Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Використання методу Лукаса-Канаде для аналізу руху та відстеження об'єктів

Текстова частина до курсової роботи
за спеціальністю «Комп’ютерні науки» 122

Керівник курсової роботи
ст. викл. Бучко О.А.

_____

(*підпис*)

“____” _____ 2022 р.

Виконав студент БП КН-3
Федоров Д.М.

“____” _____ 2022 р.

Календарний план виконання роботи

Тема: «Використання методу Лукаса-Канаде для аналізу руху та відстеження об’єктів»

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	23.10.2021	
2.	Пошук та огляд літератури за темою роботи.	20.12.2021	
3.	Реалізація програми з мінімальним функціоналом.	20.01.2022	
4.	Пошук альтернативних алгоритмів та способи покращення існуючого.	25.02.2022	
5.	Покращення роботи алгоритму.	10.04.2022	
6.	Пошук інших досліджень щодо алгоритму, їхнє вивчення.	10.05.2022	
7.	Написання теоретичної частини роботи.	31.05.2022	

ЗМІСТ

Вступ 4
Розділ 1. КОМП'ЮТЕРНИЙ ЗІР ТА АНАЛІЗ ЙОГО ВИКОРИСТАННЯ 6
1.1 Загальні відомості 6
1.2 Використання комп'ютерного зору 7
1.2.1 Медицина 7
1.2.2 Промисловість 7
1.2.3 Військова справа 8
1.2.4. Автономні транспортні засоби 8
1.2.5 Tactile feedback 9
1.2.6 Розпізнавання 9
1.2.7 Аналіз руху 10
1.2.8 Реконструкція сцен та відновлення зображень 11
Розділ 2. ОПТИЧНИЙ ПОТІК ТА ЙОГО ТИПИ 12
2.1 Загальні відомості 12
2.2 Оптичні потоки: Sparse і Dense 13
2.3 Нівеляція яскравості оптичного потоку 14
2.4 Градієнтний компонент потоку 16
2.4.1 Загальні відомості 16
2.4.2 Проблема отвору 17
2.4.3 Розв'язання проблеми отвору. Застосування формули апроксимації менших квадратів Брюсом Лукасом та Такео Канаде 19
2.4.5 Поєднання локальних обмежень. Робота в RGB-просторі 21
Розділ 3. АЛГОРИТМ ЛУКАСА-КАНАДЕ 23
3.1 Загальні відомості 23
3.2 Відстеження ознак за алгоритмом Лукаса-Канаде 24
3.3 Похибки та покращення алгоритму 24
Розділ 4. РЕАЛІЗАЦІЯ АЛГОРИТМУ ЛУКАСА-КАНАДЕ 26
Висновки 31
Список літератури 33
Додаток А (обов'язковий) Програма алгоритму Лукаса-Канаде 34
Додаток Б (обов'язковий) Приклад роботи програми 36

Вступ

Сьогодення сповнене комп’ютеризації та автоматизації всього навколишнього для людини: від виробництва маленьких деталей для годинників, де роботи давно замінили ручну працю, до розпізнавання облич та аналізу руху машин на жвавому шосе. Нерідко в суспільства постає питання, як же комп’ютер може робити припущення про рух об’єктів у космосі, коли все, що бачить машина крізь свій об’єктив – це пікселі без семантики.

Людське око має здатність фокусуватися виключно на одному об’єкті чи на сукупності менших за розміром, але схожих за типом, об’єктів. У цьому є недолік взаємодії органічного мозку зі звичайним зором: неможливість охопити всю картинку й проаналізувати її. У цьому природа поступається комп’ютерам: їхній зір здатний аналізувати будь-який мінімально рухомий чи необхідний для аналізу об’єкт чи їхню сукупність, незважаючи на відстань між ними чи можливу похибку зображення. До того ж, навчений комп’ютер зможе чітко розрізняти об’єкти між собою й виявляти виключно ті, що задані йому для аналізу.

Таким чином, стає явною частина переваг використання комп’ютерного зору в порівнянні з людським. Очевидно, нічого в світі не буває ідеальним, тому й машини мають свої недоліки, які будуть детально обговорені в цій роботі.

Даний текст буде присвячений тематиці комп’ютерного зору. Тут будуть досліджені його загальні принципи й використання в різних сферах людського життя; сконструйований базис для того, аби поглибитися в більш конкретні й детальні алгоритми подолання похибок та нових проблем.

З-поміж всього, у роботі досліджено поняття «оптичного потоку» та суміжних до нього, разом із прикладами конкретних рішень, що його використовують. Буде розібране питання чому напрямок руху в тривимірному просторі можна аналізувати як двовимірний вектор, а також як працювати з точками, що змінюють своє положення не тільки в просторі, а й в часі.

Проаналізувавши вище зазначений базис, буде продемонстровано конкретику та на прикладі алгоритму Лукаса-Канаде розібрано, як саме комп’ютер аналізує все, що бачить, і чому для цього йому не обов’язково заздалегідь знати семантику об’єкта.

Після цього буде розглянуто математику Алгоритму та її спрощення, побудовано функції, якими він послуговується.

Як наслідок, будуть зроблені висновки щодо роботи Алгоритму в цілому та виявлені його недоліки й переваги. Також будуть визначені вплив розміру обраного «вікна» пікселів на точність аналізу; нівеляція яскравості та ускладнення процесу від використання RGB-відео. Наостанок буде розглянуто вплив плавного та різкого руху на результат і як поводить себе Алгоритм на об’єктах, які не рухаються.

В останньому розділі 4 описано наявні рішення реалізації алгоритму Лукаса-Канаде в сучасних мовах програмування на прикладі бібліотеки OpenCV.

Розділ 1. КОМП'ЮТЕРНИЙ ЗІР ТА АНАЛІЗ ЙОГО ВИКОРИСТАННЯ

1.1 Загальні відомості

«Computer vision is the enterprise of automating and integrating a wide range of processes and representations used for vision perception» [1].

Тобто, іншими словами – це технологія, яка в сучасності стала повноцінною сферою розвитку в комп'ютерній інженерії та робототехніці. Основними її завданнями слід назвати спроби зрозуміти способи та навчитися аналізувати зображення та відео.

Технологія комп'ютерного зору була представлена на початку 1970-х років, аби роботи мали змогу мімічно відтворювати поведінку людини й таким чином, були б наділені інтелектуальною поведінкою [2]. У ті ж роки були винайдені більшість базових алгоритмів, серед яких:

- Визначення границь зображення – Canny edge detector, thresholding та зокрема алгоритм для знаходження границь Shi-Tomasi;
- Обробка більших зображень як колекції менших за розмірами об'єктів;
- Оптичний потік;
- Аналіз руху за рахунок векторів руху та багато інших.

З часом, розвиток комп'ютерного зору перестав обмежуватися двовимірними ресурсами й з'явилася потреба аналізувати тривимірні моделі, базис для чого тоді лише з'являвся й поступово розвивався. Такі техніки також мали покращити розвиток робототехніки, адже від інтелектуального аналізу побаченого, комп'ютер може мати змогу робити більш точні й детальніші припущення. Паралельно розвивалася ідея використання математичного аналізу, концепти якого могли бути застосовані в аналізі зображень.

Розвиток комп'ютерного зору також позитивно вплинув і на фотографію, адже дані технології ділять схожі оптимізаційні й стабілізаційні методи.

Зрештою в 1990-х з'явилася можливість відтворення 3D-об'єкту шляхом співставлення кількох зображень. Це допомогло, зокрема, відтворювати зруйновані об'єкти та покращило конструювання та проектування архітектури. Розвинулися техніки перетворення зображень, рендерингу на основі зображень, створення панорамних високоякісних зображень з багатьох фотографій [2], які зараз широко використовуються, наприклад, в Google Maps.

В сучасному світі комп'ютерний зір тісно поєднаний з розвитком машинного та глибокого навчання, що використовують дані, надані

результатами роботи алгоритмів такого зору. Зокрема вони використовуються в задачах класифікації та сегментації. [3]

1.2 Використання комп’ютерного зору

У сучасному світі розвиток комп’ютерного зору досяг високих показників і заполонив багато сфер діяльності людини: від медицини – до військової справи, а також досі продовжує розвиватися, маючи як підґрунтя машинне навчання та сучасні принципи побудови технологічних об’єктів – машин, камер та інших.

