Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Systemy wbudowane i sterowniki
Sterowanie silnikiem BLDC

Maksymilian Łuc
Tomasz Talarczyk

rok akademicki 2023/2024

Contents

1 Wprowadzenie

1.1 Cel projektu

Celem projektu było uzyskanie możliwości sterowania bezszczotkowym silnikiem pradu stałego (BLDC) z poziomu aplikacji na komputer PC. Aplikacja miała zapewniać następujące funkcjonalności:

- Regulacja prędkości obrotowej silnika;
- Regulacja momentu obrotowego;
- Ustawienie kierunku obrotu;
- Podgląd aktualnie ustawionych danych;
- Możliwość przejścia na sterowanie prędkością z użyciem potencjometru.

Implementacja powyższych podpunktów ma na celu zapewnić instynktowne i przejrzyste sterowanie silnikiem dla użytkownika końcowego. Kontrola z poziomu komputera miała być możliwa poprzez podłączenie urządzenia/ urządzeń przez USB. Do realizacji tego celu dobrano odpowiednie komponenty.

1.2 Wykorzystane komponenty

W projekcie wykorzystane zostały poniższe komponenty:

1. **STM-F411 E-Disco** – jest to płytka często stosowana w celach edukacyjnych. Posiada wiele różnych funkcjonalności, takich jak przetwornik ADC, przycisk dostępny dla użytkownika, wiele diod, pozwala na komunikację przez UART itp.

2. **X-NUCLEO-IHM07M1** – płytka będąca rozszerzeniem do płytek Nucleo, pozwalająca na sterowanie silnikami za pomocą sterownika L6230. Niestety płytka Nucleo z zestawu, który został wykorzystany nie działała – niemożliwa była komunikacja przez ST-Link. Przysporzyło to dodatkowych trudności – konieczne było odnalezienie w dokumentacji portów wejściowych koniecznych do użycia na płytce X-Nucleo.

   Płytka ta pozwalała na wzmacnianie wygenerowanych przez płytkę F411 sygnałów PWM i kierowanie ich na 3 fazy silnika BLDC. Posiadała również 3 piny *ENABLE*, służące do aktywacji tych faz.

3. **Silnik BLDC** – poniżej przedstawione zostało zdjęcie użytego w projekcie silnika

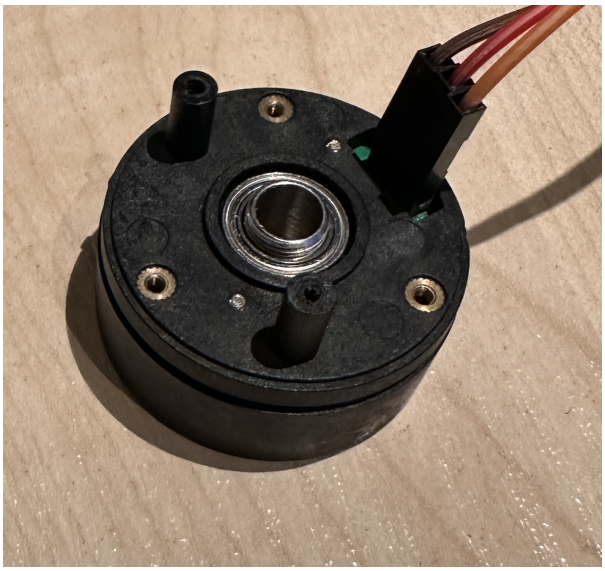

Figure 1: Użyty silnik BLDC

4. **Zasilacz laboratoryjny KORAD KD3005D 30V 5A** – w projekcie użyty został laboratoryjny zasilacz, aby zapewnić możliwość monitorowania poboru prądu przez silnik. Było to bezpieczniejsze, niż podpięcie uproszczonego zasilacza z odpowiednio dobranymi parametrami.

Poniżej przedstawione zostało zdjęcie wykorzystanych komponentów połączonych przewodami.

