НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Система автопілотування літаків побудована на pid регуляторах

Виконав : студент __4__ курсу, групи _ІО-91_
(шифр групи)

_____Діденко Владислав Віталійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Каплунов А.В._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант(нормоконтроль) ст. викладач Виноградов Ю.М _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __ _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Діденка Владислава Віталійовича

1. Тема проєкту *Система автопілотування літаків побудована на pid регуляторах*
керівник проєкту Каплунов Артем Володимирович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 31 травня 2023 року №2101-с

2. Термін здачі студентом закінченого проєкту *8 червня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Постановка задачі та дослідження

Розділ 2. Проектування додатку

Розділ 3. PID-регулятори

Розділ 4. Деталі розробки системи

Розділ 5. Дослідження та аналіз розробленої системи

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) Система автопілотування літаків побудована на pid регуляторах Структура систем літака (Структурна схема), Система автопілотування літаків побудована на pid регуляторах Діаграма варіантів користування (Функціональна схема), Система автопілотування літаків побудована на pid регуляторах Алгоритм дій програмного забезпечення (Принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання «9» лютого 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2022-15.12.2022*	
2.	*Вивчення та аналіз завдання*	*15.12.2022-15.03.2023*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2023-25.03.2023*	
4.	*Розробка структур окремих підсистем*	*25.03.2023-5.04.2023*	
5.	*Програмна реалізація системи*	*5.04.2023-15.04.2023*	
6.	*Оформлення пояснювальної записки*	*15.04.2023-05.06.2023*	
7.	*Захист програмного продукту*	*25.04.2023*	
8.	*Передзахист*	*05.06.2023*	
9.	*Захист*	*22.06.2023*	

Студент-дипломник _____ Владислав ДІДЕНКО
(підпис)

Керівник проєкту _____ Артем КАПЛУНОВ
(підпис)

АНОТАЦІЯ

У даній дипломній роботі було систематично досліджено тему автопілотування літаків на основі PID-регуляторів. Основною метою роботи було створення комплексу систем, який забезпечує виключення впливу факторів пілота та інших зовнішніх змінних параметрів при проведенні багаторазового пілотування, щоб забезпечити повторюваність та надійність тестового процесу.

В роботі представлено детальний огляд технологій та інструментів для розробки систем, зокрема, мови Lua, двигуна XPlane та PID-регуляторів.

В результаті була розроблена високоефективна система, що дозволяє проводити експерименти над літаком для отримання важливих даних без ризику шкоди для літака та людей.

Ключові слова: автопілот, PID-регулятори, тести, XPlane, Lua.

ANNOTATION

In this project for a Bachelor's Degree, of autopiloting of aircraft based on PID controllers was systematically investigated. The main purpose of the work was to create a complex of systems that ensures the exclusion of the impact of pilot factors and other external variable parameters during repeated piloting to ensure repeatability and reliability of the test process.

The work presents a detailed review of technologies and tools for the development of systems, in particular, the Lua language, XPlane engine and PID regulators.

As a result, a highly efficient system was developed that allows you to conduct experiments on the plane to obtain important data without the risk of damage to the aircraft and people.

Key words: autopilot, PID-controllers, tests, XPlane, Lua

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	*ІАЛЦ.466530.002 ТЗ*	Система автопілотування	4		
			літаків побудована на pid			
			регуляторах			
			Технічне завдання			
	А4	*ІАЛЦ.466530.003 ПЗ*	Система автопілотування	80		
			літаків побудована на pid			
			регуляторах			
			Пояснювальна записка			
	А4	*ІАЛЦ.466530.004 Д1*	Система автопілотування	1		
			літаків побудована на pid			
			регуляторах			
			Структура систем літака			
			(Структурна схема)			
	А4	*ІАЛЦ.4665308.005 Д2*	Система автопілотування	1		
			літаків побудована на pid			
			регуляторах			
			Діаграма варіантів			
			користування			
			(Функціональна схема)			
	А4	*ІАЛЦ.466530.006 Д3*	Система автопілотування	1		
			літаків побудована на pid			
			регуляторах			
			Алгоритм дій			
			програмного забезпечення			
			(Принципова схема)			
	А4	*ІАЛЦ.466530.007 Д4*	Система автопілотування	15		
			літаків побудована на pid			
			регуляторах			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	

ІАЛЦ.466530.001 ОА

Розроб		Діденко В.В.			*Система автопілотування літаків побудована на pid регуляторах* Опис альбому	Літ.	Аркуш	Аркушів
Перев		Каплунов А.В					1	1
						КПІ ім. Ігоря Сікорського ФІОТ ІО-91		

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система автопілотування літаків побудована на pid регуляторах»

Київ – 2023 р.

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку системи автоматичного управління (автопілота) для симулятора літака.

Областю застосування даної системи є віртуальна авіація, тренування пілотів, а також наукові дослідження в області авіоніки. Даний автопілот може бути корисним інженерам та науковцям для тестування та валідації аеродинамічних моделей і систем керування.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом "Інформатики та обчислювальної техніки" кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи автопілотування літаків побудована на pid регуляторах.

4. ДЖЕРЕЛА РОЗРОБКИ

1. uavAP: A Modular Autopilot Framework for UAVs [Електронний ресурс] / Mirco Theile, D. Dantsker, Richard Nai, Marco Caccamo. – 2020. –

Режим доступу до ресурсу: http://rtsl-edge.cs.illinois.edu/UAV/inc/uavAP_AIAA-Aviation-2020.pdf.
2. Autopilot [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Autopilot.
3. Lua [Електронний ресурс] – Режим доступу до ресурсу: http://www.lua.org/about.html
4. X-Plane [Електронний ресурс]: Доступ: https://flight.fandom.com/wiki/X-Plane_11
5. X-Plane (docs) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.x-plane.com/docs/.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Автоматизація процесів керування польотом та виконання польотних завдань.
- Надання можливості користувачам самостійно доповнювати та налаштовувати параметри автопілота згідно з потребами сценарію польоту.
- Забезпечення надійності роботи системи та коректності виконання польотних маневрів.
- Забезпечення повторюваності та надійності тестового процесу.

5.2 Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.

- XPlane 11.

5.3 Вимоги до апаратної частин

- ЦП не менше ніж Intel Core i3, i5, або i7 CPU з 2 або більше ядрами, або AMD еквівалент.
- ROM не менше ніж 20 ГБ.
- RAM не менше ніж 8 ГБ.

  Відеокарта з підтримкою DirectX 11 від NVIDIA або AMD з щонайменш 512 МБ VRAM

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2022-15.12.2022
Вивчення та аналіз завдання	15.12.2022-15.03.2023
Розробка архітектури та загальної структури системи	15.03.2023-25.03.2023
Розробка структур окремих частин системи	25.03.2023-5.04.2023
Програмна реалізація системи	5.04.2023-15.04.2023
Виправлення помилок	15.04.2023-20.04.2023
Оформлення пояснювальної записки	15.04.2023-05.06.2023

Пояснювальна записка
до дипломного проекту
на тему: «Система автопілотування літаків побудована на pid регуляторах»

Київ – 2023 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. РУД – регулятор управління двигуном.
2. ЗУС – загальне положення в системі.
3. ДСУ – допоміжна силова установка.
4. ПК – персональний комп’ютер
5. ЦП – центральний процесор
6. ЗПП - Злітно-посадкова смуга
7. БПЛА - Безпілотний літальний апарат
8. РЛС - Радіолокаційна станція
9. ТА - Турбінний двигун
10. ЦАП - Цифровий аналого-цифровий перетворювач

ВСТУП

Сучасна авіаційна індустрія стикається з необхідністю розвитку нових технологій та систем, які забезпечують високу рівень безпеки, ефективності та надійності під час польотів літаків. Одним із важливих напрямків розробки є автопілотування, яке дозволяє зменшити вплив людського фактору на процес пілотування та забезпечити точне виконання заданих маневрів та сценаріїв польоту.

Метою даного дипломного проекту є створення комплексу систем, який забезпечує виключення впливу факторів пілота та інших зовнішніх змінних параметрів при проведенні багаторазового пілотування. Головним завданням цієї системи є забезпечення повторюваності та надійності тестового процесу, що дозволяє гарантувати отримання точних результатів під час проходження заданих сценаріїв.

Одним з ключових елементів розробки цієї системи є використання PID-регуляторів, які є широко використовуваними в автоматичному керуванні та регулюванні систем. Вони дозволяють регулювати параметри керування залежно від відхилень від заданої точки, забезпечуючи стабільну та точну роботу системи автопілотування.

Основною перевагою цієї системи є здатність перевірити відповідність цифрового двійника справжній моделі літака. Це досягається шляхом використання тестових сценаріїв та порівняння отриманих результатів під час пілотування цифрової моделі з реальними даними, зібраними під час польотів справжнього літака. Такий підхід дозволяє перевірити точність та достовірність цифрового двійника, що має велике значення в розробці авіаційних систем.

У подальшому розвитку дипломного проекту будуть розглянуті детальні аспекти реалізації системи автопілотування на базі PID-регуляторів, включаючи проектування, моделювання, тестування та впровадження

системи. Результати реалізації проекту сприятимуть покращенню ефективності та безпеки авіаційних польотів, а також допоможуть в розвитку автоматизованих систем управління повітряним транспортом.

Таким чином, дана робота має на меті за рахунок використання системи автоматичного керування та тестів мати можливість перевірити відповідність цифрового двійника літака реальній моделі, провівши порівняння результатів польотів в симуляторі з реальними польотами літака, що в свою чергу дозволить проводити експерименти над літаком для отримання важливих даних, без ризику шкоди для літака та людей.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА ДОСЛІДЖЕННЯ

1.1 Опис задачі

Розробити систему автопілотування для цифрової моделі літака, яка буде використовуватися для проведення тестів та верифікації моделі. Основна задача полягає в тому, щоб цифрова модель літака, керована автопілотом, могла успішно пройти тести, що підтвердять її правильність та відповідність реальному літаку.

Система автопілотування буде розроблена з урахуванням вимог тестування і верифікації моделі. Кожен тест може мати свій власний автопілот, який буде контролювати літак та забезпечувати його рух відповідно до встановлених параметрів тесту. Його функціональність обмежується виконанням тестових сценаріїв та забезпеченням правильного проходження тестів.

Основні вимоги до системи автопілотування для тестування моделі включають:

1. Розробка програмного забезпечення, яке забезпечить взаємодію з цифровою моделлю літака та керування ним за допомогою автопілоту.
2. Створення тестових сценаріїв, які визначатимуть параметри руху літака під час тестування.
3. Реалізація механізмів валідації та перевірки результатів тестів для підтвердження правильності моделі літака.

Розроблена система автопілотування забезпечить контрольований рух цифрової моделі літака під час тестування та дозволить перевірити відповідність моделі справжньому літаку. Це дозволить отримати точні та

надійні результати тестів, що є важливим для підтвердження правильності та надійності моделі.

1.2 Аналоги

Одним з аналогічних проектів, спрямованих на валідацію та верифікацію цифрових моделей їх справжніми аналогами, є проект "Digital Twin Validation" виконавчої компанії Siemens. Цей проект використовує цифрові двійники промислових систем для перевірки правильності їх функціонування та ефективності.

Сильні сторони проекту "Digital Twin Validation" включають широкий спектр аналітичних інструментів та алгоритмів для порівняння даних з цифрових двійників з реальними даними, що дозволяє виявляти відхилення та неузгодженості. Використання цифрових двійників також дозволяє ефективно відстежувати та аналізувати стан системи в режимі реального часу [1].

Однак, слабкою стороною цього проекту може бути складність налаштування та калібрування цифрових двійників, особливо при комплексних системах. Крім того, залежно від складності системи, процес валідації та верифікації може зайняти значну кількість часу та ресурсів.

Інший приклад проекту, спрямованого на валідацію та верифікацію цифрових моделей, є "Digital Twin Verification and Validation" компанії NASA. Цей проект використовує цифрові двійники літаків та космічних апаратів для перевірки правильності їх проектування та функціонування в умовах атмосфери та космосу.

Сильною стороною проекту NASA є використання високоточних симуляторів та алгоритмів моделювання, які дозволяють детально відтворювати умови атмосферного та космічного середовища. Це дозволяє проводити реалістичні тестування та перевірку систем на відповідність заданим критеріям.

Однак, слабкою стороною цього проекту є висока вимогливість космічних проектів, валідація та верифікація можуть зайняти тривалий час та потребувати значних ресурсів.

Варто відзначити, що існують висококласні аналоги, які здатні виконувати схожі завдання, проте вони мають обмежений доступ і є закритими для використання військовими організаціями та компаніями-виробниками. Це може створювати значні перешкоди для дослідників та розробників, які прагнуть провести свої власні експерименти та верифікаційні тести.

Таким чином, дана робота заповнює цю прогалину, надаючи доступну та гнучку систему для верифікації та валідації цифрових моделей літаків. Цей підхід дозволяє дослідникам та розробникам незалежно від військових або комерційних обмежень проводити експерименти та тести, що сприяє розвитку та вдосконаленню авіаційної технології.

Підсумовуючи, дана робота має важливе значення в контексті доступу до інструментів верифікації та валідації для широкого спектру дослідників та розробників, забезпечуючи прозорий та відкритий підхід до перевірки цифрових моделей літаків.

Порівнюючи ці проекти з моїм, можна зазначити, що моя робота має перевагу в тому, що вона спрямована на розробку та тестування автопілота для цифрової моделі лише літака, що робить проект більш спеціалізованим та сфокусованим. У відміну від широкого спектра систем, що використовуються в проектах Siemens та NASA, яка охоплює різні промислові системи та космічні апарати, моя робота концентрується на конкретному об'єкті - автопілоті для літака. Це дозволяє мені глибше досліджувати особливості та вимоги автопілотів для літаків, враховуючи їх специфіку та важливість безпеки польоту. Крім того, розробка та тестування автопілота для конкретного об'єкта дозволяє мені використовувати оптимальні ресурси та ефективно зосереджуватися на досягненні поставлених цілей. Це дозволяє валідувати та верифікувати роботу автопілота в різних умовах та сценаріях, забезпечуючи високу стабільність та точність керування літаком. Завдяки

кількості тестів та умов для їх проходження можна чітко визначати ступінь відповідності цифрової моделі справжньому літаку.

Додатковою перевагою є можливість використання різних автопілотів для проведення тестів та забезпечення успішного проходження кожного тесту окремо, що прискорює розробку тестів, оскільки реалізація повноцінного автопілота, який може керувати літаком в будь-яких умовах є складною задачею та потребує не тільки велику кількість часу для виконання та й має високі вимоги до апаратної частини.

1.3 Вимоги до додатку

1.3.1 Функціональні вимоги

- Реалізувати алгоритми PID-регуляції для автоматичного керування параметрами польоту, зокрема швидкістю, висотою та кутом нахилу.
- Забезпечити стабільність та точність керування параметрами польоту, зокрема шляхом належного налаштування коефіцієнтів PID-регуляторів та оптимального використання даних з сенсорів.

1.3.2 Нефункціональні вимоги

- Забезпечити високу швидкодію та ефективність роботи додатку, щоб забезпечити плавність керування та мінімальні затримки в реакції системи.
- Забезпечити надійність та стабільність додатку, мінімізуючи можливість виникнення помилок та аварійних ситуацій.

Вимоги до додатку повинні бути відповідним чином документовані та визначені з метою забезпечення успішної розробки та впровадження системи автопілотування на основі PID-регуляторів.

1.4 Аналіз предметної області

Одним із основних аспектів предметної області є використання PID-регуляторів у системах автопілотування. PID-регулятори є широко застосовуваними в автоматичному керуванні і регулюванні систем, включаючи автопілотування літаків. Вони забезпечують стабільну та точну роботу системи шляхом використання пропорційної, інтегральної та диференціальної зворотного зв'язку. Аналіз різних методів та підходів до налаштування PID-регуляторів є важливим для досягнення оптимальних параметрів системи автопілотування. [2]

Крім того, управління факторами пілота та зовнішніми змінними параметрами є одним із викликів у розробці систем автопілотування. Велика кількість факторів, таких як погодні умови, рух повітряного трафіку, динамічні обмеження, можуть впливати на роботу автопілота. Аналіз існуючих методів та стратегій управління цими факторами допомагає забезпечити стабільність та надійність пілотування.

Також важливим аспектом предметної області є взаємодія цифрового двійника літака з реальною моделлю. Цифрові двійники є віртуальними моделями, які дозволяють симулювати та відтворювати реальну поведінку літака. Аналіз сучасних методів верифікації та валідації цифрових двійників допомагає забезпечити точність та достовірність результатів тестування.

Проведений аналіз предметної області надає загальний огляд сучасних підходів та методів, що використовуються в системах автопілотування літаків на основі PID-регуляторів та управління факторами пілота та зовнішніми змінними параметрами. Він створює базу для подальшого розроблення та вдосконалення комплексу систем, який буде забезпечувати повторюваність та надійність тестового процесу, а також перевіряти відповідність цифрового двійника реальній моделі літака.

1.5 Розробка вимог до характеристик об'єкта проектування

Вимоги до характеристик об'єкта проектування визначають його функціональні та технічні властивості, які повинні бути забезпечені для досягнення мети проекту.

1. Точність: Однією з головних вимог є досягнення високої точності системи автопілотування. Це означає, що система повинна забезпечувати точність в управлінні літаком, забезпечуючи точне виконання заданих курсів, висоти та швидкості.
2. Стабільність: Система автопілотування повинна бути стабільною та забезпечувати стійке управління літаком. Це означає, що система повинна швидко реагувати на зміни у вхідних параметрах та забезпечувати стійкий режим польоту без небажаних коливань або перенапружень.
3. Відновлюваність: У випадку виникнення відмов або помилок, система повинна мати механізми відновлення та автоматичного переключення на резервні компоненти. Це забезпечить безперебійну роботу системи та запобіжить можливим аварійним ситуаціям.
4. Надійність: Система повинна бути надійною та забезпечувати безперебійну роботу протягом тривалого періоду часу. Компоненти системи повинні бути стійкими до впливу зовнішніх факторів, а алгоритми управління повинні бути ефективними та надійними.
5. Сумісність: Система автопілотування повинна бути сумісною з існуючими системами та обладнанням, що встановлено на літаку. Вона повинна інтегруватися з іншими бортовими системами та забезпечувати взаємодію з ними без конфліктів або суперечностей.
6. Практичність: Система повинна бути практичною в експлуатації та обслуговуванні. Це означає, що вона повинна мати зрозумілий та зручний інтерфейс користувача, легкість налаштування та

супроводження, а також доступність запасних частин та технічної підтримки.

Розробка вимог до характеристик об'єкта проектування допомагає чітко визначити очікувані властивості та функціональність системи автопілотування на основі PID-регуляторів. Вони слугують основою для подальшого проектування та розробки системи, забезпечуючи досягнення мети проекту.

1.6 Висновки до розділу

Відповідно до проведеного дослідження та аналізу предметної області, можна впевнено говорити про актуальність і важливість задачі автопілотування в контексті верифікації та валідації цифрових моделей літака. У зв'язку з обмеженнями існуючих систем, стає очевидною потреба у створенні спеціалізованого комплексу, що максимально відповідає вимогам цієї сфери.

У цій роботі акцент зміщується з широкого спектра функцій на спеціалізований інструментарій, що спрямований на ефективність та точність. Націлений на конкретний об'єкт дослідження, він дає змогу зосередитися на досягненні оптимальних результатів у верифікації та валідації цифрових моделей.

Застосування такого підходу дозволяє збільшити гнучкість та ефективність використання автопілотів в контексті різних тестових сценаріїв. Основну цінність у цьому контексті представляє відкритість для експериментів з різними конфігураціями та алгоритмами, що сприяє пошуку оптимальних рішень для кожного конкретного випадку.

В результаті даної роботи ми маємо на меті створити новий інструмент для автоматизації авіаційних систем, який буде відповідати сучасним вимогам і особливостям авіації, а також даватиме можливість адаптуватися та експериментувати для різних завдань та цілей.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ

2.1 Огляд технологій та інструментів для розробки системи

2.1.1 Вибір мови та двигуна

Серед найбільш популярних комбінацій мов та двигунів є:

1. Python та FlightGear
2. C++ та Microsoft Flight Simulator
3. JavaScript та Microsoft Flight Simulator
4. Lua та X-Plane

Переваги комбінації Lua та X-Plane:

1. Простота розробки: Lua відома своєю простотою та легкістю використання. Вона має простий синтаксис і невелику кількість ключових слів, що дозволяє швидше освоювати мову та швидше розробляти програми. В порівнянні з, наприклад, C++, який є мовою нижчого рівня та має більшу складність, розробка на Lua може бути більш зручною та швидшою.
2. Швидкість розробки: Завдяки простоті та експресивності Lua, розробка на цій мові зазвичай займає менше часу порівняно з іншими мовами. Lua має підтримку динамічної типізації, автоматичного збирання сміття та інші функції, що спрощують процес розробки.
3. Інтеграція з X-Plane: Lua має вбудовану підтримку для інтеграції з X-Plane. Це означає, що ви зможете легко використовувати API та функціональні можливості, надані X-Plane, для розробки системи автопілотування. Ця інтеграція спрощує розробку та дозволяє вам використовувати реалістичні дані та поведінку польоту.
4. Підтримка спільноти розробників: Lua має активну та дружню спільноту розробників, яка надає підтримку, документацію та приклади коду. Ви

зможете отримати допомогу та поради від інших розробників, а також використовувати плагіни та інші ресурси, що створені спільнотою.

