סיכום עיבוד תמונה וראייה ממוחשבת:
סוכם ע״י : נעמי צברי

סילבוס:

- Topic 1 – Image Enhancement: histogram, quantization
- Topic 2 – Filtering: smoothing, median filtering, sharpening – Low level detection: Template matching, Edges, Line, Circles
- Topic 3 – Image Pyramids and Blending, Optical Flow
- Topic 4 – Geometry: 2D Transformation, Image Warping, Camera Model
- Topic 5 – Stereo – Homography, Image rectification, Image Stitching (Mosaic/Panorama)
- Topic 6 – Features, Robust Estimation, RANSAC
- Topic 7 – Single view geometry
- Topic 8 – Two views geometry, essential matrix, fundamental matrix, rectification
- Topic 9 – Triangulation, Structure From Motion
- Topic 10 – Deep learning, CNN

מושגים:

- היסטוגרמה - היסטוגרמת תמונה מתארת את פילוג רמות האפור על פני התמונה. מציין את השכיחות היחסית של רמת האפור בתמונה לפי הנוסחה הבאה:

To normalize (probability):

$$p(r_k) = \frac{n_k}{N}$$

כאשר N- גודל התמונה, n_k - מספר הפיקסלים בעלי רמת אפור k.

- Probability- עבור גון כלשהו כמה פעמים הוא מופיע.
- Intensity transform - כשאתה עובד עם תמונות בקנה מידה אפור, לפעמים אתה רוצה לשנות את ערכי העוצמה. לדוגמה, ייתכן שתרצה להפוך את השחור ואת העוצמות הלבנות, או שתרצה להפוך את הגוונים הכהים יותר והגוונים הבהירים יותר. יישום של טרנספורמציות עוצמה מגדיל את הניגודיות בין ערכי עוצמה מסוימים, כך שתוכלו לבחור דברים בתמונה. לדוגמה, שתי התמונות הבאות מציגות תמונה לפני טרנספורמציה בעוצמה ואחריה. במקור, הז׳קט של איש המצלמה נראה שחור, אך עם שינוי עוצמה, ההבדל בין ערכי

העוצמה השחורה, שהיו קרובים מדי לפני כן, הוגדל כך שהכפתורים והכיסים הפכו לנראים.

יש כמה דרכים לבצע זאת:

- **Image negative** : ניקח כל פיקסל ונחסיר ממנו את הערך הגבוהה ביותר- הבהיר, ז״א בגווני אפור- 255 (בבינארי -1).

```
# Read pixels and apply negative transformation
for i in range(0, img.size[0]-1):
    for j in range(0, img.size[1]-1):
        # Get pixel value at (x,y) position of the image
        pixelColorVals = img.getpixel((i,j));

        # Invert color
        redPixel    = 255 - pixelColorVals[0]; # Negate red pixel
        greenPixel  = 255 - pixelColorVals[1]; # Negate green pixel
        bluePixel   = 255 - pixelColorVals[2]; # Negate blue pixel

        # Modify the image with the inverted pixel values
        img.putpixel((i,j),(redPixel, greenPixel, bluePixel));
```

- **Log Transform** - $s = c*\log(1+r)$ להחליף כל פיקסל בלוג שלו, למעשה נעשה זאת כאשר נרצה להעצים את הפיקסלים הנמוכים, על חשבון של אובדן מידע של הפיקסלים הכהים.

- PDF - probability density function פונקצית הסתברות הדחיסות של הצבעים בתמונה, עבור כל גיון אפור נחשב את ההסתברות שלו.

- CDF - The Normalized Histogram פונקציית חלוקה מצטברת לינארית המייצגת בהיסטוגרמה את כמות ההצטברות בכל נקודה.

- סוגי תמונות:

- Intensity (or grayscale) images : תמונה אפורה

Either uint8 in the range [0,255] or doubles in the range [0,1]

- Binary images – תמונה בצבעים או שחור או לבן – 0 ו1.

-RGB images – תמונה עם 3 מטריצות של צבע -אדום, ירוק, כחול. שכל הצבעים יחס נותנים לנו תמונה צבעונית.

- Histogram Equalization – כאשר נרצה לשפר את התמונה, להפוך אותה לחדה וברורה יותר נשנה את ה CDF כך שנזיז תחילה לאמצע הסכום (למשל מ0 עד 255 נשים ב127) ולאחר מכן נמתח את הסכומים כך שמספר גווני האפור יהיה כמה שיותר מתוח על כל הגוונים, מ1 עד 255. איך נעשה זאת?

דוגמה:

Formulation

- The solution (R= g_{old} ,S= g_{new})

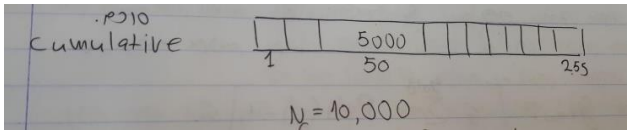
$$g_{new} = \left[G \sum_{g=g_{min}}^{g_{old}} p_{old}(g) - 1 \right]$$

- Notes:
  - g_{new} is unknown, g_{old} s known.
  - $p_{old}(g)$ is the histogram of pixel level g, not the cumulative histogram.

החישוב שנעשה:

cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())

1. נעביר את המשוואה לאמצע "המערך" נרצה שחצי מהגבהים יהיו באמצע.



נעביר מ50 ל 127 את כל ה5000 שנמצאים בגוון 50.

2. נשמור על יחס סדר, אם פיקסל A יותר בהיר מפיקסל B נרצה שהיחס בן הצבעים ישמר במעבר.

ז"א אם כמות הפיקסלים הקטנים מפיקסל R הוא 1700 אז גם כשנעביר את פיקסל R לגוון אחר עדין יהיו 1700 פיקסלים קטנים ממנו.

$$\frac{5000}{10000} \cdot 255 = 127$$

זה החישוב שלפיו נדע איך להזיז כל סכום של אינטנסיטי,

ולשמור על היחס סדר ולדעת לאן להעביר .

שלבי האלגוריתם:

1. מחשבים הסטוגרמה של התמונה
2. מחשבים את ה cumulative ההסתברות המצטברת (ההסתברות של הפיקסל הראשון, ההסתברות של הפיקסל השני ועוד הראשון וכן הלאה..)
3. מנרמלים (לוקחים את קבוצת הפעמים שהאינטנסיטי נמצא בתמונה ומחלקים אותו במספר האינטנסיטי שיש)-אפשר לנרמל לפני.
4. מכפילים את הנרמול בערך המקסימלי של ה greay level (255).
5. מעגלים (ערך תחתון) מעתיקים את ה intensiti הקודם לחדש. ועושים ככה לכולם.
6. יכול להיות מצב שבגלל שאנחנו שומרים על יחס סדר בהעתקה אז לא נשתמש בכל טווח האינטנסיטי. לכן אחרי המעבר נמתח את הכל מ0 עד 255 מההתחלה עד הסוף (וכך נשמור על היחס).

מתי האלגוריתם לא יעבוד? שההנחות לא נכונות, שיש 2 תמונות יחד או שההתפזרות של ה greay level לא אותו דבר.

איך נפתור את זה?

במקרים של תמונות שה intensity צריך להיות בצורה שהיא- קיצוני לצבע מסוים. אז נצטרך לעבור ל high level– נעבד את התמונה לפי אזורים.

עבור כל חלק מהתמונה נעשה כמו שעשינו בתמונה שלמה (נחשב לאן היא צריכה לעבור בתמונה המשופרת) רק שכאן נחשב עבור האזור הקטן אבל ניקח רק את הפיקסל האמצעי **ורק אותו** מעבירים לאינטנסיטי של התמונה המשופרת. וכך נעשה עבור **כל** פיקסל בתמונה.

* בקצוות נכפיל את הערכים עד שנקבל גודל החלון.

איפה זה יכשל? במעברים, אז איזה חלון ניקח? צריך שלא יהיה קטן מידי כי אז הוא לא ישנה כלום אבל גם לא גדול מידי-שיטשטש יותר מידי.

הסבר נוסף- https://www.youtube.com/watch?v=PD5d7EKYLcA

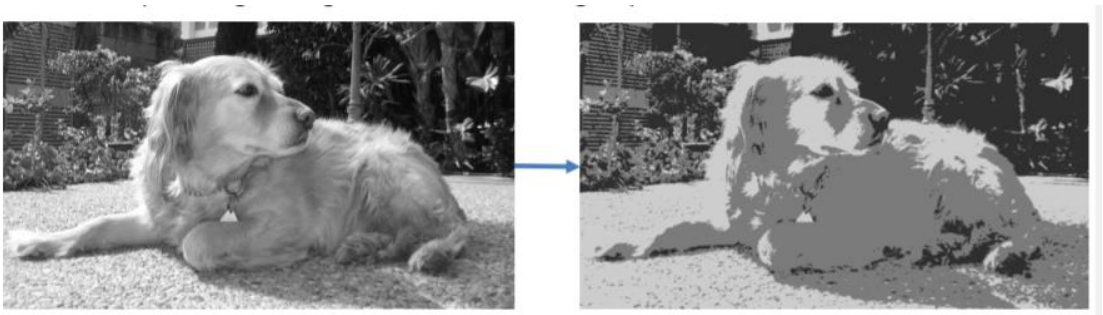
Pixel Intensity	1	2	3	4	5	6	7	8	9	10
No. of pixels	1	3	3	2	2	1	3	1	0	0
Probability	.0625	.1875	.1875	.125	.125	.0625	.1875	.0625	0	0
Cumulative probability	.0625	.25	.4375	.5625	.6875	.75	.9375	1	1	1
C.P * 20	1.25	5	8.75	11.25	13.75	15	18.75	20	20	20
Floor Rounding	1	5	8	11	13	15	18	20	20	20

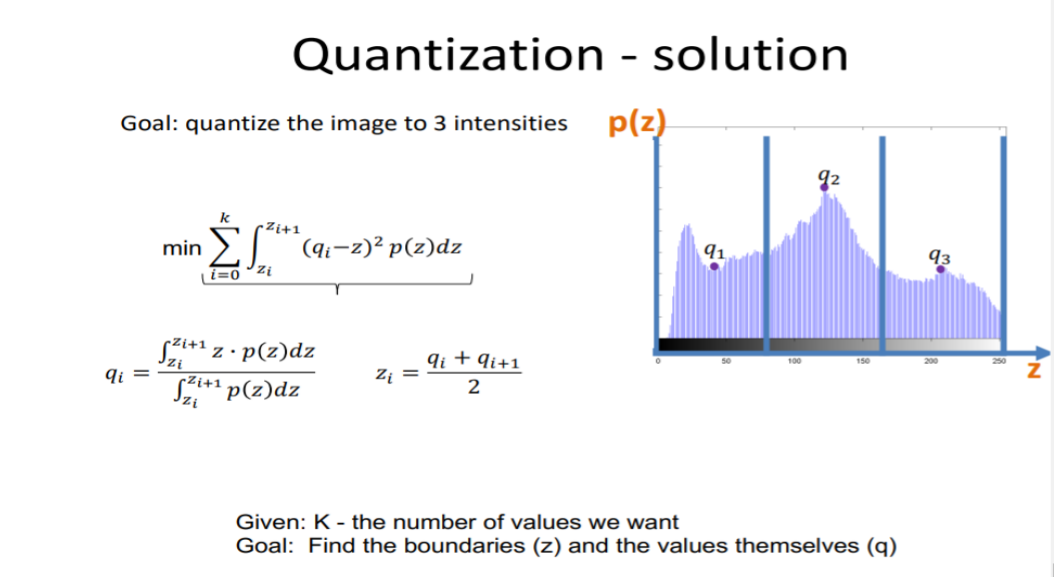
- Quantization – להפוך תמונה מהרבה גווני אפור להרבה פחות. טווח גווני האפור מצומצם יותר, נצטרך להעביר כל גוון וגוון למספר מצומצם של גוונים, שימוש- דחיסה.

הגדרה פורמלית: המרת אות אחד למשנהו תוך תחימת ערכי האות המומר לתחום (קטן בד״כ) מהתחום המקורי.

לאחר שבחרנו את מס׳ הצבעים, השאלה היא: איך מנפים כל גוון לגוונים המצומצמים?

נבחר סיגמנטים של חלוקה, נבחר גודל של סיגמנט (נבחר כמות סיגמנטים בהתאמה לכמות הערכים - סיגמנטים -מחיצות) ומתוך האינטרוול (בין כל 2 סיגמנטים) נבחר את הערך שנרצה. למשל עבור 3 ערכים נשים 4 סיגמנטים.





qi – הערך שאתה רוצה שיהיה מינימלי.

zi – החלוקה.