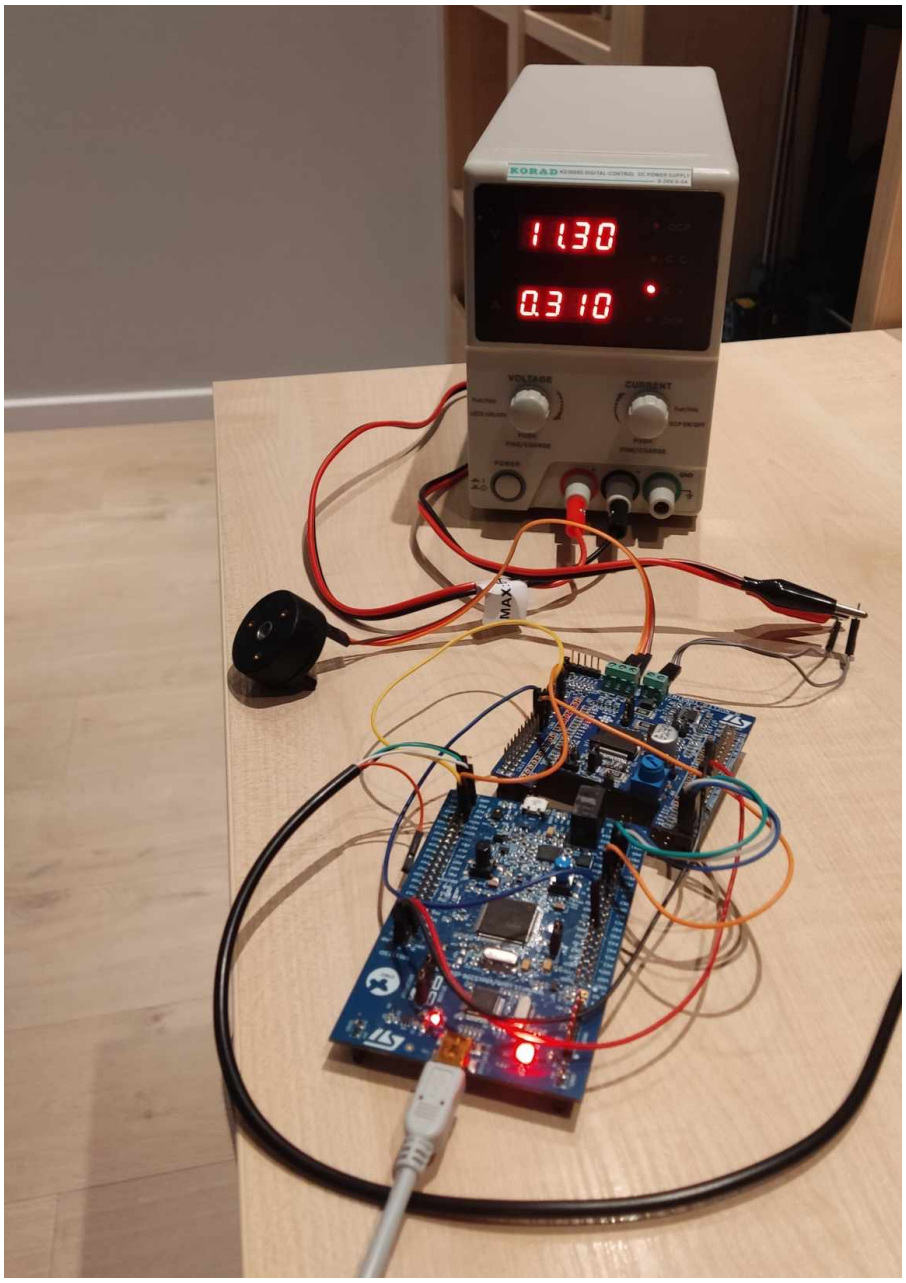

Figure 2: Podłączone komponenty

2 Przebieg projektu

W tym rozdziale przedstawiony został przebieg projektu – opisana została konfiguracja, schemat połączeń oraz przedstawiona została aplikacja do sterowania silnikiem.

2.1 Konfiguracja mikrokontrolera

Poniższe zdjęcie przedstawia konfigurację portów wyjściowych.

Diody oraz przycisk zostały skonfigurowane do celów developerskich, ale nie są używane w projekcie.