5. Легкість інтеграції з іншими системами: Lua має простий та гнучкий інтерфейс для інтеграції з іншими мовами програмування, такими як C++, які використовуються в інших частинах проекту. Ви можете легко обмінюватись даними та функціями між Lua та C++, що дозволяє використовувати потужності обох мов.

Загалом, комбінація Lua та X-Plane має свої переваги у вигляді простоти використання, швидкодії, можливості інтеграції з симулятором та підтримки спільноти розробників. Враховуючи ваші потреби та вимоги до диплому, ця комбінація може бути вигідним вибором для розробки системи автопілотування.

2.1.2 Мова Lua

2.1.2.1 Історія та походження мови

Lua, названа на честь португальського слова "місяць", з'явилась в 1993 році як інструмент для розширення програмного забезпечення. Розроблена в Понтіфікському католицькому університеті Ріо-де-Жанейро, вона є продуктом масштабного дослідження в області мов програмування.[3]

2.1.2.2 Основні властивості та характеристики Lua

Lua, яка була створена з метою об'єднання потужності мови програмування з простотою налаштування, представляє собою динамічну мову програмування[3], основні характеристики якої включають наступне:

1. Легкість інтеграції: Lua була спроектована для використання в якості вбудованої мови, з метою розширення програм на C/C++. Lua взаємодіє

з C через простий API, що не вимагає ніяких спеціальних знань з обох мов.

2. Мультипарадигменне програмування: Lua підтримує кілька парадигм програмування, включаючи процедурне, об'єктно-орієнтоване та функціональне програмування, що дає розробникам значну гнучкість.
3. Метапрограмування: Lua надає потужні засоби для метапрограмування, включаючи "метатаблиці", які дозволяють змінювати поведінку об'єктів, і "метаметоди" для перевизначення поведінки операторів.
4. Продуктивність: Lua відома своєю високою продуктивністю, особливо як для мови високого рівня. LuaJIT, Just-In-Time компілятор для Lua, надає виконання коду на швидкостях, порівнянні з виконанням оптимізованого C коду.
5. Відкритий вихідний код: Lua є вільною та відкритою мовою програмування, що дозволяє розробникам вільно її використовувати, модифікувати та поширювати. Lua розробляється та підтримується широкою спільнотою.
6. Портативність: Lua може працювати на багатьох платформах та операційних системах без необхідності змін у коді.
7. Маленький розмір: Ядро Lua відносно невелике, що робить її особливо підходящою для вбудованих систем.
8. Гнучкість: Lua надзвичайно гнучка завдяки своїм потужним засобам метапрограмування та можливості виконання коду на льоту ("eval" функція). Lua також дозволяє створювати анонімні функції та використовувати замикання, що надає розробникам велику свободу при створенні складних програмних структур.
9. Автоматичне управління пам'яттю: Lua забезпечує автоматичне управління пам'яттю (систему сміттєзбірника), що звільняє розробників від необхідності ручного керування пам'яттю, що допомагає зменшити кількість помилок і поліпшує продуктивність.

10. Підтримка корутин: Lua підтримує корутини, які дозволяють реалізовувати неблокуючі операції та створювати декілька потоків виконання всередині одного процесу. Ця функція може бути особливо корисною для розробки систем, які вимагають високого рівня паралелізму або асинхронного виконання.
11. Підтримка єдиного типу даних для колекцій: В Lua єдиний тип даних "таблиця" використовується для представлення масивів, множин, списків, словарів і т.д. Це спрощує використання структур даних і підвищує гнучкість мови.

Ці характеристики роблять Lua потужним інструментом для розробки складних систем, і ідеально підходять для використання в контексті цього проекту.

2.1.2.3 Lua в контексті автопілотування

В контексті автопілотування, мова програмування Lua має декілька переваг та застосувань. Основна перевага Lua полягає в його легкості та простоті використання, що робить його ідеальним вибором для реалізації автопілотного програмного забезпечення.

Завдяки своєму скриптовому характеру, Lua може бути використаний для реалізації гнучких та модульних систем автопілотування. Він дозволяє швидко створювати та тестувати нові алгоритми та логіку без необхідності компіляції коду, що спрощує розробку та експериментування з різними підходами.

Крім того, Lua підтримує механізми для легкої взаємодії з іншими мовами програмування та системами. Це означає, що Lua може бути використана для інтеграції з існуючими системами автопілотування, додатками або платформами, що спрощує розширення функціональності та використання існуючих ресурсів.

Одним з важливих аспектів використання Lua в автопілотних системах є його швидкодія. Lua відома своєю ефективністю та низьким споживанням ресурсів, що робить його ідеальним вибором для реалізації розрахункової логіки автопілота, де важлива швидкість обробки сигналів та прийняття рішень.

Загалом, використання мови програмування Lua в системі автопілотування надає багато переваг, включаючи легкість використання, гнучкість, модульність та ефективність. Ця мова дозволяє розробникам створювати складні алгоритми та логіку

2.1.2.4 Lua і X-Plane

Lua і X-Plane доповнюють один одного в контексті симуляції і тестування систем автопілотування.

X-Plane - це потужний програмний продукт для симуляції польотів, який надає детальне і реалістичне моделювання різних аспектів польоту, включаючи аеродинаміку, погодні умови, навігацію, системи літака і багато іншого. X-Plane є широко використовуваним інструментом в авіаційній промисловості та наукових дослідженнях, завдяки своїй точності, гнучкості і розширюваності.

Lua, як було описано вище, є гнучкою, потужною та високопродуктивною мовою програмування, яка підтримує різні парадигми програмування і проста в освоєнні.

Lua і X-Plane можуть бути інтегровані за допомогою модуля для X-Plane, відомого як SASL. Цей модуль дозволяє розробникам писати сценарії Lua, які взаємодіють з X-Plane, даючи їм можливість автоматизувати багато аспектів симуляції польоту, включаючи управління системами літака, використання навігаційних систем і імітацію дій пілота.

У контексті системи автопілотування, Lua може бути використана для програмування логіки автопілоту, тоді як X-Plane може бути використаний для

тестування цієї логіки в реалістичних умовах польоту. Деталі авіоніки та показники польоту можуть бути зчитувані та впливати на них за допомогою Lua, що дозволяє створити складні сценарії тестування та виконати їх автоматично. Це робить комбінацію Lua і X-Plane особливо вартісною для розробки і тестування систем автопілотування.

2.1.2 Двигун X-Plane

2.1.2.1 Загальний огляд X-Plane

X-Plane є одним з найпопулярніших симуляторів польотів, який широко використовується як в авіаційній промисловості, так і у галузі симуляції польотів. Він відомий своєю високою реалістичністю, точністю моделювання та широким спектром функціональних можливостей[4]. Основні характеристики та застосування X-Plane в авіаційній промисловості і симуляції польотів включають:

1. Реалістична модель польоту: X-Plane відомий своєю точністю моделювання польоту. Він використовує фізичні принципи та аеродинамічні моделі для створення реалістичного відтворення поведінки літаків у повітрі. Це дозволяє пілотам, розробникам та іншим зацікавленим особам вивчати та вдосконалювати навички польоту.
2. Широкий вибір літальних апаратів: X-Plane надає доступ до великої кількості літальних апаратів, включаючи різні типи літаків, вертольоти, планери та інші повітряні судна. Це дозволяє користувачам відтворювати польоти на різних типах та моделях літаків, розширюючи їх можливості для навчання, тренування та випробувань.
3. Графічне середовище високої якості: X-Plane володіє вражаючими графічними можливостями, що створюють реалістичне зображення ландшафту, аеропортів, погодних умов та інших деталей. Це допомагає

створювати гарне середовище для польотів і забезпечує більшу візуальну реалістичність.

4. Розширення та модифікації: X-Plane надає можливість розширювати його функціональність та модифікувати за допомогою плагінів, сценаріїв та інших засобів розробки. Це дає користувачам можливість створювати власні додатки, вдосконалювати функції та досліджувати нові можливості.
5. Використання в авіаційній промисловості: X-Plane також використовується в авіаційній промисловості для тренування пілотів, випробування нових концепцій та технологій, а також для вивчення польотних характеристик та безпеки. Його точність моделювання та розширені можливості роблять його популярним інструментом у цій галузі.

2.1.2.2 Архітектура X-Plane

Архітектура X-Plane може бути розглянута як система, що складається з декількох основних модулів, які взаємодіють між собою для забезпечення реалістичної симуляції польотів[4]. Основні модулі X-Plane включають:

1. Модуль графіки: Цей модуль відповідає за візуалізацію сцени польоту, включаючи ландшафт, об'єкти, аеропорти, хмари та ефекти погоди. Він забезпечує реалістичну графіку та деталізацію.
2. Модуль фізики: Цей модуль відповідає за моделювання фізичних аспектів польоту, включаючи аеродинаміку, динаміку руху, системи керування та реакцію на зовнішні умови. Він враховує параметри літака, такі як маса, обтік повітря, сили тяжіння та тяги, що дозволяє точно відтворити поведінку літального апарату.
3. Модуль системи керування: Цей модуль відповідає за керування параметрами польоту, такими як курс, крен, альтитуда, швидкість тощо.

Він виконує обчислення та встановлює відповідні керувальні сигнали для пілотажних поверхонь та системи приводу.

4. Модуль моделі літака: Цей модуль включає цифрову модель літака, яка детально описує його характеристики, системи, поведінку та реакцію на вхідні сигнали. Він використовує різні фізичні та математичні моделі для реалістичного моделювання літального апарату.

Ці модулі взаємодіють між собою, обмінюючи необхідну інформацію для забезпечення координованої симуляції польоту. Наприклад, модуль графіки отримує дані про положення та стан літака від модуля фізики, щоб коректно відобразити його на екрані. Модуль системи керування взаємодіє з модулем моделі літака, передаючи керувальні сигнали та отримуючи відповідні дані про стан літака.

Архітектура X-Plane покликана забезпечити ефективну та реалістичну симуляцію польотів, використовуючи взаємодію між різними модулями для досягнення максимального рівня точності.

2.1.2.3 Моделювання польотів в X-Plane

Моделювання польотів в X-Plane забезпечує реалістичне відтворення різних аспектів польоту, що дозволяє пілотам, навчальним закладам та дослідникам отримати цінний досвід та провести реалістичні тренування[4]. Деякі основні аспекти моделювання польотів в X-Plane включають:

1. Динаміка польоту: X-Plane використовує реалістичні фізичні моделі для відтворення динаміки польоту. Це включає моделювання аеродинаміки, керування повітряним судном, реакції на зміни вхідних сигналів, таких як керування педалями, кермом та розміщенням пілотажних поверхонь. Всі ці аспекти співпрацюють для створення реалістичної поведінки літального апарату в різних умовах польоту.
2. Погода: X-Plane дозволяє моделювати різні погодні умови, включаючи хмарність, опади, вітер, турбулентність та інші атмосферні явища. Це

дозволяє пілотам відчувати вплив погодних умов на політ та реагувати на них відповідним чином. Моделювання погоди в X-Plane допомагає зробити симуляцію польоту ще більш реалістичною.

3. Навігація: X-Plane включає системи навігації, що відтворюють реальні прилади та системи, які використовуються в авіації. Це включає навігаційні прилади, аеронавігаційні системи, радіонавігаційні засоби та багато іншого. Відтворення цих систем дозволяє пілотам тренувати навігаційні навички та виконувати складні процедури польоту.
4. Аеропорти та територія: X-Plane включає детально відтворені аеропорти та місцевість, що дозволяє пілотам отримати реалістичне відчуття польоту в різних місцевостях. Детально пророблені аеропорти, злітно-посадкові смуги, зони міської забудови та природні об'єкти створюють відчуття присутності та допомагають виконувати точні маневри та процедури польоту.

Загальне моделювання польотів в X-Plane дозволяє створити високоякісну симуляцію польоту з реалістичною фізикою, погодою, навігацією та деталізованою місцевістю, що дає пілотам змогу отримати цінний досвід та тренуватися у різних сценаріях польоту.

2.1.3 Шаблони програмування

2.1.3.1 Адаптер

Шаблон програмування "Адаптер" є структурним шаблоном проектування, який дозволяє об'єднати різні несумісні інтерфейси в одному класі. Цей шаблон дозволяє об'єктам з різними інтерфейсами спілкуватися між собою, що робить його особливо корисним при інтеграції нового коду з вже існуючим.

Основна ідея шаблону "Адаптер" полягає в тому, що він використовує проміжний об'єкт, відомий як адаптер, для взаємодії між двома іншими

об'єктами. Адаптер містить логіку перетворення запитів одного інтерфейсу на запити до іншого інтерфейсу.

Використання шаблону "Адаптер" дозволяє вирішити проблему несумісності між двома інтерфейсами. Основні проблеми, які можуть виникати і призводити до необхідності використання адаптера, включають:

1. Несумісність інтерфейсів: Коли два класи або компоненти мають різні інтерфейси, виникає проблема з їх взаємодією. Наприклад, класи можуть мати різні назви методів, різні типи параметрів чи різні способи повернення результатів. Використання адаптера дозволяє забезпечити злагоджену взаємодію між цими класами, перетворюючи один інтерфейс на інший.
2. Використання старого коду з новим кодом: Коли вам потрібно інтегрувати новий код або компонент у вже існуючу систему, може виникнути проблема несумісності інтерфейсів. Новий код може використовувати інші структури даних або інші способи виконання операцій. Використання адаптера дозволяє вам інтегрувати новий код, перетворюючи його інтерфейс на інтерфейс, який відповідає вимогам системи.
3. Повторне використання існуючого коду: Часто вам може знадобитись використати існуючий код або компонент, який вже має визначений інтерфейс, у новому контексті, де вимагається інший інтерфейс. Використання адаптера дозволяє вам перефразувати існуючий код, роблячи його сумісним з новим контекстом через адаптерний шар.
4. Залежність від сторонніх бібліотек або інших систем: Коли ви використовуєте сторонню бібліотеку або систему, яка має свій власний інтерфейс, виникає проблема взаємодії з цими компонентами. Використання адаптера дозволяє вам створити проміжний шар, який перетворює інтерфейс сторонньої системи на інтерфейс, який більш зручний для вашої системи.

Використання адаптера допомагає забезпечити гнучкість, розширюваність та підтримку взаємодії між компонентами, що мають різні інтерфейси. Він дозволяє вам уникнути необхідності внесення змін у вже існуючий код і забезпечує прозору інтеграцію між різними компонентами системи.

Важливо враховувати, що шаблон "Адаптер" має різні варіації, такі як "Класичний адаптер" та "Об'єктний адаптер". У кожній варіації є власні особливості та переваги, тому оберіть підходящий для вашої конкретної ситуації.

Класичний адаптер

Класичний адаптер - це одна з варіацій шаблону "Адаптер" і передбачає використання успадкування для забезпечення взаємодії між клієнтом і адаптованим класом. У цій варіації адаптер використовує множинне успадкування для отримання властивостей як цільового інтерфейсу, так і адаптованого класу.

Для реалізації класичного адаптера виконайте такі кроки:

1. Створіть цільовий інтерфейс: Визначте інтерфейс, який клієнти будуть використовувати для взаємодії з адаптером. Цей інтерфейс має методи, які відповідають потребам клієнта.
2. Створіть адаптований клас: Створіть клас, який вже існує і має несумісний інтерфейс з цільовим інтерфейсом. Цей клас містить функціональність, яку ви хочете використовувати через адаптер.
3. Створіть клас адаптера: Створіть новий клас, який успадковує цільовий інтерфейс і одночасно має залежність від адаптованого класу. Він використовує успадковані методи цільового інтерфейсу та реалізує їх шляхом виклику відповідних методів адаптованого класу.
4. Реалізуйте методи цільового інтерфейсу в адаптері: Реалізуйте методи, визначені в цільовому інтерфейсі, в класі адаптера. У цих методах використовуйте відповідні методи адаптованого класу для забезпечення необхідної функціональності.

5. Встановіть зв'язок між клієнтом та адаптером: Клієнт повинен використовувати об'єкт адаптера через цільовий інтерфейс. Завдяки успадковуванню, клієнт може взаємодіяти з адаптером так само, як і з будь-яким іншим об'єктом, що реалізує цільовий інтерфейс.
6. Тестування та налагодження: Переконайтеся, що клієнт успішно взаємодіє з адаптером і отримує очікувані результати.

Класичний адаптер дозволяє клієнтам прозоро взаємодіяти з адаптованим класом, навіть не знаючи про його існування. Він також дозволяє легко замінювати адаптовані класи, не змінюючи клієнтський код, оскільки клієнт спілкується через цільовий інтерфейс.

Об'єктний адаптер

Об'єктний адаптер є іншою варіацією шаблону "Адаптер" і передбачає використання композиції замість успадкування для забезпечення взаємодії між клієнтом і адаптованим класом. У цій варіації адаптер використовує об'єкт адаптованого класу як одну зі складових частин адаптера.

Для реалізації об'єктного адаптера виконайте такі кроки:

1. Створіть цільовий інтерфейс: Визначте інтерфейс, який клієнти будуть використовувати для взаємодії з адаптером. Цей інтерфейс повинен містити методи, які відповідають потребам клієнта.
2. Створіть адаптований клас: Створіть клас, який вже існує і має несумісний інтерфейс з цільовим інтерфейсом. Цей клас містить функціональність, яку ви хочете використовувати через адаптер.
3. Створіть клас адаптера: Створіть клас адаптера, який реалізує цільовий інтерфейс і містить об'єкт адаптованого класу як свою складову частину. Це може бути досягнуто шляхом композиції адаптованого об'єкта всередині адаптера.
4. Реалізуйте методи цільового інтерфейсу в адаптері: Реалізуйте методи, визначені в цільовому інтерфейсі, в класі адаптера. У цих методах ви викликаєте відповідні методи адаптованого об'єкта для виконання необхідних операцій.

5. Встановіть зв'язок між клієнтом та адаптером: Клієнт повинен використовувати об'єкт адаптера через цільовий інтерфейс. Переконайтеся, що клієнт взаємодіє з адаптером та використовує методи цільового інтерфейсу для виконання потрібних операцій.
6. Тестування та налагодження: Переконайтеся, що клієнт успішно взаємодіє з адаптером і отримує очікувані результати.

Об'єктний адаптер дозволяє динамічно замінювати адаптовані класи, так як адаптер скомпонований з об'єктом адаптованого класу. Він також дозволяє клієнту спілкуватися з адаптером через цільовий інтерфейс, забезпечуючи абстракцію та незалежність від конкретного адаптованого класу.

2.1.3.2 Стратегія

Шаблон "Стратегія" є поведінковим шаблоном проектування, який дозволяє вибирати один з алгоритмів або стратегій залежно від потреби і використовувати його гнучко в контексті без змін коду контексту. Цей шаблон дозволяє розділити поведінку від контексту і робить її замінюваною та незалежною.

Основна ідея шаблону "Стратегія" полягає в тому, що він визначає різні алгоритми або стратегії як окремі класи, які реалізують спільний інтерфейс. Контекст має залежність від цього інтерфейсу і може динамічно замінювати конкретну стратегію, не змінюючи свій власний код.

Шаблон "Стратегія" вирішує проблему залежності алгоритмів від контексту та необхідності зміни поведінки виконання алгоритмів без зміни коду контексту. Основні проблеми, які вирішує цей шаблон, включають:

1. Гнучкість вибору алгоритму: Шаблон "Стратегія" дозволяє вибирати алгоритм або стратегію з декількох доступних варіантів, що відповідають певним умовам або потребам. Це дає можливість динамічно змінювати алгоритм виконання без необхідності модифікувати контекст.

2. Розділення поведінки від контексту: Шаблон "Стратегія" дозволяє розділити реалізацію алгоритмів від коду контексту. Це сприяє поліпшенню структури програми та забезпечує більшу зрозумілість та підтримку коду. Контекст не залежить від конкретних реалізацій алгоритмів і може працювати з будь-яким об'єктом стратегії, що реалізує відповідний інтерфейс.
3. Легка заміна алгоритму: Використання шаблону "Стратегія" дозволяє легко замінити алгоритм виконання без зміни коду контексту. Достатньо просто підключити новий клас стратегії, що реалізує відповідний інтерфейс, і встановити його у контексті. Це особливо корисно, коли потрібно розширювати або модифікувати поведінку системи без переписування великого обсягу коду.
4. Зменшення дублювання коду: Завдяки шаблону "Стратегія" можна уникнути дублювання коду, якщо різні алгоритми мають спільну частину реалізації. Замість повторного кодування цієї спільної логіки для кожного алгоритму, вона може бути винесена в окрему стратегію, що реалізується усіма алгоритмами, і використовуватись повторно.

Загалом, шаблон "Стратегія" забезпечує більшу гнучкість, розширюваність та підтримку виконання різних алгоритмів в програмі, що веде до поліпшення структури коду та спрощення його модифікації.

