Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра
(бакалавра, спеціаліста, магістра)

Студента Оленченко Георгій Михайлович
(ПІБ)
академічної групи 126-20-1
(шифр)
спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)
за освітньо-професійною програмою «Інформаційні системи та технології»
(офіційна назва)
на тему Розробка симулятора керування дроном в умовах радіоелектронної боротьби
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
Кваліфікаційної роботи:	доц. Соколова Н.О.			
Розділів:				

Рецензент	доц. Клименко А.В.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро

2024

ЗАТВЕРДЖЕНО:
завідувач кафедри

інформаційних технологій та
комп'ютерної інженерії
(повна назва)

(підпис) (прізвище, ініціали)
«_____» _____ 2024 року

ЗАВДАННЯ

на кваліфікаційну роботу
ступеня бакалавра
(бакалавра, спеціаліста, магістра)

студенту Оленченку Г. М. **академічної групи** 126-20-1
(прізвище та ініціали) (шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

за освітньо-професійною програмою «Інформаційні системи та технології»
(офіційна назва)

на тему Розробка симулятора керування дроном в умовах радіоелектронної боротьби
(назва за наказом ректора)

затвердженну наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 № 469-с

Розділ	**Зміст**	**Термін виконання**
Розділ 1	Аналіз стану області рішення задачі	
Розділ 2	Проєктні рішення	
Розділ 3	Розробка проєктного рішення	

Завдання видано _____ _____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ _____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 83 с., 38 рис., 1 табл., 3 додатки, 36 джерел.

Об’єкт розробки: Керування дроном в умовах радіоелектронної боротьби.

Мета роботи: розробити симулятор керування безпілотним апаратом (дроном) в умовах радіоелектронної боротьби завдяки аналізу об’єктів на зображенні.

У вступі наведено інформацію про актуальність проблеми використання дронів в сучасній війні, їх можливості в умовах радіоелектронної боротьби та поточний стан застосування технології машинного зору в безпілотних апаратах.

В першому розділі розглянуто поняття дрон, наведено та проаналізовано характеристики відомих повітряних, надводних та підводних дронів, розроблених в Україні. Розглядається історія розвитку і застосування машинного зору та штучного інтелекту в різних галузях.

У другому розділі наведена проєктна складова вирішення завдання. Розглянуто методи аналізу об’єктів на зображенні, а саме виділення контурів, сегментація та робота зі згортковою нейронною мережею (CNN). Обрання моделі Segment Anything Model (SAM) для виконання сегментації зображенні.

Практичне значення кваліфікаційної роботи полягає у розробці прототипу програми, яка виконує автоматичний запрограмований рух дрона на основі аналізу зображення на наявність об'єктів, в умовах відсутності радіосигналу.

Ключові слова: ДРОН, ВІЙСЬКОВА ТЕХНІКА, БЕЗПІЛОТНІ АПАРАТИ, РАДІОЕЛЕКТРОННА БОРОТЬБА, МАШИННИЙ ЗІР, ШТУЧНИЙ ІНТЕЛЕКТ, НЕЙРОННІ МЕРЕЖІ, СЕГМЕНТАЦІЯ ЗОБРАЖЕННЯ, SEGMENT ANYTHING MODELS.

ABSTRACT

Explanatory note: 83 pages, 38 pictures, 1 table, 3 appendices, 36 sources.

Development object: Simulator of controlling a drone in the conditions of electronic warfare.

The purpose of the qualification work: to develop a simulator of controlling an unmanned aircraft (drone) in the conditions of radio-electronic warfare through the analysis of objects in the image.

The introduction provides information on the relevance of the problem of the use of drones in modern warfare, their capabilities in the conditions of electronic warfare and the current state of application of machine vision technology in unmanned aerial vehicles.

In the first section, the concept of a drone is considered, the characteristics of well-known aerial, surface and underwater drones developed in Ukraine are given and analyzed. The history of the development and application of machine vision and artificial intelligence in various fields is considered.

In the second section, the project component of solving the task is given. The methods of object analysis in the image are considered, namely contour selection, segmentation and work with a convolutional neural network (CNN). Choosing the Segment Anything Model (SAM) to perform image segmentation.

The practical value of the qualification work consists in the development of a prototype of the program, which performs automatic programmed movement of the drone based on the analysis of the image for the presence of objects, in the absence of a radio signal.

Keywords: DRONE, MILITARY EQUIPMENT, UNMANNED APPARATUS, RADIO ELECTRONIC WARFARE, MACHINE VISION, ARTIFICIAL INTELLIGENCE, NEURAL NETWORKS, IMAGE SEGMENTATION, SEGMENT ANYTHING MODELS.

ЗМІСТ

ВСТУП

Актуальність роботи. В умовах сучасної війни технології безпілотного бою відіграють критичну роль у протистоянні російському вторгненню. Використання повітряних, надводних та підводних дронів дає значний успіх на полі бою. Серед повітряних безпілотників є популярними FPV дрони - відносно низька вартість зброї, швидкісні та маневрені. Вони наздоганяють та влучають майже в будь-яку ціль на полі бою. Один такий дрон — це мінімум 50 врятованих життів військових у найгарячіших точках фронту та знищена дороговартісна техніка. Надводні та підводні дрони виконують свої задачі на морських горизонтах.

Однак основним недоліком даних апаратів є глушіння радіосигналу засобами радіоелектронної боротьби (РЕБ), що позбавляє можливості виконати поставлене завдання. Наявність машинного зору дозволить дрону долітати до встановленої цілі, незважаючи на те, як налаштована система РЕБ. Автоматичне наведення на техніку, що рухається, надає майже 100% попадання, ніж при ручному керуванні, а також розширюється дальність польоту апарату у випадку втрати зв’язку з гарнітурою керування природнім шляхом (тобто вихід за максимальний радіус дії радіосигналу).

Машинний зір інтегрований у всі баражуючі боєприпаси, які відповідного класу є зараз у світі, коли мова йде про промислові зразки баражуючих (баражування — режим польоту літального апарата, що забезпечує якнайдовшу тривалість польоту для оперативного реагування на появу загрози [2]) боєприпасів. Польський WARMATE є одним з таких зразків. Прикладом цивільного використання технології машинного зору є Mavic від компанії DJI, де ще у першій версії була функція ActiveTrack. [3]

Однак такі елементи як машинний зір проблемно інтегрувати в FPV-дрони через значне зростання вартості літального апарату, через що використовуються допоміжні модулі.

Мета роботи – розробити програму аналізу зображення на наявність об'єктів, що симулює втрату радіосигналу з дроном та на основі отриманих даних з зображення програмно виконує керування дроном.

Предмет роботи – розробка симулятора керування безпілотним апаратом (дроном) в умовах радіоелектронної боротьби завдяки аналізу об’єктів на зображенні.

Об'єкт досліджень – безпілотні апарати та технології обробки і аналізу зображень для автоматичного управління.

Для вирішення мети роботи необхідно виконати наступні задачі:

1. *Провести аналіз стану області рішення задачі*: проаналізувати відомі типи безпілотних апаратів, розглянути історію розвитку та застосування машинного зору та штучного інтелект, оцінити існуючі рішення для автоматичного управління дронами в умовах радіоелектронної боротьби.
2. *Ознайомитися з проєктними рішеннями*: розглянути методи виявлення та аналізу об'єктів на зображенні, обрати відповідні моделі та алгоритми для сегментації зображень та автоматичного управління дроном, обґрунтувати вибір конкретних технологій і методів, які будуть використовуватися в розробці.
3. *Розробити проєктне рішення*: розробити прототип програми, що буде виконувати автоматичне програмоване переміщення дрона на основі отриманих даних після аналізу та сегментації зображення (відеопотоку), провести тестування розробленого коду.

РОЗДІЛ 1. АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1.1. Поняття дрон. Аналіз відомих типів безпілотних апаратів

Дрон – це безпілотний частіше всього літальний апарат (БПЛА), військового чи цивільного призначення, є різновидом військового робота. Зазвичай є мобільним автономним апаратом, запрограмованим на виконання певних завдань. Існує багато різних типів безпілотних літальних апаратів (рис. 1.1). Зазвичай їх можна поділити на дві основні категорії: перші використовуються для розвідки і спостереження, а другі оснащені ракетами та бомбами (ударні або дрони-камікадзе) [4].

Рисунок 1.1 – «Народний» Bayraktar TB2

В 1948 році, США розробили один з перших розвідувальних безпілотних літальних апаратів — AQM-34. Перший політ відбувся у 1951 році, і тоді ж цей безпілотний літальний апарат був запущений у серійне виробництво, ставши першим серійним БПЛА [4]. БПЛА Taranis UCAV, розроблений британською компанією BAE Systems, з двигуном Adour Mk951, є найсучаснішим британським апаратом (розробка триває з 2006 року) [5]. США, Великобританія та Ізраїль є єдиними країнами, які використовували озброєні безпілотні літаки в бойових умовах [4].

Основними напрямами застосування дронів є наступні [4]:

- контроль над пересуванням людей та техніки, ідентифікація тварин.
- *логістика*: перевезення вантажів та постачання (у тому числі медичне) до важкодоступних місцевостей.
- *сільське господарство*: обробка культур пестицидами, висівання
- *розвідка*: оптична, лазерна, телевізійна, радіаційна (частково використовується в першому напрямі)

Сьогодні дрони активно використовуються у війні безпілотників. Це форма повітряної війни з використанням бойових безпілотних літальних апаратів (unmanned combat aerial vehicle, UCAV) або озброєних комерційних безпілотних літальних апаратів (unmanned aerial vehicle, UAV). Станом на початок 2022 року бойові БПЛА виробляли США, Велика Британія, Ізраїль, Китай, Південна Корея, Туреччина та декілька інших європейських країн [6].

Серед безпілотних апаратів можна виділити декілька категорій [7]:

- Літальні апарати (БПЛА).
- Морські (надводні) дрони.
- Підводні дрони

1.1.1.1. Літальні апарати. FPV-дрони.

FPV-дрони (First Person View, «вид від першої особи») – це пристрої, які дозволяють користувачеві зануритися в атмосферу польоту, ніби він сам перебуває на борту дрона. Цей режим керування відомий як FPV-пілотування. Завдяки спеціальним камерам, які транслюють відео в реальному часі на екран пілота або до спеціальних окулярів, користувач може отримати неперевершений досвід польоту. Початок історії FPV-дронів починається з 1980-х років, коли ці пристрої вперше почали використовуватися у військових цілях (рис. 1.2). Проте вже в 2000-х роках, як зазвичай буває з більшістю

військових розробок, дрони з FPV-технологією стали доступними для кожної людини. Відтоді вони швидко здобули популярність серед ентузіастів і професіоналів в різних галузях: від зйомки кіно до надзвичайних ситуацій. Також популярні серед спортсменів та фотографів.

Рисунок 1.2 – FPV-дрон, модифікований під військові потреби

Даний вид безпілотників є нескладним в збірці, але дуже складним в керуванні, що потребує високої майстерності пілота. FPV-дрони-камікадзе або дрони зі скидом, діють у парі з розвідувальними. Це забезпечує більшу ефективність застосування та ситуаційну обізнаність для операторів. Найкращі моделі коштують удвічі менше, ніж звичайний квадрокоптер Mavic. Одним з таких українських камікадзе є дрон KH-S7 [8].

1.1.1.2. Технічні характеристики FPV-дрона KH-S7.

Ядром, головною частиною будь-якого FPV-дрона є його контролер польоту. В ньому відбувається обробка вхідних сигналів з пульта та контроль моторів. Окрім цього, FPV-дрон містить в собі камеру, передавач відео, приймач, мотори, пропелери та акумулятор. Для пілотування також необхідні спеціальні окуляри чи монітор та пульт дистанційного керування [9].

На FPV-дронах використовується аналогова система передачі відео, яка хоч і менш вразлива до засобів РЕБ, ніж цифрова, але все ж таки можлива до заглушення. У разі втрати оператором контакту з безпілотником, апарат не зможе вразити ціль. У випадку FPV-дронів з машинним зором, обидва ці фактори не є проблемою - після того як дрон зафіксував ціль і став самостійним, глушіння каналів передачі зображення чи керування вже ні на що не вплине.

У випадку дрону KH-S7 (рис. 1.3.) тактико-технічні характеристики (ТТХ) є наступними: корисна вага – 1 кг на 7 км, дальність польоту – 9.5 км. Також ведеться процес реєстрації нової моделі – KH-S10: перенесення до 3 кг ваги. Інші деталі не розкриваються.

Рисунок 1.3 – FPV-дрон KH-S7

1.1.2.1. Морські дрони. Дрон-камікадзе MAGURA V5.

Безпілотний надводний апарат (USV, Unmanned Surface Vessel), також зветься *морським дроном*, має можливість рухатися на поверхні води повністю без втручання екіпажу на борту. Прикладом USV є плавуча в океані платформа «Автономний безпілотний космопорт», яка використовується компанією SpaceX для посадки першого ступеня ракет сімейства Falcon. У військових безпілотних дронів виконуються такі функції, як протичовнова війна, протимінна боротьба, патрулювання і захист акваторій. Вперше радіокероване мініатюрне судно було розроблено і продемонстровано у 1898 р. Ніколою

Теслом (хоча перші заявки на патенти на подібні рішення подавалися винахідниками Великобританії ще в 1897 році) [10].

Робота над розробкою ударних морських безпілотників в Україні розпочалася після успішного застосування малих автономних суден берегової оборони (розвідка, розмінування прибережної акваторії та патрулювання), отриманих від США. Серед таких розробок - відомий ударний морський дрон MAGURA V5 (рис. 1.4), вперше представлений у 2023 році на виставці International Defence Industry Fair (IDEF) у Стамбулі.

Рисунок 1.4 – Надводний дрон Magura V5 на поштовому блоці «Зброя Перемоги. Made in UA»

Цей безпілотник є невеликим, швидкісним та високоманевреним дистанційно пілотованим судном, що несе на борту бойову частину, яка детонує при ударі з ворожим кораблем. Не дивлячись на складні умови застосування, такі дрони мають доволі просту конструкцію, що дозволяє їх виготовляти на малих непрофільних підприємствах в умовах війни [11]. Це унікальна українська розробка, яка може виконувати широкий набір операцій – спостереження, розвідка, патрулювання, пошуково-рятувальні роботи, протимінну боротьбу, охорону морського флоту та бойові місії.

1.1.2.2. Технічні характеристики морського дрону MAGURA

Керування морським дроном здійснюється за допомогою систем супутникової геолокації, а також через відеоканал, де сигнал з камери передається через систему супутникового зв’язку Starlink на пульт керування у

командному пункті [11]. ТТХ першого покоління таких дронів наступні: довжина 5.5м, операційний радіус дії до 400 км, дальність ходу до 800 км, автономність до 60 годин, повна вага до 1000 кг, максимальна швидкість до 80 км/год. Передача інформації шифрується криптозахистом (шифрування 256 біт) [12]. Нове покоління дронів отримало більш гідродинамічний корпус, змінена система супутникового зв'язку через заборону компанії Starlink на застосування їх технологій у військових цілях. Комплекс включає в себе не лише сам дрон з системою автопілотування, відеопідсистемами (включаючи нічне бачення), резервними модулями зв'язку та бойовою частиною, але й наземну автономну станцію управління, систему транспортування і зберігання, центр обробки даних.

1.1.3.1. Керовані підводні безпілотники. Безпілотник Toloka

Безпілотні підводні апарати (UUV, Unmanned underwater vehicle), також звуться *підводними дронами*, працюють тільки під водою як за допомогою керування людиною-оператором, так і без (рис. 1.5). Ці апарати поділяються на дві категорії: дистанційно керовані підводні апарати (ROUV, Remotely operated underwater vehicle) і автономні підводні апарати (AUV, Autonomous underwater vehicle) [13].

Перші прототипи дистанційно-керованих підводних апаратів були розроблені на запит військово-морських сил США у 1950-х роках і використовувалися для пошуку навчальних торпед. В Україні роботи з проєктування і створення телекерованих підводних апаратів ведуться з середини 1980-х років Науково-дослідним інститутом підводної техніки Національного університету кораблебудування імені адмірала Макарова. В цей період створено низку наукових шкіл у напрямку проєктування глибоководної техніки, спроєктовані і побудовані перші зразки вітчизняних дистанційно-керованих підводних апаратів [14].

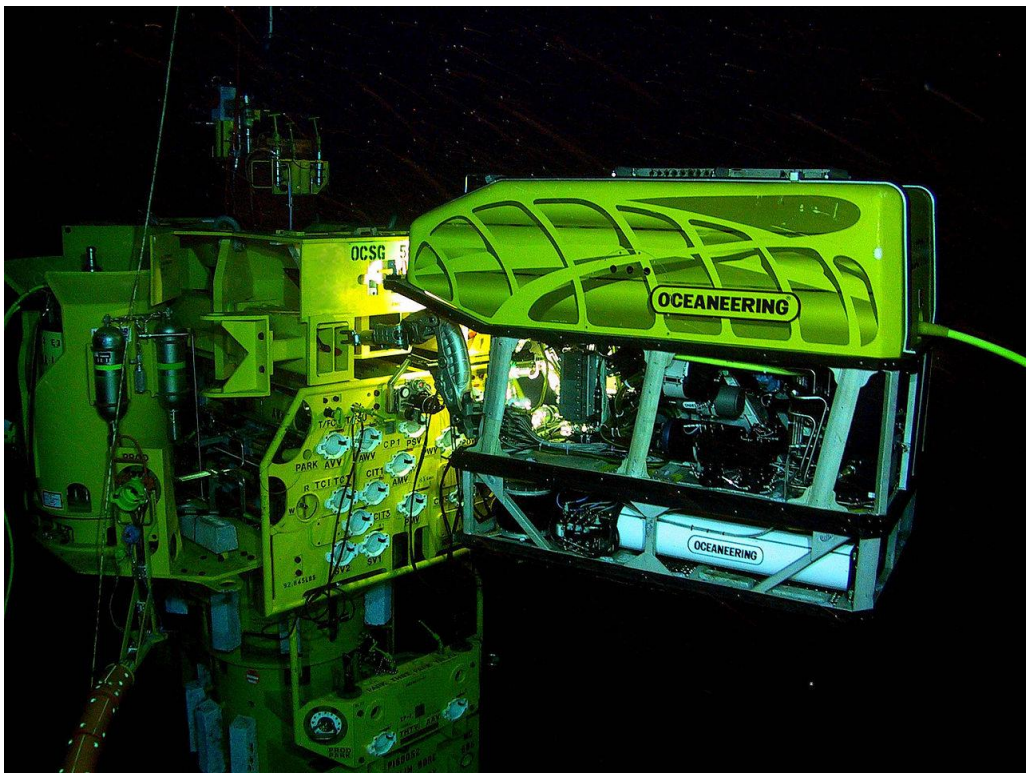


Рисунок 1.5 - Дистанційно керований підводний апарат під час роботи на підводному родовищі нафти та газу

У випадку коли ROUV дистанційно керує людина-оператор, частіше всього через складні підводні умови зникає можливість ефективної роботи дрона. AUV працюють незалежно від прямого втручання людини, мають вагу від кількох до тисяч кілограмів. До недавніх часів, автоматизовані підводні апарати (АПА) використовувались лише в невеликій кількості галузей застосування. Однак з розвитком технологій обробки даних і вдосконалення джерел живлення, АПА стали розвиватися (особливо у військовій сфері) і частіше застосовуватися. Військові апарати використовуються для патрулювання акваторій, протимінної оборони (включаючи виявлення та знищення мін), а також підводної розвідки. Одним з таких АПА є TALISMAN (рис. 1.6) від британського підприємства BAE Systems (спеціалізується в оборонній та аерокосмічній галузях).

Рисунок 1.6 – АПА TALISMAN від BAE Systems

24 березня 2023 року стало відомо, що Північна Корея провела випробування свого нового підводного безпілотника, здатного нести ядерну зброю. Зазначається, що місія підводної ядерної стратегічної зброї полягає в потаємному зануренні в оперативну зону і створенні надпотужної радіоактивної хвилі за допомогою підводного вибуху для знищення груп ворожих кораблів і ключових портів операції. Нову підводну наступальну систему зброї назвали «ядерним безпілотним ударним судном Цунамі» [15].

В Україні активний розвиток підводних дронів розпочався майже з самого початку повномасштабного вторгнення росії в Україну. На конференції оборонного кластеру Brave1 була представлена одна з кількох моделей керованих підводних дронів – Toloka TLK 150. Це невеликий дрон-торпеда, оснащений щоглою з антеною та поворотною камерою для наведення на ціль і підтримки зв’язку. У разі глушіння зв’язку засобами РЕБ, апарат виявляє джерело та виходить із зони їх впливу. Він вміє автоматично сканувати простір для пошуку цілей або за допомогою 3D-сонару та гідрофону, або в ручному режимі за допомогою камери. Проєкт знаходиться в початковій стадії розробки, передбачено розробку нейромережі, яка б ідентифікувала цілі по відеоканалу та автоматично наводила дрон на них та розширення кількості версій дронів-торпед до трьох: TLK 150 (2,5 метри); TLK 400 (4-6 метрів); TLK

1000 (4-12 метри). Дрон поєднує в собі обидві категорії керування ROUV та AUV [11].

