INTRODUCTION TO DIGITAL IMAGE PROCESSING

361.1.4751

EXERCISE 5 - Feature Descriptors and Geometric Transformations

Submission Date: 11.01.2023

Noam Atias – 311394357

Chanel Michaeli – 208491787

תוכן עניינים

1. Mona Surfaliza *3*

1.1 Introduction to SURF **3**

1.1.1 Read about the MATLAB function detectSURFFeatures(I,Name,Value) 3

1.1.2 Read about the MATLAB function extractFeatures(I,points). 4

1.1.3 Reading the image and convert it to grayscale normalized image. 4

1.1.5 Find features in the image using 'ROI', [59, 5, 128, 120]. 5

1.1.6 Find features in the image using 2 different *'NumOctaves'* 7

1.1.7 Find features in the image using 2 different *'NumScaleLevels'* 8

1.2 Make Mona Straight Again **10**

1.2.1 Reading the images 'straight_mona.PNG' and 'crooked_mona.jpg' 10

1.2.2 Extract the SURF feature points of each of the images and display the ten strongest features 10

1.2.3 Using the extracted feature points to straighten Mona's face 11

1.3 Fake or Real **12**

1.3.1 Reading the image *'mona_org.jpg'* and detect images with real Mona Liza face exist 12

1.3.2 Display the images we found 12

1.3.3 Explaining the algorithm using a block diagram 17

1.3.4 Analyze the results 18

2. QR Code Reader *19*

2.1 Insert (311394357) ID to the attached function ID2QR(id) **19**

2.2 three photos of the QR code from three different angles **19**

2.3 Locate the points on the QR code – manually **21**

2.4 Transform the image such that the QR code is straightened **22**

2.5 Explain the result for each transformation **23**

2.6 Extract the binary values **24**

2.7 Convert every 4 bits to an integer **24**

2.8 Verified the correct ID **25**

3. Automatic Corner Detector *26*

1. Mona Surfaliza

1.1 Introduction to SURF

1.1.1 Read about the MATLAB function detectSURFFeatures(I,Name,Value).

בסעיף זה התבקשנו לקרא על פונקציית ה – MATLAB: *'detectSURFFeatures (I , Name , Value)'* ולהסביר על הפרמטרים אותה היא מקבלת ואת ערכי ברירת המחדל של אותם פרמטרים.

פונקציה זו מזהה את מאפייני התמונה ומחזירה את הפיצ'רים שחולצו מהתמונה. ניתן הסבר על כל הפרמטרים של הקלט אותם הפונקצייה מקבלת:

- $I - Input\ Image - Gray\ Scale\ Image$: פרמטר זה מקבל את התמונה בגווני אפור.
- $Name\ ,Value$: שם הארגומנט / פרמטר והערכים שמאפיינים של הפרמטר:
  - $MetricThreshold$: ערך הסף הקובע מי הם הפיצ'רים החזקים ביותר, כלומר מבטא את ערכי הסף של מאפייני ה – SURF. ככל שנוריד את הסף נקבל יותר נקודות וההפך. ברירת המחדל עבור ערך זה הינו $Value = 1000$. (ערך זה מקבל ערכים סקלרים אי – שליליים).
  - $NumOctaves$: מקבל סקלר גדול או שווה ל – 1. מציין את כמות האוקטבות עבור אלגוריתם זה עם הגדלת הערך נסתכל על רזולוציות יותר גדולות, כלומר גודל הפילטר יהיה גדול יותר. כל אוקטבה מכילה סדרה של פילטרים בגדלים שונים, כאשר כמות הפילטרים תקבע על ידי $NumScaleLevels$ לדוגמא:

Octave	Filter Sizes
1	9-by-9, 15-by-15, 21-by-21, 27-by-27, ...
2	15-by-15, 27-by-27, 39-by-39, 51-by-51, ...
3	27-by-27, 51-by-51, 75-by-75, 99-by-99, ...
4	....

צורת ברירת המחדל עבור פרמטר זה הינו $Value = 3$, כאשר מומלץ לבחור ערך בין 1 ל – 4.

  - $NumScaleLevels$: מספר הסקאלות שדרוש לחבר בכל אוקטבה. גודל זה יקבע כמה פילטרים יופעלו מתוך אחת השורות שבטבלה בהתאם ל – $NumOctaves$. ערכו של פרמטר זה צריך להיות גדול או שווה ל – 3 , כאשר הערכים המומלצים הינם בטווח של $3 - 6$. ערך ברירת המחדל עבור פרמטר זה הינו $Value = 4$.
  - $'ROI'$: אזור עניין מלבני. כאשר מציינים ROI מסוים, הפונקציה מזהה פינות בתחום המתאים. ערך זה מקבל וקטור של ערכים $[x, y, Wigth, Height]$ ומציין אזור שמעניין אותנו בתמונה שהכנסנו, כלומר הפרמטרים (x, y) מציינים את הקואורדינטות של הפינה השמאלית שעליונה של תחום העניין המלבני, והפרמטרים $[Wigth, Height]$ מציינים את רוחב ואורך המלבן בהתאמה. ערך ברירת המחדל של פרמטר זה הוא הגודל המקורי של התמונה, כך:

$$Value = [1,1, size(I,1), size(I,1)]$$

דוגמא לשימוש בפונקציה:

$$detectSURFFeatures(\mathrm{I}, '\mathrm{MetricThreshold}', 100)$$

1.1.2 Read about the MATLAB function extractFeatures(I,points).

בסעיף זה התבקשנו לקרוא על פונקציית ה – MATLAB: *'extractFeatures (I , points)'* ובנוסף להסביר מה מטרת הפונקציה הזו ועל ערכי ה – Points שלה.

מטרת הפונקציה היא לקבל נקודות עניין (Points) שהן פלט של אחד מהאלגוריתמים למציאת פיצ'רים (לדוגמא: SURF, BRISK), המסמנים היכן הפיצ'רים נמצאים בתמונה. הפונקציה לוקחת את נקודות אלו ובונה לכל נקודה descriptor אשר מייצג את הפיצ'ר ומחזירה את הפיקסל המרכזי של ה – descriptor ומאפיינים נוספים באותו פורמט של Points שהוכנסו כקלט.

1.1.3 Reading the image and convert it to grayscale normalized image.

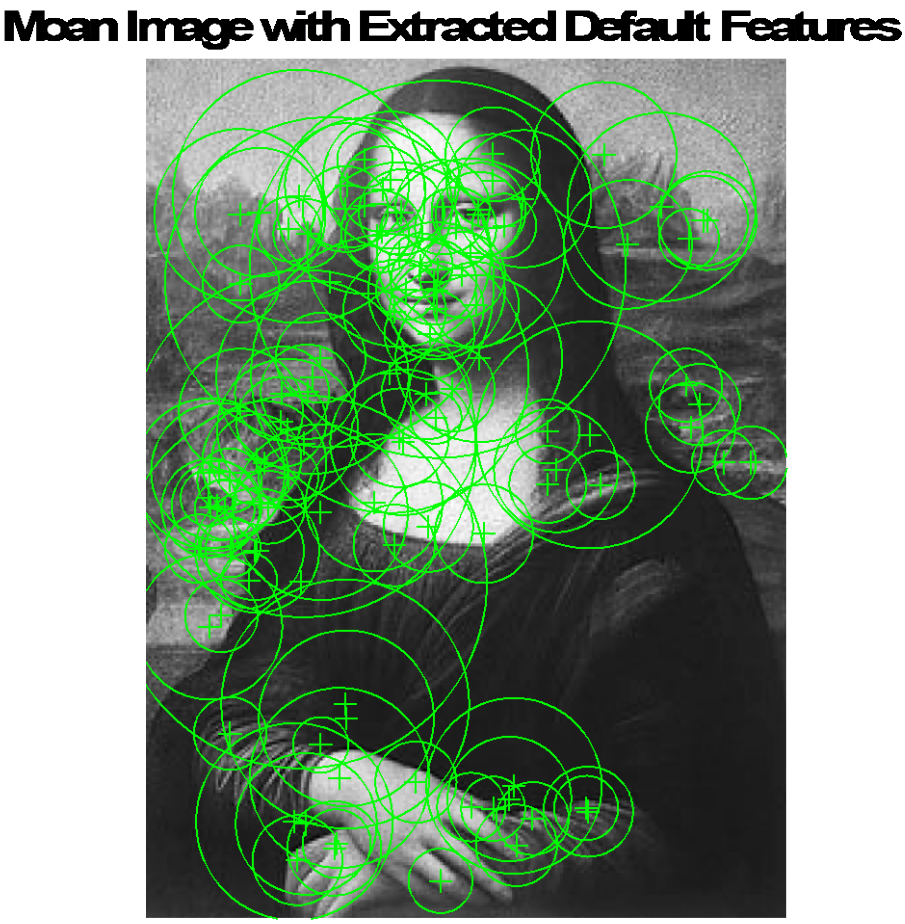
נקרא את התמונה *'mona_org.jpg'* ונמיר אותה לתמונת Gray Scale מנורמלת. נציג את התמונה שקיבלנו:

Normalized Mona Image

1.1.4 Find all the features in the image.

בסעיף זה נחלץ את הפיצ׳רים על ידי שימוש בפונקציה *'detectSURFFeatures (I)'* ונציג את התוצאות על גבי התמונה המקורית, ונמדוד את זמן הריצה של הפונקציה. נשים לב כי במקרה הזה כל הפרמטרים שתוארו הסעיף 1.1.1 הוגדרו לפי ערכי ברירות המחדל. נשים לב שקיבלנו 111 פיצ׳רים שחולצו.

נציג את התוצאות שקיבלנו על התמונה:



מבחינת זמני ריצה קיבלנו:

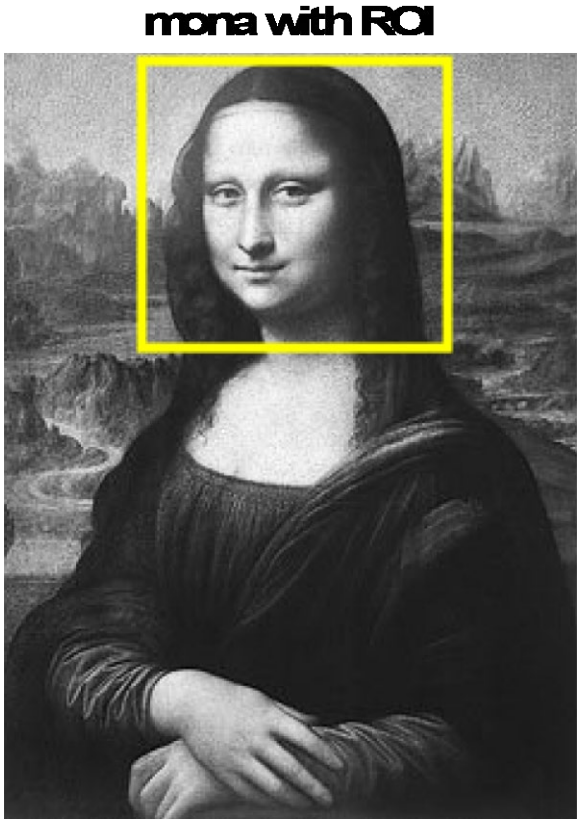
```
time1 =

    0.3193
```

נוכל לראות כי הפיצ׳רים מפוזרים על כל התמונה (האלגוריתם דואג לפיזור לאורכה ולרוחבה של התמונה וזאת כדי להימנע ממקבץ גדול סביב אזור מסוים ולאפשר זיהוי פיצ׳רים מכלל אזורי התמונה, אלא אם כן נדרשנו אחרת). בנוסף נוכל לראות מגוון של גדלים בפיצ׳רים השונים.

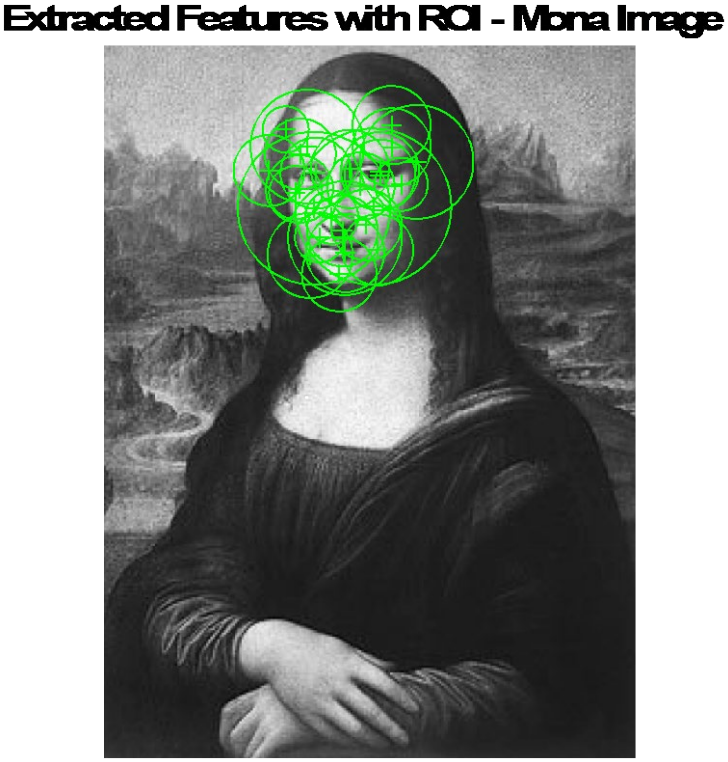
1.1.5 Find features in the image using ’ROI’, [59, 5, 128, 120].

