ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

«ΑΠΟΔΟΤΙΚΗ ΑΝΙΧΝΕΥΣΗ ΠΕΡΙΓΡΑΜΜΑΤΩΝ ΣΕ ΨΗΦΙΑΚΕΣ ΕΙΚΟΝΕΣ ΜΕ ΧΡΗΣΗ ΠΛΗΡΟΦΟΡΙΑΣ ΑΚΜΩΝ»

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μπεκιάρης Θεόδωρος

Επιβλέπων Καθηγητής : Αθανάσιος Νικολαΐδης

ΣΕΡΡΕΣ

ΜΑΙΟΣ 2016

Περιεχόμενα

Πρόλογος

Η πτυχιακή αυτή εργασία με θέμα «**Αποδοτική ανίχνευση περιγραμμάτων σε ψηφιακές εικόνες με χρήση πληροφορίας ακμών**» υλοποιήθηκε υπό την εποπτεία του Αναπληρωτή Καθηγητή Αθανάσιου Νικολαΐδη.

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον κ.Νικολαΐδη για την βοήθεια και την υποστήριξη που μου παρείχε με τις συμβουλές του και την καθοδήγησή του καθ'όλη την πορεία εκπόνησης της εργασίας. Παράλληλα, θα ήθελα να ευχαριστήσω την οικογένεια μου για την υποστήριξη που μου παρείχε όλα αυτά τα χρόνια .

Μπεκιάρης Θεόδωρος

1. Εισαγωγή

Ο στόχος αυτής της πτυχιακής εργασίας είναι η αποδοτική ανίχνευση περιγραμμάτων σε ψηφιακές εικόνες με χρήση πληροφορίας ακμών.Η ανίχνευση περιγραμμάτων σε ψηφιακές εικόνες είναι μια πολύ σημαντική διαδικασία σε διάφορες εφαρμογές όπως η κατάτμηση εικόνας, η ανάκτηση εικόνας, η ανίχνευση και η αναγνώριση αντικειμένων. Πολλές σχετικές τεχνικές έχουν προταθεί μέχρι σήμερα, οι οποίες ανιχνεύουν με μεγάλη ακρίβεια τα περιγράμματα της εικόνας, όμως οι περισσότερες είναι πολύ απαιτητικές υπολογιστικά.

Στα πλαίσια της εργασίας θα μελετηθεί και θα υλοποιηθεί τεχνική ανίχνευσης περιγραμμάτων η οποία απαιτεί μειωμένο υπολογιστικό κόστος. Συγκεκριμένα, θα χρησιμοποιηθεί πληροφορία ακμών για να επιταχυνθεί η ανίχνευση περιγραμμάτων. Αν και οι ανιχνευτές ακμών από μόνοι τους δεν προσφέρονται για ανίχνευση περιγραμμάτων γιατί δεν διακρίνουν περιγράμματα αντικειμένων από περιοχές με έντονη υφή, ωστόσο μπορούν να παρέχουν οικονομικά εκ των προτέρων γνώση στον ανιχνευτή περιγραμμάτων ώστε να επιταχυνθεί η όλη διαδικασία. Ως αποτέλεσμα ο τελικός ανιχνευτής, αν και ταχύτερος, εξακολουθεί να δίνει αποτελέσματα ποιότητας εφάμιλλης με αυτήν των πιο έντονων υπολογιστικά τεχνικών.

