МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія визначення змін в потокових відео методом зонування»

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

Керівник роботи **Петров С.О.**

Студента групи ІН.м-91н **Черкас В. Є.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Черкасу Віталію Євгенійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія визначення змін в потокових відео методом зонування*

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

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для визначення змін в потокових відео.3) Розробка інтелектуальної системи з застосуванням оптимізації методом зонування. 4) Аналіз результатів.*

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

_____

_____

_____

_____

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Аналіз проблеми предметної області, постановка й формування завдань дослідження		
2.	Огляд технологій, що використовуються для визначення змін в потокових відео		
3.	Розробка інтелектуальної системи з застосуванням оптимізації методом зонування		
4.	Аналіз отриманих результатів		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис

РЕФЕРАТ

Записка: 45 стор., 31 рис., 1 додаток, 23 джерела.

Об'єкт дослідження — Системи розпізнавання руху для стрімінгових систем.

Мета роботи — розробка інтелектуальної системи визначення змін для звичайних та потокових відеозаписів, з використанням метода зонування для оптимізації точності знаходження руху, та можливістю для користувача обирати необхідну обмежувальну рамку.

Методи дослідження — алгоритми знаходження руху та інструменти обробки відеоматеріалів бібліотеки OpenCV для мови Python.

Результати — розроблено інтелектуальну систему, яка зчитує відезаписи, оброляє їх, надає змогу користувачу обирати необхідну зону для дослідження на наявність руху, віднаходить періоди руху та записує знайдені дані щодо часу періодів руху в CSV файл. Проведено тестування розробки на реальних данних з камер відеоспотреження м. Суми.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, MOTION DETECTION,
ДЕТЕКТУВАННЯ РУХУ, BOUNDING, ПОТОКОВІ ВІДЕО,
КАМЕРИ ВІДЕОСПОСТЕРЕЖЕННЯ, CCTV.

ЗМІСТ

ЗМІСТ 2
ВСТУП 3
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 5
1.1 Дослідження актуальності проблеми 5
1.2 Аналіз алгоритмів пошуку руху на певних ділянках 8
1.2.1 Класифікації алгоритмів пошуку руху 8
1.2.2 Алгоритми виокремлення ділянок відеозаписів 10
1.3 ПОСТАНОВКА ЗАДАЧІ 13
2 ВИБІР МЕТОДУ РІШЕННЯ 15
2.1 Вибір мови програмування 15
2.2 Вибір фреймворку для детектування руху 17
2.3 Вибір інструментів зонування потоково відео 20
3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ 24
3.1 Формування та обробка вхідних даних 24
3.2 Програмне забезпечення для оптимізації виявлення змін методом зонування 27
3.3 Опис програмної реалізації 29
3.4 Аналіз результатів 34
ВИСНОВКИ 37
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 38
Додаток 41

ВСТУП

Світ стає дедалі безпечнішим та зручнішим. Здебільшого такі позитивні зміни зумовлені чисельними впровадженнями високотехнологічних систем як в специфічних газулях, так для вирішення побутових питань. Дуже багато обладнання, яке працює завдяки складним алгоритмам машинного навчання використовують різні технології для аналізу навколишної ситуації, та завдяки отриманій інформації можуть правильно виконувати свої функції.

Сучасні міста посупово стають все більш безпечними, рівень злочинності зменшується, а завдяки IT технологіям правоохоронні органи можуть швидше реагувати на діяння правопорушників та швидше і якісніше знаходити достовірну інформацію щодо злочинів. Все більше і більше власників бізнесів, директори установ та прості мешканці міст встановлюють камери спостереженная як зовні, так і в середині своєї власності. Такі запобіжні заходи допомагають не тільки захистити цілістність підконтрольного майна, а ще й допомагають у вирішенні суперечностей, які можуть викликати та допомагати віднаходити достовірні факти щодо подій, які відбуваються навколо камеро захищенної властності. Так, багато крадіжок, ДТП, бійок чи інших правопорушень можуть бути справедливо розсуджені, а правопорушники швидко розпізнані завдяки відеозаписам, які можуть бути отримані з камер близьких до події місця.

Кожна камера спостереження знімає та записує багато годин матеріалу, який потрібно не тільки зберігати впродовж певного часового проміжку, а ще й обробляти, щоб отримати потрібну інформацію, щодо подій, які могли статися, як то крадіжка, чи псування майна. Задача пошуку потрібного часового інтервалу є досить ресурсозатратною та не легкою, адже потрібно фізично проаналізувати багато часу відзнятого відеоматеріалу, та віднайти потрібні уривки запису. Якщо взяти до уваги, що кожна камера безперервно записує дані, відеоматеріали здебільшого є статичними, а матеріал збирається з багатьох камер одночасно, то стає очевидним, що пошук необхідної події

стає реальним іспитом через велику кількість необхідної для аналізу інформації. Беручи це до уваги, можна прийти висновку, що задача оптимізації пошуку рухів на відеозаписах є достатньо важливою та актуальною, адже завдяки оптимізації процессів пошуку нестатичних ділянок записів можна зекономити велику кількість часу та ресурсів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

Згідно з дослідженням [1], в світі стрімко збільшується кількість камер відеоспостереження, особливо в густонаселених містах. Найбільше технології стеження впроваджені у містах Китаю, але динаміка росту кількості встановлень нових камер є позитивною і для інших країн та міст.

Місто	Країна	Кількість камер	Населення	Кількість камер на 1000 осіб	Індекс злочинності	Індекс безпеки
1-5 місця						
Чунцин	Китай	2,579,890	15,354,067	168.03	33.18	66.82
Шеньчжень	Китай	1,929,600	12,128,721	159.09	42.91	57.09
Шанхай	Китай	2,985,984	26,317,104	113.46	40.87	59.13
Тяньцзінь	Китай	1,244,160	13,396,402	92.87	29.15	70.85
Цзинань	Китай	540,463	7,321,200	73.82	15.93	84.07
45-50 місця						
Бостон	США	1,552	694,784	2.23	33.74	66.26
Прага	Чехія	2,820	1,298,804	2.17	25.16	74.84
Кардіфф	Велика Британія	1,022	474,187	2.16	35.50	64.50
Київ	Україна	6,200	2,950,819	2.10	47.23	52.77
Рим	Італія	8,300	4,234,019	1.96	52.42	47.58

Рисунок 1.1 –[**Error! Reference source not found.**] Топ міст по кількості встановлених камер відеоспостереження

Згідно з дослідження [2] роль камер спостереження в викритті злочинів з часом зростає, здебільшого завдяки покращенню як якості обладнання, так і точності алгоритмів, які застосовуються для обробки отриманої інформації.

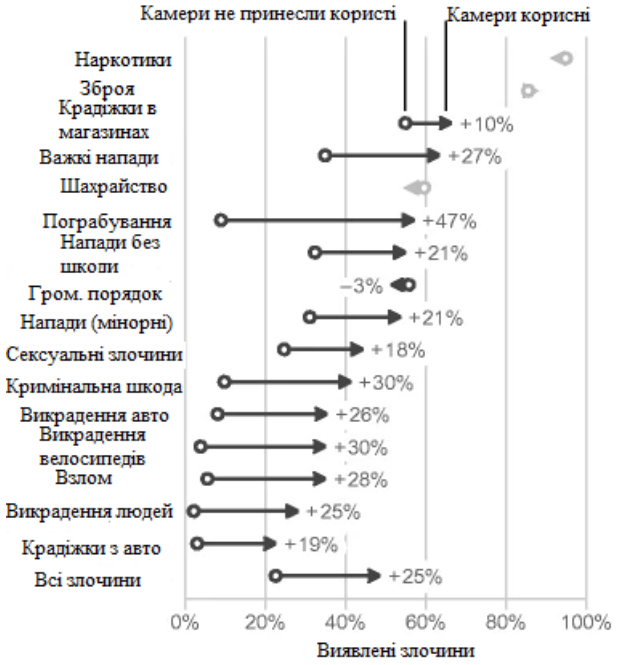

Рисунок 1.2 – [2] Роль камер спостережння в викритті злочинів

Більш того, відсоток виявлених злочинів камерами спостереження також виріс, а кількість цих користно використаних камер наближається до загальної кількості наявних камер.

