStruct.Pin = GPIO_PIN_9;
776     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
777     GPIO_InitStruct.Pull = GPIO_NOPULL;
778     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
779     GPIO_InitStruct.Alternate = GPIO_AF2_TIM3;
780     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
781
782     // Configurar GPIO pin: PB11 Y PB13  salidas digitales a los drivers de los motores
783
784     /*Configure GPIO pin : PB11 Y PB13  */
785     GPIO_InitStruct.Pin = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_5 | GPIO_PIN_6 |  GPIO_PIN_7 | GPIO_PIN_8
    |  GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_15 ;  // 16 salidas digitales del PB0 AL PB15
786     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
787     GPIO_InitStruct.Pull = GPIO_NOPULL;
788     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
789      HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
790   }
791
792   /* USER CODE BEGIN 4 */
793
794   /* USER CODE END 4 */
795
796   /**
797     * @brief  This function is executed in case of error occurrence.
798     * @param  None
799     * @retval None
800     */
801   void Error_Handler(void)
802   {
803     /* USER CODE BEGIN Error_Handler */
804     /* User can add his own implementation to report the HAL error return state */
805     while(1)
806     {
807     }
808     /* USER CODE END Error_Handler */
809   }
810
811   #ifdef USE_FULL_ASSERT
812
813   /**
814      * @brief Reports the name of the source file and the source line number
815     * where the assert_param error has occurred.
816     * @param file: pointer to the source file name
817     * @param line: assert_param error line source number
818     * @retval None
```

```
819     */
820   void assert_failed(uint8_t* file, uint32_t line)
821   {
822     /* USER CODE BEGIN 6 */
823     /* User can add his own implementation to report the file name and line number,
824       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
825     /* USER CODE END 6 */
826
827   }
828
829   #endif
830
831   /**
832     * @}
833     */
834
835   /**
836     * @}
837   */
838
839   /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
840
```

11.3 Presupuesto

PLACA DE CIRCUITO IMPRESO				
MATERIALES				
Uds.	Descripción	Cantidad	Precio	Total
u	Fabricación PCB 22x25 mm, doble cara	1	30.00 €	30.00 €
u	Convertidor analógico-digital de 16 bits	1	38.92€	38.92 €
u	Sensor acelerómetro de 3 ejes	1	4.99 €	4.99 €
u	Fuente de alimentación	1	8.43 €	8.43€
u	Condensador SMD 10 µF	1	1.350 €	1.350 €
u	Condensador SMD 100 nF	1	0.087 €	0.087 €
u	Condensador SMD 3.3 nF	1	0.177 €	0.177 €
u	Condensador SMD 1 µF	1	0.208 €	0.208 €
u	Condensador SMD 2 µF	1	0.147 €	0.147 €
u	Condensador SMD 47 µF	1	0.356 €	0.356 €
u	Resistencia SMD 10 kΩ	1	0.021 €	0.021 €
u	Conectores de 2 bornes	3	0.606 €	1.818 €
	Subtotal Materiales			85.44 €

MANO DE OBRA				
Uds.	Descripción	Cantidad	Precio	Total
h	Diseño de la placa	2	15.00 €	30.00 €
h	Fabricación PCB	1.5	12.00 €	18.00 €
h	Soldadura y montaje	1.5	12.00 €	18.00 €
	Subtotal Mano de obra			66.00 €

COSTE TOTAL DE FABRICACIÓN	151.44 €

CIRCUITO DE CONTROL POR VOZ

MATERIALES				
Uds.	**Descripción**	**Cantidad**	**Precio**	**Total**
u	Driver para motor de DC	2	3.37 €	6.74 €
u	Kit brazo robótico	1	47.28 €	47.28 €
u	Alimentación externa de 9 V (Pila 9V)	1	4.326 €	4.326 €
u	Diodo led Rojo	1	0.164 €	0.164 €
u	Diodo led Verde	4	0.096 €	0.384 €
u	Tarjeta STM32F407VG Discovery	1	20.78 €	20.78 €
u	Tarjeta de voz SpeakUp Click	1	38.74 €	38.74 €

Subtotal Materiales	118.42 €

MANO DE OBRA				
Uds.	Descripción	Cantidad	Precio	Total
h	Montaje del circuito	1.5	15.00 €	45.00 €

Subtotal Mano de obra	45.00 €

COSTE TOTAL DE FABRICACIÓN	163.42 €

RESUMEN DEL PRESUPUESTO DEL PROYECTO

Denominación	Materiales	Mano de obra	Total
Placa de circuito impreso	85.44 €	66.00 €	151.44 €
Circuito de control por voz	118.41 €	45.00 €	163.41 €
TOTAL	203.85 €	111.00 €	314.85 €

El total del proyecto asciende a 314.85 € (TRESCIENTOS CATORCE EUROS CON OCHENTE Y CINCO CENTIMOS)

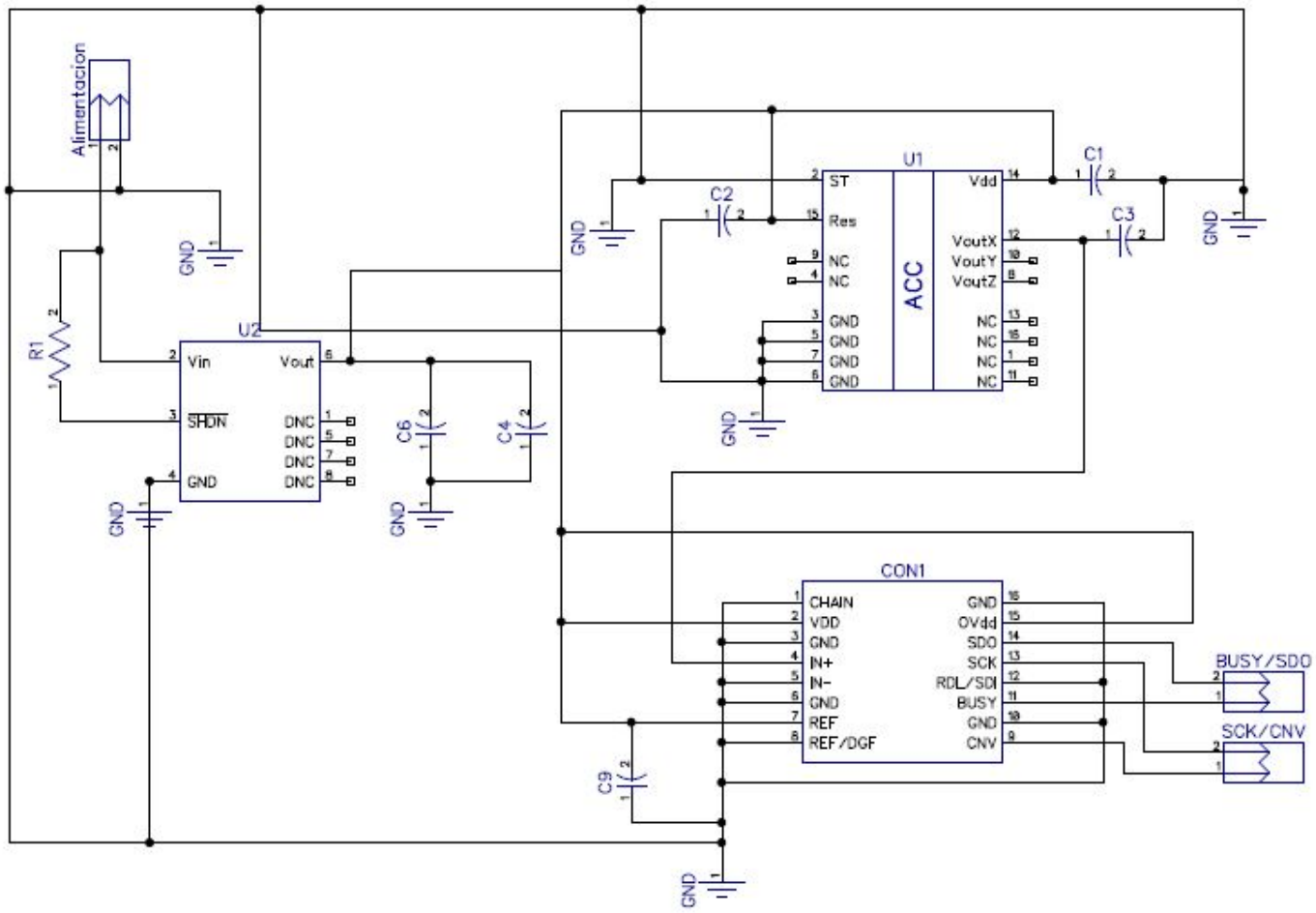
Los precios de mano de obra han sido extrapolados de proyectos anteriores.

