Моделирование на тему: «Гомография»

Выполнил: Курепин Даниил Денисович

Группа: M32041

**Задачи моделирования:**

Получить изображение, на котором присутствуют искажения пространства, например, квадрат превращается в трапецию при фотографии с определенного угла. Провести проективное преобразование более сложного объекта, чем квадрат. В результате обработки изображения получить неискаженное изображение объекта.

**Гомография (Компьютерное зрение) - Homography (Computer Vision):**

В области компьютерного зрения любые два изображения одной и той же плоской поверхности в пространстве связаны гомографией. Это имеет много практических применений, таких как исправление изображения, регистрация изображения или перемещение камеры — поворот и перевод — между двумя изображениями. После того, как срез камеры выполнен на основе предполагаемой матрицы гомографии, эта информация может быть использована для навигации или для вставки моделей 3D-объектов в изображение или видео, чтобы они отображались в правильной перспективе и выглядели как часть исходной сцены .

**Что такое гомография?**

Гомография – это преобразование (матрица 3 х 3), которое отражает точки одного изображения в точки соответствия другого изображения.

Теперь, поскольку гомография является матрицей 3 х 3, мы можем записать ее как:

Рассмотрим первый набор соответствующих точек – в первом изображении и во втором. Далее гомография H отображает их следующим образом:

**Как расчитать гомографию?**

Чтобы рассчитать гомографию между двумя изображениями, нужно знать как минимум 4 точки соответствия между двумя изображениями.

Если таких точек больше, то это даже лучше. OpenCV надежно проведет оценку гомографии по всем точкам наилучшим образом. Обычно эти точки соответствия обнаруживаются автоматически путем сопоставления между изображениями таких функций, как SIFT или SURF.

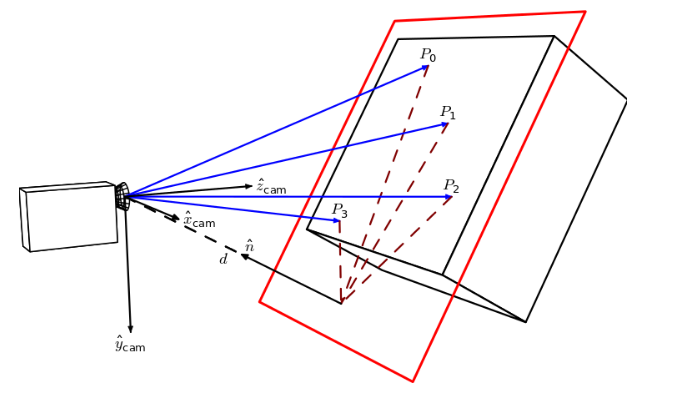
**Получим 3D уравнение от плоскости к плоскости:**

У нас есть две камеры a и b, которые смотрят на точки в плоскости. Переход от проекции в b к проекции в a:

Где и – координаты z для P в каждом кадре камеры и где матрица гомографии задается

R – матрица поворота, с помощью которой b поворачивается относительно a; t – вектор перемещения из a в b; n и d – вектор нормали плоскости и расстояние от начала координат до плоскости соответственно.

Ka и Kb - это матрицы внутренних параметров камер.



На рисунке показана камера *b*, смотрящая на плоскость с расстояния *d*.

**Примечание**: Из приведенного выше рисунка, предполагаемого {\displaystyle n^{T}P\_{i}+d=0} в качестве модели плоскости,

{\displaystyle n^{T}P\_{i}} является проекцией вектора  {\displaystyle P\_{i}} вдоль {\displaystyle n}nnn и равна -d{\displaystyle -d}. Итак {\displaystyle t=t\cdot 1=t\left(-{\frac {n^{T}P\_{i}}{d}}\right)}. И у нас есть {\displaystyle H\_{ab}P\_{i}=RP\_{i}+t}где {\displaystyle H\_{ab}=R-{\frac {tn^{T}}{d}}}.

Эта формула действительна только в том случае, если камера *b* не имеет поворота и перемещения.

