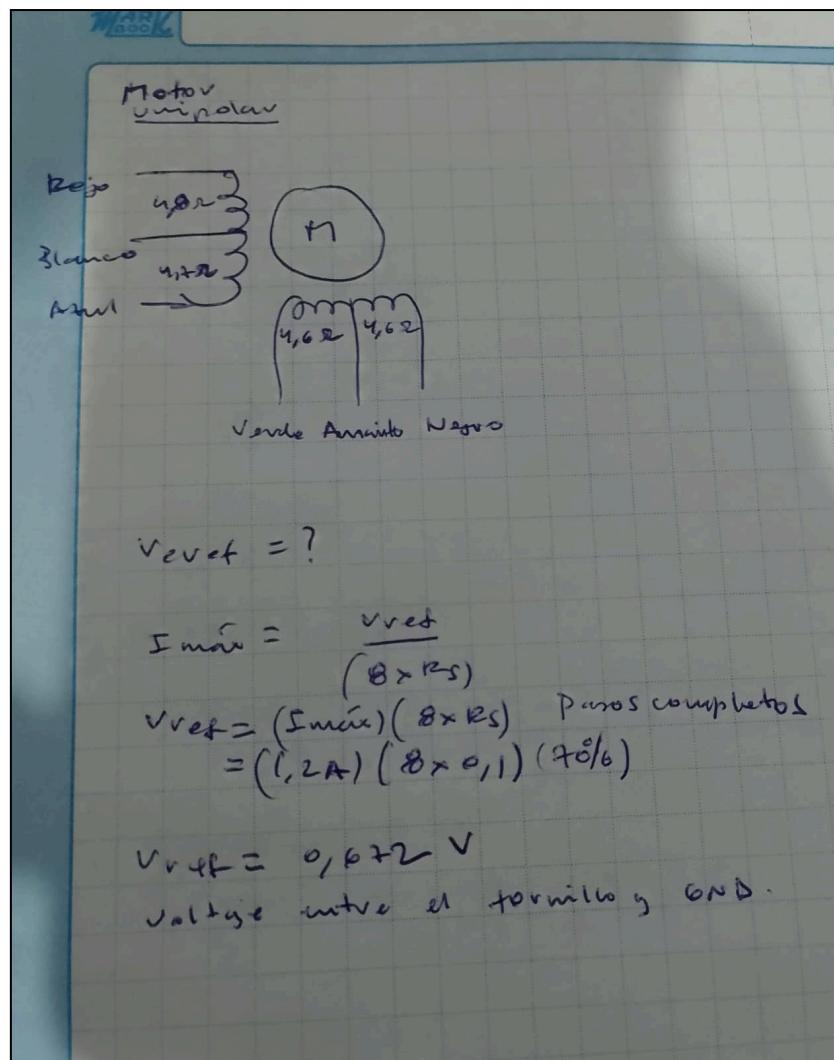


Identificación de pines para el motor paso a paso

Ref: <https://www.youtube.com/watch?v=bx5XrROhr-E>

Calibración del driver A4988

Ref: https://www.youtube.com/watch?v=zliZ_gSi77Y

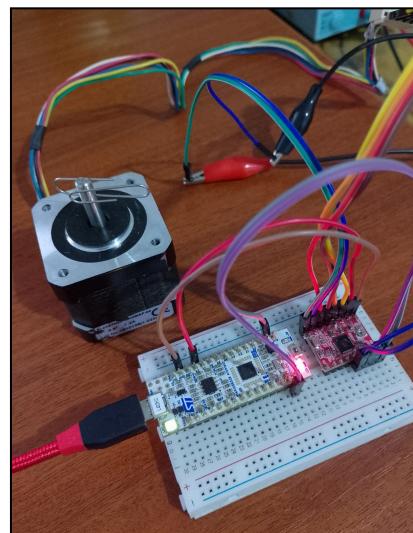
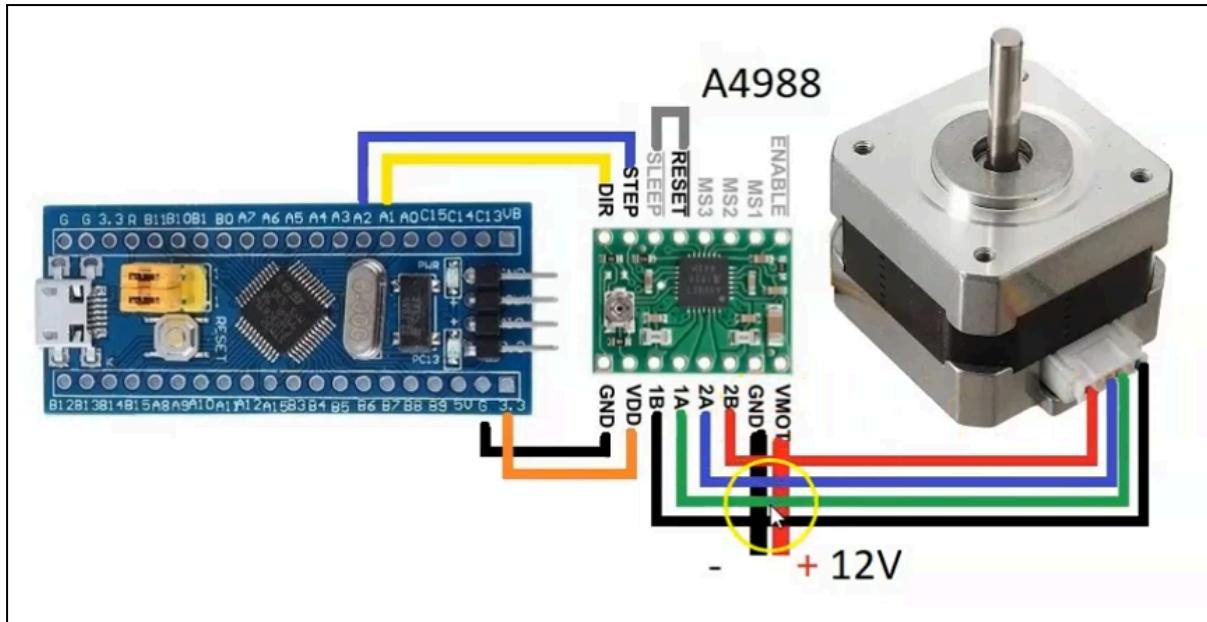


Referencia de motor: https://www.fullingmotor.com/en/product/66_169

El voltaje de referencia 0.672V es correcto para una alimentación del driver de 5V. Para 3.3V es diferente.

El motor es de 0.9°, es decir, necesita 400 pasos para dar una vuelta completa.

Los dos motores que tenemos funcionan de la misma forma. Son idénticos, también en el cableado.



