НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних систем, мереж та кібербезпеки

_____ Касаткін Д.Ю., к.пед.н., доц.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: «Розробка системи симуляції для БПЛА»

Спеціальність F7 «Комп’ютерна інженерія»

Гарант освітньої програми

к.фіз.-мат.н., доц. _____ / Нікітенко Є.В. /

(підпис) (ПІБ)

Керівник дипломного проекту: _____ / Назаренко В.А. /

(підпис) (ПІБ)

Виконав: _____ / Сікорський О.І. /

(підпис) (ПІБ)

КИЇВ-2025НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.пед.н., доц. /
(підпис) (ПІБ, вчене звання і ступінь)
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТУ

Сікорський Олександр Ігорович
(прізвище, ім'я, по батькові)
Спеціальність (напрям підготовки): комп'ютерна інженерія
Тема кваліфікаційної бакалаврської роботи: «Розробка системи симуляції для БПЛА»
затверджена наказом ректора НУБіП України від "16" 12 2024р. № 2250 «С»
Термін подання завершеної роботи на кафедру
Вихідні дані до кваліфікаційної бакалаврської роботи література по використанню біометрії
Перелік питань, що підлягають розробці:
1. Теоретичні основи розробки системи симуляції для БПЛА
2. Проєктування архітектури системи симуляції
3. Реакція та тестування
4. Аналіз результатів та перспективи подальшої роботи
Перелік графічного матеріалу (за потреби)

Дата видачі завдання "16" 12 2024 р.

Керівник кваліфікаційної роботи _____ Назаренко В.А., к.т.н.
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання_____ Сікорський О.І.
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ВСТУП

У XXI столітті технології безпілотних літальних апаратів (БПЛА) набули значного поширення та стали невід'ємною частиною сучасних інженерних і наукових рішень. БПЛА використовуються в найрізноманітніших сферах: у моніторингу навколишнього середовища, агропромисловості, геодезії, логістиці, пошуково-рятувальних операціях, а також у військовій та оборонній галузях. Завдяки своїй мобільності, економічності та здатності працювати в складнодоступних умовах, ці апарати відкривають нові можливості для збору та обробки даних, автоматизації процесів та реалізації автономних систем.

Зі зростанням складності БПЛА, зокрема з впровадженням алгоритмів автономного керування, штучного інтелекту та сенсорної інтеграції, зростає і потреба в надійних та гнучких інструментах для проєктування, тестування і вдосконалення таких систем. Реальні випробування є дорогими, ризикованими і обмеженими з погляду варіативності сценаріїв. Саме тому системи симуляції набувають особливого значення. Вони дозволяють відтворювати різноманітні умови польоту, моделювати поведінку апарата в реальному часі, тестувати алгоритми керування без ризику пошкодження обладнання або завдання шкоди оточенню.

Система симуляції для БПЛА — це програмний або програмно-апаратний комплекс, що імітує фізичну модель польоту, поведінку сенсорів, взаємодію з навколишнім середовищем, а також дає змогу перевіряти програмне забезпечення БПЛА у безпечному віртуальному середовищі. Такі системи не лише зменшують витрати на розробку, але й підвищують надійність кінцевого продукту, дозволяючи виявити помилки на ранніх етапах.

Актуальність теми дослідження зумовлена потребою в універсальних, масштабованих та реалістичних симуляторах, які б могли ефективно підтримувати процеси розробки, тестування та валідації систем управління

БПЛА. Особливо це стосується академічного та дослідницького середовища, де вартість реального обладнання є критичним фактором.

Метою цієї дипломної роботи є розробка системи симуляції для БПЛА, яка забезпечує можливість моделювання динаміки польоту, сенсорного середовища та сценаріїв взаємодії з зовнішніми об'єктами. Основна увага приділяється розробці гнучкої архітектури симулятора, яка дозволить легко інтегрувати нові модулі, алгоритми керування та інтерфейси взаємодії.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

Аналіз існуючих систем симуляції для БПЛА та визначення їхніх переваг і недоліків.

Вибір програмних засобів та інструментів, що найкраще відповідають поставленим вимогам.

Реалізація моделі динаміки руху БПЛА.

Інтеграція модулів сенсорного моделювання.

Розробка інтерфейсу користувача для керування симуляцією.

Проведення тестування системи на основі прикладних сценаріїв.

Об’єкт дослідження - процеси проєктування, тестування та валідації програмного забезпечення та алгоритмів керування для безпілотних літальних апаратів.

Предмет дослідження - методи та засоби симуляції динаміки польоту БПЛА, моделювання сенсорного середовища і розробка програмної архітектури симулятора.

Практична значущість роботи. Результати роботи можуть бути використані для створення ефективних інструментів навчання операторів БПЛА, розробки та тестування нових алгоритмів керування без потреби у використанні дорогого фізичного обладнання. Запропонована система симуляції також може застосовуватись у дослідницьких цілях, в освітніх установах та у стартапах, що працюють у сфері автономних літальних апаратів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ СИСТЕМИ СИМУЛЯЦІЇ ДЛЯ БПЛА

1.1. Класифікація та особливості БПЛА

БПЛА використовують понад 45 років, але широке застосування почалося недавно. Їх застосовують у військових, цивільних, наукових та інших сферах. За призначенням розрізняють військові, пошукові, спеціальні служби, МНС, лісове господарство, нафту і газ, енергетику, зв’язок, кадастр, дороги, сільське господарство, охорону об’єктів і навколишнього середовища. За типом конструкції бувають аеростатичні, реактивні, літакові, вертолітні, мультикоптерні та гібридні[1].

БПЛА є безпечними джерелами достовірної інформації в районах та в умовах, що становлять загрозу для життя людини. При цьому безпілотники можуть здійснювати моніторинг та виконувати різні завдання у будь-який час доби, у сприятливих та обмежених метеоумовах. Робота БПЛА проводиться за спеціальним польотним завданням або за маршрутом оператора польоту. БПЛА в порівнянні з пілотованими повітряними суднами має явну перевагу у зв'язку з меншою вартістю експлуатації, зменшенням регламентних робіт, відсутністю витрат на навчання та підготовку пілотів [2].

Безпілотні літальні апарати поділяються за методом управління на три види:

1. Дистанційно керовані — управляються оператором через радіоканал, 5G, супутниковий або оптичний зв’язок.
2. Автономні — виконують польоти без оператора, оснащені мікропроцесорами, навігацією (ГНСС з RTK) та інерційними модулями (IMU).

3. Гібридні — поєднують автономний і ручний режими, оператор може втрутитись або апарат літає сам при втраті зв’язку.

За конструкцією виділяють: літаки з нерухомим крилом, одноротрні вертольоти, мультикоптери (з кількома роторами) та гібридні апарати. Найпопулярніші — квадрокоптери з чотирма двигунами через простоту керування.[2].

Квадрокоптер R1000

Крім легкості управління до безумовних переваг мультироторних БПЛА відноситься їх відносно низька вартість, можливість вертикального зльоту з будь-якого непідготовленого майданчика та здатність зависання у разі потреби над наземним об'єктом, а також висока точність позиціонування. Головними недоліками є низька швидкість, високі енерговитрати, обмежений радіус дії, недостатня вантажопідйомність і короткий час польоту (як правило в межах години) [3].

Найстаріший і найшвидший тип БПЛА — з нерухомим крилом, що мають велику дальність і тривалість польоту. Більшість середніх і важких БПЛА — літакового типу, наприклад, "Оріон" (ДК "Кронштадт"). Перспективною конструкцією вважають «монокрило» (БПЛА "С-70"). Для зльоту та посадки таких апаратів потрібна спеціальна інфраструктура, а керування вимагає високої кваліфікації оператора.

Інший клас — одноротрні безпілотники, схожі на вертольоти з різними схемами гвинтів. Вони мають хорошу стабілізацію та швидкодію, можуть оснащуватись різними двигунами (ДВР, газотурбінними, електродвигунами чи гібридними). Недоліком є складна конструкція, велика енергоспоживаність і великі габарити гвинта.[4].

БПЛА "ВРТ300" (НВП "Стріла")

БПЛА гібридної конструкції

Найбільш відомим видом гібридного безпілотника є т.зв. «Vtol», який поєднує в собі конструкційні принципи літака та квадрокоптера:

БПЛА "Zelator-28" (Airbus)

Такий БПЛА має переваги вертикального зльоту за рахунок наявності чотирьох додаткових двигунів, розташованих під площиною крила (по два з кожного боку) і одночасно він може розвивати серйозну швидкість у польоті завдяки аеродинамічній схемі літального апарата з нерухомим крилом.

Крім цього, існує і безліч інших гібридних систем БПЛА. Ось лише деякі з них:

Гібрид безпілотного вертольота та літака (tailsitter) [5];

Колеоптер (кільцеплан) – БПЛА з кільцевим замкнутим крилом, яке дозволяє збільшити кути атаки та маневреність безпілотника, а також покращити співвідношення злітної маси та корисного навантаження;

БПЛА на основі ефекту Коанда (зонтоліт), крім безпрецедентної маневреності такий безпілотник не має рухомих частин, що дозволяє йому ефективно обходити перешкоди, що розташовані поблизу;

БПЛА з гнучким крилом (параплан), як відомо, ефект планування при вільному польоті може забезпечити значну економію енергоресурсів двигуна;

Аеростатичний БПЛА (безпілотний дирижабль), який відрізняється безшумністю та високою вантажопідйомністю;

БПЛА з махаючим крилом (орнітоптер), що імітує політ птиці, завдяки чому досягається висока енергоефективність.

І хоча поки що здебільшого розробки гібридних БПЛА існують лише у вигляді досвідчених зразків, у майбутньому вони цілком можуть знайти собі гідне застосування у різних сферах людської діяльності.

1.2. Призначення та вимоги до симуляційних систем

Симуляційні системи важливі для розробки та тестування БПЛА, дозволяючи моделювати польоти в реальних умовах без ризиків і великих витрат. Вони призначені для:

- віртуального тестування ПЗ і алгоритмів керування;
- моделювання різних умов польоту;
- підготовки операторів без реального обладнання;
- дослідження аварійних ситуацій;
- швидкого прототипування нових рішень.

Основні вимоги до симуляційних систем [6]

Для досягнення вищезгаданих цілей симуляційна система повинна відповідати низці вимог, які поділяються на функціональні та нефункціональні.

Функціональні вимоги:

1. Динаміка польоту. Симулятор має точно відтворювати фізичні процеси: аеродинаміку, інерцію, вплив сили тяжіння, турбулентності тощо.
2. Сенсорна модель. Важливо мати змогу моделювати роботу GPS, гіроскопа, акселерометра, барометра, камери, радара, LiDAR та інших сенсорів.
3. Інтеграція з алгоритмами. Симулятор повинен підтримувати взаємодію з зовнішніми алгоритмами (ПІД-регулятори, SLAM, навігація, штучний інтелект).
4. Моделювання середовища. Система повинна дозволяти створювати 3D-сцени, ландшафти, об’єкти та перешкоди.
5. Сценарії та автоматизація. Можливість задавати та запускати типові або довільні сценарії польоту, місії та симуляційні експерименти.

Нефункціональні вимоги:

1. Масштабованість. Підтримка симуляції кількох БПЛА одночасно або варіантів одного апарата.

2. Реалістичність. Висока відповідність між симульованими та реальними даними.

3. Продуктивність. Підтримка роботи в реальному часі з мінімальними затримками.

4. Модульність. Можливість розширення функціоналу без значної модифікації основного ядра.

5. Інтерфейсність. Зручний графічний або командний інтерфейс користувача.

6. Сумісність. Підтримка стандартних протоколів і систем, таких як ROS, PX4, MAVLink, Gazebo, Unreal Engine тощо.

Таким чином, ефективна симуляційна система є важливим інструментом у всьому циклі розробки БПЛА — від ідеї до практичного впровадження. Вона забезпечує безпечне, контрольоване й гнучке середовище для експериментів і навчання.

1.3. Огляд сучасних платформ симуляції БПЛА (Gazebo, AirSim, PX4 SITL тощо)

У сфері розробки безпілотних літальних апаратів (БПЛА) існує широкий вибір симуляційних платформ, кожна з яких має свої особливості, переваги та сферу застосування. Нижче розглянуто найпопулярніші сучасні системи, що активно використовуються в дослідницькому, навчальному та промисловому середовищах.

Gazebo — це потужний симулятор з відкритим вихідним кодом, який широко використовується у сфері робототехніки. Gazebo підтримує фізично коректне моделювання, візуалізацію у 3D та інтеграцію з популярними фреймворками, зокрема ROS (Robot Operating System) [7].

AirSim — симулятор, розроблений компанією Microsoft, побудований на базі ігрового рушія Unreal Engine. Призначений для досліджень в галузі автономного водіння та керування БПЛА [8].

PX4 SITL — це інструмент для тестування програмного забезпечення польотного контролера PX4 у віртуальному середовищі без використання фізичного обладнання [9].

1.4. Аналіз існуючих підходів до моделювання польоту і навігації

Моделювання польоту та навігації є центральними завданнями при створенні симуляційної системи для безпілотних літальних апаратів (БПЛА). Ці підходи охоплюють як фізичне моделювання динаміки руху апарата, так і алгоритмічне забезпечення його здатності орієнтуватись у просторі та виконувати автономні місії.