למעשה נחלק למחיצות, בתוך המחיצות נמצא את הגוון המרכזי- לפי תנאים שנחליט עליהם. וכך נסווג- כל גוון יסווג לערך(אינטרוול) שנקבע הכי קרוב אליו.

K means- נעשה את האיטרציות האלו כמה פעמים- נבחר קבוצות רנדומליות(לפי מספר הגוונים שרצינו לשנות) וניצור נקו לפי הממוצע שלהם. לאחר מכן עשה איטרציה נוספת שבה כל פיקסל בודק מה המרכז (הנקו ממוצע) שקרוב אליו ולאחר מכן נחשב שוב את המרכזים. נעשה את זה לרוב 7 עמדים או עד מה שהגדרנו- עד שניראה שהשינוי לא משנה עוד .

לא יעבוד תמיד.

שיטה נוספת- (סיגמנטים)

נחלק לחוצצים, בכל בין לבין חוצץ נמצא ממוצע משוכללת ואז נזיז את המחיצה לאמצע בין כל ממוצע משוכלל, ואז שוב נחשב את ההמוצע המשוכלל נעשה זאת גם כאן עד שאין שינוי. יותר מהר.

```
def kmeans(pnts: np.ndarray, k: int, iter_num: int = 7) ->
(np.ndarray, List[np.ndarray]):
    """
    Calculates K-Means
    :param pnts: The data, a [nXm] array, **n** samples of **m**
```

```
dimentions
    :param k: Numbers of means
    :param iter_num: Number of K-Means iterations
    :return: [k,m] centers,
             List of the history of assignments, each assignment is a
[n,1]
             array with assignment of each data point to center,
    """
    if len(pnts.shape) > 1:
        n, m = pnts.shape
    else:
        n = pnts.shape[0]
        m = 1
    pnts = pnts.reshape((n, m))
    assign_array = np.random.randint(0, k, n)
    centers = np.zeros((k, m))

    assign_history = []
    for it in range(iter_num):
        for i, _ in enumerate(centers):
            if sum(assign_array == i) < 1:
                continue
            centers[i] = pnts[assign_array == i, :].mean(axis=0)

        for i, p in enumerate(pnts):
            center_dist = np.sqrt((centers - p) ** 2).sum(axis=1)
            assign_array[i] = np.argmin(center_dist)
        assign_history.append(assign_array.copy())

    return centers, assign_history


def quantizeImageKmeans(imOrig: np.ndarray, nQuant: int, nIter: int)
-> (List[np.ndarray], List[float]):
    """
    Quantized an image in to **nQuant** colors
    :param imOrig: The original image (RGB or Gray scale)
    :param nQuant: Number of colors to quantize the image to
    :param nIter: Number of iterations for the KMeans
    :return: (List[qImage_i],List[error_i])
    """

    gray = imOrig / imOrig.max()
    if len(gray.shape) > 2:
        data = gray.reshape((-1, 3))
        height, width, _ = gray.shape
    else:
        data = gray.reshape(-1)

    cs, assignment = kmeans(data, nQuant, nIter)

    assign_list = []
    error_list = []
    for assign_i in assignment:
        qunt_img = np.zeros(data.shape)
        for i, c in enumerate(cs):
            qunt_img[assign_i == i] = c
        qunt_img = qunt_img.reshape(gray.shape)

        error = np.sqrt(((qunt_img - gray) ** 2).sum())
        assign_list.append(qunt_img)
```

```
        error_list.append(error)

    return assign_list, error_list
```

טרנפורמציה חסרת זיכרון הממירה אות אחת לאות אחרת ע״י העתקת הערכים המותרים לאות לתחום חדש, בדרך כלל קטן יותר מהתחום המקורי.
ייצוג האות השני כרוך בשגיאה, למשל, אתהקוונטיזציה האחידה (בה המרווח בין הרמות אחיד) ניתן לתאר ע״י פונקציית המרה, ואת שגיאת הקוונטיזציה ניתן לחשב עפ״י הרמות השונות של אות הכניסה.

קוונטיזציה יוניפורמית (uniform quantization):

המרווח בין רמות הקוונטיזציה אחיד (שגיאות הקוונטיזציה המקסימלית הינה מינימלית).
(חלוקה שווה בסיגמנטים, למשל אם יש לי להמיר ל3 צבעים נחלק ל4 בצורה שווה).

קוונטיזציה לא יוניפורמית (non-uniform quantization):

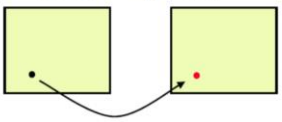
המרווח בין רמות הקוונטיזציה אינו אחיד (עבור אינטרוולים שווים של אות כניסה). כללית, אם אות הכניסה הינה בעל פילוג p(z), אזי קביעת רמות הקוונטיזציה תעשה עפ״י קריטריון <u>השגיאה הריבועית המינימלית</u>:

$$\epsilon^2 = \sum_{i=0}^{K-1} \int_{z_i}^{z_{i+1}} (q_i - z)^2 p(z)\,dz \to \min$$

- Point operation – (פעולת נקודה) היא שינוי לערך פיקסל המבוסס על ערך פיקסל זה ואינו תלוי במיקום או בערכים שכנים (ללא הקשר (פחות זיכרון)).
ניתן ליישם על ידי: יישום אריתמטי של קבוע, יישום לוגי של אופרטור בוליאני, שינוי היסטוגרמה.

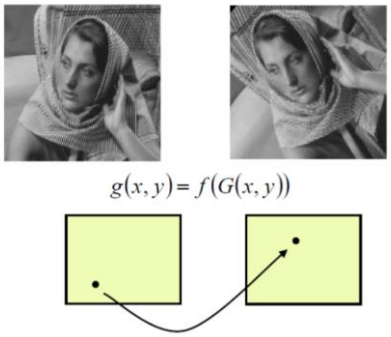
Example: $g(x,y) = \alpha \cdot f(x,y) + \beta$

$g(x,y) = M(f(x,y))$



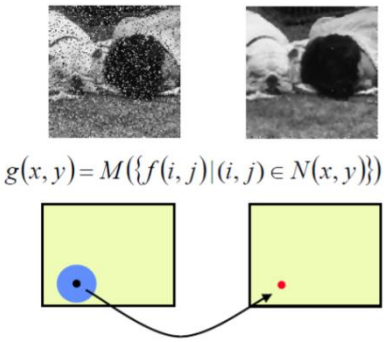
- Geometric operations – הפעולה תלויה בקואורדינטות של הפיקסלים. הקשר חופשי. לא תלוי בערך הפיקסלים. לרוב ישמש להזזה, הפיכה וכדו של תמונה.

Example: $g(x,y)=f(x+a,y+b)$



- Spatial operations - הפעולה תלויה בערך הפיקסלים ובקואורדינטות. תלוי הקשר - תלוי גם בפיקסלים הסמוכים.

Example: $g(x,y)=\sum_{i,j\in N(x,y)} f(i,j)/n$



- Filtering – להריץ את אותה פונקציה עבור כל אופציה של גודל חלון בתמונה.

105	114	112	110	109	109	111	114	116	124	124
114	116	114	111	109	109	110	112	114	127	125
124	119	116	112	109	108	109	111	112	126	123
127	120	118	113	110	108	108	109	110	123	120
125	121	118	114	110	108	107	108	110	120	117
129	115	107	99	98	102	105	103	98	117	118
125	127	119	110	108	112	114	110	105	111	114
119	132	122	113	110	112	113	109	103	102	107
113	125	116	106	102	104	105	100	94	96	102

Sliding window- עבור כל פיקסל בתמונה:

ניקח גודל חלון kxk.

חישוב הפונקציה על כל הפיקסלים בחלון.

שינוי הערך של הפיקסל האמצעי בחלון לתוצאה של החישוב לפי הפונקציה.

לפונקציה קוראים- filter . (הפעולה הזו נקראת כך כיון שנעשה זאת עבור כל פיקסל בתמונה).

נעשה פעולה זאת עבור:

1. Template Matching- התאמת וזיהוי תבניות.
2. Clean noise – ניקוי רעשים מתמונה.
3. Detect Edges- זיהוי אובייקטים (edges) בתמונה ע״י זיהוי קצוות האובייקט.

- Filter Kernel -

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- Min Filter- הקרנל "מוצא" את המינימום בחלון ושם אותו במרכז
- Max Filter– הקרנל "מוצא" את המקסימום בחלון ושם אותו במרכז.

- רעש הערות כלליות-

במציאות הייצוג של סוגי הרעש השונים הינו דיסקרטי (ולא רציף), כלומר היסטוגרמות בניגוד ל pdf מניחים כי הרעש אינו קורלטיבי, כלומר שינויי רמות האפור (כתוצאה מהרעש) אינם תלויים מרחבית כשעוברים מפיקסל לפיקסל, ופילוג הרעש בכל פיקסל אינו תלוי במיקום הפיקסל .

הוספת רעש (באופן סינטטי) לתמונה נתונה

$$g(x,y) = f(x,y) + n(x,y)$$

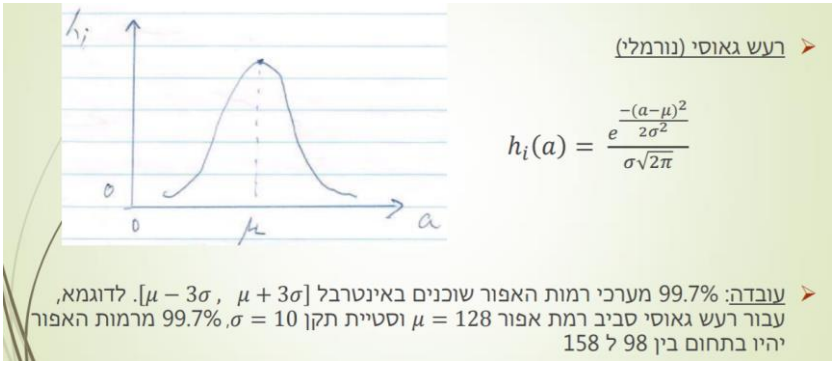
או הכפלת התמונה ברעש $g(x,y) = f(x,y) \cdot n(x,y)$

סילוק רעש (רעש מוכפל) multiplicative קשה יותר ומצריך עיבוד לא-לינארי.

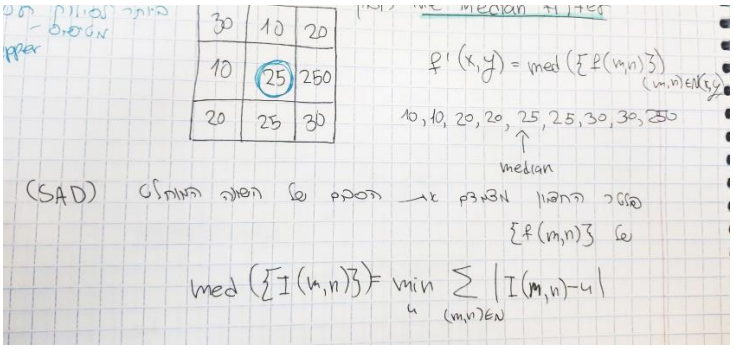
- Salt & Pepper Noise – (רעש מלח פלפל) הוא רעש על תמונה של כל מיני נקודות רנדומליות של לבן ושחר ז"א ערכים קיצונים בפיקסלים מסוימים. נוצר לפעמים מחומר התמונה, הצטברות אבק במערכת האופטית וכדו. נוכל לייצר אותו באופן סינטטי ע"י המשוואה :

$$f_n(x,y) = \begin{cases} f(x,y) & \text{with probability } p \\ 255 & \text{with probability } (1-p)/2 \\ 0 & \text{with probability } (1-p)/2 \end{cases}$$

- רעש גאוסי -



- The Median filter – (חציון- בעל הביצועים הטובים ביותר לסילוק רעש מטיפוס - s&p)

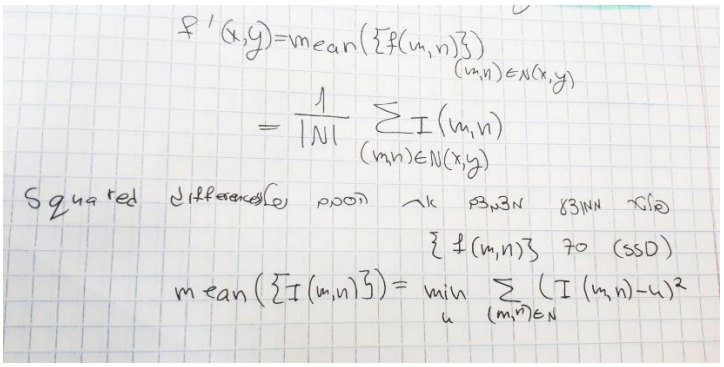


לוקח את כל המספרים בחלון, שם אותם "בשורה" ולוקח את החיציון.

