МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

_____ Олександр ЛИТВИНЕНКО

"_____"_____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмні та фізичні засоби виявлення конфліктів траєкторій руху повітряних суден

Виконавець: Євгеній КОЗЬМІН

Керівник: Дмитро КУЧЕРОВ

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо–професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Олександр ЛИТВИНЕНКО
«_____»_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Козьміна Євгенія Вікторовича

1. Тема кваліфікаційної роботи: «Програмні та фізичні засоби виявлення конфліктів траєкторій руху повітряних суден»

затверджена наказом ректора від «28» серпня 2023 р. № 1494/ст

2. Термін виконання роботи: з 2 жовтня 2023 р. по 31 грудня 2023 р.

3. Вхідні дані до роботи: документація, тестові дані, програмні продукти,

4. Зміст пояснювальної записки: вступ, визначення проблеми та обґрунтування актуальності, аналіз існуючих методів виявлення конфліктів траєкторій руху повітряних суден, аналіз вимог програмного забезпечення, розробка програмного забезпечення, висновки.

5. Перелік обов'язкового графічного (ілюстрованого) матеріалу:

1) ієрархія нефункціональних вимог;

2) діаграма потоків даних нульового рівня;

3) діаграма *Use Case*;

4) загальна структура застосунку;

5) головна сторінка застосунку.

6. Календарний план

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Проаналізувати наявні методи побудови маршрутів повітряних суден та та виявлення конфліктів на них	02.10.2023 – 10.10.2023	
2	Розділ 1: Визначення проблеми та обґрунтування актуальності	11.10.2023 – 01.11.2023	
3	Розділ 2: Аналіз існуючих методів виявлення конфліктів траєкторій руху повітряних суден	02.11.2023 – 13.11.2023	
4	Розділ 3: Аналіз вимог програмного забезпечення	14.11.2023 – 30.11.2023	
5	Розділ 4: Розробка програмного забезпечення	01.12.2023 – 12.12.2023	
6	Оформлення пояснювальної записки	13.12.2023 – 17.12.2023	
7	Розробка презентації для захисту роботи та підготовка до захисту	18.12.2023 – 31.12.2023	

7. Дата видачі завдання: « 02 » жовтня 2023 р.

Керівник кваліфікаційної роботи _____ Дмитро КУЧЕРОВ

Завдання прийняв до виконання _____ Євгеній КОЗЬМІН

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмні та фізичні засоби виявлення конфліктів траєкторій руху повітряних суден»: 80 сторінок, 31 рисунок, 1 таблиця, 33 використаних джерела .

ЕФЕКТИВНІСТЬ ВИКОРИСТАННЯ ЛІТАКІВ, УПРАВЛІННЯ ПОВІТРЯНИМ РУХОМ, ПРОГРАМНИЙ МОДУЛЬ.

Об’єкт дослідження кваліфікаційної роботи – програмний модуль.

Предмет дослідження кваліфікаційної роботи – алгоритми та програмні рішення для виявлення та управління конфліктами траєкторій в умовах зростаючого трафіку та різноманіття повітряних суден.

Мета кваліфікаційної роботи – розробка та впровадження систем виявлення та управління конфліктами траєкторій руху повітряних суден.

Методи дослідження – аналіз вимог, тестування, дослідження алгоритмів.

В межах кваліфікаційного проєкту проведено комплексний аналіз існуючих фізичних та програмних засобів виявлення конфліктів руху повітряних засобів з метою підвищення безпеки авіаційних процесів. В роботі розглянуті різноманітні технології та методи, включаючи радарні системи, системи інерційної навігації, алгоритми штучного інтелекту та машинного навчання.

Здійснено аналіз переваг та недоліків кожного типу засобу виявлення конфліктів руху, враховуючи їхню ефективність у різних умовах та сценаріях авіаційних операцій. Досліджено процес виявлення та управління конфліктами руху з метою визначення оптимальних підходів до підвищення безпеки та ефективності повітряного руху.

Додатково, проведено огляд існуючих програмних рішень для виявлення конфліктів руху та реалізовано власні програмні компоненти, спрямовані на мінімізацію та управління конфліктами руху повітряних засобів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ADS-B	–	*Automatic Dependent Surveillance-Broadcast*
API	–	*Application Programming Interface*
ICAO	–	*International Civil Aviation Organization*
MLAT	–	*Multilateration*
EASA	–	*European Aviation Safety Agency*
БД	–	база даних
ІТ	–	інформаційні технології
КІ	–	комп’ютерна інженерія
КПР	–	керування повітряним рухом
ОПР	–	обслуговування поповітрянного руху
ПЗ	–	програмний засіб
ПС	–	повітряне судно
СП	–	системне програмування
СШІ	–	системи штучного інтелекту

ВСТУП

Здавалося б, небеса – нескінченний простір вільності, але зі зростанням обсягів повітряного руху стає важливим завданням забезпечення безпеки та координації руху повітряних суден. Сучасна авіаційна система стикається з викликами, пов'язаними з технічними новаціями та різноманітністю аерокосмічних платформ.

В авіації використовуються різноманітні системи для забезпечення безпеки, навігації, комунікації та інших аспектів польоту.

Авіоніка (*Avionics*). Представляє собою електронні системи на борту літака, включаючи системи навігації, комунікації, контролю польоту та вимірювальні прилади.

Система управління польотом (*Flight Control System*). Керує керуванням повітряним судном, включаючи автопілот, системи стабілізації та системи аварійного управління.

Система навігації (*Navigation System*). Забезпечує точне визначення місцезнаходження літака в просторі, використовуючи супутникові системи, радіолокацію та інші технології.

Система управління двигуном (*Engine Control System*). Моніторить та керує роботою літакових двигунів для забезпечення оптимальної ефективності та безпеки.

Система термоконтролю (*Thermal Management System*). Контролює та регулює температуру різних систем та компонентів літака для запобігання перегріву чи недостатнього охолодження.

Система аварійного сповіщення (*Emergency Locator Transmitter – ELT*). Автоматично виводить сигнал аварійного сповіщення у випадку аварії, допомагаючи рятувальним службам визначити місцезнаходження літака.

Система термовізійного спостереження (*Forward Looking Infrared – FLIR*). Дозволяє пілотам бачити теплове випромінювання об'єктів, що полегшує виявлення та уникнення небезпеки.

Система контролю обтікання (*Anti-Ice System*). Забезпечує захист від обмерзання або накипу на критичних частинах поверхні літака.

Цей кваліфікаційний проект націлений на вивчення та розвиток систем виявлення конфліктів траєкторій руху повітряних суден, щоб відповісти на розмаїття сучасних викликів авіаційного простору. Зростання авіаційної діяльності, впровадження безпілотних літальних апаратів та розвиток розширених систем управління повітряним рухом вимагають нових, високоефективних рішень для забезпечення безпеки та надійності.

Авіація, як важливий елемент сучасного світового транспорту, потребує не лише технологічних, але й програмних засобів для виявлення та управління конфліктами траєкторій. Дослідження, яке впроваджується у цьому проекті, спрямоване на розуміння складних взаємодій у повітряному просторі та створення ефективних засобів для протидії можливим конфліктам.

Проект орієнтований на створення автоматизованих систем управління повітряним рухом, які базуються на аналізі та передбаченні траекторій руху повітряних суден. Велика увага приділяється точності та швидкості виявлення конфліктів, щоб забезпечити ефективне управління та надійність авіаційних процесів.

РОЗДІЛ 1
ВИЗНАЧЕННЯ ПРОБЛЕМИ ТА ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ

1.1. Визначення проблеми

Логістика вивчає оптимальне управління матеріальними та іншими взаємопов'язаними потоками у будь-якій сфері людської діяльності. У сучасному світі логістика стала однією з ключових галузей як у науці, так і в практиці, охоплюючи не лише економіку, але й політику, містобудування, інженерію, екологію та інші області. Основна мета логістики – оптимізувати функціонування систем, що базуються на різних потоках. Ці потоки можуть включати традиційні розуміння слова "потоки", такі як нафтопроводи і газопроводи, а також транспортні, виробничі та інформаційні потоки в мережах.

В сучасному повітряному просторі актуальність питань безпеки та управління рухом повітряних суден стає вельми критичною. Спостерігається значне зростання обсягів авіаційного трафіку, що породжує неабиякі труднощі у виявленні та управлінні конфліктами траєкторій руху.

Проблема полягає в тому, що існуючі системи не завжди в змозі ефективно враховувати різноманітні фактори, такі як збільшення обсягів трафіку, різноманіття літальних апаратів, включаючи безпілотні, та стрімке впровадження новітніх технологій. Крім того, традиційні системи контролю та управління можуть виявитися неефективними в умовах навантаження, що може призвести до збільшення ризику аварій та зіткнень у повітрі.

Також важливо враховувати різні види повітряних суден, включаючи безпілотні дрони, які стають все більш поширеними в цивільному та комерційному використанні. Це створює новий рівень складності, оскільки такі судна можуть мати інші характеристики та стратегії руху, порівняно з традиційними літаками.

Основна проблема визначається необхідністю створення та впровадження систем, які забезпечать високопродуктивне виявлення та управління конфліктами

траєкторій в умовах різноманітності та динаміки повітряного простору. Успішне вирішення цієї проблеми має величезне значення для підвищення безпеки, надійності та ефективності авіаційного руху.

Процес автоматизації та оптимізації управління повітряним рухом стає надзвичайно важливим у контексті швидкого технологічного розвитку та збільшення обсягів авіаційного трафіку. Однак існуючі системи, незважаючи на свою ефективність, можуть не досить ефективно враховувати велику кількість даних та динамічні зміни в авіаційному просторі.

Додатковою проблемою є використання безпілотних літальних апаратів (БПЛА), які стають все більш поширеними в різних сферах, включаючи комерційний та приватний сектори. Безпілотники мають відмінні характеристики руху та взаємодії, і їх впровадження потребує нових стратегій управління та виявлення конфліктів.

Зазначена проблема обумовлює необхідність створення інтегрованих систем, які здатні працювати в умовах складного та динамічного середовища, забезпечуючи ефективне виявлення та управління конфліктами траєкторій. Це включає в себе розробку нових алгоритмів, технологічних рішень та стратегій управління для врахування всієї різноманітності літальних апаратів та умов їхнього руху.

Враховуючи вищезазначені аспекти, визначення проблеми полягає в потребі в створенні інноваційних систем, що враховують технологічні виклики та різноманітність повітряного руху, забезпечуючи при цьому високий рівень безпеки та ефективності управління авіаційним трафіком.

1.2. Керування повітряним рухом

Керування повітряним рухом (КПР) є важливою частиною обслуговування повітряного руху (ОПР), включаючи взаємодію між авіадиспетчерами, екіпажами повітряних суден та іншими службами, такими як метеорологічні, технічні та аеродромні. Головні завдання КПР включають запобігання зіткненням повітряних суден, підтримку впорядкованого потоку руху, надання інформації та допомоги в

аварійних ситуаціях, а також видачу необхідної інформації, такої як метеорологічна та радіотехнічна. Головні завдання КПР включають у себе:

– Запобігання зіткнення повітряних суден в повітрі, на майданчику та на злітно-посадковій смузі, а також уникнення зіткнень повітряних суден з перешкодами.

– Прискорення та підтримка впорядкованого потоку повітряного руху.

– Надання необхідної інформації та допомога ЕПС в аварійних ситуаціях.

– Видача іншої необхідної інформації, такої як метеорологічна та радіотехнічна.

Першочергове завдання КПР, яке включає запобігання зіткненням повітряних суден, представляє найбільшу небезпеку. Нескоординована дія складових системи управління повітряним рухом, включаючи електроніку та технічне непорозуміння, може призводити до авіакатастроф зі значними наслідками для безпеки та навколишнього середовища.

Авіадиспетчери, використовуючи різні засоби спостереження та технічні засоби, постійно моніторять повітряну обстановку, виявляють потенційно конфліктні ситуації та вирішують їх за допомогою різних методів, таких як зміна висоти, курсу, маршруту чи швидкості повітряних суден.

Несприятливі аспекти сучасної системи повітряного транспорту, такі як зростання обсягів повітряних перевезень, призводять до досягнення максимальних можливостей системи та збільшують ризик аварій. Збільшення інтенсивності повітряного руху призводить до збільшення кількості конфліктних ситуацій, що впливає на безпеку та ефективність польотів. Проте існуючі системи УПР не завжди можуть надійно контролювати польоти на малих висотах та в регіонах, де спостереження ускладнене.

1.3. Типи конфліктів в повітряному просторі

Протягом тривалого періоду вивчення всіх авіаційних інцидентів стало очевидним, що більшість них виникає через проблеми в управлінні повітряним рухом (УПР). Іншими словами, більшість неприємностей у повітрі спричинені конфліктними ситуаціями між літаками, які виникають через зіткнення різних обставин і дрібних помилок, що створюють небезпечні умови для багатьох життів.

Давайте розглянемо більш детально, що таке конфліктна ситуація і потенційна конфліктна ситуація (ПСС), розрізняючи ці терміни, та визначимо, як відбувається пошук ПСС та яка роль диспетчера у цьому процесі.

Авіадиспетчер – це ключова професія, яка забезпечує безпеку в повітрі. Диспетчер контролює ешелонування літаків, виявляє та вирішує конфлікти в повітрі. Це вимагає від нього високих розумових навичок, здатності працювати в команді та швидкого прийняття рішень в стресових ситуаціях.

Робота диспетчера полягає у контролі відстані між літаками, униканні їх зближення та, у випадку можливого конфлікту, у вирішенні ситуації з максимальною безпекою та ефективністю. Він також відповідає за пошук, визначення та усунення потенційних конфліктних ситуацій.

Конфліктна ситуація – це передбачуване зближення між літаками при порушенні ешелонування, яке може призвести до небезпечної ситуації. Пошук конфліктних ситуацій включає обчислення та моніторинг траєкторій польотів для виявлення можливих конфліктів.

Усунення конфліктної ситуації вимагає вибору оптимальних траєкторій розходження конфліктуючих літаків для максимальної безпеки та швидкості.

Авіадиспетчери виконують основну роботу виявлення та усунення потенційних конфліктів у відповідальних зонах. Розуміння авіадиспетчерами природи конфліктних ситуацій дає змогу ефективно впоратись із складними ситуаціями, що може виникнути в повітрі. Між різними типами конфліктів можна визначити основні:

– Коли об'єкти рухаються паралельно один одному, може відбутися зіткнення, якщо вони знаходяться на одному рівні і відстань між ними менша від установлених норм ешелонування. Такий конфлікт, де об'єкти рухаються паралельно, називається догоном.

Рис.1.1. Конфліктна ситуація під час руху ПС попутно

– Об'єкти рухаються у напрямку один до одного по одному маршруту. Це є ситуацією зустрічних об'єктів.

Рис.1.2. Конфліктна ситуація під час руху ПС назустріч

– Об'єкти рухаються маршрутами, які перетинаються, і це призводить до ситуації, коли повітряні судна можуть перетнутись на одній з висотних точок. Ця ситуація характеризується взаємодією об'єктів, які перетинаються.

Рис.1.3. Конфліктна ситуація під час руху ПС, що перетинаються

– Узагальнена ситуація, у якій беруть участь більше двох об’єктів.

1.4. Історичні приклади авіакатастроф через конфліктні ситуації у повітряному просторі

Авіакатастрофа над Гранд-Каньйоном. Подія відбулась у суботу, 30 червня 1956 року, та стала масштабною авіакатастрофою між літаками *United Air Lines (UAL)* Дуглас *DC*-7 (рейс *UA718*, маршрут Лос-Анджелес–Чикаго) та *Lockheed L-1049-54-80* Супер сузір'я *Trans World Airlines (TWA)* під номером *TWA002* (маршрут Лос-Анджелес, Канзас-Сіті). Обидва літаки, що перебували в повітрі над Гранд-Каньйоном в Арізоні, США, зазнали страшної події, яка відбулася на борту. Загальна кількість загиблих сягнула 128 осіб, включаючи 58 на *DC–7* (53 пасажирів і 5 членів екіпажу) та 70 на *L-1049* (64 пасажири і 6 членів екіпажу).

Ця трагедія, яка сталася 30 червня 1956 року, зафіксувала своєю жахливою статистикою низку подій. Зокрема, вона була першою авіаційною катастрофою в історії цивільної авіації, внаслідок якої загинуло понад 100 людей. Реакція на цю катастрофу була значущою, викликаючи великі зміни в організації управління повітряним рухом в країні.

Зіткнення над Нью–Йорком. Рано вранці 16 грудня 1960 року в Нью–Йорку сталася трагедія: два літаки, *Douglas DC-8-11* авіакомпанії *United Air Lines* і *Lockheed L-1049-54 Super Constellation* авіакомпанії *Trans World Airlines*, зіткнулися під час

польоту та розбилися в різних аеропортах міста. В результаті цієї події загинули 134 людини, включаючи 6 осіб на землі, роблячи це наймасштабнішою авіаційною катастрофою світу того часу. ЗМІ назвали інцидент "катастрофою літака на схилі" через місце приземлення літака *United.*

У цій авіакатастрофі загалом загинуло 134 людини, що робить це зіткнення найбільшою авіакатастрофою в Нью-Йорку того часу та однією з найбільших у світі, за винятком події над Гранд-Каньйоном за чотири роки до цього. Важливо зауважити, що проблеми повітряних конфліктів залишаються актуальними навіть зараз, незважаючи на технологічний прогрес та покращену підготовку фахівців.

Інцидент над Боденським Озером. Катастрофа, яка відбулася 1 липня 2002 року над Боденським озером, стала великою авіакатастрофою. У німецькому повітрі поблизу Уберлінгена та Боденського озера турецькі літаки "Башкирські авіалінії" Ту–154М (рейс *BTC 2937* Москва-Барселона) та вантажний літак *Boeing 757–200PF* від *DHL* (*DHX 611* Мухарру-Берга) зіткнулися. Всі 71 осіб, які були на борту обох літаків, загинули, включаючи 2 пілотів *Boeing 757* та 69 людей на Ту–154 (9 членів екіпажу та 60 пасажирів, серед яких 52 діти).

Ту–154М був виготовлений Авіаційним виробничим об'єднанням ім. Він здавався в оренду *Transeuropean Airlines* та *Shaheen Air International. Boeing 75-200PF* належав *DHL* і втратив контроль після зіткнення, розбившись через деякий час.

Журнал польотів показав, що приватна швейцарська компанія *Skyguide* в Цюріху контролювала повітряний рух, і трагедія виникла через непорозуміння між диспетчерами та несправності обладнання. Зіткнення сталося на висоті 10 634 метри, і обидва літаки зазнали непоправних пошкоджень, що призвело до загибелі всіх на борту.

Ця подія відобразила серйозні проблеми у координації та обладнанні систем повітряного руху, визначивши недоліки у роботі диспетчерського центру та системи попередження про зіткнення.

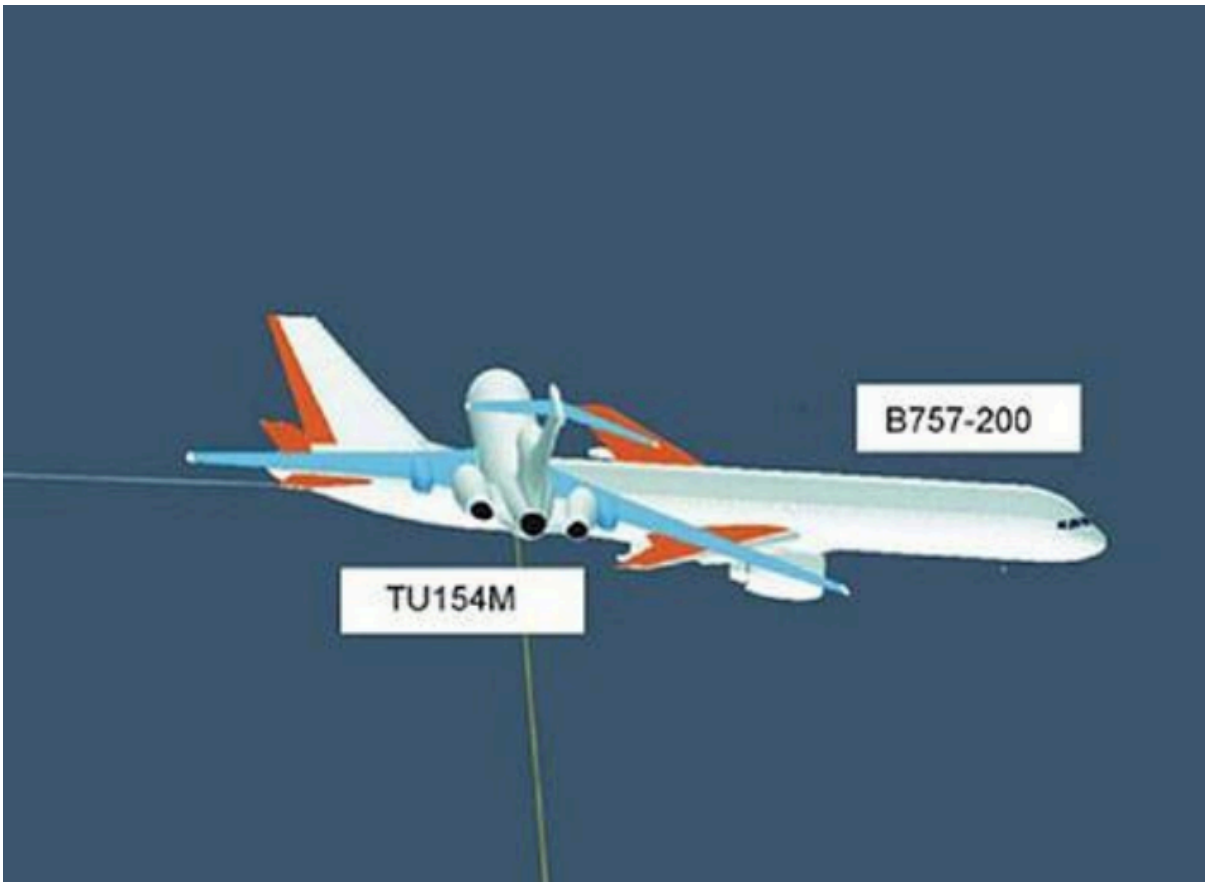


Рис.1.4. Симуляція зіткнення

1.5. Мета та завдання дослідження

Основна мета проекту. розробка та впровадження систем виявлення та управління конфліктами траєкторій руху повітряних суден. Ця мета виникає із визнання необхідності адаптації авіаційних технологій до сучасних викликів, таких як збільшення обсягів повітряного трафіку, різноманіття літальних апаратів, а також необхідності ефективного управління конфліктами траєкторій у складних умовах повітряного простору.

