СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___*інформаційних технологій та електроніки*_____

Кафедра _____*інформаційних технологій та програмування*_____

Пояснювальна записка

до магістерської дипломної роботи

_____*Магістр*_____
(освітньо-кваліфікаційний рівень)

на тему: ____*Система верифікації цифрового двійника літака*_____
_____

Виконав: *студент 2 курсу, групи ІСТ-23дм*
126 «Інформаційні системи та технології»
(шифр і назва спеціальності)
_____*Діденко В.В.*_____
(прізвище та ініціали)
Керівник __*Меняйленко О.С.*_____
(прізвище та ініціали)
Рецензент_*Ратов Д.В.*_____
(прізвище та ініціали)

Київ – 2024 року

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет *інформаційних технологій та електроніки*
Кафедра *інформаційних технологій та програмування*
Освітньо-кваліфікаційний рівень *магістр*
Спеціальність *126 «Інформаційні системи та технології»*
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
_____ д.т.н., доц. Захожай О.І.
(підпис)
« ___ » _____2024 р.

ЗАВДАННЯ

на магістерську дипломну роботу студенту
Діденко Владислав Віталійович
(прізвище, ім'я, по батькові)

1.Тема роботи *Система верифікації цифрового двійника літака*

Керівник роботи *д.т.н., проф. Меняйленко Олександр Сергійович,*
(вчене звання, науковий ступінь, прізвище, ім'я, по батькові)

Затверджений наказом університету від «*6*» *12* 2024року №361/15.15-С

2. Строк подання студентом роботи: *14 грудня 2024 р.*

3. Вихідні дані до роботи: *Матеріали науково-дослідної практики, технічна документація; дані інтернет-мережі*

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

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 зазначенням обов'язкових креслень)

Структура систем літака (Структурна схема), Діаграма варіантів користування (Функціональна схема), Алгоритм дій програмного забезпечення (Принципова схема)

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1. Постановка задачі та дослідження	Меняйленко О.С.		
Розділ 2. Проектування додатку	Меняйленко О.С.		
Розділ 3. PID-регулятори	Меняйленко О.С.		
Розділ 4. Деталі розробки системи	Меняйленко О.С.		
Розділ 5. Дослідження та аналіз розробленої системи	Меняйленко О.С.		

7. Дата видачі завдання __8 листопада 2024р._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Одержання завдання на виконання роботи	20.10.2024	
2.	Укладання і погодження з керівником плану і етапів виконання роботи	24.10.2024	
3.	Узагальнення даних літературних джерел	28.10.2024	
4.	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху виконання завдання	01.11.2024	
5.	Аналіз технічних засобів та існуючих систем	07.11.2024	
6.	Реалізація практичної частини завдання	24.11.2024	
7.	Укладання, оформлення та погодження пояснювальної записки з керівником	11.12.2024	
8.	Надання пояснювальної записки на кафедру	14.12.2024	
9.	Підготовка доповіді та презентації	14.12.2024	

Студент _____ __Діденко В.В._
(підпис) (прізвище та ініціали)

Керівник роботи _____ Меняйленко О.С.
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Сучасна авіаційна галузь потребує розробки передових технологій та систем, що забезпечують високі стандарти безпеки, ефективності та надійності під час виконання польотів. Одним із ключових напрямків інновацій є автоматизація процесів управління літаком, зокрема систем автопілотування, що знижує ризики, пов’язані з людським фактором, і гарантує точність виконання заданих маневрів та сценаріїв польоту.

Метою цього дипломного проєкту є створення комплексної системи автопілотування, здатної мінімізувати вплив як людських, так і зовнішніх факторів під час багаторазового виконання польотів. Основним завданням є досягнення повторюваності та високої надійності тестування, що дозволяє отримувати точні результати для перевірки заданих сценаріїв.

Однією з центральних технологій, що використовується у розробці, є PID-регулятори, які довели свою ефективність у системах автоматичного керування. Їх застосування дозволяє адаптувати параметри керування на основі відхилень від заданих значень, забезпечуючи стабільну та точну роботу автопілота.

Основною перевагою розробленої системи є можливість валідації цифрової моделі літака через порівняння її роботи з фактичними даними реальних польотів. Використання тестових сценаріїв дозволяє точно оцінити відповідність цифрового двійника літального апарата його фізичному прототипу, що є критично важливим у процесі розробки та впровадження авіаційних систем.

У подальших розділах роботи будуть детально описані етапи проєктування, моделювання, тестування та впровадження системи автопілотування. Очікувані результати сприятимуть підвищенню безпеки та ефективності польотів, а також розвитку автоматизованих технологій керування повітряним транспортом.

Отже, метою цієї роботи є забезпечення можливості перевірки відповідності цифрового двійника літака його реальній моделі за допомогою системи автоматичного керування та тестувань. Це досягається шляхом порівняння результатів польотів у симуляторі з даними реальних польотів, що, своєю чергою, дозволяє безпечно проводити експерименти для отримання важливих даних, уникаючи ризику пошкодження літака чи загрози для людей.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА ДОСЛІДЖЕННЯ

1.1 Опис задачі

Метою проєкту є розробка системи автопілотування для цифрової моделі літака, яка використовуватиметься для тестування та перевірки її відповідності реальному літаку. Основне завдання полягає у забезпеченні можливості успішного проходження тестів цифровою моделлю під управлінням автопілота, що підтверджує її коректність та точність.

Система автопілотування розроблятиметься з урахуванням специфічних вимог до тестування і верифікації моделі. Кожен тест передбачає наявність окремого автопілота, що керуватиме літаком відповідно до заданих параметрів сценарію. Функціонал системи зосереджений на коректному виконанні тестових завдань і забезпеченні точності результатів.

Основні вимоги до системи включають:

- створення програмного забезпечення для інтерактивного керування цифровою моделлю літака за допомогою автопілота;
- розробку тестових сценаріїв для визначення параметрів руху літака під час тестування;
- впровадження механізмів перевірки результатів тестів з метою підтвердження точності моделі.

Завдяки цій системі вдасться забезпечити контрольовані умови руху цифрової моделі літака під час тестування, що дозволить детально оцінити її відповідність характеристикам реального прототипу.

1.2 Аналоги

Одним із прикладів аналогічних проєктів є програма "Digital Twin Validation" від компанії Siemens, яка використовує цифрові двійники промислових систем для перевірки їхньої точності та ефективності. Головними перевагами цієї програми є широкий набір аналітичних інструментів і алгоритмів, що дозволяють порівнювати дані цифрових моделей із реальними результатами. Це допомагає виявляти невідповідності та відхилення. Проте складність налаштування та калібрування двійників, особливо у випадку з комплексними системами, може бути істотним обмеженням.

Іншим прикладом є проєкт "Digital Twin Verification and Validation" від NASA, який використовує цифрові двійники літаків і космічних апаратів для перевірки їхнього проєктування та роботи в різних умовах. Цей підхід вирізняється застосуванням високоточних симуляторів і моделей, які детально відтворюють умови атмосфери й космосу, забезпечуючи реалістичність тестувань. Недоліком є значна тривалість і високі витрати на реалізацію таких проєктів через складність космічних систем.

Додатково слід зазначити, що низка аналогічних проєктів має закритий доступ і використовується лише у військовій чи комерційній сфері, що ускладнює роботу дослідників, які не мають до них доступу.

На відміну від зазначених проєктів, дана робота зосереджена на розробці системи автопілотування для цифрової моделі літака. Завдяки спеціалізації на конкретному об'єкті, проєкт дозволяє глибше досліджувати особливості та вимоги до таких систем, оптимально використовуючи ресурси для забезпечення точності та стабільності роботи автопілота.

1.3 Вимоги до додатку

1.3.1 Функціональні вимоги

- Реалізація алгоритмів PID-регуляції для автоматичного управління ключовими параметрами польоту, такими як швидкість, висота та кут нахилу.
- Забезпечення стабільного та точного контролю над параметрами польоту шляхом оптимального налаштування коефіцієнтів PID-регуляторів і використання даних із сенсорів.

1.3.2 Нефункціональні вимоги

- Забезпечення високої швидкодії та ефективності додатку для плавного управління і мінімізації затримок у реакції системи.
- Гарантування надійності та стабільності роботи, що передбачає мінімізацію ризиків виникнення помилок і запобігання аварійним ситуаціям.

Ці вимоги спрямовані на створення високоефективної системи автопілотування, яка відповідатиме сучасним стандартам точності, стабільності й надійності.

1.4 Аналіз предметної області

Одним із ключових аспектів досліджуваної області є застосування PID-регуляторів у системах автопілотування. Ці регулятори широко використовуються для автоматичного керування та регулювання завдяки їхній здатності забезпечувати стабільну й точну роботу системи за рахунок комбінованого використання пропорційної, інтегральної та диференціальної

складових. Ретельний аналіз підходів до налаштування параметрів PID-регуляторів є критичним для оптимізації роботи автопілота.

Важливим викликом у розробці таких систем є вплив зовнішніх змінних і людського фактору. Зокрема, на стабільність і точність автопілотування впливають погодні умови, щільність повітряного трафіку та динамічні обмеження. Дослідження існуючих методів і стратегій для врахування цих факторів допомагає розробляти більш стійкі та надійні системи.

Ще одним важливим напрямком аналізу є інтеграція цифрових двійників літаків із їхніми реальними прототипами. Цифрові двійники надають можливість моделювати поведінку літака віртуально, що значно спрощує процес верифікації. Сучасні методи перевірки точності цифрових двійників дозволяють отримувати достовірні результати, підвищуючи якість і ефективність тестувань.

Проведений аналіз предметної області створює основу для подальшого розроблення системи автопілотування, яка забезпечить стабільність, повторюваність тестових процесів і точність у перевірці відповідності цифрового двійника реальному літаку.

1.5 Розробка вимог до характеристик об'єкта проектування

Вимоги до характеристик об’єкта проєктування визначають його функціональні та технічні властивості, необхідні для досягнення цілей проєкту. Основними вимогами є:

- Точність: Система автопілотування повинна забезпечувати високу точність керування літаком, що включає чітке дотримання заданих курсів, висоти та швидкості.
- Стабільність: Автопілот має гарантувати стабільне керування літаком, швидко реагуючи на зміни вхідних параметрів і забезпечуючи плавність польоту без небажаних коливань.

- Відновлюваність: У разі виникнення збоїв система повинна мати можливість автоматичного переключення на резервні компоненти або механізми відновлення, щоб уникнути аварійних ситуацій.
- Надійність: Робота системи має бути безперебійною протягом тривалого часу, а її компоненти повинні бути стійкими до зовнішніх впливів, зокрема погодних умов чи механічних навантажень.
- Сумісність: Система повинна інтегруватися з існуючими бортовими системами літака, забезпечуючи їхню узгоджену взаємодію без конфліктів.
- Практичність: Інтерфейс користувача має бути зручним і зрозумілим, а система — легкою у налаштуванні та обслуговуванні. Також важлива доступність необхідних запасних частин і технічної підтримки.

Ці вимоги слугують основою для проєктування й розробки системи автопілотування, забезпечуючи її відповідність поставленим завданням та ефективність у досягненні цілей проєкту.

1.6 Висновки до розділу

Проведений аналіз предметної області та поставлених завдань свідчить про актуальність і значущість розробки системи автопілотування в контексті верифікації та валідації цифрових моделей літаків. Існуючі системи мають певні обмеження, що зумовлює необхідність створення спеціалізованого комплексу, який би відповідав сучасним вимогам.

Основний акцент у цій роботі зроблено на розробці вузькоспеціалізованого інструменту, спрямованого на досягнення максимальної точності й ефективності. Завдяки зосередженню на конкретному об’єкті, система надає можливість досягти оптимальних результатів у перевірці й тестуванні цифрових моделей.

Запропонований підхід дозволяє підвищити гнучкість і продуктивність систем автопілотування для різних тестових сценаріїв. Особливу цінність становить можливість проведення експериментів із різними конфігураціями та алгоритмами, що сприяє пошуку оптимальних рішень для конкретних завдань.

Результатом цієї роботи стане створення сучасного інструменту для автоматизації авіаційних систем, який відповідатиме вимогам галузі, забезпечуватиме високу точність і дозволятиме адаптацію до різних умов і цілей.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ

2.1 Огляд технологій та інструментів для розробки системи

2.1.1 Вибір мови та двигуна

Для реалізації системи автопілотування розглянуто кілька популярних комбінацій мов програмування та симуляційних платформ, таких як:

- Python у поєднанні з FlightGear
- C++ у поєднанні з Microsoft Flight Simulator
- JavaScript у поєднанні з Microsoft Flight Simulator
- Lua у поєднанні з X-Plane

Найбільш доцільним варіантом визнано використання мови Lua разом із симуляційним двигуном X-Plane, що забезпечує низку переваг:

Простота розробки: Lua має легкий у засвоєнні синтаксис і мінімальну кількість ключових слів, що прискорює процес програмування порівняно зі складнішими мовами, наприклад, C++.

Висока швидкість розробки: Завдяки динамічній типізації та автоматичному керуванню пам’яттю Lua дозволяє скоротити час розробки.

Інтеграція з X-Plane: Мова має вбудовану підтримку API X-Plane, що значно полегшує створення та тестування систем автопілотування, використовуючи реалістичні дані симулятора.

Підтримка спільноти: Lua має активну спільноту розробників, яка забезпечує документацію, приклади коду та готові плагіни, що сприяє швидшому вирішенню технічних питань.

Гнучкість інтеграції: Lua легко взаємодіє з іншими мовами, такими як C++, що дозволяє ефективно комбінувати її з іншими компонентами проєкту.

Таким чином, вибір Lua у поєднанні з X-Plane є оптимальним рішенням для розробки системи автопілотування, яке поєднує простоту, гнучкість та потужність для реалізації поставлених завдань.

2.1.2 Мова Lua

2.1.2.1 Історія та походження мови

Lua, що в перекладі з португальської означає «місяць», була створена у 1993 році в Понтіфікському католицькому університеті Ріо-де-Жанейро. Її розробка стала результатом досліджень у галузі мов програмування, спрямованих на створення ефективного інструменту для розширення функціоналу програмного забезпечення.

2.1.2.2 Основні властивості та характеристики Lua

Lua поєднує потужність сучасної мови програмування з простотою використання. Її основні властивості включають:

- Інтегративність: Lua створена для роботи як вбудована мова програмування, що легко взаємодіє з C/C++ через простий API, не вимагаючи спеціальних знань.
- Підтримка кількох парадигм: Lua підтримує процедурне, об’єктно-орієнтоване та функціональне програмування, що забезпечує гнучкість у розробці.
- Метапрограмування: Потужні засоби, такі як метатаблиці та метаметоди, дозволяють змінювати поведінку об’єктів і перевизначати оператори.
- Продуктивність: Lua працює швидко завдяки оптимізації, а використання LuaJIT (Just-In-Time компілятора) дозволяє досягти продуктивності, порівнянної з кодом на C.

- Відкритий код: Lua має відкриту ліцензію, що дозволяє її безкоштовне використання, зміну та розповсюдження.
- Портативність: Мова сумісна з багатьма платформами та операційними системами без необхідності адаптації коду.
- Малий розмір: Компактне ядро Lua підходить для вбудованих систем.
- Гнучкість: Lua підтримує динамічне виконання коду, замикання, анонімні функції, що полегшує створення складних структур.
- Управління пам’яттю: Мова використовує автоматичний збір сміття, що зменшує кількість помилок і підвищує стабільність.
- Підтримка корутин: Lua дозволяє реалізовувати асинхронні операції та багатопоточність у межах одного процесу.

Завдяки цим особливостям, Lua є відмінним вибором для створення складних програм, і ідеально підходить для нашого проекту.

2.1.2.3 Lua в контексті автопілотування

Мова Lua є ідеальним вибором для розробки систем автопілотування завдяки своїй гнучкості, простоті та ефективності. Її скриптовий характер дозволяє легко створювати модульні рішення та швидко впроваджувати нові алгоритми управління без необхідності компіляції коду, що значно спрощує експериментування та розробку.

Крім того, Lua забезпечує ефективну інтеграцію з іншими мовами програмування та системами, що дає змогу використовувати її у складі вже існуючих платформ чи доповнювати поточний функціонал. Завдяки низьким вимогам до ресурсів і високій швидкодії, Lua добре підходить для реалізації логіки автопілота, яка потребує оперативної обробки даних і швидкого прийняття рішень.

Таким чином, Lua виступає як потужний інструмент для створення й адаптації систем автопілотування, забезпечуючи баланс між продуктивністю, простотою використання та можливістю розширення функціоналу.

2.1.2.4 Lua і X-Plane

Мова Lua чудово поєднується із симулятором X-Plane завдяки можливості інтеграції через плагін SASL. Цей інструмент дозволяє створювати скрипти на Lua, які забезпечують автоматизацію різних аспектів симуляції польотів, включаючи управління літаком, використання навігаційних систем і моделювання дій пілота.

