מבוא לעיבוד ספרתי של תמונות

עבודה 5

מגישים:

איילה ראובן 314077033

ליאור עבדייב 206087611

שאלה 1- Mona Surfaliza:

Inroduction to SURF 1.1

(1) קראנו על הפונקציה detectSURFFeatures(I,Name,Value) כפי שהתבקשנו. הפונקציה מקבלת תמונה בסקאלה של אפורים ומזהה את ה-features המתאימים לName ול-Value שנבחרו ע"י אלגוריתם SURF. הפונקציה מחזירה SURFpoints שמחזיק את המידע של ה-features שנמצאו.

הפרמטרים האופציונליים של הפונקציה:

- **'MetricThreshold':** ערך הסף של ה-feature החזק ביותר. ערכו הוא אי-שלילי וערך הדיפולט שלו הוא 1000. על מנת לקבל יותר blobs נקטין את הערך.
- **'NumOctaves':** הערך של מספר האוקטבות. ערכו הוא סקלר גדול או שווה ל- 1 והערך הדיפולטי שלו הוא 3. עבור ערך גדול יותר נקבל blobs גדולים יותר. (הערכים המומלצים עבורו הם 1-4).
- **'NumScaleLevels':** הערך של מספר הרמות בכל אוקטבה. סקלר שלם גדול או שווה ל3 והערך הדיפולטי שלו הוא 4. הגדלת מספק הרמות תוביל למציאת blobs רבים יותר בקנה מידה הרצוי.
- **'ROI-Rectangular region of interest':** הפונקציה תקבל מלבן בו יתבצע החיפוש של הfeatures. ערכי המלבן נתונים בצורת הוקטור $[x\ y\ width\ height]$ כאשר הערכים x,y מצביעים על הפינה השמאלית העליונה של המלבן והערכים $width\ height$ הם מימדי המלבן. המלבן הדיפולטי יהיה התמונה כולה.

(2) קראנו על הפונקציה extractFeatures(I,points) כפי שהתבקשנו. הפונקציה מקבלת תמונה וpoints . מטרת הפונקציה היא לחלץ וקטור המכיל את הfeatures שמצאנו מתוך האובייקט points. הפונקציה מחזירה שני פרמטרים- $[features, validPoints]$ שמחזיקים את features ואת המיקומים המתאימים שלהם.

(3) קראנו את התמונה של המונה ליזה, העברנו אותה לסקאלה של אפורים ונרמלנו אותה.

התוצאה:

(4) התבקשנו למצוא את כל הfeatures של התמונה ע"י שימוש בפונקציה detectSURFFeatures(I), למדוד כמה זמן לוקח לפונקציה למצוא את כל הפיטצרים ולהציג את הפיטצרים שמצאנו על התמונה. חילצנו את המרכזים של הפיטצרים בעזרת הפונקציה extractFeatures(I,points).

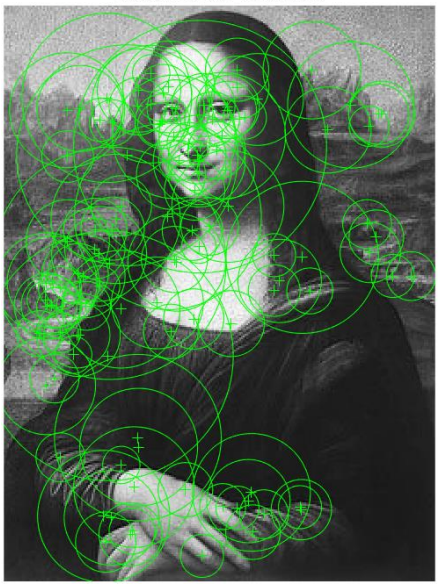
זמן הריצה:

```
Elapsed time is 0.006925 seconds.
```

סה"כ קיבלנו **111** features.

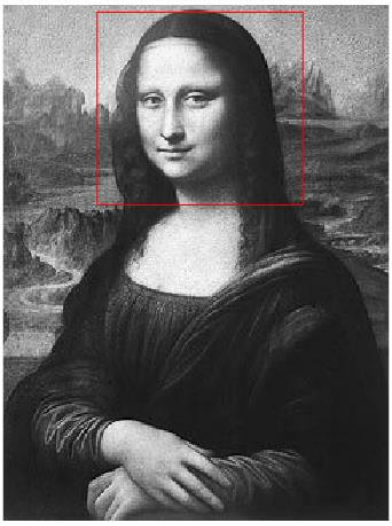
הצגנו את התמונה ועליה את הfeatures כפי שהתבקשנו. התוצאה:

Q1.1.4: SURF Features



(5) כעת התבקשנו למצוא את הfeatures בתמונה עבור '**ROI**' בעל הערכים: [59 5 128 120].
התבקשנו להציג את ה'**ROI**' על התמונה. התוצאה:

Q1.1.5: Present the selected 'ROI'

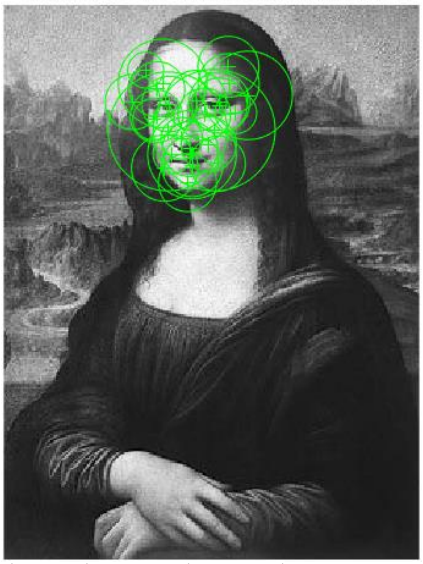


כלומר בסעיף זה נחפש את הfeatures שבתוך המלבן בלבד- כלומר את הפיטצרים שנמצאים באיזור הפנים.
זמן הריצה:

```
Elapsed time is 0.003107 seconds.
```

נמצאו **26** features.
הצגנו את הפיטצרים שמצאנו על גבי התמונה, התוצאה:

Q1.1.5: SURF Features with ROI



כלומר כאשר הגבלנו את גילוי הפיטצרים לפנים של המונה ליזה בלבד זמן הריצה קטן פי יותר מ2 וזיהה בערך רבע מכמות הפיטצרים שמצאנו בסעיף הקודם. בעצם קיבלנו שיש trad-off בין זמן הריצה לכמות הפיטרצים שאנחנו מחפשים.