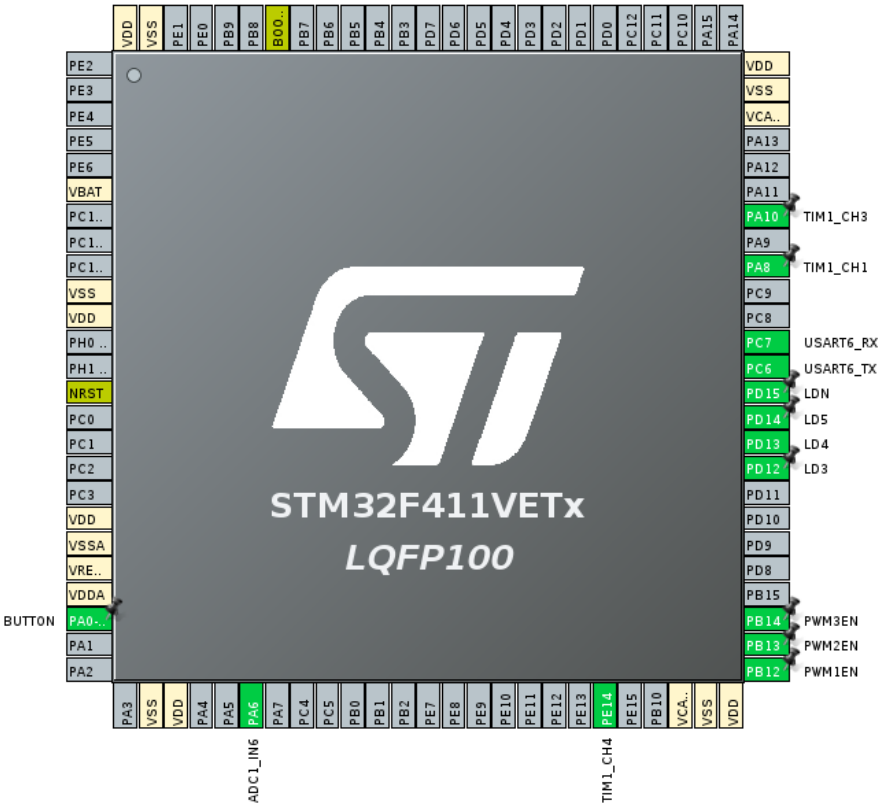

Figure 3: Konfiguracja portów wyjściowych w programie STM32 Cube IDE

Poniżej przedstawiona została konfiguracja zegara TIM1, wykorzystanego w projekcie.

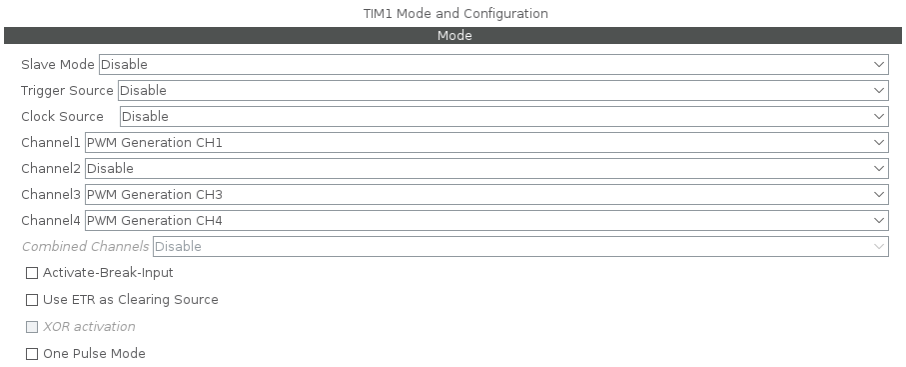

Figure 4: Konfiguracja sygnałów PWM zegara w Cube IDE

Poniżej przedstawione zostało zdjęcie pokazujące konfigurację zegara 1 – został on ustawiony na częstotliwość 20kHz, tak aby silnik poruszał się płynnie, a generowane sygnały nie były słyszalne dla ucha.