У контексті розробки систем автопілотування Lua використовується для програмування логіки управління, тоді як X-Plane забезпечує реалістичні умови для тестування цієї логіки. Зокрема, Lua дозволяє отримувати дані про стан літака, змінювати параметри управління та створювати складні сценарії тестувань.

Поєднання Lua з X-Plane забезпечує потужний інструментарій для моделювання та тестування автопілотів у реалістичному середовищі, що значно полегшує розробку, перевірку та вдосконалення таких систем.

2.1.2 Двигун X-Plane

2.1.2.1 Загальний огляд X-Plane

X-Plane є одним із провідних симуляторів польотів, який широко використовується в авіаційній індустрії та навчанні завдяки своїй високій точності моделювання та широкому функціоналу. Основні переваги X-Plane включають:

- **Реалістичну модель польоту**: Використання фізичних і аеродинамічних принципів забезпечує точне відтворення поведінки літаків у різних умовах польоту.
- **Широкий вибір літальних апаратів**: Симулятор підтримує безліч моделей літаків, вертольотів і планерів, дозволяючи проводити різноманітні тренування та випробування.
- **Детальну графіку**: Високоякісна візуалізація ландшафту, аеропортів і погодних умов створює реалістичне середовище польоту.
- **Можливості розширення**: Завдяки підтримці плагінів і кастомних сценаріїв X-Plane легко адаптується до потреб розробників.
- **Практичне застосування**: Використовується для навчання пілотів, випробування нових технологій і дослідження характеристик польоту.

2.1.2.2 Архітектура X-Plane

X-Plane має модульну архітектуру, яка забезпечує взаємодію між кількома ключовими компонентами:

- **Графічний модуль**: Відповідає за візуалізацію польотного середовища, включаючи ландшафт, об’єкти, погодні ефекти та аеропорти.
- **Фізичний модуль**: Моделює аеродинаміку, динаміку польоту та взаємодію з зовнішнім середовищем, відтворюючи поведінку літака з урахуванням його конструкції й умов польоту.
- **Модуль керування**: Здійснює обчислення та управління параметрами польоту, такими як курс, висота, швидкість тощо.
- **Модуль моделі літака**: Реалізує цифрову модель літального апарата, яка враховує його конструктивні особливості та системи.

Ці модулі працюють у тісній взаємодії, забезпечуючи реалістичну симуляцію польотів та інтеграцію зовнішніх компонентів.

2.1.2.3 Моделювання польотів в X-Plane

X-Plane дозволяє створювати реалістичні умови польоту завдяки:

- Динаміці польоту: Відтворення фізичних характеристик руху літака, таких як керованість і реакція на управління.
- Погодним умовам: Моделювання різних погодних сценаріїв, включаючи турбулентність, вітер і опади, що впливають на поведінку літака.
- Навігації: Підтримка реальних навігаційних систем для відпрацювання польотних процедур.
- Деталізованій місцевості: Відображення аеропортів, міських і природних об’єктів, що сприяє точному виконанню польотних завдань.

Завдяки цим можливостям X-Plane виступає потужним інструментом для моделювання польотів, навчання, дослідження та тестування систем автопілотування.

2.1.3 Шаблони програмування

2.1.3.1 Адаптер

Шаблон проєктування "Адаптер" належить до структурних шаблонів і використовується для забезпечення сумісності між несумісними інтерфейсами. Він дозволяє поєднувати компоненти з різними інтерфейсами в єдину систему, створюючи проміжний шар, який перетворює один інтерфейс в інший.

Основні цілі використання шаблону "Адаптер":

- **Сумісність інтерфейсів**: Шаблон забезпечує взаємодію між компонентами, що мають різні інтерфейси, наприклад, через зміну назв методів або форматів параметрів.
- **Інтеграція старого та нового коду**: "Адаптер" дозволяє використовувати наявний код із новими компонентами, уникаючи значних змін у вихідному коді.
- **Повторне використання компонентів**: Шаблон дає змогу адаптувати існуючі компоненти до нових контекстів, роблячи їх універсальнішими.
- **Робота з сторонніми бібліотеками**: Створення адаптерного шару дозволяє використовувати сторонні API або інші зовнішні системи в поточному проєкті.

Види шаблону "Адаптер":

- **Класовий адаптер**: Реалізується за допомогою успадкування, де адаптер одночасно успадковує інтерфейси як клієнта, так і адаптованого класу. Такий підхід дозволяє використовувати методи обох інтерфейсів, однак обмежується мовами, що підтримують множинне успадкування.
- **Об'єктний адаптер**: Використовує композицію, коли адаптер містить об'єкт адаптованого класу. Такий підхід є гнучкішим, оскільки дозволяє змінювати адаптований клас без зміни клієнтського коду.

Етапи реалізації шаблону:

- Визначення інтерфейсу клієнта, що описує методи, необхідні для роботи.
- Створення адаптованого класу, який реалізує існуючий інтерфейс із несумісними методами.
- Реалізація класу адаптера, що перетворює методи адаптованого класу на ті, що підтримує клієнт.
- Впровадження адаптера у систему через клієнтський код.

- Переваги використання "Адаптера":
- Забезпечує гнучкість і розширюваність системи.
- Мінімізує зміни в наявному коді, підвищуючи його повторне використання.
- Полегшує інтеграцію зовнішніх бібліотек та систем.
- Недоліки:
- Ускладнює структуру програми через додавання проміжного шару.
- Може створювати додаткову накладну витрату ресурсів.

Шаблон "Адаптер" є універсальним інструментом, що спрощує інтеграцію компонентів із різними інтерфейсами, підвищуючи гнучкість та масштабованість програмної системи.

2.1.3.2 Стратегія

Шаблон "Стратегія" належить до поведінкових шаблонів проєктування і використовується для динамічного вибору одного з кількох алгоритмів залежно від умов. Він дозволяє інкапсулювати різні варіанти поведінки у вигляді окремих класів, що реалізують спільний інтерфейс, і замінювати їх без модифікації коду, який використовує ці алгоритми.

Основна ідея

Шаблон "Стратегія" дозволяє розділити реалізацію алгоритму та контекст його використання, забезпечуючи гнучкість у зміні або розширенні функціональності системи. Контекст використовує загальний інтерфейс для взаємодії зі стратегіями, залишаючись незалежним від конкретних реалізацій.

Складові шаблону

Контекст (Context): Клас, який взаємодіє з об'єктом стратегії через загальний інтерфейс. Він не знає про деталі реалізації конкретної стратегії.

Інтерфейс стратегії (Strategy): Спільний інтерфейс або абстрактний клас, який визначає методи, що реалізуються конкретними стратегіями.

Конкретні стратегії (Concrete Strategies): Реалізації інтерфейсу стратегії, які містять логіку конкретних алгоритмів або поведінок.

Етапи реалізації

Визначення спільного інтерфейсу для стратегій.

Створення класів, які реалізують цей інтерфейс і визначають конкретні алгоритми.

Реалізація класу контексту, який використовує об'єкт стратегії через спільний інтерфейс.

Додавання можливості змінювати стратегії під час виконання програми (зазвичай через конструктор або сеттер).

Переваги

Гнучкість: Легко додавати нові стратегії без змін у коді контексту.

Розширюваність: Шаблон сприяє дотриманню принципу відкритості/закритості (Open/Closed Principle).

Зниження залежностей: Контекст не залежить від конкретних реалізацій стратегій, що полегшує підтримку коду.

Зменшення дублювання: Логіка різних алгоритмів інкапсулюється в окремих класах, уникаючи дублювання коду.

Недоліки

Збільшення кількості класів у проєкті.

Ускладнення системи через додатковий рівень абстракції.

Приклад використання

У системах автопілотування шаблон "Стратегія" можна використовувати для реалізації різних методів управління літаком (наприклад, підтримка висоти, курсу або швидкості). Залежно від умов польоту, контекст вибирає відповідну стратегію, забезпечуючи оптимальне керування.

Шаблон "Стратегія" забезпечує динамічний вибір алгоритмів, підвищуючи гнучкість і модульність системи, що особливо важливо для складних програмних продуктів із різноманітною поведінкою.

2.2 Висновки до розділу

Розглянуті в цьому розділі технології та інструменти демонструють обґрунтованість обраного підходу до розробки системи автопілотування. Вибір мови програмування Lua, яка забезпечує простоту й гнучкість, у поєднанні з симуляційним двигуном X-Plane, що відзначається високою точністю моделювання польотів, дозволяє створити ефективну і реалістичну платформу для тестувань.

Крім того, використання шаблонів проєктування, таких як "Адаптер" і "Стратегія", сприяє побудові структурованого та легко модифікованого коду, що адаптується до змінних вимог системи автопілотування.

Усі обрані рішення формують міцну основу для подальшого етапу розробки, зокрема проєктування, впровадження й тестування додатку, що відповідає визначеним вимогам та цілям проєкту.

РОЗДІЛ 3. PID-РЕГУЛЯТОРИ

3.1 Принцип роботи PID-регулятора

3.1.1 Пропорційна складова

Пропорційна складова (P) є ключовим елементом PID-регулятора, що забезпечує базову реакцію системи на помилку між поточним і бажаним значенням керованого параметра. Її робота ґрунтується на тому, що вихідний сигнал регулятора пропорційний до величини цієї помилки: чим більше відхилення, тим більший вплив пропорційної складової на управління.

Основним параметром, який визначає ефективність пропорційної складової, є коефіцієнт пропорційності K_p. Величина цього коефіцієнта визначає, наскільки чутливо система реагує на помилку:

- **Занадто високий K_p** може викликати надмірну реакцію системи, що призводить до перевищення заданого значення (overshoot) і нестабільності.
- **Занадто низький K_p** може спричинити недостатню реакцію, що уповільнює процес регулювання.

Хоча пропорційна складова забезпечує швидку реакцію системи на зміну помилки, вона не здатна повністю усунути її. Як наслідок, навіть у стабільному стані може залишатися незначна статична помилка, яку необхідно коригувати іншими складовими PID-регулятора.

Для оптимального налаштування пропорційної складової необхідно враховувати характеристики системи, її динаміку та вимоги до стабільності й точності. Правильно підібраний коефіцієнт K_p дозволяє досягти необхідного балансу між швидкістю реакції та стійкістю системи.

Таким чином, пропорційна складова є фундаментальною частиною PID-регулятора, яка дозволяє системі швидко реагувати на зміни помилки, забезпечуючи базовий рівень управління.

3.1.2 Інтегральна складова

Інтегральна складова (I) є невід'ємною частиною PID-регулятора, яка відповідає за компенсацію статичної помилки, що може залишатися після дії пропорційної складової. Її функція полягає у врахуванні сукупного впливу помилки за певний проміжок часу, що дозволяє системі поступово усувати відхилення від бажаного значення.

Робота інтегральної складової базується на обчисленні інтегралу помилки, що накопичується з часом. Її вплив збільшується, якщо помилка залишається тривалий час, забезпечуючи постійне коригування вихідного сигналу регулятора.

Важливим параметром, який визначає ефективність інтегральної складової, є коефіцієнт інтеграції K_i:

- **Занадто високий K_i** може спричинити нестабільність системи, оскільки регулятор реагуватиме надто різко, що може викликати коливання.
- **Занадто низький K_i** може уповільнити усунення статичної помилки, залишаючи систему у стані з незначним, але постійним відхиленням.

Інтегральна складова має особливе значення для досягнення точності системи, оскільки вона забезпечує усунення навіть малих помилок. Однак її налаштування потребує врахування характеристик системи, включно з її динамікою та чутливістю до зовнішніх впливів.

При належному налаштуванні інтегральна складова ефективно компенсує статичні помилки й сприяє досягненню заданих параметрів системи навіть у складних умовах.

3.1.3 Диференціальна складова

Диференціальна складова (D) є важливим компонентом PID-регулятора, що забезпечує поліпшення стабільності й швидкості реакції системи. Вона прогнозує майбутню зміну помилки, аналізуючи її швидкість зміни, і використовує цю інформацію для коригування вихідного сигналу регулятора.

Принцип роботи диференціальної складової полягає в обчисленні похідної помилки по часу. Якщо зміни помилки відбуваються швидко, диференціальна складова генерує пропорційно більший сигнал, що допомагає зменшити коливання та уникнути різких реакцій системи на зовнішні збурення.

Коефіцієнт диференціальної дії K_d визначає, наскільки сильно ця складова впливає на вихідний сигнал:

- **Занадто велике значення K_d** може призвести до підвищеної чутливості до шумів і викликати небажані коливання.
- **Занадто мале значення K_d** знижує ефективність прогнозування, залишаючи систему вразливою до різких змін.

Диференціальна складова найбільш корисна в системах, де важливим є швидке реагування на динамічні зміни. Вона допомагає зменшити затримки, покращує плавність роботи системи та мінімізує ризик переналаштувань.

Правильне налаштування K_d дозволяє досягти оптимального балансу між стабільністю й швидкістю реакції, що є важливим для забезпечення точного управління.

У результаті диференціальна складова виконує роль "амортизатора", допомагаючи системі реагувати на зміни швидко, але плавно, без зайвих коливань.

3.1.4 Вагові коефіцієнти

Вагові коефіцієнти у PID-регуляторі визначають рівень впливу кожної складової – пропорційної, інтегральної та диференціальної – на вихідний сигнал системи. Вони дозволяють адаптувати роботу регулятора до конкретних умов, забезпечуючи баланс між стабільністю, точністю й швидкістю реакції.

Кожна складова регулятора має свій ваговий коефіцієнт:

- **Пропорційна складова** має коефіцієнт K_p, який регулює рівень миттєвої реакції на поточну помилку.
- **Інтегральна складова** використовує коефіцієнт K_i, що визначає швидкість накопичення впливу помилки за час.
- **Диференціальна складова** має коефіцієнт K_d, який відповідає за інтенсивність реакції на зміну помилки.

Вагові коефіцієнти можуть налаштовуватися вручну або автоматично, наприклад, за допомогою методів оптимізації, таких як метод Зіглера-Ніколсона. Їх налаштування потребує врахування характеристик системи, включаючи її динаміку, стійкість і вимоги до точності.

Залежно від завдання:

- **Підвищення K_p** дозволяє швидше реагувати на помилку, але може спричинити коливання.
- **Збільшення K_i** сприяє усуненню статичної помилки, проте може зменшити стабільність.
- **Зростання K_d** допомагає зменшити вплив різких змін, але збільшує чутливість до шумів.

Оптимально налаштовані вагові коефіцієнти дозволяють досягти гармонійної роботи PID-регулятора, що забезпечує стабільне й точне управління системою в різних умовах.

3.2 Розрахунок вагових коефіцієнтів

Для налаштування вагових коефіцієнтів пропорційної, інтегральної та диференціальної складових PID-регулятора застосовуються різні методи та формули, що дозволяють оптимізувати роботу системи. Нижче наведено деякі із найпоширеніших підходів:

3.2.1 Метод Зіглера-Ніколсона

Метод Зіглера-Ніколсона є одним із класичних підходів до налаштування PID-регуляторів, який базується на експериментальному визначенні параметрів системи. Його основна ідея полягає у встановленні таких значень коефіцієнтів регулятора, які забезпечують стійке керування із заданими характеристиками.

Процедура налаштування:

1. Увімкніть лише пропорційну складову регулятора, встановивши коефіцієнти інтегральної та диференціальної дії (K_i і K_d) рівними нулю.
2. Поступово збільшуйте коефіцієнт пропорційності (K_p), поки система не почне коливатися з постійною амплітудою.
3. Запишіть отримане значення K_p (граничний коефіцієнт K_u) та період коливань системи (T_u).
4. Використовуйте отримані значення для розрахунку параметрів PID-регулятора за формулами:
   - $K_p=0.6\cdot K_u$
   - $K_i=2\cdot K_p/T_u$
   - $K_d=K_p\cdot T_u/8$

Цей метод дозволяє отримати початкові значення коефіцієнтів регулятора, які забезпечують ефективне керування системою. Однак, для

складних або нестабільних систем може знадобитися подальше уточнення параметрів.

Метод Зіглера-Ніколсона є швидким і простим у застосуванні, що робить його популярним для базового налаштування PID-регуляторів у багатьох галузях.

3.2.2 Метод оптимальної реакції на ступінь

Метод оптимальної реакції на ступінь є підходом до налаштування вагових коефіцієнтів PID-регулятора, який ґрунтується на аналізі реакції системи на стрибкоподібну зміну вхідного сигналу. Цей метод дозволяє визначити оптимальні параметри регулятора, що забезпечують швидке та точне досягнення заданого стану системи.