В общем случае, где {\displaystyle R\_{a},R\_{b}}\ и  {\displaystyle t\_{a},t\_{b}} являются соответствующими поворотами и перемещениями камер *a* и *b*, {\displaystyle R=R\_{a}R\_{b}^{T}} и матрица  {\displaystyle H\_{ab}}гомографии становится

, {\displaystyle H\_{ab}=R\_{a}R\_{b}^{T}-{\frac {(-R\_{a}\*R\_{b}^{T}\*t\_{b}+t\_{a})n^{T}}{d}}}где *d* - расстояние камеры *b* до плоскости.

Матрица гомографии может быть вычислена только между изображениями, полученными с одной и той же камеры, снятыми под разными углами. Не имеет значения, что присутствует на изображениях. Матрица содержит искаженную форму изображений.

**Аффинная гомография**

Когда область изображения, в которой вычисляется гомография, мала или изображение было получено с большим фокусным расстоянием, аффинная гомография является более подходящей моделью смещений изображения. Аффинная гомография - это особый тип общей гомографии , последняя строка которой привязана к

**Реализация гомографии (код):**

from tkinter import \*  
import cv2  
import numpy as np  
from PIL import Image, ImageTk  
  
  
# Ниже можно увидеть две функции, основная из них get\_gomography, а вторая get\_rotation\_of\_image  
# get\_gomography - функция для получения гомографии  
# get\_rotation\_of\_image - функция для получения поворота изображения  
# Работа функций осуществляется с помощью библиотеки cv2 (OpenCV) и библиотеки numpy  
# Все функции работают с изображениями, которые находятся в папке Photos  
# К сожалению, пути к изображению приходится вводить вручную, поскольку это сильно усложняет работу с функциями  
  
def get\_rotation\_of\_image(path):  
 new\_window = Toplevel()  
  
 # Чтение изображения  
 image = PhotoImage(file="C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\cube1.png")  
 img = cv2.imread("C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\cube1.png", 0)  
 height, width = img.shape[:2]  
 label = Label(new\_window, image=image)  
 label.pack()  
 label.place(x=0, y=0)  
  
 # Поворот изображения  
 new\_window.geometry(str(width \* 2) + "x" + str(height))  
 im\_src = cv2.imread("C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\cube1.png")  
 pts\_src = np.array([[100, 200], [400, 200], [100, 50], [400, 50]])  
 im\_dst = cv2.imread("C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\cube1.png")  
 pts\_dst = np.array([[150, 150], [430, 170], [130, 20], [440, 10]])  
  
 # Получение матрицы гомографии  
 h, status = cv2.findHomography(pts\_src, pts\_dst)  
  
 # Применение матрицы гомографии  
 im\_out = cv2.warpPerspective(im\_src, h, (im\_dst.shape[1], im\_dst.shape[0]))  
  
 # Вывод изображения  
 image\_output = np.array(im\_out)  
   
 # Перевод изображения в формат, который может быть прочитан библиотекой PIL  
 final\_output = ImageTk.PhotoImage(image=Image.fromarray(image\_output))  
 canvas = Label(new\_window, image=final\_output)  
 canvas.pack()  
 canvas.place(x=width, y=0)  
 new\_window.mainloop()  
  
  
# Функция для получения гомографии  
def get\_gomography(first\_path, second\_path):  
 # Чтение изображений  
 image1 = cv2.imread("C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\cube1.png")  
 image2 = cv2.imread("C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\cube2.png")  
  
 # Дальше идет подготовка изображений для поиска ключевых точек (Мы использвуем несколько методов библиотеки OpenCV)  
 # Инициализация SIFT детектора  
 detected\_image1 = cv2.cvtColor(image1, cv2.COLOR\_BGR2GRAY)  
 detected\_image2 = cv2.cvtColor(image2, cv2.COLOR\_BGR2GRAY)  
 orb = cv2.ORB\_create(50)  
  
 # Поиск ключевых точек и их дескрипторов  
 kp1, des1 = orb.detectAndCompute(detected\_image1, None)  
 kp2, des2 = orb.detectAndCompute(detected\_image2, None)  
  
