**עבודה בית 1**

**ראייה ממוחשבת**

**מרצה: סימון קורמאן**

**המגישים:**

1. **שם: באסל סגיר**

**ת.ז: 316543909**

1. **שם: אסיה חטיב**

**ת.ז: 206217028**

**תאריך ההגשה: 25/04/2021**

**שאלות:**

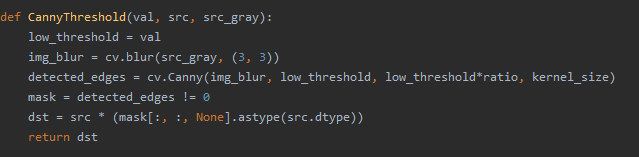
1. **Hough Transform on equilateral triangles**

במימוש של האלגוריתם השתמשתי בשלבים מוצעים שהיו בתרגיל הבית ורוב המדדים שעשיתי היו דירך ניסויים שהרצתי אותם על הקוד ובדקתי תשובות עבורם היה כמה TRADEOFF שעשיתי בקוד הקוד היה ניסוי להשתמש ב NSM קוד מהאינטרנט אבל לא עבד בצורה טובה לגבי ההסתברות לכל משולש אז הוא היה בוחר את המשולש הכי טוב או המתאים לצייר אותו אז ממשתי קוד קצר שבודק את המשולש והפיקסלים שנמצאים עליו אם הם EDGE ב EDGE MAP אבל לחוסר הזמן לא הספקתי לממש אותו בצורה מושלמת אז הפסקתי אותו בקוד הוא לא פעיל .

נסביר את השלבים ונציג את הקוד המתאים עבור כל שלב ,

* + **א. חשבו map edge בינארי )ניתן להשתמש במימושים קיימים כלשהם, למשל של Canny detector - )אפשר לכוון את הפרמטרים של המימוש )אפילו פר תמונה( לשיפור התוצאה.**

בשלב הזה חישבתי את ה EDGE MAP עבור התמונה המקורית השתמשתי בקוד שנמצא ב DOCS של הספרייה CV שקוראים לו CANNYTHRESHOLD



הוא משתמש בקבועים שנמצאים בהתחלת ה MODULE של הקוד

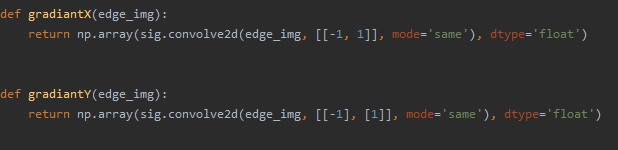
max\_lowThreshold = 100  
ratio = 4  
kernel\_size = 3

ועבור כל תמונה שלחתי VAL שמתאים ל ערך תחתון של ה THRESHOLD בהתאמה לתאורה התמונה .

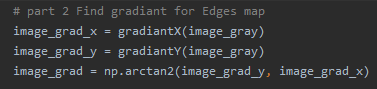
* + **ב. חשבו את כיווני ה-edges בעזרת חישוב גרדיאנטים**

חישבתי את ה גרדיאנטים לפי קונבלוציה השתמשתי במסכה שלמדנו בקורס עיבוד תמונה על ידי המתודה לחישוב קונבלוציה של הספרייה CV

הקוד:



ואחר כך חישבתי את ה ARCTAN לגבי גרדיאנטים למצוא את הזווית של ה קו על ידי הקוד:



* + **ג. הציעו פרמטריזציה של Transform Hough ותארו איך מתבצעת ה"תרומה" של כל נקודת edge מכוונת לטבלה הצוברת של הטרנספורם.**

אחרי כמה חיפושים באינטרנט והתייעצות עם חברים

מצאתי שיטה גיאומטרית שהיא לשמור במרחב HOUGH את נקודה המרכז של המשולש ואת הזווית שהמשולש מסובב בה את גודל המרחב הוא הרזולוציה של התמונה\*120 מכיוון שכמעט כל פיקסל יכול להיות מרכז המשולש ו כי משולש שווה צלעות אחרי כל 120 דרגות סיבוב יחזור למצבו שהיה בהתחלה.

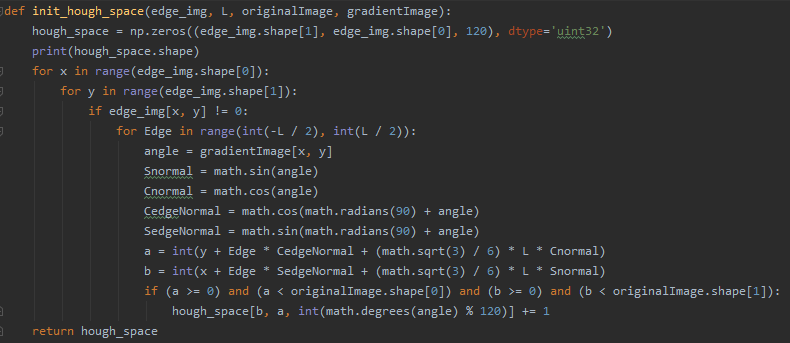
מקור הנוסחאות למצוא את המרכז עם זווית נתונה וצלע שחשבתיה עבור כל תמונה באופן ידני עבור משולש אחד הוא :  
<https://byjus.com/jee/centre-of-mass-of-equilateral-triangle/#:%7E:text=In%20geometry%2C%20the%20equilateral%20triangle,the%20three%20sides%20are%20equal.&text=In%20an%20equilateral%20triangle%2C%20the,at%20which%20its%20medians%20meet>

אתחול המרחב היה בצורה הבאה :

Hough space = np.zeros((edge\_img.shape[1], edge\_img.shape[0], 120), dtype='uint32')

שיטה ה VOTING הייתה בצורה הבאה שעוברים על כל פיקסל בתמונה ה EDGE ובודקים אם הערך שלו שונה מ 0 כלומר EDGE אז אנו מחשבים את המרכז של המשולש שהוא נמצא עליו מכיוון שאנו לא יודעים את היחס שלו איפו הוא נמצא על הצלע אז הרצנו על כל היחסים האפשריים שלו ושזה היה TRADEOFF עבור מקרים שעושים הצבעה למקום לא נכון אבל בצורה זו גם בודקים את כל המשולשים האפשריים שהוא יכול להיות EDGE להם ומצבעים להם.

