

סיכום עיבוד תמונה וראייה ממוחשבת:

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** -
- **PDF** -
- **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 ו 1.

- 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 שנוזיז 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

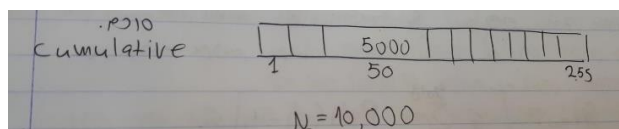
- The solution ($R = g_{old}$, $S = g_{new}$)

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

- Notes:

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

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



נעביר מ0 ל 127 את כל ה5000 שנמצאים בגוון 50.

2. נשמור על יחס סדר, אם פיקסל A יותר בהיר מפיקסל B נרצה שהיחס בין 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.

שלבי האלגוריתם:

1. מחשבים הסטוגרמה של התמונה
 2. מחשבים את ה cumulative
 3. מנרמלים.
 4. מכפילים את הנרמול בערך המקסימלי של ה gray level (255).
 5. 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 מהתחלה עד הסוף (וכך נשמור על היחס).
- מתי האלגוריתם לא יעבוד?** שהנחות לא נכונות, שיש 2 תמונות יחד ושההתפזרות של ה graylevel לא אותו דבר.

איך נפתור את זה?

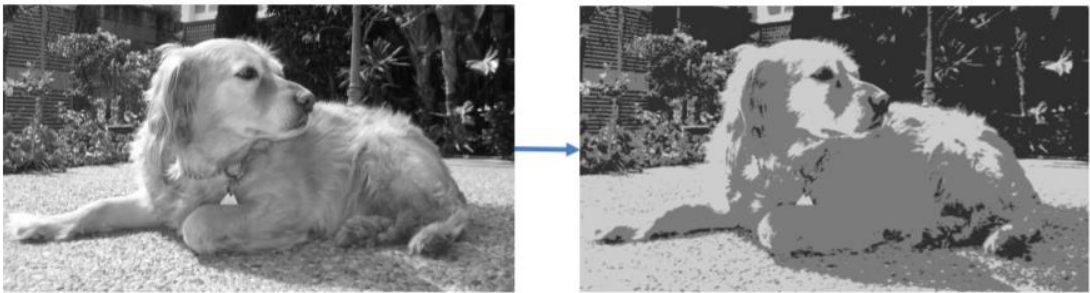
במקרים של תמונות שה intensity צריך להיות בצורה שהיא- קיצוני לצבע מסוים. אז נצטרך לעבור ל high level – נעבד את התמונה לפי איזורים.

עבור כל חלק מהתמונה נעשה כמו שעשינו בתמונה שלמה (נחשב לאן היא צריכה לעבור בתמונה המשופרת) רק שכאן נחשב עבור האיזור 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.

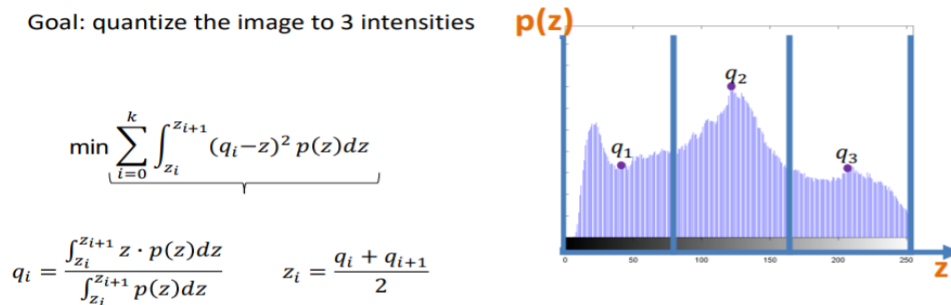
איפה זה יכשל? במעברים, אז איזה חלון ניקח? צריך שלא יהיה קטן מידי כי אז הוא לא ישנה כלום אבל גם לא גדול מידי-שיתשתש יותר מידי.

