НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

«На правах рукопису»
УДК 004.85+004.9]:[629.05:623.746-519](043.3)

До захисту допущено:
В.о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

зі спеціальності 122 «Комп'ютерні науки»

на тему: «Глибоке навчання з підкріпленням для керування дроном в задачі переслідування»

Виконала:
студентка ІІ курсу групи КІ-21мп
Рибалко Анастасія Анатоліївна
_____

Науковий керівник:
проф. кафедри ММСА, д.т.н., професор
Касьянов Павло Олегович
_____

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

Київ – 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – другий (магістерський)
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В.о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Рибалко Анастасії Анатоліївні

1. Тема дисертації «Глибоке навчання з підкріпленням для керування дроном в задачі переслідування», науковий керівник дисертації Касьянов Павло Олегович, професор кафедри СМШІ Інституту прикладного системного аналізу, затверджені наказом по університету від «08» листопада 2023 р. №5200-с
2. Термін подання студентом дисертації 07 січня 2024 року.
3. Об'єкт дослідження: задача переслідування, методи навчання з підкріпленням в задачі переслідування.
4. Вихідні дані: наукові статті по методам навчання з підкріпленням зазначені у переліку джерел.
5. Перелік завдань, які потрібно розробити:
    1) Дослідити предметну область та існуючі системи
    2) Здійснити огляд технічної літератури
    3) Розробити алгоритми навчання моделі різними методами навчання з підкріпленням та дослідити їх ефективність в контексті задачі переслідування.

4) Провести аналіз результатів
5) Розробити аналіз ринкових можливостей стартап-проєкту
6) Зробити висновки з отриманих результатів та оформити пояснювальну записку.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

Презентація

7. Орієнтовний перелік публікацій

Заплановано апробувати роботу "Порівняння ефективності методів навчання з підкріпленням в задачі переслідування автономним дроном" авторів Рибалко А.А., Касьянова П.О. на V Міжнародній науково-практичній конференції «Scientific Community: Interdisciplinary Research» (Hamburg, Germany, 2024).

8. Дата видачі завдання: 30 серпня 2023р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми МД	01.09.2023 - 07.09.2023	Виконано
2	Ознайомлення зі структурою МД згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	08.09.2023 - 14.09.2023	Виконано
3	Ознайомлення з ДСТУ 3008-2015 та стандартами ЄСПД	15.09.2023 - 21.09.2023	Виконано
4	Проведення дослідження за темою МД під керівництвом керівника	22.09.2023 - 28.09.2023	Виконано
5	Завершення роботи над першим варіантом частини МД	29.09.2023 - 05.10.2023	Виконано
6	Проведення роботи над експериментальною частиною МД	06.10.2023 - 12.10.2023	Виконано
7	Проведення роботи над програмним продуктом	13.10.2023 - 03.12.2023	Виконано
8	Оформлення МД та аналіз отриманих результатів	03.12.2023 - 07.01.2023	Виконано

Студент Рибалко Анастасія Анатоліївна

Науковий керівник Касьянов Павло Олегович

РЕФЕРАТ

Магістерська дисертація: 139с., 29 рис., 11 табл., 1 додаток, 52 посилання.

ГЛИБОКЕ НАВЧАННЯ З ПІДКРІПЛЕННЯМ ДЛЯ КЕРУВАННЯ ДРОНОМ В ЗАДАЧІ ПЕРЕСЛІДУВАННЯ, МАШИННЕ НАВЧАННЯ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, DQN, ЗАДАЧА ПЕРЕСЛІДУВАННЯ, PPO, ДРОН, БПЛА, КАРТА ГЛИБИН.

Об’єкт дослідження – завдання керування дроном в рамках задачі переслідування об’єктів.

Предмет дослідження – застосування методів глибокого навчання з підкріпленням для вирішення завдання керування дроном у задачі переслідування об'єктів.

Мета роботи – розробка та оптимізація системи керування дроном на основі глибокого навчання з підкріпленням з метою ефективного переслідування об'єктів в умовах обмежених обчислювальних ресурсів.

Досліджено, наскільки використання навчання з підкріпленням є життєздатним методом для ведення переслідування за допомогою автономного дрона. Результати показали, що існує потенціал у використанні методів RL, зокрема мереж DQN та PPO, для цього завдання, особливо у порівнянні із прямими статичними підходами. Крім того, реалізація представлень стану, які включають інформацію про динамічні рухи об’єктів та їхні відстані, демонструє значні переваги перед алгоритмами RL, які покладаються виключно на вхідні дані камери.

Результати цієї роботи рекомендується використовувати для навчання моделей напівкерованого навчання в задачах переслідування.

Результати цієї роботи заплановано апробувати на міжнародній конференції.

ABSTRACT

Master's thesis: 139c., 29 figures, 11 tables, 1 appendix, 52 references.

DEEP REINFORCEMENT LEARNING FOR AUTONOMOUS DRONE TRACKING AND FOLLOWING, MACHINE LEARNING, REINFORCEMENT LEARNING, DQN, DRONE FOLLOWING, PPO, DRONE, UAV, DEPTH MAP.

The object of research is the task of controlling a drone within the framework of the task of chasing objects.

The subject of the study is the application of deep learning methods with reinforcement to solve the task of controlling a drone in the task of tracking objects.

The purpose of the work is to develop and optimize a drone control system based on deep learning with reinforcement for the purpose of effective pursuit of objects in conditions of limited computing resources.

Investigated the extent to which the use of reinforcement learning is a viable method for autonomous drone pursuit. The results showed that there is potential in using RL methods, particularly DQN and PPO networks, for this task, especially compared to direct static approaches. In addition, the implementation of state representations that include information about dynamic object motions and their distances shows significant advantages over RL algorithms that rely solely on camera input.

The results of this work are recommended to be used for training semi-supervised learning models in pursuit tasks.

The results of this work are planned to be tested at an international conference.

ЗМІСТ

ВСТУП

В сучасному світі розвиток технологій безпілотних літальних апаратів, відомих як дрони, набуває все більшого значення в різних сферах життя, включаючи такі галузі як агропромисловий сектор, військова галузь, метеорологія, медицина, фото- та відеозйомка, геодезія та картографія, рятувальні операції тощо. Однак для досягнення максимальної ефективності та безпеки використання дронів вимагається розробка надійних та ефективних систем керування, здатних адаптуватися до різних ситуацій та завдань.

Глибоке навчання з підкріпленням стало однією з передових технологій у галузі автономного керування дронами. Ці алгоритми дозволяють дронам навчатися приймати рішення та виконувати завдання, максимізуючи рівень продуктивності й мінімізуючи потенційні складності та ризики для залучених у виконання завдань людей. Особливо це актуально в контексті задачі переслідування.

Задача переслідування, в якій дрон повинен автоматично визначати та слідкувати за об'єктом, є однією з найбільш складних для автономних систем. Це завдання вимагає від машини швидкості реакції, точності в рухах, здатності передбачати траєкторію об'єкта та адаптуватися до змінних умов середовища.

Не всі реальні ситуації можна передбачити та запрограмувати в дрон, щоб забезпечити правильну поведінку в кожній із них. Від автоматизованого агента вимагається здатність приймати рішення в умовах непередбачуваності. Це перетворюється на необхідність генералізації та екстраполяції поведінки агента на нові ситуації.

Нещодавній розвиток глибокого навчання показав, що машинне навчання застосовне для вирішення великої кількості різних проблем. Однією з переваг використання нейронних мереж є їх здатність узагальнювати навчальні дані на

нові екземпляри, які модель до цього не бачила. Ця можливість узагальнення дуже добре підходить для розвитку автономності в роботизованих системах. Застосування нейронних мереж у сфері навчання з підкріпленням (RL) також пережило величезний бум в останні роки. Складні середовища, де алгоритми RL можуть забезпечити рішення, вимагають обробки великої кількості даних. Тут нейронні мережі є цінним рішенням, вони дають агентам RL можливість обробляти велику кількість даних для свого навчання. Крім того, використання нейронних мереж у контексті RL також додає до здібності агентів до генералізації. Тож, використання нейронних мереж у проблемах RL, так званий Deep Reinforcement Learning (DRL), можна вважати корисною парадигмою для дослідження в контексті її застосовності в задачах переслідування об’єктів.

У цій дипломній роботі проводиться детальний аналіз та дослідження методів глибокого навчання з підкріпленням для розробки системи керування дроном в задачі переслідування. Вивчаються основні принципи та алгоритми глибокого навчання з підкріпленням, а також розглядається їх застосування до практичних завдань керування дронами. Робота націлена на побудову ефективної системи керування дроном в задачі переслідування в умовах обмежених ресурсів.

Актуальність дослідження підкріплена наступними чинниками:

1. Швидке розширення використання дронів. У сучасному світі дрони стали однією з найбільш динамічно розвиваючихся технологій. Вони використовуються у різних сферах: від фото- та відеозйомки до контролю за дотриманням законодавства, проведення рятувальних операцій та виконання військових завдань.
2. Необхідність автономної роботи дронів. У багатьох ситуаціях людське втручання у керування дроном може бути обмеженим, неможливим або небезпечним. Тому розробка автономних систем

керування є важливим кроком для покращення ефективності та безпеки експлуатації дронів.

3. Складність задачі переслідування. Автоматичне переслідування вимагає навчання складним стратегіям переслідування, високої адаптивності до змін у середовищі та швидкої реакції на непередбачувані обставини.
4. Попит на ефективні рішення в умовах обмежених обчислювальних ресурсів. В завданнях з високим ризиком втрати дрона (наприклад, у пошуково-рятівних операціях в екстремальних умовах або військових задачах) найоптимальнішим є використання ефективних бюджетних рішень.

Отже, дослідження методів глибокого навчання з підкріпленням для керування дроном у задачі переслідування в умовах обмежених обчислювальних потужностей є досить актуальним та важливим.

Метою дослідження є розробка та оптимізація системи керування дроном на основі глибокого навчання з підкріпленням з метою ефективного переслідування об'єктів в середовищах різного ступеня складності.

Її досягнення передбачає вирішення таких завдань як представлено нижче.

1. Аналіз вітчизняних та зарубіжних джерел
2. Підготовка середовищ різного ступеня складності для розробки, навчання та тестування моделі
3. Дослідження алгоритмів глибокого навчання з підкріпленням для задач керування дроном
4. Розробка алгоритму керування дроном у задачах переслідування на основі навчання з підкріпленням.
5. Реалізація алгоритму
6. Валідація результатів

Об'єктом дослідження у даній дипломній роботі виступає завдання керування дроном в рамках задачі переслідування об’єктів.

Предметом дослідження є застосування методів глибокого навчання з підкріпленням для вирішення завдання керування дроном у задачі переслідування об'єктів.

В якості інструментів були використані: мова програмування Python, відкрита програмна бібліотека для машинного навчання Tensorflow та симулятор для безпілотних апаратів AirSim.

РОЗДІЛ 1 АНАЛІЗ ДОСЛІДЖЕНЬ В ПРЕДМЕТНІЙ ОБЛАСТІ

1.1 Введення

Дрони, або безпілотні літальні апарати (БПЛА), набули надзвичайного сплеску популярності у багатьох сферах завдяки універсальності їх застосування, яке охоплює такі галузі як агропромисловий сектор, військова галузь, метеорологія, медицина тощо[1-3]. Однією з найактуальніших задач у сфері БПЛА є відстеження об’єктів. Завдання постійного відстеження цілі при врахуванні власного руху дрона, руху цілі та факторів навколишнього середовища, є нетривіальним.

Умови обмежених обчислювальних ресурсів вимагають моделей, які були б легкими та ефективними. Попередні реалізації, незважаючи на ефективність, часто потребують надійного апаратного забезпечення, що обмежує їх застосування в середовищах з обмеженими ресурсами. Акцент поточної роботи на обчислювальній ефективності та точності при використанні обраного технологічного стеку, відрізняє її від кількох існуючих рішень [4-6].

Цей огляд представляє огляд існуючих методологій для відстеження об’єктів за допомогою дронів, приділяючи особливу увагу технологічному стеку, що використовується у поточній роботі.

1.2 Застосовність RL до поточної задачі

В останні десятиліття дрони зазнали трансформаційних удосконалень як апаратного, так і програмного забезпечення. Одним із застосувань, для якого дрони були ефективно застосовані, є відстеження об’єктів, враховуючи їх мобільність і точку огляду з висоти пташиного польоту. Надійна система

відстеження об’єктів необхідна для отримання точних і ефективних результатів [7].

Ідея реалізації алгоритмів RL у контексті керування дронами не є новою [8-13]. Дослідження показали, що використання RL у різноманітних ситуаціях забезпечує ряд переваг. По-перше, завдяки специфіці процесу навчання агенти RL демонструють ефективність у роботі зі складними та динамічними середовищами [11]. Вони можуть навчатись адаптивній поведінці для вирішення необхідного завдання і екстраполювати свою вивчену поведінку на нові ситуації [13]. Це особливо корисно в ситуаціях, коли завдання передбачають необхідність стеження за динамічними об'єктами.

Існують й інші методи, які застосовувалися в контексті відстеження об’єктів або людей ([14-17]). Ці методи використовують різноманітні технології для покращення можливостей дрона щодо виявлення об’єктів, відстеження об’єктів і процесу прийняття рішень.

Процес вибору дій — це простий цільовий алгоритм, який зосереджується на тому, щоб цільовий об’єкт знаходився в центрі огляду. Додаткові технології вводяться як засіб для цільової поведінки в нестандартних ситуаціях або для покращення загальної стабільності та надійності інформації, яку використовує цей алгоритм.

Вищезазначені переваги використання агентів RL все ще дуже застосовні до процесу вибору дії в цих запропонованих алгоритмах. Здатність бути гнучким і адаптованим забезпечує стійкість агента до нових ситуацій і потенційно може також усунути потребу в додаванні складніших технологій до дрона взагалі. З цих причин дослідження того, яку роль RL міг би взяти на себе в розробці агентів, які керують дронами в задачах відстеження динамічних об’єктів, актуальні.

У сфері RL існує безліч алгоритмів, які можна вибрати для виконання процесу навчання [18-22].

Багато з сучасних алгоритмів показують дуже багатообіцяючі результати. Однак більш фундаментальний алгоритм Q-навчання все ще є широко поширеним вступним методом у різноманітних пошукових дослідженнях [23].

Загалом алгоритми Q-навчання використовують табличне представлення процедури вибору дії, яка характеризується визначенням відповідної дії для кожного можливого стану. Такі методи прості у реалізації та надають широке уявлення про те, як агент RL функціонує в нових середовищах.

Багато досліджень займалося пошуком методів, за допомогою яких RL можна було б використовувати у випадку керування дронами [9-11,24]. Тим не менш, проблема полягає в тому, що багато з цих досліджень досліджували лише специфічні завдання, які повинен виконувати агент RL. Наприклад, одним із таких завдань є навігація різними типами середовищ [10,11,24,25]. Інші більше зосереджуються на дуже конкретних елементах керування безпілотником, як-от зліт і посадка пристрою [26]. Багато з цих досліджень показали сильний позитивний аспект використання RL для подібних завдань.

1.3 Специфіка алгоритмів RL у контексті поточної задачі

Залишається питання, чи може завдання відстеження об’єктів виграти від алгоритмів RL. Були певні спроби дослідити цю проблему [27], що показало багатообіцяючі результати. Однак RL дуже чутливий до змінних навколишнього середовища. Визначення простору станів, простору дій та функції винагороди є ключовими аспектами, які визначають, яку поведінку буде вивчено та наскільки добре ці алгоритми будуть виконувати завдання.

1.4 Визначення функції винагороди

Визначення функції винагороди є дуже важливим у процесі навчання агента з поведінковою метою. Однак визначення винагороди несе з собою деякі проблеми, які необхідно вирішити. Одним із таких викликів є компроміс між тим, скільки заздалегідь визначеної інформації реалізовано у функції винагороди [28,29]. Надання цієї інформації відбувається в тому сенсі, що функція винагороди визначає деякі проміжні стани як більш бажані порівняно з іншими для досягнення мети. Слідуючи цій ієрархії, агент змушений вивчати поведінку певним чином з урахуванням цих обмежень.

Однак недоліком створення такого впорядкування є те, що воно вимагає специфічних для області використання знань, щоб визначити, коли агент ближче до мети. Це проникнення певних заздалегідь визначених знань у функцію винагороди також блокує агента від пошуку нових шляхів вирішення проблеми та є проблемою в ситуаціях, коли спектр допустимої поведінки не є відомим до навчання. Крім цього, такі функції винагороди надзвичайно чутливі до дрібних помилок у порядку станів, що призводить до неоптимальної продуктивності [30].

Альтернатива веде до спрощеної функції винагороди, де цільові стани позначені позитивним сигналом, а решта нейтральним або негативним.

Проблема тут, однак, полягає в розрідженості винагород. Наявність невеликого набору станів, які створюють позитивний сигнал у більшому просторі станів, означає, що існують великі смуги станів, у яких сигнал не подається. Це означає, що під час дослідження простору станів агенту стає важче знайти стани, де спостерігається позитивний сигнал. Крім цього, згадана раніше проблема неочікуваної поведінки, від якої страждають алгоритми RL, також стає більш актуальною [31]. З попередньо визначеною ієрархією в станах свобода агента розробляти абсолютно нові набори поведінки, які оптимізують функцію

винагороди, є менш імовірною. Однак із розрідженістю винагород агент має набагато більше шансів розвинути несподівану поведінку.

Тим не менш, переваги використання розріджених винагород у поєднанні з новітніми технологіями у вдосконаленні процесів навчання алгоритмів RL у таких середовищах [32] надають цьому методу розробки винагород більші можливості.

1.5 Вибір середовища

Вибір середовища, в якому буде проходити навчання агента, є ще одним важливим аспектом для визначення проблеми. Вони можуть варіюватися від середовища типу сусідства [10] до внутрішніх коридорів і кімнат [8] до більш спрощеного абстрактного середовища [9]. Існує чіткий компроміс між використанням спрощених абстрактних середовищ порівняно з більш складними середовищами.

Ступінь, до якого нейронна мережа здатна узагальнювати, не є нескінченною, і розміщення агента в середовищі, абсолютно відмінному від того, у якому його навчали, може спричинити проблеми з його роботою. Крім того, використання більш специфічних середовищ також обмежує можливості агента щодо узагальнення через специфічність ситуацій, що виникають під час навчання. Попередні спроби тестування узагальнювальних здібностей алгоритмів RL у складних ситуаціях показали, що існує великий потенціал для перенесення вивченої поведінки в нове середовище [8].

Однак є ряд проблем, які досі не вирішені. Середовища, які були протестовані, здебільшого схожі за складністю. Можливість узагальнення особливо цікава, коли навчання можна проводити в більш спрощених

середовищах, а знання можна перенести в більш складні ситуації. Ця проблема також залежить від того, як агент сприймає середовище через його введення стану. Навчання агентів зі звичайними (RGB) вхідними сигналами камери, як це було виконано в цьому дослідженні, може створити додаткову проблему, пов’язану з різними колірними просторами. Таким чином, ще є можливість дослідити, чи може інша репрезентація станів потенційно передати ту саму інформацію, не стикаючись із цими проблемами.

1.6 Спосіб представлення станів

Спосіб представлення станів є вирішальним елементом підготовки агентів [33]. Як було показано раніше, представлення стану може бути проблемою, коли йдеться про численні аспекти агентів RL, включаючи їх узагальнюваність [24]. Хороший вибір стану є основою для сприйняття агентом середовища. Він також визначає, яку інформацію агент може використовувати для прийняття рішень. Інформація повинна містити важливі аспекти, які впливають на сигнал винагороди в усьому середовищі. Якщо агент не отримує жодних вхідних даних про важливі умови, які визначають сигнал винагороди, він не може змінити свою поведінку, щоб оптимізувати цей сигнал.

Якщо конкретно розглядати завдання поведінки у задачі переслідування, об’єкт, який слід відстежувати, є динамічним. Це означає, що існують різні рухи від цілі, які можуть призвести до втрати цільового об’єкта.

Ми розглянемо два варіанти, за допомогою яких можна покращити здатність агента відстежувати такий динамічний об’єкт.

Динамічні рухи людини можуть призвести до того, що людина буде закрита перешкодами. Такі перешкоди, як кути та стіни, можуть бути

потенційними пастками для дрон. Необхідно, щоб агент мав можливість передбачати рухи цільового об’єкта, щоб змінити своє положення відповідно, щоб не втратити людину з поля зору. У цьому випадку потрібне розуміння спрямованості людини. Є два способи надати агенту цю інформацію.

Найновіші сучасні підходи до цієї проблеми використовують рекурентні нейронні мережі (RNN) як засіб сприйняття агентом певних змін у вхідних даних часових рядів [13,34]. RNN — це нейронні мережі, де вихідний сигнал нейрона або шару використовується як вхідний сигнал для того самого нейрона або шару під час наступного проходу. Мережа здатна збирати шаблони з комбінацій вхідних даних, а не лише з одного входу. У контексті RL це означає, що мережа здатна приймати рішення також з урахуванням попередніх моментів. У деяких програмах це використовувалося як засіб визначення напрямку цільового об’єкта та прийняти рішення щодо дії.

Прикрим недоліком використання RNN є те, що вони є обчислювально більш складними та вичерпними, ніж CNN [35]. Ця перешкода не тільки скорочує час висновку, що може серйозно знизити загальну продуктивність агента, але також скорочує час навчання.

Однак існує простіший, зрозуміліший підхід, який можна використовувати. Замість того, щоб змінювати архітектуру базової нейронної мережі, стан також можна представити як відео або стек кадрів. Подача такого стеку кадрів також може повідомляти про рухи об’єктів у стані. Цей підхід також можна застосувати в RL, оскільки введення відео тривалістю кілька секунд може використовуватися для передачі через мережу.

Деякі проблеми в RL були вирішені за допомогою цього методу [36], і він є попередньою альтернативою використанню більш сучасних підходів [37]. Такі уявлення про стан вимагають мінімальних змін в архітектурі та методології, але при цьому можуть передавати необхідну інформацію агенту.

Разом з отриманням інформації про напрямок людини існує також потреба сприймати навколишні предмети. Щоб виконувати задачі переслідування в середовищах з перешкодами, агент повинен отримати певну інформацію про своє положення щодо об’єктів поблизу.

Є кілька способів передавати дані про відстань дронам [15,37]. Однак, дотримуючись обмежень щодо використання лише вхідних даних камери, використання методів комп’ютерного зору є найбільш очевидним методом вирішення цієї проблеми.

Одним із таких методів є створення карти глибини. Карти глибини створюються шляхом представлення кожного пікселя відстанню від камери до об’єкта, який знаходиться в цьому конкретному пікселі. Отримане зображення є таким, на якому агент може сприймати відстані до всіх об’єктів у своєму полі зору. Чим темніший піксель, тим далі цей об’єкт. Незважаючи на те, що існує кілька технологій, які можна використати для обчислення цих відстаней [38-40], вони все одно мають схожість у представленні цих відстаней у формі карти глибин.

Використання карт глибини як введення стану для агентів RL використовувалось і раніше [9,10,40].

Однак дослідження щодо того, чи дозволяє застосування таких карт глибини в контексті управління безпілотниками агенту краще генералізуватись, досі відсутні.

Як описано раніше, генералізація за допомогою RGB може нести за собою деякі проблеми, які потенційно може вирішити використання карт глибин. Крім того, важливо, які наслідки цієї репрезентації стану в конкретному завданні переслідування.

1.7 Виявлення об’єктів

Спроби вирішення проблеми виявлення об’єктів за допомогою нейронних мереж є темою інтересу вже більше десяти років [41]. Першим найбільшим проривом у цій галузі стало застосування комбінації мереж [41].

Першою частиною цієї комбінації є згортковаа нейронна мережа (CNN), яка є моделлю, яка виконує класифікацію об’єктів, вилучаючи із зображень низькорівневі просторові характеристики та об’єднуючи їх у високорівневі характеристики перед класифікацією зображення на набір класів. Друга частина – мережа регіональних пропозицій, яка пропонує регіони, де можуть бути представлені об’єкти. Завдяки тому, що перша частина мережі виконує регіональні пропозиції та надсилає їх до CNN, мережа R-CNN значно підвищила швидкість і точність моделей виявлення об’єктів. Тим не менш, ця мережа була незабаром вдосконалена шляхом виконання входу до CNN одним проходом вперед для всіх різних запропонованих на попередньому кроці регіонів [42]. Нарешті, поєднання всього процесу забезпечило останній крок у точному та надійному виявленні об’єктів [43].

Однак ці вдосконалення, незважаючи на те, що значно підвищили швидкість і точність моделей, все ж залишили місце для мереж, які можуть пожертвувати частиною точності заради швидкості. Найпомітніша модель, яка змогла це зробити, називається You Only Look Once (YOLO) [44].

Алгоритм YOLO (You Only Look Once) з’явився як новаторський підхід до виявлення об’єктів завдяки механізму виявлення за один прохід. Замість того, щоб сканувати зображення кілька разів, YOLO обробляє його один раз, різко збільшуючи швидкість без значного погіршення точності.

Враховуючи динамічну природу відеозйомки безпілотників, обробка в реальному часі стає обов’язковою, що призводить до популярності YOLO у сценаріях відстеження об’єктів дронами [45].

Ідея полягає в тому, щоб виконати лише один прохід через мережу для кожного зображення та отримати необхідні прогнози. Спочатку зображення ділиться на сітку S × S. Для кожної з комірок сітки модель передбачає N кількість обмежувальних рамок і відповідні показники достовірності. Ці рамки є можливими прямокутниками, які охоплюють об’єкт. Потім для кожної з цих обмежувальних рамок виконується розподіл ймовірностей за можливими класами. З цього набору обмежувальних рамок із відповідними ймовірностями класу зберігаються обмежувальні рамки з найвищою достовірністю, залишаючи позаду набір обмежувальних рамок та їхні відповідні передбачення класу для кожного об’єкта. Виконання виявлення об’єктів за допомогою цієї техніки значно покращує швидкість, мінімально втрачаючи точність. Крім того, завдяки використанню методів оптимізації нейронної мережі ця модель зазнала значних покращень у наступні роки, оскільки були представлені нові її версії [4, 46, 47]. Ці алгоритми YOLO забезпечують набагато вищу швидкість при мінімальному зниженні точності. Поряд із збільшенням обсягу досліджень алгоритмів YOLO, проводилися також дослідження щодо розробки цих моделей для пристроїв з малими обчислювальними ресурсами [5, 6, 48]. Метод, який використовувався в них для зменшення обчислювального навантаження, був або методом скорочення, коли параметри, найменш важливі для кінцевого результату, відкидалися, або шляхом створення подібної мережевої архітектури з меншими параметрами та виконання процесу навчання заново. Тим не менш, вони все ще жертвують значною часткою точності заради швидкості на вбудованих пристроях.

1.8 Розробка середовища для тренування моделі

AirSim, розроблений Microsoft, надає реалістичну платформу моделювання для навчання дронів, автомобілів тощо в насиченому 3D-середовищі. Його головні переваги полягають у високоякісних візуальних ефектах, великому наборі API та можливостях інтеграції, які він пропонує, що робить його поширеним вибором для навчання та тестування алгоритмів ШІ. Забезпечуючи складні сценарії навчання без ризиків і витрат на навчання в реальному світі, AirSim є безцінним інструментом для дослідників і розробників [49]. Однією з ключових пропозицій AirSim є його сумісність із навчанням з підкріпленням. Завдяки його інтеграції з бібліотеками RL розробники схильні використовувати AirSim для навчання моделей безпілотників. Фізика симуляції достатньо наближено відображає умови реального світу, забезпечуючи підготовку навчених моделей до практичного застосування [50].

1.9 Висновки до розділу 1

У цьому розділі було досліджено поточний стан наукових знань щодо відповідних тем цієї роботи, проаналізовано запропоновані в попередніх дослідженнях способи вирішення проблем, що розглядаються, та окреслено відмінність поточної роботи від вже існуючих рішень, обґрунтовано вибір теми дослідження, алгоритмів, що будуть в ньому застосовуватись та технологічного стеку.

РОЗДІЛ 2 ІНСТРУМЕНТИ ДЛЯ ВИКОНАННЯ ЗАДАЧІ ПЕРЕСЛІДУВАННЯ

2.1 Опис задачі переслідування

Завдання полягає у визначенні та слідкуванні за цільовим об'єктом за допомогою безпілотного літального апарату (дрона). Об'єктом переслідування може бути рухомий (людина, транспортний засіб) або статичний (будівля, лісовий масив). У залежності від сценарію застосування, дрон може виконувати завдання в режимах реального часу або відкладеного перегляду.

Історія переслідування об'єктів дроном розпочалася від військових застосувань, де основною метою було визначення і відстеження потенційних загроз. З розвитком технологій та зменшенням вартості дронів, ця задача стала актуальною і для цивільних застосувань, таких як моніторинг дикої природи, нагляд за трафіком або кінооперації.

Для успішного вирішення задачі переслідування об'єктів, дрон повинен мати на борту ефективні системи розпізнавання, навігації та обробки зображень. Ці системи мають забезпечувати швидке визначення цілі, її локалізацію в просторі та визначення траєкторії руху для оптимального слідкування.

Задача переслідування об'єктів дроном має ряд викликів. До них можна віднести: рухливість та непередбачуваність об'єктів переслідування, обмеження в апаратному забезпеченні дрона, нестабільні погодні умови та потенційні перешкоди на шляху дрона.

З розвитком технологій штучного інтелекту та машинного навчання, задача переслідування об'єктів дроном отримує нові можливості для її вирішення. Зокрема, дедалі більше використовуються методи навчання з підкріпленням для автоматичного визначення оптимальних стратегій слідкування.

2.2 Класичні методи

Класичні методи визначаються як ті, які були розроблені та широко використовувалися до введення сучасних методів, базованих на штучному інтелекті та машинному навчанні. Найвідоміші класичні методи вирішення поточної задачі включають в себе перелічені нижче.

Оптичний потік. Метод для визначення руху об'єкта на основі зміни пікселів між послідовними кадрами. Він дозволяє визначити швидкість та напрямок руху об'єкта. Цей метод вимірює зміну інтенсивності пікселів між послідовними кадрами відео для визначення напрямку та швидкості руху. У відстеженні об'єктів, оптичний потік може бути використаний для визначення траєкторії руху об'єкта на основі зміни його позиції на відео в реальному часі. Це особливо корисно для відстеження швидко рухаючихся або непередбачувано змінюючих напрямку об'єктів.

Калмановий фільтр - це рекурсивний алгоритм, який дозволяє оцінювати стан динамічної системи на основі ряду неповних і шумних вимірів. Його ключова ідея полягає в тому, щоб комбінувати прогноз (або прогнозоване значення) з новим вимірюванням, щоб отримати більш точну оцінку стану. У контексті переслідування об'єктів дронами, Калманові фільтри можуть бути використані для прогнозування майбутнього положення об'єкта, враховуючи минулі спостереження та рухову динаміку. Це забезпечує гладке та стабільне слідкування навіть у присутності шуму в даних або тимчасових втрат цілі.

Lucas-Kanade метод. Lucas-Kanade - це диференціальний метод відстеження, який використовує градієнтний підхід для вимірювання руху між двома зображеннями. Основна ідея полягає в тому, що в невеликому вікні вокруг об'єкта, зображення не змінюється значною мірою між кадрами. Цей метод

широко використовується для відстеження руху об'єктів на відео. Він ефективно працює на коротких відстанях і при невеликих змінах вигляду об'єкта, але може зіткнутися з труднощами при значних деформаціях або обертанні об'єкта.

