МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інтерактивний тренажер для навчання оператора квадрокоптера»

Завідувач
випускаючої кафедри Довбиш А.С.
Керівник роботи Шелехов І.В.
Студента гр. ІН–62 Гладких Є.О.

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую _____

Зав. кафедрою Довбиш А.С.

“_____” _____20__ р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, ІН – 62 спеціальності “Комп’ютерні науки” денної форми навчання Гладких Є.О.

Тема: “Інтерактивний тренажер для навчання оператора квадрокоптера ”

Затверджена наказом по СумДУ

№_____ від _____20___ р.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) проектування тренажер-симулятор; 3) програмна реалізація.

Дата видачі завдання “_____” _____20__ г.

Керівник випускної роботи _____Шелехов І.В.

Завдання прийняв до виконання _____Гладких Є.О.

РЕФЕРАТ

Записка: 54 стор., 27 рис., 1 додаток, 17 джерел.

Об'єкт дослідження — процес проектування та реалізації інформаційного та програмного забезпечення інтерактивного тренажеру

Мета роботи — розробка та програмна реалізація інтерактивного тренажеру для тренування управлінням квадрокоптеру.

Результати — було виконано розробку інформаційного та програмного забезпечення інтерактивного тренажеру. В роботі було описано склад та класифікація квадрокоптерів, описані їх слабкі та сильні сторони, позитивний вплив симуляторів в навчанні. Програмну реалізацію виконано за допомогою середовища Unity.

БПЛА, СИМУЛЯТОР, СЕРЕДОВИЩЕ РОЗРОБКИ, ІГРОВИЙ РУШІЙ ІНТЕРАКТИВНИЙ ТРЕНАЖЕР, UNITY

ЗМІСТ

ВСТУП

Застосування безпілотних літальних апаратів(дронів) – БПЛА, раніше, аж до кінця 20 століття, було виключно в військових цілях. Тільки 10 -15 років тому, безпілотні апарати міцно почали входити в наш побут і отримали широке застосування в бізнесі, житті держави та населенні. Розглянемо деякі історичні факти зародження БПЛА.[9]

Перша задокументована згадка про безпілотники, це застосування під час облоги Венеції 12 липня 1849 року австрійськими військами. Два запущених по місту аеростата з вибухівку. На цих аеростатах були відсутні пілоти, але були механізми які скинули бомби в потрібний час. У 1910 році американський військовий інженер Чарльз Кеттерінг, запропонував ідею безпілотної бомби з крилами, тим самим, заклав початок розвитку безпілотної ударної авіації.

На початку 30 років 20 століття технології побудови повітряних суден відкрили нові горизонти – впровадження автопілотів. У 1935 році англійські інженери створили БПЛА багаторазового використання, який отримує назву "QueenBee". Ця модель використовувалася як мішень в Королівському флоті Великобританії до 1947 року. Вже в середині 20 століття перебували люди, які вірили в те, що ці передові технології будуть використовуватися не тільки у військових але і в мирних цілях. Але роздуми про масове використання БПЛА в мирних цілях, були в категорії наукової фантастики, аж до теперішнього часу.[10]

Сектор авіації, який динамічно розвивається і охоплює дуже багато сфер діяльності – це безпілотні літальні апарати(БПЛА). Застосування цих приладів підняло на більш ефективний рівень діяльність багатьох галузей. Дрони знайшли застосовуються у цивільних та військових областях. Для вирішення завдань екологічного моніторингу, дистанційного зондування поверхні Землі, стратегічного та тактичного розвідування, спостереження за об’єктами транспортної інфраструктури, в фото– та відеозйомці і т. п.[1]. Ці літальні апарати безумовно будуть актуальними протягом тривалого часу,

судячи з попиту, що росте з кожним днем. І для цього будуть потрібні оператори дронів – люди, які вміють керувати БПЛА.

Фахівець, який управляє сучасними літальними апаратами дистанційно є оператор. Умови роботи оператора БПЛА можуть відрізнятися в залежності від сфери діяльності. Оператор дрона, що працює в межах прямої видимості, може стояти на вулиці з пультом дистанційного керування і направляти дрон, знімаючи цілий квартал. Військові оператори працюють за межами прямої видимості, знаходячись на авіабазі, відстежуючи обстановку навколо апарату, який знаходиться за десятки тисяч кілометрів від бази.[2]

Вважається, що успішними операторами БПЛА стають любителі відеоігор. Це обумовлюється тим що, ігри дуже реалістично повторюють фізику справжніх польотів. Пілотування дрона дуже схоже на звичні ігрові симулятори. Сучасні відеоігри допомагають виробити швидкість реакції, навички навігації в повітрі та навички керування літальним апаратом при різних погодних умовах і позаштатних ситуаціях.

Немає кращого способу для тренувань, ніж польоти в симуляторі. Перш ніж почати польоти на реальному БПЛА або мультикоптері, новачок повинен навчитися літати в симуляторі. В іншому випадку, перший політ, швидше за все, закінчиться падінням. До того ж, польоти в симуляторі реально підвищують навички та здібності до польотів, навіть у досвідчених пілотів. Правда, так буде, якщо симулятор якісний і в ньому передана вся реальна фізика польоту.[3]

В рамках даної роботи буде описано створення інтерактивного симулятора управління квадрокоптером під назвою "FreeRider" на програмному двигуні Unity. Даний продукт буде створений на базі ОС Windows, але також в подальшому може бути портований на інші ОС.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд БПЛА

У світі розвиваються технології з використанням дронів, які застосовуються в фактично во всіх сферах життєдіяльності. [4] Вони використовуються для аерофотозйомки, а саме дистанційного обліку тварин в мисливських господарствах, стану нафто– і газопроводу та електромереж, моніторингу лісних ресурсів та охоронюваних об'єктів, в координації пошуково рятувальних операцій. Впроваджуються технології точного землеробства, наприклад, для розрахунку кількості посівного насіння, обсягу польових робіт, вибірковість внесення добрив, виявлення шкідників. Також застосування дронів актуальне в екологічному моніторингу, це єдиний спосіб ефективно контролювати стан навколишнього середовища. Завдяки застосуванню БПЛА вирішуються кур'єрські завдання, наприклад доставка вантажу клієнтам, як в американському магазині Amazon. Одним з перспективних напрямків є створення мережі супутників і дронів, що утворюють інтернет покриття по всій поверхні землі. [5]

БПЛА важко класифікувати так як вони мають різні характеристики. Виробники БПЛА поки ще не обмежені ніякими стандартами. З боку авіаційних регуляторів немає загальних вимог, якими повинен бути обладнаний дрон. Вони відзначаються габаритами, функціональністю, дальністю польоту, вагою, типу зльоту, корисним навантаженням, спеціалізацією, рівнем автономності і іншими характеристиками.

Безпілотні літальні апарати поступово стають популярні та є головною продукцією багатьох авіаційних фірм. З'являється велика кількість розробників і виробників БПЛА, що займаються виключно безпілотними апаратами і системами. Це відбувається з ряду причин. Самі БПЛА набагато дешевше пілотованих літаків і вертольотів. Також дешевше ніж підготовка льотчика, обходиться і підготовка оператора безпілотної системи. Відсутність пілота дозволяє виключити бортові системи життєзабезпечення, зменшити

масу і габарити БПЛА, а також збільшити діапазон допустимих перевантажень та можливостей. [8]

Сучасні моделі дронів можуть мати як просту конструкцію, призначену в основному для розваг, так і складатися зі складних елементів, що дозволяють розширювати їх функціонал в залежності від їх застосування.

Незважаючи на те, що внутрішній устрій дрона (мультикоптера) може відрізнятися, його базова складова завжди незмінна. У загальному вигляді кожному дрону можна виділити наступні частини: рама, польотний контролер, двигун, акумулятор, комунікаційний контролер, сенсори.

Термін «безпілотний літальний апарат» включає як великі літаки, аналогічні за розміром і складністю пілотованому літаку так і невеликі електронні пристрої для персонального використання. Більшість невеликих дронів зазвичай поставляються, готовими до польоту. Їх використання широко, від розваг до різноманітних професійних напрямків.[6]

Як і більшість мобільних пристроїв, дрони або мультикоптери живляться від акумуляторів. Вони поставляються з окремим контролером, який використовується як пульт управління дроном. Хоч і конструктори намагаються спроектувати міцні конструкції та розробити інтелектуальні системи, це не врятує дрона від недосвідченого оператора.

1.2 Класифікація та різновиди мультикоптерів(БПЛА)

Мультикоптер є безпілотний літальний апарат вертолітного типу, що має від двох та більше електродвигунів з повітряними гвинтами. У зв'язку з цим рівновага реактивних моментів досягається за рахунок попарно різноспрямованого обертання несучих гвинтів.[**Ошибка! Источник ссылки не найден.**]

Підйомна аеродинамічна сила у мультикоптерів створюється, за рахунок обертових лопатей несучих гвинтів, а не за рахунок крил. Крила або їх відсутність, відіграють допоміжну роль. Зависання в точці і висока маневреність є очевидними перевагами БПЛА вертолітного типу. Також

однією з переваг мультикоптерів є відсутність спеціально обладнаних аеродромів для їх запусків.

Класифікація мультикоптерів по тактико–технічним характеристикам є дуже важливою. Вона дуже сильно спрощує роботу в практично во всіх галузях. За призначення мультикоптери поділяються :

–аматорські, призначені для фото та відеозйомки;

–для бізнесу, доставка пошти або різних товарів;

–для наукових досліджень, спостереження за різними об'єктами або збір інформації;

–наземні (в атмосфері), використання для астрономічних або метеорологічних спостережень;

–для силових структур або військових, патруль прикордонної смуги.

В залежності від маси та дальності застосування. У цій класифікації безпілотні апарати коливаються в розмірах від мікрокоптеров менше 10 кг, які знаходяться поблизу від оператора, до важких, здатних підніматися на 20 км і бути в повітрі 24 г:

- мікро– і міні– мультикоптери мають вагу менше 4 кг і здатні віддалятися від оператора на відстань до 30–40 км;
- легкі(або малі) мають вагу до 100 кг і здатні відлітати на відстань до 120 км;
- середні мають вагу до 310 кг і здатні перебувати на відстані до 1100 км від оператора;
- середньотяжкі мультикоптери мають вагу до 500 кг і здатні відлітати на відстань до 2000 км;
    - важкі квадрокоптера мають вагу від 500 кг і здатні віддалятися від оператора на відстань до 3000 км, при цьому вони можуть перебувати в повітрі досить тривалий час.

Класифікація Мультикоптер в залежності від розміру рами:

- мікро– і міні–клас, мають довжину рами менш 250 мм і застосовуються в основному для аматорських цілей;

- від 250 до 350 мм, можуть бути обладнані фото– або відеокамерою і здатні розвивати чималу швидкість;
- від 350 до 450 мм, здатні піднімати вагу обладнання, достатній для здійснення прямих відеотрансляцій;
- від 450 до 550 мм, часто застосовуються для здійснення бізнес–діяльності і доставки різних товарів;
- від 550 до 750 мм, застосовуються в основному для здійснення наукової діяльності, так як здатні нести на борту достатню кількість дослідницького обладнання;
- понад 750 мм, входять в розділ важких мулькоптерів і здатні здійснювати військову діяльність

Класифікація Мультикоптер в залежності від кількості двигунів. У мультикоптера може бути непарна або парна кількість гвинтів постійного кроку. Кожен гвинт працює від власного двигуна. Як правило, половина гвинтів обертається за годинниковою стрілкою, а друга частина – проти годинникової стрілки. Гвинтів може бути по одному на двигун або коаксіальною (соосною) по два. [11]

1) Бікоптери і трікоптери. Вони мають відповідно два(рис 1.1) і три(рис 1.2) мотора, які обертаються в різних напрямках для компенсації обертального моменту. Їх загальною перевагою є відносна простота конструкції, дешевизна, а також висока енергоефективність. Конструкція бікоптера забезпечує найменший рівень шуму в польоті. В свою чергу конструкція рами трікоптера надає апарату здатність поєднувати переваги гвинтокрила і літака, що позначається на його покращених польотних характеристиках, на відміну від квадрокоптера. В основному всі вони знайшли застосування у спортсменів моделістів.

