ראייה ממוחשבת ועיבוד תמונה
Computer vision and Image processing

~

אוניברסיטת אריאל

סמסטר קיץ 2019

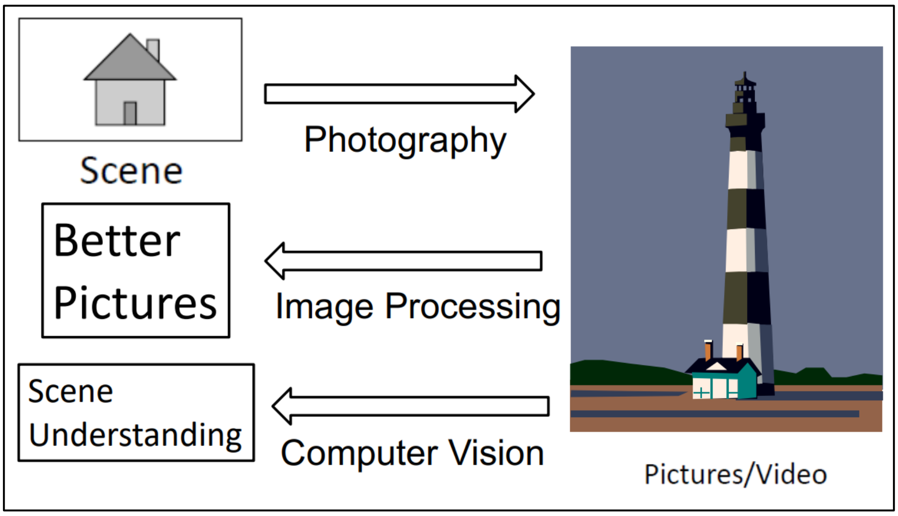
סיכום הקורס על סמך ההרצאות והתרגולים

מרצה: ד"ר גיל בן-ארצי

סוכם ונערך על ידי: חן אסרף

1. הקדמה – סקירה כללית

1.1 מה זה עיבוד תמונה? (image processing)



עיבוד תמונה מתמקד בשיפור התמונה. מקבלים תמונה כקלט- והפלט הוא תמונה גם כן.

יישומי התחום לדוגמא (ואותם נממש בתרגילים) הם:

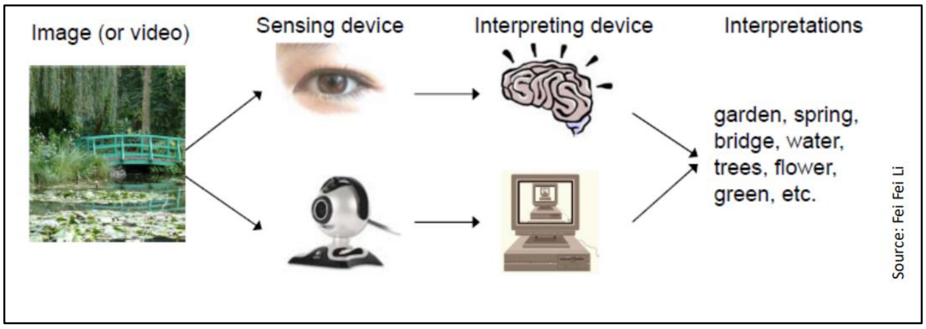
1. הוצאת 'רעש' מתמונה (denoising)
2. עירבול (blending)
3. חיבור תמונות לתמונה פנורמית
4. צביעה מעל תמונה (inpainting)

1.2 מה זה ראייה ממוחשבת? (computer vision)

מה הכוונה *לראייה*?

כשאומרים *ראייה* מתכוונים לתהליך שמערב את המוח. רק אורגניזמים חכמים ונעים יכולים לראות! הראייה מתפתחת בשלבי התפתחות מאוד מוקדמים של היצורים המסוגלים לראות. למעשה, 60 אחוזים מהמוח האנושי מוקדשים לתהליך הראייה הזה.

המטרה שלנו להבין מתמונה אלו אובייקטים נמצאים שם, אילו תזוזות/ פעולות מתרחשות בעת קבלת התמונה.



נתחיל לצלול לתוך תהליך הראייה המלאכותית.

- **3D**
- **Segmentation**
- **Recognition**

3D - כל תמונה מיוצגת באמצעות מטריצה דו ממדית של פיקסלים.

1.3 מושגים חשובים:

Model Based : נקרא 'תכנון מבוסס מודל', וזו מתודולוגיה המיושמת בעיצוב תוכנה מוטמעת.

בהינתן בעיה, יוצרים **מודל חישובי** המתאר את הבעיה בצורה מתמטית.

יוצרים **אלגוריתם** הפותר את הבעיה המתמטית הזו.

הפלט של האלגוריתם- זה פלט הבעיה.

לדוגמא : הבעיה- למצוא קו שחור בודד בתמונה לבנה.

המודל החישובי : $-Y = aX + b$

האלגוריתם : עבור על כל התמונה, חפש אחר פיקסלים שחורים. חבר את כל הפיקסלים. אם נוצרה משוואה מהסוג של המודל – אז מצאנו קו.

הפלט : המיקום של הקו

(Deep) Learning Based : בהינתן בעיה, אוספים כמות גדולה של תמונות עם בעיה דומה.

מאמנים אלגוריתם-למידה למצוא את **הפרמטרים האופטימליים** לפתרון הבעיה.

מכניסים כקלט את הבעיה (לאחר תהליך הלמידה), והאלגוריתם יחד עם הפרמטרים האופטימליים מוצא את הפתרון ומוציא אותו כפלט.

1.4 שלבים בראייה ממוחשבת:

רמה פיזית : הסצנה במציאות (תאורה, השתקפות)

רמה פיזית : תנאי המצלמה : אופטיקה (עדשות), חיישנים (CCD, CMOS)

עיבוד תמונה : קידוד (העברה, דחיסה), שיפור (שינוי צבעים, הוצאת רעשים)

עיבוד תמונה + ראייה ממוחשבת (IP-CV) : זיהוי פיצ'רים (אובייקטים, תנועות, פעולות)

ראייה ממוחשבת : שחזור הסצנה במציאות (תלת ממד)

ראייה ממוחשבת : סיווג אובייקטים/ זיהוי אובייקטים

1.5 מודל מצלמה- pinhole camera

נתחיל בלהבין מה קורה בשלבים המוקדמים של עיבוד התמונה. נתמקד בהבנת מודל המצלמה, שלבסוף יוביל אותנו לדברים יותר מורכבים כמו *stereo* ו *גיאומטריה פרויקטיבית*.

אז מה זו תמונה?

תמונה מיוצגת במחשב על ידי מטריצה דו-ממדית של פיקסלים. כל תא במטריצה מכיל מספר המתאר את הגוון של הפיקסל.

ויקיפדיה : **פִּיקְסֶל** היא יחידת מידע גרפית בסיסית במחשב, המתארת נקודה בתמונה דיגיטלית .

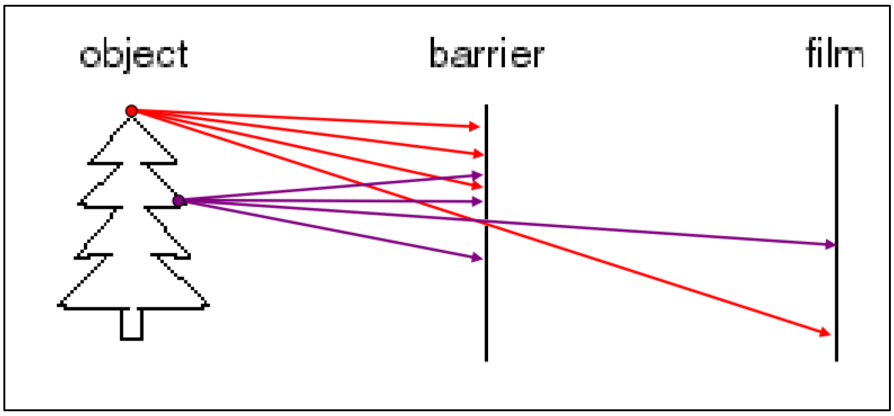
כלומר תמונות הן **פונקציות בעולם הדו ממדי של עוצמת הצבע (intensity) בהינתן המיקום.**

אבל, הפונקציות האלו (המטריצות האלו) הן לא פונקציות שרירותיות, הן הטלה של סצנה אמתית בעולם שלנו התלת ממדי. מה זה אומר?

בצורה פשוטה ניתן לתאר זאת כך: מצלמה מאפשרת הטלה של קרני אור בעולם תלת ממדי, למדיום כלשהו (פילם, חיישן וכו') המאפשר קליטה של קרני אור אלו. אבל למעשה המילה החשובה פה היא *projection – הטלה*. כלומר כשעוברים מעולם תלת ממדי לעולם דו ממדי- לתמונה, מאבדים מידע כלשהו בלי ברירה.

מודל המצלמה - איך נעשית ההטלה של קרני האור למשטח התמונה?

למעשה אם פשוט היינו חושפים את הפילם (נקרא ה *'image plane'*) לכל התמונה- כל קרן אור בסצנה הייתה פוגעת בכל הפילם. אנחנו צריכים להציב "חוצץ" בין הפילם לקרני האור בעולם האמתי, ובחוצץ הזה יהיה חור אחד דרכו יחדרו כל קרני האור. חור זה נקרא *'aperture'* (צמצם).

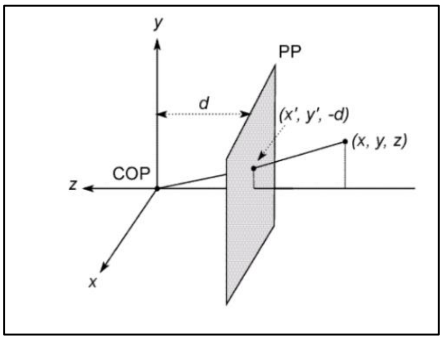


(ניתן לראות כי כעת כל קרן אור תתקבל במשטח התמונה במיקום יחיד והופכי למיקום בסצנה האמתית. תתקבל תמונה הפוכה)

לאחר הוספת עדשה שמשחקת עם הפוקוס של הדברים בתמונה (לא נפרט על זה, ונתמקד במודל הפשוט של pinhole), נוסיף מושג חדש: *'focal point'* . אחרי שהקרניים נקלטות בעדשה ונשברות- הן נחתכות כולן בסופו של דבר בנקודה אחת. זוהי ה focal point .

<u>1.6 Perspective projection- מעולם 3D לתמונה 2D</u>

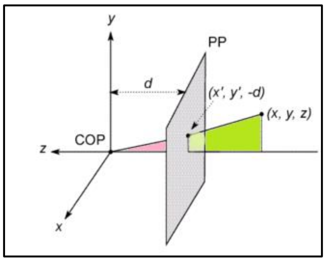
פעולה בסיסית שצריך להבין כדי להבין את מושג התמונה: הטלה (projection) . כדי להבין את המושג הזה טוב נתאר קודם כל את מערכת הקואורדינטות הבסיסית של מודל המצלמה שלנו :



נשתמש במערכת צירים סטנדרטית של x,y,z כדי להעביר את המושגים למתמטיקה (נשים לב שציר הy גדל כלפי מעלה וציר הx גדל כלפי הימין, וכשנצטרך להמיר לתמונה נצטרך לזכור לעשות את ההיפוך הזה)

- נניח את המרכז האופטי בראשית הצירים, **COP – Center Of Projection**. נשים לב שנקודת ה(0,0) מונחת במרכז ולא בצד שמאל למטה, כמו שאנו מכירים מתמונות במחשב.
- נניח בשביל הנוחות המתמטית שמשטח התמונה (image plane) נמצא לפני המרכז האופטי, ולא אחריו כפי שקורה במציאות. (כיוון שכך התמונה לא מתהפכת)
- נשים לב שציר הz כרגע פונה בכיוונו החיובי לתוך המצלמה- ובכיוונו השלילי לעולם האמתי.
- הנקודה שבה קרן אור מנקודה (x,y,z) בעולם חותכת את משטח התמונה תקרא (x',y'-d) והd- בגלל שזהו המרחק מה COP – focal length, וכיוון שמרחק הוא ערך חיובי נוסיף לו מינוס.

נסתכל על משולשים דומים שנוצרו במערכת :



בעזרת המשולשים הללו (הנוצרים מקרני האור, COP, והחיתוך במשטח התמונה) ניצור את משוואת ההטלה, ***Projection equations*** :

Projection equations

$$(x, y, z) \rightarrow (-d\frac{x}{z},\ \ -d\frac{y}{z},\ \ -d)$$

- We get the projection by throwing out the last coordinate:

$$(x, y, z) \rightarrow (-d\frac{x}{z},\ \ -d\frac{y}{z})$$

כלומר אם נדע מהן ערכי הנקודה בעולם האמתי לפי הקואורדינטות של המצלמה שלנו, ומהו ה d כלומר ה focal length נוכל לדעת איפה קרני האור חותכות את התמונה, נוכל לדעת את המיקום של הנקודה בתמונה.

נוריד את הקואורדינטה המיותרת (עוברים לדו-ממד) ונישאר עם שתי קואורדינטות.

(נקודה למחשבה : בגלל שהקואורדינטות בתמונה מושפעות ישירות ממד המרחק- ככל שהאובייקט המצולם רחוק יותר, כך נחלק בערך z גבוה יותר ולכן האובייקט בתמונה יופיע קטן יותר).

1.7 מקואורדינטות מעל המספרים הממשיים לקואורדינטות מעל המספרים הטבעיים (2D real to 2D discrete)

אז מהי תמונה במחשב? למעשה זוהי פונקציה. נקרא לה I, בהינתן מיקום בתמונה (x,y) הפונקציה מחשבת את עוצמת הגוון במיקום.

$$I(x, y): R^2 \rightarrow R$$

למעשה עיבוד תמונה תמיד יהיה לקחת פונקציה כזאת ולחשב ממנה פונקציה אחרת, בדרך כלל פונקציה דומה.

אבל הפונקציות הללו לא שרירותיות, למעשה נגביל אותן מעל מלבן מסוים במרחב עם טווח מוגבל של עוצמה :

$$I: [a, b] \cdot [c.d] \rightarrow [\min, \max]\ intensity$$

כאשר הערך המינימלי של העוצמה יהיה שחור, הערך המקסימלי יהיה לבן.

לרוב נעבוד ונדגים דברים על תמונה בגווני אפור (לשם הנוחות וגם כיוון שהדברים לבסוף זהים).

תמונה צבעונית היא למעשה שלוש פונקציות שמחברים ביחד: $\mathrm{I}(x, y) = \begin{matrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{matrix}$

ניתן לחשוב על תמונה צבעונית כעל "פונקציית וקטור" כך שכל מיקום פיקסל הנכנס לפונקציה יוצא כווקטור של עוצמות לפי הגוון.

(נקודה חשובה להמשך : בקוד שנכתוב image(x,y) למעשה יהיה image(y,x), כיוון שבמטריצות הערך הראשון הוא מספר השורה, הערך השני הוא מספר העמודה. ציר ה-y הוא השורות, ציר ה-x הוא העמודות)

בראייה ממוחשבת נעבוד עם תמונות דיגיטליות- הערכים שנמדדים בדידים. ולכן נצטרך לבצע שני דברים :

- **לדגום** *(sample)* את המרחב לרשת דו ממדית- כלומר למספר סופי של פיקסלים
- **לכמת** *(Quantization)* כל דגימה (לעגל לערך הקרוב ביותר) מספר סופי של גוונים

נקודה חשובה: למרות שאנחנו דוגמים למספר סופי, בקוד שנכתוב נתייחס לערכים כערכי *floating points* וזאת כיוון שאם נבצע פעולות על תמונות עם משתנים שלמים *uint8* לדוגמא, הקוד פשוט לא יעבוד. נראה זאת בהמשך.