1.2.1 Медицина

Медицина є однією з найбільш впливових та розвинутих сфер застосунку для комп’ютерного зору. Алгоритми машини тісно пов’язані з глибоким навчанням, і оброблюючи величезні об’єми інформації та зображень, комп’ютер може покращувати точність та обробку матеріалів, що широко використовуються в медицині. Зокрема, комп’ютерний зір використовується для досягнення таких цілей:

- Виявлення пухлин та раку;
- Обробка медичних зображень – рентген-знімків, МРТ та інших – зменшення шуму, поліпшення контурів та виявлення аномалій;
- Допомога в діагностуванні та вчасному виявленні захворювання у пацієнта;
- Покращує вивчення роботи та структури мозку й решти органів людини та інше.[4]

1.2.2 Промисловість

Нерідко можна почути, що роботи замінюють людей на робочих місцях. Це дійсно так, але слід пам’ятати, що не всюди комп’ютер – замінник людини.

Однак, така теза точно не стосується автоматизованого процесу виготовлення деталей згаданих у вступі годинників, чи надзвичайної точності, необхідної при створенні літаків, де помилка людини може бути занадто дороговартісною.

У промисловості комп’ютерний зір використовується для того, щоб навіть автоматичні роботи мали змогу ідентифікувати позицію деталей, які вони складають докупи, чи піднімають, чи переставляють на складах.

Загальновідомою є проблема: як же відділити крупу булгура від кускуса й від проса, коли вони раптом опинилися в одній банці?

Життя, звичайно, вчить розділяти крупи, але в сучасній агропромисловості розповсюдженим є метод оптичного сортування. Цей метод базується на комп'ютерному зорі, який вміє розділяти розмір, колір, форму та інші характеристики об'єкту, що дає змогу аграріям ідентифікувати непридатні до вжитку культури й овочі, забираючи їх із продукції, а також визначати ступінь зрілості вирощуваного продукту.

1.2.3 Військова справа

Нарівні з медициною, військова галузь займає провідне місце серед сфер застосування комп'ютерного зору. Війна вже давно не мала б вестися у світі, а тим паче в демократичних країнах, які мають сміливість себе такими називати, але бойові дії досі тривають.

З розвитком технологій та зброї, все більшого впливу набувають ракети. Прогрес відходить від історично закладених систем вогню, що орієнтуються на враження цілей в певній зоні, а все більше переходить до керованих засобів, якими зручніше й надійніше управляти.

Алгоритми комп'ютерного зору допомагають в наведенні на цілі, зменшуючи шум зображення, покращуючи роботу сенсорів та екранів. Вони також допомагають безпосередньо точно наводитися на ціль ураження, ізолюючи контури об'єкта від деталей, що їх приховують.

Таким чином, навіть загальновідомий Bayraktar – безпілотник вищого класу, що продемонстрував свої здібності в російсько-українській війні, користується технологіями штучного інтелекту та розпізнавання об'єктів – базового та Gimbal object detection. [5]

1.2.4. Автономні транспортні засоби

Окремою галуззю є використання комп'ютерного зору в розробці й проектуванні самохідних транспортних засобів. До таких, зокрема, можна віднести субмарини, сучасні автопілотні машини Tesla [6], безпілотні літальні апарати.

На яскравому прикладі розробок компанії Ілона Маска можна впевнитися, що алгоритми комп'ютерного зору не просто корисні, а й є одними з базових для побудови машин майбутнього. Зокрема, вміння автомобіля зрозуміти, що його оточує, плавно переходить з поєднання використання сенсорів з камерами до повноцінного аналізу ситуації, базуючись виключно на даних, отриманих з камер корпусу машини.[6]

Вище згаданий Bayraktar також є чудовим прикладом, як окрім допомоги в наведенні на ціль, комп’ютерний зір надає перевагу в зручному користуванні приладом та прокладанні літальних маршрутів до цілі й розвідки об’єктів.

1.2.5 Tactile feedback

Існує галузь в світі, яка орієнтується на вібрації, та чиї методи на ній базуються. Сенс Tactile feedback, яку можна перекласти як «зворотній зв’язок від контакту», у посиланні вібрації та хвиль, що використовуються для передачі даних користувачу або оператору електронного девайсу.[7]

Компоненти таких систем мають можливість надсилати вібрації, які потім аналізуються людиною чи кінцевим користувачем, який може їх сприймати. Людина дуже чутлива до вібрацій і може органічно їх відрізняти. У статті [7] така процедура порівнюється з вивченням нової мови, але процес набагато легший.

Така техніка використовується, наприклад, у засобах паркування в сучасних машинах. Водію надсилається вібрація різної сили та частоти, що покращує розуміння того, скільки відстані до перепони має машина.

Сенсори, що забезпечують таку технологію, можуть використовувати також і вбудовані камери, які забезпечують більш точну та зважену інформацію щодо об’єкту тактильної взаємодії, базуючись на алгоритмах комп’ютерного зору.

1.2.6 Розпізнавання

Однією з основних сфер застосування комп’ютерного зору також є методика розпізнавання, що включає в себе багатогранні реалізації:

- Ідентифікація – завданням машини є точно ідентифікувати конкретного індивіда. Такі технології використовуються, зокрема, в ідентифікації людини шляхом аналізу обличчя або відбитку пальців;
- Розпізнавання об’єкту – сучасні машини, що першочергово навчаються на попередньо заданих прикладах, можуть бути використані для знаходження ідентичних об’єктів на тих медіа файлах, що вони використовують. Наприклад, розпізнавати й відрізняти котів на зображенні з тваринами;
- Визначення – така технологія використовується в вище згаданій області медицини, коли на медичних зображеннях комп’ютер має здатність визначити аномалії й надати більш точні регіони, які згодом можна буде аналізувати іншими алгоритмами, та інші.

Також гарним прикладом є камери сучасних телефонів, які на фотографії чи просто в режимі зйомки можуть пропонувати користувачеві скопіювати текст із зображення чи перейти за посиланням QR-коду.

Однак, сучасні реалії розвитку комп'ютерного зору не дозволяють йому повноцінно наблизитись до розуміння, яким наділена людина. Так, наприклад, для додаткового захисту від спам-ботів і небажаних реєстрацій акаунтів машинами використовуються методи Captcha. На зображеннях, наданих цим тестом, машина не зможе чітко виявити об'єкти, які слід обрати чи записати, у той час як для людини цей тест зазвичай не створює проблем. Це пов'язано з тим, що комп'ютерний зір гірше працює з тонкими об'єктами, або об'єктами, які накладаються один на одного чи використаними на зображенні фільтрами.

На противагу цьому, комп'ютер краще працює з точною ідентифікацією об'єкту. Не кожна людина є знавцем усіх порід пташок чи комах, а добре навчена машина не матиме проблем з такого роду завданням.

1.2.7 Аналіз руху

Дана технологія є автоматизацією аналізу будь-якого рухомого процесу, який вдалося зняти на камеру. Вона включає в себе як розрахування перенесення певних точок чи тривимірних об'єктів в просторі, так і зміни зображення за рахунок переміщення камери. Основними прикладами застосування є:

- Оптичний потік – аналіз руху, що базується на перенесенні всіх точок (наприклад, пікселів) на зображенні. Така технологія називається apparent motion, намагаючись аналізувати переміщення об'єктів на відео як зміну положення пікселів за позиціями x, y та t (часом). Також поняття оптичного потоку розглядає як переміщення власне об'єктів, так і зміну положення камери;
- Відслідковування – методика, що дозволяє слідкувати за зміною положення певних пікселів чи об'єктів на екрані. Нерідко використовується в аналізі орбітального руху, руху машин та й, у цілому, будь-якого рухомого об'єкту для створення подальших припущень на базі отриманих спостережень;
- Egomotion – аналіз руху камери відносно тривимірних елементів сцени навколо неї. Прикладом використання є визначення положення машини в просторі шляхом аналізу зсуву позиції встановленої на ній камери.