Основні складові шаблону "Стратегія" такі:

1. Контекст (Context): Це клас, який має залежність від інтерфейсу стратегії і використовує його для виконання потрібних операцій. Контекст не знає про конкретну стратегію, але використовує загальний інтерфейс для взаємодії зі стратегією.
2. Стратегія (Strategy): Це інтерфейс або абстрактний клас, який визначає спільний інтерфейс для всіх конкретних стратегій. Він містить методи, які контекст використовує для взаємодії зі стратегією.
3. Конкретні стратегії (Concrete Strategies): Це класи, які реалізують інтерфейс стратегії і визначають конкретні алгоритми або стратегії.

Кожна конкретна стратегія виконує певні дії, які можуть варіюватись залежно від потреби.

Процес використання шаблону "Стратегія" включає наступні кроки:

1. Визначення інтерфейсу стратегії: Спочатку ви визначаєте інтерфейс або абстрактний клас, який визначає спільні методи для всіх стратегій.
2. Реалізація конкретних стратегій: Далі ви створюєте класи, які реалізують інтерфейс стратегії і визначають конкретні алгоритми або стратегії.
3. Реалізація контексту: Створіть клас контексту, який має залежність від інтерфейсу стратегії. В контексті ви використовуєте методи інтерфейсу стратегії для виконання потрібних операцій.
4. Встановлення зв'язку між контекстом і стратегією: В контексті повинен бути механізм для встановлення вибраної стратегії. Зазвичай це реалізується через сеттер або конструктор, який дозволяє встановити об'єкт конкретної стратегії.
5. Використання стратегії: Контекст може викликати методи стратегії для виконання певних операцій. Залежно від встановленої стратегії, контекст буде використовувати відповідну реалізацію алгоритму.

Шаблон "Стратегія" дозволяє легко змінювати алгоритми або стратегії без необхідності внесення змін у контекст. Він допомагає забезпечити гнучкість, розширюваність та підтримку різних варіантів поведінки в програмі.

2.2 Висновки до розділу

На основі розглянутих у даному розділі технологій та інструментів, можемо стверджувати, що ми знаходимося на правильному шляху до розробки ефективної та надійної системи автопілотування. Вибір мови програмування Lua, що надає простоту та гнучкість, та двигуна X-Plane, з його високим рівнем точності моделювання польотів, вказують на прагнення до максимальної автентичності та реалістичності при моделюванні.

Додатково, використання шаблонів програмування Адаптер і Стратегія підкреслює прагнення до створення структурованого, модифікованого коду, що забезпечує гнучкість у відповіді на потреби системи автопілотування.

Таким чином, усі розглянуті інструменти та технології спільно формують сильну платформу для подальшого проектування додатку, враховуючи вимоги, що були встановлені в першому розділі.

РОЗДІЛ 3. PID-РЕГУЛЯТОРИ

3.1 Принцип роботи PID-регулятора

3.1.1 Пропорційна складова

Пропорційна складова (P) є однією з ключових складових PID-регулятора і відіграє важливу роль у процесі керування. Вона базується на величині поточної помилки між виміряним значенням системи (процесу) і бажаним значенням (заданим значенням). [5]

Робота пропорційної складової полягає у тому, що чим більша помилка, тим більша впливу має ця складова на вихідний сигнал регулятора. Вона забезпечує пропорційну реакцію системи на зміни і допомагає регулювати вихідний сигнал пропорційно до величини помилки.

У роботі з пропорційною складовою важливо визначити коефіцієнт пропорційності (Kp), який визначає, наскільки сильно реагує система на помилку. Величина цього коефіцієнта впливає на швидкість реакції системи і стабільність регулятора. Занадто велике значення Kp може спричинити перевищення (overshoot) і недостатню стабільність, тоді як занадто мале значення може привести до повільної реакції і недостатнього коригування помилки.[5]

Пропорційна складова допомагає системі швидко реагувати на зміни, але може призводити до появи статичної помилки. Це означає, що в деяких випадках, коли помилка стає мала, пропорційна складова не здатна забезпечити повністю точне регулювання і може залишатися невелика помилка між виміряним значенням і бажаним значенням.

При налаштуванні пропорційної складової важливо знати характеристики самої системи, її відгуки на зміни вхідних сигналів та вимоги до стабільності і точності. Це дозволить встановити оптимальне значення

коефіцієнта пропорційності для досягнення найкращої регуляції системи з мінімальною помилкою.

Отже, пропорційна складова є важливим елементом PID-регулятора, який дозволяє системі реагувати пропорційно до величини помилки. Її правильна настройка є ключем до швидкої реакції системи і забезпечення необхідної стабільності та точності регулювання.

3.1.2 Інтегральна складова

Інтегральна складова (I) є однією з основних складових PID-регулятора і відіграє важливу роль у забезпеченні точності регулювання системи. Її основна функція полягає у компенсації статичної помилки, яка може виникнути при наявності постійного зсуву між виміряним значенням і бажаним значенням системи.

Робота інтегральної складової базується на сумуванні інтегралу від помилки в часі. Чим довше триває помилка, тим більший вплив має інтегральна складова на вихідний сигнал регулятора. Це дозволяє системі поступово зменшувати помилку і наближатися до бажаного значення.

У роботі з інтегральною складовою важливо визначити коефіцієнт інтегральності (Ki), який визначає, наскільки швидко система реагує на кумулятивну помилку. Величина цього коефіцієнта впливає на швидкість реакції системи і стабільність регулятора. Занадто велике значення Ki може спричинити перевищення (overshoot) і нестабільність системи, тоді як занадто мале значення може призвести до повільної реакції і недостатнього коригування помилки.

Інтегральна складова дозволяє системі досягти точного регулювання, особливо при наявності статичної помилки. Вона накопичує вплив попередніх помилок і забезпечує постійне коригування системи для досягнення бажаного значення.

При налаштуванні інтегральної складової важливо враховувати характеристики самої системи, її часову постійну, стабільність та чутливість до змін. Також потрібно уникати переналаштування інтегральної складової, оскільки це може призвести до нестабільності і появи коливань системи.

Отже, інтегральна складова є важливим елементом PID-регулятора, який допомагає компенсувати статичну помилку і забезпечує точне регулювання системи. Правильне налаштування коефіцієнта інтегральності (Ki) дозволяє досягти оптимальної швидкості реакції і стабільності системи.

3.1.3 Диференціальна складова

Диференціальна складова (D) є третьою основною складовою PID-регулятора і відіграє важливу роль у покращенні стабільності і динаміки системи. Її головна функція полягає у прогнозуванні зміни помилки регулювання та використанні цієї інформації для коригування вихідного сигналу регулятора.

Робота диференціальної складової базується на обчисленні похідної від помилки регулювання по часу. Це дозволяє виявляти швидкі зміни помилки і включати компенсаційні заходи, що допомагають уникнути різких коливань системи і забезпечити більш плавну реакцію на зміни вхідних параметрів.

Диференціальна складова дозволяє системі реагувати на зміну помилки швидше, а також зменшує чутливість до шумів та випадкових збурень. Це особливо корисно в системах, де швидкість реакції є важливою, наприклад, у системах автоматичного керування, де потрібно швидко коригувати помилку для забезпечення точності і стабільності.

Для ефективної роботи диференціальної складової необхідно визначити коефіцієнт диференціювання (Kd), який контролює масштаб впливу цієї складової на вихідний сигнал регулятора. Занадто велике значення Kd може призвести до появи збурень і нестабільності системи, тоді як занадто мале

значення може призвести до повільної реакції на зміни і недостатньої корекції помилки.

Отже, диференціальна складова відіграє важливу роль у забезпеченні стабільності та динаміки системи. Правильне налаштування коефіцієнта диференціювання (Kd) дозволяє досягти оптимальної швидкості реакції системи і уникнути зайвих коливань.

3.1.4 Вагові коефіцієнти

Вагові коефіцієнти є додатковими параметрами, які використовуються в PID-регуляторі для налаштування впливу кожної складової (пропорційної, інтегральної, диференціальної) на вихідний сигнал регулятора. Вони дозволяють балансувати і налаштовувати поведінку системи залежно від її особливостей та вимог.

Кожен складовий член PID-регулятора має свій ваговий коефіцієнт, який визначає його вплив на вихідний сигнал. Загальний вихідний сигнал регулятора обчислюється шляхом змішування складових з врахуванням їх вагових коефіцієнтів.

Вагові коефіцієнти можуть бути налаштовані вручну або за допомогою автоматичних методів налаштування, таких як метод Зіглера-Ніколсона або інші методи оптимізації. Налаштування вагових коефіцієнтів вимагає досвіду та експертного знання, оскільки вони впливають на реакцію системи на зміни та її стабільність.

Вагові коефіцієнти дозволяють вибирати, яку складову регулятора пріоритетніше використовувати при корекції помилки. Збільшення вагового коефіцієнта для певної складової підсилює її вплив на вихідний сигнал регулятора, тоді як зменшення вагового коефіцієнта зменшує її вплив.

Правильне налаштування вагових коефіцієнтів дозволяє досягти оптимальної реакції системи на зміни та уникнути нежаданої нестабільності або надмірних коливань. Вони дозволяють регулювати та контролювати вплив

кожної складової в залежності від потреб і вимог системи, що допомагає досягти бажаних результатів регулювання.

3.2 Розрахунок вагових коефіцієнтів

Для розрахунку вагових коефіцієнтів складових PID-регулятора (пропорційної, інтегральної, диференціальної) існують різні методи та формули, які можна застосувати. Ось кілька загальновідомих методів:

3.2.1 Метод Зіглера-Ніколсона

Метод Зіглера-Ніколсона (також відомий як метод пропорційного - інтегрального - похідного (П-І-Д) коефіцієнтів) є одним з методів налаштування параметрів П-І-Д контролерів в системах автоматичного керування.

Цей метод був розроблений Джоном Зіглером і Натаніелем Ніколсоном у 1942 році. Він базується на знаходженні оптимальних значень коефіцієнтів П-І-Д контролера, щоб досягти бажаної динаміки системи.

Процедура налаштування методом Зіглера-Ніколсона включає наступні кроки:

1. Першим кроком є встановлення контролера у режимі пропорційного регулювання з коефіцієнтом пропорційності K_p. Значення K_p вибирається достатньо великим, щоб забезпечити достатню швидкість реакції системи.
2. Далі проводиться оцінка осциляторних характеристик системи. Це включає встановлення початкового значення коефіцієнта інтегрального впливу K_i та коефіцієнта похідного впливу K_d на нуль. Значення K_i та K_d обираються досить малими (наприклад, K_i = K_d = 0).

3. Відразу після встановлення початкових значень коефіцієнтів виконується перехід до кроку 4, де системі надається деяка збурююча дія, така як скачок керуючого сигналу або зміна вхідного сигналу.
4. Спостерігається вихідна відповідь системи і визначаються період коливань (T_u) та амплітуда коливань (P_u). Період коливань - це час між сусідніми максимумами або мінімумами відповіді системи, а амплітуда коливань - різниця між значеннями пікових точок.
5. На основі виміряних значень періоду та амплітуди коливань обчислюються оптимальні коефіцієнти для П-І-Д контролера:
    - Коефіцієнт пропорційності K_p: K_p = 0.6 * Ku
    - Коефіцієнт інтегрального впливу K_i: K_i = 1.2 * Ku / Tu
    - Коефіцієнт похідного впливу K_d: K_d = 0.6 * Ku * Tu / 8, де Ku - апробаційний коефіцієнт (Ku = P_u / A_u) і Tu - період коливань.
6. Контролер налаштований. Якщо результати не є задовільними, можна повторити процедуру, змінюючи початкові значення коефіцієнтів та проводячи додаткові експерименти.

Метод Зіглера-Ніколсона є простим у використанні і дозволяє швидко налаштувати коефіцієнти П-І-Д контролера без потреби в математичних моделях системи. Проте, його застосування може бути обмеженим в складних системах з нестійкими або нелінійними динаміками, де можуть знадобитися інші методи налаштування контролера.

3.2.2 Метод оптимальної реакції на ступінь

Метод оптимальної реакції на ступінь (також відомий як метод Заде) є одним з методів налаштування параметрів П-І-Д контролерів в системах автоматичного керування.

Цей метод був розроблений в 1940-х роках. Він базується на аналізі реакції системи на ступінь збурення і використовується для визначення оптимальних значень коефіцієнтів П-І-Д контролера.

Процедура налаштування методом оптимальної реакції на ступінь включає наступні кроки:

1. Першим кроком є встановлення контролера у режимі пропорційного регулювання з коефіцієнтом пропорційності K_p. Значення K_p вибирається достатньо великим, щоб забезпечити достатню швидкість реакції системи.
2. Далі проводиться збурення системи, шляхом введення ступінчатого сигналу або зміни вхідного сигналу.
3. Спостерігається вихідна відповідь системи та вимірюється час перехідного процесу T_r - час, за який система досягає 90-95% свого усталеного стану.
4. На основі виміряного часу перехідного процесу обчислюються оптимальні коефіцієнти для П-І-Д контролера:
    - Коефіцієнт пропорційності K_p: K_p = 1 / (K_u * T_r),
    - Коефіцієнт інтегрального впливу K_i: K_i = K_p / (2 * T_u),
    - Коефіцієнт похідного впливу K_d: K_d = K_p * T_u / 8, де K_u - апробаційний коефіцієнт, який залежить від типу контрольованого об'єкта, а T_u - час періоду коливань системи без контролера.
5. Контролер налаштований. Якщо результати не є задовільними, можна повторити процедуру з різними значеннями K_p та проводити додаткові експерименти.

Метод оптимальної реакції на ступінь є простим у використанні і не потребує математичних моделей системи. Він дозволяє швидко налаштувати коефіцієнти П-І-Д контролера, але його застосування може бути обмеженим в складних системах з нестійкими або нелінійними динаміками. У таких випадках можуть знадобитися інші методи налаштування контролера.

3.2.3 Метод Заде (Ziegler-Nichols)

Метод Заде (також відомий як метод Зіглера-Ніколсона або метод часу спрацювання) є одним з найпоширеніших методів налаштування П-І-Д контролерів в системах автоматичного керування. Цей метод був розроблений в 1942 на основі раніше використаного методу Зіглера-Ніколсона.

Метод Заде базується на аналізі реакції системи на ступінь збурення і використовується для визначення оптимальних значень коефіцієнтів П-І-Д контролера.

Процедура налаштування методом Заде включає наступні кроки:

1. Початковим кроком є встановлення контролера у режимі пропорційного регулювання зі значенням коефіцієнта пропорційності K_p, рівним нулю (K_p = 0).
2. Далі проводиться збурення системи шляхом введення ступінчатого сигналу або зміни вхідного сигналу.
3. Спостерігається вихідна відповідь системи та вимірюється час спрацювання процесу T_u - час, за який система досягає свого усталеного стану (наприклад, час, за який відхилення стає сталою величиною).
4. На основі виміряного часу спрацювання обчислюються оптимальні коефіцієнти для П-І-Д контролера:
    - Коефіцієнт пропорційності K_p: K_p = 0.6 / Ku,
    - Коефіцієнт інтегрального впливу K_i: K_i = 1.2 / (Ku * Tu),
    - Коефіцієнт похідного впливу K_d: K_d = 0.6 * Ku * Tu / 8, де Ku - апробаційний коефіцієнт (Ku) визначається як граничне значення коефіцієнта пропорційності, при якому система починає виявляти нестійкість, а Tu - час спрацювання процесу.

5. Контролер налаштований. Якщо результати не задовольняють вимоги, можна повторити процедуру з іншими початковими значеннями коефіцієнтів та проводити додаткові експерименти.

Метод Заде є простим у використанні і не вимагає математичних моделей системи. Він дозволяє швидко налаштувати коефіцієнти П-І-Д контролера, але його застосування може бути обмеженим в складних системах з нестійкими або нелінійними динаміками. У таких випадках можуть знадобитися більш продвинуті методи налаштування контролера.

3.2.4 Метод Cohen-Coon

Метод Cohen-Coon (також відомий як метод часу спрацювання або метод коефіцієнта чутливості) є одним з методів налаштування П-І-Д контролерів в системах автоматичного керування.

Цей метод був розроблений Жоржем Коеном і Гарольдом Куному 1953 році. Він базується на аналізі реакції системи на ступінь збурення та використовується для визначення оптимальних значень коефіцієнтів П-І-Д контролера.

Процедура налаштування методом Cohen-Coon включає наступні кроки:

1. Початковим кроком є встановлення контролера у режимі пропорційного регулювання зі значенням коефіцієнта пропорційності K_p, рівним нулю (K_p = 0).
2. Далі проводиться збурення системи шляхом введення ступінчатого сигналу або зміни вхідного сигналу.
3. Спостерігається вихідна відповідь системи та вимірюється час спрацювання процесу T_u - час, за який система досягає свого усталеного стану.
4. На основі виміряного часу спрацювання обчислюються оптимальні коефіцієнти для П-І-Д контролера:
    - Коефіцієнт пропорційності K_p: K_p = (1.35 / Ku) * (T_u / T_g),

- Коефіцієнт інтегрального впливу K_i: K_i = (2.5 / Ku) * (T_u / T_g),
- Коефіцієнт похідного впливу K_d: K_d = (0.37 / Ku) * (T_u / T_g), де Ku - апробаційний коефіцієнт (Ku) визначається як граничне значення коефіцієнта пропорційності, а T_g - час спрацювання процесу при коефіцієнті пропорційності K_p = Kp/4.

5. Контролер налаштований. Якщо результати не задовольняють вимоги, можна повторити процедуру з іншими початковими значеннями коефіцієнтів та проводити додаткові експерименти.

Метод Cohen-Coon є простим у використанні і не вимагає математичних моделей системи. Він дозволяє швидко налаштувати коефіцієнти П-І-Д контролера, але його застосування може бути обмеженим в складних системах з нестійкими або нелінійними динаміками. У таких випадках можуть знадобитися більш продвинуті методи налаштування контролера.

3.2.5 Метод налаштування за допомогою критерію МакКаула-Свідлера

Метод налаштування за допомогою критерію МакКаула-Свідлера (також відомий як метод мінімального часу спрацювання) є одним з методів налаштування П-І-Д контролерів в системах автоматичного керування. Цей метод був розроблений МакКаулом і Свідлером у 1950-х роках.

Цей метод базується на мінімізації часу спрацювання процесу досягнення усталеного стану після збурення системи. Основна ідея полягає в тому, що оптимальна настройка контролера забезпечує найшвидше досягнення усталеного стану після збурення.

Процедура налаштування за допомогою критерію МакКаула-Свідлера включає наступні кроки:

1. Початковим кроком є встановлення контролера у режимі пропорційного регулювання з коефіцієнтом пропорційності K_p, рівним нулю (K_p = 0).

2. Далі проводиться збурення системи шляхом введення ступінчатого сигналу або зміни вхідного сигналу.
3. Спостерігається вихідна відповідь системи та вимірюється час спрацювання процесу T_u - час, за який система досягає свого усталеного стану.
4. На основі вимірювань обчислюється оптимальне значення коефіцієнта пропорційності K_p за формулою:
    - Коефіцієнт пропорційності K_p: K_p = 0.5 * K_u, де K_u - апробаційний коефіцієнт, який визначається як граничне значення коефіцієнта пропорційності, при якому система починає виявляти нестійкість.
5. Коефіцієнти інтегрального та похідного впливу (K_i і K_d) можуть бути встановлені вручну або настроєні за допомогою інших методів налаштування, наприклад, методу Заде або методу оптимальної реакції на ступінь.
6. Контролер налаштований. Якщо результати не задовольняють вимоги, можна коригувати значення коефіцієнтів та проводити додаткові експерименти.

Метод налаштування за допомогою критерію МакКаула-Свідлера дозволяє отримати оптимальні коефіцієнти для П-І-Д контролера на основі мінімізації часу досягнення усталеного стану після збурення системи. Проте, він може бути чутливим до шумів і вимагати додаткової настройки в реальних умовах.

3.2.6 Метод налаштування за допомогою рівняння Журайда (Ziegler-Nichols)

Метод налаштування за допомогою рівняння Журайда (також відомий як метод Зіглера-Ніколсона або метод критичного демпфування) є одним з

методів налаштування П-І-Д контролерів в системах автоматичного керування.

Цей метод був розроблений Жоржем Журайдом на основі підходу, запропонованого раніше Джоном Зіглером і Натаніелем Ніколсоном. Він базується на аналізі реакції системи на ступінь збурення та використовується для визначення оптимальних значень коефіцієнтів П-І-Д контролера.

Процедура налаштування за допомогою рівняння Журайда включає наступні кроки:

1. Початковим кроком є встановлення контролера у режимі пропорційного регулювання з коефіцієнтом пропорційності K_p, рівним нулю (K_p = 0).
2. Далі проводиться збурення системи шляхом введення ступінчатого сигналу або зміни вхідного сигналу.
3. Спостерігається вихідна відповідь системи та вимірюється період коливань T_u - час, який пройшов між сусідніми максимумами або мінімумами відповіді системи.
4. На основі вимірювань обчислюються оптимальні коефіцієнти для П-І-Д контролера за рівнянням Журайда:
    - Коефіцієнт пропорційності K_p: K_p = 0.6 * Ku,
    - Коефіцієнт інтегрального впливу K_i: K_i = 1.2 * Ku / Tu,
    - Коефіцієнт похідного впливу K_d: K_d = 0.6 * Ku * Tu / 8, де Ku - апробаційний коефіцієнт, який визначається як граничне значення коефіцієнта пропорційності, при якому система починає виявляти нестійкість, а Tu - період коливань.
5. Контролер налаштований. Якщо результати не задовольняють вимоги, можна коригувати значення коефіцієнтів та проводити додаткові експерименти.

