بسم الله الرحمن الرحیم

درس بینایی ماشین

دکتر هدی محمد زاده

گزارش تمرین سری 4

امیرحسین رستمی 96101635

بهار 99

پاسخ بخش کامپیوتری:

سوال اول:

با اجازه تی ای محترم درس که استفاده از AKAZE را به جای SIFT اجازه دادند در این قسمت بنده به جای استفاده از روش SIFT از روش AKAZE استفاده می کنم(علت این تغییر انتخاب،یک سری incompatibility در کتاب خانه SIFT نسبت به روش های دیگر بود که کارکردن با آن کتاب خانه را سخت میکرد).

ابتدای خروجی روش هارا ذکر می کنیم:

1 ـ روش AKAZE:ابتدا نقاط کلیدیِ سمت چپ و راست تصاویر را نشان می دهیم و سپس به رسم می پردازیم:

نقاط کلیدی سمت راست

نقاط کلیدی سمت چپ

حال به کمک Matcher به رسم خطوط بین این نقاط کلیدی می پردازیم:

نکته 1 :در حالت عادی به خاطر شباهت زیاد دو تصویر تعداد نقاط کلیدی زیادی بین دو تصویر پیدا می شود و رسم کردن همه آن ها شکل نهایی را بسیار شلوغ می کند لذا بنده همانطورکه در کد نیز مشخص است بخشی از این نقاط را ترسیم کرده ام:

تصویر حاصل از اتصال نقاط کلیدی تصویر چپ و راست

2ـ روش ORB:ابتدا نقاط کلیدیِ سمت چپ و راست تصاویر را نشان می دهیم و سپس به رسم می پردازیم:

نقاط کلیدی تصویر سمت راست

نقاط کلیدی تصویر سمت چپ

حال به کمک Matcher به رسم خطوط بین این نقاط کلیدی می پردازیم:

نکته 1 :در حالت عادی به خاطر شباهت زیاد دو تصویر تعداد نقاط کلیدی زیادی بین دو تصویر پیدا می شود و رسم کردن همه آن ها شکل نهایی را بسیار شلوغ می کند لذا بنده همانطورکه در کد نیز مشخص است بخشی از این نقاط را ترسیم کرده ام:

تصویر حاصل از اتصال نقاط کلیدی تصویر چپ و راست

حال در صفحه بعد به مقایسه دو روش می پردازیم.

پس از انجام تست های مختلف به کمک این دو الگوریتم به نتایج زیر دست یافتیم:

به طور میانگین روش AKAZE نسبت به روش ORB از دقت بیشتری در هنگام مواجه شدن با

- چرخش
- تغییر در عمق
- تغییر در مقدار روشنایی

دارد.

هم چنین در برخی تست ها که تعداد بیشتری از interestPoint ها را رسم و بررسی کردیم مشاهده کردیم که الگوریتم AKAZE علاوه بر بیشتر مقاوم بودن نسبت به موارد ذکر شده در تشخیص نقاط از **دقت بیشتری برخوردار** است و کمتر در اتصال **نقاط کلیدی مختلف به یکدیگر دچار خطا** می شود.

توضیحات نحوه کار کردن با کتاب خانه در کد به صورت کامنت شده آورده شده است.