1.2.8 Реконструкція сцен та відновлення зображень

Як зазначалося у вступі до роботи, комп’ютерний зір дозволяє відтворювати 3D-сцени за кількома знімками об’єкту. Така технологія використовується в архітектурі, відтворенні зруйнованих об’єктів, моделюванні повноцінних моделей поверхонь. Елементарним результатом роботи алгоритму є створення масиву тривимірних точок; у сучасності ж комп’ютери дозволяють відновлювати такі 3D-моделі з певної кількості тривимірних зображень, тим самим дозволяючи аналізувати об’єкт з будь-якого ракурсу глядача.

Окремою сферою застосунку є відновлення зображень. Звичайними вище згаданими застосуваннями є зменшення шуму на зображенні безпілотників, чіткіше виявлення контурів на медичних знімках. Аналізуючи схожі зображення, комп’ютер також може вивчати й підбирати певні методики, які покращують візуальну структуру конкретно цієї групи знімків чи об’єктів.

Розділ 2. ОПТИЧНИЙ ПОТІК ТА ЙОГО ТИПИ

2.1 Загальні відомості

«Optical flow or optic flow is the pattern of apparent motion of objects, edges and surface in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene».[8]

Оптичний потік – це методика, яка широко використовується в таких галузях комп'ютерного зору як відстеження, розпізнавання, відслідковування та інших. Його ідеєю є аналіз руху на зображенні за рахунок створення вектору руху кожного пікселя, враховуючи зміну його позиції на кожному з кадрів відео.

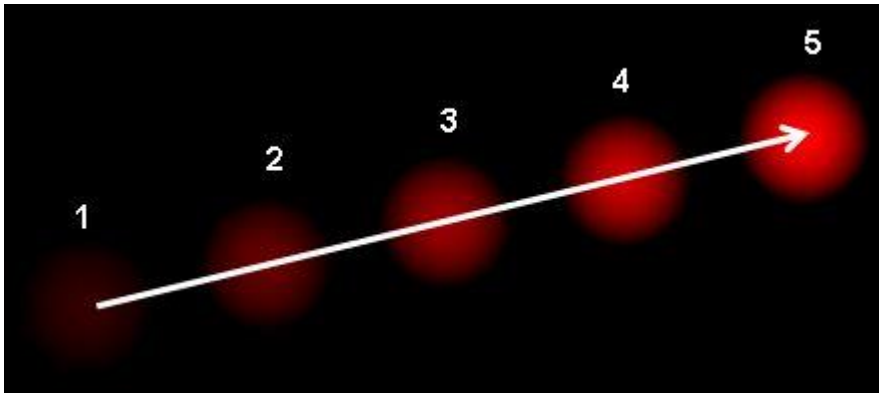


Рисунок 1. Вектор руху, визначений оптичним потоком

Оптичний потік базується на певних припущеннях:

- Інтенсивність пікселів для кожного окремого об'єкту не змінюється з плином часу;
- Сусідні пікселі мають схожий напрямок руху, що буде проаналізовано на прикладі використання «вікна» пікселів.

Однією з застав ідеї оптичного потоку також є вище згаданий аналіз руху тривимірних об'єктів як зміни положення пікселів, що належать цим об'єктам, у двовимірному просторі. Тобто, іншими словами, аналізується рух справжніх об'єктів на зображенні через рух пікселів, які, за припущенням, описують ці об'єкти.

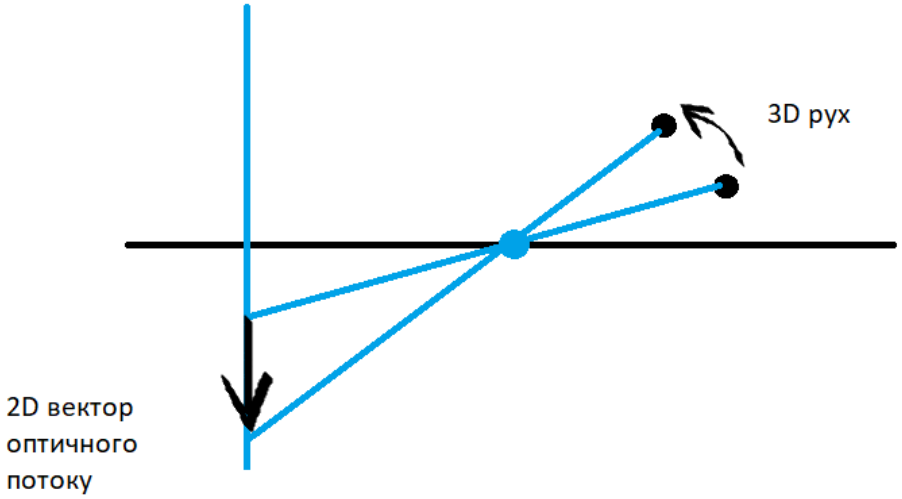


Рисунок 2. Трактування 3D-руху як 2D-вектору

Незважаючи на те, що технологія оптичного потоку доводить, що трактувати тривимірні переміщення можна за рахунок двовимірних векторів, математика цієї методики все одно включає в себе поняття часу. Нівелювати їм не є можливим, тому на відміну від координат пікселів в теорії обробки зображень, надалі для кожного пікселю використовуватимуться координати (x, y, t), де x та y – відповідно, координати осей абсцис та ординат, а t – час.

2.2 Оптичні потоки: Sparse і Dense

Якщо намагатися типізувати оптичні потоки, слід виділити два основні напрямки:

- Оптичний потік типу Sparse – «розріджений» – цей метод базується на аналізі зображення й побудові векторів руху лише кількох найбільш рухомих пікселів між кадрами;
- Оптичний потік типу Dense – «щільний» – такий спосіб намагається охопити й побудувати вектори руху для кожного пікселя на відео.

Очевидним є більше навантаження на машину, а отже повільніше виконання, другого алгоритму. Також слід зазначити, що від кількості пікселів на екрані в цілому ускладнюється чи полегшується робота оптичного потоку[9].

Рисунок 3. Порівняння роботи Sparse (зліва) і Dense (праворуч) оптичних потоків

2.3 Нівеляція яскравості оптичного потоку

Припускається, що розглядається точка на відео з координатами I(x, y, t) на першому кадрі. Ця точка рухається за напрямком вектору (u, v), що відповідно змінює її координати абсциси на x + u, а ординат – на y + v. Однак, як було згадано вище, точки зображення розглядаються виключно в декартовій системі координат, а отже залишається визначити час, який у цьому випадку просто переходить до наступного кадру, а тому його позиція стає t + 1, що видно з рис. 4.

Рисунок 4. Переміщення точки в координатній площині (x, y, t)

Це підводить нас до, так званого, рівняння сталої яскравості [9]:

I (x, y, t) = I (x + u, y + v, t + 1), де

x, y, t – відповідні координати абсцис, ординат та часу до зміни положення точки,

u, v – зміна відповідних координат у просторі,

t + 1 – результуючий кадр після зміни положення точки.

Дане твердження також може бути записане як:

$$0 = I\ (x + u,\ y + v,\ t + 1) - I\ (x,\ y,\ t)$$

Оскільки в цьому конкретному прикладі розглядається зміна положення точки з одного кадру в наступний, нормальним буде припустити, що рух був незначним. Розклад ряду Тейлора може бути використаний, аби узагальнити рівняння, розклавши I (x + u, y + v, t + 1):

$$0 \approx I\ (x, y, t + 1) + I_x u + I_y v - I(x, y, t),$$ де

$I_x u$ – похідна в напрямку абсцис,

$I_y v$ – похідна в напрямку ординат

Також слід зазначити, що $I_x = \frac{\partial I}{x}$, і $I_y = \frac{\partial I}{y}$ для t та t + 1. Продовжуючи припускати, що рух незначний, допускається, що похідна в точці буде приблизно однакова як для t, так і для t + 1.

$$0 \approx [I\ (x, y, t + 1) - I(x, y, t)] + I_x u + I_y v,$$ що слідує в:

$$0 \approx I_t + I_x u + I_y v$$

Заміненою є різниця $I\ (x, y, t + 1)$ та $I(x, y, t)$ на I_t. I_t називається похідною часу, а отже відповідає за зміну I відносно часових рамок. Повертаючись назад, точка складалася з функції $I(x, y, t)$, отже похідна береться відносно трьох координат та рівняння може бути записане як:

$$0 \approx I_t + \nabla I * [u, v],$$ де

$\nabla I * [u, v]$ – градієнт зображення.