Vref del primer driver (verde) = 0.679V

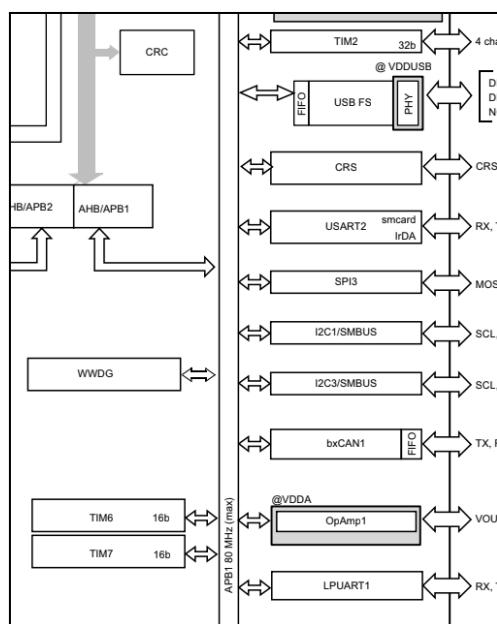
Vref del segundo driver (rojo) = 0.672V

Pines

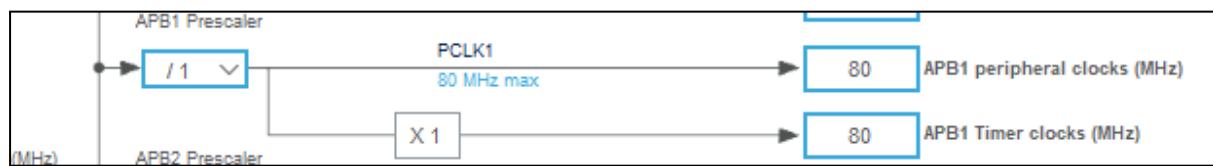
DIR1	PA9	D1
DIR2	PB1	D6
STEP1	PB3	D13
STEP2	PB5	D11

Timers en el NUCLEO

Usaremos el **TIMER 2** debido a que es de 32 bits y nos ayudará a manejar mejor la función implementada.



El TIMER 2 se encuentra dentro del bus APB1. Definimos la frecuencia del TIMER en 80 MHz (máxima).

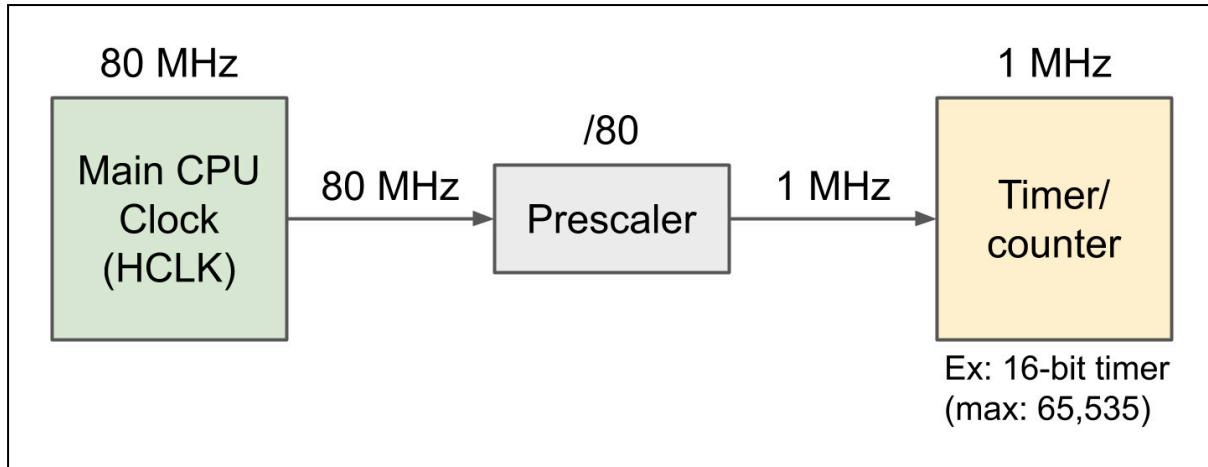


NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	<input checked="" type="checkbox"/>	0	0

Definimos el Prescaler en un valor de 80-1 lo que nos dará la generación de un tick cada 1 us debido a que $\text{tick} = (1/80 \text{ ns}) \times ((80-1)+1)$.

El Counter Period al ser de 32 bits se obtiene como valor máximo 2^{32} . Este debe ser mayor que el delay entregado por la función hecha para que funcione correctamente.

Básicamente el Counter Period nos indicará hasta cuánto podría contar el TIM2. Esto es importante debido a que según la función hecha el CP no llega a cierto valor el TIM delay en us no funcionará correctamente.



Conectaremos un LED en PB3 D13 para ver el funcionamiento del TIM2 como Delay en us.

Código

```

/* USER CODE BEGIN PFP */
void usDelay(uint16_t delay);

/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start(&htim2);

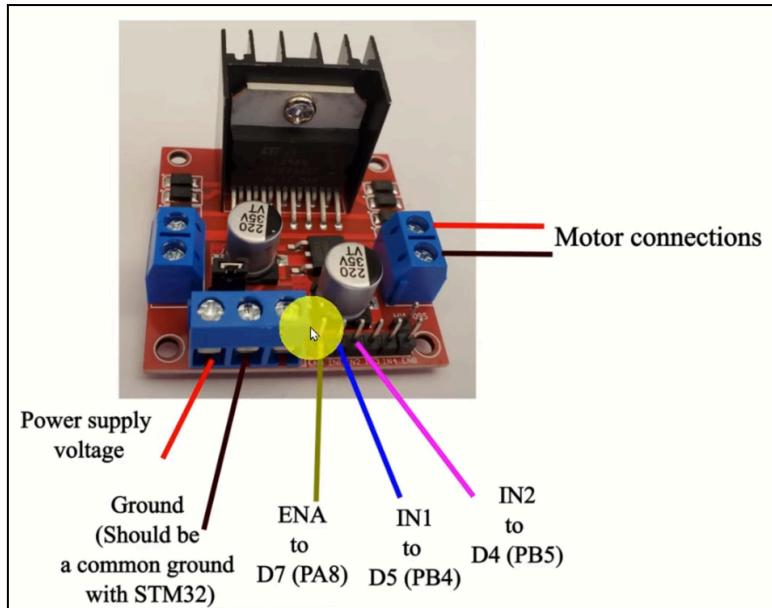
while (1)
{
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
    msDelay(1000000);
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
    msDelay(1000000);
}

/* USER CODE BEGIN 4 */
// Timer function
void usDelay(uint32_t delay)
{
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    while (__HAL_TIM_GET_COUNTER(&htim2) < delay);
}

```

Se llega a visualizar de las pruebas que el delay en el programa es de 1000000 equivalente a 1 segundo. Corroborado en el LED físicamente.

