ב. תיאור קצר של האלגוריתם

1) אנחנו מתחילים בלשמור כל החלקים (pieces) של ה puzzle הנתון בתוך imgArr[].

2) טענו את המטריצה הנתונה בקובץ warp_mat_x_H... כדי שנוכל לבצע warp לתמונה שבאינדקס imgArr[0] ע"י warp.Pespective

3) מבצעים sift לכל התמונות ב imgArr ונקבל לכל תמונה Keypoints, descriptors ולתמונה ה i שומרים את ה KeyPoints במערך במקום KeyPoint(i), ואותו דבר עושים במערך descriptors

4) לולאת for שמתבצעת #pieces פעמים כאשר עבור האיטרציה ה- i. נמצא את התמונה שהכי מתאימה לתמונה imgArr[0] בצורה הבאה:

5) נעבור על כל שאר התמונות מאינדקס 1 עד n: - לכל תמונה נמצא matchingPoints ע"י פונקציית isMatching שמקבלת dscrp(i), dscrp(j) ומוצאת נקודות התאמה לפי ה threshold. נשמור בנוסף את ה src_Kp, dst_Kp נקודות עניין של מקור וtarget - נקרא ל ransac שתרוץ משהו כמו 2000 איטרציות שתחזיר לנו מטריצת טרנספורמציה ע"י בדיקת כל פעם 3/4 נקודות ומצאת מספר inliers מקסימלי כפי שלמדנו, כלומר ransac תמצא טרנספורמציה עבור src,dst ע"י מצאת max_inliers

6) לאחר מצאת מטריצת טרנספורמציה ותמונה מתאימה ביותר ל img(0) נבצע warp לתמונה המתאימה ביותר להתאים ל imgArr(0).

7) נבצע stitching או blending לשתי התמונות ע"י mask_blending עם מסיכה בשורות, בעצרת שיטה זו נוכל לבצע blend יפה שמיר על בהירות אחידה

8) לאחר בצוע blend של best_img עם imgArr(0) נשמור את המיצוג ב imgArr(0) ונמחק את best_img מהמערך ונמשיך בלולאה!

ב.

בחירה 1: תכנון פונקציית isMatching:

פונקציית isMatching היא פונקציה שמבצעת שלב 3 בשלבי הפתרון כך שהיא מחלצת את ההתאמות בין בוט וקטורי descriptors ומהתחלה שמנו לב שיעילות הפונקציה הזו מבחינת סיבוכיות זמן היא אחד מהדברים החשובים עבור בניית הקוד, מכיוון שהקוד שלנו משתמש בפונקציה הזו הרבה פעמים במהלך רידת הקוד ובידוע חישובים על מספר גדול של וקטורים בעלי מימד 128 פורש הרבה זמן, לכן עבדנו שוב ושוב על תכנון פונקציה זו באופן הכי יעיל שאפשר ובסוף בעבורת השינוי הזה קיבלנו תוצאות בהרבה פחות זמן ובכך יכולנו לבצע בדיקות ולפתור את שאר הבאגים.

בחירה 2: משחק Thresholds:

משנו הכרחי כדי להבין בהרכבת באגים שונים היא קביעת Thresholds מתאימים, ש לנו 4 משתנים אשר בחירת תווך ערכים נכון הוכל לאיזות יותר גבוהה/פתרון נכון יותר. המשתנים הללו הם:

Ratio threshold ← לרוב השתמשנו בערכים בין 0.8 ← 0.6 אבל הין כמה באגים שהצטרכנו להשתמש בערך Threshold מחוץ לתווך

RANSAC iterations ← מספר האיטרציות של RANSAC השפיע על זמן הריצה וגם על איכות התוצאה הסופית

RANSAC threshold ← threshold אשר בוחר מי inlier ומי outlier מנקודות ה match

matrix det ← הוספנו תנאי כך שמטריצת Transformation היא טובה אך ורק ה det שלה נמצא בתווך מסוים רוב הזמן בחרנו ב $0 < det < 5$

ג. איכות התוצאה:

- קיבלנו את הפאזלים האופיים מ puzzle_affine_1-8 פתרנו בצורה 100% בהרבה הרצאשונה למרות שבפאזל 8 הצלחנו להרכיב 33/36 בצורה טובה מאוד

- פאזל 10-9 באופן התקשינו הכי הרבה במציאת הפאזל ולא הצלחנו להרכיב את כל החלקים בצורה טובה, ולכן החלטנו להרכיב בחוץ אבל עם איכות טובה יותר. התקשינו מכיוון שיש הרבה חלקים לאותו אזור מה שגרם לשבושים רבים. כמו כן יש הרבה חלקים למשל פרצופים עם בגדים שונים, מה שגם גרם לשבושים.

- בהומוגרפי קיבלנו תוצאות עם 100 בפאזלים 1-5, בפאזל 6 התקשינו מאוד לחבר חלק 14 לתמונה לא הצלחנו, אבל קיבלנו תוצאה מצוינת

- פאזל 8-7 עם הלוח, הכי קשה היה 8 כי יש הרבה חלקים וכולם דומים, מה שגרם לנו להריץ הרבה הרצות עד שהצלחנו

- פאזל 9 הומוגרפי היה הפאזל הכי קשה מכן ה20, בהתחלה לא הצלחנו להרכיב שום חלק, כי הסקנו שהאלגוריתם לא מצא מספיק נק' התאמה. ולכן לא עבד, שמנו לב בחלקים שחלק 14 הכי מתאים ל 1 ולכן הרכבנו אותו ידנית בקוד, והמשכנו משם וזה עבד, בינתיים גם תשתוש באוטומטי תוך כדי.

- עוד קושי היה לקבל בהירות אחידה עם רמיזות, כשהשתמשנו בהתחלה ב addWeighted כדי גרם לבהירות לא אחידה, פתרנו בזכות ע"י שימוש ב mask_blending למיזוג תמונות.