МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія самонавчання для посадки літального апарату»

Завідувач
випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Москаленко В. В.**

Студента групи ІН.м – 81н **Поскачея Д. В.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЕЛІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Поскачея Демида Віталійовича
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія самонавчання для посадки літального апарату*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми та постановка задачі 2) Інформаційно інтелектуальна технологія 3) Інформаційне та програмне забезпечення системи самонавчання*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми та постановка задачі*		
2.	*Інформаційно інтелектуальна технологія*		
3.	*Інформаційне та програмне забезпечення системи самонавчання*		
4.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 51 стор., 11 рис., 1 додаток, 11 джерел.

Об’єкт дослідження — процес керування посадкою літального апарату.

Мета роботи — підвищення ефективності керування посадкою літального апарату.

Методи дослідження — інформаційна технологія самонавчання та критерії оптимізації параметрів системи керування посадкою.

Результати — Розроблено модель, алгоритм та програмна реалізація інформаційної технології машинного навчання для ефективної посадки безпілотного літального апарату. Розроблений алгоритм реалізовано на мові python.

СИСТЕМА БЕЗПІЛОТНОЇ ПОСАДКИ ЛІТАЛЬНХ АПАРАТІВ
ЕКСПЕРИМЕНТАЛЬНЕ СЕРЕДОВИЩЕ, ІНФОРМАЦІЙНА
ІНТЕЛЕКТУАЛЬНА ТЕХНОЛОГІЯ.

ЗМІСТ

ВСТУП

Створення алгоритмів самонавчання на основі Reinforcement Learning (навчання з підкріпленням) є актуальним завданням у сучасному світі, оскільки цей метод працює без заздалегідь підготовлених відповідей у системі про яку нічого не відомо. Ця здатність цього методу відкриває можливості для вирішення нового типу проблеми. Зокрема, цей метод використовується для самонавчання систем складних об’єктів. Створення таких алгоритмів самонавчання має велике практичне та наукове значення, оскільки такі алгоритми можуть слугувати основою для подальших досліджень та вдосконалення для створення алгоритмів, що вирішують реальні проблеми.

Загальною тенденцією в галузі автоматичного керування в основному використовуються алгоритми теорії автоматичного управління з пропорційно інтегрально диференціальними регуляторами для стабілізації параметрів системи. В процесі виконання роботи було проведено огляд цих алгоритмів.

Застосування самонавчающихся алгоритмів здатне показати не меншу та навіть кращу швидкість та точність оптимізації параметрів які потрібно влаштувати для посадки літального засобу.

Головним завданням магістерської роботи є створення симульованого середовища та алгоритму який здатен на основі навчання з підкріпленням виконати посадку літального засобу на вказану площину швидко, точно та з найменшим використанням палива.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасний стан та тенденції розвитку інформаційних технологій керування літальними апаратами

На даний момент для посадки літальних апаратів використовують теорію автоматичного керування (ТАК), що за допомогою математичних засобів розкривають властивості автоматичних систем управління та розробляються рекомендації щодо їх проектування. Основна мета автоматизації – виключити безпосередню участь людини в управлінні виробничими процесами та іншими технічними об'єктами. Поєднання елементів управління та об'єкта утворює систему управління. Система, в якій лише частина операцій автоматизована, а інша частина керується людьми, називається автоматизованою (частково автоматичною). Особливий випадок управління – регулювання. При регулюванні координат процесу (тиск, температура, витрата, положення тощо) підтримують задану величину за допомогою спеціальних пристроїв – автоматичних контролерів. Поєднання регульованого об'єкта та автоматичного регулятора утворює систему автоматичного регулювання. Об'єкти регулювання та контролю за своєю фізичною природою дуже різноманітні, але принципи побудови систем управління та методи їх дослідження однакові [3].

Класична система автоматичного управління показана на наступному малюнку (Рис. 1.1):

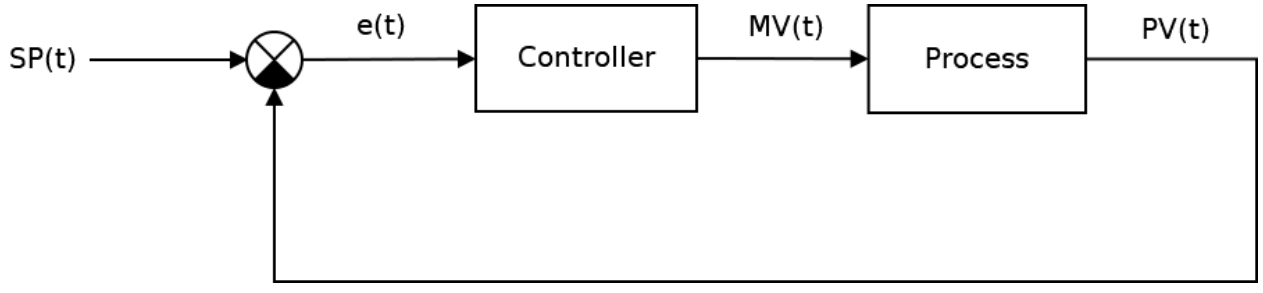

Рисунок 1.1 – Структурна схема системи автоматичного управління

Ключовим елементом будь-яких самохідних апаратів є регулятор, який представляє собою пристрій, який контролює стан об'єкта управління та забезпечує необхідний закон управління. Процес управління включає:

обчислення похибки управління або сигналу невідповідності e(t) як різниці між бажаною заданою точкою і поточним значенням процесу, після чого контролер генерує керуючі сигнали [3].

Одним із видів регуляторів є пропорційно-інтегрально-диференціюючий (ПІД) регулятор (Рис. 1.2), який генерує керуючий сигнал, який є сумою трьох термінів: пропорційного, інтегрального та диференціального.

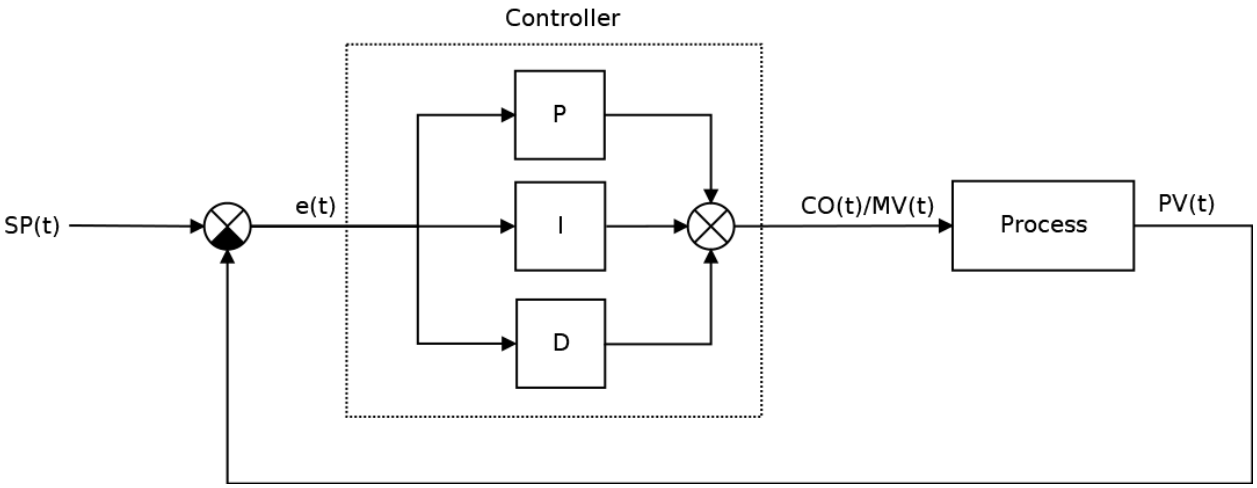

Рисунок 1.2 – Структурна схема системи керування з ПІД-регулятором

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L} \right) \qquad (1.1)$$

Де $e(t)$ помилка невідповідності, як і, $P = K_P \cdot e(t)$ (1.2) – пропорційна, $I = K_i \cdot \int_0^t e(\tau)\, d\tau$ (1.3) – є інтегральною, $D = K_d \cdot \frac{de(t)}{dt}$ (1.4) – є диференційована складова (умови) закону управління, що в остаточному вигляді описується наступними формулами.

$$MV(t) = \underbrace{K_p \cdot e(t)}_{P} + \underbrace{K_i \cdot \int_0^t e(\tau)d\tau}_{I} + \underbrace{K_d \cdot \frac{de(t)}{dt}}_{D}, \qquad (1.5)$$

$$e(t) = SP(t) - PV(t) \qquad (1.6)$$

$$MV(t) = \underbrace{K_p \cdot e(t)}_{P} + \underbrace{K_i \cdot \int_0^t e(\tau)d\tau}_{I} + \underbrace{K_d \cdot \frac{de(t)}{dt}}_{D} \qquad (1.7)$$

Пропорційна складова P – відповідає за так званий пропорційний контроль, сенс якого полягає в тому, що вихідний сигнал регулятора протидіє відхиленню керованої змінної (невідповідність помилок також називається залишком) від заданого значення. Чим більша помилка невідповідності, тим більше відхилення команди від контролера. Це найпростіший і очевидний закон управління. Недоліком закону пропорційного керування є те, що регулятор ніколи не стабілізується при заданому значенні, а збільшення коефіцієнта пропорційності завжди призводить до самоколивань. Ось чому, крім закону про пропорційний контроль, треба використовувати цілісні та диференційні. Інтегральна складова I акумулює (інтегрує) помилку управління, що дозволяє PID-контролеру усунути статичну помилку (стаціонарна помилка, залишкова невідповідність). Або іншими словами: інтегральне посилання завжди вносить певні упередження, і якщо система зазнає якихось постійних помилок, то вона їх компенсує (через свою упередженість). Але якщо цих помилок немає або вони знехтують невеликими, то ефект буде протилежним – цілісна складова сама введе помилку зміщення. Саме з цієї причини він не використовується, наприклад, у проблемах надточного позиціонування. Основним недоліком закону інтегрального управління є ефект відключення інтегратора.

Диференціальна складова D пропорційна швидкості зміни відхилення керованої змінної і призначена для протидії відхиленням від цільового значення, що прогнозуються в майбутньому. Важливо, що диференціальний компонент усуває затухаючі коливання. Диференціальний контроль особливо ефективний для процесів, які мають великі затримки. Недоліком закону диференціального управління є його нестійкість до впливу шуму (шум диференціації).

Таким чином, залежно від ситуації, P-, PD-, PI- та PID-контролери можуть використовуватися, але основний закон управління є переважно пропорційним (хоча в деяких конкретних завданнях можуть використовуватися лише зв'язки диференціатора та інтегратора).

Але тенденції розвитку направлені на самонавчальних алгоритмів які можуть на основі даних більш чітко врахувати різні взаємозв’язки і дозволяють більш оптимально реагувати на зміни стану навколишньої середовища. Та без допомоги людини знаходять оптимальні рішення які згодом можуть бути покращені спеціальним вузько спрямованим алгоритмом.

