Отчет по моделированию.

Гомография.

Группа:

Хлучин Георгий

Гумбатов Влад

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 x 3), которое отражает точки одного изображения в точки соответствия другого изображения. Теперь, поскольку гомография является матрицей 3 x 3, мы можем записать ее как

$$H = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix}$$

Рассмотрим первый набор соответствующих точек – (x_1, y_1) в первом изображении и (x_2, y_2) во втором. Далее гомография H отображает их следующим образом:

$$\begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = H \times \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix}$$

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.

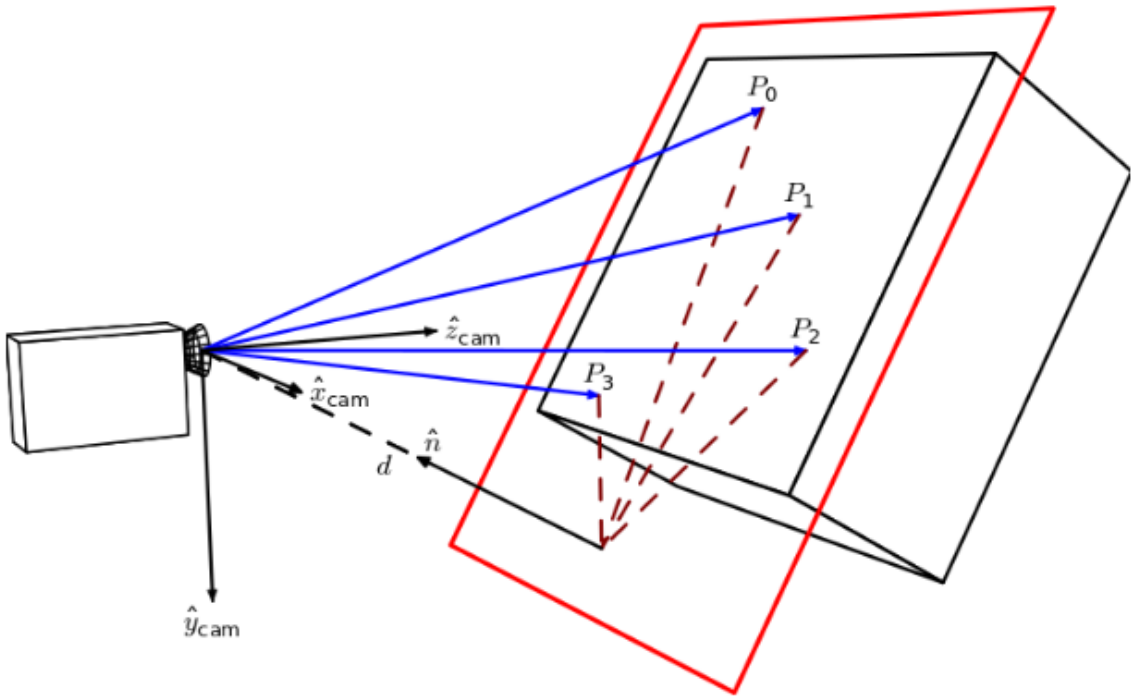
<u>3D уравнение от плоскости к плоскости.</u>

У нас есть две камеры a и b, которые смотрят на точки P_i в плоскости. Переход от проекции ${}^{b}p_i = ({}^{b}u_i;\ {}^{b}v_i; 1)\ P_i$ в b к проекции ${}^{a}P_i = ({}^{a}u_i;\ {}^{a}v_i; 1)\ P_i$ в a:

$${}^{a}p_i = \frac{{}^{b}z_i}{{}^{a}z_i} K_a \cdot H_{ab} \cdot K_b^{-1} \cdot {}^{b}p_i$$

Где ${}^{a}z_i$ и ${}^{b}z_i$ – координаты z для P в каждом кадре камеры и где матрица H_{ab}гомографии задается $H_{ab} = R - \frac{tn^T}{d}$

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 $n^T P_i + d = 0$ в качестве модели плоскости, $n^T P_i$ является проекцией вектора P_i вдоль n и равна -d. Итак $t = t \cdot 1 = t\ (-\frac{n^T P_i}{d})$. И у нас есть $H_{ab} P_i = RP_i + t$ где $H_{ab} = R - \frac{tn^T}{d}$.