Градієнтом зображення називають напрямок зміни інтенсивності або кольору зображення[10]. Він, як і описано в формулах вище, вираховується шляхом отримання похідних в горизонтальному та вертикальному напрямках,

що в ситуації є зміна вектору руху [u, v] для показників (x, y) на відповідних осях.

Упродовж обрахунків підведено до такого вигляду формули:

$$I_x u + I_y v + I_t = 0 \quad (1)$$

Дане рівняння має дві невідомі, а отже не може бути вирішено. Така проблема й має назву «aperture problem», або «проблема отвору», якою буде її назва далі.

Фінальний варіант формули сталої яскравості може мати вигляд рівняння одного вектору[9]:

$$\vec{\nabla} I \cdot \vec{u} = -I_t \text{ , де}$$

$$\vec{\nabla} I = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

$$\vec{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

2.4 Градієнтний компонент потоку

2.4.1 Загальні відомості

Базуючись на формулі (1), виведеної в попередньому розділі, постає звичне питання: скільки ж невідомих в одному рівнянні для кожного пікселя й скільки таких рівнянь існує? [9]

В формулі (1) згадуються лише дві невідомі – (u, v) й одне рівняння для пікселю.

Однак, є спосіб визначити ці невідомі.

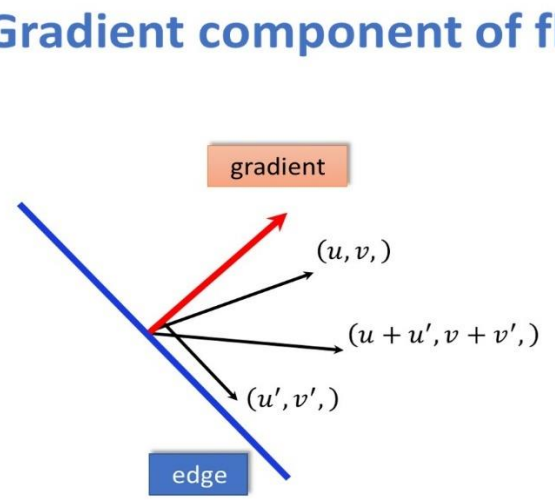


Рисунок 5. Градієнтний компонент потоку

З рисунку 5 чітко зазначені наступні вектори й напрямки:

- edge – границя нашого зображення;
- gradient – напрямок зміни градієнту;
- паралельний до границі вектор (u′, v′) – додатковий компонентний вектор;
- (u, v) – вектор з невідомими, заданий у рівнянні;
- (u + u′, v + v′) – вектор, що має ідентичний до вектору (u, v) напрямок, але також й інший кут нахилу до границі.

Проблема отвору полягає в тому, що визначити індивідуально для цієї ситуації можливо винятково дистанцію, що була пройдена точкою паралельно відносно границі. Таку ситуацію можна порівняти, наче на зображення дивитися через отвір. Оскільки видно рух не всього об’єкта, але лише його частину, це може дати хибне припущення й неправильний аналіз руху[9].

2.4.2 Проблема отвору

Моделюється ситуація, що виникає спроба аналізувати рух певного досить тонкого об’єкту, наприклад, переміщення лінії на GIF-зображенні.

Однак, проблема в тому, що не є очевидним повний рух, оскільки на екран, який розглядається, накладено суцільне полотно з вирізаним отвором, що дає перспективу, представлену на рисунку 6.

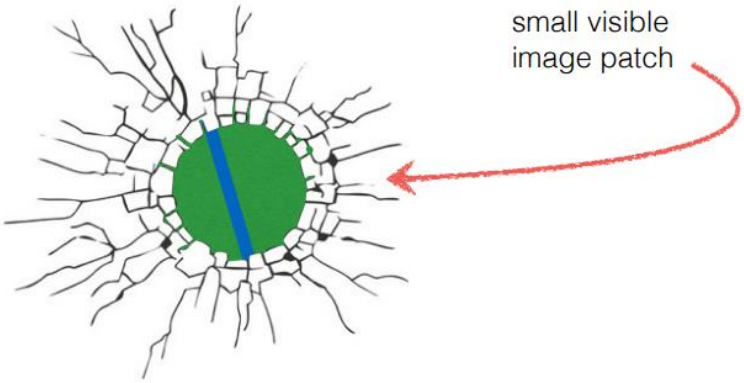


Рисунок 6. Перспектива аналізу зображення крізь отвір

Нехай спостереження ведеться за цим об’єктом у спробі аналізувати як він рухається на зображенні й видно переміщення, як на рисунку 7.

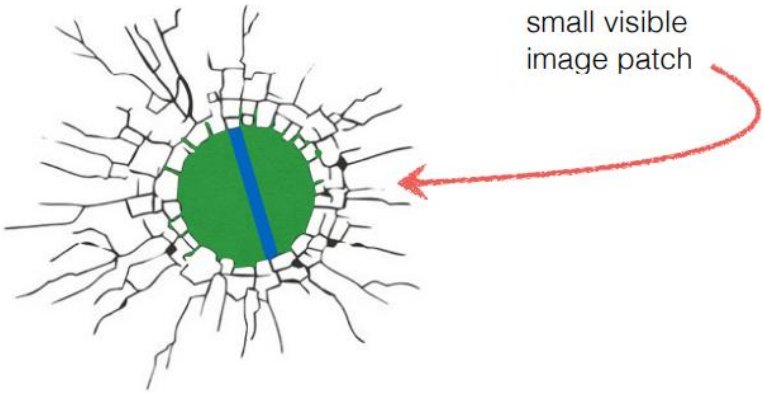


Рисунок 7. Перспектива аналізу зображення крізь отвір після руху

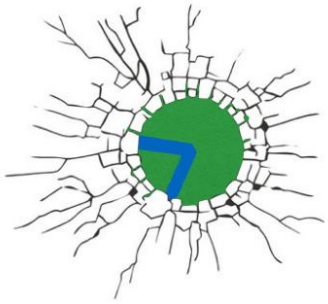
Очевидно здається, що об’єкт переміщується перпендикулярно до границь отвору. Неможливо передбачити, чи рухається об’єкт вгору чи вниз, але можемо виникнути припущення, що рух здійснюється праворуч.

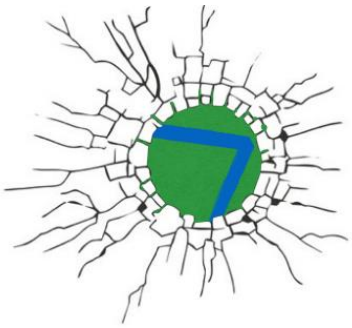
Насправді ж, прибравши полотно з отвором, можна побачити наступний перехід від рисунку 8 до рисунку 8.1.

Рисунки 8 та 8.1. Аналіз руху зображення без полотна

З цього прикладу видно, що насправді ж об’єкт рухається вгору, а через його форму, перспектива спостерігача (чи то людини, чи комп’ютера) створила хибні припущення, дозволивши припустити рух праворуч.

Для запобігання такої похибки слід використовувати інші перспективи погляду на об’єкт через «отвір» - використовувати ділянки з різними градієнтами, аби уникнути проблеми отвору[11].





Рисунки 9 та 9.1. Використання ділянок з різними градієнтами при проблемі отвору

2.4.3 Розв’язання проблеми отвору. Застосування формули апроксимації менших квадратів Брюсом Лукасом та Такео Канаде

Шляхом розв’язання даної проблеми є спроба додати більше локальних обмежень, тим самим підвищивши кількість рівнянь для кожного пікселя. Як зазначалося вище, припускається, що рух об’єкта від кадру до кадру дуже плавний й рівномірний [9]. Насправді ж, в аналізі руху орієнтування відбувається на зміну положення не одного пікселя, а околу навколо нього.