Процедура налаштування:

1. **Увімкніть лише пропорційну складову регулятора**: Встановіть коефіцієнти K_i (інтегральний) та K_d (диференціальний) рівними нулю.
2. **Подайте на вхід системи стрибкоподібний сигнал**: Це може бути різка зміна керуючого впливу або вхідного параметра.
3. **Спостерігайте вихідну реакцію системи**: Виміряйте час перехідного процесу (T_r), який визначається часом досягнення 90–95% усталеного значення.
4. **Обчисліть вагові коефіцієнти за формулами**:
   - $K_p=1/(K_u \cdot T_r)$
   - $K_i=K_p/(2 \cdot T_u)$
   - $K_d=K_p \cdot T_u/8$,

     де K_u— коефіцієнт чутливості системи, а T — період коливань.

Переваги методу:

- Простота застосування: Метод не потребує складних математичних моделей системи.

- Швидкість налаштування: Дає змогу отримати параметри за мінімальну кількість тестів.
- Доступність: Може використовуватися навіть без спеціального обладнання для аналізу динаміки системи.

Обмеження:

- Менш ефективний для систем із високою складністю або нелінійною динамікою.
- Може знадобитися додаткове уточнення коефіцієнтів для досягнення стабільності в реальних умовах.
- Метод оптимальної реакції на ступінь забезпечує базове налаштування PID-регулятора, дозволяючи досягти балансу між швидкістю реакції, точністю та стабільністю системи.

3.2.3 Метод Заде (Ziegler-Nichols)

Метод Заде (або Зіглера-Ніколсона) є одним із найпопулярніших підходів до налаштування PID-регуляторів. Цей метод базується на аналізі поведінки системи при граничних значеннях коефіцієнта пропорційності та дозволяє визначити оптимальні параметри регулятора для забезпечення стабільної роботи.

Процедура налаштування:

1. **Увімкніть лише пропорційну складову регулятора**: Встановіть інтегральний (K_i) та диференціальний (K_d) коефіцієнти рівними нулю.
2. **Поступово збільшуйте коефіцієнт пропорційності (K_p)**: Робіть це до досягнення стану, коли система почне демонструвати стійкі коливання з постійною амплітудою.
3. **Визначте ключові параметри**: Зафіксуйте значення граничного коефіцієнта пропорційності (K_u) та період коливань (T_u).
4. **Обчисліть параметри PID-регулятора** за наступними формулами:

- Для P-регулятора:
  - $K_p=0.5\cdot K_u$
- Для PI-регулятора:
  - $K_p=0.45\cdot K_u$
  - $K_i=1.2\cdot K_p/T_u$
- Для PID-регулятора:
  - $K_p=0.6\cdot K_u$
  - $K_i=2\cdot K_p/T_u$
  - $K_d=0.125\cdot K_p\cdot T_u$

Переваги методу:

- Простота і швидкість у застосуванні.
- Ефективне початкове налаштування для більшості систем із лінійною динамікою.
- Підходить для систем, які демонструють коливальну поведінку.

Недоліки:

- Може бути недостатньо точним для систем із сильно нелінійною або складною динамікою.
- Налаштування потребує перебування системи на межі стійкості, що не завжди безпечно або можливо.

Метод Заде є ефективним інструментом для базового налаштування PID-регулятора, забезпечуючи баланс між простотою застосування й точністю отриманих параметрів. Однак, у складних випадках можуть знадобитися додаткові коригування або більш точні методи налаштування.

3.2.4 Метод Cohen-Coon

Метод Cohen-Coon є одним із класичних підходів до налаштування PID-регуляторів, який базується на аналізі реакції системи на стрибкоподібний вплив. Цей метод широко використовується для систем із лінійною динамікою

та надає можливість отримати оптимальні параметри регулятора для забезпечення точного і стабільного управління.

Процедура налаштування:

1. **Подайте на вхід системи ступінчастий сигнал**: Це може бути різка зміна вхідного параметра або керуючого впливу.
2. **Запишіть реакцію системи**: Зафіксуйте часові характеристики, зокрема:
   - T_d— затримка системи (час до початку реакції).
   - T — час, за який система досягає усталеного стану.
   - K — коефіцієнт підсилення системи.
3. **Обчисліть вагові коефіцієнти регулятора** за формулами:

- Для P-регулятора:
  - $K_p=1/K$,
- Для PI-регулятора:
  - $K_p=0.9/K$,
  - $K_i=3/(T+T_d)$,
- Для PID-регулятора:
  - $K_p=1.35/K$,
  - $K_i=2.5/(T+T_d)$,
  - $K_d=0.37 \cdot T_d$.

Переваги методу:

- Забезпечує швидке й ефективне налаштування для систем зі стабільною динамікою.
- Може використовуватися без детального моделювання системи.
- Надає збалансовані параметри, що підходять для більшості випадків.

Недоліки:

- Не підходить для складних або сильно нелінійних систем.
- Параметри можуть потребувати додаткового уточнення у випадках нестабільної роботи.

Метод Cohen-Coon є універсальним інструментом для налаштування PID-регуляторів, особливо для систем зі стабільною та передбачуваною динамікою. Він дозволяє швидко отримати параметри, які забезпечують оптимальний баланс між швидкістю реакції, точністю та стійкістю.

3.2.5 Метод налаштування за допомогою критерію МакКаула-Свідлера

Метод МакКаула-Свідлера є підходом до налаштування PID-регулятора, який використовує спеціальний критерій оптимізації для досягнення максимальної ефективності управління. Основна ідея полягає у виборі параметрів регулятора, які мінімізують інтегральну квадратну помилку (Integral of Squared Error, ISE), забезпечуючи оптимальний баланс між швидкістю реакції, точністю та стабільністю.

Критерій МакКаула-Свідлера

Критерій оптимізації виражається інтегральною функцією:

$$ISE = \int_0^\infty e^2(t)dt$$

де e(t) — похибка системи в кожний момент часу. Завдання полягає в мінімізації цієї функції шляхом налаштування параметрів PID-регулятора (K_p, K_i, K_d).

Процедура налаштування:

1. **Виберіть початкові значення параметрів**: Встановіть початкові значення коефіцієнтів K_p, K_i, K_d відповідно до приблизних оцінок або інших методів.
2. **Оцініть реакцію системи**: Виміряйте похибку e(t) у процесі роботи системи.
3. **Обчисліть ISE**: Використовуйте формулу для обчислення інтегральної квадратної помилки.

4. **Коригуйте параметри**: Змінюйте значення K_p, K_i, K_d, щоб зменшити ISE. Цей процес може виконуватися вручну або автоматично за допомогою алгоритмів оптимізації.
5. **Повторіть процес**: Продовжуйте коригування, поки значення ISE не стане мінімальним.

Переваги методу:

- Забезпечує високий рівень точності та оптимальну стабільність системи.
- Гарантує ефективну реакцію на зміну вхідних умов.
- Може використовуватися для різних типів систем, включаючи складні та нелінійні.

Недоліки:

- Висока обчислювальна складність, особливо для систем із довгими перехідними процесами.
- Потребує детальної оцінки динаміки системи та похибок.

Метод МакКаула-Свідлера є потужним інструментом для тонкого налаштування PID-регуляторів. Він дозволяє мінімізувати помилки управління та досягти високої якості роботи навіть у складних умовах.

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

Метод налаштування за допомогою рівняння Журайда, також відомий як удосконалення класичного підходу Зіглера-Ніколсона, є інструментом для визначення оптимальних параметрів PID-регулятора. Цей метод враховує особливості перехідного процесу системи, такі як затримка та постійна часу, що дозволяє отримати точніші результати для налаштування.

Основна ідея

Метод використовує модифіковані формули для розрахунку вагових коефіцієнтів (K_p, K_i, K_d), ґрунтуючись на експериментально визначених

характеристиках системи. Він спрямований на забезпечення стабільності, швидкої реакції та мінімізації коливань.

Процедура налаштування:

1. **Експериментальний етап**:

Проведіть тест системи, подавши на вхід стрибкоподібний сигнал.

Визначте три ключові параметри:

K — коефіцієнт підсилення системи.

T_d — затримка системи (час до початку реакції).

T — постійна часу системи (характерний час досягнення усталеного значення).

2. **Обчисліть параметри PID-регулятора**:

Використовуйте рівняння Журайда для розрахунку коефіцієнтів:

- Для P-регулятора:
  - $K_p=1/K \cdot T/T_d$
- Для PI-регулятора:
  - $K_p=0.9/K \cdot T/T_d$,
  - $K_i=K_p/(3 \cdot T_d)$
- Для PID-регулятора:
  - $K_p=1.2/K \cdot T/T_d$,
  - $K_i=2 \cdot K_p/T$,
  - $K_d=0.5 \cdot T_d \cdot K_p$

3. **Перевірте налаштування**:

- Впровадьте розраховані параметри у систему.
- Оцініть її поведінку, зокрема стабільність, швидкість перехідного процесу та точність.
- При необхідності виконайте незначні коригування.

Переваги методу:

- Забезпечує точні результати для систем із передбачуваною динамікою.

- Простий у застосуванні, базується на базових експериментальних даних.
- Враховує специфічні характеристики перехідного процесу, що підвищує стабільність і точність.

Недоліки:

- Не підходить для сильно нелінійних систем.
- Для складних систем може знадобитися додаткове коригування отриманих параметрів.

Метод налаштування за допомогою рівняння Журайда є ефективним інструментом для швидкого та точного визначення параметрів PID-регулятора, забезпечуючи стабільність та оптимальну продуктивність системи.

3.3 Структура PID-регулятора

PID-регулятор (Proportional-Integral-Derivative) є базовим елементом у системах автоматичного керування, який дозволяє забезпечити точне регулювання параметрів системи за рахунок поєднання пропорційної, інтегральної та диференціальної складових.

Основні компоненти PID-регулятора

1. **Пропорційна складова (P)**
   - Реагує на поточну похибку між заданим і фактичним значенням параметра.
   - Забезпечує основний вплив на управління, пропорційний до величини помилки.
   - Визначається коефіцієнтом K_p, який регулює ступінь чутливості до похибки.
2. **Інтегральна складова (I)**
   - Накопичує вплив помилки з часом, що дозволяє усунути статичну похибку.

- Реагує на тривалі помилки, які не може компенсувати пропорційна складова.
- Керується коефіцієнтом K_i, який регулює швидкість накопичення впливу.

3. **Диференціальна складова (D)**
   - Реагує на швидкість зміни помилки, передбачаючи її майбутню поведінку.
   - Допомагає зменшити коливання та покращує стабільність системи.
   - Регулюється коефіцієнтом K_d, який визначає силу впливу на вихідний сигнал.

Математичне представлення

Загальний вихідний сигнал PID-регулятора можна описати рівнянням:

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

де:

- u(t) — керуючий сигнал,
- e(t) — поточна похибка,
- K_p, K_i, K_d — коефіцієнти відповідних складових.

Схема PID-регулятора

PID-регулятор зазвичай представлений у вигляді схеми, яка ілюструє взаємодію його компонентів:

1. **Вхідний сигнал**: Задається бажане значення параметра.
2. **Обчислення похибки**: Різниця між заданим і поточним значенням.
3. **Обробка сигналу**:
   - Пропорційна складова обчислює миттєву реакцію.
   - Інтегральна складова усуває тривалі похибки.
   - Диференціальна складова прогнозує зміни.
4. **Суматор**: Об’єднує результати трьох складових.
5. **Вихідний сигнал**: Передається на виконавчий механізм.

Переваги PID-регулятора

- Забезпечує баланс між швидкістю, точністю та стабільністю системи.
- Легко налаштовується для різних типів об'єктів управління.
- Ефективний у лінійних системах і системах із невеликою нелінійністю.

Обмеження PID-регулятора

- Може бути неефективним у сильно нелінійних системах або системах зі значними затримками.
- Вимагає ретельного налаштування коефіцієнтів для уникнення нестабільності чи коливань.

PID-регулятор є універсальним засобом керування, що знаходить застосування в різних галузях, від промисловості до авіації, забезпечуючи високий рівень точності та надійності управління.

3.4 Різновиди PID-регулятора

Різновиди PID-регуляторів, зокрема пропорційно-інтегруючий (ПІ), пропорційно-диференціюючий (ПД), пропорційний (П) та інші, відрізняються комбінацією та співвідношенням складових у своїй структурі. Їх застосовують для управління системами з різними динамічними характеристиками та реакціями.

3.4.1 Пропорціональний (П) регулятор

Пропорційний регулятор є найпростішим різновидом регуляторів, у якому вихідний сигнал пропорційно залежить від поточної похибки системи. Його робота описується рівнянням:

$$u(t) = K_p \cdot e(t)$$

де:

- u(t) — вихідний сигнал регулятора,
- K_p — коефіцієнт пропорційності,
- e(t) — потчна похибка між заданим і фактичним значенням.

Основна функція П-регулятора полягає у забезпеченні швидкої реакції на відхилення системи від заданого стану. Чим більше похибка, тим сильніше впливає регулятор, пропорційно до коефіцієнта K_p.

Переваги:

- Простота реалізації та налаштування.
- Забезпечення швидкої реакції на зміни похибки.

Недоліки:

- Не може повністю усунути статичну похибку.
- При надто великому K_p може викликати коливання системи.

Пропорційний регулятор зазвичай використовується в системах, де невеликі статичні помилки є прийнятними або де достатньо базового рівня управління для досягнення цілей.

3.4.2 Пропорціонально-інтегруючий (П-І) регулятор

Пропорційно-інтегруючий регулятор поєднує у своїй структурі дві складові: пропорційну та інтегральну. Це дозволяє не лише забезпечити миттєву реакцію на похибку (через пропорційну складову), але й поступово усувати статичні помилки (завдяки інтегральній складовій).

Робота ПІ-регулятора описується рівнянням:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau$$

де:

- u(t) — вихідний сигнал регулятора,
- e(t) — поточна похибка,
- K_p — коефіцієнт пропорційної складової,

- K_i — коефіцієнт інтегральної складової.

Особливості ПІ-регулятора:

- **Пропорційна складова** забезпечує оперативну реакцію на зміни похибки, пропорційно до її величини.
- **Інтегральна складова** накопичує вплив помилки за час, поступово усуваючи статичну похибку, яка залишається після дії пропорційної складової.

Переваги:

- Забезпечує усунення статичної похибки.
- Підходить для систем із повільною динамікою, де потрібна стабільність.

Недоліки:

- Може викликати нестабільність або перерегулювання за умов неправильного налаштування.
- Реакція інтегральної складової є повільнішою, ніж пропорційної.

ПІ-регулятор часто застосовується в системах, де важливим є точне досягнення заданого значення без залишкової похибки, наприклад, у регулюванні температури, рівня рідини або швидкості.

3.4.3 Пропорціонально-диференціюючий (П-Д) регулятор

Пропорційно-диференціюючий регулятор поєднує пропорційну та диференціальну складові. Його головна перевага полягає у здатності передбачати зміну похибки, що дозволяє зменшити коливання та підвищити стабільність системи.

Робота ПД-регулятора описується рівнянням:

$$u(t) = K_p \cdot e(t) + K_i \cdot \frac{de(t)}{dt}$$

де:

u(t) — вихідний сигнал регулятора,

e(t) — поточна похибка,

K_p — коефіцієнт пропорційної складової,

K_d — коефіцієнт диференціальної складової,

Особливості ПД-регулятора:

- **Пропорційна складова** забезпечує миттєву реакцію на поточну похибку, пропорційну її величині.
- **Диференціальна складова** реагує на швидкість зміни похибки, дозволяючи зменшити коливання та стабілізувати систему.

Переваги:

- Підвищує стабільність системи завдяки врахуванню динаміки зміни похибки.
- Зменшує перерегулювання й коливання системи.
- Забезпечує швидшу реакцію на зовнішні збурення.

Недоліки:

- Не усуває статичну похибку, оскільки відсутня інтегральна складова.
- Чутливий до шумів у вхідному сигналі, оскільки похідна підсилює вплив високочастотних складових.

Застосування:

ПД-регулятор найчастіше використовується в системах, де важлива швидкість реакції й стабільність, але невелика залишкова похибка є прийнятною. Наприклад, він ефективний у керуванні системами стабілізації, такими як балансування або підтримка рівноваги.

3.4.4 Пропорціонально-інтегруючо-диференціючий (ПІД) регулятор

Пропорційно-інтегруючо-диференціюючий (ПІД) регулятор є найбільш універсальним і потужним типом регуляторів, що поєднує пропорційну, інтегральну та диференціальну складові. Це дозволяє забезпечити баланс між швидкістю реакції, точністю й стабільністю управління.

Робота ПІД-регулятора описується рівнянням:

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

де:

- u(t) — вихідний сигнал регулятора,
- e(t) — поточна похибка,
- K_p — коефіцієнт пропорційної складової,
- K_i — коефіцієнт інтегральної складової,
- K_d — коефіцієнт диференціальної складової.

