

# POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI

INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ

ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ



PROJEKT ZALICZENIOWY - WSKAŹNIK POZYCJI  
KĄTOWEJ

SYSTEMY MIKROPROCESOROWE

RAPORT LABORATORYJNY

KACPER KRASIŃSKI, 151234

KACPER.KRASINSKI@STUDENT.PUT.POZNAN.PL

BARTOSZ KUREK, 151188

BARTOSZ.KUREK@STUDENT.PUT.POZNAN.PL

PROWADZĄCY:

MGR INŻ. ADRIAN WÓJCIK

ADRIAN.WOJCIK@PUT.POZNAN.PL

29 STYCZNIA 2024



## Spis treści

<b>1 Opis Projektu</b>	<b>3</b>
1.1 Logika programu	3
1.2 Schemat połączeń	4
<b>2 Weryfikacja założeń</b>	<b>5</b>
2.1 Pomiar zmiennej stabilizowanej	5
2.2 Stabilizacja obiektu	5
2.3 Github	6
2.4 Doxygen	7
2.5 Aplikacja Desktopowa	7
2.6 MATLAB - symulacje/ skrypty	8
2.7 Wyświetlacz LCD	8
2.8 Dodatkowe wejście potencjometr/enkoder	9
2.9 System plików na karcie SD	9
2.10 Komunikacja sieciowa poprzez gniazdo ETHERNET	10
2.11 Wartość uchybu ustalonego wynosząca 1%	10
<b>3 Rozwiązania napotkanych problemów i wnioski</b>	<b>11</b>
3.1 Dodatkowe komponenty	11
1 Silnik krokowy Nema 17	11
2 Sterownik do silnika krokowego	11
3 Enkoder magnetyczny	12
3.2 Przejście $0^\circ \leftrightarrow 360^\circ$	12
3.3 Dobór nastaw regulatora PID	13
<b>Bibliografia</b>	<b>14</b>

## OPIS PROJEKTU

Obiektem sterowania - elementem wykonawczym układu regulacji będzie ramię działające jako wskaźnik kąta obrotu. Sterowanie tego ramienia będzie zaimplementowane poprzez sterownik TB6560 który sygnał będzie wysyłał do silnika Nema 17. Urządzenie pomiarowe to w naszym przypadku IMU (ang. Inertial measurement unit) dzięki któremu uzyskamy informację o aktualnym położeniu końca wahadła, jego prędkości i przyspieszeniu. Najważniejsze dane będą widoczne na wyświetlaczu LCD. Dane dokładniejsze, w tym parametry regulatora i wskaźniki jakościowe będą dostępne w aplikacji desktopowej z kądem będzie istniała możliwość strojenia regulatora i obserwacji przebiegów. Tryb manualny - sterowanie impulsatorem jak również wysyłanie danych wejściowych z komputera będą dostępne. Karta SD będzie zapewniała logi - zbiór najważniejszych danych z układu.

### 1.1 LOGIKA PROGRAMU

Program jest przygotowany do rozbudowywania i implementacji wielu struktur definiowanych jak poniżej:

*Listing 1. Implementacja struktur.*

```
01. /**
02.  * @struct _BUFFER_UARTHandle
03.  * @brief Structure for UART menu handling.
04. */
05. typedef struct {
06.     int8_t active; /*< Indicates if the UART menu is active. */
07.     char rxBuffer[MAX_BUFFER_SIZE]; /*< Receive buffer for UART communication. */
08.     uint8_t rxIndex; /*< Index of the receive buffer. */
09.     Menu_States state; /*< Current state of the menu. */
10.     MenuComs com; /*< Menu command enumeration. */
11.     MenuStrings compStrings; /*< Structure containing completion strings. */
12. } _BUFFER_UARTHandle;
```

Taka konwencja pozwala na łatwe zmiany w innych miejscach programu. Większość logiki znajduje się w pliku "interrupts.c", w którym wykorzystano poniższe przerwania:

- void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)
- void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)
- void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)
- void HAL\_TIM\_IC\_CaptureCallback(TIM\_HandleTypeDef \*htim)

Całość logiki uzależniona jest od zmiennej `hmen.state` która przyjmuje wartości enumeratora `Menu_States`

