UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA

LABORATÓRIO DE VEÍCULOS NÃO TRIPULADOS

Algoritmo para malha de controle de cabo conectado a submarino

Meu Primeiro Projeto uC - OS III

AUTOR: DIEGO MAGNO MONTANS ZUCHERAM 12544329

SÃO PAULO

2024

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Submarinos devem ser ligados a um cabo para não serem perdidos caso alguma falha ocorra. Para isso, foi feita uma malha de controle para controlar a liberação do cabo ou o seu enrolamento em um carretel. De forma que há um sensor que lê a tração no cabo e de acordo com a força lida o cabo vai se enrolando perfeitamente na bobina, otimizando a área dela, ou o cabo vai sendo solto, permitindo que o submarino navegue normalmente mesmo em velocidades mais altas.

Para melhorar ainda mais esse controle, foi proposta uma segunda malha de controle, utilizando uma outra placa, com MCU STM32F407VG. Para esta malha, há sete sensores ópticos espaçados, de forma que o terceiro sensor é a posição desejada para o cabo, o primeiro indica que o cabo está frouxo, logo deve tracionado e o sétimo indica que o cabo está tracionado ao máximo, logo, cabo deve ser liberado. Para isso, foi colocado um motor ligado a uma polia por onde passa o cabo e o motor tem a capacidade de soltar/liberar o cabo ou puxar/tracionar o cabo. O motor é um motor maxon e é alimentado pelo driver de motor ESCON 4-Q-DC Servoamplifier ADS 50/10. Esse driver controla o motor com base na diferença de potencial de duas entradas analógicas, Set+ e Set-, de forma que se Set+ > Set-, o motor gira no sentido anti-horário, e se Set+ < Set-, ele gira no sentido horário. Assim, com base na disposição do aparato físico, quando o motor gira no sentido anti-horário, ele libera cabo e quando ele gira no sentido horário ele puxa cabo. Em termos dos sensores, caso seja lido High dos sensores 4 ao 7 devemos ter Set+ > Set- e dos sensores 1 ao 2 Set+ < Set-.

2 METODOLOGIA

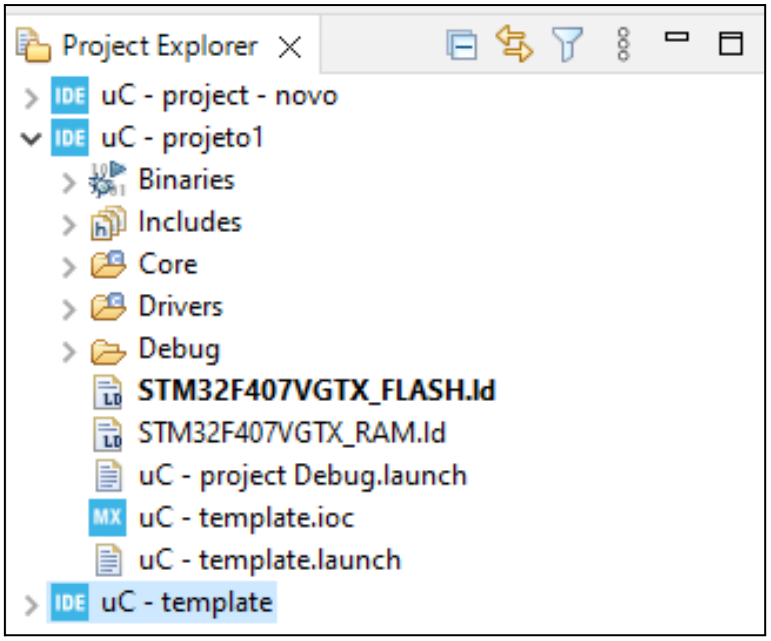
A metologia pode ser dividida em duas partes. Primeiro foi feita uma “transferência de conhecimento” do projeto “uC - template” para o projeto do problema em questão. E segundo foram criadas as funções e tarefas para resolução do problema.

