НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»

УДК _____

До захисту допущено:

Завідувач кафедри

Сергій СТІРЕНКО

“__” _____2023р.

Магістерська дисертація

на здобуття ступеня магістра

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

зі спеціальності 123 “Комп`ютерна інженерія”

на тему: «Метод навігації безпілотного літального апарату в умовах агресивного середовища»

Виконав:
студент ІІ курсу, групи ІО-22мп
Коломієць Дмитро Юрійович _____

Науковий керівник:
Проф., д.ф.-м.н., с.н.с.
Гордієнко Юрій Григорович _____

Консультант з нормоконтролю:
Проф., д.т.н, проф.,
Кулаков Юрій Олексійович _____

Рецензент:
Доц., к.н., доц.,
Писаренко А.В. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський)
Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Освітньо-професійна програма *Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Стіренко С.Г.
(підпис) (ініціали, прізвище)

« » _____ 20___ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Коломієць Дмитро Юрійович
(прізвище, ім'я, по батькові)

1. Тема дисертації Метод навігації безпілотного літального апарату в умовах агресивного середовища

Науковий керівник дисертації проф., д.ф.-м.н., с.н.с. Гордієнко Ю. Г.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «***07***» ***11*** ***2023*** р. № 5168

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження: процес оцінки поведінки безпілотного літального апарату в реальному часі з використанням машинного навчання, нейронних мереж, глибоких нейронних мереж та інших методів штучного інтелекту.

4. Предмет дослідження: способи оцінювання безпілотного літального апарату, які повинні бути достатньо ефективними для використання в режимі реального часу в умовах агресивного середовища.

5. Перелік завдань, які потрібно розробити: розробити метод оцінювання безпілотного літального апарату, які повинні бути достатньо ефективними для використання в режимі реального часу в умовах агресивного середовища.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль			
1			
2			
3			
4			
5			
6			

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Перегляд літературних джерел та Збір інформації	2.11.2023–16.11.2023	
2	Огляд існуючих рішень. Пошук правильного рішення	16.11.2023–30.11.2023	
3	Оформлення документації ДП	30.11.2023–25.12.2023	
4	Попередній захист та проходження нормативного контролю	25.12.2023	

Студент Дмитро КОЛОМІЄЦЬ

Науковий керівник дисертації Юрій ГОРДІЄНКО

РЕФЕРАТ

на магістерську дисертацію

Тема магістерської дисертації: «Метод навігації безпілотного літального апарату в умовах агресивного середовища».

Актуальність. В останні десятиліття світ став свідком стрімкого розвитку безпілотних літальних апаратів (БПЛА). Вони широко застосовуються в різних галузях: від аграрного сектору до безпеки і військових дій. Однак, разом з поширенням застосувань БПЛА, зросла і потреба в їх надійності та універсальності, особливо в умовах, які можуть бути ворожими або непередбачуваними.

Мета і завдання дослідження. Метою магістерської роботи є розроблення методу, який є основою навігації безпілотного літального апарату в умовах агресивного середовища.

Завдання:

1. Аналіз агресивного середовища: Зібрати та аналізувати дані про особливості агресивного середовища в обраній області.
2. Розробка алгоритму уникнення загроз: Розробити алгоритм, який дозволяє безпілотному літальному апарату виявляти потенційні загрози та приймати рішення щодо уникнення їх.
3. Вдосконалення системи навігації: Розробити алгоритми для поліпшення точності та надійності системи навігації у складних умовах, включаючи відхилення від траєкторії через агресивні чинники.
4. Адаптація до змін у середовищі: Створити алгоритми для автоматичної адаптації до змін у середовищі.
5. Тестування та симуляція: Розробити віртуальні середовища для тестування алгоритмів навігації в різних агресивних умовах.

Об’єкт дослідження – процес створення методу навігації безпілотного літального апарату в умовах агресивного середовища.

Предмет дослідження – способи та дослідження створення методів навігації в різних агресивних умовах.

Методи досліджень. Аналіз та порівняння інших методів навігації безпілотного літального апарату в умовах агресивного середовища.

Наукова новизна отриманих результатів полягає в тому, що:

- Розроблений новий метод навігації безпілотного літального апарату.
- Порівняння існуючих методів з новим і отримання нових висновків та результатів, що дають змогу оцінити новизну та користь від нових або старих методів.

Особистий внесок здобувача. Магістерське дослідження є особистим внеском в дослідження та відображає глибокий аналіз проблеми, системний підхід до розробки нових методів та їх практичне тестування.

Практична цінність. Дослідження в області методів навігації безпілотних літальних апаратів (БПЛА) в агресивних умовах має значущу практичну цінність, що виражається в наступних аспектах:

- Підвищення безпеки: Завдяки новим методам навігації можливість аварійних ситуацій під час використання БЛА в складних умовах знижується. Це дозволяє запобігти матеріальним збиткам та можливим людським втратам.
- Розширення області застосування: Нові методи дозволять використовувати БЛА в умовах, де раніше їх використання було обмеженим або неможливим, таких як райони з високою густотою перешкод, радіоелектронне перешкоджання чи екстремальні погодні умови.

Ключові слова. ДРОН, БЕЗПІЛОТНІ ЛІТАЛЬНІ ПРИСТРОЇ, НАВІГАЦІЯ, ВІРТУАЛЬНЕ СЕРЕДОВИЩЕ, АГРЕСИВНЕ СЕРЕДОВИЩЕ.

ABSTRACT

Relevance. In the past decade, the world has witnessed the rapid development of unmanned aerial vehicles (UAVs). They are widely used in various sectors, from agriculture to security and military operations. However, as the applications of UAVs expand, so does the need for their reliability and versatility, especially in conditions that might be hostile or unpredictable.

Objective and research tasks. The purpose of the master's thesis is to develop a method that underpins the navigation of an unmanned aerial vehicle in a hostile environment.

Tasks:

1. Analysis of the hostile environment: Collect and analyze data on the features of the hostile environment in the selected area.
2. Development of a threat avoidance algorithm: Create an algorithm that allows the UAV to detect potential threats and make decisions on how to avoid them.
3. Improvement of the navigation system: Develop algorithms to enhance the accuracy and reliability of the navigation system under challenging conditions, including deviations from the trajectory due to aggressive factors.
4. Adaptation to environmental changes: Design algorithms for automatic adaptation to environmental changes.
5. Testing and simulation: Develop virtual environments for testing navigation algorithms under various hostile conditions.

Research object - the process of creating a UAV navigation method in a hostile environment.

Research subject - the ways and studies of creating navigation methods under various aggressive conditions.

Research methods. Analysis and comparison of other UAV navigation methods in a hostile environment.

Scientific novelty of the results consists of:

- A new method of UAV navigation has been developed.

- Comparison of existing methods with the new one, yielding new conclusions and results that allow assessing the novelty and benefits of either new or old methods.

Personal contribution of the researcher. The master's research represents a personal contribution to the study, reflecting a deep analysis of the problem, a systematic approach to developing new methods, and their practical testing.

Practical value. Research in the field of UAV navigation methods under hostile conditions has significant practical value, expressed in the following aspects:

- Enhanced safety: New navigation methods reduce the possibility of accidents during the use of UAVs in challenging conditions, preventing material losses and potential human casualties.
- Expanding the application area: New methods will allow the use of UAVs in conditions where their use was previously limited or impossible, such as areas with a high density of obstacles, electronic interference, or extreme weather conditions.
- Economic benefit: Ensuring the stable operation of UAVs in hostile environments can lead to reduced costs for repairs and replacements and opens up new opportunities for commercial drone use.

Keywords. DRONE, UNMANNED AERIAL VEHICLES, NAVIGATION, VIRTUAL ENVIRONMENT, HOSTILE ENVIRONME

Зміст

Список умовних скорочень

БПЛА	Безпілотний Літальний Апарат.
FPV	First Person View.
GPS	Глобальна система позиціонування.
INS	Inertial Navigation Systems.
ArUco	синтетичним квадратний маркер, що складається з внутрішньої двійкової матриці, яка визначає його ідентифікатор.
SLAM	(Simultaneous Localization and Mapping) – одночасна локалізація і картографування.
Pos	(Position) позиція в просторі.
ROS	(Robot Operating System) – система управління роботом.
UAV	(Unmanned aerial vehicle) – безпілотний літальний апарал
MAC	(Media access control) – унікальний адресний код
STD	(Standart) – в контексті «стандартна помилка»
PID	(Proportional–integral–derivative) – алгоритм функціонування автоматичного регулятора
АУШ	Адаптивне управління шумом

Вступ

В останні десятиліття технологічний прогрес привів до численних інновацій, які перетворили різноманітні сектори світової економіки. Серед найбільш значущих технологічних досягнень слід звернути увагу на безпілотні літальні апарати (БПЛА). Їх швидке поширення та інтеграція у різні галузі вимагає постійного дослідження та вдосконалення їх систем навігації, зокрема в агресивних умовах. Ефективність, надійність та універсальність БПЛА стають ключовими чинниками для їх широкого застосування. У цьому контексті актуальність дослідження методів навігації для безпілотних літальних апаратів у ворожому або непередбачуваному середовищі набуває особливого значення.

Адже не тільки ефективність, але і безпека експлуатації дронів у різних умовах визначає їхнє майбутнє в ряді ключових галузей. З урахуванням зростання складності завдань, які вирішують дрони, а також їхнього поширення в сучасному світі, дослідження та розробка нових методів навігації стають все більш важливими. Ця магістерська робота має на меті не лише аналізувати існуючі методи, а й пропонувати нові підходи та технології, які зможуть підняти стандарти безпеки та ефективності використання БПЛА на новий рівень.

Ця магістерська робота спрямована на розроблення та аналіз таких методів, що відкривають нові можливості для використання дронів в найбільш вимушених умовах. Агресивним середовищем може бути будь-яка зона, де природні або штучні фактори створюють підвищені ризики для функціонування БПЛА. Це може включати в себе райони із сильними метеорологічними умовами, райони радіоелектронного перешкоджання, а також території, де існує ймовірність ворожої дії.

Мета цієї дисертації полягає у вивченні існуючих методів навігації БПЛА та розробці нового методу, який би був оптимальним для використання в агресивному середовищі. Це включає в себе не лише розробку математичних моделей та алгоритмів, але й практичне тестування, нажаль, тільки у віртуальному середовищі.

Відсутність ефективних методів навігації для роботи в агресивних умовах обмежує можливості використання БПЛА в певних сферах діяльності, що актуалізує потребу в даному дослідженні.

Головна ідея полягає в тому, щоб протестувати існуючі методи навігації та розробити власні. Та протестувати їх у віртуальному середовищі завдяки програмному забезпеченню gazebo.

Як заключення, створення нового методу навігації для роботи в агресивних умовах має великий потенціал для покращення галузі в цілому, сприяючи покращенню навігації, швидкості, точності та безпеки використання квадрокоптерів і різних сферах діяльності.

Розділ 1
Огляд контексту проблеми навігації безпілотного літального апарату в агресивному середовищі

Для того щоб зрозуміти актуальність даної теми потрібно розглянути історію, сучасні стани навігаційних систем, потенційні застосування та зрозуміти які задачі має виконувати новостворений метод, також в рамках цього розділу буде наведений коротке пояснення всіх термінів які будуть використовуватися надалі, огляд існуючих методів для вирішення задачі.

1.1 Історичний контекст

Історія безпілотних літальних апаратів налічує понад сто років, протягом яких технології та застосування БПЛА перетворювалися відповідно до потреб та можливостей.

1916-1917 рр. - Перші спроби використання дронів датуються часами Першої світової війни. Британія експериментувала з авіаційними мішенями, що служили для тренування стрільців.

1939-1945 рр. - Під час Другої світової війни різні країни активно використовували безпілотні літальні апарати як літаючі мішені або для розвідування.

1960-1970 рр. - В'єтнамська війна стала переломним моментом для розвитку дронів. США використовували їх для розвідувальних місій, що зменшило втрати пілотів.

1980-1990 рр. - Цей період свідчить про початок комерційного використання БПЛА, особливо в аграрному секторі для моніторингу врожаю.

2001 р. - Після атак 11 вересня в США, активно почали розробляти військові БПЛА. Серед найвідоміших - Predator drone, який був використаний у військових операціях в Афганістані.

2010-2022 рр. - З розвитком технологій, дрони стали все більш доступними для загального споживача. Це призвело до великого розмаїття моделей та застосувань, включаючи фото та відеозйомку, доставку вантажів, аграрний моніторинг тощо.

Рисунок 1.1 – Приклад сучасних БПЛА

2022-2023 рр. – Повномасштабне вторгнення російської федерації в Україну. Нова епоха використання звичайних комерційних так і воєнних літальних апаратів. Модифікація звичайних дронів в дронів-бомбардирів або fpv-дронів.

Загалом, історія розвитку безпілотних літальних апаратів відображає постійний прогрес технологій і розширення сфер їх застосування, від військових до комерційних.

1.2 Сучасний стан навігаційних систем

Сучасні навігаційні системи представляють собою складну інтеграцію різних технологій, що надають користувачам можливість визначення свого місцезнаходження, планування маршрутів та взаємодії з навколишнім світом. Основні характеристики сучасного стану навігаційних систем включають:

GPS (Глобальна система позиціонування): Стало золотим стандартом для зовнішньої навігації завдяки своїй високій точності та доступності. Проте, сигнали GPS можуть бути заблоковані або викривлені в гірських районах, високих будівлях або під час сильних сонячних бурь.

INS (Inertial Navigation Systems): Використовуються для навігації без залежності від зовнішніх сигналів. Вони базуються на механічних або оптичних гіроскопах та акселерометрах, але можуть накопичувати похибки з часом.

Супутникова система навігації (GNSS — Global Navigation Satellite System) — система наземного та космічного обладнання, що призначена для позиціонування в просторі та часі, а також для визначення швидкості, напрямку та інших параметрів руху об’єкта.

Принцип дії супутникових систем навігації заснований на вимірюванні відстані від антени приймача на об'єкті до навігаційних супутників, місцезнаходження яких відоме з великою точністю. Таблиця положень супутників («альманах») є в кожному приймачі супутникового сигналу до початку вимірювань. Зазвичай приймач зберігає альманах у пам'яті з часу останнього ввімкнення. Кожний супутник передає в своєму сигналі весь альманах. Таким чином, знаючи відстані до декількох супутників системи, за допомогою звичайних геометричних побудов на основі альманаху вираховується положення об'єкта в просторі.

Метод вимірювання відстані від супутника до антени приймача заснований на визначенні швидкості поширення радіохвиль. Для реалізації можливості вимірювання часу поширюваного радіосигналу кожен супутник навігаційної системи випромінює сигнали точного часу, використовуючи синхронізований з системним часом атомний годинник. При роботі супутникового приймача його годинник синхронізується з системним часом, і при подальшому прийомі сигналів супутників обчислюється затримка між часом випромінювання, що міститься в самому сигналі, і часом прийому сигналу антеною приймача. Маючи дану інформацію, навігаційний приймач вираховує координати антени. Решта параметрів руху (швидкість, напрямок, пройдена відстань) обчислюється на

основі вимірювання часу, який об'єкт витратив на переміщення між двома або більше точками з координатами, визначеними за попередніми обчисленнями.

Всі вони працюють за схожим принципом: для середнього за точністю позиціонування в просторі антена приймача має отримувати сигнал мінімум від 4 супутників системи (або від 3, якщо 1 з координат відома, наприклад, висота над рівнем моря судна в океані – 0 м), але є певні відмінності. Наприклад, кожна супутникова навігаційна система визначає місцезнаходження в «своїй» системі координат, кожна з систем супутникової навігації належить різним країнам чи угрупуванням країн. Але ці чинники не є важливими для користувачів, набагато важливішими відмінностями є нахил та кількість орбіт, на яких знаходяться супутники, а також їх кількість, період обертання навколо Землі, оскільки саме ці параметри найбільше впливають на точність позиціонування.

ГЛОНАСС (Глобальна Навігаційна Супутникова Система) - російська радіонавігаційна супутникова система, розроблена на замовлення Міністерства оборони СРСР.

Основою системи є 24 активні супутники, що обертаються на орбіті середньою висотою 19 100 км над поверхнею Землі з нахилом 64,8° та періодом близько 11 годин в трьох орбітальних площинах із 8 рівномірно розподіленими супутниками в кожній, а також резервні апарати, призначення яких – в будь-який момент часу замінити супутники, що за певних причин вийшли з ладу. Значення періоду дозволило створити стійку орбітальну систему, що не потребує для своєї підтримки корегувальних імпульсів. Сигнали передаються з направленістю в 38° з використанням правої кругової поляризації, із потужністю 316—500 Вт (EIRP 25-27 dBW). Супутники системи ГЛОНАСС стало передають радіовипромінювання двох типів: навігаційний сигнал діапазону L1 (1,6 ГГц) та навігаційний сигнал високої точності діапазонів L2 и L3 (1,2 ГГц). Наземний комплекс керування ГЛОНАСС складається з центру керування системою та контрольних станцій. Орбіта супутників ГЛОНАСС надає можливість

застосування навігаційної системи на високих широтах (північний і південний полярний регіон), де сигнал GPS приймається погано.

NAVSTAR GPS (Global Positioning System Navigation Satellite Time and Ranging) — високоточна супутникова система навігації , яка дозволяє визначити місцезнаходження об'єкта, його широту, довготу та висоту над рівнем моря, а також напрямок і швидкість його руху. Комплекс NAVSTAR розроблений, утілений і належить Міністерству оборони США.

Станом на сьогодні основою системи є 32 супутники (активні та резервні), що працюють у єдиній мережі й обертаються на шести кругових орбітах, розташованих під кутом 60° одна до одної. На кожній орбіті розміщено по 4 активні супутники, висота орбіт приблизно дорівнює 20 200 км, нахил орбіти - 55°, а період обертання кожного супутника навколо землі дорівнює 12 годинам. Таким чином, із будь-якої точки земної поверхні зазвичай одночасно видно від чотирьох до дванадцяти таких супутників. Кожні 30 секунд супутник передає навігаційні повідомлення, в яких містяться дані про положення супутника в певний момент часу, дані про якість сигналу, похибку супутникового годинника та коефіцієнти моделі іоносфери. Передача сигналу з супутника відбувається на частоті 1575,42 МГц.

Таблиця 1

Точність позиціонування провідних навігаційних систем

СНР (супутникова навігаційна система)	GPS (США)	ГЛОНАС (росія)	GALILEO (Європа)	BAIDOU (КНР)	QZSS (Японія)	IRNSS (Індія)
Точність, м	3.6 метри	7 метрів	До 1 метра	10 метрів	До 1 метра	До 10 метрів

Wi-Fi та Bluetooth навігація: В основному використовуються для внутрішньої навігації, наприклад, в торгових центрах або аеропортах. Вони визначають місцезнаходження на основі відстані до відомих точок доступу.

Multi-modal навігація: Комбінація декількох технологій для підвищення точності та надійності, особливо в агресивних середовищах. Наприклад, може поєднувати GPS з INS та іншими датчиками для підтримки навігації, коли один із сигналів стає недоступним.

FANET (англ. Flying Ad Hoc Network). Це Схоже на мобільні однорангові мережі MANET (Mobile Ad Hoc Network) та автомобільні однорангові мережі VANET (Vehicular Ad Hoc Network), існує специфічний тип мережі, яка тимчасово самостійно організовується на базі безпілотних літальних апаратів (БПЛА).

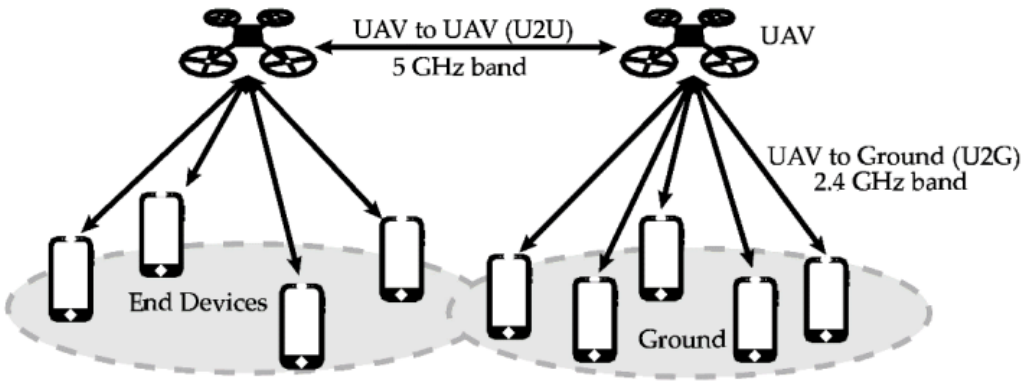


Рисунок 1.2 – Схема роботи FANET

Ці мережі відкривають широкі можливості для цивільних та комерційних потреб. Утворення такого зв'язку потрібне не лише для задач спостереження та контролю, але й, наприклад, для оптимізованої координації руху транспортних засобів, збільшення безпеки (як засіб уникнення зіткнень тощо). Рух в 3D–просторі та постійно змінювана топологія створюють численні виклики на фізичному рівні (англ. Physical layer, PHY), рівні контролю доступу до середовища (англ. Media access control, MAC) та мережевому рівні.

З розвитком технологій та збільшенням потреб користувачів, навігаційні системи продовжують розвиватися та адаптуватися, щоб краще відповідати сучасним викликам.

Виклики навігації в агресивних середовищах

Агресивне середовище може в себе включати багато різних природних факторів

- Змінні погодні умови. Швидкі зміни клімату, такі як пориви вітру, туман, снігопади або бурі.
- Температурні відхилення. Екстремальні температури можуть впливати на роботу електронних компонентів БЛА, скорочуючи термін їх служби або змінюючи характеристики датчиків.
- Проблеми з комунікацією. В агресивних середовищах може відбуватися значна втрата сигналу через густі водні маси, гірські перевали або електромагнітні перешкоди.

Спроби вирішити ці проблеми включають в себе розробку нових технологій, таких як більш ефективні батареї, вдосконалені датчики, системи навігації, які можуть працювати без GPS, та більш надійні методи комунікації для роботи в агресивних умовах.

1.3 Потенційні застосування

Безпілотні літальні апарати (БЛА) стають все більш популярними в різних галузях завдяки їхній гнучкості, маневреності та можливості працювати в надзвичайних умовах. Ось деякі нові та емерджентні напрямки застосування:

1. Моніторинг та реагування на надзвичайні ситуації. Дрони можуть швидко вивчати райони, які стали жертвами природних катастроф, визначати потреби у допомозі або допомагати в координації рятувальних операцій.

2. Наукові дослідження. БЛА можуть використовуватися для збору даних у важкодоступних місцях, таких як полярні регіони, джунглі або підводні глибини.
3. Військове використання. Дрони можуть виконувати завдання розвідки, спостереження або навіть нападу в агресивних умовах, де ризик для людського життя є надзвичайно високим.
4. Сільське господарство. За допомогою дронів можна аналізувати стан посівів, контролювати шкідників або оптимізувати полив.
5. Геологічні та археологічні дослідження. Дрони можуть виявляти нові місцезнаходження мінералів, археологічні розкопки або допомагати в картографуванні невідомих територій.
6. Постачання в агресивних умовах. БЛА можуть доставляти товари або медичне обладнання в райони конфліктів, зони стихійних лих або в інші місця, де традиційні методи доставки є неможливими.
7. Екологічний моніторинг. Використання дронів для відстеження змін довкілля, таких як втрата лісів, забруднення водних об'єктів або вивчення міграції тварин.

З розвитком технології, можливості застосування БЛА продовжують розширюватися, відкриваючи нові горизонти для досліджень, операцій та комерційних можливостей.

1.4 Технології візуального визначення

1.4.1. Глибинні моделі відображення

Використання глибинних нейронних мереж для створення тривимірних карт оточуючого простору. Це дозволяє дронам краще розпізнавати та уникати перешкод, а також взаємодіяти з тризначним середовищем більш точно. Глибинні моделі відображення є важливим компонентом технологій візуального визначення, які використовуються в безпілотних літальних апаратах (дронах). Ці моделі використовують глибинні нейронні мережі для аналізу

візуальних даних та створення тривимірних карт оточуючого простору.

1.4.2 Сегментація зображень

Сегментація зображень - це процес визначення та виділення окремих областей або об'єктів на зображенні. У відношенні до безпілотних літальних апаратів (дронів), техніка сегментації зображень є важливим елементом візуального визначення та навігації. Технології сегментації дозволяють виділяти окремі об'єкти або області на зображеннях. Це корисно для визначення меж об'єктів та забезпечення точної ідентифікації перешкод.

1.4.3 Трекінг об'єктів

Алгоритми трекінгу дозволяють дронам відстежувати рух об'єктів в реальному часі. Це важливо для уникнення зіткнень та взаємодії з рухомими об'єктами, такими як транспортні засоби або інші дрони. Трекінг об'єктів є ключовою технікою візуального визначення, особливо у випадку безпілотних літальних апаратів (дронів). Ця технологія дозволяє системам автономної навігації визначати і відстежувати рух об'єктів в реальному часі

1.4.4. Візуальне визначення орієнтації

Візуальне визначення орієнтації — це процес визначення орієнтації об'єкта або системи в просторі на основі візуальної інформації, отриманої від камер чи інших сенсорів. Для безпілотних літальних апаратів (дронів) визначення орієнтації має ключове значення для точної навігації та управління. Технології, що дозволяють визначити орієнтацію дрона у просторі на основі візуальних даних. Це допомагає забезпечити стабільну навігацію, особливо в умовах, коли GPS-сигнали можуть бути обмеженими.

1.4.5. Забезпечення роботи в умовах обмеженої видимості

Забезпечення роботи безпілотних літальних апаратів (дронів) в умовах обмеженої видимості є важливим завданням, оскільки це може включати такі ситуації, як погана погода, туман, або низька освітленість. Розробка технологій, які дозволяють дронам пристосовуватися до обмеженої видимості, такої як погана погода або туман. Інфрачервоні та лідар сенсори можуть бути використані для покращення ефективності в умовах обмеженої видимості.

1.4.6. Врахування особливостей терену

Врахування особливостей терену є ключовим аспектом для ефективної навігації та безпечної роботи безпілотних літальних апаратів (дронів). Технології, що враховують рельєф, тип покриття, нахил та інші особливості терену, дозволяють дронам адаптуватися до різноманітних умов та забезпечувати ефективну навігацію. Системи візуального визначення повинні бути здатні адаптуватися до різних типів теренів, включаючи міські ландшафти, гірські регіони або водні поверхні.

Ці технології спільно використовуються для створення повноцінного візуального середовища, яке допомагає дронам в навігації та взаємодіяти з оточенням у реальному часі.

1.5 Сучасні тенденції та перспективи розвитку

З ростом потреб в галузі навігації безпілотників в агресивних умовах, дослідники та інженери шукають нові підходи та технології для вдосконалення цієї системи. Штучний інтелект, машинне навчання та розширена реальність - це лише кілька напрямків, що можуть принести революційні зміни в області навігації безпілотників.

Одна з важливих подій відбулася не так давно. Набрало чинності положення про обов'язкове присвоєння цивільним дронам ідентифікатора Remote ID для їхнього впізнання. Усі виробники повинні були запровадити цю технологію у випущену продукцію після 30 травня 2023 року.

Щодо сучасних тенденцій можна виділити чотири сфери застосування дронів, які будуть розвиватися з величезними темпами:

- **сільське господарство**. Ми вже згадували про агросектор, однак довгострокова перспектива ґрунтується на прогнозі, що до 2050 року населення Землі зросте до 9,7 млрд осіб. Отже сільгосппродукції знадобитися багато і без дронів не обійтися;
- **сфера страхування**. Згідно з оцінками світової колегії зі страхування з 1970 року кількість виплат зі страхування від стихійних лих зросла у 8 разів. Страхові компанії починають активно використовувати дрони, щоб отримувати фото і відео з місць стихійних лих, для більш точного аналізу ситуації;
- **будівництво та гірничо видобувна промисловість**. У цих, вельми різних на перший погляд галузях, дрони виконують одну й ту саму функцію - дають змогу контролювати дотримання умов безпеки праці в особливо небезпечних місцях;
- **військова галузь**. Тут зростання спостерігатиметься щонайменше наступне десятиліття. Головні напрямки розвитку це здешевлення і збільшення дальності польоту дронів. Також активно ведеться робота над впровадженням ШІ та створенням "свідомості рою" для великих груп FPV дронів, які можуть самостійно групуватися або розділятися для ураження цілей залежно від обставин. Активний розвиток очікується в технологіях точної навігації, переважно в частині розвитку датчиків типу Lidar.

Висновок до розділу 1

Аналізуючи сучасний контекст тем розділу, стає очевидною критична потреба в удосконаленні методів керування дронів в агресивних умовах. Можна побачити, що агресивні середовища можуть включати різноманітні фактори, від погодних умов до технічних викликів. Для розв'язання цієї проблеми було сформульовано ключову мету нашої роботи: "Підвищення якості навігації безпілотних літальних апаратів у складних умовах дії". Це передбачає інтеграцію новітніх досліджень, технологій та методів. При цьому враховується потенційний вплив історичних, технічних та прикладних аспектів.

У наступних розділах ми звернемося до докладного розгляду цих методів та технологій, а також розглянемо можливі сценарії їхнього застосування.

Розділ 2

Огляд існуючих методів для вирішення задачі навігації безпілотного літального апарату в агресивному середовищі

Навігація безпілотного літального апарату (БПЛА) в агресивному середовищі є дуже складною задачею, яка вимагає розробки ефективних методів та алгоритмів. У цьому розділі розглядаються деякі існуючі методи та підходи до вирішення цієї задачі.

1.1 Методи на основі SLAM.

SLAM – це техніка, яка дозволяє роботам або безпілотним літальним апаратам одночасно визначати своє місцезнаходження та створювати карту навколишнього середовища в реальному часі. Цей процес допомагає роботам орієнтуватися в невідомих або динамічно-змінюваних середовищах. В цілому, принцип роботи SLAM виглядає наступним чином. В будь-який момент часу робот повинен знати своє місцезнаходження та поступово сканувати навколишній простір за допомогою сенсорів, створюючи, отже, карту території. Карта формується поступово, по мірі дослідження роботом нових областей. Основним джерелом інформації про місцезнаходження робота є одометрія, отримана різними способами (колеса, комп'ютерний зір, IMU або їх комбінація). Однак, при будівництві карти робот починає порівнювати своє положення з картою. Наприклад, якщо робот проїжджає ту частину приміщення, яку він вже сканував, відбувається порівняння за певними паттернами. В результаті, якщо апарат розуміє, що поточні вимірювання одометрії не відповідають вимірюванням карти, відбувається корекція одометрії.

В свою чергу SLAM поділяється на **Візуальний** та **LiDAR**

2.1.1. **Візуальний SLAM** (або vSLAM) використовує зображення, отримані з візуальних датчиків. У цьому випадку робот створює карту оточення та визначає своє місцезнаходження на основі даних з камер, зазвичай відстежуючи орієнтири для тріангуляції положення камери в просторі. У останні роки візуальний SLAM може бути реалізований з мінімальними витратами

завдяки відносно недорогим камерам. Порівняно з іншими датчиками, камери швидко надають великий обсяг інформації, і з їх допомогою легко виявити відомі орієнтири. Незважаючи на свій потенціал, vSLAM зараз не може вирішувати всі завдання через обмеження щодо умов освітлення та швидкості руху пристрою. Крім того, залишається відкритим питання про якість vSLAM у присутності хмар, туману, димки та інших кліматичних явищ.

2.1.2. **Пристрої LiDAR** розраховують відстані до об'єктів, відправляючи імпульси світла і вимірюючи час, який потрібен для їх відображення. Отримане хмара точок забезпечує високоточні вимірювання відстані та є ефективним для побудови карт за допомогою SLAM. Хмари точок не такі детальні за щільністю, як зображення, і не завжди забезпечують достатню кількість характеристик для порівняння орієнтирів. Наприклад, у місцях, де мало перешкод, важко порівняти хмари точок, що може призвести до втрати даних про місцезнаходження дрону. Крім того, LiDAR може не класифікувати орієнтири, які мають занадто великі або занадто маленькі розміри, а деякі орієнтири можуть бути апроксимовані послідовностями точок.

Недоліки даного методу:

- Обмежена точність: Точність SLAM може бути обмеженою у випадку слабких сигналів або відсутності чітких орієнтирів у середовищі.
- Висока обчислювальна складність: SLAM вимагає значних обчислювальних ресурсів, що може бути проблематичним для БЛА з обмеженим обчислювальним потужностям.

1.2 Методи машинного навчання.

Методи машинного навчання включають в себе використання алгоритмів для виявлення закономірностей в даних та використання цієї інформації для прийняття рішень або прогнозування. У контексті навігації БЛА, машинне навчання може бути використане для виявлення та уникнення перешкод, планування маршрутів, та іншого.

У найзагальнішому випадку розрізняють два типу машинного навчання: навчання по прецедентах, або індуктивне навчання, і дедуктивне навчання. Оскільки останнє прийнято відносити до області експертних систем, то терміни «машинне навчання» і «навчання по прецедентах» можна вважати синонімами. Цей метод навчання зараз, як прийнято говорити, в тренді, а ось експертні системи переживають кризу. Бази знань, що лежать в їх основі, важко узгоджувати з реляційною моделлю даних, тому промислові СУБД неможливо ефективно використовувати для наповнення баз знань експертних систем.

Навчання по прецедентах, в свою чергу, поділяють на три основних типи: контрольоване навчання, або навчання з учителем (supervised learning), неконтрольоване навчання (unsupervised learning), або навчання без учителя, і навчання з підкріпленням (reinforcement learning).

Крім названих, розробляються і інші методи навчання: активне, багатозадачне, різноманітне, трансферне і т.д. Особливо успішно розвивається в останні роки «глибоке навчання», при використанні якого можуть успішно поєднуватися алгоритми навчання з вчителем і без вчителя.

Контрольоване навчання

Цей метод навчання застосовується у випадках, коли є великі обсяги даних, припустимо -— тисячі фотографій домашніх тварин з маркерами (мітками, ярликами): це кішка, а це собака. Необхідно створити алгоритм, за допомогою якого машина могла б по фотографії, яку «не бачила» раніше, визначити, хто на ній зображений: кішка або собака. У ролі «вчителя» в даному випадку виступає людина, яка заздалегідь проставила маркери. Машина сама вибирає ознаки, за якими вона відрізняє кішок від собак. Тому в подальшому знайдений нею алгоритм може бути швидко переналаштований на рішення іншої задачі, наприклад, на розпізнавання курей і качок. Машина знову-таки сама виконає складну і копітку роботу по виділенню ознак, за якими буде розрізняти цих птахів. А нейромережа, яку навчили розпізнавати кішок, можна швидко навчити обробляти результати комп'ютерної томографії.

Неконтрольоване навчання

Хоча маркованих, розмічених даних накопичилося вже досить багато, даних без маркерів (міток) все ж набагато більше. Це зображення без підписів, аудіозаписи без коментарів, тексти без анотацій. Завдання машини при неконтрольованому навчанні — знайти зв'язку між окремими даними, виявити закономірності, підібрати шаблони, упорядкувати дані або описати їх структуру, виконати класифікацію даних. Неконтрольоване навчання використовується, наприклад, в рекомендаційних системах, коли в інтернет-магазині на основі аналізу попередніх покупок покупцеві пропонуються товари, які можуть зацікавити його з більшою ймовірністю, ніж інші. Або коли на після перегляду якогось відеокліпу на порталі YouTube відвідувачеві пропонують десятки посилань на ролики, чимось схожі на переглянутий. Або коли Google у відповідь на один і той же запит ранжує посилання в результатах пошуку для одного користувача інакше, ніж для іншого, оскільки враховує історію пошуків.

Навчання з підкріпленням

Таке навчання є окремим випадком контрольованого навчання, але вчителем в даному випадку є «середовище». Машина (її в цій ситуації часто називають «агент») не має попередньої інформацією про середовище, але має можливість здійснювати в ній будь-які дії. Середа реагує на ці дії і тим самим надає агенту дані, які дозволяють йому реагувати на них і вчитися. Фактично агент і середовище утворюють систему зі зворотним зв'язком.

Навчання з підкріпленням використовується для вирішення більш складних завдань, ніж навчання з учителем і без вчителя. Воно використовується, наприклад, в системах навігації для роботів, які навчаються уникати зіткнень з перешкодами шляхом набуття досвіду, отримуючи зворотний зв'язок при кожному зіткненні. Навчання з підкріпленням використовується також в логістиці, при складанні графіків і плануванні завдань, при навчанні машини логічним іграм (покер, нарди, го і ін.).

Нейронні мережі і глибоке навчання

Для машинного навчання використовують різні технології та алгоритми. Зокрема, можуть застосовуватися дискримінантний аналіз, байєсовські класифікатори та багато інших математичних методів. Але в кінці XX століття все більше уваги почали приділяти штучним нейронним мережам (ANN). Черговий вибух інтересу до них почався в 1986 році, після істотного розвитку т.зв. «Методу зворотного поширення помилки», який з успіхом застосували при навчанні нейронної мережі.

ANN є системою з'єднаних і взаємодіючих між собою штучних нейронів, виконаних на основі порівняно простих процесорів. Кожен процесор ANN періодично отримує сигнали від одних процесорів (або від сенсорів, або від інших джерел сигналів) і періодично посилає сигнали іншим процесорам. Всі разом ці прості процесори, з'єднані в мережу, здатні вирішувати досить складні завдання.

Найчастіше нейрони розташовуються в мережі за рівнями (їх ще називають шарами). Нейрони першого рівня — це, як правило, вхідні. Вони отримують дані ззовні (наприклад, від сенсорів системи розпізнавання осіб) і після їх обробки передають імпульси через синапси нейронів на наступному рівні. Нейрони на другому рівні (його називають прихованим, оскільки він безпосередньо не пов'язаний ні з входом, ні з виходом ANN) обробляють отримані імпульси і передають їх нейронам на вихідному рівні. Оскільки мова йде про імітацію нейронів, то кожен процесор вхідного рівня пов'язаний з декількома процесорами прихованого рівня, кожен з яких, в свою чергу, пов'язаний з декількома процесорами рівня вихідного. Така архітектура найпростішої ANN, яка здатна до навчання і може знаходити прості взаємозв'язку в даних.

Глибоке (глибинне) навчання може бути застосоване лише по відношенню до більш складних ANN, що містить кілька прихованих рівнів. При цьому рівні нейронів можуть чергуватися з шарами, які виконують складні логічні перетворення. Кожен наступний рівень мережі шукає взаємозв'язки в попередньому. Така ANN здатна знаходити не тільки прості взаємозв'язки, а й

взаємозв'язки між взаємозв'язками. Саме завдяки переходу на нейромережу з глибинним навчанням компанії Google вдалося різко підвищити якість роботи свого популярного продукту «Перекладач». Зокрема, якість перекладу між англійською та французькою мовами підвищився відразу на 7 балів, тобто більш ніж на 20%. Попередня система, яка виконувала фразовий статистичний машинний переклад, домоглася подібного поліпшення за весь час свого існування (з 2006 року).