Рисунок 1.1 – Вигляд квадрокоптеру

2) Квадрокоптер – чотирьохроторний безпілотник, який має гарні показники автономності а також здатен нести вантаж. Завдяки конструкції дрон менш схильний до вібрації і популярний в якості платформи для зйомки з повітря. Основний недолік такий , як і у попередніх – відмова одного двигуна призводить до падіння. Квадрокоптер є оптимальним рішенням для використання в прикладних цілях, завдяки його невисокій ціні і надійності.

Рисунок 1.2 – Вигляд квадрокоптеру

3) Гексакоптери та Октокоптери.– апарати з шістьма та вісьма двигунами – істотно надійніші і дорожчі. Вони одні з найбільш просунутих рішень, на ринку мультикоптерів. Їх великою перевагою є – відмовостійкість, у разі поломки вони здатні здійснити аварійне планування на уцілілих двигунах. Це дорогі, але надійні апарати, яким довіряють цінний вантаж, наприклад, професійну відео– та фототехніку.

Рисунок 1.3 – Вигляд октокоптеру та гексакоптеру

4) Многороторний мультикоптер на базі більш ніж восьми моторів створюються для підвищення надійності, маневреності, стабілізації. Літальні апарати, ціна яких істотно вище в порівнянні з розглянутими раніше, призначені для виконання спеціальних завдань.

Рисунок 1.4 – Вигляд многороторного мультикоптера

У цій роботі буде розглядатися БПЛА типу «квадрокоптер». Саме ця гілка БПЛА набуває найбільшої популярності серед людей, як засіб зйомки фото та відео.

1.3 Складова квадрокоптера

У теперішній час, моделі дронів можуть мати як різні конструкції (проста конструкція призначена в основному для розваг) так і складатися зі складних складових елементів, що дозволяють перетворити ці літальні апарати по–справжньому в вузькоспеціальні пристрої[12].

Внутрішній устрій квадрокоптера може відрізнятися від моделі до моделі, не дивлячись на це, його базові складові завжди незмінні. У загальному вигляді кожному дрону можна виділити наступні частини:

1. Рама
2. Двигуни
3. Регулятори обертів
4. Пропелери
5. Польотний контролер

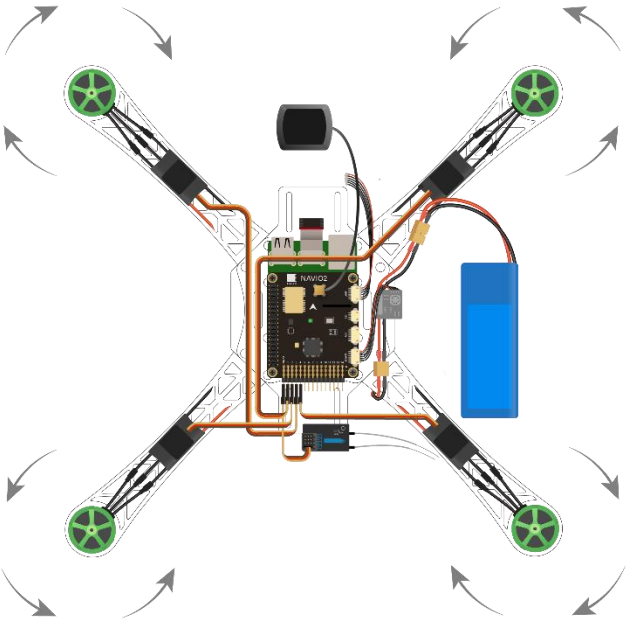


Рисунок 1.5 – Типовий устрій квадрокоптера

Рама. Основою дрона є рама, до якої кріпляться всі складові його елементи. Головне завдання при виготовленні рами, створити пристрій який буде легким, ударостійким, довговічним і також ергономічним.

Застосування надлегких матеріалів, дозволяє апарату за невеликий проміжок часу розвивати свою максимальну швидкість польоту. Для того щоб рама була легкою і міцною використовують полімери або легкі сплави. Щоб забезпечити максимальну жорсткість рами, використовують карбон, скловолокно та інші матеріали. Найлегше добитися більшої маневреності і легкості апарату – використовувати раму з багатьох складових деталей. Для зручності монтажа електропроводки, є конструктивні отвори, які з'єднують пролітний контролер з усіма іншими частинами дрона.

Рисунок 1.6 – Вигляд двигуна з пропелером

Двигуни та пропелери. Квадрокоптер має чотири двигуни з пропелерами, які надають рух зі швидкістю, що генерується регуляторами обертів. Залежно від вступників від польотного контролера команд, відбувається регулювання обертів двигуна.

Рисунок 1.7 – Вигляд двигуна з пропелером

Найбільш швидкозношуваними частинами дрона є пропелери і двигуни, оскільки при його польоті основне навантаження лягає саме на них. Найчастіше, при аварійних ситуаціях першими ламаються пропелери, які являються відкритими частинами конструкції. Ця проблема вирішується установкою захисних дуг, або кожухів.

«Мозок» дрона – це польотний контролер. Завдяки встановленим датчикам він здатний розрізняти сигнали, що надходять з дистанційного пульта оператора. Універсальність дрона характеризується кількістю оброблюваних їм сигналів. Кожен з чотирьох двигунів, з'єднаний шлейфом з контролером, що дозволяє подавати на них сигнали (запрограмовані команди).

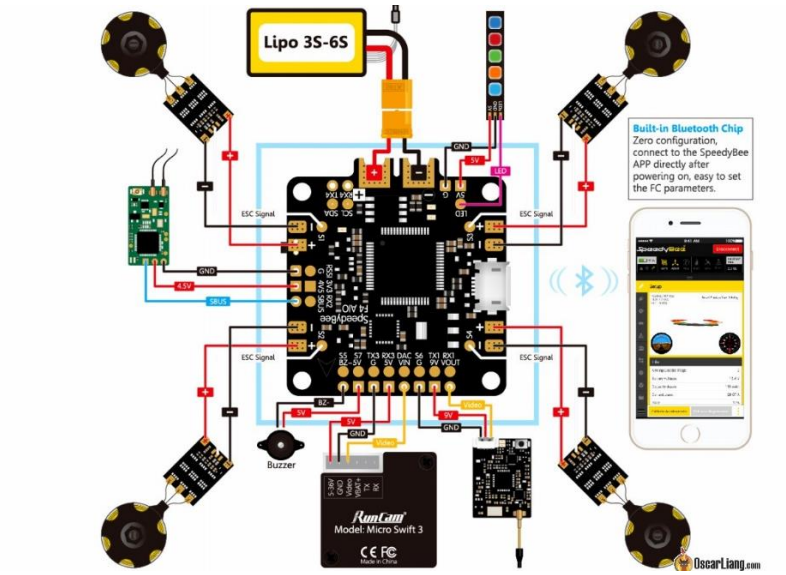


Рисунок 1.8 – Вигляд контролера(в центрі)

Для стабільності роботи польотного контролера, при створенні дронів застосовуються методи, які дозволяють максимально віброізолювати елементи управління, від яких залежить точність управління польоту. Тому в теперішній час, випускаються дрони у яких достатня віброізоляція. Як правило, чим краще віброізольован контролер, тим стабільніше буде літати дрон.

Польотний контролер складається з різних наборів датчиків (GPS, лазерного далекоміра, барометра, гіроскопу, акселерометра, лідара та інші), які передають йому свої показання. Завдяки передавачу установленому на корпусі, здійснюється зворотний зв'язок. Оператор задає параметри польоту дрона в залежності від показань різних датчиків. Ціна контролера залежить від якості комплектуючих і програмних алгоритмів. Наприклад, контролер може самостійно змінити параметри польоту безпілотника, не вдаючись при цьому до допомоги оператора.[13]

Основними технічними параметрами квадрокоптерів є:

1) Габарити. Довжина і висота вироба. Впливає в основному на розмір двигунів і гвинтів, що в свою чергу впливає на максимальну швидкість, швидкість підйому і зниження апарату;
2) Акумулятор. Вимірюється ємність батареї в міліампер на годину. Чим показник вище, тим довше квадрокоптер літає на одному заряді;
3) Радіус дії. Діаметр, в межах якого зможе літати квадрокоптер;
4) Частота передавача / приймача. Вливає на якість прийому і передачі сигналу:
    - 900 Mhz. Плюс цієї частоти – сигнал з легкістю проходить крізь дерева, поєднується з апаратурою, що працює на 2.4 Ghz;
    - 1.2 Ghz. Працюючі на цій частоті дрони, долають довгі дистанції. Сигнал легко проходить через тверді предмети;
    - 2.4 Ghz. Сигнал гірше проходить через тверді предмети, але дрон відлітає на цій частоті на далекі дистанції;

5) Вага. Впливає на керованість, тобто здатність чинити опір вітру. Напряму залежить від габаритів конструкції;
6) Розширення камери. Впливає на деталізацію картинки. При низькій роздільній здатності не вийде схопити багато об'єктів і розгорнути їх на широкоформатному екрані. Іноді якість зйомки у камер, що знімають в низькій роздільній здатності, краще. До базових факторів, що впливають на якість відео, відносять: оптику, оптичну стабілізацію, розмір матриці.

Різноманітність характеристик та класифікацій впливає на складність керування дронів. Для найбільш вдалого тренування керуванням БПЛА існують тренажери–симулятори.

1.4 Огляд симуляторів

Окремою нішею в ігровій індустрії, стоять симулятори. Симулятор – це жанр комп'ютерних ігор, геймплей яких зосереджується на імітації певних дій, наприклад, керування літаком, автомобілем, квадрокоптером.

З розвитком сучасних технологій, можливості ігрових симуляторів сильно зросли. Вони показують дуже високий рівень графіки, а також здатні відтворити реалістичну фізичну модель.

Рисунок 1.9 – Тренажер-симулятор водіння

Один із прикладів – симулятори пілотування, які використовуються при навчанні пілотів. Зазвичай це тривимірні і досить реалістичні самовчителі. Вони допомагають початківцям – пілотам відпрацювати базові навички управління літальними апаратами в різних умовах: при поганій погоді, в темну пору і так далі. Їх основне завдання – допомогти пілоту перемогти страх, придбати базові реакції на позаштатні ситуації і відпрацювати на практиці вивчені в теорії правила пілотування. Вони дозволяють відпрацьовувати сценарії реагування і, головне, терпіти невдачу, яка не тягне за собою катастрофічних наслідків, втрату життя, матеріальних цінностей.

Рисунок 1.9 – Навчання пілотування гвинтокрилу

Крім того, навчальні симуляції вчать «виходити за рамки», мислити креативно і пробувати нове, не ризикуючи життям і технікою. Наприклад, пілот може навчитися виконувати фігури, які він не наважився б відразу спробувати в реальності.

Вони виявляються особливо корисні, коли тренуватися «в природних умовах», не можна через дорожнечу або потенційну небезпеку такого випробування.

1.5 Постановка задачі

В даному розділі було проаналізовано предметну область дослідження, розглянуто призначення, область застосування та види безпілотних літальних апаратів, наведено класифікацію та складові частини дронів. Було розглянуто значимість симуляторів. На основі розглянутого матеріалу було зроблено висновок, що робота з написанням тренажеру по управлінню квадрокоптером є доцільною.

Метою роботи є розробка та програмна реалізація тренажеру для надання навичок управління квадрокоптера. Для досягнення поставленої мети необхідно виконати такі завдання:

1. Розробка концепції інтерактивного тренування:
    a. створення логіки та правил інтерактивного тренування
    b. проектування механіки інтерактивного тренування
    c. створення дизайну інтерактивного тренування.
2. Програмна реалізація інтерактивного тренування:
    a. вибір програмного середовища
    b. імплементація логіки та правил інтерактивного тренування
    c. програмна реалізація польотної математики квадрокоптера
    d. реалізація дизайну оточення
3. Тестування системи

2 ПРОЕКТУВАННЯ ВІРТУАЛЬНОЇ СИСТЕМИ

2.1. Польотна математика квадрокоптера

Квадрокоптер складається з двох пар обертових роторів і гвинтів, розташованих у вершині квадратної рами. Він здатний виконувати вертикальний зліт і посадку (VTOL), подібно до типових гвинтокрилам. Однак система управління між гвинтокрилами та квадрокоптерами суттєво відрізняється від одного до іншого за рахунок динаміки польоту відповідно.

Використовуючи цю систему координат в якості основи нашого квадрокоптера , як показано на рисунку 2.1, двигуни фізичний розподіл двигунів був прикріплений за принципом «×» конфігурації.

