КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

СТВОРЕННЯ ЗАСТОСУНКУ ДЛЯ ВИДАЛЕННЯ ФОНУ ЗОБРАЖЕННЯ

Дипломна робота бакалавра
студента 4 року навчання
Спеціальність: 123 «Комп'ютерна інженерія»
Желдака Іллі Сергійовича

Науковий керівник
к. ф.-м. н., доцент кафедри МТРФ
Іваненко Дмитро Олександрович

Рецензент
д. ф.-м. н., професор кафедри алгебри механіко-математичного факультету
Шевченко Георгій Михайлович

До захисту допускаю
Протокол засідання кафедри від

«___»_____2022р. №____

Завідувач кафедрою
Проф. Погорілий Сергій Дем'янович

КИЇВ - 2022

РЕФЕРАТ

Пояснювальна записка: – 30 с., 13 рис., 6 джерел.

Об'єкт дослідження – методи видалення фону зображення.

Мета роботи: Створення застосунку для видалення фону зображення.
Передбачувані наукові результати: створений застосунок для видалення фону зображення на основі сучасних технологій, порівняння з попередніми технологіями.
Ключові слова Background subtraction, saliency, edge detection.

ЗМІСТ

ВСТУП

На сьогоднішній день існує багато різних методів визначення основного об’єкта на зображенні, його виділення, та видалення фону, проте більшість найбільш ефективних методів попри відкритість наукових досліджень на практиці доступні лише у рішеннях з закритим вихідним кодом. Метою цієї роботи є створення програмної реалізації одного з найефективніших методів видалення фону зображення.

РОЗДІЛ 1 ПОМІТНІСТЬ ОБ’ЄКТІВ НА ЗОБРАЖЕННІ

Для видалення фону зображення потрібно в першу чергу виділити основний об’єкт на зображенні, фон якого має бути видалений. Існує велика кількість методів відділення основного об’єкта від фону, які відрізняються за ефективністю, швидкодією та результативністю.

Базове виявлення руху

Найпростішим способом виділення фону об’єкта це використати окреме зображення у відтінках сірого без рухомих об'єктів. Це
зображення може бути знімком, зробленим завчасно, або отриманим за допомогою медіанного фільтра з камери відеоспостереження. Після цього видалення фону досягається підрахунком пікселів, які суттєво змінилися в порівнянні із фоновим зображенням.

Даний метод підходить як правило для серії зображень зроблених з однієї точки у приблизно однаковий час доби, та за схожих погодних умов.

До недоліків цього методу можна віднести те що він не дозволяє визначити конкретні об’єкти на зображенні, а, в основному, лише сам факт зміни зображення, також варто відмітити його вразливість до шуму зображення що погано впливає на якість результату. Тож його доцільно застосовувати коли необхідно визначити лише наявність суттєвих змін за короткий час.

Рис. 1 – Базове видалення фону.

Єдине зображення за Гаусом

В ідеальних умовах при моделюванні видалення фону за одним зображенням, воно не має містити візуального шуму та спотворень. Оскільки на практиці досягнення таких умов практично неможливе, для обходу цієї вимоги на зазвичай використовують серію зображень, з якої обчислюють ймовірність належності кожного пікселя до фону за розподілом за Гаусом від середнього кольору фону і коваріаційної матриці конкретного пікселю і часу. Тобто, невеликі зміни у зоні з низьким рівнем шуму сприйматимуться краще аніж набагато більші відмінності у зоні з високим шумом, що дозволяє брати до уваги лише появу значимих об’єктів. Недоліком цього методу є те що, в області високого шуму, значення коваріаційної матриці є високими, тож у разі зміни зображення в області високого шуму, ця зміна може бути проігнорована алгоритмом.

Цей алгоритм, як і базове видалення фону, є простим і все ще потребує кількох зображень для визначення помітного об’єкту, але спектр застосування вже включає у себе виділення об’єктів відносно фону.

Метод визначення помітних об’єктів за Рахту та Хеіккіла.

Цей метод використовує кількісні вимірювання помітності. Припускаємо що у нас є прямокутна рамка, яка поділена на внутрішню частину та границю. Для внутрішньої точки xx береться значення інтенсивності кольору F(xx), хоча метод припускає можливість роботи з градієнтом та текстурою. Висуваються два припущення: H0: «точка xx не є помітною», та H1: «точка xx є помітною». Та позначаються ймовірності кожної гіпотези як P(H0) та P(H1). Після цього робиться початкове припущення що H0 дійсне для усіх точок у граничній рамці, а H1 для усіх точок внутрішньої частини. За допомогою теореми Баєса можна визначити що $P\big(H_1|F(x)\big) = \frac{P(F(x)|H_1)P(H_1)}{P\big(F(x)\big)}$, з чого виводиться те що $S(x) = P\big(H_1|F(x)\big)$, що відповідає контрасту між внутрішньою частиною та граничною рамкою.

Основною особливістю цього методу являється те що помітність визначається відразу для зони, а не попіксельно, і відповідно відпадає необхідність параметричних обчислень контрасту для кожного пікселя. Це підвищує його ресурсоефективність в порівнянні з результативними алгоритмами що використовують машинне навчання, а також, на відміну від більшості традиційних алгоритмів, даний алгоритм не потребує кількох зображень для визначення окремого об’єкта.