1.1 *Η ιστορία των ανιχνευτών*

Το 2004, ο Martin [6] πρότεινε τον κλασσικό ανιχνευτή κλίσης, γνωστό και ώς πρότυπο Pb, στον οποίο ένας συνδυασμός τοπικών στοιχείων φωτεινότητας, χρώματος και καναλιών υφής (texture channels) γίνεται γνωστός μέσω του Berkeley Segmentation Data Set (BSDS300) [7]. Από τότε ,έχει προταθεί και επιβεβαιωθεί ένας μεγάλος αριθμός ανιχνευτών περιγραμμάτων σύμφωνα με αυτό το σύνολο δεδομένων(BSDS300) , όπως ο Boosted Edge Learning (BEL) [8], ο Ultrametric Contour Map (UCM) [9], ο mPb [10] και ο Min-Cover [11]. Μια επιπλέον μέθοδος ανίχνευσης περιγραμμάτων που προτάθηκε ήταν και η επονομαζόμενη καθολική πιθανότητα περιγραμμάτων (gPb) ,η οποία

επιτυγχάνει την μέγιστη ακρίβεια ως προς το F-measure στον BSDS300 και στο εκτεταμένο σύνολο δεδομένων BSDS500.

Παρ'όλα αυτά, ο gPb είναι αρκετά απαιτητικός υπολογιστικά καθώς εξάγει τοπικά χαρακτηριστικά κλίσης σε πολλαπλές κλίμακες και εμπλέκει την επίλυση ενός γενικευμένου προβλήματος ιδιοτιμών (eigenproblem) [17]. Πολλές εφαρμογές του διαδικτύου δεν μπόρεσαν να αξιοποιήσουν έναν τόσο αποτελεσματικό ανιχνευτή περιγραμμάτων λόγω του υψηλού υπολογιστικού του κόστους. Προκειμένου να επιταχυνθεί η διαδικασία ανίχνευσης, προτάθηκε μια μέθοδος βασισμένη στην GPU (Graphics Processing Unit) [13]. Στην πράξη, στις περισσότερες περιπτώσεις όπου οι GPUs είναι πολύ ακριβές ή μη πρακτικές, η βελτιστοποίηση του αλγορίθμου είναι ιδιαιτέρως επιθυμητή. Οπότε, στην σύγχρονη εποχή που ο όγκος των δεδομένων είναι πάρα πολύ μεγάλος, ο τρόπος με τον οποίο θα επιταχύνουμε τους βέλτιστους αλγορίθμους ανίχνευσης και θα διατηρήσουμε παράλληλα την υψηλή απόδοση γίνεται ένα σημαντικό πρόβλημα, το οποίο είναι και ο κύριος στόχος αυτής της εργασίας.

Στην εργασία, θα επιχειρήσουμε να αναλύσουμε την σχέση μεταξύ ανίχνευσης ακμών και ανίχνευσης περιγραμμάτων καθώς και να επιταχύνουμε τους αλγορίθμους ανίχνευσης περιγραμμάτων κάνοντας χρήση της πληροφορίας ακμών. Σε αντίθεση με την ανίχνευση ορίων, η οποία έχει στόχο την ανίχνευση «μεσαίου-επιπέδου» ορίων, αντικειμένων ή σκηνών, η ανίχνευση ακμών εξάγει «χαμηλού-επιπέδου» πληροφορίες όπως απότομες ασυνέχειες στη φωτεινότητα. Παρ'ό,τι αναφέρθηκε[6] πως οι ανιχνευτές ακμών δεν είναι ιδιαιτέρως αποδοτικοί στην ανίχνευση περιγραμμάτων επειδή δεν μπορούν να διακρίνουν περιοχές με υφή από όρια αντικειμένου ή σκηνής, θα δείξουμε στην εργασία πως η πληροφορία ακμών μπορεί να γίνει πολύ χρήσιμη με την έννοια ότι παρέχει οικονομική εκ των προτέρων γνώση για τον ανιχνευτή περιγραμμάτων , βάση της οποίας προτείνεται ενας βασισμένος σε ακμές ανιχνευτής Pb. Ο προτεινόμενος ανιχνευτής επιταχύνει τη διαδικασία ανίχνευσης σχεδόν κατά μία βάση τάξη μεγέθους σε σύγκριση με τον πρότυπο Pb διατηρώντας την υψηλή ακρίβεια. Πρέπει να σημειωθεί ότι άν και κυρίως μελετάμε τον πρότυπο Pb, η ίδια μεθοδολογία μπορεί να εφαρμοστεί και σε άλλους ανιχνευτές βασισμένους στον Pb όπως ο mPb και ο gPb.