Trabajo con los motores reductores DC



$$f = 80 \text{ MHz}$$

TIM2 APB /
(Timer clocks)

$$f_{\text{PWM}} = 2000 \text{ Hz}$$

$$k_{PS} = 79$$

(prescaler)

$$k_p = ?$$

(period)

$$2000 = \frac{80 \times 10^6}{(79+1)(k_p+1)}$$

$$k_p + 1 = \frac{10^6}{2000}$$

$$k_p = 499$$

$\overbrace{\quad\quad\quad}$
ARR

$$CCR = 0, 1, 2, \dots, ARR + 1$$

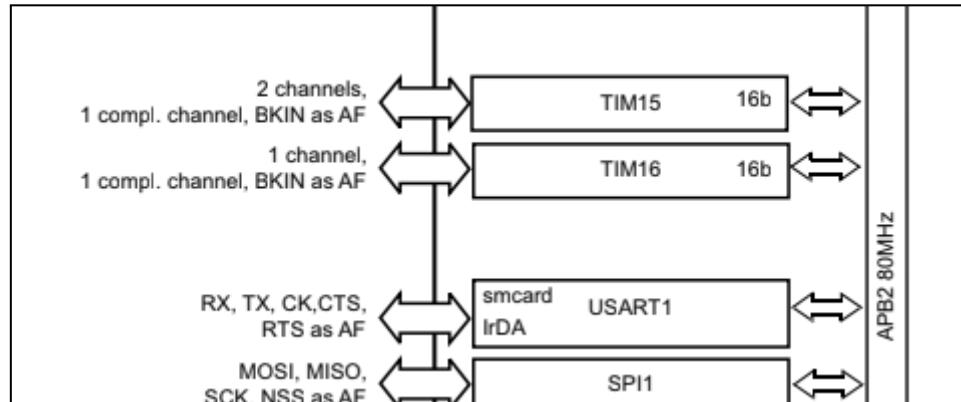
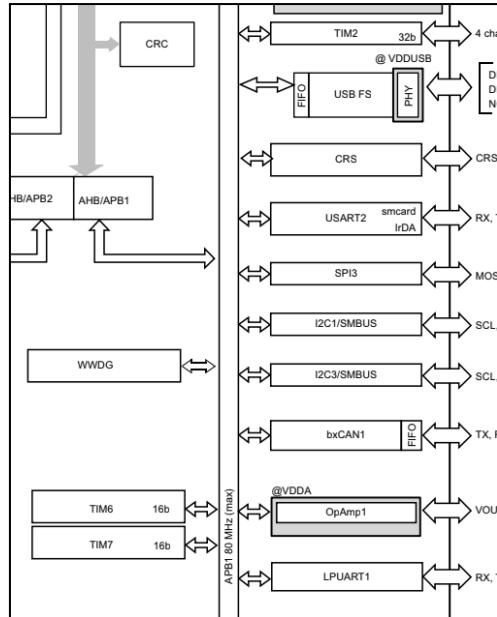
$$\gamma = \frac{CCR}{ARR}$$

Pines

IN1	PA10	D0
IN2	PB1	D6
IN3	PA11	D10
IN4	PB5	D11

PWM Timer

TIM2_CH1	PA0	A0
TIM15_CH1	PA2	A7



Consumo de corriente

V = 6V (para los dos motores)

PWM = 100%

100 ms de cambio → 800 mA (Puesto en la tabla Excel)

Continuo → 240 mA

Carga total → 990 mA ≈ 1A

V = 5V (para los dos motores)

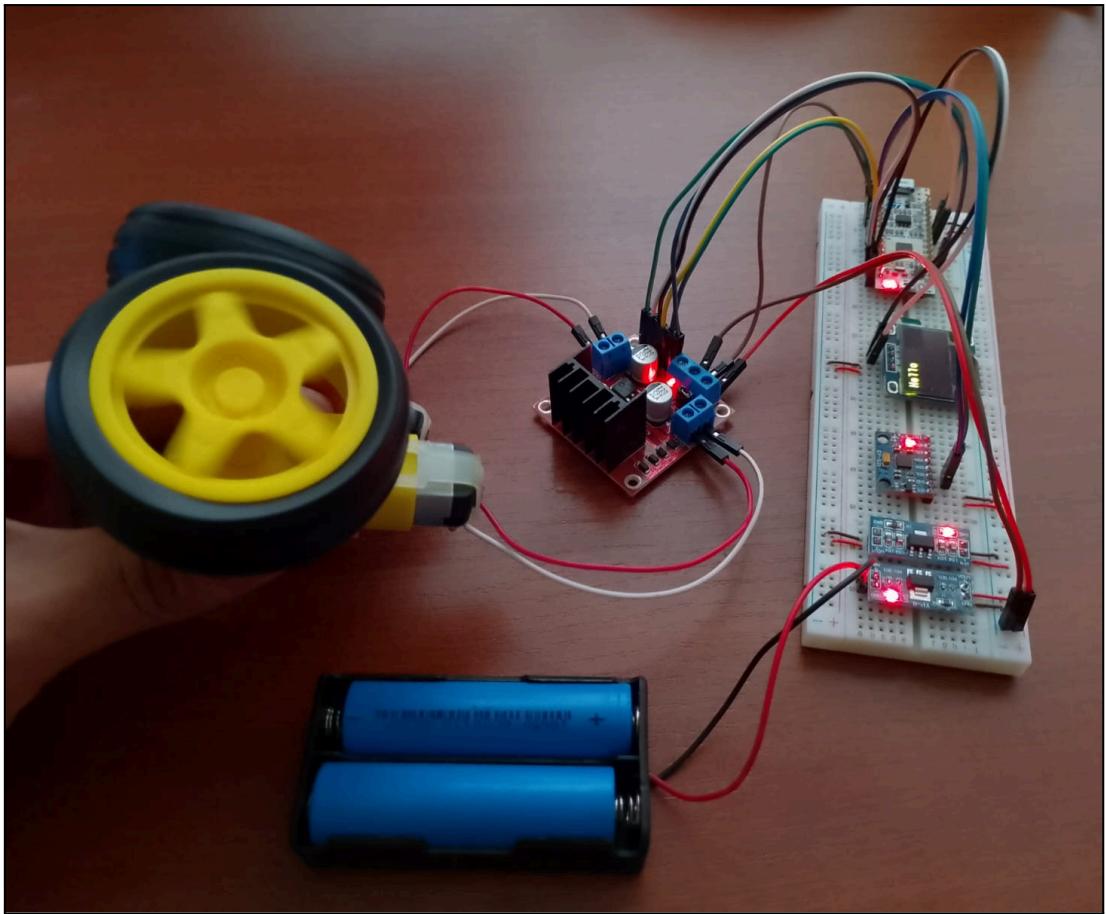
PWM = 100%

100 ms de cambio → 620 mA (Puesto en la tabla Excel)

