Άσκηση 2η – Μηχανική Όραση

Σοφία Καφρίτσα Γεωργαντά, 2016030136

- *Σκοπός*

Σκοπός της 2ης άσκησης ήταν να υλοποιήσουμε ένα μηχανισμό για Laplacian Blob Detection. Όταν αναφερόμαστε σε blob detection μηχανισμούς, εννοούμε μηχανισμούς που ανιχνεύουν περιοχές ψηφιακή εικόνας, οι οποίες διαφέρουν σε κάποιες ιδιότητες όπως φωτεινότητα ή χρωματικές αποχρώσεις, συγκριτικά με τις γειτονικές τους περιοχές. Μέσω Scale – Space Blob Detection μπορούμε να βρούμε σε διαφορετικές κλίμακες πού βρίσκονται τα σημεία ενδιαφέροντος και να τα αντιστοιχίσουμε μεταξύ τους, ώστε να μπορούμε να τα εντοπίσουμε ακόμα και υπό διαφορετική φωτεινότητα ή οπτική γωνία.

- *Επεξεργασία*

Στην υλοποίησή μου εργάστηκα ως εξής: αρχικά, ορίζω τις αρχικές τιμές των παραμέτρων στο αρχείο *evaluate*, όπου μετέτρεψα την εικόνα σε gray-scaled και έπειτα σε double. Στη συνέχεια, αρχικοποιώ τις παραμέτρους: scale «σ», k (ο παράγοντας με τον οποίο πολλαπλασιάζεται το scale κάθε φορά) , levels (επίπεδα επαναλήψεων) και threshold. Για να καταλήξω στις τελικές τιμές των παραμέτρων, έκανα κάποιες δοκιμές και αποφάσισα να εφαρμόσω τις παρακάτω τιμές.

1) σ = 1.8
2) k = $\sqrt{2}$
3) levels = 10
4) threshold για εικόνα fishes.jpg = 0.003
5) threshold για εικόνα sunflowers.jpg = 0.015
6) threshold για εικόνα escher.jpg = 0.045
7) threshold για εικόνα dots.jpg = 0.003

Έπειτα, καλώ τη συνάρτηση *detect_blobs*, όπου ορίζω μέγεθος φίλτρου ίσο με *filter_size = 2*ceil(3*sigma)+1*, το οποίο θέλουμε να είναι περιττός αριθμός για να διατηρείται η συμμετρία γύρω από το κεντρικό pixel, και παράγω το *Laplacian of Gaussian filter*, ως εξής: *filter = fspecial('log', filter_size, sigma).* Το φίλτρο αυτό το έχω βάλει σε σχόλια και το ορίζω στους επόμενους υπολογισμούς μέσα σε for loop.
Όσον αφορά την παράμετρο 'log', θέλω το φίλτρο να είναι συμμετρικό ως προς την περιστροφή (rotationally symmetric).
Στη συνέχεια, δεσμεύω χώρο *scale_space = zeros(h,w,levels)* για τους μετέπειτα υπολογισμούς και υλοποιώ τις δύο μεθόδους blob detection.

Μέθοδος 1: Αυξάνοντας το μέγεθος του φίλτρου

```
%---------- Method 1 : Increasing filter size ---------------
if method == 1
    tic
    for i=1:levels

        filter = fspecial('log', filter_size, sigma); %1st level
        filter = filter.* (sigma^2); % Scale Normalize Laplacian

        figure; % check window size
        surf(abs(filter),'EdgeColor','none');

        im_filtered = imfilter(img, filter,'replicate');  %
        im_filtered = im_filtered.^2;
        sigma = k * sigma;     %increasing s for next level
        filter_size = 2*(ceil(3*sigma)) + 1; %new filter size
        scale_space(:,:,i) = im_filtered; %storing

    end
    toc
end
```

Όπως φαίνεται στην εικόνα, ορίζω ένα for loop που τρέχει πάνω στα επίπεδα αρχίζοντας από το πρώτο, και υλοποιώ τον παρακάτω αλγόριθμο:

1. Υπολογίζω φίλτρο Laplacian of Gaussian filter συναρτήσει του *σ.*
2. Πολλαπλασιάζω το φίλτρο με σ^2 – scale-normalized Laplacian.
3. Κάνω plot το φίλτρο με τη συνάρτηση surf, ώστε να εξακριβώσω ότι δεν χάνεται πληροφορία λόγω του φίλτρου.
4. Δημιουργώ τη φιλτραρισμένη εικόνα με τη συνάρτηση *imfilter* και χρησιμοποιώντας παράμετρο *'replicate'.*
5. Πολλαπλασιάζω την εικόνα με τον εαυτό της.
6. Αυξάνω το *σ* για το επόμενο επίπεδο συναρτήσει του *k*.
7. Υπολογίζω το καινούριο μέγεθος φίλτρου με το νέο *σ* συναρτήσει του *k*.
8. Αποθηκεύω την εικόνα κάθε επιπέδου στον τρισδιάστατο πίνακα scale_space.

