Symulator Układu Automatycznej Regulacji

Rodzaj zajęć i nazwa przedmiotu

Programowanie Komputerów - Projekt

Rok akademicki	Kierunek	Prowadzący	Semestr	Sekcja
2025/2026	IPpp	Dr inż. Łukasz Maliński	3	1

Imię i nazwisko

Artur Zabor,
Michał Walterowski

1. Skład sekcji wraz z podziałem obowiązków w realizacji projektu

Michał Walterowski:
- Modelowanie obiektowe – implementacja ARX

- Programowanie GUI w Qt – tworzenie interfejsu graficznego

- Wizualizacja danych – tworzenie wykresów

- Tworzenie dokumentacji technicznej (schemat UML)

- Tworzenie prezentacji podsumowywujące poszczególne etapy projektu

Artur Zabor:
- Modelowanie obiektowe – implementacja PID

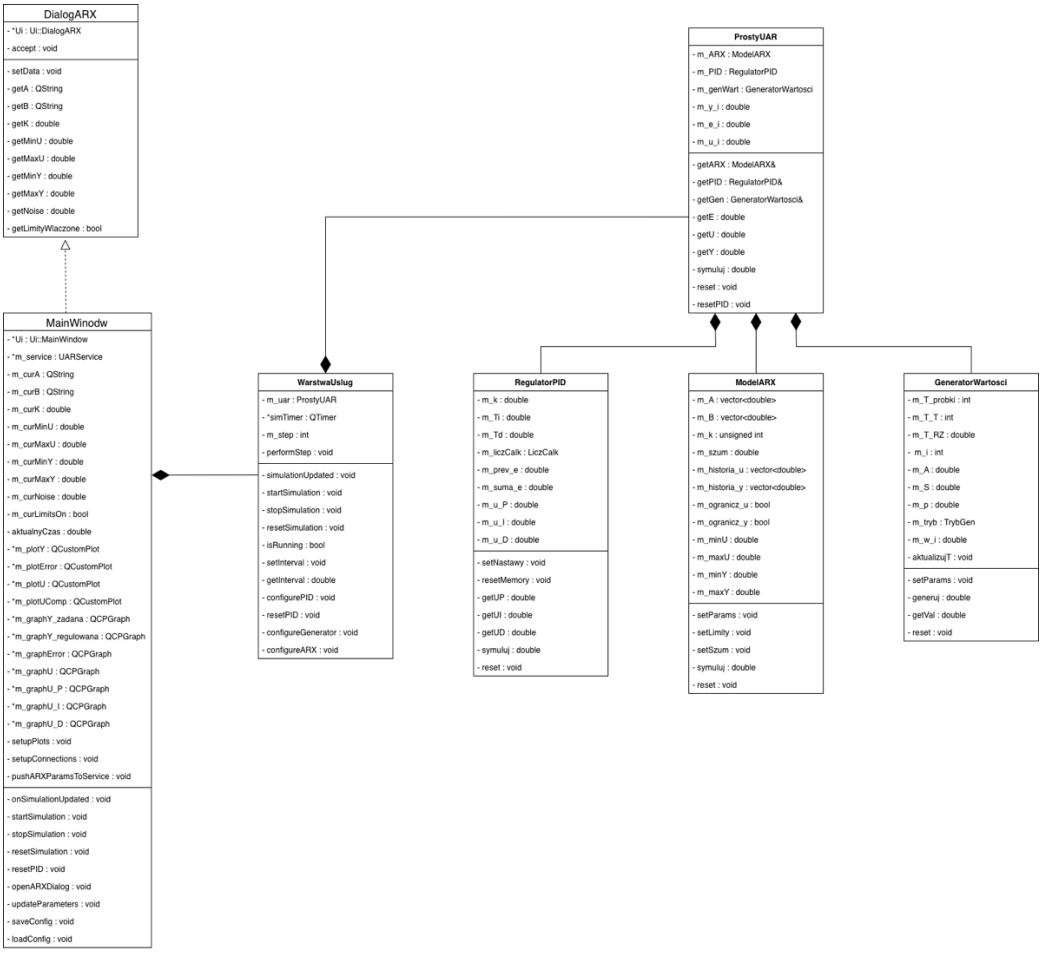
- Architektura warstwowa – tworzenie warstwy danych i warstwy usług

- Testowanie – testy jednostkowe

- Tworzenie dokumentacji technicznej (schemat UML)

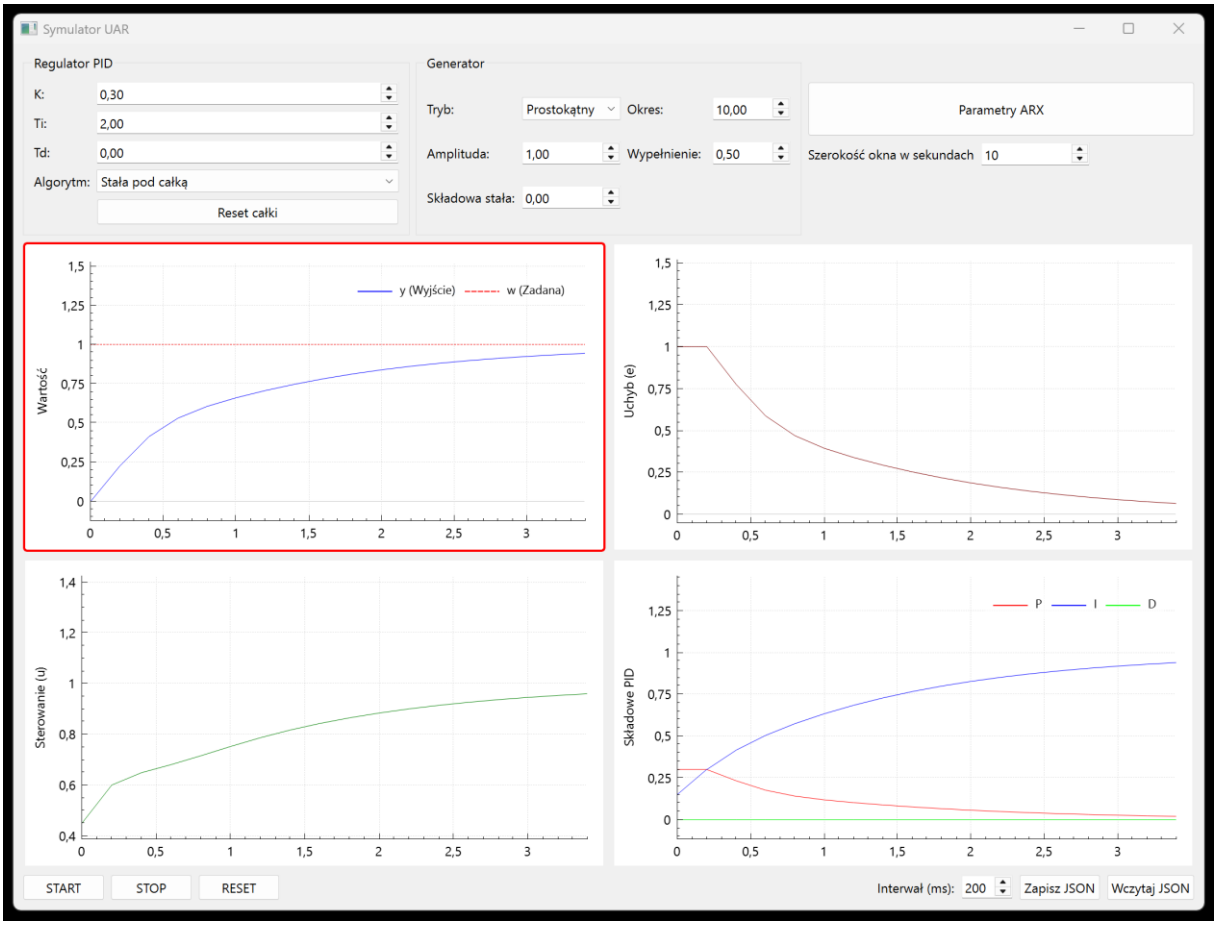
- Tworzenie prezentacji podsumowywujące poszczególne etapy projektu