Алгоритми машинного навчання – це програми, які можуть навчатися з даних і покращувати результат на основі досвіду без втручання людини. Завдання навчання можуть включати в себе вивчення функції, яка відображає вхідні дані у вихідні дані, вивчення прихованої структури в немаркованих даних; або «навчання на основі примірників», де мітка класу створюється для нового примірника шляхом порівняння нового екземпляра (рядки) з примірниками з навчальних даних, які були збережені в пам'яті. «Навчання на основі примірників» не створює абстракцію від конкретних випадків [8].

Контрольоване навчання використовує помічені дані навчання для вивчення функції відображення, яка перетворює вхідні змінні (X) в вихідну змінну (Y). Іншими словами, він вирішує для f в наступному рівнянні:

$$Y = f(X) \quad (1.8)$$

Це дозволяє нам точно генерувати результати, коли їм дають нові входи.

Класифікація використовується для прогнозування результатів даної вибірки, коли вихідна змінна має форму категорій. Класифікаційна модель може дивитися на вхідні дані і намагатися передбачити мітки «правильний» або «не правильний».

Регресія використовується для прогнозування результату даної вибірки, коли вихідна змінна має форму реальних значень. Наприклад, регресійна модель може обробляти вхідні дані для прогнозування кількості опадів, зростання людини і т. п.

Ансамбль є ще одним типом контрольованого навчання. Це означає об'єднання прогнозів декількох моделей машинного навчання, які окремо є слабкими, для отримання більш точного прогнозу на новій вибірці.

Моделі навчання без вчителя використовуються, коли у нас є тільки вхідні змінні (X) і немає відповідних вихідних змінних. Вони використовують не помічені навчальні дані для моделювання базової структури даних.

Асоціація використовується для виявлення ймовірності одночасної появи предметів в рядку. Він широко використовується в аналізі кошика. Наприклад, модель асоціації може бути використана для виявлення того, що якщо покупець купує хліб, він з 80% ймовірністю також придбає яйця.

Кластеризація використовується для групування вибірок таким чином, що об'єкти в одному кластері більше схожі один на одного, ніж об'єкти з іншого кластера.

Зменшення розмірності використовується для зменшення кількості змінних в наборі даних, забезпечуючи при цьому передачу важливої інформації. Зменшення розмірності може бути зроблено з використанням методів вилучення елементів і методів вибору елементів. Вибір об'єктів виділяє підмножина вихідних змінних. Функція Extraction виконує перетворення даних з багатовимірного простору в низькорозмірний простір.

Навчання з підкріпленням – це тип алгоритму машинного навчання, який дозволяє агенту вибирати кращу наступну дію на основі його поточного стану, вивчаючи поведінку, яке максимізує винагороду [8].

Алгоритми з підкріпленням зазвичай знаходять оптимальні дії методом проб і помилок. Уявіть собі, наприклад, відеогру, в якій гравцеві необхідно переміщатися в певні місця в певний час, щоб заробити очки. Алгоритм підкріплення, який грає в цю гру, буде починатися з випадкового руху, але з часом методом проб і помилок він дізнається, де і коли йому потрібно перемістити персонажа в грі, щоб максимізувати загальну кількість очок.

1.2 Моделі та методи інтелектуального керування складними об'єктами

Методи вирішення задач лінійного програмування за одним критерієм інтенсивно розроблялися протягом декількох десятиліть. Але незабаром, з розвитком інформатики та технологій, до цього дня можна з упевненістю

сказати, що будь-яка серйозна проблема характеризується більш ніж одним критерієм. Можна зрозуміти, що завдання багатокритеріальної оптимізації виникають у тих випадках, коли існує кілька цілей, які не можуть бути відображені одним критерієм (наприклад, вартість, надійність тощо). Ті, хто приймає рішення, значно більшою мірою, ніж будь-коли, відчувають необхідність оцінювати альтернативні рішення за кількома критеріями [1,2].

Результати дослідження проблем планування та контролю показують, що в реальній постановці ці завдання є багатокритеріальними. Отже, вираз "часто досягати максимального ефекту за найменших витрат" вже означає прийняття рішення за двома критеріями. Оцінка діяльності підприємств та планування як системи прийняття рішень базується на більш ніж десятках критеріїв: виконання плану виробництва за обсягом, номенклатури, плану реалізації, прибутку за показниками рентабельності, продуктивності праці тощо [1,2].

Раніше при дослідженні багатокритеріальної проблеми часто всі критерії, крім одного, обраного домінуючим, приймалися за обмеження, оптимізація проводилася за домінуючим критерієм. Такий підхід до вирішення практичних проблем значно знижує ефективність рішень. Тут потрібно знайти точку в області можливих рішень, яка мінімізує або максимально використовує всі ці критерії. У зв'язку з цим дослідники почали розробляти наявні теоретичні та практичні результати методів розв’язання задач за одним критерієм, щоб вони були застосовні для вивчення багатокритеріальних задач лінійного програмування [1,2].

З огляду на це, в теорії багатокритеріальної оптимізації поняття оптимальності отримує різні інтерпретації, і тому сама теорія містить три основні напрямки:

1. Розробка концепції оптимальності.
2. Доведення існування рішення, яке є оптимальним у відповідному значенні.
3. Розробка методів пошуку оптимального рішення.

Ми позначимо і-й конкретний критерій через, а область можливих рішень – Q. Ми враховуємо, що, змінюючи знак функції, завжди можна звести задачу мінімізації до задачі максимізації, і навпаки, ми можемо сформулювати коротко проблему оптимізації вектора наступним чином:

$$Z(x) = \begin{pmatrix} z_1(x) \\ z_2(x) \\ z_m(x) \end{pmatrix} \rightarrow max, \text{при } x\ Q(1,5) \qquad (1.9)$$

Якщо подивитися так, то по суті багатокритеріальна задача відрізняється від звичайної задачі оптимізації лише наявністю декількох об'єктивних функцій замість однієї. Але на відміну від завдань оптимізації з одним критерієм, в багатокритеріальній оптимізації існує невизначеність цілей. Дійсно, існування рішення, яке максимізує кілька об'єктивних функцій, є рідкісним винятком, тому з математичної точки зору проблеми оптимізації багатокритерій є невизначеними, і рішенням може бути лише компромісне рішення [4].

Наприклад, вибираючи роботу, людина, як правило, керується кількома критеріями. Припустимо, хтось хоче, щоб ці умови були дотримані одночасно:

- зарплата була максимально високою;
- умови праці були максимально комфортними;
- місце роботи було якомога ближче до дому.

Іншим прикладом завдання з багатьма критеріями є модернізація виробництва, в процесі якої хочеться досягти максимального підвищення ефективності за найменших витрат. Очевидно, що неможливо досягти обох цілей одночасно, оскільки чим вище витрати, тим більше має бути продукції і тим більше прибутку.

Іншим прикладом є вибір інвестиційного рішення, коли ви хочете отримати максимальний дохід (або рентабельність) з найменшим ризиком.

Для того, щоб отримати більш повний опис переваг та недоліків об'єкта, що проектується, слід врахувати більше критеріїв якості. В результаті проблеми проектування складних систем завжди є багатокритеріальними, оскільки багато різних вимог до системи доводиться враховувати при виборі

найкращого варіанту. Зі звичайної точки зору, проблема з багатьма критеріями не має рішення, але, на щастя, це не так, завжди існує можливість одночасного задоволення всіх заданих умов. І так як практично будь-яка така ситуація дозволяє різні компромісні рішення, підходи до їх пошуку численні та дуже різноманітні [4].

Ось кілька підходів до вирішення задач оптимізації багатьох критеріїв:

1. Метод поступок – особа, яка приймає рішення, зводиться до вибору рішення шляхом поступового послаблення початкових вимог, як правило, одночасно невиконаних.
2. Ідеальний точковий метод – у діапазоні допустимих значень невідомого шукається така множина, яка здатна забезпечити набір значень критеріїв, у тому чи іншому сенсі, найближчих до найкращого варіанту.
3. Метод згортання – особа, яка приймає рішення, зводить задачу багатокритеріальної до задачі з одним критерієм.

Метод послідовних поступок на вирішення багатокритеріальної задачі застосовується у тому випадку, коли окремі критерії можуть бути впорядковані у порядку зменшення важливості. Припустимо, що всі критерії максимізовані та пронумеровані у порядку зменшення. Спочатку визначається максимальне значення Z_1, перший критерій в області можливих рішень, вирішивши проблему $Z_1(x) \to \max$, при $x[\,]Q$. Тоді, виходячи з практичних міркувань та прийнятої точності, визначається значення допуску (економічно обґрунтованого призначення) критерію і визначається максимальне значення другого критерію за умови, що значення першого має відхилятися від максимального не більше ніж значення допустимої поступки, тобто вирішення проблеми [1,2,4].

$$Z_2(x) \to max \qquad (1.10)$$

$$Z_1(x) \geq Z_1 - \delta_1 \qquad (1.11)$$

$$\text{при } x[\,]Q \qquad (1.12)$$

Знову ж таки присвоюється значення поступки $\delta_2 > 0$, а другим критерієм, яке разом з першим використовується при виявленні умовного екстремуму третього приватного критерію і т. д. Нарешті, крайнє значення останнього виявляється у важливості критерію Z_2, за умови, що значення кожного з критеріїв першої $m-1$ частинки відрізняється від екстремального не більше ніж значення допустимої поступки. Отриманий на останній стадії рішення, вважається оптимальним [4].

Метод основних критеріїв передбачає зведення проблеми багатокритеріальної оптимізації до однокритеріальної оптимізації. Для цього ми вибираємо один із розглянутих критеріїв як основний критерій, а решту перетворюємо на обмеження.

В якості основного критерію в різних галузях часто вибирають:

1. Вартість продажу
2. Обсяг виробництва
3. Продуктивність
4. Інтенсивність ресурсів (споживання палива, енергоємність, ...)
5. Терміни роботи тощо.

Проблему оптимізації в цьому випадку можна сформулювати так. Спочатку потрібно встановити головний критерій. Після цього вводиться система орієнтирів для решти критеріїв. Для кожного критерію слід вказати, що він не повинен перевищувати або бути меншим деякого заданого значення. Після цього вирішується задача однокритеріальної умовної оптимізації, де вона прагне до максимуму або мінімуму, за умови, що кожен конкретний критерій, крім основного критерію, буде більшим або рівним заданому контрольному значенню,

$$\begin{cases} f_1(x) \to max, x[\,]X \\ f_i \geq, \text{при} = 2, \dots, n \end{cases} \quad (1.13)$$

При чому всі значення X повинні належати до розглянутих допустимих множин. Найчастіше цей метод застосовується в інженерній практиці.

До переваг методу головного критерію належать простота інтерпретації результатів, відсутність високих вимог до математичної підготовки фахівців, програмного забезпечення та засобів обчислювальної техніки. Після того як ми зводимо завдання багатокритеріальної оптимізації до проблеми однокритеріальної оптимізації, ми отримуємо можливість використовувати стандартні програмні засоби, зокрема такі, як пошук рішення в Microsoft Excel.

Основним недоліком способу є:

1) надмірне спрощення структури завдань;
2) не завжди можна виділити яскраво виражений головний критерій, іноді це важко або неможливо зробити;
3) можливість втрати ефекту від комбінованого впливу декількох вторинних критеріїв;
4) обмеження щодо решти критеріїв повинні бути виправданими, а не прийматись якось;
5) можливість отримання неефективних рішень;
6) навіть якщо є критерій, який набагато важливіший за будь-який інший, то це не факт, що, в підсумку, решта критеріїв будуть не дуже значущими. Це може бути особливо вираженим у проблемах, де n великий.