Метод моделювання Контекстових Гіперграфів.

Використовує машинне навчання для виділення з зображення складових моделюючих пікселів (суперпікселів) що представляють зображення у вигляді гіперграфа, що складається з цих моделюючих пікселів. Моделюючі пікселі визначаються контекстом замість значення кожного з них.

Через це, проблема помітності зводиться до визначення помітних вершин та гіперребер у гіперграфі на багатьох осях. Підхід контрасту центр-оточення формулюється як задача класифікації ціночутливої макс-маржі. Як наслідок, ступінь помітності регіону зображення вимірюється за його пов'язаною нормалізованою довжиною опорного вектора.

Виявлення рельєфу типово поставлене як проблема контекстуального аналізу контрасту центру проти оточення. Для вирішення цієї проблеми у даному методі використовується метод виявлення помітності, заснований на незбалансованому максимальномаржовому навчанні, який здатний ефективно виявляти локальні помітні регіони зображення, які значно відрізняються від оточуючих регіонів. У цьому випадку зображення розбивається на прямокутні вікна, що перетинаються, які в свою чергу перевіряються на помітність. Контекстом для кожного вікна є вікна, які його перекривають.

Основу навчання з максимальною маржею можна легко розширити для виявлення помітності в глобальному масштабі. А саме,
межі зображення можна розглядати як негативні зразки
в той час як інші плями зображення використовуються як позитивні
зразки. За допомогою процедури навчання з максимальною маржею
над такими навчальними вибірками ступінь помітності кожного фрагмента зображення можна виміряти шляхом обчислення його відстані до роздільної гіперплощини.

Хоча карти рельєфу точно локалізують характерний об'єкт у кожному випадку, вони також страждають від нечіткості навколо меж об’єкта та локально однорідних регіонах. В основному це пов’язано з локальним контекстом центру, на якому вони базуються. Ключовий підхід методу гіперграфів, заснований на контексті на основі сегментації, пом’якшує ці проблеми.

Щоб ефективніше знаходити регіони помітних об’єктів, даний метод формує контексти суперпікселів для фіксування як внутрішньої консистенції, так зовнішній поділ. На відміну від парного
ребра в стандартному графі, гіперребро в гіперграфі
– це ребро високого порядку, пов’язане з вершинною клікою, що зв’язує більше двох вершин. Ефективне конструювання таких
гіперребер є вирішальним для кодування внутрішнього контексту інформації про вершини в гіперграфі.

Цей метод є ефективним через використання машинного навчання для розпізнавання сегментів зображення та подальшої побудови контекстного гіперграфа без необхідності залучати глибоке машинне навчання, яке може зайняти багато часу, та вимагало б бази зображень для тренування.

Визначення помітних об’єктів за допомогою глибокого навчання згорткових нейронних мереж

Надзвичайно результативний метод, але вимагає від розробників мати великий набір зображень для тренування мережі, та велику кількість ресурсів. Можливе тренування готових мереж на невеликому датасеті для більш вузької спеціалізації конкретної мережі, що включає у себе визначення помітних зображень, проте це все одно вимагає підготовки та класифікації матеріалів до того як можна було б почати тренування.

Висновок розділу

Найбільш доцільним для даної роботи виглядає метод гіперграфів через одночасно свою простоту і гнучкість застосування без необхідності використання набору зображень фону з основним об’єктом і без, та високу ефективність в порівнянні з методами без машинного навчання.

РОЗДІЛ 2 ПОРІВНЯННЯ ТА ВИБІР ТЕХНОЛОГІЙ

Наразі для обробки зображень була створена велика кількість бібліотек з широким набором функцій. Основними вимогами для інструментарію обробки зображень є швидкодія, кросплатформеність і універсальність.

Torch

Torch представляє собою бібліотеку машинного навчання та наукових обчислень. Вона дозволяє у стислі терміни будувати нейронні мережі для багатьох різних цілей. В якості основних мов програмування на Torch виступають Lua і Python. Є часткова підтримка C++, але вона знаходиться у стані розробки.

CUDA

CUDA (від англ. Compute Unified Device Architecture) –

Технологія CUDA з'явилася в 2006 році і є програмно-апаратним комплексом виробництва компанії Nvidia, що дозволяє ефективно писати програми під графічні адаптери. З 2006 року компанія Nvidia обіцяє, що всі графічні адаптери їхнього виробництва незалежно від серії матимуть подібну архітектуру, яка повністю підтримує програмну частину технології CUDA. Програмна частина, у свою чергу, містить у собі все необхідне для розробки програми: розширення мови C/C++, компілятор, API для роботи з графічними адаптерами та набір бібліотек. Щоб зрозуміти, чим технологія CUDA відрізняється від інших існуючих технологій розпаралелювання, треба зрозуміти, чим стандартні багатопроцесорні системи відрізняються від графічних.

Більшість площі обчислювального кристала займає кеш-пам'ять, а обчислювальні модулі (ALU) займають лише чверть кристала. Крім того, кожен окремий ALU є повноцінним центральним процесором. Він здатний підтримувати всі апаратні переривання, може працювати з усіма пристроями

вводу/виводу, що, є корисним для його функціонування як центрального процесора, однак стає зайвим для використання як векторного обчислювального модуля. Крім того, потоки, що виконуються на центральному процесорі, є дуже обчислювально важкими, ними управляє операційна система, тому їх не може бути багато.