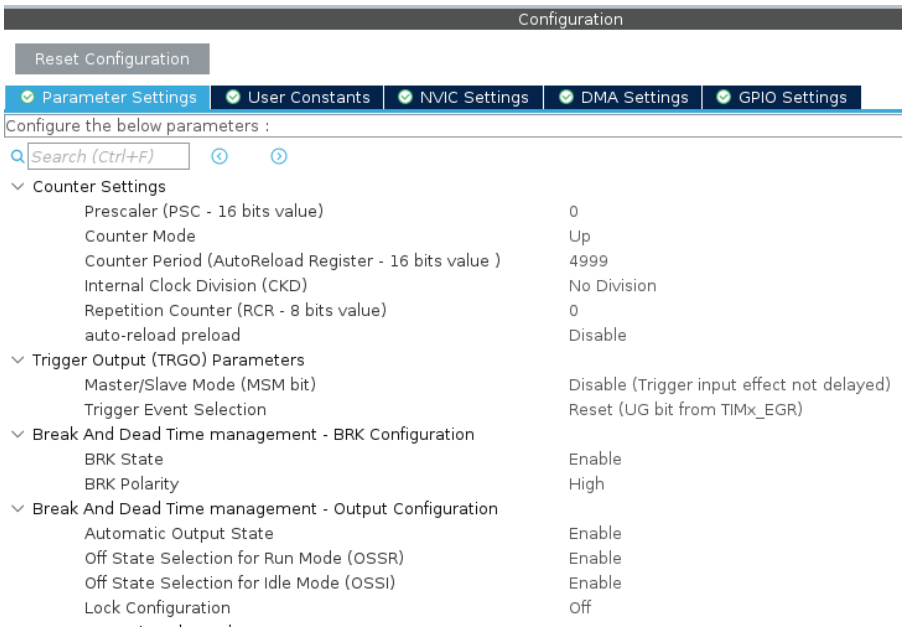

Figure 5: Konfiguracja zegara w Cube IDE

Sygnały PWM zostały skonfigurowane według ustawień domyślnych – zrzuty ekranu zostały pominięte. Zostały użyte kanały 1, 3 oraz 4 zegara pierwszego, ponieważ zegar drugi dawał niezadowalające rezultaty – nie był w stanie generować pożądanych sygnałów PWM.

2.2 Schemat połączeń

Poniżej przedstawiono schemat połączeń między wykorzystanymi komponentami, umożliwiający ich poprawną współpracę, realizującą założenia projektowe:

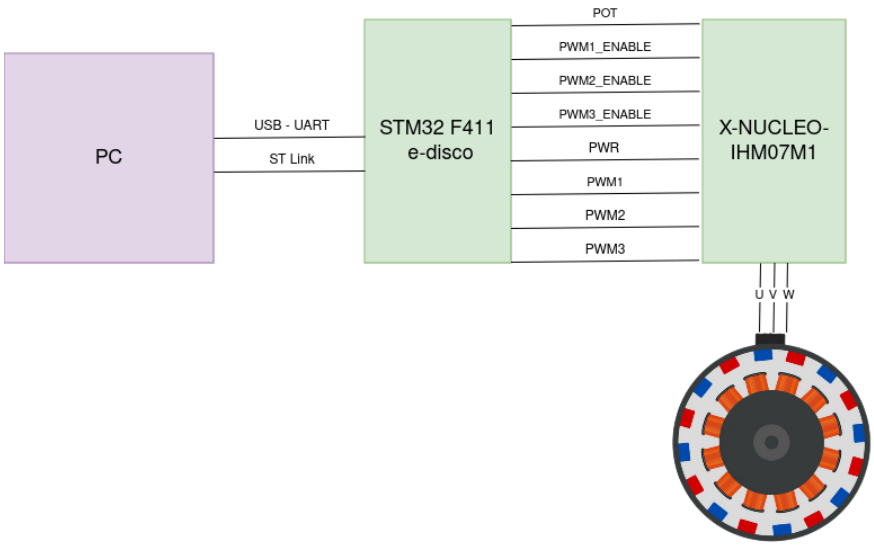

Figure 6: Schemat połączeń w projekcie

2.3 Struktura programu

W projekcie zastosowano system operacyjny czasu rzeczywistego FreeRTOS, który pozwala na efektywne zarządzanie wielozadaniowością, umożliwiając wykonanie kilku niezależnych zadań w sposób równoległy.

W ramach projektu utworzono trzy taski, z których każdy odpowiada za odrębne funkcje:

- **Sterowanie silnikiem** - odpowiada za kontrolę pracy silnika BLDC, wykorzystując algorytm sterowania opisany w następnym rozdziale.
- **Obsługa potencjometru** - zajmuje się odczytem wartości z potencjometru, który wpływa na prędkość obrotową silnika.
- **Komunikacja z użytkownikiem** - umożliwia interakcję z użytkownikiem, pozwalając na monitorowanie i sterowanie silnikiem.