1.1.3.2. Технічні характеристики підводного безпілотника Toloka TLK 150

Бойова частина: 20-50 кг вибухівки, запас ходу: 100 км, система навігації: сигнал GPS та інерціальна система навігації в умовах відсутності GPS (під водою) або під час глушіння сигналу, в розробці нейромережа для автоматичної ідентифікації цілей (рис. 1.7).

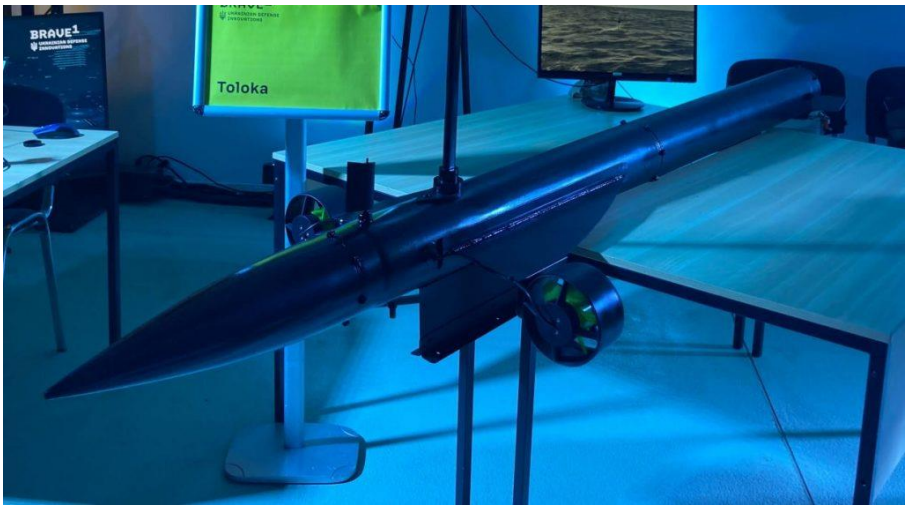


Рисунок 1.7 – Безпілотний дрон-торпеда Toloka

1.2. Машинний зір: від комп'ютерного бачення до штучного інтелекту.

Часто, в різних джерелах інформації, можна побачити два терміни – машинний і комп’ютерний зір. Це схожі поняття, але між ними існує певна відмінність.

Комп’ютерний зір — технологія, що дає змогу машинам знаходити й розпізнавати об’єкти, відстежувати, здійснювати відеоаналітику, описувати зміст зображень, розпізнавати жести та обробляти отримані результати. Вона відноситься до галузі штучного інтелекту. Відбувається зосередження на обробці тривимірних сцен, спроєктованих на одне чи декілька зображень.

Найбільшою областю застосування комп’ютерного зору є військова. Це виявлення противника і транспортних засобів, а також керування ракетами та безпілотними апаратами. В таких випадках визначення цілей відбувається тоді, коли ракета чи дрон досягають заданої області, базуючись на відеоданих, що надходять.

З іншого боку, *машинний зір* — це застосування технології комп’ютерного зору у певних сферах діяльності. Тобто комп’ютерний зір містить у собі загальний набір технологій, а машинний зір використовує аналіз зображень, щоб вирішувати поставленні завдання. Окрім цього, сьогодні машинний зір вже є невід'ємною частиною штучного інтелекту. Дана технологія дозволяє комп’ютерам вдосконалюватися на власному досвіді завдяки глибокому навчанню.

Перші згадки про комп’ютерний зір (computer vision) або машинний зір (machine vision) з’являються у 1960-х рр.. На сьогодні в англомовних джерелах перше поняття є суттєво частіше згадуваним (рис. 1.8).

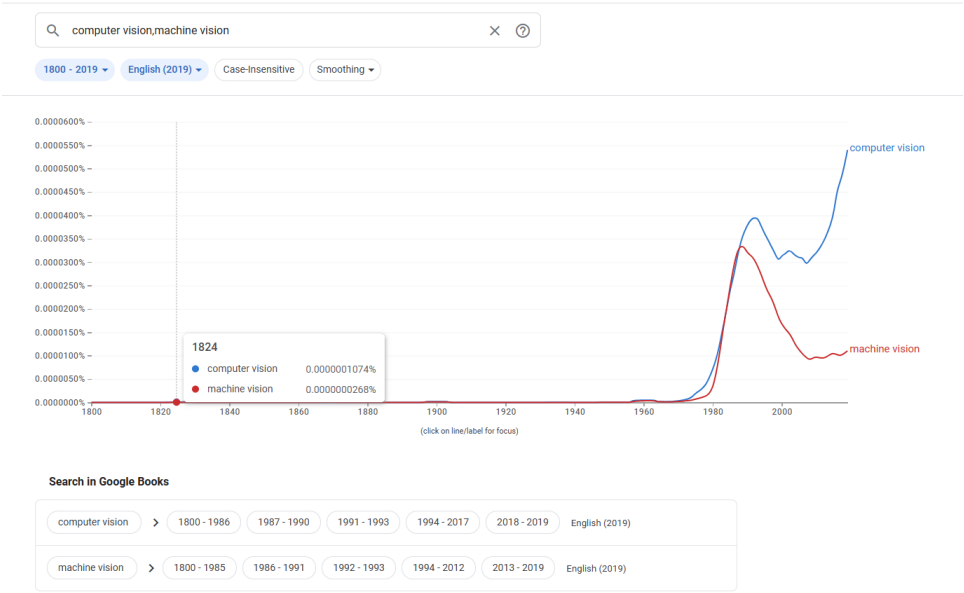


Рисунок 1.8 – Згадування понять computer vision та machine vision в друкованих виданнях [20] (дані отримані з ресурсу Google Books Ngram Viewer)

В області штучного інтелекту важливою частиною є автоматичне планування та ухвалення рішень у системах, здатних виконувати механічні дії, такі як пересування пристрою крізь певне середовище. Цей тип обробки зазвичай потребує вхідних даних, отриманих системами комп'ютерного зору, що функціонують як відеосенсори і надають високорівневу інформацію про середовище та роботу [17].

Для точної обробки зображень чи відео в програмах комп’ютерного зору життєво важливу роль відіграє точне виділення ознак та отримання отримання якісних даних (без таких даних похибки в аналізі зображень стають завеликими). Цей процес передбачає визначення домінуючих характеристик на зображеннях, які можна використовувати для аналізу (наприклад країв або текстур об'єктів). Зазвичай для виконання комп’ютерної візуалізації використовуються бібліотеки, зокрема мови програмування Python, такі як OpenCV, Pillow, Tensorflow та інструменти CUDA, MATLAB, Keras і SimpleCV.

1.2.1. Історія розвитку машинного зору

Поняття машинного зору бере свій початок з 50-х років XX століття у галузі комп’ютерного зору, яка остаточно сформувалася в 1960-х роках, коли дослідники шукали способи дати комп’ютерам змогу інтерпретувати візуальну інформацію так само, як це робить людина. Перші зусилля були зосереджені на базових завданнях, таких як аналіз та обробка зображень, розпізнавання образів і виявлення об’єктів.

Значний вплив на розвиток машинного зору вплинув прогрес у комп’ютерних технологіях, зокрема збільшення обчислювальної потужності, обсягу пам’яті та складності алгоритмів. Коли комп’ютери стали більш потужними і здатними обробляти великі обсяги візуальних даних, дослідники почали вивчати більш складні застосування комп’ютерного зору [21].

1.2.1.1. Ранній період

Концепція машин, що імітують людський зір, бере свій початок на початку 1930-х років з винайденням електронних сортувальних машин, що використовували фотоелектронні детектори для ідентифікації об’єктів за кольором і розміром. Галузь почала активно розвиватися у 1960-х роках завдяки першим дослідженням у Массачусетському технологічному інституті та іншим науково-дослідним установам. Дослідники вивчали фундаментальні концепції обробки зображень і розпізнавання образів, закладаючи основу для майбутніх розробок. Саме в цей рік було побудовано пристрій, що імітував схему розпізнавання образів людським мозком - апаратний варіант Mark I Perceptron (рис. 1.9), призначений для розпізнавання зорових образів, моделюючи роботу людського мозку [17].

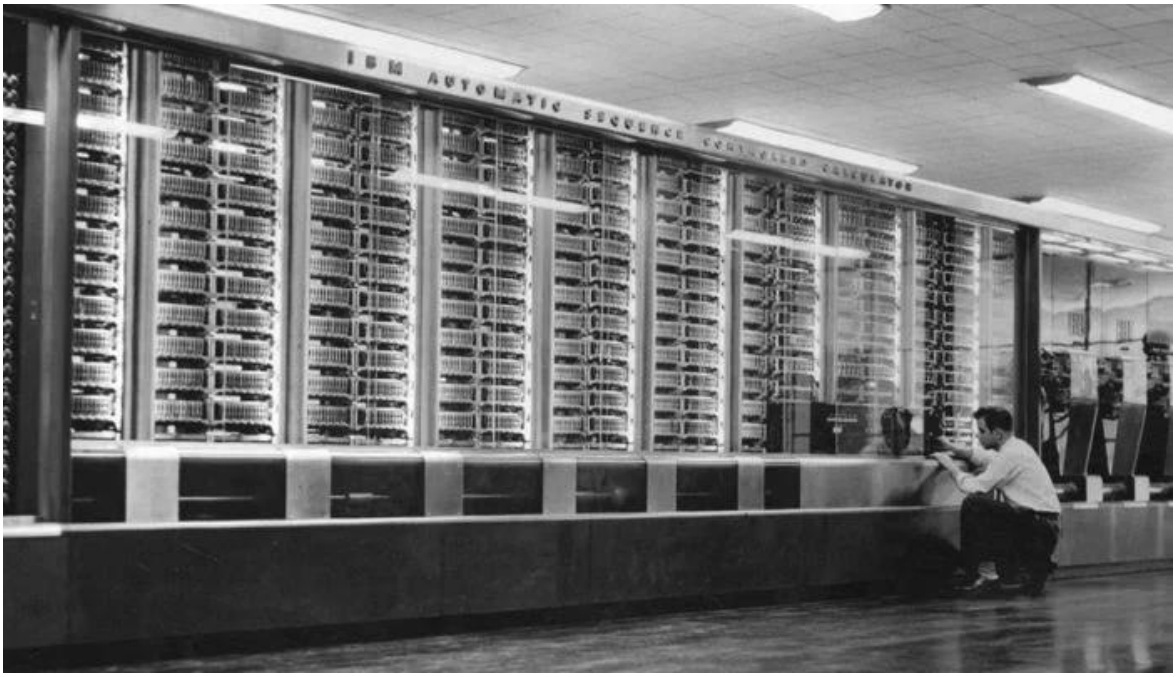


Рисунок 1.9 - Mark I Perceptron

На початку 1960-х років завдання комп'ютерного зору в основному охоплювали область космічних досліджень, що вимагали обробки великої кількості цифрової інформації.

1.2.1.2. Розвиток алгоритмів

1970-ті роки стали часом значного прогресу в алгоритмах обробки зображень, таких як виявлення країв і сегментація. Це дозволило пристроям виокремлювати основні ознаки на зображеннях. Лауренс Робертс, аспірант Массачусетського технологічного інституту (MTI), висунув концепцію машинної побудови тривимірних образів об'єктів на основі аналізу їх двовимірних зображень. На цьому етапі став проводитися глибший аналіз даних. Почали розвиватися різні підходи до розпізнавання об'єктів на зображенні, наприклад структурні, ознакові і текстурні. Пізніше, 1972 року, створено перший курс «Машинний зір» у Лабораторії штучного інтелекту MTI, що готував дослідників і додавав академічний інтерес до цієї галузі. У 1980-х роках з’явилися перші комерційно доступні системи машинного зору, які використовувалися в таких галузях, як виробництво для автоматизованого контролю якості на основі системи оптичного розпізнавання символів OCR (optical character recognition), що могла сприймати текст із зображень. Пізніше з’явилися перші комерційні системи автоматичної навігації автомобілів [17].

1.2.1.3. Епоха глибокого навчання

Глибоке навчання (Deep Learning) — це «розділ» комп’ютерного зору високого рівня, що використовує нейронні мережі для розуміння складних візуальних моделей. Передбачається навчання великих нейронних мереж на величезних обсягах розмічених даних для розпізнавання об’єктів або виконання конкретних завдань з високим рівнем точності. Глибоке навчання призвело до прогресу в таких сферах, як виявлення об’єктів і сегментація [23].

2000-ні роки відкрили еру глибокого навчання — потужної технології з багатошаровими штучними нейронними мережами. Це викликало революцію у машинному зорі, дозволивши комп’ютерам вивчати складні особливості та закономірності напряму з даних. Алгоритми, що засновані на глибокому навчанні, призвели до значних проривів у розпізнаванні об’єктів, класифікації

зображень та інших завданнях. Ця епоха також стала свідком розробки складних програмних інструментів і доступного обладнання, що зробило машинний зір більш доступним і широко застосовуваним. 2003 рік - на ринок були випущені перші досить надійні корпоративні системи розпізнавання осіб [22].

До справжнього моменту теорія комп'ютерного зору повністю склалася як самостійний розділ кібернетики, що спирається на солідну наукову і практичну базу знань [24].

1.3. Використання та розвиток безпілотних літальних апаратів в Україні: військовий аспект та інновації

За останні декілька років дрони набули великої популярності в Україні у військовій сфері. На початку російського вторгнення в Україну турецький безпілотник Bayraktar TB2 зіграв важливу роль в уповільненні та відбитті вторгнення: він використовувався для визначення цілей, коригування вогню артилерії, забезпечення підтримки з повітря, знищення колон постачання, а також російських систем ППО «Панцир-С1», «Тор», «Бук», «Стріла-10», артилерії та РСЗВ. В ході атаки ЗСУ на острів Зміїний, за допомогою Bayraktar було знищено 4 патрульні катери «Раптор», 1 десантний катер «Серна», 3 системи ППО, командний пункт та склад боєприпасів на острові. Знищено транспортний вертоліт Мі-8 під час розвантаження російського десанту. Пізніше ефективність використання даного безпілотника сильно впала через малу швидкість і невелику висоту польоту та розгортання систем РЕБ, які глушили сигнал керування. У 2025 році Baykar планує відкрити в Україні завод з виробництва Bayraktar TB2 та Bayraktar Akinci.

Проєкт Армія дронів, спільний проєкт Генштабу ЗСУ, Міністерства оборони, Міністерства цифрової трансформації і Держслужби спецзв'язку та захисту інформації, що був створений за ініціативою Президента

фандрейзингової платформи United24, активно сприяють розвитку українського ринку БПЛА. Тут контрактуються українські виробники дронів на постачання різних типів безпілотників для потреб ЗСУ, здійснюється їх технічне обслуговування — ремонт і постачання комплектуючих, проводяться систематичні навчання пілотів дронів. Окрім того, було створено абсолютно нові підрозділи в історії ЗСУ — роти ударних безпілотних авіаційних комплексів (РУБпАК). Генштаб ЗСУ відмітив, що це перші у світі роти ударних дронів [25]. Також регулярно проводяться івенти для виробників дронів та дотичних технологій, інструкторів-практиків та волонтерських організацій.

У 2023 році було запущено кластер Brave1 віцепрем'єр-міністром з інновацій, розвитку освіти, науки та технологій, міністром цифрової трансформації Михайлом Федоровим, тодішнім міністром оборони України Олексієм Резніковим, міністром з питань стратегічних галузей промисловості Олександром Камишіним. Це платформа для співпраці Defense Tech компаній («компаній оборонних технологій»), держави та військових, а також інвесторів, волонтерських фондів, медіа і всіх, хто допомагає наближати перемогу через технології. Вже було представлено розробки, які отримали гранти на системи ситуаційної обізнаності, використання штучного інтелекту, супутникових даних, таких як Sirko-S1 (наземна платформа для перевезення вантажу), Ironclad (багатофункціональний наземний роботизований комплекс, призначений для вирішення бойових завдань), ЛЮТЬ (дистанційно керований бойовий робот, озброєний танковим кулеметом ПКТ), HIMERA (компактна цифрова радіостанція солдата для спілкування в межах підрозділу), Піранья – 5 РАдК (антидронний мобільний п'яти діапазонний переносний комплекс по купольному та спрямованому захисту призначений для знешкодження ворожих БПЛА та Дронів) та ін [26-28].

На даний момент прогрес відбувається в розвитку зв'язку, додається захоплення цілі двома шляхами: нейромережа, яку навчають, і машинний зір. Фахівці компанії AirUnit (молода компанія, яка є постачальником техніки

таких брендів, як DJI, Aaronia AG, EcoFlow та інших, заснована українцями з великим експертним досвідом у сфері безпілотних літальних апаратів) працюють над системою самонаведення для FPV-дронів, за умови наявності якої оператору БПЛА потрібно буде лише захопити ціль виділивши об’єкт на екрані, далі в справу вступить програмне забезпечення та машинний зір [29].

Восени 2023 року Міноборони України допустило до експлуатації у ЗСУ дрон SAKER SCOUT зі штучним інтелектом. Закрите програмне забезпечення SAKER, побудоване на алгоритмах штучного інтелекту, допоможе військовим ефективніше бити ворога. Система за допомогою просунутої оптики самостійно розпізнає і фіксує координати техніки ворога (навіть замаскованої) негайно передаючи інформацію в командний пункт для прийняття відповідного рішення. В комплекс входять флагманський дрон-розвідник, а також кілька дронів-камікадзе типу FPV, які корегуються за допомогою флагманського дрона. SAKER SCOUT (рис. 1.10) може оснащуватися інфрачервоною оптикою та інерціальною системою наведення, що значно підвищує його стійкість до засобів РЕБ. Дальність польоту – до 10 км [30].

Рисунок 1.10 – Знімок, зроблений з дрона SAKER SCOUT з відміченими цілями

1.4. Висновки по розділу

Війна безпілотників поступово переходить на новий рівень: до бойових систем інтегрують елементи штучного інтелекту, у тому числі машинний зір. Це докорінно змінює характер війни, надаючи переваги тому, хто може виробляти та використовувати на полі бою апарати, здатні автоматично захоплювати та супроводжувати цілі у реальному часі.

Не дивлячись на те, що технології машинного зору вже давно активно розвиваються, впровадження даного функціоналу в безпілотні пристрої досі відбувається повільно, тому розробка систем керування дроном в умовах радіоборотьби є актуальною.

Основною задачею даної кваліфікаційної роботи є розробка симулятора програмного керування безпілотним апаратом (дроном) в умовах радіоелектронної боротьби завдяки аналізу об’єктів на зображенні, елементи якого в подальшому можуть бути імплементовані в реальні пристрої.

РОЗДІЛ 2. ПРОЄКТНІ РІШЕННЯ

2.1. Огляд методів аналізу об’єктів на зображенні

Розпізнавання об’єктів на зображенні може виконуватися використовуючи традиційні методи обробки зображень, або сучасних мереж глибокого навчання.

Перший випадок, як правило, не потребує попередньо відібраних даних для навчання, анотованих зображень (ручне маркування об’єктів) та є неконтрольованим, через що такі методи обмежені декількома факторами, такими як складні сценарії (не одноколірний фон, частково сховані об’єкти, освітлення, тіні тощо).

Методи глибокого навчання зазвичай залежать від контрольованого чи неконтрольованого навчання, при цьому контрольовані методи є стандартом в задачах комп’ютерного зору. Визначення об’єктів за допомогою глибокого навчання значно більш стійке до оклюзії (ситуація, в якій два об'єкти розташовані приблизно на одній лінії і один об'єкт, розташований ближче до віртуальної камери або вікна перегляду, частково або повністю закриває видимість іншого об'єкта [31]), складним сценам та освітленню. Продуктивність обмежена обчислювальною потужністю графічних процесорів, яка стрімко зростає з кожним роком.

2.1.1. Виділення контурів об’єктів на зображенні

На сьогоднішній день популярними алгоритмами виділення контурів є оператори Робертса, Прюітта, Собеля та Кенні.

Оператор Собеля - дискретний диференціальний оператор, що обчислює наближене значення градієнта яскравості зображення - оператор використовує значення інтенсивності тільки в околиці 3×3 кожного пікселя для отримання наближення відповідного градієнта зображення, і використовує

тільки цілі значення вагових коефіцієнтів яскравості для оцінки градієнта. Так знаходиться напрямок найбільшого збільшення яскравості та величина її зміни в цьому напрямку. Результат показує, наскільки «різко» чи «плавно» змінюється яскравість зображення у кожній точці, отже, ймовірність знаходження точки межі, і навіть орієнтацію контуру. Математичний результат оператора Собеля в точці, що лежить в області постійної яскравості, буде нульовий вектор, а в точці, що лежить на межі областей різної яскравості, вектор, що перетинає кордон у напрямку збільшення яскравості.

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.1}$$

$$G_y = \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{vmatrix} * A \tag{2.2}$$

$$G_x = \begin{vmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{vmatrix} * A \tag{2.3}$$

Згідно з формулою оператора Собеля (2.1), Gx та Gy – дві матриці, де кожна точка містить наближені похідні по X та У, розраховуються шляхом множення матриць одна на одну та сумування обох матриць, після чого отриманий результат записується в поточні координати X та У та нове зображення.