Метод налаштування за допомогою рівняння Журайда є простим у використанні і не вимагає складних математичних моделей системи. Він дозволяє швидко налаштувати коефіцієнти П-І-Д контролера, проте може бути

чутливим до шумів і не підходити для складних систем з нестійкими або нелінійними динаміками.

3.2.7 Метод налаштування за допомогою адаптивного керування

Метод налаштування за допомогою адаптивного керування є підходом до налаштування параметрів контролера, що базується на здатності системи автоматичного керування адаптуватися до змін у властивостях об'єкта керування.

В адаптивному керуванні контролер неперервно налаштовує свої параметри, щоб забезпечити оптимальне керування системою, враховуючи зміни у динаміці об'єкта керування або в зовнішніх умовах. Це дозволяє системі автоматичного керування ефективно пристосовуватися до різноманітних умов та забезпечувати високу якість регулювання.

Процедура налаштування за допомогою адаптивного керування включає наступні кроки:

1. Моделювання об'єкта керування: Спочатку необхідно побудувати математичну модель об'єкта керування або отримати експериментальні дані про його динаміку.
2. Вибір адаптивного алгоритму: Вибирається підхід до адаптивного керування, який найкраще відповідає характеристикам системи та поставленим завданням. Наприклад, можуть використовуватися алгоритми, такі як адаптивне керування зі зворотним поширенням (adaptive control with feedback) або модельно-передбачуване керування (model predictive control).
3. Формулювання критерію адаптації: Визначається критерій, що вимірює відхилення між вимогами до системи і її фактичною поведінкою. Цей критерій використовується для коригування параметрів контролера.

4. Ітерації адаптації: Застосовуються адаптивні алгоритми для неперервного оновлення параметрів контролера на основі отриманої інформації про динаміку об'єкта керування.
5. Тестування та налаштування: Система автоматичного керування піддається тестуванню на реальному об'єкті або на модельних симуляціях для перевірки її ефективності та налаштування параметрів з метою досягнення оптимального керування.

Метод налаштування за допомогою адаптивного керування дозволяє системі автоматичного керування самостійно адаптуватися до змін в об'єкті керування і навчатися оптимальному керуванню без необхідності використання заздалегідь заданих моделей. Це робить його особливо корисним у випадках, коли динаміка системи змінюється з часом або коли немає точних знань про модель системи.

3.2.8 Метод налаштування за допомогою оптимального керування

Метод налаштування за допомогою оптимального керування є підходом до налаштування параметрів контролера, що базується на математичних моделях системи та оптимальних критеріях керування.

У цьому методі використовується теорія оптимального керування, яка дозволяє знайти набір оптимальних параметрів контролера, які мінімізують певний критерій продуктивності або витрат. Цей критерій може варіюватися залежно від поставленої задачі, такої як мінімізація помилки регулювання, максимізація швидкості реакції, енергоефективність тощо.

Процедура налаштування за допомогою оптимального керування включає наступні кроки:

1. Моделювання системи: Побудова математичної моделі системи, що включає динаміку об'єкта керування та контролер.

2. Визначення оптимального критерію: Вибір критерію керування, який відображає бажану продуктивність системи і може бути математично виражений.
3. Розв'язання оптимальної задачі керування: Застосування методів оптимального керування, таких як принцип максимуму Понтрягіна або метод динамічного програмування, для знаходження набору оптимальних параметрів контролера.
4. Тестування та налаштування: Проведення експериментів на реальному об'єкті або симуляційних моделях з отриманими оптимальними параметрами контролера, перевірка ефективності та налаштування параметрів для досягнення заданих критеріїв продуктивності.

Метод налаштування за допомогою оптимального керування забезпечує високу продуктивність системи керування, оскільки враховує математичну модель системи та оптимальні критерії керування. Проте, він може вимагати складних обчислень та точних знань про модель системи, що може бути обмеженням у практичних застосуваннях.

3.2.9 Експериментальний метод

Експериментальний метод - це підхід до налаштування систем автоматичного керування, який базується на проведенні реальних експериментів на об'єкті керування з метою збору даних та визначення оптимальних параметрів контролера.

Процедура налаштування за допомогою експериментального методу включає наступні кроки:

1. Вибір початкових параметрів: Встановлення початкових значень параметрів контролера, які будуть використовуватися для експерименту. Ці значення можуть бути вибрані на основі попередніх досліджень або на основі експертної оцінки.

2. Проведення експерименту: Застосування контролера на об'єкті керування і збір даних про його реакцію на різні сигнали вхідного збурення або зміни параметрів. Це може включати введення ступінчатого сигналу, сигналу зміни амплітуди або частоти, або будь-якого іншого сигналу, який дозволяє вивчити властивості системи.
3. Аналіз результатів: Аналіз отриманих даних для визначення оптимальних параметрів контролера. Це може включати вимірювання часу спрацювання процесу, перехідні характеристики, сталість системи, помилку регулювання тощо. Цей аналіз може вимагати використання статистичних методів або інших аналітичних підходів для встановлення зв'язку між параметрами контролера та показниками продуктивності системи.
4. Модифікація параметрів контролера: На основі аналізу результатів експерименту здійснюється зміна параметрів контролера. Це може включати збільшення або зменшення значень коефіцієнтів пропорційності, інтегрального або похідного впливу, чи їх комбінацію.
5. Повторення експерименту та оптимізація: Процес повторюється з новими значеннями параметрів контролера, і експериментальний цикл триває до отримання оптимальних результатів згідно з поставленими критеріями продуктивності.

Експериментальний метод дозволяє отримати оптимальні параметри контролера на основі безпосереднього вивчення системи у реальних умовах. Він дозволяє враховувати нелінійність, нестійкість та інші особливості об'єкта керування, які можуть бути складно описати аналітично. Проте, вимагає витрат часу та ресурсів на проведення експериментів і аналіз даних.

3.3 Структура PID-регулятора

Структура PID-регулятора включає пропорційну, інтегральну та диференціальну складові, які взаємодіють між собою та з об'єктом керування для досягнення бажаної регуляції.

1. Пропорційна складова (P): Пропорційна складова реагує на поточну відхилення виміряної величини від заданого значення. Вона множиться на пропорційний коефіцієнт Kp і використовується для генерації керувального сигналу. Чим більше відхилення, тим сильніший керувальний сигнал вводиться для скоригування системи.
2. Інтегральна складова (I): Інтегральна складова враховує суму попередніх відхилень системи від заданого значення. Ця сума множиться на інтегральний коефіцієнт Ki і використовується для компенсації систематичних похибок, таких як постійне зміщення. Інтегральна складова дозволяє збирати "інтегральну помилку" і використовувати її для стабілізації системи.
3. Диференціальна складова (D): Диференціальна складова враховує швидкість зміни відхилення системи від заданого значення. Вона множиться на диференціальний коефіцієнт Kd і використовується для загасити швидкість зміни системи. Диференціальна складова допомагає уникнути різких змін і перекошень в системі.

Загальна структура PID-регулятора полягає у сумуванні цих трьох складових:

Керувальний сигнал = P + I + D

Пропорційна, інтегральна та диференціальна складові мають свої коефіцієнти (Kp, Ki, Kd), які можуть бути налаштовані для досягнення оптимальної регуляції системи. Кожна складова вносить свій внесок у керування, а їх взаємодія забезпечує точність та стабільність регуляції системи.

Взаємодія пропорційної, інтегральної та диференціальної складових з об'єктом керування може бути наступною:

- Пропорційна складова (P): Якщо відхилення системи від заданого значення збільшується, пропорційна складова генерує сильний керувальний сигнал для швидкого впливу на систему і зменшення відхилення. Пропорційна складова впливає безпосередньо на поточний стан системи.
- Інтегральна складова (I): Якщо система має постійне зміщення або систематичні похибки, інтегральна складова збирає цю помилку в часі і компенсує її. Чим довше система знаходиться у відхиленому стані, тим більше внесок робить інтегральна складова у зміну керувального сигналу. Це допомагає забезпечити точність регуляції та виправити систематичні похибки.
- Диференціальна складова (D): Диференціальна складова реагує на швидкість зміни відхилення системи. Якщо система швидко змінює свій стан, диференціальна складова внесе значний вплив на керування, щоб уникнути різких змін і перекошень. Вона допомагає забезпечити стабільність системи і уникнути перерегулювання.

Взаємодія цих складових залежить від вагових коефіцієнтів (Kp, Ki, Kd), які встановлюються в процесі налаштування PID-регулятора. Ці коефіцієнти визначають, як сильно кожна складова впливає на керування. Зрозуміло налаштовані вагові коефіцієнти дозволяють досягти бажаної регуляції системи з точністю та стабільністю.

3.4 Різновиди PID-регулятора

Різновиди під регуляторів, такі як пропорціонально-інтегруючий (П-І), пропорціонально-диференціюючий (П-Д), пропорціональний (П) та інші, відрізняються за наявністю та співвідношенням різних компонентів у їхній структурі. Вони використовуються для керування системами з різними динамічними властивостями та відповідями.

3.4.1 Пропорціональний (П) регулятор

Пропорціональний регулятор є одним з основних різновидів під регуляторів у системах автоматичного керування. Він дозволяє забезпечити регулювання системи шляхом пропорційного впливу на вихідний сигнал контролера в залежності від помилки регулювання.

Пропорціональний регулятор розрахований на основі поточної помилки регулювання, яка визначається різницею між поточним значенням вихідної величини системи і заданою цільовою величиною. Регулятор генерує вихідний сигнал, який є пропорційним до цієї помилки.

Математично, вихідний сигнал пропорціонального регулятора (u) обчислюється за формулою:

$$u = Kp * e, \quad (1.1)$$

де u - вихідний сигнал регулятора,

Kp - коефіцієнт пропорційності,

e - помилка регулювання.

Коефіцієнт пропорційності (Kp) визначає, наскільки сильно впливає помилка регулювання на вихідний сигнал регулятора. Чим більше значення Kp, тим сильніше вплив регулятора на систему.

Пропорціональний регулятор надає негативний зворотний зв'язок, що дозволяє системі автоматичного керування реагувати на помилки регулювання та змінювати свої вихідні сигнали для компенсації цих помилок.

Однак, пропорційний регулятор може мати статичну помилку, тобто постійну різницю між вихідним значенням системи та заданою цільовою величиною в усталених умовах. Ця статична помилка може бути компенсована додатковими компонентами, такими як інтегральний або диференціальний, у різновидах під регуляторів, які комбінуються з пропорційним регулятором.

3.4.2 Пропорціонально-інтегруючий (П-І) регулятор

Пропорціонально-інтегруючий (П-І) регулятор є різновидом під регулятора, який комбінує компоненти пропорційного та інтегрального регуляторів. Цей тип регулятора широко використовується в системах автоматичного керування для забезпечення точного регулювання та компенсації статичної помилки.

Пропорціональна складова П-І регулятора працює так само, як і пропорційний регулятор. Вона залежить від поточної помилки регулювання і генерує вихідний сигнал, пропорційний до цієї помилки.

Інтегральна складова П-І регулятора накопичує інтеграл від помилок регулювання з часом. Це означає, що чим триваліший період часу, протягом якого система має помилку регулювання, тим більше внесок робить інтегральна складова в вихідний сигнал регулятора. Інтегральна складова допомагає компенсувати статичну помилку, яка може бути присутня у пропорційного регулятора.

Математично, вихідний сигнал пропорціонально-інтегруючого регулятора (u) обчислюється за формулою:

$$u = Kp * e + Ki * \int e\,dt, \qquad (1.2)$$

де u - вихідний сигнал регулятора,

Kp - коефіцієнт пропорційності,

Ki - коефіцієнт інтегрального впливу,

e - помилка регулювання.

Коефіцієнти Kp та Ki визначаються в процесі налаштування регулятора і впливають на силу пропорційного та інтегрального впливів в системі.

Пропорціонально-інтегруючий регулятор дозволяє забезпечити високу точність регулювання та компенсувати статичну помилку. Однак, він може страждати від надмірної реакції на збурення і починати реагувати повільно на зміни в системі. Тому в деяких випадках використовуються більш складні різновиди під регуляторів, такі як пропорціонально-інтегруючо-диференціючий (П-І-Д) регулятор, щоб забезпечити кращу компенсацію динамічних збурень та швидку реакцію системи.

3.4.3 Пропорціонально-диференціюючий (П-Д) регулятор

Пропорціонально-диференціюючий (П-Д) регулятор є різновидом під регулятора, який комбінує компоненти пропорційного та диференціального регуляторів. Цей тип регулятора широко використовується в системах автоматичного керування для поліпшення стійкості, швидкості реакції та компенсації швидких змін в системі.

Пропорційна складова П-Д регулятора працює так само, як і пропорційний регулятор. Вона залежить від поточної помилки регулювання і генерує вихідний сигнал, пропорційний до цієї помилки.

Диференціальна складова П-Д регулятора враховує швидкість зміни помилки регулювання з часом. Це означає, що регулятор аналізує швидкість, з якою помилка регулювання змінюється, і генерує вихідний сигнал, пропорційний до цієї швидкості з використанням коефіцієнта диференціювання. Диференціальна складова допомагає підсилити реакцію регулятора на швидкі зміни і забезпечити більш стійку роботу системи.

Математично, вихідний сигнал пропорціонально-диференціюючого регулятора (u) обчислюється за формулою:

$$u = Kp * e + Kd * (de/dt), \quad (1.3)$$

де u - вихідний сигнал регулятора,

Kp - коефіцієнт пропорційності,

Kd - коефіцієнт диференціювання,

e - помилка регулювання,

de/dt - похідна помилки регулювання по часу.

Коефіцієнти Kp та Kd визначаються в процесі налаштування регулятора і впливають на силу пропорційного та диференціального впливів в системі.

Пропорціонально-диференціюючий регулятор дозволяє забезпечити швидку реакцію на швидкі зміни в системі, покращити стійкість та підсилити вплив регулятора. Однак, він може страждати від надмірної реакції на шуми або випадкові збурення, що може вплинути на стабільність системи. Тому в деяких випадках можуть бути використані більш складні комбінації під регуляторів, такі як пропорціонально-інтегруючо-диференціючий (П-І-Д) регулятор, для досягнення більш точного та стабільного керування.

3.4.4 Пропорціонально-інтегруючо-диференціючий (ПІД) регулятор

Пропорціонально-інтегруючо-диференціючий (П-І-Д) регулятор є комплексним під регулятором, який комбінує компоненти пропорційного, інтегрального та диференціального регуляторів. Цей тип регулятора широко використовується в системах автоматичного керування для досягнення точного та стійкого регулювання, компенсації статичної помилки, швидкої реакції та зменшення чутливості до збурень.

Пропорційна складова П-І-Д регулятора працює аналогічно до пропорційного регулятора. Вона залежить від поточної помилки регулювання і генерує вихідний сигнал, пропорційний до цієї помилки.

Інтегральна складова П-І-Д регулятора накопичує інтеграл від помилок регулювання з часом, так само, як у пропорціонально-інтегруючому (П-І) регуляторі. Інтегральна складова допомагає компенсувати статичну помилку та забезпечує точне регулювання системи в усталених умовах.

Диференціальна складова П-І-Д регулятора враховує швидкість зміни помилки регулювання, так само, як у пропорціонально-диференціюючому (П-Д) регуляторі. Вона аналізує швидкість, з якою помилка регулювання змінюється, і генерує вихідний сигнал, пропорційний до цієї швидкості. Диференціальна складова допомагає виявляти та реагувати на швидкі зміни в системі, покращує стійкість та зменшує перерегулювання.

Математично, вихідний сигнал пропорціонально-інтегруючо-диференціючого регулятора (u) обчислюється за формулою:

$$u = Kp * e + Ki * \int e\,dt + Kd * (de/dt), \quad (1.4)$$

де u - вихідний сигнал регулятора,

Kp - коефіцієнт пропорційності,

Ki - коефіцієнт інтегрального впливу,

Kd - коефіцієнт диференціювання,

e - помилка регулювання,

de/dt - похідна помилки регулювання по часу.

Коефіцієнти Kp, Ki та Kd визначаються в процесі налаштування регулятора і впливають на силу пропорційного, інтегрального та диференціального впливів в системі.

Пропорціонально-інтегруючо-диференціючий регулятор дозволяє досягнути високої точності регулювання, стійкості, швидкої реакції та зменшення впливу збурень на систему. Це один з найбільш складних та потужних типів під регуляторів, який застосовується у вимогливих задачах керування.

3.5 Використання PID-регулятора в системі автопілота

Використання PID-регулятора в системі автопілота цифрової моделі літака є важливою складовою для забезпечення точного та стабільного керування. При використанні PID-регулятора в системі автопілота виконуються різноманітні завдання, які допомагають забезпечити безпеку та

ефективність польоту. Основні завдання, які виконує автопілот за допомогою PID-регулятора, включають:

1. Підтримка певного рівня альтитуди: PID-регулятор дозволяє автопілоту підтримувати літак на певній висоті шляхом регулювання кута нахилу або потужності двигунів. Використовуючи вихідний сигнал PID-регулятора, автопілот може компенсувати будь-які зміни альтитуди та забезпечити стабільний політ на заданій висоті.
2. Стабілізація курсу та крену: PID-регулятор використовується для підтримки стабільного курсу та крену літака. Він реагує на будь-які зміни у курсі або крені та забезпечує відповідне коригування управління, щоб зберегти стабільну польотну конфігурацію.
3. Реакція на зміни вхідних сигналів: PID-регулятор дозволяє автопілоту ефективно реагувати на зміни вхідних сигналів, такі як швидкість вітру, перепади температури, масу або розподіл навантаження. За допомогою вихідного сигналу PID-регулятора, автопілот може компенсувати ці зміни і забезпечити стабільний політ та точність керування.

Використання PID-регулятора в системі автопілота дозволяє досягти точного та стабільного керування літаком. При правильній настройці та належному використанні PID-регулятор може забезпечити плавний політ, підтримку бажаної альтитуди, стабілізацію курсу та крену, а також ефективну реакцію на зміни у вхідних сигналах. Це сприяє покращенню безпеки та ефективності польоту, зменшенню навантаження на пілота та забезпеченню більш точного виконання завдань.

3.6 Переваги та обмеження використання PID-регулятора

PID-регулятор має декілька переваг, які роблять його ефективним для багатьох систем керування. Основні переваги використання PID-регулятора включають:

1. Простота і простота реалізації: PID-регулятор має просту структуру та легко реалізується в програмному або апаратному забезпеченні. Він не вимагає складних обчислень або високої обчислювальної потужності.
2. Гнучкість і універсальність: PID-регулятор може бути застосований до різноманітних систем керування, незалежно від їх складності або типу. Він підходить для широкого спектру застосувань, від простих систем температурного керування до складних систем автопілоту.
3. Добра стабільність і швидкість відгуку: PID-регулятор забезпечує стабільність системи керування та швидку відповідь на зміну вхідних сигналів або завдань. Він може ефективно управляти системою, щоб досягти бажаної відповіді.
4. Просте налаштування: PID-регулятор має параметри, які можуть бути налаштовані для досягнення оптимальної продуктивності системи керування. Існує досвід та методи налаштування, які дозволяють досягти заданих цілей керування.

Незважаючи на переваги, використання PID-регулятора має деякі обмеження, особливо у складних системах або в разі наявності нестачі точної математичної моделі об'єкта керування. Деякі з обмежень використання PID-регулятора включають:

1. Чутливість до збурень і нестійкість: PID-регулятор може бути чутливим до збурень або шумів в системі, що може призводити до нестабільності. Це особливо відчутно у випадку наявності сильних змін в системі або нерівномірних умов.
2. Обмежена адаптивність: PID-регулятор не є повністю адаптивним до зміни параметрів системи або зовнішніх умов. Він вимагає регулярного налаштування та пристосування для досягнення оптимальної продуктивності.
3. Відсутність врахування нестатичних характеристик: PID-регулятор не враховує нестатичних характеристик об'єкта керування, таких як час

запізнення, змінний коефіцієнт підсилення тощо. Це може призводити до недостатньої продуктивності в деяких системах.

Незважаючи на обмеження, PID-регулятор залишається популярним і широко використовуваним методом керування в багатьох системах, завдяки своїм простоті, ефективності та широкому спектру застосувань.

3.7 Висновки до розділу

На основі досліджень, проведених у цьому розділі, можна сказати, що PID-регулятори є важливим елементом в системах зворотного зв'язку, забезпечуючи стабільність, точність та швидкодію в автоматичних керувальних системах, таких як автопілот.

Принцип роботи PID-регулятора, який включає в себе пропорційну, інтегральну та диференціальну складові, відображає його гнучкість та універсальність, дозволяючи йому адаптуватися до різних об'єктів керування і сценаріїв.