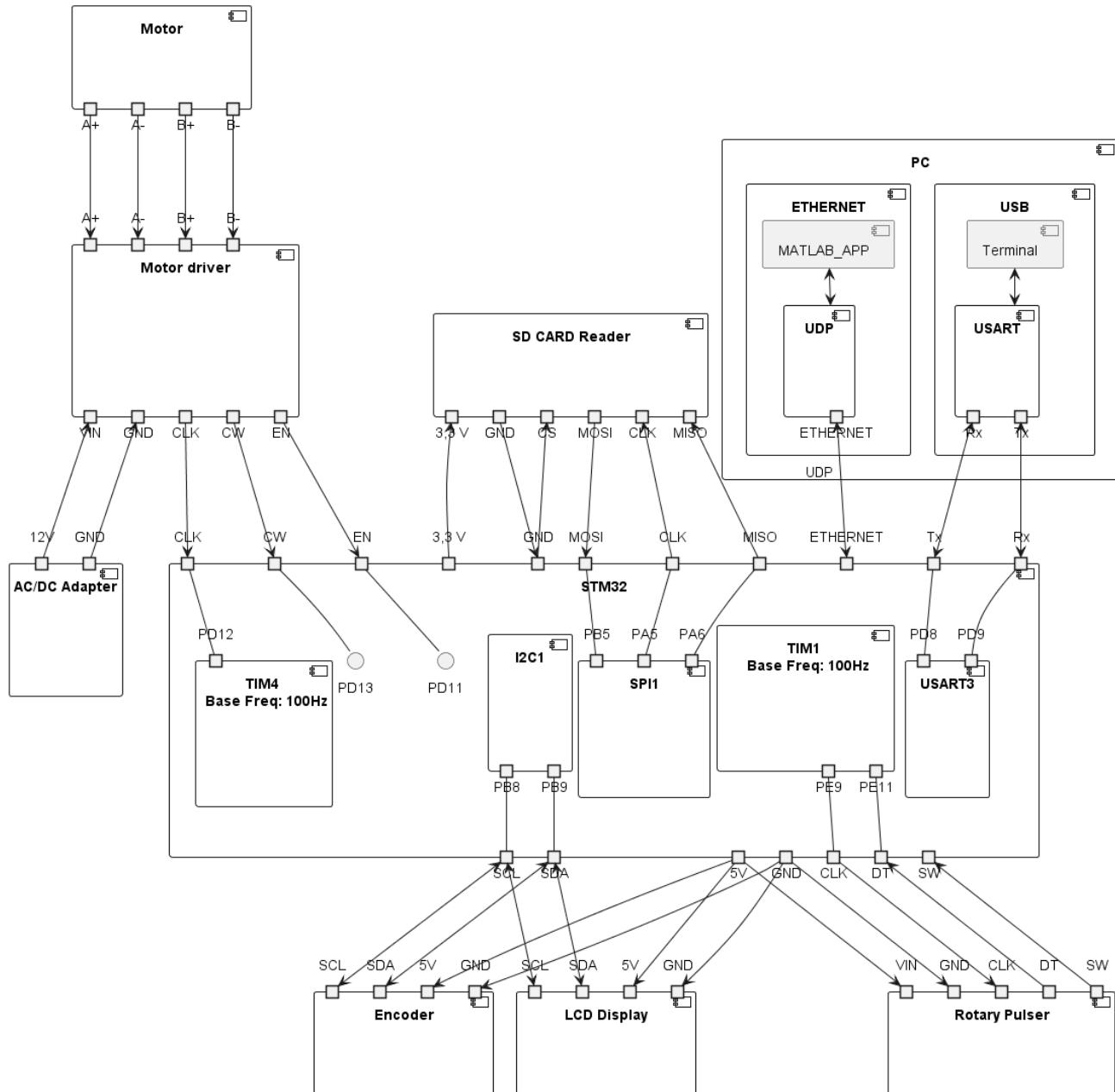
*Listing 2. Enum Menu\_States.*

```
01. typedef enum {
02.     _DEBUG    = 0,      // Debug state
03.     _REMOTE   = 1,      // Remote state
04.     _MANUAL   = 2,      // Manual state
05.     _IDLE     = 9       // Idle state
06. } Menu_States;
```

Wartości te zmieniane są protokołem UART. Dzięki przerwaniom jesteśmy w stanie wykonywać wiele czynności na raz - odczyt do pliku na karcie SD, wyświetlanie na LCD, odczyt z enkodera, komunikacja UART, komunikacja ETHERNET, wysyłanie wartości sterujących na silnik, wyliczanie wartości sterujących na silnik krokowy poprzez sterownik.

## 1.2 SCHEMAT POŁĄCZEŃ

Schemat połączeń wraz z wykorzystanymi interfejsami i pinami został przedstawiony na poniższym grafie.



Rys. 1. Schemat połączeń w układzie

## WERYFIKACJA ZAŁOŻEŃ

### 2.1 POMIAR ZMIENNEJ STABILIZOWANEJ

Jak zostało przedstawione w poleceniu pomiar zmiennej stabilizowanej - kąta obrotu wskaźnika odbywa się ze stałym okresem próbkowania. Możemy tak stwierdzić dzięki wywoływanemu pomiaru przerwaniem Timera o wartościach inicjalizacyjnych:

*Listing 3. Inicjalizacja TIMERów.*

```
01. htim3.Instance = TIM3;
02. htim3.Init.Prescaler = 719;
03. htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
04. htim3.Init.Period = 999;
05. htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
06. htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
```

Zegar podpięty do tego Timera jest o częstotliwości 72MHz co daje częstotliwość wywołania przerwania 100Hz oraz wywołaniu przerwania:

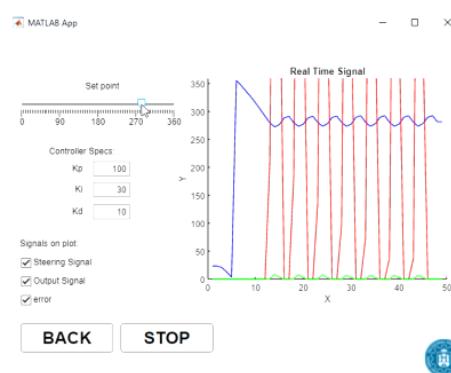
*Listing 4. Przerwanie obsługujące odczyt z enkodera.*

```
01. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
02. {
03.     ...
04.     if (htim->Instance == TIM3)
05.     {
06.         (AS5600_Angle(&henc) != HAL_OK) ? (_Error_Handler(__FILE__, __LINE__))
07.                                         : 1 ;
08.    }
}
```

### 2.2 STABILIZACJA OBIEKTU

Układ stabilizuje się do wartości zadanej poprzez aplikację desktopową - suwak na zrzucie ekranu 10 - tryb REMOTE, bądź za pomocą impulsatora - MANUAL. W tych trybach za sterowanie odpowiada regulator PID, którego implementacja została przedstawiona w sekcji 3.2. Regulacja odbywa się w bezpiecznym zakresie - od 0 do 360 stopni inne wartości przechodzą saturację do tych parametrów. Dobór nastaw regulatora został opisany w sekcji 3.3. Poniżej przykładowy przebieg dla wzmacnienia jak na zrzucie ekranu.

