НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«До захисту допущено»
В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.040301 «Прикладна математика»

на тему: «Сегментація руху з виділенням фону та оптичним потоком»

Виконав студент 4 курсу групи ФІ-51

Варежкін Сергій Євгенович

Керівник доц. кафедри ІБ, к. ф.-м. наук Южакова Г. О.

Консультант

Рецензент

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.040301 «Прикладна математика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

Варежкіна Сергія Євгеновича _____
(прізвище, ім'я, по батькові)

1. Тема роботи Сегментація руху з виділенням фону та оптичним потоком _____ _____ ,
науковий керівник роботи доц. кафедри ІБ, к. ф.-м. наук Южакова Г. О. _ _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «____ » 2019 р. № _____

2. Термін подання студентом роботи 10 червня 2019 р.

3. Вихідні дані до роботи _____ _____ _____

4. Зміст роботи_____ _____ _____ _____ _____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____ _____ _____ _____

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка

Студент _____ _____
(підпис) (ініціали, прізвище)

Керівник роботи _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Дипломна робота містить 42 сторінки, 7 ілюстрацій, 2 таблиці, 1 додаток і 7 джерел посилань.

Для розв'язання багатьох проблем комп'ютерного зору, таких як трекінг, знаходження теплокарт може знадобитись попередньо отримана сегментація зображень на об'єкти.

Об'єктом дослідження є рух на послідовності зображень.

Предметом дослідження є алгоритм сегментації руху на послідовності зображень.

Метою даної роботи є дослідження та розробка алгоритму сегментації руху на послідовності зображень, що забезпечить більш точні результати порівняно з іншими методами.

Для досягнення мети було використано

– Маска переднього плану, щоб отримати вхідні дані

– Оптичний потік зображення, щоб покращити попередні результати

– Алгоритм k-середніх, щоб розмежувати два об'єкти

– Мова C++ та бібліотека OpenCV, для програмної реалізації

СЕГМЕНТАЦІЯ РУХУ, ВИДІЛЕННЯ ФОНУ, ОПТИЧНИЙ ПОТІК, K-СЕРЕДНІХ, ТРЕКІНГ

ABSTRACT

Graduate work contains 42 pages, 7 figures, 2 tables, 1 appendix and 7 references.

Many computer vision problems such as tracking or calculating heat maps may require pre-computed image segmentation.

The object of study is motion in image sequence.

The subject of study is algorithm of motion segmentation in image sequence.

The study aims to explore and develop algorithm of motion segmentation in image sequence, which will increase segmentation quality compared to other methods.

To perform the study

– Foreground mask has been used to obtain input data

– Dense optical flow has been used to improve results from previous step

– k-means has been used to separate two objects

– C++ and OpenCV library to implement the algorithm

MOTION SEGMENTATION, BACKGROUND SUBTRACTION,
OPTICAL FLOW, K-MEANS, TRACKING

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Обмежувальний бокс — найменший прямокутник який повністю містить певний регіон зображення. Позначається R або r.

Маска — одноканалове зображення, пікселі якого приймають значення 0 або 255 (0 або 1).

px — піксель

OpenCV — відкрита бібліотека для комп'ютерного зору та машинного навчання [7].

ВСТУП

Актуальність роботи. Сегментація руху може знадобитись під час вирішення таких задач, як трекінг, чи знаходження теплокарт. Існує велика кількість методів та підходів до вирішення цієї задачі. Було визначення, що багато з цих методів не здатні давати адекватні результати під час наявності перекрить, тобто коли об'єкти переднього плану перекривають об'єкти заднього та в сценах з великою щільністю рухомих об'єктів.

Об'єкт дослідження — рух на послідовності зображень.

Предмет дослідження — алгоритм сегментації руху на послідовності зображень.

Мета дослідження. Аналіз та розробка алгоритму сегментації руху на послідовності зображень, що забезпечить більш точні результати порівняно з іншими методами.

Завдання:

1. Розглянути існуючі підходи до вирішення задачі.
2. Застосовуючи оптичний потік та алгоритм k-середніх покращити вхідну сегментацію.
3. Програмно реалізувати отриманий алгоритм.
4. Кількісно перевірити роботоздатність алгоритму.

Практичне значення одержаних результатів. Отриману сегментацію можна використовувати подалі під час обробки послідовності зображень. Програмну реалізацію було порівняно з декількома іншими методами та було визначено, що запропонований метод дає якісно та кількісно кращі результати у порівнянні з ними.

1 ПОСТАНОВКА ТА МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ СЕГМЕНТУВАННЯ РУХУ

В загальному випадку задача сегментації руху полягає у виділенні рухомих об’єктів з послідовності зображень. Загальної класифікації методів до вирішення цієї задачі не має, проте багато авторів виділяють описані подалі підходи до такої сегментації[2]. Більшість існуючих методів (наш у тому числі) використовують деякі з зазначених технологій, або їх комбінацію, тою чи іншою мірою.

У першому пункті цього розділу ми розглянемо підходи до вирішення нашої задачі, а у двох останніх як можна оптимально вирішувати задачу передбачення руху за допомогою оптичного потоку.

1.1 Огляд підходів до поставленої задачі

Найбільш істотним, а також найпростішим для реалізації є різницевий метод. Маючи послідовну пару кадрів ми можемо взяти різницю інтенсивностей у кожному пікселі першого кадру з наступним. Якщо ця різниця більше деякого критичного значення, то вважаємо що цей піксель відповідає руху. При використанні такого методу робиться припущення про статичне освітлення. Також, одразу зрозуміло, що цей метод є дуже чутливим до різних шумів, які можуть надходити як із самого відео, так і наприклад із руху камери.

Логічним розвитком попереднього методу є статистичні методи. Ці алгоритми роблять бінарну класифікацію пікселів зображення на пікселі заднього та переднього плану, присвоюючи відповідні ймовірності. Для цього використовуються гарно досліджені та відомі статистичні методи. Дуже поширеним та легким для реалізації є EM-алгоритм, на основі якого розроблено багато методів сегментації[5, 6].

Можливо також проводити сегментацію руху за допомогою оптичного потоку. Оптичний потік є векторним полем швидкості зміни інтенсивності пікселів на зображені. Головним недоліком є чутливість до шумів, складність отримати чітке уявлення о геометрії сегментованих об’єктів, та великі обчислювальні витрати. Однак останній недолік було подолано останнім часом внаслідок розвитку потужності обчислювальної техніки та розроблених апроксимативних алгоритмів [3, 4]. Оптичний потік також застосовується для вирішення задач іншого класу, наприклад, стабілізування руху камери.

В останні роки великої популярності набули нейронні мережі та машинне навчання. На разі вони дозволяють отримувати дуже точні результати, іноді навіть перевищуючи точність людини. Тем не менш, високо точні мережі володіють складною архітектурою та потребують великого обсягу даних для тренування. До того ж у міському середовищі може спостерігатись велика кількість різноманітних класів об’єктів які могли би бути потрібні, а це означає, що обсяг даних потрібних для тренування буде дуже швидко зростати.

