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

Кафедра інформатики

3D РЕКОНСТРУКЦІЯ СЦЕНИ ЗА ВІДЕО З ДЕКІЛЬКОХ КАМЕР

3D SCENE RECONSTRUCTION FOR VIDEO FROM MULTIPLE CAMERAS

Текстова частина до магістерської роботи
за спеціальністю «Інженерія програмного забезпечення»

Виконав: студент 2-го року навчання
Томащук Вадим Миколайович

Керівник: Крюкова Г.В., кандидат
фізико-математичних наук, доцент

Рецензент_____
(прізвище та ініціали)

Магістерська робота захищена
з оцінкою «_____»

Секретар ДЕК _____

«____» _____2020 р.

Київ 2020

Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики
Магістерська програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики, доц.,
к.ф.-м.н.

_____С. С. Гороховський
(підпис)
7 листопада 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на дипломну роботу

студенту 2 року навчання МП «Інженерія програмного забезпечення» інформатики
Томащуку Вадиму Миколайовичу
на тему 3D реконструкція сцени за відео з декількох камер

Зміст ТЧ до магістерської роботи:
Зміст
Анотація
Актуальність
Вступ
1. Виявлення та опис особливостей та особливості застосовних алгоритмів
1.1. Алгоритми виявлення та опису особливостей об’єктів та їх порівняння
2. Підготовка вхідних даних для 3D реконструкції сцени
2.1. Епіполярна геометрія, триангуляція та вирівнювання (Epipolar Geometry, Triangulation and Rectification)
2.2. Відслідковування точок (Point tracking)
2.3. Оцінка позиції (Pose estimation)
3.Програмна реалізація 3D реконструкції сцени за відео з декількох камер
4. Аналіз практичної частини та простір для вдосконалення
Висновки
Список літератури
Додатки

Дата видачі 25 жовтня 2019 р.

Керівник доц., к.ф.-м.н. Г.В. Крюкова

(підпис)
Завдання отримав В. М. Томащук
(підпис)

Календарний план виконання роботи

Тема: _____

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	10.11.2019	
2.	Огляд технічної літератури за темою роботи.	15.12.2019	
3.	Оцінити можливість реалізації застосування, врахувавши оцінку аналізу відеопотоків.	25.01.2020	
4.	Поетапне проектування застосування.	23.02.2020	
5.	Підбір алгоритмів для спроектованої декомпозованої задачі.	15.03.2020	
6.	Програмування розробленого алгоритму	15.04.2020	
7.	Виконання порівняльного аналізу результатів експериментального дослідження.	25.04.2020	
8.	Написання пояснювальної роботи.	05.05.2020	
9.	Створення слайдів для доповіді та написання доповіді.	20.05.2020	
10.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист магістерської роботи.	31.05.2020	
11.	Коригування роботи за результатами попереднього захисту.	05.06.2020	
12.	Остаточне оформлення пояснювальної роботи та слайдів.	10.06.2020	
13.	Захист магістерської роботи (проекту).	16.06.2020	

Студент _____

Керівник _____

“_____”_____

ЗМІСТ

АНОТАЦІЯ

Робота складається з чотирьох розділів. В першому розглянуто базові поняття виявлення та опису особливостей об’єктів, які є основою для будь якого обраного підходу 3D реконструкції сцени, а також розглянуто найпопулярніші алгоритми для кращого сприйняття цього етапу в процесі розробки. В другому розділі описана теоретична база епіполярної геометрії, вирівнювання та триангуляції, а також згадано про алгоритми відстежування та оцінки позиції об’єкта. Третій розділ повністю присвячений поетапній розробці практичного застосування та частково описується необхідна теорія, така як побудова карт невідповідностей та глибини. В четвертому розділі звернено увагу на недоліки проведеної роботи, оцінено ефективність практичного застосування та простір для подальшого вдосконалення.

ВСТУП

Насьогодні дуже активно розвивається сфера комп’ютерного зору, яка вирішує величезну кількість нетривіальних задач, таких як доповнена реальність, розпізнавання об’єктів, створення 3D моделей тощо. Для вирішення цих проблем уже створено багато алгоритмів (деякі з них залишаються актуальними вже понад 20 років, як, наприклад, SIFT), але попри це вони постійно вдосконалюються, комбінуються та допомагають вирішити нові задачі комп’ютерного зору.

Ми живемо в 3D-світі, де порожній і зайнятий простір визначається фізичною присутністю предметів. Щоб успішно орієнтуватися всередині світу та взаємодіяти з ним, важливо розуміти як тривимірну геометрію, так і семантику навколишнього середовища.[18] Людина сприймає ці речі вже на інтуїтивному рівні, наприклад, коли вона бачить передню частину будинку, то вона не задумується, що в нього є і задня частина. Наразі люди працюють над вирішенням багатьох питань, щоб навчити комп’ютер розпізнавати 3D світ, а також створювати його віртуальні моделі, які можна модифікувати. Аналогічно, роботи повинні розпізнавати тривимірні об’єкти та навколишнє середовище та “бачити” їх, щоб взаємодіяти з ними. Вирішення таких задач дозволяє виділяти смислове значення предметів у сцені та виконувати завдання високого рівня, наприклад, пошук об'єктів.

Так у цій царині має місце така задача комп’ютерного бачення, як 3D реконструкція сцени. В основі лежить ідея відтворення усіх об’єктів у визначеному просторі та самого простору у віртуальній тривимірній формі. Дану реконструкцію, завдяки дослідженням і розробкам, можна проводити як із зображень, так і з потоку відео. Якщо ми декомпозуємо озвучені два завдання, то всього лише за нашого бажання та наукового підходу ми зможемо знайти схожі підходи в реалізації та в певній ітеративній послідовності декомпозовані задачі будуть перегукуватись.

Сфера 3D реконструкції сцени є досить добре досліджена, але якщо проглянути всі дослідження, то кожне з них заточене під конкретні вимоги, які вимагаються для конкретної сфери (наприклад, реконструкції на вулиці або в приміщенні, з кількох зображень або з відеопотоку, дрібних деталей або великих об’єктів тощо).

Наша задача полягає дослідити можливість та створення 3D реконструкції сцени за відео з декількох камер в режимі реального часу. В даній роботі будуть поетапно досліджено найголовніші аспекти реалізації, на які буде декомпозовано основне завдання. Також ми розглянемо ефективність нашого дослідження та реалізації, визначимо моменти, яким буде необхідне вдосконалення та за можливості буде запропонована альтернативний підхід до задачі. Зокрема особливу увагу буде приділено трансляції відео з кількох камер в режимі реального часу та можливість його аналізу, безпосередньо процесу 3D реконструкції сцени та графічному представленню. Вкінці розглянемо можливість подальших вдосконалень.

РОЗДІЛ 1. ВИЯВЛЕННЯ ТА ОПИС ОСОБЛИВОСТЕЙ ТА ОСОБЛИВОСТІ ЗАСТОСОВНИХ АЛГОРИТМІВ

Пропоную розглянути ситуацію, коли вам потрібно витягнути з картинки, чи фотографії, один невеликий об’єкт. Якщо цей об’єкт складний, то це завдасть вам дуже багато клопоту. Звісно, зараз комп’ютери вирішують більшість проблем з графікою без всяких проблем. Потім, поєднавши такі маленькі вирізані картинки, ви можете створити власну композицію, склеївши їх. Якщо ми можемо проводити такі маніпуляції з фотографіями чи картинками, то що ви скажете щодо розпізнавання такого об’єкту чи структури та створення 3D моделі за допомогою аналізу відео в реальному часі.

Відео це потік кадрів (картинок), що швидко змінюють одна одну. Є відоме поняття кількість кадрів на секунду, що передбачає плавність і природність рухів на відео (чим більша частота кадрів на секунду, тим вища плавність та реалістичність рухів). В кінематографі використовують 24 кадри на секунду, але в реальному житті і при 10-ти карах на секунду рухи будуть видаватись реалістичними. Тож так як відео складається з картинок, то ми можемо навчитись розпізнавати об’єкти на картинці (а потім на серії картинок), і це забезпечить нам можливість аналізу відео та створення тривимірної моделі об’єкта чи структури.

Але перш ніж перейти до, так званого, аналізу відео, нам потрібно зрозуміти як же комп’ютер може розпізнавати об’єкти та виділяти їх з поміж сотень інших на одній картинці (Рис 1.1). Ми легко ідентифікуємо різні особливості, і, навіть, іноді їх важко описати словами, але ми знаємо їхнє визначення. Якщо вам вкажуть пальцем на кілька об’єктів і запитають чи можете ви їх відрізнити за певними відмінностями, то звичайно ви вкажете на більше, ніж одну особливість, які легко допомагають ідентифікувати той чи інший об’єкт. Наступним запитанням ми спробуємо пояснити ці особливості, щоб вони були зрозумілими навіть для

комп’ютера. Є моменти, коли важко описати *як люди знаходять ці особливості,* бо це ніби запрограмовано в нас в голові. Якщо ми заглянемо глибше в якісь картинки і шукатимемо різні візерунки чи патерни, ми знайдемо щось цікаве. Наприклад, візьміть нижче зображення:

Рисунок 1.1 - Визначення особливостей та кутів на одній картинці

Зверху є три вирізаних фігури з даної фотографії. Чи можете ви точно ідентифікувати розташування цих об’єктів на повних фото?

Помітно, що на картинках А і Б є межа з небом, тому це частини даху, що ідентифікувати не складно. З картинкою С складніше, а основним ідентифікатором

є колони, хоча вони покривають велику площу. Тепер, коли ми хочемо визначити конкретні місця, то ми будемо звертати увагу на особливості кутів, вигинів, дрібних деталей. На це ми також відповіли інтуїтивно, тобто шукали регіони на зображеннях, які мають максимум змін при переміщенні (на невелику відстань) у всіх напрямках навколо нього. Це буде проектуватися на комп'ютерну мову в наступних кроках. Тому пошук цих особливостей зображення називається *Feature Detection* (виявлення особливостей).

Коли ми визначили основні особливості на маленькій картинці, ми починали описувати велику фотографію, щоб визначити таку ж саму ділянку, як і на попередній. В основному ви описуєте особливості. Аналогічним чином комп'ютер також повинен описати область навколо патерна, щоб він зміг знайти її в інших зображеннях. Такий процес називається *Feature Description* (опис особливості). Отримавши функції та їх опис, ви зможете знайти однакові функції у всіх зображеннях і вирівняти їх, зшити їх або зробити все, що завгодно.[1]

1.1 SIFT, SURF, ORB алгоритми

Після загального огляду виявлення деталей та особливостей зупинимося детальніше на алгоритмах, які задовольняють виконання цих завдань. Для розуміння ми розглянемо алгоритми та їх дескриптори для того, щоб в подальших розділах визначити принципи нашого алгоритму.

1.1.1 SIFT

Алгоритмом SIFT ключові точки об'єктів спочатку витягуються з набору еталонних зображень [10] і зберігаються в базі даних. Об'єкт розпізнається в новому зображенні шляхом індивідуального порівняння кожної особливості з ознаками, записаними в базу даних, та пошуку відповідності кандидатів за допомогою обчислення евклідової відстані векторів їхніх ознак. З повного набору збігів ідентифікуються підмножини ключових точок, які підтверджують об'єкт та його розташування, масштаб, орієнтацію в новому зображенні, щоб відфільтрувати релевантні співпадіння. Визначення послідовних кластерів здійснюється швидко за допомогою ефективної хеш-таблиці втілення узагальненої трансформації Хаффа. Кожен кластер із 3 або більше функцій, які погоджуються щодо об'єкта та його позиції, потім підлягає подальшій детальній перевірці моделі, а згодом залишків відкидається. Нарешті, обчислюється ймовірність того, що певний набір ознак свідчить про наявність об'єкта, враховуючи точність прилягання та кількість ймовірних помилкових збігів. Збіги об'єктів, які пройшли всі ці тести, можна з високою впевненістю визначити як правильні. [11]

Покроково даний процес можна розписати так [1]:

1) Масштабно-просторове визначення екстремумів (*Scale-space Extrema Detection*) - обчислення та знаходження особливостей, зокрема кутів різної величини. Для цього використовують збільшення та зменшення вікон аналізу за допомогою масштабно-просторової фільтрації, яка містить лапласіан гаусівського. Так як знаходження лапласіан гаусівського є досить дорогою операцією, то в алгоритмі SIFT використовується різниця гаусіанів.

2) Локалізація ключових точок (*Keypoint Localization*) - після виявлення потенційних ключових точок їх потрібно вдосконалити для отримання більш точних результатів. Було використано розширення ряду Тейлора для масштабування простору, щоб отримати більш точне розташування екстремумів, і якщо інтенсивність у цьому екстремумі менша за порогове значення (0,03 за даними

документації OpenCV), то його відкидають. Цей поріг у OpenCV називається *contrastThreshold*.

