# AR on QR

## Augmented Reality on Quick Response Code

**Aviya Levy**

**Shany shmuely**

# תוכן עניינים

# מבוא

מטרת העל  של הפרויקט היא היכולת לשלב פרסומות  בצורת מציאות רבודה (Augmented Reality-AR) עם אפליקציית מצלמה בסמארטפון.

האפליקציה תעבד את התמונה תוך כדי הצגתה, תחפש בה מרקרים כגון אייקונים וסמלים מוכרים ותציג תוכן רלוונטי.

דוגמה להמחשה:

ע"פ דרישות הפרויקט התמקדנו בהצגת AR על QR Code.


Figure 1: project motivation

# יעדים

1. בניית אפליקציית מצלמה לפלטפורמת אנדרואיד
2. זיהוי QR Code בframe הנוכחי.
3. זיהוי הטרנספורמציה המרחבית של הQR Code ביחד לQR Code מיושר – מקביל למישור המצלמה
4. פענוח המחרוזת אותה הQR Code מקודד בעזרת אפליקציות קיימות
5. הצגת AR דו מיימדי בהקשר של המחרוזת המפוענחת

# מילון מונחים

AR- Augmented Reality הוא תחום במיחשוב העוסק בשילוב אלמנטים וירטואליים עם הסביבה האמיתית בזמן אמת ובאופן אינטראקטיבי.

יישומים מסוג מציאות רבודה פועלים או על בסיס מצלמה המחוברת למחשב או על בסיס מצלמה בטלפון הנייד, כאשר המצלמה לוכדת בשלב הראשון תמונות וידאו מהעולם האמיתי. לאחר מכן יישום מיוחד מזהה את הסצנה, את מיקום המצלמה ואת הכיוון במרחב במקרה של שימוש בטלפון סלולרי. בהתבסס על מידע זה היישום מעבד את התמונה המוקרנת על גבי מסך התצוגה של הטלפון או המחשב ומשלב עליהם בזמן אמת אלמנטים וירוטואלים כגון טקסט, אובייקטים דו ממדיים או אובייקטים תלת ממדיים - או אפילו תצלומים, אנימציות, קטעי וידאו או קטעי אודיו.

Qr Code- קוד המאפשר לשמור נתונים דיגיטליים ולקודד אותם לריבוע גראפי קטן יחסית שניתן להדפיס על נייר, למשל, ושניתן לאחר מכן לקרוא בקלות ע"י כל מכשיר סלולארי נייד בעל מצלמה ואפליקציה פשוטה המותקנת עליו ויודעת לפרש את הקוד ולהציג את הטקסט, או כל נתון אחר, המוטבע בקוד ה .QR-הקוד יכול להכיל כמות גדולה של טקסט, תמונה כלשהי, קישור לאתר אינטרנט או לסרטון וידאו, כרטיס ביקור שכולל את כל פרטי הקשר, קופון הנחה למוצר כלשהו או אפילו את כל הדוגמאות ביחד ע"י קוד QR בודד. ע"י שימוש בקוד QR ובמצלמת טלפון פשוטה, המשתמש יכול לסרוק קוד גראפי ולקבל את הנתונים המוטמעים בו בצורה פשוטה ויעילה.

Finder Patterns- שלושת הריבועים המופיעים בקצוות של כל QR Code , אלו הם סימני הזיהוי של הQR Code. גודלם משתנה ביחס הפוך לגודל המחרוזת המקודדת בQR Code.

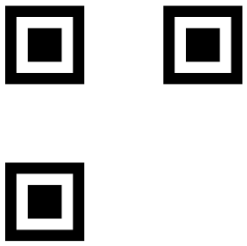


Figure 2: finder patterns

# נסיונות קודמים לעקיבה

## ניסיון מספר 1 - Vuforia

Vuforia זהו כלי שפותח ע"י qualcomm, מאפשר עקיבה אחרי עצמים דו מימדיים ותלת מימדיים. Vuforia מצליח לזהות את המארקר ומוצא את הטרנספורמציה התלת מיממדית שנדרשת להזזת האובייקט הרבוד.

**המטרה הראשונית של הפרוייקט** – להשתמש ב- Vuforia למטרות עקיבה

על המשתמש בכלי לבחור את המארקר לעקיבה, המונח המקובל הוא image target, אשר לפיו יש לגלות את הטרנספורמציה של משטח ה-QR Code.

למרבה הצער המארקרים שניתנים לעקיבה צריכים להיות ללא דוגמאות שחוזרות על עצמם, פיצ'רים שפזורים בכל המארקר, וניגוד בין הצבעים (high contrast)

QR-Code לא עונה על דרישות מהסיבה הפשוטה שמה שמשותף לכל ה-QR-Code אלו הם שלוש ה-finder patterns ואלו אינם מספיקים לשמש כתמונת עקיבה.

לכן אי אפשר להשתמש ב-QR-Code כמארקר, משמע – ניסיון השימוש ב- Vuforia נכשל

גם במקרה שנסיון העקיבה היה מצליח, העקיבה יכלה להתבצע רק אחרי finder patterns בגודל מסוים, ז"א האפליקציה תעבוד על מספר QR Code קטן יחסית - בעלי אותו גודל של finder pattersn

Vuforia מספק כלי לדירוג משטחי עקיבה –

ניתן לראות כי ה-QR-Code קיבל דירוג 0 מתוך 5:

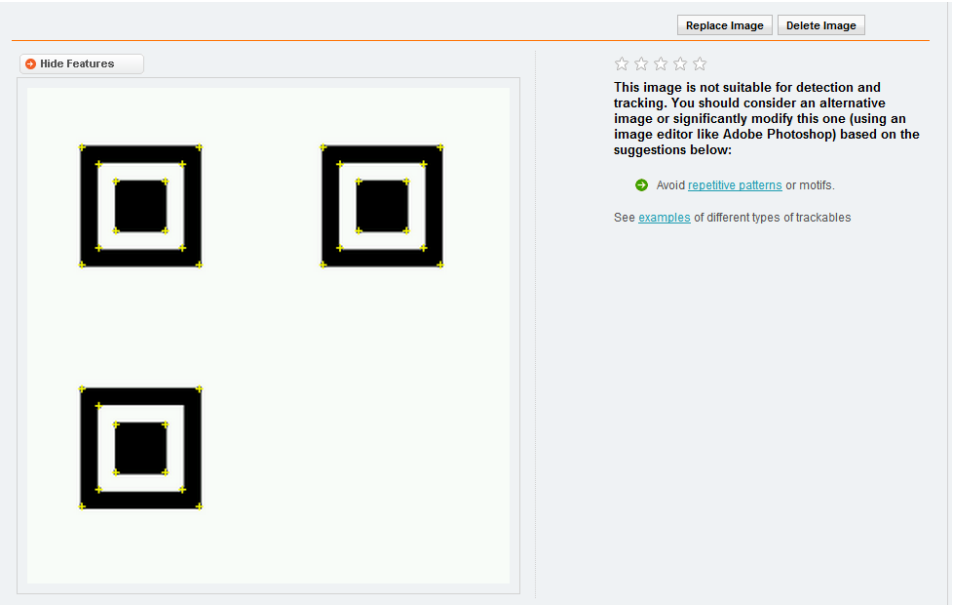



Figure 3: Vuforia grading tool

## ניסיון מספר 2 - ZXing detection

ZXing זוהי אפליקציה לזיהוי ופענוח QR Code אשר נגישה בספריית קוד פתוח ומכילה:

Detector – מחלקה לזיהוי

Decoder – מחלקה לפענוח המחרוזת המקודדת

הקו הכללי בה פועל ZXing הוא לזהות את היחס של הפיקסלים השחורים והלבנים ב-finder patterns וכך לזהות את מרכזם.

