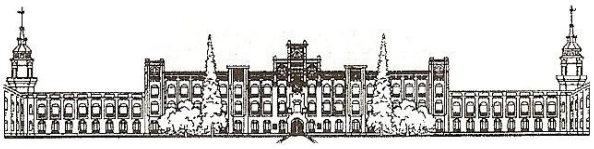


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Лабораторна робота №4

з дисципліни «Комп'ютерний зір»

«Калібрування камери»

Виконали студентки групи: КВ-11

ПІБ: Михайліченко Софія Віталіївна
Шевчук Ярослава Олегівна

Перевірила: _____

Київ 2024

Постановка задачі:

Реалізувати наведені нижче завдання у вигляді відповідного застосунку/застосунків, що передбачають завантаження зображень із файлів, їх обробку відповідним чином та візуалізацію і збереження файлів результату (рекомендовані мови програмування: C++ або Python, але за бажання припустимі і інші).

Для роботи із зображеннями рекомендовано використовувати бібліотеку OpenCV (зокрема, cv::Mat, cv::imread, cv::imwrite, cv::imshow, cv::waitKey), хоча, за бажання припустимо використовувати і інші бібліотеки, що дозволяють роботу з зображеннями (завантаження, збереження, візуалізація, доступ до значень пікселів). Завдання мають бути виконані самостійно, не використовуючи існуючі реалізації відповідних алгоритмів (в т.ч. реалізацію операції згортки) з бібліотеки OpenCV або інших бібліотек.

Бажано забезпечити автоматичне налаштування проекту із його залежностями (наприклад, через відповідний сценарій CMake, або відповідно сформований файл requirements.txt системи керування пакунками pip).

1. Реалізувати проектування точок (заданих у світовій системі координат), відповідно до моделі камери-стенопа для заданих параметрів камери (внутрішніх та зовнішніх). Дисторсія об’єктива може не розглядатися.

2. Виконати симуляцію отримання даних для калібрування камери (див. допоміжну інформацію). Навести зображення проекцій точок.

3. Виконати калібрування камери за допомогою функції OpenCV calibrateCamera (див. допоміжну інформацію). Порівняти параметри камери, отримані в результаті калібрування із початково заданими параметрами камери.

4. Додати ще одну камеру зі своїми параметри та сформувати дані для моно та стерео калібрування (див. допоміжну інформацію). Обрати складний випадок, щоб система камер не відповідали канонічному випадку (тобто, камери мають

дещо різні внутрішні параметри, напрямлені дещо в різних напрямках та мають зміщення за всіма осями). Навести зображення проекцій точок.

5. Виконати калібрування 2-ї камери за допомогою функції OpenCV calibrateCamera (див. допоміжну інформацію). Порівняти параметри камери, отримані в результаті калібрування із початково заданими параметрами камери.

6. Виконати калібрування пари камер за допомогою функції OpenCV stereoCalibrate (див. допоміжну інформацію). Порівняти параметри камер (зовнішні), отримані в результаті стерео калібрування із початково заданими параметрами.

7. Розрахувати фундаментальну та істотну матриці та порівняти їх із отриманими в результаті стерео калібрування функцією stereoCalibrate.

8. Спроектувати декілька довільних тривимірних точок (за допомогою функцій з п.1.) на обидві камери та впевнитися, що виконується умова $p_R^T \cdot F \cdot p_L = 0$, де p_L та p_R – проекції точки на ліву та праву камеру, а F – фундаментальна матриця, отримана в п. 7.

9. Розрахувати матриці гомографії для виконання ректифікації пар стереозображень (для обраних камер). Для цього знайти спільний напрямок візирування, а також обрати деякі спільні внутрішні параметри камери (значення матриці внутрішніх параметрів камери). Можна використати параметри однієї з камер, або ж обрати деякі середні значення (для зручності можна взяти, наприклад, $f_x = f_y = \frac{f_{xL}+f_{xR}+f_{yL}+f_{yR}}{4}, c_x = \frac{W}{2}, c_y = \frac{H}{2}$, $W \times H$ – розміри зображення).

10. Спроектувати декілька довільних тривимірних точок на зображення обидвох камер, а далі виконати ректифікацію цих зображень за допомогою матриць гомографії, отриманих в п. 9. Отримані зображення (оригінальні та ректифіковані) навести в звіті (для цього зручно зобразити точки різними

кольорами). Показати, що на ректифікованих зображеннях відповідні точки знаходяться в одному рядку (а на не ректифікованих – ні).

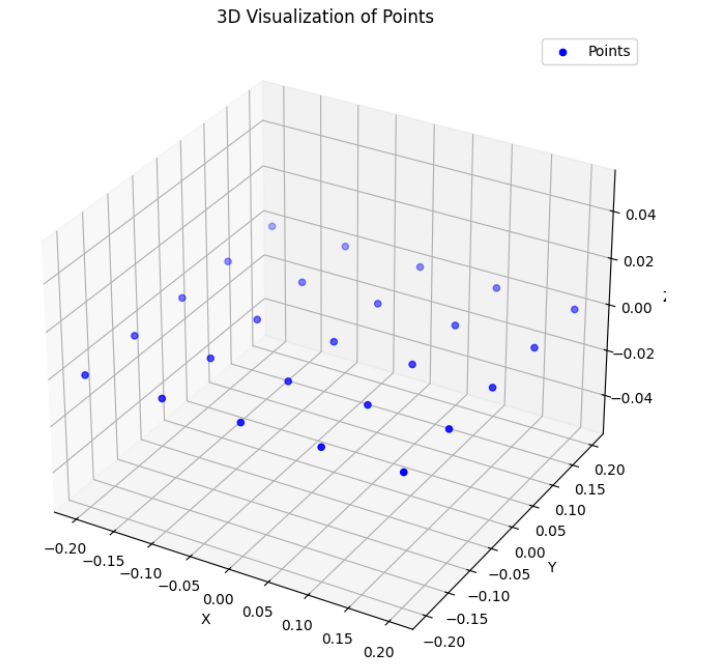
Додаткові завдання (за бажанням):

1. Де це можливо, виконати порівняння результатів роботи функцій, реалізованих самостійно, із результатами роботи відповідних функцій бібліотеки OpenCV (зокрема, cv::equalizeHist, cv::blur, cv::GaussianBlur, cv::medianBlur, cv::Sobel, cv::Sharr, cv:: Laplacian, cv:Canny, cv::cornerHarris, cv::FastFeatureDetector).

Порядок виконання роботи

Завдання 1

Реалізувати проектування точок (заданих у світовій системі координат), відповідно до моделі камери-стенопа для заданих параметрів камери (внутрішніх та зовнішніх). Дисторсія об’єктива може не розглядатися. Результати виконання:




```
Comparison of Rodrigues methods: 0.0
```

Дане завдання передбачає перетворення заданих у світовій системі координат 3D точок у 2D точки зображення, використовуючи модель камери-стенопа. У рамках наших результатів створюється просторовий об'єкт — набір точок, що розташовані у світовій системі координат. Потім модель камери, що включає параметри внутрішньої (інтринсикс) та зовнішньої (екстрінсикс) калібровки, використовується для проектування цих точок на площину зображення.