Mean Shift. Mean Shift - це нелінійний метод пошуку моди розподілу ймовірностей. Він працює, аналізуючи плотність точок в просторі особливостей та "зсуваючи" кожну точку до напрямку найвищої збільшення її плотності. У задачах відстеження, Mean Shift може бути використаний для визначення центру мас об'єкта на зображенні. Особливо ефективний для відстеження об'єктів із змінною формою або розміром. Однак, він може бути менш ефективним у випадках з низькою контрастністю або слабко вираженими особливостями об'єкта.

Particle Filter (Фільтр частинок). Particle Filter, також відомий як Sequential Monte Carlo метод, використовує набір частинок (або зразків) для представлення апостеріорного розподілу стану системи. Цей метод полягає у пропонуванні набору можливих рішень, які потім вагуються на основі їх відповідності спостереженням. Частинки з високим ваговим коефіцієнтом відображають більш ймовірні стани системи. У контексті переслідування дронами, фільтр частинок може бути використаний для оцінювання та прогнозування динаміки руху об'єкта, особливо коли динаміка цілі є нелінійною або неоднорідною. Він дозволяє ураховувати різноманітні можливі шляхи руху об'єкта і адаптуватися до непередбачуваних змін у його траєкторії.

Методи на основі моделей: Тут використовуються предварительно визначені моделі об'єктів (наприклад, моделі автомобілів, людей тощо), які потім використовуються для розпізнавання аналогічних об'єктів на відео.

Хоча класичні методи можуть бути досить ефективними в ідеальних умовах, вони часто стикаються з труднощами при роботі в реальних умовах через обмеження в апаратному забезпеченні, змінних погодних умов, а також через непередбачуваність руху об'єкта.

Незважаючи на свої обмеження, класичні методи мають ряд переваг. Вони часто менш обчислювально вимогливі порівняно з сучасними методами і можуть працювати в реальному часі на менш потужному обладнанні. Однак їх недолік полягає в тому, що вони можуть виявитися менш надійними в складних умовах або при перешкодах на шляху дрона.

2.3 Сучасні методи

У контексті розвитку комп'ютерного зору та машинного навчання, сучасні методи для вирішення задачі переслідування об'єктів дроном здобули популярність завдяки своїй ефективності та адаптивності до різних умов.

Конволюційні нейронні мережі (CNN). Ці мережі використовуються для розпізнавання та локалізації об'єктів на зображеннях. Вони можуть бути навчені розпізнавати різні типи об'єктів з великої точністю та навіть в умовах низької освітленості чи перешкод.

R-CNN та їх варіації (Fast R-CNN, Faster R-CNN). Спеціалізовані архітектури для виявлення об'єктів, які комбінують CNN з регіонами інтересу для підвищення швидкості та точності розпізнавання.

YOLO (You Only Look Once). Ще один популярний метод для детекції об'єктів, який розпізнає об'єкти на зображенні за один прохід, що робить його швидким і ефективним для використання в реальному часі.

Q-learning. Цей алгоритм навчає агентів взаємодіяти з навколишнім середовищем, намагаючись максимізувати загальну винагороду. Ідеально підходить для динамічних сценаріїв, де об'єкт постійно змінює своє положення.

Deep Q-Networks (DQN). Комбінація Q-learning з конволюційними нейронними мережами, що дозволяє агентам навчатися в складних середовищах з великою кількістю можливих станів.

Proximal Policy Optimization (PPO). Сучасний алгоритм для навчання з підкріпленням, який оптимізує стратегії агента, намагаючись збалансувати досвід минулого і нові відкриття.

Сучасні методи, особливо ті, які базуються на глибокому навчанні, можуть вимагати значних обчислювальних ресурсів для навчання, але вони зазвичай забезпечують високу точність та робустність в різних умовах. З іншого боку, методи на основі глибокого навчання можуть виявитися чутливими до аномалій або змін у даних, що вимагає регулярного оновлення моделей.

2.4 МППР

Марківський процес – це кортеж (S, A, π, r) , де S – набір усіх станів системи, A – набір можливих дій агента,

$\pi : S \times A \rightarrow \Delta(S)$ функція зміни станів системи,

$r : S \times A \rightarrow R$ функція винагороди.

Тут $\Delta(S)$ набір усіх розподілів ймовірностей на множині S. Ймовірність зміни станів π залежить тільки від поточного стану середовища і поточних дій агентів:

$$\pi(s_{t+1} = s' | (s_\tau, a_\tau, \tau = 0,1,2,\dots,t) = \pi(s_{t+1} = s' | s_\tau, a_\tau)) \quad (1)$$

У кожен момент часу t середовище перебуває у стані $s \in S$ і агент вибирає та реалізує дію $a \in A(s)$. В окремих випадках для спрощення можна прийняти, що $A(s) = A, \forall s \in S$.

На основі вибору $a \in A$ середовище змінює свій стан згідно з розподілом ймовірностей $\pi(s, a)$, і агент отримує випадковий виграш r . Агент взаємодіє з недетермінованим середовищем, модель якого в загальному випадку йому не відома. Агенту доступні для спостереження лише поточні стани середовища та власні стани і стратегії поведінки.

Функція розподілу станів середовища π набуває значення на відрізку [0,1]:

$$\forall s \in S, \forall a \in A \quad \sum_{s' \in S} \pi(s, a, s') = 1 \qquad (2)$$

де s – поточний стан системи, s' – стан у наступний момент часу.

Центральною концепцією марківського прийняття рішень є функція π вибору стратегій, яка описує реальну поведінку агента, тобто спосіб перетворення станів у дії:

$$\pi: S \to A \qquad (3)$$

Функція π визначає імовірності вибору стратегій $a \in A$ агентом у кожному стані середовища:

$$\forall s \in S \quad \sum_{a \in A} \pi(s,a) = 1 \qquad (4)$$

Розподіл π набуває значення на відрізку [0,1] . Якщо $\pi\,(s,a) \in 0{,}1$, то агент здійснює детермінований вибір варіантів рішень.

Метою агента є максимізація функції сумарних виграшів R за рахунок формування ефективної стратегії π :

$$E_\pi[R] \to \max_\pi \qquad (5)$$

Цільова функція очікуваного виграшу (5) залежить від станів середовища, дій агента, функції вибору стратегій та ін., які обумовлюють стохастичну природу системи прийняття рішень. Оскільки поточний виграш є випадковою величиною, то функція очікуваного виграшу формулюється у кумулятивному вигляді, наприклад, сумарної, середньої або сумарної дисконтованої винагороди. Функція сумарної винагороди нагромаджує поточні виграші за обмежену кількість кроків взаємодії агента з середовищем:

$$R_t = \sum_{i=0}^{h} r_{t+i} \qquad (6)$$

де r_t – значення поточних виграшів у момент часу t , h – часовий горизонт.

На відміну від (6), функція середньої винагороди виконує усереднення виграшів у часі:

$$R_t = \frac{1}{h}\sum_{i=0}^{h} r_{t+i} \quad (7)$$

Функції очікуваної винагороди у вигляді (6) та (7) використовується переважно для детермінованих та обмежених за кількістю станів систем.

Критерії оптимальності стохастичних систем, як правило, формулюються на безмежному відрізку часу, що дає змогу отримати стійкі значення характеристик системи прийняття рішень за рахунок згладжування їх випадкових складових. Наприклад, функція (7) може бути визначена так:

$$R_t = \lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i} \qquad (8)$$

Функція дисконтованої винагороди визначається на безмежному відрізку часу, причому поточні виграші зважуються додатними коефіцієнтами $\gamma \in (0,1]$, які визначають співвідношення між поточними та прогнозованими (майбутніми) виграшами:

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad (9)$$

Дисконтування поточних виграшів здійснюється за законом геометричної прогресії і при γ <1 забезпечує швидку стабілізацію функції очікуваного виграшу. Крім того, значення коефіцієнта g визначає характер прийняття рішень агентами. Близьке до 0 значення коефіцієнта надає перевагу виграшам на короткому відрізку часу (за принципом „краще синиця у руці, ніж журавель у небі”). Навпаки, близьке до 1 значення цього коефіцієнта надає перевагу перспективним виграшам на довгому відрізку часу.

З перерахованих цільових функцій найчастіше у самонавчальних системах прийняття рішень використовується функція (9) з дисконтуванням виграшів. Переважно її використання обумовлено результативністю та легкістю

математичних перетворень. Ефективність (вартість) станів системи визначається значенням функції

$$V_\pi(s) = E_\pi[R|s_0 = s] \quad (10)$$

де E_π – очікувана винагорода агента за реалізацію стратегії π , починаючи зі стану середовища s .

Обчислення (10) може бути виконано у рекурсивній формі, відомій у літературі як рівняння Беллмана. Враховуючи (9), після нескладних перетворень отримаємо:

$$V_\pi(s|s_t = s) = E(r_t) + \gamma^k E(r_{t+k+1}) = E(r_t) + \gamma V_\pi(s_{t+1}) =$$

$$= r\big(s, \pi(s)\big) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V_\pi(s') \quad (11)$$

де s – можливі майбутні стани системи.

Метою агента є знаходження функції вибору стратегій π^* , яка максимізує функцію (11) для всіх станів середовища:

$$\forall \pi \forall s \in S \quad V^{\pi*}(s) \geq V^\pi(s) \quad (12)$$

Оскільки вибирають варіанти дій випадково, то для порівняння ефективності дій, коли система перебуває у стані $s \in S$, поточні виграші корисно отримати з (11). Для цього використовується спеціально побудована Q -

функція середніх виграшів, яка визначає ціну дії – сумарний виграш агента, який у стані s вибрав дію a :

$$Q_\pi(s,a) = E_\pi[R|s_0 = s, a_0 = a] \qquad (13)$$

Тут $Q_\pi(s,a)$ є табличною функцією значень варіантів дій a у станах s .

Аналогічно до (8) отримаємо:

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, V_\pi(s') \qquad (14)$$

Дотримання принципу оптимальності Беллмана забезпечує оптимальний виграш агента з досягнутого поточного стану $s \in S$ в усі майбутні моменти часу. Застосування цього принципу для усіх станів забезпечує досягнення глобального оптимального розв’язку.

Для оптимальної функції вибору стратегій π^* для кожного стану $s \in S$ отримаємо:

$$V_{\pi*}(s) = \max_{a \in A}[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, V_{\pi*}(s')] \qquad (15)$$

З (15) можна отримати оптимальну функцію вибору стратегій

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} Q_{\pi*}(s,a) \qquad (16)$$

Оптимізація (16) може бути виконана зокрема методами динамічного програмування.

2.5 Задача переслідування як МППР

Визначення станів (S):

Вхідні дані камери агентів матимуть роздільну здатність 128 x 72. Поруч із цим зображенням буде отримано обмежувальну рамку людини в полі зору дрона. Це надасть дрону інформацію про місцезнаходження та відстань до людини в полі зору камери. Обмежувальну рамку буде отримано за допомогою карт сегментації AirSim.

Зображення саме по собі також може бути картою глибини, і в цьому випадку подальша обробка не виконується (рис. 1.1). Якщо зображення є звичайним RGB-зображенням, воно спочатку буде переведено в градації сірого. Це видаляє три канали зображення RGB, зберігаючи всю необхідну інформацію. Перш ніж передати цей стан агенту для навчання та прийняття рішень, зображення нормалізується, щоб залишатися в діапазоні від 0 до 1. Потім це нормалізоване зображення використовується як вхідні дані для агента, щоб вирішити, яку дію виконати.

Рисунок 2.1 – Карта глибин

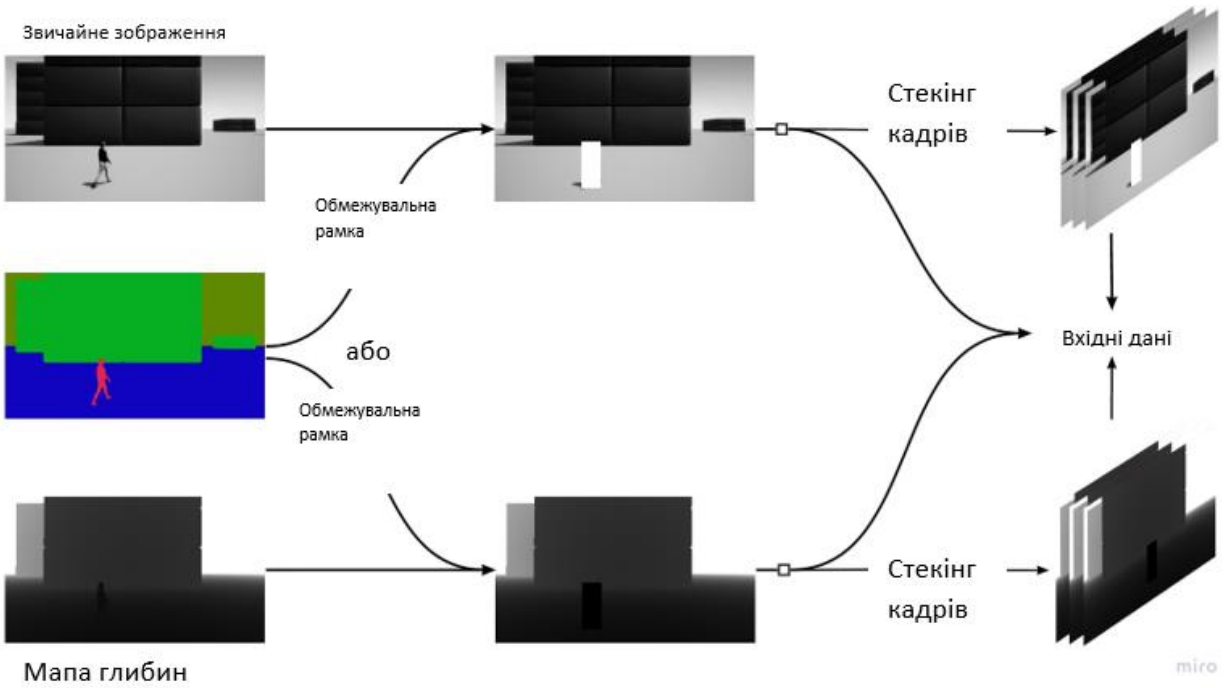

Рис 2.2 - Вхідні дані для визначення стану

Визначення дій (A):

Дії моделі можна визначити наступним шляхом. Вихід моделі буде цілим числом у діапазоні від 0 до 5, кожне з яких представляє дію агента. Ці дії вибрано таким чином, щоб агент міг маневрувати в середовищі в більшості напрямків. Однак така орієнтація можлива лише по горизонтальній осі. По вертикалі дрон залишатиметься на статичній висоті.

Важливо зауважити, що дрон не може рухатися назад. Причина цього полягає в тому, що дрон не може відчути, що відбувається позаду. Тим не менш, рухи вправо і вліво були включені, бо дрон здатний частково спостерігати перешкоди в цих напрямках.

Функція винагороди (R):

Якщо є обмежувальна рамка і зіткнення не відбувається, є три умови, які повинні бути виконані.

Перша з них – це розташування центру обмежувальної рамки. Якщо це значення знаходиться в діапазоні 20% від центру зображення, ця умова виконується.

Друга умова, яка повинна бути виконана, полягає в тому, щоб висота обмежувальної рамки була в межах 30% від висоти зображення.

Останньою умовою, яка має бути виконана, є розташування центру обмежувальної рамки у верхніх 80% зображення. Це гарантує, що дрон не розташований надто близько до людини. Коли всі ці умови виконуються, винагорода дорівнює 1. У разі виявлення зіткнення або відсутності обмежувальної рамки повертається -1. Усі інші випадки повертають 0.

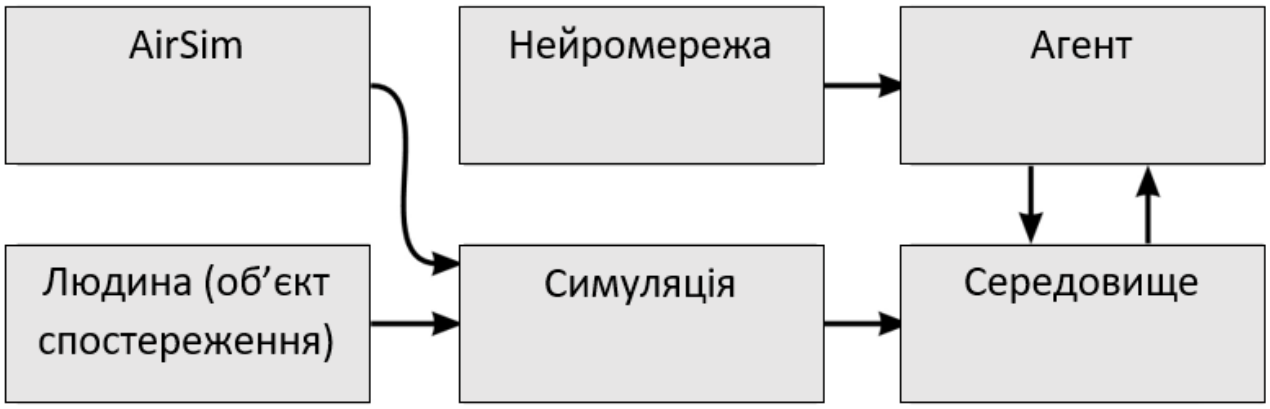

Рисунок 2.3 - Схема роботи системи

2.6 Навчання з підкріпленням

Навчання з підкріпленням – це метод машинного навчання, який базується на винагороді за бажану поведінку та покаранні за небажану. Загалом, агент навчання з підкріпленням – суб’єкт, який навчається – здатний сприймати та інтерпретувати своє оточення, виконувати дії та навчатися шляхом проб і помилок.

Навчання з підкріпленням є одним із кількох підходів, які розробники використовують для навчання систем машинного навчання. Важливість цього підходу полягає в тому, що він дає можливість агенту, будь то функція у відеогрі чи робот у промисловому середовищі, навчитися орієнтуватися в складному середовищі, для якого його було створено. З часом за допомогою системи зворотного зв’язку, яка зазвичай включає винагороди та покарання, агент вчиться у свого оточення та оптимізує свою поведінку.

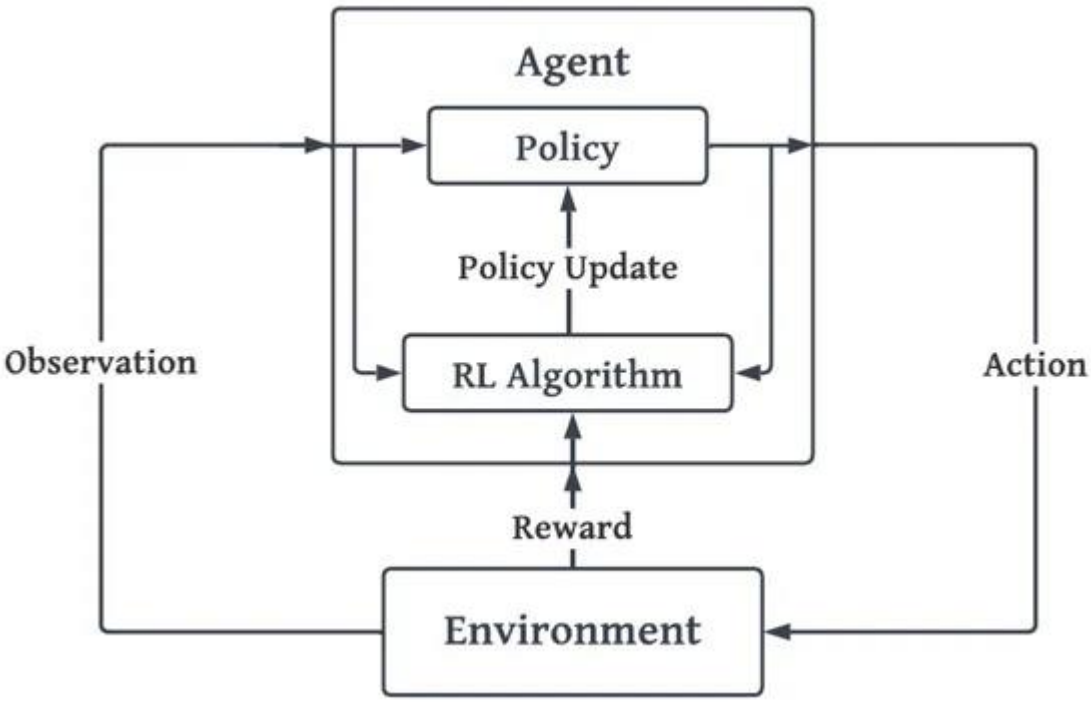

Рисунок 2.4 – Принцип роботи навчання з підкріпленням[1]

Підкріплюючи навчання, розробники винаходять метод винагороди за бажану поведінку та покарання за негативну поведінку. Цей метод присвоює позитивні значення бажаним діям, щоб заохотити агента до їх використання, тоді як негативні значення призначаються небажаній поведінці, щоб перешкодити їм. Це програмує агента на пошук довгострокових і максимальних загальних винагород для досягнення оптимального рішення.

Ці довгострокові цілі допомагають запобігти тому, щоб агент застряг на менш важливих цілях. Цей метод навчання був прийнятий у сфері штучного інтелекту (ШІ) як спосіб керування неконтрольованим машинним навчанням за допомогою винагород або позитивного підкріплення та покарань або негативного підкріплення.

[1] Джерело зображення – https://www.mdpi.com/drones/drones-07-00245/article_deploy/html/images/drones-07-00245-g002-550.jpg

Процес прийняття рішень Маркова служить основою для систем навчання з підкріпленням. У цьому процесі агент існує в певному стані в середовищі; він повинен вибрати найкращу можливу дію з багатьох потенційних дій, які він може виконати в своєму поточному стані. Певні дії пропонують винагороду за мотивацію. У наступному стані йому доступні нові винагородні дії. З часом кумулятивна винагорода – це сума винагород, які агент отримує від дій, які він вирішив виконати.

Навчання з підкріпленням може діяти в ситуації, якщо можна застосувати чітку винагороду. У робототехніці навчання з підкріпленням знайшло свій шлях до обмежених тестів. Цей тип машинного навчання може надати роботам здатність вивчати завдання, які не може продемонструвати вчитель-людина, адаптувати вивчені навички до нового завдання та досягати оптимізації, навіть якщо аналітичне формулювання недоступне.

Навчання з підкріпленням також використовується в дослідженні операцій, теорії інформації, теорії ігор, теорії управління, оптимізації на основі моделювання, багатоагентних системах, інтелекті роїв, статистиці, генетичних алгоритмах і поточних зусиллях промислової автоматизації.

Навчання з підкріпленням, незважаючи на високий потенціал, має певні компроміси. Однією з перешкод для розгортання цього типу машинного навчання є його залежність від дослідження середовища.

Наприклад, якщо ви розгортаєте робота, який покладається на навчання з підкріпленням для навігації в складному фізичному середовищі, він шукатиме нові стани та виконуватиме різні дії під час руху. Однак з таким типом проблеми навчанню з підкріпленням важко послідовно виконувати найкращі дії в реальному середовищі через те, як часто змінюється середовище.

Час, необхідний для забезпечення належного навчання за допомогою цього методу, може обмежити його корисність і бути інтенсивним для обчислювальних

ресурсів. У міру того як навчальне середовище стає складнішим, зростають і вимоги до часу та обчислювальних ресурсів.

Контрольоване навчання може дати компаніям швидші та ефективніші результати, ніж навчання з підкріпленням, якщо доступний належний обсяг даних, оскільки його можна використовувати з меншими ресурсами.

2.6.1 Глибокі Q-мережі (DQN)

Алгоритм Deep Q-network (DQN) — це метод навчання з підкріпленням, який не залежить від моделі. Це один із перших алгоритмів, який включає глибокі нейронні мережі в навчання з підкріпленням. Впровадження глибоких нейронних мереж у навчання з підкріпленням раніше було невдалим через їх нестабільність. Важко узагальнити глибокі нейронні мережі в моделях навчання з підкріпленням, тому що вони схильні до переобладнання. Алгоритми DQN вирішують цю проблему, надаючи різноманітні та декорельовані навчальні дані. Це робиться шляхом зберігання всіх досвідів агента, випадкового відбору та повторного відтворення цих досвідів. В алгоритмі DQN дії вивчаються за допомогою Q-навчання, а функція Q-значення оцінюється за допомогою глибокої нейронної мережі. Q-навчання – це алгоритм RL, який не відповідає політиці. У найосновнішій формі він використовує таблицю для відстеження всіх значень Q для всіх очікуваних комбінацій стану-дій. Рівняння Беллмана використовується для оновлення цієї таблиці, і дія здебільшого вибирається за допомогою є жадібної політики. Зв'язок між певним станом (або парою стан-дія) та його наступником описується рівнянням Беллмана. Рекурсивна форма Q-функції задана як у рівнянні (17).

$$Q_t = r_t + \gamma Q_t + 1, \qquad (17)$$

де

Q_t – Q-значення кроку t

r_t – отримана винагорода

γ – коефіцієнт дисконтування в діапазоні [0,1]

Оптимальне значення Q досягається шляхом вибору дії, яка принесе найбільший прибуток, як показано в рівнянні (17). Оскільки після останнього кроку епізоду більше немає винагород, Q-функція дорівнює останній винагороді, отже, рівняння (16) змінено, щоб скоротити останнє значення Q, і введено позначку, яка називається «виконано». Мета полягає в тому, щоб поступово наблизити поточне значення Q до оптимального значення Q.

$$Q^*(S_t, at) = r(S_t, a_t) + \gamma\ (1 - d) \text{ максимум } a\ Q(S_{t+1}, a_{t+1}), \qquad (18)$$

де $Q^*(S_t, a_t)$ – оптимальне Q-значення стану S_t і дія a_t

$r(S_t, a_t)$ – негайно отримана винагорода

γ – коефіцієнт дисконтування в діапазоні [0,1]

d – позначка завершення, яка дорівнює 1 для останнього кроку, інакше 0

максимум $a\ Q(S_{t+1}, a_{t+1})$ – максимальне значення Q, отримане для стану S_{t+1} за дію a_{t+1}

Архітектура DQN складається з двох нейронних мереж, Q-мережі та цільової мережі, а також компонента відтворення досвіду, як показано на

малюнку 2.7. Q-мережа є навченим агентом для визначення оптимального значення стану та дії. За структурою цільова мережа точно така ж, як і Q-мережа. DQN оцінює цільові та прогнозовані Q-значення, використовуючи дві незалежні мережі з однаковим дизайном, щоб алгоритм Q-навчання був стабільним. Ваги в Q-мережі оновлюються після кожної ітерації, але ваги в цільовій мережі не оновлюються, доки не пройде N ітерацій. Результати цільової мережі використовуються як основна правда для Q-мережі.

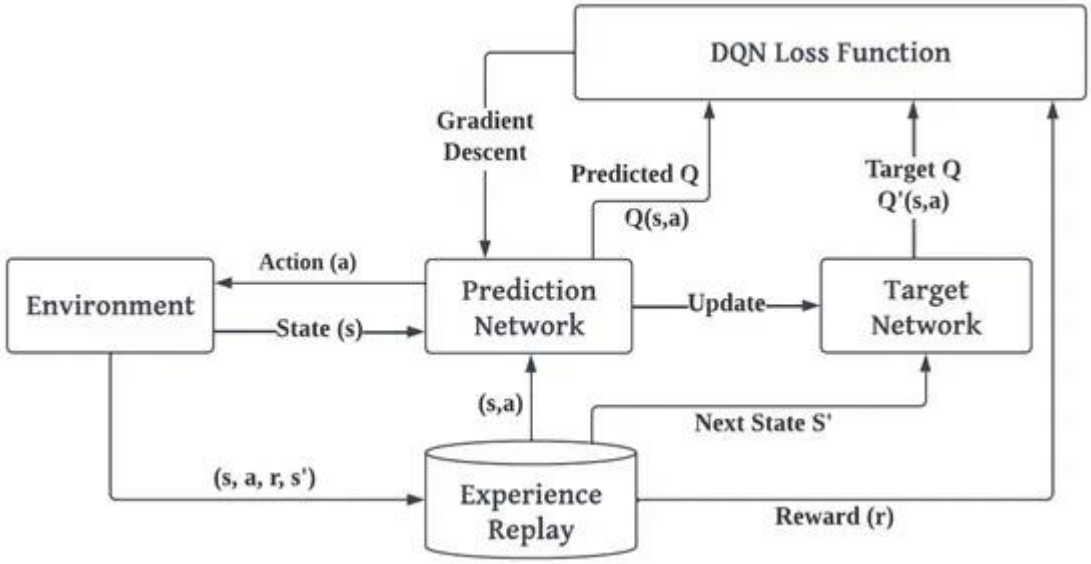

Рисунок 2.5 - Архітектура DQN[2]

Функція втрат обчислює похибку та допомагає оптимізувати вагові коефіцієнти, таким чином мінімізуючи помилку під час процесу навчання. Втрати обчислюються на основі прогнозованого значення Q, цільового значення Q та спостережуваної винагороди для вибірки даних, як зазначено в рівнянні (19).

$$L(\theta) = [Q(S, a;\ \theta) - (r(S, a) + \gamma(1 - d)maxa'\ \hat{Q}(S', a;\ \theta))]^2, \qquad (19)$$

[2] Джерело зображення – https://html.scribdassets.com/23k2vtk0lcbgr6kg/images/11-8bf0a7d20e.jpg

де

$Q(S, a;\ \theta)$ – прогнозоване значення Q

$(r(S, a) + \gamma(1 - d)maxa'\hat{Q}\ (S', a;\ \theta))$ – цільове значення Q

θ – навчальні ваги мережі

Процес Q-навчання коливається або розходиться з процесом нейронної мережі. Оскільки політика змінюється або швидко коливається з незначними змінами Q-значення, розподіл даних може переходити від однієї крайності до іншої. Щоб вирішити вищевказану проблему, переходи зберігаються в буфері відтворення, а невелика вибірка записаного досвіду використовується для оновлення Q-мережі. З відтворенням досвіду послідовні кореляції між зразками будуть порушені, і мережа зможе краще використовувати досвід.

2.6.2 Проксимальна оптимізація політики (PPO)

Метод проксимальної оптимізації політики (PPO) – це підхід до навчання з підкріпленням градієнта політики. Алгоритм PPO натхненний A2C, щоб мати кілька робітників, і TRPO, щоб використовувати регіон довіри для покращення актора. Ключова ідея тут полягає в тому, щоб після оновлення політики нова політика не відрізнялася суттєво від старої політики. PPO використовує відсікання в таких ситуаціях, щоб запобігти надто великим оновленням. Цей алгоритм відбирає дані на основі взаємодії навколишнього середовища, а обрізану сурогатну функцію втрат оптимізують за допомогою стохастичного градієнтного спуску. Простір дії може бути дискретним або безперервним.

