

Graficzne programowanie mikrokontrolerów

M5Stack / UIFlow

Laboratorium 6

Sterowanie napędami

Czas trwania: 2 × 45 min**Tryb:** studia stacjonarne**Jednostka:** Katedra Elektrotechniki i Technologii Nadprzewodnikowych**Prowadzący:** dr inż. Michał Łanczont

Cel ćwiczenia

Celem ćwiczenia jest poznanie podstaw sterowania trzema rodzajami napędów dostępnymi na Demo Board: serwomechanizmem, silnikiem DC i silnikiem krokowym. Student wykorzystuje PWM, sterowanie kierunkiem i prędkością, pozycjonowanie krokowe oraz implementuje bezpieczny stan zatrzymania.

Wstęp

Serwomechanizm jest sterowany wartością odpowiadającą żadanemu położeniu. Silnik DC wymaga określenia kierunku i poziomu sterowania prędkością. Silnik krokowy wykonuje ruch jako sekwencję kroków, dzięki czemu możliwe jest pozycjonowanie względne.

Nie należy utożsamiać tych trzech metod sterowania. Interfejs operatora może być podobny, ale algorytm sterowania napędem jest inny.

Stan bezpieczny

Każdy program sterujący napędem musi posiadać jednoznaczny stan ZATRZYMANIE. Po uruchomieniu programu napęd nie powinien rozpocząć ruchu bez polecenia użytkownika. Utrata sterowania lub błąd powinny prowadzić do zatrzymania, o ile dany układ wykonawczy na to pozwala.

Bezpieczeństwo

Nie dotykaj ruchomych elementów podczas pracy. Przed zmianą przewodów wyłącz zasilanie odpowiedniego modułu. Nie blokuj mechanicznie wału silnika i nie wymuszaj ruchu serwa poza jego dopuszczalnym zakresem. **Nie wymuszaj ręcznie ruchu serwomechanizmu!!!**

Przebieg ćwiczenia

Tor sterowania napędem

polecenie operatora → ograniczenie i walidacja → sterownik napędu → napęd → stan/diagnostyka

Zadanie 1. Serwomechanizm

Steruj pozycją serwa wartością 0–100, przeliczaną na bezpieczny zakres kątowy ustalony przez prowadzącego. Użyj enkodera do zmiany nastawy. Dodaj ograniczenie programowe zapobiegające przekroczeniu zakresu.

Sposób realizacji i weryfikacja

1. Ustal dozwolony zakres mechaniczny przed rozpoczęciem ciągłego sterowania; nie testuj skrajnych kątów bez kontroli stanowiska.
2. Przelicz 0–100 na zakres kątowy i zawsze zastosuj ograniczenie przed wysłaniem polecenia do serwa.
3. Zmieniaj pozycję małymi krokami enkodera i obserwuj, czy serwo nie dociska mechanizmu do ogranicznika.

Pseudokod / zapis algorytmu:

```
kat = mapuj(nastawa, 0, 100, kat_min, kat_maks)
kat = ogranicz(kat, kat_min, kat_maks)
ustaw_serwo(kat)
```

Zadanie 2. Pozycje zapamiętane

Zdefiniuj trzy pozycje serwa: MIN, SRODEK i MAKS. Przypisz je do przycisków A/B/C. Na ekranie pokazuj aktualną wartość zadaną i nazwę pozycji.

Sposób realizacji i weryfikacja

1. Zdefiniuj MIN/SRODEK/MAKS jako stałe nastawy, a nie trzy osobne sekwencje sterowania.
2. Po naciśnięciu przycisku najpierw ustaw wartość zadaną, potem wykorzystaj tę samą funkcję do wysłania polecenia i odświeżenia GUI.
3. Sprawdź przejścia między każdą parą pozycji; w razie potrzeby zastosuj łagodną zmianę zamiast skoku.

