אלגוריתמים בראייה ממוחשבת

046746

רטוב 1

דניאל טייטלמן – 207734088 Daniel.tei@campus.technion.ac.il

יאיר נחום – 034462796 nahum.yair@campus.technion.ac.il

פרק 1:

1. טענו את התמונה של model\_chickenbroth.jpg והצגנו אותה.



1. על מנת ליצור פירמידה גאוסית השתמשנו בפונקציה הנתונה, כאשר ביצענו את שלבי ה – PREPROCESSING הבאים לתמונה: המרה מ – RGB ל GRAY ולאחר מכן מעבר מתצוגה ב UINT8 להצגה ע"י FLOAT בין 0 ל 1. תוצאת הפירמידה הגאוסית שהתקבלה היא בעלת הממדים הבאים:



*והתמונה המתקבלת מוצגת היא באמצעות פריסה ע"י HSTACK של רכיבי הפירמידה.*

**

1. *מימשנו את פירמידת ה – DoG, ביצענו זאת באמצעות ההפרשים בין הרמות השונות של הפירמידה הגאוסית. כאשר לבסוף קיבלנו פירמידה בעלת הממדים הבאים:*

**

*התמונה מוצגת באמצעות פריסה ע"י HSTACK של רכיבי הפירמידה.*

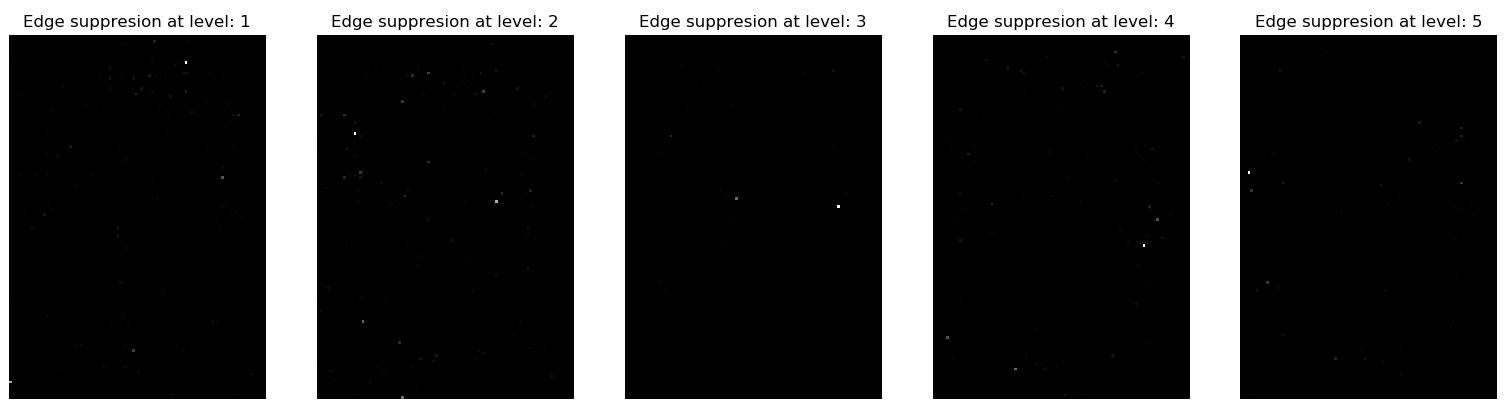
**

1. *כעת מימשנו את EDGE SUPPRESSION ביצענו זאת בשלבים הבאים:*
2. *חישוב של ההסיאן באמצעות ביצוע 3 פילטרי SOBEL בעלי המאפיינים הבאים: כך שפילטר SOBEL מבוצע פעמיים על ציר ואפס פעמיים על ציר , כך שפילטר SOBEL מבוצע פעמיים על ציר ואפס פעמיים על ציר , כך שפילטר SOBEL מבוצע פעם אחת על ציר ופעם אחת על ציר (הפילטר סימטרי). גודל פילטר סובל הוא 3 בכל המקרים אצלנו.*
3. *לאחר מכן חישבנו את ה - ו בעבור כל נקודה בצורה הבאה:*