Підходи до навігації

Сучасні системи навігації БПЛА включають комбінацію апаратного та програмного забезпечення. Основні підходи до навігації [10]:

- Інерціальна навігація (IMU). Використання гіроскопа та акселерометра для визначення орієнтації й прискорення. Потребує корекції через накопичення похибок.
- Супутникова навігація (GPS/GNSS). Забезпечує абсолютне позиціонування, але має затримки, обмежену точність та залежність від зовнішніх умов (наприклад, втрати сигналу).
- Візуальна навігація (Visual Odometry, SLAM). Використання камер та алгоритмів комп’ютерного зору для оцінки руху та побудови карти середовища. Підходить для GPS-заборонених середовищ.

- Інтегровані підходи. Комбінація різних джерел (IMU + GPS + камера + LiDAR) з використанням фільтрів Калмана або байєсівських методів для отримання надійної оцінки стану.
- Штучний інтелект і машинне навчання. Застосовується для адаптивної навігації, розпізнавання об'єктів, планування маршруту у змінному середовищі.

Алгоритми керування та стабілізації

Для забезпечення стійкого польоту БПЛА використовуються такі типи алгоритмів:

- ПІД-регулятори (PID). Найпоширеніший клас, що забезпечує просту і надійну стабілізацію в більшості застосунків.
- Модельно-орієнтоване керування (MPC). Забезпечує точне керування з урахуванням обмежень, проте потребує значних обчислювальних ресурсів.
- Методи нечіткої логіки, нейронні мережі. Підходять для адаптивного керування в умовах неповної або неточної інформації.

Таким чином, моделювання польоту і навігації — це складний, багаторівневий процес, який вимагає поєднання точного математичного опису фізичних процесів та ефективних алгоритмів обробки даних. Вибір підходів залежить від цілей симуляції, доступних обчислювальних ресурсів та типу БПЛА.

Приклади реалізації в симуляційних системах

Gazebo + PX4 SITL

У зв’язці Gazebo + PX4 SITL реалізовано [11]:

- Модель динаміки польоту на основі рівнянь Ньютона-Ейлера з урахуванням тяги двигунів, інерції, сили тяжіння та аеродинамічного опору.

- Інтеграцію сенсорів: GPS, IMU, магнітометр, а також можливість підключення віртуальної камери або LiDAR.
- Підтримку стандартних алгоритмів керування PX4 (позиційне, швидкісне, кутове).
- Візуальну навігацію та SLAM через інтеграцію з ROS, що дозволяє підключати пакети ORB-SLAM, Cartographer тощо.

AirSim

У AirSim (на базі Unreal Engine) реалізовано:

- Високоточне фізичне моделювання з використанням параметрів аеродинаміки та конфігурації БПЛА.
- Симуляція камер та глибини зображення, що дає змогу тестувати алгоритми комп'ютерного зору та візуальної навігації (наприклад, visual odometry або CNN для розпізнавання).
- Сенсорний шум та збої для тестування стійкості алгоритмів.
- Навчання штучного інтелекту, з можливістю інтеграції з Python API для reinforcement learning або imitation learning.

JMAVSim

JMAVSim — легший варіант симуляції, використовуваний із PX4:

- Містить базову модель мультикоптера без складної аеродинаміки.
- Використовується переважно для тестування ПІД-регуляторів і прошивки в Software-In-The-Loop (SITL).
- Має просту візуалізацію, але достатню для перевірки базових сценаріїв польоту.

Ці приклади демонструють, що сучасні симулятори дозволяють реалізувати широкий спектр підходів до моделювання польоту та навігації, що охоплюють як фізику руху, так і обробку сенсорних даних у режимі реального

часу. Вибір конкретного симулятора залежить від технічних вимог до точності, графіки, сумісності та цільового застосування.

Висновки до розділу 1

У даному розділі було здійснено огляд та аналіз сучасного стану досліджень у сфері симуляції безпілотних літальних апаратів (БПЛА). Встановлено, що симуляційні системи є ключовим інструментом для розробки, тестування, навчання та дослідження автономних систем керування БПЛА. Їх застосування дозволяє значно зменшити витрати, знизити ризики та підвищити якість програмного забезпечення.

Проаналізовано призначення симуляційних систем, визначено основні функціональні та нефункціональні вимоги до них, серед яких особливе місце займають точність динамічного моделювання, підтримка сенсорів, гнучкість архітектури та сумісність із сучасними фреймворками (ROS, PX4, MAVLink).

Проведено огляд найбільш поширених платформ моделювання, таких як Gazebo, AirSim, PX4 SITL, JMAVSim, RotorS та інших. З'ясовано, що кожна з них має свої переваги і сфери застосування, зокрема: Gazebo — для досліджень у сфері робототехніки, AirSim — для високоякісної візуалізації та роботи зі штучним інтелектом, PX4 SITL — для тестування прошивок у реальному часі.

Розглянуто існуючі підходи до моделювання польоту і навігації, включаючи класичні кінематичні та аеродинамічні моделі, алгоритми ПІД-регулювання, фільтри Калмана, SLAM та візуальну одометрію. Визначено, що сучасні симулятори дозволяють інтегрувати ці підходи у віртуальному середовищі з високим рівнем достовірності.

Отримані результати стануть основою для формування архітектури власної симуляційної системи, яка буде реалізована у наступних розділах роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ СИМУЛЯЦІЇ

2.1. Визначення функціональних та нефункціональних вимог

Симулятор безпілотного літального апарата (БПЛА) повинен забезпечувати повноцінну імітацію умов реального середовища, в яких функціонує дрон. Це включає не лише відтворення фізики польоту, а й моделювання роботи сенсорів та системи керування. Такі можливості є критично важливими для розробки та тестування програмного забезпечення, зокрема алгоритмів автономного керування, стабілізації, навігації та уникнення перешкод. Віртуальне тестування дає змогу уникнути ризиків, пов’язаних із реальними випробуваннями, скоротити витрати та пришвидшити цикл розробки.

Імітація польоту БПЛА — моделювання руху апарата з урахуванням фізичних законів, таких як гравітація, опір повітря, інерція та зовнішні збурення (наприклад, вітер) [12].

Моделювання сенсорів — емуляція реалістичних даних від GPS, IMU, камери, лідару, висотоміра тощо, включно з шумами, затримками та похибками, які притаманні реальним пристроям [13].

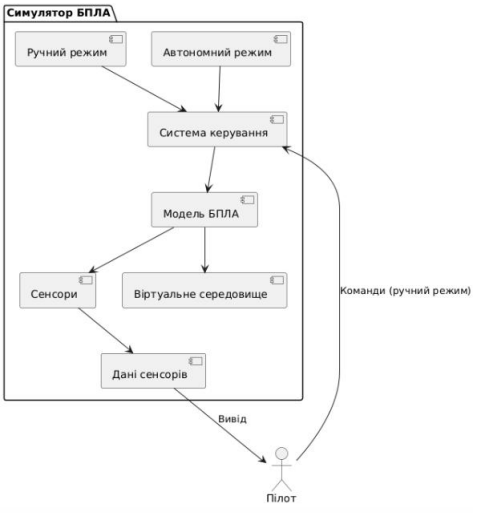
Система керування польотом — реалізація контролерів стабілізації, підтримки висоти, орієнтації, а також можливість підключення зовнішніх автопілотів (наприклад, PX4, ArduPilot) для тестування складніших сценаріїв [14].

Ці компоненти утворюють єдину симуляційну екосистему, де кожен блок взаємодіє з іншими через визначені інтерфейси або шини обміну даними, такі як ROS.

Реалізація сценаріїв польоту — ключова функція симулятора БПЛА, що дозволяє відтворювати різні режими та оцінювати поведінку апарата в умовах, близьких до реальних. Основні режими — ручний і автономний.

У ручному режимі пілот самостійно керує орієнтацією, швидкістю й висотою БПЛА. Симулятор має надавати точні дані про стан апарата (висота, координати, швидкість тощо) та забезпечувати зміну траєкторії в реальному часі. Важливою є також можливість моделювання зовнішніх впливів, як-от вітер чи погодні умови.

У автономному режимі БПЛА самостійно виконує політ за заздалегідь визначеними алгоритмами або маршрутами, без активного втручання оператора. Тут симулятор повинен підтримувати не тільки базові функції, такі як зліт і посадка, але й складні сценарії польоту, зокрема динамічну зміну маршруту в реальному часі, уникнення перешкод або корекцію траєкторії через зовнішні фактори, такі як зміна погодних умов. Алгоритми автономного польоту повинні включати, наприклад, використання GPS для навігації або сенсорів для визначення місцеположення та уникнення перешкод [15].



Симулятор БПЛА має підтримувати перехід між ручним і автономним режимами, що необхідно для моделювання втручання оператора або збоїв у роботі автономних алгоритмів. Також важливо забезпечити гнучкість сценаріїв: зміну погодних умов, втрату зв'язку, обмежену видимість чи аварійні ситуації. Це підвищує реалістичність та дозволяє краще готувати систему до реального застосування.

Такі можливості роблять симулятор ефективним інструментом для тестування обох типів управління — автономного й ручного — знижуючи ризики при реальних польотах.

Окрім функціоналу, симулятор має відповідати нефункціональним вимогам — бути стабільним, масштабованим і кросплатформеним, що гарантує його ефективну роботу на різних пристроях.[16].

Продуктивність — ключова вимога до симулятора БПЛА, оскільки обробка великого обсягу даних у реальному часі потребує високої швидкості та точності. Система має ефективно обробляти складні математичні моделі, що описують фізику польоту, сенсори й навігацію, без затримок чи втрати точності. Це забезпечує реалістичне відображення подій у режимі реального часу.

Масштабованість також критично важлива. Симулятор повинен підтримувати моделювання багатьох БПЛА одночасно, великих територій та

складних сценаріїв. Архітектура має бути гнучкою для легкого додавання нових функцій, сенсорів або інтеграції з іншими системами без суттєвих змін. Це дозволяє ефективно тестувати взаємодію між кількома апаратами або з іншими роботизованими платформами. [17].

Інтерфейс користувача (UI) є важливим елементом системи симуляції для БПЛА, оскільки забезпечує зручність взаємодії оператора з системою. Для забезпечення ефективного використання симулятора необхідно врахувати низку вимог до інтерфейсу, а також системи виведення даних, щоб забезпечити чітке та інтуїтивно зрозуміле відображення інформації [18].

Загалом, інтерфейс користувача та система виведення даних повинні забезпечити ефективну взаємодію між оператором і системою симуляції, дозволяючи швидко та точно контролювати та налаштовувати симуляційні параметри, а також забезпечити чітке відображення важливих даних у реальному часі.

2.2. Вибір середовища моделювання

Вибір середовища симуляції БПЛА є ключовим етапом розробки, оскільки впливає на точність моделювання умов польоту та ефективність тестування алгоритмів. Основний критерій — реалістичність, тобто точне відтворення фізики польоту (аеродинаміка, вітер, зовнішні фактори) та достовірна імітація сенсорів (камера, лідар, ультразвук). Обов’язковою є підтримка 3D-візуалізації для наочного спостереження за поведінкою БПЛА та оперативного внесення змін.[19].

Масштабованість та гнучкість також були важливими чинниками. Gazebo забезпечує можливість створення масштабованих моделей для симуляції не тільки одного БПЛА, а й цілих груп безпілотників, що є важливою вимогою для подальшого розвитку проєкту. Це дозволяє створювати складні сценарії, де декілька БПЛА взаємодіють між собою або з навколишнім середовищем, що

відкриває можливості для реалізації колективних місій або тестування алгоритмів для багатобортових операцій[20].

Таким чином, вибір Gazebo як середовища для симуляції в даному проєкті аргументується його високою реалістичністю, відкритістю, можливістю інтеграції з апаратним забезпеченням, а також гнучкістю та підтримкою масштабованих сценаріїв. Ці фактори дозволяють створити потужну основу для тестування та розвитку алгоритмів управління БПЛА в умовах, максимально наближених до реальних.

2.3. Архітектура програмного забезпечення симулятора

Архітектура симуляції БПЛА має бути модульною, гнучкою і масштабованою для ефективного тестування і легкої адаптації до змін. Вона складається з окремих компонентів (фізика польоту, сенсори, управління тощо), які працюють незалежно через чіткі інтерфейси. Це спрощує оновлення, тестування та підвищує надійність системи. [25].

Компоненти системи — модулі фізики, сенсорів, управління і виведення даних — взаємодіють через чіткі інтерфейси. Модуль фізики передає стан БПЛА та умови навколишнього середовища модулю сенсорів і управління, які у реальному часі обробляють ці дані для керування траєкторією. Архітектура підтримує гнучкість і масштабованість: легко додавати сенсори або змінювати алгоритми без впливу на інші модулі. Модуль фізики моделює поведінку БПЛА з урахуванням аеродинаміки, вітру і гравітації для максимальної реалістичності польоту. [26].

Модуль сенсорів симулює камери, ультразвук, лідари, GPS, гіроскопи та акселерометри для навігації та виявлення перешкод. Модуль управління отримує дані від сенсорів і фізики, щоб коригувати траєкторію і швидкість польоту.

Модуль виведення забезпечує 3D-візуалізацію та звіти про стан БПЛА й історію польотів[21].

Модулі можуть об’єднуватися в підсистеми для автономного або ручного керування, що дає гнучкість у створенні різних сценаріїв. Система має модульну, масштабовану архітектуру для реалістичних симуляцій у реальному часі.