Pseudokod / zapis algorytmu:

```
pozycja_zapisana(przycisk):
    nastawa = POZYCJE_ZAPISANE[przycisk]
    ustaw_serwo_z_nastawy(nastawa)
```

Zadanie 3. Silnik DC – kierunek

Przygotuj trzy stany: LEWO, ZATRZYMANIE, PRAWO. Zmiana kierunku musi przechodzić przez ZATRZYMANIE. Nie przełączaj bezpośrednio z pełnej prędkości w jednym kierunku na pełną prędkość w drugim.

Sposób realizacji i weryfikacja

1. Reprezentuj polecenie napędu znakiem: wartość ujemna LEWO, 0 ZATRZYMANIE, dodatnia PRAWO.
2. Jeżeli znak nowego polecenia jest przeciwny do poprzedniego, najpierw ustaw PWM=0, odczekaj krótki czas i dopiero zmień kierunek.
3. ZATRZYMAJ ma mieć priorytet nad innymi poleceniami i natychmiast zerować wartość wysterowania.

Pseudokod / zapis algorytmu:

```
if znak(nowe_polecenie) != znak(poprzednie_polecenie) and poprzednie_polecenie != 0:
    pwm = 0
    czekaj_ms(czas_bezpiecznej_zmiany)
ustaw_kierunek(znak(nowe_polecenie))
ustaw_pwm(abs(nowe_polecenie))
```

Zadanie 4. Silnik DC – joystick i prędkość

Wykorzystaj oś X joysticka do sterowania kierunkiem i wartością prędkości od -100 do $+100$. Dodaj martwą strefę wokół zera. Na GUI prezentuj kierunek, wartość zadaną i stan napędu.

Sposób realizacji i weryfikacja

1. Użyj znormalizowanej osi X z laboratorium analogowego; zastosuj martwą strefę przed wyznaczeniem kierunku.
2. Przelicz wartość bezwzględną na PWM, a znak wykorzystaj wyłącznie do wyboru kierunku.
3. Na GUI pokazuj jednocześnie polecenie $-100...100$, kierunek oraz rzeczywisty stan ZATRZYMANIE/PRACA.

Pseudokod / zapis algorytmu:

```
polecenie = wychylenie_x
if abs(polecenie) < martwa_strefa: polecenie = 0
kierunek = znak(polecenie)
pwm = abs(polecenie)
```

Podstawowe algorytmy sterowania silnikiem krokowym

Silnik krokowy na płycie Demo Board jest silnikiem **czterofazowym, pięcioprzewodowym**. Poszczególne uzwojenia są załączane przez układ sterownika LV8548MC. W przeciwieństwie do typowego sterownika STEP/DIR, program musi określać **kolejność pobudzania faz A, B, C i D**. Kolejność ta decyduje o kierunku ruchu, a czas pomiędzy kolejnymi stanami - o prędkości obrotowej. Demo Board wykorzystuje właśnie taki czterofazowy silnik krokowy sterowany przez LV8548MC.

Idea sterowania

Jeden krok algorytmu polega na ustawieniu odpowiedniego stanu czterech faz silnika, odczekaniu zadanego czasu i przejściu do kolejnego stanu sekwencji. Zmiana kierunku polega na odwróceniu kolejności przechodzenia przez sekwencję.

1. Sterowanie pełnokrokowe - jedna faza aktywna

Najprostsza sekwencja sterowania polega na załączaniu kolejno jednej fazy. Daje mały pobór prądu, ale również mniejszy moment obrotowy.

Krok	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Pseudokod:

```
sekwencja = [1000, 0100, 0010, 0001]
indeks = 0

dopoki liczba_krokov > 0:
    ustaw_fazy(sekwencja[indeks])
    czekaj_ms(opoznienie)
    indeks = (indeks + kierunek) modulo 4
    liczba_krokov = liczba_krokov - 1
```

2. Sterowanie pełnokrokowe - dwie fazy aktywne

Włączenie dwóch sąsiednich faz jednocześnie zwiększa moment trzymający i moment podczas ruchu. Typowa sekwencja ma postać:

Krok	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Pseudokod:

```
sekwencja = [1100, 0110, 0011, 1001]
wykonaj_kroki(liczba_krokow, kierunek, opoznienie, sekwencja)
```

3. Sterowanie półkrokowe

Sterowanie półkrokowe przeplata stany z jedną i dwiema aktywnymi fazami. Powoduje to zwiększenie liczby pozycji pośrednich, a więc zmniejszenie kąta pojedynczego kroku i uzyskanie płynniejszego ruchu.