Για το βήμα 5, βλέπουμε ότι κάθε επίπεδο θα είναι πολλαπλασιασμένο επί *k*, συγκριτικά με το προηγούμενο. Συγκεκριμένα, για οποιαδήποτε τιμή του σ και για κάθε επίπεδο θα έχουμε:

σ = τιμή
$\sigma_1 = k \cdot \sigma = k \cdot$τιμή
$\sigma_2 = k \cdot \sigma_1 = k \cdot k \cdot$τιμή $= k^2 \cdot$τιμή
$\sigma_3 = k \cdot \sigma_2 = k \cdot k \cdot k \cdot$τιμή $= k^3 \cdot$τιμή
…

$\sigma_{levels} = k \cdot \sigma_{levels-1} = k^{levels} \cdot$τιμή

Μέθοδος 2: Μειώνοντας το μέγεθος της εικόνας

```
%---------- Method 2 : keeping filter size constant ---------
%---------------and downsampling the image ---------------

if method == 2
    tic
    filter_constant = fspecial('log', filter_size, sigma).*(sigma^2);
    for i=1:levels

         img_down = imresize(img,(1/(k^(i-1))),'bicubic'); %Decreasing image size
         im_filtered = imfilter(img_down, filter_constant,'replicate'); %Applying the Laplacian of Gaussian filter
         im_filtered = imresize(im_filtered, [h w],'bicubic'); %resize to original size
         im_filtered = im_filtered.^2; %square of log
         scale_space(:,:,i) = im_filtered; %storing

    end
    toc
end
```

Σε αυτή τη μέθοδο το φίλτρο είναι σταθερό και υποδειγματοληπτούμε την εικόνα. Ο αλγόριθμος είναι ο εξής:

1. Ορίζω σταθερή τιμή για το φίλτρο εκτός του for-loop.
2. Υποδειγματοληπτώ την εικόνα (εκτός από το πρώτο επίπεδο) με τη συνάρτηση *imresize* κατά παράγοντα *k* σε κάθε επίπεδο με παράμετρο 'bicubic'.
3. Εφαρμόζω το φίλτρο με τη συνάρτηση imfilter.
4. Επαναφέρω την εικόνα στο αρχικό μέγεθος γιατί στην επόμενη επανάληψη το i θα είναι αυξημένο κατά 1, επομένως το μέγεθος κάθε εικόνας σε κάθε επανάληψη υπολογίζεται κατευθείαν από την υποδειγματοληψία στο βήμα 2.
5. Παίρνω το τετράγωνο της εικόνας.
6. Αποθηκεύω το αποτέλεσμα στον τρισδιάστατο πίνακα scale_space.

Για interpolation method επέλεξα να εφαρμόσω bicubic, καθώς γνωρίζουμε εξάγει τις ακμές πιο αποτελεσματικά συγκριτικά με τις άλλες μεθόδους.

Στη συνέχεια, για την εύρεση των σημείων ενδιαφέροντος εργάστηκα με τα ακρότατα. Εφαρμόζω φίλτρο 3x3 και συγκρίνουμε τα επίπεδα ανά δύο. Έτσι, έχουμε 8 γείτονες στο ίδιο επίπεδο και 9+9=18 στα άλλα δύο. Ένα σημείο είναι σημείο ενδιαφέροντος αν είναι μικρότερο ή μεγαλύτερο από όλη αυτή τη γειτονιά.

Για τα σημεία ενδιαφέροντος του ίδιου επιπέδου, εφάρμοσα τις συναρτήσεις ordfilt2, nlfilter και colfilt και κατέληξα στην ordfilt2 ως πιο γρήγορη. Δέσμευσα μέρος για την έξοδο για όλα τα επίπεδα έτρεξα τη συνάρτηση ως εξής, με suppression_size = 3.

```
for i=1:levels
    max_scale(:,:,i) = (ordfilt2(scale_space(:,:,i), suppression_size^2, ones(suppression_size)));
    %max_scale(:,:,i) = (nlfilter(scale_space(:,:,i), [3 3], fun));
    %max_scale(:,:,i) = (colfilt(scale_space(:,:,i), [3 3], fun));
end
```