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



Quantization - solution

Goal: quantize the image to 3 intensities



Given: K - the number of values we want

Goal: Find the boundaries (z) and the values themselves (q)

q_i – הערך שאתה רוצה שיהיה מינימלי.

z_i – 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.

- **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.

Sliding window - עבור כל פיקסל בתמונה :

ניקח גודל חלון $k \times k$.

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

• -Max Filter

• רעש הערות כלליות-

במצאות הייצוג של סוגי הרעש השונים הינו דיסקרטי (ולא רציף), כלומר היסטוגרמות בניגוד ל $s'pdf$

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 סינטי) לתמונה נתונה

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

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

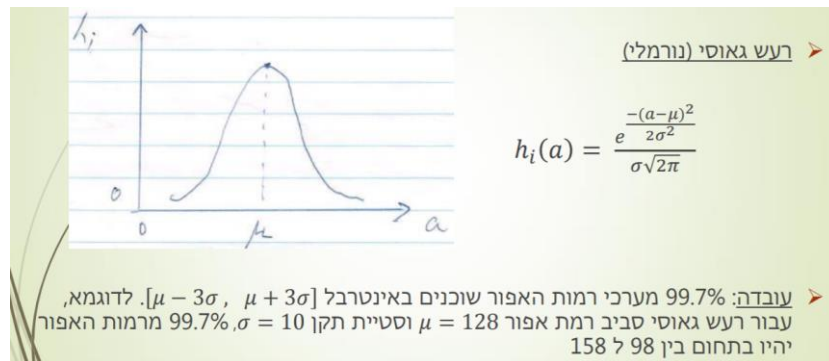
סילוק רעש multiplicative קשה יותר ומצריך עיבוד לא-ליניארי.

• **Salt & Pepper Noise** – (רעש מלח פלפל) הוא רעש על תמונה של כל מיני נקודות רנדומליות של לבן

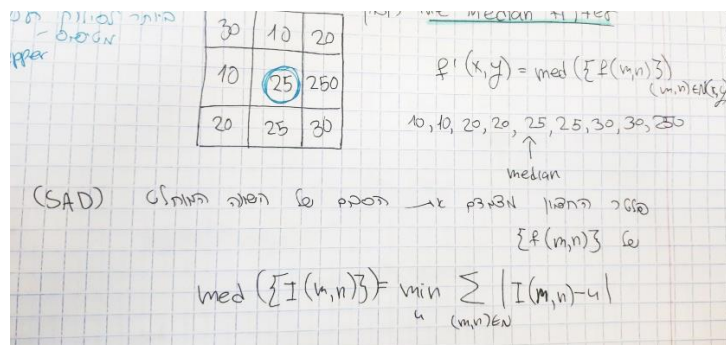
ושחור ז"א ערכים קיצוניים בפיקסלים מסוימים. נוצר לפעמים מחומר התמונה, הצטברות אבק

במערכת האופטית וכדו. נוכל לייצר אותו אותו באופן סינטי ע"י המשוואה :

• **רעש גאוס** -



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



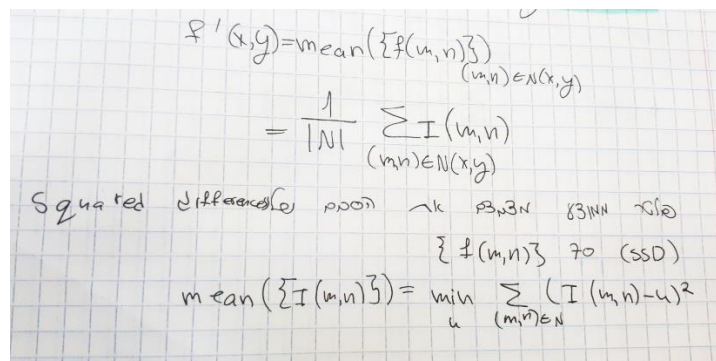
Median filter algo

```

1 def medianFilter(img, ksize):
2     hsize = ksize // 2
3     retImage = np.zeros(img.shape)
4     h, w = img.shape
5     for i in range(h):
6         for j in range(w):
7             box = img[i-hsize:i+1+hsize, j-hsize:j+1+hsize]
8             retImage[i, j] = np.median(box)
9     return retImage

```

The Average filter (ממוצע) -



Filter, kernel and mask are the same!

