МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут інноваційних освітніх технологій
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Савченко А.С.
«___»_____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТРА"
ЗА СПЕЦІАЛІЗАЦІЄЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ (ЗА ГАЛУЗЯМИ)"

Тема: «Побудова траєкторії польоту зліт/посадка на платформі UNITY»

Виконавець: Лінник Олександр Сергійович

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер: Райчев І. Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут інноваційних освітніх технологій

Кафедра комп'ютерних інформаційних технологій

Галузь знань 12 «Інформаційні технології»

Спеціальність 122 «Комп'ютерні науки»

Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Савченко А.С.

“ ” _____2020р.

З А В Д А Н Н Я

на виконання дипломної роботи студента

Лінника Олександра Сергійовича

(прізвище, ім'я, по батькові випускника в родинному відмінку)

1. **Тема роботи:** «Побудова траєкторії польоту зліт/посадка на платформі UNITY» затверджена наказом ректора від 06.10.2020р. № 1939/ст.

2. Термін виконання роботи: з 05.10.2020 р. по 21.12.2020 р.

3. Вихідні данні до роботи: Програма побудови траєкторії польоту зліт/посадка на платформі UNITY.

4. Зміст пояснювальної записки: вступ, об'єкт дослідження та постановка задачі, розбір основних властивостей повітряного судна ті його характеристики на різних етапах польоту, розгляд платформи UNITY для написання програми по побудові траєкторії польоту зліт/посадка.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: 3D модель повітряного судна, ілюстрація траєкторії польоту ПС «по колу», ілюстрації роботи програми по побудові траєкторій зліт/посадка.

5. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1	Виконати детальний аналіз літератури та Інтернет джерел за темою дипломної роботи. Розробити план дипломної роботи.	05.10.2020р. – 10.10.2020р.	
2	Затвердити план дипломної роботи. Написати та затвердити вступ до дипломної роботи Провести консультації з науковим керівником, щодо першого розділу.	11.10.2020р. – 15.10.2020р.	
3	Розробка розділу 1: Об'єкт дослідження та постановка задачі	16.10.2020р. – 18.10.2020р.	
4	Розробка розділу 2: Етапи польоту по колу. Характеристики повітрянного судна під час польоту.	19.10. 2020р. – 29.10.2020р.	
5	Розробка розділу 3: Unity як середовище розробки.	30.10.2020р. – 10.11.2020р.	
6	Написати висновки до виконаного дипломного проекту. Оформити пояснювальну записку. Підготовити доповідь та презентацію.	11.11.2020р. – 21.11.2020р.	
7	Підписати необхідні документи у встановленому порядку. Підготуватися до захисту дипломного проекту.	05.12.2020р.– 20.12.2020р.	

6. Дата видачі завдання: 05.10.2020р.

Керівник дипломної роботи _____ Холявкіна Т.В.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Лінник О.С.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Побудови траєкторії польоту зліт/посадка на платформі UNIT» викладена на 88 сторінках. Містить 60 рисунків, 2 таблиці, 26 джерел.

Метою роботи є створення програми для побудови траєкторій зльоту посадки та демонстрація польоту БПЛА по колу на платформі UNITY, використовуючи мову програмування C#.

Наукова новизна полягає у використанні платформи UNITY в розробці або тестуванні певних ситуацій які виникають в авіаційній сфері.

Практична цінність результатів дипломної роботи включає в себе аналіз траєкторій руху БПЛА по колу та розробка програми по побудові траєкторій.

Для вирішення поставлених завдань в роботі використовувалися такі методи:

1) метод системного аналізу;

2) метод ієрархічної оптимізації процесів;

3) метод дискретної математики.

Ключові слова: UNITY, БПЛА, ЗЛІТ, ПОСАДКА, ТРАЄКТОРІЯ, ГЛІСАДА, ТАНГАЖ, КРЕН, РИСКАННЯ, ПІДІЙМАЛЬНА СИЛА, СИЛА ТЕРТЯ, ЛОБОВИЙ ОПІР, КОЕФІЦІЄНТ ПІДІЙМАЛЬНОЇ СИЛИ, КУТ НАХИЛУ ГЛІСАДИ, МЕХАНІЗАЦІЯ КРИЛА, ПІДКРИЛКИ, ЗАКРИЛКИ, ЕЛЕРОНИ, КЕРМО ВИСОТИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ..... 7

ВСТУП..... 8

РОЗДІЛ 1. ОБ'ЄКТ ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ..... 10

1.1. Особливості БПЛА 11
1.2. Типи БПЛА..... 13
1.3. Область реалізації обраного об’єкта дослідження “MQ-9“ Reaper ” ... 16
1.4. Основні технічні характеристики “MQ-9“ Reaper ”..... 17
1.5. Особливості планування траєкторій БПЛА 24

РОЗДІЛ 2. ЕТАПИ ПОЛЬОТУ ПО КОЛУ. ХАРАКТЕРИСТИКИ ПС ПІД ЧАС ПОЛЬОТУ..... 27

2.1. Зліт 27
2.2. Політ по колу 31
2.3. Захід на посадку та глісада..... 35
2.4. Механізація крила та хвостового оперення літака 38

РОЗДІЛ 3. UNITY ЯК СЕРЕДОВИЩЕ РОЗРОБКИ..... 53

3.1 Загальна характеристика..... 53
3.2. Особливості середовища в розробці авіасимуляції 54
3.2.1. Vector2 54
3.2.2. Vector3 54
3.2.3. Rigidbody 55
3.2.4. Rigidbody.rotation..... 57
3.2.5. Rigidbody.AddForce 57
3.2.6. GameObject..... 58
3.2.7. Кватерніони та кути Ейлера 59
3.2.8. Вікно Inspector 61
3.2.9. Unity Assets..... 63
3.2.10. Сцени 65
3.2.11. Колайдери..... 67
3.2.12. Камера..... 69
3.2.13. Префаби..... 70

РОЗДІЛ 4. РОЗРОБКА ПРОГРАМИ ДЛЯ ПОБУДОВИ ТРАЄКТОРІЇ ПОЛЬОТУ ЗЛІТ/ПОСАДКА НА ПЛАТФОРМІ UNITY 73

4.1. Інтеграція 3D-моделі MQ-9 Reaper в Unity..... 73

4.2. Контроль механізації крила та хвостового оперення за допомогою Unity. 76
4.3. Рух по колу. Програмування траєкторій польоту руху по колу. Зліт та посадка. 79

ВИСНОВКИ 88

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 89

Додаток А. Скрипт поведінки MQ9 92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

БПЛА	Безпілотний літальний апарат
ПС	Повітряне судно
API	Application program interface

ВСТУП

Безпілотні літальні апарати (БПЛА) стають все більш корисними та цінними для завдань та місій, в яких участь людини є неможливою, оскільки це є небезпечним або достатньо складним. Серед цих транспортних засобів військові безпілотні літальні апарати стають дедалі популярнішим об'єктом для вивчення БПЛА з багатьох точок зору, таких як розвідка, реле зв'язку, бойові завдання тощо. Оптимізація траєкторії руху БПЛА, що стосується розвитку часу траєкторії польоту, є дуже важливою частиною системи автономного управління БПЛА.

У більшості випадків планування траєкторії та планування шляху дуже схожі. Їх можна назвати оптимізацією траєкторії. Але в реальному сенсі справи ці дві речі насправді відрізняються. Планування траєкторії - це процес, при якому безпілотний літальний апарат знаходить тривимірний (3D) космічний шлях від вихідної точки до пункту призначення. 3D космічний шлях - це статичний геометричний шлях. Він не включає поняття часу. Однак результатами процесу планування траєкторії є шляхи польоту, що змінюються в часі. Результати включають стан польоту транспортних засобів. Як правило, модель та алгоритм рішення проблеми планування траєкторії складніші, ніж алгоритми задачі планування траєкторії. Однак багато ідей з алгоритмів планування шляху можуть допомогти вирішити проблему планування безпілотного літального апарату. Для багатьох простих сценаріїв простіші алгоритми планування траєкторії можуть швидко та ефективно запропонувати деякі схематичні та плавні результати польоту для автоматичної системи управління безпілотного літального апарата. Обидва вони дуже важливі для автономної системи безпілотного літального апарата.

Проблема планування та оптимізації траєкторії безпілотного літального апарата широко виникає у багатьох умовах, що включають спостереження,

пошук, рятувальні місії, географічні дослідження, військові та охоронні програми.

Завданням даної роботи є вирішення проблеми оптимізації спільної траєкторії руху безпілотного літального апарата MQ-9 Reaper. Головне - знайти більш ефективні, більш реалістичні та неоптимальні траєкторії для безпілотних літальних апаратів, не роблячи занадто багато спрощуючих припущень щодо траєкторій. Але на цьому шляху виникає багато проблем. Перша проблема полягає у відсутності розумних систематичних досліджень загальних систем вирішення проблем. Інша проблема полягає в тому, що для спрощення проблеми існує занадто багато припущень щодо спрощення моделей літаків. Зіткнувшись із цими проблемами, для вирішення цієї задачі оптимізації траєкторії висувається ієрархічна стратегія оптимізації, заснована на офлайн-процесі планування шляху та процесі онлайн-планування траєкторії. Ця ієрархічна стратегія оптимізації може покращити ефективність рішення, підвищити фактичність результату та забезпечити результати в реальному часі.

Метод ієрархічної оптимізації включає процес планування шляху та процес планування траєкторії. Для того, щоб зробити процес планування траєкторії більш ефективним та більш реальним, вводиться процес автономного планування шляху для завершення окремих попередніх робіт з планування точок траєкторії. Динамічні та кінематичні обмеження безпілотного літального апарату в цьому процесі просто включають обмеження швидкості та нормальне обмеження перевантаження. Результати моделювання покажуть, що алгоритм стратегії ієрархічної оптимізації ефективний для проблеми оптимізації траєкторії безпілотного літального апарату.

Достовірність та обгрунтованість результатів, отриманих в дипломної роботи, забезпечується відповідністю розроблених моделей та алгоритмів відомим теоретичним результатам і реальним процесам польоту ПС.

РОЗДІЛ 1
ОБ'ЄКТ ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

За сучасним визначенням, «безпілотником» є тільки той апарат, який знаходиться під постійним дистанційним контролем пілота або пілотів і призначений для повернення на аеродром і для подальшого повторного використання.

Раніше радіокеровані та повністю автоматизовані апарати об'єднували поняттям безпілотна авіація — літаки, керування (пілотування) якими здійснюється без пілота, за допомогою приладів різних систем, що засобами радіо (радіолокації, телебачення) подають команди на автопілот. Елементи системи керування містяться поза літаком і можуть бути на землі, на воді і в повітрі, на місці старту, на маршруті польоту і в районі цілі. Для передачі на пункт управління даних, отриманих з бортових сенсорів, у складі БПЛА є радіопередавач, що забезпечує зв'язок з наземним обладнанням. Залежно від формату зображень та їхнього стискання, регламентованих, наприклад, в STANAG 4609, швидкість передавання цифрових радіоканалів зв'язку з БПЛА, може становити одиниці-сотні Мбіт/с. Перед передаванням з борту БПЛА отриманих зображень високої чіткості, їх піддають сегментації.

Об’єктом дослідження у даній роботі є безпілотний літальний апарат, який здійснює зліт або посадку на ВПП і або має на своєму шляху певні перешкоди, або має відхилення від курсу. Отже постає проблема в розрахунку необхідної траєкторії польоту або зльоту. Ця проблема ділиться на два процеси.

Перший процес - це раннє офлайн-планування шляху, яке обчислюється за допомогою інтелектуальних алгоритмів оптимізації. У цьому процесі проблема перетворюється на проблему планування шляху, метою якої є отримання ряду оптимізованих точок траєкторії для наступної задачі планування траєкторії. Офлайн-результати можуть допомогти розділити всю траєкторію на кілька невеликих сегментів траєкторії. Таким чином, величезна проблема планування може бути вирішена кожною невеликою проблемою планування, обсяг розрахунку та складність планування якої значно менші, ніж у вихідної задачі. У той же час, завдяки стратегії ієрархічної оптимізації, регулювання відстані між кожними двома пунктами допомагає контролювати тривалість обчислювального часу. Іншими словами, це може допомогти реалізувати запит онлайн-планування. Крім того, оскільки невелика сцена для проблеми планування малої траєкторії може покращити частку можливого рішення у визначеній області незалежних змінних, здатність проблеми планування малої траєкторії в реальному часі додатково покращується.

Згодом другий процес - це онлайн-планування траєкторії. Проблема планування траєкторії, яка є оптимальною проблемою управління, може бути вирішена, як правило, двома підходами, а саме, непрямим підходом та прямим підходом. Прямими методами є перетворення безперервної задачі оптимального управління в нелінійну задачу оптимізації параметрів [1].

1.1. Особливості БПЛА

Технологія автономії, яка стане важливою для майбутньої розробки БПЛА, підпадає під наступні категорії:

• Злиття датчиків: поєднання інформації від різних датчиків для використання на борту транспортного засобу;

• Зв'язок: обробка зв'язку та координація між кількома агентами за наявності неповної та недосконалої інформації;

• Планування руху (також зване плануванням шляху): Визначення оптимального шляху руху транспортного засобу при виконанні певних цілей та обмежень, таких як перешкоди;

• Генерування траєкторії: Визначення оптимального маневру управління, яким потрібно пройти, щоб пройти заданий шлях або перейти з одного місця в інше;

• Розподіл завдань та планування: Визначення оптимального розподілу завдань між групою агентів з обмеженнями часу та обладнання;

• Тактика співпраці: Формулювання оптимальної послідовності та просторового розподілу діяльності між агентами з метою максимізації шансів на успіх у будь-якому даному сценарії місії.

Автономію зазвичай визначають як здатність приймати рішення без втручання людини. Тобто метою автономії є навчити машини бути «розумними» та діяти більше як люди.

Певною мірою кінцевою метою розвитку технології автономії є заміна пілота-людини. Залишається зрозуміти, чи майбутні розробки технології автономії, сприйняття технології, а головне, політичний клімат навколо використання такої технології обмежать розвиток та корисність автономії для застосування БПЛА.

Ця версія безпілотника здатна здійснити автоматичну посадку на іншому аеродромі у випадку несприятливих погодних умов, пошкодження посадкової смуги на основному аеродромі чи зміни місії. Інформацію щодо зміни аеродрому оператор може вносити під час польоту безпілотника.

Безпілотник може здійснити посадку навіть на незнайомому аеродромі, адже тепер він не потребує наявності наземної станції управління на запасному аеродромі, а буде використовувати засоби управління

супутниковим зв’язком. Для цього він спочатку вивчає місцевість за допомогою сенсорів, а згодом здійснює посадку в автоматичному режимі.

У General Atomics Aeronautical Systems зазначили, що завдяки вдосконаленням може бути зменшена кількість операторів, необхідна для управління безпілотником. Крім того, модернізація MQ-9A Reaper значно підвищує гнучкість безпілотника під час виконання місії.

Зрештою, не потребуючи наземного пункту управління на аеродромі, безпілотник може прилетіти безпосередньо в район проведення операції, де його обслуговуватиме місцевий персонал.

1.2. Типи БПЛА

БЛА поділяються на такі типи:

• ціль і приманка - забезпечення наземної та повітряної стрілецької зброї цілі, що імітує літаки чи ракети противника;

• Розвідка - забезпечення розвідки на полі бою;

• Бойові - забезпечення можливості атаки для місій високого ризику (див. Безпілотний бойовий повітряний апарат);

• Дослідження та розробки - використовуються для подальшого розвитку технологій БПЛА, які будуть інтегровані в розгорнуті літаки БПЛА;

• Цивільні та комерційні БПЛА - БПЛА, спеціально розроблені для цивільних та комерційних застосувань [2].

Базова система MQ-9 має систему мультиспектрального націлювання, яка має надійний набір візуальних датчиків для націлювання. MTS-B поєднує в собі інфрачервоний датчик, кольорову / монохромну телевізійну камеру денного світла, посилену зображення телевізійну камеру, лазерний дальномір і лазерний підсвічувач. Повнометражне відео від кожного з датчиків зображення можна переглядати як окремі відеопотоки або злити.

Блок також включає в себе лазерний далекомір / покажчик, який точно визначає цілі для використання лазерно-керованих боєприпасів, таких як Направлена бомба-12 Paveway II. Reaper також оснащений радіолокатором із синтетичною апертурою, щоб забезпечити майбутнє націлювання на боєприпаси GBU-38 Joint Direct Attack. MQ-9 може також використовувати чотири ракети Hellfire з ракетою "земля-земля" (AGM) -114 з лазерним наведенням, які забезпечують надзвичайно точні пошкодження з низьким побічним ефектом, можливості протитанкової та протипіхотної взаємодії.

На другому рівні як актив ISR, MQ-9 є частиною системи, яка підтримує ударні літаки та наземні командири шляхом набуття та відстеження динамічних цілей або іншої корисної розвідки. Він також здатний підтримувати широкий спектр операцій, таких як прибережне та прикордонне спостереження, відстеження зброї, застосування ембарго, гуманітарна допомога / допомога при стихійних лихах, підтримка миротворчих операцій та боротьби з наркотиками. Використовуючи супутникові лінії зв'язку, RPA може отримувати та передавати дані зображень у реальному часі наземним користувачам цілодобово та поза зоною видимості (BLOS).

Дистанційно керований літак може бути розібраний і завантажений в єдиний контейнер для розгортання по всьому світу. Усю систему можна транспортувати на літаку C-130 Hercules або більшому. Літак MQ-9 працює від стандартних аеродромів США з чіткою прямою видимістю до антени наземного терміналу даних, яка забезпечує зв’язок прямої видимості для зльоту та посадки. PPSL забезпечує надмірний зв’язок для літака та датчиків.

Основна концепція операцій, операцій дистанційного розбиття, використовує наземну станцію запуску та відновлення для зльоту та посадки в прямому оперативному місці, тоді як екіпаж, що базується в континентальній частині Сполучених Штатів, виконує командування та управління рештою місія за межами прямої видимості. Віддалені операції

розділення призводять до того, що менша кількість персоналу розміщується в прямому місці, консолідує контроль різних польотів в одному місці, і як такий, спрощує функції командування та управління, а також проблеми з матеріально-технічним постачанням для системи озброєння [3].

Безпілотні (англ. unmanned — без людини на борту) літальні апарати, відповідно до стандартів НАТО, так само, як і літаки із пілотом на борту (англ. manned aircraft), керуючись значенням повної злітної маси розділено на 3 класи: I — повна злітна маса до 150 кг, II — повна злітна маса до 600 кг, III — повна злітна маса більше 600 кг. (JDN 2/11 2011, p. 2-5) (англ.)

Клас I підрозділяється на категорії: «мікро» — до 2 кг, «міні» — до 15 кг, «малі» — від 15 кг. (JDN 2/11 2011, p. 2-7) (англ.)

Від наведеної вище класифікації НАТО дещо відрізняється класифікація безпілотних авіаційних систем (UAS), що її застосовано у документі Департаменту оборони США (DOD-USRM-2013 2013, p. 6) (англ.). Згідно цього документу, виділяють п'ять груп UAS:

- Група 1 (мікро-, міні тактичні) — від 0 до 9 кг, до 300 метрів над ґрунтом, основний представник — «RQ-11 Raven».
- Група 2 (малі тактичні) — від 9.5 до 25 кг; до 1000 метрів над ґрунтом, представник — «Scan Eagle»
- Група 3 (тактичні) — менш, ніж 600 кг, представник — «RQ-7 Shadow»
- Група 4 (персистентні) — більш, ніж 600 кг; представник — «MQ-1B Predator»
- Група 5 (пенетрувальні) — більш, ніж 600 кг; представник — «MQ-9 Reaper»

1.3. Область реалізації обраного об’єкта дослідження “MQ-9“ Reaper ”

MQ-9 Reaper - це озброєний дистанційно керований літальний апарат із середньою висотою та довговічністю, що виконує багато завдань і застосовується в основному проти динамічних цілей виконання, а в другу чергу - як актив розвідки. Враховуючи значний час виходу з ладу, датчики широкого діапазону, багаторежимний набір засобів зв'язку та високоточну зброю, він забезпечує унікальну здатність виконувати удар, координацію та розвідку по високоцінних, швидкоплинних та чутливих до часу цілях.

БПЛА повсюдно застосовуються у військовій справі, в першу чергу для ведення повітряної розвідки — як тактичної, так і стратегічної. Безпілотники під-класів «міні-» та «мікро-» все ширше застосовуються під час бойових дій на рівні взводу та відділення для термінового отримання інформації типу «що за тим пагорбом», тобто для вирішення завдань військової розвідки. Далекосяжним напрямком їх застосування є вирішення завдань у складі рою. Також використовуються БПЛА для коригування вогневих ударів по наземних цілях та як ударні.

Крім того, невійськові дрони застосовуються для розв'язання широкого кола завдань, виконання яких пілотованими літальними апаратами з різних причин недоцільно. Такими завданнями є:

- моніторинг повітряного простору, земної й водної поверхонь,
- екологічний контроль,
- керування повітряним рухом,
- контроль морського судноплавства,
- розвиток систем зв'язку,
- польова логістика (трансфер запчастин, акумуляторних батарей, боєприпасів, медикаментів тощо),
- художня фотографія

1.4. Основні технічні характеристики “MQ-9“ Reaper ”

Військово-повітряні сили США запропонували систему MQ-9 Reaper у відповідь на директиву Міністерства оборони для підтримки ініціатив закордонних надзвичайних операцій. Він більший і потужніший, ніж MQ-1 Predator, і призначений для того, щоб наполегливо і точно виконувати цілі, чутливі до часу, а також знищувати або вимикати ці цілі. "M" - це позначення DOD для багатоцільової роботи, а "Q" означає дистанційно керовану систему літаків. "9" означає, що це дев'яте місце в серії дистанційно керованих літальних систем [3].

Озброєння

БПЛА MQ-9 (рис. 1.1) Predator може також використовуватися як озброєний багатоцільовий БПЛА, запускаючи ракети AGM-114C / K Hellfire та іншу керовану зброю. Загалом літак може нести до 14 ракет Hellfire, у порівнянні з двома на MQ-1 Predator. Predator B може нести бомби з лазерним керуванням, такі як GBU-12. MQ-9 оснащений як SAR Lynx II, так і 20-дюймовим карданним підвісом MTS-B, вдосконаленою версією корисного навантаження MQ-9 з розширеним діапазоном дії.

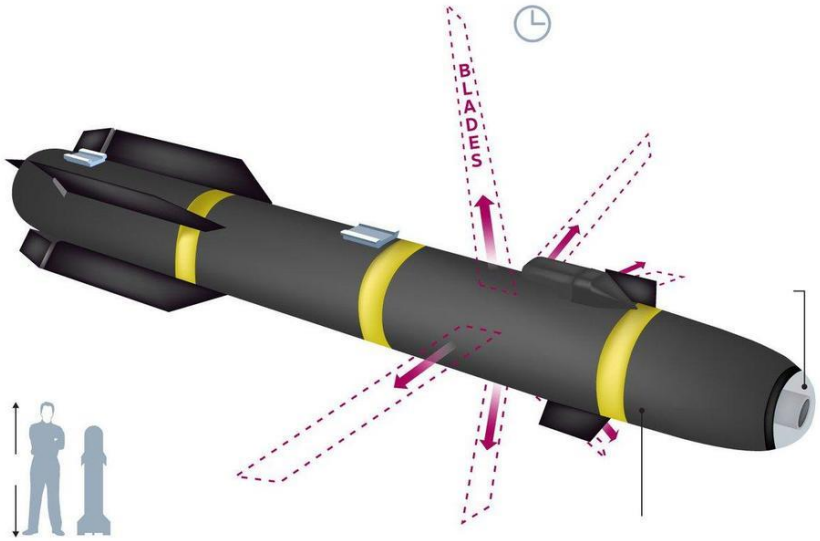

Рис. 1.1. AGM-114C / K Hellfire

Дизайн

Літак MQ-9 Reaper (рис. 1.2) був розроблений для роботи над горизонтом на середній та висоті для тривалих бойових вильотів. Літак був розроблений головним чином для притягнення до відповідальності критично важливих цілей, що чутливі до часу (TST), як радіолокаційного об’єкта атаки з бортовою здатністю важкого вбивства (мисливець-вбивця), а також для здійснення розвідки, спостереження, розвідки та придбання цілей (ISR TA) як другорядна роль. У ролі мисливця-вбивці літак застосовуватиме сплавлені мультиспектральні датчики для автоматичного пошуку, фіксації та відстеження наземних цілей (автоматичне наведення на ціль (ATC), точність розташування цілі (TLA), метричний датчик та інші можливості) та оцінку посту -ударні результати. MQ-9 також слід було дослідити щодо можливостей датчиків розвідки сигналів (SIGINT).