Median filter algo

```python
def medianFilter(img,ksize):
    hsize = ksize//2
    retImage = np.zeros(img.shape)
    h,w = img.shape
    for i in range(h):
        for j in range(w):
            box = img[i-hsize:i+1+hsize,j-hsize:j+1+hsize]
            retImage[i,j] = np.median(box)
    return retImage
```

- The Average filter – (ממוצע)



נשתמש בפונקצית ssd -sum of squared differences.

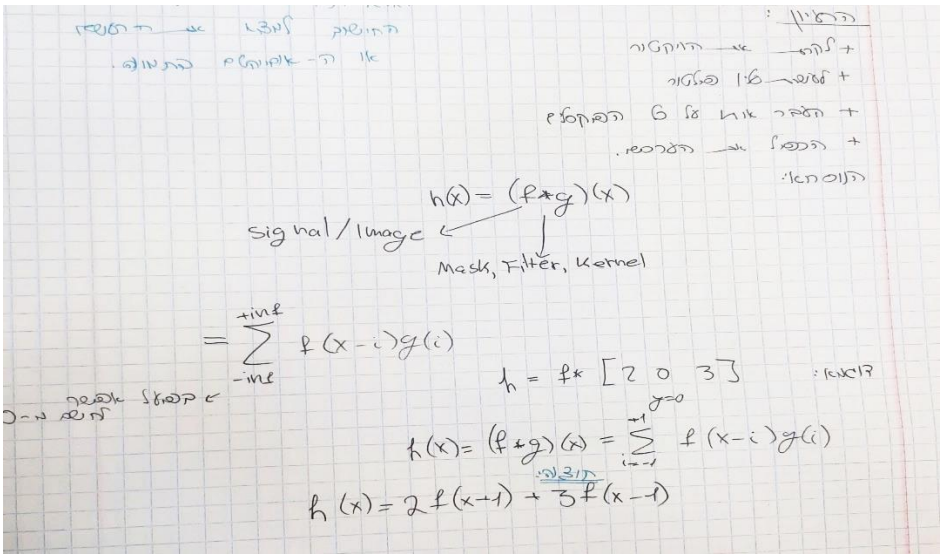
ניקח את כל המספרים בשורה, ניקח את הממוצע וזה מה שנשים באמצע.

- **Filter, kernel and mask are the same!**

- Convolution – במתמטיקה (בפרט, ניתוח פונקציונלי) קונבולוציה היא פעולה מתמטית בשני פונקציות (g ו f) לייצור פונק' שלישית המבטאת את אופן שינוי הצורה של האחד על ידי האחר. (המונח קונבולוציה מתייחס הן לתפקוד התוצאה והן לתהליך המחשוב שלה. זה מוגדר כאינטגרל של המוצר משני הפונקציות לאחר שהאחת הופכת ומועברת). פעולה לינארית על אות או על שינוי האות (כמובן שכאן נשתמש בזה לשינוי תמונות) למעשה כדי להשתמש ב filter נריץ אותם כמטריצה על התמונה ע"י הפעולה Convolution ובכך נקבל את אפקט החלון שרצינו. הפעולה תתבצע כך :
עבור כל פיקסל, ניקח חלון בגודל KxK, נעשה הכפלת מטריצות בין החלון עם הקרנל- את התוצאה

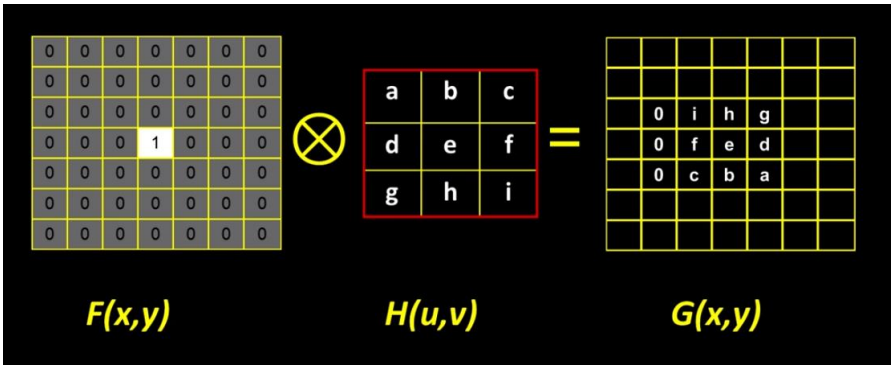
של האמצעי ,ונשנה את הערך של הקורדינטה של הפיקסל.

עבור D1 : (חד ממדית)



עבור D2 : (דו ממדית)

מדוע צריך להפוך את הקרנל :



https://www.youtube.com/watch?v=KTB_OFoAQcc

```
def conv2D(inImage: np.ndarray, kernel2: np.ndarray) -> np.ndarray:
    """
    Convolve a 1-D array with a given kernel
    :param inImage: 2D image
    :param kernel2: A kernel
    :return: The convolved image
    """

    inv_k = kernel2[::, ::]
    kernel_shape = np.array([x for x in kernel2.shape])
    img_shape = np.array([x for x in inImage.shape])
    out_len = np.max([kernel_shape, img_shape], axis=0)
    midKernel = kernel_shape // 2
    paddedSignal = np.pad(inImage.astype(np.float32),
                          ((kernel_shape[0], kernel_shape[0]),
                           (kernel_shape[1], kernel_shape[1]))
```

```
                                , 'edge')

    outSignal = np.ones(out_len)
    for i in range(out_len[0]):
        for j in range(out_len[1]):
            st_x = j + midKernel[1] + 1
            end_x = st_x + kernel_shape[1]
            st_y = i + midKernel[0] + 1
            end_y = st_y + kernel_shape[0]

            outSignal[i, j] = (paddedSignal[st_y:end_y, st_x:end_x] *
inv_k).sum()

    return outSignal
```

- Correlation – בדיקה של מדד הדמיון בין 2 דברים. (התהליך דומה לconvolution ואם ה kernel סימטרי התוצאות של שניהם יהיו זהות).
- Smoothing by Spatial Filtering – טשטוש ע"י kernel וקונבולוציה.

- Template Matching - בהינתן תמונה / אותת, אנו רוצים למצוא את המיקום של תבנית נתונה.

כדי לבצע זאת נצטרך להבחין במספר דברים:

נייצג את התבנית על ידי מטריצה. האתגר העיקרי הוא:

1. התמונה והתבנית עשויים להיות מעט שונים אחד מהשני.
2. רעש.
3. מודל שמיוצג ע"י כפל וחיבור.

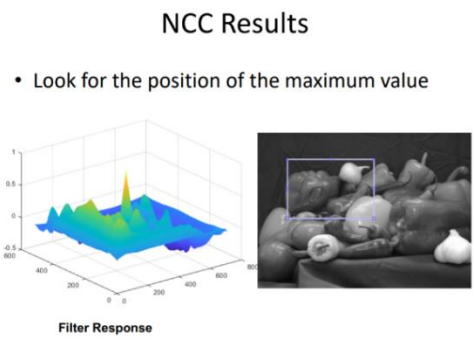
לכן נשתמש ב - Normalized Cross Correlation.

הסבר -

https://en.wikipedia.org/wiki/Template_matching#targetText=Template%20matching%20is%20a%20technique,to%20detect%20edges%20in%20images.

האלגוריתם:

1. צריך – מטריצת תבנית בגודל KxK , מטריצת תמונה בגודל NxN .
2. עבור כל KxK חלון אפשרי בתמונה נעשה NCC בין התבנית לחלון (חלק מהתמונה בגודל התבנית).
3. תסמן את המצב עם הערך הכי גבוה כמצב של התבנית .



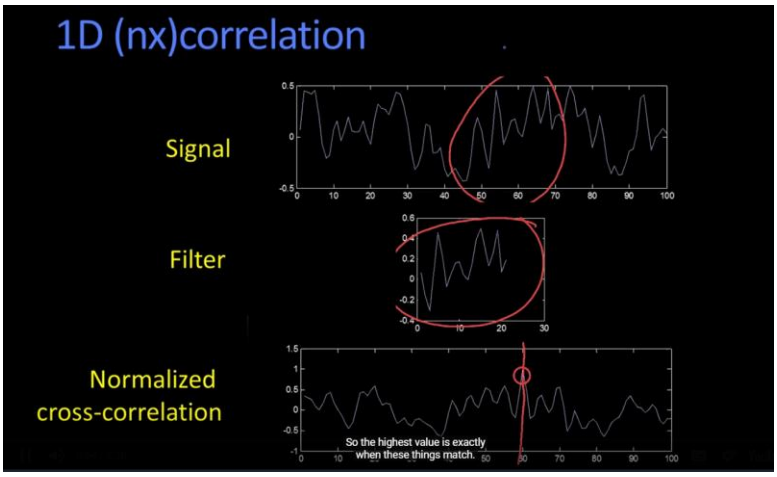
- Normalized Cross Correlation – (NCC) השוואה בין שני מטריצות.

למעשה NCC היא הדרך לעשות Template Matching, נריץ קונבולוציה על התמונה עם פילטר שהוא למעשה התבנית שלנו- מה שאנחנו רוצים לחפש בתמונה. כאשר נגיע בתמונה למקום שבו נימצא התבנית- נקבל סכום גבוהה יחסית (נוכל לחשב אותו לפני) ולפי זה נידע שזה אכן התבנית.

$$\cos(\theta) \stackrel{\text{def}}{=} \frac{\vec{\alpha} \cdot \vec{\beta}}{|\vec{\alpha}| \cdot |\vec{\beta}|}$$

Zero normalized cross correlation

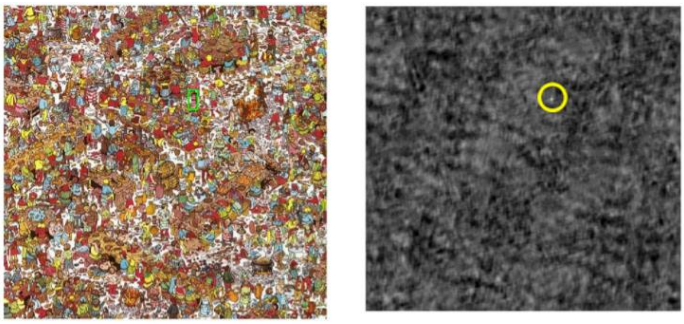
- $\vec{\alpha} = \vec{\alpha}$ -mean($\vec{\alpha}$)
- Same for beta



הסבר - https://anomaly.io/understand-auto-cross-correlation-normalized-shift/

זוהי הדרך הנפוצה ביותר להשוואה בין תמונה לתבנית, יעבוד אפילו אם התבנית לא זהה.