Основні компоненти: ядро симуляції (моделює фізику польоту, аеродинаміку, гравітацію, часову синхронізацію), модуль візуалізації (графічне відображення БПЛА, середовища і руху) та модуль керування і сенсорів. Візуалізація служить не лише для зручності, а й для аналізу та перевірки правильності роботи симулятора. [27].

Модуль керування польотом приймає рішення на основі даних сенсорів і стану БПЛА, реалізуючи ПІД-регулятори та складні алгоритми автономного польоту, стабілізації, уникнення перешкод і координації з іншими БПЛА. Він тісно взаємодіє з ядром симуляції, передаючи команди керування віртуальній моделі[24].

Модуль сенсорів імітує GPS, акселерометри, гіроскопи, камери, лідари та інші пристрої, генеруючи дані з урахуванням шумів і затримок. Ці дані надходять до модуля керування або зовнішніх систем для тестування поведінки БПЛА у реалістичних умовах. [28].

Таким чином, кожен з компонентів — ядро симуляції, візуалізація, керування польотом і сенсори — виконує свою унікальну функцію, але їхня інтеграція утворює цілісну систему, що дозволяє здійснювати повноцінну симуляцію польоту БПЛА. У майбутньому структура може бути розширена додатковими модулями, такими як взаємодія з хмарними обчисленнями, симуляція комунікацій або кооперація між кількома апаратами в складі рою [29].

ROS також підтримує синхронні services для запитів і actions для асинхронних дій із зворотним зв’язком (наприклад, посадка БПЛА). Для високої

продуктивності або кросплатформеності використовують REST API або WebSocket, що дає змогу запускати модулі в Docker чи хмарі. [30].

Ще один важливий засіб — транспорт повідомлень, що забезпечується middleware-рівнем, як-от ROS 2 DDS (Data Distribution Service). DDS дозволяє організувати більш масштабовану та безпечну взаємодію між вузлами в розподіленому середовищі, що особливо корисно для симуляції роїв БПЛА або для взаємодії з віддаленими клієнтами [31].

Таким чином, зв’язки між модулями можуть будуватися на основі ROS topics/services, динамічного API, або засобів міжпроцесної комунікації залежно від обраної архітектури. Ключовим принципом залишається забезпечення узгодженого обміну інформацією з мінімальними затримками, підтримкою паралельної роботи та масштабованості.

У розробці симуляційної системи для БПЛА важливу роль відіграє грамотна організація взаємодії між її модулями. Для цього доцільно використовувати перевірені шаблони проєктування, що дозволяють зменшити зв’язність компонентів, полегшити масштабування та забезпечити гнучкість у зміні чи розширенні функціоналу [32].

Іншим прикладом є шаблон Observer (спостерігач), який реалізує схожий принцип, однак частіше використовується на рівні взаємодії всередині окремих підсистем. Наприклад, у модулі візуалізації можна створити механізм, де об'єкти сцени «підписуються» на зміну стану БПЛА або сенсорних даних, і автоматично оновлюють своє відображення при отриманні нового стану [33].

Також у деяких випадках доцільно використовувати шаблон Singleton для компонентів, що повинні існувати в єдиному екземплярі, наприклад, для глобального логгера, менеджера параметрів або конфігураційного сервісу.

Застосування цих шаблонів не лише покращує читабельність та підтримуваність коду, а й створює фундамент для гнучкої, масштабованої та

стабільної архітектури, що легко адаптується до змін у вимогах або розширення функціоналу симулятора.

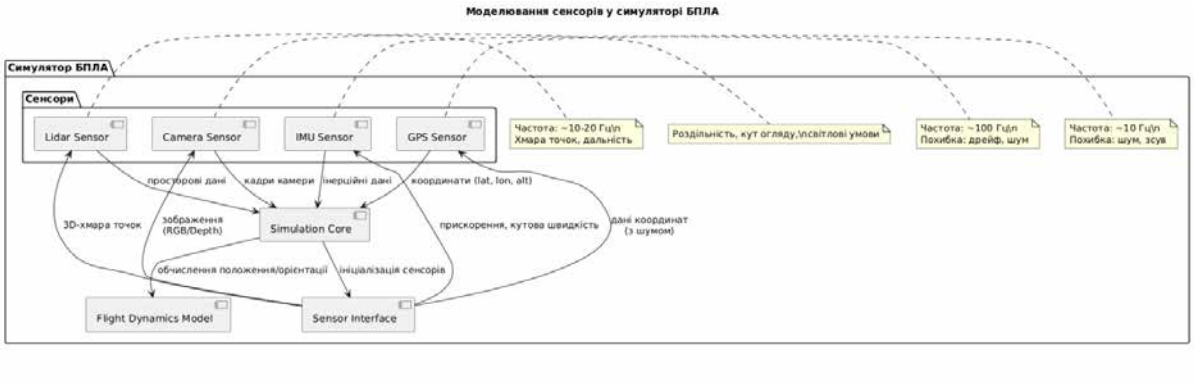
2.4. Інтеграція моделі БПЛА

Аеродинамічна модель є ключовим елементом симуляційної системи для БПЛА, оскільки саме вона забезпечує реалістичну поведінку апарата в повітряному середовищі. У залежності від типу БПЛА, що моделюється, використовуються різні підходи до побудови такої моделі — найпоширенішими є моделі з фіксованим крилом (fixed-wing) або з роторною системою (multirotor, наприклад квадрокоптер) [22].

Для БПЛА з фіксованим крилом основна увага приділяється розрахунку підйомної сили, що генерується крилами, а також сили опору, моментів нахилу й крену, які виникають при зміні кута атаки, швидкості та напрямку повітряного потоку. Такі моделі зазвичай базуються на рівняннях руху шести ступенів свободи (6DoF) і враховують гравітаційні, аеродинамічні й рушійні сили. Крім того, може використовуватись параметрична модель, побудована на основі експериментальних даних або CFD-розрахунків [34].

У симуляційних середовищах, як-от Gazebo або AirSim, моделі аеродинаміки зазвичай реалізуються як фізичні плагіни, які взаємодіють з рушієм фізики (наприклад, ODE, Bullet, PhysX). Вони дозволяють моделювати ефекти турбулентності, взаємодії з вітром, а також зміну аеродинамічних параметрів в залежності від висоти, швидкості та кута атаки [35].

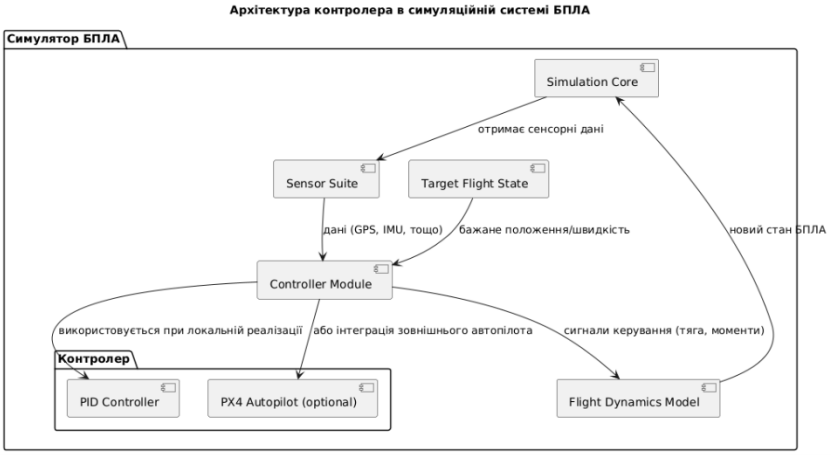
Реалістичність такої моделі є критично важливою для достовірного тестування навігаційних алгоритмів, контролю стабільності та симуляції сценаріїв польоту. Таким чином, обраний тип аеродинамічної моделі повинен відповідати фізичним характеристикам конкретного типу БПЛА та забезпечувати точне відтворення його поведінки в різних умовах.



Моделювання сенсорів є невід'ємною частиною симуляційної системи для БПЛА, оскільки саме сенсори забезпечують апарат необхідною інформацією про його положення, рух, оточення та навігаційні орієнтири. Для створення реалістичного середовища тестування необхідно не лише імітувати наявність сенсорів, а й враховувати їхню похибку, затримки та частоту оновлення даних [36].

Таким чином, поєднання реалістичного моделювання сенсорів і гнучких контролерів польоту створює потужний інструмент для розробки, тестування та вдосконалення систем управління БПЛА в безпечних та керованих умовах симуляції. [37].

Таким чином, вибір між власноручною реалізацією PID-контролера або імпортом готових рішень залежить від цілей проєкту: для академічного моделювання та аналізу систем керування може бути достатньо власного реалізованого алгоритму, а для реалістичної перевірки сценаріїв автономного польоту — доцільно інтегрувати відкриті платформи на кшталт PX4.

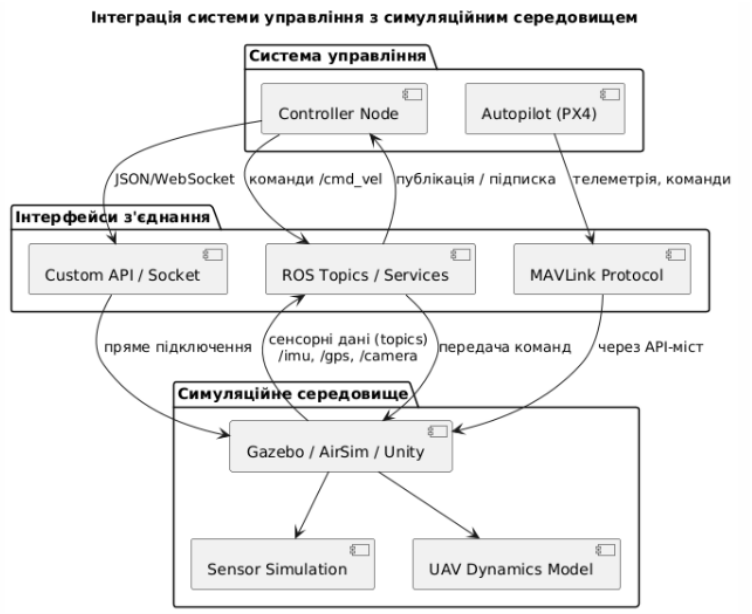


Підключення системи управління до симуляційного середовища є критичним етапом розробки, що забезпечує обмін даними між симульованим апаратом і модулем керування. Це з'єднання дозволяє надсилати команди руху, отримувати сенсорні дані, контролювати стани системи та реалізовувати замкнене коло управління. Існує декілька основних способів організації такого підключення, і вибір залежить від архітектури симулятора та використовуваних технологій[23].

Найпоширенішим інтерфейсом у сучасних роботизованих системах є ROS (Robot Operating System). Завдяки своїй модульності, ROS надає стандартизований спосіб обміну повідомленнями між компонентами системи через topics, services, і actions. Контролер може публікувати команди руху (наприклад, у топік /cmd_vel), у той час як сенсорні дані, отримані з симулятора, надходять через відповідні топіки (/imu/data, /gps/fix, /camera/image_raw тощо). Такий підхід дозволяє розділити логіку моделювання та керування, спростивши налагодження та тестування [38].

У випадках, коли використовується автопілот типу PX4, підключення до симуляційного середовища зазвичай реалізується через протокол MAVLink, що дозволяє взаємодію з симуляторами, такими як Gazebo чи AirSim, через API-шлюзи або MAVROS — мост ROS ↔ MAVLink. У такій конфігурації модуль управління може бути фізично окремим процесом або навіть запущеним на

апаратному блоці, і взаємодія з моделлю в симуляторі відбувається через повідомлення про телеметрію та управління.



Альтернативно, у середовищах, що не підтримують ROS напряму (наприклад, Unity), можлива реалізація кастомного API або сокет-з'єднання, що передає команди та дані між клієнтом керування та ядром симуляції. Такий підхід часто використовується при інтеграції нейромережевих агентів або систем навчання з підкріпленням.