Рис. 1.2. Дизайн MQ-9 Reaper

Мобільність

MQ-9 Reaper має турбогвинтовий двигун (рис. 1.3) потужністю 950 кінських сил (712 кВт), набагато потужніший, ніж поршневий двигун Predator потужністю 115 к.с. запуск ракет AGM-114C / K Hellfire та іншої керованої зброї. Збільшення потужності дозволяє Жатку нести в 15 разів більше боєприпасів та здійснювати круїз зі швидкістю, що втричі перевищує швидкість MQ-1 Predator.

Рис. 1.3. Двигун MQ-9 Reaper

Аксесуари

MQ-9 Reaper оснащений надійними датчиками для автоматичного пошуку, фіксації, відстеження та націлювання критично важливих цілей, що чутливі до часу. У MQ-9 коефіцієнт питомого поглинання був замінений радаром AN / APY-8 Lynx II (рис.1.4), замінивши TESAR на більш вдосконалену систему радіолокаційного зображення з високою роздільною здатністю.

Рис. 1.4. AN / APY-8 Lynx II

Бойове використання

MQ-9 Reaper здатен нести максимальне внутрішнє корисне навантаження 800 фунтів та більш вдосконалені датчики вагою майже вдвічі більше, ніж MQ-1 Predator. Наявність високоефективних датчиків та велика потужність високоточної керованої зброї дозволяють новому Predator працювати як ефективна платформа "Мисливець-вбивця", шукаючи та вражаючи цілі з великою ймовірністю успіху. Він оснащений комунікаційним тактичним загальним посиланням L-3 (TCDL). Сегмент наземного управління Predator B є загальним для всіх попередніх систем Predator. ВПС США розвиває здатність керувати кількома літаками з однієї наземної станції, фактично збільшуючи загальну бойову ефективність над полем бою. MQ-9 Reaper Predator B був розроблений для того, щоб наполегливо і точно виконувати цілі, чутливі до часу, а також знищувати або виводити ці цілі з використанням 500-фунтових бомб і ракет HELLFIRE. Reaper представляє значний розвиток технологій БПЛА та зайнятості. Повітряні сили перейшли від використання БПЛА, головним чином, до розвідки, спостереження та розвідки до операції "Іракська свобода", до справжньої ролі мисливця-вбивці з "Жнець".

Типова система складається з декількох повітряних транспортних засобів, наземної станції управління, обладнання зв'язку / зв'язку, запасних частин та персоналу, який може бути сумішшю активного персоналу та персоналу підрядника (рис. 1.5). Екіпаж MQ-9 Reaper - це пілот і оператор датчика, який керує літаком із віддаленого GCS. Щоб задовольнити вимоги командирів бойових дій, MQ-9 Reaper надає спеціальні можливості, використовуючи набори місій, які можуть містити різні комбінації зброї та корисного навантаження датчиків.

Рис. 1.5. Оператори MQ-9 Reaper

Наземна станція управління (GCS) функціонує як кабіна літака і може керувати літаком як в межах прямої видимості (LOS), так і поза LOS (BLOS) за допомогою комбінації супутникового ретрансляційного та наземного зв'язку. GCS є або мобільним для підтримки прямих операційних місць, або встановленим на об'єкті для підтримки віддалених розділених операцій (RSO). GCS має можливість виконувати планування місій, забезпечувати засоби для ручного та / або автономного управління, а також конфігурацію GCS, що дозволяє контролювати кілька літаків та корисних навантажень, дозволяти персоналу запускати, відновлювати та контролювати літаки, корисні навантаження та системні комунікації статус, безпечні лінії передачі даних для отримання даних датчика корисного навантаження та командних посилань, моніторинг загроз літаку, відображення загальної картини експлуатації та забезпечення функцій підтримки. Крім того, GCS для запуску та відновлення (LRGCS) дозволяє обслуговувати, перевіряти системи, обслуговувати, запускати та відновлювати літаки під контролем LOS для передачі на мобільний або стаціонарний GCS. Очікувалось, що GCS буде

продовжувати розвиватися та вдосконалювати свої можливості, щоб йти в ногу з можливостями літаків MQ-9 та місіями, які вони виконують [4].

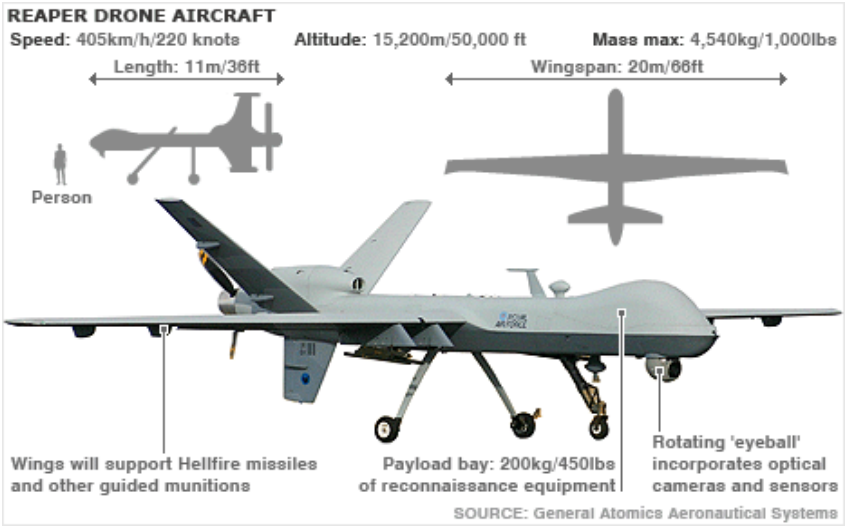

Рис. 1.6. Оснащення та характеристики MQ-9 Reaper

Таблиця 1.1

Загальні характеристики

Оснащення	Висота
Поєднання ракет AGM-114 HELLFIRE, GBU-12 та GBU-38 JDAM	7,500 m
Користувачі	Вага
США, Італія, Великобританія	2,223 kg
Країна виробник	Швидкість
США	482 km/h
Облаштування	Дальність
Теплова камера, датчик AN / APY-8 Lynx II радар, AN / DAS-1 MTS-B Мультиспектральна система націлювання	14 -28 годин - 5,926 km
Оператор	Розміри
2, один пілот і один оператор датчика	Довжина: 11,0 м; Розмах крил: 20,1 м; Висота: 3,8 м

Вихідні характеристики

Основна функція: знайти, виправити і завершити завдання

Підрядник: General Atomics Aeronautical Systems, Inc.

Оснащення двигуном: турбовинтовий двигун Honeywell TPE331-10GD

Тяга: максимум 900 кінських сил валу

Розмах крил: 66 футів (20,1 метра)

Довжина: 11 метрів

Висота: 12,8 футів (3,8 метра)

Максимальна злітна вага: 10500 фунтів (4760 кілограмів)

Паливна ємність: 4000 фунтів (602 галони)

Корисне навантаження: 3750 фунтів (1701 кілограм)

Швидкість: круїзна швидкість близько 230 миль / год (200 вузлів)

Дальність: 1150 миль (1000 морських миль)

Стеля: до 50 000 футів (15 240 метрів)

Озброєння: поєднання ракет AGM-114 Hellfire, GBU-12 Paveway II та GBU-38 Joint Direct Attack боєприпасів

Екіпаж (дистанційно): два (пілот і оператор датчика)

Вартість одиниці: $ 64,2 млн

Початкова експлуатаційна спроможність: жовтень 2007 р

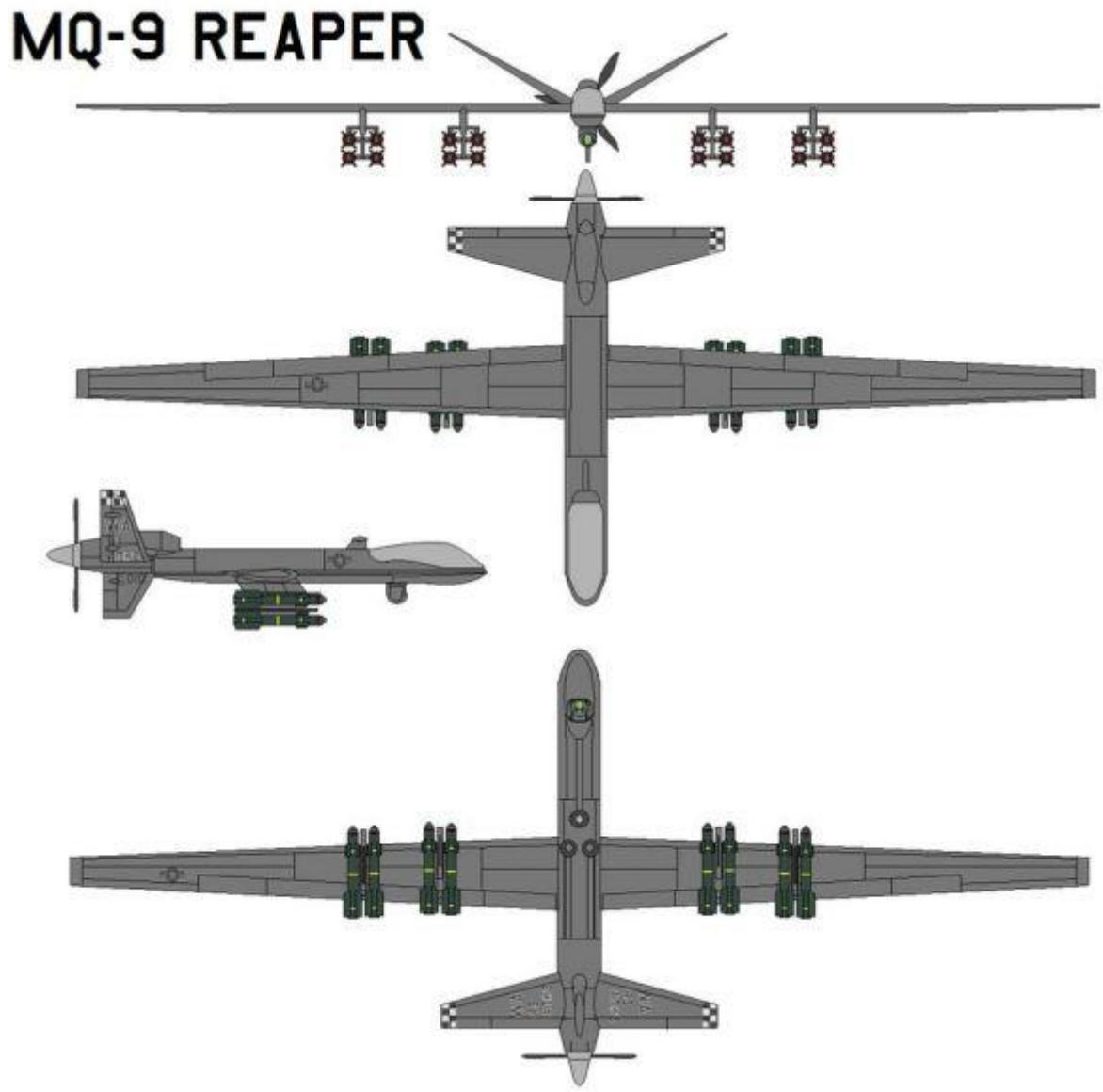

Рис.1.7. БПЛА MQ-9 Reaper

1.5. Особливості планування траєкторій БПЛА

Однією з проблем є планування оптимальної траєкторії руху БПЛА, щоб мінімізувати час досягнення цілі об'єкта. Сучасні БПЛА оснащені системами комп'ютерного управління і ГЛОНАСС- або GPS-приймачем [7], що дозволяє направляти БПЛА по заданому маршруту. Тому проблема найбільш результативно може бути вирішена з використанням супутникової карти і комп'ютерної оптимізації траєкторії руху БПЛА, а також координат і часу зупинок для БПЛА вертолітного або мультікоптерного типу. У разі

використання БПЛА мультікоптерного типу завдання пошуку оптимального алгоритму польоту зводиться до мінімізації довжини траєкторії у вигляді ламаної лінії, або часу польоту. Перед початком оператор задає точки зависання БПЛА і передачі голосових повідомлень. Складемо математичні основи пошуку оптимального алгоритму руху БПЛА [8]. Вихідними даними служить набір точок P0, P1, P2, ... Pn, заданих оператором і прив'язаних до супутниковій карті місцевості. завдання побудови оптимальної траєкторії полягає у виборі послідовності перельоту між точками,таким чином, щоб сумарний час польоту був мінімальним. Час польоту може немонотонно залежати від довжини траєкторії, так як на швидкість польоту може робити істотний вплив (близько 10-50%) вітер (від 3 до 10 м / с). Тому доцільно мінімізувати саме сумарний час, але не довжину траєкторії. Крім того, в разі оптичного розпізнавання координат будівель в алгоритм оптимізації може включатися і вибір координат точок зависання (xPi, yPi, zPi). Для розрахунку сумарного часу польоту tсум пропонується використовувати наступну формулу:

$$t_{сум} = n \cdot t_{оп} + \sum_{\substack{i=0,\\ i=n+1 \Rightarrow i=0}}^{n+1} \left(t_p + \frac{\sqrt{\left(x_{P_{i-1}} - x_{P_i}\right)^2 + \left(y_{P_{i-1}} - y_{P_i}\right)^2} - L_p - L_m}{v_м + v_в \mathrm{Cos}\left(\mathrm{Arctg}\left(\frac{y_{P_{i-1}} - y_{P_i}}{x_{P_{i-1}} - x_{P_i}} \right) - \varphi_в \right)} + t_m \right), \quad (1.1)$$

де n - кількість точок зупинки, tоп -час голосового оповіщення;

tр і tт - час розгону і гальмування при русі по прямолінійній ділянці (близько 2-5 с);

Lр і Lт - довжина ділянок розгону і гальмування (близько 10 30 м);

(XPi, yPi) - координати і-й точки зупинки мультикоптер;

Vм - швидкість Мультикоптер;

vв і φв - швидкість і кут напрямку вітру;

прописні букви в позначеннях Cos і Arctg означають корекцію кутів під косинусом на перехід 0-3600 і подвоєння діапазону 1800 для арктангенса.

Дія «i = n + 1 => i = 0» означає, що останній точкою зупинки Мультикоптер буде не точка Pn + 1, а точка P0 (повернення в вихідну точку запуску).

Завдання оптимізації плану польоту записується аналітично наступним чином:

$$t_{сум}\left(P_0, P_i, P_j, \ldots P_0, v_в, \varphi_в\right) \rightarrow \min \Rightarrow i, j, \ldots$$

Тобто, необхідно домогтися мінімуму часу оповіщення tсум шляхом вибору оптимальної послідовності точок i, j, і т.д. В даний час ведеться робота по розробці комп'ютерної програми, що реалізує запропонований метод оптимізації плану польоту БПЛА. Таким чином, розроблена методологічна основа вибору оптимального алгоритму руху БПЛА при оповіщенні населення в малих населених пунктах і ізольованих об'єктах [9].

Висновок до розділу 1

У цьому розділі були описані сфери застосування MQ-9 та основні аеродинамічні характеристики, які необхідно буде застосувати в майбутній програмі планування траєкторії.

РОЗДІЛ 2
ЕТАПИ ПОЛЬОТУ ПО КОЛУ. ХАРАКТЕРИСТИКИ ПС ПІД ЧАС ПОЛЬОТУ

2.1. Зліт

Діагностичні моделі злітних характеристик ПС

Рулювання - проводиться після виконання попередніх операцій по підготовці відповідно до карти контрольних перевірок. Керування напрямком рулювання проводиться поворотом передньої стійки шасі від спеціального штурвала на основному штурвалом (Іл-62) або від педалей (Ту-154). Кут повороту передньої стійки шасі -55 ° (Ту-154), потрібна ширина розвороту до 45 м.

Літаки з ТРД і ТВД можуть виконувати рулювання по розмоклому трав'яному і не утрамбованому сніжному покриву при зниженому тиску пневматики коліс до 4.5-6.0 кг / см2, на бетонованій смузі 6.5-9.5 кг / см2.

Рулювання на ґрунтовій смузі не рекомендується - пошкодження грунту, утворення колії, пил від струменя реактивного двигуна. Застосовується вкрай рідко в разі гострої необхідності (Ан-24, Ан-26, Ан-12). Іноді роблять металізовану ВПП.

Зліт - рух літака від моменту старту до набору умовної висоти перешкод на підходах до аеродрому і придбанні безпечної швидкості зльоту ($H_{УСЛ}$=10.7 м=35 футів).

При досягненні швидкості V_R літак перекладається на кут атаки відриву (але так, щоб не було торкання хвостовій частині про ВПП).

Зменшення кута атаки збільшує довжину розбігу, при більшому куті атаки відбувається передчасний відрив на малій швидкості ($\alpha_{отр}$=10-12°, $C_{Y\ отр}$=1.2-1.7).

Прибирання шасі - починається при швидкості на 20-30 км / год більше швидкості відриву на висоті 5-10 м, час збирання 8-12 с.

Прибирання закрилків - на висоті 100-120 м, пікіруючий момент знімається за допомогою тримерів.

Сили діючі на літак:

- підйомна Y
- сила тяги P
- вага G
- сила тертя $F=F_1+F_2$

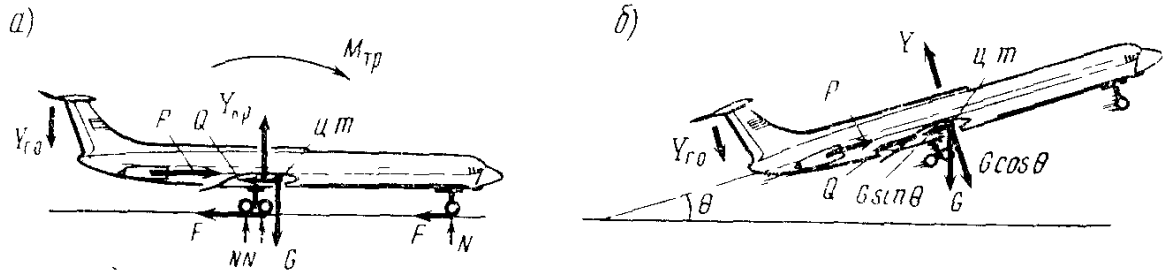

Рис 2.1. Схема сил, що діють на літак при розбігу (а) і після відриву при наборі висоти (б).

В результаті взаємодії сил тяги, тертя, лобового опору з'являється різниця - прискорює сила $R_{УСК}$ = P-X-F, під дією якої літак отримує прискорення:

$$j_X = \frac{R_{УСТ}}{m} = \frac{R_{УСТ}}{G} \cdot g = 9.81 \cdot \frac{R_{УСТ}}{G}, \text{ або } j_X = 9.81 \cdot (\frac{P}{G} - \frac{X}{G} - \frac{F}{G}) \qquad (2.1)$$

P - визначається параметрами двигуна, зовнішніми умовами.

Величина $\frac{P}{G}$ - тягооснащеність літака, чим більше, тим більше прискорення, менше довжина розбігу. Для літаків з 2-ма двигунами тягооснащеність 0.28-0.33, з 4-ма - 0.22-0.26 кт тяги / кт ваги.

Сила тертя

$$F = f_{TP} \cdot (G - Y). \qquad (2.2)$$

Коефіцієнт тертя кочення при сухому бетоні f_{TP} = 0.03-0.04, при сухому грунтовому покрові і битій сніговому покриві - 0.07, при мокрому трав'яному покриві - 0.1.

Лобове опір - в момент відриву:

$$X_{OTP} = C_{X.OTP} \cdot S \cdot q. \qquad (2.3)$$

Підіймальна сила — складова аеродинамічної сили, перпендикулярна до вектора швидкості руху тіла в потоці рідини або газу, на відміну від паралельної складової — аеродинамічного опору. Підіймальна сила виникає через несиметричність обтікання тіла потоком. Відповідно до закону Бернуллі, статичний тиск середовища в тих місцях, де швидкість потоку вища, буде нижчим, і навпаки. Ця різниця тисків і породжує підіймальну силу.

Підіймальна сила — це інтеграл від тиску навколо контуру профілю крила (без урахування індуктивних втрат через тривимірні ефекти).

$$Y = \int_{\partial\Omega} pn \, d\partial\Omega, \qquad (2.3)$$

де:

- **Y** — підіймальна сила,
- $\partial\Omega$ — межа профілю,
- p — тиск,
- **n** — нормаль до профілю

Коефіцієнт підіймальної сили

Коефіцієнт підіймальної сили — безрозмірна величина, що характеризує підіймальну силу крила певного профілю при відомому куті атаки. Коефіцієнт визначається експериментально в аеродинамічній трубі, або за теоремою Жуковського. Формулу розрахунку підіймальної сили через коефіцієнт розробили брати Райт і Джон Смітон на початку XX століття. Формула має вигляд:

$$Y = C_y \frac{pV^2}{2} \mathrm{S} \qquad (2.4)$$

де:

- Y — підіймальна сила (Н)
- C_y — коефіцієнт підіймальної сили
- ρ — густина повітря на висоті польоту (кг/м³)
- V — швидкість набігаючого потоку (м/с)
- S — характерна площа (м²)

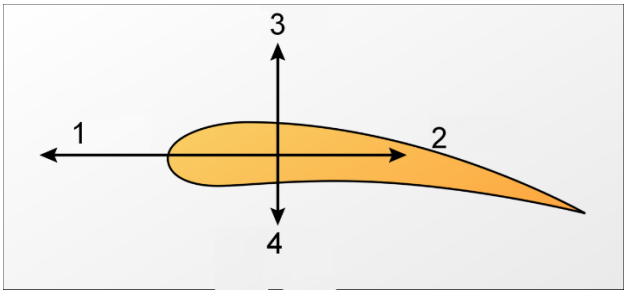

Рис. 2.2. Аеродинамічні сили, що діють на крило літака: 1 — тяга; 2 — лобовий опір; 3 — підіймальна сила; 4 — вага

2.2. Політ по колу

Політ по колу («політ по коробочці») - політ за встановленим маршрутом (зазвичай прямокутному) в районі аеродрому для відпрацювання зльоту, заходу, розрахунку на посадку і посадки, а також для догляду і підходу до аеродрому. Є важливою частиною заходу на посадку і навчально-тренувальних польотів на літаках і планерах.

Характеристики кола і його позначення

Як розташований коло щодо ВПП і місцевості, його висота та інші характеристики - вказуються у відповідній навігаційної документації, з якої пілот або курсант знайомляться в процесі підготовки до польоту [5].

Є дві системи позначення елементів кола - вітчизняна та зарубіжна. У вітчизняній системі позначаються кути-розвороти, в західній - відрізки. На схемі вони відповідно позначені чорними і синіми прямокутниками. Зліт і посадка на смугу проводиться завжди проти вітру - в тому випадку, якщо вітер не строго проти посадкового курсу, вважається напрямок його проекції на напрям посадкового курсу. Західна система в цьому плані логічно описує становище перших трьох відрізків щодо напрямку вітру:

- Upwind Leg - відрізок проти вітру
- Crosswind Leg - відрізок поперек вітру
- Downwind Leg - відрізок за вітром

Решту два останніх відрізка:

- Base Leg - базовий відрізок, підстава
- Final Leg - фінальний відрізок

У вітчизняній термінології ділянки вказуються як відрізки між розворотами - наприклад "від першого до другого". Остання ділянка - фінальний відрізок - по-іншому називається або "глісада", або "пряма".