Особливості ПІД-регулятора:

- **Пропорційна складова** забезпечує миттєву реакцію на поточну похибку.
- **Інтегральна складова** усуває залишкові статичні похибки за рахунок накопичення впливу помилки з часом.
- **Диференціальна складова** реагує на швидкість зміни похибки, стабілізуючи систему та зменшуючи коливання.

Переваги:

- Забезпечує точне регулювання з усуненням статичних помилок.
- Підвищує стабільність системи, мінімізуючи перерегулювання.
- Ефективний у системах із динамічними властивостями, які змінюються з часом.

Недоліки:

- Потребує ретельного налаштування коефіцієнтів (K_p, K_i, K_d) для досягнення оптимальної роботи.
- Чутливий до шумів, особливо в частині диференціальної складової.
- Висока складність налаштування порівняно з іншими типами регуляторів.

Застосування:

ПІД-регулятор широко використовується в автоматизованих системах управління різних галузей, таких як:

- керування температурою,
- регулювання швидкості двигунів,
- підтримка тиску та рівня рідини,
- управління польотом у авіації.

ПІД-регулятор є стандартом у сфері автоматизації завдяки своїй універсальності та здатності забезпечувати високий рівень точності й стабільності у широкому спектрі застосувань.

3.5 Використання PID-регулятора в системі автопілота

PID-регулятори широко застосовуються в системах автопілота завдяки їхній здатності забезпечувати точне та стабільне управління літальними апаратами. Використовуючи комбінацію пропорційної, інтегральної та диференціальної складових, вони дозволяють досягти оптимального балансу між швидкістю реакції, точністю та стабільністю під час польоту.

Основні функції PID-регулятора в автопілоті

1. **Управління висотою:** PID-регулятор забезпечує підтримку заданої висоти, компенсуючи вплив зовнішніх факторів, таких як вітер або зміна маси літака. Пропорційна складова реагує на відхилення, інтегральна усуває залишкові похибки, а диференціальна стабілізує реакцію.
2. **Підтримка курсу:** Регулятор коригує відхилення літака від заданого напрямку, забезпечуючи стабільне управління. Він враховує як поточні відхилення, так і тенденцію до їх зміни, щоб уникнути коливань.
3. **Контроль швидкості:** PID-регулятор регулює тягу двигунів для підтримки постійної швидкості польоту. Це особливо важливо під час зльоту, посадки або польоту в умовах змінного опору повітря.

4. **Стабілізація польоту:** Регулятор стабілізує рух літака навколо поздовжньої, поперечної та вертикальної осей, забезпечуючи плавність польоту та комфорт пасажирів.

Застосування

PID-регулятори використовуються у всіх аспектах управління польотом, від стабілізації літальних апаратів до автоматизації складних маневрів. Завдяки своїй універсальності вони залишаються невід'ємною частиною сучасних систем автопілота, забезпечуючи надійність і точність управління.

3.6 Переваги та обмеження використання PID-регулятора

PID-регулятор є одним із найпоширеніших інструментів управління завдяки своїй простоті, ефективності та універсальності. Проте його використання має як переваги, так і обмеження, які слід враховувати під час розробки систем управління.

Переваги PID-регулятора

1. **Простота реалізації:** PID-регулятор має зрозумілу структуру, яка легко інтегрується в різні системи.
2. **Універсальність:** Використовується для управління системами з різними характеристиками, такими як температура, швидкість, висота або тиск.
3. **Точність:** Забезпечує високий рівень точності в досягненні заданих параметрів, особливо в лінійних системах.
4. **Баланс між швидкістю, стабільністю та точністю:** Комбінація пропорційної, інтегральної та диференціальної складових дозволяє адаптувати регулятор до конкретних завдань.
5. **Широке застосування:** PID-регулятори успішно використовуються в промисловості, авіації, робототехніці, медицині та інших галузях.
6. **Модульність:** Можна використовувати окремі складові (П, ПІ, ПД) залежно від вимог конкретної системи.

Обмеження PID-регулятора

1. **Чутливість до шумів:** Диференціальна складова може посилювати вплив високочастотних шумів, що погіршує стабільність системи.
2. **Потреба в налаштуванні:** Для досягнення оптимальної роботи регулятора необхідне ретельне налаштування коефіцієнтів (K_p, K_i, K_d), що може бути складним і трудомістким процесом.
3. **Неадаптивність:** PID-регулятор не враховує змінних умов роботи системи, таких як зовнішні збурення або зміни параметрів об'єкта.
4. **Обмеження для нелінійних і складних систем:** У системах із нелінійною динамікою, значними затримками або високим рівнем взаємодії компонентів ефективність PID-регулятора знижується.
5. **Можливість перерегулювання:** Неправильне налаштування може спричинити перерегулювання або навіть нестабільність системи.
6. **Реакція на великі затримки:** У системах із великими затримками час реакції PID-регулятора може бути надто повільним, що призводить до погіршення якості управління.

3.7 Висновки до розділу

У цьому розділі було детально розглянуто структуру, принципи роботи та різновиди PID-регуляторів, а також їхнє застосування в системах автоматичного управління. Аналіз показав, що PID-регулятори завдяки своїй універсальності та простоті є ефективним інструментом для регулювання широкого спектра динамічних систем. Особлива увага приділялася їхньому використанню в системах автопілотування, де вони забезпечують точне управління основними параметрами польоту, такими як висота, швидкість і курс.

Водночас розглянуто переваги PID-регуляторів, зокрема їхню здатність забезпечувати баланс між швидкістю реакції, точністю й стабільністю системи. Однак аналіз також виявив обмеження цих регуляторів, зокрема їхню чутливість до шумів, складність налаштування та зниження ефективності в умовах нелінійної динаміки або великих затримок.

Висновки, зроблені в цьому розділі, створюють основу для подальшої розробки системи управління, де PID-регулятор відіграватиме ключову роль у забезпеченні надійності та ефективності автоматичного керування.

РОЗДІЛ 4. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

На основі проведених досліджень і вибраного підходу до реалізації, розпочинається етап розробки системи автопілотування. Для створення надійної, адаптивної та точної системи, що використовує PID-регулятори для управління літаком, необхідно детально описати ключові компоненти проєкту, функціональні блоки системи та їхню взаємодію.

У цьому розділі буде представлено детальний огляд розробленої системи, включаючи архітектуру програмного забезпечення, методику реалізації основних елементів і принципи їхньої інтеграції. Також буде розглянуто особливості використання мови програмування Lua та симуляційного двигуна X-Plane у контексті даного проєкту.

Окрему увагу буде приділено забезпеченню точності моделювання польотів, відповідності цифрової моделі літака його реальному прототипу, а також стабільності й повторюваності тестових сценаріїв.

4.1 DataRefs

DataRefs є основним механізмом доступу до даних у симуляційному двигуні X-Plane. Вони представляють собою змінні, які зберігають поточний стан системи або параметри літака, такі як швидкість, висота, положення рулів чи стан двигунів. DataRefs дозволяють зовнішнім програмам, таким як плагіни, взаємодіяти з симулятором, змінюючи або отримуючи необхідні дані в реальному часі.

Призначення DataRefs

DataRefs забезпечують:

- **Читання даних**: Наприклад, отримання значень швидкості, висоти, кута атаки чи положення шасі.

- **Запис даних**: Модифікація параметрів, таких як положення органів управління чи потужність двигуна.
- **Моніторинг стану**: Постійне спостереження за змінами параметрів симуляції для використання у зовнішніх системах, таких як автопілоти чи тренажери.

Структура DataRefs

Кожна DataRef у X-Plane має унікальне ім’я, що відображає її функціональність і приналежність до певної системи. Наприклад:

- `sim/cockpit2/controls/yoke_roll_ratio` — кут нахилу штурвала.
- `sim/flightmodel/position/latitude` — географічна широта літака.
- `sim/cockpit/electrical/instrument_brightness` — яскравість приладової панелі.

DataRefs також класифікуються за типами даних: числові, логічні, текстові, а також масиви для складніших параметрів.

Використання DataRefs у проєкті

У контексті розробки системи автопілота DataRefs слугують каналами обміну даними між симулятором X-Plane та розробленим програмним забезпеченням. Вони дозволяють:

- Отримувати параметри польоту для аналізу й прийняття рішень системою автопілота.
- Передавати команди автопілота до органів управління літаком, таких як елерони, тримери чи двигуни.
- Налагоджувати тестові сценарії, забезпечуючи точність і стабільність моделювання.

Переваги використання DataRefs

DataRefs спрощують інтеграцію зовнішніх систем із симулятором, забезпечують гнучкість у розробці та дозволяють досягти високого рівня деталізації управління літаком. У межах цього проєкту DataRefs є ключовим елементом зв'язку між симуляційним середовищем і алгоритмами автопілота, що забезпечує надійну та ефективну взаємодію.

4.2 Розробка програмного коду

4.2.1 Скрипт main.lua

Скрипт **main.lua** є ключовим елементом проєкту, який відповідає за запуск і управління всіма компонентами системи автопілотування. Він виконує роль основної точки входу в програму та реалізує кілька важливих функцій для забезпечення коректної роботи системи.

На початку скрипта виконуються налаштування SASL, які дозволяють активувати віртуальну панель літака, інтерактивні елементи й 3D рендеринг. Далі визначаються глобальні параметри проєкту, включаючи режим роботи (клієнтський чи автономний), налаштування віртуальних панелей та управління камерою.

У залежності від режиму роботи скрипт завантажує необхідні модулі, серед яких є засоби для роботи з конфігураціями, виконання математичних обчислень, управління файлами та камерою. Основні компоненти системи ініціалізуються через стартову функцію, яка відповідає за завантаження конфігурацій та створення словника пристроїв.

Скрипт також виконує перевизначення деяких DataRefs, що дозволяє контролювати елементи управління літаком, такі як поверхні керування та рульові механізми. Крім цього, за необхідності створюються власні DataRefs та команди для специфічних задач.

Для забезпечення роботи різних підсистем скрипт встановлює шляхи до допоміжних скриптів і ініціалізує компоненти літака, які будуть контролюватися автопілотом. Основні функції, такі як оновлення параметрів, обробка команд і відображення віртуальних елементів, виконуються за допомогою спеціальних обробників.

Скрипт також включає обробники завантаження, які активуються під час зміни сцен, аеропортів або літаків, і завершується функцією, що відповідає за

коректне завершення роботи модуля, включаючи зупинку всіх команд і відновлення початкових значень DataRefs.

Таким чином, **main.lua** виступає центральним компонентом, що координує роботу всіх частин системи, забезпечуючи стабільну та ефективну роботу автопілота.

4.2.2 Скрипт custom_datarefs.lua

Цей файл містить інструкції для створення глобальних змінних, відомих як "datarefs", які використовуються у програмному забезпеченні для симуляції реальності. Вони забезпечують моделювання поведінки й моніторинг стану різних систем і компонентів літака.

Основними розділами цього файлу є:

- **Налаштування**: Визначаються загальні параметри проєкту, які динамічно задаються за допомогою об’єкта `project_settings`.
- **Команди**: Створюються глобальні змінні, що відповідають за управління елементами літака, такими як елерони, елеватори, руль, стан шасі, розподіл ваги та інші функції.
- **Панелі управління**: Розробляються datarefs для різних панелей у кабіні пілота, які відповідають за взаємодію з керуючими елементами.
- **Системи управління поверхнями**: Визначаються datarefs для регулювання максимальних кутів і параметрів управління поверхнями, зокрема крилами, стабілізаторами та руддерами.
- **Шасі**: Описуються datarefs для контролю шасі, включаючи параметри висоти, повороту, гальмування та інших характеристик.

Цей файл є ключовим для забезпечення точного моделювання систем управління літаком і підтримки інтерактивності в симуляційному середовищі.

4.2.3 Скрипт pid.lua

Модуль **pid.lua** реалізує пропорційно-інтегрально-диференційний (PID) контролер, який використовується для управління системами, забезпечуючи зменшення похибки між поточним і цільовим станами. PID-контролери є одним із найпоширеніших інструментів у системах автоматизації, оскільки дозволяють досягти точного й стабільного регулювання.

Опис функцій модуля

1. **PID.New(Kp, Ki, Kd)**

   Функція створює новий екземпляр PID-контролера, використовуючи задані коефіцієнти K_p, K_i та K_d, які відповідають за пропорційну, інтегральну та диференціальну складові алгоритму.

2. **PID:GetSetting(targetValue, currentValue, currentSetting, dt, k)**

   Ця функція реалізує PID-алгоритм для розрахунку нового налаштування, враховуючи:

   - цільове значення (targetValue),
   - поточне значення (currentValue),
   - попереднє налаштування (currentSetting),
   - інтервал часу (dt),
   - додатковий множник (k).

3. **PID:Reset()**

   Функція скидає накопичені помилки контролера, зокрема значення попередніх помилок (prevE та prevPrevE), що дозволяє почати новий цикл регулювання.

4. **PID:Clone()**

   Створює копію поточного PID-контролера з усіма параметрами оригінального екземпляра.

5. **PID.Resets(...)**

   Скидає стан кількох PID-контролерів одночасно, викликаючи метод **Reset()** для кожного з переданих у список.

6. **PID.__tostring(self)**

   Ця функція визначає формат відображення PID-контролера як текстового рядка, що спрощує діагностику й налагодження.

7. **fields.default()**

   Повертає новий PID-контролер із встановленими за замовчуванням коефіцієнтами, які підходять для базових налаштувань.

Призначення модуля

Модуль **pid.lua** дозволяє створювати й налаштовувати PID-контролери для різноманітних завдань управління, таких як стабілізація систем, регулювання параметрів або компенсація похибок. Функції модуля підтримують гнучке налаштування параметрів, скидання накопичених помилок і створення копій контролерів для використання в різних частинах системи.

Завдяки своїй функціональності цей модуль забезпечує зручний інструментарій для управління PID-регуляторами у складних динамічних системах.

4.2.4 Скрипт global_constraints.lua

Файл **global_constraints.lua** є сховищем для констант і глобальних змінних, що використовуються у проєкті, пов’язаному з авіаційним симулятором. Він забезпечує централізоване управління ключовими параметрами, що сприяє спрощенню їхнього використання та модифікації.

Основні елементи файлу

1. **sim_speed і framerate_period**

   Ці змінні зберігають поточну швидкість симуляції та період частоти кадрів. Значення отримуються з глобальних властивостей симулятора й використовуються для обчислень, пов’язаних із часом і продуктивністю системи.

2. **EPSILON**

   Константа, яка представляє собою мале число. Вона використовується для вирішення проблем із числами з плаваючою точкою та для порівняння значень із врахуванням похибки.

3. **BASE_DREF_NAME та інші _G-змінні**

   Змінні цієї групи містять рядки, що визначають шляхи до різних систем симулятора. Наприклад:

   - `PANEL` використовується для доступу до системи панелі управління,
   - `PHYSICAL` — для фізичних налаштувань симулятора.

4. **PC_NAME, X-PLANE_PATH, AIRCRAFT_PATH, CONFIGS_PATH і MODULES_PATH**

   Ці змінні зберігають важливі шляхи, пов’язані з проєктом і симулятором. Вони отримуються за допомогою методів SASL, які визначають місцезнаходження файлів і папок, необхідних для роботи системи.

Призначення файлу

Файл **global_constraints.lua** виконує роль центрального місця для визначення важливих глобальних параметрів проєкту. Завдяки цьому спрощується доступ до ключових значень, їхнє налаштування та подальша підтримка. Така структура забезпечує зручність у розробці та модифікації, зменшуючи ризик помилок під час роботи з різними частинами симулятора.

4.2.5 Скрипт custom_commands.lua

Файл **custom_commands.lua** відповідає за визначення набору спеціальних команд, які використовуються для управління різними системами й компонентами літака в симуляторі. Завдяки цьому файлу розробники можуть створювати власні команди, які розширюють можливості управління моделлю літака, забезпечуючи точність і гнучкість.

Призначення команд

Визначені у файлі команди можуть бути прив’язані до кнопок, джойстиків або інших пристроїв введення, що дозволяє реалізувати інтуїтивне керування функціями літака. Це особливо корисно для налаштувань, які не передбачені стандартними командами симулятора.

Використання

Файл **custom_commands.lua** дозволяє розробникам легко додавати нові команди до симулятора, інтегруючи їх з існуючими системами управління. Це значно покращує можливості кастомізації та забезпечує гнучке налаштування інтерфейсу користувача для моделювання різних сценаріїв польоту.

4.2.6 Скрипт UnitTestManager.lua

Файл **UnitTestManager.lua** виконує функції управління юніт-тестами в проєкті, забезпечуючи автоматизацію їхнього запуску та контролю. Він дозволяє створювати, реєструвати та виконувати тести для різних компонентів системи, спрощуючи процес налагодження та перевірки функціоналу.