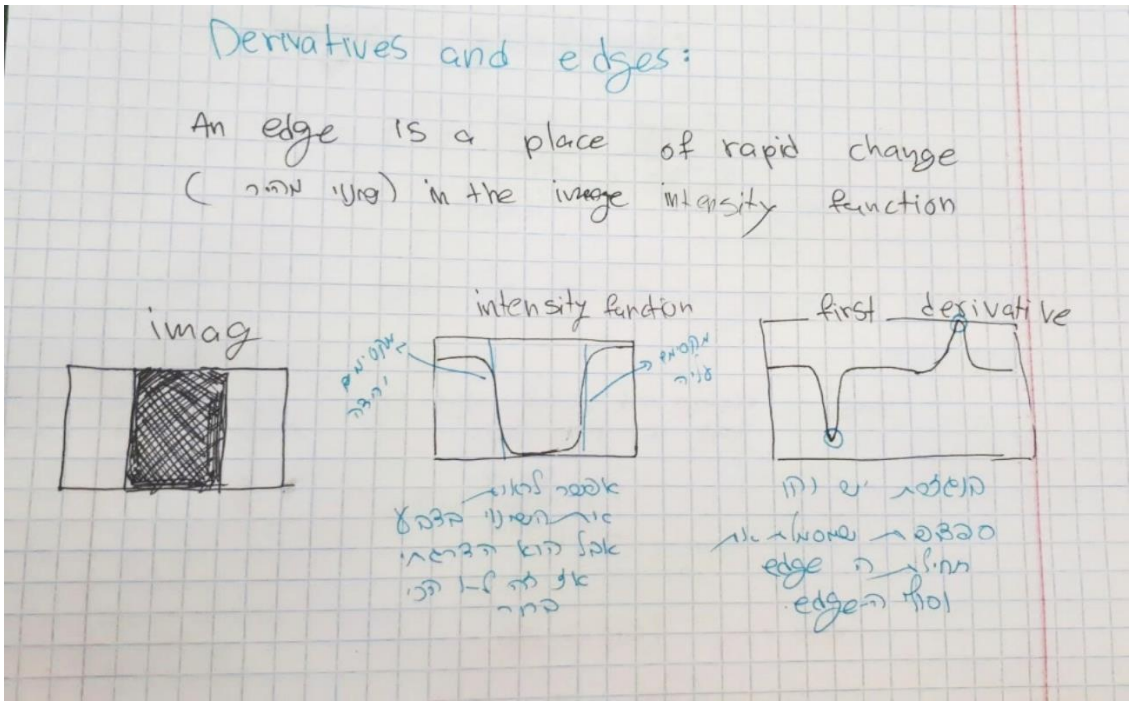
- Correlation Map – עבור תבנית נתונה יתן לנו ערך לבן-הבהיר ביותר עבור ההתאמה המרבית. (אפילו דברים מאוד קטנים נמצא ע״י התבנית).



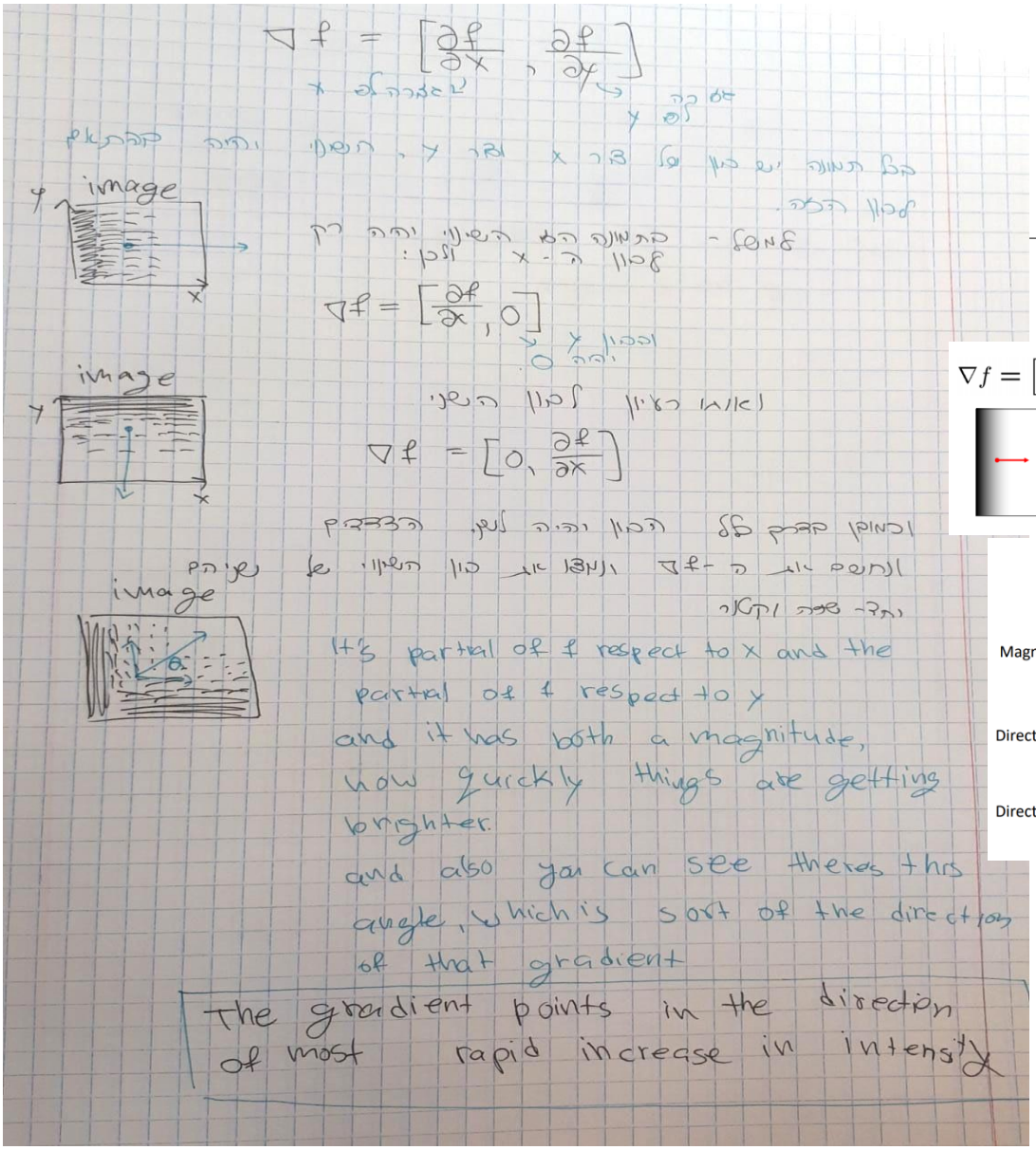
Filter Response

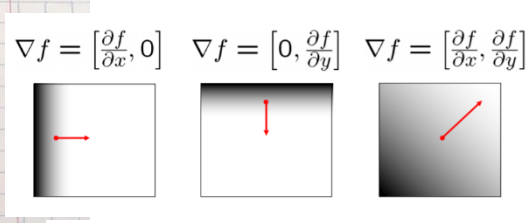
:Image Derivatives: Edges and Sharpening

- Edges -



- Image gradient -



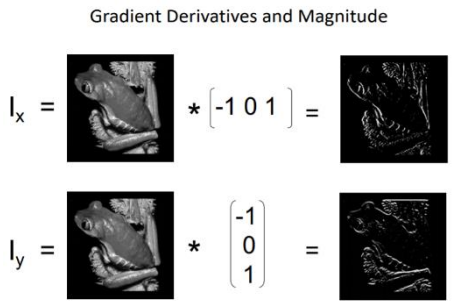


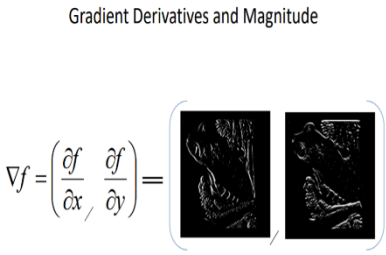
The Gradient - Properties

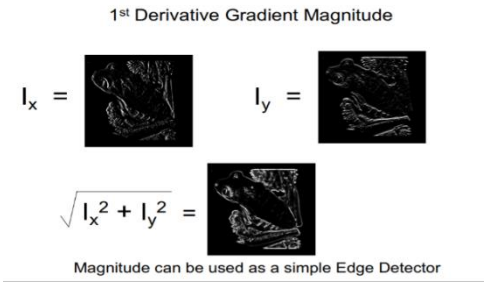
Magnitude - $|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

Direction (orientation angle) - $\alpha = \tan^{-1}\left(\left(\frac{\partial f}{\partial y}\right) \Big/ \left(\frac{\partial f}{\partial x}\right)\right)$

Directional derivative - $\cos(\alpha)\frac{\partial f}{\partial x} + \sin(\alpha)\frac{\partial f}{\partial y}$







1. וקטור הגראדינט מצביע על הכיוון בו קצב השינוי של f(x,y) הוא מירבי.
2. גודל הגראדינט $\sqrt{G_x^2 + G_y^2}$ שווה לקצב השינוי המירבי של f(x,y) ליחידת מרחק בכיוון G .
3. כיוון הגראדינט מוגדר ע"י

$$\alpha(x,y) = \arctan\left(\frac{G_y}{G_x}\right) = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

- Edge Detection - זיהוי ה edge בתמונה. בשביל למצוא את ה edge נצטרך למצוא איפה הוא מתחיל ואיפה הוא נגמר, באיזה מיקום. בשביל זה נשתמש ב gradient שיבליט לנו את השינויים בתמונה.
לכל edge ניתן לייחס עוצמה (יחסית לגודל השינוי)וכיוון (אוריינטציה), באשר הכיוון הוא בניצב (normal)לedge.
צעדים אופייניים ל- Edge Detection:
1. סינון ראשוני (tradeoff)בין סינון רעש לחדות ה-edges.
2. שיפור (enhancement) למציאת ה- edges.
3. גילוי ה -edges.
4. מיקום (location) מחייב עבודה ברמת דיוק של תת-פיקסל (subpixel resolution).
או יותר מפורט:

• Main steps in edge detection using masks

(1) Smooth the input image ($\hat{f}(x,y) = f(x,y) * G(x,y)$)

(2) $\hat{f}_x = \hat{f}(x,y) * M_x(x,y)$

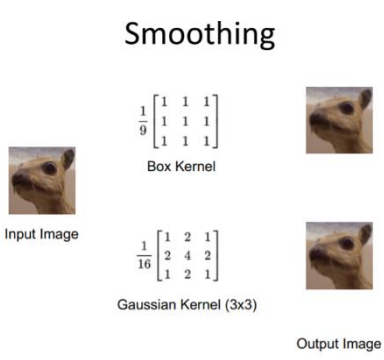
(3) $\hat{f}_y = \hat{f}(x,y) * M_y(x,y)$

(4) $magn(x,y) = |\hat{f}_x| + |\hat{f}_y|$

(5) $dir(x,y) = \tan^{-1}(\hat{f}_y/\hat{f}_x)$

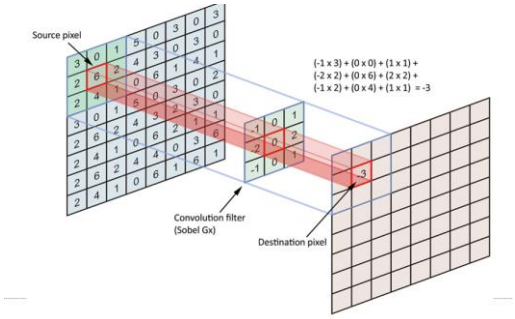
(6) If $magn(x,y) > T$, then possible edge point

נקודת שפה(point edge)- פיקסל במקום בו קיים שינוי עוצמה לוקאלי משמעותי בתמונה.
בשביל לעשות זאת נשתמש בkernel כמו-Sobel, Zero Crossing Simple , Zero Crossing LOG, Canny.

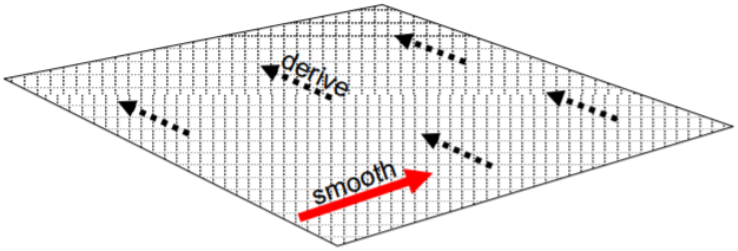


- Blurring/Smoothing – טשטש. תמיד טשטש את התמונה לפני ביצוע אופרטורים.
הטשטוש בדרך כלל הוא box filter או Gaussian filter.

בעיה: ברצוננו להפחית את השפעות הרעש על החלקה הנגזרת אך הסימטרית smoothing עשויה למחוק את קצוות הedge (התגובה של הנגזרת). הפיתרון הוא:



- Sobel – להחליק (נטשטש) את התמונה לכיוון אחד, ולחשב את הנגזרת בכיוון האורתוגונלי.(כך אנחנו מורידים רעש כיוני ולא הורסים את הedge). בעזרת הנגזרת השנייה אני מוצא את המיקום של ה edge כי רוב הפעמים הedge לא יהיה בינארי, ההבדל בינו לבין השאר הוא הדרגתי ולכן לא מיידי, הנגזרת השנייה נותנת לנו את האפשרות הזו- המספרים ה "פיקים" הם ה-edge.



****לבדוק תקינות של הקוד****

חשוב לשים לב – לחלק את הכל ב 1/8 (אפשר לראות בקוד בשורה האחרונה) אפשר לראות את המטריצה של sobel כאן שהיא למעשה עושה את כל הפעולה של הנגזרת והטישטוש, נעשה איתה קונוולוציה. למעשה יש לי החלקה לצד x שבה הגזירה תהיה לצד של x והטשטוש יהיה לכיון השני וכן לy.