Цей проект націлений на створення систем, які будуть не лише надійними та безпечними, але й адаптивними до динамічних змін у сучасному авіаційному середовищі.

Посилення безпеки в повітряному просторі сприятиме розвитку авіаційної галузі та взаємній координації між різними учасниками авіаційного середовища.

Отже, головна мета проекту є створення ефективної системи виявлення та управління конфліктами траєкторій для покращення якості та безпеки авіаційного руху в сучасному світі. Завдання дослідження:

– Аналіз існуючих систем. Провести глибокий аналіз існуючих систем виявлення та управління конфліктами траєкторій, визначити їхні переваги та недоліки в контексті сучасних викликів авіації.

– Інтеграція технологій. Розробити стратегії інтеграції новітніх технологій, таких як штучний інтелект та машинне навчання, у системи виявлення та управління конфліктами траєкторій.

– Експериментальне дослідження. Провести експериментальне дослідження розроблених систем для оцінки їхньої ефективності, точності та відповідності вимогам сучасного повітряного руху.

– Рекомендації для впровадження. На основі отриманих результатів розробити конкретні рекомендації щодо впровадження нових систем виявлення та управління конфліктами траєкторій у повітряний транспорт.

1.6. Об'єкт та предмет дослідження

Об'єкт дослідження. Об'єктом дослідження є системи виявлення та управління конфліктами траєкторій руху повітряних суден в сучасному авіаційному середовищі.

Предмет дослідження. Предметом є дослідження існуючих алгоритмів, технічних рішень і стратегій, спрямованих на покращення функціональності систем виявлення та управління конфліктами траєкторій в умовах зростаючого трафіку та різноманіття літальних апаратів, зокрема безпілотних літальних апаратів (БПЛА), та розробка ПЗ для виявлення конфліктів траєкторій руху ПС.

Технічні аспекти. Розробка нових алгоритмів та технічних рішень для виявлення та управління конфліктами траєкторій. Адаптація систем до різноманітних літальних апаратів, включаючи традиційні літаки, вертольоти та БПЛА.

Технологічні аспекти. Використання новітніх технологій, таких як штучний інтелект, машинне навчання та аналіз даних, для покращення ефективності систем.

Безпека та координація. Розробка стратегій для забезпечення безпеки та ефективної координації між різними видами повітряних суден.

1.7. Висновки до розділу

У цьому розділі було проведено аналіз об'єкта та предмета нашого дослідження, розглянуті ключові аспекти управління повітряним рухом (УПР), а також визначено фактори, які визначають актуальність нашої наукової роботи. Здійснений аналіз вказує на необхідність подальшого розвитку систем УПР, особливо враховуючи постійне зростання обсягів повітряних перевезень та розширення флоту повітряних суден.

У процесі дослідження було виявлено, що сучасний стан системи повітряного транспорту характеризується стійким зростанням обсягів авіаперевезень. Це призводить до збільшення конфліктних ситуацій у повітряному просторі та підкреслює необхідність подальшого вдосконалення систем УПР. Зокрема, важливо адаптувати їх до нових викликів, пов'язаних із різноманіттям та технологічним розвитком повітряних суден. Висновки:

– Об'єкт та предмет дослідження. Об'єктом є системи УПР, а предметом – алгоритми та ПЗ для виявлення та управління конфліктами траєкторій в умовах зростаючого трафіку та різноманіття повітряних суден.

– Актуальність дослідження. Сучасний стан світової системи повітряного транспорту характеризується стійким збільшенням обсягів повітряних перевезень. Зростання інтенсивності повітряного руху призводить до збільшення кількості конфліктних ситуацій та потребує подальшого розвитку систем УПР для забезпечення безпеки та ефективності польотів.

– Виклики та перспективи. Суттєва частина досліджень у цьому напрямку пов'язана з розробкою нових технологій, використанням штучного інтелекту та машинного навчання для вдосконалення систем виявлення конфліктів. Подальший

розвиток систем УПР є ключовим для забезпечення безпеки та координації у зростаючому повітряному просторі.

РОЗДІЛ 2
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ВИЯВЛЕННЯ КОНФЛІКТІВ ТРАЄКТОРІЙ РУХУ ПОВІТРЯНИХ СУДЕН

За останні десятиліття збільшення обсягів повітряних перевезень та різноманіття повітряних суден поклали перед галуззю авіації надзвичайно складні виклики, пов'язані з управлінням повітряним рухом. Збільшення інтенсивності повітряного руху призводить до появи конфліктних ситуацій, що вимагають вдосконалення систем виявлення та управління конфліктами траєкторій повітряних суден.

Цей розділ присвячений аналізу існуючих методів виявлення конфліктів траєкторій руху повітряних суден. Широкий спектр технологій та підходів виявлення конфліктів буде ретельно розглянутий з метою розкриття їхніх переваг, недоліків та потенційних областей вдосконалення.

Аналіз існуючих методів дозволить здійснити глибоке занурення в сучасні технологічні рішення, врахувати їхній вплив на безпеку та ефективність авіаційних операцій, а також визначити напрямки подальших досліджень та вдосконалень у цій критично важливій області.

2.1. Технологічні аспекти існуючих систем

Радіотехнічні системи. Важливий аспект сучасних систем виявлення конфліктів траєкторій – використання радарів. Ці технологічні рішення базуються на принципах ехолокації, що визначає положення та швидкість повітряних суден. В аналізі технологічних аспектів існуючих систем виявлення конфліктів траєкторій у сучасній авіаційній індустрії, радіотехнічні рішення виступають визначальною складовою. Застосування радарів, які працюють на різних частотах та хвилях, набуло широкого використання у цій галузі. Радари забезпечують ефективне

виявлення повітряних суден у різноманітних метеоумовах та різних регіонах, що стало ключовим елементом систем безпеки повітряного руху.

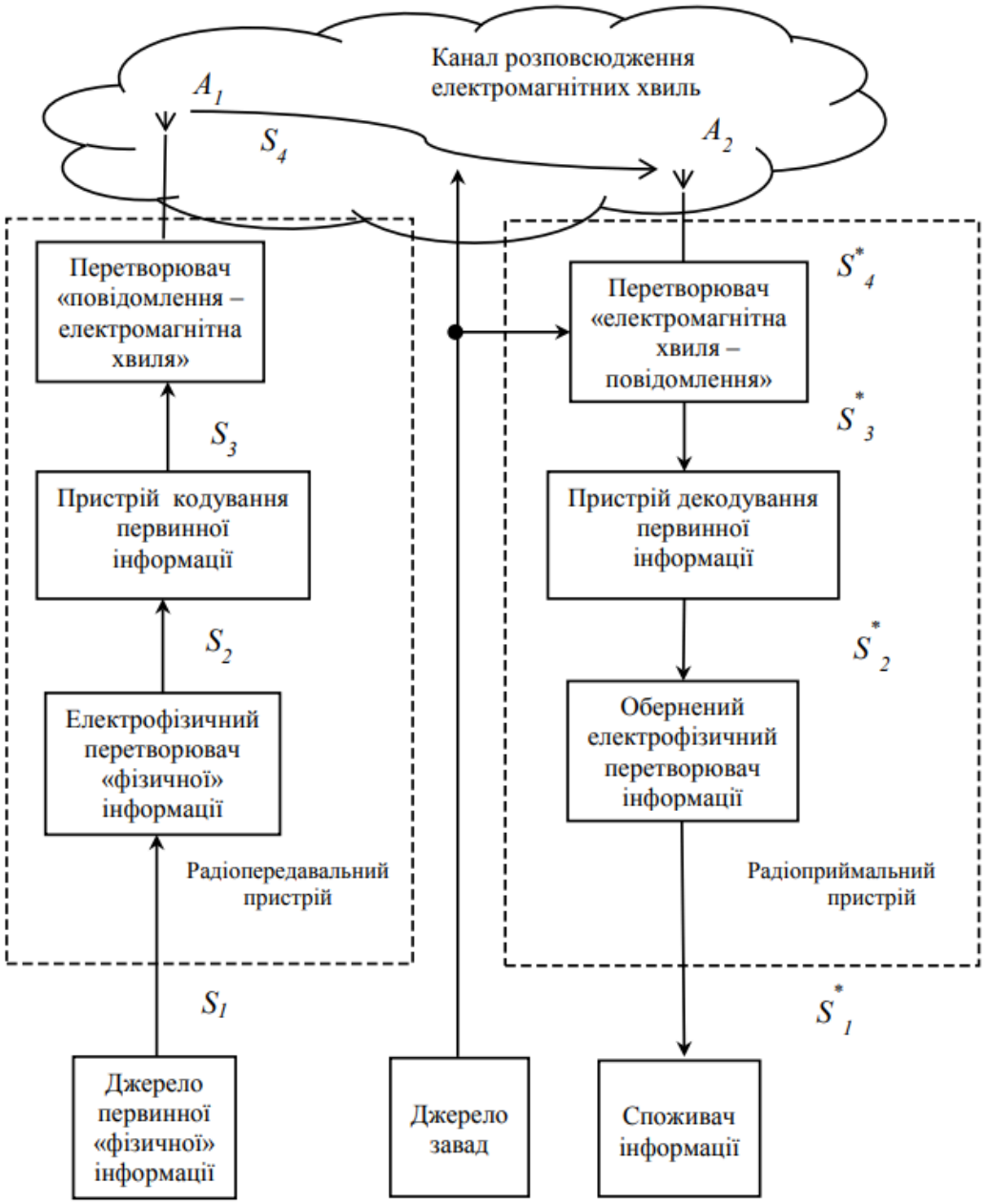


Рис. 2.1. Типова функціональна схема системи передавання інформації

Основні типи радіотехнічних систем:

– Радіорелéйна Система (РРС). Використовується для бездротового передачі сигналів між точками через мікрохвильовий діапазон. Застосування в забезпеченні зв'язку між віддаленими об'єктами, такими як міста чи регіони.

– Радарна Система. Вимірює відстань, напрямок та швидкість об'єктів за допомогою високочастотних радіосигналів. Застосовується для військових та цивільних систем для виявлення, визначення та відстеження об'єктів в повітрі чи на землі.

– Радіотелефонія. Система передачі аудіосигналів через радіохвилі для телефонних зв'язків. Застосовується для бездротового телефонного зв'язку.

– Супутниковий радіозв'язок. Використовує супутники для передачі та прийому радіосигналів для глобального зв'язку. Застосовується для Глобального Інтернету, телекомунікацій, навігації.

– Радіолокаційні Системи. Використовується для визначення положення та руху об'єктів за допомогою радіосигналів. Застосування в авіаційній, морській та промисловій безпеці.

– Радіоприймачі та Передавачі. Елементарні системи для прийому та відправлення радіосигналів. Застосовується в різноманітних пристроях, від телевізійних антен до бездротових мереж.

– Радіонавігаційні Системи. Використовують радіосигнали для визначення місцезнаходження об'єктів. Застосовується для *GPS*, Глонасс.

– Радіотехнічні Засоби Збору та Аналізу Інформації. Системи для перехоплення та обробки радіосигналів для розвідки або безпеки. Військові та цивільні застосування.

Основні технологічні аспекти включають використання різних частот та хвильових довжин у роботі радарних систем. Це дозволяє адаптувати їх до різних умов та ефективно виявляти повітряні судна. Сучасні антенні технології, такі як фазові антенні решітки та антени з активним електронним керуванням, вдосконалюють продуктивність та точність вимірювань.

Використання високочастотних діапазонів дозволяє отримувати деталізовану інформацію та точно визначати параметри об'єктів у повітрі. Такий підхід особливо важливий у сучасній авіаційній індустрії, де висока точність і швидкість вимірювань є вирішальними факторами для забезпечення безпеки та ефективності.

Низькочастотні діапазони, у свою чергу, стають ефективними в умовах обмеженої видимості та поганих погодних умов. Це дозволяє радарам працювати стабільно та надійно навіть у важких атмосферних умовах.

Окрема увага приділяється адаптивним алгоритмам, які враховують метеозалежність та атмосферні перешкоди. Ці алгоритми допомагають компенсувати вплив невизначеності погодних умов на точність визначення траєкторій повітряних суден.

Загальна концепція відзначає, що існуючі радіотехнічні рішення відіграють ключову роль у забезпеченні безпеки повітряного руху, вносячи невід'ємний вклад у сучасну авіаційну систему. Такий підхід виводить системи виявлення конфліктів траєкторій на новий рівень ефективності та надійності в авіаційному середовищі.

Сучасні антенні технології. Сучасний розвиток антенних технологій виявляє значущий вплив на покращення точності та чутливості радіотехнічних систем, зокрема в системах виявлення конфліктів траєкторій повітряних суден. Тут важливо розглянути використання фазових антенних решіток, антен з активним електронним керуванням та адаптивних антенних структур:

– Фазові антенні решітки. Фазові антенні решітки виявляються важливою інновацією, що дозволяє ефективно керувати напрямком та фокусувати радіосигнал. Зміна фази сигналів в кожній антенні дозволяє створювати зони підвищеної або зменшеної інтенсивності, забезпечуючи точне визначення положення повітряних об'єктів.

– Антени з активним електронним керуванням. Антенні з активним електронним керуванням забезпечують можливість швидкого та динамічного змінювання напрямку польоту сигналу без необхідності фізичного переміщення антени. Це особливо корисно в умовах великої кількості повітряних об'єктів, де швидкість реакції визначає ефективність системи виявлення конфліктів.

– Адаптивні антенні структури. Адаптивні антенні структури використовують алгоритми, що дозволяють системі автоматично адаптуватися до змін у вихідних умовах. Це важливо для врахування різноманітних метеоумов,

рельєфу місцевості та інших факторів, що можуть впливати на роботу радіотехнічних систем.

– Використання таких антенних технологій не тільки підвищує продуктивність систем виявлення конфліктів траєкторій повітряних суден, але і робить їх більш адаптивними та ефективними в різноманітних умовах, що забезпечує надійність та точність визначення положення об'єктів у повітрі.

Метеозалежність та атмосферні перешкоди. В контексті радіотехнічних систем виявлення конфліктів траєкторій повітряних суден, метеозалежність та атмосферні перешкоди стають важливими факторами, які можуть впливати на ефективність таких систем. Розглянемо, як сучасні технології включають адаптивні алгоритми для компенсації цих факторів та забезпечення точності визначення траєкторій повітряних суден.

Метеорологічні умови, такі як туман, дощ, сніг або турбулентність, становлять серйозні виклики для радіотехнічних систем виявлення конфліктів траєкторій повітряних суден. Розглянемо, як адаптивні алгоритми дозволяють ефективно подолати ці труднощі та забезпечити стабільну та надійну роботу систем в умовах непередбачуваних атмосферних умов.

Туман та обмежена видимість. Умови низької видимості, зокрема, туман, можуть значно обмежити здатність радіотехнічних систем виявлення конфліктів. Адаптивні алгоритми враховують ці умови та автоматично коригують параметри систем для оптимальної роботи в умовах обмеженої видимості, дозволяючи ефективно визначати траєкторії повітряних суден.

Дощ, сніг та турбулентність. Погіршення атмосферних умов, таке як дощ, сніг або турбулентність, може викликати непередбачені зміни у роботі радіотехнічних систем. Адаптивні алгоритми аналізують зміни у середовищі та розпізнають їх вплив на зібрані дані, допомагаючи системі підтримувати стійку роботу та високу ефективність у важких погодних умовах.

Автоматичне врахування змін. Основною перевагою адаптивних алгоритмів є їх здатність автоматично враховувати зміни в метеорологічних умовах. Це означає,

що система може оперативно реагувати на будь-які зміни у видимості чи інших параметрах, утримуючи стабільну та надійну роботу в умовах непередбачуваності.

Забезпечення стійкості та надійності. Адаптивні алгоритми є ключовим елементом для забезпечення стійкості та надійності радіотехнічних систем в умовах метеорологічної змінливості. Їх використання дозволяє системі ефективно функціонувати навіть у складних атмосферних умовах, забезпечуючи невід'ємну складову безпеки повітряного руху.

Узагальнюючи, адаптивні алгоритми визначають успішну роботу радіотехнічних систем в умовах невизначеності, гарантуючи їх ефективність та надійність в будь-яких атмосферних умовах.

2.2. Радіолокатори та їх роль в авіації

Радар первинного спостереження (*PSR – Primary Surveillance Radar*). Працює за принципом відправлення та прийому електромагнітних хвиль від певних об'єктів.

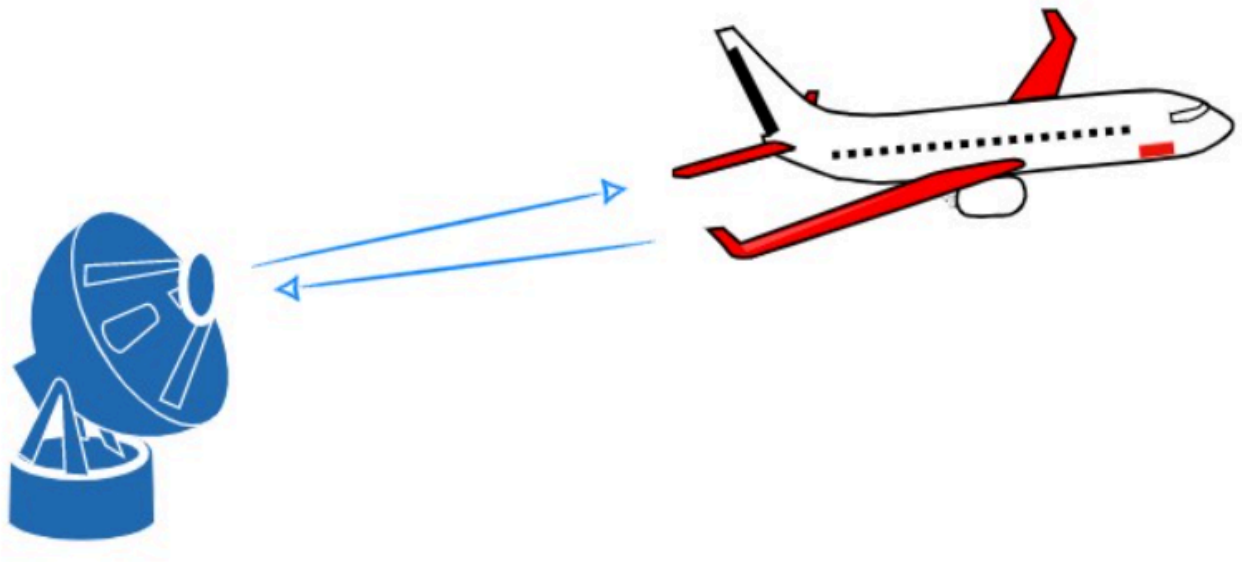


Рис. 2.2. Випромінювання та повернення хвиль від літака

Операційний принцип такого радара базується на ехолокації. Антена радара висилає імпульси із значною потужністю, які поширюються зі швидкістю електромагнітної хвилі, що дорівнює приблизно 300 000 *км/с* (швидкість світла).