Можливості налаштування, які надають методи, розглянуті в розділі, вказують на можливість вибору оптимального балансу між стабільністю та точністю, а також між швидкодією та відхиленням від заданого значення.

Поширене використання PID-регулятора в системах автопілота підтверджує його ефективність та надійність. Однак, потрібно враховувати, що кожен метод налаштування має свої особливості, а PID-регулятори можуть виявитися недостатніми у складних системах або при відсутності точної математичної моделі.

РОЗДІЛ 4. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Враховуючи попередні дослідження та вибраний спосіб реалізації, можна переходити до стадії розробки системи автопілотування. Для створення надійної, адаптивної та точної системи, що використовує PID регулятори для автопілотування літаків, нам потрібно детально описати основні компоненти розробленого продукту, функціональні блоки системи та схему їх взаємодії.

В цьому розділі буде наданий деталізований огляд створеної системи, включаючи архітектуру програми, методику реалізації ключових компонентів та їх взаємодії, а також розглядаємо особливості використання мови програмування Lua і двигуна X-Plane в контексті проекту.

Особлива увага буде приділена забезпеченню точності моделювання польотів і відповідності цифрового двійника літака до його реального аналогу, а також гарантуванню повторюваності та надійності тестового процесу.

4.1 DataRefs

DataRef'и є важливою складовою системи X-Plane і відіграють ключову роль у взаємодії з внутрішніми параметрами літака та середовищем симуляції. Вони є механізмом доступу до даних, які використовуються для контролю, візуалізації та моделювання різних аспектів польоту.

DataRef'и в X-Plane представляють собою посилання на конкретні дані в системі. Ці дані можуть бути параметрами літака (такими як швидкість, кут нахилу, положення поверхонь керування тощо), станом середовища (атмосферні умови, погода, час) або внутрішніми змінними X-Plane (наприклад, позиція літака, стан системи управління).

DataRef'и можуть бути використані для зчитування та запису даних. Це дозволяє програмним продуктам та плагінам взаємодіяти з різними аспектами симуляції. Наприклад, плагіни можуть зчитувати дані про положення літака та

виконувати обчислення для розрахунку відповідних керуючих сигналів для системи автопілотування.

DataRef'и також використовуються для зміни стану симуляції, що дозволяє плагінам змінювати параметри літака, середовища чи системи управління в реальному часі. Наприклад, плагін може змінити швидкість або положення літака, налаштувати погодні умови або активувати/вимкнути певні системи.

DataRef'и використовуються для отримання та зміни даних у симуляторі X-Plane за допомогою мови програмування Lua та бібліотеки SASL 3. SASL (Scriptable Aviation Simulation Library) - це набір інструментів для створення плагінів та додатків для X-Plane з використанням мови Lua.

Для створення DataRef'ів та роботи з ними використовується функція createGlobalPropertyf у бібліотеці SASL 3. Ця функція дозволяє створити глобальну властивість змінної типу float (число з плаваючою комою), яка може бути використана як DataRef.

Наприклад, для створення DataRef'у з назвою "myDataRef" та початковим значенням 0.0, можна використати наступний код у Lua з використанням SASL 3:

```
myDataRef = createGlobalPropertyf("myDataRef", 0.0)
```

Отримання значення DataRef'у здійснюється шляхом звернення до змінної, створеної за допомогою createGlobalPropertyf. Наприклад, можна отримати значення DataRef'у "myDataRef" та вивести його на консоль:

```
print(get(myDataRef))
```

Зміна значення DataRef'у також здійснюється шляхом присвоєння нового значення змінній DataRef. Наприклад, для зміни значення DataRef'у "myDataRef" на 1.5, можна використати такий код:

```
set(myDataRef, 1.5)
```

Таким чином, використовуючи createGlobalPropertyf у Lua з SASL 3, можна створювати DataRef'и та працювати з ними для отримання та зміни даних у симуляторі X-Plane. Це дозволяє програмістам створювати

різноманітні плагіни та додатки, які взаємодіють з внутрішніми параметрами літака та середовищем симуляції.

4.2 Розробка програмного коду

4.2.1 Скрипт main.lua

Головним скриптом проекту є main.lua головним скриптом проекту і відповідає за ініціалізацію та управління всіма компонентами системи автопілотування. Він слугує як точка входу в систему і виконує ряд ключових функцій.

1. Установка опцій SASL: Перші три рядки скрипта встановлюють опції SASL, що дозволяють використовувати віртуальну панель літака, інтерактивність та 3D рендеринг.
2. Ініціалізація глобальних налаштувань: Скрипт встановлює глобальні налаштування проекту, такі як режим клієнта, використання віртуальних панелей та налаштування камери.
3. Включення необхідних модулів: У випадку, якщо система працює не в режимі клієнта, скрипт включає допоміжні модулі, які необхідні для роботи системи, включаючи модулі для обробки конфігурацій, математичних операцій, маніпуляцій з файлами, управління камерою та інші.
4. Стартова функція: Функція start викликається для ініціалізації основних компонентів системи, таких як читання конфігурацій та створення словника пристроїв.
5. Перевизначення деяких datarefs: Скрипт перевизначає декілька datarefs для контролю поверхонь та напряму керма.
6. Ініціалізація власних datarefs і команд: Якщо система працює не в режимі клієнта, скрипт включає модулі власних datarefs і команд.

7. Встановлення шляхів до скриптів: Скрипт встановлює шляхи до різних скриптів, які необхідні для роботи системи.
8. Ініціалізація компонентів: Скрипт ініціалізує різні системи літака, які він буде контролювати.
9. Обробники оновлень: Скрипт містить декілька обробників оновлень, які виконуються на різних стадіях роботи системи, включаючи ініціалізацію параметрів, встановлення камери інструктора, виконання команд посадки і оновлення налаштувань.
10. Функція відображення: Функція draw відповідає за відображення всіх компонентів.
11. Обробники завантаження: Скрипт містить ряд обробників, які виконуються при завантаженні аеропорту, сцени та літака.
12. Обробник завершення модуля: На кінець, скрипт містить обробник onModuleDone, який виконується при завершенні роботи модуля і відповідає за зупинку всіх команд та відновлення перевизначених datarefs.

Отже, main.lua відповідає за координацію всіх компонентів системи і виконує центральну роль в контролі автопілотування літака.

4.2.2 Скрипт custom_datarefs.lua

Цей файл містить набір вказівок для створення глобальних властивостей або "datarefs" у рамках програмного забезпечення, що симулює реальність. Він використовується для моделювання поведінки та відслідковування стану різних систем та елементів поведінки літака.

Ось основні розділи, які можна виділити у цьому файлі:

1. Настройки: Тут задаються, деякі загальні настройки проекту, які визначаються динамічно на основі об'єкта project_settings.

2. Команди: Тут створюються глобальні властивості, пов'язані з керуванням поверхнями літака (наприклад, елеронами, елеваторами, рулем), станом шасі, розподілом ваги та іншими командами.
3. Панелі управління: Тут створюються datarefs для різних панелей управління в кабіні пілота.
4. Системи поверхонь контролю: Тут створюються datarefs, що контролюють максимальний кут і значення контролера для різних систем управління поверхнями, таких як крила, руддери, стабілізатори та ін.
5. Шасі: Тут створюються datarefs для контролю шасі, включаючи висоту, поворот, гальма та ін.

4.2.3 Скрипт pid.lua

pid.lua є модулем, який використовує контролер пропорційно-інтегрально-диференційного (PID) типу для виконання управління. PID-контролери широко використовуються в системах управління для забезпечення згасання похибки між поточним і цільовим станами системи.

Давайте детально розглянемо, що робить кожна функція у цьому модулі:

1. PID.New(Kp, Ki, Kd): Ця функція створює новий екземпляр PID-контролера з заданими коефіцієнтами Kp, Ki, Kd (пропорційний, інтегральний та диференційний відповідно)
2. PID:GetSetting(targetValue, currentValue, currentSetting, dt, k): Ця функція використовує PID-алгоритм для розрахунку налаштувань, використовуючи цільове значення, поточне значення, поточне налаштування, інтервал часу (dt), та додатковий множник (k).
3. PID:Reset(): Ця функція скидує попередні помилки (prevE та prevPrevE) до 0.
4. PID:Clone(): Ця функція створює новий екземпляр PID-контролера з тими ж параметрами, що й в оригінального контролера.

5. PID.Resets(...): Ця функція приймає список PID-контролерів і скидує їх, викликаючи для кожного з них метод Reset().
6. PID.__tostring(self): Ця функція визначає, як об'єкт PID буде відображатися при виведенні в якості рядка.
7. fields.default(): Ця функція повертає новий PID-контролер із замовчуваними коефіцієнтами.

У цілому, цей модуль надає клас PID для створення та управління PID-контролерами, включаючи методи для налаштування, скидання та клонування контролерів, а також для отримання та встановлення їхніх значень.

4.2.4 Скрипт global_constraints.lua

Файл global_constraints.lua містить константи та глобальні змінні, які використовуються в проекті, який має відношення до авіаційного симулятора. Давайте розглянемо його зміст:

1. sim_speed і framerate_period: Ці дві змінні використовуються для зберігання поточної швидкості симуляції та періоду частоти кадрів. Ці дані витягуються з глобальних властивостей симулятора.
2. EPSILON: Ця константа використовується як мале число, що може бути використане для уникнення проблем з числами з плаваючою точкою або для зрівняння чисел з певною малою похибкою.
3. BASE_DREF_NAME і інші _G-змінні: Ці змінні містять рядки, що використовуються для зберігання імен шляхів до різних систем в симуляторі. Наприклад, PANEL зберігає шлях до системи панелі, а PHYSICAL - шлях до фізичних налаштувань симулятора.
4. PC_NAME, X-PLANE_PATH, AIRCRAFT_PATH, CONFIGS_PATH і MODULES_PATH: Ці змінні використовуються для зберігання важливих шляхів, пов'язаних із проектом і симулятором. Ці значення отримуються за допомогою методів SASL, які отримують відповідні шляхи.

У цілому, global_constraints.lua використовується для зберігання ряду глобальних констант і змінних, що використовуються в проекті. Це допомагає забезпечити централізоване місце для управління цими значеннями, що спрощує їхнє використання та зміну.

4.2.5 Скрипт global_constraints.lua

Файл custom_commands.lua містить набір команд, які використовуються для контролю різних систем і компонентів літака. Цей файл використовується для створення власних команд у симуляторі літака. Команди можуть бути прив'язані до конкретних кнопок або джойстиків, що дозволяє більш точне і гнучке управління моделлю літака в симуляторі.

Всі ці команди використовують функцію createCommand, яка приймає два аргументи: рядок, що вказує на команду, і опис команди.

4.2.6 Скрипт UnitTestManager.lua

Файл UnitTestManager.lua служить для управління юніт-тестами у проекті. Ось основні елементи його функціоналу:

1. Визначення властивостей та глобальних змінних: В цьому файлі встановлюються ряд властивостей і глобальних змінних, що використовуються для управління статусами тестів та їх результатами.
2. Команди: Файл містить набір команд, які використовуються для запуску різних юніт-тестів. Кожен з цих тестів має свій обробник, який визначає, як відпрацьовує конкретна команда.
3. Статуси: Визначені статуси для тестів, включаючи "None", "Start", "Test" і "Stop".
4. Реєстрація обробників команд: Функція registerCommandHandlers використовується для реєстрації обробників команд юніт-тестів та команди для закінчення юніт-тесту.

5. Функція update: Ця функція виконується кожного кадру і використовується для перевірки стану юніт-тесту та виконання різних дій в залежності від цього стану.
6. Масив components: Масив components містить об'єкти, кожен з яких представляє окремий юніт-тест.
7. Масив _testCommands: Цей масив включає об'єкти, які представляють команди для запуску юніт-тестів. Кожен об'єкт містить команду для запуску тесту та відповідний обробник команди.

Цей файл представляє менеджер юніт-тестів, що дозволяє автоматично запускати і контролювати юніт-тести для різних компонентів системи. Команди SASL, що використовуються для керування юніт-тестами, можуть бути прив'язані до кнопок або джойстиків для легкого доступу.

4.2.7 Скрипт testStatsPrinter.lua

Файл testStatsPrinter.lua служить для збереження результатів юніт-тестів у текстовому форматі. Ось його основні функції:

1. Імпортування необхідних модулів: Цей файл використовує модуль fileUtils з папки utils для роботи з файлами.
2. Визначення шляху до директорії з результатами тестів: Змінна DIRECTORY_PATH визначає шлях до папки, в якій зберігатимуться результати тестів.
3. Функція save: Ця функція приймає ім'я тесту та результат тесту як аргументи. За допомогою цієї функції результати тесту записуються у файл у визначеному раніше каталозі з додаванням дати і часу виконання тесту.

   Ось, як працює save:

   - Спочатку визначається шлях до файлу, в якому будуть зберігатися результати тесту. Цей шлях генерується за допомогою імені тесту.

- Потім генерується контент, який буде записаний в файл. Контент складається з імені тесту, поточного часу і дати, результату тесту та роздільників для кращої читабельності.
- Якщо файл не існує, він створюється.
- Далі весь існуючий контент файлу зчитується і зберігається у таблицю lines.
- Новий контент додається на початок таблиці lines.
- Всі зміни записуються в тимчасовий файл.
- Старий файл видаляється.
- Тимчасовий файл перейменовується в оригінальний файл.

Ця функція дозволяє зберігати результати юніт-тестів у текстовому форматі, зберігаючи історію всіх попередніх результатів.

4.2.8 Скрипт EngineSystems.lua

Скрипт EngineSystems.lua визначає три компоненти системи двигунів у проекті: EnginesCS, EngineSystem, та EngineFailures.

1. EnginesCS: Цей компонент визначає загальні властивості для системи двигунів. Він включає параметри, такі як вібрація різних двигунів (engine1, engine2, engine3, engine4), мінімальний тиск олії (minDavlMaslaIndicator) та інші.
2. EngineSystem: Цей компонент визначає специфічні властивості для окремого двигуна. Він включає параметри, такі як напруга, обертальний момент, стан двигуна, споживання палива, вібрації, індикатори небезпечних ситуацій та багато інших. Цей компонент повторюється для кожного двигуна.
3. EngineFailures: Цей компонент відстежує стани збоїв для всіх двигунів. Він включає вхідні та вихідні властивості, які пов'язані з різними типами збоїв, такими як збій стартера, збій паливної системи, збій насосу, збій системи охолодження двигуна, небезпечні вібрації тощо.

Загалом, цей скрипт забезпечує важливі деталі про функціонування двигунів у проекті, включаючи їхній стан, параметри, і можливі збої.

4.2.9 Скрипт EngineSystem.lua

EngineSystem.lua, є скриптом, що моделює поведінку системи двигуна для X-Plane. Він використовує datarefs та команди X-Plane для контролю двигуна.

Основні функції та змінні:

- Datarefs: це глобальні змінні X-Plane, які використовуються для передачі даних між різними частинами симулятора. В даному скрипті вони використовуються для передачі даних про стан двигуна, наприклад стан приводу двигуна, реверсу, ступінь обертання (N1, N2) тощо.
- EngineState: це стани, в яких може знаходитися двигун - OFF, TurningON, ON, TurningOff.
- AutoReverseStage: це стани для автоматичного реверсу.
- readParameters(): ця функція зчитує параметри системи двигуна.
- start(): ця функція ініціалізує запуск двигуна, знаходячи потрібні команди за допомогою номера двигуна.
- readInputs(): ця функція зчитує вхідні сигнали.
- TryStartEngines(), TryStartEnginesInAir(): ці функції займаються логікою запуску двигунів на землі, в повітрі та фейкового запуску двигунів.
- SetThrottle(): ця функція контролює відкриття дросельної заслінки.
- Reverse(), AutoReverse(): ці функції контролюють реверс двигуна.
- GetVibrations(): ця функція оцінює вібрації двигуна.
- StopEngine(): ця функція зупиняє двигун.
- MapGasTemp(), MapN1(), MapN2(): ці функції мапують температуру газу та ступені обертання двигуна (N1, N2).

- SetActualPanelValues(), SetEnginePanelValues(): ці функції встановлюють значення на панелі керування.
- GetEngineStatus(): ця функція отримує поточний статус двигуна.
- setOutputs(), writeOutputs(): ці функції встановлюють та записують вихідні значення.
- settingThrottle_Handler(), stopEngine_Handler(), quickStart_Handler(), reverseOn_Handler(), reverseOff_Handler(), StruzhkaVMasle_Handler(): це обробники подій для різних команд.

Загалом, скрипт моделює поведінку двигуна, використовуючи реалістичні параметри та вхідні команди, та дозволяє змінювати стан двигуна засобами коду.

4.2.10 Скрипт SurfaceControls.lua

Скрипт SurfaceControls.lua використовується для моделювання контролю поверхонь повітряного судна, включаючи елерони (ailerons), спойлери (spoilers), елеватори (elevators), руль напряму (rudder), стабілізатор (stab), закрилки (flaps) і передкрилки (slats).

Ось детальний опис кожного компонента:

- ailerons: відповідають за контроль елеронів. Вони визначають максимальний та мінімальний кути відхилення, максимальну швидкість переміщення елеронів (VPS - Velocity per Second), а також вхідні дані з контролерів елеронів (_iRoll).
- spoilers: контролюють спойлери. Вони мають подібні параметри до елеронів, але додатково включають контроль гідравлічного споживання.
- elevators: відповідають за контроль елеваторів. Подібно до елеронів, вони визначають максимальні та мінімальні кути, максимальну швидкість переміщення, а також вхідні дані з контролерів елеваторів.
- rudder: контролюють руль напряму. Вони мають параметри, подібні до інших контролерів, з вхідними даними з контролерів руля (_iYaw).

- stab: моделює стабілізатор, включаючи максимальний та мінімальний кути, максимальну швидкість переміщення, та гідравлічне споживання.
- flaps: контролюють закрилки. Вони включають параметри, подібні до інших контролерів, але також контролюють гідравлічне споживання.
- slats: контролюють передкрилки. Вони мають параметри, подібні до інших контролерів, але також контролюють гідравлічне споживання.

У загальному вигляді, цей скрипт моделює фізичні характеристики контрольних поверхонь повітряного судна, включаючи їх діапазон руху, швидкість переміщення та вплив на динаміку політного апарату. Крім того, він також враховує різні обмеження, такі як гідравлічне споживання. Всі ці функції дозволяють досить точно відтворювати реальну поведінку повітряного судна в різних умовах польоту.

4.3 Висновки до розділу

Основу розробки системи автопілотування літаків становить компетентне використання мови програмування Lua та механізму DataRefs у середовищі X-Plane. Спрямовуючи наші зусилля на створення ефективних скриптів, ми встановлюємо робочий процес, який покликаний забезпечити точність, надійність та повторюваність при тестуванні системи.

Впровадження DataRefs у систему автопілотування відкриває можливості для надзвичайно гнучкого контролю та візуалізації даних польоту. Цей механізм стає "мостом", який з'єднує програми та плагіни з реальними параметрами літака та станом середовища в режимі реального часу.

Створені нами скрипти на мові Lua відіграють вирішальну роль в реалізації системи автопілотування, дозволяючи здійснювати необхідні операції. Вони являють собою важливі інструменти, кожен з яких виконує свою специфічну роль в системі.

Підсумовуючи, ефективність розробки системи автопілотування вимагає глибокого розуміння та вправного застосування інструментарію, який ми обрали - мови програмування Lua та механізму DataRefs у X-Plane. Це створює надійну основу для подальшого вдосконалення та розвитку системи.

РОЗДІЛ 5. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

В рамках даного розділу будуть представлені результати дослідження та аналізу розробленої системи автопілотування літаків, побудованої на основі PID-регуляторів з використанням мови програмування Lua та DataRefs у середовищі X-Plane.

Метою цього розділу є дослідження та оцінка роботи системи автопілотування в різних умовах та сценаріях польоту. Зокрема, будуть проаналізовані результати польотів, проведених з використанням розробленої системи, і порівняно з результатами реальних польотів літака. Це дозволить оцінити точність, стабільність та надійність системи автопілотування.

5.1 Тест зліту

Тримаючи літак на гальмах, перевести плавно важелі керування зовнішніх, а потім внутрішніх двигунів на злітний режим;

Плавно відпустити гальма та , включити секундомір (в момент початку руху полоси), почати розбіг.

Розбіг виконувати з відхиленням від себе штурвалом. Напрямок витримувати відповідним відхиленням педалі (управління поворотом коліс носової ноги шасі повинно бути включено).

При наближенні до швидкості відриву носового колеса відхиленням штурвалу на себе виконати відрив коліс носової ноги шасі. Кут тангажу при відриві літака не має перевищувати 8°.

Під час відриву літака , відміченого по варіометру, виключити секундомір та зафіксувати значення швидкості відриву літака та час розбігу.

Таблиця 5.1 – Результати тесту зліту

Параметри	Очікуваний результат	Отриманий результат
Швидкість відриву носового колеса	245±25 км/год	231 км/год
Швидкість відриву літака	260±25 км/год	239 км/год
Час розбігу	28±3 с	26.58 с
Кут тангажу при відриві	< 8º	5.69º

5.2 Тест гальмування під час зльоту

Для завдання початкових умов та виконання злету використовувати методику, вказану в підпункті 5.1.