Весь квадрокоптер складається з двох двигунів, що обертаються за годинниковою стрілкою, і двох двигунів, що обертаються проти годинникової стрілки, на вершині квадратного каркаса, як видно на рисунку 2.1. Рух у просторі досягається варіацією кінцевої сили і крутного моменту на кожній осі. Для досягнення правильної зміни крутного моменту та сили в системі, мотори налаштовані на роботу в парах, як показано в таблиці 2.2 M1, M2, M3 і M4 відповідно вважаються Мотор–1, Мотор–2, Мотор–3 і Мотор –4.

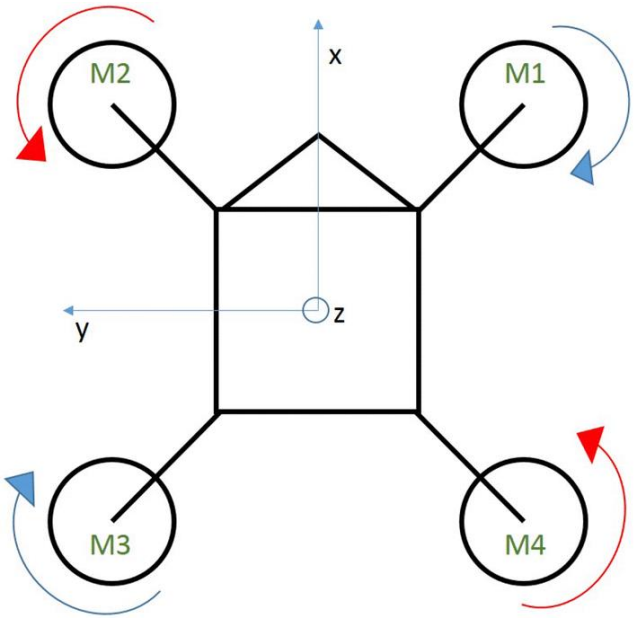


Рисунок 2.1 – Складова квадрокоптера

Таблиця 2.1 – Парна робота двигунів

	Газ				Крен				Тангаж				Рискання			
Мотори	1	2	3	4	1	2	3	3	1	2	3	4	1	2	3	4

Поступальний рух квадрокоптера вимагає нахилу платформи до потрібної осі. Опорний кут квадрокоптера задається креном ϕ, тангажем θ, рисканням ψ умовами руху в трьох вимірах (3D), див. Рис. 2.3.

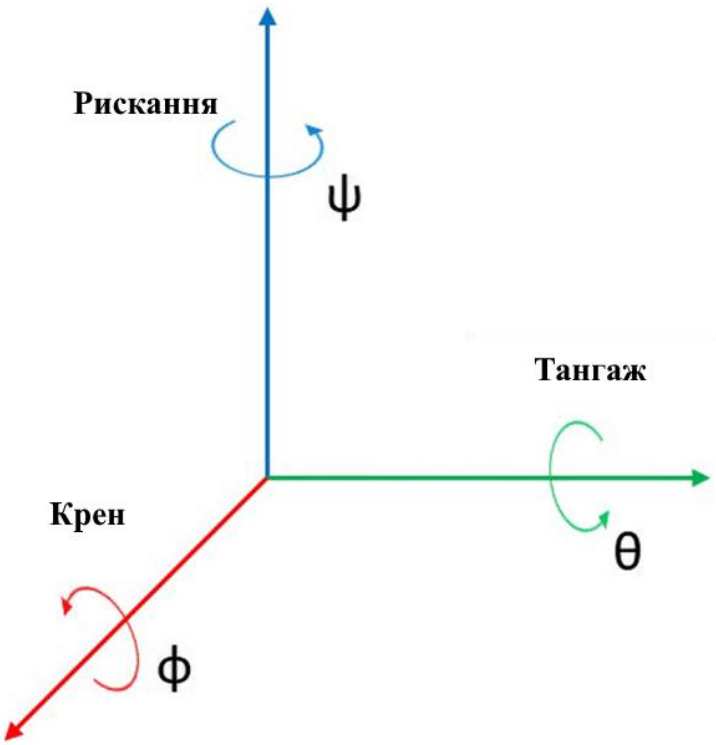


Рисунок 2.3 – Позначення для руху в 3D

Виходячи з розташування моторів, описаного в табл. 2.2, просто потрібно змінити швидкість руху одной з пар двигунів, щоб викликати рух у шести ступенях свободи (DOF). Це причина, що дозволяє квадрокоптеру рухатися в шести DOF і керувати, використовуючи лише чотирма входами [14]

2.1.1 Пересування

Динаміка польоту квадрокоптера заснована на співвідношенні між кутовою швидкістю двигунів, тобто силою підйому (F_t) і кінцевим крутним

моментом (τ_t), виробленою двигунами і прикладеною до квадрокоптера. Чотири рухи досягаються при зміні F_t і τ_t:

1. По вертикалі 'z' Він утворюється шляхом підсумовування всіх сил ротора на осі z.
2. Тангаж 'θ' Цей рух досягається маніпулюванням остаточним крутним моментом навколо осі y. Це робиться за допомогою зміни швидкості передніх роторів проти задніх роторів (збільшення або зменшення швидкості відповідно до бажаного руху).
3. Крен 'ϕ' Принцип цього руху такий самий, як і для тангажа, але нова опорна вісь – x. Лівий і правий ротори забезпечують рух з креном.
4. Рискання 'ψ' Рух рискання створюється реактивним моментом роторів. Цей рух вводиться різними швидкостями обертання, застосованими до пари двигунів проти обертання. Це викликає різницю крутного моменту між ними і надає остаточний крутний момент навколо осі z.

2.1.2 Математичне моделювання

Зазвичай для розробки математичної моделі більшості мобільних роботів використовуються два підходи [15]:

1) Рівняння Ньютона – Ейлера.
2) Рівняння Лангранжа.

У цій роботі було зосереджено на класичному математичну модель, заданому законами Ньютона; тому ми розробляємо нашу модель на основі рівнянь Ньютона – Ейлера. Виходячи з цих теорій, квадрокоптер повинен бути описаний як жорстке тіло, яке рухається через різні системи відліку на основі нерухомої системи координат(рисунок 2.4). Вони допомагають нам описати положення орієнтації літаючої платформи в просторі.

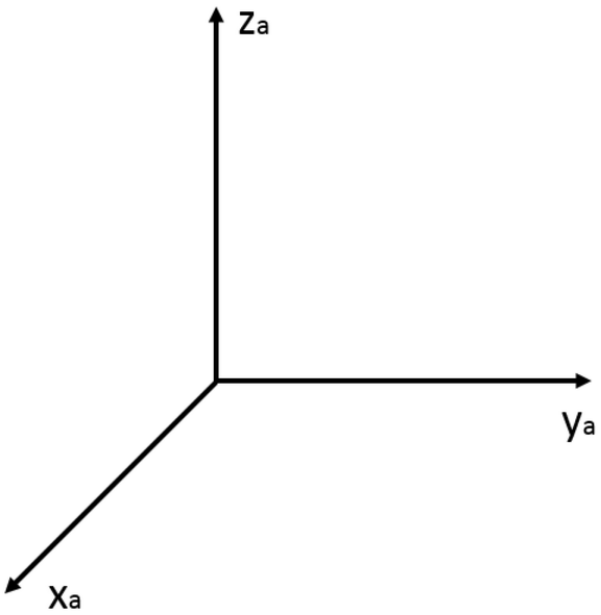


Рисунок 2.4 – Базова система координат

2.1.3 Координатна система

По–перше, слід розуміти, як рухається квадрокоптер у просторі (див. пункт 2.1.1) і які переміщення пов'язані з кожної еталонної системою координат. Вони допомагають нам зрозуміти, як об’єкти рухаються в просторі і як їх математично представляти. Далі представлені координатні системи, що описують динаміку квадрокоптера.

- Інерційна система F^i: нерухома система координат.
- Система апарата F^v: Початок координат є центром маси квадрокоптера і воно вирівняне з F^i.
- Система тіла F_b: F_v повертається на ϕ^+, див. рівняння 2.4.

Вектори на системі апарата F^v описуються матрицею, наведеною у рівнянні (2.1.)

$$F^v(1) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

Вектори системи координат апарату відповідають матриці F^v (див. Рис. 2.4), вважаються основою для аналізу наших контрольних системах. Початкова частина цього аналізу полягає в обертанні векторної системи 'F_v'

стосовно якоїсь конкретної осі, щоб створити наші математичні посилання для аналізу рухів у 3D–просторі.

- Система апарату–1 F^{v1}: Базується на F_v, але повертається у ψ^+.

$$R_z(\psi) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

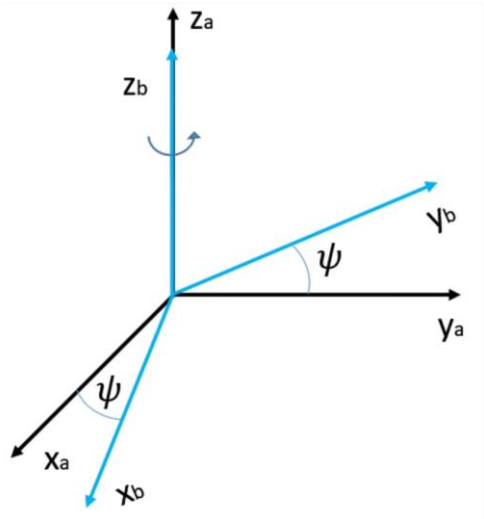


Рисунок 2.5 – Система апарату-1

- Система апарату–2 F^{v2}: Базується на F^{v1}, але обертається у θ^+.

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{2.3}$$

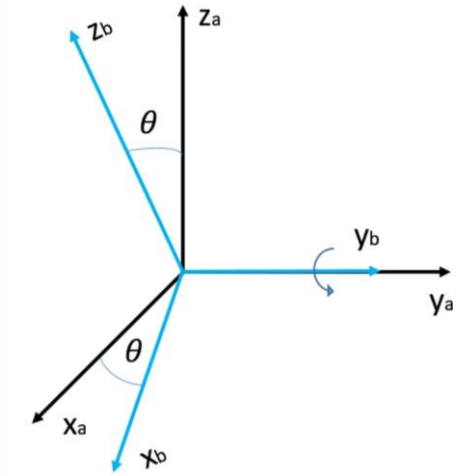


Рисунок 2.6 – Система апарату–2

- Система тіла F_b: заснований на F^{v2}, але обертається у ϕ^+.

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix} \quad (2.4)$$

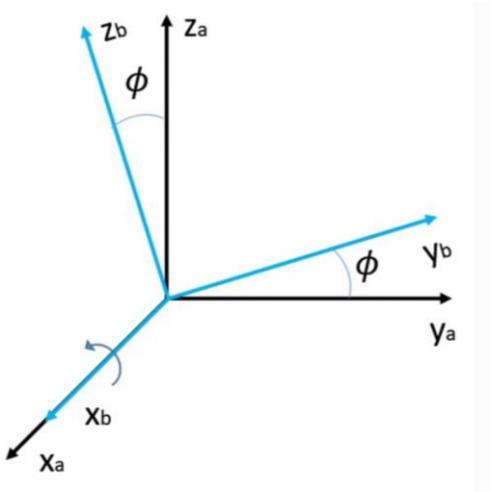


Рисунок 2.7 – Система апарату–1

Після того, як координатні системи були визначені, можна зобразити рухи від одной системи до іншої за допомогою двох операцій: обертання та трансляції [16].

2.1.4 Кінематичний та динамічний аналіз

Першим кроком є визначення лінійної швидкості в координатної системі апарату , як показано у рівнянні. 2.5

$$\frac{d}{dt}\begin{pmatrix} x^i \\ y^i \\ z^i \end{pmatrix} = R_b^v \begin{pmatrix} v_x^b \\ v_z^b \\ v_z^b \end{pmatrix} = \begin{pmatrix} v_x^i \\ v_z^i \\ v_z^i \end{pmatrix} \quad (2.5)$$

По–друге, ми визначаємо кутову швидкість типу $w_b = (p \quad q \quad r)^T$ і вважаємо, що діапазон виведення кута буде меншим або рівним ± 20 градусів; отже, можна використовувати критерії наближення малого кута.

Поєднуючи визначення описаної вище кутової швидкості та поняття, можна отримати кутову швидкість, виконавши кроки, задані рівняннями. 2.6, 2.7).