Pseudocode implementation [edit]

```
function sobel(A : as two dimensional image array)
    Gx=[-1 0 1; -2 0 2; -1 0 1]
    Gy=[-1 -2 -1; 0 0 0; 1 2 1]

    rows = size(A,1)
    columns = size(A,2)
    mag=zeros(A)

    for i=1:rows-2
        for j=1:columns-2
            S1=sum(sum(Gx.*A(i:i+2,j:j+2)))
            S2=sum(sum(Gy.*A(i:i+2,j:j+2)))

            mag(i+1,j+1)=sqrt(S1.^2+S2.^2)
        end for
    end for

    threshold = 70 %varies for application [0 255]
    output_image = max(mag,threshold)
    output_image(output_image==round(threshold))=0;
    return output_image
end function
```

- Canny- שלבי האלגוריתם:

1. Smooth נפלטר את התמונה עם נגזרת של gaussian (לפי X ו Y).
2. נמצא magnitude (גודל-יחיד) ו- orientation (הכיוון- הזיות) של הגרדיאנט.

$$|G(x,y)| = \sqrt{Ix^2 + Iy^2} \quad , \quad \alpha = \tan^{-1}\left({Iy}/{Ix} \right)$$

3. ניקח את החלקים העבים של ה edge ונצר אותם.

איך נעשה את זה?

עבור כל פיקסל (x,y) מוצאים את כל הגרדיאנטים אם אתה לא פיק (local peak) ז״א החלק שהוא משתנה מאוד מהר אז נשנה אותו ל-0.

3. Threshold M:

$$M_T(m,n) = \begin{cases} M(m,n) & \text{if } M(m,n) > T \\ 0 & \text{otherwise} \end{cases}$$

where T is so chosen that all edge elements are kept while most of the noise is suppressed.

4. Hysteresis- 1. ניקח T1>T2 ז״א threshold.

2. כל פיקסל |G(x,y)|>T1 הוא חשוד בלהיות פיקסל edge.

אם הוא שווה ומחובר לt2 אז גם אותו נבחר להיות חלק מ-edge .

****לבדוק תקינות של הקוד****

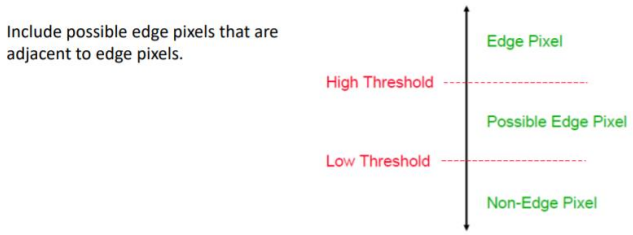
```
def edgeDetectionCanny(I:np.ndarray)->(np.ndarray,np.ndarray):
    I = cv2.GaussianBlur(I, (7, 7), 0)
    mag, img_x, img_y = convDerivative(I)
    D = orientation(I, img_x, img_y)
    res, weak, strong = threshold(non_max_suppression(mag, D))
    return hysteresis(mag, weak, strong)

def threshold(img, lowThresholdRatio=0.05, highThresholdRatio=0.09):
    highThreshold = img.max() * highThresholdRatio
    lowThreshold = highThreshold * lowThresholdRatio

    M, N = img.shape
    res = np.zeros((M, N), dtype=np.int32)

    weak = np.int32(25)
    strong = np.int32(255)

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >= lowThreshold))

    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak

    return (res, weak, strong)
```

```
def hysteresis(img, weak, strong=255):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1,
j] == strong) or (img[i+1, j+1] == strong)
                        or (img[i, j-1] == strong) or (img[i,
j+1] == strong)
                        or (img[i-1, j-1] == strong) or
(img[i-1, j] == strong) or (img[i-1, j+1] == strong)):
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

- Hysteresis – יחבר לנו edge שלא נראים מחוברים בהכרח (משתמשים ברעיון זה גם ב canny).

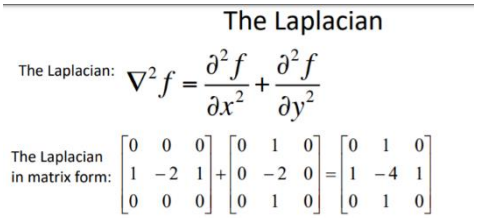
האלגוריתם הוא :

- נשתמש ב threshold גבוה ונמוך.
- כל edge מעל הגובה- הוא edge אמיתי (נשמור אותו).
- כל edge מתחת לנמוך -הוא לא edge אמיתי (נמחוק אותו).
- כל edge שהוא בין הנמוך לגבוה , נשמור אותו רק אם הוא מחובר לedge חזק (ברור).

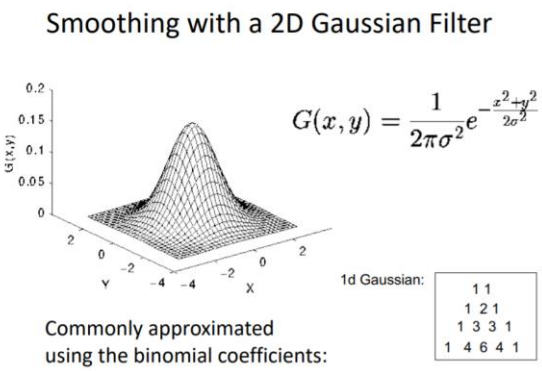


- **** – Zero Crossing Simple

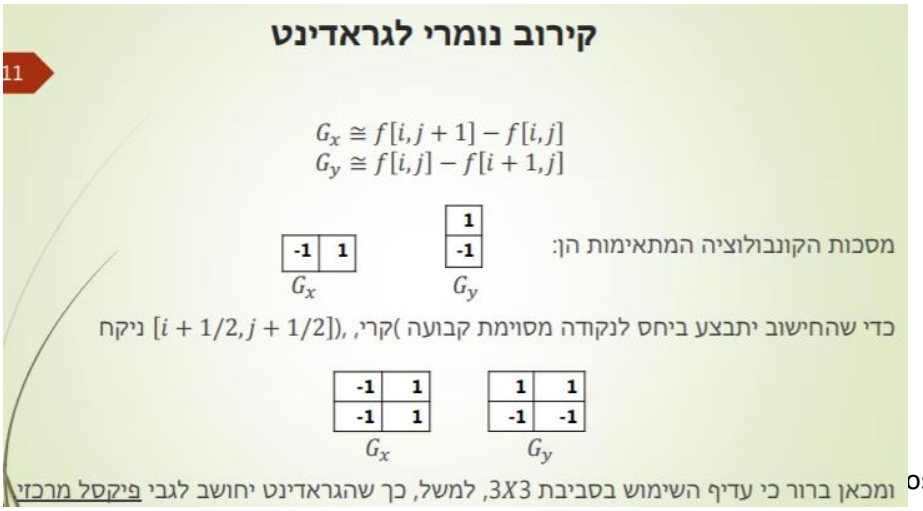
- Zero Crossing LOG -



- Gaussian filter – זה עוזר לי כאשר אני רוצה לתת משקל לedge, לכן נעשה נגזרת "תלת מימד" הבעיה שאם יש רעש זה ישפיעה. לכן ברגע שרוצים לטשטש תמונה- נעשה גואסיאן- תמיד עובד.



- -Derivative – Numeric



- Laplacian gaussian – ossing

- Laplacian sharping -גזירה שנייה.

- Image Sharpening – ?????

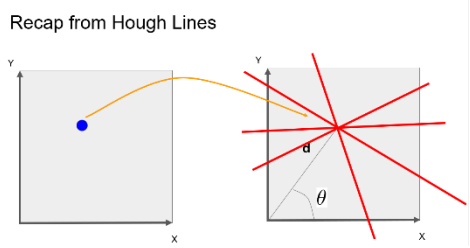
- Sharpening via Laplacian Subtraction - ??????????????

- איך מוצאים קו? (Line Detection) - לרצף נקודות שיקוים עבורם y=ax+b -מודל זה נקרא: מודל פרמטרי.

היינו רוצים שעבור נקודה (בלי לדעת כלום מראש) נדע לאיזה קווים הנקודה יכולה להשתתף בצורה יעילה, אז אם יש כמה נקודות שקשורות לאותו קו – אז יש לנו קו בתמונה.

- Lines – Hough space – עבור כל נקודה יש ישר בHough space שבו יש את כל ה -m,b האפשרים בשבילו. ועבור 2 נקו' נמצא ב- Hough space את נקו/ החיתוך בין 2 המשוואות ושם זה הקו של 2 הנקודות. אם נעשה את זה להרבה נקודות – החיתוך ביניהם- נימצא נקו שה m,b שלה זה המשוואה שלנו. אבל בעולם לא אידיאלי – הנקודות זזות ולא מדויקות לכן נחלק את הHough space ל"בינים" וניקח את ה"בין" עם הכי הרבה קווים שנמצאים בו.

מה הבעיות שיש:



סוכם ע״י נעמי צברי

-הטווח אינסופי

- מה קורה שהקו מאונך? אין לנו m...

הסבר-

https://classroom.udacity.com/courses/ud810/lessons/2870588566/concepts/34824487840923

כאשר הישרים שאנחנו מחפים לא בהכרח ישרים מאוד ונרצה למצוא את ההתחלה והסוף שלהם:

https://he.wikipedia.org/wiki/%D7%94%D7%AA%D7%9E%D7%A8%D7%AA_%D7%94%D7%90%D7%A3

- Hough Circles –

```
4 Hough Circles
def
houghCircle(I:np.ndarray,minRadius:float=18,maxRadius:float=20
)->np.ndarray:
    # Find circles
    rmin = 18
    rmax = 20
    steps = 100
    threshold = 0.4

    points = []
    for r in range(rmin, rmax + 1):
        for t in range(steps):
            points.append((r, int(r * math.cos(2 * pi * t /
steps)), int(r * math.sin(2 * pi * t / steps))))

    acc = defaultdict(int)
    for x, y in edgeDetectionCanny(I):
        for r, dx, dy in points:
            a = x - dx
            b = y - dy
            acc[(a, b, r)] += 1

    circles = []
    for k, v in sorted(acc.items(), key=lambda i: -i[1]):
        x, y, r = k
        if v / steps >= threshold and all((x - xc) * 2 + (y -
yc) * 2 > rc ** 2 for xc, yc, rc in circles):
            print(v / steps, x, y, r)
            circles.append((x, y, r))