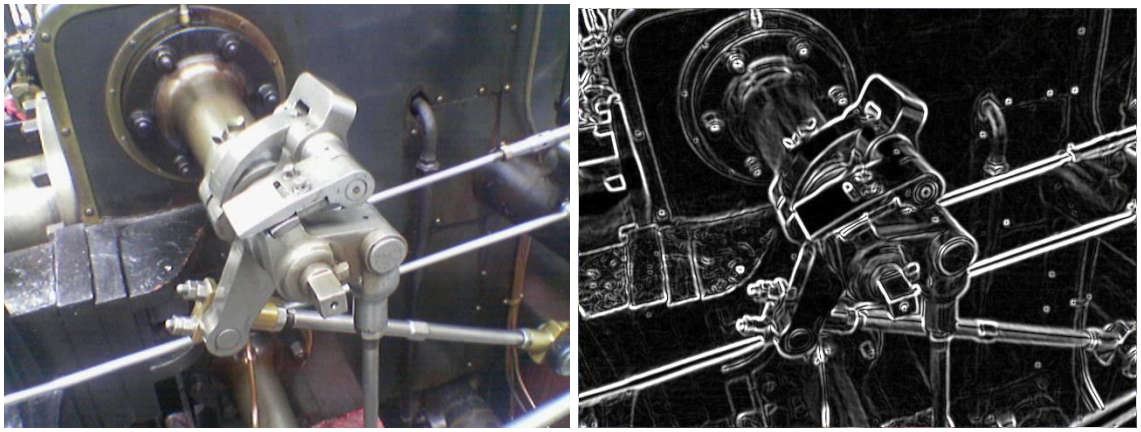


Рисунок 2.1 – Виділення контурів на зображенні оператором Собеля (зліва – оригінальне зображення, праворуч – виділені контури)

Оператор Прюїтт обчислює максимальний час відгуку на безлічі ядер згортки для знаходження локальної границі в кожному пікселі. Він був створений доктором Джудіт Прюїтт (Judith Prewitt) для виявлення меж медичних зображень. Схожий на оператор Собеля, але використовуються іншого вигляду матриці. Аналогічно оператору Собеля використовує значення інтенсивності тільки в околиці 3x3 кожного пікселя.

$$G_x = \begin{vmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{vmatrix} * A \tag{2.4}$$

$$G_y = \begin{vmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{vmatrix} * A \tag{2.5}$$

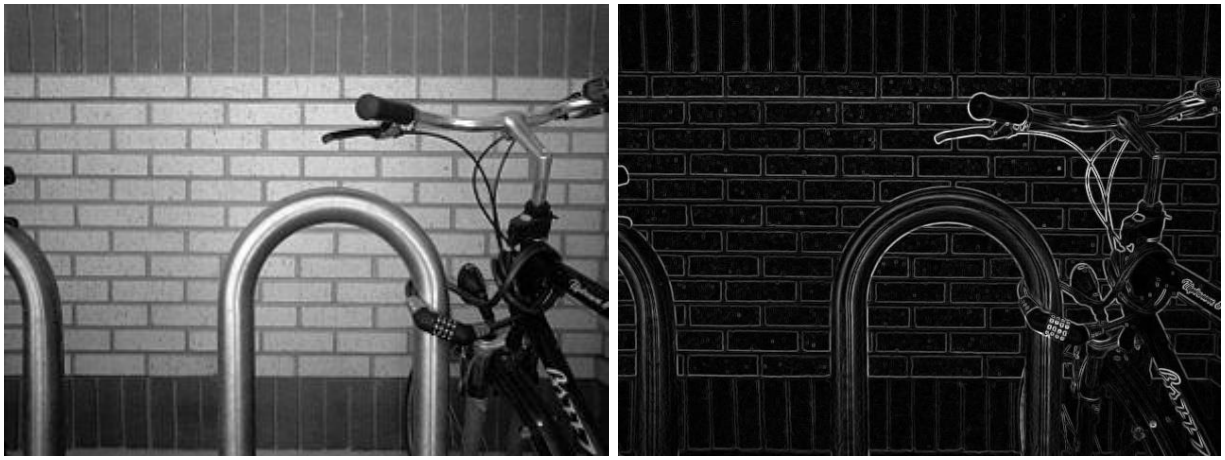


Рисунок 2.2 – Виділення контурів на зображенні оператором Прюїтт (зліва – оригінальне зображення, праворуч – виділені контури)

Оператор Робертса - один із ранніх алгоритмів виділення контурів, який обчислює на плоскому дискретному зображенні суму квадратів різниць між діагонально суміжними пікселями. Це може бути виконано згорткою зображення з двома ядрами (2.6). Іншими словами, величина перепаду G отриманого зображення розраховується з вихідних значень параметра У дискретних точках растру з координатами Х та У (2.7-2.9).

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \tag{2.6}$$

$$G_1 = Y_{x,y} - Y_{x+1,y+1} \tag{2.7}$$

$$G_2 = Y_{x+1,y} - Y_{x,y+1} \tag{2.8}$$

$$G = \sqrt{G_1^2 + G_2^2} \tag{2.9}$$

Оператор Робертса все ще використовується для швидкості обчислень, але він програє в порівнянні з альтернативами через значну чутливість до шуму, що часто є неприйнятним. Він дає лінії тонше, ніж інші методи виділення контурів, що майже рівнозначно обчисленню кінцевих різниць уздовж координат X та Y. Іноді цей оператор називають «фільтром Робертса» [32].

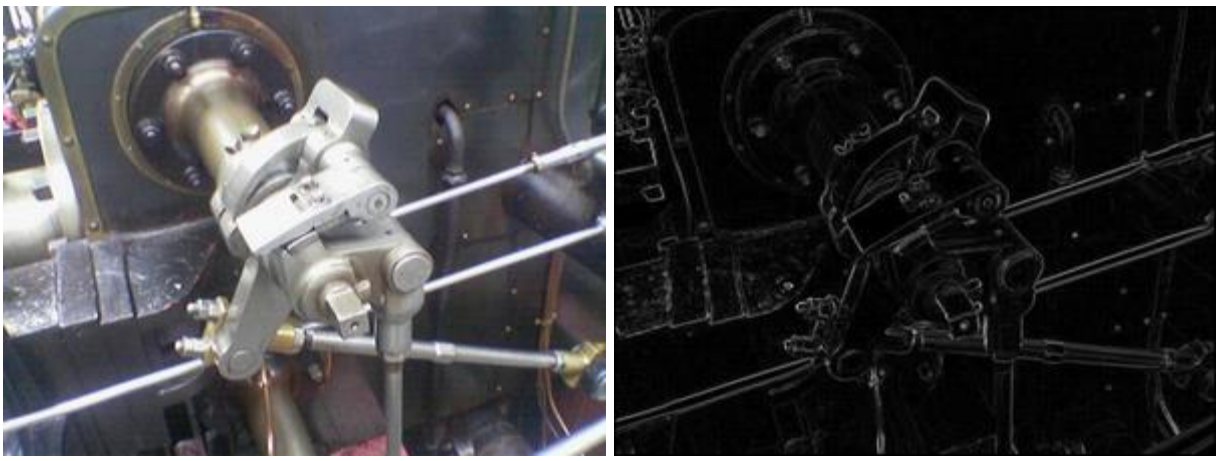


Рисунок 2.3 – Виділення контурів на зображенні оператором Робертса (зліва – оригінальне зображення, праворуч – виділені контури)

Алгоритм Кенні – це оператор виявлення контурів, що використовує багатоетапний алгоритм для роботи з контурами на зображеннях. Розроблений Джоном Кенні у 1986 році на основі розробок обчислювальної теорії виявлення контурів, яка пояснює, чому його методика діє. Один із найбільш строго визначених методів, що забезпечує з високою точністю надійне

виявлення контурів та є одним із найпопулярніших алгоритмів. Дана методика виділяє різні об'єкти бачення й корисну структурну інформацію та використовує низький обсяг даних для обробки. Широко застосовується в різних системах комп'ютерного бачення. Так як вимоги до застосування виявлення контурів у різних системах бачення відносно подібні, тому загальними критеріями до виявлення є наступні: виявлення контуру з низьким рівнем похибки, що означає, що виявляння має точно вловлювати якомога більше контурів, що показані на зображенні; точка контуру, виявлена оператором, повинна точно розміщуватися в центрі контуру; заданий контур на зображенні повинен бути позначений одноразово, шум зображення не повинен створювати хибних контурів де це можливо. Для задоволення цих вимог, Кенні використав варіаційне числення (методика, яка знаходить функцію, що оптимізує заданий функціонал). Оптимальну функцію виявлення контурів Кенні описують суму чотирьох показникових членів, але її можливо наближувати першою похідною гауссіана [33].

Алгоритм виявляння контурів Кенні можливо розбити на п'ять різних етапів:

1) *Застосування гауссового фільтра для згладження зображення задля усунення шуму.* Рівняння для ядра гауссового фільтра розміром (2k+1)*(2k+1) задають як (2.10), де *A означає операцію згортання. Приклад гауссового фільтра 5x5 з σ=1 для створення зображення (рис. 2.4) наведено в (2.11). Чим більшим є розмір ядра, тим менша чутливість виявляча до шуму і відповідно дещо більша похибка у виявлянні контурів.

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left(i-(k+1)\right)^2+\left(j-(k+1)\right)^2}{2\sigma^2}\right); 1 \le i,j \le (2k+1) \qquad (2.10)$$

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A \qquad (2.11)$$

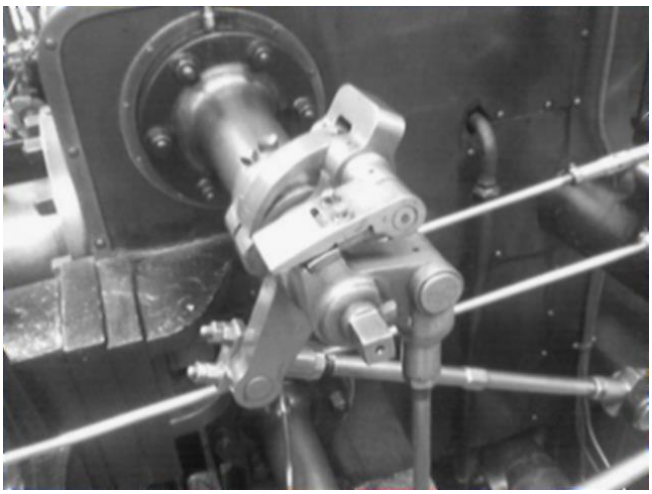


Рисунок 2.4 – Оброблене зображення після гауссівського фільтра

2) *Знаходження градієнтів яскравості зображення.* Алгоритм Кенні використовує 4 фільтри для виявлення на розмитому зображенні горизонтальних, вертикальних та діагональних контурів. Оператор виявлення контурів Робертса, Прюїтта або Собеля повертає значення для першої похідної в горизонтальному G_x та вертикальному G_y напрямках (2.1). З цього можна визначити градієнт та напрямок контуру (2.12), де atan2 – функція арктангенса з двома аргументами. Кут напрямку контуру округлюється до 0°, 45°, 90° та 135°. У випадку G_x=0° або G_y=180°, θ=0°.

$$\theta = atan2(G_y, G_y)$$

3) *Порогування величини градієнта для позбуття паразитної реакції на виявляння контурів*. Виконується для витончування контурів. У певних втіленнях алгоритм категоризує безперервні напрямки градієнта у невеликий набір дискретних напрямків, а потім переміщує фільтр 3x3 над результатом попереднього кроку. У кожному пікселі він пригнічує вираженість контуру центрального пікселя (встановлюючи його значення в 0), якщо його величина не перевищує величину двох сусідів у напрямку градієнта.

4) *Подвійне порогування.* Фільтруються контурні пікселі зі слабким значенням градієнта та зберігаються з високим (обранням верхнього та нижнього

порогових значень). Якщо значення градієнта контурного пікселя менше ніж нижнє порогове, його буде пригнічено. Якщо значення градієнта вище – позначається як сильний контурний піксель. Обидва значення визначаються емпірично, залежать від вмісту вхідного зображення.

5) *Простежування контурів за допомогою гістерезису*. Останній крок. Для досягнення точного результату усуваються слабкі контури, викликані попередніми кроками. Зазвичай слабкий контурний піксель буде з'єднано з сильним, тоді як шумові не з'єднані. Для простеження з'єднань використовується плямний аналіз шляхом розгляду слабкого контурного пікселя та пікселів його 8-зв'язного околу. Доки в цій плямі є сильні контурні пікселі, цю слабку контурну точку можливо визначити як таку, яку слід зберегти.

Покроковий результат виявлення контурів зображено на рис. 2.5.

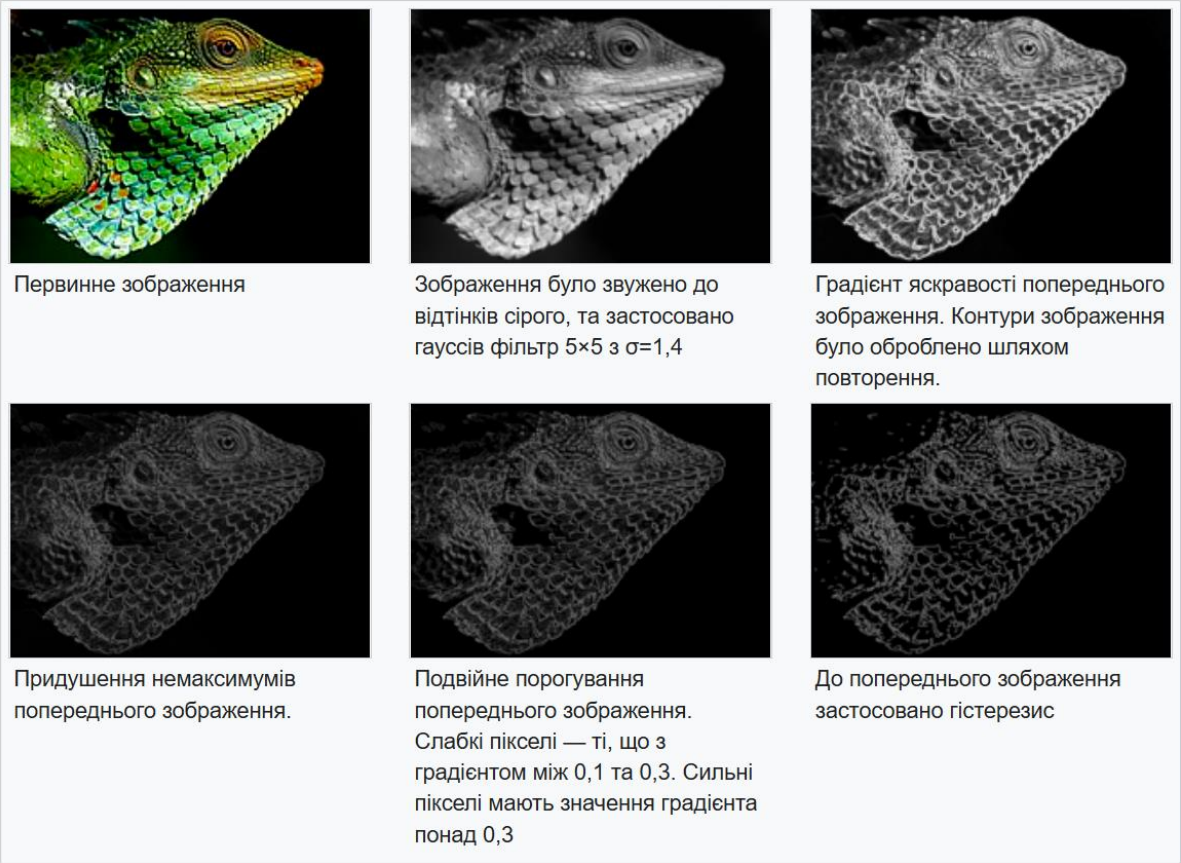


Рисунок 2.5 – Застосування алгоритму Кенні для виявлення контурів на зображенні

Усі зазначені методи ґрунтуються на одній із базових властивостей сигналу яскравості – розривності. Найбільш загальним способом пошуку розривів є обробка зображення за допомогою ковзної маски, званої також фільтром, ядром, вікном або шаблоном, яка є якоюсь квадратною матрицею, що відповідає зазначеній групі пікселів вихідного зображення. Елементи матриці прийнято називати коефіцієнтами. Оперування такою матрицею у будь-яких локальних перетвореннях називається фільтрацією або просторовою фільтрацією.

2.1.2. Сегментація зображення

Сегментація зображення – це розбиття зображення на велику кількість областей, які покривають знайдені об’єкти (набір пікселів з однаковими мітками) на зображенні, для яких виконується певний критерій однорідності – наприклад приблизно однакова яскравість. Сегментація застосовується в багатьох сферах діяльності людини (медична, військова, промислова, картографічна та ін.). Є однією з основних задач обробки та аналізу зображень.

Серед різновидів сегментації виділяють семантичну та екземплярну. *Семантична сегментація* – це процес розбиття зображення на декілька сегментів та присвоєння кожному з них мітки класу. Кожному пікселю надається свій клас. Такий підхід використовується при розпізнаванні об’єктів, аналізі медичних зображень, класифікації територій на супутникових знімках. *Екземплярна сегментація* – більш складний процес: окрім мітки класу, кожний об’єкт на зображенні отримує унікальний ідентифікатор. Наприклад якщо на фотографії зображено декілька мотоциклів, окрім надання класу «мотоцикл», кожен об’єкт буде мати свій ідентифікатор. Це робиться для навчання моделі для вміння відрізняти об’єкти однакового класу один від одного. Саме цей підхід застосовують в віртуальній реальності та сфері безпілотного керування.

На даний момент популярними є наступні алгоритми сегментації: алгоритм на основі випадкового лісу (Random forest), алгоритм на основі згорткової нейронної мережі (convolutional neural network, CNN), алгоритм на основі метода К-середніх (K-means), порогова сегментація, методи перетворення пікселів в ознаки, сегментація на основі графів (рис. 2.6).

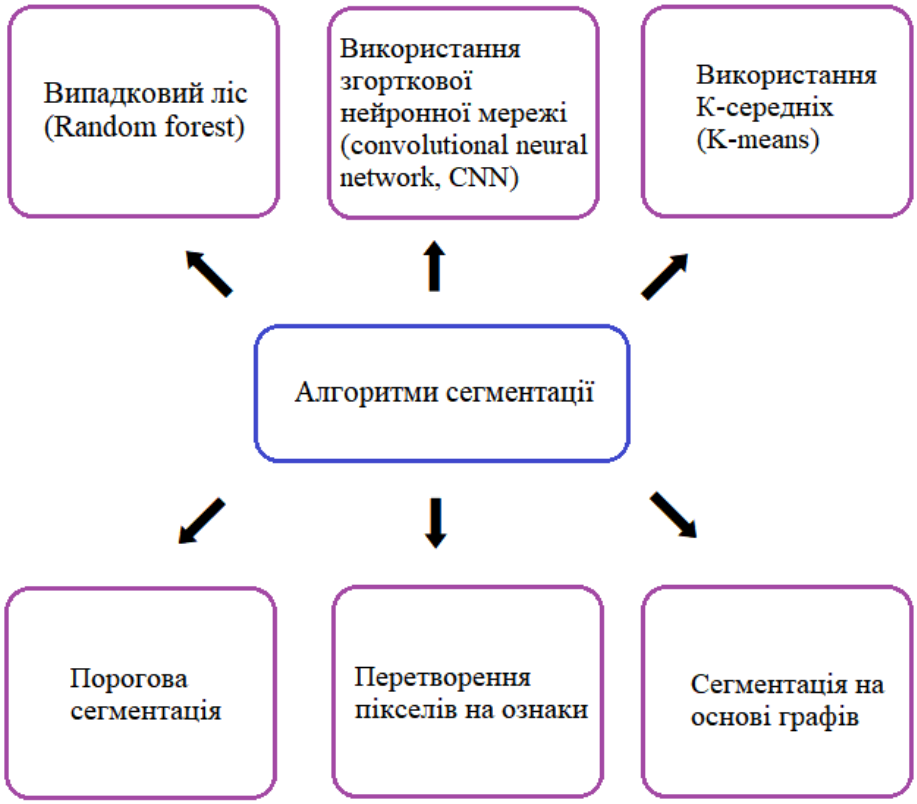


Рисунок 2.6 – Методи сегментації зображень

Алгоритм випадкового лісу використовує комплексний набір дерев на основі вибірок даних з кількома змінними для прийняття рішень про класифікацію пікселів на зображеннях. Кожен піксель буде проходити через дерево рішень, яке визначить його належність до певного класу. Після цього будуть використані результати усіх дерев для визначення найбільш ймовірного класу кожного пікселя.

Алгоритм на основі згорткової нейронної мережі (CNN) використовує згортковий шар для виявлення особливостей зображень, а потім застосовує нейромережу для прийняття рішень класифікації пікселів на зображеннях. Такий алгоритм може бути як повністю автоматичним, так і з використанням ручного корегування під час перевірки отриманих результатів навчання для покращення точності. Один із прикладів використання CNN для сегментації зображення – мережа U-Net, яка була розроблена для медичної сегментації зображень. Вона була навчена на наборі даних, що містить медичні зображення та відповідні їм маски, де кожен піксель маски зазначає, до якого класу він належить.