 # Создание BFMatcher объекта  
 matcher = cv2.DescriptorMatcher\_create(cv2.DESCRIPTOR\_MATCHER\_BRUTEFORCE\_HAMMING)  
  
 # Использование knnMatch() для получения двух лучших совпадений  
 matches = matcher.match(des1, des2, None)  
 matches = sorted(matches, key=lambda x: x.distance)  
  
 # Отрисовка совпадений  
 points1 = np.zeros((len(matches), 2), dtype=np.float32)  
 points2 = np.zeros((len(matches), 2), dtype=np.float32)  
  
 # Получение координат точек  
 for i, match in enumerate(matches):  
 points1[i, :] = kp1[match.queryIdx].pt  
 points2[i, :] = kp2[match.trainIdx].pt  
 h = cv2.findHomography(points1, points2, cv2.RANSAC)[0]  
 height, width, channels = image2.shape  
  
 # Получение гомографии  
 im1Reg = cv2.warpPerspective(image1, h, (width, height))  
 img3 = cv2.drawMatches(detected\_image1, kp1, detected\_image2, kp2, matches[:10], None)  
 print(h)  
  
 # Вывод изображений  
 cv2.imshow('Keypoint matches', img3)  
 cv2.imshow('Registered image', im1Reg)  
 cv2.waitKey(0)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Здесь создается среда для работы с графическим интерфейсом  
 window = Tk()  
 window.geometry("680x500")  
 window.title("Homography")  
  
 # Загрузка иконки для окна  
 icon = PhotoImage(file="C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\icon.png")  
 window.iconphoto(True, icon)  
 background\_image = PhotoImage(file="C:\\Users\\DNK\\PycharmProjects\\pythonProject2\\Photos\\cube1.png")  
 background\_label = Label(window, image=background\_image)  
 background\_label.place(x=0, y=0, relwidth=1, relheight=1)  
  
 # Работа с графическим интерфейсом (кнопки, поля ввода и т.д.)  
 entry\_first = Entry(window)  
 entry\_first.pack()  
  
 # При работе с приложением не нужно вводить путь к файлу, так как он уже указан в коде  
 entry\_first.insert(0, "Path to first image")  
 entry\_first.place(x=22, y=115)  
 entry\_second = Entry(window)  
 entry\_second.pack()  
 entry\_second.insert(0, "Path to second image")  
 entry\_second.place(x=22, y=135)  
 entry = Entry(window, width=14)  
 entry.pack()  
 entry.insert(0, "Path")  
 entry.place(x=22, y=75)  
  
 # Кнопка для получения изображения с поворотом по точкам (заданы в функции get\_rotation\_of\_image)  
 button\_first = Button(window, text="Rotate Image",  
 command=lambda: get\_rotation\_of\_image(entry.get()),  
 font=("Arial", 10), fg="#00FF00", bg="black")  
 button\_first.pack()  
 button\_first.place(x=22, y=50)  
  
 # Кнопка для получения изображения с перспективным преобразованием (заданы в функции get\_gomography)  
 button\_second = Button(window, text="Get Homography",  
 command=lambda: get\_gomography(entry\_first.get(), entry\_second.get()),  
 font=("Arial", 10), fg="#00FF00", bg="black")  
 button\_second.pack()  
 button\_second.place(x=22, y=155)  
 window.mainloop()

**Примечание к коду:**

У данной библиотеки есть некоторые недостатки:

1. Если изображение будет одинаковой световой гаммы или будет очень сильный поворот, то функции библиотеки не смогут найти точки соединения и гомография будет найдена неверно, следовательно изображение построится неправильно.
2. Так же надо сказать что программа не умеет распознавать неверно введенные данные, то есть ответственность за правильность введенных данных ложится на совесть пользователя. К неверно введенным данным могут относиться к примеру разные картинки, засвеченные фотографии и т.д.
3. Для того, чтобы гомография была более точной, я использовал определнные маркеры на фото, чтобы OpenCV мог адекватно определить точки.