Літак відбиває ці хвилі, що повертаються назад, і радар ідентифікує їх, обертаючись навколо своєї осі з певною періодичністю, а охоплює весь простір 360°.

Хоча первинний радар залишається в експлуатації у службах контролю повітряного руху, використовуючись як резервна або додаткова система поряд із вторинним радаром, його покриття та інформація є менш обширними.

Вторинний радар. На відміну від первинного радара, вторинний радар (*SSR – Secondary Surveillance Radar*) використовує спеціальний передавач на борту літака, відомий як транспондер.

Антена вторинного радіолокатора здійснює обертальні рухи з визначеною частотою (в середньому 5-12 обертів за хвилину). При цьому передається імпульс, який отримується та опрацьовується системами літака. Літак “відправляє” відповідь у вигляді закодованого сигналу з визначеною інформацією. Об’єм даних описано режимом передавача (*SSR Mode*).

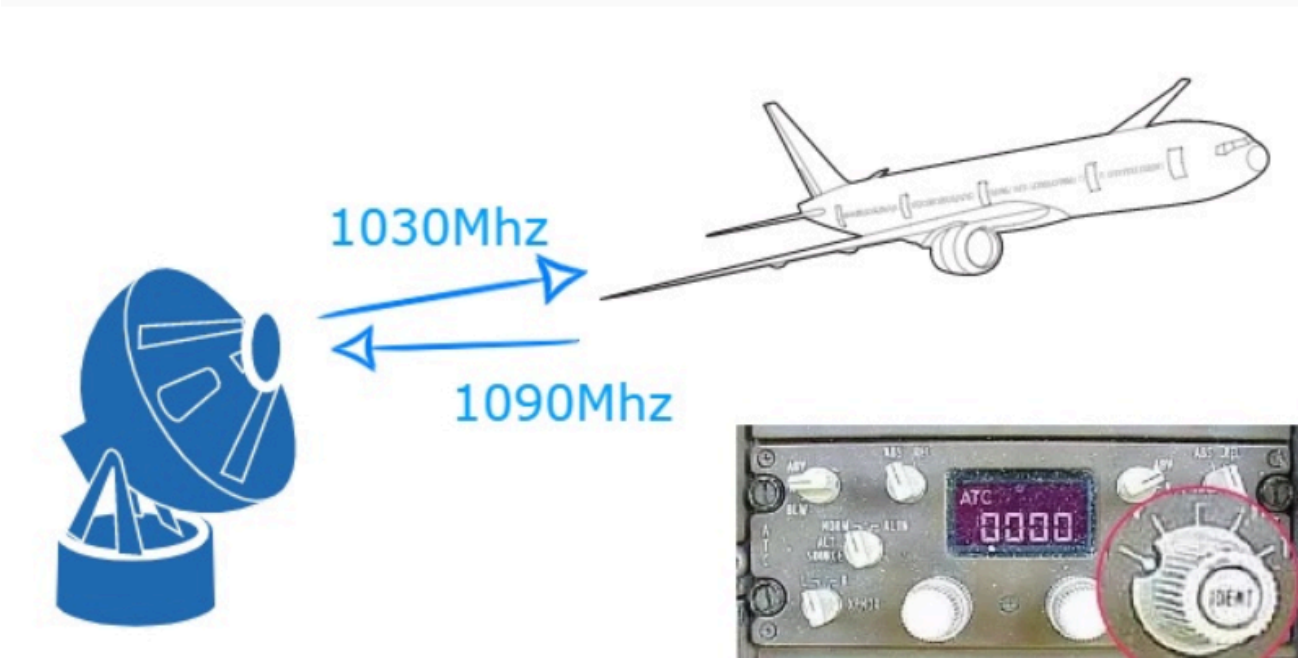


Рис. 2.3. Робота *SSR* та налаштування транспондера на борту літака (*SQUAWK*)

Транспондер. Земні радари проводять аналіз як первинних, так і вторинних міток. Первинні мітки надходять від первинного радіолокатора і мають обмежену кількість інформації. У той час як вторинні мітки, які відображаються на екрані радара, представляють інформацію, отриману від літаків, оснащених транспондерами. Транспондери працюють у режимах (*A, C, S*) та передають відповідну інформацію.

***TCAS* (*Traffic Alert and Collision Avoidance*).** Якщо система виявляє потенційну загрозу зіткнення, відповідне повідомлення відображається перед пілотами, і враховуючи ступінь небезпеки, пілот отримує аудіосигнал одного з типів:

– *Traffic advisory* (*TA*). Зауваження щодо іншого повітряного транспорту; транспорт.

– *Resolution advisory* (*RA*). Сигнали, які обов'язково слід виконати, незважаючи на інструкції диспетчера чи інших. У таких випадках авіадиспетчер не бере на себе відповідальність за контроль повітряного судна до тих пір, поки конфлікт не буде усунено. Всю відповідальність несе екіпаж. Серед загальних інструкцій: Піднятися; піднятися. Знизитися; знизитися. Моніторити вертикальну швидкість та інші.

– *Clear of conflict*. Небезпеки більше немає. *TCAS* встановлюється, якщо вага повітряного судна перевищує 5700 *кг* або кількість пасажирів перевищує 19 осіб.

TCAS має декілька поколінь:

– *TCAS I* надає оповіщення про потенційну небезпеку, яка повинна бути взята до уваги пілотами.

– *TCAS II* надає пілоту *resolution advisories*(*RAs*) в додаток до *TA*. Це забезпечує пілотів інформацією про правильні маневри у вертикальній площині для уникнення зіткнення.

– *TCAS III* та *TCAS* IV повинні надавати вказівки щодо розходження в горизонтальній площині, однак поки що не використовуються в силу певних проблем.

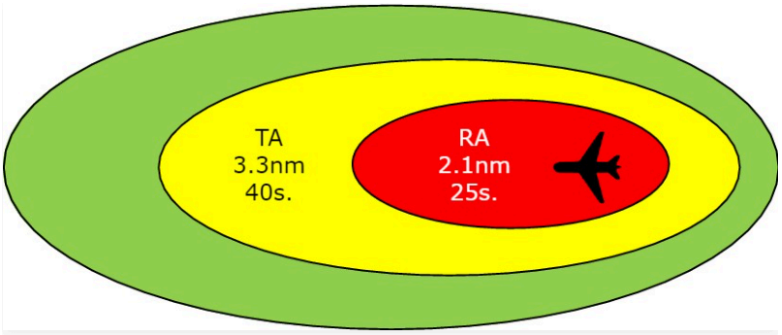
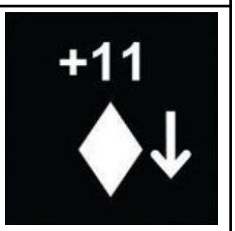
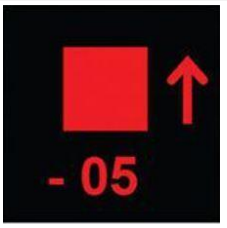


Рис. 2.4. *TA*, *RA* та *Clear of Conflict* зони

Таблиця 2.1

Відображення на приладі

Опис	**Умовне позначення**
Own aircraft. Власний літак, як варіант також може відображатись символ літака (*A320 etc.)*	
Other aircraft. Інший літак, висота невідома	
Proximate traffic. Літак в діапазоні від *6NM* до 1200 футів. Відносна висота вимірюється у сотнях футів над літаком.	
Traffic advisory. Літак знаходиться на висоті 900 футів нижче.	

Resolution advisory. Літак знаходиться на висоті 500 футів нижче і піднімається.	

Елементи *TCAS*:

– Процесор *TCAS* – включає в себе набір алгоритмів для моніторингу повітряного простору, виявлення потенційних загроз, визначення та вибір маневрів *RA*, а також генерацію повідомлень.

– Антени.

– Представлення в кабіні пілота – зазвичай вбудовано в рамках *EFIS* або у вигляді окремого приладу.

Раніше функціонування *TCAS* включало вимикання автопілота, *flight* директорів та перехід до ручного управління. Проте, на новіших літаках, таких як *A350*, *A380*, або нових версіях *A320*, була представлена можливість *Autopilot*/*Flight Director* (*AP*/*FD*) *TCAS*, що дозволяє автоматизованому *TCAS* працювати без втручання пілота.

Здається, що *TCAS III* є частиною майбутнього розвитку системи.

Важливо відзначити, що система, як *ADS-B*, стане частиною комплексу систем попередження зіткнень і допоможе більш точно визначати загрозу. Новий стандарт цього гібридного підходу був опублікований у 2013 році.

Також варто відзначити перспективний проєкт *TCAS* "*light*", відомий як *Traffic Situation Awareness with Alerts* (*TSAA*), стандарт якого був недавно представлений. Протягом наступних кількох років очікується, що виробники почнуть пропонувати продукти на основі цього стандарту.

2.3. Аналіз програмних рішень

Автоматизовані системи виявлення. Вони є ключовим елементом сучасної авіаційної безпеки та ефективності.

***Multilateration* (*MLAT*).** Системи, що використовують *MLAT*, засновані на отриманні даних від різних приймачів. Метод визначає положення повітряного судна шляхом аналізу різниці часу приходу сигналів.

Широко використовується для визначення точних координат повітряних суден в режимі реального часу.

Використовується для точного визначення місця розташування повітряних суден, що дозволяє уникати конфліктів та забезпечувати безпеку в повітрі.

MLAT потрібен для відстеження руху літаків на аеродромах та в околицях, щоб уникнути ситуацій загрозливого наближення.

Визначення точного положення літаків за допомогою *MLAT* сприяє ефективному використанню повітряного простору та оптимізації розкладів рейсів. *MLAT* визначається як важливий елемент систем виявлення в авіації, принципово важливий для забезпечення безпеки та ефективності повітряного руху. Ефективність цієї технології полягає в її здатності точно визначати місцезнаходження повітряних суден у режимі реального часу.

Приймачі. *MLAT* використовує тривимірні приймачі розміщені на різних пунктах, щоб отримати широкий огляд повітряного простору.

Синхронізація часу. Для точності визначення місцезнаходження, усі приймачі повинні бути синхронізовані за часом, щоб вимірювати різницю часу прибуття сигналів.

Геометричні обчислення. Аналізуючи час прибуття сигналів на різних приймачах, система використовує геометричні принципи для визначення точного положення об'єкта.

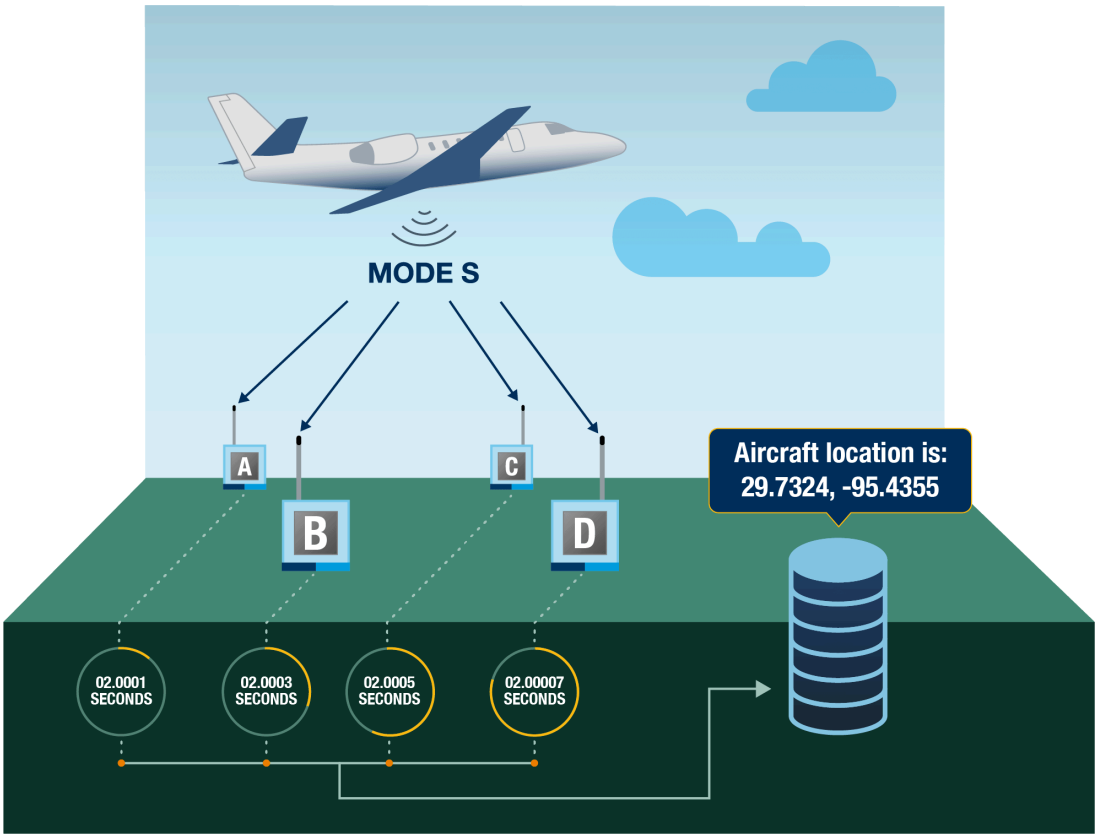


Рис. 2.5. Зображення роботи *MLAT*

***Automatic Dependent Surveillance-Broadcast* (*ADS-B*).** Це передова технологія в сфері авіаційного нагляду, яка базується на концепції автоматичного залежного спостереження та передачі.

У рамках *ADS-B*, повітряні судна використовують *GPS*-навігацію для визначення свого положення та інших параметрів. За допомогою цих даних, судно транслює інформацію в ефір (*broadcast*), що робить її доступною для інших повітряних суден та земних станцій. *ADS-B* відіграє важливу роль у покращенні системи повітряного нагляду та безпеки авіаційного руху. Основна ідея полягає в тому, що повітряні судна автоматично висилають свої параметри (такі як положення, висота, швидкість та інші) через *ADS-B*, надаючи більш точну та надійну інформацію для контролю повітряного руху.

ADS-B має ряд переваг, зокрема, висока частота оновлення даних, що робить її ефективною для виявлення конфліктів траєкторій та попередження зіткнень. Крім того, ця технологія дозволяє отримувати більш широкий обсяг інформації про стан повітряного руху в реальному часі, що сприяє покращенню координації та безпеки.

ADS-B є ключовою частиною сучасних систем авіаційного нагляду та виступає важливим кроком у напрямку автоматизації та удосконалення повітряного руху.

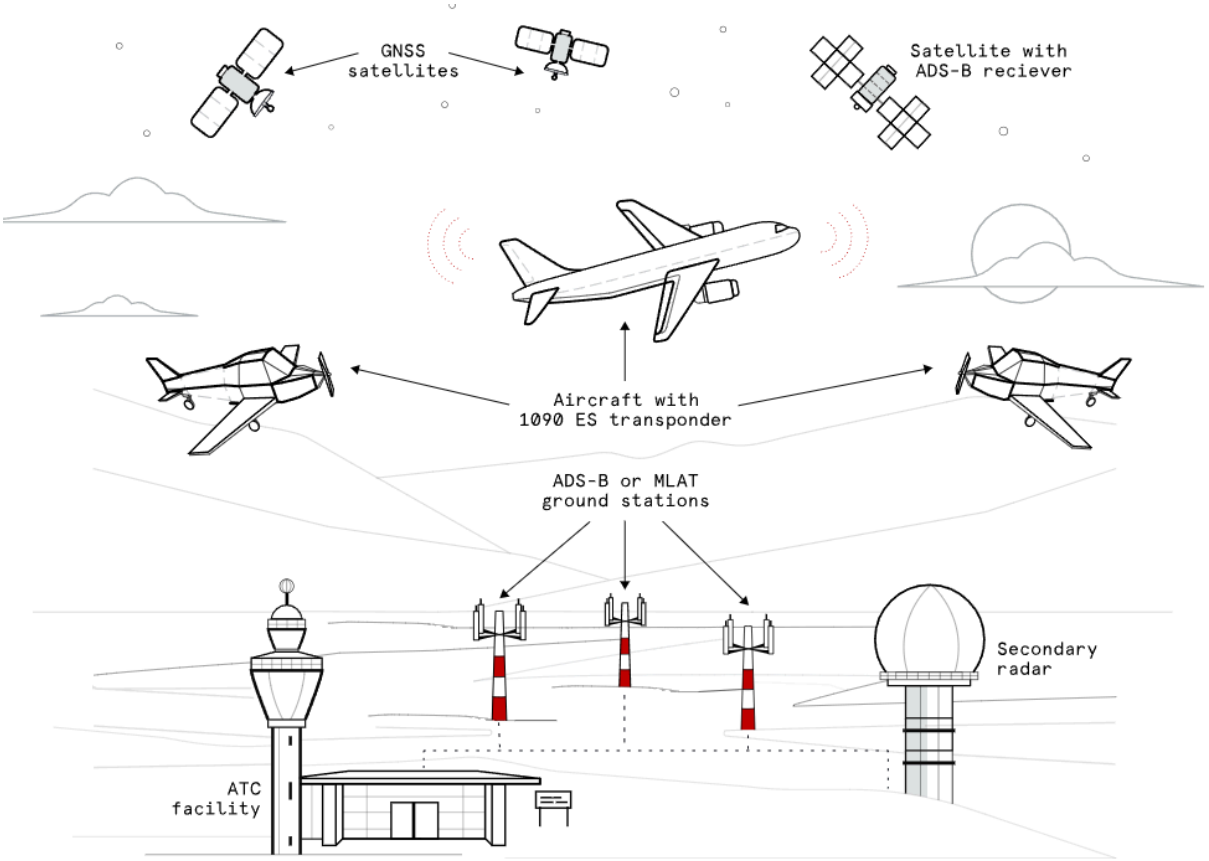


Рис. 2.6. Зображення роботи *ADS-B*

***Collaborative Decision Making* (*CDM*).** *CDM* використовується для об'єднання та обміну даними між різними авіаційними учасниками, забезпечуючи взаємодію та вирішення проблем ефективного управління повітряним простором. *CDM* допомагає оптимізувати розклади рейсів, реагувати на непередбачені обставини та забезпечує ефективний обмін інформацією між всіма сторонами. *Collaborative Decision Making* (*CDM*) представляє собою стратегічний підхід до управління повітряним рухом, що ґрунтується на взаємодії та обміні інформацією між різними стейкхолдерами в авіаційній системі. Цей підхід розглядається як ключовий елемент сучасних технологій, спрямованих на вдосконалення ефективності та безпеки повітряних перевезень.

У рамках *CDM*, учасники авіаційного процесу, такі як авіакомпанії, аеропорти, служби повітряного руху та інші, взаємодіють та обмінюються інформацією для

спільного прийняття рішень. Це охоплює різні аспекти, включаючи планування рейсів, розклади аеропортів, управління повітряним рухом та реагування на непередбачені ситуації.

CDM допомагає оптимізувати використання ресурсів та скорочує час затримок, сприяючи покращенню пасажирського обслуговування та зменшенню викидів *CO2*. Це досягається завдяки вдосконаленню координації між різними частинами авіаційної системи, що дозволяє пристосовувати операції до змінних умов та забезпечувати високий рівень безпеки.

CDM не лише покращує продуктивність, але й створює основу для майбутніх технологічних інновацій у сфері авіації. Цей підхід активно використовується в ініціативах сучасних авіаційних систем, спрямованих на збільшення автоматизації та інтеграції всіх частин системи повітряних перевезень.

Рис. 2.7. Зображення роботи *Airport-CDM*

2.4. Алгоритми обробки даних в авіації

Kalman Filter **(Калманівський фільтр).** Математичний алгоритм, який використовується для обробки вимірювань та оцінювання стану динамічної системи. В контексті виявлення траєкторій повітряних суден, цей фільтр грає важливу роль у покращенні точності та надійності прогнозів.

Калманівський фільтр працює на основі двох основних кроків: прогноз та корекція. По-перше, за допомогою моделі динамічної системи, фільтр робить прогноз майбутнього стану об'єкта. Потім, коли з'являється нове вимірювання, фільтр коригує прогноз, враховуючи точність вимірювань та неоднорідність динаміки об'єкта.

Калманівські фільтри широко використовуються в авіації для визначення точного положення, швидкості та напрямку руху повітряних суден. У виявленні траєкторій повітряних суден, Калманівський фільтр використовує інформацію від радіотехнічних систем та інших датчиків для прогнозування майбутніх координат та швидкості кожного судна.

Калманівський фільтр дозволяє підтримувати високу точність та стійкість в умовах обмежених даних, роблячи його незамінним інструментом в авіаційних системах виявлення траєкторій.