- **Convolution** – במתמטיקה (בפרט, ניתוח פונקציונלי) קונבולוציה היא פעולה מתמטית בשני פונקציות (f ו g) לייצור פונק' שלישית המבטאת את אופן שינוי הצורה של האחד על ידי האחר. (המונח קונבולוציה מתייחס הן לתפקוד התוצאה והן לתהליך המחשוב שלה. זהמוגדר 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 קונבולושן ובכך נקבל את אפקט החלון שרצינו. הפעולה תתבצע כך : עבור כל פיקסל, ניקח חלון בגודל $K \times K$, נחשב את הסכום של החלון ונשנה את הערך של הקורדינטה של הפיקסל.

עבור D1: (חד ממדית)

הרעיון: + אות -> הדיקטור + מספרים של פילטר + המער או 8 על 6 הפקסל + הפסל -> הסכום. הנוסחה:

$$h(x) = (f * g)(x)$$

Signal/Image ←
Mask, Filter, Kernel

$$= \sum_{i=-\infty}^{+\infty} f(x-i)g(i)$$

דוגמה: $h = f * [2 \ 0 \ 3]$

$$h(x) = (f * g)(x) = \sum_{i=-\infty}^{+\infty} f(x-i)g(i)$$

תוצאה: $h(x) = 2f(x-1) + 3f(x-1)$

עבור D2: (דו ממדית)

- **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 / אות, אנו רוצים למצוא 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. אורת 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%20

[20is%20a%20technique,to%20detect%20edges%20in%20images.](#)

האלגוריתם:

1. צריך – מטריצת תבנית בגודל $K \times K$, מטריצת תמונה בגודל $N \times N$.
2. עבור כל $K \times K$ 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.

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

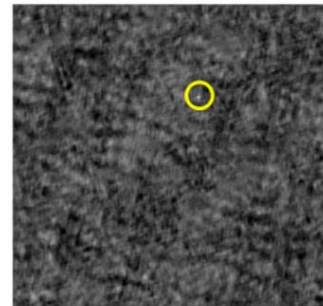
Zero normalized cross correlation

- $\vec{\alpha} = \vec{\alpha} - \text{mean}(\vec{\alpha})$
- 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.