Структуру архітектури PPO можна побачити на малюнку 8. Підхід актор-критика використовується агентом, який використовує дві моделі CNN, одну під назвою «Актор» і іншу під назвою «Критик». PPO дотримується підходу до навчання на основі політики. Політика прийняття рішень оновлюється на основі невеликої вибірки досвіду, отриманого під час взаємодії з навколишнім середовищем. Дослідження відкидаються, коли цей пакет досвіду використовується для оновлення політики, а новий набір збирається за допомогою останньої версії політики. Параметр політики оновлюється на основі функції обмежених втрат, наведеної у рівнянні (20). Функція втрат містить гіперпараметр ϵ,

це говорить про те, наскільки переглянута політика має відхилятися від старої політики.

$$L^{clip}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_t}}[\min(\rho(\theta)A^{\pi_{\theta_t}}, clip(\rho(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta_t}})], \quad (20)$$

де

$\rho(\theta)$ – відношення ймовірностей,

$A^{\pi_{\theta_t}}$ – функція переваги,

ϵ – гіперпараметр.

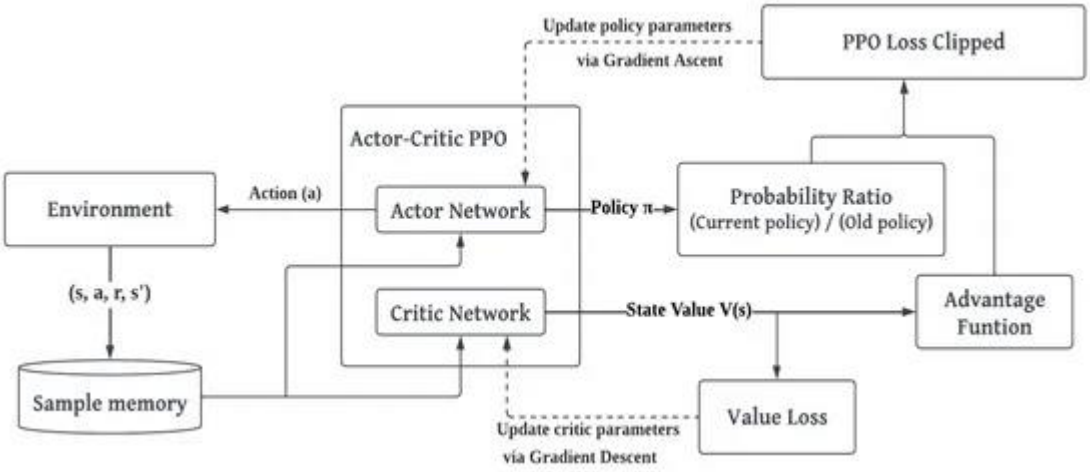

Рисунок 2.6 - Архітектура PPO[3]

Функція обмежених втрат обчислює відношення ймовірностей $\rho(\theta)$ між поточною політикою та старою політикою, як зазначено в рівнянні (21). Коли $\rho(\theta)$>1, дія вибирається більше в поточній політиці та коли $\rho(\theta)$<1, дія вибирається менше в поточній політиці.

$$\rho(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, \quad (21)$$

де

$\pi_\theta(a|s)$ – поточна політика,

$\pi_{\theta_{old}}(a|s)$ – стара політика.

Функція переваги – це різниця між функцією цінності політики та майбутнім дисконтованим підсумком винагород за даний стан і дію, як зазначено в рівнянні (22). Функція переваги вказує, наскільки ефективною буде дія відносно прибутку на основі середньої дії. Коли зміна оцінок градієнта між старою та новою політикою зменшується, алгоритм RL стає більш стабільним.

[3] Джерело зображення – https://html.scribdassets.com/23k2vtk0lcbgr6kg/images/13-687ec35e08.jpg

$$A(S_t) = r_t + \gamma\, V(S_{t+1}) - V(S_t), \qquad (22)$$

де

$V(S_t)$ – функція значення стану для стану S_t,

r_t – винагорода за крок t,

γ – дисконтний коефіцієнт.

Станово-ціннісна функція оцінює, наскільки сприятливі обставини в державі. Рівняння функції стану-значення подано як рівняння (23).

$$V_{\emptyset}(S_t) = \mathbb{E}_{s,a\sim\pi_{\emptyset}}\left[Q(S_t, a;\ \emptyset)\right], \qquad (23)$$

де $Q(S_t, a;\ \emptyset)$ – Q-значення стану S_t і дія a для параметра $\emptyset$.

Функція переваги вказує, кращий чи гірший стан, ніж очікувалося. Перевага є позитивною, коли поточна дія є більш сприятливою, ніж попередня. За порогом $1+\epsilon$, поточний план дій стає надто сприятливим, а врахування такої великої кількості змін збільшує дисперсію та зменшує послідовність. Щоб запобігти надмірному оновленню функції політики, якщо дія надто успішна, обрізана версія графіка має рівну лінію, коли функція співвідношення збільшується більше, ніж $1+\epsilon$. Коли дія призводить до негативної переваги, це означає, що поточна дія не набагато краща за попередню, і що вона намагатиметься знизити функцію відношення нижче 1. Однак занадто значне зниження функції відношення призведе до значної зміни , таким чином, він обрізається, щоб бути вищим за $1-\epsilon$. Функція втрат критичної мережі визначається як середній квадрат втрат із віддачами, як наведено в рівнянні (24).

$$L^{VF}(\emptyset) = \mathbb{E}_{s,a\sim\pi_\emptyset}\,[V_\emptyset(S_t) - \widehat{R_t}]^2, \qquad (24)$$

де

$V(S_t)$ – функція значення стану критичної мережі

$\widehat{R_t}$ – дисконтована сума майбутніх винагород у траєкторії, наведеній у рівнянні (25)

$$\widehat{R_t} = \sum_{i-t}^{T} \gamma^{i-t} r'_i, \qquad (25)$$

де

γ^{i-t} – коефіцієнт знижки за крок $i{-}t$

r_i – винагорода за крок i

Висновки до розділу 2

У даному розділі було розглянути теоретичний матеріал, необхідний для проведення поточного дослідження. Розглянуто класичні та нові алгоритми, застосовні в контексті задачі переслідування, основні концепції навчання з підкріпленням, а також загальну архітектуру алгоритмів, що використовуються у цій дисертації. Формалізовано задачу переслідування як марківський процес підтримки прийняття рішень.

РОЗДІЛ 3 ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ НАВЧАННЯ З ПІДКРІПЛЕННЯС У ЗАДАЧІ ПЕРЕСЛІДУВАННЯ ОБ'ЄКТІВ

3.1 Введення

У цьому розділі буде більш предметно обговорено дослідницьку спрямованість цієї дипломної роботи разом із підпроблемами, які розділятимуть основну проблему.

Спрямованість. Оскільки цільовий об'єкт є динамічним рухомим об'єктом, необхідна реалізація певного типу інформації про спрямованість у представленні стану. У цій дисертації було прийнято рішення застосувати стек зображень як засіб передачі цієї інформації. Це може покращити процес тренування, швидкість і конвергенцію, а також може змінити спосіб, у який виконується переслідування. З літератури все ще незрозуміло, чи отримає агент користь від додавання цієї інформації до вхідних. Щоб дослідити це, реалізацію агента, навченого представленню стану, що містить спрямованість, буде перевірено, щоб відповісти на запитання «Чи покращує реалізація складених зображень як репрезентації стану навчання агента RL ефективність переслідування»?

Уникнення перешкод. Бажаною поведінкою було б не лише стежити за людиною, але й успішно розпізнавати та уникати об'єктів, які стоять на шляху. Таким чином, ми запровадимо такий тип представництва стану, який може передати агенту RL інформацію про те, де знаходяться потенційні перешкоди. Для цього подання стану буде змінено зі звичайних зображень на карти глибини як реалізацію цього зондування об'єкта. Реалізація цього дозволяє нам побачити, чи впливає це на процес навчання чи навчену поведінку, і допоможе відповісти на друге запитання дослідження. Чи покращує реалізація карт глибин як представлення стану продуктивність агента RL у контексті поточної задачі?

Порівняння з базовою моделлю. Ще один аспект, який буде досліджено, це переваги використання RL порівняно з базовою моделлю. Загалом, мета полягає в тому, щоб побачити, чи працює навчання з підкріпленням у контексті задачі

переслідування. Цікавий підхід до відповіді на це запитання полягає в тому, щоб побачити, як працює найкращий робочий агент RL порівняно із попередньо запрограмованою базовою моделлю, подібною до методів, які використовувалися в попередніх дослідженнях. Чи дає використання RL якусь перевагу порівняно з базовим агентом?

Узагальнюваність. Нарешті, щоб побачити, чи здатний агент узагальнити свою вивчену поведінку для нових, складніших середовищ, агенти будуть перевірені в ситуаціях, які раніше не бачили. Навчаючи агентів у середовищах, де були додані систематичні перешкоди, і, отже, тестуючи їх у складніших середовищах, можна зробити висновки про те, яка частина поведінки з попереднього середовища передається. Чи агенти RL узагальнюють свою поведінку в невидимому та складнішому середовищі?

Методи. Буде обговорено програмне забезпечення, яке було використано для побудови основи для RL-фреймворку. Крім того, будуть пояснені показники та експерименти.

Архітектура. Буде розглянуто кожну частину архітектури, яка використовуватиметься для навчання та тестування агента. Огляд можна побачити на малюнку.

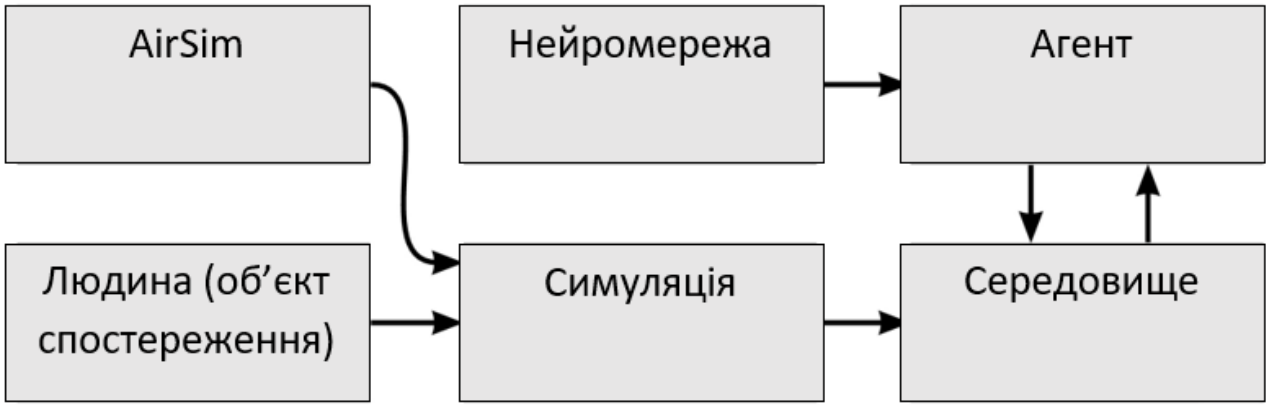

Рисунок 3.1 – Схема роботи

Симуляція. Найважливішим аспектом фреймворку та архітектури є система моделювання, в якій працюватиме дрон. Як показано на малюнку, симуляція є найважливішим аспектом, у якому працює структура RL. Симуляція — це комбінація кількох аспектів, які будуть обговорюватися далі, а саме особи, AirSim і фізичного механізму, що визначає фізичне середовище дрона. Кожен аспект середовища моделювання буде розглянуто далі.

AirSim. Програмою, яка використовуватиметься для керування дроном, буде AirSim від Microsoft. AirSim — це бібліотека, яка використовується для зв'язку з дронами, створеними в симуляційних середовищах, або фізичними дронами. Окрім цього, програма дозволяє користувачеві створювати екземпляр квадрокоптера безпосередньо у віртуальному середовищі разом із імітованими можливостями зору, включаючи звичайну камеру та огляд у глибину. Це можна побачити внизу малюнку 3.2. Крім того, за допомогою Unreal Engine ця програма дозволяє використовувати безліч створених середовищ. Також AirSim дозволяє легко підключатися до PixHaw API, що полегшує впровадження дрона на фактичний фізичний дрон, дозволяючи розробку для реальних застосувань у зовнішньому світі.

Рисунок 3.2 – Приклад середовища

Середовища. Іншим аспектом симуляції є фізичне середовище, в якому літатиме дрон. Для цього створено три середовища. Набір агентів буде навчено та перевірено в кожному середовищі, яке буде зазначено в розділі з експериментами. Знімки середовищ з виду зверху і траекторію цільових об'єктів в них видно на малюнку 3.3.

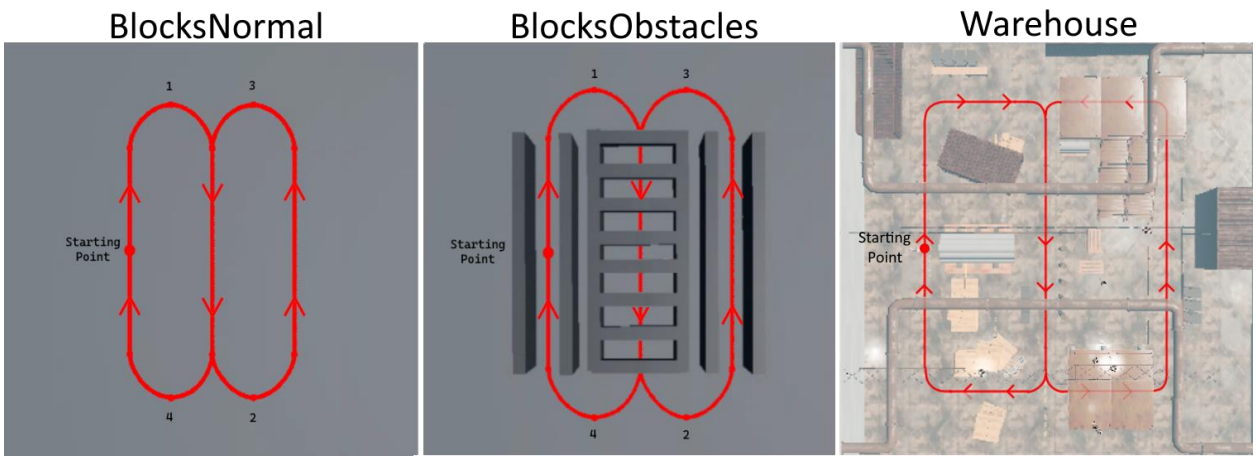

Рисунок 3.3 – Шляхи людини в різних середовищах

Форма шляху людини всередині середовища залишається постійною змінною. Цю форму було обрано, щоб збалансувати кількість часу, протягом якого людина буде йти прямо та робити повороти. Збереження пішохідних маршрутів незмінними виключає цю змінну як можливе пояснення того, чому певні моделі можуть бути менш ефективними в певних тестах.

Люди, за якими стежитиме дрон, можуть відрізнятися залежно від середовища. Як стане зрозуміло далі, стани введення не міститимуть жодної інформації про конкретну особу, за якою стежать. З цієї причини під час навчання та тестування може бути реалізована інша цільова особа. Тим не менш, у будь-який момент часу в оточенні буде щонайбільше одна людина, за якою безпілотник повинен буде стежити. Знову ж таки, щоб виокремити необхідне

завдання агента для стеження за окремою особою, було прийнято рішення не включати кількох людей.

Дивлячись на реалізовані середовища, першим середовищем буде BlocksNormal, яке не буде містити перешкод для агента. У цій ситуації буде оцінено здатність агента навчитися цільовій поведінці. Другим буде BlocksObstacles, у якому додано різні типи перешкод для імітації трьох різних ситуацій, а саме: тісних коридорів, широких коридорів і кутів. Їх розташування можна побачити на малюнку 3.5. У цьому середовищі буде оцінено здатність агентів справлятися з такими ситуаціями. Нарешті, середовище Warehouse міститиме подібні ситуації, але з набагато більшою кількістю деталей. Тут перешкоди складаються з різних типів об’єктів і текстур, але створюють такі ж перешкоди, яких дрон повинен уникати, як і в середовищі BlocksObstacles. Таким чином поведінку можна проаналізувати, дивлячись на те, як агент справляється з кожною конкретною ситуацією, щоб побачити, чи здатні агенти узагальнювати свою поведінку на нові ситуації.

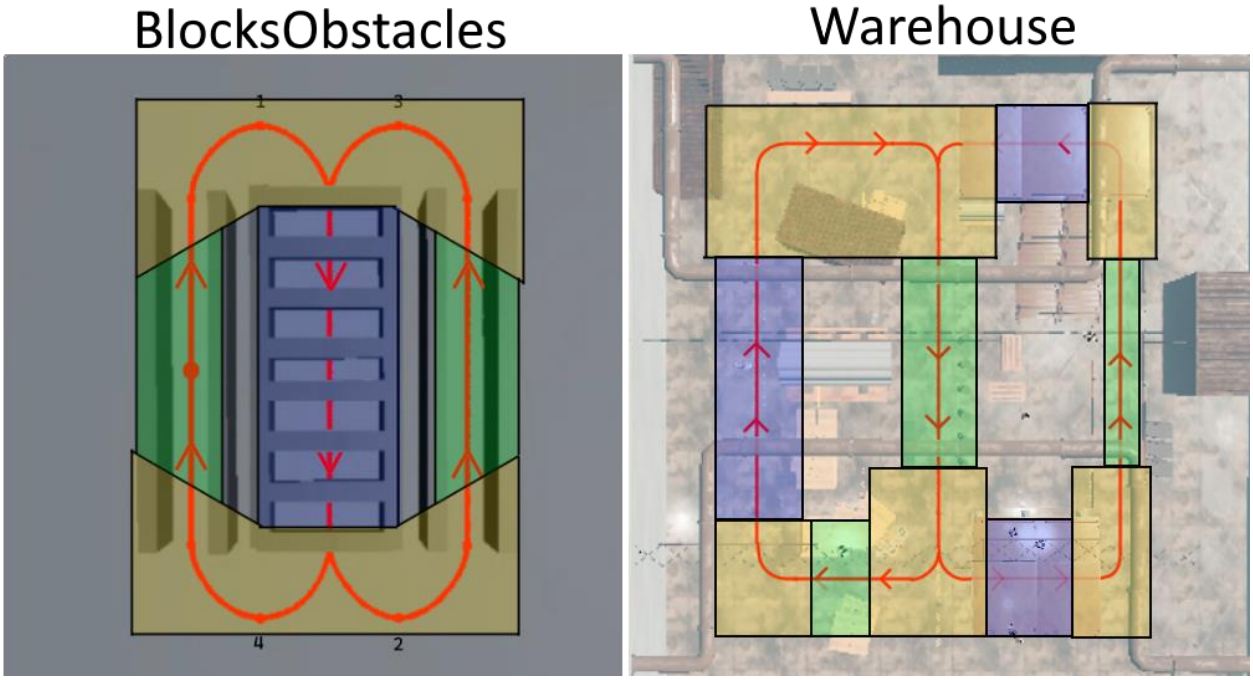

Рисунок 3.4 – Схема зон у середовищах

Структура агента. У цій дипломній роботі DQN та PPO буде реалізовано як агенти, які виконуватимуть навчання.

Вхідні дані. Вхідні дані камери агентів матимуть роздільну здатність 128 x 72. Поруч із цим зображенням буде отримано обмежувальну рамку людини в поданні дрона. Це надасть дрону інформацію про місцезнаходження та відстань до людини в полі зору камери. Обмежувальну рамку буде отримано за допомогою карт сегментації AirSim. Ці карти забезпечують те саме зображення, що й вид з камери транспортного засобу, але натомість кожен піксель відображає, який об’єкт видно. Приклад однієї з таких карт сегментації можна побачити на малюнку 3.6. Колір у цій карті сегментації для людини попередньо встановлюється під час ініціалізації всієї програми, і коли дрон отримує цей кадр, він шукає пікселі, що відповідають цьому кольору. Коли це доступно, простий вибір крайніх значень на обох осях надасть агенту обмежувальну рамку людини. Тоді всі пікселі в цій області обмежувальної рамки отримають значення -1, а решта зображення буде оброблено відповідно до обраного типу комбінації представлення стану. Ці зображення можуть бути складені, що призведе до повторення цього процесу тричі. Саме зображення також може бути картою глибини, і в цьому випадку подальша обробка не виконується. Якщо зображення є звичайним RGB-зображенням, воно спочатку має градації сірого. Це видаляє три канали зображення RGB, зберігаючи всю необхідну інформацію. Перш ніж передати цей стан агенту для навчання та прийняття рішень, зображення нормалізується, щоб залишатися в діапазоні від 0 до 1. Важливо зауважити, що діапазон обмежувальної рамки на зображенні залишатиметься -1 , завдяки чому можливі значення для кожного вхідного сигналу для агента коливаються від -1 до 1. Потім це нормалізоване зображення використовується як вхідні дані для агента, щоб вирішити, яку дію виконати.

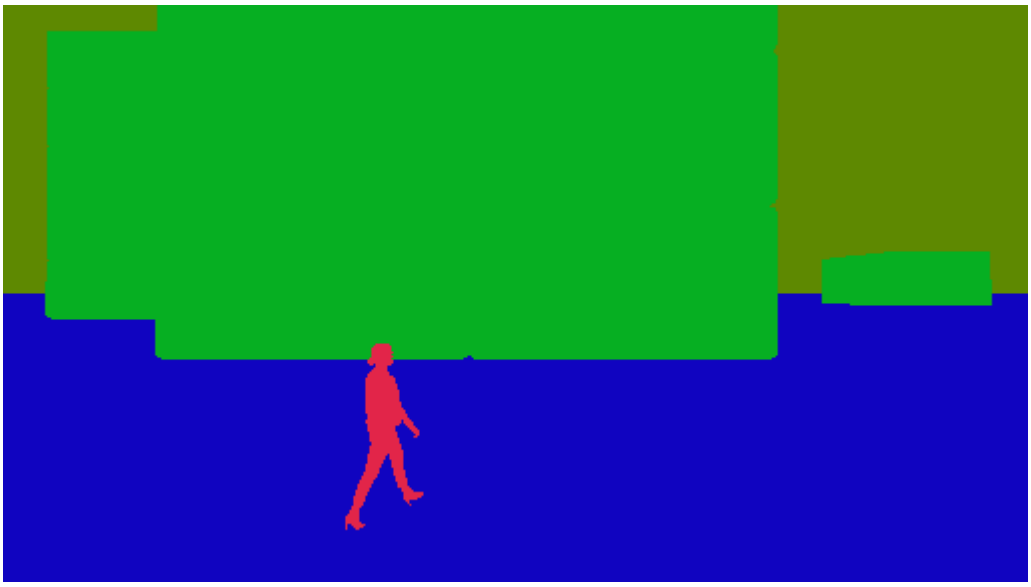

Рисунок 3.6 – Мапа сегментації

Мережа та гіперпараметри. Кожен тип агента буде варіацією агента DQN і, отже, міститиме ту саму мережу. Цю мережу можна побачити на малюнку. Будучи досить скромним за розміром, ця нейронна мережа має приблизно 2 млн параметрів, що робить її обчислювально легкою для навчання та розгортання. Цю архітектуру було обрано через те, що потрібна згорткова мережа, оскільки вона має обробляти дані зображення. Буде використано оптимізатор середньоквадратичного поширення (RMSProp) зі швидкістю навчання 0,001.

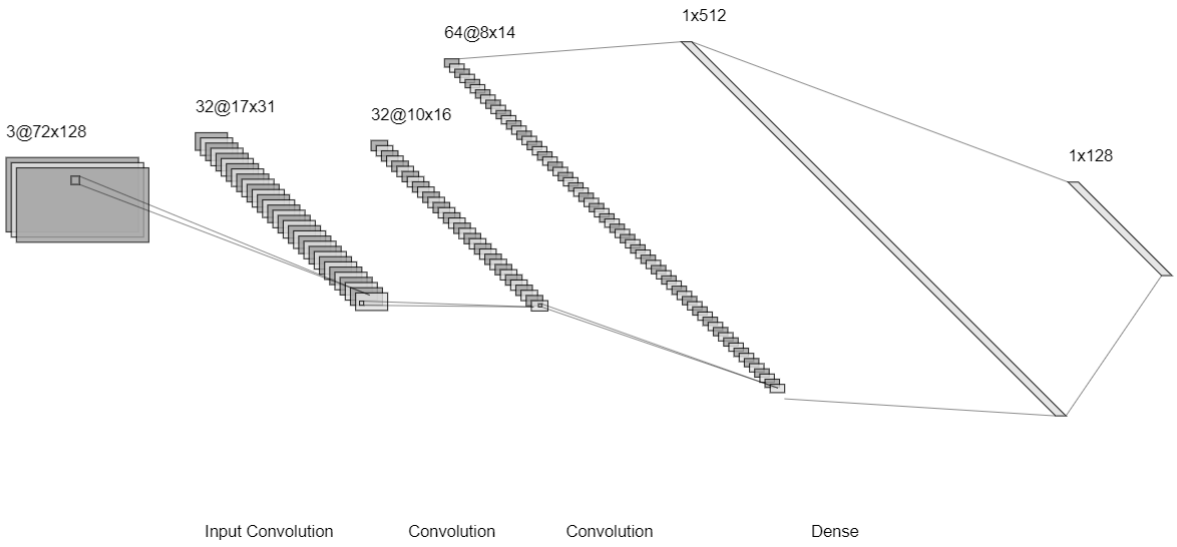

Рисунок 3.7 – Архітектура мережі DQN

Вихідні дані. Вихідні дані DQN будуть цілими числами в діапазоні від 0 до 5, кожне з яких представлятиме певну дію. Ці дії вибрано таким чином, щоб агент міг маневрувати в середовищі в більшості напрямків. Здатність орієнтуватися є необхідною умовою для того, щоб змінити свій напрямок. Однак така орієнтація можлива лише по горизонтальній осі. По вертикалі дрон залишатиметься на статичній висоті. Його вертикальна орієнтація не може змінитися, оскільки ці рухи також призведуть до горизонтального зміщення. Зміна положення камери є синонімом переміщення самого дрона. Важливо зауважити, що дрон не може рухатися назад. Причина полягає в тому, що дрон не може бачити, що відбувається позаду. Тим не менш, рухи вправо і вліво були включені, оскільки дрон може частково спостерігати за перешкодами в цих напрямках.

Базова модель. По-перше, буде реалізовано базову модель, яка не використовує методи RL, а більш прямі евристичні методи для прийняття рішення щодо дії. Метод, використаний для цього, був натхненний технікою, яка використовується в керуванні дронами, коли RL не використовується. У цих методах використовується просте припущення, що дрон стежить за об’єктом, коли виконується ряд умов. Ці умови включають те, що об’єкт знаходиться в

центрі зору камери та що розмір об’єкта відповідає певній пропорції. За цими умовами можна визначити відстань до об’єкта та визначити, чи дивиться на нього дрон.

Використовуючи ці принципи, було розроблено наступний агент. За допомогою отриманих вхідних даних камери базовий агент визначає, де знаходиться обмежувальна рамка на зображенні. Якщо центр цієї обмежувальної рамки знаходиться на лівому боці, агент обертатиметься ліворуч. Якщо обмежувальна рамка знаходиться в правій частині перегляду, агент повертатиметься праворуч. Якщо центр обмежувальної рамки знаходиться в межах діапазону від центру огляду, агент перевірить висоту обмежувальної рамки, щоб побачити, наскільки вона відрізняється від висоти цілі. Якщо висота цілі становить 20% від висоти зображення, буде дозволена додаткова похибка, щоб запобігти постійним рухам агента. У випадку, якщо висота обмежувальної рамки знаходиться в межах запасу 25%, агент не рухатиметься. В іншому випадку агент буде рухатися вперед, наближаючись до людини.

Використання цих методів усуває необхідність виконувати розрахунки щодо точного місцезнаходження людини, а також підтримує достатню відстань від людини. Ці значення були налаштовані за допомогою функції винагороди, щоб також максимізувати функцію винагороди за допомогою цього методу. Цей агент не потребує навчання, тому він просто використовуватиметься як базова модель для порівняння з моделями RL.

Варіанти представлення стану. Агент RL, який буде використано, буде DQN. Однак його представлення стану може змінюватися, і вони будуть перевірені відповідно. У цій дипломній роботі буде реалізовано та досліджено два таких варіанти.

Першим варіантом представлення стану, який можна зробити, є використання стекування. Враховуючи обмеження цього дослідження щодо підтримки агента з низькими обчислювальними ресурсами, було прийнято

рішення вибрати відеовхід як засіб для передачі агенту спрямованості. Використання RNN потребувало б занадто великої обчислювальної потужності. Це було реалізовано шляхом отримання трьох послідовних кадрів із середовища з інтервалом 0,1 секунди як засіб для формування відео. Це відео також можна вважати складеним зображенням останніх трьох кадрів. Це складене зображення можна створити шляхом отримання кадру від AirSim (звичайного або глибокого), отримання з нього обмежувальної рамки, обробки, а потім, нарешті, простого складання їх у тривимірне зображення, як можна побачити на малюнку 3.8.

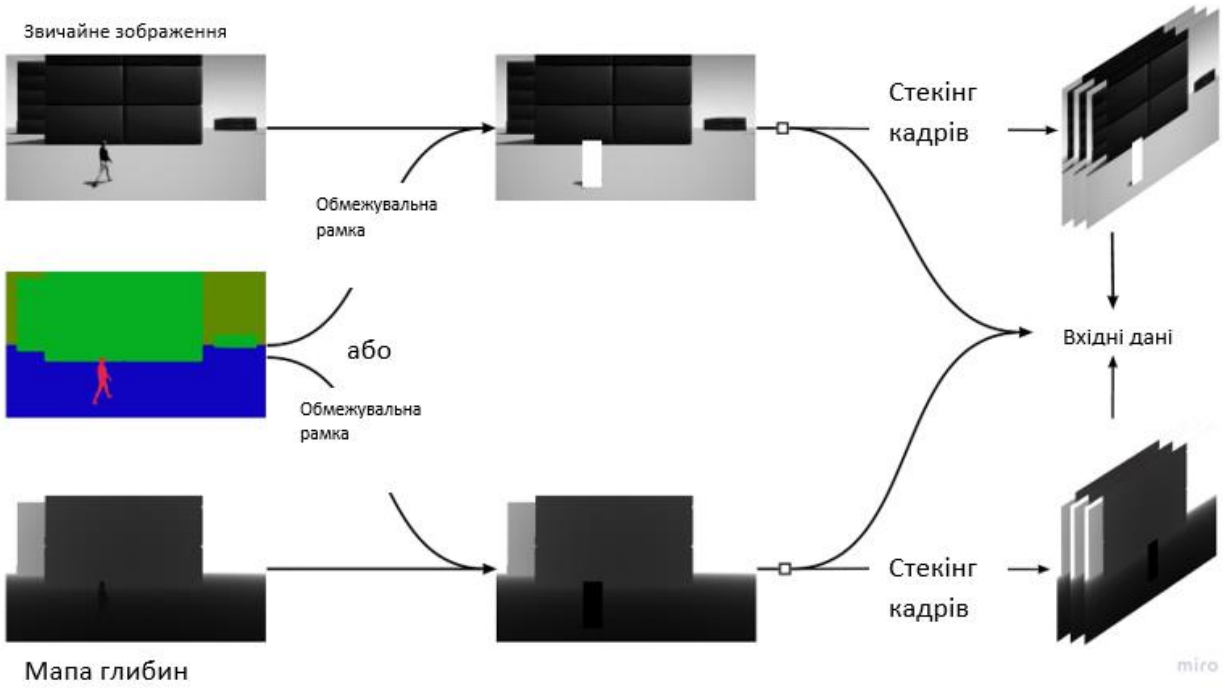

Рисунок 3.8 – схема роботи алгоритму представлення станів

Іншим варіантом є використання зображень глибини. Щоб дозволити агенту відчувати перешкоди в його оточенні, було обрано карти глибини. Ця опція поєднує в собі здатність агента отримувати вхідні зображення, сприймати людину та визначати відстань до перешкод навколо неї. AirSim забезпечує

легкість простого запиту карт глибини в навколишньому середовищі, надаючи агенту доступ до правдивих відстаней до всього оточення. Важливо відзначити, що ці карти глибини можна легко поєднати з їх стеком, дозволяючи перевірити всі можливі комбінації цих агентів.