Спочатку задаються внутрішні параметри камери, включаючи фокусну відстань та координати головної точки (принципової точки). Ці параметри формують матрицю внутрішніх параметрів камери, що застосовується для трансформації координат. Відносно зовнішніх параметрів задається вектор

положення камери (вектор трансляції) та її орієнтація (вектор повороту, що представлений у форматі Родрігеса). Для обчислень здійснюється перетворення цього вектора повороту у матрицю обертання. Використовуючи ці параметри, кожна точка у світовій системі координат спочатку трансформується у систему координат камери. Потім, знаючи координати у системі камери, вони проектуються на площину зображення з використанням перспективного поділу (розподіл координат X та Y за Z). Далі ці координати модифікуються за допомогою матриці внутрішніх параметрів для отримання піксельних координат зображення.

Додатково виконується перевірка на валідність точок (фільтруються ті, що знаходяться за площиною камери). Уся методика може бути реалізована за допомогою вбудованих функцій OpenCV, таких як cv2.projectPoints, а також за допомогою власної реалізації з матричними операціями.

Отриманий результат відображає розташування проектованих точок на площині зображення у вигляді 2D координат, готових для візуалізації або подальшого аналізу.

Порівняння з функціями OpenCV:

У розв’язанні реалізовано два підходи до проектування точок зі світової системи координат на площину зображення за допомогою моделі камери-стенопа: використання вбудованої функції cv2.projectPoints із бібліотеки OpenCV і власну реалізацію алгоритму "з нуля". Обидва методи базуються на одних і тих самих математичних принципах, проте їхня реалізація та гнучкість мають певні відмінності.

Використання cv2.projectPoints є простішим і зручнішим з огляду на компактність коду. Ця функція приймає 3D точки, параметри камери та повертає 2D координати точок на зображенні разом із додатковою інформацією. Такий підхід має значну перевагу у випадках, коли точність розрахунків гарантована завдяки добре протестованій бібліотечній реалізації.

Однак її недоліком є менша прозорість алгоритму та обмежена можливість внесення модифікацій для спеціалізованих задач.

Реалізація власного методу є більш деталізованою та демонструє повний процес обчислень, що стоїть за бібліотечними функціями. Починаючи з перетворення вектора обертання у матрицю обертання, обчислюється трансформація 3D координат світової системи в камеру з урахуванням параметрів трансляції. Далі виконується перспективний поділ для проєкції точок, після чого застосовується матриця внутрішніх параметрів для перетворення координат камери у координати зображення. Власний підхід також враховує випадки, коли точки поза полем зору камери, і заповнює такі координати спеціальними значеннями, що полегшує аналіз результатів.

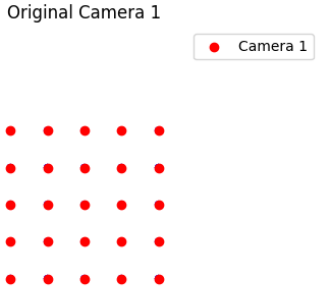
Основною різницею між цими підходами є ступінь контролю над розрахунками: вбудований метод є "чорною скринькою", яка швидко виконує завдання, у той час як власний метод дозволяє детально зрозуміти та контролювати кожен етап трансформації, а також адаптувати алгоритм до специфічних потреб.

Завдання 2

Виконати симуляцію отримання даних для калібрування камери (див. допоміжну інформацію). Навести зображення проекцій точок.

Результати виконання:

```
Comparison of project_points methods: 0.0
```



Симуляція отримання даних для калібрування камери передбачає генерацію 3D-точок і їхню проекцію на площину камери за допомогою двох методів. Спершу 3D-точки проектуються на площину камери за допомогою функції project_points, що використовує матрицю камери, вектори обертання та трансляції для перетворень. Потім ці ж точки проектуються другим методом — функцією project_points_from_scratch, яка моделює базову математику проекції вручну, без використання готових бібліотечних функцій.

Результати обох методів порівнюються за середньоквадратичною похибкою, яка свідчить про точність реалізації обох алгоритмів.

Для візуалізації результатів, як бачимо створюється чисте біле зображення, на якому зображуються спроєктовані точки. Це досягається нанесенням червоних маркерів у відповідних координатах, отриманих після обчислень. Результуюче зображення відображається у вигляді графіка, де точкам надаються кольори та координати для зручної ідентифікації.

Завдання 3

Виконати калібрування камери за допомогою функції OpenCV calibrateCamera (див. допоміжну інформацію). Порівняти параметри камери, отримані в результаті калібрування із початково заданими параметрами камери.

Результати виконання:

```
Camera 1 Parameters:
Initial Camera Matrix:
[[500.000 0.000 320.000]
 [0.000 500.000 240.000]
 [0.000 0.000 1.000]]
Calibrated Camera Matrix:
[[497.669 0.000 319.500]
 [0.000 498.450 239.500]
 [0.000 0.000 1.000]]
```

```
Camera 1 Parameters:
Initial Camera Matrix:
[[500.000 0.000 320.000]
 [0.000 500.000 240.000]
 [0.000 0.000 1.000]]
Calibrated Camera Matrix:
[[5138.287 0.000 449.301]
 [0.000 5152.899 246.744]
 [0.000 0.000 1.000]]
```

Калібрування камери виконується для визначення її внутрішніх параметрів, таких як фокусна відстань, центр проекції, коефіцієнти дисторсії, які коригують спотворення, спричинені оптикою камери.

Завдання починається з моделювання набору об’єктних і проєкційних точок, які відповідають теоретично відомим положенням об’єктів у просторі (3D-точки) та їхньому проєктуванню на двовимірну площину зображення (2D-точки). Для цього створюються однорідні набори об'єктних точок (наприклад, сітка координат у просторі), які повторюються кілька разів (по одному для кожного калібрувального кадру).

Щоб симулювати реальну роботу камери, на точках застосовуються випадкові невеликі обертання і зміщення в тривимірному просторі (за допомогою рандомізованих векторів повороту й зміщення). Потім об’єктні точки перетворюються на зображувані 2D-точки, використовуючи попередньо задану матрицю камери, до яких додаються невеликі випадкові шуми, які імітують помилки вимірювань.

Далі використовують функцію cv2.calibrateCamera із OpenCV, щоб визначити внутрішні параметри камери, включно з її каліброваною матрицею. Функція обчислює результати за допомогою співвідношення між 3D-точками у просторі та 2D-точками на зображенні.

Після калібрування результуючі параметри камери (калібрована матриця та коефіцієнти дисторсії) порівнюються з початково заданими. Порівняння демонструє, наскільки ефективно процес калібрування відновлює параметри камери, незважаючи на шуми та зміщення.