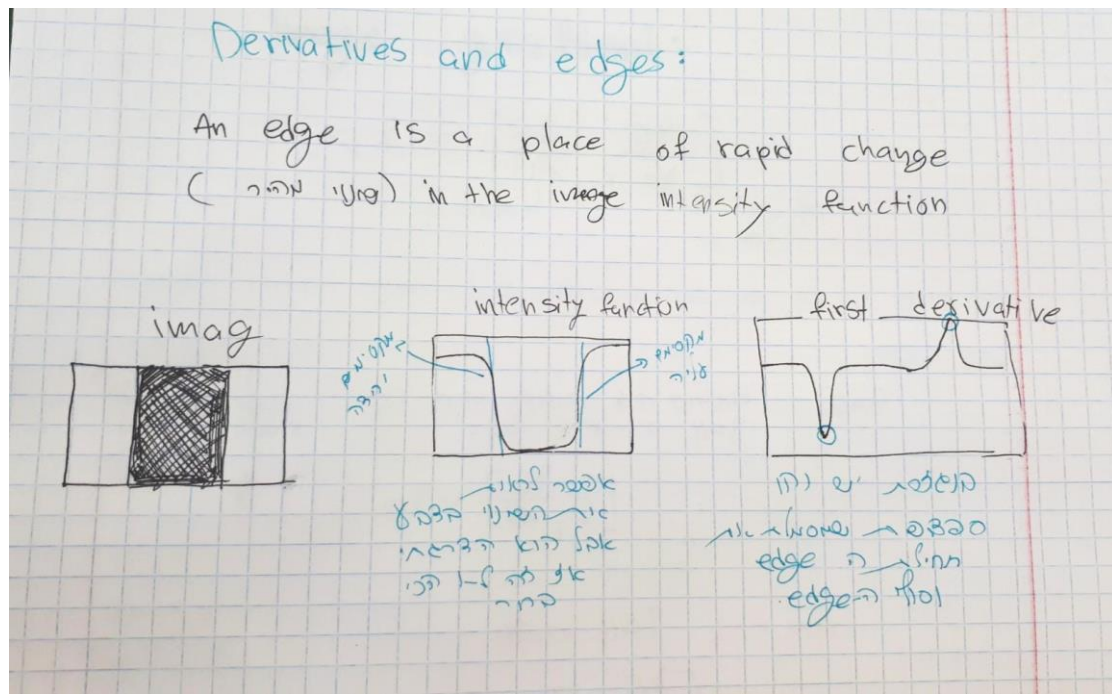
• **Correlation Map** - *****



Filter Response

:Image Derivatives: Edges and Sharpening

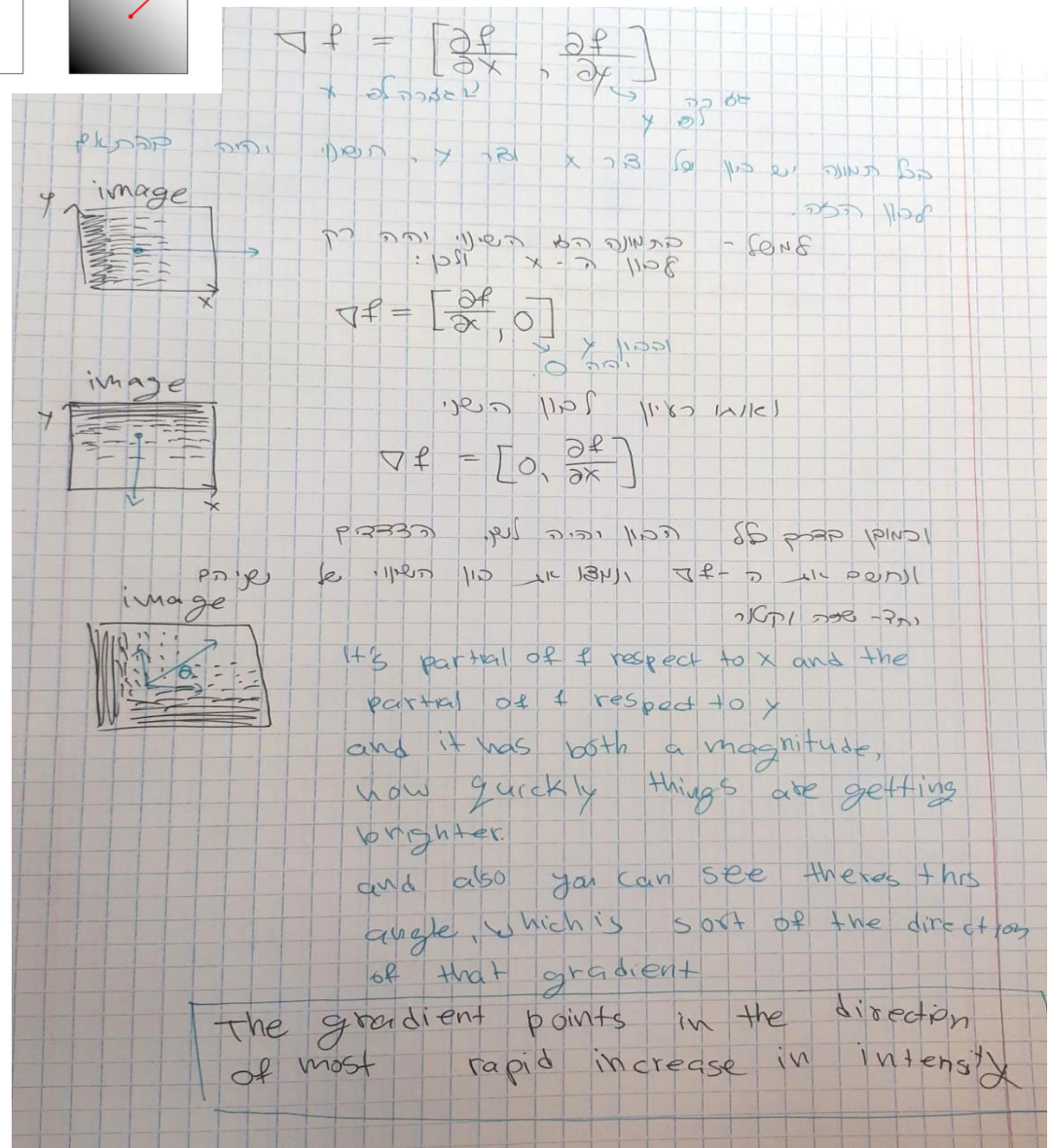
- Edges •



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0 \right] \quad \nabla f = \left[0, \frac{\partial f}{\partial y} \right] \quad \nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$



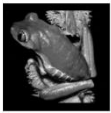

- Image gradient •


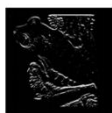


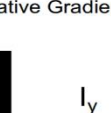
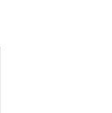
The Gradient - Properties

Gradient Derivatives and Magnitude


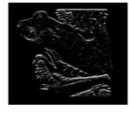
Gradient Derivatives and Magnitude

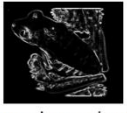
Magnitude - $|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$ $I_x =$  $\ast \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} =$ 

Direction (orientation angle) - $\alpha = \tan^{-1}\left(\frac{\left(\frac{\partial f}{\partial y}\right)}{\left(\frac{\partial f}{\partial x}\right)}\right)$ $I_y =$  $\ast \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} =$ 