Continuo → 235 mA

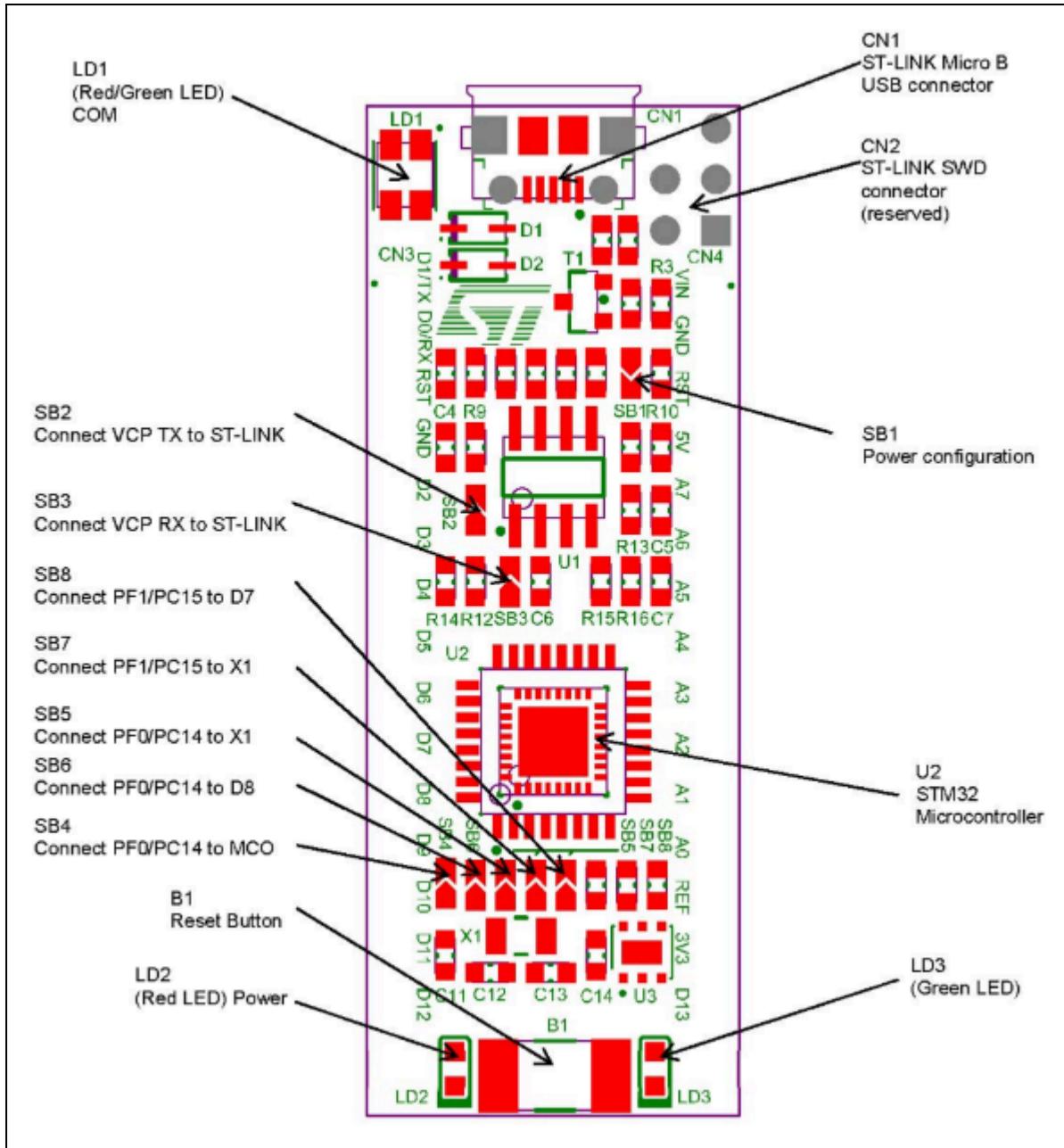
Carga total → 770 mA

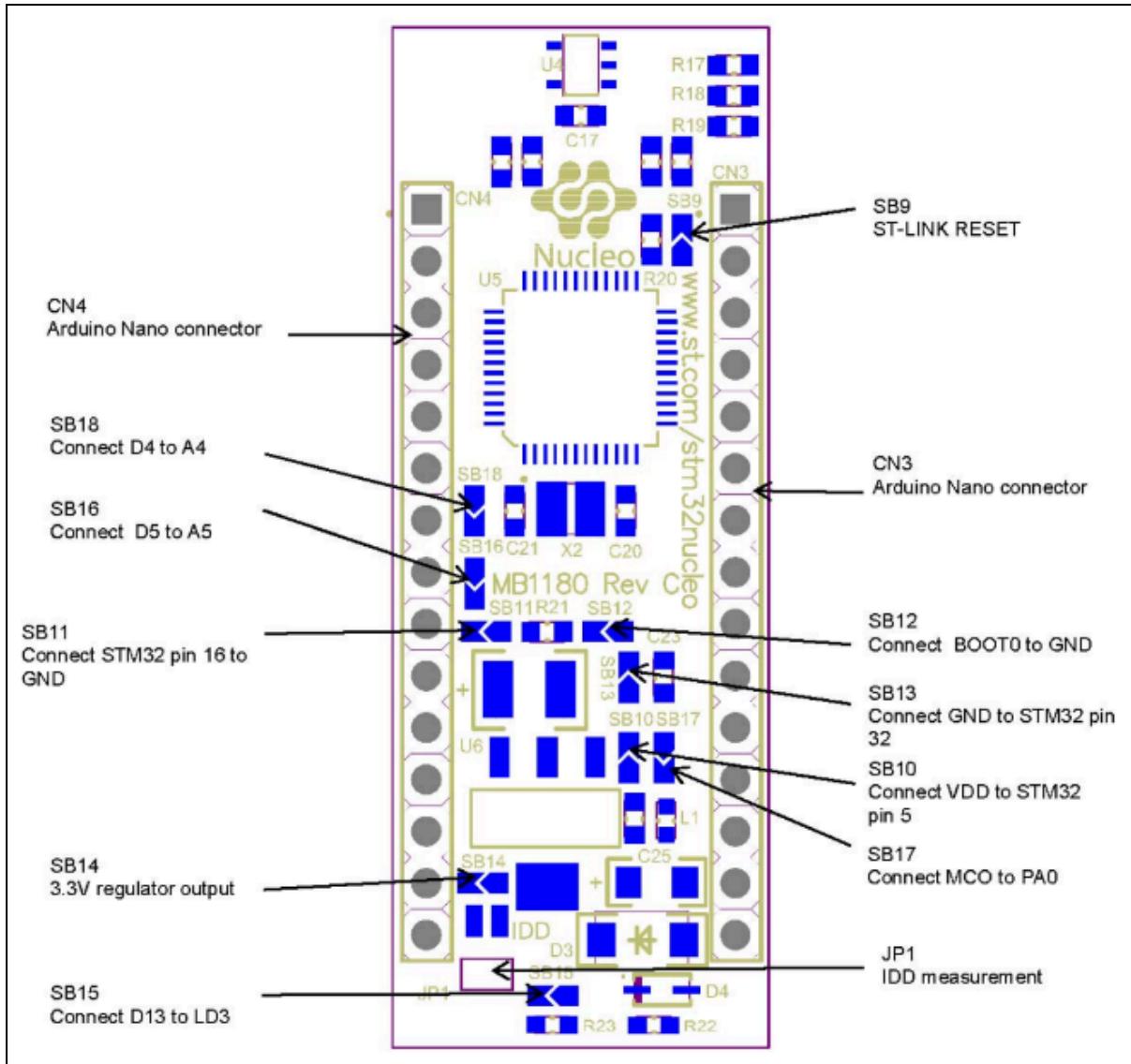
Resultado Final



Funcionamiento del NUCLEO por alimentacion externa

El SB1 debe estar OFF (sin soldadura).
El SB9 debe estar en OFF (sin soldadura).





La alimentación de +5V se hace a través del pin 5V y GND.

Para los próximos proyectos, primero cargamos el programa con el dispositivo conectado por USB y enlazado al componente que se desea utilizar; luego verificamos que el resultado sea el esperado, desconectamos el USB y finalmente alimentamos todo el sistema con una fuente de +5V.

Pruebas del I2C Scan

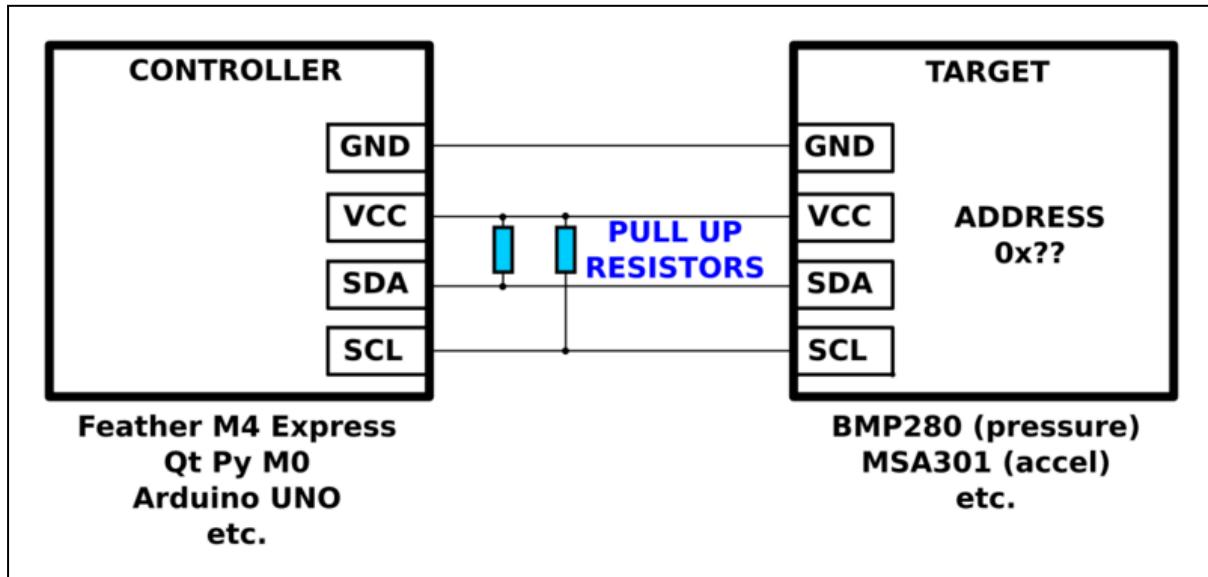
Pines del I2C1:

SDA: PB7

SCL: PB6

A5	PA0	12	PA6
A6	PA7	13	PA7
D9	PA8	18	PA8
D1	PA9	19	PA9
D0	PA10	20	PA10
D10	PA11	21	PA11
D2	PA12	22	

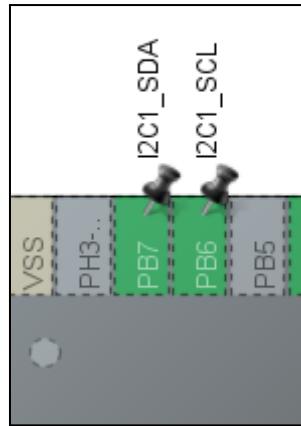
Recordar las resistencias Pull Up



En nuestra placa, la conexión SB16 y SB18 están en ON.

CN4	7	A5 ⁽²⁾	PA6	ADC12_IN11 I2C1_SCL