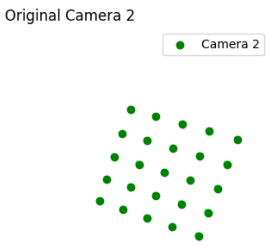
Як бачимо з результатів при заданих параметрам різним методом отримуємо різні значення калібрування, при чому в одному з цих методів результати кращі. Різниця в якості калібрування між двома підходами зумовлена відмінностями у способі створення набору 2D-проєкційних точок. У першому випадку об’єктні точки трансформуються до нового положення за допомогою

невеликих випадкових обертань (rvec) і зміщень (tvec). Таким чином, кожен кадр імітує різну перспективу або орієнтацію камери відносно об’єкта в просторі. Це створює більш реалістичний набір даних, адже у реальному калібруванні камеру переміщують у просторі, щоб зібрати точки під різними кутами та на різних відстанях. Такі дані забезпечують багатшу геометричну інформацію для калібрування, що дозволяє отримати параметри камери з високою точністю. У другому підході 2D-проєкційні точки створюються додаванням шумів до вже прорахованого набору точок із зафіксованого положення (projected_points1). Оскільки відсутні варіації у перспективах або орієнтації, система отримує менше інформації про поведінку камери у просторі, обмежуючи здатність моделі точно відновлювати параметри. Це особливо важливо для визначення таких параметрів, як центр проекції або дисторсія, які сильно залежать від різноманітності даних. Отже, кращі результати в першому випадку досягаються завдяки тому, що дані калібрування більш повно відображають різні аспекти проєкції в тривимірному просторі.

Завдання 4

Додати ще одну камеру зі своїми параметри та сформувати дані для моно та стерео калібрування (див. допоміжну інформацію). Обрати складний випадок, щоб система камер не відповідали канонічному випадку (тобто, камери мають дещо різні внутрішні параметри, напрямлені дещо в різних напрямках та мають зміщення за всіма осями). Навести зображення проекцій точок.

Результати виконання:



Як бачимо з результатів ми додали другу камеру, яка дещо відрізняється від першої. Вона має певні внутрішні та зовнішні параметри, які роблять конфігурацію системи складною. У другій камері внутрішні параметри мають відмінності у фокусній відстані та координатах головної точки. Зовнішні параметри також варіюються: камера має поворот навколо трьох осей та зміщення за координатами x, y і z. Це забезпечує нерівномірну систему, яка не відповідає канонічному випадку, коли камери ідеально узгоджені. У ході виконання, використовується тривимірний набір об'єктних точок, які проектуються на площину зображення другої камери за допомогою її параметрів. Це виконується для моделювання роботи камери у просторі. Результатом є набір точок у площині зображення, які візуалізуються як точки зеленого кольору на зображенні білої заливки. Ця візуалізація дозволяє проаналізувати розташування точок, спричинене налаштованими параметрами. Для імітації процесу калібрування формується набір тестових даних, які включають кілька реалізацій спроєктованих точок з невеликими випадковими коливаннями. Ці точки імітують реальні умови з урахуванням похибок знімання. За допомогою функції cv2.calibrateCamera виконується калібрування камери, в результаті чого отримуються її скориговані параметри.

Після калібрування порівнюються отримані параметри камери з початково заданими, що дозволяє оцінити точність та ефективність калібрування. Це допомагає зрозуміти, наскільки відхилення вплинуло на отримані внутрішні параметри, такі як матриця камери.

Завдання 5

Виконати калібрування 2-ї камери за допомогою функції OpenCV calibrateCamera (див. допоміжну інформацію). Порівняти параметри камери, отримані в результаті калібрування із початково заданими параметрами камери.

Результати виконання:

```
Camera 2 Parameters:
Initial Camera Matrix:
[[400.000 0.000 270.000]
 [0.000 400.000 190.000]
 [0.000 0.000 1.000]]
Calibrated Camera Matrix:
[[432.000 0.000 326.000]
 [0.000 420.000 209.000]
 [0.000 0.000 1.000]]
```

```
Camera 2 Parameters:
Initial Camera Matrix:
[[400.000 0.000 270.000]
 [0.000 400.000 190.000]
 [0.000 0.000 1.000]]
Calibrated Camera Matrix:
[[402.000 0.000 275.000]
 [0.000 404.000 189.000]
 [0.000 0.000 1.000]]
```

Дане завдання виконується аналогічно пункту 3, але уже для нової сформованої камери з іншими параметрами. Як бачимо для цієї камери різниця не є суттєвою у двох методах, так як тут більш точніше задаються параметри.

Завдання 6

Виконати калібрування пари камер за допомогою функції OpenCV stereoCalibrate (див. допоміжну інформацію). Порівняти параметри камер (зовнішні), отримані в результаті стерео калібрування із початково заданими параметрами.

Результат виконання:

```
Camera 1 Parameters:
Initial Camera Matrix:
[[500.000 0.000 320.000]
 [0.000 500.000 240.000]
 [0.000 0.000 1.000]]
Stereo calibrated Camera Matrix:
[[498.795 0.000 319.500]
 [0.000 499.113 239.500]
 [0.000 0.000 1.000]]

Camera 2 Parameters:
Initial Camera Matrix:
[[400.000 0.000 270.000]
 [0.000 400.000 190.000]
 [0.000 0.000 1.000]]
Stereo calibrated Camera Matrix:
[[409.032 0.000 319.500]
 [0.000 414.493 239.500]
 [0.000 0.000 1.000]]
```

Калібрування пари камер виконується для визначення відносного положення двох камер (зовнішніх параметрів) і уточнення їх внутрішніх параметрів, якщо

вони відомі лише з певною точністю. Це дозволяє виправляти спотворення зображення та поліпшувати точність 3D-реконструкції.

Процес стерео калібрування передбачає використання функції cv2.stereoCalibrate з OpenCV, яка потребує підготовки кількох ключових вхідних даних. Спочатку визначають об'єктні точки (координати фізичних точок на об'єкті в просторі) та відповідні їм точки на зображеннях, отриманих з двох камер (зображені точки). Також задаються попередньо визначені внутрішні параметри камер (матриця камери та коефіцієнти дисторсії).

Функція stereoCalibrate виконує оптимізацію, визначаючи параметри камер, ротаційну матрицю R і вектор зміщення T, які характеризують відносне положення однієї камери відносно іншої. В результаті обчислюються також матриця основи E та фундаментальна матриця F.

Після виконання стерео калібрування параметри камер порівнюють із початковими, що дозволяє оцінити, як саме змінились внутрішні та зовнішні параметри в результаті уточнення. Це порівняння виводиться у вигляді початкової та оновленої матриць камер для обох пристроїв. Зовнішні параметри (матриця ротації та вектор зміщення) порівнюють для розуміння точності стерео калібрування відносно попередньо заданих даних.

Завдання 7

Розрахувати фундаментальну та істотну матриці та порівняти їх із отриманими в результаті стерео калібрування функцією stereoCalibrate.

Результати виконання:

```
Essential matrix | stereoCalibrate:
[[0.508 0.395 -0.804]
 [-0.116 -0.102 0.195]
 [0.766 -0.720 0.128]]

Essential matrix | Custom:
[[0.508 0.395 -0.804]
 [-0.116 -0.102 0.195]
 [0.766 -0.720 0.128]]


Fundamental matrix | stereoCalibrate:
[[0.000 0.000 -0.004]
 [-0.000 -0.000 0.001]
 [0.001 -0.002 1.000]]

Fundamental matrix | Custom:
[[0.000 0.000 -0.004]
 [-0.000 -0.000 0.001]
 [0.001 -0.002 1.000]]
```

Для виконання даного завдання ми визначаємо істотну (Essential) та фундаментальну (Fundamental) матриці двома способами: за допомогою функції stereoCalibrate, яка виконує стерео калібрування вбудованими алгоритмами, та вручну, використовуючи задані матриці камер та матриці трансляції і обертання. Істотна матриця обчислена методом стерео калібрування (stereoCalibrate) збігається з обчисленою вручну матрицею (Custom). Це свідчить про правильність ручного розрахунку. Отримані матриці мають однакові елементи, що підтверджує їхню узгодженість та відсутність похибок. Фундаментальні матриці, отримані двома методами, також виявляються ідентичними. Збіг результатів демонструє узгодженість між використаними алгоритмами обчислень та підтверджує точність ручного розрахунку через матриці камер.