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = R_{v2}^{b}(\phi)\begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + R_{v2}^{b}(\dot{\phi})R_{v1}^{v2}(\dot{\theta})\begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + R_{v2}^{b}(\dot{\phi})R_{v1}^{v2}(\dot{\theta})R_{vv}^{v1}(\dot{\psi})\begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \quad (2.6)$$

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{pmatrix}\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \quad (2.7)$$

2.1.5 Динаміка твердого тіла

Другий закон Ньютона визначається рівнянням (2.8) для поступального руху.

$$m\frac{d^2x}{dt^2} = m\frac{dv}{dt_i} = m\,a = F \quad (2.8)$$

Застосування ефекту Коріоліса до рівняння (2.9), отримуємо:

$$m\frac{dv}{dt_i} = m(\frac{dv}{dt_i} + w_{b/i} \times v) = F \quad (2.9)$$

Для обчислення кутового прискорення в системі координат тіла необхідно застосувати 2–й закон Ньютона для обертального руху і включити ефект Коріоліса.

$$\frac{dH^b}{dt_i} = \frac{dH^b}{dt_i} + w_{b/i} \times H^b = \sum \tau^b \quad (2.10)$$

Для обчислення інертів вважається ідеальною моделлю, як спостерігається на рис. 2.8. Квадрокоптер передбачається сферичним щільним центром з масою M і радіусом R. Мотори моделюються як чотири точки мас, розташованих на відстані l від центру масою m. Якщо ці параметри зрозумілі, тепер можна обчислити відповідну інерцію всієї системи (рис. 2.8).

$$J_x = J_y = \frac{2MR^2}{5} + 2ml^2 \quad (2.11)$$

$$J_z = \frac{2MR^2}{5} + 4ml^2 \quad (2.12)$$

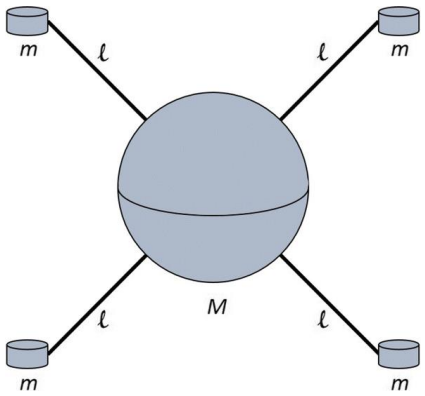


Рисунок 2.8 – Ідеальна модель квадрокоптера

2.1.6 Сили та моменти

Сили та моменти на квадрокоптері, описані на рисунку 2.9, зумовлені, в першу чергу, силою тяжіння та чотирма гвинтами.

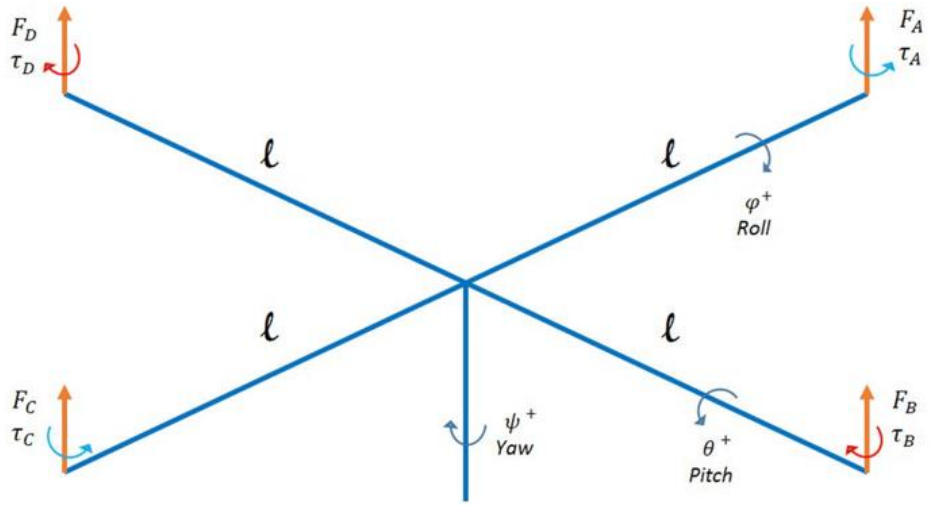


Рисунок 2.9 – Сили та момент на квадрокоптері (ідеальні)

Кожен двигун створює силу вгору F і крутний момент τ. Таким чином, загальна сила квадрокоптера задається сумою всіх сил ($F_t = \sum_{i=1}^{4} F_i$). Цей же принцип застосовується і для крутного моменту ($\tau_t = \sum_{i=1}^{4} \tau_i$). Враховуючи це, визначаються крутні моменти кочення, розбивання та похитування.

$$\begin{pmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} l(F_d - F_b) \\ l(F_d - F_b) \\ (\tau_b + \tau_d) - (\tau_a + \tau_c) \end{pmatrix} \quad (2.13)$$

Сила тяжіння також чинить силу на систему координат транспортного засобу *Fv* квадрокоптера, і вона просто впливає на напрямок *z*. Однак, щоб співставити цю силу з еталоном, наведеним на корпусі *Fb*, необхідно помножити на відповідну матрицю обертання.

$$F_g^v = R_v^b F_g^b \rightarrow F_g^v = \begin{pmatrix} -mg\sin(\theta) \\ mg\cos(\theta)\sin(\phi) \\ mg\cos(\theta)\cos(\phi) \end{pmatrix} \quad (2.14)$$

2.2. Дизайн системи управління PID

Для систем контролера квадрокоптера важливо врахувати, наскільки міцною повинна бути система щодо зовнішніх сил, таких як вітер або зіткнення. Зазвичай сила, викликана вітром, може бути постійною або змінною (поступова чи ні). З цієї причини важливо врахувати, наскільки швидкою є реакція квадрокоптера щодо зовнішніх сил, що діють за короткий проміжок часу.

Перший контролер відповідає за стабілізацію кута. Ідея полягає в тому, щоб мати контролер якомога швидше і стабільніше. Для цього реалізовано контролер P.

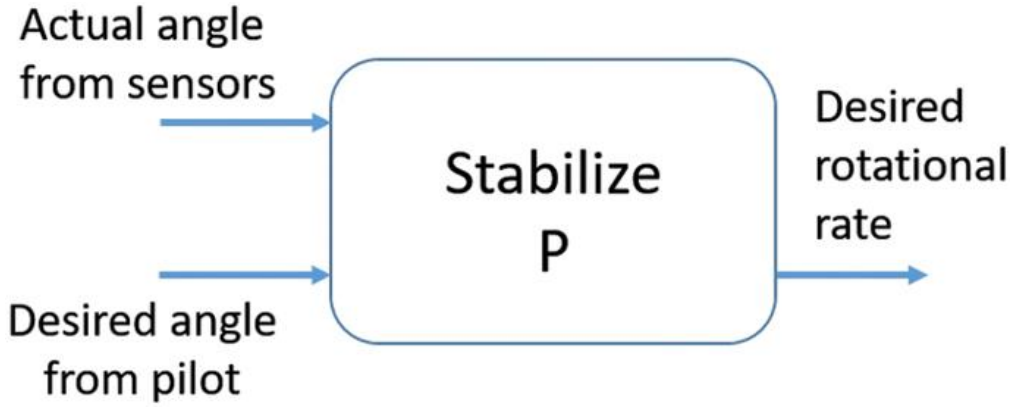


Рисунок 2.10 – Контролер P

Опорний сигнал для контролера подається пілотом (або функцією) і вважається бажаним кутом руху на основі системи координат квадрокоптера. Вхід зворотного зв'язку задається датчиками, у цьому випадку значенням

зворотного зв'язку вхідного сигналу є фактичний кут квадрокоптера. Вихід буде розглядатися як бажана швидкість обертання, яка буде входом для наступної частини керуючої системи.

Наступний крок – зробити систему стабільною проти зовнішніх сил. З цієї причини розглядаються зміни швидкості обертання квадрокоптера з метою компенсації небажаних змін (поступових чи ні). PID–контролер вибирається завдяки перевагам, які задаються інтегральною та похідною частинами, та простоті в майбутній реалізації коду. Введений вхід цього контролера – це прямий вихід контролера стабільності.

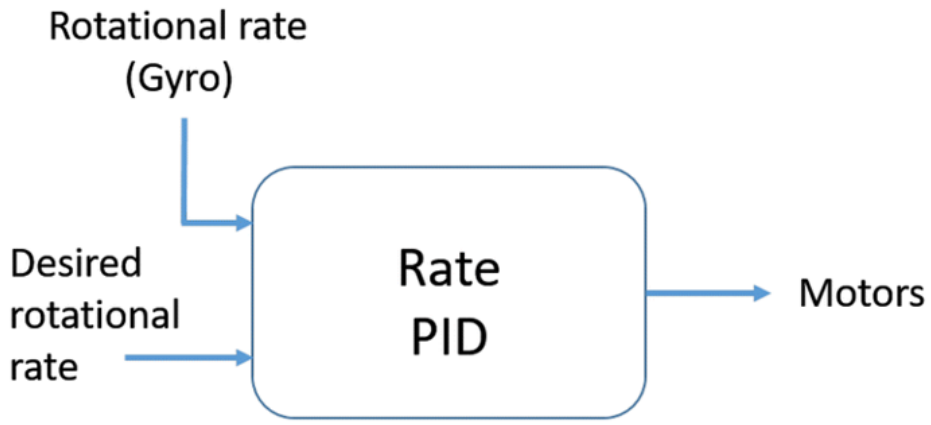


Рисунок 2.11 – Контролер PID

Мета – повернути квадрокоптер з певною швидкістю (заданою градусами в секунду) у будь–якому з можливих напрямків (котиться ϕ, крок θ, позіхає ψ). Вхід зворотного зв'язку подається безпосередньо з датчиків. Ця реалізація вимагає, щоб введення зворотного зв'язку було в градусах за секунду, щоб відповідати бажаній частоті обертання, яка має цю одиницю. Вихід контролера направляється безпосередньо на двигуни.

Весь контролер, рис. 2.12, – це склад регулятора стійкості та швидкості в каскадному режимі (одна гілка на кут). Ідея полягає в тому, що коли перший контролер досягне потрібного кута, його вихід буде нульовим; таким чином, вхід для контролера швидкості буде дорівнює нулю, і квадрокоптер не матиме обертального руху. З іншого боку, коли різниця першого контролера по

відношенню до керуючого сигналу не дорівнює нулю, другий контролер матиме ненульовий вхід, а квадрокоптер матиме обертальний рух.

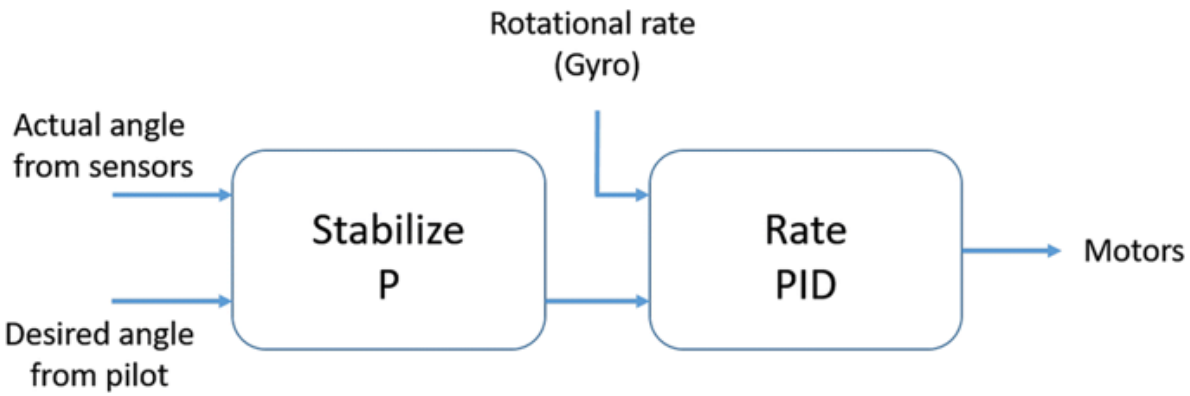


Рисунок 2.12 – Зв'язок контролерів P та PID

Контролер стабілізації (рис. 2.10) вимагає реалізації P–контролера, тоді як регулятор швидкості (рис. 2.11) вимагає PID–контролера. Або P, або PID–контролер, обидва мають однакову структуру коду. Різниця задається параметрами контролера, де *kp*, *ki* і *kd* визначають тип контролера (P–PI–PD–PID).

