اصول پردازش تصویر

نیم‌سال دوم ۱۳۹۹-۱۴۰۰

مدرس: دکتر مصطفی کمالی تبریزی

دانشکده علوم ریاضی

تمرین سری اول

نام و نام‌خانوادگی: امین کشیری

شماره دانشجویی: ۹۷۱۰۱۰۲۶

توضیحات کلی

- عکس‌های ورودی در پوشه‌ی inputs/images قرار گرفته‌اند.
- خروجی تمام کدها، با همان اسم گفته شده در صورت سوال، در پوشه‌ی outputs/images قرار خواهد گرفت (اگر عکس دیگری نیز در این پوشه باشد، در توضیحات سوال مربوطه نوشته‌ام).
- بعضی از خط‌ها در کد، دارای comment هستند، که با uncomment کردن آن‌ها معمولاً می‌توانید عکس را در مراحل میانی ببینید (توضیحات بیشتر را در صورت لزوم، در توضیحات هر سوال داده‌ام).
- همراه فایل‌های یک فایل requirements.txt قرار دارد که محیط اجرای کدهای من است (در صورت نیاز).

سوال ۱

در خطوط اول ورودی خوانده می‌شود. سپس متغییر n مقدار دهی می‌شود که اندازه‌ی پنجره‌ای است که در روش هریس در نظر گرفته‌ام. من آن را برابر با $n = 9$ در نظر گرفته‌ام. ابتدا برای هر دو عکس نقاط کلیدی را پیدا می‌کنم. برای هر عکس، مشتق عکس را در دو راستای افقی و عمودی محاسبه می‌کنم (به کمک فیلتر سوبل). سپس مقادیر I_x^2 ، I_y^2 و I_xI_y را برای هر پیکسل محاسبه می‌کنم (البته نتایج را در ۳ ماتریس نگه داشته‌ام). سپس بزرگی گردایان را محاسبه می‌کنم و با اسم خواسته شده ذخیره می‌کنم. در چند خط بعدی، یک فیلتر گاوس روی I_x^2 ، I_y^2 و I_xI_y اعمال می‌کنم و نتیجه را به ترتیب در S_x^2 ، S_y^2 و S_xS_y ذخیره می‌کنم.

همانطور که می‌بینید، من kernel_size را برابر $(11, 11)$ قرار دادم. اما چون انحراف از معیار را صفر داده‌ام، opencv خود به صورت اتوماتیک انحراف از معیار را از روی سایز کرنل حساب می‌کند. فرمول محاسبه‌ی opencv نیز

$$\sigma = 0.3 * ((ksize - 1) * 0.5 - 1) + 0.8$$

است و این یعنی اگر $ksize = 11$ (سایز کرنل من) آنگاه داریم $\sigma = 2$. سایز کرنل را نیز به صورت تجربی به دست آوردم. حال می‌توانیم تنسور ساختار را به دست بیاوریم. با توجه به فرمول‌هایی که در کلاس یاد گرفتیم، دترمینان و اثر تنسور ساختار را نیز محاسبه کرده‌ام و آن‌ها را در det و tr ذخیره کرده‌ام. در خطوط بعدی نیز به کمک این دو مقدار، تابع هریس را برای تمام پیکسل‌ها محاسبه کرده‌ام و در R قرار دادم و عکس آن را ذخیره کرده‌ام.

مقدار k در تابع هریس را نیز به صورت تجربی محاسبه کردم: $k = 0.05$. در خط بعدی، یک threshold در نظر گرفتم، و مقادیر بیشتر از آن را ۱ قرار دادم، و بقیه را حذف کردم (صفر قرار دادم). این مقدار آستانه‌ی من نیز برابر بود با: $threshold = 11333444$. ماسک نهایی را در binary_mask قرار دادم و آن را ذخیره کردم.

حال با استفاده از الگوریتم non_maximum_supression نقطه‌های کلیدی را به دست می‌آورم:

- non_maximum_supression

این تابع ابتدا مولفه‌های همبندی یک تصویر باینری را به دست می‌آورد. سپس برای هر مولفه یک ماسک درست می‌کند، و روی هریس کل تصویر اعمال می‌کند. حال در عکس باقیمانده، پیکسل بیشینه را به دست می‌آورد. در نهایت به ازای هر مولفه همبندی، مختصات پیکسل بیشینه را برمی‌گرداند.