* + ד. ממשו את לולאת ה-Hough שממלאת את הטבלה - בחרו באורך צלע משולש ובגדלי steps לכל תמונה בנפרד, לתוצאות מיטביות

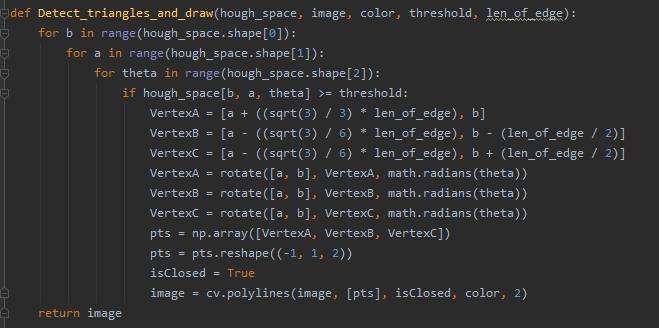
 הקוד:

עברתי על הלולאה של תמונה ה EDGE ועבור כל פיקסל אם הוא EDGE אז חישבתי לפי ההסבר בסעיף קודם את ואם הוא נשאר במרחב של התונה אז הוספתי 1 לערך המשולש במקום המתאים במרחב HOUGH .

* + **ה. בחרו בסף, כלומר threshold( לכל תמונה( על כמות ה-votes כדי לדווח על משולש. במקרה הצורך, ניתן להפעיל suppression maximal-non לשיפור התוצאה.**

בסעיף זה עבור ה THRESHOLD של ה VOTING לפי דעתי היה אמור להיות 3\*אורך הצלע פחות או יותר, מסיבה שכל פיקסל על ה EDGE היה אמור להוסיף 1 לערך במרחב HOUGH אבל כנראה יש בעיה בנוסחאות הגיאומטריות שמצאתי באתר או במימוש אז השתדלתי לשפר ככל האפשר עבור כל תמונה הדפסתי את הערכים וראיתי מה הערך המקסימאלי ועבורו בחרתי את ה THRESHOLD ראיתי כמה משולשים הגיעו לערך הזה ובחרתי אותו והיה מספר נמוך לא כמו דעתי שהזכרתי למעלה .

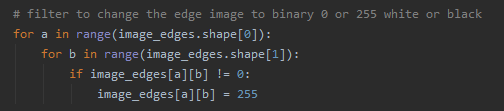
עבור NON MAXIMAL SUPPRESSION דיברתי בהקדמה של השאלה בדוך ..

הקוד :

בקוד עברתי על כל המשולשים במרחב ה HOUGH ומצאתי מי שגדול מה THRESHOLD שבחרתי וציירתי את המשולש על התמונה המקורית.

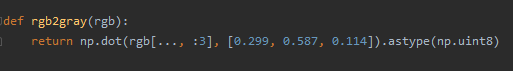
הסברים על שיטה המימוש :

השתמשתי גם ב פילטר שהופך את התמונה לתמונה בינארית הקוד :



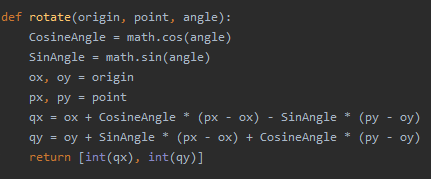
היה שיפור בתוצאות אחרי השימוש של הפילטר הזה

ממשתי כמה פונקציות עזר כמו



שהופכת את התמונה לתמונה GRAYSCALE

וגם :



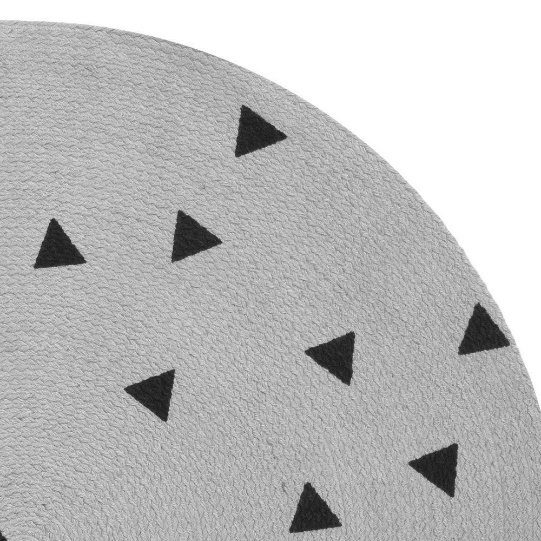
פונקציה שעוזרת לי בציור של המשולשים שהופכת את המשולש לקחתי אותה מהאינטרנט .

def process\_img\_q1(image, len\_of\_edge, vote\_threshold, canny\_threshold):

זזת הפונקציה הראשית של הMODULE של HOUGH שמקבלת כי פרמטרים התמונה גודל הצלע ה THRESHOLD של ההצבעה ו הTHRESHOLD של ה EDGE MAP .

**תוצאות:**

**עבור התמונה :**



**בחרתי פרמטרים :**

# len\_of\_edge = 100  
# vote\_threshold = 5  
# canny\_threshold = 100

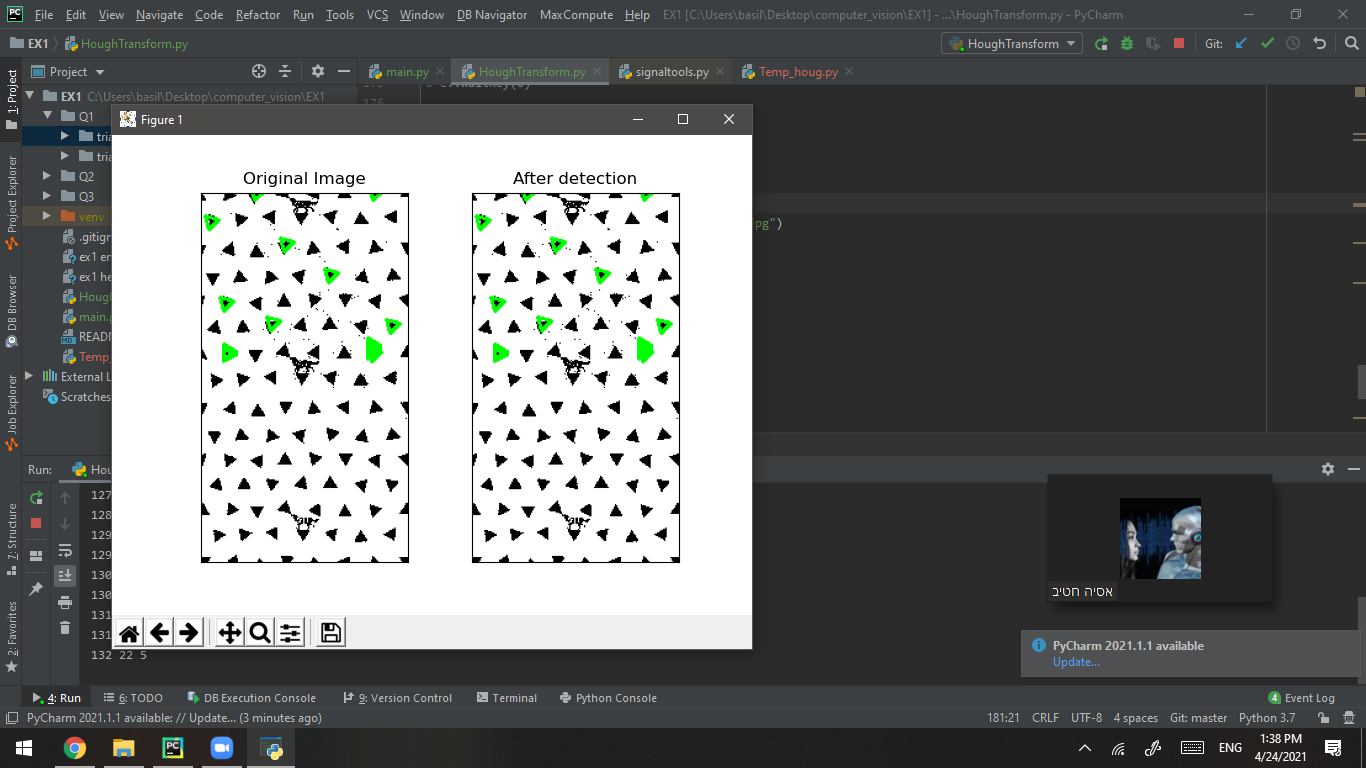
**גודל הצלע חישבתי באופן ידני גודל ה THRESHOLD היה על ידי ניסוי של כמה מספרים והדפסת המרחב קיבלתי את התוצאות הבאות :**