Завдання 8

Спроектувати декілька довільних тривимірних точок (за допомогою функцій з п.1.) на обидві камери та впевнитися, що виконується умова $p_R^T \cdot F \cdot p_L = 0$, де p_L та p_R – проекції точки на ліву та праву камеру, а F – фундаментальна матриця, отримана в п. 7.

Результат виконання:

```
Point 1: pR^T * F * pL = -0.088062
Point 2: pR^T * F * pL = -0.092742
Point 3: pR^T * F * pL = -0.097843
Point 4: pR^T * F * pL = -0.103363
Point 5: pR^T * F * pL = -0.109297
Point 6: pR^T * F * pL = -0.115642
Point 7: pR^T * F * pL = -0.122393
Point 8: pR^T * F * pL = -0.129546
Point 9: pR^T * F * pL = -0.137099
Point 10: pR^T * F * pL = -0.085770
Point 11: pR^T * F * pL = -0.090240
Point 12: pR^T * F * pL = -0.095135
Point 13: pR^T * F * pL = -0.100451
Point 14: pR^T * F * pL = -0.106185
Point 15: pR^T * F * pL = -0.112332
Point 16: pR^T * F * pL = -0.118889
Point 17: pR^T * F * pL = -0.125852
Point 18: pR^T * F * pL = -0.133216
Point 19: pR^T * F * pL = -0.082994
Point 20: pR^T * F * pL = -0.087253
Point 21: pR^T * F * pL = -0.091941
Point 22: pR^T * F * pL = -0.097054
Point 23: pR^T * F * pL = -0.102587
Point 24: pR^T * F * pL = -0.108537
Point 25: pR^T * F * pL = -0.114900
Point 26: pR^T * F * pL = -0.121672
Point 27: pR^T * F * pL = -0.128848
Point 28: pR^T * F * pL = -0.079727
Point 29: pR^T * F * pL = -0.083776
Point 30: pR^T * F * pL = -0.088257
Point 31: pR^T * F * pL = -0.093166
Point 32: pR^T * F * pL = -0.098498
Point 33: pR^T * F * pL = -0.104251
Point 34: pR^T * F * pL = -0.110419
Point 35: pR^T * F * pL = -0.116999
Point 48: pR^T * F * pL = -0.079387
Point 49: pR^T * F * pL = -0.083888
Point 50: pR^T * F * pL = -0.088818
Point 51: pR^T * F * pL = -0.094175
Point 52: pR^T * F * pL = -0.099954
Point 53: pR^T * F * pL = -0.106151
Point 54: pR^T * F * pL = -0.112762
Point 55: pR^T * F * pL = -0.066916
Point 56: pR^T * F * pL = -0.070331
Point 57: pR^T * F * pL = -0.074188
Point 58: pR^T * F * pL = -0.078484
Point 59: pR^T * F * pL = -0.083213
Point 60: pR^T * F * pL = -0.088372
Point 61: pR^T * F * pL = -0.093956
Point 62: pR^T * F * pL = -0.099961
Point 63: pR^T * F * pL = -0.106384
Point 64: pR^T * F * pL = -0.061618
Point 65: pR^T * F * pL = -0.064821
Point 66: pR^T * F * pL = -0.068470
Point 67: pR^T * F * pL = -0.072561
Point 68: pR^T * F * pL = -0.077089
Point 69: pR^T * F * pL = -0.082049
Point 70: pR^T * F * pL = -0.087439
Point 71: pR^T * F * pL = -0.093252
Point 72: pR^T * F * pL = -0.099486
Point 73: pR^T * F * pL = -0.055794
Point 74: pR^T * F * pL = -0.058785
Point 75: pR^T * F * pL = -0.062226
Point 76: pR^T * F * pL = -0.066112
Point 77: pR^T * F * pL = -0.070438
Point 78: pR^T * F * pL = -0.075200
Point 79: pR^T * F * pL = -0.080394
Point 80: pR^T * F * pL = -0.086016
Point 81: pR^T * F * pL = -0.092061
```

Для виконання даного завдання було спроектовано декілька тривимірних точок у світовій системі координат. Кожна точка визначалась у просторі з координатами x, y, z, де z було фіксовано як 1, а x і y змінювались у межах від −0.2 до 0.2 з рівномірним розподілом. Точки було спроектовано на дві камери за допомогою функції cv2.projectPoints. Для цього використовувались параметри камер, включаючи матриці внутрішніх параметрів, вектори обертання та трансляції. Отримані координати проекцій були перетворені у форму однорідних координат, додаючи 1 як останній компонент для кожної точки. Це забезпечило можливість коректного множення з фундаментальною

матрицею F, яка була визначена на попередньому етапі. Для перевірки виконання епіполярної умови — це проекції точки на ліву та праву камери відповідно, було виконано множення вектора проекції з правої камери pR, транспонованого вектора, фундаментальної матриці F та вектора проекції з лівої камери pL. Результати обчислень були виведені для кожної точки.

Аналіз показав, що обчислені значення наближені до нуля, їх 81, що підтверджує правильність виконання проекцій та відповідність фундаментальній матриці. Невеликі відхилення можуть бути зумовлені чисельними похибками під час обчислень.

Завдання 9

Розрахувати матриці гомографії для виконання ректифікації пар стереозображень (для обраних камер). Для цього знайти спільний напрямок візирування, а також обрати деякі спільні внутрішні параметри камери (значення матриці внутрішніх параметрів камери). Можна використати параметри однієї з камер, або ж обрати деякі середні значення (для зручності можна взяти, наприклад, $f_x = f_y = \frac{f_{xL}+f_{xR}+f_{yL}+f_{yR}}{4}, c_x = \frac{W}{2}, c_y = \frac{H}{2}$, $W \times H$ – розміри зображення).

