אלגוריתמים בראייה ממוחשבת - 046746

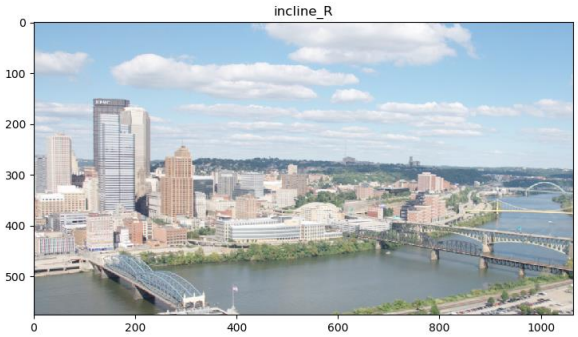
רטוב 4

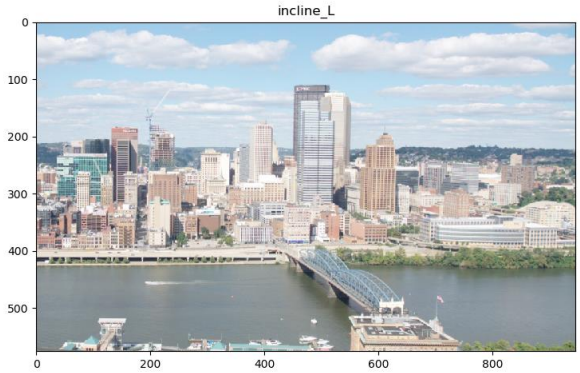
דניאל טייטלמן – 207734088 – Daniel.tei@campus.technion.ac.il

יאיר נחום – 034462796 – nahum.yair@campus.technion.ac.il

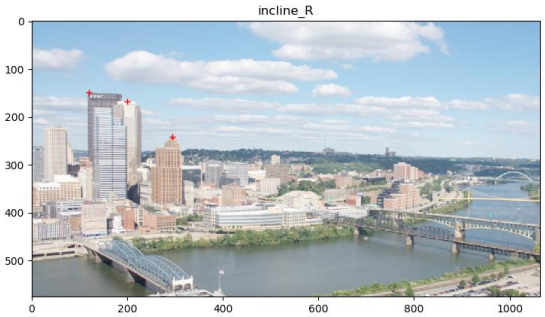
פרק 1:

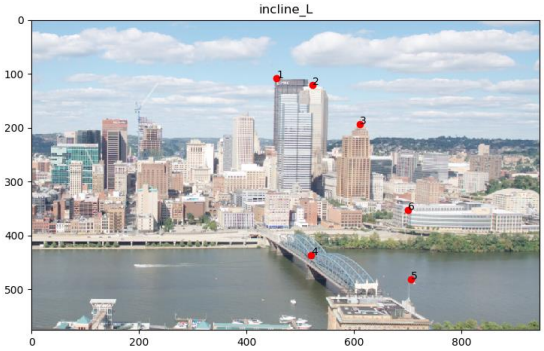
ראשית נטען את התמונות אשר נעבור עליהן במהלך סעיף זה:



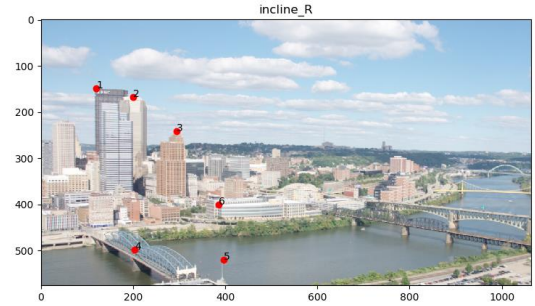


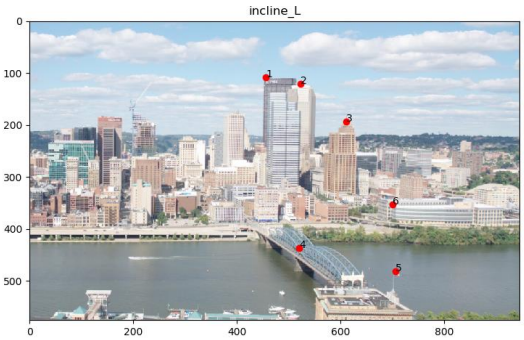
1. מימשנו את *GetPoints* ניתן לראות בתמונה הבאה, דוגמא לבחירת נקודות: (מסומן סדר בחירת הנקודות).





ובסוף הבחירה:





2. מימוש *ComputeH*, על מנת לחשב את מטריצת הטרנספורמציה ביצענו את השלבים הבאים:

א. בניית המטריצה *A* על פי ההוראות כך ש:

$$\begin{bmatrix} & & & & & \cdots & & & \\ x_i & y_i & 1 & 0 & 0 & 0 & -x_iu_i & -y_iu_i & -u_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_iv_i & -y_iv_i & -v_i \\ & & & & & \cdots & & & \end{bmatrix}$$

ב. חישבנו את המטריצה: $A^T A$.
ג. חישוב ע"ע וו"ע של המטריצה $A^T A$.
ד. בחירת וקטור עצמי (וקטור עמודה) בעל הע"ע הקטן ביותר.
ה. סידור הוקטור למטריצה 3 על 3 שנסמן h.

כאשר נרצה לבצע פרדיקציה בעבור הטרנספורמציה על נקודה $p_1 = (x, y, 1)^T$ נבצע את הפעולה הבאה:

$$p_2' = hp_1$$

לאחר מכן נעבור להצגה הומוגנית כלומר:

$$p_2 = \frac{p_2'}{p_2'(3)}$$

וכך נקבל את ההטלה של הנקודה.