**עבור התמונה :**



**הייתה תמונה קשה ובחרתי באותה שיטה את בפרמטרים קיבלתי את התוצאות הבאות :**

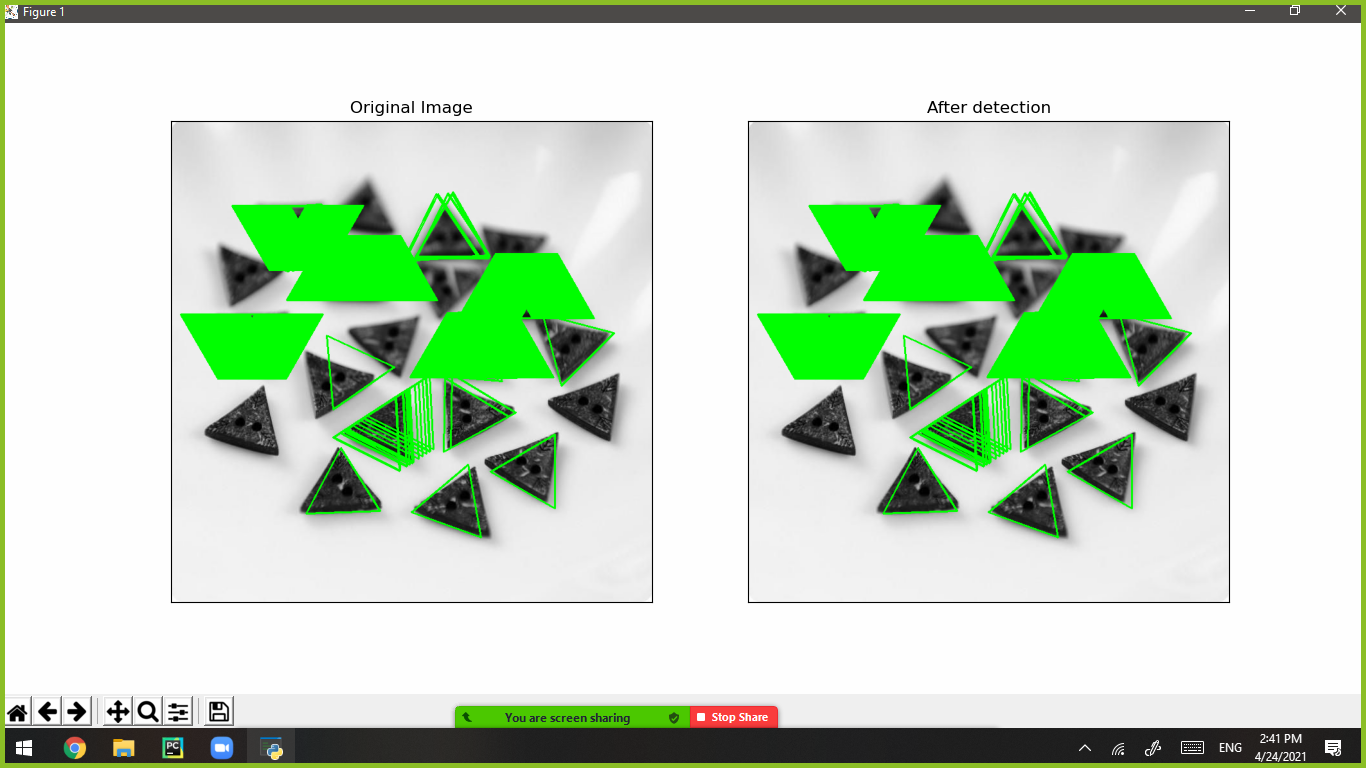


**היה פספוס של הרבה משולשות אבל כשמעלה את ה THRESHOLD שתופס את כל המשולשות היה הרבה FALSE POSITIVE אז הייתי צריך לממש את NMS לשפר אבל הסברתי בהתחלה שלא הספקתי .**