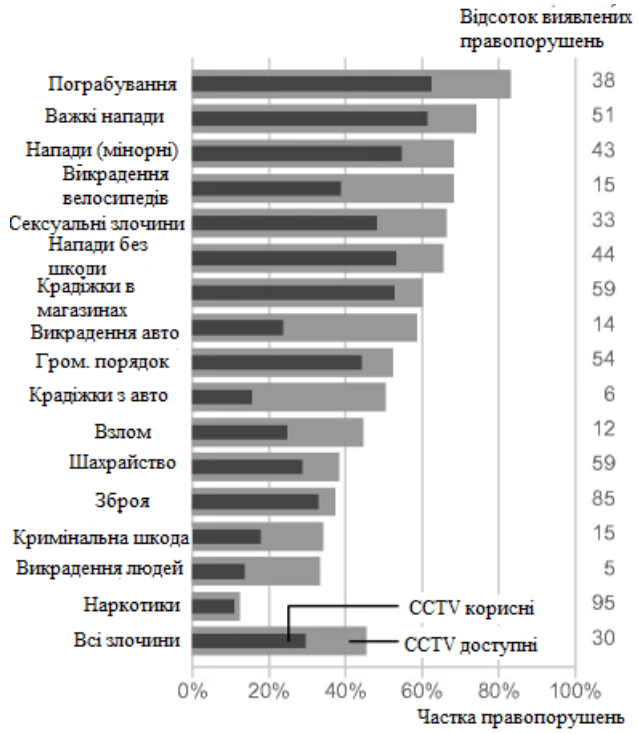

Рисунок 1.3 – [2] Відношення виявлення злочинів камерами та загальною їх наявною кількістю

Згідно зі статтями [3,4], вплив наявності камер спостереження буде і надалі зростати в області виявлення правопорушень та знаходження доказів, але також зазначається, що можливе збільшення складності обробки отриманою інформації з камер, а як наслідок це може призвести до погіршення результативності використання цих технологічних засобів для запобігання та викриття злочинів.

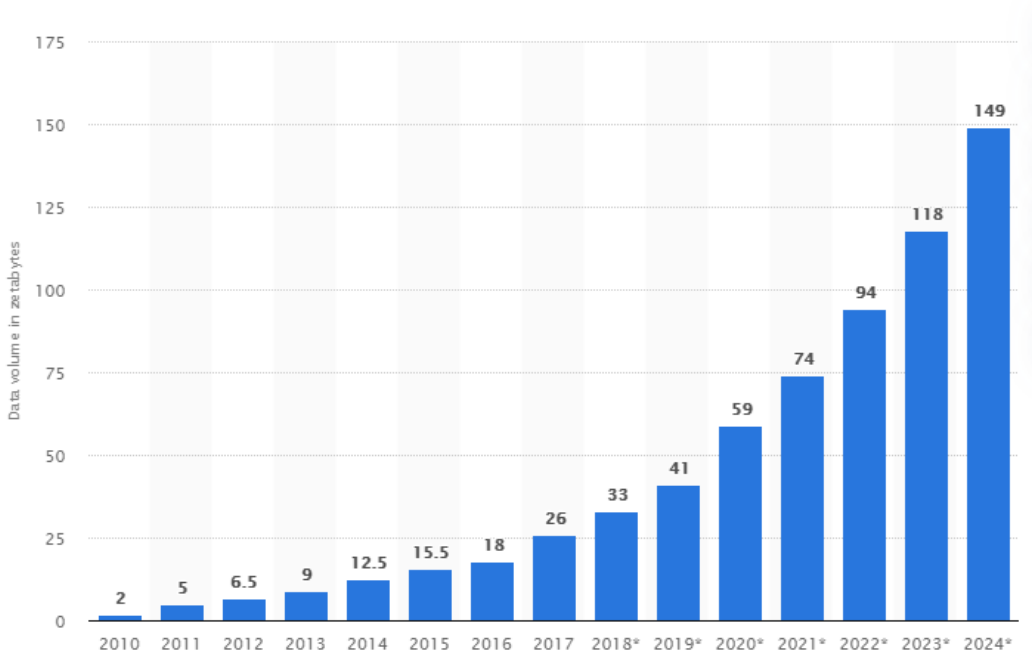

Рисунок 1.4 – Приріст кількості зібранної та створеної інформації в світі

Статистика наведена в дослідженні [5] зазначає, що кількість створеної та зібраної інформації в світі стрімко зростає та збільшує темпи росту. Це означає, що проблеми швидкої обробки великої кількості інформації будуть і надалі мати досить велику актуальність.

1.2 Аналіз алгоритмів пошуку руху на певних ділянках

1.2.1 Класифікації алгоритмів пошуку руху

Для пошуку руху існує декілька загальновідомих алгоритмів. У роботі [6] зазначено, що найбільш результативними та популярними є чотири можливих алгоритма для пошуку руху: Flow Analysis, Temporal Differencing, Background Subtraction, Dynamic Threshold. Результативність цих алгоритмів залежить від декількох факторів:

- Необхідна точність виявлення
- Кількість можливих ресурсів для рішення задачі
- Швидкість дії алгоритму
- Якість відеозапису

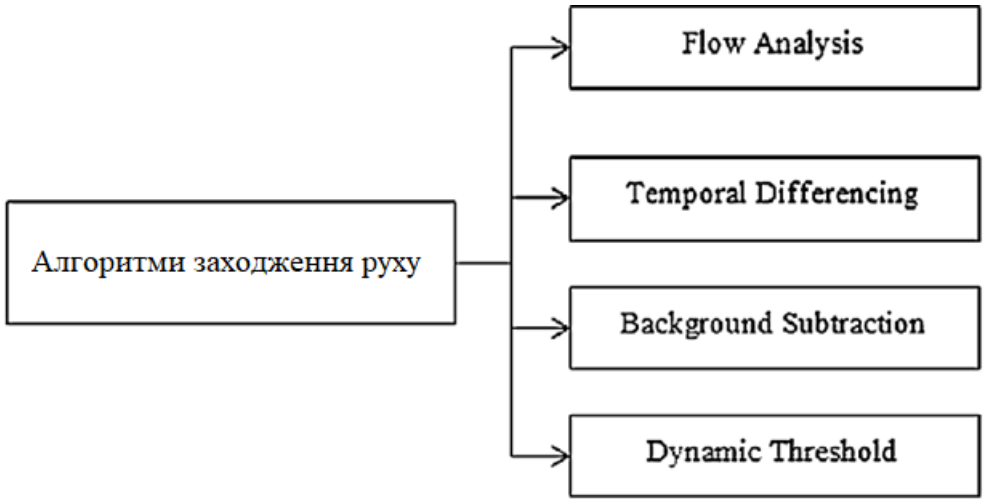

Рисунок 1.5 – [6] Основні алгоритми для пошуку руху на відео

Одним з можливих алгоритмів, що використовують для визначення руху є алгоритм «Locality Sensitive Hashing (LSH)». Як зазначає автор [11],Алгоритм було реалізовано для архівації одноточковою камерою, при якій зображення перетворюються у піксельний файл за допомогою процедури растеризації. Потім пікселі токенізуються та хешуються, використовуючи процедуру міншінгу, яка використовує рандомізований алгоритм для швидкої оцінки

подібності Жакара. LSH виявляє несхожість між кадрами у відео, розбиваючи міншеші на ряд смуг, що складаються з рядків. Запропонована процедура реалізована на декількох наборах даних, і з експериментального аналізу ми зробили висновок, що вона здатна відокремити рухи у відеофайлі.

Одним з найбільш часто застосовуваним алгоритмом для пошуку руху є алгоритм Background Subtraction [7]. Цей алгоритм - загальноприйнята та широко застосовувана техніка генерації маски переднього плану [8] (а саме, двійкового зображення, що містить пікселі, що належать рухомим об'єктам у сцені) за допомогою статичних камер. Як випливає з назви, BS обчислює маску переднього плану, виконуючи віднімання між поточним кадром та фоновою моделлю, що містить статичну частину сцени або, загальніше, все, що можна вважати фоном, враховуючи характеристики спостережуваної сцени. Фонове моделювання складається з двох основних етапів [9]:

- Ініціалізація фону
- Фонове оновлення.

На першому кроці обчислюється початкова модель фону, а на другому кроці ця модель оновлюється з метою адаптації до можливих змін сцени.