Усі з вище зазначених методів окрім останнього також мають суттєві недоліки при застосуванні до складних сцен, наприклад руху на вулиці, де декілька об’єктів можуть вільно перетинатись, або наявні перепони, які частково перекривають деякі об’єкти. Це може призводити до неправильного розпізнавання, або декілька об’єктів будуть розпізнанні, або навпаки, відповідно. Це може суттєво позначитись на подальшій обробці вихідних даних, наприклад під час трекінгу. Запропонований подалі метод буде намагатись розв’язувати саме ці проблеми під час сегментації.

1.2 Рух твердого тіла на послідовності зображень

Як відомо кінематику твердого тіла у тривимірному просторі можна змоделювати як евклідове перетворення координат тіла з одного в інший момент часу,

якщо при цьому не відбувається деформацій. Нагадаємо, що евклідове перетворення складається з обертання та переміщення:

$$\vec{X}' = R \cdot \vec{X} + \vec{T} \tag{1.1}$$

де,

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \vec{T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \tag{1.2}$$

R - матриця обертання, яка володіє усіма відомими властивостями, а T - вектор переміщення координат (X, Y, Z). Тривимірному руху реального об'єкту буде відповідати двовимірний рух пікселів на послідовності зображень. Але в більш загальному випадку перетворення що буде відповідати цьому руху вже не буде евклідовим, адже якщо присутній, наприклад рух від камери, то розміри об'єкту вже будуть змінюватись, а отже і відстань між пікселями, тобто це перетворення не буде ізометрією. На справді це буде більш загальне перетворення, а саме афінне (евклідове перетворення є окремим випадком афінного).

Афінне перетворення ϕ задається лінійним відображенням та вектором перенесення. Якщо існує матриця A лінійного відображення, і $\vec{t}$ - це вектор перенесення, то афінне перетворення буде мати вигляд

$$\vec{y} = \phi(x) = A \cdot \vec{x} + \vec{t} \tag{1.3}$$

Дуже зручним є представлення у вигляді розширеної матриці:

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \left[\begin{array}{c|c} A & \vec{t} \\ 0 \dots 0 & 1 \end{array}\right] \cdot \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} \tag{1.4}$$

Маючи декілька послідовних перетворень $\phi_1, \phi_2 \dots \phi_k$, завдяки представленню у розширеній формі можна записати кінцеве перетворення як матричний добуток $\phi = \phi_1\phi_2 \dots \phi_k$.

Перетворення ϕ має обернене перетворення ϕ^{-1} тоді і тільки тоді, коли матриця A є оберненою. При цьому обернене перетворення буде мати наступний вигляд:

$$\phi^{-1} = \left[\begin{array}{c|c} A^{-1} & -A^{-1}\vec{t} \\ 0 \dots 0 & 1 \end{array}\right] \tag{1.5}$$

Для вирішення задачі передбачення руху насправді непотрібно знаходити перетворення у саме такому вигляді як ми його ввели, тобто окремо матрицю A та вектор $\vec{t}$. Рух об'єктів можна досліджувати використовуючи динаміку інтенсивностей на зображенні, а саме використовуючи оптичний потік.

1.3 Оптичний потік

Нехай $I(x, y, t)$ — інтенсивність у пікселі p з координатами (x, y) в момент часу t. Через час Δt піксель p перейде у точку $(x + \Delta x, y + \Delta y)$, тоді для інтенсивності це означатиме, що

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{1.6}$$

або

$$\Delta I(x, y, t) = 0 \tag{1.7}$$

Якщо розкласти інтенсивність у ряд Тейлора до лінійного члену, отримаємо наступне рівняння:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \tag{1.8}$$

або розділивши на Δt:

$$\frac{\partial I}{\partial x}v_x + \frac{\partial I}{\partial y}v_y = -\frac{\partial I}{\partial t} \tag{1.9}$$

де v_x, v_y — це $\frac{\Delta x}{\Delta t}$ та $\frac{\Delta y}{\Delta t}$ відповідно та віповідають компонентам швидкості інтенсивності $I(x, y, t)$.

Як видно рівняння має дві невідомі і тому не може бути вирішеним. Тому для розв'язку такого рівняння потрібні деякі доповняючі обмеження чи припущення.

Розв'язавши це рівняння ми отримаємо вектор швидкості переміщення для кожного пікселя $\vec{v}_p = (v_x^p, v_y^p)$. Якщо ми обчислювали оптичний потік для пари послідовних кадрів, тобто $\Delta t = 1$ кадр, то $(v_x^p, v_y^p) = (\Delta x, \Delta y)$. Тоді положення пікселя p' у наступному кадрі можна знайти як $(x', y') = (x + \Delta x, y + \Delta y)$.

Розглянемо приклад вище зазначених обмежень на методі Лукаса-Канаде[3], який ми застосуємо подалі.

1.3 Метод Лукаса-Канаде

Метод Лукаса-Канаде використовує припущення що переміщення в околі $Q(p)$ пікселя p співпадає з переміщенням самого пікселя. На зображені окіл $Q(p)$ зображається "вікном" — квадратною ділянкою з центром у p, висотою та шириною a. Кількість пікселів у цьому околі тоді дорівнює $n = a^2$.

Тоді, запропонована умова математично записується як:

$$\begin{cases} I_x(p_1)v_x + I_y(p_1)v_y = -I_t(p_1) \\ I_x(p_2)v_x + I_y(p_2)v_y = -I_t(p_2) \\ \vdots \\ I_x(p_n)v_x + I_y(p_n)v_y = -I_t(p_n) \end{cases} \tag{1.10}$$

де I_x, I_y, I_t — значення відповідних часткових похідних у координаті пікселя p_i у поточному часі.

Ця система рівнянь є перевизначеною і тому в загальному випадку може взагалі не мати розв'язків. Тому для розв'язку цієї системи використовують метод найменших квадратів, а саме, якщо переписати цю систему у матричній формі:

$$A\vec{v} = \vec{b} \tag{1.11}$$

де

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \\ I_x(p_n) & I_y(p_n) \end{bmatrix}, b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \vdots \\ -I_t(p_n) \end{bmatrix} \tag{1.12}$$

то остаточне рівняння буде мати вигляд:

$$A^T A\vec{v} = A^T\vec{b} \tag{1.13}$$

Розв'язок такого рівняння відомий і дорівнює:

$$\vec{v} = (A^T A)^{-1} A^T\vec{b} \tag{1.14}$$

На практиці часто застосовують ваговий метод, адже накладена умова є дуже жорсткою і може завжди не виконуватись. Реально переміщення більш віддалених пікселей у вікні може суттєво відрізнятись і тому їх вага повинна зменшуватись. Для такої системи розв'язок буде наступним:

$$\vec{v} = (A^T W A)^{-1} A^T W\vec{b} \tag{1.15}$$

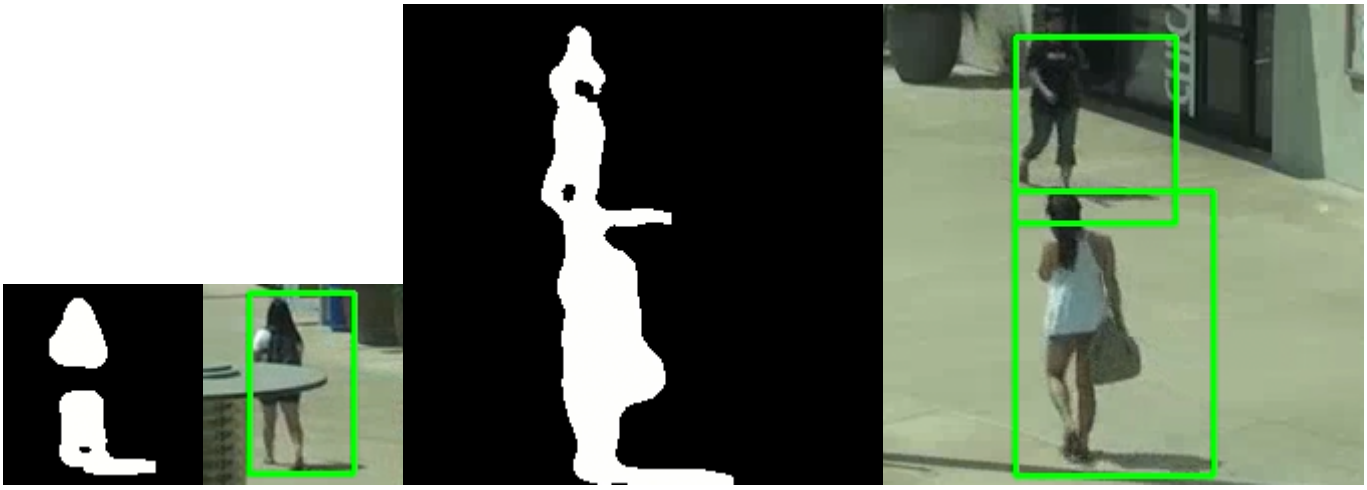
де W — матриця ваг.

Висновки до розділу 1

У цьому розділі ми розглянули деякі підходи до задачі сегментації руху, та виявили їх переваги та недоліки, особливо під час використання до складних сцен. Також було розглянуто як можна використовувати оптичний потік для передбачення руху. Це нам знадобиться у запропонованому методі, який буде описано у наступному розділі.

2 МЕТОД СЕГМЕНТАЦІЇ РУХУ

Як було вже зазначено, сегментація відео у міському середовищі викликає труднощі, але й в той самий час має великий інтерес для подальшої обробки результатів в різних галузях. Запропонований метод дозволяє в більшості випадків розв’язати ці проблеми.



(а) людина поділена навпіл ламаною

(б) цілий обмежувальний після обробки нашим алгоритмом

(в) дві людини мають спільну маску

(г) різні обмежувальні бокси після обробки нашим алгоритмом

Рисунок 2.1 — Приклад роботи запропонованого методу

Алгоритм сегментації складається з трьох етапів: знаходження маски переднього плану, об’єднання отриманих на попередньому кроці обмежувальних боксів з допомогою інформації отриманої з оптичного потоку, та на решті роз’єднання використовуючи k-середніх. Остаточно ми отримаємо набір більш точних обмежувальних боксів для подальшої обробки.

У наступних трьох пунктах будуть детально розглянуті вище зазначенні кроки разом с нюансами, що виникали під час практичної реалізації.

2.1 Виділення фону

Якість остаточної сегментації сильно залежить від початкової моделі фону, тобто від послідовності масок переднього плану, тому вибір алгоритму виділення заднього плану є досить суттєвим. Проте, алгоритм безпосередньо не влипає на подальшу роботу нашого методу, адже все що необхідно це лише остаточна маска. Незалежність від методу виділення фону дозволяє вільно використовувати той чи інший алгоритм, адже в залежності від умов відео що обробляється та іншої інформації, що може бути наявна, один може мати перевагу над іншим. Для реалізації було використано алгоритм MOG2, але як вже було зазначено могло бути використано будь-який інший. Ми вибрали саме цей метод через його робастність та простоту застосування.

Під час застосування маски переднього плану дуже важливим є її постобробка. Незалежно від якості застосованого методу все одно будуть траплятись шуми чи деякі об’єкти будуть менш відділенні від фону ніж інші. Для цього можуть бути використанні такі морфологічні операції як розкриття та закриття, або згладжування (наприклад медіанним фільтром) чи комбінація цих методів. Ми використовували наступний підхід до обробки маски:

1. По-перше поточне зображення згладжується гаусовим фільтром. Це робить остаточну модель фону більш розмитою, але також більш стійкою від шумів, що позитивно впливає на результуючу маску. Попереднє згладжування — часто зустрічаєма процедура під час обробки зображень.

2. Отримана маска згладжується медіанним фільтром. Це позбавляє остаточну маску від залишкових шумів та сильніше відділяє деякі складні для розпізнавання для алгоритму об’єкти. Розмір вікна фільтра залежить від роздільної здатності зображення. Для відео в HD[1] використовувалось вікно розміром

[1] Ширина та висота кадру — 1280x720

15 на 15.

На остаточній, вже обробленій, масці буде набір плям $\{A_i\}$. В загальному контексті під плямою розуміється суцільний регіон пікселів на зображенні однакової чи схожою інтенсивністю, в нашому випадку це буде лише білий колір. Навколо кожної плями A_i побудуємо прямокутник найменшої площі — обмежувальний бокс R_i. Набір $\{R_i\}$ боксів і є остаточним результатом першого кроку, який буде використовуватись подалі.

2.2 Об'єднання декількох об'єктів

Нехай R_i та R_j довільні обмежувальні бокси отримані з попереднього кроку. Нехай також $\vec{v}_k$, v^k_{min} і v^k_{max} — оптичний потік центру, мінімальне та максимальне значення за модулем оптичного потоку k-го обмежувального боксу відповідно.

Введемо наступну функцію відстані між боксами R_i та R_j:

$$d(R_i, R_j) = \max\{0, x_i - \tilde{x}_j, x_j - \tilde{x}_i\} + \max\{0, y_i - \tilde{y}_j, y_j - \tilde{y}_i\} \tag{2.1}$$

де, (x_k, y_k), $(\tilde{x_k}, \tilde{y_k})$ — координати крайнього лівого та правого кутів боксу R_k на зображенні. Якщо w_k та h_k — ширина та висота боксу, то $(\tilde{x_k}, \tilde{y_k}) = (x_k + w_k, y_k + h_k)$.

Якщо бокси хоч якось перетинаються то $d = 0$. В інших випадках d дорівнюватиме Мангеттенській відстані між найближчими кутами прямокутників.

Об'єднаємо бокси R_i та R_j якщо виконуються наступні 3 умови:

$$d(R_i, R_j) < d_{threshold} \tag{2.2}$$

$$\gamma = \angle(\vec{v}_i, \vec{v}_j) = \frac{\vec{v}_i \cdot \vec{v}_j}{\sqrt{||\vec{v}_i|| ||\vec{v}_j||}} < \gamma_{threshold} \tag{2.3}$$

$$v_i \sim v_j \tag{2.4}$$

Умова (2.2) перевіряє, що бокси знаходяться на відстані не більше $d_{threshold}$. Параметр $d_{threshold}$ є невідомим і визначається експериментально. Він залежить від таких факторів як роздільної здатності зображення, ракурсу, максимальної та мінімальної відстані для розпізнавання, тощо. Тому щоб підібрати ${}_d threshold$ потрібно зробити декілька, а іноді й більше тестових випробувань, та в чомусь покладатись на інтуїцію, маючи інформацію о відео що обробляється. Під час наших випробувань ми використовували значення $d_{threshold} = 14px$, для HD відео.

