

Sprawozdanie z Projektu z przedmiotu Programowanie komputerów

Mirostaw Franczak

Informatyka Przemysłowa

Semestr III

Rok 2025/2026

Prowadzący: Łukasz Maliński

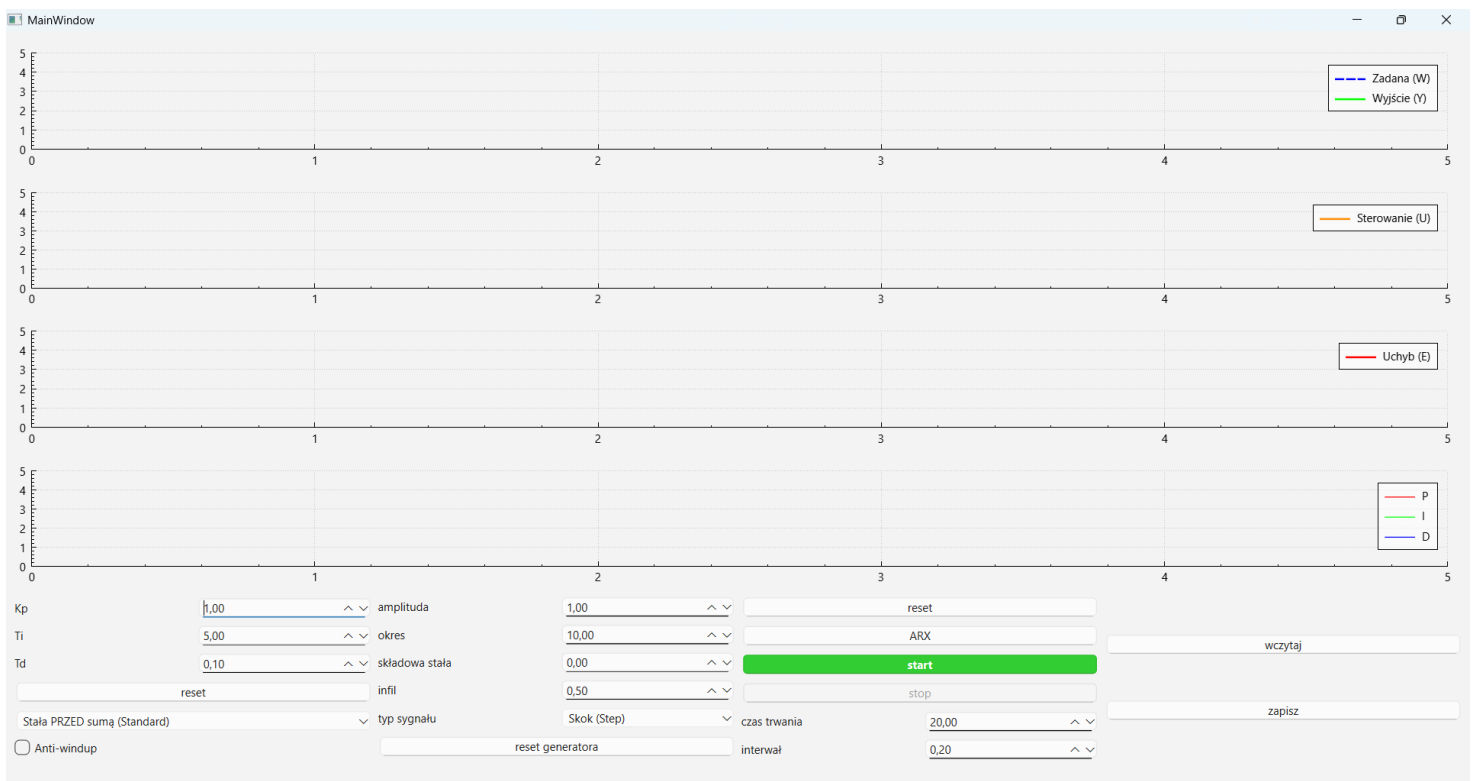
1. Skład sekcji wraz z podziałem obowiązków w realizacji projektu:

Autor projektu: Mirosław Franczak

Ze względu na indywidualny charakter realizacji projektu, ja odpowiadam za całość prac programistycznych i projektowych. Zakres obowiązków obejmował:

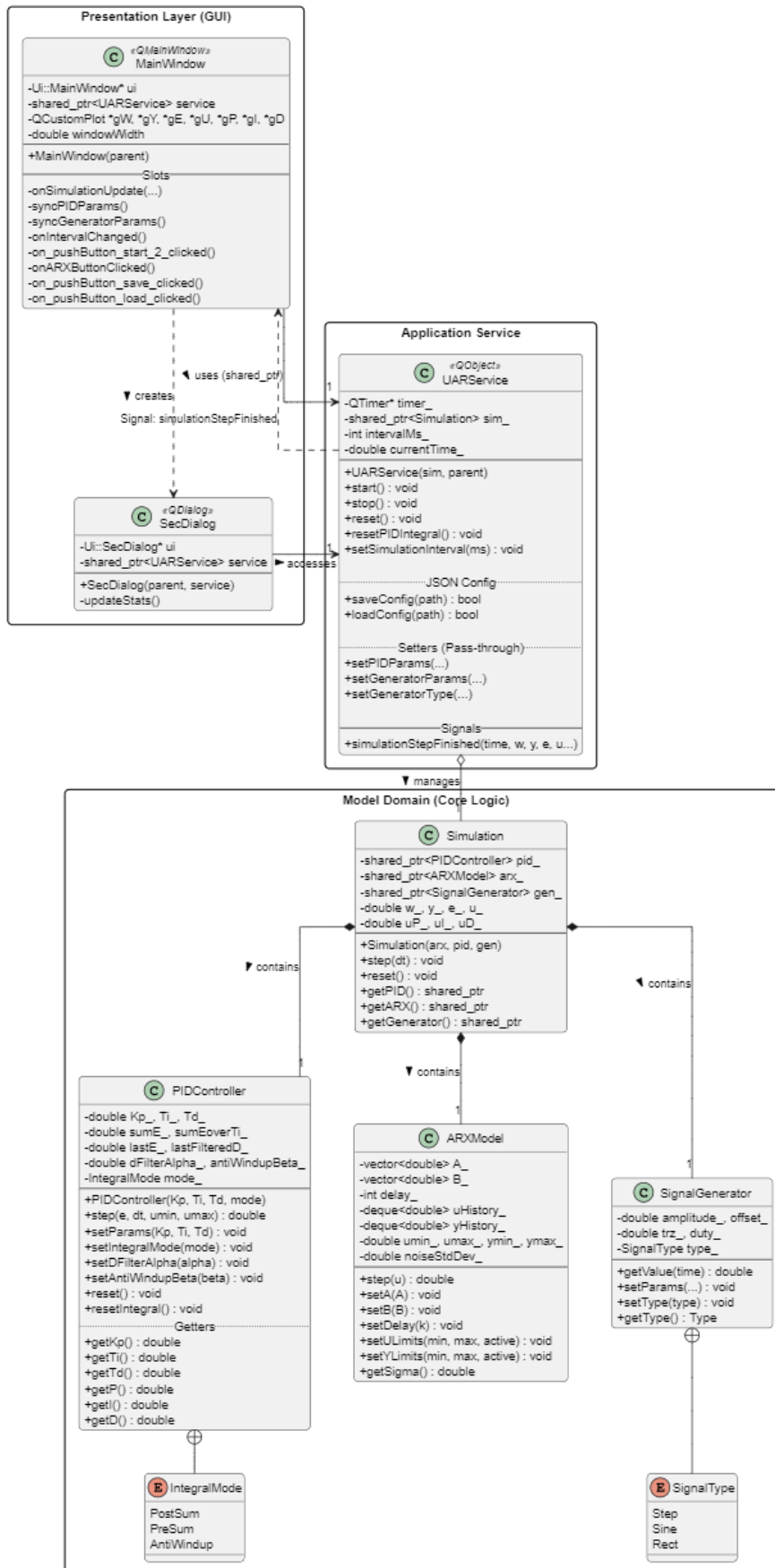
- **Backend:** Implementacja kompletnej logiki sterowania (PID, ARX, Generator) w języku C++.
- **Frontend:** Stworzenie interfejsu graficznego aplikacji oraz mechanizmów wizualizacji przebiegów czasowych.
- **Integracja:** Połączenie warstwy obliczeniowej z interfejsem użytkownika oraz testowanie stabilności układu regulacji.