На практиці критерії дуже часто мають різні масштаби та шкали вимірювання, і тоді виникає необхідність нормалізувати ці критерії.

Велике різноманіття проблем в робототехніці може бути натурально вирішене за допомогою навчання с підкріпленням. Навчання з підкріпленням (НП) дозволяє роботу самостійно взаємодіяти з навколишнім середовищем і знаходити оптимальні рішення методом проб і помилок. Замість того, щоб детально описувати вирішення проблеми. В посиленій формі навчання конструктор контрольного завдання забезпечує зворотній зв'язок у вигляді скалярної цільової функції, яка вимірює крок за кроком ефективність дій робота. Розглянемо, наприклад, спробу робота повернути м'яч для настільного тенісу через сітку. У цьому випадку робот може зробити спостереження за

динамічними перехідними, визначальними положеннями і швидкістю шара і внутрішнім динамічним суглобом положення і швидкість. Це насправді може добре захватити стан системи – надання повних статистичних даних для прогнозування майбутніх спостережень. Дії доступні для робота можуть бути крутним моментом, подавати сигнали на двигунах, або прискорення, що надсилаються до системи управління зворотною динамікою. Функція π, яка генерує моторні команди (тобто дії), засновані на м'ячі і нинішній стан внутрішньої руки і буде називатись політикою. Задача НП знаходиться в тому, щоб знайти політику, яка оптимізує довгострокову суму нагород R (s, a); НП що застосовується, є таким, щоб знайти оптимальну політику. Функція нагороди на приклад може бути заснована на успішному попаданні, а також вторинні критерії, такі як потреба енергії [7,8].

1.3 Формалізована постановка задачі

За результатами проведеного аналітичного огляду однією з важливих практичних проблем, що виникають в ході розробки систем керування літальними апаратами, є задача ефективного керування їх посадкою. Таким чином, метою роботи є розробка технології керування літальним апаратом саме на завершальному етапі польоту з застосуванням методів машинного навчання.

До параметрів оптимізації належать: швидкість приземлення, положення та місце посадки. Потрібно мінімізувати швидкість при посадці, мінімізувати використання двигунів, звісно потрібно приземлитися потрібною стороною та мінімізувати відхилення від центру посадкової площі.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) Сформувати вхідний математичний опис.

2) Розробити математичну модель процесу.

3) Розробити алгоритм керування посадкою на базі машинного навчання

4) Сформувати критерій ефективності машинного навчання

5) Програмно реалізувати запропонований алгоритм

6) Перевірити працездатність розробленої системи в режимі навчання і екзамену

2. ІНФОРМАЦІЙНО ЕКСТРЕМАЛЬНА ІНТЕЛЕКТУАЛЬНА ТЕХНОЛОГІЯ

2.1 Модель системи керування посадкою літального апарату

Навчання з підкріпленням (НП) є галуззю машинного навчання, яка займається навчанням на основі взаємодії з навколишнім середовищем, де зворотний зв'язок може бути відсутній. Хоча НП є дуже потужним інструментом, який успішно застосовується для вирішення різних завдань – від оптимізації хімічних реакцій до навчання комп'ютера грі у відеоігри, з історичної точки зору було важко почати роботу через відсутність цікавих і складних задач та середовища в якому можна експериментувати.

Саме тут потрібен OpenAI Gym. OpenAI Gym – це пакет Python, що включає в себе набір середовищ НП, від простих «іграшкових» середовищ до більш складних середовищ, в тому числі симульованих робототехнічних середовищ і середовищ відеоігор Atari. Він був розроблений з метою стати стандартизованої середовищем і еталоном для досліджень в галузі природничих наук [6].

Всі середовища НП мають простір станів (тобто набір всіх можливих станів середовища, в якій ви можете перебувати) і простір дій (тобто набір всіх дій, які ви можете виконувати в середовищі). Ви можете побачити розмір цих просторів (Рис. 2.1), використовуючи:

```
> print('State space: ', env.observation_space)
State space: Box(2,)

> print('Action space: ', env.action_space)
Action space: Discrete(3)
```

Рисунок 2.1 – Ілюстрація коду для задавання габаритів простору пошуку оптимальних параметрів

Це говорить нам про те, що простір станів є двовимірний прямокутник, тому кожне спостереження за станом являє собою вектор з 2 (плаваючих)

значень, і що простір дій включає в себе три дискретних дії. За замовчуванням ці три дії представлені цілими числами 0, 1 і 2. Однак ми не знаємо, які значення можуть приймати елементи вектора стану. Це можна знайти за допомогою (Рис. 2.2):

```
> print(env.observation_space.low)
[-1.2  -0.07]

>print(env.observation_space.high)
[0.6  0.07]
```

Рисунок 2.2 – Ілюстрація коду для виведення вектору стану системи

Звідси видно, що перший елемент вектора стану (що представляє позицію) може приймати будь-яке значення в діапазоні від -1,2 до 0,6, тоді як другий елемент (що представляє швидкість) може приймати будь-яке значення в діапазон від -0,07 до 0,07. Алгоритм Q-навчання гарантовано сходиться за умови, що кожна пара стан-дія відвідується досить велику кількість разів. У цій ситуації, однак, ми маємо справу з безперервним простором станів, що означає, що існує нескінченно багато пар стан-дія, що робить неможливим виконання цієї умови. Одним із способів вирішення цієї проблеми є використання глибоких Q-мереж (DQN). DQN об'єднують глибоке навчання з Q-навчанням, використовуючи глибоку нейронну мережу в якості аппроксіматора для Q-функції. DQN були успішно застосовані для розробки штучного інтелекту, здатного грати у відеоігри Atari. Альтернативний підхід – просто дискретизувати простір станів. Один простий спосіб зробити це – округлити перший елемент вектора стану з точністю до 0,1, а другий елемент з точністю до 0,01, а потім (для зручності) помножити перший елемент на 10, а другий на 100. Це зменшує кількість пар «стан-дія» до 855, що тепер дозволяє виконати умову, необхідну для сходження Q-навчання.

При використанні цього алгоритму ми припускали одномірний простір станів, тому нашою метою було знайти оптимальну Q-таблицю Q(s, a). У цьому завданні, оскільки ми маємо справу з двовимірним простором станів, ми

замінюємо Q (s, a) на Q (s1, s2, a), але в іншому алгоритм Q-навчання залишається більш-менш незмінним. Нагадаємо, що алгоритм виглядає наступним чином:

1. Ініціалізує Q (s1, s2, a), встановивши всі елементи рівними невеликим випадковим значенням;
2. Дотримуйтесь поточного стану, (s1, s2);
3. Грунтуючись на стратегії розвідки, виберіть дію, яку треба зробити *a*;
4. Прийміть заходи *a* і дотримуйтесь отриманої нагороди *r* і новий стан навколишнього середовища (s1 ', s2');
5. Оновлення Q (s1, s2, a) на основі правила поновлення: Q '(s1, s2, a) = (1 - w) * Q (s1, s2, a) + w * (r + d * Q (s1', s2 ', argmax a' Q (s1 ', s2'), a '))). Де w – швидкість навчання, a d – значення дисконту;
6. Повторіть кроки 2-5 до сходження.

2.2 Алгоритм машинного навчання для керування посадкою літального апарату

Градієнтні методи – це широкий клас алгоритмів оптимізації, що використовуються не лише в машинному навчанні. Його головна особливість в тому, що на одній ітерації ми віднімаємо не вектор градієнта, обчислений по всій вибірці, а робимо наступне. Ми випадково вибираємо один об'єкт з навчальної вибірки, і далі обчислюємо градієнт функціоналу тільки на цьому об'єкті, тобто градієнт тільки одного додатка в функціоналі помилки, і віднімаємо саме цей градієнт з поточного наближення вектора ваг [5].

Дуже показово подивитися на графік збіжності для градієнтного спуску і стохастичного градієнтного спуску. У градієнтному спуску ми намагаємося на кожну ітерацію зменшити помилку на всій вибірці, і тому графік виходить гладким. У міру збільшення числа ітерацій помилка зменшується монотонно, оскільки ми зменшуємо її на всій вибірці. У випадку ж із стохастичним градієнтним спуском, ми зменшуємо на кожну ітерацію помилку тільки на одному об'єкті, але при цьому ми можемо збільшити її на іншому об'єкті, тому графік виходить пилкоподібний (рис. 2.1).

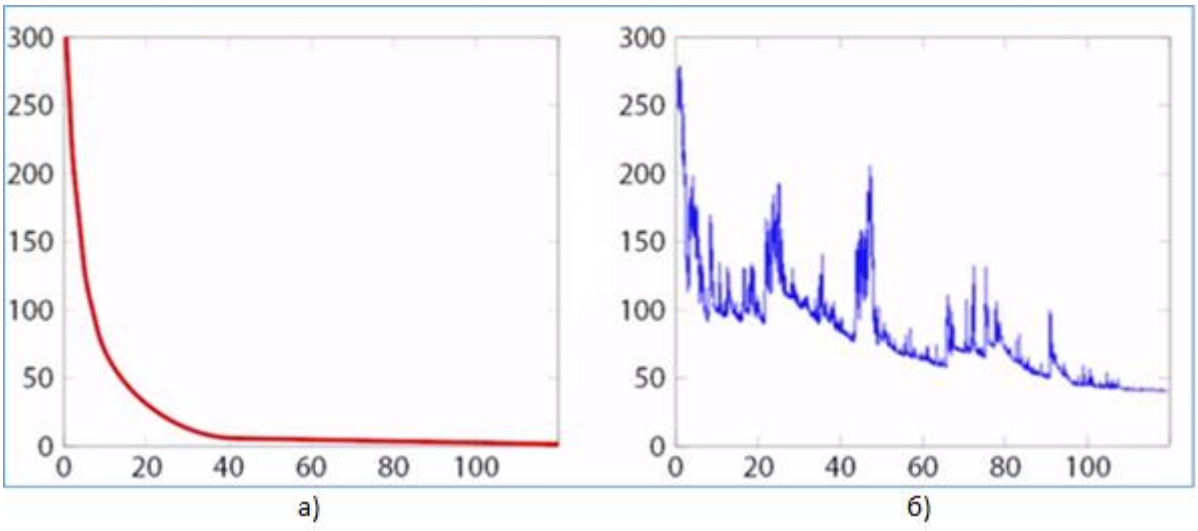

Рисунок 2.3 – Графіки зміни функції втрат під час навчання : а -- за навчальною вибіркою; б -- за тестовою вибіркою.

Ми на якийсь ітерації можемо збільшувати помилку, але при цьому в цілому він зменшується. І рано чи пізно ми виходимо на досить непогану якість, та на досить низьку помилку.

Тут градієнтний підхід буде розглянуто як спосіб вибору синаптичного вектора ваги ω у лінійному класифікаторі.