*כאשר היא מכפלה איבר איבר, וזאת מפני שאנו משתמשים בכתיב מטריצי לחישוב.*

1. *חישוב יחס העקמומיות איבר איבר בצורה הבאה:*

*אם נציג תמונה ללא סף בינארי נקבל.*

**

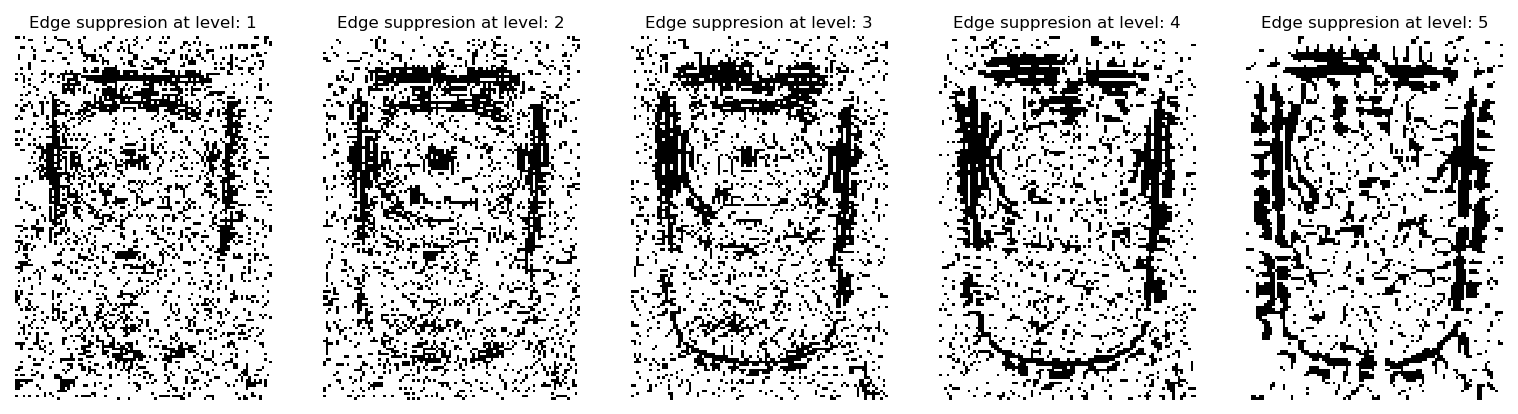
1. *ביצוע סף בינארי למטריצת ערכי העקמומיות עם הגודל .*

*חישוב זה בוצע בעבור כל רמה בפירמידה הגאוסית. (חישוב זה מבוצע ב 1.5 אך מוסף כאן, כדי לקבל את התמונות והוא מבוצע כסף קשיח ולא כ"גמיש")*

*הממדים של מטריצת ה – Principal Curvature הם:*

**

*התוצאה הסופית בעבור 4 רמות הפירמידה לאחר EDGE SUPPRESSION.*

**

1. *בסעיף זה מצאנו את המינימות והמקסימות המקומיות בצורה הבאה.*

*ראשית כל יצרנו MASK שמוגדר באמצעות Thresholding על פי הערכים של thcontrast בעבור הפרש הפירמידות הגאוסיות (בערך מוחלט) כאשר תנאי זה מתקיים יחד עם התנאי כי העקמומיות הראשית קטנה מ – thr.*

*לאחר מכן עברנו על כל הנקודות ב MASK שמקיימות את התנאי ובדקנו בשכנות של 8 (באותה הרמה) ובעומק אחד מעל ומתחת בשכנות 9 האם מתקבל מינימום או מקסימום מקומי. אם כן אז הוספנו על פי הסדר של האינדקסים שהתבקשנו למערך locsDoG.*