3) Задача координування (*Orientation Assignment*) - побудова орієнтаційної гістограми, завдяки якій ми зможемо обертати зображення без загрози порушити стабільність співпадінь

4) Дескриптор ключових точок (*Keypoint Descriptor*)

5) Узгодження ключових точок (*Keypoint Matching*)

1.1.2 SURF

У SIFT було апроксимовано Лапласіана Гаусса з різницею Гаусса для знаходження. SURF йде трохи далі і наближає LoG до Box Filter. Нижче зображено демонстрацію такого наближення[1]. Однією з великих переваг цього наближення є те, що згортання з Box Filter можна легко обчислити за допомогою цілих зображень. І це можна зробити паралельно для різних масштабів.[1]

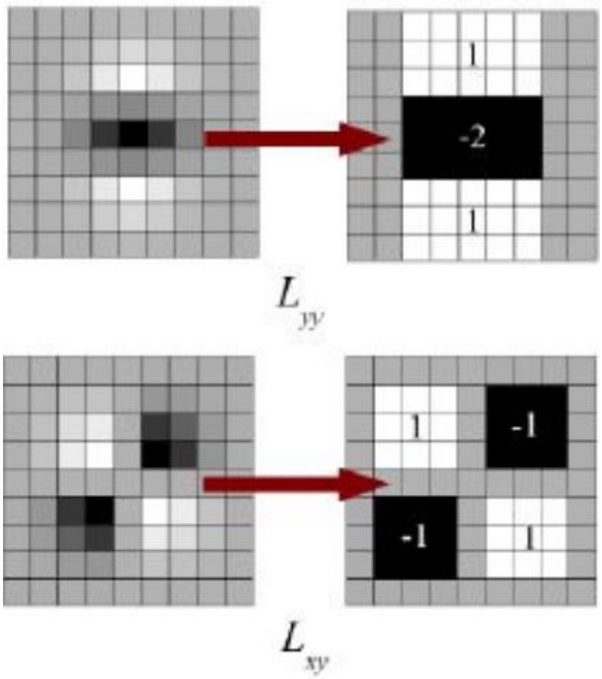

Рисунок 1.2 - Box Filter [1]

Виявлення особливих точок алгоритмом SURF засноване на обчисленні детермінанта матриці Гессе (гессіан H). Значення гессіан використовується для знаходження локального мінімуму або максимуму яскравості зображення. У цих точках значення гессіан досягає екстремуму. Розрахунок похідних відбувається за допомогою згортки пікселів зображення з фільтрами, представленими на Рис. 1.2, де білі області відповідають значенням +1, чорні - значенням -2 (на другому фільтрі - значенням -1), сірі - нулю.[5]

Для призначення орієнтації SURF використовує вейвлет-відповіді в горизонтальному та вертикальному напрямку для сусідства розміром 6s. До нього також застосовуються адекватні гуасійні ваги. SURF надає таку функціональність під назвою Upright-SURF або U-SURF. Він покращує швидкість і є надійним до $\pm 15°$.[1]

Для опису особливостей SURF використовує wavelet-відповіді в горизонтальному та вертикальному напрямку (знову ж таки, використання цілісних зображень полегшує роботу). Поруч розміром 20X20 взято навколо ключової точки, де розмір s. Він поділений на 4x4 субрегіони. Тут спрацьовує правило - зменшення розмірності призводить до збільшення швидкості.[1]

Ще одним важливим удосконаленням є використання знака Лапласіана (слід Гессіанської матриці) для основної точки інтересу. Він не додає витрат на обчислення, оскільки він обчислюється під час виявлення. Це дозволяє пропускати обчислення обернених ситуацій, якщо такі вже було виконано.[1]

Якщо узагальнити, то, внаслідок покращення кожного кроку алгоритму SIFT, ми отримали пришвидшений алгоритм SURF. SURF хороший для обробки розмитих зображень та зображень з обертанням, але погано себе покаже при зміні освітлення.

1.1.3 ORB

ORB - це в основному синтез детектора ключових точок FAST та дескриптора BRIEF з багатьма модифікаціями для підвищення продуктивності. Алгоритм спочатку використовує FAST для пошуку ключових точок, а потім застосовує кутовий показник Гарріса, щоб знайти серед них N точок. Він також використовує піраміду для створення багатомасштабних функцій. Але одна проблема полягає в тому, що FAST не обчислює орієнтацію. Тому автори тут теж зробили серйозні зміни для коректної роботи зображень з обертанням.

У оригінальній публікації зазначено, що ORB набагато швидше, ніж SURF і SIFT, а дескриптор ORB працює краще, ніж SURF. ORB - це хороший вибір пристроїв малої потужності для панорамного зшивання тощо.[6]

1.2 Підсумки опису алгоритмів виявлення особливостей

Одним з основних відмінностей запропонованої структури системи розпізнавання є те, що методи SIFT і SURF є запатентованими [4], тому їх не можна безоплатно застосувати для розробки застосування. Також результати порівняння популярних алгоритмів SIFT і SURF показують, що застосування методики розпізнавання на базі лісу рандомних дерев (ще один алгоритм, який було розгллянуто російськими дослідниками) дають кращі оцінки якості розпізнавання при більш високих вимогах до оперативної пам'яті і більшій, але цілком прийнятному часу роботи системи в залежності від розмірності дерева [4]. До того ж вони є повільними і добре спрацюють для статичних порівнянь, а не для динамічних, необхідних для нашої роботи. ORB за свідченнями авторів є швидшою альтернативою для SIFT та SURF, та при цьому безплатною. Тобто ми бачимо, що є багато ефективних алгоритмів, які можна використати, або ж запозичивши певні кроки з існуючих алгоритмів - написати свою реалізацію.

РОЗДІЛ 2. ПІДГОТОВКА ВХІДНИХ ДАНИХ ДЛЯ 3D РЕКОНСТРУКЦІЇ СЦЕНИ

2.1 Епіполярна геометрія, триангуляція та вирівнювання (Epipolar Geometry, Triangulation and Rectification)

Процеси описані в даному підрозділі необхідні, для правильної ідентифікації спільних точок на парі зображень. Щоб зрозуміти процес, ми покроково розглянемо епіполярну геометрію, вирівнювання зображень та триангуляції.

2.1.1 Епіполярна геометрія

Часто в геометрії декількох зображень виникають цікаві взаємозв'язки між декількома камерами, 3D-точкою та проекціями цієї точки в площині зображення камери. Геометрія, що стосується камер, вказує на 3D та відповідні спостереження, називається епіполярною геометрією стерео пари.[8] Давайте розглянемо зображення, на якому змодельовано встановлення двох камерам, що знімають одну і ту ж сцену (Рисунку 1.3).

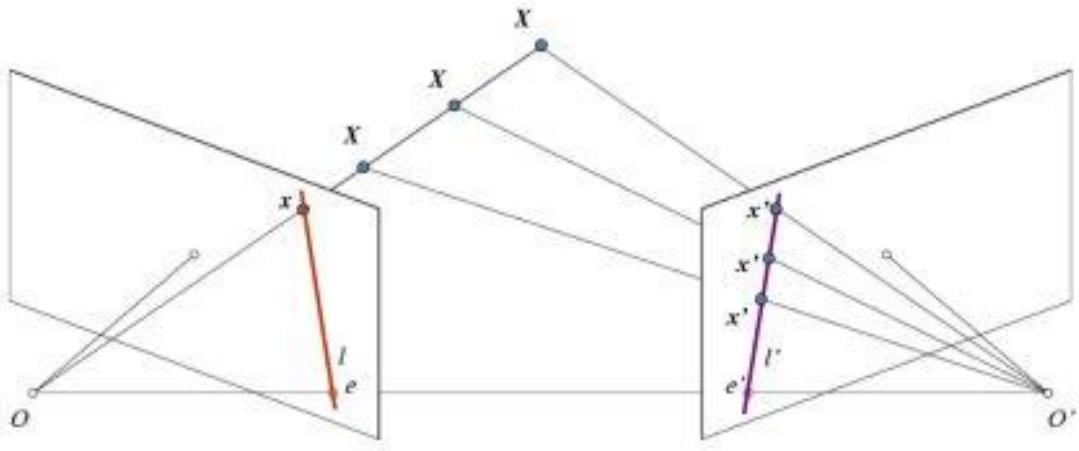

Рисунок 2.1 - Дві камери спрямовані на одну сцену з різних ракурсів [1].

Як показано на рисунку 1.3, стандартне встановлення епіполярної геометрії включає в себе дві камери, що спостерігають ту саму 3D-точку X, проекція якої у

кожній з площин зображення розташована в x та x' 'відповідно. Центр камери знаходиться в O та O', а лінія між ними називається базовою лінією. Назвемо площину, визначену двома центрами камери та точкою X, епіполярною площиною. Місця, де базова лінія перетинає дві площини зображення, відомі як епіполюси e та e'. Нарешті, лінії, визначені перетином епіполярної площини та двох площин зображень відомі як епіполярні лінії (ми їх називатимемо епілайнами, з англ. *epiline*). Епіполярні лінії мають властивість, що вони перетинають базову лінію у відповідних епіполях у площині зображення.[8]

Епіполярне обмеження (Epipolar Constraint) свідчить, що «правильна відповідність повинна лежати на епіполярній лінії».[9]

- x та X можуть лежати будь-де на промені від O через x.
- Усі епіполярні лінії проходять через епіполюс камери.

Зображення цього променя на зображенні правого зображення є епіполярною лінією через відповідну точку x'. Тобто щоб знайти точку відповідності на іншому зображенні, вам не потрібно шукати все зображення, а просто шукати по епілайн.[9]

Важливість епіполярного обмеження:

- Відповідні точки завжди повинні лежати на сполучених епіполярних лініях.
- Пошук відповідностей зводиться до 1D проблеми.
- Дуже ефективне відкидання помилкових співпадінь внаслідок оклюзії [9]

O і O' - центри камер. З налаштування, наведеного вище, можна побачити, що проекція правої камери O' розташована на лівому зображенні в точці e. Її називають епіполюсом. Епіполюс - точка перетину лінії через центри камери та площини зображення. Аналогічно e' є епіполюсом лівої камери. У деяких випадках ви не

зможете знайти епіполюси на зображенні, вони можуть знаходитись поза зображенням (це означає, що одна камера не бачить іншу).[1]

Усі епілайни проходять через епіполюс площини камери. Отже, щоб знайти місце розташування епіполюса, ми можемо знайти багато епілайнів і знайти їх точку перетину.[1]

Тож на цьому етапі ми зосередимось на пошуку епіполярних ліній та епіполюсів. Але щоб їх знайти, нам потрібні ще два інгредієнти: фундаментальна матриця (F) та основна матриця (E). Основна матриця містить інформацію про переміщення та обертання, які описують розташування другої камери відносно першої у глобальних координатах (Рис 1.4.)[1]:

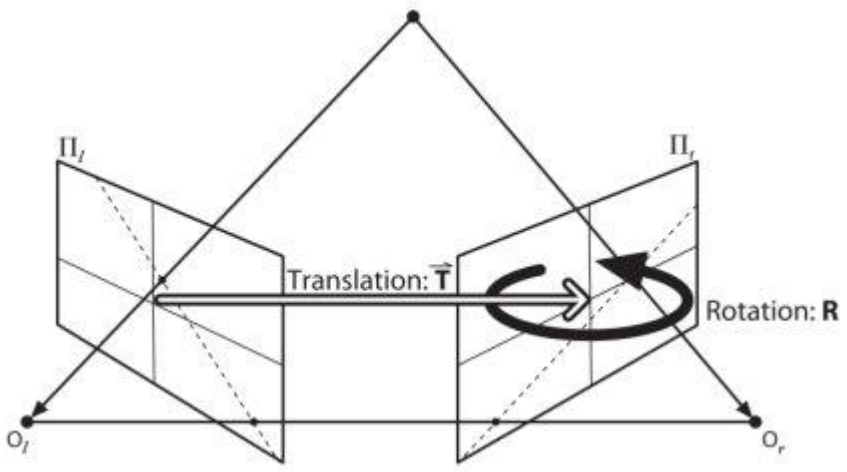

Рисунок 2.2 [1]

Але ми вважаємо за краще вимірювання проводити в піксельних координатах, правда? Фундаментальна матриця містить ту саму інформацію, що і основна матриця, тому додатково до інформації про властивості обох камер, щоб ми могли співвідносити дві камери в піксельних координатах. (Якщо ми використовуємо

випрямлені зображення та нормалізуємо точку шляхом ділення на фокусні відстані, F = E). Простими словами, фундаментальна матриця F, відображає точку в одному зображенні на лінії (епілайні) на іншому зображенні. [1]

При використанні OpenCV для ведення обчислень епіполярної геометрії є можливість використовувати *8-th point algorithm* або *RANSAC* (складніші обчислення, але кращий результат).

2.1.2 Вирівнювання (rectification)

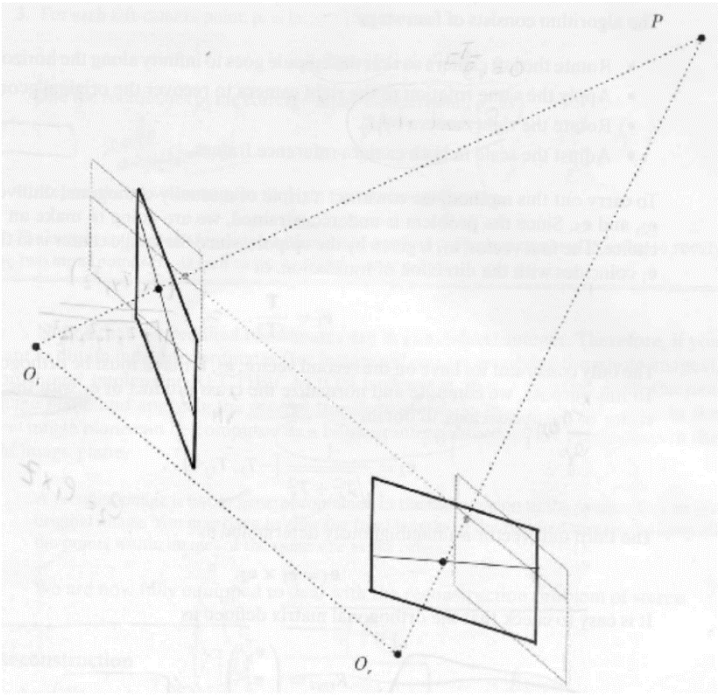

Рисунок 2.3 - Вирівнювання площини зображень

Вирівнюванням називають перетворення кожного зображення таким чином, що пари сполучених епіполярних ліній стають колінеарними та паралельними горизонтальній осі.