	8	A4 ⁽²⁾	PA5	ADC12_IN10 I2C1_SDA
SB16	ON	STM32 PB6 is connected to CN4 pin 7 for I ² C SCL support on ARDUINO® Nano A5. In such case, STM32 PB6 does not support ARDUINO® Nano D5 and PA6 must be configured as input floating.		
SB18	ON	STM32 PB7 is connected to CN4 pin 8 for I ² C SDA support on ARDUINO® Nano A4. In such case, STM32 PB7 does not support ARDUINO® Nano D4 and PA5 must be configured as input floating.		
	OFF	CN4 pin 8 is used as ARDUINO® Nano analog input A4 without I ² C support and CN3 pin 7 is available as ARDUINO® Nano D4.		

En el STM32 CubeID:



De acuerdo con la tabla, los pines del I2C1 serían:

SDA: A4 (PA5)

SCL: A5 (PA6)

NUCLEO-LxxxKx					
PA9	1	D1	VIN	1	VIN
PA10	2	D0	GND	2	GND
NRST	3	NRST	NRST	3	NRST
GND	4	GND	+5V	4	+5V
PA12	5	D2	A7	5	PA2
PB0	6	D3	A6	6	PA7
PB7	7	D4	A5	7	PA6
PB6	8	D5	A4	8	PA5
PB1	9	D6	A3	9	PA4
PC14	10	D7	A2	10	PA3
PC15	11	D8	A1	11	PA1
PA8	12	D9	A0	12	PA0
PA11	13	D10	AREF	13	AREF
PB5	14	D11	+3V3	14	+3V3
PB4	15	D12	D13	15	PB3
CN3			CN4		
 ARDUINO®					