- **sygnał niebieski** - aktualny kąt wskaźnika
- **sygnał czerwony** - aktualna wartość sygnału sterującego
- **sygnał zielony** - aktualny uchyb



*Rys. 2. Przebieg odpowiedzi układu zamkniętego*

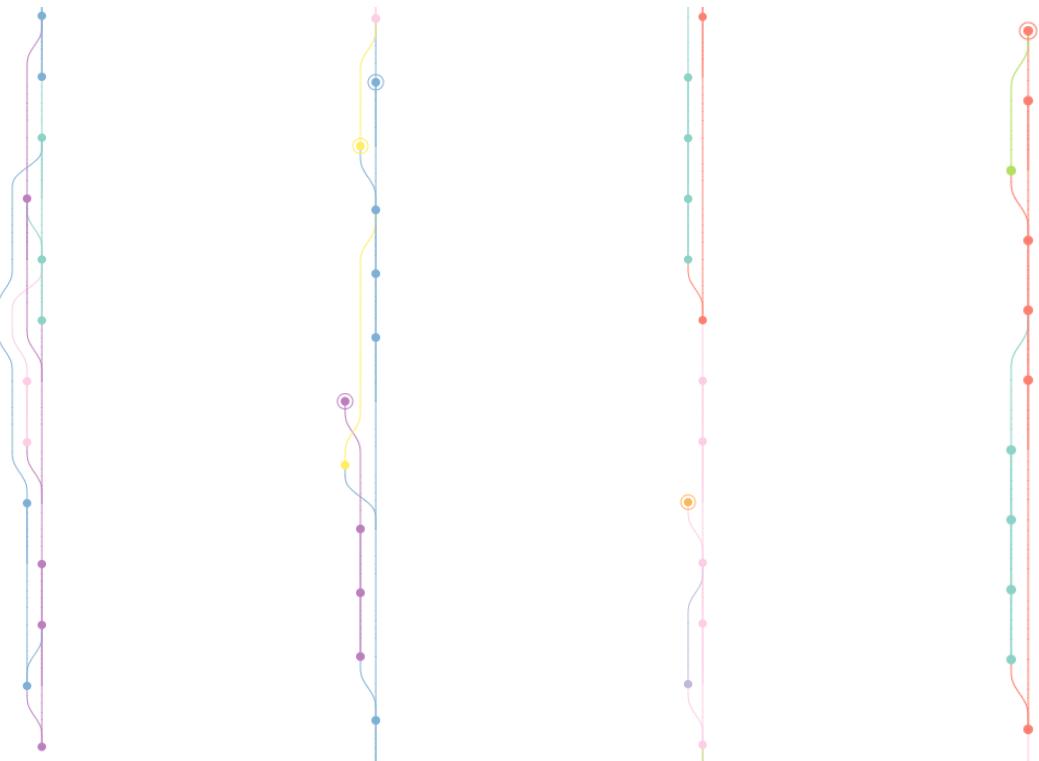
Sygnal czerwony i zielony nie jest wyświetlany podczas trybu DEBUG. W tym trybie do stabilizacji - oscylacji w okół punktu zadaneego wykorzystywany jest regulator dwupołożeniowy:

Listing 5. Implementacja regulatora dwupołożeniowego.

```
01. if (hmen.state == _DEBUG)
02. {
03.     if (((hpsr.set_angle > henc.angle) && (hpsr.set_angle - henc.angle <
04.           180)) ||
05.         ((hpsr.set_angle < henc.angle) && (henc.angle - hpsr.
06.           set_angle > 180)))
07.     {
08.         (MOTOR_SET_INCREASE(&hmtr) != HAL_OK) ? (_Error_Handler(__FILE__,
09.           __LINE__)): 1 ;
10.     }
11.   }
12. }
```

## 2.3 GITHUB

Jak zamierzono projekt był realizowany z wykorzystaniem systemu kontroli wersji. Podgląd na projekt/kod źródłowy i tworzone branche znajduje się pod poniższym linkiem <https://github.com/Krasa35/inverted-pendulum>. Poniżej również podgląd na rozgałęzienia branchy i zarządzanie wersjami. Praca była wykonywana zarówno w tym samym czasie jak i popołudniami, a zatem merge/dodawanie osobnych branchy było różnorodnie udostępniane. Każdy z nas korzystał z innego środowiska: Git bash oraz Github Desktop co pozwoliło poznać zalety i wady kazdego z nich.



