ראיה ממוחשבת

0510-6251

סמסטר א׳, תש״פ

מרצים:

פרופסור נחום קרייתי

פרופסור שי אבידן

סיכם:

צבי לדרר

תוכן

מציאת שפות (edge detection) 4

מציאת שפות 4

שיטות מבוססות גרדיאנטים 4

שיטות מבוססות לפלסיאן 6

מציאת קווים ומעגלים 8

Hough Transform 8

התמרת Duda & Hart 9

מציאת מעגל בעל רדיוס ידוע 11

מציאת מעגל בעל רדיוס לא ידוע 11

Thin Plate Spline Interpulation 12

Chain Code 14

Chain Code 14

Crack Code 14

משפט גרין - Green's Theorem 14

תמונות עומק 17

מערכת מצלמות קנונית 17

מערכת מצלמות לא קנונית 20

קואורדינטות הומוגניות 22

העתקת נקודה מהמרחב לתמונה 24

קליברציה 26

Structured Light 28

Photometric Stereo 30

תמונת סטריאו 38

Essential matrix and Fundamental matrix 43

שערוך disparity בעזרת Block Matchinng 46

שערוך disparity בעזרת dynamic programming (Viterbi ALG) 48

Markov Random Field (MRF) 49

שערוך disparity בעזרת Graph cut 52

אלגוריתם Dynamic Programming משופר 55

נקודות עניין – Local Invariant Features 56

Harris 56

Scale-Space 59

SIFT 61

חיפוש תמונה - "Video Google" 63

K-means 64

Viola-Jones 67

AdaBoost 67

בניית מרחב המאפיינים 70

Cascade of Rejectors 72

רשתות נוירונים 73
פרספטרון בודד... 73
בעיית ה-XOR 76
רשת נוירונים.. 78
Stochastic Gradient Descent.. 80

מציאת שפות (edge detection)

מציאת שפות

קיימות שתי שיטות מרכזיות למציאת שפות בתמונה:

1. שיטות מבוססות גרדיאנטים
2. שיטות מבוססות לפלסיאן

נסקור בקצרה את שתי השיטות.

שיטות מבוססות גרדיאנטים

שיטות אלו מבוססות על גרדיאנטים של התמונה. עבור תמונה רציפה (פונקציה רציפה), הגרדיאנט מוגדר כך:

$$\nabla f(x,y) = \frac{df(x,y)}{dx}\,\hat{x} + \frac{df(x,y)}{dy}\,\hat{y}$$

במקרה הבדיד, הנגזרת מוגדרת כך:

$$\frac{df[x,y]}{dx} = f[x+1,y] - f[x,y]$$
$$\frac{df[x,y]}{dy} = f[x,y+1] - f[x,y],$$

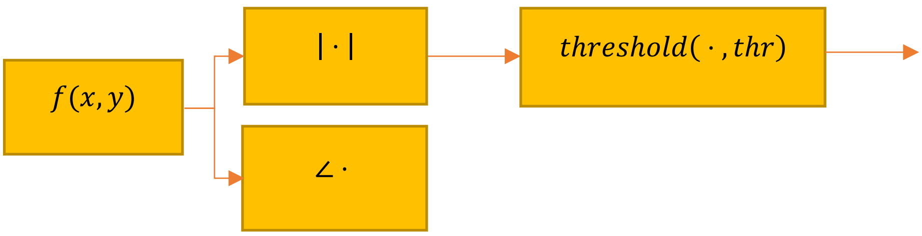
מתקבלת פונקציה וקטורית. **גודל** הפונקציה בכל נקודה:

$$|\nabla f(x,y)| = \sqrt{\left(\frac{df}{dx}\right)^2 + \left(\frac{df}{dy}\right)^2}$$

כיוון הפונקציה בכל נקודה:

$$\angle\nabla f(x,y) = \arctan\left(\frac{\frac{df}{dy}}{\frac{df}{dx}}\right)$$

להלן דוגמה לאלגוריתם מבוסס גרדיאנט:



תמונה 1: דוגמה ל-flow של אלגוריתם מבוסס גרדיאנט. מחושב הגרדיאט, הגודל והכיוון בכל נקודה, כאשר הגודל עובר סף. הפיקסלים שעברו את הסף נכללים בשפות.

מגודל הגרדיאנט ניתן להפיק את מיקום השפות ומכיוונו ניתן להפיק נתונים נוספים (למשל מתבצע בו שימוש באלגוריתם Canny).

גזירה של אות רגישה לרעש, ולכן קיימת בעיה של רעשים בשיטות מבוססות גרדיאנט. פתרון אפשרי לבעיה הוא הוספת LPF (פילטר מיצוע) לפני הגזירה על מנת להפחית רעשים. מכיוון שהן ה-LPF והן הגזירה הינן פעולות לינאריות, נהוג לאחד אותם לפילטר אחד אשר מבצע את שתי הפעולות יחד. נסקור מספר אלגוריתמים כאלה.

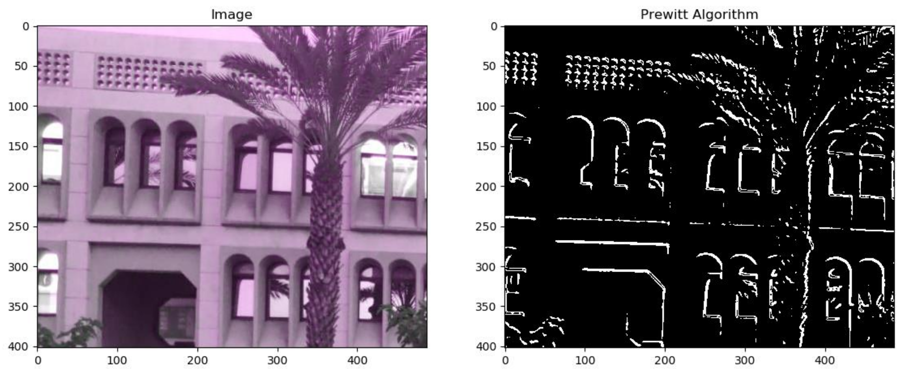
<u>אלגוריתם Prewitt</u> :

מתבצע שימוש בפילטר הבא :

$$\frac{df[x,y]}{dx} = \frac{1}{6} \cdot \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * f[x,y]$$

$$\frac{df[x,y]}{dy} = \frac{1}{6} \cdot \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * f[x,y]$$

לאחר פלטור, התמונה מועברת בסף (threshold), להלן דוגמה :



תמונה 2 : דוגמה לאלגוריתם Prewitt. התמונה המקורית בצד ימין, תמונת השפות כפי שנתגלו ע״י Prewitt בצד ימין.

<u>אלגוריתם Sobel</u> :

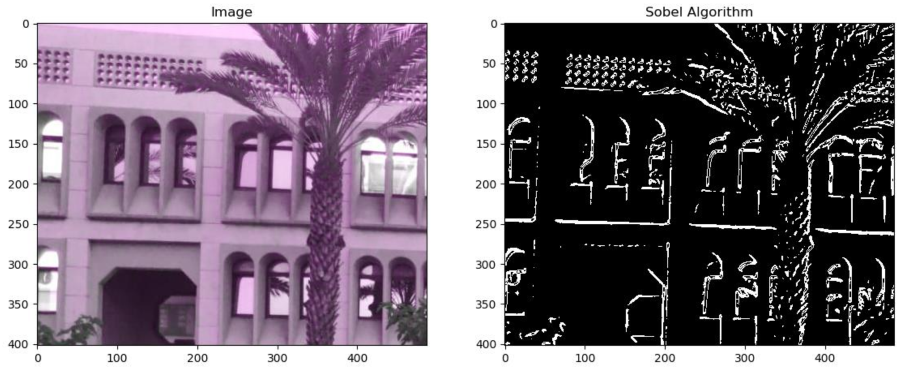
מתבצע שימוש בפילטרים הבאים :

$$\frac{df[x,y]}{dx} = \frac{1}{6} \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f[x,y]$$

$$\frac{df[x,y]}{dy} = \frac{1}{8} \cdot \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * f[x,y]$$

בשני האלגוריתמים, הנירמול מתבצע על מנת שתוצר האלגוריתם יישאר חסום בין ערכי התמונה המקוריים.

גם כאן, התמומה מועברת בסף.



תמונה 3 : דוגמה לאלגוריתם Sobel. התמונה המקורית בצד ימין, תמונת השפות כפי שנתגלו ע״י Sobel בצד ימין.

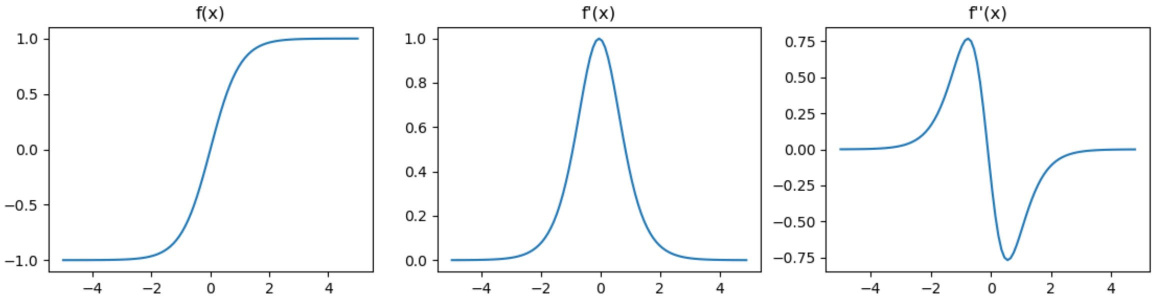
שיטות מבוססות לפלסיאן

לפלסיאן מוגדר בצורה הבאה :

$$\nabla^2 f(x,y) = \frac{d^2 f(x,y)}{dx^2} + \frac{d^2 f(x,y)}{dy^2}$$

בשונה מגרדיאנט, תוצר הלפלסיאן הינו סקלר.

באיור הבא מתוארת ״שפה חד מימדית״ ($f(x)$), נגזרת שלה, ונגזרת שניה.



תמונה 4 : פונקציית arctan(x), נגזרת ונגזרת שנייה שלה.

ניתן לראות כי כאשר גוזרים פעמיים שפה, או במקרה הרב-מימדי – כאשר מבצעים לפלסיאן, ניתן למצוא שפה ע״י איתור המקומות בהן הוא חוצה את ה-0.

ניתן להוכיח כי במקרה הבדיד ניתן לקרב לפלסיאן ע״י :

$$\nabla^2 f[x,y] = f[x,y+1] + f[x,y-1] + f[x+1,y] + f[x-1,y] - 4 \cdot f[x,y]$$

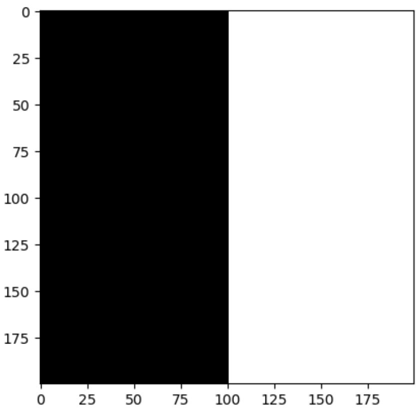
או :

$$\nabla^2 f[x,y] = k * f[x,y]$$
$$k = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

הבעיה העיקרית בשיטה זו היא שבדומה לשיטות מבוססות גרדיאנט, הרעש מוגבר (מכיוון שזו נגזרת שניה הוא מוגבר אף יותר). אלגוריתם Canny נועד להתגבר על הבעיה.

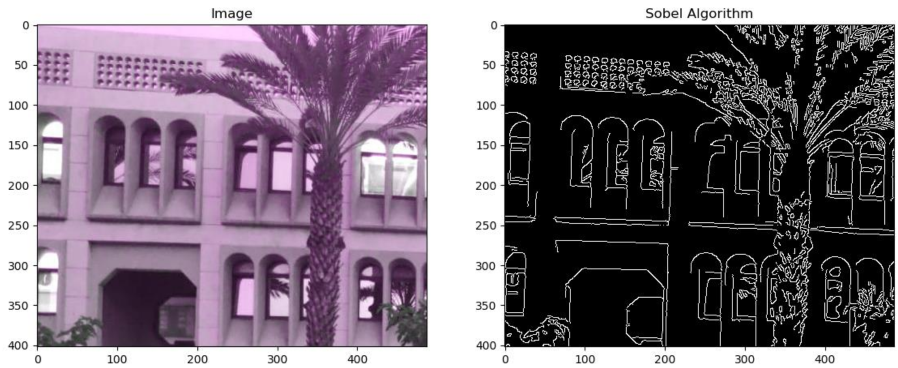
<u>אלגוריתם Canny :</u>

הלפלסיאן מורכב מחיבור של נגזרת בכיוון x ובכיוון y : $\frac{d^2f(x,y)}{dx^2}+\frac{d^2f(x,y)}{dy^2}$, אך במקרה של מציאת שפה, האיבר הרלוונטי הוא האיבר בכיוון הגרדיאט, ואילו האיבר המאונך לגרדיאט רק מוסיף רעש. למשל בתמונה הבאה האיבר $\frac{d^2f(x,y)}{dx^2}$ יתרום למציאת השפה ואילו $\frac{d^2f(x,y)}{dy^2}$ רק יוסיף רעש :



תמונה 5 : תמונת שפה בכיוון y. במקרה זה $\frac{df^2[x,y]}{dx^2}$ יניב ערך גבוה בגלל השפה, ואילו האיבר $\frac{df^2[x,y]}{dy^2}$ יוסיף רעש.

Canny מאפשר הורדת רעש משמעותית ע״י כך שהוא מוצא את כיוון הגרדיאנט בכל נקודה בתמונה ומחשב את הלפלסיאן רק בכיוון שלו.



מציאת קווים ומעגלים

Hough Transform

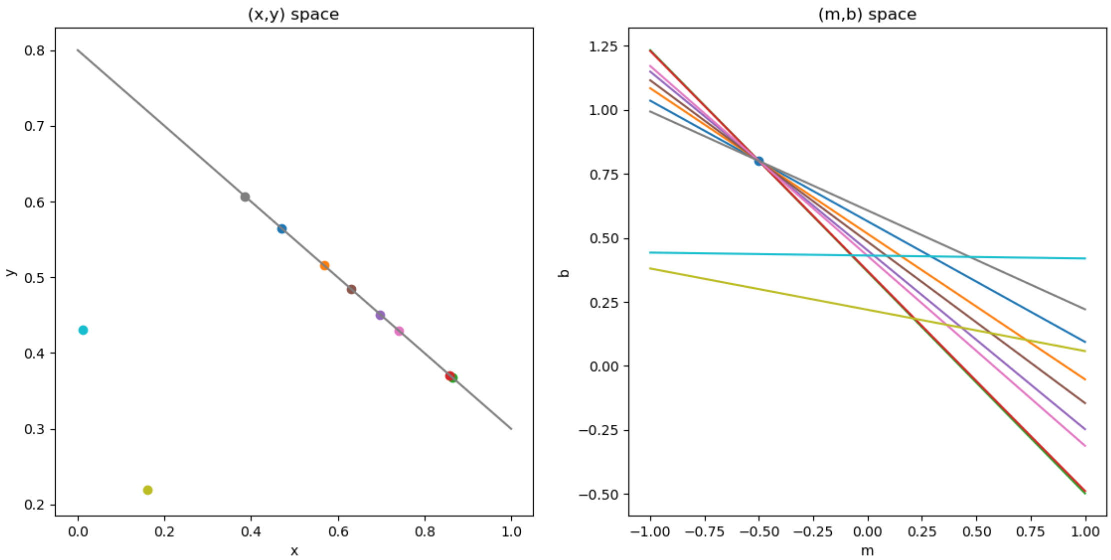
אלגוריתם ה-Hough Transform מיועד למציאת קווים ישרים בתמונה בינרית. האלגוריתם מבצע התמרה ממרחב x, y בו הנקודות נמצאות למרחב הפרמטרים m, b.

<u>תיאור פורמלי:</u>

נתון סט של נקודות: $P = \{p_i = (x_i, y_i) | i = 1 \dots N\}$. נדרש למצוא את הקו הישר עליו שוכנות המספר הגבוה ביותר של נקודות.

ניתן לייצג קו במרחב x, y ע"י המשוואה $y = mx + b$ כאשר הנקודות (x_i, y_i) מקיימות את המשוואה $y_i = mx_i + b$, המטרה היא למצוא את הפרמטרים m, b של הקו. ע"י שינוי המשוואה ניתן להגיע לביטוי: $b = -x_i m + y_i$, כאשר הנקודה (x_i, y_i) מתפקדת כפרמטרים של קו ישר במרחב החדש בו המשתנים הם m ו-b. המשוואה מייצגת אילוץ על m, b בהינתן קו העובר בנקודה (x_i, y_i).

כל נקודה במרחב המקורי מועתק לקו במרחב החדש. שתי נקודות (x_i, y_i) , (x_j, y_j) מועתקות לשני ישרים: $b = -x_i m + y_i$, $b = -x_j m + y_j$ כאשר נקודת החיתוך (m, b) של הישרים מייצגות פרמטרי קו ישר (m, b) במרחב המקורי אשר עובר דרך שתי הנקודות. ובאופן כללי, N נקודות הנמצאות על קו אחד יותמרו ל-N קוים הנפגשים בנקודה אחת.



תמונה 6: דוגמה ל-Hough transform. בצד שמאל נקודות הנמצאות על קו $y = -0.5x + 0.8$ ושתי נקודות שאינן עליו. בצד ימין הנקודות הועתקו לקווים, ניתן לראות כי כל הנקודות שהיו על הקו הועתקו לקווים הנפגשים בנקודה $(-0.5, 0.8)$.

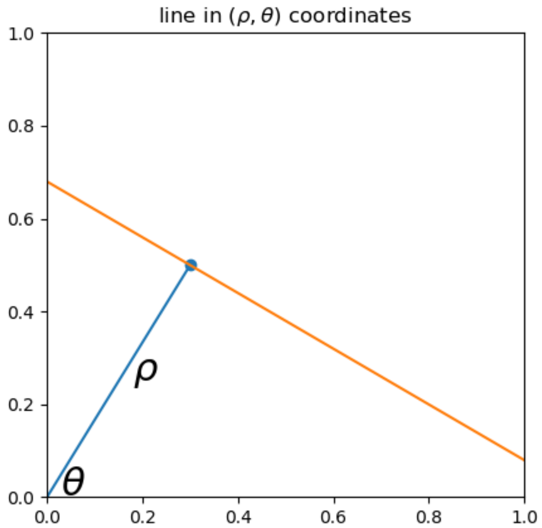
<u>תיאור האלגוריתם:</u>

1. העברת הנקודות לקוים במרחב הפרמטרים.
2. חלוקת המרחב החדש ל-grid.
3. מציאת המשבצת ב-grid בו עוברים מספר הקווים הגדול ביותר.
4. החזרת הפרמטרים (m, b).

בעיה עקרונית בהתמרת Hough היא שמרחב סופי (כמו תמונה) מועתק למרחב אין סופי. למשל ה-m של קו אנכי שואף לאינסוף ולכן הנקודות המרכיבות אותו ייפגשו באינסוף במרחב הפרמטרים. פיתרון אחד הוא להתמיר בנוסף את התמונה מסובבת ב-$90°$ ולבדוק את כל אחד ממרחבי הפרמטרים בטווחים $-1 < m < 1$. פיתרון נוסף הוא התמרת Duda & Hart.

התמרת Duda & Hart

מטרת ההתמרה היא להתגבר על בעיית העתקת מרחב סופי למרחב אינסופי. התמרה זו מתבססת על ייצוג קו ישר בעזרת פרמטרים (ρ, θ) במקום (m, b). כל קו ניתן להגדרה חד-חד ערכית ע"י הישר היוצא מראשית הצירים ומאונך לישר. זווית הקו מראשית הצירים תסומן ב-θ וגודלו ב-ρ.



תמונה 7 : ייצוג של קו (הישר הכתום) בעזרת הנורמל שלו .

הנקודות (x, y) אשר שוכנות על הישר מקיימות את המשוואה :

$$\rho = x\cos\theta + y\sin\theta$$

ניתן לבטא את הנקודה (x_i, y_i) כך :

$$x_i = \rho_i \cos\theta_i$$
$$y_i = \rho_i \sin\theta_i$$

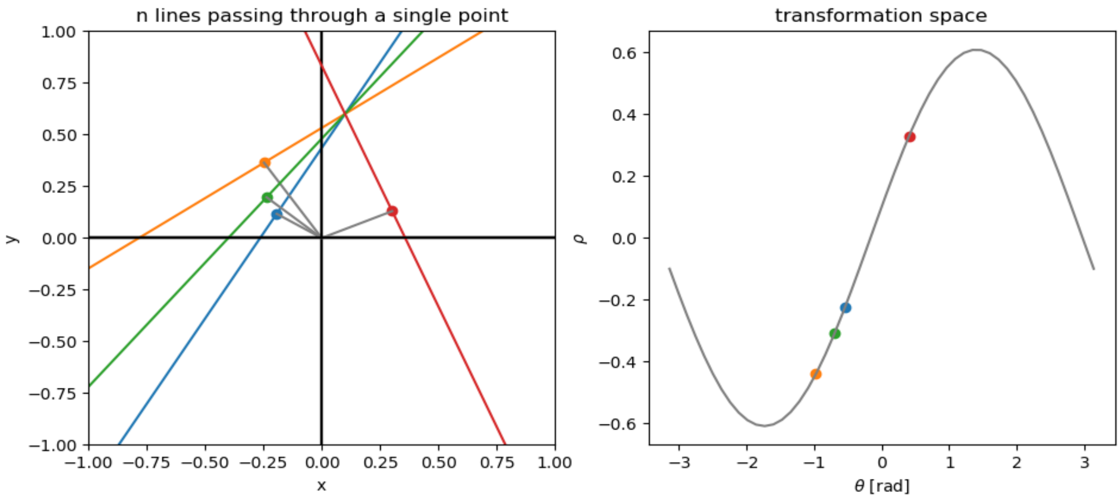
ולכן, עבור נקודה (x_i, y_i) אוסף הישרים העוברים דרך הנקודה מקיימים את האילוץ :

$$\begin{aligned}\rho &= x_i\cos\theta + y_i\sin\theta \\ &= \rho_i\cos\theta_i\cos\theta + \rho_i\sin\theta_i\sin\theta \\ &= \rho_i\cos(\theta - \theta_i)\end{aligned}$$

כאשר את הפרמטרים ρ_i, θ_i ניתן למצוא בעזרת :

$$\theta_i = \arctan\frac{y_i}{x_i}$$

$$\rho_i = \sqrt{x_i^2 + y_i^2}$$



תמונה 8: העתקה של קווים העוברים דרך נקודה אחת $(0.1,0.6)$ במישור x,y לדגימות של סינוסואיד במרחב ρ,θ.

ניתן לסכם את הטרנספורמציה כך:

מרחב x,y	**מרחב ρ,θ**
נקודה	סינוסואיד
קו	נקודה
נקודות קו-ליניאריות (על ישר)	סינוסואידים נחתכים (בנקודה (ρ,θ) המתאימה לפרמטרים m,n של הישר)
קווים שנפגשים בנקודה	נקודות על סינוסואיד

לאחר שמתבצעת הטרנספורמציה עבור כל נקודה בתמונה, מוקצית מטריצה אקומולציה ע"י האלגוריתם וכל תא שהסינוסואידים עוברים דרכו ייספר. בסופו של דבר, התא בו ייצבר הערך הגבוה ביותר, כלומר שעברו דרכו המספר הגבוה ביותר של סינוסואידים, יימצא במיקום הפרמטרים של הקו המשמעותי ביותר בתמונה המקורית.

פסאודו-קוד:

$$
\begin{aligned}
&For\ i = 1\ to\ N_{point}:\\
&\quad For\ \theta = 1\ to\ \pi\ step\ \Delta\theta:\\
&\quad\quad \rho = x_i \cos\theta + y_i \sin\theta\\
&\quad\quad round(\rho)\\
&\quad\quad A(\rho,\theta) += 1
\end{aligned}
$$

הערות:

- מכיוון שהלולאה עוברת עבור כל נקודה על אותם זוויות, אין צורך לחשב את הסינוס והקוסינוס בכל איטרציה, ניתן לבנות *lookup table* ובכל לחסוך חישובים מיותרים.
- <u>סיבוכיות האלגוריתם:</u>
  - בניית מטריצת האקומולציה (*voting*): $O(N_{points} \cdot N_\theta)$
  - חיפוש (*searching*): $O(N_\theta \cdot N_\rho)$
- ברוב המקרים מתקיים $N_\theta \gg N_{points}$ ולכן הגורם המשמעותי יהיה N_{points}. במקרה בו מספר הנקודות קטן, החלק היקר יהיה החיפוש.
- קיים *trade-off* בין רזולוציה גבוהה לנמוכה. רזולוציה נמוכה מידי תחזיר פרמטרים לא מדוייקים, ואילו רזולוציה גבוהה תגרום לסיבוכיות חישובית וכן לרגישות לרעש (מכיוון שהנקודות לא שוכנות על הקו בצורה מדוייקת, רזולוציה גבוהה מידי תגרום לנקודת המפגש לא להתרחש באותו "תא" במטריצת האקומולציה).
- אלגוריתמים אלו מכסים את כל מרחב הפתרונות, בכך שהם נותנים ציון לכל סט פרמטרים אפשרי. לכן לא קיימים כיום אלגוריתמים שיעלו על ביצועיהם בצורה משמעותית.

מציאת מעגל בעל רדיוס ידוע

בדומה למציאת קו ישר, גם מעגל ניתן למצוא בתמונה בעזרת התמרה למרחב פרמטרים. משוואת המעגל היא:

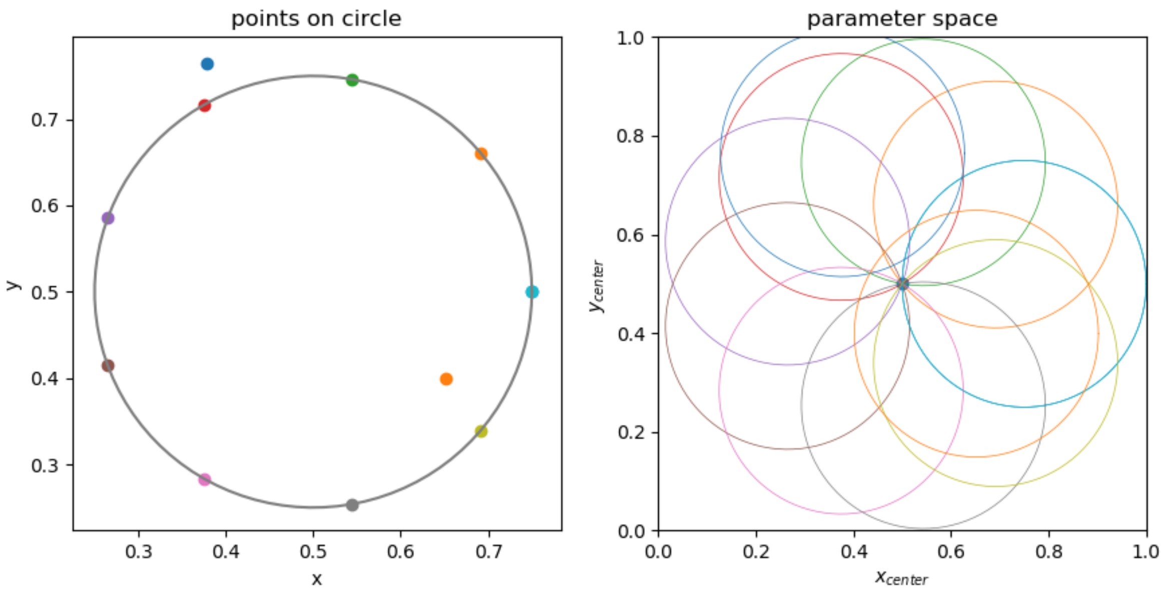
$$R^2 = (x - x_c)^2 + (y - y_c)^2$$

במעגל בעל רדיוס ידוע R הפרמטרים החסרים הם קואורדינטות המרכז (x_c, y_c). בהנחה כי נקודה (x_i, y_i) נמצאת על המעגל, הפרמטרים x_c, y_c חייבים לקיים את האילוץ:

$$R^2 = (x_i - x_c)^2 + (y_i - x_c)^2$$

כלומר, מרכז המעגל מוכרח להימצא ע״ג מעגל ברדיוס R מסביב לנקודה.

בדומה להתמרות הקודמות, גם כאן נעתיק כל נקודה למעגל במרחב הפרמטרים, ומרכז המעגל (x_c, y_c) יהיה בנקודת החיתוך של המעגלים במרחב הפרמטרים.



תמונה 9: העתקה של נקודות על מעגל ברדיוס ידוע במרחב x, y למרחב x_{center}, y_{center}. ניתן לראות כי המעגלים המועתקים מהנקודות שע״ג המעגל במרחב x, y נפגשים במרחב x_{center}, y_{center} ב-$(0.5,0.5)$ – מרכז המעגל.

מציאת מעגל בעל רדיוס לא ידוע

מציאת מעגל בעל רדיוס לא ידוע דומה לאלגוריתם הקודם, ההבדל הוא שמתווסף עוד פרמטר R למרחב הפרמטרים, כך שהמרחב יהיה בעל שלושה מימדים. במקרה זה כל נקודה מועתקת לאוסף של מעגלים בעלי רדיוסים שונים – קונוס. בדומה לאלגוריתמים הקודמים, ניצור מטריצת אקומולציה, אלא שהפעם היא תהיה טנזור תלת מימדי. נעבור על כל הנקודות, ונחשב את כל המרכזים האפשריים עבור כל רדיוס, ולבסוף נמצא את נקודת החיתוך.