Основні елементи функціоналу

1. **Властивості та глобальні змінні**

   У файлі задаються властивості та глобальні змінні, які використовуються для відстеження статусу тестів і збереження їхніх результатів.

2. **Команди**

   Включає набір команд для запуску окремих юніт-тестів. Кожна команда пов’язана з обробником, який визначає логіку виконання відповідного тесту.

3. **Статуси тестів**

   У системі передбачені статуси, що відображають стан юніт-тестів:

   - "None" — тест не запущено,
   - "Start" — тестування розпочато,

- "Test" — тест виконується,
- "Stop" — тест завершено.

4. **Реєстрація обробників команд**

   За допомогою функції `registerCommandHandlers` виконується реєстрація обробників для запуску тестів і завершення юніт-тестування.

5. **Функція update**

   Ця функція виконується кожного кадру й перевіряє поточний стан юніт-тестів. У залежності від статусу вона виконує необхідні дії, наприклад, змінює статус або обробляє події.

6. **Масив components**

   Містить об’єкти, які представляють окремі юніт-тести. Кожен об’єкт відповідає за виконання тесту для певного компонента системи.

7. **Масив _testCommands**

   Зберігає об’єкти, що відповідають за команди запуску юніт-тестів. У кожному об’єкті визначена команда й пов’язаний із нею обробник.

Призначення

Файл **UnitTestManager.lua** забезпечує централізоване управління юніт-тестами, дозволяючи автоматизувати їхній запуск і контроль. Команди SASL, інтегровані в систему, можуть бути пов’язані з кнопками чи джойстиками, що забезпечує зручний доступ до тестування. Цей менеджер спрощує процес перевірки й налагодження компонентів системи, підвищуючи її стабільність і надійність.

4.2.7 Скрипт testStatsPrinter.lua

Файл **testStatsPrinter.lua** відповідає за збереження результатів юніт-тестів у текстовому форматі, забезпечуючи зручний спосіб ведення історії тестувань.

Основні функції

1. **Імпортування модулів**

   Файл використовує модуль **fileUtils** із папки **utils** для роботи з файлами, включаючи створення, читання та запис.

2. **Визначення шляху до папки з результатами**

   Змінна **DIRECTORY_PATH** задає місце розташування каталогу, в якому зберігаються результати тестів.

3. **Функція save**

   Основна функція **save** приймає ім’я тесту та його результат, після чого записує цю інформацію у файл, додаючи дату й час виконання тесту.

Принцип роботи функції save

- Генерується шлях до файлу, де будуть зберігатися результати, на основі імені тесту.
- Формується текст, що містить назву тесту, дату та час його виконання, результат і роздільники для підвищення читабельності.
- Якщо файл ще не існує, він створюється.
- Читається весь поточний вміст файлу, який зберігається у вигляді таблиці **lines**.
- Новий результат додається на початок таблиці **lines**, зберігаючи хронологічний порядок записів.
- Зміни спочатку записуються у тимчасовий файл.
- Оригінальний файл видаляється, а тимчасовий перейменовується в основний.

Призначення

Функція **save** забезпечує автоматичне оновлення файлу результатів, дозволяючи зберігати всі попередні записи. Це надає змогу зберігати історію тестувань, аналізувати результати й забезпечувати контроль якості в процесі розробки.

Файл **testStatsPrinter.lua** є важливим інструментом для організації тестувань, забезпечуючи ефективний спосіб документування результатів у текстовому форматі.

4.2.8 Скрипт EngineSystems.lua

Скрипт **EngineSystems.lua** відповідає за реалізацію трьох основних компонентів, що моделюють роботу системи двигунів у проєкті: **EnginesCS**, **EngineSystem** та **EngineFailures**. Ці компоненти охоплюють загальні властивості, індивідуальні параметри кожного двигуна й механізми обробки збоїв.

Основні компоненти

1. **EnginesCS**

   Цей компонент визначає загальні характеристики системи двигунів. Серед основних параметрів:

   - Вібрації для кожного двигуна (наприклад, engine1, engine2, engine3, engine4),
   - Мінімальний тиск оливи (minDavlMaslaIndicator),
   - Інші спільні властивості, які є загальними для всієї системи.

2. **EngineSystem**

   Компонент, який відповідає за специфічні параметри кожного двигуна. Для кожного двигуна задаються:

   - Напруга,
   - Обертальний момент,
   - Стан роботи,
   - Споживання палива,
   - Рівень вібрацій,
   - Індикатори небезпечних ситуацій.

     Ця структура дозволяє детально моделювати роботу кожного двигуна окремо, з урахуванням його характеристик і стану.

3. **EngineFailures**

   Компонент для моніторингу та обробки збоїв у роботі двигунів. Він охоплює:

   - Вхідні та вихідні властивості, пов’язані зі збоями,
   - Типи збоїв, включаючи:
     - Збої стартера,
     - Проблеми з паливною системою,
     - Збої насосів і системи охолодження,
     - Небезпечні рівні вібрацій,
     - Інші критичні ситуації.

Призначення

Скрипт **EngineSystems.lua** забезпечує комплексне моделювання стану, параметрів і збоїв у роботі двигунів. Завдяки цьому створюється деталізована й реалістична симуляція роботи двигунів, яка враховує широкий спектр характеристик і сценаріїв відмов. Цей підхід дозволяє ефективно відслідковувати стан системи й реагувати на можливі проблеми в реальному часі.

4.2.9 Скрипт EngineSystem.lua

Скрипт **EngineSystem.lua** реалізує моделювання роботи системи двигуна для X-Plane. Він використовує глобальні змінні (datarefs) і команди X-Plane для взаємодії з двигуном, дозволяючи керувати його станом і параметрами в симуляторі.

Основні функції та змінні

1. **Datarefs**

   Глобальні змінні X-Plane, які використовуються для обміну даними між компонентами симулятора. У цьому скрипті вони зберігають інформацію про стан двигуна, такі як ступінь обертання (N1, N2), стан реверсу, приводу двигуна тощо.

2. **EngineState**

   Визначає можливі стани двигуна:

   - OFF — двигун вимкнений,
   - TurningON — двигун запускається,
   - ON — двигун увімкнений,
   - TurningOff — двигун зупиняється.

3. **AutoReverseStage**

   Зберігає стани автоматичного реверсу.

4. **readParameters()**

   Зчитує параметри роботи двигуна із системи.

5. **start()**

   Виконує ініціалізацію запуску двигуна, знаходячи необхідні команди через номер двигуна.

6. **readInputs()**

   Зчитує вхідні сигнали для роботи двигуна.

7. **TryStartEngines(), TryStartEnginesInAir()**

   Визначають логіку запуску двигунів на землі, у повітрі, а також для фейкових запусків.

8. **SetThrottle()**

   Контролює відкриття дросельної заслінки.

9. **Reverse(), AutoReverse()**

   Керують включенням і режимами реверсу двигуна.

10. **GetVibrations()**

    Оцінює рівень вібрацій двигуна.

11. **StopEngine()**

    Зупиняє роботу двигуна.

12. **MapGasTemp(), MapN1(), MapN2()**

    Відображають температуру газів і ступені обертання двигуна (N1, N2).

13. **SetActualPanelValues(), SetEnginePanelValues()**

    Встановлюють значення на панелі керування двигуном.

14. **GetEngineStatus()**

    Отримує поточний статус двигуна.

15. **setOutputs(), writeOutputs()**

    Встановлюють і записують вихідні параметри роботи двигуна.

16. **Обробники подій**

- `settingThrottle_Handler()`, `stopEngine_Handler()`, `quickStart_Handler()`: обробляють команди запуску, зупинки або швидкого запуску двигуна.
- `reverseOn_Handler()`, `reverseOff_Handler()`: керують вмиканням і вимиканням реверсу.
- `StruzhkaVMasle_Handler()`: обробляє події, пов’язані з несправностями, наприклад, появою сторонніх часток в оливі.

Призначення

Скрипт **EngineSystem.lua** забезпечує реалістичне моделювання поведінки двигуна в симуляторі, підтримуючи точне управління його станом і параметрами. Це досягається за допомогою datarefs, команд і функцій, які дозволяють запускати двигун, контролювати реверс, аналізувати вібрації й мапувати параметри. Такий підхід забезпечує інтерактивність і точність у відтворенні роботи системи двигуна.

4.2.10 Скрипт SurfaceControls.lua

Скрипт **SurfaceControls.lua** відповідає за моделювання систем управління поверхнями повітряного судна, включаючи елерони, спойлери, елеватори, руль напряму, стабілізатор, закрилки й передкрилки. Його функціонал забезпечує відтворення реалістичної поведінки цих компонентів у симуляторі.

Компоненти скрипта

- **ailerons**

  Реалізують контроль елеронів. Параметри включають:

  - Максимальні та мінімальні кути відхилення,
  - Максимальну швидкість переміщення (Velocity per Second, VPS),
  - Вхідні сигнали з контролерів елеронів (_iRoll).

- **spoilers**

  Контролюють спойлери, додаючи до параметрів елеронів функціонал для моніторингу й управління гідравлічним споживанням.

- **elevators**

  Відповідають за управління елеваторами. Їхні параметри аналогічні елеронам і включають:

  - Граничні кути,
  - Швидкість переміщення,
  - Вхідні дані з контролерів елеваторів.

- **rudder**

  Контролює руль напряму, використовуючи подібні параметри, зокрема:

  - Максимальні кути відхилення,
  - Швидкість переміщення,
  - Вхідні сигнали (_iYaw).

- **stab**

  Реалізує управління стабілізатором, включаючи параметри, як-от:

  - Граничні кути,
  - Максимальну швидкість переміщення,
  - Гідравлічне споживання.

- **flaps**

  Відповідають за контроль закрилків. Крім стандартних параметрів, включають моделювання гідравлічного споживання.

- **slats**

  Контролюють передкрилки, маючи аналогічні параметри, але також враховуючи гідравлічне навантаження.

Призначення

Скрипт **SurfaceControls.lua** забезпечує точне моделювання фізичних характеристик і динамічних властивостей поверхонь управління повітряного судна. Він враховує:

- Діапазон руху й швидкість переміщення компонентів,
- Вплив на аеродинамічні параметри літака,
- Споживання гідравлічної енергії.

Завдяки деталізації скрипт дозволяє відтворювати реалістичну поведінку літака в різних умовах польоту, забезпечуючи інтерактивність і високу точність симуляції.

4.3 Висновки до розділу

Розробка системи автопілотування літаків базується на ефективному використанні мови програмування Lua та механізму DataRefs у середовищі X-Plane. Зосереджуючи увагу на створенні продуктивних скриптів, ми формуємо процес, що забезпечує точність, надійність і стабільність під час тестування системи.

Інтеграція DataRefs у систему автопілотування відкриває широкі можливості для гнучкого управління й візуалізації даних польоту. Цей механізм слугує "містком", що пов’язує програми та плагіни з реальними параметрами літака й умовами середовища в режимі реального часу.

Скрипти, створені на основі Lua, виконують ключові функції в реалізації автопілота, забезпечуючи виконання необхідних операцій. Вони є важливими компонентами, кожен із яких має специфічне призначення у загальній структурі системи.

Таким чином, успішна розробка системи автопілотування потребує глибокого розуміння й грамотного використання обраних інструментів — мови Lua та механізму DataRefs у X-Plane. Цей підхід створює міцну основу для подальшого вдосконалення та розвитку системи.

РОЗДІЛ 5. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

У цьому розділі представлені результати дослідження й аналізу розробленої системи автопілотування літаків, яка базується на використанні PID-регуляторів, мови програмування Lua та механізму DataRefs у симуляційному середовищі X-Plane.

Основною метою є оцінка роботи системи автопілотування в різних умовах і сценаріях польоту. Особливу увагу приділено аналізу результатів польотів, виконаних із застосуванням створеної системи, та їхньому порівнянню з показниками реальних польотів літака. Це дає змогу оцінити точність, стабільність і надійність автопілота, визначити його відповідність поставленим вимогам і перспективи для подальшого вдосконалення.

5.1 Тест зліту

1. **Підготовка до розбігу**
   - Утримуючи літак на гальмах, плавно переведіть важелі керування спочатку зовнішніх, а потім внутрішніх двигунів у злітний режим.
   - Плавно відпустіть гальма та ввімкніть секундомір у момент початку руху літака злітною смугою. Почніть розбіг.
2. **Розбіг**
   - Під час розбігу утримуйте штурвал у відхиленому від себе положенні. Для підтримання напрямку руху використовуйте педалі, враховуючи, що управління поворотом коліс носової опори шасі повинно бути активним.
   - Коли швидкість наближається до значення, необхідного для відриву носового колеса, відхиліть штурвал на себе, щоб виконати підйом

носової частини літака. Стежте, щоб кут тангажу під час відриву не перевищував 8°.

3. **Фіксація даних**

   У момент фактичного відриву літака, який можна визначити за показанням варіометра, зупиніть секундомір. Зафіксуйте значення швидкості, при якій відбувся відрив, а також час, витрачений на розбіг.

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

1. **Підготовка до розбігу**

   Використовуйте методику, описану в підпункті 5.1 для задання початкових умов і виконання злету.

2. **Імітація відмови двигунів**

   - Під час розбігу, коли швидкість досягне 220 км/год, активуйте відмову подачі палива для 1-го двигуна.
   - Утримуйте напрямок руху літака педалями, компенсуючи тенденцію до розвороту.

3. **Гальмування після відмови**

   - Увімкніть секундомір і переведіть РУД (режим управління двигунами) усіх двигунів на "Малий газ".
   - Виведіть на монітор інформацію про кути відхилення руля напряму.
   - Активуйте реверс тяги для 2-го та 3-го двигунів.
   - Випустіть спойлери та гальмівні щитки.
   - Натисніть на гальмівні підніжки.

4. **Зниження швидкості та вимкнення механізації**
    - При досягненні швидкості 60 км/год приберіть механізацію крила.
    - Коли швидкість впаде до 50 км/год, вимкніть реверс тяги.
5. **Фіксація часу**
    - Продовжуйте гальмування до повної зупинки літака.
    - У момент, коли швидкість досягне 0 км/год, зафіксуйте 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

1. **Завдання початкових умов**

   Використовуйте методику, описану в підпункті 5.1, для підготовки до злету. Встановіть комерційне навантаження 12100 кг так, щоб загальна маса літака G досягала максимальної злітної ваги 170000 кг. Задайте кут нахилу -4,8∘.
2. **Розбіг та відмова двигуна**
    - Під час розбігу, досягнувши швидкості 250 км/год, імітуйте відмову подачі палива 1-го двигуна.
    - Продовжуйте розбіг, компенсуючи відхилення літака від осі ЗПС за допомогою педалей.
3. **Фіксація параметрів під час розбігу**

   Коли швидкість літака досягне 260 км/год, виведіть на монітор АРМІ інформацію про кут відхилення руля напряму.
4. **Відрив літака**

   Виконайте відрив літака, спостерігаючи за показаннями приладів. У момент відриву вимкніть секундомір.

5. **Після відриву**
    - Приберіть шасі.
    - Встановіть вертикальну швидкість набору висоти так, щоб підтримувати постійну повітряну швидкість 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. **Виконання злету**

   Використовуйте методику, описану в підпункті 5.1 для злету.
2. **Налаштування параметрів польоту**

   Після злету встановіть такі параметри:
    - Усі чотири двигуни перебувають у робочому стані.
    - Загальна маса літака G=145000 кг.
    - Шасі прибрані.
    - Балансувальний кут ϕ.
3. **Задайте висоту**

   Встановіть висоту горизонтального польоту на рівні H=2650 м.
4. **Відключення програми розходу палива**

   Деактивуйте програму, яка відповідає за розрахунок і управління витратою палива.
5. **Горизонтальний політ**
    - Продовжуйте набір швидкості до досягнення 530 км/год.
    - Витримуйте горизонтальний політ із цією швидкістю протягом двох хвилин.
    - Зафіксуйте значення обертів двигунів наприкінці двохвилинного відрізка польоту.

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. **Зліт**

   Використовуйте методику, описану в підпункті 5.1 для злету.

2. **Задання параметрів**

   Встановіть наступні параметри:

   - Висота польоту H=600 м,
   - Вага палива = 33000 кг,
   - Вага комерційного навантаження = 7100 к.

3. **Перевірка часу розгону**

   - Виконуйте горизонтальний політ із постійною швидкістю 350км/год.
   - Встановіть за допомогою РУД оберти двигунів у режим "Злітний".
   - Виміряйте час розгону від 350 км/год до 500 км/год.