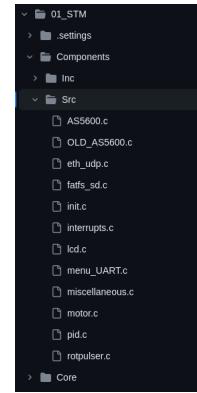
Rys. 3. 1. część drzewka Rys. 4. 2. część drzewka Rys. 5. 3. część drzewka Rys. 6. 4. część drzewka

## 2.4 DOXYGEN

Projekt został napisany z myślą o implementacji standardu komentarzy Doxygen, niektóre opisy oraz wykazy parametrów wejściowych/wyjściowych zostały wygenerowane przez sztuczną inteligencję [1]. Przykładowy komentarz do funkcji oraz zawartość folderu Components - podział programu na pliki znajduje się poniżej:

*Listing 6. Przykładowy komentarz w standardzie Doxygen.*

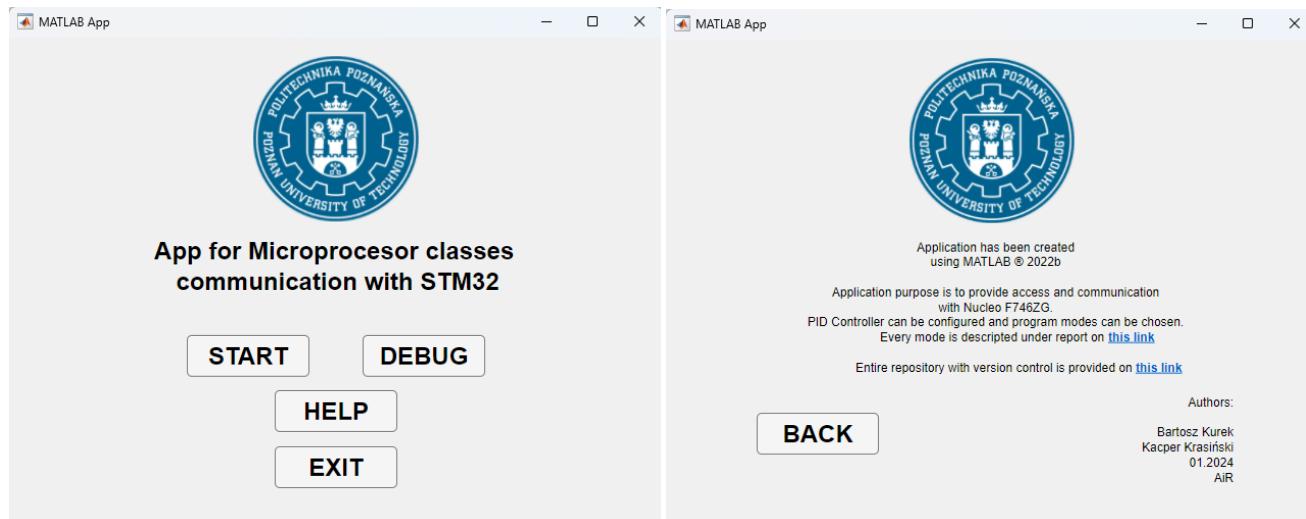
```
01. /**
02.  * @brief Callback for periodic timer
03.  *        interrupts
04.  * @param htim Pointer to the
05.  *        TIM_HandleTypeDef structure
06.  */
07. void HAL_TIM_PeriodElapsedCallback(
08.     TIM_HandleTypeDef *htim)
{ ... }
```



*Rys. 7. Struktura folderów*

## 2.5 APLIKACJA DESKTOPOWA

Aplikacja stworzona w środowisku MATLAB App Designer [2] przysporzyła sporo wyzwań związanych z cyklicznym, oraz stałym odbiorem pakietów. Aplikacja w pełni komunikuje się protokołem UDP, tak że w końcowym etapie pracy wykonywaliśmy osobno nad aplikacją - komputer połączony złączem ETHERNET i nad kodem mikroprocesora - komputer połączony interfejsem UART poprzez USB. Poniżej zrzuty ekranu z aplikacji. Aplikacja pozwala na zmianę parametrów używanych w regulatorze PID zaimplementowanym z biblioteki ARM CMSIS [3]. Wartość zadana również może być przekazywana z poziomu aplikacji.