Алгоритм з урахуванням методу K-середніх (K-Means) поділяє зображення на сегменти на основі їх характеристик, найчастіше колірних. Для цього спочатку визначається кількість сегментів K, далі ініціалізуються центри кластерів випадково, кожен піксель відноситься до кластера з найближчим до нього центру. Далі алгоритм обчислює нові центри кластерів і повторює процедуру доти, доки досягнуто максимальне число ітерацій чи доки центри кластерів не перестануть змінюватися. У такий спосіб можна створювати ефектні фотофільтри або вирішувати інші завдання.

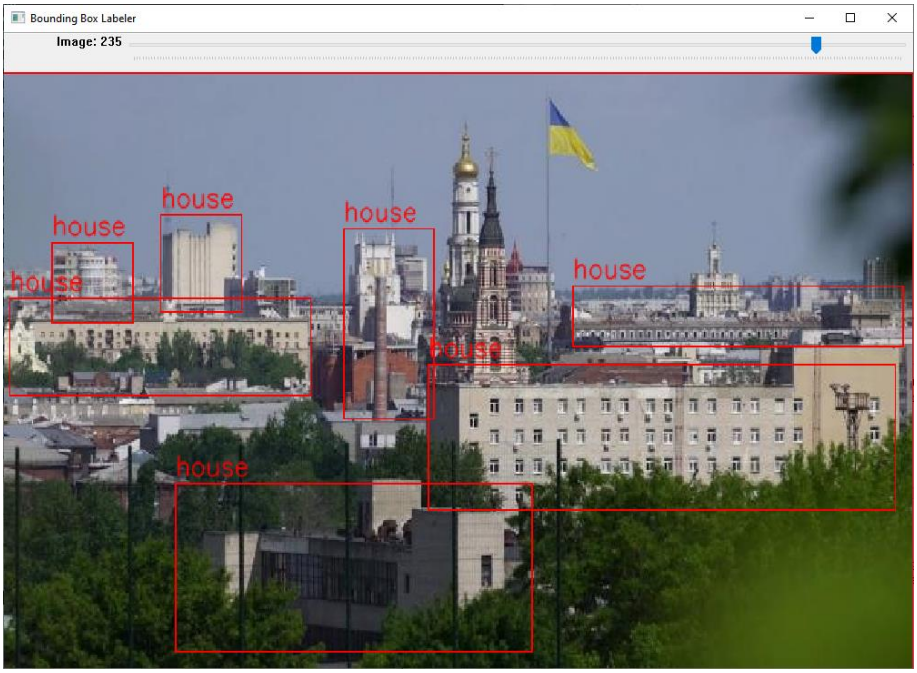
Порогова сегментація відмінно працює для бінарних зображень (зображення, що мають тільки два можливих значення кожного пікселя, зазвичай це білий та чорний). Поріг – це властивість, яка допомагає розділити шуканий сигнал на класи. Метод ґрунтується на встановленні певного порогу ймовірності для пікселів та подальшому присвоєнні кожному пікселю мітки класу в залежності від його яскравості. Наприклад, за допомогою алгоритму Оцу для сегментації зображень на два класи із заданим порогом яскравості пікселів. Таким чином можна відокремити об'єкти від фону на чорно-білій фотографії, зокрема при обробці медичних зображень. Є одним з основних та простих способів сегментації.

Методи перетворення пікселів на ознаки. Відбувається переведення пікселів зображення в числові ознаки, які можна використовувати в аналізі та класифікації. Серед таких методів наявні метод головних компонентів (PCA), метод гістограм орієнтованих градієнтів (HOG) та метод градієнтної карти (SIFT).

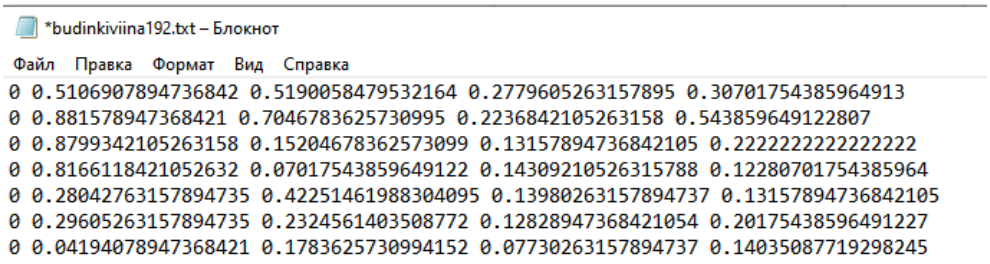
Сегментація з урахуванням графів. Такі алгоритми ґрунтуються на графових моделях, таких як марківські випадкові поля (МВП) та графові згорткові мережі (GCNN). Наприклад, модель МВП можна використовувати для сегментації медичних зображень: щоб розділити області здорових тканин та пухлинних утворень. Для цього за допомогою стохастичного алгоритму модель визначає межі між сегментами, розбиваючи їх на два класи – здоровий чи пухлинний. Метод МВП враховує безліч факторів, включаючи зв'язки між пікселями, текстури, кольори та форми, що дозволяє отримувати високоякісні результати сегментації.

В наш час для сегментації зображень частіше використовуються алгоритми сегментації зображень з використанням нейронних мереж. Процес навчання таких мереж складається з двох основних етапів:

- навчання на основі датасету (набору зображень з заздалегідь позначеними об’єктами) модель нейромережі навчається визначати структуру об’єктів та групувати їх у сегменти (рис. 2.7).
- тестування моделі. Для перевірки роботи на вхід нейронної мережі подається тестове зображення. Якщо модель визначила сегменти вірно, її можна використовувати в реальних умовах. Якщо ж ні – відбувається донавчання (рис. 2.8).



а)



б)

Рисунок 2.7 – Підготовка зображень для навчання моделі (а – зображення з ручним виділенням будинків, б – файл з координатами прямокутників, де перший стовпець є номером класу, з другого по п’ятий є координатами)

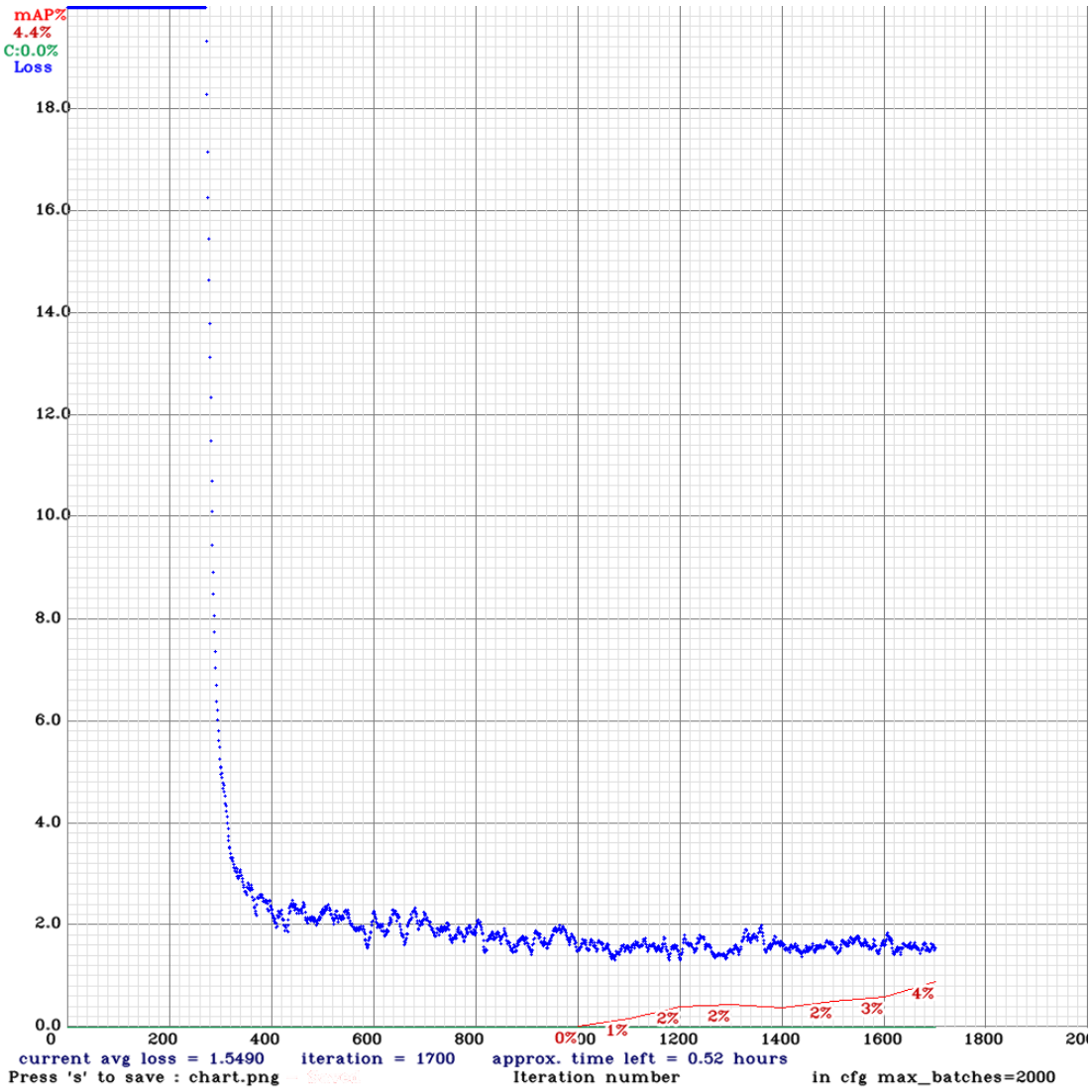


Рисунок 2.8 – Графік процесу навчання моделі. Сині крапки – втрати при навчанні (рекомендоване значення менше 0.3), червона лінія (mAP%, Mean Average Precision) - Середня середня точність, чим вище, тим краще відбувається розпізнавання об’єктів

2.2. Segment Anything Models

2.2.1. Segment Anything Model SA-1B (SAM)

Segment Anything Model SA-1B (SAM) дослівно перекладається як «Модель сегментування будь-чого» - нова модель ШІ від компанії Meta AI, що створена у 2023 році, яка може «вирізати» будь-який об’єкт в будь-якому зображенні. SAM — це швидка система сегментації з нульовим узагальненням

на незнайомі об’єкти та зображення без необхідності додаткового навчання. Це найбільший на сьогодні набір даних сегментації, з понад 1 мільярдом масок на 11 мільйонів ліцензованих зображень (на відміну від популярного набору даних COCO, середній розмір зображення моделі SAM 3300x4950 пікселів (проти 480x640 пікселів), через що точність розпізнавання об’єктів є дуже високою). Модель розроблено та навчено працювати з підказками (промптами) [34]. Кількість фотографій зроблених в Україні використаних для навчання, посідає дев’яте місце (рис. 2.9).

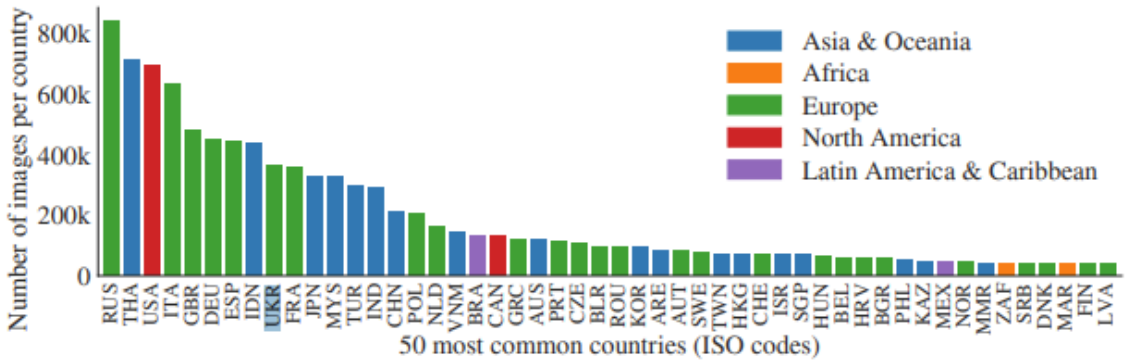


Рисунок 2.9 - Розрахунковий географічний розподіл зображень SA-1B.

SAM складається з трьох основних компонентів (рис. 2.10):

1. *Енкодер зображень*. Використовується попередньо навчена модель Vision Transformer (ViT), яка адаптована для обробки високороздільних зображень. Енкодер створює цифрове представлення зображення, яке може бути використане для подальшої обробки. Містить у собі 632 мільйони параметрів. Обробка займає приблизно 0.15 секунд на графічному процесорі NVIDIA A100.
2. *Енкодер підказок*. Підтримуються різні типи підказок, такі як точки, прямокутники, текстові описи і маски. Точки і прямокутники кодуються за допомогою позиційних кодувань і навчальних вбудовувань для кожного типу підказки. Текстові підказки обробляються за допомогою

текстового енкодера з моделі CLIP. Містить у собі 4 мільйони параметрів. Обробка займає приблизно 50 мілісекунд.

3. *Декодер масок*. Ця частина моделі відповідає за створення масок об'єктів на основі обробленого зображення і підказок. Використовує методи, що оновлюють цифрові представлення і допомагають моделі зрозуміти, які області зображення відповідають об'єктам. На виході модель створює маски, які показують контури і форми об'єктів на зображенні. Обробка займає приблизно 50 мілісекунд.

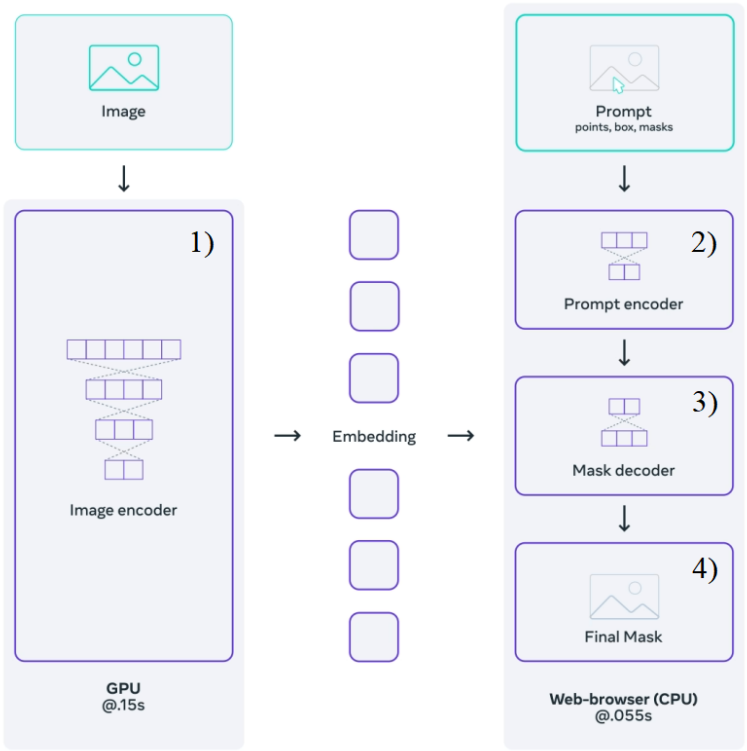


Рисунок 2.10 – Принцип взаємодії основних компонентів моделі SAM. 1) енкодер зображень, 2) енкодер підказок, 3) декодер масок, 4) результуюча маска

2.2.2. Fast Segment Anything Model (FastSAM)

Модель *Fast Segment Anything Model (FastSAM)* – це модель CNN Segment Anything, навчена лише на 2% набору даних SA-1B, опублікованого авторами SAM. За рахунок цього FastSAM досягає порівнянної

продуктивності з методом SAM у 50-кратній вищій швидкості роботи (рис. 2.11). FastSAM використовує модель YOLOv8-x або YOLOv8-s як основну частину своєї архітектури з розміром вхідних даних 1024.

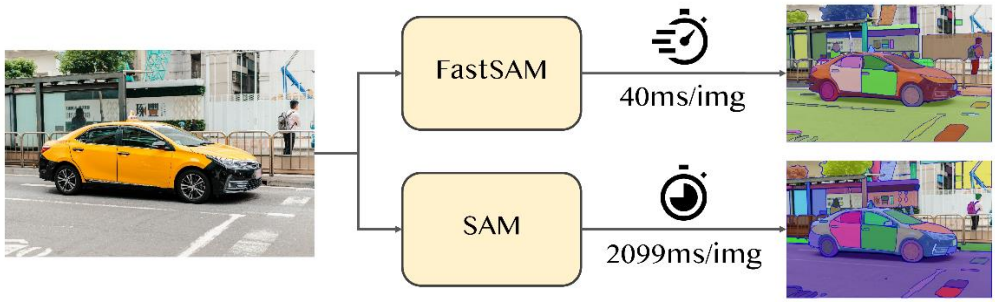


Рисунок 2.11 – Різниця швидкості обробки зображень моделями FastSAM та SAM (час обробки приблизний, результати для NVIDIA GeForce RTX 3090)

FastSAM значно знижує обчислювальні вимоги, зберігаючи при цьому конкурентоспроможну продуктивність, що робить її практичним вибором для різних завдань комп'ютерного зору. Доступні моделі, підтримувані завдання та режими роботи наведені в табл. 2.1 [35].

Таблиця 2.1 - Доступні моделі, підтримувані завдання та режими роботи

Тип моделі	**Назва файлів отриманих weight (ваг)**	**Задачі що підтримуються**	**Передбачення**	**Валідація**	**Тренування**	**Експорт**
FastSAM-s	FastSAM-s.pt	Сегментація екземплярів	✅	❌	❌	✅
FastSAM-x	FastSAM-x.pt	Сегментація екземплярів	✅	❌	❌	✅

Схематично архітектуру моделі FastSAM зображено на рис. 2.12.

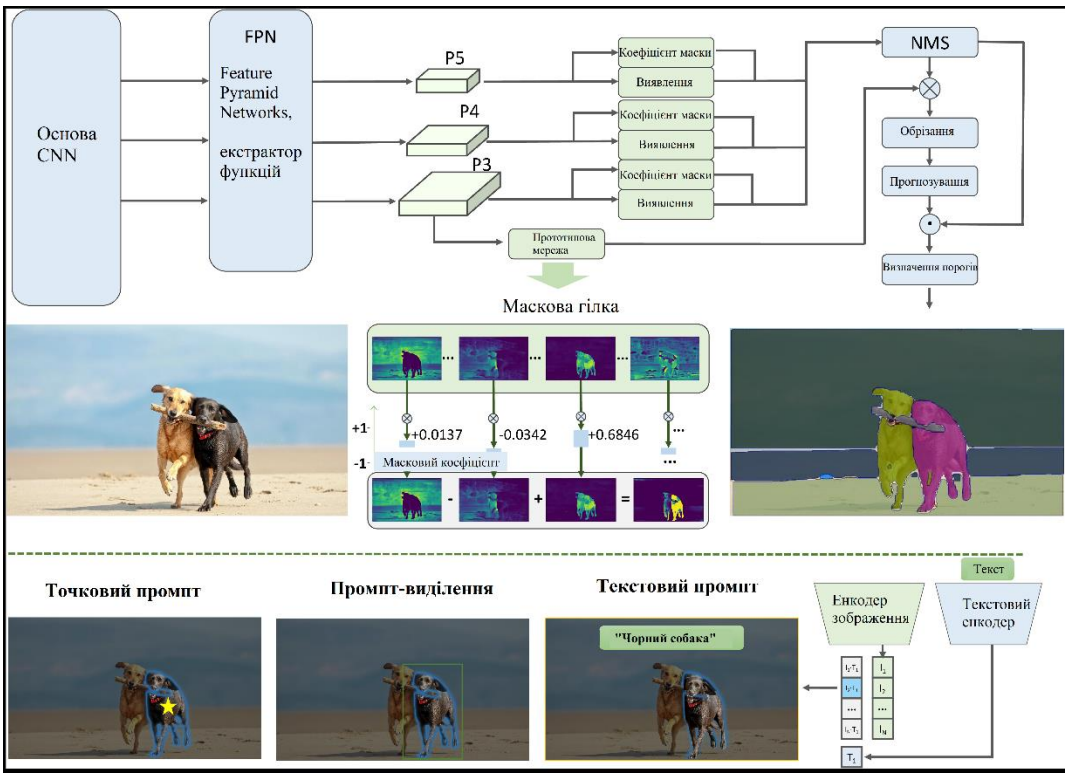


Рисунок 2.12 – Архітектура моделі SAM.

1. *Гілка «Основа»:* Модель CNN приймає зображення, далі «відправляє» у мережу піраміди функцій, або FPN. Це екстрактор функцій, який приймає одномасштабне зображення довільного розміру як вхідні дані та виводить карти функцій пропорційного розміру на кількох рівнях у повністю згортковий спосіб. Цей процес не залежить від магістральних згорткових архітектур. Тому він діє як загальне рішення для побудови пірамід функцій у глибоких згорткових мережах для використання в таких завданнях, як виявлення об’єктів. Побудова піраміди включає шляхи знизу вгору та шляхи зверху вниз.

2. *Гілка «Знаходження»:* Виконується розрахунок коефіцієнту маски кожного рівня.