4. **Перевірка часу гальмування**

   - Виконуйте горизонтальний політ із постійною швидкістю 390 км/год.
   - Встановіть оберти двигунів за допомогою РУД на рівні 93%.
   - Виміряйте час гальмування від 500 км/год 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. **Зліт**

   Використовуйте методику, описану в підпункті 5.1 для злету.

2. **Введення параметрів польоту**

   Встановіть такі початкові параметри:

   - Висота H=2650 м,
   - Вага палива =35000 кг,
   - Вага комерційного навантаження =17100 кг.

3. **Горизонтальний політ**

   - Використовуючи РУД (режим управління двигунами), встановіть номінальний режим роботи двигунів.
   - Після досягнення необхідної швидкості виконуйте горизонтальний політ із її витримуванням 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. **Зліт**

   Використовуйте методику, описану в підпункті 5.1 для злету.

2. **Введення параметрів**

   Встановіть початкові параметри польоту:

   - Вага палива =42000 кг,
   - Вага комерційного навантаження =35100 кг.

3. **Розгін на малій висоті**

   - Підтримуйте висоту в межах 20−25 м.

- Розганяйте літак до заданої приладної швидкості.

4. **Набір висоти**
   - У момент досягнення заданої швидкості ввімкніть секундомір.
   - Почніть набір висоти, підтримуючи встановлену приладну швидкість.
5. **Фіксація часу**

   Після досягнення висоти H=2000 м зафіксуйте показання 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 тестування підтвердили здатність розробленої цифрової моделі ефективно відтворювати фізичні характеристики літака. Результати низки експериментів не лише підтвердили функціональність і надійність створеної моделі, але й продемонстрували її високу точність у відтворенні реальних параметрів повітряного судна.

Особливо важливим є те, що очікувані результати кожного тесту були засновані на реальних даних, що забезпечило об'єктивність і достовірність отриманих висновків. Це підкреслює ключову роль використання фактичних даних у перевірці й адаптації цифрових моделей.

Проєкт став успішним прикладом створення цифрової моделі, яка здатна точно відтворювати фізичні характеристики літака. Досягнуті результати свідчать про перспективність такого підходу для майбутніх досліджень у цій сфері. Успішна реалізація роботи підтверджує, що цифрові моделі можуть слугувати ефективним інструментом для моделювання реальних умов і параметрів, відкриваючи нові горизонти для їхнього застосування в майбутньому.

ВИСНОВКИ

У рамках цієї дипломної роботи було підтверджено можливість застосування цифрових моделей для моделювання реальних фізичних процесів, що відбуваються під час польоту літака. Робота демонструє, як інтеграція таких технологій, як мова програмування Lua, симуляційний двигун X-Plane і PID-регулятори, може забезпечити створення ефективної та надійної системи автопілотування.

Аналіз предметної області та визначення вимог до характеристик проектування дозволили глибше зрозуміти завдання та підходи до його вирішення. Вивчення PID-регуляторів стало основою для розробки механізмів управління динамікою польоту, забезпечивши точність і стабільність.

Процес розробки включав роботу з механізмом DataRefs і написання скриптів, що дозволило створити цифрову модель, яка відтворює характеристики літака з високою точністю. Проведені тести й аналіз підтвердили надійність і функціональність розробленої системи.

Ця робота підкреслює значення й потенціал цифрових моделей в автоматичних системах керування, особливо в авіації. Подальший розвиток і впровадження таких систем може мати значний практичний ефект, сприяючи підвищенню безпеки польотів, полегшенню роботи пілотів і покращенню ефективності управління літаками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Autopilot Control System for Fixed-Wing UAVs: Design and Implementation [Електронний ресурс] / J. Zhang, K. Li, M. Wang. – 2019. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S0967066119302278.
2. Basics of PID Control [Електронний ресурс] / National Instruments. – 2021. – Режим доступу до ресурсу: https://www.ni.com/en-us/innovations/pid-control.html.
3. Lua Programming Language Overview [Електронний ресурс]. – Режим доступу до ресурсу: http://www.lua.org/manual/5.4/.
4. Flight Simulation in X-Plane [Електронний ресурс] – Режим доступу до ресурсу: https://www.x-plane.com/.
5. UAV Navigation Systems [Електронний ресурс] / P. Mohan, R. Singh. – 2022. – Режим доступу до ресурсу: https://uav-systems.tech/navigation.
6. Fundamentals of Autopilot Design [Електронний ресурс] / E. Johnson, S. Ames. – 2018. – Режим доступу до ресурсу: https://autopilotdesign.com/.
7. X-Plane SDK Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.x-plane.com/sdk/.
8. Advanced Lua Scripting Techniques [Електронний ресурс] / K. Brown. – 2020. – Режим доступу до ресурсу: https://www.luadevguide.com/.
9. Analysis of Flight Dynamics in Simulation [Електронний ресурс] / A. Davis. – 2021. – Режим доступу до ресурсу: https://flightdynamics.com/resources.
10. Implementing Realistic Aircraft Models in X-Plane [Електронний ресурс] – Режим доступу до ресурсу: https://xplanemodeling.com/.

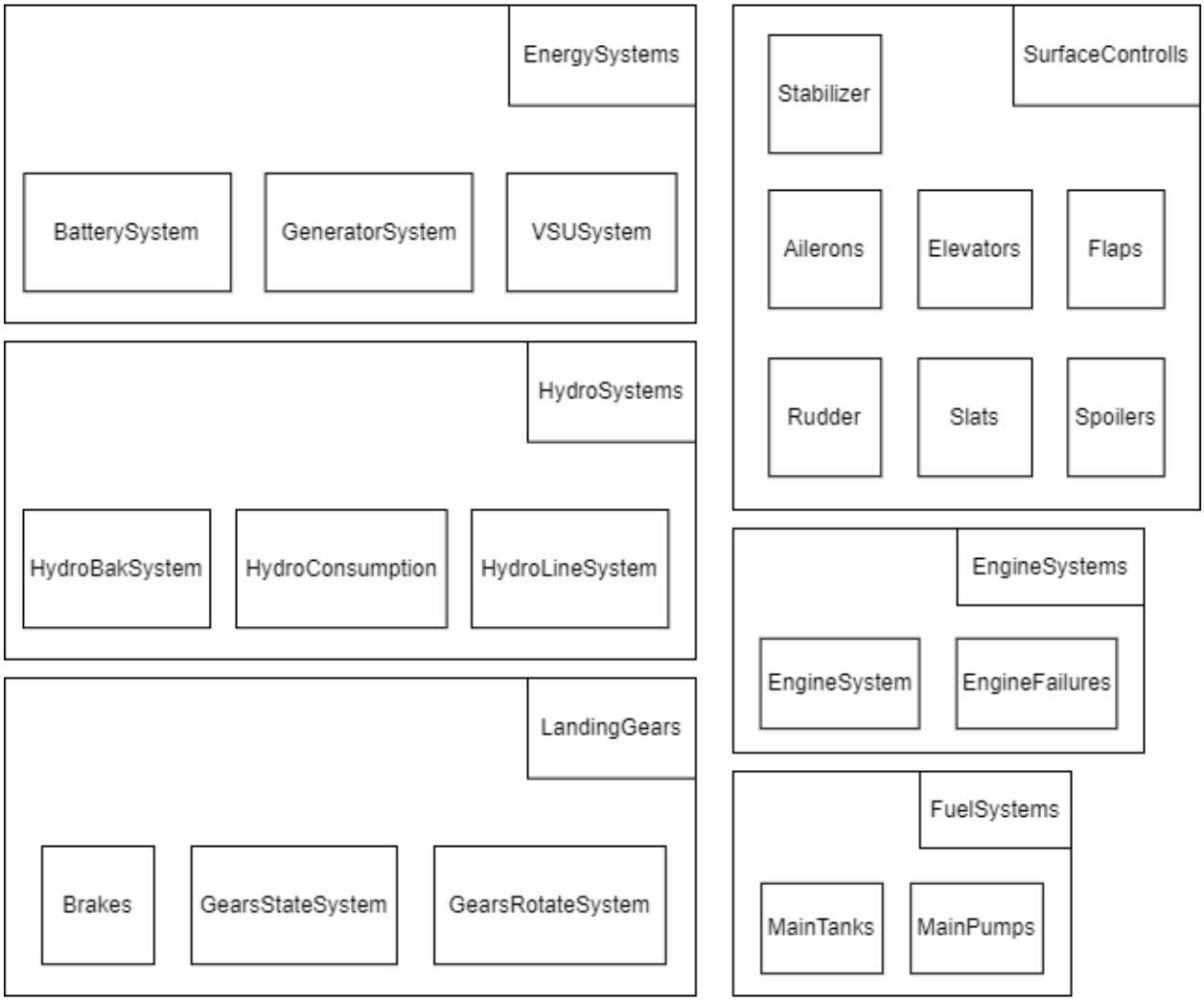
ДОДАТОК 1

Система верифікації цифрового двійника літака

Структура систем літака
(Структурна схема)

Аркушів 1

Київ – 2024 р.


EnergySystems
BatterySystem
GeneratorSystem
VSUSystem
SurfaceControlls
Stabilizer
Ailerons
Elevators
Flaps
Rudder
Slats
Spoilers
HydroSystems
HydroBakSystem
HydroConsumption
HydroLineSystem
EngineSystems
EngineSystem
EngineFailures
LandingGears
Brakes
GearsStateSystem
GearsRotateSystem
FuelSystems
MainTanks
MainPumps

ДОДАТОК 2

Система верифікації цифрового двійника літака

Діаграма варіантів користування
(Функціональна схема)

Аркушів 1

Київ – 2024 р.


Користувач
Головне меню
<include>
<include>
<include>
Меню вибору літака
Кнопка початку симуляції
Меню вибору полігону
<extend>
<extend>
Меню настройки літака
<include>
Меню настройки полігону
Меню вибору тесту
<extend>
Відобразити результати

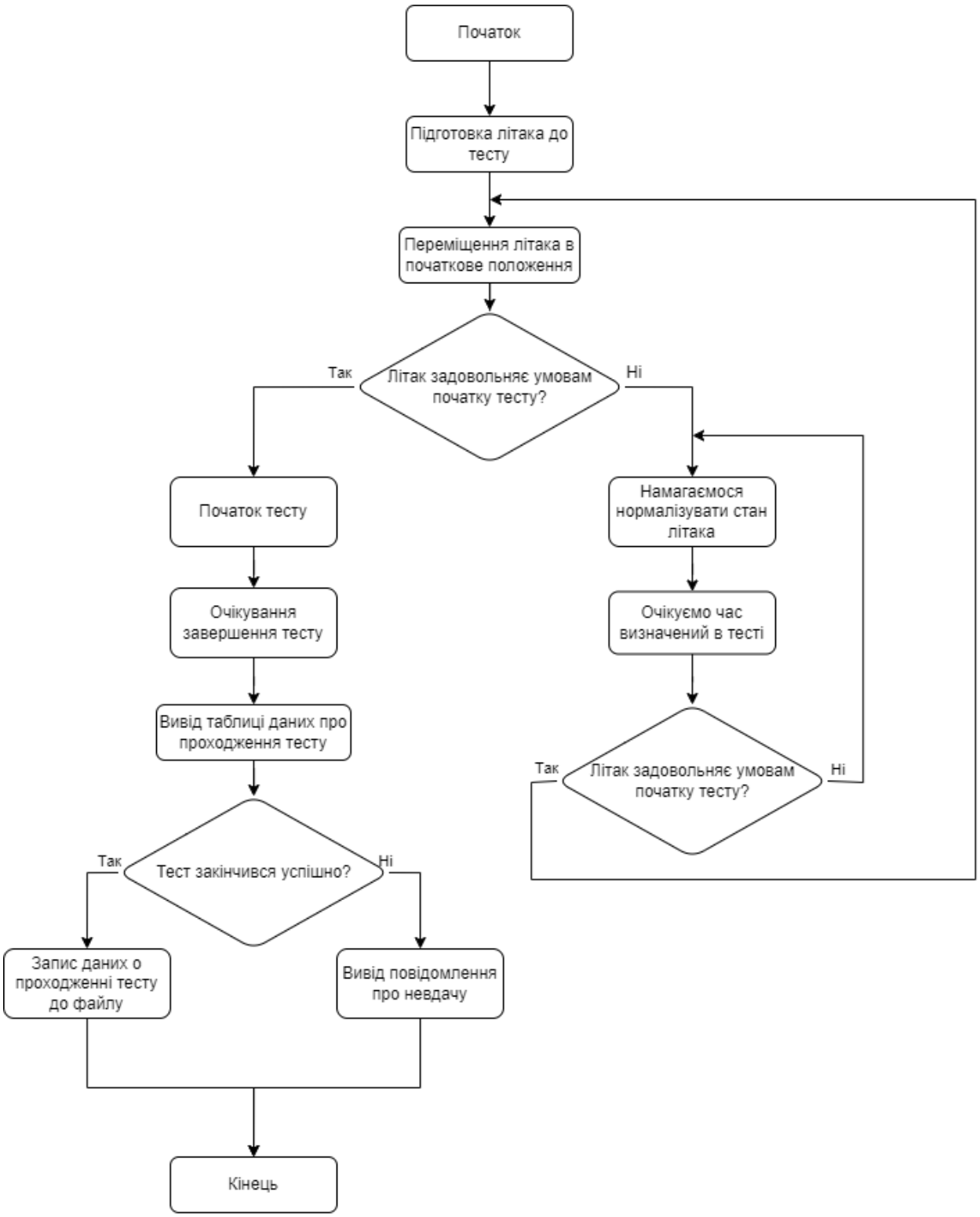
ДОДАТОК З

Система верифікації цифрового двійника літака

Алгоритм дій програмного забезпечення (Принципова схема)

Аркушів 1

Київ – 2024 р.


Початок
Підготовка літака до тесту
Переміщення літака в початкове положення
Літак задовольняє умовам початку тесту?
Так
Ні
Початок тесту
Очікування завершення тесту
Вивід таблиці даних про проходження тесту
Тест закінчився успішно?
Так
Ні
Запис даних о проходженні тесту до файлу
Вивід повідомлення про невдачу
Кінець
Намагаємося нормалізувати стан літака
Очікуємо час визначений в тесті
Літак задовольняє умовам початку тесту?
Так
Ні