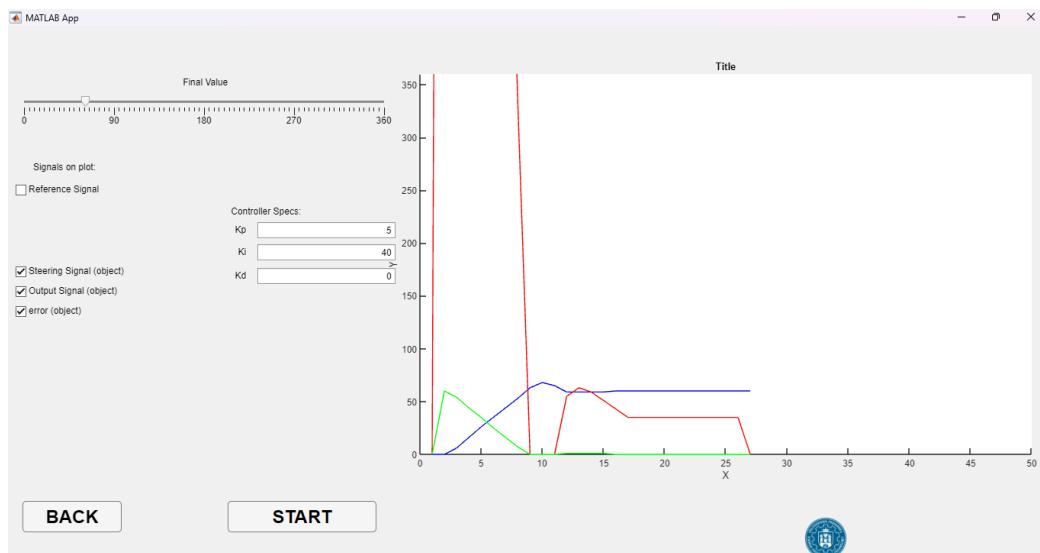


*Rys. 8. Główne okno aplikacji*

*Rys. 9. Okno informacyjne w aplikacji*

## 2.6 MATLAB - SYMULACJE / SKRYPTY

W aplikacji wykorzystano przetwarzanie odebranych danych w macierzach i wykonano interaktywne wykresy, które przedstawiają aktualną pozycję kątową wskaźnika, sygnał sterujący z regulatora PID zaimplementowanego w STM32 oraz uchyb między wartością zadaną a aktualną. Połączenie z aplikacją odbywa się za pomocą protokołu UDP poprzez wyjście ETHERNET płytki NUCLEO.



Rys. 10. okno sterowania aplikacji

## 2.7 WYSWIETLACZ LCD

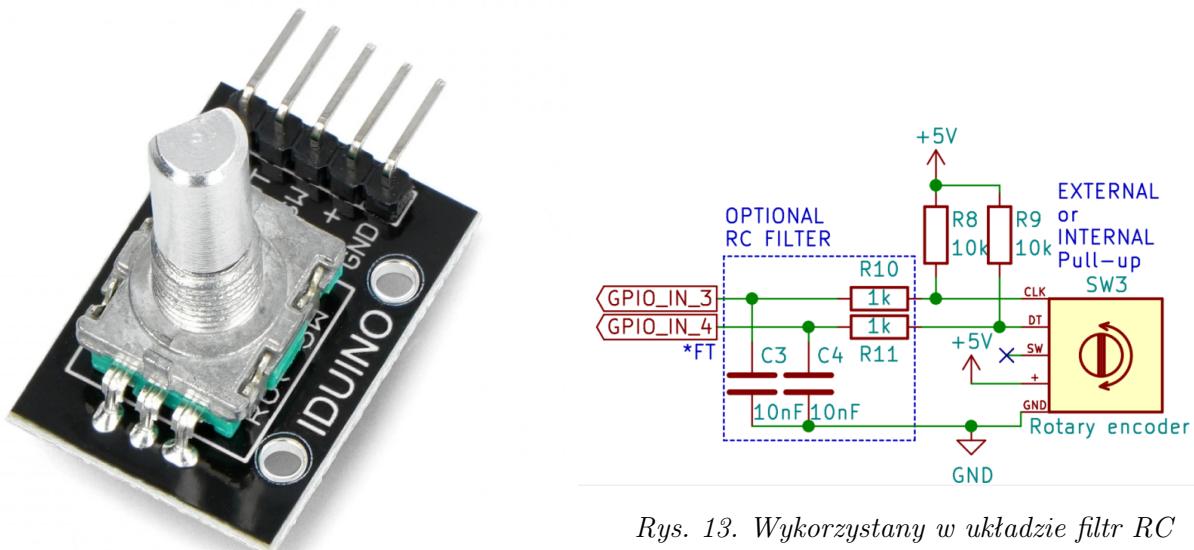
Na wyświetlaczu LCD [4] wyświetlono wartość zadaną, oraz aktualną pozycję wskaźnika. Jest również informacja o trybie w jakim działa silnik. Wyświetlacz jest odświeżany z poziomu płytka STM z częstotliwością 100Hz. Prawdopodobnie jednak taka częstotliwość odświeżania nie jest dostępna sprzętowo, a więc pracuje on z najszybszym możliwym odświerzaniem ekranu. Połączenie z wyświetlaczem LCD odbywa się za pomocą interfejsu I<sup>2</sup>C dzięki konwerterowi dla wyświetlacza [5]. Poniżej przykładowy stan wyświetlacza.



Rys. 11. Wyświetlacz LCD wykorzystany w projekcie

## 2.8 DODATKOWE WEJŚCIE POTENCJOMETR/ENKODER

Dodatkowe wejście dzięki potencjometrowi [6] i przyciskowi bedzie pozwalało nam zmienić pozycję ramienia za pomocą zmiany pozycji potencjometru i wciśnięcia przycisku. Aby zredukować szумy i niekontrolowane skoki wartości dołożono filtr RC użyty podczas laboratorium.

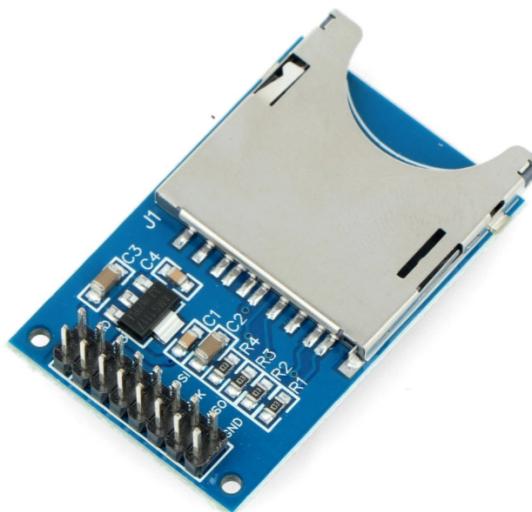


Rys. 13. Wykorzystany w układzie filtr RC

Rys. 12. Impulsator kwadraturowy Iduino SE055 wykorzystany w projekcie

## 2.9 SYSTEM PLIKÓW NA KARCIE SD

Na karcie SD zapisywane są wartości kontrolera PID a także aktualny tryb układu za pomocą czytnika [7]. Do zapisu tych wartości skorzystano z bibliotek `fatfs_sd.h` [8]. Połączenie z czytnikiem odbywa się za pomocą interfejsu SPI. Przykładowy plik załączono poniżej.