Збільшуючи значення $d_{threshold}$ ми збільшимо якість сегментації поблизу камери, при цьому більш далекі об'єкти будуть сегментуватись гірше. Аналогічна, але повністю навпаки ситуація буде якщо $d_{threshold}$ зменшувати.

Наступна умова перевіряє, що відхилення між напрямками руху не перевищує $\gamma_{threshold}$. В цей час цей параметр вже менш залежний від інших зовнішніх чинників. Ми вибрали значення $\gamma_{threshold} = \pi/3$. Це значення не дуже велике щоб об'єднувати насправді різні об'єкти й в той самий не час не настільки мале щоб пропускати об'єкти які треба було б об'єднати. Наприклад, якщо через перешкоду на задньому плані було відділено ногу від тулуба[2], то, в залежності від кадру, нога сама по собі буде мати дещо різну динаміку у порівнянні з тулубом. Тому порогове значення кута треба вибирати маючи такі ситуації на увазі. Треба також розуміти що великі значення $\gamma_{threshold}$ можуть призвести до не коректної сегментації. В загалі рекомендовано також цей параметр підбирати експериментально.

Було зроблено припущення, що $\vec{v}_k$ — це вектор оптичного потоку в центрі боксу R_k. На справді можна також використовувати й середнє значення оптичного потоку боксу R_k, причому в деяких випадках остаточна сегментація може опинитись кращою. Це залежить на сам перед від наявності тіней на масці та ракурсу зображення.

Остання умова потребує щоб бокси рухались приблизно з однією швидкістю.

[2]Мається на увазі, на масці

Треба зазначити, що в загальному випадку річ не іде про швидкість центру чи її середнє значення, а про швидкість боксів в загалом. Ми використовували наступний критерій в якості цієї умову:

$$[v^i_{min}, v^i_{max}] \cap [v^j_{min}, v^j_{max}] \neq \emptyset \tag{2.5}$$

де, v^k_{min} та v^k_{max} — мінімальне та максимальне значення по модулю оптичного потоку боксу R_k.

Можливо також використовувати критерій $|v_i - v_k| < \epsilon$, але на наш погляд це ще більше б ускладнило алгоритм, бо потрібно було б визначати ще один параметр ϵ.

Ця умова потрібна, щоб випадково не об'єднувати близькі об'єкти які рухаються з дуже різними швидкостями. Такі ситуації можуть дуже часто траплятись у міському середовищі, наприклад, пішохід та машина.

Таким чином, якщо два бокси R_i та R_j задовольняють умовам (2.2)-(2.4), то ми об'єднуємо їх в новий бокс $\tilde{R}$, побудувавши найменший прямокутник, що містить R_i та R_j.

Обробивши таким чином усі бокси ми отримаємо, вже меншого розміру, набір нових боксів для подальшої обробки.

2.3 Роз'єднання об'єкту

На цьому кроці ми будемо намагатись розділяти бокси що містять два об'єкти, що рухаються в різні напрямки. Робити ми це будемо за допомогою алгоритму k-середніх та оптичного потоку боксу.

Якщо бокс R містить два об'єкти, то пікселі цього боксу можна розбити на 3 неперетні класи: перший і другий об'єкти та фон. Оптичний потік заднього плану повинен бути майже нульовими, можливо з деякими незначними коливаннями від

нуль вектора. Якщо два об'єкти мають різну динаміку то їхні оптичні потоки також повинні відрізнятись. Нехай F — множина векторів оптичного потоку в кожному пікселі боксу R. Використаємо алгоритм k-середніх на цій множині для $k = 3$ і отримаємо 3 неперетні множини $F = F_1 \sqcup F_2 \sqcup F_3$. Тепер для кожної множини F_i побудуємо обмежувальний бокс r_i, що містить усі пікселі значення оптичного потоку в яких відповідає усім елементам F_i.

Вочевидь отримані бокси будуть якось між собою перетинатись. Треба дослідити як саме. Для цього введемо наступне відношення:

$$\rho_{ij} = \frac{S(r_i \cap r_j)}{\min(S(r_i), S(r_j))} \tag{2.6}$$

де, $S(\cdot)$ — площа обмежувального боксу, і вимірюється у квадратних пікселях.

С формули видно, що ρ_{ij} — безрозмірна величина і приймає дійсні значення від 0 до 1. Якщо $\rho_{ij} = 0$, то бокси r_i та r_j зовсім не перетинаються, а якщо $\rho_{ij} = 1$, то один повністю міститься в іншому. У нашому випадку ρ_{ij} приймає лише 3 різні значення.

Використовуючи розраховані значення ρ_{ij} розділ боксу R будемо виконувати наступним чином. Якщо $\rho_{ij} < \rho_{threshold}$ і для пари r_i та r_j не виконується умова (2.3) та до того ж $\rho_{ik} > \rho_{threshold}$ і $\rho_{jk} > \rho_{threshold}$, то бокс R розділяємо на r_i та r_j. У випробуваннях використовувалось значення $\rho_{threshold} = 0.4$.

Не виконуваність умови (2.3) потрібна, адже можливі ситуації коли об'єднаний на попередньому кроці об'єкт буде роз'єднано. З цього також випливає, що якість роз'єднання також залежить від параметра $\gamma_{threshold}$, тому це треба мати на увазі при знаходженні значення цього параметру. Якщо ж трапиться так, що умова $\rho_{ij} < \rho_{threshold}$ виконується для різних пар i та j, то взагалі в такій ситуації ми нічого сказати не можемо. Тому в такому випадку ми залишаємо бокс незмінним.

Розглянемо випадок коли R справді містить два об'єкти. Фон в загалі може

бути де завгодно в межах розглядаємого боксу, тому в загальному випадку бокс що відповідає задньому плану буде майже повністю містить обидва інших об'єкти. Якщо r_1 — бокс фону, то $\rho_{1i} \simeq 1$, для $i = 2, 3$. Якщо $\rho_{23} > \rho_{threshold}$, то більше ніж $\rho_{threshold}$ частини одного з об'єкти закрито іншим. В залежності від подальшої обробки даних, цього може бути недостатньо, тому ми залишаємо об'єкт без змін. В іншому випадку, остаточне рішення буде залежить від того чи виконується умова (2.3) чи ні.

Обробивши вище описаним чином усі вхідні з попереднього етапу бокси ми отримаємо остаточну сегментацію, яка і подається на виході нашого алгоритму.

Висновки до розділу 2

У цьому розділі було розглянуто та запропоновано метод сегментації руху, що використовує виділення заднього плану та інформацію щодо динаміки на зображені отриманої з оптичного потоку. Теоретично цей метод дає змогу суттєво покращити якість вихідної сегментації. Алгоритм був програмно реалізовано мовою програмування C++ за допомогою бібліотеки OpenCV [7]. У наступному розділі будуть наведені кількісні оцінки роботи цього методу.

3 ОЦІНКА ЯКОСТІ РОЗПІЗНАВАННЯ

Для випробувань та аналізу точності розпізнавання було використано відкритий датасет VIRAT Video Dataset [1]. Він містить понад 8 годин наземних та повітряних відео в різних умовах та ракурсах. Відео представлено в хорошій якості. Основною причиною вибору саме цього датасету є його відкритий доступ та наявність анотацій до кожного відео, тобто відомі істинні (GT) бокси об’єктів. Це дозволяє провести аналіз якості розпізнавання згідно з наведеною подалі методології.