מצורף חישוב לדוגמא:

p_1:

p1 - NumPy object array

	0	1	2	3	4
0	453.144	507.749	545.552	609.959	523.15
1	109.242	106.442	126.044	191.85	434.073

p_2:

p2 - NumPy object array

	0	1	2	3	4
0	121.202	184.127	218.735	294.245	207.724
1	148.57	151.716	170.594	241.384	486.79

h:

	0	1	2
0	-0.000912327	-9.33125e-05	0.00813653
1	0.00171019	-0.00253876	-0.999938
2	8.26694e-06	2.9067e-07	-0.006933

$$p_2\ reconstructed$$

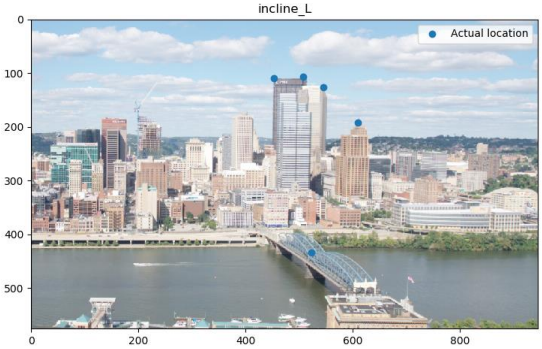
p2reconstructed - NumPy object array

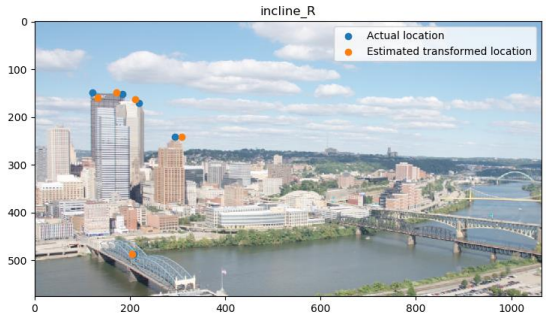
	0	1	2	3	4
0	131.681	171.944	210.093	308.625	205.341
1	159.206	148.574	162.148	241.917	486.411

$$l_2\ error$$

```
my_homography
Select 5 points
Estimation error: 25.69
```

תמונה עם הטרנספורמציה שחושבה:





3. נסביר את תהליך ביצוע ה – *Image warping* בשלבים:
   1. שלב ראשון נחשב *H* התחלתי כלשהו.
   2. לאחר מכן נרצה למצוא את גודל התמונה לאחר ביצוע הטלה ביחס למצלמה השנייה. לכן נחשב מה הוא המיקום החדש של קצוות התמונה בצורה הבאה:

$$transformedCorners = H^{-1}\begin{pmatrix} 0 & 0 & y_{max} & y_{max} \\ 0 & x_{max} & 0 & x_{max} \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

לאחר ביצוע הטרנספורמציה אנחנו חוזרים לקואורדינאטות רגילות (אין צורך בהומוגניות) ובוחרים את המינימום והמקסימום בכל כיוון. לקבלת 4 הקצוות. מכאן נרצה להגדיר את גודל out_size ובנוסף נרצה להזיז את *H* המקורי כך כפי שהמצלמה רואה אותם גדלים אלה יקבעו ע"י:

$$out_{size} = (Bottom - Top, Right - Left)$$

והזזה שנפעיל על *H* תהיה:

$$H_{shifted} = H\begin{pmatrix} 1 & 0 & Left \\ 0 & 1 & Top \\ 0 & 0 & 1 \end{pmatrix}$$

כעת יש לנו את גודל התמונה שהמצלמה תראה ובנוסף $H_{shifted}$ מטריצת הטרנספורמציה המותאמת להזזה הנדרשת בעבור המצלמה.

כעת נפעיל את פונקצית *warpH* יחד עם $H_{shifted}$ ו out_{size} נבצע את השלבים הבאים:

א. נמיר את התמונה לתמונה בין 0 ל 1 (נוחות בעבור שלב האינטרפולציה וחזרה לתחום של בין 0-255 בסוף).

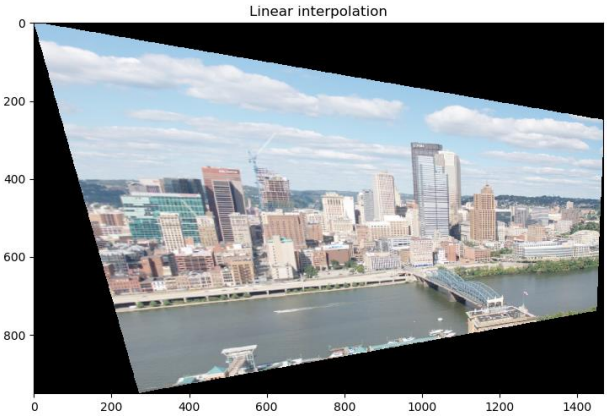
ב. נגדיר *Grid* ביחס לגדלי out_{size}.

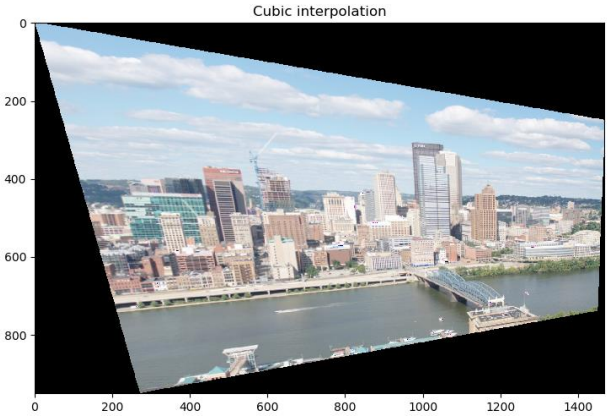
ג. נטיל את ה – *Grid* באמצעות הטרנספורמציה $H_{shifted}$. נבחר רק הנקודות בתוך גודל התמונה $im1$.

ד. נבצע לולאה על כל ערוץ, ונבצע אינטרפולציה ביחס לכל הערכים שבתוך התמונה.

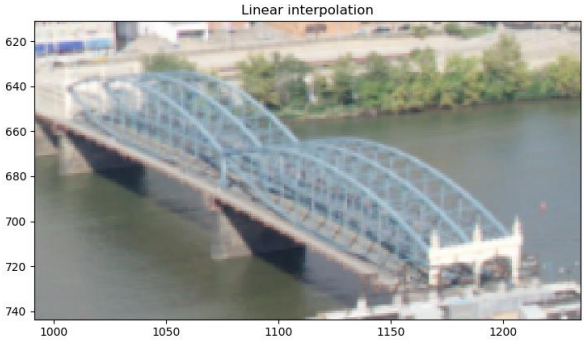
ה. נכפיל ב 255 ונחזור ל – *8UINT* ונקבל את *1IM_WARP*.

