**דוח סופי- פנורמה**

בס"ד

Shalom weinberg

neil michaeli

Tal siman tov

Github: <https://github.com/sholmiw/-myPanorama>

**הקדמה:**

* הבעיה:
* תמונת מצלמה קולטת חלק מצומצם משדה הראיה של בני אדם. אך נרצה לקבל תמונה המצליחה לקלוט שדה ראיה רחב.
* המטרה:
* נרצה בעזרת אוסף של תמונות הנקלטות באופן מסודר מצד לצד מאותה מצלמה ליצר תמונה אחת המאחדת את כל התמונות
* הקשיים:
* המצלמה זזה קלות בין תמונה לתמונה ולא נעה בצורה מושלמת, אנו רוצים לבצע חיבור של התמונות ללא מגע אדם דבר המאלץ: למצוא קשרים כין תמונות, לחשב את הסטייה והכיוון שהתמונות זזות ולחבר את התמונות לכדי תמונה אחת.
* האלגוריתם:

1. עבור כול 2 תמונות מחלצים תכונות (פיצ'רים) ע"י אלגוריתם ORB
2. מחשבים את ההומוגרפיה בעזרת הנקודות משתי התמונות. עושים זאת ע"י אלגוריתם RANSAC:

* דוגמים כול פעם 4 נקודות מתאימות
* מחשבים הומוגרפיה
* מחשבים inliersכאשר ||pi ’ -H pi || < ε
* לבסוף לוקחים את קבוצת inliers הכי גדולה
* מחשבים מחדש את ה inliers ע"פ ההומוגרפיה הכי גדולה.

1. מציגים את הפיצ'רים המתאימים על גבי 2 התמונות זו לצד זו.
2. לאחר מכן מבצעים Warping לתוך משטח שגודלו נקבע לפי ה x,y המינימליים והמקסימליים של פינות התמונות שהתקבלו לאחר הפעלת ההומוגרפיה עליהן.
3. לבסוף, מחזירים את התמונה הסופית- הפנורמה

**גישות ושיטות:**

* כאשר ניסינו לצייר את ההתאמות עבור תמונות שאנחנו דגמנו, לפעמים היינו צריכים להקטין את מימדי התמונות כדי לקבל פלט בגודל סביר.
* עבדנו עם תמונות בגווני אפור לחסוך סיבוכיות קוד
* מימשנו את מציאת הפיצ'רים ע"י orb שנתן לנו תוצאה טובה יותר מאשר sift או SURF

Orb מקבל מספר זהה של נקודות מפתח גם בתמונה המקורית וגם כאשר מסובבים אותה (לעומת השניים האחרים שמקבלים מספרי שונים בכול תמונה) והביצועים שלו מהירים יותר משל השניים האחרים. בנוסף ORB חינמי ולכן לא גורר התאמות נוספות לתוכנה כמו SIFT

* איך ORB עובד?

ORB משתמש ב FAST למצוא נקודות מפתח ואז מכיל את Harris corner measure כדי למצוא n נקודות מובילות מתוכם. הוא משתמש גם בפירמידה כדי לייצר multiscale-features

מה הן ההגבלות? FAST לא מחשב את הכיוון

<https://medium.com/@shehan.a.perera/a-comparison-of-sift-surf-and-orb-333d64bcaaea>

<https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html>

* לאחר שהתמשנו בORB בדקנו מספר אופציות לסנן אותן: פעם השתמשנו בBF של אופן סיבי, עשינו גם בעזרת חישוב של SSD בין ההפרשים ובסוף השתמשנו בנרמול של ההפרש.
* לאחר סינון הפיצרים הפעלנו את פונק הransac
* בפונק ransac השתמשנו בהמרה של נקודה לנקודה הומוגרפית ומשם הפעלה של ההומוגרפיה על הנקודה.
* לאחר שמקבלים מטריצת הומוגרפיה והתאמות סופיות יצרנו פונקציה המראה את ההתאמות.
* עדכנו את מטריצת ההומוגרפיה לפי השכן שלה (במידה ויש יותר מ2 תמונות)
* שלחנו את התמונות עם הצבע ביחד עם ההומוגרפיות החדשות ויצרנו תמונה סופית.

**תוצאות:**

* תיאור הניסויים שעשינו: ניסינו לבצע את מציאת ההתאמות ע"י sift וע"י ORB ובחרנו בסוף להשתמש בORB מכיוון שהוא חינמי ומהיר.
* כאשר ניסינו לצייר את ההתאמות עבור תמונות שאנחנו דגמנו שהיו גדולות מידי היה לנו בעיות ולכן בחרנו להקטין את התמונות.
* פרמטרים באלגוריתם ה RANSAC :

1000 הרצות ,ו 1 לסף .

ניסינו גם עוד משתנים אבל 1 לסף נותן תוצאות מאוד טובות.

בעיקרון אנחנו רוצים סף נמוך ואנו מעוניינים שכמות ההרצות תהיה כמה שיותר גבוהה (בזמן סביר) כי כך זה יותר "מדויק". היות ולמדנו שאם אנחנו עושים 1000 הרצות אז גם אם יש כמעט 50% דאטה לא תקין אנחנו נצליח להגיע להצלחה של 99% .

* באיזה מדדים השתמשנו כדי למדוד הצלחה ולמה?

באמצעות כמות ההרצות וסף בransac ניתן למדוד הצלחה של המודל (במקרה שלנו הומוגרפיה)

את התמונה הסופית בחנו לפי הראיה- האם קיבלנו תמונה אחת שאכן נראית כחיבור של כמה תמונות לתמונה אחת

מספר דוגמאות:













