Projekt Regulatora PID

Wykonali:
Wojciech Wojtkowiak 159760
Maciej Wnuk 161327
Dawid Wawrzyniak 159360

1. Założenia projektowe

Głównym celem projektu jest zaprojektowanie i implementacja mikroprocesorowego układu automatycznej regulacji kąta otwarcia okna w zależności od natężenia oświetlenia zewnętrznego. Układ ma pracować w pętli zamkniętego sprzężenia zwrotnego, wykorzystując algorytm regulacji PID.

Szczegółowe założenia funkcjonalne:
- Pomiar sygnału wejściowego: Ciągły pomiar natężenia światła za pomocą cyfrowego czujnika BH1750 komunikującego się przez interfejs I2C.

- Algorytm regulacji: Zastosowanie regulatora PID miało na celu zapewnienie płynnej, stabilnej oraz możliwie szybkiej reakcji układu na zmiany oświetlenia, przy jednoczesnym ograniczeniu oscylacji oraz minimalizacji uchybu ustalonego.

- Element wykonawczy: Sterowanie kątem wychylenia okna za pomocą serwomechanizmu SG90, wykorzystującego sygnał PWM (Pulse Width Modulation).

- Przycisk: Zatrzymanie działania układu do momentu puszczenia, wtedy układ ustawia się w pozycji 90 stopni i zaczyna od nowa działać.

- Platforma sprzętowa: Wykorzystanie modułu STM32 Nucleo-F756ZG jako jednostki centralnej systemu.

2. Wykorzystane komponenty

W projekcie wykorzystaliśmy następujące elementy sprzętowe:

- czujnik natężenia światła BH1750,
- mikrokontroler STM32 Nucleo 756ZG,
- serwomechanizm SG90,
- przewody połączeniowe i płytka stykowa.

3. Opis wykorzystanych środowisk programistycznych

Do realizacji projektu wykorzystaliśmy pakiet narzędziowy dedykowany dla systemów wbudowanych z rodziny STM32:

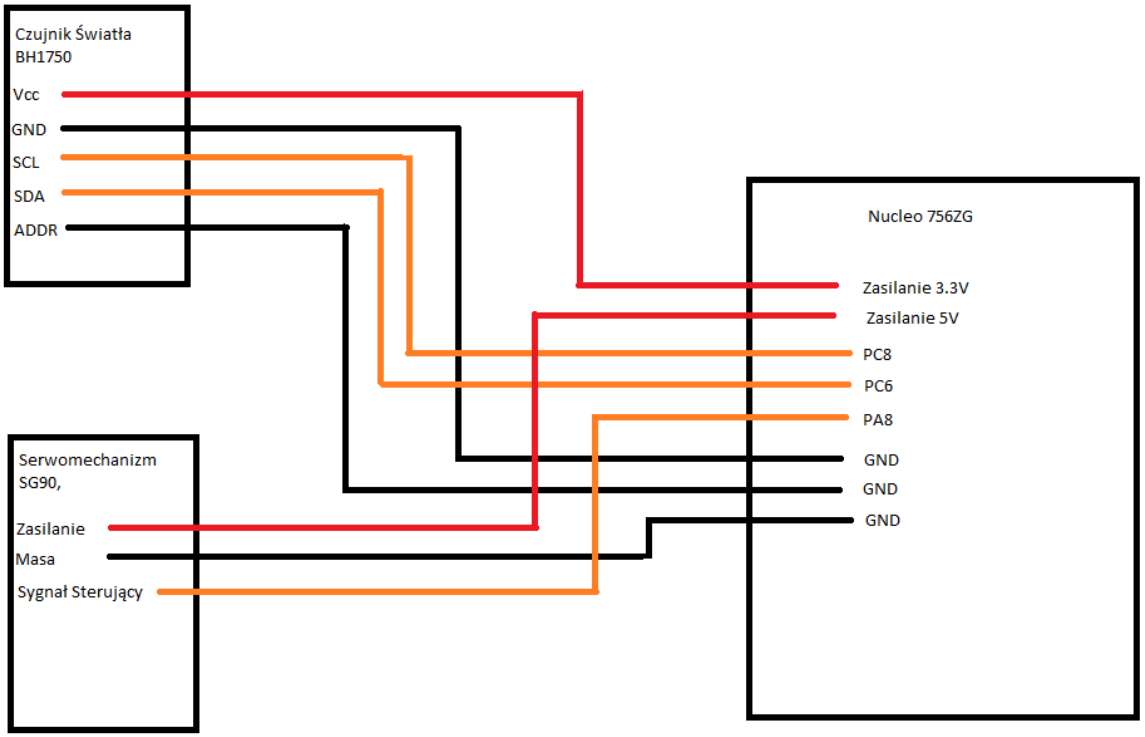
STM32CubeMX - narzędzie graficzne służące do wstępnej konfiguracji mikrokontrolera. Wykorzystaliśmy je do:

- Konfiguracji zegarów systemowych.
- Ustawienia pinów GPIO dla interfejsu I2C (komunikacja z czujnikiem BH1750).
- Konfiguracji Timera w trybie generowania sygnału PWM (sterowanie serwomechanizmem SG90).

STM32CubeIDE - zintegrowane środowisko programistyczne (IDE), w którym:

- Zaimplementowaliśmy algorytm regulatora PID w języku C.
- Przeprowadziliśmy proces kompilacji i debugowania kodu źródłowego.
- Wykorzystaliśmy wbudowany programator/debugger ST-LINK do wgrania oprogramowania do pamięci mikrokontrolera Nucleo

4. Schemat elektroniczny układu pomiarowego



5. Analiza modelu regulatora

5.1 Identyfikacja obiektu

W celu zaprojektowania układu sterowania dokonaliśmy analizy obiektu, którym jest system: serwomechanizm - mechaniczny model okna - otoczenie świetlne. Z punktu widzenia automatyki, obiekt ten został zidentyfikowany jako:

Obiekt inercyjny: charakteryzujący się opóźnieniem w odpowiedzi na sygnał sterujący wynikającym z bezwładności mechanicznej serwa.

Obiekt nieliniowy: o ograniczonym zakresie ruchu (zakres pracy serwa)..

5.2 Modelowanie

Na potrzeby analizy stabilności oraz doboru nastaw, przyjęliśmy uproszczony model matematyczny w postaci obiektu inercyjnego pierwszego rzędu. Transmitancję operatorową obiektu opisano wzorem:

$$G_o(s) = \frac{K}{T_S+1}$$

gdzie:

K - wzmocnienie obiektu,
T - stała czasowa.

Regulator PID w postaci ciągłej opisany jest równaniem:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

Ze względu na implementację cyfrową w STM32, zastosowaliśmy dyskretyzację algorytmu (metoda prostokątów dla członu całkującego i różnic wstecznych dla różniczkującego).

5.3 Dobór nastaw regulatora

Nastawy regulatora (K_p, K_i, K_d) zostały dobrane metodą prób i błędów. Proces strojenia przebiegał w następujących krokach:

1. Człon P: Wyznaczenie wzmocnienia krytycznego, przy którym układ zaczyna reagować dynamicznie, ale stabilnie.
2. Człon I :W projekcie przyjęto wartość $K_I = 0$, ponieważ zaimplementowany algorytm sterowania ma charakter przyrostowy). Operacja sumowania wyjścia regulatora (*current_angle* += *contro*l) wprowadza do układu działanie całkujące, które naturalnie eliminuje uchyb ustalony. Dodanie programowego członu I doprowadziłoby do

powstania układu z podwójnym całkowaniem, co skutkowałoby niestabilnością i silnymi oscylacjami serwomechanizmu.
3. Człon D: Dodanie składowej różniczkującej w celu poprawy tłumienia drgań i skrócenia czasu regulacji.