היחס המבוקש הוא 1:1:3:1:1 , ZXing מחפש את היחס הזה שורה אחר שורה

על האפליקציה לתמוך בהטיית מצלמת המשתמש בזוויות גדולות יחסית על מנת שנוכל באמת להציג Augmented Reality (אם המסך אינו מוטה AR מוצגת באופן הריק – כתמונה רגילה ללא הטיה והתאמה למציאות).

כשאר מישור הQR Code אינו מקביל למישור הפלאפון היחס (1,1,3,1,1) לא נשמר. ולכן טווח הזיהוי בהטיה הוא יחסית מצומצם.

הנה דוגמא להטיה שבה היחס המתקבל אינו מתאים ליחס המתבקש:

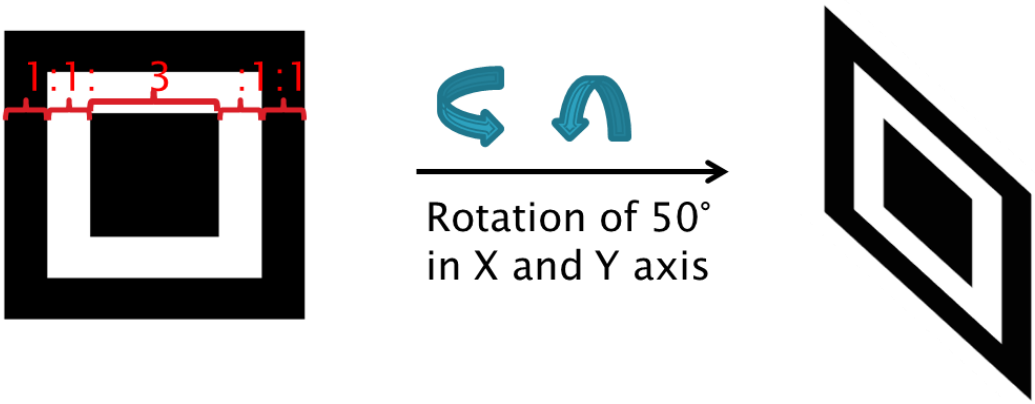



Figure 4: Ratio in rotation

## ניסיון מספר 3 - Hough Lines algorithm

זהו אלגוריתם אשר מזהה קווים ישרים ומהם משליך מה יהיו הפינות

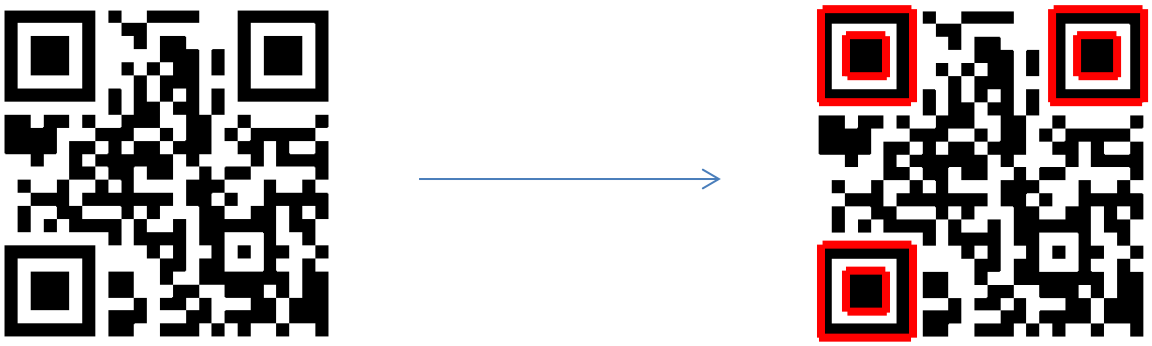


Figure 5: Hough Lines abilities

האלגוריתם מתגבר על הבעיה עם זוויות גדולות של הטיה שהייתה לנו ב-ZXing כיוון שקווים ישרים נשמרים גם בזוויות הטיה גדולות.

ניתן לראות את פעולת האלגוריתם בזווית הטיה גדולה יחסית:

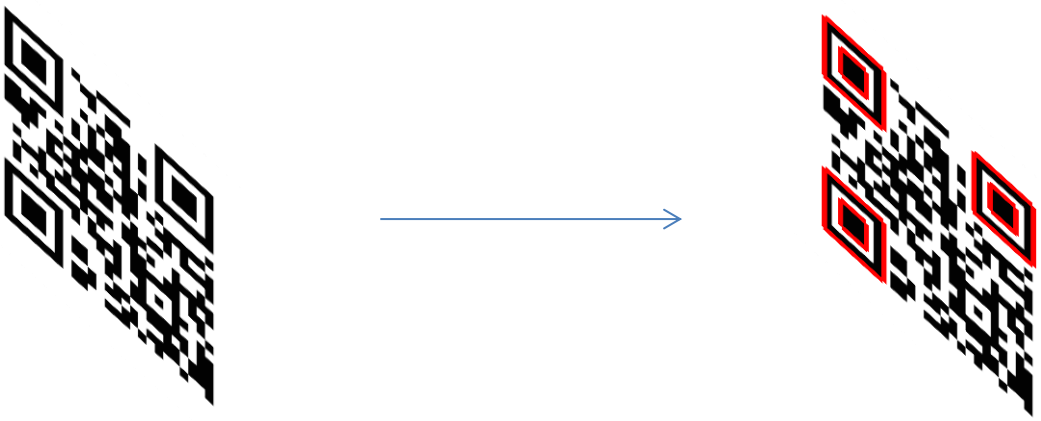


Figure 6: Hough Lines proves itself at large angles

על מנת לזהות את הקווים ב-finder patterns כאשר הQR Code במרחק סביר מהמצלמה יש להוריד מאוד את הסף לזיהוי קו (קלט של האלגוריתם, עד כמה "עקמומי" יכול להיות הקו על מנת שייחשב קן)

הנה דוגמא לסף מתאים לזיהוי הקווים של ה-QR Code

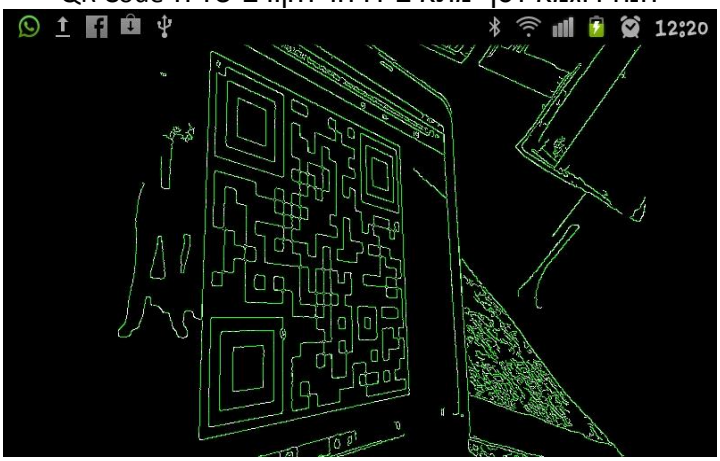



Figure 7: Hough Lines Algorithm on QR Code

כמובן שככל שמורידים את הסף, כך מזהים יותר קווים, לכן בתמונה יזוהו בנוסף הרבה קווים פרט ל-QR Code.מה שמקשה על הזיהוי ומאריך את זמן הריצה מאוד:

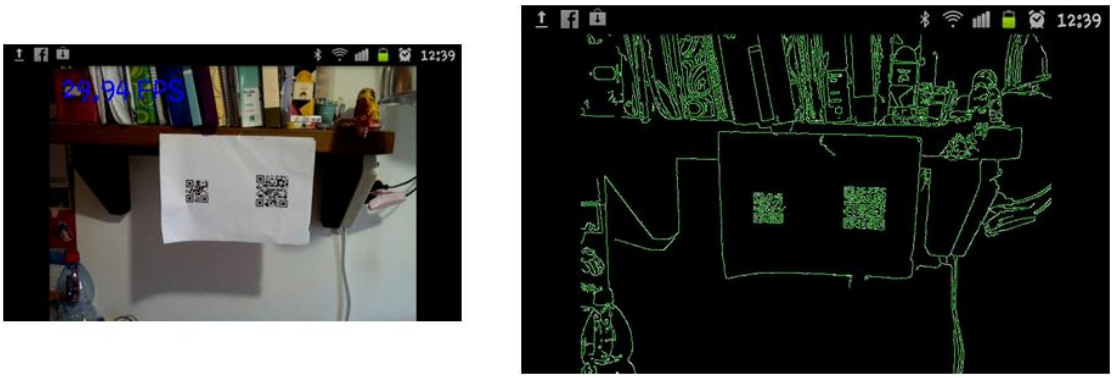
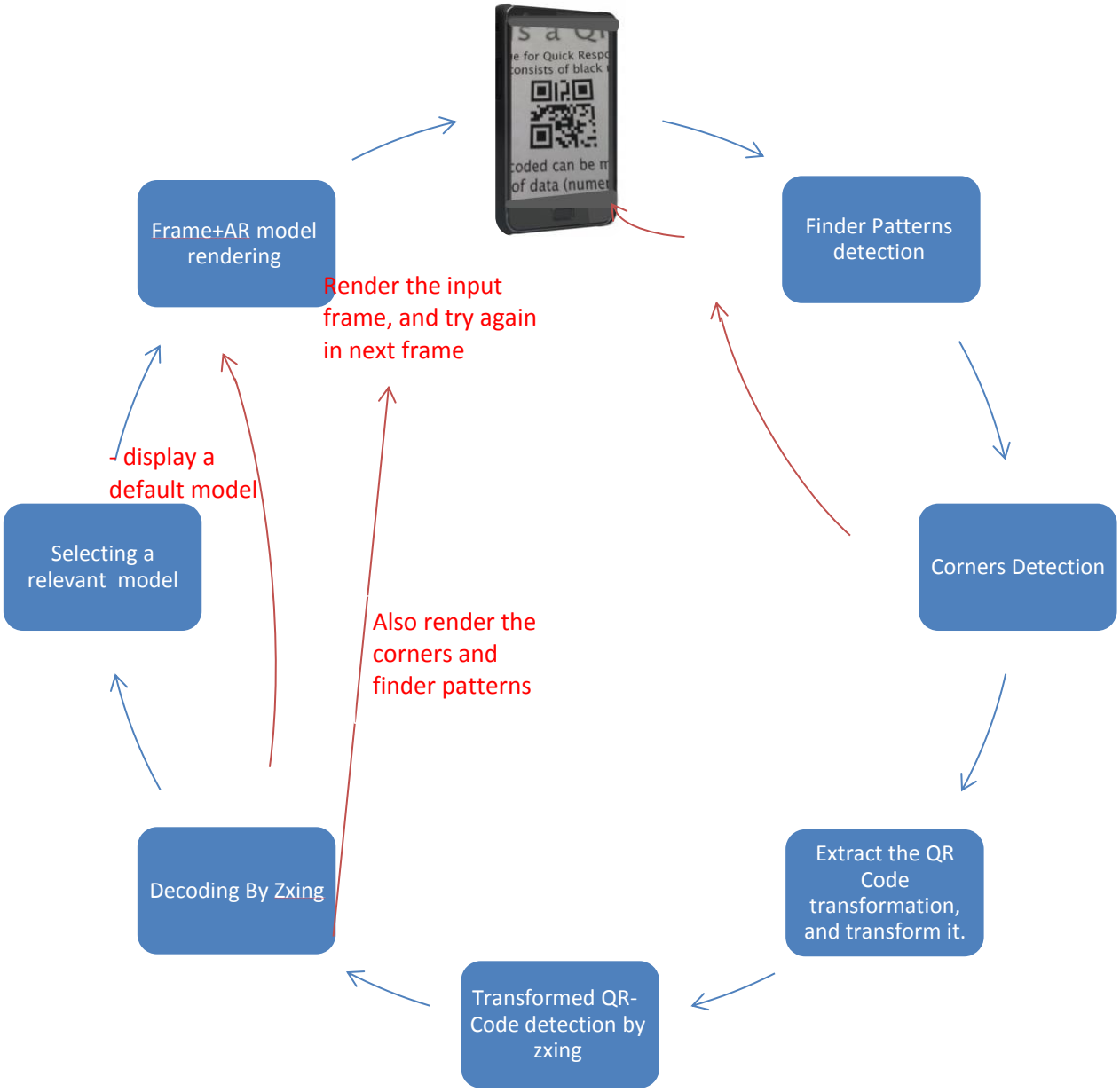



Figure 8: Hough Lines Algorithm on QR Code with background details

# נסיון מוצלח לזיהוי QR-Code מוטה - זרימת המידע במערכת

להלן תרשים של זרימת המידע במערכת, כל שלב מקבל את הקלט מהשלב הקודם והפלט הוא הקלט לשלב הבא, המידע יתקדם בכיוון החצים הכחולים במקרה של הצלחה, אחרת המידע יזרום בכיוון החצים האדומים.



# שלבי האלגוריתם

## Finder Patterns detection

*קלט-* תמונה שנקלטה במצלמה.

*פלט-* שלוש קואורדינטות על התמונה במיקומים שזוהו כ- finder patterns.

*מוטיבציה לאלגוריתם -*

נסתכל על המבנה הגיאומטרי של הfinder patterns, כל finder pattern מורכב מקו מתאר חיצוני שחור של מרובע, ובתוכו עוד מרובע שחור (ראה איור 2).

מרכזי המאסה הדו מימדים של שני המרובעים הללו צריכים להיות קרובים אחד לשני, גם אם ה-QR Code מוטה מאוד.

*אלגוריתם-*

- נבצע בינאריזציה לתמונה – כל פיקסל יסווג כ-0 או 1
- נפעיל אלגוריתם לזיהוי רכיבים קשירים שחורים בתמונה
- נחשב מרכזי מאסה של כל רכיב.
- נמצא 3 זוגות של רכיבים קשירים בעלי מרכזי המאסה הכי קרובים.
- נחזיר מרכז מאסה מכל זוג- סה"כ 3 קואורדינטות.

*מימוש האלגוריתם-*

נפרט את אופן מימוש האלגוריתם למציאת רכיבים קשירים.

מבני הנתונים הנדרשים:

Bitmap - 2D bool array representing the binary frame.
Label - 2D array in size of Bitmap.size(), save the label of every pixel

UF -Union find data structure.

בנוסף נשמרים 2 אינדיקטורים המתוחזקים עבור כל פיקסל:

connectedToLeft - true if the current pixel and its left neighbor are black.
connectedToUp - true if the current pixel and its top neighbor are black.

נעבור 2 מעברי סריקה על התמונה, בשלב ראשון נבצע סיווג פיקסלים לרכיבים קשירים ותחזוק הרכיבים. במעבר השני נתקן פיקסלים שסיווגם התברר כשגוי.

מעבר ראשון:

עוברים על התמונה במעבר raster, ומסווגים כל פיקסל על סמך מידע מהשכן העליון והשמאלי.
בנוסף מתחזקים מבנה UF של מזהיי הרכיבים, אם השכן השמאלי והעליון צבועים בשחור מאחדים אותם.

```
for( int i=0; i<Bitmap. Width, i++)
for( int j=0; j<Bitmap. Height, j++)
{
    if( Bitmap.get(i,j)  == BLACK )
    {
        if( connectedToLeft )
        {
            Label.Set(i, j, Label.getLeft(i,j));
            if( connectedToUp )
            {
                UF.Union( Label.getLeft(i,j) , Label.getUp(i,j) );
            }
        } else if( connectedToUp )
        {
            Label.Set(i, j, Label.getUpLeft(i,j) );
        } else
        {
            Label.Set(i, j , Label.NewLabel());
        }
    }
}
```

מעבר שני:

במעבר השני מתקנים פיקסלים שהמזהה שניתן להם שגוי.
תיקון פיקסל יכול להיווצר למשל בתמונה הבאה:

Figure 4.1:Binary image

| | | |
|---|---|---|
| 1 | - | 2 |
| 1 | - | 2 |
| 1 | 1 | 1 |

Figure 3.2:First pass result

| | | |
|---|---|---|
| 1 | - | 1 |
| 1 | - | 1 |
| 1 | 1 | 1 |

Figure 2.3 Second pass result

```
for( int i=0; i<Bitmap. Width, i++)
for( int j=0; j<Bitmap. Height, j++)
{
    Label.Set(i, j , UF.Find(Label.get(i,j)));
}
```

השתמשנו בUF במימוש עם עצים הפוכים וכיווץ מסלולים על מנת להאיץ את הקוד.