בסעיף זה נפעיל את הפונקציה כמו בסעיף הקודם, אך כעת נשתמש בפרמטר 'ROI', כאשר נתון $[59, 5, 128, 120]$. כלומר נחפש פיצ׳רים אך ורק באזור המלבני הנ״ל, כאשר $(59, 5)$, רוחב המלבן

הוא $Wigth = 128$, ואורך הלבן הוא $Height = 120$. כלומר פרמטר זה מכוון את האלגוריתם לחלץ את הפיצ'רים מהמלבן המסומן בתמונה הבאה:



נציג את מציאת הפיצ'רים בתמונה באזור המבוקש:



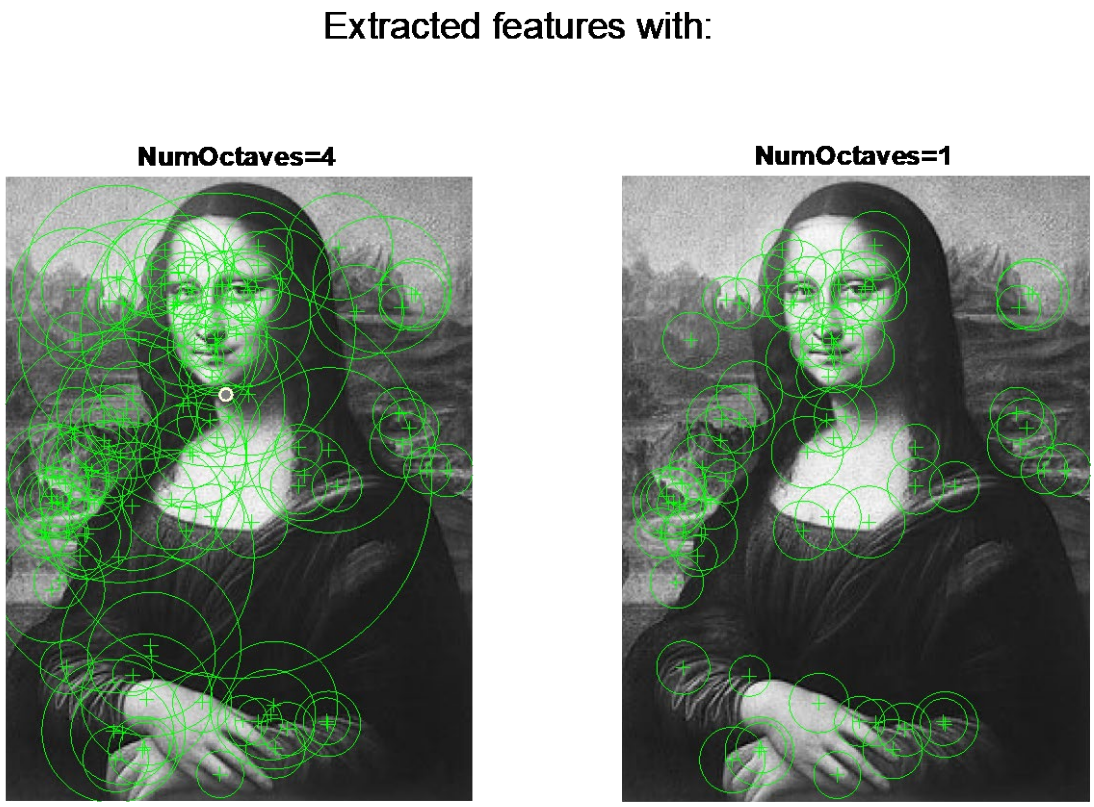
זמן הריצה למציאת הפיצ׳רים הנמצאים בתחום המבוקש הוא:

```
time2 =

0.0247
```

נשים לב שזמן הריצה קטן יותר (כצפוי), ומכיוון שכעת האלגוריתם עובד על אזור קטן יותר עם אותה כמות של אוקטבות, ואותה כמות של רמות. לכן האלגוריתם צריך לעבור על כמות פיקסלים קטנה יותר.
במקרה הקודם מצאנו 111 פיצ׳רים ובמקרה הזה מצאנו 26 פיצ׳רים רלוונטיים באזור המבוקש.
משום שלא שינינו את עוד פרמטרים, נצפה לאותה כמות פיצ׳רים שהאלגוריתם מצא בסעיף הקודם באזור המבוקש ולכן כמות הפיצ׳רים קטנה משמעותית.

1.1.6 Find features in the image using 2 different *'NumOctaves'*.

בדומה לשני הסעיפים הקודמים, כעת נפעיל את הפונקציה כאשר אנחנו משנים את הפרמטר *'NumOctaves'*. נציג את התוצאות ואת זמן ריצת האלגוריתם עבור 2 גדלים שונים של הפרמטר המבוקש, ונבצע השוואה לסעיפים הקודמים. בסעיף זה בחרנו לעבור עם $NumOctaves = \{1,4\}$ כאשר בסעיפים הקודמים עבדנו עם ערך ברירת המחדל $NumOctaves = 3$. נציג את התוצאות שקיבלנו:



נציג את זמני הריצה עבור גילו הפיצ׳רים:

- עבור $NumOctaves = 1$ התגלו 72 פיצ׳רים וזמן הריצה הינו:

```
time4 =

    0.0115
```

- עבור $NumOctaves = 4$ התגלו 112 פיצ׳רים וזמן הריצה הינו:

```
time3 =

    0.0364
```

נוכל לראות שכאשר הערך של האוקטבה גדול יותר כך נקבל יותר פיצ׳רים שמתגלים והפיצ׳רים שהתגלו הרבה יותר גדולים. בנוסף אם נשווה את מספר הפיצ׳רים שהתקבלו כעת אל מול הערך של ברירת המחדל בסעיפים הקודמים נוכל לראות כי קיבלנו שמספר מציאת הפיצ׳רים הינו הגיוני, כלומר עבור ערך של ברירת המחדל קיבלנו 111 פיצ׳רים, עבור ערך של 1 קיבלנו 72 פיצ׳רים ועבור ערך של 4 קיבלנו 112 פיצ׳רים.

מבחינת זמני הריצה:
נוכל לראות שההורדה של כמות האוקטבות ל – 1 השפיעה שצורה יחסית משמעותית על זמן הריצה ביחס לערך של ברירת המחדל בסעיפים הקודמים. בנוסף נוכל לראות שהגדלה של כמות הרמות אינה משפיעה משמעותית על זמן הריצה בין אוקטבה 1 ל – 4.

באלגוריתם הנתון מבצעים שימוש במסנן ריבועי שמקרב את המסנן הגאוסי. כאשר לוקחים מסנן גדול יותר המשמעות היא מיצוע על יותר פיקסלים עבור פיקסל מרכזי נתון, ולכן כל פיצ׳ר מייצג אזור גדול יותר בתמונה, כלומר הגדלת האוקטבות מאפשרת מציאת פיצ׳רים גדולים יותר בתמונה שכי שניתן לראות בתוצאות.

1.1.7 Find features in the image using 2 different *'NumScaleLevels'*

בדומה לשני הסעיפים הקודמים, כעת נפעיל את הפונקציה כאשר אנחנו משנים את הפרמטר *'NumScaleLevels'*. נציג את התוצאות ואת זמן ריצת האלגוריתם עבור 2 גדלים שונים של הפרמטר המבוקש, ונבצע השוואה לסעיפים הקודמים. בסעיף זה בחרנו לעבור עם $NumScaleLevels = \{3,8\}$ כאשר בסעיפים הקודמים עבדנו עם ערך ברירת המחדל $NumScaleLevels = 4$. נציג את התוצאות שקיבלנו:

Extracted features with:

NumScaleLevels=3

NumScaleLevels=8

נציג את זמני הריצה עבור גילו הפיצ׳רים:

- עבור $NumScaleLevels = 3$ התגלו 52 פיצ׳רים וזמן הריצה הינו:

```
time6 =

0.0068
```

- עבור $NumScaleLevels = 8$ התגלו 179 פיצ׳רים וזמן הריצה הינו:

```
time5 =

0.0518
```

נוכל לראות שזמני הריצה עבור הערכים שבחרנו נמוכים יותר מאשר מה שיצא לנו עם הפרמטרים של ברירת המחדל, כאשר ערך ברירת המחדל הינו $NumScaleLevels = 4$, אך נשים לב שזמן הריצה עבור $NumScaleLevels = 8$ גבוה מזה של $NumScaleLevels = 3$.

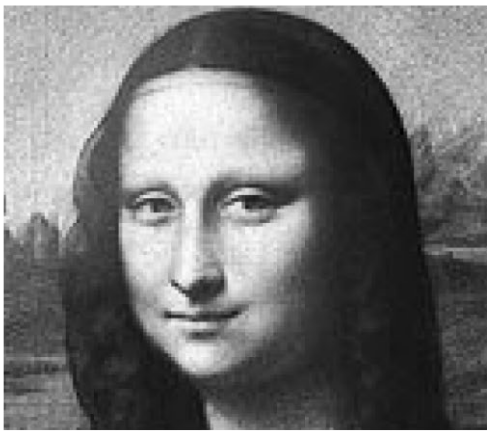
נאמר כי כמות הסקאלות קובעת כמה רמות של מסננים יופעלו על התמונה בכל אוקטבה נתונה. ככל שניקח יותר מסננים נראה שינויים בתדרים הנמוכים, ובנוסף נקבל כי כמות החישובים גדולה ולכן זמן הריצה עולה. מהתוצאות נוכל לראות כי עבור 8 רמות מצאנו יותר פיצ׳רים בהשוואה לסעיפים הקודמים בהם היה ערך של ברירת המחדל. כלומר נוכל לומר כי ככל שנעלה במספר ברמות של הסקאלה כך נמצא יותר פיצ׳רים.

1.2 Make Mona Straight Again

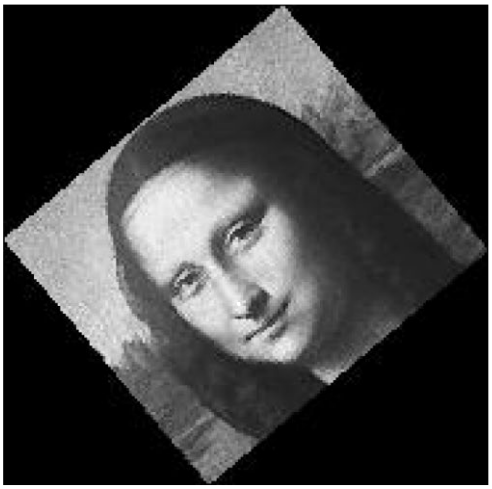
1.2.1 Reading the images ’straight_mona.PNG’ and ’crooked_mona.jpg’.