Ми починаємо з деякої ініціалізації вектора ваг, наприклад, нулями або іншими значеннями, і далі повторюємо в циклі градієнтні кроки. Способи ініціалізації ваг:

- ініціалізуйте вектор ω нулями. Цей метод використовується в багатьох системах, але не завжди є найкращим.
- $\omega_j := rand\left(-\frac{1}{n}, \frac{1}{n}\right)$ (2.1), де n – розмірність простору функцій. Цей підхід є значно більш успішним, ніж попередній, якщо характеристичний опис належним чином нормалізується.
- Інший підхід полягає у вирішенні початкової задачі оптимізації у випадку статистично незалежних ознак, лінійної функції активації (φ) та квадратичної функції втрат (L). Тоді рішення має вигляд: $\omega_j := \frac{<y,f_j>}{<f_j,f_j>}$ (2.2)

Градієнтний крок полягає в тому, що ми віднімаємо з поточного наближення вектора ваг $\omega(t-1)$ вектор градієнта, з деяким коефіцієнтом ηt. Повторюючи ці кроки до тих пір, поки не настане збіжність, тобто поки вектор ваг не почне змінюватися досить слабо.

Нехай y^*: X → Y – цільова залежність, відома лише від об'єктів навчальної вибірки:

$$X^l = (x_i, y_i)_{i=1}^l, y_i = y^*(x_i) \qquad (2.3)$$

Знайдемо алгоритм $a(x, w)$, який наближає залежність y^*. У випадку лінійного класифікатора потрібний алгоритм має вигляд:

$$a(x, w) = \varphi(\sum_{j=1}^n \omega_j x^j - \omega_0) \qquad (2.4)$$

Де $\varphi(z)$ грає роль функції активації у найпростішому випадку ми можемо поставити $\varphi(z) = sign(z)$).

Відповідно до принципу мінімізації емпіричного ризику, для цього достатньо вирішити оптимізаційну задачу: $Q(\omega) = \sum_{j=1}^n L(a(x_i, \omega), y_i) \rightarrow \min(\omega)$, де $L(a, y)$ (2.5) – задана функція втрат.

Для мінімізації застосовуйте метод градієнтного спуску. Це покроковий алгоритм, при кожній ітерації якого вектор ω змінюється в напрямку найбільшого зниження функціонального Q (тобто в напрямку антиградієнта): $\omega := \omega - \eta \nabla Q(\omega)$ – де η – позитивний параметр, який називається швидкістю навчання. Існує 2 основних підходи до здійснення градієнтного спуску:

1. Batch (партія), коли на кожній ітерації навчальний зразок розглядається як ціле, і лише після цього ω змінюється. Це потребує великих обчислювальних здібностей.
2. Стохастичний (стохастичний / онлайн), коли при кожній ітерації алгоритму з навчального набору якимось випадковим чином вибирається лише один об’єкт. Таким чином, вектор ω налаштовується на кожен щойно обраний об'єкт.

Стохастичний алгоритм градієнта (SG).

Вхід:

- X^l – навчальний набір
- η – темп навчання
- λ – параметр згладжування функції Q

Вихід:

- Вектор ваги ω.

Тіло:

1. Ініціалізувати ваги ω_j $j = 0, \cdots, n$;
2. Ініціалізувати поточну оцінку: функціоналу:
   $Q := \sum_{j=1}^{n} L\left(a(x_i, \omega), y_i\right)$ (2.6)
3. Повторювати:
   a. Виберіть об'єкт x_i з X^l (наприклад, випадковим чином);
      Обчисліть вихідне значення алгоритму $a(x_i, \omega)$ та помилку: $L(a(x_i, \omega), y_i)$ (2.7)
   b. Зробіть спуск по градієнту:
      $\omega := \omega - \eta L_a^l(a(x_i, \omega), y_i)\varphi'(< \omega, x_i >)x_i$ (2.8);
   c. Оцініть значення функціоналу:$Q := (1 - \lambda)Q + \lambda\varepsilon_i$ (2.9)
4. Доки значення Q не стабілізується та / або ваги ω не перестануть змінюватися.

Вище було сказано, що у випадку стохастичного градієнтного спуску предмети слід вибирати випадковим чином. Однак є евристики, спрямовані на поліпшення конвергенції, які дещо змінюють звичайний випадковий вибір.

Перемішування – пропонується випадковим чином вибирати об'єкти, але по черзі з різних класів. Ідея полягає в тому, що об'єкти з різних класів, швидше за все, менш "схожі", ніж об'єкти з одного класу, тому вектор ω щоразу змінюватиметься більше. Варіант алгоритму можливий, коли вибір кожного об'єкта неоднаковий, а ймовірність випадання об'єкта обернено пропорційна величині помилки на об'єкті. Слід зазначити, що при такій евристиці метод стає дуже чутливим до шуму.

Параметр згладжування в алгоритмі для оцінки функціонального Q при кожній ітерації його приблизне значення використовується методом експоненціального згладжування, звідки краще взяти λ порядку $\frac{1}{l}$. Якщо довжина зразка надмірно довга, то λ слід збільшувати.

Метод SG (з відповідним вибором функцій активації та втрати) являє собою узагальнення наступних широко розповсюджених евристичних методів вибору ω та класифікаційних алгоритмів:

1. Адаптивний лінійний елемент (Adalines);
2. Правило Хабба;
3. Алгоритм K-середніх (K-Means);
4. Learning Vector Quantization (LVQ).

Отже, у стохастичного градієнтного спуску є багато переваг. По-перше, в ньому набагато швидше обчислюється один крок, один градієнтний крок. Так само йому не потрібно зберігання всієї навчальної вибірки в пам'яті. Ми можемо зчитувати по одному об'єкту з вибірки і для кожного наступного об'єкта робити градієнтний крок. За рахунок цього стохастичний градієнтний спуск дозволяє навчати лінійні моделі на дуже великих вибірках, які не поміщуються в пам'ять комп'ютера. Так само він підходить для онлайн навчання коли навчальні об’єкти надходять у потік, і необхідно швидко оновлювати вектор ω, тобто ситуації, в якій ми отримуємо за кожен крок тільки один об'єкт, і повинні якось змінити модель, щоб врахувати цей об'єкт. Отже, ми обговорили, що градієнтний спуск вимагає підсумовування по всіх об'єктах навчальної вибірки на кожній ітерації, що може бути проблемою, якщо вибірка велика і не поміщається в пам’яті комп’ютера. Стохастичний градієнтний спуск вирішує цю проблему, використовуючи лише один об'єкт навчальної вибірки на кожній своїй ітерації. Алгоритм здатний вчитися на надмірно великих зразках завдяки тому, що випадкове підсистемування може бути достатньо для навчання. Можливі різні стратегії навчання. Якщо зразок надмірно великий, або навчання відбувається динамічно, то допустимо не зберігати навчальні

об’єкти. Якщо зразок невеликий, то ви можете повторно представити ті самі предмети для тренування.

Недоліки SG. Алгоритм може не занадто повільно збігатися або конвергуватися.

Як правило, функціональний Q є багатоекстремальним, і процес спуску градієнта може «застрягнути» в одному з локальних мінімумів. Для боротьби з цим використовується техніка коефіцієнтів струшування. Він полягає у внесенні випадкових модифікацій вектора ω у кожному досить стійкому сусідстві поточного значення та запуску градієнтного спуску з нових точок.

При великому вимірі простору n та / або невеликої довжини вибірки l перекваліфікація можлива, тобто класифікація стає нестабільною, а ймовірність помилок зростає. При цьому норма вектора ваг сильно зростає. Для боротьби з цим недоліком використовується розпад ваг. Він полягає у обмеженні можливого зростання норми ω шляхом додавання штрафу до $Q(\omega)$: $Q_\tau(\omega) = Q(\omega) + \frac{\tau}{2}\,||\omega||^2$. (2.10) В результаті правило для оновлення шкал набуває вигляду: $\omega := \omega(1 - \eta\tau) - \eta\nabla Q(\omega)$ (2.11)

Якщо функція активації має горизонтальні асимптоти, то процес може впасти в стан «паралічу». При великих значеннях скалярного добутку $< \omega, x_i >$ значення φ' стає близьким до нуля і вектор ω перестає суттєво змінюватися. Тому поширеною практикою є попередня нормалізація ознак: $x^j := \frac{x^j - x^j_{min}}{x^j_{max} - x^j_{min}}, j = 1, \cdots, n$ де x^j_{max}, x^j_{min} (2.12) – мінімальні та максимальні відхилення j-го атрибута відповідно. Якщо одночасно $\omega_j \epsilon \left[-\frac{1}{n}, \frac{1}{n}\right]$, то $< \omega, x > \epsilon[-1,1]$ (2.13)

Регуляризація (наприклад, розпад ваг) також є способом запобігання паралічу.

Збіжність в загальному випадку не гарантується, проте встановлено, що в разі опуклої функції *Q (w)* і при виконанні наступних 3-х умов:

1. $\eta_t^{t\to\infty} \to 0$; (2.14)

2. $\sum_{t=1}^{\infty} \eta_t = \infty$ (2.15)

3. $\sum_{t=1}^{\infty} \eta_t^2 < \infty$ (2.16)

Процес градієнтного спуску буде сходитися. Наприклад, можна покласти: $\eta_t = \frac{\eta_0}{t}$ (2.17). Однак, як показує практика, це не дуже вдалий спосіб.

Адам, метод ефективної стохастичною оптимізації, який вимагає тільки градієнти першого порядку з невеликим об'ємом пам'яті. Метод розраховує індивідуальні адаптивні показники навчання для різних параметрів з оцінок першого і другого моментів градієнту. Адам розшифровується як адаптивна оцінка моменту. Метод покликаний об'єднати переваги двох популярних останнім часом методів: AdaGrad, який добре працює з розрідженими градієнтами, і RMSProp, який добре працює в он-лайн і нестаціонарних настройках. Деякі з переваг Адама полягають в тому, що величини оновлення не залежать від масштабування градієнта, його розмір кроку приблизно обмежений гіперпараметром кроку, він не вимагає стаціонарної мети, він працює з розсіяними градієнтами і виконує ступеневу нормалізацію [9].

Нехай $f(\theta)$ – цільова функція з шумом: стохастична скалярная функція, яка є диференційованою за допомогою параметра θ. Ми зацікавлені в мінімізації очікуваного значення цієї функції $\mathbb{E}[f(\theta)]$ його параметри θ. З $f_1(\theta), \ldots, f_T(\theta)$ позначимо реалізації стохастичною функції на наступних кроках $1, \ldots, T$. стохастичною може виходити з оцінки в випадкових підвибірках (міні-пакетах) точок даних, або виникають через шум власної функції. При $g_t = \nabla_\theta f_t(\theta)$ ми позначаємо градієнт, тобто вектор приватних похідних f, θ, оцінений на кроці t [9].

Алгоритм оновлює експоненціальні середні градієнта (m_t) і квадрата градієнта (v_t) де гіперпараметри $\beta_1, \beta_2 \in [0, 1)$ керують експоненційною швидкістю згасання цих рухів середні. Самі середні є оцінками 1-го моменту (середнього) і 2-го сирого момент (нецентрована дисперсія) градієнта. Проте, ці середні ініціалізуються як (вектори) 0, що призводить до моментних оцінок, які зміщені до нуля, особливо протягом початку, і особливо, коли швидкості

загасання малі (тобто βs близькі до 1). Доброю новиною є те, що цього зміщення ініціалізації можна легко протидіяти, що призводить до виправлення зміщення оцінки m_t і v_t [9].