در ادامه، ممکن است بعضی از نقاط کلیدی نزدیک مرزها باشند و به همین دلیل نتوانیم برای آن‌ها توصیف‌گر مناسب تعریف کنیم. برای همین آن‌ها را حذف می‌کنم. تابع remove_border_points لیستی از نقاط می‌گیرد و آن‌هایی که از نصف سایز پنجره ما به مرزها نزدیک ترند را از این لیست حذف می‌کند. سپس نتیجه‌ای که به دست آوردم را ذخیره می‌کنم:

- save_detection_result

در این تابع، روی نقاط کلیدی حرکت می‌کنم و آن‌ها را با یک دایره روی تصویر مشخص می‌کنم و درنهایت نتیجه را ذخیره می‌کنم.

در حلقه‌ی بعدی، به ازای هر نقطه‌ی کلیدی، یک پنجره حول آن در نظر می‌گیرم، و مقادیر آن پنجره را در هر سه کانال در یک بردار می‌چینم. اندازه‌ی این بردار نیز $3 \cdot n^2$ است که n را در اول کد مقداردهی کرده بودیم. همانطور که گفتم من قرار دادم $n = 3$ اما دقت کنید که بردار من در هر ۳ کانال مقدار دارد و برای همین هم اندازه‌اش ۳ برابر بردار خواسته شده است. می‌توانستم اول عکس را سیاه و سفید کنم سپس یک پنجره با سایز n در نظر بگیرم، اما به نظرم در نظر گرفتن رنگ‌ها در این مسئله‌ی خاص بهتر بود.

در حلقه‌ی بعدی کارهای متناظر کردن نقطه‌ها را انجام می‌دهم. فاصله‌ی هر نقطه (در فضای ویژگی) از تصویر ۱ را با تمام نقاط تصویر دو محاسبه می‌کنم، دو نقطه‌ی از تصویر ۲ که کمترین فاصله را با این نقطه دارند پیدا می‌کنم، اگر فاصله‌ی نقطه‌ی دوم (تشخیص دورتر) نسبت به نقطه اول (تشخیص نزدیکتر) از حد مشخصی بیشتر بود، نقطه‌ی نزدیک تر را به عنوان یک تناظر تشخیص می‌دهیم. این حد مشخص (مقدار آستانه) نیز در متغییر threshold قرار گرفته است. من میزان آن را برابر با $threshold = 1.8$ قرار دادم. سپس یک بار دیگر همینکارها رو از تصویر دوم به اول انجام می‌دهیم. در نهایت، اگر دو نقطه در هردو تکرار به هم متناظر شده بودند، آن‌ها را به عنوان یک تناظر نهایی در نظر می‌گیریم. در واقع من برای تناظرها از یک به دو و از دو به یک یک ماتریس ساختم، و اگر تناظر تشخیص دادم در درایه‌ی مورد نظر ۱ قرار دادم، و در نهایت این دو ماتریس را در هم ضرب کردم. در خانه‌هایی که الان ۱ قرار گرفته باشد، من یک تناظر نهایی تشخیص داده‌ام.