Эта формула действительна только в том случае, если камера *b* не имеет поворота и перемещения. В общем случае, где R_a, R_b и $t_a; t_b$ являются соответствующими поворотами и перемещениями камер *a* и *b*, $R = R_a R_b^T$ и матрица H_{ab} гомографии становится

$$H_{ab} = R_a R_b^T - \frac{(-R_a * R_b^T * t_b + t_a) n^T}{d}$$

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 омография - это особый тип общей омографии , последняя строка которой привязана к $h_{31} = h_{32} = 0, h_{33} = 1$.

Код:

```
from tkinter import *
import cv2
import numpy as np
import io
from PIL import Image, ImageTk


def click_first_button(path):
    new_window = Toplevel()
    image =
PhotoImage(file="C:\\Users\\user\\PycharmProjects\\image_editor\\images\\img.
png")
    img =
cv2.imread("C:\\Users\\user\\PycharmProjects\\image_editor\\images\\img.png",
0)
    height, width = img.shape[:2]
    label = Label(new_window, image=image)
    label.pack()
    label.place(x=0, y=0)
    new_window.geometry(str(width*2)+"x"+str(height))

    im_src =
cv2.imread("C:\\Users\\user\\PycharmProjects\\image_editor\\images\\img.png")

    pts_src = np.array([ [100, 200], [400, 200], [100, 50],[400, 50] ])
    im_dst =
cv2.imread("C:\\Users\\user\\PycharmProjects\\image_editor\\images\\img.png")
    pts_dst = np.array([ [150, 150],[430, 170],[130, 20],[440, 10] ])
    h, status = cv2.findHomography(pts_src, pts_dst)

    im_out = cv2.warpPerspective(im_src, h,
(im_dst.shape[1],im_dst.shape[0]))

    image_ = np.array(im_out)
    image__ = ImageTk.PhotoImage(image=Image.fromarray(image_))

    canvas = Label(new_window, image=image__)
    canvas.pack()
    canvas.place(x=width, y=0)

    new_window.mainloop()



def click_second_button(first_path, second_path):
    im1 = cv2.imread('book1.jpg')
    im2 = cv2.imread('book2.jpg')

    img1 = cv2.cvtColor(im1, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)

    orb = cv2.ORB_create(50)

    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)

    matcher =
cv2.DescriptorMatcher_create(cv2.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING)

    matches = matcher.match(des1, des2, None)

    matches = sorted(matches, key=lambda x: x.distance)
```

```
    points1 = np.zeros((len(matches), 2), dtype=np.float32)
    points2 = np.zeros((len(matches), 2), dtype=np.float32)

    for i, match in enumerate(matches):
        points1[i, :] = kp1[match.queryIdx].pt
        points2[i, :] = kp2[match.trainIdx].pt

    h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)

    height, width, channels = im2.shape

    im1Reg = cv2.warpPerspective(im1, h, (width, height))

    img3 = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None)

    print(h)

    cv2.imshow('Keypoint matches', img3)
    cv2.imshow('Registered image', im1Reg)
    cv2.waitKey(0)


if __name__ == '__main__':
    window = Tk()
    window.geometry("680x400")
    window.title("image editor")

    icon =
PhotoImage(file="C:\\Users\\user\\PycharmProjects\\image_editor\\images\\img.
png")
    window.iconphoto(True, icon)

    background_image =
PhotoImage(file="C:\\Users\\user\\PycharmProjects\\image_editor\\images\\img.
png")
    background_label = Label(window, image=background_image)
    background_label.place(x=0, y=0, relwidth=1, relheight=1)

    entry_first = Entry(window)
    entry_first.pack()

    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=440, y=75)

    button_first = Button(window,
                    text="rotate image",
                    command=lambda: click_first_button(entry.get()),
                    font=("Comic Sans",10),
                    fg="#00FF00",
                    bg="black")
```

```
    button_first.pack()

    button_first.place(x=440, y=50)

    button_second = Button(window, text="get homography",
                           command=lambda:
click_second_button(entry_first.get(), entry_second.get()),
                           font=("Comic Sans", 10),
                           fg="#00FF00",
                           bg="black")

    button_second.pack()

    button_second.place(x=22, y=155)

    window.mainloop()
```

недостатки библиотеки:

Если изображение будет одинаковой световой гаммы или будет очень сильный поворот, то функции библиотеки не смогут найти точки соединения и гомография будет найдена неверно => изображение построится неправильно. 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 или картинки снимки которых сделаны с разных ракурсов.