Los precios de los materiales están obtenido del distribuidor: www.rs.es

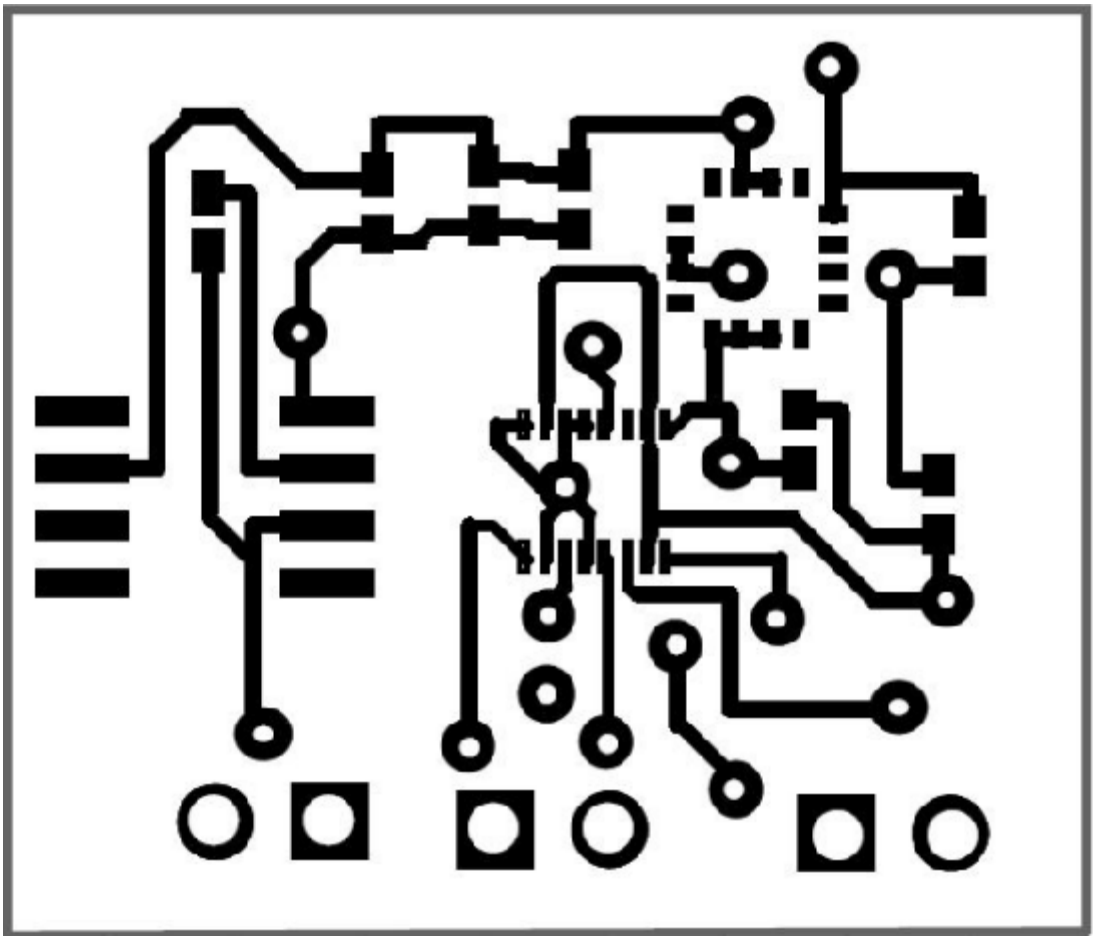
El precio de fabricación de la PCB corresponde con: http://www.pcbya.com/es/

El precio del Kit de brazo robótico está extraído de: https://www.amazon.es/

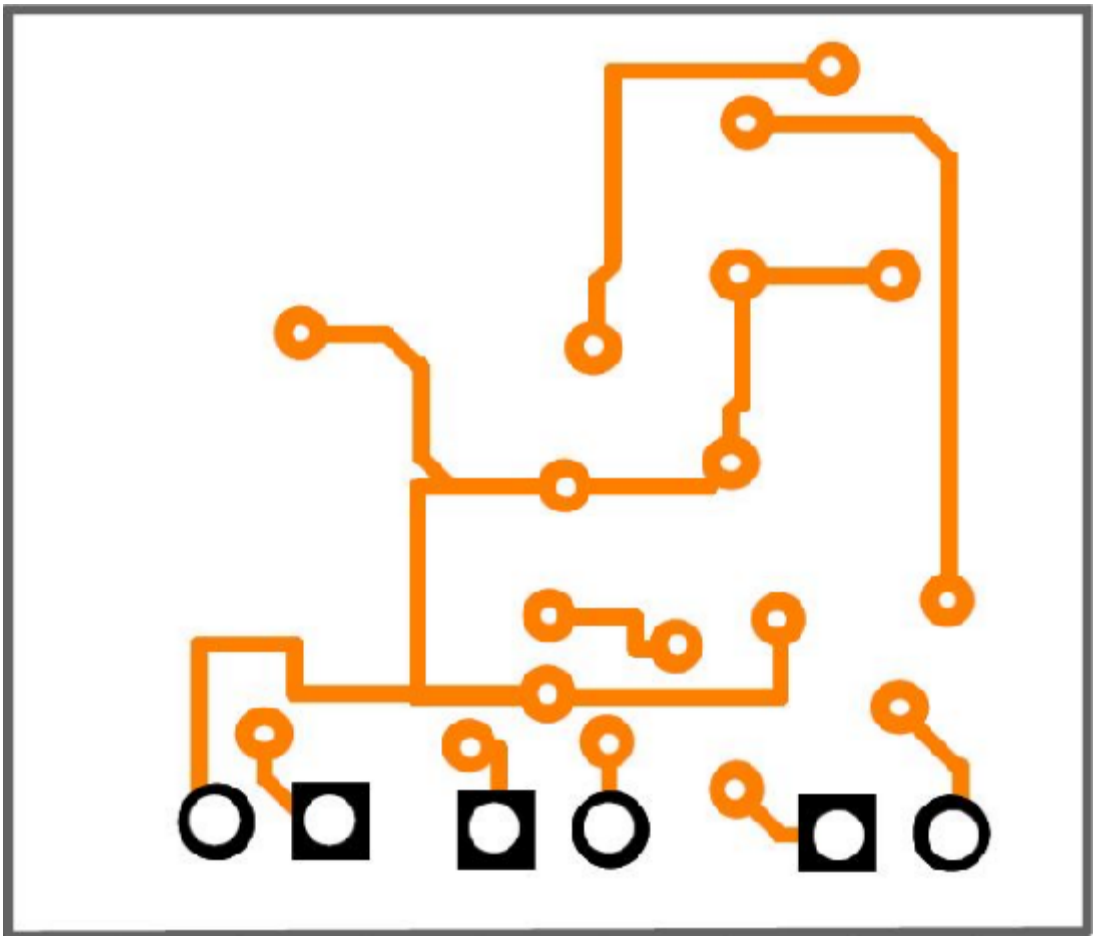
11.4 Planos



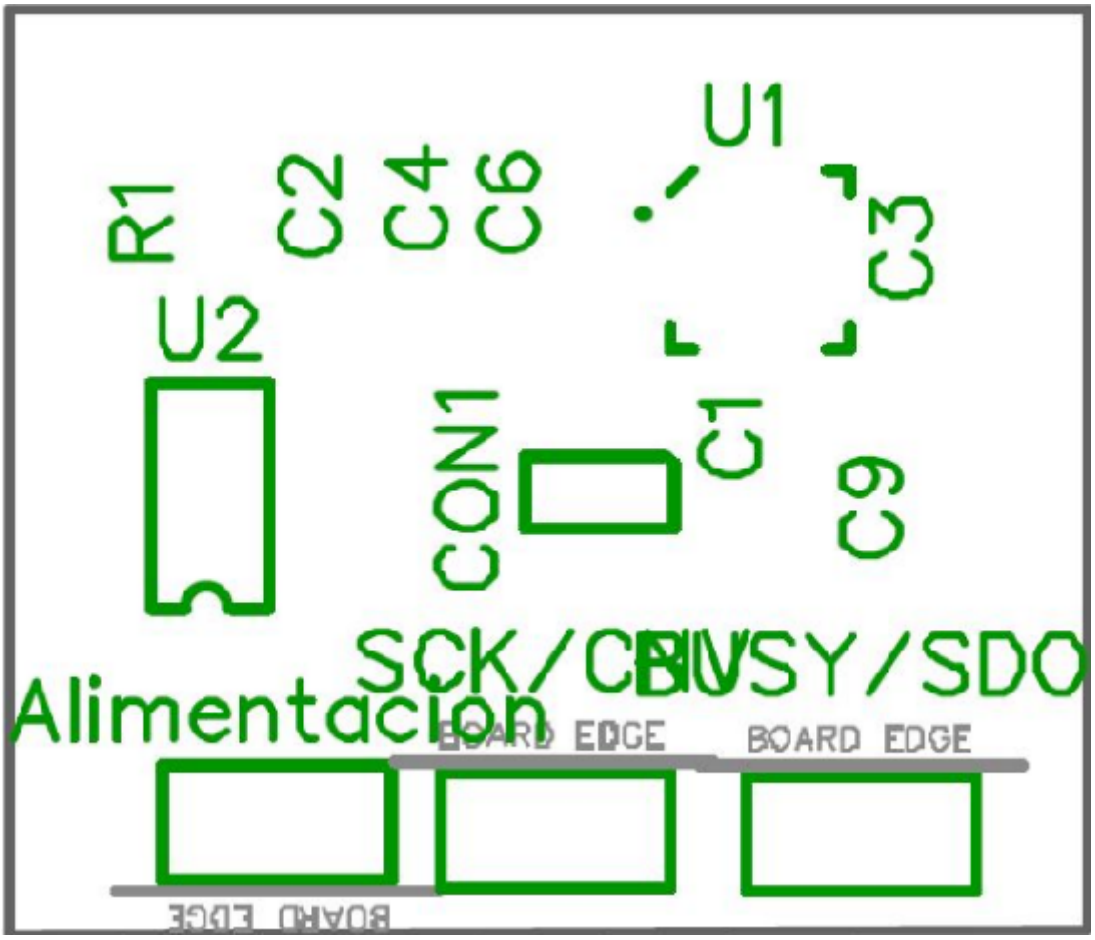
PROYECTO: DESARROLLO DE SOFTWARE DE BAJO NIVEL PARA UN BRAZO ROBOT PORTÁTIL	**Trabajo Fin de Grado**
PROYECTISTA Marqués Villarroya, Sara	TÍTULO: Plano de conexiones PCB