2.1 “TRANFERÊNCIA DE CONHECIMENTO” EM PROJETO uC

Para início do projeto, o “Tutorial Projeto uC - template no STMCubeIDE” foi concluído com sucesso.

Este projeto foi copiado e colado no próprio workspace e dado o nome de “uC - projeto1”, como pode ser visto na Figura 1. Note que o “uC - project - novo” é o projeto já finalizado, mas aqui está sendo feito um passo a passo da minha aplicação.

Figura 1 - Project explorer com projetos uC - template e uC - projeto1 recém copiado



É necessário renomear o arquivo “uC - template.ioc” para o “nome do projeto.ioc”, ficando neste caso, “uC - projeto1.ioc”. Abrindo esse arquivo .ioc, foi configurado os pinos a serem utilizados no projeto.

O RCC foi configurado conforme a Figura 2, onde mudou-se apenas o HSE e o LSE. E, com base nas necessidades do projeto, criou-se sete pinos GPIO Input que receberão os sinais enviados pelos sensores ópticos, a setagem desses pinos pode ser vista na Figura 3, note que foram mudados os nomes dos pinos para facilitar o código e o seu entendimento e foram configurados para pull down para que quando estiverem em alta impedância estarem em Low. Ainda, é necessário dois pinos DAC que são os sinais análogicos a serem enviados para o driver ESCON do motor, em que a rotação e sentido são definidos pela diferença de potencial entre essas entradas análogicas, e a setagem pode ser vista na Figura 4, nota-se que as configurações estão todas no default mas mudou-se o nome dos pinos.

Ademais, não foram configurados LEDs, apesar de que serão utilizados, pois no uC - template eles já estão configurados e já há funções destinadas a eles. Assim, a Figura 5 mostra o .ioc final da placa.