РОЗДІЛ 3. РЕАКЦІЯ ТА ТЕСТУВАННЯ

3.1. Реалізація ключових компонентів симуляції

ROS-нод для симуляції IMU на Python [39].

```
#!/usr/bin/env python3

import rospy
import random
import math
from sensor_msgs.msg import Imu
from std_msgs.msg import Header
import tf.transformations

def generate_noisy_value(mean=0.0, stddev=0.01):
    return random.gauss(mean, stddev)

def imu_simulator():
    rospy.init_node('imu_simulator_node')
    imu_pub = rospy.Publisher('/imu/data_raw', Imu, queue_size=10)
    rate = rospy.Rate(50)  # 50 Hz

    while not rospy.is_shutdown():
        imu_msg = Imu()
        imu_msg.header = Header()
        imu_msg.header.stamp = rospy.Time.now()
        imu_msg.header.frame_id = "imu_link"

        # Імітація лінійних прискорень (акселерометр)
        imu_msg.linear_acceleration.x = generate_noisy_value(0.0, 0.02)
        imu_msg.linear_acceleration.y = generate_noisy_value(0.0, 0.02)
        imu_msg.linear_acceleration.z = generate_noisy_value(9.81, 0.02)  # гравітація по осі Z
```

```
        # Імітація кутових швидкостей (гіроскоп)
        imu_msg.angular_velocity.x = generate_noisy_value(0.0, 0.001)
        imu_msg.angular_velocity.y = generate_noisy_value(0.0, 0.001)
        imu_msg.angular_velocity.z = generate_noisy_value(0.0, 0.001)

        # Орієнтація в форматі кватерніонів (для прикладу фіксована)
        quaternion = tf.transformations.quaternion_from_euler(0, 0, 0)
        imu_msg.orientation.x = quaternion[0]
        imu_msg.orientation.y = quaternion[1]
        imu_msg.orientation.z = quaternion[2]
        imu_msg.orientation.w = quaternion[3]

        # Запис похибок орієнтації
        imu_msg.orientation_covariance = [0.01, 0, 0,
                                          0, 0.01, 0,
                                          0, 0, 0.01]
        imu_msg.angular_velocity_covariance = [0.001, 0, 0,
                                               0, 0.001, 0,
                                               0, 0, 0.001]
        imu_msg.linear_acceleration_covariance = [0.02, 0, 0,
                                                  0, 0.02, 0,
                                                  0, 0, 0.02]

        imu_pub.publish(imu_msg)
        rate.sleep()

if __name__ == '__main__':
    try:
        imu_simulator()
    except rospy.ROSInterruptException:
        pass
```

Вузол публікує повідомлення типу sensor_msgs/Imu у тему /imu/data_raw із частотою 50 Гц, що відповідає типовій швидкості оновлення даних справжнього IMU. Для генерації сигналів використовується функція generate_noisy_value, яка додає випадковий гаусовий шум до базових значень прискорення та кутової швидкості. Зокрема, вертикальна компонента прискорення імітує дію гравітації із значенням близько 9.81 м/с²[40].

Орієнтація моделюється у вигляді кватерніонів, тут для спрощення використовується фіксований кватерніон (без обертання). Для більш реалістичного моделювання можна інтегрувати модель руху та дрейф орієнтації[41].

Всі дані супроводжуються вказанням матриць коваріації, що описують рівень похибок та шумів сенсора. Такий підхід дозволяє симуляції максимально наблизитися до реальних умов роботи IMU, забезпечуючи адекватне тестування навігаційних та стабілізаційних алгоритмів[42].

ROS-архітектура забезпечує легке підключення цього модуля до інших компонентів симуляції, що дозволяє розробляти комплексну систему моделювання польоту БПЛА.

3.2. Розробка сценаріїв польоту

ROS-нод сценарію польоту на Python [44].

```
#!/usr/bin/env python3

import rospy
from geometry_msgs.msg import PoseStamped

def flight_scenario():
    rospy.init_node('flight_scenario_node')
    pub = rospy.Publisher('/setpoint_position', PoseStamped, queue_size=10)
```

```
rate = rospy.Rate(10)  # 10 Гц оновлення команд

# Сценарій польоту: підняття, утримання, повернення
waypoints = [
    {'x': 0.0, 'y': 0.0, 'z': 2.0},   # Від старту піднятись на 2 м
    {'x': 0.0, 'y': 0.0, 'z': 2.0},   # Утримання позиції 5 сек (потрібно дотримуватись часу у циклі)
    {'x': 0.0, 'y': 0.0, 'z': 0.0}    # Повернення на старт
]

hold_time = 5  # секунд для утримання позиції

current_waypoint_index = 0
hold_start_time = None

while not rospy.is_shutdown():
    pose_msg = PoseStamped()
    pose_msg.header.stamp = rospy.Time.now()
    pose_msg.header.frame_id = "map"

    waypoint = waypoints[current_waypoint_index]

    pose_msg.pose.position.x = waypoint['x']
    pose_msg.pose.position.y = waypoint['y']
    pose_msg.pose.position.z = waypoint['z']

    # Орієнтація фіксована (без повороту)
    pose_msg.pose.orientation.x = 0
    pose_msg.pose.orientation.y = 0
    pose_msg.pose.orientation.z = 0
    pose_msg.pose.orientation.w = 1

    pub.publish(pose_msg)
```

```
        # Логіка переходу між точками
        if current_waypoint_index == 1:
            # Утримуємо позицію 5 секунд
            if hold_start_time is None:
                hold_start_time = rospy.Time.now()
            elif (rospy.Time.now() - hold_start_time).to_sec() > hold_time:
                current_waypoint_index += 1
        else:
            current_waypoint_index += 1

        if current_waypoint_index >= len(waypoints):
            # Сценарій завершено — зупиняємось
            rospy.loginfo("Flight scenario complete")
            break

        rate.sleep()

if __name__ == '__main__':
    try:
        flight_scenario()
    except rospy.ROSInterruptException:
        pass
```

В рамках симуляційної системи для безпілотних літальних апаратів (БПЛА) важливо не лише моделювати сенсори та фізичну поведінку, але й реалізувати різноманітні сценарії польоту, які дозволяють тестувати алгоритми керування, навігації та планування траєкторії[45].

Розроблений ROS-вузол відповідає за генерацію команд керування позицією апарата у вигляді послідовності заданих точок (waypoints). У наведеному прикладі сценарій полягає у послідовному виконанні трьох етапів: підняття БПЛА на висоту 2 метри, утримання позиції протягом п’яти секунд, та повернення на початкову точку з висотою 0 метрів.

Для передачі команд використовується повідомлення geometry_msgs/PoseStamped, яке містить позицію і орієнтацію у глобальній системі координат. Частота оновлення команд встановлена на 10 Гц, що забезпечує достатню реактивність системи[46].

Логіка переходу між етапами реалізована через індекс поточного waypoint та врахування часу утримання позиції на другому етапі. Після виконання усіх заданих дій вузол виводить інформаційне повідомлення і завершує роботу.

Такий підхід дозволяє гнучко створювати складні послідовності польотів, які можуть використовуватися для перевірки ефективності контролерів, алгоритмів автономного польоту та інтеграції сенсорних даних.

3.3. Інтеграція з алгоритмами навігації

ROS-нод інтеграції алгоритму навігації (Python) [47].

```
#!/usr/bin/env python3

import rospy
from geometry_msgs.msg import PoseStamped, TwistStamped
from std_msgs.msg import Float64
import math
import time

class SimpleNavigationNode:
    def __init__(self):
        rospy.init_node('navigation_node')

        # Цільова позиція (для прикладу - фіксована)
        self.target_position = {'x': 5.0, 'y': 5.0, 'z': 2.0}

        # Підписка на фактичну позицію БПЛА (наприклад, з симулятора або сенсорів)
```

```
    rospy.Subscriber('/current_pose', PoseStamped, self.pose_callback)

    # Публікація команд керування (швидкості або позиції)
    self.cmd_pub = rospy.Publisher('/cmd_vel', TwistStamped, queue_size=10)

    # PID-параметри для простої позиційної регуляції
    self.kp = 0.5
    self.ki = 0.0
    self.kd = 0.1

    self.prev_error = {'x': 0.0, 'y': 0.0, 'z': 0.0}
    self.integral = {'x': 0.0, 'y': 0.0, 'z': 0.0}
    self.last_time = None

    self.current_pose = None
    self.rate = rospy.Rate(20)  # 20 Гц

def pose_callback(self, msg):
    self.current_pose = msg.pose

def compute_pid(self, target, current, axis):
    now = time.time()
    dt = now - self.last_time if self.last_time else 0.01
    self.last_time = now

    error = target - current
    self.integral[axis] += error * dt
    derivative = (error - self.prev_error[axis]) / dt if dt > 0 else 0.0

    output = self.kp * error + self.ki * self.integral[axis] + self.kd * derivative

    self.prev_error[axis] = error
    return output
```

```
    def run(self):
        while not rospy.is_shutdown():
            if self.current_pose is None:
                rospy.loginfo("Waiting for current pose...")
                self.rate.sleep()
                continue

            # Визначаємо помилки по координатах
            error_x = self.target_position['x'] - self.current_pose.position.x
            error_y = self.target_position['y'] - self.current_pose.position.y
            error_z = self.target_position['z'] - self.current_pose.position.z

            # PID-контроль для кожної координати
            cmd_vel = TwistStamped()
            cmd_vel.header.stamp = rospy.Time.now()
            cmd_vel.twist.linear.x = self.compute_pid(self.target_position['x'],
self.current_pose.position.x, 'x')
            cmd_vel.twist.linear.y = self.compute_pid(self.target_position['y'],
self.current_pose.position.y, 'y')
            cmd_vel.twist.linear.z = self.compute_pid(self.target_position['z'],
self.current_pose.position.z, 'z')

            # Для простоти, кутові швидкості залишаємо нульовими
            cmd_vel.twist.angular.x = 0.0
            cmd_vel.twist.angular.y = 0.0
            cmd_vel.twist.angular.z = 0.0

            self.cmd_pub.publish(cmd_vel)

            self.rate.sleep()

    if __name__ == '__main__':
```

```
try:
    node = SimpleNavigationNode()
    node.run()
except rospy.ROSInterruptException:
    pass
```

Для ефективного керування безпілотним літальним апаратом (БПЛА) у симуляційній системі критичною є інтеграція з алгоритмами навігації, що забезпечують точне слідування за заданою траєкторією або досягнення цільових позицій.

У наведеному прикладі реалізовано простий навігаційний модуль, що використовує PID-регулятор для керування лінійними швидкостями у трьох вимірах. Модуль отримує поточну позицію апарата у вигляді повідомлень PoseStamped та обчислює необхідні корекції команд управління для досягнення цільової позиції. Команди управління подаються у вигляді TwistStamped — у простому варіанті це лінійні швидкості по осях X, Y, Z.

Цей підхід є базовим і ілюструє ключові етапи інтеграції: отримання даних про поточне положення апарата, обробка помилок відхилення від цілі за допомогою ПІД-контролера, формування команд управління, та їх трансляція у систему керування БПЛА.

У реальних симуляторах і системах автопілотів алгоритми навігації можуть бути значно складнішими, включати обробку орієнтації, вітрових впливів, обхід перешкод, а також використовувати розширені методи фільтрації даних (наприклад, калманівські фільтри) та планування траєкторій. Однак навіть базова реалізація PID-навігації дає змогу протестувати інтеграцію контролера з моделлю руху апарата та сенсорними даними, що є важливим етапом розробки [48].

3.4. Тестування та оцінка ефективності симуляції

Оцінка ефективності роботи симуляційної системи для безпілотного літального апарата (БПЛА) є важливим етапом для визначення її точності, надійності та стабільності в різних умовах. Для цього використовуються кілька важливих метрик, які дозволяють виміряти ефективність виконання завдань, таких як слідування маршруту, стабільність польоту та час відгуку системи.

ROS є потужною платформою для створення робототехнічних систем, що включає велику кількість пакетів для вирішення різних завдань (від навігації та локалізації до комп'ютерного зору та симуляції), інструменти для високорівневої взаємодії із залізом робота, протоколи для обміну даними та багато іншого. Таким чином, ROS сильно полегшує розробку проектів для робототехніки, надаючи універсальний засіб розробки, незалежний від конкретної апаратної платформи робота.

Підготовка до встановлення. Налаштування репозиторіїв.

Почнемо з того, що існують чотири основні репозиторії:

- Main – безкоштовне програмне забезпечення з відкритим вихідним кодом, що підтримується Canonical.
- Universe - безкоштовне програмне забезпечення з відкритим вихідним кодом, яке підтримує спільнота.
- Restricted – пропрієтарні драйвери для пристроїв.
- Multiverse - програмне забезпечення, обмежене авторським правом або юридичними питаннями.

Отже, починаємо налаштування. Спочатку відкриваємо вкладку "Програмне забезпечення Ubuntu" і в налаштуваннях "Ubuntu Software" ставимо галочки так, як показано на рисунку 3.1.