## Corners Detection - זיהוי ארבעת הפינות של ה QR Code

*קלט-* שלושת מרכזי ה -finder patterns, בסדר שרירותי.

*פלט-* ארבע קואורדינאטות של פינות הQR Code.

*מוטיבציה לאלגוריתם-* על מנת למצוא את הטרנספורמציה המישורית שה-QR Code עבר בתמונה אנו צריכים למצוא 4 נקודות התאמה בין QR Code מיושר (מקביל מישור המצלמה) לבין QR Code מוטה (בזווית למישור המצלמה)

*האלגוריתם-*

שלב ראשון: סידור שלושת ה-finder patterns.

שלב שני: מציאת 8 נקודות על המסגרת של הQr Code.

שלב שלישי: מציאת 4 הפינות על סמך שמונת הנק' משלב שני.

### שלב ראשון- סידור שלושת ה-finder patterns:

נרצה לסווג את שלושת הנקודות לפינה הימנית העליונה (- p1) פינה שמאלית עליונה(-p0) פינה שמאלית תחתונה (-p2) .

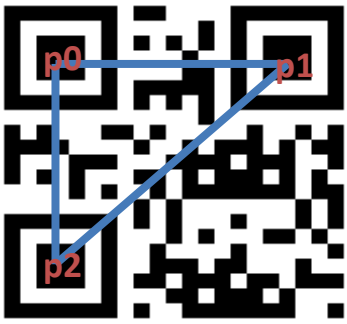



Figure 9: finder pattern classification

- נחשב את שלושת הזויות במשולש.
- נקבע את הקודקוד בעל הזווית הגדולה ביותר להיות p0.
- נקבע שרירותית את שני הקודקודים שנותרו להיות p1 ו p2.
- נסמן $v1=(p1_x-p0_x, p1_y-p0_y,1)$ , $v2=(p2_x-p0_x, p2_y-p0_y,1)$
- ונחשב את $W=V2\times V1$
- אם $W_z<0$ נחליף בין p1 ל p2.

*לעיתים p1 או p2 נקבעות להיות הנקודות עם הזווית הגדולה ביותר, מצב זה קורה כאשר הQR Code מוטה בצורה יוצאת דופן שלא קורית בדרך כלל.

## שלב שני- מציאת 8 נקודות על המסגרת של הQr Code:.

נרצה למצוא 8 נקודות על מסגרת הQR Code, אחרי שנמצא אותם נוכל למצוא את ארבעת הפינות של ה-QR-Code.

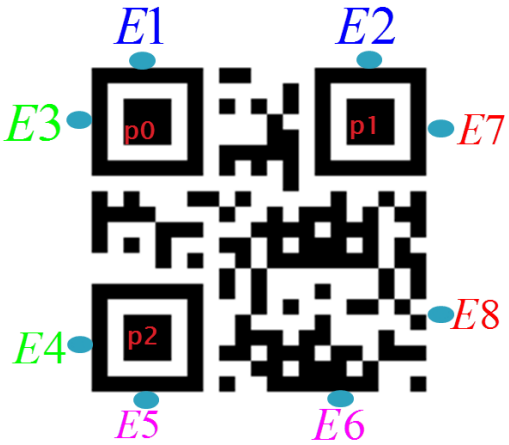



Figure 10: 8 points on frame.

**נקודות חשובות להבהרה:**

- אנו לא משתמשים בשמונת נקודות אלו למציאת הטרנספורמציה מכיוון של QR Codes שונים יש finder patterns בגודל שונה, ולכן מיקומם ישתנה ביחס למסגרת הQR Code.

אנו מחפשים נקודות כאלו שנוכל להתאים לאיזשהו ריבוע בגודל קבוע ולכן ארבעת נקודות המסגרת הן נקודות מתאימות יותר - נוכל פשוט להתאים אותם לארבעת הפינות של מרובע בגודל מסוים.

- כזכור התמונה שאני עובדים עלייה עברה בינאריזציה, כאשר פיקסל שחור מסומן על ידי 1 , ולבן ע"י 0.
- בשלב זה קודם נמצא את 6 הנקודות ששייכות לfinder pattern (E1, E2, E3, E4, E5, E7).
- ואחר-כך בעזרת ששת נקודות אלו נמצא את הנקודות שלא שייכות לשום finder pattern (E7,E8).

### אלגוריתם למציאת נקודה שקרובה לfinder pattern:

לשם פשטות נדגים את האלגוריתם על הנקודה E1

הקלט לאלגוריתם הוא ווקטור ההתקדמות V נציב בו את הווקטור $p0 - p2$

ונקודה להתחלת דגימה X נציב בו את הנקודה P0

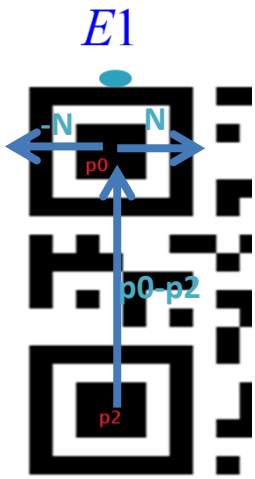



Figure 11

- נסמן את N להיות ווקטור ניצב לווקטור V .
- נדגום מעט פיקסלים החל מהנקודה X בכיוון N ובכיוון N-.
- כל עוד ממוצע צבע הפיקסלים הנדגמים גדול מ-0.2
  - הזז את קו הדגימה בכיוון V ודגום מחדש.
- כל עוד ממוצע צבע הפיקסלים הנדגמים קטן מ-0.8
  - הזז את קו הדגימה בכיוון V ודגום מחדש.
- כל עוד ממוצע צבע הפיקסלים הדגומים גדול מ0.2
  - הזז את קו הדגימה בכיוון V ודגום מחדש.
- קבע את הנקודה האמצעית של הקו האחרון שנדגם להיות E1.

באופן זהה אנו מספקים ווקטור התקדמות V ונקודת התחלת הדגימה X מתאימים עבור כל נקודה מתוך 6 הנקודות:

| נקודה | E7 | E5 | E4 | E3 | E2 | E1 |
|---|---|---|---|---|---|---|
| V | p1-p0 | p2-p0 | p0-p1 | p0-p1 | p0-p2 | p0-p2 |
| X | p1 | p2 | p2 | p0 | p1 | p0 |

### אלגוריתם למציאת נקודה שלא נמצאת על finder pattern

לשם פשטות נדגים את האלגוריתם על הנקודה E6

הקלט לאלגוריתם הוא ווקטור ההתקדמות V ניקח אותו להיות $p2-p0$ ושתי נקודות שייקבעו לנו את קו הדגימה הראשון X,Y - ניקח אותם להיות E5 ו p1 בהתאמה.

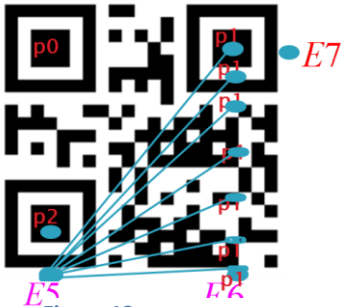


Figure 12

- דגום את הקו המחבר בין X ל Y.
- כל עוד ממוצע צבע הפיקסלים הדגומים גדול מ0.2
  - הזז את X בכיוון V ודגום מחדש.
- קבע את E6 להיות X.