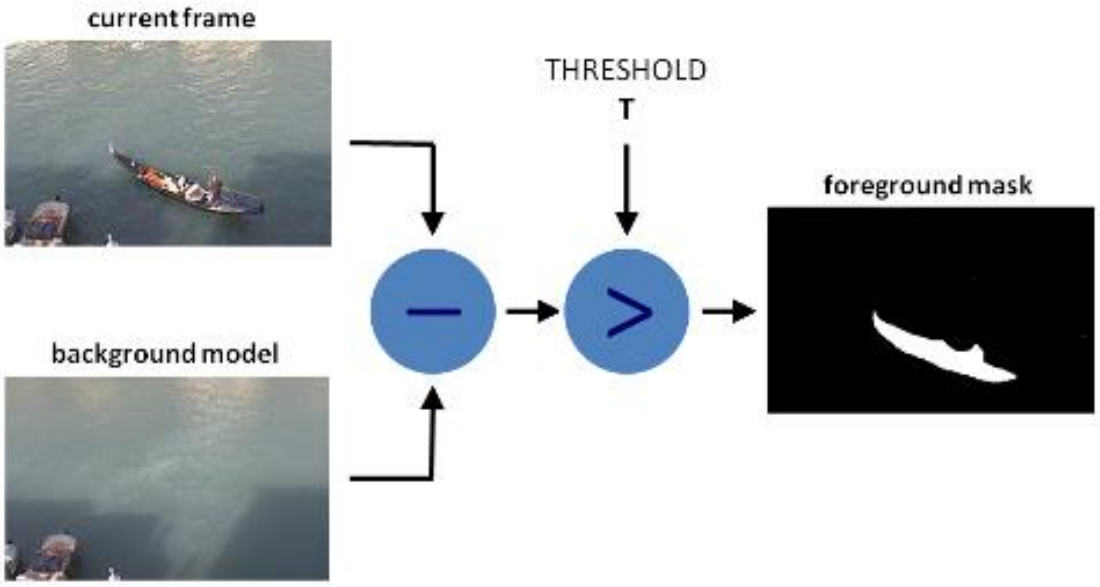

Рисунок 1.6 – Приклад створення маски на основі методу Background Subtraction

Для супроводження об’єктів що рухаються, зазвичай застосовують два методи Meanshift та Camshift [10]. Meanshift метод завичай використовується так: Початкове вікно показано синім колом з назвою "C1". Його початковий центр позначений синім прямокутником, названим "C1_o". "C1_r" – його центроїд (позначену маленьким синім колом), яка є справжнім центроїдом вікна. Потім вікно переміщається так, щоб коло нового вікна збігалося з попереднім центроїдом. Потім шукається новий центроїд. Потім зміщують його ще раз і продовжують ітерації так, щоб центр вікна та його центроїд потрапляли в те саме місце (або в межах невеликої бажаної помилки).

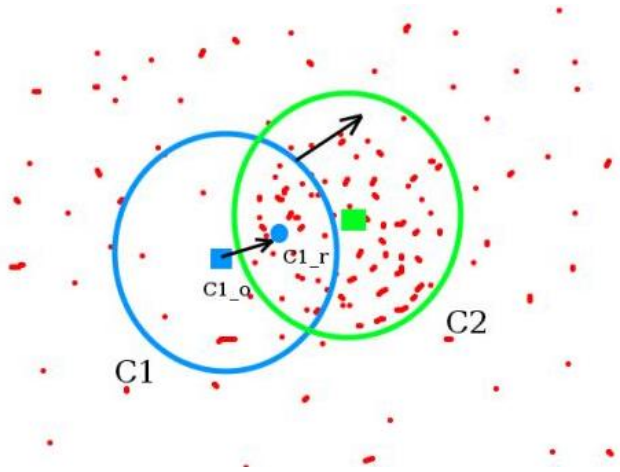

Рисунок 1.7 – Ілюстрація роботи Meanshift методу

1.2.2 Алгоритми виокремлення ділянок відеозаписів

Для вирішення деяких задач пов’язаних з обробкою відеозаписів застосовують різні методи обрізання, масштабування та конкретизації робочої ділянки алгоритму на відео. Такий підхід дозволяє звужувати необхідний діапазон роботи, а значить і зменшувати кількість необхідних ресурсів та часу на обробку. Прикладом вдалого використання такого методу є робота з ретагретингу [18] фокусу камери. В цій роботі пропонується метод перенацілювання відео, який використовує методи обрізання та дволінійного масштабування зображення.

Для впровадження цього методу застосовують як і вбудовані можливості мов програмування, так і окремі бібліотеки. Прикладом реалізацій можуть слугувати бібліотеки та функціонал розроблені на мові Python [12]. На основі цієї мови програмування було розроблено велику кількість бібліотек, які дозволяють обробляти відеозаписи та корегувати їх. Одними з найбільш використовуваних бібліотек є: MoviePy [13], Scikit-image [14] , OpenCV [15].

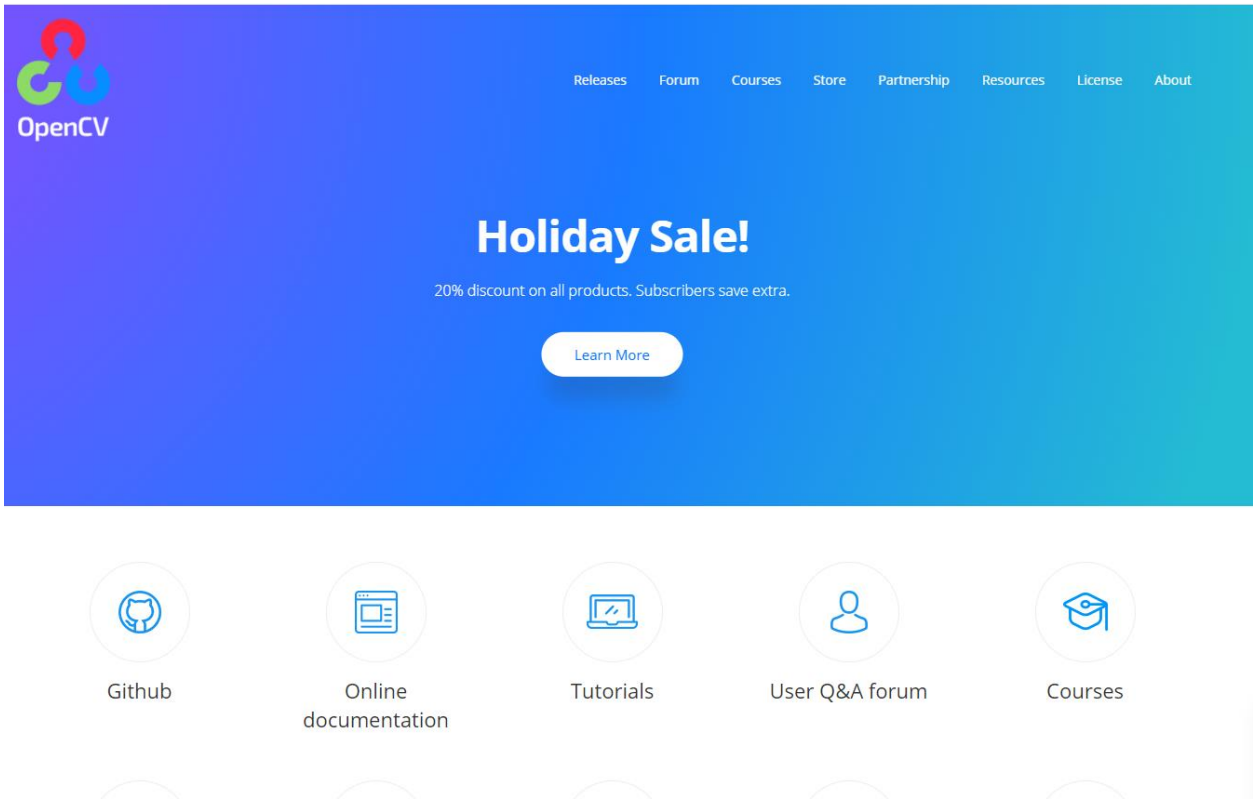

Рисунок 1.8 – Бібліотека OpenCV

На багатьох ресурсах зазначається, що бібліотека OpenCV є дуже часто використовуваною бібліотекою, адже вона включає в себе велику кількість можливого функціоналу. OpenCV - це бібліотека комп'ютерного бачення з відкритим кодом для вилучення та обробки значущих даних із зображень [17]. Ці значущі дані можуть включати знаходження всіх або частин об’єктів, розпізнавання всіх або частин об’єктів, відстеження руху (частин) об’єктів у 2D або 3D між послідовними зображеннями, визначення 2D або 3D форми об’єктів за одним або кількома зображеннями, і пов’язування даних зображень із категоричним значенням. Також ця бібліотека включає в себе можливості

використання алгоритмів комп’ютерного зору[16]. До функціоналу цієї бібліотеки входять такі можливості:

- Геометричні методи: контури, теселяція простору та тріангуляція.
- Вимірювання зображення: статистика зображення, просторові моменти та контурні моменти.
- Службові програми: піраміди зображень, структури даних, зв’язані списки, управління зображеннями, математичні функції, швидкий доступ до пікселів, креслення ліній і конічних розрізів та відображення тексту.
- Сегментація: морфологія зображення, порогове значення, кольорові та фактурні піраміди, віднімання фону, зворотне проектування гістограми, вітербі HMM, K-засоби та нормалізований зріз.
- Виявлення особливостей: трансформація Хафа, детектор краю Canny, виявлення кутів, точна лінія та кут розташування субпікселів, контури, похідні зображення для третього порядку.
- Розпізнавання: відповідність гістограмі, відповідність шаблону, відстань Махаланобіса, HMM, вбудований HMM, дескриптори фігури, власні об’єкти, LDA, 3D-жест
- Відстеження: середній зсув, CAMSHIFT, оптичний потік, афінний потік, шаблони руху, енергетичні змії, фільтри Кальмана та конденсації.
- Камера: калібрування, перетворення виду, 8-точковий алгоритм, листування та підтримка стерео.
- Форма: 2D та 3D лінія та еліпс, а також набір інструментів форми.