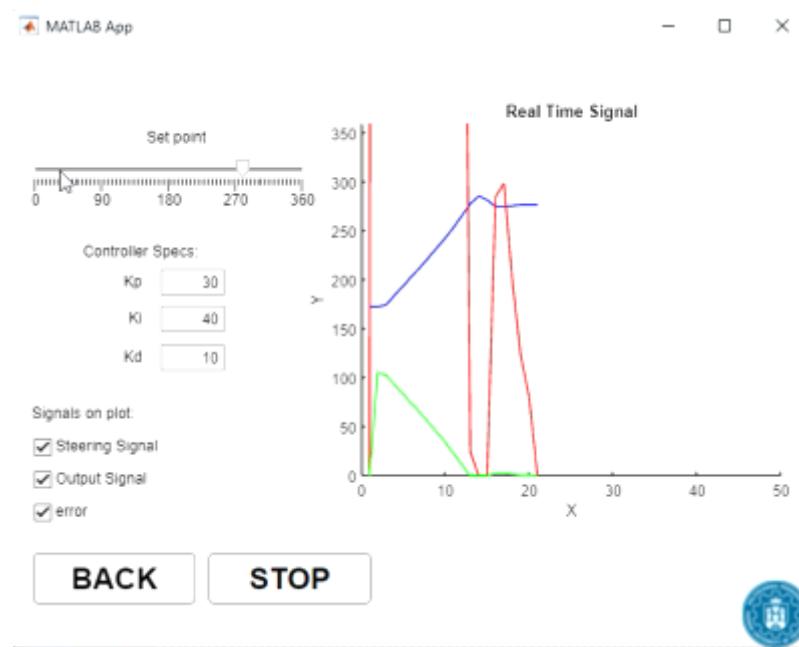
Rys. 14. Czytnik kart SD wykorzystany w projekcie

## 2.10 KOMUNIKACJA SIECIOWA POPRZEZ GNIAZDO ETHERNET

STM32 komunikuje się z komputerem poprzez protokół UDP. Aplikacja odbiera i wysyła pakiety za pomocą obiektu tworzonego funkcją `udpprot()`. Struktura wysyłanych danych z aplikacji to "y%f, p%f, i%f, d%f", gdzie wartości oznaczone jako %f to kolejno: wartość zadana, wartość współczynnika Kp, wartość współczynnika Ki, wartość współczynnika Kd A przyjmowane wartości z enkodera oraz dane obliczeniowe z mikroprocesora w strukturze "%d, %d, %d", gdzie wartości oznaczone jako %d to kolejno: aktualnie wskażany kąt - wartość z enkodera, wartość sygnału sterującego, oraz wartość uchybu - wartość z enkodera minus wartość zadana uwzględniając fakt opisany w sekcji 3.3. Realizacja połączenia po stronie mikrokontrolera odbywała się za pomocą LwIP, którego implementacja była analogiczna do wykładu Doktora habilitowanego Dominika Łuczaka [9].

## 2.11 WARTOŚĆ UCHYBU USTALONEGO WYNOSZĄCA 1%

Układ jest stabilny i dochodzi do zadanego kąta z uchybem ustalonym mniejszym niż 1%, dla niektórych parametrów jest to dokładność nawet do 0.3%. Zależy to od ustawienia DIP Switchy znajdujących się na sterowniku do silnika. Na poniższym wykresie widać że uchyb mieści się w podanym zakresie.



Rys. 15. Przykład regulacji o uchybie ustalonym poniżej 1%

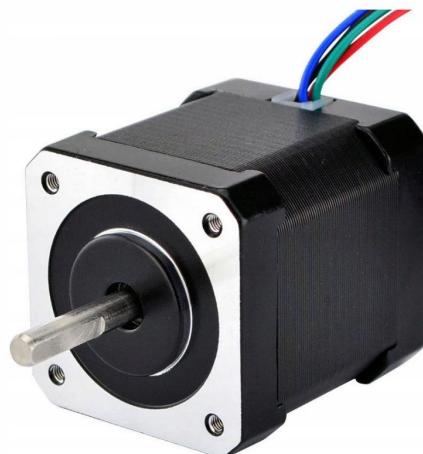
## ROZWIAZANIA NAPOTKANYCH PROBLEMÓW I WNIOSKI

### 3.1 DODATKOWE KOMPONENTY

Dodatkowo poza wymienionymi komponentami dodatkowymi do wykonania podstawowego zakresu zadania regulacji potrzebowaliśmy:

#### 1 SILNIK KROKOWY NEMA 17

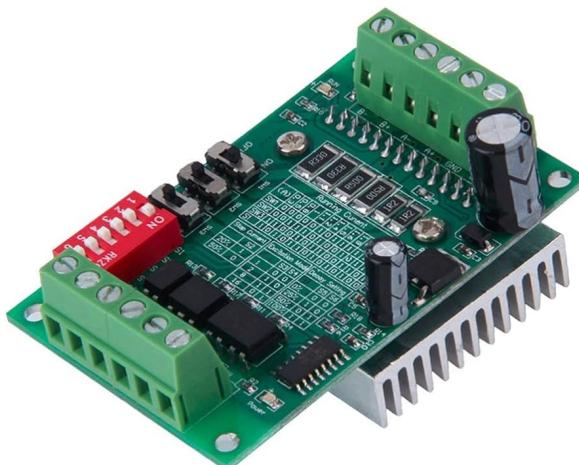
Silnik sterujący wskaźnikiem pozycji kątowej wydrukowanym na drukarce 3D.