2.2.1 Работа PID регулятора

PID – це функція в польотному контролері. Ця функція зчитує дані з датчиків і передає двигунів, як швидко їм потрібно обертатися. В кінцевому підсумку, саме так досягається стабільність і ідеальність польоту квадрокоптера. PID позначає похідну пропорційного інтеграла. ПІД–регулятор являє собою замкнуту систему управління, яка намагається отримати фактичний результат ближче до бажаного результату, внісши корективи у вихідні дані, які відправляються двигунів. Якщо відбувається помилка, вона повертається в початок і цикл повторюється. ПІД–регулятор обчислює значення «помилка» як різниця між вимірюваною величиною і бажаної величиною. Контролер намагається звести до мінімуму помилку, відрегулювати надходять значення управління. Працює це так: в квадрокоптера PID отримує дані з датчиків і порівнює їх з поданими даними.

Різниця між цими дані називається «помилка» або «error» по–англійськи і намагається зменшити в подальшому цю помилку.

В ПІД–регулятор є три функції: P, I і D рисунок 2.11 . Ці значення можуть бути інтерпретовані з точки зору часу:

- P обробляє помилку, яка відбувається в даний момент – чим далі вона від заданого значення, тим сильніше вона коригується
- D – коригує майбутні помилки, дивлячись на те, як швидко ви наближаєтеся до заданої точки та наскільки відбувається протидія P при наближенні до заданої точки
- I – Аналізує минулі помилки, які відбуваються протягом якогось часу (наприклад, якщо вісь постійно зміщується від заданого значення через вітер) і протидіє цій силі збільшуючи швидкість потрібних двигунів.

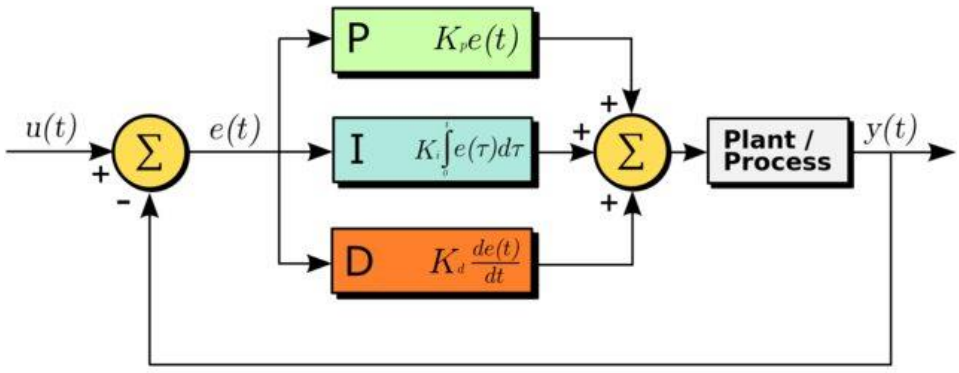


Рисунок 2.13 – Схема PID регулятора

Теоретичні методи аналізу системи з ПІД–регулятором рідко застосовуються на практиці. Основна складність практичного застосування – незнання характеристик об'єкта управління. Крім того, істотну проблему представляють не лінійність і не стаціонарність системи. Практичні регулятори працюють в обмеженому зверху і знизу діапазоні, тому в принципі нелінійні. У зв'язку з цим набули поширення методи експериментальної настройки регулятора, підключеного до об'єкта управління. Пряме

використання формованої алгоритмом керуючої величини також має свою специфіку.

Для ручної настройки PID – параметрів потрібно багато практики. Є різноманітні аналітичні методи їх обчислення і аналізу, але вони потребують кваліфікованої технічної підготовки та предметне знання великої кількості параметрів конкретної системи, яка настроюється. Різними дослідниками пропонується численний ряд емпіричних методів – середнє між аналітичним обчисленням та ручним підбором.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Вибір програмного середовища

Як середовище для розробки програми було обрано програмний рушій Unity 2019.3.7f1. Для редагування та компілювання коду – Visual Studio 2019. На сьогоднішній час, існує багато середовищ для розробки програмного забезпечення. Вибрані середовища зарекомендували себе як самі прогресивні та ефективні для розробки. Сама Unity має інструменти з якими дуже комфортно працювати, не витрачаючи багато зусиль на освоювання.

На відміну від багатьох ігрових движків, у Unity є три основні переваги: наявність візуального середовища розробки, міжплатформенна підтримка і модульна система компонентів. Перший фактор включає не тільки інструментарій візуального моделювання, а й інтегровану середу, що направлено на підвищення продуктивності розробників. Міжплатформенна підтримка надається не тільки місцями розгортання (установка на персональному комп'ютері, на мобільному пристрої, консолі і інші), але і наявність інструментарію розробки (інтегроване середовище може використовуватися під Windows і Mac OS). Модульна система компонентів за допомогою якої відбувається конструювання ігрових об'єктів, коли останні являють собою комбіновані пакети функціональних елементів.

Проект в Unity ділиться на сцени – окремі файли, що зберігають об'єкти, сценарії і їх конфігурації. Движок використовує компонентно–орієнтований підхід. Вся сцена складається з об'єктів типу GameObject. Це порожній універсальний об'єкт, до якого додаються компоненти, реалізовані скриптами поведінки (MonoBehaviour), також є вже готові компоненти вбудовані в движок. В GameObject можна додати безліч компонентів, наприклад: графічний компонент (для відтворення геометрії), звуковий компонент (щоб об'єкт міг видавати звук), компонент Rigidbody(відтворює фізику об’єктів), Navmesh (реалізує пошук шляху), компонент анімації(оживляє об’єкти) та інші. Цей об'єкт ми можемо зберегти в префаб (це файл, який зберігає

характеристики GameObject і його дочірні об'єкти). В наслідок цього, потім можемо повторно використовувати, модифікувати сам префаб, і ці зміни позначається на всіх подібних об'єктах. Щоб зробити зручну інтеграцію між класом та ігровою сценою, існує клас MonoBehaviour. Від нього можна спадкувати базові методи Start(визивається перший кадр), Update (визивається кожен кадр), OnEnable(визивається коли об'єкт включений) та інші. Це дозволяє розподілити і впорядкувати код, що робить його більш читабельним і зрозумілим.

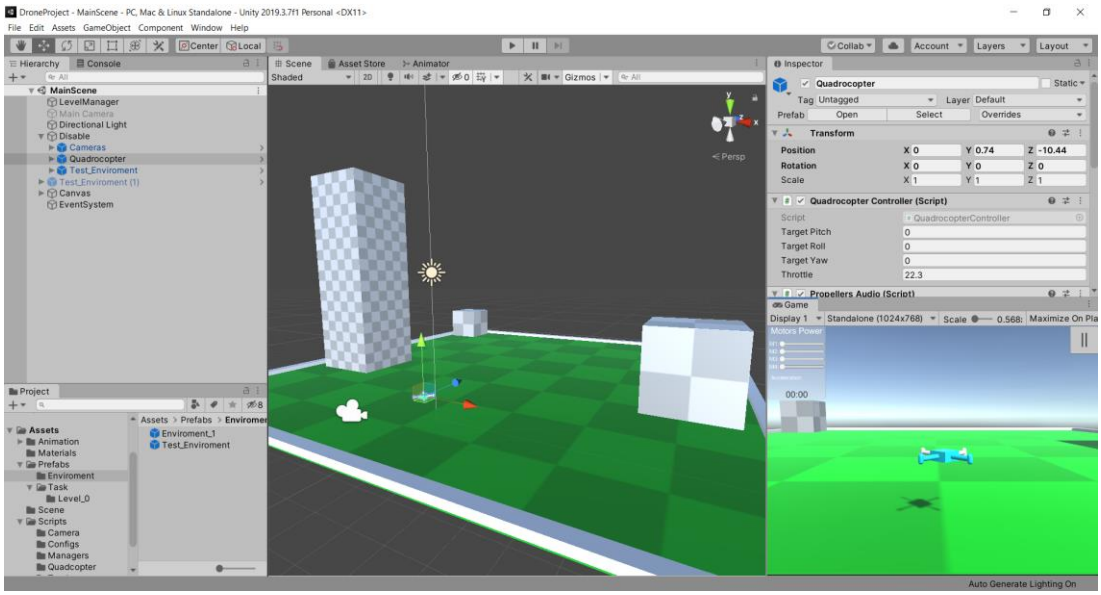
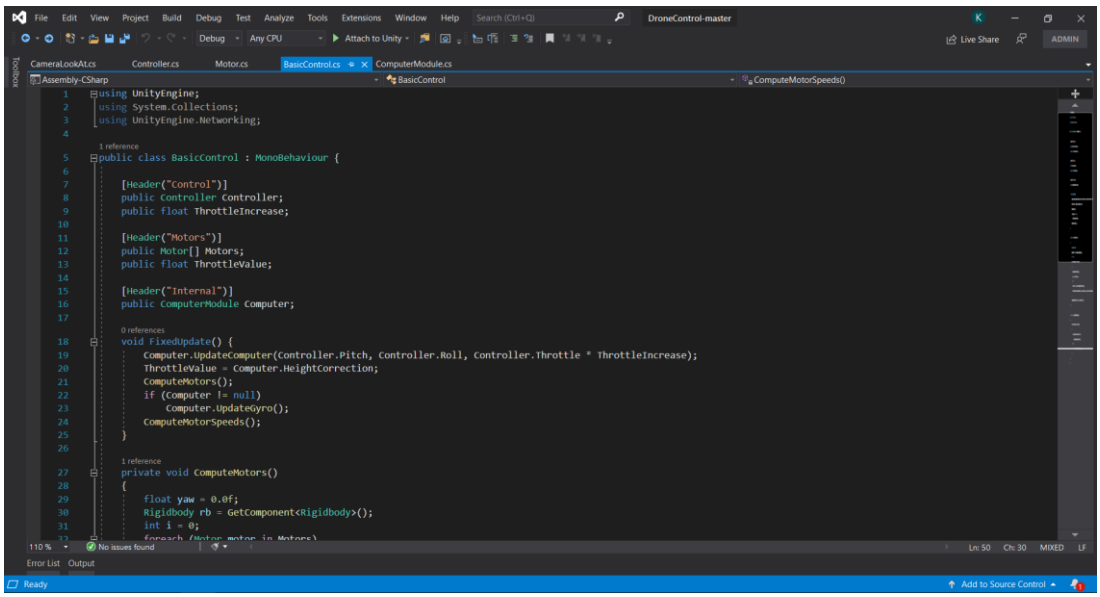


Рисунок 3.1 – Зовнішній вигляд середовища розробки.



```
using UnityEngine;
using System.Collections;
using UnityEngine.Networking;

public class BasicControl : MonoBehaviour {

    [Header("Control")]
    public Controller Controller;
    public float ThrottleIncrease;

    [Header("Motors")]
    public Motor[] Motors;
    public float ThrottleValue;

    [Header("Internal")]
    public ComputerModule Computer;

    void FixedUpdate() {
        Computer.UpdateComputer(Controller.Pitch, Controller.Roll, Controller.Throttle * ThrottleIncrease);
        ThrottleValue = Computer.HeightCorrection;
        ComputeMotors();
        if (Computer != null)
            Computer.UpdateGyro();
        ComputeMotorSpeeds();
    }

    private void ComputeMotors()
    {
        float yaw = 0.0f;
        Rigidbody rb = GetComponent<Rigidbody>();
        int i = 0;
```

Рисунок 3.2 – Зовнішній вигляд редактору додатку.

3.2 Короткий опис програмної реалізації

Процес навчання в тренажері-симуляторі починається з запуску рівня. Учень освоює управління і виконує поставлені завдання за відведений час. Як тільки всі завдання виконані успішно, пропонується перейти на наступний рівень. Якщо учень втратить із виду керований об'єкт або втратить управління, рівень можна перезапустити. Також якщо завдання не були виконані, пропонується повторити спробу.