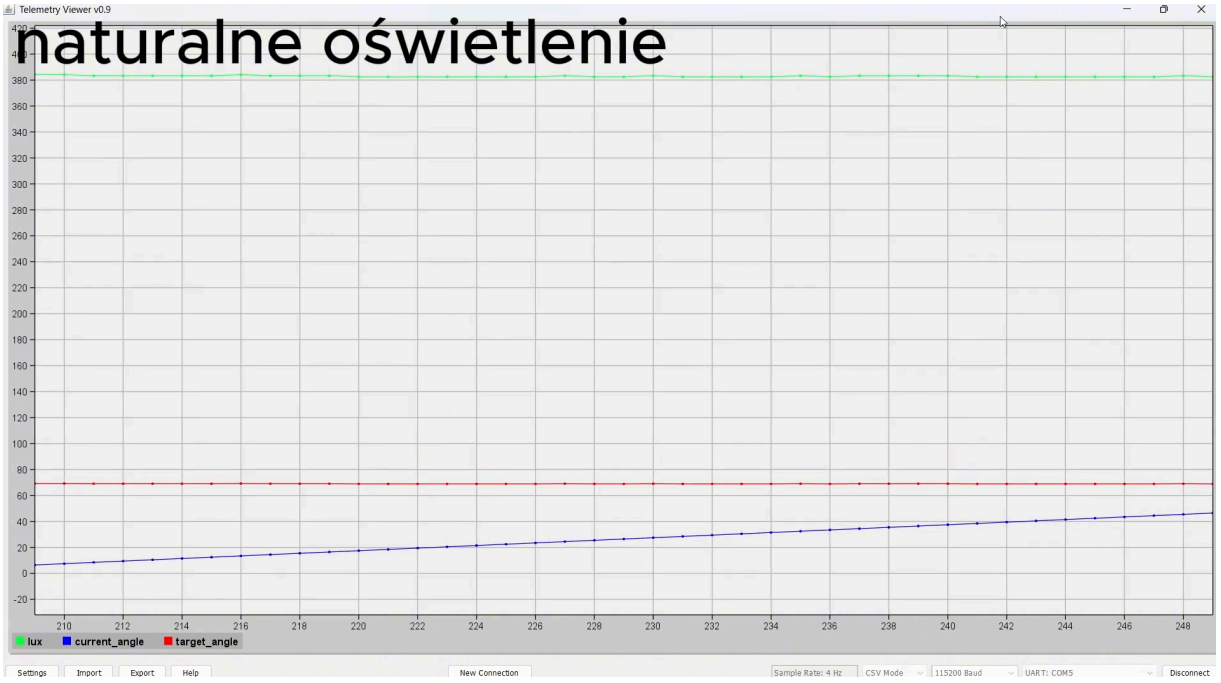
Dobrane nastawy pozwoliły na uzyskanie stabilnej odpowiedzi przy minimalnym przeregulowaniu.

6. Opis skryptów lub modeli symulacyjnych do logowania sygnałów sterujących i pomiarowych

W celu weryfikacji poprawności działania algorytmu sterowania oraz rejestracji sygnałów pomiarowych w czasie rzeczywistym, wykorzystaliśmy oprogramowanie Telemetry Viewer v0.9. Wybór tego narzędzia podyktowany był rekomendacją zawartą w specyfikacji projektowej oraz jego wysoką kompatybilnością z interfejsem szeregowym mikrokontrolerów STM32.

Aplikacja łączy się z układem sterowania poprzez wirtualny port COM , emulowany przez programator ST-LINK. Aby zapewnić poprawną konwersje danych, parametry połączenia w programie Telemetry Viewer skonfigurowano identycznie jak w module USART3 mikrokontrolera

Wewnątrz środowiska Telemetry Viewer zdefiniowano układ danych (Data Layout), który przypisuje poszczególne kolumny strumienia do odpowiednich zmiennych na wykresie.



Zrzut ekranu z programu Telemetry Viewer v0.9 podczas wizualizacji danych na żywo