Зверніть увагу, що ефективність алгоритму 1 може бути поліпшена за рахунок зміни порядку обчислення, наприклад, замінивши останні три рядки в циклі такими рядками $a_t = a \cdot \sqrt{1 - \beta_2^t}/(1 - \beta_1^t)$ та $\theta_t \leftarrow \theta_{t-1} - a_t \cdot m_t/(\sqrt{v_t + \epsilon})$

Адам використовує умови виправлення зміщення ініціалізації. Ми будемо тут виводити термін для оцінки другого моменту; висновок для оцінки перших кроків повністю аналогічно. Нехай g – градієнт стохастичною мети f, і ми хочемо оцінити його другий необроблений момент (нецентрована дисперсія) з використанням експоненціального змінного середнього квадрата градієнта, зі швидкістю розпаду $\beta 2$. Нехай $g1, \ldots, gT$ – градієнти на наступних кроках, кожен з яких взято з градієнтного розподілу $gt \sim p\,(gt)$. Давайте форматувати експонентну середню як $v0 = 0$ (вектор нулів). Спочатку зверніть увагу, що оновлення на часовому кроці t експоненціальної середньої змінної $vt = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ – де g_t^2 вказує на елементарний квадрат $g_t \odot g_t$ можна записати у вигляді функції градієнтів на всіх попередніх етапах:

$$v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2 \qquad (2.17)$$

Ми хочемо знати, як $\mathbb{E}[v_t]$, очікуване значення експоненційної середньої на часовому кроці t, відноситься до істинного другого моменту $\mathbb{E}[g_t^2]$, Так що ми можемо виправити невідповідність між ними. Беручи очікування лівої і правої частини рівняння:

$$\mathbb{E}[v_t] = \mathbb{E}\left[(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\cdot g_i^2\right]$$
$$= \mathbb{E}[g_t^2]\cdot(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i} + \zeta$$
$$= \mathbb{E}[g_t^2]\cdot(1-\beta_2^t) + \zeta \quad (2.18)$$

де $\zeta = 0$, якщо істинний другий момент $\mathbb{E}[g_i^2,]$ Є стаціонарним; в іншому випадку ζ може бути невеликим, так як швидкість експоненціального затухання β1 можна (і потрібно) вибирати так, щоб експоненціальна змінна середня присвоювала малі ваги градієнтам занадто далеко в минулому. Те, що залишилося, це термін $(1-\beta_2^t)$ Котрий викликан ініціалізацією змінного середнього з нулями. Тому в алгоритмі ми ділимо на термін для виправлення зміщення ініціалізації. У разі рідкісних градієнтів для достовірної оцінки другого моменту необхідно усереднити по багато градієнтів, вибираючи невелике значення $\beta 2$; проте саме цей випадок малого $\beta 2$, де відсутність корекції зміщення ініціалізації призведе до набагато більших початковим крокам.

Проаналізуємо конвергенцію Адама за допомогою онлайн-бази навчання, запропонованої в Зінкевич, 2003. Дано довільну, невідому послідовність функцій випуклих витрат $f1(\theta), f2(\theta) \ldots, fT(\theta)$. В кожного разу t, наша мета – передбачити параметр θt та оцінити його за раніше невідомою вартістю функції f_t. Оскільки природа послідовності заздалегідь невідома, ми оцінюємо наш алгоритм використовуючи похибку – суму всієї попередньої різниці між онлайн-прогнозуванням $f_t\ (\theta_t)$ і найкращий параметр фіксованої точки $f_t(\theta^t)$ з можливої безлічі X для всіх попередніх етапів. Конкретно, похибку визначають як:

$$R(T) = \sum_{t=1}^{T}[f_t(\theta_t) - f_t(\theta^*)] \quad (2.19)$$

де $(\theta^*) = arg\ \min(\theta \epsilon x \sum_{t=1}^{T} f_t(\theta))$. Ми показуємо, що Адам має $O\sqrt{T}$ похибку [9].

2.3 Критерії оцінки ефективності навчання

Ціль навчання машин є мінімізація емпіричного ризику, що полягає в мінімізації функції втрат для кожного вхідного зразка. Функція витрат – це метод оцінки того, наскільки добре конкретний алгоритм моделює дані. Якщо прогнози занадто сильно відхиляються від фактичних результатів, функція втрат поверне дуже велике число. Поступово, за допомогою деякої функції оптимізації, функція втрат вчиться зменшувати помилки в прогнозуванні.

В алгоритмах машинного навчання не існує універсальної функції втрат на всі випадки життя. Існують різні фактори, пов'язані з вибором функції втрат для конкретного завдання, такі як тип обраного алгоритму машинного навчання, простота обчислення похідних і в деякій мірі відсоток сторонніх в наборі даних.

В цілому функції втрат можна класифікувати на дві основні категорії в залежності від типу завдання навчання, з якою ми маємо справу – регресивні втрати і класифікаційні втрати. При класифікації ми намагаємося передбачити вихідні дані з набору кінцевих категоріальних значень, тобто, з огляду на великий набір даних зображень рукописних цифр, класифікуючи їх в одну з 0-9 цифр. Регресія, з іншого боку, має справу з прогнозуванням безперервного значення, наприклад, з урахуванням площі, кількості кімнат, розміру кімнат, прогнозування ціни кімнати.

Ми можемо прагнути до максимізації або мінімізації цільової функції, тобто ми шукаємо рішення, яке має найвищий або найнижчий показник. Як таку, цільову функцію часто називають функцією витрат або функцією збитку, а значення, обчислене функцією витрат, називають просто "витрати". Функція, яку ми хочемо мінімізувати або максимізувати, називається об'єктивною функцією або критерієм. Коли ми мінімізуємо його, ми можемо також назвати його функцією витрат, функцією втрати або функцією помилок. Функція витрат або збитків має важливе завдання, оскільки вона повинна сумлінно переділити всі аспекти моделі в єдине число таким чином, щоб поліпшення цього числа були ознакою кращої моделі. Функція витрат зводить всі різні хороші та погані

аспекти можливо складної системи до єдиного числа, скалярного значення, що дозволяє класифікувати та порівнювати рішення кандидатів. При обчисленні похибки моделі в процесі оптимізації необхідно вибрати функцію втрат. Це може бути складною проблемою, оскільки функція повинна охоплювати властивості проблеми та мотивуватись проблемами, важливими для проекту та зацікавлених сторін. Тому важливо, щоб ця функція сумлінно відображала наші цілі проектування. Якщо ми обираємо погану функцію помилок і отримуємо незадовільні результати, вина полягає в тому, що ми погано визначаємо мету пошуку. Тепер, коли ми знайомі з функцією втрати та втратами, нам потрібно знати, які функції використовувати.

Середня квадратична помилка / квадратична втрата / втрата L2

$$MSE = \frac{\sum \sum_{i=1}^{n}(y_i - y`_i)^2}{n} \quad (2.20)$$

Як випливає з назви, середня квадратична помилка вимірюється як середня квадратна різниця між прогнозами та фактичними спостереженнями. Це стосується лише середньої величини помилок незалежно від їх напряму. Однак завдяки квадрату прогнози, далекі від фактичних значень, сильно штрафуються порівняно з менш відхиленими прогнозами. Плюс MSE має хороші математичні властивості, що полегшує обчислення градієнтів [10,11].

Середня абсолютна похибка

$$MBE = \frac{\sum_{i=1}^{n}|y_i - y`_i|}{n} \quad (2.21)$$

з іншого боку, вимірюється як середня сума абсолютних різниць між прогнозами та фактичними спостереженнями. Як і MSE, це також вимірює величину помилок, не враховуючи їх напрямок. На відміну від MSE, для обчислення градієнтів для MAE потрібні складніші інструменти, такі як лінійне програмування. Плюс до цього, MAE є більш надійним для людей, які не працюють, оскільки не використовує квадрат [10,11].

Середня помилка упередження

$$MBE = \frac{\sum_{i=1}^{n}(y_i - y`_i)}{n} \quad (2.22)$$

набагато рідше в домені машинного навчання порівняно з його аналогом. Це те саме, що MSE з тією лише різницею, що ми не приймаємо абсолютних значень. Очевидно, що потрібно бути обережними, оскільки позитивні та негативні помилки можуть скасувати один одного. Хоча на практиці менш точний, він може визначити, чи модель має позитивний ухил чи негативний ухил [10,11].

Класифікаційні втрати, простіше кажучи, оцінка правильної категорії повинна бути більшою, ніж сума балів усіх неправильних категорій на деякий запас міцності (як правило, один). А отже, втрати використовуються для класифікації максимальної маржі, особливо для машин, що підтримують вектор. Хоча це не відрізняється, це опукла функція, яка полегшує роботу зі звичайними опуклими оптимізаторами, які використовуються в машинному навчанні [10].

$$SVMLoss = \sum_{i \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad (2.23)$$

Перехресна втрата ентропії /негативна логарифмічна правдоподібність. Це найпоширеніша установка для проблем класифікації. Перехресні ентропії втрати збільшуються, коли прогнозована ймовірність відхиляється від фактичної мітки [11].

$$CrossEntropyLoss = -(y_i log(y'_i) + (1 - y_i)\log(1 - y'_i) \quad (2.24)$$

Зауважте, що коли фактична мітка дорівнює 1 (y (i) = 1), друга половина функції зникає, тоді як у випадку, коли фактична мітка дорівнює 0 (y (i) = 0), перша половина випадає. Коротше кажучи, ми просто множимо фактичну передбачуваної ймовірність для класу основної істини. Важливим аспектом цього є те, що перехресні втрати ентропії сильно штрафують впевнені, але неправильні прогнози [11].

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Формування вхідних даних для навчання

Програма починається з встановлення констант які наведено в таблиці 3.1

Таблиця 3.1 – Константи

Назва	Опис
FPS	Частота оновлення рендеру
Scale	Значення з якою проходить імітація для пришвидшення імітації
Main engine power	Сила з якою основний двигун протидіє силі притягання
Side engine power	Сила з якою основний двигун протидіє нахилу та вирівнює положення
Initial Random	Швидкість руху апарату з місця появи у випадковому напрямі для ускладнення процесу навчання
Lander Poly	Форма спускного апарату
Leg away	Відстань ноги по горизонталі
Leg Down	Відстань ноги по Вертикалі
Leg Width/Height	Висота та ширина ніг
Leg Torque	Сила з якою нога змінює положення при посадці чим більше тим міцніше вона тримається до апарату
Side Engine Away	Відстань розташування бокових двигунів від центру по горизонталі
Side Engine Height	Відстань розташування бокових двигунів від центру по вертикалі
View Port Width/Height	Ширина екрану при збільшенні впливає на кількість ітерацій та легкість процесу навчання

Продовження таблиці 3.1

Using Side Engine Penalty	Зменшення винагороди за використання бокового двигуна
Using Main Engine Penalty	Зменшення винагороди за використання головного двигуна
Leg Contact	Збільшення винагороди за доторк ніжки до поверхні
Crashed/Landed	Зміна винагороди за посадку та падіння
Landing Zero Speed Penalty	Збільшення винагороди за нульову швидкість перед контактом
Completed Reward	Значення винагороди потрібне для виконання

По-перше відбудеться поява літального засобу та прикладення на нього відхилення в випадковому напрямку з силою Initial Random. Звісно перший рендер буде без керуючих сигналів. Після цього виконається обчислення положення апарату відносно посадкової площі. Буде порахована його кутова швидкість, кут між вертикальною віссю та центром площі, а також швидкість падіння. На основі цих даних визначається керуючий сигнал який потрібно зробити це можуть бути такі сигнали: нахили вліво або вправо для вирівнювання кута та зменшення швидкості або підйом від поверхні, які приведуть до зменшення кута до 22 градусів та доторку обох ніжок. Включно після керуючих сигналів буде порахована нагорода для них. Таким чином після певної кількості ітерацій алгоритм зможе відсіяти сигнали які не підходять для виконання та залишить ті які сприяють досягненню цілі.