Рисунок 3.3 – Блок-схема процес навчання

З блок–схеми 3.3, можна виділити робочий цикл тренажера, розділивши на три етапи:

- Етап підготовки, передбачає собою створення тренувального оточення, керованого об'єкта (квадрокоптера), візуальних позначок (міток), настройки конфігурації аудіосупровіда та камери в залежності від обраного рівня;
- Етап тренування включає в себе виконання завдань. Щоб пройти рівень, учневі потрібно досягти всіх візуальних позначок, які розподілені на відповідному рівні. Наприклад згідно з рівнем,

активування дрону, підйом, початок переміщення, польот за направленням до вибраної мітки, маневреність між візуальними позначками за визначений час;

- Етап підсумків, оповіщає учня що той може перейти на новий рівень (для підвищення складності) або почати спочатку минулий рівень (для поліпшення своїх показників та виправлення помилок). Також в цей етап входить і інший випадок, коли учень не зміг пройти рівень, йому пропонується повторити спробу.

3.3 Формування тренувального простору

Діаграма класів є ключовим елементом в об'єктно–орієнтованому моделюванні. Діаграма класів проекту подана на рисунку 3.4

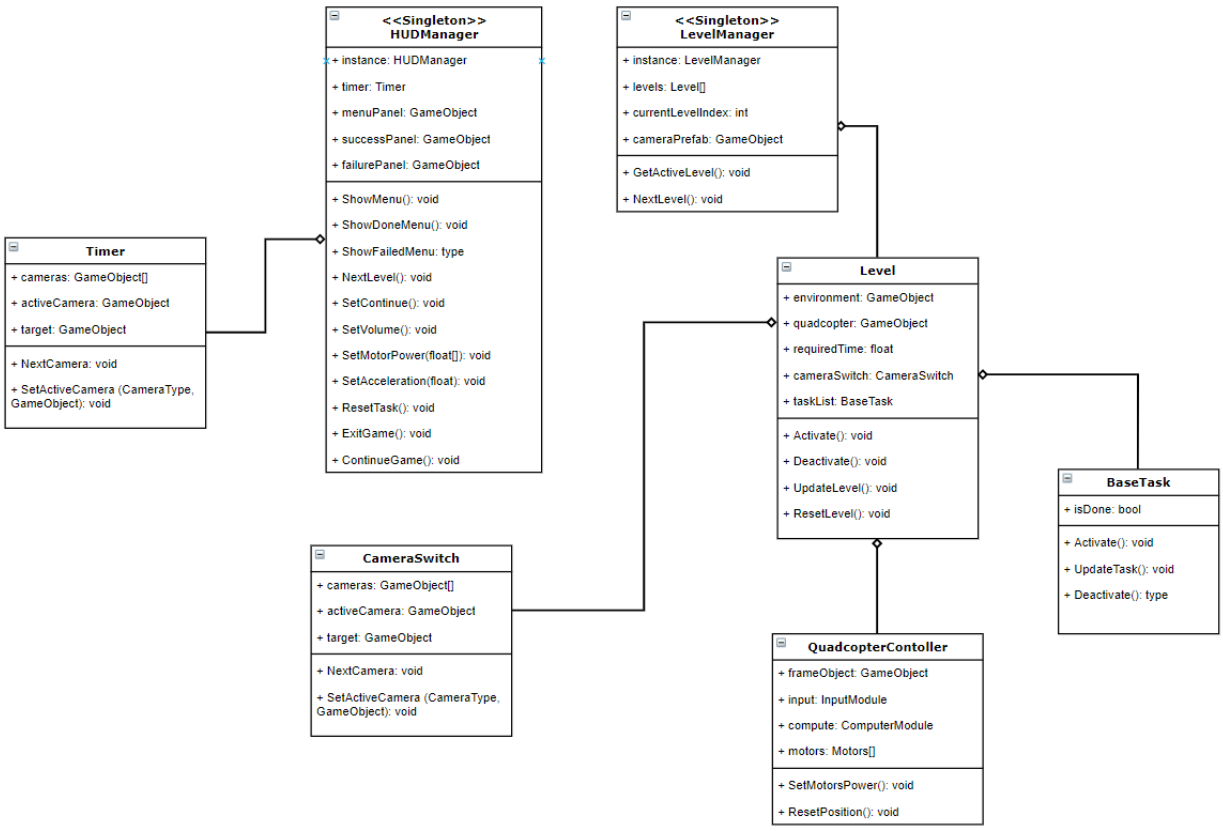


Рисунок 3.4 – Діаграма класів

Було реалізовано дві частини: логіка тренажера і призначений для користувача інтерфейс.

Класи які представляли логіку тренажера:

- LevelManager. Зберігає в собі префаби рівнів, створює камеру. Цей класс стежить за станом, активує, деактивує і оновлює клас Level. У разі виконання всіх завдань переводить на новий рівень.
- Level. Зберігає в собі префаб квадрокоптера, оточення, камери, а також список завдань. Клас стежить за активацією, оновленням, скиданням рівня. В активації створює об'єкти оточення і конфігурує камеру і аудіо компонент. В оновленні стежить за виконанням завдання. Деактивування і скидання, відповідно, стежить за видаленням і переініціалізацією.
- CameraSwitch. Зберігає в собі префаб камер (FPV, TPV, CCTV). Клас відповідає за настройку і активації вибраної камери.
- BaseTask. Це абстрактний клас, який дозволяє створювати конкретні завдання. Наприклад, клас PositionTask, він дозволяє стежити за дорожнім маркером в тренажері.

Класи які представляли інтерфейс тренажера:

- HUDManager. Зберігає в собі основні панелі (вікна). Клас визначає і конфігурує вікна, стежить за малюванням, призначеного для користувача, інтерфейсу і взаємодіями користувача з панелями.
- Timer. Зберігає в собі час в хвилинах і секундах, а також оновлює таймер кожен кадр.

Також можна особливо виділити клас QuadcopterController рисунок 3.5. Завдяки йому користувач може взаємодіяти з ігровим оточенням. Цей клас можна розділити на 3 основні частини:

- InputModule. Клас для зчитування вводу користувача.
- Motor. Клас симулює роботу мотора, створює фізичну підйомну, крутний силу.
- ComputerModule. Цей клас розраховує підйомну силу (PID) в залежності від вводу користувача і поточного стану апарату (GyroModule).

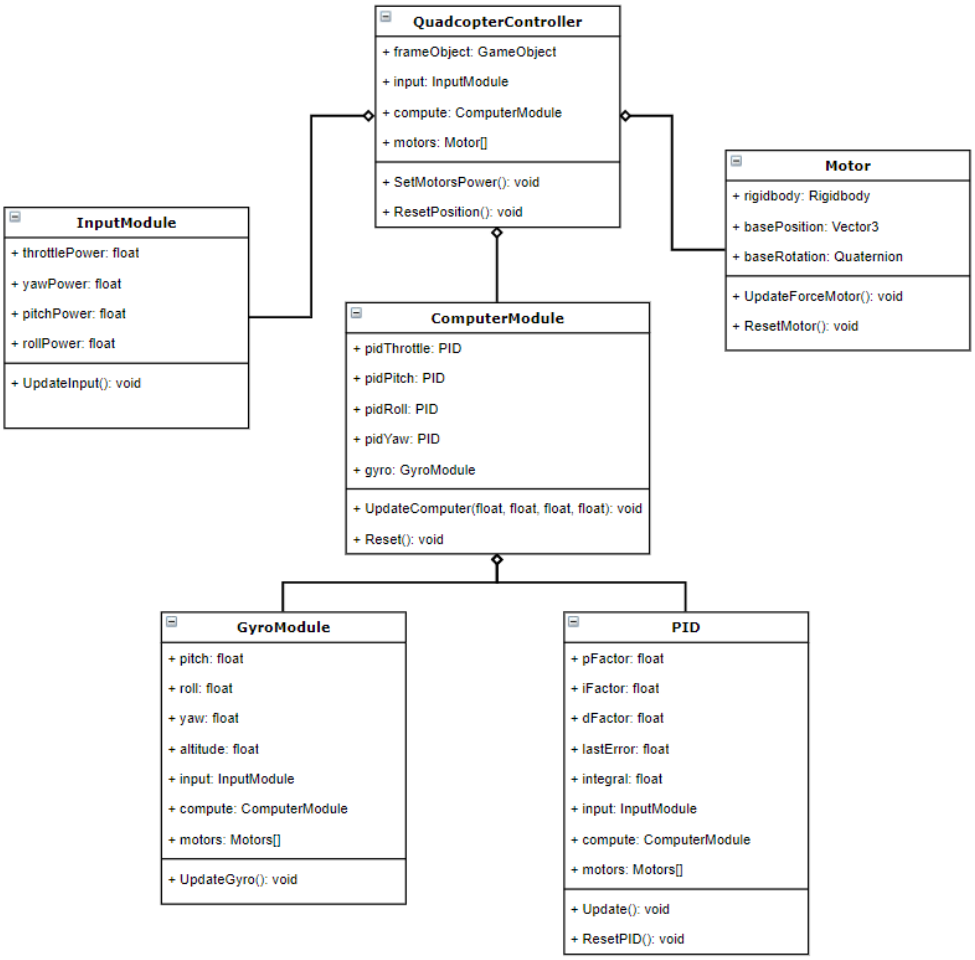


Рисунок 3.5 – Діаграма QuadcopterController

3.4 Тестування програмного забезпечення

Для перевірки працездатності системи була створена нова тренувальна сесія. Задачею першого рівня є навчання управлінням, дається зрозуміла по положенню камера(TPV) та багато часу.

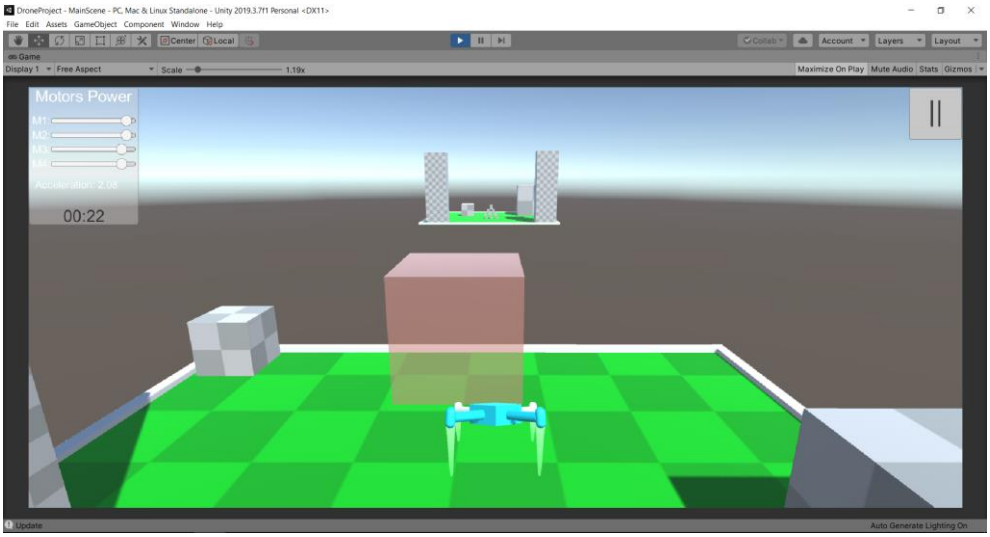


Рисунок 3.5 – Екран гри на початку тренування

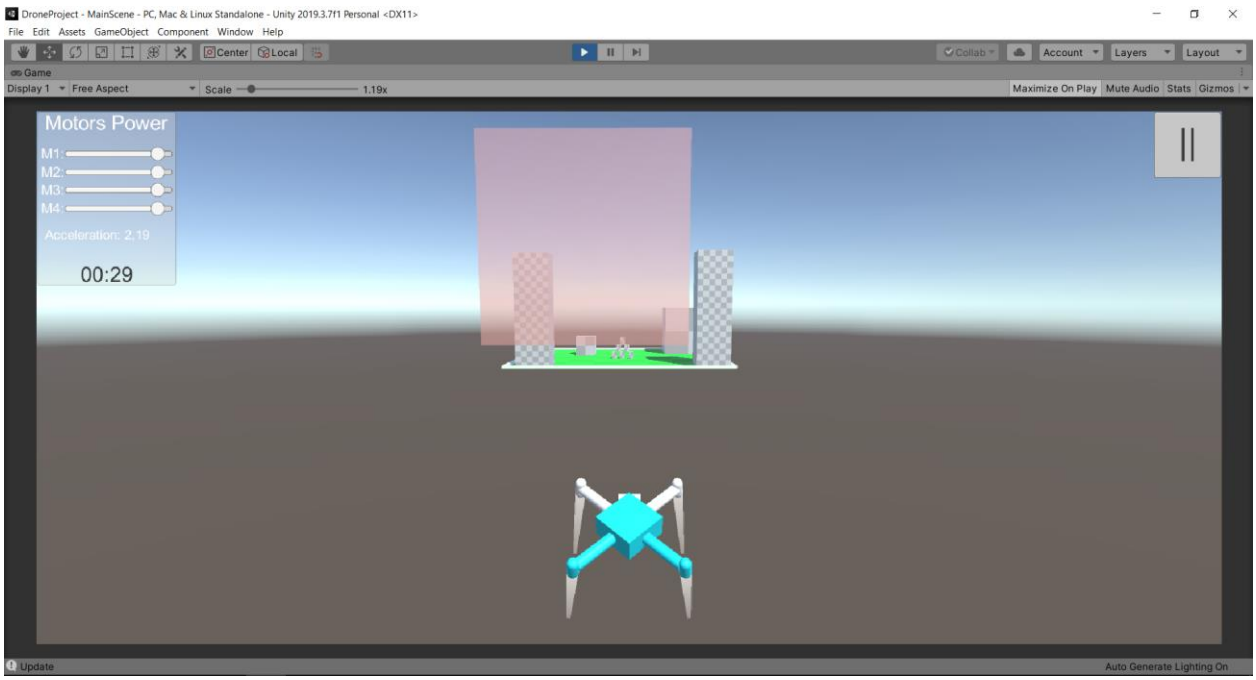


Рисунок 3.6 – Політ до наступного маркеру

Коли ми добираємося до останньої мітки, з’являється вікно, яке інформує про здачу рівня. Тепер ми можемо перейти на наступний рівень.