RL-фреймворк. Структура навчання з підкріпленням також є важливою частиною реалізації, яка потребує деякого обговорення. Було вибрано використання бібліотеки TF-Agents, яка дозволяє високорівневу абстракцію реалізацій RL. Тим не менш, необхідно розробити більш конкретні аспекти середовища RL, кожен з яких буде детально розглянуто в наступних розділах.

Середовище RL. Важливим аспектом навчання з підкріпленням є середовище RL. Тут ми маємо на увазі не змодельоване середовище, у якому летить безпілотник, а середовище RL, яке створює стани та повертає винагороди, якщо це необхідно, як показано на малюнку 3.1. Агент взаємодіє з середовищем RL, яке взаємодіє з симульованим середовищем, щоб отримати необхідну інформацію. Важливо те, що алгоритми RL працюють в епізодах. Епізод характеризується початковим станом і кінцевим станом із переходами між кроками, які виконує агент. Після цього термінального стану середовище скидається та починається новий епізод.

У цій реалізації вихідний стан людини знаходиться безпосередньо позаду людини на невеликій відстані. Дотримання початкової дистанції від людини усуває упередженість у винагороді на ранніх етапах епізоду. Після цього агент робить крок.

Цей поетапний процес визначається наступним чином. Спочатку отримується стан, який виконується. Отже, дія вибирається агентом відповідно до цього стану, після чого розраховується винагорода для цього нового стану. Нарешті, на кожному кроці буде зроблено оцінку того, чи завершився епізод, щоб визначити, чи досягнуто термінального стану. Це буде зроблено шляхом перевірки, чи була виконана одна з наступних трьох вимог:

- агент зіткнувся з об’єктом;
- агент не має обмежувальної рамки в камері, тобто особа була втрачена з поля зору;
- зроблено понад 50 кроків.

Ці умови гарантують, що епізод складається з кінцевої послідовності дій, з яких DQN може навчатися. Між цими епізодами середовище перезавантажується. Це скидання гарантує, що дрон повертається в правильне положення. У цьому випадку безпілотник розташовується безпосередньо позаду людини, і він також орієнтується в напрямку людини. Ці скидання призначені для того, щоб не витрачати час на простори станів під час навчання, що не сприяють навчанню агента. Стани, коли він зіткнувся або втрачає людину з поля зору, не мають значення для дрона у навчанні успішно слідувати. Таким чином, до того, як буде втрачено надто багато часу в цих станах, середовище скидається до моменту, з якого може успішно продовжуватись процес навчання.

Навчання. Перш ніж моделі зможуть почати навчання, виконуються деякі підготовчі кроки. Для DQN потрібен буфер відтворення, у якому зберігається великий набір даних досвіду. Це необхідно для DQN, оскільки для кожного кроку навчання потрібні зразки з цього буфера як вхідні дані для мережі. Оскільки це також призведе до зміщення основних етапів навчання, необхідно заповнити частину буфера відтворення перед початком навчання. Таким чином, перед початком навчання агент, який виконує випадкові дії, переміщається у світі протягом 500 кроків, заповнюючи частину буфера відтворення, розмір якого становить 10 000 досвідів. Це гарантує, що початкова частина простору станів досліджується ще до початку навчання.

Далі процес навчання можна описати як цикл, у якому щоразу виконуються ті самі кроки, що також називається епохою. Цей цикл починається з виконання 50 рухів у середовищі. Ці 50 рухів відповідають повному епізоду до скидання середовища. Деякі з попередніх епізодів можуть не досягти ліміту в 50

кроків, однак, незважаючи на це, загалом буде зроблено 50 кроків, перш ніж мережа буде навчена цьому досвіду. Це стабілізує збільшення розміру буфера відтворення протягом кожної епохи. Розмір партії вибірки становить 64, і кожна модель буде навчена на 2000 епох або до сходимості.

Важливо зазначити, що бібліотека TF-agents підтримує дві політики для агента. Політика One collect, яка має на меті завжди зберігати певний ступінь випадковості, щоб завжди підтримувати певний рівень дослідження. Поряд із цим є політика оцінки, яка є оптимальною політикою, яку засвоїв агент. Кожні 50 ітерацій виконується крок оцінки. Тут 10 епізодів розігрується політикою оцінки. Необхідні показники, як обговорюватиметься пізніше, зберігаються, і цикл навчання продовжується.

Функція винагороди. Важливим аспектом RL є функція винагороди. Оскільки навчання в цьому контексті залежить від максимізації функції винагороди, поведінка, якої навчиться агент, сильно залежить від цієї функції. У цій дипломній роботі вибір був зроблений у бік розрідженої функції винагороди. Це пов’язано з тим, що, як згадувалося раніше, немає необхідності давати агенту заздалегідь визначені знання про те, як досягти цілей. Це надасть агенту свободу тлумачення того, як вирішити проблему.

Обмежувальна рамка буде використовуватися як об’єкт для визначення винагороди, оскільки вона містить як інформацію про відносне розташування, так і відстань до людини. Тут будуть зроблені припущення щодо відносного розташування людини, а також її відстані від дрона. Цільова відстань до людини була визначена як чотири метри, що в поєднанні з висотою дрона призводить до того, що людина займає 30% висоти зображення. Поєднання умов центрування та відстані у функції винагороди призводить до наступного набору правил.

Якщо є обмежувальна рамка, а зіткнення не відбувається, є три умови, які повинні бути виконані.

Перша з них – це розташування центру обмежувальної рамки. Якщо це значення знаходиться в діапазоні 20% від центру зображення, ця умова виконується.

Друга умова, яка повинна бути виконана, полягає в тому, щоб висота обмежувальної рамки була в межах 30% від висоти зображення.

Останньою умовою, яка має бути виконана, є розташування значення у центру обмежувальної рамки у верхніх 80% зображення. Це гарантує, що дрон не розташований надто близько до людини.

Коли всі ці умови виконуються, винагорода дорівнює 1. У разі виявлення зіткнення або відсутності обмежувальної рамки повертається -1. Усі інші випадки повертають 0.

У всіх тестах функція винагороди залишається постійною, щоб використовуватись як показник ефективності кожного агента. Таким чином, усі тести можна порівняти відповідно до отриманої винагороди.

Метрики та методи аналізу. Загальним вирішальним показником, який буде використовуватися в цій дипломній роботі, є середня винагорода. У цьому показнику фіксується середня винагорода, отримана в епізоді. Цей показник використовуватиметься для оцінки як процедури навчання, так і загальної продуктивності агента, оскільки ця метрика охоплює всі вимоги до поведінки агентів, а саме уникнення перешкод, центрування людини та дотримання дистанції.

Метрики, специфічні для процесу навчання, наступні.

По-перше, поряд із середньою винагородою відстежуватиметься середня тривалість епізоду. Оскільки епізод може закінчитися раніше, коли сталася аварія або людина зникла з поля зору, чим довше триває епізод, тим краще дрон стежить за людиною. Обмеження тут становить 50 кроків, оскільки тоді епізод скидається незалежно від успішності агента.

Під час етапу оцінювання ці показники змінюються. Замість середньої тривалості епізоду та втрати фіксується мінімальна та максимальна винагорода. Вони виражають найгірший і найкращий епізод, які виконала модель. Бажана ситуація, коли діапазон між цими двома значеннями не надто великий. Однак, якщо це не так, перегляд цих крайнощів може пояснити, чого все ще бракує моделі.

Що стосується кількісної оцінки поведінки агента, для отримання поглибленої інформації було використано ряд методів аналізу. По-перше, для кожного з 50 можливих часових кроків, які може тривати епізод, буде записано середню отриману винагороду. Ця інформація дає уявлення про те, як просувається середній епізод для агентів, і вказує на потенційні вузькі місця агента, оскільки демонструє розподіл отриманої винагороди за середній епізод. Крім цього, було записано шляхи, які пройшов дрон під час тестового запуску, а також місце та тип закінчення епізоду. Ця інформація надає дані про поведінку агента та про те, в яких ситуаціях йому найважче.

Експерименти. Для того, щоб відповісти на дослідницькі запитання цієї дисертації, буде запущено серію експериментів. Експерименти проводитимуться для кожного середовища, складність яких зростатиме. Спочатку тести будуть запущені в BlocksNormal, після чого BlocksObstacles буде використано для запуску тестів. Нарешті, тести будуть запущені в Warehouse.

Спочатку агенти пройдуть навчання та перевірку в середовищі BlocksNormal. В першу чергу, буде навчено DQN з одиночними звичайними зображеннями та DQN із складеними нормальними зображеннями. Після цього їх буде порівняно. Найкраще робоче представлення стану буде використано для навчання кінцевого агента за допомогою глибинних зображень. Це означає, що DQN буде навчено та перевірено або на одному зображенні глибини, або на складеному зображенні глибини. Нарешті, базова модель також буде перевірена та використана для порівняння. Після виконання всіх цих тестів буде

проаналізовано поведінку разом із загальною ефективністю. Це означає, що кожен із агентів порівнюватиметься з базовою моделлю, щоб кількісно визначити, наскільки краще працюють агенти RL.

Потім буде виконано навчання в середовищі BlocksObstacles. Продуктивність не буде просто порівнюватися з базовою моделлю. Після двох тестів у середовищах BlocksNormal і BlocksObstacle очікується, що продуктивність знизиться. Ця частка погіршення також буде використана для порівняння зниження продуктивності кожного агента RL. Виконання цього порівняння дасть уявлення про те, чи агенти порівняно краще справляються з складнішим середовищем, ніж базова модель. Нарешті, найкращий робочий агент у цьому середовищі буде перенавчений за допомогою дещо зміненої функції винагороди. Ця функція винагороди матиме більш жорсткі умови успішності. Там, де звичайна функція винагороди мала запас 25 %, ця функція буде використовувати 10 %. Виконання цього тесту покаже, як можна націлити поведінку за допомогою функції винагороди.

3.2 Результати

3.2.1 Базова модель

Початкові тести спочатку виконував базовий агент. Цей агент був запущений у кожному середовищі, щоб побачити, як він працює та поводиться. Були записані ті самі показники, що й для агентів RL, і ці значення використовуватимуться як інструмент вимірювання для агентів RL. Точніше, буде розраховано погіршення продуктивності у порівнянні з попереднім середовищем.

Таблиця 3.1 – Середні винагороди базової моделі

Середовище	Середня винагорода	Різниця
Без перешкод	40,2	-
Помірні перешкоди	22,8	-43,1 %
Багато перешкод	9,7	-57 %

З цих початкових результатів можна зробити висновок, що ефективність базової моделі значно погіршується з ускладненням середовища. З додаванням стін у середовищі BlocksObstacles агент працює значно гірше. Однак стає зрозуміло, що впровадження агента в ще більш складне середовище призводить до ще сильнішого зниження продуктивності.

Для подальшого дослідження того, як базова модель виконує середні епізоди, було створено розподіл винагороди, як показано на малюнку 3.9. Зелена лінія відображає середнє значення останніх 5 кроків.

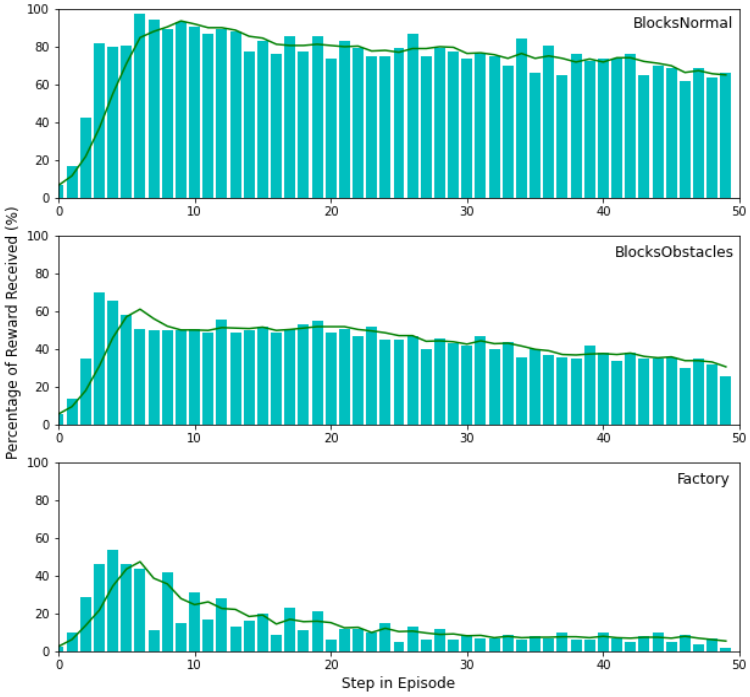

Рисунок 3.9 – Графіки винагороди базового агенту

Перші п'ять кадрів середнього епізоду виглядають однаково для кожного агента. Це відображається на зображенні різким збільшенням винагороди на початку епізоду. Під час перших кадрів агенту важко отримати винагороду через відстань скидання між дроном і людиною. Як показано на малюнку, початкові кадри містять низькі значення. Відсутність винагороди на початкових етапах також є поясненням того, чому агент ніколи не зможе отримати середню винагороду в 50, оскільки отримати винагороду на цих початкових етапах надзвичайно важко. Однак послідовні кроки різко збільшуються, коли дрон наближається до людини. Така поведінка трапляється в кожному епізоді, тому що в перші моменти ймовірність того, що людина зайде за перешкоду, невелика, в результаті чого агент може слідувати за нею. Тим не менш, все ще дуже важко підтримувати 100% винагороду на кожному наступному кроці, оскільки агент іноді діє занадто пізно, що призводить до того, що деякі кроки не отримують винагороду. Ця ж картина спостерігається в середовищі з двома перешкодами, однак із пропорційним погіршенням винагороди на пізніших етапах. Це вказує на нездатність цього агента уникати перешкод після першого набігу на людину.

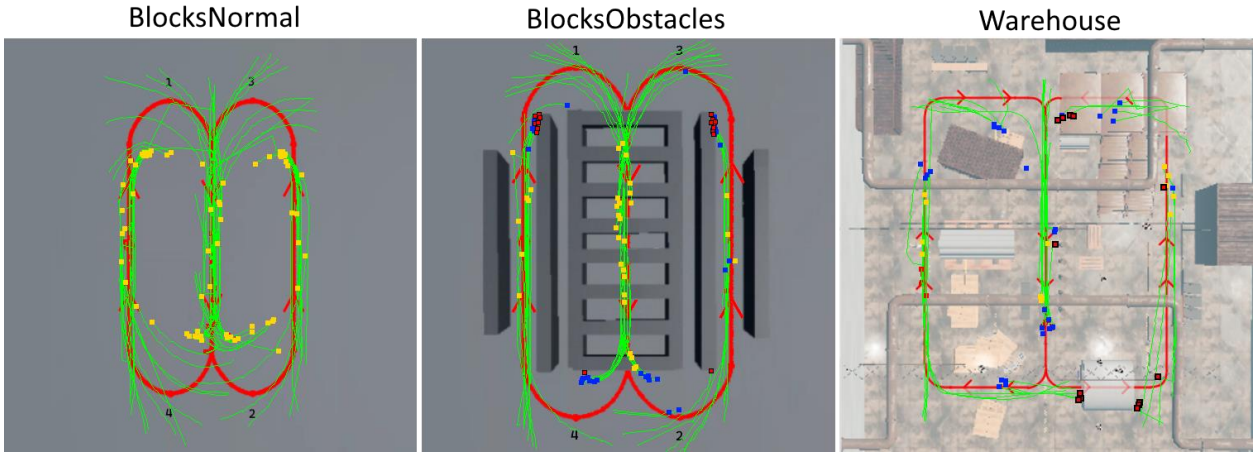

Рисунок 3.10 – Траєкторії базового агенту

Дивлячись на шлях базової моделі всередині середовища BlocksNormal на малюнку, стає зрозуміло, що агент чітко слідує шляху людини. Під час поворотів шляхи агента опиняються всередині діаметра повороту, що й слід було очікувати, оскільки в ці моменти все, що потрібно зробити агенту, це просто повернути, щоб утримати людину в її полі зору з спорадичним рухом вперед, щоб тримати людину на потрібній відстані. Однак виконання такої поведінки всередині двох інших середовищ створило проблеми для цього агента, що можна побачити за збільшенням кількості зіткнень і моментів втрати людини, що додатково показано в таблиці 3.2.

Таблиця 3.2 – Невдалі закінчення епізоду базової моделі

Середовище	Тип завершення епізоду	Кількість епізодів
BlocksNormal	Поза межами поля зору	2
	Зіткнення	0
BlocksObstacles	Поза межами поля зору	33
	Зіткнення	20
Warehouse	Поза межами поля зору	16
	Зіткнення	60

Можна побачити, що як тісні коридори на зовнішніх сторонах пішохідного маршруту, так і ширший коридор посередині карти є легкими ситуаціями для базової моделі. Це має сенс, оскільки базова модель запрограмована на просування вперед у таких ситуаціях. Але в кутових ситуаціях агент не може працювати належним чином. В цих ситуаціях на поворотах необхідно мати можливість уникнути перешкоди. Така поведінка не запрограмована, що призводить до невдач. Приклад невдалого закінчення епізоду можна побачити на малюнку 3.11.

Рисунок 3.11 – Втрата людини з поля зору

Оскільки людина знаходиться на правильній відстані, агент не наближається, а натомість продовжує центрувати особу в своєму полі зору. Однак, як видно на останніх трьох кадрах, стіна стає видимою. Це не впливає на його поведінку, тому людина йде за стіну, через що дрон втрачає її з поля зору.

Нарешті, дивлячись на продуктивність базової моделі в найскладнішому середовищі, стає зрозуміло, що агент має ще більше проблем. Як видно з таблиці 3.2, переважна більшість епізодів закінчується зіткненням або втратою дроном людини з поля зору.

Завдяки цим експериментам стає зрозуміло, що базова модель — це агент, який найкраще працює в середовищі BlocksNormal, де немає перешкод. Однак із введенням перешкод цей евристичний метод стає все більш нездатним впоратися з доповненнями до середовища. Перевага навчання з підкріпленням в тому, що агенти здатні адаптуватись до навколишнього середовища.

3.2.2 Навчання в середовищі без перешкод

У цьому розділі буде обговорено навчання та результати тестування агентів у середовищі BlocksNormal.

Процес навчання. Усередині BlocksNormal було навчено декілька агентів. Який варіант агентів було навчено, було обрано за результатами кожного послідовного тесту. Після навчання на єдиному нормальному зображенні і складеному нормальному зображенні DQN виконуються тестові прогони, щоб побачити, що працює краще. Агент із вищою середньою прибутковістю використовується для наступного тренувального сеансу, де він перенавчається з використанням зображень глибини.

Хід навчання можна спостерігати на малюнку 3.12. Дані цього процесу було згладжено, щоб спостерігати основну тенденцію в мінливих даних. Діапазон даних також був доданий.

Як обговорювалося раніше, процеси навчання перериваються моментами оцінювання кожні 50 ітерацій. Відмінності між навчанням і оцінюванням можна побачити на малюнку 3.12. Двома основними показниками, які реєструються під час навчання, є середня тривалість епізоду та середня винагорода протягом кожного епізоду, тоді як цикли оцінювання спостерігаються через середню винагороду, максимальну винагороду та мінімальну винагороду.

Початкові точки кожної середньої тривалості епізоду під час навчання розходяться. Це відбувається тому, що початкові параметри кожної моделі створюються випадковим чином. Коли це відбувається, агент також діє випадково, знижуючи ймовірність того, що дрон втратить людину з поля зору. Втрата людини з поля зору відбувається тому, що необхідна поведінка для втрати людини складається з послідовності однакових рухів, зокрема обертання в будь-якому напрямку. Імовірність того, що така послідовність відбудеться під дією випадкового агента, дуже мала. У міру того, як агенти починають навчатися, з’являються нові способи поведінки, у тому числі неправильні, такі як послідовності, які втрачають особу з поля зору.

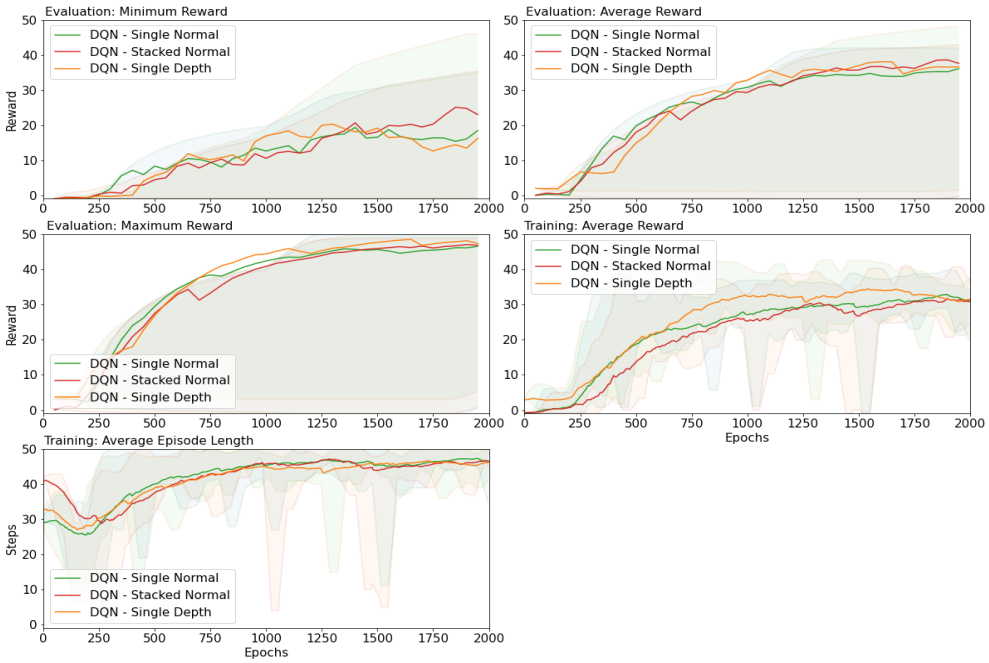

Рисунок 3.12 – Розподіл винагород по епохах

Після цього агент вивчає кращу поведінку, що веде до відновлення середньої винагороди.

Крім того, під час процесу навчання всі агенти змогли наблизити свою середню тривалість епізоду до максимальної кількості. Оскільки епізод може містити щонайбільше 50 кроків, якщо агент здатний утримувати особу в зоні огляду протягом усього цього часу, тривалість епізоду становитиме 50. Оскільки всі агенти можуть сходитися близько до цієї межі, це означає, що що всі агенти змогли навчитися тримати людину в зоні зору, незалежно від відстані. Це також відображається на середній винагороді від навчання, де крива має подібну форму до середньої тривалості епізоду. Що довші епізоди, то вищі середні винагороди, які агент отримує протягом кожного епізоду. Здатність утримувати людину в зоні зору означає перший крок до навчання цільовій поведінці.

Середня винагорода під час циклів навчання в усіх агентів сходиться до подібних значень, які становлять близько 30. Це означає, що з 50 кроків, які робить агент, у середньому 30 кадрів були витрачені на цільові стани. Тому

кадри, що залишилися, були витрачені на стани, коли людина була або не в центрі, або недостатньо близько.

Середня винагорода під час оцінки є вищою, близько 35, бо цикли навчання відбуваються з політикою, яка включає певний ступінь випадковості. Цикли оцінювання виконуються за допомогою жадібної політики.

Модель, що використовувала одинарне зображення глибини змогла досягти майже ідеальних епізодів найшвидше порівняно з іншими моделями. Тож, здатність відчувати відстань між собою та іншими об’єктами позитивно вплинула на процес навчання агента в середовищі, де немає перешкод.

Мінімальний прибуток під час оцінювання є дуже мінливим, це пов’язано з тим, що деякі епізоди все ще настільки заплутують агента, що він дуже швидко закінчує епізод, втрачаючи людину з поля зору. Тим не менш, загальна тенденція мінімальних винагород має позитивний нахил. Кінцеве значення, яке сходиться у всіх цих агентів, становить близько 20, що означає, що всі агенти навчилися такому типу поведінки, який принаймні здатний збирати близько 20 балів винагороди за епізод.

Таблиця 3.2 – Порівняння результатів у середовищі BlocksNormal

Агент	Середня винагорода	Різниця
Базова модель	40,2	
DQN (звичайні зображення)	35,2	-5
DQN (стековані звичайні зображення)	31,9	-8,3
DQN (мапа глибин)	44,5	+4,3

Загалом ці результати показують, що агенти RL у цьому середовищі приблизно відповідають продуктивності базової моделі. Цікавим результатом є те, що накладання зображень не покращило продуктивність і мало найбільше зниження продуктивності порівняно з базовою моделлю.

Крім того, використання зображень глибини замість звичайних зображень справді підвищило продуктивність агента порівняно з базовою моделлю.

Щоб проаналізувати, як кожен агент виконує середній епізод і де є найсильніші аспекти кожного агента, було зроблено малюнок 3.13, де зелена лінія представляє поточне середнє значення останніх 5 значень.

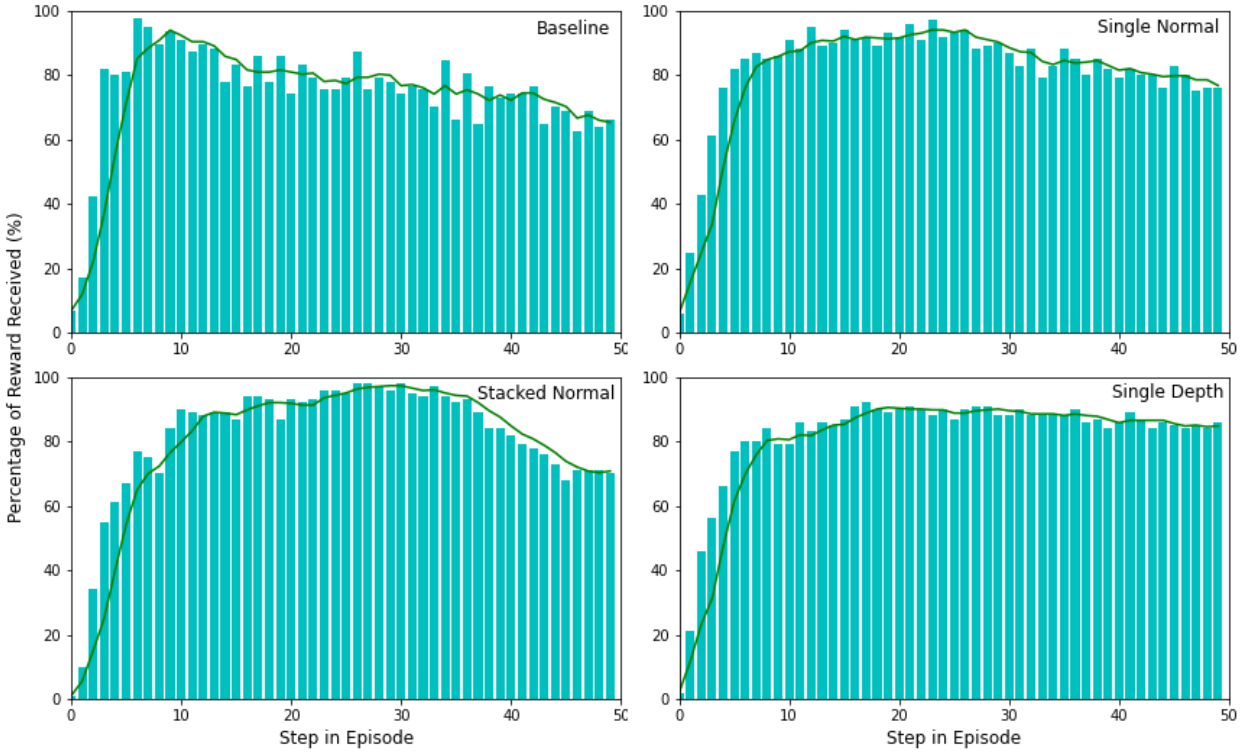

Рисунок 3.13 – Розподіл відсотку винагороди

Як видно, існують відмінності в тому, як кожен агент виконує середній епізод. Найбільш помітним є те, що, за винятком агента глибини, більшість агентів мали проблеми із збереженням стабільної винагороди довше ніж 25

кроків. Ці результати демонструють деякі недоліки моделей RL, які не використовували зображення глибини. Обидва ці агенти були менш стабільними у своїй поведінці протягом середнього епізоду, ніж базовий агент. Шляхи агентів можна побачити на малюнку 3.14.

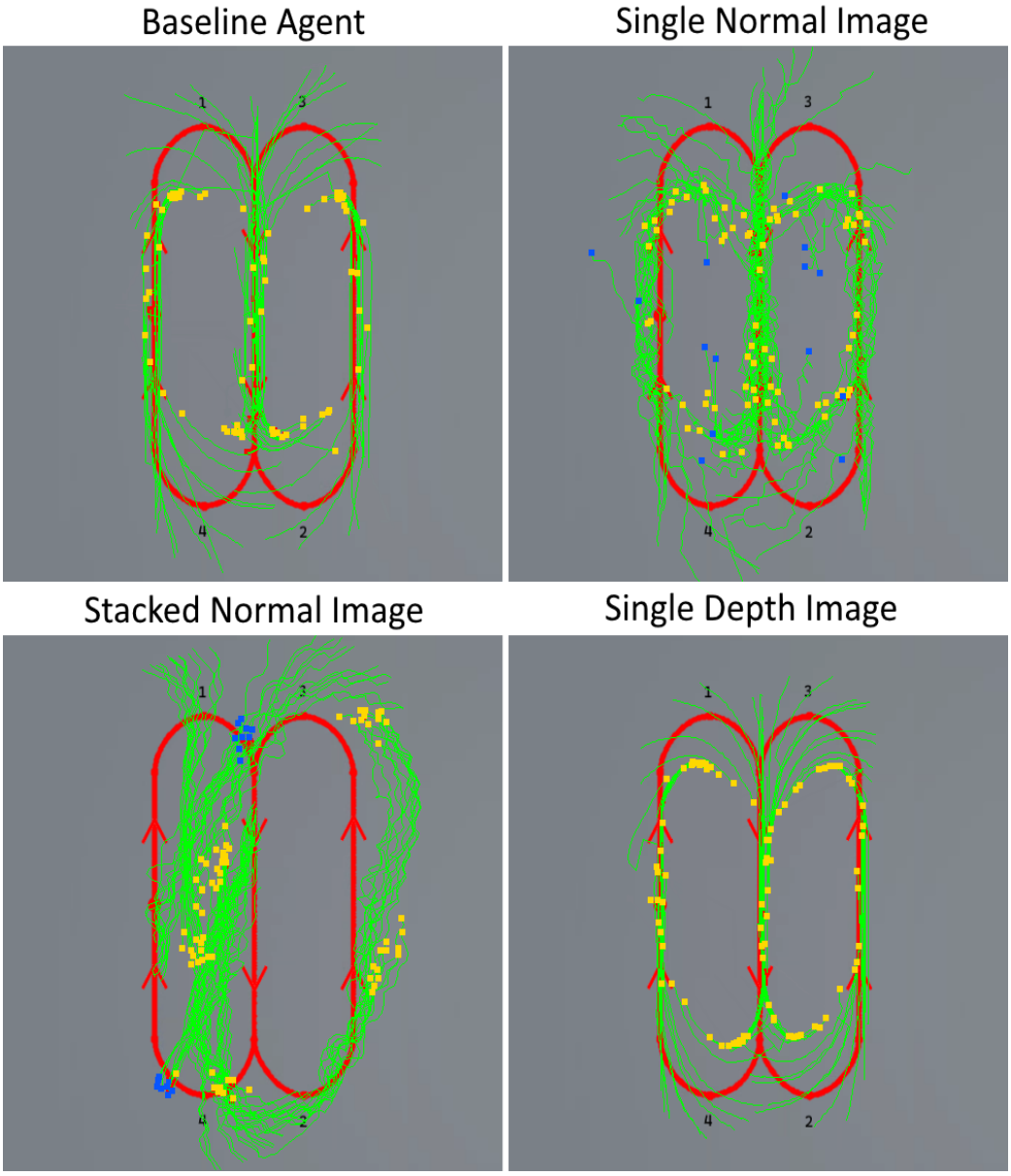

Рисунок 3.14 – Траєкторії моделей

На малюнку зображено шляхи та кінці епізодів усіх агентів під час тестового запуску зі 100 епізодів. Зелені лінії представляють траєкторії польоту агента. Крапки відповідають наступним кінцям епізоду: червоний – зіткнення, синій – поза межами поля зору, жовтий – нормальний.