Krok	1	2	3	4	5	6	7	8
A	1	1	0	0	0	0	0	1
B	0	1	1	1	0	0	0	0
C	0	0	0	1	1	1	0	0
D	0	0	0	0	0	1	1	1

Pseudokod:

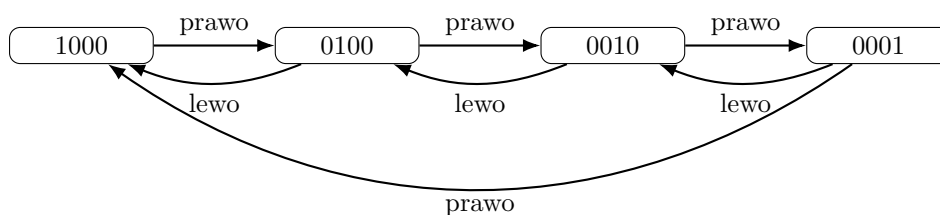
```
sekwencja = [1000, 1100, 0100, 0110,
              0010, 0011, 0001, 1001]
indeks = nastepny_indeks(indeks, kierunek, 8)
ustaw_fazy(sekwencja[indeks])
```

Kolejność faz

Podane sekwencje zakładają kolejność wyprowadzeń A–B–C–D zgodną z oznaczeniami modułu. Jeżeli rzeczywista kolejność przewodów silnika jest inna, silnik może drgać, zatrzymywać się albo obracać w inną stronę. W takim przypadku należy sprawdzić przyporządkowanie faz przed zwiększaniem prędkości.

4. Zmiana kierunku obrotów

Nie jest potrzebna osobna sekwencja dla drugiego kierunku. Wystarczy przechodzić przez tę samą tablicę w odwrotnej kolejności.



Pseudokod zmiany indeksu:

```

jezeli kierunek == PRAWO:
    indeks = (indeks + 1) modulo liczba_stanow
w_przeciwnym_razie:
    indeks = (indeks - 1 + liczba_stanow) modulo liczba_stanow

```

5. Sterowanie prędkością

Prędkość silnika krokowego zależy przede wszystkim od odstępu czasu pomiędzy kolejnymi krokami. Im krótszy czas oczekiwania, tym większa częstotliwość kroków i większa prędkość obrotowa.

$$f_k = \frac{1}{T_k}$$

Jeżeli silnik wykonuje N_k pełnych kroków na jeden obrót, prędkość obrotową można oszacować z zależności:

$$n = \frac{60 f_k}{N_k} \quad [\text{obr/min}]$$

Pseudokod:

```

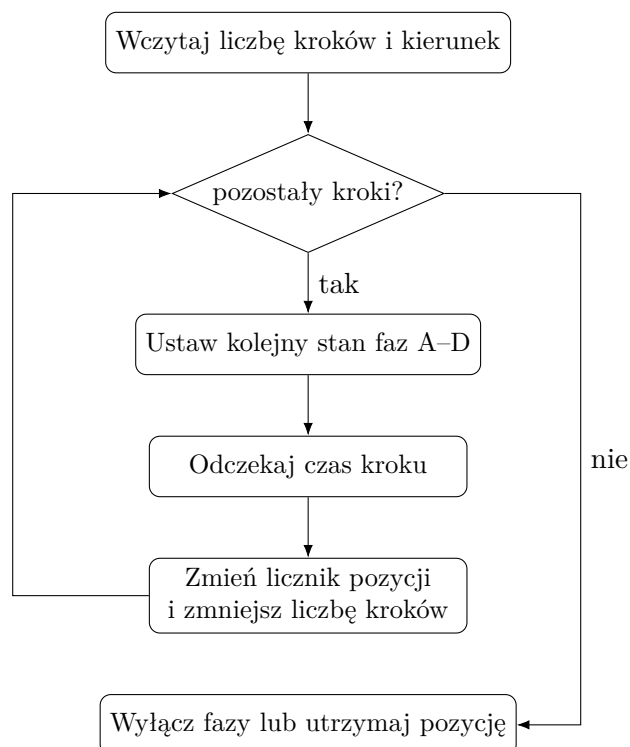
opoznienie = mapuj(nastawa_predkosci,
                   0, 100,
                   opoznienie_maks, opoznienie_min)

ustaw_fazy(sekwencja[indeks])
czekaj_ms(opoznienie)