Потужна бібліотека для виконання обчислень на однойменних графічних ядрах відеокарт Nvidia. Враховуючи архітектуру ядер CUDA, це дозволяє використовувати паралельні обчислення, що добре підходить для машинного навчання.

Щодо графічних процесорів, більшість площі обчислювального кристала зайнята обчислювальними пристроями. Проща що виділена під кеш і пристрої управління дуже мала. Процесори на GPU набагато простіші, ніж ядра CPU, вони можуть виконувати лише математичні операції і не здатні до самостійної діяльності (є співпроцесорами до центрального процесора).

Програмна частина технології CUDA

Введемо основні терміни та відносини між ними

- Хост (Host) – центральний процесор, що управляє виконанням програми.
- Пристрій (Device) — відеоадаптер, який є співпроцесором центрального процесора.
- Грид (Grid) – об'єднання блоків, які виконуються на одному пристрої.
- Блок (Block) – об'єднання тредів, яке виконується повністю на одному SM. Має свій унікальний ідентифікатор усередині гриду.
- Тред (Thread, потік) – одиниця виконання програми. Має унікальний ідентифікатор всередині блоку.
- Варп (Warp) - 32 послідовних треда, виконується фізично одночасно.
- Ядро (Kernel) - паралельна частина алгоритму, що виконується на гриді.

На центральному процесорі (хості) виконуються лише послідовні частини алгоритму програми, підготовка та копіювання даних на пристрій, завдання

параметрів для ядра та його запуск. Паралельні частини алгоритму оформляються в ядра, які виконуються на великій кількості тредів на пристрої. Для реалізації програми під GPU компанія NVIDIA випустила свої розширення для мови C, компілятор NVCC для збирання таких програм, узвичаїла нове розширення *.cu для файлів, які містять CUDA виклики. До розширень мови C відносяться:

- специфікатори для функцій та змінних,
- нові вбудовані типи даних,
- вбудовані змінні (всередині ядра),
- директива для запуску ядра з коду C.

Оскільки не усі комп'ютери мають графічні процесори Nvidia, це накладає певні обмеження для використання на різних апаратних платформах.

VIGRA

VIGRA – "Vision with Generic Algorithms", містить у собі багато загальних алгоритмів комп'ютерного бачення. Ця бібліотека добре підходить для методів, які працюють з високовимірними зображеннями, такими як суперпікселі та фільтри. Є підтримка багатопоточного накладання фільтрів, визначення країв. Основна мова для використання бібліотеки – C++. Є підтримка Python.

OpenCV

OpenCV – бібліотека комп'ютерного зору з дуже широким вибором інструментів. Бібліотека кросплатформена, містить у собі функції роботи з суперпікселями, сегментацією зображення, а також власну бібліотеку для роботи з машинним навчанням, яка підтримує опорні вектори.

OpenCV випущено під ліцензією BSD і, отже, є безкоштовною як для академічного, так і для комерційного використання. Вона має інтерфейси C++, C, Python і Java і підтримує Windows, Linux, Mac OS, iOS і Android. Коли OpenCV була розроблена, основним акцентом були додатки реального

часу для обчислювальної ефективності. Усе написано на оптимізованому C/C++, щоб користуватися перевагами багатоядерної обробки. Підтримуються програмні інтерфейси для C++, Python, MATLAB, Java. Існує багато задач, які вирішуються за допомогою OpenCV:

- Розпізнавання обличчя
- Автоматизований контроль і нагляд
- кількість людей – кількість (пішохідний рух у торговому центрі тощо)
- Транспортний засіб розраховує на шосе разом зі своєю швидкістю
- Інтерактивні арт-інсталяції
- Виявлення анамолій (дефектів) у процесі виробництва
- Зшивання зображень перегляду вулиць
- Пошук і отримання відео та зображень
- Автомобільна навігація та керування без водія
- розпізнавання об'єкта
- Аналіз медичного зображення
- 3D структура з руху
- Розпізнавання реклами на телеканалах

Функціональність OpenCV

- Виявлення об'єктів/функцій (objdetect, features2d, невільні)
- Монокулярний або стереокомп'ютерний зір на основі геометрії (calib3d, зшивання, відеостаб)
- Комп'ютерна фотозйомка (фото, відео, superres)
- Машинне навчання та кластеризація (ml, flann)
- Прискорення CUDA (GPU)

Вибір мови програмування

У більшості випадків для роботи з зображеннями та машинним навчанням використовуються мови високого рівня, такі як Python і Java. Це надає більше зручності при проектуванні методів, але при цьому нижча швидкодія. Враховуючи вимогу до швидкості обробки зображення, для створення

застосунку було обрано мову програмування C++, яка є мовою програмування високого рівня, але при цьому має високу швидкодію.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

Виходячи з попередніх розділів, нами було вирішено використати метод контекстових гіперграфів та реалізувати його за допомогою бібліотеки OpenCV.

З даної бібліотеки були використані такі інструменти:

Засоби поділення зображення на суперпікселі

Наявно три класи що відповідають за роботу з суперпікселями, усі три знаходяться у модулі ximgproc, що не входить у стандартний, завчасно скомпільований набір, тому для його використання бібліотека була побудована з сирцевих кодів.