Нехай обирається вікно розміру 5x5, це 25 пікселів. За припущенням, пікселі в цьому околі будуть рухатись рівномірно за одним вектором (u, v), який так само накладається на центральний піксель, який досліджується. У цьому разі отримано 25 рівнянь для визначення положення одного пікселя, і виглядатимуть вони так:

$$0 = I_t(p_i) + \nabla I(p_i) * [u \;\; v]$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}, \quad (2)$$

де вектор $\begin{bmatrix} u \\ v \end{bmatrix}$ – вектор зміни положення точки,

$p_1 \ldots p_{25}$ – точки в околі 5x5.

Як результат обрахування формули (2), отримується вектор розміру 25x1, який по суті є негативним значенням всіх похідних часу для точок.

Виявлено наступну проблему – як розв’язати систему, що має більше рівнянь, аніж невідомих? Для цього варіантом є використання підходу менших квадратів, таким чином мінімізувавши квадратну дистанцію[9].

Формула апроксимація менших квадратів виглядає так:

$$\hat{x} = \arg\min\|Ax - b\|^2, \text{ що дорівнює розв}'\text{язку } A^T A\hat{x} = A^T b$$

У випадку, що розглянутий вище:

$$A^T A = \begin{bmatrix} \sum_{p \in P} I_x I_x(p) & \sum_{p \in P} I_x I_y(p) \\ \sum_{p \in P} I_y I_x(p) & \sum_{p \in P} I_y I_y(p) \end{bmatrix}$$

$$\hat{x} = \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A^T b = -\begin{bmatrix} \sum_{p \in P} I_x I_t(p) \\ \sum_{p \in P} I_y I_x(p) \end{bmatrix}$$

де p – кожна окрема точка з множини точок в околі 5x5 P

Такий метод носить назву псевдо-інверсійного й остаточно може бути записаний як:

$$\begin{bmatrix} \sum_{p \in P} I_x I_x(p) & \sum_{p \in P} I_x I_y(p) \\ \sum_{p \in P} I_y I_x(p) & \sum_{p \in P} I_y I_y(p) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{p \in P} I_x I_t(p) \\ \sum_{p \in P} I_y I_x(p) \end{bmatrix}$$

Формула 3

Обрахувавши формулу 3 можна вирішити метод найменших квадратів. Однак, аби ця формула мала розв'язок, необхідні наступні умови:

- $A^T A$ має бути невиродженою, а отже її визначник не дорівнює нулю;
- $A^T A$ не має бути малою. Тобто окіл має бути доцільного розміру;
- $A^T A$ не мусить мати завеликі власні вектори λ_1 та λ_2[11].

Така техніка використання формули найменших квадратів уперше була запропонована Брюсом Д. Лукасом та Такео Канаде в 1981 році[9].

2.4.5 Поєднання локальних обмежень. Робота в RGB-просторі

У прикладі, розглянутому в пункті 2.4.4, було використано один спільний градієнт для пікселів в околі. Однак, у сучасному світі градієнтна зміна можлива в трьох напрямках відомої RGB-системи – червоному, зеленому та синьому.

Нехай, до цього використовувався градієнт синього кольору. Однак, як видно з рисунку нижче, накладаються також і градієнти червоного й зеленого кольорів. Розв'язком завдання для вектору (u, v) стане знаходження точки перетину трьох ліній.

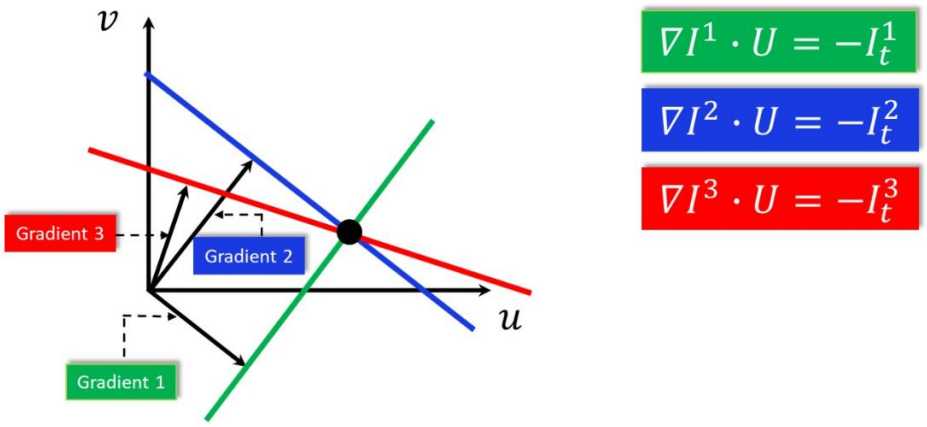


Рисунок 10. Використання RGB в градієнтному компоненті потоку

З точки зору математичних розрахунків, у цій роботі припускалась матриця розміру 5x5 з 25 елементами – пікселями. У випадку RGB-системи мають використовуватися ідентичні матриці для кожного з кольорів. Це призведе до виникнення 75 рівнянь (5x5 матриця на 3 потоки кольорів).

$$0 = I_t(p_i)[0,1,2] + \nabla I(p_i)[0,1,2] * [u\ v]$$

$$\begin{bmatrix} I_x(p_1)\{0,1,2\} & I_y(p_1)\{0,1,2\} \\ \vdots & \vdots \\ I_x(p_{25})\{0,1,2\} & I_y(p_{25})\{0,1,2\} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1)\{0,1,2\} \\ \vdots \\ I_t(p_{25})\{0,1,2\} \end{bmatrix}, \text{ де}$$

0, 1, 2 – потоки кольорів RGB.

Внаслідок накладання кольорової палітри, для одного пікселю виникає тепер не одне, а три рівняння й ті самі дві невідомі (u, v).

Розділ 3. АЛГОРИТМ ЛУКАСА-КАНАДЕ

3.1 Загальні відомості

«The Lucas–Kanade method is a widely used in differential method for optical flow estimation and computer vision»[12].

За книгою Брюса Лукаса та Такео Канаде, розроблений науковцями метод вирішує базові рівняння оптичного потоку, розглянуті вище, для кожного пікселя в околі, шляхом застосування формули апроксимації менших квадратів. Недоліком алгоритму є можливість його застосування виключно на локальні позиції точок, а отже використання такої техніки для визначення загального потоку на зображенні не є релевантним[12].

Слід нагадати, що алгоритм Лукаса-Канаде базується на двох припущеннях:

- Стабільної інтенсивності руху пікселя;
- Напрямок руху сусідніх пікселів є схожим.

Як наслідок вищезазначених припущень, провідною ідеєю цього алгоритму є використання околу пікселів для точнішого вирахування оптичного потоку центрального пікселя та застосування формули найменших квадратів для апроксимації вектору руху.

Адаптацією алгоритму можна назвати використання зваженого околу пікселів. Ідеєю використання «ваги» пікселів в вікні є підвищена зваженість впливу ближніх сусідів на центральний піксель у порівнянні з віддаленими від нього точками.

Зважена Формула 3 має такий вигляд:

$$\begin{bmatrix} \sum_{p \in P} wI_xI_x(p) & \sum_{p \in P} wI_xI_y(p) \\ \sum_{p \in P} wI_yI_x(p) & \sum_{p \in P} wI_yI_y(p) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum_{p \in P} wI_xI_t(p) \\ \sum_{p \in P} wI_yI_x(p) \end{bmatrix}$$

Де w – коефіцієнт ваги пікселя p ∈ P

Коефіцієнт ваги пікселя зазвичай вимірюється функцією Гауса для дистанції між p та центральним пікселем[13].

3.2 Відстеження ознак за алгоритмом Лукаса-Канаде

Для знаходження найнеобхідніших, а отже найбільш рухомих, об’єктів на зображенні, був покращений базовий алгоритм Лукаса-Канаде.

Замість того, аби вираховувати зміну положення кожного пікселя шляхом базовим: обрання околу, вирахування за Формулою 3 чи за зваженою Формулою 3 оптичного потоку пікселя, за Джастіном Гремом[14], було запропоновано першочергово обраховувати власні значення градієнтних матриць пікселів за наступним алгоритмом:

1. Обрахування інтенсивності кожного пікселя;
2. Вирахування матриці градієнту та власних значень матриці;
3. Збереження власних значень кожного пікселя в загальній базі за посортованим списком;
4. Обрання пікселів з найбільшими власними значеннями, які були отримані в результаті обрахувань пункту 2.