Για τα σημεία ενδιαφέροντος μεταξύ των επιπέδων, επέλεξα να συγκρίνω τα pixel τα οποία βρίσκονται ένα επίπεδο πάνω και ένα επίπεδο κάτω από αυτό που βρισκόμαστε όταν διατρέχουμε το loop.

```
for i=1:levels
    max_scale(:,:,i) = max(max_scale(:,:,max(i-1,1):min(i+1,levels)),[],3);
end
```

Τέλος, δέσμευσα χώρο για τις συντεταγμένες x,y και την ακτίνα των blobs, και σε ένα for-loop για όλα τα επίπεδα, εφαρμόζω τα εξής βήματα:

1. Μηδενίζω όλες τις θέσεις οι οποίες δεν είναι τοπικά ακρότατα.
2. Ορίζω ένα threshold ώστε να επιτρέπω να εμφανίζονται μόνο τα blobs που έχουν μεγαλύτερο squared Laplacian από αυτή την τιμή και αποθηκεύω τα ακρότατα που απομένουν σε δύο μεταβλητές *[row,col]*. Χρησιμοποιώ τη συνάρτηση *find.*
3. Αποθηκεύω τις τιμές των *[row,col]* στα x,y.
4. Υπολογίζω την ακτίνα για κάθε level χρησιμοποιώντας το αρχικό sigma_init.

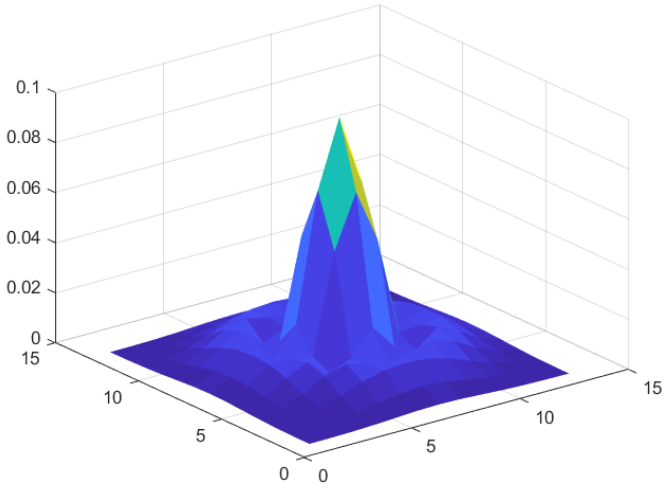
Παρακάτω φαίνεται το κομμάτι του κώδικα.

```
maxima = zeros(h,w,levels);
radius =[];
x =[]; %x axis blob
y =[]; %y axis blob

for i=1:levels

    % Zero Out All Positions that are not the Local Maxima
    maxima(:,:,i) = max_scale(:,:,i).*(max_scale(:,:,i)==scale_space(:,:,i));

    % setting a threshold
    [row,col] = find(maxima(:,:,i)>= threshold);

    x = cat(1,x,row); %concat x and r along dim 1
    y = cat(1,y,col); %concat y and c along dim 1
    blob_num = length(row);

    rad_array = sqrt(2) * sigma_init * k^(i-1);
    rad_array = repmat(rad_array,blob_num,1);
    radius = cat(1,radius,rad_array);

end
```

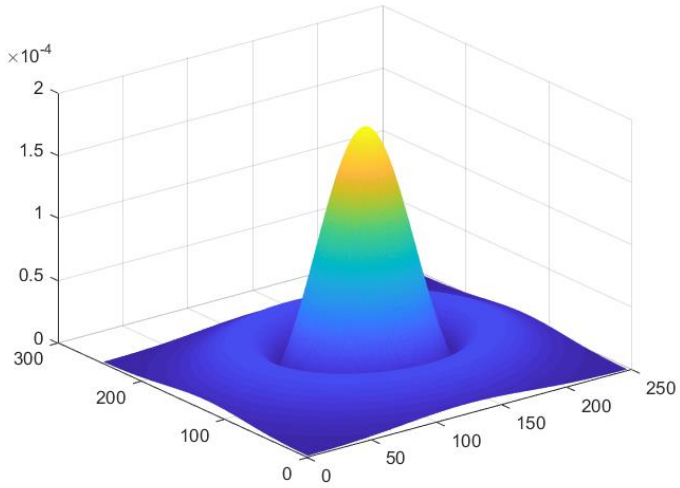
Τέλος, καλώ τη συνάρτηση *show_all_circles* , η οποία είναι δεδομένη.