נעשה כנ"ל ע"מ למצוא את הנקודה E8, עם פרמטרים $p1-p0=V$ , E7=Y, p2=X

## שלב שלישי- מציאת ארבעת הפינות

לאחר שיש בידנו את שמונת הנקודות E1-E8, נמצא את ארבעת הפינות c1,c2,c3,c4:

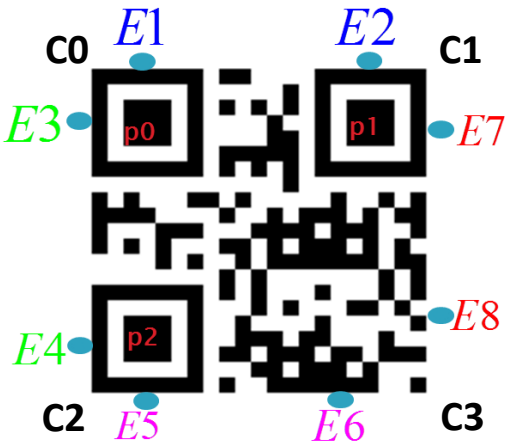



Figure 13: 8 edge point and 4 corners

לשם פשטות נדגים את האלגוריתם על הנקודה C0.

הקלט לאלגוריתם- ארבעת הנקודות היושבות על 2 הקווים שהפינה Ci משותפת להם. עבור C0 אלו הן הנקודות E1,E2,E3,E4.

### מציאת הנקודה C0

- נמצא את מקדמי משוואת הקו עליו יושבות הנקודות E1,E2 ע"י חישוב:

$$(E1_x, E1_y, 1) \times (E2_x, E2_y, 1) = (a_1, b_1, c_1)$$

- נמצא את מקדמי משוואת הקו עליו יושבות הנקודות E3,E4 ע"י חישוב:

$$(E3_x, E3_y, 1) \times (E4_x, E4_y, 1) = (a_2, b_2, c_2)$$

- נחשב $(a_1, b_1, c_1) \times (a_2, b_2, c_2) = (x, y, z)$
- נחשב $(x, y, z) / z = (C0_x, C0_y, 1)$
- נחזיר $C0 = (C0_x, C0_y)$

באותו אופן נחשב את שאר הפינות.

**הוכחת נכונות האלגוריתם-**

ניקח נקודה $(x,y)$ שנמצאת על ישר $ax+by+c=0$. אפשר לכתוב את המשוואה בצורה וקטורית

$(a,b,c)\cdot(x,y,1)=0$. ז"א הווקטור $(x,y,1)$ מאונך לווקטור $(a,b,c)$.

אם נמצא עוד נקודה $(x',y')$ שנמצאת על הישר, כמובן שגם במקרה זה הווקטור $(x',y',1)$ יהיה מאונך לווקטור $(a,b,c)$.

מכפלה וקטורית בין שני וקטורים מניבה את הווקטור המאונך לשניהם ולכן $(x',y',1)\times(x,y,1)=(a,b,c)$.

בכיוון ההפוך- ניקח 2 קווים $(a,b,c),(\alpha,\beta,\gamma)$ ונרצה למצוא את נקודת החיתוך של שני הישרים $(x,y)$.

מתקיימות שתי המשוואות $(a,b,c)\cdot(x,y,1)=0,\ \ (\alpha,\beta,\gamma)\cdot(x,y,1)=0$

ז"א הווקטור $(x,y,1)$ מאונך לשניהם.

נחשב $(a,b,c)\times(\alpha,\beta,\gamma)=(x',y',z')$, נקבל ווקטור המאונך לשני משוואות הקו הישר.

נרצה את הנקודה כאשר $z=1$ ולכן ננרמל את הוקטור - נחלק ב Z ונקבל את הנקודה $(x,y,1)$.

## **QR Code** transformation - מציאת ההעתקה בין QR Code מיושר (מקביל למישור המצלמה) לבין הQR Code הנקלט במצלמה.

*קלט-* ארבעת פינות הQR Code.

*פלט-* מטריצת ההעתקה + תמונה של ה-frame הנקלט במצלמה לאחר ביצוע ההעתקה ליישור ה-QR Code.

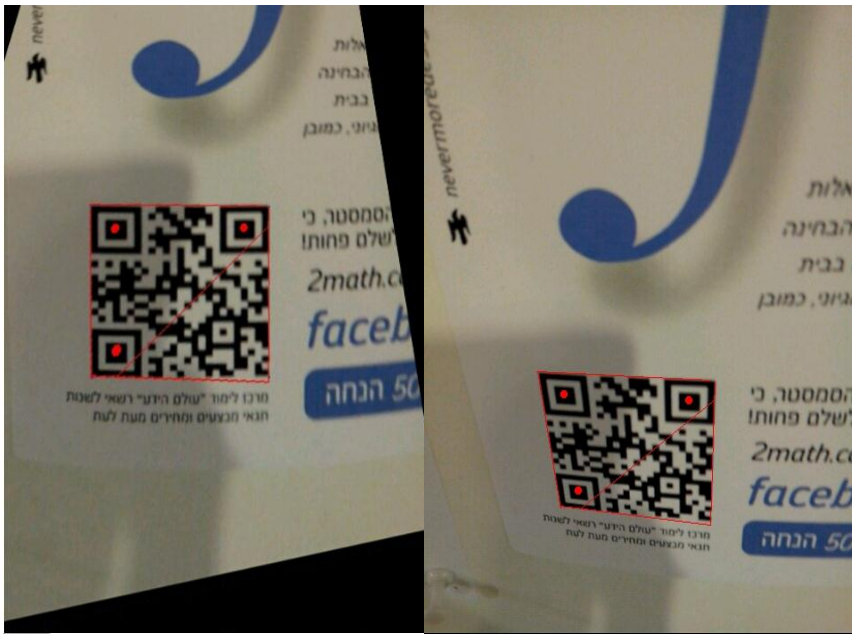


Figure 14: The source (right) image and the transform (left) image

בשלב זה נשתמש באלגוריתם קיים של OpenCV למציאת הומוגרפיה (*findHomography*)- העתקה המתארת מה קורה לאוביקט דו מימדי כשאר נקודת התצפית עליו משתנה.

ננסה להעתיק את הQR Code כפי שהוא נראה מעין המצלמה, לריבוע בגודל קבוע מראש.

על מנת למצוא את ההומוגרפיה נצטרך לפחות 4 נקודות התאמה בין הQR Code לריבוע.

מצאנו בשלב קודם את ארבעת הפינות של הQR Code (C0,C1,C2,C3),כעת נעתיק אותם לריבוע מיושר, המקביל לעין המצלמה:

$$[C0,(0,0)\ ],[C1,(0,255)\ ],[C1,(255,0)\ ],[C3,(255,255)\ ]$$

אחרי מציאת מטריצת ההומוגרפיה, נשתמש בפונקציה נוספת של OpenCV (*warpPerspective*) שמפעילה טרנספורמציה על תמונה. כך נקבל תמונה של הQR Code המיושר, באופן דומה ניתן להפעיל את הטרנספורמציה ההפוכה ולהשרות את הטרנספורמציה על אובייקט דו מימדי כך שיהיה מונח על ה-QR Code בתצורה של AR.

## Decoding By Zxing- פענוח תוכן הQR Code

*קלט-* תמונה של ה-frame הנקלט במצלמה לאחר ביצוע ההעתקה ליישור ה-QR Code

*פלט-* מחרוזת אשר ה-QR Code מקודד

בידנו QR-Code המקביל למישור הפלאפון, ולכן ZXing יכול לפענח את המחרוזת.