Під час розбігу на швидкості 220 км/год включити відмову подачі палива 1-го двигуна при 1-й перевірці та відмові подачі палива 1-го та 2-го двигунів. Утримуючи літак від розвороту, за допомогою педалей провести гальмування, для чого:

- Включити секундомір, перевести РУД усіх двигунів на режим "Малий газ".
- Вивести на монітор інструктора кути відхилення руля напрямку.
- Включити реверс тяги 2 та 3 двигунів.
- Випустити спойлери та гальмівні щитки.
- Натиснути гальмівні підніжки.
- При досягненні швидкості 60 км/год прибрати механізацію крила та при 50 км/год реверс тяги вимкнути.
- Коли швидкість впаде до нуля, зафіксувати покази секундоміра.

Таблиця 5.2 – Результати тесту гальмування під час зльоту

Параметри	Очікуваний результат	Отриманий результат
Час гальмування	15±5 с	19.76 с
Кут відхилення кута напрямку	< 10º	6.78º

5.3 Тест зліту з навантаженням

Для завдання початкових умов та виконання злету використовувати методику, вказану в підпункті 5.1.

Ввести комерційне навантаження 12100 кг з тим, щоб повна вага G дорівнювала максимальній злітній вазі 170000. Встановити φ=-4,8º. Під час розбігу на швидкості 250 км/год ввімкнути відмову подачі палива 1-го двигуна та продовжити розбіг, витримуючи за допомогою педалей напрямок по осі ЗПС. На швидкості 260 км/год вивести на монітор АРМІ значення кута відхилення руля направлення. Під час відриву літака виключити секундомір. Після відриву прибрати шасі, встановити вертикальну швидкість набора висоти, витримуючи постійною та рівною 290 км/год.

Таблиця 5.3 – Результати тесту зліту з навантаженням

Параметри	Очікуваний результат	Отриманий результат
Час розбігу	40±4 с	37 с
Кут відхилення кута напрямку	15±8º	14.149º

5.4 Тест крейсерської швидкості горизонтального польоту

1. Запустити робочу програму
2. Запустити двигуни
3. Провести зліт по методиці пункту 5.1
4. Встановити параметри:
    - Працюють 4 двигуна
    - G=145000 кг
    - Шасі прибрані
    - φ - балансувальні
5. Задати висоту H = 2650
6. Відключити програму розходу палива

7. Виконати горизонтальний політ та після досягнення при 530 км/год швидкості витримувати горизонтальний політ протягом двох хвилин. Зафіксувати значення обертів.

Таблиця 5.4 – Результати тесту крейсерської швидкості горизонтального польоту

Параметри	Очікуваний результат	Отриманий результат
Значення обертів	85±2%	86.61%

5.5 Тест розгону та гальмування горизонтального польоту

1. Запустити робочу програму
2. Запустити двигуни
3. Виконати зліт по пункту 5.1
4. Ввести параметри:
   - Висоту H = 600 м,
   - Вагу палива = 33000кг,
   - Вагу комерційного навантаження = 7100кг,
5. Під час перевірки часу розгону витримати горизонтальний політ зі швидкістю 350 км/год
6. Встановити за допомогою РУД оберти в стан "Злітний". Виміряти час розгону від до 500 км/год
7. Під час перевірки часу гальмування витримати горизонтальний політ зі швидкістю 390 км/год
8. Встановити за допомогою РУД оберти на 93%. Виміряти час гальмування від 500 до 370 км/год.

Таблиця 5.5 – Результати тесту розгону та гальмування горизонтального польоту

Параметри	Очікуваний результат	Отриманий результат
Час гальмування	33±7 с	38 с

5.6 Тест на максимальну швидкість горизонтального польоту

1. Запустити робочу програму
2. Запустити двигуни
3. Виконати зліт по пункту 5.1
4. Ввести параметри:
    - Висоту H = 2650 м,
    - Вагу палива = 35000 кг,
    - Вагу комерційного навантаження = 17100 кг,
5. За допомогою РУД встановити номінальний режим роботи двигунів та після досягнення встановленої швидкості витримувати горизонтальний політ протягом двох хвилин

Таблиця 5.6 – Результати тесту на максимальну швидкість горизонтального польоту

Параметри	Очікуваний результат	Отриманий результат
Максимальна швидкість	835±80 км/год	798.68 км/год

5.7 Тест швидкості підйому

1. Запустити робочу програму
2. Запустити двигуни
3. Виконати зліт по пункту 5.1
4. Ввести параметри:
    - Вагу палива = 42000 кг,
    - Вагу комерційного навантаження = 35100 кг,
5. Провести розгін біля землі на висоті 20-25м до заданої приборної швидкості та, включивши секундомір, перейти в набір висоти, витримуючи задану приладну швидкість. На значеннях висоти H = 2000 м, зафіксувати показник секундоміра.

Таблиця 5.7 – Результати тесту швидкості підйому

Параметри	Очікуваний результат	Отриманий результат
Час підйому	2хв 43с ± 20с	3хв 2с

5.8 Висновки до розділу

Проведені дослідження та аналіз підтвердили здатність розробленої цифрової моделі до ефективного моделювання фізичних характеристик літака. Низка проведених тестів не лише дозволила перевірити дієздатність і надійність створеної цифрової моделі, але і підтвердила її ефективність і точність відтворення реальних характеристик літака.

Важливим моментом є те, що очікувані результати кожного тесту були базовані на реальних даних, що забезпечує об'єктивність і достовірність отриманих результатів. Це підкреслює наше визнання важливості використання реальних даних для перевірки та адаптації наших цифрових моделей.

Цей проект виявився успішним у створенні цифрової моделі, яка точно відтворює фізичні характеристики літака. Це показує, що такий підхід може бути корисним для подальших досліджень в цій області. Успішність цієї роботи також підтверджує, що цифрові моделі можуть бути використані для точного моделювання реальних умов і параметрів, відкриваючи нові можливості для їх використання в майбутньому.

ВИСНОВКИ

У ході виконання цієї дипломної роботи було підтверджено можливість використання цифрових моделей для моделювання реальних фізичних процесів, що відбуваються під час польоту літака. Ця робота демонструє, як комплексне використання технологій та інструментів, таких як мова програмування Lua, двигун X-Plane та PID-регулятори, може привести до розробки ефективної та надійної системи автопілотування.

Було показано, як аналіз предметної області та визначення вимог до характеристик проектування можуть підсилити розуміння задачі та її рішення. Вивчення PID-регуляторів дало змогу зрозуміти, як можна використовувати ці інструменти для управління динамікою польоту.

Процес розробки системи, що включав роботу з DataRefs та написання скриптів, дозволив створити цифрову модель, яка точно відтворює реальні характеристики літака. Це підтверджено ретельними тестами та аналізом, що вказують на високу дієздатність та надійність розробленої системи.

Цей проект ілюструє значення та потенціал використання цифрових моделей в автоматичних системах керування, зокрема в авіації. Подальший розвиток і впровадження подібних моделей можуть мати практичну цінність, допомагаючи поліпшити безпеку польотів, роботу пілотів та ефективність управління літаками.

Завдання, поставлене в дипломній роботі, виконано повністю із дотриманням всіх вимог, які висуваються до дипломних робіт кафедри ОТ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. uavAP: A Modular Autopilot Framework for UAVs [Електронний ресурс] / Mirco Theile, D. Dantsker, Richard Nai, Marco Caccamo. – 2020. – Режим доступу до ресурсу: http://rtsl-edge.cs.illinois.edu/UAV/inc/uavAP_AIAA-Aviation-2020.pdf.
2. Autopilot [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Autopilot.
3. Lua [Електронний ресурс] – Режим доступу до ресурсу: http://www.lua.org/about.html
4. X-Plane [Електронний ресурс]: Доступ: https://flight.fandom.com/wiki/X-Plane_11
5. X-Plane (docs) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.x-plane.com/docs/.
6. Пропорційно-інтегрально-диференціальний закон регулювання [Електронний ресурс] – Режим доступу до ресурсу: https://org2.knuba.edu.ua/pluginfile.php/78728/mod_resource/content/1/Tema2.pdf
7. Theile, M., Dantsker, O. D., Nai, R., and Caccamo, M., “uavEE: A modular, power-aware emulation environment for rapid prototyping and testing of uavs,” 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), IEEE, 2018, с. 217–224.
8. Yu, S., Flight “Maneuver Automation for System Analysis of Small Fixed-Wing UAVs”, Бакалаврська робота, University of Illinois at UrbanaChampaign, Department of Electrical and Computer Engineering, Urbana, IL, 2019.

9. Dantsker, O. D., Yu, S., Vahora, M., and Caccamo, M., “Flight Testing Automation to Parameterize Unmanned Aircraft Dynamics,”.
10. Lugo-Cardenas, I., Flores, G., Salazar, S., and Lozano, R., “Dubins path generation for a fixed wing UAV,” ´ 2014 International Conference on Unmanned Aircraft Systems (ICUAS), Травень 2014, с. 339–346.
11. Dantsker, O. D., Ananda, G. K., and Selig, M. S., “GA-USTAR Phase 1: Development and Flight Testing of the Baseline Upset and Stall Research Aircraft,” AIAA Paper 2017-4078, AIAA Applied Aerodynamics Conference, Denver, Colorado, Червень 2017.
12. Петунін В.І. Синтез законів управління каналом тангажа автопілота / Петунін В.І. // Вісник УГАТУ. Управління, ВТ та І. Системний аналіз, управління та обробка інформації. – УГАТУ, 2007. - №2(20) – С.25-31
13. Репнікова Н.Б. Теорія автоматичного керування: класика і сучасність: поруч. / Н.Б. Репнікова. – К. : НТУУ «КПІ», 2011. – 328 с.

ДОДАТОК 1

Система автопілотування літаків побудована на pid регуляторах

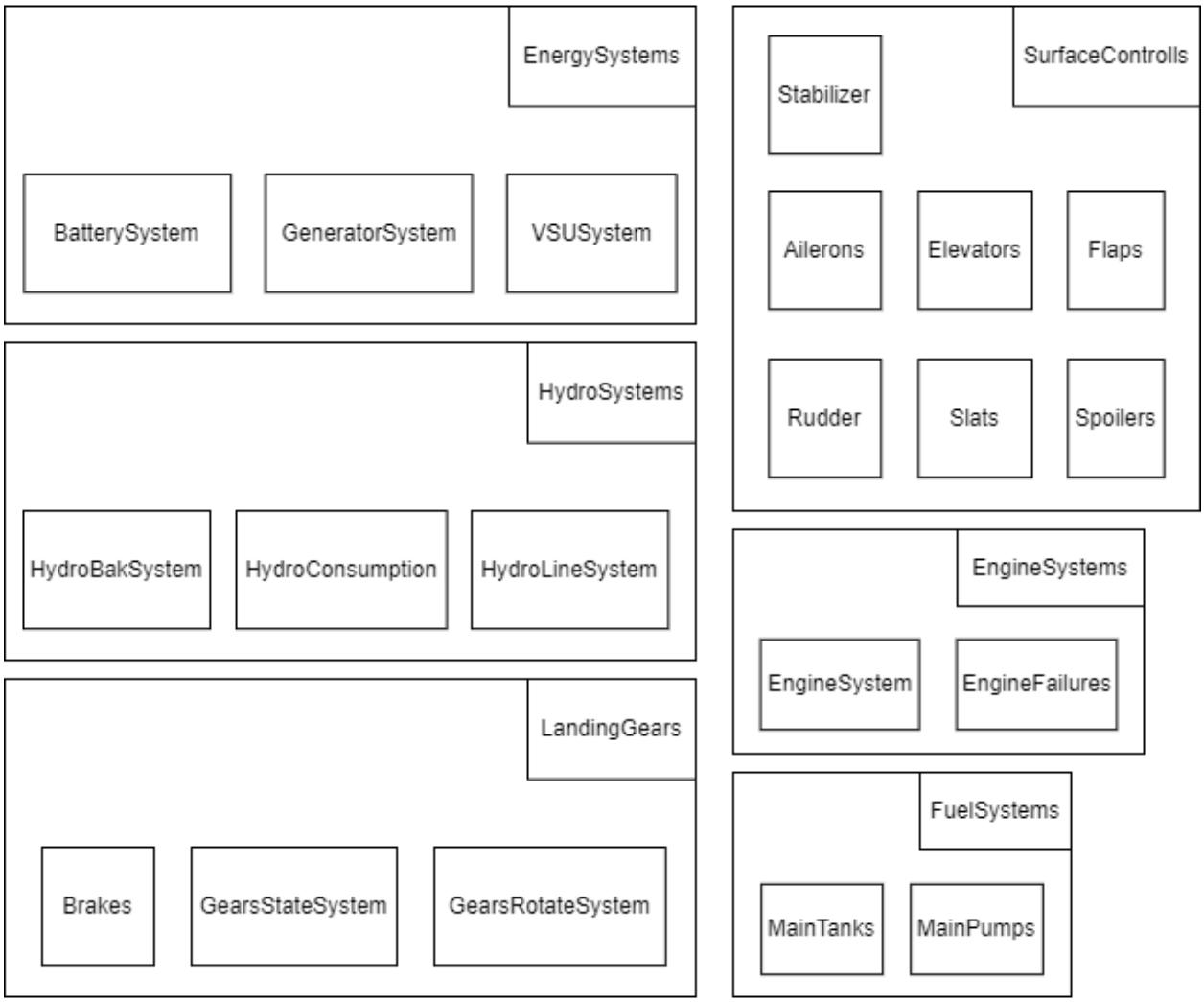
Структура систем літака

(Структурна схема)

ІАЛЦ. 466530.004 Д1

Аркушів 1

Київ – 2023 р.



					ІАЛЦ.466530.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Діденко В.В.			Система автопілотування літаків побудована на pid регуляторах. Структура систем літака (Структурна схема)	Лит. / Арк. 1 / Аркушів 1
Перев.		Каплунов А.В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-91
Реценз.						
Н. Контр.		Виноградов Ю.М.				
Затверд		Стіренко С. Г.				

ДОДАТОК 2

Система автопілотування літаків побудована на pid регуляторах

Діаграма варіантів користування

(Функціональна схема)

ІАЛЦ. 466530.005 Д2

Аркушів 1

Київ – 2023 р.

					ІАЛЦ.466530.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Діденко В.В.			Система автопілотування літаків побудована на pid регуляторах. Діаграма варіантів користування (Функціональна схема)	Лит. / Арк. 1 / Аркушів 1
Перев.		Каплунов А.В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-91
Реценз.						
Н. Контр.		Виноградов Ю.М.				
Затверд		Стіренко С. Г.				

ДОДАТОК З

Система автопілотування літаків побудована на pid регуляторах

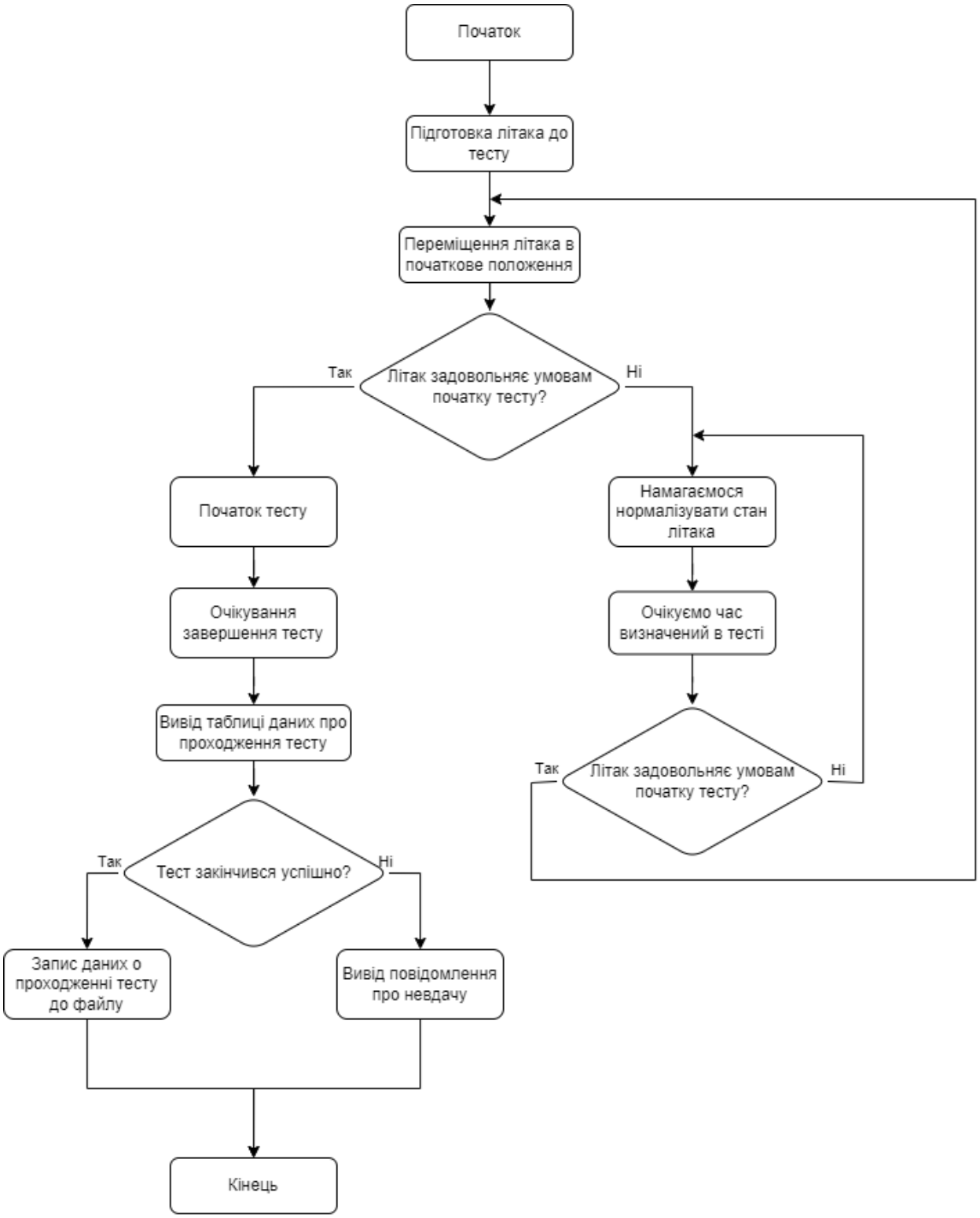
Алгоритм дій програмного забезпечення

(Принципова схема)

ІАЛЦ. 466530.006 ДЗ

Аркушів 1

Київ – 2023 р.



					ІАЛЦ.466530.006 ДЗ	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Діденко В.В.			Система автопілотування літаків побудована на pid регуляторах. Алгоритм дій програмного забезпечення (Принципова схема)	Лит. / Арк. 1 / Аркушів 1
Перев.		Каплунов А.В.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-91
Реценз.						
Н. Контр.		Виноградов Ю.М.				
Затверд		Стіренко С. Г.				