- *Ανάλυση – Αποτελέσματα*

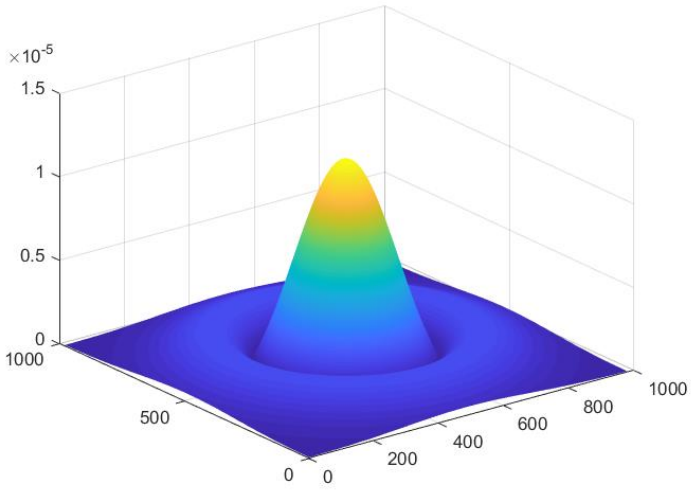
Όσον αφορά την 1η μέθοδο και την εμφάνιση του φίλτρου, παρατηρώ ότι για 10 levels το φίλτρο που έχω σχεδιάσει είναι κατάλληλο, καθώς δεν χάνεται πληροφορία και το ίδιο συμβαίνει και στην υποδειγματοληψία. Εάν αυξήσω τα levels, για παράδειγμα γίνουν 15, τότε πάνω από το 10ο level το μέγεθος του φίλτρου γίνεται μεγαλύτερο από το μέγεθος της εικόνας στην 1η μέθοδο, και στη 2η το μέγεθος της εικόνας γίνεται αμελητέο συγκριτικά με το μέγεθος του φίλτρου (ουσιαστικά δημιουργείται το ίδιο πρόβλημα). Παρακάτω φαίνεται το φίλτρο στα επίπεδα 1,10 και 13.