דוגמא לכימות של תמונה – *Quantization*:

נניח שיש לנו תמונה עם ערכים שונים החל מ-1.3- עד ל 7.5 אבל התמונה הדיגיטלית שלנו יכולה להציג רק גוונים בערכי 0-5. איך נכמת את התמונה?

1. נעגל כלפי מטה לשלם את הערך.

2. כל ערך שמתחת לגבול התחתון- מומר ל-0. כל ערך מעל הגבול העליון- מומר ל-5.

דוגמא נוספת המדגימה היטב כיצד *Quantization* גורמת לאיבוד ערכים במעבר לתמונה דיגיטלית, לפי כמות ערכי הגוונים שניתן להציג:

256 Levels　　8 Levels　　4 Levels　　2 Levels

<u>1.8 סוגי תמונות שונות:</u>

- תמונת שחור ולבן: 1 bit לכל פיקסל
- תמונת גוני-אפור: 8 bit (grayscale) לכל פיקסל
- תמונת צבעים: 8 bit x 3 channels

<u>1.9 Intensity Transformations:</u>

שינוי טווח הצבע (העוצמה של הצבע) בתמונה.

לדוגמא: תמונה הופכית = צבע נוכחי – 1

<u>1.10 טרנספורמציות על היסטוגרמות</u>

<u>Histogram Equalization</u>:

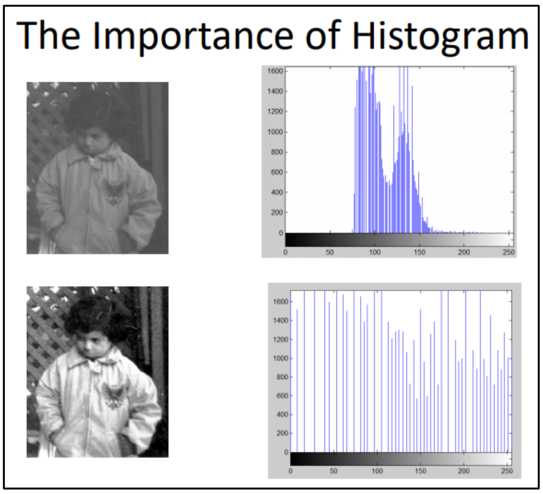
← מה זה *היסטוגרמה (histogram)*?

ההיסטוגרמה היא מאפיין של התמונה, ממנו ניתן ללמוד על איכותה. היא מוגדרת כפונקציית השכיחות של רמות האפור, מבלי להתייחס למיקום הפיקסלים בהם מופיעות רמות אפור אלו. אם למשל משתמשים ב n - ביטים לקידוד רמת האפור של כל פיקסל, אז ניתן לייצג 2^n רמות אפור שונות. ההיסטוגרמה היא פונקציה של רמת האפור, וערכה, עבור רמת אפור α, הוא מספר הפיקסלים בתמונה שבהם רמת האפור היא α.

$$hist_{img}(\alpha) = ||(i,j) \mid img(i,j) = \alpha||$$

אם ננרמל את ההיסטוגרמה נקבל פונקציה הדומה לפונקציית הסתברות- כלומר מה אחוזי הגוון הספציפי בתמונה.

החשיבות של היסטוגרמה: ניצול אפשרויות התצוגה של התמונה ובכך שיפור החדות, על ידי מיפוי של תחום קבוצת גוונים אחד לתחום אחר. מיפוי זה יקבע על ידי ההיסטוגרמה בלבד, ללא צורך בחקירת התמונה באופן ישיר.



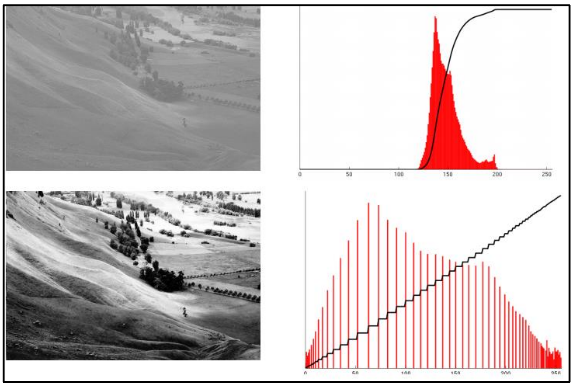
נציג כעת דרך לשפר את הניגודיות בתמונה בעזרת היסטוגרמה.

← *שלבים כלליים בפתרון בעיות:*

1. יצירת מודל (עם נוסחה) שמתאר את העולם שבתחומו הבעיה בשימוש פרמטרים/ מאפיינים מתאימים
2. הגדרת הבעיה והפתרון הרצוי
3. יצירת אלגוריתם לפתרון הבעיה בשימוש המודל
4. בדרך כלל נתחיל בלהשתמש בדוגמא ספציפית ולפתור אותה, ולאחר מכן נכליל

המטרה: שיפור הניגודיות בתמונה

1. *יצירת מודל המתאר את העולם-*
פרמטריזציה של תמונה: היסטוגרמה.
נרצה להבין כיצד היסטוגרמה 'מתנהגת'? מה נחשב היסטוגרמה 'טובה' לבעיה ומה נחשב 'רע'?
היחסים בין ניגודיות להיסטוגרמה:



הנחה בסיסית מקובלת היא שתמונה טובה מנצלת את כל רמות האפור, כמו שניתן לראות בדוגמא לעיל.
תמונה בעלת תחום דינמי מוגבל של רמות אפור אינה מנצלת כראוי את כל אפשרויות התצוגה של המסך ויהיה קשה להבחין בה בפרטים.

2. *הגדרת הבעיה והפתרון הרצוי-*
הבעיה: התפרסות לא אחידה של גווני אפור.
הפתרון: שימוש אחיד בכל רמות האפור שיש.

3. *יצירת אלגוריתם לפתרון הבעיה בשימוש המודל-*
איזון היסטוגרמה (Histogram equalization). נשתמש במה שנקרא 'היסטוגרמה נצברת לינארית' - *linear cumulative histogram*.

- שלב ראשון : נמצא את ההיסטוגרמה של התמונה (נספור כמה פיקסלים יש מכל גוון אפור אפשרי)

$$\begin{bmatrix} 3 & 2 & 4 & 5 \\ 7 & 7 & 8 & 2 \\ 3 & 1 & 2 & 3 \\ 5 & 4 & 6 & 7 \end{bmatrix}$$

Pixel Intensities	1	2	3	4	5	6	7	8	9	10
No. of pixels	1	3	3	2	2	1	3	1	0	0

- שלב שני : נחשב את ההסתברות לכל גוון אפור בתמונה – ההסתברות כאן היא אחידה ולכן תהיה : מספר הפיקסלים מגוון האפור הזה חלקי סה"כ מספר הפיקסלים בתמונה.
- שלב שלישי : לחשב את ההסתברות המצטברת לכל גוון אפור (פשוט לחבר את ההסתברויות). CDF מלשון cumulative distributive function. וזו פונקציית סכום ההסתברויות

Pixel Intensity	1	2	3	4	5	6	7	8	9	10
No. of pixels	1	3	3	2	2	1	3	1	0	0
Probability	.0625	.1875	.1875	.125	.125	.0625	.1875	.0625	0	0
Cumulative probability	.0625	.25	.4375	.5625	.6875	.75	.9375	1	1	1

$$cdf(R) = \sum_{g}^{R} p(g)$$

- שלב רביעי : נחשב את התוצאות האחרונות ברמה המקסימלית של גווני האפור החדשים
- שלב חמישי : נעגל למטה לשלב הכי קרוב
- נמפה כל פיקסל בגוון אפור ישן לגוון האפור החדש לפי התוצאות בשלב החמישי
- נוודא שהגוון המינימלי נשאר 0 והגוון המקסימלי הוא K-1 (כאשר K הוא מספר גווני האפור החדש) אחרת צריך לעשות מתיחה להיסטוגרמה

← Adaptive Histogram Equalization :

שונה מהשוואת היסטוגרמה רגילה בכך שהיא מחשבת כמה היסטוגרמות לכל אזור בתמונה.
מה הסיבה שצריך עוד שיטה לשיפור הניגודיות בתמונה?
במקרה שיש תמונה שבה עוצמת האור לא אחידה – כאשר יש אזורים שמשמעותית בהירים יותר או כהים יותר מהשאר.

1.11 Quantization - קוונטיזציה

המרת ערכים אנלוגיים (אותות, תדרים) לערכים דיגיטליים (ביטים) דורשת עיגול של הערכים האנלוגיים הרציפים לערכים בדידים. השיטה היא לדגום מספר אותות, לצרף אותם ולעגל לערך הכי קרוב יחידי. תהליך זה נקרא *קוונטיזציה*.

ניתן להתייחס לקוונטיזציה גם כאל המרת אות אחד למשנהו תוך תחימת ערכי האות המומר לתחום אחר (שקטן בדרך כלל מהתחום המקורי.

בעיבוד תמונה, קוונטיזציה היא למעשה חלוקת אות לקוונטה (חלקים).

למעשה , קוונטיזציה מתעסקת במספר הרמות של גווני האפור שיש. כשנרצה להקטין אותם – נעשה קוונטיזציה.

בכללי – אם אות הכניסה בעל פילוג של p(z) (רמות האפור) אזי קביעת רמות הקוונטיזציה תעשה על פי עקרון השגיאה הריבועית :

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

$$q_i = \frac{\int_{z_i}^{z_{i+1}} z \cdot p(z) dz}{\int_{z_i}^{z_{i+1}} p(z) dz} \qquad z_i = \frac{q_i + q_{i+1}}{2}$$

זאת בהינתן ש k זה מספר גווני האפור החדש שרוצים, הגבולות של כל דגימה הם z והערכים עצמם הם q.

2. עיבוד תמונה – פעולות מרחביות על תמונות

פעולות על תמונות:

- פעולות נקודה (פעולות על ערך פיקסל מסוים, חסרות הקשר- אין צורך לזכור את הנתונים הקודמים. דוגמא: היסטוגרמה)
- פעולות גיאומטריות (פעולות על הקואורדינטות של הפיקסל, חסרות הקשר. לדוגמא: רוטציה של התמונה)
- פעולות מרחביות (פעולות שתלויות בערך הפיקסל ובקואורדינטות שלו. בעלות הקשר- תלויות בסביבה של הפיקסל)

בשיעור זה נתמקד בפעולת מרחביות על תמונות.

2.1 מסנן- filtering

הפעלת אותה פונקציה עבור כל חלון קבוע של פיקסלים בתמונה.

טכניקת sliding-window:

- עבור כל חלון פיקסלים בגודל KxK
- חשב את הפונקציה עבור כל הפיקסלים בחלון
- עבור הפיקסל הממוקם במרכז החלון- תשנה את ערכו לערך הפונקציה מעל כל הפיקסלים

כל מסנן (filter) יוגדר באמצעות מטריצה של משקולות המכפילות כל פיקסל בחלון. מטריצה דו-ממדית זו תיקרא kernel (גרעין).

<u>דוגמאות למסננים להפחתת רעשים בתמונה:</u>

Input Image

Sharpen kernel (filter kernel)

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

Output Image

Original Image

Min/max filter

$$\min filter: f'(x,y) = \min_{\text{m,n}\in\text{N(x,y)}} \{f(m,n)\}$$

$$\max filter: f'(x,y) = \max_{\text{m,n}\in\text{N(x,y)}} \{f(m,n)\}$$

with Minimum Filter

Original Image

with Maximum Filter

Original Image

Salt & pepper noise filter

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

Salt & Pepper Noise

Median filter

$$f'(x,y)=med(\{f(m,n)\})_{(m,n)\in N(x,y)}$$

ממיינים את עוצמת הפיקסלים ולוקחים
את גוון העוצמה האמצעי ברשימה

Noisy image

Average filter

$$f'(x,y)=\text{mean}(\{f(m,n)\})_{(m,n)\in N(x,y)}=\frac{1}{|N|}\sum_{(m,n)\in N(x,y)} I(m,n)$$

סוכמים את כל הגוונים בחלון ומחלקים
במספר הפיקסלים בחלון – ממוצע

7x7 average

2.2 Convolution – קונוולוציה:

קונבולוציה זו פעולה מתמטית בינארית בין שתי פונקציות או סדרות ערכים. מקבול לרשום את סימן הקונבולוציה בעזרת *. תכונות הקונבולוציה הן : אסוציאטיביות וקומוטטיביות ,ודיסטריבוטיביות ביחס לחיבור, (ולכן יכולה להיות מבוטאת בחשבון מטריצות).

תזכורת :

- *אסוציאטיביות* (חוק הקיבוץ) : $a*(b*c)=(a*b)*c$
- *קומוטטיביות* (חוק החילוף) : $a*b=b*a$
- *דיסטריבוטיביות* (חוק הפילוג) : $c*(a+b)=c*a+c*b$

קונבולוציה ב 1D

מבצעת את הדברים הבאים :

1. בהינתן וקטור
2. תהפוך אותו שהסוף יהיה ההתחלה
3. תעביר אותו מעל כל הוקטור השני
4. תכפיל את הערכים

במתמטיקה הגבולות הם מינוס אינסוף לאינסוף. בפועל נחשיב את המספרים שמעבר לגבולות הוקטור 0.

<u>פסאודו קוד convolution, באופציית 'same':</u>

קלט: מטריצה דו ממדית של תמונה, kernel דו ממדי.

פלט: התמונה לאחר קונבולוציה

אלגוריתם:

- להוציא את אורך ורוחב התמונה, אורך ורוחב הגרעין
- האורכים המקסימליים הם יהיו מידות התמונה בפלט
- להוציא את הערך התחתון של חצי מרוחב הגרעין (ההנחה היא שהגרעין תמיד יהיה ריבוע- אורך ורוחב שווים, ואורכם יהיה אי זוגי כך שיהיה פיקסל יחידי במרכז)
- ליצור מטריצה 'מרופדת' המוכנה לקבל את ערכי הפלט ('מרופדת' הכוונה למטריצה שמכילה אפסים במסגרת/ ערכי הקצה משוכפלים, כדי שתמיד יהיה פיקסל במרכז כשנכפיל בגרעין . והמסגרת היא חצי מרוחב הגרעין)
- עבור כל ערך y בשורות של המטריצה, ועבור כל ערך x בעמודות של המטריצה:
  - נחשב את הסכום של החלון כאשר הפיקסל (x,y) נמצא במרכזו
    (image[y-half_kernel : y+half_kernel , x-half_kernel : x+half_kernel] * kernel).sum()
  - נכניס את הערך הזה למטריצת הפלט
- נחזיר את התמונה לאחר convolution

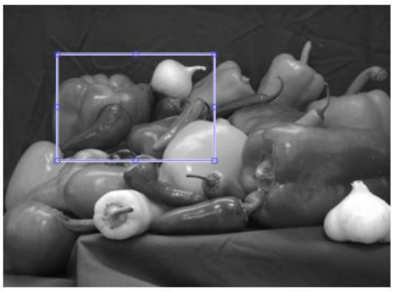
2.3 Correlation:

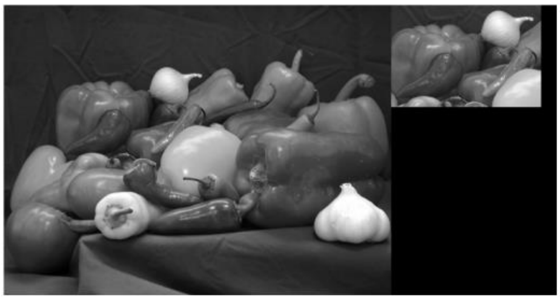
קורולציה היא אותו הדבר כמו קונוולוציה רק בלי להפוך את הווקטור. לכן במקרה שהווקטור זהה מההתחלה לסוף ומהסוף להתחלה, ה- Correlation וה- Convolution עם הווקטור הזה זהות.