Пошук відповідних точок стає набагато простішим у випадку випрямлених зображень: щоб знайти ліву точку, відповідну (i_l, j_l), нам просто потрібно подивитися на праву картинку по лінії сканування $j = j_l$.

Розмінність зображень є лише у напрямку x (немає розбіжності y). Тобто після вирівнювання ми дивимось на два зображення, між якими проведені епіполярні лінії, які по суті сполучають відповідні точки на двох зображеннях. Процес можна подивитись на рисунку 2.4.

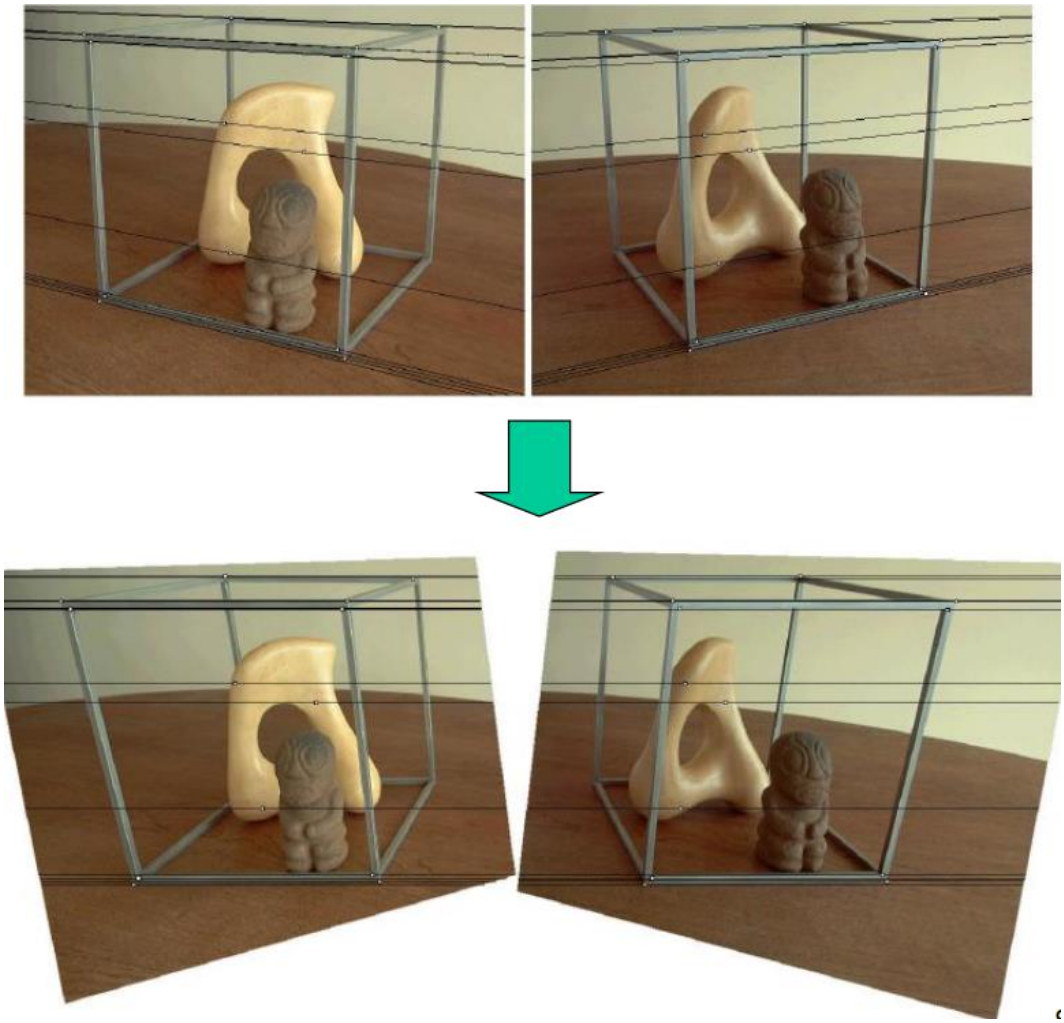

Рисунок 2.3 - Зображення до і після вирівнювання

2.1.3 Триангуляція точок (triangulation)

Тепер власне перейдемо до того, як визначити тривимірні координати точки за координатами її проекцій. Цей процес в літературі називається тріангуляції.

Нехай є дві відкалібровані камери з матрицями P1 і P2. x1 і x2 - однорідні координати проекцій деякої точки простору X. Тоді можна скласти наступну систему рівнянь [7]:

$$\begin{cases} x_1 = P_1 X \\ x_2 = P_2 X \end{cases}$$

На практиці для вирішення цієї системи застосовується наступний підхід. Векторно множать перше рівняння на x1, друге на x2, позбавляються від лінійно залежних рівнянь і призводять систему до виду $AX = 0$, де A має розмір 4×4. Далі можна виходити з того, що вектор X є однорідними координатами точки, і покласти його останню компоненту рівною 1 і вирішувати отриману систему з трьома рівняннями з трьома невідомими. Альтернативний спосіб - взяти будь-який ненульовий розв'язок системи $AX = 0$, наприклад обчислене, як сингулярний вектор, який відповідає найменшому сингулярного числа матриці A. [7]

Хоча цей процес виглядає як прямим, так і математично обгрунтованим, на практиці він не дуже добре працює. У реальному світі тому, що спостереження і шум шуму та параметри калібрування камери не є точними, а пошук точки перетину може бути проблематичним. У більшості випадків вона взагалі не буде існувати, оскільки дві лінії можуть ніколи не перетинатися.[12]

Попри явні перешкоди у використанні триангуляції є різні алгоритми його використання, як от лінійний чи нелінійний. А також за допомогою триангуляції знаходять рішення для проблеми руху. Це описано в роботі Stereo Systems and Structure from Motion, Kenji Hata and Silvio Savarese [12].

2.2. Відслідковування точок та об’єктів

Визначення звучить конкретно, але в комп’ютерному зорі та машинному навчанні відстеження - це дуже широкий термін, який охоплює концептуально подібні, але технічно різні ідеї. Тому в цьому розділі ми розглянемо загальні аспекти відстеження об’єктів та наведемо приклади простих алгоритмів,

використання яких можна дослідити за допомогою бібліотеки OpenCV, або ж написати самому.

Якщо ви коли-небудь грали з розпізнаванням обличчя OpenCV, ви знаєте, що воно працює в режимі реального часу, і ви можете легко виявити обличчя в кожному кадрі. Давайте вивчимо різні причини, за якими ви можете відстежувати об’єкти у відео, а не робити повторні розпізнавання.[14]

Відстеження працює швидше, ніж *розпізнавання*: Зазвичай алгоритми відстеження швидші, ніж алгоритми виявлення. Причина проста. Коли ви відстежуєте об'єкт, виявлений у попередньому кадрі, ви багато знаєте про зовнішній вигляд об'єкта. Ви також знаєте розташування в попередньому кадрі та напрямок та швидкість його руху. Отже, у наступному кадрі ви можете використовувати всю цю інформацію, щоб передбачити місце розташування об'єкта в наступному кадрі та здійснити невеликий пошук навколо очікуваного місця розташування об'єкта, щоб точно знайти об’єкт. Хороший алгоритм відстеження використовує всю інформацію, яку він має про об'єкт до цього моменту, тоді як алгоритм виявлення завжди починається з нуля. Також алгоритми відстеження також є звичними для накопичення помилок, і обмежувальне поле, що відстежує об'єкт, повільно зміщується не на нашу користь, який він відстежує. Щоб виправити ці проблеми з алгоритмами відстеження, алгоритм розпізнавання запускається досить часто разом із відстежуванням. [14]

Відстеження може допомогти, коли *виявлення* не вдається: якщо ви запускаєте детектор обличчя на відео, а обличчя людини закривається предметом, то детектор обличчя, швидше за все, не спрацює. Хороший алгоритм відстеження, з іншого боку, буде обробляти певний рівень оклюзії.

Відстеження зберігає ідентичність: вивід виявлення об'єкта - це масив прямокутників, що містять об'єкт. Так при на першій картинці точка може

знаходитись на 10-ій позиції масиву, а на наступній - на 17-ій, і нам це відомо, а відтак точка не втрачається. [14]

2.2.1 Алгоритмізація Meanshift та CAMshift

Ідея, що стоїть за Meanshift, проста. Наприклад, у вас є набір точок. (Це може бути розподіл пікселів на зразок зворотної проекції гістограми). Вам надається невелике вікно (наприклад, коло), і вам доведеться перемістити це вікно в область максимальної щільності пікселів (або максимальної кількості точок).

Ми можемо охопити роботу алгоритму кластеризації середнього зсуву за допомогою наступних кроків [13]:

1. Ідентифікація точок даних, приналежних власному кластеру.
2. Алгоритм обчислює центроїди.
3. Оновлення розташування нових центроїдів.
4. Тепер процес буде переглянуто і переміщено до області вищої щільності.
5. Алгоритм буде зупинений, як тільки центроїди досягнуть місця, звідки не зможуть рухатися далі.

У цього алгоритму існує проблема, і вона помітна, коли об’єкт приближається до камери, а відповідно збільшується. Наше вікно завжди має однаковий розмір, коли об’єкт знаходиться далі, і він дуже близька до камери. Це не добре. Нам потрібно пристосувати розмір вікна до розміру та обертання цілі. Рішення було запропоновано "OpenCV Labs", і воно називається CAMshift (Continuously Adaptive Meanshift), опублікованим Гарі Бредським у своїй роботі “Computer Vision Face Tracking for Use in a Perceptual User Interface”.[1]

CAMshift спочатку застосовує Meanshift. Після того, як засіб швидкого зближення сходиться, воно оновлює розмір вікна як [1]:

$$s = 2 \times \sqrt{\frac{M_{00}}{256}}$$

Він також обчислює орієнтацію найбільш підходящого вікна. Потім він знову застосовує Meanshift з новим масштабним вікном пошуку та попереднім розташуванням вікна. Процес продовжується до досягнення необхідної точності.[1]

2.3 Оцінка позиції об’єкта

На сьогоднішній день доповнена реальність є однією з найважливіших досліджень у галузі комп’ютерного зору та робототехніки. Найбільш елементарною проблемою в доповненій реальності є оцінка позиції камери щодо об'єкта у випадку комп’ютерного зору, щоб зробити пізніше 3D-рендерінг або в разі роботизації отримати позу об'єкта, щоб зрозуміти його і зробити деякі маніпуляції . Однак це не тривіальна проблема для вирішення через те, що найбільш частою проблемою в обробці зображень є обчислювальна вартість застосування безлічі алгоритмів або математичних операцій для вирішення проблеми, яка є базовою для людини. [15]

Наступні пояснення були взяті зі статті про оцінку позиції об’єкта в режимі реального часу[15]. Дана стаття розміщена в бібліотеці OpenCV з інструкціями реалізації. Там описано покроково наступні частини реалізації на прикладі алгоритму. Звісно можна вносити свої зміни в кожен з кроків та покращити реалізацію [15]:

1. Зчитати 3D текстуровану модель об'єкта та об’єктну сітку.
2. Візьміть дані з камери чи відео.
3. Витягніть функції та дескриптори ORB зі сцени.

4. Зіставляйте дескриптори сцени з дескрипторами моделей, використовуючи співпадіння Флана.

5. Оцінка пози за допомогою PnP + Ransac.

6. Лінійний фільтр Кальмана для відхилення поганих поз.

Розглянемо спрощену проблему, коли ми орієнтуємось на параметри калібрації, яку називають Perspective-*n*-Point (Рисунок 2.4).