Directional derivative - $\cos(\alpha)\frac{\partial f}{\partial x} + \sin(\alpha)\frac{\partial f}{\partial y}$ $I_y =$  $\ast \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} =$ 

1st Derivative Gradient Magnitude

$I_x =$  $I_y =$ 

$\sqrt{I_x^2 + I_y^2} =$ 

Magnitude can be used as a simple Edge Detector

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 edgen בתמונה. בשביל למצוא את ה edge נצטרך למצוא איפה הוא מתחיל ואיפה הוא נגמר, באיזה מיקום. בשביל זה השתמש ב gradient שיבליט לנו את השינויים בתמונה. לכל edge ניתן לייחס עוצמה (יחסית לגודל השינוי) וכיוון (אוריינטציה), באשר הכיוון הוא בניצב (normal) edge.
- 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).
- 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.

Smoothing

Box Kernel

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$


Input Image



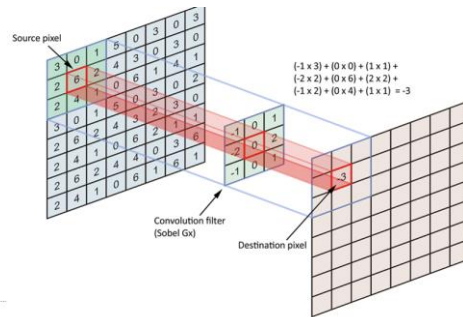
Gaussian Kernel (3x3)