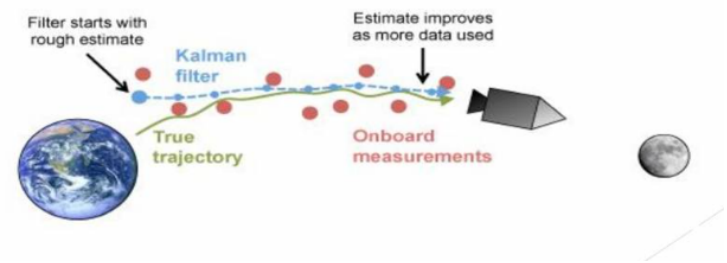


Рис. 2.8. Зображення роботи Калманівського фільтру

Particle Filter **(Частинковий фільтр).** Алгоритм статистичної оцінки, який використовує набір частинок або "частинок стану" для апроксимації розподілу стану системи. У контексті виявлення траєкторій повітряних суден, частинкові фільтри можуть допомагати враховувати невизначеність та забезпечувати стійкість в умовах

змінюючоїся динаміки. Частинковий фільтр працює шляхом створення великої кількості частинок, кожна з яких представляє можливий стан системи. Ці частинки розподіляються відповідно до ймовірностей, а потім оновлюються на основі вимірювань та динаміки системи. Такий підхід дозволяє адаптуватися до невизначеності та нелінійності процесів.

В контексті авіаційних систем виявлення траєкторій, частинкові фільтри використовують інформацію від різних датчиків та вимірювань для оцінки стану кожного повітряного судна. Це особливо ефективно при роботі з нелінійними динамічними системами. Частинкові фільтри широко використовуються в авіаційних системах для визначення траєкторій, особливо в умовах, де інші фільтри можуть бути менш ефективними через складні динамічні умови.

Частинкові фільтри надають ефективний метод обробки даних в умовах невизначеності, роблячи їх важливим інструментом для точного визначення траєкторій повітряних суден.

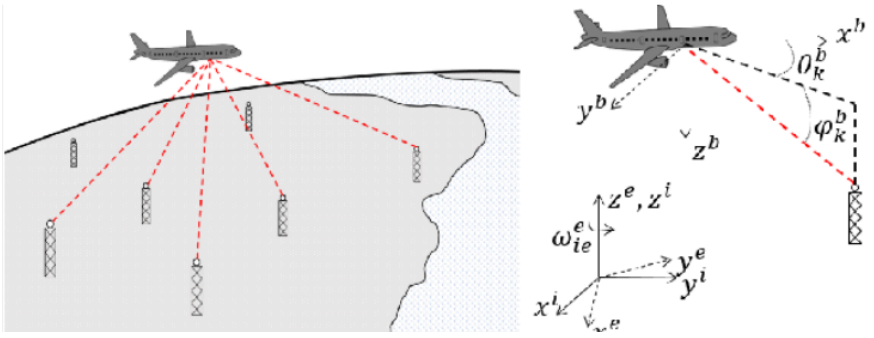


Рис. 2.9. Зображення роботи Частинкового фільтру

Neural Networks **(Нейронні мережі).** комп'ютерні системи, інспіровані структурою та функціями людського мозку, які використовуються для вирішення різних завдань, включаючи аналіз даних, розпізнавання образів та прогнозування.

Нейронні мережі складаються зі з'єднаних штучних нейронів, які утворюють велику мережу. Ці нейрони розділені на шари: вхідний, прихований (якщо присутній), та вихідний. Кожен зв'язок між нейронами має вагу, яка визначає важливість сигналу. Нейронні мережі "навчаються" на основі вхідних даних та

бажаного вихідного результату шляхом оптимізації ваг для досягнення бажаної функції.

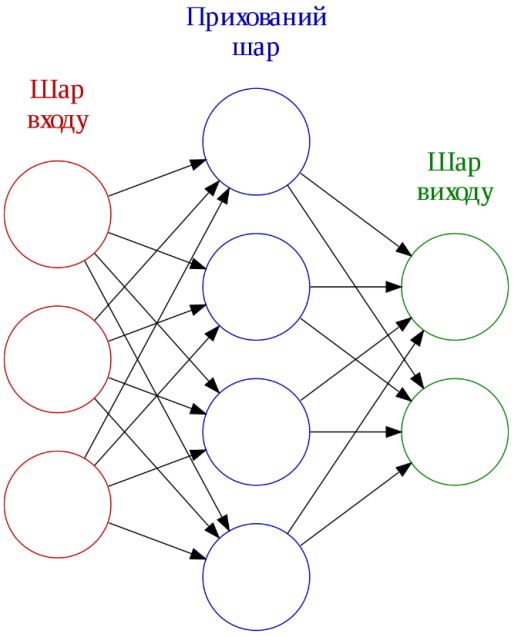


Рис. 2.10. Шари та “Нейрони” нейронної мережі

У сфері авіації нейронні мережі використовуються для різноманітних завдань, таких як прогнозування траєкторій повітряних суден, аналіз повітряного руху та систем управління та безпеки. Вони дозволяють автоматизувати процеси та робити системи більш адаптивними. В авіаційній галузі нейронні мережі використовуються для прогнозування та управління повітряним рухом, а також для оптимізації використання ресурсів та забезпечення безпеки.

Нейронні мережі стають важливим компонентом в авіаційній технології, забезпечуючи зростаючу автоматизацію та покращення ефективності систем управління.

Genetic Algorithms **(Генетичні алгоритми).** Еволюційний підхід до оптимізації та пошуку рішень, інспірований процесами природної селекції та генетики. Вони використовують принципи спадковості, мутації та відбору для створення нових поколінь потенційних рішень з метою знаходження оптимальних параметрів.

Генетичні алгоритми починаються зі створення початкової популяції рішень, представлених у вигляді "хромосом". Кожен елемент цієї популяції відображає потенційне рішення. Потім, застосовуючи оператори схрещування, мутації та відбору, генетичний алгоритм створює нове покоління рішень, відбираючи та зберігаючи найкращі.

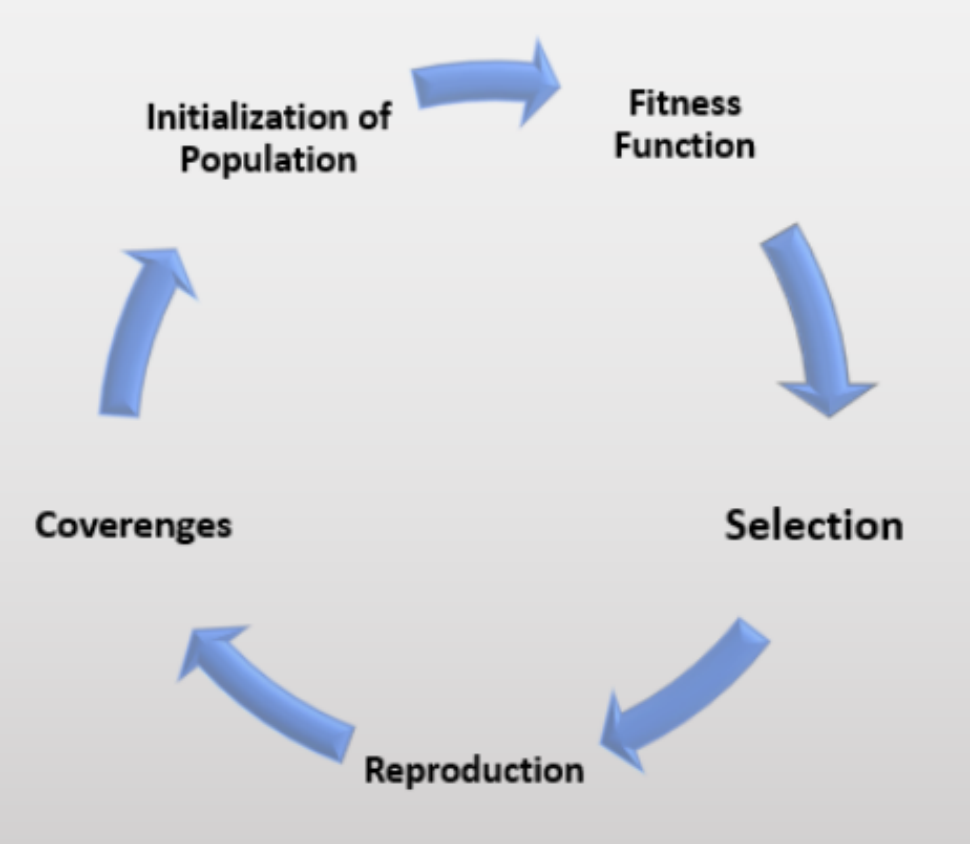


Рис. 2.11. Генетичний алгоритм в загальному вигляді

У сфері авіації генетичні алгоритми використовуються для оптимізації різних завдань, таких як маршрутизація повітряних суден, планування та розкладання рейсів, а також для проектування та оптимізації систем автоматизованого управління.

З розвитком обчислювальної потужності та вдосконаленням алгоритмів, генетичні алгоритми продовжують розвиватися та застосовуватися для розв'язання великої кількості завдань в авіаційній галузі.

2.5. Моделювання траєкторій

У системах УПР моделювання траєкторій використовується для аналізу та прогнозу руху повітряних суден в різних сценаріях. Це дозволяє адаптувати маршрути, швидкості та висоти для оптимізації потоку повітряного руху та уникнення конфліктів.

Головною метою моделювання траєкторій є прогнозування та управління рухом повітряних суден з максимальною точністю та ефективністю. Це допомагає уникнути конфліктів, забезпечити оптимальний потік та використовувати повітряний простір ефективно.

Моделювання траєкторій широко використовується в авіаційній галузі для планування маршрутів, прогнозування часу прильоту та взлету, а також для адаптації до змінних погодних умов та інших факторів.

В моделюванні траєкторій використовуються різні технології та алгоритми, включаючи генетичні алгоритми, методи оптимізації, аналітичні та числові методи для забезпечення точності та швидкості розрахунків.

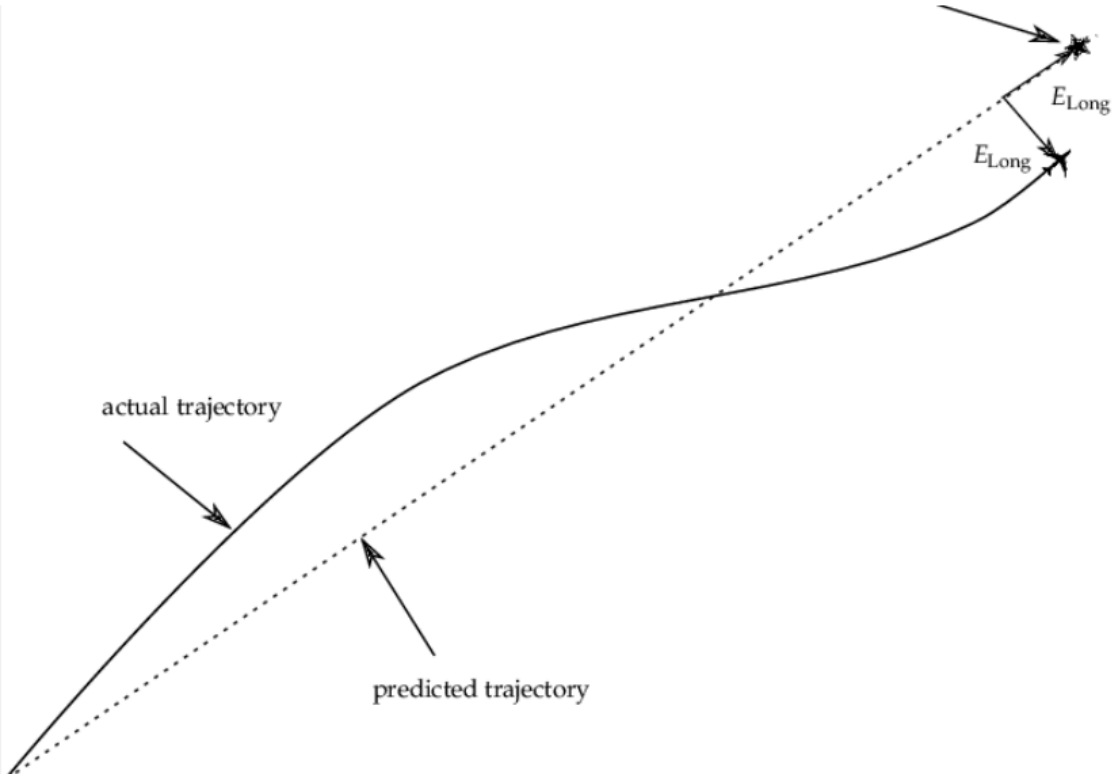


Рис. 2.12. Змодельована та фактична траєкторія руху ПС

Моделювання траєкторій відіграє ключову роль у підвищенні безпеки авіаційних операцій, забезпечуючи операторам та системам УПР інструменти для ефективного управління повітряним рухом та прогнозування можливих конфліктів.

2.6. Стандартизація протоколів

Стандартизація протоколів в авіаційних системах грає ключову роль у забезпеченні сумісності та ефективного обміну даними між різними компонентами авіаційної інфраструктури. Цей процес спрощує інтеграцію та забезпечує єдність комунікаційного середовища. Давайте розглянемо докладно кілька аспектів стандартизації протоколів в авіаційних системах.

***Controller Pilot Data Link Communications* (*CPDLC*) та *Aeronautical Telecommunications Network* (*ATN*).** *CPDLC* представляє систему обміну даними між повітряними суднами та диспетчерськими центрами, яка використовує цифровий канал зв'язку. Ця технологія впроваджується для полегшення та покращення комунікації між екіпажами та контролерами повітряного руху.

CPDLC визначається рядом стандартів, які визначають протоколи та процедури обміну даними між повітряними суднами та контролерами повітряного руху. Найбільш загальні стандарти, що стосуються *CPDLC*, визначаються організацією *ICAO* (Міжнародна організація цивільної авіації) та Європейським агентством з безпеки авіації (*EASA*).

CPDLC використовується для обміну текстовими повідомленнями між диспетчерським центром та повітряним судном. Він може включати попередження про небезпеку, деталізацію маршруту, зміни висоти, часові інструкції та інші важливі повідомлення. Деякі ключові стандарти:

– *Aeronautical Telecommunications Network Baseline* 1 (*ATN B1*). *ATN B1* визначає стандарти для цифрового обміну даними в цивільній авіації. Включає в себе протоколи для *CPDLC*, *FANS* (*Future Air Navigation Systems*) та інші авіаційні

додатки. Організація *ICAO* визнає *ATN B1* як найважливіший стандарт для гармонізації *CPDLC* у всьому світі.

– Правила *EASA*. *EASA* визначає стандарти безпеки та ліцензування для впровадження *CPDLC* в європейській авіації.

– Стандартизація Європейського Співтовариства. Для забезпечення сумісності та єднання, Європейське Співтовариство визначає деякі стандарти, пов'язані з *ATN*.

– *ARINC* (*Aeronautical Radio*, *Incorporated*). *ARINC*, яка розробляє стандарти для авіаційної індустрії, також вносить свій внесок у визначення протоколів для *ATN*

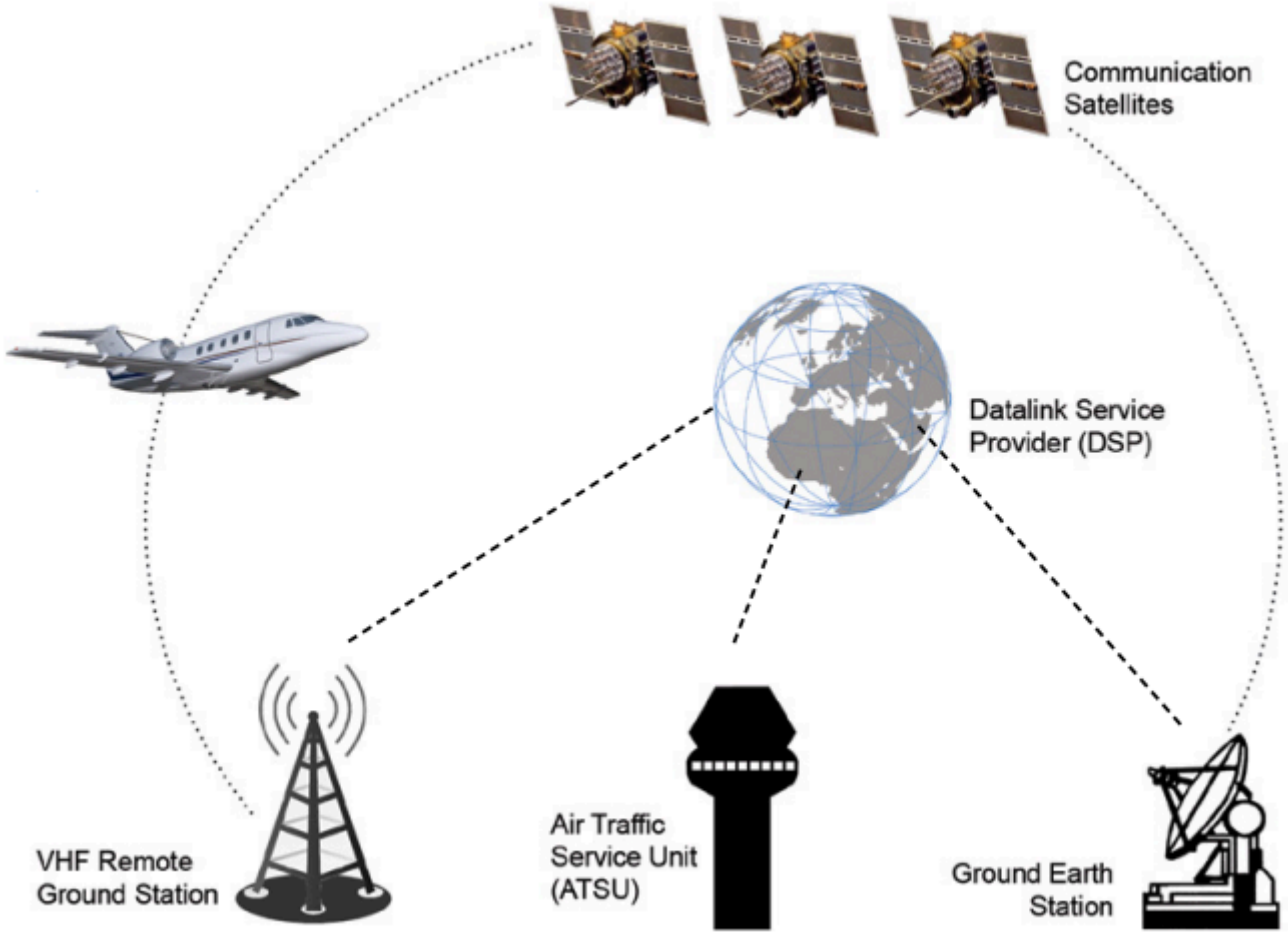


Рис. 2.13. Стандартизований обмін даними

Враховуючи ці стандарти, *CPDLC* та *ATN* гарантують сумісність, безпеку та ефективність в області обміну даними в цивільній авіації, сприяючи модернізації та удосконаленню авіаційних систем.

2.7. Камери та оптичні сенсори

Камери та оптичні сенсори в системах виявлення конфліктів траєкторій повітряних суден грають важливу роль у візуальному спостереженні та відстеженні об'єктів у повітрі.

Камери. Вони є оптичними пристроями, призначеними для фотографування або відеозйомки зображень. В авіаційних системах, камери використовуються для візуального виявлення та відстеження повітряних суден. Наприклад, системи нагляду на аеродромах використовують камери для моніторингу руху суден на злітно-посадкових смугах та платформах.

Інфрачервоні камери. Інфрачервоні камери здатні реєструвати теплове випромінювання об'єктів, що дозволяє виділяти їх у низькій видимості або в умовах обмеженої освітленості. В авіаційних системах інфрачервоні камери можуть використовуватися для виявлення теплових слідів повітряних суден у різних погодних умовах.

Оптичні сенсори зі здатністю до зуму. Оптичні сенсори, які можуть збільшувати об'єкти на зображенні без втрати якості. Такі сенсори використовуються для подробного вивчення повітряних суден на великій відстані.

Камери з високою роздільною здатністю. Камери, які мають велику кількість пікселів, забезпечуючи високу роздільну здатність зображення. В авіаційних системах вони можуть використовуватися для точного визначення деталей на повітряних суднах та визначення їхнього стану.

Системи відеоспостереження. Сукупність камер, розташованих в різних точках, що надають комплексне відображення подій. Системи відеоспостереження можуть використовуватися на аеродромах для слідкування за рухом повітряних суден та забезпечення безпеки.

Оптичні сенсори вносять вагому частину в системи виявлення конфліктів траєкторій, сприяючи надійному та ефективному спостереженню та аналізу повітряного простору.