1.3 ПОСТАНОВКА ЗАДАЧІ

Метою роботи є розробка програмного забезпечення на основі бібліотек ком'ютерного зору з можливістю виділяти окремі ділянки для аналізу. Дане дослідження виконується в рамках в рамках кваліфікаційної роботи магістра за спеціальністю 122 «Комп'ютерні науки».

Для реалізації поставленої мети необхідно вирішити такі задачі:

- Проаналізувати можливі алгоритми аналізу руху
- Проаналізувати можливі алгоритми для конкретизування ділянки використання алгоритмів руху
- Визначити тестові матеріали для системи
- Налаштувати знайдені алгоритми для умов задачі
- Вибір оптимального алгоритму конкретизування області роботи алгоритму знаходження руху
- Тестування системи на реальних умовах
- Проаналізувати отримані результати на якість розпізнавання, в разі успішної реалізації підготувати систему до впровадження

Для експерементального впровадження системи буде використано алгоритми, які представлені у безкоштовній бібліотеці OpenCV. Порівняльний експеримент будет проводитись для декількох алгоритмів конкретизування області роботи основного алгоритму на основі одного і того самого відео.

Об'єктом дослідження є системи розпізнавання руху для стрімінгових систем.

Наукова новизна полягає в тому, що на відміну від простого застосування алгоритмів знаходження руху на відео, система з можливістю конкретизування ділянки роботи алгоритму може значною мірою покращити результати аналізу. Алгоритм для конкретизування ділянки відео має не тільки звужувати зону дії алгоритмів, а ще й можливість налаштовувати нелінійні ділянки дії та можливість масштабування за допомогою кількістного використання конкретизуючих ділянок. В результатах будуть як і приклад

реалізації системи, так і рекомендації для подальшого отпимального налаштування систем з подібним функціоналом.

Практичною значимістю буде те, що застосування такого підходу для оптимізації алгоритму з пошуку руху може зменшити кількість необхідних ресурсів та часу для знаходження необхідного руху в тій чи іншій області.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір мови програмування

За даними інтернет ресурсу TIOBE [19], найбільш популярними мовами програмування є:

- C
- Java
- Python
- C++
- C#

Apr 2021	Apr 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	14.32%	-2.40%
2	1	▼	Java	11.23%	-5.49%
3	3		Python	11.03%	+1.72%
4	4		C++	7.14%	+0.36%
5	5		C#	4.91%	+0.16%

Рисунок 2.1 – Топ 5 мов програмування за даними TIOBE

Всі представлені мови програмування достатньо часто використовуються для розробки як звичайних програм, так і для розробки алгоритмів штучного інтелекту. Потужності цих мов дозволяють достатньо ефективно розробляти програми з використанням машинного навчання для вирішення різних проблем розпізнавання.

C/C++ - одна з найбільш традиційних мов програмування. C++ ідеально підходить для динамічного балансування навантаження, адаптивного кешування та розробки великих фреймворків великих даних та бібліотек. Більшість бібліотек глибокого навчання були реалізовані за допомогою C++. Scylla, відома своєю наднизькою затримкою та надзвичайно високою пропускною здатністю, кодується за допомогою C++. Завдяки деяким унікальними перевагами C ++ як мови програмування (включаючи управління

пам’яттю, характеристики продуктивності та системне програмування), він безумовно служить одним з найефективніших інструментів для розробки швидко масштабованих бібліотек машинного навчання.

C# - один з найпотужніших, що швидко розвиваються і затребуваних мов в IT-галузі. На даний момент на ньому пишуться найрізноманітніші програми: від невеликих десктопних програмок до великих веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів. Для роботи з машинним навчанням існує спеціальна бібліотка – ML.NET [20]. ML.NET - це безкоштовна платформа машинного навчання з відкритим кодом, створена спеціально для розробників .NET. За допомогою ML.NET можна розробляти та інтегрувати моделі машинного навчання у програми .NET, не потребуючи попереднього досвіду машинного навчання. ML.NET - це розширювана платформа з інструментарієм у Visual Studio, а також міжплатформеним CLI, що забезпечує розпізнані функції Microsoft, такі як Windows Hello, Bing Ads, PowerPoint Design Ideas та багато іншого.

Java - є одним з найбільш поширених і популярних мов програмування. Java - це об'єктно-орієнтована мова програмування на базі класів, яка розроблена так, щоб мати якомога менше залежностей від реалізації. Це мова програмування загального призначення, призначена для того, щоб розробники додатків могли писати один раз, запускати де завгодно. Java застосовують при створенні рішень для машинного навчання, нейронних мереж, алгоритмів пошуку, генетичного програмування і мульти-робототехнічних систем. Такі властивості, як об'єктно-орієнтованість і масштабованість, обов'язкові для ІІ-проектів, а тому Java підходить їм якнайкраще. Оскільки сьогодні ІІ-технології вже активно застосовуються бізнесом, дуже затребувані можливості Java, що дозволяють створити єдину версію програми, яка буде працювати на декількох платформах.

Python - мова програмування загального призначення високого рівня. Філософія дизайну Python підкреслює читабельність коду завдяки помітному

використанню значних відступів. Його мовні конструкції, а також об’єктно-орієнтований підхід мають на меті допомогти програмістам писати чіткий логічний код для малих та великих проектів. Python динамічно набирається та збирається сміття. Він підтримує кілька парадигм програмування, включаючи структуроване (зокрема, процедурне), об'єктно-орієнтоване та функціональне програмування. Існує також безліч модулів та бібліотек на вибір, що забезпечують різні способи виконання кожного завдання. Python відіграє життєво важливу роль у мові кодування штучного інтелекту, надаючи йому хороші фреймворки, такі як scikit-learn: машинне навчання на Python, яке задовольняє майже всі потреби в цій галузі та D3.js - документи, керовані даними в JS, який є найпотужніші та прості у використанні інструменти для візуалізації.

З огляду на поставлену задачу, доцільним буде обрати Python як основну мову програмування для розробки програмного рішення. Великим плюсом буде як проста і потужність мови програмування, так і велика кількість бібліотек у вільному доступі для машинного навчання та математичних обчислень.

2.2 Вибір фреймворку для детектування руху

На мові Python написано велику кількість бібліотек, що значно спрощують вирішення поставлених завдань, спрямованих на використання алгоритмів штучного інтелекту. Для різного роду завдань існує велика кількість різних рішень, тож для проєкту було обрано найбільш популярні і доцільні бібліотеки:

NumPy - це популярний пакет обробки масивів Python [21]. Він забезпечує хорошу підтримку різних розмірних об'єктів масиву, а також матриць. Numpy не тільки обмежується лише наданням масивів, але він також надає різноманітні інструменти для управління цими масивами. Дві життєво важливі переваги, які може запропонувати NumPy, - це підтримка потужних N-мірних

об'єктів масиву та вбудованих інструментів для проведення інтенсивних математичних та наукових обчислень.

Інші вражаючі особливості NumPy включають використання оптимізованого ядра C для забезпечення високої продуктивності, взаємодії з різними обчислювальними платформами та обладнанням та простоти використання.

TensorFlow – популярна обчислювальна бібліотека для створення моделей машинного навчання. [22] TensorFlow підтримує різноманітні набори інструментів для побудови моделей на різних рівнях абстракції.

- Основна бібліотека підходить для широкого сімейства технік машинного навчання, а не тільки для глибинного навчання.
- Лінійна алгебра та інші нутрощі добре видно зовні.
- На додаток до основної функціональності машинного навчання, TensorFlow також включає власну систему логування, власний інтерактивний визуализатор логів і навіть потужну архітектуру з доставки даних.
- Модель виконання TensorFlow відрізняється від scikit-learn мови Python і від більшості інструментів в R.

Pandas - бібліотека для вирішення практичного аналізу даних у реальному світі на Python [23]. Ця бібліотека, забезпечує швидкі, гнучкі та виразні структури даних, покликані зробити роботу зі структурованими (табличними, багатовимірними, потенційно неоднорідними) та даними часових рядів одночасно легкими та інтуїтивно зрозумілими. Ця бібліотека має такі важливі функції як:

- Інструменти для читання та запису даних між структурами даних у пам’яті та різними форматами файлів.
- Вирівнювання даних та інтегрована обробка відсутніх даних.
- Переформатування та обертання наборів даних.

- Нарізка на основі міток, вигадане індексування та підмножина великих наборів даних.
- Вставка та видалення стовпців структур даних.
- Злиття та приєднання масивів даних.

OpenCV - open source бібліотека комп'ютерного зору, яка призначена для аналізу, класифікації та обробки зображень [15]. Широко використовується в таких мовах як C, C ++, Python і Java. Області застосування OpenCV включають:

- Набори інструментів для обробки 2D та 3D зображень.
- Системи розпізнавання обличчя.
- Розпізнавання жестів
- Взаємодія людина-комп'ютер (HCI).
- Виявлення об’єктів.
- Сегментація та визнання.
- Відстеження руху.
- Доповнена реальність.