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

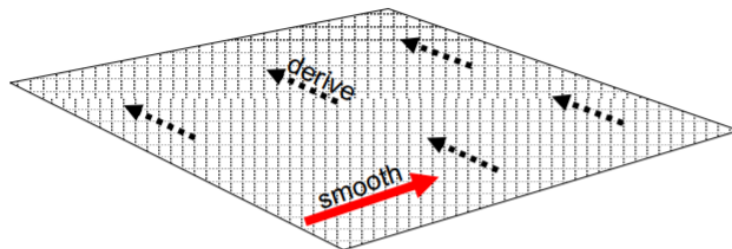
Output Image



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



- **Sobel** – להחליק את התמונה לכיוון אחד, ולחשב את הנגזרת בכיוון האורתוגונלי. (כך אנחנו מורידים רעש כיוני ולא הורסים את edge). בעזרת הנגזרת השנייה אני מוצא את המיקום של ה edge כי רוב הפעמים edge לא יהיה בינארי, ההבדל בינו לבין השאר הוא הדרגתי ולכן לא מייד, 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 קולנבולוציה.

```
# 3 Edge detection
def edgeDetectionSobel(I:np.ndarray)-
>(np.ndarray,np.ndarray):
    m, img_x, img_y = convDerivative(I)
    m, img_x, img_y = convDerivative(m)
    sobel_x = np.array([[ -1,  0,  1], [ -2,  0,  2],
                        [ -1,  0,  1]])
    sobel_y = np.array([[ 1,  2,  1], [ 0,  0,  0], [ -1,
                        -2, -1]])
    f_x = cv2.filter2D(img_x, cv2.CV_64F, sobel_x)
    f_y = cv2.filter2D(img_y, cv2.CV_64F, sobel_y)
    # (1/8)*((f_x ** 2 + f_y ** 2) ** (0.5))
    return (1/8)*f_x, (1/8)*f_y
```

- **Canny** - שלבי האלגוריתם:

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

$$|G(x,y)| = \sqrt{I_x^2 + I_y^2} \quad , \quad \alpha = \tan^{-1}\left(\frac{I_y}{I_x}\right)$$

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.

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 ל $T2$ 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)
```

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

האלגוריתם הוא:

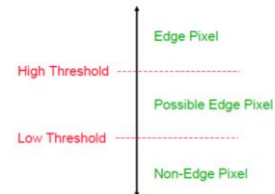
- נשתמש ב threshold גבוה ונמוך.

- 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 ל edge חזק (ברור).

Include possible edge pixels that are adjacent to edge pixels.

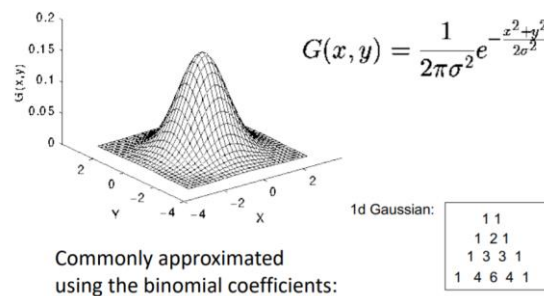


• **Zero Crossing Simple** - ***

• **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.

Smoothing with a 2D Gaussian Filter



- **-Derivative – Numeric**

קירוב נומרי לגראדינט

11

$$G_x \cong f[i, j + 1] - f[i, j]$$

$$G_y \cong f[i, j] - f[i + 1, j]$$

מסכות הקונבולוציה המתאימות הן:

-1	1
----	---

 G_x

1
-1

 G_y

כדי שהחישוב יתבצע ביחס לנקודה מסוימת קבועה (קרי, $[i + 1/2, j + 1/2]$), ניקח

-1	1
-1	1

 G_x

1	1
-1	-1

 G_y

ומכאן ברור כי עדיף השימוש בסביבת 3x3, למשל, כך שהגראדינט יחושב לגבי פיקסל מרכזי

- **Laplacian Zero Crossing – Laplacian gaussian** ??????????????????????

- **Laplacian sharpening** ???????

- **Image Sharpening** ?????

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

- איך מוצאים קו? (Line Detection) לרצף נקודות שיקווים 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.

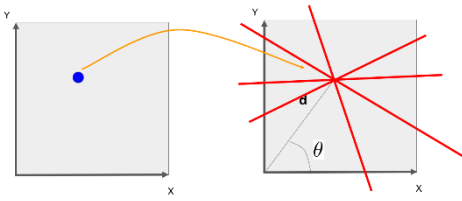
- **Lines – Hough space** – עבור כל נקודה יש ישר בHough spaces שבו יש את כל ה m, b האפשריים

בשבילו. ועבור 2 נקוי 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 Hough space ל"בינים" וניקח את ה"בין" עם הכי הרבה קווים שנמצאים בו. מה הבעיות שיש:

-הטווח אינסופי

- מה קורה שהקו מאונך? אין לנו m...

Recap from Hough Lines



• 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))
```