2.8. Висновки до розділу

Розділ 2 систематично досліджує аспекти технологій та методів, використовуваних в системах виявлення конфліктів траєкторій руху повітряних суден. Аналізуючи існуючі технологічні рішення, розділ дозволяє отримати глибше розуміння складних завдань, які постають перед сучасною авіаційною індустрією та системами управління повітряним рухом (УПР).

Переважно, висвітлено радіотехнічні аспекти, зосереджуючись на важливості радарів у виявленні повітряних суден. Зазначено, що сучасні радари, які використовують різні частоти та хвилі, стали необхідністю в контексті зростання обсягів повітряних перевезень та різноманіття повітряних суден.

Також було висвітлено поширені алгоритм изо використовуються в побудові траєкторій руху повітряних суден і зроблено висновки що алгоритми обробки даних в авіації відіграють ключову роль у забезпеченні ефективного та безпечного управління повітряним рухом. Ці алгоритми використовуються для обробки інформації, отриманої від різних джерел, таких як радіотехнічні системи, сенсори, *GPS*, та інші джерела даних, що стосуються руху повітряних суден.

Аналіз технологічних аспектів дозволив виокремити використання різних частот та хвильових довжин у роботі радарних систем. Це підкреслило адаптивність та універсальність таких систем у різних умовах експлуатації. Фазові антенні решітки та антенни з активним електронним керуванням додатково покращують продуктивність та точність вимірювань.

Розглянуто стандарти обміну інформацією в системах управління повітряним рухом (УПР).

Висвітлено також проблему метеозалежності та атмосферних перешкод, що можуть впливати на ефективність радіотехнічних систем. Адаптивні алгоритми розглядаються як рішення для компенсації впливу атмосферних факторів на точність визначення траєкторій повітряних суден.

РОЗДІЛ 3
АНАЛІЗ ВИМОГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У сфері системної інженерії та розробки програмного забезпечення, аналіз вимог акцентує увагу на завданнях, які визначають потреби чи умови для задоволення нового чи зміненого продукту або проекту. Це враховує можливі конфліктуючі вимоги різних сторін, проводячи аналіз, документацію, валідацію та управління вимогами до програмного забезпечення чи системи. Значення аналізу вимог для успіху чи невдачі системного чи програмного проекта визначальне. Вимоги повинні бути чітко задокументовані, ефективні, вимірювані, перевірені, простежені та пов'язані із виявленими потребами чи можливостями бізнесу, визначені на детальному рівні для проектування системи.

В аналізі вимог для розробки сервісу виявлення конфліктних траєкторій руху повітряних суден акцент робиться на розгляді фундаментальних потреб та умов, що визначають його функціональність. Цей етап становить ключову частину проектування системи, враховуючи необхідність точного та вчасного виявлення можливих конфліктів у повітряному просторі.

На першому етапі важливо чітко визначити потреби та обмеження, які визначатимуть основні характеристики сервісу. Глибокий аналіз спрямований на ідентифікацію функціональних вимог, які забезпечать ефективність та надійність системи.

У контексті даного проекту особлива увага приділяється розгляду різних сценаріїв повітряного руху та специфічних характеристик, які можуть впливати на процес виявлення конфліктів. Врахування інтересів та вимог різних стейкхолдерів, таких як оператори повітряного простору, авіакомпанії та технічні експерти, дозволяє створити балансовану систему, що враховує різноманітні потреби та очікування.

Отже, аналіз вимог для розробки сервісу виявлення конфліктних траєкторій руху повітряних суден визначає стратегічні напрямки роботи, що спрямовують подальший процес проектування та впровадження системи.

Головна мета теми даної частини полягає в створенні простого та ефективного сервісу для виявлення конфліктних траєкторій руху повітряних суден. Аналіз вимог у даному контексті є важливим завданням, яке може вимагати тривалого та ретельного процесу, залучаючи психологічні навички. Враховуючи зміни в середовищі та взаємодії між особами, необхідно ідентифікувати всі зацікавлені сторони, узгоджувати їхні потреби та переконатися, що вони розуміють наслідки впровадження нових систем.

3.1. Аналіз нефункціональних вимог

Аналіз нефункціональних вимог для сервісу виявлення конфліктних траєкторій руху повітряних суден є важливим етапом у проектуванні системи, оскільки визначає не лише її функціональність, а й параметри, якості та обмеження, що впливають на взаємодію із користувачем та оточуючим середовищем.

Аналітики мають різні методи виявлення вимог від клієнта. Це включає розробку сценаріїв, які можуть бути представлені у вигляді історій користувача в гнучких методах, визначення варіантів використання, спостереження за робочим місцем або етнографію, проведення інтерв'ю або фокус-груп (відомих також як семінари за вимогами або сесії огляду вимог) та формування списків вимог. Також може використовуватися прототипування для розробки еталонної системи, яку можна продемонструвати зацікавленим сторонам для отримання актуального відгуку.

Нефункціональні вимоги. ПЗ повинне бути:

- Юзер-френдлі.
- Легкодоступне.
- Зручне в користуванні.
- Зрозуміле.

– Готове до масштабування.

Доступне. Система буде доступна 24/5 та 22/2, з використанням 4 годин в неділю на усунення проблем та оновлення системи. Також система може бути недоступною більше 4 годин в разі критичних проблем. Максимальний час недоступності системи не перевищить 7 годин в неділю. Час оновлення системи буде використаний в момент найменшої активності системи.

ПЗ повинно запускатись на більшості пристроїв, і тому вибір веб-сайту як реалізації є логічним. Він легкий для пристроїв і вимагає лише браузера та інтернет-з'єднання.

Масштабоване. Для реалізації сайту буде використано *TypeScript*, а для побудови *UI React*.

Актуальне. Актуальність є основною вимогою, оскільки важливо, щоб сервіс був потрібним користувачам. Без актуальності немає охоплення аудиторії, існування та значення веб-сайту стають сумнівними.

Юзер-френдлі. Зручний інтерфейс є невід'ємною складовою ПЗ. Інтерфейс повинен бути простим і не перевантаженим зайвими елементами.

Зручне. ПЗ повинно легко використовуватись без зайвих зусиль. Без нормального користувацького досвіду навіть потужний продукт може залишитися непопулярним.

Зрозуміле Важливо враховувати цільову аудиторію. Оформлення повинно бути стриманим та функціональним.

Вимоги вказані вище є необхідними для кінцевого користувача, і важливо узгоджувати їхні побажання та знаходити спільні точки.

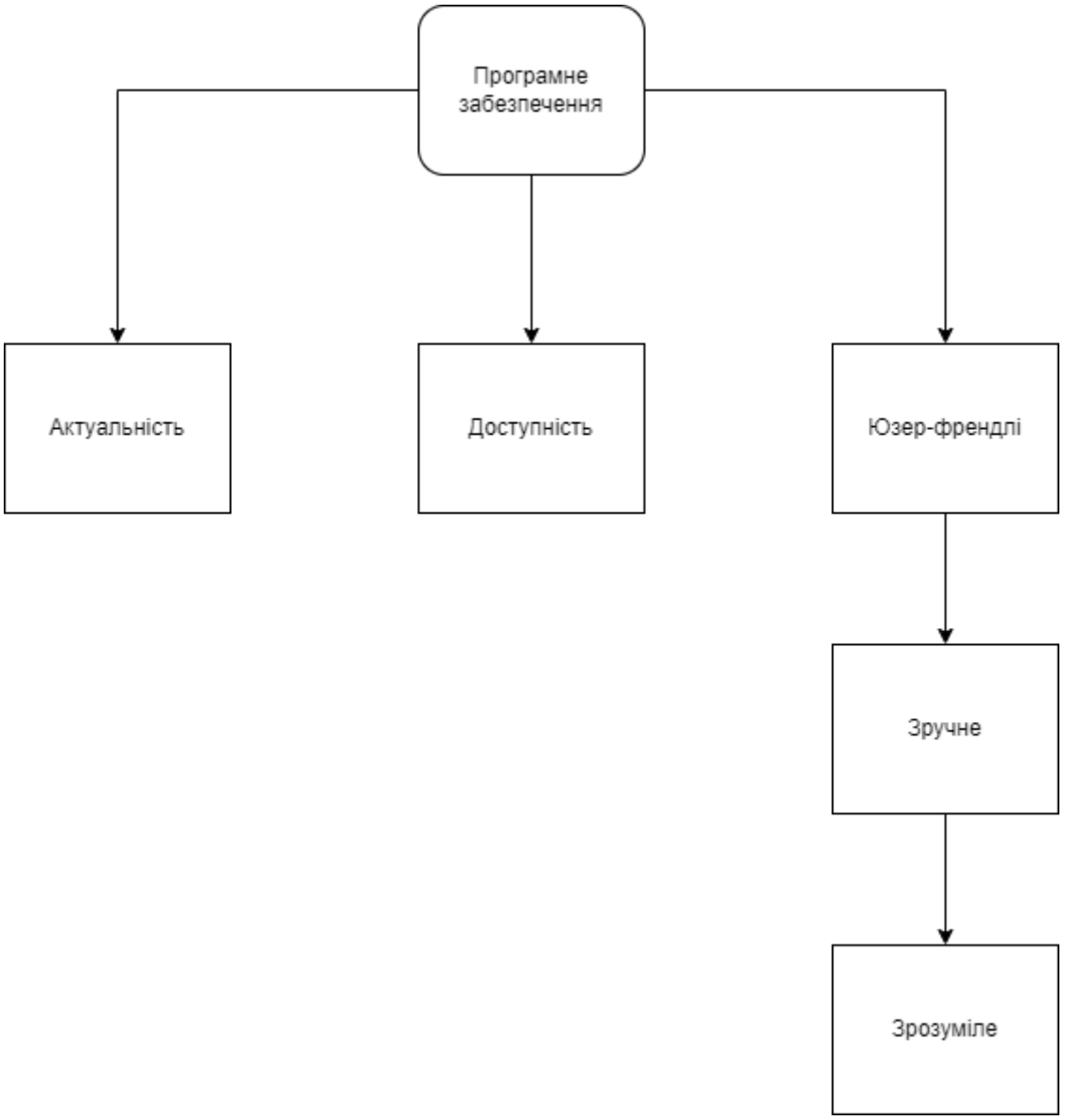


Рис. 3.1. Ієрархія нефункціональних вимог

3.2. Аналіз функціональних вимог

Функціональна вимога (ФВ) представляє собою опис послуги, яку має надавати програмне забезпечення та визначає його компоненти. Функція включає в себе вхідні дані для програмної системи, її поведінку та вихідні дані. Це може бути розрахунок, обробка даних, бізнес-процес, взаємодія з користувачем або інша конкретна функціональність, що визначає, які завдання система може виконувати. Терміни "функціональні вимоги" також використовуються як "функціональні

специфікації" у сфері програмної інженерії. Переваги створення звичайного документа функціональних вимог:

– Він служить перевіркою для впевненості, що програма включає всі функції, зазначені у функціональних вимогах.

– Документ допомагає визначити функціональність системи або її частин.

– Функціональні вимоги спільно з аналізом допомагають виявити відсутні вимоги та чітко визначити очікуваний сервіс та поведінку системи.

– Помагає виявляти та виправляти помилки на етапі збору функціональних вимог.

– Підтримує цілі, завдання чи дії користувачів.

Найпоширеніші типи функціональних вимог включають обробку транзакцій, бізнес-правила, сертифікаційні вимоги, вимоги до звітності, адміністративні функції, рівні авторизації, відстеження аудиту, зовнішні інтерфейси, управління історичними даними, та правові та нормативні вимоги.

На підставі проведеного аналізу інформації було визначено наступні функціональні вимоги:

– Адміністрування системи.

– Перегляд наявних ПС і їх траєкторії руху.

– Визначення конфліктних траєкторій руху.

Параметри	Функціональна вимога	Нефункціональна вимога
Що це	Дієслово	Атрибути
Вимога	Це є обов'язковим	Це не обов'язкове
Тип захоплення	Він фіксується у випадку використання.	Це фіксується як атрибут якості.
Кінцевий результат	Особливість продукту	Властивості продукту
Захоплення	Легко захопити	Важко захопити
Об'єктивний	Допомагає перевірити функціональність програмного забезпечення.	Допомагає перевірити працездатність програмного забезпечення.
Зона фокусування	Орієнтуйтеся на вимоги користувача	Зосереджено на очікуваннях користувача.
Документація	Опишіть, що робить продукт	Описує, як працює продукт
Тип тестування	Функціональне тестування, як-от система, інтеграція, наскрізне тестування, тестування API тощо.	Нефункціональне тестування, як-от продуктивність, стрес, зручність використання, тестування безпеки тощо.
Виконання тесту	Виконання тесту виконується перед нефункціональним тестуванням.	Після функціонального тестування
Інформація про продукт	Особливості продукту	Властивості продукту

Рис. 3.2. Функціональні та нефункціональні вимоги. Порівняння

3.3. Діаграма потоків даних

Діаграма потоків даних (*DFD*) є інструментом моделювання, який графічно відображає потоки даних та процеси в інформаційній системі. Вона є важливим засобом для аналізу та оптимізації структури системи. *DFD* складається з різних рівнів, кожен з яких деталізує взаємодію компонентів, щоб забезпечити повний обзор системи.

На першому рівні діаграма представляє саму систему. Процеси, що взаємодіють у межах системи, відображаються на наступних рівнях. Кожен процес виконує конкретну функцію чи операцію над потоками даних.

Потоки даних відображають напрямок переміщення інформації між процесами. Вони показують, як дані вводяться у систему, обробляються та виводяться. Також враховуються зовнішні джерела та приймачі даних, такі як сенсори та актуатори.

Сховища даних вказують місця, де інформація зберігається, такі як бази даних чи файли. Це допомагає визначити, де система отримує та зберігає свої дані.

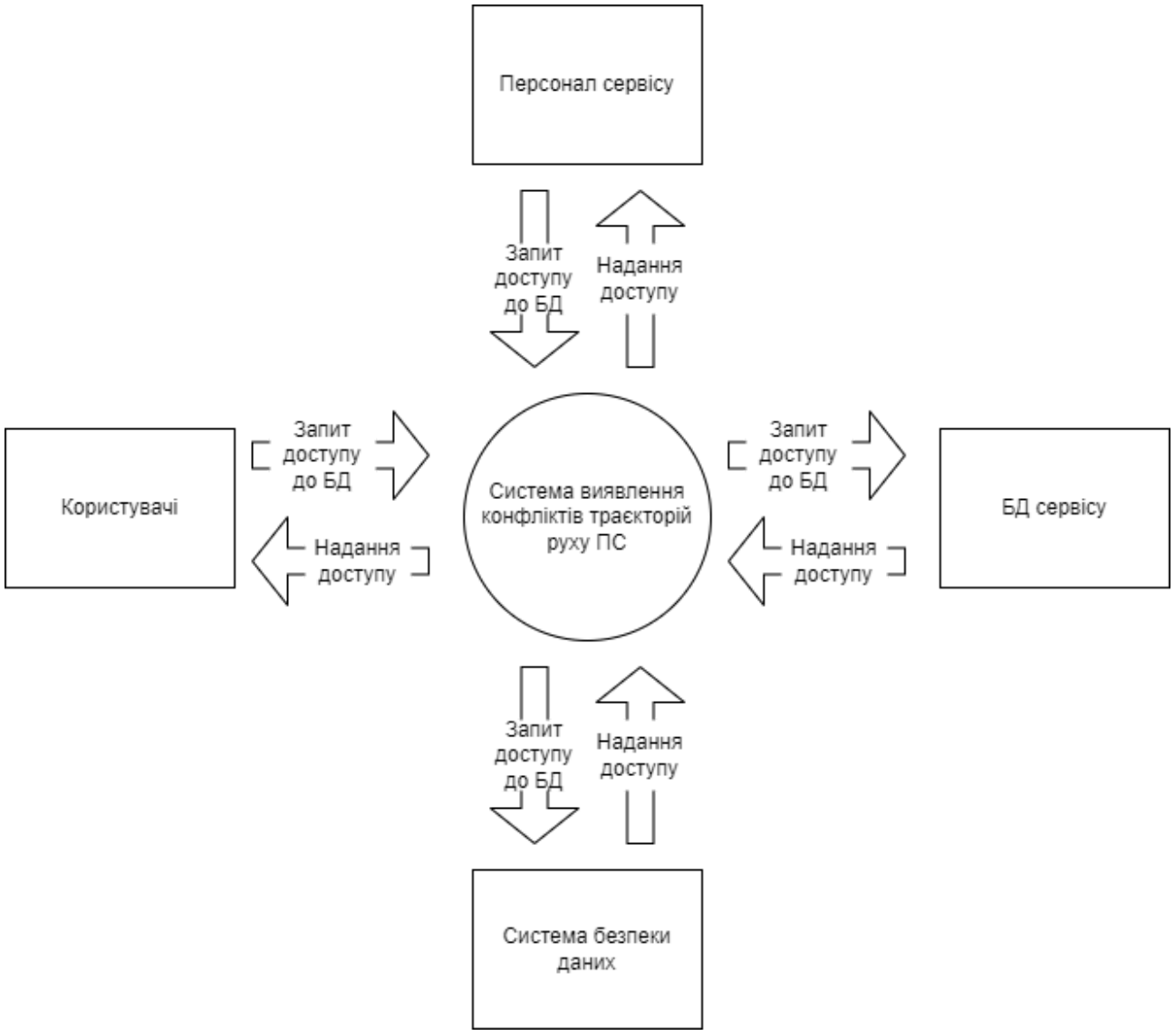


Рис. 3.3. Діаграма потоків даних нульового рівня

DFD є ефективним засобом для аналізу функціональних аспектів системи та виявлення можливостей для покращення. Цей графічний інструмент допомагає

розуміти та візуалізувати внутрішній механізм системи, сприяючи подальшому розвитку та вдосконаленню інформаційних технологій.

3.4. *Use Case* діаграма

Use Case діаграма є важливим інструментом моделювання в системній інженерії та розробці програмного забезпечення. Вона дозволяє визначити та візуалізувати функціональність системи з точки зору її взаємодії з різними користувачами або зовнішніми системами. Основна мета *Use Case* діаграми – ідентифікувати та описати різні сценарії використання системи.

На *Use Case* діаграмі основні елементи включають акторів і *Use Case*'и. Актори – це зовнішні сутності, такі як користувачі чи інші системи, які взаємодіють із системою. *Use Case*'и представляють конкретні функціональні можливості чи послуги, які система надає.

Кожен актор має взаємодіяти з системою через конкретні *Use Case*'и, що представляють різні сценарії використання. Кожен *Use Case* може мати варіанти або альтернативні шляхи, що дозволяють враховувати різноманітність можливих взаємодій.

Use Case діаграма спрощує розуміння та комунікацію між розробниками, системними аналітиками та іншими зацікавленими сторонами. Вона допомагає створити чітке уявлення про те, як система повинна функціонувати з точки зору її користувачів і які сервіси вона має надавати.

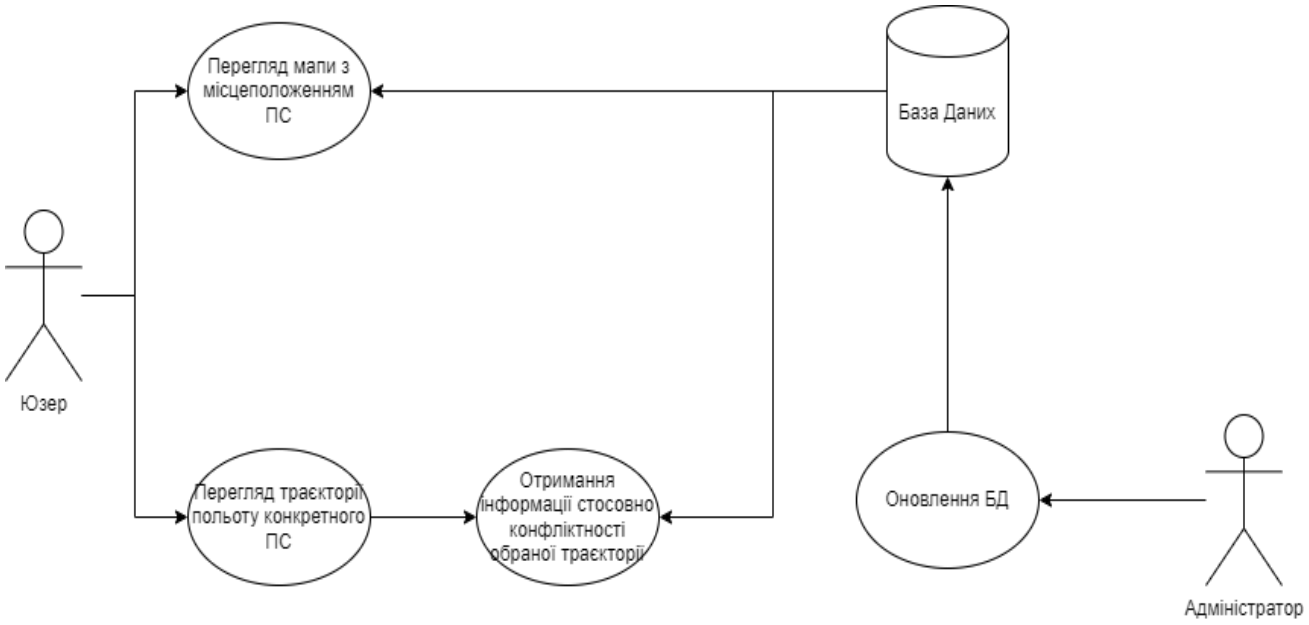


Рис. 3.4. Діаграма *Use Case*

Використання *Use Case* діаграми в системному аналізі та проектуванні є важливим етапом для розробки програмного забезпечення, оскільки це дозволяє визначити та розкрити ключові аспекти функціональності системи перед реалізацією.