Такий спосіб значно полегшує та прискорює роботу алгоритму в цілому, а також позбавляє безпосереднього користувача чи робота зайвих візуальних ефектів на зображенні, якщо прокладається шлях пікселя на відео.

Зрозуміло, що це досі не ідеальний підхід до аналізу руху на зображенні, однак, покладаючись на базові шляхи розв’язку, він є легким і досить невибагливим, орієнтуючись на загальновідомі формули та наявні в пікселів показники.

3.3 Похибки та покращення алгоритму

Алгоритм Лукаса-Канаде є стабільним за умови виконання 2 основних припущень. Однак, очевидним є той факт, що забезпечувати їхнє дотримання штучно шляхом імплементації якихось конкретних чи абстрактних алгоритмів неможливо, а сподіватись, що іншого просто не станеться – на цьому, в принципі, і базуються припущення.

Отже, слід підсумувати похибки алгоритму Лукаса-Канаде:

1. Неможливо передбачити, що рух буде стабільним, а не різким; іншими словами, з точністю сказати, що рух буде незначним, неможливо;
2. Піддається сумніву використання околу пікселів, адже постає проблематика його розміру: якщо вікно завелике, це дасть похибку; як знайти ідеальний параметр розміру?
3. Чи завжди буде зберігатися сталість яскравості?
4. Збереження правил щодо матриці $A^T A$, згаданих вище;
5. Похибка на границях зображення.

Серед покращень, згаданих вище:

- Використання зваженої Формули 3, запропоноване Брюсом Лукасом та Такео Канаде;
- Покращення відстеження об’єктів за допомогою першочергової фільтрації малорухомих точок, згадане в пункті 3.2;
- Використання екстраполяції векторів руху, вирахуваних на попередніх кадрах для уточнення напрямку;
- Накладання алгоритму Лукаса-Канаде на зменшене зображення, що підвищує точність й важливість околу під час використання.

Розділ 4. РЕАЛІЗАЦІЯ АЛГОРИТМУ ЛУКАСА-КАНАДЕ

На рисунку 3 вже було продемонстровано практичний вигляд пошуку оптичного потоку, а точніше двох його типів – Sparse і Dense.

У цьому розділі буде розглянуто імплементацію алгоритму Лукаса-Канаде на основі бібліотеки OpenCV від Intel Corporation на мові програмування Java.

OpenCV – за Вікіпедією, бібліотека комп'ютерного зору з відкритим кодом [15]. У ній реалізовані близько 2500 з сучасних та базових алгоритмів комп'ютерного зору, зокрема для таких цілей [16]:

- Обробка зображень;
- Аналіз відео;
- Машинне навчання;
- Калібрація камери та 3D реконструкція й інші.

Далі розглянутим буде базовий алгоритм Лукаса-Канаде з використанням пошуку точок методом Ши-Томасі.

Першочергово додаються імпорти в проект, які є необхідними Java та OpenCV модулями:

```
import java.util.ArrayList;
import java.util.Random;
import org.opencv.core.*;
import org.opencv.highgui.HighGui;
import org.opencv.imgproc.Imgproc;
import org.opencv.video.Video;
import org.opencv.videoio.VideoCapture;
```

opencv.core – використовується для базових математичних операцій, які необхідні для застосування в методах;

opencv.highgui.HighGui – створення та керування вікнами користувацького інтерфейсу;

opencv.imgproc.Imgproc – робота з зображенням та базові способи створення малюнків на зображенні;

opencv.video.Video – аналіз відео, імплементація алгоритму Лукаса-Канаде саме в цьому модулі;

opencv.videio.VideoCapture – зчитування відео.

Наступні стрічки коду додаються в клас задля завантаження бібліотеки OpenCV та зчитування відео:

```
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
String filename = args[0];
VideoCapture capture = new VideoCapture(filename);
if (!capture.isOpened()) {
    System.out.println("Unable to open this file");
    System.exit(-1);
}
```

Наступним кроком створюється метод для генерації кольорів, які позначатимуть переміщення об’єктів на відео:

```
public static void generateColors() {
    Random random = new Random();

    for (int i = 0 ; i < colors.length; i++) {
        int redScalar = random.nextInt(256);
        int greenScalar = random.nextInt(256);
        int blueScalar = random.nextInt(256);

        colors[i] = new Scalar(redScalar, greenScalar, blueScalar);
    }
}
```

Даний метод викликається в main, попередньо створивши глобальну змінну Scalar colors[].

Слід зазначити, що Scalar знаходиться в модулі opencv.core та дозволяє динамічно створювати кольори за палітрою RGB.

Після ініціалізації кольорів створюються два об’єкти класу Mat() – матриці – також з модуля core: previousFrame, previousGrayGradient.

Аналогом реалізації методу Ши-Томасі є goodFeaturesToTrack(…), який знаходиться в ImgProc. Для його ініціалізації необхідний елемент класу MatOfPoint – матриця точок, тому його створено під назвою firstMatOfPoint.

Далі, користуючись засобами VideoCapture викликається наступна стрічка коду:

```
capture.read(previousFrame);
```

Метод read, приймаючи нині порожню матрицю на вхід, повертає в цьому випадку перший кадр відео.

Далі викликаються два методи з ImgProc:

```
Imgproc.cvtColor(previousFrame, previousGrayGradient, Imgproc.COLOR_BGR2GRAY);
Imgproc.goodFeaturesToTrack(previousGrayGradient, firstMatOfPoint,100,0.3,7, new
Mat(),5,false,0.04);
```

cvtColor – трансформує зображення з кольорової матриці в матрицю сірого градієнту для подальшого зручного аналізу зображення (слід згадати про використання RGB в алгоритмі Лукаса-Канаде);

goodFeaturesToWork, як вже згадувалось – імплементація методу Ши-Томасі для знаходження потрібних для аналізу точок. Він отримує:

- Матрицю сірого градієнту;
- Порожню матрицю точок, яка в результаті наповниться необхідними для аналізу границями;
- Максимальну кількість таких точок;
- Мінімальний поріг для визначення точки необхідною;
- Мінімальну дистанцію між точками;
- Порожню матрицю, тим самим задавши, що шуканими є необхідні точки на всьому зображенні;
- Розмір околу для знаходження сусідів.

Надалі конвертується отримана матриця точок, які необхідно дослідити, в матрицю MatOfPoint2f, яка відрізняється тим, що використовує значення 32-float, які є точнішими. І створюється порожня матриця ідентичного типу для подальшого використання в обрахуванні алгоритму Лукаса-Канаде.

Після цього ініціалізується матриця mask для відмічання точок на зображенні.

На цьому етапі визначеними є точки, які будуть розглянуті на зображенні, та відбувається перехід до основної частини імплементації алгоритму.

Наступні кроки відбуватимуться в циклі while(true), з якого процес вийде тоді, коли наступного кадру на зображенні вже не буде. Завдання циклу – покроково для кожного кадру визначати напрямок оптичного потоку шляхом обрахування формули Лукаса-Канаде та залишати «слід» на зображенні конкретними методами модуля ImgProc.

Першочергово в циклі ініціалізується кадр шляхом його зчитування та записування в матрицю, як на початку розробки, та переводиться кадр в градієнт сірого:

```
Mat frame = new Mat(), currentGrayGradient = new Mat();
capture.read(frame);
if (frame.empty()) {
    break;
}
Imgproc.cvtColor(frame, currentGrayGradient, Imgproc.COLOR_BGR2GRAY);
```

Наступним кроком є створення трьох змінних для calcOpticalFlowPyrLK(…) – методу Лукаса-Канаде, розробленого OpenCV:

- Порожні матриці для заповнення їх вищезгаданої функцією:

```
MatOfByte status = new MatOfByte();
MatOfFloat err = new MatOfFloat();
```

- Об'єкт класу TermCriteria – загальний клас, який визначає певні критерії для роботи алгоритмів:

```
TermCriteria criteria = new TermCriteria(TermCriteria.COUNT +
TermCriteria.EPS,10,0.03);
```

TermCriteria.COUNT = 1 за замовченням – кількість ітерацій алгоритму;

TermCriteria.EPS = 2 за замовченням – точність, яка досягається за рахунок роботи алгоритму, зміни елементів або порогу.