3.1 Загальна методология оцінки якості роспізнавання бінарного класифікатора

Введемо декілька понять які нам знадобляться подалі. Бінарною класифікацією будемо називати класифікацію заданої множини об’єктів у дві неперетинні групи. Для оцінки такої класифікації існує безліч різних і специфічних метрик та підходів, які використовуються в абсолютно різних областях класифікації. Однак, існує більш менш універсальний метод, яких використовує чотири базові характеристики: істинно позитивна (TP), істинно негативна (TN), помилково позитивна (FP) й помилково негативна (FN) класифікації.

Назви "позитивна" та "негативна" прийшли з медицини, де в якості об’єктів для розпізнавання є хвороби. В цьому випадку "позитивна" класифікація відповідає наявності хвороби, в той час як "негативна" — відсутності. В загальному випадку коли наявні лише два класи, наприклад 1 та 2, то "позитивна" класифікація буде відповідати першому, а "негативна" — другому. Істинно позитивна класифікація означає, що об’єкт який насправді належить класу 1, було розпізнано як елемент класу 1, у свою чергу помилково негативна класифікація говорить, що

цей об'єкт було (помилково) віднесено до класу 2. Аналогічно визначаються інші дві характеристики.

Позначимо через TP кількість усіх істинно позитивних класифікацій. Аналогічно введемо змінні TN, FP та FN. Кількість усіх позитивних класифікацій P тоді виражається як $P = TP + FN$; кількість усіх негативних класифікацій N дорівнює $N = TN + FP$. Тепер ми можемо ввести першу пару характеристик, які можуть дати кількісну оцінку якості розпізнавання, а саме — чутливість та специфічність:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{3.1}$$

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{3.2}$$

При досить великих P та N, TPR та TNR будуть збігатись до ймовірностей позитивної та негативної класифікацій відповідно. Якщо ми також введемо спряженні змінні $FNR = 1 - TPR$ та $FPR = 1 - TNR$, то вони у свою чергу будуть дорівнювати помилкам першого та другого роду відповідно.

Іншими важливими характеристиками є PPV та NPV:

$$PPV = \frac{TP}{TP + FP} \tag{3.3}$$

$$NPV = \frac{TN}{TN + FN} \tag{3.4}$$

Якщо TPR визначає ймовірність істинно позитивної класифікації в загалом, то PPV визначає ймовірність, що позитивна класифікація буде насправді істинно позитивною, а не помилковою. Аналогічно визначається характеристика NPV, але для негативної класифікації.

Приклад 3.1. Нехай $TP = 2$, $TN = 1$, $FP = 1$ і $FN = 0$. Тоді $TPR = 1$, $TNR = \frac{2}{3}$, $PPV = \frac{1}{2}$, $NPV = 1$.
Бачимо, що хоча чутливість дорівнює 1, ймовірність що позитивна класифікація є істинною дорівнює $\frac{2}{3}$, бо присутня помилкова позитивна класифікація. Навпаки, специфічність дорівнює $\frac{1}{2}$, проте усі негативні класифікації є істинними.

Рідше, але також зустрічається точність ACC:

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.5}$$

Використовуючи вище приведенні характеристики можна розрахувати кількісну оцінку розпізнавання для бінарного класифікатора, як для підбору параметрів так і для порівняння різних методів. Вагомою перевагою цього підходу є його незалежність від досліджуваного методу та його області застосування.

3.2 ROC простір

Знов розглянемо довільний бінарний класифікатор. Нехай класифікація здійснюється відносно якогось керуючого параметру T, тобто перевіряється нерівність $X < T$, де X - довільна функція яка залежить від параметрів об'єктів що класифікуються. Треба визначити оптимальне значення T щоб покращити якість класифікації. Це можно зробити користуючись введеними характеристиками у попередньому параграфі.

Проваріюємо довільним чином параметр T і здійснемо класифікацію відносно нового значення параметру T. Отримуємо набір класифікацій, якість яких кількісно та якісно відрізняються. Введемо тепер так званий ROC (receiver operating characteristics) простір. Ось y буде відповідати TPR, а ось x — FPR. Обчисливши лише ці характеристики, кожний результат попередньої класифікації тоді можно зобразити як точку у чьому просторі.

Ми хочемо максимізувати TPR, при цьому мінімізуючи FPR. Тоді найкращий результат буде лежати у точці $(0, 1)$. Ця точка відповідає 100% роспізнаванню першого класу і відсутності його міскласифікації. Класифікацію що лежить у цій точці також називають ідеальною класифікацією. Бісектриса буде відповідати випадковій класифікації з ймовірностями $\{\frac{1}{2}, \frac{1}{2}\}$, для кожного з класів. Точки зверху діагоналі відповідають кращим за випадкову класифікаціям, а знизу відповідно гіршим. Стоїть зазначити, що якщо результат поганої класифікації повністю зробити навпаки то ми отримаємо гарну класифікацію яка буде знаходитись симетрично відносно діагоналі зверху.

Нанесевши усі класифікації на графік ROC простору, отримаємо криву, яку називають ROC крива. Бо ми хочемо отримати класифікацію близьку до ідеальної, то саме оптимальне значення параметру T буде у крайній лівій зверху класифікації.

На жаль даний метод не дуже сильно підходить до класифікаторів, що залежать від декількох різних параметрів, адже тоді потрібно варіювати їх усіх водночас, що може займати великий обсяг часу та ресурсів.

3.3 Застосування до нашого методу

Хоча ми вели розмову лише о бінарних класифікаторах, попередньо введену методологію оцінювання також можна застосувати й до нашого алгоритму. Розглянемо наш метод як бінарний класифікатор, що класифікує пікселі послідовності зображень, на пікселі що містять рухомі, або динамічні об'єкти та які містять нерухомі, або статичні об'єкти.

Введемо ще декілька визначень та операцій, що нам знадобляться подалі.

Нехай $\wedge$, $\vee$, $\neg$, $\oplus$ — двійкове І, АБО, заперечення та виключаюче АБО (додавання по модулю 2) відповідно. Ці операції визначенні для 0 та 1, але їх можна розширити до натуральних чисел. Покажемо яким чином це можна зробити на наступному прикладі.

Нехай ми хочемо знайти чому дорівнює $14 \wedge 23$. Переведемо обидва числа в їх двійкове представлення, отримаємо, що $14_2 = 00001110$, $23_2 = 00010111$. Застосував I побітно ми отримаємо наступний результат — 00000110. Якщо це число перевести назад у десяткову систему відліку то отримаємо $00110_{10} = 6$. Таким чином маємо, що $14 \wedge 23 = 6$.

Тобто, щоб застосувати двійкову операцію до нормальних чисел, необхідно спочатку перевести числа у двійкове представлення, потім застосувати цю операцію побітно і в кінці перевести результат назад, у десяткове представлення.

Нехай I_1 й I_2 — два одноканальні зображення однакових розмірів. Тоді ми можемо перенести усі двійкові операції на такі зображення, як їх поелементне застосування:

$$I_1 \wedge I_2 = \{p_{ij} \wedge q_{ij}\} \tag{3.6}$$

$$I_1 \vee I_2 = \{p_{ij} \vee q_{ij}\} \tag{3.7}$$

$$\neg I = \{\neg p_{ij}\} \tag{3.8}$$

$$I_1 \oplus I_2 = \{p_{ij} \oplus q_{ij}\}. \tag{3.9}$$

Для нас більший інтерес будуть мати випадки коли пікселі зображень можуть приймати лише два конкретні значення $p_{ij} \in \{0, 255\}$. Зображення для яких це вірно для усіх p_{ij} називаються бінарними. Для зручності замість значення 255, можна використовувати 1. Ясно, що це ніяк не впливає на результат двійкових операцій, бо двійкове представлення $255_2 = 11111111$.