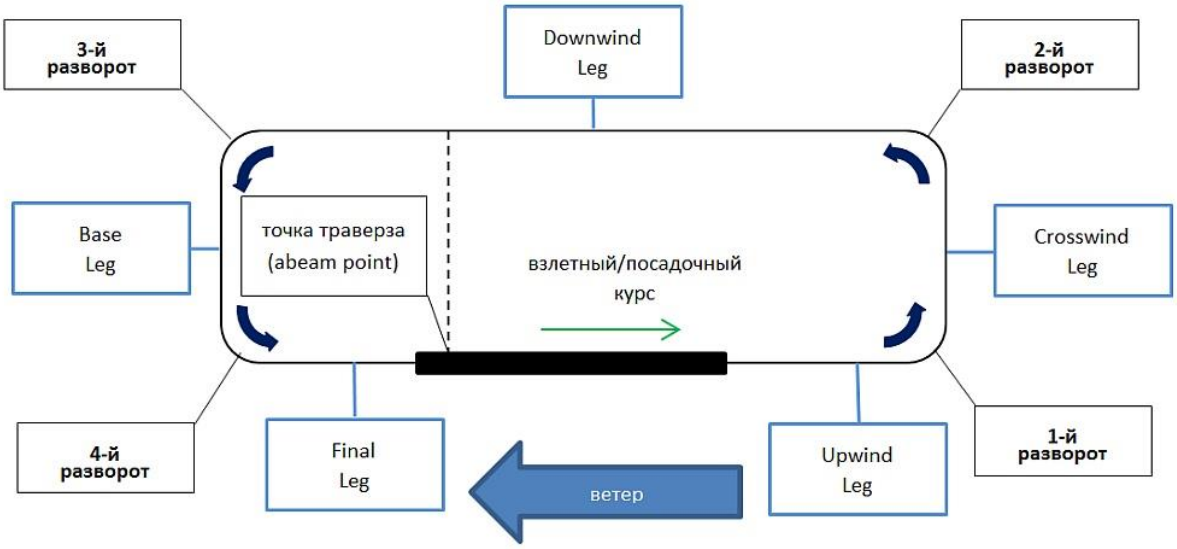

Рис. 2.3. Траєкторія польоту по колу

Окремої пояснення заслуговує показана на схемі точка траверзу (abeam point), але для її пояснення треба розглянути політ по колу в тривимірному вигляді, враховуючи висоту на різних ділянках:

С урахуванням висоти політ по колу виглядає так:

- Проводиться зліт

- Ділянка до першого розвороту виконується в режимі набору висоти після зльоту

- Перший розворот також робиться в режимі набору висоти

- На ділянці між першим і другим розворотом набирається встановлена висота кола - TPA (Traffic Pattern Altitude) - зазвичай 300 метрів.

- Другий розворот виконується на висоті кола

- На ділянці від другого до третього розвороту політ до точки траверзу проводиться на режимі горизонтального польоту на висоті кола

- Точка траверзу - початок зниження для заходу на посадку

- Від точки траверзу до третього розвороту ділянку проходиться в режимі зниження

- Третій розворот виконується в режимі зниження, коли смуга (точка траверзу) буде видна під кутом 45 градусів назад. Це аналогічно тому як з

точки траверзу на смузі сам літак був би видний під кутом 45 градусів щодо перпендикуляра на лінію між другим і третім розворотом - тобто точки початку зниження

- Ділянка від третього до четвертого розвороту також йде в режимі зниження

- Четвертий розворот виконується в режимі зниження і так, щоб вийти у ворота посадкової смуги на посадковому курсі

- Від четвертого розвороту до смуги літак знижується по глісаді

- Проводиться посадка

Висота кола - ТРА - описується зазвичай в термінах "перевищення" - AGL - Above Ground Level - над рівнем землі. Реально по ходу кола рельєф місцевості може бути неоднорідним, тому в якості відправної точки приймають перевищення над рівнем смуги. Фактично це аналогічно висоті польоту літака в системі QFE (пояснення є в попередній частині) стосовно до даного аеродрому.

При заданій висоті кола в 300 метрів перший розворот повинен бути виконаний на висоті не більше ніж 200 метрів AGL. Згідно вітчизняним програмам місце другого розвороту також як і третього визначається по деякому заданому розі візування на смугу. Згідно із західними програмами щодо явного визначення місця другого розвороту нічого не говориться, а третій визначається аналогічно.

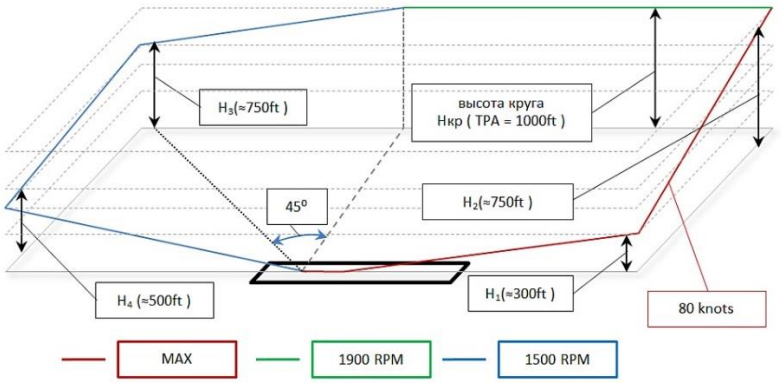

Рис. 2.4. Траєкторія польоту по колу у тривимірному просторі

Виконується політ так:

- Виконуємо зліт на максимальному режимі двигуна (режим MAX);

- Після зльоту набираємо швидкість в 80 вузлів і тримаємо її такою - літак буде набирати висоту зі швидкістю близько 300 метрів в хвилину;

- Після досягнення 100 метрів виконуємо перший розворот в режимі набору висоти;

- Після закінчення першого розвороту висота буде близько 150 метрів;

- Продовжуємо набір зберігаючи приладову швидкість в 80 вузлів;

- По досягненню висоти близько 250 метрів виконуємо другий розворот - висота береться з розрахунку, щоб після закінчення другого розвороту виявитися на висоті кола;

- Після закінчення другого розвороту прибираємо оберти двигуна і переводимо його на режим горизонтального польоту - близько 1900 RPM;

- Виконуємо політ до точки траверзу на висоті кола – 300 метрів - в режимі горизонтального польоту;

- На точці траверзу переводимо двигун в режим зниження - приблизно 1500 RPM і починаємо зниження - швидкість зниження близько 150 метрів в хвилину;

- Точку третього розвороту визначаємо за кутом візування назад на точку траверзу на смузі / торець смуги. Висота перед третім розворотом буде близько 250 метрів;

- Виконуємо третій розворот в режимі зниження;

- Четвертий розворот виконуємо з розрахунком візування, щоб опинитися на посадковому курсі в створі лінії. Висота перед четвертим розворотом - близько 150 метрів;

- Йдемо по глісаді до смуги;

- На висоті близько 30 метрів прибираємо обороти приблизно до 1200 - 1000 RPM і виробляємо посадку.

При неправильному дотриманні курсів на відрізку від другого до третього розвороту можна виконати третій розворот або раніше, або пізніше і вийти на четвертий з меншою або більшою втратою висоти - тобто виявитися після четвертого або нижче, або вище глісади. Аналогічно при неправильному дотриманні напрямки і / або неправильному розташуванні відрізка від третього до четвертого розвороту можна після четвертого вийти або лівіше, або правіше створу смуги. При цьому витримування напрямків на перший і другий розворот для початку - менш критично, особливо - на перший розворот [11].

2.3. Захід на посадку та глісада

Захід на посадку — один з кінцевих етапів польоту повітряного судна, що безпосередньо йде перед посадкою літального об'єкта. Забезпечує вивід повітряного судна на траєкторію (глісаду), яка є передпосадковою прямою, що веде до точки приземлення.

Захід на посадку може здійснюватися як з використанням радіонавігаційного обладнання (називається в такому випадку заходом на посадку за приладами), так і візуально, при якому орієнтування здійснюється екіпажем за природною лінією горизонту та інших орієнтирах на місцевості. Захід може здійснюватись за приладами, що переважно є продовженням польоту за ППП (правила польоту за приладами) або ж візуальним заходом згідно ПВП (Правила візуальних польотів).

Глісада— траєкторія польоту літального апарата, по якій він знижується безпосередньо перед посадкою. В результаті польоту по глісаді літальний апарат попадає у зону приземлення на злітно-посадковій смузі [10].

У парапланеризмі базовою глісадою називається пряма траєкторія безпосередньо перед посадкою.

Кут нахилу глісади — кут між поверхнею глісади та поверхнею горизонту. Кут нахилу глісади (КНГ) є однією з важливих характеристик злітно-посадкової смуги аеродрома. Для сучасних цивільних аеродромів зазвичай знаходиться в інтервалі 2…4,5°. На величину КНГ може впливати наявність перешкод в зоні аеродрома.

В СРСР типовим значенням кута нахилу глісади було прийнято 2°40′. Міжнародна організація цивільної авіації рекомендує КНГ 3°.

Також глісадою інколи називають сам процес зниження літака перед посадкою.

«Захоплення» глісади

Від правильного «захоплення» глісади залежить якість подальшої стабілізації літака на глісаді.

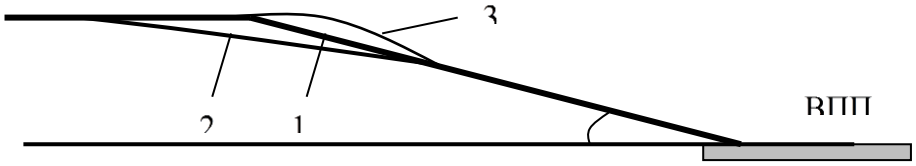

Рис. 2.5. «Захват» глісади

1 – ідеальний «захват» глісади $V_y = 0 \rightarrow V_y = W\theta_{гл}$

2 – ранній «захват» глісади

3 – пізній «захват» глісади

З точки зору безпеки польотів краще, щоб літак перебував над глісадою. Тому при автоматичному «захваті» перемикання в режим зниження здійснюється в момент або після перетину глісади.

Автоматизація управління посадкою (приземлення)

Траєкторія руху літака при посадці складається з декількох ділянок:

- планування по глісаді Vпл=70-85 м/с, Vy=3,5-4,5 м/с;
- вирівнювання;
- витримування;
- парашутування Vy=0,5-0,6 м/с

Рис. 2.6. Рух літака по глісаді

Управління зльотом і відходом на друге коло.

Особливості зльоту: max режим роботи двигунів, значна довжина розбігу, min час.

Фази розбігу:

1. – розгін літака до $V_R \approx V_{отр}$ (V_R – відрив носового колеса);
2. – відрив носового колеса;
3. – контроль швидкості на Нусл=10,7 м
4. – набір висоти до Н=400 м.

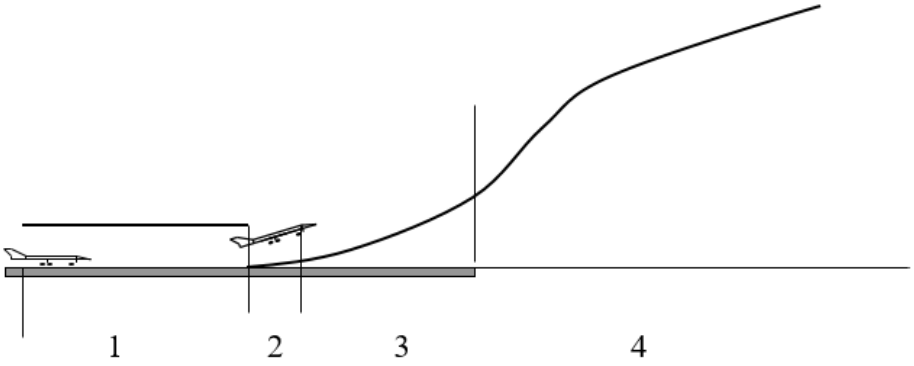

Рис. 2.7. Фази розбігу

На першій фазі автоматизація зводиться до обчислення Lкрит. для контролю критичної швидкості, навіть якщо не було відмови двигуна (індикатора зльоту).

Принцип реалізації:

- порівняння дійсного графіка зміни швидкості і пройденого шляху з розрахунковими;

- порівняння дійсного графіка зміни прискорення з розрахунковим;

- передобчислення дистанції розбігу або швидкості літака в контрольній точці і порівняння з розрахунковим значенням.

2.4. Механізація крила та хвостового оперення літака

На сучасних літаках для досягнення великих швидкостей польоту значно зменшені площа крила і його подовження. А це негативно позначається на аеродинамічному якості літака на злітно-посадочних режимах.

Для утримання літака в повітрі необхідно, щоб підйомна сила дорівнювала вазі літака: Y = G. Оскільки

$$Y = Cy\frac{\rho v^2}{2} \cdot S, \qquad (2.5)$$

то можна записати:

$$G = C_y \frac{\rho v^2}{2} \cdot S. \qquad (2.6)$$

З формули випливає, що для утримання літака в повітрі з даними вагою G на найменшій швидкості потрібно, щоб коефіцієнт підйомної сили Су був більше.

Профілі крила, що мають великий Су, мають, як правило, великим лобовим опором. А збільшення Сх перешкоджає збільшенню максимальної швидкості польоту.

При проектуванні профілів крила літака прагнуть в першу чергу забезпечити максимальну швидкість, а для зменшення швидкості на зльоті і посадці застосовують спеціальні пристрої, які називаються механізацією крила.

За допомогою механізації крила збільшується кривизна профілю (в деяких випадках і площа крила). В результаті максимальне значення коефіцієнта підйомної сили значно зростає.

Ці пристосування при польоті на малих кутах атаки (при великих швидкостях польоту) не використовуються, а застосовуються лише на зльоті і посадці.

Основними видами механізації крила є: щитки, закрилки, передкрилки.

Щиток являє собою відхиляючу поверхню, яка в прибраному положенні примикає до нижньої, задньої поверхні крила (рис. 2.8).

Збільшення $C_{умакс}$ при відхиленні щитка пояснюється зміною форми профілю крила і угнутості (кривизни) профілю.

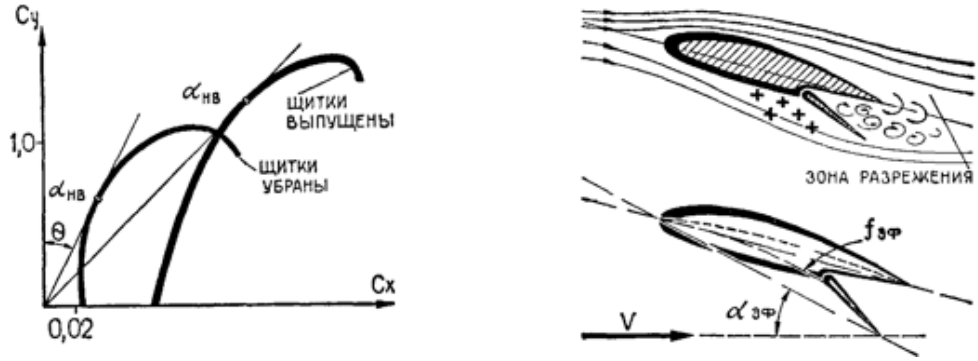

Рис. 2.8. Профіль крила з щитком

При відхиленні щитка утворюється вихрова зона підсмоктування між крилом і щитком. За рахунок відсмоктує дії щитка, швидкість потоку над крилом зростає, а тиск зменшується. Крім того, відхилення щитка підвищує тиск під крилом. Завдяки цьому випуск щитків збільшує різницю тисків над крилом і під крилом, а, отже, і коефіцієнт підйомної сили Су.

Одночасно зі збільшенням коефіцієнта підйомної сили збільшується і коефіцієнт лобового опору, аеродинамічна якість крила при цьому зменшується.

Закрилки.

Закрилком називається хвостова профільна частина крила, яка може відхилятися вниз(Рис. 2.9).

Використовуються кілька різновидів закрилків.

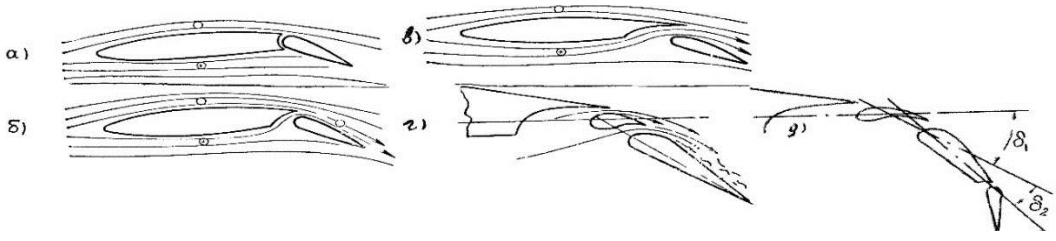

Рис. 2.9. Різновиди закрилків (а) поворотні; б) щілинні поворотні; в) висувні; г) двохщілисті; д) двохланкові).

Поворотні закрилки (Рис. 2.9, а). Поворотні закрилків, так само як і щиток, призначений для збільшення кривизни профілю крила. підвищення коефіцієнта C_y при відхиленні закрилка дещо більше, ніж при відхиленні щитка, а зростання СХ, особливо при малих кутах відхилення закрилки, менше, тому менше втрати аеродинамічного якості.

При зльоті для зменшення втрат якості кут відхилення закрилків не повинен перевищувати 15 - 20 °; при посадці він може досягати 35 - 40 °. При подальшому збільшенні кута відхилення закрилків зростання C_y зменшується внаслідок відриву прикордонного шару з закрилка.

Щілинні поворотні закрилки(Рис. 2.9 , б). Зрив потоку з поверхні закрилка можна кілька зупинити за допомогою звужується щілини, яка утворюється між крилом і закрилком при його відхиленні. Такий закрилків називають щілинним.

Висувні закрилки (Рис. 2.9, в). Такі закрилки застосовуються для збільшення площі крила і використання ефекту кривизни профілю і щілинного ефекту.

Збільшення площі крила залежить від величини висування закрилка. Закрилків може висуватися назад майже на всю величину своєї хорди.

Для попередження передчасного зриву потоку при великих кутах відхилення висувні закрилки роблять двохщілистими (Рис. 2.9, г). Друга щілину підсилює ефект природного сдуву прикордонного шару. Висувні закрилки широко застосовуються на сучасних літаках.

Багатоланкові закрилки. Багатоланкові називаються розрізні закрилки, які складаються з 2 - 3 частин, що повертаються щодо один одного, (Рис. 2.9, д). У прибраному положенні закрилки всі його ланки зрушені; в випущеному - ланки розсуваються, утворюючи профільовані щілини.

При такій кінематики відхилення закрилків максимально збільшуються кривизна профілю, площа крила і щілинний ефект. При багатоланкових

закрилках досягається найбільше зростання підйомної сили. Але при цьому втрати аеродинамічного якості також найбільші.

Для швидкого збільшення лобового опору в разі необхідності застосовуються інтерцептори - так звані повітряні аеродинамічні гальма. Зазвичай ці гальма виконуються у вигляді щитків і застосовуються для зменшення довжини пробігу літака при посадці.

Передкрилки.

Передкрилками називається висувається вперед і відхиляється на певний кут профільований носок крила(Рис. 2.10).

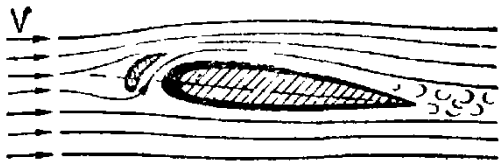

Рис. 2.10. Передкрилок

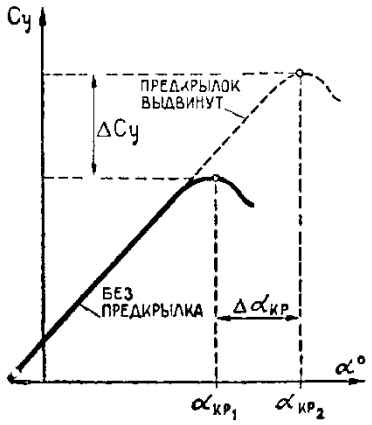

Рис. 2.11. Залежність Су від кута атаки

У притиснутому положенні передкрилок вписується в обводи профілю крила, в висунутому - утворює з крилом профільовану щілину. Відхилення передкрилка не збільшує підйомну силу крила. Робота його полягає у взаємодії з основною частиною крила.

При висунутому передкрилки між крилом і передкрилки утворюється звужується щілину. Цівки повітря, що проходять через щілину, збільшують кінетичну енергію прикордонного шару крила, зміщуючи точку його відриву назад до задньої крайки.

Передкрилки можуть встановлюватися на передній крайці крила по всьому розмаху крила або тільки на кінцях крила (кінцеві передкрилки).

Кінцеві передкрилки не збільшують $C_{y\max}$, А тільки сприяють тому, щоб зрив потоку стався при можливо більшому куті атаки. Це сприятливо позначається на роботі елеронів, покращує стійкість і керованість літака при зльоті та посадці. Передкрилки, розташовані по всьому розмаху, призводять до збільшення максимального значення C_y всього крила в середньому на 50% (Рис. 2.11) [12].

Призначення, типи та принцип роботи механізації крила

Для зменшення злітно-посадочних швидкостей і дистанції необхідно збільшувати на зльоті і посадці несучі властивості ЛА. Це досягається застосуванням різної механізації крила. Розрізняють механізацію задньої і передньої кромок крила.

Механізація задньої кромки:

1 - закрилки (прості, щілинні, багатощілинні, висувні, струменево-ежекторні, реактивні)

2 - щитки (прості, висувні)

Механізація передньої кромки:

1 - передкрилки,

2 - носові щітки,

3 - відхиляються шкарпетки,

4 - видув струменів уздовж передньої кромки.

Розглянемо принцип дії різних видів механізації. При відхиленні звичайних закрилків збільшується кривизна крила, що призводить до зростання коефіцієнта тиску, отже, і коефіцієнта Суа сумарною підйомної

сили. При великих кутах відхилення закрилки на крилі починається зрив потоку і подальшого зростання Суа не відбувається. Щоб затягнути зрив на великі кути dз застосовують щілинні закрилки. Проходячи через профільовану щілину між крилом і закрилком потік розганяється поблизу поверхні закрилка, підвищуючи стійкість прикордонного шару до відриву.

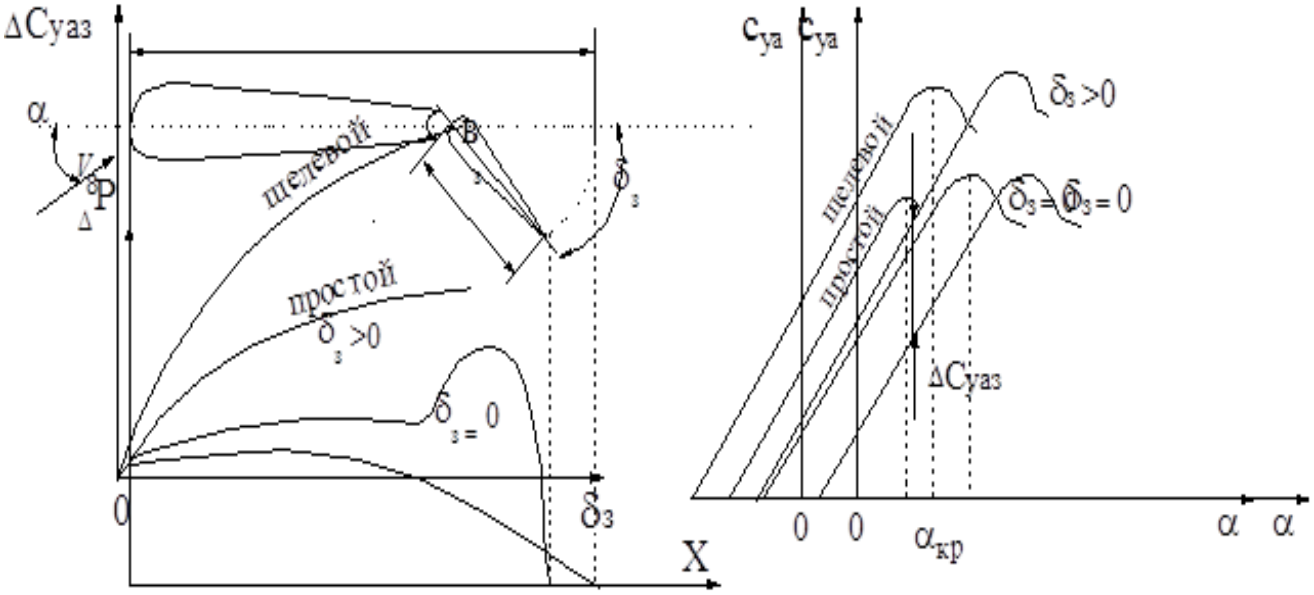

Рис. 2.12. Порівняння простого та щільовогу закрилків

У випадку висувного закрилка зменшується подовження крила, зменшується градієнт тиску і збільшується місцеві числа Re (за рахунок збільшення хорди), збільшується і площа крила. Все це веде до збільшення DCуаз і акр.