-----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



affine



Projective



translation

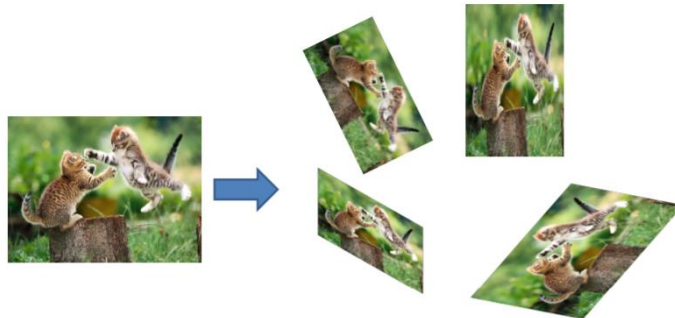


rotation



aspect

Affine transformation



Affine transform (6 DoF) = translation + rotation + scale + aspect ratio + shear

Preserves: Parallelism

תמונה	התמרה	הפרמטרים	מספר פרמטרים	שם
	translation	$\begin{pmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{pmatrix}$	3	rigid
	similarity	$\begin{pmatrix} k \cos \theta & -k \sin \theta & t_x \\ k \sin \theta & k \cos \theta & t_y \\ 0 & 0 & 1 \end{pmatrix}$	4	similarity
	affine	$\begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix}$	6	affine
	projective	$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$	8	projective
	translation	$x' = x + t$	2	translation

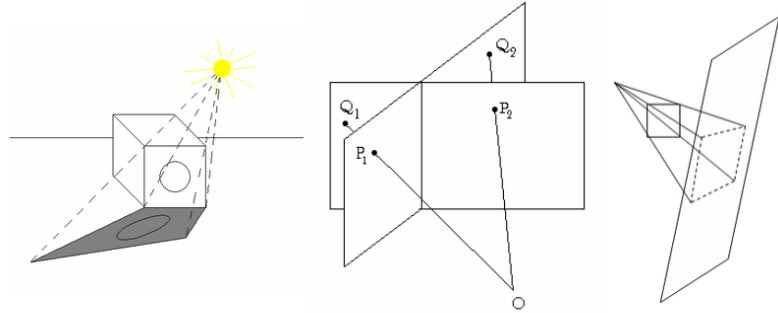
Scaling - מגדילים כל אחד מהקורדינטות בקבוע. לפעמים לא נרצה שיהיה לנו שינוי בפיסקל מסוים

ולכן נכפיל ואז נזיז אותי בחזרה.

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

④ טרנספורמציה אפיינה - ממשק - ממשק

- **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%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 משוואות 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. (עיי מה שלמדנו למציאת רמת השינוי בין שני תמונות- נעשה חישוב לא וזה הפיכים 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.