נקרא את התמונות: *'straight_mona.PNG' , 'crooked_mona.jpg'* נמיר אותם ל – Gray Scale ולאחר מכן ננרמל אותן. נציג את התמונות שקיבלנו:

Normalized Straight Mona Image



Normalized Crooked Mona Image

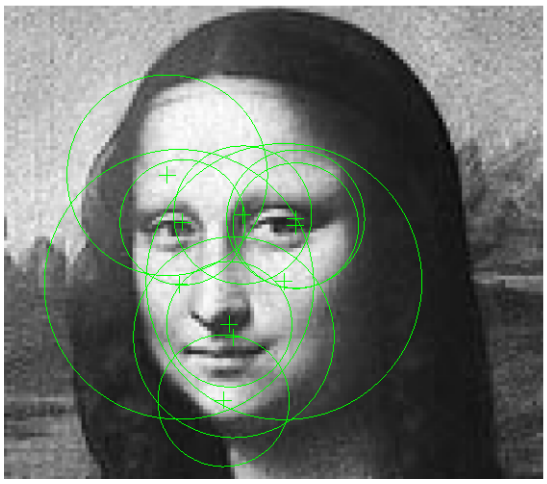


1.2.2 Extract the SURF feature points of each of the images and display the ten strongest features

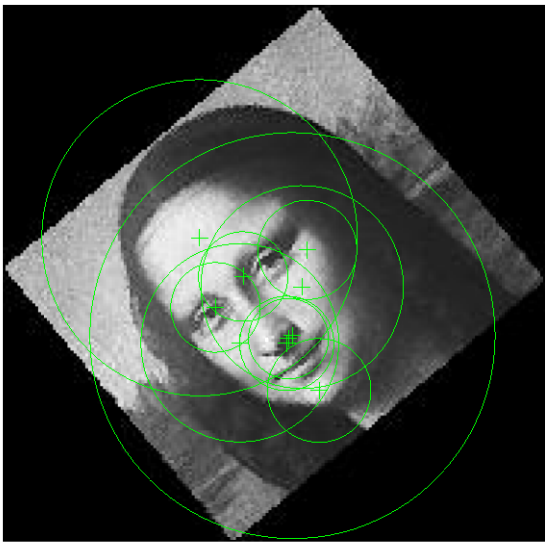
כעת נחלץ את הפיצ’רים מכל אחת מן התמונות על ידי שימוש באלגוריתם SURF, ובנוסף נציג את 10 הפיצ’רים החזקים ביותר:

Ten strongest features of:

Straight Mona image



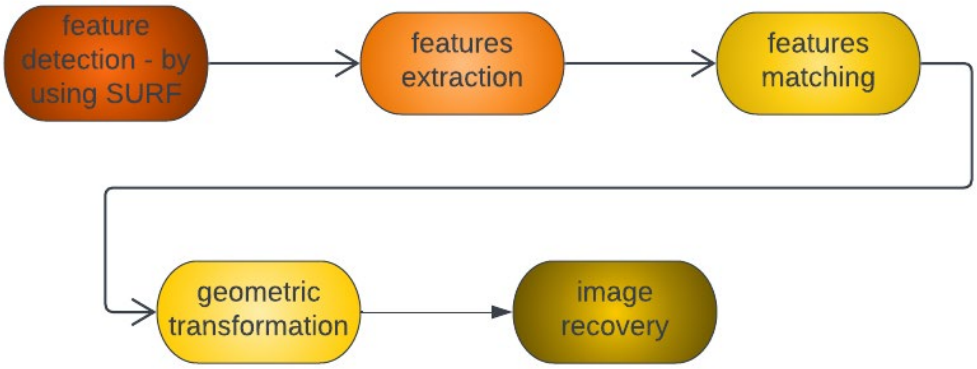
Crooked Mona image



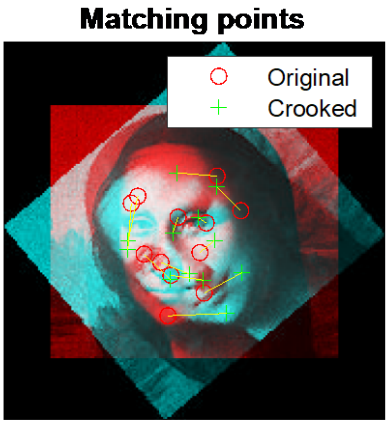
1.2.3 Using the extracted feature points to straighten Mona's face

כעת נכתוב אלגוריתם שיחזיר את התמונה המסובבת למקום, כלומר מיישר את הפרצוף של מונה, וזאת על ידי הפיצ'רים שחולצו בסעיף הקודם.

נתאר את האלגוריתם על ידי דיאגרמת בלוקים ונסביר כל בלוק בתהליך:



- תחילה ביצענו feature detection. בחלק זה השתמשנו בפונקציה המובנית של MATLAB: *'detectSURFFeatures'* שמזהה פיצ'רים ומחזירה אובייקט שנקרא "SURFPoints", אשר מכיל מידע על הפיצ'רים.
- לאחר מכן נבצע feature extract על ידי שימוש בפונקציה *'extractFeatures'* אשר מקבל את האובייקט משלב קודם ואת התמונה ממנה חולצו הפיצ'רים ומחזיר מטריצה כאשר כל שורה בה היא descriptor בודד.
- בשלב הבא נבצע feature matching כאשר בחלק זה השתמשנו בפונקציית *'matchFeatures'* אשר מקבלת את הפיצ'רים של כל אחת מהתמונות בצורה של descriptor עבור כל אחת מהתמונות שנרצה להתאים ביניהן. בפלט של הפונקציה נקבל אינדקסים מתאימים מכל אחד מהפיצ'רים שהכנסנו כקלט. נאמר כי שני descriptors הם תואמים אם המרחק ביניהם קטן מערך שהוגדר ע"י הפרמטר MatchTreshold. נציג את התוצאות:



- לאחר מציאת התאמת הנקודות נבצע geometric transformation. נשתמש בפונקציה *'estimateGeometricTransfom'* אשר מקבלת הנקודות שהותאמו מכל תמונה. הפונקציה משערכת את ה – Transform ומייצרת אובייקט מסוג "affine2d" שמייצג טרנספורמציה אפינית (דו ממדית). אובייקט זה מבצע מיפוי של ההתאמות ל inliers ול – outliers. האובייקט שנוצר מתעלם מ – outliers בחישוב הטרנספורמציה וכך משפר את הדיוק שלה. כדי להפעיל את הטרנספורמציה על התמונה המסובבת נשתמש בפונקציה *'imwrap'*. נציג את התוצאה הסופית שקיבלנו:

Fix crooked image to straight image

1.3 Fake or Real

1.3.1 Reading the image *'mona_org.jpg'* and detect images with real Mona Liza face exist.

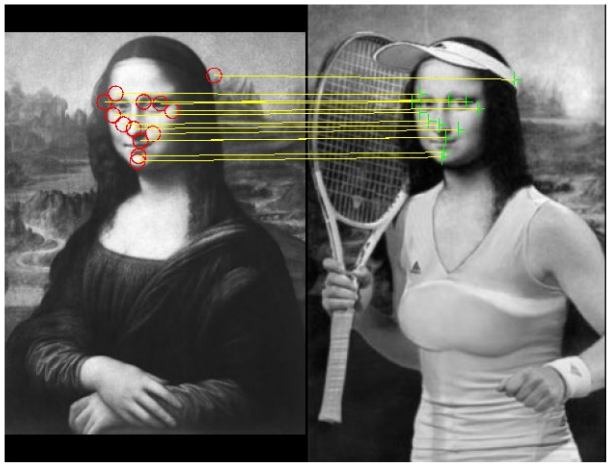
בחלק זה נקרא את התמונה המקורית של המונה ליזה: 'mona_org.jpg'. נחתוך את התמונה על מנת לקבל רק את החלק של הפנים בדומה לדוגמא של השאלה. נשתמש ב – SURF על מנת לזהות בצורה אוטומטית מי מהתמונות מכילה פנים אמיתיות של המונה ליזה ומי פנים מזויפות של מונה ליזה.

1.3.2 Display the images we found

נציג את התמונות והזיהוי של הפנים של כל תמונה, האם מדובר הפנים האמיתיות של המונה ליזה או לא:

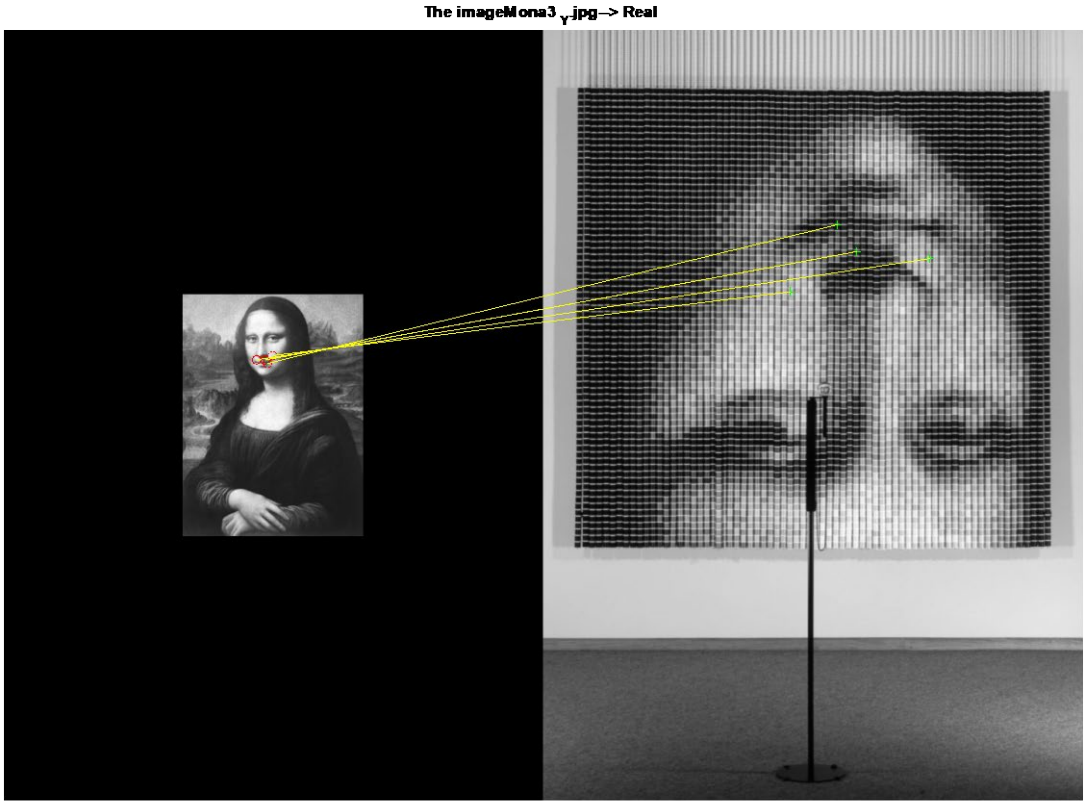
עבור Mona_1:

The imageMona1 $_{Y}$.jpg–> Real

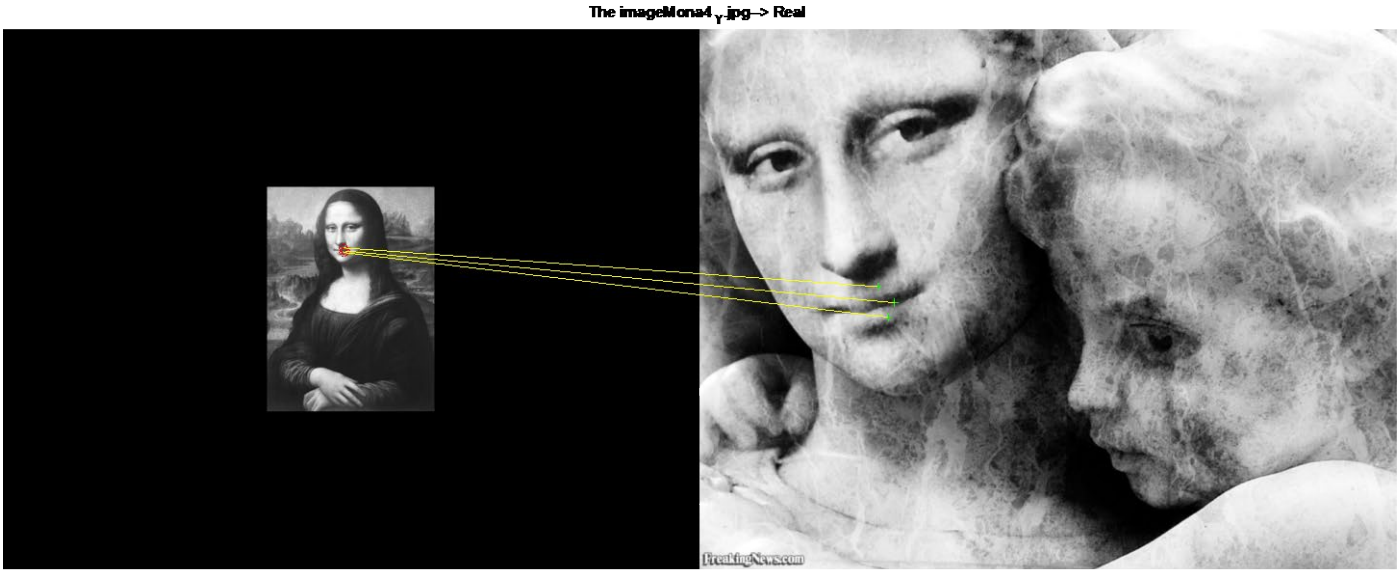


עבור Mona_2:

עבור Mona_3:

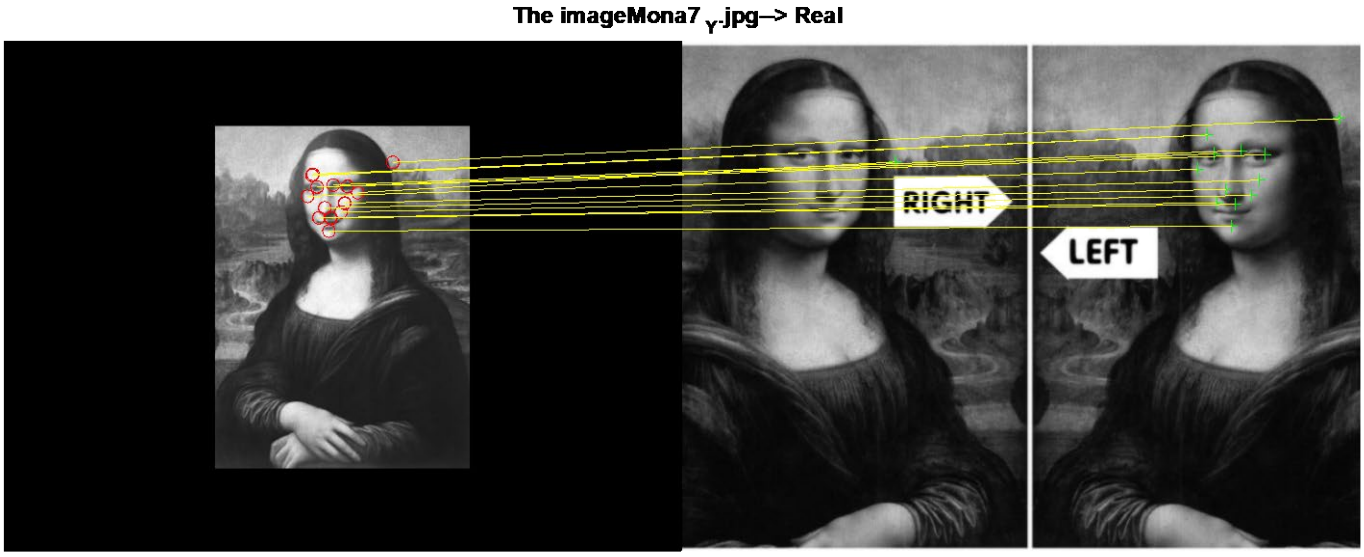


עבור Mona_4:



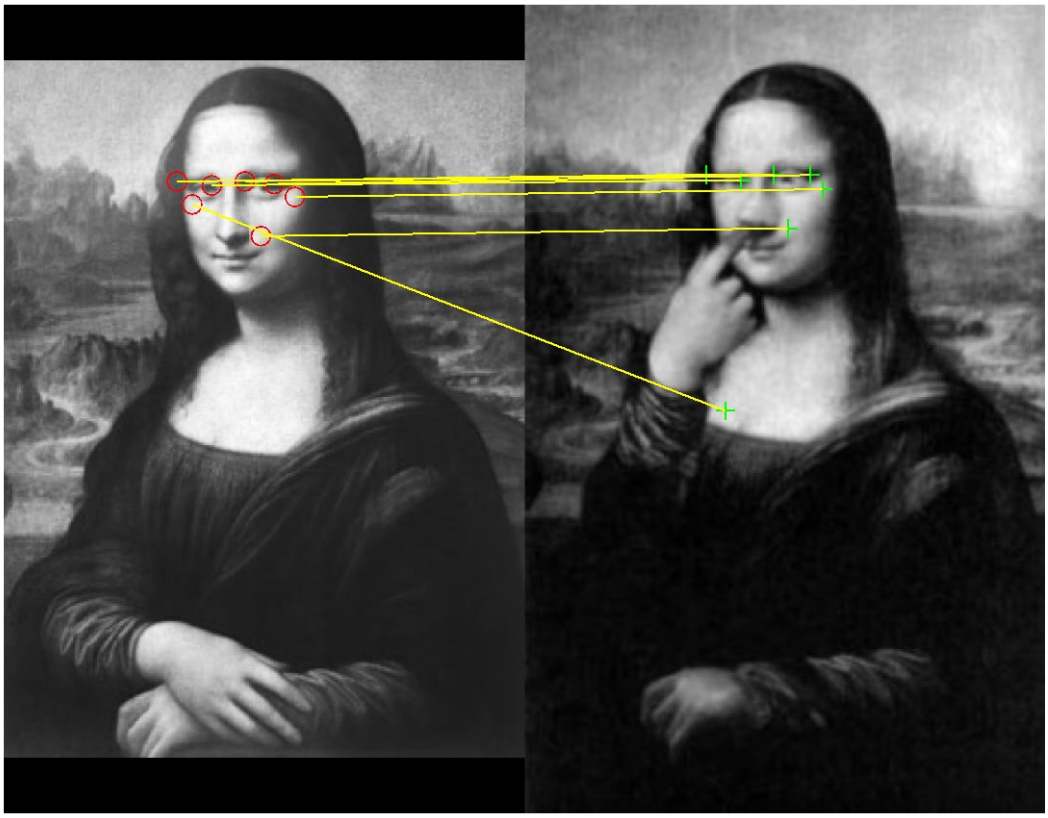
עבור Mona_6 , Mona_5 נקבל כי הפנים של מונה מזויף, כלומר הפנים של מונה לא מוכלות בתמונות אלו.

עבור Mona_7:



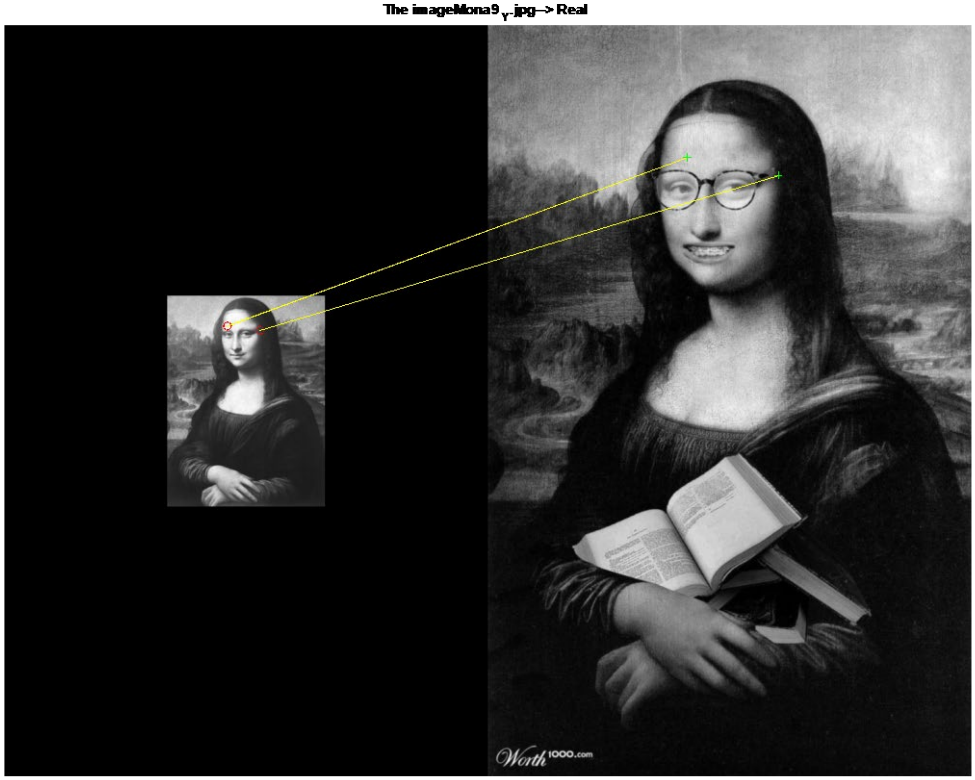
עבור Mona_8:

The imageMona8 $_Y$.jpg--> Real



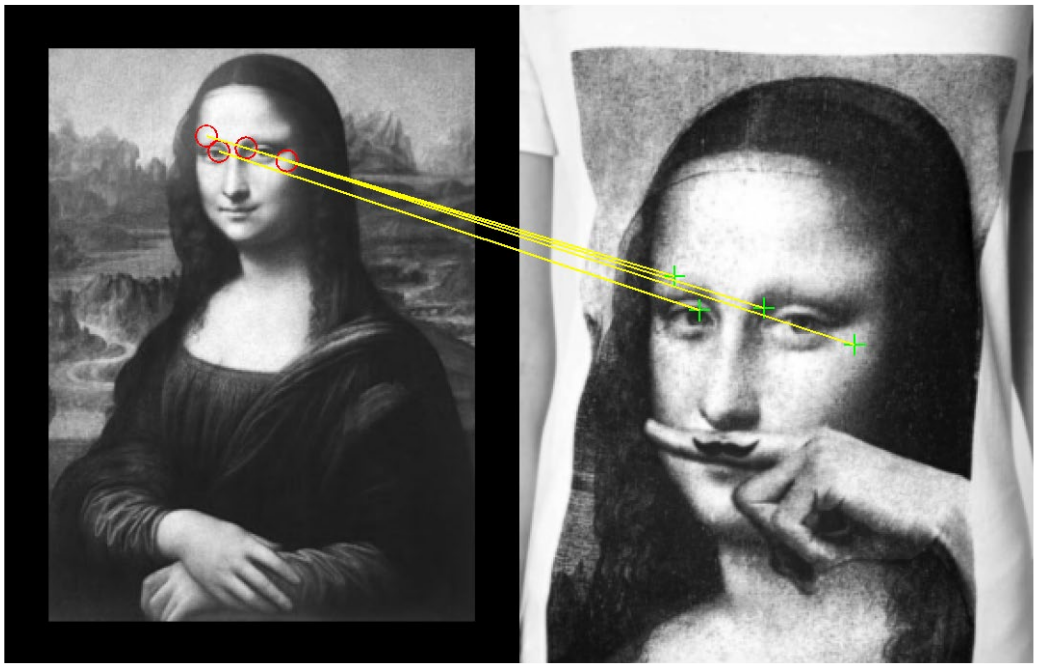
עבור Mona_9:

The imageMona9 $_Y$.jpg--> Real



עבור Mona_10:

The imageMona10 $_Y$.jpg--> Real



עבור Mona_12 , Mona_11 נקבל כי הפנים של מונה מזויף, כלומר הפנים של מונה לא מוכלות בתמונות אלו.

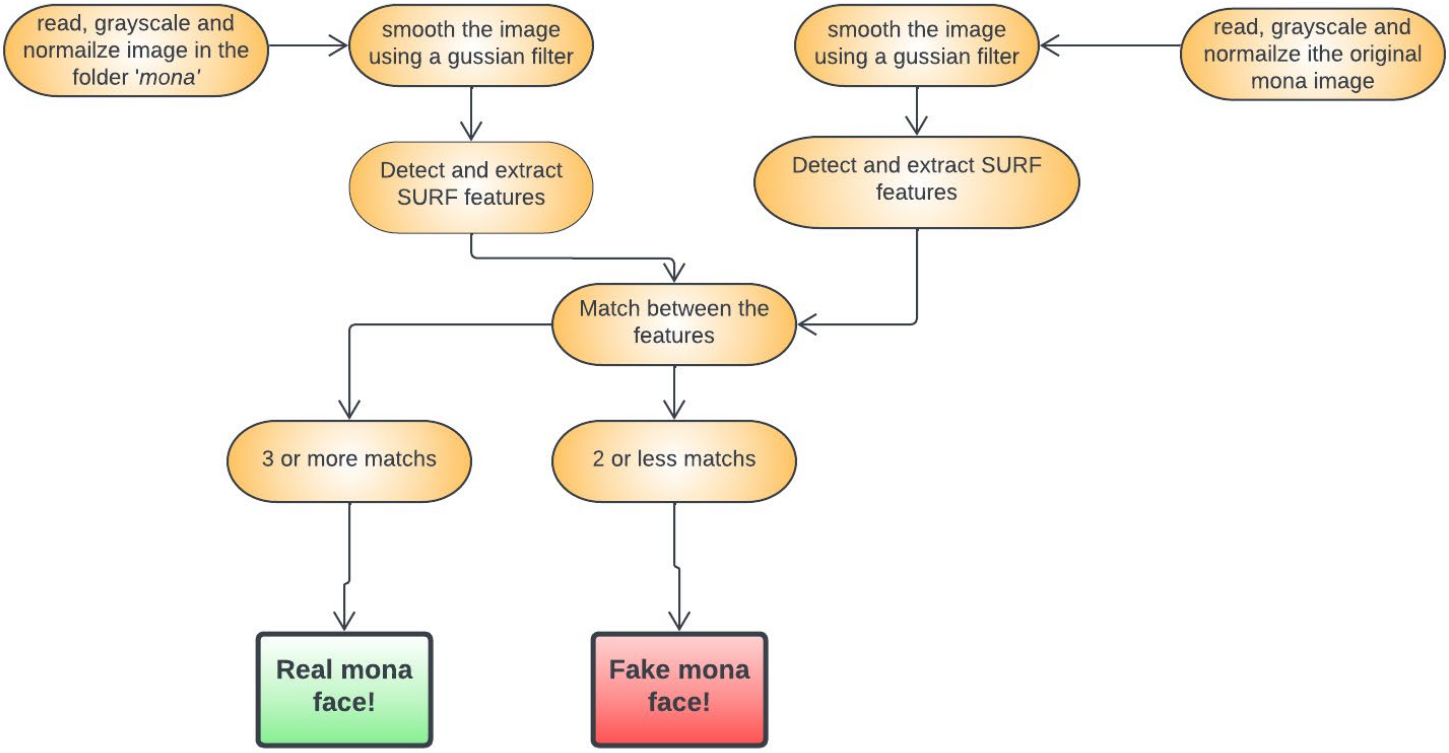
לסיכום התמונות שזיהינו כפנים אמיתיות של מונה הן:

```
The imageMona10_Y.jpg is Mona
The imageMona1_Y.jpg is Mona
The imageMona2_Y.jpg is Mona
The imageMona3_Y.jpg is Mona
The imageMona4_Y.jpg is Mona
The imageMona7_Y.jpg is Mona
The imageMona8_Y.jpg is Mona
The imageMona9_Y.jpg is Mona
```

הצלחנו לזהות את כל התמונות שמכילות את הפנים האמיתיות של מונה ולא קיבלנו שום זיהוי שיקרי, כלומר תמונה בה יש פנים מזויפות של מונה וזיהו אותן כפנים אמיתיות של מונה.