Під реактивним закрилком розуміється видув стисненого повітря (газу) з задньої кромки крила під кутом q до набігаючого потоку.

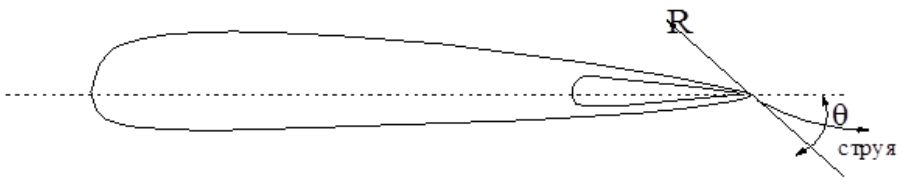

Рис. 2.13. Реактивний закрилок

Граючи роль "рідкого" закрилка, струмінь гальмує потік під крилом.

В результаті ежектуючої дії струменя потік над крилом розганяється. Перепад тиску Dp зростає. Крім того, підйомна сила зростає за рахунок вертикальної складової сили реакції струменя R. Приріст несучих властивостей крила зі струменевим закрилком забезпечується не тільки збільшенням кута видування струменя q, а й збільшенням коефіцієнта імпульсу струменя cm, який визначається виразом де rc і Vc щільність і швидкість витікання газу на виході з щілини видування, hщ-ширина щілини, lщ-довжина щілини за розмахом крила.

У струменево - ежекторному закрилку стиснений газ подається з задньої кромки нерухомої частини крила в порожнину, утворену верхньою поверхнею закрилка і сегмента.

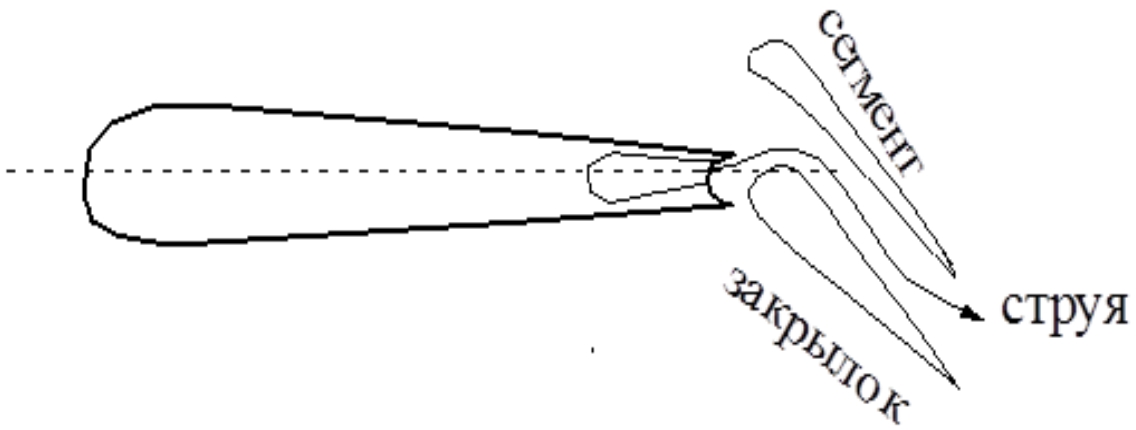

Рис. 2.14. Струменево - ежекторний закрилок

У такому закрилку крім простого і струменевого закрилків проявляється додаткова ежекція повітря з верхньої поверхні крила. Щитком називається відхиляюча нижня частина крила. При його відхиленні відбувається гальмування потоку під крилом і коефіцієнт тиску збільшується. В випадку висувного щитка несучі властивості зростають додатково за рахунок збільшення площі крила. При відхиленні щитка зменшується критичний кут атаки акр.

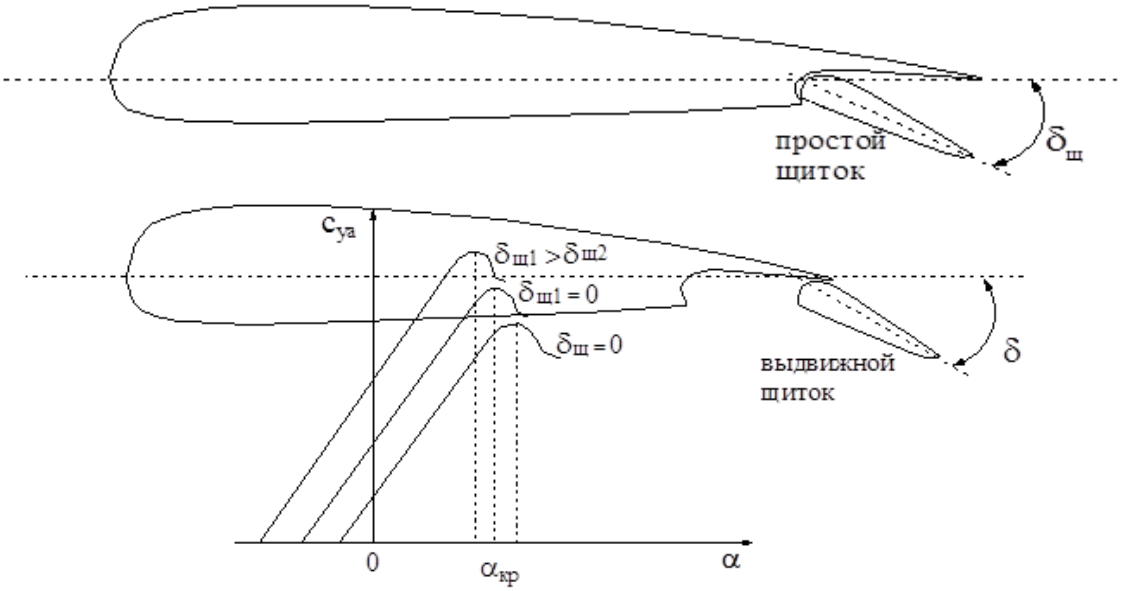

Рис. 2.15. Простий та висувний щитки

Механізація передньої кромки призначена в основному для запобіганню відриву потоку і покращений аеродинамічних характеристик на великих кутах атаки. Передкрилки представляють собою невеликі крила, встановлені вздовж передньої кромки основного крила. При висунення передкрилка між ним і крилом утворюється профільована щілина, в якій повітря прискорюється і виходить на верхню поверхню крила, збільшуючи швидкість частинок в прикордонному шарі, ніж та підвищує стійкість до відриву. В результаті цього збільшується акр і Суа макс

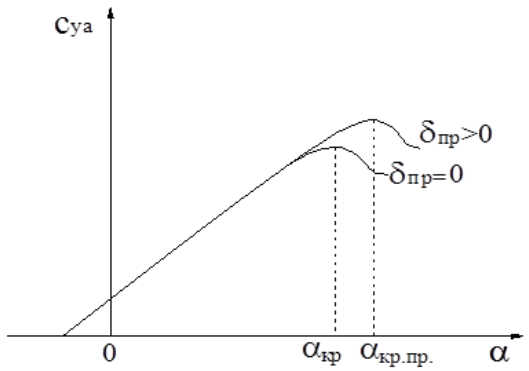

Рис. 2.16. Залежність Суа від а та передкрилку

Носовий щиток змінює кривизну крила, забезпечуючи більш плавне обтікання передньої кромки крила, що затягує зрив потоку на великі кути атаки. Відхиляється носок діє аналогічно носовому щитку.

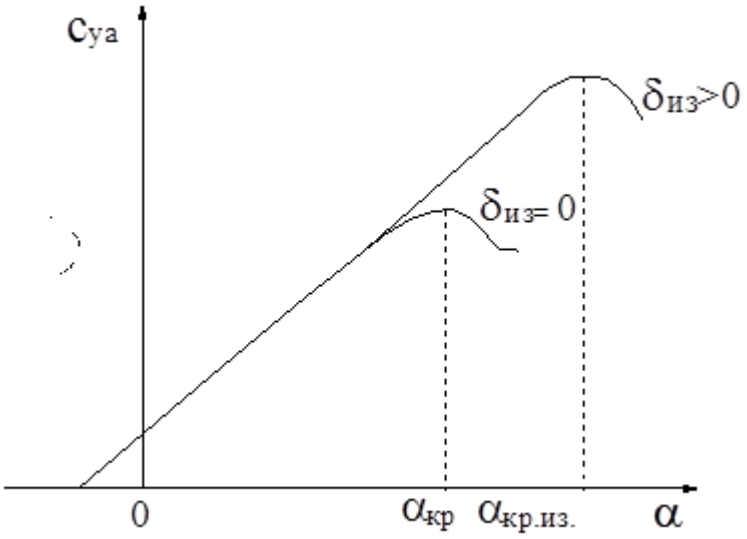

Рис. 2.17. Залежність Суа від а та щитка

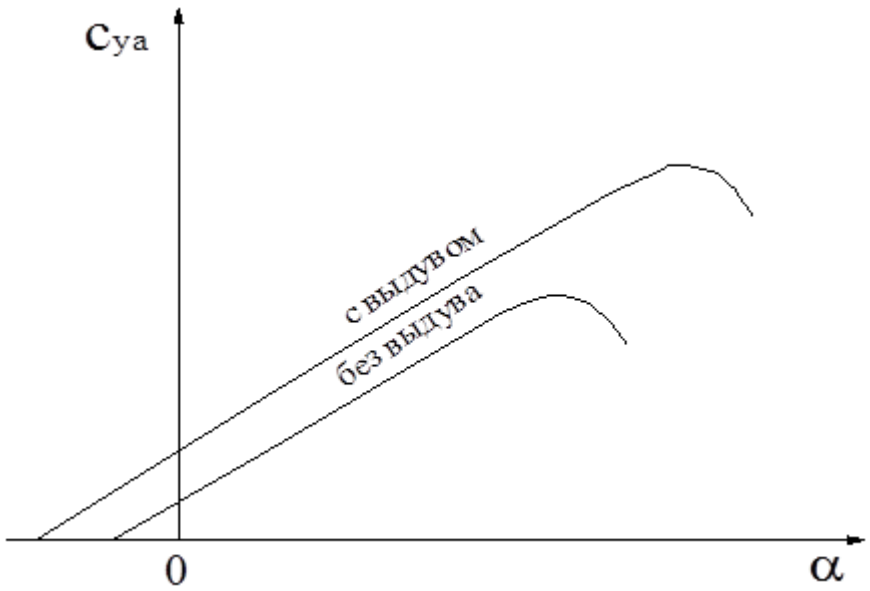

Рис. 2.18. Порівняння залежності Суа від а при видуві струї повітря та без неї

Видувши струменів уздовж передньої кромки призводить до виникнення на ній вихрових джгутів або збільшення їх інтенсивності (якщо на передній кромці є відрив), що підвищує несучі властивості крила за рахунок додаткового розрядження на верхній поверхні крила Ріс.10.9 Крім того руйнування цих джгутів затягується на великі кути атаки, тобто збільшує Відхилення механізації завжди призводить до збільшення лобового опору. Використання струменевої і струменево - ежекторної механізації призводить до відносно меншого збільшення опору в порівнянні з іншими видами механізації, так як в цих випадках сила реакції струменя зменшує опір. Вплив відхилення механізації на аеродинамічну якість неоднозначно. При малих кутах відхилення механізації, які використовують на зльоті, аеродинамічна якість зазвичай дещо підвищується, а при великих кутах відхилення механізації за рахунок значного зростання опору аеродинамічну якість падає Це падіння при використанні щілинних закрилків менше, ніж при відхиленні простих закрилків, а в разі струменевих і струменево - ежекторних ще менше. Крім несучих властивостей і опору механізація крила впливає і на моментні характеристики ЛА. При відхиленні механізації задньої кромки основний приріст аеродинамічного навантаження спостерігається в районі розташування механізації, в результаті центр тиску зміщується назад, що призводить до появи додаткового пікіруючого моменту. З іншого боку, через розташування механізації у кореневих перетинів при її відхиленні в цих перетинах збільшується скіс потоку, підйомна сила горизонтального оперення збільшується і з'являється момент на кабрування. Залежно від особливостей аеродинамічного компонування ЛА сумарний момент від механізації може мати різний знак. Затягування кінцевого зриву на стрілоподібним крилом за рахунок відхилення механізації передньої кромки покращує моментні характеристики крила.

Відхилення механізації крила змінює його кривизну, тому практично не впливає на похідну C^{a}_{ya} на лінійній ділянці залежності Cy = f (a). Але при

цьому змінюється а0. Вплив форми крила, бойових і експлуатаційних пошкоджень на ефективність механізації Приріст коефіцієнта Суа за рахунок відхилення закрилків (або інших видів механізації) можна визначити з залежності, де аоз - зміна а0 за рахунок відхилення механізації. Форма крила в плані впливає на суаа, а аоз визначається типом і кутом відхилення механізації і розташуванням її на крилі. Зменшення подовження і збільшення стрілоподібності крила призводить до зменшення суаа і, отже, до зменшення Dcуаз

Стреловидність задньої кромки впливає на ефективність механізації аналогічно стрілоподібності передньої кромки. У зв'язку з цим іноді у стреловидних крил в районі розміщення закрилків спрямовують задню кромку. Крім того, для зменшення шкідливого впливу малого подовження і великий стрілоподібності збільшують звуження крила При цьому збільшується відносна площа крила, яку обслуговує механізацією. Закрилки зазвичай розміщують в районі кореневих перетинів крила. Хорди закрилків складають до 35% хорд крила [13].

Хвостове оперення

Оперення (оперення літального апарату, стріли) - називаються аеродинамічні поверхні, що забезпечують стійкість, керованість і балансування літака в польоті. Воно складається з горизонтального і вертикального оперення. До оперення зазвичай відносять і елерони - органи поперечної керованості і балансування.

Призначення хвостового оперення:

Створення моменту забезпечує стійкість, балансування, керованість літака.

Стійкість - здатність ВС самостійно без участі льотчика зберігати заданій режим польоту.

Керованість - здатність літака виконувати за бажанням льотчика у відповідь на його дію будь-маневреності передбачених умовами льотної експлуатації.

Вертикальне оперення забезпечує шляхову стійкість, керованість і балансування.

Горизонтальне оперення забезпечує поздовжню стійкість, керованість і балансування

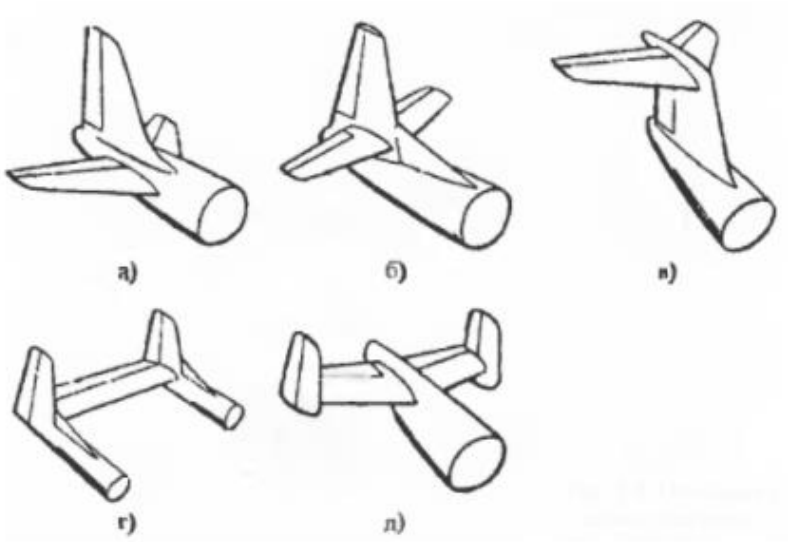

Рис. 2.19. Види хвостового оперення

Найбільш поширеною є схема з одним кілем і стабілізатором, встановленим на фюзеляжі або кілі - (Рис 4 а, б, в). Вона забезпечує конструктивну простоту і жорсткість, хоча в разі Т-образного хвостового оперення (Рис 4в) необхідно вживати заходів, що запобігають флаттеру.

Схема Т-образного оперення має і низку привілеїв. Розташування горизонтального оперення у верхній частині кіля створює для останнього ефект кінцевий шайби, що може сприяти зменшенню потрібної площі вертикального оперення. З іншого боку високорозташованого горизонтальне оперення знаходиться в зоні невеликого скоса потоку від крила при середніх (польотних) кутах атаки, що дозволяє зменшити потрібну площу

горизонтального оперення. Таким чином площа Т-образного оперення може бути менше площі оперення з низьким розташуванням горизонтального оперення.

Призначення хвостового оперення - створення моментів; забезпечує стійкість, керованість і балансування літака. Стійкість - здатність літака самостійно, без участі льотчиків, зберігати заданий режим польоту. Керованість - здатність літака виконувати побажання льотчика у відповідь на його дії (будь-маневр, передбачений умовами льотної експлуатації). Балансування - приведення до нуля (щодо центра ваги) суми всіх сил і моментів, що діють на літак.

Переваги V-образного оперення: 1) полегшення ваги; 2) зниження опору.

Недоліки V-образного оперення: 1) ускладнення механізму відхилення рульових поверхонь; 2) збільшення крутильних навантажень на планер при маневруванні; 3) висока аварійність.

Вертикальне оперення складається з нерухомого кіля, форкіль, підфюзеляжного гребеня, керма напряму.

Кіль стабілізує політ, приблизно як оперення у стріли. Без кіля літак смикався б в різні боки. На кілі знаходяться рулі, керуючі рисканням. Два невеликих горизонтально розташованих крила (стабілізатор) стабілізують літак по вертикалі.

Форкіль збільшує шляхову статичну стійкість літака на великих кутах ковзання.

Підфюзеляжний гребінь. Один або кілька підфюзеляжних гребенів підвищують шляхову статичну стійкість літака на великих кутах атаки. Щоб забезпечити більш сприятливі умови для зльоту, посадки і стоянки літального апарату, ці гребені можуть виконуватися складаються (забираються).

Кермо напряму. Призначене для керування літаком щодо нормальної осі (тобто за допомогою керма напряму змінюється кут рискання - кутові руху літального апарата щодо вертикальної осі, а також невеликі зміни курсу вправо або вліво, властиві судну).

Горизонтальне оперення складається з стабілізатора і керма висоти.

Стабілізатор. Забезпечення стійкості у вертикальній площині (забезпечує поздовжню стійкість польоту). При випадковому відхиленні літака від стану рівноваги кут атаки горизонтального оперення змінюється так, що діюча на неї сила вирівнює літак, повертає його до попереднього стану.

Кермо висоти. Призначена для забезпечення поздовжньої керованості літака (аеродинамічний орган управління літака, який здійснює його обертання навколо поперечної осі).

Б) суцільноповоротним стабілізатор служить для забезпечення стійкості і керованості. Суцільноповоротним стабілізатор зазвичай встановлюється на надзвукових літаках, коли недостатньо ефективні рулі висоти при польотах на великих висотах; він підвищує маневреність літака на надзвукових режимах. Так що використання такого стабілізатора не є раціональним для звичайних вантажних і пасажирських літаків.

Висновок до розділу 2

У розділі було розглянута основні етапи зльоту, посадки та польоту по колу, а також основні аеродинамічні сили та характеристики, які впливають на побудову траєкторій БПЛА.

РОЗДІЛ 3
UNITY ЯК СЕРЕДОВИЩЕ РОЗРОБКИ

3.1 Загальна характеристика

Unity - це крос-платформний ігровий движок, розроблений Unity Technologies, вперше анонсований і випущений в червні 2005 року на всесвітній конференції розробників Apple Inc. як ексклюзивний ігровий движок для Mac OS X. Станом на 2018 рік двигун був розширений для підтримки більш ніж 25 платформ. Механізм може бути використаний для створення тривимірних, двовимірних ігор у віртуальну реальність та доповнену реальність, а також моделювання та іншого досвіду. Двигун був прийнятий на виробництво за межами відеоігор, таких як кіно, автомобільна промисловість, архітектура, машинобудування та будівництво.

Unity надає користувачам можливість створювати ігри та досвід як у 2D, так і в 3D, а движок пропонує основний API сценаріїв на C# як для редактора Unity у формі плагінів, так і для самих ігор, а також функцію перетягування. До того, як C # був основною мовою програмування, що використовувалася для движка, він раніше підтримував Boo, який було видалено випуском Unity 5 та версію JavaScript під назвою UnityScript, яка була припинена в серпні 2017 року після випуску Unity 2017.1, на користь C#.

У 2D-іграх Unity дозволяє імпортувати спрайти та вдосконалений 2D-світовий рендерінг.

Для 3D-ігор Unity дозволяє визначити стиснення текстур, мипмапи та налаштування роздільної здатності для кожної платформи, яку підтримує ігровий движок, і забезпечує підтримку відображення ударних зображень відображення відображення, відображення паралакса, оклюзія навколишнього середовища екрану (SSAO), динамічний тіні з використанням карт тіней, рендерингу до текстури та повноекранних ефектів післяобробки.

3.2. Особливості середовища в розробці авіасимуляції

3.2.1. Vector2

Представлення двовимірних векторів і точок.

Ця структура використовується в деяких місцях для представлення 2D-позицій та векторів (наприклад, координати текстур у Mesh або зміщення текстур у Material). У більшості інших випадків використовується Vector3 [14].

3.2.2. Vector3

Представлення тривимірних векторів і точок.

Ця структура використовується в Unity для передачі 3D-позицій та напрямків навколо. Вона також містить функції для виконання загальних векторних операцій. Окрім перелічених нижче функцій, для маніпулювання векторами та точками також можна використовувати інші класи. Наприклад, класи Quaternion і Matrix4x4 корисні для обертання або перетворення векторів і точок [15].

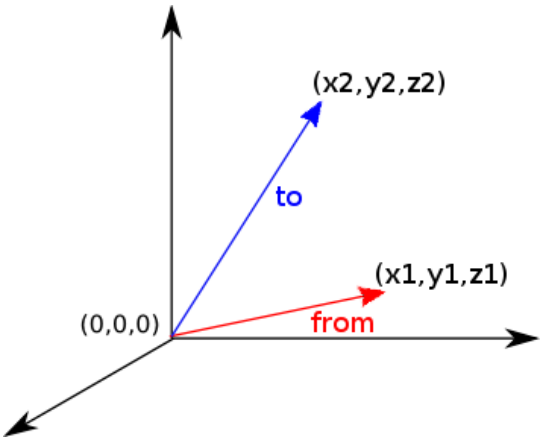

Рис. 3.1. Представлення трьовимірного вектору в Unity

3.2.3. Rigidbody

Контроль положення об'єкта за допомогою фізичного моделювання.

Додавання компонента Rigidbody до об’єкта поставить його рух під контроль фізичного механізму Unity. Навіть без додавання будь-якого коду, об’єкт з **Rigidbody** буде тягнути вниз гравітацією і реагуватиме на зіткнення з об’єктами, що надходять, якщо також присутній правий компонент Колайдера.

Rigidbody також має сценарій API, який дозволяє застосовувати сили до об'єкта та управляти ним фізично реалістично. Наприклад, поведінка повітряного судна може бути визначена з точки зору сил, що застосовуються навколишнім середовищем. З огляду на цю інформацію, фізичний двигун може обробляти більшість інших аспектів руху літака, тому він буде реалістично прискорюватися і правильно реагувати на зіткнення. Вид дерева **Rigidbody** представлений на рис. 3.2.