Результат виконання:

```
Camera 1 Homography matrix:
[[0.699 0.249 125.630]
 [-0.314 0.897 71.066]
 [-0.000 0.000 1.098]]


Camera2 Homography matrix:
[[0.945 0.491 3.385]
 [-0.587 0.915 251.298]
 [-0.000 -0.000 1.060]]
```

З результатів бачимо, що відбулось перетворення зображень з двох камер так, щоб їхні епіполярні лінії стали горизонтальними, що спрощує подальшу обробку. Для початку потрібно визначити спільний напрямок візирування обох камер. Це досягається шляхом обчислення матриць обертання R1 і R2,

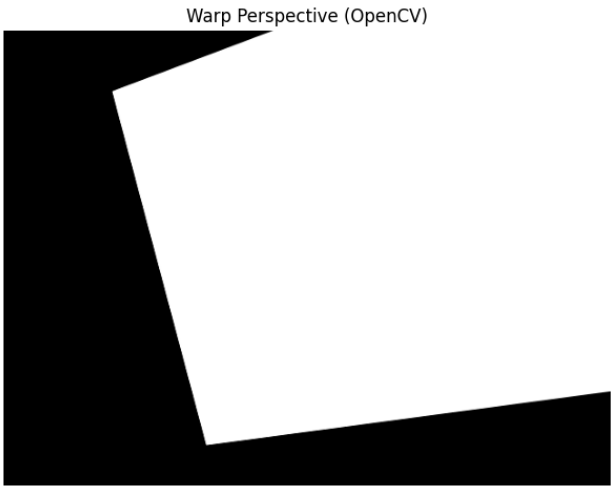
використовуючи відповідні векторні параметри обертання камер rvec1 та rvec2. Потім обчислюється відносне обертання R і трансляція T між камерами. Відносна трансляція враховує відстань між камерами та їхню орієнтацію у просторі. Далі для ректифікації використовується функція cv2.stereoRectify. Цей метод обчислює нові матриці обертання R1rect і R2rect для кожної з камер, а також нові проєкційні матриці P1 і P2. Крім того, формується матриця Q, яка може використовуватися для реконструкції глибини, та області ROI (region of interest), що вказують на зони ефективного зображення. На етапі стандартизації параметрів обирається спільна матриця внутрішніх параметрів камер. Це дозволяє працювати з єдиною системою координат для обох зображень. Для спрощення розрахунків значення фокусної відстані fx і fy можуть бути усереднені на основі параметрів обох камер, а центр зображення встановлюється як середина кадру. На основі обчислених параметрів формується матриця гомографії H1 та H2 для кожної з камер. Ці матриці дозволяють перетворити оригінальні зображення таким чином, щоб вони відповідали одній системі координат. Результати зберігаються у вигляді матриць, які можна застосувати до стереозображень для їхньої ректифікації.

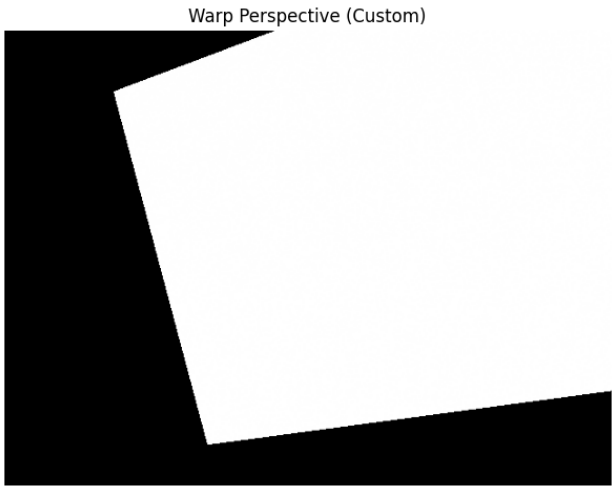
Завдання 10

Спроектувати декілька довільних тривимірних точок на зображення обидвох камер, а далі виконати ректифікацію цих зображень за допомогою матриць гомографії, отриманих в п. 9. Отримані зображення (оригінальні та ректифіковані) навести в звіті (для цього зручно зобразити точки різними кольорами). Показати, що на ректифікованих зображеннях відповідні точки знаходяться в одному рядку (а на не ректифікованих – ні).

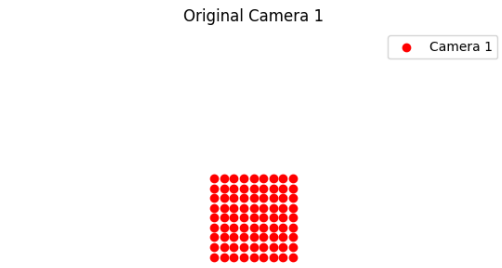
Результат виконання:

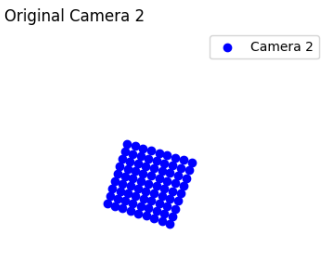
```
Comparison of warp_perspective methods: 0.0008845340934576444
```