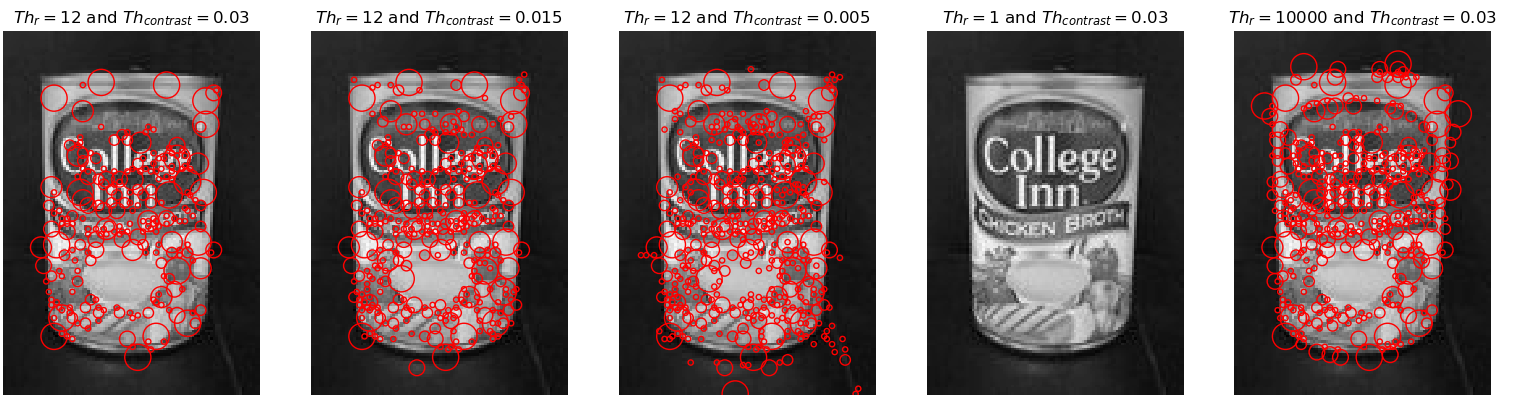
**

1. *כעת חיברנו את כלל החלקים השונים של האלגוריתם כעת נבחן את האלגוריתם על כמה תמונות שונות, ובמצבים שונים.*

*ראשית נתחיל מן התמונה הראשונית שקיבלנו.*



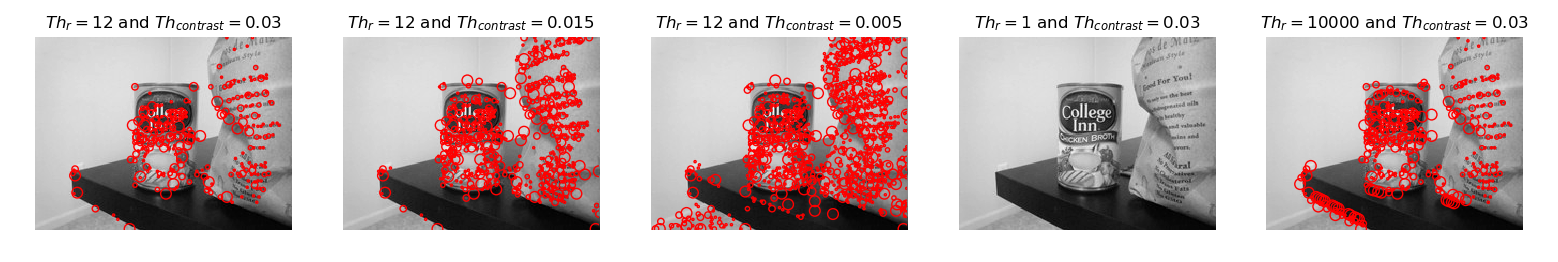
*נפעיל את האלגוריתם ראשית עם פרמטרים שונים. בכלל התמונות העיגולים מסמנים את מיקום המאפיינים, וגודלם קשור לרמה בה הם מופיעים.*

**

*כפי שניתן לראות הערך של הוא בחירה יחסית טובה ויחסית רובסטי, לעומת זאת הבחירה של היא בעלת השפעה משמעותית ולכן אנו עלולים להגיע למקרה הקיצון כמו התמונה האמצעית או למקרה השני בו יש מעט מאפיינים כמו בתמונה השמאלית ביותר, על כן יש חשיבות לבחירת מקדם זה.*