Передаючи вищезазначені параметри в конструктор об’єкту, зазначаються:

1. Тип критерії;
2. Максимальна кількість ітерацій;
3. Точність.

Після цього викликається наступну стрічку коду, яка й починає обраховувати алгоритм Лукаса-Канаде:

```
Video.calcOpticalFlowPyrLK(previousGrayGradient, currentGrayGradient,
firstMatOfPoint2F, secondMatOfPoint2F, status, err, new Size(15,15),2,
criteria);
```

- previousGrayGradient – попередній кадр в сірому градієнті;
- currentGrayGradient – поточний кадр в сірому градієнті;
- firstMatOfPoint2F secondMatOfPoint2F – зазначені вище вхідна та вихідна матриці точок;
- status – результуючий вектор з 0, якщо для конкретної точки не було знайдено вектору руху, та 1 в противному разі;
- err – масив помилок в разі обрахування;
- new Size(15, 15) - розмір околу пошуку;
- 2 – максимальний рівень пірамідального пошуку вектору, що застосовується в алгоритмі;
- Criteria – вищезазначений критерій для зупинки роботи алгоритму.

У результаті роботи алгоритму, отримуються основні дані про зміну положення точки шляхом аналізу двох матриць та вектору status.

Надалі результат роботи зводиться до простого відображення точок та ліній на зображенні й використанні методів ImgProc, core та HighGui для цього.

Однак, слід відмітити, що також використовується вектор status для більш вдалого відсіювання тих точок, статус яких алгоритмом був повернутий як 0. Це зроблено для того, аби уникнути повторення аналізу об’єктів, які не рухалися в попередньому кадрі.

Висновки

У цій роботі було розібране поняття комп’ютерного зору й досліджено основні галузі його застосування в суспільстві:

- Медицина;
- Військова справа;
- Будування автономних машин та пристроїв;
- Tactile feedback та інші.

Було швидко пройдено крізь історію комп’ютерного зору та його розвиток і класифікації. Засвоєно якими методиками оперує дана технологія та розглянуто застосування однієї конкретної – оптичного потоку.

У роботі продемонстровано загальну ідею оптичного потоку, яка полягає в виявленні векторів руху для кожного чи певних пікселів на зображенні. Визначено, що дана технологія розділяється на два типи: Sparse і Dense та на прикладі показано, чим вони відрізняються, охоплюючи лише необхідні для аналізу точки або намагаючись охопити переміщення всіх об’єктів на кадрі.

Вся подальша робота базувалася на двох припущеннях:

- Сталості й незначності руху;
- Вектори руху сусідніх пікселів є схожими.

Базуючись на вищезазначеному, було доведено й використано нівеляцію яскравості, якою користується оптичний потік, та виведено формулу (1).

Після цього, у роботі проаналізовано проблематику та математику градієнтної складової оптичного потоку. Сконструйовано проблему отвору з можливими варіантами її розв’язання шляхом використання околу пікселів для підвищення точності в центральному пікселі. Розібраним був варіант розв’язку системи, де рівнянь для кожного пікселю було більше, аніж невідомих у них.

Метою роботи було розглянути тактику використання формули апроксимації менших квадратів, запропонованої Брюсом Лукасом та Такео Канаде. Завдяки ній, матричним шляхом вдалося знайти розв’язок. Однак, виникла проблема сучасності – використання не одного потоку кольору (наприклад, сірого градієнту), а RGB-системи, що підвищує складність обрахувань втричі, оскільки для кожного потоку використовується та сама матриця розв’язання з 25 рівнянь, а отже сумарно їх з’являється 75.

В розділі 3 проаналізовано алгоритм Лукаса-Канаде, його загальну ідею та користь. Також розказано про використання зваженої Формули 3 для більш точного використання алгоритму шляхом накладання кожному пікселю в формулі додаткового параметру ваги, який зазвичай вираховується за

формулою Гауса. Розібрано адаптацію алгоритму, аби нівелювати малорухомі точки на об’єкті.

У кінці розділу 3 підбито підсумки похибок використання алгоритму Лукаса-Канаде, а також розглянуто варіанти покращення його продуктивності та результату.

В останньому розділі детально розібрано використання бібліотеки OpenCV для імплементації алгоритму Лукаса-Канаде на Java. На конкретній програмі продемонстровано, яким чином він працює та як його роботу можна покращити, аналізувавши зображення в градієнті сірого, замість RGB, та шляхом відкидання малорухомих точок.

Завданням роботи було розглянути й проаналізувати засоби та техніки використання комп’ютерного зору. Розглянуто основні теоретичні засади та базис оптичного потоку й його варіацій. Після чого розібрано алгоритм Лукаса-Канаде й продемонстровано варіант вирішення проблеми отвору, запропонований авторами. Як результат, доведено, що комп’ютерний зір має видатні здібності до аналізу руху та відстеження об’єктів, але також і свої похибки й проблеми, які слід вирішувати.

Список літератури

[1] - Dana H. Ballard; Christopher M. Brown (1982) «Computer vision»

[2] - Richard Szeliski (30 September 2010). «Computer Vision: Algorithms and Applications»

[3] - https://en.wikipedia.org/wiki/Computer_vision#History

[4] - https://viso.ai/applications/computer-vision-in-healthcare/#:~:text=Computer%20vision%20has%20been%20used,enable%20a%20more%20accurate%20diagnosis.

[5] - https://www.baykartech.com/en/artificial-intelligence/

[6] - https://electrek.co/2021/05/25/tesla-vision-without-radar-warns-limitations-first/

[7] - https://www.hallmarknameplate.com/tactile-feedback-works/#:~:text=Haptic%20feedback%20(also%20known%20as,operator%20of%20an%20electronic%20device.

[8] – Huston S. J., Krapp HG. Kurtz, Rafael. «Visuomotor Transformation in the Fly Gaze Stabilization System» Bielefeld University, Germany, Year 2008.

[9] - https://datahacker.rs/calculating-sparse-optical-flow-using-lucas-kanade-method/#Optical-Flow-Constraint

[10] - https://en.wikipedia.org/wiki/Image_gradient

[11] – «Lucas-Kanade Optical Flow Computer Vision 16-385» Carnegie Mellon University (Kris Kitani)

[12] - Bruce D. Lucas «Image Matching by the Method of Differences», Carnegie Mellon University, year 1984

[13] - https://research.ijcaonline.org/volume61/number10/pxc3884611.pdf

[14] - Justin Graham, «Target tracking with Lucas-Kanade optical flow», University of Texas, year 2010.