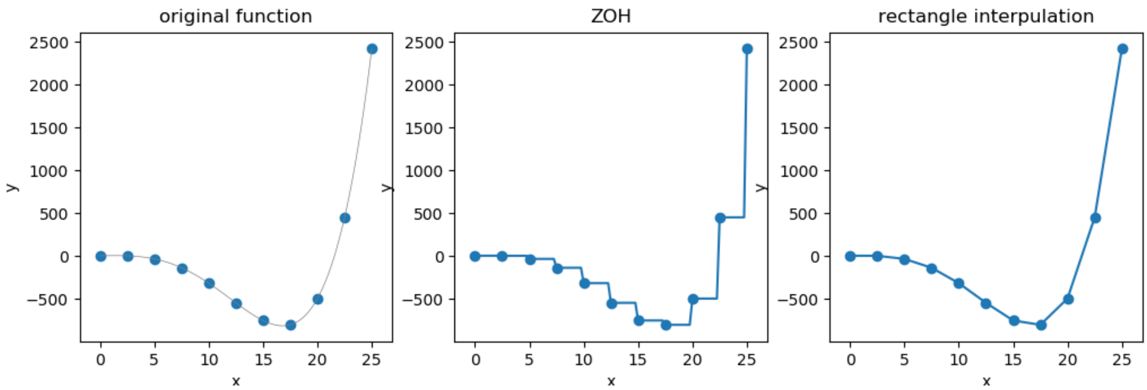
<u>סיבוכיות האלגוריתם:</u>

- בניית מטריצת האקומולציה (*voting*): $O(N_{points} \cdot N_{x_c} \cdot N_{y_c})$
- חיפוש (*searching*): $O(N_{x_c} \cdot N_{y_c} \cdot N_R)$

Thin Plate Spline Interpulation

במקרה של תמונה, בו האות דגום במרווחים קבועים קיימים אלגוריתמי אינטרפולציה סטנדרטיים, כגון:

- פונקציית *sinc*
- אינטרפולציה מסדר 0 (קונבולוציה עם מלבן)
- אינטרפולציה מסדר 1 (קונבולוציה עם משולש)



תמונה 10: אינטרפולציה חד מימדית. בצד שמאל – הפונקציה והדגימות שלה. במרכז – אינטרפולציה מסדר 0. בצד ימין – אינטרפולציה מסדר ראשון.

במקרים בהם האות אינו דגום בתדר קבוע נדרש אלגוריתם אחר.

<u>הבעיה:</u>

נתון סט נקודות $\{p_i\}_{i=1}^N = \{(x_i, y_i)\}_{i=1}^N$ ודגימות Z_i מתוך פונקציה לא ידועה כך ש-$Z_i = T(x_i, y_i)$.
נדרש למצוא את הפונקציה $T(x, y)$. עלנו לדרוש כי הפונקציה תהיה "חלקה". הקריטריון שנקבע הוא:

$$T(x, y) = argmin \iint T_{xx}^2 + 2T_{xy} + T_{yy}^2 dx\, dy$$

ניתן לפתור את המשוואה בצורה אנליטית ולקבל כי הפתרון מהצורה:

$$T(x, y) = \sum_{j=1}^{N} a_j E\left(\left\|(x, y) - (x_j, y_j)\right\|\right) + b_0 + b_1 x + b_2 y$$

תחת האילוצים:

(1) $Z(x_i, y_i) = T(x_i, y_i) = \sum_{j=1}^{N} a_j E\left(\left\|(x, y) - (x_j, y_j)\right\|\right) + b_0 + b_1 x_i + b_2 y_i = \boldsymbol{A}\vec{a} + \boldsymbol{B}\vec{b}$

(2) $\boldsymbol{B}^T \vec{a} = 0$

כאשר:

$$E(r) \triangleq r^2 \log r^2$$
$$\boldsymbol{A} = [E_{ij}]$$
$$\boldsymbol{B} = \begin{bmatrix} \vdots & \vdots & \vdots \\ 1 & x_i & y_i \\ \vdots & \vdots & \vdots \end{bmatrix}$$
$$\vec{a} = [a_1, a_2, \dots, a_N]$$

$$\vec{b} = [b_0, b_1, b_2]^T$$

אילוץ (1) מאלץ את הפונקציה לעבור דרך הדגימות, ואילוץ (2) מאלץ את הפרמטרים a_j לקיים:

1. $\sum a_j = 0$
2. $\sum a_j x_j = 0$
3. $\sum a_j y_j = 0$

נמצא את $\vec{a}, \vec{b}$. ע"פ אילוץ (1):

$$\vec{Z} = \boldsymbol{A}\vec{a} + \boldsymbol{B}\vec{b}$$
$$\Rightarrow \boldsymbol{A}\vec{a} = \vec{Z} - \boldsymbol{B}\vec{b}$$
$$\Rightarrow \vec{a} = \boldsymbol{A}^{-1}(\vec{Z} - \boldsymbol{B}\vec{b})$$

ע"פ אילוץ (2):

$$\boldsymbol{B}^T\vec{a} = \boldsymbol{B}^T\boldsymbol{A}^{-1}\left(\vec{Z} - \boldsymbol{B}\vec{b}\right) = 0$$
$$\Rightarrow \vec{b} = (\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{A}^{-1}\vec{Z}$$

הערות:

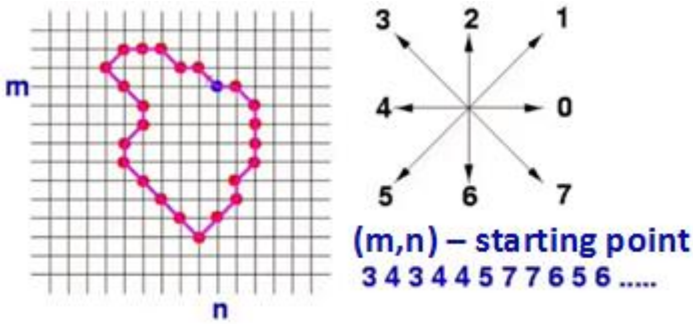
- במטלב קיימת פונקציה *tpaps.m* אשר מחשבת את המקדמים.
- באלגוריתם שהצגנו האילוץ הראשון גורם לכך שהפונקציה מוכרחת לעבור דרך הדגימות. במקרה בהם קיים רעש בדגימות הפיתרון עלול להשתבש ואף להגיע לחוסר יציבות. קיימים אלגוריתמים בהם "מרשים" לפונקציה שלא להיות על הדגימות וכך האלגוריתם יציב יותר.
- ניתן לכתוב את המשוואה כך: $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B} & \boldsymbol{0} \end{pmatrix}\begin{pmatrix} \vec{a} \\ \vec{b} \end{pmatrix} = \begin{pmatrix} \vec{\boldsymbol{Z}} \\ \vec{0} \end{pmatrix}$ ולהפוך את המטריצה השמאלית, אך הדבר יעלה בסיבוכיות גדולה יותר (סיבוכיות של הפיכת מטריצה היא $O(n^3)$).
- ניתן לחלק את הביטוי $T(x,y) = \sum_{j=1}^{N} a_j E\left(\left\|x - x_j, y - y_j\right\|\right) + b_0 + b_1 x + b_2 y$ לשני רכיבים. הביטוי $b_0 + b_1 x + b_2 y$ מייצג משוואת המשטח הישר "הקרוב ביותר" לדגימות, ואילו הביטוי $\sum_{j=1}^{N} a_j E\left(\left\|x - x_j, y - y_j\right\|\right)$ מייצג את ה"תיקונים" שיש להוסיף למשטח על מנת שיעבור דרך הדגימות.

Chain Code

נניח כי נתון מסלול ע"ג תמונה, קיימות מספר שיטות לקידוד המסלול. נתאר את *chain code* ו- *crack code*.

Chain Code

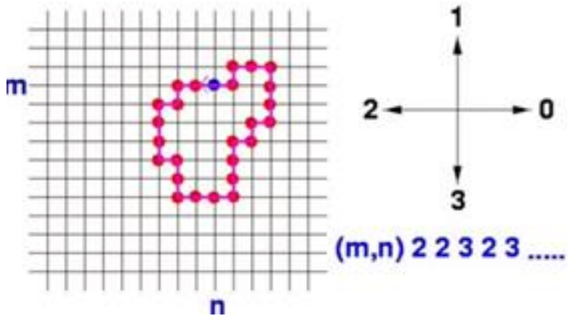
בשיטה זו מתבצע קידוד של 8 כיוונים – מעבר מפיקסל לכל אחד מ-8 הפיקסלים הסמוכים לו. נבחרת נקודת התחלה שרירותית ע"ג המסלול (הנקודה הכחולה) וממנה מתקדמים צעד אחר צעד, ע"פ הקוד המוגדר, עד לחזרה לנקודת ההתחלה (בהנחה שהצורה סגורה).



תמונה 11 : דוגמה לייצוג מסלול ע"י Chain cods. בצד ימין מוצג הקוד, נקודת ההתחלה של המסלול וקידוד המסלול, ובצד שמאל מוצג המסלול ע"ג התמונה.

Crack Code

קוד זה דומה ל-*chain code*, וההבדל ביניהם הוא ש-*Crack code* מקודד את המסלול ע"י 4 כיווני תנועה, ולא שמונה.



תמונה 12 : בדומה לתמונה הקודמת, בתמונה זו מוצג ייצוג Crack code.

משפט גרין - Green's Theorem

משפט גרין הוא משפט באנליזה מתמטית המגדיר קשר בין אינטגרל קווי של פונקציה על עקום סגור ופשוט לבין האינטגרל המשטחי על השטח החסום על ידי העקום (מקרה פרטי דו מימדי של משפט סטוקס).

המשפט :

תהי C מסילה פשוטה סגורה וגזירה למקוטעין החוסמת שטח ב- $\mathbb{R}^2$, ונסמן R את השטח החסום על ידי המסילה C. אם $f(x,y), g(x,y)$ פונקציות בעלות נגזרות חלקיות רציפות עד סדר ראשון בתוך סביבה המכילה את R, אזי :

$$\oint_C (fdx + gdy) = \iint_R \left(\frac{\partial g}{\partial x} - \frac{\partial f}{\partial y}\right) dxdy$$

כאשר הביטוי משמאל מגדיר אינטגרל קווי על עקום סגור ומימין מבוטא אינטגרל משטחי בתחום הסגור D.

דוגמה שימושית למשפט גרין כאשר צורה מתוארת ברשת דיסקרטית על ידי Chain Code, ניתן למצוא את השטח מבלי לחשב את האינטגרל בצורה מפורשת:

נניח: $f = 0 \Leftarrow$
$$\oint_C (gdy) = \iint_R \left(\frac{\partial g}{\partial x}\right) dxdy$$

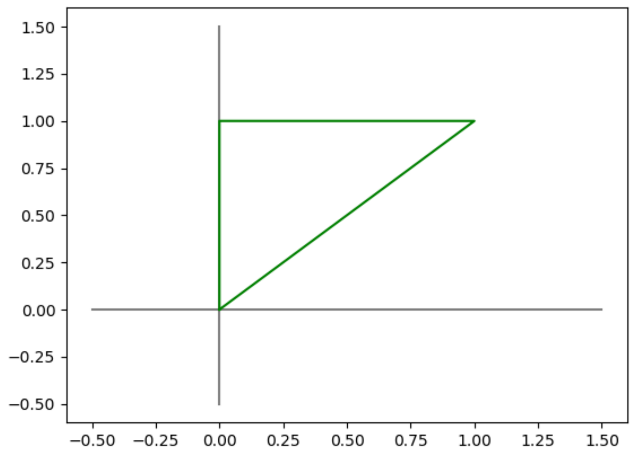
ונניח גם: $g = x \Leftarrow$
$$\oint_C (xdy) = \iint_R \left(\frac{\partial x}{\partial x}\right) dxdy$$

$$\oint_C xdy = \iint_R 1 \cdot dxdy$$

בצורה זו ניתן לחשב את השטח החסום ע"י אינטגרל מסלולי על המסלול החוסם. בצורה זו ניתן בקלות לחשב את השטח, המומנט, הסנטרואיד ועוד מאפיינים נוספים, על ידי שימוש במשפט גרין על מסלול המיוצג בייצוג Chain Code.

דוגמה למקרה הרציף:

נניח כי אנו מעוניינים לחשב את השטח החסום בין הנקודות: $(0,0), (1,1), (0,1)$.



תמונה 13: בדומה לתמונה הקודמת, בתמונה זו מוצג ייצוג Crack code.

ע"פ המשוואה שגזרנו ממשפט גרין ניתן לומר כי:

$$S = \iint_R 1 \cdot dxdy = \oint_C x\, dy = \int_{y=0}^{y=1} y\, dy + \int_{y=1}^{y=1} x\, dy + \int_{y=1}^{y=0} 0\, dy = \int_{y=0}^{y=1} y\, dy$$
$$= \left[\frac{1}{2} y^2\right]_{y=0}^{1} = \frac{1}{2}$$

דוגמה ליישום ב-Crack code:

ניתן בקלות למדוד שטח של צורה המיוצגת ב-crack code. נשים לב לעובדה כי עבור crack code ישנם שני מקרים:

1. ההתקדמות היא בציר x ולכן הביטוי $\oint_C x\, dy$ מתאפס ($dy = 0$).
2. ההתקדמות היא יחידה אחת בציר ה-y, במקרה זה x קבוע ובכל צעד, האינטגרל גדל/קטן ב-x (ע"פ כיוון ההתקדמות ב-y).

ניתן להגדיר אלגוריתם:

$get\ start\ point\ (x, y), path_{code}$
$sum = 0$

$\boldsymbol{For}\ step\ \boldsymbol{in}\ path_{code}$:
    $\boldsymbol{if}\ step == 0$:
        $x += 1$
    $\boldsymbol{elif}\ step == 1$:
        $sum += x$
    $\boldsymbol{elif}\ step == 2$:
        $y -= 1$
    $\boldsymbol{elif}\ step == 3$:
        $sum -= x$
$\boldsymbol{return}\ sum$

תמונות עומק

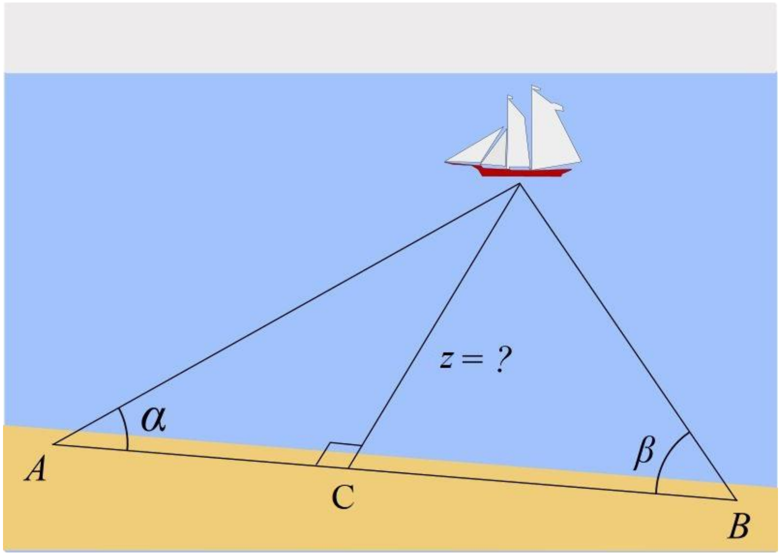
מערכת מצלמות קנונית

בהנחה כי ידועות לנו מיקומם של שתי מצלמות, וקיים עצם בזווית ידועה מכל מצלמה, ניתן למצוא את המרחק של העצם משתי המצלמות.

מערכת קנונית (*canonical stereo system*) הינה מערכת מצלמות המקיימת את התנאים הבאים:

- ציר אופטי של המצלמות מקביל
- מישורי שתי התמונות מוכלות במישור אחד (המישורים מתלכדים)
- *Pinhole camera*

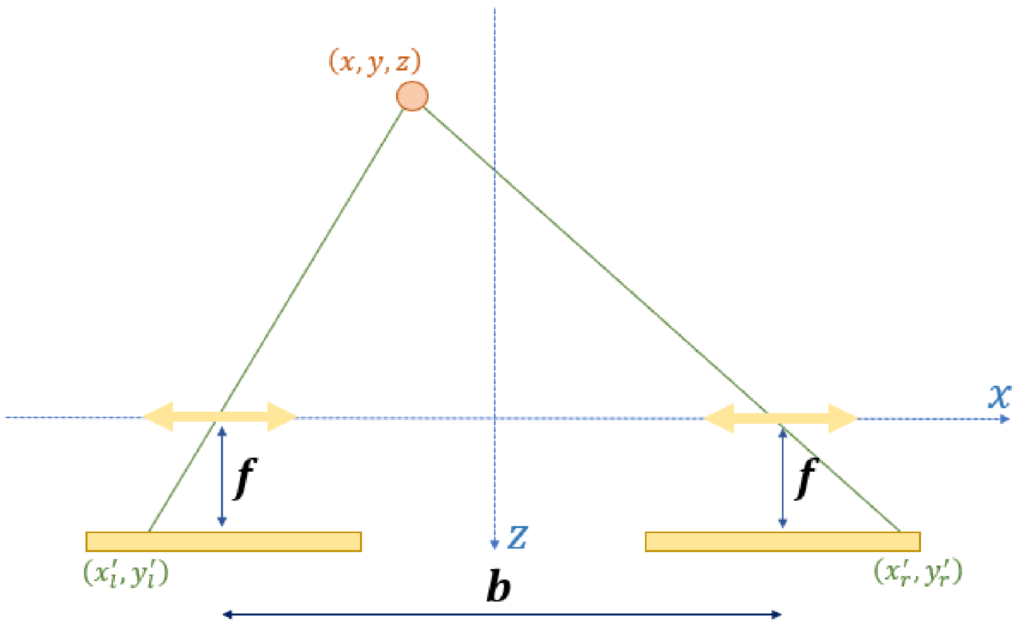
תחת הנחות אלה ניתן להוכיח בצורה פשוטה את הטענה שלעיל. בהינתן שתי מצלמות המונחות בנקודות A, B ועצם במרחק Z, כפי שמצוייר:



תמונה 14: מבט על עצם משתי נקודות שונות אשר המרחק ביניהם והזוויות לעצם ידועים. בהינתן $|\overrightarrow{AB}|, \alpha, \beta$ ניתן למצוא את המרחק z.

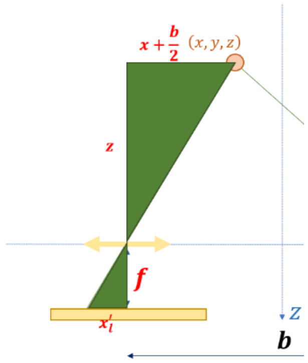
$$b = |\overrightarrow{AB}| = |\overrightarrow{AC}| + |\overrightarrow{CB}| = z\tan^{-1}\alpha + z\tan^{-1}\beta = z\left(\frac{\cos\alpha}{\sin\alpha} + \frac{\cos\beta}{\sin\beta}\right)$$
$$= z\frac{\cos\alpha\sin\beta + \cos\beta\sin\alpha}{\sin\alpha\cos\alpha} = z\frac{\sin(\alpha+\beta)}{\sin\alpha\sin\beta}$$
$$\Rightarrow z = b\frac{\sin\alpha\sin\beta}{\sin(\alpha+\beta)}$$

ע״פ הפיתוח, בהנחה כי ידוע לנו b ושתי הזווית α, β ניתן למצוא את המרחק Z מהעצם.



תמונה 15: שתי מצלמות l, r, המרחק בין מרכזיהם מסומן ב-b המרחק בין מישורי התמונות וה-pinhole מסומן ב-f. העצם בנקודה (x, y, z) נצפה במצלמה הימנית והשמאלית בנקודות (x'_r, y'_r) ו-(x'_l, y'_l) בהתאמה.

ע״י דמיון משולשים:



$$(1) \quad \frac{x'_l}{f} = \frac{x+\frac{b}{2}}{z}$$

$$(2) \quad \frac{x'_r}{f} = \frac{x-\frac{b}{2}}{z}$$

$$(3) \quad \frac{y'_l}{f} = \frac{y'_r}{f} = \frac{y}{z}$$

$$(1) + (2) \quad \frac{x'_l - x'_r}{f} = \frac{b}{z} \Rightarrow z = \frac{bf}{x_l - x_r} = -\frac{bf}{d}$$

אחת האתגרים בבניית תמונת סטריאו היא התאמת נקודות מתמונה אחת לשניה. הנחה מקילה שניתן להניח במקרה של מערכת מצלמות קנונית היא שנקודות תואמות יהיו בעלי ערך y זהה (ניתן להיווכח ממשוואה (3)).

הערות:

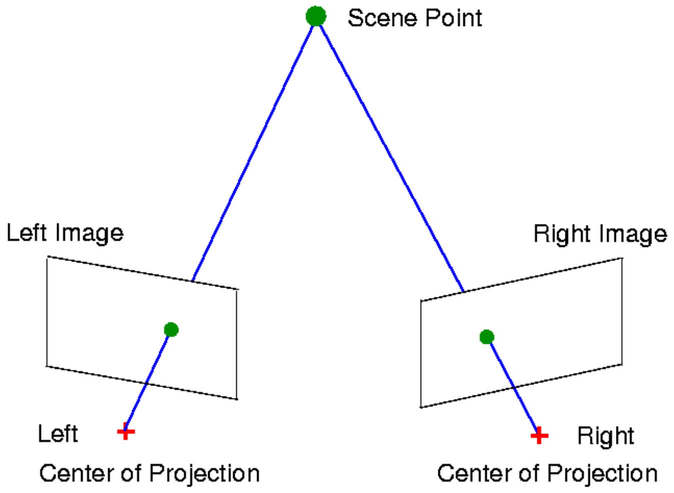
- במשוואה הנ״ל, d נקרא $disparity$ והוא המרחק ע״ג התמונה בין העצם כפי שנצפה במצלמה אחת לבין אותו העצם במצלמה השניה.
- קיים *trade-off* לגבי המרחק בין המצלמות. ככל שהמרחק b גדול יותר, כך ניתן למדוד z גדול יותר (f קבוע, ו-d חסום ע״י הרזולוציה של המצלמה). מצד שני b גדול מידי יכול לגרום להסתרות ולמחסור בחפיפה בין התמונות.
- אתגרים בבניית פנורמה:
  - דיסטורציה של העדשה (הנחת *pinhole* לא מתקיימת).
  - הסתרות – עצם שמופיע במצלמה אחת לא יופיע בהכרח במצלמה השנייה כתוצאה מהסתרות.
  - מציאת נקודות תואמות בין התמונות.
- נקודות תואמות בין התמונות (x'_r, y'_r) (x'_l, y'_l) נקראות ***conjugate points***.

תמונה 16 : דוגמה להסתרה. בצד ימין – תמונת סטריאו, כאשר ההסתרות מסומנות באדום, בצד שמאל – הדגמה למיקומי הסתרות.

מערכת מצלמות לא קנונית

עד עתה הנחנו כי שתי המצלמות נמצאות על אותו מישור וכי הן מסתכלות לאותו כיוון (כיווני הסתכלות מקבילים). אך אין הנחה זו הכרחית. לצורך ההכללה יש להציג את הגיאומטריה האפיפולרית.

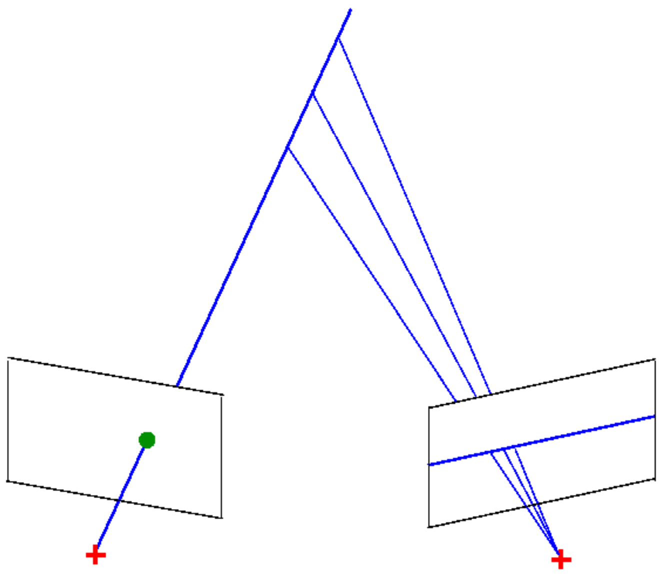
נניח שתי מצלמות לא קנוניות:



תמונה 17 : שני מישורי תמונה במערכת לא קנונית.

נתונה נקודה במרחב, וכן נתונים ה-*center of projection*. נהוג לשקף את מישור התמונות לצד השני של ה-*center of projection*, ניתן להוכיח בקלות כי נקודת החיתוך של הישר המחבר בין נקודת התצפית וה-*center of projection* עם מישור התמונה זהה לנקודת החיתוך שלו עם המישור המשוקף – הנקודות הירוקות (עד כדי שיקוף התמונה סביב שני הצירים).

עצם במרחק לא ידוע אשר מופיע על פיקסל ספציפי יכול להיות במרחב בכל נקודה ע"ג הקרן היוצאת מה-*center of projection* לכיוון הנקודה. מכאן ניתן לראות כי הנקודה יכולה להיראות על גבי התמונה השניה על קו (ולא בכל התמונה), כפי שהיה במערכת הקנונית. ההבדל הוא שהקו אינו בהכרח מקביל לציר x.

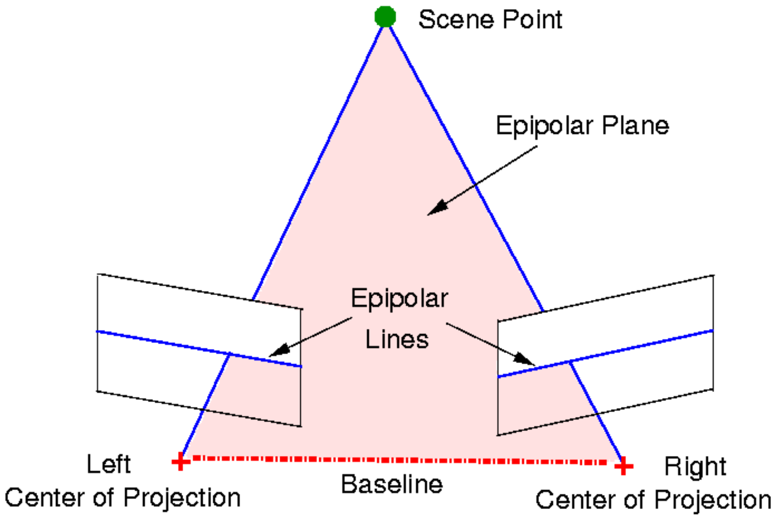


תמונה 18 : במצלמה הימנית נצפה עצם בפיקסל הירוק, העצם יכול להיות בכל נקודה על פני הישר המחבר בין ה-center of projection והפיקסל. נקודה זו יכולה "ליפול" ע"ג כל נקודה שעל הקו האפיפולרי שבתמונה הימנית.

ע"מ למצוא את הקו עליו הנקודה תהיה יש להגדיר מספר הגדרות:

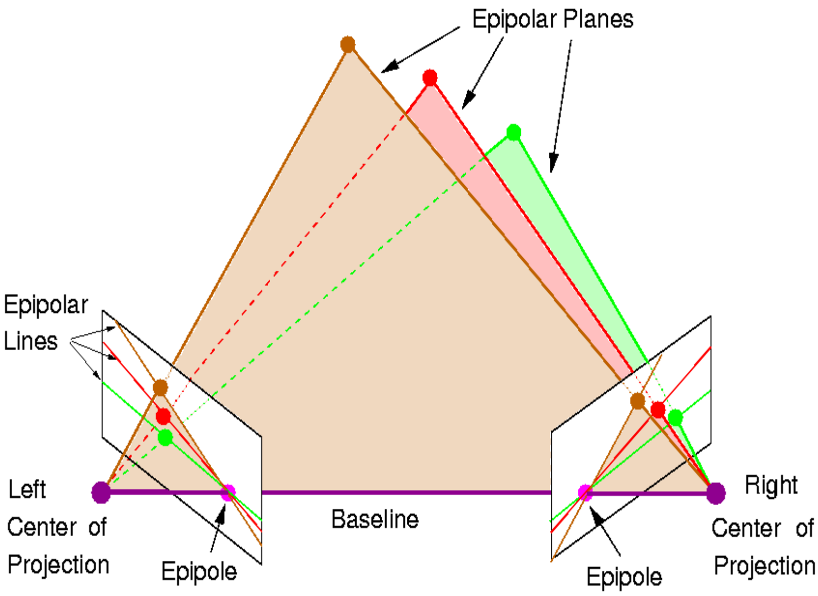
- ה-***baseline*** הוא הקו המחבר את ה-*center of projection* של שתי התמונות.
- **המישור האפיפולרי (*Epipolar plane*)** הוא המישור הנוצר ע"י ה-*baseline* ונקודת התצפית.

- **קווים אפיפולריים (*epipolar lines*)** הם הקווים בתמונה אליהם מוטל הישר המחבר את ה-*baseline* של המצלמה השניה ונקודת התצפית.



תמונה 19 : המישור האפיפולרי והקווים האפיפולריים.

ניתן להוכיח כי כל הקווים האפיפולריים (המוטלים מכל נקודה במרחב) נפגשים בנקודה אחת ע״ג כל אחת מהתמונות, נקודה זו היא ה-***epipole***.



תמונה 20 : קווים אפיפוליים ממספר נקודות במרחב. ניתן לראות כי כל הקווים נפגשים ב-epiploe.

קואורדינטות הומוגניות

קליברציה מאפשרת להעתיק נקודה ממערכת צירים כלשהי במרחב ("נקודה בעולם") למיקומה על גבי התמונה. לשם כך נדרש להציג את המושג **קואורדינטות הומוגניות**.