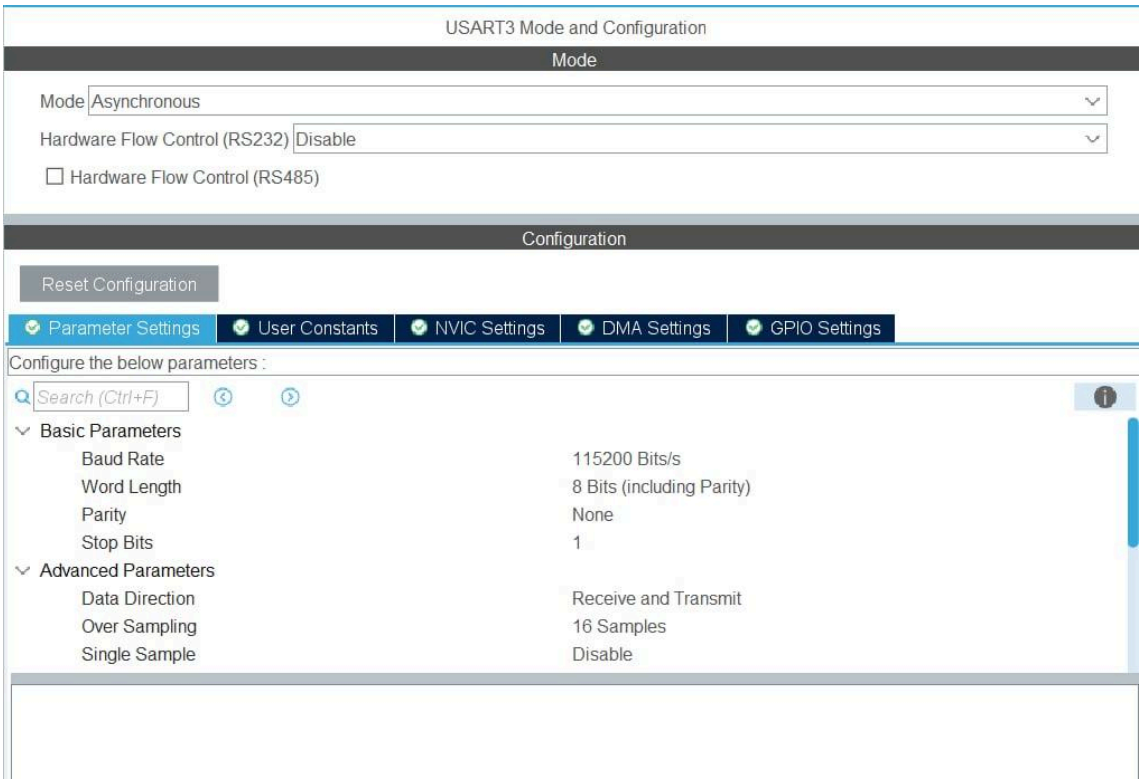
Dzięki takiemu rozwiązaniu możliwe jest bieżące monitorowanie dynamiki układu, wykrywanie oscylacji oraz logowanie przebiegów do pliku CSV w celu późniejszej analizy .

7. Prezentacja i opis konfiguracji projektu w środowisku STM32CubeIDE.

Poniższe podrozdziały przedstawiają szczegółowe ustawienia kluczowych modułów wykorzystanych w projekcie: interfejsu komunikacyjnego USART, licznika systemowego TIM1 oraz magistrali czujników I2C.

7.1 Konfiguracja interfejsu komunikacyjnego USART3

Do przesyłania danych telemetrycznych do komputera PC wykorzystaliśmy interfejs USART3 pracujący w trybie asynchronicznym. Ustawienia te pozwalają wizualizację danych na żywo.