ДОДАТОК 4

Система верифікації цифрового двійника літака

Текст програмного коду

Аркушів 19

Київ – 2024 р.

```
-- #region datarefs
local throttleArray =
globalPropertyfa('sim/cockpit2/engi
ne/actuators/throttle_ratio')
local forcedThrottleArray =
globalPropertyfa('sim/flightmodel/e
ngine/ENGN_thro_use')
local reverserOnArray =
globalPropertyfa('sim/cockpit2/annu
nciators/reverser_on')



local N1Array =
globalPropertyfa('sim/flightmodel/e
ngine/ENGN_N1_')
local N2Array =
globalPropertyfa('sim/flightmodel/e
ngine/ENGN_N2_')
local ActualFuelPreasureArray =
globalPropertyfa('sim/cockpit2/engi
ne/indicators/fuel_pressure_psi')
local ActualOilPreasureArray =
globalPropertyfa('sim/flightmodel/e
ngine/ENGN_oil_press_psi')
local ActualOilTemperatureArray =
globalPropertyfa('sim/flightmodel/e
ngine/ENGN_oil_temp_c')
local ActualFuelConsumptionArray =
globalPropertyfa('sim/flightmodel/e
ngine/ENGN_FF_')
local ActualGasTemperatureArray =
globalPropertyfa('sim/flightmodel/e
ngine/ENGN_EGT_c')
local IntroTime =
globalPropertyf('sim/aircraft/engin
e/fuel_intro_time_jet')
local forcedThrottle
local forcedReverseThrottle
local reverseOnCommand
local reverseOffCommand
local struzhkaVMasleCommand

defineProperty('_systemName', nil)

defineProperty('_pEngineNumber',
nil)
defineProperty('_pHighVibrationValu
e', nil)
defineProperty('_pDangerVibrationVa
lue', nil)
defineProperty('_pIndicationPower',
nil)
defineProperty('_pNormalVoltage',
nil)
defineProperty('_pMinV', nil)
defineProperty('_pMin_VSU_RPM',
nil)
defineProperty('_pOilAmount', nil)
defineProperty('_pNormalFuelPressur
e', nil)

defineProperty('_iVoltage', nil)
defineProperty('_iThrotle', nil)
defineProperty('_iThrottleAll',
nil)
defineProperty('_iSeparateThrottle'
, nil)
defineProperty('_iVibrationCheckBut
ton', nil)
defineProperty('_iVibrationCheckRot
ator', nil)
defineProperty('_iEngineStart',
nil)
defineProperty('_iStopSwitch', nil)
defineProperty('_iRodRaboty', nil)
```

```
defineProperty('_iReverseSwitch',
nil)
defineProperty('_iReverseSwitchAll'
, nil)
defineProperty('_iSeparateReverse',
nil)
defineProperty('_iVSU_RPM', nil)
defineProperty('_iEngineStartInAirB
utton', nil)

defineProperty('_iStarterFailed',
nil)
defineProperty('_iFuelFireFailed',
nil)
defineProperty('_iZabrosTVGFail',
nil)
defineProperty('_iStruzhkaVMasle',
nil)
defineProperty('_iPovyshRashodMasla
', nil)
defineProperty('_iMinDavlTopliva',
nil)
defineProperty('_iMinDavlMasla',
nil)
defineProperty('_iPodachaToplFailur
e', nil)
defineProperty('_iPodkachNasosFailu
re', nil)
defineProperty('_iZamokReversaFailu
re', nil)
defineProperty('_iNasosFailure',
nil)
defineProperty('_iProizvVklReversa'
, nil)
defineProperty('_iOpasnayaVibraciya
', nil)

defineProperty('_oEngineState',nil)
defineProperty('_oConsumption',nil)
defineProperty('_oVibrationFWD',nil
)
defineProperty('_oVibrationBACK',ni
l)
defineProperty('_oHighVibration',ni
l)
defineProperty('_oDangerVibration',
nil)
defineProperty('_oPanelZapuskaRab',
nil)
defineProperty('_oReverseOnIndicati
on',nil)
defineProperty('_oEngineRPM_Percent
',nil)
defineProperty('_oOpasOborotyStater
a',nil)
defineProperty('_oMinOilRemaining',
nil)

defineProperty('_oOpasnayaVibraciya
Indicator', nil)
defineProperty('_oMinDavlToplIndica
tor', nil)
defineProperty('_oStruzhkaVMasleInd
icator', nil)
defineProperty('_oOstTop2000Indicat
or', nil)
defineProperty('_oMinDavlMaslaIndic
ator', nil)
defineProperty('_oToplFiltrNeRabInd
icator', nil)
defineProperty('_oPerezapuskVozdOtk
rIndicator', nil)
defineProperty('_oVNARabIndicator',
nil)
defineProperty('_oZamRevOtkrytIndic
ator', nil)
```

```
defineProperty('_oVNAPuskIndicator'
, nil)

defineProperty('_oActualN1', nil)
defineProperty('_oActualN2', nil)
defineProperty('_oSuppostedN1',
nil)
defineProperty('_oSuppostedN2',
nil)
defineProperty('_oActualFuelPressur
e', nil)
defineProperty('_oSuppostedFuelPres
sure', nil)
defineProperty('_oActualOilPressure
', nil)
defineProperty('_oSuppostedOilPress
ure', nil)
defineProperty('_oActualOilTemperat
ure', nil)
defineProperty('_oSuppostedOilTempe
rature', nil)
defineProperty('_oActualFuelConsump
tion', nil)
defineProperty('_oSuppostedFuelCons
umption', nil)
defineProperty('_oActualFuelRemaini
ng', nil)
defineProperty('_oSuppostedFuelRema
ining', nil)
defineProperty('_oActualGasTemperat
ure', nil)
defineProperty('_oSuppostedGasTempe
rature', nil)
defineProperty('_oSuppostedOilRemai
ning', nil)
defineProperty('_oSuppostedFuelPres
sure', nil)
```

```
-- #endregion

-- #region parameters
local pEngineNumber
local pHighVibrationValue
local pDangerVibrationValue
local pIndicationPower
local pNormalVoltage
local pMinV
local pMin_VSU_RPM
local pOilAmount
local pNormalFuelPressure
local pEngineTurningOnTime = 40
-- #endregion

-- #region inputs
local iVoltage
local HasEnergy
local iThrotle
local iThrottleAll
local iSeparateThrottle
local iVibrationCheckButton
local iVibrationCheckRotator
local iEngineStart
local iStopSwitch
local iRodRaboty
local iReverseSwitch
local iReverseSwitchAll
local iSeparateReverse
local iVSU_RPM
local iEngineStartInAirButton

local iStarterFailed
local iFuelFireFailed
local iZabrosTVGFail
local iStruzhkaVMasle
```

```
local iPovyshRashodMasla
local iMinDavlTopliva
local iMinDavlMasla
local iPodachaToplFailure
local iPodkachNasosFailure
local iZamokReversaFailure
local iNasosFailure
local iProizvVklReversa
local iOpasnayaVibraciya
-- #endregion

-- #region outputs
local oEngineState = 1
local oConsumption = 0
local oVibrationFWD = 0
local oVibrationBACK = 0
local oHighVibration = 0
local oDangerVibration = 0
local oPanelZapuskaRab = 0
local oReverseOnIndication = 0
local oEngineRPM_Percent = 0
local oOpasOborotyStatera = 0
local oMinOilRemaining = 0


local oOpasnayaVibraciyaIndicator =
0
local oMinDavlToplIndicator = 0
local oStruzhkaVMasleIndicator = 0
local oOstTop2000Indicator = 0
local oMinDavlMaslaIndicator = 0
local oToplFiltrNeRabIndicator = 0
local oPerezapuskVozdOtkrIndicator
= 0
local oVNARabIndicator = 0
local oZamRevOtkrytIndicator = 0
local oVNAPuskIndicator = 0


local oActualN1 = 0
local oActualN2 = 0
local oSuppostedN1 = 0
local oSuppostedN2 = 0
local oActualFuelPressure = 0
local oSuppostedFuelPressure = 0
local oActualOilPressure = 0
local oSuppostedOilPressure = 0
local oActualOilTemperature = 0
local oSuppostedOilTemperature = 0
local oActualFuelConsumption = 0
local oSuppostedFuelConsumption = 0
local oActualFuelRemaining = 0
local oSuppostedFuelRemaining = 0
local oActualGasTemperature = 0
local oSuppostedGasTemperature = 0
local oSuppostedOilRemaining = 0
local oSuppostedFuelPressure = 0

-- #endregion

-- #region commands
local engage_starter
local hold_reverse
local reverse_toggle_command
local
start_engine_without_conditioning
local settingThrottle
local stopEngine
-- #endregion

-- #region prev values
local prevStopSwitch
local prevReverseSwitch
local prevReverseSwitchAll
local prevEngineStartInAirButton
```

```
local prevStartedFailed
local prevSuppostedGasTemperature
local prevSuppostedOilTemperature
local prevPodachaToplFailure
-- #endregion

-- #region local
local _started
local _dt
local manualControl = true
local panelZapuskaRabTimer = 30000
local startInAirFailTimer = 10000
local startInAirTimer = 2000
local startInAirStarted
local forcedThrottleOn = false
local forceQuickStart = false
local quickStartTimer = 0
local reverseOnInProgress = false
local reverseOffInProgress = false
local savedN1whenN2is17 = 0
local gasTempReached800 = 0
-- #endregion

AutoReverseStage = {
  Off = 1,
  GoingDown = 2,
  Toggle = 3,
  GoingUp = 4,
  Finished = 5,
}

EngineState = {
      OFF = 1,
      TurningON = 2,
      ON = 3,
      TurningOff = 4,
}
local autoReverseOnStage =
AutoReverseStage.Off
local autoReverseOffStage =
AutoReverseStage.Off

local engineState = EngineState.OFF
local engineTurningOnTimer = 0

function readParameters()
      pEngineNumber =
get(_pEngineNumber)
  pHighVibrationValue =
get(_pHighVibrationValue)
      pDangerVibrationValue =
get(_pDangerVibrationValue)
      pIndicationPower =
get(_pIndicationPower)
      pNormalVoltage =
get(_pNormalVoltage)
      pMinV = get(_pMinV)
  pMin_VSU_RPM = get(_pMin_VSU_RPM)
  pOilAmount = get(_pOilAmount)
      pNormalFuelPressure =
get(_pNormalFuelPressure)
end

local soundID
function start()
      systemName = get(_systemName)

      prevStopSwitch = 0
      prevReverseSwitch = 0
      prevReverseSwitchAll = 0
      prevEngineStartInAirButton =
0
  prevStartedFailed = 0
  prevSuppostedGasTemperature = 0
```

```
  prevSuppostedOilTemperature = 0
  prevPodachaToplFailure = 0

      oActualFuelRemaining = 24000
      oSuppostedFuelRemaining =
24000
      engage_starter =
findCommand("sim/ignition/engage_st
arter_"..pEngineNumber)
      hold_reverse =
findCommand("sim/engines/thrust_rev
erse_hold_"..pEngineNumber)
  reverse_toggle_command =
findCommand("sim/engines/thrust_rev
erse_toggle_"..pEngineNumber)
      start_engine_without_conditio
ning =
findCommand(COMMANDS.."/engines/sta
rt_engine_without_conditioning")
  settingThrottle =
findCommand(COMMANDS.."/engines/eng
ine"..pEngineNumber.."/settingThrot
tle")

registerCommandHandler(settingThrot
tle, 0, settingThrottle_Handler)
  stopEngine =
findCommand(COMMANDS.."/engines/eng
ine"..pEngineNumber.."/stopEngine")

registerCommandHandler(stopEngine,
0, stopEngine_Handler)
  forcedThrottle =
globalPropertyf(ENGINES_NAME.."/eng
ine"..pEngineNumber.."/forcedThrott
le")
  forcedReverseThrottle =
globalPropertyf(ENGINES_NAME.."/eng
ine"..pEngineNumber.."/forcedRevers
eThrottle")
  qiuckEngineStart =
findCommand(COMMANDS.."/engines/eng
ine"..pEngineNumber.."/quickStart")

registerCommandHandler(qiuckEngineS
tart, 0, quickStart_Handler)
  reverseOnCommand =
findCommand(COMMANDS.."/engines/eng
ine"..pEngineNumber.."/reverseOn")

registerCommandHandler(reverseOnCom
mand, 0, reverseOn_Handler)
  reverseOffCommand =
findCommand(COMMANDS.."/engines/eng
ine"..pEngineNumber.."/reverseOff")

registerCommandHandler(reverseOffCo
mmand, 0, reverseOff_Handler)
  -- struzhkaVMasleCommand =
findCommand(COMMANDS.."/engines/eng
ine"..pEngineNumber.."/reverseOff")

  if (get(N2Array,pEngineNumber) >
55) then engineState =
EngineState.ON end

      _started = true
end
function prevValues()
      prevStopSwitch = iStopSwitch
      prevReverseSwitch =
iReverseSwitch
      prevReverseSwitchAll =
iReverseSwitchAll
      prevEngineStartInAirButton =
iEngineStartInAirButton
```

```
  prevStartedFailed =
iStarterFailed
  prevSuppostedGasTemperature =
oSuppostedGasTemperature
  prevSuppostedOilTemperature =
oSuppostedOilTemperature
  prevPodachaToplFailure =
iPodachaToplFailure
end

function readInputs()
      _dt = get(dt)
      iVoltage = get(_iVoltage)
      iThrotle = get(_iThrotle)
      iThrottleAll =
get(_iThrottleAll)
      iSeparateThrottle =
get(_iSeparateThrottle)
      iVibrationCheckButton =
get(_iVibrationCheckButton)
      iVibrationCheckRotator =
get(_iVibrationCheckRotator)
      iEngineStart =
get(_iEngineStart)
      iStopSwitch =
get(_iStopSwitch)
      iRodRaboty = get(_iRodRaboty)
      iReverseSwitch =
get(_iReverseSwitch)
      iReverseSwitchAll =
get(_iReverseSwitchAll)
      iSeparateReverse =
get(_iSeparateReverse)
      iVSU_RPM = get(_iVSU_RPM)
  iEngineStartInAirButton =
get(_iEngineStartInAirButton)
  iStarterFailed =
get(_iStarterFailed)
  iFuelFireFailed =
get(_iFuelFireFailed)
  iZabrosTVGFail =
get(_iZabrosTVGFail)
  iStruzhkaVMasle =
get(_iStruzhkaVMasle)
  iPovyshRashodMasla =
get(_iPovyshRashodMasla)
  iMinDavlTopliva =
get(_iMinDavlTopliva)
  iMinDavlMasla =
get(_iMinDavlMasla)
  iPodachaToplFailure =
get(_iPodachaToplFailure)
  iPodkachNasosFailure =
get(_iPodkachNasosFailure)
  iZamokReversaFailure =
get(_iZamokReversaFailure)
  iNasosFailure =
get(_iNasosFailure)
  iProizvVklReversa =
get(_iProizvVklReversa)
      iOpasnayaVibraciya =
get(_iOpasnayaVibraciya)
end


local testSwitch =
globalPropertyf(PHYSICAL.."/RUD_Swi
tch/discreteInput/ProverkaLamp")
local prevTestSwitch = 0
function update()
      readParameters()
      prevValues()
      if not _started then start()
end
```

```
      readInputs()

      oConsumption =
pIndicationPower / (iVoltage > 0
and iVoltage or pNormalVoltage)
      HasEnergy = iVoltage > pMinV

      TryStartEngines()
      TryStartEnginesInAir()
      TryFakeStartEngines()

  if (not manualControl) then
    AutoReverse()
  else
       SetThrottle()
       Reverse()
  end
      GetVibrations()
      StopEngine()

  SetActualPanelValues()
  SetNs()
  SetGasTemperature()
  SetOilRemaining()
  SetOilPressure()
  SetOilTemperature()
  SetFuelRemaining()
  SetFuelPressure()


      SetEnginePanelValues()
      SetIndicators()

  GetEngineStatus()

      setOutputs()
      writeOutputs()
```

```
      prevTestSwitch =
get(testSwitch)
      updateAll(components)
end

function TryStartEngines()
  if (forceQuickStart) then
      commandOnce(engage_starter)
      engineState =
EngineState.TurningON
      engineTurningOnTimer = 2
      quickStartTimer =
quickStartTimer + _dt
      if (quickStartTimer > 2) then
        quickStartTimer = 0
        forceQuickStart = false
        -- engineState =
EngineState.ON
        set(IntroTime,40)
      end
    return
  end
      if (iRodRaboty == 2) then
            if (iEngineStart == 1
and iStopSwitch == 0 and iVSU_RPM >
pMin_VSU_RPM) then
      engineState =
EngineState.TurningON
      engineTurningOnTimer =
pEngineTurningOnTime

      commandOnce(engage_starter)
                  oPanelZapuskaRab
= 1

      panelZapuskaRabTimer =
iStarterFailed == 1 and 56 or 30
            end
```

```
      else
            oPanelZapuskaRab = 0
      end

      if (oPanelZapuskaRab == 1)
then
            panelZapuskaRabTimer =
panelZapuskaRabTimer - _dt
            if
(panelZapuskaRabTimer < 0) then
                  oPanelZapuskaRab
= 0
      if (iFuelFireFailed == 1)
then
        shut_down =
findCommand("sim/starters/shut_down
_"..pEngineNumber)
        commandOnce(shut_down)
        engineTurningOnTimer = 0
        engineState =
EngineState.TurningOff
      end
            end
      end
end

function TryStartEnginesInAir()
      if (iEngineStartInAirButton
== 1 and iStopSwitch == 1 and
oSuppostedN2 > 3) then

      commandOnce(engage_starter)
    engineState =
EngineState.TurningON
    engineTurningOnTimer =
pEngineTurningOnTime
            startInAirTimer =
startInAirTimer - _dt
            if (startInAirTimer <=
0) then
                  startInAirStarted
= true
                  startInAirTimer =
2
            end
      else
            startInAirTimer = 2
      end