***Translation* - הזזה**

נניח כי קיים וקטור $\vec{x} = (x, y, z)^T$. אם נרצה לבצע עליו פעולת הזזה, כלומר להעתיקו למערכת קואורדינטות בה $\vec{x}' = (x + \Delta x, y + \Delta y, z + \Delta z)^T$, לא ניתן לתאר את הפעולה כמכפלה של מטריצה בוקטור מהצורה $\boldsymbol{A}\vec{x}^T = \vec{x}'^T$. ניתן להוכיח זאת ע"י המקרה בו $\vec{x} = (0,0,0)^T$, במקרה כזה $\overrightarrow{\boldsymbol{A}x^T} \equiv \vec{0}$ ולא ניתן לבצע פעולת הזזה. ע"מ להתגבר על הבעיה ניתן להמיר את הוקטור $\vec{x}$ לקורדינטות הומוגניות: $\vec{x} = (x, y, z, 1)^T$. בצורה כזאת ניתן לבטא את הפעולה כך:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

במקרה זה, מטריצת המעבר היא:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

והמטריצה ההפוכה:

$$\boldsymbol{T^{-1}} = \begin{pmatrix} 1 & 0 & 0 & -\Delta x \\ 0 & 1 & 0 & -\Delta y \\ 0 & 0 & 1 & -\Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling

ניתן לבטא את פעולת ה-scaling ביחס לראשית הצירים בעזרת המטריצה הבאה:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x x \\ S_y y \\ S_z z \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

במקרה זה, מטריצת המעבר היא:

$$\boldsymbol{S} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

והמטריצה ההפוכה:

$$\boldsymbol{S^{-1}} = \begin{pmatrix} S_x^{-1} & 0 & 0 & 0 \\ 0 & S_y^{-1} & 0 & 0 \\ 0 & 0 & S_z^{-1} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

***Rotation* – סיבוב**

פעולת הסיבוב מוגדרת עם כיוון השעון של ציר כלשהו כאשר המבט מהצד החיובי לעבר ראשית הצירים. ניתן לסובב את הווקטורים מסביב לכל אחד משלושת הצירים. מטריצת הסיבוב מוגדרת כך:

$$\boldsymbol{R}_{\theta,z}=\begin{pmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

המטריצה ההפוכה מתקבלת ע"י הצבת $-\theta$ בזווית:

$$\boldsymbol{R}_{\theta,z}^{-1}=\begin{pmatrix}\cos\theta & \sin\theta & 0 & 0\\ -\sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

באותה צורה ניתן להגדיר עבור שאר הצירים:

$$\boldsymbol{R}_{\theta,x}=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -\sin\theta & 0\\ 0 & \sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$$\boldsymbol{R}_{\theta,y}=\begin{pmatrix}\cos\theta & 0 & -\sin\theta & 0\\ 0 & 1 & 0 & 0\\ \sin\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

הערות:

- ניתן לסובב בכל כיוון ע"י קומבינציה של סיבובים סביב כל אחד מהצירים.
- סדר הפעולות חשוב (סיבוב סביב ציר y ואח"כ סביב x יתן תוצאה שונה מאשר סיבוב סביב x ואח"כ סביב y).

ניתן לבטא פעולה של סיבוב והזזה יחד ע"י מטריצה יחידה:

$$\begin{pmatrix}\boldsymbol{R}_{3\times3} & \vec{T}_{3\times1}\\ \vec{0}_{1\times3} & 1\end{pmatrix}$$

המרת קואורדינטות

ניתן להמיר חזרה קואורדינטות מהייצוג ההומוגני לייצוג הסטנדרטי ע"י חלוקה של כל אחד מהאיברים באיבר האחרון של הוקטור:

$$\begin{pmatrix}U\\ V\\ W\\ k\end{pmatrix}\to\begin{pmatrix}U/k\\ V/k\\ W/k\end{pmatrix}$$

חשוב להדגיש כי לאיברים U,V,W אין משמעות במרחב המקורי ללא החלוקה ב-k.

ניתן באותה מידה להכיל את כל הכללים על קואורדינטות הומוגניות במרחב דו מימדי.

$$\begin{pmatrix}x\\ y\end{pmatrix}\to\begin{pmatrix}x\\ y\\ 1\end{pmatrix}$$

$$\begin{pmatrix}U\\ V\\ k\end{pmatrix}\to\begin{pmatrix}U/k\\ V/k\end{pmatrix}$$

מהכללים שהצגנו נובע כי וקטור הומוגני "אדיש" לכפל בסקלר, וניתן לייצג כל וקטור (x,y,z) כ-(kx,ky,kz,k). כאשר האיבר האחרון הינו 0 – הנקודה נמצאת במרחב המקורי באיסוף.

"Actually, infinity is a very good place to be"

העתקת נקודה מהמרחב לתמונה

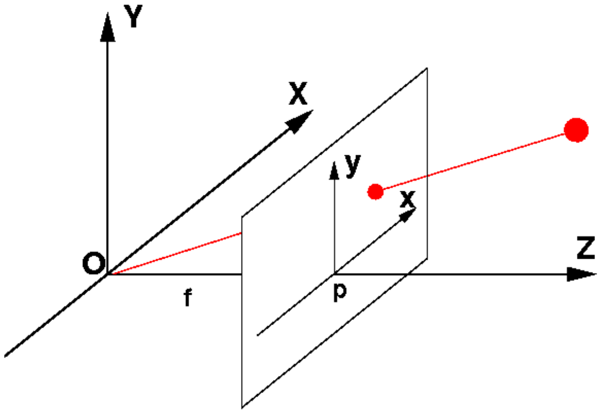
העתקה למישור התמונה במערכת קנונית

כזכור, במערכת קנונית מישור (x,y) של התמונה מקביל למישור (X,Y) במרחב, וציר Z חוצה את מישור התמונה במרכזה. במקרה זה, נרצה למצוא את ההעתקה של נקודה (X_c,Y_c,Z_c) לנקודה (x_c,y_c).
מדמיות משולשים ניתן להסיק:

$$\frac{x_c}{f}=\frac{X_c}{Z_c}\Rightarrow x_c=f\frac{X_c}{Z_c}$$
$$\frac{y_c}{f}=\frac{Y_c}{Z_c}$$

ובקואורדינטות הומוגניות:

$$\begin{pmatrix}x_c\\y_c\\f\end{pmatrix}=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}X_c\\Y_c\\Z_c\\1\end{pmatrix}$$
$$\Rightarrow\frac{x_c}{f}=\frac{X_c}{Z_c}$$



תמונה 21: נקודה (X,Y,Z) במרחב מועתקת לנקודה (x,y) במישור התמונה. בהתאם לקונבנציה, מישור התמונה משוקף ביחס למישור X,Y.

הערה:

- האינדקס c ב-x_c מבטא מערכת קנונית ($canonical$).

מעבר מקואורדינטות (x,y) בציר התמונה לפיקסלים (u,v)

המעבר מנקודה (x,y) לפיקסל (u,v) מתקבל ע"י המשוואות הבאות:

$$u=u_0+k_ux_c$$
$$v=v_o+k_vy_c$$

כאשר (u_0,v_0) הוא ראשית הצירים ביחידות של פיקסלים, ו-(k_u,k_v) הוא יחידת ההמרה מיחידת אורך לפיקסל (למשל $\frac{pixel}{mm}$). ניתן לבטא את הטרנספומציה כמכפלת מטריצה:

$$\vec{x}_i = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \cong f\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_c \\ y_c \\ f \end{pmatrix}$$

הערה:

- האינדקס i ב-x_i מבטא תמונה ($image$).

מעבר מקואורדינטות "העולם" לקואורדינטות קנוניות

ראינו כיצד לעבור מנקודה במרחב עם מערכת קואודינטות קנוניות למרחב התמונה, וכיצד לעבור מקואורדינטות x, y במישור התמונה ליחידות של פיקסלים u, v. נותר המעבר מנקודה בקואורדינטות במערכת צירים כלשהי של "העולם" לאותה נקודה במערכת הקנונית. לאחר שנוכל לעשות זאת – נוכל לדעת עבור כל נקודה "בעולם" בכל מערכת צירים היכן היא מועתקת על התמונה (באיזה פיקסל).

מעבר כזה הוא פשוט – מספיק סיבוב של מערכת הצירים והזזה כך שראשית הצירים תעבור לראשית הצירים של המערכת הקנונית (הזזה) וציר z יהיה מקביל לציר האופטי של המצלמה (סיבוב). ניתן לבטא את הטרנספורמציה בעזרת מטריצה:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{R}_{3\times 3} & \vec{T}_{3\times 1} \\ \vec{0}_{1\times 3} & 1 \end{pmatrix}\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

מעבר מנקודה "בעולם" לפיקסל

אנו יודעים לעבור מנקודה במרחב במערכת קואורדינטות כלשהי למערכת הקנונית, משם למישור התמונה ומשם לביטוי בפיקסלים:

$$\vec{x_i} = f\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_c \\ y_c \\ f \end{pmatrix} = \begin{pmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} =$$

$$\begin{pmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} \boldsymbol{R}_{3\times 3} & \vec{T}_{3\times 1} \\ \vec{0}_{1\times 3} & 1 \end{pmatrix}\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \boldsymbol{CP}\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

כאשר:

$$\boldsymbol{C} = \begin{pmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

מטריצה $\boldsymbol{C}$ נקראת **המטריצה האינטרינזית**, מכיוון שהיא טומנת בחובה את הפרמטרים האינטרינזיים של המצלמה, כגון מרחק מישור התמונה מה-*pinhole* (f) וגודל הפיקסל (k_u, k_v). מטריצה $\left(\boldsymbol{R}_{3\times 3} \mid \vec{T}_{3\times 1}\right)$ נקראת **המטריצה האקסטרינזית**, מכיוון שהיא מייצגת את הפרמטרים של "העולם" – מערכת הצירים של המרחב ביחס למערכת הקנונית.

מטריצה $\boldsymbol{P}$ נקראת מטריצה ההטלה – *projection matrix* – ומוגדרת כ:

$$\boldsymbol{P} = \boldsymbol{C}\cdot\left(\boldsymbol{R}_{3\times 3} \mid \vec{T}_{3\times 1}\right) = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix}$$

הערות:

- מכיוון שהקואורדינטות ההומוגניות אדישות לכפל בסקלר יש ניתן להכפיל את המטריצה בסלקר ללא שינוי, ולכן יורדת דרגת חופש אחת ולמטריצה $\boldsymbol{P}$ 11 דרגות חופש (במקום 12).
- במטריצה $\boldsymbol{C}$ ניתן להוסיף איבר ב-c_{12} – skew – אשר מגדיר עיוות של התמונה, למשל במקרה בו מישור התמונה אינו מלבני אלא מקבילי, אך תופעה זו אינה נפוצה.

קליברציה

בהינתן נקודה במרחב $\overrightarrow{x_w} = (X_w, Y_w, Z_w)$ והנקודה (u, v) המתאימה ע"ג התמונה, ע"פ המשוואה שלעיל, ההעתקה מוגדרת:

$$u = \frac{p_{11}X_w + p_{12}Y_w + p_{13}Z_w + p_{14}}{p_{31}X_w + p_{32}Y_w + p_{33}Z_w + p_{34}}$$
$$v = \frac{p_{21}X_w + p_{22}Y_w + p_{23}Z_w + p_{24}}{p_{31}X_w + p_{32}Y_w + p_{33}Z_w + p_{34}}$$

ע"מ למצוא את הפרמטרים p_{ij}, נדרשות שש נקודות על מנת ליצור 12 משוואות. ניתן לפרמל את המשוואות כמערכת לינארית:

$$\begin{aligned}&u(p_{31}X_w + p_{32}Y_w + p_{33}Z_w + p_{34}) = p_{11}X_w + p_{12}Y_w + p_{13}Z_w + p_{14} \Rightarrow\\&0 = p_{11}X_w + p_{12}Y_w + p_{13}Z_w + p_{14} - p_{31}uX_w - p_{32}uY_w - p_{33}uZ_w - p_{34}u =\\&\overrightarrow{a_{u,1}}^T \cdot \vec{p} = 0\end{aligned}$$

כאשר:

$$\vec{p} = (p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}, p_{31}, p_{32}, p_{33}, p_{34})^T$$
$$\vec{a}_{u,1} = (X_w, Y_w, Z_w, 1,0,0,0,0, -uX_w, -uY_w, -uZ_w, -u)^T$$

ובדומה לכך עבור v:

$$\begin{aligned}&v(p_{31}X_w + p_{32}Y_w + p_{33}Z_w + p_{34}) = p_{21}X_w + p_{22}Y_w + p_{23}Z_w + p_{24} \Rightarrow\\&0 = p_{21}X_w + p_{22}Y_w + p_{23}Z_w + p_{24} - p_{31}vX_w - p_{32}vY_w - p_{33}vZ_w - p_{34}v =\\&\overrightarrow{a_{v,1}}^T \cdot \vec{p} = 0\end{aligned}$$

כאשר:

$$\vec{a}_{v,1} = (0,0,0,0, X_w, Y_w, Z_w, 1, -vX_w, -vY_w, -vZ_w, -v)^T$$

כך ניתן להפוך את המשוואות למערכת משוואות מהצורה:

$$\boldsymbol{A}\vec{p} = \vec{0}$$

כאשר:

$$\boldsymbol{A} = \begin{pmatrix} -\vec{a}_{u,1}{}^T - \\ -\vec{a}_{v,1}{}^T - \\ \vdots \\ -\vec{a}_{u,6}{}^T - \\ -\vec{a}_{v,6}{}^T - \end{pmatrix}$$

מכיוון שאלה קואורדינטות הומוגניות, מכפלת $\boldsymbol{P}$ בסקלר אינה משפיעה על המשוואה, לכן נרצה להוסיף אילוץ:

$$|\vec{p}| = 1$$

קיים רעש במערכת, לכן ישנם מקרים בהם לא נוכל למצוא פרמטרים אשר מאפסים את המשוואה, אך נרצה להביא למינימום את הביטוי:

$$|\boldsymbol{A}\vec{p}|^2 = (\boldsymbol{A}\vec{p})^T(\boldsymbol{A}\vec{p}) = \vec{p}^T\boldsymbol{A}^T\boldsymbol{A}\vec{p} \to 0$$

ניתן להוכיח כי $\boldsymbol{A}^T\boldsymbol{A}$ היא סימטרית ומוגדרת חיובי (בהנחה כי אתם algebra wizard) ולכן קיימים 12 ו״ע אורטוגונליים.במקרים בהם הנקודות נבחרות בצורה ״לא אקראית״, למשל כאשר בוחרים את אותה הנקודה מספר פעמים, או מספר נקודות על אותו הקו, המטריצה $\boldsymbol{A}^T\boldsymbol{A}$ לא תהיה מדרגה מלאה. תחת הנחה זו ניתן למצוא ו״ע וע״ע:

$$\{\vec{e}_i\}_{i=1}^{12}, 0 \le \lambda_1 \le \cdots \le \lambda_{12}$$

הו״ע פורשים את המרחב ולכן כל וקטור $\vec{x}$ להכתב כקומבינציה לינארית של $\vec{e}_i$: $\vec{x} = \sum_{i=1}^{12}\mu_i e_i$. מכיוון שהו״ע אורטונורמלים מתקיים: $1 = |\vec{x}|^2 = \left|\sum_{i=1}^{12}\mu_i e_i\right|^2 = \sum_{i=1}^{12}\mu_i^2$. נבטא את המשוואה שלעיל בצורה זו:

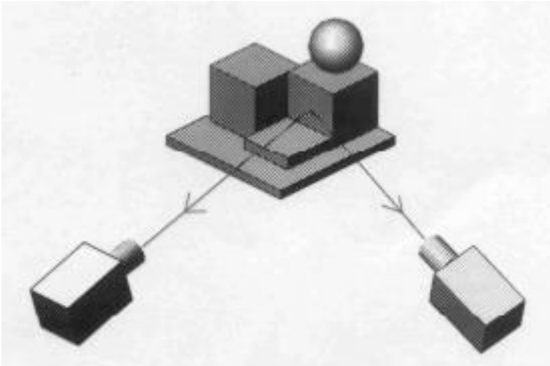
$$|\boldsymbol{A}\vec{x}|^2 = \vec{x}^T\boldsymbol{A}^T\boldsymbol{A}\vec{x} = \sum_i \mu_i\vec{e}_i\left(\boldsymbol{A}^T\boldsymbol{A}\sum_j \mu_j\vec{e}_j\right) = \sum_i \mu_i\vec{e}_i\sum_j \mu_j\lambda_j\vec{e}_j = \sum_i\left(\mu_i\vec{e}_i\sum_j \mu_j\lambda_j\vec{e}_j\right)$$
$$= \sum_i(\mu_i\mu_i\lambda_i) = \sum_i \mu_i^2\lambda_i \ge \lambda_1\sum_i \mu_i^2 = \lambda_1 = \vec{e}_1^T\lambda_1\vec{e}_1 = \vec{e}_1^T\boldsymbol{A}^T\boldsymbol{A}\vec{e}_1 = |\boldsymbol{A}\overrightarrow{e_1}|^2$$

- מעבר 3: נובע מכך שהו״ע של $\boldsymbol{A}^T\boldsymbol{A}$ הם $\overrightarrow{e_i}$.
- מעבר 5: נובע מכך ש-$\vec{e}_i$ אורטונורמליים (מטריצה סימטרית)
- מעבר 7: נובע מכך ש-$\vec{e}_1$ מוגדר כו״ע בעל ע״ע הקטן ביותר.
- מעבר 8: נובע מהאילוץ $|\vec{x}|^2 = 1$.

כלומר, קיבלנו שכל וקטור המקיים את האילוץ $|\vec{x}|^2 = 1$ גם מקיים: $|\boldsymbol{A}\vec{x}|^2 \ge |\boldsymbol{A}\overrightarrow{e_1}|^2$ כאשר $\vec{e}_1$ הוא הו״ע בעל הע״ע הקטן ביותר. ולכן, הוקטור $\vec{p}$ אשר מקיים את התנאי $\vec{p} = argmin_{\vec{x}}|\boldsymbol{A}\vec{x}|^2$ הוא $\overrightarrow{e_1}$.

Structured Light

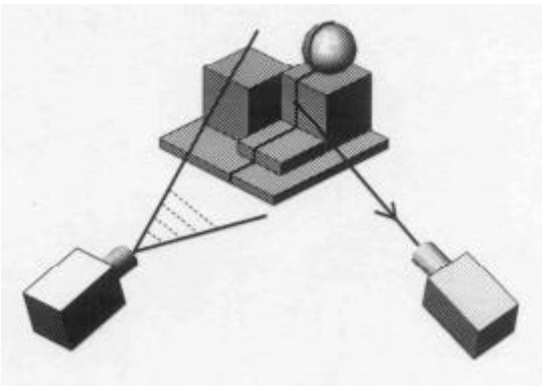
נניח כי אנו בונים *setup* של שתי מצלמות מכוילות המסתכלות על עצם. בהנחה כי מזוהה נקודה ספציפית בשתי המצלמות, ניתן למצוא את המרחק של הנקודה ע״י טריאנגולציה. ידועות לנו כיוון הקרניים היוצאות מכל אחת מהמצלמות (בזכות הקליברציה), ונקודת החיתוך שלהן היא הנקודה במרחב בו נמצא העצם. הבעיה בשיטה זו היא שנקודות תואמות הן נדירות, ולכן לא ניתן לשחזר את כל העצם התלת-ממדי. לרוב גם לא ניתן לעשות אינטרפולציה מכיוון שפרטים קטנים שלא זוהו בשתי המצלמות יאבדו. שיטה נוספת לבניית תמונת תלת ממד של סצנה, אשר נועדה להתגבר על בעיה זו, היא *structure light*.



תמונה 22 : סטאפ של תמונת סטריאו. אם מצליחים לזהות נקודה ספציפית בכל אחת מהמצלמות ניתן למצוא את מיקומה במרחב בעזרת חיתוך הקרניים.

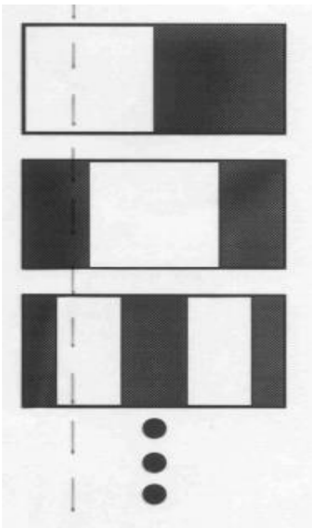
נתון סטאפ של מצלמה ומקור אור נקודתי, וידועים המיקום והזוית של כל אחד מהם. במצב כזה ניתן לשחזר את תמונת העומק של סצנה ע״י הארה של כל אחת מהנקודות בתמונה וביצוע טריאנגולציה למציאת המרחק. מאירים על כל נקודה ע״ג הסצנה, ועבור כל נקודה מוארת ידוע כיוון הקרן במקרן וכיוון הקרן המתקבלת במצלמה המכוייל. הבעייה בשיטה זו היא שנדרש להציב את מקור האור בכל אחת מהנקודות בתמונה ולכן התהליך עשוי להיות ארוך מאוד $-(O(n^2)$ תמונות, כאשר n היא הרזולוציה בציר אחד.

אפשרות יעילה יותר למימוש, במקום להשתמש במקור אור נקודתי ניתן להשתמש במקור אור ״משטחי״ – פליטה של אור דרך סדק צר היוצר אלומות המוכלות במישור אחד. אם אנו יודעים את זווית המשטח ואת כיוון האלומה המתקבלת במצלמה המכויילת, ניתן למצוא את מיקום הנקודה במרחב – נקודת החיתוך בין הקרן למשטח. בצורה זו ניתן להשיג בנייה של כל המרחב ב-$O(n)$ תמונות.



תמונה 23 : מצלמה ומקור אור ״משטחי״. בהינתן זווית המשטח והקרן המתקבלת במצלמה – ניתן לשחזר את מיקום הנקודה במרחב.

שיטה יעילה למימוש אלגוריתם זה היא על ידי *gray code labeling*. שיטה זו מאפשרת קידוד של כל ״משטח אור״ המתקבל במצלמה ע״י הקרנת סט ״מסכות״ מהצורה הבאה :



תמונה 24: gray code labeling, סט של תבניות המוקרנות, כל קו ("משטח אור") מקבל קוד. למשל הקו המקווקו בציור יקבל את הקוד (1,0,1, ...) – לבן במסכה הראשונה, שחור בשנייה, לבן בשלישית וכן הלאה.

בשיטה זו, כל פיקסל בתמונה מקבל ערך 1 או 0 כתלות בהארה שלו עבור כל אחת מהמסכות. עבור n תבניות כל פיקסל יקבל קידוד של n ביטים. ע"י קידוד זה ניתן להסיק על איזה "משטח" הפיקסל נמצא, ומכיוון שאנו יודעים על איזה קרן הוא נמצא אפשר לשחזר את מיקומו במרחב – חיתוך של הקרן במשטח.

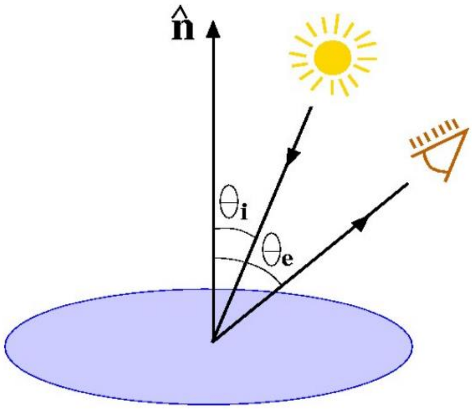
Photometric Stereo

Structure light מצריך צילום של מספר גבוה יחסית של תמונות על מנת לשערך ולשחזר סצנה תלת ממדית. קיימת שיטה נוספת, *photometric stereo*, אשר תחת הנחות מסוימות מאפשרת לשחזר תמונת עומק ע״י צילום מספר בודד של תמונות, ואף ע״י תמונה אחת.

באופן כללי, הגורמים המשפיעים על רמת הבהירות של פיקסל בתמונה הם:

- תאורה
- אור מוחזר
- מאפייני מצלמה
- גיאומטריה של הסצנה

שיטה זו מניחה שניתן לבודד את הגורם האחרון, ובהנחה כי כל שאר הגורמים קבועים וידועים **ניתן לשערך את הגיאומטריה של הסצנה ע״י רמת הבהירות של התמונה**.



תמונה 25: הנורמל למשטח, כיוון ההסתכלות וכיוון התאורה.

<u>מספר הגדרות:</u>

- $\hat{n}$ – הנורמל למשטח בנקודה
- זווית θ_i – הזווית בין הנורמל של המשטח לכיוון התאורה
- זווית θ_e – הזווית בין הנורמל של המשטח לכיוון הצפייה

אם נמצא את כיוון הנורמל עבור כל נקודה שע״ג המשטח, ניתן לשחזר את העצם בתלת ממד.

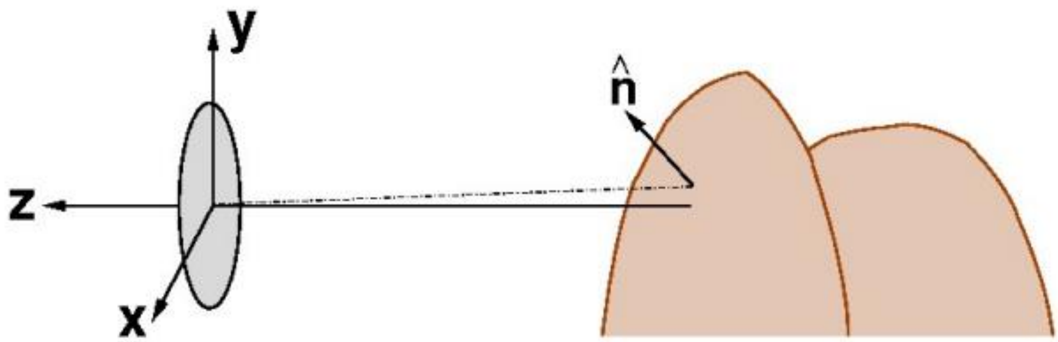
זווית ההסתכלות וזווית התאורה

לצורך הפשטות נניח משטח גזיר שניתן לתאר כ-$z(x,y)$. ניתן למצוא את המשטח המשיק לנקודה (x,y) ע״י מכפלה וקטורית בין שני וקטורים המוכלים במשטח:

$$\partial z_x = \frac{\partial z}{\partial x}\cdot\partial x + e = p\cdot\partial x + e \Rightarrow (\partial x, 0, p\cdot\partial x) \Rightarrow \vec{r}_x = (1,0,p)$$
$$\partial z_y = \frac{\partial z}{\partial y}\cdot\partial y + e = q\cdot\partial y + e \Rightarrow (0, \partial y, q\cdot\partial y) \Rightarrow \vec{r}_y = (0,1,q)$$
$$\vec{n} = \vec{r}_x \times \vec{r}_y = (-p,-q,1)^T$$
$$\hat{n} = \frac{\vec{n}}{|\vec{n}|} = \frac{(-p,-q,1)^T}{\sqrt{1+p^2+q^2}}$$

כלומר, ניתן להגדיר כיוון של נורמל ע״י שני פרמטרים (p,q).

עבור זוויות קטנות (פיקסלים קרובים למרכז/עדשה עם זום גדול), ניתן להניח כי הישר בין הנקודה שע״ג המשטח למרכז העדשה מקביל בקירוב לציר האופטי, ולכן $\cos\theta_e \cong \hat{n}\cdot(0,0,1) = \frac{1}{\sqrt{1+p^2+q^2}}$. לפיכך **ניתן לדעת את הזווית שבין הנורמל לציר האופטי של המצלמה ע״י (p,q)**. נותר למצוא את הפרמטרים הללו ע״מ לשחזר את מיקום הנקודה במרחב.



תמונה 26 : הזווית בין הנקודה לעדשה קרוב מאוד לציר z ולכן $\cos\theta_e \cong \hat{n}\cdot(0,0,1)$

בהנחה כי התאורה מגיעה בקווים מקבילים (למשל תאורה של השמש), ניתן לבטא את כיוון התאורה ע״י :

$$\vec{n}_s = (-p_s, -q_s, 1)$$

ואת הנורמל באותו כיוון :

$$\hat{n}_s = \frac{\vec{n}_s}{|\vec{n}_s|} = \frac{(-p_s, -q_s, 1)}{\sqrt{1+p_s^2+q_s^2}}$$

את זווית θ_i ניתן למצוא ע״י המכפלה הסקלרית של הנורמל עם כיוון התאורה $\hat{n}_s$:

$$\cos\theta_i = \hat{n}\cdot\hat{n}_s = \frac{(-p,-q,1)}{\sqrt{1+p^2+q^2}}\cdot\frac{(-p_s,-q_s,1)}{\sqrt{1+p_s^2+q_s^2}} = \frac{1+p_s p+q_s q}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}}$$

מפת החזרה ***Reflection Map***

מפת החזרה היא פונקציה המגדירה את כמות ההחזרה של התאורה עבור כל זווית של המשטח. הפונקציה תלויה בסוג החומר ובתאורה.

הנחות :

- משטח מחומר אחיד
- תאורה אחידה וקבועה
- מיקום המצלמה והזווית ידועים

תחת הנחות אלו בהירות הנקודה תלויה אך ורק בכיוון המשטח :

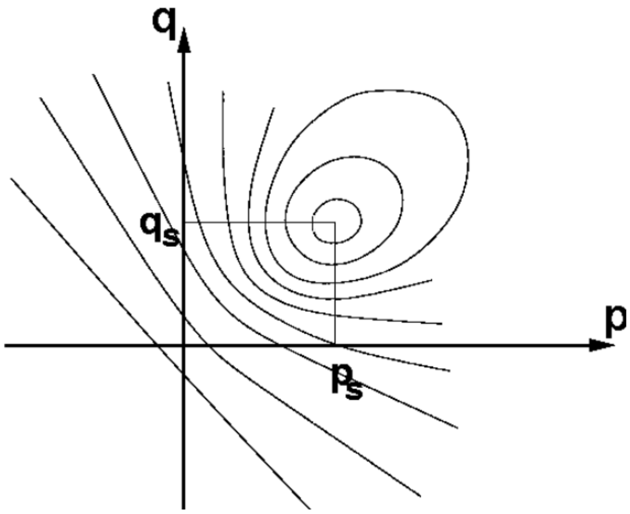
$$E(x,y) \propto R\big(p(x,y), q(x,y)\big)$$

כאשר $E(x,y)$ הוא הבהירות הנקלטת בפיקסל (x,y) ו-$R(p,q)$ הוא מפת ההחזרה (reflection map).
כאשר אין נקודות סינגולריות ע״ג המשטח ניתן ע״י נרמול לקבל :

$$E(x,y) = R\big(p(x,y), q(x,y)\big)$$

ע״פ המשוואה, אם ידועות הנגזרות החלקיות נקודה ע״ג המשטח, ניתן לדעת מה תהיה הבהירות של הפיקסל הספציפי. הבעיה היא שהפונקציה $R(p,q)$ אינה הפיכה – לרוב קיימות מספר נגזרות (p,q) אשר

יניבו את אותה החזרה, ולכן בהינתן בהירות לא ניתן לשחזר בוודאות את הנגזרות בנקודה. עם זאת, ניתן למפות את הקווים שווי הפוטנציאל ב-$R(p,q)$ ע"מ לדעת את סט הערכים (p,q) האפשרי.