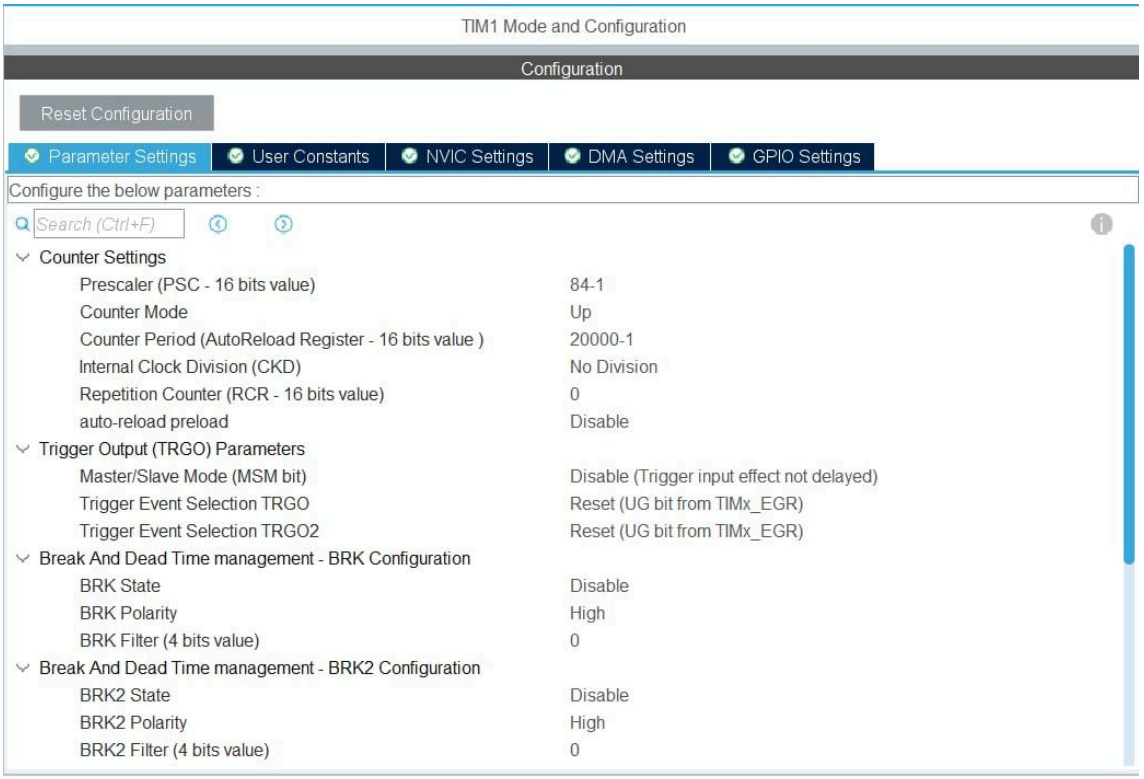


Rys. Konfiguracja parametrów USART3 w STM32CubeIDE

7.2 Konfiguracja licznika TIM1 (Podstawa czasu)

Timer TIM1 skonfigurowaliśmy jako źródło przerwań cyklicznych, które wyznaczają częstotliwość próbkowania sygnału. Dzięki temu pomiary oraz wysyłanie danych odbywają się w stałych, deterministycznych odstępach czasu.

Rys. Konfiguracja parametrów TIM1 w STM32CubeIDE



Rys. Konfiguracja parametrów TIM1 w STM32CubeIDE

Wartości parametrów dobraliśmy dla częstotliwości taktowania szyny timera wynoszącej 84 MHz (zgodnie z zegarem systemowym):

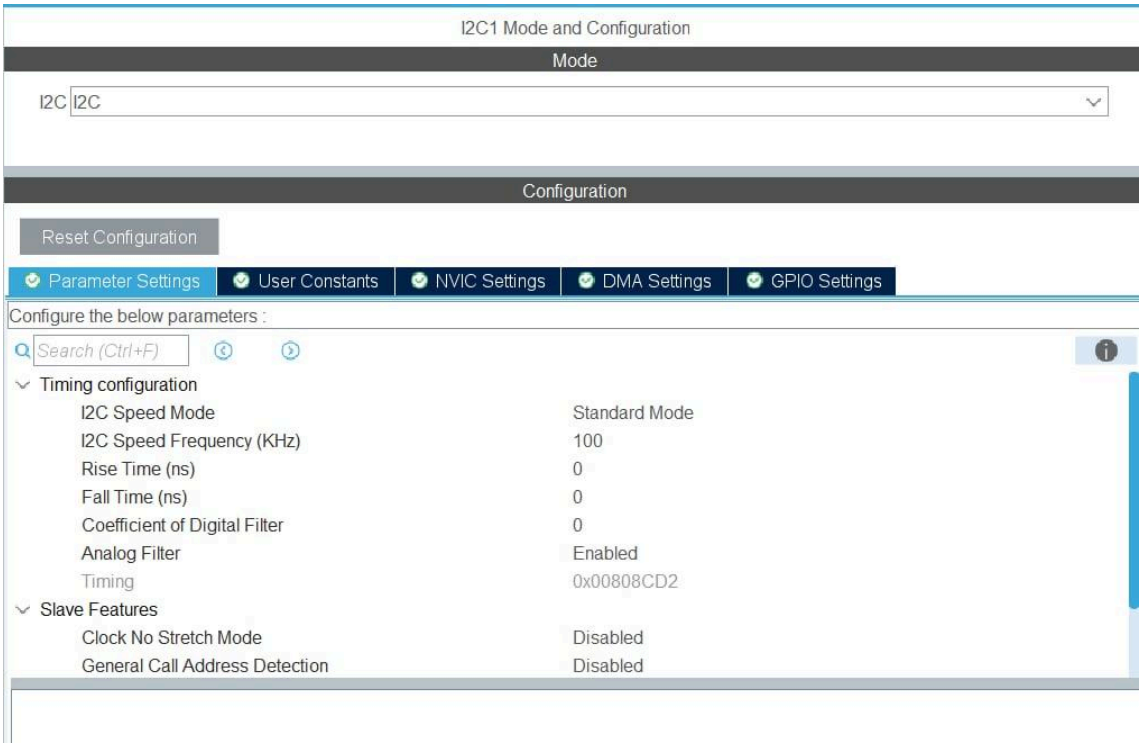
Prescaler : Ustawiliśmy na wartość 84-1. Powoduje to podział zegara wejściowego przez 84, co daje częstotliwość zliczania licznika równą 1 MHz.