2 Ανάλυση του τρόπου λειτουργίας

2.1 *Ο πρότυπος ανιχνευτής Pb και οι βασισμένοι στον Pb ανιχνευτές περιγραμμάτων*

Στον πρότυπο ανιχνευτή Pb , μια συνάρτηση f(x,y,θ) ορίζεται σε κάθε εικονοστοιχείο(pixel) της εικόνας (x,y) ώστε να να προβλέψουμε την πιθανότητα ενός ορίου να έχει προσανατολισμό θ. Λειτουργεί μετρώντας τα τοπικά σημεία που υπολογίζονται από τέσσερα κανάλια : τα l , a ,b απο το χρωματικό χώρο του CIELab και ένα κανάλι υφής που προέρχεται από το texton map. Κάθε τοπικό σημείο υπολογίζεται σαν διαφορά ιστογράμματος μεταξύ των διανομών χαρακτηριστικών σε δύο «ημί-δισκους» με ακτίνα «σ» σε ένα σύνολο 8 προσανατολισμών στο διάστημα [0,π).

Ο Pb ανιχνευτής κατασκευάζεται από τον γραμμικό συνδυασμό τοπικών σημείων στην κλίμακα «σ» και η συνάρτηση είναι η εξής:

$$Pb(x, y, \theta) = \sum_i \alpha_i G_i(x, y, \theta), \qquad (1)$$

Όπου G(x,y,θ) είναι ο operator κλίσης στα (x,y) για τους προσανατολισμούς θ, ο οποίος περιγράφεται στο [6], το «ι» δηλώνει τα κανάλια που χρησιμοποιούνται. Οι σταθερές αι γίνονται γνωστές στο εκπαιδευτικό data set μέσω λογιστικής παλινδρόμησης (logistic regression). Η ισχύς των ορίων στο (x,y) δίνεται από την συνάρτηση :

$$Pb(x, y) = \max_{\theta} Pb(x, y, \theta), \qquad (2)$$

Με βάση την οποία η πιθανότητα του εικονοστοιχείου (x,y) να βρίσκεται σε ένα όριο επιτυγχάνεται με την ομαλοποίηση του Pb(x,y). Ο mPb προτάθηκε αργότερα [10][12] καθώς αξιοποιεί πολυεπίπεδα χαρακτηριστικά για την βελτίωση του Pb. Ο mPb μπορεί να περιγραφεί ώς εξής :

$$mPb(x, y, \theta) = \sum_{s} \sum_{i} \alpha_{i,s} G_{i,s}(x, y, \theta), \qquad (3)$$

Όπου το

$$s \in \{\tfrac{\sigma}{2}, \sigma, 2\sigma\}$$

Ο τελευταίας τεχνολογίας ανιχνευτής περιγραμμάτων gPb [14] παίρνει σαν όρισμα τον mPb για να υπολογίσει φασματικά στοιχεία sPb , και τελικά συνδυάζει τους mPb και sPb . Ο gPb επιτυγχάνει υψηλής ακρίβειας περιγράμματα πάνω σε αρκετά σύνολα δεδομένων αλλά η εφαρμοσιμότητά του περιορίζεται από το απαγορευτικά υψηλό υπολογιστικό κόστος του. Πρόσφατα επιταχύνθηκε με χρήση GPU [13] αλλά απ'όσο γνωρίζουμε δεν έχει προταθεί στη βιβλιογραφία κάποια ξεκάθαρη μέθοδος βελτιστοποίησης στη βιβλιογραφία.