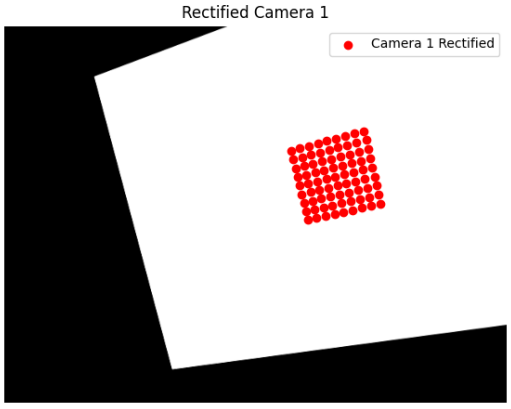


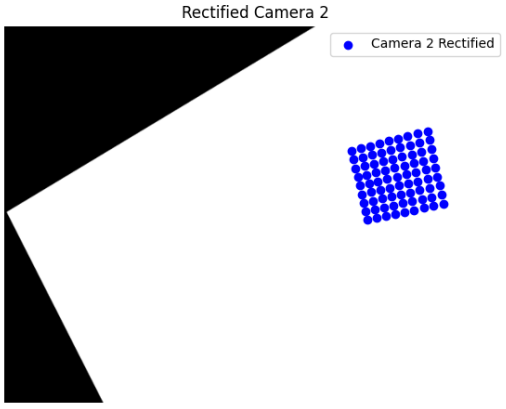



```
Comparison of warp_perspective methods: 5.223436719786606e-11
```










```
Original y-coordinates (Camera 1): [190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000
 190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000
 190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000
 190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000
 190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000
 190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000
 190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000
 190.000 202.500 215.000 227.500 240.000 252.500 265.000 277.500 290.000]
Original y-coordinates (Camera 2): [145.943 155.627 165.248 174.809 184.309 193.749 203.130 212.451 221.715
 148.807 158.525 168.181 177.776 187.309 196.782 206.195 215.549 224.844
 151.697 161.450 171.141 180.769 190.336 199.842 209.288 218.674 228.001
 154.613 164.402 174.127 183.790 193.391 202.930 212.409 221.827 231.186
 157.557 167.381 177.141 186.839 196.474 206.047 215.558 225.009 234.400
 160.527 170.388 180.183 189.916 199.585 209.192 218.736 228.220 237.643
 163.526 173.422 183.254 193.021 202.724 212.365 221.944 231.460 240.916
 166.553 176.485 186.352 196.155 205.893 215.568 225.180 234.730 244.219
 169.608 179.577 189.480 199.318 209.092 218.801 228.447 238.030 247.552]
```

```
Point 0: Rectified y-coordinates (Camera 1, Camera 2): (158.36, 158.36)
Point 1: Rectified y-coordinates (Camera 1, Camera 2): (169.48, 169.48)
Point 2: Rectified y-coordinates (Camera 1, Camera 2): (180.57, 180.57)
Point 3: Rectified y-coordinates (Camera 1, Camera 2): (191.63, 191.63)
Point 4: Rectified y-coordinates (Camera 1, Camera 2): (202.67, 202.67)
Point 5: Rectified y-coordinates (Camera 1, Camera 2): (213.67, 213.67)
Point 6: Rectified y-coordinates (Camera 1, Camera 2): (224.65, 224.65)
Point 7: Rectified y-coordinates (Camera 1, Camera 2): (235.60, 235.60)
Point 8: Rectified y-coordinates (Camera 1, Camera 2): (246.52, 246.52)
Point 9: Rectified y-coordinates (Camera 1, Camera 2): (155.33, 155.33)
Point 10: Rectified y-coordinates (Camera 1, Camera 2): (166.51, 166.51)
Point 11: Rectified y-coordinates (Camera 1, Camera 2): (177.67, 177.67)
Point 12: Rectified y-coordinates (Camera 1, Camera 2): (188.81, 188.81)
Point 13: Rectified y-coordinates (Camera 1, Camera 2): (199.91, 199.91)
Point 14: Rectified y-coordinates (Camera 1, Camera 2): (210.99, 210.99)
Point 15: Rectified y-coordinates (Camera 1, Camera 2): (222.03, 222.03)
Point 16: Rectified y-coordinates (Camera 1, Camera 2): (233.05, 233.05)
Point 17: Rectified y-coordinates (Camera 1, Camera 2): (244.04, 244.04)
Point 18: Rectified y-coordinates (Camera 1, Camera 2): (152.25, 152.25)
Point 19: Rectified y-coordinates (Camera 1, Camera 2): (163.51, 163.51)
Point 20: Rectified y-coordinates (Camera 1, Camera 2): (174.74, 174.74)
Point 21: Rectified y-coordinates (Camera 1, Camera 2): (185.95, 185.95)
Point 22: Rectified y-coordinates (Camera 1, Camera 2): (197.12, 197.12)
Point 23: Rectified y-coordinates (Camera 1, Camera 2): (208.27, 208.27)

Point 55: Rectified y-coordinates (Camera 1, Camera 2): (151.12, 151.12)
Point 56: Rectified y-coordinates (Camera 1, Camera 2): (162.65, 162.65)
Point 57: Rectified y-coordinates (Camera 1, Camera 2): (174.15, 174.15)
Point 58: Rectified y-coordinates (Camera 1, Camera 2): (185.61, 185.61)
Point 59: Rectified y-coordinates (Camera 1, Camera 2): (197.05, 197.05)
Point 60: Rectified y-coordinates (Camera 1, Camera 2): (208.46, 208.46)
Point 61: Rectified y-coordinates (Camera 1, Camera 2): (219.84, 219.84)
Point 62: Rectified y-coordinates (Camera 1, Camera 2): (231.18, 231.18)
Point 63: Rectified y-coordinates (Camera 1, Camera 2): (136.29, 136.29)
Point 64: Rectified y-coordinates (Camera 1, Camera 2): (147.93, 147.93)
Point 65: Rectified y-coordinates (Camera 1, Camera 2): (159.53, 159.53)
Point 66: Rectified y-coordinates (Camera 1, Camera 2): (171.11, 171.11)
Point 67: Rectified y-coordinates (Camera 1, Camera 2): (182.65, 182.65)
Point 68: Rectified y-coordinates (Camera 1, Camera 2): (194.16, 194.16)
Point 69: Rectified y-coordinates (Camera 1, Camera 2): (205.64, 205.64)
Point 70: Rectified y-coordinates (Camera 1, Camera 2): (217.09, 217.09)
Point 71: Rectified y-coordinates (Camera 1, Camera 2): (228.52, 228.52)
Point 72: Rectified y-coordinates (Camera 1, Camera 2): (132.98, 132.98)
Point 73: Rectified y-coordinates (Camera 1, Camera 2): (144.70, 144.70)
Point 74: Rectified y-coordinates (Camera 1, Camera 2): (156.38, 156.38)
Point 75: Rectified y-coordinates (Camera 1, Camera 2): (168.03, 168.03)
Point 76: Rectified y-coordinates (Camera 1, Camera 2): (179.65, 179.65)
Point 77: Rectified y-coordinates (Camera 1, Camera 2): (191.23, 191.23)
Point 78: Rectified y-coordinates (Camera 1, Camera 2): (202.79, 202.79)
Point 79: Rectified y-coordinates (Camera 1, Camera 2): (214.32, 214.32)
Point 80: Rectified y-coordinates (Camera 1, Camera 2): (225.81, 225.81)
```

У процесі виконання завдання спочатку було згенеровано кілька тривимірних точок, які проєктувалися на зображення двох камер. Ці точки представляють

собою синтетичні дані, які створені для моделювання проєкції тривимірного простору на площину камер. Проєкція виконувалася з урахуванням параметрів камер, таких як матриці калібрування, обертання та зсув.

Після цього отримані зображення камер було піддано ректифікації за допомогою гомографічних матриць, отриманих у попередньому пункті (п. 9). Цей етап полягає у трансформації зображень таким чином, щоб епіполярні лінії двох камер стали паралельними і горизонтальними. Використовувалися як стандартні функції бібліотеки OpenCV, так і власна реалізація методу трансформації з використанням матриць гомографії для перевірки точності алгоритму.

На оригінальних та ректифікованих зображеннях точки було відображено різними кольорами для зручності візуалізації. У результаті, на оригінальних зображеннях відповідні точки двох камер знаходилися на різних висотах (y-координатах), що є типовим для неректифікованих зображень. Після ректифікації точки відповідали однаковим y-координатам, що демонструє правильність проведеної трансформації.

Крім цього, виконувалася оцінка відповідності результатів OpenCV та кастомної реалізації гомографічної трансформації, а також аналіз координат точок до та після ректифікації. У звіті наведено графічні результати всіх етапів: оригінальні зображення з проєкцією точок, ректифіковані зображення з відповідними точками, а також числові результати аналізу координат точок до і після трансформації.

Висновок:

Дана робота передбачала розробку програм для роботи із зображеннями, який реалізує основні аспекти комп’ютерного зору та калібрування камер. Основна частина завдань пов’язана із симуляцією та математичним моделюванням процесів проектування точок на площину зображення за допомогою моделі камери-стенопа, а також калібруванням камер із використанням бібліотеки OpenCV. Робота передбачала використання таких основних компонентів: завантаження зображень, моделювання параметрів камер, проектування точок, симуляція калібрувальних даних, розрахунок параметрів камер та їх валідація. У завданні необхідно було виконати калібрування однієї камери, а також стерео-калібрування пари камер із довільно заданими параметрами, що ускладнюють задачу. Крім цього, має верифікуватись отриманий результат шляхом перевірки математичних залежностей та геометричних умов, зокрема, виконання рівності для фундаментальної матриці. Лабораторна робота також охоплювала роботу з матрицями гомографії для виконання ректифікації пар стереозображень, демонструючи коректність отриманих результатів за допомогою візуалізації. Для цього передбачалось графічне представлення проекцій точок і зображень до та після ректифікації.