תמונה 27 : הפונקציה $R(p,q)$ מגדירה את כמות ההחזרה עבור כל נורמל. בהינתן כמות החזרה מסויימת ניתן למפות קו שווה פוטנציאל אשר בו שוכנים אוסף הנורמלים האפשריים. הנקודה $(0,0)$ היא הנקודה בה הנגזרות שוות לאפס – כיוון ההסתכלות של המצלמה. בנקודה (p_s,q_s) – כיוון התאורה, ההחזרה תהיה מקסימלית ותתקבל נקודת מקסימום.

מציאת מפת ההחזרה $R(p,q)$

משטח למברטי

משטח למברטי (Lambertian surface) הוא משטח "מט" (לא מבריק). במשטח כזה הבהירות אינה תלויה בנקודת המבט (היא תהייה זהה עבור כל זווית הסתכלות) אלא רק בזווית בין מקור התאורה למשטח.

באופן כללי, מפת ההחזרה לא ידועה, ויש למדוד את ההחזרה עבור כל כיוון תאורה. אך במקרה של משטח למברטי קיימת משוואה ידועה. במקרה זה הפונקציה פורפורציונלית לקוסינוס הזווית שבין כיוון התאורה לנורמל למשטח :

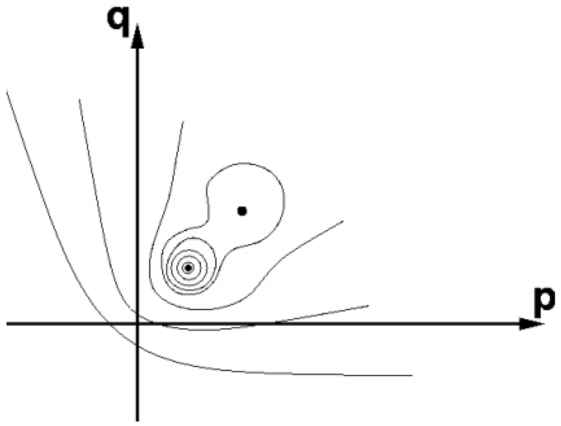
$$R(p,q)=\hat{n}\cdot\hat{n}_s=\frac{(-p,-q,1)}{\sqrt{p^2+q^2+1}}\frac{(-p_s,-q_s,1)}{\sqrt{p_s^2+q_s^2+1}}$$

התאורה המקסימלית תתקבל עבור נורמל המקיים : $(p_s,q_s)=(p,q)$, כפי שניתן לראות באיור לעיל.

משטח מבריק Specular

במשטח מבריק, נראה החזרה מקסימלית כאשר הנורמל למשטח משמש כחוצה זווית בין זווית הפגיעה של האור לזווית ההסתכלות, כלומר :

$$(p,q)\approx(p_{s/2},q_{s/2})$$



תמונה 28 : משטח מבריק למחצה. ניתן לראות מקסימום לוקלי עבור $(p,q) = (p_s, q_s)$ ומקסימום גלובלי עבור נורמל המשמש חוצה זווית בין כיוון ההסתכלות וכיוון התאורה – $(\frac{p_s}{2}, \frac{q_s}{2})$.

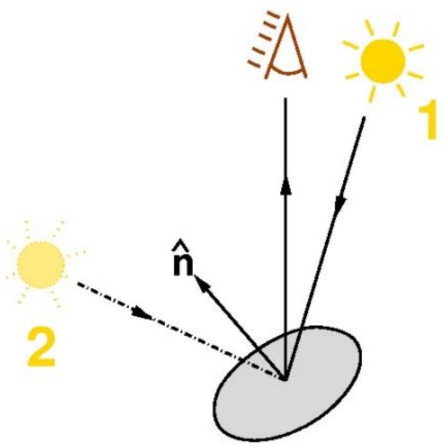
מבריק למחצה – *Glossy*

משטח מבריק למחצה מחזיר אור כמו קומבינציה של משטח מבריק ומט.

Photometric Stereo

נתון $R(p,q)$ ונתון כי $E(x,y) = R(p,q)$. בנקודות המקסימום של $R(p,q)$ ($R(p,q) = 1$) ברור כי הנורמל בכיוון של מקור האור, אך במקומות בהן הוא קטן מ-1 הוא יכול להיות בכל מסלול שווה פוטנציאל במישור (p,q) ולכן לא ניתן לשחזר את הנורמל.

על מנת להתגבר על הבעיה ניתן להשתמש בשני מקורות אור במיקומים שונים, כך שהזוויות יחסית לנורמל ישתנו וכן גם הקווים שווי הפוטנציאל :



תמונה 29 : הוספת מקור אור נוסף תיתן זווית אחרת לנורמל ופונקציית החזרה אחרת

מתקיים :

$$E_1(x,y) = R_1(p,q)$$
$$E_2(x,y) = R_2(p,q)$$

הערות:

- הסיבה לכך שה-R שונים היא שהפונקציה תלויה בין השאר במיקום התאורה.
- מכיוון שהמשוואות לא תמיד לינאריות, בד"כ שתי משוואות לא מספיקות ויש להוסיף מקורות תאורה.

בהנחת משטח למברטי, מתקיים:

$$E_1(x,y) = R_1(p,q) = \frac{1+p_1p+q_1q}{\sqrt{1+p_1^2+q_1^2}\sqrt{1+p^2+q^2}}$$
$$E_2(x,y) = R_2(p,q) = \frac{1+p_2p+q_2q}{\sqrt{1+p_2^2+q_2^2}\sqrt{1+p^2+q^2}}$$

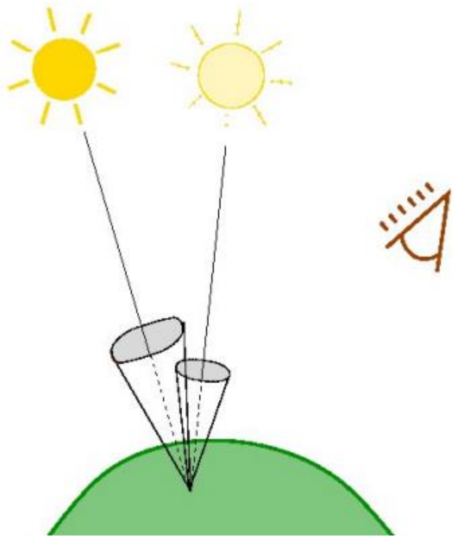
כאשר $R_1(p,q), R_2(p,q)$ הן משוואות ההחזרה עבור כל אחת מהתאורות ו-(p_i,q_i) הם כיווני ההארה.

במקרה זה ניתן להוכיח שהקווים שווי הפוטנציאל הם אליפסות, ולכן נחפש את נקודת החיתוך של שתי אליפסות. במקרה הכללי יכולות להיות ארבעה נקודות חיתוך, אך במקרה שלנו ישנם שתי נקודות.

<u>הוכחה:</u>

$$\frac{E_1(x,y)}{E_2(x,y)} = \frac{1+p_1p+q_1q}{1+p_2p+q_2q}\frac{\sqrt{1+p_1^2+q_1^2}}{\sqrt{1+p_2^2+q_2^2}} \Rightarrow$$
$$Const_1 = \frac{1+p_1p+q_1q}{1+p_2p+q_2q}Const_2$$

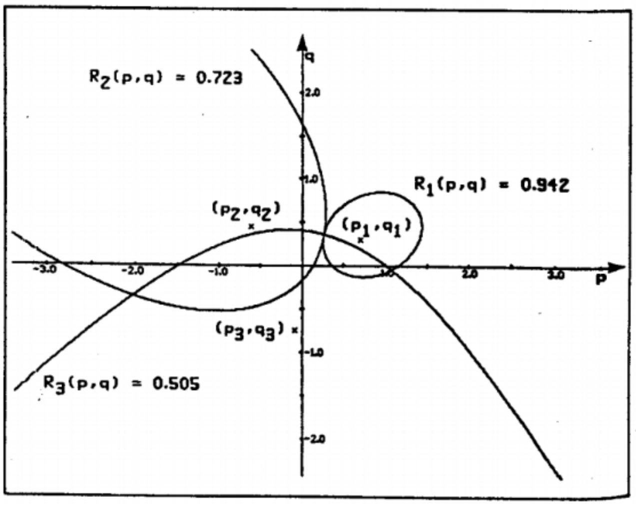
ניתן להפוך את המשוואה למשוואה לינארית, ולכן הפרמטרים נמצאים על קו ישר וקיימים רק שתי פתרונות.



תמונה 30: עבור ערך של פונקציית החזרה $E(x,y)$ זווית תאורה אחת תיתן אליפסה במישור (p,q). זווית תאורה נוספת תיתן אליפסה נוספת – נקודת החיתוך של האליפסות יתנו זוג אפשרויות ל-(p,q).

מתקבלים שני פתרונות, ולכן אין לנו עדיין את כיוון הנורמל של הנקודה. פתרון אפשרי לבעיה הוא הוספת מקור תאורה שלישי. היתרונות בפתרון כזה:

- ניתן לבצע שחזור של הנורמל המדויק למשטח.
- חיתוך של שלוש מקורות נותן תוצאה יותר רובסטית ופחות רגישה לרעש.
- ניתן לשחזר את התמונה התלת ממדית אפילו אם ה-albedo (כלומר הצבע) של העצם משתנה.
- בנוסף, הפתרון שמתקבל הינו אלגנטי, לינארי ובדרך כלל גם יחיד.



תמונה 31 : נקודת חיתוך של שלושה קווים שווי פוטנציאל הנובעים משלושה מקורות תאורה

במקרה זה המשוואות מהצורה :

$$E_i(x,y) = R_i(x,y) = \rho(x,y)\left(\hat{S}_i \cdot \hat{n}\right) \qquad i = 1,2,3$$

כאשר $\widehat{S_i}$ הוא כיוון התאורה עבור מקור i.

ניתן לייצג את המשוואות בצורה מטריצית :

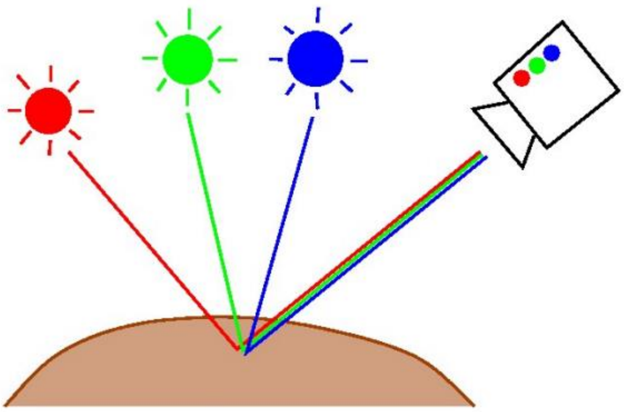
$$\vec{E} = \rho\, \boldsymbol{S} \cdot \hat{n}$$

כאשר : $\vec{E} = \begin{pmatrix} E_1 \\ E_2 \\ E_3 \end{pmatrix}$ ו- $\boldsymbol{S} = \begin{pmatrix} -\hat{S}_1 - \\ -\hat{S}_2 - \\ -\hat{S}_3 - \end{pmatrix}$.

ניתן לבחור מקורות אור S_i כך שהמטריצה $\boldsymbol{S}$ תהיה לא סינגולרית ולכן הפיכה. כך ניתן למצוא את $\rho(x,y)$ ואת $\hat{n}$.

$$\rho\, \hat{n} = \widehat{\boldsymbol{S}}^{-1} \cdot \vec{E} \Rightarrow \rho = \|\rho\, \hat{n}\| = \left\|\widehat{\boldsymbol{S}}^{-1} \cdot \vec{E}\right\| \Rightarrow \hat{n} = \frac{1}{\rho}\widehat{\boldsymbol{S}}^{-1} \cdot \vec{E}$$

על מנת לפשט את התהליך, ניתן להשתמש בשלוש מקורות אור – R,G,B ובמצלמה אחת, כך שמספיק לצלם תמונה אחת בעלת שלוש ערוצים.



תמונה 32 : שלוש מקורות תאורה שונים בצבעים (R,G,B). במקרה זה ניתן ע״י תמונת צבע אחת לקלוט את שלושת המקורות.

הערות:

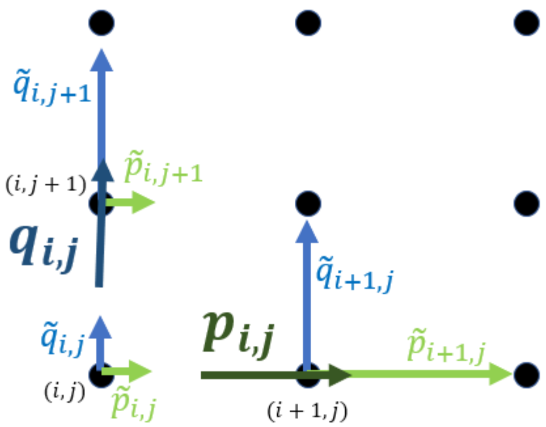
- אחרי שמוצאים את הנורמל של כל נקודה, נדרש עוד לשחזר את הסצנה, משימה זו לא תמיד קלה.
- לאחר שחזור כזה ניתן לבנות מחדש אובייקט, אך לא ידוע המרחק האבסולוטי (מה שנמצא היה נגזרות וכאשר מבצעים "אינטגרל" יש סקלר קבוע ש"אבד").
- קיימות הרחבות לנושאים הנ"ל, למשל – משטחים לא למברטיים.

שחזור משטח מגרדיאנטים

בעזרת photometric stereo הצלחנו להגיע למפת גרדיאנטים (p,q). על מנת להגיע לשחזור המשטח נותר למצוא דרך לשחזור משטח מתוך needle map של גרדיאנטים.

נגדיר $\{(\tilde{p}_{i,j},\tilde{q}_{i,j})\}_{i,j}$ את הנגזרות החלקיות בנקודות (i,j). נדרש למצוא את $z_{i,j}$ – גובה המשטח בנקודה. נגדיר:

$$p_{i,j}=\frac{\tilde{p}_{i,j}+\tilde{p}_{i+1,j}}{2}$$
$$q_{i,j}=\frac{\tilde{p}_{i,j}+\tilde{p}_{i,j+1}}{2}$$



תמונה 33: הגדרת p_{ij},q_{ij} כממוצע הנגזרות החלקיות ליד הנקודה (i,j)

$p_{i,j}$ הוא בקירוב הנגזרת החלקית לפי x בין פיקסל (i,j) ל-$(i+1,j)$, וכך בהתאמה $q_{i,j}$ הנגזרת לפי y בין (i,j) ל-$(i,j+1)$. ניתן לדרוש:

$$\begin{cases} z_{i+1,j}-z_{i,j}=p_{i,j} \\ z_{i,j+1}-z_{i,j}=q_{i,j} \end{cases}$$

כלומר שהנגזרת החלקית הדיסקרטית של $z_{i,j}$ לפי x ו-y תהיה שווה ל-$p_{i,j}$ ו-$q_{i,j}$ בהתאמה.

כאשר נאסוף את המשוואות עבור כל i,j נקבל עבור כל נקודה נעלם אחד - $z_{i,j}$ ושתי משוואות. מערכת כזאת הינה "מוגדרת יתר" וברוב המקרים לא יהיה ניתן לפתור (מכיוון שיש רעש לא נוכל להגיע לפתרון שיקיים את כל המשוואות). על מנת להתגבר על הבעיה נגדיר פונקציית מטרה, על ידי מציאת המינימום לפונקציה נמצא פתרון לבעיה. הפונקציה היא:

$$argmin\left(\sum_{i,j}\left(z_{i+1,j}-z_{i,j}-p_{i,j}\right)^2+\left(z_{i,j+1}-z_{i,j}-q_{i,j}\right)^2\right)$$

מינימיזציה של פונקציה זו תביא ערכים ל-$z_{i,j}$ שיניבו נגזרות קרובות ככל הניתן (במובן של MSE) לנגזרות הנתונות.

נגזור את הביטוי על מנת למצוא מינימום :

$$\frac{d}{dz_{i',j'}}\sum_{i,j}\left(z_{i+1,j}-z_{i,j}-p_{i,j}\right)^2+\left(z_{i,j+1}-z_{i,j}-q_{i,j}\right)^2$$
$$=\frac{d}{dz_{i',j'}}\Big(\left(z_{i',j'}-z_{i'-1,j'}-p_{i'-1,j'}\right)^2+\left(z_{i',j'}-z_{i',j'-1}-q_{i',j'-1}\right)^2$$
$$+\left(z_{i'+1,j'}-z_{i',j'}-p_{i',j'}\right)^2+\left(z_{i',j'+1}-z_{i',j'}-q_{i',j'}\right)^2\Big)=$$
$$=2\left(z_{i',j'}-z_{i'-1,j'}-p_{i'-1,j'}\right)+2\left(z_{i',j'}-z_{i',j'-1}-q_{i',j'-1}\right)-2\left(z_{i'+1,j'}-z_{i',j'}-p_{i',j'}\right)$$
$$-2\left(z_{i',j'+1}-z_{i',j'}-q_{i',j'}\right)$$
$$=2z_{i',j'}-2z_{i'-1,j'}-2p_{i'-1,j'}+2z_{i',j'}-2z_{i',j'-1}-2q_{i',j'-1}-2z_{i'+1,j'}$$
$$+2z_{i',j'}+2p_{i',j'}-2z_{i',j'+1}+2z_{i',j'}+2q_{i',j'}=$$
$$=2\Big(\left[4z_{i',j'}-z_{i'-1,j'}-z_{i'+1,j'}-z_{i',j'+1}-z_{i',j'-1}\right]+\left[p_{i',j'}-p_{i'-1,j'}\right]$$
$$+\left[q_{i',j'}-q_{i',j'-1}\right]\Big)=0$$

ניתן להבחין כי הביטוי :

$$\left[4z_{i',j'}-z_{i'-1,j'}-z_{i'+1,j'}-z_{i',j'+1}-z_{i',j'-1}\right]+\left[p_{i',j'}-p_{i'-1,j'}\right]+\left[q_{i',j'}-q_{i',j'-1}\right]=0$$

הוא המקבילה הדיקרטית לפתרון הרציף :

$$-\nabla^2 z+\frac{dp}{dx}+\frac{dq}{dy}=0$$

את הביטוי ניתן לפתור בצורה אנליטית בעזרת אלגברה אך גם בצורה איטרטיבית. מגרילים (בצורה אקראית או בצורה "מחוכמת" יותר) את מפת הגבהים $z_{i,j}$ ופותרים שוב ושוב את הביטוי :

$$z_{i',j'}^{(k+1)}=\frac{1}{4}\Big(\left[z_{i'-1,j'}^{(k)}+z_{i'+1,j'}^{(k)}+z_{i',j'+1}^{(k)}+z_{i',j'-1}^{(k)}\right]-\left[p_{i',j'}-p_{i'-1,j'}\right]-\left[q_{i',j'}-q_{i',j'-1}\right]\Big)$$

הערות

- ברוב המקרים הביטוי יתכנס, אם כי בצורה איטית.
- ישנן בעיות בהן לא נדרש שחזור של הסצנה התלת ממדית וניתן להסתפק במפת הגרדיאנטים.
- היתרון הגדול של השיטה הוא שהיא מאפשרת שחזור תלת ממדי בעזרת תמונה אחת, דבר זה מאפשר פתרון לבעיות כמו שחזור תלת ממדי של סצנה לא סטטית.
- ניתן להרחיב את שיטות ה-Photometric stereo גם למשטחים לא למברטיים ע"י שימוש בארבעה מקורות תאורה.

תמונת סטריאו

כעת נחזור לגיאומטריה אפיפולרית ולמערכת של שתי מצלמות.

נזכיר את המשוואה הבאה:

$$\vec{x} \cong \boldsymbol{C_{3\times4}}\left[\boldsymbol{R_{4\times3}} \mid \vec{t}_{\boldsymbol{4\times1}}\right]\vec{X}_{4\times1} = \boldsymbol{P}\vec{X}$$

הסימן $\cong$ מבטא כי השוויון נכון עד כדי מכפלה בסקלר (scaling). המטריצה $\boldsymbol{C}$ היא המטריצה האינטרינזית והיא כומסת את כל הפרמטרית של המצלמה, והמטריצה $\left[\boldsymbol{R} \mid \vec{t}\,\right]$ היא המטריצה האקסטרינזית אשר מבטאת את המעבר ממערכת הצירים של "העולם" למערכת הצירים של המצלמה.

הוקטור $\vec{X}$ הוא הקורדינטות של העצם במרחב (מבוטא כקואורדינטות הומוגניות) ו-$\vec{x}$ הוא אותו עצם כפי שמושלך ע"ג התמונה.

נשים לב כי הפתרון אינו יחיד, שכן ניתן לכתוב:

$$\vec{x} = [\boldsymbol{PG}]\left[\boldsymbol{G^{-1}}\vec{X}\right]$$

ניתן להעתיק את הנקודה $\vec{X}$ בעזרת טרנספורמציה $\boldsymbol{G^{-1}}$, ועדיין לקבל את הנקודה במישור התמונה $\vec{x}$. דבר זה שקול לבחירת מערכת צירים אחרת "בעולם" – הקואורדינטות של $\vec{X}$ ישתנו וכך גם המטריצה האקסטרינזית. תכונה זו נקראת gauge freedom.

קיימות שלוש בעיות בתחום זה:

1) *DLT – direct linear transform*
2) טריאנגולציה
3) *Structure from motion*

DLT – direct linear transform

בהינתן $\{\vec{x}_i\}, \{\vec{X}_i\}$ – זוגות של נקודות – לשחזר את $\boldsymbol{P}$. ראינו לעיל שנדרשות 6 זוגות של נקודות.

טריאנגולציה

בהינתן $\{\vec{x}_i\}$ ו-$\boldsymbol{P}$ לשחזר את $\vec{X}$. לצורך זה נדרשות שתי מצלמות מכויילות, ודגימה אחת מכל מצלמה $\vec{x}, \vec{x}', \boldsymbol{P}, \boldsymbol{P}'$.

<u>הוכחה:</u>

נתונים הערכים הבאים: $\vec{x}, \vec{x}', \boldsymbol{P}, \boldsymbol{P}'$. מתקיים:

$$\vec{x} \cong \boldsymbol{P}\vec{X}$$
$$\overrightarrow{x'} \cong \boldsymbol{P}'\vec{X}$$

$$u = \frac{\vec{P}_1 \cdot \vec{X}}{\vec{P}_3 \cdot \vec{X}}$$

$$v = \frac{\vec{P}_2 \cdot \vec{X}}{\vec{P}_3 \cdot \vec{X}}$$

כאשר $\vec{P}_i$ הוא השורה ה-i במטריצה $\boldsymbol{P}$.

כך גם עבור המצלמה השנייה:

$$u' = \frac{\vec{P'}_1 \cdot \vec{X}}{\vec{P'}_3 \cdot \vec{X}}$$

$$v' = \frac{\vec{P'}_2 \cdot \vec{X}}{\vec{P'}_3 \cdot \vec{X}}$$

ניתן לבטא את המשוואות במטריצה בצורה הבאה:

$$u' = \frac{\vec{P'}_1 \cdot \vec{X}}{\vec{P'}_3 \cdot \vec{X}} \Rightarrow u'\vec{P'}_3 \cdot \vec{X} = \vec{P'}_1 \cdot \vec{X} \Rightarrow 0 = \left(\vec{P'}_1 - u'\vec{P'}_3\right) \cdot \vec{X}$$

ובאופן דומה עבור v, u, v':

$$0 = \left(\vec{P'}_2 - v'\vec{P'}_3\right) \cdot \vec{X}$$
$$0 = \left(\vec{P}_1 - u\vec{P}_3\right) \cdot \vec{X}$$
$$0 = \left(\vec{P}_2 - v\vec{P}_3\right) \cdot \vec{X}$$

ובמטריצה:

$$\begin{pmatrix} \vec{P}_1 - u\vec{P}_3 \\ \vec{P}_2 - v\vec{P}_3 \\ \overrightarrow{P'}_1 - u'\overrightarrow{P'}_3 \\ \overrightarrow{P'}_2 - v'\overrightarrow{P'}_3 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ \lambda \end{pmatrix} = 0$$

מאלגברה לינארית, הפתרון למשוואה הוא הו"ע בעל הע"ע הקטן ביותר. על מנת למצוא את הנקודה במרחב כל מה שנותר הוא לחלק באיבר האחרון:

$$\overrightarrow{X_w} = \begin{pmatrix} X_w/\lambda \\ Y_w/\lambda \\ Z_w/\lambda \end{pmatrix}$$

במקרה בו קיימות שלוש מצלמות ניתן להרחיב את המטריצה לצורה הבאה:

$$\boldsymbol{A} = \begin{pmatrix} \vec{P}_1 - u\vec{P}_3 \\ \vec{P}_2 - v\vec{P}_3 \\ \overrightarrow{P'}_1 - u'\overrightarrow{P'}_3 \\ \overrightarrow{P'}_2 - v'\overrightarrow{P'}_3 \\ \overrightarrow{P''}_1 - v''\overrightarrow{P''}_3 \\ \overrightarrow{P''}_2 - v''\overrightarrow{P''}_3 \end{pmatrix}$$

מערכת כזאת היא "מוגדרת יתר" (over-determined) מכיוון שיש שש משוואות וארבעה נעלמים. ע"מ להתגבר על הבעיה ניתן למצוא את הו"ע המתאים לע"ע הנמוך ביותר של $\boldsymbol{A^T A}$ (גודלה 4×4).

הערות

- כאשר שני הע"ע הקטנים ביותר קרובים זה לזה – סימן ששני הו"ע "כמעט תלויים" זה בזה.

Structure from motion

בבעיה זו נתונות זוגות נקודות משתי מצלמות $\{\vec{x}_i\}, \{\vec{x'}_i\}$ ונדרש לשחזר את $\boldsymbol{P}, \boldsymbol{P'}, \vec{X}_i$. בהנחה שמצאנו את $\boldsymbol{P}, \boldsymbol{P'}$ ניתן למצוא את $\vec{X}_i$ כפי שראינו לעיל. הבעיה היא מציאת $\boldsymbol{P}, \boldsymbol{P'}$.

על מנת להתגבר על ה-gauge freedom שפגשנו לעיל, נגדיר את מערכת הקואורדינטות של מצלמה אחת להיות מערכת הצירים של "העולם". במקרה זה המטריצה האקסטרינזית של המצלמה תהיה (עבור קואורדינטות לא-הומוגניות):

$$[\boldsymbol{R} \mid \vec{t}] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

נרצה לעבור בין נקודה (u, v) במצלמה ראשונה ל-(u', v') שבמצלמה השניה. הטרנספורמציה הפשוטה ביותר תהיה **הטרנספורמציה הלינארית**:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} u' \\ v' \end{pmatrix} = \boldsymbol{L} \begin{pmatrix} u' \\ v' \end{pmatrix}$$

מטריצה כזו מאפשרת:

- סיבוב סביב ראשית הצירים
- Non-uniform scaling – ניתן לכווץ/למתוח בשני צירים באופן שונה
- Shear – שינוי זווית בין צירים

במטריצה זו ישנן ארבעה נעלמים – ולכן נצטרך שתי נקודות כדי למצוא את המטריצה.

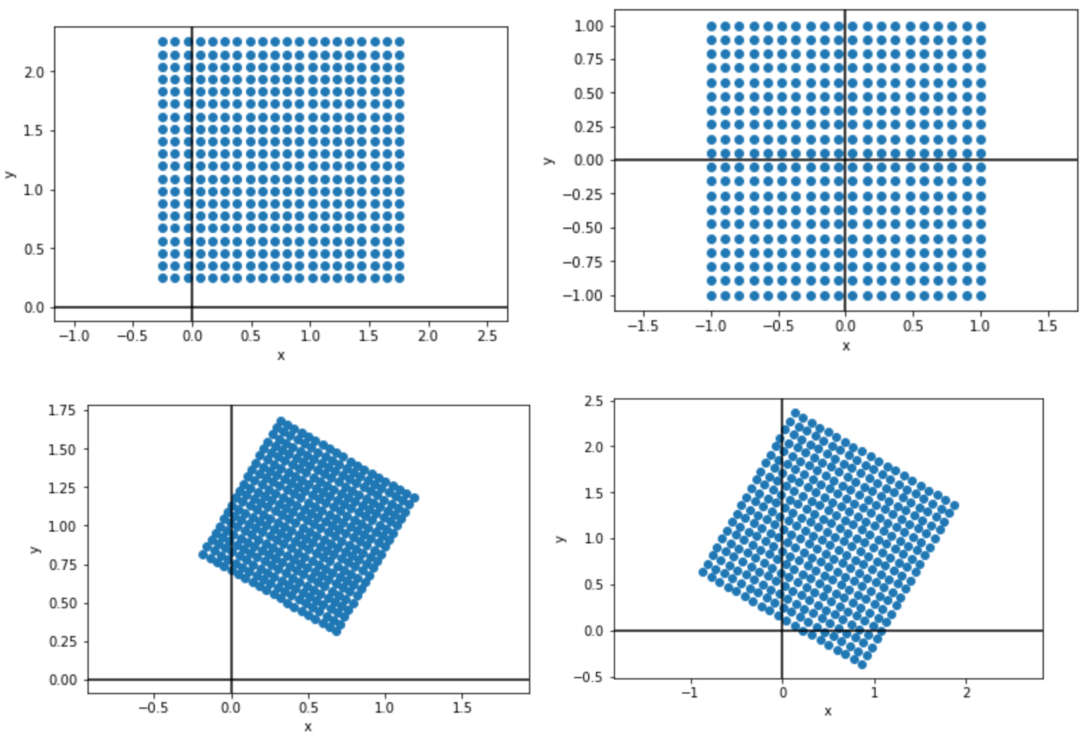
על מנת לאפשר הזזה של הנקודה, נצטרך לשכלל את המטריצה:

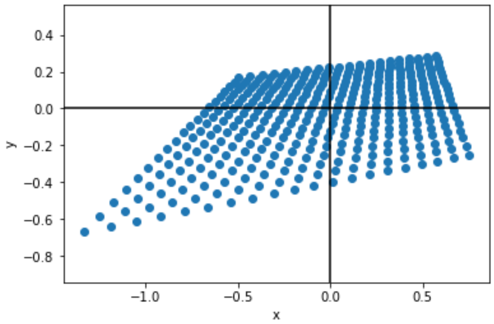
$$\begin{pmatrix} u \\ v \end{pmatrix} = (\boldsymbol{L} \mid \vec{t}) \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = \boldsymbol{A} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix}$$

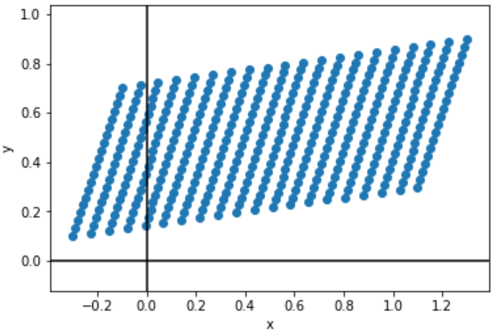
טרנספורמציה כזו נקראת **טרנספורמציה אפינית**. במטריצה זו ישנן שישה נעלמים – ולכן נצטרך שלוש נקודות כדי למצוא את המטריצה.