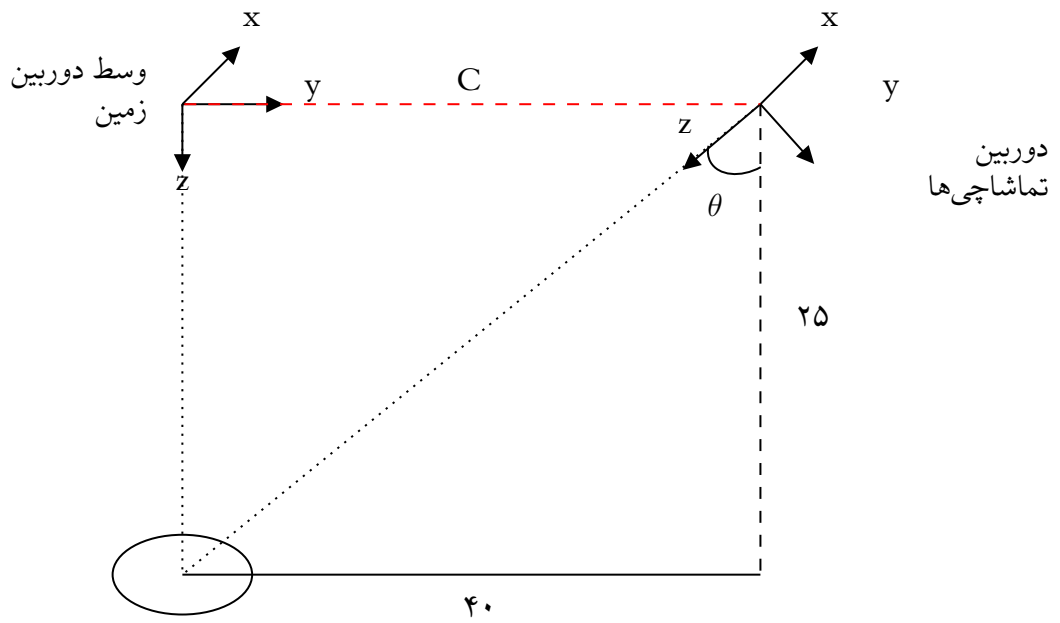
سپس، ابتدا نقاط کلیدی را در دو عکس نشان دادم و ذخیره کردم، بعد هم در یک حلقه، به ازای هر دو نقطه‌ی متناظر شده، یک خط بین آن‌ها رسم کرده‌ام. در نهایت نیز تصویر را ذخیره کردم.

با این که به مقادیر خواسته شده در کد اشاره کردم، برای راحتی یک بار دیگر آن‌ها را ذکر می‌کنم:

$n = 11$
$k = 0.05$
$threshold1 = 11333444$
$\sigma = 2$
$threshold2 = 1.8$

سوال ۲

دو تا دوربین در نظر بگیرید. دوربین وسط زمین (۱)، و دوربین تماشاچی‌ها (۲). اگر محورهای دو دوربین شبیه شکل پایین باشد، این دو دوربین تنها به اندازه‌ی زاویه θ حول محور x چرخیده‌اند، و به اندازه‌ی C نیزجابجا شده‌اند.



حال چون این دو دوربین از یک صفحه‌ی یکسان عکس می‌گیرند، طبق توضیحات کلاس می‌توانیم تصویر دیده شده در این دو دوربین را با رابطه‌ی زیر به هم مرتبط کنیم:

$x' = Hx$
$t = -RC$
$H = K'(R - \frac{tn^t}{d})K^{-1}$

که متغییرهای بدون پریم، نماینده‌ی دوربین دوم هستند، اگر دوربین اول را متخصات جهانی بگیریم. حال اگر دوربین وسط زمین را مختصات جهانی بگیریم، می‌توانیم یک هموگرافی به دست بیاوریم که تصویری که دوربین وسط زمین می‌بیند را به تصویر تماشاچی‌ها تبدیل کند. برعکس، با وارونه کردن این هموگرافی، می‌توانیم بفهمیم چه تصویری دوربین وسط زمین می‌تواند ببیند، اگر دوربین تماشاچی‌ها یک تصویر خاص را ببیند. پس کاری که انجام می‌دهیم این است که از فرمول‌های بالا H را به دست می‌آوریم، سپس با اعمال وارون این ماتریس روی تصویری که داده شده (که از دیدگاه تماشاچی‌ها است) تصویری که باید وسط زمین دیده شود را به دست می‌آوریم.

p_x و p_y را برای دوربین تماشاچی‌ها می‌توانیم برابر نصف ابعاد عکس در نظر بگیریم. برای تصویر دوم نیز من آن‌ها را برابر نصف ابعاد عکس گرفتم، اما ابعاد عکس نهایی را ۲۰۰۰ در ۲۰۰۰ در نظر گرفتم، تا کل نتیجه در آن قرار بگیرد. با توجه به شکل بالا، پارامترهای استفاده شده به صورت زیر است (دقت کنید که ماتریس دوران من فقط حول محور ایکس است):

$C = [0, 40, 0]$
$n = [0, 0, 1]$
$d = -25$
$\theta = -tg^{-1}(\frac{40}{25})$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix}$$

$$K = \begin{bmatrix} 500 & 0 & 1000 \\ 0 & 500 & 1000 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K' = \begin{bmatrix} 500 & 0 & 128 \\ 0 & 500 & 128 \\ 0 & 0 & 1 \end{bmatrix}$$

توجه: من θ را منفی زاویه نشان داده‌شده در شکل نشان دادم، زیرا دوران θ درجه محورهای مختصات در جهت پادساعتگرد، معادل با دوران مختصات نقاط به اندازه θ درجه در جهت پادساعتگرد است، یا به عبارت ساده‌تر، باید θ را منفی کرد.