SuperpixelLSC – суперпікселізація за лінійним спектральним кластерінгом. Докладний опис алгоритму наведено у джерелі 1.

Перелік функцій:

enforceLabelConnectivity(int min_element_size = 25) – відповідає за злиття суперпікселів які занадто малі.

Параметр «min_element_size» відповідає за мінімальний розмір елемента у відсотках, який слід поглинути в більший суперпіксель. Враховуючи отримане значення середнього розміру суперпікселя, дійсне значення має бути в діапазоні 0-100, при чому 25 означає, що суперпікселі менші за чверть мають бути поглинуті. Це, також, є значенням за замовчуванням для цього параметру.

Функція зливає між собою компоненти, які занадто малі, та призначає раніше знайдену суміжну мітку цьому компоненту. Виклик цієї функції може змінити остаточну кількість суперпікселів у меншу сторону.

getLabelContourMask(image, bool thick_line)

Повертає маску сегментації контурів суперпікселів що міститься у об’єкті SuperpixelLSC у масив image.

getLabels(labels_out)
Повертає сегментаційні мітки суперпікселя, де одна мітка – один суперпіксель, а кожен піксель відповідає призначеній мітці суперпікселя. Мітки можуть бути записані у будь-який числовий масив, що відповідає попіксельному розміру оригінального зображення.

getNumberOfSuperpixels()
Повертає значення суперпікселів після сегментації.

iterate(int num_iterations = 10)
Обчислює сегментацію суперпікселів на зображенні. Дана функція може бути викликана повторно без необхідності переініціалізації алгоритму за допомогою createSuperpixelLSC(). Це дозволяє зберегти ресурси при виділенні пам’яті, в тому числі час на завантаження/вивантаження об’ємних зображень.
Дана функція приймає параметр num_iterations в якості числа ітерацій алгоритму розбиття на суперпікселі. Більша кількість ітерацій, як правило, покращує результат.
Функція обчислює суперпіксельну сегментацію зображення з параметрами, ініціалізованими функцією createSuperpixelLSC(). Алгоритми починає з сітки суперпікселів, а потім уточнює межі, пропонуючи оновлення меж ребер.

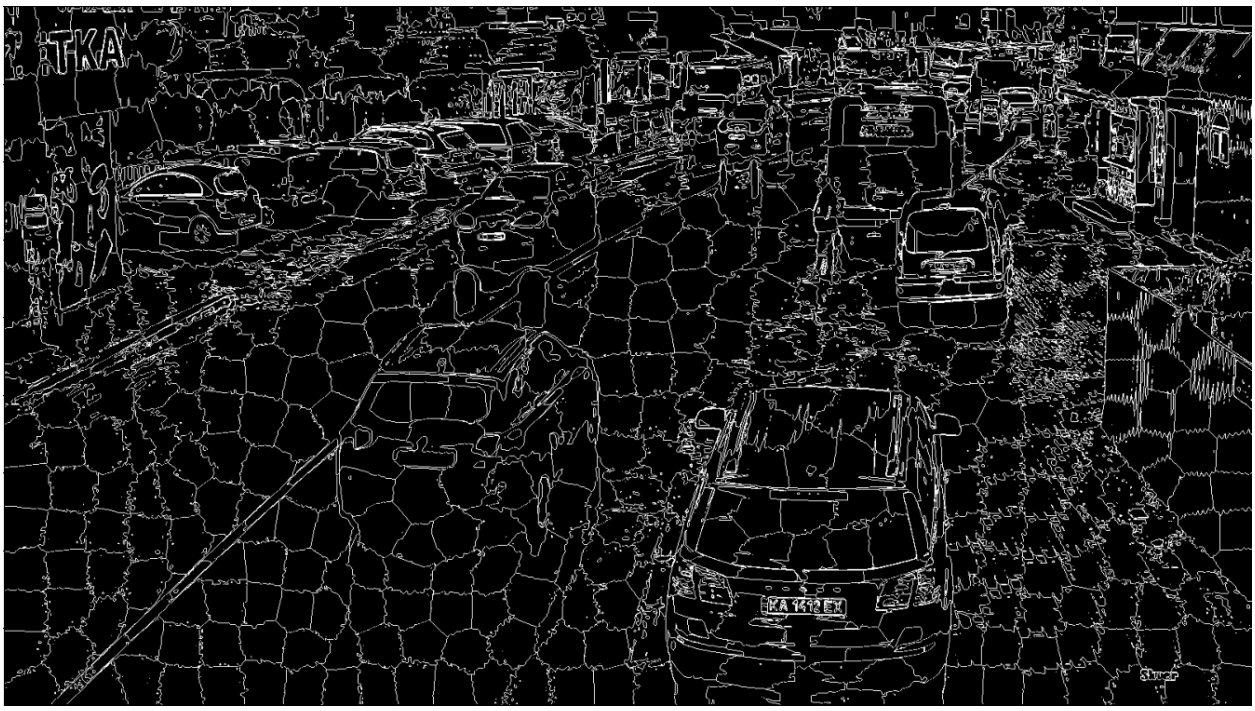

Рис. 2, Маска контурів алгоритму SuperpixelLSC з середнім розміром суперпікселя 50 пікселів.

Рис. 3 Маска контурів алгоритму SuperpixelLSC з середнім розміром суперпікселя 10 пікселів.