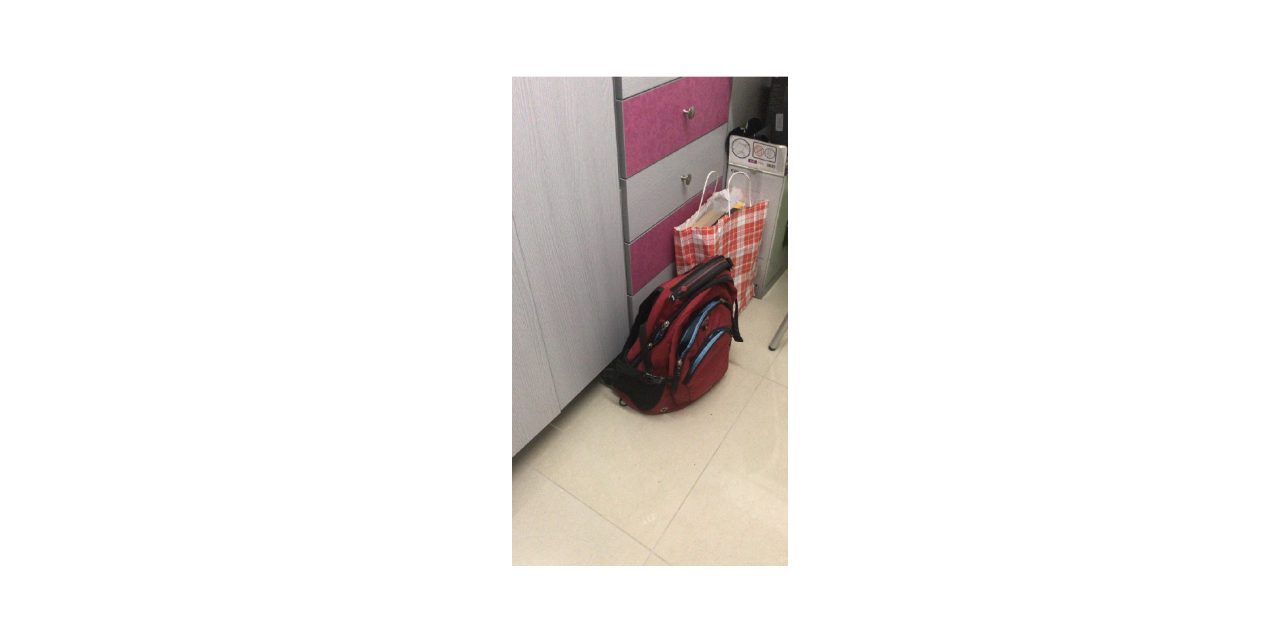
*בעבור תמונה נוספת שבדקנו:*



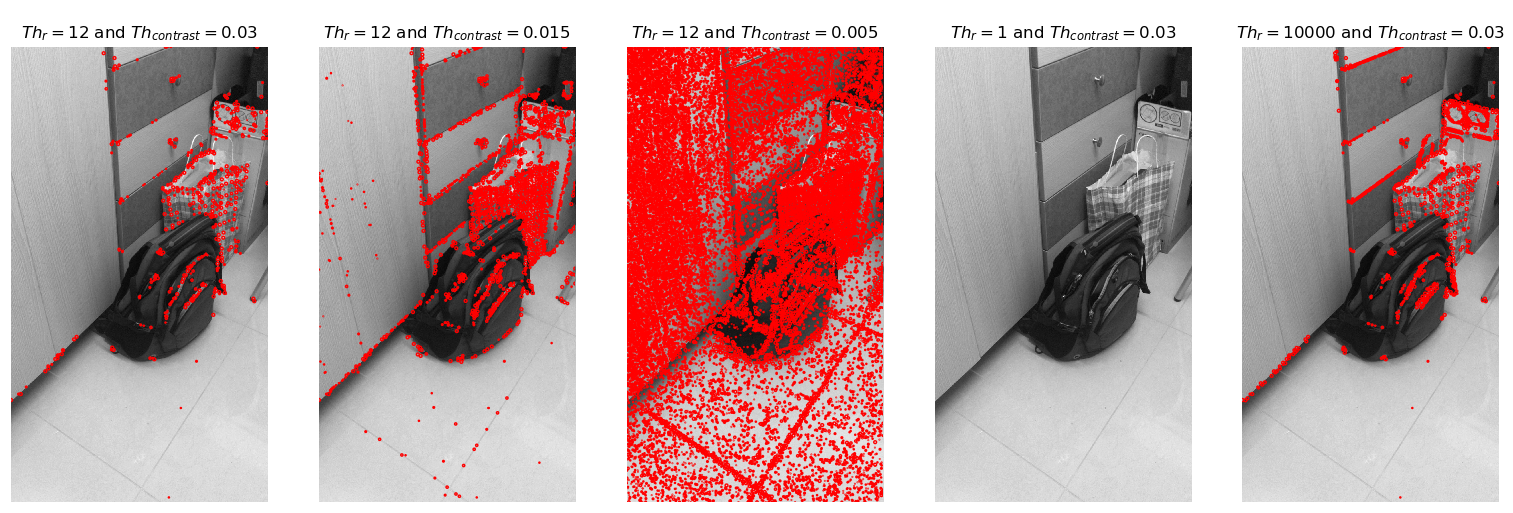
**

*שוב ניתן לראות תוצאות דומות להסבר שתואר לפני כן.*

*כעת בדקנו בעבור תמונה טבעית שצילמנו.*



*ולאחר הרצת האלגוריתם.*

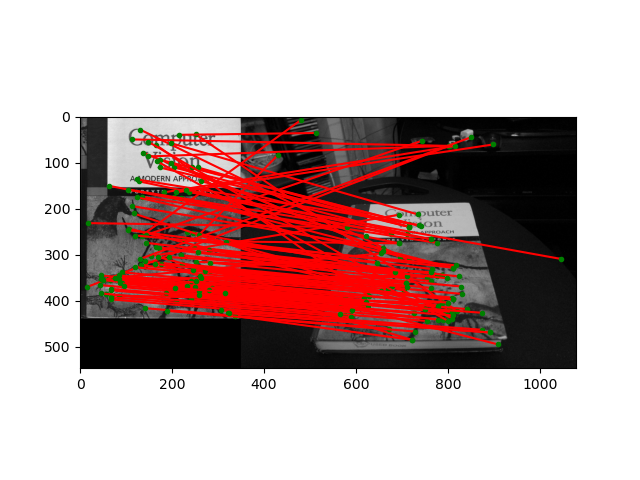
**

*כעת התמונה האמצעית היא הגרועה ביותר, לעומת השמאלית ביותר אשר היא הטובה ביותר. על כן יש לבדוק כי אכן פרמטרי האלגוריתמים מותאמים היטב לתמונות טבעיות או באופן מדויק יותר בעבור סוג התמונות אשר האלגוריתם אמור להתמודד איתם בעולם האמיתי.*

Part B:

1. Brief Descriptor (code is at code/part\_B/ the results are at data/my\_data/)
   1. See implementation in my\_BRIEF.py code. The main.py module uses it and creates the testPattern.mat file at every run of main.py   
      From the article methods we’ve implemented the uniform distribution randomization on patch locations.
   2. See in my\_BRIEF.py. We’ve implemented comupteBrief function by first filtering out any point that doesn’t have the required support (9x9) within the image pixels’ boundaries.  
      Then, we’ve built the ‘m’ locations’ descriptors:  
      We’ve converted the 9x9 support per location to a flat vector of 81x1. Then, we’ve taken the 2 series of pairs, Xn and Yn, and get the vector according the Rho function defined (to get the binary vector as a descriptor per feature location). It could have been bits (to save memory and get better performance on distance calculations) but we’re not required to save in bits.
   3. Here, we simply load the testPattern.mat in which we save the parameters for briefLite function (not only compareX and compare fields in hash).   
      We then get the DoG features and the Gaussian pyramid and use them together with other parameters to calculate the filtered (inner pixels) features’ locations and their descriptors.
   4. Descriptors’ matching  
      We’ve implemented the testMatch.py script (also run from part B main.py) and plotted the results:
      1. Chickenbroth image 01 vs image 04 (ratio=0.6):  
         A picture containing diagram