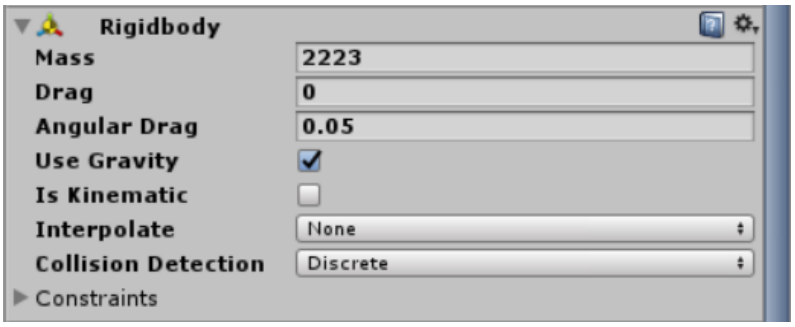

Рис. 3.2. Rigidbody дерево

У сценарії рекомендується використовувати функцію FixedUpdate як місце для застосування сил та зміни параметрів жорсткого тіла (на відміну від оновлення, яке використовується для більшості інших завдань оновлення кадру). Причиною цього є те, що оновлення фізики здійснюється за виміряні тимчасові кроки, які не збігаються з оновленням кадру. FixedUpdate викликається безпосередньо перед кожним оновленням фізики, тому будь-які внесені там зміни будуть оброблятися безпосередньо.

Поширеною проблемою, починаючи з Rigidbodies, є те, що фізика гри, здається, працює у "повільному темпі". Це насправді пов’язано з масштабом, який використовується для моделей. За замовчуванням параметри гравітації передбачають, що одна світова одиниця відповідає одному метру відстані. Для нефізичних ігор не має великої різниці, якщо всі ваші моделі мають довжину 100 одиниць, але при використанні фізики вони будуть розглядатися як дуже великі предмети. Якщо для об’єктів, які вважаються малими, застосовується великий масштаб, вони, здається, падають дуже повільно - фізичний двигун вважає, що це дуже великі об’єкти, що падають на дуже великі відстані. З огляду на це, не потрібно забувати тримати свої об’єкти більш-менш у їх масштабі в реальному житті (так, автомобіль повинен складати приблизно 4 одиниці = 4 метри, наприклад) [16].

3.2.4. Rigidbody.rotation

Обертання твердого тіла.

Rigidbody.rotation Використовується, щоб отримати та встановити обертання Rigidbody за допомогою фізичного механізму.

Зміна обертання твердого тіла за допомогою Rigidbody.rotation оновлює перетворення після наступного кроку фізичного моделювання. Це швидше, ніж оновлення обертання за допомогою Transform.rotation, оскільки Transform.rotation змушує всі приєднані Colliders перерахувати обертання відносно Rigidbody, тоді як Rigidbody.rotation встановлює значення безпосередньо до фізичної системи [17].

3.2.5. Rigidbody.AddForce

Додає сили до твердого тіла.

Сила подається безперервно вздовж напрямку вектора сили. Вказівка режиму ForceMode дозволяє змінити тип сили на прискорення, імпульс або зміну швидкості.

Застосовувана сила обчислюється у FixedUpdate або явним викликом методу Physics.Simulate.

Сила може бути застосована лише до активного твердого тіла. Якщо GameObject неактивний, AddForce не впливає. Крім того, тверде тіло не може бути кінематичним.

За замовчуванням стан твердого тіла встановлюється як пробуджений, коли застосовується сила, якщо сила не є Vector3.zero [18].

3.2.6. GameObject

GameObjects - це основні об'єкти в Unity, які представляють ассети. GameObjects виступають контейнерами для компонентів, які реалізують реальну функціональність.

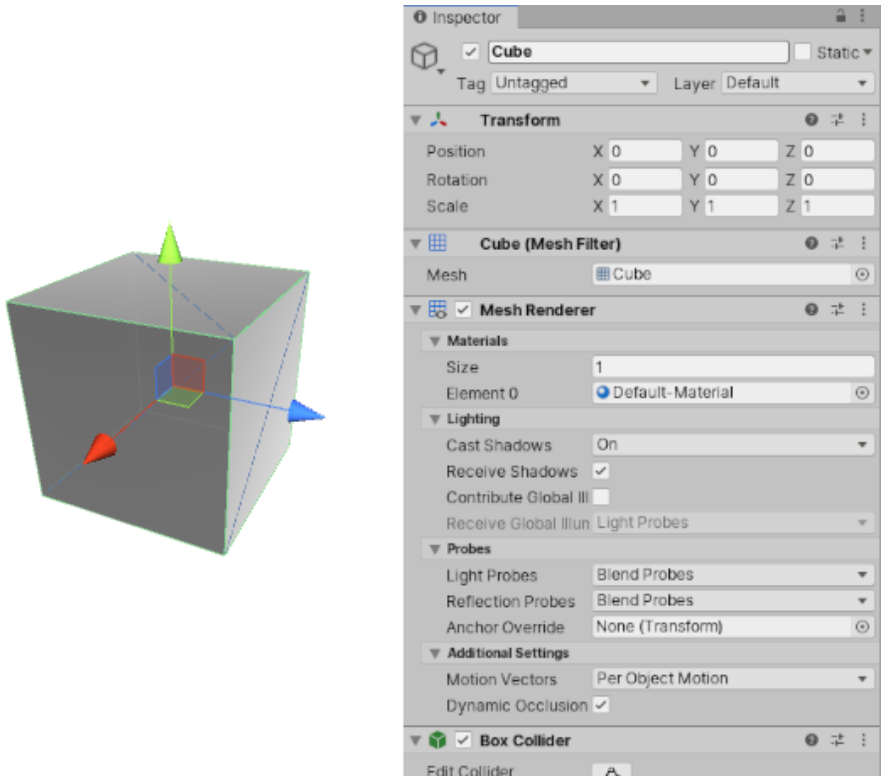

Рис. 3.3. Представлення GameObject в Unity

GameObject завжди має приєднаний компонент Transform (для представлення положення та орієнтації), і це неможливо видалити. Інші компоненти, що надають об'єкту функціональність, можна додати з меню "Компонент" редактора або зі сценарію. У меню GameObject> 3D Object також доступно багато корисних заздалегідь побудованих об'єктів (примітивні фігури, камери тощо) [19].

GameObjects - це будівельні блоки для сцен в Unity і виступають контейнером для функціональних компонентів, які визначають, як виглядає GameObject і що робить GameObject.

У сценаріях клас GameObject надає набір методів, що дозволяють працювати з ними у коді, включаючи пошук, встановлення з'єднань та надсилання повідомлень між GameObjects, а також додавання або видалення компонентів, приєднаних до GameObject, і встановлення значень, що стосуються їхній статус на місці події.

Є кілька властивостей, які ви можете змінити за допомогою сценарію, які стосуються статусу GameObject на сцені. Вони, як правило, відповідають елементам управління, видимим у верхній частині інспектора коли в редакторі вибрано GameObject.

GameObjects активні за замовчуванням, але їх можна деактивувати, що відключає всі компоненти, приєднані до GameObject. Це, як правило, означає, що він стане невидимим і не отримає жодних звичайних зворотних викликів або подій, таких як Update або FixedUpdate.

Активний статус GameObject представлений прапорцем ліворуч від імені GameObject. Ви можете керувати цим за допомогою GameObject.SetActive.

Також можливо прочитати поточний активний стан за допомогою GameObject.activeSelf, а також чи дійсно GameObject активний у сцені за допомогою GameObject.activeInHierarchy. Останнє з цих двох необхідне, оскільки те, чи є GameObject насправді активним, визначається його власним активним станом плюс активний стан усіх його батьків. Якщо хтось із батьків не активний, тоді він не буде активним, незважаючи на власні активні налаштування.

3.2.7. Кватерніони та кути Ейлера

Повороти в 3D-програмах зазвичай представлені одним із двох способів - кватерніонами або кутами Ейлера. Кожен має своє власне використання та недоліки. Unity використовує Quaternions внутрішньо, але

показує значення еквівалентних кутів Ейлера в інспекторі, щоб полегшити вам редагування.

Кути Ейлера мають більш просте зображення, тобто три значення кута для X, Y та Z, які застосовуються послідовно. Щоб застосувати обертання Ейлера до певного об'єкта, кожне значення обертання застосовується по черзі як обертання навколо своєї відповідної осі.

• Перевага: кути Ейлера мають інтуїтивно зрозумілий для читання формат, що складається з трьох кутів;

• Перевага: кути Ейлера можуть представляти обертання від однієї орієнтації до іншої через поворот більше 180 градусів;

• Обмеження: кути Ейлера страждають від Gimbal Lock. Застосовуючи три обертання по черзі, можливо, що при першому або другому обертанні третя вісь буде спрямована в тому ж напрямку, що і одна з попередніх осей. Це означає, що «ступінь свободи» втрачено, оскільки третє значення обертання не може застосовуватися навколо унікальної осі.

Кватерніони можна використовувати для представлення орієнтації або обертання об'єкта. Це представлення внутрішньо складається з чотирьох чисел (в Unity згадуються як x, y, z & w), однак ці числа не представляють кутів чи осей, і зазвичай вам ніколи не потрібно безпосередньо отримувати до них доступ. Якщо вас особливо не цікавить заглиблення в математику кватерніонів, вам дійсно потрібно лише знати, що кватерніон являє собою обертання в тривимірному просторі, і вам ніколи не знадобиться знати або змінювати властивості x, y & z.

Так само, як Вектор може представляти або позицію, або напрямок (де напрям вимірюється від початку координат), Кватерніон може представляти або орієнтацію, або обертання - де обертання вимірюється від обертального "початку" або " Ідентичність ”. Через те, що обертання вимірюється таким чином - від однієї орієнтації до іншої - кватерніон не може представляти обертання понад 180 градусів.

• Перевага: обертання кватерніону не страждають від блокування Gimbal;

• Обмеження: окремий кватерніон не може представляти обертання, що перевищує 180 градусів в будь-якому напрямку;

• Обмеження: чисельне представлення кватерніону інтуїтивно не зрозуміле [20].

3.2.8. Вікно Inspector

Проекти в редакторі Unity складаються з безлічі GameObjects, які містять сценарії, звуки, сітки та інші графічні елементи, такі як Lights. У вікні Inspector (іноді його називають «Інспектором») відображається детальна інформація про вибраний на даний момент GameObject, включаючи всі вкладені компоненти та їх властивості, а також дозволяє змінювати функціональність GameObjects у Вашій Сцені.

Використовуйте інспектор для перегляду та редагування властивостей та налаштувань майже всього в редакторі Unity, включаючи фізичні елементи гри, такі як GameObjects, Assets та Materials, а також налаштування та налаштування в редакторі.

Коли ви вибираєте GameObject в режимі ієрархії або сцени, інспектор відображає властивості всіх компонентів та матеріалів цього GameObject. Використовуйте інспектор для редагування налаштувань цих компонентів та матеріалів. Погляд дерева інспектора представлений на рис. 3.4.

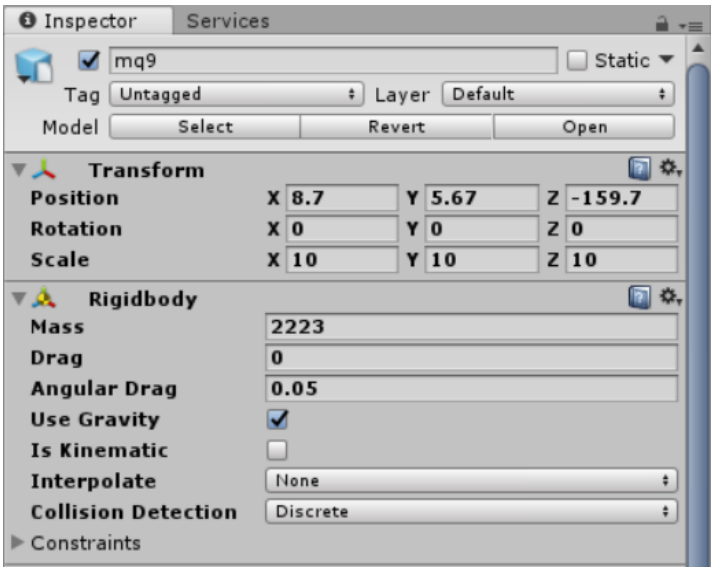

Рис. 3.4. Вікно Inspector

Коли до GameObjects приєднані власні компоненти сценарію, Інспектор відображає загальнодоступні змінні цього сценарію. Ви можете редагувати ці змінні як параметри так само, як і редагувати параметри вбудованих компонентів редактора. Це означає, що ви можете легко встановлювати параметри та значення за замовчуванням у своїх сценаріях, не змінюючи код [21]. Вид сценарію представлений на рис. 3.5.

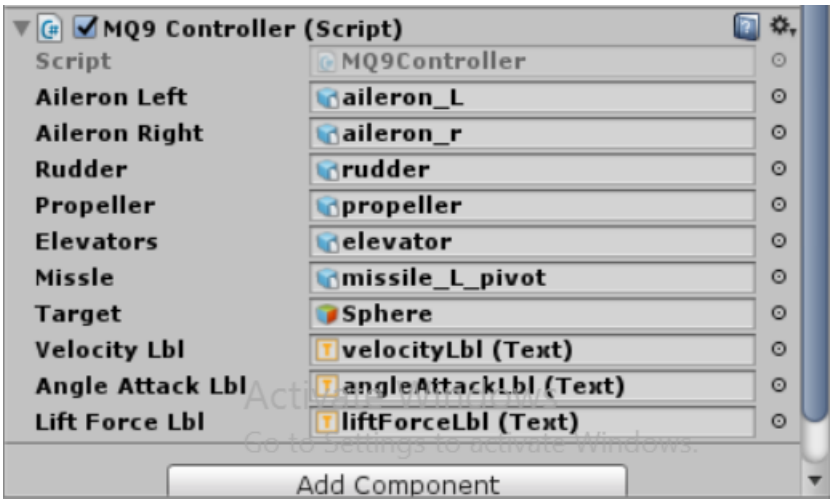

Рис. 3.5. Дерево скрипту

3.2.9. Unity Assets

Asset - це представлення будь-якого предмета, який можна використовувати у грі чи проекті. Asset може походити з файлу, створеного за межами Unity, наприклад 3D-моделі, аудіофайлу, зображення або будь-якого іншого типу файлу, який підтримує Unity. Є також деякі типи асетів, які можна створити в Unity, наприклад, контролер Animator, аудіомікшер або рендеринг текстури. На рис. 3.6 зображено приклад імпорту асетів в Unity [22].

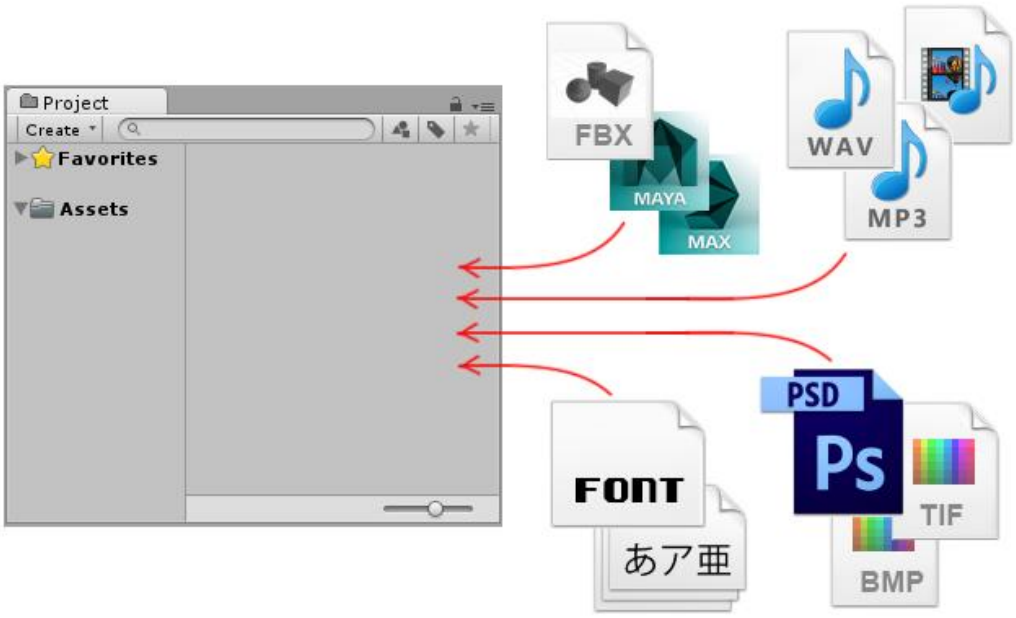

Рис. 3.6. Приклад імпорту асетів в Unity

Створення ассетів

Використати будь-який підтримуваний пакет програм для 3D моделювання, щоб створити чорнову версію Ассету. Підготувати і зберегти Ассет.

Імпорт

При першому збереженні Ассету, його варто зберегти в папку Assets, яка знаходиться в папці проекту. При відкриванні проекту в Unity,

збережений Ассет буде виявлений і імпортовано. Якщо подивитися у вікно Project View, то виявиться, що Ассет розташований саме там, де його зберегли. Unity використовує FBX-експортер, вбудований в програму 3D моделювання, щоб конвертувати моделі в формат FBX. Тому необхідно, щоб FBX-експортер використовуваної програми 3D моделювання був доступний для Unity. В іншому випадку, можна відразу експортувати моделі в FBX формат, і зберігати в папці проекту FBX файли.

Налаштування імпорту

Якщо вибрати Ассет у вікні Project View, то на панелі Inspector відобразяться настройки імпорту для даного Ассету. Доступні для налаштування опції імпорту змінюються в залежності від типу обраного Ассету.

Додавання Ассету в сцену

Необхідно перетягнути мишкою меш з вікна Project View в вікно Hierarchy або Scene View для того, щоб додати його в Сцену. При перетягуванні в сцену, створюється ігровий об'єкт (GameObject), у якого є компонент Mesh Renderer. Якщо йде робота з текстурою або звуковим файлом, доведеться додавати їх в уже створений GameObject в сцені або в вікні Project.

Угруповання різних Ассетів разом

Нижче наведено короткий опис зв'язків між найбільш часто використовуваними типами Ассет.

Текстура застосовується в матеріалі (Material)

Матеріали використовуються в GameObject'ах (в компоненті Mesh Renderer)

Анімації використовуються для ігрових об'єктів (в компоненті Animation)

Звукові файли використовуються з ігровими об'єктами (компонент Audio Source)

Оновлення Ассетів

Імпортувавши, інстанціювавши і прив'язавшиш Ассет до префаб є можливість змінити вихідний Ассет, двічі клацнувши на ньому мишкою в панелі Project. Запуститься відповідний додаток і можна буде внести в нього будь-які правки. Коли закінчується правка Ассету, необхідно зберегти його. Повернувшись в Unity, оновлення Ассет буде виявлено автоматично, і він буде переімпортірований. При цьому зв'язок між Ассет і префаб, який його використовує залишиться. Таким чином, префаб оновиться.

3.2.10. Сцени

Сцени містять об'єкти або ассети проекту. З їх допомогою можна створити головне меню, окремі рівні та будь-що інше. У кожній сцені можливо розмістити своє середовище, перешкоди та об’єкти, по суті розробляючи та будуючи проект по частинах [23].

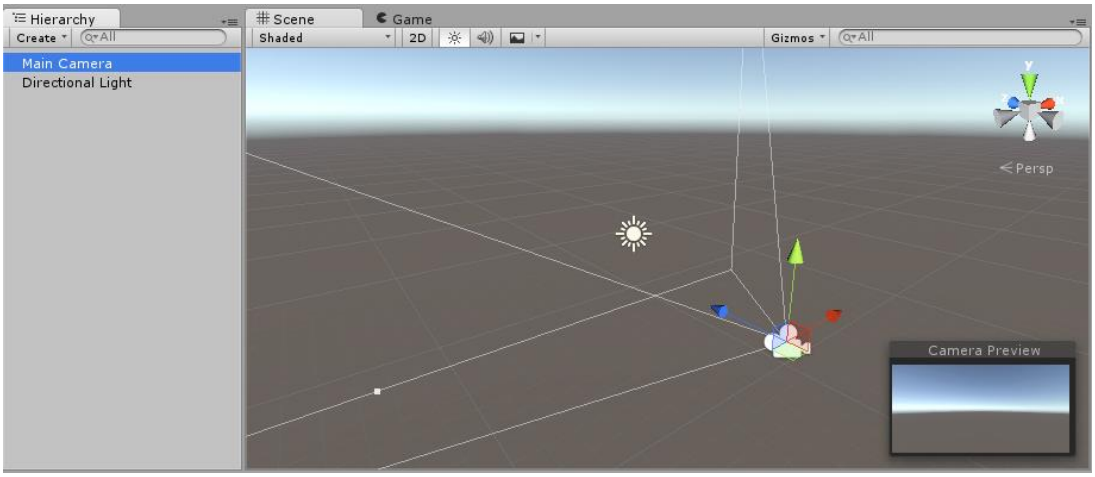

Рис. 3.7. Приклад сцени

Сцени складаються з об'єктів проекту. Вони можуть бути використані для створення головного меню, окремих рівнів і всього іншого. Кожен файл

сцени можна розглядати як окремий рівень. У кожній сцені Ви розміщується оточення, перешкоди, декораці.

Для створення екземпляра префаб перетягніть його з вікна Проекту в Ієрархію або Сцену. Тепер Ви маєте екземпляр Вашого префаб і можетет його позиціонувати і налаштувати за бажанням.

Після вибору префаб або будь-якого ігрового об'єкта можливо додати функціональність до нього використовуючи Компоненти. Скрипти - це теж один з типів компонентів. Для додавання компонента просто виберіть потрібний об'єкт, а потім компонент з Меню компонентів. Після цього Ви побачите компонент в вікні інспектор. Скрипти також містяться в Меню компонентів.

Якщо компонент розриває зв'язок об'єкта з префаб, Ви завжди можете використовувати GameObject-> Apply Changes to Prefab з меню для відновлення зв'язку.

Після розміщення об'єкта в сцені Ви можете використовувати Інструмент Transform для позиціонування його. Додатково Ви можете безпосередньо задавати значення в Інспектора для точного размещенеія і повороту. Прочитайте сторінку Компонент Transform для додаткової інформації про позиціонування і повороті об'єктів.

Все що бачить гравець програється однієї або декількома камерами. Камера це звичайний об'єкт гри з прикріпленим до нього компонентом камери. Ви можете позиціонувати, повернути або удочерити камеру точно так само як будь-який іно об'єкт гри. Тобто вона може робити все, що роблять звичайні об'єкти, плюс деякі специфічні для камер функції. Є деякі корисні скрипти камер поставляються з Юніті. Вони можуть бути включені в проект відразу, або можливо використовувати меню Assets-> Import Package ... Скрипти, які Ви імпортуєте можуть бути знайдені в меню Components-> Camera-Control.

3.2.11. Колайдери

Компоненти колайдера визначають форму GameObject для цілей фізичних зіткнень. Колайдер, який невидимий, не повинен мати абсолютно однакову форму, як сітка GameObject. Груба апроксимація сітки часто є більш ефективною і не відрізняється в ігровому процесі.

Найпростіші (і найменш трудомісткі) коллайдери - це примітивні типи коллайдерів. У 3D це Box Collider, Sphere Collider та капсульний колайдер. У 2D можна використовувати Box Collider 2D та Circle Collider 2D [24].

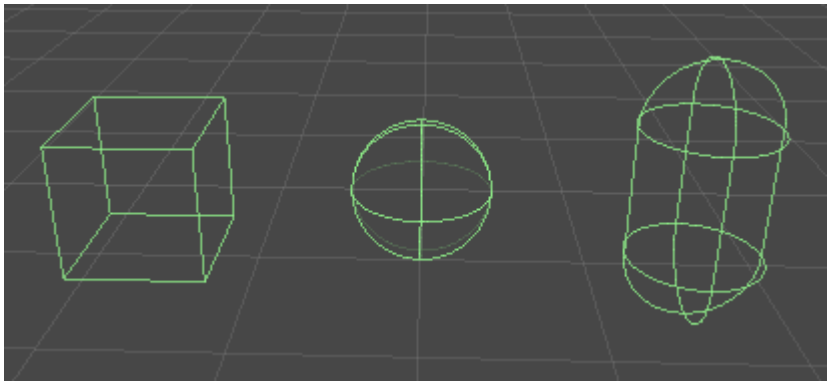

Рис. 3.8. Типи колайдерів в Unity

Будь-яку їх кількість можна додати до одного об’єкта для створення складених колайдерів.