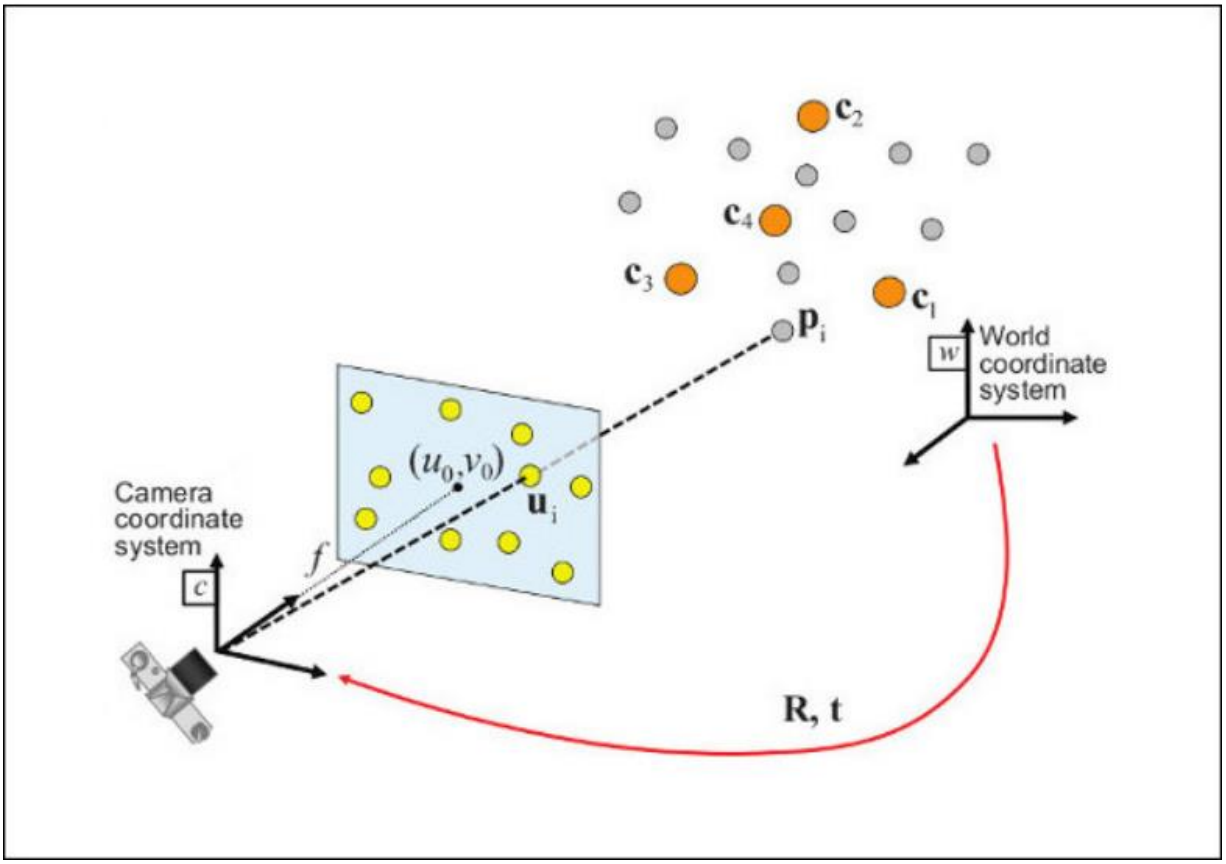

Рисунок 2.4 - Проблема Perspective-*n*-Point

Враховуючи набір відповідностей між 3D-точками p_i та їх 2D-проекціями на зображення, ми прагнемо відновити позицію (R і t) камери відповідно до системи координат. і фокусної відстані f.

За допомогою OpenCV, за необхідності, можна дослідити різні підходи до вирішення цієї проблеми, а потім за можна спроектувати 3D точки у площину зображення за формулою [15]:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Після проектування 3D точок у площину зображення та отримаємо результат, як приклад наведено Рисунку 2.5 (взято зі статті-інструкції по реалізації):

Рисунок 2.5 - Результат оцінки позиції об’єкта [15]

Якщо ми перейдемо до складніших задач, то зараз чи не найпопулярнішою задачою в цій сфері є оцінка позиції людини. Цікаве дослідження було проведено у 2017 році, на яке я звернув увагу під назвою “Recurrent 3D Pose Sequence Machines”. На рис 2.6 показано приклад результатів цієї роботи в порівнянні з різними підходами.[16]

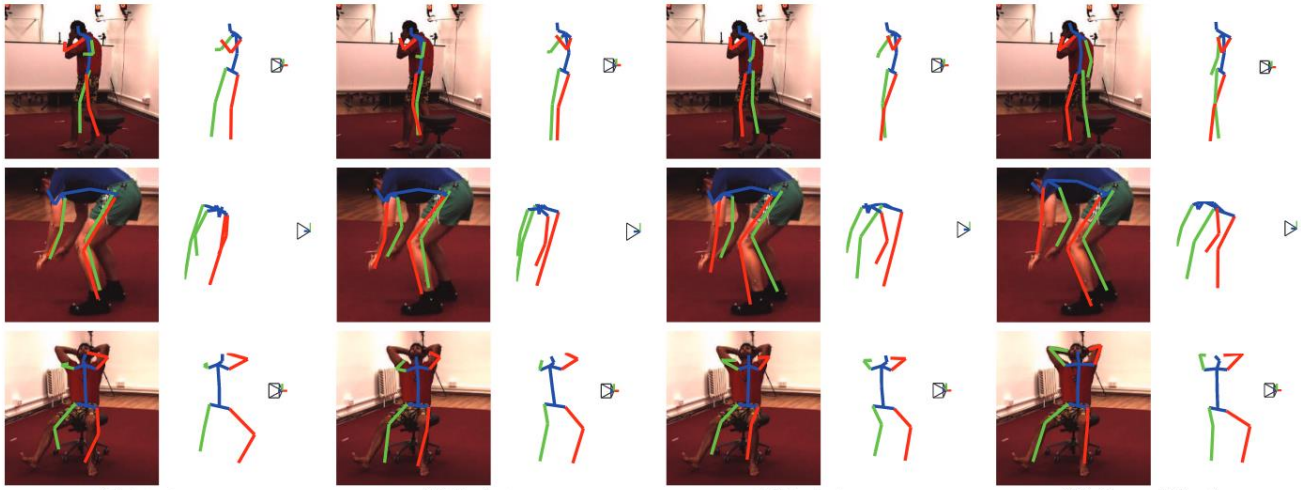

Рисунок 2.6 - Порівняння підходів у “Recurrent 3D Pose Sequence Machines” [16]

Отже, оцінка позиції допомагає оцінити та зрозуміти об’єкт, тобто його розташування, позиції точок в 3D моделі, що в свою чергу дає розуміння про глибину зображення та допомагає в побудові карти глибини (Depth Map).

РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ ТА РЕАЛІЗАЦІЯ 3D РЕКОНСТРУКЦІЇ ЗА ВІДЕО З ДЕКІЛЬКОХ КАМЕР

3.1 Вступ до практичної частини

Головна ідея та головна мета даної роботи було розглянути можливості побудови системи для створення тривимірних моделей об’єктів за допомогою відео з декількох камер. Для розуміння і дослідження принципів достатньо двох камер. Для розуміння звичайного користувача в нього є можливість запустити дві камери (наприклад, зі смартфонів), спрямовувати їх на один об’єкт, а в результаті отримати 3D реконструкцію сцени та об’єктів, яку буде змога оглянути з усіх боків.

Пропоную дещо ідеалізовано та детальніше розглянути повністю реалізоване таке застосування на прикладі використання в різних сферах. Тож уявимо, що у нас є кілька камер які розміщені, наприклад, на різних позиціях. Перша ситуація коли камери у нас нерухомі натомість можуть рухатись об'єкти по кімнаті. В результаті роботи застосування, камер та алгоритму ми отримуємо 3D модель об'єкта який рухається. топ ми можемо його оглянути комп'ютеризована з усіх боків точністю до найменших деталей. сказано з точністю до найменших деталей тому, що ми ідеалізуємо умови і вважаємо, що камери у нас постачають дуже якісну картинку, що в сучасному світі книги не є чимось дуже вражаючим. Що ж ми можемо робити з такою 3D моделлю? Для прикладу побудувавши 3D модель ми застосуємо нейронну мережу для розпізнавання об'єкта що рухається. Наприклад, ми розпізнали, що це собака, а далі алгоритм розпізнавання зрозуміє що це наша собака, і дверцята для тварин автоматично відчиняться, а потім зачиняється коли алгоритм зрозуміє якщо собака уже зайшла.

Інша ситуація, для прикладу, коли ми маємо квадратну кімнату, в яку заїжджає авто після ДТП а камери розміщені на різних позиціях з усіх боків можуть

аналізувати різні типи пошкоджень, наприклад, вмятина чи відламана деталь. Або також є секція велике приміщення з великою кількістю кімнат в кожній з яких стоять камери. І тому випадку, коли в приміщенні немає людей, немає потреби записувати відео на сервер адже воно займає дуже багато місця. Для вирішення такої проблеми достатньо щоб працювали камери навідних точках або біля вікон. зображення з цих камер буде аналізуватися алгоритмом і в разі виявлення об'єкту, який рухається, ми будемо вмикати інші камери за потреби та записувати лише той період часу коли відбувається безпосередній рух на камерах.

Можливості застосування такого підходу є дуже широкими, але зосередимось на реалізації.

Ідея реалізації була заснована на підході поступового аналізу картинок попарно з двох камер, що буде розглянуто в наступному підрозділі опису практичної частини

3.2 Реалізація передачі відео в режимі реального часу з двох камер та виділення картинок з певною частотою

Перша задача, яку нам потрібно вирішити - це задача трансляції відео з кількох камер та обробка даних з цих камер, а саме доступ до зображень які будуть діставатися з відеопотоку. Обмеження ресурсів з технічної сторони в цьому дослідженні вимагає, щоб ми скористалися камерами смартфонів, тобто у нас є два смартфона і ми будемо отримувати два відео потоки з них на комп'ютер. Завдання також полягає в тому, щоб дістати зображення якомога кращої якості, але так як ми обмежені в ресурсах через умови пандемії, то нам доводиться проводити трансляцію відео по домашній WIFI мережі.

Для початку Нам необхідно перетворити камеру смартфона на IP камеру. Для цього скористаємося одним уже готових застосувань які запускають в сервер на стороні смартфона на віддають потоки даних на запит. Мені довелося порівняти два застосування: RTSP Camera Server та IP Webcam (*рисунок 3.1*).

Рисунок 3.1 - Серверні застосування для перетворення камери смартфона на IP камеру

Порівнявши дані серверні застосування, була виявлена очевидна різниця. RTSP Camera показала себе в роботі як досить стабільний сервер який передає постійний потік зображень. Нажаль, якість була не найкращою, але з огляду на те, що дані експерименти проводяться в недостатньо підготовлених умовах, то це було цілком виправдано. Той час як IP Webcam показала себе дещо гіршої сторони, відеопотоки були з перервами і дуже часто втратачалося зображення. Хоча в деяких моментах ми отримували зображення кращі за якістю, аніж з RTSP сервера, втрата даних не була повністю виправдана, бо для нас важлива якомога менша переривчастість даних, адже нам потрібно діставати картинки з певною частотою і, відповідно, чим і більша втрата даних, тим більша можливість не дістати картинку та внести неправильні дані у 3D реконструкцію сцени.

Так як наше застосування базується на аналізі зображень попарно, то нам необхідно діставати з відеопотоків зображення з певною частотою, а для цього достатньо зрозуміти як працює передача відеопотоку з сервера на клієнтське застосування, тобто на комп'ютер. Для передачі відеопотоків ми використовуємо бібліотеку OpenCV, як і для більшості наших обчислень. Поглянемо на рисунок 3.2, на якому зображено частину коду для передачі відеопотоків.

```
def videoFromRedmi4X_RTSP():
    os.environ["OPENCV_FFMPEG_CAPTURE_OPTIONS"] = "rtsp_transport;udp"
    vcap = cv2.VideoCapture("rtsp://192.168.1.100:5554/camera", cv2.CAP_FFMPEG)
    vcap2 = cv2.VideoCapture("rtsp://192.168.1.101:5554/camera", cv2.CAP_FFMPEG)

    while(True):
        ret, frame = vcap.read()
        ret2, frame2 = vcap2.read()
        if ret == False:
            print("Frame is empty")
            break;
        if ret2 == False:
            print("Frame is empty")
            break;
        cv2.imshow('VVV', frame)
        cv2.imshow('AAA', frame2)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            cv2.destroyAllWindows()
            break;
```

Рисунок 3.2 - Процес передачі відеопотоків

Даний метод був написаний для телефона Redmi 4X, основного мобільного девайсу для досліджень в даній роботі. Ми бачимо, що потік даних проходить по протоколу UDP. Далі за допомогою методу cv2.videoCapture ми за допомогою IP адреси розв'язуємо камеру смартфона з клієнтським застосуванням на комп'ютері. Наступний процес відбувається циклі while(true) та показує як ми передаємо відео потік за допомогою постійної передачі зображень. якщо ми хочемо зупинити

трансляцію для двох камер нам достатньо натиснути кнопку Q після отримання наступного фрейму цикл зупиниться. Таким що я нам з будь-якою частотою яка нам потрібна не можемо діставати картинки з відеопотоку і використовувати їх для для аналізу нашим алгоритмом. Схожі методи які зображені на Рисунку 3.2 було реалізовано для веб камери комп'ютера та, окремо для використання лише смартфона Redmi 4X. На Рисунку 3.3 ви можете побачити результати роботи даної реалізації.