משהו מעניין נוסף שאפשר לראות בתוצאות זה שהפיטצרים שהאלגוריתם נמצאים כולם על הפרצוף. כלומר האלגוריתם התעלם מהרקע (למרות שהמלבן כן מכיל חלק מהרקע), זאת מכיוון שהלאגוריתם מזהה את נקודות העניין בתחום החיפוש ומתעלם מהרעשים- במקרה הזה- הרקע .

(6) התבקשנו למצוא את הפיטצרים בתמונה ע"י שימוש בשני ערכים שונים של **'NumOctaves'** ולהשוות את הזמן ריצה שלהם לזמן ריצה של הפונקציה עם הערכים הדיפולטים.
הרצנו עבור **'NumOctaves'=1** ועבור **'NumOctaves'=2**.
עבור **'NumOctaves'=1** :

```
Elapsed time is 0.065579 seconds.
```

נמצאו **72** features.

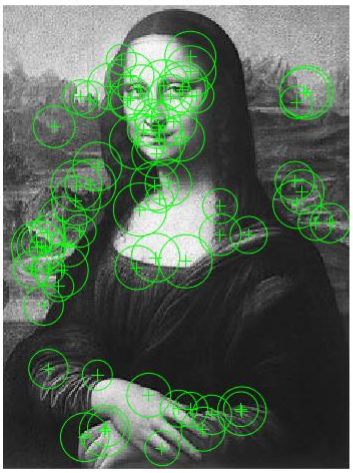
עבור **'NumOctaves'=2** :

```
Elapsed time is 0.008597 seconds.
```

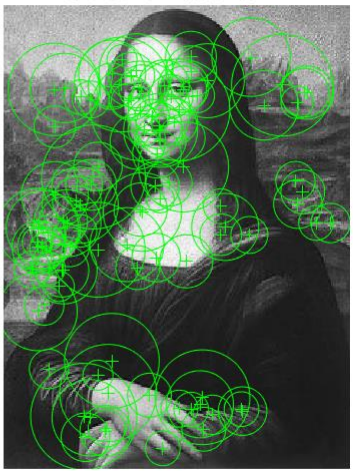
נמצאו **103** features.

הצגנו את הfeatures שקיבלנו על גבי התמונות. התוצאות :

Q1.1.6: NumOctaves = 1



Q1.1.6: NumOctaves = 2



הסבר על התוצאות : אכן ניתן לראות שהblobs (גודל העיגולים) קטן יותר עבור אוקטבה אחת מאשר 2 אוקטבות או 3 אוקטבות.
בנוסף קיבלנו שככל שמספר האוקטבות גדול יותר, זמן הריצה קצר יותר. זאת מכיוון שככל שמספר האוקטבות גדול יותר, כך נסרוק את התמונה מהר יותר. (עבור ערך הדיפולט קיבלנו את זמן הריצה הקצר ביותר)

בנוסף נשים לב שעבור ערך נמוך קיבלנו פחות פיטצרים. תוצאות אלה הן הגיוניות מכיוון שככל שמספר האוקטבות גבוה יותר, כך נחפש בסקאלות גבוהות יותר לכן כמות הפיטצרים תהיה גבוהה יותר.
קיבלנו במקרה הזה treade-off של דיוק מול כמות וזמן ריצה - ככל שמספר האוקטבות יותר קטן הפיטצרים שנצליח למצוא יהיו עדינים וממוקדים יותר אך זמן הריצה יהיה ארוך יותר וסך הפיטצרים שנמצא יהיה קטן יותר.
(7) התבקשנו למצוא את הפיטצרים בתמונה ע"י שימוש בשני ערכים שונים של **'NumScaleLevels'** ולהשוות את הזמן ריצה שלהם לזמן ריצה של הפונקציה עם הערכים הדיפולטים.
הרצנו עבור **'NumScaleLevels'**=3 ועבור **'NumScaleLevels'**=5. התוצאות:
עבור **'NumScaleLevels'**=3:

```
Elapsed time is 0.006846 seconds.
```

נמצאו **56** features.
עבור **'NumScaleLevels'**=5:

```
Elapsed time is 0.008864 seconds.
```

נמצאו **135** features.

הצגנו את הfeatures שקיבלנו על גבי התמונות. התוצאות:

הסבר על התוצאות: נשים לב שעבור מספר רמות גדול יותר זמן הריצה ארוך יותר ונמצאו יותר פיטצרים. זאת מכיוון שהאלגוריתם מחפש פיטצרים ביותר רמות. כלומר כמות החישובים גדלה ולכן לוקח לו זמן רב יותר לעבור על כל התמונה בכל הרמות. מסיבה זאת הוא גם מוצא מספר רב יותר של פיטצרים כי הוא מחפש ביותר רמות.
הtrade-off במקרה הזה שיש קשר בין כמות הfeatures שמצאנו לזמן הריצה של הפונקציה - עבור מספר גבוה יותר של רמות נמצא יותר features אך זמן הריצה יהיה ארוך יותר.

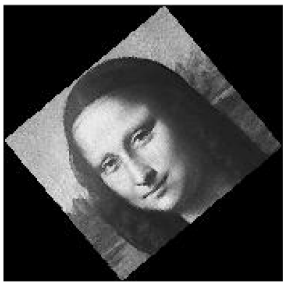
Make Mona Straight Again 1.2

(1) קראנו את שתי התמונות של מונה הישרה ומונה העקומה. נרמלנו והעברנו לסקאלה של אפורים.

התמונות :

Q1.2.1: Mona Liza Straight

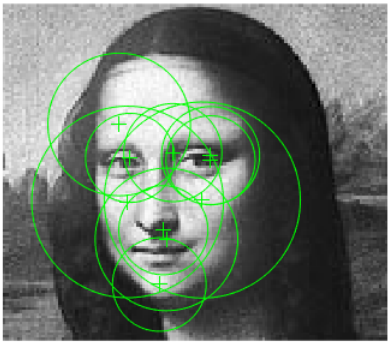
Q1.2.1: Mona Liza Crooked



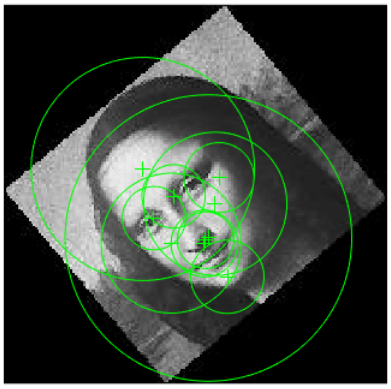
(2) התבקשנו לחלץ SURF feature points לכל תמונה ולהציג את עשר הפיצרים החזקים ביותר.