הטרנספורמציה הבאה נקראת **טרנספורמציה הומוגרפית** והיא מהצורה:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \cong \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = \boldsymbol{H}_{3\times 3} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix}$$







תמונה 34: וקטורים מועתקים בטרנספורמציות: הזזה (translation), סיבוב (euclidian), מתיחה וכיווץ (similarity), אפינית (affine) והומוגרפית (homography, projective)

ע״י בחירת ערכי פרמטרים מתאימים ניתן לבטא גם טרנספורמציה אפינית בעזרת 3×3:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} = \boldsymbol{A}$$

מכיוון שהמטריצה $\boldsymbol{H}$ פועלת על וקטורים בקואורדינטות הומוגניות, היא לא מושפעת מכפל בסקלר, לכן ישנן רק שמונה דרגות חופש ומספיקות ארבעה נקודות על מנת למצוא את הטרנספורמציה.

המשוואות אותן יש לפתור:

$$u_i = \frac{\overrightarrow{H_1} \cdot \vec{x}'_i}{\overrightarrow{H_3} \cdot \vec{x}'_i}$$
$$v_i = \frac{\overrightarrow{H_2} \cdot \vec{x}'_i}{\overrightarrow{H_3} \cdot \vec{x}'_i}$$
$$i = 1,2,3,4$$

נהפוך את המערכת למערכת משוואות מטריצית:

$$\begin{aligned} u_i\, \overrightarrow{H_3} \cdot \vec{x}'_i = \overrightarrow{H_1} \cdot \vec{x}'_i \quad &= \quad u_i(gu'_i + hv'_i + i) = (au' + bv' + c\,) \\ &= u_i(0,0,0,0,0,0,u'_i,v'_i,1) \cdot \vec{h} = (u'_i,v'_i,1,0,0,0,0,0,0) \cdot \vec{h} \;\Rightarrow \\ 0 = [(u'_i,v'_i,1,0,0,0,0,0,0) - u_i(0,0,0,0,0,0,u'_i,v'_i,1)] \cdot \vec{h} \end{aligned}$$

בדומה ניתן למצוא את המשוואה עבור v_i. לאחר שנציב את הביטויים במטריצה נקבל:

$$\boldsymbol{H_{8\times 9}} \cdot \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{pmatrix} = \vec{0}$$

ניתן לקבע את h_9 להיות שווה ל-1 על מנת להשאיר שמונה דרגות חופש, אך פתרון זה עלול להיות לא יציב. במקום זאת נקבע אילוץ $|\vec{h}| = 1$.

כמו שעשינו בעבר, ניתן למצוא את $\vec{h}$ ע״י מציאת הו״ע המתאים לע״ע הקטן ביותר.

הערות

- כמו שהערנו לעיל, במקרה בו ישנן יותר נקודות ניתן לפתור את $\boldsymbol{H^T H}$, אך להלן נדון בשיטה נוספת שפחות רגישה לרעש (RANSAC).
- שימוש בהומוגרפיה מתאים למקרים הבאים:
  - כאשר בין המצלמות יש רק סיבוב סביב ציר התמונה ללא הזזה.
  - שתי תמונות של סצנה פלנרית.

RANSAC

בנושאים שדנו לעיל היו פתרונות שונים למקרה בו קיימות יותר נקודות מאשר המודל דורש. הצענו פתרונות להתגבר על הבעיה, אך כל הפתרונות רגישים ל-*outliers*, כלומר – נקודות של רעש משפיעות על המודל. אלגוריתם ה-*RANSAC* (*RANdom SAmpling Concensus*) נועד להתגבר על הבעיה.

דוגמה להסבר האלגוריתם:

נתונות דגימות $\{x_i, y_i\}$ הדגומות מתוך קו ישר. ידוע כי ישנן דגימות רועשות – *outliers*. נדרש למצוא את (a, b, c) אשר מקיימים את משוואות הקו הישר $ax + by + c = 0$. ניתן לפתור כפי שפתרנו בעבר:

$$\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_i & y_i & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \vec{0}$$

הבעיה בפתרון שכל נקודה תורמת לפתרון ולכן נקודות רועשות משפיעות לרעה. אלגוריתם ה-*RANSAC* מתגבר על הבעיה. האלגוריתם מגדיר תהליך כדלהלן:

1) הגרלה של שתי נקודות (-מספר הנקודות המינימלי הדרוש להגדרת קו ישר)
2) מציאת הפרמטרים (a, b, c) של הקו העובר בין הנקודות
3) בדיקת כמות הנקודות אשר נמצאות על הקו
4) שמירת הפרמטרים אשר נותנים את המודל הטוב ביותר

מספר הגדרות לאלגוריתם:

- $w -$ אחוז ה-*inlier*
- $p -$ הסתברות לתוצאה נכונה (דרישה מהאלגוריתם)
- $n -$ מספר נקודות מינימלי להגדרת מודל
- $t -$ מרחק מינימלי להגדרת נקודה כ-*outlier*
- $d -$ מספר נקודות מינימלי ל"אישור" המודל
- $k -$ מספר איטרציות מקסימלי

w היא ההסתברות שבהגרלה אקראית נגריל *inlier*. מכיוון שעלינו להגריל n דגימות להגדרת מודל, ההסתברות שכולן יהיו *inliers* היא w^n. ההסתברות שלפחות אחד מהנקודות היא *outlier* היא $(1 - w^n)$. אם נגריל שוב ושוב נקודות, ההסתברות שאחרי k הגרלות ב<u>כל</u> ההגרלות יהיה *outlier* היא $(1 - w^n)^k$, ולכן ההסתברות שלפחות באחת ההגרלות נמצא את המודל הנכון היא:

$$p = 1 - (1 - w^n)^k$$

מכאן ניתן לגזור את מספר האיטרציות הדרוש כך שלפחות בהגרלה אחת נמצא את הפרמטרים של המודל:

$$k = \frac{\log(1 - p)}{\log(1 - w^n)}$$

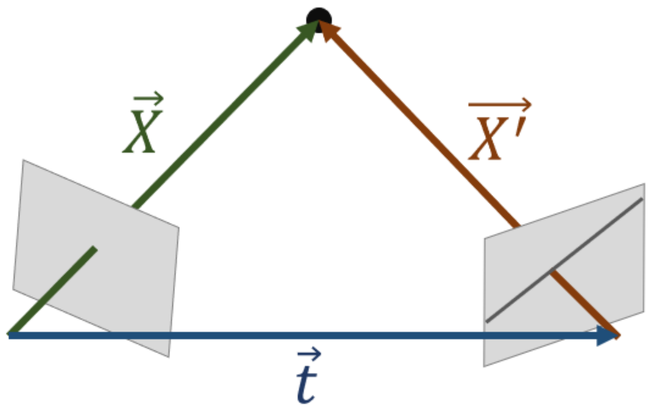
Essential matrix and Fundamental matrix

נחזור לבעיית *structure from motion*. אנו עוסקים בשתי מצלמות מכויילות, אשר מערכת הצירים של המצלמה הראשונה מיושרת עם "העולם" – מטריצה אקסטרינזית מהצורה:

$$\left(\boldsymbol{R} \mid \vec{t}\right) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

והמעבר למערכת הצירים של המצלמה השנייה לא ידוע. נקודה $\vec{X} = (x, y, z)^T$ במערכת הצירים של מצלמה ראשונה מועתקת ל-$\vec{X}' = (x', y', z')^T$ במערכת הצירים של המצלמה השנייה ע"י הטרנספורמציה:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \boldsymbol{R} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \vec{t}$$



תמונה 35: נקודה במערכת הצירים של שתי המצלמות. ניתן לראות כי הטרנספורמציה היא מהצורה $\vec{X}' = \boldsymbol{R}\vec{X} + \vec{t}$.

נחלץ אילוץ מהמשוואה:

$$\begin{aligned}
&\vec{X}' = \boldsymbol{R}\vec{X} + \vec{t} \Rightarrow \\
&\vec{t} \times \vec{X}' = \vec{t} \times \left(\boldsymbol{R}\vec{X} + \vec{t}\,\right) = \vec{t} \times \boldsymbol{R}\vec{X} + \vec{t} \times \vec{t} = \vec{t} \times \boldsymbol{R}\vec{X} \Rightarrow \\
&\vec{X}'^{T} \cdot \left(\vec{t} \times \vec{X}'\right) = \vec{X}'^{T} \cdot \left(\vec{t} \times \boldsymbol{R}\vec{X}\right) \Rightarrow \\
&0 = \vec{X}'^{T} \cdot \left(\vec{t} \times \boldsymbol{R}\vec{X}\right) = \vec{X}'^{T}\left(\left[\vec{t}\,\right]_{\times}\boldsymbol{R}\right)\vec{X}
\end{aligned}$$

כאשר $\left[\vec{t}\,\right]_{\times}$ בשורה האחרונה הוא ההצגה המטריצית של מכפלה וקטורית:

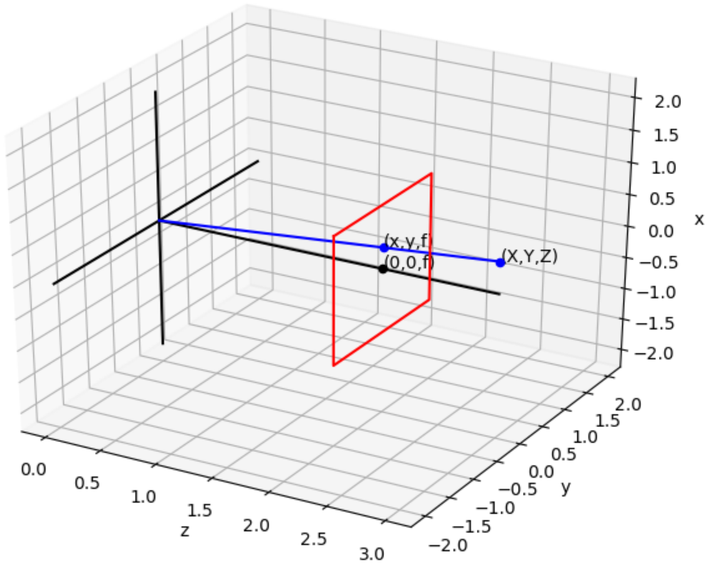
$$\vec{a} \times \vec{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [\vec{a}]_{\times} \cdot b = \vec{c}$$

המטריצה $\boldsymbol{E} = \left(\left[\vec{t}\,\right]_{\times}\boldsymbol{R}\right)$ נקראת Essential matrix, והיא מגדירה את המשטח שנוצר על ידי שלושת הנקודות: המוקד של שתי המצלמות והנקודה הנצפית.

המשוואה שקיבלנו מניחה שהמצלמות מכוילות, אך נרצה לקבל משוואה אשר לא מניחה זאת. במקום להשתמש בנקודות $\vec{X}, \vec{X}'$ ניתן להשתמש ב"קרניים", בווקטורים $(x_c, y_c, f), (x_c', y_c', f)$ אשר מצביעים לאותו כיוון. המשוואה שתתקבל תהיה:

$$(x_c', y_c', f)\boldsymbol{E}\begin{pmatrix} x_c \\ y_c \\ f \end{pmatrix} = 0$$

המשוואה מגדירה מעבר של נקודה כפי שמופיעה במערכת הקואורדינטות של מצלמה ראשונה למערכת המצלמה השנייה. נרצה ללכת עוד שלב אחד, ולהגיע להעתקה של **פיקסל** מתמונה אחת לשנייה.



תמונה 36 : וקטור (x, y, f) במישור המצלמה מצביע לכיוון הנקודה (X, Y, Z)

את הוקטורים הנ"ל נוכל לבטא בקואורדינטות התמונה כפי שראינו לעיל בפרק על קואורדינטות הומוגניות :

$$\vec{X}_{im} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{C}\vec{X}_c \Rightarrow$$
$$\vec{X}_c = \boldsymbol{C}^{-1}\vec{X}_{im}$$

כאשר $\boldsymbol{C}$ היא המטריצה עם הפרמטרים האינטרינזיים.

תזכורת :

$$\vec{x}_{im} \cong \begin{pmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ f \end{pmatrix}$$

נחזור למשוואה :

$$0 = \overrightarrow{X_c}'^{T} \boldsymbol{E}\overrightarrow{X_c} = \overrightarrow{X_{\imath m}}'^{T} \boldsymbol{C}'^{-1}\boldsymbol{E}\boldsymbol{C}^{-1}\overrightarrow{X_{\imath m}} = \vec{X}_{im}'^{\ T}\boldsymbol{F}\vec{X}_{im}$$

המטריצה $\boldsymbol{F} = \boldsymbol{C}'^{-1}\boldsymbol{E}\boldsymbol{C}^{-1}$ היא ה-Fundamental matrix, והיא מגדירה את הקשר בין הנקודה המופיעה בשתי התמונות **בקואורדינטות התמונה (פיקסל לפיקסל)** ללא צורך בכיול המצלמה וידיעת הפרמטרים האינטרינזיים, מכיוון שפרמטרי הכיול $\boldsymbol{C}, \boldsymbol{C}'$ מוכלים בתוכה.

נשים לב שנקודה ספציפית (u, v) תיצור על ידי המשוואה $(u', v', 1)\boldsymbol{F}(u, v, 1)^T = 0$ קו ישר מהצורה : $au' + bv' + c = 0$ בתמונה השנייה, כפי שלמדנו בעבר כי נקודה מועתקת לקו אפיפולרי.

את מטריצה $\mathbf{F}$ נמצא כפי שפתרנו בעבר. עבור זוג נקודות אחד ניתן להגיע לביטוי :

$$(u', v', 1)\boldsymbol{F}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \vec{x}'\boldsymbol{F}\vec{x}^T = \vec{x}' \cdot \begin{pmatrix} \vec{x}\vec{F}_1 \\ \vec{x}\vec{F}_2 \\ \vec{x}\vec{F}_3 \end{pmatrix} = u'\vec{x}\vec{F}_1 + v'\vec{x}\vec{F}_2 + \vec{x}\vec{F}_3 \ = \ (u'\vec{x}, v'\vec{x}, \vec{x}) \begin{pmatrix} \vec{F}_1^T \\ \vec{F}_2^T \\ \vec{F}_3^T \end{pmatrix}$$
$$= \vec{0}$$

כאשר $\vec{F}_i$ היא השורה ה-i במטריצה $\boldsymbol{F}$.

המשוואה היא עבור זוג אחד של נקודות. מכיוון שיש תשעה משתנים עם שמונה דרגות חופש נצטרך שמונה זוגות, ונקבל מטריצה מהצורה:

$$\begin{pmatrix} u_1'\vec{x}_1, v_1'\vec{x}_1, \vec{x}_1 \\ \vdots \\ u_8'\vec{x}_8, v_8'\vec{x}_8, \vec{x}_8 \end{pmatrix} \begin{pmatrix} \vec{F}_1^T \\ \vec{F}_2^T \\ \vec{F}_3^T \end{pmatrix} = \vec{0}$$

ניתן לפתור את המשוואה כפי שהיא ע״י הו״ע בעל הע״ע הקטן ביותר, אך לעיתים פתרון זה אינו יציב, מכיוון שהערכים במטריצה יכולים להיות גדולים (סדר גודל של $1e6$ מכיוון שערכי הפיקסלים יכולים להיות גדולים). לכן פותח Normalized 8 points ALG. באלגוריתם זה מנורמלות הנקודות:

$$\bar{X} = \frac{1}{n}\sum_i \vec{X}_i$$

$$S = \frac{\sqrt{2}}{\frac{1}{n}\sum_i \sqrt{u_i^2 + v_i^2}}$$

$$\tilde{X} = S(\vec{X} - \bar{X}) = \begin{pmatrix} S & 0 & -S\bar{u} \\ 0 & S & -S\bar{v} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{L} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

כעת יש לפתור את ה-fundamental matrix עבור $\tilde{X}$:

$$\tilde{X}'^T \tilde{\boldsymbol{F}} \tilde{X} = 0$$

את משוואה זו פותרים כפי שמוצג לעיל:

$$\begin{pmatrix} \tilde{u}_1'\tilde{x}_1, \tilde{v}_1'\tilde{x}_1, \tilde{x}_1 \\ \vdots \\ \tilde{u}_8'\tilde{x}_8, \tilde{v}_8'\tilde{x}_8, \tilde{x}_8 \end{pmatrix} \begin{pmatrix} \tilde{F}_1^T \\ \tilde{F}_2^T \\ \tilde{F}_3^T \end{pmatrix} = \vec{0}$$

מוצאים את הו״ע בעל הע״ע הקטן ביותר, וזה הפיתרון ל-$\tilde{\boldsymbol{F}}$. מכאן דרך טרנספורמציה הפוכה:

$$\boldsymbol{F} = \boldsymbol{L}'^{\boldsymbol{T}} \tilde{\boldsymbol{F}} \boldsymbol{L}$$

הערות

- דנו במקרה בו יש שתי מצלמות. במקרה כללי בו קיימות j מצלמות, יש למצוא מינימום לביטוי $argmin_P \left(\sum_j \sum_i \left|\vec{x}'^j_i - P^j \vec{x}_i\right|^2\right)$. בעיה זו נקראת boundle adjustment.

שערוך disparity בעזרת Block Matchinng

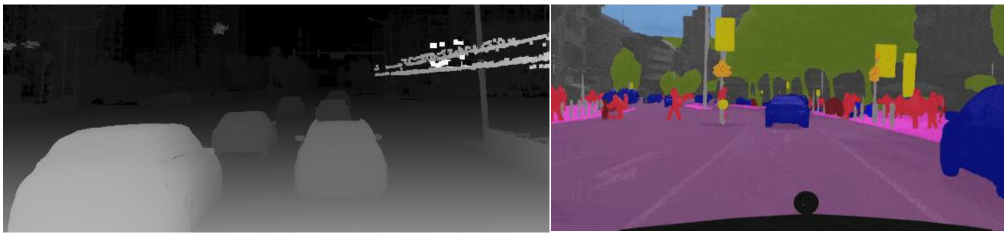
עד כה, דנו במדידת טווח מנקודות עניין אותן ניתן לזהות משתי המצלמות ולהתאים ביניהן. השלב הבא הוא מציאת תמונה בה לכן פיקסל נתון ערך שממנו ניתן לשערך מרחק. ראינו כי:

$$Z \propto \frac{1}{d}$$

כאשר d הוא ה-disparity, כלומר המרחק בין הקואורדינטות של זוג נקודות שזוהו בכל אחת מהתמונות. למשל אם נמצא זוג נקודות תואמות בקואורדינטות $\vec{x} = (100{,}50), \vec{x}' = (110{,}50)$ ה-disparity יהיה שווה ל-10.

הבעיה הכללית של תיוג כל פיקסל נקראת pixel labeling problem, דוגמאות לבעיות כאלה:

- Disparity map (מציאת disparity לכל פיקסל)
- Semantic segmentation (תיוג כל פיקסל בתמונה ל-class מסוים)
- Image denoising (מציאת הרעש שהתווסף לכל פיקסל על מנת לסנן אותו)



תמונה 37: סגמנטציה סמנטית (בצד ימין) ו-disparity map (בצד שמאל)

הנחות:

- שתי מצלמות מכוילות
- תמונות שעברו rectification – כלומר שהקווים האפיפולריים אופקיים.

הסיבה לדרישה השנייה היא חישובית –קל יותר למצוא פיקסלים תואמים כאשר ידוע כי החיפוש של נקודה מתאימה נדרש להתבצע רק על פני שורה אחת בתמונה עם ערך y קבוע וידוע.

הקושי בבעיה הוא שמספר הנקודות התואמות דליל מאוד יחסית למספר הפיקסלים בתמונה (סדר גודל של עשרות או מאות נקודות תואמות לעומת מיליוני פיקסלים שקשה להתאים ביניהם).

הדרך הפשוטה לפתור את הבעיה היא להגדיר פונקציית מטרה מהצורה:

$$SSD(\vec{p}, d) = \sum_{\Delta x} \sum_{\Delta y} \left(I_1(x + \Delta x, y + \Delta y) - I_2(x + d + \Delta x, y + \Delta y)\right)^2$$

כאשר:

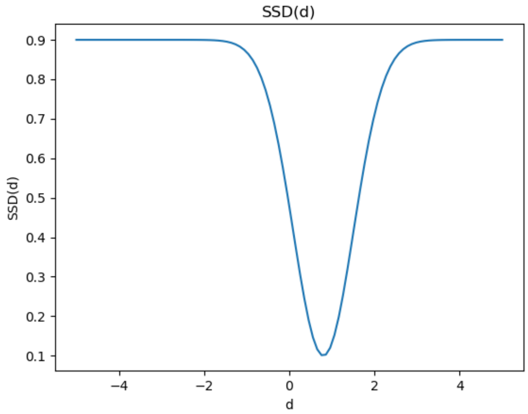
$\vec{p} = (x, y)$ – נקודה בתמונה

$\Delta x, \Delta y$ – חלון חיפוש בסביבות הנקודה

$I_1(x, y), I_2(x, y)$ – זוג התמונות

d – disparity

במשוואה זו, אנו מזיזים חלון בגודל $(\Delta x, \Delta y)$ על פני ציר ה-x. עבור כל מרחק d של החלון מהפיקסל ההתחלתי מתקבל ערך $SSD(p, d)$ אשר מכמת את ההתאמה של ה-patch ל-disparity הנוכחי. במצב האידיאלי נקבל פונקציה מהצורה:



תמונה 38 : דוגמה לפונקציית $SSD(\vec{p}, d)$.

נקודת המינימום בפונקציה תתקבל עבור ערך ה-disparity המתאים.

הערה

- בפתרון כזה קיים tradeoff בגודל החלון, חלון קטן מידי יושפע מרעש ועלול שלא לתפוס את המאפיינים האזוריים של ה-patch ואילו חלון גדול מידי עלול להיות מושפע מנקודות בסביבה הרחוקה של הנקודה אשר לא בהכרח חולקות את אותו ה-disparity.

פתרון זה שקול למציאת Maximum Likelihood (ML), מכיוון שהוא בודק עבור האזור את ערך ה-disparityשיניב את ההסתברות הגבוהה ביותר.

ביצועי האלגוריתם אינם טובים, על מנת לשפר את הביצועים ניתן להשתמש בהנחות נוספות על התמונה כפי שנראה להלן.

שערוך disparity בעזרת dynamic programming (Viterbi ALG)

האלגוריתם של ויטרבי מוסיף לבעיה הנחה שמסייעת בפתרון – ההנחה כי לרוב העומק לא משתנה בצורה דרסטית מפיקסל לפיקסל. פונקציית המטרה של ויטרבי:

$$L'(x,d) = C(x,d) + \min\Big(L'(x-1,d), L'(x-1,d-1)+p_1, L'(x-1,d+1)+p_1, \min_i L'(x-1,i)+p_2\Big)$$

הרעיון מאחורי האלגוריתם הוא למצוא מסלול מקצה אחד של התמונה לקצה השני אשר כולל את פונקציית המטרה של SSD אך מוסיף אילוץ על כמות השינויים ב-disparity. עבור בחירה של ערך disparity זהה לזה של הפיקסל הסמוך המחיר לא יעלה, בכל פעם שהאלגוריתם "יבחר" להוסיף ל-disparity ± 1 הוא ייאלץ לשלם בנוסף מחיר של p_1 ובכל פעם שה-disparity תשתנה ביותר מ-1 המחיר יהיה p_2.

באופן עקרוני פתרון של כל המסלולים האפשריים הוא קשה מידי, אך ויטרבי משתמש בעקרון ה- back tracking, ובעזרת dynamic programing פותר את הבעיה בצורה יעילה. הרעיון שמאחורי האלגוריתם הוא שאם ידועה לנו הדרך עם המחיר c_i הנמוך ביותר מפיקסל i עד סוף התמונה, הדרך עם המחיר הנמוך ביותר מפיקסל $i-1$ עד סוף התמונה שווה למחיר הנמוך ביותר ממנו לפיקסל x_i ועוד c_{i-1}.

$$\min\big(cost(x_{i-1} \to x_w)\big) = \min\big(cost(x_{i-1} \to x_i)\big) + \min\big(cost(x_i \to x_w)\big)$$

האלגוריתם מתחיל מסוף התמונה ומחשב את המחיר: $L'(x=w,d) = \mathrm{SSD}(\vec{p},d)$. לאחר מכן מתקדם צעד אחד אחורה ומחשב את כל האפשרויות: שמירה על ערך disparity זהה, הוספה/החסרה של 1, הוספה/החסרה של יותר מ-1. כעת ידועה לנו ה"דרך הקצרה ביותר" מהפיקסל לפני אחרון עד הסוף. ניתן להמשיך כך עד תחילת התמונה וכך לחשב את המסלול עם המחיר הנמוך ביותר.

הבעיה העיקרית באלגוריתם היא שהוא לא מכיל אילוצים בין פיקסלים משורות שכנות אלא רק על פיקסלים באותה שורה. זה גורם לך שייווצרו ארטיפקטים של "streaks". על מנת להתגבר על הבעיה נדרש להוסיף אילוצים על קשרים של פיקסלים בשורות שונות. הסיבה שלא ניתן להרחיב כך את האלגוריתם היא שעלולים להיווצר מסלולים מעגליים, אך האלגוריתם מבוסס HMM (Hidden Markov Model) אשר מניח שמסלולים כאלה לא קיימים.

תמונה 39: תופעת "streaks" בתמונת disparity.

Markov Random Field (MRF)

האלגוריתם השלישי שנציג מתגבר על בעיית חוסר התלות בין שורות, פיתרון זה מבוסס MRF. על מנת להסביר את העיקרון נדון בבעיה מקבילה – סינון רעש מתמונה (image denoising).

נתונה תמונה בינרית (ערכי פיקסלים $\{0,1\}$), שמתווסף לה רעש. המטרה היא שערוך התמונה ללא הרעש. האלגוריתם מבוסס Markov Random Field (MRF).

נתונים:

- אוסף מדידות $S = \{1, \dots, N\}$ – המדידות הם ערכי הפיקסלים בתמונה הרועשת.
- אוסף משתנים אקראיים $\vec{X}_{1,\dots,N} = \{x_i\}_{i=1}^{N}$ – התפלגות הרעש.
- קבוצת השכנויות (neighborhood structure), קליקות (cliques) $\{\mathcal{N}_n\}_{n=1}^{N}$ – מגדיר את היחס בין פיקסל לשכנים. למשל שכנות של 4/8 פיקסלים סמוכים.

המשמעות של מרקוביות בתמונה דו מימדית היא שערך פיקסל (כמשתנה אקראי) תלוי רק בשכניו ולא בכל התמונה:

$$\Pr\left(x_n | x_{S\setminus n}\right) = \Pr\left(x_n | x_{\mathcal{N}_n}\right)$$

לכן, פונקציית ההסתברות עבור כל התמונה היא:

$$\Pr\left(\vec{X}_{1,\dots,N}\right) = \frac{1}{Z}\prod_{j=1}^{J}\phi_j\left[\vec{X}_{C_j}\right] = \frac{1}{Z}\exp\left\{\log\left(\prod_{j=1}^{J}\phi_j\left[\vec{X}_{C_j}\right]\right)\right\}$$
$$= \frac{1}{Z}\exp\left\{-\sum_{j=1}^{J}-\log\left(\phi_j\left[\vec{X}_{C_j}\right]\right)\right\} = \frac{1}{Z}\exp\left\{-\sum_{j=1}^{J}\psi_j\left(X_{C_j}\right)\right\}$$

כאשר:

- $\frac{1}{Z}$ הוא גורם נרמול כדי שסכום הפונקציה יהיה 1.
- ϕ היא פונקציית פוטנציאל (אי שלילית) אשר מגדירה את ההסתברות עבור אוסף דגימות.
- $\vec{X}_{C_j}$ הוא אוסף הדגימות (פיקסלים) בקליקה C_j.
- C_j הוא כל הקליקות האפשרויות, כלומר כל אוספי הפיקסלים האפשריים עבור מבנה השכנות הנתון.
- הפונקציה $\psi(\cdot)$ מוגדרת: $\psi(\cdot) = -\log\left(\phi(\cdot)\right)$

למשל עבור הדוגמה הבאה:

תמונה 40: דוגמה ל-*Markov Random Field*.

בדוגמה זו קיימות ארבעה קבוצות (קליקות) - $(x_1,x_2),(x_2,x_3),(x_3,x_4),(x_4,x_1)$. פונקציית ההסתברות של ארבעת המשתנים האקראיים היא:

$$\Pr\left(\vec{X}_{1,\dots,N}\right)=\frac{1}{Z}\phi_{12}(x_1,x_2)\phi_{23}(x_2,x_3)\phi_{34}(x_3,x_4)\phi_{41}(x_4,x_1)$$

בהנחה כי אין הבדל בפונקציית ההסתברות בין הקליקות וכי היא מוגדרת:

$$\phi_{mn}(0,0)=1,\phi_{mn}(0,1)=0.1$$
$$\phi_{mn}(1,0)=0.1,\phi_{mn}(1,1)=1$$

ניתן לחשב את ההסתברות עבור כל קליקה לקבל את כל אחת מהאופציות, וכך להגיע לפונקציה $\Pr\left(\vec{X}_{1,\dots,N}\right)$. את Z ניתן לחשב ע"י סכום על כל הקומבינציות האפשריות (0000,0001,...,1111).