Недоліки даного методу:

- Потреба у великих наборах даних: Ефективність моделей глибокого навчання залежить від наявності великих та різноманітних наборів даних для тренування.
- Час на тренування: Тренування моделей може бути затратним по часу процесом.

1.3 Традиційні методи навігації.

Ці методи базуються на використанні стандартних інструментів та технологій, таких як GPS, гіроскопи, акселерометри тощо. Ці системи можуть бути менш ефективними в агресивних умовах через втрату сигналу GPS або електромагнітні завади.

Недоліки даного методу:

- Залежність від GPS: Багато традиційних систем навігації засновані на GPS. Проте у забудованих зонах або під час поганих погодних умов сигнал GPS може бути слабким або відсутнім, що призводить до збоїв у навігації.
- Електромагнітні завади: Наявність електромагнітних завад може впливати на роботу навігаційних датчиків, таких як компаси, гіроскопи та акселерометри.

- Обмежена автономія: Залежно від системи живлення, деякі БПЛА можуть мати обмежений час польоту, що обмежує їхню автономію та використання в довготривалих місіях.

1.4 Гібридні системи.

Гібридні системи поєднують елементи реактивних та деліберативних методів, а також можуть інтегрувати інші технології, такі як SLAM або машинне навчання, для створення більш гнучких та надійних систем навігації.

Недоліки даного методу:

- Складність інтеграції: Інтеграція різних методів може бути складною та дуже затратною по часу.
- Потенційні конфлікти: Можливі конфлікти між різними системами та алгоритмами, що можуть призвести до нестабільності системи навігації.

1.5 Методи оптимізації траєкторії.

Оптимізація траєкторії включає в себе використання математичних технік для знаходження оптимального маршруту через середовище, мінімізуючи витрати (наприклад, час, енергію) або ризики.

Недоліки даного методу:

- Обчислювальна складність: Оптимізація траєкторії може вимагати значних обчислювальних ресурсів, особливо у випадку складних моделей та середовищ.

1.6 Системи відстеження та виявлення перешкод.

Системи відстеження та виявлення перешкод використовують сенсори та алгоритми для ідентифікації та відстеження перешкод у середовищі. Це важливо для безпечної навігації, особливо в агресивних або динамічно змінюваних середовищах.

Недоліки даного методу:

- Відсутність повної інформації: Системи відстеження можуть мати обмежену інформацію про динамічні об'єкти та інші потенційні загрози у середовищі.
- Затримки: Можливі затримки в обробці даних та відгуку на зміни у середовищі, що може бути критичним у агресивних умовах.

Висновок до розділу 2

Можна побачити, що представлені методи мають певні обмеження, такі як обмежена точність, висока обчислювальна складність, потреба у великих наборах даних, час на тренування, обмежена передбачуваність, відсутність довгострокового планування, складність інтеграції, потенційні конфлікти, обчислювальна складність оптимізації траєкторії, відсутність повної інформації та затримки в системах відстеження та виявлення перешкод. Ці обмеження можуть впливати на ефективність та надійність навігаційних систем безпілотних літальних апаратів в агресивних середовищах.

Також існують певні недоліки, асоційовані з кожним з цих методів, які можуть включати в себе проблеми з точністю визначення місцезнаходження, затримки в реакції на зміни у середовищі, відсутність гнучкості для адаптації до нових умов, та інше.

Ці обмеження та недоліки вказують на необхідність подальшого дослідження та розробки в цій області. Зокрема, може бути корисним розглянути нові підходи та технології, які можуть допомогти подолати ці виклики та покращити надійність та ефективність навігаційних систем БПЛА в агресивних середовищах.

Деякі з цих методів, як метод машинного навчання є більш виправданим. В зв’язку із великими темпами розвитку штучного інтелекту, це є найбільш потенціально успішним методом з усіх наявних.

Розділ 3
Вибір і побудова оточення і методів

Тестування алгоритмів навігації важливо проводити в контрольованих умовах перед їх реальним застосуванням. Вибір оточення дозволяє відтворити різні сценарії та умови, що можуть зустрічатися БПЛА під час реальних польотів

1.1 Вибір методу. Автономний політ

Для забезпечення повної автономності таких проектів важливо розробити ефективне та надійне рішення для навігації та посадки, особливо в тих місцях, де відбувається підзарядка чи заміна енергетичних блоків. Крім того, дуже важливо розробити методики для безперебійного переходу від польотів на відкритому повітрі до польотів у закритих просторах, незалежно від погодних умов, таких як сильний вітер. Оскільки БПЛА зазвичай має приземлятися на спеціалізованих станціях підзарядки або заміни акумуляторів, така посадка повинна відзначатися високим рівнем точності та контролю.

В даному розділі розглядається впровадження автономного польоту БПЛА. В якості контролера польоту використовується автопілот PX4, який надає багато можливостей для автономного польоту, про що йде мова у розділі 3.1.3. Поряд з програмним забезпеченням PX4, буде використовуватися ROS, який дозволяє виконувати обчислення на борту системи та передавати повідомлення до PX4 за допомогою пакету MAVROS. ROS розглядається у Розділі 3.1.2. Для зручного налаштування та доступу до БПЛА використовується QGroundControl, який розглядається у Розділі 3.1.1. Основна причина вибору PX4, ROS та QGroundControl полягає в їх відкритому коді та легкому налаштуванню.

Тобто, я обрав саме цей метод через наявність відкритого коду у PX4, ROS та QGroundControl, що надає гнучкість та можливість адаптації під конкретні задачі.

3.1.1. QGroundControl

QGroundControl - це безкоштовне та відкрите програмне забезпечення для керування безпілотними літальними апаратами (БПЛА) і роботами. Ця платформа забезпечує повний набір функцій для планування маршруту, налаштування параметрів, моніторингу польоту та аналізу даних логів.

Основне використання цього програмного забезпечення буде полягати в зміні параметрів БПЛА в польоті, наприклад, вертикальної і горизонтальної швидкості, зміні параметрів ПІД для керування положенням тощо. Крім того, калібрування датчиків компаса, гіроскопа.та акселерометра можна легко виконати перед кожним польотом для досягнення найкращих характеристик БПЛА.

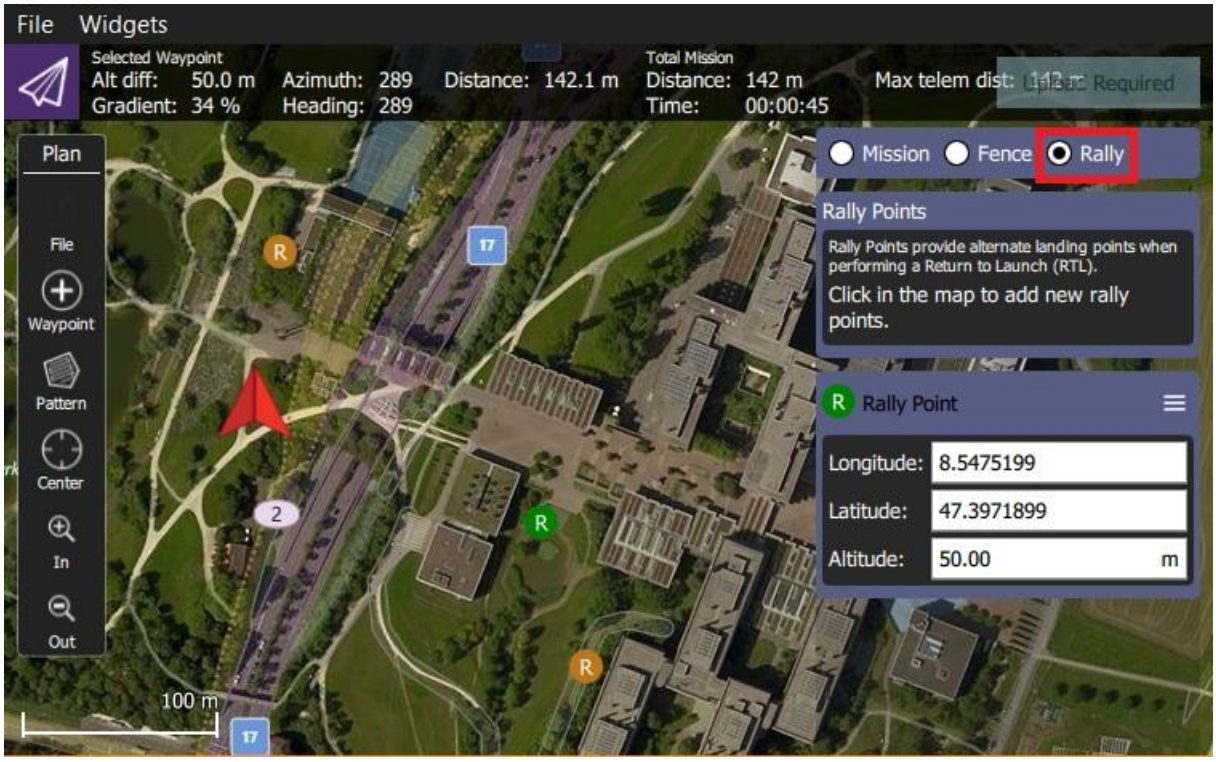


Рисунок 3 - Ілюстрація графічного інтерфейсу в QGroundControl.

3.1.2. Бортовий контроль

Для реалізації бортового управління, а також зв'язку між блоками буде використовуватися ROS.

Багато компаній використовують ROS (Robot Operating System) для розробки своїх роботів. Тепер це доволі відома у світі система, і над версією ROS 2 співпрацює низка провідних організацій, а в Technical Steering Committee (TSC) входять такі відомі компанії, як Amazon, Intel, Microsoft, Toyota.

Вона включає в себе ряд бібліотек та інструментів з метою спрощення створення нових робо-технічних рішень. Іншою важливою особливістю є обробка повідомлень між вузлами в архітектурі ROS, яка візуалізовано на Рисунок 4. Тут створено три створено три вузли ROS, але кількість вузлів може бути набагато більшою відповідно до конкретного застосування

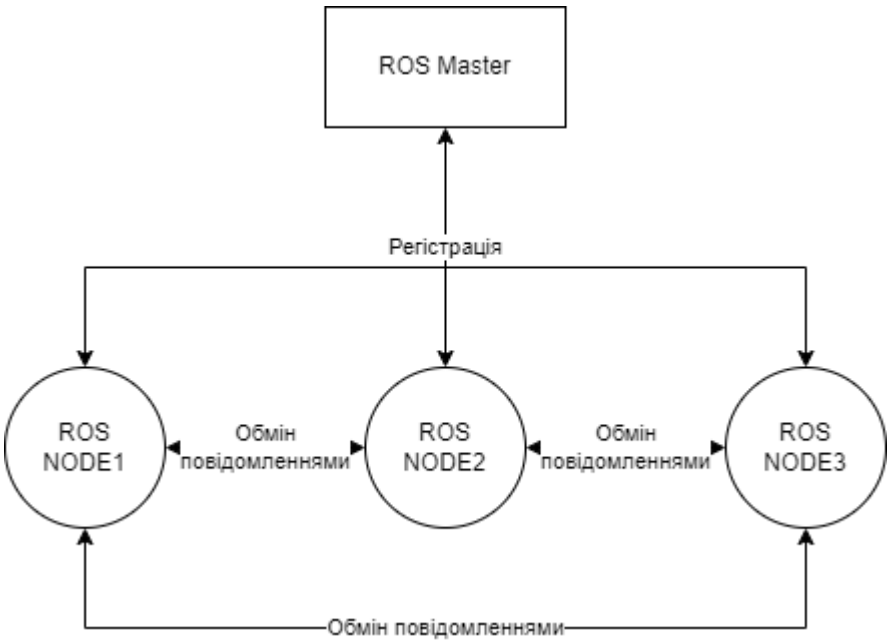


Рисунок 4 - Ілюстрація високорівневого погляду на архітектуру зв'язку між вузлами в ROS.

В ROS майстер відповідає за моніторинг усіх вузлів у системі, із якими кожен вузол спілкується через реєстрацію у майстра. Згодом, будь-який вузол може розсилати чи отримувати повідомлення за посередництвом майстра. У контексті ROS такий механізм спілкування називається "темами". Ця

особливість ROS буде застосована в бортових системах управління, де вимагається взаємодія між декількома вузлами.

Отже, ROS забезпечує апаратну абстракцію, драйвери пристроїв, бібліотеки, візуалізатори, передачу повідомлень, керування пакетами й багато іншого. Оскільки роботи, яких будували десятиліття тому, були вкрай ненадійними, систему проєктували так, щоб відмова одного з вузлів не зупинила роботу всього робота. Тому ядро системи побудоване навколо обміну повідомлень між вузлами, які нічого не знають один про одного. Все, що відомо — це назва каналу обміну повідомленнями, його формат і тип виклику.

Є три типи виклику:

1. Звичайний відправник і отримувач (publisher – subscriber). Відправник надсилає своє повідомлення отримувачеві й не чекає на відповідь, а просто й далі виконує своє завдання. Наприклад, камера може публікувати фото з частотою 30 Гц.

2. Виклик сервісу (service). У цьому випадку відправник зацікавлений в отриманні відповіді на свій запит і очікує відповіді. Наприклад, сервіс із пошуку найкоротшого шляху на мапі з точки A в точку B.

3. Сервіси зі зворотним зв’язком (action server), що працюють як другий тип з можливістю отримання зворотного зв’язку про стан виконання завдання. Наприклад, сервіс завантаження файлу в хмарне сховище може повертати відсоток пересланих даних щосекунди

Для реалізації автономного польоту було створено ряд вузлів ROS, які будуть розглянуті в розділі 3.3.

1.2 Короткий опис наявних оточень

Існує великий вибір між різними оточеннями для використання вище зазначеного методу в пункті 3.1.

Для тестування нашого методу з маркерними дошками ArUco не вистачить 2D симулятора. Тому сконцентруємося на повномаштабних 3D симуляторах.

3.2.1. MORSE

MORSE (Modular OpenRobots Simulation Engine) є робототехнічним середовищем симуляції, призначеним для симуляції роботів і датчиків в тривимірному світі. Він базується на ігровому рушії Blender, він підтримує ROS, YARP та сировий сокет.

Він є доволі простим у використанні, що дозволяє користувачам легко розробляти тривимірні моделі роботів і сцен для їх симуляції.

Ось деякі особливості MORSE:

- Модульність: MORSE дозволяє користувачам додавати моделі роботів та датчиків з гнучкістю.
- Багатоплатформеність: Він працює на багатьох операційних системах, включаючи Linux, macOS та Windows.
- Інтеграція з ROS та іншими робототехнічними фреймворками: MORSE може інтегруватися з популярними робототехнічними платформами, такими як ROS (Robot Operating System).
- Реалістична графіка: Завдяки ігровому рушію Blender MORSE може надавати реалістичну візуалізацію та фізику.

Головні недоліки це висока залежність від ігрового рушія Blender, очевидно, тому що він на ньому базується. І наступна, це певна направленість, в основному для академічних досліджень, не практичних або промислових.

Тобто, дане оточення **не підходить** для відтворення наших методів.

3.2.2. Gazebo

Gazebo є відкритим програмним забезпеченням для симуляції роботів, динамічних систем і зовнішнього середовища в 3D-просторі, який інтегрує фізичний рушій відкритої динаміки (ODE), рендеринг OpenGL та підтримує код як для керування приводами, так і для симуляції датчиків.

Ось деякі ключові особливості Gazebo:

- Реалістична Симуляція: Використання передових методів для симуляції фізики, світла, і атмосферних умов.
- Інтеграція з ROS: Gazebo інтенсивно використовується в спільноті ROS (Robot Operating System) для симуляції роботів перед реальним розгортанням.
- Модульність: Користувачі можуть створювати та додавати свої моделі датчиків, діячів та роботів.
- Велика бібліотека: Забезпечує бібліотеку реалістичних моделей роботів, датчиків та середовищ.
- Багатоплатформенність: Gazebo працює на багатьох операційних системах, таких як Linux, macOS і Windows.

Головна проблема цього оточення полягає в тому, що Gazebo є ресурсо-затратним, особливо при великих симуляціях.

Тобто, дане оточення **підходить** для відтворення наших методів.

3.2.3. V-REP

V-REP (або Virtual Robot Experimentation Platform) — це інтегрована платформа для моделювання та симуляції роботів та інших динамічних систем. Це програмне забезпечення надає розширений набір інструментів для моделювання, програмування та симуляції роботів в різних середовищах.

Ось деякі ключові особливості V-REP:

- Сценарії та Скрипти: Користувачі можуть створювати та модифікувати сценарії за допомогою вбудованого скриптового мови Lua.
- Багатоплатформенність: V-REP доступний для Linux, macOS і Windows.
- Інтеграція з Різними Технологіями: Підтримка різних протоколів комунікації, включаючи ROS, remote API, додатки зовнішніх модулів тощо.

Головний недолік окрім великого навантаження на систему, це ліцензування. Тільки комерційна версія має весь масштаб функцій, які може запропонувати нам Gazebo, тому дане оточення **не підходить** для виконання методів.

1.3 Опис обраного оточення

3.3.1. ROS Пакети

Таблиця 2

Завантажені плагіни для використання даних IMU та барометра в середовищі моделювання Gazebo

№	Таблиця встановлених пакетів для середовища Gazebo для конфігурації ROS	
	Назва конфігурації	Тип пакету
1	ROS пакет	Hector localization-catkin
2	ROS пакет	Hector quadrotor-kinetic-devel

Щоб надати огляд ROS-пакетів та використаних у них python-скриптів, було сформовано таблиця 3, таблиця 4, таблиця 5, що містить невеликий опис призначення кожного python-скрипту.

Таблиця 3

Список скриптів на python, використаних для виявлення маркерів пакунків ROS catkin зі створеного робочого простору ROS

Пакет ROS: Виявлення маркерів	
marker_detection.py	Відповідає за виявлення та оцінку положення ArUco маркерні дошки
marker_detection_ros_interface.py	Інтерфейс між скриптом виявлення маркерів та ROS
log_data.py	Відповідає за ведення журналу даних про оцінене положення дошки маркера ArUco.

Таблиця 4

Перелік python-скриптів, використаних у ROS catkin пакеті "sensor fusion", створеному в робочому просторі ROS

Пакет ROS: Злиття сенсорів	
sensor_fusion.py	Відповідає за злиття датчиків пози ArUco, IMU та даних барометра
sensor_fusion_ros_interface.py	Інтерфейс між скриптом злиття датчиків та ROS
ukf.py	Реалізація фільтра Unscented Kalman (UKF).
log_data.py	Відповідає за реєстрацію даних злиття датчиків

Таблиця 5

Список python-скриптів, що використовуються у пакеті ROS catkin для керування поза бортом зі створеного робочого простору ROS

Пакет ROS: Бортовий контроль	
autonomous_flight.py	Відповідає за виконання місій та автономні польоти
drone_control.py	Відповідальний за зміну стану на борту за командами з клавіатури та передачу заданих точок і локальної позиції до PX4.
ground_control.py	Відповідає за прослуховування команд з клавіатури, які передаються програмі з ноутбука для керування БПЛА.
loiter_pilot.py	Відповідає за керування БПЛА за заданими командами з клавіатури waypoint_checking.py Відповідає за перевірку того, чи досягнув БПЛА місця призначення
waypoint_checking.py	Відповідає за перевірку, чи досяг БПЛА свого пункту призначення.
waypoint_generator.py	Відповідає за створення контрольних точок для маршруту.
transformations_calculations.py	Відповідає за розрахунок орієнтації БПЛА відносно приблизної позиції дошки маркерів ArUco.

uav_flight_modes.py	Відповідає за підготовку до польоту, зміну режиму, виклики сервісів та зміни параметрів БПЛА до PX4.
log_data.py	Відповідає за ведення журналу даних БПЛА під час тестувань.

3.3.2. Структура вузлів ROS

Блок-схему системи, що використовується для бортового управління, можна побачити на рисунку 5. Це дає спрощений огляд основних повідомлень, що розподіляються між вузлами. Загалом у цій схемі використовується **шість вузлів**. Основний вузол називається **drone_control** для якого встановлено 30 оновлень щосекунди. Цей вузол відповідає за публікацію заданих значень на PX4, а також за стан бортової системи. Причиною того, що тільки один вузол публікує задані значення, є міркування безпеки, щоб не більше ніж один вузол міг публікувати нові

ніж один вузол може публікувати нові задані значення для БПЛА.

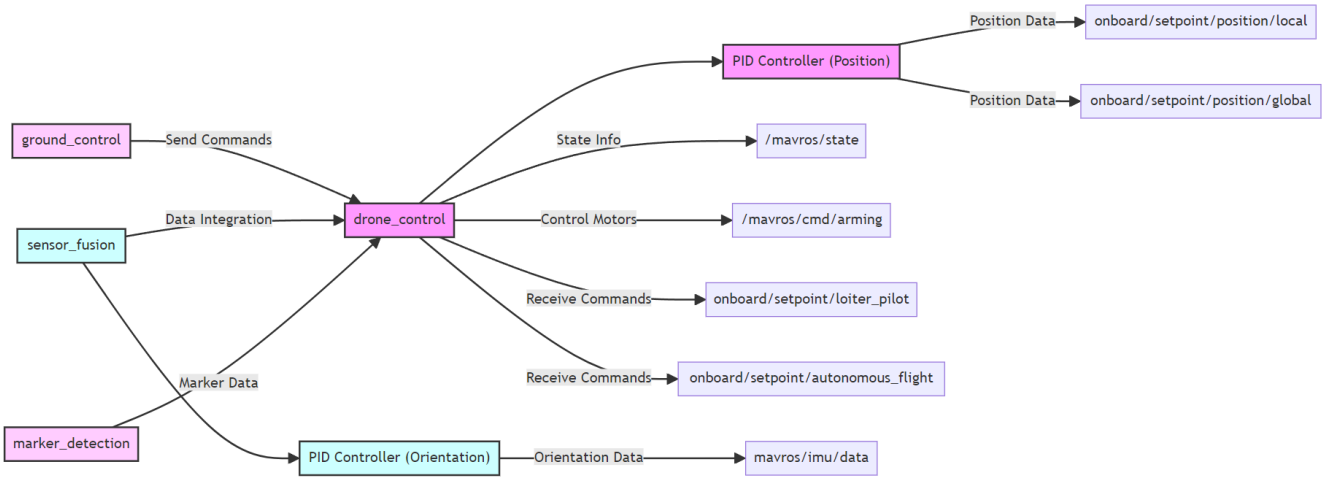


Рисунок 5 - Огляд бортової системи управління.

Стани бортової системи включають зліт, очікування, повернення додому або деякі з місій, які були визначені для виконання під час польоту. Таким чином,

за допомогою команд з клавіатури бортовий стан може бути змінений. Це робиться за допомогою вузла **ground_control**, який оновлюється 10 разів на секунду. Цей вузол перераховує команди клавіатури і публікує їх, якщо відбувається оновлення, наприклад, натискання клавіші. Причиною виділення цього вузла у окремий є те, що натискання клавіш відображаються у головному терміналі, що ускладнює аналіз важливих оновлень з PX4 та інших вузлів який публікує оновлення користувачеві з терміналу.

Інший вузол, який підписується на оновлення від натискань клавіш, називається **loiter_pilot**, який налаштований на 20 оновлень щосекунди. Цей вузол дозволяє користувачеві повністю контролювати БПЛА за допомогою натискання клавіш, що значно полегшує налагодження.

Всі місії, передані БПЛА, будуть контролюватися у вузлі **autonomous_flight**, який налаштований на 10 оновлень щосекунди.

Вузол **marker_detection** буде аналізувати вхідні зображення з фронтальної та нижньої камер і оцінювати позицію ArUco якщо на зображенні є маркери.

Вузол, який використовує позицію маркера ArUco, називається **sensor_fusion** і оновлюється 20 разів на секунду. Цей вузол приймає позицію та оптимізує її на основі злиття з даними IMU та висотоміра (барометра). Це робиться для того, щоб гарантувати, що БПЛА все ще отримує оновлену оцінку положення, навіть якщо на зображенні не виявлено жодних маркерів.

Висновки до розділу 3

Як бачите, платформи Gazebo, MORSE та V-REP мають певні переваги:

- Gazebo відомий своєю здатністю до детального 3D-моделювання, інтеграцією з ROS і багатим набором інструментів для фізичного моделювання.
- MORSE базується на двигуні Blender і володіє гнучкістю в представленні реалістичних середовищ і підтримкою мов програмування Python.
- V-REP вирізняється своєю підтримкою багатьох мов програмування та платформ, включаючи Windows і C#.

Проте, ці платформи також мають обмеження:

- Для Gazebo потрібні значні комп'ютерні ресурси для детального моделювання.
- MORSE, хоча і гнучкий, може не завжди бути оптимальним для додатків, що потребують високої точності.
- V-REP має безкоштовну та комерційну версії. Нажаль, безкоштовна версія не має багатьох функцій як в MORSE або Gazebo.

Враховуючи це, варто скористатися лише деякими платформами, і на мій погляд, Gazebo є найбільш оптимальним вибором завдяки його глибокій інтеграції з ROS, детальному 3D-моделюванню та здатності ефективно моделювати фізичні процеси. Отже, я обрав Gazebo та ROS пакети та nodes для нього, які були детально описаний у цьому розділі.

Розділ 4

Створення оточення і тестування обраних методів для вирішення задачі навігації безпілотного літального апарату в агресивному середовищі

1.1 Системні вимоги

По-перше хотів відмітити системні вимоги щодо програмного забезпечення Gazebo9. Для більш зручного використання мені не підійшов не один з ноутбуків, потрібно використовувати комп’ютер з дискретною відеокартою.

Мої характеристики, які відображені на рисунку 6:

- Intel(R) Core(TM) i5-3330 CPU @ 3.00GHz
- 8 GB RAM
- NVIDIA GeForce GTX 750 Ti/PCIe/SSE2

Інше, що є дуже важливим — підбір правильної ОС. Ubuntu 16.04 (Xenial Xerus) та Ubuntu 20.04 (Focal Fossa) на стадії завантаження програмного забезпечення Gazebo9 виникає забагато помилок із-за не підтримки пакетів у випадку з 20.04 (більшість пакетів просто вже не підтримуються і нереалізовані в бібліотеці пакетів debian) та з 16.04, що була оптимізована під ПО Gazebo8.

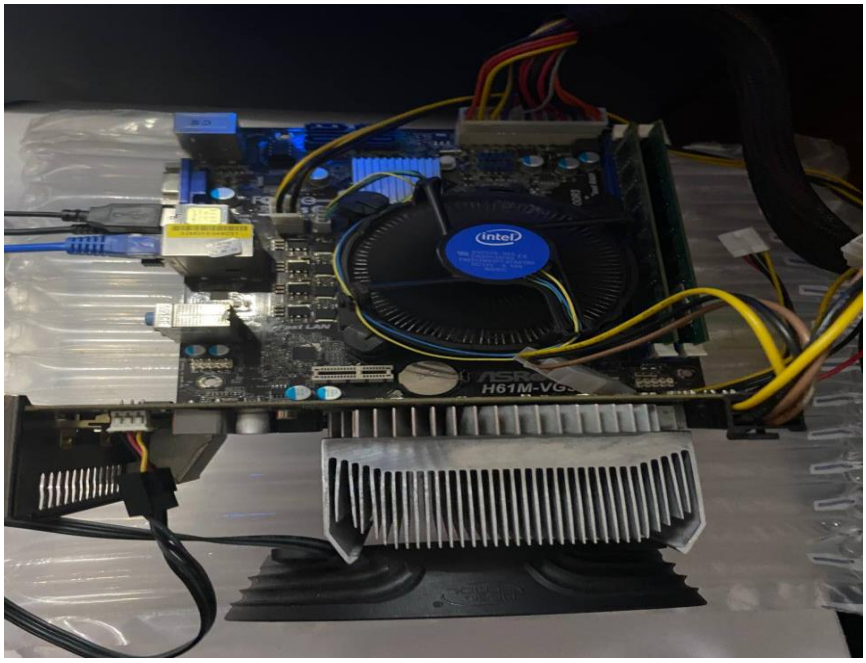


Рисунок 6 - реалізована система для проведення тестів

Тобто, найкраще для тестування методів підходить Ubuntu 18.04 (BionicBeaver).

В ході тестів, результат симуляцій напряму залежав від конфігурації обладнання. Тобто, на певному hardware деякі тести просто приймали статус crashed. Тестування на 6-ти системах привели тільки до одного висновку, залежність від типу пам'яті (DDR3-DDR3L між DDR4). Висновок, симуляції безпроблемно працюють тільки на пам'яті типу DDR3-DDR3L на операційній системі Ubuntu 18.04 (BionicBeaver).

1.2 Оцінка положення за допомогою ArUco для переходу від GPS до візуального сприйняття.

4.2.1. Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при відсутності вітру.

Мета: дослідження здатності дрона утримувати позу перед маркерною дошкою GPS2Vision. Результат цього дасть результативність алгоритмів керування для керування позою та приведе до висновку, чи потрібно виконувати подальшу оптимізацію в управлінні позою.

Стартове положення: дрон буде тримати свою позицію приблизно на відстані 4 метрів від дошки GPS2Vision без вітру.

Кінцеве положення: тестування закінчиться в тому ж положенні в якому і починалося, тобто, приблизно на відстані в 4 метри від дошки GPS2Vision.

Умови: вакуум.

Цей тест був проведений в тестовій симуляції.

```
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world drone_control_args:="hold_aruco_pose_test" x:=-4.0 y:=0.0
```

Лістинг 1 - Запуск тесту «Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при відсутності вітру»

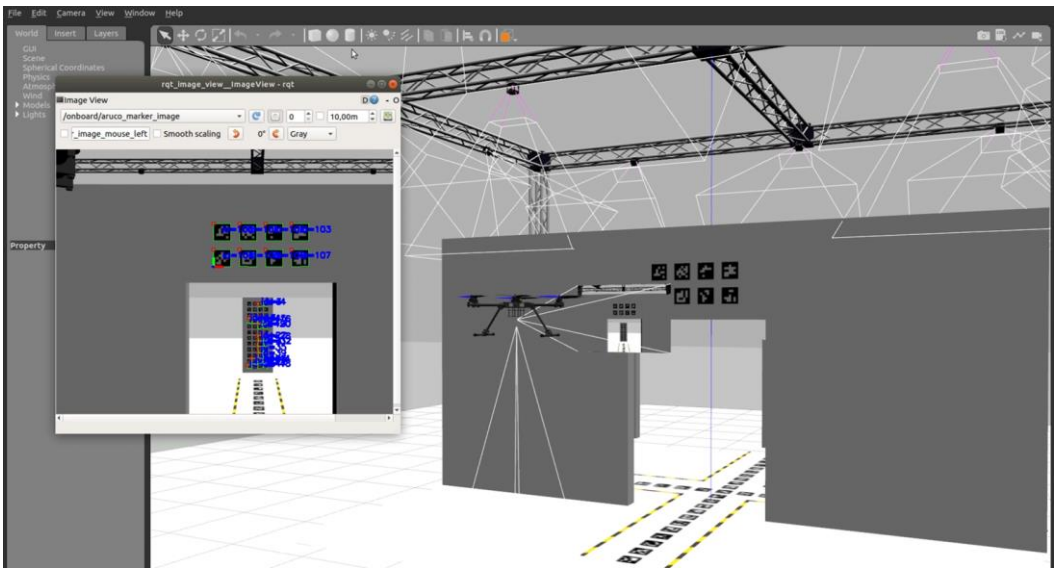


Рисунок 7 - Середовище тестування «Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при відсутності вітру»

Графік поводження дрону, завдяки gps відслідковуванню.

Характеристики на графіку:

- червоний – висота
- фіолетовий – швидкість на схід
- зелений - швидкість вгору
- синій - швидкість на північ

Така система контролю графіку буде використовуватися в інших тестових симуляціях.

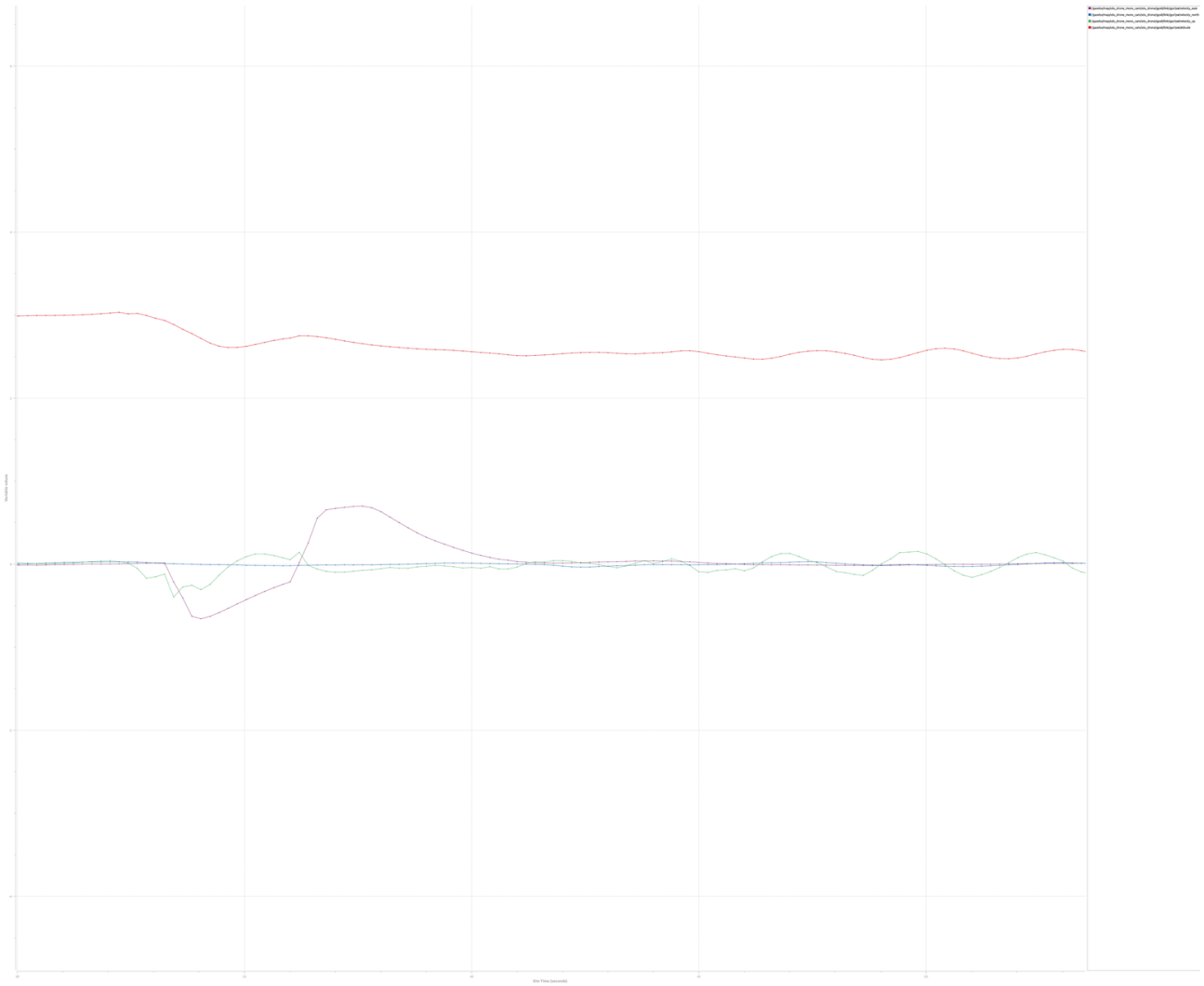


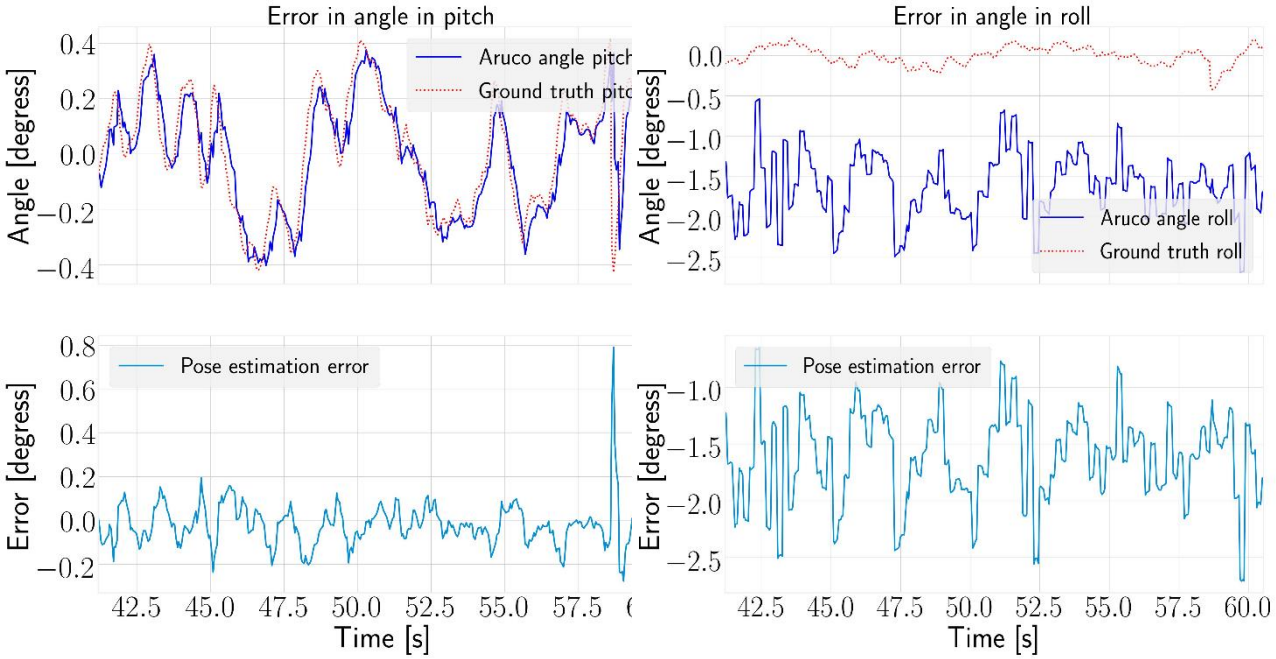
Рисунок 8 - Графік тестування «Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при відсутності вітру»

Таблиця 6

Вихідні результати тестування

№ тесту	Вітер	Середня помилка aruco pos	STD помилка aruco pos	Середня похибка кута aruco	STD помилка кута апроксимації
1	Hi	0.044	0.017	0.62	0.19