2. Końcową wersję schematu UML całej aplikacji z wyraźnym podziałem na warstwy i z poprawnym zaznaczeniem relacji obiektowych

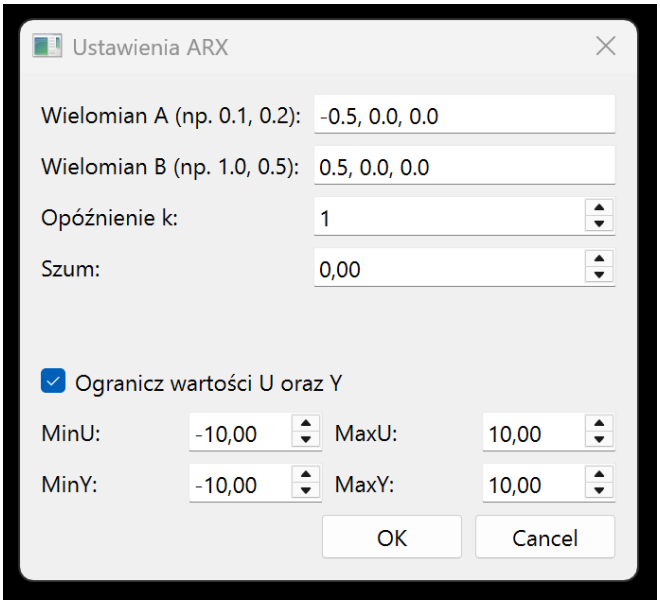


3. Kocówy wygląd GUI programu

Główne okno programu symulatora UAR



Okno z ustawieniami ARX



4. Krótką historię rozwoju projektu aplikacji z uwzględnieniem zmian w stosunku do pierwotnego projektu, przedstawioną w punktach, ale z opisem tego co się zmieniło i dlaczego

- Etap I: Od koncepcji do modelu matematycznego

Co się zmieniło: Pierwotny projekt zakładał umieszczenie logiki modelu ARX bezpośrednio w klasie zarządzającej symulacją. Po analizie wymagań zdecydowano się na pełną hermetyzację modelu ARX jako niezależnej klasy w warstwie danych.

Dlaczego: Pozwoliło to na zaliczenie rygorystycznych testów jednostkowych dostarczonych przez prowadzącego jeszcze przed budową GUI. Dzięki temu zyskaliśmy pewność, że fundament matematyczny (wielomiany A, B, opóźnienie k oraz szum) działa.

- Etap II: Architektura trójwarstwowa i wprowadzenie Warstwy Usług

Co się zmieniło: W fazie realizacji back-endu zrezygnowano z bezpośredniego łączenia GUI z obiektami ARX/PID. Wprowadzono klasę Warstwa Usług.

Dlaczego: Była to kluczowa modyfikacja w stosunku do wstępnego schematu UML. Wprowadzenie Warstwy Usług pozwoliło odseparować logikę sterowania od samej matematyki obiektu.

- Etap III: Dynamiczne GUI i separacja parametrów ARX

Co się zmieniło: Wstępny projekt GUI zakładał edycję wszystkich parametrów w jednym oknie. W fazie implementacji parametry ARX przeniesiono do osobnego, modalnego okna.

Dlaczego: Podczas testów okazało się, że wpisywanie długich wektorów współczynników A i B "na żywo" w głównym oknie powoduje błędy symulacji (odczyt niekompletnych wartości). Osobne okno zapewniło zbiorcze zatwierdzanie parametrów dopiero po zakończeniu edycji, co zagwarantowało stabilność pętli sprzężenia zwrotnego.