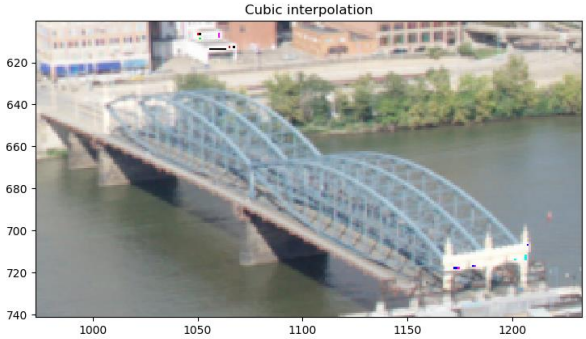
התוצאות:





נעשה זום אין על הגשר:





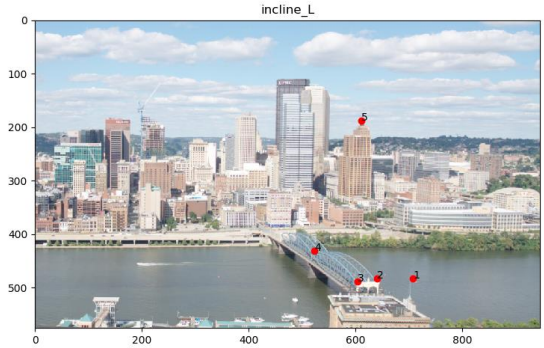
כפי שניתן לראות האינטרפולציה הליניארית מכילה פחות אזורים "לא מוגדרים" אך, נראית מעט פחות חדה.

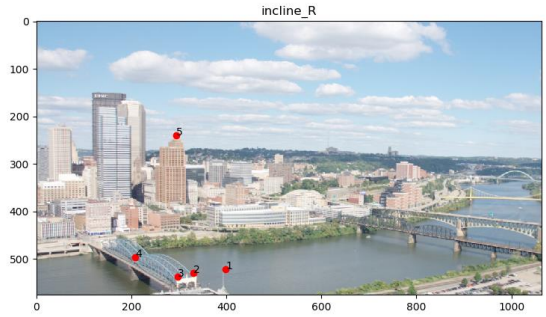
4. כעת נבצע *Image stitching*, על כן עלינו ראשית להגדיר את גודל התמונה כולה ולאחר מכן להפעיל המסכה במקומות הרלוונטים.
   - בשלב הראשון נגדיר כי גודל התמונה הוא:

$$y = \max(Bottom, img1[0]) - \min(Top, 0)$$
$$x = \max(Right, img1[1]) - \min(Left, 0)$$
$$pic\ size = (y, x, 3)$$

   - לאחר מכן נמקם את התמונה של ה – *WARP* במקומה על פי האינדקסים, ואת תמונת האפס *IM* נמקם ב *OFFSET* המתאים ביחס לגובה ורוחב.
   - כעת לאחר שכל תמונה נמצאת בתמונה נפרדת נאחד אותן, נבחר את כל האינדקסים הדו ממדים כך שהסכום לאורך ציר הצבע הוא גדול מ *OFFSET* כלשהו (נעשה זאת בעבור שתי התמונות), לאחר מכן באמצעות האינדקסים נשים אותם בתמונת הפנורמה ונקבל:

הדבר התקבל בעבור הבחירה הידנית הבאה של נקודות לחישוב *H*:

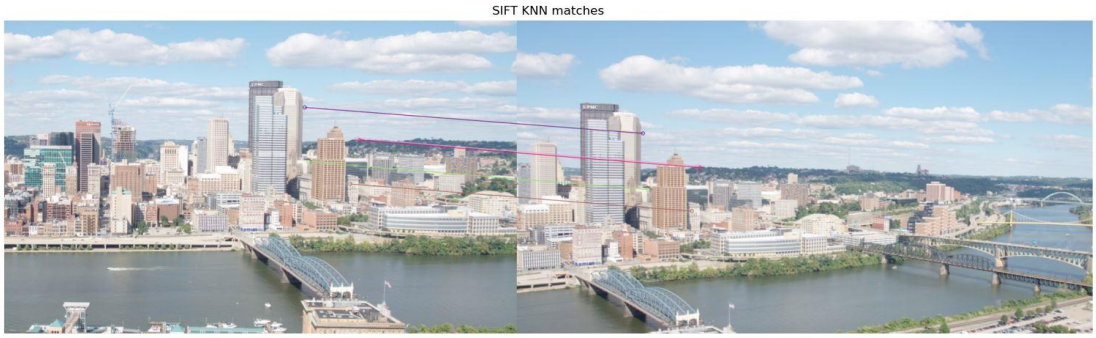




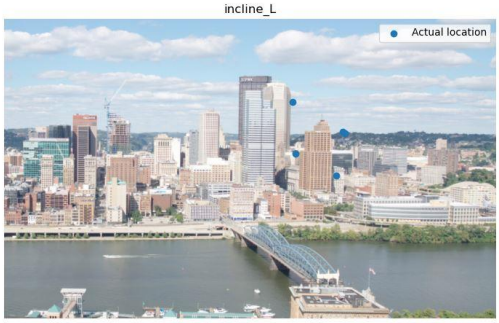
השגיאה בחישוב מטריצה *H* ביחס לנקודת הנ"ל היא:

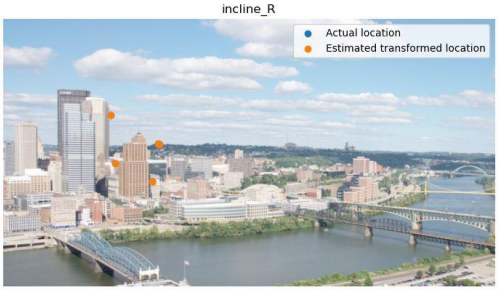
```
Estimation error: 2.415
```

5. בחרנו באלגוריתם התאמה *BFMathcer* מבוסס *KNN* עם 2 שכנים כפי שראינו בלינק. ע"י *SIFT* מוצאים את ה *feature points (FP)* ואת הדסקריפטורים המתאימים שלהן שכוללים מיקום (*x,y*), אוריינטציה, וסקאלה.
לאחר מכן בכדי למצוא נקודות תואמות בין שתי תמונות *(query/train)* אנו מריצים את אלגוריתם ה *knnMatch*.
אלגוריתם ההתאמה מחפש את ההתאמות הקרובות ביותר בין כל אחת מהנקודות שנמצאו בתמונה אחת לכל אחת מהנקודות בתמונה השנייה.לפי המרחק הדיפולטי שהוא נורמת *L2*. ושומר את ה *k* (פרמטר לפונקציה) הכי קרובות שנמצאו. אנו חיפשנו רק את שתי ההתאמות הכי טובות לכל נקודה ב *query image*.
לאחר מכן הפעלנו *ratio test* שלפי מאמר של *lowe* כדבר שמסווג התאמות חזקות ע"פ חלשות ורועשות. התהליך מסנן נקודות שיש להם שכן קרוב במיוחד ע"פ שכן שני הכי קרוב שרחוק לעומת ההתאמה הכי קרובה.
לאחר מכן אנו ממיינים את אלו שנשארו לפי המרחק הכי קטן (התאמה הכי טובה) ובוחרים את ההתאמות הכי טובות לפי פרמטר לפונקציה (כמה נקודות התאמה הכי טובות לקחת). אנו לקחנו רק את ה *N=5* הטובות ביותר וביצענו את חישוב מטריצת *H* , *warp* ו *panorama stitching* כמו בסעיפים הקודמים לפי ההתאמות שמצאנו עתה.
ההתאמות לאחר סינון, מיון ולקיחת ה 5 הכי טובות.



מכיוון שההומוגרפיה טובה, השחזור המדויק נופל ממש על ההתאמה (מסתיר את העיגולים הכחולים למטה).





אחרי *warp*:

ה *stitching* גם נראה יותר מדויק (החיבורים על שפת הנהר):

6. כשעבדנו בסעיף הקודם על תמונות ה *incline* עם התאמות מבוססות *SIFT* קיבלנו שגיאת שחזור מאוד קטנה *Estimation error: 0.025* לעומת השחזור של הנקודות שנבחרו ידנית *Estimation error: 2.415*.

גם ה *stitching* נראה יותר טוב אחרי התאמה עם *SIFT* וזה הגיוני שכן ההומוגרפיה יותר מדויקת.
בבחירה ידנית קיבלנו:

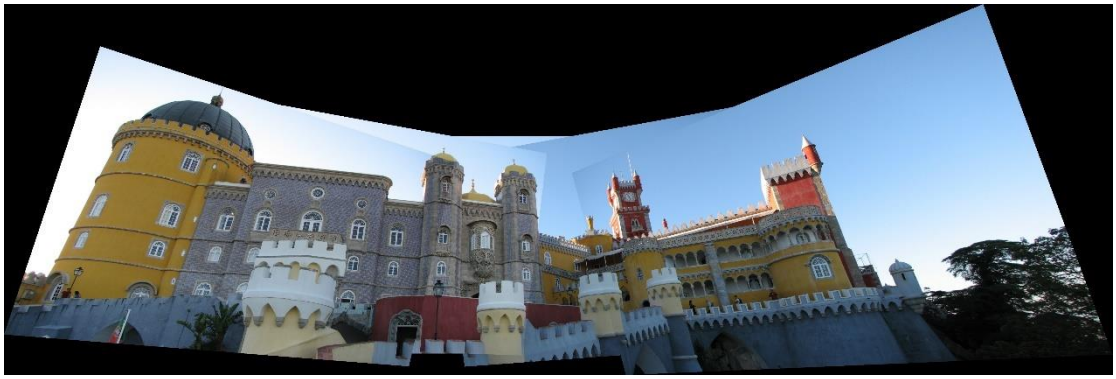
ובאמצעות *SIFT*:

ניתן לראות שהחיבור בשפות הנחל יותר מדויק עם *SIFT*.
בבחירה ידנית היתרון הוא שהאדם יודע לעשות את ההתאמה בצורה מדויקת לעומת *SIFT* שיכול למצוא *outliers* שיהרסו את ההתאמה לגמרי.
לעומת זאת, *SIFT* הוא תהליך אוטומטי שלא מצריך התערבות של מומחה שייתן את ההתאמות, ולכן עדיף לשימוש באלגוריתמי *CV* ללא התערבות אדם (שזו מטרת *CV*).
מימשנו פונקציה שעושה *panorama stitching* לרשימה של תמונות ע"י בחירת האמצעית כעוגן *(anchor)* וחישוב מטריצת *warp* בין כל צמד תמונות מהעוגן להתחלת הרשימה ומהעוגן לסוף הרשימה. בכל איטרציה מצרפים לפנורמה הכוללת את התמונה שעברה *warp*.

בעזרת *SIFT* ו *KNN matcher* התקשנו למצוא התאמות ללא *outliers* במיוחד בתמונת הצל התחתונה של פנורמת החוף.
לאחר ששינינו את אלגוריתם ה *matching* ל *bf.match* (במקום *bf.knnMatch*) והורדנו את מספר הנקודות (אחרת ה *outliers* הרסו) קיבלנו פנורמה טובה:

הפנורמה של תמונת החוף עם *SIFT*:

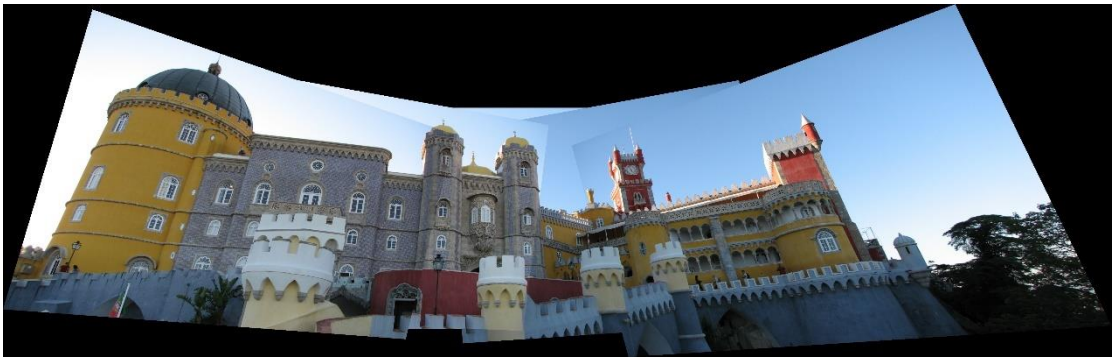
הפנורמה של הארמון בעזרת *SIFT*:



7. לאחר סינון של נקודות התאמה באמצעות *RANSAC* קיבלנו תוצאות הרבה יותר טובות ויכולנו לתת יותר נקודות ש *SIFT* יימצא שכן לאחר מכן אנו יודעים ש *RANSAC* יסנן בצורה טובה וישאיר רק את ההתאמות שייתנו את ההומוגרפיה עם השגיאה הקטנה ביותר (מכלילה כמה שיותר *inliers*).

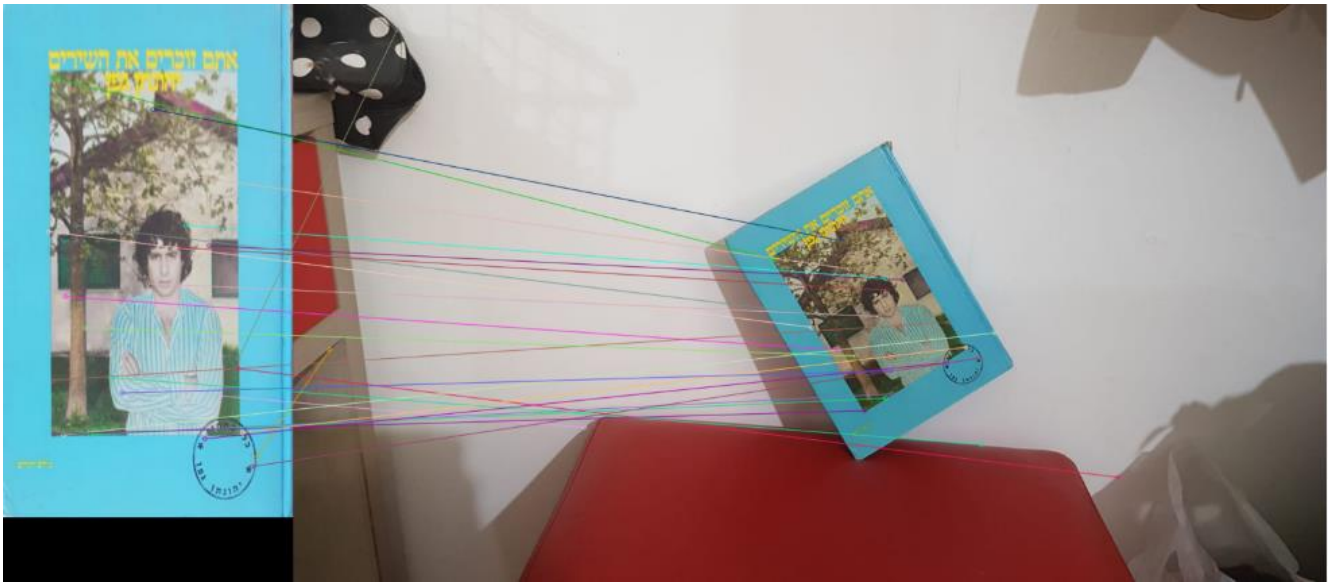
תמונת הפנורמה בחוף עם *SIFT + RANSAC* . ניתן לראות שההתאמה והתפירה יותר טובים. החלק העליון יותר ישר ביחס לגבולות התמונה:

תמונת הארמון אחרי *SIFT+RANSAC*:

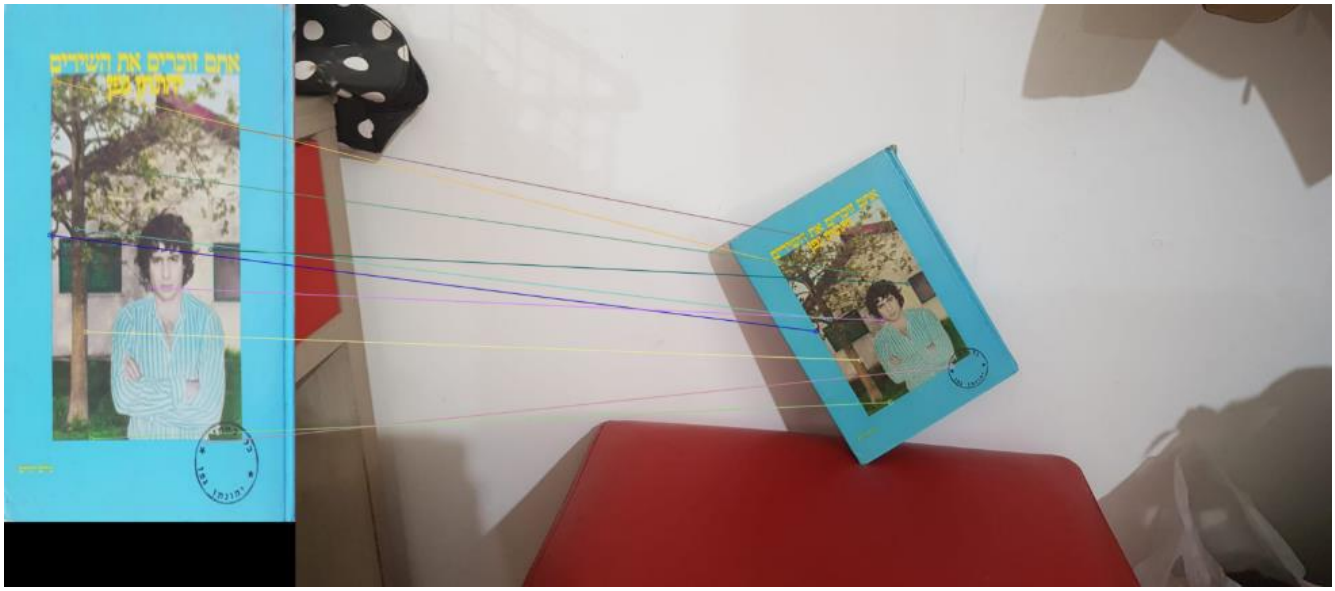


דוגמא טובה לשיפור הביצועים ומציאת ההתאמות הטובות ביותר באמצעות *RANSAC* ניתן לראות במיוחד בתמונה מתרגיל *2* שבה יש הרבה *outliers* שמסוננים בצורה טובה באמצעות *RANSAC*.