      if (startInAirStarted) then
            startInAirFailTimer =
startInAirFailTimer - _dt
            if (startInAirFailTimer
<= 0) then
                  if (iStopSwitch
== 1) then

      commandOnce(shut_down)
        engineTurningOnTimer = 0
        engineState =
EngineState.TurningOff
                  end

      startInAirFailTimer = 10
                  startInAirStarted
= false
            end
      else
            startInAirFailTimer =
10
      end
end

function TryFakeStartEngines()

end
```

```
function SetThrottle()
  if (forcedThrottleOn) then

set(throttleArray,get(forcedThrottl
e),pEngineNumber)
  else
       if (iSeparateThrottle == 1)
then

      set(throttleArray,iThrotle,pE
ngineNumber)
       else

      set(throttleArray,iThrottleAl
l,pEngineNumber)
       end
  end
  --
set(throttleArray,0.7538,pEngineNum
ber)
end

function Reverse()
            if
(get(throttleArray,pEngineNumber) <
0.05) then
                  if
(iReverseSwitch == 1 and
prevReverseSwitch == 0) then

      commandBegin(hold_reverse)

      oReverseOnIndication = 1
                  end
                  if
(iReverseSwitch == 0 and
prevReverseSwitch == 1) then

      commandEnd(hold_reverse)

      oReverseOnIndication = 0
                  end
            end
end

function AutoReverse()
  if (reverseOnInProgress) then
    if (autoReverseOnStage ==
AutoReverseStage.Off) then

set(throttleArray,0.0,pEngineNumber
)
      autoReverseOnStage =
AutoReverseStage.GoingDown
    end
    if (autoReverseOnStage ==
AutoReverseStage.GoingDown) then
      if (oSuppostedN1 < 35) then
        autoReverseOnStage =
AutoReverseStage.Toggle
      end
    end
    if (autoReverseOnStage ==
AutoReverseStage.Toggle) then

commandOnce(reverse_toggle_command)
      autoReverseOnStage =
AutoReverseStage.GoingUp
    end
    if (autoReverseOnStage ==
AutoReverseStage.GoingUp) then

set(throttleArray,get(forcedReverse
Throttle),pEngineNumber)
```

```
      autoReverseOnStage =
AutoReverseStage.Finished
    end
    if (autoReverseOnStage ==
AutoReverseStage.Finished) then
      manualControl = true
      reverseOnInProgress = false
      autoReverseOnStage =
AutoReverseStage.Off
    end
  end

  if (reverseOffInProgress) then
    if (autoReverseOffStage ==
AutoReverseStage.Off) then

set(throttleArray,0.0,pEngineNumber
)
      autoReverseOffStage =
AutoReverseStage.GoingDown
    end
    if (autoReverseOffStage ==
AutoReverseStage.GoingDown) then
      if (oSuppostedN1 < 35) then
        autoReverseOffStage =
AutoReverseStage.Toggle
      end
    end
    if (autoReverseOffStage ==
AutoReverseStage.Toggle) then

commandOnce(reverse_toggle_command)
      autoReverseOffStage =
AutoReverseStage.GoingUp
    end
    if (autoReverseOffStage ==
AutoReverseStage.GoingUp) then

set(throttleArray,get(forcedReverse
Throttle),pEngineNumber)
      autoReverseOffStage =
AutoReverseStage.Finished
    end
    if (autoReverseOffStage ==
AutoReverseStage.Finished) then
      manualControl = true
      reverseOffInProgress = false
      autoReverseOffStage =
AutoReverseStage.Off
    end
  end

end

function GetVibrations()
  if (iOpasnayaVibraciya == 1) then
    oVibrationFWD =
pvoUtils.toValue(oVibrationFWD, 90,
10, _dt)
    oVibrationBACK =
pvoUtils.toValue(oVibrationBACK,
90, 10, _dt)
  else
       oVibrationFWD =
pvoUtils.toValue(oVibrationFWD,
oSuppostedN1 / 10, 1000, _dt)
    oVibrationBACK =
pvoUtils.toValue(oVibrationBACK,
oSuppostedN2 / 10, 1000, _dt)
  end
     if (iVibrationCheckButton ==
1 and
math.floor((iVibrationCheckRotator
+ 1) / 2) == pEngineNumber) then
            oVibrationFWD = 80
```

```
            oVibrationBACK = 70
      end

      if (oVibrationFWD >
pHighVibrationValue or
oVibrationBACK >
pHighVibrationValue) then
                  oHighVibration =
1
            else
                  oHighVibration =
0
            end

  if (oVibrationFWD >
pDangerVibrationValue or
oVibrationBACK >
pDangerVibrationValue) then
      oDangerVibration = 1
    else
      oDangerVibration = 0
    end
end



function StopEngine()
      if (iStopSwitch == 1 and
prevStopSwitch == 0) then
            shut_down =
findCommand("sim/starters/shut_down
_"..pEngineNumber)
            commandOnce(shut_down)
            oPanelZapuskaRab = 0
    engineTurningOnTimer = 0
    engineState =
EngineState.TurningOff
      end
end

local gasTempMapFrom =
{374,678,724,743,787}
local gasTempMapTo =
{374,460,515,545,620}

function MapGasTemp(from)
      if (from <=
gasTempMapFrom[1]) then return from
end
      local result = 0

      for i=2,5 do
            if (from <=
gasTempMapFrom[i]) then
                  result = ((from -
gasTempMapFrom[i-1]) /
(gasTempMapFrom[i] -
gasTempMapFrom[i-
1]))*(gasTempMapTo[i] -
gasTempMapTo[i-1]) +
gasTempMapTo[i-1]
                  break
            end
      if (result == 0) then result
= from / 1.26 end
      end
      return result
end

local N1MapFrom =
{0,30,72.8,81.3,85.1,93.5,110}
local N1MapTo =
{0,30,72.5,80,83.5,91,110}

function MapN1(from)
```

```
	if (from <= N1MapFrom[1])
then return from end
	local result = 0

	for i=2,7 do
		if (from <=
N1MapFrom[i]) then
			result = ((from -
N1MapFrom[i-1]) / (N1MapFrom[i] -
N1MapFrom[i-1]))*(N1MapTo[i] -
N1MapTo[i-1]) + N1MapTo[i-1]
			break
		end
	if (result == 0) then result
= from end
	end
	return result
end

local N2MapFrom = {0,   60.3  ,87.5
		,91.7 ,93.4 ,97.5 ,110}
local N2MapTo =   {0,   60
	,87.25	,91		,93
		,97		,110}

function MapN2(from)
	if (from <= N2MapFrom[1])
then return from end
	local result = 0

	for i=2,7 do
		if (from <=
N2MapFrom[i]) then
			result = ((from -
N2MapFrom[i-1]) / (N2MapFrom[i] -
N2MapFrom[i-1]))*(N2MapTo[i] -
N2MapTo[i-1]) + N2MapTo[i-1]
			break
		end
	if (result == 0) then result
= from end
	end
	return result
end

function SetActualPanelValues()
    oActualN1 =
get(N1Array,pEngineNumber)
    oActualN2 =
get(N2Array,pEngineNumber)
    oActualFuelPressure =
get(ActualFuelPreasureArray,pEngine
Number)
    oActualOilPressure =
get(ActualOilPreasureArray,pEngineN
umber)
    oActualOilTemperature =
get(ActualOilTemperatureArray,pEngi
neNumber)
    oActualFuelConsumption =
get(ActualFuelConsumptionArray,pEng
ineNumber) * 3600
    oActualFuelRemaining =
oActualFuelRemaining >= 0 and
oActualFuelRemaining - _dt *
oActualFuelConsumption / 3600 /
1000 or 0
    oActualGasTemperature =
get(ActualGasTemperatureArray,pEngi
neNumber)
end

function SetEnginePanelValues()
```

```
    if (iPodachaToplFailure == 1)
then
      oSuppostedFuelConsumption =
pvoUtils.toValue(oSuppostedFuelCons
umption,50,100,_dt)
      if (prevPodachaToplFailure ==
0) then
        shut_down =
findCommand("sim/starters/shut_down
_"..pEngineNumber)
        commandOnce(shut_down)
        engineTurningOnTimer = 0
        engineState =
EngineState.TurningOff
      end
    else
      oSuppostedFuelConsumption =
get(ActualFuelConsumptionArray,pEng
ineNumber) * 3600
    end

end

function SetNs()
  oSuppostedN1 = MapN1(oActualN1)
  oSuppostedN2 = MapN2(oActualN2)
  if (iFuelFireFailed == 1) then
    if (oSuppostedN2 > 17) then
      oSuppostedN2 = 17
      oSuppostedN1 =
savedN1whenN2is17
    else
      savedN1whenN2is17 =
oSuppostedN1
    end
  end
end
function SetGasTemperature()
  if (iFuelFireFailed == 1) then
    oSuppostedGasTemperature =
prevSuppostedGasTemperature
  else
    if (iZabrosTVGFail == 1 and
engineState ==
EngineState.TurningON) then
      if (gasTempReached800 == 0)
then
        oSuppostedGasTemperature =
pvoUtils.toValue(oSuppostedGasTempe
rature, 900, 80, _dt)
      else
        oSuppostedGasTemperature =
pvoUtils.toValue(oSuppostedGasTempe
rature,
MapGasTemp(oActualGasTemperature),
40, _dt)
      end
      if (oSuppostedGasTemperature
> 800) then
        gasTempReached800 = 1
      end
      if (oSuppostedN2 < 10) then
        gasTempReached800 = 0
      end
    else
      gasTempReached800 = 0
      oSuppostedGasTemperature =
pvoUtils.toValue(oSuppostedGasTempe
rature,
MapGasTemp(oActualGasTemperature),
40, _dt)
    end
  end
end

function SetOilRemaining()
```

```
  if (iPovyshRashodMasla == 1) then
    oSuppostedOilRemaining =
pvoUtils.toValue(oSuppostedOilRemai
ning, 0, 2, _dt)
  else
    oSuppostedOilRemaining =
pvoUtils.toValue(oSuppostedOilRemai
ning, pOilAmount, 25, _dt)
  end
end

function SetOilPressure()
  if (iMinDavlMasla == 1) then
    oSuppostedOilPressure =
pvoUtils.toValue(oSuppostedOilPress
ure, 1.5, 0.5, _dt)
  else
    if (oSuppostedOilRemaining < 5)
then
      oSuppostedOilPressure =
pvoUtils.toValue(oSuppostedOilPress
ure, 0, 0.5, _dt)
    else
      oSuppostedOilPressure =
get(ActualOilPreasureArray,pEngineN
umber)
    end
  end
end

function SetOilTemperature()
  if (iFuelFireFailed == 1) then
    oSuppostedOilTemperature =
prevSuppostedOilTemperature
  else
    oSuppostedOilTemperature =
get(ActualOilTemperatureArray,pEngi
neNumber)
  end
end

function SetFuelRemaining()
  oSuppostedFuelRemaining =
oSuppostedFuelRemaining > 0 and
oSuppostedFuelRemaining - _dt *
oSuppostedFuelConsumption / 3600
/1000 or 0
end

function SetFuelPressure()
  if (iMinDavlTopliva == 1) then
    oSuppostedFuelPressure =
pvoUtils.toValue(oSuppostedFuelPres
sure,0, 5,_dt)
  end
  if (iPodachaToplFailure == 1)
then
    oSuppostedFuelPressure = 0
  end
  if (iMinDavlTopliva == 0 and
iPodachaToplFailure == 0) then
    local psi_to_kg =
get(ActualFuelPreasureArray,pEngine
Number) * 0.70307
    oSuppostedFuelPressure =
pvoUtils.toValue(oSuppostedFuelPres
sure,psi_to_kg, 20,_dt)
  end
end

function SetIndicators()
      if (HasEnergy) then
    oReverseOnIndication =
get(reverserOnArray,pEngineNumber)
```

```
            oMinDavlMaslaIndicator
= oSuppostedOilPressure < 2.2 and 1
or 0
            oMinDavlToplIndicator =
oSuppostedFuelPressure < 1.5 and 1
or 0

      oPerezapuskVozdOtkrIndicator
= oSuppostedN2 < 13 and 0 or 1
            oVNARabIndicator = 1
            oVNAPuskIndicator = 1


    if (iStarterFailed == 1 and
oSuppostedN2 >= 44) then
      oOpasOborotyStatera = 1
    end
    if (iStarterFailed == 0 and
prevStartedFailed == 1) then
      oOpasOborotyStatera = 0
    end

    oZamRevOtkrytIndicator =
iZamokReversaFailure
    oStruzhkaVMasleIndicator =
iStruzhkaVMasle
    oOpasnayaVibraciyaIndicator =
(oVibrationFWD > 89 or
oVibrationBACK > 89) and 1 or 0
    oMinOilRemaining =
oSuppostedOilRemaining < 2.2 and 1
or 0
      -- if (pEngineNumber == 2)
then oStruzhkaVMasleIndicator = 1
end
-- if (pEngineNumber == 4) then
oOpasnayaVibraciyaIndicator = 1 end
      else
      oOpasnayaVibraciyaIndicator =
0
            oMinDavlToplIndicator =
0

      oStruzhkaVMasleIndicator = 0
            oOstTop2000Indicator =
0
            oMinDavlMaslaIndicator
= 0

      oToplFiltrNeRabIndicator = 0

      oPerezapuskVozdOtkrIndicator
= 0
            oVNARabIndicator = 0
            oZamRevOtkrytIndicator
= 0
            oVNAPuskIndicator = 0
    oMinOilRemaining = 0
      end

end

function GetEngineStatus()
  if (engineTurningOnTimer > 0)
then
    engineTurningOnTimer =
engineTurningOnTimer - _dt
    if (engineTurningOnTimer <= 0)
then
      engineTurningOnTimer = 0
      engineState = EngineState.ON
    end
  end
  if (oSuppostedN2 == 0) then
    engineState = EngineState.OFF
```

```
  end
end


function setOutputs()
      oEngineRPM_Percent =
oSuppostedN2 / 100
  oEngineState = engineState
end

function writeOutputs()
  set(_oEngineState,oEngineState)
      set(_oVibrationFWD,oVibration
FWD)
      set(_oVibrationBACK,oVibratio
nBACK)

set(_oHighVibration,oHighVibration)
      set(_oDangerVibration,oDanger
Vibration)
      set(_oPanelZapuskaRab,oPanelZ
apuskaRab)
      set(_oReverseOnIndication,oRe
verseOnIndication)

set(_oEngineRPM_Percent,oEngineRPM_
Percent)

set(_oOpasOborotyStatera,oOpasOboro
tyStatera)
      set(_oMinOilRemaining,oMinOil
Remaining)

      set(_oOpasnayaVibraciyaIndica
tor,oOpasnayaVibraciyaIndicator)
      set(_oMinDavlToplIndicator,oM
inDavlToplIndicator)
      set(_oStruzhkaVMasleIndicator
,oStruzhkaVMasleIndicator)
      set(_oOstTop2000Indicator,oOs
tTop2000Indicator)
      set(_oMinDavlMaslaIndicator,o
MinDavlMaslaIndicator)
      set(_oToplFiltrNeRabIndicator
,oToplFiltrNeRabIndicator)
      set(_oPerezapuskVozdOtkrIndic
ator,oPerezapuskVozdOtkrIndicator)
      set(_oVNARabIndicator,oVNARab
Indicator)
      set(_oZamRevOtkrytIndicator,o
ZamRevOtkrytIndicator)
      set(_oVNAPuskIndicator,oVNAPu
skIndicator)

      set(_oActualN1,oActualN1)
      set(_oActualN2,oActualN2)
      set(_oSuppostedN1,oSuppostedN
1)
      set(_oSuppostedN2,oSuppostedN
2)
      set(_oActualFuelPressure,oAct
ualFuelPressure)
      set(_oSuppostedFuelPressure,o
SuppostedFuelPressure)
      set(_oActualOilPressure,oActu
alOilPressure)
      set(_oSuppostedOilPressure,oS
uppostedOilPressure)
      set(_oActualOilTemperature,oA
ctualOilTemperature)
      set(_oSuppostedOilTemperature
,oSuppostedOilTemperature)
      set(_oActualFuelConsumption,o
ActualFuelConsumption)
```

```
      set(_oSuppostedFuelConsumptio
n,oSuppostedFuelConsumption)
      set(_oActualFuelRemaining,oAc
tualFuelRemaining)
      set(_oSuppostedFuelRemaining,
oSuppostedFuelRemaining)
      set(_oActualGasTemperature,oA
ctualGasTemperature)

set(_oSuppostedGasTemperature,oSupp
ostedGasTemperature)

set(_oSuppostedOilRemaining,oSuppos
tedOilRemaining)
      set(_oSuppostedFuelPressure,o
SuppostedFuelPressure)
end

function
settingThrottle_Handler(phase)
  if (phase == SASL_COMMAND_BEGIN)
then
    forcedThrottleOn = true
  end
  if (phase == SASL_COMMAND_END)
then
    forcedThrottleOn = false
  end
end

function stopEngine_Handler(phase)
  shut_down =
findCommand("sim/starters/shut_down
_"..pEngineNumber)
  commandOnce(shut_down)
  oPanelZapuskaRab = 0
  engineTurningOnTimer = 0
  engineState =
EngineState.TurningOff
  return 1
end

function quickStart_Handler(phase)
  set(IntroTime,1)
  forceQuickStart = true
  return 1
end

function reverseOn_Handler(phase)
  if (phase == SASL_COMMAND_BEGIN
and
get(reverserOnArray,pEngineNumber)
== 0) then
    manualControl = false
    reverseOnInProgress = true
    reverseOffInProgress = false
  end
end

function reverseOff_Handler(phase)
  if (phase == SASL_COMMAND_BEGIN
and
get(reverserOnArray,pEngineNumber)
== 1) then
    manualControl = false
    reverseOffInProgress = true
    reverseOnInProgress = false
  end
end


function
StruzhkaVMasle_Handler(phase)
      if (phase ==
SASL_COMMAND_BEGIN) then
```

```
      oIndicatorStruzhkaVMasle = 1
      end

      if (phase ==
SASL_COMMAND_END) then

      oIndicatorStruzhkaVMasle = 0
      end
end

components = {}
```