התוצאות :

Q1.2.2: 10 SURF feature

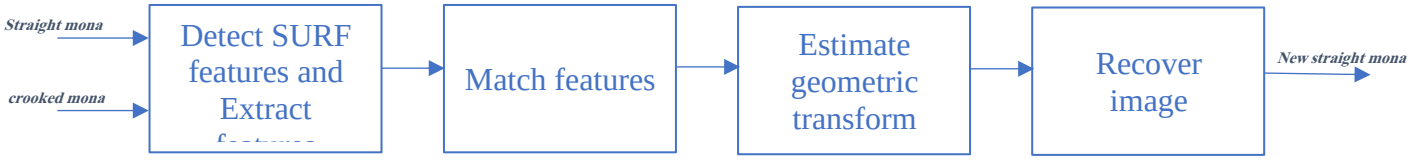


Q1.2.2: 10 SURF feature



אפשר להבחין שעשרת הפיטצרים החזקים ביותר זהים פחות או יותר בין שתי התמונות. כלומר עשרת הפיטצרים החזקים ביותר כמעט ולא מושפעים מהסיבוב.

(3) התבקשנו ליישר את התמונה המסובבת. נסביר את האלגוריתם לפיו סיבבנו את התמונה ע"י דיאגרמת בלוקים ולאחר מכן נסביר כל שלב באלגוריתם בנפרד :



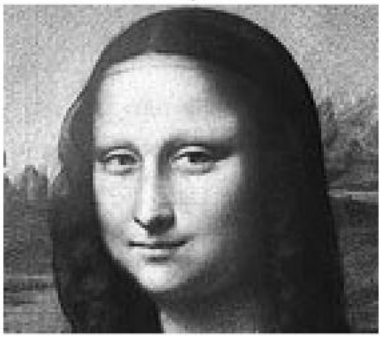
- בשלב הראשון נמצא ונחלץ את הfeatursה של כל אחת מהתמונות בנפרד. את התוצאות ניתן לראות בסעיף הקודם.
- בשלב השני נמצא את הקואורדינטות של הfeatures המתאימים בין שתי התמונות ע"י הפונקציה matchFeatures. הצגנו את הזוגות שזוהו. התוצאה (בעמוד הבא) :

Q1.2.3: Mona Matched SURF Points

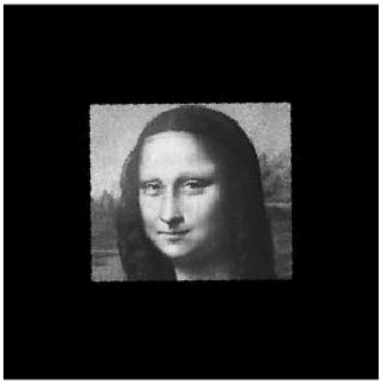
- בשלב השלישי עזרת מיקומי הקוארדינטות שמצאנו ננסה להעריך מה הטרנספורמציה הגאומטרית שקיבלה את מונה הישרה והוציאה את מונה המסובבת. נעשה זאת ע״י שימוש בפונקציהestimateGeometricTransform . (השתמשנו trabsformType='similarity' שקובע שיש לזהות לפחות שתי זוגות תואמים.)
- בשלב האחרון ביצענו את הטרנספורמציה שמצאנו לתמונה המסובבת בעזרת הפונקציה imwrap על מנת לקבל את התמונה המסובבת ישרה.

התוצאה :

Q1.2.3: Original Mona



Q1.2.3: Recovered Mona



הסבר : אכן הצלחנו לקבל את מונה ישרה כמו שרצינו.
למרות שהגבולות של התמונה נראים עדיין עם שיפוע קטן אנחנו מרוצים מהתוצאה מכיוון שאם מסתכלים על הפנים של מונה אפשר לראות שהם ישרים ובכיוון התמונה המקורית.
כלומר האלגוריתם שלנו אכן יישר את התמונה לפי הפנים של התמונה המקורית והצלחנו לשערך נכון את הטרנספורמציה הגאומטרית.

Fake or Real 1.3

קראנו את התמונה פנים הישרה של המונה ליזה (straight_mona). קראנו את כל התמונות של המונה ליזה מהתיקיה. כתבנו אלגוריתם ש קובע האם התמונה מכילה את הפנים האמיתיות של המונה הליזה.
עבור כל תמונה האלגוריתם מדפיס אם התמונה אמיתית או מזוייפת.
נסביר את האלגוריתם, נציג את התמונות שמצאנו, לכל תמונה נראה את הפילטרים שמתאימים לו עם התמונה המקורית וננתח את התוצאות.

האלגוריתם לפיו נקבע אם התמונה אמיתית או מזוייפת:

הסבר:
תחילה נמצא ונחלץ את הfeatures של תמונת הפנים של המונה המקורית.
עבור כל תמונה:

- נמצא ונחלץ את הfeatures.
- בשלב השני נמצא את הקואורדינטות של הfeatures המתאימים בין התמונה לפנים של מונה המקורית ע"י הפונקציה matchFeatures. נרצה למצוא Features הקרובים ביותר. לכן הערכים שנכנסים לפונקציה הם:

'MatcheThreshold'=5

'MaxRatio'=0.7

'Metric'='SAD'

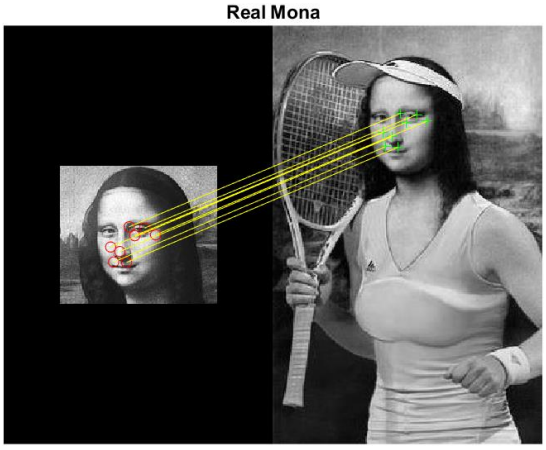
- אם הצלחנו למצוא features מתאימים, נקבע כי התמונה מכילה את הפנים האמיתיות של המונה ליזה והיא אמיתית. אם לא הצלחנו למצוא features מתאימים, נקבע כי התמונה לא מכילה את הפנים האמיתיות של המונה ליזה והיא מזויפת.