Escuela Técnica Superior de Ingeniería del Diseño UNIVERSITAT POLITÈCNICA DE VALÈNCIA	N.º DE DIBUJO 1
	ESCALA:1:1 — HOJA 1 DE 1



PROYECTO: DESARROLLO DE SOFTWARE DE BAJO NIVEL PARA UN BRAZO ROBOT PORTÁTIL		Trabajo Fin de Grado
PROYECTISTA Marqués Villarroya, Sara	UNIVERSITAT POLITÈCNICA DE VALÈNCIA	TÍTULO: Plano de pistas Pistas Principal Superior
Escuela Técnica Superior de Ingeniería del Diseño		N.º DE DIBUJO 2
		ESCALA:1:1 / HOJA 1 DE 1



PROYECTO: DESARROLLO DE SOFTWARE DE BAJO NIVEL PARA UN BRAZO ROBOT PORTÁTIL		Trabajo Fin de Grado
PROYECTISTA Marqués Villarroya, Sara	UNIVERSITAT POLITÈCNICA DE VALÈNCIA	TÍTULO: Plano de pistas Pistas Principal Inferior
Escuela Técnica Superior de Ingeniería del Diseño		N.º DE DIBUJO 3
		ESCALA:1:1 / HOJA 1 DE 1



Alimentacion, BUSY/SDO, SCK/CNV	-	644456-2
CON1	LTC2380-16	Convertidor ADC 16 bits
R1	10k	-
U2	LT1461ACS8	Fuente de alimentación
C9	47 uF	-
C6	2 uF	-
C4	1 uF	-
C3	3.3 nF	-
C2	100 nF	-
C1	10 uF	-
U1	LIS344AL	Acelerómetro
Referencia	**Valor**	**Observaciones**

PROYECTO: DESARROLLO DE SOFTWARE DE BAJO NIVEL PARA UN BRAZO ROBOT PORTÁTIL