3. עיבוד תמונה – Low level detection

3.1 Template Matching:

בהינתן תמונה/אות ותבנית של חלק מהתמונה/אות, נרצה למצוא את המיקום של התבנית בתמונה.





נייצג את התבנית כמטריצה. האתגר פה שיכול להיות שהתבנית תהיה מעט שונה מהתמונה בגלל סיבות שונות- רעש בתמונה, אור שונה. נרצה להשתמש ב *Normalized Cross Correlation* :

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

<u>פסאודו קוד template matching, עם NCC:</u>

קלט: מטריצה דו ממדית של תמונה NxN, תבנית דו ממדית KxK.

פלט: מיקום התבנית בתמונה

אלגוריתם:

- עבור כל חלון KxK אפשרי בתמונה- חשב את ה NCC בין התבנית לחלק בתמונה
- הערך הגבוה ביותר הוא מיקום התבנית המשוער
- החזר את מיקום התבנית המשוער

3.2 Edge Detection:

אנחנו רוצים להמשיך לאפיין דברים חשובים בתמונות. דיברנו על זיהוי תבניות, ועכשיו נניח ואין לנו מראש תבנית אותה צריך למצוא בתמונה. עדיין, נוכל למצוא מאפיינים חשובים בתמונה אם לרגע נוכל 'לחלץ' מהאובייקטים בתמונה את הקווים המגדירים אותם- הצלעות, edges.

במילים אחרות- בהינתן תמונה, שכפי שכבר אמרנו היא פונקציה של מיקום הנותנת גוון, אנחנו רוצים לחלץ את הפיקסלים החשובים ביותר.

ההנחה הבסיסית בזיהוי 'הצלעות' היא שיהיה שינוי משמעותי בגוונים בין הנקודות שהן צלעות לנקודות אחרות בסביבתן.

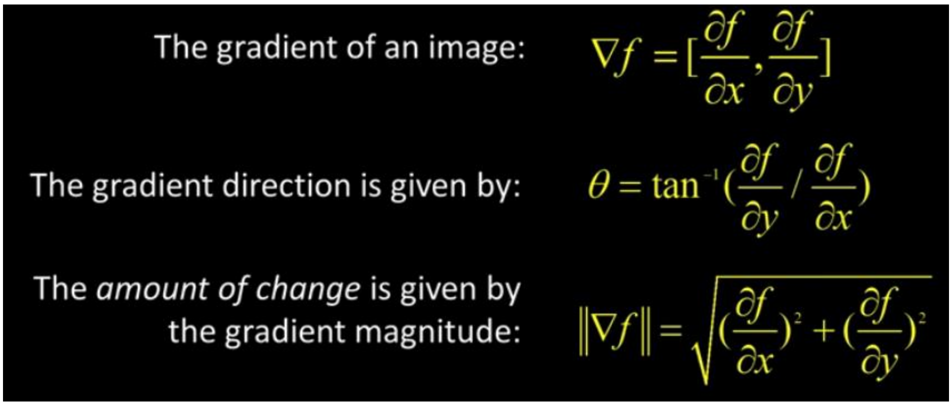
שתי שאלות שצריך לשאול את עצמנו כאן :

1. מה הסביבה שצריך לבדוק?
2. איך נזהה את השינוי?

נענה על השאלה השנייה קודם : שינוי בפונקציה מוצאים באמצעות נגזרת. את השינויים הכי קיצוניים- מוצאים באמצעות נקודות מקסימום ומינימום בנגזרת.

נמצא את הנגזרת על ידי kernel וקונבולוציה- הפעם עם מטריצה שמותאמת במיוחד כדי למצוא את הנגזרת. במקרה של פונקציה של יותר משני משתנים- מציאת *גרדיאנט*.

גרדיאנט זהו וקטור המכיל את הנגזרות החלקיות בכיוון ציר הx ובכיוון ציר הy. וקטור הגרדיאנט עצמו זה הכיוון בו יש את השינוי הכי גבוה בערכי הפונקציה (עוצמות הגוונים במקרה שלנו), והגודל שלו זה כמות השינוי.



(תמונה מסרטון באתר udacity)

הבעיה שהעולם לא מושלם, וכשדוגמים חלק ממנו באמצעות תמונה/אותות גם הדגימה לא תהיה מושלמת ותמיד יהיו רעשים.

רעשים יגררו לשינויים צפופים בעוצמה של הפיקסל/האות, ולכן כאשר נחשב גרדיאנט לקלט נקבל תוצאה ממנה יהיה קשה לחלץ את המקומות בהם באמת נעשה שינוי משמעותי.

על מנת להימנע מהשפעה של רעשים בתמונה על חישוב הנגזרות נפעיל את חישוב הנגזרת עם Gaussian kernel , ככל שגודל הגרעין של ההחלקה (Gaussian) יהיה יותר גדול כך השינויים בתמונה יהיו לאורך יותר מרחב ופחות דרמטיים.

$$\sigma \coloneqq gaussian\ kernel\ size$$

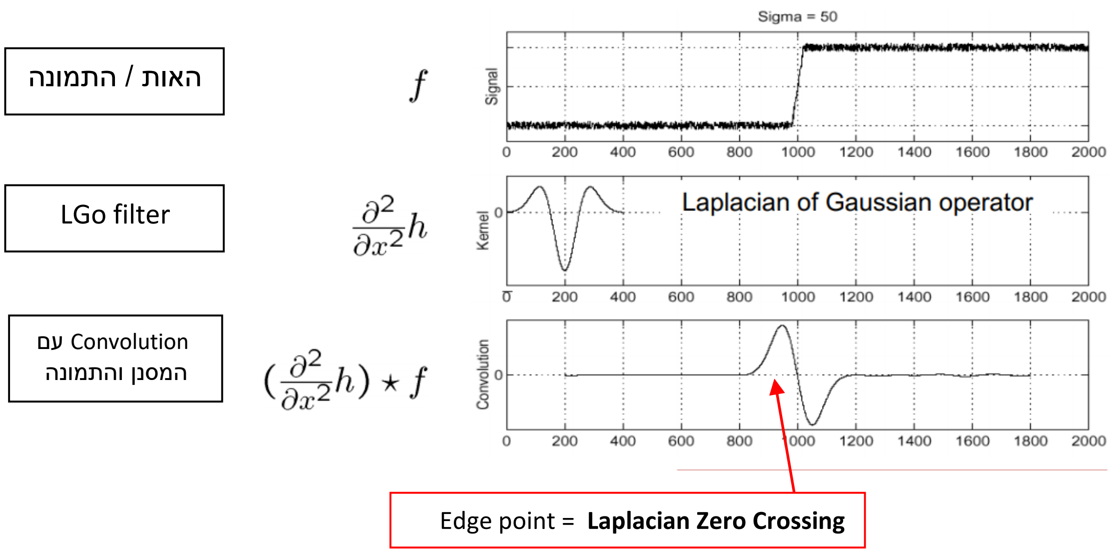
בגלל שconvolution זו פעולה לינארית, כלומר לא משנה אם נבצע נעשה החלקה לתמונה ורק אז נגזור את התמונה, או שנעשה קונבולוציה בין המטריצה להחלקה למטריצה לנגזרת ואז נפעיל את זה על התמונה.

לאחר שהחלקנו וגזרנו- נרצה למצוא את נקודות הקיצון שיצאו בנגזרת. כמו שאנחנו כבר יודעים, כדי למצוא את נקודות הקיצון נצטרך לגזור שוב את הפונקציה שהתקבלה, ולאחר מכן למצוא את הנקודות שערכן שווה ל – 0.

בממד אחד (בדגימת אותות) כך זה יראה :

$$\frac{\partial^2}{\partial x^2}(h*f)=\boxed{\left(\frac{\partial^2}{\partial x^2}h\right)}*f$$

Laplacian of Gaussian



בשני ממדים (בתמונה לדוגמא) הדבר יהיה שונה מעט- כיוון שניתן לקחת את הנגזרת בכמה כיוונים. איך נדע באיזה כיוון לחשב את הנגזרת?

ניקח את הנגזרת השנייה בכל כיוון ונחבר אותה:

The Laplacian: $$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

The Laplacian in matrix form: $$\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

לסיכום, יצרנו filter חדש: **laplacian of Gaussian**

אלגוריתם נוסף למציאת קווי מתאר: **Canny Edge Detector**

<u>תוכנית כללית</u>

1. נפעיל convolution/ correlation עם מסנן גאוסייני + נגזרת, על מנת 'לדכא' רעשים בתמונה ולחשב נגזרות.
2. Threshold (בצורה מסוימת) לתמונה - על ידי כך נוכל למצוא את האזורים עם הגרדיאנט המשמעותי. (דוגמא לשיטה- עבור כל פיקסל שקיבל מתחת לערך סף מסוים בנגזרת- להפוך אותו לשחור וכל השאר בלבן)
3. 'thin'- לדלל את התמונה כך שצלע בעלת גרדיאנט זהה תהיה קו מתאר.
4. לבסוף- לחבר את הקווים האלו כדי שיהיו באמת קווי מתאר ברורים. (Hysteresis)

<u>אופרטור/אלגוריתם canny למציאת קווי מתאר:</u>

1. סינון התמונה על ידי נגזרת גאוסיינית
2. מציאת הגודל והכיוון של הגרדיאנט
   a. אזורים גדולים שכולם עלו על הסף של ה Threshold – נהפוך אות לקו ברור על ידי שימוש בגרדיאנט. המיקום בו השינוי הכי מרבי (יהיה בדרך כלל במרכז) יהיה המקום בו יעבור הקו.
3. חיבור ו Threshold של קווי המתאר. בעיה: יש קווי מתאר שלא שרדו את סינון הסף (Threshold). כדי לפתור את הבעיה וליצור קו מתמשך אחיד- נעשה את הדבר הבא:
   - נפעיל דילול עם סף גבוה כדי למצוא את הפיקסלים החזקים של קווי המתאר
   - נחבר את הפיקסלים החזקים וניצור קווי מתאר חזקים
   - נפעיל דילול עם סף נמוך- כדי למצוא קווי מתאר חלשים ודקיקים יותר ועדיין אפשריים
   - נרחיב את קווי המתאר החזקים להיות מחוברים להמשכם הדקיק

   פתרון זה מתבסס על ההנחה כי כל קו מתאר שחשוב יכיל פיקסלים חזקים מספיק כדי לעבור את הסינון הראשוני.

נשים לב: האלגוריתם למציאת קווי מתאר של canny רגיש לרעשים בהתאם לגודל ה σ שנבחר.

לסיכום, שיטות למציאת קווי מתאר:

1. הזזה בפיקסל אחד של התמונה שמאלה, הזזה של פיקסל אחד ימינה, וחיסור בין שתי התוצאות
2. Laplacian of Gaussian
3. Canny edge detector

3.3 Hough transform – זיהוי קווים ועיגולים:

לאחר שהבנו איך מוצאים קווי מתאר בתמונה של אובייקט. האם אנחנו יכולים לזהות צורות גיאומטריות- כמו קווים, עיגולים?

מושג ראשוני שחשוב להכיר:

Parametric model: מגדיר מחלקה של אובייקטים באמצעות פרמטרים, כאשר כל אובייקט מזוהה באמצעות הפרמטרים.
לדוגמא: קו ניתן לבטא באמצעות משוואת ישר: $y = mx + b$.

כשנשתמש במודל פרמטרי לצורך זיהוי צורות בתמונה נשים לב לשלוש נקודות חשובות:

1. באיזה מודל נשתמש, איך נייצג את הצורה?
2. השיוך למחלקה מסוימת תלוי בסביבה גדולה של נקודה ולא רק במיקומה
3. הסיבוכיות של החישוב והאפשרויות גורם מכריע

<u>זיהוי קווים</u>

אתגרים:

1. אקסטרה קווים- יהיו כמה וכמה מודלים מתאימים
2. לפעמים לא מקבלים קווים שלמים, ויש בהם חלקים שנעלמו בתהליך
3. רעשים בתמונה- גורמים לקווים להיות עם מעט סטיות ומקשים על תהליך זיהויים

לא ניתן לבצע בדיקה של כל האפשרויות של הקווים במרחב התמונה ולראות מה באמת מתאים למתאר שנמצא כבר. מה שכן אפשר לעשות- לתת למידע לדבר בעד עצמו.

Voting- הצבעה, טכניקה כללית שבה נותנים לפיצ'רים ל'הצביע' לכל המודלים שהם יכולים להתאים להם.

1. נעבור על כל הפיצ'רים בתמונה, כל פיצ'ר 'מצביע' למודל
2. נחפש את המודלים שמקבלים הכי הרבה 'הצבעות'

למה זה יעבוד?

1. גם אם יהיו הרבה רעשים שיצביעו למודלים לא נכונים- ההנחה היא שטעויות מתפרסות על פני הרבה מודלים ולכן עדיין המודלים שיקבלו את רוב ה'הצבעות' יהיו המודלים הנכונים
2. גם אם יש פיצ'רים שנפספס בגלל שנפלו בתהליך זיהוי הצלעות- זה בסדר כי נקודות אחרות שתפסנו יצביעו למודל הזה.

כדי להתאים קווים אנחנו צריכים לשאול כמה שאלות:

1. בהינתן נקודות השייכות לקו, מהו הקו?
2. כמה קווים כאלה יש?
3. איזה נקודות שייכות לאיזה קו?

אלגוריתם Hough transform מוצא קווים באמצעות טכניקת ה'הצבעה'-

1. כל נקודה מצביע לקו אפשרי
2. נחפש את הקווים שקיבלו הכי הרבה הצבעות.

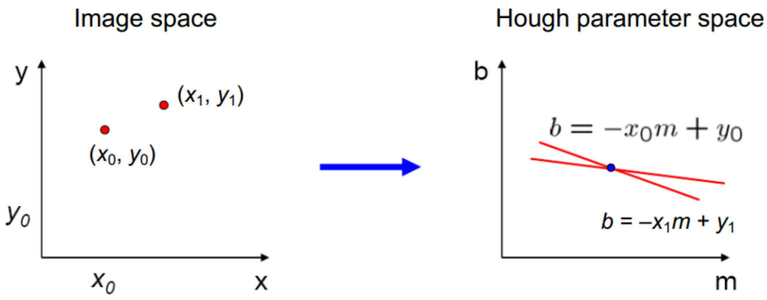
<u>Hough space:</u> כל קו ניתן לייצוג על ידי משוואת ישר: $y = mx + b$

ניתן להעביר את ייצוג הישר ממרחב התמונה למרחב Hough, בו הצירים הם ציר m וציר b.

כל קו בתמונה (ובמרחב הקואורדינטות של x ו y) – הוא **נקודה** במרחב Hough.