התוצאות:

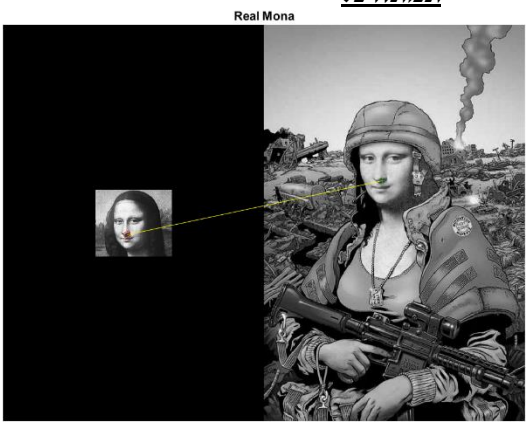
<u>תמונה 1:</u>

Output= `Mona1_Y.jpg is Real Mona`

עבור התמונה Mona1_Y:
זיהינו את מונה האמיתית.
האלגוריתם צודק.
ניתן לראות שהאלגוריתם הצליח למצוא יחסית הרבה features מתאימים בין שתי התמונות.



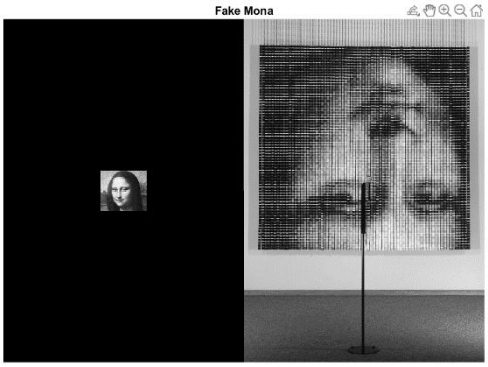
תמונה 2:



Output= `Mona2_Y.jpg is Real Mona`

עבור התמונה Mona2_Y:
זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם הצליח למצוא feature אחד מתאים בין שתי התמונות.

תמונה 3:



Output= `Mona3_Y.jpg is Fake Mona`

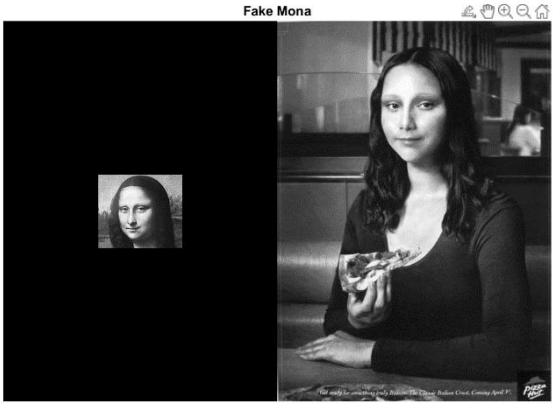
עבור התמונה Mona3_Y:
לא זיהינו את מונה האמיתית.
האלגוריתם טועה.
במקרה הזה האלגוריתם לא הצליח למצוא feature מתאימים בכלל בין שתי התמונות. כנראה שהגריד האלגוריתם התקשה למצוא פיטצרים מתאימים בן שתי התמונות ולכן התמונה לא זוהתה כמונה אמיתית.

תמונה 4:

Output= `Mona4_Y.jpg is Fake Mona`

עבור התמונה Mona4_Y:
לא זיהינו את מונה האמיתית.
האלגוריתם טועה.
במקרה הזה האלגוריתם לא הצליח למצוא feature מתאימים בכלל בין שתי התמונות. כנראה שבגלל הגוון של התמונה שונה ממש משל התמונה המקורית, והסקאלה של התמונה שונה משל התמונה המקורית, האלגוריתם התקשה לזהות את המונה ליזה בתוך התמונה.

תמונה 5:



Output= `Mona5_N.jpg is Fake Mona`

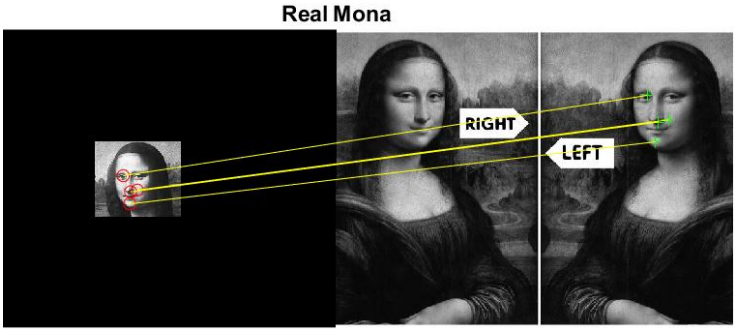
עבור התמונה Mona5_Y:
לא זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם לא הצליח למצוא features מתאימים בין שתי התמונות.

תמונה 6:

Output= `Mona6_N.jpg is Fake Mona`

עבור התמונה Mona6_Y:
לא זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם לא הצליח למצוא
features מתאימים בין שתי התמונות.

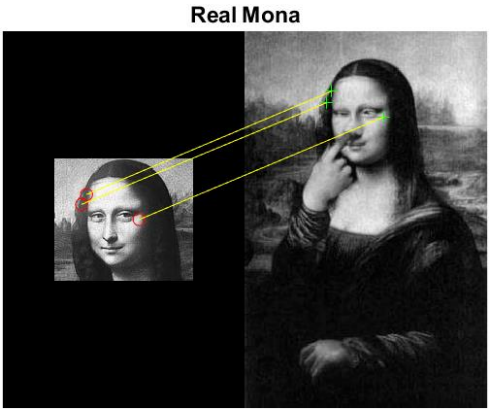
תמונה 7:



Output= `Mona7_Y.jpg is Real Mona`

עבור התמונה Mona7_Y:
זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם הצליח למצוא
features מתאימים בין שתי התמונות.

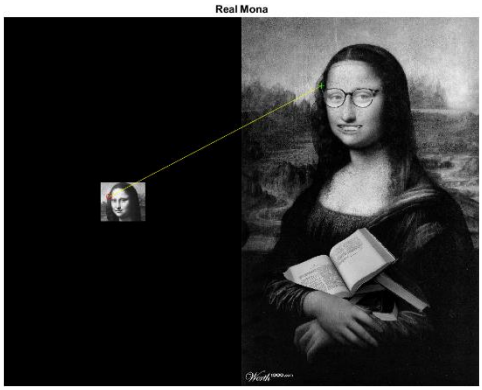
תמונה 8:



Output= `Mona8_Y.jpg is Real Mona`

עבור התמונה Mona8_Y:
זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם הצליח למצוא features מתאימים בין
שתי התמונות.