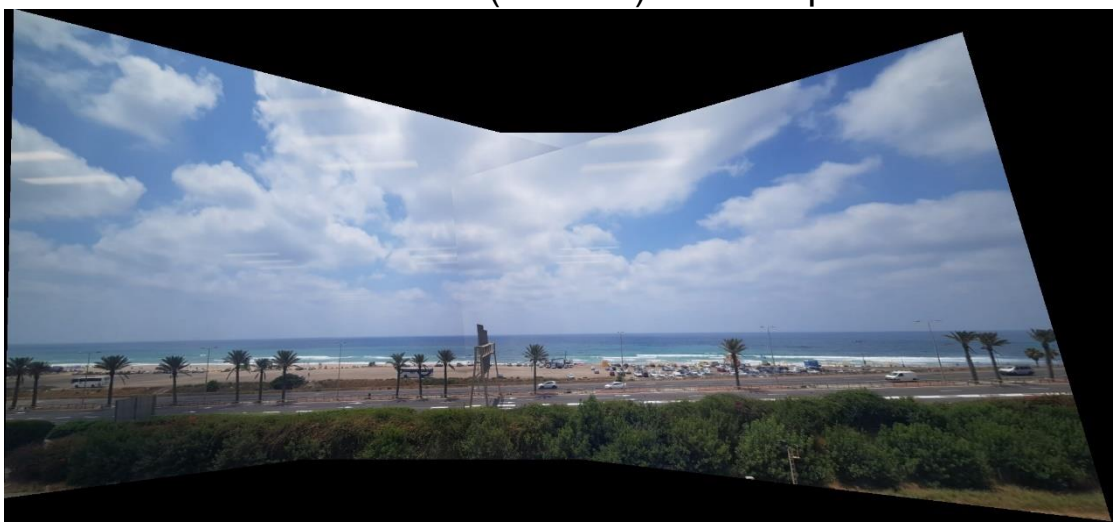
לפני *RANSAC*:



ואחרי (ממש רואים שה *outlier* לכיסא והצל על הקיר סוננו):

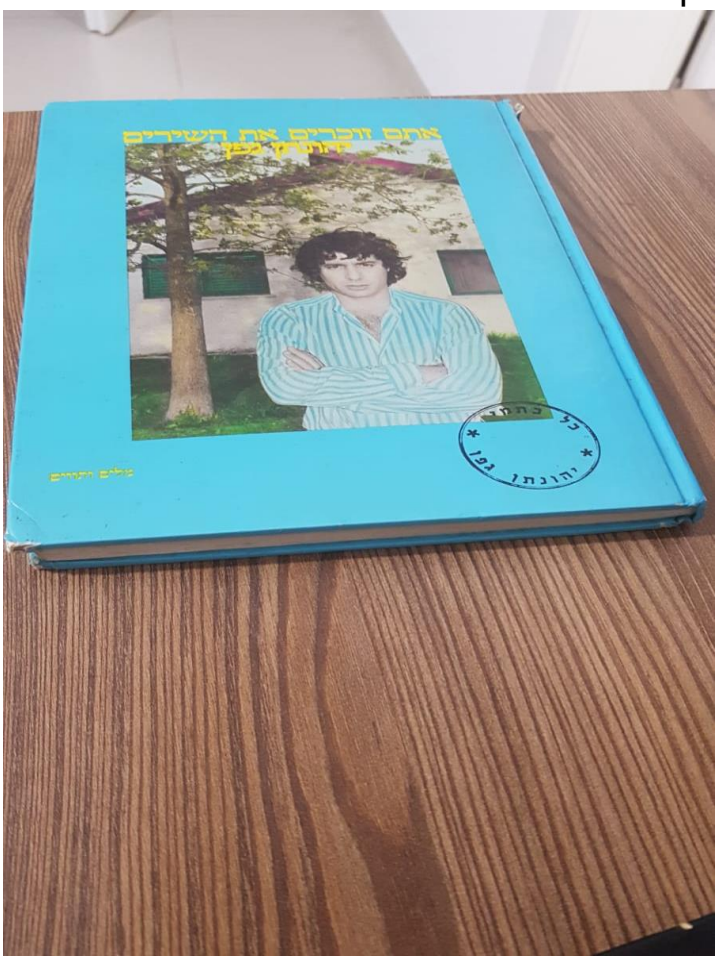


8. תמונת פנורמה מהחוף של חיפה (3 תמונות):



פרק 2:

1. לקחנו את התמונה הבאה:

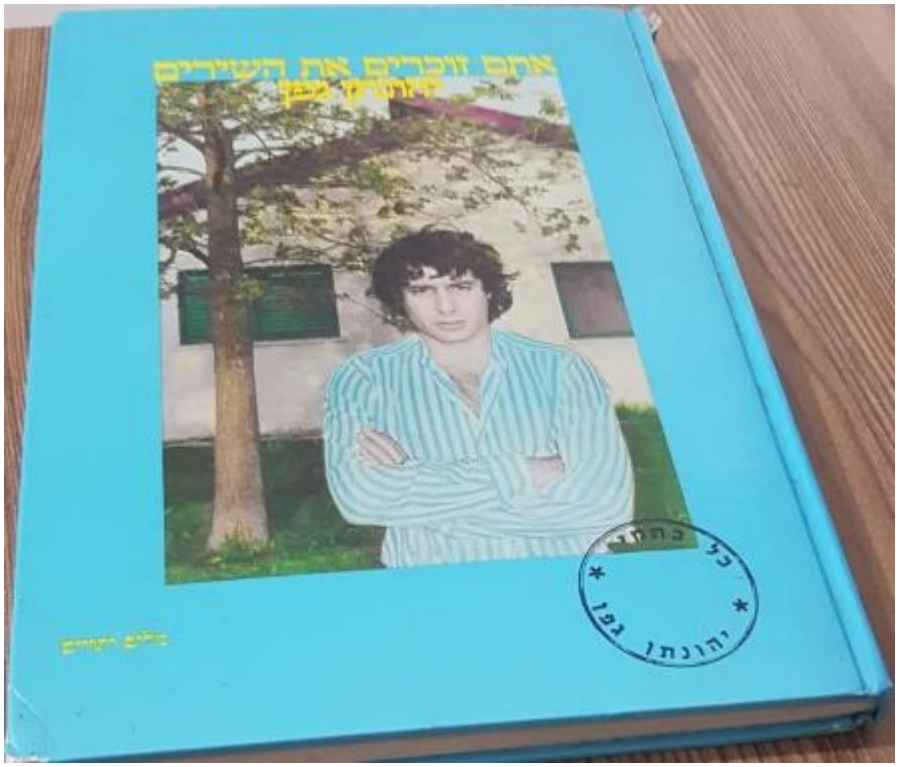


אחרי *downscaling* ב 2 בכל מימד (להאצת חישובים), באמצעות *manual selection* של נקודות הפינות של הספר והתאמתם לנקודות מטרה בגודל תמונה סביר חישבנו את מטריצת ההומוגרפיה ועשינו *warp*.

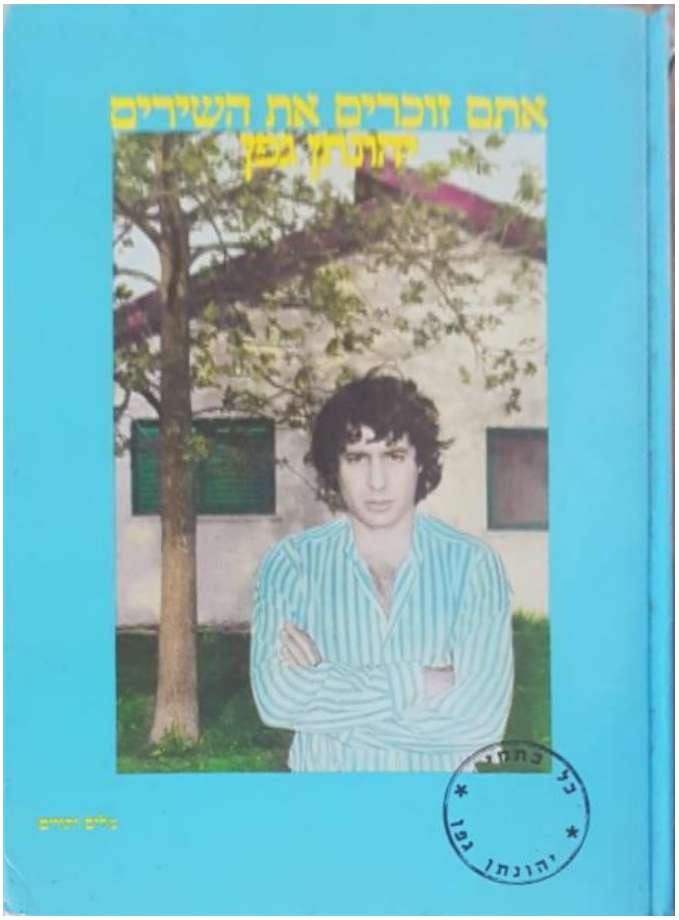
* ה *aspect ratio* של הספר לא ממש נשמר לפי איך שהוא אמור להיות במציאות אבל עדיין החישבנו את ההומוגרפיה לתמונה בגדלים סבירים ותמונת הרפרנס יצאה בסדר.

* עשינו גם *crop* לתמונה לפני ה *warp* (לפי הנקודות שניתנו) בכדי לחסוך פעולות אינטרפולציה ב *warp*.

אחרי *crop* קיבלנו:

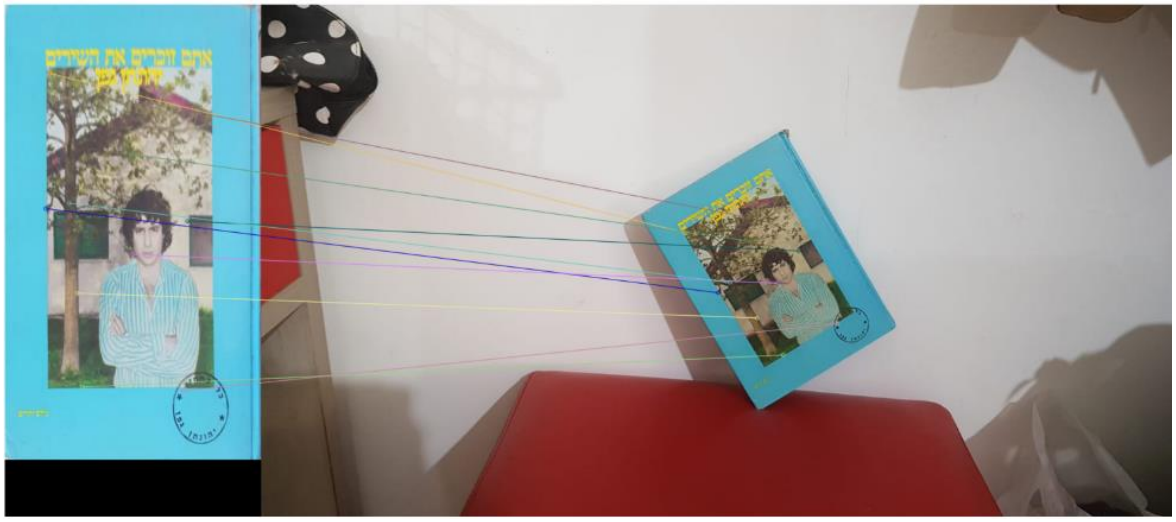


ואחרי *warp* קיבלנו את תמונות הרפרנס:

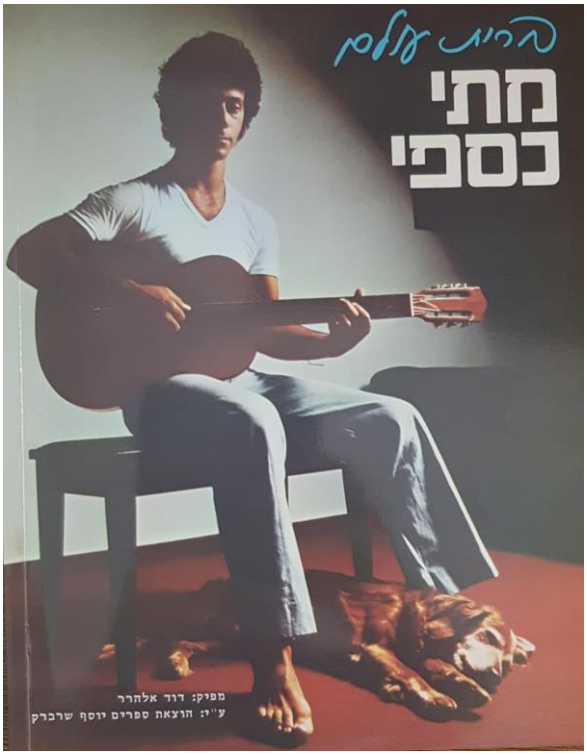


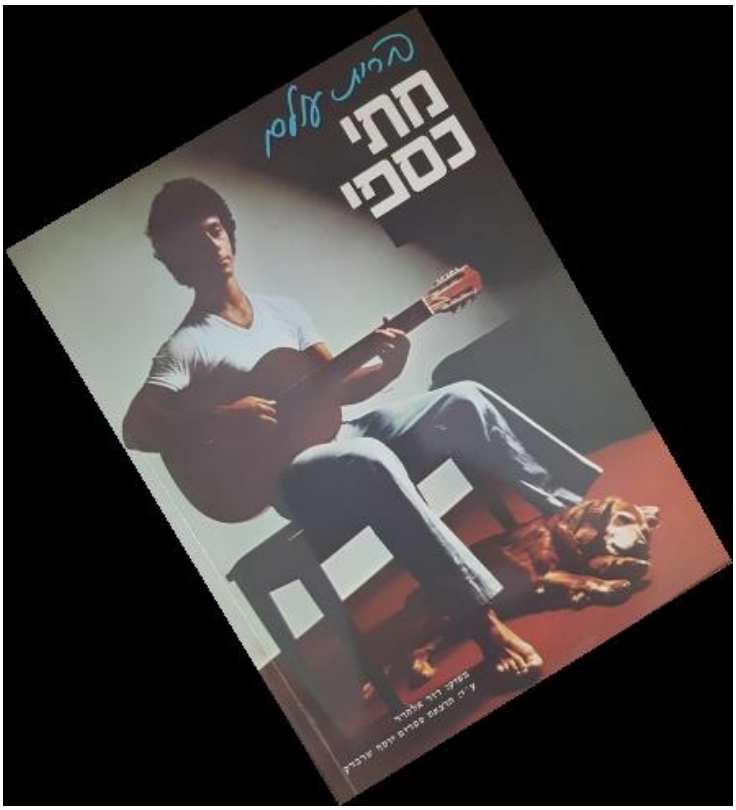
2. תמונת הסצנה שיצרנו:

מצאנו את ההומוגרפיה בין תמונת הרפרנס לתמונה בסצנה באמצעות *SIFT+RANSAC*:



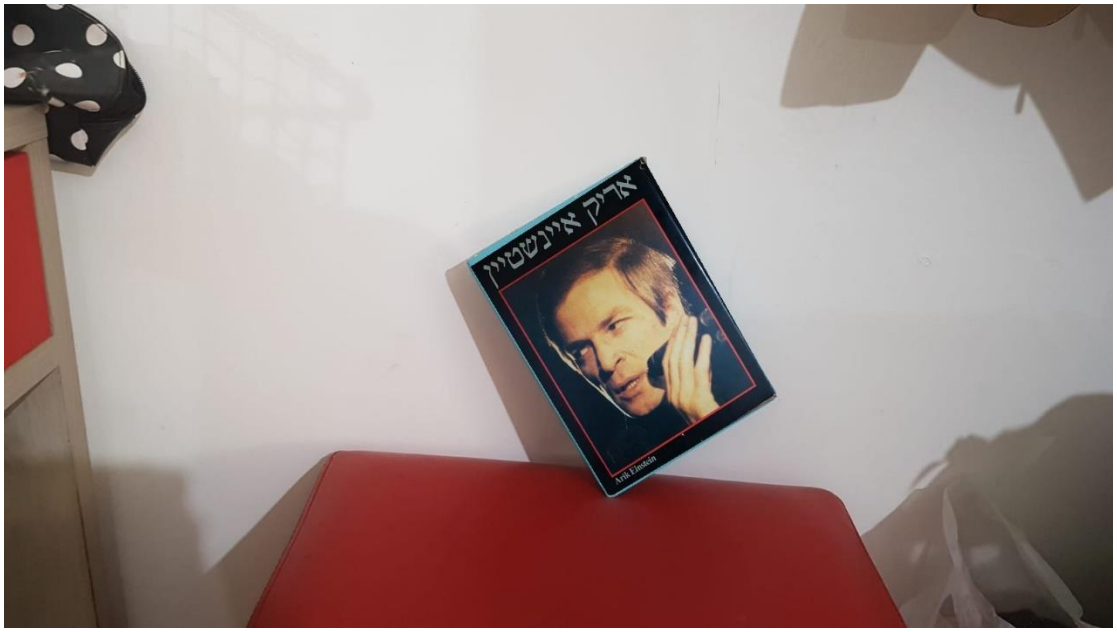
ביצענו *warp* לתמונת ה *input*:





ולאחר מכן תפירה לתוך התמונה באמצעות הפונקציות שמימשנו לתפירה לפי המיפוי של הפינות. הפינות של תמונת הרקע הם פשוט לפי גודל תמונת הרקע:

עוד שתי תמונות:


אריק איינשטיין
Arik Einstein