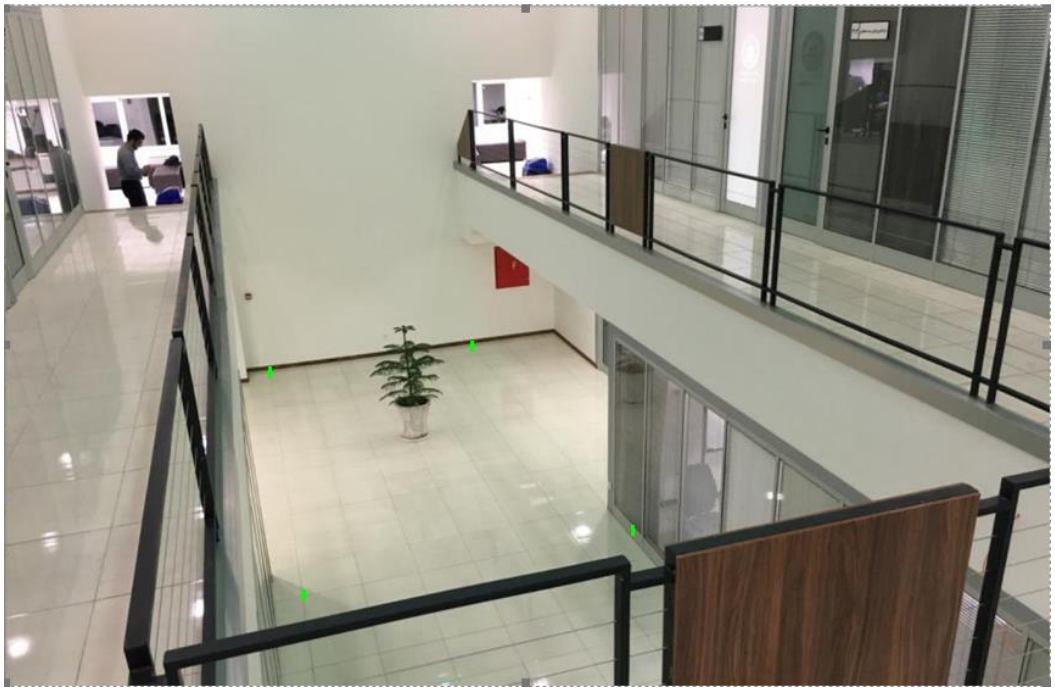
سوال دوم:

همانطورکه در صورت تمرین نیز اشاره شد ابتدا لازم است که چهار نقطه از تصویر با دید مایل انتخاب شود تا به کمک تبدیلات و ماتریس هموگرافی به دید در حالت قایم برسیم،ابتدا نقاط اولیه ای که در نظر گرفتیم را ذکر می کنیم و سپس به خروجی حالت قایم می پردازیم،در شکل زیر گوشه های مستطیل انتخابی را نشان داده ایم:

نقاط مستطیلی انتخابی در تصویر 1 ـ 1:

حال به خروجی دید در زاویه قایم می پردازیم:

نقاط مستطیلی انتخابی در تصویر 2-1:



حال به خروجی دید در زاویه قایم می پردازیم:

نتیجه گیری:

همانطور که مشاهده می کنید تصویر قایم از اجسامی که هندسه ساده ای دارند همانند "کاشی" به خوبی محاسبه شده است اما اجسامی که هندسیِ پیچیده ای دارند از قبیل گلدان،محاسبه تصویر قایم از آن ها با کیفیت بالا صورت نمی گیرد و دلیل آن هم مشخص است چرا که ما تصویرِ جدیدی نمیگیریم بلکه تعدادی نگاشت ریاضی که هر کدام محدودیت های فرایض خودشان را دارند را در تصویر اعمال می کنیم.

توضیحات نحوه کار کردن با کتاب خانه در کد به صورت کامنت شده آورده شده است.

سوال سوم:

نکات نحوه کار کردن با GUI:

گوشه های مستطیلی که انتخاب می کنید به صورت حالت های زیر باشد:

- بالا راست بالا چپ پایین راست پایین چپ و نیز حالت کاملا بالعکس.
- بالا راست پایین راست بالا چپ پایین چپ و نیز حالت کاملا بالعکس.

و الگوهای مشابه با حالت های ذکر شده.

نکته:می توانستیم با مقداری کدِ اضافه نحوه انتخاب نقاط را مستقل از این شرایط کنیم ولی به علت تنگی نرسیدم و از آنجا که در صورت تمرین نیز اشاره ای به کیفیت GUI نشد،بنده به کیفیت عملکرد فعلی بسنده کردم.

پس از انتخاب 4 نقطه،صفحه فعلی بسته می شود و صفحه ای باز می شود که خروجی مطلوب را دربردارد و برای خارج شدن کافیست حرفِ e را به نشانه exit وارد کنید.

نکات طراحی GUI:

1- تعیین listener فشار دادن کلیک موس
2- شمارش کلیک ها تا وارد شدن 4 نقطه(در هر مرحله مختصات ها استخراج می گشت و در اختیار ادامه برنامه قرار داده می شد).
3- پس از فشردن 4 کلید،صفحه فعلی بسته شده و صفحه ای جدید با پاسخ مطلوب باز می شود.