Figura 2 - Configuração RCC no arquivo .ioc

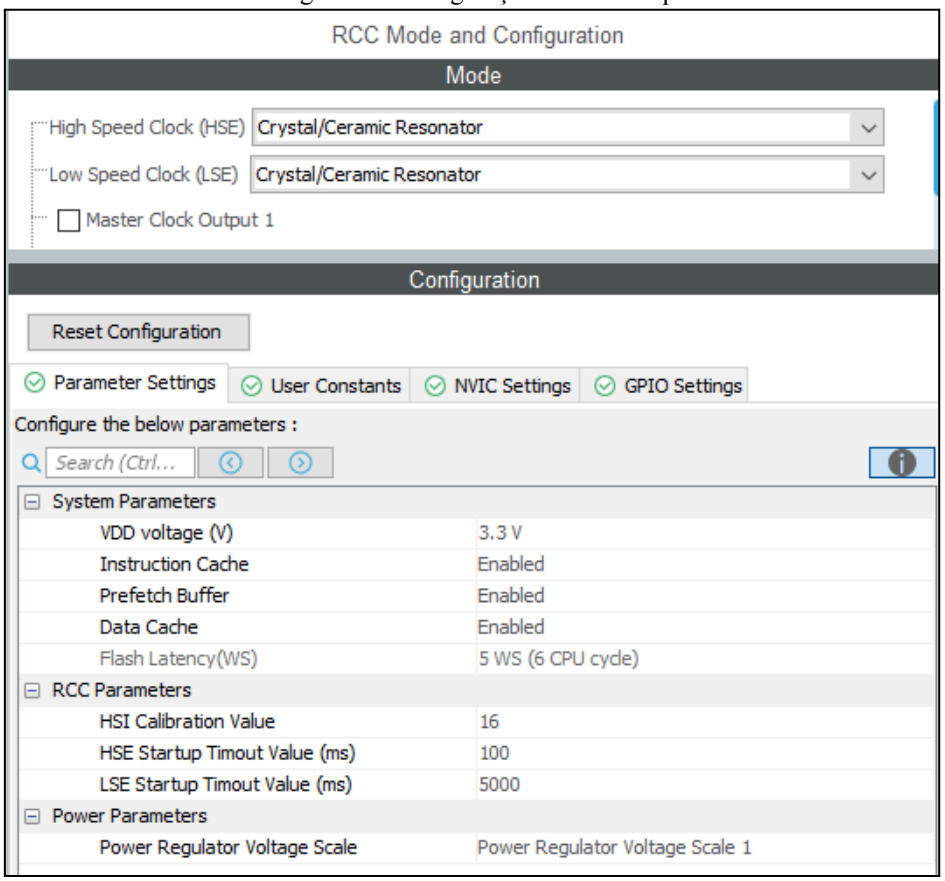
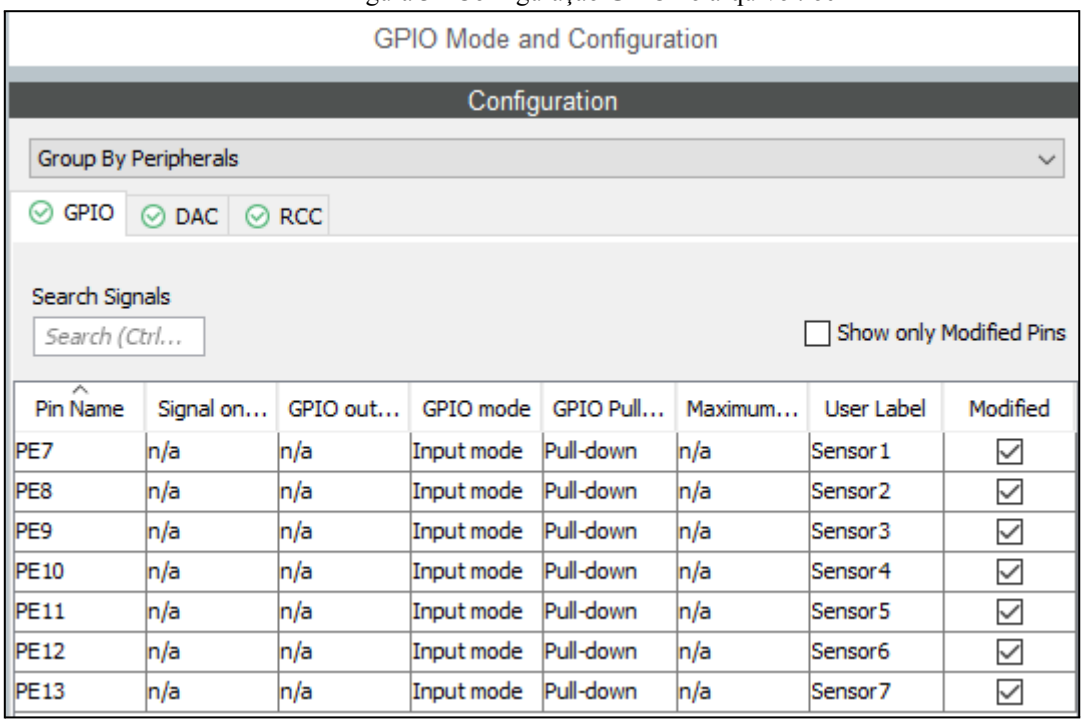


Figura 3 - Configuração GPIO no arquivo .ioc



Pin Name	Signal on...	GPIO out...	GPIO mode	GPIO Pull...	Maximum...	User Label	Modified
PE7	n/a	n/a	Input mode	Pull-down	n/a	Sensor1	☑
PE8	n/a	n/a	Input mode	Pull-down	n/a	Sensor2	☑
PE9	n/a	n/a	Input mode	Pull-down	n/a	Sensor3	☑
PE10	n/a	n/a	Input mode	Pull-down	n/a	Sensor4	☑
PE11	n/a	n/a	Input mode	Pull-down	n/a	Sensor5	☑
PE12	n/a	n/a	Input mode	Pull-down	n/a	Sensor6	☑
PE13	n/a	n/a	Input mode	Pull-down	n/a	Sensor7	☑