2	Hi	0.044	0.016	0.60	0.22
3	Hi	0.047	0.017	0.61	0.20



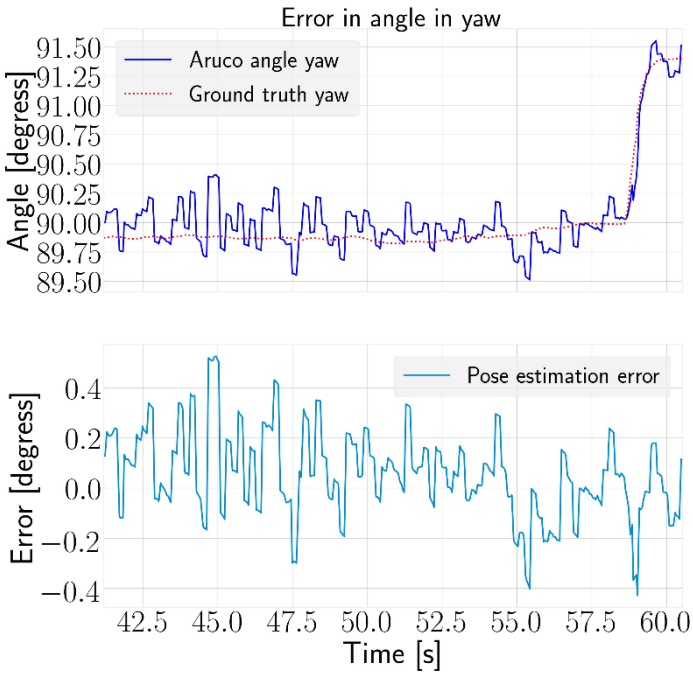
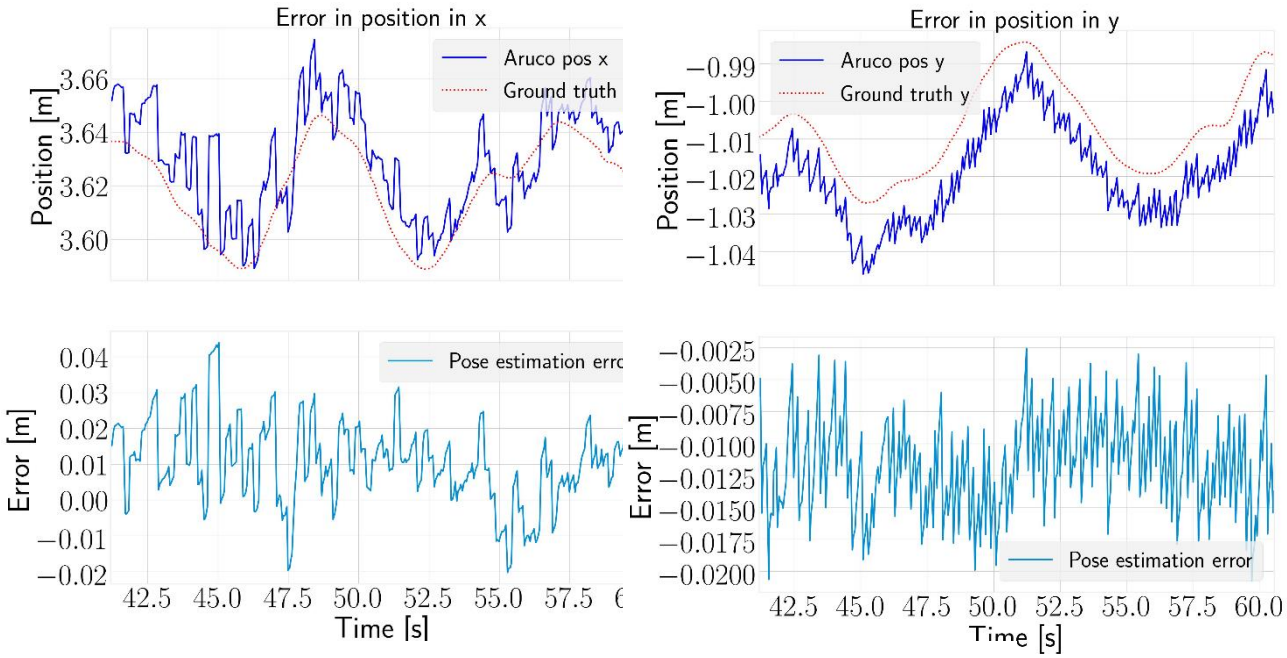


Рисунок 9, 10, 11 - Графіки помилки кута нахилу, тангажу, повороту відповідно для тесту 1



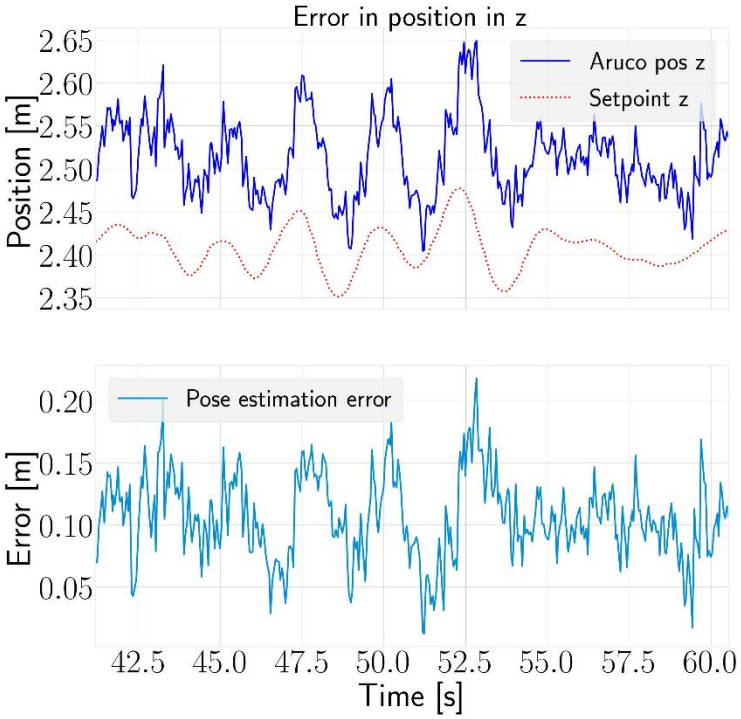


Рисунок 12, 13, 14 - Графіки помилки позицій x, y, z для тесту 1

Висновок:

За результатами можна побачити невелике відхилення під час зміни дистанції до дошки GPS2Vision, що в свою чергу спричинило стрибок швидкості на схід.

Щодо іншого можна додати, що графік є більш стабільний, як для дрону який намагається зафіксувати фіксоване положення.

З графіків видно, що є невеликі відхилення у всіх тестах, найбільша помилка в графіку помилки тангажу на -2.5 градуси, що не є занадто критичним значенням.

Можна зауважити, що алгоритми керування є результативними, тобто, якщо середовище має ідеальні умови (без сильних поривів вітру, тощо), оптимізація в управлінні позою зовсім не потрібна.

4.2.2. **Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при вітрі**.

Мета: дослідження здатності дрона утримувати позу перед маркерною дошкою GPS2Vision. Результат цього дасть результативність алгоритмів керування для керування позою та приведе до висновку, чи потрібно виконувати подальшу оптимізацію в управлінні позою.

Стартове положення: дрон буде тримати свою позицію приблизно на відстані 4 метрів від дошки GPS2Vision без вітру.

Кінцеве положення: тестування закінчиться в тому ж положенні в якому і починалося, тобто, приблизно на відстані в 4 метри від дошки GPS2Vision.

Умови: вітер 1, 2, 5, 10, 15, 20, 30 м/с.

```
roslaunch px4 gazebo_sim_v1.0.launch
worlds:=optitrack_big_board_onepattern.world
drone_control_args:="hold_aruco_pose_test" x:=-4.0 y:=0.0
```

Лістинг 2 - Запуску тесту «Утримання положення за допомогою оцінки положення
ArUco для дошки переходу від GPS до візуального сприйняття при вітрі»

Середні показники будуть вказані в середньому значенні для трьох тестів в зв’язку з великою кількістю тестів для різних умов.

Таблиця 7

Вихідні результати тестування для всіх типів тестів

№ тесту	Вітер	Середня помилка aruco pos	STD помилка aruco pos	Середня похибка кута aruco	STD помилка кута апроксимації
1-3	1 м/с	0.04	0.025	0.69	0.33
4-6	2 м/с	0.06	0.029	0.7	0.42
7-9	5 м/с	0.07	0.052	1.07	0.86
10-12	10 м/с	0.09	0.067	1.47	1.05
13-15	15 м/с	0.3	0.1	4.7	2.3
16-18	20 м/с	1.3	1.1	6.12	3.5
19-21	30 м/с	4.2	2.7	6.6	3.8

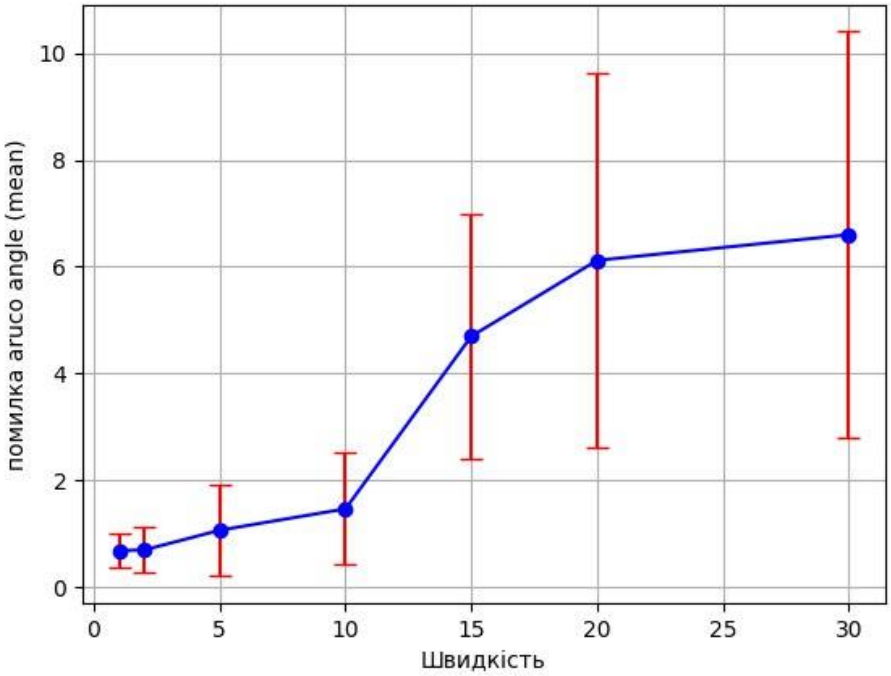


Рисунок 15 - Графік похибки aruco кута до швидкості, з error bars до STD помилки кута апроксимації

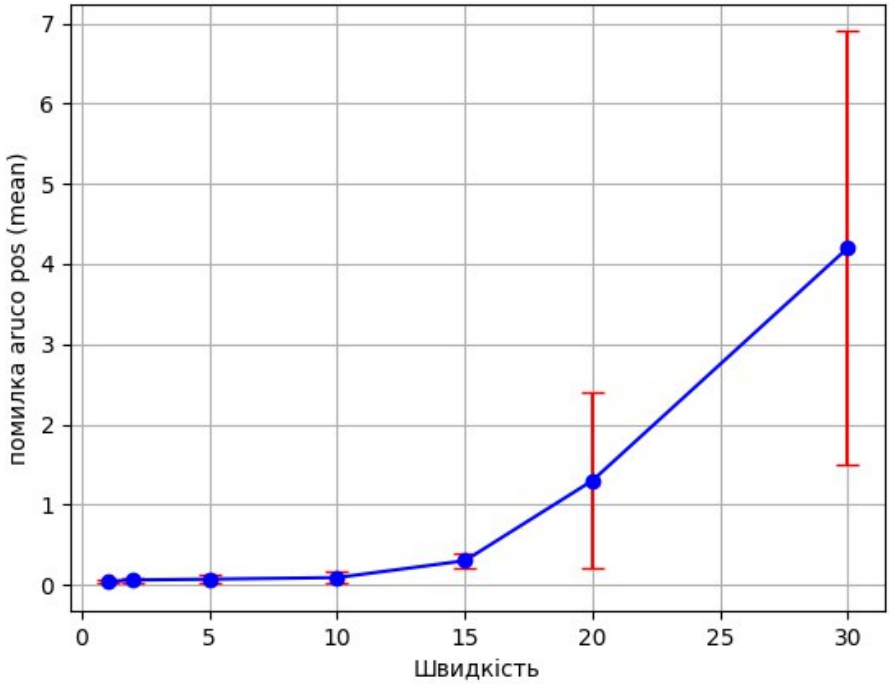
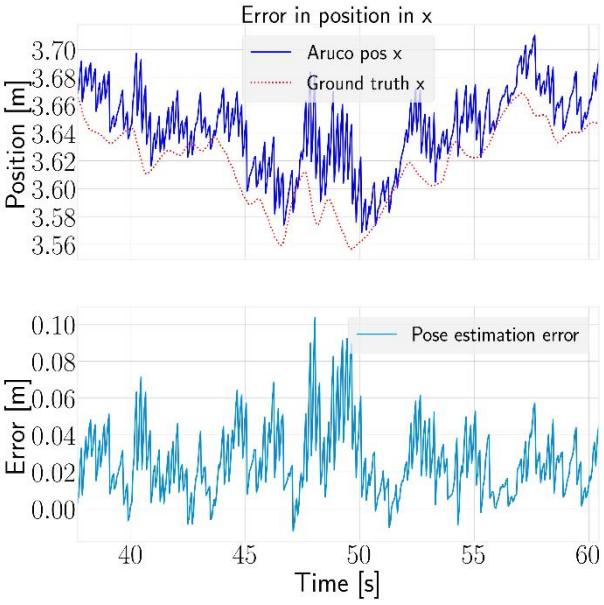
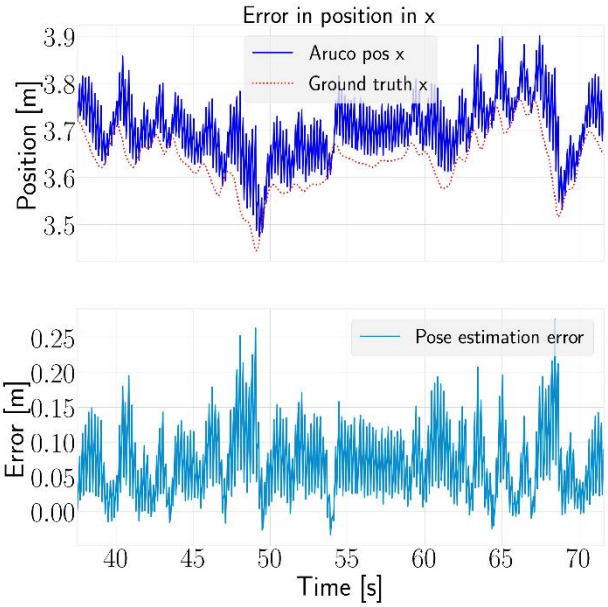


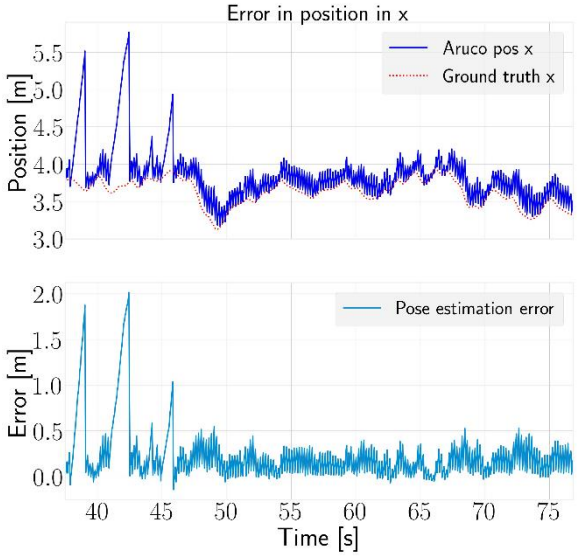
Рисунок 16 - Графік середньої помилки aruco pos до швидкості, з error bars до STD помилка aruco pos



Тест 1 (для 1 **м/с)**

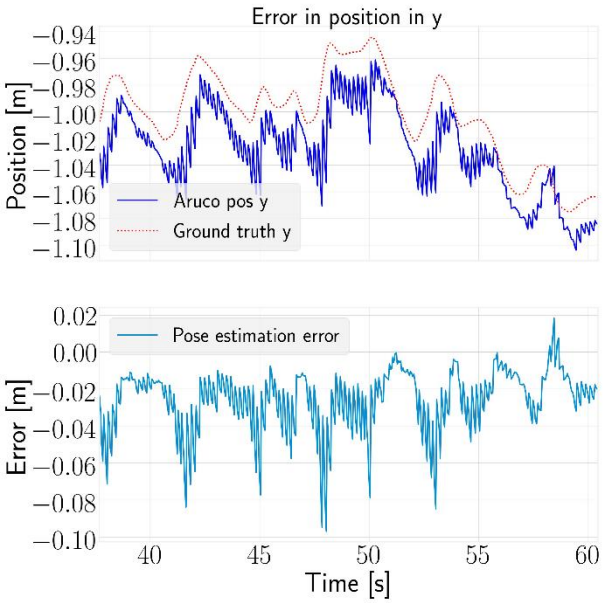


Тест 7 (для 5 **м/с)**

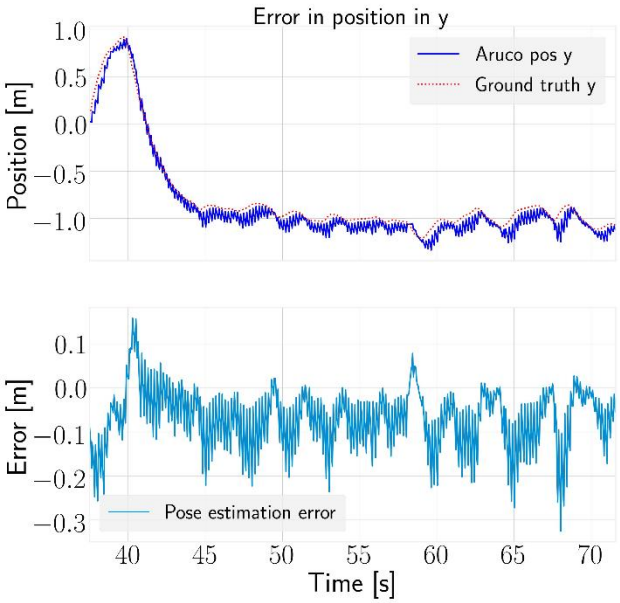


Тест 16 (для 20 **м/с)**

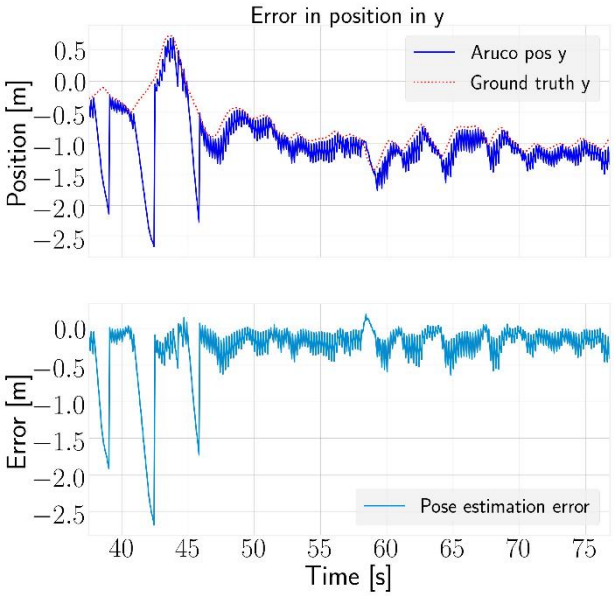
Рисунок 17, 18, 19 - Графіки тестування «Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при вітрі» для координати X.



Тест 1 (для 1 **м/с**)

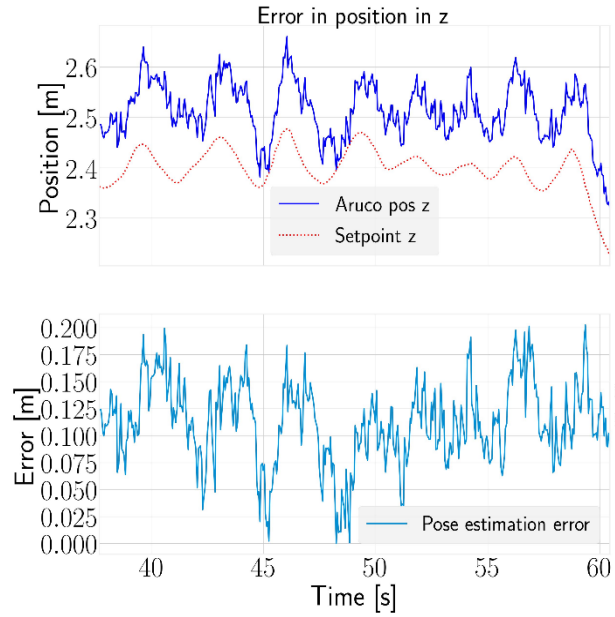


Тест 7 (для 5 **м/с**)

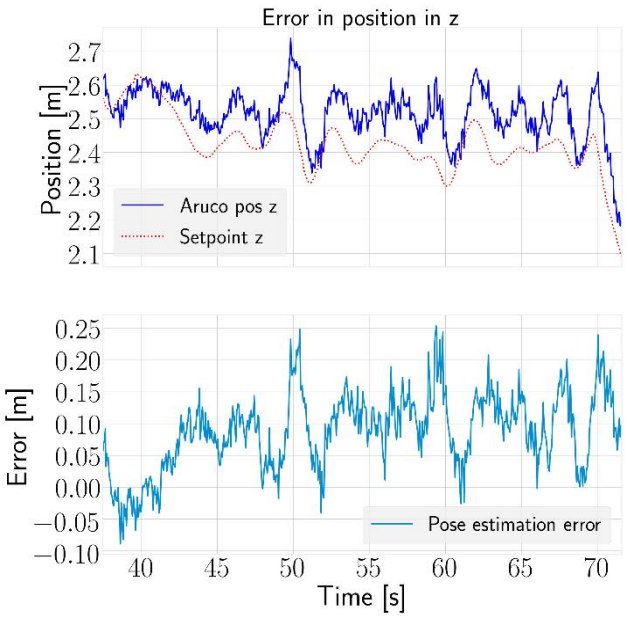


Тест 16 (для 20 **м/с**)

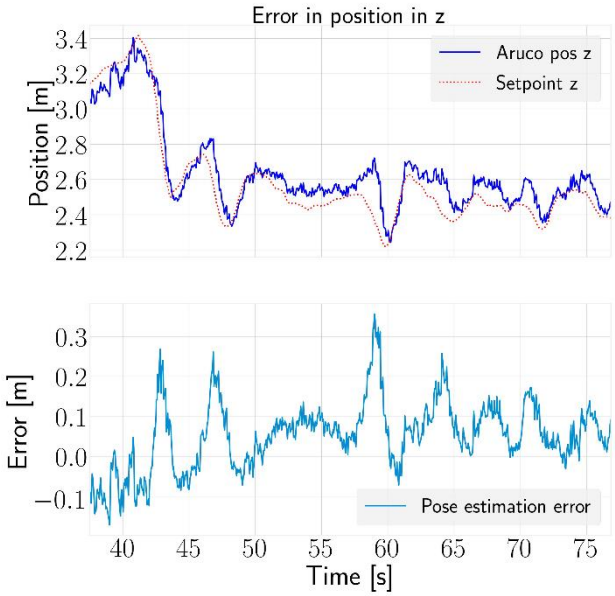
Рисунок 20, 21, 22 - Графіки тестування «Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при вітрі» для координати Y.



Тест 1 (для 1 м/с)



Тест 7 (для 5 **м/с**)



Тест 16 (для 20 **м/с**)

Рисунок 23, 24, 25 - Графіки тестування «Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при вітрі» для координати Z

Налаштування середовища:

```
<plugin name='wind_plugin' filename='libgazebo_wind_plugin.so'>
  <frameId>base_link</frameId>
  <!--<robotNamespace/>-->
  <namespace>/foo/bar</namespace>
  <xyzOffset>0 0 0</xyzOffset>
  <windDirectionMean>1 1 1</windDirectionMean> <!--1 1 1-->
  <windDirectionVariance>0.05</windDirectionVariance> <!--0.05-->
  <windVelocityMean>1.0</windVelocityMean> <!--1.0 and 1.5-->
  <windVelocityVariance>0.05</windVelocityVariance> <!--0.05-->
  <windGustDirection>1 1 1</windGustDirection> <!--1 1 1-->
  <windPubTopic>world_wind</windPubTopic>
</plugin>
```

Рисунок 26 - Налаштування середовища симуляції для вітру в 5-7 **м/с**

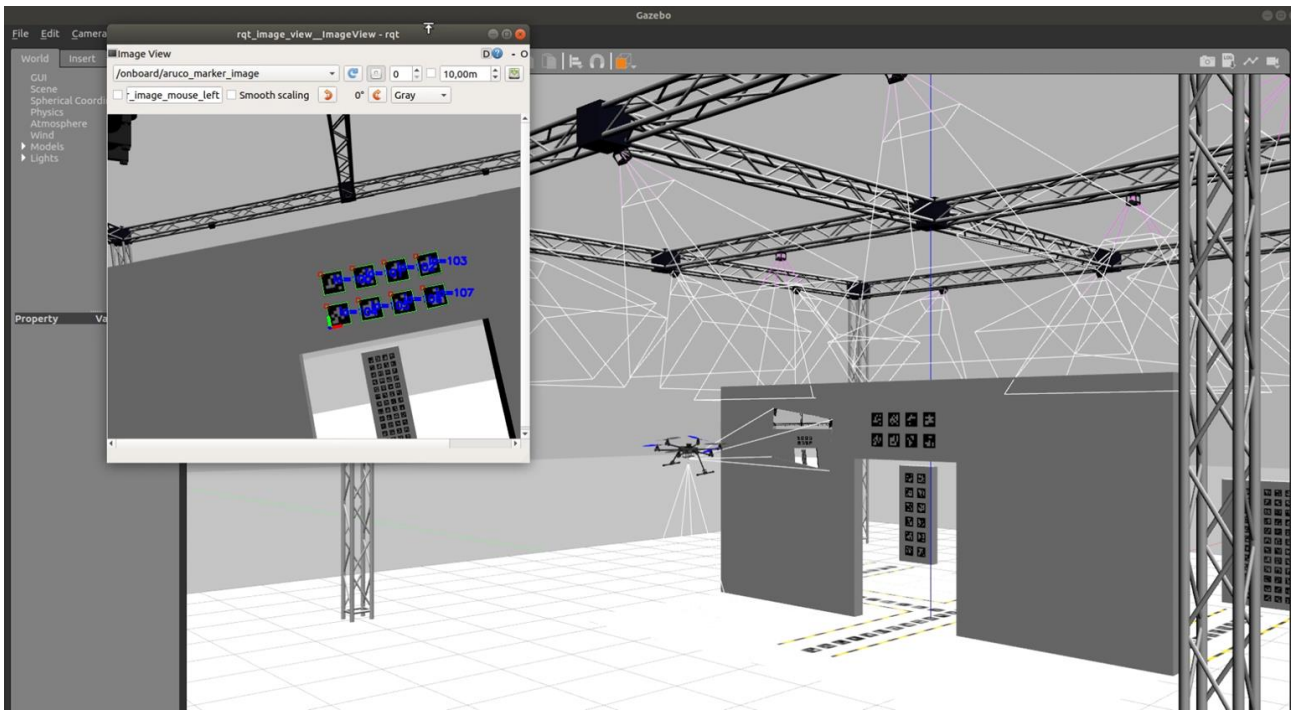


Рисунок 27 - Тестування симуляції «Утримання положення за допомогою оцінки положення ArUco для дошки переходу від GPS до візуального сприйняття при вітрі»

Висновок:

Як можна побачити на графіку, швидкість дрону стала дещо хаотичною. Як в північному, східному напрямку. Висота та швидкість вгору змінювалась за тестом, тому в цьому випадку відхилень немає.

Немає циклічності і поступовості, як в тесті 4.3.1.

На координатах X, Y, Z, за різними тестами з різною швидкістю вітру можна побачити, що зі збільшенням сили вітру бачимо більше відхилення, а також збільшення середньої помилки aruco pos. Більш оптимально дрон поводить себе в середовищі до 15 **м/с.** Тому що, середня помилка більше чим 1.0.

Можна зробити висновок, що вітер має великий вплив на стабілізацію дрону в просторі, що видно по середній похибці на рисунках 16 та 17. Також, можна зауважити, що дрон намагається стабілізуватися в просторі і в сумі в нього це виходить, якщо швидкість вітру є не дуже значною (до 15 м/с).

Для покращення опору до вітру дроном потрібно:

- Найпростіший метод покращення, це вдосконалення аеродинамічного дизайну дрона, для зменшення аеродинамічного опору та покращення стабільності при сильному вітрі.
- Наступний метод полягає в регулюванні пропелерів для дрону. Наприклад, використання пропелерів зі змінним кроком або різною швидкістю обертання.
- Більш складніший метод, це запровадження машинного навчання або штучного інтелекту які будуть адаптуватися під середовище відповідно до умов. Тобто, якщо маємо занадто велику похибку щодо положення – управлінням позою займається ШІ, який завдяки акселерометрам, гіроскопам реагує на пориви вітру.

1.3 Перехід з GPS до візуальної навігації

Найважливішою частиною симуляцій був фактичний перехід від використання GPS до навігації за допомогою зору. Оскільки вплив вітру відіграє важливу роль у стабільності цього переходу, симульоване середовище буде включати вітер (і без вітру) з різним діапазоном швидкостей для стрес-тесту системи. Та будуть перекриті маркери Aruco завдяки методу стекового розмиття.

4.3.1. Перехід з GPS до візуальної навігації без вітру

Мета: побачити як дрон працює при переході від GPS до навігації на основі зору. У цьому стані дрон повинен знайти маркерну дошку ArUco, розташовану на стіні, після польоту до точки, визначеної користувачем за допомогою GPS.

Стартове положення: дрон буде розміщено на відстані двадцяти метрів перед дошкою GPS2Vision.

Фінальне положення: в кінці тесту, дрон займає положення перед дошкою GPS2Vision приблизно на 2 метри.

Умови: вакуум.

Оскільки GPS має певну похибку, остаточна точка була встановлена з похибкою +-6 метрів по осях x та y, щоб проілюструвати цю невизначеність.

Це було зроблено для стрес-тесту здатності дрона знайти дошку з урахуванням цієї похибки.

В цьому випадку БПЛА рухається до визначеної локації GPS. Після цього, як БПЛА досягає цієї локації, тобто, GPS, починається пошук дошки GPS2Vision. Коли дошка GPS2Vision була знайдена, БПЛА рухається до дошки ще завдяки GPS навігації. Коли відстань не більше 5 метрів від дошки GPS2Vision то відбувається перехід до візуальної навігації. Після того, як БПЛА досягає останньої точки у своєму маршруті, він починає використовувати камеру, розташовану внизу, для здійснення візуальної навігації.

Було проведено десять стартів БПЛА за умов відсутності вітру. Точки, розташовані між лініями, показують реальну точку визначення позиції БПЛА. Сині, червоні та жовті лінії ідентифікують використання GPS для оцінки позиції БПЛА для навігації, в той час як зелені лінії позначають використання дошки GPS2Vision для цієї ж мети.

В наступних тестах використання цієї ідентифікації буде аналогічне.

```
roslaunch px4 gazebo_sim_v1.0.launch
worlds:=optitrack_big_board_onepattern.world
drone_control_args:="GPS2Vision_test" x:=-21.0 y:=0.0 headless:=false gui:=true
```

Лістинг 3 - Запуску тесту «Перехід з GPS до візуальної навігації без вітру»

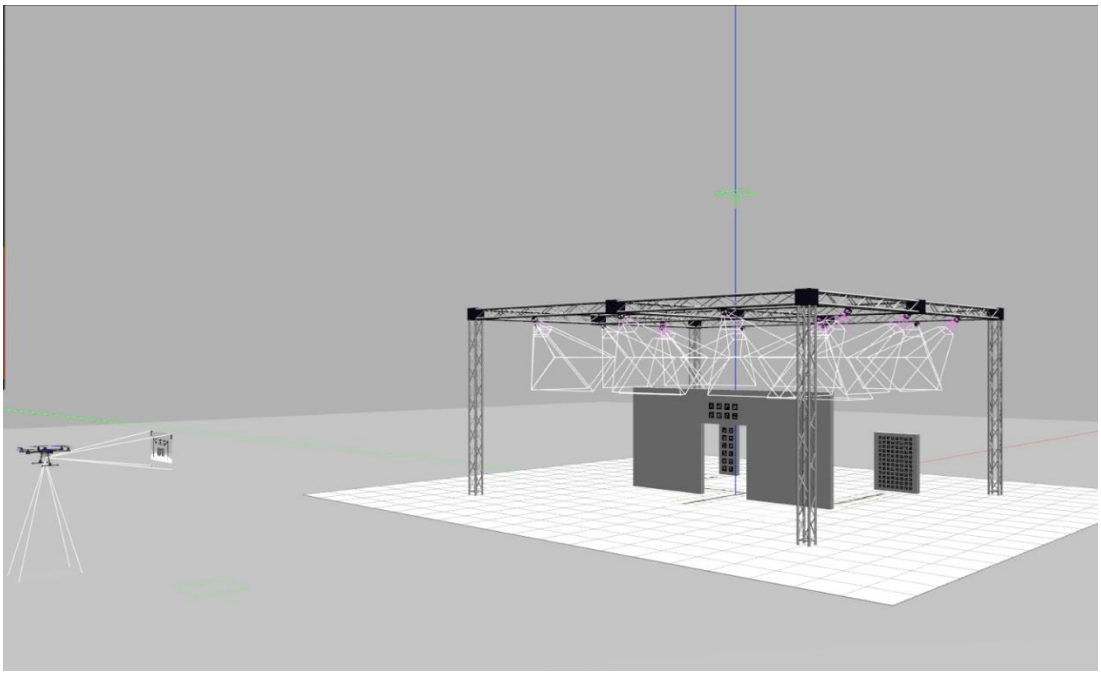


Рисунок 28 - Тест «Перехід з GPS до візуальної навігації без вітру»

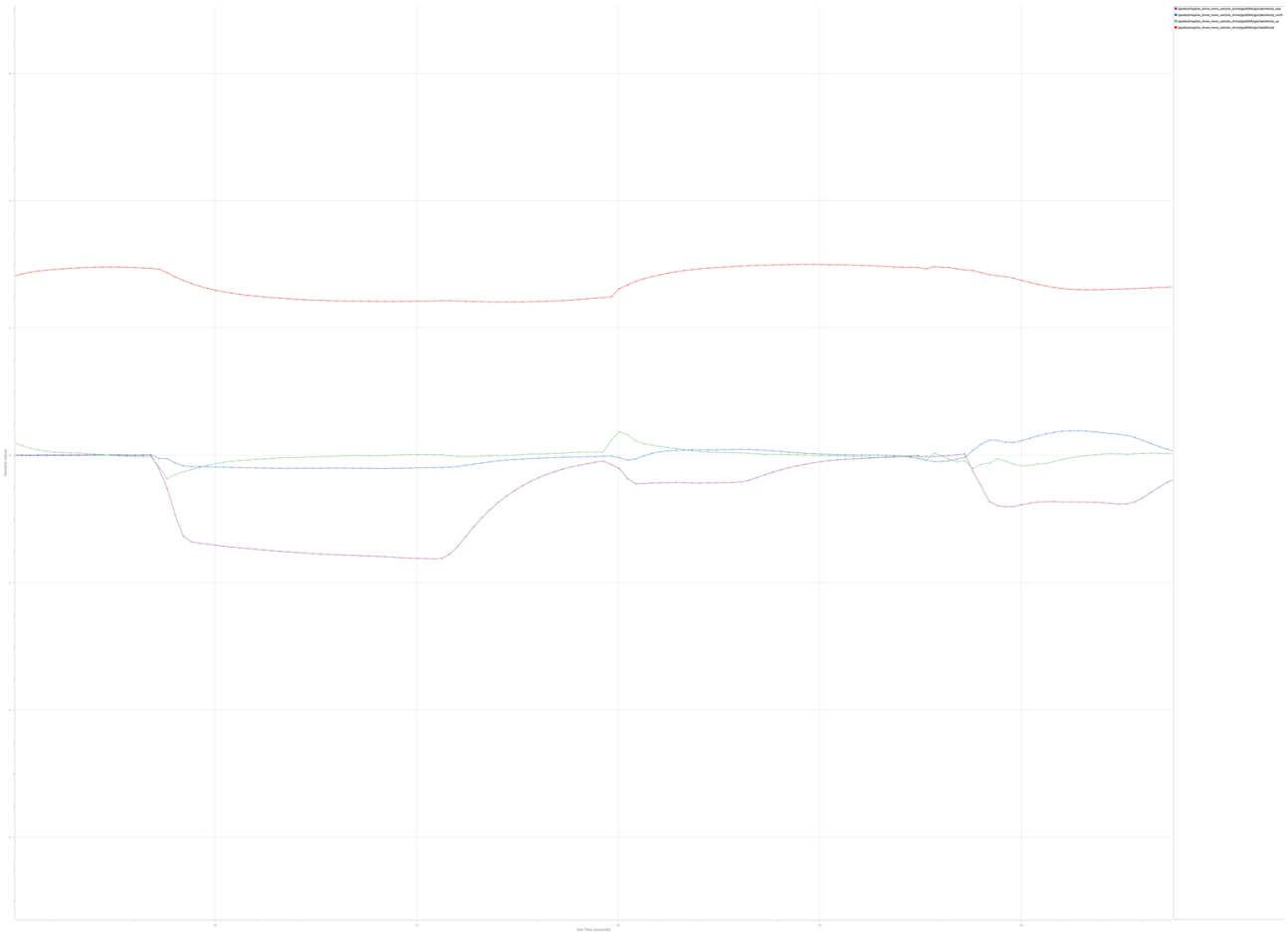


Рисунок 29 - Графік тестування «Перехід з GPS до візуальної навігації без вітру»

Таблиця 8

Вихідні середні результати виконання тестів

Запусків	10
Час польоту GPS	7.7724
Визначений час дороги борту	1.6164
Навігаційний час дороги борту	12.5172
GPS2Vision час польоту	6.3864

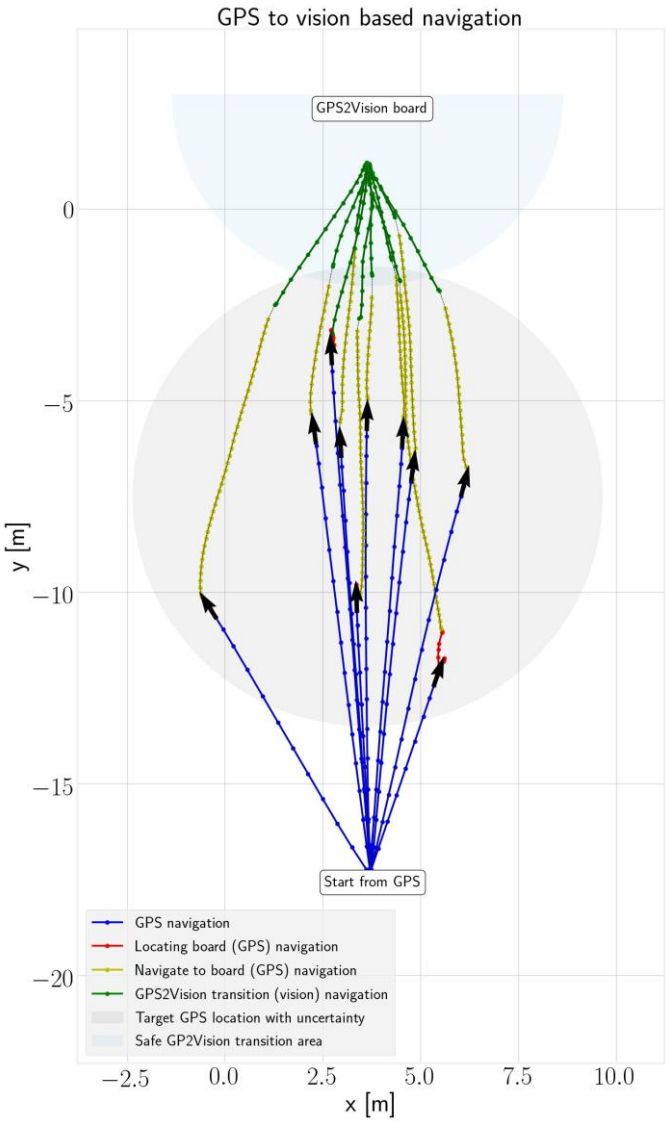


Рисунок 30 - Графік маршруту «Перехід з GPS до візуальної навігації без вітру»

Висновок:

По графіку можна побачити, що рух відбувається поступально, висота та швидкість змінюється динамічно.