         Description automatically generated  
         In the above example we can see that the brief is robust and invariant against illumination changes. This is so, since we calculate the R value in DoG filtering as a ratio between trace^2 and det of the gradient’s covariance matrix. Thus, as we saw in lecture, it will be invariant to scale and illumination (better in this sense from Harris Detector).
      2. incline\_L vs incline\_R (ratio=0.4):  
         Graphical user interface, application

         Description automatically generated  
         We can see that BRIEF works well also when there is a different perspective on the scene. It matches the same high building’s corners. But due to the amount of details that are missing in one image vs the other, there are many false positive.
      3. pf\_scan\_scaled vs pf\_desk (ratio=0.6):  
         under perspective transformation, it seems that there are feature points which match (for example the ‘p’ letter in ‘Computer’. Reducing the r parameter might show better robust results.
      4. pf\_scan\_scaled vs pf\_floor (ratio=0.6):A picture containing diagram

         Description automatically generated  
         pf\_scan\_scaled vs pf\_floor (ratio=0.4):A picture containing diagram

         Description automatically generated  
         Since the CV book image is not rotated on the floor, we can see that the BRIEF operates very well. It also finds some false matches against other books though.
      5. pf\_scan\_scaled vs pf\_floor\_rot (ratio=0.6):  
         A picture containing diagram

         Description automatically generated  
         When rotation is applied, we see the performance degrade and BRIEF can’t match the correct book over the others. It finds matches similarly against all books. Even the most evident corners like the white corners cannot match between the 2 images.
      6. pf\_scan\_scaled vs pf\_pile (ratio=0.6):A picture containing chart

         Description automatically generated  
         Again, poor performance when rotating. Now the shape of the book against the background is also not clear and BRIEF doesn’t match OK.
      7. pf\_scan\_scaled vs pf\_stand (ratio=0.6): A picture containing chart

         Description automatically generated  
         pf\_scan\_scaled vs pf\_stand (ratio=0.4): A picture containing diagram

         Description automatically generated

The standing book is not rotated against the original and the change in perspective is not significant. We mainly have translation. Brief handles it very well.

When we compare an image to itself we found a perfect match between the features:  
Graphical user interface

Description automatically generated

* + 1. As we saw in the comparison of cheickenbroth images 01 and 04 the BRIEF performed well even under changes of illumination. This is so, since we calculate the R value in DoG filtering as a ratio between trace^2 and det of the gradient’s covariance matrix. Thus, as we saw in lecture, it will be invariant to scale and illumination (better in this sense from Harris Detector). Also, multiplying each pixel in the image (and also adding a bias to the pixel BTW) won’t change the relation between I(x) < I(y). it will preserve the same sign as we wish under illumination changes.
    2. We can relate to the scale in which the feature was detected and get a patchWidth that is scaled to that scale (not only 9x9 for all scales). This is what’s done in SIFT and other similar approaches.
    3. Took the model\_chickenbroth.jpg test image and matched it to itself while rotating the second image in increments of 10 degrees. Got the following bar plot:  
       Chart, histogram

       Description automatically generated  
       It shows that we have a poor matching when the rotation is above +/-10 degrees. We can explain it by not preserving the orientation as in SIFT (in SIFT we also rotate in the needed scale to horizontal/vertical direction all the patches).
  1. In ORB we’ve run the same rotation test and got the following results:  
     Chart, bar chart, histogram

     Description automatically generated  
     It shows that the ORB keeps matching correctly even with any rotation angle.  
     We can explain it by ORB borrowing from SIFT the orientation saving.  
     The article about OBR mentions the “Orientation by Intensity Centroid”. Which is a method to keep the feature orientation.