    for x, y, r in circles:
        I.ellipse((x - r, y - r, x + r, y + r), outline=(255,
0, 0, 0))
```

Image Pyramids :

- Image Pyramids – לוקחים תמונה ושומרים אותה בגדלים שונים- עותקים שונים- יורדים בחצי בכל ממד. כל רמה (גודל תמונה) נקרא- level.

למה נעשה את זה?

זיהוי edges -ככל שנקטין, המרחק בין כל פיקסל קטן (בין פיקסל לפיקסל) וכך הedge יותר ברור.

בתמונות הקטנות- פחות אינפורמציה אבל אז נקבל דברים גדולים בצורה יותר ברורה.

molty scaly – כאשר רוצים למצוא משהו- נקטין ונקטין- עד שנמצא את זה. שימוש ב scaling שונים.

נלמד איך להקטין את התמונה בצורה אופטימלית- כך שנאבד כמה שפחות נתונים (דחיסת נתונים).

האלגוריתם להקטנה:

- **naïve**- כל פיקסל שני נמחק- לא טוב כי אז נאבד עמודים שלמים.

- **Aliasing** - בעקבות שינוים שמתרחשים בתמונה או דסיגנל חד פעמים אם דוגמים בצורה נאיבית ולא מתחשבים בדברים קריטיים (שמפורטים בהמשך) התוצאה שנקבל לא תייצג בצורה אמיתית את הסיגנל שדגמנו ממנו.

- **sub-sampling** - הפעולה שבה אני לוקח תמונה ומקטין אותה פיזית.

הקושי בנושא זה הוא שכאשר אעשה down sampling (נוריד כמה פיקסלים) ואז נעשה up sampling לא נקבל את התמונה המקורית בכלל- אפילו לא דומה לה.

הדרך הכי טובה בהקטנת תמונה היא:

נעשה טישטוש כל פעם שמקטינים(subsampling).

- **Reduce** :

מקבל תמונה NxM והפלט הוא- N/2xM/2 (יודע ליצור עוד רמה בפירמידה.

האלגוריתם :

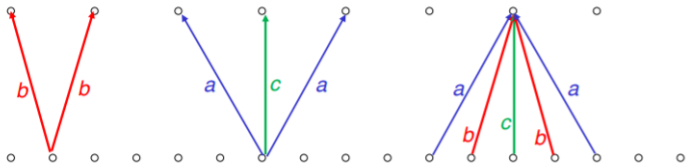
1. Blur- convolve and gaussian filter.
2. Sub-sample- הקטנת גודל התמונה (בחירת 2 פיקסלים להוצאה).

אפשר לחשב עבור כל פיקסל איך הוא יתרום לשלב הבא.

Gaussian Smoothing Kernel

The contribution of each pixel for the pixels in the next level

Example: 5 Weights: (a, b, c, b, a)



Conditions:
c>b>a (Unimodal)
2a+2b+c=1 (Receiving weights)
c+2a=2b (Contributing weights)

Commonly Used - Binomial Coefficients
1 2 1
1 4 6 4 1
1 6 15 20 15 6 1

- **reduce operation** – נעשה פילטור- ניקח 3 פיקסלים ונעשה להם ממוצע ונעשה קפיצות של 1

– כך גם נעשה filter וגם נצמצם ב1.

בסופו של דבר לא יהיה הרבה זיכרון לשמור:

$$NxN\ (1+1/4+1/16+...) = 4/3 * NxN$$

- Gaussian Pyramid - זה תהליך טוב כי אני נפטר מכל הaliasing.

- Reconstruct – איך חוזרים לתמונה גדולה יותר? בכל מקרה איבדנו מידע, המטרה: לאבד דברים שהעין לא מבחינה בהם.

- Expand Operation – שני השלבים:

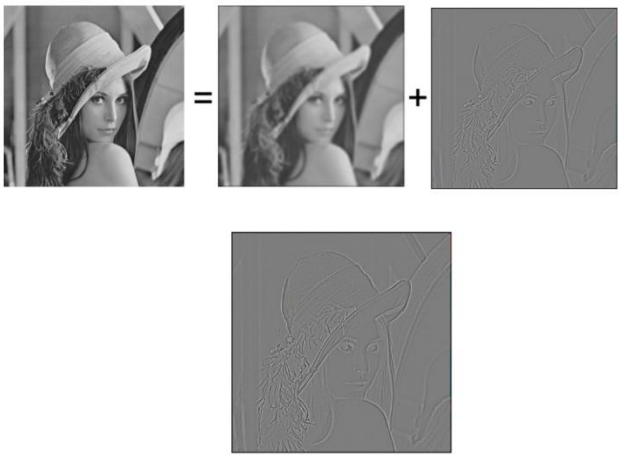
zero padding: לוקחים את הפיקסלים שיש לנו ומרפדים בניהם לגודל הרצוי באפסים.ולאחר מכן עושים: blur עם ה kernel שאיתו הקטנו אבל מנרמלים רק בחצי- לא בכל הסכום.

blur: נעשה נורמליזציה, ע״י blur kernel שיכול להיות שונה.

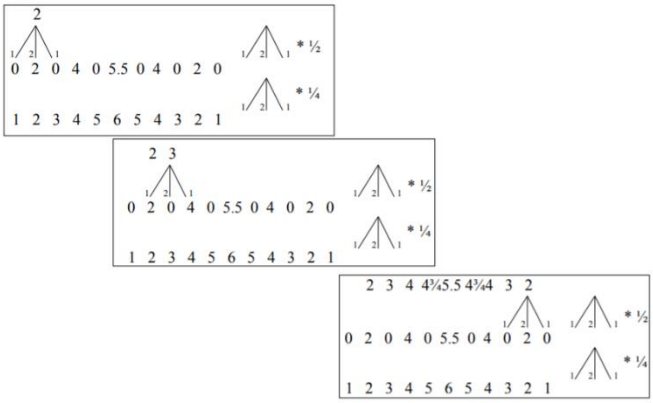
הפעולות שנעשה- נמצא ממוצע מ 3 פיקסלים נחלק במה שהעלנו(למשל ב1/2) ונמשיך לבא (לא נקפוץ כמו במצב ההפוך).

- Pyramid Level Difference -

$$L0=G0 - Expand(G1)$$



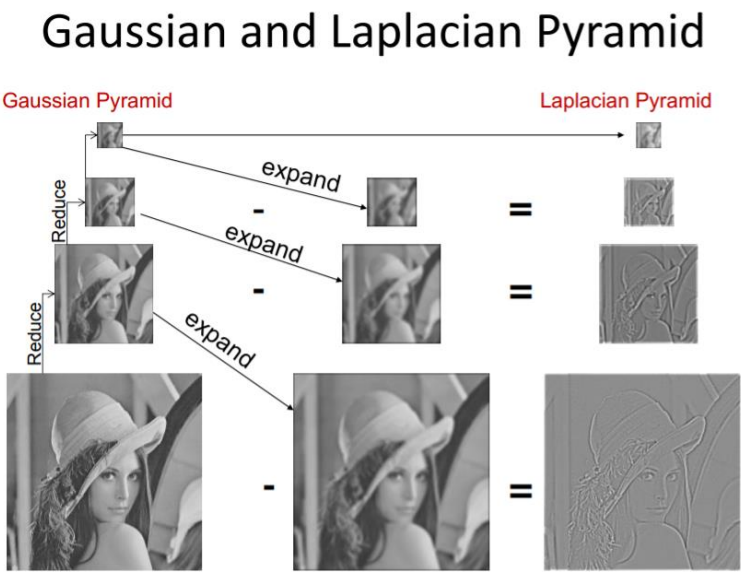
Reconstruct: Expand



חילקנו ב4 כי סכום הקרנל ב4 וננרמל את הקרנל בסכום הזה בשביל שהבהירות לא תעלה.

אם יש לי את ה expand(G1) (השחזור) וה L0 (Laplacian) אני יכול לשחזר את ה G0 (התמונה

המקורית) . כך אם כאשר יש לי את התמונה המקורית, אני מקטין אוטתה ומחסיר מהמקורית את ההקטנה אני מקבל תמונה של כל מה שחסר לי בהקטנה. וכך למעשה עם התמונה המוקטנת התמונת לפלסיאן אני יכול לשחזר בצורה מדויקת את תמונת המקור.



- Gaussian and Laplacian Pyramid -

פירמידת ההפרשים(Laplacian)- בשביל למצוא את כל הפרמידה המקורית (gaussian) אני צריך לשמור רק את Laplacian Pyramid. אבל רוב הערכים הם 0 אז אני יכול לעשות קוונטיזציה (הורדת רמת הפיקסלים) וכך לחסוך עוד יותר.

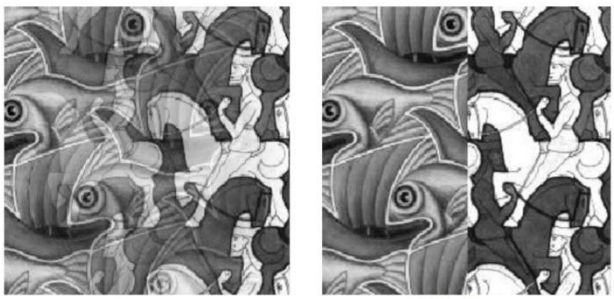
לסיכום- נצטרך לשמור רק את Laplacian Pyramid ועוד תמונה ממש קטנה (בפועל לא צריך גם את הקטנה כי בLaplacian Pyramid התמונה הקטנה ביותר כבר ממש זהה לתמונה הכי קטנה ב gaussian pyramid.

- Blending – חיבור בין שני תמונות ע״י mask (תמונה בינארית, כך שבלבן ניקח מתמונה 1 ובשחור ניקח מתמונה 2).

למה זה קשה? 2 תופעות שצריך להתמודד איתם:

1. Ghosting – רואים את 2 התמונות אחת על השנייה, נוצר
2. Visible seams-יהיה קו תפר, לקחנו בצד אחד פיקסל אר

למעשה הקושי הוא במציאת האיזון הנכון לחיבור בין התמונות.



Ghosting Visible Seams

איך נפתור את הבעיה? ניצטרך להבין כמה פיקסלים לוקחים מאיזה תמונה ואיזה תדירות. נעשה זאת ע״י חלון : בהינתן 2 תמונות, נרצה לבחור בקו התפר גודל חלון שיקח מתמונה 1 ומתמונה 2. וגם נרצה לבחור משקל שלפי הכמות שאקח מכל תמונה, המעבר יהיה נעים לעין.

בקיצור- לא רוצים שיהיה יותר מידי ghosting ולא יותר מידי visible seams כדי למנוע נבחר גודל חלון לפי השינויים בתמונה.

איך נמצא את השינויים בתמונה? ע״י פירמידות, השלבים הם :

1. ניצור pyramid Laplacian לתמונות A (La) ו B (Lb).
2. ניצור pyramid gaussian לmask (Gm).
3. נתחיל ברמה הכי נמוכה ונעשה עליה את הנוסחה :

$$L_c(i,j) = G_m(i,j)L_a(i,j) + (1 - G_m(i,j))L_b(i,j)$$

Pyramid Blending Arbitrary Shape

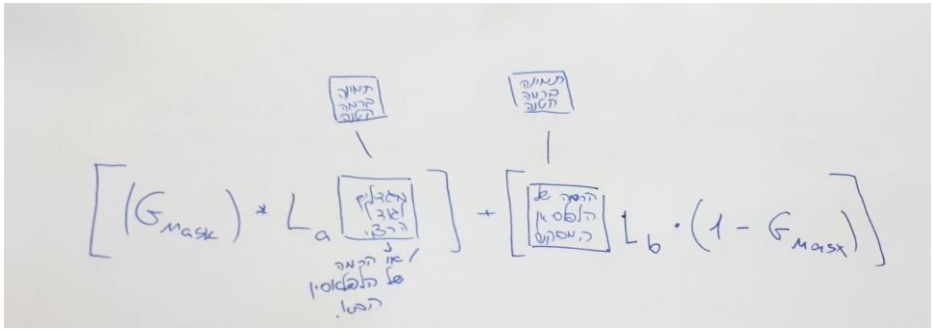
- Given two images *A* and *B,* and a binary image mask *M*
- Construct Laplacian Pyramids L_a and L_b
- Construct a Gaussian Pyramid G_m
- Create a third Laplacian Pyramid L_c where for each level *k*

$$L_c(i,j) = G_m(i,j)L_a(i,j) + (1 - G_m(i,j))L_b(i,j)$$

- Sum all levels L_c in to get the blended image

4. וכך נמשיך עוד ועוד עד הרמה האחרונה.