Додаток А:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

np.random.seed(42)
np.set_printoptions(formatter={'float': '{:.3f}'.format})

points_3d = np.array([[x, y, 0] for x in np.linspace(-0.2, 0.2, 5) for y
in np.linspace(-0.2, 0.2, 5)], dtype=np.float32)
x = points_3d[:, 0]
y = points_3d[:, 1]
z = points_3d[:, 2]

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, c='blue', label='Points')

ax.set_title("3D Visualization of Points")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.legend()

plt.show()
image_size = (640, 480)

focal_length1 = 500
cx1, cy1 = image_size[0] // 2, image_size[1] // 2
camera_matrix1 = np.array([
    [focal_length1, 0, cx1],
    [0, focal_length1, cy1],
    [0, 0, 1]
], dtype=np.float32)

rvec1 = np.zeros(3, dtype=np.float32)
tvec1 = np.array([0.0, 0.0, 1.0], dtype=np.float32)
def rodrigues(rvec):
    if rvec.shape == (3, 1) or rvec.shape == (1, 3) or rvec.shape == (3,):
        theta = np.linalg.norm(rvec)
        if theta == 0:
            return np.eye(3)

        k = rvec / theta
        K = np.array([
            [0, -k[2], k[1]],
            [k[2], 0, -k[0]],
            [-k[1], k[0], 0]
        ])
```

```
        R = np.eye(3) + np.sin(theta) * K + (1 - np.cos(theta)) *
np.dot(K, K)
        return R
    elif rvec.shape == (3, 3):
        R = rvec
        theta = np.arccos((np.trace(R) - 1) / 2)
        if np.isclose(theta, 0):
            return np.zeros((3, 1))

        k = np.array([
            R[2, 1] - R[1, 2],
            R[0, 2] - R[2, 0],
            R[1, 0] - R[0, 1]
        ]) / (2 * np.sin(theta))
        rvec = theta * k
        return rvec.reshape(3, 1)
    else:
        raise ValueError("Input must be a 3x1 rotation vector or a 3x3
rotation matrix.")
print(f'Comparison of Rodrigues methods: {((rodrigues(rvec1) -
cv2.Rodrigues(rvec1)[0])**2).mean()}')
import numpy as np
import cv2
import matplotlib.pyplot as plt

def project_points(points_3d, camera_matrix, rvec, tvec):
    points_2d, _ = cv2.projectPoints(points_3d, rvec, tvec, camera_matrix,
None)
    return points_2d.reshape(-1, 2)

def project_points_from_scratch(points_3d, camera_matrix, rvec, tvec):
    R, _ = cv2.Rodrigues(rvec)

    tvec = tvec.reshape(3, 1)

    points_cam = (R @ points_3d.T) + tvec
    points_cam = points_cam.T

    valid_mask = points_cam[:, 2] > 0
    if not np.any(valid_mask):
        raise ValueError("All points are behind the camera or on the
camera plane.")

    valid_points_cam = points_cam[valid_mask]

    x = valid_points_cam[:, 0] / valid_points_cam[:, 2]
    y = valid_points_cam[:, 1] / valid_points_cam[:, 2]

    fx = camera_matrix[0, 0]
    fy = camera_matrix[1, 1]
    cx = camera_matrix[0, 2]
```

```
    cy = camera_matrix[1, 2]

    u = fx * x + cx
    v = fy * y + cy

    points_2d = np.full((points_3d.shape[0], 2), np.nan, dtype=np.float32)
    points_2d[valid_mask] = np.column_stack((u, v))

    return points_2d

projected_points1 = project_points(points_3d, camera_matrix1, rvec1,
tvec1)
projected_points12 = project_points_from_scratch(points_3d,
camera_matrix1, rvec1, tvec1)

print(f'Comparison of project_points methods: {((projected_points12 -
projected_points1)**2).mean()}')
image1 = np.ones((image_size[1], image_size[0], 3), dtype=np.uint8) *
255  # White image
for point in projected_points1:
    cv2.circle(image1, (int(point[0]), int(point[1])), 5, (0, 0, 255), -
1)  # Draw points in red

plt.figure(figsize=(6, 6))
plt.imshow(image1)
plt.scatter(projected_points1[:, 0], projected_points1[:, 1], c='r',
label="Camera 1")
plt.title("Original Camera 1")
plt.legend()
plt.axis('off')

plt.show()
object_points = [points_3d] * 10
image_points1 = []

for i in range(10):
    rvec = np.random.uniform(-1, 1, 3)
    tvec = np.random.uniform(-1, 1, 3)
    points = project_points(points_3d, camera_matrix1, rvec, tvec)
    image_points1.append(points + np.random.normal(0, 0.5,
points.shape).astype(np.float32))

# image_points1 = [(projected_points1 + np.random.normal(0, 0.5,
projected_points1.shape).astype(np.float32)) for _ in range(10)]

ret1, mtx1, dist1, rvecs1, tvecs1 = cv2.calibrateCamera(
    object_points, image_points1, image_size, camera_matrix1, None
)


print("\nCamera 1 Parameters:")
```

```
print("Initial Camera Matrix:")
print(camera_matrix1)
print("Calibrated Camera Matrix:")
print(mtx1)



focal_length2 = 400
cx2, cy2 = image_size[0] // 2 - 50, image_size[1] // 2 - 50
camera_matrix2 = np.array([
    [focal_length2, 0, cx2],
    [0, focal_length2, cy2],
    [0, 0, 1]
], dtype=np.float32)

rvec2 = np.array([0.1, 0.2, 0.3])
tvec2 = np.array([0.5, 0.1, 1])

projected_points2 = project_points(points_3d, camera_matrix2, rvec2,
tvec2)

image2 = np.ones((image_size[1], image_size[0], 3), dtype=np.uint8) * 255
for point in projected_points2:
    cv2.circle(image2, (int(point[0]), int(point[1])), 5, (0, 255, 0), -
1)

plt.figure(figsize=(6, 6))
plt.imshow(image2)
plt.scatter(projected_points2[:, 0], projected_points2[:, 1], c='g',
label="Camera 2")
plt.title("Original Camera 2")
plt.legend()
plt.axis('off')

plt.show()

image_points2 = []

for i in range(10):
    rvec = np.random.uniform(-1, 1, 3)
    tvec = np.random.uniform(-1, 1, 3)
    points = project_points(points_3d, camera_matrix2, rvec, tvec)
    image_points2.append(points + np.random.normal(0, 0.5,
projected_points2.shape).astype(np.float32))

# image_points2 = [projected_points2 + np.random.normal(0, 0.5,
projected_points2.shape).astype(np.float32) for _ in range(10)]
ret2, mtx2, dist2, rvecs2, tvecs2 = cv2.calibrateCamera(object_points,
image_points2, image_size, None, None)

print("\nCamera 2 Parameters:")
```

```
print("Initial Camera Matrix:")
print(camera_matrix2)
print("Calibrated Camera Matrix:")
print(np.round(mtx2))

ret_stereo, cm1, dc1, cm2, dc2, R, T, E, F = cv2.stereoCalibrate(
    object_points, image_points1, image_points2, mtx1, dist1, mtx2, dist2,
image_size, None, None, None, None,
    flags=cv2.CALIB_FIX_INTRINSIC, criteria=(cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-6)
)
print("\nCamera 1 Parameters:")
print("Initial Camera Matrix:")
print(camera_matrix1)
print("Stereo calibrated Camera Matrix:")
print(cm1)

print("\nCamera 2 Parameters:")
print("Initial Camera Matrix:")
print(camera_matrix2)
print("Stereo calibrated Camera Matrix:")
print(cm2)
def vector_product(t):
    return np.array([
        [0, -t[2], t[1]],
        [t[2], 0, -t[0]],
        [-t[1], t[0], 0]
    ])