-Etap IV: Optymalizacja wizualizacji i skalowania

Co się zmieniło: System wykresów został przeprojektowany tak, aby implementować "płynne okno obserwacji" zamiast nieskończonego przyrostu danych na osi czasu. Dodano autorski algorytm dynamicznego skalowania osi pionowej.

Dlaczego: Bez tej zmiany wykresy stawały się nieczytelne po kilku minutach pracy. Wprowadzenie limitu próbek w warstwie prezentacji przy jednoczesnym zachowaniu pełnej historii w warstwie danych pozwoliło na płynne działanie aplikacji przy interwale 50 ms i spełnienie wymagań z check-listy.

- Etap V: Implementacja trwałości danych (JSON)

Co się zmieniło: Mechanizm zapisu konfiguracji został rozbudowany o walidację danych wejściowych przed zapisem do pliku .json.

Dlaczego: Aby uniknąć błędów przy wczytywaniu uszkodzonych plików konfiguracyjnych, które mogłyby doprowadzić do awarii aplikacji (np. ujemne opóźnienie transportowe). Dzięki temu proces zapisu/odczytu stał się "odporny na błędy" użytkownika.

5. Krótkie wypunktowanie przynajmniej czterech najistotniejszych napotkanych trudności w realizacji tematu i sposobów ich rozwiązania

5.1. Zarządzanie historią sygnałów w modelu ARX przy dynamicznym opóźnieniu

Trudność: Implementacja równania różnicowego ARX wymaga dostępu do wartości sygnałów sterujących (u) i regulowanych (y) z wielu kroków wstecz. Wyzwanie stanowiło wydajne zarządzanie tą historią oraz obsługa dynamicznej zmiany opóźnienia transportowego k w trakcie pracy.

Rozwiązanie: W klasie ModelARX zastosowano kontenery std::deque zamiast zwykłych tablic. Pozwoliło to na szybkie dodawanie nowych próbek metodą push_front i usuwanie najstarszych. Przy zmianie parametrów mechanizm resize dba o odpowiedni zapas miejsca w buforze, co zapobiega błędom naruszenia pamięci przy zwiększaniu rzędu wielomianów lub opóźnienia.

5.2. Skoki sygnału przy zmianie trybu pracy i nastaw części całkującej PID

Trudność: Wymagania projektowe narzucały dwa tryby liczenia całki (stała przed i pod sumą). Zmiana nastawy T_i w trybie "stała przed sumą" powodowała gwałtowne skoki składowej u_i, co destabilizowało układ.

Rozwiązanie: W metodzie RegulatorPID::setNastawy zaimplementowano mechanizm przeliczania zgromadzonej sumy uchybu podczas przełączania trybów. Dzięki temu wartość składowej całkującej pozostaje ciągła w momencie zmiany parametrów, co eliminuje niepożądane szpilki na sterowaniu.

5.3. Wydajność wizualizacji danych w czasie rzeczywistym

Trudność: Ciągłe dopisywanie punktów do wykresów QCustomPlot przy niskim interwale (50 ms) prowadziło do narastającego zużycia pamięci i spowolnienia interfejsu (GUI) wraz z upływem czasu trwania symulacji.

Rozwiązanie: W klasie MainWindow zaimplementowano algorytm usuwania starych danych metodą removeBefore. Dane starsze niż zdefiniowane przez użytkownika "okno obserwacji" są na bieżąco usuwane z wykresów. Dodatkowo zastosowano funkcję applySmartScale, która automatycznie dopasowuje osie pionowe, dbając o czytelność sygnałów przy minimalnym obciążeniu procesora.

5.4. Synchronizacja parametrów między oknem modalnym a Warstwą Usług

Trudność: Zgodnie z architekturą trójwarstwową, parametry modelu ARX edytowane w DialogARX nie mogły bezpośrednio wpływać na symulację, dopóki użytkownik nie zatwierdził zmian. Istniało ryzyko niespójności danych między GUI a logiką.