כל נקודה בתמונה – היא **קו** במרחב Hough.

שתי נקודות בתמונה- הן **שני קווים** במרחב Hough, ולכן הקו שעובר דרך שתי הנקודות בתמונה הוא **נקודת החיתוך בין שני הקווים** במרחב Hough. (דואליות)



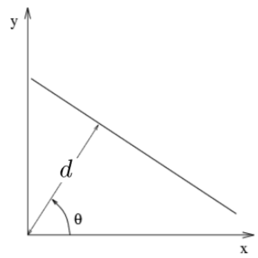
רעיון האלגוריתם של Hough:

1. נחלק את מרחב Hough ל'סריגים' – הם יהיו הפרמטרים להם 'נצביע'.
2. כל נקודה במרחב התמונה תהיה קו במרחב Hough
3. התא במרחב Hough שיהיה בו הכי הרבה נקודות חיתוך מבין כל התאים במרחב מייצג את הקו הנבחר במרחב התמונה.

קצת מתמטיקה:

כדי להימנע מבעיות מתמטיות בהמשך, נשים לב לייצוג של קו אנכי. לקו המאונך לציר הx יש שיפוע של אינסוף, אבל במקרה שלנו זה יצור בעיות. לכן נבחר לייצג קו בצורה פולארית.

ייצוג קו בצורה פולארית מתבצע על ידי המרחק מראשית הצירים והזווית של הקו ביחס לצד החיובי של ציר הx (טריגונומטריה):



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

נשים לב שבצורה הפולארית, הקו יראה במרחב Hough כפונקציית גל.

רלוונטי לכתיבת קוד:

- אם d יכול להיות רק חיובי, אז θ יכולה להיות כל הערכים מ 0 עד 360 מעלות.
- אם d יכול להיות גם שלילי, אז θ יכולה להיות רק הערכים מ 0 עד 180 מעלות

:Hough Algorithm

בהינתן מערך של edge points

1. ניצור מטריצה H המייצגת את מרחב Hough, בה מספר השורות מייצגות את מספר הערכים האפשריים ל d ומספר העמודות θ (hough accumulator array). נצטרך להחליט כמה תאים יהיו, ונאתחל את ערכיה ל-0
2. לכל edge point (x,y) בתמונה
   עבור מ 0 עד 180 (בעבור θ . נשים לב שחייבים לעגל פה למספר שלמים ולכן לא ניתן לרוץ עד $2\cdot\pi$)
   קבע את d בתור: $d = xcos(\theta) + ycos(\theta)$
   תגדיל את הכניסה במטריצה H במיקום θ, d ב1
3. תמצא את המיקום (או מיקומים) במטריצה H בו הערך מקסימלי
4. החזר את הקו על ידי משוואת הישר: $d = xcos(\theta) + ycos(\theta)$

- אם הנקודות המקסימליות שנמצא במרחב Hough נמצאות קרוב אחת לשנייה- זה למעשה אומר שהקווים בתמונה שוכנים בסמוך אחד לשני.
- בתמונה אמתית לא נרצה קו אינסופי אלא קטעים (segmentation)

הרחבות:

1. אם בתמונה יש רעשים, נקודת החיתוך בין הגלים במרחב Hough לא תהיה נקודה אחת (ואם עשינו את הפערים בין התאים יחסית קטנים נקודת החיתוך הזו לא 'תיתפס' באחד מהם). מה שאפשר לעשות- לעשות החלקה לתמונה ואז לבחור את התא המקסימלי.
2. הרחבה שנמצאת בשימוש די הרבה: במקום לחשב עבור כל θ, נמצא באמצעות הגרדיאנט את הזווית בנקודה הזו. כך נצטרך 'להצביע' עבור כל נקודה ספציפית (ולא עבור כל ערך אפשרי של הזווית)
3. לתת משקל חזק יותר (יותר 'הצבעות') לedge points חזקות יותר
4. לחלק ליותר/פחות תאים את הדגימה של θ, d

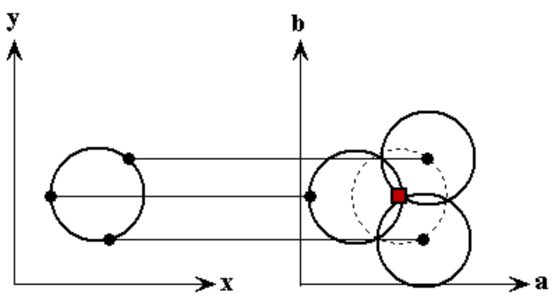
<u>זיהוי מעגלים (נמצא במצגת של התרגול)</u>

נוסחה אלגברית למעגל: $(x-a)^2 + (y-b)^2 = r^2$, (a,b) היא הנקודה של מרכז המעגל r הוא הרדיוס.

ההנחה שהרדיוס ידוע מראש

טרנספורמציה של נקודה למרחב Hough:

במקרה המעגל, המרחב בנוי מציר הa ומציר הb (הנחנו שהרדיוס ידוע), והנקודה הזו יכולה להיות על כל אחד מהמעגלים



שנמצא במרחק הרדיוס r ממנה. לכן נקודה תועבר להיות מעגל.

הנקודה בה הכי הרבה מעגלים נחתכים תהיה הנקודה לה 'הצביעו' הכיהרבה להיות נקודת מרכז המעגל.

במקרה ולא ידוע מה הרדיוס של המעגל שמחפשים:

מרחב Hough הופך להיות מרחב תלת ממדי, בתוספת הציר r. עכשיו למעשה כל נקודת edge point במרחב התמונה יוצרת שטח פנים של *קונוס* מעל המרחב.

מציאת הנקודה בה הכי הרבה חיתוכים במרחב תלת ממדי קשה יותר (ונוגעת גם לעניין RANSAC שנלמד בהמשך), אבל כרגע ניזכר באופציה לחישוב הגרדיאנט בנקודה.אופציה זו חוסכת הצבעה לכל ערך אפשרי, וכך אם נחשב את הגרדיאנט בנקודה כלשהי נוכל להגיד שמרכז המעגל של נקודה זו יכול לשבת בכל מקום אפשרי בקו המאונך לגרדיאנט.

זה הופך את המעגל בטרנספורמציה למרחב Hough לקו.

4. <u>עיבוד תמונה – Image Pyramids</u>

<u>4.1 Gaussian pyramid:</u>

שומרים כמה עותקים של אותה תמונה בגדלים שונים. הגדלים- תמיד יורדים בחצי בכל ממד (אורך ורוחב). (מורידים למעשה בריבוע, לדוגמא: 10*10 יורד ל 5*5). הסיבה שזו סדרה גאומטרית משיקולי נוחות בעיקר. לא נכנס לקורס- תדרים- FFT וכו', וזה אחד מהשיקולים בעניין הירידה בחצי. למה זה חשוב? ככל שהתמונה קטנה המרחק בין נקודות מסוימות בתוך התמונה קטן.

כל גודל של תמונה נקרא **level**.

איך עושים את הפירמידה הזאת?

פתרון נאיבי- לוקחים כל פיקסל רביעי- לדוגמא מוחקים כל עמודה שניה וכל שורה שניה. נקרא **sub sampling.** את בחירת הפיקסל הרביעי אפשר לממש לפי איזה כלל שרוצים, יכול להיות גם אקראי. והתוצאה שאני אקבל תלויה בתמונה. כלומר מה שאני מאבד בתמונה בגודל הקטן יותר תלוי בכלל שבחרנו.

איבוד האינפורמציה הזה בעקבות דגימה "לא חכמה" יכול להתבצע ברמה האחרונה של הsub-sampling אבל יכול לקרות גם בכל הרמות שלפני.

דבר זה נקרה **aliasing**- בעקבות הדגימה מתפספסים שינויים חשובים בתמונה.

אם מסתכלים על תדרים, ודוגמים את הסיגנל החד ממדי במרווחים קבועים, ניתן לראות באופן חד משמעי שבאיזשהו שלב אין דרך למנוע את הaliasing. ניתן לראות במיוחד בתמונות עם תדרים גבוהים- שיש הרבה שינויים חדים. (תמונה במצגת של תדר).

אז לסיכום - בעקבות שינויים שמתרחשים בתמונה או בסיגנל חד ממדי, אם אנחנו דוגמים בצורה נאיבית (sub-sampling) התוצאה שנקבל לא תייצג בצורה אמתית את הסיגנל שדגמנו ממנו.

Reduce- פעולה שמייצרת עוד רמה בפירמידה. (מחלק את הממדים כל אחד ב2). איך עושים את זה? עושים blur- לפי הGaussian פילטר. זה מטשטש את התמונה. ואז עושים את הדגימה, מה שנקרא Gaussian smoothing kernel. ניתן לראות מבחינה מתמטית שכל פיקסל משפיע במשקל מסוים על כמה פיקסלים ברמה הבאה.

דוגמא לפעולת reduce (שקופית במצגת)- עושים פעולת צמצום כל פיקסל שני (הקפיצות האלו בין הפיקסלים זה כבר למעשה ממשש את הדגימה). לכל שלושה פיקסלים צמודים יש 3 מקדמים- 1,2,1. כל פיקסל מחשבים את התדר שלו כפול המקדם ומחברים את כל הביטויים, ואז <u>מנרמלים</u>, כי אנחנו רוצים את הממוצע הממושקל האמיתי (מחלקים במקדמים). יש שקופית במצגת של כמה כל פיקסל תורם – אפשר לעבור על זה.

כמה זיכרון דורש כל התמונות בפירמידה? זה סה"כ סכום של סדרה הנדסית- לא הרבה בכלל.

כל רמה אפשר לבצע בעזרת קונוולוציה אחת.

בדרך כלל כשמדברים על פירמידות בimage- מדברים על Gaussian pyramid- כי זה מה שפותר את הaliasing.

<u>4.2 Laplacian pyramid:</u>

עכשיו השאלה המעניינת- איך חוזרים רמה אחורה?

איבדנו אינפורמציה אז <u>אי אפשר לחזור בדיוק לאותו הדבר</u>. גם בדחיסות מאבדים אינפורמציה. אבל החכמה היא לאבד אינפורמציה שהעין לא רגישה אליו, כלומר אם אני משחזר את התמונה אז התדרים שאיבדתי היו תדרים של שינויים עדינים ככה שזה לא יורגש לעין האנושית.

Reconstruct- "מרפדים" את התמונה בכל פיקסל שני שיהיה 0. ואז משחזרים מכל שלושה פיקסלים (שכל שניים מה הם 0- כי ריפדנו כל פיקסל שני) פיקסל חדש בתמונה המוגדלת.

הנורמליזציה של הblurring kernel יכולה להיות שונה. כלומר כשמקטינים תמונה – מקטינים אותה תמיד כך שהממוצע של העוצמות (גוונים) יהיה אותו הדבר (מקדם 1). זה מה שעושים כשמחלקים בסכום המקדמים של הפיקסלים שבוחרים לפיקסל אחד. אבל כשמגדילים את התמונה- הקבוע נורמליזציה הזה יכול להיות שונה.

התמונה המקורית- נקראת G0 בפירמידה, היא למעשה התמונה של ה expand(G0) - זו התמונה G0 אחרי צמצום ואז שחזור, פלוס ההפרש בין התדרים של שתי התמונות (המקורית וזו שאחר הצמצום)- נקרא L0.

למה זה רלוונטי? כי במקום לשמור את התמונה המקורית, עדיף מבחינת זיכרון לשמור את הL0 ואת התמונה המצומצמת.

כל הפירמידה של הפרשי התמונות בין הexpand לרמה הבאה- נקראת: **Laplacian pyramid**. התמונה ברמה האחרונה של הפירמידה הגאוסיינית מועתקת לרמה האחרונה בפירמידה של ההפרשים. ולמעשה, כדי להגיע לרמה הראשונה בפירמידה הגאוסיינית- כלומר לתמונה המקורית- צריך לשמור את הפירמידה של ההפרשים בלבד.

ובגלל שרוב הערכים בפירמידה הזו הם 0- ניתן לעשות קוונטיזציה ולשמור רק את הערכים שמעל ל-0.

<u>4.3 Blending :</u>

Mask בינארי- 1/0 כלומר שחור/לבן. מגדיר איזה חלק לוקחים מתמונה אחת ואיזה חלק לוקחים מאחרת.

מושגים שחשובים לדעת כדי להבין את השפה של המקצוע:

Ghosting – כשרואים את שתי התמונות אחת על גבי השניה. נוצר כתוצאה מערבוב של שני פיקסלים פיקסל מתמונה. יש אפשרות לקחת window size שמייצג איזה אחוז ערך לוקחים מכל פיקסל- ואז ליצור הדמיה של הדרגה ברמת השקיפות של תמונה אחת על גבי השנייה. ככל שהטווח גדול יותר השקיפות גדולה יותר והghosting מורגש יותר. הwindow size מתחיל ברמה גבוהה של תמונה אחת ומקטין את הערכים שלה ככל שהערכים של התמונה השנייה גדלים והיא מורגשת יותר.

אם הטווח מאוד מאוד קטן – יש מעבר חד מאוד בין התמונות ורואים את "התפירה" בין שתי התמונות. כמו ב Visible seams – כשלוקחים פיקסל רק מתמונה אחת (יש דוגמא במצגת על חצי חצי)

הבעיה: בהינתן שתי תמונות, אני רוצה לבחור בקו התפר של חיבורן: גודל חלון שבו אני אקח פיקסלים משתי התמונות, ולייצר איזושהי פונקציה שתיתן לבחור משקל כך שאם נערבב את ערכי התמונות נקבל מעבר נעים לעין, "נקי".

כדי להימנע מghosting – שקיפות של תמונה על גבי השנייה, ומ seams- קו תפר מורגש, אנחנו צריכים להתחשב בשינויים החדים שיש בתמונה.

אבל איך מחשבים את גודל החלון לפי השינויים החדים שיש בתמונה?

אלגוריתם:

בהינתן שתי תמונות, A ו B. ומקבלים תמונת mask, תמונה בינארית (שחורה/לבנה). לצורך העניין 0 זה מה שלוקחים מ A ו1 מה שנלקח מB .

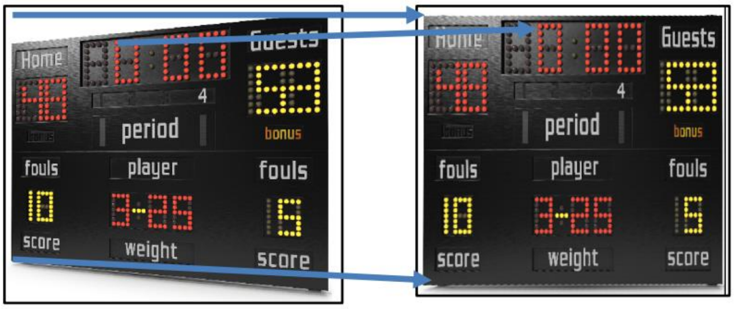
- בונים פירמידת הפרשים Laplacian , L_A L_B
- בונים פירמידה גאוסיינית מהmask, G_M
- יוצרים פירמידת הפרשים (Laplacian) מפירמידת ההפרשים L_A L_B ומהפירמידה G_M . לפי נוסחא לכל רמה (יש את הנוסחה במצגת).
- נוצרת מכל השלבים פירמידה של הפרשים של שתי התמונות בחלוקה לפי הmask.

למעשה כך נוצרת תמונה שנראית חלקה.

5. Geometry: Image warping & 2D transformations

5.1 Image warping (Image to image projection):

מיפוי מתמונה אחת לתמונה אחרת, או שינוי הקואורדינטות של הפיקסלים בתמונה. (להבדיל מ׳פילטרינג׳ על תמונה: שינוי של ערך הintensity של הפיקסלים)



סוגי מיפויים – 2Dב transformations:

- Translation- תזוזה של התמונה במערכת. $translation(a,b): (x,y) \rightarrow (x+a, y+b)$
- Scale- שינוי גודל התמונה. $scale(a,b): (x,y) \rightarrow (ax, by)$
- Rotation- סיבוב. $rotation(\theta): (x,y) \rightarrow (x \cdot \cos(\theta) + y \cdot \sin(\theta), -x \cdot \sin(\theta) + y \cdot \cos(\theta))$
- Mirror- למעשה זהו scaling פי 1 בסימן שונה בציר שבו נרצה שתהיה ההשתקפות.

$$mirror(against\ y\ axis): (x,y) \rightarrow (-x, y)$$

$$mirror(against\ x\ axis): (x,y) \rightarrow (x, -y)$$

- Shear- (״גזירה״), עיוות מקבילי של התמונה. $shear(a,b): (x,y) \rightarrow (x+ay, y+bx)$

שילוב של טרנספורמציות מוכרות:

- Rigid- Translation + Rotation
- Similarity- Translation + Rotation + uniform Scale
- Affine- Translation + Rotation + Scale + Shear

בכל הטרנספורמציות הללו אפשר לייצג אותן במטריצה דו ממדית (כיוון שאנו רוצים בסוף לקבל תמונה. אם נרצה לקבל עבור כל וקטור בשני ממדים וקטור אחר אחרי טרנספורמציה בדו ממד- נצטרך להכפיל במטריצה 2 על 2).

רק translation אין אפשרות לייצג את הטרנספורמציה במטריצה 2x2:

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x+a \\ y+b \end{bmatrix}$$

הפתרון: שימוש במערכת קואורדינטות הומוגנית.

מערכת קואורדינטות הומוגנית: מיפוי מ R^n לממד R^{n+1}

וזה יעשה כך:

$$\begin{bmatrix}1 & 0 & a\\0 & 1 & b\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}x+a\\y+b\\1\end{bmatrix}$$

וכשנרצה לחזור חזרה לקואורדינטות לא הומוגניות נעשה מה שנקרא inverse mapping :

$$(X,Y,W)\rightarrow\left(\frac{X}{W},\frac{Y}{W}\right)$$

<u>דרגות חופש במטריצות המוכרות:</u>

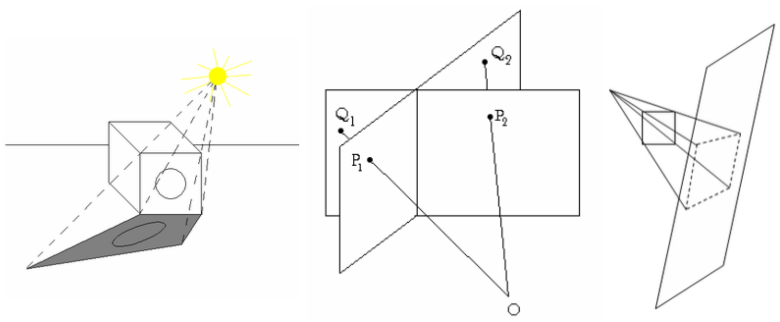
- Translation משתמשים במערכת קואורדינטות הומוגנית – 2 דרגות חופש
- Rigid = translation + rotation - 3 דרגות חופש
- Similarity = translation+ rigid + scaling – 4 דרגות חופש
- Affine = translation + rigid + scaling + skew (מתווספת הטיה שניתנת לביצוע בשני כיוונים) – 6 דרגות חופש

<u>5.2 הומוגרפיה – העתקה פרויקטיבית:</u>

הומוגרפיה זו טרנספורמציה חשובה (Homography – General projective transformations) .

זוהי טרנספורמציה בין שתי תמונות שנלקחו מאותו מרכז אופטי (מאותו מיקום שלו בעולם) ויש לה 8 דרגות חופש (יש 8 נעלמים, לכן צריך להציב 4 נקודות כדי לקבל מערכת של 8 משוואות).

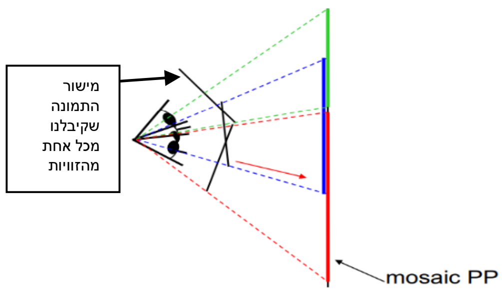
הומוגרפיה עונה על השאלה- מה היה קורה אם היינו שמים את המישור של התמונה במיקום אחר, בזווית אחרת ביחס לעולם? כמו להסתכל על טלוויזיה מהצד- איך התמונה שנקבל מהמסך תראה.



<u>שימוש ראשון</u> : ממפים נקודה בעולם בתלת ממד לנקודה בדו ממד (על ידי הטלה של הנקודה בתלת ממד לעולם דו ממדי), ותחת טרנספורמציה דו ממדית ניתן לדעת מה הייתה ההטלה של הנקודה בתלת ממד למישור דו ממדי אחר.
<u>שימוש שני</u> : בהינתן משטח, שצילמנו משתי זוויות. נוכל לדעת איפה כל פיקסל בתמונה הראשונה נמצא גם בתמונה השנייה בעזרת הומוגרפיה. (להבנה רעיונית- אם יש לי תמונה עקומה ואני רוצה לקבל אותה בצורה ישרה, אפשר לעשות הומוגרפיה על התמונה הזאת ביחד עם תמונה וירטואלית שבה הפינות של התמונה העקומה נמצאים במקביל. כמו מה שקורה בcamscanner)

למה בכלל צריך הומוגרפיה?

במידה וצילמנו סדרת תמונות מכמה זוויות (כמו מצלמה על חצובה שמזיזים רק קצת בזווית), בעזרת מיפוי הומוגרפיה נעביר את כל התמונות למישור אחד (פנורמה!) נחזור לכך בהמשך כאשר נתעסק בimage stiching.



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

טרנספורמציה בתלת ממד זה תמיד רוטציה והזזה (טרנספורמציה אוקלידית) אבל כשעושים הטלה לדו-ממד בתמונה זה כבר לא תופס הכל. הטרנספורמציה שתופסת הכי הרבה תזוזות בדו ממד זאת הטרנספורמציה ההומוגרפית.

הומוגרפיה בין שתי תמונות אפשר למצוא בשתי שיטות:

1. השיטה הלא הומוגנית– לא בקואורדינטות הומוגניות (מורידים את ה -1 האחרון במטריצה) ופותרים בשיטת ה LS – Least Square
2. השיטה ההומוגנית – באמצעות Singular Value Decomposition – SVD

ההתחלה בשתי שיטות דומה:

מספר דרגות החופש של ההומוגרפיה היא 8, כלומר צריך 4 פיקסלים מכל תמונה כדי לפתור את מערכת המשוואות (כל פיקסל נותן שני נעלמים לx ולy וכל זוג פיקסלים נותן שתי משוואות). על ידי פתיחה של המטריצה כפול הפיקסלים משתי התמונות מקבלים שלוש משוואות, אנחנו מחלצים את הפיקסלים של תמונה אחת ורוצים למצוא את הנעלמים ה-h.

$$x'_i = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$
$$y'_i = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

בסופו של דבר אנחנו רוצים להגיע לתצוגה מטריציונית של וקטור נעלמים · פיקסלים.

על ידי העברת אגפים, הופכים את המשוואות לתצוגה מטריציונית כאשר כל משוואה בעלת 9 נעלמים (עמודות).

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix}}_{\substack{\mathbf{A} \\ 2n \times 9}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\substack{\mathbf{h} \\ 9}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\substack{\mathbf{0} \\ 2n}}$$

מכיוון שקיבלנו מערכות משוואות הומוגנית (הפתרון הוא וקטור אפס), יוצא שקיבלנו משוואת מינימיזציה שהפתרון האופטימלי שלה הוא מטריצת אפסים. זו בעיה כיוון שאנחנו מחפשים מטריצת מעבר הומוגרפית שאינה אפסית (אחרת מה עשינו..).

$$\text{minimize } \|\mathbf{Ah} - \mathbf{0}\|^2$$

הפתרון לבעיה: (וזה מה שתמיד עושים במקרה בו פותרים משוואה **הומוגנית** על ידי האלגוריתם של **least square**)

הוספנו אילוץ למשוואה: **הפתרון, הנורמה, חייב להיות שווה ל- 1**. כלומר מבין כל הווקטורים שהנורמות (כפול המטריצה A) שוות ל-1: תמצא את הווקטור המינימלי. זה לא פוגע במרחב הפתרונות ומוריד פתרון ולידי מלבד הפתרון 0. זה לא פוגע במרחב הפתרונות - כיוון שהומוגרפיה אינווריאנטית (שקולה) לscaling, ניתן להגיד שלא שינינו כלום בפתרון אם הכפלנו את הנורמה בα שהוא סקלר כלשהו. ולכן, ברגע שאנחנו מצמצמים את מרחב הפתרונות להיות 1 בעצם הבטחנו שהפתרון יהיה במרחב כל המספרים הממשיים כל עוד הם לא שווים ל0 (כי 1 כפול α כלשהו שנבחר שהוא לא 0 לא יצא 0).

ניתן לקבוע אילוץ של נורמה שרירותי, כל עוד אפשר להכפיל בסקלר כלשהו ולהגיע לפתרון האופטימלי.

הפתרון למשוואה תחת האילוץ:

שיטה ראשונה: **שיטת כופל לגראנז'** (צריך למצוא פתרון מינימלי של משוואה תחת אילוצים, פשוט מכניסים את האילוצים כעוד פרמטר).

שיטה שנייה: פיתוח של SVD –

נהפוך את המטריצה A לריבועית על ידי הכפלתה במשוחלפת: A^TA ונמצא לה את הווקטורים העצמיים.

נבחר את הווקטור העצמי עם הערך העצמי הכי נמוך וזה יהיה וקטור הנעלמים m.

הפתרון מבוסס על:

- שיטת SVD - singular value decomposition שמייצרת פירוק של המטריצה A: כל מטריצה ניתן לבטא באמצעות כמה מטריצות מוכפלות: $A = UDV^T \mid D\ is\ diagonal, U\ and\ V\ orthogonal$ הערכים ב D מסודרים בהתאם לערכם בסדר יורד.
- הכפלה במטריצה המורכבת מווקטורי יחידה אורתוגונליים זה לזה לא משנה את הגודל של המטריצה. לכן ניתן לבצע את המהלך הבא:
$$||Am|| = ||UDV^Tm|| = ||DV^Tm|| = ||DV^Tm||$$
והעברנו את המגבלה לצורה כזו: $||m|| = ||V^Tm|| = 1$
- אם הערכים בD מסודרים בסדר יורד, אז הווקטור שיקטין את התוצאה של הגודל שלעיל הוא וקטור יחידה שרק האינדקס האחרון הוא 1. כלומר V^Tm צריך להיות הווקטור הזה.
- $m = VV^Tm$ כיוון שעבור מטריצות אורתוגונליות המטריצות המשוחלפות שלהן זהות למטריצות ההופכיות שלהן. למעשה VV^T היא מטריצת היחידה.
- לסיכום: m הוא הווקטור האחרון במטריצה V.
- $A^TA = VD^TU^TUDV^T = VD^TDV^T = VD^2V^T$
- V הם הערכים העצמיים של A^TA ועל ידי בחירת הווקטור האחרון בV אנחנו מוציאים את הווקטור האחרון בA^TA.

(בקוד: mat = svd(A^TA); vector = mat[:,:,3] **)**

נשים לב שמקבלים **וקטור** פתרון ואנחנו מחפשים מטריצה של 3 על 3 (מטריצת ההומוגרפיה)

נעשה reshape למטריצה (3,3), וכיוון שקונבנציונלי שהערך האחרון במטריצה הוא 1 (לא באמת קריטי איזה ערך 1 אבל כך נהוג)- נחלק את כל המטריצה בערך זה.

5.3 <u>Forward/backward warping</u>:

נתייחס הרבה פעמים להעתקות על תמונות כאל 'עטיפת' תמונות – image warps, ויהיה לאינטרפולציה חלק בזה.

יש שתי דרכים לבצע warping מתמונה לתמונה – forward (דרך לא טובה) ו backward (הדרך הטובה).

<u>Forward warping</u>- בהינתן תמונה א' שעליה נרצה לבצע העתקה וליצור ממנה תמונה ב'. ניקח כל פיקסל מתמונה א' והקואורדינטה שתצא לאחר הפעלת ההעתקה על הקואורדינטה של הפיקסל תצבע באותו הגוון. 'שולחים' כל פיקסל מתמונה א' למיקום שלו בתמונה ב'.

זאת הדרך הלא טובה כיוון שיש בעיה – מה קורה אם המיקום שיוצא לאחר ההעתקה הוא מיקום בעייתי – לדוגמא חצי הדרך בין שני פיקסלים.

הפתרון לזה בforward warping הוא ל'מרוח' את גוון הפיקסל שנופל בין קואורדינטות לכל הפיקסלים שנמצאים סביב המיקום שבו נפל.

<u>Backward warping</u>- דרך שנייה לבצע את ההעתקה, היא להסתכל על התמונה ב' שאנו רוצים שתצא מהפעלת ההעתקה, ולהבין **מאיפה** הפיקסל הגיע בתמונה א'. כלומר לבצע את ההעתקה ההופכית.

הבעיה עכשיו- מה קורה אם פיקסל הגיע ממיקום שנמצא בין שני פיקסלים בתמונה א'?

הפתרון בbackward warping: אינטרפולציה בין הערכים סביב המיקום ש'נפל' בתמונה א'.

אינטרפולציה (interpolation) זה מושג כללי המבטא תהליך של הערכת פלט של פונקציה בין נקודות שיודעים מראש מה הפלט מהפונקציה שלהן.

כאן אנו מתייחסים לתמונות כעל פונקציה של גוונים בהינתן מיקום, וניתן להשתמש בכל מיני שיטות של אינטרפולציה כדי להעריך את הגוון של הפיקסל שמגיע ממיקום לא ברור בתמונה א׳. לדוגמא, מה שנקרא bilinear interpolation והנוסחה שלו היא זו:

$$\begin{aligned} f(x,y) = \; & (1-a)(1-b) && f[i,j] \\ & +a(1-b) && f[i+1,j] \\ & +ab && f[i+1,j+1] \\ & +(1-a)b && f[i,j+1] \end{aligned}$$

פנורמה (או mosaic)- ניתן ליצור ממספר תמונות הנלקחות מאותו מרכז אופטי. צריך לחשב עליהם טרנספורמציה של הומוגרפיה (טרנספורמציה ממשטח תמונה אחד למשטח תמונה אחר), בעזרת הנקודות הזהות באותן תמונות. זה יצור חפיפה בין התמונות וכדי שזה יראה חלק צריך לעשות עליהם blending.