```

6. Ruch o zadaną liczbę kroków

Podstawowy algorytm pozycjonowania względnego zlicza wykonane kroki. Kierunek może być zapisany jako +1 lub -1, dzięki czemu ta sama zmienna służy zarówno do wyboru kolejności faz, jak i do aktualizacji pozycji programowej.



Pseudokod:

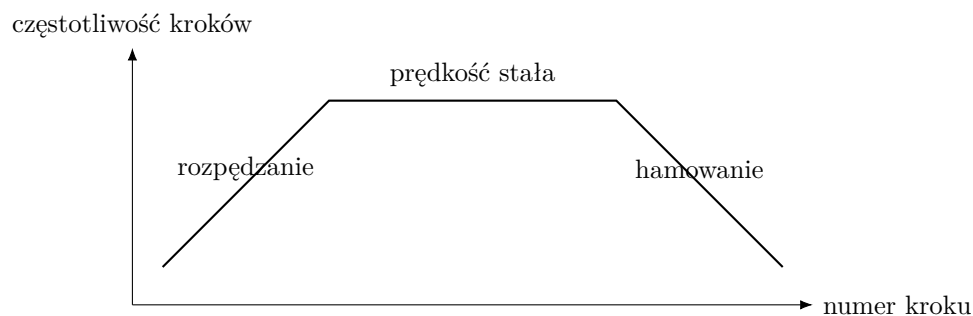
```

wykonaj_ruch(ile_krokow, kierunek):
    powtorz abs(ile_krokow) razy:
        ustaw_nastepny_stan_faz(kierunek)
        czekaj_ms(opoznienie)
        pozycja = pozycja + kierunek

```

7. Łagodne rozpędzanie i hamowanie

Silnik krokowy nie zawsze może rozpocząć ruch od razu z dużą częstotliwością. Przy zbyt gwałtownym starcie moment może być niewystarczający i silnik zgubi kroki. Prostym rozwiązaniem jest zastosowanie rampy: początkowo odstęp między krokami jest duży, następnie stopniowo maleje, a przed zatrzymaniem ponownie rośnie.

**Prosty algorytm rampy:**

```

co kazdy krok:
    jezeli wykonane_kroki < liczba_krokow_rampy:
        opoznienie = opoznienie - zmiana_opoznienia
    jezeli pozostale_kroki < liczba_krokow_rampy:
        opoznienie = opoznienie + zmiana_opoznienia

    ustaw_nastepny_stan_faz(kierunek)
    czekaj_ms(opoznienie)

```

Uwaga dydaktyczna

W prostym ćwiczeniu wystarczy sterowanie ze stałym opóźnieniem. Rampa przyspieszania i hamowania jest rozszerzeniem pokazującym, dlaczego rzeczywiste układy pozycjonujące nie powinny zawsze przełączać silnika natychmiast na maksymalną prędkość.

Zadanie 5. Silnik krokowy

Przygotuj komendy ruchu względnego: -100 , -10 , $+10$, $+100$ kroków oraz logiczną funkcję POZYCJA_0 oznaczającą wyzerowanie licznika pozycji programu. Jeżeli stanowisko nie posiada fizycznego czujnika bazowania, POZYCJA_0 nie oznacza rzeczywistego pomiaru położenia.