توجه: من از سوال برداشت کردم که فاصله‌ی افقی بین دو دوربین ۴۰ است. یعنی فاصله‌ی دوربین کنار زمین تا وسط زمین، روی زمین ۴۰ است. دو صورتی که منظور فاصله‌ی واقعی بوده است، کافی است تنها خط ۲۶ کد من را uncomment کنید.

توضیح کد: کد تنها مقادیری که در بالا ذکر کردم را مقداردهی می‌کند. سپس ماتریس‌ها را در هم ضرب می‌کنم تا H را به دست آورم، در نهایت آن را وارونه می‌کنم، تصویری که داریم را نیز با آن warp می‌کنم، و تصویر را ذخیره می‌کنم.

سوال ۳

در خط‌های اولیه عکس‌ها خوانده می‌شوند، و سپس با کمک SIFT ، نقاط کلیدی و توصیف‌گرها را به دست می‌آوریم. در خطوط بعدی توابعی تعریف کردم که توضیح خواهم داد. بعد از تعریف توابع، ابتدا نقاط کلیدی را به کمک تابع زیر رسم کرده‌ام:

- draw_keypoints

  دو عکس خالی temp۱ و temp۲ را با اندازه‌های مورد نیاز می‌سازم. سپس به کمک تابع drawKeypoints نقاط کلیدی را می‌کشم و در این دو عکی خالی می‌گذاریم. چون ابعاد دو عکس یکسان نیست، ارتفاع آن‌ها را با هم یکی می‌کنم و به صورت افقی کنار هم قرار می‌دهم و سپس عکس را ذخیره می‌کنم.

حال با کمک تابع get_matches نقاط متناظر را به دست می‌آورم:

- get_matches

  ابتدا با کمک تابع knnMatch از کلاس BFMatcher برای هر نقطه‌ی کلیدی، ۲ تا بهترین متناظر را پیدا می‌کنم، و سپس اگر اختلاف فاصله‌ی (فاصله در فضای ویژگی) این دو نقطه تا نقطه‌ی ما از حد مشخصی بیشتر بود، نقطه‌ی نزدیکتر را به عنوان نقطه‌ی متناظر در نظر می‌گیرم.

در خط بعدی، نقاط کلیدی که با هم متناظر شده‌اند را از بقیه نقاط کلیدی جدا می‌کنم:

- get_matched_keypoints_and_coords

  در این تابع با داشتن اندیس نقاط متناظر، آن‌ها را از تمام نقاط کلیدی جدا می‌کنم. در این تابع علاوه بر گرفتن خود نقاط ویژگی (که یک کلاس پایتون است) مختصات آن‌ها رو نیز به صورت جدا گانه در دو آرایه می‌ریزم و برمی‌گردانم که برای راحتی در مراحل بعدی استفاده می‌شود.

حال نقاط متناظر را با رنگ دیگری روی عکس‌ها می‌کشم:

- draw_correspondence

  این تابع بسیار شبیه به تابع draw_keypoints است. تنها کار اضافه‌ای که انجام می‌دهد این است که علاوه بر کشیدن تمام نقاط کلیدی، تمام نقاط کلیدی‌ای که با هم متناظر شده‌اند را نیز با رنگ آبی می‌کشد و سپس عکس را ذخیره می‌کند.

سپس نقاط متناظر را به هم وصل می‌کنیم و رسم می‌کنیم:

- draw_matched_keypoints

  این تابع با استفاده از تابع آماده‌ی drawMatches نقاط متناظر را به هم وصل می‌کند و رسم می‌کند. سپس نتیجه را ذخیره می‌کنم.