Motion estimation .6

נושא כללי : motion estimation technique : ניתוח תנועה של אובייקטים בסדרת תמונות.
מוטיבציה : סרט וידאו מורכב מסדרת תמונות, זיהוי תנועה בסדרה הזו.
שיטות : שיטה ראשונה וכללית יותר- זיהוי (detect) תכונות בתמונה ומיקומם בסדרה עקבית של תמונות.
שיטה שניה : direct, dense methods- שיטה ישירה של חישוב תנועה בכל פיקסל בתמונה. וכחלק מזה זה הoptical flow, כאשר מחשבים תנועה עבור כל פיקסל נוצרת תמונה שממחישה את ה"זרימה האופטית של התמונה".

6.1 __Optical flow:__

עבור כל פיקסל מחשבים את התזוזה שלו במרחב. נשים לב שעבור אזורים שנעים קרוב יותר אלינו נקבל חיצים גדולים יותר- כי התנועה הייתה יותר גדולה ביחס לתנועה מרוחקת יותר.
Optical flow מחשבת את התנועה הגלויה למצלמה- עבור פיקסלים שלא ניתן להבחין בשינוי הבהירות שלהם ביחס לסביבה המרחבית שהם נמצאים בה לא נראה תמונה ולכן להם לא תחושב תנועה.

הבעיה המרכזית שoptical flow מתמקדת בה- איך לשחזר את וקטורי התנועה של אובייקטים הנמצאים בשתי תמונות שנלקחו בזמנים עוקבים.
אם נפרמל את זה קצת : בהינתן פיקסל בתמונה I בזמן t : I(x,y,t), תחפש פיקסלים **באותו הגוון** והנמצאים **קרוב** לאותן קואורדינטות בתמונה בזמן העוקב : I(x,y,**t+1**).
עכשיו צריך להבין- מה זה אומר פיקסל באותו הגוון? ומה זה פיקסל קרוב?
ההנחות המוקדמות שלנו הן שהצבעים של האובייקט נשארים זהים (בתמונה בגווני אפור זה יהיה אותם גווני בהירות), ושנקודות בשתי תמונות שנלקחו בזמנים עוקבים לא זזות כל כך מהר.
אם ננסח אלגוריתם פשוט למציאת התזוזה בין שתי תמונות על סמך עקרון הבהירות : בהינתן שתי תמונות I_1 ו I_2 נחפש טרנסלציה (הזזת הפיקסלים בתמונה) כך שפונקציית שגיאת-הריבועים (squared error) תוציא ערך מינימלי.
אם נוסיף על כך את עקרון התזוזה הקטנה, ונניח כי תזוזה קטנה = תזוזה של פחות מפיקסל אחד, נוכל להשתמש בטור טיילור.

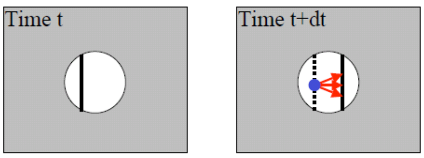
6.2 __אלגוריתם Lucas-Kanade להערכת תנועה בין תמונות:__

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

הנתונים שאנו צריכים לדעת כדי לחשב באמצעות LK :
הפרש התמונות (יקרא I_t)
הנגזרת של התמונה השנייה בכיוון x (יקרא I_x)
הנגזרת של התמונה השנייה בכיוון y (יקרא I_y)

בינתיים זה יוצר לנו משוואה אחת עם שני נעלמים (u , v). מה שזה אומר בפועל – נוצרת לנו בעיה שנקראת בעיית הצמצם, "the aperture problem".

__בעיית הצמצם :__ בהינתן חלון פיקסלים, תנועה יכולה להתפרש לכיוון מסוים בעוד היא בפועל זזה לכיוון שונה, וזאת כיוון שמסתכלים דרך חלון פיקסלים מצומצם ולא על כל התמונה.



נוסיף מגבלות על מנת לפתור את הבעיה :

1. נוסיף פילטר שמטשטש את התמונה
2. נניח שכל התמונה זזה בכיוון אחד

בזכות המגבלה השנייה עכשיו נוכל להוסיף כמה זוגות פיקסלים שנרצה למשוואות, כי לכל הפיקסלים אותה התזוזה – אותם u ו-v.

$$\underset{25\times 2}{\underbrace{\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix}}_{A}} \underset{2\times 1}{\underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d}} = - \underset{25\times 1}{\underbrace{\begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}}_{b}}$$

את מערכת המשוואות פותרים באמצעות least square :

$$minimize\ ||Ad-b||^2$$

7. ראייה ממוחשבת – Stereo

מתחילים לעבור את התפר שבין עיבוד התמונה לראייה ממוחשבת.

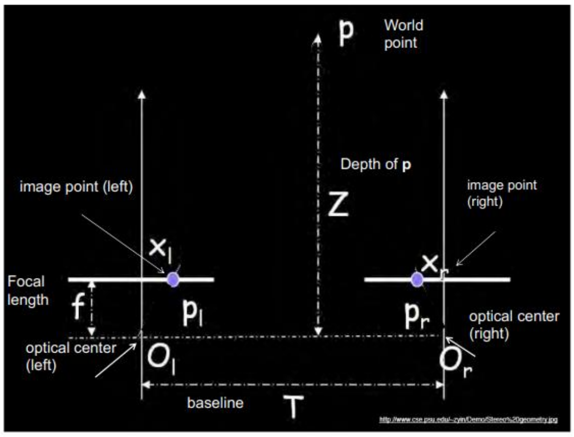
הראייה התלת ממדית שיש לנו כבני אדם (היכולת להבדיל בין עומקים) זה בזכות זה שיש לנו שתי עיניים.

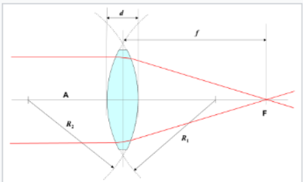
המוח שלנו מחבר את הנקודות שהוא מקבל משתי העיניים בצורה חישובית מאוד "גסה" ובסיסית. מה שיוצר את העומק זה תזוזה של מספר פיקסלים שמקובץ בצורה מסוימת (דוגמא בשתי תמונות עם רעש שהזיזו פיקסלים שמקובצים בצורת 2 ימינה בתמונה השנייה, המוח מזהה את ה"עומק" של הצורה 2)

Stereo= אנחנו רוצים למצוא את העומק, את הממד השלישי- על ציר הz.

מסתכלים על חתך בתלת ממד- על נקודה שנמצאת על מישור שנמצא במקביל לעדשת המצלמה. אנחנו לוקחים נקודה בעולם p, ושתי נקודות שמתקבלות מהטלה של הנקודה (בעולם התלת ממדי) לשתי תמונות (כמו שתי מצלמות- מקביל לשתי עיניים) אחת מצד שמאל ואחת מצד ימין- x_l x_r בהתאם. (הנקודות במישור שמקביל למוקד)

המרחק בין שתי עדשות המצלמה הוא T, ונקרא baseline. המרחק בין מישור התמונה למקום שבו כל הקרניים עוברות נקרא f.





איור 1: דוגמה של עדשה מרכזת - קרניים מקבילות (אדומות) מגיעות משמאל במקביל לציר האופטי הראשי, A, ומתמקדות בנקודת המוקד, F (באיור זה מוראה המוקד הימני בלבד). f הוא מרחק המוקד, d הוא עובי העדשה, ו-R הוא רדיוס העיקום של העדשה (עדשה כדורית).

המטרה: למצוא את העומק.

יש לנו שני משולשים דומים- אחד x_r x_l וpl. השני- מרכז המוקד של מצלמה ימין, של מצלמה שמאל וpl.

עכשיו עם קצת גאומטריה של משולשים דומים נוכל למצוא את ממד העומק- שזהו הגובה במשולשים הדומים- קו הz. נשים לב שx_l נמצא בחלק ה<u>שלילי</u> של מערכת הקואורדינטות שקבענו בתמונות. אנחנו קבענו שO_R הוא ה(0,0) של התמונה (וככה בהתאם לתמונה השמאלית) ולכן x_r נמצא בחלק החיובי.

- x_r x_l זה הפיקסלים בתמונה (ההיטל של הנקודה בעולם האמיתי לדו-ממד)

חשוב שברגע שקבענו את הסימנים של הצירים- זה יהיה סימטרי בשתי התמונות.

לכן הערך של הבסיס של המשולש הקטן יותר יהיה : $T+x_l-x_r$, הגובה יהיה : $Z-f$ וסה"כ המשוואה כדי

$$\frac{T+x_l-x_r}{Z-f}=\frac{T}{Z}$$

למצוא את z :

מחלצים את z, ומקבלים את העומק כפונקציה של focal length-f . אנחנו מניחים שהוא אותו הדבר בשביל שתי המצלמות (אחרת צריך לעשות נוסחא שונה).

למעשה העומק הוא יחסי למרחק בין הפיקסלים x_r x_l, והמרחק הזה נקראה **disparity**- הפער.

Disparity-map: תמונה בה הפערים בין אותה הנקודה, בשתי תמונות המצולמות משתי זוויות שונות ועל אותו scanline, מבוטאים באמצעות intensity בפיקסל. מיקום הפיקסל עצמו הוא מיקום הפיקסל בתמונה אחת (שמאל) והערך שלו- ככל שהפער יותר משמעותי בין שתי התמונות הוא יהיה יותר לבן. (יותר intensity). ***נקודה חשובה:*** למעשה בלי T ובלי f ניתן לבטא כבר משהו לגבי העומקים בתמונה.

← שאלה: מה ההשפעה של שינוי בdisparity במקרה שהאובייקט נמצא מאוד קרוב, ולחלופין כשהאובייקט נמצא מאוד רחוק? (לחשוב על מה קורה גם בעיניים שלנו באמת כשמקרבים משהו שמסתכלים עליו ממרחק מזערי)

מבחינה מתמטית: $z = \frac{T}{d}$ מה קורה כשמקבעים את שאר המשתנים במשוואה ורק משחקים עם הפרמטר d.

✓ תשובה: ככל שהאובייקט יותר רחוק- הdisparity יותר קטן. (המכנה יותר קטן – המונה יותר גדול ולכן ה Z יותר גדול). ככל שהאובייקט יותר קרוב- ה disparity יותר גדול (יחס הפוך) ולכן העומק באמת יותר קטן.

ככל שהאובייקט מתרחק יותר קשה להעריך את העומק. לכן בעולם האמתי, המצב היחידי בו אם נסתכל מכל עין בנפרד – נקבל את אותה התמונה, יהיה אם נסתכל על אובייקט רחוק כמו האופק. למה זה קורה מבחינה גאומטרית? מכיוון שהקווים של המשולש בין נקודות הבסיס לקדקוד הולכים ונהיים מקבילים כשהמרחק בין הנקודה במציאות לנקודות המוטלות בתמונה הולך וקרב לאינסוף. **וזהו החיסרון בstereo.**

אנחנו בתור אנשים לא בנויים להבדיל בין מרחקים של עומק כשהאובייקטים רחוקים מאתנו מאוד. לכן כשאובייקטים מאוד רחוקים ניתן להתייחס אליהם כאל מישור אחד.

<u>איך עושים stereo בסיסי:</u>

מקבלים שתי תמונות ונקודה, מניחים שהנקודה נמצאת על אותו קו- אותו scanline בשתי התמונות.

צריך למצוא את אותה הנקודה גם בתמונה השנייה- עוברים על אותו הקו בתמונה השנייה עד שמוצאים את אותה הנקודה. מחשבים את ערך הפער, וקובעים את העומק עם f וt.

- איך מוצאים את הנקודה עצמה? לוקחים חלון סביב הנקודה, ועוברים על הקו ובעזרת פונקציה מסוימת (יש כל מיני אפשרויות, לדוגמא- normalized cross correlation) עד שמוצאים נקודה המתאימה ביותר.

דוגמאות לפונקציות:

- **SAD:** W זה החלון. לוקחים את כל האינדקסים ששייכים לחלון וסוכם את כל ההפרשים בכל חלון. לוקח את החלון עם ההפרשים המינימליים. (: להבין יותר)
- **Norm.corr:** פה לוקחים דווקא את הערך המקסימלי שהתקבל מבין כל החלונות.

האפקט של גודל החלון שנבחר: ככל שהחלון יותר קטן יש יותר התאמות, נוכל לקבל יותר פרטים, כי עבור כל חלון נקבל עומק שונה. הבעיה בזה- זה יותר "רועש". ככל שניקח סביבה יותר גדולה (חלון יותר גדול) יותר סיכוי שיהיו יותר שינויים ולכן הסיכוי למצוא את כל השינויים יהיה יותר נמוך. אבל לכל החלון נקבל אותו העומק ולכן למעשה יוצרו פערים במפת עומק.

איפה נכשל החיפוש בעזרת החלון? כשיש משטחים לגמרי באותו צבע בתמונה המקורית.

<u>לפי תצפיות בעולם האמיתי, נוסיף מידע מוקדם שנכניס לחישובי עומק בצילום stereo:</u>

- <u>ייחודיות:</u> לכל נקודה מתאימה נקודה אחת בלבד בתמונה השנייה.
- <u>יחס:</u> בין כל שתי נקודות בתמונה אחת מתאימה נקודה שנמצאת בין שתי הנקודות התואמות בתמונה השנייה.
- <u>smoothness:</u> אנו מניחים שעומק של שתי נקודות הנמצאות אחת ליד השנייה בתמונה יהיה דומה. השינויים בעומק איטיים יחסית. (ברוב הנקודות)

מבין כל הפתרונות האפשריים נרצה למצוא את הפתרון שמקיים כמה שיותר את האילוצים האלו.

מידול האילוצים (ניסוח שלהם בצורה פורמלית כאלגוריתם):

- <u>data term</u>: אבר באלגוריתם שמודד התאמה של כל פיקסל בתמונה א לכל פיקסל בתמונה ב.
- <u>Smoothness term</u>: מודד את ההתאמה של החלון של הפיקסל בתמונה א לחלון של פיקסל בתמונה ב.

8. Epipolar Geometry (Two views geometry)

כהמשך לנושא בו למדנו על stereo- שלמעשה נותן לנו כלי להעריך את הdisparity – ״פער״ בין המיקומים של שתי נקודות בשתי תמונות של עצם משתי זוויות שונות, וזה פרופורציונאלי לעומק של הנקודה. אבל איך מוצאים את ה״פער״ הזה? בהינתן נקודה אחת בתמונה השמאלית אני צריכה למצוא אותה בתמונה הימנית. האינטואיציה אומרת שאם אנחנו יודעים משהו על הנקודה השמאלית, היא לא יכולה להיות בכל מיקום בתמונה הימנית אלא חייבת להיות תחת אילוצים מסוימים בהתאם למיקום של המצלמה השנייה ובהתאם למיקום של הנקודה עצמה.

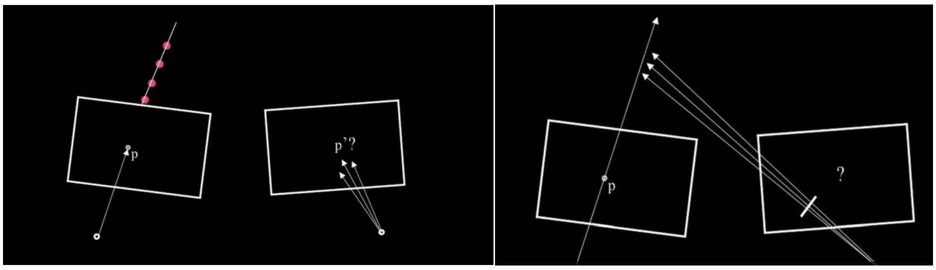
אנחנו נדבר עכשיו על אילוצים אלו.