3. *Гілка «Постобробка»:* Виконується фільтрація прогнозів виявлених об’єктів – процес називається «Немаксимальне придушення» (Non-maximum Suppression, NMS). Отримані результати обрізаються,

прогнозуються, визначаються пороги та виводиться отриманий кінцевий результат.

4. *«Маскова Гілка»* - відбувається визначення масок об’єктів. Є частиною гілки «Постобробка»

У випадку, коли використовується точковий, текстовий промпти або «Промпт-виділення», відбуваються кроки 1-4, які визначають усі об’єкти на зображенні, після чого енкодер зображення та тексту «обирають» необхідний об’єкт.

2.3. Висновки по розділу

Для аналізу об’єктів на зображенні використовується широкий спектр методів, включаючи як традиційні підходи обробки зображень, так і сучасні методи глибокого навчання.

Методи аналізу об'єктів на зображенні постійно еволюціонують, інтегруючи класичні методи та новітні досягнення глибокого навчання, що дозволяє досягати високої точності та ефективності у вирішенні завдань комп'ютерного зору.

- Традиційні методи, такі як виділення контурів за допомогою операторів Собеля, Прюітта, Робертса та алгоритму Кенні, залишаються важливими інструментами. Вони ефективно використовують властивості градієнтів яскравості для виявлення контурів об’єктів, але обмежені своєю чутливістю до шуму, складністю сцен та необхідністю ручної настройки параметрів.
- Згорткові нейронні мережі (CNN) значно підвищують ефективність аналізу зображень, забезпечуючи стійкість до оклюзії, складних сцен та змін освітлення.
- Методики сегментації, такі як семантична та екземплярна сегментація, стали стандартом у різних галузях, включаючи медицину, безпілотне

керування та аналіз супутникових знімків. Сучасні алгоритми сегментації використовують складні моделі, такі як Random Forest, K-Means, та новітні мережі, наприклад, U-Net для точного розпізнавання та класифікації об'єктів на зображеннях.

Особливої уваги заслуговують моделі Segment Anything Model (SAM) від Meta AI та FastSAM, що демонструють високу універсальність і здатність сегментувати будь-які об’єкти на зображеннях без додаткового навчання. Це досягнення свідчить про значний прогрес у розвитку методів аналізу зображень, де глибоке навчання відкриває нові можливості для автоматизації та точності обробки візуальних даних. Саме цей метод сегментації було обрано для розробки функціоналу програмного забезпечення.

РОЗДІЛ 3. РОЗРОБКА ПРОЄКТНОГО РІШЕННЯ

3.1. Обрання мови програмування. Дистрибутив Anaconda.

В якості мови програмування було обрано Python. *Python* — це універсальна високорівнева інтерпретована мова програмування (тобто може працювати в різних операційних системах без необхідності будь-яких змін у коді), яка широко використовується для різноманітних програм, таких як веб-розробка, аналіз даних, штучний інтелект, машинне навчання та наукові обчислення. Має велике та активне співтовариство розробників, які роблять внесок у його зростання та розвиток.

Python підтримує об’єктно орієнтоване програмування (ООП), яке дозволяє створювати «компонентний» код складних програм, розбиваючи їх на менші керовані об’єкти. Має величезну колекцію бібліотек і фреймворків, які пропонують рішення для таких завдань як: Маніпуляції та очищення даних (Panda), Наукові обчислення (NumPy & SciPy), Веб-розробка (Django & Flask), Машинне навчання (TensorFlow & PyTorch) та ін.

Для коректної роботи проєкту, було використано модифікований Python від Anaconda. У той час як Python є мовою загального призначення, Anaconda більше зосереджена на наукових дослідженнях даних і програмах машинного навчання. Тому це може бути не найкращим вибором для розробників, які працюють над проєктами, які не передбачають аналіз даних або наукові обчислення. Anaconda – це дистрибутив із відкритим вихідним кодом мов програмування Python і R, який використовується для обробки даних, машинного навчання та програм штучного інтелекту і містить понад 250 популярних наукових пакетів даних та інструментів керування для спрощення встановлення та розгортання пакетів. Anaconda поставляється з власним пакетом, середовищем і системою керування залежностями під назвою "conda" (це програма для управління пакетами наукових даних і управління середовищем). Подібно до Python, Anaconda також сумісна з різними

платформами, що дозволяє користувачам розробляти та розгортати програми на різних операційних системах без будь-яких змін у коді [33].

3.2. Загальний принцип роботи програми

Керування дроном відбувається автоматично без участі людини. Орієнтування у просторі відбувається завдяки сегментуванню відеопотоку (який є набором зображень). При визначенні цілі в заданому полі зору камери, відбувається активація коду, який виконує рух. Орієнтована схема автоматизованого переміщення дрону зображена на рис. 3.1: 1 – дрон летить при втраченому радіосигналі та виявляє перешкоду, 2 – виконується зупинка та рух вліво доти, доки перешкода не «вийде» з поля зору камери, 3 – поворот дрона на 90 градусів в сторону перешкоди та рух вліво доки перешкода не «вийде» з поля зору камери, 4 – рух вперед, повернення на пряму, з якої було виконано зміщення на кроці 2, 5 – поворот на 90 градусів вліво та продовження руху вперед до наступної перешкоди.

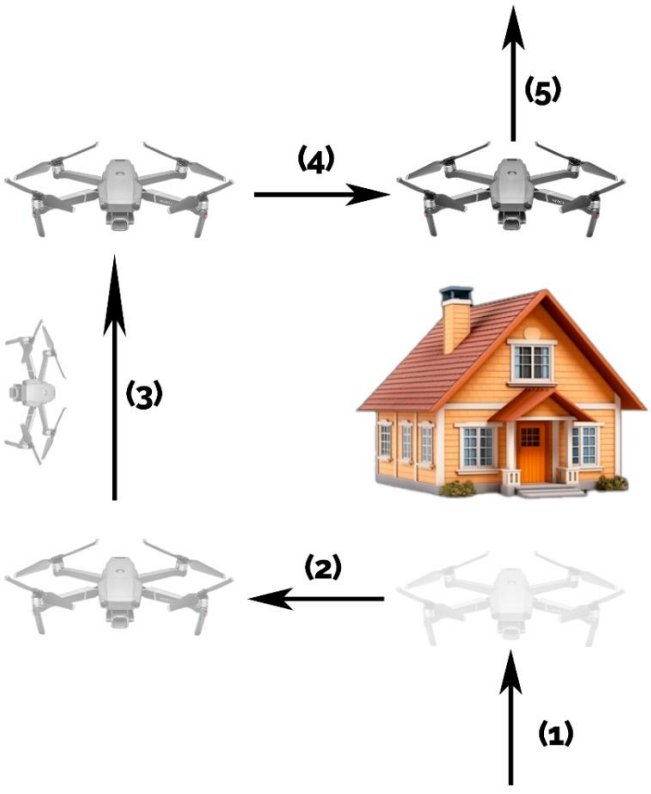


Рисунок 3.1 – Схема переміщення дрона при виявленні перешкоди

Алгоритм вирішення поставленої задачі зображений на рис. 3.2.

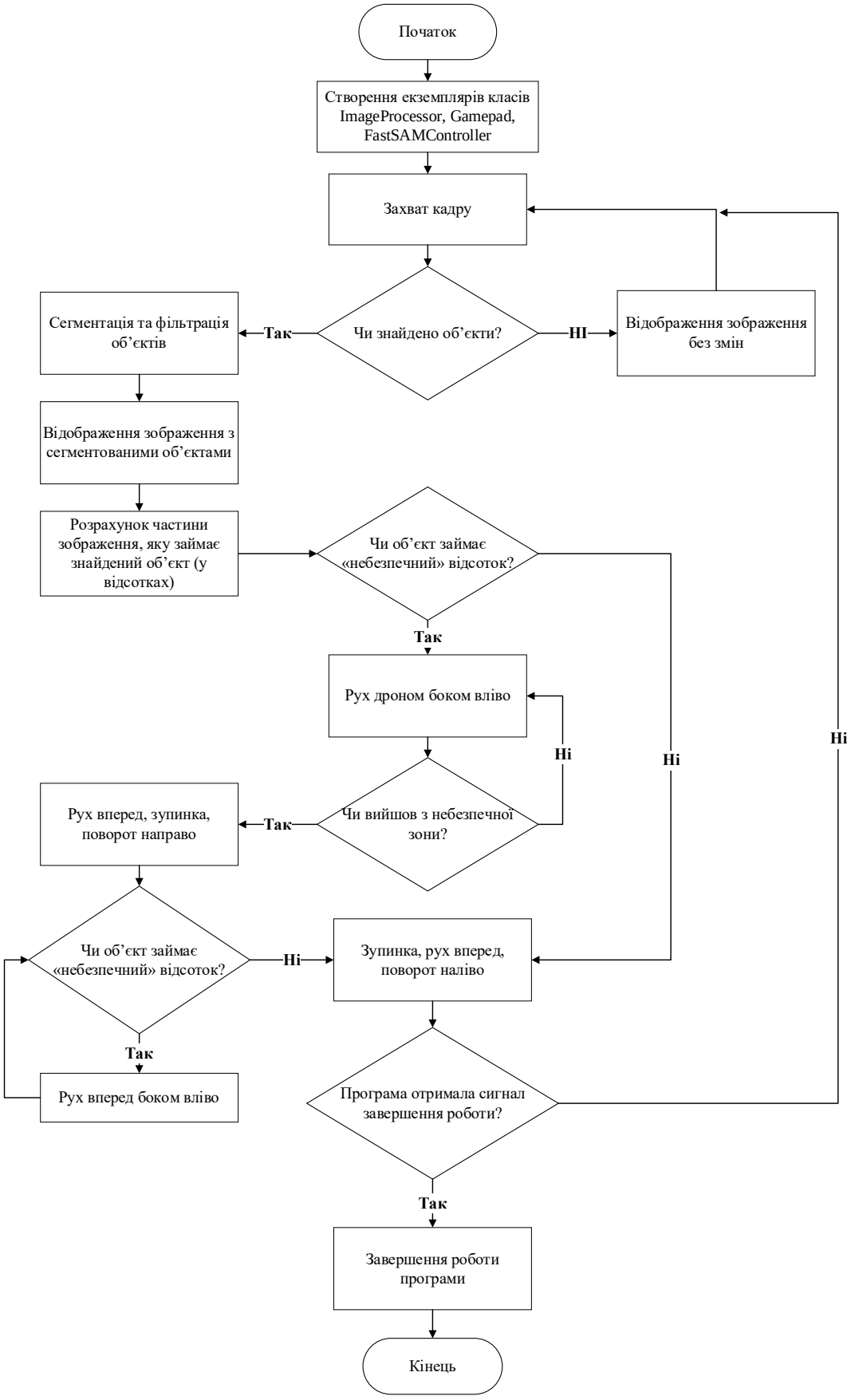


Рисунок 3.2 – Алгоритм вирішення задачі

3.3. Розробка програми

Розробимо архітектуру програмного забезпечення, діаграму класів та розгортання.

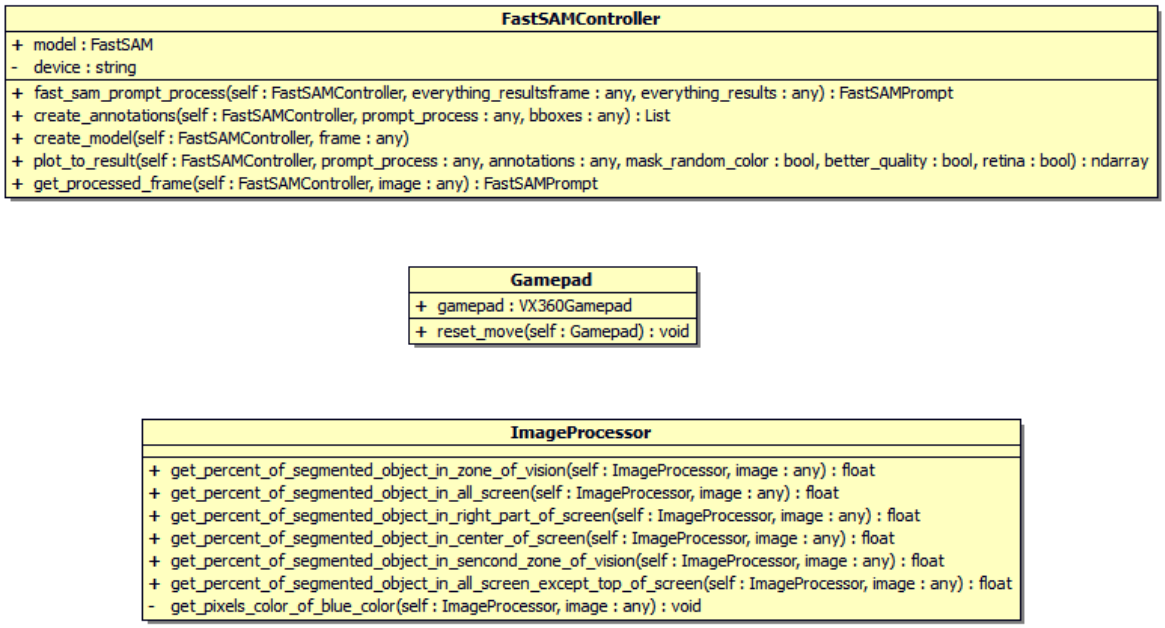


Рисунок 3.3 – UML-діаграма пакета segmentation_library.library

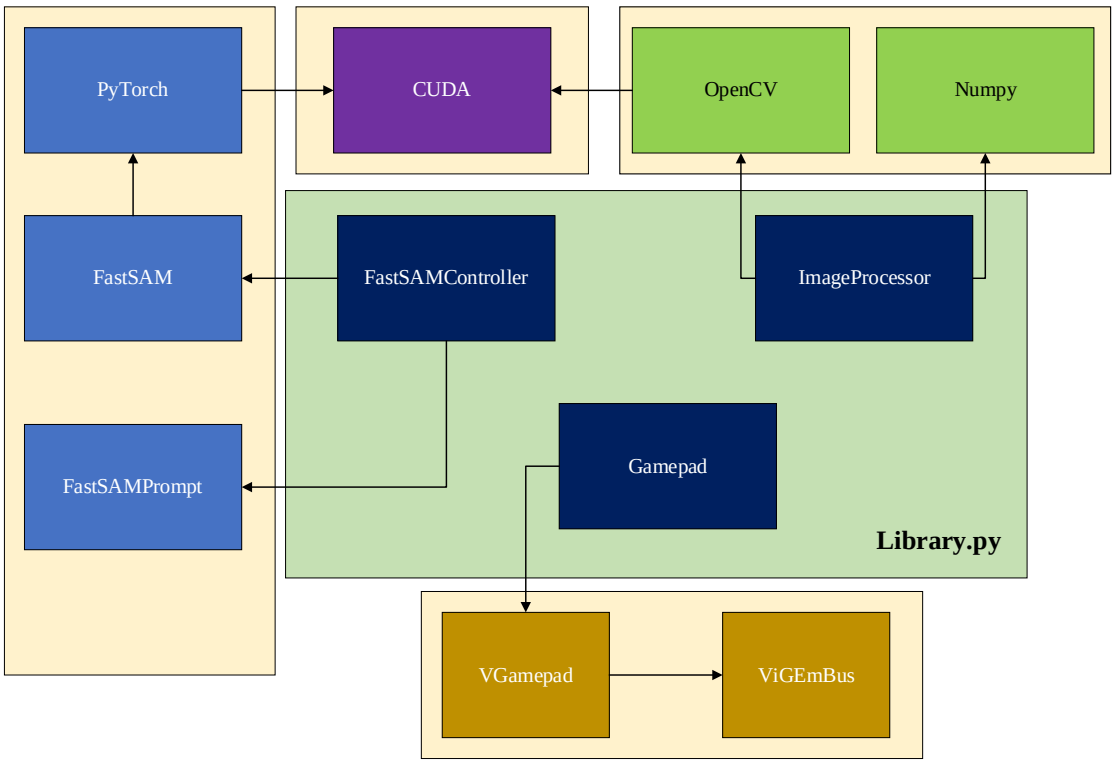


Рисунок 3.4 – Архітектура програмного забезпечення

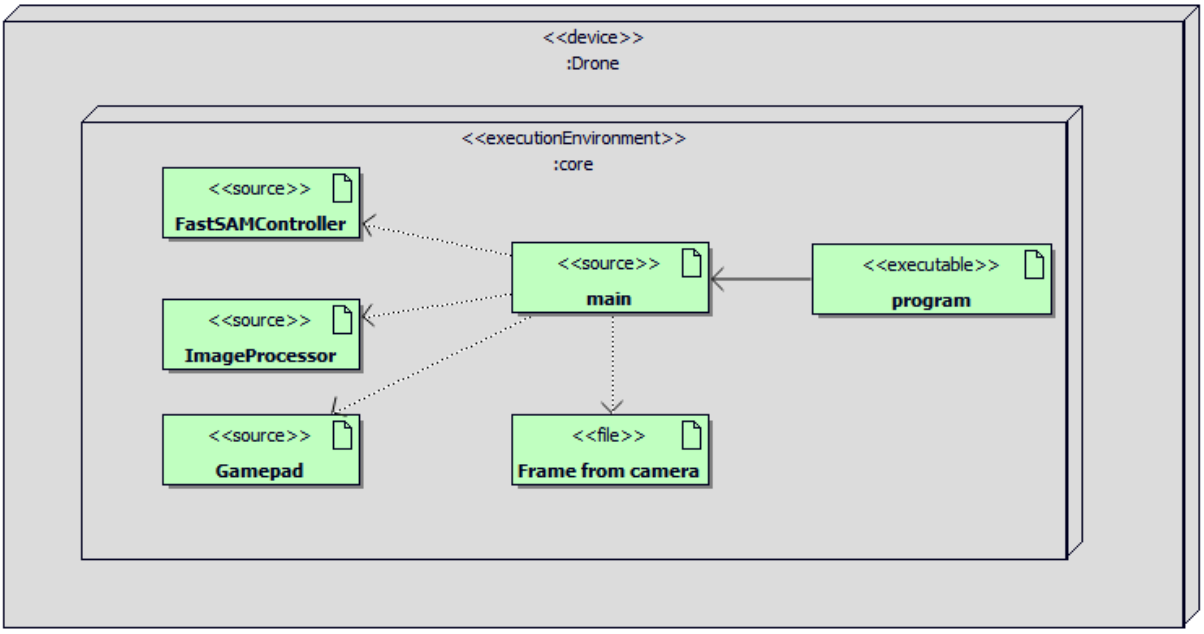


Рисунок 3.5 – Діаграма розгортання

Під час розробки програми буде використано наступні бібліотеки: vgamepad (для емуляції контролера), opencv і torch (для роботи з зображенням та виконання операцій машинного навчання), numpy (для роботи з зображенням у вигляді масивів і матриць) та FastSAM (сегментування зображення). Рекомендовано використання віртуального середовища: ініціалізація - python -m venv "E:\path_to_project\code\venv", активація - venv\Scripts\activate.

Встановлення пакетів виконується командою pip install opencv-python vgamepad numpy. Пакет fastsam встановлюється двома способами: 1) git clone https://github.com/CASIA-IVA-Lab/FastSAM.git, cd FastSAM, pip install -r requirements.txt, pip install git+https://github.com/openai/CLIP.git

2) pip install segment-anything-fast

У випадку розробки та тестування програми, що використовує Cuda, пакет torch повинен містити в собі інструменти для роботи з даними ядрами.

Для цього необхідно перейти на сайт https://pytorch.org/get-started/locally/ та згенерувати команду на встановлення пакету.