Figura 4 - Configuração DAC no arquivo .ioc

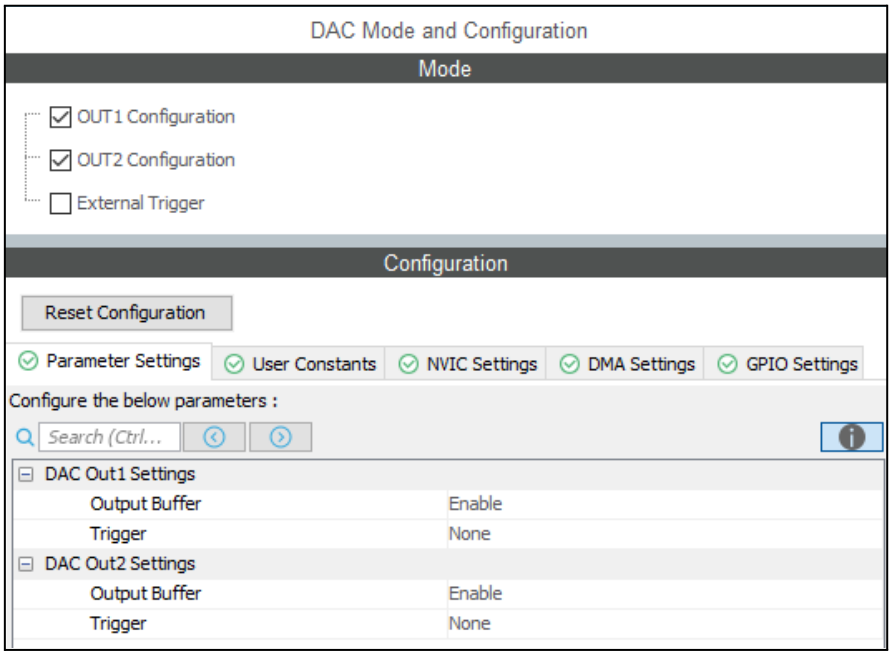
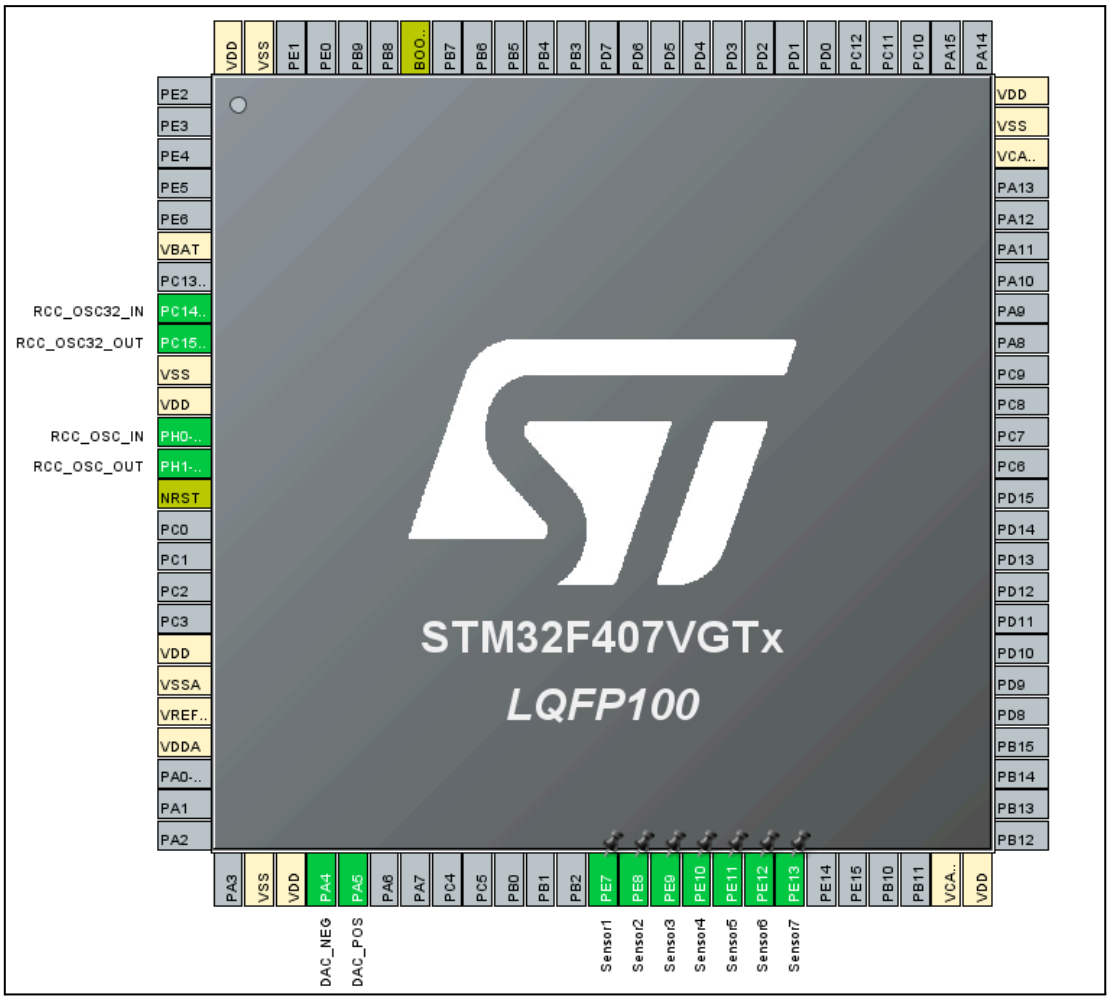