OpenCV написаний на C ++, а його основний інтерфейс - на C ++, але він все ще зберігає менш вичерпний, хоча розширений старший інтерфейс C. Усі нові розробки та алгоритми з'являються в інтерфейсі C ++. Обгортки кількома мовами програмування були розроблені для заохочення прийняття ширшою аудиторією. OpenCV працює на наступних операційних системах:

- Windows
- Linux
- MacOS
- FreeBSD
- NetBSD
- OpenBSD

Для досягнення цілей та розв’язання різних задач, у бібліотеці використовуються такі моделі машинного навчання як:

- Підсилення.
- Навчання дереву рішень.
- Градієнтний підйом дерев.
- Алгоритм очікування-максимізації.
- k-алгоритм найближчого сусіда.
- Наївний класифікатор Байєса.
- Штучні нейронні мережі.
- Випадковий ліс.
- Підтримка векторної машини (SVM).
- Глибокі нейронні мережі.

Для поставленної задачі цих обраних бібліотек буде достатньо, щоб швидко та ефективно розробити алгоритм, що буде виконувати потрібні функції, без використання зайвих ресурсів. Для математичних операцій здебільшого буде використовуватись бібліотека Pandas. Для детектування рухів та обробку потокового відео буде використовуватися OpenCV, так як вона є безкоштовною, та має достатньо можливостей для реалізації поставленої задачі.

2.3 Вибір інструментів зонування потоково відео

Для виконання поставлених завдань, необхідно обрати такі інструменти для розробки:

- Потокове завантаження та обробка відео.
- Детектування руху.
- Зонування відео.

Для завантаження та обробки відео, буде використовуватися функціонал, який вже є в OpenCV. Усі функціональні можливості, необхідні для маніпулювання відео, інтегровані в клас cv :: VideoCapture C ++. Це само по собі базується на бібліотеці з відкритим кодом FFmpeg. Це основна залежність OpenCV. Відео складається з послідовності зображень – кадрів. У випадку з відеофайлом частота кадрів визначає, скільки часу між двома кадрами. Хоча для відеокамер зазвичай існує обмеження кількості кадрів, які вони можуть оцифрувати в секунду, ця властивість менш важлива, оскільки в будь-який момент камера бачить поточний знімок світу.

Для детектування руху буде використано Методи фонового віднімання та Оптичний потік (Метод Лукаса-Канаде). Віднімання фону в основному базується на статичній фоновій гіпотезі, яка часто не застосовується в реальному середовищі. У сценах у приміщенні відображення або анімовані зображення на екранах призводять до змін фону. Подібним чином, через зміни вітру, дощу чи освітлення, спричинені погодою, методи статичного фону мають труднощі із сценами на вулиці. Оптичний потік - це картина очевидного руху об’єктів зображення між двома послідовними кадрами, спричинена рухом об’єкта або камери. Це двовимірне векторне поле, де кожен вектор є вектором переміщення, що показує рух точок від першого кадру до другого.

Метод Лукаса-Канаде приймає пластиру розміром 3x3 навколо точки. Отже, усі 9 точок мають однаковий рух. можемо знайти (fx, fy, ft) для цих 9 балів. Отже, тепер наша задача стає розв’язуванням 9 рівнянь з двома невідомими змінними, що надмірно визначається. Краще рішення отримано методом найменшої квадратної посадки. Нижче наведено остаточне рішення, яке є двома рівняннями-двома невідомими завданнями і вирішуються для отримання рішення.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}{}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}{}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

Рис. 2.2 Метод Лукаса-Канаде

Отже, з точки зору користувача, ідея проста, дається кілька точок для відстеження, отримується оптичні вектори потоку цих точок. Отже, застосовуючи метод Лукас-Канаде, отримуємо оптичний потік разом із шкалою. OpenCV забезпечує все це в одній функції, cv.calcOpticalFlowPyrLK (). Беремо перший кадр, виявляємо в ньому кілька кутових точок Ши-Томасі, потім ітеративно відстежуємо ці точки, використовуючи оптичний потік Лукаса-Канаде. Для функції cv.calcOpticalFlowPyrLK()передаємо попередній кадр, попередні точки та наступний кадр. Він повертає наступні точки разом із деякими номерами стану, що має значення 1, якщо знайдено наступну точку, інакше нуль. Повторно передаємо ці наступні пункти як попередні пункти на наступному кроці. Метод Лукаса-Канаде обчислює оптичний потік для розрідженого набору функцій.

Також бібліотека OpenCV має в собі такі функції для модифікації зображень:

- Пошук контурів на зображенні.
- Створення обмежувальних коробок і кіл для контурів.
- Створення обмежувальних поворотних коробок та еліпсів для контурів.
- Нефокусований фільтр розмиття.
- Фільтр розмиття руху.
- Анізотропна сегментація зображення тензором градієнтної структури.
- Фільтр для періодичного видалення шуму.
- Сегментація зображень за допомогою перетворення відстані та водозбірного алгоритму.

Завдяки цим вбудованим функціям, буде достатньо легко розробити програмне забезпечення, яке буде вирішувати зазначені задачі. До того ж

бібліотека OpenCV є досить гарно відточеною, та часто використовується для проектів різного плану, а це означає, що обрані функції будуть виконуватись належним чином, що дозволить отримувати точні результати роботи програмного забезпечення.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

3.1 Формування та обробка вхідних даних

Для того, щоб алгоритм зміг обробити надані відеоматеріали, програмне забезпечення мусить мати змогу завантажувати як вже збережені дані, так і дані, отримані з потокових зчитувачів. Для того, щоб завантажити відео, необхідно застосувати метод *VideoCapture* з бібліотеки OpenCV. Його аргументом може бути як індекс пристрою, так і ім'я відеофайлу. Індекс пристрою - це лише число, яке вказує, яку камеру. Зазвичай підключається одна камера. Для того, щоб завантажувати вже збережені відеозаписи – необхідно як аргумент подати ім’я файлу, з якого буде відбуватися зчитування. Після того, як цей метод отримує відеоматеріали, за допомогою засобів бібліотеки можна почати обробку даних покадрово.

```
parser = argparse.ArgumentParser()
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()
```

Рис. 3.1 Приклад завантаження відео для подальшої обробки

У випаду, коли необхідно завантажити відеозапис з прямого етеру, наприклад з стріму на YouTube, необхідно звернутися до додаткової бібліотеки – Pafy.

```
import pafy
url = "https://www.youtube.com/watch?v=0-MQC_G9jTU"
videoPafy = pafy.new(url)
best = videoPafy.getbest(preftype="mp4")
video = cv2.VideoCapture(best)
```

Рис. 3.2 Приклад завантаження стріму з YouTube

За допомогою об’єкта *videoPafy*, можно отримати зображення напряму з YouTube, для цього необхідно вказати посилання на потоковий відеозапис,

потім застосувати метод *getbest*, результатом якого будуть дані, які можно надати в якості аргумента методу VideoCapture.

Для подальшої обробки відеозапису необхідно створити цикл, який буде покадрово обробляти дані, які надходять з завантаженного відео.

```
# Створення циклу для обробки відео покадрово
while True:
    # Зчитування кадру відео
    check, frame = video.read()
    if(frame is None):
        break
    # Ініціювання детектору руху
    motion = 0
```

Рис. 3.3 Приклад обробки відеозапису покадрово в циклі

Методом *read* можна отримувати окремі кадри, які представленні у вигляді об’єктів-картинок, та застосовувати необхідні маніпуляції до них, отримуючи необхідні результати.

Для того, щоб отримувати дані про початок та кінець періоду знаходення руху, необхідно отримувати поточний час та записувати його до файлу-таблиці. Для отримання часу було застосовано методи, доступні в бібліотеці *datetime*. Методом *datetime.now()* можна отримати об’єкт з даними про точний поточний час.

```
# Запис часу початку знайденого руху
if motion_list[-1] == 1 and motion_list[-2] == 0:
    time.append(datetime.now())

# Запис часу завершення знайденого руху
if motion_list[-1] == 0 and motion_list[-2] == 1:
    time.append(datetime.now())
```

Рис. 3.4 Приклад запису поточного часу знайденого руху

Після того, як час було зафіксовано та додано до відповідного елементу масиву з даними про початок і кінець знайденого руху, необхідно здійснити запис отриманих даних до відповідної таблиці.

```
# Запис отриманих даних часу в DataFrame
for i in range(0, len(time), 2):
    df = df.append({"Початок": time[i], "Кінець": time[i + 1]}, ignore_index=True)

# Створення CSV файлу в який буде збережено отримані результати
df.to_csv("Результати.csv")
```

Рис. 3.5 Приклад запису отриманної інформації до CSV файлу

Після того, як інтелектуальна система завершить свою роботу, користувач матиме змогу побачити та обробити отримані дані про віднайдені періоди руху на відеозаписі.