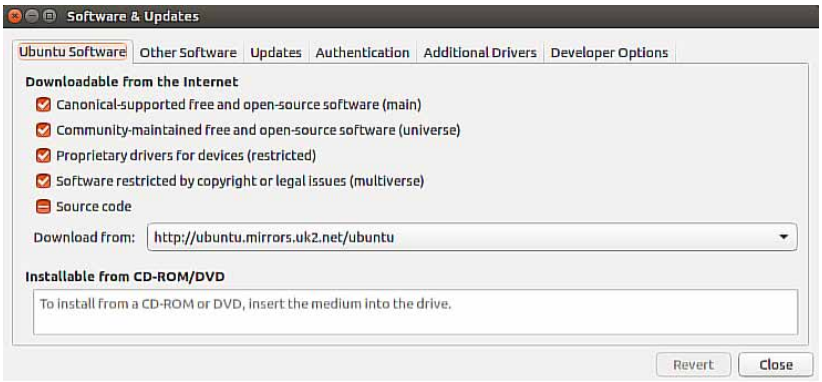


Рисунок 3.1. – Налаштування Ubuntu Software

Далі переходимо у вкладку "Other software" та ставимо галочки на пунктах Canonical Parthners та Canonical Parthners (source code)

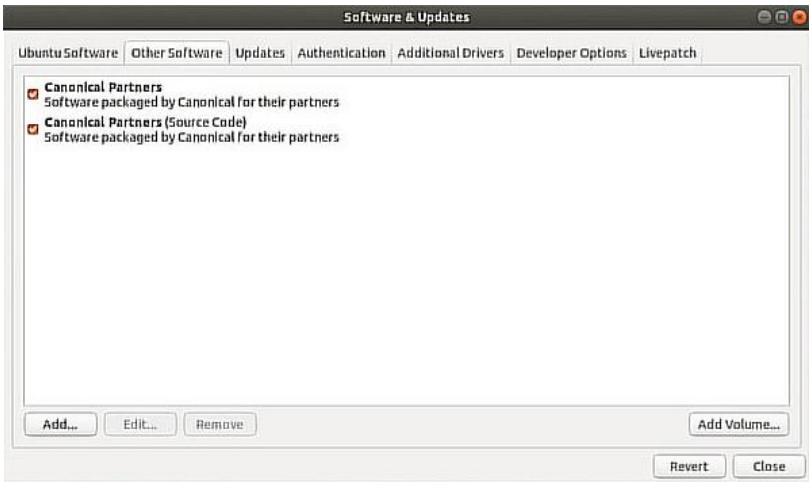


Рисунок 3.2. – Налаштування Other software

Після налаштування репозиторіїв можемо приступати до встановлення. Для початку настроюємо систему на прийом пакетів програмного забезпечення з packages.ros.org. Вставляємо в термінал наступну команду:

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $ (lsb_release -sc) main"> /etc/apt/sources.list.d/ros-latest.list'

Після цього налаштовуємо свої ключі. Після додавання репозиторію вводь одну з наступних команд:

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

В якості альтернативи ми можемо використовувати curl замість команди apt-key, що може бути корисним, якщо ви знаходитесь за проксі-сервером:

curl -sSL

'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0xC1CF6E31E6BADE8868B172B4F42ED6FBAB17C654' | sudo apt-key add

По-перше, оновлюємо пакети за допомогою команди

sudo apt-get update

Починаємо встановлення. Для цього треба вирішити, який ROS ви хочете собі встановити:

Desktop Full - та збірка, яку я вам рекомендую ставити. Ця версія встановлює все, що тільки можна: 2D/3D симулятори та програми сприйняття. Для того, щоб встановити цю версію, пишемо в терміналі:

sudo apt install ros-noetic-desktop-full

Desktop Instal - цей пакет містить у собі всі компоненти на базі ROS, а також такі елементи як rqt і rviz. Для встановлення цієї версії пишемо в терміналі наступну команду

sudo apt install ros-noetic-desktop

3. ROS-Base: (Bare Bones) – це просто голі бібліотеки та пакети ROS. Це той варіант, який підійде для професіоналів. Для встановлення цього варіанта пишемо в терміналі:

sudo apt install ros-noetic-ros-base

Також ви можете встановити будь-який конкретний пакет ROS, скориставшись командою:

sudo apt install ros-noetic

Встановлюємо пакет bash, де ми будемо використовувати ROS. Пишемо в терміналі:

source /opt/ros/noetic/setup.bash

До цього часу ви встановлювали все, що потрібно для запуску основних пакетів ROS. Для створення та керування власними робочими просторами ROS існують різні інструменти та вимоги, які розповсюджуються окремо. Наприклад, rosinstall - інструмент командного рядка, що часто використовується, який дозволяє легко завантажувати безліч вихідних кодів для пакетів ROS за допомогою однієї команди. Для його встановлення пишемо у терміналі:

sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential

Перед тим, як ви зможете використовувати багато інструментів ROS, вам потрібно буде ініціалізувати rosdep. Він дозволяє легко встановлювати системні залежності для вихідного коду, який ви хочете скомпілювати, і потрібний для запуску деяких основних компонентів у ROS. Якщо ви ще не встановили rosdep, саме час його встановити. Для цього пишемо у терміналі:

sudo apt встановити python3-rosdep

Для ініціалізації rosdep пишемо в терміналі:

sudo rosdep init

І потім пишемо наступне:

rosdep update

В результаті даної маніпуляється проводиться встановлення ROS на обран платформу Ubuntu 20.04

RViz (Robot Visualization) - це інструмент для візуалізації даних в робототехніці. Він є однією з ключових складових систем роботів ROS (Robot Operating System) та використовується для відображення інформації про роботів у вигляді тривимірних моделей, точок чи інших візуальних елементів.

Основні характеристики RViz:

1. Візуалізація сенсорів:

RViz дозволяє візуалізувати дані, отримані від різних сенсорів, таких як лідари, камери, відомості з відомостей про та інші.

2. Модель робота:

Ви можете завантажити та візуалізувати тривимірну модель вашого робота. Це дозволяє спостерігати за рухом робота та його взаємодією з оточуючим середовищем.

3. Відображення шляхів:

RViz може відображати шляхи, які були заплановані для руху робота, що корисно при тестуванні та налагодженні алгоритмів управління.

4. Візуалізація та аналіз даних:

Користувач може візуалізувати та аналізувати різноманітні дані, такі як облака точок, вектори, траєкторії тощо.

5. Інтерактивний інтерфейс:

RViz має інтерактивний інтерфейс, що дозволяє користувачам взаємодіяти з візуалізацією, змінювати параметри та отримувати реальний час зміни.

RViz є потужним інструментом для розробників та дослідників в галузі робототехніки для візуалізації та налагодження алгоритмів управління та обробки даних в реальному часі.

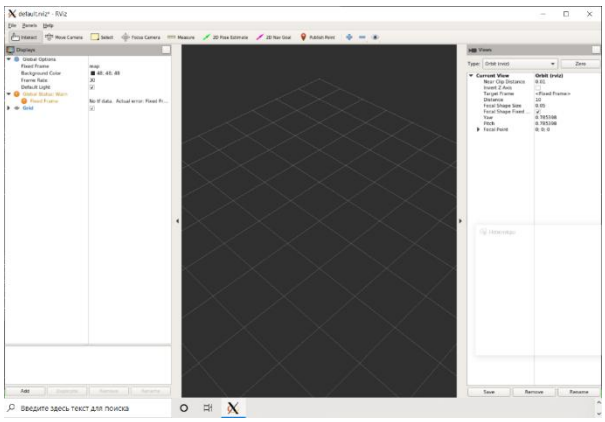


Рисунок 3.3. – Використання Rviz

Gazebo - це відкрите програмне забезпечення для симуляції роботів, яке дозволяє моделювати та тестувати робототехнічні системи в віртуальному

середовищі. Він надає інструменти для створення динамічних моделей роботів, встановлення різних сценаріїв та взаємодії між різними об'єктами у симульованому світі.

Основні риси Gazebo:

1. Симуляція фізики:

Gazebo враховує фізичні властивості об'єктів, що дозволяє точно моделювати рух, зіткнення та взаємодію між об'єктами.

2. Віртуальні середовища:

Користувач може створювати власні віртуальні світи або використовувати готові моделі для симуляції робототехнічних завдань у різних умовах.

3. Моделювання роботів:

Розширений інструментарій для створення моделей різних типів роботів та додаткових сенсорів.

4. Візуалізація:

Надає візуалізацію симуляції, що дозволяє спостерігати за роботами та середовищем у реальному часі.

5. Взаємодія з кодом:

Розвита програмований інтерфейс для взаємодії з симуляцією з використанням різних мов програмування, таких як C++, Python та інші.

Gazebo часто використовується в робототехніці для розробки та тестування алгоритмів управління, систем обробки зображень та інших технічних рішень перед їхнім впровадженням на реальних роботах.

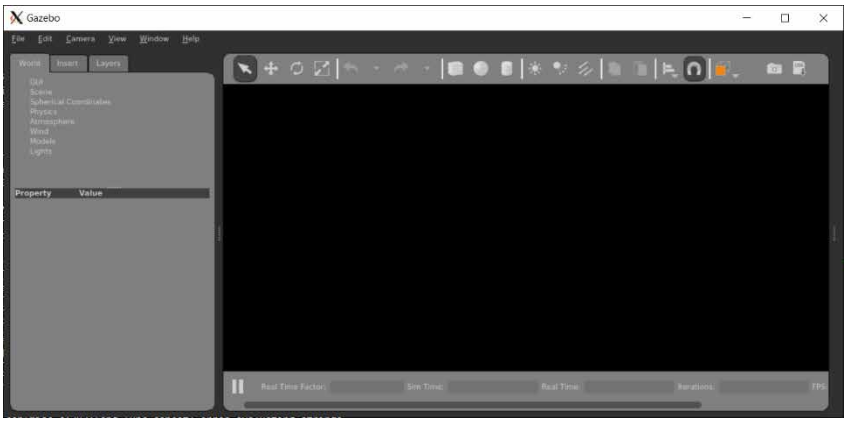


Рисунок 3.4. – Використання Gazebo

Опишимо по етапну розробку кожної з частини програмного забезпечення разом з наведеним кодом.

Функція отримання символу з консолі без блокування введення.

Використовується для отримання клавіші з клавіатури.

```
char getch_noblocking()
{
  fd_set set;
  struct timeval timeout;
  int rv;
  char buff = 0;
  int len = 1;
  int filedesc = 0;
  FD_ZERO(&set);
  FD_SET(filedesc, &set);

  timeout.tv_sec = 0;
  timeout.tv_usec = 1000;

  rv = select(filedesc + 1, &set, NULL, NULL, &timeout);

  struct termios old = {0};
  if (tcgetattr(filedesc, &old) < 0)
    ROS_ERROR("tcsetattr()");
  old.c_lflag &= ~ICANON;
```

```
  old.c_lflag &= ~ECHO;
  old.c_cc[VMIN] = 1;
  old.c_cc[VTIME] = 0;
  if (tcsetattr(filedesc, TCSANOW, &old) < 0)
    ROS_ERROR("tcsetattr ICANON");

  if(rv == -1)
    ROS_ERROR("select");
  else if(rv == 0)
    ;
    //ROS_INFO("no_key_pressed");
  else
    read(filedesc, &buff, len );

  old.c_lflag |= ICANON;
  old.c_lflag |= ECHO;
  if (tcsetattr(filedesc, TCSADRAIN, &old) < 0)
    ROS_ERROR ("tcsetattr ~ICANON");
  return (buff);
}
```

Визначення можливих станів місії та руху

```
static enum Mission_STATE {
  IDLE,
  TAKEOFF,
  KEYBOARD_CTR,
  LAND,
} mission_state=IDLE;

static enum MANEUVER{
  KB_NONE,
  KB_TAKEOFF,
  KB_LAND,
  KB_FORWARD,
  KB_BACKWARD,
```

```
  KB_LEFT,
  KB_RIGHT,
  KB_UP,
  KB_DOWN,
  KB_TURNLEFT,
  KB_TURNRIGHT,
} kb_state=KB_NONE;

static mavros_msgs::State current_state;
static bool local_pose_received = false;
static SE3 latest_pos;
static geometry_msgs::PoseStamped last_published_pose;
static geometry_msgs::PoseStamped publish_pose;
static double h_speed;
static double v_speed;
static double turn_speed;
static double update_rate=20.0;
```

Функція зворотнього виклику для отримання стану системи від mavros. Використовується для оновлення глобальної змінної current_state.

```
void state_cb(const mavros_msgs::State::ConstPtr& msg){
  current_state = *msg;
}
```

Функція зворотнього виклику для отримання локального положення безпілотника.

Використовується для оновлення глобальної змінної latest_pos.

```
void local_odom_cb(const geometry_msgs::PoseStampedConstPtr& msg){
  mtx_states_RW.lock();
  local_pose_received = true;
  latest_pos.translation().x()=msg->pose.position.x;
  latest_pos.translation().y()=msg->pose.position.y;
  latest_pos.translation().z()=msg->pose.position.z;
```

```
  latest_pos.so3() = SO3(Quaterniond(msg->pose.orientation.w,msg-
>pose.orientation.x,
                                     msg->pose.orientation.y,msg-
>pose.orientation.z));
  mtx_states_RW.unlock();
}
```

Функція конвертації об'єкта SE3 в об'єкт geometry_msgs::PoseStamped. Використовується для публікації нового положення безпілотника.

```
geometry_msgs::PoseStamped update_pose_from_se3(SE3 se3)
{
  geometry_msgs::PoseStamped pose;
  pose.header.frame_id = "world";
  pose.pose.position.x = se3.translation().x();
  pose.pose.position.y = se3.translation().y();
  pose.pose.position.z = se3.translation().z();
  pose.pose.orientation.x = se3.so3().unit_quaternion().x();
  pose.pose.orientation.y = se3.so3().unit_quaternion().y();
  pose.pose.orientation.z = se3.so3().unit_quaternion().z();
  pose.pose.orientation.w = se3.so3().unit_quaternion().w();
  return pose;
}
```

А також опишемо головну функцію

```
int main(int argc, char **argv)
{
    // Ініціалізація ROS та створення вузла з ім'ям "offb_node"
    ros::init(argc, argv, "offb_node");
    ros::NodeHandle nh;

    // Створення підписника для отримання стану від mavros
    ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>("mavros/state",
10, state_cb);

    // Створення видавця для публікації локального положення безпілотника
```

```
    ros::Publisher local_pos_pub =
nh.advertise<geometry_msgs::PoseStamped>("mavros/setpoint_position/local", 10);

    // Створення службових клієнтів для армування та зміни режиму системи mavros
    ros::ServiceClient arming_client =
nh.serviceClient<mavros_msgs::CommandBool>("mavros/cmd/arming");
    ros::ServiceClient set_mode_client =
nh.serviceClient<mavros_msgs::SetMode>("mavros/set_mode");

    // Встановлення частоти публікації
    ros::Rate rate(20.0);

    // Очікування на з'єднання з mavros
    while(ros::ok() && !current_state.connected){
        ros::spinOnce();
        rate.sleep();
    }

    // Ініціалізація об'єкта geometry_msgs::PoseStamped для визначення початкового
положення
    geometry_msgs::PoseStamped pose;
    pose.pose.position.x = 0;
    pose.pose.position.y = 0;
    pose.pose.position.z = 2;

    // Створення об'єкта для встановлення режиму "OFFBOARD"
    mavros_msgs::SetMode offb_set_mode;
    offb_set_mode.request.custom_mode = "OFFBOARD";

    // Створення об'єкта для армування безпілотника
    mavros_msgs::CommandBool arm_cmd;
    arm_cmd.request.value = true;

    // Час останнього запиту для управління режимом та армуванням
    ros::Time last_request = ros::Time::now();
```

```
    // Основний цикл програми
    while(ros::ok()){
        // Перевірка та зміна режиму OFFBOARD
        if( current_state.mode != "OFFBOARD" &&
            (ros::Time::now() - last_request > ros::Duration(2.0))){
            if( set_mode_client.call(offb_set_mode) &&
                offb_set_mode.response.mode_sent){
                ROS_INFO("Offboard enabled");
            }
            last_request = ros::Time::now();
        } else {
            // Перевірка та здійснення армування безпілотника
            if( !current_state.armed &&
                (ros::Time::now() - last_request > ros::Duration(2.0))){
                if( arming_client.call(arm_cmd) &&
                    arm_cmd.response.success){
                    ROS_INFO("Vehicle armed");
                }
                last_request = ros::Time::now();
            }
        }

        // Публікація нового положення безпілотника
        local_pos_pub.publish(pose);

        ros::spinOnce();
        rate.sleep();
    }

    return 0;
}
```

Так як проект побудований на схемі взаємодій деяких систем що формують систему ROS опишемо кожну з цих папок для розуміння того що вхожить до їх складу.