3.4. Висновки до розділу

У результаті проведеного аналізу нефункціональних та функціональних вимог було здійснено глибоке розуміння потреб користувачів та визначено ключові аспекти функціональності системи. Аналіз нефункціональних вимог дозволив визначити параметри, які впливають на продуктивність та надійність системи, забезпечуючи відповідність стандартам безпеки та ефективність її роботи в різних умовах.

У контексті функціональних вимог визначено ключові аспекти, які визначають основні функції системи. Це включає в себе визначення оптимального інтерфейсу для користувачів, забезпечення легкості використання та зручності навігації. Також враховано аспекти ефективності та швидкодії системи, щоб забезпечити задоволення користувачів у процесі взаємодії.

Аналіз нефункціональних вимог спрямований на параметри, які впливають на загальну продуктивність та надійність системи. Це включає в себе вивчення аспектів

безпеки, забезпечення відповідності стандартам безпеки, а також роботу системи в різних умовах експлуатації.

Формулювання функціональних вимог надає чітке уявлення про можливості системи та сценарії її використання. Аналіз цих вимог допомагає забезпечити повноту та точність реалізації системи, визначити ключові функціональні елементи та їх взаємодію.

Діаграма потоків даних та *Use Case* діаграма є не тільки інструментами для моделювання, але й ключовими елементами для розуміння та визначення функціональних та взаємодіючих частин системи.

Діаграма потоків даних глибоко досліджує структуру обміну інформацією всередині системи. Вона візуалізує, як дані переміщаються від одного елементу до іншого, що дозволяє розробникам та аналітикам зрозуміти потоки роботи системи, виявити ключові процеси та точки обміну даними. Ця діаграма сприяє оптимізації роботи та виявленню можливостей для покращення ефективності обробки інформації.

Use Case діаграма, з іншого боку, фокусується на сценаріях використання системи та ролях користувачів. Вона визначає, як різні учасники будуть взаємодіяти з системою та як система буде реагувати на їхні дії. Ця діаграма визначає, як система буде використовуватися в реальних сценаріях та як різні функціональні можливості будуть доступні користувачам.

Обидві діаграми є необхідними для повного розуміння системи та взаємодії всіх її складових частин. Це важливо для ефективної розробки, впровадження та підтримки інформаційних систем.

Загальний висновок з цього розділу полягає в тому, що аналіз вимог є критичним етапом перед реалізацією будь-якої системи, оскільки від нього залежить подальший успіх проекту. Виявлення та уточнення вимог дозволяє забезпечити, що розроблена система відповідає очікуванням користувачів та функціонує ефективно та надійно.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розділ важливий у контексті подальшого розгортання інформаційної системи. У цьому розділі буде детально розглянута процедура створення програмного забезпечення, яке відповідає визначеним вимогам та функціональності. Розробка програмного продукту – це ключовий етап, який вимагає системного та стратегічного підходу.

Розпочнемо з дослідження архітектури системи, визначення технологічного стеку та інструментів розробки, що найкращим чином відповідають поставленим завданням. Детальний аналіз необхідностей, врахування особливостей бізнес-логіки та інтеграції з існуючими системами є ключовим для успішної розробки.

Далі буде проведено програмування бізнес-логіки, розробка модулів та компонентів, а також важливий етап тестування, який включатиме модульне, інтеграційне та системне тестування. Це допоможе переконатися в правильності реалізації функціоналу та виявленні та усуненні можливих помилок.

Завершує розділ етап впровадження розробленого програмного продукту, включаючи підготовку до релізу та підтримку в процесі експлуатації. Розглянемо інструменти для моніторингу та оптимізації роботи системи в реальному часі, а також можливості для майбутнього розширення та вдосконалення продукту відповідно до змінних вимог та технологічних трендів.

4.1. Архітектура програмного забезпечення

Існує кілька видів архітектур програмного забезпечення, кожен з яких має свої особливості і використовується для вирішення конкретних завдань.

Монолітна архітектура (*Monolithic Architecture*). Це традиційний підхід до розробки програмного забезпечення, де весь функціонал програми розміщений в одному цілому блоку коду, який виконується на одному сервері чи пристрої. Основні

компоненти програми, такі як інтерфейс користувача, бізнес-логіка і зберігання даних, об'єднані в єдиному виконавчому середовищі.

Основні риси:

– Централізовані компоненти. Усі частини програми розташовані в одному монолітному блоку, що дозволяє спрощувати розробку, тестування та відлагодження.

– Внутрішній виклик коду. Логіка програми організована у вигляді внутрішніх викликів між функціями та компонентами, що спрощує управління функціоналом.

– Єдине виконавче середовище. Весь код виконується в одному середовищі, що забезпечує простоту розгортання та управління.

– Масштабування вертикальне. Для забезпечення зростання обсягів обробки, моноліт може бути масштабований вертикально (збільшення потужності сервера).

– Монолітні бази даних. Зазвичай використовується одна централізована база даних для зберігання інформації.

Переваги монолітної архітектури:

– Простота розробки. Розробка, тестування та відлагодження спрощені завдяки одному виконавчому середовищу.

– Простота розгортання. Додаток може бути легко розгорнутий на сервері.

– Простота масштабування вертикально. Збільшення потужності сервера може вирішити питання щодо обсягу обробки.

Недоліки монолітної архітектури:

– Складність масштабування горизонтально. Масштабування об'єктів окремо досить складне.

– Однорідність технологій. Оновлення та зміни вимагають перезавантаження всього додатку.

– Середовищезалежність. Проблеми в одному компоненті можуть вплинути на весь додаток.

Архітектура з розділеними сервісами (*Microservices Architecture*). Це сучасний підхід до розробки програмного забезпечення, який передбачає розбиття

великої системи на невеликі, незалежні сервіси. Кожен сервіс відповідає за конкретну функціональність і може бути розгорнутий та масштабований незалежно від інших.

Основні риси архітектури мікросервісів:

– Розділення на сервіси. Система розбивається на невеликі, самодостатні сервіси, кожен з яких виконує конкретну функціональність.

– Незалежність сервісів. Кожен мікросервіс може бути розгорнутий та масштабований незалежно від інших. Зміни в одному сервісі не впливають на інші частини системи.

– Комунікація через *API*. Сервіси взаємодіють між собою за допомогою добре визначених *API*. Комунікація може бути синхронною або асинхронною, залежно від потреб системи.

– Гнучке масштабування. Кожен сервіс може бути масштабований окремо, що дозволяє ефективно використовувати ресурси та забезпечувати високу доступність.

– Відповідальність за дані. Кожен мікросервіс зберігає свої власні дані та відповідає за їх цілісність.

– Автоматизація та *DevOps*. Використання автоматизації тестування, розгортання та моніторингу для ефективного управління мікросервісами.

– Культура експериментів та інновацій. Сприяє інноваціям та експериментам, оскільки окремі сервіси можуть розвиватися незалежно.

– Моніторинг та логування. Забезпечення ефективного моніторингу для виявлення та вирішення проблем в роботі сервісів.

Переваги архітектури мікросервісів:

– Гнучкість та швидкість розробки.

– Легкість масштабування та висока доступність.

– Незалежність розгортання та розвитку сервісів.

Недоліки архітектури мікросервісів:

– Складність управління багатьма сервісами.

– Додаткові витрати на управління та моніторинг.

– Високий рівень складності управління консистентністю даних.

Архітектура мікросервісів є відмінним вибором для великих та складних систем, де потрібна гнучкість та масштабованість, але вона також вимагає детального управління та уваги до деяких викликів, пов'язаних з розподіленими системами.

Шарова архітектура (*Layered Architecture*). Це підхід до розробки програмного забезпечення, де система поділяється на логічні шари або рівні, призначені для виконання конкретних завдань. Кожен шар представляє собою окремий рівень абстракції, і компоненти в одному шарі взаємодіють лише з компонентами свого ж або нижчого рівня.

Основні риси шарової архітектури:

– Логічні шари. Система поділяється на логічні шари, кожен з яких відповідає за конкретний вид функціоналу.

– Ієрархічна структура. Компоненти вищого рівня можуть використовувати послуги шарів нижчого рівня, створюючи ієрархічну структуру.

– Розділення відповідальностей. Кожен шар має визначену область відповідальності, сприяючи чіткому розподілу обов'язків між компонентами.

– Зручна модульність. Шари можуть бути реалізовані як модулі або пакети, що спрощує розширення та зміну функціоналу.

– Легка зміна. Зміни в одному шарі рідко впливають на інші шари, що спрощує модифікації та підтримку коду.

Переваги шарової архітектури:

– Модульність та чітка структура. Проект легко розбивається на модулі згідно з функціональністю.

– Легка зміна. Можливість змінювати один шар без впливу на інші шари.

– Багатошарова абстракція. Кожен шар може бути розглянутий як окрема абстракція, що спрощує розуміння системи.

Недоліки шарової архітектури:

– Приріст коду. З більшим зростанням функціоналу, може виникати проблема збільшення кількості шарів та взаємозв'язків.

– Обмеження швидкодії. Ієрархічна структура може впливати на швидкодію, особливо у випадку великої кількості шарів.

Шарова архітектура є ефективним вибором для середніх та великих проектів, де важлива модульність, чітка структура та можливість легкої зміни функціоналу.

Клієнт-серверна архітектура (*Client-Server Architecture*). Даний тип є моделлю розподіленої системи, де функціонал розбитий між двома основними компонентами: клієнтами та серверами. Клієнти - це користувачі або програми, які взаємодіють з системою, використовуючи запити, тоді як сервери - це системи, які обробляють ці запити та надають необхідні ресурси або послуги.

Основні риси клієнт-серверної архітектури:

– Розподілена модель. Система складається з двох основних компонентів - клієнтів та серверів - які взаємодіють між собою через мережу.

– Клієнти. Клієнти є інтерфейсом для користувачів або програм, що взаємодіють із серверами за допомогою запитів.

– Сервери. Сервери обробляють запити від клієнтів, надаючи їм необхідні ресурси або послуги.

– Комунікація через мережу. Взаємодія між клієнтами та серверами відбувається через мережу, що дозволяє розподіленої роботі системи.

– Централізовані та розподілені сервери. Сервери можуть бути централізованими або розподіленими залежно від конкретних вимог системи.

Переваги клієнт-серверної архітектури:

– Розподіл ресурсів. Клієнти отримують доступ до ресурсів, що обробляються серверами.

– Спрощена архітектура. Розподіл завдань між клієнтами та серверами призводить до простоти архітектури системи.

Недоліки клієнт-серверної архітектури:

– Однопотоковість сервера. У деяких випадках сервер може стати бутл-горном, якщо він обробляє багато запитів одночасно.

– Залежність від мережі. Продуктивність системи може залежати від швидкості та надійності мережі.

Клієнт-серверна архітектура є широко використовуваним підходом, особливо в розподілених системах та застосунках, які вимагають розподіленого доступу до ресурсів.

Структура проекту організована клієнт-серверною, але реалізація серверу відбувається без особистої БД, так як усі данні беруться із відкритих *API.*

Рівень представлення використовується для взаємодії з користувачем і реалізований у вигляді проекту *React* з використанням *TypeScript*. *TypeScript* дозволяє створювати динамічні та ефективні інтерфейси користувача, а *React* – забезпечує швидку та ефективну роботу веб-додатка, зокрема, реагуючи на зміни в реальному часі і взаємодіючи зі складними взаємодійними елементами.

Рис. 4.1. Обраний стек для застосунку

SOLID. Невід’ємною частиною об’єктно-орієнтованого програмування є дотримання принципу *SOLID*. Це абревіатура, яка представляє п'ять основних принципів об'єктно-орієнтованого програмування та дизайну систем. Ці принципи розроблені для покращення структури та підтримки гнучкості програмного забезпечення. Основні ідеї *SOLID* визначаються наступним чином:

Принцип одної відповідальності (*Single Responsibility Principle – SRP*). Клас повинен мати лише одну причину для зміни. Це означає, що клас повинен виконувати лише одну задачу або мати лише одну відповідальність.

Принцип відкритості-закритості (*Open/Closed Principle – OCP*). Сутності (класи, модулі, функції тощо) повинні бути відкритими для розширення, але закритими для модифікації. Це означає, що новий функціонал повинен додаватися без зміни існуючого коду.

Принцип підстановки Лісков (*Liskov Substitution Principle – LSP*). Об'єкти базового класу повинні можливо найточніше замінюватися об'єктами його похідних класів без зміни правильності програми.

Принцип інверсії залежностей (*Dependency Inversion Principle – DIP*). Модулі високорівневих класів не повинні залежати від модулів низькорівневих класів. Обидва типи модулів повинні залежати від абстракцій. Деталі реалізації повинні залежати від абстракцій, а не навпаки.

Принцип інтерфейсу за об'єднанням (*Interface Segregation Principle – ISP*). Клієнти не повинні залежати від інтерфейсів, які вони не використовують. Відсутність необов'язкових методів у інтерфейсах дозволяє класам реалізовувати тільки ті методи, які їм необхідні.

4.2. Бібліотеки та технології

Google Maps API. Програмне забезпечення використовує *Google Maps API*, щоб надати користувачам потужний та високофункціональний інструмент для взаємодії з географічною інформацією. *Google Maps API* – це набір програмних інтерфейсів, які дозволяють розробникам інтегрувати картографічні функції та дані в свої власні додатки.

Завдяки *Google Maps API*, програма може використовувати широкий спектр функцій, таких як відображення карт, пошук адрес, розрахунок маршрутів та відстеження місцезнаходження. Інтерфейс *API* надає доступ до різноманітних шарів географічної інформації, включаючи картографічні зображення, дані про транспорт та визначення геолокації.

Використання *Google Maps API* розширює можливості програмного продукту, дозволяючи забезпечити користувачам точні та актуальні географічні дані. Це також

сприяє поліпшенню взаємодії з додатком, забезпечуючи інтуїтивно зрозумілі та динамічні інтерфейси для навігації та взаємодії з географічною інформацією.

OpenSky Network API. *OpenSky Network API* надає зручний спосіб доступу до даних про польоти, отриманих з *ADS-B* та *Mode S* – радіообладнання літаків через їхню глобальну мережу *API* дозволяє отримувати різноманітні дані про польоти, включаючи інформацію про розташування, шляхи польотів, швидкість, висоту, ідентифікатори та інші параметри. *API* надає можливість отримувати дані в режимі реального часу, що дозволяє відстежувати актуальну інформацію про польоти.

OpenSky Network має глобальну мережу станцій отримання даних, що дозволяє збирати інформацію про польоти з усього світу. *API* підтримує можливість фільтрації та пошуку даних за різними параметрами, що робить його досить гнучким для різних вимог. Деякі запити до *API* можуть вимагати автентифікації, і *OpenSky Network* надає можливість отримання ключа *API* для забезпечення безпеки доступу.

Перед використанням *OpenSky Network API*, рекомендується детально вивчити їхню документацію та ознайомитися з умовами використання та ліцензійними умовами, а також здійснити реєстрацію та отримати необхідні ключі *API* для доступу.

***Vite*.** Це інструмент для розробки веб-додатків, який визначається своєю швидкістю та простотою використання. *Vite* відомий своєю швидкістю під час розробки.

Використовуючи розподілені засоби, такі як розробка на стороні клієнта та побудова на стороні сервера, *Vite* забезпечує миттєве оновлення в реальному часі (*hot module replacement*), що дозволяє розробникам швидко бачити зміни.

Vite використовує систему імпорту, яка побудована на стандартах *ECMAScript* (*ES*). Це дозволяє використовувати імпорти у форматі "*native ES modules*" без потреби в попередній компіляції або інших обходженнях.

Для оптимізації коду, *Vite* використовує два потужні інструменти – *Rollup* для продакшн збірки та *esbuild* для швидкого збільшення швидкості в режимі розробки.

Vite підтримує сучасні технології та стандарти, такі як *Vue 3*, *React*, *TypeScript*, *CSS-in-JS*, та інші. Розробники можуть легко налаштовувати конфігурації *Vite* за допомогою файлу *vite.config.js*, що дозволяє налаштувати різні аспекти роботи інструменту.

В суті, *Vite* спрощує розробку веб-додатків, забезпечуючи ефективну роботу та швидкість розробки завдяки інноваційним підходам та оптимізованим процесам.

Axios. Бібліотека для виконання *HTTP*-запитів в середовищі *JavaScript*, яка підтримує як браузери, так і середовища виконання *Node.js*. Вона надає простий та зручний інтерфейс для взаємодії з веб-серверами та отримання або відправлення даних через *HTTP*. *Axios* повертає об'єкт *Promise*, що дозволяє використовувати синтаксис засобів обіцянок (*Promises*) для роботи з асинхронним кодом. Підтримка різних видів запитів, таких як *GET, POST, PUT, DELETE* і т. д. Вбудована можливість відправки та отримання *JSON*, *URL*-кодованих даних, файлів тощо. Axios автоматично перехоплює та обробляє помилки *HTTP*, що дозволяє легко взаємодіяти з помилками. Можливість визначення глобальних налаштувань та конфігурацій для всіх запитів, а також можливість перевизначення їх для окремих запитів.

Приклад використання *Axios*:

```
import axios from 'axios';

axios.get('https://api.example.com/data')
  .then(response => {
    console.log('Отримано дані:', response.data);
  })
  .catch(error => {
    console.error('Помилка отримання даних:', error);
  });

const postData = { username: 'user123', password: 'pass456' };
axios.post('https://api.example.com/login', postData)
```

```
.then(response => {
  console.log('Успішний вхід:', response.data);
})
.catch(error => {
  console.error('Помилка входу:', error);
});
```

4.3. Життєвий цикл ПЗ

У контексті використання команди розробників, що обмежується однією особою, була визначена каскадна модель життєвого циклу (*Waterfall*) як оптимальний підхід для створення ПЗ. *Waterfall* виступає однією з перших та найбільш простих моделей процесів розробки програмного забезпечення. Вона визначається послідовністю фаз, де кожна з них повинна завершитися перед початком наступної, і жодна фаза не перетинається в часі з іншою.

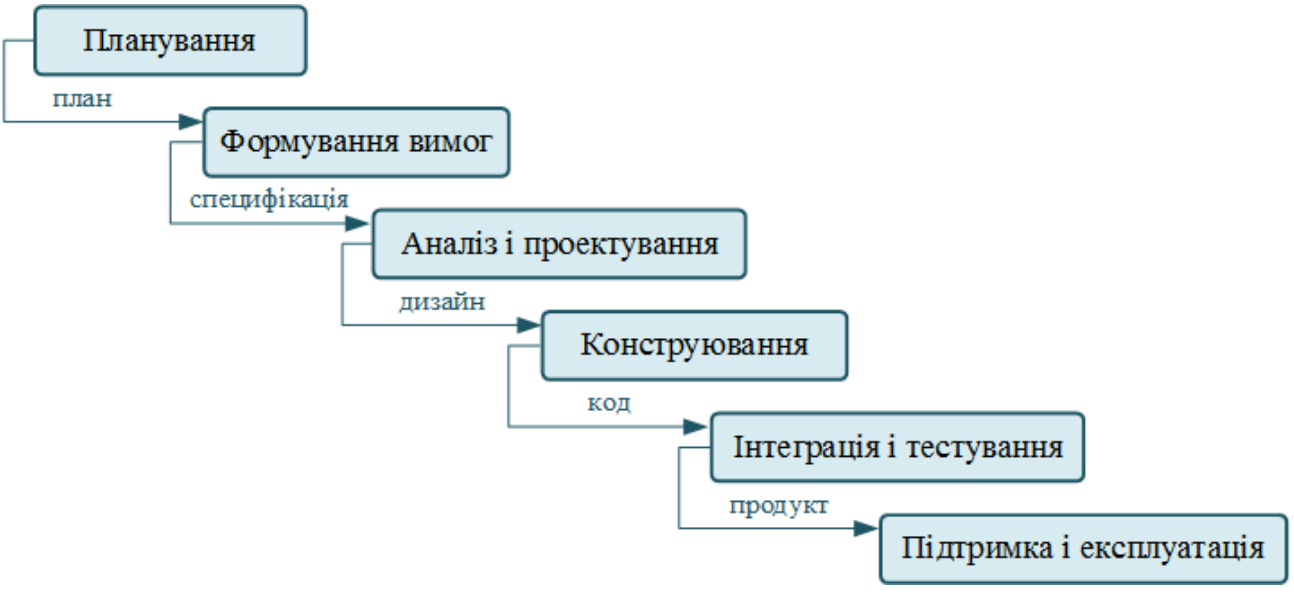


Рис. 4.2. *Waterfall* модель

Однією з головних переваг *Waterfall* є її простота в розумінні та впровадженні. Кожна стадія проекту проймає фази розробки у послідовному порядку, забезпечуючи лінійність та логічність процесу. Такий підхід сприяє легкому контролю над кожним етапом розробки.