Dzięki zastosowaniu FreeRTOS, program został zaprojektowany w sposób modułowy i przejrzysty, co ułatwia zarówno rozwój, jak i utrzymanie kodu w przyszłości.

Poniżej przedstawiono zrzut ekranu z konfiguracji systemu w Cube IDE. Zadaniom przypisano priorytety oraz zdefiniowano odpowiednio duży rozmair stosu.

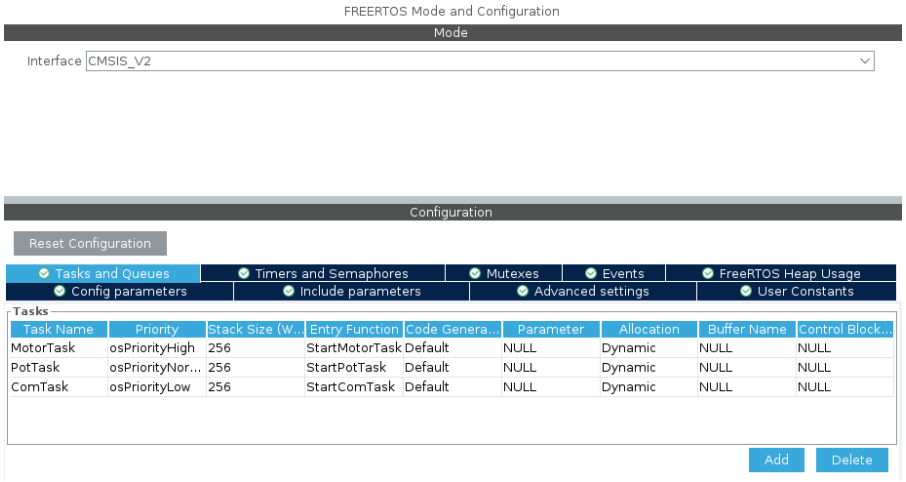

Figure 7: Konfiguracja FreeRTOS w Cube IDE

Funkcje realizowane wewnątrz zadań podzielono zgodnie z poniższym schematem:

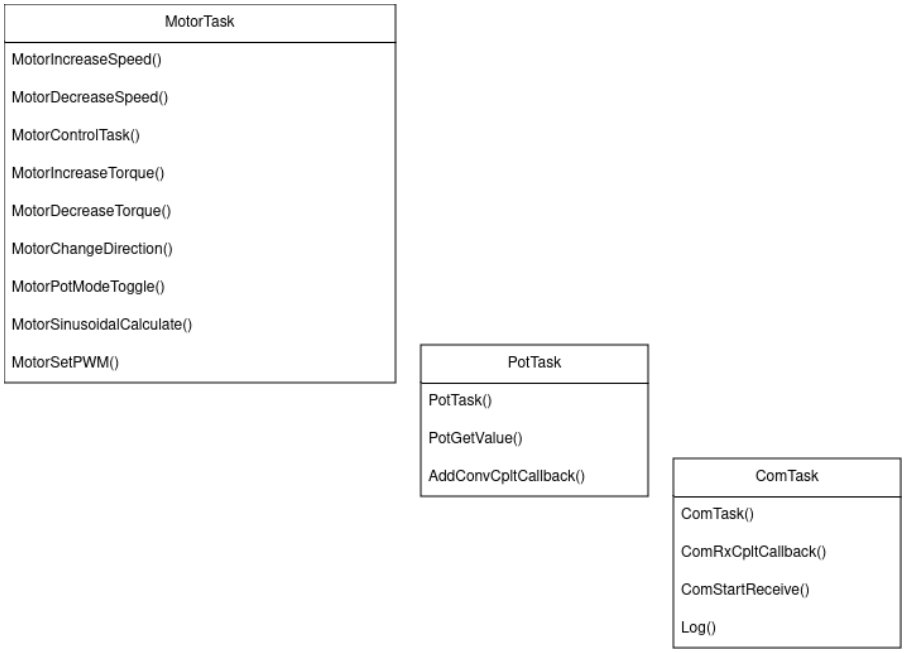

Figure 8: Taski w programie wraz z zawartymi w nich funkcjami

Większość funkcji wykonuje operacje wynikające wprost ich nazwy, pozostałe odpowiadają za komunikację, opisaną szczegółowo w odpowiednim podrozdziale.