Scan I2C

Ref: <https://www.youtube.com/watch?v=ux12i3qZEXg&t=146s>

Código:

```

while (1)
{
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    for(uint8_t i=0; i<127; i++)
    {
        if(HAL_I2C_IsDeviceReady(&hi2c1, i << 1, 1, HAL_MAX_DELAY) == HAL_OK)
        {
            printf("Device Address: 0x%x\r\n", i);
        }
    }

    HAL_Delay(2000);
}

```

Dirección I2C MPU6050: 0x68

Librería MPU6050

Configuración del MPU en STM32

Referencias:

<https://www.youtube.com/watch?v=iJn70hPxT7E>
<https://www.youtube.com/watch?v=P7a6gxacnO4>

Pines I2C1

SDA: PB7 D4

SCL: PB6 D5

Tutorial FreeCAD

Ref: https://www.youtube.com/watch?v=9Pk_5m_OqiY

AMS1117

Entrada de 4.75V a 12V

Salida depende del módulo: 3.3V o 5V

800 mA máximo

Ref:

<https://naylampmechatronics.com/fuentes-y-reguladores/166-modulo-regulador-ams1117-33v.html>

Pantalla OLED

Display Oled 1.3" I2C 128x64 SH1106

Ref: <https://naylampmechatronics.com/oled/638-display-oled-i2c-130-12864-sh1106.html>

Library: <https://github.com/afiskon/stm32-ssd1306>

Video: <https://www.youtube.com/watch?v=Ye6fF2GAhKE>

Voltaje de alimentación: 3.3V

I2C3_SDA: D12 PB4

I2C3_SCL: PA7 A6

Prueba de dirección I2C

Dirección I2C3: 0x3C

Para el trabajo con el hi2c3, tenemos que cambiar la variable SSD1306_I2C_PORT en los archivos `ssd1306.h` y `ssd1306_conf.h`. De la misma forma si queremos trabajar con otra variable I2C.

La separación en 3 filas, se coloca como índices (0, 0:16:34).

Código

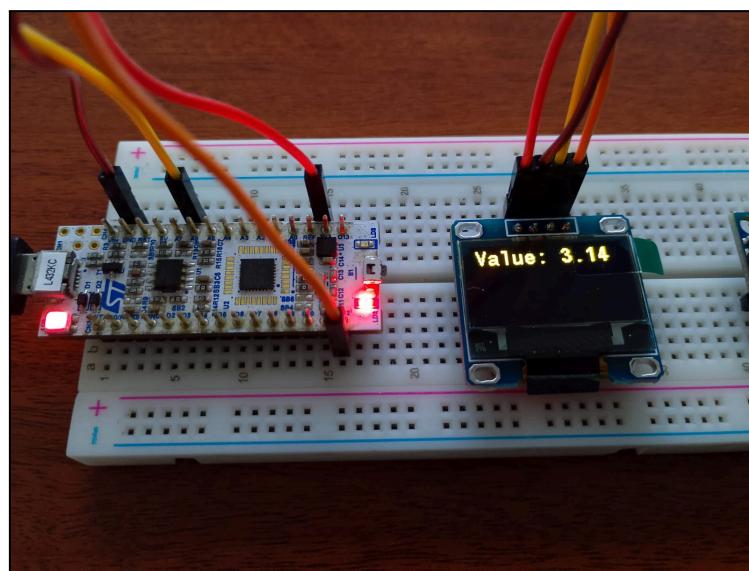
```
//SH1106
ssd1306_Init();
ssd1306_Fill(0);
ssd1306_UpdateScreen();

while (1)
{
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    float num1 = 3.1415;
    char buffer[50];

    sprintf(buffer, "Value: %f", num1);

    ssd1306_Fill(0);
    ssd1306_SetCursor(0,0);
    ssd1306_WriteString(buffer,Font_11x18,1);
    ssd1306_UpdateScreen();
}
```

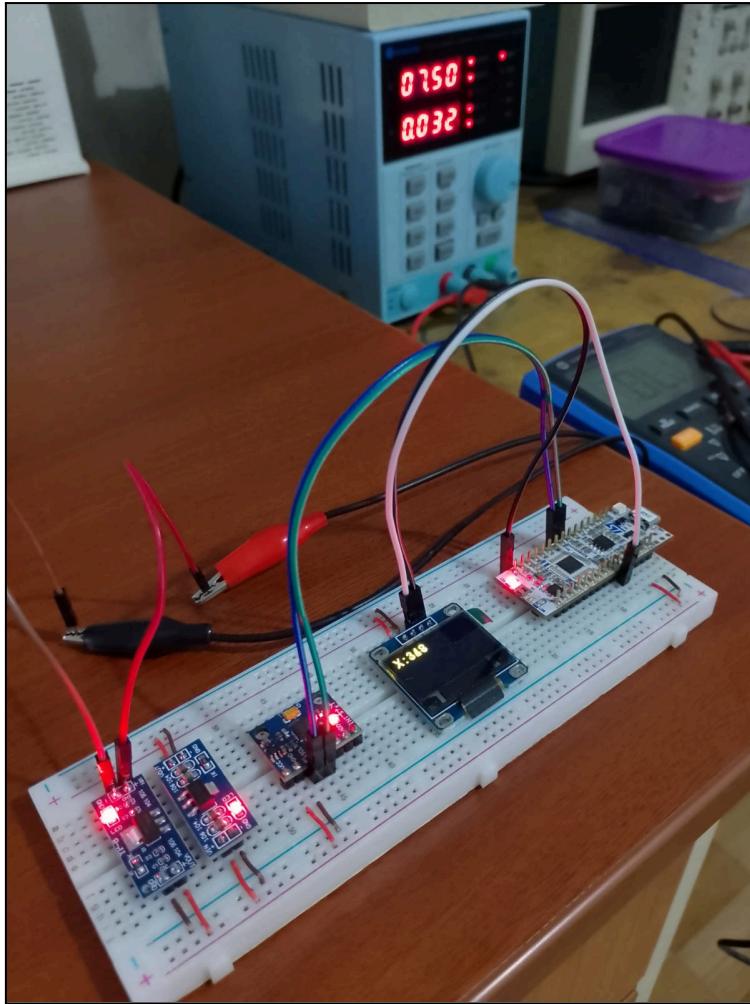
Resultados



Consumo de corriente

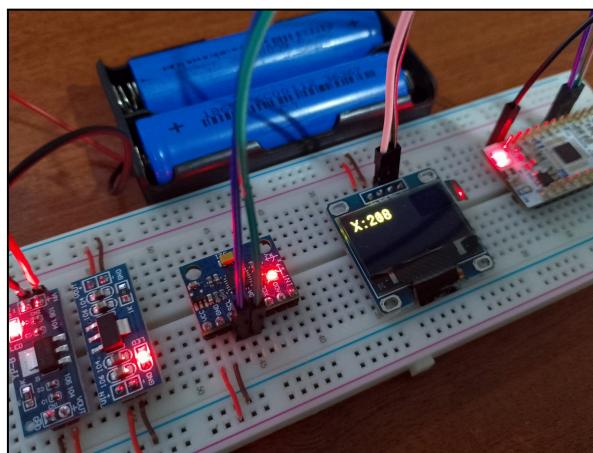
Para las pruebas de consumo de corriente de cada sensor, usamos la alimentación USB para el NUCLEO y cada sensor es alimentado por la fuente.
Para las pruebas de consumo de los pines del NUCLEO, alimentamos el NUCLEO con la fuente y cada sensor con las baterías LIPO.