Використовуючи методологію з попереднього пункту, нам потрібно знайти TP, TN, FP й FN. У нашому випадку класифікуються пікселі зображення. Наприклад, щоб порахувати значення TP, треба підрахувати кількість пікселів які ми розпізнали як рухаючись та які насправді відносяться до рухомого об'єкта, згідно

з анотацією відео з датасету. Використовуючи тільки що введені визначення це легко зробити.

Нехай після обробки зображення ми отримали набір із n обмежувальних боксів $\{R_i\}$. Побудуємо наступну маску I наступним чином:

$$I = \{p_{ij}\}, p_{ij} = \begin{cases} 1 & p_{ij} \in A_k \\ 0 & p_{ij} \notin A_k \end{cases}, \forall k = \overline{1, n} \tag{3.10}$$

Тобто ми маскуватимемо отриманні бокси. Аналогічно отримаємо маску I_{GT} згідно з анотацією. Тоді, якщо $||I|| = \sum_i \sum_j p_{ij}$, в нашому випадку це буде кількість білих пікселів, то потрібні нам характеристики легко виражаються наступним чином:

$$TP = ||I \wedge I_{GT}|| \tag{3.11}$$

$$TN = ||\neg(I \vee I_{GT})|| \tag{3.12}$$

$$FP = ||(I \wedge I_{GT}) \oplus I_{GT}|| \tag{3.13}$$

$$FN = ||(I \wedge I_{GT}) \oplus I|| \tag{3.14}$$

Даний підхід є дуже ефективним, адже використовує лише двійкові операції, які ЕОМ виконує за один такт. Також в більшості програмних пакетах ці операції вже реалізовані для матриць в тому самому вигляді як ми їх ввели, тому програмна реалізація не викликає жодних зусиль.

Таким чином, обчислив такі класифікаційні маски на кожному кадрі, ми легко можемо обчислити усі необхідні нам характеристики потрібні для подальшого дослідження та аналізу якості розпізнавання.

3.4 Отриманні результати

Для випробувань було використано 3 відео с датасету VIRAT Video Dataset кожне обсягом приблизно у 10 хвилин. Кожне відео було знято з частотою кадрів 24 кадрів за секунду, тобто було оброблено приблизно 43200 кадрів. Ми обробили відео нашим метод використовуючи вхідні маски отриманні алгоритмом MOG2, а також ті самі маски, але з видаленими тінями, які може видаляти MOG2. Результати були порівнянні з сегментацією використовуючи саму маску. Отримані дані були занесені до наступної таблиці.

Таблиця 3.1 — Результати випробувань (жирним виділенно найкращі результати)

	Наш метод		Наш метод з видаленням тіней		Маска з MOG2	
Відео	TPR	PPV	TPR	PPV	TPR	PPV
VIRAT_S_010200	**0.656**	0.598	0.433	**0.808**	0.635	0.596
VIRAT_S_010201	**0.709**	0.582	0.461	**0.820**	0.691	0.592
VIRAT_S_010202	**0.686**	0.623	0.380	**0.898**	0.656	0.643

Проаналізувавши отриманні результати, перше що кидається в очі — це порівняно великі значення PPV при видалені тіней, тобто лише приблизно 15% позитивних розпізнавань є хибними. Це пояснюється тим, що включаючи в отриманий бокс тінь ми також включаємо велику кількість зайвого фону. Проте, як видно метод видалення тіней, що реалізовано в алгоритмі MOG2, також видаляє велику частину об'єктів. Таким чином розвиток задачі відділяння тіней, та така постобробка масок переднього плану може суттєво покращити результати нашого методу та інших методів. Стоїть зазначити що отримані значення TPR є дещо заниженими через те що сторони боксів, на яких немає тіней виходять більш точними у порівнянні з запропонованими. На значення TPR суттєво впливає ще той факт, що використаний алгоритм виділення фону не в змозі відділяти об'єкти на

великих відстанях (хоча це також залежить від ракурсу камери), а також об'єкти що зупиняються на деякий проміжок час будуть губитись. На жаль з цими недоліками ми нічого не можемо зробити, проте отриманні результати дають нам змогу чітко зрозуміти їх наявність.

(а) наш метод

(б) наш метод з видаленням тіней

(в) маска переднього плану

Рисунок 3.1 — Порівняння методів

Проте, як виявилось, ці характеристики не дають змоги повністю кількісно порівняти роботу декількох, у порівнянні з якісною оцінкою. Тому, введемо ще одну, наступну, метрику для оцінки якості сегментації.

Нехай N — кількість отриманих боксів нашим методом, а N_{GT} — кількість істинних боксів. Обчислимо наступну величину:

$$\nu = \frac{N}{N_{GT}} \tag{3.15}$$

З визначення видно, що ν може приймати значення від 0 до ∞. При порівняні методів, звісно кращим буде той, значення ν якого буде ближче до 1. Треба зазначити, що такий підхід сам по собі може давати дещо хибні результати, наприклад, якщо ми будемо розпізнавати ту саму кількість об'єктів з іншого класу. Тому ми використовуємо цю метрику разом з іншою. Якщо друга дає змогу сказати скільки саме об'єктів ми розпізнаємо, то перша дає змогу переконатись як ці отримані об'єкти відповідають істинним.

Також можна ввести характеристику $|1-\nu|$. Вона буде показувати яку частку істинних об'єктів ми не розпізнали, або розпізнали зайві, в залежності чи $\nu < 1$, чи $\nu > 1$.

Якщо трапляється така ситуація, що для одного метод $\nu_1 < 1$, а для іншого $\nu_2 > 1$, причому $|1-\nu_1| = |1-\nu_2|$, то зазвичай перший метод буде краще. Це пов'язано з тим, що у першому випадку ми пропускаємо деякі об'єкти, коли у другому ми точно розпізнаємо зайві, чи якісь об'єкти розпізнаються не як один. У випадку трекінгу проблему що виникає у першому випадку набагато легше розв'язати ніж другу.

Зробивши необхідні розрахунку ми отримали наступні результати для нашого методу, та маски з MOG2.

Таблиця 3.2 — Результати випробувань (жирним виділенно найкращі результати)

	Наш метод		Маска з MOG2	
Відео	ν	$\|1-\nu\|$	ν	$\|1-\nu\|$
VIRAT_S_010200	**0.76382206**	0.23617794	4.803753906	3.803753906
VIRAT_S_010201	**0.834352128**	0.165647872	7.513383765	6.513383765
VIRAT_S_010202	**0.771273709**	0.228726291	5.463877758	4.463877758

Як видно наш метод видає в середньому 79% боксів, той час як результати сегментації за допомогою маски перевищують істинні майже у 6 раз. "Від'ємний" результат знов пояснюється вхідною маскою — об'єкти на великих відстанях (приблизно 25 метрів від камери) не розпізнаються, або розпізнаються дуже погано. Також використаний метод виділення заднього плану не в змозі відділяти від фону об'єкти що перестали рухатись на протязі деякого часу.