حال برای این که بتوانیم نتیجه را بهتر ببینم تنها ۲۰ تا از نقاط متناظر را نمایش می‌دهیم:

- draw_matched_keypoints_sample

  این تابع دقیقا مانند تابع قبلی است، اما تنها ۲۰ تا از نقاط متناظر را می‌کشد. اگر ۲۰ نقطه‌ی تصادفی انتخاب کنیم خیلی نمی‌شود نتیجه‌ی کار را به خوبی دید. برای این که بهتر نتیجه را ببینیم، ۲۰ تا از بهترین تناظرها را انتخاب می‌کنم، و آن‌ها را در تصویر res16_best نمایش می‌دهم.

در خط بعدی، ماتریس هموگرافی را به کمک روش RANSAC بدست می‌آوریم. تعداد تکرارهای این روش برای تصویر من ۸۰۰ است. ماتریس هموگرافی به دست آمده نیز برابر است با:

$$H = \begin{bmatrix} 4.4 \cdot 10^{0} & 3.8 \cdot 10^{-1} & -2.9 \cdot 10^{3} \\ 9.3 \cdot 10^{-2} & 2.9 \cdot 10^{0} & -1.4 \cdot 10^{3} \\ 2.4 \cdot 10^{-4} & -1.0 \cdot 10^{-5} & 1.0 \cdot 10^{0} \end{bmatrix}$$

در خط بعدی (وتابع بعدی) دوباره نقاط متناظر را به هم وصل می‌کنیم و نمایش می‌دهیم، اما این‌بار inlier ها را با قرمز می‌کشیم:

- draw_matched_keypoints_and_inliers

این تابع بسیار شبیه به draw_matched_keypoints است، منتهی پس از این که کل نقاط متناظر را کشیدیم، عکس نتیجه را دوباره جدا می‌کنیم، این بار این دو عکس را به همان تابع drawMatches می‌دهیم، اما این بار به آن ماسک inlier ها را نیز می‌دهیم تا تنها آن‌ها را بکشد. نتیجه را نیز ذخیره می‌کنیم.

تمام نقاط inlier من به درستی متناظر شده بودند. بنابرین من تصویر ۱۸ را ندارم. اما چون در تصویر ۱۷ این‌ها به خوبی دیده نمی‌شدند، در تصویر res_INLIERS تنها inlier ها را نشان دادم.

من ماتریس هموگرافی را روی عکس اعمال کردم، و در res_19 نشان دادم. اما چون در این حالت کل عکس دیده نمی‌شد، ماتریس انتقال T را با ابعاد مناسب ساختم، و نتیجه را علاوه بر تبدیل با ماتریس هموگرافی، انتقال نیز داده‌ام. این نتیجه را در تصویر res_19_2 ذخیره کرده‌ام.

سوال ۴

کد این سوال دقیقا مانند سوال قبلی است، و تنها به آن یک تابع اضافه شده است (و توابع اضافی که برای ذخیره کردن عکس‌ها بوده‌اند حذف شده‌اند). این تابع نیز تنها کاری که انجام می‌دهد این است که به کمک روش RANSAC ماتریس هموگرافی را محاسبه می‌کند. ادامه‌ی کار دقیقا مانند قسمت قبلی است. پس تنها این تابع را توضیح می‌دهم:

- findHomography

این تابع یک بخش اصلی دارد، و علاوه بر آن از ۳ تابع کمکی استفاده می‌کند. در خط‌های اولیه متغییرهای مورد نیاز مقداردهی شده‌اند. تابع به عنوان ورودی یک مقدار maxIters می‌گیرد که تعداد تلاش‌های مورد نیاز RANSAC را اول کار برابر با آن قرار می‌دهیم. w احتمال inlier بودن است که اول کار ۰ است. p نیز احتمالی است که حداقل یکی از sample ها outlier نداشته باشد که آن را برابر ۹۹ درصد قرار می‌دهم. سپس در یک حلقه شروع به اجرای RANSAC می‌کنم. این حلقه تا زمانی ادامه پیدا می‌کند که یا به maxIters رسیده باشیم یا به N که تعداد تلاش‌هایی مورد نیاز RANSAC است. یعنی اگر این تعداد بار تلاش کنیم، با احتمال p حداقل یکی از sample ها outlier ندارد.