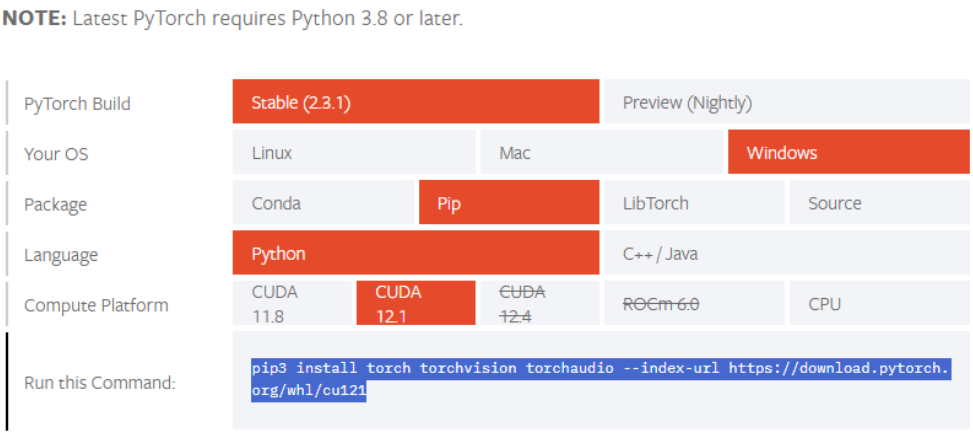


Рисунок 3.6 – Завантаження пакету torch

Оберемо пристрій, який буде відповідати за обробку зображення: cuda, mps, cpu. CPU – обробка ведеться на ядрах процесора. Cuda – імплементація тих самих функцій, що при використанні CPU, але розрахунки ведуться на GPU. MPS - GPU для MacOS пристроїв з фреймворком Metal. Для автоматичного обрання програмою пристрою використаємо наступний код:

```
DEVICE = torch.device(
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
```

Роздільна здатність віртуальної камери – 1024x576 пікселі. Задамо такі параметри в коді (cv2.CAP_PROP_FRAME_WIDTH та cv2.CAP_PROP_FRAME_HEIGHT відповідно). cv2.VideoCapture(1) – номер камери у системі:

```
cap = cv2.VideoCapture(1)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1024)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 576)
```

У випадку, якщо успішно вдалося ініціалізувати камеру, головний код програми буде розміщуватися в циклі while (цикл необхідний для захоплення кожного кадру відео), в якому обов'язково додається «код очікування» виходу з програми (оператор if). Після нього необхідно знищити усі відкриті бібліотекою opencv вікна (cv2.destroyAllWindows()) та від'єднати камеру (cap.release()):

```
import cv2
import torch

def main():
  while cap.isOpened():
    ...
    ...
    if cv2.waitKey(1) & 0xFF == ord('q'):
      break

  cv2.destroyAllWindows()
  cap.release()

if __name__ == "__main__":
    main()
```

Розробимо окремий пакет Python, який буде містити в собі увесь програмний код, що відповідає за роботу над зображенням. Для цього створимо папку segmentation_library, а в ній файл library.py. Побудова коду відбувається методом ООП (об'єктно орієнтоване програмування, метод структурування програми шляхом об'єднання пов'язаних властивостей і поведінки в окремі об'єкти), тому в файлі library.py буде представлено 3 головні класи:

- *Gamepad*: відповідає за операції керування дроном

- *FastSAMController*: відповідає за роботу з моделлю FastSAM (аналіз та сегментація обраних елементів)

- *ImageProcessor*: відповідає за роботу з зображенням

1) *Клас Gamepad*:

```
class Gamepad:
    def __init__(self):
        # увімкнення віртуального геймпада
        self.gamepad = vg.VX360Gamepad()

    def reset_move(self):
        self.gamepad.reset()
        self.gamepad.update()
```

Ініціалізується створенням екземпляру VX360Gamepad (Virtual Xbox360). Керування відбуватиметься через змінну gamepad. Функція reset_move() скидає поточне значення контролера керування (зупиняє рух).

2) *Клас FastSAMController*:

```
class FastSAMController:
    def __init__(self, model_path, device):
        self.model = FastSAM(model_path)
        self.device = device if device is not None else "cpu"
```

Ініціалізується модель FastSAM та в змінну device заноситься назва пристрою, що займається обробкою зображення. Якщо воно порожнє, зазначається cpu.

3) *Клас ImageProcessor*:

```
class ImageProcessor:
    def __init__(self):
        pass
```

На початку цей клас порожній. В головному файлі створимо екземпляри даних класів:

```
fast_sam_controller = FastSAMController('./fastsam/FastSAM-x.pt', DEVICE)
# ініціалізація контролера Геймпаду
gamepad_controller = Gamepad()
gamepad = gamepad_controller.gamepad
# ініціалізація оброблювача зображень
image_processor = ImageProcessor()
```

Переходимо до зчитування та обробки зображення. У циклі while cap.isOpened() виконаємо захват кадра:

```
frame_is_available, frame = cap.read()
```

frame_is_available і frame – кадр отриманий (True чи False) та сам кадр у вигляді масиву.

Ініціалізуємо (створимо) модель fastsam з отриманим кадром та виконаємо пошук усіх об’єктів на зображенні. Дану операцію необхідно виконувати кожен раз отримуючи новий кадр.

```
# ініціалізація моделі та сегментація усіх об'єктів на зображенні
everything_results = fast_sam_controller.create_model(frame)
```

Метод create_model(frame) розміщений в класі FastSAMController та виглядає наступним чином:

```
def create_model(self, frame):
    # Доступні налаштування:
    #
    # device - ім'я пристрою обробки зображення
    # retina_masks - промальовування високо деталізованих масок сегментації
    # conf - object confidence threshold (поріг довіри до об'єкта)
    # iou - threshold for filtering the annotations (iou поріг для фільтрації
анотацій), стандартний 0.7
    return self.model(
        source=frame,
        device=self.device,
        retina_masks=False,
        conf=0.4,
        iou=0.7,
        verbose=False,
    )
```

Метод self.model() є атрибутом екземпляра FastSAM(). Серед доступних налаштувань моделі зазначаємо наступні:

- source=frame – захоплений кадр

- device=self.device – ім’я пристрою, який відповідає за обробку зображення

- retina_masks=False – відповідає за промальовування деталізованих масок сегментації

- conf=0.4 – поріг довіри до об’єкта

- iou=0.7 – поріг для фільтрації промпту

- verbose=False – відображення інформації для відлагодження

У випадку, якщо на зображенні було знайдено хоча б один об’єкт, продовжуємо роботу над зображенням. В іншому випадку просто відображаємо зображення без змін (тобто захоплений кадр).

```
everything_results = fast_sam_controller.create_model(frame)
if everything_results:
  ...
else:
    cv2.imshow('img', frame)
```

Змінна everything_results містить у собі усю зібрану інформацію. Для виведення інформації про кожну категорію об’єктів в консоль можна використати наступний код:

```
# Відображення масок (masks) об'єкта та квадратів (boxes) з категорії під індексом
нуль [0].
print(everything_results[0].masks.shape) print(everything_results[0].boxes.shape)

# Виділення зеленим квадратом усіх знайдених об'єктів з категорії під індексом нуль
[0]
 for box in everything_results[0].boxes:
     box = box.xyxy.cpu().numpy()[0]
     cv2.rectangle(frame, (int(box[0]), int(box[1])), (int(box[2]), int(box[3])), (0,
255, 0), 2)
```

Знайдені об’єкти необхідно сегментувати та відфільтрувати. Для цього frame та everything_results надаються в метод fast_sam_prompt_process(). Метод fast_sam_prompt_process розміщений в класі FastSAMController:

```
def fast_sam_prompt_process(self, frame, everything_results):
    return FastSAMPrompt(image=frame, results=everything_results, device=self.device)
```

Виклик методу:

```
prompt_process = fast_sam_controller.fast_sam_prompt_process(frame,
everything_results)
```

Зображення відсегментоване для всіх об’єктів. Тепер необхідно залишити визначення лише тих, які потрапляють в заданий прямокутник (умовний кут огляду камери).

```
def create_annotations(self, prompt_process, bboxes):
    return prompt_process.box_prompt(bboxes=bboxes)
```

```
ann = fast_sam_controller.create_annotations(prompt_process, bboxes=
[[ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
ZONE_OF_VISION_X2_Y2[1]]])
```

ZONE_OF_VISION_X1_Y1, ZONE_OF_VISION_X2_Y2 – координати лівого верхнього та правого нижнього кута прямокутника (рис. 3.7).

В даному випадку було обрано метод створення анотацій box_prompt. Усього наявно 4 методи:

1) *prompt_process.everything_prompt()* – виділення усіх об’єктів, стандартний запит

2) *prompt_process.text_prompt(text='house, walls etc')* – виділення лише тих об’єктів, які словами були описані (в даному випадку будинок, стіни).

3) *prompt_process.point_prompt(points=[[620, 360]], pointlabel=[1])* – точковий запит. Виділення об’єктів, в яких лежать точкм за координатами points.

4) *prompt_process.box_prompt(bboxes=bboxes)* – виділення усіх об’єктів, які знаходяться в прямокутних областях визначених координатами bboxes.

Хоча метод box_prompt не є надійним і відбувається захоплення зайвих об’єктів, метод text_prompt, що більш точний в роботі, вимагає більше часу та потужностей на обробку зображення.

Отриману інформацію про сегментовані об’єкти в заданій області необхідно відобразити на зображенні. Метод plot_to_result() розміщений в класі FastSAMController та виглядає наступним чином:

```
img = fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
```

```
def plot_to_result(self, prompt_process, annotations, mask_random_color=False,
better_quality=False, retina=True):
    return prompt_process.plot_to_result(annotations=annotations,
mask_random_color=mask_random_color,
                                     better_quality=better_quality, retina=retina)
```

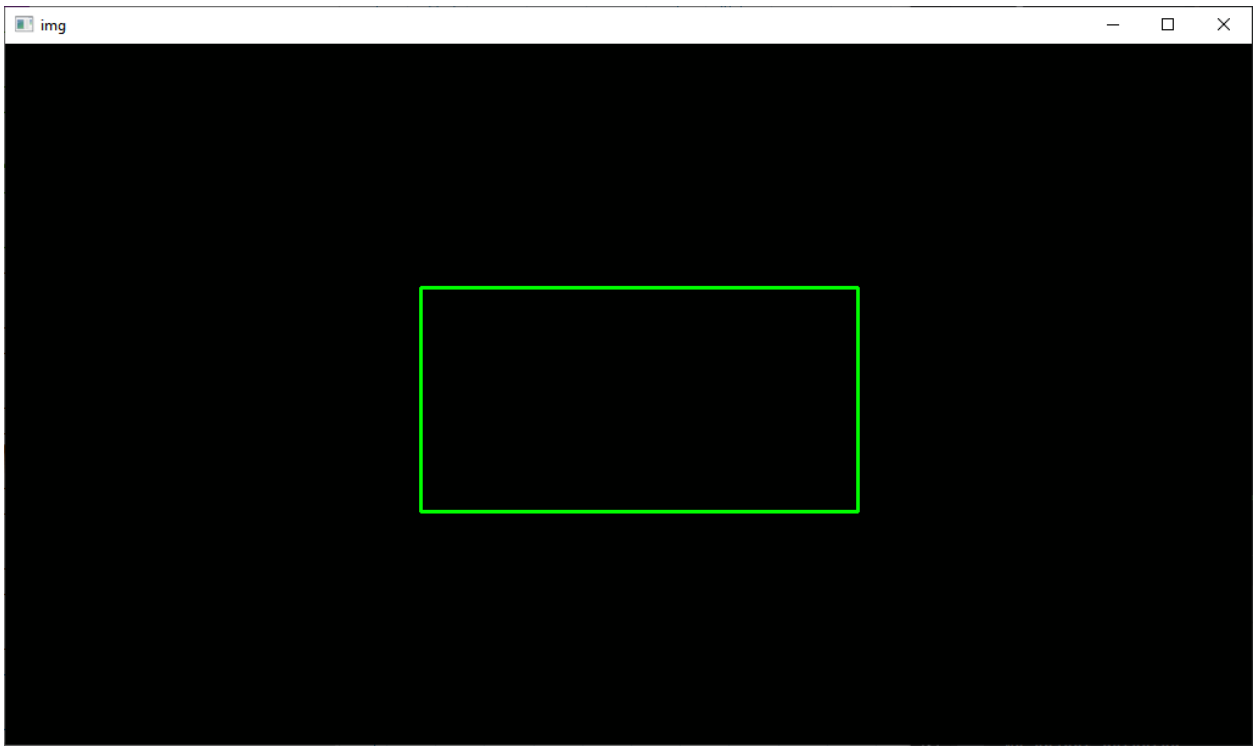


Рисунок 3.7 – «Поле зору» камери

Розрахуємо відсоток місця, яке займає сегментований об’єкт на зображенні. Дана інформація буде необхідна при визначенні ступеня «небезпеки» - тобто близькість до об’єкта:

```
percent = image_processor.get_percent_of_segmented_object_in_all_screen(img)
print(f"\rPercentage of the segmented object on the entire screen: {percent}",
end="")
```

```
def get_percent_of_segmented_object_in_all_screen(self, image):
    # Виділяємо область зеленого прямоугольника
    x1, y1 = (0, 0)
    x2, y2 = (1024, 576)
    roi = image[y1:y2, x1:x2]

    # Перетворимо область на колірний простір HSV
    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

    # Визначаємо діапазон синього кольору в HSV
    lower_blue = np.array([100, 50, 50])
    upper_blue = np.array([140, 255, 255])

    # Створюємо маску для синього кольору
    blue_mask = cv2.inRange(hsv_roi, lower_blue, upper_blue)
    cv2.imshow('blue_mask', blue_mask)
    # Розраховуємо відсоток площі, зайнятої синім кольором
    blue_area = cv2.countNonZero(blue_mask)
    total_area = roi.shape[0] * roi.shape[1]
```

```
    blue_percentage = (blue_area / total_area) * 100
    return blue_percentage
```

Результат сегментації зображено на рисунку 3.8.

Рисунок 3.8 – Результат сегментації зображення

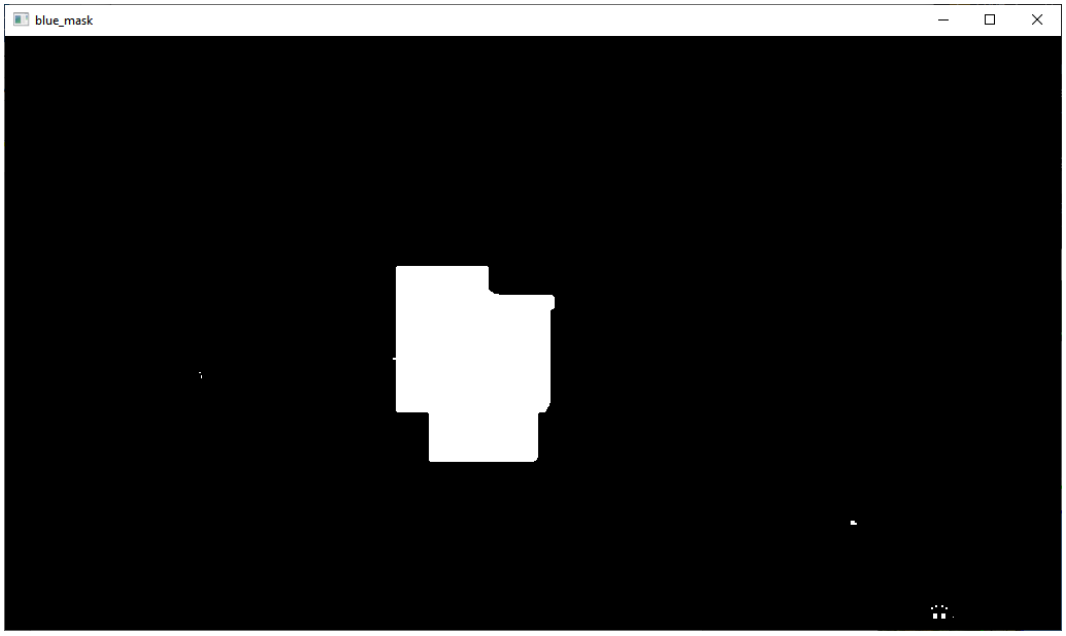


Рисунок 3.9 – Маска виділеного об’єкта

Перевіряємо «близькість» камери до об’єкта. Якщо об'єкт займає "небезпечний" відсоток місця, запускається код на автоматичне керування дроном (практичним шляхом було встановлено оптимальний відсоток від 40% до 100%).

Програмний код руху дрона умовно ділиться на декілька частин:

1) Розпочинаємо підрахунок часу, який дрон витратить на зміщення вбік від об’єкта – *на рис. 3.1 це стрілка №2*. Для цього в змінну time_start_automatic помістимо значення time.perf_counter() – час у вигляді timestamp. Даний крок необхідний для повернення дрона на початковий напрямок руху (рис. 3.1, стрілка №4).

```
if MIN_PERCENT_OF_DANGER < percent <= MAX_PERCENT_OF_DANGER:
    time_start_automatic = time.perf_counter()
```

Доки відсоток є «небезпечним», отримується кадр, виконуються ті самі кроки щодо аналізу зображення та його сегментації, при цьому відбувається рух боком вліво джойстиком. Для виключення хибних спрацювань щодо percent_automatic < MIN_PERCENT_OF_DANGER - MINUS_PERCENT_OF_DANGER додано потрійну перевірку. Коли дрон виходить з «небезпечної зони», робиться двосекундна пауза і після цього зупиняється рух.

```
i = 0
while MIN_PERCENT_OF_DANGER < percent_automatic:
    frame_is_available, frame = cap.read()
    prompt_process = fast_sam_controller.get_processed_frame(frame)
    ann = fast_sam_controller.create_annotations(prompt_process, bboxes=
    [[ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
      ZONE_OF_VISION_X2_Y2[1]]])
    img = fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
    percent_automatic =
image_processor.get_percent_of_segmented_object_in_zone_of_vision(img)
    while percent_automatic < MIN_PERCENT_OF_DANGER - MINUS_PERCENT_OF_DANGER and i >
3:
        frame_is_available, frame = cap.read()
        prompt_process = fast_sam_controller.get_processed_frame(frame)
        ann = fast_sam_controller.create_annotations(prompt_process, bboxes=
        [[ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
          ZONE_OF_VISION_X2_Y2[1]]])
        img = fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
        percent_automatic =
```

```
image_processor.get_percent_of_segmented_object_in_all_screen(img)
        print(
            f"\rPercentage of the segmented object on the right part of the screen:
{percent_automatic}",
            end="")
        i += 1

    gamepad.right_joystick_float(x_value_float=-0.6, y_value_float=0)
    gamepad.update()
time.sleep(2)
gamepad_controller.reset_move()
```

Загальний час руху вираховується наступним чином:

```
time_stop_automatic = time.perf_counter()
total_time_automatic = time_stop_automatic - time_start_automatic
print(f"\nStop. Total moving time: {total_time_automatic}")
```

Варто відмітити ключовий момент – керування віртуальним геймпадом виконується в два етапи: задання напрямку руху (****_joystick_float, відповідає за нахил джойстика, x_value_float та y_value_float є осями OX та OY відповідно в межах від -1.0 до 1.0) та «активації» руху (update()). У випадку, якщо другий метод не викликано, буде продовжуватися попередній активований напрям руху. Значення нахилу джойстиків та часу time.sleep() визначається практичним шляхом.

2) *Рис. 3.1, стрілка №3*. Спочатку рух вперед протягом 3 секунд (для «заїзду» в область видимості лівої стінки об'єкта), зупинка, поворот на 90 градусів вправо, зупинка.

```
print(f"Moving forward...")
gamepad.right_joystick_float(x_value_float=0.0, y_value_float=0.8)
gamepad.update()
time.sleep(3)
gamepad_controller.reset_move()
print(f"Stop.")
time.sleep(2)
print("Turning right by 90 degrees...")
gamepad.left_joystick_float(x_value_float=0.7, y_value_float=0.0)  # поворот направо
на 90 градусів
gamepad.update()
time.sleep(3.5)
gamepad_controller.reset_move()
print(f"Stop.")
time.sleep(2)
```

Далі відбувається рух лівим боком вліво доки перед зоною видимості камери з координатами SECOND_ZONE_OF_VISION_X1_Y1 та SECOND_ZONE_OF_VISION_X2_Y2 буде видно об’єкт і він займатиме менше 10 відсотків центрального прямокутника (екран умовно ділиться на прямокутники 3x3, тобто координати 341;0 та 682;576) та в межах від 40 до 60 відсотків в правій частині зображення (з потрійною перевіркою). В першому випадку це означатиме, що дрон має можливість рухатися вперед, в другому – об’єкт точно не завадить рухові. Cv2.imwrite записує вигляд з камери в даний момент в файл.

```
percent_automatic_two = 100
if percent_automatic_two > 20:
    i = 0
    while True:
        frame_is_available, frame = cap.read()
        prompt_process = fast_sam_controller.get_processed_frame(frame)
        ann = fast_sam_controller.create_annotations(prompt_process, bboxes=
        [[SECOND_ZONE_OF_VISION_X1_Y1[0] + 171, SECOND_ZONE_OF_VISION_X1_Y1[1],
         SECOND_ZONE_OF_VISION_X2_Y2[0] + 512,
         SECOND_ZONE_OF_VISION_X2_Y2[1]]])
        img = fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
        percent_automatic_two =
image_processor.get_percent_of_segmented_object_in_right_part_of_screen(
            img)
        percent_automatic_two_center =
image_processor.get_percent_of_segmented_object_in_center_of_screen(
            img)
        if percent_automatic_two_center < 10:
            cv2.imwrite('img percent_automatic_two_center ' + str(time.time()) +
".jpg", img)
            break
        if (60 > percent_automatic_two and percent_automatic_two_center < 40) and i >
3:
            cv2.imwrite('img percent_automatic_two' + str(time.time()) + ".jpg", img)
            break
        print(f"\rMoving sideways to the left... ({percent_automatic_two})", end='')
        gamepad.right_joystick_float(x_value_float=-0.7,
                                     y_value_float=0.0)  # Поки буде видно об'єкт,
продовжувати рух вбік боком
        gamepad.update()
        cv2.rectangle(img, SECOND_ZONE_OF_VISION_X1_Y1, SECOND_ZONE_OF_VISION_X2_Y2,
(0, 255, 0), 2)
        cv2.imwrite('img' + str(time.time()) + ".jpg", img)
        i += 1

gamepad_controller.reset_move()
print(f"\nStop.")
time.sleep(2)
```