Rys. 16. Silnik krokowy NEMA17

#### 2 STEROWNIK DO SILNIKA KROKOWEGO

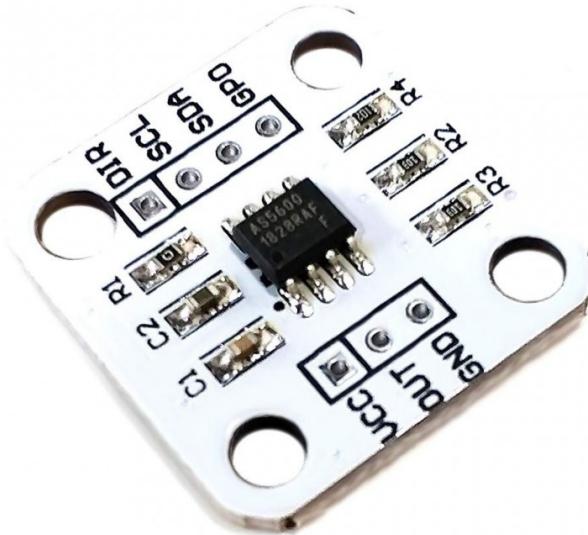
Do sterowania silnika potrzebowaliśmy sterownika [10] by przesyłać prędkość silnika modulując częstotliwość sygnału PWM wysyłanego z STM32.



Rys. 17. Sterownik do silnika krokowego

### 3 ENKODER MAGNETYCZNY

Do odczytu wartości aktualnego położenia wskaźnika posłużył nam enkoder magnetyczny AS5600 [11]. Połączenie z enkoderem odbywa się za pomocą interfejsu I<sup>2</sup>C.



Rys. 18. Enkoder magnetyczny wykorzystany w układzie

#### 3.2 PRZEJŚCIE $0^\circ \leftrightarrow 360^\circ$

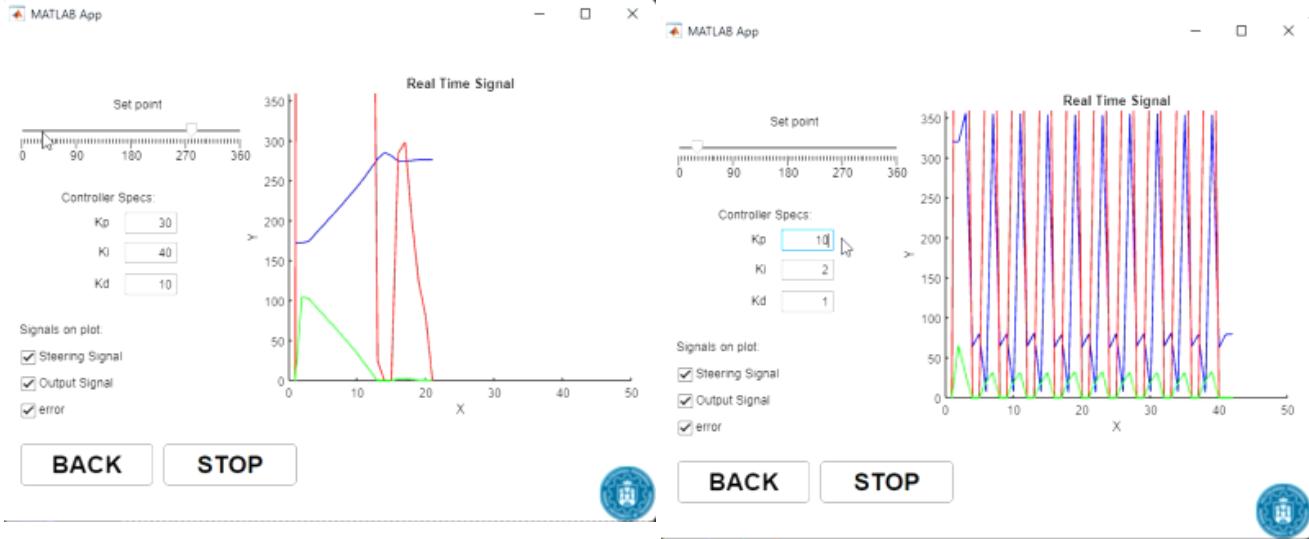
Z powodu, że wartość zadana sygnału sterującego może zawierać się w przedziale od  $0^\circ$  do  $360^\circ$  regulator nie uwzględnia tego że dojście do wartości kąta  $180^\circ + \alpha$  z wartości  $\alpha - \beta$ , dla  $0^\circ < \alpha < 180^\circ$  i  $0^\circ < \beta < 180^\circ$ , gdzie  $\alpha > \beta$  szybsze jest zwiększać wartość kąta i przechodząc przez  $360^\circ$  niż zmniejszając wartość co spowoduje że droga dojścia będzie dłuższa. Rozwiązano to za pomocą poniższych warunków. Pokazano również w jaki sposób należy zmienić w takim przypadku uchyb zadawany na regulator oraz całą implementację przetwarzania informacji z regulatora PID.