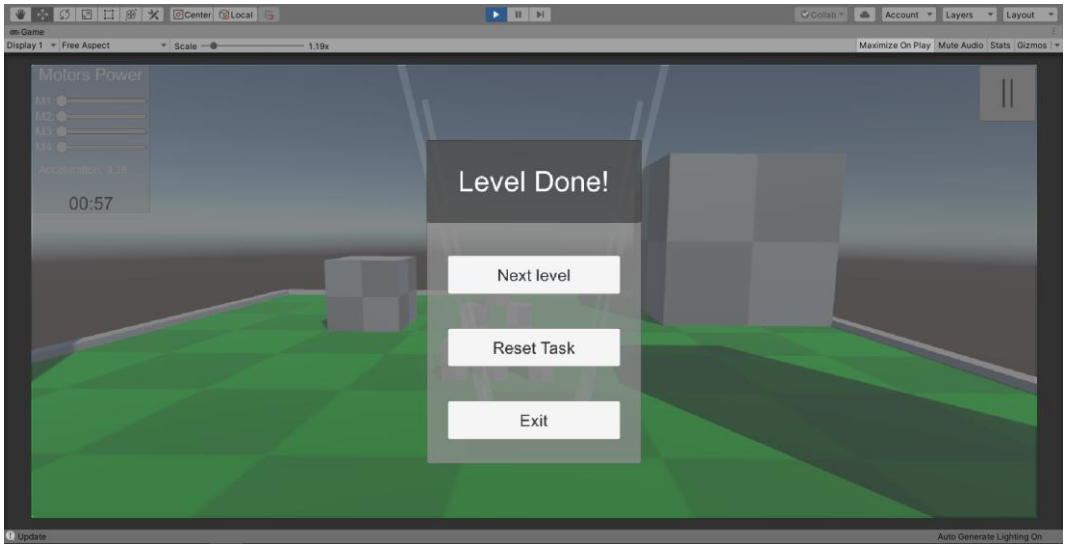


Рисунок 3.7 – Вікно позитивного проходження

Задачею другого рівня є навчання маневрування між стінами, дається другий вид камери (FPV) та достатньо часу.

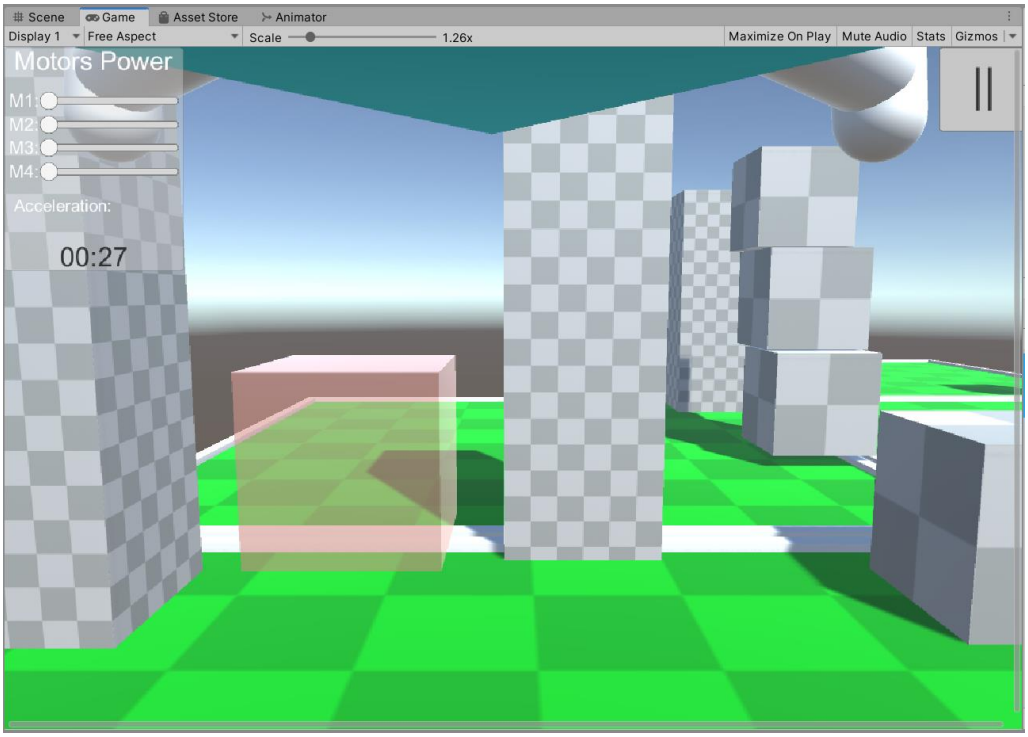


Рисунок 3.8 – Приклад польоту

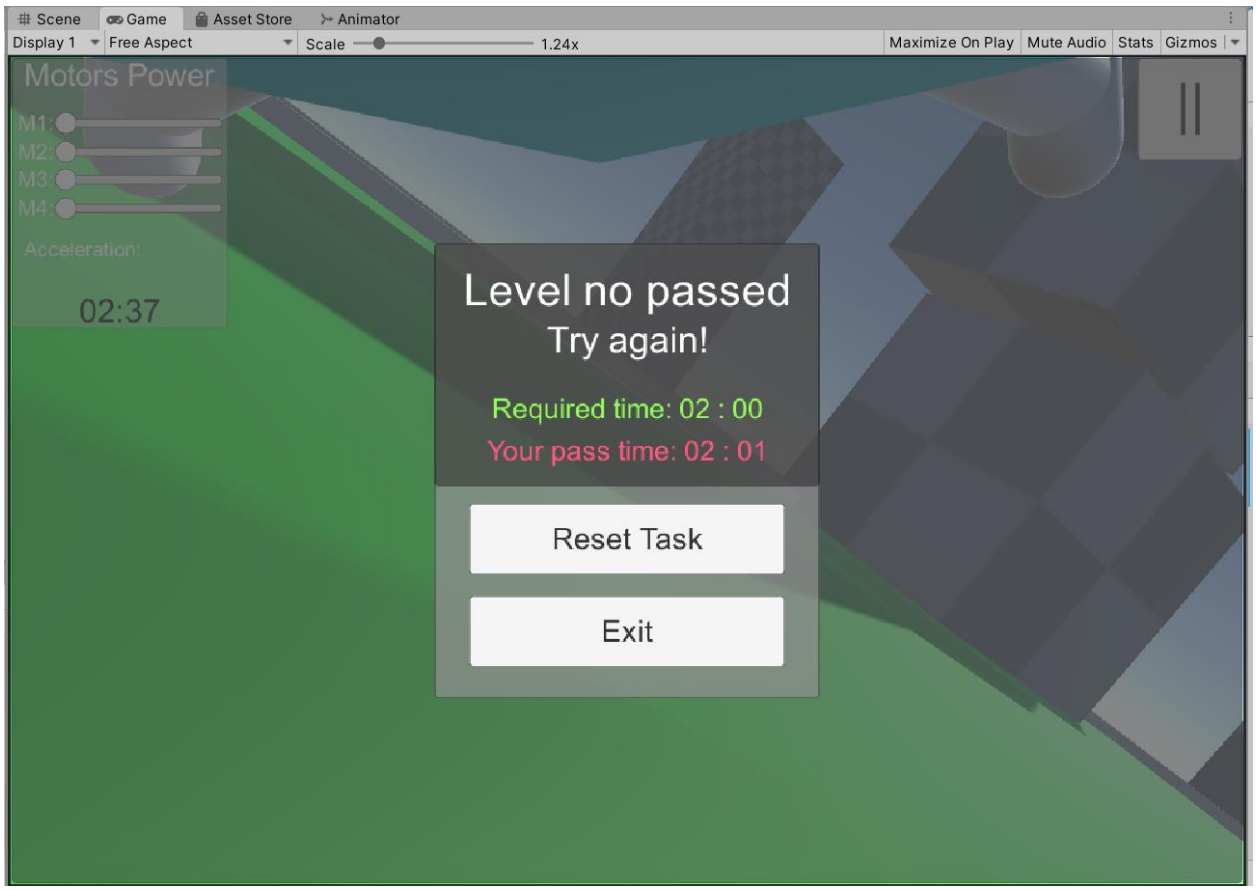


Рисунок 3.9 – Вікно негативного проходження

При виконанні тестування були вирішені наступні задачі: отримання досвіду управління та орієнтації в просторі, поліпшення навичок управління і координації, отримання нового досвіду за допомогою нового положення камери.

ВИСНОВКИ

В випускній роботі було виконано розробку інформаційного та програмного забезпечення системи. На основі проведеного математичного аналізу роботи квадрокоптеру було розроблено концепцію інтерактивного тренування, спроектовано механіки та прототип дизайну тренажеру. Для програмної реалізації системи використовувалося середовище розробки Unity, що дозволило виконати якісну імплементацію концепції інтерактивного тренування та розроблених алгоритмів, спростити процес виробництва контенту та дизайну тренажеру.

Для перевірки працездатності системи було проведено її закрите бетта-тестування. Сам тренажер-симулятор показав себе з позитивної сторони. Дрон на тренажері симуляторі був легкокеруємим, може орієнтуватися в просторі та виконувати поставлену мету. Він чуйний в управлінні. В цій роботі було доказано, що підготовка на симуляторі, підготувала учня до наступної роботи безпосередньо з самим фізичним дроном.