	Початок	Кінець
1	18:57.3	19:13.6
2	19:13.6	19:14.0
3	19:14.5	19:19.0

Рис. 3.6 Приклад отриманих результатів щодо руху на відеозаписі

3.2 Програмне забезпечення для оптимізації виявлення змін методом зонування

Для того, щоб оптимізувати алгоритм пошуку руху методом зонування, необхідно створити два алгоритми: отримання відокремленної зони на відеоряді та звуження роботи базового алгоритму пошуку руху до розмірів обраної зони.

Для отримання користувачем обмежуючої рамки, було застосовано метод з бібліотеки OpenCV – *selectROI()*. Цей метод озволяє користувачам вибрати обмежувальну рамку для даного зображення. Функція створює вікно і дозволяє користувачам вибрати рамку за допомогою миші. Цей метод має такі параметри:

- *windowName* - ім'я вікна, де буде показано процес відбору.
- *img* – об’єкт для вибору рамки.
- *showCrosshair* - чи буде показано справжнє перехрестя прямокутника виділення.
- *fromCenter* - чи справжній центр виділення буде відповідати початковому положенню миші. У протилежному випадку кут прямокутника виділення буде відповідати початковому положенню миші.

Завдяки методу *selectROI()*, можна отримати дані вибрані користувачем та застосувати їх для алгоритму звуження дії базового алгоритму.

```
    r = cv2.selectROI(frame, fromCenter, showCrosshair)
    height, width, layers = frame.shape

frame = frame[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2])]
frame = resize_image(frame, width, height)
```

Рис. 3.7 Приклад отримання звужувальної рамки та виклик методу перерахунку зони дії базового алгоритму

Після того, як користувач вказав в якій саме зоні необхідно почати роботу алгоритма пошуку руху, необхідно застосувати метод для перерахунку обмежувальної рамки роботи дії базового алгоритму пошуку руху. За допомогою отриманих даних, необхідно провести розрахунки нових розмірів рамки та впровадити зміни до відповідного поточного зображення-кадру. Для того, щоб легко і правильно впровадити зміни розмірів, необхідно застосувати метод *resize()* з бібліотеки OpenCV. Зміна розміру зображення означає зміну його розмірів, будь то по ширині, лише по висоті або зміні обох. Крім того, співвідношення сторін оригінального зображення може бути збережене на зображенні зі зміненим розміром. Щоб змінити розмір зображення, OpenCV надає функцію *cv2.resize()*. Після того, як розмір зображення вираховано, необхідно зробити нові обмежувальні рамки, за домогою методу *copyMakeBorder()*.

```
def resize_image(image, width, height,COLOUR=[0,0,0]):
    h, w, layers = image.shape
    if h > height:
        ratio = height/h
        image = cv2.resize(image,(int(image.shape[1]*ratio),int(image.shape[0]*ratio)))
    h, w, layers = image.shape
    if w > width:
        ratio = width/w
        image = cv2.resize(image,(int(image.shape[1]*ratio),int(image.shape[0]*ratio)))
    h, w, layers = image.shape
    if h < height and w < width:
        hless = height/h
        wless = width/w
        if(hless < wless):
            image = cv2.resize(image, (int(image.shape[1] * hless), int(image.shape[0] * hless)))
        else:
            image = cv2.resize(image, (int(image.shape[1] * wless), int(image.shape[0] * wless)))
    h, w, layers = image.shape
    if h < height:
        df = height - h
        df /= 2
        image = cv2.copyMakeBorder(image, int(df), int(df), 0, 0, cv2.BORDER_CONSTANT, value=COLOUR)
    if w < width:
        df = width - w
        df /= 2
        image = cv2.copyMakeBorder(image, 0, 0, int(df), int(df), cv2.BORDER_CONSTANT, value=COLOUR)
    image = cv2.resize(image,(1280,720),interpolation=cv2.INTER_AREA)
    return image
```

Рис. 3.7 Приклад реалізації методу перерахунку нової обмежувальної рамки зображення

Цей метод утворює обмежувальну рамку навколо зображення. Функція копіює вихідне зображення в середину цільового. Області ліворуч, праворуч, над і під скопійованим вихідним зображенням будуть заповнені екстрапольованими пікселями. Це не те, що роблять функції фільтрації на його основі (вони екстраполюють пікселі на льоту), а те, що інші більш складні функції, включаючи вашу власну, можуть зробити для спрощення обробки меж зображення. Функція підтримує режим, коли вхідне зображжженя вже знаходиться посередині цільового. У цьому випадку функція не копіює саме вхідне зображення, а просто створює межу.

Завдяки комбінуванню цих двох методів: отримання розмірів обмежувальної рамки та створення нової, базовий алгоритм отримує покращення роботоспроможності. Результати дії цих алгоритмів буде зазначено в розділі *3.4 Аналіз результатів*.

3.3 Опис програмної реалізації

Для роботи інтелетектуальної системи, спочатку підлючаються необхідні бібліотеки, зазначені на Рис. 3.8

```
# Підключення необхідних бібліотек
import cv2, time, pandas
import argparse
import pafy
from datetime import datetime
```

Рис. 3.8 Підключення необхідних бібліотек

Після успішного підключення бібліотек, відбувається завантаження відеозапису для обробки.

```
url = "https://www.youtube.com/watch?v=0-MQC_G9jTU"
videoPafy = pafy.new(url)
best = videoPafy.getbest(preftype="mp4")
video = cv2.VideoCapture(best)
```

Рис. 3.9 Завантаження відеозапису з YouTube

Для завантаження можна використовувати як завантаження з YouTube, як це показано на Рис. 3.9, або можно завантажити завчасно збережене відео, як це показано на Рис. 3.10.

```
# Завантаження відеозапису
parser = argparse.ArgumentParser()
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()
video = cv2.VideoCapture(args.image)
```

Рис. 3.10 Завантаження завчасно збереженого відеозапису

На Рис. 3.11 зображено реалізацію методу для створення обмежувальної рамки, за зазначиними розмірами.

```
def resize_image(image, width, height,COLOUR=[0,0,0]):
    h, w, layers = image.shape
    if h > height:
        ratio = height/h
        image = cv2.resize(image,(int(image.shape[1]*ratio),int(image.shape[0]*ratio)))
    h, w, layers = image.shape
    if w > width:
        ratio = width/w
        image = cv2.resize(image,(int(image.shape[1]*ratio),int(image.shape[0]*ratio)))
    h, w, layers = image.shape
    if h < height and w < width:
        hless = height/h
        wless = width/w
        if(hless < wless):
            image = cv2.resize(image, (int(image.shape[1] * hless), int(image.shape[0] * hless)))
        else:
            image = cv2.resize(image, (int(image.shape[1] * wless), int(image.shape[0] * wless)))
    h, w, layers = image.shape
    if h < height:
        df = height - h
        df /= 2
        image = cv2.copyMakeBorder(image, int(df), int(df), 0, 0, cv2.BORDER_CONSTANT, value=COLOUR)
    if w < width:
        df = width - w
        df /= 2
        image = cv2.copyMakeBorder(image, 0, 0, int(df), int(df), cv2.BORDER_CONSTANT, value=COLOUR)
    image = cv2.resize(image,(1280,720),interpolation=cv2.INTER_AREA)
    return image
```

Рис. 3.11 Алгоритм створення обмежувальної рамки

Початок виконання обробки відеозапису відбувається разом з початком виконання циклу, в якому кожний кадр відео зберігається в змінну *frame* та потім обробляється окремо.

```
# Створення циклу для обробки відео покадрово
while True:
    # Зчитування кадру відео
    check, frame = video.read()
    if(frame is None):
        break
```

Рис. 3.12 Початок обробки відеозапису покадрово

На Рис. 3.13 зображено код алгоритму для надання отриманих від користувача даних щодо обмежувальної рамки до методу *resize_image*, в якому за зазначиними координатами формується нова рамка.

```
# Ініціювання детектору руху
motion = 0
if borderSet is False:
    showCrosshair = False
    fromCenter = False
    borderSet = True
    r = cv2.selectROI(frame, fromCenter, showCrosshair)
    height, width, layers = frame.shape

frame = frame[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2])]
frame = resize_image(frame, width, height)
```

Рис. 3.13 Надання отриманих даних для створення обмежувальної рамки

Після того, як нова обмежувальна рамка була сформована, в обраній області починає діяти алгоритм знаходження руху, код якого зазначено на Рис. 3.14.

```
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (21, 21), 0)
    if static_back is None:
        static_back = gray
        continue
    diff_frame = cv2.absdiff(static_back, gray)

    thresh_frame = cv2.threshold(diff_frame, 30, 255, cv2.THRESH_BINARY)[1]
    thresh_frame = cv2.dilate(thresh_frame, None, iterations=2)
    cnts, _ = cv2.findContours(thresh_frame.copy(),
                               cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    asd = (r[1] + r[3]) * (r[0] + r[2])
    for contour in cnts:
        if cv2.contourArea(contour) < (r[1] + r[3]) * (r[0] + r[2])/1000:
            continue
        motion = 1

        (x, y, w, h) = cv2.boundingRect(contour)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

    motion_list.append(motion)
```

