سید محمد طاها طباطبایی – تمرین سری هفتم

9812762838

چکیده:

توضیحات فنی:

* توضیح جزییات پیاده‌سازی توابع در محل پیاده‌سازی کد، به صورت کامنت و داکیومنت نوشته شده است.

7.1.1

7.2.1

برای پیاده‌سازی این بخش، از الگوریتم sift استفاده کرده ایم. در سلول اول، یک آبجکت از کلاس sift ساخته می‌شود. در سلول بعد، با استفاده از تابع detectAndCompute ، نقاط کلیدی و بردار های دیسکریپشن تصویر sl را محاسبه می‌کنیم. در نهایت با استفاده از تابع drawKeyPoints ، نقاط کلیدی به‌دست آمده را روی تصویر نمایش می‌دهیم. در دو سلول دیگر، این عملیات را برای تصاویر sm و sr نیز انجام دادم. استفاده از تصویر grayscale، برای پیدا کردن نقاط کلیدی، به این دلیل است که طبق تجربه، خروجی نهایی بهتری حاصل شد.

در فاز دوم، با استفاده از تابع BruteForceMatcher ، نقاط توجه(interest points) متناظر بین دو تصویر را با کمک بردار های دیسکریپشن دو تصویر محاسبه می‌کنیم. برای پیدا کردن نقاط، 2 نقطه با بیشترین تشابه از تصویر test ، نسبت به تصویر src را با تابع knnMatch پیدا می‌کنیم(برای محاسبه نسبت lowe در مرحله بعد). در ادامه یک حلقه وجود دارد که طبق روش پیشنهادی lowe، یک ترشولد اعمال می‌کند تا برخی نقاط که همچنان فاصله زیادی از هم دارند(تشابه کمی دارند) را نادیده بگیرد. با اینکار نسبتا مطمئن می‌شویم، نقاط توجه پیدا شده بین دو تصویر، نقاط با کیفیت بهتری هستند.

سپس، درصد مشابهت بین دو تصویر را با کمک تابع howSimilar محاسبه کرده ایم. همانطور که مشخص است، درصد تشابه بین تصاویر sl و sm و درصد تشابه بین تصاویر sm و sr ، نسبت به درصد تشابه sl و sr ، تقریبا دو برابر است. دلیل این اتفاق تغییر زیاد زاویه دید بین دو تصویر sl و sr است. البته ممکن است به نظر بیاید، درصد مشابهت تصاویر به طور کلی کم است، اما دلیل این اتفاق، ترشولدی است که در مرحله قبل اعمال کردیم. با توجه به اینکه بسیاری از نقاط توجه پیدا شده ضعیف، در مرحله قبل کنار گذاشته شده اند، اگر ترشولد را آسان تر تعیین کنیم(عدد بالاتر) تعداد نقاط انتخاب شده بیشتر می‌شود، و دو تصویر از لحاظ آماری شبیه تر محسوب می‌شوند. این انتخاب ترشولد نسبی است. درصد مشابهت حدود 30 برای دو تصویر که تغییر زاویه دید دارند، همچنان به نظر عدد مناسبی است. برای تست، ترشولد 0.99 بر روی مقایسه sm و sr تست شد که نتیجه مشابهت 96.8 حاصل شد، یعنی اگر تقریبا تمام نقاط توجه پیدا شده را در نظر بگیریم، تشابه مورد انتظار نزدیک به 100 خواهد بود، هرچند طبیعتا 100 نخواهد بود.

در انتهای این بخش، با فراخوانی تابع ransac، ماتریس تبدیل هموگرافی H را استخراج می‌کنیم. Random sample consensus یا به اختصار RANSAC، در واقع یک الگوریتم پیدا کردن نقاط نویز است. مزیت استفاده و تعمیم الگوریتم RANSAC برای یافتن H(ماتریس هوموگرافی)، این است که با اجرای به دفعات زیاد مراحل، سعی می‌کند، بهترین H ممکن را با در نظر گرفتن زیرمجموعه های مختلف، از نقاط مچ شده دو تصویر، برای یافتن تبدیل هموگرافیک متناظر بین دو تصویر است.

در این تابع دو اتفاق اصلی رخ می‌دهد. 1) پیدا کردن یک ماتریس H 2) جایگزین کردن ماتریس H جدید در صورت بهتر بودن نسبت به مراحل قبلی

در گام اول، الگوریتم 4 نقطه مچ شده از دو تصویر را توسط تابع random\_point انتخاب می‌کند. با کمک این 4 جفت نقطه، توسط تابع homography، ماتریس H را می‌سازیم. از جایی که نقاط ورودی به صورت رندوم انتخاب شده اند، ممکن است بهترین 4 نقطه ممکن برای به‌دست آوردن تبدیل هموگرافی دو تصویر نباشند، برای همین الگوریتم در تعداد دفعات بالا تکرار می‌شود، تا بهترین حالت ممکن انتخاب شود. همچنین رندوم بودن انتخاب، باعث می‌شود تا نتایج هر بار اجرای مراحل این اسکریپت از ابتدا، کمی متفاوت باشد.

در گام دوم، از تابع get\_error برای سنجش کیفیت ماتریس به‌دست آمده استفاده می‌کنیم. این تابع با کمک ماتریس حاصل، یک سری نقاط از تصویر دیگر را از روی نقاط تصویر اصلی، بازسازی می‌کند، و حاصل اختلاف آنها را محاسبه می‌کند. سپس بر اساس یک ترشولد، اندیس نقاطی که اختلافشان از ترشولد کمتر باشد انتخاب می‌شوند. در نهایت براساس تعداد این نقاط کم اختلاف( هر چه تعداد این نقاط کمتر، بهتر) به این شکل که اگر یک H پیدا شود که تعداد نقاط بیشتری پیدا کند، پس بهتر است، H جدید را به عنوان بهترین H پیدا شده تا این گام از الگوریتم انتخاب می‌کند. در انتهای اتمام حلقه، بهترین H ممکن انتخاب شده است.

در آخرین گام این بخش، تابع stitch برای میکس کردن دو تصویر استفاده می‌شود. این تابع با کمک ماتریس H، اقدام به یافتن تبدیل مناسبی از نقاط تصویر سمت راست در تصویر سمت چپ است. میانگین این نقاط را جایگزین نقاط مورد نظر در تصویر سمت چپ می‌کند.

در فاز سوم، از حاصل دوخته شده دو تصویر sm و sr ، و دوختن آن با تصویر sl، یک تصویر کلی به‌دست می‌آوریم. مراحل و توابع مورد استفاده دقیقا مطابق مراحل قبلی است.

دومین روش محاسبه ، استفاده از ORB است. کد های این بخش، دقیقا مشابه با بخش SIFT است. فقط کافی است، از تابع سازنده مدل ORB به جای SIFT استفاده شود. پیاده‌سازی توابع را ظوری انجام دادیم تا با هر دو مدل سازگار باشد.

از مقایسه تصاویر حاصل از SIFT و ORB، می‌توان متوجه شد که روش SIFT، بهتر عمل ‌می‌کند، هر چند که ORB سریعتر عمل ‌می‌کند، اما کیفیت فیچر های استخراج شده توسط SIFT بهتر است. به طور مثال در SIFT ، در آخرین مرحله دوختن 3 تصویر، تعداد مچ های با کیفیت به شرح زیر است:



در حالی که در مرحله آخر ORB داریم:



همانطور که مشخص است، در مرحله آخر دوختن، در الگوریتم SIFT تعداد مچ های مناسب، از کل مچ های باقی مانده در روش ORB بهتر است.

7.2.2