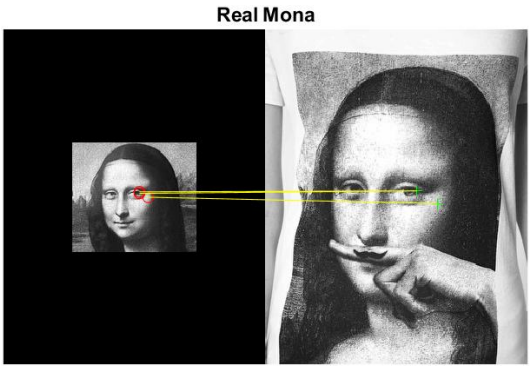
תמונה 9:



Output= `Mona9_Y.jpg is Real Mona`

עבור התמונה Mona9_Y:
זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם הצליח למצוא feature אחד מתאים
בין שתי התמונות.

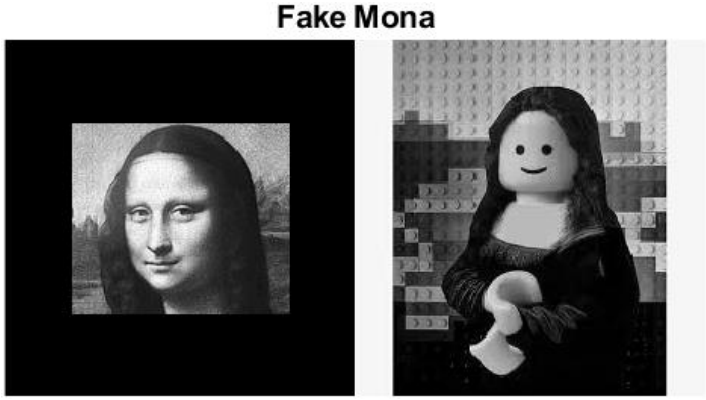
תמונה 10:



Output= `Mona10_Y.jpg is Real Mona`

עבור התמונה Mona10_Y :
זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם הצליח למצוא features מתאימים בין שתי התמונות.

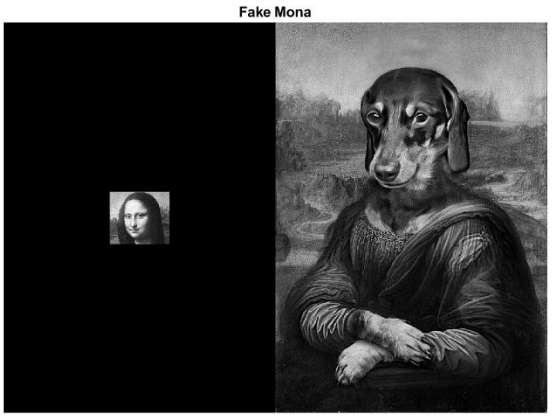
תמונה 11



Output= `Mona11_N.jpg is Fake Mona`

עבור התמונה Mona11_Y:
לא זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם לא הצליח למצוא features מתאימים בין שתי התמונות.

תמונה 12:



Output= `Mona12_N.jpg is Fake Mona`

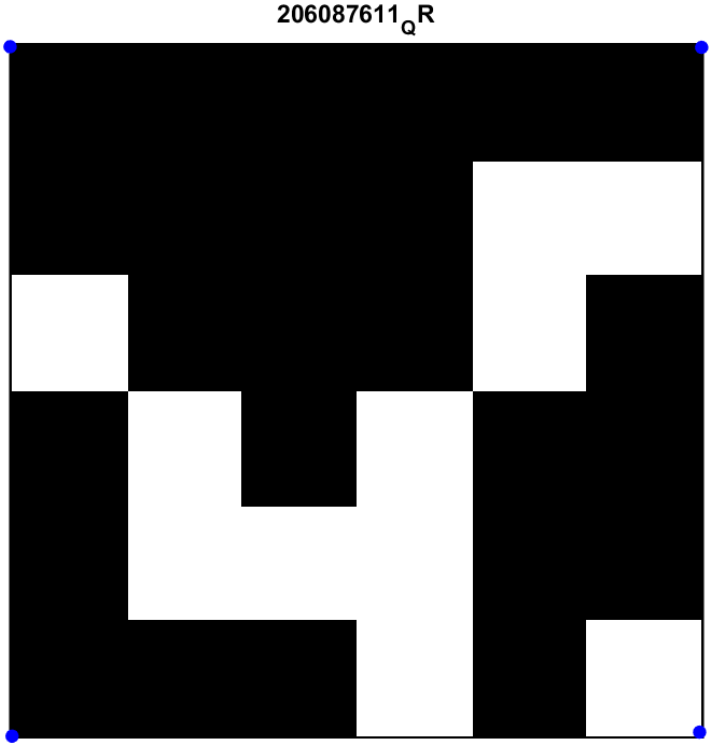
עבור התמונה Mona12_Y:
לא זיהינו את מונה האמיתית.
האלגוריתם צודק.
במקרה הזה האלגוריתם לא הצליח למצוא features מתאימים בין שתי התמונות.

הסבר: סה"כ קיבלנו שהאלגוריתם הצליח לנחש נכון עבור 10 מתוך 12 מקרים. הוא טעה בזיהוי המונה בתמונה עבור תמונות מספר 3 ו-4. בתמונות אלה הסקאלה של התמונות והגוונים של התמונות הקשו עליו בלזהות features מתאימים.
נשים לב שהטעויות רק בלזהות נכון כזיוף. לעומת זאת האלגוריתם הצליח לזהות את כל הזיופים.

שאלה 2 – QR CODE READER

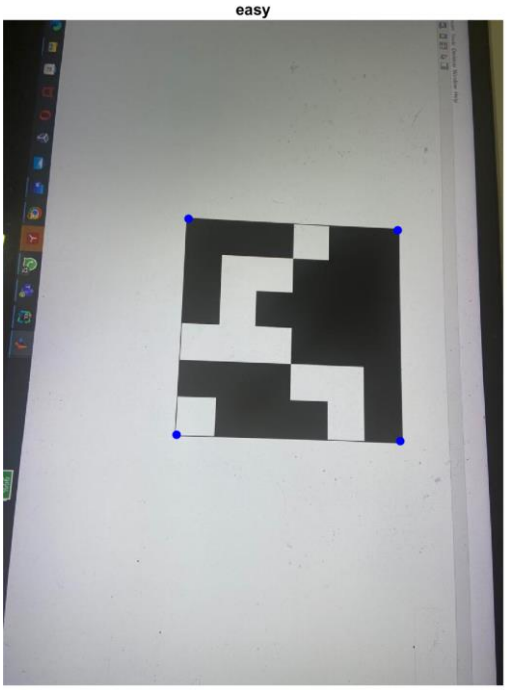
2.1.