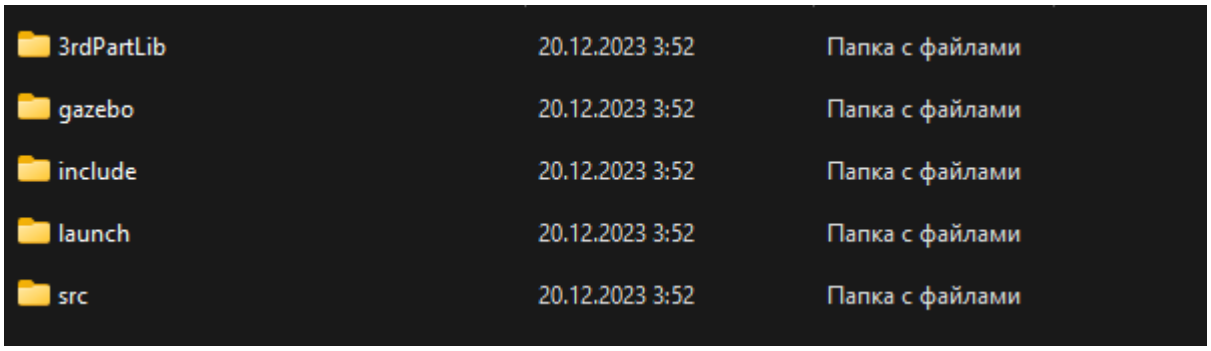


Рисунок 3.5. – Корінь папки проекту

Опишемо файли що знаходяться в папці 3rdPartLib.

Файл з назвою "install3rdPartLib.sh" є скриптом у середовищі Unix-подібних операційних систем. Скрипт використовує для автоматизованої установки сторонніх бібліотек чи пакетів у проектах розробки програмного забезпечення.

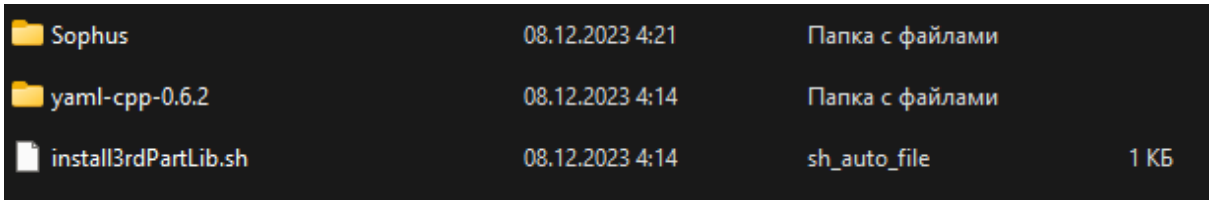


Рисунок 3.6. – Папка 3rdPartLib

В роботі була використана YAML-CPP. YAML-CPP - це бібліотека на мові C++, яка дозволяє читати та записувати дані у форматі YAML. YAML (YAML Ain't Markup Language) - це людино-читаемий формат обміну даними, який часто використовується для конфігураційних файлів та обміну даними між мовами програмування.

Sophus - це бібліотека на мові C++, яка спеціалізується на роботі з групами Лі, які виникають в робототехніці та комп'ютерному зорі. Групи Лі включають у себе групи обертань та групи ейлерових перетворень, які широко використовуються для представлення та обробки обертань та положень в просторі.

Sophus надає зручний інтерфейс для роботи з обертаннями та трансформаціями в робототехніці та комп'ютерному зорі. Вона дозволяє вам ефективно виконувати операції, такі як обчислення експоненціального відображення, логарифмічного відображення, композиції та інші операції над обертаннями та трансформаціями.

Папка "gazebo" містить конфігураційні файли, моделі роботів, сценарії, текстури та інші ресурси, які використовуються в Gazebo для симуляції роботів та інших об'єктів у віртуальному середовищі. Ця папка зберігається в директорії робочого простору для проектів, які використовують Gazebo.

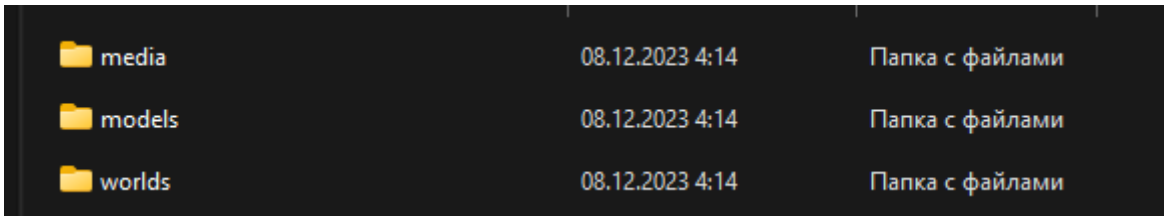


Рисунок 3.7. – Папка gazebo

Вміст папки "gazebo" виглядає наступним чином:

models/ - папка, де розташовані моделі роботів, об'єктів або інших об'єктів, які можуть бути використані у симуляціях.

worlds/ - папка, де можуть зберігатися конфігурації сцен, в яких відбувається симуляція.

textures/ - папка, де можуть розміщуватися текстури для візуалізації в симуляціях.

config/ - папка, де можуть зберігатися конфігураційні файли для Gazebo.

В папці include знаходиться папка realsense_gazebo_plugin в якій знаходяться файли що використовує ROS (Robot Operating System) і Gazebo для симуляції з камерами RealSense, бібліотека realsense_gazebo_plugin та є частиною ROS-пакету.

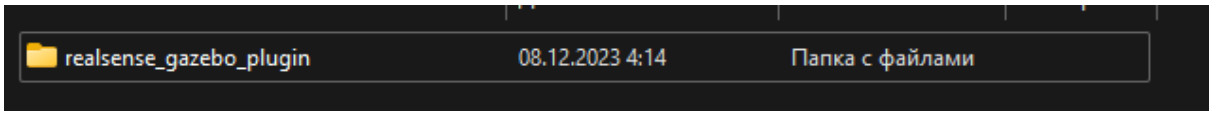


Рисунок 3.8. – Папка include

Файли .launch в контексті ROS (Robot Operating System) використовуються для запуску роботизованих програм, моделей та нод ROS. Ці файли містять конфігурації для запуску різних компонентів системи, таких як датчики, вузли обробки даних, симуляційні середовища тощо.

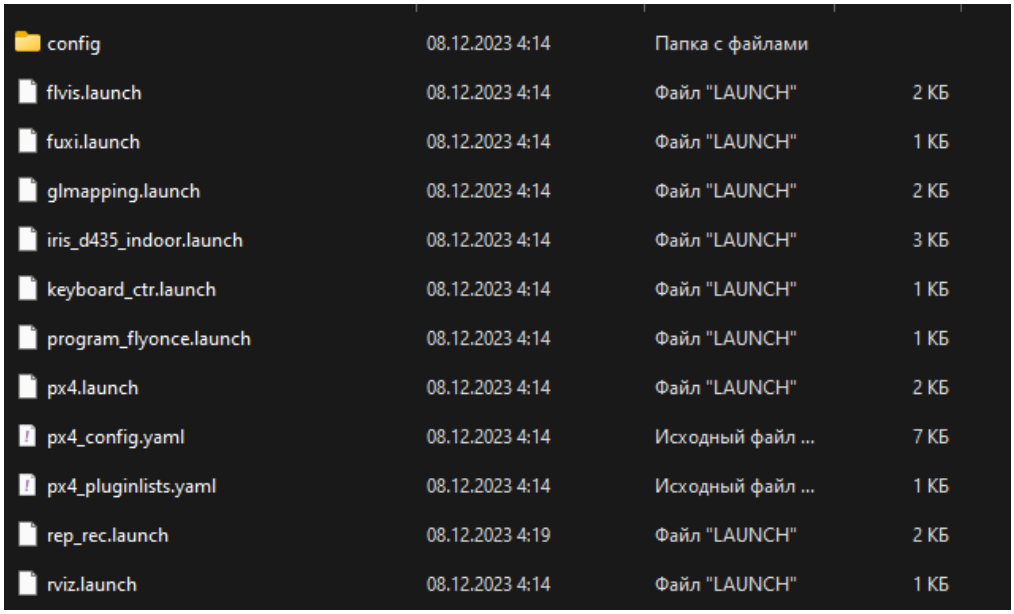


Рисунок 3.9. – Папка launch

Папка src містить наступні файли:

1. RealSensePlugin.cpp
2. gazebo_ros_realsense.cpp
3. gt2visionpose.cpp
4. keyboard_ctl.cpp
5. offb.cpp
6. папки utils та movement

Опишемо кожен файл детальніше:

1. RealSensePlugin.cpp:

- Цей файл містить реалізацію плагіну (Gazebo plugin) для інтеграції камер RealSense з симулятором Gazebo.
- Він включає логіку для відтворення реалістичних зображень та поведінки камер у симуляції.

2. gazebo_ros_realsense.cpp:
   - Цей файл, є частиною ROS-пакету для взаємодії з симулятором Gazebo та камерами RealSense.
   - Включає вузли ROS для обробки даних від камер або взаємодії з іншими компонентами системи.
3. gt2visionpose.cpp:
   - Цей файл, має відношення до визначення положення об'єктів на основі візуальних даних.
   - Містить реалізацію алгоритмів комп'ютерного зору для визначення позицій об'єктів у візуальному полі.
4. keyboard_ctl.cpp:
   - Цей файл включає реалізацію класу або модуля для керування системою за допомогою клавіатури.
   - Містить обробник подій клавіш та логіку керування для робота чи іншого об'єкта.
5. offb.cpp:
   - Цей файл, стосується системи відслідковування об'єктів або польоту за допомогою клавіатури.
   - Містить логіку керування польотом та позиціонуванням у просторі.
6. Папка utils:

- Ця папка, містить різні корисні службові файли і інструменти, які використовуються в інших частинах програмного забезпечення.

7. Папка movement:

- Ця папка може иістить файли, пов'язаний з рухом і керуванням об'єктами в симуляції.

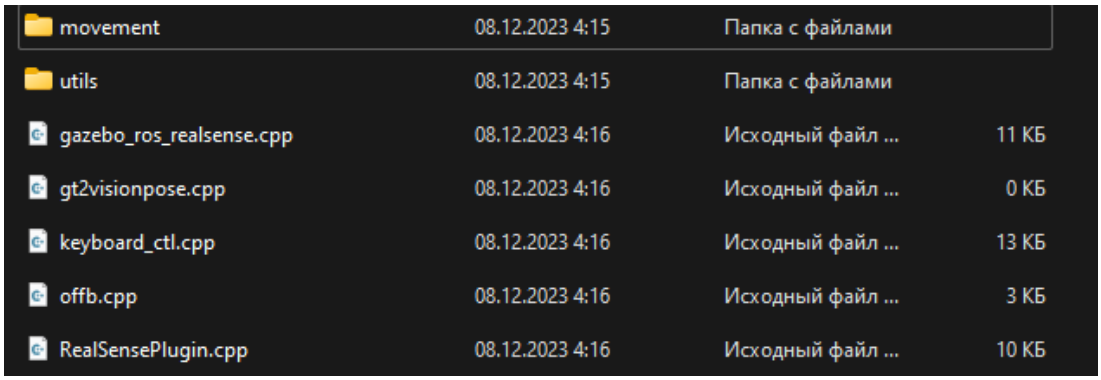


Рисунок 3.10. – Папка src

Розглянемо просту реалізацію пошуку оптимального маршруту за допомогою розробленого алгоритму. Для цього розмістимо на площу декілька різних об'єктів, які можуть бути перешкодами або цілями для безпілотного літального апарату (БПЛА). Об'єкти можуть включати будівлі, дерева чи інші теренові об'єкти.