T_ = vector_product(T.flatten())

E_custom = T_ @ R

F_custom = np.linalg.inv(cm2.T) @ E_custom @ np.linalg.inv(cm1)

F_custom_norm = F_custom / F_custom[-1, -1]
print(f'Essential matrix | stereoCalibrate:\n{E}')
print(f'\nEssential matrix | Custom:\n{E_custom}')

print(f'\n\nFundamental matrix | stereoCalibrate:\n{F}')
print(f'\nFundamental matrix | Custom:\n{F_custom_norm}')
points_3d = np.array([[x, y, 1] for x in np.linspace(-0.2, 0.2, 9) for y
in np.linspace(-0.2, 0.2, 9)], dtype=np.float32)
projected_points1, _ = cv2.projectPoints(points_3d, rvec1, tvec1,
camera_matrix1, None)
projected_points2, _ = cv2.projectPoints(points_3d, rvec2, tvec2,
camera_matrix2, None)

pL = np.hstack((projected_points1.squeeze(), np.ones((len(points_3d), 1))
* points_3d[0, -1]))
```

```
pR = np.hstack((projected_points2.squeeze(), np.ones((len(points_3d), 1))
* points_3d[0, -1]))

for i in range(len(points_3d)):
    result = pR[i].T @ F @ pL[i]
    print(f"Point {i + 1}: pR^T * F * pL = {result:.6f}")

R1, _ = cv2.Rodrigues(rvec1)
R2, _ = cv2.Rodrigues(rvec2)

R = np.dot(R2, R1.T)
T = tvec2 - np.dot(R, tvec1)

R1_rect, R2_rect, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(
    camera_matrix1, None, camera_matrix2, None, image_size, R, T, alpha=0
)

focal_length3 = (focal_length1 + focal_length2) / 2
camera_matrix3 = np.array([[focal_length3, 0, image_size[0] / 2],
                    [0, focal_length3, image_size[1] / 2],
                    [0, 0, 1]], dtype=np.float64)

H1 = np.dot(np.dot(camera_matrix3, R1_rect),
np.linalg.inv(camera_matrix1))
H2 = np.dot(np.dot(camera_matrix3, R2_rect),
np.linalg.inv(camera_matrix2))

print('Camera 1 Homography matrix:')
print(H1)

print('\n\nCamera2 Homography matrix:')
print(H2)
image1 = np.ones((image_size[1], image_size[0], 3), dtype=np.uint8) *
255
image2 = np.ones((image_size[1], image_size[0], 3), dtype=np.uint8) *
255
import numpy as np

def warp_perspective(image, H, output_size):
    width, height = output_size

    H_inv = np.linalg.inv(H)

    if len(image.shape) == 3:
        warped_image = np.zeros((height, width, image.shape[2]),
dtype=image.dtype)
    else:
        warped_image = np.zeros((height, width), dtype=image.dtype)

    for y in range(height):
        for x in range(width):
```

```
            src_coords = np.dot(H_inv, np.array([x, y, 1]))
            src_coords /= src_coords[2]
            u, v = src_coords[:2]

            if 0 <= u < image.shape[1] - 1 and 0 <= v < image.shape[0] -
1:
                x0, y0 = int(u), int(v)
                x1, y1 = x0 + 1, y0 + 1

                a, b = u - x0, v - y0
                if len(image.shape) == 3:
                    pixel_value = (
                        (1 - a) * (1 - b) * image[y0, x0] +
                        a * (1 - b) * image[y0, x1] +
                        (1 - a) * b * image[y1, x0] +
                        a * b * image[y1, x1]
                    )
                else:
                    pixel_value = (
                        (1 - a) * (1 - b) * image[y0, x0] +
                        a * (1 - b) * image[y0, x1] +
                        (1 - a) * b * image[y1, x0] +
                        a * b * image[y1, x1]
                    )

                warped_image[y, x] = pixel_value

    return warped_image

wp_opencv = cv2.warpPerspective(image1, H1, image_size) / 255.0
wp_custom = warp_perspective(image1, H1, image_size) / 255.0

print(f'Comparison of warp_perspective methods: {((wp_custom -
wp_opencv)**2).mean()}')
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(wp_opencv)
plt.title("Warp Perspective (OpenCV)")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(wp_custom)
plt.title("Warp Perspective (Custom)")
plt.axis('off')

plt.tight_layout()
plt.show()

def perspective_transform(points, H):
    transformed_points = []
```

```
    for point in points:
        x, y = point
        homogeneous_point = np.array([x, y, 1.0])

        transformed_point = np.dot(H, homogeneous_point)

        transformed_point /= transformed_point[2]
        transformed_points.append(transformed_point[:2])

    return np.array(transformed_points)
projected_points1 = projected_points1.squeeze()

ps_opencv = cv2.perspectiveTransform(np.expand_dims(projected_points1,
axis=0), H1).squeeze()
ps_custom = perspective_transform(projected_points1, H1).squeeze()

print(f'Comparison of warp_perspective methods: {((ps_custom -
ps_opencv)**2).mean()}')

projected_points1, _ = cv2.projectPoints(points_3d, rvec1, tvec1,
camera_matrix1, None)
projected_points2, _ = cv2.projectPoints(points_3d, rvec2, tvec2,
camera_matrix2, None)
projected_points1 = projected_points1.squeeze()
projected_points2 = projected_points2.squeeze()

rectified_image1 = cv2.warpPerspective(image1, H1, image_size)
rectified_image2 = cv2.warpPerspective(image2, H2, image_size)

rectified_points1 =
cv2.perspectiveTransform(np.expand_dims(projected_points1, axis=0),
H1).squeeze()
rectified_points2 =
cv2.perspectiveTransform(np.expand_dims(projected_points2, axis=0),
H2).squeeze()

plt.figure(figsize=(18, 9))

plt.subplot(2, 2, 1)
plt.imshow(image1)
plt.scatter(projected_points1[:, 0], projected_points1[:, 1], c='r',
label="Camera 1")
plt.title("Original Camera 1")
plt.legend()
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(image2)
plt.scatter(projected_points2[:, 0], projected_points2[:, 1], c='b',
label="Camera 2")
plt.title("Original Camera 2")
```

```
plt.legend()
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(rectified_image1)
plt.scatter(rectified_points1[:, 0], rectified_points1[:, 1], c='r',
label="Camera 1 Rectified")
plt.title("Rectified Camera 1")
plt.legend()
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(rectified_image2)
plt.scatter(rectified_points2[:, 0], rectified_points2[:, 1], c='b',
label="Camera 2 Rectified")
plt.title("Rectified Camera 2")
plt.legend()
plt.axis('off')

plt.tight_layout()
plt.show()


print("Original y-coordinates (Camera 1):", projected_points1[:, 1])
print("Original y-coordinates (Camera 2):", projected_points2[:, 1])

for i, (y1, y2) in enumerate(zip(rectified_points1[:, 1],
rectified_points2[:, 1])):
    print(f"Point {i}: Rectified y-coordinates (Camera 1, Camera 2):
({y1:.2f}, {y2:.2f})")
```