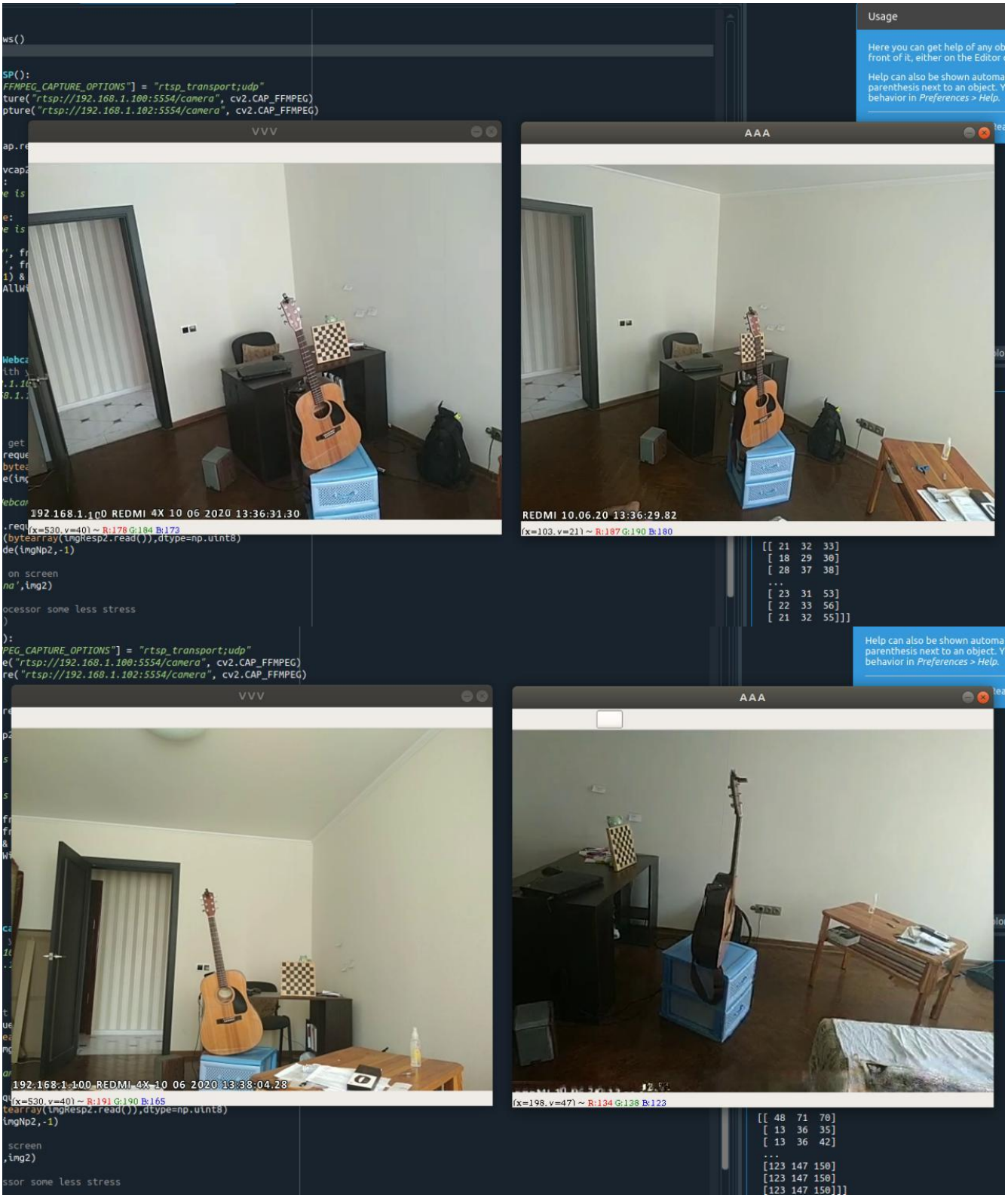

Рисунок 3.3 - Трансляція відеопотоків з двох камер на комп’ютер

Таким чином без зайвих проблем ми можемо проводити трансляцію відео з кількох камер, при цьому з можливістю аналізувати потік зображень.

3.3 3D реконструкція сцени з двох зображень

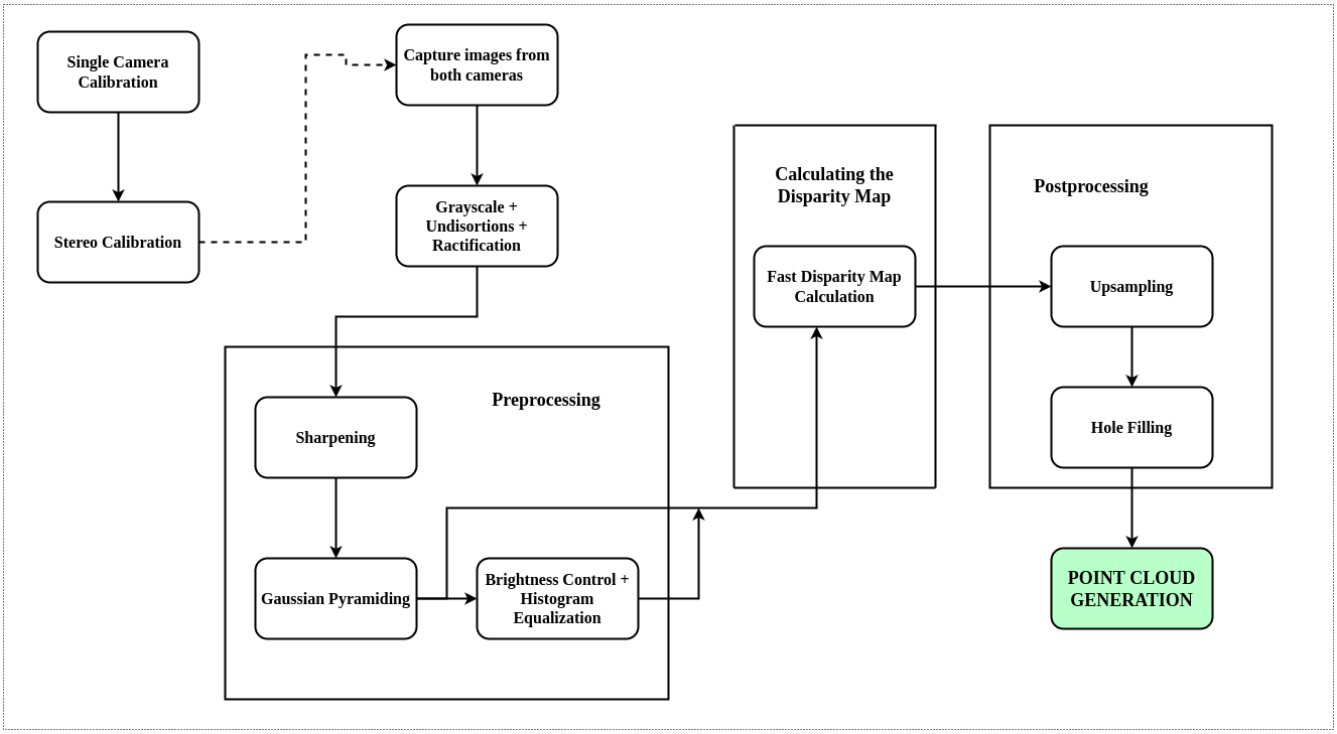

Рисунок 3.4 - Процес побудови тривимірної моделі

На Рисунку 3.4 ми можемо оглянути весь процес створення 3D реконструкції сцени, кожен крок якого буде розглянуто детальніше в наступних підрозділах. Для глибшого розуміння можна відкривати вихідний код.

Для створення тривимірної реконструкції наших зображень, необхідно знайти відстані від кожної точки сцени (простір, що покривають зображення) до камер.

Спочатку створюється так звана карта невідповідностей (Disparity Map). Для її обчислення реалізовано нескладний алгоритм відповідності блоків з використанням суми абсолютних відстаней (Sum of Absolute Distances - SAD). SAD була використана для того щоб кожному пікселю першого зображення поставити у відповідність пікселі з другого зображення. Суть алгоритму полягає в знаходженні зміщення кожного пікселя від одного зображення до іншого. Далі використовуючи метод триангуляції знайти необхідну нам точку в просторі. Тут важливо пам’ятати, що величина зсуву обернено-пропорційна до відстаней між камерами та об’єктами в просторі. Мається на увазі, що чим ближче знаходиться об’єкт тим більшим буде зсув, і, аналогічно, чим більша відстань до об’єкта, тим зсув буде меншим.[1]

Загалом є досить багато алгоритмів обчислення та створення Disparity Map, але необхідно зважати на те які ресурси ви маєте для даного обчислення, щоб максимізувати результат. Мається на увазі технічні можливості. Для даної роботи це має величезне значення, оскільки на даний момент не має доступу до комп’ютера з достатньо потужною GPU, яка б підтримувала програмно-апаратну архітектуру CUDA. Тому реалізація алгоритмів та тестування їх швидкодії проводиться на CPU, що як відомо значно погіршить результати. Наперед варто зауважити, що використовується. Переглянемо які технічні ресурси будуть використовуватись в даній роботі:

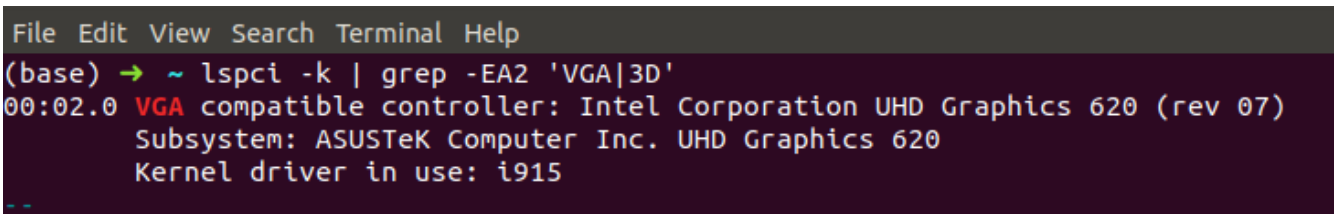

Рисунок 3.5 - Графічний контролер

В дослідженні немає відеокарти, яка б могла виконувати розподілені обчислення.

```
File  Edit  View  Search  Terminal  Help
(base) ➜  ~ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               142
Model name:          Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Stepping:            10
CPU MHz:             800.024
CPU max MHz:         4000,0000
CPU min MHz:         400,0000
BogoMIPS:            3999.93
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            8192K
NUMA node0 CPU(s):   0-7
```

Рисунок 3.6 - Властивості CPU

В розпорядженні маємо процесор Intel Core i7-8550U CPU. Це восьмиядерний процесор шостого покоління Intel Core.

Повернемось до алгоритмів для Disparity Map. Неважко зрозуміти, що попри досить хороший процесор, але за відсутності GPU для розподілених обчислень, ми будемо значно програвати в часі, витраченого на обчислення, тому, скориставшись порівняльною характеристикою Gabriele Galfrè, Arturo Cardone, Federico Sandrelli,

яка була виконана у грудні 2018 року, зупинимось на Fast Disparity Map Calculation (FDMC). На Рисунку 3.7 ми бачимо порівняння ефективності різних алгоритмів.

Algorithm	Execution Time
Basic Block Matching	$309s$
Basic Block Matching with sub-pixel estimation	326
Block Matching with Horizontal Smoothing (Dynamic Programming)	$306s$
Matrix-Based implementation of Basic Block Matching	$1s$

Рисунок 3.7 - Час виконання при обчислення Disparity Map картинки розмірами 480x720 [2]

Тепер, зробивши очевидні висновки, розглянемо алгоритм FDMC трішки детальніше, бо саме в цьому місці ми ми будемо витрачати найбільше часу на обчислення. Нажаль, Python не є дуже ефективним в цьому секторі обчислень, але для пришвидшення FDMC можна реалізувати алгоритм на основі матриці, який паралелізує обчислення значень невідповідності (disparity values) за допомогою numpy масивів.[2]

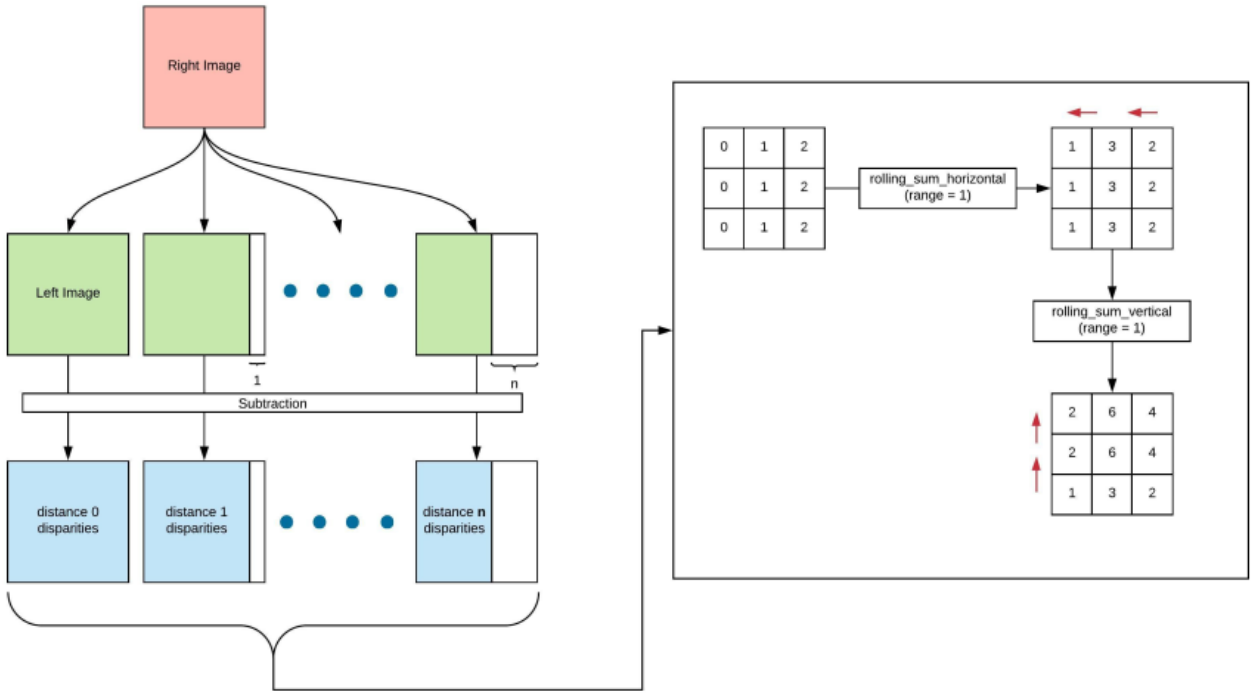

Рисунок 3.8 - Імплементація обчислення Disparity Map на основі матриць[2]

Цей алгоритм отримує на вхід праве і ліве зображення в чорнобілому діапазоні, створюючи набір n матриць розміру вихідних зображень, де n - горизонтальний діапазон порівняння, який ми використовуємо (зазвичай це близько 70). Кожна з цих матриць містить ліве зображення, зсунене на $i \in [0, n]$ пікселів вліво (ліва частина обрізана, права сторона - нульова). Від кожного з них віднімаємо потрібне зображення і приймаємо абсолютне значення віднімання. У цей момент до кожного пікселя з n матриць підсумовуємо перші n значень праворуч, а потім перші n значень під ним. На зображенні ми бачимо, як ця операція працює на простому прикладі з n = 1.[2]

В результаті ми отримали n матриць, де значення кожного пікселя в матриці є значенням SAD для відповідного пікселя правого зображення. Взявши мінімум за п'ять матриць, ми отримаємо карту диспропорційності базового блоку (Basic Block Matching disparity map).