Level 1



Level 10

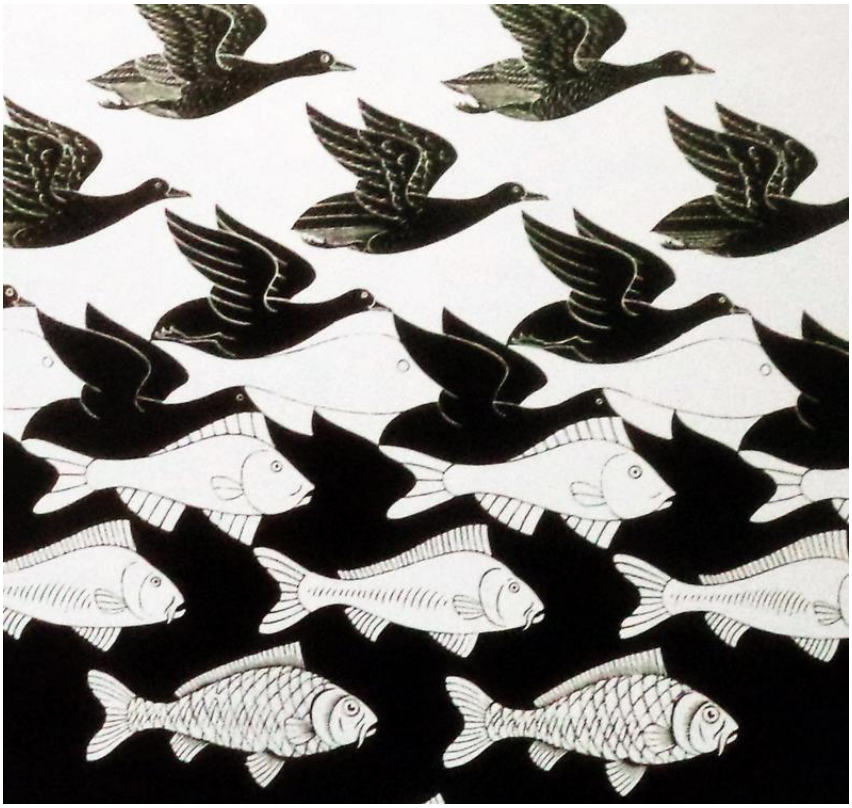


Level 13

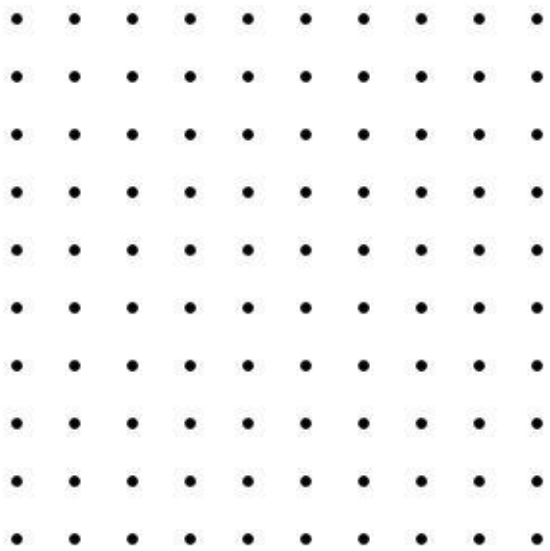
Συγκρίνοντας τις δύο μεθόδους και <u>όσον αφορά το χρόνο</u>, παρατηρούμε ότι η $2^{η}$ μέθοδος είναι σημαντικά πιο γρήγορη από την $1^{η}$ και παρακάτω βλέπουμε τις διαφορές μεταξύ των μεθόδων σχετικά με το χρόνο και τον αριθμό των blobs.

Εικόνα	**Μέθοδος**			
	1: Increasing filter size		**2: Downsampling image**	
	Χρόνος sec	**Αριθμός blobs**	**Χρόνος sec**	**Αριθμός blobs**
dots.jpg	1.25	100	0.06	110
sunflowers.jpg	1.99	934	0.06	937
fishes.jpg	2.30	853	0.07	844
escher.jpg	6.64	1369	0.27	1411

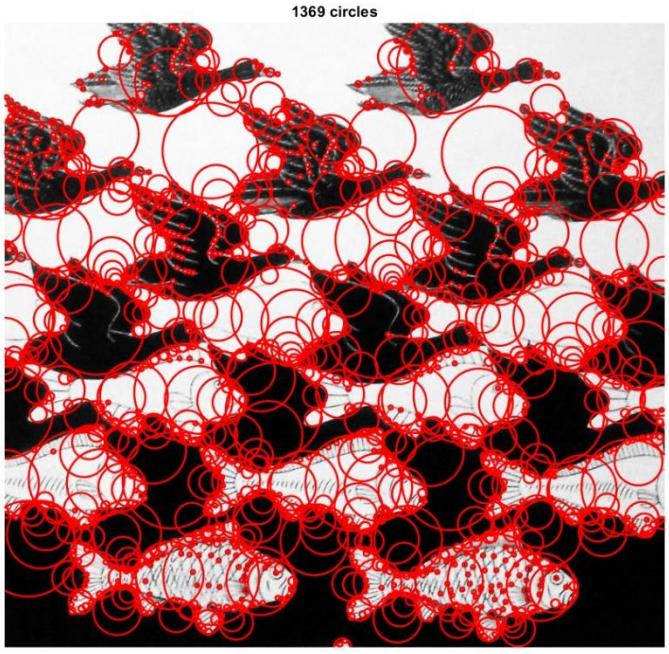
Σχετικά με τις εικόνες που επέλεξα, θεώρησα ότι χρειαζόμασταν εικόνες στις οποίες υπάρχει κάποιο pattern ή κάποιο επαναλαμβανόμενο σχέδιο/σχήμα. Γι' αυτό, σκέφτηκα να χρησιμοποιήσω τον πίνακα του M.C.Escher "Sky and Water I", στο οποίο υπάρχει ένα optical illusion και απεικονίζονται πουλιά στον ουρανό και ψάρια στη θάλασσα. Ο πίνακας μετασχηματίζεται σε κάθε επίπεδό του, προσθέτοντας στα σχέδια επιπλέον πληροφορίες, αλλά τα σχέδια κάθε γραμμής του πίνακα ταυτίζονται μεταξύ τους. Αυτό όμως δεν αποτελεί πρόβλημα, καθώς εγώ ήθελα να δείξω ότι αναγνωρίζονται και τα σχέδια κατά μήκος των επιπέδων των γραμμών, αλλά ότι αντιστοιχίζονται και τα κοινά σχέδια πάνω σε όλη την εικόνα. Παρακάτω παρατίθεται ο πίνακας.