הכנסנו את הת.ז "206087611" לפונקציית הQR וקיבלנו את התמונה הבאה:

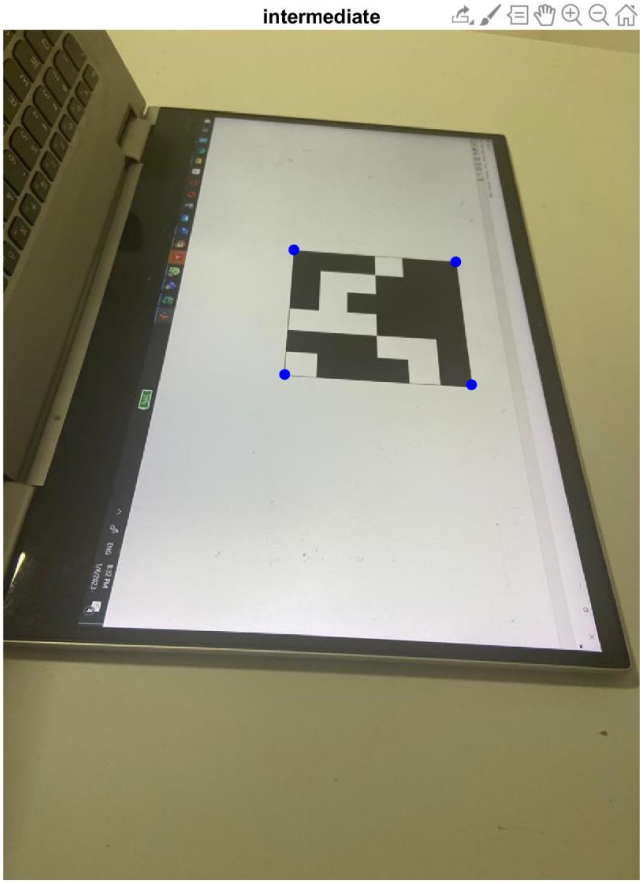


2.2-2.3.

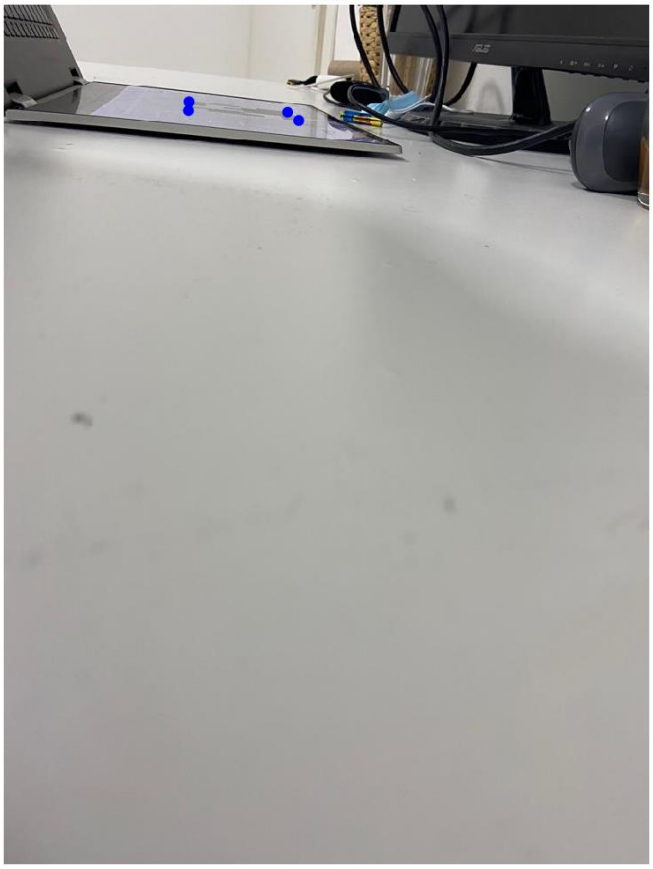
צילמנו את תמונת הQR מ3 זוויות שונות: קלה, בינונית וקשה ומיקמנו את הנקודות בפינות, קיבלנו את התמונות הבאות:



intermediate



hard



2.4.

ביצענו טרנספורמציה לתמונות בשלושת הדרכים בהן התבקשנו :

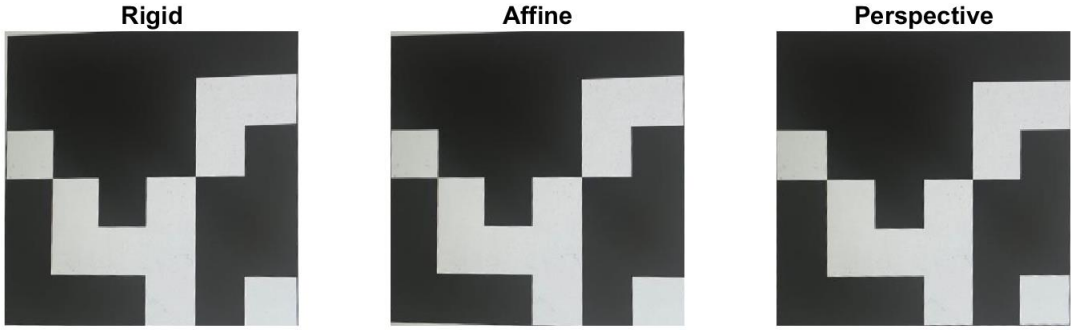
Rigid – אין עיוות לתמונה, המרחקים בין הנקודות בתמונה נשארים כמו שהם לאחר הסיבוב.

Affine – בניגוד לRigid פה כן מתבצע עיוות לתמונה, המרחקים בין הנקודות כן משתנים בטרנספורמציה זאת אבל היחסים בין המרחקים נשמרים. בנוסף יש שינוי גם בזויות של הקווים בתמונה, אבל צורות גיאומטריות לא משנות את צורתן.

Perspective – שיטה זו מיועדת לשינוי פרספקטיבה במרחב תלת מימדי לכדי יצירת תמונה דו מימדית. הטרנפורמציה יכולה ליצור שינוי בצורות ובזויות (למשל טרפז יכול להפוך לריבוע).

ביצענו טרנספורמציה לתמונות בשלושת הדרכים בהן התבקשו קיבלנו את הפלטים הבאים :

Easy :



Intermediate :

Hard :

2.5.