توضیحات نحوه کار کردن با کتاب خانه در کد به صورت کامنت شده آورده شده است.

در ادامه به بیان خروجی دو عکس می پردازیم.

عکس 1-2 و خروجی:(ترتیب انتخاب نقاط: بالا چپ،پایین چپ،پایین راست و بالا راست).

عکس 2-2 و خروجی:

ترتیب انتخاب نقاط: بالا راست_ بالا چپ_ پایین راست و پایین چپ.

سوال چهارم:

جزییات پیاده سازی این بخش(و نیز همه بخش ها) در کد به صورت کامنت شده آورده شده است و بنده در اینجا فقط به اشاره کلی پله پله مراحل انجامی می پردازم:

- Resize کردن تصاویر جهت دست یابی به کیفیت خروجی بیشتر.
- ابتدا نقاط کلیدی را استخراج می کنیم
- به علت زیاد بودن تعداد کلِ نقاط کلیدی استخراج شده یک threshold بر روی نقاط کلیدی اعمال می کنیم.
- با بررسی ترشهولد های مختلف حدی مناسب جهت concat تبدیل یافته تصاویر می یابیم.
- ماتریس هموگرافی و تبدیل لازم را استخراج و اعمال می کنیم.
- در نهایت عکس کنار هم قرار داده شده را از cropping که عملکردش در ادامه توضیح داده می شود عبور می دهیم.

عملکرد cropping:پس از کنار هم قرار دادن تصاویر داریم که کاملا در اسکیل هم نیستند و لذا کنار هم قرار دادن آن ها باعث ایجاد نقاط تیره زیاد حول قسمت هایی که برای هم اسکیل شدن دو تصویر لازم است می شود و ما به کمک این تابع خطوط افقی/عمودی تیره را از عکس خارج می کنیم.

```
# removing Black parts
def cropping(frame):

    # if "np.sum(frame[sth:] or frame[:sth])" is zero so this line
    # is completely Black so lets remove it.

    # top horizontal line
    if not np.sum(frame[0]):
        #removingBlackPart
        return cropping(frame[1:])

    # bottom horizontal line
    if not np.sum(frame[-1]):
        return cropping(frame[:-2])

    # left vertical line
    if not np.sum(frame[:,0]):
        return cropping(frame[:,1:])

    # right vertical line
    if not np.sum(frame[:,-1]):
        return cropping(frame[:,:-2])

    return frame
```

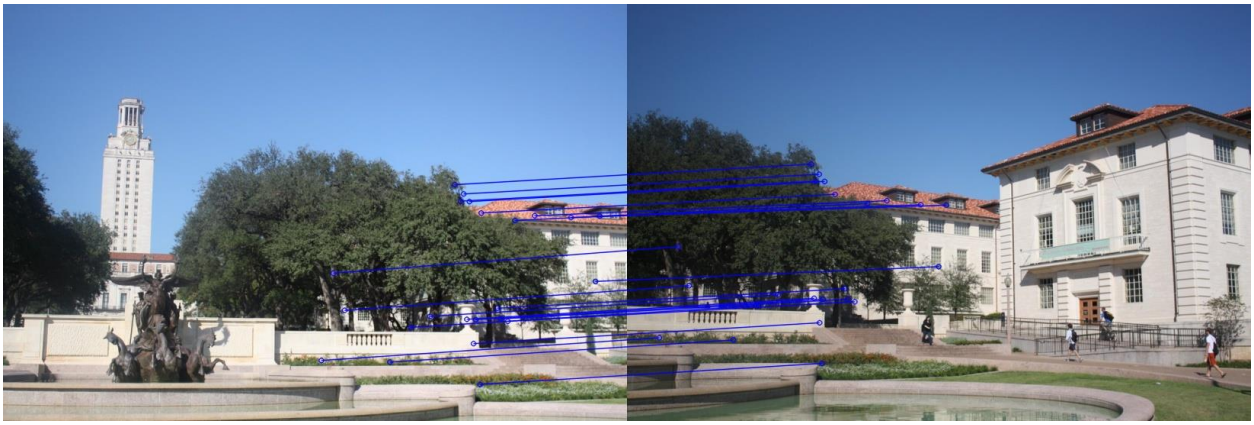
توضیحات نحوه کار کردن با کتاب خانه در کد به صورت کامنت شده آورده شده است.