Висновки до розділу 3

Отримані результати тепер повністю дають змогу переконатись у перевазі нашого методу над наївною сегментацією за допомогою маски переднього плану.

Також як можна побачити результати нашого методу суттєво залежать від якості самої маски, як було зазначено при видалені тіней точність PPV зросла майже до 84 відсотків. Тем не менш кількість істинно позитивних розпізнавань суттєво впала — це на сам перед, як вже було пояснено, пояснюється тим, що разом як тіні класифікуються деякі частини багатьох об'єктів. Окрім уже описаних двох випадків негативного розпізнавання, а саме далеких від камери та статичних об'єктів, ще одною причиною заниженого показника TPR є більш точні бокси на сторонах де відсутні тіні. Це компенсується під час обрахунків PPV, але негативно впливає на значення TPR, хоча на справді являється ще одним плюсом нашого методу.

Таким чином проведений аналіз дав змогу побачити суттєві переваги нашого методу над іншим та також допоміг виявити деякі недоліки. Як виявилось, у багатьох ситуаціях ці недоліки можна врахувати під час подальшої обробки результатів.

ВИСНОВКИ

Під час виконання роботи було досліджено декілька підходів до вирішення задачі сегментації руху, та виявлено їх суттєві недоліки. Запропонований метод, використовує маску переднього плану, оптичний потік та алгоритм k-середніх, щоб вирішити деякі з цих проблем.

Алгоритм було програмно реалізовано на мові C++ з використанням бібліотеки OpenCV. Ця реалізація була потім використана для оцінки якості.

Було розглянуто декілька метрик для порівняння якості розпізнавання. Отриманні результати свідчать про те, що наш метод кількісно та якісно кращий за попередньо розглянуті нами підходи. Також було з’ясовано, що покращуючи видалення тіней та моделі фону, можна суттєво покращити результати нашого методу.

Одним із недоліків запропонованого методу є велика кількість параметрів, що суттєво залежать від відео, що обробляється. Наразі невідомо чи можна ці параметри знаходити динамічно чи ні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

[1] Amitha Perera Naresh Cuntoor Chia-Chih Chen Jong Taek Lee Saurajit Mukherjee J.K. Aggarwal Hyungtae Lee Larry Davis Eran Swears Xiaoyang Wang Qiang Ji Kishore Reddy Mubarak Shah Carl Vondrick Hamed Pirsiavash Deva Ramanan Jenny Yuen Antonio Torralba Bi Song Anesco Fong Amit Roy-Chowdhury Sangmin Oh, Anthony Hoogs and Mita Desai. ”a large-scale benchmark dataset for event recognition in surveillance video”.

[2] Luca Zappella, Xavier Lladó, and Joaquim Salvi. Motion segmentation: A review. In *Proceedings of the 2008 conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 398–407. IOS Press, 2008.

[3] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[4] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[5] Zoran Zivkovic et al. Improved adaptive gaussian mixture model for background subtraction. In *ICPR (2)*, pages 28–31. Citeseer, 2004.

[6] Zoran Zivkovic and Ferdinand Van Der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