Trabajo Fin de Grado

PROYECTISTA

Marqués Villarroya, Sara

Escuela Técnica Superior de Ingeniería del Diseño

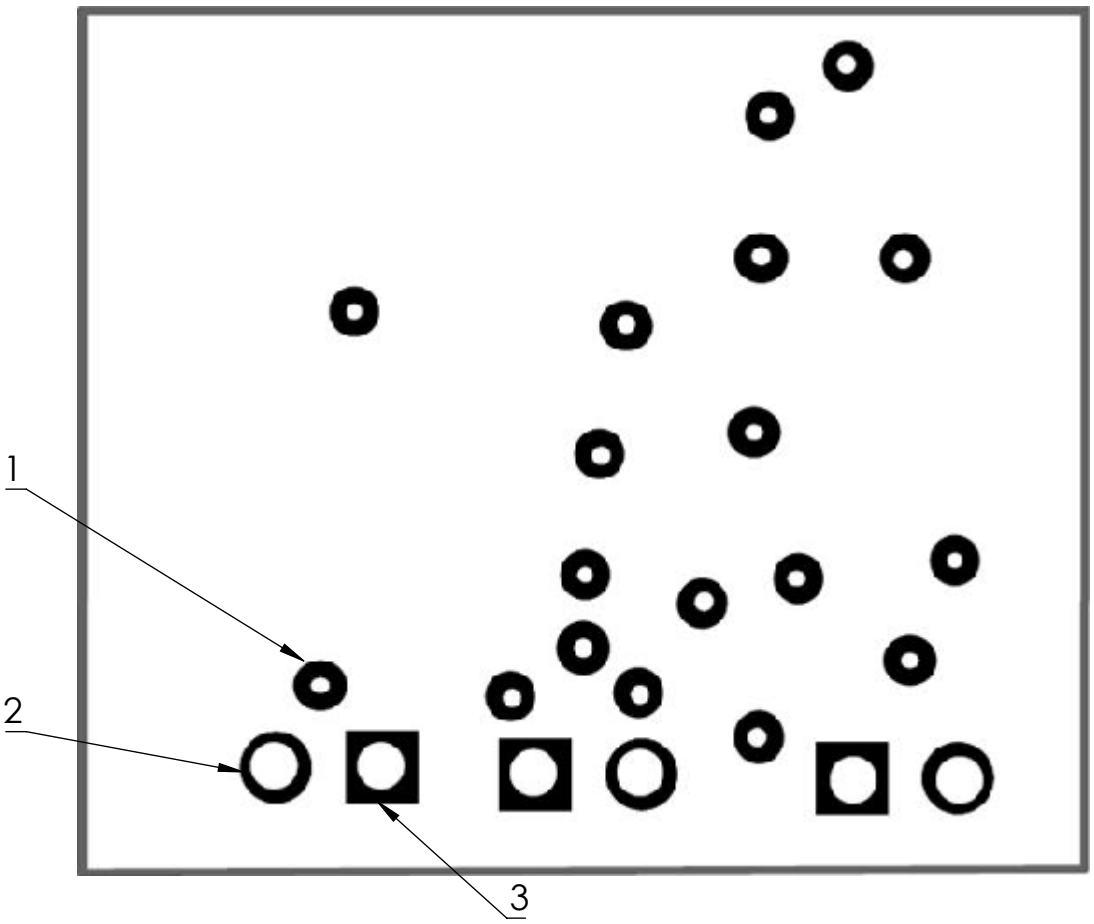
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TÍTULO: Plano de serigrafia de componentes PCB

N.º DE DIBUJO

4

ESCALA:1:1 | HOJA 1 DE 1



3	3	Cuadrada	2.286 mm	1.016 mm
2	3	Circular	2.286 mm	1.016 mm
1	18	Circular	1.2 mm	0.4 mm
Nº	Cantidad	Tipo	Dim. Exterior	Dim. Interior

PROYECTO: DESARROLLO DE SOFTWARE DE BAJO NIVEL PARA UN BRAZO ROBOT PORTÁTIL

Trabajo Fin de Grado