1.3.3 Explaining the algorithm using a block diagram

הסבר על האלגוריתם שבנינו – דיאגרמת בלוקים:



הסבר מילולי של שלבי האלגוריתם:

- ראשית נקרא את התמונה המקורית של מונה ואת שאר התמונות מהתיקייה של 'mona' שבהן נבדוק אם הן מכילות את הפנים של מונה. לאחר מכן נבצע נרמול והמרה ל – Gray Scale.
- לאחר מכן נעביר את התמונות בגווני האפור המנורמלות דרך מסנן גאוסי לצורך החלקה של התמונות כדי לשפר את התוצאות, כלומר האלגוריתם יהיה יותר עמיד לרעש.
- נבצע זיהוי של מאפייני הSURF עבור התמונה המקורית של מונה כאשר נגדיר את אזור העניין שלנו לאזור הפנים של מונה. לצורך כך נשתמש ב-Roi ובנוסף נגדיר $NumOcatves = 4$ ו-$NumScaleLevels = 8$.
- נשתמש בפונקציה *extractFeatures* על מנת לחלץ את הפיצ'רים של התמונה המקורית.
- נקרא תמונה מהמאגר שמכיל 12 תמונות
- ננרמל את התמונה ונמיר אותה ל-grayscale
- נחליק את התמונה על ידי שימוש במסנן גאוסי.
- נזהה את מאפייני הSURF של התמונה (נגדיר $NumOcatves = 4$ ו-$NumScaleLevels = 8$).

- נחלץ את הפיצ׳רים על ידי שימוש בפונקציה *extractFeatures*
- נשתמש בפונקציה *matchFeatures* עם ערך סף של 1.6 עבור התמונה המקורית והתמונה מהמאגר .
- במידה ונמצאו לפחות 3 התאמות בין התמונות, נכריז על התמונה מן המאגר שמצאנו את מונה.
- נבצע את אותו התהליך לכל אחת מהתמונות במאגר.

1.3.4 Analyze the results

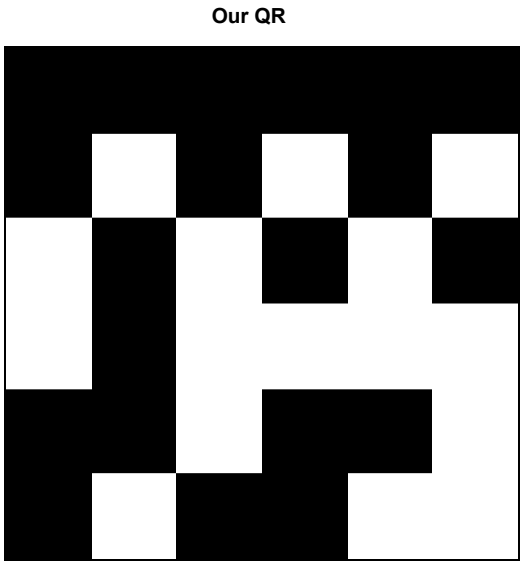
קיבלנו זיהוי ב100% דיוק וכמו כן לא זיהינו שום תמונה שיקרית, לכן ניתן להסיק כי האלגוריתם שבנינו מתאים לתמונות והבחירה של הפרמטרים בפונקציה המזהה את מאפייני הSURF מתאימים לזיהוי פיצ׳רים בכל התמונות עד כדי התאמה מספקת בין התמונה המקורית לתמונות שבמאגר.

2. QR Code Reader

2.1 Insert (311394357) ID to the attached function ID2QR(id)

הכנסנו את תעודת הזהות '311394357' לפונקציה 'ID2QR(id)' שמקבלת ID ומחזירה QR.

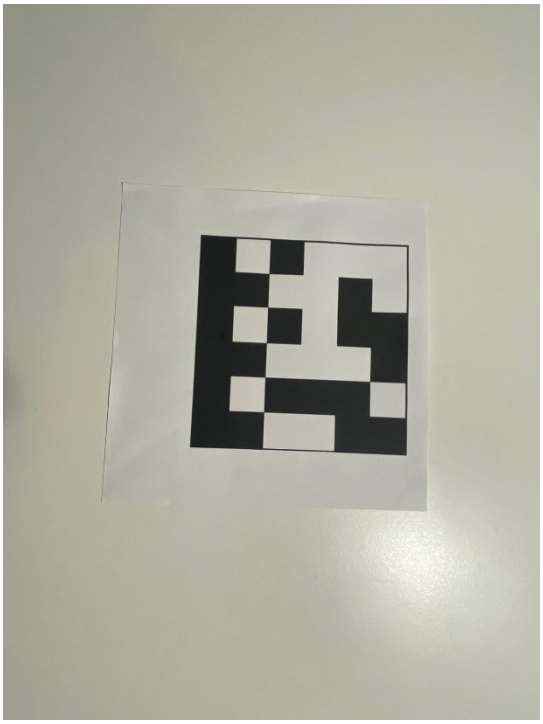
הדפסנו את ה-QR הבא:



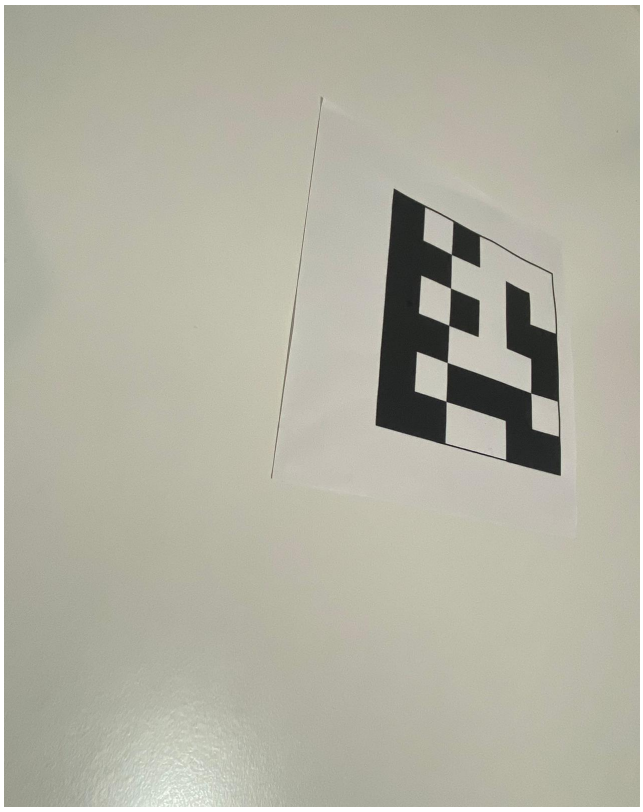
2.2 three photos of the QR code from three different angles

לקחנו שלושה תמונות ברמות קושי שונות של הQR:

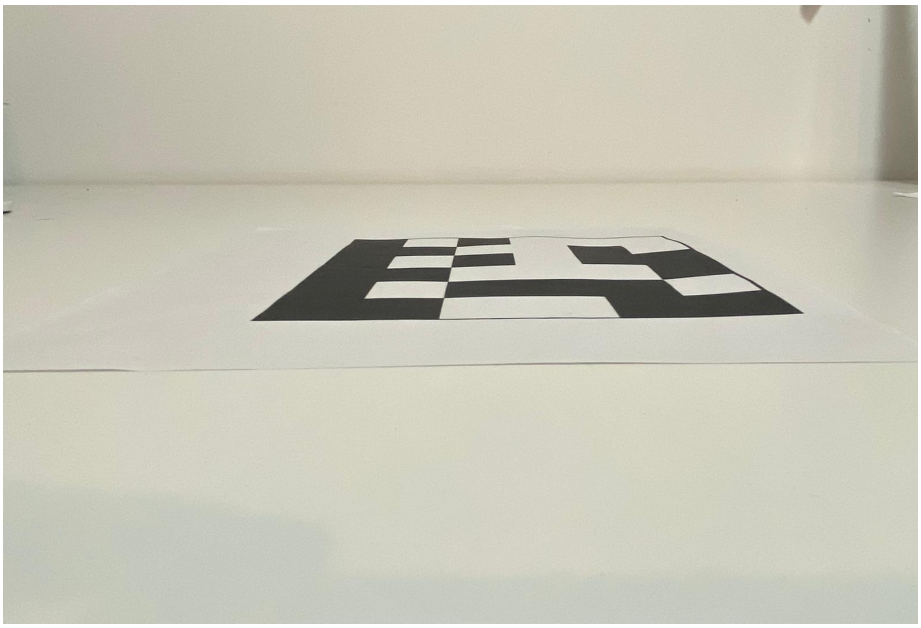
הראשונה ברמת קושי קלה:



השנייה ברמת קושי בינונית:



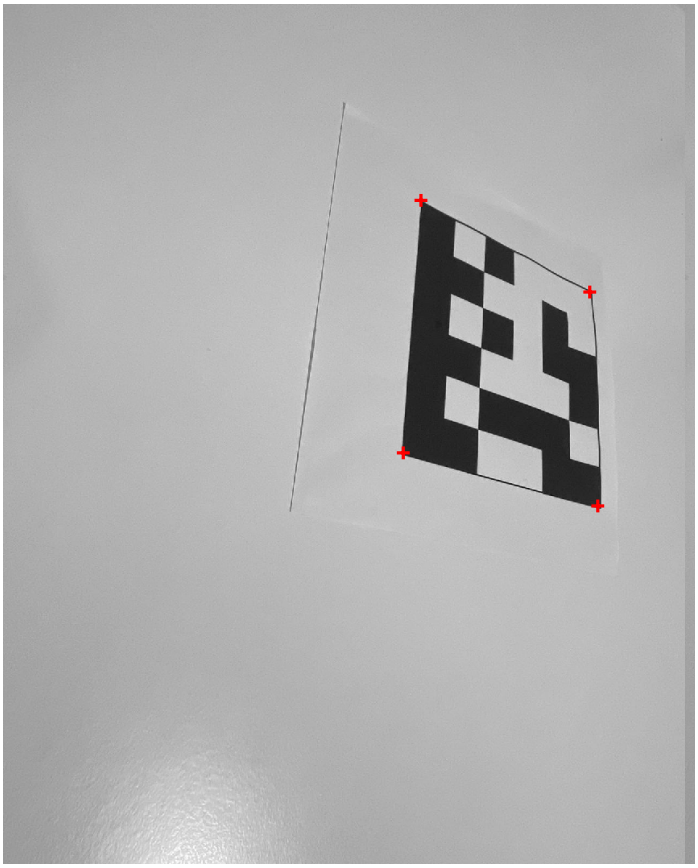
השלישית ברמת קושי קשה:



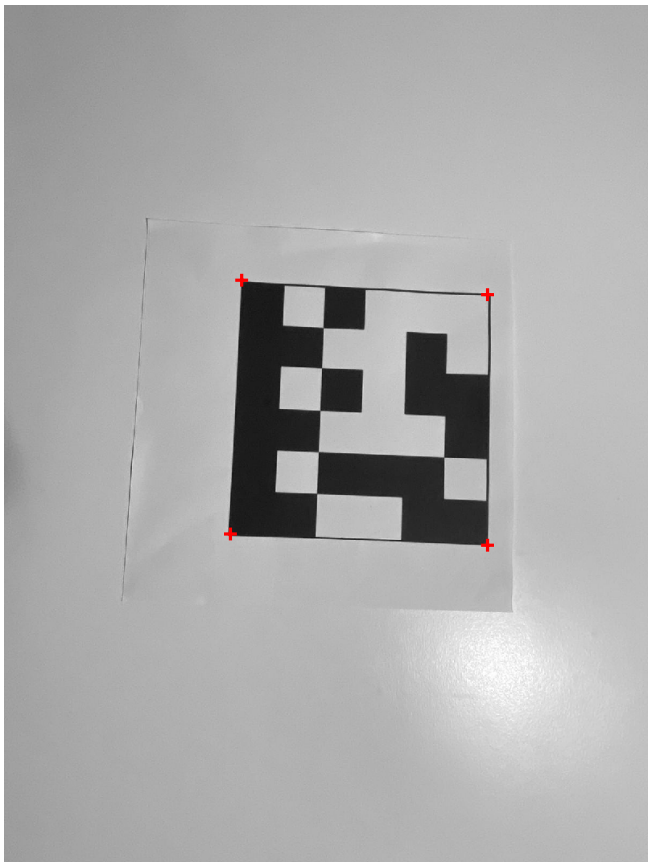
2.3 Locate the points on the QR code – manually

מצאנו באופן ידני את הקואורדינטות של ארבעת הפינות של הQR בכל אחת מהרמות, על ידי שימוש בפונקציה *'ginput(4)'* ושמרנו אותם. נציג את התמונות ב*grayscale* יחד עם הנקודות השמורות:

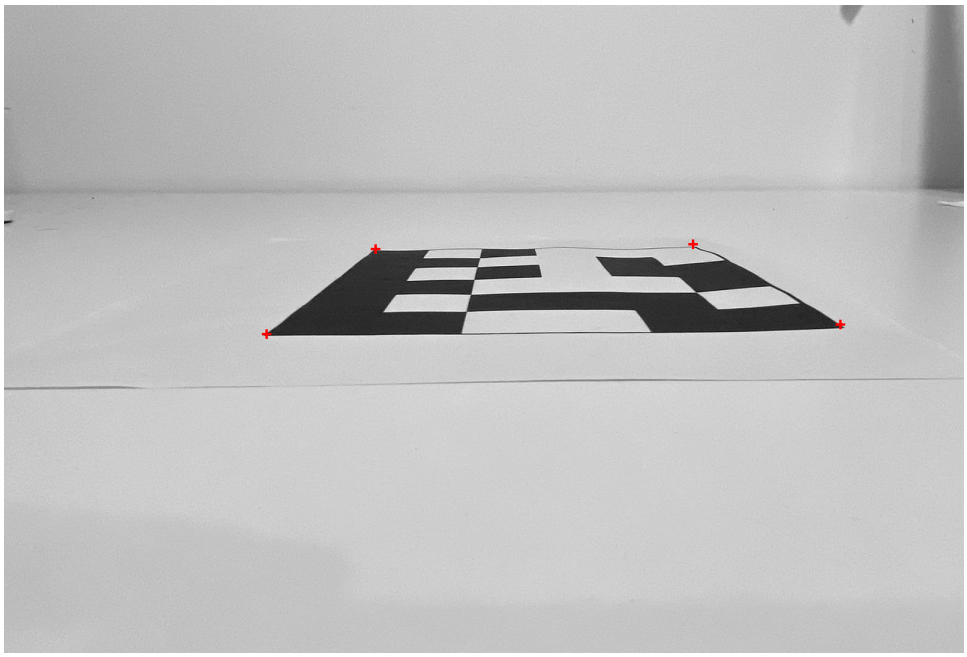
Intermediate QR



Easy QR



Hard QR



2.4 Transform the image such that the QR code is straightened

נבצע טרנספורמציה לתמונות כך שיהיו ישרות על ידי כל אחד מהטרנספורמציות שנלמדו בכיתה.

לצורך כך, בנינו פונקציה שנקראת *'straight_image'* שמקבלת את תמונת הQR, את סט הנקודות שבחרנו ידנית בפינות הQR, את סט הנקודות המתוקנות שמגדירות את הפינות של התמונה הישרה, ואת סוג הטרנספורמציה.

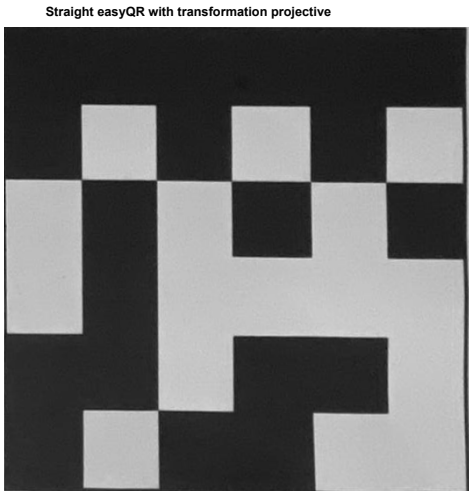
אנו בחרנו ליצור תמונה מתוקנת (ישרה) בגודל 1200x1200 כך שתהיה ריבועית ותתחלק ב-6 באופן שווה בשני הצירים, מאחר שקוד הQR מורכב מ- 6x6 ריבועים. לכן פינות התמונה המיושרת יהיו בקואורדינטות (1200,1200) , (1200,1) , (1,1200) , (1,1).

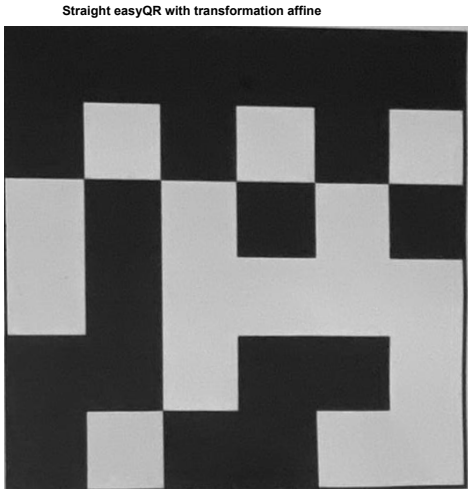
בפונקציה שלנו, נעזרנו בפונקציה מובנת של מטלב *'fitgeotrans'* שמקבלת את הקואורדינטות של הפינות של תמונת הQR , את הקואורדינטות של הפינות שלפיו הQR שלנו יתיישר ואת סוג הטרנספורמציה. הפונקציה מחזירה את מטריצת הטרנספורמציה. לכל תמונה נבצע את כל אחד מהטרנספורמציות Prespective, Affine ו-Rigid.

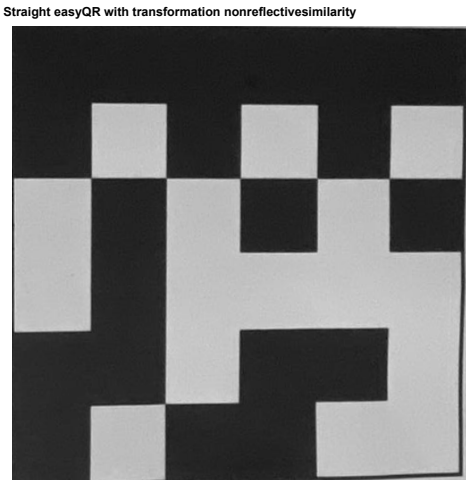
לאחר מכן, השתמשנו בפונקציה imwarp שמקבלת את התמונה עליה נרצה לבצע טרנספורמציה, את מטריצת הטרנספורמציה, ממדי התמונה המיושרת וכך נעביר את התמונה בטרנספורמציה המבוקשת ונקבל תמונה מיושרת.

נציג את התוצאות שקיבלנו לאחר יישור של כל אחד מהתמונות עבור כל אחת מהטרנספורמציות:

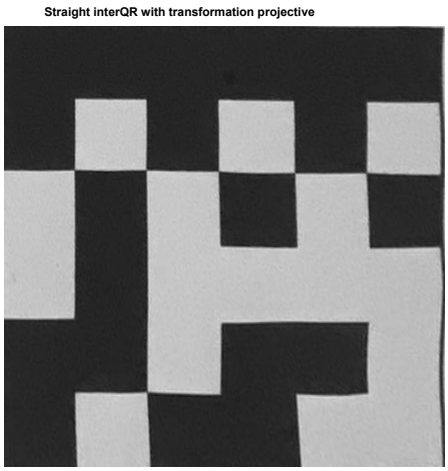
התמונות המיושרות המתקבלות בדרגת קושי קלה לכל אחת מהטרנספורמציות:

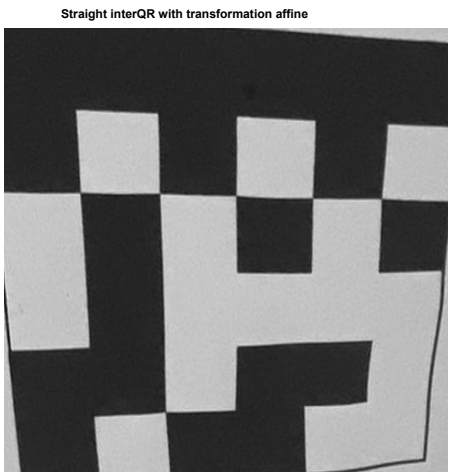


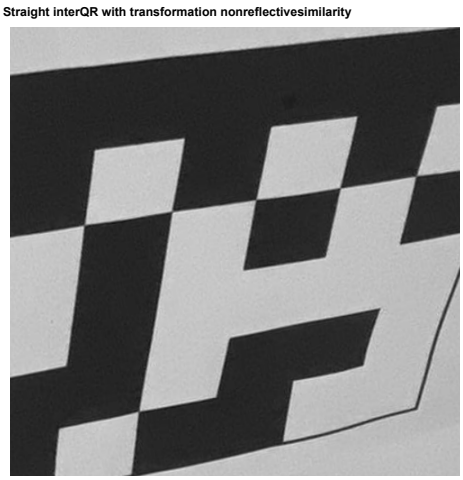




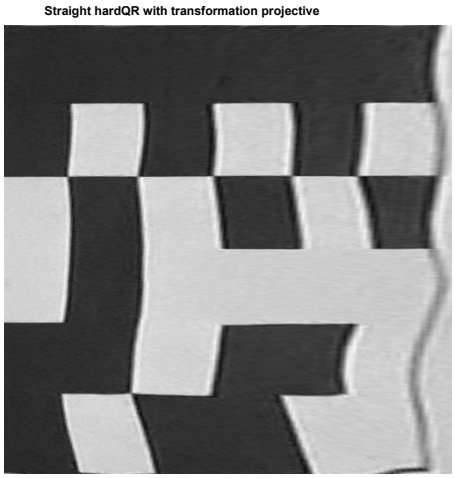
התמונות המיושרות המתקבלות בדרגת קושי בינונית לכל אחת מהטרנספורמציות:

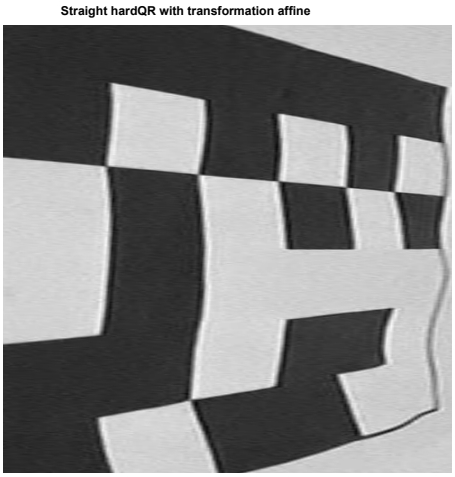


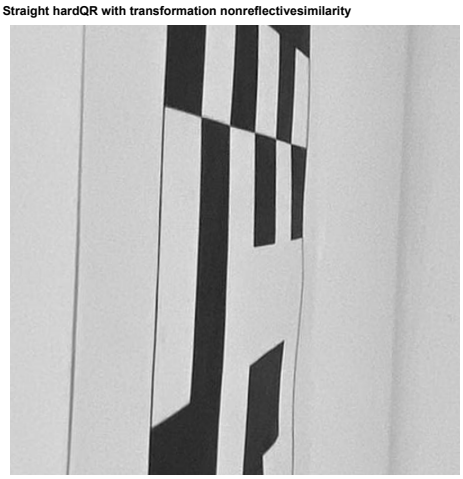




התמונות המיושרות המתקבלות בדרגת קושי קשה לכל אחת מהטרנספורמציות:







2.5 Explain the result for each transformation

עבור התמונה בדרגת קושי קלה, הצלחנו ליישר אותה עבור כל אחד מהטרנספורמציות, מאחר שתמונה זו צולמה מלמעלה וכל מה שהיה צריך לעשות בשביל ליישר אותה זה לסובב אותה, מה שכל אחת מהטרנספורמציות מסוגלת לעשות.