Тест був виконаний успішно, що в свою чергу демонструють результати виконання на рисунку. Дрон завдяки GPS легко знайшов маркерну дошку ArUco, розташовану на стіні в двадцяти метрах від нього.

Середній час виконання тесту становив 1:24.

4.3.2. Перехід з GPS до візуальної навігації з вітром

Мета: побачити як дрон працює при переході від GPS до навігації на основі зору. У цьому стані дрон повинен знайти маркерну дошку ArUco, розташовану на стіні, після польоту до точки, визначеної користувачем за допомогою GPS.

Стартове положення: дрон буде розміщено на відстані двадцяти метрів перед дошкою GPS2Vision.

Фінальне положення: в кінці тесту, дрон займає положення перед дошкою GPS2Vision приблизно на 2 метри.

Умови: вітер 1, 2, 5, 10, 15, 20 м/с.

```
roslaunch px4 gazebo_sim_v1.0.launch
worlds:=optitrack_big_board_onepattern.world
drone_control_args:="GPS2Vision_test" x:=-21.0 y:=0.0 headless:=false gui:=true
```

Лістинг 4 - Запуску тесту «Перехід з GPS до візуальної навігації з вітром 5-7 м/с»

Середовище налаштовується аналогічно як на рисунку 26.

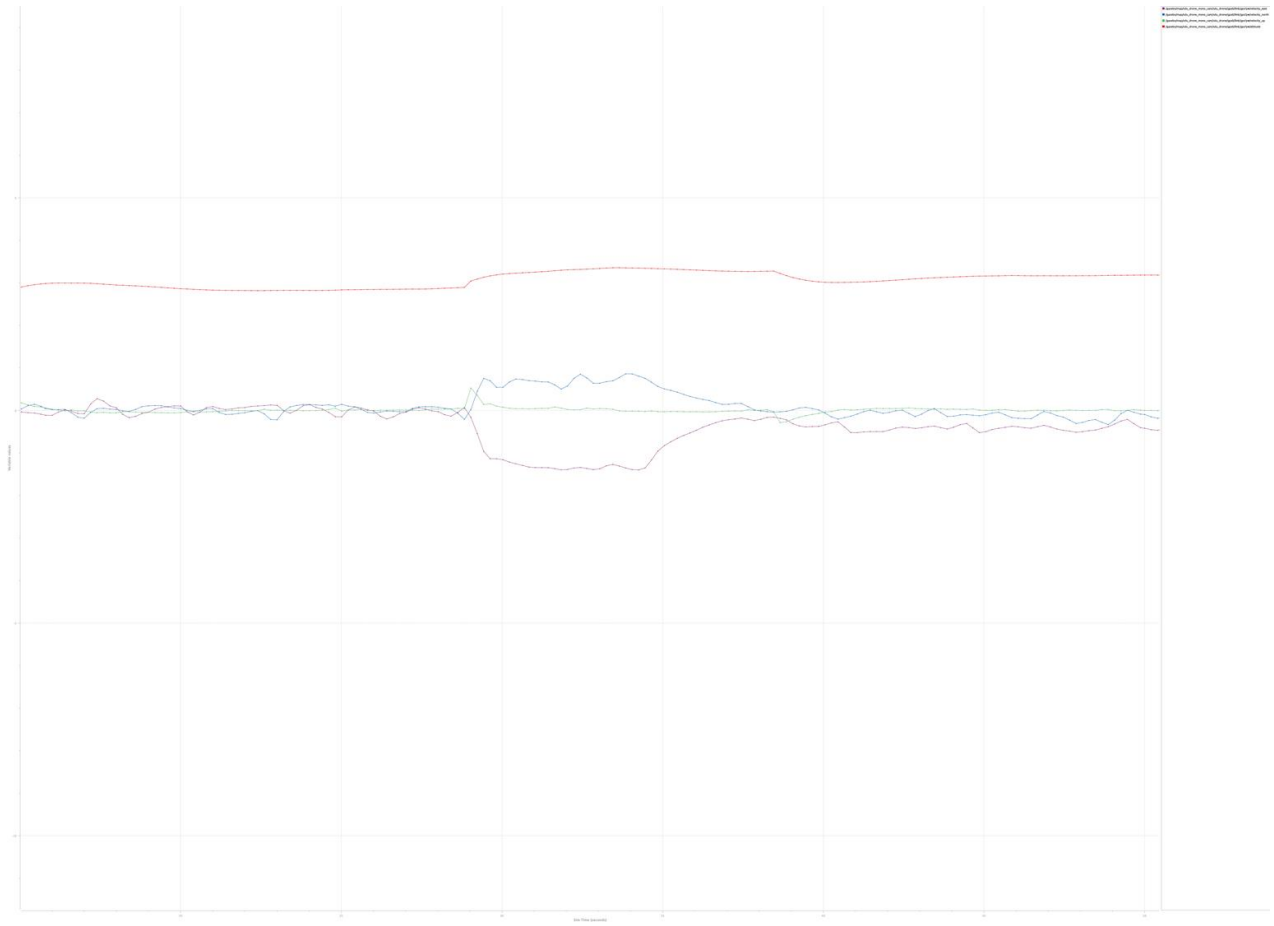


Рисунок 31 - Графік «Перехід з GPS до візуальної навігації з вітром»

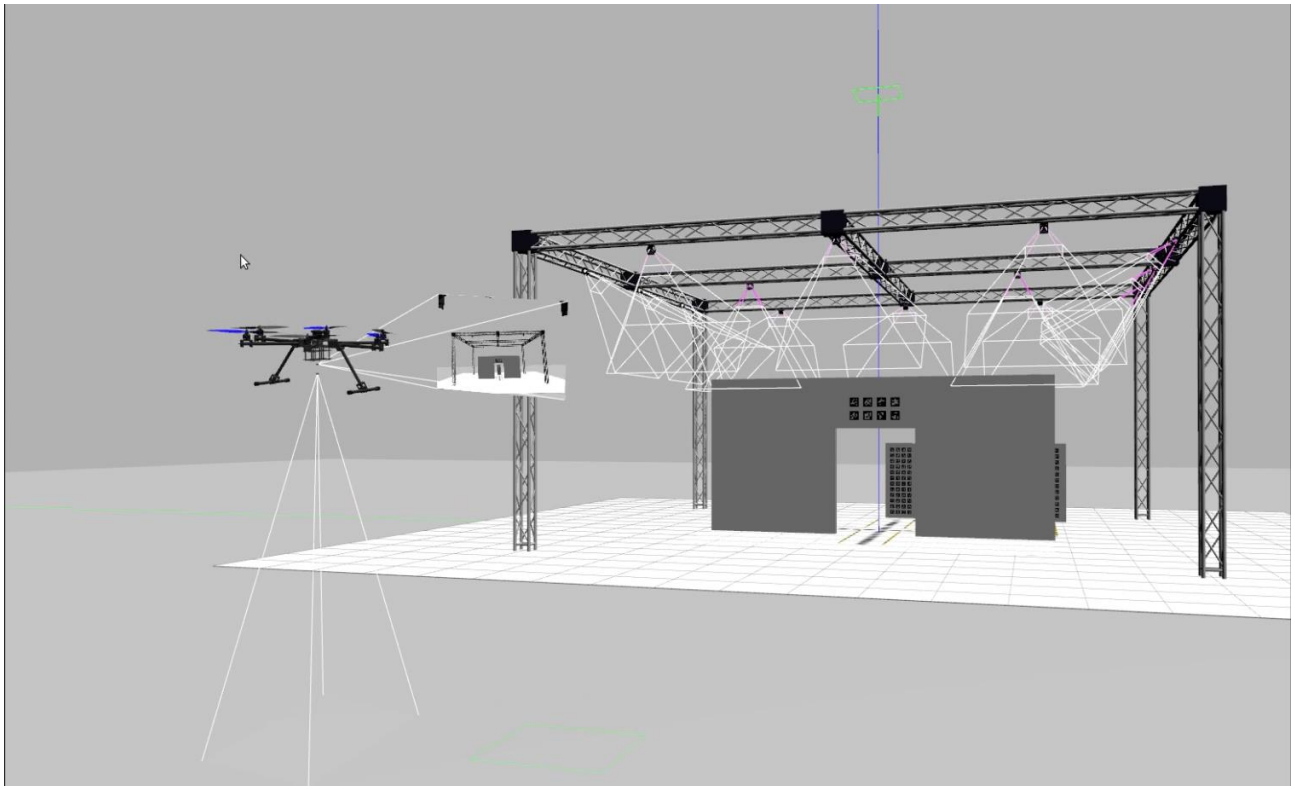


Рисунок 32 - Скріншот «Перехід з GPS до візуальної навігації з вітром»

Таблиця 9

Вихідні середні результати виконання тестів для 1 **м/с**

Запусків	10
Час польоту GPS	7.6860
Визначений час дороги борту	2.4480
Навігаційний час дороги борту	13.7340
GPS2Vision час польоту	6.3756
Вітер	1 **м/с**

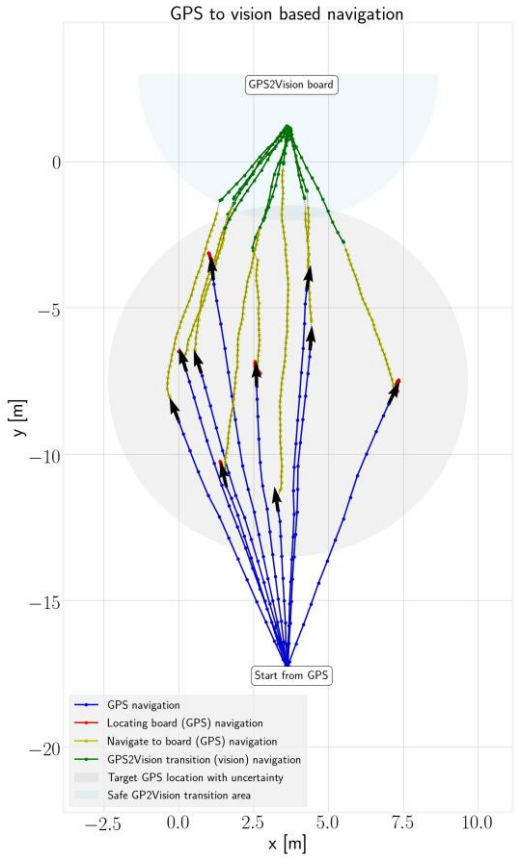


Рисунок 33 - Графік маршруту дрону з 10 запусків для 1 **м/с**

Таблиця 10

Вихідні середні результати виконання тестів для 2 **м/с**

Запусків	10
Час польоту GPS	6.8904

Визначений час дороги борту	1.1268
Навігаційний час дороги борту	14.3952
GPS2Vision час польоту	6.0912
Вітер	2 **м/с**

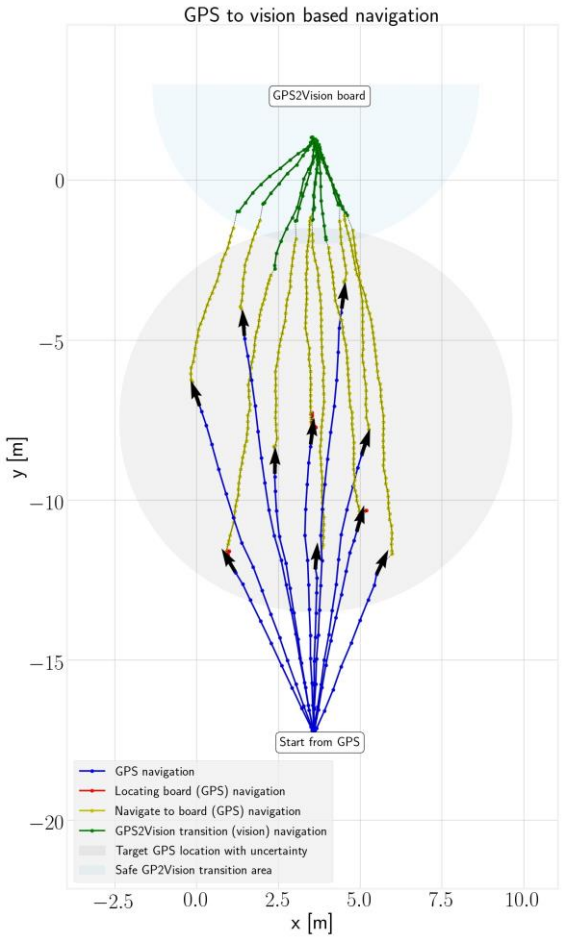


Рисунок 34 - Графік маршруту дрону з 10 запусків для 2 **м/с**

Таблиця 11

Вихідні середні результати виконання тестів для 5 **м/с**

Запусків	10
Час польоту GPS	7.1028
Визначений час дороги борту	1.5228
Навігаційний час дороги борту	16.2504

GPS2Vision час польоту	6.9588
Вітер	5 **м/с**

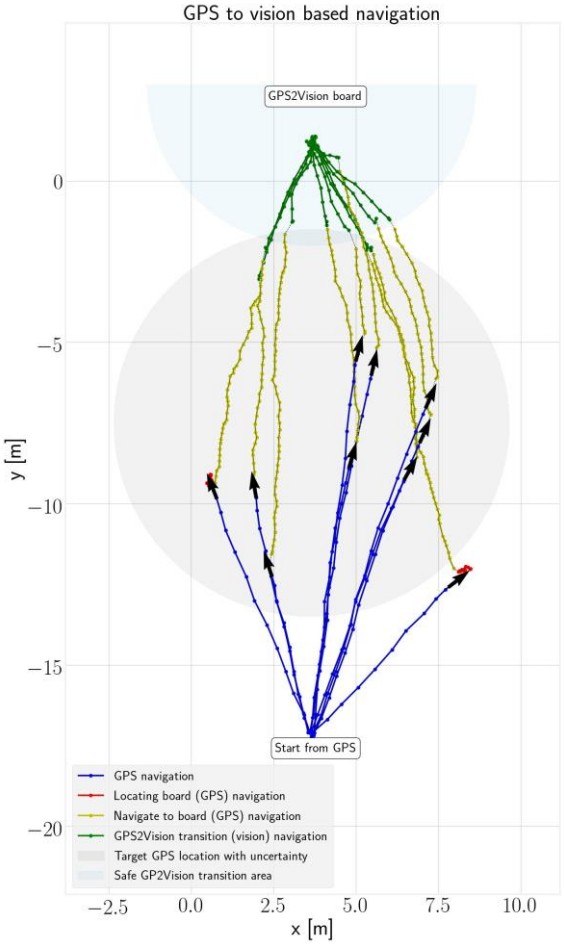


Рисунок 35 - Графік маршруту дрону з 10 запусків для 5 **м/с**

Таблиця 12

Вихідні середні результати виконання тестів для 10 **м/с**

Запусків	10
Час польоту GPS	7.3476
Визначений час дороги борту	6.1236
Навігаційний час дороги борту	19.8072
GPS2Vision час польоту	127.1592
Вітер	10 **м/с**

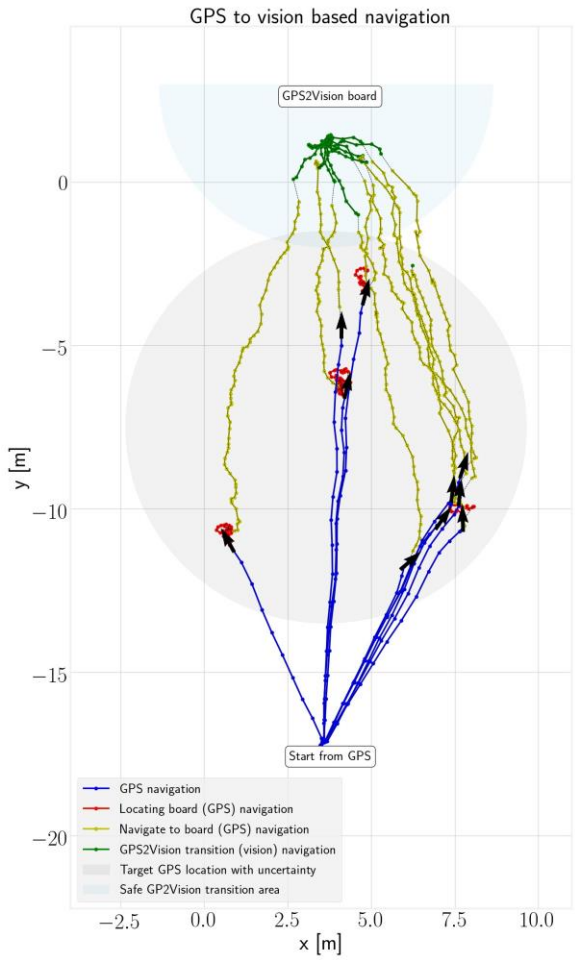


Рисунок 36 - Графік маршруту дрону з 10 запусків для 10 **м/с**

Висновок:

Порівнюючи рис. 31 і рис. 30 можна побачити, що в діапазоні від 15 до 50 секунд є відчутно збільшена тривалість польоту дрона. Також чудово видно по рисункам 33, 34, 35, 36, що є нестабільність та різниця швидкостей в різних площинах, що може вказувати на те, що дрон намагається стабілізуватися в просторі.

Отже, вітер відчутно впливає на задачу поставлену дрону.

Середній час виконання тесту становив 1:56. Що є відчутно більшим від тесту 4.5.1.

Тестування з вітром силою 20 м/с було неуспішним. Дрон не знаходив маркерну дошку GPS2Vision і тест не міг пройти далі по сценарію. Тобто, якщо вітер більше 15 м/с, загальна навігація завдяки GPS працює не успішно.

За таблицями 9, 10, 11, 12 можна побачити, що при збільшенні вітру залежність від часу польоту зростає, тобто, чим більша сила вітру, тим більше зростає час польоту. За таблицею 12 видно, що в одному із тестів час польоту GPS2Vision був настільки довгим, що середній час виріс аж до 127 секунд.

1.4 Візуальна навігація

Щоб проаналізувати, як працює система, коли маркери не завжди видно на зображенні, було запропоновано набір різних плат ArUco розташованих на землі. Це зроблено для стрес-тестування системи та оцінки продуктивності за допомогою реалізованого об'єднання датчиків. БПЛА має переміщатися із заздалегідь визначеного місця до однієї з посадочних станцій, а потім назад.

4.4.1. Візуальна навігація за допомогою повної дошки ArUco з горизонтальною швидкістю 1.0 м/с

Мета: цей тест був проведений для оцінки ефективності навігації безпілотника на основі зору з використанням маркерів ArUco. Тут оцінка пози буде порівнюватися з наземною істиною дрона, щоб побачити, наскільки добре працює емуляція пози, коли дрон рухається в змодельованому середовищі.

Стартове положення: дрон буде розміщено на відстані приблизно двох метрів перед дошкою GPS2Vision.

Фінальне положення: в кінці тесту, дрон займає положення перед дошкою GPS2Vision приблизно на 2 метри.

Умови: вакуум. Використовується повна маркерна дошка ArUco розміром 25 на 25 за допомогою нижньої камери дрона, задля більшої точності навігації.

Швидкість руху вперед встановлена на 1 м/с.

```
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world drone_control_args:="vision_navigation_test" x:=-3.0 y:=0.0 headless:=false gui:=true
```

Лістинг 5 - Запуску тесту «Візуальна навігація за допомогою повної дошки ArUco з горизонтальною швидкістю 1.0 м/с»

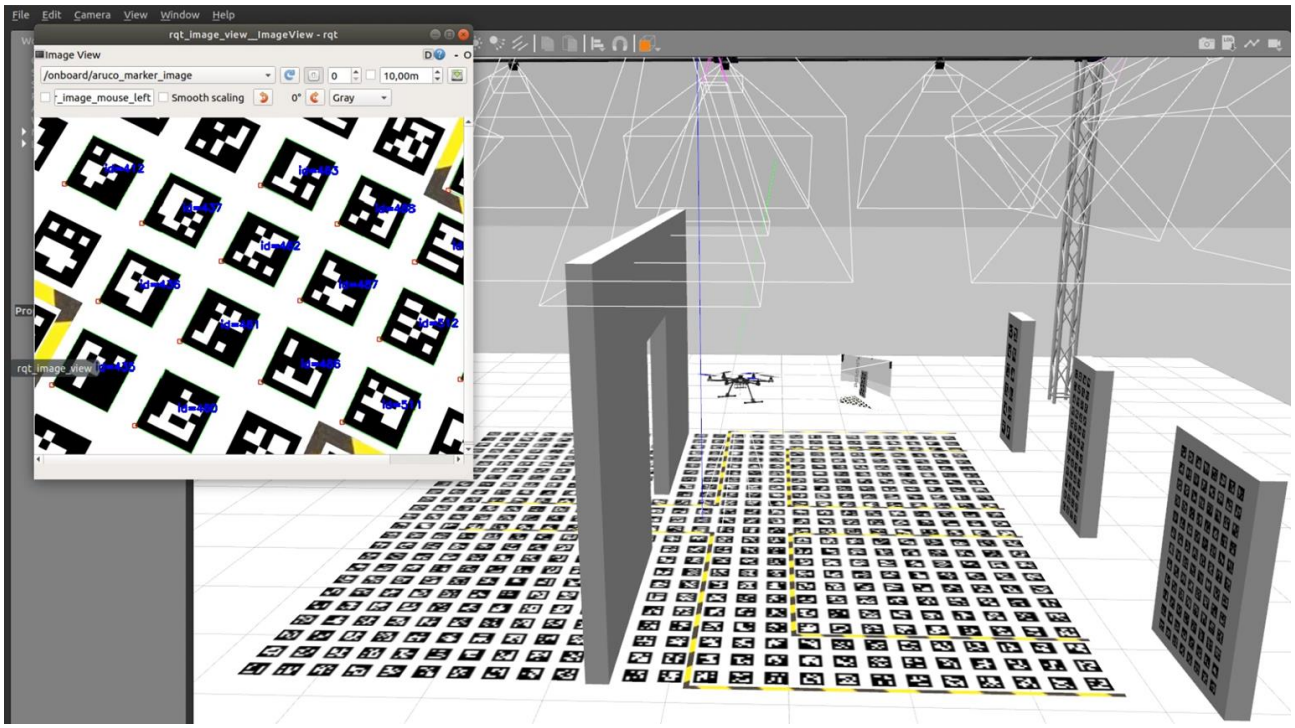


Рисунок 37 - Скріншот проведення Vision navigation тесту

На рисунку 38 старт дрону починається з дошки GPS2Vision і потім прямує до однієї зі станцій посадки.

- Ідеальний маршрут вказаний за сірою лінією.
- За чорною лінією можна відслідкувати реальний маршрут.
- Синім кольором вказані кінцеві і початкові цілі маршруту

Висновок:

З десяти тестів можна привести середні показники відхилення:

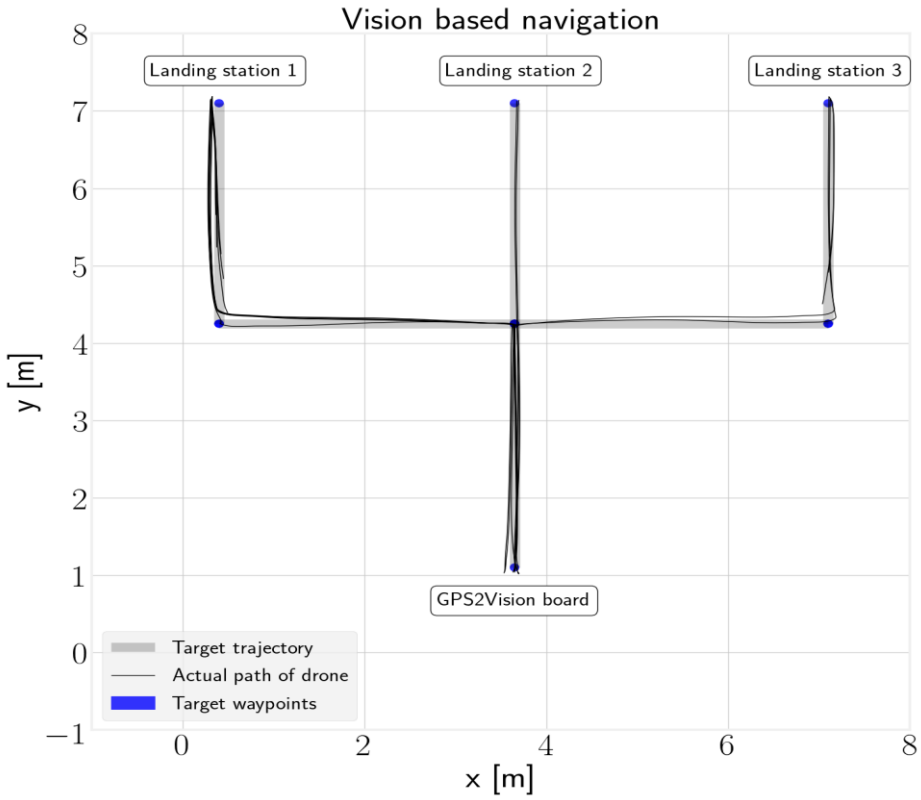


Рисунок 38, графік маршруту для 10 тестів

Таблиця 13

Вихідні результати тестування

Середня помилка aruco pos	0.02942
STD помилка aruco pos	0.03002
Середня похибка кута aruco	1.44933
STD помилка кута апроксимації	5.21816

Можна зауважити, що середні помилки та середня похибка кута є переважно невеликими, на це вказує таблиця 13. Позначу те, що на це вплинула швидкість самого дрона та кількість маркерів ArUco в зв’язку з чим була краща навігація з нижньої камери дрону.

4.4.2. Візуальна навігація за допомогою дошки ArUco без деяких маркерів з горизонтальною швидкістю 5.0 м/с

Мета: цей тест був проведений для оцінки ефективності навігації безпілотника на основі зору з використанням маркерів ArUco. Тут оцінка пози буде порівнюватися з наземною істиною дрона, щоб побачити, наскільки добре працює емуляція пози, коли дрон рухається в змодельованому середовищі.

Стартове положення: дрон буде розміщено на відстані приблизно двох метрів перед дошкою GPS2Vision.

Фінальне положення: в кінці тесту, дрон займає положення перед дошкою GPS2Vision приблизно на 2 метри.

Умови: вакуум. У цьому випадку для стрес-тестування системи була використана маркерна дошка ArUco з відсутніми маркерами.

У цьому випадку злиття датчиків, яке базується на оцінці пози на основі оцінки пози ArUco, даних IMU і барометра, буде перевірено в змодельованому середовищі за допомогою нижньої камери дрона.
Швидкість руху встановлена на рівні 5 м/с.

```
roslaunch px4 gazebo_sim_v1.0.launch worlds:=optitrack_big_board_onepattern.world drone_control_args:="vision_navigation_test" x:=-3.0 y:=0.0 headless:=false gui:=true
```

Лістинг 6 - Запуску тесту «Візуальна навігація за допомогою повної дошки ArUco з горизонтальною швидкістю 5.0 м/с

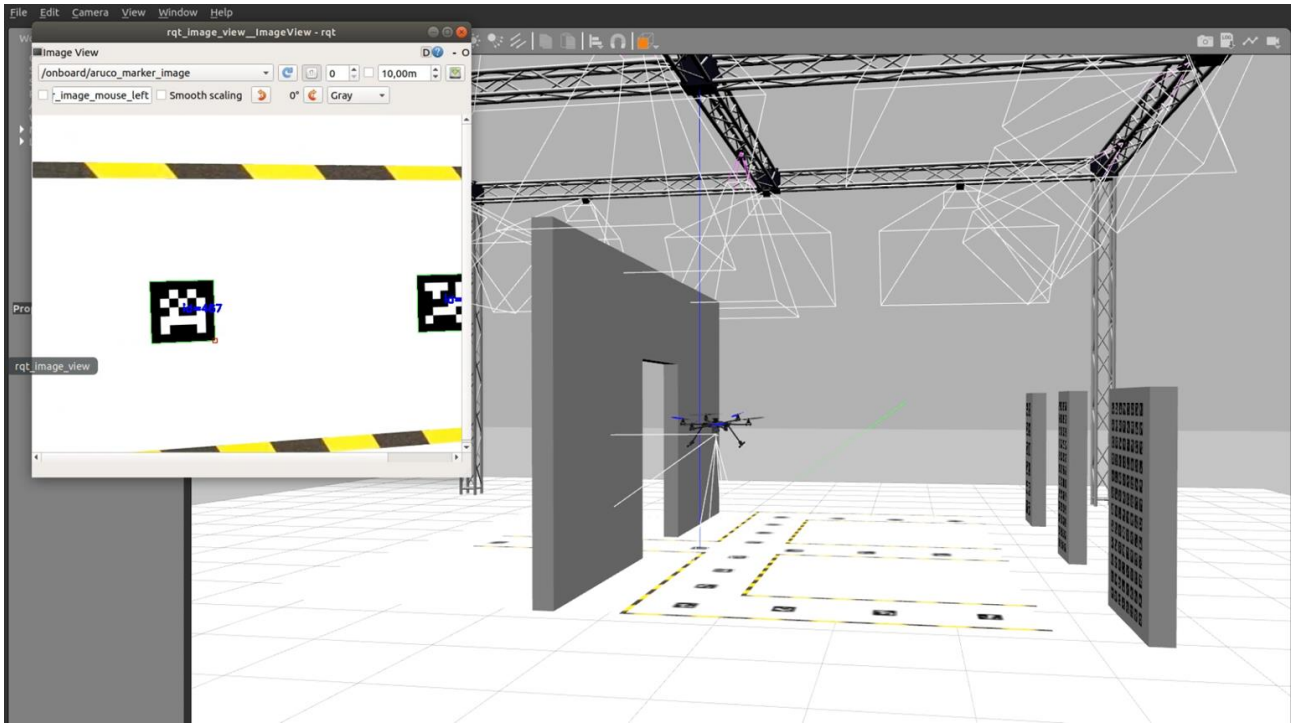


Рисунок 39 - Скріншот проведення «Візуальна навігація за допомогою дошки ArUco без деяких маркерів з горизонтальною швидкістю 5.0 м/с»

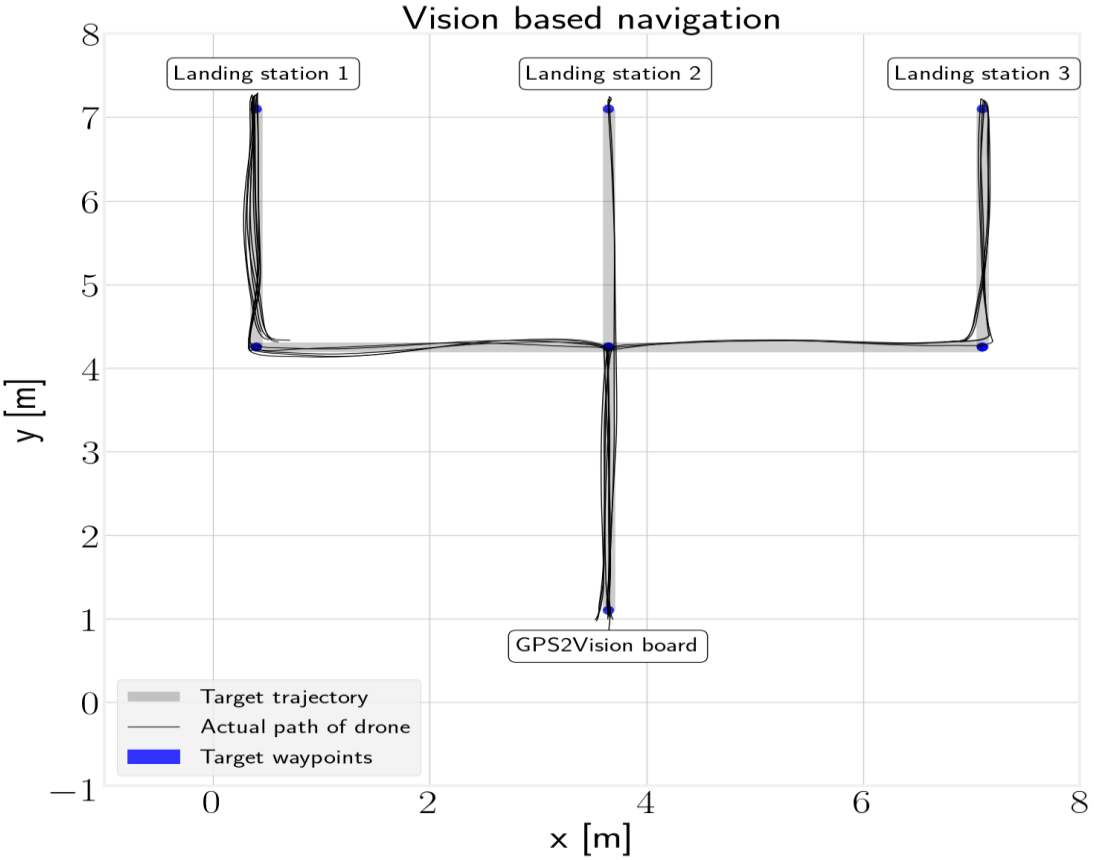


Рисунок 40 - Ілюстрація істинної траєкторії БПЛА на основі 10 тестів

Як і в тесті 4.5.1 були проведені аналогічні тести щодо істинної траєкторії БПЛА.

Як висновок, з десяти тестів можна привести середні показники відхилення:

Таблиця 14

Вихідні результати тестування

Середня помилка aruco pos	0.04628
STD помилка aruco pos	0.04184
Середня похибка кута aruco	5.51314
STD помилка кута апроксимації	16.95964

Висновок:

Можна побачити, що посередні результати апроксимації та похибки кутів є більші, чим в тесті 4.5.1. Це все зумовлено іншою силою вітру, що в свою чергу призводить до таких результатів.

З таблиці 14 видно, що результати є більш негативними порівнюючи з таблицею 13. Деякі результати є гіршими в 3 рази, що є очікувано з меншою кількістю маркерів ArUco та більшою швидкістю дрону, яка є більшою в п’ять разів.

4.4.3. Візуальна навігація за допомогою дошки ArUco без деяких маркерів зі стеком розмиття

Мета: цей тест був проведений для оцінки ефективності навігації безпілотника на основі зору з використанням маркерів ArUco які будуть розмиті з різним радіусом. У цьому стані дрон повинен знайти маркери які будуть перекриті різними радіусами стеку розмиття та відпрацювати по запланованому маршруту.

Стартове положення: дрон буде розміщено на відстані приблизно двох метрів перед дошкою GPS2Vision.

Фінальне положення: в кінці тесту, дрон займає положення перед дошкою GPS2Vision приблизно в 2 метрах.

Умови: вакуум, та вітер **1, 5 м/с.**

В цьому тесті будуть використані видозміненого маркерного напрямку Aruco як на рисунку 42. Основна ідея полягає в тому, щоб побачити як дрон буде поводитися в середовищі, якщо не бачить маркери.