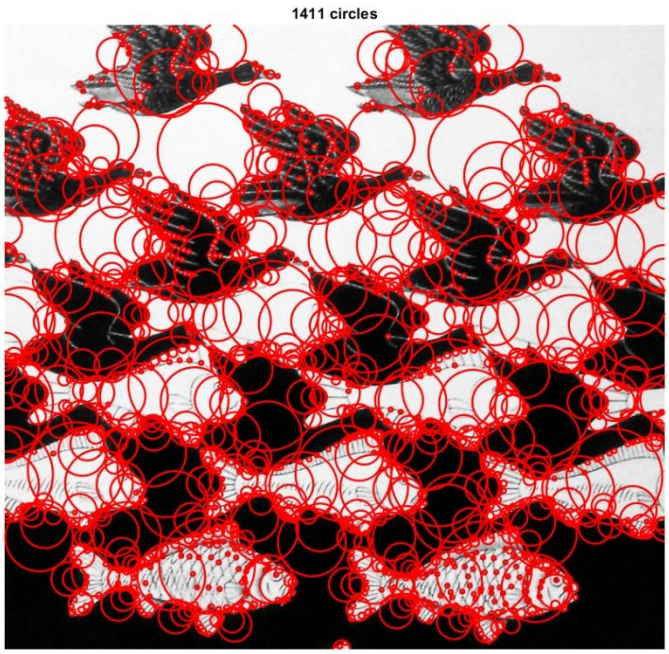


Σχετικά με τη δεύτερη εικόνα, ήθελα να είναι ένα εμφανές pattern για να μπορέσω να διακρίνω καλύτερα τα σφάλματα. Έτσι, διάλεξα μία εικόνα με 100 κουκίδες, όπως φαίνεται παρακάτω.

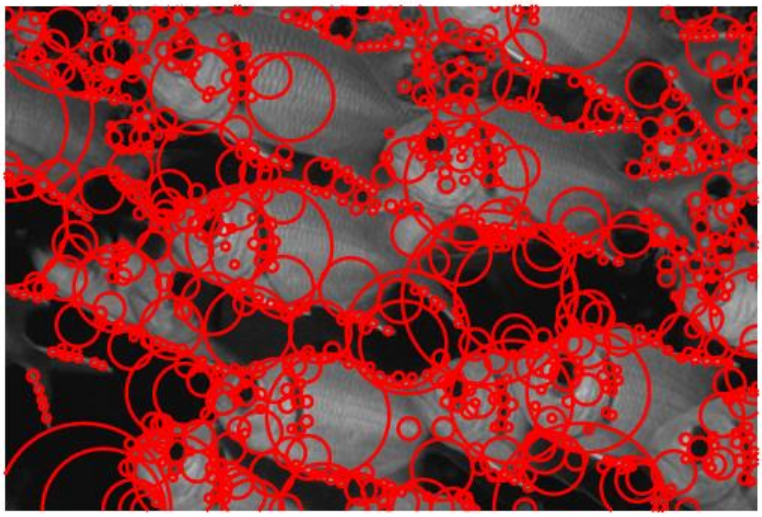


Στη συνέχεια, παραθέτω τις τελικές εικόνες του αλγορίθμου με τα σημεία ενδιαφέροντος. Στις εικόνες αριστερά έχει εφαρμοστεί η μέθοδος 1 με την μεγέθυνση του φίλτρου και δεξιά έχει εφαρμοστεί η μέθοδος 2 με την υποδειγματοληψία της εικόνας.