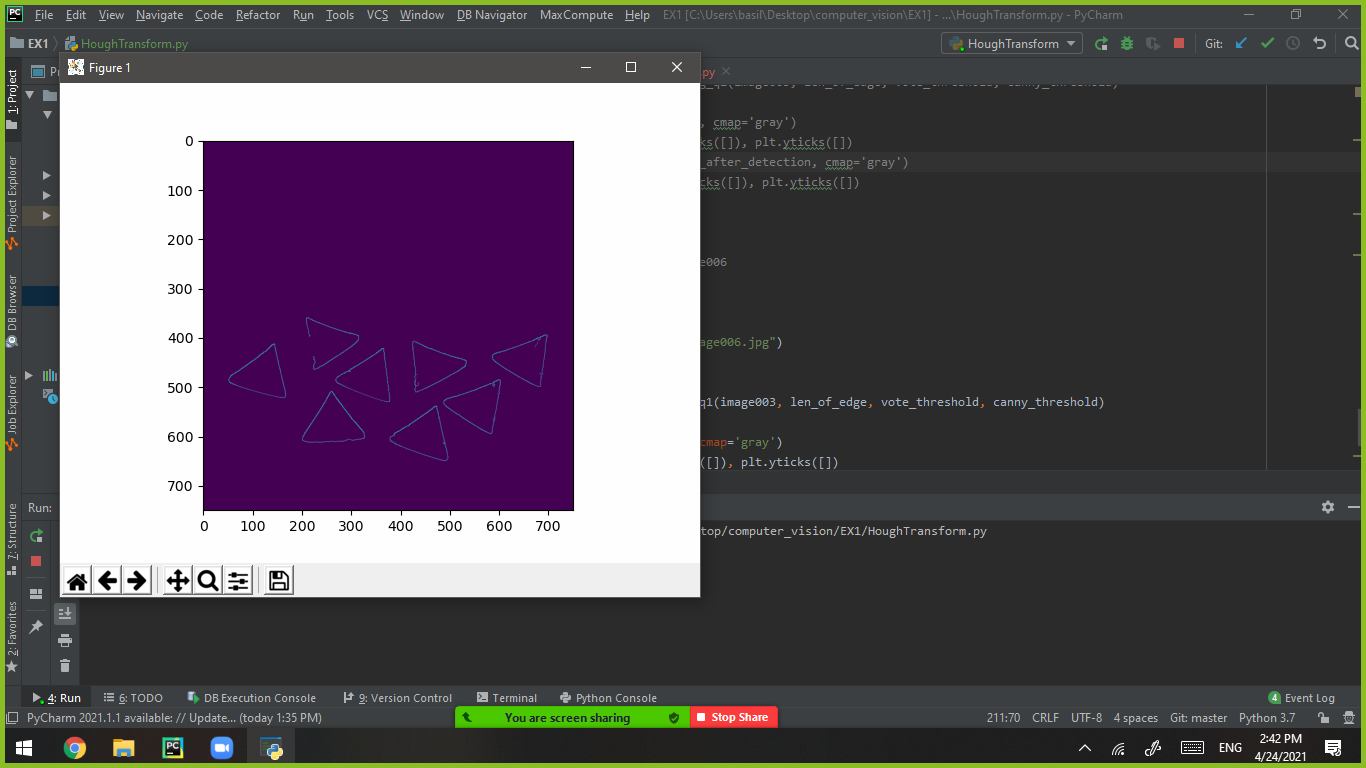
**עבור התמונה :**



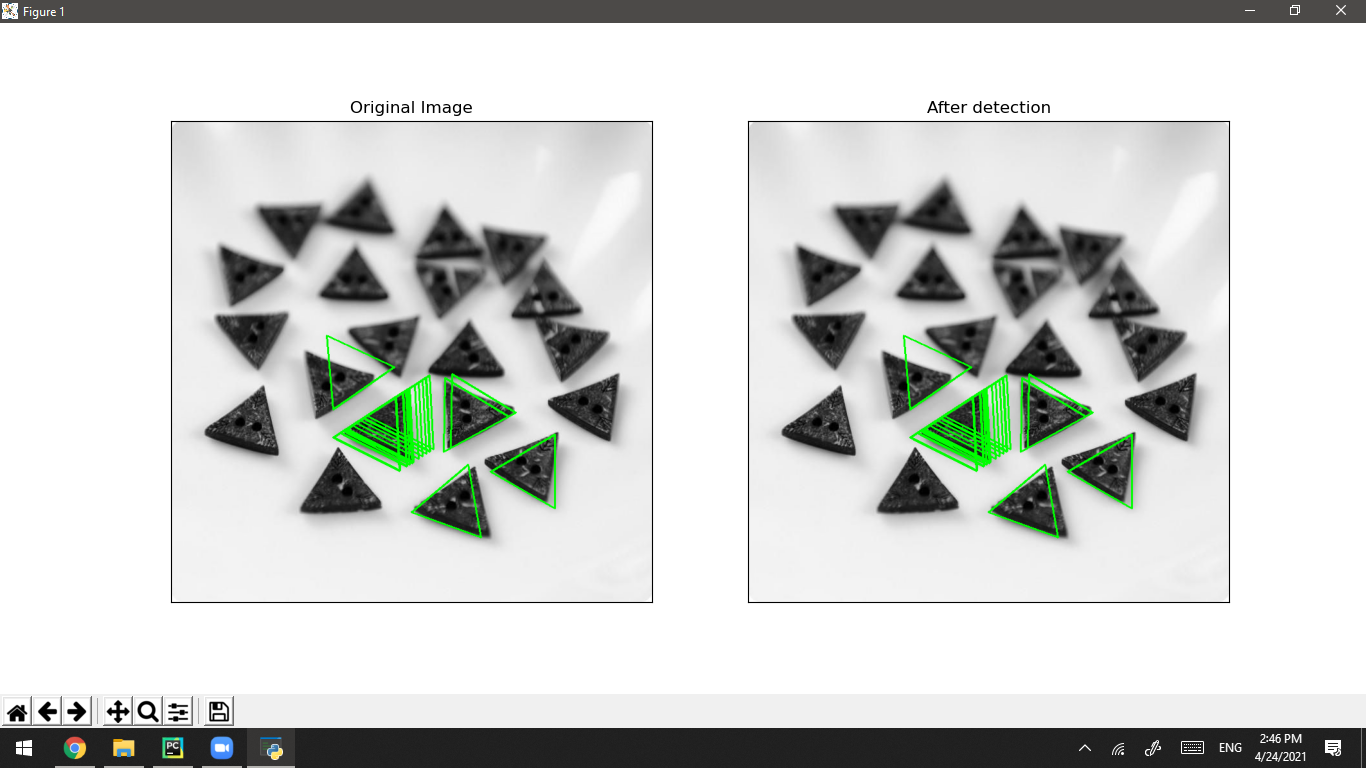
**מכיוון שיש BLURING אז עשיתי פילטר לתקן ב EDGE MAP והקטנתי את ה THRESHOLD של CANNY אבל גם כאן היה TRADEOFF עבור כמות המשולשים תוצאות :**



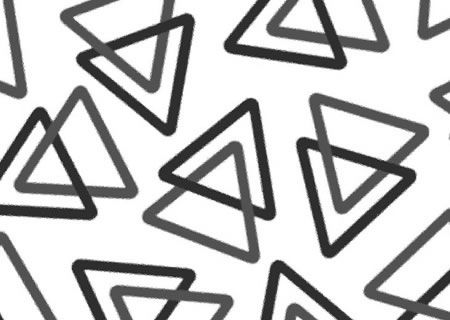
**למרות שהיה זיהוי של רוב המשולשות אבל היה הרבה FALSE POSITIVE אז הקטנתי את ה THRESHOLD של CANNY ויצא EDGE MAP כזה**