Рисунок 3.10 – Знімок консолі під час автоматичного руху

Рисунок 3.11 – Знімок, зроблений під час автоматичного руху повз об’єкт

3) *Рис. 3.1, стрілка №4*. Рух на вихідну позицію та поворот вліво на 90 градусів.

```
print(f"Moving forward...")
gamepad.right_joystick_float(x_value_float=0.0, y_value_float=0.6)  # вперед
gamepad.update()
time.sleep(total_time_automatic)
gamepad_controller.reset_move()
print(f"Stop.")
time.sleep(2)
print(f"Turning left by 90 degrees...")
gamepad.left_joystick_float(x_value_float=-0.7, y_value_float=0.0)  # поворот вліво
на 90 градусов
gamepad.update()
time.sleep(3.5)
gamepad_controller.reset_move()
```

```
print(f"Stop.")
time.sleep(1)
```

Завершується код нанесенням на зображення зони видимості камери.

```
cv2.rectangle(img, ZONE_OF_VISION_X1_Y1, ZONE_OF_VISION_X2_Y2, (0, 255, 0), 2)
cv2.imshow('img', img)
```

Проведемо модульне тестування (Unit-тестування). Для цього створимо файл unit_tests.py з іменем класу TestSegmentation та аргументомм unittest.TestCase. Імпортуємо необхідні класи та константи. Реалізуємо функцію setUp, в якій відбувається задання необхідних змінних для проведення тестування.

```
import unittest
import cv2
from segmentation_library.library import FastSAMController, Gamepad, ImageProcessor,
ZONE_OF_VISION_X1_Y1, \
    ZONE_OF_VISION_X2_Y2

class TestSegmentation(unittest.TestCase):
    def setUp(self):
        self.fast_sam_controller = FastSAMController('./fastsam/FastSAM-x.pt', 'cpu')
        self.gamepad_controller = Gamepad()
        self.image_processor = ImageProcessor()
        self.cap = cv2.VideoCapture(1)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1024)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 576)

if __name__ == '__main__':
    unittest.main()
```

В ньому виконаємо тестування таких функцій:

- *ініціалізація моделі* (test_create_model):

```
def test_create_model(self):
    frame, everything_results, _ = self._get_frame_and_results()
    self.assertIsNotNone(everything_results)
```

- *виконання сегментації зображення з фільтрування результатів (test_fast_sam_prompt_process):*

```
def test_fast_sam_prompt_process(self):
    _, _, prompt_process = self._get_frame_and_results()
    self.assertIsNotNone(prompt_process)
```

- *створення анотацій для об'єкта в заданій області (test_create_annotations):*

```
def test_create_annotations(self):
    _, _, prompt_process = self._get_frame_and_results()
    ann = self.fast_sam_controller.create_annotations(prompt_process, bboxes=[
        [ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
ZONE_OF_VISION_X2_Y2[1]]])
    self.assertIsNotNone(ann)
```

- *нанесення сегментованих об'єктів на зображення (test_plot_to_result):*

```
def test_plot_to_result(self):
    _, _, prompt_process = self._get_frame_and_results()
    ann = self.fast_sam_controller.create_annotations(prompt_process, bboxes=[
        [ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
ZONE_OF_VISION_X2_Y2[1]]])
    img = self.fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
    self.assertIsNotNone(img)
```

- *отримання відсотку зайнятості сегментованим об'єктом простору поля видимості камери (test_get_percent_of_segmented_object_in_all_screen):*

```
def test_get_percent_of_segmented_object_in_all_screen(self):
    _, _, prompt_process = self._get_frame_and_results()
    ann = self.fast_sam_controller.create_annotations(prompt_process, bboxes=[
        [ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
ZONE_OF_VISION_X2_Y2[1]]])
    img = self.fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
    percent = self.image_processor.get_percent_of_segmented_object_in_all_screen(img)
    self.assertGreaterEqual(percent, 0)
    self.assertLessEqual(percent, 100)
```

Для оптимізації коду було створено додаткову функцію _get_frame_and_results, яка повертає отриманий кадр, отримані результати сегментації та відфільтровані результати:

```
def _get_frame_and_results(self):
    frame_is_available, frame = self.cap.read()
    everything_results = self.fast_sam_controller.create_model(frame)
    prompt_process = self.fast_sam_controller.fast_sam_prompt_process(frame,
everything_results)
    return frame, everything_results, prompt_process
```

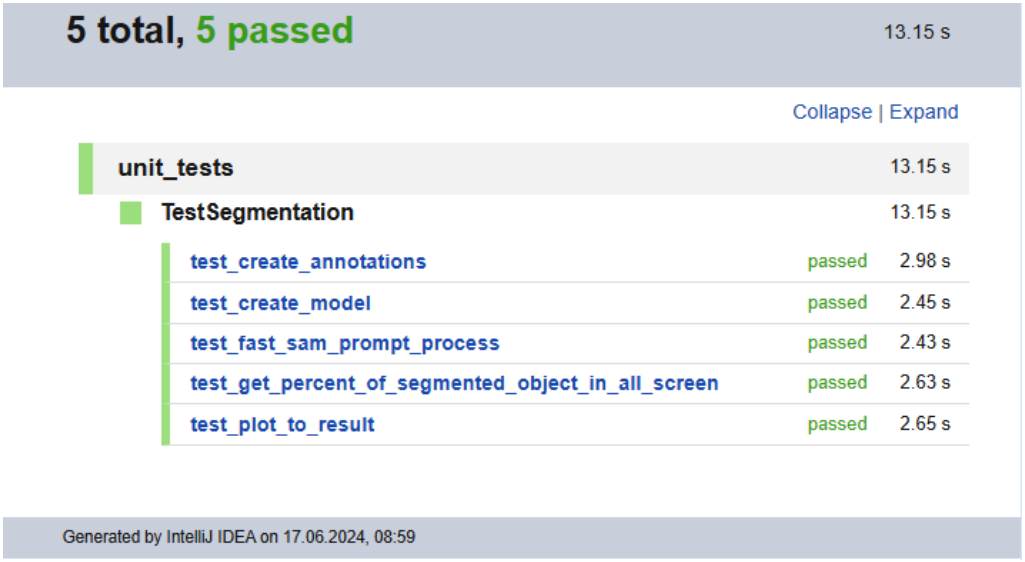


Рисунок 3.12 – Результати виконання тестування

3.4. Висновки по розділу

Було розроблено програмне рішення для автоматизованого керування дроном, орієнтуючись на відеопотік з камери та використовуючи сучасні методи сегментації зображень. Вибір мови програмування Python та дистрибутиву Anaconda був обґрунтований потребою в математичних обчисленнях і аналізі даних, що є ключовими аспектами у задачах комп'ютерного зору.

Основні переваги та ключові моменти розробленого рішення є наступними:

- *Універсальність і простота використання Python*: Високорівнева мова програмування з великим набором бібліотек дозволяє ефективно вирішувати задачі комп'ютерного зору. Python широко використовується в наукових дослідженнях завдяки своїй зрозумілості та потужним можливостям.
- *Anaconda*: Забезпечує простоту встановлення та управління пакетами, необхідними для обробки даних, спрощує процес управління залежностями та пакетами.
- *Сегментація зображень з використанням згорткової нейронної мережі*: Ефективний алгоритм сегментації дозволяє точно визначати об'єкти на

зображеннях, що є критично важливим для орієнтації дрона в просторі. Використання моделі FastSAM забезпечило високу точність і швидкість обробки зображень у реальному часі.

- *Автоматизація процесу керування дроном*: Використовуючи обробку відеопотоку, дрон може самостійно виявляти перешкоди та обходити їх, що зменшує необхідність втручання людини та підвищує безпеку і надійність роботи дрона. Така автоматизація дозволяє розширити сфери застосування дронів, зокрема в умовах, де пряме керування оператором є ускладненим або неможливим.
- *Тестування і оптимізація*: Система була ретельно протестована в різних умовах, що дозволило виявити і усунути потенційні проблеми.
- *Масштабованість і адаптивність*: Система може бути легко адаптована для різних типів дронів та різних сценаріїв використання. Завдяки модульній архітектурі, можна швидко змінювати і вдосконалювати окремі компоненти системи

Розроблена система показала високу ефективність у тестових умовах, демонструючи здатність автоматично виявляти та обходити перешкоди. Це рішення може бути корисним для різних застосувань, включаючи автономну навігацію, пошуково-рятувальні операції та інші області, де важлива автономність і точність керування.

ВИСНОВКИ

Дана робота є надзвичайно актуальною в умовах сучасної війни, оскільки впровадження технологій безпілотного бою, особливо дронів, відіграє важливу роль у протистоянні агресору. Описані у роботі FPV дрони є ефективними завдяки своїй низькій вартості, високій швидкості та маневреності. Однак відсутність радіосигналу внаслідок використання РЕБ знижує позитивні аспекти даної «зброї».

Розробка симулятора керування дроном в умовах радіоелектронної боротьби є проєктом, що може забезпечити успішне застосування програмного керування дроном навіть у випадку втрати радіосигналу. Технологія машинного зору в безпілотних апаратах є новою і такою, що швидко розвивається. Майже всі аналогічні проєкти є закритого типу та мають непублічний код.

Головним «інструментом» в роботі машинного зору стала модель FastSAM (на основі моделі SAM), що є новою, потужною та відносно ресурсоефективною розробкою. Функціональність програми включає автоматизоване керування дроном з можливістю виявлення та обходу перешкод, реалізоване через сегментування відеопотоку та активацію відповідних рухів дрона на віртуальному пульті керування, що дозволяє здійснювати точне виявлення цілей та перешкод.

Під час розробки програми було встановлено наступне:

- оптимальним шляхом оминання перешкод є рух повз об'єкт з лівої чи правої сторони об'єкта;
- на певних етапах автономного руху запроваджено потрійну перевірку на хибну логіку спрацювання коду внаслідок невірної сегментації зображення;
- результати сегментації слабо залежать від форми об’єкта, його контрастності, яскравості і т.д.;

- середній час оминання перешкоди – одна хвилина (залежить від форми та площі, яку займає перешкода).

Оцінюючи результати роботи, можна зазначити, що:

- було успішно розроблено прототип програми аналізу зображення для автоматичного управління безпілотним дроном
- розроблений код має можливості до подальшого розвитку та вдосконалення алгоритмів машинного зору, інтеграції з іншими системами, таких як супутникові системи зв'язку, та розвитку нових технологій, таких як машинний зір, для покращення ефективності та безпеки застосування літальних апаратів, додання нового функціоналу тощо.
- інтеграція розробленого програмного забезпечення з іншими системами керування та моніторингу дозволить створити комплексні рішення для різних сфер застосування. Використання хмарних обчислень для обробки даних з дронів може значно підвищити ефективність і масштабованість системи.
- розширення можливостей застосування технології на різні типи дронів та інші роботизовані системи, а також використання технології в нових сферах, таких як агрономія, екологічний моніторинг та інші, відкриває нові горизонти для розвитку та впровадження цих рішень.

Подальший розвиток алгоритмів сегментування зображень дозволить підвищити точність і швидкість виявлення перешкод та цілей. Можливе впровадження більш складних методів машинного навчання, таких як глибокі нейронні мережі, для покращення розпізнавання об'єктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ArmyInform: "Як за допомогою FPV-дронів можна ефективно нищити ворога" [Електронний ресурс]. - Режим доступу: https://armyinform.com.ua/2023/04/07/yak-za-dopomogoyu-fpv-droniv-mozhna-efektyvno-nyshhyty-voroga/ .

2. Glavcom: "В Україні розробили військовий «дрон-камікадзе»" [Електронний ресурс]. - Режим доступу: https://glavcom.ua/techno/hitech/v-ukrajini-rozrobili-viyskoviy-dron-kamikadze-763484.html .

3. RadioSvoboda: "FPV атакуватимуть без участі людини? Що таке машинний зір, який вже на фронті" [Електронний ресурс]. - Режим доступу: https://www.radiosvoboda.org/a/viyna-droniv-mashynnyy-zir-fpv/32882525.html .

4. Wayback Machine: "Дрон" [Електронний ресурс]. - Режим доступу: https://web.archive.org/web/20240113133525/https://uk.wikipedia.org/wiki/%D0%94%D1%80%D0%BE%D0%BD .

5. New Atlas: "Taranis drone demonstrates stealth in latest test" [Електронний ресурс]. - Режим доступу: https://newatlas.com/taranis-stealth/32963/

6. Вікіпедія: "Війна безпілотників" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B9%D0%BD%D0%B0_%D0%B1%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D1%82%D0%BD%D0%B8%D0%BA%D1%96%D0%B2 .

7. Вікіпедія: "Безпілотний апарат" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D1%82%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82

8. Мілітарний: "В Україні розробили ударний FPV-дрон KH-S7" [Електронний ресурс]. - Режим доступу: https://mil.in.ua/uk/news/v-ukrayini-rozrobyly-udarnyj-fpv-dron-kh-s7/

9. BestDrone: "Як FPV дрони змінюють обличчя сучасного неба України" [Електронний ресурс]. - Режим доступу: https://www.bestdrone.com.ua/fpv-%D0%B4%D1%80%D0%BE%D0%BD%D0%B8/ .

10. Вікіпедія: "Безпілотний надводний апарат" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D1%82%D0%BD%D0%B8%D0%B9_%D0%BD%D0%B0%D0%B4%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82

11. Мілітарний портал: "Ударні морські дрони" [Електронний ресурс]. - Режим доступу: https://mil.in.ua/uk/articles/udarni-morski-drony/

12. Вікіпедія: "Українські безпілотні надводні апарати" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D1%81%D1%8C%D0%BA%D1%96_%D0%B1%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D1%82%D0%BD%D1%96_%D0%BD%D0%B0%D0%B4%D0%B2%D0%BE%D0%B4%D0%BD%D1%96_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%B8_%D0%BF%D1%96%D0%B4_%D1%87%D0%B0%D1%81_%D0%A0%D0%BE%D1%81%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%BE-%D1%83%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D1%81%D1%8C%D0%BA%D0%BE%D1%97_%D0%B2%D1%96%D0%B9%D0%BD%D0%B8_(%D0%B7_2022)

13. Вікіпедія: "Безпілотний підводний апарат" [Електронний ресурс]. - Режим доступу:

https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D1%82%D0%BD%D0%B8%D0%B9_%D0%BF%D1%96%D0%B4%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82

14. Вікіпедія: "Дистанційно-керований підводний апарат" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D0%B0%D0%BD%D1%86%D1%96%D0%B9%D0%BD%D0%BE-%D0%BA%D0%B5%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D0%BF%D1%96%D0%B4%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82

15. «Експерт», український урядовий портал: "Північна Корея випробувала підводний безпілотник, здатний нести ядерну зброю" [Електронний ресурс]. - Режим доступу: https://web.archive.org/web/20230324195641/https://expert.in.ua/novosti/24032023-%D0%9F%D1%96%D0%B2%D0%BD%D1%96%D1%87%D0%BD%D0%B0-%D0%9A%D0%BE%D1%80%D0%B5%D1%8F-%D0%B2%D0%B8%D0%BF%D1%80%D0%BE%D0%B1%D1%83%D0%B2%D0%B0%D0%BB%D0%B0-%D0%BF%D1%96%D0%B4%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%B9-%D0%B1%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D1%82%D0%BD%D0%B8%D0%BA-%D0%B7%D0%B4%D0%B0%D1%82%D0%BD%D0%B8%D0%B9-%D0%BD%D0%B5%D1%81%D1%82%D0%B8-%D1%8F%D0%B4%D0%B5%D1%80%D0%BD%D1%83-%D0%B7%D0%B1%D1%80%D0%BE%D1%8E/

16. Кабінет Міністрів України: "Відкриття ринку БпЛА, перші у світі ударні роти та морські дрони: результати Армії дронів за 2023 рік" [Електронний ресурс]. - Режим доступу: https://www.kmu.gov.ua/news/vidkryttia-rynku-bpla-pershi-u-sviti-udarni-roty-ta-morski-drony-rezultaty-armii-droniv-za-2023-rik. .

17. Українська правда: "Як нове покоління дронів може змінити хід війни" [Електронний ресурс]. - Режим доступу: https://www.pravda.com.ua /articles/2024/01/25/7438746/ .

18. УНІАН: "РЕБ не врятує: фахівець пояснив небезпеку дронів з «машинним зором»" [Електронний ресурс]. - Режим доступу: https://www.unian.ua/weapons/reb-ne-vryatuye-fahivec-poyasniv-nebezpeku-droniv-z-mashinnim-zorom-12479343.html .

19. AI Conference: "Штучний інтелект і машинний зір: можливості технологій" [Електронний ресурс]. - Режим доступу: https://aiconference.com.ua/uk/news/iskusstvenniy-intellekt-i-mashinnoe-zrenie-vozmognosti-tehnologiy-97504 .

20. Google Ngram Viewer: "Computer Vision vs Machine Vision" [Електронний ресурс]. - Режим доступу: https://books.google.com/ngrams/graph?content=computer+vision%2Cmachine+vision&year_start=1800&year_end=2019&corpus=en-US-2019&smoothing=3 .

21. Вікіпедія: "Комп’ютерний зір" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B8%D0%B9_%D0%B7%D1%96%D1%80

22. The Transmitted: "Що таке машинний зір (Machine Vision, MV)" [Електронний ресурс]. - Режим доступу: https://thetransmitted.com/adlucem/shho-take-mashynnyj-zir-machine-vision-mv/ .

23. FacerUA: "Пакет для новачків: що таке комп’ютерний зір?" [Електронний ресурс]. - Режим доступу: https://www.facerua.com/pakiet-dlia-novachkiv-shcho-takie-kompiutiernii-zir/

24. Robotics UA: "Системи машинного зору роботів. Історія розвитку, сфери застосування, плани на майбутнє" [Електронний ресурс]. - Режим доступу: https://robotics.ua/systemy-mashynnoho-zrenyia.-ystoryia-prymery-plany/ .

25. UA Кореспондент: "Армія дронів: що відомо про нові роти у ЗСУ" [Електронний ресурс]. - Режим доступу: https://ua.korrespondent.net/ukraine/4557754-armiia-droniv-scho-vidomo-pro-novi-roty-u-zsu .

26. Міністерство цифрової трансформації: "В Україні запустили Defense Tech Cluster «Brave1», який стимулює розвиток військових інновацій та оборонних технологій" [Електронний ресурс]. - Режим доступу: https://thedigital.gov.ua/news/v-ukraini-zapustili-defense-tech-cluster-brave1-yakiy-stimulyuvatime-rozvitok-viyskovikh-innovatsiy-ta-oboronnikh-tekhnologiy .

27. Brave1: "Кластер підтримки defense tech розробок в Україні" [Електронний ресурс]. - Режим доступу: https://brave1.gov.ua/ .

28. Вікіпедія: "Brave1" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Brave1 .

29. Dev UA: "AirUnit: як пройти шлях від постачальника дронів до великого тренінгового центру для ЗСУ та виробника FPV" [Електронний ресурс]. - Режим доступу: https://dev.ua/news/promo-1688461910 .

30. Міністерство оборони України: "Дрон SAKER SCOUT зі штучним інтелектом Міноборони допустило до експлуатації у ЗСУ" [Електронний ресурс]. - Режим доступу: https://t.me/ministry_of_defense_ua/7951 .

31. Харківський національний університет радіоелектроніки: " Розпізнавання тривимірних об'єктів у режимі реального часу за допомогою згорткових

штучних нейронних мереж" [Електронний ресурс]. - Режим доступу: https://openarchive.nure.ua/server/api/core/bitstreams/0233a226-d9d9-4446-8083-bcc989ee65c6/content.