Розглянемо сценарій, де на площі розташовані декілька об'єктів: декілька як цілі для БПЛА та один як можлива перешкода. Ми будемо використовувати наш алгоритм для обчислення оптимального маршруту БПЛА, який обходить перешкоду та досягає обраної цілі.

Спочатку, ініціалізуємо початкове положення та стани об'єктів на площі. Потім визначимо цілі, до яких ми хочемо, щоб БПЛА дійшов, і вказатимемо можливі перешкоди. Запустимо наш алгоритм пошуку маршруту, який автоматично обчислить оптимальний шлях для досягнення цілей, уникнення перешкод та мінімізації витрат енергії.

Результатом буде маршрут, який БПЛА буде слідувати, обходячи перешкоду та ефективно досягаючи цілей. Цей підхід може бути використаний в різних сценаріях, таких як автономна доставка, патрулювання або інші завдання, де оптимальне планування маршруту є важливим елементом.

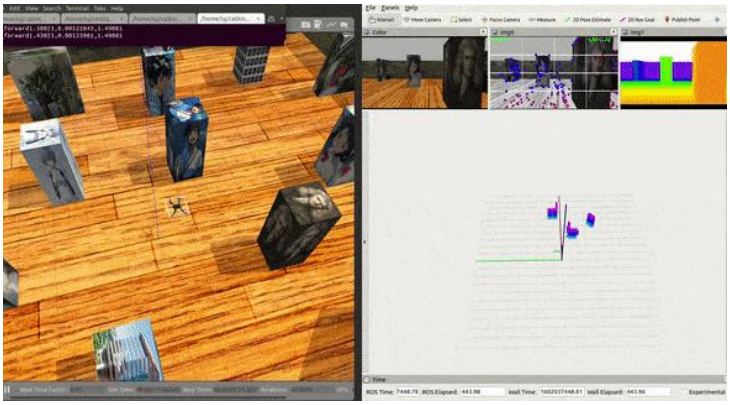


Рисунок 3.11. Загальний вигляд програмного продукту

Під час подорожі на місцевості дрон використовує вбудовані сенсори та алгоритми обробки даних для аналізу навколишнього середовища та будування тривимірної карти. Цей процес, відомий як картографування, є ключовою частиною систем автономної навігації для дронів. Під час аналізу дрон отримує інформацію про рельєф місцевості, розташування об'єктів та інші аспекти оточуючого простору.

Отримана тривимірна карта може використовуватися для планування оптимального маршруту на основі географічних особливостей місцевості. Дрон може автоматично побудувати маршрут, обходячи різні об'єкти або перешкоди на своєму шляху. Це особливо важливо в сценаріях, де потрібно облітати конкретні об'єкти або виконувати завдання навколо конкретних місць інтересу.

Побудована карта дозволяє дрону ефективно планувати свій рух, уникати перешкод та мінімізувати витрати енергії. Такий метод застосовується у різноманітних областях, включаючи агрономію, вивчення дикої природи, пошук

та рятування, де точне визначення маршруту може визначати успішність виконання завдань.

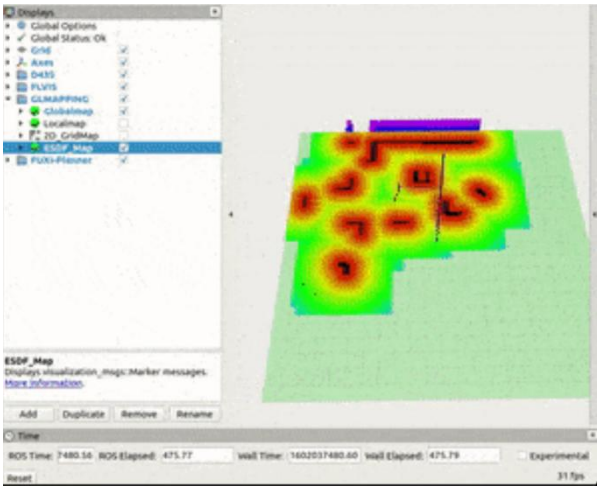


Рисунок 3.12. Побудова карти

При подорожі по місцевості дрон використовує свої вбудовані камери для аналізу оточуючого середовища та збирання великої кількості візуальних даних. Цей процес включає захоплення зображень і відео високої роздільної здатності з повітря та їхнє подальше оброблення для отримання повної та деталізованої картини навколишньої місцевості.

Зібрана візуальна інформація дозволяє дрону виявляти різноманітні деталі, такі як ландшафт, рельєф, водойми, рослинність та будівлі. Шляхом об'єднання цих даних, дрон може побудувати загальну картину або мапу області, що забезпечує повний обзор місцевості та розташування різних об'єктів.

Отримана в результаті аналізу камер інформація може використовуватися для різних цілей, таких як планування оптимальних маршрутів, визначення точок інтересу, моніторинг стану довкілля чи виконання завдань обстеження. Цей підхід робить дрон ефективним інструментом для вивчення і визначення параметрів великих територій та надає можливість оперативного прийняття рішень заснованих на візуальних даних.

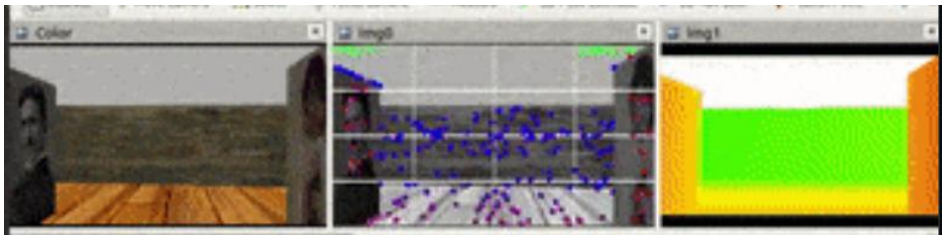


Рисунок 4.3. Побудова карти за допомогою камер вбудованих в дрон

РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ ПОДАЛЬШОЇ РОБОТИ

4.1. Аналіз результатів моделювання

Порівняння очікуваних і отриманих результатів є ключовим етапом оцінки ефективності системи симуляції БПЛА. Воно допомагає виявити розбіжності між теоретичними моделями і реальними тестами, що дає змогу оптимізувати систему. Очікувані результати базуються на вимогах до точності польоту, стабільності і уникненні перешкод, а відхилення часто пов’язані з обмеженнями сенсорів, неточностями моделей або помилками в алгоритмах. Аналіз причин відхилень дозволяє визначити, чи потрібно доопрацювати симуляцію, а також підвищити її реалістичність і надійність. Точність залежить від коректності фізичних моделей, налаштувань сенсорів і алгоритмів керування, а також від порівняння з реальними даними. Покращення досягається вдосконаленням моделей, калібруванням системи і використанням більш точних сенсорів.

4.2. Порівняння з іншими симуляторами

Таблиця порівняння функцій між створеною системою симуляції для БПЛА та існуючими системами надасть детальну картину того, як ваш проєкт співвідноситься з іншими подібними рішеннями на ринку. Вона допоможе оцінити, наскільки ваша система задовольняє вимоги, порівняно з іншими популярними платформами симуляції, такими як Gazebo, AirSim, Unity, Webots тощо.

Функція	Розроблена система симуляції	Gazebo	AirSim	Unity	Webots
3D візуалізація	Так	Так	Так	Так	Так
Моделювання аеродинаміки БПЛА	Так	Так	Так	Частково	Частково
Моделювання сенсорів (GPS, IMU, Лідар)	Так	Так	Так	Частково	Так
Автономне керування	Так	Так	Так	Так	Так
Підтримка реального середовища	Так	Так	Так	Так	Так
Моделювання перешкод та навігація	Так	Так	Так	Так	Так
Підтримка різних типів БПЛА	Так	Так	Так	Так	Так
Інтеграція з ROS	Так	Так	Так	Частково	Так
Штучний інтелект та машинне навчання	Планується	Планується	Так	Так	Так
Кросплатформеність (Windows/Linux)	Так	Так	Так	Так	Так
Інтерфейс користувача	Так	Так	Так	Так	Так

Реалістичність поведінки	Висока	Висока	Висока	Середня	Середня
Підтримка різних сценаріїв польоту	Так	Так	Так	Так	Так

Усі системи, включаючи розроблену, підтримують 3D візуалізацію, що є критично важливим для реалістичного відображення польоту БПЛА в різних умовах. Це дозволяє чітко бачити процес симуляції, оцінювати її результати та коригувати налаштування відповідно до потреб.

Розроблена система підтримує модель аеродинаміки як для фіксованого, так і для роторного крил, що дозволяє більш точно симулювати поведінку БПЛА в різних умовах, включаючи підйом, стабільність та керованість. Газебо та AirSim також підтримують такі моделі, що робить їх конкурентними в цьому аспекті.

Усі системи забезпечують підтримку сенсорів, таких як GPS, IMU та Лідар, що необхідно для реалістичної симуляції навігації та обходу перешкод. Однак у Unity та Webots підтримка може бути частковою, що обмежує точність і можливості в моделюванні реальних умов.

Розроблена система підтримує автономне керування з використанням алгоритмів, таких як PID та LQR, а також інтеграцію з алгоритмами машинного навчання для покращення ефективності управління. Інші платформи, такі як AirSim та Gazebo, також надають ці можливості, але розроблена система може бути гнучкішою завдяки більшій налаштовуваності алгоритмів.

Система підтримує складні сценарії навігації, включаючи уникнення перешкод, що є важливим компонентом для розробки автономних систем. Інші системи також пропонують ці можливості, хоча реалістичність може варіюватися залежно від конкретної реалізації в системах, таких як Gazebo чи Webots.

Розроблена система підтримує інтеграцію з ROS, що дозволяє здійснювати більш гнучке управління та взаємодію з іншими модулями, такими як системи обробки даних або зовнішні контролери. Подібні інтеграції доступні і в Gazebo та Webots, що робить ці платформи популярними для розвитку автономних систем.

Хоча в розробленій системі ще не реалізовано використання ШІ для машинного навчання, це є частиною планів розвитку. Аналогічно, в системах AirSim та Unity використання ШІ активно інтегрується для підвищення ефективності систем автономного керування, і в майбутньому це може стати конкурентною перевагою.

Усі згадані системи мають можливість працювати як на Windows, так і на Linux, що є важливим для забезпечення широкої сумісності і використання на різних платформах. Це дозволяє розробникам адаптувати симуляцію до своїх потреб і використовувати її в різноманітних середовищах.

Усі системи мають зручний графічний інтерфейс для налаштування та моніторингу симуляцій, що забезпечує зручність у використанні і можливість візуалізації результатів. Це критично важливо для тих, хто працює зі складними симуляціями, як це відбувається у розробленій системі та у подібних рішеннях.

Розроблена система, на основі вибору аеродинамічних моделей та високої точності сенсорів, має високу реалістичність, що дозволяє точніше моделювати умови реального польоту. Газебо та AirSim також забезпечують високий рівень реалістичності, що дозволяє їх використовувати для розробки та тестування автономних систем в умовах, наближених до реальних.

Це порівняння дозволяє оцінити всі переваги та недоліки розробленої системи в контексті існуючих платформ, що використовуються для симуляції БПЛА та роботизованих систем. Залежно від специфічних потреб проекту, це порівняння допоможе зробити правильний вибір інструментів і методів для подальшої розробки та вдосконалення розробленої системи.

4.3. Обмеження створеної системи

Технічні обмеження розробленої системи стосуються, в основному, продуктивності та реалізму моделювання. Однією з основних проблем є висока вимогливість до обчислювальних ресурсів, особливо коли йдеться про складні сценарії та детальне моделювання. Для точного відображення поведінки БПЛА в різноманітних умовах (наприклад, реалістичне моделювання аеродинаміки, сенсорів, навігації) система потребує значних обчислювальних потужностей. Тому для забезпечення належної продуктивності при складних симуляціях необхідне використання потужних комп'ютерних систем, що може обмежувати її використання в обмежених технічних умовах.

Крім того, при високій складності моделювання, такій як інтеграція великої кількості сенсорів (лідар, камера, IMU) та реалістична симуляція навколишнього середовища, може спостерігатися зниження реалістичності моделі або збільшення затримок у відображенні. Це може бути проблемою, особливо для сценаріїв, що потребують високої точності та швидкої реакції на зміну умов. Відсутність активної інтеграції з системами штучного інтелекту також обмежує адаптивність системи до непередбачених змін у середовищі, що знижує її ефективність у реальних умовах.

Розроблена система має деякі обмеження в сценаріях використання, які зумовлені як технічними, так і методологічними аспектами. Одним із таких обмежень є складність реалізації дуже динамічних або змінних сценаріїв, що вимагають оперативної адаптації. Наприклад, в умовах зміни навколишнього середовища, таких як вітер або зміни в ландшафті, система може мати труднощі з адаптацією без активного використання алгоритмів машинного навчання, яких наразі немає в розробленій версії. Крім того, при моделюванні інтенсивних

ситуацій, таких як аварії або різке пошкодження сенсорів, система не завжди може забезпечити повну точність у відображенні цих сценаріїв.

Ще одним обмеженням є підтримка складних сценаріїв з великою кількістю об'єктів або перешкод. У таких випадках система може потребувати додаткової оптимізації та ресурсів для коректної симуляції, що не завжди можливо за умов обмежених апаратних засобів.