Sposób realizacji i weryfikacja

1. Prowadź programowy licznik pozycji i aktualizuj go dopiero po wykonaniu zadanej liczby kroków.
2. Komendy względne mają być kolejgowane albo blokowane podczas trwającego ruchu - nie nadpisuj aktywnego polecenia bez świadomej decyzji.
3. Wyraźnie odróżnij POZYCJA_0=0 programowe od fizycznego bazowania z czujnikiem krańcowym.

Pseudokod / zapis algorytmu:

```

cel = pozycja + zmiana
dopoki pozycja != cel and not zatrzymanie_aktywne:
    wykonaj_krok(kierunek)
    pozycja += kierunek

```

Zadanie 6. Wspólny panel sterowania

Utwórz ekran wyboru napędu. Po wyborze SERWO, DC lub SILNIK KROKOWY pokaż tylko właściwe elementy sterowania. Przycisk ZATRZYMAJ ma być dostępny niezależnie od wybranego napędu.

Sposób realizacji i weryfikacja

1. Przechowuj aktywny napęd w zmiennej `wybrany_naped`; wybór zmienia GUI, ale nie może sam uruchamiać napędu.
2. Wspólny przycisk ZATRZYMAJ wywołuje funkcję zatrzymującą wszystkie trzy napędy, niezależnie od aktualnego ekranu.
3. Po zmianie napędu odśwież tylko odpowiednie kontrolki, ale stale pokazuj globalny stan ZATRZYMANIE/ALARM.

Pseudokod / zapis algorytmu:

```

zatrzymaj_wszystko():
    ustaw_serwo_bezpiecznie()
    zatrzymaj_silnik_dc()
    zatrzymaj_silnik_krokowy()

```

Wymagany rezultat

Student potrafi wskazać różnicę między sterowaniem serwowym, silnikiem DC i silnikiem krokowym, a każdy tryb posiada ograniczenia oraz funkcję bezpiecznego zatrzymania.

Zadanie końcowe**Integracja zadań etapowych**

Zadanie końcowe jest wspólnym rozwinięciem zadań 1–6. Wykorzystaj sterowanie serwowym z zadania 1 i zapamiętane pozycje z zadania 2, bezpieczną zmianę kierunku silnika DC z zadania 3, sterowanie joystickiem z zadania 4, ruch krokowy z zadania 5 oraz wspólny panel z zadania 6. Każdy napęd ma korzystać z wcześniej przygotowanej funkcji sterującej.

Opracuj sterownik trzech napędów z jednym interfejsem operatorskim. Joystick steruje silnikiem DC, enkoder ustala położenie serwa, a przyciski realizują ruch względny silnika krokowego. Zaimplementuj wspólny stan ZATRZYMANIE oraz blokadę ruchu po wejściu w ALARM symulowany przyciskiem.

Rozszerzenie: po skasowaniu alarmu układ pozostaje w ZATRZYMANIE i wymaga ponownego świadomego polecenia ruchu dla wybranego napędu.

Kryteria zaliczenia

- sterowanie położeniem serwa z ograniczeniem zakresu,
- bezpieczna zmiana kierunku silnika DC,

- regulacja prędkości silnika DC,
- sterowanie ruchem względnym silnika krokowego,
- wspólny panel operatorski,
- jednoznaczny stan ZATRZYMANIE/ALARM.

Sprawozdanie

Podstawą zaliczenia ćwiczenia jest sprawozdanie dotyczące wykonania zadania końcowego. Zadania etapowe są zaliczane na zasadzie demonstracji działania programu na stanowisku. Sprawozdanie należy przesłać w formie pliku **.pdf* oraz pliku projektu programu do weryfikacji. W dokumencie należy przedstawić:

- kod programu w UIFlow,
- zdjęcia działającego stanowiska lub czytelne zrzuty ekranu,
- krótki opis zastosowanego algorytmu wraz ze schematem blokowym, diagramem stanów lub pseudokodem,
- opis sposobu testowania programu,
- wnioski.