SuperpixelSEEDS – Суперпікселі видобуті за допомогою вибірки що керується енергійністю.

Ініціалізується конструктором createSuperpixelSEEDS().

В якості параметрів приймає такі дані:

image_width Ширина зображення.

image_height Висота зображення.

image_channels Кількість каналів зображення.

num_superpixels Бажана кількість суперпікселів. Справжня кількість може бути меншою і буде залежати від параметру num_levels та розміру зображення, що ділиться. Для отримання справжньої кількості суперпікселів, потрібно застосувати метод getNumberOfSuperpixels().

num_levels Кількість рівнів блоків. З більшою кількістю блоків підвищується точність сегментації, але вимагає більше пам’яті та процесорного часу.

prior Якщо даний параметр більше 0, то застосовує згладжування формою 3*3. Більше значення призведе до більш гладких форм, але не має бути вище 5.

histogram_bins Кількість використовуваних наборів гістограм.

double_step Якщо дорівнює True, то кожний блок ітерується двічі для вищої точності.

Функція ініціалізує об’єкт SuperpixelSEEDS для вхідного зображення. У ній зберігаються параметри зображення: image_width, image_height та image_channels. Вона також встановлює параметри алгоритму, а саме: num_superpixels, num_levels, use_prior, histogram_bins і double_step.

Кількість рівнів у num_levels визначає кількість рівнів блоків, які алгоритм використовує при оптимізації. Ініціалізація — це сітка, в якій суперпікселі рівномірно розподілені по ширині та висоті зображення. Більші блоки відповідають розміру суперпікселя, а рівні з меншими блоками формуються шляхом поділу більших блоків на 2 x 2 блоки пікселів, рекурсивно до меншого рівня блоку. Приклад ініціалізації 4 рівнів блоку проілюстрований на наступному рисунку.

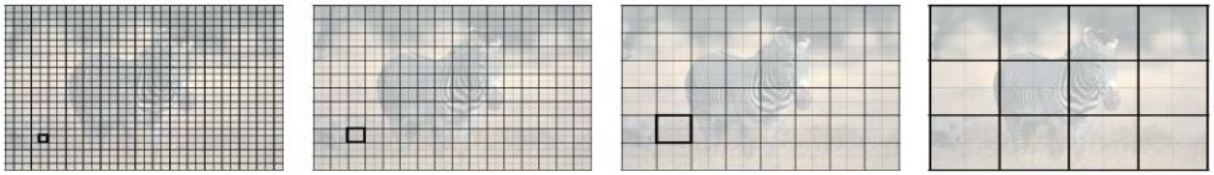

Рис. 4

Об’єкт має у собі функції:

getLabelContourMask()

getLabels()

getNumberOfSuperpixels()

iterate()

Ці функції мають призначення ідентичне до подібних функцій попереднього алгоритму.

SuperpixelSLIC – алгоритм простого лінійного ітеративного кластерування. Докладно описаний у джерелі 3.

Конструктор об’єкту – і

Параметри:

image Зображення що буде просегментоване.

algorithm Вибір варіанту алгоритму для використання: SLIC сегментує зображення, використовуючи бажаний region_size. SLICO оптимізує процес за допомогою адаптивного коефіцієнта компактності, тоді як MSLIC оптимізує за допомогою використання методу багатогранних з’єднань, що призведе до більш чутливих до вмісту суперпікселів. Це є різними оптимізаціями одного алгоритму

region_size Обирає бажаний середній розмір суперпікселя виражений у кількості пікселів.

ruler Визначає ступіль зглажування суперпікселів.

Об’єкт має у собі функції:

getLabelContourMask()

getLabels()

getNumberOfSuperpixels()

iterate()

Ці функції мають призначення ідентичне до подібних функцій попередніх двох алгоритмів.

Було обрано алгоритм SLIC через його оптимізацію MSLIC яка підвищує якість сегментації.