**וכאן זהה רוב המשולשים לדוגמא :**



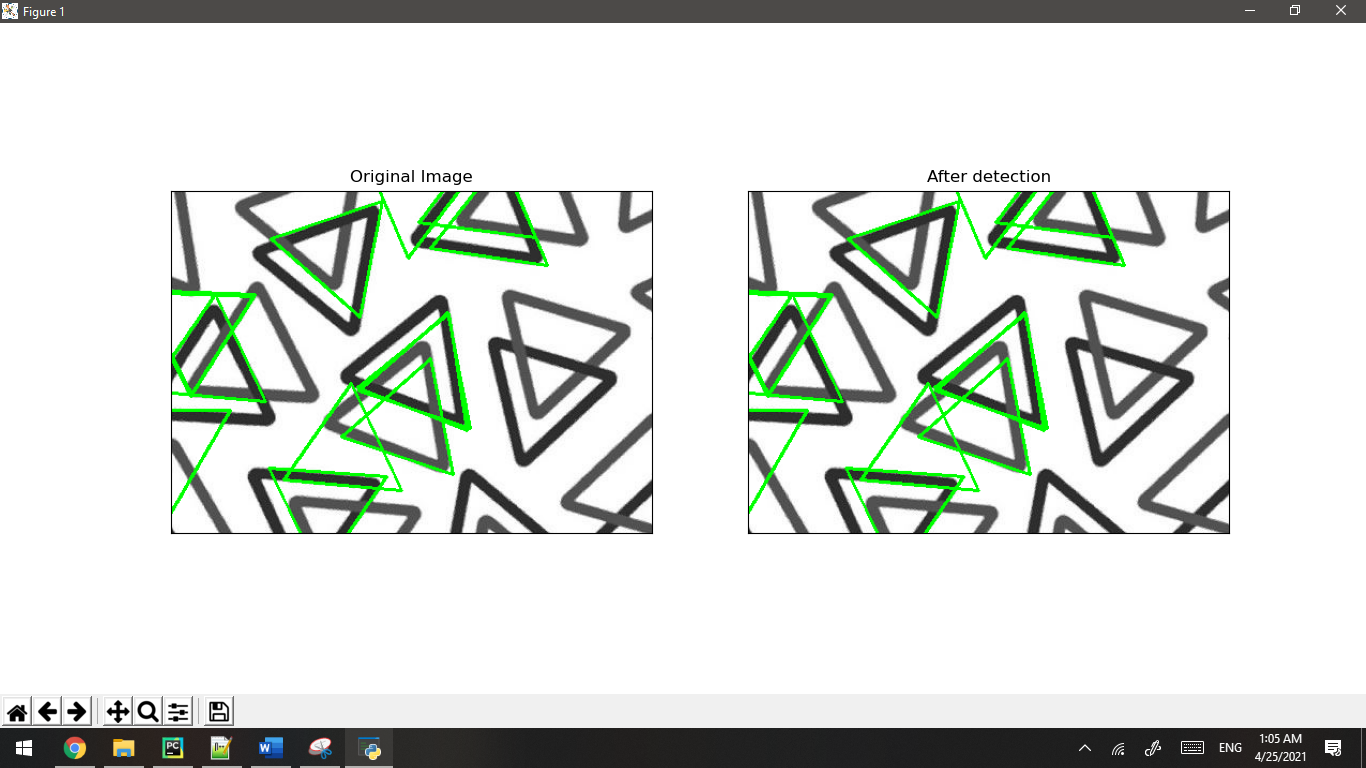
**עבור התמונה הזו :**



**בתמונה זו יש הרבה רעש וכל המשולשים הם צמודים אחד לשני אז התוצאות גם היו עם רעש למורת זזת NMS היה משפר את התוצאות עוד יותר טוב את הפרמטרים שבחרתי היו כך :**

len\_of\_edge = 110  
vote\_threshold = 10  
canny\_threshold = 100

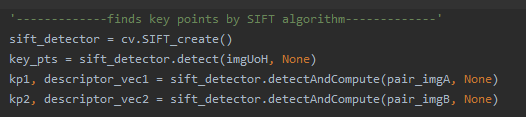
**והתוצאה :**



1. **SIFT-based correspondences**

במימוש של השאלה הזו השתמשתי בשלבים מוצעים שהיו בשאלה תרגיל הבית המודל של הקוד הוא הקובץ sift\_detection קיים פונקציה ראשית שקוראים לה main function שהיא מריצה את האלגוריתם ואת שלביו וחלקתי את הקוד לפונקציות ממוששות שיהיה קוד מודולארי ככל שאפשר.

* **השתמשו במימוש קיים של SIFT והפעילו אותו על כל אחד מזוגות התמונות. תוצאת ה-detector, לכל נקודת עניין )IP ,)הוא וקטור ]t, r, y, x ,]כאשר x ו-y הם קואורדינטות מיקום r ו-t הם ה-scale וה-orientation ,בנוסף לווקטור ה-descriptor ה-128 ממדי.**

בשלב זה השתמשתי במימוש קיים מספרייה OPENCV של SIFT הקוד :

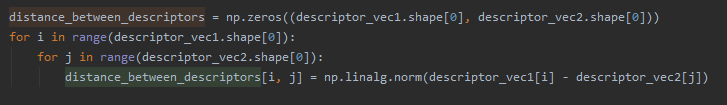
השתמשתי בפונקציה detectAndCompute של ספריה CV שעבורה שהחתימה שלה היא :  
cv.Feature2D.detectAndCompute(image, mask[, descriptors[, useProvidedKeypoints]]) ->keypoints, descriptors

מחזירה KEYPOINTS וגם DESCRIPTORS עבורם שמרתי אותם בשני וקטורים.

* **בהינתן שיש N ו-M נקודות עניין בזוג התמונות בהתאמה, חשבו את מטריצת המרחקים מגודל NxM בין כל זוגות וקטורי ה-descriptors.**

בשלב זה הגדרנו מבני נתונים שמכיל את מרחק בין שני וקטורים בתא I,J כך שהוא מטריצה בגודל N\*M שכל אחד מהם מייצג מספר ה KEYPOINTS שקיבלנו בשלה הקודם עבור כל תמונה.

הקוד:

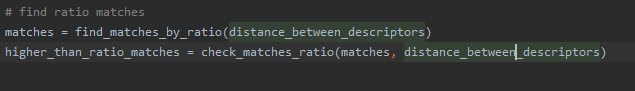


למציאה מרחק בין שני וקטורים השתמשתי בפונקציה LINALG NORM של הספרייה OPENCV לחשב את המרחק האוקלידי בין שני הווקטורים או הדיסקריבטורז.