حال در صفحه بعد خروجی هارا مشاهده می کنیم.

نقاط کلیدی تصویر سمت چپ(منظوری تصویری است که در خروجی پانوراما در سمت چپ قرار میگیرد)

نقاط کلیدی تصویر سمت راست(منظوری تصویری است که در خروجی پانوراما در سمت راست قرار میگیرد).

خروجی Matching:

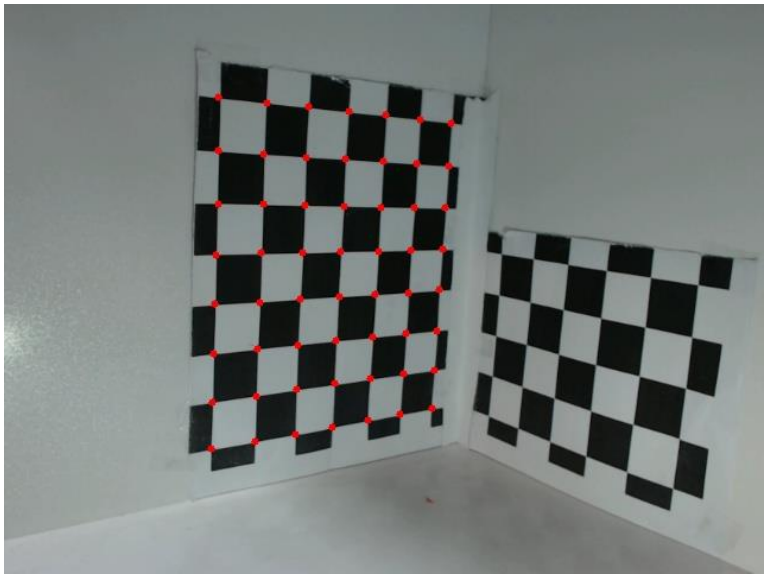


خروجی نهایی:

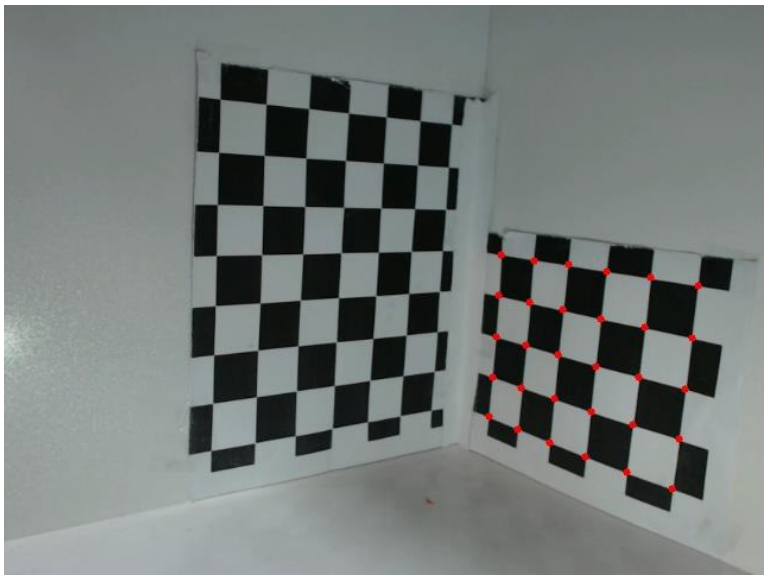
جهت تعمیق در مراحل انجامی به کامنت های موجود در کد مراجعه کنید.

سوال5:

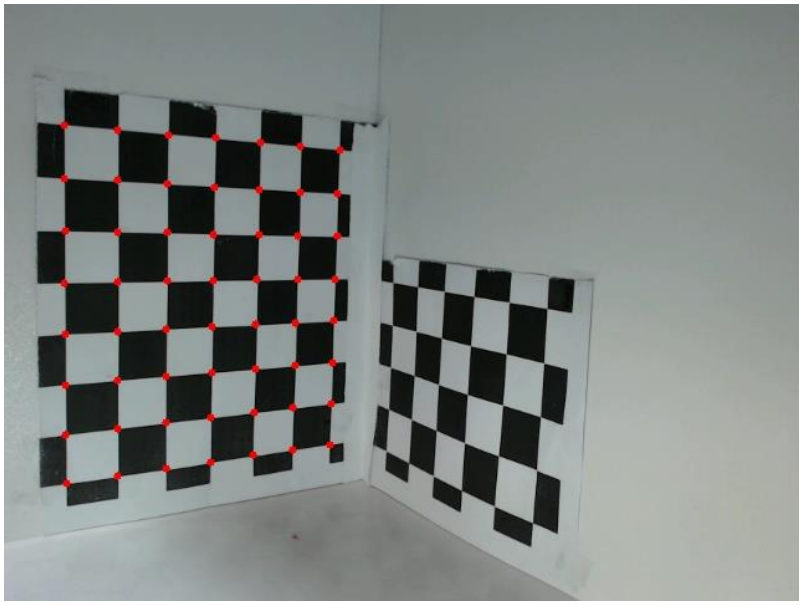
<u>زیربخش اول:</u> باید نقاط متناظر در دو تصویر را پیدا کنیم که اینکار را از روی **تصویر اموزشی** که در انها لیوان قرار **ندارد** انجام می دهیم برای این کار از تابع findChessboardCorners استفاده می کنیم که در ان برای پیدا کردن نقاط میانی هر صفحه شطرنج باید تعداد تغییر رنگ از سیاه به سفید در سطر و ستون را به ان داد و نقاط میانی صفحه شطرنج را دریافت کرد،این کار را برای هر 4 صفحه شطرنج(در هر تصویر train دو صفحه شطرنج قرار دارد) انجام می دهیم.خروجی findChessboardCorners برای هر صفحه شطرنج:



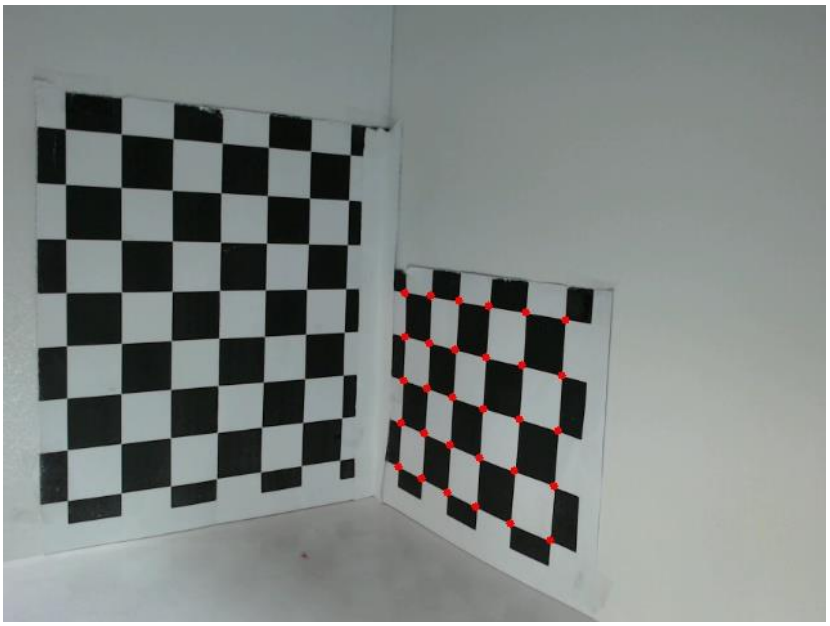
تصویر اول،صفحه شطرنج چپ



تصویر اول،صفحه شطرنج راست



تصویر دوم،صفحه شطرنج چپ



تصویر دوم،صفحه شطرنج راست

ادامه در صفحه بعد.

زیربخش دوم و سوم:

با تابع معرفی شده ماتریس فاندامنتال را پیدا کرده و ذخیره میکنیم حال در مرحله بعدی epipolar line متناظر به یک نقطه را باید پیدا کنیم ورودی های تابع computeCorrespondEpilines ابتدا نقطه مورد نظر سپس اینکه این نقطه درکدام تصویر قرار دارد و سپس ماتریس فاندامنتال میباشد و خروجی پارامتر های خط به صورت$ax+by+c =0$ را به ما می دهد،در نهایت به کمک تابع زیر(که از یکی از صفحات stackOverFlow استخراج شده است) به ترسیم خطِ خروجی می پردازیم:

```
def drawLines(img1,img2,lines,pts2):
    _,c,_ = img1.shape
    r = lines.reshape(3)
    color = (0,0,255)
    x0,y0 = map(int, [0, -r[2]/r[1] ])
    x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
    img1 = cv2.line(img1, (x0,y0), (x1,y1), color,1)
    img2 = cv2.circle(img2,tuple(pts2),5,color,-1)
    return img1,img2
```

ابتدا نقطه مدنظر را در تصویر رسم می کنیم و سپس خط epipolar را رسم می کنیم.

محل نقطهِ مدنظر:

زیربخش 4:

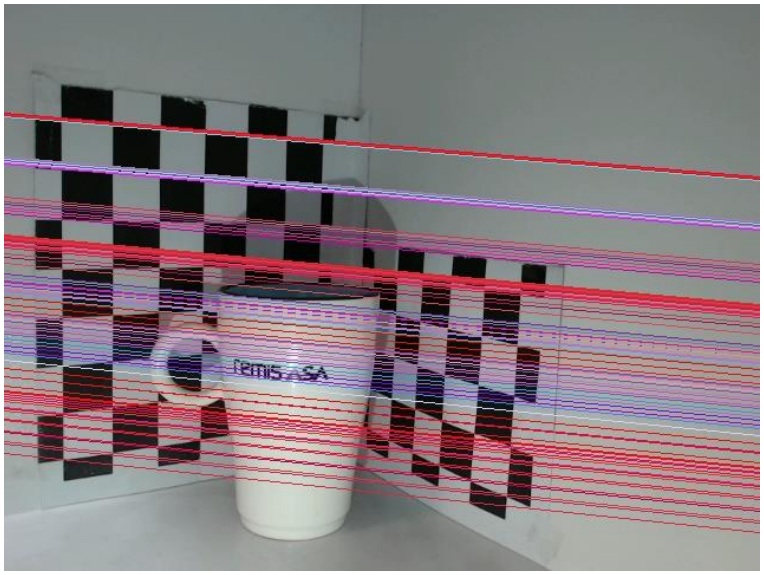
حال یکبار محل نقاط گوشه تصویر اول را در تصویر دوم و باردیگر نقاط گوشه تصویر دوم را در تصویر اول را رسم می کنیم و این کار به سادگی به طریق زیر انجام می شود:

- For روی نقاط کلیدی هر تصویر
- اجرا عملیات قسمت 2و3.

حالت اول: رسم خطوط epipolar نقاط گوشه تصویر 3-4 روی تصویر 4-4:

تصویر اول: نقاط کلیدی تصویر 4-3.jpg.

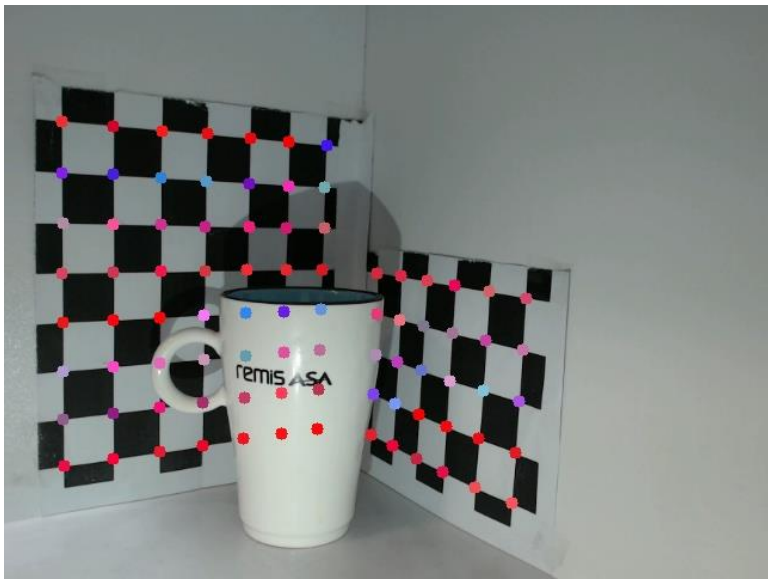
تصویر دوم: خطوط epipolar این نقاط روی تصویر 4-4.jpg است.