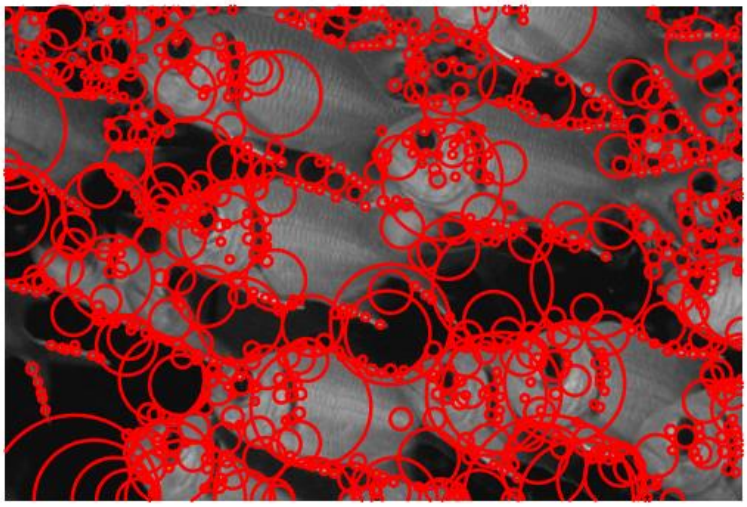




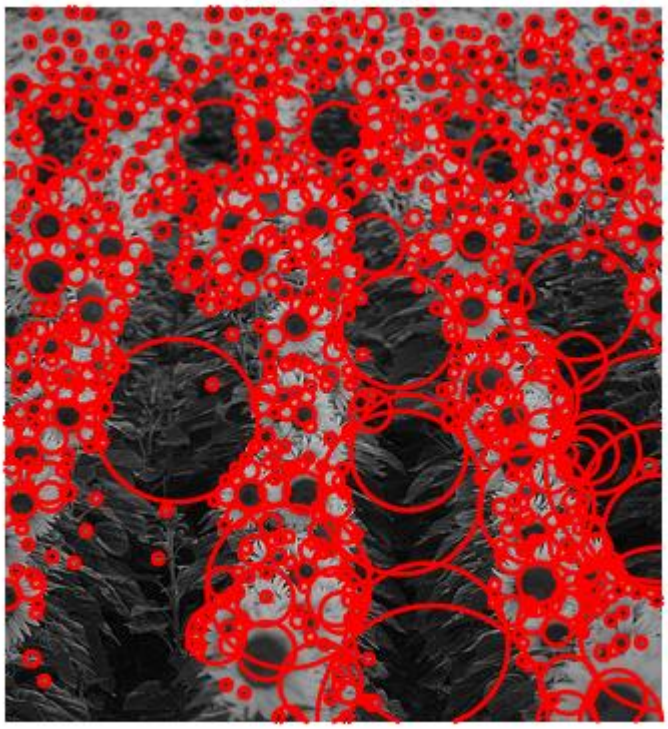
853 circles



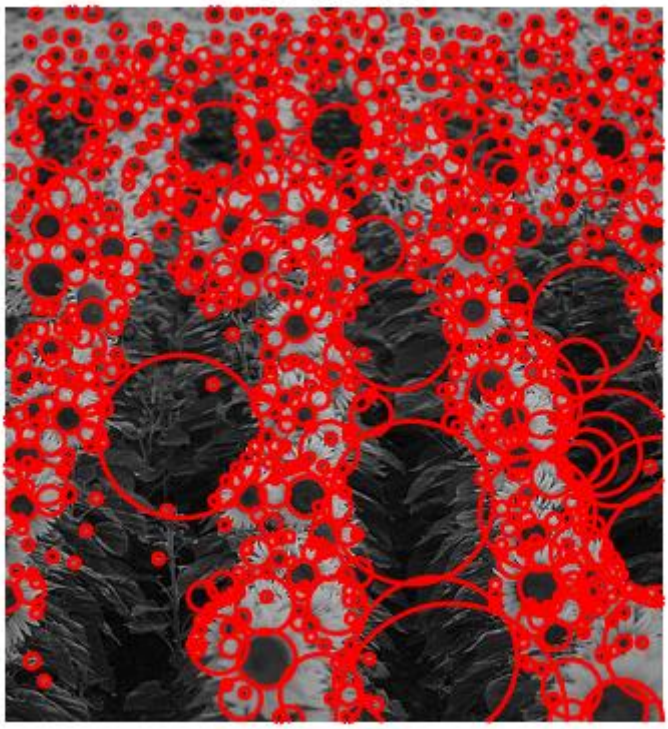
844 circles

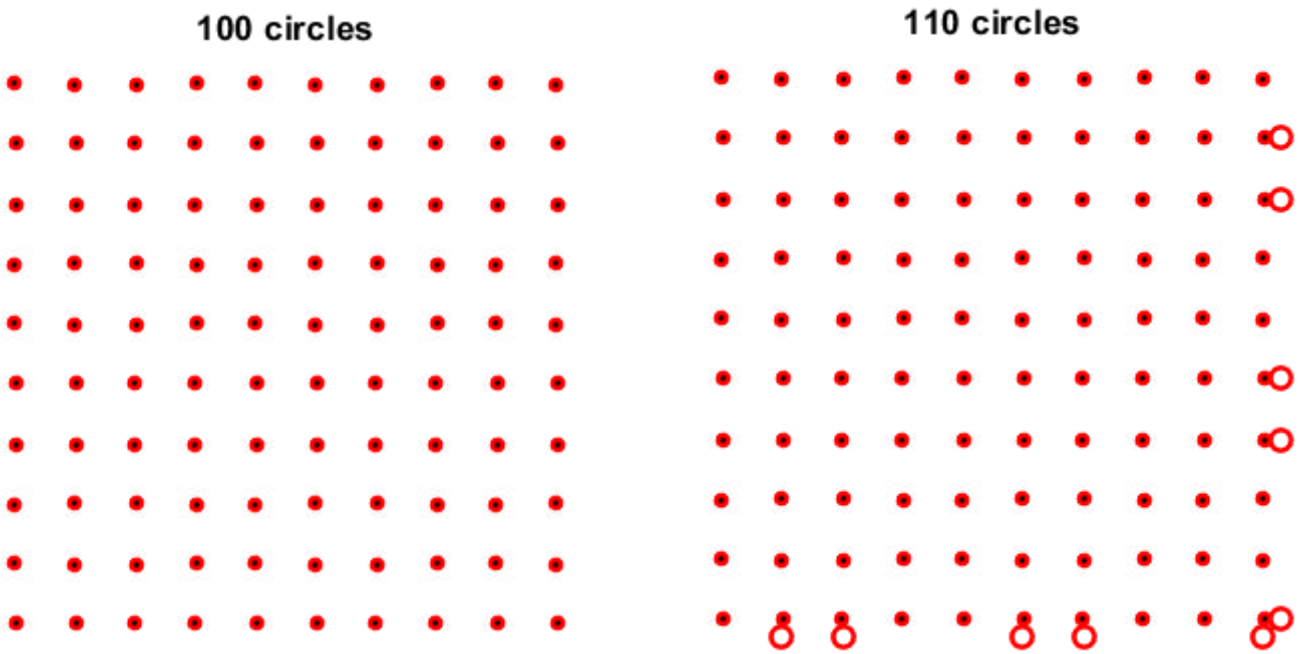


934 circles



937 circles





Παρατηρήσεις – Σχόλια

- Όπως φαίνεται χαρακτηριστικά από τις τελευταίες δύο εικόνες, ορθότερα δουλεύει η 1η μέθοδος κατά την οποία αυξάνουμε το μέγεθος του φίλτρου, παρ' όλο που είναι πιο αργή από την υποδειγματοληψία της εικόνας. Βλέπουμε, μάλιστα, ότι στις περισσότερες περιπτώσεις κατά τη 2η μέθοδο, εμφανίζονται περισσότερα blobs, τα οποία είναι σφάλματα.
- Όσο μεγαλύτερο είναι το threshold, τόσο πιο λίγα σημεία ενδιαφέροντος αποτυπώνονται στην εικόνα, όπως ανέφερα και προηγουμένως στην επεξεργασία του αλγορίθμου. Τα thresholds τα επέλεξα με τη λογική, πρώτον, να μην είναι πάρα πολλά τα σημεία ενδιαφέροντος ώστε να είναι διακριτά με το μάτι, και, δεύτερον, να μην είναι πολύ λίγα, για να μπορώ να εξάγω συμπεράσματα ότι ο αλγόριθμος δουλεύει αποδοτικά.