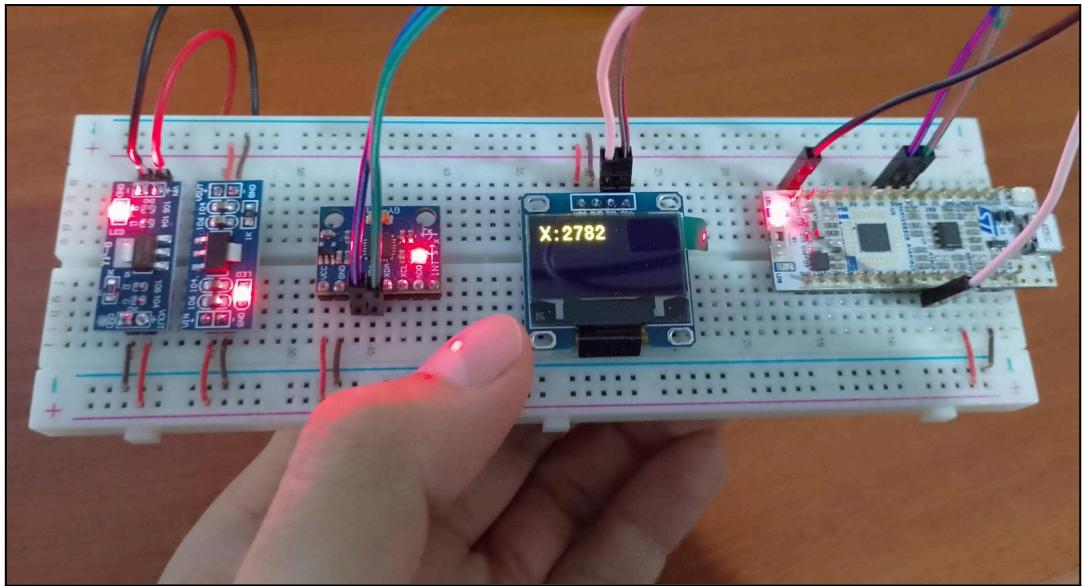
	V	mA
OLED Screen	3,3	0
MPU6050	5	2
TOTAL SEN		2
L432KC	5	
I2C1		15
I2C3		14
TOTAL uC		29
TOTAL		31