Counter Period : Ustawiliśmy na wartość 20000-1. Licznik zlicza 20 000 impulsów zegara 1 MHz przed przepełnieniem

Wynikowy okres przerwania obliczyliśmy zgodnie z zależnością: T = 1 µs * 20000 = 20 ms. Taka konfiguracja zapewnia stabilne próbkowanie sygnału, co jest wystarczające dla wizualizacji charakterystyki zmian bez nadmiernego obciążania łącza komunikacyjnego.

7.3 Konfiguracja magistrali I2C

Do komunikacji z czujnikiem pomiarowym wykorzystaliśmy interfejs sprzętowy I2C. Magistrala została skonfigurowana w trybie Master, co pozwala mikrokontrolerowi na inicjowanie transmisji i odczyt rejestrów czujnika.



Rys. Konfiguracja parametrów TIM1 w STM32CubeIDE

8. Prezentacja i opis najważniejszych fragmentów kodu mikrokontrolera

8.1 Implementacja algorytmu PID

```c
float PID_Compute(PID_Controller *pid, float setpoint, float measured) {
    float error = setpoint - measured;
    pid->integral += error;
    float derivative = error - pid->prev_error;

    float output = pid->Kp * error + pid->Ki * pid->integral + pid->Kd * derivative;

    pid->prev_error = error;
    return output;
}
```

Algorytm regulatora został wydzielony do osobnego modułu. Wylicza on sygnał sterujący na podstawie uchybu, sumy uchybów (człon całkujący) oraz różnicy uchybów (człon różniczkujący).

8.2 Skalowanie sygnału pomiarowego na wartość zadaną

```c
float LuxToAngle(float lux)
{
    if (lux < 0) lux = 0;
    if (lux > 1000) lux = 1000;

    return (lux / 1000.0f) * 180.0f;
}
```

W celu powiązania danych z czujnika oświetlenia z elementem wykonawczym, zaimplementowaliśmy funkcję skalującą. Funkcja ta mapuje zmierzone natężenie światła (zakres 0 - 1000) na docelowy kąt otwarcia okna (zakres 0 - 180 stopni). Zawiera ona również mechanizm nasycenia, który zapobiega błędom obliczeniowym w przypadku przekroczenia zadanego zakresu oświetlenia.

8.3 Funkcja sterowania serwomechanizmem

```c
void Servo_SetAngle(uint8_t angle)
{
    uint16_t pulse = 500 + (angle * 2000) / 180;
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, pulse);
}
```

Bezpośrednie sterowanie kątem otwarcia modelu okna realizowane jest poprzez funkcję Servo_SetAngle. Przelicza ona kąt wyrażony w stopniach na odpowiednią szerokość impulsu PWM (Pulse Width Modulation).

- Wykorzystaliśmy Timer 1, gdzie wartość rejestru Compare (pulse) jest dynamicznie zmieniana w zakresie od 500 do 2500 jednostek.
- Pozwala to na uzyskanie standardowego sygnału sterującego dla serwomechanizmu SG90 (wypełnienie impulsu od 1 ms do 2 ms).

8.4 Komunikacja szeregowa (UART)

```
len = sprintf(uartBuf, "%.2f,%.2f,%.2f\r\n", target_angle_live, current_angle, bh1750_lux_live);
HAL_UART_Transmit(&huart3, (uint8_t*)uartBuf, len, 100);
```

W celu monitorowania parametrów pracy w czasie rzeczywistym, układ wysyła dane o kącie i natężeniu światła przez interfejs UART w formacie tekstowym.

8.5 Realizacja pętli sprzężenia zwrotnego i kontrola dynamiki

```
bh1750_lux_live = BH1750_ReadLux(&hi2c1);
          target_angle_live = LuxToAngle(bh1750_lux_live);

          float error = target_angle_live - current_angle;

          if (fabsf(error) < ANGLE_DEADBAND)
          {

          }
          else
          {
              float control = PID_Compute(&pid, target_angle_live, current_angle);

              if (control > 1.0f)  control = 1.0f;
              if (control < -1.0f) control = -1.0f;

              current_angle += control;

              if (current_angle < 0)   current_angle = 0;
              if (current_angle > 180) current_angle = 180;

              current_angle_live = current_angle;


              Servo_SetAngle((uint8_t)current_angle);
          }
```

Serce programu stanowi pętla główna, która w każdym cyklu wykonuje pełny algorytm sterowania. Kluczowym elementem jest tutaj implementacja limitera prędkości, który zapobiega gwałtownym skokom serwomechanizmu. W każdym obiegu pętli sprawdzany jest stan przycisku użytkownika, co pozwala na natychmiastową zmianę trybu pracy z automatycznego na manualny i resetowanie nastaw regulatora w sytuacjach awaryjnych.