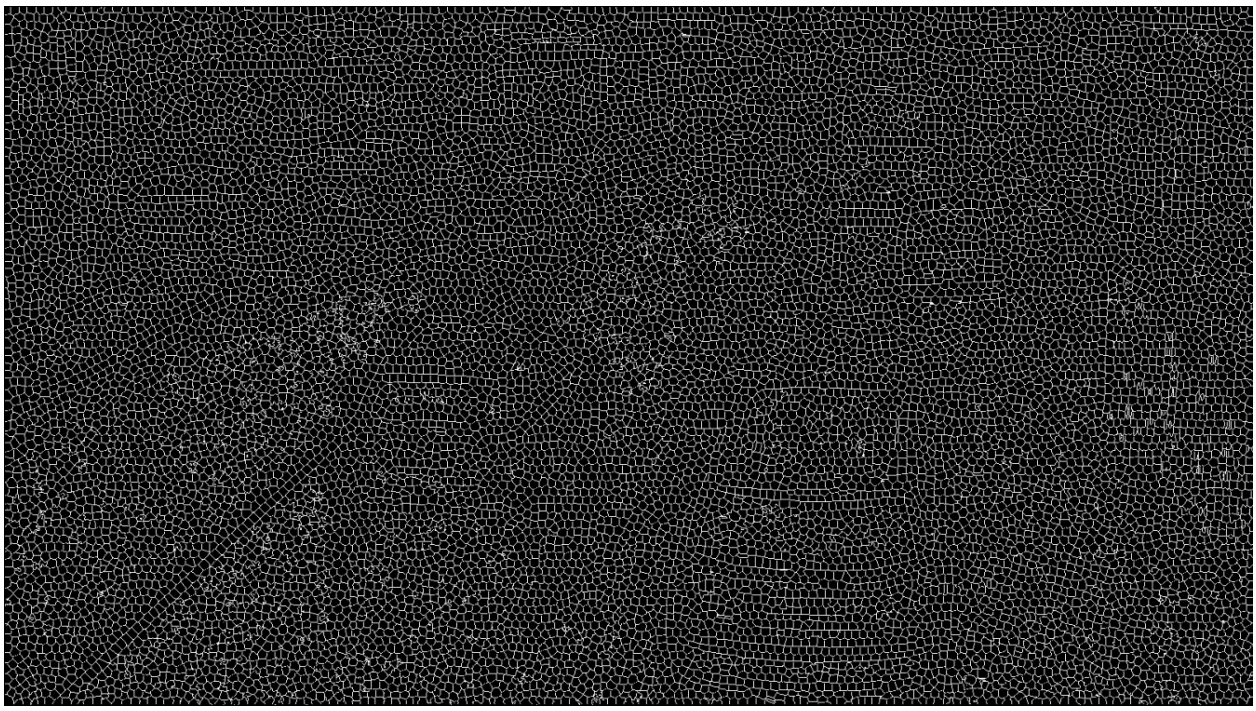

Рис. 5 Маска контурів функції SuperpixelISLIC з середнім розміром суперпікселя 10 пікселів із застосуванням алгоритму SLICO.

Рис. 6 Маска контурів функції SuperpixelISLIC з середнім розміром суперпікселя 10 пікселів із застосуванням алгоритму SLIC.

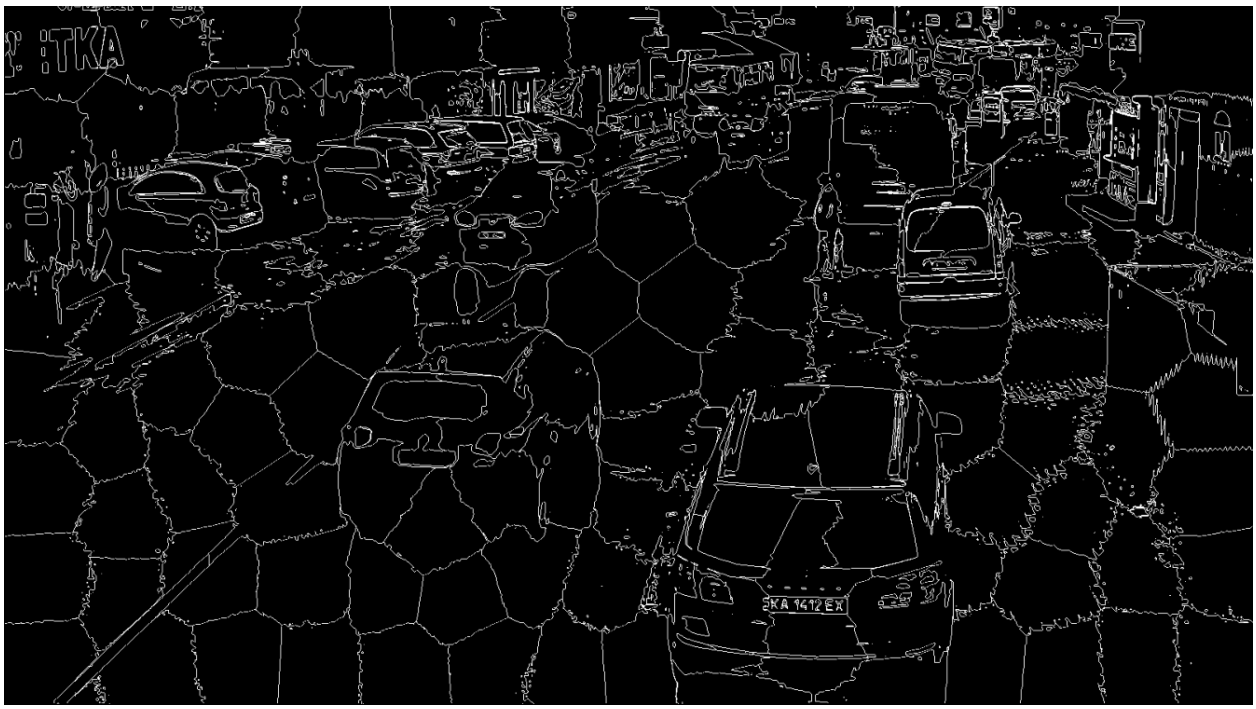

Рис. 7 Маска контурів функції SuperpixelISLIC з середнім розміром суперпікселя 100 пікселів із застосуванням алгоритму SLIC.