עבור הבעיה של סינון הרעשים נניח כי ההתפלגות של הרעש מתנהגת כך:

$$\Pr(y_n|x_n=0)=Bern_{x_n}(p)$$
$$\Pr(y_n|x_n=1)=Bern_{x_n}(1-p)$$

כלומר ההסתברות לשגיאה ($x_n=0$ ו-1 $y_n=$ או להיפך) עבור כל פיקסל היא p. על מנת לחשב את ההסתברות לקבל תמונה $x_1,\dots,x_N$ בהינתן דגימות $y_1,\dots,y_N$ נשתמש בבייס:

$$\begin{aligned}\Pr(x_1,\dots,x_N|\,y_1,\dots,y_N)&=\frac{\Pr(\,y_1,\dots,y_N|x_1,\dots,x_N)\Pr(x_1,\dots,x_N)}{\Pr(y_1,\dots,y_N)}\\&=\frac{\prod_{n=1}^{N}\Pr(y_n|x_n)\Pr(x_1,\dots,x_N)}{\Pr(y_1,\dots,y_N)}\end{aligned}$$

קיימות שתי דרכים לפתור את ה-*MRF*:

1. *Loopy Belief Propagation* – הרחבה של *backtracking* למקרים בהם קיימים מסלולים מעגליים.
2. *Graph Cut*

בהתבסס על *MAP – Maximum A posteriori Likelihood* ניתן לשערך את התמונה ע"י חישוב ההסתברויות:

$$\begin{aligned}\hat{X}_{1,\dots,N}&=argmax_{x_1,\dots,x_N}[\Pr(x_1,\dots,x_N|\,y_1,\dots,y_N)]\\&=argmax_{x_1,\dots,x_N}[\Pr(y_1,\dots,y_N|x_1,\dots,x_N)\Pr(x_1,\dots,x_N)]\\&=argmax_{x_1,\dots,x_N}\left[\left(\prod_{n=1}^{N}\Pr(y_n|x_n)\right)\Pr(x_1,\dots,x_N)\right]\\&=argmax_{x_1,\dots,x_N}\left[\log\left(\left(\prod_{n=1}^{N}Pr(y_n|x_n)\right)Pr(x_1,\dots,x_N)\right)\right]\\&=argmax_{x_1,\dots,x_N}\left[\sum_{n=1}^{N}(\log(Pr(y_n|x_n))+\log(Pr(x_1,\dots,x_N))\right]\\&=argmin_{x_1,\dots,x_N}\left[\sum_{n=1}^{N}-\log(Pr(y_n|x_n))+\sum_{mn\in C}-\log(Pr(x_1,\dots,x_N))\right]\end{aligned}$$

$$= argmin_{x_1,\dots,x_N}\left[\sum_{n=1}^{N} -\log(Pr(y_n|x_n)) + \sum_{mn\in C} -\log\left(\frac{1}{Z}\exp\{-\psi(x_m,x_n,\theta)\}\right)\right]$$
$$= argmin_{x_1,\dots,x_N}\left[\sum_{n=1}^{N} -\log(Pr(y_n|x_n)) + \sum_{m,n} \psi(x_m,x_n,\theta)\right]$$
$$= argmin_{x_1,\dots,x_N}\left[\sum_{n=1}^{N} U_n(x_n) + \sum_{m,n\in C} V_{mn}(x_m,x_n)\right]$$

קיבלנו סכום של שני ביטויים, $U_n(x_n)$ (*unary*) אשר מייצג את התלות של x_n במדידה y_n, ו- $V_{mn}(x_m,x_n)$ (*binary*) אשר מייצג את הקשר בין הפיקסלים הסמוכים.

הסיבה שהיינו מתקשים לפתור את הבעיה בעזרת *MAP* היא שה-*prior* ($\Pr(x_1,\dots,x_N)$) דורש את ההסתברות המשותפת של כל המשתנים. הנחת ה-*MRF* אפשרה להניח חוסר תלות בין רוב המשתנים וכך לפשט את הבעיה.

שערוך disparity בעזרת Graph cut

כעת נשוב לבעיה המקורית – שערוך תמונה *disparity* בעזרת תמונת סטריאו. האלגוריתם מבוסס *graph-cut* – אלגוריתם מתחום תורת הגרפים.

נתון גרף לא כיווני (*undirected graph*) $G = (V, E)$. עבור חלוקה של G לשתי קבוצות $G = S \cup \bar{S}$, נגדיר חיתוך של גרף (*cut*) :

$$C = \{(u, v) \mid u \in S, v \in \bar{S}\}$$

כלומר C הוא אוסף הקשתות שעוברות מקבוצה S ל-$\bar{S}$. נגדיר משקל של חיתוך C :

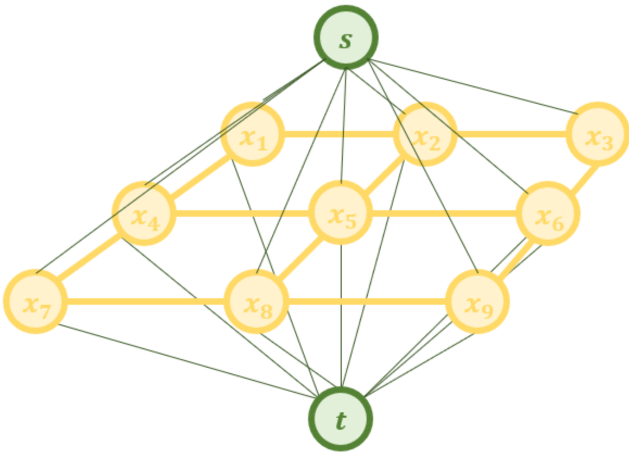
$$weight(C) = \sum_i (u_i, v_i)$$

Graph cut מחפש את החיתוך המינימלי של הגרף :

$$argmin_C\big(weight(C)\big)$$

קיים משפט תורת הגרפים – *min cut max flow theorem* האומר כי החיתוך המינימלי של גרף שווה לזרם (*flow*) המקסימלי שיכול לעבור בגרף. למשל אם הגרף הוא אוסף של כבישים עם קדקוד המוגדר כ-*source* וקדקוד המוגדר כ-*drain*, ומתבצע חיתוך מינימלי של הגרף בצורה שה-*source* בקבוצה אחת וה-*drain* בקבוצה השנייה, ונניח כי משקלי הקשתות מגדירות את כמות הרכבים המקסימלית האפשרית ליחידת זמן דרך הקשת, כמות הרכבים המקסימלית ליחידת זמן אשר יכולה לעבור מה-*source* ל-*drain* שווה לסכום הקשתות השייכות ל-C כאשר C הוא החיתוך המינימלי.

בתור התחלה נניח כי ישנן רק שני ערכי *disparity* אפשריים. במקרה כזה ניתן להציג את הבעיה בצורה הבאה :



תמונה *41* : ייצוג תמונה של 3×3 כגרף.

בייצוג זה s, t מייצגים את שני ה-*class*-ים השונים (שני ערכי *disparity*) הקשתות מ-s ומ-t ימושקלו כמשקל ה-*unary* – $U_n(x_n)$ והקשתות בין שני פיקסלים ימושקלו כמשקל ה-*binary* - $V_{mn}(x_m, x_n)$. נשים לב שפתרון ה-*graph-cut* הוא בדיוק הפתרון שחיפשנו לעיל :

$$\hat{X}_{1,\dots,N} = argmin_{x_1,\dots,x_N}\left[\sum_{n=1}^{N} U_n(x_n) + \sum_{m,n\in C} V_{mn}(x_m, x_n)\right]$$

ולכן פתרון ה-graph-cut ייתן את ערכי ה-disparity האופטימליים ע"פ MAP. ניתן להוסיף למשוואה איבר רגולריזציה λ כך שיהיה ניתן לשלוט על ה-trade-off בין האיבר ה-unary לאיבר ה-binary :

$$\hat{X}_{1,\dots,N} = argmin_{x_1,\dots,x_N}\left[\sum_{n=1}^{N} U_n(x_n) + \lambda \sum_{m,n\in C} V_{mn}(x_m, x_n)\right]$$

כעת נותר להרחיב את הפתרון למספר גדול יותר של ערכי disparity. על מנת לעשות זאת ניתן לעבור בצורה איטרטיבית על כל ערכי ה-disparity ולחשב את התיוג של כל פיקסל אחד אחרי השני כאשר ב-class אחד נמצא ה-disparity הנוכחי ובשני כל השאר:

$$\begin{aligned}
&1.\ Start\ with\ some\ label\ assignment.\\
&2.\ Loop\ over\ labels: \alpha \in \{1,\dots,l\}.\\
&3.\ Solve\ binary\ MRF\ problem\ for\ \alpha.\\
&4.\ If\ E\ (enengy)\ drops\ update\ E.\\
&5.\ If\ E\ did\ not\ change: Quit\\
&\quad else: GOTO\ (2)
\end{aligned}$$

הערה:

- אלגוריתם ה-MRF היה ה-state of the art בבעיית מציאת ה-disparity עד לעידן ה- deep learning.

באופן כללי, על מנת לפתור בעיה ע"י Graph Cut נדרש להגדיר את הערכים הבאים:

1. מרחב התיוגים
2. ה-unary term
3. ה-binary term

דוגמאות:

<u>Disparity Map</u>:

1. מרחב התיוגים: $\mathcal{L} = \{1,\dots,d\}$ – כמספר ערכי ה-disparity הרצויים.
2. Unary term: $U(x_n) = \sum_{(u,v)\in w_n} \left|Y_{left}(u,v) - Y_{right}(u - x_n, v)\right|$
3. Binary term: יכול להיות $V(x_n, x_m) = [x_n \neq x_m]$ או לחילופין $V(x_n, x_m) = \frac{1}{|Y_n - Y_m| + \epsilon}$

$V(x_n, x_m) = [x_n \neq x_m]$ נקרא *potts model* והוא מכיל בתוכו מחיר על השמה של שני פיקסלים שכנים בערכי *disparity* שונים.

$V(x_n, x_m) = \frac{1}{|Y_n - Y_m| + \epsilon}$ נקרא *edge preserving cost* והוא מאלץ פיקסלים שכנים בעלי צבע דומה להיות עם אותו ערך *disparity*.

<u>*Interactive Image Segmentation*</u>:

במקרה זה נתונה תמונה ונדרש להפריד אובייקט מהרקע. המשתמש מתייג חלק קטן מהפיקסלים כאובייקט וחלק כרקע. במקרה זה נתוני ה-*graph cut* הם:

1. מרחב תיוגים: $\mathcal{L} = \{0,1\}$ – אובייקט או רקע.
2. *Unary term*: $U(x_n) = d(Y_n, GMM_{F/B})$
3. *Binary term*: $V(x_n, x_m) = \frac{1}{|Y_n - Y_m| + \epsilon}$

$GMM_{F/B}$ הוא *Gaussian Mixture Model* על מרחב הצבע לקבלת הסתברות על בסיס צבע הפיקסל מול צבע הפיקסלים המתוייגים.

תמונה *42* : דוגמה לשימוש ב-*Interactive Image Segmentation*. המשתמש מתייג חלק מהתמונה ואלגוריתם ממשיך את התיוג.

אלגוריתם Dynamic Programming משופר

הבעיה העיקרית ב-*Viterbi Algorithm* (*DP*) הייתה בעיית ה-*streaking*. על מנת להתגבר על הבעיה ניתן לשכלל את האלגוריתם, השיטה נקראת *SGM – Semi Global Matching*. הרעיון הוא שבמקום להריץ את ה-*Back tracking* לאורך ציר ה-*x* בלבד, ניתן להריץ אותו בכיוונים שונים (סדר גודל של 8-16 כיוונים שונים), כאשר בכל כיוון מחשבים את $C(x,d)$ לאורך ציר x בלבד כפי שהוצג, אך התקדמות ה- *back tracking* היא לאורך צירים אחרים. במקום:

$$\min\left(L'(x-1,d), L'(x-1,d-1)+p_1, L'(x-1,d+1)+p_1, \min_i L'(x-1,i)+p_2\right)$$

נריץ לאורך ציר אחר:

$$\min\left(L'(\vec{x}-\Delta\vec{x},d), L'(\vec{x}-\Delta\vec{x},d-1)+p_1, L'(\vec{x}-\Delta\vec{x},d+1)+p_1, \min_i L'(\vec{x}-\Delta\vec{x},i)+p_2\right)$$

כאשר $\vec{x}$ הוא מיקום הפיקסל בצעד הנוכחי ו-$\Delta\vec{x}$ הוא כיוון ההתקדמות. ערכי הפונקציה נסכמים עבור כל המסלולים לכל פיקסל ולכל ערכי ה-disparity האפשריים:

$$S(\vec{p},d) = \sum_r L'_r(\vec{p},d)$$

כאשר r הוא המסלול, וה-disparity הסופית מתקבלת על ידי:

$$Disparity(\vec{p}) = argmin_d\left(S(\vec{p},d)\right)$$

כך נבחרת ה-disparity אופטימלית. היתרון בדרך זו הוא שמתווספים אילוצים כך שלא קיים "כיוון אופייני" כמו באלגוריתם המקורי.

דרך זו יעילה משמעותית מ-graph cut ומגיעה לביצועים דומים.

נקודות עניין – Local Invariant Features

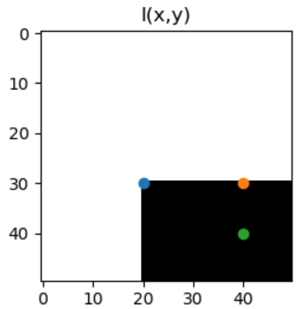
עד עכשיו השתמשנו בנקודות תואמות נתונות, ולא דנו כיצד ניתן למצוא נקודות כאלה. כעת נדון בבעיה של מציאת נקודות תואמות. נקודות עניין, או פיצ'רים, הם כלי יעיל מאוד לאפיון תמונות ועבודה איתן, ראינו דוגמה לשימוש עבור תמונות עומק אך נראה עוד מספר אפליקציות לכלי זה.

נפריד את הדיון לשני נושאים:

1. מציאת "נקודות מעניינות" בתמונה – detection.
   אלגוריתם קלאסי למטלה זו הוא Harris corner detector.
2. התאמת נקודות נתונות בין שתי תמונות – description.
   אלגוריתם קלאסי – SIFT (Scale-invariant feature transform).

Harris

ראשית יש להגדיר מהי "נקודה מעניינת". נקודה הנמצאת באזור "שטוח" בעל צבע שאינו משתנה קשה להגדרה, מכיוון שקשה להגדיר היכן היא בדיוק בתמונה השנייה. נקודה הנמצאת על שפה ניתנת להגדרה על ציר אחד (הציר המאונך לשפה) אך קשה לאיתור בציר השני (כיוון השפה). נקודה הנמצאת על פינה ניתנת להגדרה בצורה קלה מכיוון שאנו יודעים היכן היא בכל אחד מהצירים.

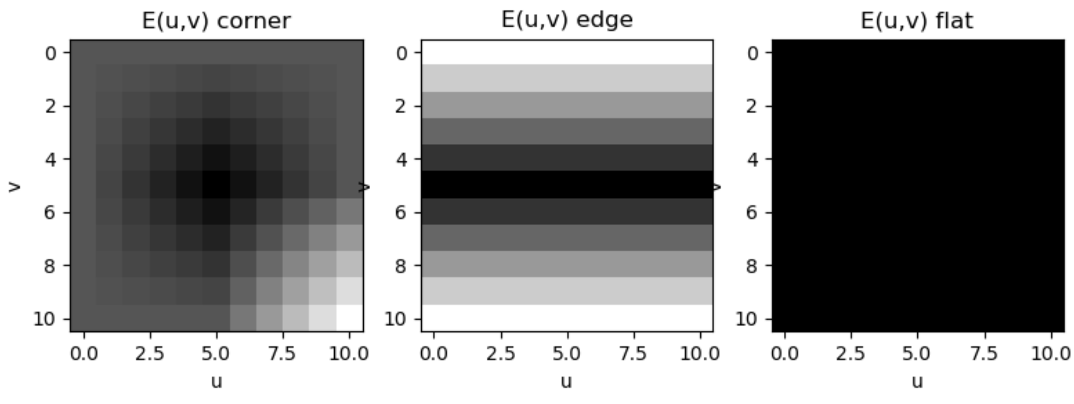


תמונה 43: תמונה $I(x,y)$ עם שלוש נקודות: איזור חלק, איזור של שפה ואיזור של פינה.

נגדיר "משטח שגיאה" (error surface) – פונקציה עבור חלון w מסביב לפיקסל מסויים:

$$E_w(u,v) = \sum_{(x,y)\in w} [I(x+u,y+v) - I(x,y)]^2$$

הפונקציה תקבל ערכים נמוכים באזורים בהם $I(x,y)$ חלקה, אך באזורים בהם התמונה משתנה $E(u,v)$ תקבל ערכים גבוהים, מכיוון ש-$I(x+u,y+v)$ – התמונה המוזזת – תהיה שונה מ-$I(x,y)$.



תמונה 44: הפונקציה $E(u,v)$ בכל אחת מהנקודות שבתמונה הקודמת. באזור חלק הפונקציה תהיה נמוכה, באזור של שפה היא תהיה נמוכה לאורך הציר של השפה וגבוהה בציר המאונך לה, ובאזור של פינה היא תקבל ערכים גבוהים בשני הצירים.

נרצה לבדוק באיזה כיוון יש שינוי בפונקציה, אם יש שינוי בשני כיוונים – נגדיר את הנקודה כ״פינה״. לצורך כך נמצא קירוב טיילור מסדר ראשון :

$$I(x+u,y+v)\cong I(x,y)+\frac{dI}{dx}(x,y)u+\frac{dI}{dy}(x,y)v=I(x,y)+\left(I_x,I_y\right)\binom{u}{v}$$

$$E(u,v)=\sum_{(x,y)\in w}[I(x+u,y+v)-I(x,y)]^2\cong\sum_{(x,y)\in w}\left[I(x,y)+\left(I_x,I_y\right)\binom{u}{v}-I(x,y)\right]^2$$

$$=\sum_{(x,y)\in w}\left[\left(I_x,I_y\right)\binom{u}{v}\right]^2=(u,v)\sum_{(x,y)\in w}\begin{pmatrix}I_x^2 & I_xI_y\\ I_xI_y & I_y^2\end{pmatrix}\binom{u}{v}$$

הערה

- אנו מחפשים כיוון שינוי מינימלי, אך אם נבחר את הפתרון הטריוויאלי $(u,v)=(0,0)$ הוא יתן הערך המינימלי. כדי להתגבר על העניין, כפי שעשינו בעבר, אנו מאלצים את (u,v) להיות עם נורמה בגודל קבוע, וכך הפתרון יהיה הכיוון עם השינוי המינימלי..

הערך המינימלי של הפונקציה הוא הע״ע הנמוך ביותר ל-$\sum_{(x,y)\in w}\begin{pmatrix}I_x^2 & I_xI_y\\ I_xI_y & I_y^2\end{pmatrix}$, הוא מגדיר את עוצמת הגרדיאנט בכיוון בו השינוי מינימלי. אם נראה שהערך נמוך – סימן שהנקודה שנבחרה היא באיזור שטוח או על גבי שפה. אם הערך גבוה – סימן שבשני הכיוונים השינוי גדול – כלומר הנקודה היא פינה.

<u>האלגוריתם בפסאודוקוד :</u>

$Input: I(x,y)-image$
$k-w\ kernel$
$Output: R-response\ image$

$I_x,I_y=gradient\big(I(x,y)\big)$

$I_{xx}=I_x^2$
$I_{yy}=I_y^2$
$I_{xy}=I_x\cdot I_y$

$S_{xx}=I_{xx}*k$
$S_{yy}=I_{yy}*k$
$S_{xy}=I_{xy}*k$

$R=\frac{1}{2}\left(S_{xx}-S_{yy}\right)-\sqrt{4S_{xy}^2+\left(S_{xx}-S_{yy}\right)^2}$

לאחר שמתקבלת מפת העירור R (response map), היא מועברת בסף, וכל איזור שעובר את הסף מוגדר כנקודת עניין.

ע״מ לקבל אלגוריתם יותר רובסטי, Harris הציע לבחור את R בצורה הבאה :

$$R=\frac{\lambda_{min}\lambda_{max}}{\lambda_{min}+\lambda_{max}}=\frac{\det(H)}{trace(H)}$$

במצב כזה, אם שני הערכים העצמיים באותו סדר גודל נקבל ערך גבוה, ואם אחד גדול ואחד קטן (שפה) נקבל ערך נמוך.

למעשה משתמשים בקירוב:

$$R = \lambda_{min}\lambda_{max} - \alpha(\lambda_{min} + \lambda_{max})^2$$
$$\alpha = \ 0.04 \sim 0.06$$

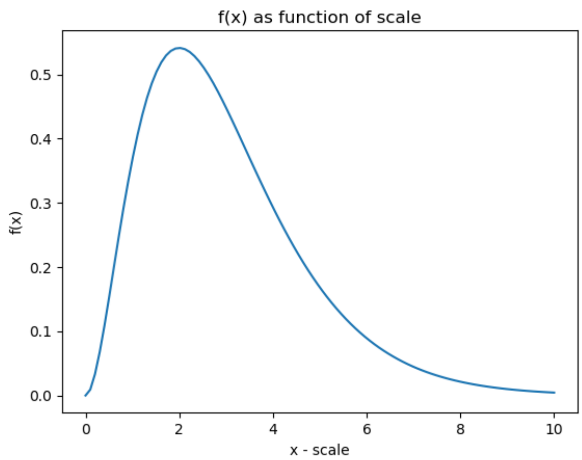
הערות

- חלון w שעבורו מחושבת $E(u,v)$ הוא בד"כ ריבועי או גאוסי.
- הצגנו חישוב של נקודות עניין עבור *scale* אחד, אך חדות או טשטוש התמונה יכול להשפיע במקרה כזה על מפת העירור. לכן למעשה נהוג להשתמש במספר רזולוציות של התמונה (או לחילופין בגדלים שונים של w) ולמצוא את הערך המקסימלי ביניהם.

Scale-Space

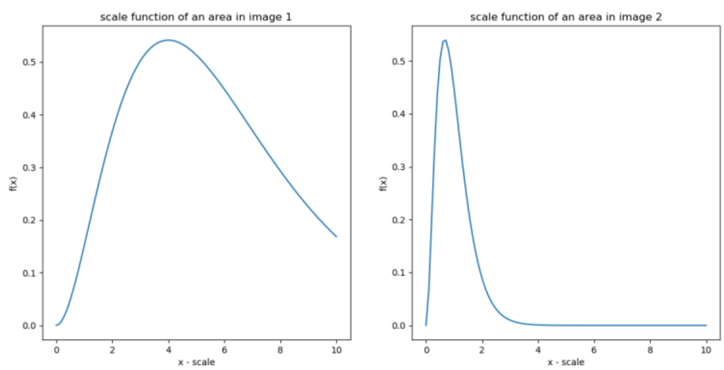
כפי שהזכרנו בהערה הקודמת, הבעיה העיקרית באלגוריתם של *Harris* היא חוסר האינווריאנטיות לשינוי ב-*scale*. כאשר נרצה למשל להתאים נקודות בין שני פריימים עוקבים בוידאו האלגוריתם יצליח מכיוון שהגודל של האובייקטים לא משתנה משמעותית מפריים לפריים, אך כנראה שלא נצליח להתאים בין נקודות מתמונות של סצנה שצולמה משתי זוויות שונות. הסיבה היא כי ככל הנראה פינה ספציפית שתזוהה כנקודת עניין בתמונה אחת לא תזוהה בתמונה שניה בגלל שינוי הסקאלה, ולכן יקשה עלינו להתאים נקודות בין שתי תמונות כאלה.

פתרון לבעיה זו הוא שימוש במרחב ה-*scale-space*. נרצה לבצע *automatic scale selection* - בחירת *scale* אוטומטי לכל אזור בתמונה. נמצא פונקציה $f(x)$ אשר מתנהגת בצורה הבאה:



תמונה 45: פונקציה $f(x)$ אשר תלויה ב-*scale* של הפיצ'ר.

פונקציה כזו תאפשר לנו למצוא *scale* אחיד לפיצ'רים מתמונות שונות. עבור כל אזור בתמונה נבדוק איזה *scale* מביא את הפונקציה למקסימום וכך נקבע את ה-*scale* האופייני לנקודה.



תמונה 46: פונקציית ה-*scale* עבור אותו פיצ'ר בשתי תמונות.

עבור כל אחת מהתמונות נבחר את ה-*scale* האופטימלי כ-*scale* אשר מביא למקסימום את הפונקציה. לצמד ה-$(x, y, scale)$ נקרא *blob*. נראה כי לפלסיאן של גאוסיאן של תמונה מקיים את התנאי.

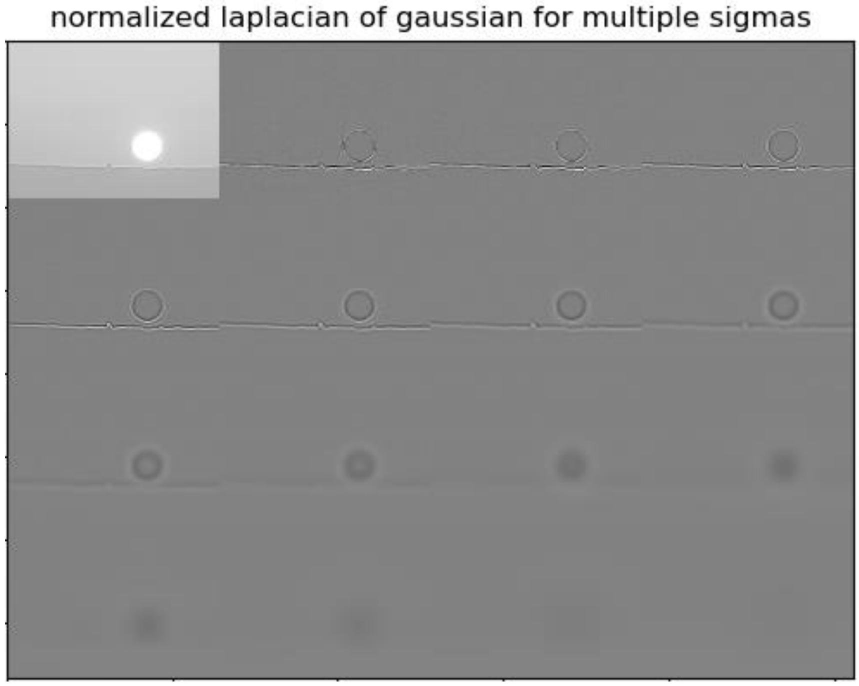
בהינתן תמונה $f(x,y)$, וגאוסיאן $g(x,y,\sigma)$ נגדיר *scale-space*:

$$L(x,y,\sigma) = g(x,y,\sigma) * f(x,y)$$

נשים לב שעבור כל σ נקבל תמונה מטושטשת ברמה אחרת. עבור $\sigma = 0$ נקבל את התמונה המקורית וככל שהוא גדל התמונה מיטשטשת. $L(x,y,\sigma)$ יהיה הייצוג של התמונה ב-*scale-space*. על מנת למצוא את ה-*blobs* נמצא את הלפלסיאן של ה-*scale-space* שהגדרנו – *LoG – Laplacian of gaussian* של התמונה:

$$\nabla_n^2 L(x,y,\sigma) = \sigma^2 \left(\frac{d^2 L(x,y,\sigma)}{dx^2} + \frac{d^2 L(x,y,\sigma)}{dy^2} \right)$$

כאשר ∇_n^2 הוא הלפלסיאן המנורמל, ו-σ הוא הנירמול. הסיבה לנירמול היא שמכיוון שנלקחים גאוסיאנים בגדלים שונים ערכי הפונקציה עבור σ שונים יהיה תלוי גם בגאוסיאן ולא רק בתמונה. הנירמול מבטל את ההשפעה הזו.



תמונה 47: ניתן לראות את מוצא הלפלסיאן של הגאוסיאן עבור ערכי σ שונים. ניתן לראות כי איזור מרכז השמש מקבל מינימום עבור σ גדול יחסית, ואילו פיצ'רים קטנים על קו האופק יקבלו ערכי קיצון ב-σ נמוך.

לאחר שמתקבלות תמונות עבור ערכי σ שונים יש למצוא נקודות קיצון (מינימום או מקסימום כתלות בשאלה אם ה-*blob* בהיר או כהה) גם בצירי x, y וגם בציר ה-σ. נקודת קיצון (x,y,σ) תייצג *blob* בנקודה x, y עם *scale* של σ.

SIFT

לאחר שמצאנו את ה-*scale* הנכון (בעזרת *LoG*) ואת נקודות העניין (בעזרת *Harris*) נותר להגדיר *descriptor* לנקודות כדי שנוכל להשוות אותם לנקודות בתמונות אחרות. אלגוריתם ידוע לפעולה זו הוא ה-*SIFT*.

<u>שלבי האלגוריתם:</u>

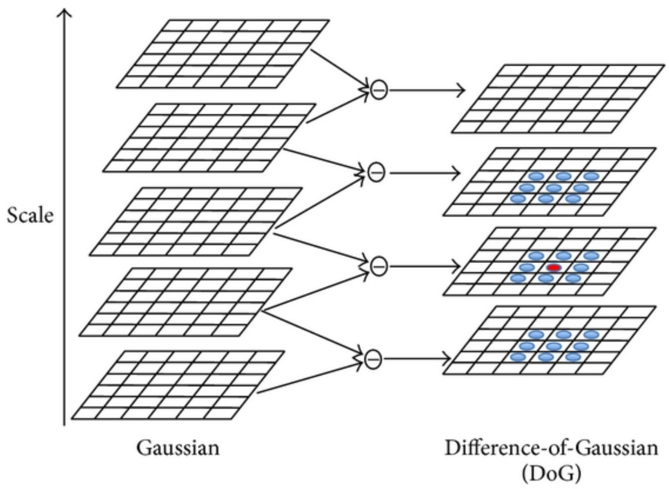
1. מעבר למרחב ה- *scale-space* ומציאת ה-*scale* הנכון.
2. *Fine tuning* של הנקודות שנמצאו (הוספת שני משטחים אלכסוניים ב-*scale space* על מנת להגיע לדיוק תת-פיקסלי).
3. מציאת הפינות ע"י *Harris* או אלגוריתם דומה.
4. יצירת *descriptor* לכל נקודה.