Для наочності виконання алгоритму проілюструємо зразки спостереження та їх нагород для кожної 20-ї ітерації. З ліва на право: горизонтальна координата, вертикальна координата, горизонтальна швидкість,

вертикальна швидкість, між осевий кут апарату та площі, кутова швидкість, дотик першої ніжки, дотик другої ніжки (Рис. 3.1).

```
observations: +0.01 +1.36 +0.53 -1.05 -0.02 -0.12 +0.00 +0.00
step 0 total_reward +0.27
observations: +0.10 +0.86 +0.37 -0.96 +0.12 +0.14 +0.00 +0.00
step 20 total_reward +50.57
observations: +0.15 +0.54 +0.12 -0.55 +0.21 +0.01 +0.00 +0.00
step 40 total_reward +113.82
observations: +0.16 +0.34 -0.07 -0.37 +0.18 -0.07 +0.00 +0.00
step 60 total_reward +151.25
observations: +0.12 +0.21 -0.21 -0.26 +0.06 -0.11 +0.00 +0.00
step 80 total_reward +178.44
observations: +0.08 +0.12 -0.13 -0.16 +0.00 -0.00 +0.00 +0.00
step 100 total_reward +202.67
observations: +0.06 +0.05 -0.09 -0.14 -0.03 -0.03 +0.00 +0.00
step 120 total_reward +207.41
observations: +0.04 +0.01 -0.03 -0.04 -0.04 +0.06 +0.00 +0.00
step 140 total_reward +218.04
observations: +0.04 +0.00 +0.00 +0.00 -0.01 -0.00 +1.00 +1.00
step 160 total_reward +246.47
observations: +0.04 +0.00 +0.00 +0.00 -0.01 +0.00 +1.00 +1.00
step 177 total_reward +346.48
```

Рисунок 3.1 – Ілюстрація виведення зміни нагород протягом навчання з підкріпленням

На основі даних спостереження буде обрано потрібний керуючий сигнал для виконання. Таким чином дані про положення та швидкість буде перетворена у сигнали які потрібно виконати.

Після вдалого результату навчання виконується рендер середи. Для середи було обрано просту 2D версію OpenAiGym – Box2D (Рис. 3.2).

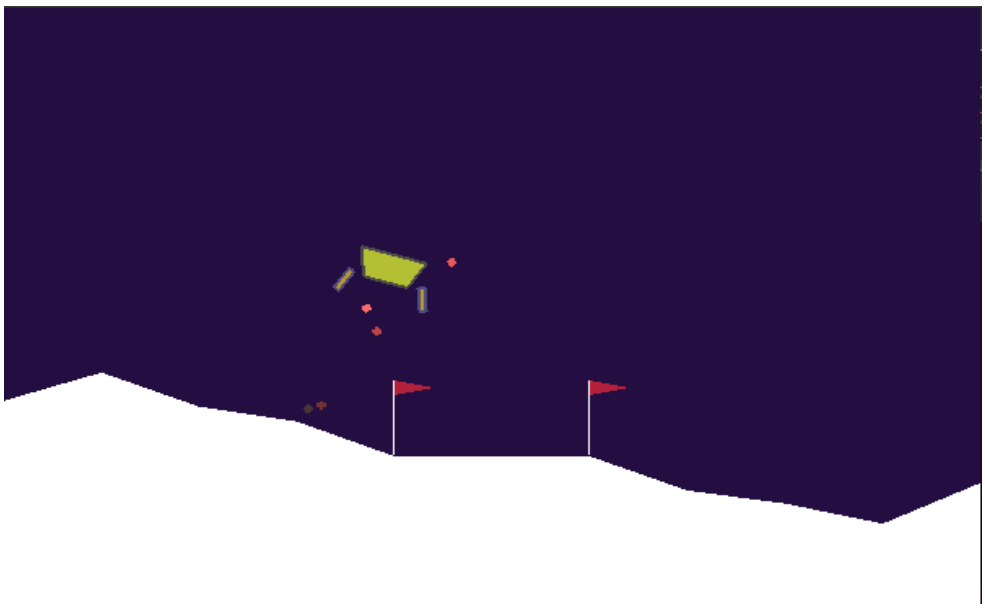

Рисунок 3.2 – Візуальне подання середовища симуляції

Яка розроблена для керування роботом в симуляції. У цьому середовищі використовується фізичний двигун MuJoCo, який був розроблений для швидкого та точного моделювання робота. Включено декілька середовищ недавнього еталону дослідників UC Berkeley.

3.2 Короткий опис програмної реалізації

Розробка здійснювалась в середовищі OpenGym AI на мові Python. Розроблене програмне забезпечення потребує використання сторонніх бібліотек, опис яких наведено в таблиці 3.2.

Таблиця 3.2 – використані бібліотеки

Назва	Опис
Numpy	пакет масивної обробки масивів. Він забезпечує високоефективний багатовимірний об’єкт масиву та інструменти для роботи з цими масивами. Це основний пакет наукових обчислень з Python
PyLab	процедурний інтерфейс до об'єктно-орієнтованої бібліотеки графіків Matplotlib
TensorBoard	забезпечує візуалізацію та інструментарій, необхідний для експериментів з машинним навчанням

Для полегшення роботи з масивами було використано Numpy. За допомогою PyLab були побудовані графіки для наочності результатів навчання алгоритму. TensorBoard була використана для візуалізації метрик втрат та процесу навчання, а саме нагород за керуючі сигнали, що показує як навчалася програма (Рис. 3.3).

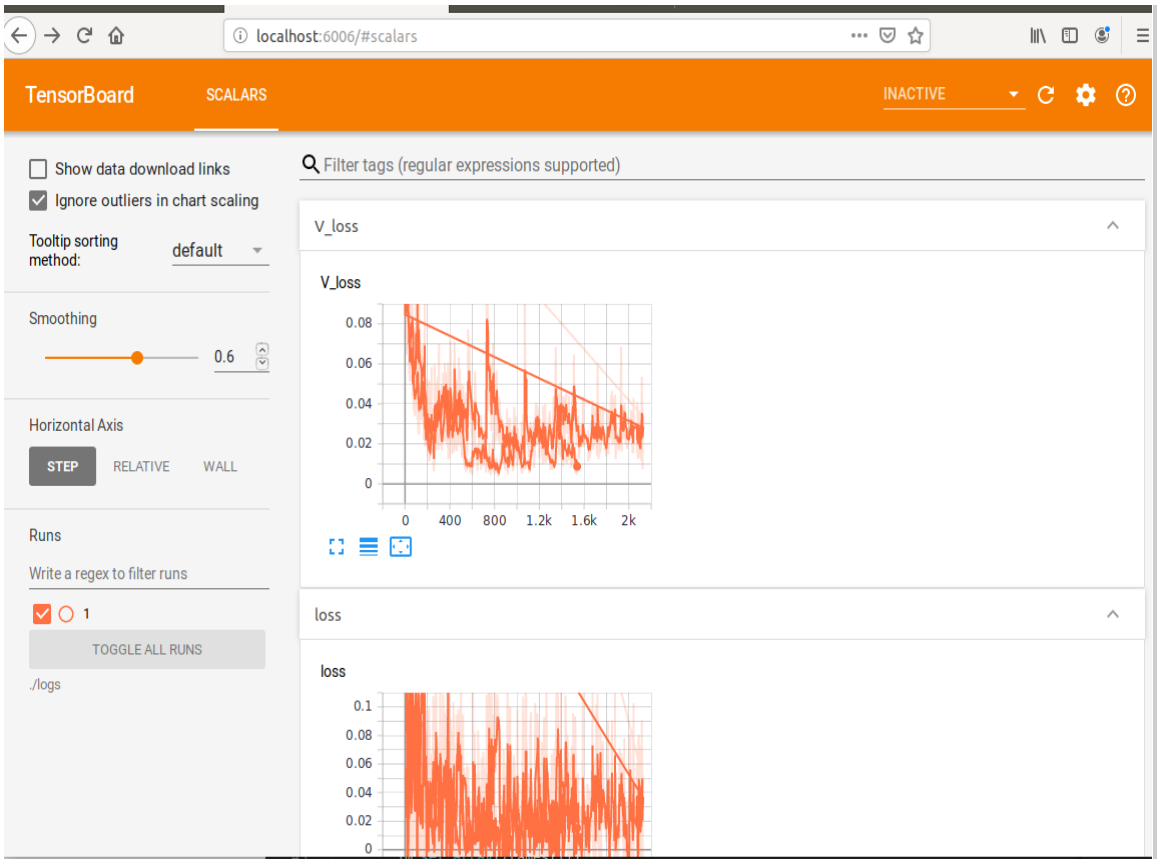

Рисунок 3.3 – Графічний інтерфейс системи моніторингу процесу навчання Tensorboard

Програма складається з таких одного файлу та містить такі класи: LunarLander, ContactDetector, LunarLanderContinuous, їх методи описані в таблицях. 3.3, 3.4, 3.5 відповідно.

Таблиця 3.3 – методи класу LunarLander

Назва	Опис
__init__	Конструктор для створення властивостей об’єкту для спускного апарату та середовища. Також викликає інші методи.
seed	Повертає числове значення для створення навколишнього середовища
_destroy	Метод очищення симуляції від об’єктів
reset	Старт нового середовища
_create_particle	Візуалізація роботи двигунів

Продовження таблиці 3.3

_clean_particles	Очищення екрану від ефектів роботи двигуна
step	Виконання керуючого сигналу та обчислення нагороди після нього
render	Виведення на екран апарату та середовища
close	Закриває вікно

Таблиця 3.4 – методи класу ContactDetector

Назва	Опис
BeginContact	Слухач події контакту апарату та вказує що посадка завершилася або не вдалася якщо контактують не ніжки
EndContact	Вказує що після нового зльоту ніжки не торкаються поверхні

Таблиця 3.5 – методи класу LunarLanderContinuous

Назва	Опис
demo_heuristic_lander	Запуск програми викликом основних методів класу LunarLander

3.3 Аналіз результатів навчання та екзамену

Для перевірки ефективності було використано графіки Tensorboard, що показують винагороду за керуючі сигнали які виконав апарат на певному кроці Рис(3.4).