* **חלצו את ההתאמות )matches )שעברו את: i .מבחן ה test-ratio( עם סף של 8.0) ii .מבחן ה test-bidirectional( דורשים שכל IP בזוג מותאם הוא neighbor-nearest של השני(**

רוב העבודה הייתה בשלב זה למציאה התאמות בין שתי נקודות KEYPOINTS השתמשתי בשטי שיטות שלמדנו בכיתה

TEST RATIO:

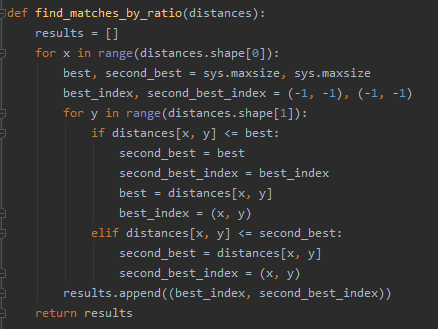
בשיטה זו היה שתי פונקציות ראשיות שקראנו להם בצורה הבאה :

הפונקציה הראשונה שהיא מוצאת את ההתאמות הכי טובות עבור כל KEYPOINT ואת ההתאמה השנייה הטובה ביותר על ידי הקוד הבא:

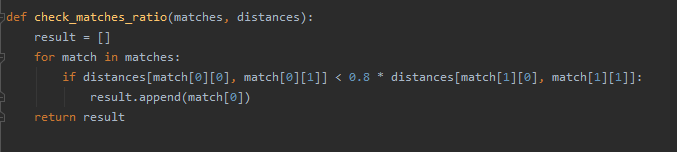
קלט : מטריצה המרחקים שחישבנו בשלב הקודם,

פלט : את ההתאמה הטובה ביותר ואת ההתאמה השנייה הטובה ביותר עבור כל KEYPOINT

קוד:



קוד שקיבלתי בראיון אחרון שלי ב NVIDIA 😊

והפונקציה הראשית השנייה היא למצוא את ההתאמות שעומדות בתנאי שלמדנו בכיתה שההתאמה הטובה ביותר חייבת להיות יותר טובה מהשנייה הטובה ביותר \* 80% הקוד:

קלט: התאמות שמצאתי שבכל תא קיים הטוב ביותר וגם השני הטוב ביותר.

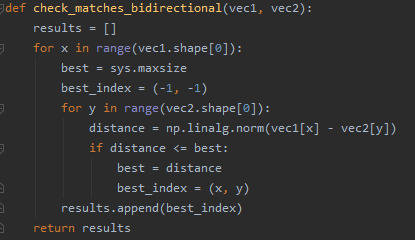
פלט:

MATCH מייצגת TUPLE שמכיל 2 TUPELS שהם הטוב ביותר והשני הטוב ביותר אז מהסיבה שעכשיו אני לא צריך את השני הטוב ביותר החלטתי להחזיר הטוב ביותר MATCH [0].

השיטה השנייה BIDIRECTIONAL שהיא מוצאת עבור כל KEYPOINT בתמונה 1 את הטוב ביותר שלה בתמונה 2 ועבור כל KEYPOINT בתמונה 2 את הטוב ביותר בתמונה 1 ואחר כל לוקח כל זוג שבחרו את עצמם 😊 כלותר כל התאמה שההתאמה הסימטרית שלה נמצאת גם.

את הקוד מומש על ידי שתי פונקציות עיקריות :

עבור כל כיוון קראתי פונקציה check\_matches\_bidirectional

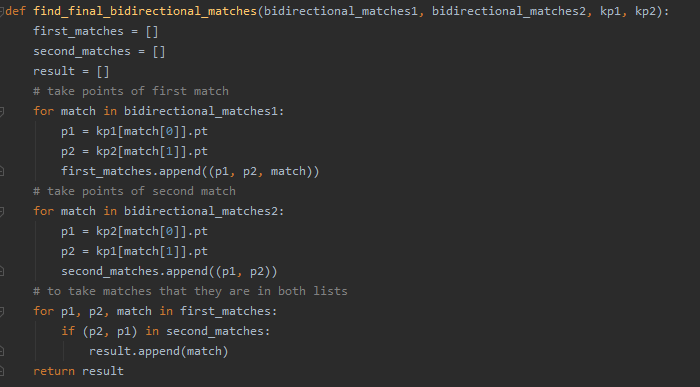


מוצאת עבור כל נקודה את הנקודה בטובה ביותר עבורה וקראתי לה פעמים כל פעם עבור כיוון אחד

אחר כך ממשתי את הפונקציה למצוא את ההתאמות הסימטריות שנמצאות בשתי הרשימות של ההתאמות שמצאתי בשלב זה

השתמשתי בקוד הבא לעשות את זה :

\*תזכורת שבוויקטור ההתאמות נמצא את האינדקס של ה KEYPOINT אז לקחתי את ערכם לפני הקוד בדף הבא..

הקוד:

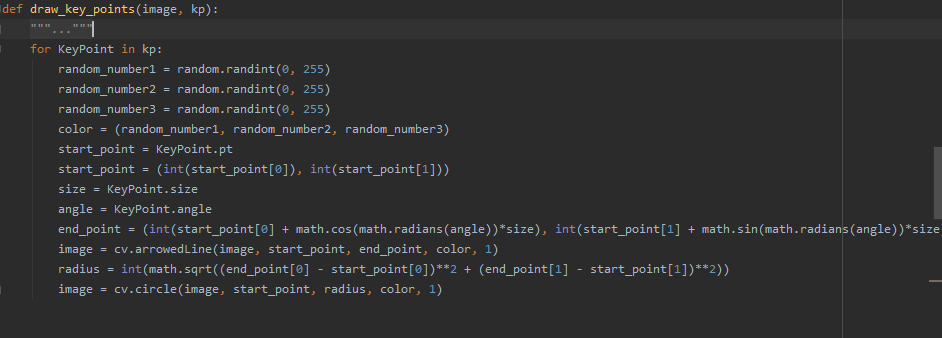
בלולאות הראשונות לקחנו את הערך של הנקודה למצוא אם היא נמצאת גם ברשימה השנייה ולקחנו את הMATCH לשמור על אותו סדר כמו שהיה כך שיהיה פונקציה צייר אחת בסוף של הקוד .

ובלולאה האחרונה בדקתי אם הסימטרית שלה נמצאת ברשימה השנייה אז לקחתי את ה MATCH הזו כי היא עומדת ב תנאי של השיטה.