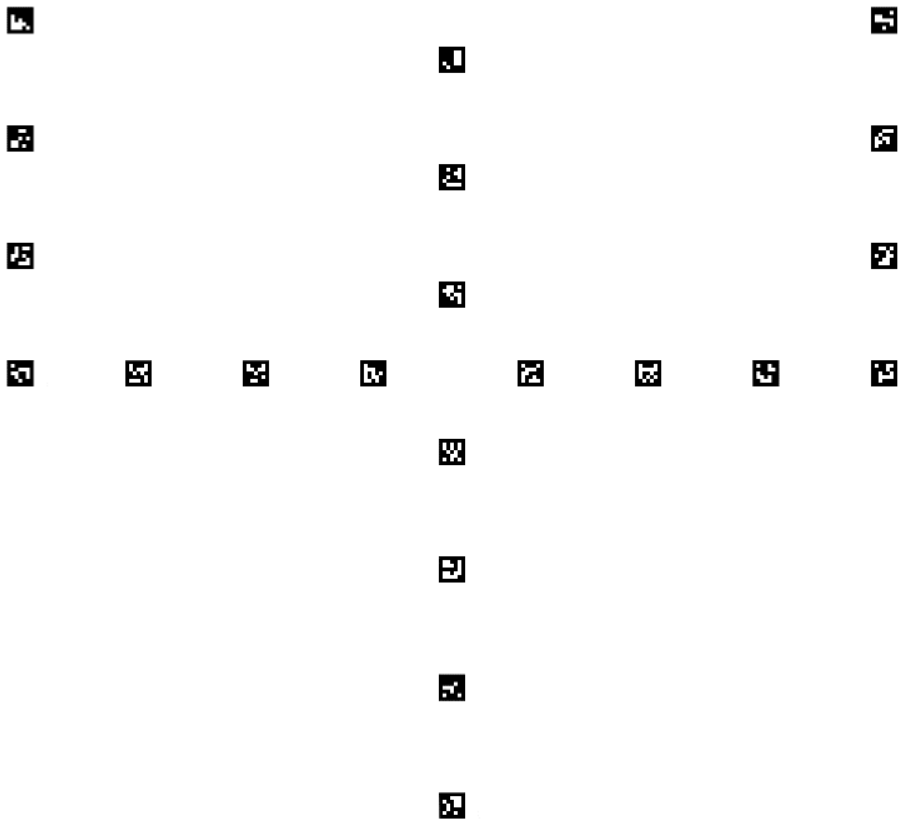


Рисунок 41, маркерний напрямок Aruco без видозмінень

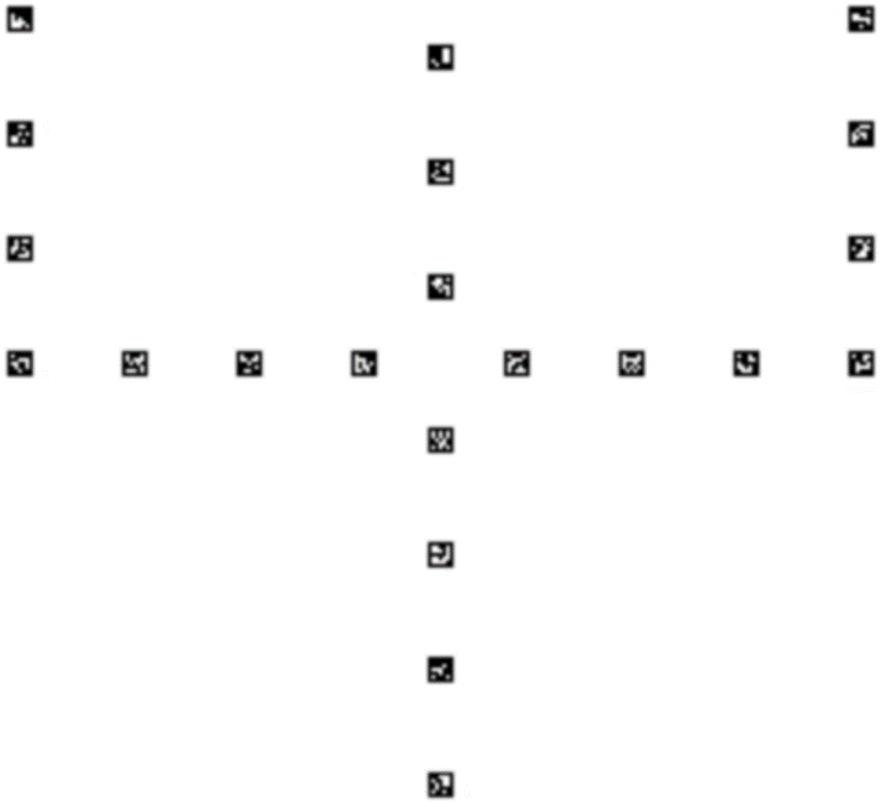


Рисунок 42 - Маркерний напрямок Aruco з видозміненням в 10 радіусів

Буде використано дві функції для всіх значень. Перша для 10 радіусів, друга для 20 радіусів для кожних із значень вітру із 10 запусків.

```
roslaunch px4 gazebo_sim_v1.0.launchworlds:=optitrack_big_board_onepattern_missing_markers.world drone_control_args:="vision_navigation_test" x:=-3.0 y:=0.0 headless:=false gui:=true
```

Лістинг 7 - Запуску тесту «Візуальна навігація за допомогою ArUco без деяких маркерів зі стеком розмиття»

Таблиця 15

Вихідні результати тестування тесту «Візуальна навігація за допомогою дошки ArUco без деяких маркерів зі стеком розмиття»

Радіус розмиття	Вітер (м/с)	Середня помилка aruco pos	STD помилка aruco pos	Середня похибка кута aruco	STD помилка кута апроксимації
10	0	0.05696	0.04879	3.14315	7.53976
12	0	0.07907	0.06797	6.43493	19.53004
14	0	0.11516	0.08719	32.73413	26.18015
16	0	0.18915	0.09077	51.88037	32.73337
18	0	0.20072	0.10601	42.27003	41.48713
20	0	0.20894	0.11031	39.50826	25.83530
10	1	0.05933	0.04997	2.91511	9.16567
12	1	0.57394	0.46245	4.07088	10.72062
14	1	2.19933	0.79676	41.59166	30.26329
16	1	2.14692	0.75695	30.90952	26.53572
18	1	2.04731	0.67101	40.03635	25.22826
20	1	2.14370	0.74299	28.60261	20.92826
10	5	5.15400	4.41656	33.25554	45.35875
20	5	5.80775	3.18839	45.62918	34.61942

Тестування в деяких випадках було фатальним, камера дрону не змогла розпізнати маркерну дошку і її за траєкторією здував вітер. Такий приклад можемо побачити для радіусу розмиття 20 та при вітрі 5 м/с, на рисунку 43.

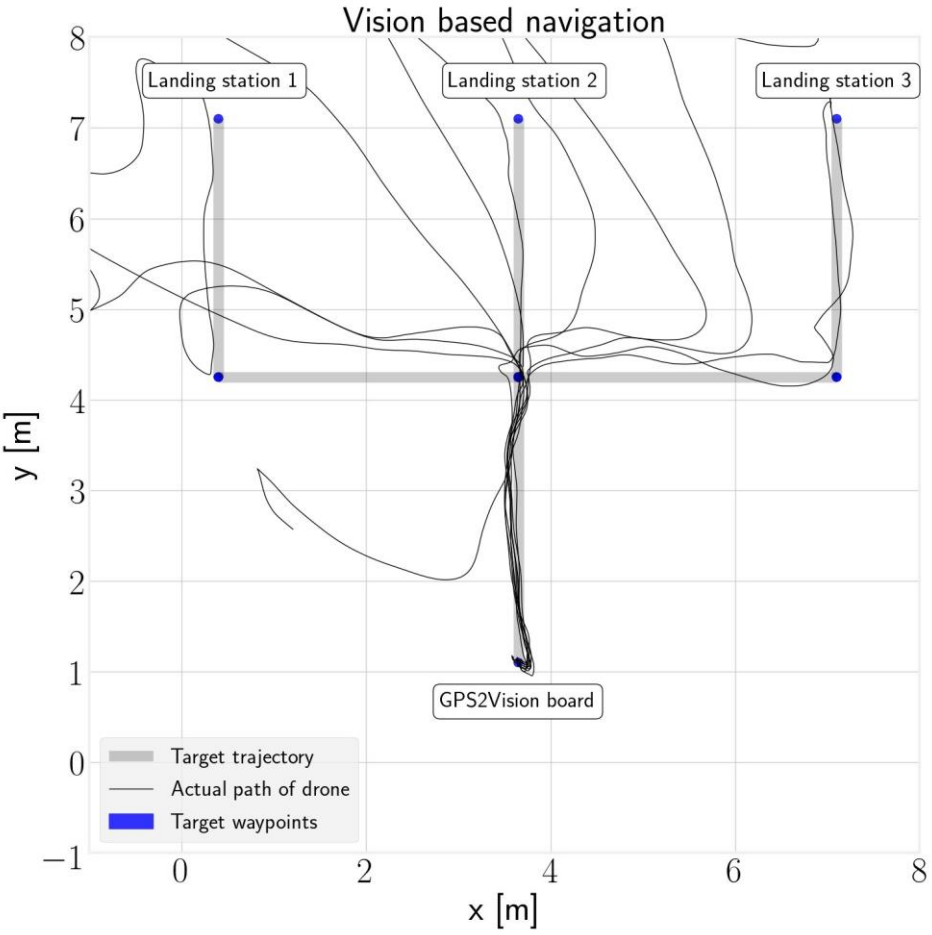


Рисунок 44 - Ілюстрація істинної траєкторії БПЛА на основі 10 тестів для радіусу розмиття 20 та при вітрі 5 м/с

Тобто, при вітрі який більше або рівний 5 м/с тестування було неуспішним, тому що дрон втрачав розмиті маркери і навігація зупинялась. Будуть приведені маркери відповідності помилки до радіусу розмиття з різною швидкістю для повного розуміння проблематики завдання.

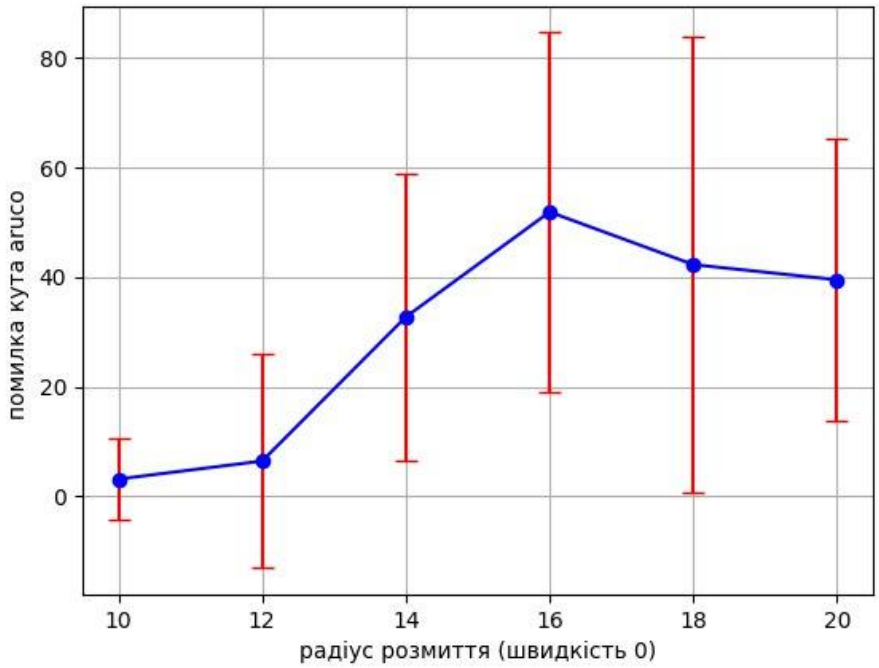


Рисунок 45 - Графік похибки aruco кута до радіусу розмиття зі швидкістю 0, з error bars до STD помилки кута апроксимації

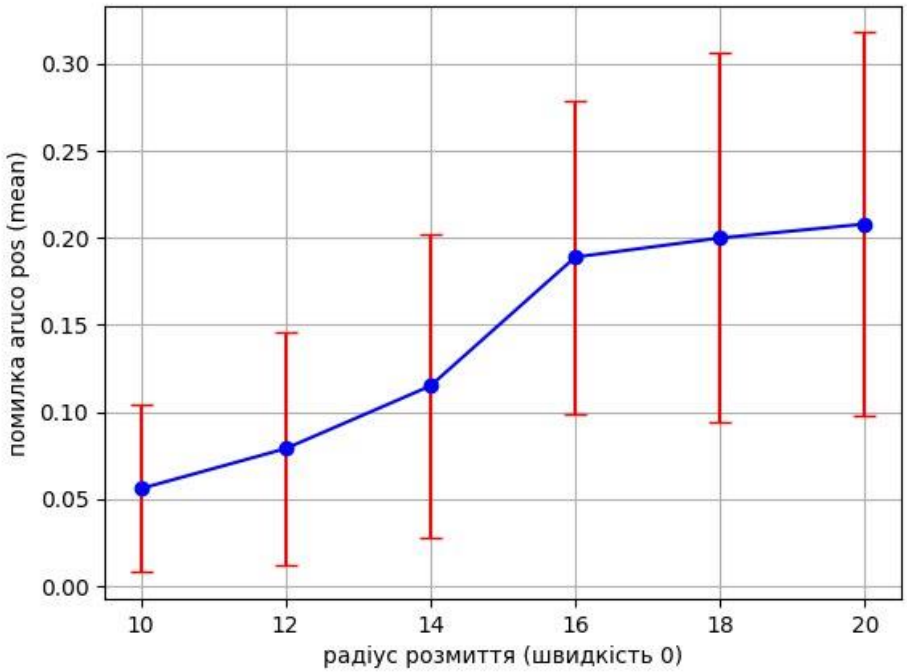


Рисунок 46 - Графік середньої помилки aruco pos до радіусу розмиття зі швидкістю 0, з error bars до STD помилка aruco pos

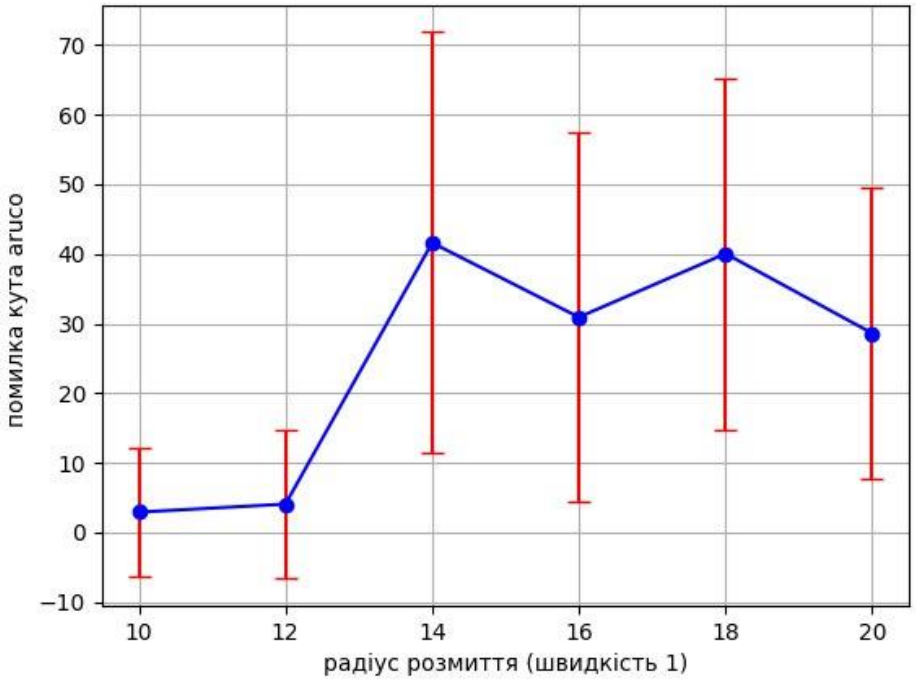


Рисунок 47 - Графік похибки aruco кута до радіусу розмиття зі швидкістю 1, з error bars до STD помилки кута апроксимації

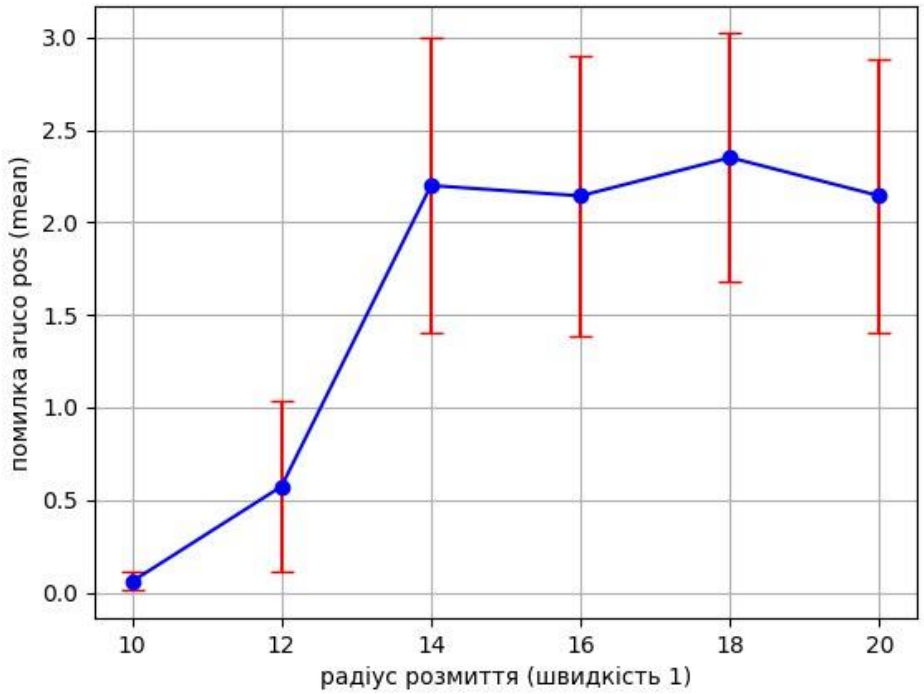


Рисунок 48 - Графік середньої помилки aruco pos до радіусу розмиття зі швидкістю 1, з error bars до STD помилка aruco pos

Висновок:

По результатам тестів добре видно залежність від гарного контуру маркера ArUco. Тобто, якщо маркер розмитий більше ніж на 10 радіусів, результат змінюється в різко негативний, візуально можна побачити на рисунку 44. Задля запобігання таких помилок, можна використати в доповнення GPS навігацію, яка була описана в тесті 4.3.2.

Середня помилка положення дрону арифметично зростає зі збільшенням радіусу розмиття, що в теорії є абсолютно логічним, але з радіусу 16, що на рисунках 46 та 48 можна побачити, що середня похибка не сильно знижується, в випадку зі швидкістю вітру, вона навіть не змінюється. В свою чергу це пов'язано з тим, що подальше розмиття вже не є логічним і пікова точка при якій **дрон оптимальніше всього може проводити навігацію зором це 12-14 радіусів розмиття**.

Але інше питання – середня помилка кута нахилу. На початкових етапах прослідкується логічне арифметичне зростання, яке можна побачити на рисунках 47 та 45 . На початкових етапах видно, що до радіусів 12-14 помилка не є занадто великою, але в подальших тестах показники є хаотичними, тобто дрон завдяки наклонами тангажу намагався знаходити втрачені Aruco маркери, що в свою чергу призвело до таких похибок.

Графіки для швидкості в 5 м/с не були приведені, тому що тести не були успішними, і показники є не точними, тому що більшість тестів були в статусі failed.

Задля покращення можливо додати більш спеціалізовані методи навігації, лідар (лазерний дальномір), завдяки якому дрон може в реальному часі вирахувати мапу оточуючого середовища, коли сканує все навколо лазером.

Розділ 5
Спосіб зменшення похибок та їх результати

Після проведених тестів та експериментів були висновки та ідеї задля покращення кінцевих результатів. Основоположником мого методу для зменшення похибок та покращення навігації це буде додання або покращення двух ключових складових:

- IMU – інерційний вимірювальний пристрій.
- PID регулятор

Мій метод полягає в тому, що якщо покращити або додати ці датчики і контролери до сенсорів – це може допомогти в покращенні навігації з GPS та сенсорними датчиками, які були встановлені до цього. Головна задача полягає в тому, щоб інтегрувати в код автоматичну зміну положення UAV пристрою в горизонтальному та вертикальному площинах. Це дасть в свою чергу стійкість проти вітру та кутами наклону тангажу. Якщо дрон буде отримувати інформацію положення дрону щодо зазначених координат. Завдяки цій інформації від датчиків, працює PID-регулятор, який він зможе керувати двигунами задля підтримання правильного положення.

PID (пропорційно-інтегрально-диференціальний) регулятор є критично важливим компонентом у навігації дронів, особливо у керуванні їх положенням, орієнтацією, швидкістю та стабільністю.

Також, буде розроблена власна система стабілізації на мові Python, завдяки інерційним вимірювальним пристроям.

Доцільно буде покращити адаптивне управління шумом, та вдосконалення моделі динаміки. Доцільність покращення АУШ полягає в тому, щоб зменшити похибку самого IMU. Тому що при вимірюваннях сенсори схильні до «шуму», тобто до випадкових або непередбачуваних відхилень, помилок, тощо.

Доцільність в додаванні вдосконаленої моделі динаміки є в тому, що середовище є агресивним, потрібно підвищити реалістичність керування до максимума. Попередньо, також буде враховуватися тепер і опір, завдяки чому

буде підвищена стабільність системи. Додавання аеродинамічного опору є важливим кроком у створенні точних, безпечних та ефективних систем керування дронами.

5.1.1. Розгляд технічної складової для покращення системи

Основні зміни будуть внесені в файл sensor_fusion.py.

Внесення покращень у код є безпосередньо викликом, тому що тестування проводиться у віртуальному середовищі gazebo, що в свою чергу ускладнює роботу вдвічі. Основні нововведення будуть спрямовані на:

- Автоматичному калібруванню для IMU.
- Адаптивному управлінню шумом (на основі змін у вимірюваннях).
- Вдосконаленні моделі динаміки (врахування аеродинамічного опору).

5.1.1.1. Автоматичне калібрування IMU

Метод calibrate_imu призначений для калібрування інерційного вимірювального блоку шляхом збору декількох вимірювань та обчислення середніх значень. Це важливий процес для забезпечення точності та надійності вимірювань IMU, особливо в системах навігації та контролю руху, в нашому випадку в безпілотних літальних апаратах.

1. Початок Калібрування: Виводиться повідомлення "Calibrating IMU...", яке повідомляє про початок процесу калібрування.
2. Ініціалізація Змінних: Створюються змінні acc_x, acc_y, acc_z для зберігання накопичених значень прискорення та gyro_x, gyro_y, gyro_z для зберігання накопичених значень гіроскопа. Вони ініціалізуються як 0.

3 Збір Даних: В циклі for відбувається збір 100 вимірювань з IMU. Значення прискорення та кутової швидкості від IMU додаються до відповідних змінних. Це робиться для кожної з осей X, Y та Z.

4 Затримка: rospy.sleep(0.01) створює коротку паузу (0.01 секунди) між кожним зчитуванням, щоб дозволити IMU стабілізуватися та надати репрезентативні дані. Обчислення Середніх Значень: Після збору даних, обчислюються середні значення для кожної осі. Це робиться шляхом ділення суми вимірювань на кількість вимірювань (num_samples).

5 Результати є оцінками зміщення для кожної осі прискорення та гіроскопа.

6 Завершення Калібрування: Виводиться повідомлення "Calibration complete.", яке сигналізує про завершення процесу калібрування.

Реалізація на мові Python виглядає ось так:

```
def calibrate_imu(self):
    print("Calibrating IMU...")
    # Збираємо декілька вимірювань для визначення середнього
    acc_x, acc_y, acc_z = 0, 0, 0
    gyro_x, gyro_y, gyro_z = 0, 0, 0
    num_samples = 100
    for _ in range(num_samples):
        acc_x += self.imu_data.linear_acceleration.x
        acc_y += self.imu_data.linear_acceleration.y
        acc_z += self.imu_data.linear_acceleration.z
        gyro_x += self.imu_data.angular_velocity.x
        gyro_y += self.imu_data.angular_velocity.y
        gyro_z += self.imu_data.angular_velocity.z
        rospy.sleep(0.01)  # коротка пауза між зчитуваннями

    # Обчислюємо середнє
    self.acc_x_offset = acc_x / num_samples
    self.acc_y_offset = acc_y / num_samples
```

```
    self.acc_z_offset = acc_z / num_samples
    self.gyro_x_offset = gyro_x / num_samples
    self.gyro_y_offset = gyro_y / num_samples
    self.gyro_z_offset = gyro_z / num_samples
    print("Calibration complete.")
```

Лістинг 8 - Частина коду реалізації автоматичного калібрування на мові Python

5.1.1.2. Адаптивне управління шумом

Функція update_noise_parameters призначена для адаптивної зміни параметрів шуму в системі навігації або трекінгу, залежно від швидкості об'єкта. Збільшення параметрів шуму при вищій швидкості може бути важливим для підвищення точності та надійності оцінки стану об'єкта, особливо в динамічних умовах, таких як швидкий політ дрона.

1. Визначення Швидкості np.linalg.norm([self.x3, self.x4, self.x5]): Цей вираз обчислює Евклідову норму (або векторну довжину) з компонент self.x3, self.x4, та self.x5. Ці компоненти можуть представляти поточні оцінки складових швидкості об'єкта по трьох осях (наприклад, X, Y, Z у тривимірному просторі). Результат цього виразу є загальною швидкістю дрону.
2. Умовна Перевірка (if speed > some_threshold): «if speed > some_threshold»: Ця умова перевіряє, чи перевищує поточна швидкість об'єкта деякий заданий поріг (some_threshold). Цей поріг визначає межу, за якої параметри шуму потрібно адаптувати.
3. Адаптація Параметрів Шуму (self.q[3:6, 3:6] *= 1.1): «self.q[3:6, 3:6] *= 1.1»: Якщо умова вище виконується (швидкість більша за поріг), то цей рядок коду модифікує певну частину матриці шуму self.q. Конкретно, він збільшує елементи матриці, які відповідають за складові швидкості (здогадуючись з контексту, це можуть бути

індекси 3, 4, та 5), множачи їх на 1.1. Це означає, що рівень шуму для цих компонентів зростає на 10%.

Реалізація на мові Python виглядає ось так:

```
def update_noise_parameters(self):
    speed = np.linalg.norm([self.x3, self.x4, self.x5]) # швидкість з останньої оцінки
    if speed > some_threshold:  # Припустимо, 'some_threshold' це певна швидкість
        self.q[3:6, 3:6] *= 1.1  # Збільшуємо шум для складових швидкості
```

Лістинг 9 - Частина коду реалізації автоматичного управління шумом на мові Python.

5.1.1.3. Додавання аеродинамічного опору

Функція iterate_x використовується для обчислення наступного стану руху об'єкта, враховуючи аеродинамічний опір. Це може бути частиною більшого моделювання руху, наприклад, в системі керування дроном, де важливо точно враховувати зовнішні сили, що впливають на динаміку польоту.

1. Додавання Аеродинамічного Опору:

drag_coefficient = 0.1: Задано коефіцієнт аеродинамічного опору. Цей коефіцієнт використовується для розрахунку сили опору, яка впливає на об'єкт.

speed = np.linalg.norm([x[3], x[4], x[5]]): Визначається швидкість об'єкта, обчислюючи Евклідову норму для трьох складових швидкості, які, імовірно, відповідають за рух у трьох вимірах (X, Y, Z).

drag_force = -drag_coefficient * speed ** 2: Розрахунок сили аеродинамічного опору. Сила опору пропорційна квадрату швидкості та вказаному коефіцієнту опору. Знак мінус вказує на те, що сила опору діє протилежно до напрямку руху.

2. Ініціалізація return масиву:

ret = np.zeros(len(x)): Створюється масив ret, ініціалізований нулями та розміром, який відповідає розміру вхідного масиву x.

3. Обчислення Впливу Опору на Рух:

«ret[3] = x[3] + drag_force * dt»: Для складової швидкості вздовж осі X (припускаючи, що x[3] це швидкість вздовж осі X) обчислюється нове значення, змінене на величину сили опору, помножену на крок часу dt. Всі подальші обчислюванян аналогічні.

4. Повернення Результату:

«return ret»: Функція повертає масив ret, який містить оновлені значення, що враховують вплив аеродинамічного опору на рух об'єкта.

Реалізація на мові Python виглядає ось так:

```
def iterate_x(self, x, dt, inputs):
    # Додаємо аеродинамічний опір
    drag_coefficient = 0.1  # Припустимо, це коефіцієнт опору
    speed = np.linalg.norm([x[3], x[4], x[5]])
    drag_force = -drag_coefficient * speed ** 2
    ret = np.zeros(len(x))

    ret[0] = x[0] + 0.6*x[3]*dt
    ret[1] = x[1] + 0.6*x[4]*dt
    ret[2] = x[2]
    ret[3] = x[3] + drag_force * dt
    ret[4] = x[4] + drag_force * dt
    ret[5] = x[5] + x[9]*dt
    ret[6] = x[6] + x[8]*dt
    ret[7] = x[7] + x[10]*dt
    ret[8] = x[8]
    ret[9] = x[9]
    ret[10] = x[10]
    ret[11] = x[11]

    return ret
```

Лістинг 10 - Частина коду реалізації додавання аеродинамічного опору на мові Python

5.1.1.4. Впровадження PID-контролера

PID-контролер є фундаментальним інструментом в сучасних системах управління, який забезпечує високу точність та стабільність у різних застосуваннях, включаючи управління дронами. Основні переваги використання PID-контролера для дронів, особливо при сильному вітрі та у процесі навігації.

Під час польоту в умовах сильного вітру, дрон може відхилятися від заданого курсу або висоти. PID-контролер динамічно коригує команди двигунів дрона, забезпечуючи стабілізацію та зберігання заданої траєкторії. Це досягається шляхом неперервного обчислення різниці між поточним станом дрона та його цільовим положенням, з подальшим виправленням цієї різниці, що і буде реалізовано в подальшому.

В реальних умовах, ефективне управління положенням дрона зменшує непотрібні рухи та корекції, що веде до оптимізації використання батареї. Таким чином, дрон може підтримувати більш тривалі часи польоту.

На основі отриманих даних, побудуємо коротку схему яка буде демонструвати знаходження PID-контролера в структурі через блок-схему на рисунок 49.

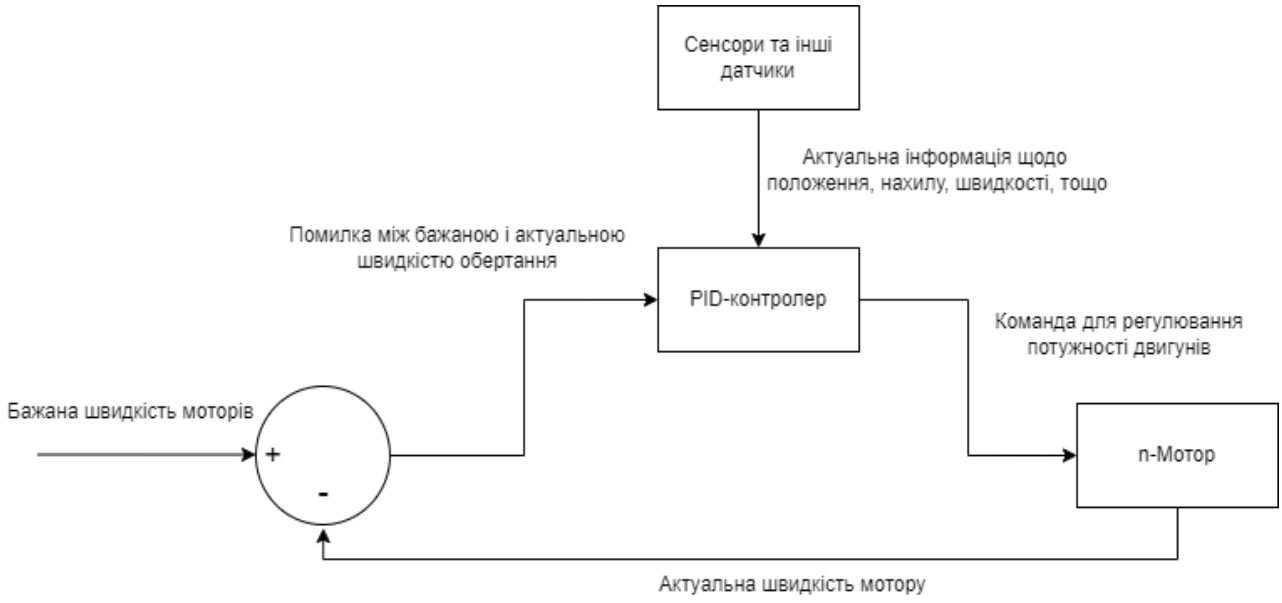


Рисунок 49 - Коротка блок-схема опису доданого PID-контролера

Класична схема контролеру, це коли вхідний сигнал – це бажане значення яке повинно бути досягнуто. Блок PID контролера, який розраховує різницю між бажаним значенням та виміряним значенням і обчислює сигнал керування на основі помилки. Потім, обов'язково існує зворотній зв'язок, який контролює PID контролер. Цей процес забезпечує неперервне регулювання та підтримку бажаного стану системи. Будову контролера можна побачити на рисунку 33.

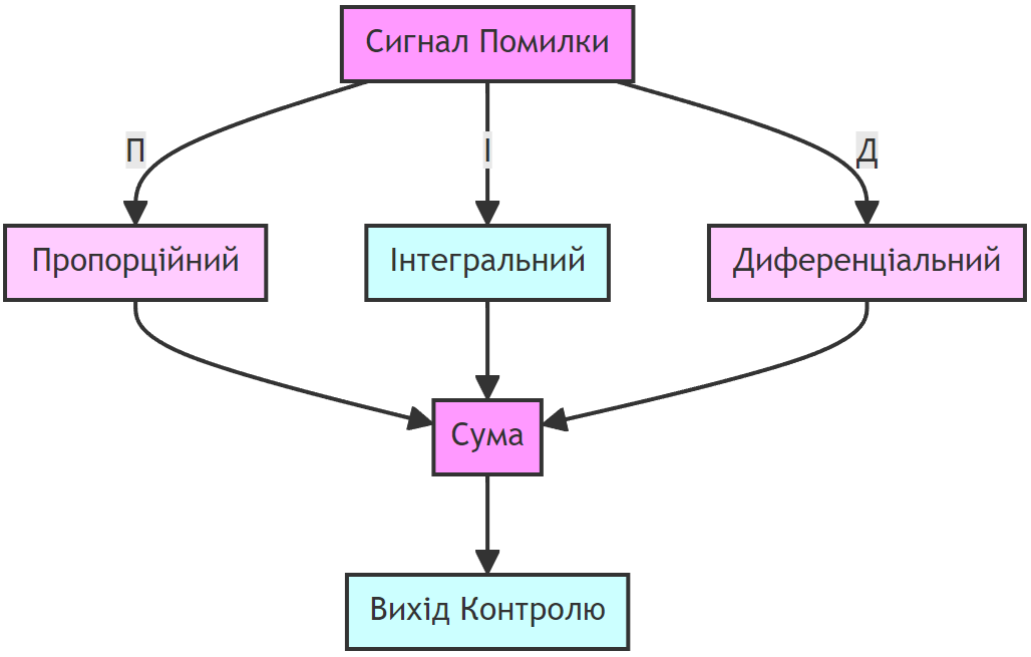


Рисунок 50 - Будова PID контролера

Як було зазначено раніше, вхідний сигнал – це сигнал помилки. Наступні гілки діляться на П (Пропорційний), І (Інтегральний) Д(Диференціальний):

- Пропорційний - Частина контролера, яка реагує пропорційно до поточної помилки.
- Інтегральний - Частина контролера, яка накопичує помилку з часом.
- Диференціальний - Частина контролера, яка реагує на швидкість зміни помилки.

Наприкінці операції є суматор, який об'єднує частини від П, І, Д, компонентів:

$PID_{out}=P_{out}+I_{out}+D_{out}$, що виходить

$PIDout = Kp \cdot e(t) + Ki \int_0^t e(\tau)d\tau + Kd \frac{de(t)}{dt}$, де

K_p - пропорційний коефіцієнт, а e(t) - поточна помилка, яка є різницею між заданим значенням та виміряним значенням, K_i - інтегральний коефіцієнт. Цей компонент допомагає усунути сталу помилку в системі, K_d - диференціальний коефіцієнт. Цей компонент допомагає стабілізувати систему, реагуючи на раптові зміни помилки.

І після створюється вихідний сигнал.

Весь код, який був написаний буде наведений в додатку Б.

5.1.2. Тестування обраних раніше методів для нової конфігурації системи.

Тестування буде проводитись з найбільш проблемними тестами, такими як:

1) Тестування 4.3.2
2) Тестування 5.3.2

Такий вибір пояснюється дуже легко: по-перше, потрібно протестувати стійкість системи до вітру при GPS навігації з новою конфігурацію і зробити висновки щодо попередніх результатів. По-друге, потрібно визначити, чи має вплив контролю дрону щодо візуальної навігації, якщо ми маємо проблеми з маркерами Aruco.

5.1.2.1. Аналіз тесту 4.3.2 з новою конфігурацією

Мета: дослідження здатності дрона утримувати позу перед маркерною дошкою GPS2Vision з внесеними змінами до конфігурації системи. Результат цього дасть результативність нових алгоритмів керування і можна буде зробити висновок чи були виправлення керування позою успішні.

Стартове положення: дрон буде тримати свою позицію приблизно на відстані 4 метрів від дошки GPS2Vision без вітру.

Кінцеве положення: тестування закінчиться в тому ж положенні в якому і починалося, тобто, приблизно на відстані в 4 метри від дошки GPS2Vision.

Умови: вакуум, вітер 1, 2, 5, 10, 15, 20, 30 м/с.

Умови запуску аналогічні лістингу №2.

Середні показники будуть вказані в середньому значенні для десяти тестів.

Таблиця 16

Вихідні результати тестування для всіх типів тестів

№ тесту	Вітер	Середня помилка aruco pos	STD помилка aruco pos	Середня похибка кута aruco	STD помилка кута апроксимації
1-10	0 м/с	0.02525	0.00952	0.34328	0.14339
11-20	1 м/с	0.03503	0.01039	0.42425	0.18661
21-30	2 м/с	0.03791	0.01364	0.51314	0.20640
31-40	5 м/с	0.03906	0.01617	0.66075	0.35651
41-50	10 м/с	0.03743	0.01791	0.84532	0.47809
51-60	20 м/с	0.04434	0.02974	1.328912	0.75608
61-70	30 м/с	помилка	помилка	помилка	помилка

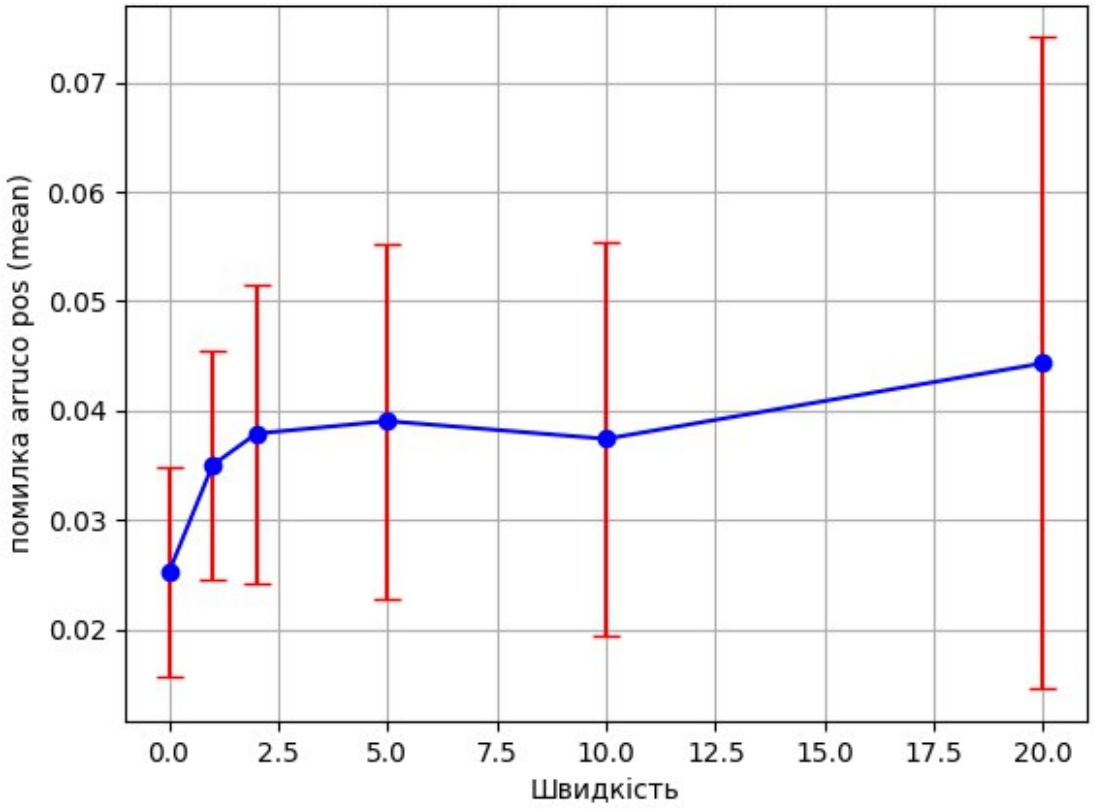


Рисунок 51 - Графік середньої помилки aruco pos до швидкості, з error bars до STD помилка aruco pos

На тестуванні 61-70 для 30 **м/с,** можна побачити помилку при всіх тестах. Це виникає в тому випадку, коли вітер перевищує відповідну норму по шкалі Бофорта, що в свою чергу дорівнює 11-ти балам – жорсткий шторм. При цих умовах похибка настільки велика, що автономна навігація не є можливою, тому що динамічно відкалібрувати IMU просто неможливо (Отримуємо помилку: check calibration on landing). В цьому випадку, дрон переходить в статус loiter (знаходитися в статусі очікування).