(-אפליקציה לזיהוי ופענוח QR Code אשר נגישה בספריית קוד פתוח כפי שתואר בניסיון מספר 2)

התמונה נשלחת למחלקה Detector אשר מזהה שקיים QR Code ב-frame וממנה למחלקה Decoder אשר מפענחת את תוכן ה-QR Code ופולטת מחרוזת.

יש לציין כי זיהוי מוצלח של הQR-Code לא גורר פענוח מוצלח.

זיהוי מוצלח של הQR-Code (גם כאשר הפענוח נכשל) גורר מקסנה שכל שרשרת השלבים עברה בהצלחה, ולכן גם אם אין את תוצאות הפענוח, נוכל להציג AR דיפולטיבי.

כאשר הזיהוי מוצלח והפענוח מוצלח - נעבוד לשלב הבא.

כאשר הזהוי מוצלח והפענוח לא מוצלח - נדלג על שלב הבא ונשתמש בתמונה דיפולטיבית (סמיילי)

כשאר הזיהוי לא מוצלח -נדלג על שלב הבא ונציג רק את המסגרת של הQR-Code ואת הfinder patterns.

## Selecting a relevant model

*קלט-* מחרוזת אשר ה-QR Code מקודד

*פלט-* תמונה המתאימה למחרוזת (במידה וקיימת)

לעת עתה יש לנו מאגר מידע סופי של תמונות, אם קיבלנו מחרוזת אשר אינה נמצאת במאגר המידע נציג תמונה מסוימת כברירת מחדל - סמיילי. מימשנו מבנה נתונים פשוט של map הממפה מחרוזת לתמונה בגודל 256*256.

אופציה לשיפור היא לקודד מחרוזת של קישור ברשת אליו האפליקציה תוכל לגשת ולהוריד את התוכן להצגה על ה QR Code.

## Frame+AR model rendering

*קלט-* תמונה המתאימה למחרוזת + מטריצת ההעתקה

*פלט-* frame של התמונה הקיימת + התמונה המתאימה בתצורה של AR

בשלב הזה אנו נדרשים להציג את התמונה הרלוונטית ל-QR Code מעליו, כלומר בנוסף ל-frame של המצלמה.

יש בידינו את התמונה הקיימת, כל שעלינו לעשות הוא להציג אותה מוטית לפי ההטיה של ה-QR Code בכל רגע – זוהי בדיוק המשמעות של AR

באותה הדרך שיישרנו את ה-QR Code למשטח המקביל למצלמה בעזרת מטריצת ההומוגרפיה, כעת נפעיל את המטריצה ההפוכה על התמונה, אשר כמובן במצבה הטבעי נראית מקבילה למישור המצלמה, ונקבל את התמונה כך שמושרה עליה המישור בו נמצא ה-QR Code ביחס למצלמה.

בנוסף אנו מציגים את ה- frame של המצלמה, לכן מתקבלת אשליה שהתמונה מונחת מעל ה-QR Code .

# תוצאות

אלו הם צילומי מסך בזמן ריצת האפליקציה.
בראש התמונה בכיתוב כחול ניתן לראות את המחרוזת המקודדת.
ועל הQR-Code ניתן לראות את הלוגו המשוייך למחרוזת.

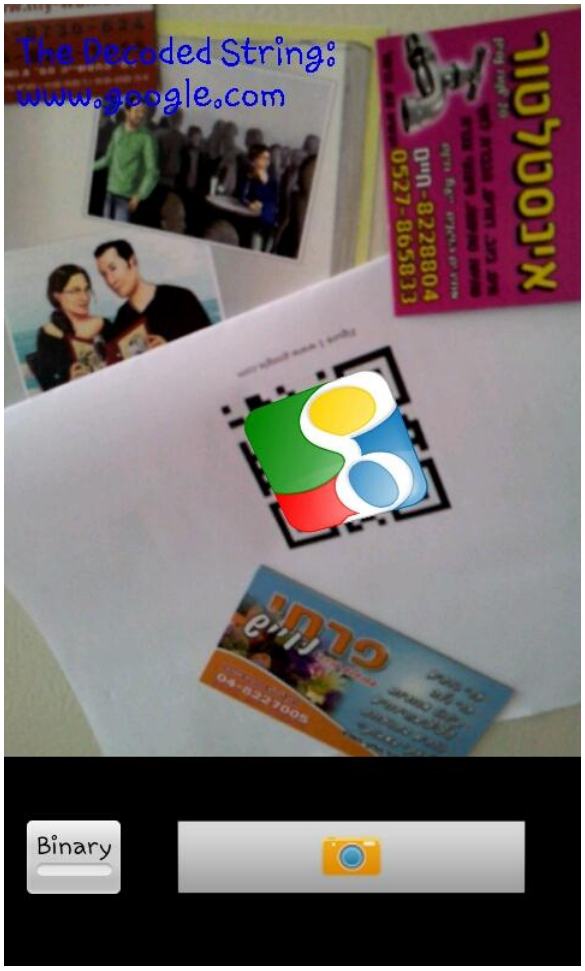



Figure 15: Google icon, note the many details in the frame

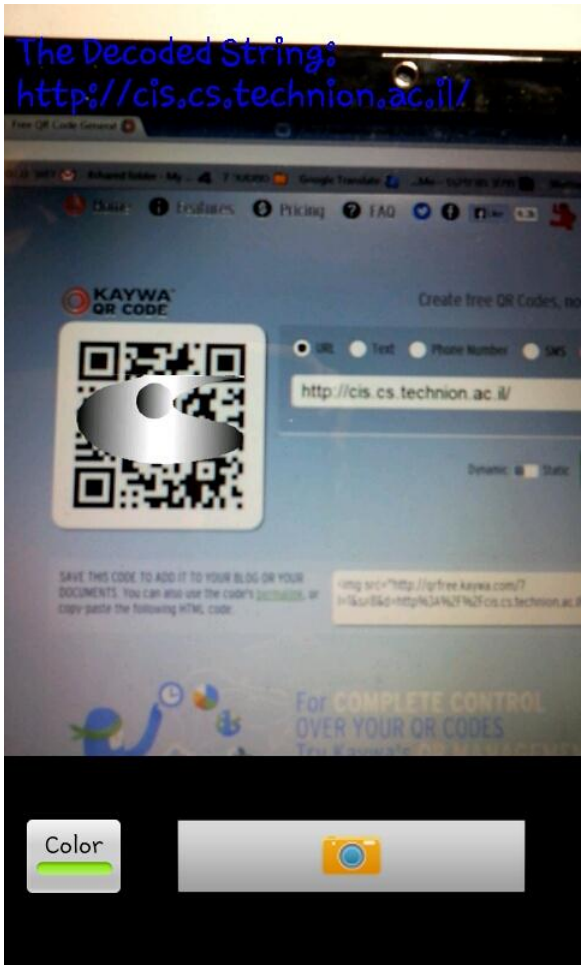



Figure 16: Here you can see the QR-Code generator tool - the URL input right the QR Code output. The same string appears at the top as our application output.

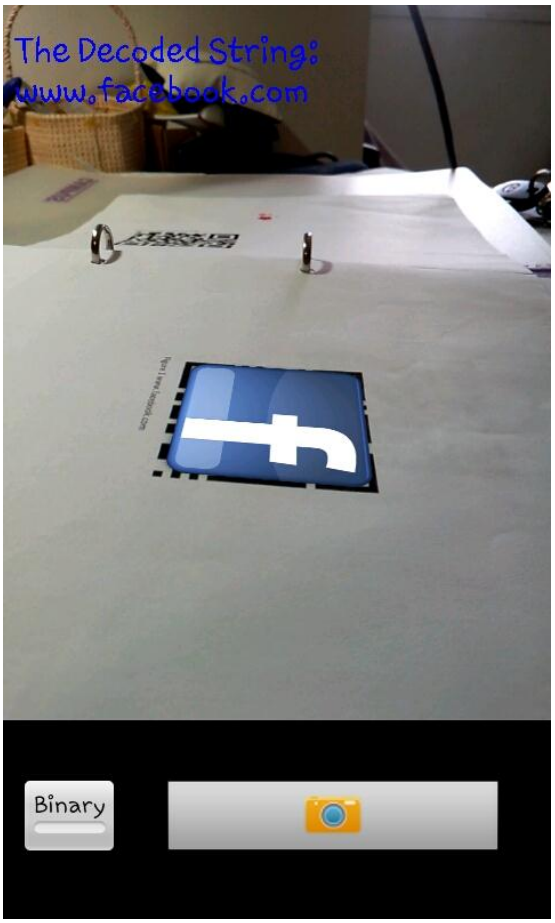



figure 17: facebook icon on a tilt QR Code


figure 18: ynet icon on a tilt QR Code


figure 19: example of seccessful detecting and abortive decoding (ZXing result). "can't decode QR" string and a defualt icon appear.


figure 20: BURGERKING icon.