در حلقه به تصادف ۴ تا از نقاط متناظر را انتخاب می‌کنیم. سپس با کمک تابع findHomographyUtil هموگرافی متناظر با این ۴ نقطه را پیدا می‌کنیم. سپس به کمک تابع بعدی تعداد inlier ها با استفاده از این هموگرافی را پیدا می‌کنیم. هرگاه تعداد inlier هایی که با این ماتریس داریم از بیشترینی که تا الان پیدا کرده‌ایم بیشتر شد، آن ماتریس را به عنوان بهترین ماتریس تا الان نگه می‌داریم و همچنین تعداد inlier ها با این ماتریس را نیز در max_inlier نگه می‌دارم. سپس w را نیز با تقسیم کردن تعداد inlier ها به کل نقاط متناظر به دست می‌آورم و N را نیز با فرمول توضیح داده شده در کلاس آپدیت می‌کنم (چون می‌خواستم که حلقه حتما تا N اجرا شود، maxIters را یک عدد خیلی بزرگ قرار دادم که به خاطر رسیدن به آن متوقف نشویم. با این شرایط، حلقه‌ی من نزدیک به ۲۲۹ هزار بار اجرا می‌شود). در نهایت پس از پایان حلقه، به کمک بهترین هموگرافی پیدا شده، تعداد inlier ها و اطلاعاتشان را به دست می‌آورم. سپس به کمک تمام این نقاط، ماتریس هموگرافی نهایی را پیدا می‌کنم و برمی‌گردانم. البته ماتریس را بر عدد درایه $(3, 3)$ اش تقسیم می‌کنم تا اعداد آن شبیه به اعداد ماتریس هموگرافی سوال قبلی شوند. دقت کنید که ضرب یا تقسیم همه‌ی اعداد یک ماتریس هموگرافی بر یک عدد ثابت، آن را تغییر نمی‌دهد (چون ۸ درجه‌ی آزادی دارد).

توضیح توابع کمکی:

- count_inliers

این تابع یک ماتریس هموگرافی ورودی می‌گیرد. سپس در خط‌های بعدی به ازای هر نقطه از تصویر دوم (تصویر مبدا) مختصات آن (در فضای projective) را در این هموگرافی ضرب می‌کند (از راست)، سپس مختصه اول و دوم بردار حاصل را بر مختصه سوم تقسیم می‌کند تا مختصات تبدیل یافته‌ی آن را پیدا کند. سپس با مقایسه‌ی آن‌ها با مختصات متناظر در تصویر اول هر کدام از نقاط تصویر دوم، آن‌هایی که به مختصات متناظر در تصویر اول از حدی (که توسط threshold مشخص می‌شود) نزدیک تر هستند را به عنوان inlier تشخیص می‌دهد و آن‌ها را به همراه تعدادشان خروجی می‌دهد. من مقدار threshold را نیز در کد خود برابر با ۲ قرار دادم.

- findHomographyUtil

این تابع به کمک ۴ نقطه‌ی متناظر، ماتریس A را که در کلاس توضیح داده شده است می‌سازد، و سپس به کمک تجزیه SVD یک جواب برای دستگاه $Ah = 0$ را پیدا می‌کند. سپس با تغییر ابعاد h ماتریس هموگرافی مورد نظر را خروجی می‌دهد.

- findHomographyUtil_with_inliers

این تابع دقیقا مانند تابع بالا کار می‌کند با این تفاوت که به جای ۴ نقطه از تمام نقاط inlier استفاده می‌کند. روش کار دقیقا مانند تابع قبلی است. با این کار بهترین هموگرافی ممکن توسط نقاط inlier به دست می‌آید.

توجه کنید که در این سوال، مانند سوال قبلی برای این که نتیجه به خوبی دیده شود دو عکس res20 و res20_2 را ذخیره کرده‌ام که فرق دومی فقط این است که تصویر جابجا شده است تا کل آن دیده شود. همچنین ماتریس هموگرافی به دست آمده ازین قسمت، به ماتریس سوال قبلی شباهت‌های زیادی دارد اما دقیقا با آن یکی نیست:

$$H = \begin{bmatrix} 3.8 \cdot 10^{0} & 3.7 \cdot 10^{-1} & -2.5 \cdot 10^{3} \\ 4.9 \cdot 10^{-2} & 2.3 \cdot 10^{0} & -1.1 \cdot 10^{3} \\ 1.2 \cdot 10^{-4} & -1.5 \cdot 10^{-4} & 1.0 \cdot 10^{0} \end{bmatrix}$$