Rozwiązanie: Zastosowano mechanizm "bufora parametrów" w MainWindow. Parametry są przechowywane w polach klasy (np. m_curA, m_curB) i przekazywane do UARService zbiorczo za pomocą metody pushARXParamsToService dopiero po wywołaniu sygnału Accepted z okna dialogowego. Zapewnia to atomowość zmian i chroni przed błędami w trakcie obliczeń.

6. Zwięzłe podsumowanie czego nauczył się każdy członek sekcji (oddzielnie dla każdej osoby)

Michał Walterowski:

Optymalizacja wizualizacji danych: Nauczyłem się zarządzać wydajnością interfejsu przy dynamicznych wykresach (QCustomPlot), stosując mechanizm usuwania starych próbek (removeBefore) oraz autorski algorytm "inteligentnego" skalowania osi pionowej.

Obsługa interakcji w oknach modalnych: Opanowałem technikę bezpiecznego przekazywania i walidacji danych między oknem głównym a oknem dialogowym (DialogARX), w tym sprawdzanie poprawności wektorów współczynników przed ich zatwierdzeniem.

Serializacja konfiguracji do JSON: Nauczyłem się wykorzystywać klasy QJsonDocument i QJsonObject do trwałego zapisu parametrów symulacji, co pozwoliło na szybkie przywracanie stanu pracy programu bez konieczności ponownego wpisywania nastaw.

Implementacja modeli matematycznych: Opanowałem praktyczne zastosowanie równań różnicowych (model ARX) przy użyciu kontenerów std::deque, co pozwoliło na wydajne zarządzanie historią sygnałów i obsługę opóźnienia transportowego.

Artur Zabor:

Projektowanie architektury warstwowej: Nauczyłem się wdrażać wzorzec Fasady (UARService), który skutecznie oddziela warstwę prezentacji od logiki matematycznej, ułatwiając zarządzanie kodem i przepływem danych.

Zaawansowana logika regulatora PID: Zrozumiałem i zaimplementowałem różnice w sposobach całkowania uchybu (stałą pod sumą, a stałą przed sumą) oraz nauczyłem się mechanizmu nasycenia sygnałów, niezbędnego dla stabilności obiektów fizycznych.

Przetwarzanie sygnałów w czasie rzeczywistym: Zrozumiałem różnicę między czasem rzeczywistym a czasem dyskretnym, implementując generator sygnałów (sinusoidalny i prostokątny) z przelicznikiem okresu na liczbę próbek zależną od interwału timera.

Automatyzacja testowania: Nauczyłem się tworzyć testy jednostkowe i integracyjne dla logiki "back-endowej", co pozwoliło na wykrycie błędów w pętli sprzężenia zwrotnego jeszcze przed uruchomieniem interfejsu graficznego.

Wspólne kompetencje miękkie i inżynierskie:

- Zarządzanie projektem i praca w grupie: Nauczyliśmy się efektywnego podziału obowiązków na etapie planowania, co pozwoliło na równoległe prowadzenie prac nad logiką symulacji oraz interfejsem graficznym. Zrozumieliśmy znaczenie wczesnego ustalenia interfejsów, dzięki czemu moduły tworzone przez różne osoby współpracowały bezbłędnie podczas integracji.

- Projektowanie w standardzie UML: Opanowaliśmy podstawy modelowania systemów obiektowych w języku UML. Nauczyliśmy się odwzorowywać relacje między klasami (kompozycja, asocjacja) oraz poprawnie dzielić strukturę programu na warstwy (Prezentacja, Usługi, Dane), co było kluczowe dla zachowania przejrzystości kodu.

- Komunikacja techniczna i prezentacja: Rozwinęliśmy umiejętność prezentowania skomplikowanych zagadnień technicznych (takich jak zasada działania modelu ARX czy algorytmu PID) podczas seminariów projektowych. Nauczyliśmy się dokumentować postępy prac w sposób zrozumiały dla odbiorcy oraz uzasadniać podejmowane decyzje projektowe.