DoG

על מנת לחסוך בחישובים, במקום להשתמש ב- *Laplacian of Gaussian* מחושב ה- *Difference of Gaussian* - $\left(G(x,y,k\sigma)-G(x,y,\sigma)\right)*f(x,y)$. ניתן לראות כי:

$$DoG=\left(G(x,y,k\sigma)-G(x,y,\sigma)\right)*f(x,y)=G(x,y,k\sigma)*f(x,y)-G(x,y,\sigma)*f(x,y)$$

שיטה זו מאפשרת לחסוך את חישוב הלפלסיאן ע"י חישוב $G(x,y,k\sigma)*f(x,y)$ עבור ערכי k שונים וחיסור k^i מ-k^{i-1}.



תמונה 48: חישוב ה-*DoG*. הנקודה האדומה מייצגת נקודה קיצון גם בצירי (x,y) וגם בציר ה-scale.

SIFT Descriptor

תהליך האלגוריתם:

1. "גזירת" *bounding box* מסביב ל-*blob* ב-*scale* שחושב.
2. חלוקת ה-*bounding box* ל-4×4 חלקים.
3. חישוב היסטוגרמה של 8 *bins* של הגרדינטים עבור כל אחד מ-16 החלקים (כלומר 8 קבוצות של גרדיאנטים בטווחים: $[0-45°,45°-90°,\dots,315°-360°]$).

עבור כל *blob* נקבל וקטור בממד $4\times4\times8=128$ (4×4 חלקים ועבור כל חלק היסטוגרמה של 8 *bins*). וקטור זה מייצג את ה-*blob*. על מנת למצוא את ה-*blob* התואם בתמונה שניה, ניתן לחפש את ה-*blob* אשר ה-*descriptor* שלו הוא הקרוב ביותר ל-*descriptor* שלו. על מנת למדוד מרחק בין הווקטורים נהוג להשתמש בנורמת L_2.

נשים לב כי:

- האלגוריתם מחלקת את התמונה ל-4×4, ולכן יש מידע מרחבי שמשתמר (אם כי לא הרבה).
- האלגוריתם מתייחס רק לגרדיאנטים, לכן הוא אינווריאנטי לבהירות התמונה.
- האלגוריתם מחלק את התמונה לאיזורים, ולכן "מקריב" דיוק מרחבי ע"מ לקבל מידע יותר מדויק על התפלגות הגרדיאנטים.
- יש עוד פרטים טכניים שלא הרחבנו, למשל, שכלול של גודל הגרדיאנטים בנוסף לכיוון, ומשקול למרחק ממרכז ה-$blob$.
- האלגוריתם אינו אינווריאנטי לסיבוב. קיים מימוש אשר מתגבר על הבעיה אך הוא פחות רובסטי.

BRIEF

Descriptor נוסף הוא ה-*BRIEF*. באלגוריתם זה, במקום לחלק את התמונה לאיזורים מוגרלות n נקודות $\{\vec{x}_i\}_{i=1}^{n} = \{(x_i, y_i)\}_{i=1}^{n}$ ומחושב הוקטור:

$$\left(\mathbf{1}_{f(x_2,y_2)>f(x_1,y_1)}, \dots, \mathbf{1}_{f(x_n,y_n)>f(x_{n-1},y_{n-1})}\right)$$

וקטור זה ייצג את ה-blob. ניתן לבחור את מספר הנקודות וכך לקבוע את גודל הווקטור. גודל אופייני הוא 128.

הערות

- ייצוג זה הוא מהיר יותר (לא דורש לעבור על כל ה-patch אלא רק על n נקודות בכל תמונה) ודורש פחות מקום (128 ביטים במקום 128 בתים).
- הרעיון ב-descriptor זה דומה ל-SIFT בכך שהוא מסתכל רק על ההפרש בין הערכים בתמונה ולכן אינווריאנטי לבהירות התמונה.

חיפוש תמונה - "Video Google"

בשנת 2003 פורסם מאמר בשם *Video Google* (נכתב באוקספורד) אשר מתמודד עם הבעיה הבאה: בהינתן תמונה ומאגר תמונות, נרצה למצוא את התמונה הדומה ביותר לתמונה שבידינו מתוך המאגר. עבור דוגמאות דומות בעולם ה-*data* הטקסטואלי (למשל חיפוש מסמך או מאמר דומה מתוך מאגר של מסמכים) אלגוריתמים נפוצים בודקים את מספר המופעים של כל מילה במסמך ובעזרת ההיסטוגרמה של המילים, יוצרים "*ID*" המאפיין את המסמך. בצורה זו ניתן לחפש את המסמך בעל ה-*ID* הקרוב ביותר למסמך המבוקש. *Video Google* נכתב בהשראת אלגוריתמים אלו. הוא יוצר "מילון" של *descriptors* ומחשב את כמות ה"מילים" – כלומר הווקטורים המאפיינים –עבור כל תמונה ויותר מזהה אופייני לתמונה אשר מגדיר את המאפיינים שלה.

<u>תיאור האלגוריתם:</u>

- מציאת נקודות אופייניות (פיצ'רים) בעזרת *SIFT*.
- קידוד הנקודות בעזרת *K-means* (הרחבה להלן) ל-5,000 "מילים" – כלומר מציאת 5,000 וקטורי פיצ'רים במרחב הפיצ'רים של *SIFT* אשר יאפיינו עבור ה-*dataset* הקיים את התמונות הקיימות בו.

הערה

- אלגוריתם זה מתבסס על ספירה של מספר המופעים של כל "מילה" (נקודת עניין) בתמונה, ולכן הוא אינו משמר את המבנה המרחבי של התמונה.

האלגוריתם מחולק לשני שלבים – *offline* ו-*online*.

<u>שלב ה-*offline*:</u>

בשלב זה מתבצע מעבר על כל ה-*database* ונאספים כל הפיצ'רים הקיימים בתמונות. שלב זה הינו השלב "היקר" באלגוריתם מכיוון שמאגר תמונות טיפוסי מכיל *1K-100K* תמונות.

לאחר איסוף הפיצ'רים, מתבצע אלגוריתם *K-means* (מפורט בהמשך) עם 5,000 מרכזים ($k = 5{,}000$). המרכזים שנמצאו יוצרים את ה-*Universal Visual Dictionary* אשר יאפשר לאפיין את התמונות שבמאגר.

לאחר בניית המילון, מתבצע מעבר נוסף על כל התמונות שבמאגר, ומחושבת ההיסטוגרמה של מופעי "מילות המילון" בכל תמונה – כמה פעמים מופיע כל פיצ'ר בתמונה.

<u>שלב ה-*online*:</u>

מתקבלת תמונה חדשה ונדרש למצוא את התמונה הדומה לה ביותר.

מחושבות כל נקודות ה-*SIFT* של התמונה החדשה, הנקודות ממופות למרכזים המתאימים בעזרת ה-*K–means*, ונבנית היסטוגרמה של פיצ'רים בתמונה.

לאחר מכן האלגוריתם מחפש את התמונה מתוך המאגר בעלת ההיסטוגרמה הקרובה אליה ביותר.

על מנת לקצר את זמן החיפוש, ניתן לייצר בשלב ה-*offline* טבלת חיפוש בה עבור כל מרכז (כל "מילה") יישמרו התמונות בהן הפיצ'רים המתאימים למרכז זה מופיעות, טבלה זו נקראת *inverted file*. דוגמה:

Visual word	Images in database
#1	17, 53, 71, 10365, …
#2	48, 69, 852, …
⋮	⋮
#5,000	45, 90, 345, …

לאחר שמתקבלת תמונה חדשה ומחושבים הפיצ'רים הנמצאים בה ניתן לפנות לתמונות המתאימות בטבלה ולמצוא את התמונה עם ההיסטוגרמה הקרובה ביותר.

K-means

אלגוריתם ה-*K-means* הוא אלגוריתם לאישכול (*clustering*) – חלוקה של וקטורים נתונים במרחב.

<u>מרחב הכניסה:</u> וקטורים מהצורה $\vec{x}_i \in \boldsymbol{R^d}$ $\{\vec{x}_i\}_{i=1}^{n}$

<u>מרחב היציאה:</u> $\{C_j\}_{j=1}^{k}$

האלגוריתם ממפה כל וקטור x_i ל-*class* C_j ספציפי. פונקציית המטרה שהאלגוריתם מחפש לה מינימום היא:

$$J(C_1, \dots, C_k; \{\vec{x}_i\}_{i=1}^{n}) = argmin_{C_1,\dots,C_k}\left[\sum_{j=1}^{k}\sum_{i\in C_j}\left|\vec{x}_i - C_j\right|^2\right]$$

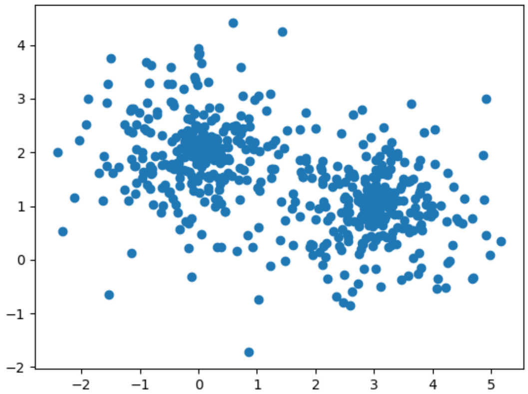
כלומר, אנו מחפשים נקודות C_j כך שסכום ריבועי המרחקים של כל הוקטורים $\vec{x}_i$ המשויכות ל-*class* C_j יהיה מינימלי.

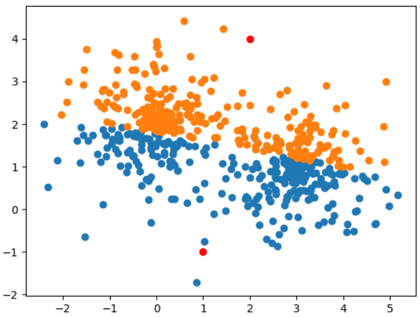
<u>תיאור האלגוריתם:</u>

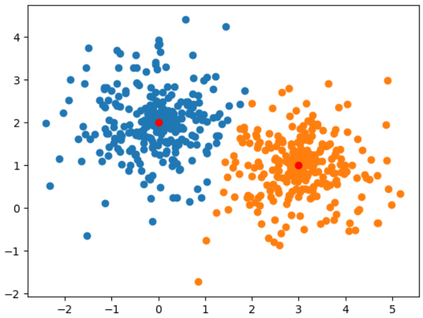
$Input: \{\vec{x}_i\}_{i=1}^{n} \quad \vec{x}_i \in \boldsymbol{R^d}$

$Output: \{C_j\}_{j=1}^{k}$

1. $Initialize\ C_1, \dots, C_k$
2. $Assign\ x_i\ to\ closest\ C_j$
3. $C_j = \frac{1}{|C_j|}\sum_{i\in C_j} x_i$
4. $If\ error\ drops\ GOTO\ 2$







תמונה 49: למעלה – אוסף כל הוקטורים $\{\vec{x}_i\}_i$. למטה בצד ימין – שני מרכזים C_1, C_2 עבורם סכום המרחקים של כל הווקטורים המשויכים לכל מרכז מינימלי. למטה בצד שמאל – שני מרכזים עבורם פונקציית המטרה לא מינימלית.

נוכיח כי אכן האלגוריתם עושה מינימיזציה לפונקציית המטרה:

קל לראות כי שלב 2 מקטין את פונקציית המטרה.

נדרש להוכיח כי גם 3 מקטין את הפונקציה.

נגדיר *encoder* ו-*decoder*:

$Enc\colon R^d \to [1, \dots, k]$

$Dec\colon [1, \dots, k] \to R^d$

ה-*encoder* ממפה נקודה $\vec{x}_i$ ל-*class* המתאים j, וה-*decoder* ממפה חזרה ל-C_j.

נגדיר *distortion*:

$$distortion(x_i) = \sum_i \left(x_i - Dec\left(Enc(x_i)\right)\right)^2$$

כלומר, סכום ריבועי המרחקים של וקטור מהמרכז אליו הוא מקודד. נרצה למצוא מינימום לפונקציית המטרה:

$$\frac{dJ(C_1, \dots, C_k; \{\vec{x}_i\})}{dC_j} = \frac{dDistortion\left(x_i; x_i \in C_j\right)}{dC_j} =$$

$$\frac{d}{dC_j}\sum_i \left(x_i - C_j\right)^2 = -2\sum_{i \in C_j}\left(x_i - C_j\right) = 0 \Rightarrow$$

$$\sum_{i \in C_j}\left(x_i - C_j\right) = \sum_{i \in C_j} x_i - nC_j = 0 \;\Rightarrow C_j = \frac{1}{n}\sum_{i \in C_j} x_i = mean_{i \in C_j}(x_i)$$

כלומר, בהינתן שיוך קבוצת וקטורים x_i לקבוצה C_j, יתקבל מינימום לפונקציית המטרה אם כל מרכז C_j יהיה ממוצע כל הווקטורים המשויכים לו.

מאחר שהוכחנו כי שני השלבים 2,3 מקטינים את פונקציית המטרה הוכחנו כי האלגוריתם מקטין את הפונקציה. נותר להראות כי האלגוריתם מתכנס.

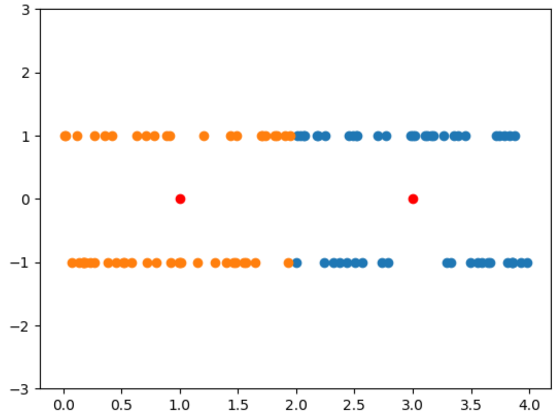
<u>הוכחה:</u>

1. פונקציית ה-*distortion* חיובית וחסומה מלמטה באפס.
2. הפונקציה תמיד יורדת (כפי שהוכחנו).
3. קיים מספר סופי של שיוך נקודות ל-*classes* מכיוון שמספר הנקודות וה-*classes* סופיים.

לכן האלגוריתם מתכנס לנקודת מינימום. אך לא מובטח כי המינימום יהיה המינימום הגלובלי.

הערות

- קיימות היוריסטיקות שונות לאיתחול ה-C_j, למשל הגרלת C_1 מתוך $\{x_i\}$, בחירת C_2 כ-x_i הרחוק ביותר מ-C_1 וכן הלאה.
- האלגוריתם מתאים ל-*cluster*-ים המפולגים רדיאלית סביב נקודה מרכזית, אך ישנן מקרים אחרים בהם האלגוריתם לא מתאים (ראה דוגמה להלן). במקרים כאלה ניתן למצוא מטריקות שונות להצגה של הווקטורים שבהן הם מפולגים רדיאלית ואז לבצע *K-means*.



תמונה 50 : דוגמה קלאסית למקרה בו *K-means* נכשל. ניתן לראות בבירור כי החלוקה "המתבקשת" היא לשני צבירים, אחד עליון ואחד תחתון, אך *K-means* יחלק אחרת.

Viola-Jones

אלגוריתם *V-J* הוא אלגוריתם למציאת פנים (*detection*). הוא היה האלגוריתם הדומיננטי לפני עידן ה-*deep learning*.

מרכיבי האלגוריתם:

1. אלגוריתם *AdaBoost*
2. מרחב מאפיינים (*feature space*) ייחודי
3. עקרון ה-*cascade of rejectors*

AdaBoost

AdaBoost הוא אלגוריתם סיווג מהצורה הבאה:

$$Input: \{\vec{x}_t, y_t\}_{t=1}^{N} \quad \vec{x}_t \in \boldsymbol{R}^d, y_t \in \{-1,1\}$$
$$Output: \ F(\vec{x}) \ \ s.t. \ \hat{y} = sign\big(F(\vec{x})\big)$$

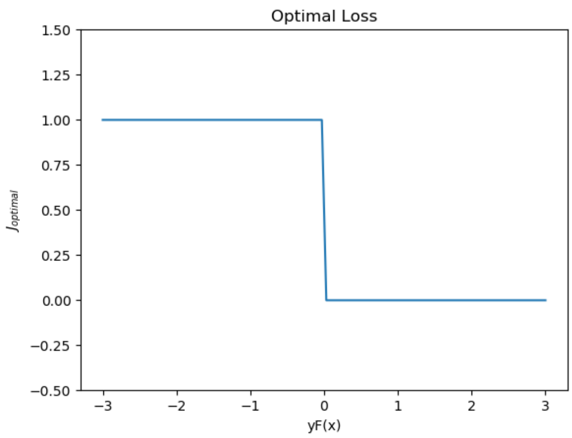
ה-*output* של האלגוריתם הוא פונקציה $F(\vec{x})$ המסווגת את ה-*input*. ערך חיובי של הפונקציה מעיד כי האלגוריתם מסווג $\hat{y} = 1$ וערך שלילי $\hat{y} = 0$. הפונקציה היא מהצורה:

$$F(\vec{x}) = \sum_{m=1}^{M} \alpha_m f_m(\vec{x})$$

כאשר הפונקציות $f_m(\vec{x})$ נקראות *weak-learner* והפונקציה $F(\vec{x})$ נקראת *Strong classifier*. המקדם α_m ממשקל את $f_m(\vec{x})$, ככל שהפונקציה אינפורמטיבית יותר לסיווג, α_m יקבל ערך גבוה יותר. על מנת למצוא את המקדמים האופטימליים יש להגדיר פונקציית *loss* שתכמת את איכות הסיווג. הפונקציה האופטימלית תהיה מהצורה הבאה:

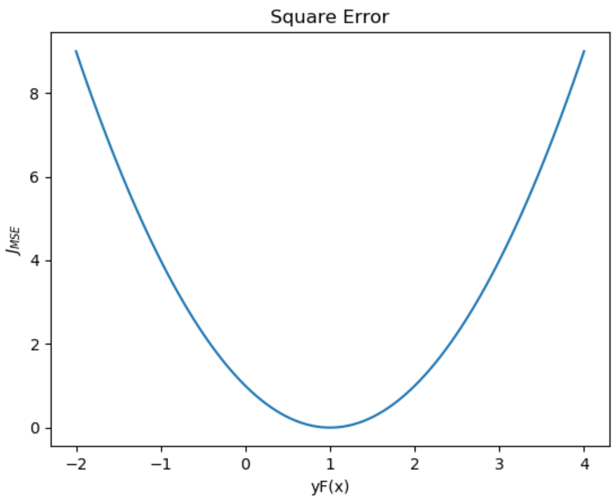
$$J\big(y_t, F(\vec{x}_t)\big) = \sum_{t=1}^{N} \delta\big(y_t \neq sign\big(F(\vec{x}_t)\big)$$

הפונקציה נותנת מחיר שווה עבור כל תיוג שגוי $(+1)$ וערך שווה עבור כל תיוג נכון (0). אך הפונקציה אינה גזירה ולכן קשה לעשות אופטימיזציה לפונקציה כזו.



תמונה 51: פונקציית *loss* אופטימלית

פונקציית ה-*square error* בה השתמשנו במספר מקרים לאופטימיזציה אינה מתאימה מכיוון שיש מקרים בהם היא נותנת מחיר גבוה גם לתיוג נכון.

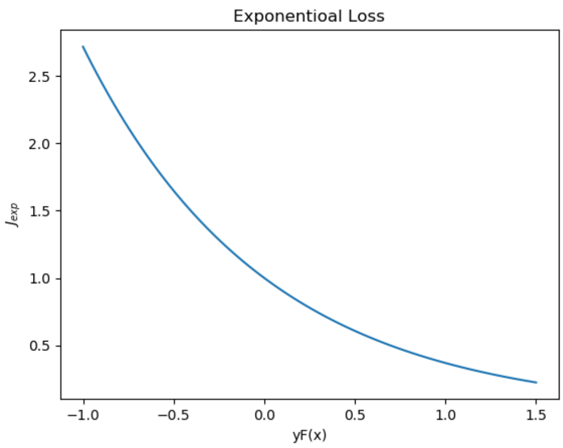


תמונה 52: *square error loss* מתמחרת גם תיוגים נכונים במחיר גבוה.

פונקציה מתאימה לבעיה זו היא ה-*exponential loss*:

$$J\big(y_t, F(\vec{x}_t)\big) = \sum_{m=1}^{M} e^{-y_t F(\vec{x}_t)}$$

פונקציה זו נותנת מחיר גבוה לשגיאות גדולות ומחיר נמוך במקרים בהם התיוג נכון. בנוסף פונקציה זו גזירה וחוסמת מלעיל את פונקציית ה-*loss* האופטימלית ושואפת אליה, ולכן היא יכולה לשמש *loss function*.



תמונה 53: *exponential loss*

לאחר שבחרנו פונקציית *loss* נגדיר את תהליך בניית $F(\vec{x})$. ניתן להגדיר את התהליך בצורה איטרטיבית כאשר:

$$F_m(\vec{x}) \leftarrow F_{m-1}(\vec{x}) + f_m(\vec{x})$$

בצורה כזו ניתן לעשות אופטימיזציה בצורה איטרציבית, כאשר בכל איטרציה מתבצעת אופטימיזציה על $f_m(\vec{x})$ בלבד:

$$\theta_m = argmin_\theta\left(\sum_{t=1}^{N} J(y_t, F_{m-1}(\vec{x}_t) + f_m(\vec{x}_t); \theta)\right)$$

כאשר θ_m הם הפרמטרים של המודל.

לכן נדרש למצוא מינימום לפונקציה:

$$J = \sum_t \exp\{-y_t(F_{m-1}(\vec{x}_t) + f_m(\vec{x}_t))\} = \sum_t \exp\{-y_t F_{m-1}(\vec{x}_t)\} \exp\{-y_t f_m(\vec{x}_t)\}$$

בחיי שאין לי מושג איך, אבל מסתבר שע"י קירוב טיילור מסדר ראשון לפונקציה ניתן להגיע לביטוי:

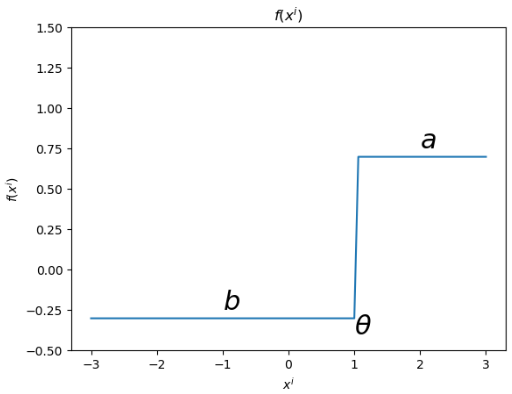
$$J \propto \sum_t e^{-y_t(F_{m-1}(\vec{x}))}(y_t - f_m(\vec{x}_t))^2$$

ניתן לראות כי הביטוי $w_m = e^{-y_t(F_{m-1}(\vec{x}))}$ קבוע ותלוי רק באופטימיזציה שהתבצעה בשלב $m-1$.
ככל שהמסווג $F_{m-1}(\vec{x})$ שגוי יותר בסיווג $\vec{x}$, ערכו של w_m יהיה גבוה יותר, ולכן ישפיע יותר על האופטימיזציה. הביטוי $(y_t - f_m(\vec{x}_t))^2$ הוא למעשה ה-*MSE* בין התיוג y_t לתחזית ה-*weak classifier*-$f_m(\vec{x}_t)$.

הפונקציות $f_m(\vec{x})$ בהן נעשה שימוש הן מהצורה:

$$f(x_t^i) = a[x_t^i \geq \theta] + b[x_t^i < \theta]$$

כאשר a, b, θ הם פרמטרים, i הוא המאפיין (feature) ו-t היא מספר הדגימה.



תמונה 54: ה-weak classifier של AdaBoost $f(x^i)$ עבור מאפיין x^i.

הפרמטרים a, b נבחרים באופן הבא:

$$a = E_w(y_t[x_t^i \geq \theta])$$

$$b = E_w(y_t[x_t^i < \theta])$$

בחירת הפרמטרים a, b מתבצעת ע"י מיצוע ($E_w(\cdot)$) של $y_t[x_t^i \geq \theta]$ עבור מאפיין i. בשלב ראשון בתהליך מתבצעת בחירה של המאפיינים i והספים θ, במחיר $O(dN)$ כאשר d הוא מספר המאפיינים ו-N מספר הדוגמאות, ובשלב שני מחושבים a, b ע"פ המשוואות הנ"ל. ככל שה-weak classifier נותן יותר פרדיקציות נכונות $a \to 1, b \to -1$, ולכן הוא יקבל משקל גבוה יותר. לאחר שמתבצעת למידת הפרמטרים מתקבלים פונקציות עם הפרמטרים:

$$f = (a, b, \theta, i)$$

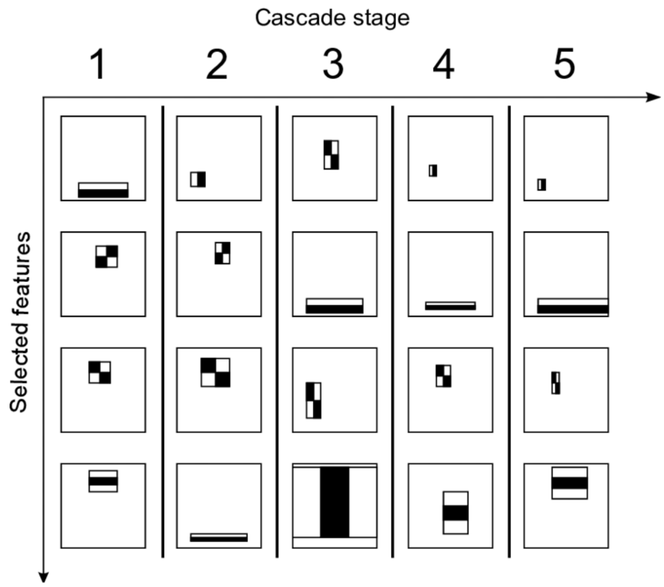
הערה:

- בתהליך הלמידה נבחרים המאפיינים התורמים ביותר לסיווג ונלמד גם הסדר שלהם כך שניתן יהיה להשתמש ב-*cascade of rejectors*.

בניית מרחב המאפיינים

ראינו כי נדרש להגדיר מרחב מאפיינים (feature space) עליו יופעלו פונקציות ה-f. האפשרות הפשוטה היא לעשות את האופטימיזציה על כל אחד מהפיקסלים של התמונה, אך מרחב זה אינו אינפורמטיבי מספיק כדי לאפיין מבנה של פנים.

המרחב אותו מגדיר V-J מורכב מ-160,000 מאפיינים: $\vec{x}_t \in \boldsymbol{R}^{160,000}$ אשר מוגדרים על גזיר (patch) בגודל 24×24 פיקסלים. המאפיינים הם סכום מכפלה של mask-ים שונים כפי שמופיע בציור, כאשר השטחים הלבנים ב-mask מייצגים ערכי $+1$ והשחורים -1.



תמונה 55: מאפיינים שונים של V-J. ניתן לראות מאפיינים שונים אשר נבחנים בשלבים השונים ב-cascade

על מנת להבין את שיטת הפעולה של חילוץ המאפיינים יש להקדים מונח: Integral Image. על מנת לסכום ערכי פיקסלים במלבן בתמונה, בצורה נאיבית הפעולה תעלה $O(n \times m)$ (כאשר n, m הם האורך והרוחב של הגזיר) מכיוון שהסכום כרוך במעבר על כל הפיקסלים שבגזיר. הפעולה הזו יקרה, ניתן על ידי עיבוד מקדים להגיע לאותה תוצאה ב-$O(1)$.

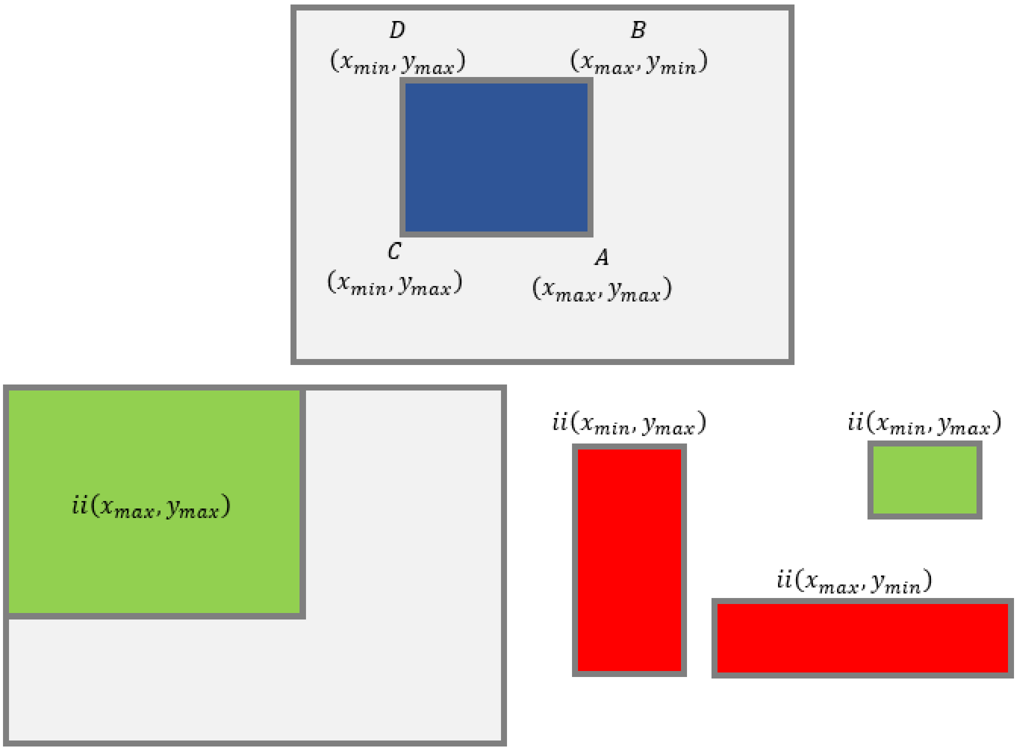
עבור תמונה נתונה $i(x, y)$ נגדיר מטריצות $S(x, y), ii(x, y)$ בצורה הבאה:

$$S(x, y) = S(x-1, y) + i(x, y)$$
$$ii(x, y) = ii(x, y-1) + S(x, y)$$

כאשר $S(x, y)$ היא סכום הפיקסלים של שורה y מ-$i(1, y)$ עד $i(x, y)$, ו-$ii(x, y)$ היא סכום הפיקסלים במלבן שהקצה העליון השמאלי ב-$i(1,1)$ והתחתון הימני ב-$i(x, y)$. על מנת לחשב את סכום הפיקסלים במלבן $(x_{min}, y_{min}, x_{max}, y_{max})$ מספיק לחשב את:

$$Area = ii(x_{max}, y_{max}) - ii(x_{max}, y_{min}) - ii(x_{min}, y_{max}) + ii(x_{min}, y_{max})$$

בצורה כזו, על ידי עיבוד מקדים של התמונה ניתן לחשב את השטח ב-$O(1)$.