Figura 5 - Pinos e configuração do MCU



Finalizando o arquivo .ioc, deve-se gerar o código, atalho Alt+K. Ao gerar o código, os arquivos das pastas Core/Inc e Core/Driver serão atualizados e para esta aplicação é necessário copiar os arquivos main.c, main.h e stm32f4xx_hal_msp.c para a pasta APP/Config, ou qualquer pasta que esteja com build para os arquivos .c e qualquer pasta que esteja dentro dos includes para o arquivo .h.

Ainda, foi necessário realizar algumas mudanças nos arquivos para funcionar corretamente como:

- No app.c, tirar a definição e declaração da função SystemClkCfg() e substituir a chamada dessa função dentro da main por SystemClock_Config().
- Na main.c (depois de colada na pasta APP/Config), apagar a função main, e trocar as funções de static void para void.
- Na main.h (depois de colado na pasta APP/Config), como há uma variável definida na main.c, isto é, DAC_HandleTypeDef hdac, adicionar extern DAC_HandleTypeDef hdac, para a mesma variável poder ser utilizada no app.c depois.

O projeto pode ser compilado para garantir que não haja nenhum erro.

2.2 APP.C - FUNÇÕES E TAREFAS PARA RESOLUÇÃO DO PROBLEMA

Nesta etapa da metodologia, foram feitas alterações apenas no arquivo app.c, isto porque os outros arquivos importados já cuidam da parte dos pinos. Mais precisamente, o main.h cria defines para atribuir os nomes dados pelo usuário no .ioc ao pino em si, a main.c cria as funções para serem utilizadas no app.c em relação aos periféricos, definindo cada pino como o periférico desejado e a stm32f4xx_hal_msp.c permite o RCC e, para este problema, cria os pinos analógicos do DAC.