2.2 *Ο ανιχνευτής ακμών*

Η ανίχνευση ακμών αποτελεί ένα από τα θεμελιώδη προβλήματα σε διάφορες εφαρμογές και για αυτό το λόγο έχει γίνει αντικείμενο πολλών ερευνητικών προσπαθειών. Έχει προταθεί μια πληθώρα από αλγόριθμους ανίχνευσης ακμών , όπως ο Sobel operator ,ο Prewitt operator, ο Laplacian ανιχνευτής και ο ανιχνευτής Canny. Για την εργασία , επιλέγουμε τον ανιχνευτή ακμών Canny [15] για να εξάγουμε την πληροφορία ακμών από την εικόνα , λόγω των πολύ ελκυστικών ιδιοτήτων του όπως η γρήγορη ανταπόκριση, η καλή ανίχνευση και η ικανότητα εντοπισμού. Ο πρότυπος Canny αποτελείται από τέσσερα βήματα : εξομάλυνση της εικόνας με ενα φίλτρο Gaussian και εφαρμογή παράγωγων κατευθυντικών φίλτρων Gaussian ώστε να αποκτήσουμε τις κλίσεις των ακμών και τους προσανατολισμούς. Τέλος , κάνουμε πιο λεπτές τις ακμές χρησιμοποιώντας μη-μέγιστη συμπίεση και κατωφλιώνουμε με υστέρηση.

Παρά το γεγονός ότι αποδεικνύεται να είναι μια πολύ αποτελεσματική τεχνική για την ανίχνευση ακμών, δυστυχώς ο ανιχνευτής ακμών Canny δεν αποδίδει ικανοποιητικά στην ανίχνευση περιγραμμάτων [6]. Ο λόγος για τον οποίο ο ανιχνευτής Canny δεν μπορεί να αποδόσει είναι το γεγονός ότι δεν μπορεί να διακρίνει περιγραμμάτα απο περιοχές υψηλής υφής.

Οπότε πλέον τίθεται το ερώτημα : είναι δυνατόν να χρησιμοποιήσουμε ενα ανιχνευτή ακμών όπως ο Canny σε μεθόδους ανίχνευσης περιγραμμάτων ;

3 Ανίχνευση περιγραμμάτων με χρήση πληροφορίας ακμών

3.1 *Παράθεση και ανάλυση του προβλήματος*

Από την εξίσωση (1), παρατηρούμε πως ο Pb υπολογίζει την ισχύ των ορίων για κάθε εικονοστοιχείο στην εικόνα και εξάγει τα τέσσερα τοπικά σημεία κατά μήκος οκτώ προσανατολισμών. Αυτό έχει σαν αποτέλεσμα να υπάρχει μεγάλο υπολογιστικό κόστος. Βλέπουμε πως χρειάζεται πάνω από ένα λεπτό για να εξάγουμε το περίγραμμα μιας εικόνας διαστάσεων 321 x 481 , αν γίνει η διαδικασία σε μονοπύρηνο CPU στην C#. Η ταχύτητα αυτή δεν επαρκεί για εφαρμογές διαδικτύου όπως η δημιουργία μιας μηχανής αναζήτησης εικόνων .

Στην εργασία, στόχος μας είναι η επιτάχυνση των τελευταίας τεχνολογίας ανιχνευτών περιγραμμάτων όσον αφορά τον αλγόριθμο. Υιοθετούμε τις σημειώσεις στο [8] , όπου $S(x,y) = 1$ άν το όριο περνάει από το σημείο (x,y) και $S(x,y) = 0$ σε αντίθετη περίπτωση. Το $F(x,y)$ υποδηλώνει το χαρακτηριστικό διάνυσμα στο σημείο (x,y) και το $P(S(x,y) = 1|e(x,y))$ υποδηλώνει την πιθανότητα το εικονοστοιχείο (x,y) είναι στο όριο δεδομένης της πληροφορίας ακμών. Έτσι, δεδομένου του $F(x,y)$ προκύπτει πως η πιθανότητα του εικονοστοιχείου (x,y) να είναι στο όριο είναι :

$$
\begin{aligned}
&P(S(x,y)|F(x,y)) \\
&= \sum_{i=0}^{1} P(S(x,y), e(x,y) = i|F(x,y)) \\
&= \sum_{i=0}^{1} P(S(x,y)|F(x,y), e(x,y) = i) P(e(x,y) = i|F(x,y))
\end{aligned}
\tag{4}
$$