# Design

הקוד בנוי משלושה packages:

***Main*** - אחראי על השכבה החיצונית:

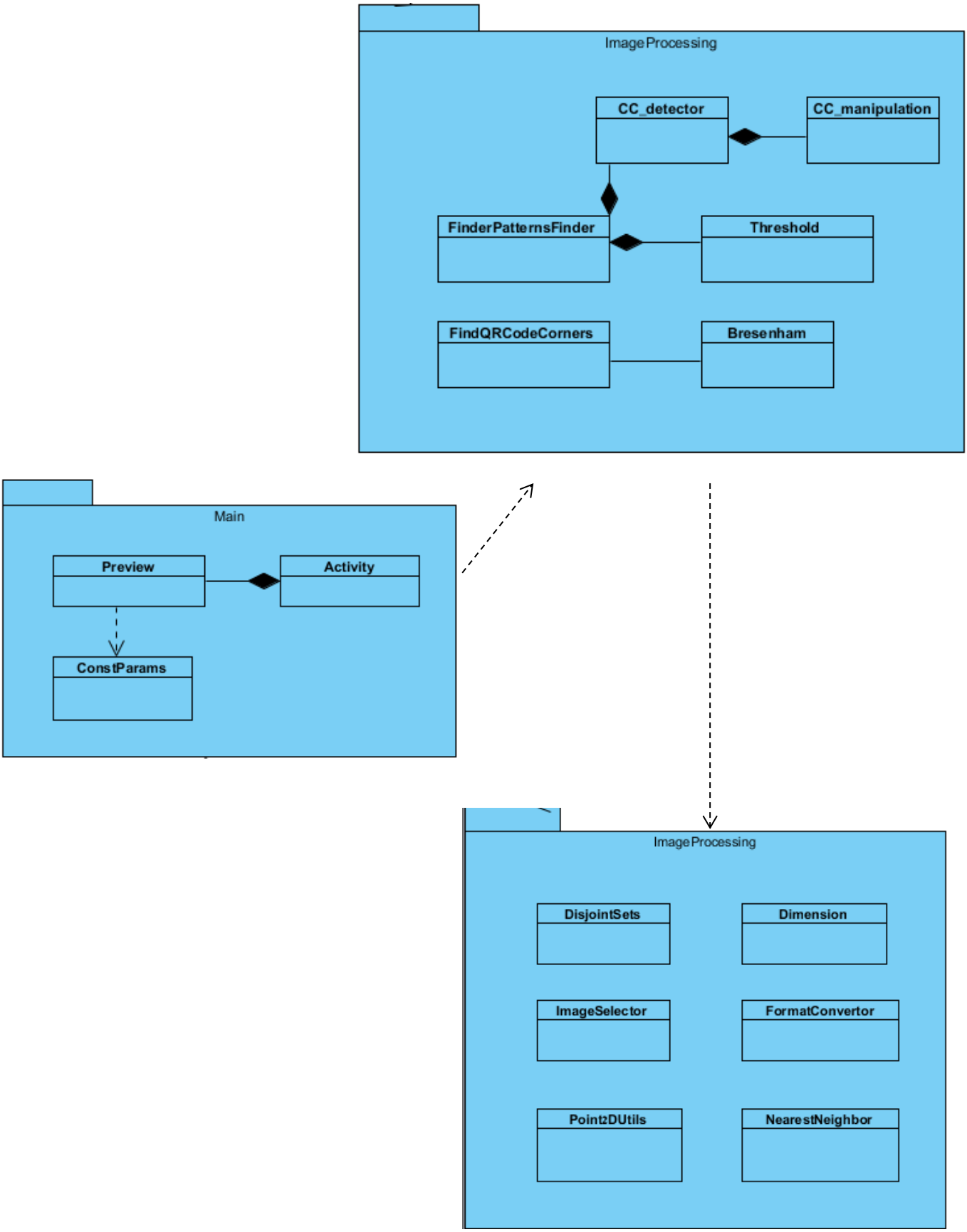
- UI
- מימוש של מחלקת Activity
- לוגיקת התצוגה - במחלקת Preview

***ImageProcessing***- בו ממומשים אלגוריתמי עיבוד תמונה:

- בינאריזציה
- מציאת הרכיבים הקשירים
- עיבוד הרכיבים הכשירים
- מציאת finderPatterns
- מציאת פינות הQR Code
- אלגוריתם למציאת הקו הישר בין 2 נקודות.

***Utiles***- בו נמצאים מבני נתונים ויכולות שונות שנדרשות

- מבנה מתוחכם של union find
- המרת תמונה לפורמט 8888RGBA
- לוגיקה של בחירה והחזקה של האובייקט הדו מימדי להצגה
- אלגוריתם למציאת השכן הקרוב ביותר במרחב
- ייצוג של נקודה במרחב ומימוש אופרטורים שונים


ImageProcessing
CC_detector
CC_manipulation
FinderPatternsFinder
Threshold
FindQRCodeCorners
Bresenham
Main
Preview
Activity
ConstParams
ImageProcessing
DisjointSets
Dimension
ImageSelector
FormatConvertor
Point2DUtils
NearestNeighbor

# מגבלות האפליקציה

### בינאריזציה ע"פ סף קבוע-

אנו מבצעים את הבינאריזציה לתמונה ע"פ רף קבוע בכל frame, מה שיוצר בעייה כאשר התמונה שנקלטת מהמצלמה חשוכה/בהירה מאוד.
במצבים אלו התמונה שתתקבל לאחר הבינאריצזיה תהייה שחורה/לבנה בהתאמה ולא נוכל להצליח להבחין בין ה-QR Code לרקע.
הפתרון הוא פשוט- סף דינאמי. משתמשים באלגוריתם דינאמי שבסביבה כהה מנמיך את הרף ובסביבה בהירה מגביהה את הרף.
לא השתמשנו באלגוריתם זה כי זמן הריצה שלו ארוך.

הוספנו לאפליקציה תפריט אפשרויות, וכך המשתמש יוכל לשלוט על הסף לבינאריזציה, בנוסף הוספנו כפתור שהופך את הסטנה לבינארית כך שהמשתמש יוכל לקבל חיווי על הרף ששינה.

### גודל ה Finder Patterns-

הגבלנו מלמטה את גודלם של הfinder patterns.
אם רכיב כשיר (ריבוע חיצוני או פנימי) בfinder pattern קטן מ45 פיקסלים, האפליקציה תיכשל בזיהוי.
הסיבה היא פשוטה, האלגוריתם למציאת הfinder patterns מסתמך על רכיבים קשירים שמרכז המאסה שלהם קרוב אחד לשני, התמונה לא עוברת ניקוי רעשים לכן ייתכן מצב של 2 רכיבים קשירים שחורים קטנים מאוד שקרובים אחד לשני והם בכלל רעש בתמונה- נרצה להימנע ממצב כזה ולכן הגבלנו את גודל הרכיבים הקשירים.