עבור התמונה בדרגת קושי בינונית, קיבלנו כי כאשר השתמשנו בטרנספורמציית Rigid , התמונה המתקבלת עקומה והשוליים של קוד הQR חתוכים. זאת מאחר שטרנספורמציה זו לא משנה את המרחק בין הפיקסלים אלא רק מגדילה, מקטינה או מסובבת את התמונה. לכן מאחר שצילמנו את הQR מהצד, הטרנספורמציה לא הצליחה לתקן את זה.

כאשר השתמשנו בטרנספורמציית Affine קיבלנו תוצאות קצת יותר טובות, התמונה מיושרת יותר אך עדיין השוליים חתוכים. טרנספורמציה זו כוללת גם הזזה, כיווץ או מתיחה כמו בטרנספורמציית Rigid, ובנוסף מבצעת shear ולכן קיבלנו תוצאה יותר טובה.

כאשר השתמשנו בטרנספורמציית Prespective קיבלנו את התוצאות הטובות ביותר עבור דרגת קושי זו, ניתן לראות כי כל הQR נמצא בתמונה בצורה מיושרת וברורה והמיפוי נעשה כראוי. טרנספורמציה זו כוללת את כל הביצועים שמתאפשרים על ידי טרנספורמציית Affine ומעבר לכך יש לה אלמנטים שמשנים את הפרספקטיבה בתמונה, לכן ברור כי נקבל תוצאות טובות יותר.

עבור התמונה בדרגת קושי קשה, קיבלנו בדומה לדרגת הקושי הבינונית, שטרנספורמציית Rigid הניבה את התוצאות הכי פחות טובות, ניתן לראות כי התמונה לא ישרה, השוליים שלה חתוכים ועדיין מוטת כמו בזווית הצילום, לומר איבדנו מידע רב לגבי הקוד QR. בטרנספורמציית Affine התמונה מיושרת יותר אך עדיין מוטת, השוליים מעט חתוכים אך הרבה פחות לעומת Rigid , ועבור Prespective קיבלנו תמונה הרבה יותר מיושרת, הקוד QR נראה מלמעלה ולא מן הצד, ולכן בטרנספורמציה זו קיבלנו את התוצאות הטובות ביותר.

2.6 Extract the binary values

כדי להוציא את הערכים הבינאריים מתוך התמונות המיושרות, בנינו פונקציה שנקראת 'img2QR' שמקבלת את התמונה המיושרת ומחלצת ממנו את הערכים הבינאריים ומכניסה אותה לתוך מטריצה בגודל 6x6 .

פונקציה זו עושה שימוש בגודל התמונה המיושרת שהגדרנו מראש 1200x1200 , ומחלקת את התמונה ל36 חלקים שווים בגודלם, כל חלק בגודל 200x200. עבור כל חלק, בדקנו את הערך הממוצע של הפיקסלים שבה, ועל ידי הגדרת ערך סף של 0.5, הצבנו במקום המתאים במטריצה הבינארית את הערך המתקבל בחלק המתאים (אם הממוצע בחלק זה גדול מ-0.5 אז הערך הבינארי במיקום המתאים הוא 1, ואילו אם הממוצע בחלק זה קטן מ-0.5 אז הערך הבינארי במיקום המתאים הוא 0).

2.7 Convert every 4 bits to an integer

כדי להמיר כל 4 ביטים במטריצה הבינארית למספר integer וכך לחלץ את תעודת הזהות, בנינו פונקציה שנקראת *'QR2ID'* שמקבלת את המטריצה הבינארית של הQR, מיישרת אותה וממירה כל ארבעה ביטים למספר דצימלי.

2.8 Verified the correct ID

כעת נשווה את התוצאות שקיבלנו לכל אחד מהתמונות עבור כל אחד מהטרנספורמציות עם תעודת הזהות המקורית שהיא 311394357:

Difficulty	Transformation	Result	Success Rate
	Rigid	3,1,1,3,9,4,3,5,7	**100%**
Easy	Affine	3,1,1,3,9,4,3,5,7	**100%**
	Prespective	3,1,1,3,9,4,3,5,7	**100%**
Intermediate	Rigid	9,0,0,3,8,4,3,4,7	**44.4%**
	Affine	3,1,1,3,9,4,3,4,7	**88.8%**
	Prespective	3,1,1,3,9,4,3,5,7	**100%**
	Rigid	15,15,15,7,9,15,15.15.15	**11.1%**
Hard	Affine	3,1,1,4,4,14,1,6,15	**33.3%**
	Prespective	3,1,1,3,9,4,3,5,7	**100%**

ניתן לראות כי קיבלנו 100% דיוק בשחזור תעודת עבור התמונה בדרגת קושי קלה בכל אחת מהטרנספורמציות, כצפוי מניתוח התמונות המיושרות שביצענו קודם.

נשים לב כי עבור דרגת קושי בינונית וקשה, קיבלנו 100% דיוק בשחזור תעודת הזהות על ידי טרנספורמציית Prespective, ואילו על ידי טרנספורמציית Rigid אחוזי הדיוק הם הנמוכים ביותר, כצפוי.

לכן ניתן להסיק כי טרנספורמציית Prespective היא המתאימה ביותר, ומניבה את התוצאות הטובות ביותר, כפי שראינו ביישור התמונות וכמו כן בשחזור תעודת הזהות, טרנספורמציה זו מאפשר יישור תמונה בכל קונסטלציה של זוויות צילום, אך מאחר שהיא מכילה הכי הרבה פרמטרים היא בעלת זמן הריצה הארוך ביותר משאר הטרנספורמציות.

3. Automatic Corner Detector

התבקשנו לכתוב פונקציה שמוצאת את ארבעת הפינות של קוד הQR באופן אוטומטי.

לכן מימשנו אלגוריתם שמקבל תמונה כקלט ומחזיר את ארבעת הקואורדינטות של הפינות בקוד הQR כפלט. נסביר את שלבי האלגוריתם:

שלב 1 – ביצענו המרה של התמונה שהפונקציה מקבלת ל-grayscale ונירמלנו אותה בין 0 ל-1.

שלב 2 – נרצה להסיר את הרקע של התמונה, לכן נצבע כל פיקסל שאינו לבן או שחור לצבע לבן (ערך 1), מאחר שאותנו מעניין רק הצבע השחור והלבן של קוד הQR ולכן נרצה להשתיק את הרקע.

בשלב זה הנחנו כי השוליים של הדף של קוד הQR הם לבנים ולכן יכולנו ליצור רקע לבן אחיד.

שלב 3 – נמיר את התמונה לתמונה עם ערכים בינאריים, כלומר נקבל תמונה עם ערכים של 0 ו-1 בלבד.

שלב 4 – נשתמש בפונקציית edge כאשר במתודולוגיה היא Canny, כך שנוכל למצוא את הקצוות של התמונה, כלומר במקרה שלנו נרצה למצוא את קווי המתאר של הQR.

שלב 5 - נשתמש בHough transform על התמונה של הקצוות על מנת למצוא ערכי שיא. אנחנו נחפש 8 ערכי שיא כדי לוודא שנמצא את הנקודות החשובות לנו. לאחר מכן, השתמשנו בפונקציית houghlines שמחלצת את הקווים המשמעותיים ביותר מתוך ערכי השיא שנמצאו על ידי הטרנספורמציה.

נקבל עבור כל קו שתי קואורדינטות שבניהם נמתח קו, rho שמתאר את המרחק מהפיקסל הראשון בתמונה (0,0) ו-theta שמתאר את השיפוע של הקו במעלות.

שלב 6 – על ידי הנתונים שמצאנו עבור כל קו משמעותי, נרצה למצוא מתוך הקווים המשמעותיים את הקווים החיצוניים של הקוד QR ולהימנע מקווים פנימיים. לכן נחפש את הקווים החיצוניים לפי שיפוע ולפי מרחק מהפיקסל הראשון בתמונה, כלומר נרצה למצוא שני שיפועים זהים (theta) בכיוונים מאונכים ועבור כל אחד מהם נמצא קו במרחק מינימלי וקו במרחק מקסימלי מהפיקסל הראשון בתמונה (rho), וכך נקבל 2 זוגות של קווים מקבילים, וזוגות הקווים מאונכים זה לזה, כלומר נקבל את ארבעת הצלעות המגדירות את מבנה קוד הQR.

שלב 7– נחשב את משוואת הישר של כל קו שמצאנו שמתאר צלע של הQR ונמצא את נקודות המפגש של הישרים, נקודות מפגש אלו מגדירות לנו את ארבעת הפינות של קוד הQR.

הסברים על הפונקציות המובנות של מטלב שבהם השתמשנו:

1) 'Edge(I,'Canny')' – מקבלת תמונה ואת המתודה שלפיה היא מחשבת את הקצוות של התמונה הנתונה. אנחנו נשתמש במתודה *'Canny'*, מתודה זו מוצאת את קצוות התמונה על ידי חישוב של הגדיאנט של התמונה ומציאת המקסימום הלוקאלי שלה. חישוב הגרדיאנט בכל נקודה בתמונה מתבצע על ידי הפעלת נגזרת של פילטר גאוסיאני. הפונקציה משתמשת בשני ערכי סף הקובעים אילו פיקסלים יהיו חלק מהקצוות של התמונה.

הפרמטרים האופציונליים שהפונקציה *'edge(I,'Canny')'* מקבלת הם:

Threshold- וקטור של שני ערכים, ערך גבוה וערך נמוך [low, high] בין 0 ל-1, הקובעים את ערכי הסף. הפונקציה מתעלמת מכל הפיקסלים שערכם קטן מערך הסף התחתון, ושומרת את כל הפיקסלים שערכם גדול מערך הסף העליון. פיקסלים בעלי ערך הנמצא בין ערכי הסף, המחוברים לקווי המתאר שנוצרו על ידי הפיקסלים שערכם גדול מערך הסף העליון, גם הם יהיו חלק מהקצוות של התמונה.

אם מכניסים לפונקציה ערך סף בודד, הפונקציה תגדיר ערך סף תחתון כ- default על ידי הכפלה של ערך הסף הקיים ב- 0.4.

אם לא הוגדרו בפונקציה ערכי סף כלל, הפונקציה תבחר כ-default ערכי סף כתלות בתמונה.

Sigma- סטיית התקן של הפילטר הגאוסיאני. אם לא מגדירים, הפונקציה מגדירה ערך דיפולטיבי השווה לשורש של 2. בהתאם לערך של sigma, הפונקציה בוחרת את גודל הפילטר.

2) 'Hough(BW)' – פונקציה זו מקבלת תמונת קצוות ומבצעת עליה את טרנספורמציית hough.

3) *'houghpeaks'* – פונקציה זו מקבלת את מטריצת hough שמצאנו, מספר הקווים המשמעותיים שאנו רוצים למצוא וערך סף 'threshold' שמגדיר את הערך המינימלי שיש להתחשב בפיק. הפונקציה מוצאת את ערכי השיא במטריצת hough.

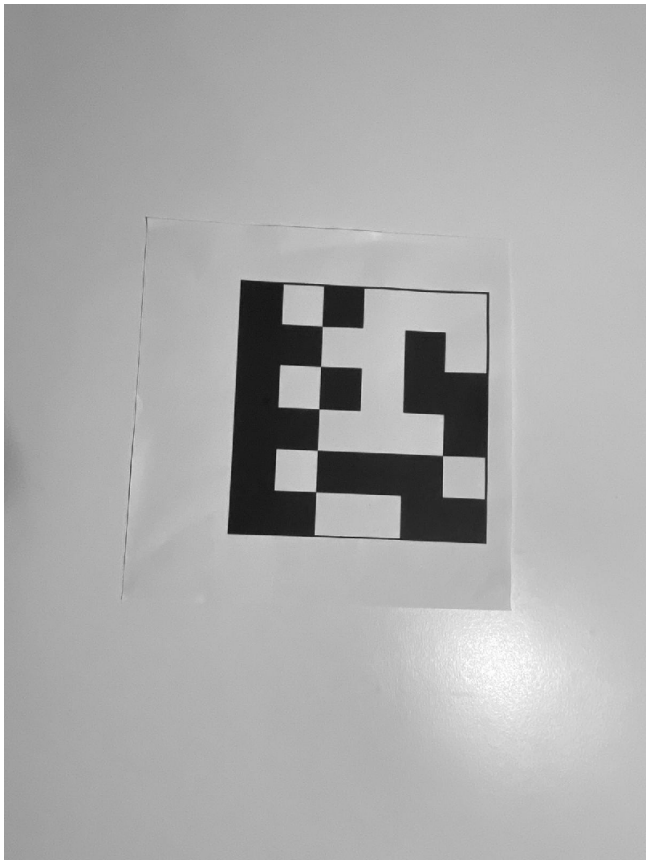
4) Houghlines – פונקציה זו מקבלת את תמונת הקצוות, נקודות השיא שמצאנו, את rho ו-theta ומחלצת קווים מתוך נקודות השיא שמצאנו.