Μπορούμε εύκολα να υπολογίσουμε την πληροφορία ακμών. Εντωμεταξύ, άν υποθέσουμε πως η πλειοψηφία των εικονοστοιχείων που βρίσκονται πάνω στα όρια, να βρίσκονται επίσης και πάνω στις ακμές είναι πολύ μικρή

, π.χ. P(S(x,y) = 1|e(x,y) = 0), τότε η εξίσωση (4) μπορεί να απλοποιηθεί στην :

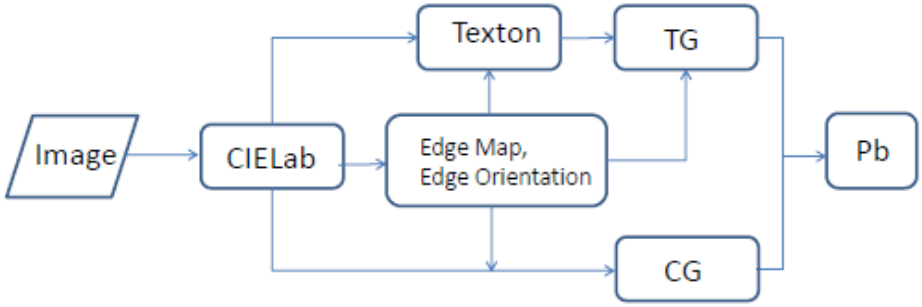
$$
\begin{aligned}
&P(S(x,y) = 1|F(x,y)) \\
&\approx P(S(x,y) = 1|F(x,y), e(x,y) = 1) P(e(x,y) = 1|F(x,y))
\end{aligned}
\tag{5}
$$

Της οποίας η ερμηνεία είναι ότι αν ο χάρτης ακμών περιέχει την πλειοψηφία των εικονοστοιχείων των περιγραμμάτων , μπορούμε να εφαρμόσουμε τον ανιχνευτή περιγραμμάτων στα εικονοστοιχεία των ακμών. Περαιτέρω, η πληροφορία προσανατολισμού των ακμών μπορεί να βοηθήσει ανιχνευτές περιγραμμάτων που βασίζονται σε στοιχεία υφής, όπως διάφορες μεθόδους βασισμένες στον Pb. Στην ιδανική περίπτωση άν ο προσανατολισμός ενός ορίου είναι ίδιος με μιας ακμής , τότε ο Pb μπορεί να απλοποιηθεί στην :

$$Pb(x, y) = \sum_i \alpha_i G_i(x, y, \theta_e), \qquad (6)$$

Όπου το (x,y) ανήκει στο σύνολο υποστήριξης του e , το e είναι ο χάρτης ακμών, και το θ_e είναι ο προσανατολισμός των ακμών στο (x,y). Είναι εμφανές πως ο νέος ανιχνευτής θα μειώσει δραστικά το υπολογιστικό κόστος, κάτι το οποίο θα επιβεβαιωθεί παρακάτω. Βέβαια, αυτό βασίζεται και στο γεγονός ότι θα κάνουμε ιδανικές υποθέσεις για λόγους επίδειξης.

3.2 *Ο ανιχνευτής περιγραμμάτων με βάση την πληροφορία ακμών*



Διάγραμμα ροής συστήματος.Το CG(ColorGradient) υποδηλώνει τις χρωματικές κλίσεις από τα κανάλια L,a και b ενώ το TG(TextonGradient)την κλίση texton.