במקרה של שתי מצלמות- הן יכולות להיות מקבילות על אותו קו אבל הן יכולות להיות גם זוויתיות זו לזו.

כאן נכנסת הגיאומטרי האפי-פולארית.

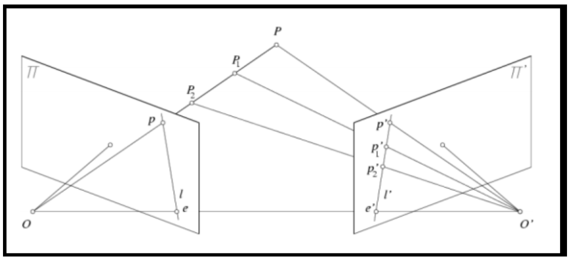
נסתכל על הדבר הבא:

נקודה P בתמונה השמאלית הבאה יכולה להיות בסצנה המציאותית (המצולמת) בכל נקודה אפשרית על הקרן (הקו) שחותכת את מרחב התמונה. לכן הקרן שחותכת את מרחב התמונה הימנית בנקודה 'P יכולה לחתוך את הקרן של הנקודה P בכל נקודה אפשרית.



תזכורת: בגיאומטריה פרספקטיבית קווים מוטלים לקווים.

לכן הקו שמכיל את מרכז ההקרנה ואת הנקודה P בתמונה השמאלית חייבת להיות מוטל לקו ותמונה הימנית.



הקו הזה שמוטל על התמונה הימנית נקרא ״the epipolar line״, קו זה נקבע למעשה על ידי המישור שנקבע על ידי שתי נקודות ההקרנה (center of projection) של שתי התמונות והנקודה האמיתית במציאות. כמובן שגם הקו בתמונה השמאלית נקרא כך, וכל נקודה שנמצאת על הקו הזה חייבת להימצא על הקו המתאים בתמונה השנייה.

וזה למעשה האילוץ שאנחנו מחפשים.

<u>כמה מושגים בגיאומטריה אפי-פולארית:</u>

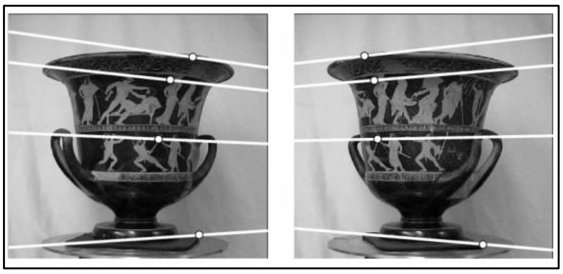
- <u>Baseline:</u> הקו שמחבר בין שני מרכזי מצלמה
- <u>Epipolar plane:</u> המישור הנקבע על ידי הbaseline ועל ידי הנקודה האמיתית.
- <u>Epipolar line:</u> החיתוך של המישור האפי-פולארי (epipolar plane) ומרחבי התמונות- תמיד יהיו בזוגות.
- <u>Epipole:</u> נקודת החיתוך של ה baseline ושל מרחבי התמונות. אנו מניחים שהתמונות גדולות מספיק כדי שיחתכו גם את קו-הבסיס. זה אומר שכל epipolar-line יחתוך גם הוא את הepipole.

למה כל כך חשוב האילוץ האפי-פולארי? כי למעשה זה מצמצם את החיפוש של הנקודה המתאימה לחיפוש **בממד אחד**.

איך הקווים האפי-פולאריים יראו?

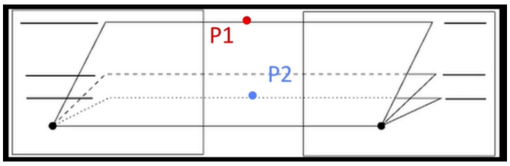
אז יש לנו את המקרה הכללי של שתי מצלמות המונחות בזווית כך שהקווים של מרכז המצלמה בסוף יחתכו (verged case), ויש את המקרה בו שתי המצלמות מונחות במקביל על אותו הקו.

במקרה הראשון כששתי המצלמות מונחות בזווית נראה דוגמא :



נשים לב שהקווים האפי-פולאריים נפגשים בepipole באינסוף (כל שני קווים שאינם מקבילים נחתכים לבסוף) כלומר זה מונח מתמטי יותר כיוון שהתמונות נחתכות לפני שמספיקים לראות את החיתוך הזה של כל הקווים. אם נגדיל את התמונה נוכל לראות שבסוף כל הקווים יחתכו בנקודה זו.

במקרה השני :



כאן הקווים מקבילים, ולכן הepipole קיים רק באנסוף.

9. מציאת פי׳צרים זהים בשתי תמונות

מוטיבציה: כדי שנוכל לחשב הומוגרפיה בין שתי תמונות עלינו להחזיק מראש בזוגות של פיקסלים תואמים בשתי התמונות.

ישנן שתי שיטות:
1. שיטה ישירה שמתבססת על חישוב כלשהו ביחס לחלון פיקסלים (מה שלמדנו כבר – LK, cross correlation)
2. שיטה שמתבססת על מציאת תכונות בסיסיות (פיצ׳רים) בשתי התמונות ועל פיהן לחשב את ההתאמה

השיטה הראשונה פחות מתאימה לתמונות שרק בחלק קטן שלהן חופפות (כמו בתמונות פנורמה).
השיטה השנייה מתבצעת בצורה כללית כך: זיהוי נקודות הפיצ׳רים (נקודות בולטות) בשתי התמונות, ועל פיהן לבנות descriptor – וקטור המאפיין את הפיצ׳רים , למצוא את הנקודות המתאימות בשתי התמונות

9.1 SIFT:

מציאת features – באמצעות SIFT האלגוריתם הזה מוצא נקודות מיוחדות בתמונה, לכל אחת מהנקודות האלו הוא בונה וקטור של 128 (descriptor- קידוד של הפיקסל המיוחד – ה feature, שמאפשר להשוות ביניהם) ומסוגל למצוא את הנקודות האלו בתמונה אחרת.

9.2 RANSAC:

אגלוריתם RANSAC (RANdom SAmple Consensus) מוצא סט אופטימלי של נקודות מתוך הנקודות הנתונות שלפיו יש לחשב את הטרנספורמציה בשיטת ה LS -(least square). סט זה נקרא סט הקונצנזוס, והמטרה היא שסט זה יכלול רק נקודות שהן inliers במודל ולפיכך חישוב הטרנספורמציה לא ייפגע עקב נקודות ouliers.

הדרך בה האלגוריתם מוצא את סט הקונסנזוס הוא בחיפוש איטרטיבי רנדומלי אחר תת-סט מסט הנקודות המקורי. מניחים כי נקודות אלה הן נקודות inliers וההנחה הזאת נבחנת כנגד אוסף הנתונים ומקבלת "ציון". תהליך זה חוזר על עצמו מספר קבוע מראש של פעמים. הרעיון הוא לקבוע מספר זה כך שבהסתברות גבוהה, באחד מהפעמים ייבחרו נקודות השייכות ל- inliers בלבד, ולפיכך ההנחה תהיה נכונה והציון יהיה גבוה. לפיכך, לבסוף נבחר המודל שהביא לציון הגבוה ביותר מבין כל המודלים שנבחנו, והנקודות מהן חושבו המודל נבחרות להיות נקודות הקונצנזוס.

הרעיון: האלגוריתם מקבל מודל, בדוגמא שלנו זה יהיה קו, כמה נקודות מגדירות מודל (2 נקודות מגדירות קו), ומה מרחק השגיאה מהמודל.

לדוגמא: האלגוריתם בוחר בכל איטרציה שתי נקודות שמגדירות קו (או ארבע נקודות שמגדירות הומוגרפיה) – ובודק כמה נקודות (דגימות) נוגעות בו.

הנחה: בכל מידע (data) יש נקודות שהן inliers והפיזור שלהן יכול להגדיר איזשהו מודל (כולל עם רעש מסוים), ונקודות שהן outliers שלא מתאימות למודל.

<u>inliers</u>- כל הנקודות שנמצאות מספיק קרוב למודל שבחרנו

<u>outliers</u>- כל הנקודות שלא

← <u>האלגוריתם בכל איטרציה:</u> בוחר 4 נקודות בצורה רנדומלית כדי ליצור מודל יחידה, מחשב את הנקודות שהן inliers

← <u>כמה איטרציות יהיו?</u>

נגדיר:

s = כמות הנקודות במודל (לדוגמא בקו- 2, בהומוגרפיה- 4)

w = הסיכוי לקבל inlier.

p = הסיכוי להצליח באלגוריתם- לקבל קבוצה שכולה inliers.

צריך לחשב: מה הסיכוי לקבל קבוצה שכולה inliers. כמה פעמים צריך לנסות במינימום כדי שמבין כל אוסף המודלים שבדקנו (הקווים/ ההומוגרפיות וכו׳) נקבל לפחות קבוצה אחת, מודל אחד שכולו נקודות inliers.

Denote the inliers probability by w:

What is N ?

$$(1-w^s)^N \le 1-p$$

$$N \ge \frac{\log(1-p)}{\log(1-w^s)}$$

הסיכוי לקבל דגימה שכולה נקודות inliers : w^s כיוון שיש s נקודות במודל ולכל אחת סיכוי של w להיות inlier.

הסיכוי שלא תיבחר קבוצה כזו של נקודות בכל ה N איטרציות הוא $(1-w^s)^N$

יתרונות : קל לחישוב, מתאים להרבה מודלים

חסרונות : אם יש הרבה נקודות שהן outliers האלגוריתם יצטרך הרבה איטרציות. אבל כרגע אין אלגוריתם אחר בתעשייה.

9.3 <u>image stitching:</u>

כעת לאחר שיש בידינו מספיק כלים, ניתן לתכנן אלגוריתם לחיבור תמונות (פנורמה).

מה הבעיה בלחבר תמונות? הבעיה שהתמונות הן אזורים שונים בעולם האמתי. צריך לדעת איזה אזורים חופפים בין התמונות במציאות- כדי לדעת איך לחבר אותן. בנוסף, צריך לדעת עבור כל פיקסל חופף איזה פיקסל הוא מייצג בתמונה השנייה.

סדר פעולות בסיסי בהדבקת תמונות :

1. לקיחת סדרת תמונות מאותה פוזיציה (אולי מזיזים את המצלמה על ציר, אולי מזיזים את המצלמה על אותו קו אופקי)
2. מחשבים טרנספורמציה דו ממדית בין התמונה השנייה לראשונה - הומוגרפיה, למעשה כך יודעים את היחסים של כל שני פיקסלים שבשתי התמונות
3. עוטפים (warp) את התמונה השנייה כדי שתחפוף עם התמונה הראשונה. (להפוך את שתי התמונות לאותו מרחב קואורדינטות)
4. לסיום- כדי שהחיתוך בין שתי התמונות לא יהיה גס- עושים ערבול (blending)
5. חוזרים כך עבור כל סדרת התמונות

10. Extrinsic-Intrinsic Geometry

מוטיבציה: כדי למצוא את ממד העומק באמצעות stereo אנו צריכים לדעת את מיקום שני המרכזים האופטיים של המצלמות שלקחו את תמונות שמאל וימין.

כדי להיות מסוגלים לעשות גיאומטריה על המרחב של העולם אנחנו חייבים לדעת כיצד לייחס את מערכת הקואורדינטות של המצלמה למערכת הקואורדינטות של העולם.

שלב ראשון: ייחוס הקואורדינטות של העולם לקואורדינטות של המצלמה – למצוא את מיקום המצלמה בפועל בעולם, Extrinsic parameters.

שלב שני: ייחוס הקואורדינטות התלת ממדיות של המצלמה לקואורדינטות דו ממדיות של התמונה – על ידי הטלה Intrinsic parameters.

10.1 Extrinsic parameters:

כשלב ראשון אנו מחפשים את המטריצה שתבצע העתקה של נקודה ממערכת הקואורדינטות של העולם לקואורדינטות של המצלמה. היא תיקרא המטריצה החיצונית.

- ✓ כל טרנספורמציה של עצם ממרחב תלת ממדי אחד למרחב תלת ממדי אחר בעצם מורכבת מטרנספורמציה של העתקה (translation) + רוטציה (rotation) = מה שנקרא rigid.

- ✓ כמה דרגות חופש יש למטריצה החיצונית?
  מספר דרגות החופש של מטריצת rigid בתלת ממד הוא 6. מדוע?
  כדי להעתיק אובייקט למרחב תלת ממדי אחר יש מספר שלבים:
  1. נקבע את מיקומה של נקודה אחת מהאובייקט במרחב (נגיד, פינה) – 3 דרגות של חופש (x,y,z)
  2. נקבע את מיקומה של נקודה שנייה כדי "לקבע" את האובייקט ביחס לשני צירים- 2 דרגות של חופש (כיוון שהנקודה הזו חייבת להיות על האובייקט במרחק קבוע מראש מהנקודה הראשונה שבחרנו, ירדה דרגה אחת)
  3. נקבע את מיקומה של נקודה שלישית כדי "לקבע" את האובייקט ביחס לציר השלישי – דרגת חופש 1.

- ✓ הרוטציה למעשה יכולה להיות סביב כל אחד מהצירים. לכן זה משנה הסדר של הרוטציות (קודם סביב ציר ה x ואז סביב ציר ה y החדש, או קודם סביב ציר ה y ואז סביב ציר ה x החדש)

- ✓ כדי לאחד את מטריצת הרוטציה וההזזה למטריצה אחת, נשתמש במערכת קואורדינטות הומוגנית – נוסיף ממד שיהיה וקטור היחידה (בגלל ההזזה):

World to camera

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

בפועל נשתמש במטריצה 3*4 אלא אם כן נרצה את ההופכית שלה:

$$[R \mid t] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]$$

10.2 **Intrinsic parameters:**

בשלב השני אנו מחפשים את המטריצה שתבצע העתקה של נקודה ממרחב הקואורדינטות של המצלמה בעולם (התלת-ממדי) למרחב הקואורדינטות של התמונה (משטח דו ממדי) על ידי הטלה.

למעשה כבר ראינו את הרעיון הזה של הטלה ממרחב תלת ממדי למרחב התמונה ב*גיאומטריה פרויקטיבית*. הסיבה שזה לא בדיוק עובד ככה בצורה אידיאלית וצריך לבנות מטריצה 'פנימית' היא מכמה סיבות:

1. עבור f focal length משתנה בהתאם לכמה פיקסלים החיישן במצלמה דוגם עבור כל מילימטר בעולם האמתי. ערך זה יכול להיות שונה עבור ציר הx וציר הy ולכן נקבל פה שתי דרגות של חופש.
2. בגיאומטריה פרויקטיבית הנחנו שמרכז התמונה (מרכז מערכת הצירים) תמיד יהיה במרכז בהתאם למיקום של המרכז האופטי, אבל מה אם הוא לא (התמונה נחתכה לדוגמא, או שהחיישן לא באמת במרכז?) אז צריך לתקן את היחסים האלו באמצעות זווית כלשהי. עוד דרגת חופש אחת.
3. לא ניתן להניח שתמיד ציר הx וציר הy תמיד יהיו מאונכים זה לזה. יכול להיות שהחיישן לא דוגם בצורה כזו את העולם ולכן יש צורך לבטא את ההבדל בין ציר הx האידיאלי לציר הx' שנדגם על ידי המצלמה (שלא תמיד יהיה מאונך), וכן בהתאם לציר הy. עוד שתי דרגות של חופש.