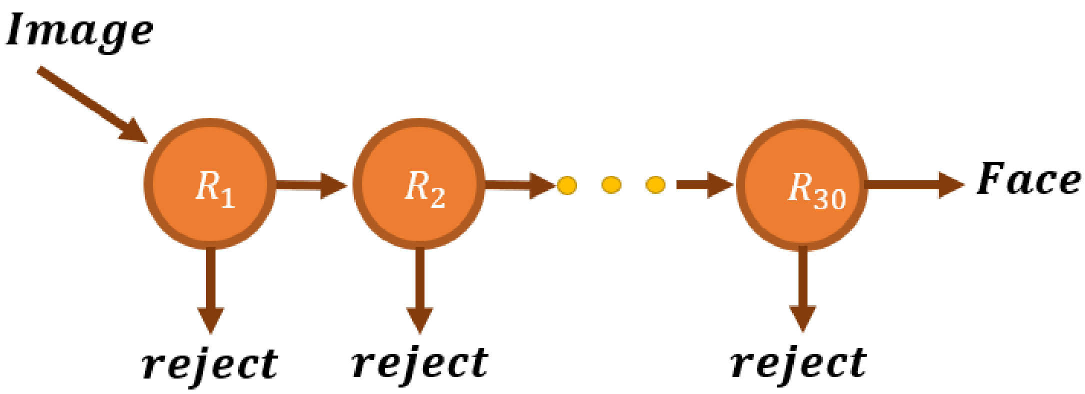


תמונה 56 : השטח הכחול שווה לסכום השטחים הירוקים פחות השטחים האדומים

ניתן ליישם את השיטה במרחב המאפיינים של $V\text{-}J$ ולחשב ביעילות את המאפיינים על ידי חישוב שטח פחות שטח.

Cascade of Rejectors

על מנת לייעל את תהליך הבדיקה, במקום לבדוק את כל המאפיינים על תמונה חדשה שמתקבלת בבת אחת, נבחנים המאפיינים אחד אחרי השני, כאשר ברגע שהמסווג יסווג תמונה כ"לא פנים" העיבוד יפסיק. בצורה כזאת, ניתן להפסיק את תהליך העיבוד ובכך לחסוך זמן עיבוד.



תמונה 57 : תהליך ה-*cascade of rejectors*

תיאור האלגוריתם:

<u>*Offline* :</u>

- מעבר על ה-*data* ויצירת *strong classifier* המיוצג ע"י 100 צמדים של (a, b, θ, i).

<u>*Online* :</u>

- קבלת תמונה.
- *Resize* לתמונה לגודל של 24×24.
- מעבר על ה-*weak classifiers* ע"פ הסדר, ב-*cascade*.

הערות

- אחת הנקודות החשובות באלגוריתם היא שהעבודה ב-*offline* ארוכה, שכן עוברים על 160,000 מאפיינים עבור כל דוגמה, אך באימפלמנטציה מספיקים בדרך כלל כ-100 מאפיינים על מנת לסווג בצורה טובה.
- ב-*offline* נבחרים ע"פ סדר, המאפיינים המשמעותיים ביותר לסיווג. אין הנחות מקדימות לגבי החשיבות, והסדר נקבע תוך כדי האימון. בשלב ה-*online* מתבצע מעבר על המאפיינים לפי הסדר שהוגדר וכך ניתן ליישם *cascade of rejectors*.

רשתות נוירונים

פרספטרון בודד

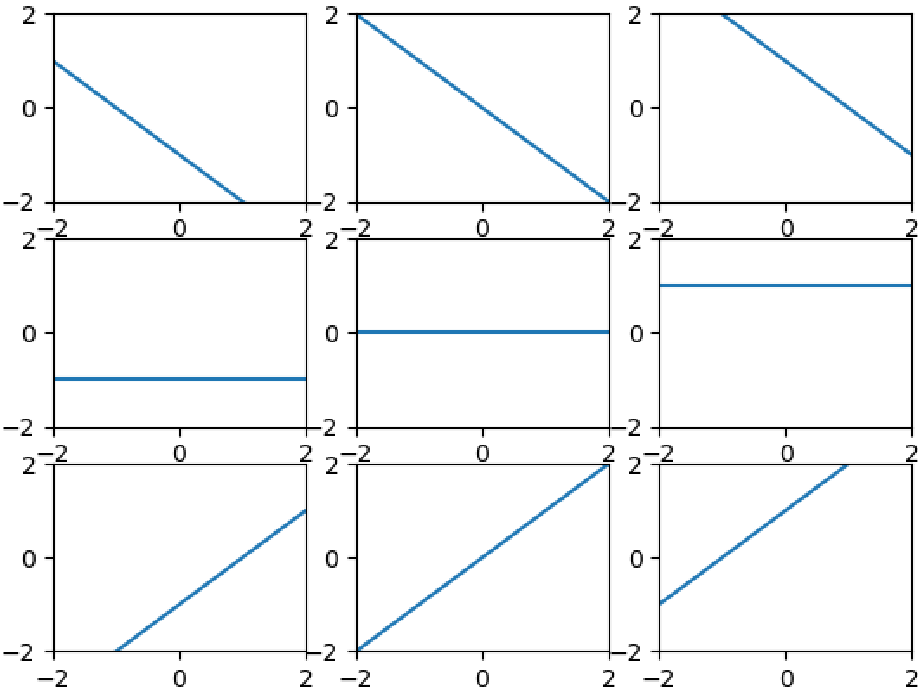
אלגוריתם הפרספטרון הוא אלגוריתם לסיווג לינארי של *data*. נתון *data* מהצורה:

$$\{\vec{x}_i, y_i\}_{i=1}^{n} \quad \vec{x}_i \in \boldsymbol{R}^d, y_i \in \{-1,1\}$$

כאשר $\vec{x}_i$ הוא וקטור ו-y_i הוא התיוג שלו. מטרת האלגוריתם היא לספק תיוג נכון y עבור וקטור חדש $\vec{x}$. הפרדיקציה של האלגוריתם מתבצעת בצורה הבאה:

$$h(\vec{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right)$$

במהלך "האימון" נבחרים הפרמטרים w_i וה-*threshold*. ניתן להגיע ע"י $h(\vec{x})$ לסיווג לינארי של המרחב.



תמונה 58: חלוקת המרחב של הפרספטרון. בכל שורה ישנו ערך $\vec{w}$ שונה ובכל עמודה ערך $threshold$ שונה.

על מנת לפשט את הכתיבה נהוג לקבוע כי $x_0 = 1$ ו-$w_0 = -threshold$ כך ש:

$$\left(\sum_{i=1}^{d} w_i x_i\right) - threshold = \left(\sum_{i=1}^{d} w_i x_i\right) + w_0 x_0 = \left(\sum_{i=0}^{d} w_i x_i\right) = \vec{w}^T \vec{x}$$

$$h(\vec{x}) = sign\left(\sum_{i=0}^{d} w_i x_i\right)$$

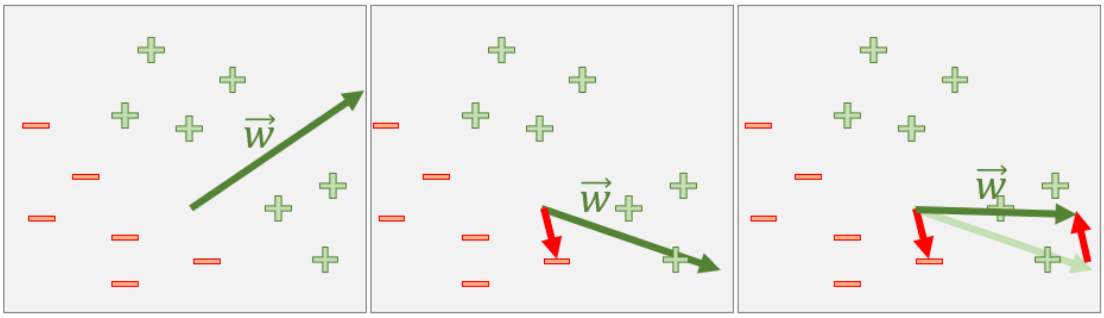
תיאור האלגוריתם:

$Input: \{\vec{x}_i, y_i\}_{i=1}^{n}$
$Output: \vec{w}$

1. $Initialize\ \vec{w}\ randomly$
2. $Pick\ a\ misclassified\ example:$
   $sign(\vec{w}^T \vec{x}_i) \neq y_i$
3. $Update\ \vec{w} \leftarrow \vec{w} + y_i \vec{x}_i$
4. $GOTO\ 2\ until\ no\ misclassified$
   $examples\ left$

<u>הסבר מזווית גיאומטרית:</u>

לאינטרפרטציה גיאומטרית ניתן לחשוב על $\vec{w}$ כווקטור אשר מסווג את הדגימות ע"פ מיקומם יחסית אליו. במקרה בו הזווית בינו לבין הדגימה לא מתאימה לסיווגה הנכון – $\vec{w}$ מתעדכן בהתאם.



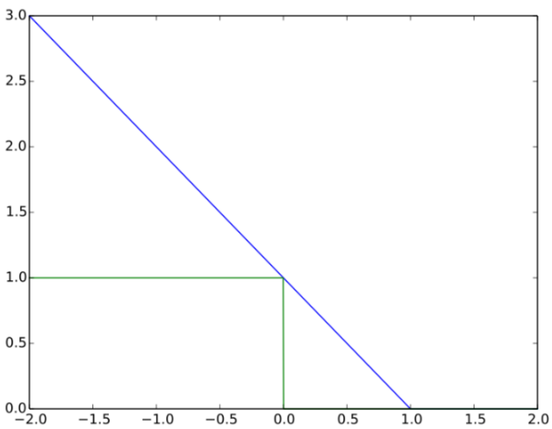
תמונה 59: בצד שמאל – וקטור $\vec{w}$ אחרי "אימון". באמצע – דוגמה לסיווג לא נכון עבור $\vec{w}$ שגוי. בצד ימין – תיקון האלגוריתם ל-$\vec{w}$.

<u>הסבר מזווית של אופטימיזציה:</u>

דרך נוספת להתבונן על האלגוריתם הוא בעזרת ה-loss function. פונקציית ה-loss עבור אלגוריתם זה היא ה-Hinge loss:

$$l(\vec{w}, \vec{x}, y) = \max(0, 1 - y\vec{w}^T \vec{x})$$

$$L(\vec{w}, \{\vec{x}_i, y_i\}_{i=1}^{n}) = \frac{1}{n}\sum_{i=1}^{n} l(\vec{w}, \vec{x}_i, y_i)$$



תמונה 60: Hinge loss ופונקציית המדרגה.

הפונקציה גזירה בכל התחום למעט עבור $x = 1$, וכן היא חוסמת מלעיל את פונקציית המדרגה. העדכון של $\vec{w}$ מתבסס על פונקציית ה-loss:

$$w^{(k)} \leftarrow w^{(k-1)} - \nabla_{\vec{w}} L_{Hinge}(y_i \vec{w}^T \vec{x}_i)$$

כאשר $\vec{x}_i, y_i$ היא דוגמה עם סיווג שגוי. במקרה בו האלגוריתם מסווג נכון, הביטוי $\vec{w}^T\vec{x}$ יהיה גדול מאפס כאשר $y = 1$ וקטן מאפס כאשר $y = -1$, לכן כאשר $y\vec{w}^T\vec{x} < 0$ ישנה שגיאה. ה-L_{Hinge} היא פונקציה של המכפלה $y\vec{w}^T\vec{x}$. ככל שהערך יותר שלילי השגיאה גבוהה יותר. מכיוון שהסיווג של $\vec{x}_i, y_i$ שגוי, $y\vec{w}^T\vec{x} < 0$ ולכן אנו דנים רק על החלק של הפונקציה שמשמאל לציר ה-y. בחלק זה הפונקציה גזירה לכן הגרדיאנט של הפונקציה הוא:

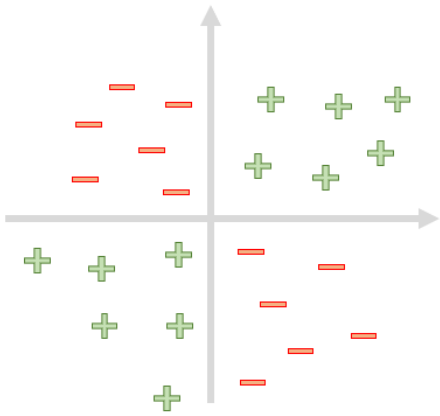
$$\nabla L_{Hinge} = -y_i \vec{x}_i$$

והעדכון של הפרמטרים הוא:

$$w^{(k)} \leftarrow w^{(k-1)} + y_i \vec{x}_i$$

בעיית ה-XOR

החיסרון של הפרספטרון הוא שהסיווג לינארי, ולא ניתן לייצר פונקציות סיווג מורכבות יותר. למשל לא ניתן יהיה לסווג data בפילוג הבא:



תמונה 61: פונקציית XOR, לא ניתן לסווג בעזרת מסווג לינארי.

הפתרון לבעיה הוא הרכבה של מספר פרספטרונים כך שיוכלו ליצור סיווג מורכב יותר. לצורך הדוגמה נראה איך ניתן לעשות זאת עבור פונקציית ה-XOR.

פונקציית XOR מוגדרת כך:

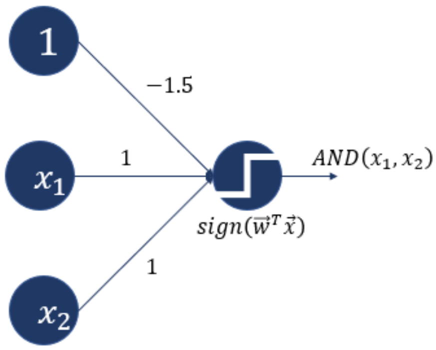
$$XOR(x_1,x_2)=\begin{cases}0 & ;x_1<0\ and\ x_2<0\\ 1 & ;x_1<0\ and\ x_2>0\\ 1 & ;x_1>0\ and\ x_2<0\\ 0 & ;x_1>0\ and\ x_2>0\end{cases}$$

ניתן להגיע לפונקציית XOR בעזרת פונקציות AND, OR, NOT בצורה הבאה:

$$XOR(h_1,h_2)=h_1\cdot\bar{h}_2+\bar{h}_1\cdot h_2$$

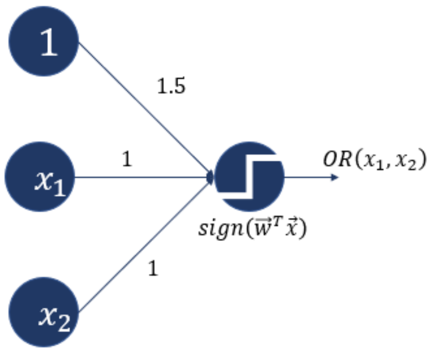
ולכן אם נצליח ליצור את פונקציות הבסיס בעזרת פרספטרון, נוכל בעזרתן ליצור XOR.

בהנחה כי $x_1,x_2\in\{-1,1\}$, ניתן לייצג פונקציית AND בצורה הבאה:



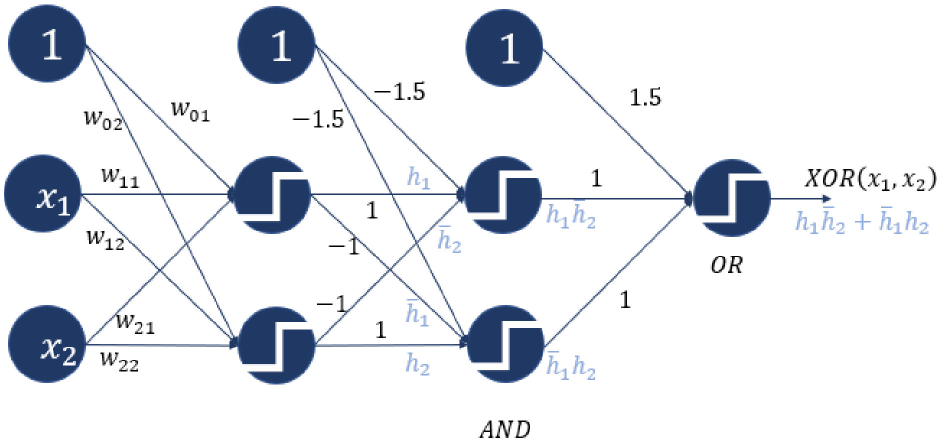
תמונה 62: פונקציית AND ממומשת בעזרת פרספטרון.

את פונקציית OR ניתן לייצג בצורה דומה:



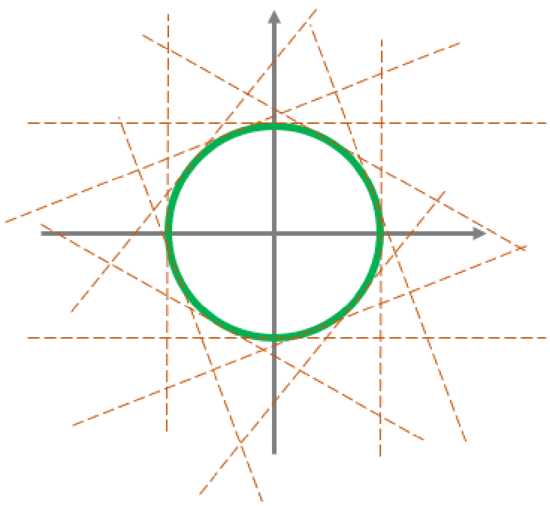
תמונה 63: מימוש פונקציית OR בעזרת פרספטרון.

לאחר שיש בידנו את ה-building blocks ניתן לממש את ה-XOR:



תמונה 64: מימוש XOR בעזרת רשת עם שלוש שכבות.

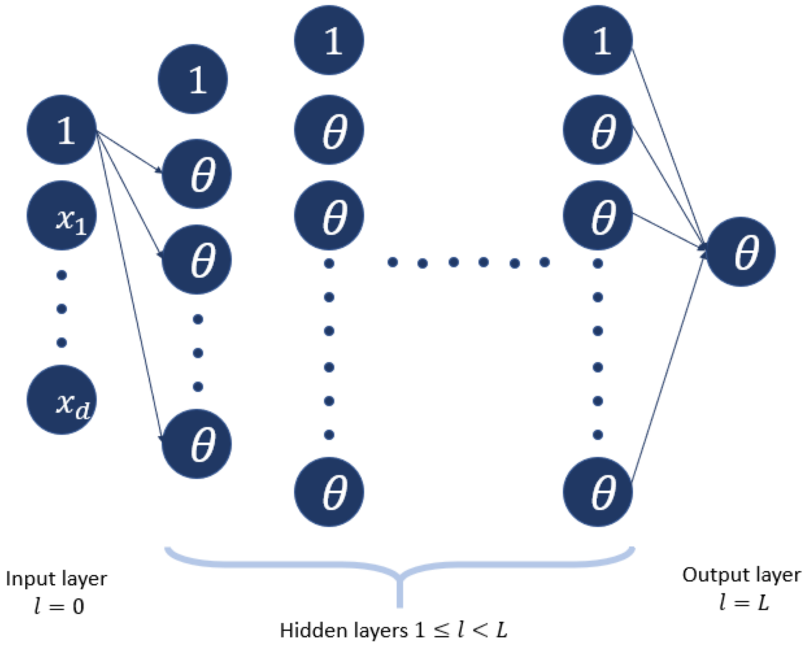
השכבה הראשונה מבצעת טרנספורמציה של סיבוב והזזה של הצירים (כך שניתן יהיה להשתמש ב-XOR גם כשהוא אינו "מיושר" עם הצירים), השכבה השנייה מבצעת את ה-AND והשלישית את ה-OR.
מהדוגמה הזו ניתן להיווכח שסיווג מורכב יותר יכול להיות מיוצג בעזרת רשת עם מספר פרספטרונים, כך ניתן להגיע לקלסיפיקציה לא לינארית של המרחב. שרשור של פרספטרונים יוצר אוסף של מסווגים לינארים אשר יש בכוחם לקרב כל פונקציה.



תמונה 65: קירוב פונקציית סיווג של עיגול ע"י אוסף של מסווגים לינארים.

רשת נוירונים

נגדיר סימונים מוסכמים עבור מרכיבי רשת נוירונים:



תמונה 66: רשת עם L שכבות, פונקציית אקטיבציה θ לא לינארית, וקטור כניסה ממימד d ויציאה אחת.

משקלים יסומנו בצורה הבאה:

$$w_{ij}^{(l)} \begin{cases} 1 \le l \le L \\ 0 \le i \le d^{(l-1)} \\ 0 \le j \le d^{(l)} \end{cases}$$

כאשר i הוא מספר הנוירון ממנו יוצא המשקל בשכבה $l-1$ ו-j הוא מספר הנוירון אליו נכנס המשקל בשכבה l.

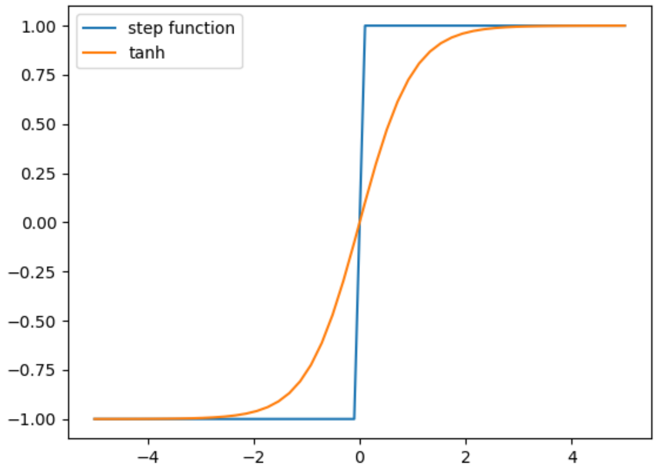
המוצא מנוירון מספר j בשכבה l מוגדר בצורה הבאה:

$$\theta\left(S_j^{(l)}\right) = \theta\left(\sum_{i=1}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right)$$

כאשר $\theta(\cdot)$ היא פונקציית האקטיבציה. במקרה של tanh הפונקציה מוגדרת:

$$\theta(S) = \tanh(S) = \frac{e^S - e^{-S}}{e^S + e^{-S}}$$

פונקציית המדרגה היא הפונקציה האופטימלית לסיווג, אך מכיוון שהיא אינה גזירה קשה לבצע עליה אופטימיזציה. פונקציית ה-tanh מהווה קירוב טוב לפונקציית המדרגה.



תמונה 67 : פונקציית tanh מול פונקציית מדרגה. tanh היא פונקציה גזירה המקרבת בצורה טובה פונקציית מדרגה.

Stochastic Gradient Descent

הגדרנו את מבנה הרשת וקונבנציה על הסימנים, כעת נותר להסביר את תהליך האימון של הרשת. האימון אותו נציג הוא ה-stochastic gradient descent. בתהליך זה ההנחה היא כי מתקיים:

$$\nabla_x\left[\sum_{i=1}^{n} g_i(x)\right] \cong \frac{1}{K}\sum_{k=1}^{K}\nabla_x g_{i_k}(x)$$

כאשר ערכי i_k מוגרלים בצורה אחידה מאוסף כל הדגימות $\{1,\dots,N\}$. המשמעות של ההנחה היא שאין צורך לחשב את הגרדיאנט עבור כל הדוגמאות, וניתן לקרב אותו על ידי דגימות מתוך הדוגמאות. היתרון בשיטה זו הוא המהירות, מכיוון שמספר הדוגמאות יכול להיות גדול מאוד (מאות אלפים, מיליונים ואף יותר) חישוב גרדיאנט על כל הדוגמאות עלול להיות איטי ולא ריאלי.

הביטוי אותו נדרש לחשב הוא:

$$\frac{de(\vec{w})}{dw_{ij}^{(l)}}$$

כאשר $e(\vec{w})$ הוא פונקציית ה-*loss*. באופן עקרוני חישוב של גרדיאנט כזה בצורה ישירה הוא אפשרי, שכן נתון לנו מבנה הרשת והמשקלים, אך חישוב כזה יהיה כבד ביותר ולא יאפשר אימון בפרק זמן סביר.
הפתרון לבעיה הוא שימוש ב-*backpropagation*. הרעיון מאחורי השיטה הוא שימוש בנגזרות חלקיות ומעבר על הרשת מהסוף להתחלה כך שחישוב הנגזרות עבור שכבה נתונה תלוי בשכבה הקודמת:

$$\frac{de(\vec{w})}{dw_{ij}^{(l)}} = \frac{de(\vec{w})}{dS_j^{(l)}}\frac{dS_j^{(l)}}{dw_{ij}^{(l)}}$$

כאשר $\frac{de(\vec{w})}{dS_j^{(l)}}$ הוא הנגזרת של ה-*loss function* לפי היציאה של הנוירון בשכבה l, ו-$\frac{dS_j^{(l)}}{dw_{ij}^{(l)}}$ הוא הנגזרת של היציאה הנ״ל לפי המשקל $w_{ij}^{(l)}$. ידוע כי:

$$S_j^{(l)} = \sum_{i=1}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$$

ולכן:

$$\frac{dS_j^{(l)}}{dw_{ij}^{(l)}} = x_i^{(l-1)}$$

נגדיר:

$$\frac{de(\vec{w})}{dS_j^{(l)}} = \delta_j^{(l)}$$

בהנחה שנמצא את $\delta_j^{(l)}$ הבעיה פתורה, ועדכון המשקלים יתבצע כך:

$$\Delta\vec{w} \leftarrow -\eta\nabla e(\vec{w}) = \frac{de(\vec{w})}{dw_{ij}^{(l)}} = \frac{de(\vec{w})}{dS_j^{(l)}}\frac{dS_j^{(l)}}{dw_{ij}^{(l)}} = \delta_j^{(l)} x_i^{(l-1)}$$

בתהליך ה-backpropagation החישוב יתחיל מהשכבה האחרונה כאשר קיים נוירון אחד ביציאה (קל להרחיב את המשוואות למקרה של מספר יציאות). בנוסף נניח loss function מהצורה של MSE ופונקציית אקטיבציה של tanh.

חישוב $\delta_j^{(L)}$ לשכבה אחרונה:

שכבה אחרונה בעלת נוירון אחד:

$$l = L$$
$$j = 1$$

פונקציית loss של MSE:

$$e(\vec{w}) = \left(x_1^{(L)} - y\right)^2$$
$$x_1^{(L)} = \theta\left(S_1^{(L)}\right)$$

פונקציית אקטיבציה tanh:

$$\theta(S) = \tanh(S)$$
$$\theta'(S) = 1 - \theta^2(S)$$

חישוב $\delta_1^{(L)}$:

$$\delta_1^{(L)} = \frac{de(\vec{w})}{dS_1^{(L)}} = \frac{d}{dS_1^{(L)}}\left(x_1^{(L)} - y\right)^2 = \frac{d}{dS_1^{(L)}}\left(\theta\left(S_1^{(L)}\right) - y\right)^2 = 2\left(\theta\left(S_1^{(L)}\right) - y\right)\theta'\left(S_1^{(L)}\right)$$
$$= 2\left(\theta\left(S_1^{(L)}\right) - y\right)\left(1 - \theta^2\left(S_1^{(L)}\right)\right) = 2\left(x_1^{(L)} - y\right)\left(1 - \left[x_1^{(L)}\right]^2\right)$$

חישוב $\delta_j^{(l)}$ ל-hidden layers:

$$\delta_i^{(l-1)} = \frac{de(\vec{w})}{dS_i^{(l-1)}} = \sum_{j=1}^{d^{(l)}} \frac{de(\vec{w})}{dS_j^{(l)}} \frac{dS_j^{(l)}}{dx_i^{(l-1)}} \frac{dx_i^{(l-1)}}{dS_i^{(l-1)}}$$
$$= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} w_{ij}^{(l)} \theta'\left(S_i^{(l)}\right)$$
$$\Rightarrow \delta_i^{(l-1)} = \left(1 - \left(x_i^{(l-1)}\right)^2\right) \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} w_{ij}^{(l)}$$

לסיכום, אלגוריתם ה-backpropagation:

1. $Initialize\ \left\{w_{ij}^{(l)}\right\}_{i,j,l}\ randomly$
2. $For\ t = 0, \dots, T:$
3. $\quad Pick\ n \in \{1, \dots, N\}$
4. $\quad Compute\ \left\{x_j^{(l)}\right\}_{j,l}$
5. $\quad Compute\ \left\{\delta_j^{(l)}\right\}_{j,l}$
6. $\quad Update\ w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$
7. $\quad Iterate\ until\ convergence$
8. $return\ \left\{w_{ij}^{(l)}\right\}_{i,j,l}$

בשלב 4 מתבצע ה-forward pass בו מחושבות כל היציאות של כל הנוירונים עבור הדוגמאות והמשקלים הנוכחיים. בשלב 5 מתבצע ה-backward pass בעזרת ה-outputs של שלב 4, ומחושבים ערכי $\delta_i^{(l)}$.