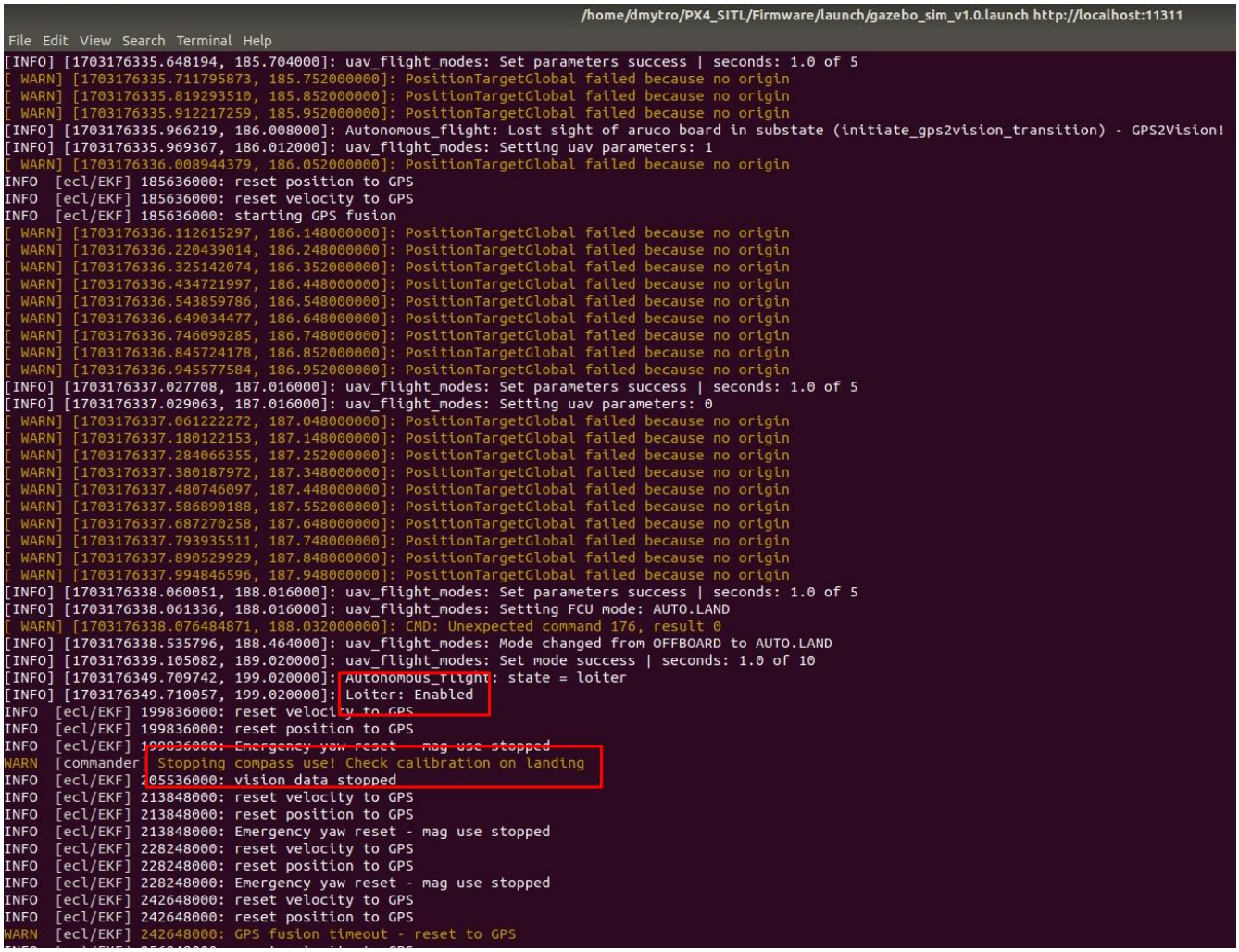


Рисунок 36 - Помилка для тестів 61-70, 30 м/с

Якщо порівняти рисунок 15 та рисунок 51 можна побачити велику зміну щодо помилки позиції дрона до 20 **м/с,** помилка зменшилася аж в 20 разів ! Що є чудовим результатом задля збереження траєкторії маршруту. Але нажаль, похибка кута до швидкості є невтішною при вітрі 20 **м/с,** що можна побачити на рисунку 52. Це трапляється в тих випадках, коли швидкість вітру занадто велика і дрон намагається стабілізувати своє положення щодо траєкторії польоту, цієї похибки уникнути ніяк не можливо.

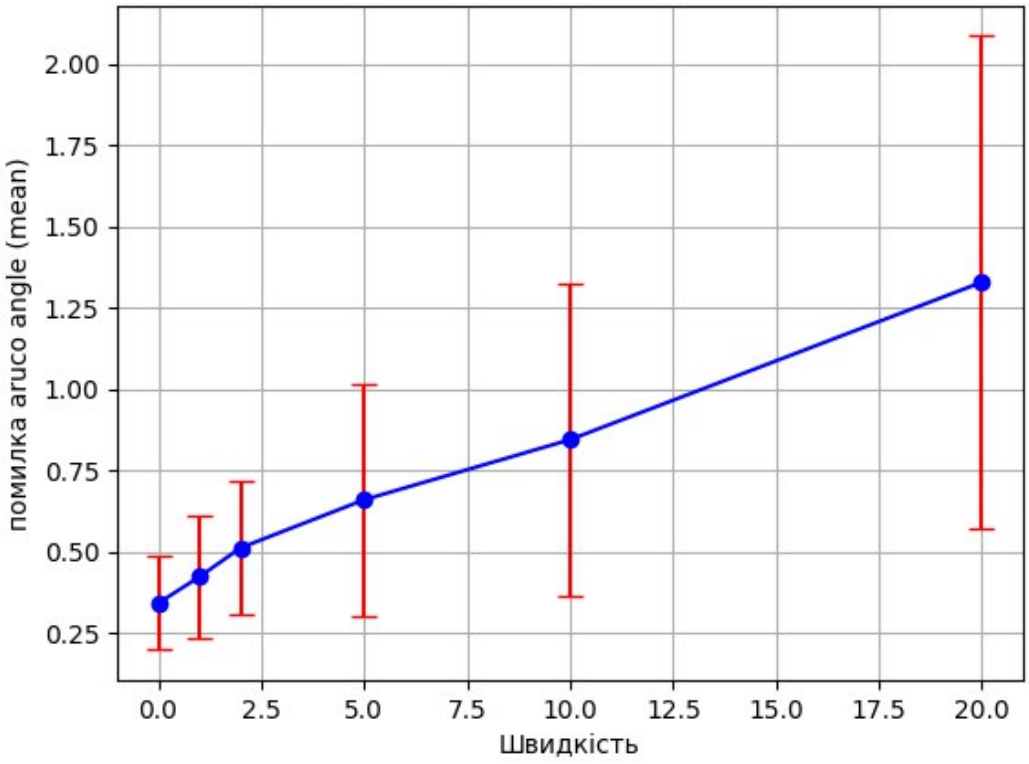


Рисунок 52 - Графік похибки aruco кута до швидкості, з error bars до STD помилки кута апроксимації

Висновок:

В попередньому тесті, вітер мав величезний вплив на позиціонування дрону в просторі. Наклони кута тангажу мали найбільші похибки.

Мета була успішно досягнута, похибка позиції була зменшена в 20 разів ! Що в свою чергу доводить успішність використання PID контролера в системі дрону. Покращення щодо динамічної калібровки IMU пристрою також були успішно використані.

5.1.2.2. Аналіз тесту 5.3.2 з новою конфігурацією

Мета: цей тест був проведений для оцінки ефективності навігації безпілотника на основі зору з використанням маркерів ArUco які будуть розмиті з різним радіусом. В результаті, дрон повинен краще утримувати позицію завдяки покращеній конфігурації та знайти всі маркери ArUco, які будуть розмиті різними радіусами стеку розмиття.

Стартове положення: дрон буде розміщено на відстані приблизно двох метрів перед дошкою GPS2Vision.

Фінальне положення: в кінці тесту, дрон займає положення перед дошкою GPS2Vision приблизно в 2 метрах.

Умови: вакуум, та вітер 1, 5 м/с.

Умови запуску аналогічні лістингу №7.

Таблиця 17

Вихідні результати тестування для всіх типів тестів

Радіус розмиття	Вітер (м/с)	Середня помилка aruco pos	STD помилка aruco pos	Середня похибка кута aruco	STD помилка кута апроксимації
10	0	0.05847	0.05182	2.71312	10.94270
12	0	0.07638	0.07321	3.72054	12.14262
14	0	0.17114	0.07900	9.29944	24.64467
16	0	0.18763	0.09077	21.65713	37.68638
18	0	0.18640	0.11876	20.53858	30.32780
20	0	0.18995	0.10240	24.96914	39.14065
10	1	0.05551	0.04888	5.31528	18.53081
12	1	0.70904	0.43670	11.10887	19.74977
14	1	1.48128	0.52817	20.40772	18.26993

16	1	2.27867	0.72563	35.11986	21.81050
18	1	2.23370	0.79092	41.99847	30.34383
20	1	5.80775	3.18839	45.62918	34.61942
10	5	5.15400	4.41656	33.25554	45.35875
20	5	5.34878	4.32011	27.88218	28.89050

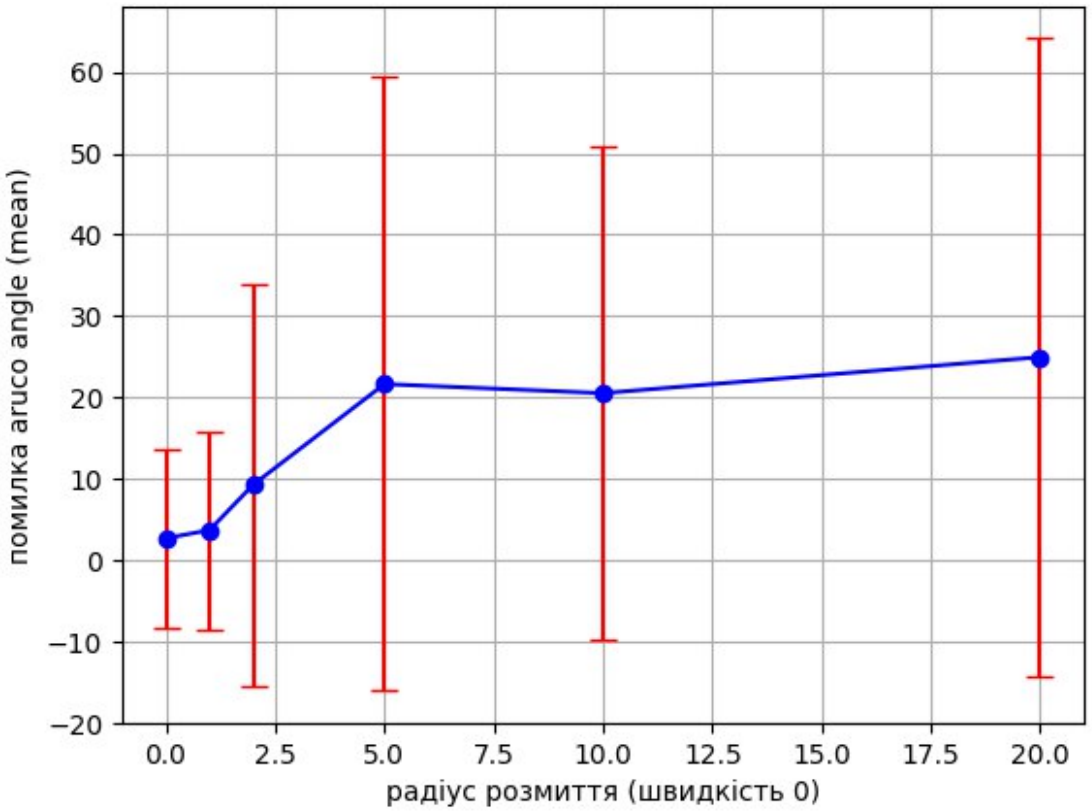


Рисунок 53 - Графік похибки aruco кута до радіусу розмиття зі швидкістю 0, з error bars до STD помилки кута апроксимації

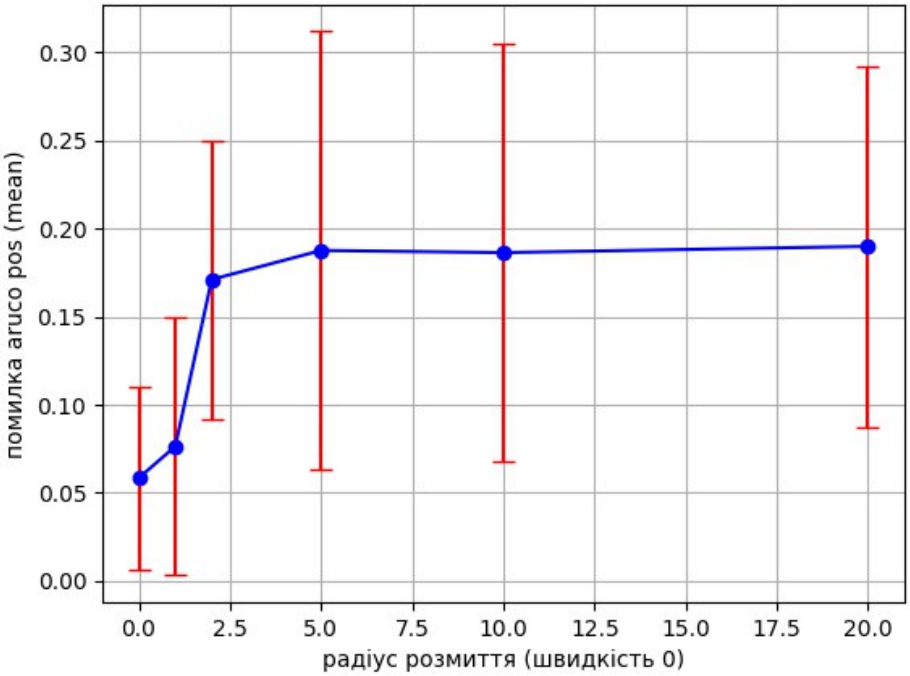


Рисунок 54 - Графік середньої помилки aruco pos до радіусу розмиття зі швидкістю 0, з error bars до STD помилка aruco pos

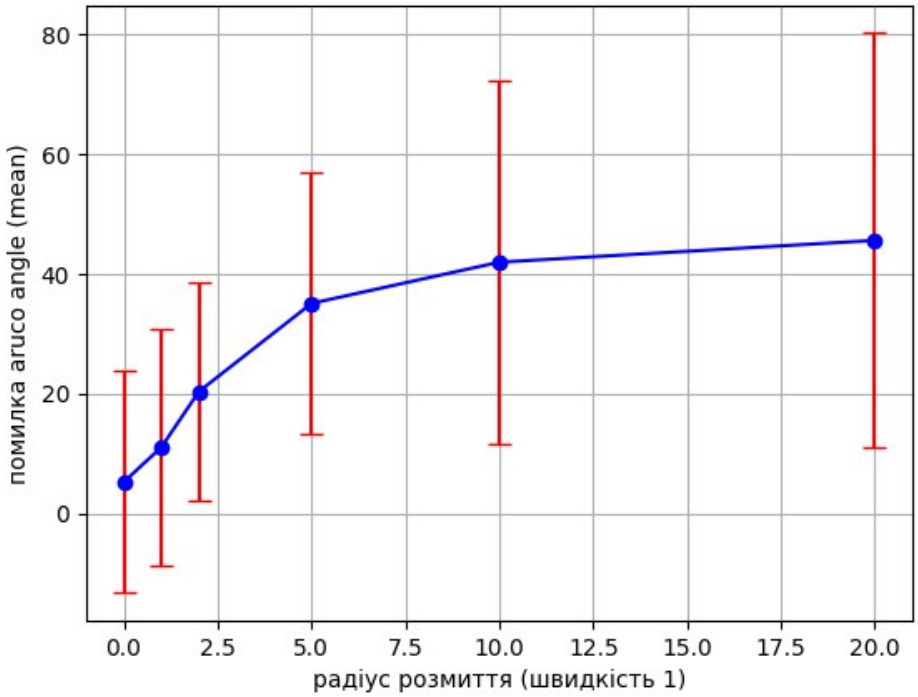


Рисунок 55 Графік похибки aruco кута до радіусу розмиття зі швидкістю 1, з error bars до STD помилки кута апроксимації

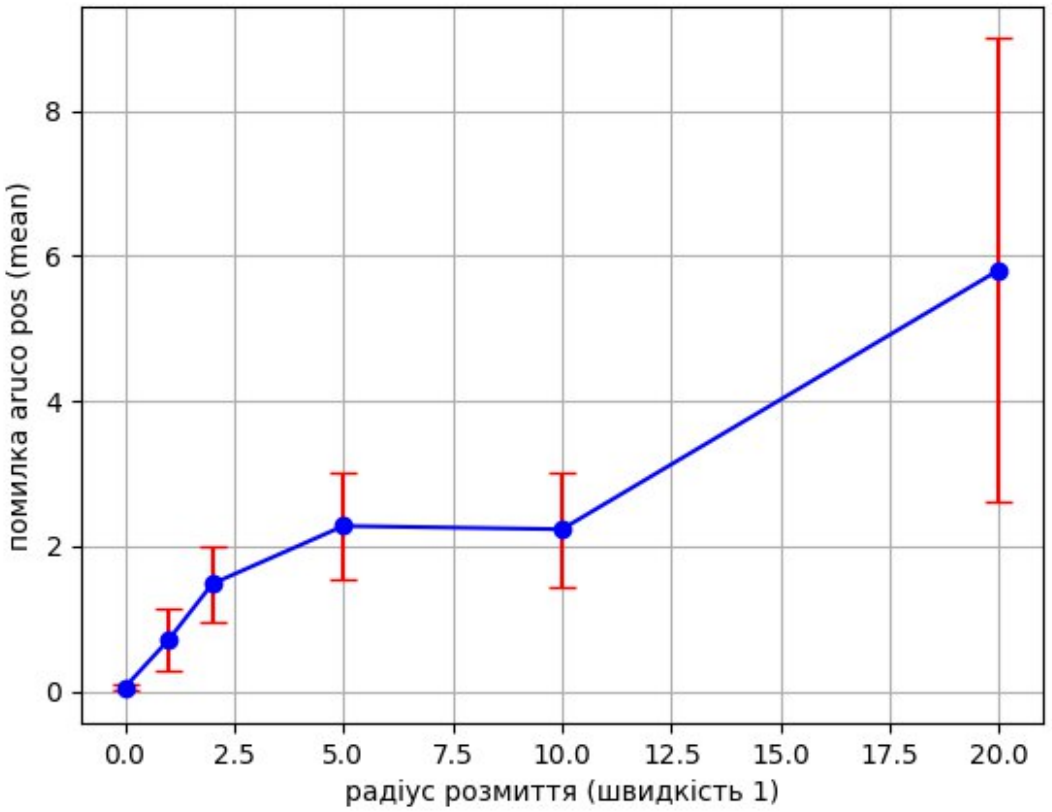


Рисунок 56 - Графік середньої помилки aruco pos до радіусу розмиття зі швидкістю 1, з error bars до STD помилка aruco pos

Висновок:

Попередні і актуальні тести не мають великої різниці, навіть в деяких випадках результати були гіршими ніж раніше. Можна відмітити покращення помилки позиціювання, але похибка кута навіть зросла в гіршу сторону.

Вочевидь, такий результат був передбачуваний в зв’язку з іншим методом навігації – зоровим, що напряму залежить від маркера ArUco який знаходиться умовно на землі. Велика похибка кута (яку можна побачити на рисунку 53 та 55) пов’язана з тим, що дрон намагається знайти маркери на землі, що і призводить до даного результату. Задля покращення цього результату, тобто, зменшення похибки, потрібно використовувати в допомогу GPS навігацію. Завдяки PID контролеру **мету не вдалося досягти**, задля цього потрібно шукати інші методи покращення навігації.

Висновки до розділу 5

Було проведено два тести, які показали наскільки нові методи управління позою були успішними. Їх успішність полягала в тому, щоб зменшити кількість похибок та стабілізувати дрон в різних площинах.

Тест 5.1.2.1, в процесі є доволі успішним завдяки доданню PID контролера, який брав участь в стабілізації системи при великих навантаженнях, тобто, навантаженнях вітру. Похибка була зменшена до 20 разів ! Але помилка aruco кута була більш неоднозначною, що в свою чергу призвело також до невеликих коливань тангажу при вирівнюванні свого положення щодо горизонтальної осі, значення помилки є не значним. Як висновок, мета була досягнута.

Результати тесту 5.1.2.2 є гіршими, щодо попередніх результатів. Похибка середньої позиції в просторі мала позитивні результати, але не настільки, щоб вважати цей тест успішним. Графіки похибки aruco кута в деяких випадках збільшилися в 1.5 рази, що несе за собою негативний результат. Як висновок, мета не була досягнута.

Основною різницею між попередніми тестами – методи навігації. Тобто, тест 5.1.2.1 – мав GPS навігацію, а тест 5.1.2.2 – візуальну. Тобто, все полягає в тому, що положення та навігації квадрокоптера регулюється різними чинниками: GPS приймачем та ArUco маркерами відповідно. До другого тесту це додає ще один незалежний чинник який абсолютно впливає на середню похибку при тестуваннях, цим і пояснюється різниця між ними.

Такі зміни як: інтеграція PID-контролера, зміна аеродинамічного опору, автоматичне калібрування IMU та адаптивне керування шумом, позитивно вплинули на результати похибок для навігації GPS методом, що в свою чергу доводить. Що мета роботи досягнута.

Розділ 6
Розробка стартап-проекту

Ринок навігації дронів у агресивному середовищі є порівняно новим, але швидко розвивається, зокрема завдяки прогресу в області автономних технологій і штучного інтелекту. Ринок охоплює застосування від аварійно-рятувальних операцій до військових місій. Потенціал цього ринку ілюструють численні дослідження та розробки, але варто зазначити, що існуючі рішення ще не вичерпують усі можливості технології. Основною відмінністю та ідеєю запропонованого проекту є впровадження новітніх технологій для підвищення ефективності навігації в складних та змінних умовах. Особливий акцент зроблено на адаптивність до агресивних середовищ, включаючи забезпечення стійкості до екстремальних погодних умов та електронних перешкод. На даному етапі необхідно розробити стратегію для подальшого розвитку та планування дій, аби створити систему, яка буде високоефективною та конкурентоспроможною. Далі в цьому розділі будуть представлені ключові маркетингові, організаційні, фінансово-економічні та комерційні аспекти запропонованої системи, а також оцінка потенційних ризиків і можливостей на цьому ринку.

6.1.1. Маркетинговий аналіз стартап-проекту

В рамках цього проекту, основна ідея полягає в розробці передової системи навігації для дронів, призначеної для використання в агресивних середовищах. Система буде інтегрувати новітні технології для точної навігації, ухилення від перешкод та автономного прийняття рішень в умовах високого ризику. Ключовою особливістю є здатність системи адаптуватися до різноманітних агресивних умов, таких як екстремальні погодні умови, електромагнітні перешкоди, або навіть ворожі дії. Система буде забезпечувати збір, обробку та аналіз даних про оточення в реальному часі, використовуючи сенсори. На основі отриманої інформації, система зможе автоматично коригувати маршрути дронів, забезпечуючи оптимальну навігацію та безпеку місії.

Таблиця 18

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка передової технологічної системи для навігації безпілотних літальних апаратів (дронів) в умовах, які характеризуються високим рівнем агресивності та непередбачуваності	Наукові дослідження та екологічний моніторинг	Дрони можуть використовуватися для збору даних у важкодоступних або небезпечних місцях, таких як вулкани, льодовики, тропічні ліси, або морські глибини.
	Аварійно-рятувальні операції	Моделювання можливих збоїв під час роботи мережі, дослідження роботи мережі у випадку невідкладних збоїв та визначення альтернативних шляхів руху трафіку у таких випадках
	Військові та оборонні операції	В умовах бойових дій дрони можуть виконувати завдання з доставки важливих вантажів, виявлення та нейтралізації ворожих об'єктів. Система навігації в агресивному середовищі забезпечить високу точність та надійність у виконанні цих місій.

Проведемо аналіз технічних переваг ідеї та визначимо, чим ідея відрізняється від існуючих аналогів та замінників, проведемо порівняльний аналіз показників: для власної ідеї визначимо показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні).

Таблиця 19

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Проекти				W	N	S
		Мій	1	2	3			
1.	Висока точність навігації	+	+	+	-		+	
2.	Адаптація до агресивних умов	+	-	-	-			+
3.	Інтеграція з різними системами та платформами	+	+	-	+		+	
4.	Модульність і гнучкість конфігурації	+	+	-	+		+	
5.	Забезпечення безпеки та цілісності даних	-	+	+	+	+		
6.	Ефективність у використанні енергії	-	-	+	-		+	

7.	Використання PID контролерів для автономної навігації	+	+	-	-			+
8.	Вартість розробки та впровадження	+	-	-	+			+
9.	Потенціал комерційного застосування	-	-	+	+	+		

Визначення переліку сильних, слабких та нейтральних характеристик і властивостей ідеї потенційного товару, як у Вашому випадку – навігаційної системи для дронів в агресивному середовищі, є ключовим для формування його конкурентоспроможності:

Сильні Сторони:

- Підтримка Роботи в Високонавантажених Умовах: продукт здатний ефективно функціонувати в складних умовах, які включають екстремальні погодні умови та електромагнітні перешкоди.
- Модульність та Гнучкість: Продукт пропонує гнучкість у конфігурації, дозволяючи користувачам адаптувати систему під свої специфічні потреби. Цілісність та Надійність Даних: Ваша система забезпечує високий рівень точності та надійності в обробці та передачі даних.

Слабкі Сторони:

- Недостатній Рівень Безпеки: Однією з основних проблем є потенційно недостатній рівень безпеки, що може включати вразливості до РЕБ.

- Відсутність Віртуалізації Вузлів: Неіснування віртуалізації може обмежити гнучкість системи та її спроможність до швидкої адаптації до змінних умов.

Нейтральні Характеристики:

- Масштабованість: Продукт може бути легко адаптований для різних масштабів використання, але це не є унікальною перевагою порівняно з конкурентами.
- Інтеграція з Іншими Системами: Хоча продукт може інтегруватися з іншими системами, ця характеристика не виділяє його серед інших рішень на ринку.

6.1.2. Технологічний аудит ідеї проекту

Таблиця 20

Технології аудиту ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Автономна навігація дронів	Алгоритми штучного інтелекту, PID контролер	Є в наявності	Доступні, деякі платні
2.	Обробка даних з сенсорів дрона	Модулі машинного зору, сенсори дрона	Є в наявності	Доступні, деякі платні
3.	Захист даних та керування	Криптографічні системи, захищені протоколи	Є в наявності	Доступні, деякі платні
4.	Адаптивність до змін середовища	Системи машинного навчання, датчики середовища	Є в наявності	Доступні, деякі платні

5.	Інтеграція з іншими системами (напр. GPS)	API інтеграції, модулі GPS	Є в наявності	Доступні
6.	Пульт дистанційного керування	Інтерфейс користувача, системи управління	Є в наявності	Доступні
7.	Система уникнення перешкод	Алгоритми обробки зображень, лідари	Є в наявності	Доступні, деякі платні

Згідно з описаними технологіями, необхідними для реалізації проекту навігації дронів в агресивному середовищі, створення і розробка такої системи є цілком можливим. Більшість необхідних технологій, включаючи сенсори дронів, криптографічні системи та інші, є в наявності та багато з них доступні безкоштовно або за помірну плату. Однак, деякі елементи, такі як системи машинного навчання для адаптації до змін середовища, можуть потребувати додаткової розробки та налаштування. Доступність цих технологій може варіюватися в залежності від доступу до великих даних, наявності апаратних ресурсів для обробки даних та наявності кваліфікованих спеціалістів. Ключовим аспектом є використання передових алгоритмів та інноваційних підходів для забезпечення високого рівня точності, безпеки та адаптивності системи в складних умовах.

6.1.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей та загроз для стартап-проекту " можна провести, аналізуючи ринкове середовище та потреби потенційних клієнтів. Нижче наведено попередню характеристику потенційного ринку для цього стартап-проекту.

Таблиця 21

Попередня Характеристика Потенційного Ринку Стартап-Проекту

№	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних гравців	Декілька компаній-лідерів, конкуренція помірна
2.	Загальний обсяг продаж, грн/ум.од.	Значний, з потенціалом зростання
3.	Динаміка ринку (якісна оцінка)	Позитивна, зі зростанням інтересу до дронів
4.	Наявність обмежень для входу (характер обмежень)	Технічні та фінансові бар'єри, високі стандарти
5.	Специфічні вимоги до стандартизації та сертифікації	Стандарти безпеки та навігації
6.	Середня норма рентабельності в галузі або по ринку, %	Висока, з огляду на технологічні інновації

За результатами аналізу, ринок навігації дронів є привабливим для входження, завдяки високому потенціалу рентабельності, зростаючому інтересу до дронів та обмеженій кількості основних гравців.

Таблиця 22

Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1.	Високий Рівень Конкуренції	Наявність великої кількості розробників дронів	Фокус на унікальних рішеннях, відрізнятися від конкурентів

2.	Технологічні Обмеження	Обмеження в сенсорах, батареях	Інвестиції у дослідження для подолання цих бар'єрів
3.	Недостатній рівень конкуренто-спроможності	Рішення конкурентів задовольняють базові потреби клієнтів	Вдосконалення цінової політики за покращення якості надання послуг, проведення маркетингової компанії за ключовими перевагами продукту
4.	Неоднозначна економічна ситуація	Відсутність фінансування, недостатня кількість інвестицій	Пошук додаткових шляхів надходження коштів, заощадження витрат
5.	Зміни у Законодавстві	Регулювання використання дронів	Стеження за законодавчими змінами, лобіювання інтересів

Таблиця 23

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1.	Зростання Інтересу до Дронів	Використання дронів у різних сферах	Розробка унікальних рішень, адаптованих під ринок
2.	Технологічні Інновації	Постійне покращення технологій дронів	Інвестиції у дослідження та розробку

3.	Співпраця з Великими Гравцями Ринку	Партнерство з провідними компаніями	Створення спільних проектів

Таблиця 24

Характеристика Потенційних Клієнтів Стартап-Проекту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги до споживачів продукту
1.	Ефективна навігація в складних умовах	Військові підрозділи, рятувальні служби	Різні тактичні потреби, відмінності в умовах використання	Надійність, точність, стійкість до перешкод
2.	Моніторинг та дослідження недоступних зон	Наукові установи, екологічні організації	Різні вимоги до точності даних, специфіка досліджень	Довговічність, точність сенсорів, надійність даних
3.	Комерційне використання	Приватні компанії, виробничі підприємства	Різні бізнес-моделі та потреби у навігації	Ефективність, скорочення витрат, гнучкість системи

Тепер розглянемо фактори, які сприяють ринковому впровадженню проекту, та фактори, які йому можуть перешкоджати.

Таблиця 25

Фактори Загроз Ринковому Впровадженню Проекту

№	Фактор	Зміст загрози	Можлива реакція компанії
1.	Технічні Обмеження	Обмеження можливостей технологій	Інвестиції у дослідження і розвиток
2.	Регуляторні Бар'єри	Законодавчі обмеження використання дронів	Активна робота з регуляторами, лобіювання інтересів
3.	Висока Конкуренція	Наявність сильних гравців на ринку	Розробка конкурентних переваг, інноваці

Таблиця 26

Фактори Сприяння Ринковому Впровадженню Проекту

№	Фактор	Зміст можливості	Можлива реакція компанії
1.	Інноваційний Підхід	Унікальні технологічні рішення	Акцент на інноваціях у маркетингу та продажах
2.	Високий Попит	Зростаючий інтерес до передових технологій дронів	Зосередження на високотехнологічних рішеннях
3.	Співпраця з Індустріальними Партнерами	Можливість розширення ринкових горизонтів	Пошук та розвиток стратегічних партнерств

За результатами цього аналізу можна розробити стратегію виходу на ринок, враховуючи існуючі можливості та загрози, та зосередити зусилля на розвитку унікальних характеристик продукту, які задовольнятимуть потреби потенційних клієнтів.

6.1.4. Розробка ринкової стратегії стартап-проекту

Розробка ринкової стратегії для стартап-проекту вимагає визначення стратегії охоплення ринку, включаючи опис цільових груп потенційних споживачів та вибір відповідних маркетингових підходів.

Таблиця 27

Вибір Цільових Груп Потенційних Споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу в сегмент
1.	Військові підрозділи	Висока	Високий	Висока	Проста
2.	Аварійно-рятувальні служби	Висока	Високий	Середня	Проста
3.	Наукові установи	Середня	Середній	Низька	Складна
4.	Екологічні організації	Середня	Середній	Середня	Проста

5.	Приватні компанії (наприклад, аграрний сектор)	Середня	Середній	Висока	Проста

Обрані цільові групи: Аварійно-рятувальні служби, екологічні організації, приватні компанії в аграрному секторі. Має бути використана стратегія диференційованого маркетингу.

Ринкова Стратегія

- Диференційований Маркетинг: Оскільки компанія працює одночасно із кількома сегментами, стратегія передбачає розробку унікальних пропозицій для кожної цільової групи. Наприклад, акцент на безпеці та надійності для аварійно-рятувальних служб, адаптація до специфічних умов досліджень для наукових установ.

Реалізація Стратегії

- Маркетингові Кампанії: Організація спеціалізованих маркетингових кампаній для кожної цільової групи. Використання специфічних каналів комунікації, які є найбільш ефективними для кожної групи.
- Партнерства: Налагодження партнерських відносин з організаціями, які мають вплив у цільових групах, наприклад, з регіональними або національними аварійно-рятувальними службами.
- Технічна Підтримка та Навчання: Надання послуг підтримки та навчання для споживачів, щоб забезпечити легкість впровадження та ефективне використання продукту.

Ця стратегія дозволить ефективно впроваджувати продукт в різних сегментах ринку, враховуючи специфічні потреби кожної цільової групи, і тим

самим забезпечити високу ринкову присутність та конкурентоспроможність продукту.

6.1.5. Розробка маркетингової програми стартап-проекту

Розробка маркетингової програми для стартап-проекту включає формування маркетингової концепції продукту, його ключових переваг, а також розробку маркетингових стратегій та каналів збуту. Нижче наведено декілька таблиць, які описують ці аспекти.

Таблиця 28

Визначення Ключових Переваг Концепції Потенційного Товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Ефективна навігація в складних умовах	Автономна система з високою точністю навігації	Точність, надійність, адаптація до екстремальних умов
2.	Надійність та тривалість роботи	Довговічність та стабільність у складних умовах	Енергоефективність, стійкість до фізичних впливів
3.	Інтеграція з іншими системами	Можливість легкої інтеграції з існуючими системами	Гнучкість, сумісність з різними платформами

Далі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

Таблиця 29

Опис Трьох Рівнів Моделі Товару

Рівні товару	*Сутність та складові*
I. Товар за задумом	Автономна система навігації для дронів в агресивних умовах
II. Товар у реальному виконанні	Властивості: висока точність, надійність, адаптивність. Розмір: в залежності від моделі дрона
III. Товар із підкріпленням	Ліцензія на використання, підтримка та оновлення програмного забезпечення

Таблиця 30

Визначення Меж Встановлення Ціни

№	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі становлення ціни на товар/послугу
1.	1500$ - 3000$/міс.	2000$ - 4000$/міс.	>40000$/міс	2000$ - 3500$/міс.

Таблиця 31

Формування Системи Збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1.	Закупівля комплексних технічних рішень	Консультації, навчання, технічна підтримка	Прямий збут через власний веб-сайт, партнерства з постачальниками дронів	Інтернет, тендерні торги

Таблиця 32

Концепція Маркетингових Комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1.	Технічно освічені споживач	Професійні форуми, соціальні мережі, вебінари	Інновації, безпека, надійність	Інформування про унікальні особливості та переваги продукту	Демонстрація реальних випадків використання, відгуки клієнтів

Висновок до розділу 6:

Шостий розділ присвячений розробці стартап-проекту "навігація дронів в агресивному середовищі", який базується на інноваційній системі автономної навігації для безпілотних літальних апаратів. У рамках розробки цього стартап-проекту було виконано наступні завдання:

Загальний Опис Ідеї Стартап-Проекту:

- Описано функціонал системи, що включає точну навігацію, адаптацію до різних умов та інтеграцію з іншими системами.
- Визначено основних конкурентів та проведено порівняльний аналіз існуючих продуктів на ринку, щоб визначити переваги розроблюваного продукту.

Аналіз Ринкових Можливостей:

- Проаналізовано ринкові тренди та можливості для успішного виходу на ринок.
- Виявлено ключові можливості та загрози для стартапу та розроблено стратегії щодо їх мінімізації або використання.

Розробка Ринкової Стратегії:

- Проведено порівняння з конкурентами та встановлено план розвитку продукту.
- Визначено цільові групи та методи просування проекту на ринку, включаючи онлайн та офлайн канали.

Маркетингова Програма:

- Розроблено детальну маркетингову програму, яка включає стратегії просування, вибір каналів збуту та визначення пріоритетних цільових груп.

- Сформульовано ключові маркетингові повідомлення для залучення нових клієнтів і збільшення клієнтської бази.

У результаті, було розроблено комплексний стартап-проект для запуску розробленої системи навігації дронів, готовий до виходу на ринок. Цей проект включає всі ключові аспекти від розробки продукту до маркетингової стратегії, що дозволить ефективно конкурувати та відповідати потребам цільових ринкових сегментів.

Список використаних джерел

1. Gazebo. Gazebo documentation [Електронний ресурс] / Gazebo. – 2023. – Режим доступу до ресурсу: https://gazebosim.org/docs.

2. MORSE. MORSE, the Modular OpenRobots Simulation Engine [Електронний ресурс] / MORSE. – 2023. – Режим доступу до ресурсу: https://www.openrobots.org/wiki/morse/.

3. ROS. ROS Documentation [Електронний ресурс] / ROS. – 2017. – Режим доступу до ресурсу: https://www.ros.org/news/2012/03/announcement-of-hector-quadrotor-stack.html.