9.Prezentacja i opis działania zaprojektowanego regulatora i zbadanie jego uchybu

```c
#include "pid.h"

float PID_Compute(PID_Controller *pid, float setpoint, float measured)
{
    float error = setpoint - measured;
    pid->integral += error;
    float derivative = error - pid->prev_error;

    float output = pid->Kp * error
                 + pid->Ki * pid->integral
                 + pid->Kd * derivative;

    pid->prev_error = error;
    return output;
}
```

Działanie zaprojektowanego układu opiera się na cyfrowej implementacji algorytmu PID pracującego w pętli zamkniętego sprzężenia zwrotnego. Układ w sposób ciągły dąży do zminimalizowania różnicy między wyliczonym kątem zadanym (wynikającym z natężenia światła), a aktualnym położeniem serwomechanizmu.

9.1. Dynamika pracy i ograniczenie prędkości

W układzie zastosowane zostało hybrydowe podejście do sterowania. Choć za wyliczenie sygnału sterującego odpowiada algorytm PID, jego wyjście podlega nasyceniu do wartości +- 1.0° na cykl pętli.

- Przy dużych zmianach oświetlenia okno nie wykonuje gwałtownego skoku, lecz porusza się z jednostajną prędkością ok. 4°/s. Zapobiega to uderzeniom mechanicznym i przeciążeniom serwomechanizmu SG90.
- Analityczne wyliczenie punktu wejścia w zakres pracy liniowej regulatora (przy nastawach $K_p = 0.1$, $K_d = 0.2$) wykazało, że układ zaczyna hamować i precyzyjnie dawkować krok serwa, gdy uchyb spadnie poniżej 12°. Powyżej tej wartości regulator pracuje w nasyceniu.

9.2. Eliminacja drgań i stabilność (Deadband)

Istotnym aspektem implementacji jest zastosowanie programowej strefy nieczułości o wartości wynoszącej 2°.

- Czujnik BH1750, jak każdy sensor cyfrowy, generuje drobne szumy pomiarowe. Bez strefy nieczułości, regulator PID próbowałby korygować błędy rzędu 0.1°, co powodowałoby nieustanny szum i drgania serwa.
- Po osiągnięciu celu z dokładnością do 2°, układ wstrzymuje pracę, co znacząco przedłuża żywotność silnika i eliminuje irytujący dźwięk pracy serwa w stanie ustalonym.

9.3. Analiza uchybu i rola członów PID

W celu oceny jakości pracy układu przeprowadziliśmy analizę uchybu regulacji, definiowanego jako różnica pomiędzy wartością zadaną a rzeczywistym położeniem serwomechanizmu. Uchyby pojawiały się głównie w fazie przejściowej, na skutek bezwładności mechanicznej serwa oraz dynamiki obiektu regulacji.

Na podstawie obserwacji przebiegów z aplikacji Telemetry Viewer zauważyliśmy, że uchyb po krótkim czasie zanika i utrzymuje się w pobliżu zera. Chwilowe odchylenia były związane głównie z nagłymi zmianami natężenia światła. Osiągnięte wyniki potwierdzają poprawność doboru nastaw regulatora oraz skuteczność zastosowanego algorytmu PID.

9.4. Bezpieczeństwo numeryczne

Mimo że matematyczna mapa obiektu zamyka się w przedziale (0 - 180)°, w kodzie zastosowaliśmy dodatkowe ograniczenia zmiennej położenia. Chroni to układ przed ewentualną akumulacją błędów liczb zmiennoprzecinkowych, gwarantując, że sygnał PWM wysyłany do serwonapędu nigdy nie przekroczy bezpiecznych granic konstrukcyjnych.

10.Prezentacja i opis funkcjonalności projektu

Czujnik światła na bieżąco odczytuje wartość natężenia światła, która następnie jest przetwarzana przez mikrokontroler. Na podstawie tej wartości wyznaczana jest pozycja serwomechanizmu w zakresie od 0 do 180 stopni. Im większe natężenie światła, tym większy kąt otwarcia „okna”, co umożliwia płynną i ciągłą regulację położenia serwa.