З даного моменту кожен опис реалізацій коду (який можна переглянути в додатках до роботи), легко відслідковувати за схемою, що зображена на Рисунку 3.4.

3.3.1 Калібрація камер

Калібрація камер досить трудомісткий і надзвичайно важливий етап для реконструкції тривимірної сцени. Цей процес складається з двох етапів, кожен з яких виключає певні викривлення, які можуть спотворити сприйняття та лінії, а в результаті і звести до абсурду всю виконану роботу. Перший етап полягає в калібруванні кожної камери окремо, а другий - в калібрації двох камер разом. Це робиться тому, що певні тангенсіальні та радіальні спотворення модифікують форми об’єктів на наших зображеннях. Щоб вирішити ці проблеми, нам потрібно знайти коефіцієнти спотворень, а також матрицю камери, що включає фокусні відстані та оптичні центри. У нашій системі для калібрування камер нам потрібен набір тривимірних точок та їх відповідних 2D точок: 2D точки отримують як розташування площини зображення внутрішніх кутів шахової дошки (точніше там, де два чорних квадрата зустрічаються один з одним). Натомість 3D-точки не можна з точністю опізнати, тому їх вибирають довільним чином, припускаючи, що вони знаходяться на одній відстані по осі Z (у нашому випадку 0) з координатами XY, рівним (0,0), (1,0), (2,0) ... (6,9).[2] Ця унітарна відстань може бути помножена на реальний розмір у мм квадрата шахової дошки [3]. Таким чином, ми припускаємо, що камера рухається навколо тих фіксованих 3D точок, які відображаються на 2D координати на площині зображення відповідно до поточного положення та орієнтації камери (навіть якщо насправді це шахова дошка, яка рухається навколо,

а не камера)[2]. Для того, щоб досягти хорошого калібрування, нам потрібно досягти:

- *X Калібрація*: переміщення шахової дошки вліво та вправо, щоб шахова дошка була виявлена по горизонтальних краях поля зору.
- *Y Калібрація*: переміщенням шахової дошки вгору та вниз, щоб шахова дошка була виявлена по вертикалі.
- *Калібрування нахилу*: обертання шахової дошки під різними кутами відносно камери.
- *Калібрування розміру*: переміщення шахової дошки ближче та далі від камери.

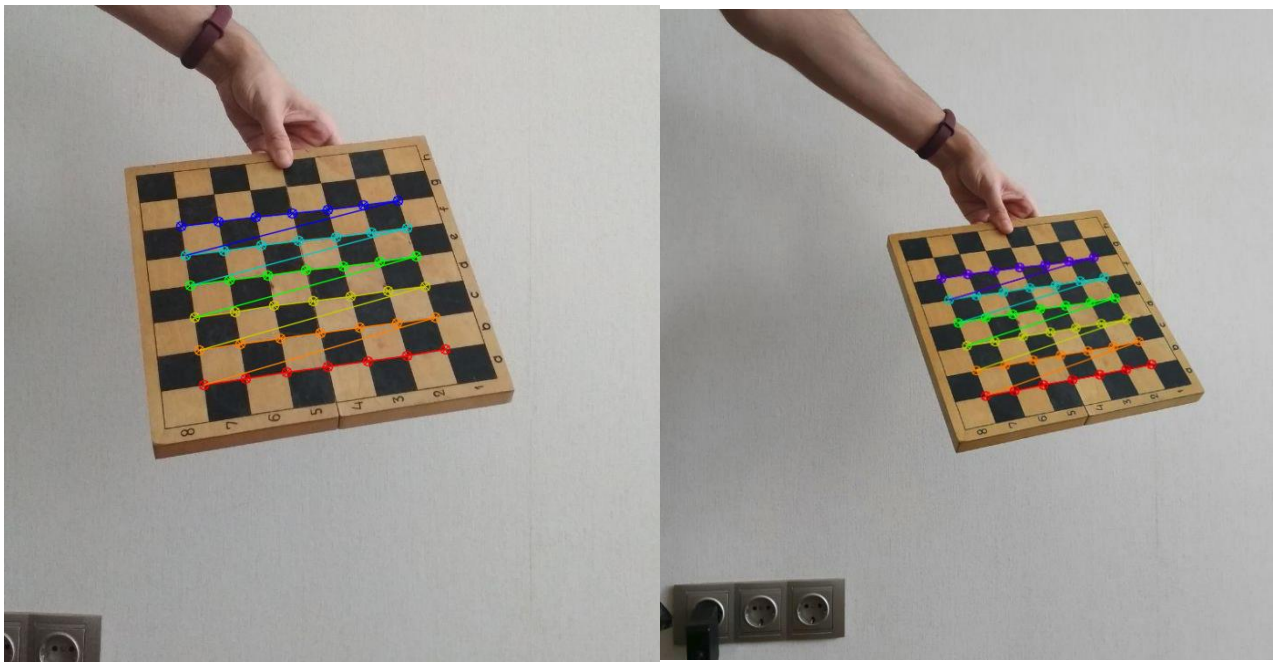

Рисунок 3.9 - Процес калібрації камер, як ідентифікація шахової дошки під різними кутами

3.3.1.1 Калібрація кожної камери окремо

Наша реалізація дозволяє покрити всі чотири випадки, роблячи аналіз n зображень для кожної камери, які ми помістили в у відповідні директорії. Потім для кожного з цих зображень ми перетворюємо зображення у відтінки сірого (Рисунок 3.10), і передаємо його в функцію бібліотеки CV2, яка здатна виявляє кути шахової дошки, та дає зрозуміти, чи існує шахова дошка з усіма її кутами, і повернути 2D координати площини зображення, де були виявлені ці кути. Важливий факт, що можна вказати в параметрах функції CV2 кількість клітинок дошки (можна вказати менше, але не більше, бо отримаємо помилку).

```
49          images_l = glob.glob('./calibration/left/*')
50          images_r = glob.glob('./calibration/right/*')
51
52          for i in range(len(images_l)):
53
54              path_l = images_l.pop()
55              path_r = images_r.pop()
56
57              im_l = cv2.imread(path_l)
58              im_r = cv2.imread(path_r)
59
60              gray_l = cv2.cvtColor(im_l, cv2.COLOR_BGR2GRAY)
61              gray_r = cv2.cvtColor(im_r, cv2.COLOR_BGR2GRAY)
62
```

Рисунок 3.10 - Діставання всіх картинок для кожної з камер за допомогою Glob, та переведення їх у відтінки сірого

Врешті-решт, ми закінчимо на n-наборах 2D-3D точкових пар. Вони передаються до іншої функції бібліотеки, яка обчислює матрицю камери, 5 коефіцієнтів спотворень та вектори обертання та перекладу. Ці параметри будуть використовуватися для викривлення зображень, коли вони будуть зняті для створення карти глибини. За експериментами дослідженими в Інтернеті вдалося

зрозуміти, що оптимальною кількістю зображень шахової дошки, для задовільних результатів, буде обмежено кількістю 15-20 штук для кожної камери.

Повернемось до функцій CV2, які обраховують 2D та 3D координати кутів на дошці:

- *'cv2.findChessboardCorners'*: повертає 2D координати кутів на зображенні, які було виявлено саме як кути на дошці.
- *'cv2.calibrateCamera'*: дана функція вимагає на вхід набір 3D точок, відповідні виявлені 2D точки та розмір зображення, а повертає матрицю камери, масив коефіцієнтів спотворень, а також матриці обертання та зсувів:

$$CameraMatrix = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \quad DistortionCoefficients = \begin{bmatrix} k1 & k2 & p1 & p2 \end{bmatrix}$$

Рисунок 3.11 - Результати обчислень функції *'cv2.calibrateCamera'*

fx, fy - горизонтальна та вертикальна фокусні відстані; *cx, cy* - координати фокусного центра; *k1, k2* - коефіцієнти радіального спотворення, а *p1* та *з2* - дотичні коефіцієнти спотворень.

3.3.1.2 Калібрація двох камер разом (стереоректифікація)

Тепер, коли ми підрахували всі параметри для кожної камери окремо, потрібно порахувати для обох камер разом. Для чого це потрібно? Після того як ми отримаємо матрицю обертання та вектор зсуву, ми маємо визначити положення обох камер одної відносно іншої. Це потрібно для того, щоб ми точно знали, що наші камери бачать один і той же предмет, і завдяки вектору зсуву чи матриці

обертання ми можемо знайти всі, необхідні нам пікселі на кожній з камер, які будуть відповідати один одному. Це забезпечує визначення точних границь та кутів об’єктів та сцени в цілому. Ми можемо отримати матрицю обертання R та вектор перекладу T між 1-ю та 2-ю системами координат камери:

$$R_2 = R \cdot R_1$$

$$T_2 = R \cdot T_1 + T$$

Де R_i і T_i - обчислені позиції шахової дошки щодо i-ої камери, тоді як R і T дозволяють обчислити положення однієї камери відносно іншої.

Ось так, завдяки уникненню викривлень ми зможемо горизонтально проводити лінії між відповідними групами пікселів та знаходити найбільші невідповідності для побудови Disparity Map.

Основними функціями OpenCV, які використовуються в цьому процесі, є:

- *’cv2.stereoCalibrate’*: дана функція вимагає на вхід матрицю камери та коефіцієнти спотворення для двох камер, а виводить вектори обертання і зсуву R і T, які необхідні для зв'язку систем координат двох камер.
- *’cv2.stereoRectify’*: використання R та T дозволяє обрахувати матриці обертання для двох камер, що дає можливість два зображення поставити в одну площину паралельно.

Ефект можна пояснити так, що фото можна обернути таким чином, що по епіполярних лініях можна відслідковувати як на лівому, так і на правому зображеннях.

3.3.2 Реконструкція

3.3.2.1 Попередня обробка зображень

Попередня обробка (preprocessing) необхідна для підготовки зображень до аналізу. В даному дослідженні присутні кілька етапів підготовки, таких як підвищення різкості, гаусівська піраміда та вирівнювання гістограми. Всі операції попередньої підготовки які ми застосували до вхідних даних мають на меті збільшити шанс алгоритму відповідності для кращого відмінності між тими зонами на які не співпадають на картинках. Це допоможе легко розпізнавати такі зони.

Перший фільтр який ми застосовуємо це підвищення різкості Лапласівським фільтром, Який обробляє картинки і повертає їх в кращій якості до деталей, підкреслюючи місця з інтенсивними змінами плану. [2]

$$LaplacianKernel = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Результатом є зображення яке показує більшу кількість інформації в сусідніх зонах та дозволяє визначити найближчі зони, які не співпадають.

Також було використано прийом, для підвищення точності алгоритму, вирівнювання гістограми. Цей прийом зазвичай застосовується для підвищення контрастності зображення. Існує проблема такого прийому коли в одній з картинок є кольорова зона якої немає на іншому зображення. Це призводить до Великих перепадів на гістограмі. Щоб боротися з цією проблемою було вирішено застосувати цей прийом тільки якщо в нас немає суттєвих відмінностей на обох зображеннях. Тобто якщо одне зображення не охоплює певну зону іншого і на ньому є яскраві кольоровий об'єкт ми просто пропускаємо прийом вирівнювання

гістограми і рухаємося далі за алгоритмом. Ще однією суттєвою проблемою є ситуація коли на двох камерах ми спостерігаємо різні значення яскравості, Тому було прийнято рішення проводити експеримент в рівномірно-освітленому приміщенні, та з камерами смартфонів, що майже не відрізняються.

Одне з головних питань ефективності алгоритмів відповідності, навіть при простому горизонтальному пошуку, було викликано кількістю пікселів які повинен сканувати алгоритм, шукаючи відповідність. Причина, чому постало це питання це полягає в швидкодії алгоритму, адже для нас важлива швидкість для ефективної обробки зображень, яка могла б легко проводитись при стрімінгу відео.