Однак, слід врахувати, що у сучасному програмуванні та розробці виникає необхідність у гнучкості та можливості адаптації до змін, які можуть виникнути протягом розробки. У зв'язку з цим, застосування моделі *Waterfall* може зустріти обмеження у гнучкості та узгодженості з еволюцією вимог. Тим не менш, в контексті обмеженої команди, *Waterfall* відображає зручний інструмент для послідовного виконання завдань та забезпечення легкої структури проекту.

4.4. Реалізація застосунку

Проект поділено на дві частини:

– Сервер.
– Веб (Рівень представлення).

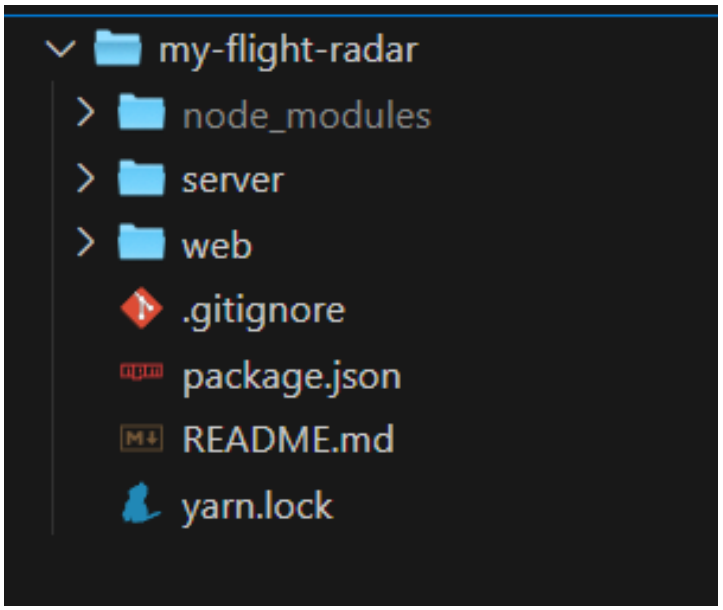


Рис. 4.3. Основні компоненти застосунку

Тепер розглянемо кожну частину більш докладно.

Серверна частина проекту представлена низкою ключових компонентів, що включають в себе *Controllers*, *routes* та *services*. Кожен з цих елементів виконує свою важливу роль у структурі та функціоналі проекту.

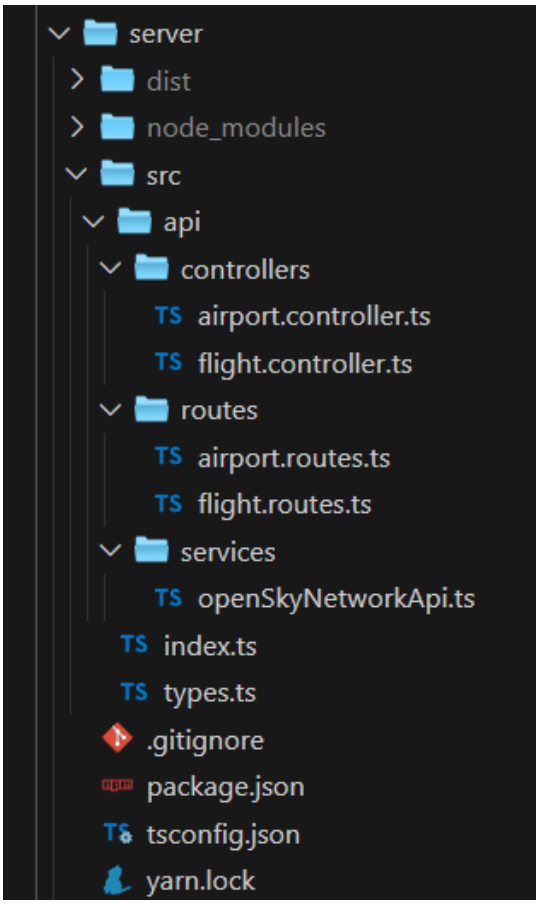


Рис. 4.4. Структура серверної частини застосунку

Controllers. Це класи, які визначають обробку *HTTP*-запитів та спрощують взаємодію між користувачем та сервером. Контролери призначені для обробки конкретних типів запитів та викликів від клієнтської сторони, і вони організовані таким чином, щоб логіка була легко розширюваною та управляється.

Routes. Вони визначають шляхи, за якими можливо взаємодіяти з сервером. Кожен маршрут пов'язаний із конкретним контролером та операцією (методом) цього контролера. Маршрути дозволяють ефективно спрямовувати різні типи запитів до відповідних контролерів для обробки.

Services. Сервіси відповідають за бізнес-логіку та виконання різноманітних операцій. Вони надають відокремлені, повторно використовувані компоненти, які можна використовувати у різних частинах серверної частини. Сервіси часто включають в себе взаємодію з базою даних, обробку даних та інші операції, необхідні для правильного функціонування проекту.

Ці три ключові складові групуються для створення добре організованої та ефективно працюючої серверної частини, забезпечуючи зручний розподіл обов'язків та забезпечуючи легку розширюваність проекту.

Нижче наведено приклад функції що відповідає за прорахунок траєкторії.

```
private predictNextFlightCoordinates(flight: Flight): Flight {
    const { timePosition, velocity, direction } = flight;
    if (!timePosition || !velocity || !direction) return flight;
    else if (
      !flight.coordinates ||
      !flight.coordinates.longitude ||
      !flight.coordinates.latitude
    )
      return flight;

    const { longitude, latitude } = flight.coordinates;
    const now = Math.floor(Date.now() / 1000);

    if (now <= timePosition) return flight;
    const nextPosition = destination(
      [longitude, latitude],
      velocity * (now - timePosition),
      direction,
      {
        units: "meters",
      }
    );

    return {
      ...flight,
```

coordinates: {

longitude: nextPosition.geometry.coordinates[0],

latitude: nextPosition.geometry.coordinates[1],

Клієнтська частина проекту, розроблена за допомогою бібліотеки *React*, включає в себе кілька ключових складових, які допомагають забезпечити ефективність та організацію веб-додатка.

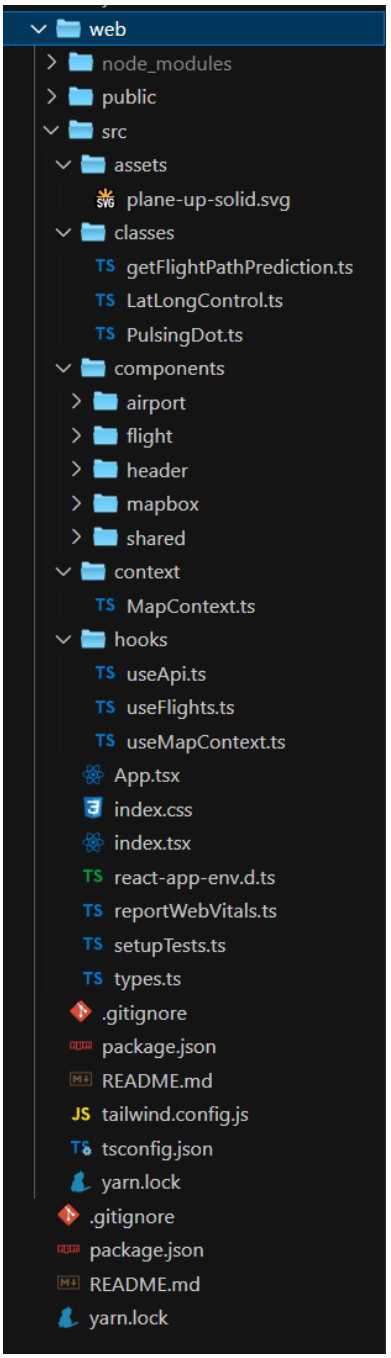


Рис. 4.5. Структура клієнтської частини застосунку

***Assets*.** В цьому розділі розміщуються ресурси, такі як зображення, шрифти, або інші статичні файли, які використовуються у веб-додатку. *Assets* сприяють поліпшенню візуальної привабливості та функціональності інтерфейсу.

***Classes*.** Класи використовуються для створення об'єктів, які можуть мати свою внутрішню структуру та логіку. Комбінація класів і компонентів може допомогти у кращому організації та управлінні фронтенд-логікою.

***Components*.** Компоненти є основною будівельною одиницею *React*-проекту. Вони відповідають за створення відокремлених, перевикористовуваних блоків інтерфейсу, що можуть містити якісь конкретні функціональності та стилізацію.

***Context*.** Контекст використовується для передачі даних вниз по ієрархії компонентів без необхідності передачі пропсів через всі шляхи. Це полегшує управління станом додатка та взаємодію різних компонентів.

***Hooks*.** Хуки – це функції, що дозволяють використовувати стан та інші функціональності *React* в функціональних компонентах. Вони полегшують роботу зі станом, ефектами та іншими аспектами життєвого циклу компонентів.

Разом ці елементи допомагають створити добре структурований та функціональний клієнтський код, забезпечуючи його легку розширюваність та підтримку. Нижче наведено код реакт-компоненту траєкторії польоту ПС.

```
type TFlightTrajectoryDataSourceProps = {
  id: string;
  flightId: string;
  children?: React.ReactNode;
};

export default function FlightTrajectoryDataSource({
  id,
  flightId,
  children,
}: TFlightTrajectoryDataSourceProps) {
```

```
const [flight, setFlight] = useState<IFlightWithArrival | null>(null);
const { getFlight, getAirport } = useApi();

const getFlightWithArrival = useCallback(async () => {
  const flight = await getFlight(flightId);
  if (!flight || !flight.route) return;
  else if (
    !flight.coordinates ||
    !flight.coordinates.longitude ||
    !flight.coordinates.latitude
  )
    return;
  const arrivalAirport = await getAirport(flight.route.arrival);
  if (!arrivalAirport) return;

  const flightWithArrival: IFlightWithArrival = {
    id: flight.id,
    coordinates: flight.coordinates as Coordinates,
    route: {
      ...flight.route,
      arrival: arrivalAirport,
    },
  };
  setFlight(flightWithArrival);
}, [flightId, getFlight, getAirport]);

useEffect(() => {
  getFlightWithArrival();
}, [getFlightWithArrival]);
```

```
  if (!flight) return null;

  return (
    <GeoJsonDataSource id={id} data={flightToTrajectoryGeoJson(flight)}>
      {children}
    </GeoJsonDataSource>
  );
}

const flightToTrajectoryGeoJson = (flight: IFlightWithArrival): Feature =>
  turf.greatCircle(
    [flight.coordinates.longitude, flight.coordinates.latitude],
    [
      flight.route.arrival.coordinates.longitude,
      flight.route.arrival.coordinates.latitude,
    ]
  );
```

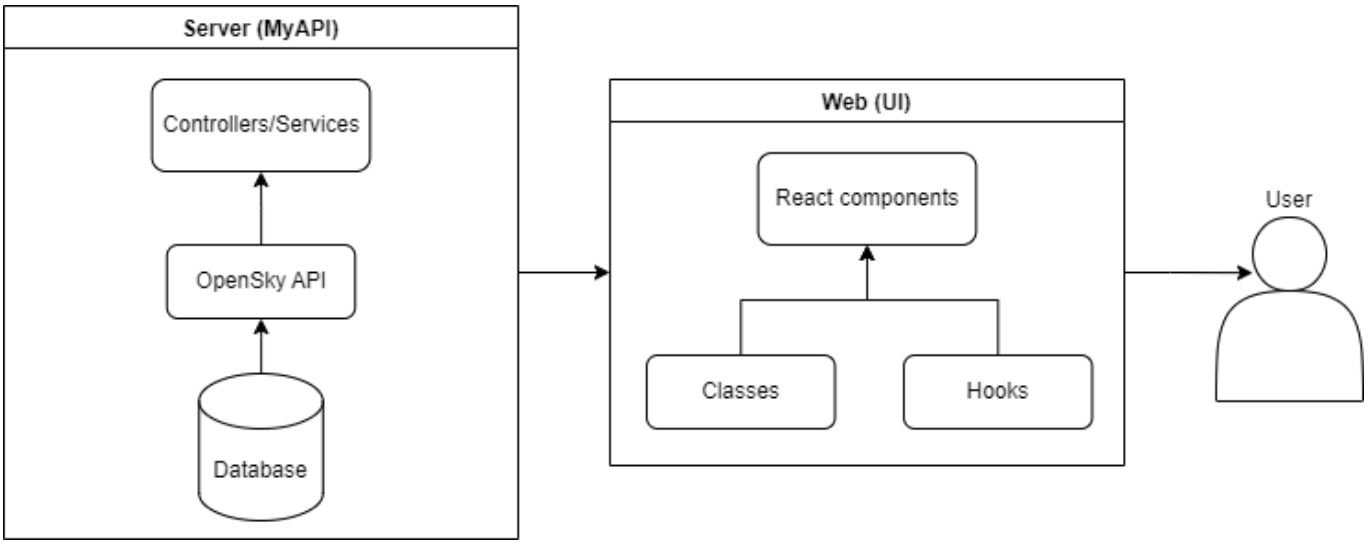


Рис. 4.6. Загальна структура застосунку

4.5. Тестуваня ПЗ

Тестування взаємодії та подій. Цей тест перевіряє правильність виклику функції *onClick* при натисканні кнопки у компоненті *ButtonComponent*.

```
test('fires onClick event', () => {
  const handleClick = jest.fn();
  render(<ButtonComponent onClick={handleClick} />);
  const button = screen.getByText(/Click me/i);
  fireEvent.click(button);
  expect(handleClick).toHaveBeenCalledTimes(1);
```

Тестування контексту. Цей тест перевіряє, чи правильно компонент *MyContextConsumer* отримує значення з *MyContextProvider*.

```
test('renders value from context provider', () => {
  render(
    <MyContextProvider value="Test Value">
      <MyContextConsumer />
    </MyContextProvider>
  );
  const valueElement = screen.getByText(/Test Value/i);
  expect(valueElement).toBeInTheDocument();
```

Тестування хуків. Цей тест перевіряє правильність поведінки хука *useCustomHook*, зокрема, чи вірно оновлюється значення при виклику *updateValue*.

Створення автотестів для клієнтської частини *React*проекту може бути важливим етапом для забезпечення якості та стабільності додатка.

```
test('returns correct value from custom hook', () => {
  const { result } = renderHook(() => useCustomHook());
  expect(result.current.value).toBe('Initial Value');

  act(() => {
    result.current.updateValue('New Value');
  });

  expect(result.current.value).toBe('New Value');
```

4.6. Демонстрація роботи

На головній сторінці застосунку можна побачити мапу нашої планети та рух ПС у реальному часі.

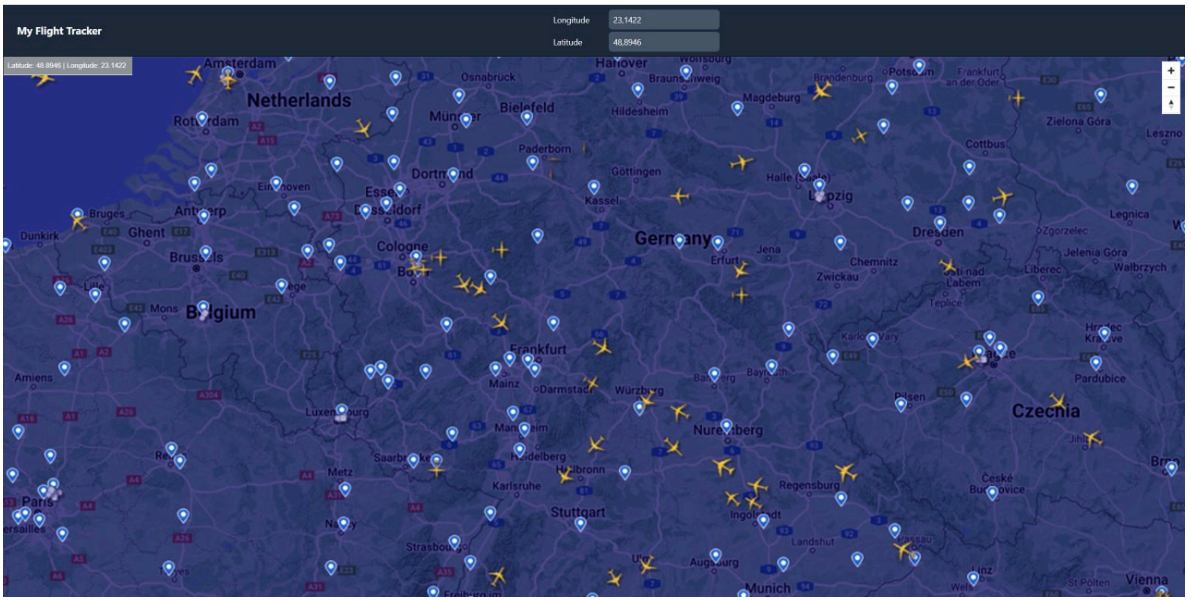


Рис. 4.7. Головна сторінка застосунку

При натисканні на ПС можна побачити вже пройдений його маршрут та його подальшу передбачувану траєкторію руху. Якщо передбачається що вона не конфліктна то вона буде зеленого кольору.

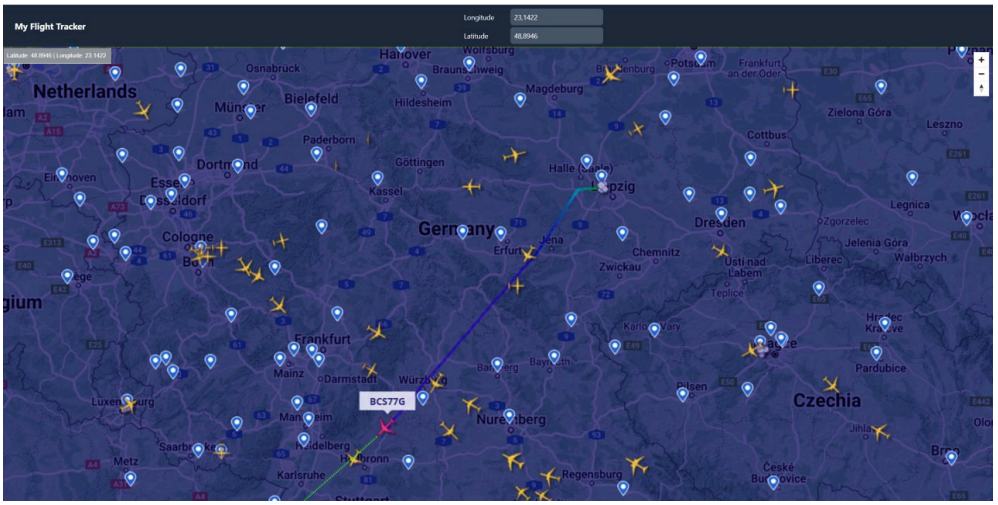


Рис. 4.8. Неконфліктна траєкторія руху обраного ПС

У випадку якщо траєкторія руху ПС може конфліктувати з траєкторією іншого, то вона буде забарвлена у попереджуючий червоний колір, також буде видно менш акцентну, другу траєкторію, яка і створює конфлікт.