Рис. 3.14 Код алгоритму знаходження руху

Кожний раз, коли алгоритм знаходить рух на відеозаписі, робиться збереження цієї події до спеціального масиву, де зберігаються дані про початок і про кінець віднайденого періоду руху.

```
# Запис часу початку знайденого руху
if motion_list[-1] == 1 and motion_list[-2] == 0:
    time.append(datetime.now())

# Запис часу завершення знайденого руху
if motion_list[-1] == 0 and motion_list[-2] == 1:
    time.append(datetime.now())

# Відображення результату
cv2.imshow("Color Frame", frame)

key = cv2.waitKey(1)

if key == ord('q'):
    if motion == 1:
        time.append(datetime.now())
    break
```

Рис. 3.15 Код запису та відображення знайденого руху

На прикінці роботи алгоритму, отримані дані щодо знайдених періодів руху на відеозаписах, записуються в CSV файл, маючи дані про початок і кінець ких періодів.

```
# Запис отриманих даних часу в DataFrame
for i in range(0, len(time), 2):
    df = df.append({"Початок": time[i], "Кінець": time[i + 1]}, ignore_index=True)

# Створення CSV файлу в який буде збережено отримані результати
df.to_csv("Результати.csv")
```

Рис. 3.16 Збереження отриманих результатів

3.4 Аналіз результатів

На початку роботи алгоритма, користувачу надається можливість обрати необхідну обмежувальну рамку самостійно, як це зображено на Рис. 3.17.

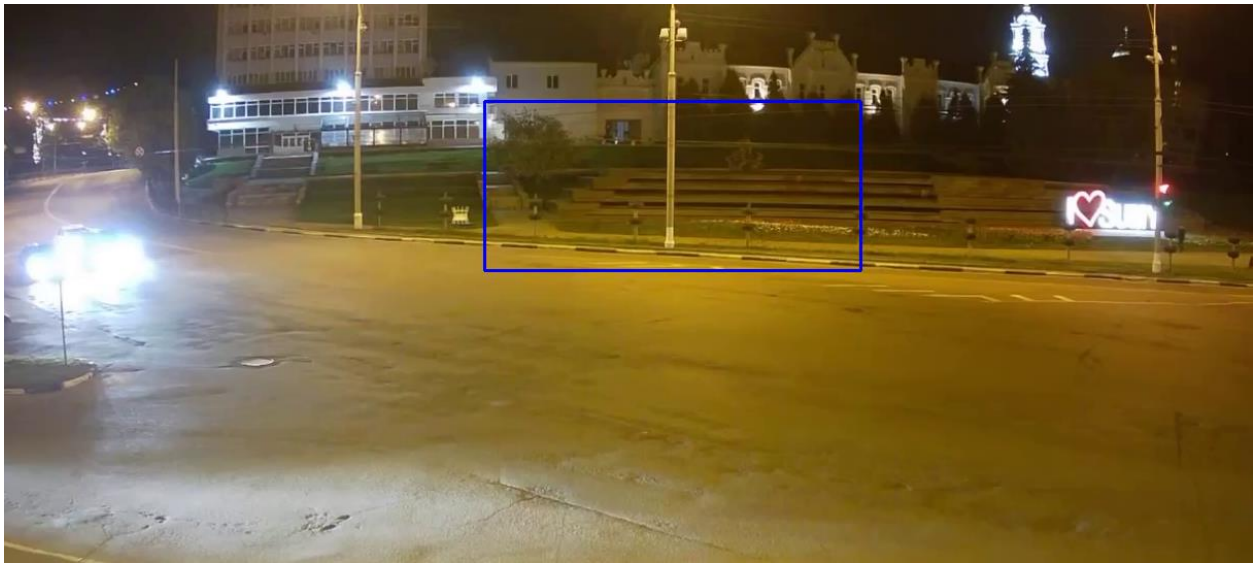

Рис. 3.17 Приклад обраної обмежувальної рамки

Після того, як користувач обрав обмежувальну рамку, за допомогою алгоритму, зазначеному на Рис. 3.14, створюється нова рамка. На новій зоні роботи, алгоритм знаходження руху оптімізує свою точність, відповідно до нових розмірів та починає виконання.

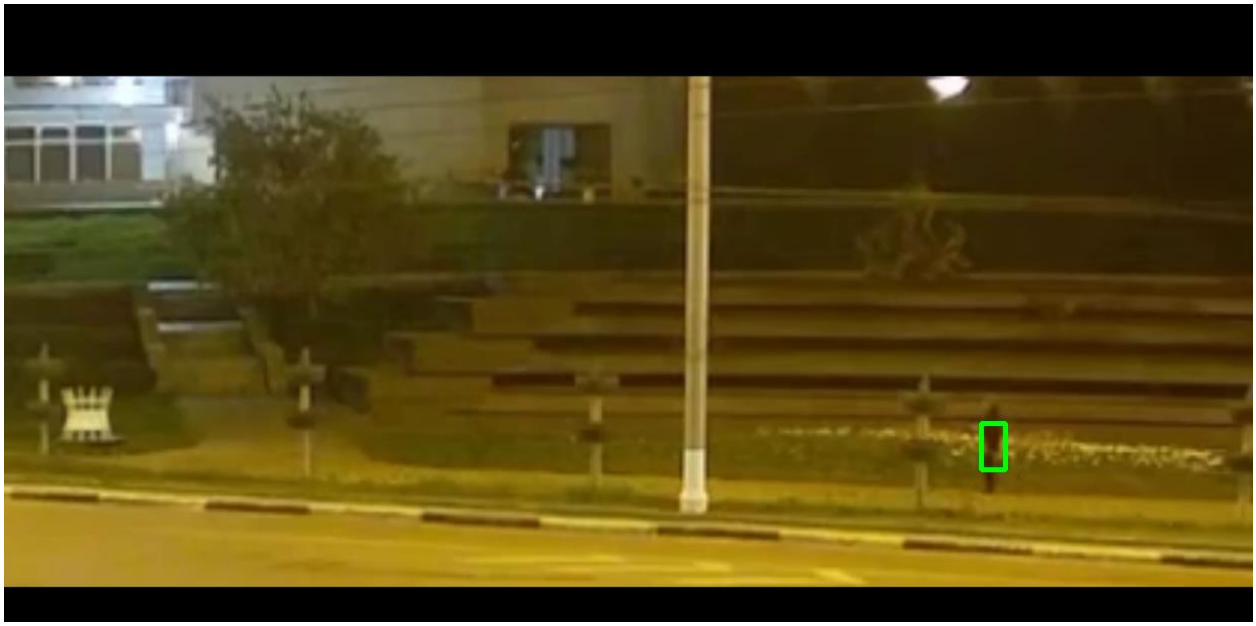

Рис. 3.18 Відеозапис після створення нової обмежувальної рамки

Рис. 3.19 Приклад роботи алгоритму без обмежувальної рамки

За результатами, отриманими після виконання алгоритму знаходження руху, без обмежувальної рамки, було отримано 2 періоди руху за 1 хвилину часу – було виявлено декілька машин, що проїзжали повз камеру, але не було виялено жодного пішохода, що проходили повз, адже їх фігури були замалі для того, щоб алгоритм міг їх розпізнати як об’єкти що рухаються.

	Початок	Кінець
1	28:02.0	28:15.0
2	28:04.0	28:18.0

Рис. 3.20 Дані, отримані алгоритмом без обмежувальної рамки

Після застосування алгоритму з обмежувальною рамкою, було отримано 3 періоди руху за ту саму 1 хвилину спостереження – дві машини та пішохід, що проходив повз.

	Початок	Кінець
1	27:55.0	29:23.0
2	28:02.0	28:15.0
3	28:04.0	28:18.0

Рис. 3.20 Дані, отримані алгоритмом з обмежувальною рамкою

За допомогою використання створенного алгоритму, було надано користувачам змогу самим обирати де камера має зосередити своє спостереження, не вдаючись до механічного впливу. Тестовий запуск алгоримту показав, що цей алгоритм призводить до значного покращення точності знаходження руху менших об’єктів, не втрачаючи можливість знаходити рухи великих об’єктів.

ВИСНОВКИ

В результаті використання роботи було створено інтелектуальну систему, яка покращує роботу детектору руху, завдяки впровадженню алгоритму стоворення додаткової обмежувальної рамки зони дії алгоритму. Було оглянуто різні можливі варіанти розробки зазначеного алгоритму, та обрано оптимальні необхідні інструменти для виконання поставленої задачі. В ході виконання кваліфікаційної магістерської роботи було виконано такі завдання:

1) Проаналізовано можливі алгоритми знаходження руху та алгоритми створення обмежувальної рамки для зони дії алгоритму.
2) Визначено тестові матерали – відеозапис з прямого етеру з публічних камер міста Суми.
3) Обрані алгоритми налаштовано для поставлених умов задачі.
4) Реалізовано роботу інтелектуальної системи, яка може розпізнавати рух на окремо вибраних ділянках, та записувати знайдені періоди часу до файлу.
5) Протестовано систему на реальних даних.
6) Проаналізовано отримані результати та зроблено висновки щодо якості роботи алгоритму.