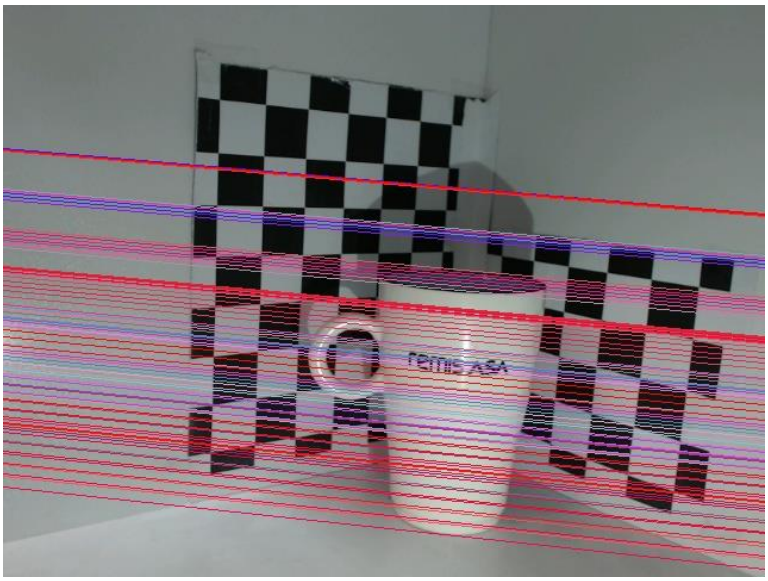


حالت دوم: رسم خطوط epipolar نقاط گوشه 4-4.jpg بر روی 4-3.jpg.

تصویر اول: نقاط کلیدی تصویر 4-4.jpg.

تصویر دوم: خطوط epipolar این نقاط روی تصویر 4-3.jpg است

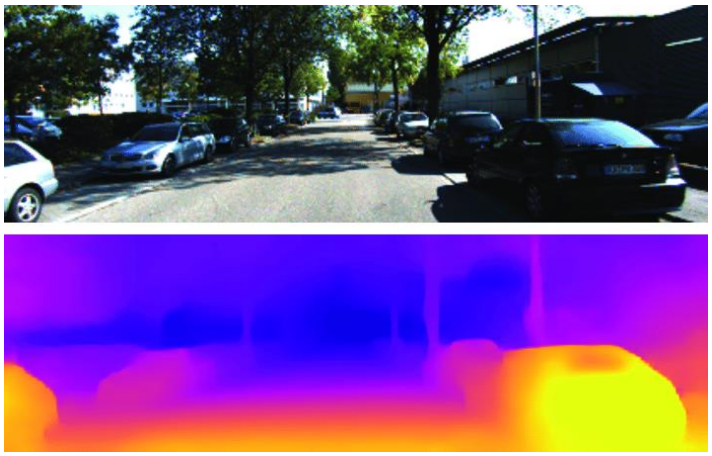




توضیحات نحوه کار کردن با کتاب خانه در کد به صورت کامنت شده آورده شده است.

پاسخ بخش پژوهشی:

Depth estimation یک کار در زمینه کامپیوتر ویژن است که طراحی شده است تا عمق یک تصویر دوبعدی را تخمین بزند این عملگر شامل این است که در ورودی یک تصویر RGB بگیرد و در خروجی تصویر عمق به ما بدهد تصویر عمق شامل اطلاعات فاصله از ViewPoint است.به عنوان مثال به تصویر زیر توجه کنید:



همانطور که ملاحظه می کنید هرچه در تصویر به داخل می رویم(ماشین های داخلی) رنگ آن ها تیره تر شده و یعنی در عمق بیشتری نسبت به نقاط دیگر قرار دارند و هم چنین ماشین ها و اشیایِ نزدیک رنگ روشن تری دارند که یعنی در عمق کمتری قرار گرفته اند.