2. Kocowy wygląd GUI programu



3. Końcową wersję schematu UML całej aplikacji

Diagram Klas Systemu Sterowania ARX



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

Projekt rozwijałem etapami, wprowadzając poprawki tam, gdzie symulacja nie działała zgodnie z oczekiwaniami. Oto najważniejsze rzeczy, które zmieniłem w porównaniu do pierwszej wersji:

1. Naprawa algorytmu PID (Całkowanie przed i pod sumą)

- **Co zmieniłem:** W pierwszej wersji miałem tylko jeden prosty wzór, który nie reagował poprawnie na zmianę wzmocnienia (K_p). Dodałem obsługę dwóch konkretnych trybów:
 - Stała przed sumą (Standard/ISA): Tutaj K_p mnoży też część całkującą. Jak zmieniam K_p , to zmienia się też siła całki.
 - Stała pod sumą (Niezależny): Tutaj część całkująca jest liczona osobno (tylko $1/T_i$) i K_p na nią nie wpływa.
- **Dlaczego:** Dzięki temu sterownik działa teraz zgodnie z teorią i widać na wykresach różnicę między tymi trybami przy zmianie nastaw.

2. Ulepszenie wykresów (Przesuwne okno)

- **Co zmieniłem:** Na początku wykres rysował wszystko od zera i po chwili linie robiły się bardzo ściśnięte. Dodałem mechanizm, który pokazuje tylko ostatnie np. 20 sekund (tzw. okno przesuwne) i usuwa stare dane z pamięci.
- **Dlaczego:** Teraz wykresy są czytelne nawet jak symulacja trwa długo, a program nie zacina się przez nadmiar danych.

3. Porządek w kodzie (Wydzielenie Serwisu)

- **Co zmieniłem:** Wcześniej wszystko (PID, ARX, Timer) było wrzucone do pliku głównego okna (MainWindow). Przeniósłem całą logikę do osobnej klasy UARService.
- **Dlaczego:** Teraz oddzieliłem obliczenia od wyglądu aplikacji (GUI). Kod jest czystszy i łatwiej było mi znaleźć błędy.

4. Poprawka Generatora (Opóźnienie skoku)

- **Co zmieniłem:** Generator sygnału skokowego działał od razu od zerowej sekundy. Przerobiłem go tak, że pole "Czas trwania" działa jak opóźnienie.
- **Dlaczego:** Mogę teraz ustawić, żeby skok nastąpił np. w 5. sekundzie, co pozwala zobaczyć, czy układ jest stabilny przed wystąpieniem wymuszenia.

5. Zapisywanie ustawień do pliku

- **Co zmieniłem:** Dodałem przyciski "Zapisz" i "Wczytaj", które zapisują wszystkie parametry (PID, ARX, Generator) do pliku JSON.
- **Dlaczego:** Nie muszę wpisywać wszystkiego ręcznie przy każdym uruchomieniu programu, co bardzo przyspieszyło testowanie.