החישובים האלו יוצרים משוואות די מסובכות. נוכל לעבור למערכת קואורדינטות הומוגנית (בגלל שההטלה חילקה כל הקואורדינטה בממד העומק, נוכל להחזיר אותו ולעבור למערכת הומוגנית).

כאן v_0 , u_0 הם בעבור ההזזה באם מרכז התמונה לא בהתאם למרכז האופטי (offset).

β , α הם בעבור היחס בין הצירים (aspect ratio). S ההטיה של הצירים (skew).

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 & 0 \\ 0 & \beta & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

במקרים שנניח שמרכז התמונה בהתאם למרכז האופטי, שהיחס בין יחידות הפיקסלים לאורך ליחידות הפיקסלים לרוחב זהה, ושאין הטיה של הצירים נקבל מטריצה פנימית פשוטה הרבה יותר:

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

שכאן f בעבור ה focal length בלבד. (החיתוך במסגרת האדומה הוא כי הווקטור האחרון הוא וקטור של אפסים ולכן ניתן להתעלם ממנו).

10.3 **מציאת מטריצת הכיול של המצלמה:**

שלב אחרון: שילוב של שתי המטריצות וקבלת 'מטריצת הכיול של המצלמה'

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \overset{5}{\begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}} \overset{6}{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

הפיקסל בתמונה במערכת הומוגנית מוכפל בממד העומק w

מטריצה פנימית 3*3

מטריצה חיצונית 3*4

הנקודה בעולם במערכת הומוגנית

לסיכום, נשלב את שתי המטריצות למטריצה אחת, נקראת M או לפעמים ℿ :

$$\mathbf{x} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{\Pi X}$$

מטריצה זו מכילה 11 דרגות של חופש : 5 בעבור הפנימית, 3 בעבור הזזה החיצונית ועוד 3 בעבור הרוטציה החיצונית.

10.3 **<u>Calibration of the camera</u>**

בהינתן נקודה בעולם נרצה למצוא את הפיקסל בתמונה. אנחנו מחפשים את הפרמטרים של המיפוי הזה.

- שתי משוואות בעבור כל פיקסל שמורכבים מהקואורדינטות של הפיקסל בעולם בתוצאת כפל במטריצת הכיול.

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \sim \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

$$\begin{aligned} u_i &= \frac{m_{00}X_i + m_{01}Y_i + m_{02}Z_i + m_{03}}{m_{20}X_i + m_{21}Y_i + m_{22}Z_i + 1} \\ v_i &= \frac{m_{10}X_i + m_{11}Y_i + m_{12}Z_i + m_{13}}{m_{20}X_i + m_{21}Y_i + m_{22}Z_i + 1} \end{aligned}$$

$$\begin{aligned} u_i(m_{20}X_i + m_{21}Y_i + m_{22}Z_i + 1) = m_{00}X_i + m_{01}Y_i + m_{02}Z_i + m_{03} \\ v_i(m_{20}X_i + m_{21}Y_i + m_{22}Z_i + 1) = m_{10}X_i + m_{11}Y_i + m_{12}Z_i + m_{13} \end{aligned}$$

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -u_iX_i & -u_iY_i & -u_iZ_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -v_iX_i & -v_iY_i & -v_iZ_i \end{bmatrix} \begin{bmatrix} m_{00} \\ m_{01} \\ m_{02} \\ m_{03} \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{20} \\ m_{21} \\ m_{22} \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix}$$

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{00} \\ m_{01} \\ m_{02} \\ m_{03} \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{20} \\ m_{21} \\ m_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

m_{21}

נוצרה מערכת קואורדינטות הומוגנית- כאשר וקטור הפיתון שווה ל – 0. זו בעיה לפתור את מערכת המשוואות הזו כי הפתרון הטריוויאלי הוא שכל הנעלמים (פרמטרים של מטריצת הכיול) יהיו שווים ל – 0.

על מנת לפתור את הבעיה נוסיף מגבלה על הפתרון : הגודל של הווקטור m חייב להיות שווה ל1.

נקרא למטריצה שמכילה נקודות מהעולם האמתי וערכן בתמונה A , מטריצה זו מכילה 12 עמודות בהתאם ל12 הנעלמים של המטריצה, ו2n שורות- שתי משוואות עבור כל נקודה, כאשר יש n נקודות.

למעשה הפכנו את פתירת מערכת המשוואות לבעיה של מציאת מינימום m למערכת הבאה בהינתן המגבלה לעיל :

$$||\mathrm{A}m||$$

נשתמש בשיטת DLT – direct linear transformation (or calibration)

נהפוך את המטריצה A לריבועית על ידי הכפלתה במשוחלפת : $\mathrm{A}^{\mathrm{T}}A$ ונמצא לה את הווקטורים העצמיים.

נבחר את הווקטור העצמי עם הערך העצמי הכי נמוך וזה יהיה וקטור הנעלמים m.

הפתרון מבוסס על שיטת SVD שמייצרת פירוק של המטריצה A (ופורט כבר בפרק שמסביר על מציאת הומוגרפיה) :

יתרונות של השיטה :

1. קלה למימוש
2. מיוחסת כהקטנת ערך הטעות האלגברית

חסרונות של השיטה :

1. לא מחזירה באופן ישיר את הפרמטרים של המצלמה
2. לא מתייחסת לרעש בתמונה
3. קשה להוסיף לשיטה מידע נתון מראש (כמו ידע על ה *focal length* שמגיע עם המצלמה)
4. לא באמת מקטין את הפונקציה הנכונה – הטעות בין איפה שהמטריצה M מסווגת את הנקודה בתמונה לאיפה שהנקודה באמת מוקרנת עליה.

11. קצת על למידת מכונה

כדאי לדעת-

- ✓ ImageNet- מאגר מידע אינטרנטי של תמונות, משתמשים בו כדי לבדוק אלגוריתמי זיהוי תמונות.
- ✓ למידת המכונה הצליחה לשפר בעזרת deep learning את אחוזי השגיאות ב 15%.

11.1 אלגוריתמי סיווג - Classification:

בשלב הזה יש שני מונחים: labels: תגיות, מה שאנחנו קובעים מראש (לדוגמא – כלב וחתול), output:

נותנים למחשב להגדיר מה רואים בתמונה, רואים תמונה ברורה של מה שצריך עם אובייקט יחיד. שלב למידה.

זיהוי - Detection:

הקלט היא תמונה עם הרבה אובייקטים, צריך שהאלגוריתם ימצא את האובייקט ויזהה אותו בעצמו.

איך עוברים מאפיון לזיהוי? הדרך לעשות זיהוי היא סוג של brute force- עוברים על "חלקים" מהתמונה (ולא פיקסל-פיקסל, באמצעות משהו שלא נלמד. יש איזשהו מודול שיודע בצורה מהירה לקבוע אם שווה להסתכל על הפיקסל הזה) ובודקים אם זה מה שמעניין אותנו.

Logistic classifier – אלגוריתם לסיווג:

נניח שיש לנו קלט X, ואנחנו רוצים למפות בין הX הזה לאיזשהו label.

נניח שהמיפוי לינארי- נייצג את X בתור אוסף של מספרים וגם את הlabel-ים בתור מספרים. המטרה תהיה למצוא מטריצה W ווקטור b (bias) כך שעבור קלט X נקבל כתשובה לאיזה מחלקת אפיון הקלט שייך.

טרמינולוגיה- המטריצה W זו מטריצה של משקולות. אותה אנחנו לא יודעים ואותה אנחנו רוצים למצוא. כלומר עבור כל קלט X **נתון** לנו **לאיזה מחלקה הוא שייך** והמטרה היא **למצוא את מטריצת המשקלים**.

זו למעשה למידה. למצוא פונקציה שבהינתן תמונה- תיתן את המחלקה שהיא שייכת אליה, כאשר פונקציה זו תלויה ב (W, b) שאותם מוצאים על בסיס כל הדוגמאות בתהליך **האימון**.

נשים לב שהמכפלה של המטריצה W בקלט X תיתן וקטור בסוף בגודל 3 על 1. וקטור זה נקרא **logit**. בעיקרון היינו לוקחים את המספר הגבוה ביותר והמיקום שלו (תא במערך) זה המיקום שמתאים למחלקה הנכונה. אבל היינו רוצים להפוך את הווקטור הזה לווקטור הסתברויות. היחס בין התוצאות הופך להסתברויות באמצעות פונקציית **soft-max**.

אז באמצעות פונקציה זו ממירים להסתברויות ולוקחים את ההסתברות הגבוה ביותר.

בתהליך האימון ווקטור התוצאה יהיה ווקטור שבו יהיה ערך 1 במיקום של המחלקה הנכונה וערך 0 בכל השאר.

One-Hot-Ncoding:

הפיכה לקטור שבו ההסתברות הגבוהה ביותר מקבלת ערך 1 בכניסה המתאימה בווקטור וכל השאר מקבלים ערך 0.

הlabels צריכים להיות בצורה זו.

Cross-Entropy:

נשאלת השאלה- איך נדע כמה קרוב הווקטור של ההסתברויות שקיבלנו לווקטור שהתכוונתי שיצא? אני רוצה לאמן את הclassifier- כלומר לשפר את המטריצה W כדי שהווקטור הסתברויות שיצא יהיה כמה שיותר מתאים לווקטור של התוצאה האמתית (שאני יודעת מראש באימון). בעזרת הפונקציה Cross-Entropy.

Loss = average cross- entropy:

פונקציה שמקשרת בין המטריצה W והווקטור b לכל מיני דוגמאות של קלט X ולכל מיני labels (מתאימים ולא מתאימים). נותנת ערך יחיד של ממוצע שגיאות.

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function

בעזרת האלגוריתם הזה מוצאים את ה W וה b הכי טובים שאפשר, ביחס לתמונות קלט והlabels שונים. מה שנקרא **training set**.

יכולה להיווצר בעיה בזמן חישוב ה gradient descend (אלגוריתם). כיוון שהמספרים במחשב יכולים להמשך הרבה אחרי הנקודה... משהו שם יוצא לא זהה בחישוב הממוצע.

צריך איכשהו להעביר את הממוצע ל 0 ושהשונות בין הערכים תהיה 1.

איך עושים את זה? מניחים שמקבלים תמונה RGB ולכן מספר הערכים שיכולים לצאת הם 255 והאמצע- 128. פשוט לכל ערוץ מחסירים 128 ומחלקים ב128. – **זה הנרמול בשלב שלפני תחילת תהליך הלמידה.**

איך מאתחלים את W?

הרבה פעמים בוחרים מטריצה אחידה (uniform) או גאוסיינית.

כלים לבדיקה של ה classifier:

Test set - מלבד הכלים של ה training set נשמור בצד תמונות שלא ניגע בהם בשלב האימון כדי לראות אם המכונה למדה.

Validation Dataset - סט של תמונות שנבחן באמצעותו את המכונה כאשר נגיע לשלב שבו אנחנו חושבים שהמכונה למדה טוב.

<u>Learning rate tuning:</u>

יכול להיווצר מצב בו תהליך הלמידה מאוד מאוד מהיר- אחוזי השגיאה ירדו בצניחה מהירה (ה loss function). אבל באיזשהו שלב הלמידה תיעצר. מבחינה מתמטית זה קורה כי בחרנו את האלפא בפונקציה של הגרדיאנט יותר מדי גבוה. (לא הסביר יותר מדי).

- חיסרון (עיקרי) של מודלים לינאריים: אי אפשר לשרשר classifier-ים אחד לשני, כי ברגע שנשרשר מטריצת W אחת למטריצה W 2 ואפילו נשרשר 100 כאלה , בסוף נקבל מטריצת W אחת ששקולה ל classifier אחד. (מבחינה לינארית, תכפיל מטריצה 4 על 4 , במטריצה 4 על 4- תקבל מטריצה 4 על 4...)

ננסה להפוך את הפונקציה ללא לינארית (?)

דוגמא: RELU – כל מה שהוא שלילי הופך ל-0 וכל מה שחיובי נשאר

ניקח classifier ונוסיף ל RELU, מסתבר שהגדלנו את כוח החישוב בצורה משמעותית (הוכחה לא במסגרת הקורס).

נגיעה בנושא מהשבוע הבא: רשת זה לקחת מטריצה W ווקטור b ולעשות עליהם RELU, לשרשר למטריצה W אחרת ווקטור b אחר ולעשות עליהם RELU וכו'.

11.2 <u>הקדמה לרשתות נוירונים:</u>

איך רשת נוירונים עובדת?

מכניסים תמונת קלט, ולאחר שהיא עוברת בכמה שכבות של הרשת מקבלים כפלט וקטור, כאשר כל תא מייצג את התגית, והספרה שבתא מייצגת את הסיכוי שהתמונה היא התגית הזו.

במידה וקיבלנו בשני תאים ספרות גבוהות צריך להבין איך "להעניש" את האלגוריתם כדי שילמד מהטעויות.

יש וקטור של התשובות האמתיות, והווקטור הפרש בין הפלט של האלגוריתם לווקטור האמתי הוא ה loss vector ומייצג כמה האלגוריתם צדק. לפי הטעות עושים גרדיאנט (הגרדיאנט מוציא את הכיוון של הטעות המינימלית), חוזרים דרך אותה קשת ברשת ומתקנים את השכבת אקטיבציה האחרונה.

- **Backward propagation**

X = [(1,1),(1,1)]

Y = X + 2

Z = Y^2*3 (z now is matrix of 27, 2 rows 2 cols)

Out = z.mean()   (out = 27)

Out.backward()

מה שקורה כאן זה שהגרדיאנט של out כלומר dout/dx מתווסף לערכים של השכבה הקודמת, כאשר x זה הערכים של הוקטור של השכבה הקודמת, ויחשב :

x.grad += dout /dx

- np.argmax – פונקציה שמקבלת וקטור ומוציאה את הערך המקסימלי. משתמשים בשלב שמחלצים מהווקטור פלט את התא הכי גבוה.
- רשת שהיא fully-connected, כל יחידת עיבוד/צומת מחוברת לכל שאר היחידות עיבוד/צמתים בשכבה שלפניה. גרף מלא. יש פונקציה fc3 שעושה את זה כשרוצים לעבור קדימה ברשת- פונקציית forward שאנחנו צריכים ליצור
- Batch-learning : במקום ללמוד כל פעם מטעות אחת (כל פעם מנסים ללמוד לפי הגרדיאנט – מחשבים אותו ומתקדמים בכיוונו לפי איזשהו מספר שקבענו מראש. אם נעשה טווח קטן מדי נתקדם מאוד לאט, אם נעשה טווח גדול מדי אנחנו עלולים כל הזמן לפספס) מכניסים כל פעם כמה דוגמאות ועושים ממוצע של הטעות שלהם ולפיהם בוחרים את הטווח להתקדם. **Learning rate** . כלומר לא קובעים מראש את הטווח הזה אלא כל הזמן משנים אותו לפי החישובים. stochastic gradient descent . (אלגוריתם סטטיסטי לאופטימזיציה של הטווח הזה). ב pytorch נקרא : torch.optim.SGD

רשת נוירונים מראש לא תוכננה לתמונות, ולכן כשמגיעים לתמונות צריך לדעת שקונוולוציה יכולה לעזור. קונוולוציה זה בעצם ממוצע משוקלל, וככה אפשר לקשר כל פיקסל לסביבה שלו ברשת.

לדוגמא : נעשה שתי שכבות ראשונות של קונוולוציה ולאחר מכן שתי שכבות של fully-connected.