СПИСОК ЛІТЕРАТУРИ

1. Історія і майбутнє квадрокоптера і дронів – Режим доступу: https://quadrone.ru/blog/stati/istoriya–i–budushchee–kvadrokopterov
2. Професія: оператор безпілотних літальних апарати (БПЛА) – Режим доступу: https://proforientator.ru/publications/articles/professiya–operator–bespilotnykh–letatelnykh–apparatov–bpla.html
3. Кращі FPV симулятори гоночного квадрокоптера – Режим доступу: https://profpv.ru/luchshie–fpv–simulyatory–gonochnogo–kvadrok/
4. Як військові дрони підвищують ефективність сільського господарства – Режим доступу: https://smartfarming.ua/ua–blog/tysyachi–gektarov–za–den–kak–voennye–drony–povyshayut–effektivnost–selskogo–hozyajstva
5. Дрони в законі: як використовують безпілотники в Німеччині – Режим доступу: https://www.dw.com/uk/%D0%B4%D1
6. Як вибрати відповідний вам квадрокоптер з камерою – Режим доступу: https://www.djimsk.ru/guides/2018/08/14/buy–drone/
7. Довідник з протиповітряної оборони / А.Я. Торопчин, І.О. Романенко, Ю.Г. Даник та ін. –Харків: ХВУ, 2003. – 368 с.
8. Інженерно-авіаційне та аеродромно-технічне забезпечення авіації – Режим доступу: http://www.hups.mil.gov.ua/assets/doc/science/stud–conf/konferencii–kursantiv–ta–studentivnaukovo–praktichna–konferentsiya–hnups–25–27–zhovtnya–2017–roku/chastina2/6.pdf
9. Drone technology uses and applications for commercial, industrial and military drones in 2020 and the future – Режим доступу: https://www.businessinsider.com/drone-technology-uses-applications
10. Історія розвитку дронів – Режим доступу: https://dronomania.ru/faq/istoriya-razvitiya-dronov.html
11. Types of multirotor – Режим доступу: https://oscarliang.com/types-of-multicopter/
12. Anatomy of a multirotor drone – Режим доступу: https://www.dronetrest.com/t/anatomy-of-a-multirotor-drone/1386

13. Anatomy of A Drone - What's inside a DJI Phantom Drone – Режим доступу: https://www.dronefly.com/the-anatomy-of-a-drone
14. Padfield Gareth D (2008) Helicopter flight dynamics: the theory and application of flying qualities and simulation modeling. Blackwell Pub. Oxford, ISBN: 978–14051–1817–0.
15. Alexander Lebedev (2013) Design and implementation of a 6DOF control system for an Autonomous Quadrocopter. Universität Würzburg, Würzburg Print
16. Diebel James (2018) Representing attitude: Euler angles, unit quaternions, and rotation vectors. Stanford University. – Режим доступу: http://www.swarthmore.edu/NatSci/mzucker1/papers/diebel2006attitude.pdf Last access: 31 July 2018
17. CHRobotics (2013) AN–1005 Understanding Euler Angles. Article. Vol. Rev 1.1. – Режим доступу: http://www.chrobotics.com/docs/AN–1005–UnderstandingEulerAngles.pdf

ДОДАТОК

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System;

public class QuadrocopterController : MonoBehaviour
{
	private GameObject frameObject;
	private Rigidbody frameRigidbody;

	private InputModule input;
	private ComputerModule computer;

	private Motor motor1;
	private Motor motor2;
	private Motor motor3;
	private Motor motor4;

	private bool buttonDown;

	public readonly float maxThrottle = 30f;
	public readonly float stepPower = 0.2f;

	public float targetPitch;
	public float targetRoll;
	public float targetYaw;

	public float throttle;

	public void Start()
	{
		motor1 = transform.Find("Motor1").GetComponent<Motor>();
		motor2 = transform.Find("Motor2").GetComponent<Motor>();
		motor3 = transform.Find("Motor3").GetComponent<Motor>();
		motor4 = transform.Find("Motor4").GetComponent<Motor>();

		frameObject = transform.Find("Frame").gameObject;
		frameRigidbody = frameObject.GetComponent<Rigidbody>();

		input = new InputModule();
		computer = frameObject.GetComponent<ComputerModule>();
	}

	private void Update()
	{
		input.UpdateInput();

		if((Mathf.Abs(input.throttlePower) == 1) && !buttonDown)
		{
			throttle += maxThrottle * stepPower *
input.throttlePower;
			throttle = Mathf.Clamp(throttle, 0, 30);
			buttonDown = true;
		}
		else if(input.throttlePower == 0)
		{
```

```
                buttonDown = false;
            }

            targetYaw += input.yawPower * 3;
            targetPitch += input.pitchPower;
            targetRoll += input.rollPower;

            targetPitch = Mathf.Clamp(targetPitch, -45, 45);
            targetRoll = Mathf.Clamp(targetRoll, -45, 45);
            targetYaw = targetYaw > 360 ? targetYaw - 360 : targetYaw;

            if (input.isReset)
            {
                ResetPosition();
            }
        }

        private void FixedUpdate()
        {
            computer.UpdateComputer(targetPitch, targetRoll, targetYaw,
throttle, Time.fixedDeltaTime);
            SetMotorsPower();

            var powers = new float[]
            {
                computer.motorPower1 / maxThrottle,
                computer.motorPower2 / maxThrottle,
                computer.motorPower3 / maxThrottle,
                computer.motorPower4 / maxThrottle,
            };
            HUDManager.instance.SetMotorPower(powers);


        HUDManager.instance.SetAcceleration(computer.currentAcceleration
Smoothed);
        }

        private void SetMotorsPower()
        {
            motor1.UpdateForceMotor(computer.motorPower1);
            motor2.UpdateForceMotor(computer.motorPower2);
            motor3.UpdateForceMotor(computer.motorPower3);
            motor4.UpdateForceMotor(computer.motorPower4);
        }

        public void ResetPosition()
        {
            throttle = 21.6f;
            targetPitch = 0;
            targetRoll = 0;
            targetYaw = 0;

            frameRigidbody.velocity = Vector3.zero;
            frameRigidbody.angularVelocity = Vector3.zero;

            computer.Reset();

            frameRigidbody.isKinematic = true;
```

```
            frameObject.transform.position = Vector3.one;
            frameObject.transform.localRotation = Quaternion.identity;

            motor1.ResetMotor(Vector3.one);
            motor2.ResetMotor(Vector3.one);
            motor3.ResetMotor(Vector3.one);
            motor4.ResetMotor(Vector3.one);

            frameRigidbody.isKinematic = false;
        }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LevelManager : MonoBehaviour
{
    public static LevelManager instance;

    [SerializeField]
    private List<Level> levels;
    private bool pause;

    public GameObject camera;
    public int currentLevelIndex;

    private void Start()
    {
        if (instance == null)
        {
            instance = this;
        }

        camera = Instantiate(camera);

        currentLevelIndex = 0;
        if (levels.Count > 0)
        {
            levels[currentLevelIndex].Activate();
        }
    }

    private void Update()
    {
        if ( levels.Count > 0 && !isPause)
        {
            if (levels[currentLevelIndex].isDone)
            {
                if (levels[currentLevelIndex].RequiredTime != 0 &&
levels[currentLevelIndex].RequiredTime >=
HUDManager.instance.timer.minutes)
                {
                    HUDManager.instance.ShowDoneMenu();
                }
                else
                {
                    HUDManager.instance.ShowFailedMenu();
                }

                isPause = true;
                return;
            }

            Debug.Log("Update");
            levels[currentLevelIndex].UpdateLevel();
        }
    }
```

```
    public Level GetActiveLevel()
    {
        return levels[currentLevelIndex];
    }

    public void NextLevel()
    {
        levels[currentLevelIndex].Deactivate();
        currentLevelIndex = currentLevelIndex + 1 == levels.Count ? 0 :
currentLevelIndex + 1;
        levels[currentLevelIndex].Activate();
    }

    public bool isPause
    {
        get
        {
            return pause;
        }
        set
        {
            pause = value;
        }
    }
}

using System;
using UnityEngine;
using UnityEngine.UI;

public class HUDManager : MonoBehaviour
{
    public static HUDManager instance;

    public Timer timer;

    public GameObject menuPanel;
    public GameObject successPanel;
    public GameObject failurePanel;

    [Space]
    public Slider powerM1;
    public Slider powerM2;
    public Slider powerM3;
    public Slider powerM4;

    [Space]
    public Text acceleration;

    [Space]
    public Text soundTxt;
    public Text requiredTimeTxt;
    public Text yourTimeTxt;

    private bool isSound;

    private void Start()
    {
```

```
        if(instance == null)
        {
            instance = this;
        }
    }

    public void ShowMenu()
    {
        PauseGame();

        menuPanel.SetActive(true);
    }

    public void ShowDoneMenu()
    {
        PauseGame();

        successPanel.SetActive(true);
    }

    public void ShowFailedMenu()
    {
        PauseGame();

        requiredTimeTxt.text = requiredTimeTxt.text + " 0" +
LevelManager.instance.GetActiveLevel().RequiredTime + " : 00";
        yourTimeTxt.text = yourTimeTxt.text + " " + timer.ToString();

        failurePanel.SetActive(true);
    }

    public void NextLevel()
    {
        ContinueGame();

        LevelManager.instance.NextLevel();

        successPanel.SetActive(false);
        timer.Reset();
    }

    public void SetContinue()
    {
        menuPanel.SetActive(false);

        ContinueGame();
    }

    public void SetVolume()
    {
        if (!isSound)
        {
            soundTxt.text = "Sound On";
            AudioListener.volume = 0;
        }
        else
        {
            soundTxt.text = "Sound Off";
```

```
            AudioListener.volume = 1;
        }

        isSound = !isSound;
    }

    public void SetMotorPower(float[] powers)
    {
        powerM1.value = Mathf.Lerp(powerM1.value, powers[0], 0.1f);
        powerM2.value = Mathf.Lerp(powerM2.value, powers[1], 0.1f);
        powerM3.value = Mathf.Lerp(powerM3.value, powers[2], 0.1f);
        powerM4.value = Mathf.Lerp(powerM4.value, powers[3], 0.1f);
    }

    public void SetAcceleration(float value)
    {
        if (value < 0.001) value = 0;

        acceleration.text   =   String.Format("Acceleration:   {0:f2}",
value);
    }

    public void ResetTask()
    {
        LevelManager.instance.GetActiveLevel().ResetLevel();

        timer.Reset();
    }

    public void ExitGame()
    {
        Application.Quit();
    }

    public void PauseGame()
    {
        AudioListener.volume = 0;
        Time.timeScale = 0;

        LevelManager.instance.isPause = true;
    }

    public void ContinueGame()
    {
        AudioListener.volume = 1;
        Time.timeScale = 1;

        LevelManager.instance.isPause = false;
    }
}

using System;

[Serializable]
public class PID {
     public float pFactor;
     public float iFactor;
     public float dFactor;
```

```
	private float lastError;
	private float integral;

	public PID (float P, float I, float D)
	{
		pFactor = P;
		iFactor = I;
		dFactor = D;
	}

	public float Update (float current, float target, float timeFrame)
	{
		float present = target - current;
		integral += present * timeFrame;
		float deriv = (present - lastError) / timeFrame;
		lastError = present;
		float force = present * pFactor + integral * iFactor + deriv
* dFactor;
		if ((force > -0.1) && (force < 0.1))
			force = 0;
		return force;
	}

	public void ResetPID()
	{
		lastError = 0;
		integral = 0;
	}
}

using UnityEngine;
public class GyroModule
{
	public float pitch; // The current pitch for the given transform
	public float roll; // The current roll for the given transform
	public float yaw; // The current Yaw for the given transform
	public float altitude; // The current altitude from the zero
position

	public void UpdateGyro(Transform transform)
	{
		pitch = transform.eulerAngles.x;
		pitch = (pitch > 180) ? pitch - 360 : pitch;

		roll = transform.eulerAngles.z;
		roll = (roll > 180) ? roll - 360 : roll;

		yaw = transform.eulerAngles.y;
		yaw = (yaw > 180) ? yaw - 360 : yaw;

		altitude = transform.position.y;
	}
}
using System;
using UnityEngine;
using UnityStandardAssets.Cameras;
```

```
public enum CameraType { FPV = 0, TPV, CCTV };

public class CameraSwitch : MonoBehaviour
{
    public GameObject[] objects;
     public GameObject activeCamera;
     public GameObject target;

    private int m_CurrentActiveObject;

     public void Update()
     {
          bool next = Input.GetKeyDown(KeyCode.C);
          if (next) {
               NextCamera ();
          }
     }

     public void SetActiveCamera(CameraType type, GameObject target)
     {
          for (int i = 0; i < objects.Length; i++)
          {
               objects[i].SetActive(false);
          }
         activeCamera = objects[0];
          switch (type)
          {
               case CameraType.TPV:
                    activeCamera = objects[0];

     activeCamera.GetComponent<AutoCam>().SetTarget(target.transform)
;
                    activeCamera.transform.parent = target.transform;
                    break;
               case CameraType.FPV:
                    activeCamera = objects[1];
                    break;
               case CameraType.CCTV:
                    activeCamera = objects[2];

     activeCamera.GetComponent<TargetFieldOfView>().SetTarget(target.
transform);

     activeCamera.GetComponent<LookatTarget>().SetTarget(target.trans
form);
                    break;
          }
          activeCamera.SetActive(true);

          this.target = target;
          activeCamera.transform.localPosition                    =
target.transform.position + new Vector3(0, 0.07f, 0.32f);
     }

     public void NextCamera()
     {
          int  nextactiveobject  =  m_CurrentActiveObject  +  1  >=
objects.Length ? 0 : m_CurrentActiveObject + 1;
```

```
            for (int i = 0; i < objects.Length; i++)
            {
                  objects[i].SetActive(i == nextactiveobject);
            }

            m_CurrentActiveObject = nextactiveobject;
      }
}
```