Щоб проаналізувати загальну поведінку агентів, шляхи протягом 100 тестових прогонів епізодів були накладені на шлях людини.

Найпомітнішим елементом у цьому зображенні є контраст, який мають агенти з одним нормальним і складеним нормальним зображенням порівняно з двома іншими агентами.

Таблиця 3.3 – Поза полем зору агентів у середовищі BlocksNormal під час тестування

Агент	Поза полем зору
Baseline	2
DQN Normal	15
DQN Stacked Normal	19
DQN Single Depth	0

Можливим поясненням цієї високої мінливості може бути той факт, що використання звичайних зображень призводить до великого простору станів, що ускладнює для агента пошук глобального оптимуму. Натомість він встановлюється на локальному оптимумі, і пошук кращої послідовності дій залишається важким і схильним до невдачі.

Стосовно іншого агента, що має недостатню продуктивність, нормальний агент зі стеком виконує зовсім іншу поведінку, ніж усі інші. Цей агент позиціонує себе праворуч від людини та намагається тримати її в полі зору з цієї сторони. Ця стратегія працює в усі моменти, коли людина йде вперед, однак, як можна спостерігати на малюнку 3.14, вона опиняється в ситуації, яку потім набагато складніше успішно обробити.

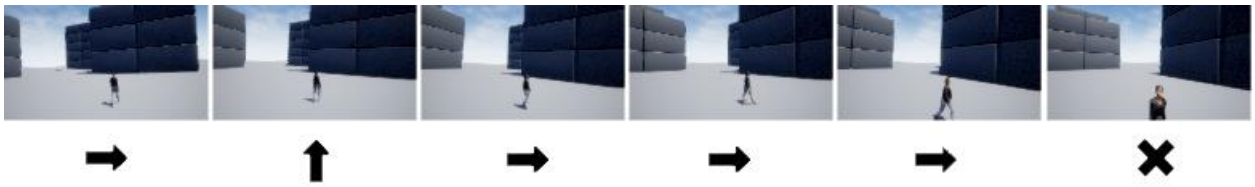

Рисунок 3.14 – Завелике наближення

Такі ситуації пояснюють раптове падіння розподілу винагороди, яке відбувається через те, що агент має справу з цими станами на наступних етапах епізоду.

Потенційним поясненням може бути той факт, що поєднання збільшеного простору станів за рахунок нагромадження зображень і здатності краще сприймати рухи, коли дрон розташований збоку, перешкоджає процесу навчання. Позиціонування себе осторонь у певний момент під час процесу навчання може призвести до більшої винагороди для агента. Після того, як ця поведінка була зміцнена великою кількістю циклів навчання, агенту стає все важче відучитися від цієї поведінки. Це особливо актуально, враховуючи, що DQN бере вибірку з більшого буфера відтворення, зміцнюючи пам’ять про те, що ці дії призводять до вищих винагород.

Нарешті, дивлячись на агента, що використовує мапу глибини, можна побачити вищі значення середньої винагороди. Загальна траєкторія польоту плавна навіть порівняно з базовою моделлю. Крім того, жоден епізод не закінчився тим, що агент втратив особу з поля зору.

3.2.3 Навчання в середовищі з помірними перешкодами

Процес навчання відбувався дуже подібно до середовища BlocksNormal, з деякими додатковими моментами. Точний процес можна спостерігати на малюнку 3.15. Найпомітнішим моментом є те, що агенти не змогли зблизити свою середню тривалість епізоду до максимальної. Проте деякі показали кращий результат, ніж інші. Загальною тенденцією в усіх показниках є значне зниження продуктивності порівняно з попереднім середовищем.

Цікавий момент, який слід зазначити, полягає в тому, що модель, що використовувала стек зображень глибини змогла швидше досягти свого конвергентного значення. Ефективність моделі, що використовувала одинарні нормальні зображення була нижчою порівняно з двома іншими моделями, які збігалися навколо подібного значення. Ці особливості також спостерігаються в оціночних показниках середньої прибутковості, де найшвидше найвищих значень досягала модель складеного зображення глибини.

Максимальна винагорода агентів досягла значення конвергенції майже 50, що знову означає, що найкращі епізоди агента були найкращими потенційно можливими. У той же час мінімальна винагорода була значно нижчою порівняно з попереднім середовищем.

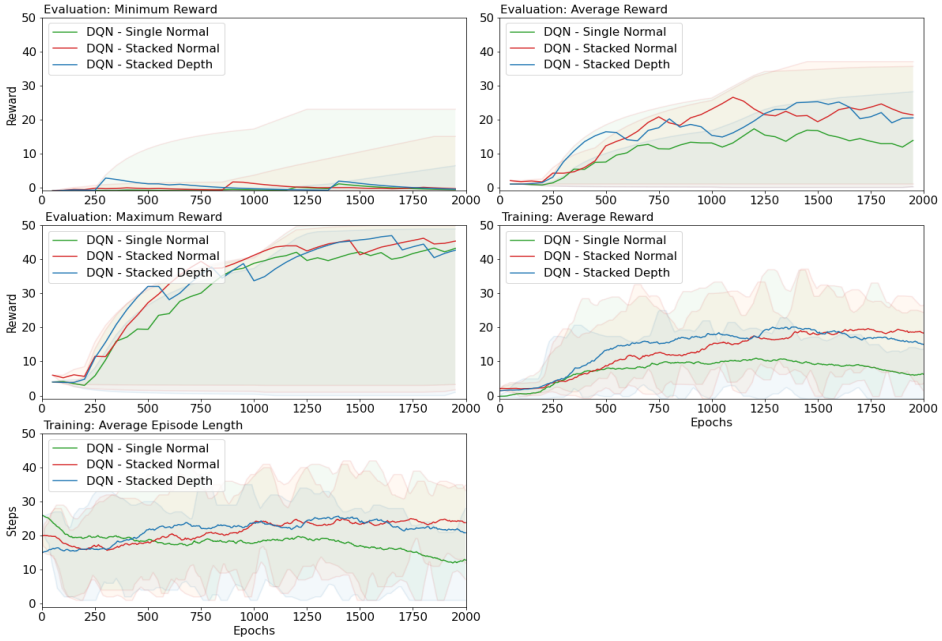

Рисунок 3.15 – Метрики для середовища з перешкодами

Результати тестових запусків агентів можна побачити в таблиці 3.4. Було додано зниження продуктивності порівняно з попереднім середовищем.

Таблиця 3.4 – Порівняння тестових прогонів у BlocksObstacles

Агент	Середня винагорода	Різниця з попереднім середовищем
Baseline	22.6	-43.7 %
DQN (звичайні зображення)	19.1	-45.7
DQN (стековані звичайні зображення)	24.8	-22.3
DQN (стекована мапа глибин)	30.1	-28.3

Ці результати показують, що доповнення мали позитивний вплив на продуктивність кожного наступного впровадження агента. Найважливішим є те, що використання складених зображень виявилося більш корисним доповненням до агента, ніж здавалось раніше. Тим часом використання глибинного зображення дало ще кращі результати.

Ці результати свідчать про те, що використання алгоритмів RL може створювати набагато більш адаптовні моделі порівняно з класичними методами. Однак репрезентація станів, яке використовується для цих агентів, має велике значення.

Дивлячись на винагороди, які кожен агент отримував за середній епізод під час тестового запуску (малюнок 3.16), ми бачимо деякі відмінності між їх поведінкою.

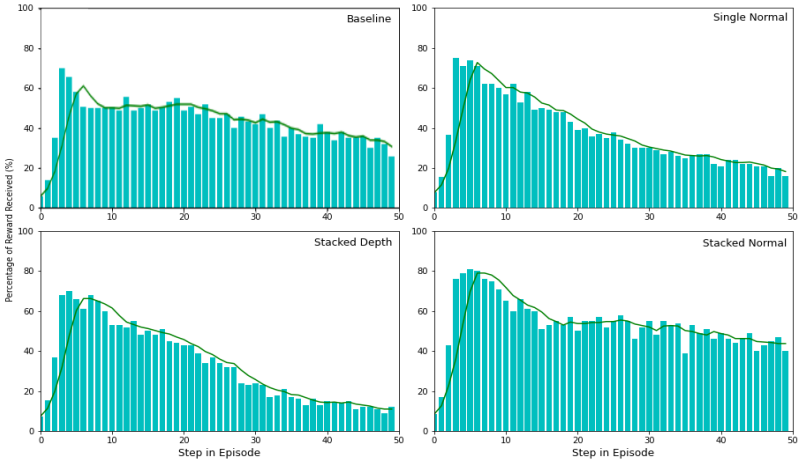

Рисунок 3.16 – Розподіл винагород

Відмінність, яку дає укладання зображень, — це більша винагорода на проміжних етапах епізодів. Це означає, що агент перебуває в цільових станах протягом більш тривалого часу, перш ніж опуститися до нижчих значень. Модель стекованого зображення глибини може зберігати постійне значення з 15-го кадру, незважаючи на падіння початкових кадрів. Зрозуміло, що

спостережувана раніше стабільна поведінка в середовищі BlocksNormal також з’являється в цьому новому середовищі.

На малюнку 3.17 можна побачити шляхи польоту агентів у середовищі BlocksObstacles а також області, в яких відбулися закінчення епізоду.

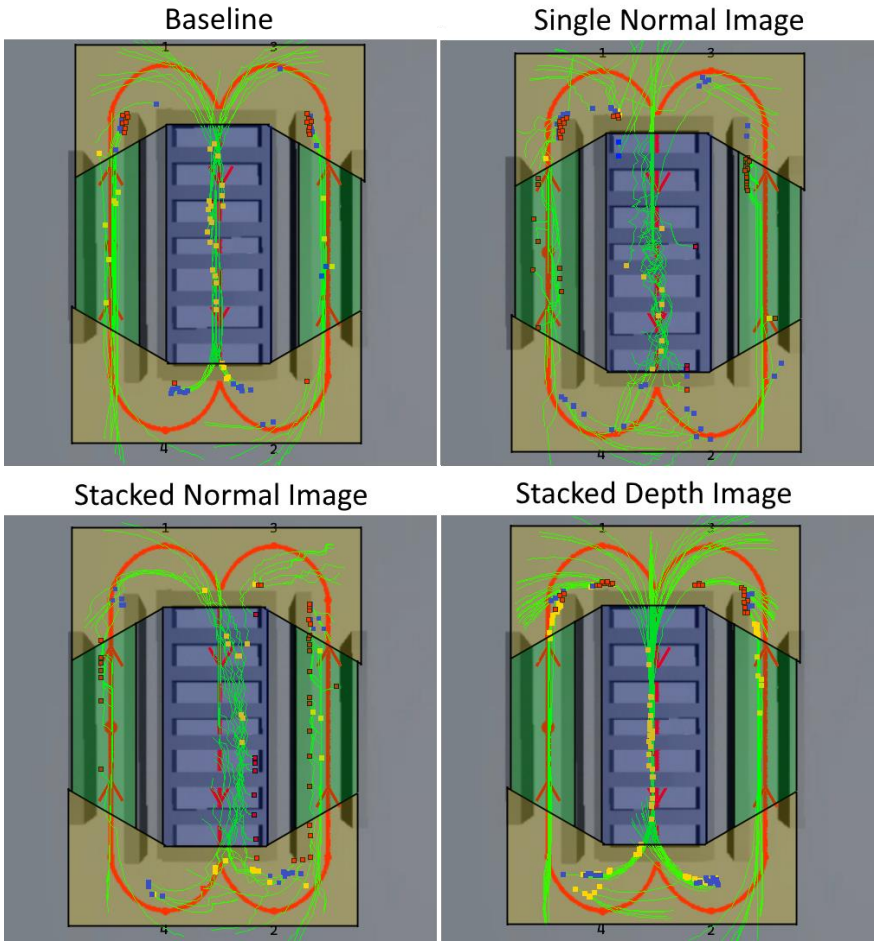

Рисунок 3.17 – Шляхи та кінці епізодів усіх агентів протягом 100 епізодів тестових запусків у BlocksObstacles

Пофарбовані області відповідають таким ситуаціям: зелений – тісні коридори, синій – широкі коридори, жовтий – кути.

На цих зображеннях знову видно нестабільну поведінку агента, що використовував єдине звичайне зображення. Крім того, стекований нормальний

агент повторює свій шаблон поведінки, позиціонуючи себе збоку від людини, але цього разу зліва.

В агента, що використовує мапу глибини, можна спостерігати ту саму схему гладких і стабільних траєкторій польоту, що й раніше. Тим не менш, у цьому середовищі кількість зіткнень і виходу з поля зору зросла.

3.2.4 Навчання в середовищі з великою кількістю перешкод

Остаточне середовище, у якому було виконано тестування, – це середовище Warehouse. Тут перевіряється найефективніший агент із кожного середовища. Оскільки вони мали різне представлення стану, кращу робочу версію цих агентів у тестових прогонах середовища Warehouse було перенавчено в цьому середовищі.

Також у цьому середовищі було начено агента PPO, який використовує репрезентацію стану, що показала себе найбільш ефективною, тобто стековані мапи глибини.

Після виконання тестів, найефективнішими моделями були ті, що використовували стековані мапи глибини. Процес навчання можна спостерігати на малюнку 3.21.

Рисунок 3.21 – Навчання агента Stacked Depth у Warehouse

Результати навчання агента, що використовував стековані мапи глибини, в складському середовищі показують подібне погіршення продуктивності навчання, яке спостерігалося в середовищі BlocksObstacles порівняно з BlocksNormal. Однак ступінь цієї деградації в цьому середовищі нижчий. Середня тривалість епізоду під час навчання була приблизно такою ж, як у середовищі BlocksObstacles, що становило близько 20 кроків.

Останні випробування, які були виконані, зосереджені на генералізації агентів RL для нового, складнішого середовища.

В таблиці 3.5 можна спостерігати середню винагороду та порівняні падіння продуктивності агентів.

Таблиця 3.5 – Порівняння результатів у середовищі Warehouse

Агент	Середня винагорода	Різниця
Базова модель	9,9	-56,2
DQN (звичайні зображення)	11,5	-72,7
DQN (стековані мапи глибин)	13,8	-54,1
DQN 2 (стековані мапи глибин)	21,6	-28,3
PPO (стековані мапи глибин)	22,1	-

Не дивно, що базова модель була найнеефективнішим агентом набору. Також додаткова складність перешкод створила набагато більшу проблему для агента, що використовував одинарні мапи глибини.

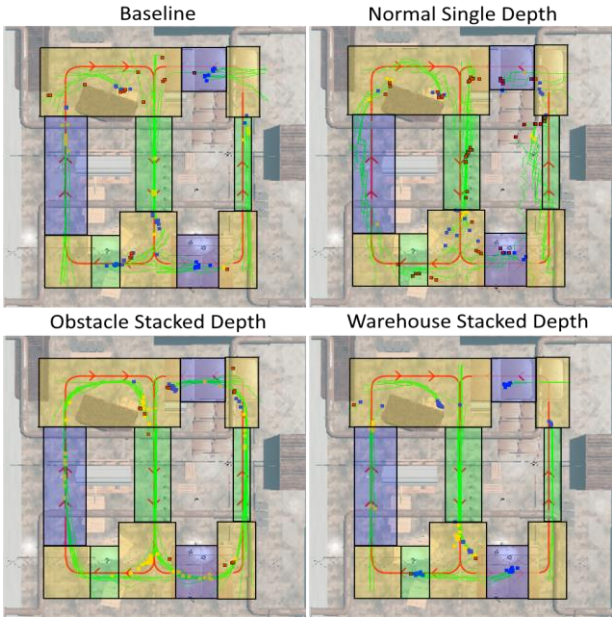

Рисунок 3.21 – Шляхи та кінці епізодів агентів під час 100 тестових епізодів

На малюнку видно той факт, що агенти, що використовують зображення глибини, навчені в середовищах з перешкодами, показали подібну стабільну поведінку, як і в попередніх двох експериментах.

Порівняння функцій винагороди. У фінальному тесті функція винагороди була дещо скоригована, як описано раніше. Використовуючи найкращий робочий агент від BlocksObstacles, яким був DQN Stacked Depth, нового агента було навчено використовувати цю нову функцію винагороди. Ця функція винагороди полягала в невеликому зниженні цільових станів. Зменшуючи межі цільової відстані до людини, функція винагороди стає суворішою. Тестування нового агента дасть уявлення про вплив функції винагороди на набуту поведінку.

Розглядаючи більш конкретно середній епізод на малюнку, стає зрозуміло, що цей агент краще здатний підтримувати початковий рівень винагород, отриманих на ранніх етапах епізоду.

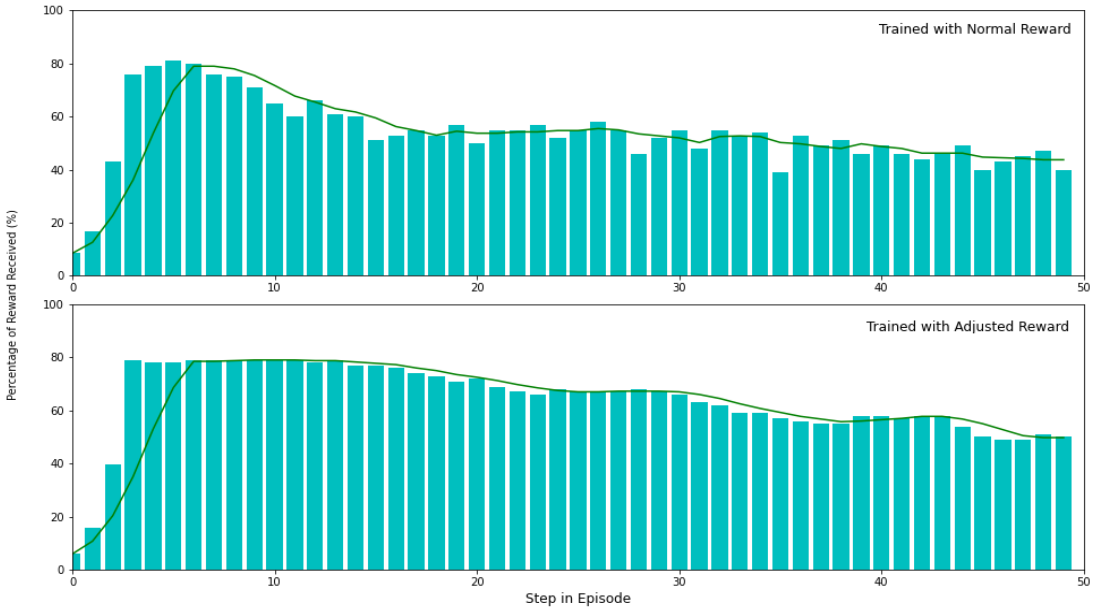

Рисунок 3.23 – Розподіл відсотку винагороди

Крім того, подальша тенденція до зниження менш очевидна, ніж в агента, навчений із використанням звичайної винагороди. На малюнку 3.24 можна побачити траєкторії агентів з різними винагородами.

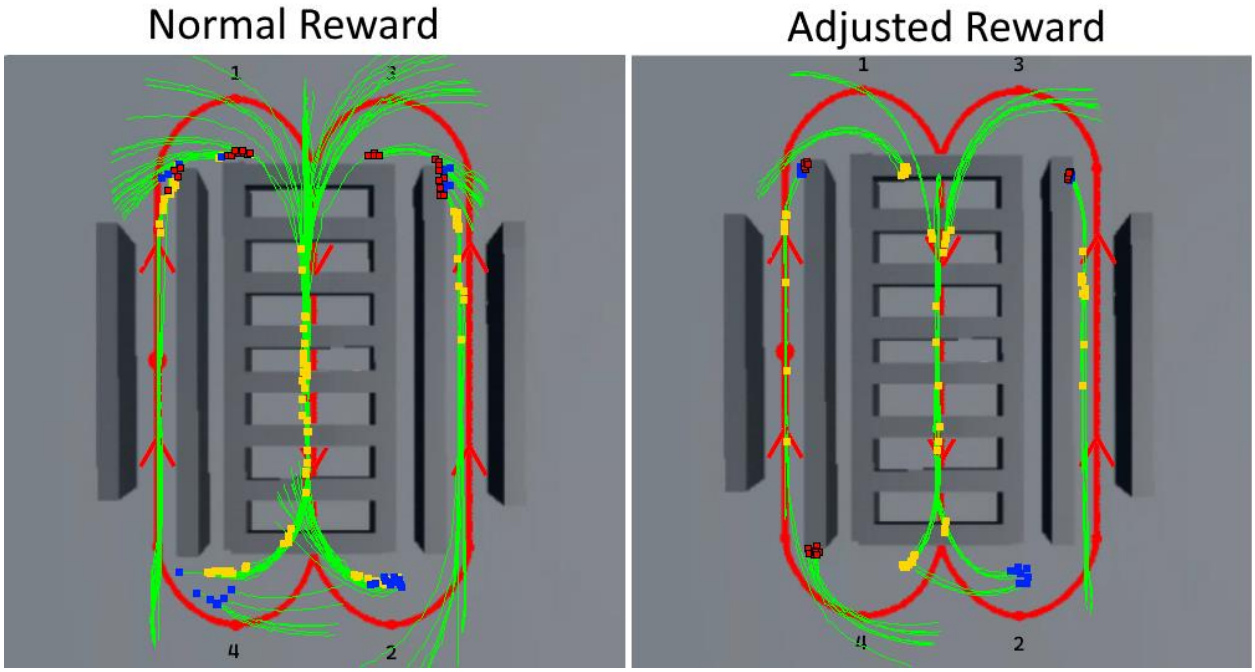

Рисунок 3.24 – Місце розташування кожного типу ситуації в середовищах

Як видно, загальна поведінка виглядає схожою. Багато з раніше згаданих моделей поведінки знову очевидні, а саме: дотримання стабільної траєкторії польоту, перебування поблизу маршруту прогулянки людини, відсутність проблем із ситуаціями в коридорах.

Однак різниця між агентами полягає в тому, що агент, навчений за допомогою скоригованої винагороди, мав загалом менше невдалих кінців епізоду.

Підсумовуючи, зміни у функції винагороди мають сильний вплив на навчену поведінку. Враховуючи результати цього експерименту, мінімальні зміни функцій винагороди значно покращили ефективність агента. Інші тести з

можливостями різних функцій винагороди все ще потрібні, але виходять за рамки цієї дипломної роботи.

3.3 Висновки до розділу

Підсумовуючи, у цьому розділі було досліджено, наскільки використання навчання з підкріпленням є життєздатним методом в контексті задачі переслідування об’єктів за допомогою автономного дрона. Результати показали, що існує потенціал у використанні методів RL для цього завдання, особливо у порівнянні з класичними підходами. Було порівняно різні типи агентів і їх ефективність у контексті задачі, що досліджується. Крім того, реалізація представлень стану, які включають інформацію про динамічні рухи об’єктів та їхні відстані, демонструють значні переваги перед алгоритмами RL, які покладаються виключно на вхідні дані камери. Незважаючи на те, що перед розгортанням у реальних застосунках необхідна подальша робота, використання навчання з підкріпленням демонструє значні переваги в адаптивних процесах прийняття рішень і можливості узагальнення поведінки.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП ПРОЄКТУ

У сучасному світі розвиток технологій безпілотних літальних апаратів (дронів) надає найрізноманітніші можливості в різних галузях, включаючи логістику, аграрну сферу, військову та цивільну безпеку, а також дослідження та розвідку. Однією зі складних задач, пов'язаних з використанням дронів, є здатність ефективно переслідувати об'єкти в режимі реального часу в умовах обмежених обчислювальних ресурсів.

Для розв'язання цієї проблеми важливим є застосування методів машинного навчання з підкріпленням. Машинне навчання з підкріпленням є підходом до навчання, при якому агент (у цьому випадку дрон) вчиться приймати рішення шляхом проб і помилок, нагороджуючи його за правильні дії та караючи за неправильні. Цей метод може бути використаний для розв'язання задачі переслідування об'єктів.

Основні виклики включають в себе обробку потокових відеоданих з камер та/або інших датчиків, визначення руху об'єктів, вибір оптимальної стратегії переслідування та керування дроном для досягнення цієї стратегії. Однак це все повинно здійснюватися в реальному часі та з обмеженими обчислювальними ресурсами, оскільки дрони мають обмежену обчислювальну потужність та об'єм пам'яті.

У даному дослідженні ми розглядаємо можливості використання методів машинного навчання з підкріпленням для оптимізації процесу переслідування об'єктів дроном в умовах обмежених обчислювальних ресурсів. Ми досліджуємо різні алгоритми навчання з підкріпленням, способи обробки відеоданих, а також оптимальні стратегії керування дроном. Наша мета - розробити ефективний та надійний метод переслідування об'єктів для безпілотних літальних апаратів у реальних умовах з обмеженими обчислювальними ресурсами.

Ця робота може мати широкий практичний застосунок у сферах військової діяльності, пошуку та рятування, відеоспостереження, а також в інших областях, де важлива можливість ефективно переслідувати об'єкти за допомогою дронів з обмеженими обчислювальними ресурсами.

4.1 План розробки стартапу

План розробки стартапу для реалізації проекту з керування дроном для задачі переслідування об'єктів в умовах обмежених обчислювальних ресурсів може бути таким:

1. Обгрунтування і концепція: Провести докладний аналіз ринку та ідентифікувати потенційні сегменти користувачів та конкурентів. Сформулювати основну концепцію проекту, визначити його цільові завдання та завдання.
2. Підготовка досліджень: Провести дослідження в області машинного навчання з підкріпленням, алгоритмів відстеження об'єктів та керування дронами. Зібрати вихідні дані для навчання моделей (відеодані, вектори об'єктів, тощо).
3. Розробка програмного забезпечення: Розробити алгоритми відстеження об'єктів на базі методів машинного навчання, таких як Deep Q-Networks (DQN), Recurrent Neural Networks (RNN), або Reinforcement Learning from Demonstrations (RLfD).
4. Створити програмне забезпечення для обробки відеоданих, визначення руху об'єктів та вибору стратегій керування дроном. Оптимізувати алгоритми для роботи на обмежених обчислювальних ресурсах.

5. Розробка апаратного забезпечення: Вибір апаратної платформи для дрона, яка підтримує програмне забезпечення та датчики, необхідні для задачі переслідування.
6. Інтеграція програмного та апаратного забезпечення.
7. Тестування та валідація: Провести внутрішнє тестування на симуляторі дрону та в контрольованих умовах. Валідувати роботу системи на реальних дронах та реальних умовах переслідування об'єктів.
8. Оптимізація та розширення функціональності: Оптимізувати систему для зменшення споживання обчислювальних ресурсів та підвищення ефективності переслідування. Розглянути можливість додавання додаткових функціональних можливостей, таких як автономний польот, маршрутне планування тощо.
9. Маркетинг та продажі: Розробити стратегію маркетингу та просування продукту на ринку. Провести презентації та демонстрації системи потенційним клієнтам та інвесторам.
10. Збір фінансування: Шукати можливості залучення фінансування, включаючи інвестиції від інвесторів, гранти та інші джерела.
11. Масштабування та підтримка: Розглянути можливість масштабування бізнесу та продукту на нові ринки та сегменти користувачів. Забезпечити підтримку та обслуговування клієнтів.
12. Постійний розвиток: Продовжувати дослідження та розробку, оновлювати програмне та апаратне забезпечення відповідно до ринкових потреб та технологічних змін.

4.2 Опис ідеї

Ідея стартап-проекту полягає в розробці та впровадженні інноваційної системи керування дроном для задачі переслідування об'єктів в умовах обмежених обчислювальних ресурсів. Цей стартап націлено на ринки, де безпілотні літальні апарати можуть бути використані для важливих завдань, таких як моніторинг, рятувальні операції, військова діяльність та багато інших.

Основні особливості та переваги ідеї стартапу:

1. Машинне навчання з підкріпленням (Reinforcement Learning): Система використовує сучасні методи машинного навчання з підкріпленням для навчання дрона ефективно переслідувати об'єкти, навіть у складних умовах.

2. Оптимізація роботи на обмежених обчислювальних ресурсах: Однією з ключових переваг є здатність працювати на дронах з обмеженими обчислювальними ресурсами, що робить систему відмінним рішенням для реальних застосувань.

3. Широкий спектр застосувань: Система може бути використана в різних галузях, включаючи пошук та рятування, логістику, військову діяльність, моніторинг навколишнього середовища, патрулювання та багато інших.

4. Автономність та ефективність: Дрон може працювати автономно, здійснюючи переслідування об'єктів без значного втручання оператора.

5. Можливість масштабування: Ідея передбачає можливість розширення функціональності та масштабування на нові ринки та сегменти користувачів з врахуванням потреб ринку.

Цей стартап спрямований на розробку передової технології, яка може змінити спосіб використання безпілотних літальних апаратів у важливих галузях. Команда стартапу має намір створити продукт, який буде корисним для вирішення складних завдань переслідування об'єктів, забезпечуючи високу ефективність та ефективність в умовах обмежених ресурсів.

Таблиця 4.1 – Інформаційна карта стартап-проєкту

Автор проєкту	Рибалко Анастасія Анатоліївна
Коротка анотація	Стартап розробляє інноваційну систему керування дроном з використанням машинного навчання з підкріпленням для задачі переслідування об'єктів в умовах обмежених обчислювальних ресурсів. Проект спрямований на забезпечення автономного та ефективного переслідування об'єктів з високою точністю в реальному часі.
Термін реалізації проєкту	2-3 роки
Необхідні ресурси	Команда експертів у галузі машинного навчання, аеродинаміки та програмної розробки. Фінансування на етапи дослідження, розробки, тестування та впровадження. Високоякісне апаратне забезпечення. Вихідні дані для навчання моделей машинного навчання. Доступ до тестових дронів та об'єктів для валідації.