При ретельному розташуванні та розмірі складні коллайдери часто можуть досить добре наблизити форму предмета, зберігаючи низькі витрати процесора. Подальшу гнучкість можна отримати, встановивши додаткові колайдери на дочірніх об'єктах (наприклад, коробки можна повертати щодо локальних осей батьківського об'єкта). Створюючи подібний складний колайдер, повинен бути лише один компонент Rigidbody, розміщений на кореневому об’єкті в ієрархії.

Примітивні колайдери не працюватимуть належним чином із зсувними перетвореннями - це означає, що якщо ви використовуєте комбінацію

обертань та нерівномірних масштабів в ієрархії трансформації, так що отримана фігура більше не буде відповідати примітивній формі, примітивний коллайдер не буде вміти правильно його представляти.

Однак є деякі випадки, коли навіть складні коллайдери недостатньо точні. У 3D ви можете використовувати Mesh Colliders, щоб точно відповідати формі сітки об’єкта. У 2D, Polygon Collider 2D, як правило, не буде повністю відповідати формі графіки спрайтів, але ви можете уточнити форму до будь-якого рівня деталізації. Однак ці коллайдери набагато інтенсивніше процесора, ніж примітивні типи, тому використовуйте їх економно, щоб підтримувати хорошу продуктивність. Крім того, сітчастий колайдер, як правило, не зможе зіткнутися з іншим сітчастим колайдером (тобто нічого не трапиться, коли вони вступить в контакт). У деяких випадках це можна обійти, позначивши сітчастий колайдер як опуклий в інспекторі. Це генерує форму колайдера у вигляді «опуклої оболонки», яка схожа на вихідну сітку, але з будь-якими заповненнями. коли у вас є рухомий персонаж з відповідною формою. Однак хорошим загальним правилом є використання сітчастих коллайдерів для геометрії сцени та наближення форми рухомих об’єктів за допомогою складених примітивних коллайдерів.

Колайдери можна додати до об'єкта без компонента Rigidbody для створення підлог, стін та інших нерухомих елементів сцени. Вони називаються статичними коллайдерами. Загалом, вам не слід переставляти статичні коллайдери, змінюючи положення перетворення, оскільки це суттєво вплине на продуктивність фізичного двигуна. Колайдери на об'єкті, що має тверде тіло, відомі як динамічні коллайдери. Статичні коллайдери можуть взаємодіяти з динамічними коллайдерами, але оскільки вони не мають твердого тіла, вони не рухатимуться у відповідь на зіткнення.

3.2.12. Камера

Камера - це пристрій, за допомогою якого об’єкт дивиться на світ.

Точка простору екрана визначається у пікселях. У нижньому лівому куті екрана (0,0); справа вгорі (pixelWidth, pixelHeight). Позиція z знаходиться у світових одиницях від Камери.

Точка простору області огляду нормалізована і відносно Камери. У нижньому лівому куті камери (0,0); угорі праворуч (1,1). Позиція z знаходиться у світових одиницях від Камери.

Космічна точка світу визначена в глобальних координатах (наприклад, Transform.position) [25].

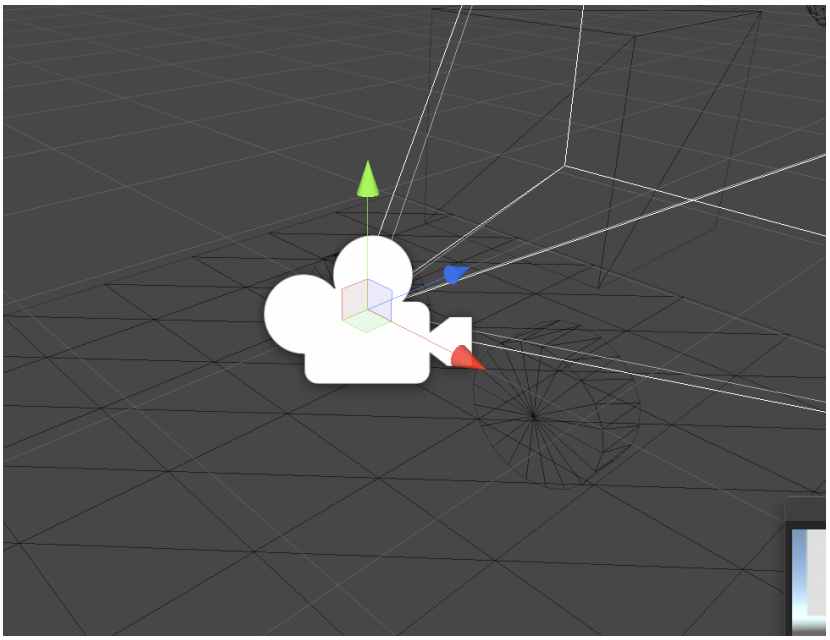

Рис. 3.9. Представлення камери в Unity

Камери є пристроями, які захоплюють і відображають світ. Шляхом налаштування і маніпулювання камерами, можливо зробити презентацію свого проекту унікальним. Ви можете мати необмежену кількість камер в сцені. Ви можете налаштувати рендеринг камерами в будь-якому порядку, на будь-якому місці екрану, або тільки в певних частинах екрану.

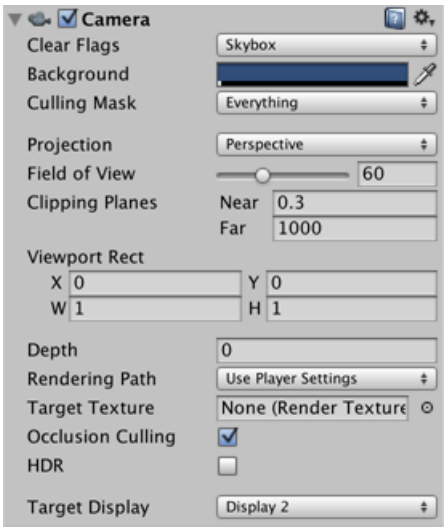

Рис. 3.10. Налаштування камери

3.2.13. Префаби

Система Unity Prefab дозволяє створювати, налаштовувати та зберігати GameObject укомплектований усіма його компонентами, значеннями властивостей та дочірніми GameObjects як багаторазовий актив. Актив Prefab діє як шаблон, за допомогою якого можливо створювати нові екземпляри Prefab в Scene.

Якщо потрібно повторно використати GameObject, сконфігурований певним чином - як неігровий персонаж (NPC), реквізит чи шматок декорації - у кількох місцях сцени або в декількох сценах проекту, слід перетворити його на Збірні.

Будь-які зміни, внесені в об’єкт Prefab, автоматично відображаються в екземплярах цього Prefab, що дозволяє легко вносити широкі зміни у весь проект без необхідності багаторазового внесення однакових змін у кожну копію об’єкта.

Можливо вкласти Prefabs всередині інших Prefabs, щоб створити складні ієрархії об’єктів, які легко редагувати на декількох рівнях.

Однак це не означає, що всі екземпляри Prefab повинні бути однаковими. Можливо замінити налаштування окремих екземплярів збірних панелей, щоб деякі екземпляри збірної панелі відрізнялися від інших. Також можливо створити варіанти збірних панелей, які дозволяють згрупувати набір перевизначень разом у значущі варіанти збірних.

Також слід використовувати Prefabs, коли є необхідність створити екземпляр GameObjects під час виконання, який не існував у Сцені на початку [26].

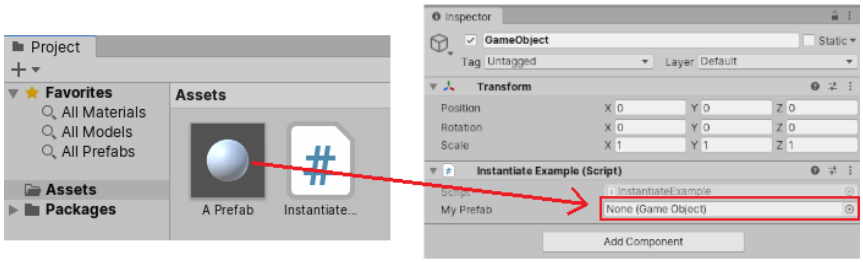

Рис. 3.11. Створення префабу

Створення префабів

Префаб - це збережена колекція, яка містить один або більше повних GameObject'ов з приєднаними компонентами і встановленими властивостями. Префаб - це типи Ассетів, тому вони не відображаються в сцені самі по собі. Але вони можуть бути використані, щоб створювати екземпляри збережених об'єктів в сцені. Кожен екземпляр - це копія оригінального префаб. Наприклад, ви можете використовувати префаб для збереження об'єкта дерева, а потім створити безліч екземплярів цього дерева в сцені лісу.

За замовчуванням, зміни зроблені в префаб, автоматично застосовуються до всіх його екземплярів, тому використання префабов є хорошим способом підтримувати однорідність набору об'єктів. Тим не менш, ви також можете відключити зв'язок між екземпляром і префаб, якщо вам потрібно зробити спеціальні варіації оригіналу. Ви також можете змінити

екземпляр, і зберегти зміни в префаб (меню: GameObject> Apply Changes to Prefab).

Щоб створити префаб з GameObject у сцені, необхідно перетягнути GameObject зі сцени в вікно Project. Після цього ім'я GameObject'a виділиться, щоб можливо було його перейменувати.

Висновок до розділу 4

У цьому розділі було описано основні можливості Unity у розробці програми для побудови траєкторії зльоту/посадки.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМИ ДЛЯ ПОБУДОВИ ТРАЄКТОРІЇ ПОЛЬОТУ ЗЛІТ/ПОСАДКА НА ПЛАТФОРМІ UNITY

4.1. Інтеграція 3D-моделі MQ-9 Reaper в Unity

Готова **3D-модель** MQ-9 Reaper може бути імпортована в Unity та використовуватися за допомогою ассетів, описаних у розділі 3.

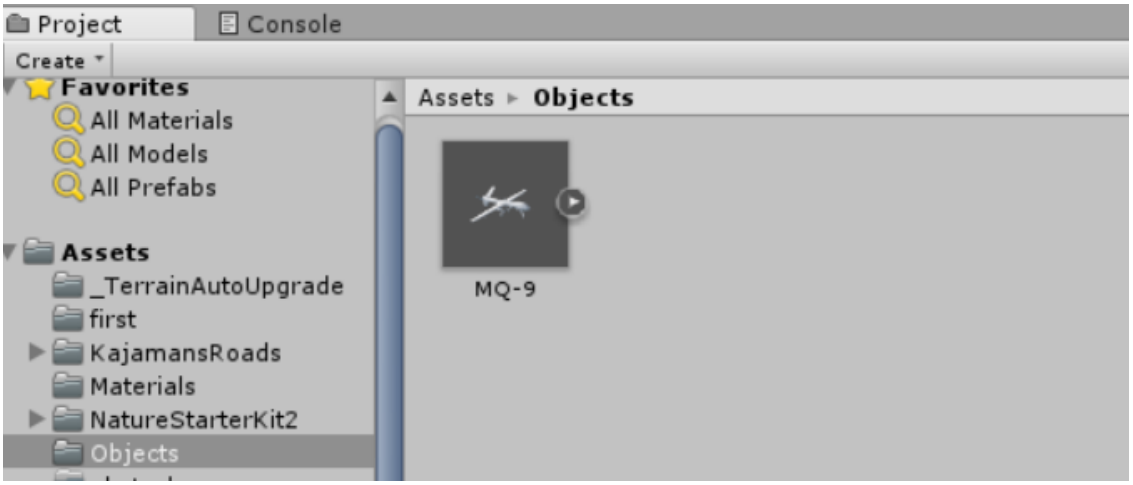

Рис. 4.1. Імпорт 3D-моделі MQ-9 Reaper в Unity ассети

Після успішної інтеграції необхідно створити сцену та імпортувати в неї раніше створений ассет.

Рис.4.2. 3-D модель MQ-9 Reaper в сцені Unity

Інтегрувавши 3D-модель, є можливість контролювати окремі частини об’єкта. У цьому випадку можна керувати елеронами, елеваторами та кермом висоти, якщо літальний апарат змінює тангаж, крен, або рискання.

Спершу необхідно створити .cs скрипт та закріпити його за імпортованим раніше ассетом. Розроблений скрипт приведений в додатку А.

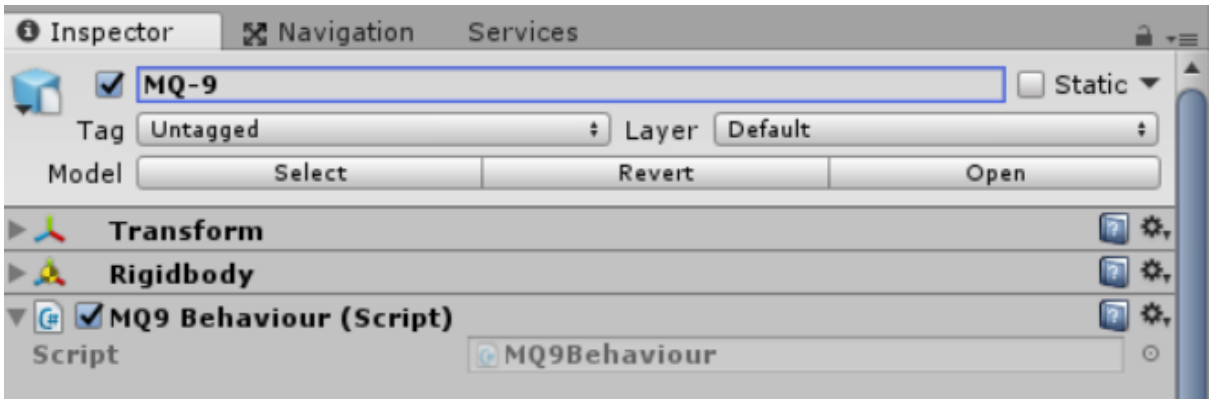

Рис. 4.3. Створення скрипту для ассету MQ-9

Далі, для більш зручного контролю окремих частин об’єкту створимо заголовок та змінні для того, щоб присвоїти їм посилання на GameObject.

```
[Header("Mechanics")]
public GameObject AileronLeft;
public GameObject AileronRight;
public GameObject Rudder;
public GameObject Propeller;
public GameObject Elevators;
public GameObject Target;
public GameObject Runway;
public GameObject GlisadePoint;
public List<GameObject> CirclePoints;

public GameObject Rear1;
public GameObject Rear2;
```

Рис. 4.4. Параметри скрипта для ассета MQ-9

Після створення змінних в скрипті з’являється можливість присвоїти змінним наступні об’єкти ассета MQ-9.

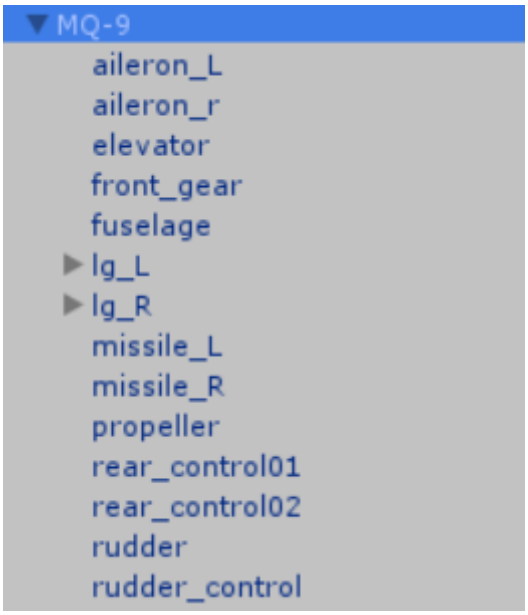

Рис. 4.5. Об’єкти ассета MQ-9

Рис. 4.6. Присвоєння об’єктів змінним

4.2. Контроль механізації крила та хвостового оперення за допомогою Unity.

В подальшому, за допомогою методу rotation стає можливим змінювати кут повороту окремих об’єктів, таких як елерони, елеватори та рулі висоти.

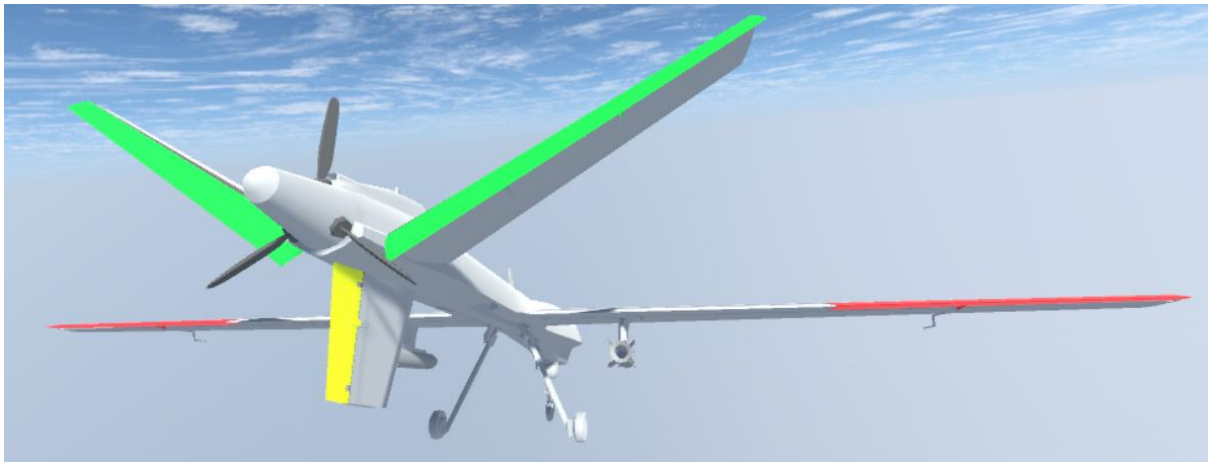

Рис.4.7. Тангаж < 0. Відхилення руля висоти вниз

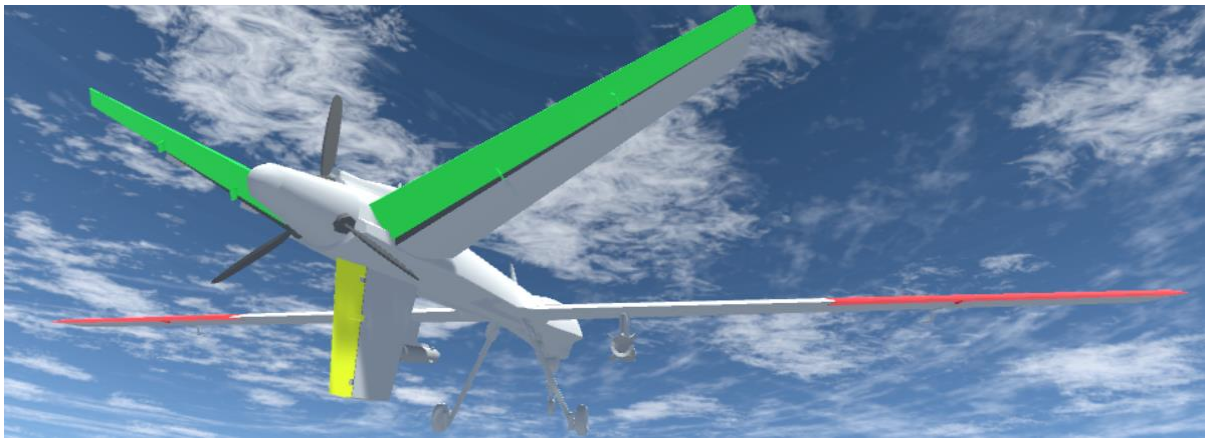

Рис.4.8. Тангаж > 0. Відхилення руля висоти вгору

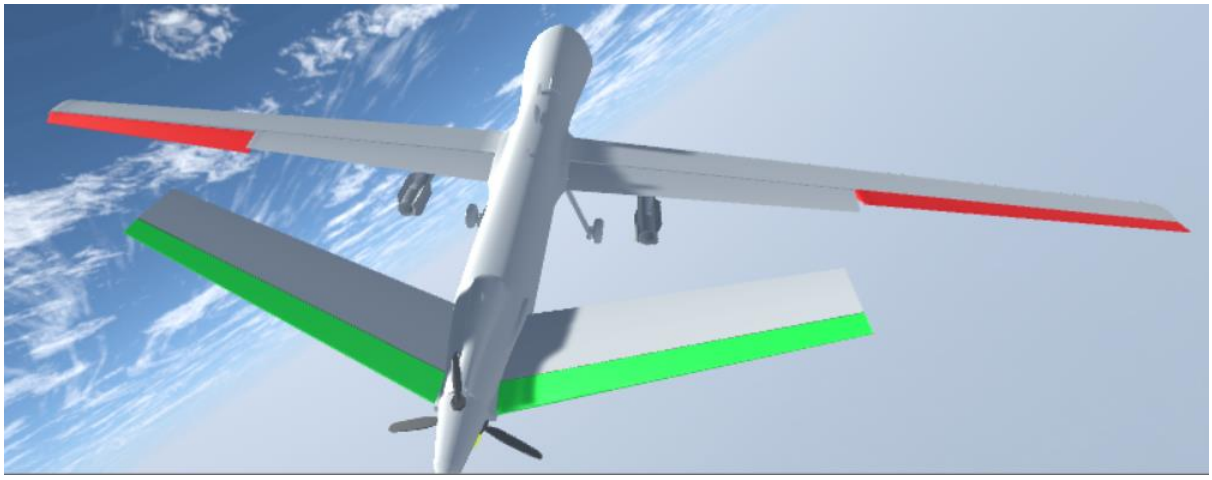

Рис.4.9. Кут крену вправо. Відхилення елеронів

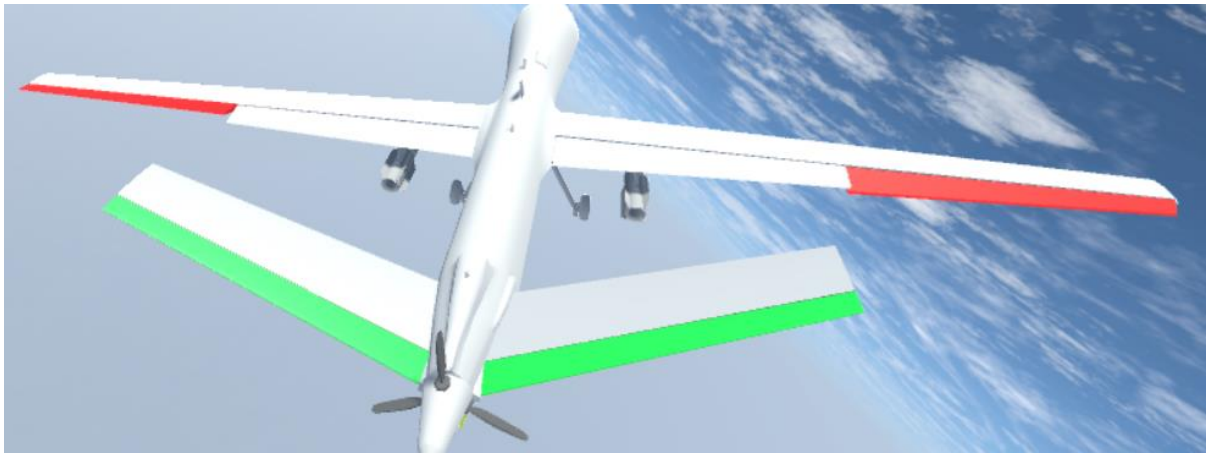

Рис. 4.10. Кут крену вліво. Відхилення елеронів

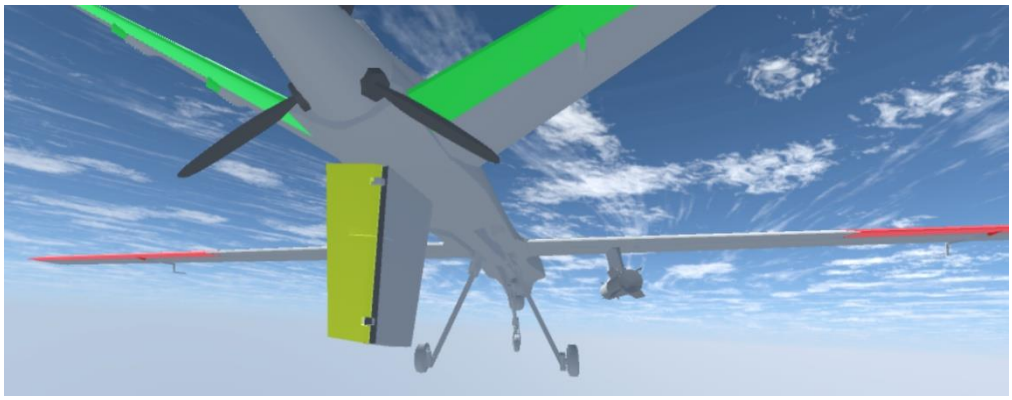

Рис.4.11. Кут рискання. Відхилення руля напрямку вліво.