Listing 7. Implementacja regulatora PID.

```
01. HAL_StatusTypeDef PID_manualProcess(_PID_HandleTypeDef* pid, _MOTOR_HandleTypeDef* mtr){  
02.     pid->error = pid->setpoint - pid->current_angle;  
03.     if (pid->error > 0) { (pid->error < 180) ? 1:(pid->error = pid->error -  
04.         360);}  
05.     if (pid->error < 0) { (pid->error < -180) ? (pid->error = pid->error +  
06.         360):1;}  
07.     arm_pid_init_f32(&pid->controller, 1);  
08.     pid->output = arm_pid_f32(&pid->controller, pid->error);  
09.     GPIO_PinState dir = (pid->output < 0) ? (GPIO_PIN_SET):(GPIO_PIN_RESET);  
10.     (MOTOR_Direction(mtr, dir) != HAL_OK) ? (_Error_Handler(__FILE__,  
11.         __LINE__)): 1 ;  
12.     (MOTOR_FindFrequency(mtr, pid->output) != HAL_OK) ? (_Error_Handler(  
13.         __FILE__, __LINE__)): 1 ;  
14.     if (pid->at_dest) { (MOTOR_Set_ENABLE(mtr) != HAL_OK) ? (_Error_Handler(  
15.             __FILE__, __LINE__)): 1 ;  
16.         pid->at_dest = 1;  
17.     }  
18.     return HAL_OK;  
19. }
```

### 3.3 DOBÓR NASTAW REGULATORA PID

Nastawy regulatora dobrano w sposób przedstawiony na nagraniu [12]. Zmiany kroku na DIP switchach na sterowniku silnika krokowego



Rys. 19. Przebiegi dojścia do wartości zadanej z przeregulowaniem - regulator PID,  $K_p = 30, K_i = 40, K_d = 10$   
Rys. 20. Nastawy regulatora na granicy stabilności,  $K_p = 10, K_i = 2, K_d = 1$

Poniżej natomiast przedstawiono przebieg w momencie wytrącenia obiektu ze stanu ustalonego w trybie regulacji PID.



Rys. 21. Przebiegi dla wytracenia ze stanu ustalonego

## BIBLIOGRAFIA

1. OPENAI. *GPT-3.5: Language Model by OpenAI*. 2022. Dostępne także z: <https://www.openai.com>. Generated by ChatGPT, an AI language model developed by OpenAI.
2. INC., The MathWorks. *MATLAB version: 9.13.0 (R2022b)*. Natick, Massachusetts, United States: The MathWorks Inc., 2022. Dostępne także z: <https://www.mathworks.com>.
3. ARM-SOFTWARE. *Github z biblioteką ARM CMSIS* [Resources]. Dostępne także z: [https://github.com/ARM-software/CMSIS\\_5.git](https://github.com/ARM-software/CMSIS_5.git).
4. *Wyświetlacz LCD* [Component]. Dostępne także z: <https://botland.com.pl/wyswietlacze-alfanumeryczne-i-graficzne/19738-wyswietlacz-lcd-2x16-znakow-zielony-justpi-5903351243063.html>.
5. *Konwerter I2C dla wyświetlacza LCD* [Component]. Dostępne także z: <https://botland.com.pl/konwertery-pozostale/2352-konwerter-i2c-dla-wyswietlacza-lcd-hd44780-5903351248693.html>.
6. *Impulsator kwadraturowy* [Component]. Dostępne także z: <https://botland.com.pl/enkodery/14273-czujnik-obrotu-impulsator-enkoder-z-przyciskiem-iduino-se055-5903351242042.html>.
7. *Moduł czytnika kart SD, IDUINO SE055* [Component]. Dostępne także z: <https://botland.com.pl/akcesoria-do-kart-pamieci/1507-modul-czytnika-kart-sd-5903351241342.html>.
8. EZIYA. *Github z biblioteką do czytnika kard SD* [Resources]. Dostępne także z: [https://github.com/eziya/STM32\\_SPI\\_SDCARD.git](https://github.com/eziya/STM32_SPI_SDCARD.git).
9. ŁUCZAK, Dominik. *Wykład Lightweight TCP/IP stack* [Resources]. Dostępne także z: [https://ekursy.put.poznan.pl/pluginfile.php/2646900/mod\\_folder/content/0/W10\\_SM%20-%20LwIP.pdf?forcedownload=1](https://ekursy.put.poznan.pl/pluginfile.php/2646900/mod_folder/content/0/W10_SM%20-%20LwIP.pdf?forcedownload=1).
10. *Sterownik do silnika krokowego* [Component]. Dostępne także z: <https://allegro.pl/oferta/tb6560-sterownik-silnika-krokowego-kompletny-cnc-7023134764>.
11. *Enkoder magnetyczny* [Component]. Dostępne także z: <https://kamami.pl/enkodery/580650-modul-enkodera-magnetycznego-z-ukladem-as5600.html>.
12. BARTOSZ KUREK, Kacper Krasiński. *Filmik z przedstawieniem działania programu* [Resources]. Dostępne także z: <https://youtu.be/mLq4kaAIWEw>.