PROYECTISTA

Marqués Villarroya, Sara

Escuela Técnica Superior de Ingeniería del Diseño

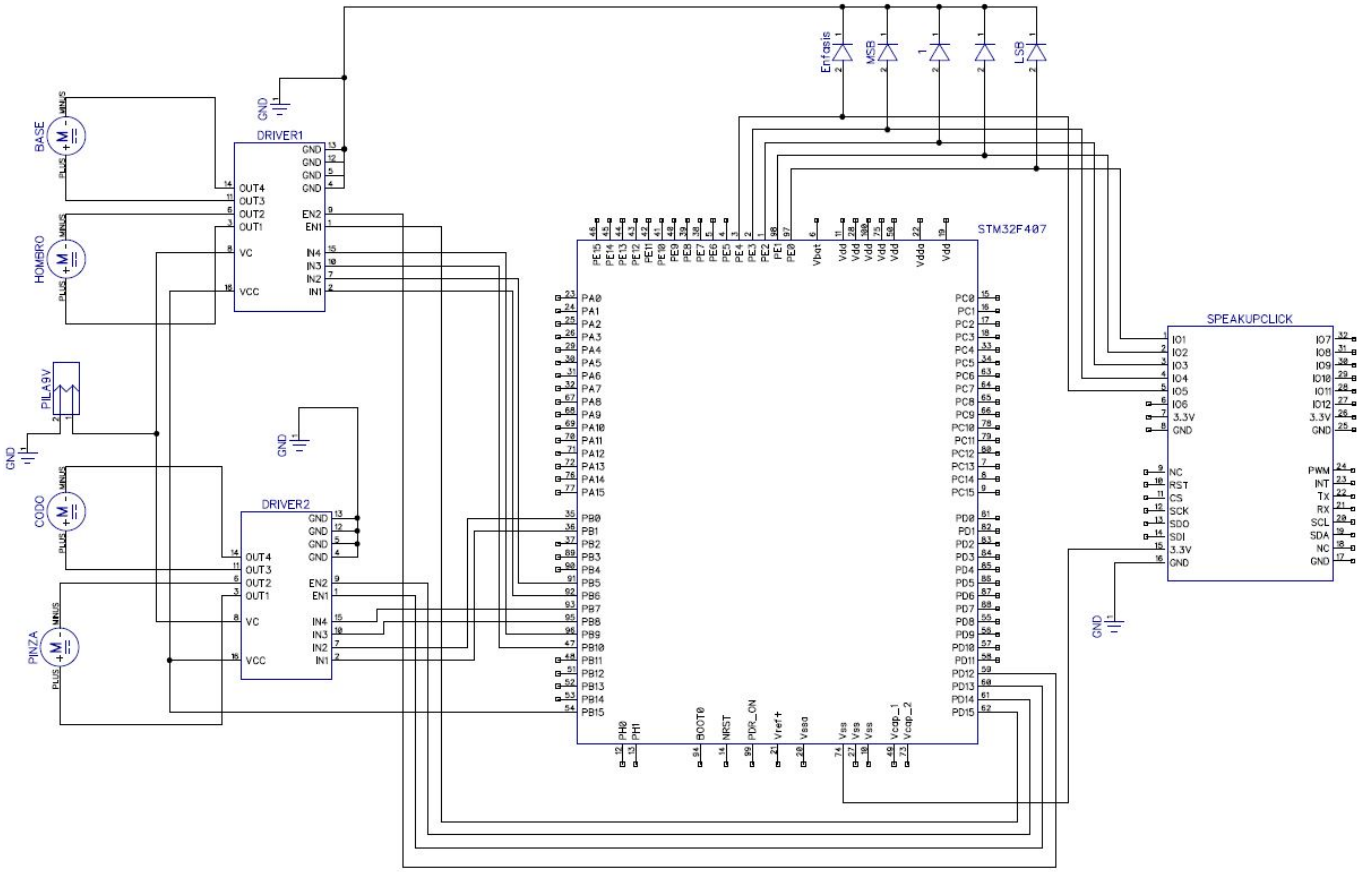
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TÍTULO: Plano de taladros

N.º DE DIBUJO

5

ESCALA:1:1 | HOJA 1 DE 1



PROYECTO: DESARROLLO DE SOFTWARE DE BAJO NIVEL PARA UN BRAZO ROBOT PORTÁTIL

Trabajo Fin de Grado

PROYECTISTA

Marqués Villarroya, Sara

Escuela Técnica Superior de Ingeniería del Diseño

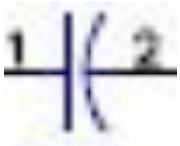
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TÍTULO: Plano de conexiones control por voz

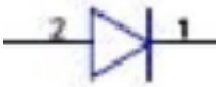
N.º DE DIBUJO

6

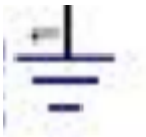
ESCALA:1:1 | HOJA 1 DE 1



Condensador



Diodo



Conexión a masa

Resistencia

PROYECTO: DESARROLLO DE SOFTWARE DE BAJO NIVEL PARA UN BRAZO ROBOT PORTÁTIL		**Trabajo Fin de Grado**
PROYECTISTA Marqués Villarroya, Sara	UNIVERSITAT POLITÈCNICA DE VALÈNCIA	TÍTULO: Plano de simbología electrónica
Escuela Técnica Superior de Ingeniería del Diseño		N.º DE DIBUJO 7
		ESCALA:1:1 / HOJA 1 DE 1