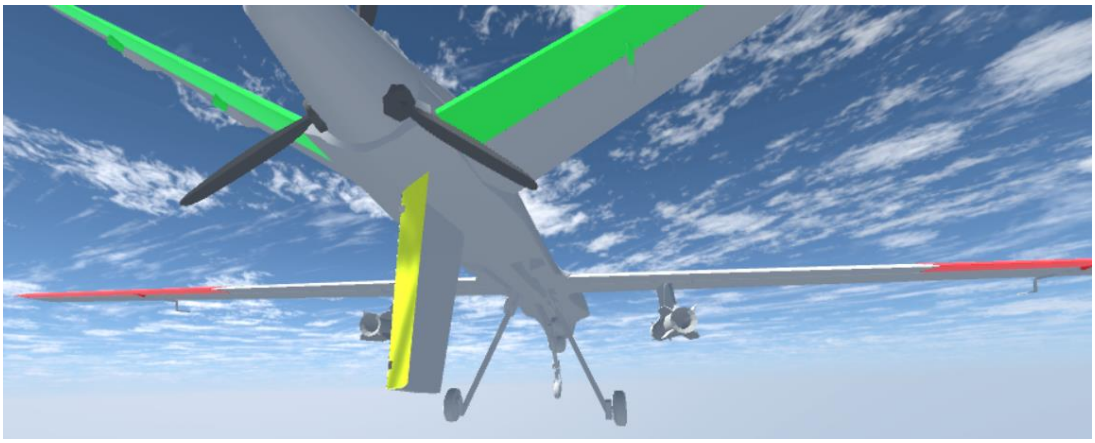

Рис.4.12. Кут рискання. Відхилення руля напрямку вправо.

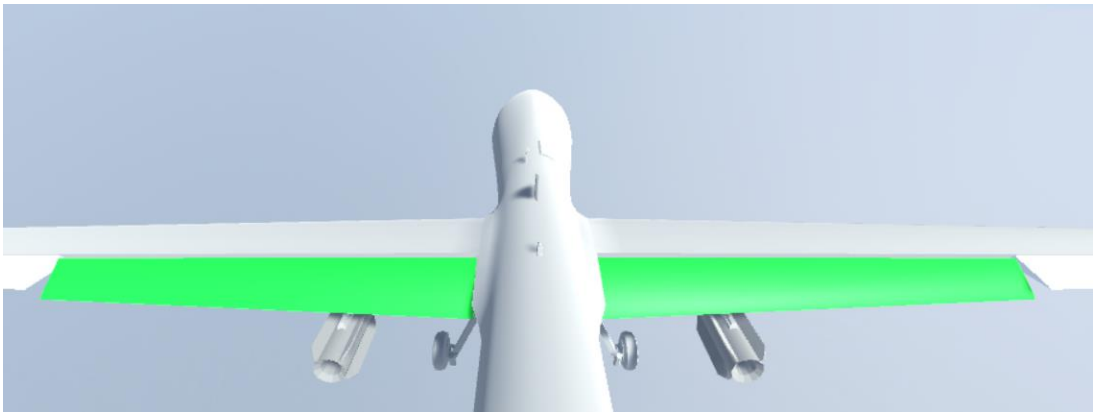

Рис 4.13. Випуск закрилків

4.3. Рух по колу. Програмування траєкторій польоту руху по колу. Зліт та посадка.

Використовуючи 3D модель куба створимо коло з декількох точок, по якому буде рухатися БПЛА. На рисунку зображено 4 точки кола та точка глісади. Також, використовуючи площину побудуємо злітно-посадкову смугу, яка буде стартовою позицією об’єкту дослідження.

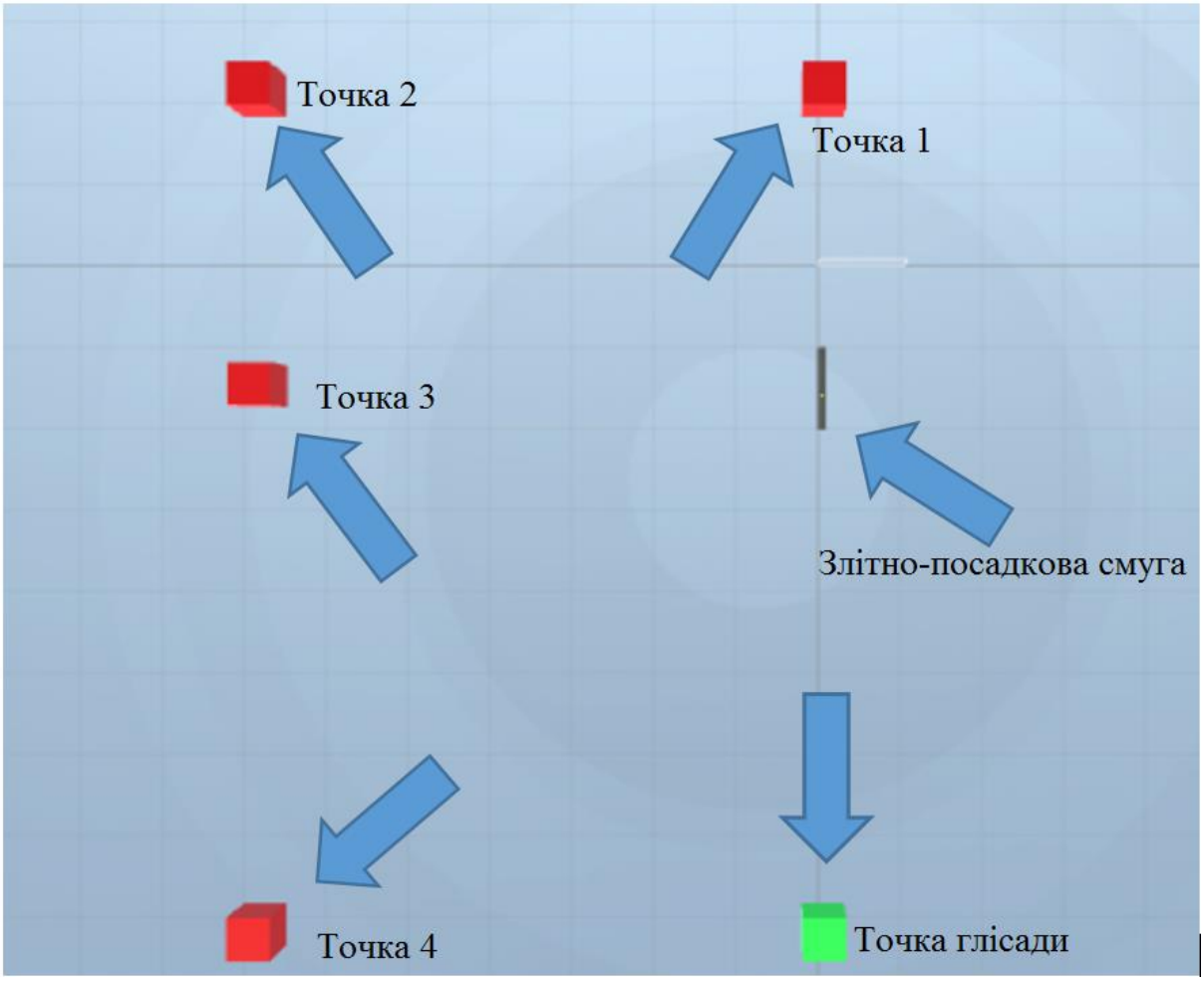

Рис. 4.14. План розташування цільових точок та ЗПС

Створивши усі необхідні точки та злітно-посадкову смугу необхідно розмістити об’єкт (БПЛА) на стартову позицію (ЗПС).

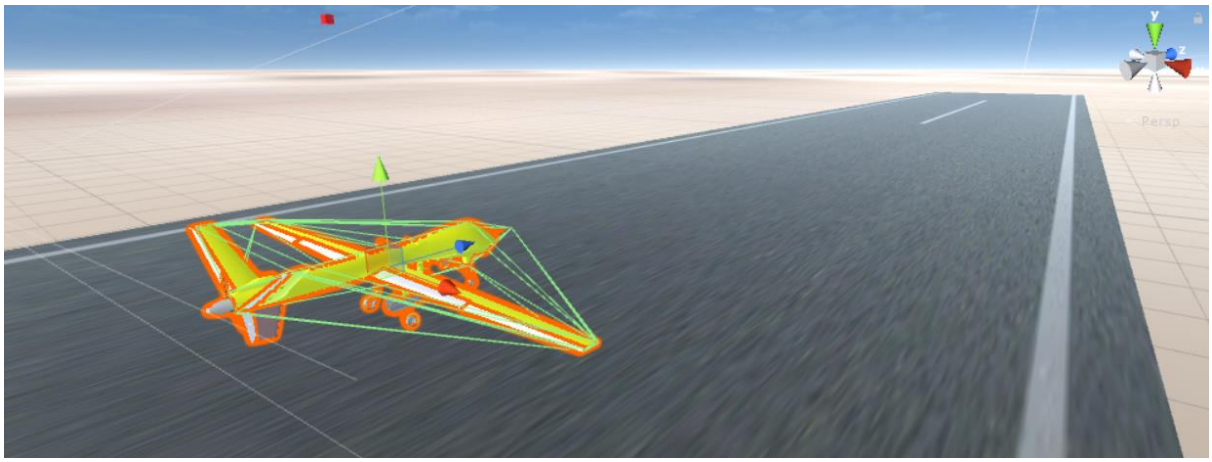

Рис. 4.15. Об’єкт дослідження на стартовій позиції

Сцена готова для використання. Після її запуску для керування БПЛА необхідно використовувати наступний маппінг клавіш:

Таблиця 4.1

Маппінг клавіш для керування БПЛА

↑	Збільшити тягу
↓	Зменшити тягу
←	Кермо напряму вліво
→	Кермо напряму вправо
A	Крен вліво
D	Крен вправо
S	Збільшити тангаж
W	Зменшити тангаж
N	Увімкнути/вимкнути автопілот
E	Випустити/прибрати закрилки

Збільшуючи тягу натисканням клавіши ↑ - збільшується швидкість БПЛА – тим самим дозволячи отримати достатню підіймальну силу для відриву від ЗПС. Отримавши достатню швидкість для відриву необхідно

збільшити тангаж клавішою **W** тим самим створивши підіймальну силу більшу ніж сила тяжіння.

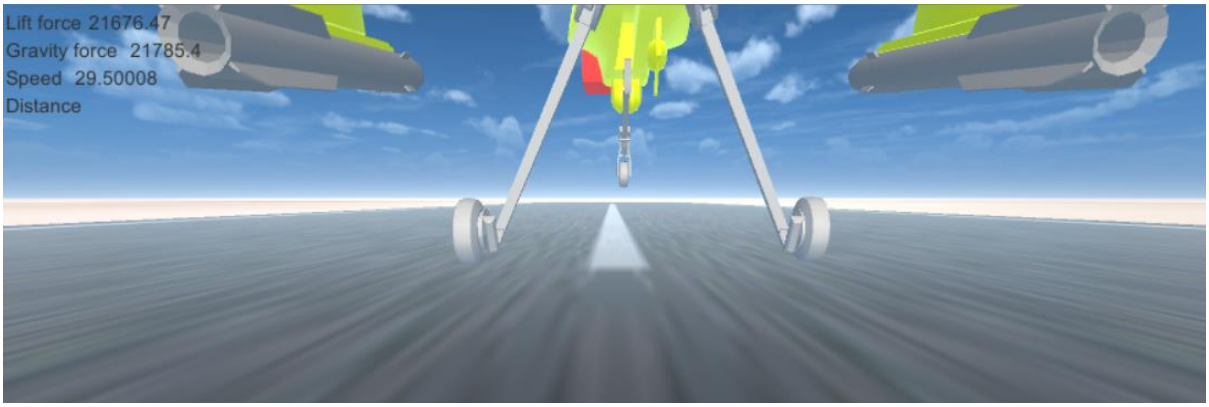

Рис. 4.16. Момент відриву об’єкту від ЗПС

Відрив безпілотного літального апарату від злітно-посадкової смуги відбувається за рахунок збільшення підіймальної сили. В Unity є можливість вирахувати цю силу та застосувати її до літального апарату. Відбувається це за допомогою методу AddForce, вхідним параметром для якого є тривимірний вектор Vector3. Розрахувати його можна за допомогою формул, які знаходяться у створенному методі ApplyForces():

```
    private void ApplyForces()
    {
 // Визначення швидкості БПЛА
var localVelocity = transform.InverseTransformDirection(mq9Rigidbody.velocity)
;
// Визначення кута атаки, що є арктангенсом осі y на вісь z.
   var angleOfAttack = Mathf.Atan2(-localVelocity.y, localVelocity.z);
//Визначення відносного подовження крила
   var aspectRatio = (wingSpan * wingSpan) / wingArea;
    // α * 2 * PI * (AR / AR + 2)
    var inducedLift = angleOfAttack * (aspectRatio / (aspectRatio + 2f)) * 2f * Mat
```

```
hf.PI;
    // CL ^ 2 / (AR * PI)
    var inducedDrag = (inducedLift * inducedLift) / (aspectRatio * Mathf.PI);
//Визначення тиску
      // V ^ 2 * R * 0.5 * A
      var pressure = mq9Rigidbody.velocity.sqrMagnitude * 1.2754f * 0.5f * wing
Area;
//Розрахунок підіймальної сили
      var lift = inducedLift * pressure;
      liftForceLbl.text = lift.ToString();
      gravityLbl.text = (mq9Rigidbody.mass*9.8).ToString();
//Розрахунок сили опору
var drag = (0.021f + inducedDrag) * pressure;
//Визначення направлення векторів сили
// *flip sign(s) if necessary*
      var dragDirection = mq9Rigidbody.velocity.normalized;
      var liftDirection = Vector3.Cross(dragDirection, transform.right);
  // Lift + Drag = Total Force
      mq9Rigidbody.AddForce(liftDirection * lift - dragDirection * drag);
      mq9Rigidbody.AddForce(transform.forward * EnginePower);
   }
```

Усі інші сили, які не розраховуються в данному методі – розраховуються автоматично середовищем Unity.

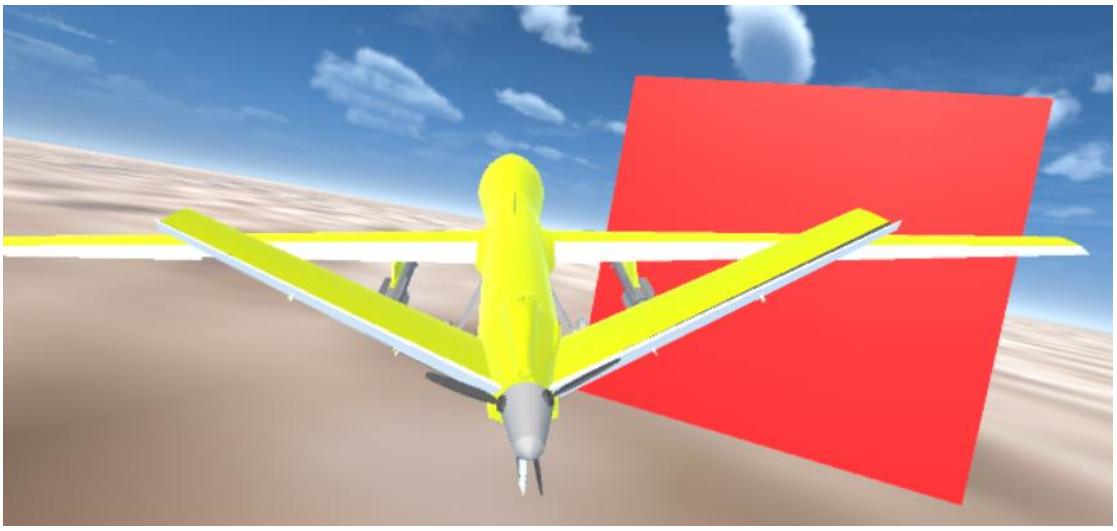

Рис. 4.17. Наближення об’єкту до першої точки, зміна курсу на точку 2

Далі розглянемо як відбувається пошук наступної контрольної точки та як вираховується відстань до неї.

```
private void SelectTarget()
{
    if (Target == null)
        Target = GetClosestTarget ();
```

//Розрахунок відстані від БПЛА до контрольної точки вимірюється за допомогою координат у тривимірному просторі. Так, маючи координати, можливо скористатися методом Distance класу Vector3 і вирахувати необхідну відстань.

```
    float distance = Vector3.Distance (mq9Rigidbody.transform.position, Target.transform.position);

    distanceLbl.text = distance.ToString ();
```

//Якщо відстань менша за 250 метрів – контрольна точка змінюється на наступну точку по колу.

```
        if (Target != Runway&&distance >= 250)
            return;

        var currIndex = CirclePoints.IndexOf (Target);
```

//Пройшовши останню точку по колу – наступною контрольною точкою вибирається точка глісади.

```
        if (currIndex == CirclePoints.Capacity - 1)
            Target = GlisadePoint;
        else
            Target = CirclePoints [currIndex+1];
    }
```