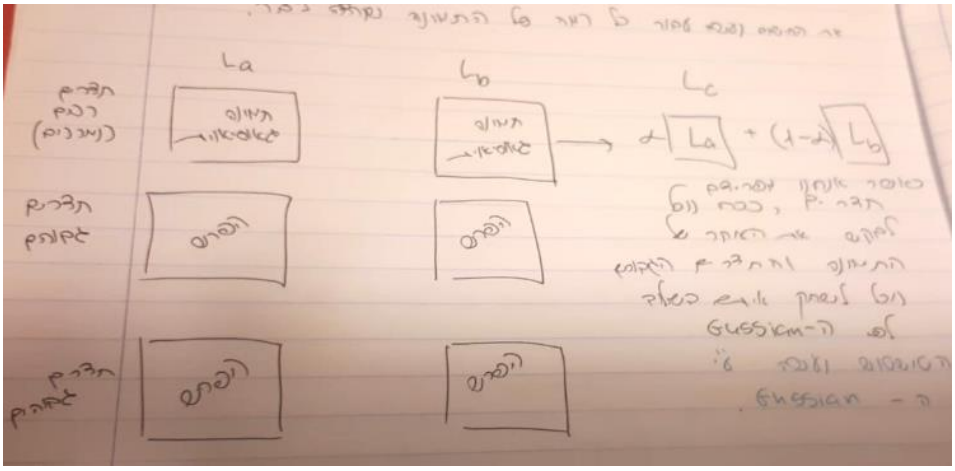
מדוע אנחנו עושים את הפירמידות והשלבים האלו? כי כאשר עושים פירמידה של לפלסיאן למעשה מקבלים את התדרים הכי חזקים ברמות הקטנות והתדרים הקטנים בתמונות הגדולות. כך אנחנו מחברים בין התדרים הגדולים תחילה וכן הלאה, דבר זה מונע ממנו לחתוך או לטשטש דברים בתמונה, יוצר את זה בצורה יותר חלקה וכך אנחנו מזהים את השינויים ומחברים אותם טוב יחד ומונעים מזה להיראות כחיתוך.



למעשה זה תמיד יהיה Laplacian ולא התמונה הרגילה. כך נקבל את התמונה A רק בmask השחור ואת הB רק בmask הלבן (כמובן שאפשר להפוך).

**********מימוש הקוד******

סוכם ע״י נעמי צברי



שיעור 6

- Optical flow - נשתמש כשנרצה לתאר תנועה בעולם למשל אם יש לי אוסף של נקודות בעולם שקיימות בצילום a והזזתי את המצלמה וצילמתי גם תמונה b אז יש הבדל במיקום של נקודות אלו- במיקום של ה- avg.

הנחות:

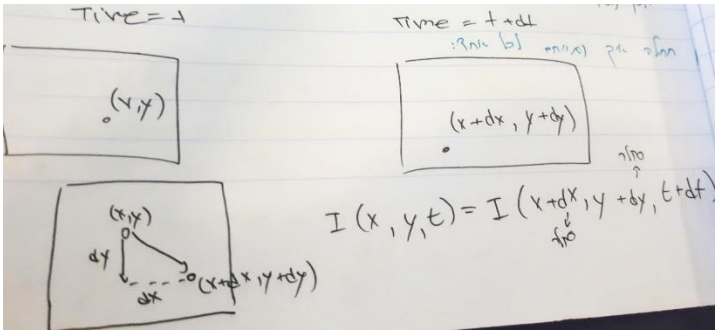
1. brightness consistency – ה intensity של פיקסל מסוים לא משתנה בפריימים (בתמונה אחרת) אם הוא היה 25 הוא יישאר 25.

2. נניח שאזורים גדולים חולקים אותו intensity.

נרצה ליצור מפה שתראה לנו מתמונה A לתמונה B כל נקו לאן היא זזה.(רק וקטורים שמתארים את התנועה של הפיקסל מזמן T לזמן T+1 מתייחס רק בדו ממד. ב motion field נתייחס כתלת ממד אז יש לי וקטור שמתאר לי גם את התנועה במרחב).

איך נדע לסווג פיקסל לפיקסל בתמונה השנייה?

הגדרות : ככל שהדגימות שלי יהיו יותר מהירות כך השינויים שאני אראה בתמונה יהיו קטנים (ואז נזהה יותר בקלות את התזוזה ונוכל לסווג לאיזה פיקסל). ולכן נוסיף עוד פרמטר T- זמן. וכך נוכל להתייחס לאוסף תמונות.



אלגוריתם :

מגדירים פונקציית error :

כמה אכן צדקנו וכמה טעינו :

סכמים את כל ה error^2 : MSE :

$$E(u,v)=\sum_x\sum_y\left(I_1(x,y)-I_2(x+u,y+v)\right)^2$$

ככל שהתוצאה תהיה קרובה ל-0 כך אנחנו יודעים שה x וה y מדויקים.

איך נמצא את ה u וה v האלו?

לא משנה לנו אם נכפיל/נוסיף כל עוד מוסיפים לכולם. יש לי 2 וקטורים ואני רוצה למצוא את ה correlation ביניהם ולהקטין אותו.(נרצה לצמצם לנו את הסטיות שאם הintensity שלי גדל/משתנה) אבל אנחנו עדין מניחים שלכל התמונה יש אותו (u,v) ואני מחפש הכל בצורה דיסקרטית (באופן נבדל) : **********המשך הסבר על :

Normalized Cross Correlation

- Given two images I_1 and I_2 , search for the translation (x, y) maximizing the cross-correlation

$$C(u,v)=\sum_x\sum_y I_1(x,y)\cdot I_2(x+u,y+v)$$

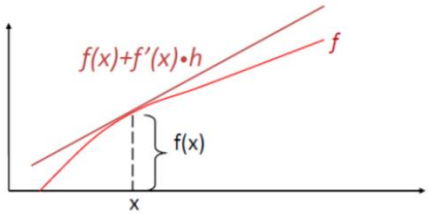
- Normalized Cross Correlation eliminates additions and multiplications effects

$$NC(u,v)=\frac{\sum(I_1(x,y)-\hat{I}_1)\cdot(I_2(x+u,y+v)-\hat{I}_2)}{\sqrt{\sum(I_1(x,y)-\hat{I}_1)^2}\sqrt{\sum(I_2(x,y)-\hat{I}_2)^2}}$$

- Motion field – אוסף של וקטורים, כל וקטור מחובר למקום בעולם, והוא אומר לי איפה זה נימצא בעולם. מתאר את התנועה של ה edges בתלת ממד.
- טור טיילור-

- Local Taylor approximation in 1D:

$$f(x+h) \approx f(x) + f'(x) \cdot h$$



- Local Taylor approximation in 2D for images:

$$f(x+u, y+v) \approx f(x,y) + \frac{\partial f}{\partial x} \cdot u + \frac{\partial f}{\partial y} \cdot v$$

- Lucas-Kanade: Taylor Approximation – נשתמש בתור טיילור למציאת השינוי בין התמונות.

Lk נותן דרך פשוטה לפטירת ההנחה brightness consistency (שהאינטנסיטי נשאר אותו דבר)
אך זה עובד רק לתזוזה קטנה (כי בתזוזה קטנה המשוואה לינארית) וכך זה נפתר ביתר קלות, כיוון שטורי טיילור מתלכדים עם המשוואה כאשר היא לינארית.
מקבלים 2 תמונות והמטרה למצוא (u,v) שמתארים את התזוזה של כל פיקסל.

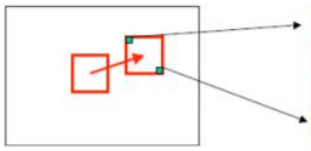
$$\text{Our cost function: } E(u,v) = \sum_{x,y} (I_x u + I_y v + I_t)^2$$

$$\text{Our goal: } \min_{u,v} E(u,v)$$

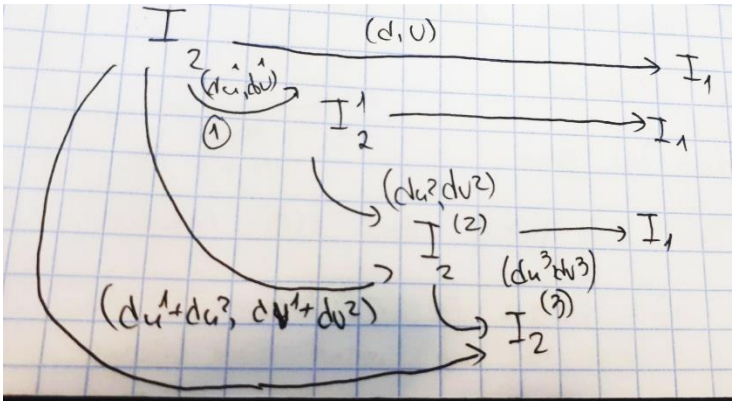
Aperture Problem – אין לי מספיק מידע בשביל לדעת מה התוצאה הנכונה בעולם. אני רואה משהו דרך חור קטן ואין לי מספיק מידע בשביל לדעת את זה. יש לי יותר נעלמים ממשוואות.
נשנה הנחה: אם לוקחים חלון אז החלון זז באותה צורה. ולכן נגדיר u ,v לכל חלון.
מה זה אומר שהחלון זז באותה צורה? שכולן חולקים את אותו u, v. ולכן נוכל לרשום:

$$I_x u + I_y v = -I_t \Rightarrow \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -I_t$$

Assume constant (u,v) in small neighborhood



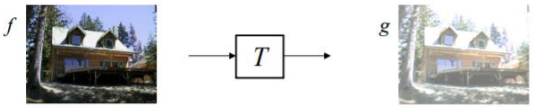
$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} I_{t1} \\ I_{t2} \\ \vdots \end{bmatrix}$$



- Image warping – נעשה זאת כאשר נרצה לקחת תמונה וליישר אותה שתראה כמו במציאות. נירצה גם לגרום שהפיקסלים יסודרו בצורה טובה, העתקת פיקסל והדבקה במקום אחר בהתאמה בתמונה החדשה.

image filtering – משנים את ה intensity של התמונה.

$g(x,y) = T(f(x,y))$



$g(x,y) = f(T(x,y))$

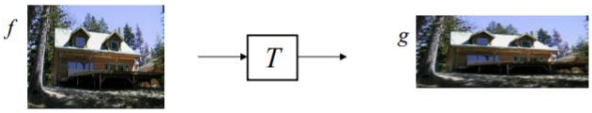


image warping – לשנות את הקורדינטות בתמונה.

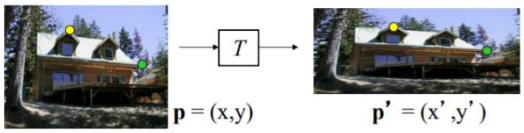
נצטרך למצוא את ה T הזה שממשיך מ (x ,y) ל (x′ ,y′)

יש T שיתאים להכל, ויש T שיתן לכל פיקסל לתזוזה שונה למשל optical flow.

T גלובלי פרמטרי- אותו T לכל התמונה (קוראים לזה פרמטרי כי יש לזה נוסחה).

T לוקלי – T שונה לכל פיקסל (כלומר כל פיקסל זז אחרת).

Global Parametric Warping



Global transformation *T*:

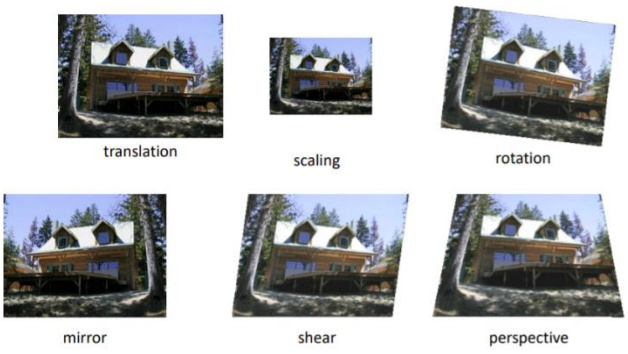
- Has the same form for any point *p*
- Determined by just a few numbers (parameters)

We will represent *T* by a matrix: p' = **M**p