Щоб зменшити навантаження алгоритму було вирішено використовувати піраміди Гаусса, зокрема довелось проаналізувати всі вхідні зображення перед подачею в алгоритм, зменшивши в 4 рази загальну кількість пікселів.

У тому числі був зменшений розмір діапазону пікселів для горизонтальної перевірки, а також було зменшено вдвічі розміри найближчих зон для порівняння (мається на увазі сусідні полігони пікселів). Хоча методика знижує точність результатів оскільки зменшення даних параметрів передбачає згладжування зображення через фільтр Гаусса (Gaussian Filter), з іншого боку вдалося зберегти оригінальні розміри мікрополігону. Також внаслідок згладжування ми отримуємо додатковий бонус, завдяки якому ми позбуваємося пікселів які додають шумів до зображення.

3.3.2.2 Пост-обробка зображень. Заповнення пустих проміжків та видалення залишків

Трапляються ситуації коли згенерований 3D моделі ми бачимо пусті та незаповнені проміжки, що свідчить про випадкові ситуації коли деяким пікселями було присвоєно нульове значення нерівномірності. це трапляється з різних причин: об'єкт занадто рівномірний, або, тому що процес вирівнювання не був виконаний вичерпно. Для пом'якшення наслідків цих неправильних значень застосовуємо процедури заповнення отворів - так для кожного пікселя який має нульове значення ми перевіряємо всі сусідні пікселі у зоні ні NxN. Якщо в цій зоні сусідні пікселі теж мають нульове значення, то це означає, що, імовірно, ця область насправді була обрахована інакше, і ми залишаємо даний Піксель з його початковим нульовим значенням. Якщо ж сусідні пікселі мають ненульові значення невідповідностей, тоді ми обираємо сусіда з найвищими значенням і використовуємо його, як нове значення невідповідності центрального пікселя.

Майже завжди алгоритм побудови відповідностей не в змозі знайти правильну відповідність для певних пікселів. У більшості випадків це спричинено наявністю монохроматичних предметів і поверхонь або яскравих джерел світла, які впроваджують наявність різних мікрорайонів, дуже схожих один на одного. Іншою причиною невідповідності є наявність об'єкта занадто близько до камери, що вимагає розміру горизонтального діапазону вище, ніж зазвичай, щоб знайти сусідні зони відмінностей.

Проаналізувавши різні ситуації, ми виявляємо залишки пікселів які мають дуже високі або дуже низькі значення невідповідності (disparity value). В результаті ці пікселі або групи пікселів неправильно представляють співпадіння. Тому перед етапом обчислення тривимірних координат ми можемо видаляти ці залишки пікселів. Це очевидно обмежує визначено нашою системою мінімальну та максимальну глибину, але значно покращує результати.

3.3.2.3 Карта глибини та 3D координати

Успішно виконана ідея просто була спрямована на створення прямої зв'язку між значеннями нерівності та глибинними заходами.

Він полягав у створенні карти невідповідності деяких сцен, вимірюванні ефективної відстані від об'єкта до камери, і прив'язуванні цих вимірювань до відповідних значень невідповідності. Після того, як ми отримали достатню пар значень, проходила інтерполяція їх з різними типами кривих, намагаючись отримати відповідність нерівномірності та глибини (disparity-depth association).[2]

Кінцеві результати були отримані, прийнявши кубічну інтерполяційну криву такого типу:

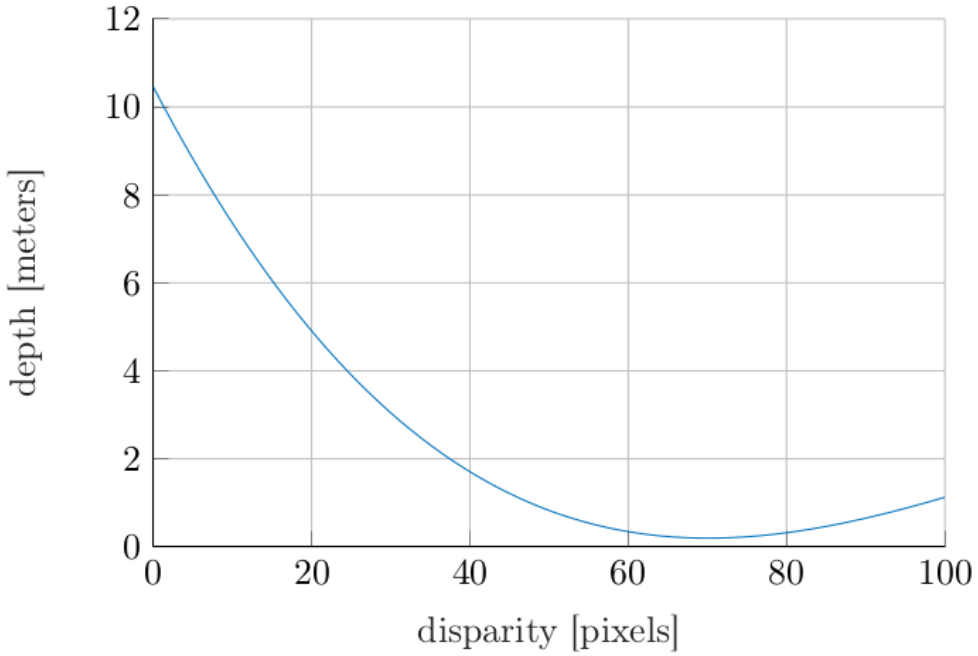

Рисунок 3.12 - Кубічна інтерполяційна крива для disparity-depth association [2]

Після отримання реальних значень глибини, нам також потрібно обчислити для кожного пікселя дві інші координати вручну.

Ідея полягає в тому, щоб уникати припущень координат фокусної точки та площини зображення камери, так як вимірювання є складними, або ж методу спроб та помилок [2]. Щоб мати можливість проектувати в просторі лінію для кожного пікселя, не знаючи цих двох об'єктів інформації, ми вирішили безпосередньо виміряти розміри повної прямокутної сцени, яку камера здатна фіксувати на заздалегідь заданій відстані.[2]

На практиці ми зафіксували зображення білої стіни на відстанях 0,75 метрів і 1,5 метра з використанням шахової дошки. Тоді ми просто виміряли висоту і ширину прямокутників. Ці вимірювання, на відстані від стіни, використовувались для прив'язки до кожного пікселя обох прямокутників трьох просторових координат, змушуючи центральний піксель мати x та y координати, рівні 0. Звідси випливає, як обчислюються координати:

$$centralX = \frac{WidthMeters}{2} \qquad centralY = \frac{HeightMeters}{2}$$

$$\begin{bmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{bmatrix} = \begin{bmatrix} \frac{WidthMeters}{WidthPixels} \cdot j - centralX \\ \frac{HeightMeters}{HeightPixels} \cdot i - centralY \\ DepthMeters \end{bmatrix} \quad \forall i, j \text{ such that } i \in [0, HeightPixels), \quad j \in [0, WidthtPixels)$$

Рисунок 3.13 - Обчислення координат [2]

- *WidthMeters* та *HeighthMeters* є розмірами прямокутника виміряного в метрах.
- *WidthPixels* та *HeightPixels* є розмірами зображення в пікселях (роздільна здатність).
- *DepthMeters* є виміряною дистанцією від камери до прямокутника.

- *i* та *j* є індексами позиції пікселя на зображенні..

Після цього кроку ми маємо всю інформацію, необхідну для обчислення фактичних координат точок нашої 3D-моделі. Вони обчислюються спочатку, ставлячи у відповідність значення глибини, отриманої з попереднього кроку, координаті z.[2]

Потім ми вважаємо, що лінії пронизують пару точок, що належать до двох прямокутників і пов'язані з тим самим пікселем, і для кожної точки фіксуємо значення z з карти глибини, отримуючи решту координат x і y. Обчислення координат [2]:

$$k = \frac{DepthMap(i,j) - RectangleB_z(i,j)}{RectangleA_z(i,j) - RectangleB_z(i,j)}$$

$$\begin{bmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{bmatrix} = \begin{bmatrix} RectangleB_x(i,j) + k \cdot (RectangleA_x(i,j) - RectangleB_x(i,j)) \\ RectangleB_y(i,j) + k \cdot (RectangleA_y(i,j) - RectangleB_y(i,j)) \\ DepthMap(i,j) \end{bmatrix}$$

Де:

- *i* та *j* є індексами позиції пікселя на зображенні.[2]
- *DepthMap(i, j)* це реальна глибина в метрах, попередньо обрахована для пікселя *(i, j)*.[2]
- $RectangleA_x(i,j)$, $RectangleA_y(i,j)$ та $RectangleA_z(i,j)$ є відповідними координатами для пікселя *(i, j)*, що належить до одного з прямокутників представлених вище.[2]
- $RectangleB_x(i,j)$, $RectangleB_y(i,j)$ та $RectangleB_z(i,j)$такі ж координати, але для іншого прямокутника.[2]

Кожен піксель був розміщений в реальній тривимірні системі координат в метрах і представлений таким самим кольором оригінального пікселя.

3.3.2.4 Тривимірна модель

Внаслідок всіх обрахувань отримуватимемо тривимірні точки, які необхідно десь зберігати. Досить простий та зручний варіант було обрано для вирішення даної задачі - зберігати дані як *.ply* файли. Перш за все, до них буде нескладно додавати нову інформацію, а сам формат ply (Polygon File Format) складається з трьох частин [2]:

- Властивості: тут необхідно визначити тип даних, що описує кожну з властивостей точки (наприклад, колір та положення). Також вказується кількість вершин і граней моделі;
- Вершини: тут ми перераховано кожну точку в тривимірній системі координат разом із її значеннями властивостей. У нашому випадку вказано позиції X, Y, Z та колір RGB;
- Faces розділ: даний частина нами не використовувалась, тож вона у нас пуста.

Щоб представити нашу хмару точок, ми просто використовуємо позиції X, Y і Z, отримані в процесі визначення глибини карти, і завдяки бібліотеці "plyfile" ми перетворюємо їх у тип даних, який можна правильно вставити у файл *.ply*.

Також ми можемо забарвити ті точки, використовуючи той факт, що наша карта невідповідності обчислюється як зміщення одного з двох зображень (правильне зображення є домінуючим у нашому випадку), і як таке ми просто використовуємо значення RGB цього зображення, застосовуючи їх до значень глибини, обчислених для кожного пікселя.

Для перегляду дуже зручним для використання є MeshLab, звідки і робились скріншоти для відображення результатів, які можна побачити на Рисунку 3.15.

Рисунок 3.14 - Карта нерівномірностей з фото

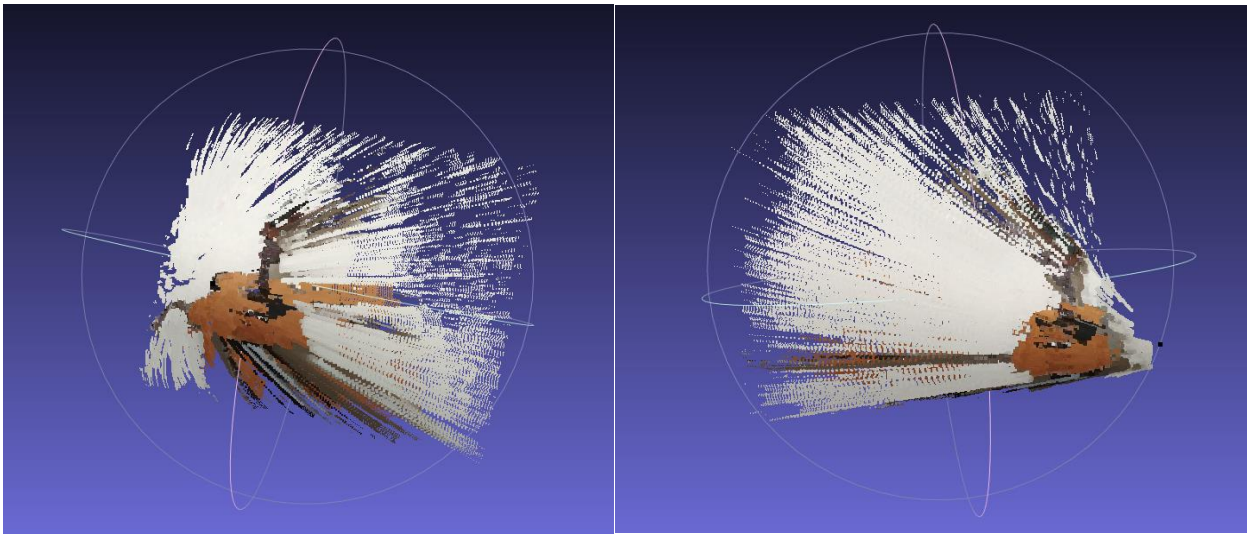

Рисунок 3.15 - Побудована 3D модель за попередніми кроками

Як ми бачимо, побудовані моделі є далеко недосконалими, бо ми можемо бачити об’єкт лише з кількох ракурсів, з інших він буде втрачати свою форму. Ми можемо уявити, що за допомогою відеопотоку ми могли б отримати набагато більше деталей і оглянути об’єкт з усіх можливих ракурсів. Звісно щоб побудувати таку модель нам необхідно було б ще попрацювати з зображеннями попарно, щоб отримувати рівномірну і пов’язану інформацію. Це ми розглянемо в наступному підрозділі.