4. Єфімов В.О. Вплив зовнішніх на ємність зв\'язку БПЛА : дис. канд. : 172 / Єфімов В.О.. – Запоріжжя, 2018. – 88 с.

5. Дудуш, А. С., et al. "Сучасний стан та проблеми протидії маловисотним, низькошвидкісним та малорозмірним БПЛА." *Сучасні інформаційні технології у сфері безпеки та оборони* 1 (2018): 121-131.

6. Мельник, О. М., & Корякін, К. С. (2021). Сучасні шляхи підвищення стандартів точності та надійності супутникових навігаційних систем. *ВЧЕНІ ЗАПИСКИ*, 62021225.

7. Повх, І. В. (2023). Комп’ютерна система підтримки проектування алгоритмів управління БПЛА.

8. Таранов, Д. (2021). Система автоматизованого проектування систем візуальної навігації.

9. GAZEBO. ROS with Gazebo Classic Simulation [Електронний ресурс] / GAZEBO. – 2023. – Режим доступу до ресурсу: https://docs.px4.io/main/en/simulation/ros_interface.html.

10. Ubuntu 18.04.5 LTS (Bionic Beaver) 64-bit server operating system. Available at: https://cdimage.ubuntu. com/releases/18.04/release/.

11. Technische Universität Darmstadt. Hector localization. Available at: https://github.com/tu-darmstadt-ros-pkg/hector_localization.

12. Technische Universität Darmstadt. Hector Quadrotor. Available at: https://github.com/tu- darmstadt-ros-pkg/hector_quadrotor.

13. Survey of Important Issues in UAV Communication Networks. Available at: https://ieeexplore.ieee.org/abstract/document/7317490.

14. Introduction to UAV systems. Available at: https://books.google.com/books?hl=uk&lr=&id=s8Z6EAAAQBAJ&oi=fnd&pg=PA17&dq=UAV&ots=E66kgccnRS&sig=C0lGJOe7XhQYJ3zX13eYUYQv2ps.

15. Review of the Current State of UAV Regulations. Available at: https://www.mdpi.com/2072-4292/9/5/459.

16. A MINI UNMANNED AERIAL VEHICLE (UAV): SYSTEM OVERVIEW AND IMAGE ACQUISITION. Available at: https://www.academia.edu/download/38864677/UAV_White_Paper.pdf.

17. Mavlink [Електронний ресурс] – Режим доступу до ресурсу: https://mavlink.io/en/

18. ArUco markers detection [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

19. GitHub-репозиторій QGroundControl [Електронний ресурс] – Режим доступу до ресурсу https://github.com/mavlink/qgroundcontrol

Додаток А

Метод навігації безпілотного літального апарату в умовах агресивного середовища

Функціональна схема будови системи дрону

Аркушів 1

Київ 2023 р.


MAVLink via MAVROS
/mavros/state
/mavros/param/get
/mavros/param/set
/mavros/set_mode
/mavros/cmd/arming
/mavros/home_position/home
/mavros/local_position/pose
/mavros/global_position/global
/mavros/setpoint_position/global
/mavros/setpoint_position/local
/mavros/vision_pose/pose
/mavros/altimeter
/mavros/imu/data
autonomous_flight
/onboard/setpoint/autonomous_flight
/gcs/command
ground_control
Keyboard
drone_control
/onboard/setpoint/
loiter_pilot
loiter_pilot
PIDController
/onboard/state
sensor_fusion
/onboard/sensor_fusion
/onboard/aruco_board
/onboard/aruco_board_found
/onboard/aruco_marker_pose
marker_detection
/onboard/aruco_board_found
onboard/aruco_marker_pose_stable
/onboard/aruco_marker_board_center
/onboard/dis_to_GPS2Vision_marker
Front camera
/mono_cam_front/image_raw
Bottom camera
/mono_cam_bottom/image_raw

Додаток Б

Метод навігації безпілотного літального апарату в умовах агресивного середовища

Лістинг програмного коду

Аркушів

Київ 2023 р.

autonomous_flight.py

```
#!/usr/bin/env python2

from std_msgs.msg import String, Int8, Float32, Bool
from os import sys
from random import randrange, uniform
from uav_flight_modes import*
from log_data import*
from transformations_calculations import*
from waypoint_checking import*
from waypoint_generator import*
from sensor_fusion import SensorFusion
from sensor_fusion_ros_interface import SensorFusionRosInterface
from PIDController import *

class autonomous_flight():

    def __init__(self):

        # Аргументи щодо того, яку місію планують завантажити
        #self.args = sys.argv

        #Ініціація ноди і таймера
        rospy.init_node('autonomous_flight', disable_signals=True)
        self.rate = rospy.Rate(10)

        # Запуск режимів польоту та скриптів реєстрації даних
        self.flight_mode = flight_modes()
        self.log_data = log_data()
        self.tc = transformations_calculations()
        self.wc = waypoint_checking()
        self.wg = waypoint_generator()

        #Змінні для скрипта
        self.uav_local_pose = PoseStamped()
        self.uav_local_setpoint = PoseStamped()
        self.uav_state = None
        self.uav_home_pose = PoseStamped()
        self.init_uav_home_pose = False
        self.aruco_pose = PoseStamped()
        self.aruco_board_found = Bool()
        self.wind_offset = Float32()
        self.aruco_marker_pose_stable = False
        self.aruco_marker_board_center = PoseStamped()
        self.waypoint_check_pose_error = 0.25
        self.aruco_offset = PoseStamped()
        self.uav_offset = PoseStamped()
        self.dis_to_GPS2Vision_marker = 100. #Init to big number
        self.landing_time = 0.0
        self.pre_landing_stabilization_time = 0.0
        self.aruco_board_found_succeses = [False, False, False, False, False]
        self.aruco_board_found_seq = 0

        # Записуйте журнали даних для аналізу
        self.write_data_log_for_landing_test = True
        self.write_data_log_for_gps2vision = True
        self.write_data_log_for_gps2vision_state = 0


        self.ground_truth = Odometry()
```

```
        # Маршрутні точки для маршрутів до першої, другої та третьої станції
посадки
        self.landing_station_one_waypoints = [[[3.65, 1.10, 1.5, 90], [3.65,
4.25, 1.5, 90],
                                               [0.40, 4.25, 1.5, 180], [0.40,
7.10, 1.5, 90]],
                                              [[0.40, 7.10, 1.5, -90], [0.40,
4.25, 1.5, -90], [3.65, 4.25, 1.5, 0],
                                               [3.65, 4.25, 1.5, -90], [3.65,
1.10, 1.5, -90]]]

        self.landing_station_two_waypoints = [[[3.65, 1.10, 1.5, 90], [3.65,
7.10, 1.5, 90]],
                                              [[3.65, 7.10, 1.5, -90], [3.65,
1.10, 1.5, -90]]]

        self.landing_station_three_waypoints = [[[3.65, 1.10, 1.5, 90], [3.65,
4.25, 1.5, 90],
                                                 [7.10, 4.25, 1.5, 0], [7.10,
7.10, 1.5, 90]],
                                                [[7.10, 7.10, 1.5, -90], [7.10,
4.25, 1.5, -90], [3.65, 4.25, 1.5, 180],
                                                 [3.65, 4.25, 1.5, -90], [3.65,
1.10, 1.5, -90]]]

        # Змінна для відстеження використаної станції посадки
        self.landing_station = 3

        #Видавець
        self.pub_setpoint =
rospy.Publisher('/onboard/setpoint/autonomous_flight', PoseStamped,
queue_size=1)
        self.pub_state = rospy.Publisher('/onboard/state', String,
queue_size=10)
        self.pub_aruco_board = rospy.Publisher('/onboard/aruco_board', Int8,
queue_size=1)
        self.pub_aruco_offset = rospy.Publisher('/onboard/aruco_offset',
PoseStamped, queue_size=1)
        self.pub_uav_offset = rospy.Publisher('/onboard/uav_offset',
PoseStamped, queue_size=1)
        self.pub_wind_offset = rospy.Publisher('/wind/offset', Float32,
queue_size=1)

        #Підписник
        rospy.Subscriber('/onboard/state', String, self.uav_state_callback)
        rospy.Subscriber('/onboard/aruco_marker_pose', PoseStamped,
self.aruco_pose_callback)
        rospy.Subscriber('/mavros/local_position/pose', PoseStamped,
self.local_position_callback)
        rospy.Subscriber('/onboard/aruco_board_found', Bool,
self.aruco_board_found_callback)
        rospy.Subscriber('/onboard/aruco_marker_pose_stable', Bool,
self.aruco_marker_pose_stable_callback)
        rospy.Subscriber('/onboard/aruco_marker_board_center', PoseStamped,
self.aruco_marker_board_center_callback)
        rospy.Subscriber('/odom', Odometry, self.ground_truth_callback)
        rospy.Subscriber('/onboard/dis_to_GPS2Vision_marker', Float32,
self.dis_to_GPS2Vision_marker_callback)

        #Ініціалізація uav моделі
        self.flight_mode.wait_for_topics(60)
```

```
        # Ініціалізація PID-контролера
        self.pid_controller = PIDController(kp=1.0, ki=0.1, kd=0.01,
max_integral=10, min_output=-10, max_output=10)

    def local_position_callback(self, msg):
        self.uav_local_pose = msg
        if not self.init_uav_home_pose:
            self.uav_home_pose = self.uav_local_pose
            self.init_uav_home_pose = True

    def ground_truth_callback(self, data):
        self.ground_truth = data

    def dis_to_GPS2Vision_marker_callback(self,msg):
        self.dis_to_GPS2Vision_marker = msg.data

    def aruco_board_found_callback(self, msg):
        self.aruco_board_found = msg.data

    def aruco_marker_pose_stable_callback(self,msg):
        self.aruco_marker_pose_stable = msg.data

    def aruco_marker_board_center_callback(self,msg):
        self.aruco_marker_board_center = msg

    # Функція для оновлення бортового стану
    def set_state(self, state):
        self.sys_state = state
        self.pub_state.publish(state)
        rospy.loginfo('Autonomous_flight: state = {}'.format(state))

    def uav_state_callback(self, msg):
        self.uav_state = msg.data

    def aruco_pose_callback(self,msg):
        self.aruco_pose = msg

    def pub_msg(self, msg, topic, frame_id_gps=True):

        if frame_id_gps:
            msg.header.frame_id = "att_pose_gps"
        else:
            msg.header.frame_id = "att_pose_vision"

        msg.header.stamp = rospy.Time.now()
        topic.publish(msg)

    """ Методи навігації дрона в автономному режимі для зльоту і посадки """
    def drone_takeoff(self, alt=1.5):

        self.flight_mode.set_arm(True,5)

        pre_pose = self.uav_local_pose
        pre_pose.pose.position.z = alt
        rospy.loginfo('Autonomous_flight: Takeoff altitude = {}
m'.format(pre_pose.pose.position.z))

        for i in range(0,5):
            self.pub_msg(pre_pose, self.pub_setpoint)

        #self.flight_mode.set_mode('AUTO.TAKEOFF',10)
        self.flight_mode.set_mode('OFFBOARD',5)
```

```
        rospy.loginfo('Autonomous_flight: UAV takeoff')

        # Чекаємо, поки відбудеться зліт
        waypoint = [self.uav_local_pose.pose.position.x,
self.uav_local_pose.pose.position.y, alt,0]
        print(pre_pose)
        while(not self.wc.waypoint_check(uav_local_pose=self.uav_local_pose,
setpoint_xyzYaw = waypoint, threshold=0.20)): #Set t0 0.05 in loiter to avoid
ascilations
            if self.uav_state == 'loiter' or self.uav_state == 'home':
                rospy.loginfo('Autonomous_flight: Takeoff disrupted!')
                return
            self.pub_msg(pre_pose, self.pub_setpoint)

        rospy.loginfo('Autonomous_flight: Takeoff complete')
        self.set_state('loiter')

    def drone_return_home(self):

        x_cur = self.uav_local_pose.pose.position.x
        y_cur = self.uav_local_pose.pose.position.y
        z_cur = self.uav_local_pose.pose.position.z

        x_home = self.uav_home_pose.pose.position.x
        y_home = self.uav_home_pose.pose.position.y
        z_home = self.uav_home_pose.pose.position.z

        x_delta = abs(x_cur-x_home)
        y_delta = abs(y_cur-y_home)
        z_delta = abs(z_cur-z_home)

        threshold = 0.25
        alt = 1.5
        if x_delta > threshold or y_delta > threshold or z_delta > threshold:

            if not self.flight_mode.state.armed:
                self.flight_mode.set_arm(True,5)
                for i in range(0,5):
                    self.pub_msg(pre_pose, self.pub_setpoint)
                self.flight_mode.set_mode('OFFBOARD',5)
                waypoints = [[x_cur, y_cur, alt, 0], [x_home, y_home, alt, 0],
[0, 0, 0, 0]]
            else:
                waypoints = [[x_home, y_home, alt, 0], [0, 0, 0, 0]]

            rospy.loginfo('Autonomous_flight: UAV returning home')

            for waypoint in waypoints:

                pre_pose = PoseStamped()
                pre_pose.pose.position.x = waypoint[0]
                pre_pose.pose.position.y = waypoint[1]
                pre_pose.pose.position.z = waypoint[2]

                # Чекаємо, поки не буде досягнута маршрутна точка
                while(not
self.wc.waypoint_check(uav_local_pose=self.uav_local_pose,
setpoint_poseStamped=pre_pose)):
                    self.pub_msg(pre_pose, self.pub_setpoint)

            rospy.loginfo('Autonomous_flight: UAV has returned home')
            self.set_state('idle')
        else:
```

```
            rospy.loginfo('Autonomous_flight: UAV already at home position')
            self.set_state('idle')

    """Methods for testing the autonomous flight system"""
    def GPS2Vision_aruco_pose_estimation_test(self):

        # Використання даних з фронтальної камери
        self.pub_aruco_board.publish(1)

        # Зберегти точки як початкову позицію дрона
        x = self.uav_home_pose.pose.position.x
        y = self.uav_home_pose.pose.position.y

        alt_ = 2.5
        self.drone_takeoff(alt = alt_)

        self.flight_mode.set_param('MPC_XY_VEL_MAX', 2.5, 5)
        self.flight_mode.set_param('MPC_Z_VEL_MAX_DN', 1.0, 5)
        self.flight_mode.set_param('MPC_Z_VEL_MAX_UP', 1.0, 5)

        self.set_state('GPS2Vision_aruco_pose_estimation_test')
        rospy.loginfo('Autonomous_flight: GPS2Vision aruco pose_estimation test
startet')

        # Генеруємо сітку маршрутних точок для тестування безпілотника
        waypoints = self.wg.generate_waypoints(x, y, alt_)

        # Тепер перемістіть дрон між точками і запишіть похибку між оцінками
позицій аруко і наземною істиною
        for waypoint in waypoints:
            while(not
self.wc.waypoint_check(uav_local_pose=self.uav_local_pose,setpoint_poseStamped =
waypoint)):
                self.pub_msg(waypoint, self.pub_setpoint)

            # Діємо так, щоб кожен розрахунок похибки виконувався протягом х
секунд            start_time = rospy.get_rostime()
            timeout = rospy.Duration(5) # Тестуємо кожну позицію протягом х
секунд
            seq = 0 # оновлюємо лише тоді, коли з'являються нові оцінки пози
аруко

            while (rospy.get_rostime() - start_time) < timeout:
                if not seq == self.aruco_pose.header.seq:
                    self.log_data.calculate_error_pose(self.aruco_pose,
self.ground_truth)
                seq = self.aruco_pose.header.seq
                self.pub_msg(waypoint, self.pub_setpoint)

            # Тепер знаходимо середню похибку при заданій позиції і записуємо
            x = waypoint.pose.position.x
            y = waypoint.pose.position.y
            self.log_data.write_GPS2Vision_marker_detection_data(x, y)

        self.set_state('home')
        self.drone_return_home()

        rospy.loginfo('Autonomous_flight: GPS2Vision aruco pose estimation test
complete')
        self.set_state('loiter')

    def GPS2Vision_test(self):
```

```
        self.drone_takeoff(alt = 2.5)
        self.set_state('GPS2Vision_test')
        rospy.loginfo('Autonomous_flight: GPS2Vision test started!')

        # Місце для входу на посадочні станції, коли дрон стартує з (-16, 0,
2.5) у симуляції
        entrance_x = 10
        entrance_y = 0
        entrance_z = 2.5

        # Випадкова похибка в x, y та z для ілюстрації невизначеності в GPS
        new_x = entrance_x + uniform(-5, 5)
        new_y = entrance_y + uniform(-5, 5)
        new_z = entrance_z + uniform(-1, 1)
        new_yaw = uniform(-45, 45)

        print(str(new_x) + " " + str(new_y) + " " + str(new_z))

        # Використовується лише для побудови положення дрона для різних станів
        self.write_data_log_for_gps2vision_state = 0

        self.GPS_navigation(waypoints_xyzYaw=[[new_x, new_y, new_z, new_yaw]])
        gps2vision_complete = self.GPS2Vision()
        if gps2vision_complete:
            self.vision_navigation(waypoints_xyzYaw=[[3.65, 1.10, 1.5, 90]])
            rospy.loginfo('Autonomous_flight: GPS2Vision test complete')
            self.flight_mode.set_param('EKF2_AID_MASK', 1, 5)
            self.flight_mode.set_param('EKF2_HGT_MODE', 0, 5)
            #self.set_state('loiter')

        rospy.signal_shutdown("Test completed!")

    def hold_aruco_pose_test(self):

        self.drone_takeoff(alt = 2.5)

        self.pub_aruco_board.publish(1)

        self.set_state('hold_aruco_pose_test')
        rospy.loginfo('Autonomous_flight: Hold position by using the aruco
marker test startet')

        self.GPS_navigation(waypoints_xyzYaw=[[0, 0, 2.5, 0]])
        gps2vision_complete = self.GPS2Vision()

        if gps2vision_complete:

            #Make the test continue for x seconds
            start_time = rospy.get_rostime()
            timeout = rospy.Duration(30.0)

            setpoint = [3.65, -1, 2.50, 0, 0, 90] #x, y, z, roll, pitch, yaw
(GPS2Vision board)
            #setpoint = [0.40, 7.10, 1.5, 0, 0, 90] #x, y, z, roll, pitch, yaw
(Landing board one)
            #setpoint = [3.65, 7.10, 1.5, 0, 0, 90] #x, y, z, roll, pitch, yaw
(Landing board two)
            #setpoint = [7.00, 7.10, 1.5, 0 ,0, 90] #x, y, z, roll, pitch, yaw
(Landing board three)

            pose = PoseStamped()
            pose.pose.position.x = setpoint[0]
            pose.pose.position.y = setpoint[1]
```

```
            pose.pose.position.z = setpoint[2]
            pose.pose.orientation = Quaternion(*quaternion_from_euler(0, 0,
np.deg2rad(setpoint[5]),'rxyz'))

            pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset, pose, True)

            seq = 0 # Пишемо лише тоді, коли з'являються нові оновлення від
aruco pose
           while (rospy.get_rostime() - start_time) < timeout:
                self.fly_route(waypoints_poseStamped=[pose])
                if not seq == self.aruco_pose.header.seq:

self.log_data.write_hold_pose_using_aruco_pose_estimation_data(self.aruco_pose,
setpoint[0], setpoint[1], setpoint[2], setpoint[3], setpoint[4], setpoint[5],
self.ground_truth)
                seq = self.aruco_pose.header.seq

            self.flight_mode.set_param('EKF2_AID_MASK', 1, 5)
            self.flight_mode.set_param('EKF2_HGT_MODE', 0, 5)

            rospy.loginfo('Autonomous_flight: Hold position by using the aruco
marker test complete!')
            rospy.signal_shutdown("Test completed!")
        self.set_state('loiter')

    def vision_navigation_test(self):

        self.drone_takeoff(alt = 2.5)

        self.set_state('vision_navigation_test')
        rospy.loginfo('Autonomous_flight: Vision navigation test startet')

        # Робимо випадковий хід до станції приземлення
        move = randrange(0,3)

        if move == 0:
            waypoints1 = self.landing_station_one_waypoints[0]
            waypoints2 = self.landing_station_one_waypoints[1]
        elif move == 1:
            waypoints1 = self.landing_station_two_waypoints[0]
            waypoints2 = self.landing_station_two_waypoints[1]
        elif move == 2:
            waypoints1 = self.landing_station_three_waypoints[0]
            waypoints2 = self.landing_station_three_waypoints[1]

        self.GPS_navigation(waypoints_xyzYaw=[[0, 0, 2.5, 0]])
        gps2vision_complete = self.GPS2Vision()

        if gps2vision_complete:
            self.vision_navigation(waypoints_xyzYaw=waypoints1)
            self.vision_navigation(waypoints_xyzYaw=waypoints2)

        self.flight_mode.set_param('EKF2_AID_MASK', 1, 5)
        self.flight_mode.set_param('EKF2_HGT_MODE', 0, 5)

        rospy.loginfo('Autonomous_flight: Vision navigation test complete!')
        rospy.signal_shutdown("Test completed!")
        #self.set_state('loiter')

    def landing_test(self):

        rospy.loginfo('Autonomous_flight: Landing test startet')
```

```
        # Маршрутні точки для маршрутів між станціями посадки один, два і три
        landing_station_one2two = [[0.40, 7.10, 1.5, 90], [0.40, 4.25, 1.5, 90],
[3.65, 4.25, 1.5, 90], [3.65, 7.10, 1.5, 90]]
        landing_station_one2three = [[0.40, 7.10, 1.5, 90], [0.40, 4.25, 1.5,
90], [7.00, 4.25, 1.5, 90], [7.00, 7.10, 1.5, 90]]
        landing_station_one_moves = [landing_station_one2two,
landing_station_one2three]

        landing_station_two2one = [[3.65, 7.10, 1.5, 90], [3.65, 4.25, 1.5, 90],
[0.40, 4.25, 1.5, 90], [0.40, 7.10, 1.5, 90]]
        landing_station_two2three = [[3.65, 7.10, 1.5, 90], [3.65, 4.25, 1.5,
90], [7.00, 4.25, 1.5, 90], [7.00, 7.10, 1.5, 90]]
        landing_station_two_moves = [landing_station_two2one,
landing_station_two2three]

        landing_station_three2one = [[7.00, 7.10, 1.5, 90], [7.00, 4.25, 1.5,
90], [0.40, 4.25, 1.5, 90], [0.40, 7.10, 1.5, 90]]
        landing_station_three2two = [[7.00, 7.10, 1.5, 90], [7.00, 4.25, 1.5,
90], [3.65, 4.25, 1.5, 90], [3.65, 7.10, 1.5, 90]]
        landing_station_three_moves = [landing_station_three2one,
landing_station_three2two]

        # Нехай стартова позиція знаходиться на станції посадки 3
        self.landing_station = 3

        # Перевірка точність приземлення x разів
        for _ in range(3):

            # Зробимо випадковий хід до станції приземлення
            move = randrange(0,2)

            # Переміщаємося для вибору конкретної станції приземлення
            new_landing_station = 0
            if self.landing_station == 3:
                waypoints = landing_station_one_moves[move]
                if not move:
                    new_landing_station = 4
                else:
                    new_landing_station = 5

            if self.landing_station == 4:
                waypoints = landing_station_two_moves[move]
                if not move:
                    new_landing_station = 3
                else:
                    new_landing_station = 5

            if self.landing_station == 5:
                waypoints = landing_station_three_moves[move]
                if not move:
                    new_landing_station = 3
                else:
                    new_landing_station = 4

            # Потім зліт, рух і посадка
            self.vision_takeoff_navigation(self.landing_station)
            self.vision_navigation(waypoints_xyzYaw=waypoints)
            self.vision_landing_navigation(new_landing_station, 0.10)

            if self.write_data_log_for_landing_test:

self.log_data.write_vision_landing_precision_and_accuracy_data(self.landing_stat
```

```
ion, self.aruco_pose, waypoints[-1][0], waypoints[-1][1], self.ground_truth,
                        self.pre_landing_stabilization_time, self.landing_time)

        rospy.loginfo('Autonomous_flight: Landing test complete!')
        self.set_state('vision_landed')

    """Methods to be used for navigating the drone in offboard mode"""
    def move2GPS_locations_from_vision(self):

        rospy.loginfo('Autonomous_flight: Drone navigation from vision to GPS
locations started!')
        #self.set_state('return_to_landing_station_one')

        self.vision2GPS_navigation()
        #self.fly_route(waypoints_xyzYaw=[[-8, 0, 2.5, 0],[-8, -5, 2.5, 0]])

        rospy.loginfo('Autonomous_flight: Drone navigation from vision to GPS
locations completed!')
        self.set_state('loiter')

    def return_to_landing_station_one(self):

        self.drone_takeoff(alt = 2.5)
        self.set_state('return_to_landing_station_one')
        rospy.loginfo('Autonomous_flight: Returning to landing station one!')

        self.GPS_navigation(waypoints_xyzYaw=[[8, 4, 2.5, 0]])
        gps2vision_complete = self.GPS2Vision()
        if gps2vision_complete:

self.vision_navigation(waypoints_xyzYaw=self.landing_station_one_waypoints[0])
            self.vision_landing_navigation(3)
            self.landing_station = 3
            rospy.loginfo('Autonomous_flight: Drone returned to landing station
one complete')
            self.set_state('vision_landed')

    def return_to_landing_station_two(self):

        self.drone_takeoff(alt = 2.5)
        self.set_state('return_to_landing_station_two')
        rospy.loginfo('Autonomous_flight: Returning to landing station two!')

        self.GPS_navigation(waypoints_xyzYaw=[[8, 0, 2.5, 0]])
        gps2vision_complete = self.GPS2Vision()
        if gps2vision_complete:

self.vision_navigation(waypoints_xyzYaw=self.landing_station_two_waypoints[0])
            self.vision_landing_navigation(4)
            self.landing_station = 4
            rospy.loginfo('Autonomous_flight: Drone returned to landing station
two complete')
            self.set_state('vision_landed')

    def return_to_landing_station_three(self):

        self.drone_takeoff(alt = 2.5)
        self.set_state('return_to_landing_station_three')
        rospy.loginfo('Autonomous_flight: Returning to landing station three!')

        self.GPS_navigation(waypoints_xyzYaw=[[8, 0, 2.5, 0]])
        gps2vision_complete = self.GPS2Vision()
        if gps2vision_complete:
```

```
self.vision_navigation(waypoints_xyzYaw=self.landing_station_three_waypoints[0])
            self.vision_landing_navigation(5)
            self.landing_station = 5
            rospy.loginfo('Autonomous_flight: Drone returned to landing station
three complete')
            self.set_state('vision_landed')

    """Methods used in autonomous flight for giving substate of drone"""
    def GPS_navigation(self, waypoints_poseStamped = None, waypoints_xyzYaw =
None):

        # Цей метод переміщує дрон між точками маршруту за допомогою GPS
        rospy.loginfo('Autonomous_flight: GPS navigation started!')

        self.flight_mode.set_param('EKF2_AID_MASK', 1, 5)
        self.flight_mode.set_param('EKF2_HGT_MODE', 0, 5)
        self.flight_mode.set_param('MPC_XY_VEL_MAX', 2.0, 5)
        self.flight_mode.set_param('MPC_Z_VEL_MAX_DN', 1.0, 5)
        self.flight_mode.set_param('MPC_Z_VEL_MAX_UP', 1.0, 5)
        self.flight_mode.set_param('MC_ROLLRATE_MAX', 90.0, 5)
        self.flight_mode.set_param('MC_PITCHRATE_MAX', 90.0, 5)
        self.flight_mode.set_param('MC_YAWRATE_MAX', 135.0, 5)

        self.fly_route(waypoints_poseStamped, waypoints_xyzYaw)

        rospy.loginfo('Autonomous_flight: GPS navigation complete!')

    """State GPS2Vision and substates (locate_marker_board,
navigate_to_marker_board and initiate_gps2vision_transition)"""
    def GPS2Vision(self):

        # Цей метод забезпечує плавний перехід від використання GPS до навігації
на основі камери (бачення)
        rospy.loginfo('Autonomous_flight: GPS2Vision started!')

        # Використання даних з фронтальної камери
        self.pub_aruco_board.publish(1)

        # Ініціювати субстрат
        substate = 'locate_marker_board'
        gps2vision_complete = False

        # Максимум помилок при визначенні місцезнаходження дошки аруко перед
посадкою та поверненням
        max_failures = 1

        # Ініціювати сітку маршрутних точок для пошуку маркера, якщо його ще не
знайдено
        start_uav_pose = self.uav_local_pose
        start_x = 0
        end_x = 5
        steps_x = 1
        start_y = 0
        end_y = 1
        steps_y = 1
        start_yaw = -45
        end_yaw = 90
        steps_yaw = 45

        # Кінцеве положення для GPS2Vision
        while not gps2vision_complete:
            if substate == 'locate_marker_board':
```

```
                # Використовується лише для побудови положення дрона для різних
станів
                self.write_data_log_for_gps2vision_state = 1

                rospy.loginfo('Autonomous_flight: Substate (locate_marker_board)
- GPS2Vision!')
                fails = 0
                marker_board_found = False
                while True:
                    # Пошук дошки в заданій сітці маршрутних точок
                    waypoints =
self.wg.generate_locate_marker_board_waypoints(start_uav_pose, start_x, end_x,
steps_x, start_y, end_y, steps_y, start_yaw, end_yaw, steps_yaw)
                    marker_board_found = self.locate_marker_board(waypoints)
                    # Зміна підкладки, якщо плату знайдено
                    if not marker_board_found:
                        fails += 1
                        rospy.loginfo('Autonomous_flight: Failed to find board -
Retries - GPS2Vision!')
                    else:
                        substate = 'navigate_to_marker_board'
                        break
                    if fails > max_failures:
                        self.flight_mode.set_mode('AUTO.LAND',10)
                        self.set_state('idle')
                        rospy.loginfo('Autonomous_flight: Critical error! Marker
board NOT found - GPS2Vision!')
                        return False
                    if fails == 1:
                        start_y = -6
                        end_y = 6
                        steps_y = 3

            # Підстанова (navigate_to_marker_board)
            if substate == 'navigate_to_marker_board':
                # Використовується лише для побудови положення дрона для різних
станів
                self.write_data_log_for_gps2vision_state = 2

                rospy.loginfo('Autonomous_flight: Substate
(navigate_to_marker_board) - GPS2Vision!')
                # Навігація дрону на маркерну дошку і переключення, тільки якщо
оцінка положення на маркерній дощі хороша
                self.flight_mode.set_param('MPC_XY_VEL_MAX', 1.0, 5)
                navigate_to_marker_board_complete =
self.navigate_to_marker_board()
                if navigate_to_marker_board_complete:
                    substate = 'initiate_gps2vision_transition'
                else:
                    rospy.loginfo('Autonomous_flight: Lost sight of aruco board
in substate (navigate_to_marker_board) - GPS2Vision!')
                    substate = 'locate_marker_board'

            #Підстанова (initiate_gps2vision_transition)
            if substate == 'initiate_gps2vision_transition':
                # Використовується лише для побудови положення дрона для різних
станів
                self.write_data_log_for_gps2vision_state = 3

                rospy.loginfo('Autonomous_flight: Substate
(initiate_gps2vision_transition) - GPS2Vision!')
                # Ініціація переходу від GPS до бачення
initiate_gps2vision_transition = self.initiate_gps2vision_transition()
```

```
                if initiate_gps2vision_transition:
                    gps2vision_complete = True
                else:
                    rospy.loginfo('Autonomous_flight: Lost sight of aruco board
in substate (initiate_gps2vision_transition) - GPS2Vision!')
                    self.flight_mode.set_param('EKF2_AID_MASK', 1, 5)
                    self.flight_mode.set_param('EKF2_HGT_MODE', 0, 5)
                    self.flight_mode.set_mode('AUTO.LAND',10)
                    start_time = rospy.get_rostime()
                    timeout = rospy.Duration(10.0) # Чекаємо деякий час, щоб
стабілізувати положення дрона
                    while (rospy.get_rostime() - start_time) < timeout:
                        pass
                    return False

        rospy.loginfo('Autonomous_flight: GPS2Vision complete!')
        return True

    def locate_marker_board(self, waypoints_poseStamped):

        #Цей метод намагається знайти маркерну дошку GPS2Vision, якщо її не
видно за допомогою фронтальної камери з
        #останньої путівної точки, переданої за допомогою GPS
        rospy.loginfo('Autonomous_flight: Locate marker board started!')

        for waypoint in waypoints_poseStamped:
            start_time = rospy.get_rostime()
            timeout_stabilize_pose = rospy.Duration(1.0) # Чекаємо деякий час,
щоб стабілізувати положення дрона
            timeout_detect_board = rospy.Duration(2.0) # Чекаємо недовго, щоб
побачити, скільки разів дошка aruco була знайдена

            aruco_board_found_success = False
            while (rospy.get_rostime() - start_time) < timeout_detect_board:
                if (rospy.get_rostime() - start_time) > timeout_stabilize_pose
and self.validate_marker_board_detections():
                    aruco_board_found_success = True
                    break
                self.update_marker_board_detections(self.aruco_board_found)
            if aruco_board_found_success:
                rospy.loginfo('Autonomous_flight: Marker board found - Locate
marker board complete!')
                return True
            self.reset_marker_board_detections()

            self.fly_route(waypoints_poseStamped=[waypoint])
        return False

    def navigate_to_marker_board(self):

        #Цей метод спрямовує дрон до виявленої маркерної дошки доти, доки оцінка
положення не стане стабільною і не перевищить певний поріг.
        #Визначення стабільності базується на ковзному середньому з класу
виявлення маркерів
        rospy.loginfo('Autonomous_flight: Navigate drone to marker board
started!')

        increment_init = True
        increment_x = 0.5
        increment_y = 0.5
        increment_z = 0.5

        pose = self.uav_local_pose
```

```
        min_dis = 4
        while (self.dis_to_GPS2Vision_marker > min_dis) and not
self.aruco_marker_pose_stable:
            #Простий спосіб тримати дрон орієнтованим на маркерну дошку
            board_center_x = self.aruco_marker_board_center.pose.position.x
            board_center_y = self.aruco_marker_board_center.pose.position.y
            error = (360 - board_center_x)/360 # Тому що ширина зображення 720 і
потрібно нормалізувати (від 0 до 1)
            #error_z = (240 - board_center_y)/240 #Бо висота зображення 480 і
потрібно нормалізувати (від 0 до 1)
             angle_yaw = np.deg2rad(error*30) #Максимальна зміна кута рискання
на 30 градусів

            angle =
euler_from_quaternion([self.uav_local_pose.pose.orientation.x,

self.uav_local_pose.pose.orientation.y,

self.uav_local_pose.pose.orientation.z,

self.uav_local_pose.pose.orientation.w])


            #Рухаємося до маркера, що базується на поточній орієнтації дрона
            if increment_init:
                new_x = self.uav_local_pose.pose.position.x
                new_y = self.uav_local_pose.pose.position.y
                new_z = self.uav_local_pose.pose.position.z
                increment_init = False
            else:
                new_x = self.uav_local_pose.pose.position.x +
np.cos(angle[2])*increment_x
                new_y = self.uav_local_pose.pose.position.y +
np.sin(angle[2])*increment_y
                new_z = 2.5 #self.uav_local_pose.pose.position.z +
error_z*increment_z

            pose.pose.position.x = new_x
            pose.pose.position.y = new_y
            pose.pose.position.z = new_z
            pose.pose.orientation = Quaternion(*quaternion_from_euler(0, 0,
angle[2]+angle_yaw,'rxyz'))

            self.pub_msg(pose, self.pub_setpoint)

            #Оновлює виявлення дошки aruco, щоб переконатися, що дрон
знаходиться на шляху до маркера.
            #Повертає false, якщо маркерна дошка не була помічена протягом
декількох ітерацій
            self.update_marker_board_detections(self.aruco_board_found)
            if not self.validate_marker_board_detections():
                return False

            if self.write_data_log_for_gps2vision:
                self.log_data.write_gps2vision_data(self.ground_truth,
self.write_data_log_for_gps2vision_state)

        #Зачекайте x секунд, щоб оцінити оцінку з ковзних середніх значень
        start_time = rospy.get_rostime()
        timeout = rospy.Duration(3.0)
        while (rospy.get_rostime() - start_time) < timeout:
            if self.aruco_marker_pose_stable:
                break
```

```
        min_dis -= 0.5
        while(not self.wc.waypoint_check(uav_local_pose=self.uav_local_pose,
setpoint_poseStamped = pose, threshold = 0.25)):
            self.pub_msg(pose, self.pub_setpoint)
            if self.write_data_log_for_gps2vision:
                self.log_data.write_gps2vision_data(self.ground_truth,
self.write_data_log_for_gps2vision_state)

        return True
        rospy.loginfo('Autonomous_flight: Navigate drone to marker board
complete!')

    def initiate_gps2vision_transition(self):

        #Тепер перетворіть позу аруко на позу GPS, щоб забезпечити плавний
перехід до GPS2Vision
        self.tc.map_GPS_pose_to_vision(self.aruco_pose, self.uav_local_pose)
        self.pub_uav_offset.publish(self.uav_local_pose)
        self.pub_aruco_offset.publish(self.aruco_pose)
        self.flight_mode.set_param('MPC_XY_VEL_MAX', 1.0, 5)
        self.flight_mode.set_param('MPC_Z_VEL_MAX_DN', 0.5, 5)
        self.flight_mode.set_param('MPC_Z_VEL_MAX_UP', 0.5, 5)
        self.flight_mode.set_param('MC_ROLLRATE_MAX', 90.0, 5)
        self.flight_mode.set_param('MC_PITCHRATE_MAX', 90.0, 5)
        self.flight_mode.set_param('MC_YAWRATE_MAX', 90.0, 5)
        self.flight_mode.set_param('EKF2_AID_MASK', 24, 5)
        self.flight_mode.set_param('EKF2_HGT_MODE', 3, 5)

        #Визначення початкової позиції в'їзду перед маркером GPS2Vision
        pose = PoseStamped()
        pose.pose.position.x = 3.65
        pose.pose.position.y = 1.10
        pose.pose.position.z = 2.50
        pose.pose.orientation = Quaternion(*quaternion_from_euler(0, 0,
np.deg2rad(90),'rxyz'))

        pose = self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
pose, calculate_setpoint=True)

        #Переміщаємо дрон до входу перед маркерною дошкою GPS2Vision
        while(not self.wc.waypoint_check(uav_local_pose=self.uav_local_pose,
setpoint_poseStamped = pose)):

            #Відома позиція маркера GPS2Vision зі зміщенням 0.5 до центру. Таким
чином, орієнтація
            #дрона завжди спрямована на маркерну дошку GPS2Vision
            delta_x = self.aruco_pose.pose.position.x - 3.2 - 0.5
            delta_y = self.aruco_pose.pose.position.y - 3.0

            theta = np.arctan2(delta_y, delta_x)
            if theta < 0:
                theta = theta + np.pi

            theta = (self.tc.GPS2Vision_offset[3][0] -
(self.tc.GPS2Vision_offset[3][1] - theta))
            pose.pose.orientation = Quaternion(*quaternion_from_euler(0, 0,
theta,'rxyz'))
            self.pub_msg(pose, self.pub_setpoint, True)

            #Оновлює виявлення дошки aruco, щоб переконатися, що дрон
знаходиться на шляху до маркера.
```

```
            #Повертає false, якщо маркерна дошка не була помічена протягом
декількох ітерацій
            self.update_marker_board_detections(self.aruco_board_found)
            if not self.validate_marker_board_detections():
                return False

            if self.write_data_log_for_gps2vision:
                self.log_data.write_gps2vision_data(self.ground_truth,
self.write_data_log_for_gps2vision_state)

        return True

    def vision2GPS_navigation(self):
        rospy.loginfo('Autonomous_flight: Vision2GPS started!')

        #Використання станції посадки, де знаходиться дрон
        self.pub_aruco_board.publish(self.landing_station)

        self.vision_takeoff_navigation(self.landing_station)

        if self.landing_station == 3:
            waypoints_xyzYaw = self.landing_station_one_waypoints[1]

        if self.landing_station == 4:
            waypoints_xyzYaw = self.landing_station_two_waypoints[1]

        if self.landing_station == 5:
            waypoints_xyzYaw = self.landing_station_three_waypoints[1]
        print(waypoints_xyzYaw)
        self.vision_navigation(waypoints_xyzYaw=waypoints_xyzYaw)
        self.GPS_navigation(waypoints_poseStamped=[self.uav_local_pose])

        rospy.loginfo('Autonomous_flight: Vision2GPS complete!')

    def vision_navigation(self, waypoints_poseStamped = None, waypoints_xyzYaw =
None):

        #Ця підстанція орієнтує дрон за допомогою зору
        rospy.loginfo('Autonomous_flight: Vision navigation started!')

        #Використання даних з нижньої камери
        self.pub_aruco_board.publish(2)

        self.fly_route(waypoints_poseStamped=waypoints_poseStamped,
waypoints_xyzYaw=waypoints_xyzYaw, GPS2Vision_offset=self.tc.GPS2Vision_offset)

        rospy.loginfo('Autonomous_flight: Vision navigation completed!')

    def vision_landing_navigation(self, landing_station,
waypoint_check_pose_error = 0.1):

        #Ця підстанція орієнтує дрон за допомогою зору
        rospy.loginfo('Autonomous_flight: Vision landing started!')

        #Використання даних з фронтальної камери для маркера посадки
        self.pub_aruco_board.publish(landing_station)

        if landing_station == 3:
            waypoints_xyzYaw = [[0.4, 7.1, 1.5, 90], [0.4, 7.1, 0.2, 90]]
        elif landing_station == 4:
            waypoints_xyzYaw = [[3.65, 7.1, 1.5, 90], [3.65, 7.1, 0.2, 90]]
        elif landing_station == 5:
            waypoints_xyzYaw = [[7.0, 7.1, 1.5, 90], [7.0, 7.1, 0.2, 90]]
```

```
        #Отримання часу приземлення (Тільки для тестування))
        start_time = rospy.get_rostime()
        self.fly_route(waypoints_xyzYaw=[waypoints_xyzYaw[0]],
GPS2Vision_offset=self.tc.GPS2Vision_offset,
waypoint_check_pose_error=waypoint_check_pose_error)
        self.pre_landing_stabilization_time = (rospy.get_rostime() -
start_time)/1000000000.

        start_time = rospy.get_rostime()
        self.fly_route(waypoints_xyzYaw=[waypoints_xyzYaw[1]],
GPS2Vision_offset=self.tc.GPS2Vision_offset, waypoint_check_pose_error=0.20)
        self.flight_mode.set_mode('AUTO.LAND',10)
        self.landing_time = (rospy.get_rostime() - start_time)/1000000000.

        self.flight_mode.force_disarm()

        start_time = rospy.get_rostime()
        timeout = rospy.Duration(2.0)
        while self.flight_mode.state.armed:
            if (rospy.get_rostime() - start_time) > timeout:
                rospy.loginfo('Autonomous_flight: Vision landing witout
disarming! Vision landing completed!')
                self.landing_station = landing_station
                break

        if not self.flight_mode.state.armed:
            self.landing_station = landing_station
            rospy.loginfo('Autonomous_flight: Vision landing completed!')

    def vision_takeoff_navigation(self, landing_station):

        rospy.loginfo('Autonomous_flight: Vision takeoff started!')

        self.pub_aruco_board.publish(landing_station)

        ret = self.flight_mode.get_param_uav(param_id='EKF2_AID_MASK')
        if ret == None or not ret.success:
            rospy.sleep(5)

        # Встановлення на оцінки бачення, якщо не ввімкнено
        EKF2_AID_MASK = self.flight_mode.get_param('EKF2_AID_MASK')

        if not EKF2_AID_MASK.value.integer == 24:
            print(EKF2_AID_MASK.value.integer)
            #Тепер перетворення пози аруко на позу GPS, щоб забезпечити плавний
перехід до GPS2Vision
            self.tc.map_GPS_pose_to_vision(self.aruco_pose, self.uav_local_pose)

            self.pub_uav_offset.publish(self.uav_local_pose)
            self.pub_aruco_offset.publish(self.aruco_pose)

            #Встановлюємо максимальний горизонтальний та вертикальний об'єми
            self.flight_mode.set_param('MPC_XY_VEL_MAX', 0.5, 5)
            self.flight_mode.set_param('MPC_Z_VEL_MAX_DN', 0.5, 5)
            self.flight_mode.set_param('MPC_Z_VEL_MAX_UP', 0.5, 5)
            #Встановити максимальні кутові швидкості
            self.flight_mode.set_param('MC_ROLLRATE_MAX', 90.0, 5)
            self.flight_mode.set_param('MC_PITCHRATE_MAX', 90.0, 5)
            self.flight_mode.set_param('MC_YAWRATE_MAX', 90.0, 5)

            #Використовувати зір для оцінки пози
            self.flight_mode.set_param('EKF2_AID_MASK', 24, 5)
```

```
            self.flight_mode.set_param('EKF2_HGT_MODE', 3, 5)

        #Озброюємо дрон і переходимо до бортового управління
        self.flight_mode.set_arm(True,10)
        self.flight_mode.set_mode('OFFBOARD',5)

        if landing_station == 3:
            waypoints_xyzYaw = [[0.4, 7.1, 1.5, 90]]
        elif landing_station == 4:
            waypoints_xyzYaw = [[3.65, 7.1, 1.5, 90]]
        elif landing_station == 5:
            waypoints_xyzYaw = [[7.0, 7.1, 1.5, 90]]
        self.fly_route(waypoints_xyzYaw=waypoints_xyzYaw,
GPS2Vision_offset=self.tc.GPS2Vision_offset)

        rospy.loginfo('Autonomous_flight: Vision takeoff completed!')

    def fly_route(self, waypoints_poseStamped = None, waypoints_xyzYaw = None,
GPS2Vision_offset = None, waypoint_check_pose_error=0.25):

        frame_id_gps = True
        if not waypoints_poseStamped == None:

            for waypoint in waypoints_poseStamped:
                if not GPS2Vision_offset == None:
                    waypoint =
self.tc.calculate_GPS2Vision_offset(GPS2Vision_offset, waypoint, True)
                    frame_id_gps = False
                while(not
self.wc.waypoint_check(uav_local_pose=self.uav_local_pose,
setpoint_poseStamped=waypoint, threshold = waypoint_check_pose_error)):
                    self.pub_msg(waypoint, self.pub_setpoint, frame_id_gps)
                    if self.write_data_log_for_gps2vision:
                        self.log_data.write_gps2vision_data(self.ground_truth,
self.write_data_log_for_gps2vision_state)

        #Список маршрутних точок у вигляді [x,y,z,yaw].
        else:
            for waypoint in waypoints_xyzYaw:
                pose = PoseStamped()
                pose.pose.position.x = waypoint[0]
                pose.pose.position.y = waypoint[1]
                pose.pose.position.z = waypoint[2]
                pose.pose.orientation = Quaternion(*quaternion_from_euler(0, 0,
np.deg2rad(waypoint[3]),'rxyz'))

                if not GPS2Vision_offset == None:
                    pose =
self.tc.calculate_GPS2Vision_offset(GPS2Vision_offset, pose, True)
                    frame_id_gps = False
                while(not
self.wc.waypoint_check(uav_local_pose=self.uav_local_pose,
setpoint_poseStamped=pose, threshold = waypoint_check_pose_error)):
                    self.pub_msg(pose, self.pub_setpoint, frame_id_gps)
                    if self.write_data_log_for_gps2vision:

self.log_data.write_gps2vision_data(self.ground_truth,self.write_data_log_for_gp
s2vision_state)

                # Використання PID-контролера для регулювання положення
                current_position = [self.uav_local_pose.pose.position.x,
self.uav_local_pose.pose.position.y,
                                    self.uav_local_pose.pose.position.z]
```

```
                desired_position = [waypoint[0], waypoint[1], waypoint[2]]

                # Розрахунок помилки для кожної осі
                error_x = desired_position[0] - current_position[0]
                error_y = desired_position[1] - current_position[1]
                error_z = desired_position[2] - current_position[2]

                # Оновлення PID-контролера
                control_x = self.pid_controller.update(error_x,
self.rate.sleep_dur.to_sec())
                control_y = self.pid_controller.update(error_y,
self.rate.sleep_dur.to_sec())
                control_z = self.pid_controller.update(error_z,
self.rate.sleep_dur.to_sec())

                # Використання PID-контролера для коригування положення
                control_x, control_y, control_z =
self.sensor_fusion.get_pid_control(desired_position, current_position)

    """ Методи перевірки кількості виявлених дощок аруко за звітний період """
    def update_marker_board_detections(self, board_found):
        if not self.aruco_marker_board_center.header.seq ==
self.aruco_board_found_seq:
            self.aruco_board_found_succeses.pop(0)
            self.aruco_board_found_succeses.append(board_found)
            self.aruco_board_found_seq =
self.aruco_marker_board_center.header.seq

    def validate_marker_board_detections(self):
        threshold = 0.4 #80% виявлення маркерних дощок дають істинні результати
        detections = 0.0
        for success in self.aruco_board_found_succeses:
            if success:
                detections += 1./len(self.aruco_board_found_succeses)
        if detections > threshold:
            return True
        else:
            return False

    def reset_marker_board_detections(self):
        self.aruco_board_found_succeses = [False, False, False, False, False]

    def run(self):
        while not rospy.is_shutdown():
            #Для зльоту і повернення в початкове положення
            if self.uav_state == 'takeoff':
                self.drone_takeoff()
            elif self.uav_state == 'home':
                self.drone_return_home()
            #Для тестування системи автономного польоту
            elif self.uav_state == 'GPS2Vision_aruco_pose_estimation_test':
                self.GPS2Vision_aruco_pose_estimation_test()
            elif self.uav_state == 'hold_aruco_pose_test':
                self.hold_aruco_pose_test()
            elif self.uav_state == 'GPS2Vision_test':
                self.GPS2Vision_test()
            elif self.uav_state == 'vision_navigation_test':
                self.vision_navigation_test()
            elif self.uav_state == 'landing_test':
                self.landing_test()
            #Для навігації безпілотника в автономному польоті
            elif self.uav_state == 'move2GPS_locations_from_vision':
                self.move2GPS_locations_from_vision()
```

```
            elif self.uav_state == 'return_to_landing_station_one':
                self.return_to_landing_station_one()
            elif self.uav_state == 'return_to_landing_station_two':
                self.return_to_landing_station_two()
            elif self.uav_state == 'return_to_landing_station_three':
                self.return_to_landing_station_three()
            self.rate.sleep()

if __name__ == "__main__":
    af = autonomous_flight()
    af.run()
```