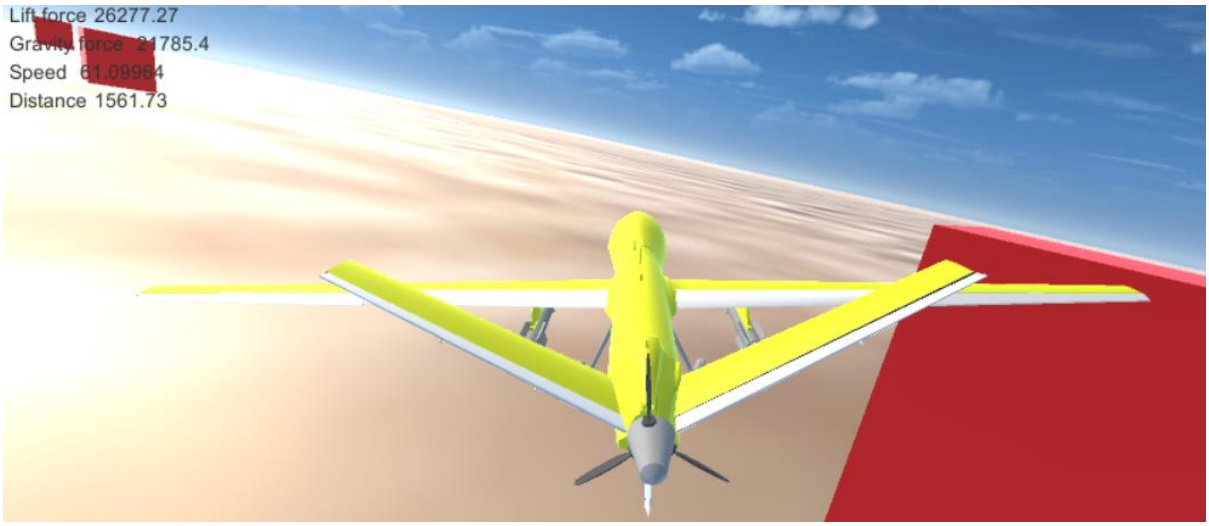

Рис. 4.18. Наближення об’єкту до другої точки, зміна курсу на точку 3

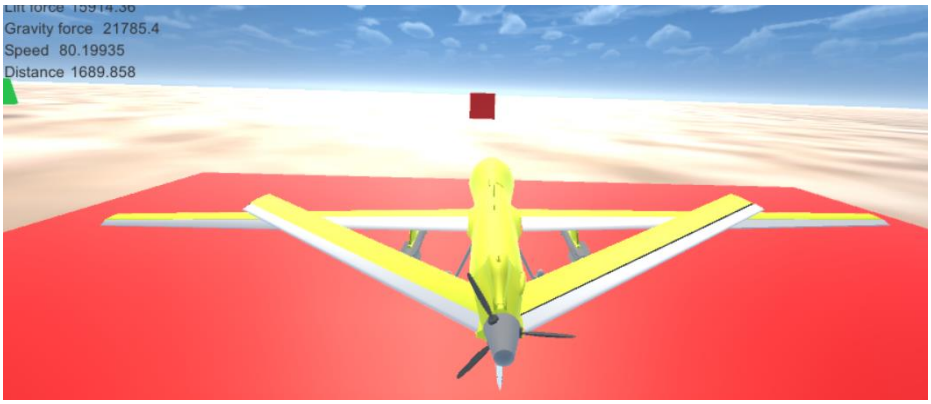

Рис. 4.19. Проходження третьої точки, встановлення курсу на точку 4

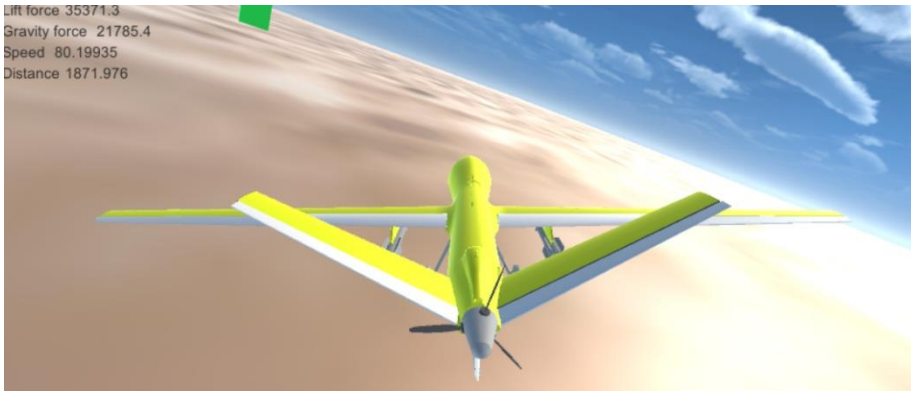

Рис. 4.20. Проходження четвертої точки, встановлення курсу на точку глісади

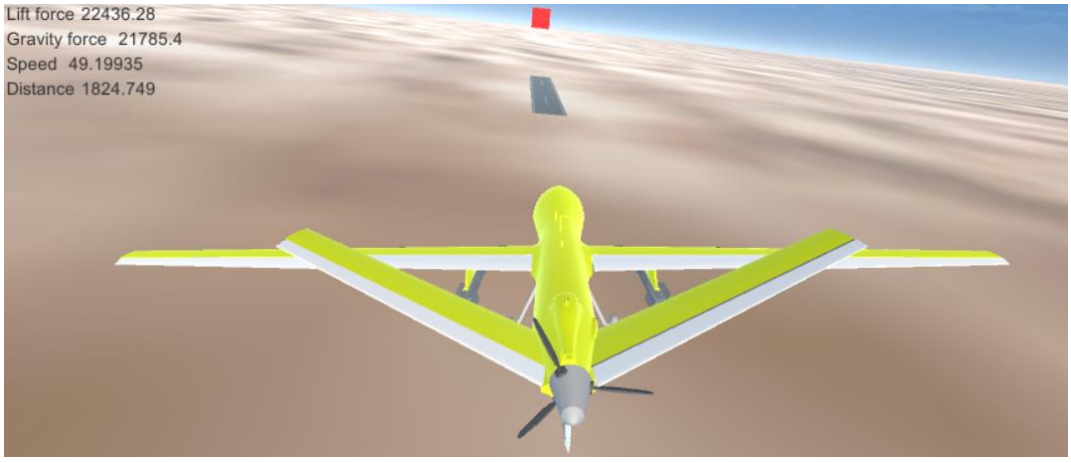

Рис. 4.21. Проходження точки глісади, встановлення курсу на ЗПС

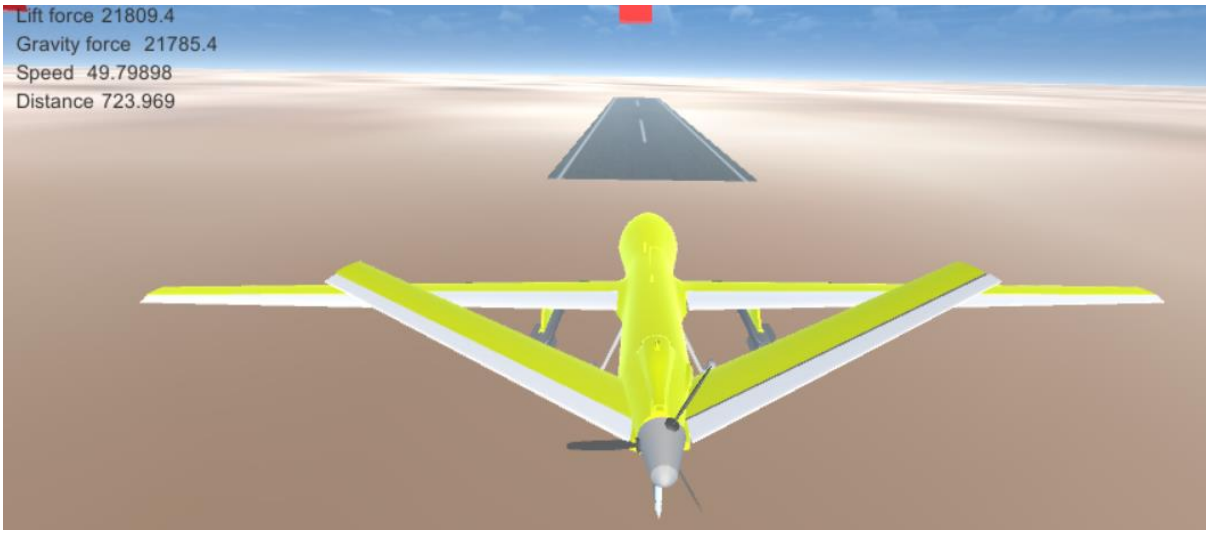

Рис. 4.22. Захід на посадку

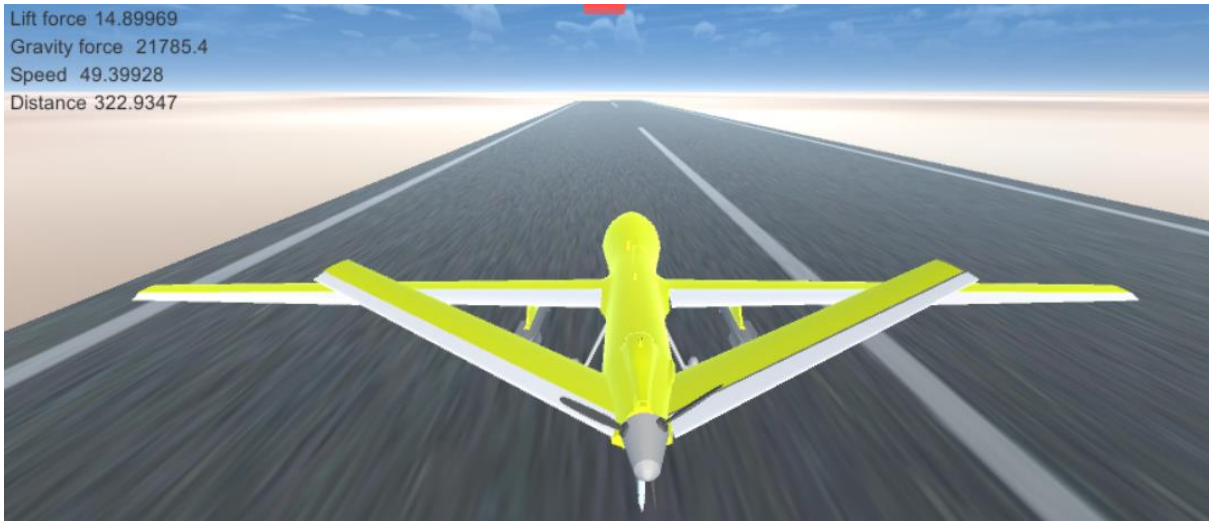

Рис. 4.23. Посадка на ЗПС

Також, врахована можливість виходити на точки кола, якщо літальний апарат стартує не з злітно-посадкової смуги або знаходиться поза межами кола. Для цього використовується метод GetClosestTarget():

```
        private GameObject GetClosestTarget()
    {
        //Початковою найближчою точкою вибирається перша точка кола
       GameObject possibleTarget = CirclePoints[0];
```

```
        float shortestDistance = Vector3.Distance (mq9Rigidbody.transform.position,
 CirclePoints[0].transform.position);

         //У наступному циклі йде пошук точки, яка є ближчою ніж попередня
        foreach (var target in CirclePoints)
        {
            float distanceToTarget = Vector3.Distance (mq9Rigidbody.transform.positi
on, target.transform.position);
            if (shortestDistance > distanceToTarget)
            {
                shortestDistance = distanceToTarget;
                possibleTarget = target;
            }
        }
        return possibleTarget;
    }
```

Висновок до розділу 4

У цьому розділі було представлено розробку та роботу програми по плануванню траєкторій при зльоті та посадці з траєкторією руху по колу.

ВИСНОВКИ

Під час виконання роботи безпілотний літальний апарат MQ-9 Reaper був розглянутий і прийнятий в якості об'єкта дослідження для вирішення проблеми планування траєкторії в Unity. Для цього були проаналізовані загальні характеристики об'єкта, які були введені в середовище розвитку, щоб створити реальні умови та знайти надзвичайно точні траєкторії для слідування цілі.

В результаті було розроблено додаток, що містить розроблене середовище та об’єкт дослідження. Ця програма може використовуватися для управління будь-яким об'єктом, його частинами та рухом, обчислення траєкторій для кращого слідування різних цілей, а також уникнення перешкод для збереження об'єкта в цілісності та безпеці.

Додаток можна використовувати в різних сферах, якщо необхідно змоделювати деякі умови та протестувати будь-який з літальних апаратів, оскільки він універсальний і простий у використанні. Використовуючи додаток немає необхідності заходити всередину коду та робити його редагування. Все, що потрібно, знаходиться поза ним і може бути легко встановлено.

Також описано можливості Unity у роботі зі створенням тренажерів, основні функції, які можуть допомогти у розробці подібного додатку, проілюстровані в розділах 3 та 4.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Trajectory optimization of multiple quad-rotor UAVs in collaborative assembling task. [Електронний ресурс]. Режим доступу: https://www.sciencedirect.com/science/article/pii/S100093611500237X (дата звернення 07.10.2020). - Назва з екрана.

2. Sayler, Kelley (June 2015). "A world of proliferated drones : a technology primer" . Center for a New American Security.

3. MQ-9 Reaper. [Електронний ресурс]. Режим доступу: http://www.af.mil/About-Us/Fact-Sheets/Display/Article/104470/mq-9-reaper/ (дата звернення 12.10.2020). - Назва з екрана.

4. MQ9 reaper predator b unmanned aircraft system uas data sheet specifications information. [Електронний ресурс]. Режим доступу: http://www.armyrecognition.com/us_american_unmanned_aerial_ground_vehicle_uk/mq9_reaper_predator_b_unmanned_aircraft_system_uas_data_sheet_specifications_information_pictures_u.html (дата звернення 10.10.2020). - Назва з екрана.

5. Моисеев В.С. Основы теории эффективного применения беспилотных летательных аппаратов: монограф / В.С. Моисеев. – Казань: Редакционно-издательский центр «Школа», 2015. – 444 с. (Сер. «Современная прикладная математика и информатика»).

6. Ефимов А.П. Акустика: Справ. / А.П. Ефимов, А.В. Никонов, М.А. Сапожков, В.И. Шоров. – М.: Радио и связь, 1989. – 336 с.

7. Иванов Н.И. Инженерная акустика. Теория и практика борьбы с шумом: учеб. / Н.И. Иванов. – М.: Университетская книга, Логос, 2008. – 424 с.

8. Советов Б.Я. Моделирование систем: учеб. пособие / Б.Я. Советов, С.А. Яковлев. – М.: Высш. шк., 1998. – 319 с.

9. Полёт по кругу. [Електронний ресурс]. Режим доступу: https://ru.wikipedia.org/wiki/Полёт_по_кругу (дата звернення 15.10.2020). - Назва з екрана.

10. [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Глісада

11. FlightGear, часть 3 - полеты по кругу - базовая часть. [Електронний ресурс]. Режим доступу: https://yacc11.livejournal.com/14088.html (дата звернення 10.10.2020). - Назва з екрана.

12. Учебное пособие по аэродинамике. Предкрылки. [Електронний ресурс]. Режим доступу: https://studfile.net/preview/2989748/page:17/ (дата звернення 16.10.2020). - Назва з екрана.

13. Аэродинамические характеристики самолета с механизацией крыла. [Електронний ресурс]. Режим доступу: https://studizba.com/lectures/129-inzhenerija/1825-ajeromehanika/35802-28-ajerodinamicheskie-harakteristiki-samoleta-s-mehanizaciej-kryla.html (дата звернення 16.11.2020). - Назва з екрана.

14. Vector2. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/ScriptReference/Vector2.html (дата звернення 02.11.2020). - Назва з екрана.

15. Vector3. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/ScriptReference/Vector3.html (дата звернення 02.11.2020). - Назва з екрана.

16. Rigidbody. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/ScriptReference/Rigidbody.html (дата звернення 02.11.2020). - Назва з екрана.

17. Rigidbody.rotation. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/ScriptReference/Rigidbody-rotation.html (дата звернення 02.11.2020). - Назва з екрана.

18. Rigidbody.AddForce. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/ScriptReference/Rigidbody.AddForce.html (дата звернення 02.11.2020). - Назва з екрана.

19. GameObject. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/ScriptReference/GameObject.html (дата звернення 02.11.2020). - Назва з екрана.

20. Quaternion and Euler rotations in Unity. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html (дата звернення 02.11.2020). - Назва з екрана.

21. Using the Inspector. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/UsingTheInspector.html (дата звернення 02.11.2020). - Назва з екрана.

22. Asset workflow. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/AssetWorkflow.html (дата звернення 02.11.2020). - Назва з екрана.

23. Creating scenes. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/CreatingScenes.html (дата звернення 02.11.2020). - Назва з екрана.

24. Colliders overwiev. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/CollidersOverview.html (дата звернення 02.11.2020). - Назва з екрана.

25. Camera. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/ScriptReference/Camera.html (дата звернення 02.11.2020). - Назва з екрана.

26. Prefabs. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/Prefabs.html (дата звернення 02.11.2020). - Назва з екрана.

Додаток А

Скрипт поведінки MQ9

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using UnityEngine;
using UnityEngine.UI;
public class MQ9Behaviour : MonoBehaviour
{
        private float rollAngle {get{ return m_Speed / 300; }}
        private float pitchAngle {get{ return m_Speed / 500; }}
        private float yawAngle {get{ return m_Speed / 300; }}
        private float maxSpeed=100;
        private float acceleration=0.1f;
    [Header("Mechanics")]
    public GameObject AileronLeft;
    public GameObject AileronRight;
    public GameObject Rudder;
    public GameObject Propeller;
    public GameObject Elevators;
    public GameObject Target;
        public GameObject Runway;
        public GameObject GlisadePoint;
        public List<GameObject> CirclePoints;
    public GameObject Rear1;
    public GameObject Rear2;
```

```
    [Header("Labels")]
public Text velocityLbl;
public Text angleAttackLbl;
public Text liftForceLbl;
    public Text gravityLbl;
    public Text distanceLbl;
[Header("PartsControl")]
public float _elevatorsSpeed = 3f;
public float _maxRearAngle = 40f;
    public float _maxElevatorAngle = 25f;
public float _rudderSpeed = 3f;
public float _maxRudderAngle = 40f;
public float _aileronSpeed = 3f;
public float _maxAileronAngle = 20f;
    public float _maxRollAngle = 45f;
float m_Speed;
private bool _isNavigate;
    private bool _useElevators;
private float _oldAngleTurn = 0f;
private float _newAngleTurn = 0f;
private bool horizontalFlight;
private Rigidbody mq9Rigidbody;
private Rigidbody rigidbodyPropeller;
[Header("Sensors")]
public float SensorLength = 200f;
public Vector3 FrontSensorPos = new Vector3(0, 4f, 0.25f);
public float SideSensorPos = 10f;
public float FrontSensorAngle = 20;
    public float wingSpan = 13.56f;
```

```
    public float wingArea = 78.04f;
    public float EnginePower=1000;
    private float aspectRatio;
void Start()
{
   //Set the speed of the GameObject
   m_Speed = 0f;
   _newAngleTurn = transform.eulerAngles.y;
   _oldAngleTurn = _newAngleTurn;
            mq9Rigidbody = GetComponent<Rigidbody>();
   rigidbodyPropeller = Propeller.GetComponent<Rigidbody>();
            mq9Rigidbody.centerOfMass = new Vector3 (0.13f, -0.03f, -2.63f);
}
void FixedUpdate()
    {
   Move();
            ApplyForces();
            if (Input.GetKey(KeyCode.N))
            {
                _isNavigate = !_isNavigate;
            }
            if (_isNavigate)
            {
                Navigate();
            }

            if (Input.GetKey(KeyCode.E))
            {
                _useElevators = !_useElevators;
```

```
        }
        UseElevators(_useElevators);

        velocityLbl.text = m_Speed.ToString ();
if (Input.GetKey(KeyCode.UpArrow))
{
                if (m_Speed < maxSpeed)
                        m_Speed += acceleration;
        }
if (Input.GetKey(KeyCode.DownArrow))
{
                if(m_Speed>0)
                m_Speed--;
}
if (Input.GetKey(KeyCode.W))
{
   PitchChange(1);
}
        else if (Input.GetKey (KeyCode.S)) {
                PitchChange (-1);
        } else {
                RearAling ();}
        #region roll
if (Input.GetKey(KeyCode.A))
{
   RollChange(1);
}
        else if (Input.GetKey (KeyCode.D)) {
                RollChange (-1);
```

```
            }
            else
            {
                AileronsAling ();
            }
            #endregion

            if (Input.GetKey (KeyCode.LeftArrow)) {
                YawChange (-1);
            } else if (Input.GetKey (KeyCode.RightArrow)) {
                YawChange (1);
            } else {
                RudderAling ();
            }
    }
        private GameObject GetClosestTarget()
        {
            GameObject possibleTarget = CirclePoints[0];
            float shortestDistance = Vector3.Distance
(mq9Rigidbody.transform.position, CirclePoints[0].transform.position);

            foreach (var target in CirclePoints)
            {
                float distanceToTarget = Vector3.Distance
(mq9Rigidbody.transform.position, target.transform.position);
                if (shortestDistance > distanceToTarget)
                {
                    shortestDistance = distanceToTarget;
                    possibleTarget = target;
```

```
                }
            }
            return possibleTarget;
        }

        private void SlowDownTo(float slowTo)
        {
            new Thread(() => {
                while(m_Speed>slowTo)
                {
                    m_Speed--;
                    Thread.Sleep(500);
                }
            }).Start();
        }
        private void SelectTarget()
        {
            if (Target == null)
                Target = GetClosestTarget ();
            float distance = Vector3.Distance (mq9Rigidbody.transform.position,
Target.transform.position);
            distanceLbl.text = distance.ToString ();
            if (Target != Runway&&distance >= 250)
                return;

            if (Target == GlisadePoint) {
                Target=Runway;
                return;
            }
```

```
        if (Target == Runway)
        {
                if(distance<=20)
                SlowDownTo (0);
                else if(distance<=50)
                        SlowDownTo (5);
                else if(distance<=100)
                        SlowDownTo (10);
                else if(distance<=200)
                        SlowDownTo (30);
                else if(distance<=5000)
                        SlowDownTo (50);
                else if(distance<=1000)
                        SlowDownTo (75);
                return;
        }
        var currIndex = CirclePoints.IndexOf (Target);

        if (currIndex == CirclePoints.Capacity - 1)
                Target = GlisadePoint;
        else
                Target = CirclePoints [currIndex+1];
}
private void Navigate()
{
        SelectTarget ();

        if (Target!=Runway&&transform.position.y < 300)
                PitchChange(-1);
```

```
            _newAngleTurn = Truncate(transform.eulerAngles.y);
            if (_oldAngleTurn < _newAngleTurn)
            {
                RollChange(-1);
            }
            else if (_oldAngleTurn > _newAngleTurn)
            {
                RollChange(1);
            }
            else
            {
                AileronsAling ();
            }
            var lookPos = Target.transform.position - transform.position;
            //lookPos.y = 0;
            var rotation = Quaternion.LookRotation(lookPos);//calculating the rotation at which it is necessary to look
            transform.rotation = Quaternion.Slerp(transform.rotation, rotation, Time.deltaTime/2);//rotate with dt
            _oldAngleTurn = _newAngleTurn;
        }

    private float Truncate(float number)
    {
        decimal bar = Convert.ToDecimal(number);
        return (float)Math.Round(bar, 1);
    }

    private void Move()
```

```
    {
            Vector3 velocityVector = new Vector3
(mq9Rigidbody.transform.forward.x * m_Speed, mq9Rigidbody.velocity.y,
mq9Rigidbody.transform.forward.z * m_Speed);
            mq9Rigidbody.velocity=velocityVector;
            rigidbodyPropeller.transform.Rotate(0, 0, 100 * m_Speed);
    }

        private void ApplyForces()
        {
            // *flip sign(s) if necessary*
            var localVelocity =
transform.InverseTransformDirection(mq9Rigidbody.velocity);
            var angleOfAttack = Mathf.Atan2(-localVelocity.y, localVelocity.z);
            var aspectRatio = (wingSpan * wingSpan) / wingArea;

            // α * 2 * PI * (AR / AR + 2)
            var inducedLift = angleOfAttack * (aspectRatio / (aspectRatio + 2f)) *
2f * Mathf.PI;

            // CL ^ 2 / (AR * PI)
            var inducedDrag = (inducedLift * inducedLift) / (aspectRatio *
Mathf.PI);

            // V ^ 2 * R * 0.5 * A
            var pressure = mq9Rigidbody.velocity.sqrMagnitude * 1.2754f * 0.5f
* wingArea;

            var lift = inducedLift * pressure;
```

```
            liftForceLbl.text = lift.ToString();
            gravityLbl.text = (mq9Rigidbody.mass*9.8).ToString();
            var drag = (0.021f + inducedDrag) * pressure;
            // *flip sign(s) if necessary*
            var dragDirection = mq9Rigidbody.velocity.normalized;
            var liftDirection = Vector3.Cross(dragDirection, transform.right);
            // Lift + Drag = Total Force
            mq9Rigidbody.AddForce(liftDirection * lift - dragDirection * drag);
            mq9Rigidbody.AddForce(transform.forward * EnginePower);
        }
        private void RollChange(int mult)
    {
            Vector3 vectorRotation = new Vector3(0, 0, mult*rollAngle);
            Vector3 aileronVectorRotation = new Vector3(mult*rollAngle, 0, 0);
       float aileronAngle =
TakeNormalAngle((float)Math.Round((double)AileronLeft.transform.localEulerA
ngles.x));
            if ((mult*rollAngle < 0 && aileronAngle > -_maxAileronAngle) ||
(mult*rollAngle > 0 && aileronAngle < _maxAileronAngle))
       {
          AileronLeft.transform.Rotate(-aileronVectorRotation * 1.2f);
          AileronRight.transform.Rotate(aileronVectorRotation * 1.2f);
       }
            var mq9Roll = TakeNormalAngle ((float)Math.Round
((double)transform.eulerAngles.z));
            if((mult==1&&mq9Roll<=_maxRollAngle)||(mult==-
1&&mq9Roll>=-_maxRollAngle))
                transform.Rotate(vectorRotation);
    }
```

```
        private void AileronsAling()
        {
                var
rightAileronAngle=TakeNormalAngle((float)Math.Round((double)AileronRight.tr
ansform.localEulerAngles.x));
                var
leftAileronAngle=TakeNormalAngle((float)Math.Round((double)AileronLeft.trans
form.localEulerAngles.x));
                if (rightAileronAngle == 0 && leftAileronAngle == 0)
                        return;
                int mult = 1;

                if (rightAileronAngle<0&&leftAileronAngle>0)
                {
                        mult = -1;
                }
                Vector3 aileronVectorRotation = new Vector3(mult*rollAngle, 0, 0);
                AileronRight.transform.Rotate(aileronVectorRotation * 1.2f);
                AileronLeft.transform.Rotate(-aileronVectorRotation * 1.2f);
        }
        private void YawChange(int mult)
   {
                Vector3 vectorRotation = new Vector3(0, mult*yawAngle, 0);
      float rudderAngle =
TakeNormalAngle((float)Math.Round((double)Rudder.transform.localEulerAngles
.y));
                if ((mult*yawAngle < 0 && rudderAngle >= -_maxRudderAngle) ||
(mult*yawAngle > 0 && rudderAngle <= _maxRudderAngle))
         Rudder.transform.Rotate(vectorRotation * 1.2f);
```

```
        transform.Rotate(vectorRotation);
    }
        private void RudderAling()
        {
                var
rudderAngle=TakeNormalAngle((float)Math.Round((double)Rudder.transform.loc
alEulerAngles.y));
                if (rudderAngle == 0)
                        return;
                int mult = 1;

                if (rudderAngle>0)
                {
                        mult = -1;
                }
                Vector3 rudderVectorRotation = new Vector3(0,mult*yawAngle, 0);
                Rudder.transform.Rotate(-rudderVectorRotation * 1.2f);
        }
        private void PitchChange(int mult)
    {
                float rearAngle =
TakeNormalAngle((float)Math.Round((double)Rear1.transform.localEulerAngles.
x));
        //float angleOfAttack =
TakeNormalAngle((float)Math.Round((double)transform.eulerAngles.x));
                Vector3 vectorRotation = new Vector3(mult*pitchAngle, 0, 0);
                //if ((angleOfAttack < -30 && mult*pitchAngle < 0) || (angleOfAttack
> 30 && mult*pitchAngle > 0)) return;
```

```
            if ((mult*pitchAngle < 0 && rearAngle <_maxRearAngle) ||
(mult*pitchAngle > 0 && rearAngle > -_maxRearAngle))
        {
          //Elevators.transform.Rotate(-vectorRotation * 2f);
          Rear1.transform.Rotate(vectorRotation * 2f);
          Rear2.transform.Rotate(-vectorRotation * 2f);
        }
            transform.Rotate(new Vector3(mult*pitchAngle, 0, 0));
    }
        private void RearAling()
        {
            var
rearAngle=TakeNormalAngle((float)Math.Round((double)Rear1.transform.localE
ulerAngles.x));
            if (rearAngle == 0)
                return;
            int mult = 1;

            if (rearAngle>0)
            {
                mult = -1;
            }
            Vector3 elevatorVectorRotation = new Vector3(mult*pitchAngle, 0,
0);
            //Elevators.transform.Rotate(-elevatorVectorRotation * 1.2f);
            Rear1.transform.Rotate(-elevatorVectorRotation * 1.2f);
            Rear2.transform.Rotate(elevatorVectorRotation * 1.2f);
        }
        private void UseElevators(bool use)
```

```
        {
            float elevatorsAngle = TakeNormalAngle ((float)Math.Round
((double)Elevators.transform.localEulerAngles.x));
            if (use)
            {
                wingArea = 1000f;
                Vector3 vectorRotation = new Vector3 (pitchAngle, 0, 0);

                if ((elevatorsAngle < _maxElevatorAngle) && (elevatorsAngle
> -_maxElevatorAngle))
                    Elevators.transform.Rotate (vectorRotation);
            }
            else
            {
                wingArea = 78.8f;
                if (elevatorsAngle == 0)
                    return;
                int mult = 1;

                if (elevatorsAngle>0)
                {
                    mult = -1;
                }
                Vector3 elevatorVectorRotation = new
Vector3(mult*pitchAngle, 0, 0);
                Elevators.transform.Rotate(-elevatorVectorRotation * 1.2f);
            }
        }
    private float TakeNormalAngle(float angle)
```

```
{
   if (angle > 180)
   {
      return angle - 360;
   }
   else
   {
      return angle;       }  }}
```