2.2.1 Includes

```
#include <os.h>
#include <stdlib.h>
#include <stdbool.h>
#include "stm32f4_discovery.h"
#include "main.h"
```

2.2.2 Defines

```
#define INITIAL_TASK_PRIO              5u
#define INITIAL_TASK_STK_SIZE          (256u)
#define SENSORES_RNT_TASK_PRIO         (INITIAL_TASK_PRIO - 2)
#define SENSORES_RNT_TASK_STK_SIZE     (256u)
#define VOLTAGEM_MAX                   3 //Voltagem máxima a ser entregue ao driver do motor
```

2.2.3 Variáveis

```
/* Parâmetros para as Tasks*/
static OS_TCB          initialTaskTCB;
static CPU_STK          initialTaskStk[INITIAL_TASK_STK_SIZE];
static OS_TCB          sensoresRntTaskTCB;
static CPU_STK          sensoresRntTaskStk[SENSORES_RNT_TASK_STK_SIZE];
/* Definição das Tasks */
static void initialTask (void *p_arg);
static void sensoresRntTask (void *p_arg);
/*Função para ler os sensores e enviar as saídas analógicas*/
static void le_sensores (void);
GPIO_PinState sensor1_read;
GPIO_PinState sensor2_read;
GPIO_PinState sensor3_read;
GPIO_PinState sensor4_read;
GPIO_PinState sensor5_read;
GPIO_PinState sensor6_read;
GPIO_PinState sensor7_read;
float positive_voltage_value = 0;
float negative_voltage_value = 0;
uint16_t positive_dac_value;
uint16_t negative_dac_value;
```

2.2.4 Main

A main é responsável por iniciar o HAL (HAL_Init()), o RCC (SystemClock_Config()), o sistema operacional (OSInit(&err) e OSStart(&err)) e criar a Task inicial (OSTaskCreate(...)).

```
int main(void)
{
  OS_ERR  err;
  HAL_Init();
  SystemClock_Config();
  OSInit(&err);
  OSTaskCreate(    (OS_TCB       *)&initialTaskTCB,
                   (CPU_CHAR     *)"Initial Task",
                   (OS_TASK_PTR   )initialTask,
                   (void         *)0u,
                   (OS_PRIO       )INITIAL_TASK_PRIO,
                   (CPU_STK      *)&initialTaskStk[0u],
                   (CPU_STK_SIZE  )initialTaskStk[INITIAL_TASK_STK_SIZE / 10u],
                   (CPU_STK_SIZE  )INITIAL_TASK_STK_SIZE,
                   (OS_MSG_QTY    )0u,
                   (OS_TICK       )0u,
                   (void         *)0u,
                   (OS_OPT        )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                   (OS_ERR       *)&err);
  OSStart(&err);
  while (DEF_TRUE) {
  }
  return 0;
}
```

2.2.5 initialTask - Task inicial da aplicação

A Task inicial inicia todos os periféricos a serem utilizados e cria a task principal da aplicação que é a sensoresRntTask (sensores read and transmit task), por ser a task principal deve ter uma prioridade maior do que a initialTask. Ainda, há um comando de LED para monitoramento e a tarefa utiliza como recurso de multi-tasking o delay.

```
/* Inicia os perifericos e a taks de ler os sensores*/
static void initialTask (void *p_arg)
{
  OS_ERR  err;
  (void)p_arg;
  MX_GPIO_Init();
```

```
  MX_DAC_Init();
  BSP_LED_Init(LED3);
  BSP_LED_Init(LED4);
  BSP_LED_Init(LED5);
  BSP_LED_Init(LED6);
  OSTaskCreate(    (OS_TCB       *)&sensoresRntTaskTCB,
                   (CPU_CHAR     *)"Sensores Read and Transmit Task",
                   (OS_TASK_PTR   )sensoresRntTask,
                   (void         *)0u,
                   (OS_PRIO       )SENSORES_RNT_TASK_PRIO,
                   (CPU_STK      *)&sensoresRntTaskStk[0u],
                   (CPU_STK_SIZE  )sensoresRntTaskStk[SENSORES_RNT_TASK_STK_SIZE / 10u],
                   (CPU_STK_SIZE  )SENSORES_RNT_TASK_STK_SIZE,
                   (OS_MSG_QTY    )0u,
                   (OS_TICK       )0u,
                   (void         *)0u,
                   (OS_OPT        )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                   (OS_ERR       *)&err);
  while (DEF_TRUE) {
     BSP_LED_Toggle(LED6);
     OSTimeDly(500u, OS_OPT_TIME_PERIODIC , &err);
  }
}
```