drone_control.py

```
#!/usr/bin/env python

import rospy
from  mavros_msgs.msg import State
from mavros_msgs.srv import SetMode
from std_msgs.msg import String, Int8, Bool
from geometry_msgs.msg import PoseStamped
from transformations_calculations import*
from PIDController import *

class drone_control():
    def __init__(self):

        #Аргументи для початкового стану uav, в якому потрібно перебувати
        self.arg = sys.argv

        #Ініціалізація ROS
        rospy.init_node('drone_control')
        self.rate = rospy.Rate(30)

        #Створити об'єкт для обчислення зміщення uav2aruco
        self.tc = transformations_calculations()

        #Змінні
        self.uav_state = 'idle'
        self.mavros_state = State()
        self.aruco_marker_pose = PoseStamped()
        self.uav_pose = PoseStamped()
        #self.autonomous_fligt_state = None
        self.sensor_fusion = PoseStamped()
        self.aruco_offset = PoseStamped()
        self.start_pub_local_vision = False

        #Використовується для розрахунків зміщення UAV2Aruco
        self.aruco_offset_init = False
        self.uav_offset = PoseStamped()
        self.uav_offset_init = False
        self.use_GPS2Vision_offset = False

        # Уставки від вузлів
        self.autonomous_flight_pose_setpoint = PoseStamped()
        self.loiterpilot_pose_setpoint = PoseStamped()
```

```
        rospy.Subscriber('/mavros/state', State, self.uav_state_callback)
        rospy.Subscriber('/gcs/command', Int8, self.gcs_command_callback)
        rospy.Subscriber('/onboard/setpoint/autonomous_flight',
PoseStamped, self.af_setpoint_callback)
        rospy.Subscriber('/onboard/state', String,
self.onboard_uav_state_callback)
        rospy.Subscriber('/onboard/setpoint/loiter_pilot', PoseStamped,
self.lp_setpoint_callback)
        rospy.Subscriber('/onboard/aruco_marker_pose', PoseStamped,
self.aruco_pose_callback)
        rospy.Subscriber('/onboard/aruco_offset', PoseStamped,
self.aruco_offset_callback)
        rospy.Subscriber('/onboard/uav_offset', PoseStamped,
self.uav_offset_callback)
        rospy.Subscriber('/onboard/sensor_fusion', PoseStamped,
self.sensor_fusion_callback)
        rospy.Subscriber('/mavros/local_position/pose', PoseStamped,
self.local_position_callback)
        self.pub_state = rospy.Publisher('/onboard/state', String,
queue_size=1)
        self.pub_local_pose =
rospy.Publisher('/mavros/setpoint_position/local', PoseStamped,
queue_size=1)
        self.pub_vision_pose =
rospy.Publisher('/mavros/vision_pose/pose', PoseStamped, queue_size=1)
        self.pub_waypoint_check =
rospy.Publisher('/onboard/waypoint_check', Bool, queue_size=1)

        #Сервіси
        self.set_mode = rospy.ServiceProxy('/mavros/set_mode', SetMode)

        #Виконуємо рукостискання MAVROS
        self.mavros_handshake()

        self.set_state(self.arg[1])

    def mavros_handshake(self):
        rospy.loginfo('Drone_control: Waiting for MAVROS Connection.')
        i=0
        time = rospy.Time.now()
        for i in range(0,3):
            print'.',
            if self.mavros_state.connected:
                rospy.loginfo("Drone_control: MAVROS Connected!")
                break
            rospy.sleep(1)
        if not self.mavros_state.connected:
            errorMsg = "Drone_control: MAVROS not connected!"
            rospy.logfatal(errorMsg)
            rospy.signal_shutdown(errorMsg)

    def pub_msg(self, msg, topic):
        msg.header.frame_id = "att_pose"
        msg.header.stamp = rospy.Time.now()
        topic.publish(msg)
        self.rate.sleep()
```

```
    def local_position_callback(self,msg):
        self.uav_pose = msg

    #Функція для оновлення бортового стану
    def set_state(self, state):
        self.uav_state = state
        self.pub_state.publish(state)
        rospy.loginfo('Drone_control: state = {}'.format(state))

    def af_setpoint_callback(self,msg):
        self.autonomous_flight_pose_setpoint = msg

    def sensor_fusion_callback(self,msg):
        self.sensor_fusion = msg

    def aruco_offset_callback(self,msg):
        offset = msg
        self.aruco_offset = msg
        self.aruco_offset_init = True

    def uav_offset_callback(self,msg):
        offset = msg
        self.uav_offset = msg
        self.uav_offset_init = True

    def onboard_uav_state_callback(self,msg):
        self.uav_state = msg.data

    def lp_setpoint_callback(self,msg):
        self.loiterpilot_pose_setpoint = msg

    def aruco_pose_callback(self,msg):
        self.aruco_marker_pose = msg

    def uav_state_callback(self, msg):
        self.mavros_state = msg

    def gcs_command_callback(self, msg):

        #Змінити стан згідно з командою GC
        command = str(chr(msg.data))
        command.lower()

        if command == 't': #Takeoff
            self.set_state('takeoff')

        if command == 'h': #Returns the drone to home
            self.set_state('home')

        if command == 'l': #Execute mission
            self.set_state('loiter')

        #Виконуйте місії з GPS до станцій посадки або місць розташування
vision2GPS
        if command == '0':
            self.set_state('move2GPS_locations_from_vision')
```

```
        if command == '1':
            self.set_state('return_to_landing_station_one')

        if command == '2':
            self.set_state('return_to_landing_station_two')

        if command == '3':
            self.set_state('return_to_landing_station_three')

        #Виконувати тести з підстанцій місій
        if command == '4':
            self.set_state('GPS2Vision_aruco_pose_estimation_test')

        if command == '5':
            self.set_state('hold_aruco_pose_test')

        if command == '6':
            self.set_state('GPS2Vision_test')

        if command == '7':
            self.set_state('vision_navigation_test')

        if command == '8':
            self.set_state('landing_test')

    def message_control(self):

        if not self.uav_state == 'idle':

            output_msg = None

            #Якщо потрібна оцінка маркерів аруко на карті для
позиціонування глобуса БПЛА
            if self.aruco_offset_init and self.uav_offset_init:
                self.tc.map_GPS_pose_to_vision(self.aruco_offset,
self.uav_offset)
                self.aruco_offset_init = False
                self.uav_offset_init = False


            if self.uav_state == 'loiter':
                output_msg = self.loiterpilot_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'takeoff':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'home':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)
```

```
            elif self.uav_state == 'move2GPS_locations_from_vision':
                output_msg = self.autonomous_flight_pose_setpoint
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.sensor_fusion)
                self.pub_msg(output_msg, self.pub_local_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'return_to_landing_station_one':
                output_msg = self.autonomous_flight_pose_setpoint
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.aruco_marker_pose)
                self.pub_msg(output_msg, self.pub_local_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'return_to_landing_station_two':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.aruco_marker_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'return_to_landing_station_three':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.aruco_marker_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state ==
'GPS2Vision_aruco_pose_estimation_test':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'hold_aruco_pose_test':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.aruco_marker_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)
```

```
            elif self.uav_state == 'GPS2Vision_test':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.aruco_marker_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'vision_navigation_test':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.sensor_fusion)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'landing_test':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.aruco_marker_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            elif self.uav_state == 'vision_landed':
                output_msg = self.autonomous_flight_pose_setpoint
                self.pub_msg(output_msg, self.pub_local_pose)
                new_pose =
self.tc.calculate_GPS2Vision_offset(self.tc.GPS2Vision_offset,
self.aruco_marker_pose)
                self.pub_msg(new_pose, self.pub_vision_pose)
                pid_output = self.sensor_fusion.pose
                self.pub_msg(pid_output, self.pub_local_pose)

            if output_msg == None:
                rospy.logfatal_once("Drone control received no message:
Has a pilot crashed?")
                self.set_mode(0, "AUTO.LOITER")
                rospy.loginfo('Drone_control: PX4 mode = AUTO.LOITER')

    def run(self):
        while not rospy.is_shutdown():
            self.message_control()
            self.rate.sleep()

if __name__ == "__main__":
    dc = drone_control()
    dc.run()
```

PIDController.py

```
import rospy
from sensor_msgs.msg import Imu
from mavros_msgs.msg import Altitude
from geometry_msgs.msg import PoseStamped, Quaternion
from std_msgs.msg import (String, Int8, Float64, Bool)
from nav_msgs.msg import Odometry
from scipy.stats import norm
from hector_uav_msgs.msg import Altimeter
from log_data import *

class PIDController:
    def __init__(self, kp, ki, kd, max_integral, min_output, max_output):
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.max_integral = max_integral
        self.min_output = min_output
        self.max_output = max_output

        self.integral = 0
        self.previous_error = 0

    def reset(self):
        self.integral = 0
        self.previous_error = 0

    def update(self, error, dt):
        self.integral += error * dt
        self.integral = max(min(self.integral, self.max_integral), -
self.max_integral)

        derivative = (error - self.previous_error) / dt
        output = self.kp * error + self.ki * self.integral + self.kd *
derivative
        output = max(min(output, self.max_output), self.min_output)

        self.previous_error = error

        return output
```

sensor_fusion.py

```
    #!/usr/bin/env python2

import numpy as np
from scipy.linalg import block_diag
from PIDController import *
# from timeit import default_timer as timer
# from pathlib import Path

import rospy
from sensor_msgs.msg import Imu
from mavros_msgs.msg import Altitude
from geometry_msgs.msg import PoseStamped, Quaternion
from tf.transformations import *
from std_msgs.msg import (String, Int8, Float64, Bool)
from nav_msgs.msg import Odometry
```

```
from scipy.stats import norm
from hector_uav_msgs.msg import Altimeter
from ukf import UKF
from log_data import*
from PIDController import*


class sensor_fusion():

    def __init__(self):
        # Ініціалізація даних IMU
        self.imu_data = Imu()

        # Ініціалізація калібровки IMU
        self.calibrate_imu()

        # Ініціалізація PID-контролерів
        self.pid_x = PIDController(kp=1.0, ki=0.1, kd=0.01,
max_integral=10, min_output=-10, max_output=10)
        self.pid_y = PIDController(kp=1.0, ki=0.1, kd=0.01,
max_integral=10, min_output=-10, max_output=10)
        self.pid_z = PIDController(kp=1.0, ki=0.1, kd=0.01,
max_integral=10, min_output=-10, max_output=10)

        # Ініціалізувати невідцентрований фільтр Калмана (UKF)
        np.set_printoptions(precision=3)

        # self.q = np.eye(12)
        # self.q[0][0] = 0.5  # x 0.5
        # self.q[1][1] = 0.5  # y 0.5
        # self.q[2][2] = 0.5  # z 0.5
        # self.q[3][3] = 0.5  # vel x 0.5
        # self.q[4][4] = 0.5  # vel y 0.5
        # self.q[5][5] = 0.5  # roll 0.5
        # self.q[6][6] = 0.5  # pitch 0.5
        # self.q[7][7] = 0.5  # yaw 0.5
        # self.q[8][8] = 0.5  # rate roll 0.5
        # self.q[9][9] = 0.5  # rate pitch 0.5
        # self.q[10][10] = 0.5  # rate yaw 0.5
        # self.q[11][11] = 3.0  # baro bias 2.5
        self.update_noise_parameters()

        # Create measurement noise covariance matrices
        self.r_imu_acc = np.zeros([2, 2])
        self.r_imu_gyro_v = np.zeros([3, 3])
        self.r_imu_acc[0][0] = 0.3  # acc x 0.5
        self.r_imu_acc[1][1] = 0.3  # acc y 0.5
        self.r_imu_gyro_v[0][0] = 0.03  # gyro roll 0.05
        self.r_imu_gyro_v[1][1] = 0.03  # gyro pitch 0.05
        self.r_imu_gyro_v[2][2] = 0.03  # gyro yaw 0.05

        self.r_vision_pos = np.zeros([3, 3])
        self.r_vision_ori = np.zeros([3, 3])
        self.r_vision_pos[0][0] = 0.011  # x 0.0005
        self.r_vision_pos[1][1] = 0.011  # y 0.0005
        self.r_vision_pos[2][2] = 0.011  # z 0.0005
        self.r_vision_ori[0][0] = 0.0054  # roll rate 0.0005
```

```
self.r_vision_ori[1][1] = 0.0054  # pitch rate 0.0005
self.r_vision_ori[2][2] = 0.0054  # yaw rate 0.0005

self.r_baro = np.zeros([1, 1])
self.r_baro[0][0] = 0.5  # 0.5

self.r_baro_offset = np.zeros([1, 1])
self.r_baro_offset[0][0] = 0.5  # 0.005

# Для візуалізації даних
self.write_data_log = True
self.log_data = log_data()

self.x0 = []  # x
self.x1 = []  # y
self.x2 = []  # z
self.x3 = []  # vel_x
self.x4 = []  # vel_y
self.x5 = []  # vel_z
self.x6 = []  # roll
self.x7 = []  # pitch
self.x8 = []  # yaw
self.x9 = []  # psi
self.x10 = []  # phi
self.x11 = []  # theta

self.vision_x = 0.0
self.vision_y = 0.0
self.vision_z = 0.0

self.vision_roll = 0.0
self.vision_pitch = 0.0
self.vision_yaw = 0.0

self.imu_accX = 0.0
self.imu_accY = 0.0

self.imu_rollV = 0.0
self.imu_pitchV = 0.0
self.imu_yawV = 0.0

self.baro = 0.0
self.baro_offset = 0.0
self.baro_corrected = 0.0

self.last_sample_time_imu = 0.0
self.last_sample_time_vision = 0.0

self.seq_imu = 0
self.seq_aruco_pose = 0
self.seq_baro = 0
self.init_ukf = True

self.inversions_z = []
self.last_step_z = 0
self.start_z = 0
self.attenuation_vel_z = 1.0
```

```
        self.ground_truth = Odometry()
        self.imu_data = Imu()
        self.baro_data = Altimeter()

        self.init_baro_altitude = True
        self.altitude_ref = []

        self.imu_data_corrected = []
        self.aruco_marker_pose = PoseStamped()
        self.sensor_fusion_pose = PoseStamped()
        self.aruco_board_found = Bool()
        self.aruco_board_found = False
        self.start_tracking = False

        self.measurements = []

        #         # self.acc_x_offset = -0.190006656546
        #         # self.acc_y_offset = -0.174740895383
        #         # self.acc_z_offset = 9.79531049538
        #         # self.gyro_x_offset = -0.000143702855887
        #         # self.gyro_y_offset = -0.000105893216729
        #         # self.gyro_z_offset = 0.0018871804653

    def calibrate_imu(self):
        print("Calibrating IMU...")
        # Збираємо декілька вимірювань для визначення середнього
        acc_x, acc_y, acc_z = 0, 0, 0
        gyro_x, gyro_y, gyro_z = 0, 0, 0
        num_samples = 100
        for _ in range(num_samples):
            acc_x += self.imu_data.linear_acceleration.x
            acc_y += self.imu_data.linear_acceleration.y
            acc_z += self.imu_data.linear_acceleration.z
            gyro_x += self.imu_data.angular_velocity.x
            gyro_y += self.imu_data.angular_velocity.y
            gyro_z += self.imu_data.angular_velocity.z
            rospy.sleep(0.01)  # коротка пауза між зчитуваннями

        # Обчислюємо середнє
        self.acc_x_offset = acc_x / num_samples
        self.acc_y_offset = acc_y / num_samples
        self.acc_z_offset = acc_z / num_samples
        self.gyro_x_offset = gyro_x / num_samples
        self.gyro_y_offset = gyro_y / num_samples
        self.gyro_z_offset = gyro_z / num_samples
        print("Calibration complete.")

    def initialize_ukf(self, aruco_board_found, aruco_marker_pose):

        if self.init_ukf and aruco_board_found and
aruco_marker_pose.header.seq > 0:
            # Обираємо початкові значення для UKF
            i = self.get_measurements()
            self.x_init = [i[0], i[1], i[2], 0, 0, i[6], i[7], i[8], 0,
0, 0, 0]
```

```
            # Ініціація УКФ
            self.state_estimator = UKF(12, self.q, self.x_init, 0.0001 *
np.eye(12), 0.04, 15.0, 2.0, self.iterate_x)

            self.init_ukf = False
            self.start_tracking = True

    def iterate_x(self, x, dt, inputs):
        # Додаємо аеродинамічний опір
        drag_coefficient = 0.1  # Припустимо, це коефіцієнт опору
        speed = np.linalg.norm([x[3], x[4], x[5]])
        drag_force = -drag_coefficient * speed ** 2
        ret = np.zeros(len(x))

        ret[0] = x[0] + 0.6 * x[3] * dt
        ret[1] = x[1] + 0.6 * x[4] * dt
        ret[2] = x[2]
        ret[3] = x[3] + drag_force * dt
        ret[4] = x[4] + drag_force * dt
        ret[5] = x[5] + x[9] * dt
        ret[6] = x[6] + x[8] * dt
        ret[7] = x[7] + x[10] * dt
        ret[8] = x[8]
        ret[9] = x[9]
        ret[10] = x[10]
        ret[11] = x[11]

        return ret

    def update_noise_parameters(self):
        speed = np.linalg.norm([self.x3, self.x4, self.x5])  # швидкість
з останньої оцінки
        if speed > some_threshold:  # Припустимо, 'some_threshold' це
певна швидкість
            self.q[3:6, 3:6] *= 1.1  # Збільшуємо шум для складових
швидкості

    def apply_pid_control(self):
        # Отримання поточного стану з UKF
        x = self.state_estimator.get_state()

        # Розрахунок помилки для PID-контролера
        error_x = desired_x - x[0]  # desired_x - це бажане значення
положення по осі X
        control_x = self.pid_x.update(error_x, dt_imu)

        error_y = desired_y - x[1]
        control_y = self.pid_y.update(error_y, dt_imu)

        error_z = desired_z - x[2]
        control_z = self.pid_z.update(error_z, dt_imu)

    def sensor_fusion_update(self, aruco_marker_pose, imu_data,
baro_data, ground_truth):

        # Виклик PID-контролю
        self.apply_pid_control()
```

```
        # Оновлення внутрішніх параметрів
        self.aruco_marker_pose = aruco_marker_pose
        self.imu_data = imu_data
        self.baro_data = baro_data
        self.ground_truth = ground_truth

        # Отримуйте дані про зір, IMU та барокамеру
        row = self.get_measurements()

        # Оновлюємо час дискретизації (він не завжди однаковий)
        time = self.imu_data.header.stamp.secs +
self.imu_data.header.stamp.nsecs / 1000000000.
        dt_imu = time - self.last_sample_time_imu
        self.last_sample_time_imu = time

        time = self.aruco_marker_pose.header.stamp.secs +
self.aruco_marker_pose.header.stamp.nsecs / 1000000000.
        dt_vision = time - self.last_sample_time_vision
        self.last_sample_time_vision = time

        # Отримуємо першу оцінку стану
        x = self.state_estimator.get_state()

        # Зсув акселерометра
        bias = [self.acc_x_offset, self.acc_y_offset]  # -0.190006656546,
-0.174740895383]

        # Корекція зсуву прискорення для орієнтації дрона та зсуву для x
та y
        R2 = self.eulerAnglesToRotationMatrix([x[6], x[5], x[7]])
        acc = np.matmul(R2, np.array([row[3], row[4], 0]))
        acc_bias = np.matmul(R2, np.array([bias[0], bias[1], 0]))

        corrected_acc_x = -acc[0]  # + acc_bias[0]
        corrected_acc_y = -acc[1]  # + acc_bias[1]

        # Матриця обертання для вирівнювання швидкості гіроскопа за
орієнтацією світу (маркер)
        R3 = self.eulerAnglesToRotationMatrix([0, 0, x[7] - np.pi])
        corrected_gyro = np.matmul(R3, np.array([row[10], -row[9],
row[11]]))

        self.vision_x = row[0]
        self.vision_y = row[1]
        self.vision_z = row[2]

        self.vision_roll = row[6]
        self.vision_pitch = row[7]
        self.vision_yaw = row[8]

        self.imu_accX = corrected_acc_x
        self.imu_accY = corrected_acc_y

        self.imu_rollV = corrected_gyro[0]
        self.imu_pitchV = corrected_gyro[1]
        self.imu_yawV = corrected_gyro[2]
```

```
        vision_data_pos = np.array([self.vision_x, self.vision_y,
self.vision_z])
        vision_data_ori = np.array([self.vision_roll, self.vision_pitch,
self.vision_yaw])
        imu_data_acc = np.array([self.imu_accX, self.imu_accY])
        imu_data_gyro_v = np.array([self.imu_rollV, self.imu_pitchV,
self.imu_yawV])

        baro_data = np.array([row[12]])
        baro_offset = self.vision_z - baro_data

        self.baro = row[12]
        self.baro_corrected = self.baro + x[11]

        # Крок прогнозування для UKF
        self.state_estimator.predict(dt_imu)

        # Оновлювати тільки якщо отримано нові дані на основі порядкового
номеру (ROS-повідомлення)
        if not self.seq_aruco_pose == self.aruco_marker_pose.header.seq:
            self.state_estimator.update([0, 1, 2], vision_data_pos,
self.r_vision_pos)
            self.state_estimator.update([5, 6, 7], vision_data_ori,
self.r_vision_ori)
            self.state_estimator.update([11], baro_offset,
self.r_baro_offset)
            self.seq_aruco_pose = self.aruco_marker_pose.header.seq

        # Оновлювати тільки якщо отримано нові дані на основі порядкового
номеру (ROS-повідомлення))
        if not self.seq_imu == self.imu_data.header.seq:
            self.state_estimator.update([3, 4], imu_data_acc,
self.r_imu_acc)
            self.state_estimator.update([8, 9, 10], imu_data_gyro_v,
self.r_imu_gyro_v)
            self.seq_imu = self.imu_data.header.seq

        if not self.seq_baro == self.baro_data.header.seq:
            self.state_estimator.update([2], baro_data + x[11],
self.r_baro)
            self.seq_baro = self.baro_data.header.seq

        # отримуємо оцінку стану
        x = self.state_estimator.get_state()

        self.sensor_fusion_pose.pose.position.x = x[0]
        self.sensor_fusion_pose.pose.position.y = x[1]
        self.sensor_fusion_pose.pose.position.z = x[2]
        self.sensor_fusion_pose.pose.orientation =
Quaternion(*quaternion_from_euler(x[5], x[6], x[7], 'rxyz'))

    def get_measurements(self):

        x, y, z, roll, pitch, yaw = 0., 0., 0., 0., 0., 0.
        acc_x, acc_y, acc_z, gyro_x, gyro_y, gyro_z = 0., 0., 0., 0., 0.,
0.
```

```
            # отримуємо орієнтацію маркера
            q = (self.aruco_marker_pose.pose.orientation.x,
                 self.aruco_marker_pose.pose.orientation.y,
                 self.aruco_marker_pose.pose.orientation.z,
                 self.aruco_marker_pose.pose.orientation.w)
            euler = euler_from_quaternion(q)
            roll = euler[0]
            pitch = euler[1]
            yaw = euler[2]

            # Отримати переклад маркера
            x = self.aruco_marker_pose.pose.position.x
            y = self.aruco_marker_pose.pose.position.y
            z = self.aruco_marker_pose.pose.position.z
            vision_seq = self.aruco_marker_pose.header.seq

            # Отримати прискорення з акселерометра
            acc_x = self.imu_data.linear_acceleration.x
            acc_y = self.imu_data.linear_acceleration.y
            acc_z = self.imu_data.linear_acceleration.z
            imu_seq = self.imu_data.header.seq

            # Отримати поточну орієнтацію на місцевості
            t_g =
quaternion_matrix(np.array([self.ground_truth.pose.pose.orientation.x,

self.ground_truth.pose.pose.orientation.y,

self.ground_truth.pose.pose.orientation.z,

self.ground_truth.pose.pose.orientation.w]))

            # Отримати поточний переклад наземного перекладу
            t_g[0][3] = self.ground_truth.pose.pose.position.x
            t_g[1][3] = self.ground_truth.pose.pose.position.y
            t_g[2][3] = self.ground_truth.pose.pose.position.z

            # Обертання для вирівнювання базової істини з маркером аруко
            r_m = euler_matrix(0, 0, np.pi / 2, 'rxyz')

            # Виконання множення матриць для вирівнювання позицій
            T = np.matmul(r_m, t_g)

            # Отримуйте ракурси та переклад у прогресі
            g_euler = euler_from_matrix(T, 'rxyz')

            g_x = T[0][3] + 7.4 / 2
            g_y = T[1][3] + 7.4 / 2
            g_z = T[2][3]
            g_roll = np.rad2deg(g_euler[0])
            g_pitch = np.rad2deg(-g_euler[1])
            g_yaw = np.rad2deg(g_euler[2])

            # Отримати вимірювання гіроскопа
            gyro_x = self.imu_data.angular_velocity.x - self.gyro_x_offset
            gyro_y = self.imu_data.angular_velocity.y - self.gyro_y_offset
```

```
        gyro_z = self.imu_data.angular_velocity.z - self.gyro_z_offset

        # Отримати висоту з барометра (висотоміра))
        altitude = self.baro_data.altitude
        baro_seq = self.baro_data.header.seq

        # Отримати час повідомлення
        time_imu = self.imu_data.header.stamp.secs +
self.imu_data.header.stamp.nsecs / 1000000000.
        time_vision = self.aruco_marker_pose.header.stamp.secs +
self.aruco_marker_pose.header.stamp.nsecs / 1000000000.

        if not self.init_ukf and self.write_data_log:
            self.log_data.write_vision_imu_data(x, y, z, acc_x, acc_y,
acc_z, roll, pitch, yaw, gyro_x, gyro_y, gyro_z,
                                                altitude, time_imu,
vision_seq, imu_seq, baro_seq, g_roll, g_pitch,
                                                g_yaw,
                                                g_x, g_y, g_z,
time_vision)

            i = self.state_estimator.get_state()
            self.log_data.write_sensor_fusion_data(i[0], i[1], i[2],
i[3], i[4],

                                                   np.rad2deg(i[5]),
np.rad2deg(i[6]), np.rad2deg(i[7]),

                                                   np.rad2deg(i[8]),
np.rad2deg(i[9]), np.rad2deg(i[10]),

                                                   self.baro,
self.baro_corrected, i[11],
                                                   self.vision_x,
self.vision_y, self.vision_z,
                                                   time_imu, g_roll,
g_pitch, g_yaw, g_x, g_y,
                                                   g_z)
        self.measurements = [x, y, z, acc_x, acc_y, acc_z, roll, pitch,
yaw, gyro_x, gyro_y, gyro_z, altitude]
        return self.measurements

    def eulerAnglesToRotationMatrix(self, theta):

        R_x = np.array([[1, 0, 0],
                        [0, np.cos(theta[0]), -np.sin(theta[0])],
                        [0, np.sin(theta[0]), np.cos(theta[0])]])

        R_y = np.array([[np.cos(theta[1]), 0, np.sin(theta[1])],
                        [0, 1, 0],
                        [-np.sin(theta[1]), 0, np.cos(theta[1])]])

        R_z = np.array([[np.cos(theta[2]), -np.sin(theta[2]), 0],
                        [np.sin(theta[2]), np.cos(theta[2]), 0],
                        [0, 0, 1]])

        R = np.matmul(R_x, np.matmul(R_y, R_z))
```

```
        return R


if __name__ == "__main__":
    sf = sensor_fusion()
```