הפעלנו את האלגוריתם על התמונה של קוד הQR בדרגת קושי קלה.

נציג את התוצאות לאורך שלבי האלגוריתם :

שלב 1 :

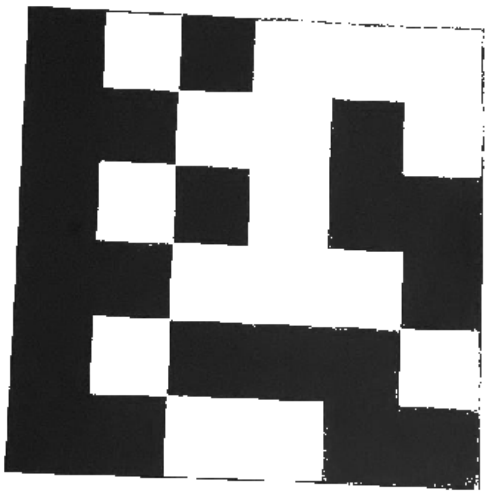
Original image normalized gray scale



שלב 2:

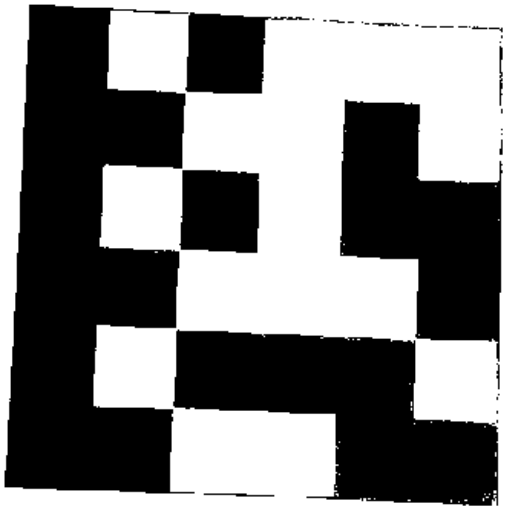
Values in range of [0.2 0.9] turn white

Original image



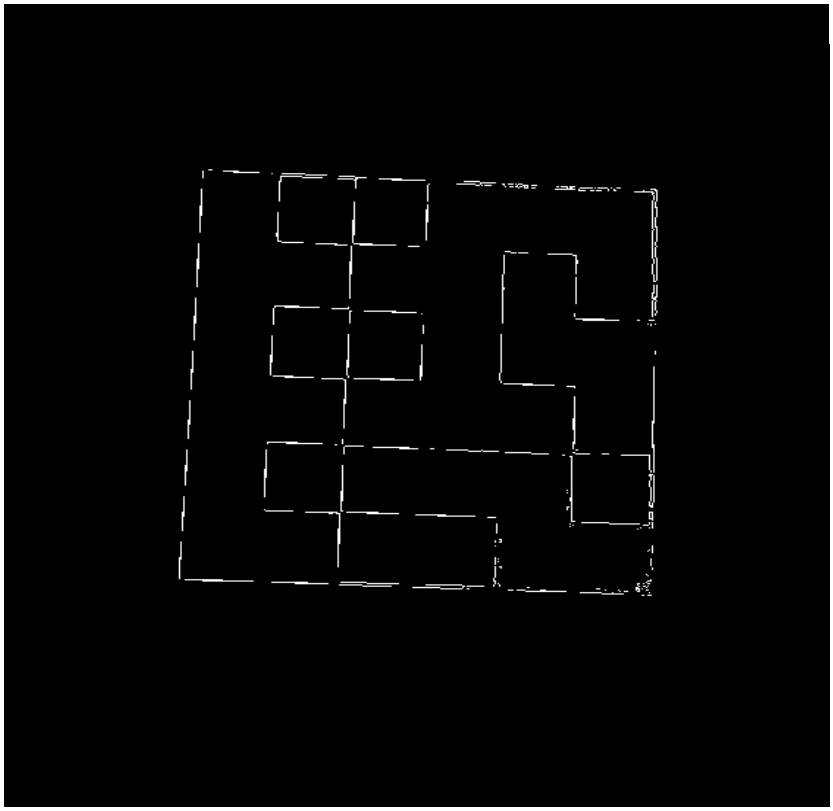
שלב 3:

Binary image
Original image



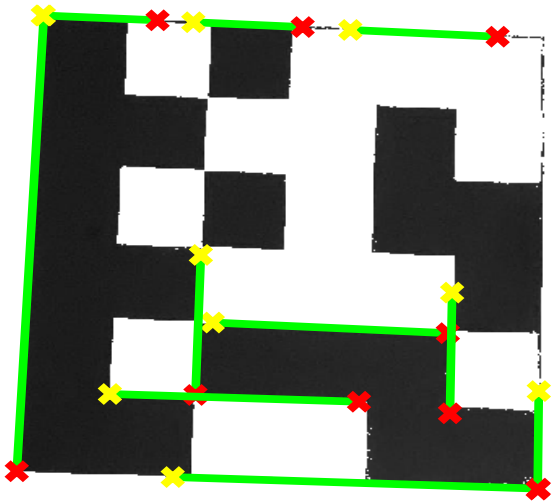
שלב 4:

Canny Edge Detector on the Binary image



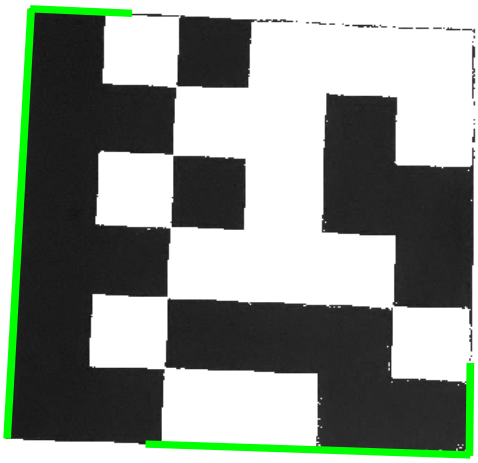
שלב 5:

All the detected lines

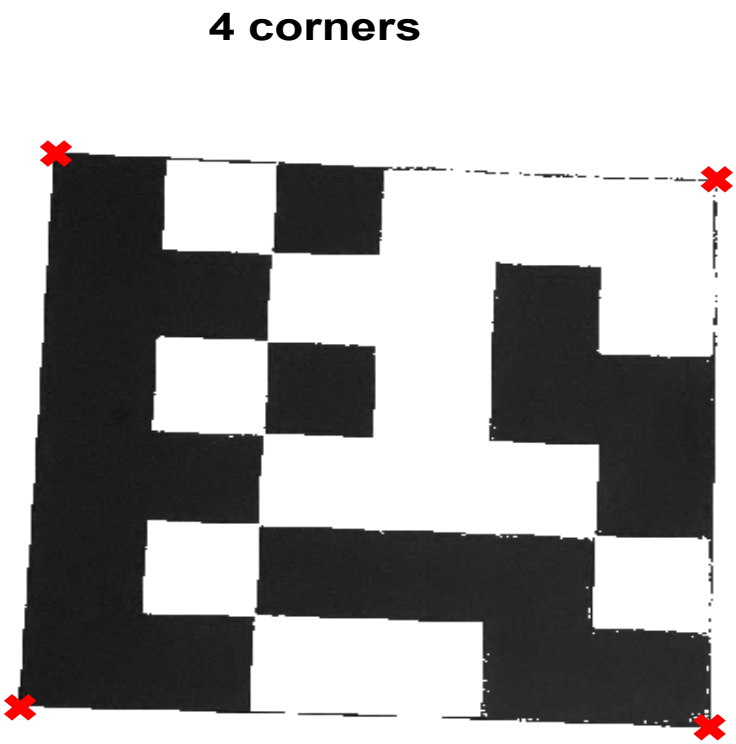


שלב 6:

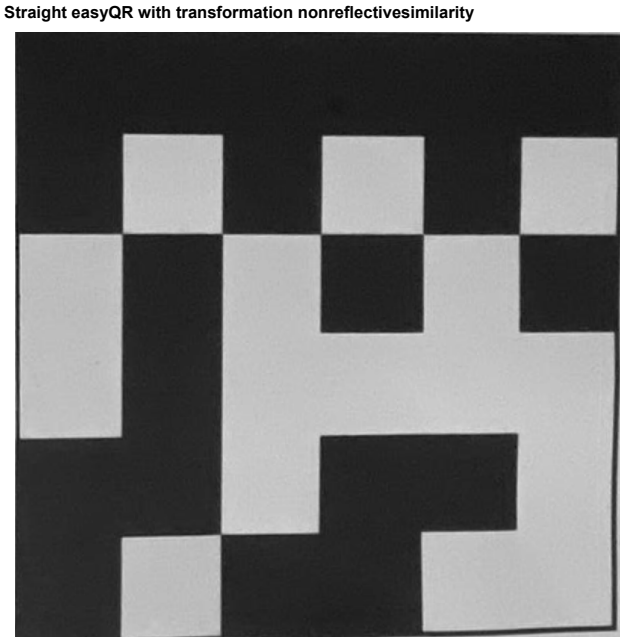
The final 4 lines



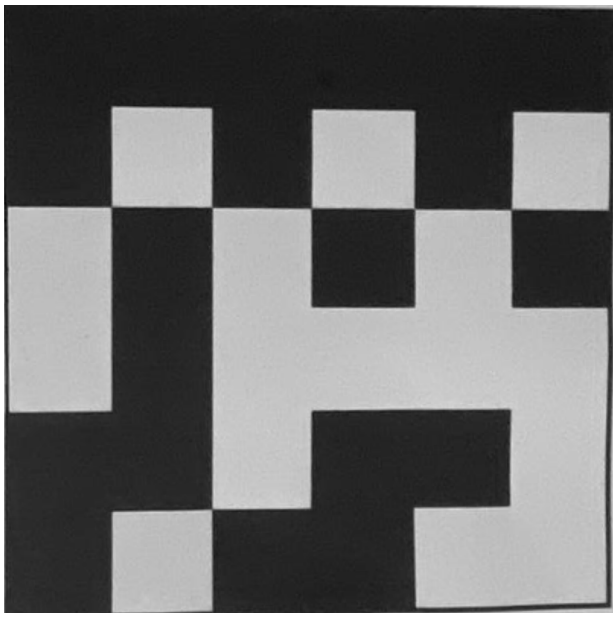
שלב 7:



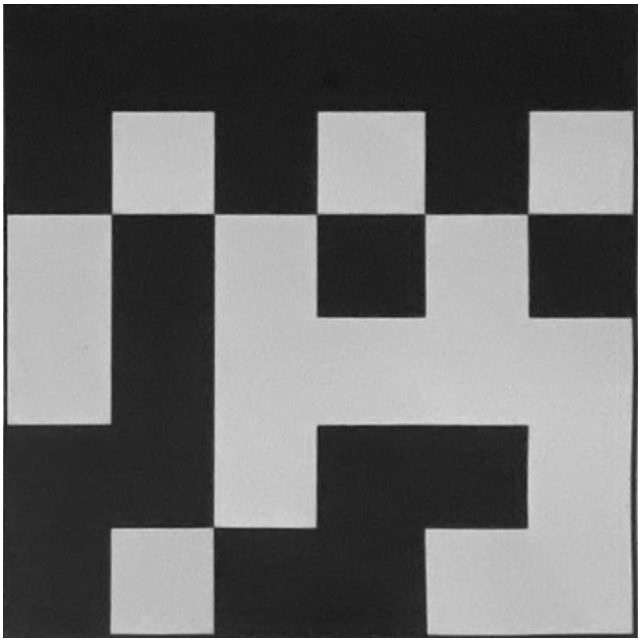
כעת נבדוק את זיהוי הפינות האוטומטי על ידי חזרה על שאלה 2, כאשר הפעם נשתמש בפינות שמצאנו עבור התמונה ברמת קושי קלה, נעשה זאת עבור כל אחת מהטרנספורמציות:



Straight easyQR with transformation affine



Straight easyQR with transformation projective



קיבלנו תמונות מיושרות בצורה מעולה, נציג את תעודת הזהות המשוחזרת על ידי כל אחת מהטרנספורמציות:

Difficulty	Transformation	Result	Success Rate
Easy	Rigid	3,1,1,3,9,4,3,5,7	**100%**
Easy	Affine	3,1,1,3,9,4,3,5,7	**100%**
Easy	Prespective	3,1,1,3,9,4,3,5,7	**100%**

קיבלנו 100% דיוק בזיהוי תעודת הזהות עבור כל אחת מהטרנספורמציות עבור התמונה כאשר השתמשנו בפינות של קוד הQR שמצאנו באופן אוטומטי על ידי האלגוריתם שבנינו.
לכן נוכל להסיק כי האלגוריתם שבנינו עובד בצורה טובה לצורך זיהוי פינות הQR.