32. Тернопільський національний економічний університет: "Інформаційна технологія структурно-статистичної ідентифікації ієрархічних об’єктів" [Електронний ресурс]. - Режим доступу: http://dspace.wunu.edu.ua/bitstream/316497/45046/1/Dis%20%D0%97%D0%B0%D0%B3%D0%BE%D1%80%D0%BE%D0%B4%D0%BD%D1%8F.pdf

33. Вікіпедія: "Алгоритм Кенні" [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%B5%D0%BD%D0%BD%D1%96

34. Arxiv.org: "Segment Anything" [Електронний ресурс]. - Режим доступу: https://arxiv.org/pdf/2304.02643 .

35. Arxiv.org: "Fast Segment Anything" [Електронний ресурс]. - Режим доступу: https://arxiv.org/pdf/2306.12156 .

36. Datacamp: "Anaconda vs Python: Exploring Their Key Differences" [Електронний ресурс]. - Режим доступу: https://www.datacamp.com/blog/anaconda-vs-python-key-differences

ДОДАТОК А
ЛІСТИНГ КОДУ ПРОГРАМИ (ГОЛОВНИЙ ФАЙЛ)

```
import torch
import cv2
import time

from segmentation_library.library import FastSAMController, Gamepad, ImageProcessor,
ZONE_OF_VISION_X1_Y1, \
    ZONE_OF_VISION_X2_Y2, MIN_PERCENT_OF_DANGER, MAX_PERCENT_OF_DANGER,
CRITICAL_ZONE_X2_Y2, CRITICAL_ZONE_X1_Y1, \
    MINUS_PERCENT_OF_DANGER, SECOND_ZONE_OF_VISION_X2_Y2, SECOND_ZONE_OF_VISION_X1_Y1

# обрання пристрою, що оброблює зображення
DEVICE = torch.device(
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)


def main():
    print(f"Using device: {DEVICE}")
    # ініціалізація контролера FastSAM
    fast_sam_controller = FastSAMController('./fastsam/FastSAM-x.pt', DEVICE)
    # ініціалізація контролера Геймпаду
    gamepad_controller = Gamepad()
    gamepad = gamepad_controller.gamepad
    # ініціалізація оброблювача зображень
    image_processor = ImageProcessor()

    # Налаштування віртуальної камери з заданням роздільної здатності (1024x576)
    cap = cv2.VideoCapture(1)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1024)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 576)

    while cap.isOpened():

        frame_is_available, frame = cap.read()

        start = time.perf_counter()
        # ініціалізація моделі та сегментація усіх об'єктів на зображенні
        everything_results = fast_sam_controller.create_model(frame)
        if everything_results:
            # Відображення масок (masks) об'єкта та квадратів (boxes) з категорії під
індексом нуль [0].
            # Кількість категорій на зображенні може бути різною.
            #
            # print(everything_results[0].masks.shape) та
print(everything_results[0].boxes.shape)

            # Виділення зеленим квадратом усі знайдені об'єкти з категорії під
індексом нуль [0]
            # for box in everything_results[0].boxes:
            #     box = box.xyxy.cpu().numpy()[0]
            #     cv2.rectangle(frame, (int(box[0]), int(box[1])), (int(box[2]),
```

```
int(box[3])), (0, 255, 0), 2)

            end = time.perf_counter()

            # знайдені об'єкти надаються в fast_sam_prompt_process, що сегментує
зображення
            prompt_process = fast_sam_controller.fast_sam_prompt_process(frame,
everything_results)

            # Промпт для пошуку об'єктів, які потрапляють лише в заданий прямокутник
            ann = fast_sam_controller.create_annotations(prompt_process, bboxes=
            [[ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1],
ZONE_OF_VISION_X2_Y2[0], ZONE_OF_VISION_X2_Y2[1]]])

            # "Об'єднання" результатів сегментації та інформації про об'єкти з
промпту
            img = fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)

            # Визначення відсотка місця, що займається об'єктом на екрані (весь
екран)
            percent =
image_processor.get_percent_of_segmented_object_in_all_screen(img)
            print(f"\rPercentage of the segmented object on the entire screen:
{percent}", end="")

            # Якщо об'єкт займає "небезпечний" відсоток місця, запускається код на
автоматичне керування дроном
            if MIN_PERCENT_OF_DANGER < percent <= MAX_PERCENT_OF_DANGER:
                time_start_automatic = time.perf_counter()
                percent_automatic = percent
                print(f"\nMoving to the left ({percent_automatic})", end="")
                i = 0
                while MIN_PERCENT_OF_DANGER < percent_automatic:
                    frame_is_available, frame = cap.read()
                    prompt_process = fast_sam_controller.get_processed_frame(frame)
                    ann = fast_sam_controller.create_annotations(prompt_process,
bboxes=
                    [[ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1],
ZONE_OF_VISION_X2_Y2[0],
                      ZONE_OF_VISION_X2_Y2[1]]])
                    img =
fast_sam_controller.plot_to_result(prompt_process=prompt_process, annotations=ann)
                    percent_automatic =
image_processor.get_percent_of_segmented_object_in_zone_of_vision(img)
                    while percent_automatic < MIN_PERCENT_OF_DANGER -
MINUS_PERCENT_OF_DANGER and i > 3:
                        frame_is_available, frame = cap.read()
                        prompt_process =
fast_sam_controller.get_processed_frame(frame)
                        ann = fast_sam_controller.create_annotations(prompt_process,
bboxes=
                        [[ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1],
ZONE_OF_VISION_X2_Y2[0],
                          ZONE_OF_VISION_X2_Y2[1]]])
                        img =
fast_sam_controller.plot_to_result(prompt_process=prompt_process, annotations=ann)
                        percent_automatic =
image_processor.get_percent_of_segmented_object_in_all_screen(img)
                        print(
                            f"\rPercentage of the segmented object on the right part
```

```
of the screen: {percent_automatic}",
                              end="")
                        i += 1

                    gamepad.right_joystick_float(x_value_float=-0.6, y_value_float=0)
# вліво
                    gamepad.update()

                time.sleep(2)
                gamepad_controller.reset_move()
                time_stop_automatic = time.perf_counter()
                total_time_automatic = time_stop_automatic - time_start_automatic
                print(f"\nStop. Total moving time: {total_time_automatic}")
                print(f"Moving forward...")
                gamepad.right_joystick_float(x_value_float=0.0, y_value_float=0.8)  #
немного вперед
                gamepad.update()
                time.sleep(3)
                gamepad_controller.reset_move()
                print(f"Stop.")
                time.sleep(2)
                print("Turning right by 90 degrees...")
                gamepad.left_joystick_float(x_value_float=0.7, y_value_float=0.0)  #
поворот направо на 90 градусов
                gamepad.update()
                time.sleep(3.5)
                gamepad_controller.reset_move()
                print(f"Stop.")
                time.sleep(2)
                percent_automatic_two = 100
                if percent_automatic_two > 20:
                    i = 0
                    while True:
                        frame_is_available, frame = cap.read()
                        prompt_process =
fast_sam_controller.get_processed_frame(frame)
                        ann = fast_sam_controller.create_annotations(prompt_process,
bboxes=
                        [[SECOND_ZONE_OF_VISION_X1_Y1[0] + 171,
SECOND_ZONE_OF_VISION_X1_Y1[1],
                         SECOND_ZONE_OF_VISION_X2_Y2[0] + 512,
                         SECOND_ZONE_OF_VISION_X2_Y2[1]]])
                        img =
fast_sam_controller.plot_to_result(prompt_process=prompt_process, annotations=ann)
                        percent_automatic_two =
image_processor.get_percent_of_segmented_object_in_right_part_of_screen(
                            img)
                        percent_automatic_two_center =
image_processor.get_percent_of_segmented_object_in_center_of_screen(
                            img)
                        if percent_automatic_two_center < 10:
                            cv2.imwrite('img percent_automatic_two_center ' +
str(time.time()) + ".jpg", img)
                            break
                        if (60 > percent_automatic_two and
percent_automatic_two_center < 40) and i > 3:
                            cv2.imwrite('img percent_automatic_two' +
str(time.time()) + ".jpg", img)
                            break
                        print(f"\rMoving sideways to the left...
({percent_automatic_two})", end='')
```

```
                        gamepad.right_joystick_float(x_value_float=-0.7,
                                                     y_value_float=0.0)  # Поки буде
видно об'єкт, продовжувати рух вбік боком
                        gamepad.update()
                        cv2.rectangle(img, SECOND_ZONE_OF_VISION_X1_Y1,
SECOND_ZONE_OF_VISION_X2_Y2, (0, 255, 0), 2)
                        cv2.imwrite('img' + str(time.time()) + ".jpg", img)
                        i += 1

                gamepad_controller.reset_move()
                print(f"\nStop.")
                time.sleep(2)
                print(f"Moving forward...")
                gamepad.right_joystick_float(x_value_float=0.0, y_value_float=0.6)  #
вперед
                gamepad.update()
                time.sleep(total_time_automatic)
                gamepad_controller.reset_move()
                print(f"Stop.")
                time.sleep(2)
                print(f"Turning left by 90 degrees...")
                gamepad.left_joystick_float(x_value_float=-0.7, y_value_float=0.0)  #
поворот вліво на 90 градусів
                gamepad.update()
                time.sleep(3.5)
                gamepad_controller.reset_move()
                print(f"Stop.")
                time.sleep(1)
            else:
                gamepad_controller.reset_move()

            fps = 1 / (end - start)
            print(f"\rFPS: {int(fps)}", end="")
            # Відображення FPS на зображенні
            cv2.putText(img, f'FPS: {int(fps)}', (20, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 0, 255), 2)
            # Для відлагодження виділення області, в якій йде пошук об'єктів
            # cv2.rectangle(img, CRITICAL_ZONE_X1_Y1, CRITICAL_ZONE_X2_Y2, (0, 0,
255), 2)
            cv2.rectangle(img, ZONE_OF_VISION_X1_Y1, ZONE_OF_VISION_X2_Y2, (0, 255,
0), 2)
            cv2.imshow('img', img)

        else:
            cv2.rectangle(frame, ZONE_OF_VISION_X1_Y1, ZONE_OF_VISION_X2_Y2, (0, 255,
0), 2)

            cv2.imshow('img', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
    cap.release()


if __name__ == "__main__":
    main()
```

ДОДАТОК Б

ЛІСТИНГ КОДУ ПРОГРАМИ (ФАЙЛ LIBRARY.PY)

```
import cv2
import numpy as np
from matplotlib import pyplot as plt # допоміжні
from sklearn.cluster import KMeans # допоміжні

from fastsam import FastSAM, FastSAMPrompt
import vgamepad as vg

# ZONE_OF_VISION_X1_Y1 = (361, 200)
ZONE_OF_VISION_X1_Y1 = (341, 200)
# ZONE_OF_VISION_X2_Y2 = (682, 384)
ZONE_OF_VISION_X2_Y2 = (700, 384)

SECOND_ZONE_OF_VISION_X1_Y1 = (0, 0)
SECOND_ZONE_OF_VISION_X2_Y2 = (512, 576)

CRITICAL_ZONE_X1_Y1 = (480, 232)
CRITICAL_ZONE_X2_Y2 = (544, 297)

ZONE_OF_RIGHT_PART_SCREEN_X1_Y1 = (512, 0)
ZONE_OF_RIGHT_PART_SCREEN_X2_Y2 = (1024, 576)

MIN_PERCENT_OF_DANGER = 40
MINUS_PERCENT_OF_DANGER = 10
MAX_PERCENT_OF_DANGER = 100


class Gamepad:
    def __init__(self):
        # увімкнення віртуального геймпада
        self.gamepad = vg.VX360Gamepad()

    def reset_move(self):
        self.gamepad.reset()
        self.gamepad.update()


class FastSAMController:
    def __init__(self, model_path, device):
        self.model = FastSAM(model_path)
        self.device = device if device is not None else "cpu"

    def fast_sam_prompt_process(self, frame, everything_results):
        return FastSAMPrompt(image=frame, results=everything_results,
device=self.device)

    def create_annotations(self, prompt_process, bboxes):
        # Задання анотацій для виділення усіх об'єктів - стандартний запит
        # ann = prompt_process.everything_prompt()

        # Текстовий запит - працює частіше краще, ніж два попередні, але дуже
повільно
        # ann = prompt_process.text_prompt(text='house, walls etc')

        # Точковий запит
        # # points default [[0,0]] [[x1,y1],[x2,y2]]
        # # point_label default [0] [1,0] 0:background, 1:foreground
```

```
        # ann = prompt_process.point_prompt(points=[[620, 360]], pointlabel=[1])

        return prompt_process.box_prompt(bboxes=bboxes)

    def create_model(self, frame):
        # Доступні налаштування:
        #
        # device - ім'я пристрою обробки зображення
        # retina_masks - промальовування високо деталізованих масок сегментації
        # imgsz
        # conf - object confidence threshold (поріг довіри до об'єкта)
        # iou - threshold for filtering the annotations (iou поріг для фільтрації
анотацій), стандартний 0.7
        # image = None
        # image_embedding = None
        return self.model(
            source=frame,
            device=self.device,
            retina_masks=False,
            conf=0.4,
            iou=0.7,
            verbose=False,
        )

    def plot_to_result(self, prompt_process, annotations, mask_random_color=False,
better_quality=False, retina=True):
        return prompt_process.plot_to_result(annotations=annotations,
mask_random_color=mask_random_color,
                                             better_quality=better_quality,
retina=retina)

    def get_processed_frame(self, image):
        results = self.create_model(image)
        return self.fast_sam_prompt_process(image, results)


class ImageProcessor:
    def __init__(self):
        pass
    def get_percent_of_segmented_object_in_zone_of_vision(self, image):
        # Виділяємо область зеленого прямоугольника
        x1, y1 = ZONE_OF_VISION_X1_Y1
        x2, y2 = ZONE_OF_VISION_X2_Y2
        roi = image[y1:y2, x1:x2]

        # Перетворимо область на колірний простір HSV
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

        # Визначаємо діапазон синього кольору в HSV
        lower_blue = np.array([100, 50, 50])
        upper_blue = np.array([140, 255, 255])

        # Створюємо маску для синього кольору
        blue_mask = cv2.inRange(hsv_roi, lower_blue, upper_blue)
        cv2.imshow('blue_mask', blue_mask)
        # Розраховуємо відсоток площі, зайнятої синім кольором
        blue_area = cv2.countNonZero(blue_mask)
        total_area = roi.shape[0] * roi.shape[1]
        blue_percentage = (blue_area / total_area) * 100
        return blue_percentage
```

```
    def get_percent_of_segmented_object_in_all_screen(self, image):
        # Виділяємо область зеленого прямоугольника
        x1, y1 = (0, 0)
        x2, y2 = (1024, 576)
        roi = image[y1:y2, x1:x2]

        # Перетворимо область на колірний простір HSV
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

        # Визначаємо діапазон синього кольору в HSV
        lower_blue = np.array([100, 50, 50])
        upper_blue = np.array([140, 255, 255])

        # Створюємо маску для синього кольору
        blue_mask = cv2.inRange(hsv_roi, lower_blue, upper_blue)
        cv2.imshow('blue_mask', blue_mask)
        # Розраховуємо відсоток площі, зайнятої синім кольором
        blue_area = cv2.countNonZero(blue_mask)
        total_area = roi.shape[0] * roi.shape[1]
        blue_percentage = (blue_area / total_area) * 100
        return blue_percentage

    def get_percent_of_segmented_object_in_right_part_of_screen(self, image):
        # Виділяємо область зеленого прямоугольника
        x1, y1 = ZONE_OF_RIGHT_PART_SCREEN_X1_Y1
        x2, y2 = ZONE_OF_RIGHT_PART_SCREEN_X2_Y2
        roi = image[y1:y2, x1:x2]

        # Перетворимо область на колірний простір HSV
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

        # Визначаємо діапазон синього кольору в HSV
        lower_blue = np.array([100, 50, 50])
        upper_blue = np.array([140, 255, 255])

        # Створюємо маску для синього кольору
        blue_mask = cv2.inRange(hsv_roi, lower_blue, upper_blue)
        cv2.imshow('blue_mask', blue_mask)
        # Розраховуємо відсоток площі, зайнятої синім кольором
        blue_area = cv2.countNonZero(blue_mask)
        total_area = roi.shape[0] * roi.shape[1]
        blue_percentage = (blue_area / total_area) * 100
        return blue_percentage

    def get_percent_of_segmented_object_in_center_of_screen(self, image):
        # Виділяємо область зеленого прямоугольника
        x1, y1 = (341, 0)
        x2, y2 = (682, 576)
        roi = image[y1:y2, x1:x2]

        # Перетворимо область на колірний простір HSV
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

        # Визначаємо діапазон синього кольору в HSV
        lower_blue = np.array([100, 50, 50])
        upper_blue = np.array([140, 255, 255])

        # Створюємо маску для синього кольору
        blue_mask = cv2.inRange(hsv_roi, lower_blue, upper_blue)
        cv2.imshow('blue_mask', blue_mask)
        # Розраховуємо відсоток площі, зайнятої синім кольором
```

```
        blue_area = cv2.countNonZero(blue_mask)
        total_area = roi.shape[0] * roi.shape[1]
        blue_percentage = (blue_area / total_area) * 100
        return blue_percentage

    def get_percent_of_segmented_object_in_sencond_zone_of_vision(self, image):
        # Виділяємо область зеленого прямоугольника
        x1, y1 = SECOND_ZONE_OF_VISION_X1_Y1
        x2, y2 = SECOND_ZONE_OF_VISION_X2_Y2
        roi = image[y1:y2, x1:x2]

        # Перетворимо область на колірний простір HSV
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

        # Визначаємо діапазон синього кольору в HSV
        lower_blue = np.array([100, 50, 50])
        upper_blue = np.array([140, 255, 255])

        # Створюємо маску для синього кольору
        blue_mask = cv2.inRange(hsv_roi, lower_blue, upper_blue)
        cv2.imshow('blue_mask', blue_mask)
        # Розраховуємо відсоток площі, зайнятої синім кольором
        blue_area = cv2.countNonZero(blue_mask)
        total_area = roi.shape[0] * roi.shape[1]
        blue_percentage = (blue_area / total_area) * 100
        return blue_percentage

    def get_percent_of_segmented_object_in_all_screen_except_top_of_screen(self,
image):
        # Виділяємо область зеленого прямоугольника
        x1, y1 = (0, 288)
        x2, y2 = (1024, 576)
        roi = image[y1:y2, x1:x2]

        # Перетворимо область на колірний простір HSV
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

        # Визначаємо діапазон синього кольору в HSV
        lower_blue = np.array([100, 50, 50])
        upper_blue = np.array([140, 255, 255])

        # Створюємо маску для синього кольору
        blue_mask = cv2.inRange(hsv_roi, lower_blue, upper_blue)
        cv2.imshow('blue_mask', blue_mask)
        # Розраховуємо відсоток площі, зайнятої синім кольором
        blue_area = cv2.countNonZero(blue_mask)
        total_area = roi.shape[0] * roi.shape[1]
        blue_percentage = (blue_area / total_area) * 100
        return blue_percentage

    def get_pixels_color_of_blue_color(self, image):
        # Виділяємо область зеленого прямоугольника
        x1, y1 = ZONE_OF_VISION_X1_Y1
        x2, y2 = ZONE_OF_VISION_X2_Y2
        roi = image[y1:y2, x1:x2]

        # Перетворимо область на колірний простір HSV
        roi_rgb = cv2.cvtColor(roi, cv2.COLOR_BGR2RGB)
        # Перетворимо зображення у двовимірний масив пікселів
        pixels = roi_rgb.reshape((-1, 3))
```

```
        # Застосовуємо K-means кластеризацію для визначення основних кольорів
зображення
        n_clusters = 5  # Обираємо кількість кластерів (основних кольорів)
        kmeans = KMeans(n_clusters=n_clusters)
        kmeans.fit(pixels)

        # Отримуємо кольори центроїдів
        colors = kmeans.cluster_centers_.astype(int)

        # Функція для відображення кольорів
        def get_main_plot_colors(colors2):
            color_rect = np.zeros((50, 300, 3), dtype='uint8')
            start_x = 0
            for color in colors2:
                end_x = start_x + 300 // len(colors2)
                color_rect[:, start_x:end_x] = color
                start_x = end_x
            plt.figure(figsize=(12, 2))
            plt.axis("off")
            plt.imshow(color_rect)
            plt.show()

        # Відображаємо основні кольори зображення
        get_main_plot_colors(colors)
        # Виведення у консоль
        for i, color in enumerate(colors, 1):
            print(f"\rColor {i} in RGB: {color}", end="")
```

ДОДАТОК В

ЛІСТИНГ КОДУ ПРОГРАМИ (ФАЙЛ UNIT_TESTS.PY)

```
import unittest
import cv2
from segmentation_library.library import FastSAMController, Gamepad, ImageProcessor,
ZONE_OF_VISION_X1_Y1, \
    ZONE_OF_VISION_X2_Y2


class TestSegmentation(unittest.TestCase):
    def setUp(self):
        self.fast_sam_controller = FastSAMController('./fastsam/FastSAM-x.pt', 'cpu')
        self.gamepad_controller = Gamepad()
        self.image_processor = ImageProcessor()
        self.cap = cv2.VideoCapture(1)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1024)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 576)

    def _get_frame_and_results(self):
        frame_is_available, frame = self.cap.read()
        everything_results = self.fast_sam_controller.create_model(frame)
        prompt_process = self.fast_sam_controller.fast_sam_prompt_process(frame,
everything_results)
        return frame, everything_results, prompt_process

    def test_create_model(self):
        frame, everything_results, _ = self._get_frame_and_results()
        self.assertIsNotNone(everything_results)

    def test_fast_sam_prompt_process(self):
        _, _, prompt_process = self._get_frame_and_results()
        self.assertIsNotNone(prompt_process)

    def test_create_annotations(self):
        _, _, prompt_process = self._get_frame_and_results()
        ann = self.fast_sam_controller.create_annotations(prompt_process, bboxes=[
            [ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
ZONE_OF_VISION_X2_Y2[1]]])
        self.assertIsNotNone(ann)

    def test_plot_to_result(self):
        _, _, prompt_process = self._get_frame_and_results()
        ann = self.fast_sam_controller.create_annotations(prompt_process, bboxes=[
            [ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
ZONE_OF_VISION_X2_Y2[1]]])
        img = self.fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
        self.assertIsNotNone(img)

    def test_get_percent_of_segmented_object_in_all_screen(self):
        _, _, prompt_process = self._get_frame_and_results()
        ann = self.fast_sam_controller.create_annotations(prompt_process, bboxes=[
            [ZONE_OF_VISION_X1_Y1[0], ZONE_OF_VISION_X1_Y1[1], ZONE_OF_VISION_X2_Y2[0],
ZONE_OF_VISION_X2_Y2[1]]])
        img = self.fast_sam_controller.plot_to_result(prompt_process=prompt_process,
annotations=ann)
        percent = self.image_processor.get_percent_of_segmented_object_in_all_screen(img)
        self.assertGreaterEqual(percent, 0)
        self.assertLessEqual(percent, 100)


if __name__ == '__main__':
    unittest.main()
```