2.2.6 sensoresRntTask - Task principal

Essa task basicamente starta os DACs e chama a função le_sensores() a cada 10ms.

```
/*sensores read and transmit task*/
static void sensoresRntTask (void *p_arg)
{
        OS_ERR  err;
        (void)p_arg;
        HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
        HAL_DAC_Start(&hdac, DAC_CHANNEL_2);
        while (DEF_TRUE) {
           BSP_LED_Toggle(LED5);
           le_sensores();
```

```
        OSTimeDly(10u, OS_OPT_TIME_PERIODIC , &err); //Máximo de 1ms de delay já que OS_CFG_TICK_RATE_HZ = 1000Hz
    }
}
```

2.2.7 le_sensores

Essa função é responsável por ler o estado de todos os pinos GPIO input, isto é, os sinais dos sete sensores ópticos e ter como saída o sinal analógico nos pinos PA4 e PA5. Isto acontece de forma que o sensor 3 indica a posição desejada do cabo, dos sensores 4 ao 7 o cabo deve ser liberado, girando o motor no sentido anti-horário e por isso o valor positivo deve ser maior que o negativo. E dos sensores 1 a 2 ocorre o contrário. Ainda, depois há uma conversão do valor desejado em volts (analógico) para o valor digital, sendo o valor em volts multiplicado por 4095 pois a resolução do DAC é de 12 bits. E por fim, esse valor digital é escrito no DAC, obtendo uma saída física em volts novamente.

```
static void le_sensores (void){
    sensor1_read = HAL_GPIO_ReadPin(Sensor1_GPIO_Port, Sensor1_Pin);
    sensor2_read = HAL_GPIO_ReadPin(Sensor2_GPIO_Port, Sensor2_Pin);
    sensor3_read = HAL_GPIO_ReadPin(Sensor3_GPIO_Port, Sensor3_Pin); /* SENSOR BASE */
    sensor4_read = HAL_GPIO_ReadPin(Sensor4_GPIO_Port, Sensor4_Pin);
    sensor5_read = HAL_GPIO_ReadPin(Sensor5_GPIO_Port, Sensor5_Pin);
    sensor6_read = HAL_GPIO_ReadPin(Sensor6_GPIO_Port, Sensor6_Pin);
    sensor7_read = HAL_GPIO_ReadPin(Sensor7_GPIO_Port, Sensor7_Pin);
    if (sensor7_read) {
        positive_voltage_value = 1*VOLTAGEM_MAX; //3
        negative_voltage_value = 0;
    }
    else if (sensor6_read) {
        positive_voltage_value = 0.75*VOLTAGEM_MAX; //2.25
        negative_voltage_value = 0;
    }
    else if (sensor5_read) {
        positive_voltage_value = 0.5*VOLTAGEM_MAX; //1.5
        negative_voltage_value = 0;
    }
    else if (sensor4_read) {
        positive_voltage_value = 0.25*VOLTAGEM_MAX; //0.75
```

```
            negative_voltage_value = 0;
        }
        else if (sensor3_read) {
            positive_voltage_value = 0;
            negative_voltage_value = 0;
        }
        else if (sensor2_read) {
            positive_voltage_value = 0;
            negative_voltage_value = 0.5*VOLTAGEM_MAX; //-1.5
        }
        else if (sensor1_read) {
            positive_voltage_value = 0;
            negative_voltage_value = 1*VOLTAGEM_MAX; //-3
        }
        positive_dac_value = (uint16_t)((positive_voltage_value*4095)/VOLTAGEM_MAX);
        negative_dac_value = (uint16_t)((negative_voltage_value*4095)/VOLTAGEM_MAX);
        HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, negative_dac_value);
//CHANNEL1 - PA4
        HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, positive_dac_value);
//CHANNEL2 - PA5
}
```