Стоит отметить, что, несмотря на недостатки, с помощью написанной программы, а также самой библиотеки можно достаточно четко получить гомографию. В силу того, что фотографии у нас в двумерном пространстве, дальше на результатах будет видно, что плоскость, точки – определяются правильно, но в силу того, что у нас объекты двумерные, например у куба не будет продолжения.

**Результаты работы кода:**

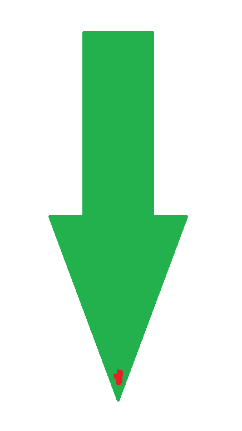
1. Я решил следовать принципу повышения сложности: сначала я использовал достаточно простой объект – стрелку зеленого цвета (Это сложнее, чем квадрат, но все же еще проще, чем например куб)

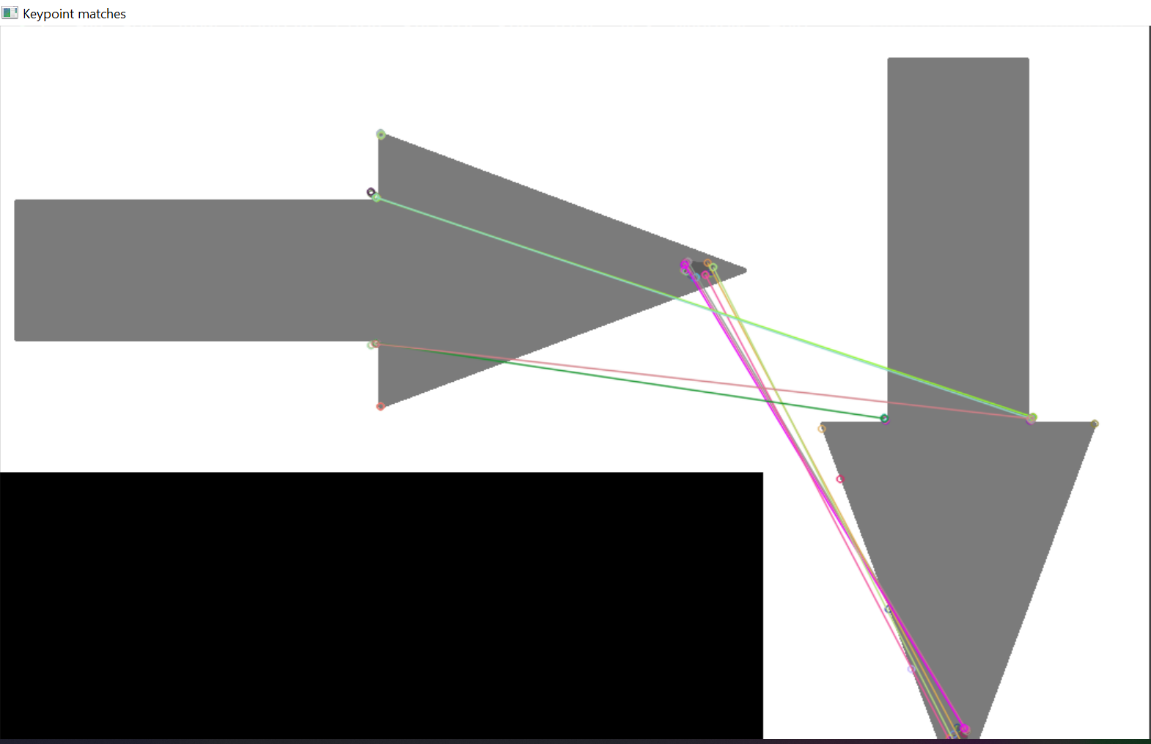
Изначальное фото:

Изображение выглядит как стрела

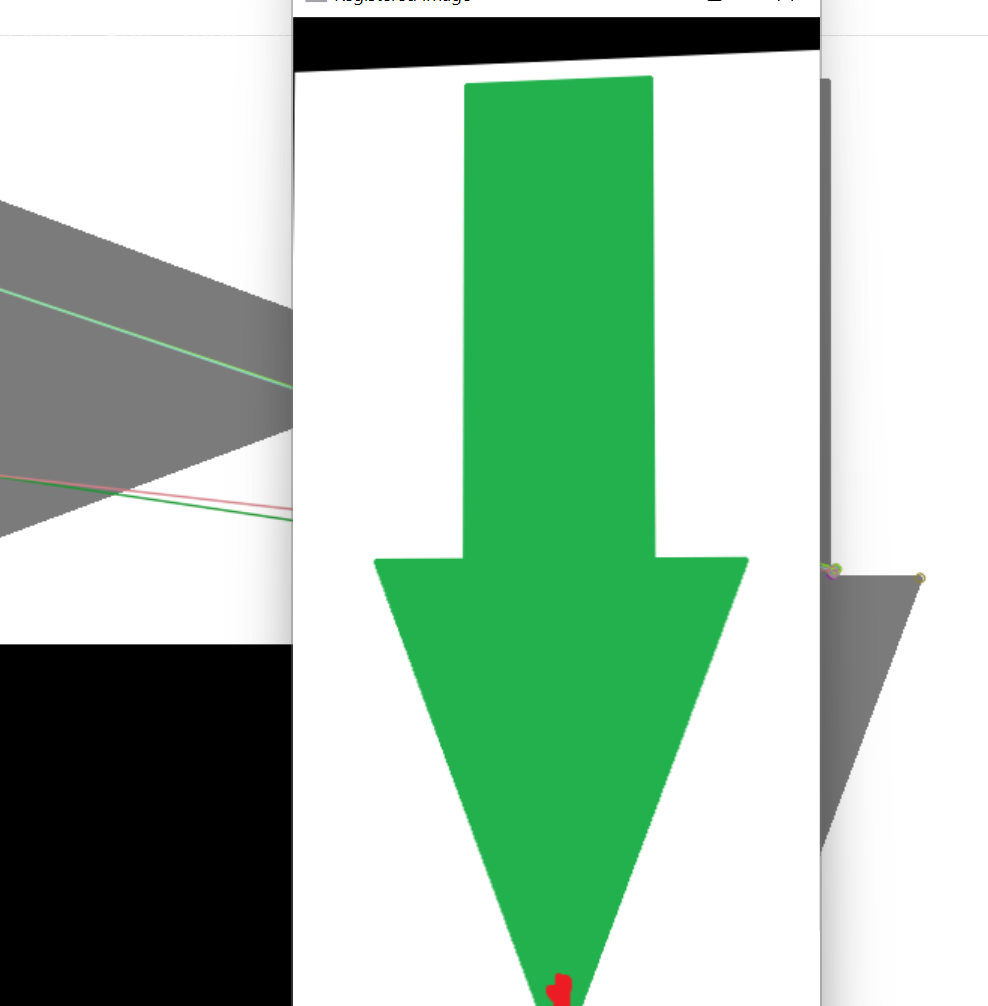
Автоматически созданное описание

Вторая фотография (референс для гомографии):



Результат поиска ключевых точек: 

Результат преобразования:



Результат преобразования соответствует второй фотографии, все получилось!

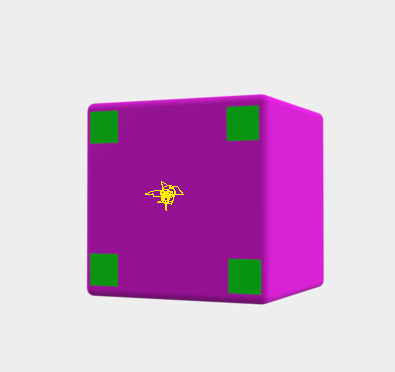
1. Дальше я сделал гомографию куба:

Изначальное фото:

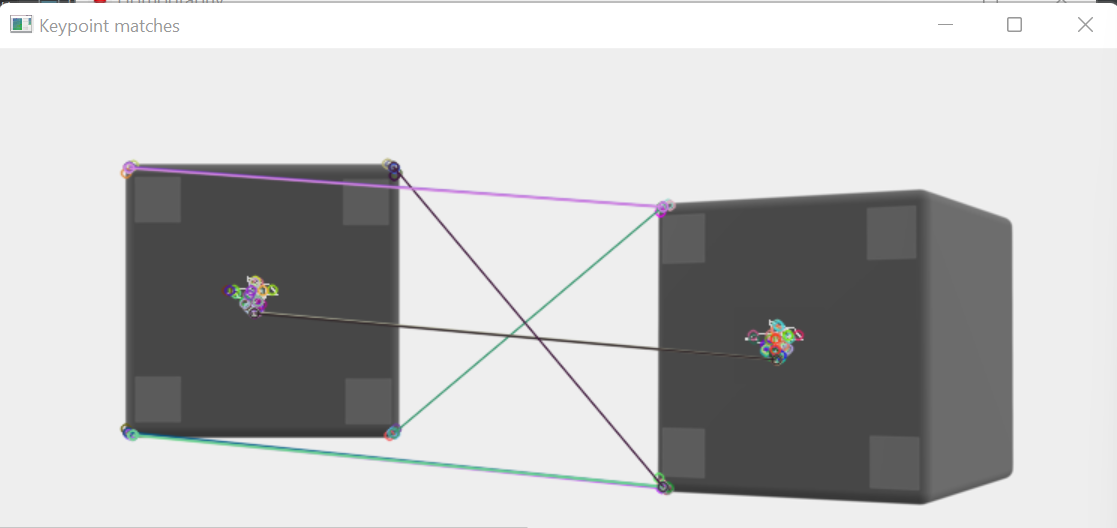
Изображение выглядит как квадрат

Автоматически созданное описание

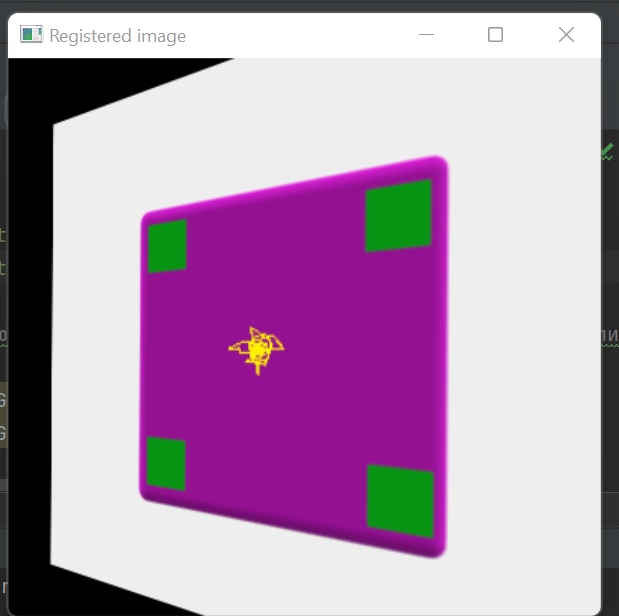
Второе фото:



Результат поиска ключевых точек:



Результат преобразования:



Как мы видим, результат преобразования соответствует нашей второй картинке, но из-за того, что это фотография и куб у нас здесь – это, по сути, квадрат, мы видим лишь то, что ракурс изменился и он такой же как и у второго фото, но если бы это была бы 3д модель, то у нас было бы полное соответствие.

1. Дальше была взяты две фото паспорта, замазаны вещи, которые потенциально могли повлиять в худшую сторону на нахождение ключевых точек, и было произведено преобразование:

Изначальное фото:

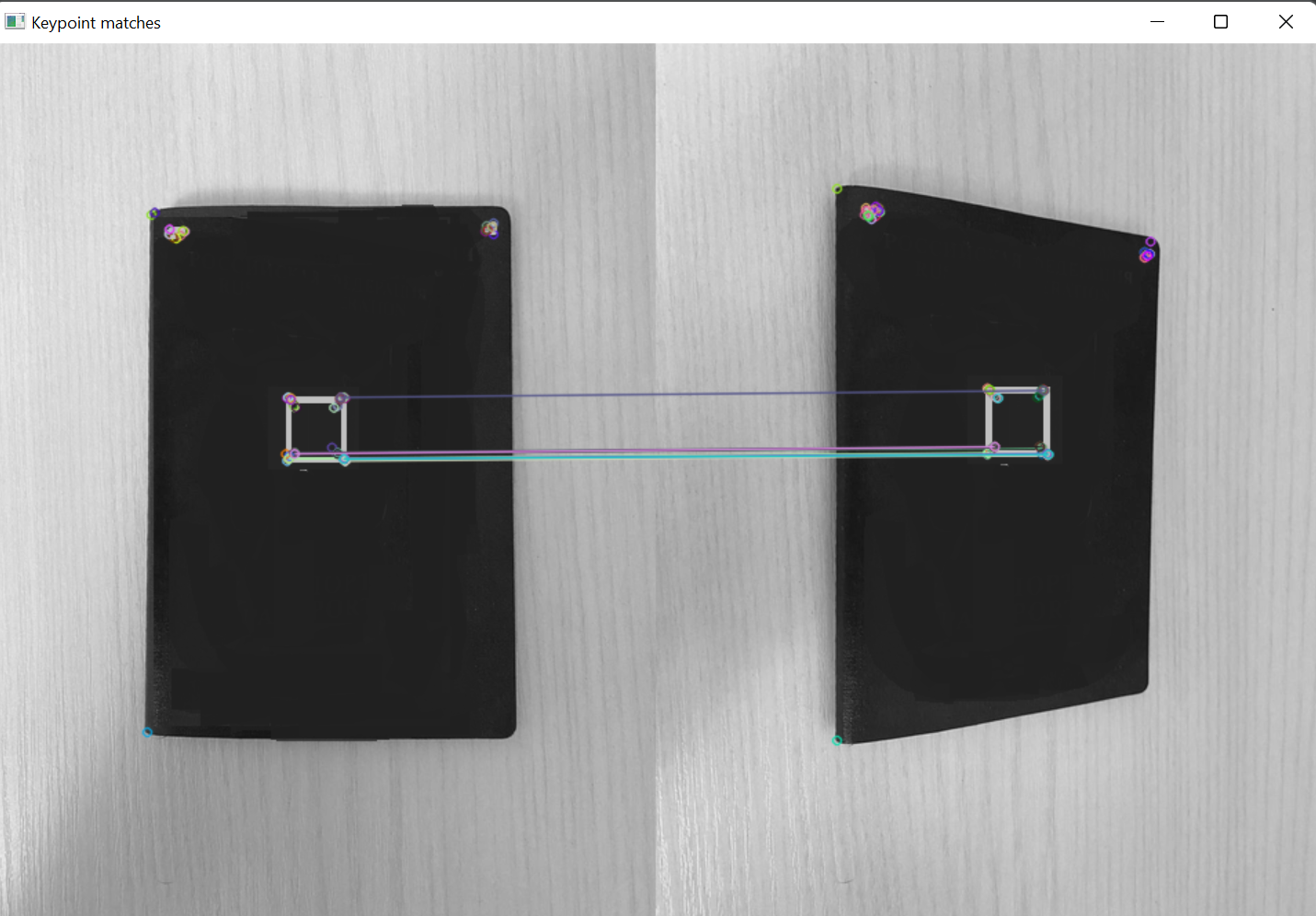


Второе фото с другого ракурса:

Изображение выглядит как текст

Автоматически созданное описание

Результат нахождения ключевых точек:



Результат преобразования:

Изображение выглядит как текст

Автоматически созданное описание

Результат хороший, все получилось, но из-за той же проблемы, что и в предыдущем пункте может казаться, что фото отличается.

**Выводы по моделированию:**

В ходе моделирования я получил изображения, на которых присутствуют искажения пространства, делав это с помощью двух фотографий с разных ракурсов. Я провел проективное преобразование сложных объектов и в результате получил искаженные изображения.

**!!! К отчету прилагается видео, которое подробно показывает как работает код и как работает преобразование (гомография) через написанные мной функции и встроенную библиотеку OpenCV**

**Также в гугл драйве есть полностью весь проект, исходные данные и отчет в .docx !!!**