3.4 Ідея простого алгоритму 3D реконструкції сцени з відео

Оскільки алгоритм створення 3D моделі полягає в аналізі двох зображень з двох камер, які були проаналізовані нашим алгоритму попередньо, то в такому випадку нам потрібно відштовхуватися від цього принципу, і для аналізу відеопотоку з двох камер, аналізувати зображення попарно при цьому враховуючи попередні 3D результати.

Припустимо, що в нас є два відеопотоки з камер Cam1 та Cam2.

Cam1.	Cam2
pic11	pic21
pic12.	Pic22
pic13.	pic23

Окрім попарного аналізу по дві картинки з двох камер нам потрібно робити перехресні аналізи картинок як однієї камери так і з двох. Робити реконструкцію і знаходити тривимірні точки таким алгоритмом:

3dObj_1 = _3dImageFrom(pic11, pic21)

3dObj_2 = _3dImageFrom(pic11, pic12, 3dObj_1)

3dObj_3 = _3dImageFrom(pic12, pic22, 3dObj_2)

3dObj_4 = _3dImageFrom(pic22, pic23, 3dObj_3)

3dObj_5 = _3dImageFrom(pic22, pic23, 3dObj_4)

Також слід врахувати, що якщо у нас рухаються камери, а не об'єкт, тобто об'єкт стоїть на місці а камери повертаються навколо нього, то в такому випадку в певний момент часу вони перестануть отримувати спільні точки, оскільки будуть знаходитися по різні сторони від об'єкта. Для цього необхідно задати додаткові умови, в яких ми будемо явно вказувати, що камера не може віддалятися одна від одної настільки, щоб алгоритм перестав знаходити співпадіння точок, або ж після закінчення співпадінь, ми перестанемо робити перехресний аналіз зображень і будемо робити аналіз лише зображень відповідної камери.

РОЗДІЛ 4. АНАЛІЗ ПРАКТИЧНОЇ ЧАСТИНИ ТА ПРОСТІР ДЛЯ ВДОСКОНАЛЕННЯ

Якщо ми поглянемо на описані процеси реалізації, то чітко виділимо три частини, які реалізовувались окремо:

- Реалізація трансляції кількох відео з камер в режимі реального часу.
- Реалізація алгоритм побудови 3D моделі з двох зображень.
- Опис алгоритму для аналізу безперервного потоку зображень, точніше відеопотоку.

Як ми бачимо перших два пункти досить успішно реалізовано, а алгоритм в третьому пункті не є складним для реалізації, але як згадувалось на початку цього розділу, в процесі даної роботи були доступні не надто потужні технічні пристрої

(камери телефонів серії Redmi та комп’ютер без потужної відеокарти для паралельних обчислень), тому зараз слід оглянути результати продуктивності дії нашого алгоритму побудови 3D реконструкції сцени (Таблиця 1) на прикладі аналізу трьох різних сцен.

1.	Disparity Map	Depth Map	Conversion to PLY
2.	2.311188 s	7.420445 s	33.9950 s
3.	3.02401 s	5.98032 s	25.3691 s
4.	2.21452 s	6.71342 s	30.9215 s

Таблиця 1 - Часові вимірювання роботи алгоритму

Найдовше виконується конвертація в ply файл, але даної операції можна виконувати не для кожної пари зображень, А наприклад, через певний проміжок часу, а може після завершення відеопотоків, але насправді це не є найкритичнішим моментом, адже обчислення карти невідповідностей та карти глибини займає в сумі середньому 10 секунд, а нам для досить швидкої побудови 3D моделі необхідно було б аналізувати пару зображень хоча б п'ять разів на секунду. П'ять разів на секунду це досить об'єктивне значення і його можна обгрунтувати частотою кадрів відеопотоку. Так на кіностудіях знімають фільми з частотою 24 кадри на секунду, але для нормального відтворення відео достатньо й половини, тобто 12-15 кадрів на секунду. Для нашої задачі не є необхідним брати для аналізу кожен кадр, адже просто потрібні зображення з невеликим відхиленням від попереднього розташування камери.

Можна зробити припущення, якби ми скористалися розпаралеленим обчисленням з відеокартою NVIDIA та можливістю застосування середовища CUDA, як це планувалося з самого початку то такому випадку ми могли б претендувати на повноцінно розроблене застосування.

При наближенні до закінчення написання даної роботи, було звернуто увагу на певні наукові роботи в яких описувалися можливості ідентифікації об'єктів з відео потоку за допомогою відстежування точок, що в теорії може значно зменшити обсяги обчислень та, відповідно, зробити можливим аналіз відео навіть на таких пристроях як було використано для даній роботі. але це потребує додаткового дослідження та порівняння з розглянутими методами аналізу. Так було натраплено на алгоритм, який базувався на відстежуванні точок за допомогою гео-камер, і за необхідних модифікацій цей підхід можна було б адаптувати згідно наших технічних особливостей. Цей алгоритм описано на Рисунку 4.1, який було знайдено в дослідженні “Detailed Real-Time Urban 3D Reconstruction from Video” та опубліковано в International Journal of Computer Vision. Саме через знайдений такий підхід було додано огляд алгоритмів відстежування точок та об’єктів.

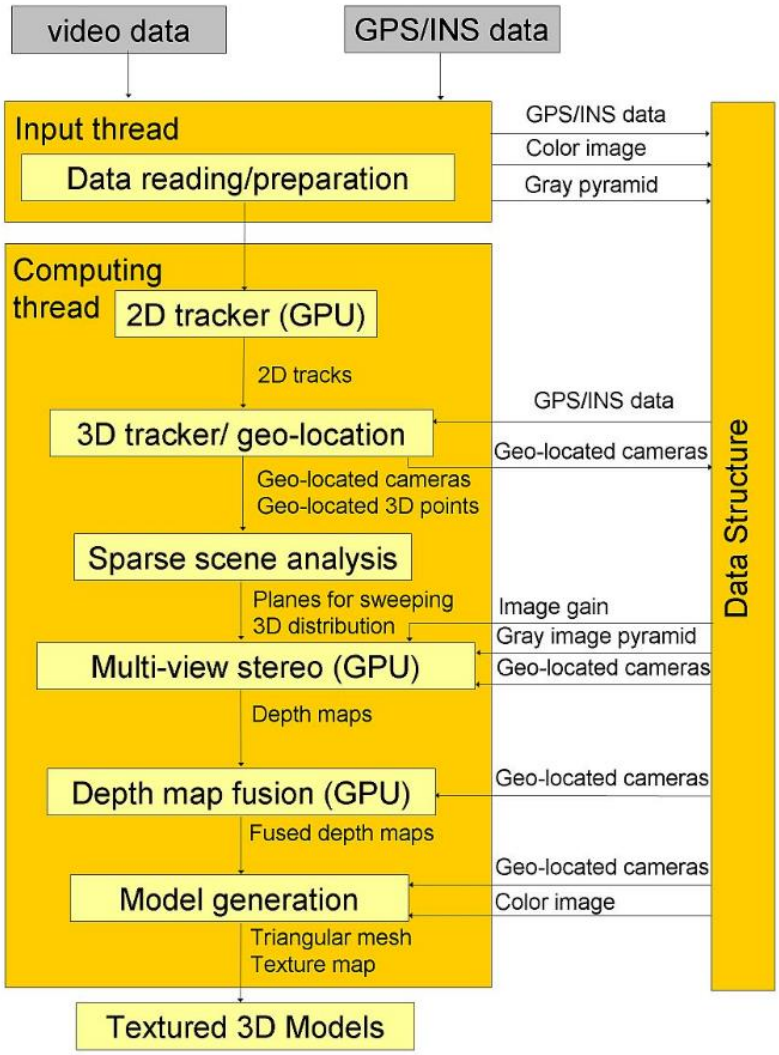

Рисунок 4.1 - Реконструкція 3D моделі в урбаністичному середовищі за допомогою гео-камер

ВИСНОВКИ

В даній роботі було розглянуто можливість реалізації задачі 3D реконструкції сцени за відео з декількох камер, тобто аналіз, в нашому випадку, двох відеопотоків, що спрямовані на одну сцену, та, як результат, отримання тривимірної моделі даної сцени. Також зробивши декомпозицію поставленої задачі, було описано кожне підзавдання, яке впливає на ефективність реалізації, та яке може бути замінене альтернативним алгоритмом.

Були розглянуті відомі алгоритми виявлення та опису особливостей, відслідковування точок та оцінки позиції об’єкта. Також була описана теоретична база, яку беззаперечно вимагали наші підходи побудови тривимірної моделі, такі як калібрація камер, епіполярна геометрія, триангуляція та вирівнювання, а також згадувалось про побудову карт невідповідностей та глибини (Disparity та Depth Maps).

В практичній частині ми відштовхувались від попарного аналізу зображень з двох камер, таким чином ідея полягала в поступовому накопиченні точок в тривимірній системі координат (завдяки запропонованому алгоритму почергового співставлення уже обчислених тривимірних точок та нових обрахованих з наступної пари зображень) зі збереженням властивостей пікселів з двовимірних зображень, а саме кольору та обрахованої глибини.

В результаті не вдалося реалізувати аналіз відеопотоку та створення 3D моделі через великі часові затрати на обчислення полігону точок в тривимірному просторі (часові затримки було наведено в Розділі 4), та, відповідно, неможливість безперешкодної швидкої обробки зображень з відеопотоку. На це повпливало кілька факторів: відсутність паралельних обчислень за допомогою середовища CUDA, які планувались на початку, також, вважаю, значно вплинуло на ефективність власноручна реалізації алгоритмів обчислень карт глибини та

невідповідностей. Тобто результати роботи не можна назвати безуспішними, адже при невеликій зміні певних підходів можна досягти реалізації тривимірної реконструкції. Це підтверджує те, що ми успішно реалізували декомпозовані задачі окермо одна від одної, але наштовхнулись на перешкоду завеликого часу обчислень.

Щодо подальших досліджень, то для початку варто спробувати використати певні алгоритми, які вже реалізовані в бібліотеці OpenCV, бо в даній роботі вони реалізовувались вручну з навчальною метою. Також протестувати швидкість обробки зображень та побудови тривимірної моделі за допомогою паралельних обчислень в середовищі CUDA, як було заплановано на початку. Ще слід дослідити альтернативні алгоритми за допомогою відстежування точок та об’єктів під час аналізу відео в режимі реального часу, що дозволить нам уникнути постійної перебудови об’єктів, які вже аналізувались.

СПИСОК ЛІТЕРАТУРИ

1. Mordvintsev A. OpenCV-Python Tutorials Documentation (Release 1)/ Mordvintsev A., Abid K. – Nov 05, 2017.
2. 3D model reconstruction from stereo 2D images.
3. Chris McCormick, Stereo Vision/ C. McCormick. – 2014 – режим доступу: http://mccormickml.com/ .
4. Финогеев А.Г., Методика распознавания изображений на основе рандомных деревьев в системах автоматизированного проектирования расширенной реальности // Современные проблемы науки и образования/ Финогеев А.Г., Финогеев А.Г., Четвергова М.В. – 2012. – № 5.
5. D.A. Gavrilov, Streaming Hardware Based Implementation of SURF Algorithm, D.A. Gavrilov 1 , A.V. Pavlov 2 // Proceedings of Universities. Electronics 2018 23(5).
6. Ethan R. ORB: An efficient alternative to SIFT or SURF/ E. Rublee, V. Rabaud, K. Konolige, G. R. Bradski.
7. А. Куракин, Основы стереозрения. – 13 октября 2011 – режим доступу: https://habr.com .
8. Kenji Hata, “CS231A Course Notes 3: Epipolar Geometry” (18c.)/ K. Hata, S. Savarese. – 2018 – режим доступу: http://web.stanford.edu/class/cs231a/course_notes/03-epipolar-geometry.pdf.
9. Dr. George Bebis, Epipolar (Stereo) Geometry Notes // CS491E/791E: Computer Vision. – 2004.
10. Lowe, David G. (1999). "Object recognition from local scale-invariant features". Proceedings of the International Conference on Computer Vision. 2. pp. 1150 – 1157. – режим доступу: https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf.

11. Lowe, David G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". International Journal of Computer Vision. 60 (2): 91–110. – режим доступу: http://citeseer.ist.psu.edu/lowe04distinctive.html.
12. Kenji Hata “CS231A Course Notes 4: Stereo Systems and Structure from Motion” (18c.)/ K. Hata, S. Savarese. – 2018.
13. Machine Learning with Python by Tutorials Point Simple Learning.
14. Satya Mallick, Object Tracking using OpenCV (C++/Python) / S. Mallick. – February 13, 2017. – режим доступу: https://www.learnopencv.com.
15. Real Time pose estimation of a textured object // OpenCV Documentation, version 3.2, 1999-2017.
16. Mude Lin, Recurrent 3D Pose Sequence Machines/ L. Lin, X. Liang, K. Wang, H. Cheng. – 2017.
17. D. Shin, Fowlkes 3D Scene Reconstruction with Multi-layer Depth and Epipolar Transformers/ D. Shin, Zh. Ren, E. B. Sudderth, C. Charless. – 27 Aug 2019.
18. Sh. Song, Semantic Scene Completion from a Single Depth Image/ Sh. Song, F.Yu, A. Zeng, A. X. Chang, M. Savva, T. Funkhouser. – Nov 2016.