برای انجام این کار از روش های مختلفی استفاده می شودکه ما برخی از این روش هارا به صورت اجمالی توضیح می دهیم:

در حالت کلی برای تخمین عمق در تصویر از روش های متعددی استفاده می شود که می توان آن هارا به صورت کلی به سه دسته نسبی تقسیم کرد:

- روش های دسته بندی بر مبنای هندسه
- روش های دسته بندی بر مبنای Sensor

به عنوان تلفیقی از دو روش فوق می توان به استخراج عمق به کمک stereo اشاره کرد: همانطور که از اسم این روش پیداست،ما محاسبات عمق را به کمک هندسه یِ تصویر انجام می دهیم و مشابه آنچه در درس راجع به استریو و تبدیلات خواندیم در اینجا نیز به عنوان مثال به کمک stereo و تبدیلات هندسی به تشخیص مکان نسبی اجسام نسبت به یکدیگر می پردازیم و در نهایت از روی این مکان نسبی به عمقِ اجزای تصویر می رسیم، سیستم بینایی استریو یکی از تکنیک های رایج بینایی ماشین است. ایده در اینجا استفاده مکان نسبی اجزا نسبت به هم است. یک صحنه واحد از دو زاویه دید مختلف ضبط می شود و عمق آن از اندازه گیری خطای اختلاف دو منظر برآورد می شود.

- روش های دسته بندی بر مبنای یادگیری ژرف:

دیتاست هایِ رایجی که در تخمینِ عمقِ عکس های تکی استفاده می شوند:

- KITTI
- NYU Depth
- Make3D CNN
- City Spaces

روش های محاسباتی متفاوتی در شبکه های یادگیری عمق استفاده می شوند که رایج ترین آن ها عبارت است از:

- MonoDepth
- LKVO Learner
- Semo Depth و ...

در این روش ها تلاش بر این است که همانند MaskRcnn ها اختلاف بین پیش بینی ها و ground truth ها مینیمم شود.

روش های دیگری نیز جهت محاسبه تخمیق عمق استفاده می شود از قبیل:

1ـ روش های مبتنی بر یادگیری adversarial: framework های متفاوتی در این زمینه وجود دارد که هرکدام دقت و کارایی خاص خودشان را دارند،در این روش ها یک شبکه Reinforcement طراحی می شود که ساختار های Local3D و global را تخمین می زنند.

2ـ روش های یادگیری دیگری وجود دارند که با تغییر معماری Loss! کار خود را انجام می دهند که برخی متریک های رایج آن طبق استناد مقاله عبارت است از:

- Loss عادی متد های supervised:

$$\mathscr{L}_2(d,d^*) = \frac{1}{N}\sum_i^N ||d-d^*||_2^2,$$

- Loss های تغییر معماری یافته:

$$\mathscr{L}(d,d^*) = \frac{1}{N}\sum_i^N y_i^2 - \frac{\lambda}{N^2}(\sum_i^N y_i)^2,$$

$$\mathscr{L}_s = \frac{1}{N}\sum_{i}^{n}[(\nabla_x D_i)^2 + (\nabla_y D_i)^2],$$

و

Berhu Loss:

$$\mathscr{L}_{Berhu}(d,d^*) = \begin{cases} |d-d^*| & \text{if } |d-d^*| \le c, \\ \frac{(d-d^*)^2+c^2}{2c} & \text{if } |d-d^*| > c, \end{cases}$$

لازم به ذکر است که در این چنین شبکه ها از متریک های متعددی جهت ارزیابیِ کیفیت و دقتِ شبکه های تخمین عمق استفاده می شود که برخی از آن ها عبارت اند از:

In order to evaluate and compare the performance of various depth estimation networks, a commonly accepted evaluation method is proposed in [35] with five evaluation indicators: **RMSE, RMSE log, Abs Rel, Sq Rel, Accuracies**. These indicators are formulated as:

- **RMSE** $= \sqrt{\frac{1}{|N|}\sum_{i\in N} \| d_i - d_i^* \|^2}$,
- **RMSE log** $= \sqrt{\frac{1}{|N|}\sum_{i\in N} \| \log(d_i) - \log(d_i^*) \|^2}$,
- **Abs Rel** $= \frac{1}{|N|}\sum_{i\in N} \frac{|d_i - d_i^*|}{d_i^*}$,
- **Sq Rel** $= \frac{1}{|N|}\sum_{i\in N} \frac{\|d_i - d_i^*\|^2}{d_i^*}$,
- **Accuracies:** % of d_i s.t. $\max(\frac{d_i}{d_i^*}, \frac{d_i^*}{d_i}) = \delta < thr$,