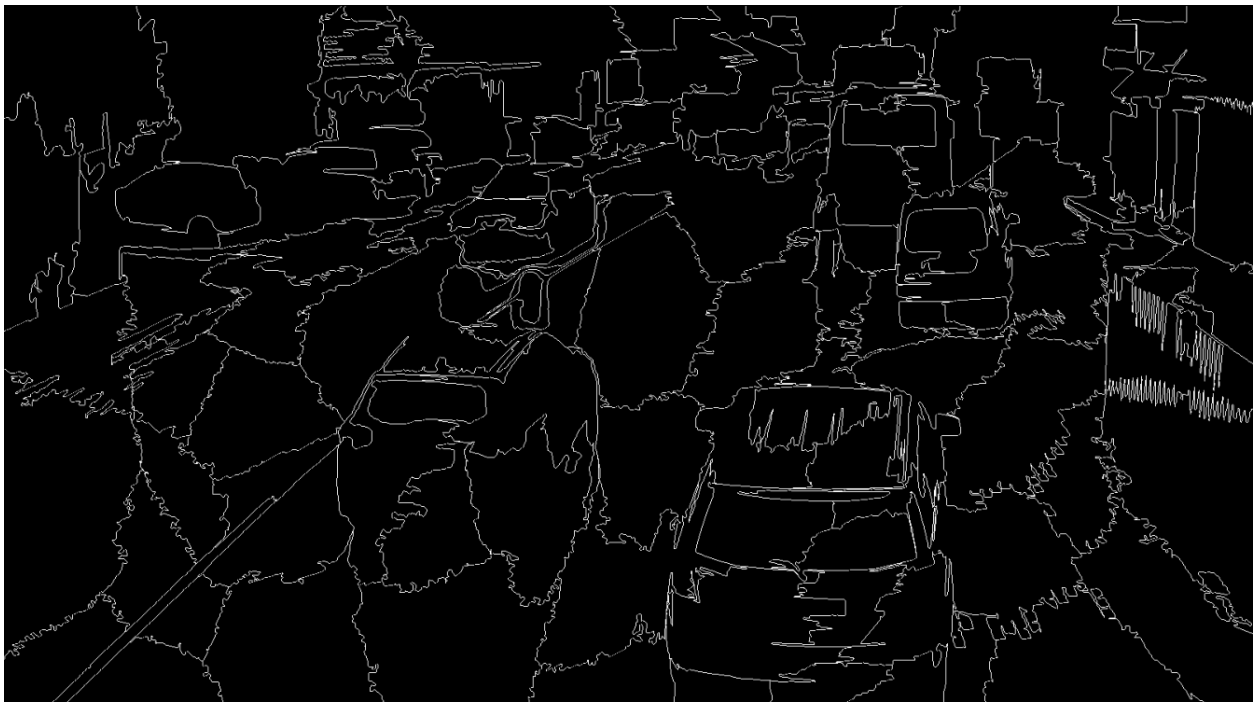

Рис. 8 Маска контурів функції SuperpixelISLIC з середнім розміром суперпікселя 200 пікселів із застосуванням алгоритму MSLIC.

Визначення градієнту зображення

Одним з етапів побудови гіперграфів є визначення найбільш помітних частин зображення. Це в описаному методі досягається шляхом виділення граней об’єктів, для чого і використовуються градієнти.

При використанні бібліотеки OpenCV рекомендуються такі методи визначення градієнтів:

Оператор Собеля.

Оператор Прюітт (Scharr operator).

Оператор Лапласа.

Оператор Собеля:

Це дискретний диференціальний оператор, що обчислює наближене значення градієнта чи норми градієнта для яскравості зображення. Оператор Собеля базується на згортці зображення невеликими сепарабельними цілочисельними фільтрами в вертикальному та горизонтальному напрямках. Оскільки функція інтенсивності цифрового зображення відома лише в дискретних точках, похідні цієї функції не можуть бути визначені, якщо ми не припустимо, що існує базова функція диференційованої інтенсивності, яка була відібрана в точках зображення. З деякими додатковими припущеннями, похідну функції безперервної інтенсивності можна обчислити як функцію від вибіркової функції інтенсивності, тобто цифрового зображення. Виявляється, що похідні в будь-якій конкретній точці є функціями значень інтенсивності практично у всіх точках зображення. Однак наближення цих похідних функцій можна визначити з меншим або більшим ступенем точності.

Оператор Собеля-Фельдмана являє собою досить неточну апроксимацію градієнта зображення, але все ще має достатню якість, щоб бути практичним у багатьох програмах. Точніше, він використовує значення інтенсивності лише в області 3×3 навколо кожної точки зображення для апроксимації відповідного градієнта зображення, а також використовує лише цілі значення

для коефіцієнтів, які зважують інтенсивність зображення для отримання наближення градієнта.

Рис. 9, Горизонтальний оператор Собеля

Рис. 10, Вертикальний оператор Собеля

Оператор Прюітт – оператор обчислює градієнт інтенсивності зображення в кожній точці, вказуючи напрямок найбільшого можливого збільшення від

світлого до темного і швидкість зміни в цьому напрямку. Таким чином, результат показує, наскільки «раптово» або «плавно» змінюється зображення в цій точці, і, отже, наскільки ймовірно, що частина зображення представляє край, а також як ця грань, ймовірно, буде орієнтована. На практиці розрахунок величини (ймовірності краю) є більш надійним і легшим для інтерпретації, ніж розрахунок напрямку.

Математично градієнт функції з двома змінними (тут функція інтенсивності зображення) є в кожній точці зображення двовимірним вектором з компонентами, заданими похідними в горизонтальному та вертикальному напрямках. У кожній точці зображення вектор градієнта вказує в напрямку найбільшого можливого збільшення інтенсивності, а довжина вектора градієнта відповідає швидкості зміни в цьому напрямку. Це означає, що результат дії оператора Прюітта в точці зображення, яка знаходиться в області постійної інтенсивності зображення, є нульовим вектором, а в точці на краю — вектором, який вказує поперек краю, від темніших значень до більш яскравих.

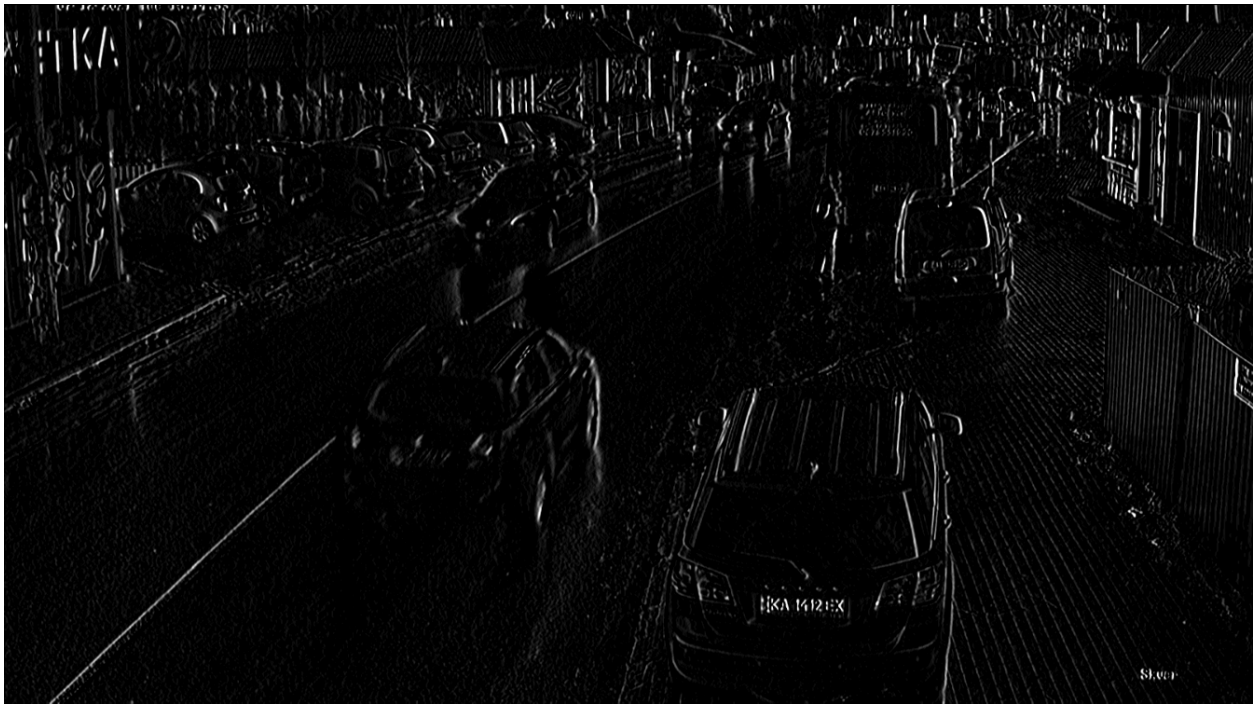

Рис. 11, Оператор Прюітта

Оператор Лапласа – диференціальний оператор, заданий розбіжністю градієнта скалярної функції на евклідовому просторі. Лапласіан задається сумою других часткових похідних функції щодо кожної незалежної змінної.

Рис. 12, Застосування оператора Лапласа до зображення з розмиттям за Гаусом.

Замість цього можна використати вбудований алгоритм Канні, що дозволяє більш точно визначити грані об’єктів.

Рис. 13, визначення граней за методом Канні.

ВИСНОВКИ

Аналіз існуючих методів показав низьку кількість методів визначення основного об’єкта для статичного зображення.

Був вибраний ефективний метод визначення чіткого основного об’єкта, та розібрані способи його реалізації.

Була використана сучасна бібліотека для роботи з зображеннями OpenCV, яка дозволяє використовувати широкий набір інструментів для реалізації вибраного методу.

Деякі частини методу були замінені на еквівалентні конструкції з обраної бібліотеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Superpixel Segmentation Using Linear Spectral Clustering. https://openaccess.thecvf.com/content_cvpr_2015/html/Li_Superpixel_Segmentation_Using_2015_CVPR_paper.html
2. Superpixels extracted via energy-driven sampling. https://www.researchgate.net/publication/230815785_SEEDS_Superpixels_Extracted_via_Energy-Driven_Sampling
3. Slic superpixels compared to state-of-the-art superpixel methods. https://dx.doi.org/10.1109/TPAMI.2012.120
4. Comparative study of background subtraction algorithms – https://www.spiedigitallibrary.org/journals/journal-of-electronic-imaging/volume-19/issue-3/033003/Comparative-study-of-background-subtraction-algorithms/10.1117/1.3456695.full?SSO=1
5. A Simple and efficient saliency detector for background subtraction – https://ieeexplore.ieee.org/abstract/document/5457577
6. Contextual Hypergraph Modeling for Salient Object Detection – https://cs.adelaide.edu.au/~yaoli/?page_id=149