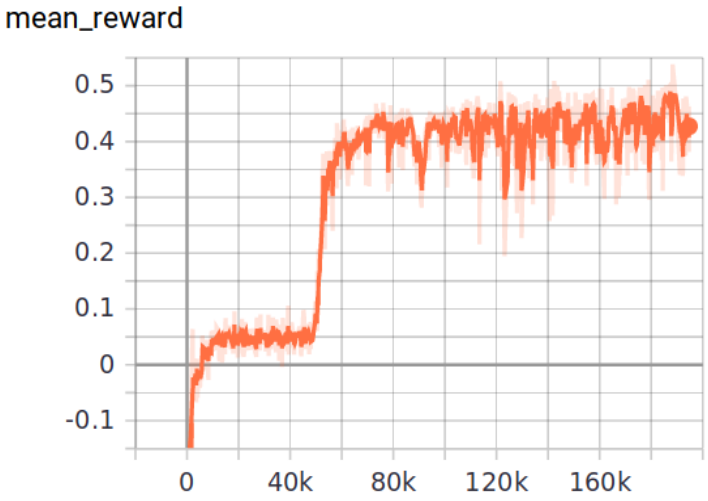

Рисунок 3.4 – Графік зміни середньої нагороди за один крок алгоритму протягом навчання

Починаючи, приблизно, з кроку 160 апарат був приземлений і збільшував винагороду з 226 до 337, завдяки простою двигунів, тому, не було штрафів через це отримано найбільший результат, що не впливало на зменшення результату функції втрат. Хоча й 226 було достатньо. Оптимальні параметри зображені на рисунку 3.5.

```
observations: -0.01 +1.37 -0.54 -0.90 +0.02 +0.13 +0.00 +0.00
step 0 total_reward -0.14
observations: -0.10 +0.89 -0.38 -0.97 -0.11 -0.18 +0.00 +0.00
step 20 total_reward +35.73
observations: -0.16 +0.56 -0.18 -0.57 -0.22 -0.05 +0.00 +0.00
step 40 total_reward +96.17
observations: -0.18 +0.35 +0.04 -0.33 -0.23 +0.10 +0.00 +0.00
step 60 total_reward +137.63
observations: -0.15 +0.22 +0.19 -0.23 -0.15 +0.09 +0.00 +0.00
step 80 total_reward +158.36
observations: -0.10 +0.13 +0.30 -0.13 -0.03 +0.13 +0.00 +0.00
step 100 total_reward +174.64
observations: -0.05 +0.06 +0.15 -0.16 +0.03 +0.02 +0.00 +0.00
step 120 total_reward +191.54
observations: -0.02 +0.01 +0.20 -0.10 +0.08 +0.04 +0.00 +0.00
step 140 total_reward +187.05
observations: +0.01 +0.00 +0.11 +0.00 -0.00 +0.00 +1.00 +1.00
step 160 total_reward +226.85
observations: +0.02 +0.00 -0.00 -0.00 -0.00 +0.00 +1.00 +1.00
step 180 total_reward +237.00
observations: +0.02 +0.00 +0.00 +0.00 -0.00 -0.00 +1.00 +1.00
step 200 total_reward +237.03
observations: +0.02 +0.00 +0.00 +0.00 -0.00 +0.00 +1.00 +1.00
step 204 total_reward +337.03
```

Рисунок 3.5 – Ілюстрація виведення отриманих параметрів системи керування під час навчання

Тут видно отриманий мінімально дозволений кут, відсутність швидкостей, тобто, пересування та дотик обох ніг. Дотик обох ніг доказує, що приземлення виконано правильною стороною. Двигуни були використані у 120 ітераціях і кут для центрування змінювався на протязі 120 ітерацій з 0.02 радіан до 0.23 радіан, тому що апарат починає здійснювати посадку з певною швидкістю, а потім стабілізується. Збіжність ітерацій використання двигунів та зміни кута підтверджує, що алгоритм намагається використовувати паливо тільки при необхідності. На ітерації 140 також видно, що кут змінився на 0 радіан тобто посадка відбулася точно по центру площі. Також по четвертому числу видно різницю зміни вертикальної швидкості яку апарат з 0.97 змінив до 0.10 майже перед посадкою, що вказує на виконання умови для плавної посадки. Таким чином було досягнуто виконання усіх вимог.

ВИСНОВКИ

В роботі було проаналізовано основні тенденції використання автоматичних систем керування та, сучасні способи стабілізації систем керування та підкреслено проблематику багато параметричної оптимізації. У роботі виконано огляд основних методів інтелектуальних алгоритмів у керування складними системами. Обґрунтовано використання ідей та методів інтелектуальних алгоритмів для керування літальними апаратами. Та розроблено модель для використання навчання з підкріпленням у задачі автоматичної посадки літального засобу

Досліджено метод самонавчання системи керування приземленням літального апарату, що оснований на алгоритмах машинного навчання з підкріпленням.

Під час реалізації запропонованого у роботі алгоритму було створено симуляцію у середовищі OpenAIGym та вирішено ряд завдань зі створення інформаційного та програмного забезпечення програми:

1) створення та налаштування середовища симуляції з урахуванням фізичних параметрів апарату;

2) реалізація алгоритму керуючих впливів літального апарату на основі математичної моделі нейромережі;

3) програмна реалізація алгоритму навчання з підкріпленням.

Практичне значення отриманих результатів полягає у створенні методологічної основи для побудови інтелектуальних алгоритмів керування літальними апаратами

СПИСОК ЛІТЕРАТУРИ

1. Аттетков А.В. Методи оптимізації: Посіб. для вузів / Аттетков А.В., Галкин С.В., Зарубін В.С; під ред. В.С. Зарубіна, А.П. Крищенко – МГТУ ім. Н.Э. Баумана — 2003. – 440с.
2. Васильев Ф.П. Методи оптимізації / Васильев Ф.П. – М.: Факторіал Пресс — 2002. – 824с.
3. Гайдук А.Р. Теорія автоматичного управління в прикладах і задачах з рішеннями в MATLAB/ Гайдук А.Р., Беляев В.Е., Пьявченко Т.А. – СПб.: Лань — 2011. – 464с.
4. Гарипов В.Р. Многокритеріальна оптимізація систем управління важкими об’єктами та методами еволюційного пошуку: Красноярск — 1999. – 166с.
5. Springer Verlag Science China Technological Sciences Issue — 5, May 2020
6. Genevieve Hayes Getting Started with Reinforcement Learning and Open AI Gym — 2019
7. Antonios Gasteratos, A. S. Xanthopoulos, Dimitrios E. Koulouriotis A Reinforcement Learning Approach for Production Control in Manufacturing Systems, — 2008
8. Jens Kober J. Andrew Bagnell Jan Peters Reinforcement Learning in Robotics: A Survey — 2012
9. Diederik P. Kingma, Jimmy Lei Ba Adam: a Method For Stochastic optimisation — 2017
10. Jason Brownlee, Loss and Loss Functions for Training Deep Learning Neural Networks — 2019
11. Ravindra Parmar, Common Loss functions in machine learning — 2018