ДОДАТОК А

```
1  #pragma once
2
3  #include <opencv2/core.hpp>
4
5  class segment {
6  public:
7    cv::Rect bounds;           //represents segment's position and size
8    cv::Point2f mean_flow;     //optical flow at the center of bounding box
9    float max_flow;            //not actual max and min values but rather statistical
         ones i.e. caluclated as mean + stddev
10   float min_flow;
11
12 public:
13   segment() = default;
14   segment(const cv::Rect& bounds, cv::InputArray flow = cv::noArray(), cv::
         InputArray mask = cv::noArray());
15 public:
16   //calcluates optical flow properties
17   void set_flow(cv::InputArray flow, cv::InputArray mask = cv::noArray());
18 public:
19   //returns segmenton the next frame based on current optical flow and assigns new
         optical flow
20   segment operator()(cv::InputArray flow, cv::InputArray flow_next = cv::noArray(),
          cv::InputArray mask = cv::noArray()) const;
21 };
```

```
1  #include "segment.hh"
2
3  #include <opencv2/core.hpp>
4  #include <opencv2/imgproc.hpp>
5
6  segment::segment(const cv::Rect& bounds, cv::InputArray _flow, cv::InputArray _mask
       ): bounds(bounds) {
7    if (!_flow.empty()) set_flow(_flow);
8  }
```

```
 9
10 void segment::set_flow(cv::InputArray _flow, cv::InputArray _mask) {
11   auto flow     = _flow.getMat();
12   auto roi      = flow(bounds);
13   auto roi_mag  = cv::Mat(roi.size(), CV_32FC1);
14
15   cv::Scalar mean_mag, stddev_mag, mean;
16   cv::Mat roi_mask;
17
18   if (!_mask.empty()) {
19     auto mask = _mask.getMat();
20     roi_mask = mask(bounds);
21   }
22
23   mean = cv::mean(roi, roi_mask);
24   mean_flow.x = mean[0];
25   mean_flow.y = mean[1];
26
27   std::transform(roi.begin<cv::Point2f>(), roi.end<cv::Point2f>(), roi_mag.begin<
       float>(),
28                  [](const cv::Point2f & a) { return cv::norm<float>(a); });
29   cv::meanStdDev(roi_mag, mean_mag, stddev_mag, roi_mask);
30   max_flow = mean_mag[0] + stddev_mag[0];
31   min_flow = mean_mag[0] - stddev_mag[0];
32 }
33
34 segment segment::operator()(cv::InputArray _flow, cv::InputArray _flow_next, cv::
     InputArray _mask) const {
35   auto flow = _flow.getMat();
36   auto points = std::vector<cv::Point2f>(bounds.width * bounds.height);
37   auto r = bounds;
38   r.x = std::max(0, std::min((int)(r.x + mean_flow.x), 1278));
39   r.y = std::max(0, std::min((int)(r.y + mean_flow.y), 718));
40   if (r.x + r.width >= 1280) {
41     r.width = 1279 - r.x;
42   }
43   if (r.y+ r.height >= 718) {
44     r.height = 719 - r.y;
```

```
45    }
46
47    return { r, _flow_next, _mask };
48  }
```

```
1   #pragma once
2
3   #include <opencv2/bgsegm.hpp>
4   #include <opencv2/core.hpp>
5
6   #include "segment.hh"
7
8   class segmenter {
9   private:
10    enum state {
11      NOT_DISCOVERED = 0,
12      DISCOVERED,
13      MERGED,
14    };
15
16  private:
17    cv::Ptr<cv::BackgroundSubtractorMOG2> bg_subtracktor;
18    int dist_threshold;
19    double learning_rate;
20
21  public:
22    segmenter(int dist_threshold = 625,  double learning_rate = -1,
23              int history = 1200, double varThreshold = 20.0);
24
25  public:
26    inline void train(cv::InputArray scene, cv::OutputArray mask) {
27      bg_subtracktor->apply(scene, mask, learning_rate);
28    }
29    inline void clear() { bg_subtracktor->clear(); }
30    inline void bg(cv::OutputArray bg_image) const { bg_subtracktor->
          getBackgroundImage(bg_image); }
31
32  private:
```

```
33    std::vector<segment> merge(const std::vector<segment>& segments, cv::InputArray
          flow);
34
35  public:
36    std::vector<segment> operator()(cv::InputArray prev, cv::InputArray next, cv::
          InputOutputArray flow, cv::InputArray mask, bool merge = true);
37  };
38
39  segment merge(const segment& a, const segment& b,
40                cv::InputArray flow = cv::noArray());
41  bool is_mergeable(const segment& a, const segment& b,
42                    int dist_threshold, double direc_threshold = CV_PI / 2);
```

```
 1  #include "segmenter.hh"
 2
 3  #include <algorithm>
 4  #include <iostream>
 5  #include <vector>
 6
 7  #include <opencv2/bgsegm.hpp>
 8  #include <opencv2/core.hpp>
 9
10  #include "math.hh"
11
12  segmenter::segmenter(int dist_threshold, double learning_rate,
13                       int history, double var_threshold) :
14    dist_threshold(dist_threshold), learning_rate(learning_rate) {
15    bg_subtracktor = cv::createBackgroundSubtractorMOG2(history, var_threshold, false
          );
16    //bg_subtracktor->setShadowThreshold(shadow_threshold);
17  }
18
19  std::vector<segment> segmenter::merge(const std::vector<segment>& segments, cv::
        InputArray _flow) {
20    auto n = (int)segments.size();
21    auto merge_state = std::vector<state>(n, NOT_DISCOVERED);
22    auto merged_objects = std::vector<segment>();
23
```

```
24    for (int i = 0; i < n; i++) {
25      auto merged = segments[i];
26
27      if (merge_state[i] == MERGED)
28        continue;
29
30      merge_state[i] = MERGED;
31
32      for (int j = 0; j < n; j++) {
33        if (merge_state[j] == MERGED)
34          continue;
35
36        merge_state[j] = DISCOVERED;
37
38        if (is_mergeable(merged, segments[j], dist_threshold)) {
39          merged = ::merge(merged, segments[j], _flow);
40          merge_state[j] = MERGED;
41          j = 0;
42        }
43      }
44      if(merged.bounds.area() > 100 && merge_state[i])
45        merged_objects.push_back(merged);
46    }
47
48    return merged_objects;
49  }
50
51  std::vector<segment> segmenter::operator()(cv::InputArray _prev, cv::InputArray
        _next, cv::InputOutputArray _flow, cv::InputArray _mask, bool _merge) {
52    auto prev = _prev.getMat(), next = _next.getMat(), mask = _mask.getMat();
53    auto flow = _flow.getMat();
54
55    cv::Mat gray_prev, gray_next;
56    std::vector<std::vector<cv::Point>> contours;
57    std::vector<cv::Vec4i> hierarchy;
58    std::vector<segment> segments;
59
60    cv::cvtColor(prev, gray_prev, cv::COLOR_BGR2GRAY);
```

```
61    cv::cvtColor(next, gray_next, cv::COLOR_BGR2GRAY);
62    cv::cvtColor(mask, mask, cv::COLOR_BGR2GRAY);
63
64    cv::findContours(mask, contours, hierarchy, cv::RETR_EXTERNAL, cv::
          CHAIN_APPROX_SIMPLE);
65    cv::calcOpticalFlowFarneback(gray_prev, gray_next, flow, 0.8, 3, 31, 3, 7, 1.5,
          cv::OPTFLOW_FARNEBACK_GAUSSIAN);
66
67    segments.resize(contours.size());
68    std::transform(contours.begin(), contours.end(), segments.begin(),
69                   [&flow, &mask](const std::vector<cv::Point> & a) -> segment {
70                     return { cv::boundingRect(a), flow, mask };
71                   });
72    segments = merge(segments, flow);
73
74    std::vector<segment> segments_final;
75
76    for (int i = 0; i < segments.size(); i++) {
77      auto roi        = flow(segments[i].bounds);
78      auto samples    = cv::Mat(roi.cols * roi.rows, 2, CV_32F);
79      auto clusters   = std::vector<std::vector<cv::Point2i>>(3);
80      auto rects      = std::vector<cv::Rect>(3);
81      auto k_segments = std::vector<segment>(3);
82      auto ratio      = std::vector<float>(3);
83
84      cv::Mat labels, centers;
85      std::vector<segment> out;
86
87      for (int x = 0; x < roi.cols; x++) {
88        for (int y = 0; y < roi.rows; y++) {
89          samples.at<float>(y * roi.cols + x, 0) = roi.at<cv::Point2f>(y, x).x;
90          samples.at<float>(y * roi.cols + x, 1) = roi.at<cv::Point2f>(y, x).y;
91        }
92      }
93
94      cv::kmeans(samples, clusters.size(), labels, cv::TermCriteria(cv::TermCriteria
            ::EPS | cv::TermCriteria::COUNT, 10, 0.1), 3, cv::KMEANS_PP_CENTERS, centers
          );
```

```
95
96      for (int x = 0; x < roi.cols; x ++) {
97        for (int y = 0; y < roi.rows; y ++) {
98          auto idx = labels.at<int>(x + y * roi.cols, 0);
99        }
100     }
101
102     for (int x = 0; x < roi.cols; x++) {
103       for (int y = 0; y < roi.rows; y++) {
104         auto idx = labels.at<int>(x + y * roi.cols, 0);
105         clusters[idx].push_back({ segments[i].bounds.x + x, segments[i].bounds.y +
              y });
106       }
107     }
108
109     for (int idx = 0; idx < clusters.size(); idx++) {
110       if (clusters[idx].empty()) continue;
111       k_segments[idx] = { cv::boundingRect(clusters[idx]), flow };
112       k_segments[idx].mean_flow = { centers.at<float>(idx, 0), centers.at<float>(
            idx, 1) };
113     }
114     for (int i = 0; i < clusters.size(); i++) {
115       if (clusters[i].empty()) continue;
116       for (int j = i + 1; j < clusters.size(); j++) {
117         if (clusters[j].empty()) continue;
118         if (out.size() == 2) break;
119         auto ratio = (float)(k_segments[i].bounds & k_segments[j].bounds).area() /
              std::min(k_segments[i].bounds.area(), k_segments[j].bounds.area());
120         if (ratio < 0.4f && angle(k_segments[i].mean_flow, k_segments[j].mean_flow)
              > CV_PI / 3) {
121           out.push_back(k_segments[i]);
122           out.push_back(k_segments[j]);
123         }
124       }
125     }
126
127     if (out.empty())
128       out.push_back(segments[i]);
```

```
129
130       segments_final.insert(segments_final.end(), out.begin(), out.end());
131     }
132     _flow.assign(flow);
133     //segments_final = segments;
134     return segments_final;
135   }
136
137   bool is_mergeable(const segment& a, const segment& b,
138                     int dist_threshold, double direc_threshold) {
139     auto direc      = angle(a.mean_flow, b.mean_flow);
140     auto dist       = distance(a.bounds, b.bounds);
141     auto is         = dist <= dist_threshold && direc <= direc_threshold &&
142                       a.min_flow <= b.max_flow && b.min_flow <= a.max_flow;
143     return is;
144   }
145
146   segment merge(const segment& a, const segment& b, cv::InputArray flow) {
147     auto bounds = a.bounds | b.bounds;
148     return segment(bounds, flow);
149   }
```