### זמן הריצה-

זמן הריצה ארוך (200-700 מילי לכל frame), זמן הריצה הארוך נובע בעיקר מהאלגוריתם למציאת רכיבים קשירים וההפעלה מאסיבית במבנה הנתונים Union Find.

# פרוייקט המשך

## יעדים

1. שיפור זמן הריצה של האלגוריתם
2. הצגת AR תלת מיימדי

## הדרך להשגת היעדים

1. שימוש במבני הנתונים של java יכול להוות בעייה לזמן הריצה, לכן אופציה לשיפור היא להשתמש בNDK ולכתוב חלק מהקוד בC- אפשרויות לשכתוב הם: בינאריזציה לתמונה / מימוש Union Find / מימוש כל האלגוריתם למציאת רכיבים קשירים.
2. על מנת להציג AR תלת מיימדי נשתמש בSDK שנקרא ARToolkit. כלי זה יודע לזהות מארקרים דו מימדיים בתמונה ולחלץ את הטרמספורצמיה התלת מיימדית הנדרשת להזזת אובייקט תלת מיימדי.
המארקרים שכלי זה יודע לזהות בעלי צורה זהה: ריבוע שחור שעוטף צורה שחורה נוספת.

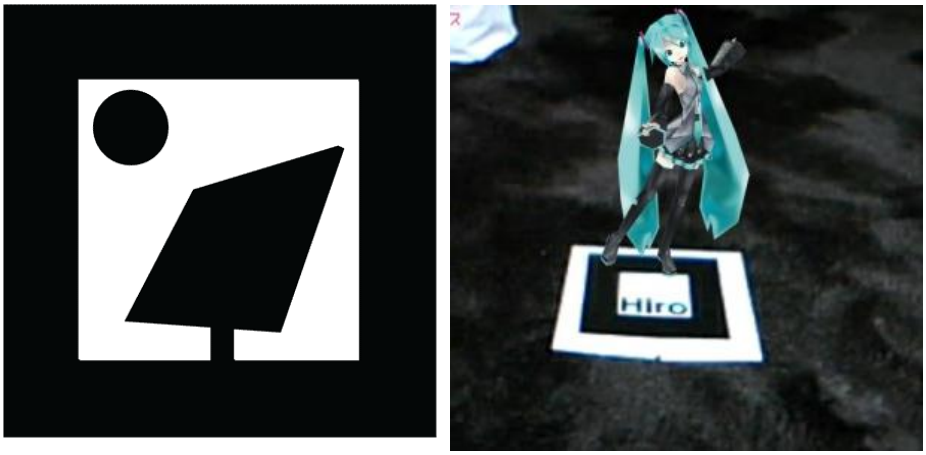



figure 21+22: AR marker example (left) , AR using ARToolkit (right).

השלב הבא יהיה פשוט להשטיח 2D marker במקום הQR-Code ולשלוח את התמונה ל ARToolkit.

# Code compilation

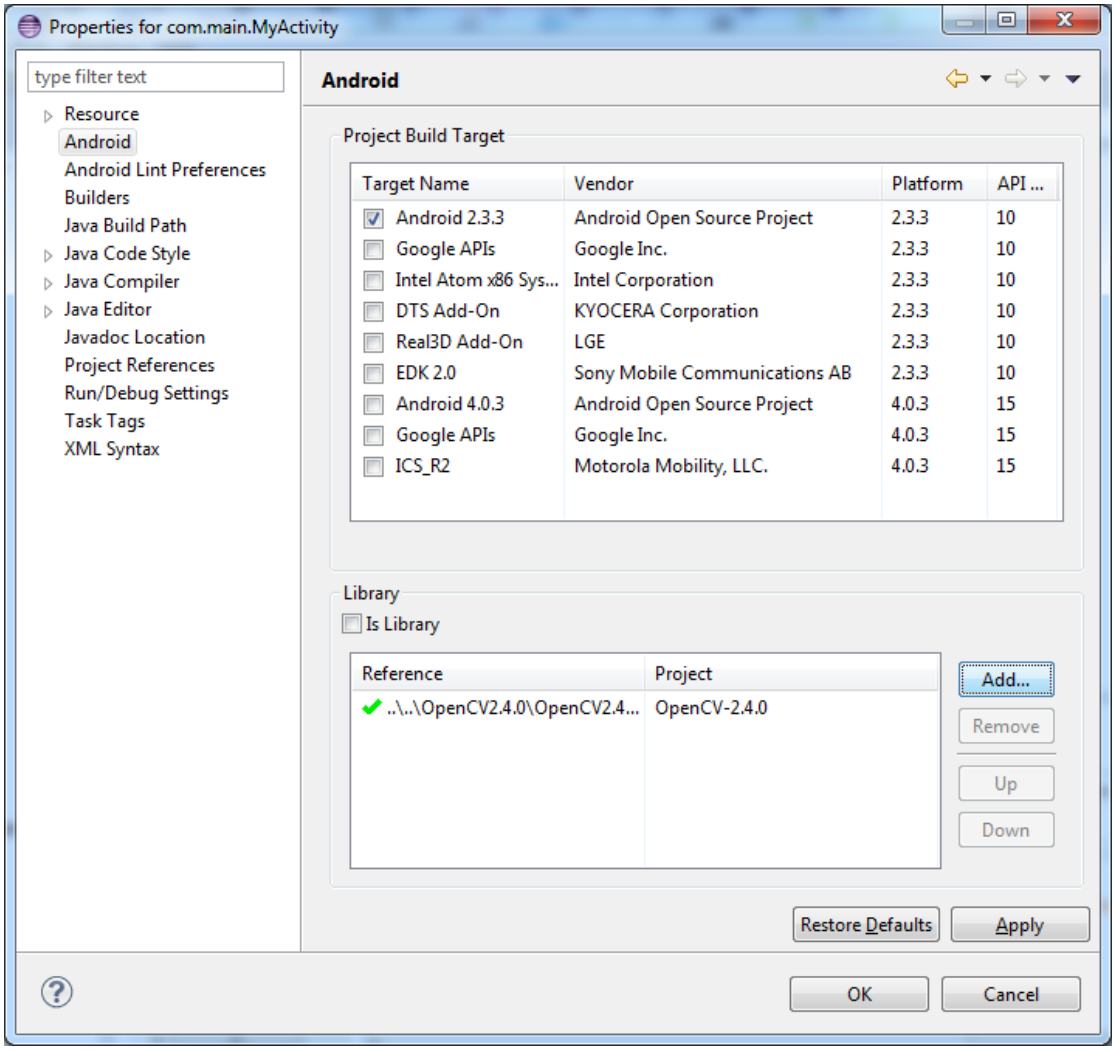
## דרישות מקדימות

- Eclipse with android ADT CDT
- Java JDK 1.6
- Android SDK API Level 10
- תקיית הsource code שלנו: source code

ZXing מקומפל עם הפרוייקט ולכן אין צורך להוריד את הקוד בנפרד.

## הוראות התקנה

- התקייה מכילה 2 ספריות zip, חלץ את הקבצים.
- Eclipse-> import an existing android project-> Import com.example.MyActivity and OpenCV2.4.0
  שים לב שגרסת java ו android מתאימה לאלו הכתובות בדרישות המקדימות.
- במקרה שיש בעיות עם הקישור לפרוייקט openCV2.4.0:
  project properties->Android->add(library panel)->opencv-2.4.0



# קישורים

- האתר שלנו - http://aronqr.wix.com/project
- יוצר QR Code חינמי - http://qrcode.kaywa.com/
- OpenCV- http://opencv.org/
- ZXing- https://code.google.com/p/zxing/

סרטונים שלנו בyoutube:

- סרטון הפרוייקט http://www.youtube.com/watch?v=qnjUayzJDQQ
- סרטון הממחיש את הזיהוי של zxing- http://www.youtube.com/watch?v=w0QCtZIjJFE
- סרטון מוטיבציה לפרוייקט הבא- http://www.youtube.com/watch?v=ZCvqDP-OXt0