נסביר על התוצאות שקיבלנו :

קלה:

מכיוון שהזווית בין המצלמה לברקוד קטנה כל שלושת השיטות הצליחו לקרוא את הברקוד בצורה מצוינת. ההפרש הקטן בזוית יוצר מצב שבו ההפרש בין המרחק בין הנקודות בתמונה לבין המרחק האמיתי שלהן זניח ולכן מכיוון שבשיטת Rigid אין שינוי בין מרחקי הנקודות היא הצליחה לקרוא את הברקוד בצורה מעולה.

ההפרש הקטן בזויות לא הקשה על שתי הטרפנספורמציות האחרות להביא לתוצאות טובות.

בינונית:

כאן הגדלנו את הזווית בין המצלמה לברקוד וניתן לראות שעבור הטרנספורמציה Rigid כבר לא קיבלנו תוצאה מספקת, זה מכיוון שההגדלה בזווית יוצרת אשליה של מרחק קטן יותר בין צלעות בריבוע מה שהיא לא ידעה להתמודד איתו.

עבור הטרנספורמציה האפינית קיבלנו תמונה טובה, עם זאת ניתן לראות אי דיוק קטן בצד השמאלי, זאת מכיוון שהתמונה הבינונית נותנת אשליה שהברקוד מעט טרפזי וכמו שציינו אין שינוי בצורות בשיטה זאת. טרנספורמציית הפרספקטיבה יודעת להתמודד טוב עם האתגרים שעמדו בפני שתי הטרנספורמציות האחרות ולכן קיבלנו גם כאן תוצאה מצוינת.

קשה:

צילמנו תמונה קשה מאוד וקיבלנו תוצאות לא טובות, עם זאת הצלחנו לקבל תמונה שניתן לזהות בה את הברקוד ואף לחלץ את המטריצה הבינארית. הטרנספורמציה הראשונה לא הייתה בכיוון מאותה סיבה שלא הצליחה בתמונה הבינונית, גם כאן האפינית והפרספקטיבית קיבלו תוצאות טובות יותר ממנה עם עדיפות קלה לפרספקטיבית. יש לציין שנדרשו מספר נסיונות לקבוע נקודות על מנת לקבל תוצאה מספקת.

2.6-2.7.

חילצנו את המטריצה הבינארית של הקוד ע״י תמונה שעברה טרנספורמציה פרספקטיבית והמרנו את הערכים הבינאריים לערכים דצימליים שהם ספרות תעודת הזהות שלנו :

```
easy_mat = 6×6
     0     0     0     0     0     0
     0     0     0     0     1     1
     1     0     0     0     1     0
     0     1     0     1     0     0
     0     1     1     1     0     0
     0     0     0     1     0     1

id_easy = 1×9
     2     0     6     0     8     7     6     1     1

inter_mat = 6×6
     0     0     0     0     0     0
     0     0     0     0     1     1
     1     0     0     0     1     0
     0     1     0     1     0     0
     0     1     1     1     0     0
     0     0     0     1     0     1

id_inter = 1×9
     2     0     6     0     8     7     6     1     1

hard_mat = 6×6
     0     0     0     0     0     0
     0     0     0     0     1     1
     1     0     0     0     1     0
     0     1     0     1     0     0
     0     1     1     1     0     0
     0     0     0     1     0     1

id_hard = 1×9
     2     0     6     0     8     7     6     1     1
```

ניתן לראות שעבור שלושת התמונות הצלחנו לחלץ את המטריצה הבינארית ולפענח את הקוד ואכן לקבל את המספר שהכנסנו ״206087611״.

2.9

שמרנו את הערכים האופקיים של הפינות כ"ציר X" והערכים האנכיים של הפינות כ"ציר Y":

נעתיק אותם גם לכאן:

```
hard_x = 4×1
    338.9379
    220.9353
    221.5909
    352.0493

inter_x = 4×1
    546.7535
    350.7382
    339.5935
    565.7650

easy_x = 4×1
    606.4104
    285.1812
    266.8252
    610.3438

hard_y = 4×1
    128.6639
    114.8969
    125.3860
    138.4974

inter_y = 4×1
    280.1005
    265.6780
    415.8035
    428.2593

easy_y = 4×1
    324.0237
    306.3233
    638.6972
    648.5307
```

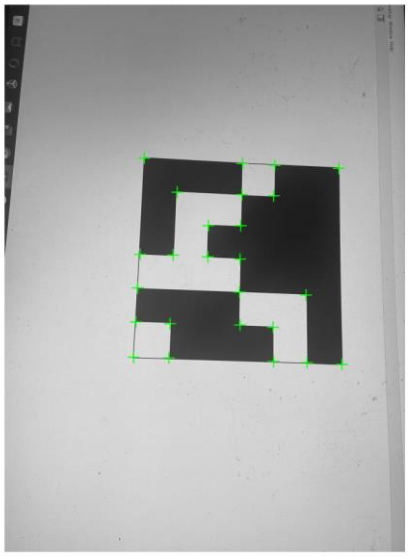
שאלה 3- Automatic corner detector

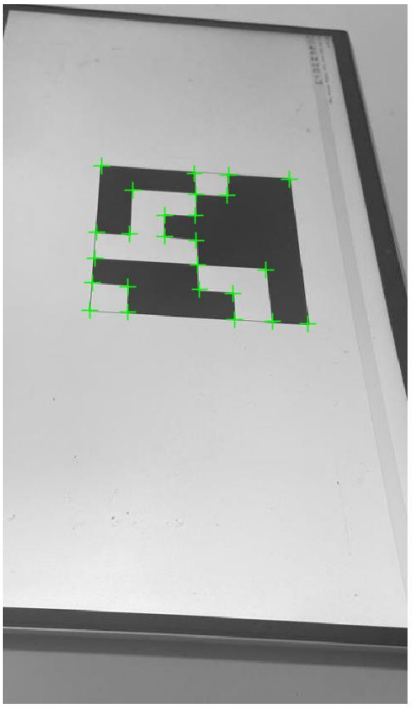
.3.2

השתמשנו באלגוריתם "detectHarrisFeatures" על מנת למצוא את הפינות, אלגוריתם זה משתמש במסנן גאוסי על מנת לנקות רעשים (שעלולים להיראות כפינות) ולאחר מכן מבצע נגזרת בציר הY ובציר הx, וכך מזהה שינויים חדים כפינות.

מכיוון שהשתמשנו בתמונה שהיא על מסך לפטופ (לצערנו אין לנו מדפסת ולא רצינו שזה מה שיעכב את ביצוע העבודה), פינות המחשב ותיבת התוכנות הוסיפו פינות שהפריעו לנו ולכן מעט חתכנו את האיזורים האלה כדי שלא יפריעו יתר על המידה.