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

Podczas pisania programu i testowania regulatora napotkałem różne problemy, które wymagały zmiany podejścia lub poprawy kodu:

1. Problem z niedochodzeniem sygnału wyjściowego do wartości zadanej (Uchyb ustalony)

- **Trudność:** Mimo włączonego członu całkującego (I), zielony wykres wyjścia często zatrzymywał się poniżej wartości zadanej i nie chciał do niej "dociągnąć". Wynikało to ze zbyt słabego działania całki przy małym czasie próbkowania.
- **Rozwiązanie:** Zmieniłem sposób sumowania błędów w kodzie regulatora. Zamiast mnożyć każdy błąd przez bardzo mały czas, zastosowałem sumowanie bezpośrednie z odpowiednim wzmocnieniem. Dzięki temu całka teraz działa poprawnie i zawsze sprowadza uchyb do zera.

2. Nieczytelność wykresów przy dłuższej symulacji

- **Trudność:** Na początku program rysował wszystkie punkty od momentu startu. Po minucie czy dwóch wykresy były tak ściśnięte, że nie dało się odczytać przebiegu, a program zaczynał zwalniać przez nadmiar danych w pamięci.
- **Rozwiązanie:** Dodałem mechanizm "okna przesuwnego". Program teraz pokazuje tylko ostatnie kilkadziesiąt sekund (zgodnie z ustawieniem "Czas trwania"), a starsze, niewidoczne dane są na bieżąco usuwane. Wykres jest czytelny niezależnie od tego, jak długo trwa test.

3. Brak możliwości opóźnienia sygnału testowego

- **Trudność:** Generator sygnału skokowego działał "na sztywno" – skok wartości następował zawsze w zerowej sekundzie. Nie mogłem przez to sprawdzić, czy układ jest stabilny przed wystąpieniem wymuszenia.
- **Rozwiązanie:** Przerobiłem logikę generatora tak, aby pole "Czas trwania" działało jako opóźnienie. Teraz sygnał zmienia się dopiero po upływie zadanego czasu, co pozwala zobaczyć stan równowagi przed skokiem.

4. Gwałtowne skoki sterowania (Szpilki) od członu różniczkującego

- **Trudność:** Kiedy zmieniałem wartość zadaną skokowo, człon różniczkujący (D) reagował bardzo gwałtownie, generując ogromne piki na wykresie sterowania. Powodowało to "szarpanie" wykresem.
- **Rozwiązanie:** Dodałem prosty filtr wygładzający (filtr dolnoprzepustowy) dla członu D. Dzięki temu pochodna jest liczona łagodniej, a sterowanie jest płynne, bez niepotrzebnych szpilek.

6. Zwięzłe podsumowanie czego nauczył się każdy członek sekcji

Realizując projekt samodzielnie, zdobyłem wiedzę w trzech kluczowych obszarach:

1. Praktyczna implementacja algorytmów sterowania: Zrozumiałem, że „książkowy” wzór PID nie wystarczy w cyfrowym sterowniku. Nauczyłem się, jak poprawnie zrealizować dyskretyzację algorytmu (sumowanie próbek zamiast całki ciągłej) oraz jak duży wpływ na stabilność układu ma wybór struktury regulatora (różnica między K_p przed sumą a pod sumą).
2. Programowanie aplikacji czasu rzeczywistego w Qt: Nauczyłem się obsługiwać bibliotekę QCustomPlot oraz zarządzać pamięcią programu przy ciągłym napływie danych. Dowiedziałem się, jak stworzyć mechanizm „okna przesuwne”, aby wykresy były płynne i czytelne nawet przy długotrwałej pracy programu.
3. Architektura oprogramowania: Przekonałem się, jak ważne jest oddzielenie logiki obliczeniowej od interfejsu graficznego. Wydzielenie klasy UARService pozwoliło mi zapanować nad kodem i łatwiej znajdować błędy, co przy pracy jednoosobowej było kluczowe dla ukończenia projektu.