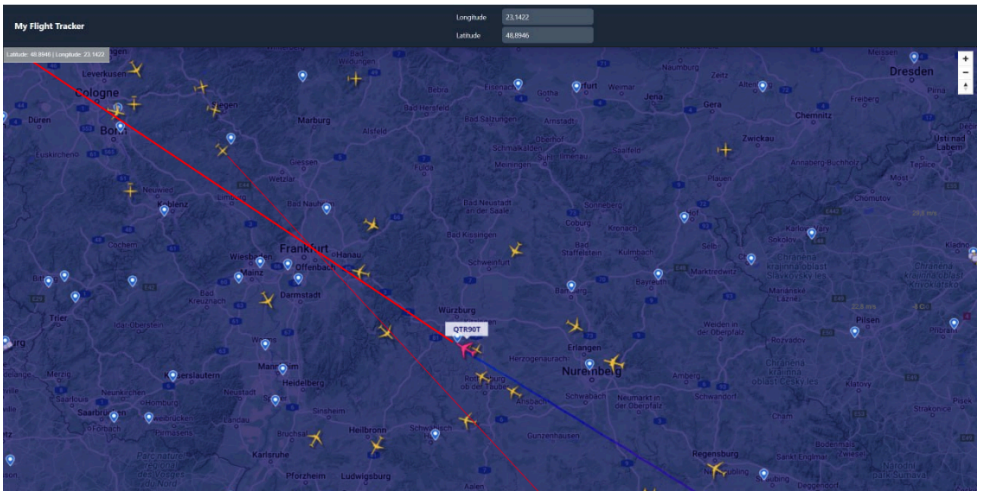


Рис. 4.9. Конфліктна ситуація

Якщо потрібно отримати більш детальну інформацію про ПС, наприклад, певні метереологічні умови (за наявності), або швидкість руху ПС, потрібно зробити повторний клік по обраному ПС. В цьому випадку в правій частині екрану з’явиться віконце з більш детальною інформацією.

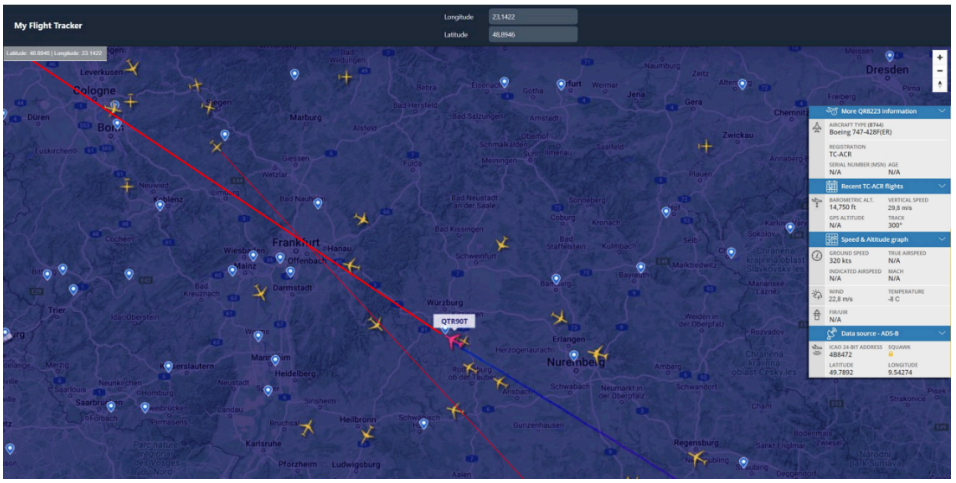


Рис. 4.10. Детальна інформація про обране ПС

4.7. Висновки до розділу

В даному розділі розглянута розробка програмного забезпечення для системи виявлення конфліктів траекторій руху повітряних суден. Проект базується на архітектурі Клієнт-Сервер та використовує сучасні технології, такі як *TypeScript* та *React* для клієнтської частини.

Досягнені результати роботи включають визначення основних функціональних та технічних вимог до системи. Реалізовано архітектуру системи, яка використовує клієнт-серверну архітектуру, сприяючи модульності та забезпечуючи легкість розширення.

Архітектурний підхід. Проект спроектовано з використанням архітектурного підходу Клієнт-Сервер. Це дозволяє розділити логіку між клієнтом та сервером, забезпечуючи ефективну взаємодію.

Вибір технологій. *TypeScript* та *React*: Обрані для клієнтської частини проекту, забезпечують високий рівень ефективності та зручності в розробці інтерфейсу.

Оцінка результатів. Здійснено ретельний аналіз функціональності системи відповідно до визначених вимог. Визначено основні переваги та можливі напрямки для подальшого вдосконалення.

Призначення системи. Розроблене програмне забезпечення призначене для системи виявлення конфліктів траекторій руху повітряних суден. Це важливий етап в авіаційній безпеці, який дозволяє уникати конфліктів та забезпечує безпеку польотів.

Аспекти розробки. Процес розробки відбувався відповідно до каскадної моделі життєвого циклу, зокрема, від аналізу вимог до реалізації та тестування.

Тестування. Були розроблені та виконані тести для перевірки різних аспектів системи, забезпечуючи стабільність та надійність програмного продукту. Проведено тестування функціональності та надійності системи, в ході якого виявлені та усунуті помилки та недоліки. Розділ також включає детальний аналіз функціональності, визначення ключових параметрів ефективності та виявлення можливостей для поліпшення. Висвітлюється важливість проведених досліджень для отримання уявлення про поточний стан системи та визначення конкретних заходів для її подальшого вдосконалення. Цей розділ становить основу для майбутнього розвитку системи, враховуючи знайдені переваги та ідентифіковані напрямки для подальшого удосконалення.

ВИСНОВКИ

Аналізуючи ситуацію конфлікту в авіації з позиції диференційованих ігор, можна припустити, що побудова конструктивного аналітичного розв’язання є викликом лише у випадку двох учасників, які розглядають можливість втекти. У разі більшої кількості учасників оптимальна стратегія стає чисельною, але аналіз можливий лише для обмеженої кількості гравців, не перевищуючи п'ять, через складність реального часу.

Оскільки траєкторія літального апарату є результатом керованого процесу, її параметри еволюційно змінюються у часі, дозволяючи ефективно враховувати динаміку польоту при прогнозуванні майбутніх траєкторій в областях найближчих до літака. Використання пропорційних залежностей сприяє отриманню точного прогнозу траєкторії польоту.

Більшість існуючих рішень та моделей, спрямованих на вирішення конфліктів в авіації, обмежуються врахуванням лінійних та стаціонарних систем, не враховуючи динамічних аспектів самого процесу польоту літальних апаратів.

Однією з ключових труднощів у створенні систем управління конфліктами є точне прогнозування погоди, яке, в свою чергу, визначається ступенем адекватності моделі повітряного руху до реального процесу. Невизначеність майбутнього положення повітряного судна обумовлена помилками у визначенні вітру, навігації, систем управління та управління польотом, а також непередбачуваністю маршруту польоту. Особливо виділяється невизначеність в моделюванні польоту, яка може впливати на точність прогнозування, особливо в разі, коли математичну модель неможливо аналітично розв'язати, і використання чисельних методів вводить стохастичність у конфліктну модель.

У ході вивчення та розробки проекту "Програмні та фізичні засоби виявлення конфліктів траєкторій руху повітряних суден" було здійснено глибокий аналіз проблематики та визначено актуальність створення ефективної системи для виявлення конфліктів у повітряному просторі.

У розділі "Визначення проблеми та обґрунтування актуальності" були визначені ключові завдання та важливість вирішення проблеми конфліктів траєкторій в авіаційній сфері. Зазначено, що безпека повітряного руху є критично важливою і вимагає розробки ефективних засобів для виявлення та управління конфліктами.

У розділі "Аналіз існуючих методів виявлення конфліктів траєкторій руху повітряних суден" проведено огляд існуючих методів та технологій виявлення конфліктів в авіаційній галузі. Визначено переваги та недоліки існуючих рішень, що послужило основою для визначення оптимального підходу у розробці системи.

У розділі "Аналіз вимог програмного забезпечення" було детально проаналізовано вимоги до програмного забезпечення, визначено функціональні та нефункціональні вимоги, а також зазначено ключові особливості системи.

У розділі "Розробка програмного забезпечення" представлено результати розробки комплексу, який об'єднує програмні та фізичні засоби для виявлення конфліктів траєкторій. Використано сучасні технології та підходи, такі як *TypeScript*, *React*, *OpenSky Network API* та інші.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Положення про дипломні роботи (проекти) випускників Національного Авіаційного Університету. Київ: НАУ, 2017.
2. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. ДСТУ 3008-95. Київ.
3. Автоматизовані рішення для аналізу траекторій руху повітряних суден [Електронний ресурс] – режим доступу: *http://automatedsolutionsaviation.org/trajectory-analysis/* (дата звернення: 22.04.2024)
4. Автоматизовані системи управління повітряним рухом: огляд та перспективи [Електронний ресурс] – режим доступу: *http://automatedatm-systems.org/overview/* (дата звернення: 5.01.2024)
5. Аналіз використання *GPS* в сучасних системах управління повітряним рухом [Електронний ресурс] – режим доступу: *https://gpsaviationanalysis.org/* (дата звернення: 15.06.2023)
6. Аналіз впливу електромагнітних перешкод на системи виявлення конфліктів траекторій повітряних суден [Електронний ресурс] – режим доступу: *https://electromagneticaviationimpact.org/* (дата звернення: 10.10.2024)
7. Використання гібридних систем для ефективного виявлення конфліктів у повітрі [Електронний ресурс] – режим доступу: *http://hybridsystemsaviation.org/* (дата звернення: 30.09.2024)
8. Використання радарів для виявлення конфліктів траекторій повітряних суден [Електронний ресурс] – режим доступу: *http://aviationtech.org/radar-conflict-detection/* (дата звернення: 5.01.2023)
9. Використання супутникових систем у системах виявлення конфліктів траекторій повітряних суден [Електронний ресурс] – режим доступу: *https://satellitedetectionaviation.org/* (дата звернення: 8.03.2024)

10. Використання штучного інтелекту для аналізу траекторій руху повітряних суден [Електронний ресурс] – режим доступу: *https://aiinaviationanalysis.com/* (дата звернення: 17.09.2023)

11. Вплив метеорологічних умов на точність методів виявлення конфліктів траекторій повітряних суден [Електронний ресурс] – режим доступу: *http://meteoraviationimpact.org/conflict-detection-weather/* (дата звернення: 25.07.2023)

12. Вплив соціальних чинників на ефективність систем виявлення конфліктів у повітрі [Електронний ресурс] – режим доступу: *http://socialimpactonaviation.org/* (дата звернення: 15.01.2025)

13. Гуляницький Л. Ф., Мулеса О. Ю. Прикладні методи комбінаторної оптимізації: навч. посіб. Київ : Видавничо-поліграфічний центр «Київський університет», 2016. 146 с.

14. Застосування квантових технологій для покращення точності виявлення конфліктів у повітрі [Електронний ресурс] – режим доступу: *http://quantumaviationtechnology.org/* (дата звернення: 20.11.2024)

15. Застосування технологій віртуальної реальності для тренування аеронавігаційних персоналів у виявленні конфліктів [Електронний ресурс] – режим доступу: *http://virtualrealityaviationtraining.org/* (дата звернення: 7.07.2024)

16. Інноваційні підходи до управління виробництвом [Електронний ресурс] – режим доступу: *https://innovationmanagement.org/articles/40920/* (дата звернення: 22.04.2023)

17. Інтеграція системи оптичної локалізації в процес виявлення конфліктів траекторій повітряних суден [Електронний ресурс] – режим доступу: *https://opticallocalizationintegration.org/* (дата звернення: 15.08.2024)

18. Методи акустичного моніторингу для вирішення проблеми конфліктів у повітрі [Електронний ресурс] – режим доступу: *https://acousticaviation.org/methods-conflict-detection/* (дата звернення: 12.02.2023)

19. Методи ідентифікації та класифікації конфліктів у повітрі [Електронний ресурс] – режим доступу: *https://conflictidentificationmethods.org/* (дата звернення: 10.05.2024)

20. Програмне забезпечення для аналізу траєкторій руху повітряних суден [Електронний ресурс] – режим доступу: *http://airtrafficsoftware.com/trajectory-analysis-tools/* (дата звернення: 20.03.2023)

21. Реалізація алгоритмів машинного навчання для виявлення конфліктів траєкторій у повітрі [Електронний ресурс] – режим доступу: *https://mlalgorithmsaviation.org/conflict-detection-ml/* (дата звернення: 22.10.2023)

22. Розвиток систем виявлення конфліктів траєкторій повітряних суден на основі інноваційних технологій [Електронний ресурс] – режим доступу: *https://innovativeaviationdetection.org/* (дата звернення: 5.12.2024)

23. Сенсорні технології в аеронавігаційних системах: виявлення конфліктів [Електронний ресурс] – режим доступу: *https://sensorsinaviation.org/conflict-detection-sensors/* (дата звернення: 1.04.2023)

24. Створення моделей прогнозування для покращення систем виявлення конфліктів у повітрі [Електронний ресурс] – режим доступу: *https://predictivemodelingaviation.org/conflict-detection-models/* (дата звернення: 8.08.2023)

25. Технології виявлення конфліктів у повітрі: аналіз сучасних підходів [Електронний ресурс] – режим доступу: *http://conflictdetectiontechnologies.org/analysis/* (дата звернення: 20.02.2024)

26. Точність інерційних систем у виявленні конфліктів траєкторій повітряних суден [Електронний ресурс] – режим доступу: *https://inertialsystemaccuracy.org/* (дата звернення: 25.06.2024)

27. Фактори, що впливають на розвиток систем виявлення конфліктів траєкторій повітряних суден [Електронний ресурс] – режим доступу: *https://conflict-detection-factors.org/* (дата звернення: 15.09.2024)

28. Чисельні методи в розв'язанні задач виявлення конфліктів траєкторій повітряних суден [Електронний ресурс] – режим доступу: *http://numericalmethodsaviation.org/* (дата звернення: 12.11.2023)

29. Аналіз використання програмних методів для виявлення конфліктів траекторій повітряних суден [Електронний ресурс] – режим доступу: https://example.com/analyzing-aircraft-conflict-detection (дата звернення: 3.10.2023)

30. Фізичні аспекти виявлення конфліктів у повітряному просторі: сучасні підходи [Електронний ресурс] – режим доступу: *https://example.org/physical-detection-approaches* (дата звернення: 3.10.2023)

31. Інноваційні програмні засоби для попередження конфліктів у повітряному русі [Електронний ресурс] – режим доступу: *https://example.net/innovative-software-conflict-prevention* (дата звернення: 3.10.2023)

32. Роль технологій інтернету речей у виявленні конфліктів траекторій літальних апаратів [Електронний ресурс] – режим доступу: *https://example.com/role-of-iot-in-conflict-detection* (дата звернення: 3.10.2023)

33. Фізичні методи виявлення конфліктів у повітряному просторі: використання радарів [Електронний ресурс] – режим доступу: *https://example.org/physical-conflict-detection-radar* (дата звернення: 3.10.2023)

Лістинг коду реалізації *OpenSky Network API*

```
import axios, { AxiosError } from "axios";
import {
IFlightState,
IOpenSkyNetworkApi,
IAirportState,
IAirportExtendedState,
TBounds,
IRouteState,
ITrajectoryState,
ITrajectoryPath,
IAirportPastFlight,
} from "../../types";
// https://openskynetwork.github.io/opensky–api/rest.html
interface IGetFlightsApiResponse {
time: string;
states: any[][];
interface IGetTrajectoryApiResponse extends ITrajectoryState {
path: any[];
const checkNested = (obj: object, field: string) => {
if (!field) return true;
// nested fields are separated by .
// e.g: test1.test2.test3
const fields = field.split(".");
for (let i = 0; i < fields.length; i++) {
if (!obj || !obj.hasOwnProperty(fields[i])) return false;
```

```
obj = obj[fields[i] as keyof typeof obj];
return true;
class OpenSkyNetworkApi implements IOpenSkyNetworkApi {
private static readonly baseURL = "https://opensky–network.org/api";
private static readonly username = process.env.OPENSKY_USERNAME;
private static readonly password = process.env.OPENSKY_PASSWORD;
private readonly axiosInstance = this.prepareAxiosInstance();
private prepareAxiosInstance() {
const headers =
OpenSkyNetworkApi.username && OpenSkyNetworkApi.password
Authorization: `Basic ${Buffer.from(
`${OpenSkyNetworkApi.username}:${OpenSkyNetworkApi.password}`,
"base64"
const instance = axios.create({
headers,
baseURL: OpenSkyNetworkApi.baseURL,
return instance;
private handleApiErrors(error: AxiosError) {
const { cause, code, request, response } = error;
if (cause) console.log(`CAUSE: ${cause}`);
if (code) console.log(`CODE ${code}`);
if (response) {
console.log(
`Error after response was received!\nRESPONSE: ${JSON.stringify({
status: response.status,
headers: response.headers,
data: response.data || [],
} else if (request) {
console.log(
`Response was not received!\nREQUEST: ${JSON.stringify({
```

```
status: request.status,
headers: request.headers,
} else {
console.log(
`Error before request was made!\nERROR: ${JSON.stringify(
error.toJSON()
public async getFlights(bounds: TBounds): Promise<IFlightState[]> {
const [[lamin, lomin], [lamax, lomax]] = bounds;
try {
const response = await this.axiosInstance.get<IGetFlightsApiResponse>(
`/states/all?lamin=${lamin}&lomin=${lomin}&lamax=${lamax}&lomax=${lomax}`
if (!checkNested(response, "data.states")) return [];
const data = response.data.states.map(
(fState) => fState as unknown as IFlightState
return data;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return [];
public async getFlight(
icao24: string,
time?: number
): Promise<IFlightState | null> {
try {
let path = `/states/all?icao24=${icao24}`;
if (time) path += `&time=${time}`;
const response = await this.axiosInstance.get<IGetFlightsApiResponse>(
path
if (
!checkNested(response, "data.states") ||
```

```
response.data.states.length === 0
return null;
const flight = response.data.states[0] as unknown as IFlightState;
return flight;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return null;
public async getAirports(bounds: TBounds): Promise<IAirportState[]> {
try {
const [[lamin, lomin], [lamax, lomax]] = bounds;
const response = await this.axiosInstance.get<IAirportState[]>(
`/airports/region?lamin=${lamin}&lomin=${lomin}&lamax=${lamax}&lomax=${lomax}
const airports = response.data;
return airports;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return [];
public async getAirport(icao: string): Promise<IAirportExtendedState | null> {
try {
const response = await this.axiosInstance.get<IAirportExtendedState>(
`/airports?icao=${icao}`
const airport = response.data;
return airport;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return null;
public async getRoute(callsign: string): Promise<IRouteState | null> {
```

```
try {
const response = await this.axiosInstance.get<IRouteState>(
`/routes?callsign=${callsign}`
const route = response.data;
return route;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return null;
public async getTrajectory(icao24: string): Promise<ITrajectoryState | null> {
try {
const response = await this.axiosInstance.get<IGetTrajectoryApiResponse>(
`/tracks/all?icao24=${icao24}&time=0`
const trajectory: ITrajectoryState = {
...response.data,
path: response.data.path
? response.data.path.map((p) => p as unknown as ITrajectoryPath)
: [],
return trajectory;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return null;
public async getArrivals(
icao: string,
begin: number,
end: number
): Promise<IAirportPastFlight[]> {
try {
const path = `/flights/arrival?airport=${icao}&begin=${begin}&end=${end}`;
```

```
const response = await this.axiosInstance.get<IAirportPastFlight[]>(path);
const arrivals = response.data;
return arrivals;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return [];
public async getDepartures(
icao: string,
begin: number,
end: number
): Promise<IAirportPastFlight[]> {
try {
const path = `/flights/departure?airport=${icao}&begin=${begin}&end=${end}`;
const response = await this.axiosInstance.get<IAirportPastFlight[]>(path);
const departures = response.data;
return departures;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return [];
const openSkyApi = new OpenSkyNetworkApi();
export { openSkyApi };
if (time) path += `&time=${time}`;
const response = await this.axiosInstance.get<IGetFlightsApiResponse>(
path
if (
!checkNested(response, "data.states") ||
response.data.states.length === 0
return null;
```

```
const flight = response.data.states[0] as unknown as IFlightState;
return flight;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return null;
public async getAirports(bounds: TBounds): Promise<IAirportState[]> {
try {
const [[lamin, lomin], [lamax, lomax]] = bounds;
const response = await this.axiosInstance.get<IAirportState[]>(
`/airports/region?lamin=${lamin}&lomin=${lomin}&lamax=${lamax}&lomax=${lomax}
const airports = response.data;
return airports;
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return [];
public async getAirport(icao: string): Promise<IAirportExtendedState | null> {
try {
} catch (error) {
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return [];
public async getFlight(
icao24: string,
time?: number
): Promise<IFlightState | null> {
try {
let path = `/states/all?icao24=${icao24}`;
if (time) path += `&time=${time}`;
```

```
const response = await this.axiosInstance.get<IGetFlightsApiResponse>(
path
if (
!checkNested(response, "data.states") ||
if (error instanceof AxiosError) this.handleApiErrors(error);
else console.log(`[getFlights] Unknown error: ${error}!`);
return [];
public async getAirport(icao: string): Promise<IAirportExtendedState | null> {
try {
const response = await this.axiosInstance.get<IAirportExtendedState>(
`/airports?icao=${icao}`
const airport = response.data;
return airport;
} catch (error) {}
```