Хоча система підтримує основні платформи, такі як Windows і Linux, існують деякі обмеження у її сумісності з іншими операційними системами або спеціалізованими апаратними платформами. Наприклад, платформи з більш специфічними вимогами до апаратних засобів, такими як вбудовані системи на базі ARM або деякі спеціалізовані сервери, можуть не підтримувати цю систему без додаткових налаштувань або адаптацій.

Крім того, система має обмежену підтримку певних форматів даних, що може ускладнити інтеграцію з іншими платформами або сторонніми додатками. Це може стати проблемою при необхідності працювати з нестандартними форматами для сенсорних даних або сценаріїв моделювання, які не входять в основний набір підтримуваних формів.

Такі обмеження вказують на необхідність подальшої оптимізації системи та розширення її сумісності з іншими платформами для забезпечення більш широкого кола можливостей у майбутньому.

4.4. Перспективи розвитку

Розроблене рішення дозволяє моделювати групову поведінку безпілотних літальних апаратів (БПЛА), зокрема роєві системи, де кілька БПЛА діють спільно з узгодженими цілями та стратегіями. Система передбачає інтеграцію моделей координації між БПЛА, що забезпечує можливість спільного виконання завдань, таких як політ у заданому напрямку, уникнення перешкод та спільна

навігація. Однак реалізація роєвої поведінки вимагає додаткових налаштувань для врахування взаємодії між безпілотниками, що можуть мати різні ролі та задачі в рамках одного сценарію. Це вимагає значних обчислювальних потужностей для забезпечення реалістичності поведінки і синхронізації рухів БПЛА в групі.

Інтеграція системи з машинним навчанням або нейронними мережами може суттєво покращити ефективність симуляції та адаптацію до змінюваних умов. Наприклад, використання нейронних мереж для покращення точності прогнозування поведінки БПЛА у складних ситуаціях або оптимізації алгоритмів навігації може забезпечити гнучкість і адаптивність системи. Однак наразі в системі не реалізована повноцінна інтеграція з технологіями машинного навчання, що обмежує можливості для динамічної адаптації до непередбачуваних змін середовища. Це є напрямком для подальшого розвитку та вдосконалення системи, що дозволить забезпечити вищу ефективність в умовах реальних польотів і складних сценаріїв.

Для покращення системи можна впровадити підтримку тренувань моделей керування за допомогою методів підкріпленого навчання (reinforcement learning, RL). Це дозволяє БПЛА вчитися на основі досвіду, оптимізуючи свою поведінку в реальному часі, наприклад, у процесі уникнення перешкод або адаптації до змін навколишнього середовища. Метод RL дозволяє створювати більш адаптивні алгоритми керування, які можуть покращити ефективність автономного польоту, навчання на основі зворотного зв'язку і розпізнавання моделей поведінки. У поточній версії система може бути вдосконалена для включення таких алгоритмів, що є перспективним напрямком для розвитку.

Розроблена система має потенціал для моделювання польотів БПЛА в урбаністичних середовищах або в умовах закритих приміщень. Це вимагає специфічних моделей для створення реалістичних міських пейзажів та внутрішніх середовищ, де можуть бути складні перешкоди, обмежена видимість

і різноманітні фактори, що впливають на політ (наприклад, відбиття сигналів або турбулентність у приміщеннях). В системі вже реалізовані базові функції для таких сценаріїв, але для досягнення максимальної реалістичності необхідно додатково оптимізувати моделі вітру, відображень та взаємодії з навколишнім середовищем. Крім того, важливо забезпечити підтримку для управління БПЛА в умовах обмеженого простору, що є важливою задачею для точних польотів у приміщеннях або міських зонах з багатьма об'єктами.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено програмно-апаратне рішення для моделювання та симуляції поведінки безпілотних літальних апаратів у різноманітних умовах навколишнього середовища. Основна мета полягала у створенні платформи, яка дозволяє досліджувати, тестувати та вдосконалювати алгоритми керування БПЛА з можливістю інтеграції зовнішніх систем та симуляції сенсорних даних.

Під час реалізації проєкту було проаналізовано існуючі симуляційні системи, такі як Gazebo, AirSim, Unity та Webots, та на основі виявлених переваг і недоліків сформовано концепцію власної симуляції. Розроблена система забезпечує підтримку 3D-візуалізації, моделювання аеродинаміки для різних типів БПЛА, сенсорного забезпечення (GPS, IMU, Лідар), а також автономного керування з використанням класичних та сучасних алгоритмів.

Особлива увага приділялася підтримці інтеграції з ROS, що дозволяє забезпечити масштабованість та взаємодію з іншими модулями, а також відкриває можливості для використання в дослідницьких проєктах. Було проведено серію експериментів у фіксованих умовах, що дало змогу оцінити точність маршруту, стабільність роботи та рівень затримки. Аналіз результатів показав задовільну відповідність очікуванням і дозволив виявити потенційні напрямки для вдосконалення.

Незважаючи на окремі технічні обмеження, такі як обмежена продуктивність на слабших системах або потреба в додатковому налаштуванні для деяких сценаріїв, розроблена симуляційна платформа демонструє високу реалістичність поведінки БПЛА та придатна для подальшого розширення. Серед можливих векторів розвитку — реалізація навчання з підкріпленням, підтримка роєвої поведінки, моделювання урбаністичних середовищ та тісніша інтеграція з технологіями штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Siciliano B., Khatib O. Springer Handbook of Robotics. 2nd ed. Cham: Springer, 2016. 2224 p.

2. Valavanis K.P., Vachtsevanos G.J. Handbook of Unmanned Aerial Vehicles. Dordrecht: Springer, 2015. 2600 p.

3. Shakhatreh H., Sawalmeh A.H., Al-Fuqaha A., Dou Z., Almaita E., Khalil I., Othman N.S., Khreishah A., Guizani M. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. IEEE Access. 2019. Vol. 7. P. 48572–48634.

4. Zhou Z., Zhang C., Liu J., Cheng P., Li Y., Shen X. Energy-Efficient UAV Communication with Trajectory Optimization. IEEE Transactions on Wireless Communications. 2018. Vol. 17, No. 6. P. 3987–3999.

5. Kumar V., Michael N. Opportunities and Challenges with Autonomous Micro Aerial Vehicles. The International Journal of Robotics Research. 2012. Vol. 31, No. 11. P. 1279–1291.

6. Mahony R., Kumar V., Corke P. Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor. IEEE Robotics & Automation Magazine. 2012. Vol. 19, No. 3. P. 20–32.

7. Mellinger D., Michael N., Kumar V. Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors. The International Journal of Robotics Research. 2012. Vol. 31, No. 5. P. 664–674.

8. Bouabdallah S., Murrieri P., Siegwart R. Design and Control of an Indoor Micro Quadrotor. Proceedings of the 2004 IEEE International Conference on Robotics and Automation. 2004. Vol. 5. P. 4393–4398.

9. Kendoul F. Survey of Advances in Guidance, Navigation, and Control of Unmanned Rotorcraft Systems. Journal of Field Robotics. 2012. Vol. 29, No. 2. P. 315–378.

10. Pounds P., Mahony R., Gresham J. Towards Dynamically Favourable Quad-Rotor Aerial Robots. Australasian Conference on Robotics and Automation. 2004. P. 1–6.

11. Hoffmann G.M., Huang H., Waslander S.L., Tomlin C.J. Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment. Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit. 2007. P. 1–20.

12. Faessler M., Fontana F., Forster C., Mueggler E., Pizzoli M., Scaramuzza D. Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle. Journal of Field Robotics. 2016. Vol. 33, No. 4. P. 431–450.

13. Krajník T., Vonásek V., Fišer D., Faigl J. AR-Drone as a Platform for Robotic Research and Education. Research and Education in Robotics—EUROBOT 2011. 2011. P. 172–186.

14. Meier L., Tanskanen P., Heng L., Lee G.H., Fraundorfer F., Pollefeys M. PIXHAWK: A Micro Aerial Vehicle Design for Autonomous Flight Using Onboard Computer Vision. Autonomous Robots. 2012. Vol. 33, No. 1–2. P. 21–39.

15. Forster C., Pizzoli M., Scaramuzza D. SVO: Fast Semi-Direct Monocular Visual Odometry. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 2014. P. 15–22.

16. Engel J., Schöps T., Cremers D. LSD-SLAM: Large-Scale Direct Monocular SLAM. European Conference on Computer Vision. 2014. P. 834–849.

17. Mur-Artal R., Montiel J.M.M., Tardós J.D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics. 2015. Vol. 31, No. 5. P. 1147–1163.

18. Qin T., Li P., Shen S. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. IEEE Transactions on Robotics. 2018. Vol. 34, No. 4. P. 1004–1020.

19. Bloesch M., Omari S., Hutter M., Siegwart R. Robust Visual Inertial Odometry Using a Direct EKF-Based Approach. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2015. P. 298–304.

20. Klein G., Murray D. Parallel Tracking and Mapping for Small AR Workspaces. 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality. IEEE, 2007. P. 225–234.

21. Mellinger D., Kumar V. Minimum Snap Trajectory Generation and Control for Quadrotors. 2011 IEEE International Conference on Robotics and Automation. IEEE, 2011. P. 2520–2525.

22. Burri M., Nikolic J., Gohl P., Schneider T., Rehder J., Omari S., Achtelik M.W., Siegwart R. The EuRoC Micro Aerial Vehicle Datasets. The International Journal of Robotics Research. 2016. Vol. 35, No. 10. P. 1157–1163.

23. Aghdam H.H., Heravi E.J. Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification. Springer, 2017. 186 p.

24. Sutton R.S., Barto A.G. Reinforcement Learning: An Introduction. 2nd ed. Cambridge: MIT Press, 2018. 552 p.

25. Schmidhuber J. Deep Learning in Neural Networks: An Overview. Neural Networks. 2015. Vol. 61. P. 85–117.

26. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767. 2018. URL: https://arxiv.org/abs/1804.02767

27. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 800 p.

28. Huang G., Liu Z., Van Der Maaten L., Weinberger K.Q. Densely Connected Convolutional Networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2017. P. 2261–2269.

29. Lillicrap T.P., Hunt J.J., Pritzel A., Heess N., Erez T., Tassa Y., Silver D., Wierstra D. Continuous Control with Deep Reinforcement Learning. arXiv preprint arXiv:1509.02971. 2015.

30. Microsoft AirSim. [Електронний ресурс]. – Режим доступу: https://github.com/microsoft/AirSim

31. PX4 Autopilot. [Електронний ресурс]. – Режим доступу: https://px4.io/

32. MAVLink Communication Protocol. [Електронний ресурс]. – Режим доступу: https://mavlink.io/en/

33. OpenAI Gym. [Електронний ресурс]. – Режим доступу: https://www.gymlibrary.dev/

34. ROS (Robot Operating System). [Електронний ресурс]. – Режим доступу: https://www.ros.org/

35. NVIDIA Isaac Sim. [Електронний ресурс]. – Режим доступу: https://developer.nvidia.com/isaac-sim

36. Unreal Engine 5. [Електронний ресурс]. – Режим доступу: https://www.unrealengine.com/

37. Blender 3D. [Електронний ресурс]. – Режим доступу: https://www.blender.org/

38. Gibson Environment for Active Agents. [Електронний ресурс]. – Режим доступу: http://gibsonenv.stanford.edu/

39. Karpathy A. The Unreasonable Effectiveness of Recurrent Neural Networks. [Електронний ресурс]. – Режим доступу: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

40. Bojarski M. et al. End to End Learning for Self-Driving Cars. NVIDIA. arXiv preprint arXiv:1604.07316. 2016.

41. Ткаченко Д.О. Моделювання автономного керування БПЛА на основі машинного навчання. Вісник НТУУ «КПІ». Серія: Автоматика і управління. 2020. №65. С. 88–95.

42. Демченко О.Є., Шевченко Ю.О. Програмне забезпечення для імітаційного моделювання автономного управління БПЛА. Системи управління, навігації та зв’язку. 2021. №2. С. 21–25.

43. Костюк І.В., Павлюк М.І. Системи автоматичного управління квадрокоптерами. Наукові праці ВНТУ. 2021. №1. С. 31–36.

44. Курило О.О., Власенко Д.С. Використання методів глибокого навчання для управління БПЛА в умовах невизначеності. Інформаційні технології і комп’ютерна інженерія. 2022. №1(69). С. 45–51.

45. Чепурнова О. В. Методи штучного інтелекту в системах автономного управління БПЛА / О. В. Чепурнова // Сучасні інформаційні системи. – 2020. – Т. 4, №1. – С. 35–40.

46. Гордієнко В. О. Побудова симуляційного середовища для тестування алгоритмів управління БПЛА / В. О. Гордієнко // Системи озброєння і військова техніка. – 2021. – №1(65). – С. 78–83.

47. Котляр В. М., Лавриненко І. І. Тестування та відлагодження систем управління БПЛА в симуляційному середовищі Gazebo / В. М. Котляр, І. І. Лавриненко // Проблеми програмування. – 2020. – №3–4. – С. 120–127.

48. Хоменко С. О. Інтеграція нейромережевих моделей з ROS при моделюванні автономного керування дронами / С. О. Хоменко // Наукові вісті НТУУ "КПІ". – 2021. – №2. – С. 41–46.