Закінчення таблиці 4.1

Опис проблеми, яку вирішує проєкт	Безпілотні літальні апарати відіграють важливу роль у багатьох сферах, але задача ефективного та автономного переслідування об'єктів залишається складною через обмежені обчислювальні ресурси дронів та вимоги до точності та швидкості переслідування
Головні цілі та завдання проєкту	Розробити алгоритми машинного навчання з підкріпленням для ефективного відстеження об'єктів. Інтегрувати розроблену систему з дронами та забезпечити автономну роботу. Оптимізувати роботу системи на обмежених обчислювальних ресурсах. Провести валідацію системи на реальних дронах та в реальних умовах.
Очікувані результати	Функціональна система переслідування, готову для комерційного використання. Підвищення ефективності та автономності дронів. Позиція на ринку та можливості для росту та розвитку бізнесу.

4.3 Технологічний аудит ідеї проєкту

Нижче буде наведено ідею стартапу та проведений конкурентний аналіз. У таблиці 4.2. наведено опис ідеї стартапу.

Таблиця 4.2 – Опис ідеї проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Основна ідея проєкту полягає у створенні системи, що буде виконувати задачу переслідування на основі даних з камери дрону.	Використання окремими користувачами	В даному випадку користувачем є будь-яка людина, що потребує інструменту для проведення переслідуавння об’єктів.
	Використання організаціями	В даному випадку користувачами можуть виступати представники силових структур, рятувальники тощо. Продукт може використовуватися для проведення переслідування у ситуаціях, де втручання людини є неможливим або небезпечним.

Порівняльний аналіз конкурентів продукту, що вже є на ринку зображено на табл. 4.3.

Таблиця 4.3 – порівняльний аналіз конкурентів продукту

Назва продукту	Розробник	Відомості про продукт	Степінь загрози для випуску нашого продукту
SkyTrak Pro	AeroTech Solutions Ltd	SkyTrak Pro - це система керування дроном, яка використовує розширені алгоритми машинного навчання для переслідування об'єктів. Вона призначена для застосування у важких умовах та підтримує обмежені обчислювальні ресурси.	Помірна загроза. SkyTrak Pro вже встановився на ринку та має деяку клієнтську базу, але існує можливість конкурувати за рахунок вдосконалення функціональності та оптимізації роботи на обмежених ресурсах.

Закінчення таблиці 4.3

DroneGuardian	SecureFlight Technologies Inc.	DroneGuardian - це інноваційна система керування дронами, яка використовує адаптивні алгоритми машинного навчання для забезпечення точного та автономного переслідування об'єктів в реальному часі. Вона також оптимізована для роботи на обмежених обчислювальних ресурсах.	Серйозна загроза. DroneGuardian має сильний технологічний стек та активно просувається на ринку. Для конкуренції потрібно буде надавати продукт із значними перевагами.

Далі проаналізуємо технічну реалізовність ідеї проєкту (табл. 4.4).

Таблиця 4.4 – Технологічна здійсненність проєкту

Технічне завдання для проєкту	Спосіб досягнення результату
Підготовка середовищ для навчання моделі	Треба підготувати середовища для навчання моделі, максимально наближені до потенційних умов, в яких доведеться працювати.
Навчання моделі	Навчання моделей в зазначених середовищах, тестування, розробка користувацького інтерфейсу.
Позиціонування продукту на ринку	В першу чергу продукт має позиціонуватися як допоміжний інструмент при виконанні задач переслідування у випадках, де є високим ризик втрати дрона і втручання людини є неможливим або небезпечним. Таким чином можна досягти виконання задач шляхом використання бюджетних рішень і зменшення ризиків для потенційних виконавців.

4.4 Аналіз ринкових можливостей запуску стартап-проєкту

Далі наведемо попередній аналіз ринку для реалізації стартап проєкту (таблиця 4.5).

Таблиця 4.5 – Попередня характеристика потенційного ринку стартап-проєкту

№ п/п	Показники (найменування)	Характеристика
1	Кількість головних гравців	3
2	Загальний обсяг продаж конкурентів, грн/ум. од	Підписка 50 ум.од за рік 7 ум.од. за одноразове використання
3	Динаміка ринку	Зростаюча, позитивна
4	Наявність обмежень для входу	Модель повинна бути достатньо точною для конкурування з існуючими рішеннями. Вартість навчання моделі і моделювання середовищ.
5	Специфічні вимоги до стандартизації та сертифікації	Можливі юридичні особливості у військовій галузі.
6	Середня норма рентабельності в галузі (або по ринку), %	20%

Далі наведено фактори конкурентоспроможності товару (табл. 4.6).

Таблиця 4.6 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування
1	Автоматизація	Не потребує унебезпечення людини.
	Ефективність та надійність	Необхідно досягти більшої точності ніж у конкурентів або меншої вартості при збереженні точності. Саме тому в проекті робиться акцент на ефективності в умовах обмежених обчислювальних ресурсів.
	Простота у використанні	Користувацький інтерфейс повинен бути зручним і інтуітивно зрозумілим, щоб бути використовним у стресових умовах.

Висновки до розділу 4

В даному розділі було розглянуто стартап проєкт у вигляді системи керування дроном для задачі переслідування об'єктів в умовах обмежених обчислювальних ресурсів. Була описана основна ідея проєкту, її технічний аудит. Була розглянута технологічна здійсненність проєкту, а також аналіз ринкових можливостей стартап-проєкту. Були визначені основні фактори конкурентоспроможності застосунку та всі необхідні ресурси для його реалізації. Загалом можна зробити висновок, що даний стартап-проєкт є реалізовним і ринкові можливості сприяють створенню подібних продуктів

ВИСНОВКИ

Підсумовуючи, у цій дипломній роботі було досліджено, наскільки використання навчання з підкріпленням є життєздатним методом в контексті задачі переслідування об’єктів за допомогою автономного дрона. Результати показали, що існує потенціал у використанні методів RL для цього завдання, особливо у порівнянні з класичними підходами.

Було послідовно реалізовано різні типи агентів навчання з підкріпленням та порівняно їх ефективність в контексті задачі переслідування в умовах обмежених обчислювальних ресурсів, що розглядається в поточній роботі.

Крім того, з’ясовано, що реалізація представлень стану, яка включає інформацію про динаміку руху об’єктів та їхні відстані, демонструє значні переваги перед алгоритмами RL, які покладаються виключно на вхідні дані камери. Незважаючи на те, що перед розгортанням у реальних застосунках необхідна подальша робота, використання навчання з підкріпленням демонструє значні переваги в адаптивних процесах прийняття рішень і можливостях узагальнення поведінки.

Результати цієї роботи заплановано апробувати на V Міжнародній науково-практичній конференції «Scientific Community: Interdisciplinary Research» (Hamburg, Germany, 2024)..

ПЕРЕЛІК ПОСИЛАНЬ

1. Niglio F, Comite P, Cannas A, Pirri A, Tortora G. Preliminary Clinical Validation of a Drone-Based Delivery System in Urban Scenarios Using a Smart Capsule for Blood. *Drones*. 2022; 6(8):195.
2. Abro GEM, Zulkifli SABM, Masood RJ, Asirvadam VS, Laouti A. Comprehensive Review of UAV Detection, Security, and Communication Advancements to Prevent Threats. *Drones*. 2022; 6(10):284.
3. Zhang S, Huang M, Cai C, Sun H, Cheng X, Fu J, Xing Q, Xue X. Parameter Optimization and Impacts on Oilseed Rape (Brassica napus) Seeds Aerial Seeding Based on Unmanned Agricultural Aerial System. *Drones*. 2022; 6(10):303.
4. Wu, Yi, Jongwoo Lim, and Ming-Hsuan Yang. "Online object tracking: A benchmark." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013.
5. Kristan, Matej, et al. "The sixth visual object tracking vot2018 challenge results." *Proceedings of the European conference on computer vision (ECCV) workshops*. 2018.
6. Zhao, Jiang, et al. "End-to-End deep reinforcement learning for image-based UAV autonomous control." *Applied Sciences* 11.18 (2021): 8419.
7. Zhang, L., & Nevatia, R. Robust multi-object tracking in crowded scenes. *International Conference on Computer Vision and Pattern Recognition*. 2013.
8. A. Anwar and A. Raychowdhury, “Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning,” *IEEE Access*, vol. 8, pp. 26549–26560, 2020.
9. Xie, Linhai, et al. "Towards monocular vision based obstacle avoidance through deep reinforcement learning." *arXiv preprint arXiv*:1706.09829 (2017).
10. Cetin, Ender, et al. "Drone navigation and avoidance of obstacles through deep reinforcement learning." *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC). IEEE*, 2019.

11. Hodge, Victoria J., Richard Hawkins, and Rob Alexander. "Deep reinforcement learning for drone navigation using sensor data." *Neural Computing and Applications* 33 (2021): 2015-2033.

12. Tsai, Jichiang, Peng-Chen Lu, and Ming-Hong Tsai. "Accuracy improvement of straight take-off, flying forward and landing of a drone with reinforcement learning." *2019 IEEE international conference on consumer electronics-taiwan (ICCE-TW). IEEE*, 2019.

13. Luo, Wenhan, et al. "End-to-end active object tracking and its real-world deployment via reinforcement learning." *IEEE transactions on pattern analysis and machine intelligence* 42.6 (2019): 1317-1332.

14. Mao, Wenguang, et al. "Indoor follow me drone." *Proceedings of the 15th annual international conference on mobile systems, applications, and services*. 2017.

15. Mac, Thi Thoa, et al. "The development of an autonomous navigation system with optimal control of an UAV in partly unknown indoor environment." *Mechatronics* 49 (2018): 187-196. control of an UAV in partly unknown indoor environment

16. Mercado-Ravell, Diego A., Pedro Castillo, and Rogelio Lozano. "Visual detection and tracking with UAVs, following a mobile object." *Advanced Robotics* 33.7-8 (2019): 388-402.

17. Do, T. Tuan, and Heejune Ahn. "Visual-GPS combined ‘follow-me’tracking for selfie drones." *Advanced Robotics* 32.19 (2018): 1047-1060.

18. Grondman, Ivo, et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012): 1291-1307.

19. Babaeizadeh, Mohammad, et al. "Reinforcement learning through asynchronous advantage actor-critic on a gpu." *arXiv preprint arXiv*:1611.06256 (2016).

20. Haarnoja, Tuomas, et al. "Soft actor-critic algorithms and applications." *arXiv preprint arXiv*:1812.05905 (2018).

21. Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv*:1707.06347 (2017).

22. Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529-533.

23. Hong, Zhang-Wei, et al. "A deep policy inference q-network for multi-agent systems." *arXiv preprint arXiv*:1712.07893 (2017).

24. Anwar, Aqeel, and Arijit Raychowdhury. "Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning." *IEEE Access* 8 (2020): 26549-26560.

25. Devos, Arne, Emad Ebeid, and Poramate Manoonpong. "Development of autonomous drones for adaptive obstacle avoidance in real world environments." *2018 21st Euromicro conference on digital system design (DSD). IEEE*, 2018.

26. Tsai, Jichiang, Peng-Chen Lu, and Ming-Hong Tsai. "Accuracy improvement of straight take-off, flying forward and landing of a drone with reinforcement learning." *2019 IEEE international conference on consumer electronics-taiwan (ICCE-TW). IEEE*, 2019.

27. Akhloufi, Moulay A., Sebastien Arola, and Alexandre Bonnet. "Drones chasing drones: Reinforcement learning and deep search area proposal." *Drones* 3.3 (2019): 58.

28. Laud, Adam Daniel. Theory and application of reward shaping in reinforcement learning. *University of Illinois at Urbana-Champaign*, 2004.

29. Vecerik, Mel, et al. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards." *arXiv preprint arXiv*:1707.08817 (2017).

30. Nair, Ashvin, et al. "Overcoming exploration in reinforcement learning with demonstrations." *2018 IEEE international conference on robotics and automation (ICRA). IEEE*, 2018.

31. Knox, W. Bradley, et al. "Reward (mis) design for autonomous driving." *Artificial Intelligence* 316 (2023): 103829.

32. Andrychowicz, Marcin, et al. "Hindsight experience replay." *Advances in neural information processing systems* 30 (2017).

33. Liu, Feng, et al. "State representation modeling for deep reinforcement learning based recommendation." *Knowledge-Based Systems* 205 (2020): 106170.

34. Rasheed, Iftikhar, Fei Hu, and Lin Zhang. "Deep reinforcement learning approach for autonomous vehicle systems for maintaining security and safety using LSTM-GAN." *Vehicular Communications* 26 (2020): 100266.

35. Patel, Dhaval K., et al. "Artificial neural network design for improved spectrum sensing in cognitive radio." *Wireless Networks* 26 (2020): 6155-6174.

36. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv*:1312.5602 (2013).

37. Z-Depth Map — Expanding EXIF Data for More Powerful Post-Processing. URL: https://the.me/z-depth-map-expanding-exif-data-for-more-powerful-post-processing/ (дата звернення: 30.09.2023).

38. Ko, Kyungtae. Visual object tracking for UAVs using deep reinforcement learning. *Diss. Iowa State University*, 2020.

39. Peter F. Jones and George J. M. Aitken, "Comparison of three three-dimensional imaging systems," *J. Opt. Soc. Am. A* 11, 2613-2621 (1994)

40. Saxena, Ashutosh, Sung Chung, and Andrew Ng. "Learning depth from single monocular images." *Advances in neural information processing systems* 18 (2005).

41. Ma, Chao, et al. "Hierarchical convolutional features for visual tracking." *Proceedings of the IEEE international conference on computer vision*. 2015.

42. Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." *arXiv preprint arXiv*:1804.02767 (2018).

43. Wu, Yuxin, et al. "Detectron2: A PyTorch-based modular object detection library." *Meta AI* 10 (2019).

44. Kang, Kai, et al. "Object detection in videos with tubelet proposal networks." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017.

45. Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

46. Feng, Di, et al. "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges." *IEEE Transactions on Intelligent Transportation Systems* 22.3 (2020): 1341-1360.

47. Bewley, Alex, et al. "Simple online and realtime tracking." *2016 IEEE international conference on image processing (ICIP). IEEE*, 2016.

48. Wang, Qiang, et al. "Fast online object tracking and segmentation: A unifying approach." *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*. 2019.

49. Shah, Shital, et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles." *Field and Service Robotics: Results of the 11th International Conference. Springer International Publishing*, 2018.

50. Nguyen, Anh, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

51. How to (Safely) Train an Autonomous Drone in a game engine. URL: https://blog.ml6.eu/implementing-a-reinforcement-learning-agent-with-tf-agents-to-train-an-autonomous-drone-in-airsim-b1e85b5994a6/ (дата звернення: 30.09.2023).

52. Feinberg, Eugene A., and Pavlo O. Kasyanov. "MDPs with setwise continuous transition probabilities." *Operations Research Letters* 49.5 (2021): 734-740.