נפעיל את האלגוריתם על התמונות ונקבל את התוצאות הבאות:

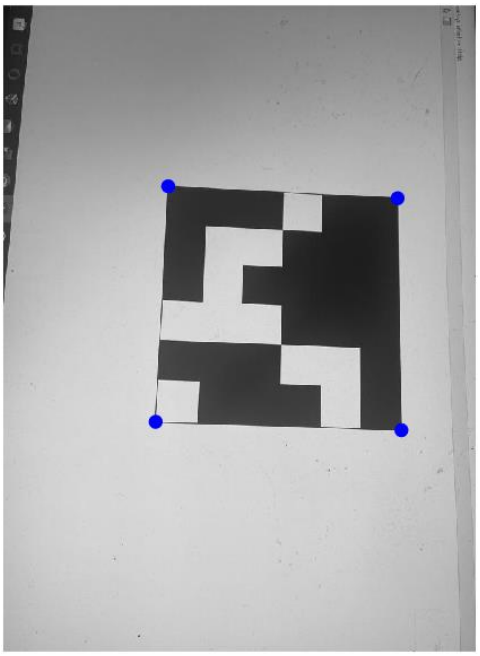




לאחר ביצוע האלגוריתם קיבלנו הרבה פינות (קוד הQR לא מכיל רק את הפינות החיצוניות מכיוון שכל כולו מורכב ממלבנים) ראשית האלגוריתם מאפשר להשתמש ב"k" הפינות החזקות ולכן צמצמנו כך את כמות הפינות הלא רלוונטיות שהאלגוריתם אפשר ולאחר מכן על ידי פעולות של מציאת מקסימום ומינימום של סכומים/הפרשים הצלחנו לחלץ את הפינות הנדרשות(סכום מינימלי בין ערכי x y של הפינות, סכום מקסימלי, הפרש מינימלי, הפרש מקסימלי)

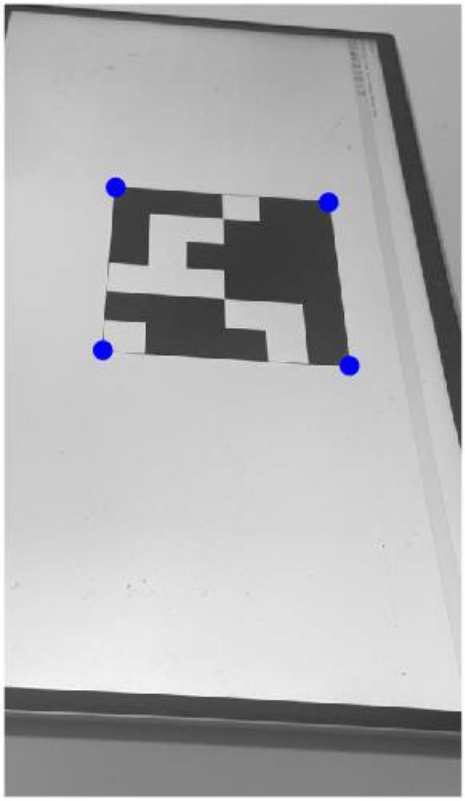
לאחר ביצוע סינון הפינות קיבלנו את הפינות הבאות :

עבור התמונה הקלה :



```
easy_corner = 4×2
    558.8690  273.5657
    234.0752  256.5859
    216.2632  590.1691
    564.3306  602.0593
```

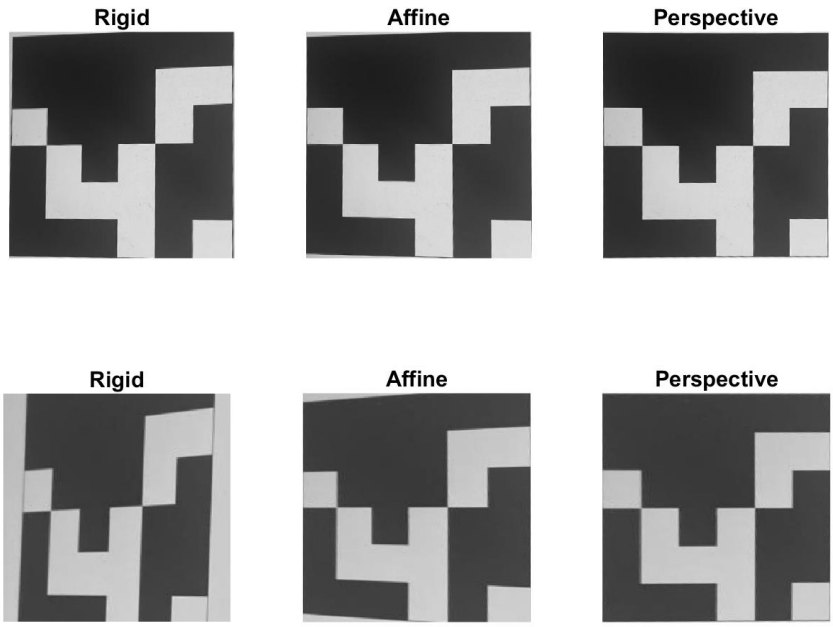
עבור התמונה הבינונית:



```
inter_corner = 4×2
    297.6828  181.8558
    102.5850  168.1754
     90.6951  317.0432
    316.8807  331.2986
```

התמונה הקשה הייתה קשה מידי ולא הצלחנו לחלץ את הפינות של הברקוד.

לאחר מכן המשכנו בסעיפים של שאלה 2, סובבנו את התמונה והצלחנו לקבל תמונות דומות לתמונות בשאלה 2 עבור שתי רמות הקושי, עם זאת בתמונות בהן זיהינו פינות באמצעות אלגוריתם ולא למראית העין קיבלנו אף תמונות עדיפות מהתמונות בשאלה 2:



עם תמונות כאלה אי אפשר שלא להצליח לחלץ את הקוד, הכנסנו את תמונת הperspective״ לפונקציה QR2ID שבנינו בשאלה 2 ומצאנו גם כאן תעודת הזהות.

```
easy_mat2 = 6×6
       0     0     0     0     0     0
       0     0     0     0     1     1
       1     0     0     0     1     0
       0     1     0     1     0     0
       0     1     1     1     0     0
       0     0     0     1     0     1

id_easy2 = 1×9
       2     0     6     0     8     7     6     1     1

inter_mat2 = 6×6
       0     0     0     0     0     0
       0     0     0     0     1     1
       1     0     0     0     1     0
       0     1     0     1     0     0
       0     1     1     1     0     0
       0     0     0     1     0     1

id_inter2 = 1×9
       2     0     6     0     8     7     6     1     1
```