- Όσο πιο μικρό είναι το threshold, τόσο μεγαλύτερη είναι η διαφορά του αριθμού των blobs μεταξύ των δύο μεθόδων. Για παράδειγμα, για την εικόνα 'fishes.jpg' με threshold = 0.0017 στην 1η μέθοδο έχουμε 1645 κύκλους και στη 2η έχουμε 1720 (διαφορά 75 κύκλων), ενώ με threshold = 0.004 στην 1η μέθοδο έχουμε 1221 κύκλους και στη 2η έχουμε 1267 (διαφορά 46 κύκλων).
- Με τη μέθοδο της δειγματοληψίας, παρατήρησα ότι το κέντρο κάποιων detected blobs έχει μετακινηθεί. Μία πιθανή εξήγηση είναι ότι κατά τη διαδικασία της δειγματοληψίας κάποια pixel άλλαξαν θέση ή τιμή.
- Σχετικά με το $\sigma=1.8$ και το $k=\sqrt{2}$, οι τιμές που επέλεξα βασίζονται στις διαφάνειες του μαθήματος και στο paper του David Lowe. Έκανα διάφορες δοκιμές, αλλά επέλεξα να εφαρμόσω τις συγκεκριμένες τιμές για καλύτερη απόδοση.
- Χρησιμοποιώ 'bicubic' interpolation, καθώς δημιουργεί περισσότερο smooth επιφάνεια από τις άλλες μεθόδους, λαμβάνοντας υπ' όψη περισσότερα pixels (4x4 =16). Αυτό σημαίνει πως η εικόνα μετά την δειγματοληψία θα αποτελεί καλύτερη αναπαράσταση της αρχικής εικόνας, άρα τα στοιχεία ενδιαφέροντος θα είναι με μεγαλύτερη πιθανότητα τα σωστά.
- Για line width των κύκλων έχω επιλέξει την default τιμή 1.5