ДОДАТКИ

Додаток А. Лістинг програми

```
from tf_agents.utils import common
from tf_agents.agents.dqn import dqn_agent
from tf_agents.networks import q_network
from tf_agents.networks.actor_distribution_network import
ActorDistributionNetwork
from tf_agents.agents.ppo.ppo_clip_agent import PPOClipAgent
from tf_agents.agents.ppo import ppo_agent
from tf_agents.networks.value_network import ValueNetwork
import tf_agents

import random
import numpy as np
import tensorflow as tf


""" Just a definition to create the agent """
def getAgent(environment, learning_rate, global_step, which = "ppo"):

   fc_layer_params = (512,128)
   conv_layer_params = ((32, 8, 4), (64, 4, 2), (64, 3, 1))

   if which == "dqn":

      # Network
      q_net = q_network.QNetwork(
         environment.observation_spec(),
         environment.action_spec(),
         conv_layer_params   = conv_layer_params,
         fc_layer_params     = fc_layer_params,
         batch_squash        = True)

      """
      q_net = sequential.Sequential([
         tf.keras.layers.Conv2D(16, 3, padding= 'same' , activation='relu'),
         tf.keras.layers.
       ])
      """

      optimizer =
tf.compat.v1.train.RMSPropOptimizer(learning_rate=learning_rate )


      agent = dqn_agent.DqnAgent(
         environment.time_step_spec(),
         environment.action_spec(),
         q_network           = q_net,
         optimizer           = optimizer,
         td_errors_loss_fn   = common.element_wise_squared_loss,
         train_step_counter  = global_step)

      return agent


   elif which == "ppo":
      actor_net, value_net = create_networks(environment.observation_spec(),
environment.action_spec(), fc_layer_params, conv_layer_params)
```

```
      optimizer =
tf.compat.v1.train.RMSPropOptimizer(learning_rate=learning_rate)
      agent = PPOClipAgent(
         time_step_spec                = environment.time_step_spec(),
         action_spec                  = environment.action_spec(),
         optimizer                   = optimizer,
         actor_net                   = actor_net,
         value_net                   = value_net,
         train_step_counter            = global_step,
            use_gae                      = True,
            importance_ratio_clipping    = 0.2,
            discount_factor              = 0.99
      )
      return agent

""" Make the networks for our PPO agent """
def create_networks(observation_spec, action_spec, fc_layer_params,
conv_layer_params):

   actor_net = ActorDistributionNetwork(
      observation_spec,
      action_spec,
      fc_layer_params     = fc_layer_params,
      conv_layer_params   = conv_layer_params
      )
   value_net = ValueNetwork(
      observation_spec,
      fc_layer_params     = fc_layer_params,
      conv_layer_params   = conv_layer_params
      )

   return actor_net, value_net


import numpy as np
import tf_agents
from tf_agents.policies import py_policy


class HardCodeAgent(py_policy.PyPolicy):
   def __init__(self, time_step_spec, action_spec, goals = [64, 20],
policy_state_spec=()):
      self._time_step_spec = time_step_spec
      self._action_spec = action_spec
      self._policy_state_spec = policy_state_spec
      self.goals = goals

   def _action(self, time_step, policy_state=()):
        # Find the bounding box pixels
      indexes = np.where(time_step.observation[0,:,:,0] == -1)

        # If its in view, get the outer box points
      if indexes[0].size != 0 or indexes[1].size != 0:
         bounding_box_begin = (min(indexes[0]), min(indexes[1]))
         bounding_box_end = (max(indexes[0]), max(indexes[1]))
         bounding_box = (bounding_box_begin, bounding_box_end)

        # Else, keep turning until you do find a bounding box
      else:
         return
tf_agents.trajectories.policy_step.PolicyStep(action=np.array(1), state=(),
info=())
```

```
        # Get the center of the bounding box
      center_box_x        = ((bounding_box[1][1] - bounding_box[0][1])/2) +
bounding_box[0][1]

        # If it centered, check if the bounding box height is correct
      if center_box_x < self.goals[0]+(self.goals[0]*0.2) and center_box_x >
self.goals[0]-(self.goals[0]*0.2):
         bb_height = abs(bounding_box[1][0] - bounding_box[0][0])
         if bb_height < self.goals[1]*0.2 and bb_height > (self.goals[1]-
(self.goals[1]*0.2)):
            action =  0
         elif bb_height < (self.goals[1]-(self.goals[1]*0.2)):
            action =  3
         elif bb_height > self.goals[1]*0.2:
            action =  0

        # In the other two cases, center the person
      elif center_box_x > self.goals[0]+(self.goals[0]*0.2):
         action = 1
      elif center_box_x < self.goals[0]-(self.goals[0]*0.2):
         action = 2
      return
tf_agents.trajectories.policy_step.PolicyStep(action=np.array(action), state=(),
info=())


   def time_step_spec(self):
      return self._time_step_spec

   def action_spec(self):
      return self._action_spec

   def policy_state_spec(self):
      return self._policy_state_spec
import subprocess
import airsim
import numpy as np
import cv2
import os
import json
import time

""" Open the environment on a specific port """
def openAirSim(port, display, multi, environment):
    filename =
r"Auxilaries\Environments\UnrealEnvironments\{}\{}\Binaries\Win64\settings.json"
.format(environment, environment)

    # If we have multiple environments in parallel, we open multiple, each on
different ports
    if multi == True:

        # Open our settings file
        with open(filename, "r") as jsonFile:
            data = json.load(jsonFile)

        # Change the port
        data["ApiServerPort"] += 1
        port = data["ApiServerPort"]

        # Change our ViewMode accordingly
        if display == False:
```

```
                data["ViewMode"] = "NoDisplay"
            else:
                data["ViewMode"] = "SpringArmChase"

            # Save the settings file
            with open(filename, "w") as jsonFile:
                json.dump(data, jsonFile)

            # Open AirSim with these settings

subprocess.call([r'Auxilaries\Environments\UnrealEnvironments\{}\PythonRun.bat'.
format(environment)], shell=True)

            # Connect to the AirSim Drone
            client = airsim.MultirotorClient(port = port)
            client.confirmConnection()
            airsim.DrivetrainType.ForwardOnly
            client.enableApiControl(True)
            client.armDisarm(True)
            client.takeoffAsync().join()

        # With a single environment we only change what is relevant.
        else:
            if port != 41451:
                with open(filename, "r") as jsonFile:
                    data = json.load(jsonFile)

                data["ApiServerPort"] = port

                with open(filename, "w") as jsonFile:
                    json.dump(data, jsonFile)

            if display == False:
                with open(filename, "r") as jsonFile:
                    data = json.load(jsonFile)

                data["ViewMode"] = "NoDisplay"

                with open(filename, "w") as jsonFile:
                    json.dump(data, jsonFile)

            # Start AirSim with given resolution

subprocess.call([r'Auxilaries\Environments\UnrealEnvironments\{}\PythonRun.bat'.
format(environment)], shell=True)

            # Connect to the AirSim Drone
            client = airsim.MultirotorClient(port = port)
            client.confirmConnection()
            airsim.DrivetrainType.ForwardOnly
            client.enableApiControl(True)
            client.armDisarm(True)
            client.takeoffAsync().join()

            # Change it back to the default value
            if port != 41451:
                with open(filename, "r") as jsonFile:
                    data = json.load(jsonFile)

                data["ApiServerPort"] = 41451

                with open(filename, "w") as jsonFile:
                    json.dump(data, jsonFile)
```

```
        if display == False:
            with open(filename, "r") as jsonFile:
                data = json.load(jsonFile)

            data["ViewMode"] = "SpringArmChase"

            with open(filename, "w") as jsonFile:
                json.dump(data, jsonFile)
    return client

""" Draw the bounding for Debug mode """
def drawBoundingBox(image, bb_begin, bb_end, method = "showImageOnly"):

    if method == "showBoundingBox":
        cv2.line(image, bb_begin, (bb_begin[0], bb_end[1]), (0,255,0),
thickness=2)
        cv2.line(image, (bb_begin[0], bb_end[1]), bb_end, (0,255,0),
thickness=2)
        cv2.line(image, (bb_end[0], bb_begin[1]), bb_end, (0,255,0),
thickness=2)
        cv2.line(image, bb_begin, (bb_end[0], bb_begin[1]), (0,255,0),
thickness=2)

    cv2.imshow("image", image)
    cv2.waitKey()

""" Gets a segmentation map and creates a bounding box """
def getBoundingBoxfromSegment(image):

    # Get the pixels with target and create bounding box points (if not empty)
    target_indexes = np.where((image[:,:,0]==73) & (image[:,:,1]==37) &
(image[:,:,2]==226))
    if target_indexes[0].size != 0 or target_indexes[1].size != 0:
        bounding_box_begin  = (min(target_indexes[0]), min(target_indexes[1]))
        bounding_box_end    = (max(target_indexes[0]), max(target_indexes[1]))
        return bounding_box_begin, bounding_box_end
    else:
        return None, None

""" Request the images from AirSim """
def getImagesfromSim(client, depth_imaging = False):

    if depth_imaging == True:

        # Get the depth image and segmentation map
        responses = client.simGetImages([airsim.ImageRequest(0,
airsim.ImageType.DepthVis, True, False),
                                         airsim.ImageRequest(0,
airsim.ImageType.Segmentation, False, False)])

        # Transform both to numpy
        depth               = responses[0]
        depth_image         =
airsim.list_to_2d_float_array(depth.image_data_float, depth.width, depth.height)

        depth_image         = np.clip(depth_image, 0, 100)
        depth_image         = np.divide(depth_image, 100, dtype=np.float32)

        segment             = np.fromstring(responses[1].image_data_uint8,
dtype=np.uint8)
        segment_img_rgb     = segment.reshape(responses[1].height,
responses[0].width, 3)
```

```
        return depth_image, segment_img_rgb

    elif depth_imaging == False:
            # Get the depth image and segmentation map
        responses = client.simGetImages([airsim.ImageRequest(0,
airsim.ImageType.Scene, False, False),
                                         airsim.ImageRequest(0,
airsim.ImageType.Segmentation, False, False)])

        # Process image to grayscale and normalized
        response = responses[0]
        img1d       = np.fromstring(response.image_data_uint8, dtype=np.uint8)
        img_rgb     = img1d.reshape(response.height, response.width, 3)
        img_gray    = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
        img_gray    = np.divide(img_gray, 255, dtype=np.float32)

        # Process segment map
        segment             = np.fromstring(responses[1].image_data_uint8,
dtype=np.uint8)
        segment_img_rgb     = segment.reshape(responses[1].height,
responses[0].width, 3)

        return img_gray, segment_img_rgb

""" Process a single image and boundingbox """
def getSingleImage(client, depth_imaging, BBmethod = "segment"):
    image, segment_img_rgb = getImagesfromSim(client, depth_imaging =
depth_imaging)

    # Get the bounding box according to method
    if BBmethod == "segment":
        bounding_box_begin, bounding_box_end =
getBoundingBoxfromSegment(segment_img_rgb)
    elif BBmethod == "yolo":
        pass

    # Set the pixels to negative values
    if bounding_box_begin is not None or bounding_box_end is not None:
        image[bounding_box_begin[0]:bounding_box_end[0], bounding_box_begin[1]:
bounding_box_end[1]]= -1

    return image, (bounding_box_begin, bounding_box_end)

""" Save the collisions """
def writeCollision(collisionInfo, name):
    if name == 'hard' or name == 'random' or name == "DefaultName":
        filename = r"Auxilaries\TrainedModels\{} - Collisions".format(name)
    else:
        filename = r"Auxilaries\TrainedModels\{}\{} - Collisions".format(name,
name)
    f=open(filename,'a')
    np.savetxt(f, collisionInfo)
    f.close()

""" Save the Reward Distributions """
def writeRewardDistribution(name, array, episode_count):
    if name == 'hard' or name == 'random' or name == "DefaultName":
        filename = r"Auxilaries\TrainedModels\{} -
RewardDistribution".format(name)
    else:
        filename = r"Auxilaries\TrainedModels\{}\{} -
RewardDistribution".format(name, name)
```

```
    np.savetxt(filename, np.append(array, episode_count))

""" Save the paths """
def writePath(positionInfo, name, end):
    if name == 'hard' or name == 'random' or name == "DefaultName":
        filename = r"Auxilaries\TrainedModels\{} - Paths".format(name)
    else:
        filename = r"Auxilaries\TrainedModels\{}\{} - Paths".format(name, name)
    f=open(filename,'a')
    np.savetxt(f, positionInfo)
    f.close()

    if end:
        with open(filename, 'a') as file:
            file.write('---\n')

""" Save the OutofViews """
def writeOutOfView(position, name):
    if name == 'hard' or name == 'random' or name == "DefaultName":
        filename = r"Auxilaries\TrainedModels\{} - OutOfView".format(name)
    else:
        filename = r"Auxilaries\TrainedModels\{}\{} - OutOfView".format(name,
name)
    f=open(filename,'a')
    np.savetxt(f, position)
    f.close()

import airsim
from pyquaternion import Quaternion

""" Control the drone using continuous movements """
def continuousMove(client, given_action, duration, z):

    speed_change_x      = (given_action[0] * 20) - 10
    speed_change_y      = (given_action[1] * 20) - 10
    rotation            = (given_action[2] * 540) - 270

    action = [speed_change_y, speed_change_x, z]

    # Get current motorstate and transform them to quaternion
    q                   = client.simGetVehiclePose().orientation
    my_quaternion       = Quaternion(w_val=q.w_val,x_val=q.x_val,y_val=
q.y_val,z_val=q.z_val)
    mvm                 = my_quaternion.rotate(action)
    velocities          =
client.getMultirotorState().kinematics_estimated.linear_velocity
    donre_vel_rota      = [velocities.x_val , velocities.y_val]

    # Perform the movement
    client.moveByVelocityZAsync(vx         = donre_vel_rota[0] + mvm[0],
                                vy         = donre_vel_rota[1] + mvm[1],
                                z          = z,
                                duration   = duration,
                                drivetrain =
airsim.DrivetrainType.MaxDegreeOfFreedom,
                                yaw_mode   = airsim.YawMode(is_rate = True,
yaw_or_rate = rotation))

""" Go Straight Movement for the Drone """
def straight(client, speed, duration, direction, z):

    typeDrivetrain = airsim.DrivetrainType.MaxDegreeOfFreedom
    if direction == "right":
```

```
        action = [0, speed, 0]
    elif direction == "left":
        action = [0, -speed, 0]
    elif direction == "straight":
        action = [speed, 0, 0]

    # Get current motorstate and transform them to quaternion
    q                   = client.simGetVehiclePose().orientation
    my_quaternion       = Quaternion(w_val=q.w_val,x_val=q.x_val,y_val=
q.y_val,z_val=q.z_val)
    mvm                 = my_quaternion.rotate(action)
    velocities          =
client.getMultirotorState().kinematics_estimated.angular_velocity
    donre_vel_rota      = [velocities.x_val , velocities.y_val]

    # Perform the movement
    client.moveByVelocityZAsync(vx          = donre_vel_rota[0] + mvm[0], #the
already existing speed + the one the agent wants to add, smoother drive?
                                vy          = donre_vel_rota[1] + mvm[1],
                                z           = z,
                                duration    = duration, #will last x
secondes or will be stoped by a new command (put a time.sleep(0.5) next to it)
                                drivetrain  = typeDrivetrain, #the camera is
indepedant of the movement, but the movement is w.r.t the cam orientation
                                yaw_mode    = airsim.YawMode(is_rate = True,
yaw_or_rate = 0)) # True means that yaw_or_rate is seen as a degrees/sec

""" Orient Right for the Drone """
def yaw_right(client, speed, duration):
    client.rotateByYawRateAsync(speed, duration)

""" Orient Left for the Drone """
def yaw_left(client, speed, duration):
    client.rotateByYawRateAsync(-1*speed, duration)
import numpy as np

# Very simple and sparse reward function that looks at whether person is
centered, close to goal height and above certain horizon
def SimpleSparseClose(boundingbox, image_center):

    # If person is not in view, give a negative reward
    if boundingbox[0] is None or boundingbox[1] is None:
        return -1

    # Calculate the center of the bounding box, the permitted margin and the
actual distance of the BB to the center of the image
    center_box_x        = ((boundingbox[1][1] - boundingbox[0][1])/2) +
boundingbox[0][1]
    center_cutoff       = image_center[1]*0.2
    distance_to_centre  = abs(center_box_x - image_center[1])

    # Calculate the BB height, position of the centre of the BB and the
proportion in which this value can fall
    bb_height           = abs(boundingbox[1][0] - boundingbox[0][0])
    bb_center_position  = ((image_center[0] * 2)  - boundingbox[1][0]) +
((abs(boundingbox[1][0] - boundingbox[0][0]))/2)
    proportion          = (bb_center_position / (image_center[0] * 2) )*100

    # Give reward according to the following conditions:
    if distance_to_centre < center_cutoff and bb_height <= 30 and bb_height >=
18  and proportion > 17:
        return 1
    else:
```

```
        return 0

# Same reward as SimpleSparseClose but with tighter margins
def SimpleSparseCloseLessMargin(boundingbox, image_center):

    # If person is not in view, give a negative reward
    if boundingbox[0] is None or boundingbox[1] is None:
        return -1

    # Calculate the center of the bounding box, the permitted margin and the
actual distance of the BB to the center of the image
    center_box_x        = ((boundingbox[1][1] - boundingbox[0][1])/2) +
boundingbox[0][1]
    center_cutoff       = image_center[1]*0.2
    distance_to_centre  = abs(center_box_x - image_center[1])

    # Calculate the BB height, position of the centre of the BB and the
proportion in which this value can fall
    bb_height           = abs(boundingbox[1][0] - boundingbox[0][0])
    bb_center_position  = ((image_center[0] * 2)  - boundingbox[1][0]) +
((abs(boundingbox[1][0] - boundingbox[0][0]))/2)
    proportion          = (bb_center_position / (image_center[0] * 2) )*100

    # Give reward according to the following conditions, this time margins are
tighter:
    if distance_to_centre < center_cutoff and bb_height <= 28 and bb_height >=
24  and proportion > 17:
        return 1
    else:
        return 0


""" Calculate reward based on center of bounding box and height """
def BoundingBoxAndCenter(boundingbox, image_center,  weight_height = 0.8,
bounding_box_ideal_height = 24):

    weight_center = 1 - weight_height

    # If person is not in view, give a negative reward
    if boundingbox[0] is None or boundingbox[1] is None:
        return 0

    else:
        # Weight ascribed to centering the person in the image
        center_box_x        = ((boundingbox[1][1] - boundingbox[0][1])/2) +
boundingbox[0][1]
        center_cutoff       = image_center[1]*0.2
        distance_to_centre  = abs(center_box_x - image_center[1]) -
center_cutoff
        if distance_to_centre <= center_cutoff:
            reward1         = weight_center
        else:
            reward1         = weight_center - (distance_to_centre /
((image_center[1]-center_cutoff) / weight_center))

        # Weight ascribed to the height of the bounding box | aka - distance to
person
        bb_height           = abs(boundingbox[1][0] - boundingbox[0][0])
        bb_goal_cutoff      = bounding_box_ideal_height -
(bounding_box_ideal_height * 0.2)

        if bb_height >= bb_goal_cutoff and bb_height <=
bounding_box_ideal_height:
```

```
            reward2         = weight_height
        else:
            reward2             =  (((-1 * weight_height) * abs(bb_height -
bb_goal_cutoff)) / (bb_goal_cutoff)) + weight_height
        if reward2 < 0:
            reward2 = 0

        reward = reward1 + reward2
        assert reward <= 1 and reward >= 0

        return reward

""" Gets you the right reward during load time """
def getReward(reward):
    if reward == "SimpleSparseClose":
        return SimpleSparseClose
    elif reward == "SimpleSparseCloseLessMargin":
        return SimpleSparseCloseLessMargin
    elif reward == "BoundingBoxAndCenter":
        return BoundingBoxAndCenter

import airsim
import numpy as np
import cv2
import math
import time
import tensorflow as tf
from pyquaternion import Quaternion
import json
import random
import psutil

#tf-agents
import tf_agents.environments as Environments
from tf_agents.specs import array_spec
from tf_agents.trajectories import time_step as ts

# My own modules
import Auxilaries.Environments.Additionals.RewardFunctions as Rewards
import Auxilaries.Environments.Additionals.Movements as Movements
import Auxilaries.Environments.Additionals.AirSimHelpers as AirSimHelpers


class Environment(Environments.py_environment.PyEnvironment):
   def __init__(self,  reward_fn,
                reset_method,
                target          = "rp_carla_rigged_001_Mobile_ue4_5",
                port            = 41451,
                display         = True,
                multi           = True,
                continuous      = False,
                depth_on        = False,
                stack_on        = False,
                environment       = "BlocksNormal",
                evaluation    = 0,
                name          = "DefaultName"
                ):

      # Open airsim, unles evaluation mode = 4, then we just connect directly
      if evaluation != 4:
         self.client         = AirSimHelpers.openAirSim(port, display, multi,
environment)
      else:
```

```
            client = airsim.MultirotorClient()
            client.confirmConnection()
            airsim.DrivetrainType.ForwardOnly
            client.enableApiControl(True)
            client.armDisarm(True)
            client.takeoffAsync().join()

        self.depth_on       = depth_on
        self.stack_on       = stack_on
        self.continuous     = continuous
        self.target_name    = target
        self.reward_fn      = reward_fn
        self.reset_method   = reset_method
        self.evaluation    = evaluation
        self.environment   = environment
        self.name        = name

        # Set the color of our target object in the segmentation map so we can
find it later in our segmentation maps
        success = self.client.simSetSegmentationObjectID(target, 12, True)
        print("Setting ID for Segmentation Map: ", success)

        # Get state
        self._state, box  = self.getState(depth_imaging = self.depth_on,
stacked_imaging = self.stack_on)

        # Set the tensor specs for actions and observations
        if continuous:
            self._action_spec = array_spec.BoundedArraySpec(shape=(3,),
dtype=np.float32, minimum=0, maximum=1, name = 'action')
        else:
            self._action_spec = array_spec.BoundedArraySpec(shape=(),
dtype=np.int32, minimum=0, maximum=5, name = 'action')

        if stack_on:
            self._observation_spec = array_spec.BoundedArraySpec(
                shape=(self._state.shape[0], self._state.shape[1],
self._state.shape[2]), dtype=np.float32, minimum=-1, maximum=1,
name='observation')
        else:
            self._observation_spec = array_spec.BoundedArraySpec(
                shape=(self._state.shape[0], self._state.shape[1], 1),
dtype=np.float32, minimum=-1, maximum=1, name='observation')

        # Set overall environment variables
        self._episode_ended             = False
        self.z                          = -2
        self.reward_distribution      = np.zeros(50)
        self.episode_count          = 0
        self.reset()
        self.timestamp                  = 0
        self.time_end                   = False
        self.steps                      = 0
        self.beginning                  = True
        self.steps_out_of_view          = 0
        self.pose                       = self.client.simGetVehiclePose()
        self.pose.position.z_val        = self.z
        self.current_bounding_box       = box
        self.image_center               = (self._state.shape[0] / 2,
self._state.shape[1] / 2)

    """ OVERRIDDEN: Reset the drone to the original state """
    def _reset(self):
```

```
        # Reset the drone to the origin position
        if self.reset_method == "OriginalPlace":
            self.client.reset()
            self.client.simSetVehiclePose(self.pose, ignore_collison=False)
            self.client.enableApiControl(True)
            self.client.armDisarm(True)
            self.client.takeoffAsync()
            self.client.hoverAsync().join()

        # Reset the drone to a random position around the person
        elif self.reset_method == "RandomPlaceAround":
            self.client.reset()
            self.client.simSetVehiclePose(self.pose, ignore_collison=False)

            # Get coordinates of drone and person
            pose                = self.client.simGetVehiclePose()
            person              = self.client.simGetObjectPose(self.target_name)

            # Set a new position
            pose.position.x_val = person.position.x_val + ([-
1,1][random.randrange(2)] * random.uniform(5, 10))
            pose.position.y_val = person.position.y_val + ([-
1,1][random.randrange(2)] * random.uniform(5, 10))
            pose.position.z_val = self.z

            # Calculate new angle to look at
            differences         = [person.position.x_val - pose.position.x_val,
person.position.y_val - pose.position.y_val]
            new_quat            =
np.array(airsim.to_eularian_angles(pose.orientation))
            new_quat[2]         = np.arctan((differences[1]/differences[0]))

            # Correction for inversed position
            if differences[0] < 0:
                new_quat[2]         += math.pi

            # Move the drone to position and rotation
            pose.orientation    = airsim.to_quaternion(new_quat[0], new_quat[1],
new_quat[2])
            self.client.simSetVehiclePose(pose, ignore_collison=False)
            self.client.enableApiControl(True)
            self.client.armDisarm(True)
            self.client.takeoffAsync()
            self.client.hoverAsync().join()

        # Reset the drone directly behind the person facing the person
        elif self.reset_method == "DirectlyBehind":

            # Get coordinates of drone and person
            pose                = self.client.simGetVehiclePose()
            person              = self.client.simGetObjectPose(self.target_name)

            # Correct the Eulerian angle and calculate the new position
            angle               = airsim.to_eularian_angles(person.orientation)[2]
            if angle < 0:
                angle += math.pi
            else:
                angle = angle-math.pi
            change_x          = 4 * np.sin(angle)
            change_y          = 4 * np.cos(angle)

            # Set the new position
```

```
            pose.position.x_val = person.position.x_val - change_x
            pose.position.y_val = person.position.y_val + change_y
            pose.position.z_val = self.z

            # Calculate new angle to look at
            differences         = [person.position.x_val - pose.position.x_val,
person.position.y_val - pose.position.y_val]
            new_quat            =
np.array(airsim.to_eularian_angles(pose.orientation))
            new_quat[2]         = np.arctan((differences[1]/differences[0]))

            # Correction for inversed position
            if differences[0] < 0:
               new_quat[2]          += math.pi

            # Move the drone to position and rotation and reconnect
            pose.orientation    = airsim.to_quaternion(new_quat[0], new_quat[1],
new_quat[2])
            self.client.simSetVehiclePose(pose, ignore_collison=True)
            self.client.enableApiControl(True)
            self.client.armDisarm(True)
            self.client.takeoffAsync()
            self.client.hoverAsync().join()

         # Reset the environment variables
         self._episode_ended     = False
         self.steps              = 0
         self.beginning          = True
         self.steps_out_of_view  = 0
         self.episode_count     += 1

         # Write the reward distributions
         if self.evaluation != 0 and self.evaluation != 2:
            AirSimHelpers.writeRewardDistribution(self.name,
self.reward_distribution, self.episode_count)

         # Return final timestep
         return ts.restart(np.array(self._state, dtype=np.float32))

      """ Just reset position, without affecting the episode """
      def ensureHeight(self):
         height = self.client.simGetVehiclePose().position.z_val
         pose = self.client.simGetObjectPose(self.target_name)
         if height > self.z + 0.3 or height < self.z - 0.3:
            pose.position.z_val = self.z
            self.client.simSetVehiclePose(self.pose, ignore_collison=False)
            self.client.enableApiControl(True)
            self.client.armDisarm(True)
            self.client.takeoffAsync()
            self.client.hoverAsync().join()

      """ OVERRIDDEN: Perform an action, get a new state and calculate reward """
      def _step(self, action):

         # Make sure the drone is at the correct height
         self.ensureHeight()

         # If the episode is done, reset the environment
         if self._episode_ended:
            return self.reset()

         # Perform the chosen move and see if there was a collision (if so, end the
episode)
```

```
      self._episode_ended = self.move(action = action, continuous =
self.continuous)

      # Get the new state
      self._state, bounding_box = self.getState(depth_imaging=self.depth_on,
stacked_imaging=self.stack_on)
      self.current_bounding_box = bounding_box

      # Determine reward
      reward = self.reward_fn(bounding_box, self.image_center)
      if self.evaluation != 0 and self.evaluation != 2:
         self.reward_distribution[self.steps]   += reward

      # Count steps
      self.steps += 1

      # Check whether this is the first time that the person is out of view and
keep track
      if bounding_box[0] is None or bounding_box[1] is None:
         self.steps_out_of_view += 1
      else:
         self.steps_out_of_view = 0

      # Call episode end if person is out of view for too long or reach our
limit
      if  (self.steps_out_of_view >= 1 or self.steps >= 50) and self.evaluation
!= 2:
         self._episode_ended = True
         self.time_end = True

      if self.evaluation != 0:

AirSimHelpers.writePath([self.client.simGetVehiclePose().position.x_val,
self.client.simGetVehiclePose().position.y_val], self.name, end =
self._episode_ended)

      # If we're at the end of an episode, which means we've hit a collision,
give a horrible reward and terminate
      if self._episode_ended:

         # If termination was not due to timeout, give corresponding reward
         if self.time_end == False or self.steps_out_of_view >= 1:
            reward = -1
            if self.evaluation != 0:

AirSimHelpers.writeOutOfView([self.client.simGetVehiclePose().position.x_val,
self.client.simGetVehiclePose().position.y_val], self.name)

         return ts.termination(np.array(self._state, dtype=np.float32), reward)

      # If were still going on we transition to the next state
      else:
         return ts.transition(np.array(self._state, dtype=np.float32),
reward=reward, discount=0.9)

   """ Take a move in the world """
   def move(self, action, duration = 0.1, continuous = False):

      # Discrete actions
      if continuous == False:
         self.client.hoverAsync()

         if self.client.simGetCollisionInfo().has_collided:
```

```
            if self.evaluation != 0:
AirSimHelpers.writeCollision([self.client.simGetCollisionInfo().impact_point.x_v
al, self.client.simGetCollisionInfo().impact_point.y_val], self.name)
            return True

         # Do nothing
         if action == 0:
            self.client.moveByVelocityAsync(0, 0, 0, 1)
            self.client.rotateByYawRateAsync(0, 1)
            if self.client.simGetCollisionInfo().has_collided:
               return True

         # Orient Right
         if action == 1:
            Movements.yaw_right(self.client, 50, 0.1)
            if self.client.simGetCollisionInfo().has_collided:
               return True

         # Orient Left
         if action == 2:
            Movements.yaw_left(self.client, 50, 0.1)
            if self.client.simGetCollisionInfo().has_collided:
               return True

         # Go straight
         if action == 3:
            Movements.straight(self.client, 6, duration, "straight", self.z)
            if self.client.simGetCollisionInfo().has_collided:
               return True

         # Go right
         if action == 4:
            Movements.straight(self.client, 6, duration, "right", self.z)
            if self.client.simGetCollisionInfo().has_collided:
               return True

         # Go left
         if action == 5:
            Movements.straight(self.client, 6, duration, "left", self.z)
            if self.client.simGetCollisionInfo().has_collided:
               return True

         return False

      # Continuous Movements
      else:
         if self.client.simGetCollisionInfo().has_collided:
            return True
         Movements.continuousMove(self.client, action, duration, self.z)
         if self.client.simGetCollisionInfo().has_collided:
               return True

   """ Get the state according to the preferred method """
   def getState(self, depth_imaging, stacked_imaging, BBmethod = "segment",
image_set_size = 3):

      # Single images
      if not stacked_imaging:
         time.sleep(0.4)
         image, bounding_box = AirSimHelpers.getSingleImage(self.client,
depth_imaging=self.depth_on, BBmethod=BBmethod)
         return np.expand_dims(image, axis=2), bounding_box
```

```
      # Stacked images
      else:
         interval = 0.1
         image_set = []

         for _ in range(image_set_size):
            image, bounding_box = AirSimHelpers.getSingleImage(self.client,
depth_imaging=self.depth_on, BBmethod=BBmethod)

            # Collect the image and wait
            image_set.append(image)
            time.sleep(interval)

         # Stack images into one object and return latest bounding box
         images = np.stack(image_set, axis = 2)
         return images, bounding_box

   def action_spec(self):
      return self._action_spec

   def observation_spec(self):
      return self._observation_spec

   def setPause(self):
      self.client.simPause(True)

   def unPause(self):
      self.client.simPause(False)

   def Kill(self, processName):
      #Iterate over the all the running process
      for proc in psutil.process_iter():
         try:
            # Check if process name contains the given name string.
            if processName.lower() in proc.name().lower():
               proc.kill()
               return True
         except (psutil.NoSuchProcess, psutil.AccessDenied,
psutil.ZombieProcess):
            pass
      return False;
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
from tf_agents.utils import common
from tf_agents.policies import policy_saver
from tf_agents.metrics import tf_metrics
import numpy as np

""" My own print statement """
def printStatusStatement(text):
  print("\n \n---------------------------- ", text, " -------------------------
--\n \n")

""" Limit Tensorflow allocation space on GPU """
def setGPUtoFraction(fraction = 0.666):
    config = ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 0.666
    session = InteractiveSession(config=config)

""" Create the checkpointers so the models can be saved """
def getCheckpointer(agent, replay_buffer, global_step, directory):
```

```
  # Checkpoint initialization
  checkpoint_dir = directory + '/checkpoints'
  checkpointer = common.Checkpointer(
      ckpt_dir        = checkpoint_dir,
      max_to_keep     = 1,
      agent           = agent,
      policy          = agent.policy,
      replay_buffer   = replay_buffer,
      global_step     = global_step
  )

  # Policy Initialization
  policy_save = policy_saver.PolicySaver(agent.policy)
  return checkpointer, policy_save

""" Create the required metrics for the type of learning that will happen """
def getMetrics(mdc, agent, num_batch):
  assert agent == "dqn" or agent == "ppo"

  if mdc == True:
    train_average_return                =
tf_metrics.AverageReturnMetric(batch_size=num_batch)
    train_average_episode_length        =
tf_metrics.AverageEpisodeLengthMetric(batch_size=num_batch)
    eval_average_return                 =
tf_metrics.AverageReturnMetric(batch_size=num_batch)
    eval_max_return                     =
tf_metrics.MaxReturnMetric(batch_size=num_batch)
    eval_min_return                     =
tf_metrics.MinReturnMetric(batch_size=num_batch)

    train_metrics   = [train_average_return, train_average_episode_length]
    eval_metrics    = [eval_average_return, eval_max_return, eval_min_return]
    metrics         = [train_metrics, eval_metrics]

    if agent == 'dqn':
      train_metrics.append(tf_metrics.MaxReturnMetric(batch_size=num_batch))
      train_metrics.append(tf_metrics.MinReturnMetric(batch_size=num_batch))

  else:
    train_average_return            = tf_metrics.AverageReturnMetric()
    train_average_episode_length    = tf_metrics.AverageEpisodeLengthMetric()
    eval_average_return             = tf_metrics.AverageReturnMetric()
    eval_max_return                 = tf_metrics.MaxReturnMetric()
    eval_min_return                 = tf_metrics.MinReturnMetric()

    train_metrics    = [train_average_return, train_average_episode_length]
    eval_metrics     = [eval_average_return, eval_max_return, eval_min_return]
    metrics          = [train_metrics, eval_metrics]

    if agent == 'dqn':
      train_metrics.append(tf_metrics.MaxReturnMetric())
      train_metrics.append(tf_metrics.MinReturnMetric())

  return metrics

""" Update Tensorboard during training time """
def updateTensorboardTrain(agent, board, metrics, loss, epoch):
  assert agent == "dqn" or agent == "ppo"

  for metric in metrics[0]:
    if metric.result() >= 2e6 or metric.result() <= -2e6:
      metric.reset()
```

```
  if agent == 'ppo':

    board.update_stats( TrainingLoss            = round(float(loss),2),
                        TrainAverageReturn      =
round(float(metrics[0][0].result()), 2),
                        TrainAVGEpisodeLength   = int(metrics[0][1].result()),
step = epoch)
    for metric in metrics[0]:
      metric.reset()

  if agent == "dqn":
    board.update_stats( TrainingLoss            = round(float(loss),2),
                        TrainAverageReturn      =
round(float(metrics[0][0].result()),2),
                        TrainAVGEpisodeLength   = int(metrics[0][1].result()),
                        TrainMaxReturn          = int(metrics[0][2].result()),
                        TrainMinReturn          = int(metrics[0][3].result()),
                        step = epoch)

""" Update Tensorboard during test time """
def updateTensorboardEval(board, metrics, epoch):

  board.update_stats(EvalAverageReturn          = int(metrics[1][0].result()),
                                EvalMaxReturn   = int(metrics[1][1].result()),
                                EvalMinReturn   = int(metrics[1][2].result()),
                                step = epoch)
  for metric in metrics[1]:
      metric.reset()

""" Creates a dictionary from a file """
def transformToDict(filename):
  f = open(filename, 'r')
  lines = f.readlines()[0:-1]
  dictionary = {}
  for line in lines:
    splits = line.split(":")
    if splits[0] != "Reward Function":
      new_value = splits[1].replace("\n", "").replace(" ", "")
    else:
      new_value = splits[1].replace("\n", "").split()[1]
    dictionary[splits[0]] = new_value
  return dictionary
import tensorflow as tf
from keras.callbacks import TensorBoard
import os

# Own Tensorboard class stolen from SentDex (https://pythonprogramming.net/deep-
q-learning-dqn-reinforcement-learning-python-tutorial/)

class ModifiedTensorBoard(TensorBoard):
    # Overriding init to set initial step and writer (we want one log file for
all .fit() calls)
    def __init__(self, name, **kwargs):
        super().__init__(**kwargs)
        self.step = 1
        self.writer = tf.summary.create_file_writer(self.log_dir)
        self._log_write_dir = os.path.join(self.log_dir, name)

    # Overriding this method to stop creating default log writer
    def set_model(self, model):
        pass
```

```
    # Overrided, saves logs with our step number
    # (otherwise every .fit() will start writing from 0th step)
    def on_epoch_end(self, epoch, logs=None):
        self.update_stats(**logs)

    # Overrided
    # We train for one batch only, no need to save anything at epoch end