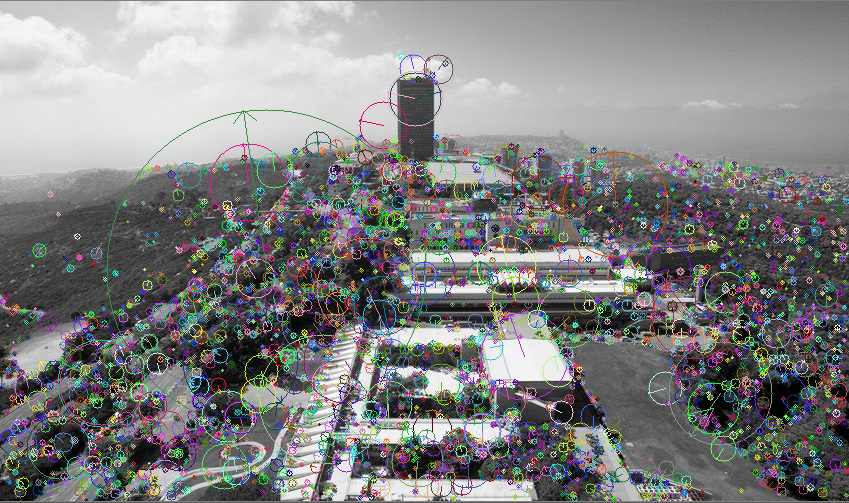
המשך הסבר על השאלה :

לצורך מימוש שלם היה לי עוד פונקציות שממשתי עבור הקוד למשל שתי הפונקציות שמצירות על התמונות בהתחלה השתמשתי בפונקציה של הספרייה של OPENCV וממשתי פונקציה שעושה אותו דבר בדף הבא הסבר על שתי הפונקציות :

KEYPOINT DRAWING

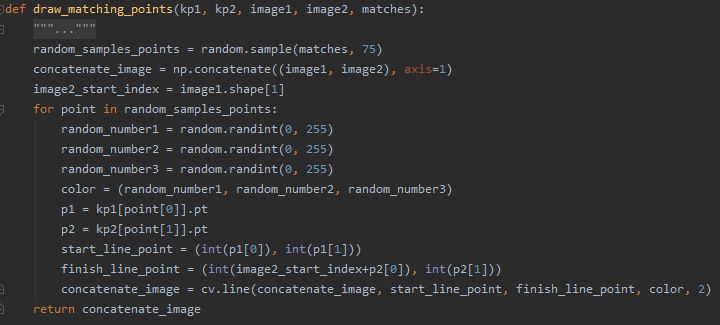
הקוד:

בפונקציה הזו ציירתי את החץ כמו שלמדנו בכיתה וגם מעגל שהרדיוס שלו הוא החץ בכל איטרציה לקחתי צבע רנדומאלי להבדיל בין המעגלים והחיצים בתמונה

תוצאה של הקוד :

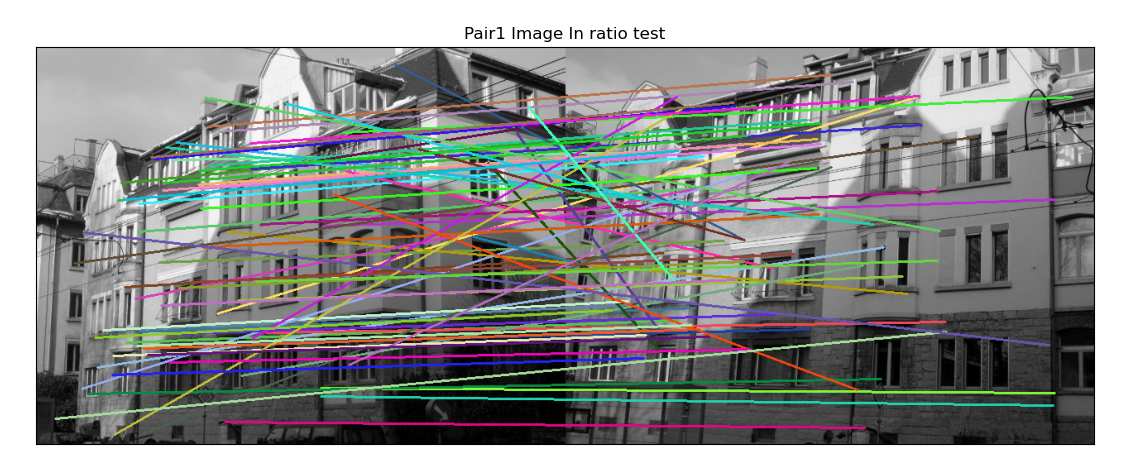
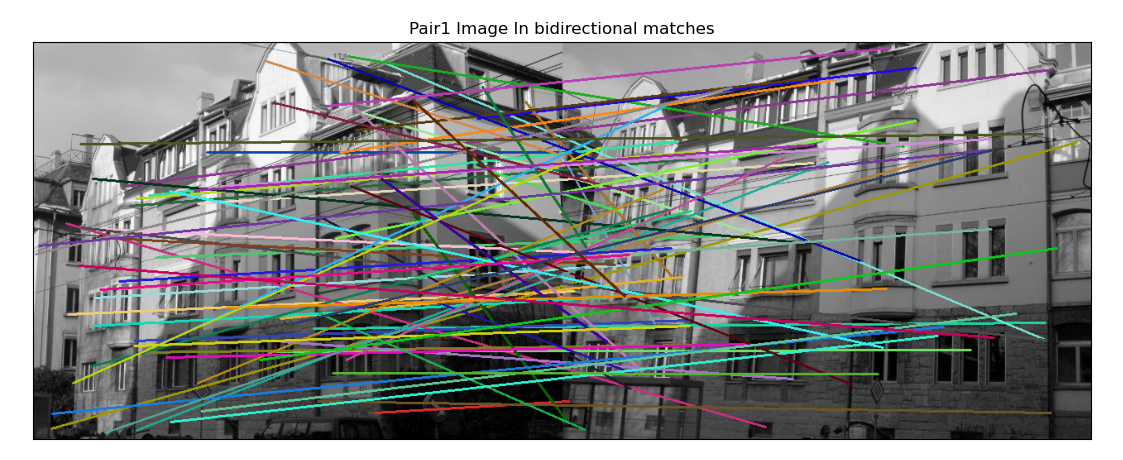
עוד תוצאה :

גם בדקתי את הפונקציה של OPENCV תוצאות דומות 😊.

הפונקציה השנייה שמציירת את הקווים בין שתי התמונות אז בהתחלת הקוד אנו מחברים את שתי התמונות אחת ליד השנייה ומתחילים לחבר בין ה KEYPOINTS שמצאנו לפי ההתאמות הקוד:

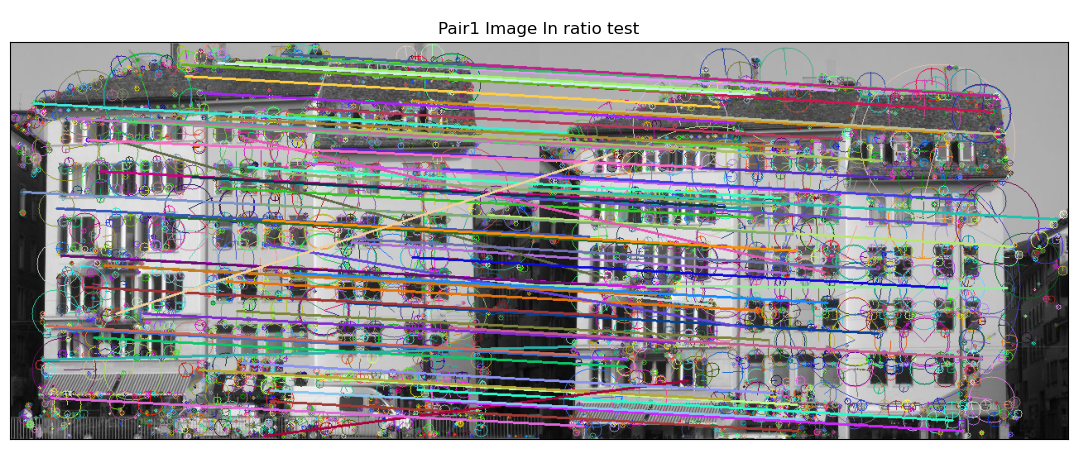
גם בקוד בחרתי בכל איטרציה צבע באופן רנדומאלי ולקחתי את הCOORDS של KEYPOINTS שהם מותאמים לפי שיטות קודמות עבור התמונה השנייה ההבדל שהם בציר X מתחילים מההתחלה של התמונה שהוספנו גודל התמונה הראשונה וצירנו את הקו ביניהם.

וגם בהתחלת הקוד ראיתי שיש הרבה דוגמאות אז לפי מה אמר המרצה לקחתי דוגמאות רנדומאלית בגודל 75 כי היה יותר התאמות אבל להיות סביר בציור של התוצאות .

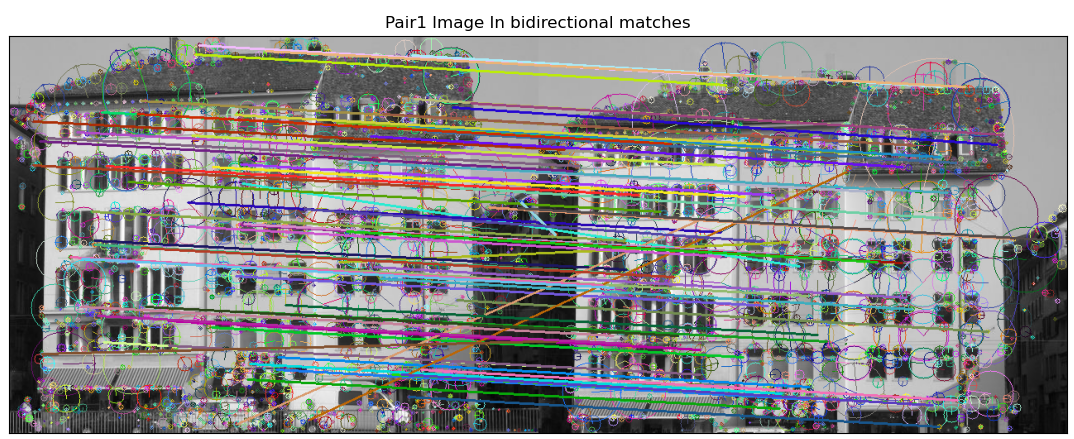
תוצאות עבור כל שיטה של ההתאמה :

תוצאות של זוג אחר עם ציור של KEYPOINTS והקווים :

RATIO TEST:

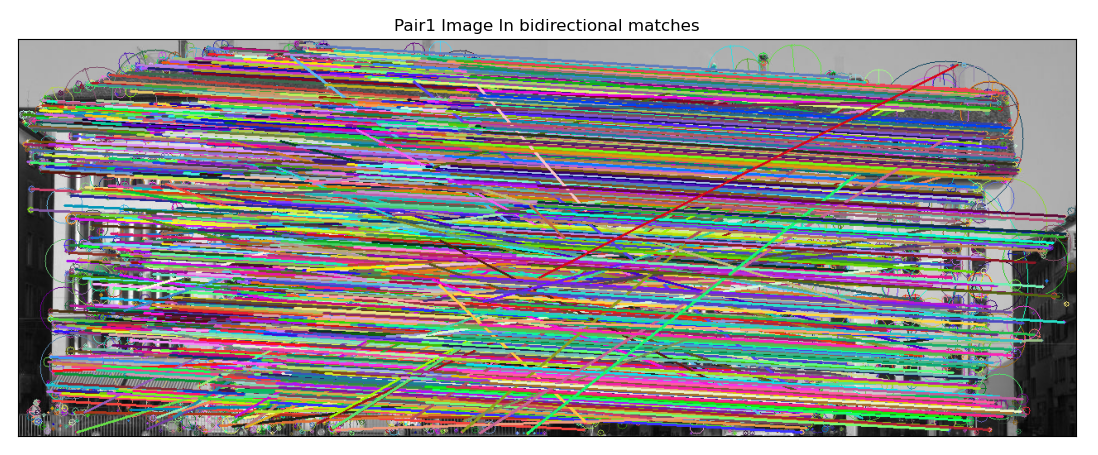


BIDIRECTIONAL:



עבור שטי השיטות ראיתי שיש הבד בקמות ה MATCHES שהם מוצאים BIDIRECTIONAL מוצא יותר הרבה מ TEST RATIO התאמות עבור KEYPOINTS בתמונה הראשונה מכיוון שהזווית של הצילום היא ממש שונה אז היה כמה FALSE POSITIVE התאמות לא נכונות בתמונה אבל נראה את העבודה המושלמת עבור זוג התמונה השני

אחרי שהדפסתי את כל ההתאמות למרות ש BIDIRECTIONAL היה יותר MATCHING אבל היה גם יותר FALSE POSITIVE מ TEST RATIO אז להשתמש ב BIDIRECTIONAL כשיש זווית יותר קשה ולקחת RANDOM SAMPLE יותר טוב מלמצוא ממש פחות התאמות עבור RATIO TEST . תוצאות עבור כל ה MATCHING:



**3) Image warping and RANSAC-based fitting**

**חלק 1**: הפעילו את ה-warps הנדרשים על מנת להעתיק את התמונה jpg.Dylan אל תוך המקבילית והמרובע שבתמונה jpg.frames.

* שלבים מוצעים
* : א. מצאו מימוש קיים של solver מינימלי - כזה שמשתמש ב-3/4 matches להתאמת טרנספורמציה אפינית/הומוגרפיה בהתאמה
* . ב. מצאו מימוש קיים של פונקצית warping ,אשר יכולה להפעיל טרנספורמציה אפינית או הומוגרפיה
* . ג. אספו באופן ידני סט התאמות מינימלי )בין פינות תמונת המקור לכל אחד מהמרובעים( והזינו אותם ל-solvers הרלוונטיים כדי לקבל את הטרנספורמציות )אחת אפינית אחת הומוגרפיה( המתאימות.
* ד. הפעילו warping והציגו את תמונת היעד שהועתקו אליה )פעמיים( תמונת המקור.