[15] - https://uk.wikipedia.org/wiki/OpenCV

[16] - https://docs.opencv.org/4.5.5/

Додаток А (обов'язковий) Програма алгоритму Лукаса-Канаде

```
import java.util.ArrayList;
import java.util.Random;
import org.opencv.core.*;
import org.opencv.highgui.HighGui;
import org.opencv.imgproc.Imgproc;
import org.opencv.video.Video;
import org.opencv.videoio.VideoCapture;

public class OptFlow {
    private static final Scalar[] colors = new Scalar[100];

    public static void main(String[] args) {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        String filename = args[0];
        VideoCapture capture = new VideoCapture(filename);
        if (!capture.isOpened()) {
            System.out.println("Unable to open this file");
            System.exit(-1);
        }

        generateColors();

        Mat previousFrame = new Mat();
        Mat previousGrayGradient = new Mat();

        MatOfPoint firstMatOfPoint = new MatOfPoint();
        capture.read(previousFrame);

        Imgproc.cvtColor(previousFrame, previousGrayGradient,
Imgproc.COLOR_BGR2GRAY);
        Imgproc.goodFeaturesToTrack(previousGrayGradient,
firstMatOfPoint,100,0.3,7, new Mat(),5);

        MatOfPoint2f firstMatOfPoint2F = new
MatOfPoint2f(firstMatOfPoint.toArray());
        MatOfPoint2f secondMatOfPoint2F = new MatOfPoint2f();

        Mat mask = Mat.zeros(previousFrame.size(), previousFrame.type());

        while (true) {
            Mat frame = new Mat(), currentGrayGradient = new Mat();
            capture.read(frame);
            if (frame.empty()) {
                break;
            }
            Imgproc.cvtColor(frame, currentGrayGradient,
Imgproc.COLOR_BGR2GRAY);

            MatOfByte status = new MatOfByte();
            MatOfFloat err = new MatOfFloat();
            TermCriteria criteria = new TermCriteria(TermCriteria.COUNT +
TermCriteria.EPS,10,0.03);

            Video.calcOpticalFlowPyrLK(previousGrayGradient,
currentGrayGradient, firstMatOfPoint2F, secondMatOfPoint2F, status, err, new
Size(15,15),2, criteria);

            byte[] StatusArr = status.toArray();
            Point[] firstMatOfPoint2FArr = firstMatOfPoint2F.toArray();
```

```
            Point[] secondMatOfPoint2FArr = secondMatOfPoint2F.toArray();

            ArrayList<Point> pointsToAnalyzeNext = new ArrayList<>();

            for (int i = 0; i < StatusArr.length ; i++ ) {
                if (StatusArr[i] == 1) {
                    pointsToAnalyzeNext.add(secondMatOfPoint2FArr[i]);
                    Imgproc.line(mask, secondMatOfPoint2FArr[i],
firstMatOfPoint2FArr[i], colors[i],2);
                    Imgproc.circle(frame, secondMatOfPoint2FArr[i],2,
colors[i],-1);
                }
            }

            Mat img = new Mat();
            Core.add(frame, mask, img);
            HighGui.imshow("Frame", img);

            HighGui.waitKey(30);


            previousGrayGradient = currentGrayGradient.clone();
            Point[] pointsToAnalyzeNext_arr = new
Point[pointsToAnalyzeNext.size()];
            pointsToAnalyzeNext_arr =
pointsToAnalyzeNext.toArray(pointsToAnalyzeNext_arr);
            firstMatOfPoint2F = new MatOfPoint2f(pointsToAnalyzeNext_arr);
        }
    }

    public static void generateColors() {
        Random random = new Random();

        for (int i = 0 ; i < colors.length; i++) {
            int redScalar = random.nextInt(256);
            int greenScalar = random.nextInt(256);
            int blueScalar = random.nextInt(256);

            colors[i] = new Scalar(redScalar, greenScalar, blueScalar);
        }
    }
}
```

Додаток Б (обов’язковий) Приклад роботи програми

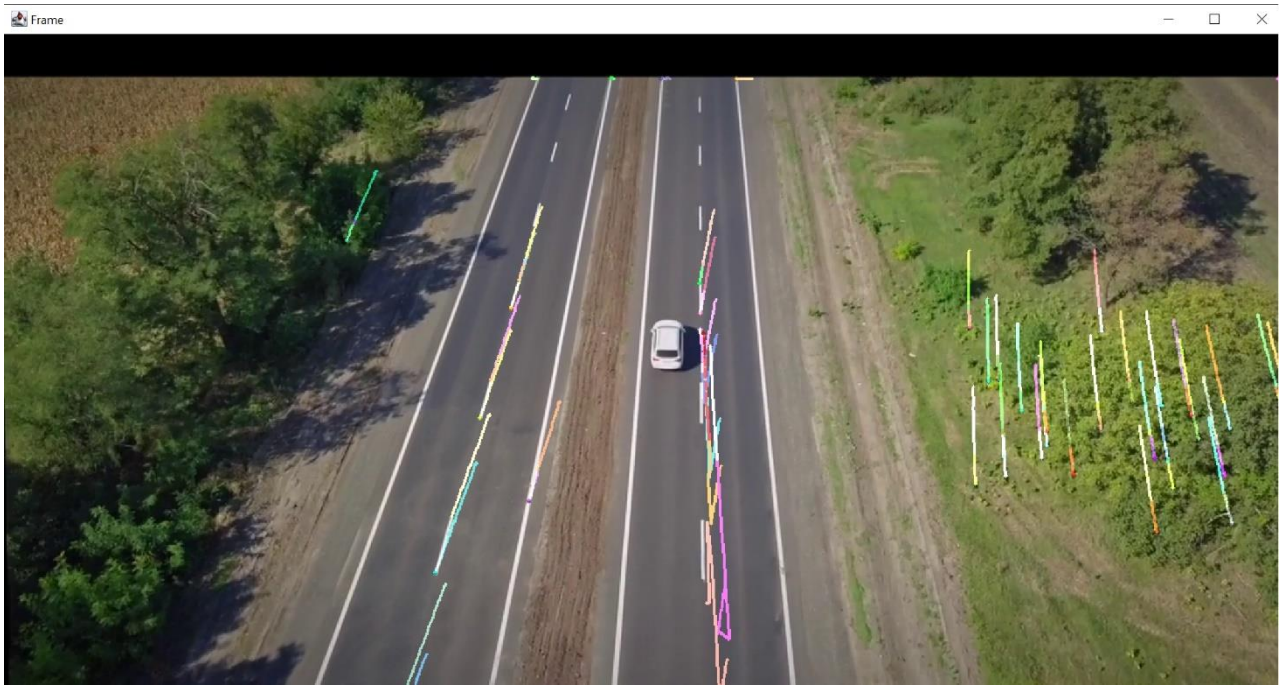


Рис. Д.Б.1 Аналіз руху машини, дерев та розмітки на дорозі

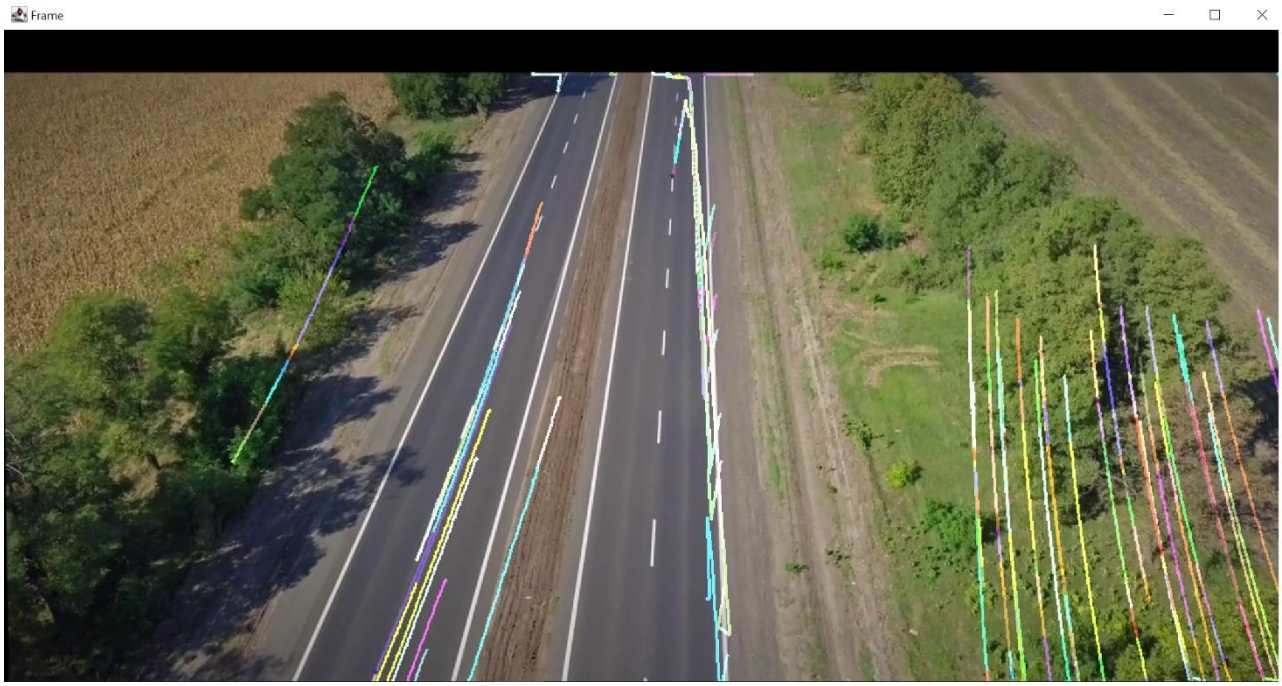


Рис. Д.Б.2 Похибка на верхній частині екрану, коли рухомі пікселі виходять за границі