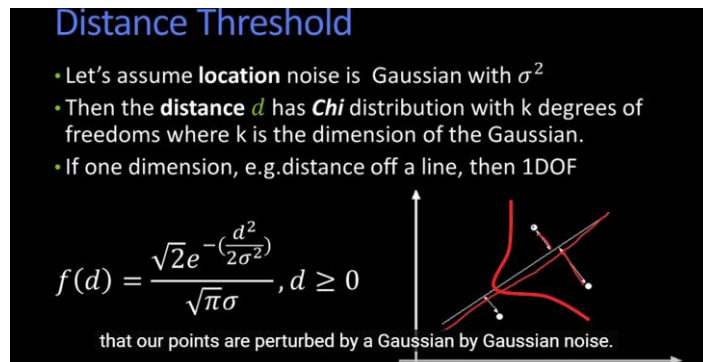
 1. רנדומלית ניבחר s נקודות (זוגות נקודות) כדי להציג דוגמה (לפי ההנחה).
 2. Instantiate the model - ליישר את המודל. - לראות כמה נקודות נמצאות על הישר שמצאנו.
 3. Get consensus set C_i – the points within error bounds(distance threshold) of the model
 4. If $|C_i| > T$, terminate and return model

repeat for Ntrials, return model with max |Cil .5

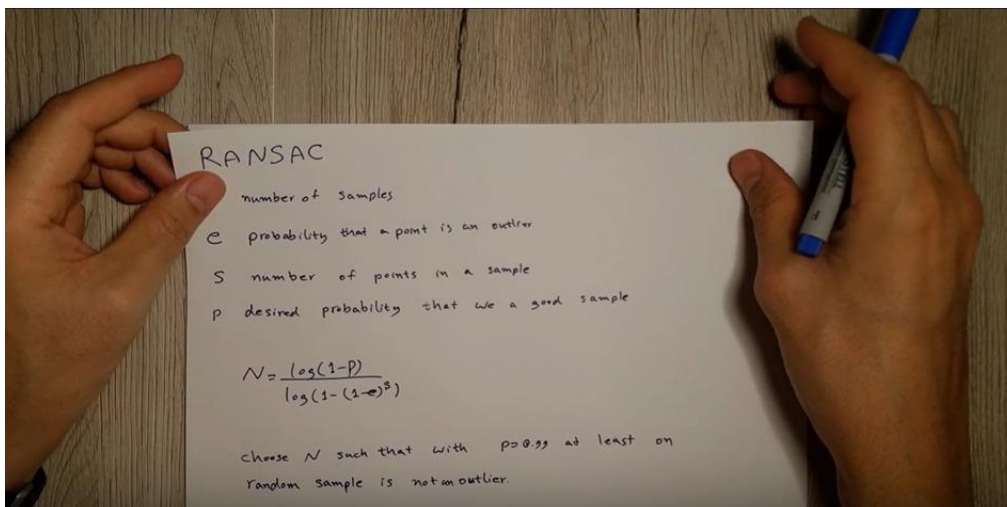
בגדול- ניקח סטים -החשב את הישר ביניהם (נוציא נקודות שהורסות את הישר לרוב הנקודות)(ונחשב כמה נקודות (אחרות) אכן קרובות לישר הזה (נמצאות בתוך הthreshold) אם threshold נמוך יותר ממה שמצאנו לפני- נחליף, נחזיר למעשה את הסט עם מספר הנקודות המקסימלי שנמצאות על הישר או הכי קרובות אליו, לפי ישר זה נחשב את החיבור 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 נקודות והמשוואות שלהם לפיהם נידע איך לחבר את התמונה, איך התמונה זזה מנקודות אלו לנתונה השניה.
2. המרחק של 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 N שלנו :



כאשר e הוא ההסתברות שהנקודה היא outlier- למעשה נחפש את ההסתברות שזה אכן בתוך הקו-inlier.

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 inlier.

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

הסבר ב udacity -

<https://classroom.udacity.com/courses/ud810/lessons/3189558841/concepts/3167938>

[9260923](#)

- **Camera calibration** – כיול (בדיקה כמה הכלי מדידה מדויק)מצלמה 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.