ДОДАТОК А

```
import sys, math
import numpy as np

import Box2D
from Box2D.b2 import (edgeShape, circleShape, fixtureDef, polygonShape, revoluteJointDef, contactListener)

import gym
from gym import spaces
from gym.utils import seeding, EzPickle

FPS = 50
SCALE = 30.0

MAIN_ENGINE_POWER = 13.0
SIDE_ENGINE_POWER = 0.6

INITIAL_RANDOM = 1000.0

LANDER_POLY =[
    (-14, +17), (-17, 0), (-17 ,-10),
    (+17, -10), (+17, 0), (+14, +17)
    ]
LEG_AWAY = 20
LEG_DOWN = 18
LEG_W, LEG_H = 2, 8
LEG_SPRING_TORQUE = 40

SIDE_ENGINE_HEIGHT = 14.0
SIDE_ENGINE_AWAY = 12.0

VIEWPORT_W = 600
VIEWPORT_H = 400


class ContactDetector(contactListener):
    def __init__(self, env):
        contactListener.__init__(self)
        self.env = env

    def BeginContact(self, contact):
        if self.env.lander == contact.fixtureA.body or self.env.lander == contact.fixtureB.body:
            self.env.game_over = True
        for i in range(2):
            if self.env.legs[i] in [contact.fixtureA.body, contact.fixtureB.body]:
                self.env.legs[i].ground_contact = True

    def EndContact(self, contact):
        for i in range(2):
            if self.env.legs[i] in [contact.fixtureA.body, contact.fixtureB.body]:
                self.env.legs[i].ground_contact = False


class LunarLander(gym.Env, EzPickle):
    metadata = {
        'render.modes': ['human', 'rgb_array'],
```

```
    'video.frames_per_second' : FPS
}

continuous = False

def __init__(self):
    EzPickle.__init__(self)
    self.seed()
    self.viewer = None

    self.world = Box2D.b2World()
    self.moon = None
    self.lander = None
    self.particles = []

    self.prev_reward = None


    self.observation_space = spaces.Box(-np.inf, np.inf, shape=(8,), dtype=np.float32)

    if self.continuous:



        self.action_space = spaces.Box(-1, +1, (2,), dtype=np.float32)
    else:

        self.action_space = spaces.Discrete(4)

    self.reset()

def seed(self, seed=None):
    self.np_random, seed = seeding.np_random(seed)
    return [seed]

def _destroy(self):
    if not self.moon: return
    self.world.contactListener = None
    self._clean_particles(True)
    self.world.DestroyBody(self.moon)
    self.moon = None
    self.world.DestroyBody(self.lander)
    self.lander = None
    self.world.DestroyBody(self.legs[0])
    self.world.DestroyBody(self.legs[1])

def reset(self):
    self._destroy()
    self.world.contactListener_keepref = ContactDetector(self)
    self.world.contactListener = self.world.contactListener_keepref
    self.game_over = False
    self.prev_shaping = None

    W = VIEWPORT_W/SCALE
```

```
H = VIEWPORT_H/SCALE


CHUNKS = 11
height = self.np_random.uniform(0, H/2, size=(CHUNKS+1,))
chunk_x = [W/(CHUNKS-1)*i for i in range(CHUNKS)]
self.helipad_x1 = chunk_x[CHUNKS//2-1]
self.helipad_x2 = chunk_x[CHUNKS//2+1]
self.helipad_y = H/4
height[CHUNKS//2-2] = self.helipad_y
height[CHUNKS//2-1] = self.helipad_y
height[CHUNKS//2+0] = self.helipad_y
height[CHUNKS//2+1] = self.helipad_y
height[CHUNKS//2+2] = self.helipad_y
smooth_y = [0.33*(height[i-1] + height[i+0] + height[i+1]) for i in range(CHUNKS)]

self.moon = self.world.CreateStaticBody(shapes=edgeShape(vertices=[(0, 0), (W, 0)]))
self.sky_polys = []
for i in range(CHUNKS-1):
    p1 = (chunk_x[i], smooth_y[i])
    p2 = (chunk_x[i+1], smooth_y[i+1])
    self.moon.CreateEdgeFixture(
        vertices=[p1,p2],
        density=0,
        friction=0.1)
    self.sky_polys.append([p1, p2, (p2[0], H), (p1[0], H)])

self.moon.color1 = (0.0, 0.0, 0.0)
self.moon.color2 = (0.0, 0.0, 0.0)

initial_y = VIEWPORT_H/SCALE
self.lander = self.world.CreateDynamicBody(
    position=(VIEWPORT_W/SCALE/2, initial_y),
    angle=0.0,
    fixtures = fixtureDef(
        shape=polygonShape(vertices=[(x/SCALE, y/SCALE) for x, y in LANDER_POLY]),
        density=5.0,
        friction=0.1,
        categoryBits=0x0010,
        maskBits=0x001,
        restitution=0.0)
        )
self.lander.color1 = (0.5, 0.4, 0.9)
self.lander.color2 = (0.3, 0.3, 0.5)
self.lander.ApplyForceToCenter( (
    self.np_random.uniform(-INITIAL_RANDOM, INITIAL_RANDOM),
    self.np_random.uniform(-INITIAL_RANDOM, INITIAL_RANDOM)
    ), True)

self.legs = []
for i in [-1, +1]:
    leg = self.world.CreateDynamicBody(
        position=(VIEWPORT_W/SCALE/2 - i*LEG_AWAY/SCALE, initial_y),
        angle=(i * 0.05),
```

```
                fixtures=fixtureDef(
                    shape=polygonShape(box=(LEG_W/SCALE, LEG_H/SCALE)),
                    density=1.0,
                    restitution=0.0,
                    categoryBits=0x0020,
                    maskBits=0x001)
                )
            leg.ground_contact = False
            leg.color1 = (0.5, 0.4, 0.9)
            leg.color2 = (0.3, 0.3, 0.5)
            rjd = revoluteJointDef(
                bodyA=self.lander,
                bodyB=leg,
                localAnchorA=(0, 0),
                localAnchorB=(i * LEG_AWAY/SCALE, LEG_DOWN/SCALE),
                enableMotor=True,
                enableLimit=True,
                maxMotorTorque=LEG_SPRING_TORQUE,
                motorSpeed=+0.3 * i
                )
            if i == -1:
                rjd.lowerAngle = +0.9 - 0.5
                rjd.upperAngle = +0.9
            else:
                rjd.lowerAngle = -0.9
                rjd.upperAngle = -0.9 + 0.5
            leg.joint = self.world.CreateJoint(rjd)
            self.legs.append(leg)

        self.drawlist = [self.lander] + self.legs

        return self.step(np.array([0, 0]) if self.continuous else 0)[0]

    def _create_particle(self, mass, x, y, ttl):
        p = self.world.CreateDynamicBody(
            position = (x, y),
            angle=0.0,
            fixtures = fixtureDef(
                shape=circleShape(radius=2/SCALE, pos=(0, 0)),
                density=mass,
                friction=0.1,
                categoryBits=0x0100,
                maskBits=0x001,
                restitution=0.3)
                )
        p.ttl = ttl
        self.particles.append(p)
        self._clean_particles(False)
        return p

    def _clean_particles(self, all):
        while self.particles and (all or self.particles[0].ttl < 0):
            self.world.DestroyBody(self.particles.pop(0))
```

```
def step(self, action):
    if self.continuous:
        action = np.clip(action, -1, +1).astype(np.float32)
    else:
        assert self.action_space.contains(action), "%r (%s) invalid " % (action, type(action))


    tip  = (math.sin(self.lander.angle), math.cos(self.lander.angle))
    side = (-tip[1], tip[0])
    dispersion = [self.np_random.uniform(-1.0, +1.0) / SCALE for _ in range(2)]

    m_power = 0.0
    if (self.continuous and action[0] > 0.0) or (not self.continuous and action == 2):

        if self.continuous:
            m_power = (np.clip(action[0], 0.0,1.0) + 1.0)*0.5
            assert m_power >= 0.5 and m_power <= 1.0
        else:
            m_power = 1.0
        ox = (tip[0] * (4/SCALE + 2 * dispersion[0]) +
              side[0] * dispersion[1])
        oy = -tip[1] * (4/SCALE + 2 * dispersion[0]) - side[1] * dispersion[1]
        impulse_pos = (self.lander.position[0] + ox, self.lander.position[1] + oy)
        p = self._create_particle(3.5,
                                  impulse_pos[0],
                                  impulse_pos[1],
                                  m_power)
        p.ApplyLinearImpulse((ox * MAIN_ENGINE_POWER * m_power, oy * MAIN_ENGINE_POWER * m_power),
                             impulse_pos,
                             True)
        self.lander.ApplyLinearImpulse((-ox * MAIN_ENGINE_POWER * m_power, -oy * MAIN_ENGINE_POWER * m_power),
                                       impulse_pos,
                                       True)

    s_power = 0.0
    if (self.continuous and np.abs(action[1]) > 0.5) or (not self.continuous and action in [1, 3]):

        if self.continuous:
            direction = np.sign(action[1])
            s_power = np.clip(np.abs(action[1]), 0.5, 1.0)
            assert s_power >= 0.5 and s_power <= 1.0
        else:
            direction = action-2
            s_power = 1.0
        ox = tip[0] * dispersion[0] + side[0] * (3 * dispersion[1] + direction * SIDE_ENGINE_AWAY/SCALE)
        oy = -tip[1] * dispersion[0] - side[1] * (3 * dispersion[1] + direction * SIDE_ENGINE_AWAY/SCALE)
        impulse_pos = (self.lander.position[0] + ox - tip[0] * 17/SCALE,
                       self.lander.position[1] + oy + tip[1] * SIDE_ENGINE_HEIGHT/SCALE)
        p = self._create_particle(0.7, impulse_pos[0], impulse_pos[1], s_power)
        p.ApplyLinearImpulse((ox * SIDE_ENGINE_POWER * s_power, oy * SIDE_ENGINE_POWER * s_power),
                             impulse_pos
                             , True)
        self.lander.ApplyLinearImpulse((-ox * SIDE_ENGINE_POWER * s_power, -oy * SIDE_ENGINE_POWER * s_power),
                                       impulse_pos,
```

```
                                                    True)

        self.world.Step(1.0/FPS, 6*30, 2*30)

        pos = self.lander.position
        vel = self.lander.linearVelocity
        state = [
            (pos.x - VIEWPORT_W/SCALE/2) / (VIEWPORT_W/SCALE/2),
            (pos.y - (self.helipad_y+LEG_DOWN/SCALE)) / (VIEWPORT_H/SCALE/2),
            vel.x*(VIEWPORT_W/SCALE/2)/FPS,
            vel.y*(VIEWPORT_H/SCALE/2)/FPS,
            self.lander.angle,
            20.0*self.lander.angularVelocity/FPS,
            1.0 if self.legs[0].ground_contact else 0.0,
            1.0 if self.legs[1].ground_contact else 0.0
            ]
        assert len(state) == 8

        reward = 0
        shaping = \
            - 100*np.sqrt(state[0]*state[0] + state[1]*state[1]) \
            - 100*np.sqrt(state[2]*state[2] + state[3]*state[3]) \
            - 100*abs(state[4]) + 10*state[6] + 10*state[7]

        if self.prev_shaping is not None:
            reward = shaping - self.prev_shaping
        self.prev_shaping = shaping

        reward -= m_power*0.30
        reward -= s_power*0.03

        done = False
        if self.game_over or abs(state[0]) >= 1.0:
            done = True
            reward = -100
        if not self.lander.awake:
            done = True
            reward = +100
        return np.array(state, dtype=np.float32), reward, done, {}

    def render(self, mode='human'):
        from gym.envs.classic_control import rendering
        if self.viewer is None:
            self.viewer = rendering.Viewer(VIEWPORT_W, VIEWPORT_H)
            self.viewer.set_bounds(0, VIEWPORT_W/SCALE, 0, VIEWPORT_H/SCALE)

        for obj in self.particles:
            obj.ttl -= 0.15
            obj.color1 = (max(0.2, 0.2+obj.ttl), max(0.2, 0.5*obj.ttl), max(0.2, 0.5*obj.ttl))
            obj.color2 = (max(0.2, 0.2+obj.ttl), max(0.2, 0.5*obj.ttl), max(0.2, 0.5*obj.ttl))

        self._clean_particles(False)

        for p in self.sky_polys:
```

```
        self.viewer.draw_polygon(p, color=(0, 0, 0))

    for obj in self.particles + self.drawlist:
        for f in obj.fixtures:
            trans = f.body.transform
            if type(f.shape) is circleShape:
                t = rendering.Transform(translation=trans*f.shape.pos)
                self.viewer.draw_circle(f.shape.radius, 20, color=obj.color1).add_attr(t)
                self.viewer.draw_circle(f.shape.radius, 20, color=obj.color2, filled=False, linewidth=2).add_attr(t)
            else:
                path = [trans*v for v in f.shape.vertices]
                self.viewer.draw_polygon(path, color=obj.color1)
                path.append(path[0])
                self.viewer.draw_polyline(path, color=obj.color2, linewidth=2)

    for x in [self.helipad_x1, self.helipad_x2]:
        flagy1 = self.helipad_y
        flagy2 = flagy1 + 50/SCALE
        self.viewer.draw_polyline([(x, flagy1), (x, flagy2)], color=(1, 1, 1))
        self.viewer.draw_polygon([(x, flagy2), (x, flagy2-10/SCALE), (x + 25/SCALE, flagy2 - 5/SCALE)],
                                 color=(0.8, 0.8, 0))

    return self.viewer.render(return_rgb_array=mode == 'rgb_array')

def close(self):
    if self.viewer is not None:
        self.viewer.close()
        self.viewer = None
class LunarLanderContinuous(LunarLander):
    continuous = True

def heuristic(env, s):
    """
    The heuristic for
    1. Testing
    2. Demonstration rollout.
    Args:
        env: The environment
        s (list): The state. Attributes:
                  s[0] is the horizontal coordinate
                  s[1] is the vertical coordinate
                  s[2] is the horizontal speed
                  s[3] is the vertical speed
                  s[4] is the angle
                  s[5] is the angular speed
                  s[6] 1 if first leg has contact, else 0
                  s[7] 1 if second leg has contact, else 0
    returns:
         a: The heuristic to be fed into the step function defined above to determine the next step and reward.
    """

    angle_targ = s[0]*0.5 + s[2]*1.0
    if angle_targ > 0.4: angle_targ = 0.4
    if angle_targ < -0.4: angle_targ = -0.4
    hover_targ = 0.55*np.abs(s[0])

    angle_todo = (angle_targ - s[4]) * 0.5 - (s[5])*1.0
    hover_todo = (hover_targ - s[1])*0.5 - (s[3])*0.5

    if s[6] or s[7]:
        angle_todo = 0
        hover_todo = -(s[3])*0.5
```

```
    if env.continuous:
        a = np.array([hover_todo*20 - 1, -angle_todo*20])
        a = np.clip(a, -1, +1)
    else:
        a = 0
        if hover_todo > np.abs(angle_todo) and hover_todo > 0.05: a = 2
        elif angle_todo < -0.05: a = 3
        elif angle_todo > +0.05: a = 1
    return a

def demo_heuristic_lander(env, seed=None, render=False):
    env.seed(seed)
    total_reward = 0
    steps = 0
    s = env.reset()
    while True:
        a = heuristic(env, s)
        s, r, done, info = env.step(a)
        total_reward += r

        if render:
            still_open = env.render()
            if still_open == False: break

        if steps % 20 == 0 or done:
            print("observations:", " ".join(["{:+0.2f}".format(x) for x in s]))
            print("step {} total_reward {:+0.2f}".format(steps, total_reward))
        steps += 1
        if done: break
    return total_reward


if __name__ == '__main__':
    demo_heuristic_lander(LunarLander(), render=True)
```