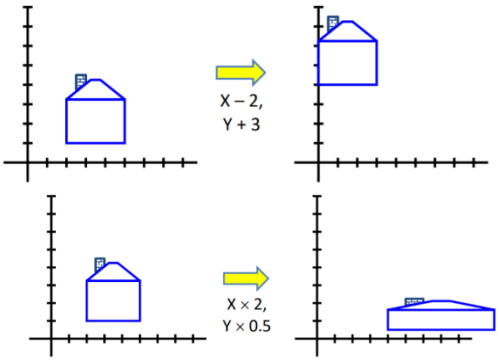
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

Global Parametric Warping

Examples of parametric warps:



- Translation – לכל אחד מהפיקסלים נוסיף סקלר למשל x-2, y+3.



scaling – מגדילים כל אחד מהקורדינטות בקבוע (סקלר).
uniform scaling- (אחיד) נזיז כל חלק בתמונה בסקלר קבוע -אותו סקלר.
non-uniform scaling – סלקר שונה לכל חלק בתמונה.
איך מייצגים זאת ככפל במטריצה?

Scaling

The scaling operation:

$$x' = ax$$
$$y' = by$$

In matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

(What is the inverse of S?)

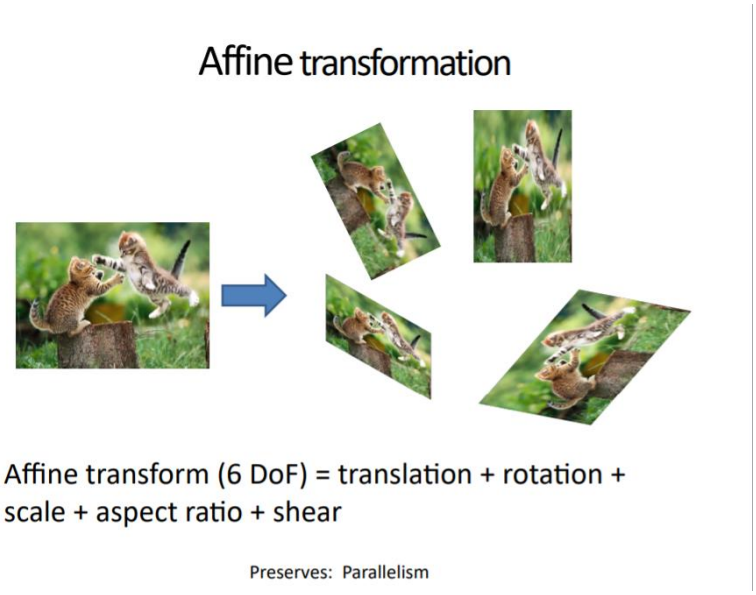
----שיעור 10-----

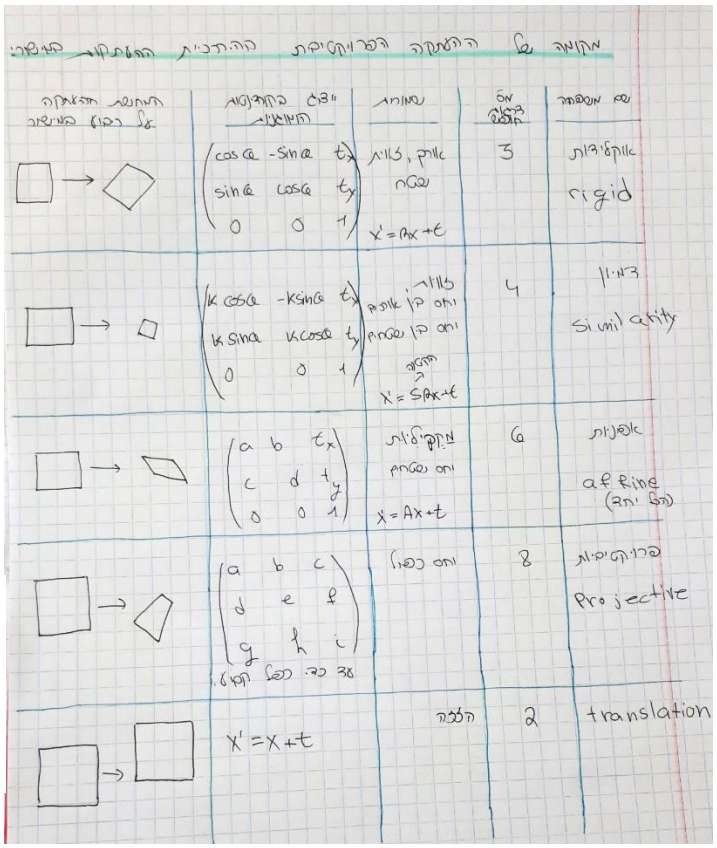
מצגת מפורטת על השיעורים הבאים- גיאומטריה:

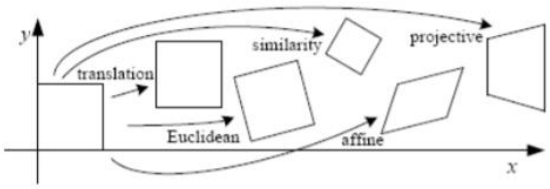
https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws11-12/3DCV_WS11-12_lec04.pdf

- Transformations –

- Parametric (global) Transformations -





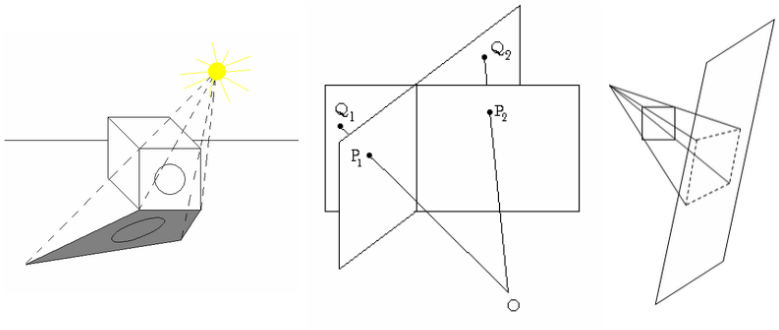


Scaling- מגדילים כל אחד מהקורדינטות בקבוע. לפעמים לא נרצה שיהיה לנו שינוי בפיקסל מסוים ולכן נכפיל ואז נזיז אותי בחזרה.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

* מטריצה אלכסונית מתאימה כמו סקלה.

- Homography- להומוגרפיה (העתקה פרויקטיבית) יש שני שימושים:
  1. איך משטח יראה אם נזיז את המצלמה ביחס אליו (רק במשטח בדו ממד) – התאמה בין תצלומים של משטחים מזוויות שונות
  2. איך העולם נראה אם נזיז את המצלמה בזווית קטנה (בתלת ממד) – התאמה בין תצלומים של העולם בתנועה בציר אחד (?)



https://he.wikipedia.org/wiki/%D7%94%D7%A2%D7%AA%D7%A7%D7%94_%D7%A4%D7%A8%D7%95%D7%99%D7%A7%D7%98%D7%99%D7%91%D7%99%D7%AA

כשמחשבים את התנועה בתמונה אנחנו תמיד מניחים מה הייתה התנועה- מה ה motion model. ואז לפי כך מחשבים את הטרנסלציה. מה יכול לקרות? שיהיו עיוותים בתמונה כי התזוזה שהנחנו לא הייתה התזוזה היחידה (לדוגמא הנחנו תזוזה מקבילה לאדמה אבל היד של הצלם קצת גרמה לסיבוב ביחס לאדמה).

----------שיעור 11-----

- RANSAC- כאשר נירצה לחבר שני תמונות (שיש להן חלק משותף-פנורמה) איך נעשה את זה?

עבור general model: כאשר ניתן לנו model type יש לו minimal set: המספר הקטן ביותר של נקודות (פיקסלים) שלפיהם המודל יוכל לחשב משוואת ישר או transform. עבור ישר- 2 נקודות אבל עבור transform? הם רק מודלים, ולכן המינימום סט של נקודות שצריך עבור transform שונות? נצטרך רק אחד, מדוע? כי ברגע שיש לי נקודה אחת בתמונה 1 אני יודעת לחשב לאן היא זזה בתמונה השנייה ולפי זה אני יכולה לחשב את כל התזוזה של כל הפיקסלים. (ע״י מה שלמדנו למציאת רמת השינוי בין שני תמונות- נעשה חישוב לx והy ולפיהם נימצא את ה-subtract).

homography- נצטרך 4 נקודות כדי לחשב זאת, מדוע? כי לכל מצלמה צריך 2 נקודות.

האלגוריתם:

הנחה: שניקח כל מיני סטים של נקודות ומתישהו נמצא סט שרוב הנקודות שלנו נמצאות על הקו.

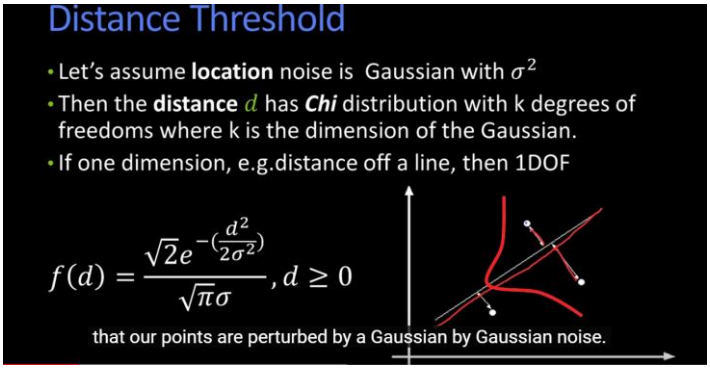
1. רנדומלית ניבחר s נקודות (זוגות נקודות) כדי להציג דוגמה (לפי ההנחה).
2. Instantiate the model- ליישר את המודל.- לראות כמה נקודות נמצאות על הישר שמצאנו.
3. Get consensusset Ci – the points within error bounds(distance threshold) of the model
4. If |Ci|>T, terminate and return model
5. repeat for Ntrials, return model with max |Ci|

בגדול- ניקח סטים -החשב את הישר ביניהם (נוציא נקודות שהורסות את הישר לרוב הנקודות)(ונחשב

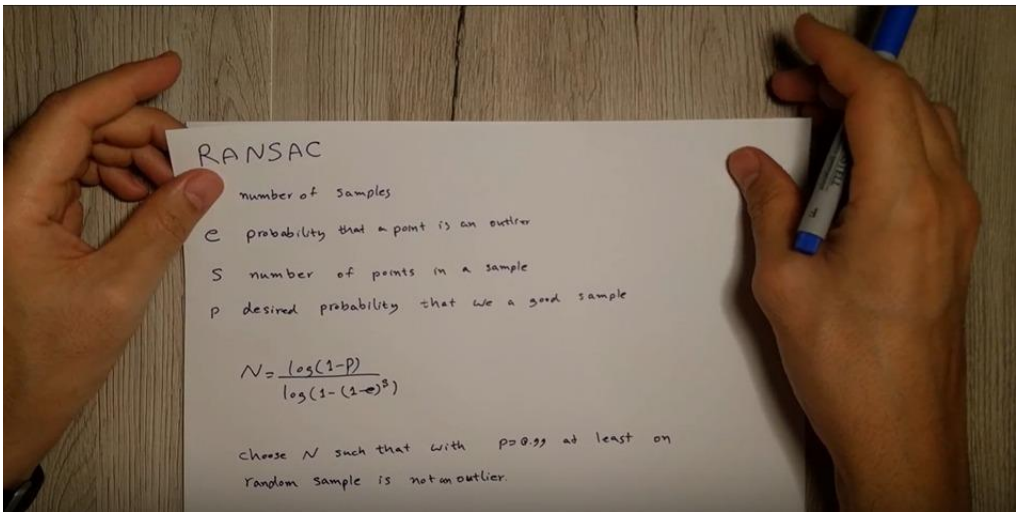
כמה נקודות (אחרות) אכן קרובות לישר הזה (נמצאות בתוך הinliner) אם הthreshold נמוך יותר ממה שמצאנו לפני- נחליף,נחזיר למעשה את הסט עם מספר הנקודות המקסימלי שנמצאות על הישר או הכי קרובות אליו, לפי ישר זה נחשב את החיבור בין כל שאר הנקודות בין שני התמונות וכדו.

בחירת הפרמטרים:

1. עבור כל מודל מסוים נצטרך למצוא עבור מספר נקודות מסוים את המשוואות ביניהם,למשל בין ישרים-2 נקודות בין homography נצטרך למצוא 4 נקודת והמשוואות שלהם לפיהם נידע איך לחבר את התמונה, איך התמונה זזה מנקודות אלו לנתונה השניה.
2. המרחק של הt threshold צריך להחליט מה הגבול שנחשב בתוך הישר-inlier.



3. מהו מספר הפעמים שנחפש סט למשוואה? נרצה לבחור N יחסית גדול, נצטרך לחשב את אחוז הטעות שלפיו נידע מהו הN שלנו :



כאשר **e** הוא ההסתברות שהנקודה היא outliner- למעשה נחפש את ההסתברות שזה אכן בתוך הקו-inliner.

ו**p** הוא ההסתברות הרצויה -שאנו מחשיבים לטובה- למשל 99%.

s- מספר הנקודות שבמודל.

המכנה מחשב לנו מה ההסתברות שלפחות נקודה אחת במודל או יותר הם inliner.

הסבר- https://www.youtube.com/watch?v=UKhh_MmGIjM

הסבר ב udacity -

https://classroom.udacity.com/courses/ud810/lessons/3189558841/concepts/31679389260923

- Camera calibration – כיול (בדיקה כמה הכלי מדידה מדויק)מצלמה הוא הוא תהליך של הערכת פרמטרים של המצלמה באמצעות תמונות של דפוס כיול מיוחד. הפרמטרים כוללים camera

intrinsics. למעשה זה תהליך של הערכת פרמטרים מהותיים ו / או חיצוניים. פרמטרים פנימיים עוסקים במאפיינים הפנימיים של המצלמה, כמו אורך המוקד שלה, הסטייה, העיוות ומרכז התמונה. פרמטרים קיצוניים מתארים את מיקומו ואת התמצאותו בעולם. הכרת פרמטרים מהותיים היא צעד ראשון חיוני לראיית מחשב תלת ממדית, מכיוון שהיא מאפשרת להעריך את מבנה הסצנה במרחב האוקלידיסטי ומסלקת את עיוות העדשות, המשפיל את הדיוק. BoofCV מספק כיול אוטומטי לחלוטין למספר סוגי יעדים מישוריים (ראה תמונות לעיל) הניתנים להדפסה בקלות על נייר בגודל סטנדרטי.