2.4 Algorytm sterowania

Na początku projektu silnik miał być sterowany algorytmem 6 krokowym. Jest to najprostszy sposób sterowania silnikami BLDC i polega na przełączaniu zasilania na odpowiednie fazy silnika w równych odstępach czasowych. Na fazach, na których płynie prąd konieczne jest ustawienie pinów `ENABLE` na stan wysoki.

Poniżej przedstawiona została wizualizacja obrazująca działanie tego algorytmu.

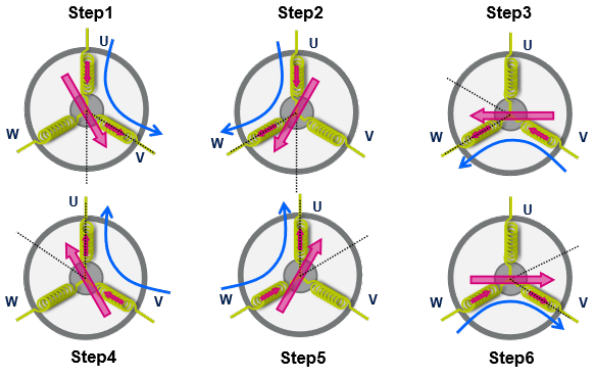

Figure 9: 6 krokowy algorytm sterowania silnikiem

Następnie zaimplementowany został algorytm sinusoidalny. Pozwala on na płynniejszą, stabilniejszą pracę silnika, oraz łatwe sterowanie prędkością ruchu silnika (za pomocą zmian wartości zmiennej `step`). Na fazy silnika wysyłane są sinusoidalne sygnały PWM, liczone za pomocą poniższej funkcji:

```
static void MotorSinusoidalCalculate(float step, float power) {
    const uint16_t period = TIM1->ARR;
    static float phase = 0.0;

    float phaseA = 0.5f + 0.5f * sinf(phase);
    float phaseB = 0.5f + 0.5f * sinf(phase + 2.0f * M_PI / 3.0f);
    float phaseC = 0.5f + 0.5f * sinf(phase + 4.0f * M_PI / 3.0f);

    uint16_t pwmA = (uint16_t)(phaseA * power * period);
    uint16_t pwmB = (uint16_t)(phaseB * power * period);
    uint16_t pwmC = (uint16_t)(phaseC * power * period);

    MotorSetPWM(pwmA, pwmB, pwmC);

    phase += step;

    if (phase > 2.0f * M_PI) {
        phase -= 2.0f * M_PI;
    }
}
```

W tym przypadku konieczne jest ustawienie pinów `ENABLE` na stan wysoki w całym okresie trwania algorytmu.

2.5 Komunikacja

W tym rozdziale opisane zostały funkcjonalności wykorzystane przy komunikacji ze sterownikiem oraz napotkane podczas pracy problemy i ich rozwiązania.

2.5.1 Wysyłanie rozkazów do sterownika

Początkowo, sterowanie silnikiem odbywało się za pomocą klawiatury, poprzez wysyłanie znaków wywołujących odpowiednie funkcje. Zostało to zaimplementowane używając `DMA`, wywołując funkcję `HAL_UART_Receive_DMA`.

Odbywało się to w przerwaniu, wywołanym w poniższy sposób w pliku `stm32f4xx_it.c`:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart->Instance == USART6) {
        ComRxCpltCallback();
    }
}
```

gdzie funkcja `ComRxCpltCallback()` wywoływała funkcje zarządzające silnikiem w zależności od odebrania odpowiedniego znaku.

2.5.2 Potencjometr

Podobnie został rozwiązany problem odbierania danych z potencjometru. Dla przetwornika ADC zostały włączone przerwania (`ADC1 Global interrupt`) oraz DMA. Do przerwania znajdującego się w pliku odpowiadającym za przerwania została dopisana funkcjonalność pobierająca dane z ADC w odstępach 10ms (nie było powodu, żeby odczyty były częściej, a dzięki temu była pewność poprawnego odczytu z ADC).

2.5.3 Odbieranie danych od sterownika silnika

Konieczne było również otrzymywanie informacji zwrotnych od sterownika o aktualnych parametrach zadanych sterownikowi. Początkowo użyta została funkcja `HAL_UART_Transmit_DMA`, ale program blokował się i nie progresował dalej. Konieczne było zwiększenie buforów w Taskach FreeRTOS do wartości 256 (zamiast 128) oraz napisanie funkcji:

```
static void Log(const char* format, ...) {
    static char logBuffer[LOG_BUFFER_SIZE];
    va_list args;
    va_start(args, format);

    vsnprintf(logBuffer, LOG_BUFFER_SIZE, format, args);

    va_end(args);

    HAL_UART_Transmit_DMA(&huart6, (uint8_t*)logBuffer, strlen(logBuffer));
}
```

2.6 Aplikacja do sterowania silnikiem

Do sterowania silnikiem napisana została prosta aplikacja w języku Python (`bldc.py`). Poniżej przedstawione zostało zdjęcie z aplikacji.

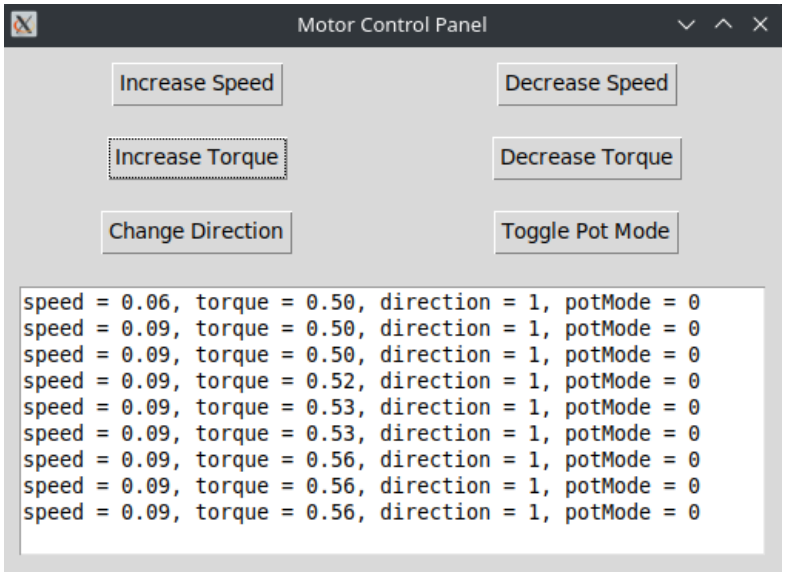

Figure 10: Zrzut ekranu z aplikacji do sterowania silnikiem

W aplikacji możliwe było sterowanie prędkością, momentem oraz kierunkiem ruchu silnika. Możliwe było również sterowanie prędkością silnika za pomocą potencjometru umieszczonego na płytce X-Nucleo.

Aplikacja co sekundę zwracała informację o aktualnych parametrach, wymienionych powyżej. Tym samym udało się zrealizować postawione założenia projektu. Uruchomienie programu przy podłączonej zaprogramowanej płytce jest wystarczające dla użytkownika końcowego, aby ten mógł uzyskać kontrolę nad silnikiem.

3 Wnioski

W wyniku pracy nad opisanym projektem udało się zrealizować wszystkie postawione wcześniej założenia. Uzyskano łatwą w obsłudze aplikację, pozwalającą na sterowanie silnikiem z poziomu komputera PC.
Zadanie wymagało przeglądu wiedzy na temat sposobów kontroli silników BLDC oraz ich ewaluacji w praktyce. Konieczne było również odpowiednie zorzumienie zagadnień związanych z mikrokontrolerem STM32 oraz z wykorzystanym w projekcie systemem czasu rzeczywistego FreeRTOS
Zrozumienie i praktyczna implementacja zagadnień okazały się wymagające i przysparzały wiele problemów, które jednak udało się skutecznie rozwiązać, zyskując tym samym cenne doświadczenie.
Projekt ma potencjał na dalszy rozwój, polegający np. na dodaniu enkodera lub wykorzystaniu bardziej zaawansowanych algorytmów sterujących - np. sterowania FOC (Field Oriented Current). Pole do poprawy jest również w przypadku interfejsu sterującego - użyteczną funkcjonalnością mogłaby być zamiana przyciskóœ na suwaki, pozwalające od razu ustawić pożądaną przez użytkownika wartość.