Consumo total experimental: 32mA

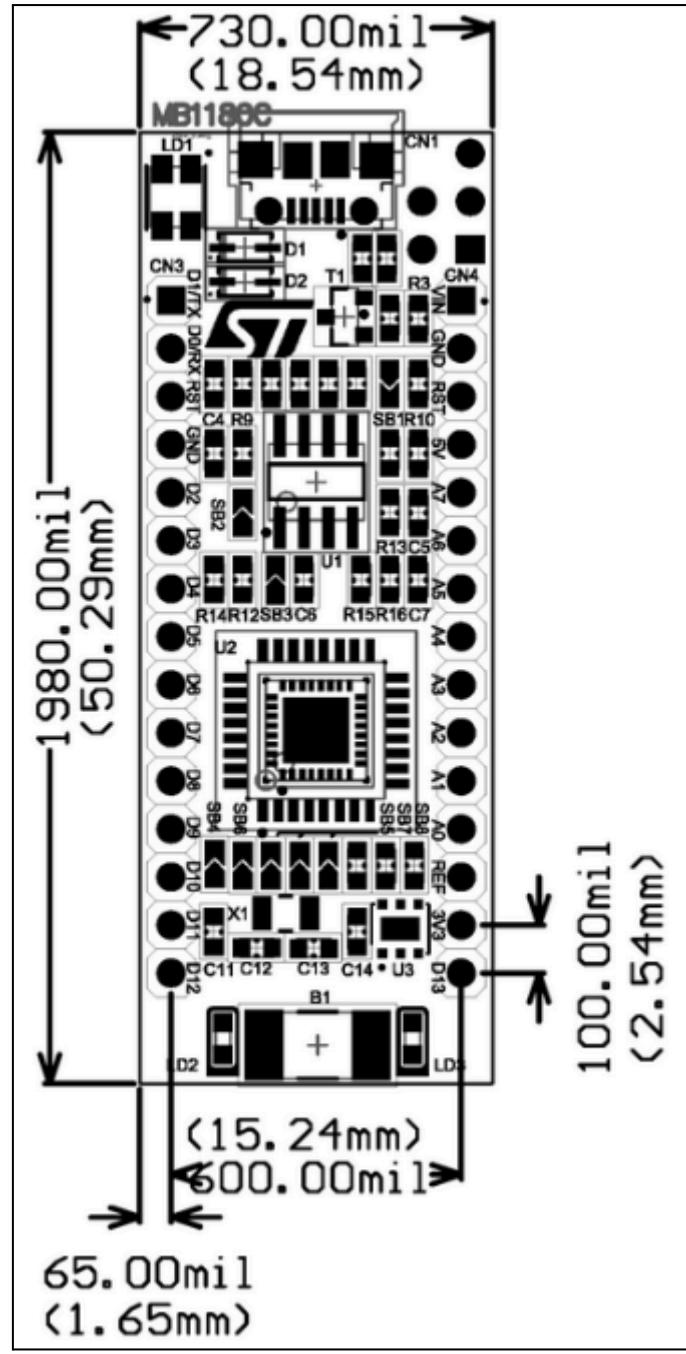
Resultados



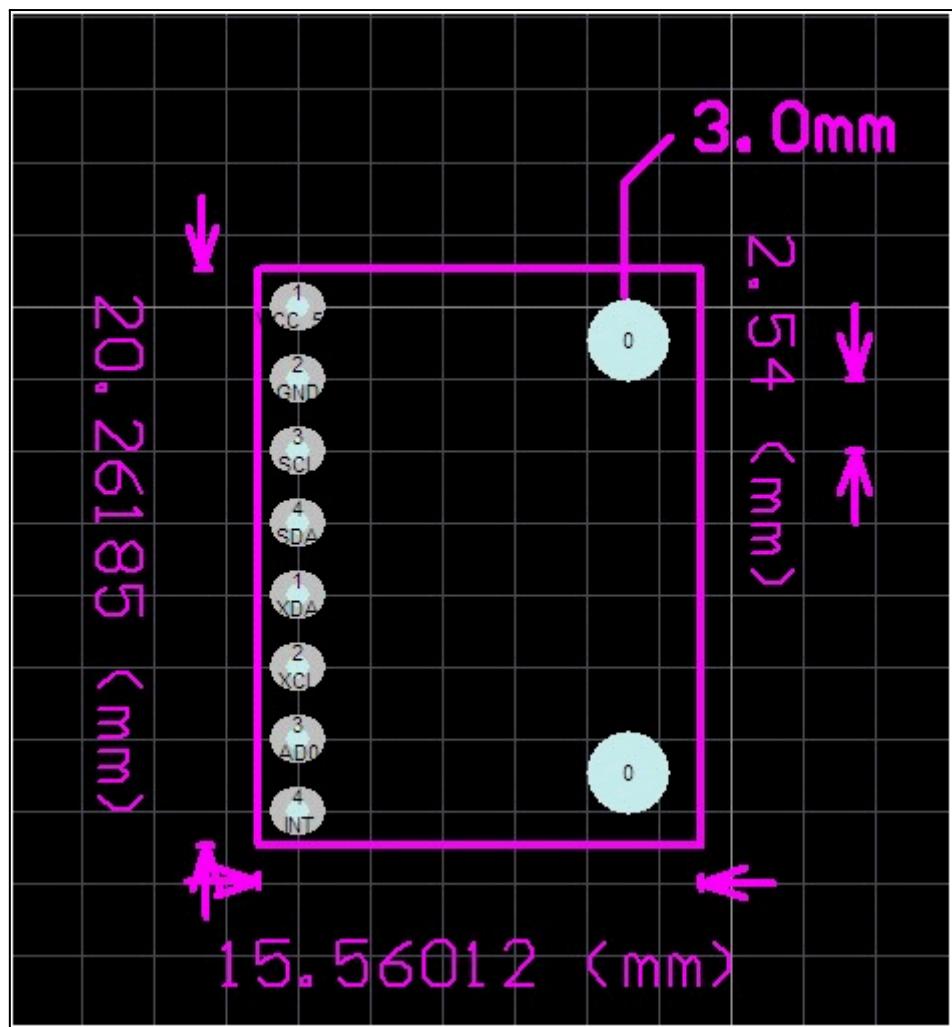


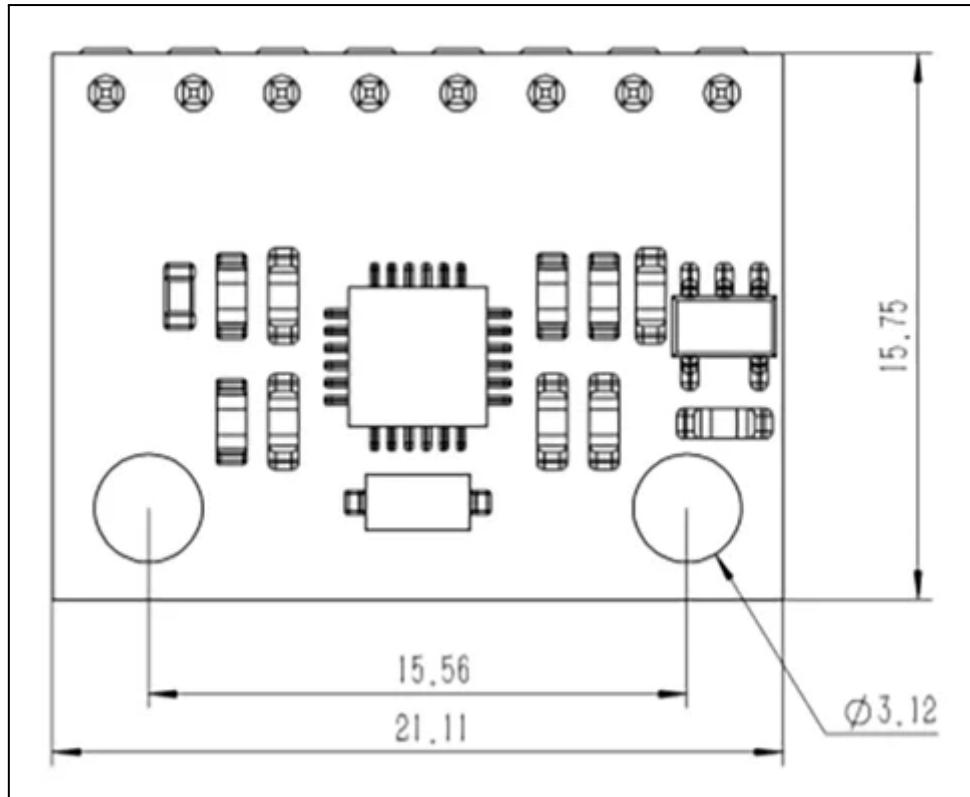
Fabricación de la PCB

Diseño 2D de L432KC

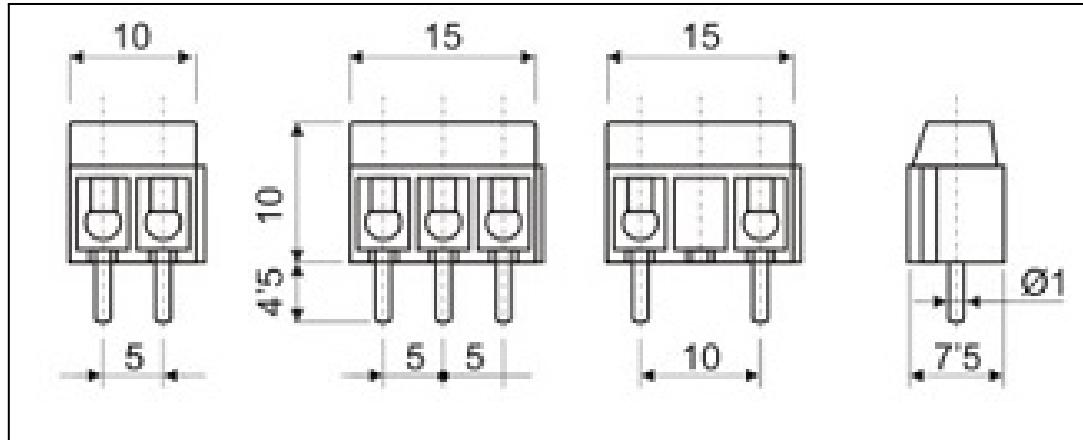


Diseño 2D de MPU6050





Diseño 2D de la bornera:



Ancho de pista estándar: 0.8 mm

Impresión 3D

Referencia:

<https://circuitdigest.com/microcontroller-projects/arduino-based-self-balancing-robot>