За результатами роботи інтелектуальної системи, можна побачити, що використання алгоритму обмеження зони роботи дає позитивні результати на точність знаходження руху, а значит мета, поставлена для цієї квалафікаційної роботи була досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Surveillance camera statistics: which cities have the most CCTV cameras? – Paul Bischoff [Електронний ресурс] – Режим доступу до ресурсу: https://www.comparitech.com/vpn-privacy/the-worlds-most-surveilled-cities/

2. Ashby, M.P.J. The Value of CCTV Surveillance Cameras as an Investigative Tool: An Empirical Analysis. Eur J Crim Policy Res 23, 441–459 (2017). https://doi.org/10.1007/s10610-017-9341-6

3. Assessing the Impact of CCTV - Martin Gill, Angela Spriggs (2005) [Електронний ресурс] – Режим доступу до ресурсу: https://techfak.uni-bielefeld.de/~iluetkeb/2006/surveillance/paper/social_effect/CCTV_report.pdf

4. The Influence of the Presentation of Camera Surveillance on Cheating and Pro-Social Behavior - Jansen Anja M., Giebels Ellen, van Rompay Thomas J. L., Junger Marianne (2018) [Електронний ресурс] – Режим доступу до ресурсу: https://www.frontiersin.org/articles/10.3389/fpsyg.2018.01937/full

5. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2024 [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/871513/worldwide-data-created/#:~:text=The%20total%20amount%20of%20data,ever%2Dgrowing%20global%20data%20sphere.

6. A novel framework for intelligent surveillance system based on abnormal human activity detection in academic environments. Neural Computing and Applications. - Al-Nawashi, Malek & Al-hazaimeh, Obaida & Saraee, Mohamad. (2017) [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/303771836_A_novel_framework_for_intelligent_surveillance_system_based_on_abnormal_human_activity_detection_in_academic_environments

7. Paul, M., Haque, S.M.E. & Chakraborty, S. Human detection in surveillance videos and its applications - a review. EURASIP J. Adv. Signal Process. 2013, 176 (2013). https://doi.org/10.1186/1687-6180-2013-176

8. Antoine Vacavant, Thierry Chateau, Alexis Wilhelm, and Laurent Lequièvre. A benchmark dataset for outdoor foreground/background extraction. In Computer Vision-ACCV 2012 Workshops, pages 291–300. Springer, 2013.

9. How to Use Background Subtraction Methods [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/4.5.0/d1/dc5/tutorial_background_subtraction.html

10. Bradski, G.R., "Real time face and object tracking as a component of a perceptual user interface," Applications of Computer Vision, 1998. WACV '98. Proceedings., Fourth IEEE Workshop on , vol., no., pp.214,219, 19-21 Oct 1998

11. Srenithi M., Kumar P. (2019) Motion Detection Algorithm for Surveillance Videos. In: Pandian D., Fernando X., Baig Z., Shi F. (eds) Proceedings of the International Conference on ISMAC in Computational Vision and Bio-Engineering 2018 (ISMAC-CVB). ISMAC 2018. Lecture Notes in Computational Vision and Biomechanics, vol 30. Springer, Cham. https://doi.org/10.1007/978-3-030-00665-5_92

12. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/

13. MoviePy [Електронний ресурс] – Режим доступу до ресурсу: https://zulko.github.io/moviepy/

14. Scikit-image [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-image.org/

15. OpenCV [Електронний ресурс] – Режим доступу до ресурсу: https://opencv.org/

16. Pulli, Kari & Baksheev, Anatoly & Kornyakov, Kirill & Eruhimov, Victor. (2012). Real-Time Computer Vision with OpenCV. Communications of the ACM. 55. 61-69. 10.1145/2184319.2184337.

17. Bradski, G. (2000) The OpenCV Library. Dr. Dobb’s Journal of Software Tools, 120; 122-125.

18. J. H. Kim, J. Hyeon Park and S. I. Cho, "Optimized Image Crop-based Video Retargeting," 2018 International SoC Design Conference (ISOCC), Daegu, Korea (South), 2018, pp. 224-225, doi: 10.1109/ISOCC.2018.8649938.

19. TIOBE Index [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/

20. ML.NET Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/machine-learning/

21. NumPY [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org/

22. TensorFlow [Електронний ресурс] – Режим доступу до ресурсу: - https://www.tensorflow.org/

23. Pandas [Електронний ресурс] – Режим доступу до ресурсу: - https://pandas.pydata.org

Додаток

```
# Підключення необхідних бібліотек

import cv2, time, pandas

import argparse

import pafy

from datetime import datetime

# Assigning our static_back to None

static_back = None


# List when any moving object appear

motion_list = [None, None]


time = []

borderSet = False

height = 0

width = 0


df = pandas.DataFrame(columns=["Start", "End"])

"""

url = "https://www.youtube.com/watch?v=O-MQC_G9jTU"

videoPafy = pafy.new(url)

best = videoPafy.getbest(preftype="mp4")

video = cv2.VideoCapture(best)

"""

# Завантаження відеозапису

parser = argparse.ArgumentParser()

parser.add_argument('image', type=str, help='path to image file')

args = parser.parse_args()

video = cv2.VideoCapture(args.image)
```

```
def resize_image(image, width, height,COLOUR=[0,0,0]):
    h, w, layers = image.shape
    if h > height:
        ratio = height/h
        image = cv2.resize(image,(int(image.shape[1]*ratio),int(image.shape[0]*ratio)))
    h, w, layers = image.shape
    if w > width:
        ratio = width/w
        image = cv2.resize(image,(int(image.shape[1]*ratio),int(image.shape[0]*ratio)))
    h, w, layers = image.shape
    if h < height and w < width:
        hless = height/h
        wless = width/w
        if(hless < wless):
            image = cv2.resize(image, (int(image.shape[1] * hless), int(image.shape[0] * hless)))
        else:
            image = cv2.resize(image, (int(image.shape[1] * wless), int(image.shape[0] * wless)))
    h, w, layers = image.shape
    if h < height:
        df = height - h
        df /= 2
        image = cv2.copyMakeBorder(image, int(df), int(df), 0, 0, cv2.BORDER_CONSTANT, value=COLOUR)
    if w < width:
        df = width - w
        df /= 2
        image = cv2.copyMakeBorder(image, 0, 0, int(df), int(df), cv2.BORDER_CONSTANT, value=COLOUR)
    image = cv2.resize(image,(1280,720),interpolation=cv2.INTER_AREA)
```

```
    return image


# Створення циклу для обробки відео покадрово
while True:
    # Зчитування кадру відео
    check, frame = video.read()
    if(frame is None):
        break
    # Ініціювання детектору руху
    motion = 0
    if borderSet is False:
        showCrosshair = False
        fromCenter = False
        borderSet = True
        r = cv2.selectROI(frame, fromCenter, showCrosshair)
        height, width, layers = frame.shape


    frame = frame[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2])]
    frame = resize_image(frame, width, height)


    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (21, 21), 0)
    if static_back is None:
        static_back = gray
        continue
    diff_frame = cv2.absdiff(static_back, gray)


    thresh_frame = cv2.threshold(diff_frame, 30, 255, cv2.THRESH_BINARY)[1]
```

```
thresh_frame = cv2.dilate(thresh_frame, None, iterations=2)
cnts, _ = cv2.findContours(thresh_frame.copy(),
                           cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

asd = (r[1] + r[3]) * (r[0] + r[2])
for contour in cnts:
    if cv2.contourArea(contour) < (r[1] + r[3]) * (r[0] + r[2])/1000:
        continue
    motion = 1

    (x, y, w, h) = cv2.boundingRect(contour)
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

motion_list.append(motion)

motion_list = motion_list[-2:]

# Запис часу початку знайденого руху
if motion_list[-1] == 1 and motion_list[-2] == 0:
    time.append(datetime.now())

# Запис часу завершення знайденого руху
if motion_list[-1] == 0 and motion_list[-2] == 1:
    time.append(datetime.now())

key = cv2.waitKey(1)

if key == ord('q'):
```

```
        if motion == 1:
            time.append(datetime.now())
        break
    # Відображення результату
    cv2.imshow("Color Frame", frame)



# Запис отриманих даних часу в DataFrame
for i in range(0, len(time), 2):
    df = df.append({"Start": time[i], "End": time[i + 1]}, ignore_index=True)


# Створення CSV файлу в який буде збережено отримані результати
df.to_csv("Результати.csv")


video.release()
cv2.destroyAllWindows()
```