3 RESULTADOS

A fim de observar o funcionamento da aplicação, utilizou-se do osciloscópio para observar a diferença de tensão entre a saída positiva e negativa para diferentes pulsos nos pinos GPIO Inputs. Assim, a Figura 6 mostra o setup de testes, com a placa, os fios vermelhos sendo os inputs e as garras do osciloscópio conectadas aos fios azuis, que são saídas analógicas. A Figura 7 mostra a diferença de potencial depois do “sensor” 6 ficar em High, gerando 2.25 V na saída analógica a ser utilizada no set de tensão positiva do driver do motor, representada pelo canal 2 (verde). A Figura 8 mostra a diferença de potencial depois do “sensor” 2 ficar em High, gerando 1.5 V na saída analógica a ser utilizada no set de tensão negativa do driver do motor, representada pelo canal 3 (laranja).

Figura 6 - Setup de testes para leitura de sinais analógicos a serem utilizados no set positivo e negativo do driver do motor

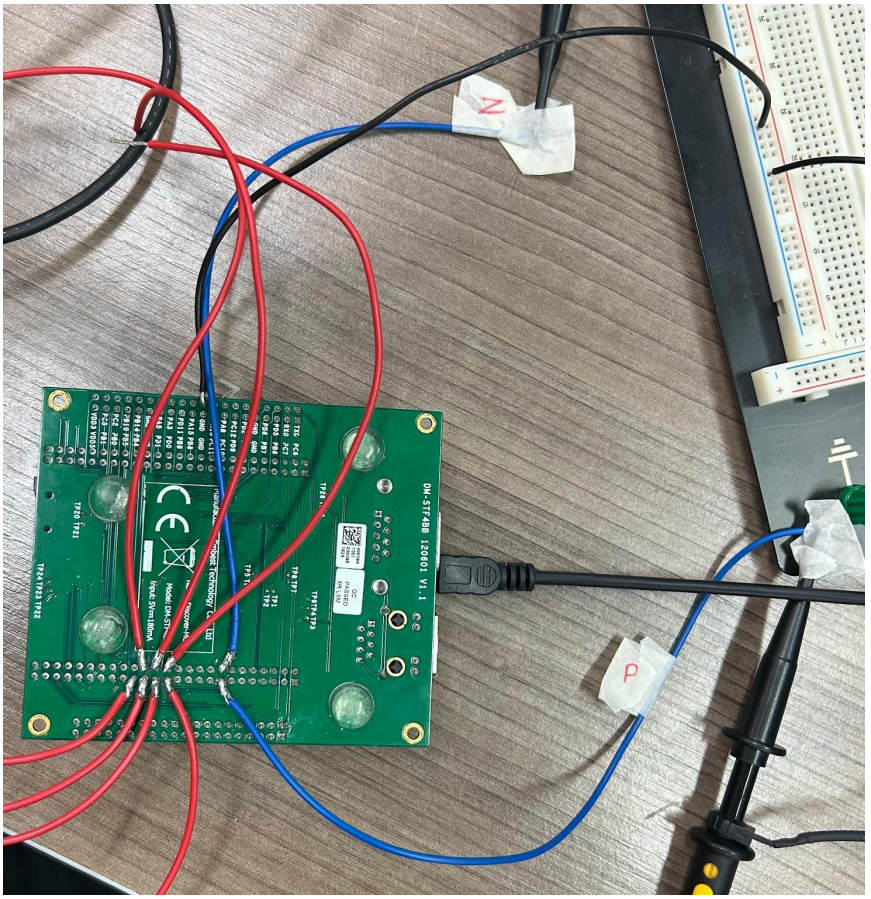


Figura 7 - Leitura dos sinais analógicos depois de pulso no sensor 6

Figura 8 - Leitura dos sinais analógicos depois de pulso no sensor 6