Do precyzyjnego sterowania ruchem serwomechanizmu zastosowano regulator PID, który minimalizuje uchyb pomiędzy wartością zadaną a rzeczywistym położeniem wyjścia. Regulator zapewnia stabilną pracę układu, szybkie reagowanie na zmiany warunków oświetleniowych oraz ograniczenie oscylacji i przeregulowań.

System wykorzystuje dwukierunkową komunikację szeregową pomiędzy mikrokontrolerem Nucleo a komputerem. Za pomocą tej komunikacji dane pomiarowe są przesyłane do aplikacji monitorującej Telemetry Viewer, który realizuje graficzną wizualizację danych w czasie rzeczywistym. Aplikacja wyświetla wykresy przedstawiające wartości natężenia światła, sygnał sterujący regulatora PID oraz aktualne położenie serwomechanizmu. Dzięki temu użytkownik ma możliwość obserwacji dynamiki pracy układu oraz analizy poprawności działania systemu sterowania.

Po wciśnięciu przycisku bezpieczeństwa działanie programu zostaje wstrzymane, a sterowanie serwem zatrzymane. Po zwolnieniu przycisku serwomechanizm automatycznie ustawia się w pozycji neutralnej (90 stopni), a następnie system wznawia normalną pracę regulacyjną. Funkcja ta stanowi element systemu bezpieczeństwa, chroniący użytkownika w przypadku niepożądanej sytuacji, na przykład zablokowania okna.

11.Link do filmu z prezentacją działania projektu

https://www.youtube.com/watch?v=kGBdzYTN9i0

12. Link do repozytorium GitHub z plikami projektu

https://github.com/Wojtaspl4-pp/Projekt-mikroprocesory

13.Podsumowanie

W ramach projektu zaprojektowaliśmy oraz zaimplementowaliśmy mikroprocesorowy układ automatycznej regulacji kąta otwarcia okna w zależności od natężenia oświetlenia zewnętrznego. Zrealizowany system spełnia wszystkie założenia funkcjonalne i sprzętowe postawione na etapie projektowania.

Zastosowanie czujnika BH1750 umożliwiło nam dokładny i stabilny pomiar natężenia światła, natomiast mikrokontroler STM32 Nucleo-F756ZG zapewnił odpowiednią wydajność obliczeniową do realizacji algorytmu regulatora PID oraz obsługi interfejsów komunikacyjnych. Sterowanie serwomechanizmem SG90 przy użyciu sygnału PWM

pozwoliło nam na precyzyjne i płynne ustawianie kąta otwarcia okna w pełnym zakresie roboczym.

Zaimplementowany przez nas regulator PID zapewnia stabilną pracę układu, szybkie reagowanie na zmiany warunków oświetleniowych oraz minimalny uchyb ustalony. Odpowiednio dobrane nastawy regulatora umożliwiły ograniczenie przeregulowań i oscylacji, co potwierdza poprawność przeprowadzonego przez nas procesu strojenia metodą eksperymentalną.

Dodatkowym atutem projektu jest zastosowanie komunikacji UART oraz narzędzia Telemetry Viewer, które umożliwiły nam bieżące monitorowanie parametrów pracy i analizę dynamiki układu w czasie rzeczywistym. Wprowadzenie przycisku bezpieczeństwa zwiększyło niezawodność i funkcjonalność systemu, umożliwiając szybkie zatrzymanie pracy w sytuacjach awaryjnych.

Projekt stanowi dobrą bazę do dalszego rozwoju, na przykład poprzez implementację mechanizmu anti-windup, automatycznego doboru nastaw regulatora lub rozbudowę systemu o dodatkowe czujniki środowiskowe. Zdobyte przez nas doświadczenia potwierdzają przydatność regulatorów PID w praktycznych zastosowaniach systemów automatyki oraz efektywność platformy STM32 w realizacji zadań sterowania w czasie rzeczywistym.

14.Bibliografia

1. M. Szumski, Mikrokontrolery STM32 w systemach sterowania i regulacji, BTC, 2018.
2. M. Galewski, STM32. Aplikacje i ćwiczenia w języku C z biblioteką HAL, BTC, 2019.
3. A. Kurczyk, Mikrokontrolery STM32 dla początkujących, BTC, 2019.
4. K. Paprocki, Mikrokontrolery STM32 w praktyce, BTC, 2009.
5. P. Hadam, Projektowanie systemów mikroprocesorowych, BTC, 2004.