ДОДАТОК 4

Система автопілотування літаків побудована на pid регуляторах

Текст програмного коду

ІАЛЦ. 466530.007 Д4

Аркушів 15

Київ – 2023 р.

```
localVIRTUAL=project_settings.panel
s.USE_VIRTUAL_PANELS
components={

EnginesCS{
_systemName="engineCS",

_iVibrationCheckRotator=globalPrope
rtyf(ENGINES_NAME.."/vibrationCheck
Rotator"),
_iVibrationEngine1FWD=globalPropert
yf(ENGINES_NAME.."/engine1/vibratio
nFWD"),
_iVibrationEngine2FWD=globalPropert
yf(ENGINES_NAME.."/engine2/vibratio
nFWD"),
_iVibrationEngine3FWD=globalPropert
yf(ENGINES_NAME.."/engine3/vibratio
nFWD"),
_iVibrationEngine4FWD=globalPropert
yf(ENGINES_NAME.."/engine4/vibratio
nFWD"),
_iVibrationEngine1BACK=globalProper
tyf(ENGINES_NAME.."/engine1/vibrati
onBACK"),
_iVibrationEngine2BACK=globalProper
tyf(ENGINES_NAME.."/engine2/vibrati
onBACK"),
_iVibrationEngine3BACK=globalProper
tyf(ENGINES_NAME.."/engine3/vibrati
onBACK"),
_iVibrationEngine4BACK=globalProper
tyf(ENGINES_NAME.."/engine4/vibrati
onBACK"),
_iMinDavlToplIndicator1=globalPrope
rtyf(ENGINES_NAME.."/engine1/minDav
lMaslaIndicator"),
_iMinDavlToplIndicator2=globalPrope
rtyf(ENGINES_NAME.."/engine2/minDav
lMaslaIndicator"),
_iMinDavlToplIndicator3=globalPrope
rtyf(ENGINES_NAME.."/engine3/minDav
lMaslaIndicator"),
_iMinDavlToplIndicator4=globalPrope
rtyf(ENGINES_NAME.."/engine4/minDav
lMaslaIndicator"),
_iHighVibration1=globalPropertyf(EN
GINES_NAME.."/engine1/highVibration
"),
_iHighVibration2=globalPropertyf(EN
GINES_NAME.."/engine2/highVibration
"),
_iHighVibration3=globalPropertyf(EN
GINES_NAME.."/engine3/highVibration
"),
_iHighVibration4=globalPropertyf(EN
GINES_NAME.."/engine4/highVibration
"),
_oVibrationArrow=globalPropertyf(EN
GINES_NAME.."/vibrationArrow"),

_oMinDavlToplIndicator=globalProper
tyf(ENGINES_NAME.."/minDavlMaslaInd
icator"),
_oHighVibration=globalPropertyf(ENG
INES_NAME.."/highVibration"),
},
```

```
EngineSystem{
_systemName="engine1",
_pEngineNumber=1,
_pIndicationPower=5*27,
_pNormalVoltage=27,
_pMinV=20,
_pMin_VSU_RPM=0.8,
_pOilAmount=globalPropertyf(ENGINES
_NAME.."/pNormalOilAmount"),
_pNormalFuelPressure=globalProperty
f(ENGINES_NAME.."/pNormalFuelPressu
re"),
_pHighVibrationValue=globalProperty
f(ENGINES_NAME.."/pHighVibrationVal
ue"),
_pDangerVibrationValue=globalProper
tyf(ENGINES_NAME.."/pDangerVibratio
nValue"),

_iVoltage=globalPropertyf(ENERGY_NA
ME.."/CruDCSystem/cruLeft_DC/oDC27_
V_A"),
_iThrotle=globalPropertyf(ENGINES_N
AME.."/engine1/throttle"),
_iThrottleAll=globalPropertyf(ENGIN
ES_NAME.."/throttleAll"),
_iSeparateThrottle=globalPropertyf(
ENGINES_NAME.."/separateThrottle"),
_iVibrationCheckButton=globalProper
tyf(ENGINES_NAME.."/vibrationCheckB
utton"),
_iVibrationCheckRotator=globalPrope
rtyf(ENGINES_NAME.."/vibrationCheck
Rotator"),
_iEngineStart=globalPropertyf(ENGIN
ES_NAME.."/engine1/startEngine"),
_iStopSwitch=globalPropertyf(ENGINE
S_NAME.."/engine1/stopSwitch"),
_iRodRaboty=globalPropertyf(ENGINES
_NAME.."/engine1/rodRaboty"),
_iReverseSwitch=globalPropertyf(ENG
INES_NAME.."/engine1/reverseSwitch"
),
_iReverseSwitchAll=globalPropertyf(
ENGINES_NAME.."/reverseSwitchAll"),
_iVSU_RPM=globalPropertyf(ENERGY_NA
ME.."/VsuSystem/vsu/engineRPM_Perce
nt"),
_iEngineStartInAirButton=globalProp
ertyf(ENGINES_NAME.."/engine1/puskV
Vozduhe"),

_iStarterFailed=globalPropertyf(ENG
INES_FAILURES.."/engine1/starterFai
led"),
_iFuelFireFailed=globalPropertyf(EN
GINES_FAILURES.."/engine1/fuelFireF
ailed"),
_iZabrosTVGFail=globalPropertyf(ENG
INES_FAILURES.."/engine1/zabrosTVGF
ail"),
_iStruzhkaVMasle=globalPropertyf(EN
GINES_FAILURES.."/engine1/struzhkaV
Masle"),
_iPovyshRashodMasla=globalPropertyf
(ENGINES_FAILURES.."/engine1/povysh
RashodMasla"),
_iMinDavlTopliva=globalPropertyf(EN
GINES_FAILURES.."/engine1/minDavlTo
pliva"),
_iMinDavlMasla=globalPropertyf(ENGI
NES_FAILURES.."/engine1/minDavlMasl
a"),
_iPodachaToplFailure=globalProperty
f(ENGINES_FAILURES.."/engine1/podac
haTopl"),
```

```
_iPodkachNasosFailure=globalPropert
yf(ENGINES_FAILURES.."/engine1/podk
achNasos"),
_iZamokReversaFailure=globalPropert
yf(ENGINES_FAILURES.."/engine1/zamo
kReversa"),
_iNasosFailure=globalPropertyf(ENGI
NES_FAILURES.."/engine1/nasos"),
_iProizvVklReversa=globalPropertyf(
ENGINES_FAILURES.."/engine1/proizvV
klReversa"),
_iOpasnayaVibraciya=globalPropertyf
(ENGINES_FAILURES.."/engine1/opasna
yaVibraciya"),

_oEngineState=globalPropertyf(ENGIN
ES_NAME.."/engine1/state"),
_oConsumption=globalPropertyf(ENGIN
ES_NAME.."/engine1/consumption"),
_oVibrationFWD=globalPropertyf(ENGI
NES_NAME.."/engine1/vibrationFWD"),
_oVibrationBACK=globalPropertyf(ENG
INES_NAME.."/engine1/vibrationBACK"
),
_oHighVibration=globalPropertyf(ENG
INES_NAME.."/engine1/highVibration"
),
_oDangerVibration=globalPropertyf(E
NGINES_NAME.."/engine1/dangerVibrat
ion"),
_oPanelZapuskaRab=globalPropertyf(E
NGINES_NAME.."/engine1/panelZapuska
Rab"),
_oReverseOnIndication=globalPropert
yf(ENGINES_NAME.."/engine1/reverseO
nIndication"),
_oEngineRPM_Percent=globalPropertyf
(ENGINES_NAME.."/engine1/engineRPM_
Percent"),
_oOpasOborotyStatera=globalProperty
f(ENGINES_NAME.."/engine1/opasOboro
tyStatera"),
_oMinOilRemaining=globalPropertyf(E
NGINES_NAME.."/engine1/minOilRemain
ing"),

_oOpasnayaVibraciyaIndicator=global
Propertyf(ENGINES_NAME.."/engine1/o
pasnayaVibraciyaIndicator"),
_oMinDavlToplIndicator=globalProper
tyf(ENGINES_NAME.."/engine1/minDavl
ToplIndicator"),
_oStruzhkaVMasleIndicator=globalPro
pertyf(ENGINES_NAME.."/engine1/stru
zhkaVMasleIndicator"),
_oOstTop2000Indicator=globalPropert
yf(ENGINES_NAME.."/engine1/ostTop20
00Indicator"),
_oMinDavlMaslaIndicator=globalPrope
rtyf(ENGINES_NAME.."/engine1/minDav
lMaslaIndicator"),
_oToplFiltrNeRabIndicator=globalPro
pertyf(ENGINES_NAME.."/engine1/topl
FiltrNeRabIndicator"),
_oPerezapuskVozdOtkrIndicator=globa
lPropertyf(ENGINES_NAME.."/engine1/
perezapuskVozdOtkrIndicator"),
_oVNARabIndicator=globalPropertyf(E
NGINES_NAME.."/engine1/VNARabIndica
tor"),
_oZamRevOtkrytIndicator=globalPrope
rtyf(ENGINES_NAME.."/engine1/zamRev
OtkrytIndicator"),
```

```
_oVNAPuskIndicator=globalPropertyf(
ENGINES_NAME.."/engine1/VNAPuskIndi
cator"),

_oActualN1=globalPropertyf(ENGINES_
NAME.."/engine1/actualN1"),
_oActualN2=globalPropertyf(ENGINES_
NAME.."/engine1/actualN2"),
_oSuppostedN1=globalPropertyf(ENGIN
ES_NAME.."/engine1/suppostedN1"),
_oSuppostedN2=globalPropertyf(ENGIN
ES_NAME.."/engine1/suppostedN2"),
_oActualFuelPressure=globalProperty
f(ENGINES_NAME.."/engine1/actualFue
lPressure"),
_oSuppostedFuelPressure=globalPrope
rtyf(ENGINES_NAME.."/engine1/suppos
tedFuelPressure"),
_oActualOilPressure=globalPropertyf
(ENGINES_NAME.."/engine1/actualOilP
ressure"),
_oSuppostedOilPressure=globalProper
tyf(ENGINES_NAME.."/engine1/suppost
edOilPressure"),
_oActualOilTemperature=globalProper
tyf(ENGINES_NAME.."/engine1/actualO
ilTemperature"),
_oSuppostedOilTemperature=globalPro
pertyf(ENGINES_NAME.."/engine1/supp
ostedOilTemperature"),
_oActualFuelConsumption=globalPrope
rtyf(ENGINES_NAME.."/engine1/actual
FuelConsumption"),
_oSuppostedFuelConsumption=globalPr
opertyf(ENGINES_NAME.."/engine1/sup
postedFuelConsumption"),
_oActualFuelRemaining=globalPropert
yf(ENGINES_NAME.."/engine1/actualFu
elRemaining"),
_oSuppostedFuelRemaining=globalProp
ertyf(ENGINES_NAME.."/engine1/suppo
stedFuelRemaining"),
_oActualGasTemperature=globalProper
tyf(ENGINES_NAME.."/engine1/actualG
asTemperature"),
_oSuppostedGasTemperature=globalPro
pertyf(ENGINES_NAME.."/engine1/supp
ostedGasTemperature"),
_oSuppostedOilRemaining=globalPrope
rtyf(ENGINES_NAME.."/engine1/suppos
tedOilRemaining"),
_oSuppostedFuelPressure=globalPrope
rtyf(ENGINES_NAME.."/engine1/suppos
tedFuelPressure"),

},

EngineSystem{
_systemName="engine2",
_pEngineNumber=2,
_pIndicationPower=5*27,
_pNormalVoltage=27,
_pMinV=20,
_pMin_VSU_RPM=0.8,
_pOilAmount=globalPropertyf(ENGINES
_NAME.."/pNormalOilAmount"),
_pNormalFuelPressure=globalProperty
f(ENGINES_NAME.."/pNormalFuelPressu
re"),
_pHighVibrationValue=globalProperty
f(ENGINES_NAME.."/pHighVibrationVal
ue"),
```

```
_pDangerVibrationValue=globalProper
tyf(ENGINES_NAME.."/pDangerVibratio
nValue"),

_iVoltage=globalPropertyf(ENERGY_NA
ME.."/CruDCSystem/cruLeft_DC/oDC27_
V_A"),
_iThrotle=globalPropertyf(ENGINES_N
AME.."/engine2/throttle"),
_iThrottleAll=globalPropertyf(ENGIN
ES_NAME.."/throttleAll"),
_iSeparateThrottle=globalPropertyf(
ENGINES_NAME.."/separateThrottle"),
_iVibrationCheckButton=globalProper
tyf(ENGINES_NAME.."/vibrationCheckB
utton"),
_iVibrationCheckRotator=globalPrope
rtyf(ENGINES_NAME.."/vibrationCheck
Rotator"),
_iEngineStart=globalPropertyf(ENGIN
ES_NAME.."/engine2/startEngine"),
_iStopSwitch=globalPropertyf(ENGINE
S_NAME.."/engine2/stopSwitch"),
_iRodRaboty=globalPropertyf(ENGINES
_NAME.."/engine2/rodRaboty"),
_iReverseSwitch=globalPropertyf(ENG
INES_NAME.."/engine2/reverseSwitch"
),
_iReverseSwitchAll=globalPropertyf(
ENGINES_NAME.."/reverseSwitchAll"),
_iVSU_RPM=globalPropertyf(ENERGY_NA
ME.."/VsuSystem/vsu/engineRPM_Perce
nt"),
_iEngineStartInAirButton=globalProp
ertyf(ENGINES_NAME.."/engine2/puskV
Vozduhe"),

_iStarterFailed=globalPropertyf(ENG
INES_FAILURES.."/engine2/starterFai
led"),
_iFuelFireFailed=globalPropertyf(EN
GINES_FAILURES.."/engine2/fuelFireF
ailed"),
_iZabrosTVGFail=globalPropertyf(ENG
INES_FAILURES.."/engine2/zabrosTVGF
ail"),
_iStruzhkaVMasle=globalPropertyf(EN
GINES_FAILURES.."/engine2/struzhkaV
Masle"),
_iPovyshRashodMasla=globalPropertyf
(ENGINES_FAILURES.."/engine2/povysh
RashodMasla"),
_iMinDavlTopliva=globalPropertyf(EN
GINES_FAILURES.."/engine2/minDavlTo
pliva"),
_iMinDavlMasla=globalPropertyf(ENGI
NES_FAILURES.."/engine2/minDavlMasl
a"),
_iPodachaToplFailure=globalProperty
f(ENGINES_FAILURES.."/engine2/podac
haTopl"),
_iPodkachNasosFailure=globalPropert
yf(ENGINES_FAILURES.."/engine2/podk
achNasos"),
_iZamokReversaFailure=globalPropert
yf(ENGINES_FAILURES.."/engine2/zamo
kReversa"),
_iNasosFailure=globalPropertyf(ENGI
NES_FAILURES.."/engine2/nasos"),
_iProizvVklReversa=globalPropertyf(
ENGINES_FAILURES.."/engine2/proizvV
klReversa"),
_iOpasnayaVibraciya=globalPropertyf
(ENGINES_FAILURES.."/engine2/opasna
yaVibraciya"),
```

```
_oEngineState=globalPropertyf(ENGIN
ES_NAME.."/engine2/state"),
_oConsumption=globalPropertyf(ENGIN
ES_NAME.."/engine2/consumption"),
_oActualN1=globalPropertyf(ENGINES_
NAME.."/engine2/actualN1"),
_oActualN2=globalPropertyf(ENGINES_
NAME.."/engine2/actualN2"),
_oVibrationFWD=globalPropertyf(ENGI
NES_NAME.."/engine2/vibrationFWD"),
_oVibrationBACK=globalPropertyf(ENG
INES_NAME.."/engine2/vibrationBACK"
),
_oHighVibration=globalPropertyf(ENG
INES_NAME.."/engine2/highVibration"
),
_oDangerVibration=globalPropertyf(E
NGINES_NAME.."/engine2/dangerVibrat
ion"),
_oPanelZapuskaRab=globalPropertyf(E
NGINES_NAME.."/engine2/panelZapuska
Rab"),
_oReverseOnIndication=globalPropert
yf(ENGINES_NAME.."/engine2/reverseO
nIndication"),
_oEngineRPM_Percent=globalPropertyf
(ENGINES_NAME.."/engine2/engineRPM_
Percent"),
_oOpasOborotyStatera=globalProperty
f(ENGINES_NAME.."/engine2/opasOboro
tyStatera"),
_oMinOilRemaining=globalPropertyf(E
NGINES_NAME.."/engine2/minOilRemain
ing"),
_oOpasnayaVibraciyaIndicator=global
Propertyf(ENGINES_NAME.."/engine2/o
pasnayaVibraciyaIndicator"),
_oMinDavlToplIndicator=globalProper
tyf(ENGINES_NAME.."/engine2/minDavl
ToplIndicator"),
_oStruzhkaVMasleIndicator=globalPro
pertyf(ENGINES_NAME.."/engine2/stru
zhkaVMasleIndicator"),
_oOstTop2000Indicator=globalPropert
yf(ENGINES_NAME.."/engine2/ostTop20
00Indicator"),
_oMinDavlMaslaIndicator=globalPrope
rtyf(ENGINES_NAME.."/engine2/minDav
lMaslaIndicator"),
_oToplFiltrNeRabIndicator=globalPro
pertyf(ENGINES_NAME.."/engine2/topl
FiltrNeRabIndicator"),
_oPerezapuskVozdOtkrIndicator=globa
lPropertyf(ENGINES_NAME.."/engine2/
perezapuskVozdOtkrIndicator"),
_oVNARabIndicator=globalPropertyf(E
NGINES_NAME.."/engine2/VNARabIndica
tor"),
_oZamRevOtkrytIndicator=globalPrope
rtyf(ENGINES_NAME.."/engine2/zamRev
OtkrytIndicator"),
_oVNAPuskIndicator=globalPropertyf(
ENGINES_NAME.."/engine2/VNAPuskIndi
cator"),

_oSuppostedN1=globalPropertyf(ENGIN
ES_NAME.."/engine2/suppostedN1"),
_oSuppostedN2=globalPropertyf(ENGIN
ES_NAME.."/engine2/suppostedN2"),
_oActualFuelPressure=globalProperty
f(ENGINES_NAME.."/engine2/actualFue
lPressure"),
```

```
_oSuppostedFuelPressure=globalPrope
rtyf(ENGINES_NAME.."/engine2/suppos
tedFuelPressure"),
_oActualOilPressure=globalPropertyf
(ENGINES_NAME.."/engine2/actualOilP
ressure"),
_oSuppostedOilPressure=globalProper
tyf(ENGINES_NAME.."/engine2/suppost
edOilPressure"),
_oActualOilTemperature=globalProper
tyf(ENGINES_NAME.."/engine2/actualO
ilTemperature"),
_oSuppostedOilTemperature=globalPro
pertyf(ENGINES_NAME.."/engine2/supp
ostedOilTemperature"),
_oActualFuelConsumption=globalPrope
rtyf(ENGINES_NAME.."/engine2/actual
FuelConsumption"),
_oSuppostedFuelConsumption=globalPr
opertyf(ENGINES_NAME.."/engine2/sup
postedFuelConsumption"),
_oActualFuelRemaining=globalPropert
yf(ENGINES_NAME.."/engine2/actualFu
elRemaining"),
_oSuppostedFuelRemaining=globalProp
ertyf(ENGINES_NAME.."/engine2/suppo
stedFuelRemaining"),
_oActualGasTemperature=globalProper
tyf(ENGINES_NAME.."/engine2/actualG
asTemperature"),
_oSuppostedGasTemperature=globalPro
pertyf(ENGINES_NAME.."/engine2/supp
ostedGasTemperature"),
_oSuppostedOilRemaining=globalPrope
rtyf(ENGINES_NAME.."/engine2/suppos
tedOilRemaining"),
_oSuppostedFuelPressure=globalPrope
rtyf(ENGINES_NAME.."/engine2/suppos
tedFuelPressure"),
},

EngineSystem{
_systemName="engine3",
_pEngineNumber=3,
_pIndicationPower=5*27,
_pNormalVoltage=27,
_pMinV=20,
_pMin_VSU_RPM=0.8,
_pOilAmount=globalPropertyf(ENGINES
_NAME.."/pNormalOilAmount"),
_pNormalFuelPressure=globalProperty
f(ENGINES_NAME.."/pNormalFuelPressu
re"),
_pHighVibrationValue=globalProperty
f(ENGINES_NAME.."/pHighVibrationVal
ue"),
_pDangerVibrationValue=globalProper
tyf(ENGINES_NAME.."/pDangerVibratio
nValue"),

_iVoltage=globalPropertyf(ENERGY_NA
ME.."/CruDCSystem/cruLeft_DC/oDC27_
V_A"),
_iThrotle=globalPropertyf(ENGINES_N
AME.."/engine3/throttle"),
_iThrottleAll=globalPropertyf(ENGIN
ES_NAME.."/throttleAll"),
_iSeparateThrottle=globalPropertyf(
ENGINES_NAME.."/separateThrottle"),
_iVibrationCheckButton=globalProper
tyf(ENGINES_NAME.."/vibrationCheckB
utton"),
```

```
_iVibrationCheckRotator=globalPrope
rtyf(ENGINES_NAME.."/vibrationCheck
Rotator"),
_iEngineStart=globalPropertyf(ENGIN
ES_NAME.."/engine3/startEngine"),
_iStopSwitch=globalPropertyf(ENGINE
S_NAME.."/engine3/stopSwitch"),
_iRodRaboty=globalPropertyf(ENGINES
_NAME.."/engine3/rodRaboty"),
_iReverseSwitch=globalPropertyf(ENG
INES_NAME.."/engine3/reverseSwitch"
),
_iReverseSwitchAll=globalPropertyf(
ENGINES_NAME.."/reverseSwitchAll"),
_iVSU_RPM=globalPropertyf(ENERGY_NA
ME.."/VsuSystem/vsu/engineRPM_Perce
nt"),
_iEngineStartInAirButton=globalProp
ertyf(ENGINES_NAME.."/engine3/puskV
Vozduhe"),

_iStarterFailed=globalPropertyf(ENG
INES_FAILURES.."/engine3/starterFai
led"),
_iFuelFireFailed=globalPropertyf(EN
GINES_FAILURES.."/engine3/fuelFireF
ailed"),
_iZabrosTVGFail=globalPropertyf(ENG