    def on_batch_end(self, batch, logs=None):
        pass

    # Overrided, so won't close writer
    def on_train_end(self, _):
        pass

    def on_train_batch_end(self, batch, logs=None):
        pass

    # Custom method for saving own metrics
    # Creates writer, writes custom metrics and closes writer
    def update_stats(self, step, **stats):
        self._write_logs(stats, step)

    def _write_logs(self, logs, index):
        with self.writer.as_default():
            for name, value in logs.items():
                tf.summary.scalar(name, value, step=index)
                self.writer.flush()
class Agent_Type():
    PPO_Agent       = "ppo"
    DQN_Agent       = "dqn"
    HARD_Agent      = "hard"
    RANDOM_Agent    = "random"

class Reset_Method():
    World_Initial_State     = "OriginalPlace"
    Random_Around_Person    = "RandomPlaceAround"
    Directly_Behind_Person  = "DirectlyBehind"

class Environments():
    BlocksNormal        = "BlocksNormal"
    BlocksObstacles     = "BlocksObstacles"
    FactoryTest         = "Factory"
    FullList            = ["BlocksNormal", "BlocksObstacles", "Factory"]
import tensorflow as tf
from tf_agents.environments import tf_py_environment
from Auxilaries.Environments.Environment import Environment
from tf_agents.drivers import dynamic_episode_driver
from tf_agents.drivers import dynamic_step_driver
import Auxilaries.Environments.Additionals.RewardFunctions as Rewards
import Auxilaries.Extras.Parameters as Parameters
from Auxilaries.Agents.hard_code_agent import HardCodeAgent
import Auxilaries.Extras.AdditionalMethods as Extras
from tf_agents.policies import random_tf_policy
from tf_agents.metrics import tf_metrics
import json
import numpy as np
import sys
import os
import time

def evaluate(EXPERIMENT = Parameters.Agent_Type.HARD_Agent, episodes = 10):
    with tf.device('/CPU:0'):
```

```
        """ Chooser menu for which model to evaluate """
        print("Available Models to Evaluate:")
        models = np.concatenate((["hard", "random"],
os.listdir("Auxilaries/TrainedModels")))
        for x in range(len(models)):
            print(x, models[x])

        model_found = False
        while not model_found:
            try:
                model = int(input("Copy the model you want to evaluate: "))
                model_found = True
            except:
                print("You can't use this try again....")

        if model != '':
            EXPERIMENT = models[model]
        else:
            EXPERIMENT = 'hard'



        """ Load in the required parameters to use for evaluation """
        parameters = {      'Reward Function'   : Rewards.SimpleSparseClose,
                            'Reset Method'      :
Parameters.Reset_Method.Directly_Behind_Person,
                            'Stacked Input'     : True,
                            'Depth Input'       : True }

        if EXPERIMENT != 'hard' and EXPERIMENT != 'random':
            parameters = Extras.transformToDict('Auxilaries/TrainedModels/' +
EXPERIMENT +'/parameters.txt')
            parameters['Reward Function'] = Rewards.getReward(parameters['Reward
Function'])
            print(parameters)



        """ Open environment """
        env = Environment(  reward_fn       = parameters['Reward Function'],
                            reset_method    = parameters['Reset Method'] ,
                            stack_on        = parameters['Stacked Input'] ==
"True",
                            depth_on        = parameters['Depth Input'] ==
"True",
                            environment     = "Factory",
                            evaluation      = 1,
                            name            = EXPERIMENT,
                            target          =
"rp_manuel_rigged_001_Mobile_ue4_2"
                          )
        environment = tf_py_environment.TFPyEnvironment(env)



        """ Load the corresponding model """
        if EXPERIMENT == 'hard':
            policy = HardCodeAgent(environment.time_step_spec(),
environment.action_spec())
        elif EXPERIMENT == 'random':
            policy =
random_tf_policy.RandomTFPolicy(action_spec=environment.action_spec(),
```

```
time_step_spec=environment.time_step_spec())
        else:
            policy_dir  = 'Auxilaries/TrainedModels/' + EXPERIMENT +'/policies'
            policy      = tf.compat.v2.saved_model.load(policy_dir)



        """ Perform the evaluation """
        if env.evaluation != 2:
            average_return            = tf_metrics.AverageReturnMetric()
            driver = dynamic_episode_driver.DynamicEpisodeDriver(environment,
                                                                 policy,
                                                                 [average_return],
                                                                 num_episodes=
int(episodes))
            start = time.time()
            driver.run()
            print("Total: ", average_return.result())
            print("Performed in:   ", time.time() - start, " seconds")
        else:
            average_return = []
            for x in range(int(episodes)):
                time_step = environment.current_time_step()
                action_step = policy.action(time_step)
                next_time_step = environment.step(action_step.action)
                #print("Ep reward: ", int(next_time_step.reward))
                average_return.append(int(next_time_step.reward))
            print("Total: ", np.sum(average_return))
            print("Avg: ", np.mean(average_return))

        """ Kill """
        env.Kill("Blocks")



if __name__ == '__main__':

    if len(sys.argv) > 1:
        evaluate(episodes = sys.argv[1])
    else:
        evaluate()

import numpy as np
import time
import tensorflow as tf
import os
import subprocess
import keyboard

# My modules
from Auxilaries.Extras import AdditionalMethods as extras
from Auxilaries.Environments.Environment import Environment
import Auxilaries.Extras.Parameters as Parameters
import Auxilaries.Environments.Additionals.RewardFunctions as Rewards


""" Open environment """
env = Environment(  reward_fn       = Rewards.SimpleSparseClose,
                    reset_method    =
Parameters.Reset_Method.Directly_Behind_Person,
                    stack_on        = True,
                    depth_on        = True,
                    environment     = "BlocksNormal",
```

```
                    evaluation      = 1,
                    #target          = "rp_manuel_rigged_001_Mobile_ue4_5"
                )


time_step = env.reset()
rewards = []
counter = 0


while counter >= 0:
   if keyboard.is_pressed('q'):
      time_step = env.step(2)
   elif keyboard.is_pressed('e'):
      time_step = env.step(1)
   elif keyboard.is_pressed('w'):
      time_step = env.step(3)
   elif keyboard.is_pressed('d'):
      time_step = env.step(4)
   elif keyboard.is_pressed('a'):
      time_step = env.step(5)
   elif keyboard.is_pressed('escape'):
      break
   else:
      time_step = env.step(0)
   counter += 1

   #print(time_step.reward)



import tf_agents
import random
import tensorflow as tf
import sys

from Auxilaries.Environments.Environment import Environment
import Auxilaries.Environments.Additionals.RewardFunctions as Rewards
import Auxilaries.Extras.Parameters as Parameters
from Auxilaries.Extras import AdditionalMethods as extras
from train import train

from numpy.random import seed


if __name__ == '__main__':

    # Setup GPU
    extras.setGPUtoFraction(0.75)
    tf_agents.system.multiprocessing.enable_interactive_mode()

    # Parameters and Experiments
    global_parameters = {   "Learning Rate"             : 1e-3,
                            "Reward Function"           :
Rewards.SimpleSparseClose,
                            "Reset Method"              :
Parameters.Reset_Method.Directly_Behind_Person,
                            "Epochs per Experiment"     : 2000}

    environment_list = Parameters.Environments.FullList
    if len(sys.argv) > 1:
        environment_list = [sys.argv[1]]
```

```
    for environment in environment_list:
        experiments = {
                        environment + " Run 3 - DQN nostack-normal" :{"Agent
Type" : Parameters.Agent_Type.DQN_Agent,
                                                                      "Stacked Input"
: True,
                                                                      "Depth Input"
: False,
                                                                      "Environment"
: environment}
        }

        random.seed(1)
        seed(1)
        tf.compat.v1.set_random_seed(1)


        # Perform Experiments
        for experiment in experiments:
            # Check if this experiment has been run at all. If not: run it
            try:
                my_file =
open("Auxilaries/TrainedModels/"+experiment+"/parameters.txt")
            except:
                train({**experiments[experiment], **global_parameters},
experiment)

            my_file =
open("Auxilaries/TrainedModels/"+experiment+"/parameters.txt")
            # Check if this experiment has been finished, if not: continue
            last_line = my_file.readlines()[-1]
            if last_line == "Done!":
                continue
            else:
                continue_from_epoch = int(last_line)
                train({**experiments[experiment], **global_parameters},
experiment, CONTINUE=True, ALREADY_PERFORMED_EPOCHS=continue_from_epoch)

import numpy as np
import time
import tensorflow as tf
import os
import subprocess
import pdb
import json

# TFAgents Modules
import tf_agents
from tf_agents.drivers import dynamic_step_driver
from tf_agents.drivers import dynamic_episode_driver
from tf_agents.environments import suite_gym
from tf_agents.environments import tf_py_environment
from tf_agents.environments import parallel_py_environment
from tf_agents.policies import random_tf_policy
from tf_agents.policies import policy_saver
from tf_agents.eval import metric_utils
from tf_agents.metrics import tf_metrics
from tf_agents.replay_buffers import tf_uniform_replay_buffer
from tf_agents.trajectories import trajectory
from tf_agents.utils import common

# My modules
from Auxilaries.Extras import AdditionalMethods as extras
```

```
from Auxilaries.Extras.CustomTensorBoard import ModifiedTensorBoard
from Auxilaries.Environments.Environment import Environment
from Auxilaries.Agents import get_DQNorPPO as Agents
import Auxilaries.Environments.Additionals.RewardFunctions as Rewards
import Auxilaries.Extras.Parameters as Parameters


# Wrapper Definition
def train(PARAMETERS, EXPERIMENT_NAME, CONTINUE = False,
ALREADY_PERFORMED_EPOCHS=0):

    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

    if PARAMETERS["Agent Type"] == 'hard' or PARAMETERS["Agent Type"] ==
'random':
        print("You can't use this agent to train")

    with tf.device('/CPU:0'):
        """ ------- Parameters ------ """
        MULTI_DATA_COLLECTION           = False
        NUM_PARALLEL_ENVIRONMENTS       = 1
        EVAL_INTERVAL                   = 50
        MODELS_SAVED_TOTAL              = 5
        SAVE_DIRECTORY                  = "Auxilaries/TrainedModels/" +
EXPERIMENT_NAME

        # DQN
        DATA_COLLECTION_STEPS           = 500
        DATA_COLLECTION_STEPS_PER_EPOCH = 50
        REPLAY_BUFFER_MAX_LENGTH        = 5_000
        BATCH_SIZE                      = 64

        # PPO
        if PARAMETERS['Agent Type'] == "ppo":
            REPLAY_BUFFER_MAX_LENGTH    = 50


        """-------------------- Create Environment --------------------"""


        # Create environment(s) and wrap them in TF Environments class
        if MULTI_DATA_COLLECTION == True:
            envs = [Environment] * NUM_PARALLEL_ENVIRONMENTS
            environment =
tf_py_environment.TFPyEnvironment(parallel_py_environment.ParallelPyEnvironment(
envs))
        else:
            env = Environment(  reward_fn       =PARAMETERS["Reward Function"],
                                reset_method    =PARAMETERS["Reset Method"],
                                depth_on        =PARAMETERS["Depth Input"],
                                stack_on        =PARAMETERS["Stacked Input"],
                                environment     =PARAMETERS["Environment"],
                                name            =EXPERIMENT_NAME
                                )
            environment = tf_py_environment.TFPyEnvironment(env)


        """-------------------- Agent Creation --------------------"""
        global_step = tf.compat.v1.train.get_or_create_global_step()

        agent = Agents.getAgent(environment, PARAMETERS["Learning Rate"],
global_step, PARAMETERS["Agent Type"])
```

```
        agent.initialize()


        """-------------------- Replay Buffer --------------------"""
        replay_buffer = tf_uniform_replay_buffer.TFUniformReplayBuffer(
            data_spec       = agent.collect_data_spec,
            batch_size      = NUM_PARALLEL_ENVIRONMENTS,
            max_length      = REPLAY_BUFFER_MAX_LENGTH)


        """-------------------- Checkpoints --------------------"""
        checkpointer, policy_save   = extras.getCheckpointer(agent,
replay_buffer, global_step, SAVE_DIRECTORY)
        policy_dir                  = SAVE_DIRECTORY + "/policies"


        """-------------------- Metrics --------------------"""

        metrics = extras.getMetrics(MULTI_DATA_COLLECTION, PARAMETERS['Agent
Type'], NUM_PARALLEL_ENVIRONMENTS)
        observer = [replay_buffer.add_batch]

        # Setup Tensorboard
        name = "event" + EXPERIMENT_NAME
        board = ModifiedTensorBoard(name,
log_dir=f"Auxilaries/TensorBoard/logs/{EXPERIMENT_NAME}")
        extras.printStatusStatement("Everything is ready! Start the Training
Cycle")

        if not CONTINUE:
            with open(SAVE_DIRECTORY + '\parameters.txt','w') as f:
                f.writelines('{}:   {} \n'.format(k,v) for k, v in
PARAMETERS.items())
                f.write('\n')

            if PARAMETERS["Agent Type"] == "dqn":
                stringlist = []
                agent._q_network.layers[0].summary(print_fn=lambda x:
stringlist.append(x))
                short_model_summary = "\n".join(stringlist)
                print("Training agent with network: ",
agent._q_network.layers[0].summary())
                with open(SAVE_DIRECTORY + '\\network_architectures.txt','w') as
f:
                        f.write(short_model_summary)
        else:
            checkpointer.initialize_or_restore()
            global_step = tf.compat.v1.train.get_global_step()



        """-------------------- Driver --------------------"""

        if PARAMETERS["Agent Type"] == "dqn":
            # Random Policy for data collection
            random_policy =
random_tf_policy.RandomTFPolicy(environment.time_step_spec(),

environment.action_spec())

            # Setup RL Driver for Initial Data Collection
            collect_driver = dynamic_step_driver.DynamicStepDriver(environment,
```

```
                                                                random_policy,
                                                                observer,

num_steps=DATA_COLLECTION_STEPS)

            # Fill our Replay Buffer with random experiences
            start = time.time()
            collect_driver.run()
            print("Time was ", time.time() - start, " seconds ")

            #Setup RL Driver for Training Data Collection
            train_driver = dynamic_step_driver.DynamicStepDriver(environment,

agent.collect_policy,
                                                                observer +
metrics[0],

num_steps=DATA_COLLECTION_STEPS_PER_EPOCH)

            eval_driver =
dynamic_episode_driver.DynamicEpisodeDriver(environment,
                                                                agent.policy,
                                                                metrics[1],
                                                                num_episodes= 10)

            # Transform Replay Buffer to Dataset
            dataset = replay_buffer.as_dataset(
                num_parallel_calls=3,
                sample_batch_size=BATCH_SIZE,
                num_steps=2).prefetch(3)
            iterator = iter(dataset)

        elif PARAMETERS['Agent Type'] == "ppo":

            # Create a driver for training and a driver for evaluation
            train_driver = dynamic_step_driver.DynamicStepDriver(environment,

agent.collect_policy,
                                                                observer +
metrics[0],

num_steps=int(REPLAY_BUFFER_MAX_LENGTH/NUM_PARALLEL_ENVIRONMENTS))

            eval_driver =
dynamic_episode_driver.DynamicEpisodeDriver(environment,
                                                                agent.policy,
                                                                metrics[1],
                                                                num_episodes= 10)
            #`as_dataset(..., single_deterministic_pass=True)` instead.



        """-------------------- Training --------------------"""
        # Optimize by wrapping some of the code in a graph using TF function.
        agent.train = common.function(agent.train)
        agent.train_step_counter.assign(0)


        # Training Loop
        for epoch in range(ALREADY_PERFORMED_EPOCHS, PARAMETERS["Epochs per
Experiment"]):
            print("Epoch: ", epoch, end=' ')
```

```
            # Collect a few steps using collect_policy and save to the replay
buffer.
            if not MULTI_DATA_COLLECTION:
                env.unPause()

            train_driver.run()

            if not MULTI_DATA_COLLECTION:
                env.setPause()

            # Sample a batch of data from the buffer and update the agent's
network.
            if PARAMETERS['Agent Type'] == 'dqn':
                experience, unused_info = next(iterator)
            elif PARAMETERS['Agent Type'] == 'ppo':
                experience = replay_buffer.gather_all()
                replay_buffer.clear()

            # Train on GPU
            with tf.device('/GPU:0'):
                print("Train" , end=' ')
                train_loss = agent.train(experience).loss
                step = agent.train_step_counter.numpy()

            # Update training metrics
            extras.updateTensorboardTrain(PARAMETERS["Agent Type"], board,
metrics, train_loss, epoch)

            # Perform evaluation step
            if epoch % EVAL_INTERVAL == 0 and epoch != 0:
                if not MULTI_DATA_COLLECTION:
                    env.unPause()
                eval_driver.run()
                extras.updateTensorboardEval(board, metrics, epoch)
                print("Evaluated", end=' ')

            # Save checkpoint
            if epoch % int(PARAMETERS["Epochs per
Experiment"]/MODELS_SAVED_TOTAL) == 0 and epoch != 0:
                policy_save.save(policy_dir)
                print("SavedPolicy", end=' ')

            # Save this model in case of emergency
            checkpointer.save(global_step)

            # Keep track of how far we are
            lines = open(SAVE_DIRECTORY + '\parameters.txt','r').readlines()
            lines[-1] = str(epoch+1)
            open(SAVE_DIRECTORY + '\parameters.txt','w').writelines(lines)
            print("")


        """-------------------- Save -------------------- """
        # Save Policy
        print("Finished! Now saving....", end= ' ')
        policy_save.save(policy_dir)
        print("All done!")

        # End the simulation
        env.Kill("Blocks")

        # Mark this experiment as done
        lines = open(SAVE_DIRECTORY + '\parameters.txt','r').readlines()
```

```
        lines[-1] = "Done!"
        open(SAVE_DIRECTORY + '\parameters.txt','w').writelines(lines)


if __name__ == '__main__':
    extras.setGPUtoFraction(0.75)
    tf_agents.system.multiprocessing.enable_interactive_mode()


    default_parameters = {  "Learning Rate"             : 1e-3,
                            "Reward Function"           :
Rewards.SimpleSparseClose,
                            "Reset Method"              :
Parameters.Reset_Method.Directly_Behind_Person,
                            "Epochs per Experiment"     : 100,
                            "Agent Type"                :
Parameters.Agent_Type.PPO_Agent,
                            "Stacked Input"             : True,
                            "Depth Input"               : True,
                            "Environment"               : 'BlocksObstacles'
                         }

    train(default_parameters, "Default")


#%%

import numpy as np
import cv2
import matplotlib.pyplot as plt

#%%

environment             = "BlocksObstacles"

#%%

f           = open("data/{}/BlocksObstacles Run 1 newreward - DQN stack-
depth(stopped) - Collisions".format(environment))

#%%

f               = np.loadtxt('data/{}/BlocksObstacles Run 1 newreward - DQN
stack-depth(stopped) - OutofView'.format(environment))
y_values        = f[[i for i in range(len(f)) if i % 2 == 1]]
x_values        = f[[i for i in range(len(f)) if i % 2 == 0]]
list_of_values  = [[x_values, y_values]]

#%%

uneven          = 0
new_line        = []
new_line.append([])
new_line.append([])
list_of_values  = []
list_of_values.append(new_line)
x_index         = 0

for x in f:
    if x == "---" or x == "---\n":
        new_line = []
        new_line.append([])
        new_line.append([])
```

```
            list_of_values.append(new_line)
            x_index += 1
        else:
            list_of_values[x_index][uneven].append(float(x))
            if uneven == 0:
                uneven = 1
            else:
                uneven = 0

#%%

"""

indexes         = [0,2,3, 7, 8, 9, 10, 11, 12, 17, 22, 23, 24, 25, 26, 27, 30,
35, 36, 37, 38, 39, 40, 41, 44, 49, 50, 51,52,  53, 56, 61, 62, 63, 64, 65, 72,
73, 74, 75, 76, 77, 85, 86, 87, 88, 89, 90, 98]
for index in sorted(indexes, reverse=True):
    del list_of_values[index]

#list_of_values = np.delete(list_of_values, deletion)

"""

#%%

background  = new_image = cv2.imread("data/{}.png".format(environment))


count = 0
for episode in list_of_values:
    if environment == "BlocksNormal":
        y_array         = (((np.array(episode[0]) / 37.5) * 500) + 215)
        x_array         = (((np.array(episode[1]) / 35) * 500) + 250)

    if environment == "BlocksObstacles":
        y_array         = (((np.array(episode[0]) / 31.5) * 500) + 218)
        x_array         = (((np.array(episode[1]) / 29) * 500) + 250)

    if environment == "Factory":
        x_array         = (((np.array(episode[0]) / 32) * 500) + 350)
        y_array         = (((np.array(episode[1]) / 34) * 500) + 400)

    y_array         = y_array.astype(int)
    y_array         = 500 - y_array

    x_array         = x_array.astype(int)
    x_array         = 500 - x_array

    test_image = cv2.imread("data/{}.png".format(environment))

    if episode[0] != []:
        #new_image[x_array[0]-3:x_array[0]+3, y_array[0]-3:y_array[0]+3] =
[0,255,0]
        #cv2.line(new_image, (y_array[-1]-3, x_array[-1]-3), (y_array[-1]+3,
x_array[-1]+3), (0,0,255), 1)
        #cv2.line(new_image, (y_array[-1]-3, x_array[-1]+3), (y_array[-1]+3,
x_array[-1]-3), (0,0,255), 1)
        new_image[x_array[-1]-3:x_array[-1]+3, y_array[-1]-3:y_array[-1]+3] =
[0,0,255]
        test_image[x_array[-1]-3:x_array[-1]+3, y_array[-1]-3:y_array[-1]+3] =
[0,0,255]

    for x in range(1, len(x_array)):
```

```
        color       = (0, 255, 0)
        thickness   = 1
        cv2.line(new_image, (y_array[x-1], x_array[x-1]), (y_array[x],
x_array[x]), color, thickness)
        cv2.line(test_image, (y_array[x-1], x_array[x-1]), (y_array[x],
x_array[x]), color, thickness)

    #cv2.imwrite("figures/testing/episode {}.png".format(count), test_image)
    count += 1


cv2.imwrite("figures/{} Adjusted Rewards Paths.png".format(environment),
new_image)


#%%

background  = new_image = cv2.imread("data/{}.png".format(environment))

print("Amount of outofview: ", len(x_values))

if environment == "BlocksNormal":
    y_array         = (((np.array(list_of_values[0][0]) / 37.5) * 500) + 215)
    x_array         = (((np.array(list_of_values[0][1]) / 35) * 500) + 250)

if environment == "BlocksObstacles":
    y_array         = ((((x_values) / 31.5) * 500) + 218)
    x_array         = ((((y_values) / 29) * 500) + 250)

if environment == "Factory":
    x_array         = ((((x_values) / 32) * 500) + 350)
    y_array         = ((((y_values) / 34) * 500) + 400)

y_array         = y_array.astype(int)
y_array         = 500 - y_array

x_array         = x_array.astype(int)
x_array         = 500 - x_array

for x in range(1, len(x_array)):
    cv2.line(new_image, (y_array[x]-6, x_array[x]-6), (y_array[x]+6,
x_array[x]+6), (0,255,0), 1)
    cv2.line(new_image, (y_array[x]-6, x_array[x]+6), (y_array[x]+6, x_array[x]-
6), (0,255,0), 1)

cv2.imwrite("figures/{}Newreward OutofView.png".format(environment), new_image)

#%%

import numpy as np
import matplotlib.pyplot as plt
import joypy
import pandas as pd
from typing import List

#%%

environment = "BlocksObstacles"
file_array = np.loadtxt("data\{}\BlocksObstacles Run 1 newreward - DQN stack-
depth(stopped) - RewardDistribution".format(environment))

#%%
```

```
environment = "BlocksObstacles"
file_array1 = np.loadtxt("data\{}\BlocksObstacles Run 5 - DQN stack-depth -
RewardDistribution".format(environment))
file_array2 = np.loadtxt("data\{}\BlocksObstacles Run 1 newreward - DQN stack-
depth(stopped) - RewardDistribution".format(environment))

#%%

counted         = file_array[:-1]
episodes        = file_array[-1]

frames          = np.arange(len(counted))

#%%

counted1         = file_array1[:-1]
episodes1        = file_array1[-1]

counted2         = file_array2[:-1]
episodes2        = file_array2[-1]


frames          = np.arange(len(counted1))

#%%

distribution    = counted/episodes

#%%

distribution1    = counted1/episodes1
distribution2    = counted2/episodes2

#%%

def smooth(scalars: List[float], weight: float) -> List[float]:  # Weight
between 0 and 1
    last = scalars[0]                                           # First value in
the plot (first timestep)
    smoothed = list()
    for point in scalars:
        smoothed_val = last * weight + (1 - weight) * point  # Calculate
smoothed value
        smoothed.append(smoothed_val)                           # Save it
        last = smoothed_val                                     # Anchor the last
smoothed value

    return smoothed

#%%

plt.figure(figsize=(14,5))
plt.title("DQN Stacked Depth - Adjusted Reward Distribution in
{}".format(environment))
plt.ylim([0, 100])
plt.bar(frames, counted, color='c')
plt.plot(frames, np.array(pd.DataFrame(counted).rolling(4,
min_periods=1).mean()), color='green')
plt.xlabel("Step in Episode")
plt.ylabel("Percentage of Reward Received (%)")
plt.xticks(frames)
plt.savefig("figures\Reward Distribution of {} NewardRun1".format(environment))
plt.show()
```

```
#%%

fig, axs = plt.subplots(2,2)

fig.set_size_inches(16, 10, forward=True)

custom_xlim = (0, 50)
custom_ylim = (0, 100)

# Setting the values for all axes.
plt.setp(axs, xlim=custom_xlim, ylim=custom_ylim)

axs[0].bar(frames, counted1, color='c')
axs[0].plot(frames, np.array(pd.DataFrame(counted1).rolling(4,
min_periods=1).mean()), color='green')
axs[0].set_title("Trained with Normal Reward", fontsize=12)

axs[1].bar(frames, counted2, color='c')
axs[1].plot(frames, np.array(pd.DataFrame(counted2).rolling(4,
min_periods=1).mean()), color='green')
axs[1].set_title("Trained with Adjusted Reward", fontsize=12)

axs[1].set_xlabel("Step in Episode", fontsize=12)

fig.text(0.08, 0.5, 'Percentage of Reward Received (%)', ha='center',
va='center', rotation='vertical')
plt.savefig("figures\Reward Distribution of {}".format(environment))

#%%

import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
from scipy.interpolate import make_interp_spline, BSpline
from typing import List

font = {'family' : 'normal',
        "weight" : "normal",
        'size'   : 8}

matplotlib.rc('font', **font)


#%%

environment = "BlocksNormal"

metric1_run1 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 1 - DQN nostack-
normal-tag-EvalMinReturn.csv".format(environment))
metric1_run2 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 3 - DQN stack-
normal-tag-EvalMinReturn.csv".format(environment))
metric1_run3 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 5 - DQN stack-
depth-tag-EvalMinReturn.csv".format(environment))

metric2_run1 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 1 - DQN nostack-
normal-tag-EvalMaxReturn.csv".format(environment))
metric2_run2 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 3 - DQN stack-
normal-tag-EvalMaxReturn.csv".format(environment))
metric2_run3 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 5 - DQN stack-
depth-tag-EvalMaxReturn.csv".format(environment))
```

```
metric3_run1 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 1 - DQN nostack-
normal-tag-EvalAverageReturn.csv".format(environment))
metric3_run2 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 3 - DQN stack-
normal-tag-EvalAverageReturn.csv".format(environment))
metric3_run3 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 5 - DQN stack-
depth-tag-EvalAverageReturn.csv".format(environment))

metric4_run1 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 1 - DQN nostack-
normal-tag-TrainAverageReturn.csv".format(environment))
metric4_run2 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 3 - DQN stack-
normal-tag-TrainAverageReturn.csv".format(environment))
metric4_run3 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 5 - DQN stack-
depth-tag-TrainAverageReturn.csv".format(environment))

metric5_run1 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 1 - DQN nostack-
normal-tag-TrainAVGEpisodeLength.csv".format(environment))
metric5_run2 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 3 - DQN stack-
normal-tag-TrainAVGEpisodeLength.csv".format(environment))
metric5_run3 = pd.read_csv("data/{}/run-logs_BlocksNormal Run 5 - DQN stack-
depth-tag-TrainAVGEpisodeLength.csv".format(environment))


experiment1 = pd.DataFrame(data = {"Run_1": metric1_run1["Value"], "Run_2":
metric1_run2["Value"], "Run_3": metric1_run3["Value"]})
experiment2 = pd.DataFrame(data = {"Run_1": metric2_run1["Value"], "Run_2":
metric2_run2["Value"], "Run_3": metric2_run3["Value"]})
experiment3 = pd.DataFrame(data = {"Run_1": metric3_run1["Value"], "Run_2":
metric3_run2["Value"], "Run_3": metric3_run3["Value"]})
experiment4 = pd.DataFrame(data = {"Run_1": metric4_run1["Value"], "Run_2":
metric4_run2["Value"], "Run_3": metric4_run3["Value"]})
experiment5 = pd.DataFrame(data = {"Run_1": metric5_run1["Value"], "Run_2":
metric5_run2["Value"], "Run_3": metric5_run3["Value"]})

#%%

def smooth(scalars: List[float], weight: float) -> List[float]:  # Weight
between 0 and 1
    last = scalars[0]                                             # First value in
the plot (first timestep)
    smoothed = list()
    for point in scalars:
        smoothed_val = last * weight + (1 - weight) * point  # Calculate
smoothed value
        smoothed.append(smoothed_val)                            # Save it
        last = smoothed_val                                      # Anchor the last
smoothed value

    return smoothed

experiment1['Run1'] = smooth(experiment1['Run_1'], 0.85)
experiment1['Run2'] = smooth(experiment1['Run_2'], 0.85)
experiment1['Run3'] = smooth(experiment1['Run_3'], 0.85)

experiment2['Run1'] = smooth(experiment2['Run_1'], 0.85)
experiment2['Run2'] = smooth(experiment2['Run_2'], 0.85)
experiment2['Run3'] = smooth(experiment2['Run_3'], 0.85)

experiment3['Run1'] = smooth(experiment3['Run_1'], 0.85)
experiment3['Run2'] = smooth(experiment3['Run_2'], 0.85)
experiment3['Run3'] = smooth(experiment3['Run_3'], 0.85)

experiment4['Run1'] = smooth(experiment4['Run_1'], 0.99)
experiment4['Run2'] = smooth(experiment4['Run_2'], 0.99)
```

```
experiment4['Run3'] = smooth(experiment4['Run_3'], 0.99)

experiment5['Run1'] = smooth(experiment5['Run_1'], 0.99)
experiment5['Run2'] = smooth(experiment5['Run_2'], 0.99)
experiment5['Run3'] = smooth(experiment5['Run_3'], 0.99)


#%%

#min_max_roll    = 10
#mean_roll       = 5



min_max_roll    = 35
mean_roll       = 20

experiment1['Run1_lower'] = experiment1['Run_1'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment1['Run2_lower'] = experiment1['Run_2'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment1['Run3_lower'] = experiment1['Run_3'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()

experiment1['Run1_upper'] = experiment1['Run_1'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment1['Run2_upper'] = experiment1['Run_2'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment1['Run3_upper'] = experiment1['Run_3'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()


experiment2['Run1_lower'] = experiment2['Run_1'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment2['Run2_lower'] = experiment2['Run_2'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment2['Run3_lower'] = experiment2['Run_3'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()

experiment2['Run1_upper'] = experiment2['Run_1'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment2['Run2_upper'] = experiment2['Run_2'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment2['Run3_upper'] = experiment2['Run_3'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()


experiment3['Run1_lower'] = experiment3['Run_1'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment3['Run2_lower'] = experiment3['Run_2'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment3['Run3_lower'] = experiment3['Run_3'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()

experiment3['Run1_upper'] = experiment3['Run_1'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment3['Run2_upper'] = experiment3['Run_2'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment3['Run3_upper'] = experiment3['Run_3'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()


experiment4['Run1_lower'] = experiment4['Run_1'].rolling(min_max_roll,
```

```
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment4['Run2_lower'] = experiment4['Run_2'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment4['Run3_lower'] = experiment4['Run_3'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()

experiment4['Run1_upper'] = experiment4['Run_1'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment4['Run2_upper'] = experiment4['Run_2'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment4['Run3_upper'] = experiment4['Run_3'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()


experiment5['Run1_lower'] = experiment5['Run_1'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment5['Run2_lower'] = experiment5['Run_2'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()
experiment5['Run3_lower'] = experiment5['Run_3'].rolling(min_max_roll,
min_periods=1).min().rolling(mean_roll, min_periods=1).mean()

experiment5['Run1_upper'] = experiment5['Run_1'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment5['Run2_upper'] = experiment5['Run_2'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()
experiment5['Run3_upper'] = experiment5['Run_3'].rolling(min_max_roll,
min_periods=1).max().rolling(mean_roll, min_periods=1).mean()




fig, axs = plt.subplots(3,2, figsize=(20,15))
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.08,
hspace=0.2)
custom_ylim = (-1, 50)
custom_xlim = (0, 2000)

x = experiment1['Run1'].index * 50 + 50

axs[0,0].plot(x, experiment1['Run1'], label='DQN - Single Normal',
color='tab:green')
axs[0,0].plot(x, experiment1['Run1_lower'], color='tab:green', alpha=0.1)
axs[0,0].plot(x, experiment1['Run1_upper'], color='tab:green', alpha=0.1)
axs[0,0].fill_between(x, experiment1['Run1_lower'], experiment1['Run1_upper'],
alpha=0.05)
axs[0,0].plot(x, experiment1['Run2'], label='DQN - Stacked Normal',
color='tab:red')
axs[0,0].plot(x, experiment1['Run2_lower'], color='tab:red', alpha=0.1)
axs[0,0].plot(x, experiment1['Run2_upper'], color='tab:red', alpha=0.1)
axs[0,0].fill_between(x, experiment1['Run2_lower'], experiment1['Run2_upper'],
alpha=0.05)
axs[0,0].plot(x, experiment1['Run3'], label='DQN - Single Depth',
color='tab:orange')
axs[0,0].plot(x, experiment1['Run3_lower'], color='tab:orange', alpha=0.1)
axs[0,0].plot(x, experiment1['Run3_upper'], color='tab:orange', alpha=0.1)
axs[0,0].fill_between(x, experiment1['Run3_lower'], experiment1['Run3_upper'],
alpha=0.05)

axs[1,0].plot(x, experiment2['Run1'], label='DQN - Single Normal',
color='tab:green')
axs[1,0].plot(x, experiment2['Run1_lower'], color='tab:green', alpha=0.1)
axs[1,0].plot(x, experiment2['Run1_upper'], color='tab:green', alpha=0.1)
```

```
axs[1,0].fill_between(x, experiment2['Run1_lower'], experiment2['Run1_upper'],
alpha=0.05)
axs[1,0].plot(x, experiment2['Run2'], label='DQN - Stacked Normal',
color='tab:red')
axs[1,0].plot(x, experiment2['Run2_lower'], color='tab:red', alpha=0.1)
axs[1,0].plot(x, experiment2['Run2_upper'], color='tab:red', alpha=0.1)
axs[1,0].fill_between(x, experiment2['Run2_lower'], experiment2['Run2_upper'],
alpha=0.05)
axs[1,0].plot(x, experiment2['Run3'], label='DQN - Single Depth',
color='tab:orange')
axs[1,0].plot(x, experiment2['Run3_lower'], color='tab:orange', alpha=0.1)
axs[1,0].plot(x, experiment2['Run3_upper'], color='tab:orange', alpha=0.1)
axs[1,0].fill_between(x, experiment2['Run3_lower'], experiment2['Run3_upper'],
alpha=0.05)

axs[0,1].plot(x, experiment3['Run1'], label='DQN - Single Normal',
color='tab:green')
axs[0,1].plot(x, experiment3['Run1_lower'], color='tab:green', alpha=0.1)
axs[0,1].plot(x, experiment3['Run1_upper'], color='tab:green', alpha=0.1)
axs[0,1].fill_between(x, experiment3['Run1_lower'], experiment3['Run1_upper'],
alpha=0.05)
axs[0,1].plot(x, experiment3['Run2'], label='DQN - Stacked Normal',
color='tab:red')
axs[0,1].plot(x, experiment3['Run2_lower'], color='tab:red', alpha=0.1)
axs[0,1].plot(x, experiment3['Run2_upper'], color='tab:red', alpha=0.1)
axs[0,1].fill_between(x, experiment3['Run2_lower'], experiment3['Run2_upper'],
alpha=0.05)
axs[0,1].plot(x, experiment3['Run3'], label='DQN - Single Depth',
color='tab:orange')
axs[0,1].plot(x, experiment3['Run3_lower'], color='tab:orange', alpha=0.1)
axs[0,1].plot(x, experiment3['Run3_upper'], color='tab:orange', alpha=0.1)
axs[0,1].fill_between(x, experiment3['Run3_lower'], experiment3['Run3_upper'],
alpha=0.05)


x = experiment4['Run1'].index * 2

axs[1,1].plot(x, experiment4['Run1'], label='DQN - Single Normal',
color='tab:green')
axs[1,1].plot(x, experiment4['Run1_lower'], color='tab:green', alpha=0.1)
axs[1,1].plot(x, experiment4['Run1_upper'], color='tab:green', alpha=0.1)
axs[1,1].fill_between(x, experiment4['Run1_lower'], experiment4['Run1_upper'],
alpha=0.05)
axs[1,1].plot(x, experiment4['Run2'], label='DQN - Stacked Normal',
color='tab:red')
axs[1,1].plot(x, experiment4['Run2_lower'], color='tab:red', alpha=0.1)
axs[1,1].plot(x, experiment4['Run2_upper'], color='tab:red', alpha=0.1)
axs[1,1].fill_between(x, experiment4['Run2_lower'], experiment4['Run2_upper'],
alpha=0.05)
axs[1,1].plot(x, experiment4['Run3'], label='DQN - Single Depth',
color='tab:orange')
axs[1,1].plot(x, experiment4['Run3_lower'], color='tab:orange', alpha=0.1)
axs[1,1].plot(x, experiment4['Run3_upper'], color='tab:orange', alpha=0.1)
axs[1,1].fill_between(x, experiment4['Run3_lower'], experiment4['Run3_upper'],
alpha=0.05)


axs[2,0].plot(x, experiment5['Run1'], label='DQN - Single Normal',
color='tab:green')
axs[2,0].plot(x, experiment5['Run1_lower'], color='tab:green', alpha=0.1)
axs[2,0].plot(x, experiment5['Run1_upper'], color='tab:green', alpha=0.1)
axs[2,0].fill_between(x, experiment5['Run1_lower'], experiment5['Run1_upper'],
alpha=0.05)
```

```
axs[2,0].plot(x, experiment5['Run2'], label='DQN - Stacked Normal',
color='tab:red')
axs[2,0].plot(x, experiment5['Run2_lower'], color='tab:red', alpha=0.1)
axs[2,0].plot(x, experiment5['Run2_upper'], color='tab:red', alpha=0.1)
axs[2,0].fill_between(x, experiment5['Run2_lower'], experiment5['Run2_upper'],
alpha=0.05)
axs[2,0].plot(x, experiment5['Run3'], label='DQN - Single Depth',
color='tab:orange')
axs[2,0].plot(x, experiment5['Run3_lower'], color='tab:orange', alpha=0.1)
axs[2,0].plot(x, experiment5['Run3_upper'], color='tab:orange', alpha=0.1)
axs[2,0].fill_between(x, experiment5['Run3_lower'], experiment5['Run3_upper'],
alpha=0.05)

size = 15
axs[2,0].set_xlabel('Epochs', fontsize=size)
axs[0,0].set_ylabel('Reward', fontsize=size)
axs[1,0].set_ylabel('Reward', fontsize=size)
axs[2,0].set_ylabel('Steps', fontsize=size)

axs[2,0].set_xlabel('Epochs', fontsize=size)
axs[1,1].set_xlabel('Epochs', fontsize=size)


axs[0,0].set_title("Evaluation: Minimum Reward",  fontsize=size)
axs[1,0].set_title("Evaluation: Maximum Reward", fontsize=size)
axs[0,1].set_title("Evaluation: Average Reward", fontsize=size)
axs[1,1].set_title("Training: Average Reward", fontsize=size)
axs[2,0].set_title("Training: Average Episode Length", fontsize=size)


plt.setp(axs, xlim=custom_xlim, ylim=custom_ylim)


for x in range(len(axs)):
    for y in range(len(axs[0])):
        axs[x,y].legend(loc='upper left',fontsize=size)
        axs[x,y].tick_params(axis='y', labelsize= size)
        axs[x,y].tick_params(axis='x', labelsize= size)

axs[2,0].legend(loc='lower left',fontsize=size)
axs[1,0].legend(loc='lower left',fontsize=size)


plt.savefig("figures\Training for {}".format(environment))
plt.show()
```