INES_FAILURES.."/engine3/zabrosTVGF
ail"),
_iStruzhkaVMasle=globalPropertyf(EN
GINES_FAILURES.."/engine3/struzhkaV
Masle"),
_iPovyshRashodMasla=globalPropertyf
(ENGINES_FAILURES.."/engine3/povysh
RashodMasla"),
_iMinDavlTopliva=globalPropertyf(EN
GINES_FAILURES.."/engine3/minDavlTo
pliva"),
_iMinDavlMasla=globalPropertyf(ENGI
NES_FAILURES.."/engine3/minDavlMasl
a"),
_iPodachaToplFailure=globalProperty
f(ENGINES_FAILURES.."/engine3/podac
haTopl"),
_iPodkachNasosFailure=globalPropert
yf(ENGINES_FAILURES.."/engine3/podk
achNasos"),
_iZamokReversaFailure=globalPropert
yf(ENGINES_FAILURES.."/engine3/zamo
kReversa"),
_iNasosFailure=globalPropertyf(ENGI
NES_FAILURES.."/engine3/nasos"),
_iProizvVklReversa=globalPropertyf(
ENGINES_FAILURES.."/engine3/proizvV
klReversa"),
_iOpasnayaVibraciya=globalPropertyf
(ENGINES_FAILURES.."/engine3/opasna
yaVibraciya"),

_oEngineState=globalPropertyf(ENGIN
ES_NAME.."/engine3/state"),
_oConsumption=globalPropertyf(ENGIN
ES_NAME.."/engine3/consumption"),
_oVibrationFWD=globalPropertyf(ENGI
NES_NAME.."/engine3/vibrationFWD"),
_oVibrationBACK=globalPropertyf(ENG
INES_NAME.."/engine3/vibrationBACK"
),
_oHighVibration=globalPropertyf(ENG
INES_NAME.."/engine3/highVibration"
),
```

```
_oDangerVibration=globalPropertyf(E
NGINES_NAME.."/engine3/dangerVibrat
ion"),
_oPanelZapuskaRab=globalPropertyf(E
NGINES_NAME.."/engine3/panelZapuska
Rab"),
_oReverseOnIndication=globalPropert
yf(ENGINES_NAME.."/engine3/reverseO
nIndication"),
_oEngineRPM_Percent=globalPropertyf
(ENGINES_NAME.."/engine3/engineRPM_
Percent"),
_oOpasOborotyStatera=globalProperty
f(ENGINES_NAME.."/engine3/opasOboro
tyStatera"),
_oMinOilRemaining=globalPropertyf(E
NGINES_NAME.."/engine3/minOilRemain
ing"),

_oOpasnayaVibraciyaIndicator=global
Propertyf(ENGINES_NAME.."/engine3/o
pasnayaVibraciyaIndicator"),
_oMinDavlToplIndicator=globalProper
tyf(ENGINES_NAME.."/engine3/minDavl
ToplIndicator"),
_oStruzhkaVMasleIndicator=globalPro
pertyf(ENGINES_NAME.."/engine3/stru
zhkaVMasleIndicator"),
_oOstTop2000Indicator=globalPropert
yf(ENGINES_NAME.."/engine3/ostTop20
00Indicator"),
_oMinDavlMaslaIndicator=globalPrope
rtyf(ENGINES_NAME.."/engine3/minDav
lMaslaIndicator"),
_oToplFiltrNeRabIndicator=globalPro
pertyf(ENGINES_NAME.."/engine3/topl
FiltrNeRabIndicator"),
_oPerezapuskVozdOtkrIndicator=globa
lPropertyf(ENGINES_NAME.."/engine3/
perezapuskVozdOtkrIndicator"),
_oVNARabIndicator=globalPropertyf(E
NGINES_NAME.."/engine3/VNARabIndica
tor"),
_oZamRevOtkrytIndicator=globalPrope
rtyf(ENGINES_NAME.."/engine3/zamRev
OtkrytIndicator"),
_oVNAPuskIndicator=globalPropertyf(
ENGINES_NAME.."/engine3/VNAPuskIndi
cator"),


_oActualN1=globalPropertyf(ENGINES_
NAME.."/engine3/actualN1"),
_oActualN2=globalPropertyf(ENGINES_
NAME.."/engine3/actualN2"),
_oSuppostedN1=globalPropertyf(ENGIN
ES_NAME.."/engine3/suppostedN1"),
_oSuppostedN2=globalPropertyf(ENGIN
ES_NAME.."/engine3/suppostedN2"),
_oActualFuelPressure=globalProperty
f(ENGINES_NAME.."/engine3/actualFue
lPressure"),
_oSuppostedFuelPressure=globalPrope
rtyf(ENGINES_NAME.."/engine3/suppos
tedFuelPressure"),
_oActualOilPressure=globalPropertyf
(ENGINES_NAME.."/engine3/actualOilP
ressure"),
_oSuppostedOilPressure=globalProper
tyf(ENGINES_NAME.."/engine3/suppost
edOilPressure"),
_oActualOilTemperature=globalProper
tyf(ENGINES_NAME.."/engine3/actualO
ilTemperature"),
```

```
_oSuppostedOilTemperature=globalPro
pertyf(ENGINES_NAME.."/engine3/supp
ostedOilTemperature"),
_oActualFuelConsumption=globalPrope
rtyf(ENGINES_NAME.."/engine3/actual
FuelConsumption"),
_oSuppostedFuelConsumption=globalPr
opertyf(ENGINES_NAME.."/engine3/sup
postedFuelConsumption"),
_oActualFuelRemaining=globalPropert
yf(ENGINES_NAME.."/engine3/actualFu
elRemaining"),
_oSuppostedFuelRemaining=globalProp
ertyf(ENGINES_NAME.."/engine3/suppo
stedFuelRemaining"),
_oActualGasTemperature=globalProper
tyf(ENGINES_NAME.."/engine3/actualG
asTemperature"),
_oSuppostedGasTemperature=globalPro
pertyf(ENGINES_NAME.."/engine3/supp
ostedGasTemperature"),
_oSuppostedOilRemaining=globalPrope
rtyf(ENGINES_NAME.."/engine3/suppos
tedOilRemaining"),
_oSuppostedFuelPressure=globalPrope
rtyf(ENGINES_NAME.."/engine3/suppos
tedFuelPressure"),
},

EngineSystem{
_systemName="engine4",
_pEngineNumber=4,
_pIndicationPower=5*27,
_pNormalVoltage=27,
_pMinV=20,
_pMin_VSU_RPM=0.8,
_pOilAmount=globalPropertyf(ENGINES
_NAME.."/pNormalOilAmount"),
_pNormalFuelPressure=globalProperty
f(ENGINES_NAME.."/pNormalFuelPressu
re"),
_pHighVibrationValue=globalProperty
f(ENGINES_NAME.."/pHighVibrationVal
ue"),
_pDangerVibrationValue=globalProper
tyf(ENGINES_NAME.."/pDangerVibratio
nValue"),

_iVoltage=globalPropertyf(ENERGY_NA
ME.."/CruDCSystem/cruLeft_DC/oDC27_
V_A"),
_iThrotle=globalPropertyf(ENGINES_N
AME.."/engine4/throttle"),
_iThrottleAll=globalPropertyf(ENGIN
ES_NAME.."/throttleAll"),
_iSeparateThrottle=globalPropertyf(
ENGINES_NAME.."/separateThrottle"),
_iVibrationCheckButton=globalProper
tyf(ENGINES_NAME.."/vibrationCheckB
utton"),
_iVibrationCheckRotator=globalPrope
rtyf(ENGINES_NAME.."/vibrationCheck
Rotator"),
_iEngineStart=globalPropertyf(ENGIN
ES_NAME.."/engine4/startEngine"),
_iStopSwitch=globalPropertyf(ENGINE
S_NAME.."/engine4/stopSwitch"),
_iRodRaboty=globalPropertyf(ENGINES
_NAME.."/engine4/rodRaboty"),
_iReverseSwitch=globalPropertyf(ENG
INES_NAME.."/engine4/reverseSwitch"
),
_iReverseSwitchAll=globalPropertyf(
ENGINES_NAME.."/reverseSwitchAll"),
```

```
_iVSU_RPM=globalPropertyf(ENERGY_NAME.."/VsuSystem/vsu/engineRPM_Percent"),
_iEngineStartInAirButton=globalPropertyf(ENGINES_NAME.."/engine4/puskVVozduhe"),


_iStarterFailed=globalPropertyf(ENGINES_FAILURES.."/engine4/starterFailed"),
_iFuelFireFailed=globalPropertyf(ENGINES_FAILURES.."/engine4/fuelFireFailed"),
_iZabrosTVGFail=globalPropertyf(ENGINES_FAILURES.."/engine4/zabrosTVGFail"),
_iStruzhkaVMasle=globalPropertyf(ENGINES_FAILURES.."/engine4/struzhkaVMasle"),
_iPovyshRashodMasla=globalPropertyf(ENGINES_FAILURES.."/engine4/povyshRashodMasla"),
_iMinDavlTopliva=globalPropertyf(ENGINES_FAILURES.."/engine4/minDavlTopliva"),
_iMinDavlMasla=globalPropertyf(ENGINES_FAILURES.."/engine4/minDavlMasla"),
_iPodachaToplFailure=globalPropertyf(ENGINES_FAILURES.."/engine4/podachaTopl"),
_iPodkachNasosFailure=globalPropertyf(ENGINES_FAILURES.."/engine4/podkachNasos"),
_iZamokReversaFailure=globalPropertyf(ENGINES_FAILURES.."/engine4/zamokReversa"),
_iNasosFailure=globalPropertyf(ENGINES_FAILURES.."/engine4/nasos"),
_iProizvVklReversa=globalPropertyf(ENGINES_FAILURES.."/engine4/proizvVklReversa"),
_iOpasnayaVibraciya=globalPropertyf(ENGINES_FAILURES.."/engine4/opasnayaVibraciya"),

_oEngineState=globalPropertyf(ENGINES_NAME.."/engine4/state"),
_oConsumption=globalPropertyf(ENGINES_NAME.."/engine4/consumption"),
_oVibrationFWD=globalPropertyf(ENGINES_NAME.."/engine4/vibrationFWD"),
_oVibrationBACK=globalPropertyf(ENGINES_NAME.."/engine4/vibrationBACK"),
_oHighVibration=globalPropertyf(ENGINES_NAME.."/engine4/highVibration"),
_oDangerVibration=globalPropertyf(ENGINES_NAME.."/engine4/dangerVibration"),
_oPanelZapuskaRab=globalPropertyf(ENGINES_NAME.."/engine4/panelZapuskaRab"),
_oReverseOnIndication=globalPropertyf(ENGINES_NAME.."/engine4/reverseOnIndication"),
_oEngineRPM_Percent=globalPropertyf(ENGINES_NAME.."/engine4/engineRPM_Percent"),
_oOpasOborotyStatera=globalPropertyf(ENGINES_NAME.."/engine4/opasOborotyStatera"),
```

```
_oMinOilRemaining=globalPropertyf(ENGINES_NAME.."/engine4/minOilRemaining"),

_oOpasnayaVibraciyaIndicator=globalPropertyf(ENGINES_NAME.."/engine4/opasnayaVibraciyaIndicator"),
_oMinDavlToplIndicator=globalPropertyf(ENGINES_NAME.."/engine4/minDavlToplIndicator"),
_oStruzhkaVMasleIndicator=globalPropertyf(ENGINES_NAME.."/engine4/struzhkaVMasleIndicator"),
_oOstTop2000Indicator=globalPropertyf(ENGINES_NAME.."/engine4/ostTop2000Indicator"),
_oMinDavlMaslaIndicator=globalPropertyf(ENGINES_NAME.."/engine4/minDavlMaslaIndicator"),
_oToplFiltrNeRabIndicator=globalPropertyf(ENGINES_NAME.."/engine4/toplFiltrNeRabIndicator"),
_oPerezapuskVozdOtkrIndicator=globalPropertyf(ENGINES_NAME.."/engine4/perezapuskVozdOtkrIndicator"),
_oVNARabIndicator=globalPropertyf(ENGINES_NAME.."/engine4/VNARabIndicator"),
_oZamRevOtkrytIndicator=globalPropertyf(ENGINES_NAME.."/engine4/zamRevOtkrytIndicator"),
_oVNAPuskIndicator=globalPropertyf(ENGINES_NAME.."/engine4/VNAPuskIndicator"),

_oActualN1=globalPropertyf(ENGINES_NAME.."/engine4/actualN1"),
_oActualN2=globalPropertyf(ENGINES_NAME.."/engine4/actualN2"),
_oSuppostedN1=globalPropertyf(ENGINES_NAME.."/engine4/suppostedN1"),
_oSuppostedN2=globalPropertyf(ENGINES_NAME.."/engine4/suppostedN2"),
_oActualFuelPressure=globalPropertyf(ENGINES_NAME.."/engine4/actualFuelPressure"),
_oSuppostedFuelPressure=globalPropertyf(ENGINES_NAME.."/engine4/suppostedFuelPressure"),
_oActualOilPressure=globalPropertyf(ENGINES_NAME.."/engine4/actualOilPressure"),
_oSuppostedOilPressure=globalPropertyf(ENGINES_NAME.."/engine4/suppostedOilPressure"),
_oActualOilTemperature=globalPropertyf(ENGINES_NAME.."/engine4/actualOilTemperature"),
_oSuppostedOilTemperature=globalPropertyf(ENGINES_NAME.."/engine4/suppostedOilTemperature"),
_oActualFuelConsumption=globalPropertyf(ENGINES_NAME.."/engine4/actualFuelConsumption"),
_oSuppostedFuelConsumption=globalPropertyf(ENGINES_NAME.."/engine4/suppostedFuelConsumption"),
_oActualFuelRemaining=globalPropertyf(ENGINES_NAME.."/engine4/actualFuelRemaining"),
_oSuppostedFuelRemaining=globalPropertyf(ENGINES_NAME.."/engine4/suppostedFuelRemaining"),
```

```
_oActualGasTemperature=globalProper
tyf(ENGINES_NAME.."/engine4/actualG
asTemperature"),
_oSuppostedGasTemperature=globalPro
pertyf(ENGINES_NAME.."/engine4/supp
ostedGasTemperature"),
_oSuppostedOilRemaining=globalPrope
rtyf(ENGINES_NAME.."/engine4/suppos
tedOilRemaining"),
_oSuppostedFuelPressure=globalPrope
rtyf(ENGINES_NAME.."/engine4/suppos
tedFuelPressure"),
},

EngineFailures{
_systemName="engineFailures",

_iEngineState1=globalPropertyf(ENGI
NES_NAME.."/engine1/state"),
_iEngineState2=globalPropertyf(ENGI
NES_NAME.."/engine2/state"),
_iEngineState3=globalPropertyf(ENGI
NES_NAME.."/engine3/state"),
_iEngineState4=globalPropertyf(ENGI
NES_NAME.."/engine4/state"),
_iEngineStopSwitch1=globalPropertyf
(ENGINES_NAME.."/engine1/stopSwitch
"),
_iEngineStopSwitch2=globalPropertyf
(ENGINES_NAME.."/engine2/stopSwitch
"),
_iEngineStopSwitch3=globalPropertyf
(ENGINES_NAME.."/engine3/stopSwitch
"),
_iEngineStopSwitch4=globalPropertyf
(ENGINES_NAME.."/engine4/stopSwitch
"),

_iEngineAZS1=deviceHelper.getDatare
f(PHYSICAL.."/DEV_RU23_AZS/discrete
Input/Zapusk1Dv"),
_iEngineAZS2=deviceHelper.getDatare
f(PHYSICAL.."/DEV_RU23_AZS/discrete
Input/Zapusk2Dv"),
_iEngineAZS3=deviceHelper.getDatare
f(PHYSICAL.."/DEV_RU24_AZS/discrete
Input/Zapusk3Dv"),
_iEngineAZS4=deviceHelper.getDatare
f(PHYSICAL.."/DEV_RU24_AZS/discrete
Input/Zapusk4Dv"),

_oStarterFailed1=globalPropertyf(EN
GINES_FAILURES.."/engine1/starterFa
iled"),
_oStarterFailed2=globalPropertyf(EN
GINES_FAILURES.."/engine2/starterFa
iled"),
_oStarterFailed3=globalPropertyf(EN
GINES_FAILURES.."/engine3/starterFa
iled"),
_oStarterFailed4=globalPropertyf(EN
GINES_FAILURES.."/engine4/starterFa
iled"),

_oFuelFireFailed1=globalPropertyf(E
NGINES_FAILURES.."/engine1/fuelFire
Failed"),
_oFuelFireFailed2=globalPropertyf(E
NGINES_FAILURES.."/engine2/fuelFire
Failed"),
_oFuelFireFailed3=globalPropertyf(E
NGINES_FAILURES.."/engine3/fuelFire
Failed"),
```

```
_oFuelFireFailed4=globalPropertyf(ENGINES_FAILURES.."/engine4/fuelFireFailed"),

_oZabrosTVG1=globalPropertyf(ENGINES_FAILURES.."/engine1/zabrosTVGFail"),
_oZabrosTVG2=globalPropertyf(ENGINES_FAILURES.."/engine2/zabrosTVGFail"),
_oZabrosTVG3=globalPropertyf(ENGINES_FAILURES.."/engine3/zabrosTVGFail"),
_oZabrosTVG4=globalPropertyf(ENGINES_FAILURES.."/engine4/zabrosTVGFail"),

_oStruzhkaVMasle1=globalPropertyf(ENGINES_FAILURES.."/engine1/struzhkaVMasle"),
_oStruzhkaVMasle2=globalPropertyf(ENGINES_FAILURES.."/engine2/struzhkaVMasle"),
_oStruzhkaVMasle3=globalPropertyf(ENGINES_FAILURES.."/engine3/struzhkaVMasle"),
_oStruzhkaVMasle4=globalPropertyf(ENGINES_FAILURES.."/engine4/struzhkaVMasle"),

_oPovyshRashodMasla1=globalPropertyf(ENGINES_FAILURES.."/engine1/povyshRashodMasla"),
_oPovyshRashodMasla2=globalPropertyf(ENGINES_FAILURES.."/engine2/povyshRashodMasla"),
_oPovyshRashodMasla3=globalPropertyf(ENGINES_FAILURES.."/engine3/povyshRashodMasla"),
_oPovyshRashodMasla4=globalPropertyf(ENGINES_FAILURES.."/engine4/povyshRashodMasla"),

_oMinDavlTopliva1=globalPropertyf(ENGINES_FAILURES.."/engine1/minDavlTopliva"),
_oMinDavlTopliva2=globalPropertyf(ENGINES_FAILURES.."/engine2/minDavlTopliva"),
_oMinDavlTopliva3=globalPropertyf(ENGINES_FAILURES.."/engine3/minDavlTopliva"),
_oMinDavlTopliva4=globalPropertyf(ENGINES_FAILURES.."/engine4/minDavlTopliva"),
_oMinDavlMasla1=globalPropertyf(ENGINES_FAILURES.."/engine1/minDavlMasla"),
_oMinDavlMasla2=globalPropertyf(ENGINES_FAILURES.."/engine2/minDavlMasla"),
_oMinDavlMasla3=globalPropertyf(ENGINES_FAILURES.."/engine3/minDavlMasla"),
_oMinDavlMasla4=globalPropertyf(ENGINES_FAILURES.."/engine4/minDavlMasla"),
_oPodachaTopl1=globalPropertyf(ENGINES_FAILURES.."/engine1/podachaTopl"),
_oPodachaTopl2=globalPropertyf(ENGINES_FAILURES.."/engine2/podachaTopl"),
```

```
_oPodachaTopl3=globalPropertyf(ENGI
NES_FAILURES.."/engine3/podachaTopl
"),
_oPodachaTopl4=globalPropertyf(ENGI
NES_FAILURES.."/engine4/podachaTopl
"),
_oPodkachNasos1=globalPropertyf(ENG
INES_FAILURES.."/engine1/podkachNas
os"),
_oPodkachNasos2=globalPropertyf(ENG
INES_FAILURES.."/engine2/podkachNas
os"),
_oPodkachNasos3=globalPropertyf(ENG
INES_FAILURES.."/engine3/podkachNas
os"),
_oPodkachNasos4=globalPropertyf(ENG
INES_FAILURES.."/engine4/podkachNas
os"),
_oZamokReversa1=globalPropertyf(ENG
INES_FAILURES.."/engine1/zamokRever
sa"),
_oZamokReversa2=globalPropertyf(ENG
INES_FAILURES.."/engine2/zamokRever
sa"),
_oZamokReversa3=globalPropertyf(ENG
INES_FAILURES.."/engine3/zamokRever
sa"),
_oZamokReversa4=globalPropertyf(ENG
INES_FAILURES.."/engine4/zamokRever
sa"),
_oNasos1=globalPropertyf(ENGINES_FA
ILURES.."/engine1/nasos"),
_oNasos2=globalPropertyf(ENGINES_FA
ILURES.."/engine2/nasos"),
_oNasos3=globalPropertyf(ENGINES_FA
ILURES.."/engine3/nasos"),
_oNasos4=globalPropertyf(ENGINES_FA
ILURES.."/engine4/nasos"),
_oProizvVklReversa1=globalPropertyf
(ENGINES_FAILURES.."/engine1/proizv
VklReversa"),
_oProizvVklReversa2=globalPropertyf
(ENGINES_FAILURES.."/engine2/proizv
VklReversa"),
_oProizvVklReversa3=globalPropertyf
(ENGINES_FAILURES.."/engine3/proizv
VklReversa"),
_oProizvVklReversa4=globalPropertyf
(ENGINES_FAILURES.."/engine4/proizv
VklReversa"),
_oOpasnayaVibraciya1=globalProperty
f(ENGINES_FAILURES.."/engine1/opasn
ayaVibraciya"),
_oOpasnayaVibraciya2=globalProperty
f(ENGINES_FAILURES.."/engine2/opasn
ayaVibraciya"),
_oOpasnayaVibraciya3=globalProperty
f(ENGINES_FAILURES.."/engine3/opasn
ayaVibraciya"),
_oOpasnayaVibraciya4=globalProperty
f(ENGINES_FAILURES.."/engine4/opasn
ayaVibraciya"),

},

}
}
```