Παραπάνω παρατηρούμε το διάγραμμα ροής του συστήματος μας για την ανίχνευση περιγραμμάτων. Για εικόνες επιπέδου του γκρί ο αλγόριθμος δουλεύει στο

κανάλι έντασης ενώ για τις έγχρωμες εικόνες τρία κανάλια υπολογίζονται *L*,*a* και *b* στο χρωματικό χώρο CIELab. Έπειτα,συνδυάζεται η πληροφορία ακμών που εξάγεται από κάθε κανάλι και γίνεται μια περαιτέρω λέπτυνση ώστε να δημιουργήσουμε ένα χάρτη ακμών με προσανατολισμούς. Στην πράξη,παρατηρήσαμε πως η εξαγωγή πληροφορίας ακμών από το κανάλι έντασης της έγχρωμης εικόνας είναι αρκετά ικανοποιητική για περαιτέρω επεξεργασία,ειδικά για φυσικές εικόνες.

Με την πληροφορία ακμών μπορούμε να μειώσουμε πάρα πολύ το υπολογιστικό κόστος του πρότυπου ανιχνευτή *Pb* με τις ακόλουθες στρατηγικές: 1) Για τον υπολογισμό των textons, τοποθετούμε δείκτες textons ***μόνο*** στα εικονοστοιχεία τα οποία βρίσκονται μέσα στον δίσκο με ακτίνα *σ* και κέντρο το κάθε εικονοστοιχείο ακμής. 2)Εξάγουμε τοπικά στοιχεία ***μόνο*** στα εικονοστοιχεία ακμών σε ένα υποσύνολο οκτώ προσανατολισμών. Το ιδανικό θα ήταν να εξάγουμε στοιχεία κλίσης κατά τον προσανατολισμό της ακμής. Ωστόσο, υπάρχουν δύο ζητήματα που πρέπει να λάβουμε υπόψιν. Πρώτον, ένας προσανατολισμός ακμής μπορεί να μην είναι σύμφωνος με τον προσανατολισμό ενός ορίου, ιδίως στις θέσεις συνδέσμων στην εικόνα. Δεύτερον, είναι γνωστό στα γραφικά υπολογιστών πως ένα εικονοστοιχείο έχει μια επιφάνεια, και ακόμα και μια οπτικά ευθεία γραμμή με κάποια κλίση στην εικόνα, δεν είναι πραγματικά ευθεία στο πεδίο των εικονοστοιχείων [16].

Αυτό έχει ως αποτέλεσμα να είναι δύσκολος ο ακριβής υπολογισμός του προσανατολισμού των ακμών ή των ορίων.Έτσι, η υπόθεση ότι ο προσανατολισμός των ακμών είναι ίδιος με τον προσανατολισμό των ορίων στον ανιχνευτή *Pb* είναι πολύ αυστηρή και μη πρακτική. Για να χρησιμοποιήσουμε την πληροφορία ακμών ώστε να επιταχύνουμε τον ανιχνευτή *Pb*, χαλαρώνουμε την υπόθεση αυτή λαμβάνοντας υπόψιν τον προσανατολισμό των ακμών που υπολογίσαμε (ο οποίος είναι κβαντισμένος) καθώς και των παρακείμενων προσανατολισμών.

Στην υλοποίηση επιλέγουμε τον ανιχνευτή ακμών Canny, ο οποίος έχει γίνει ένας από τους πιο πετυχημένους ανιχνευτές ακμών από τότε που εφευρέθηκε το 1986 [15].Οι καλές του ιδιότητες ανίχνευσης και εντοπισμού καθιστούν λογική την υπόθεση μας, ενώ η ιδιότητά του της γρήγορης απόκρισης σημαίνει πως εξάγονται λιγότερες θορυβώδεις ακμές και κατά συνέπεια υπάρχουν λιγότερα εικονοστοιχεία για περαιτέρω επεξεργασία. Η διαδικασία που ακολουθούμε είναι ελαφρώς διαφορετική από τον πρότυπο *Pb* επειδή

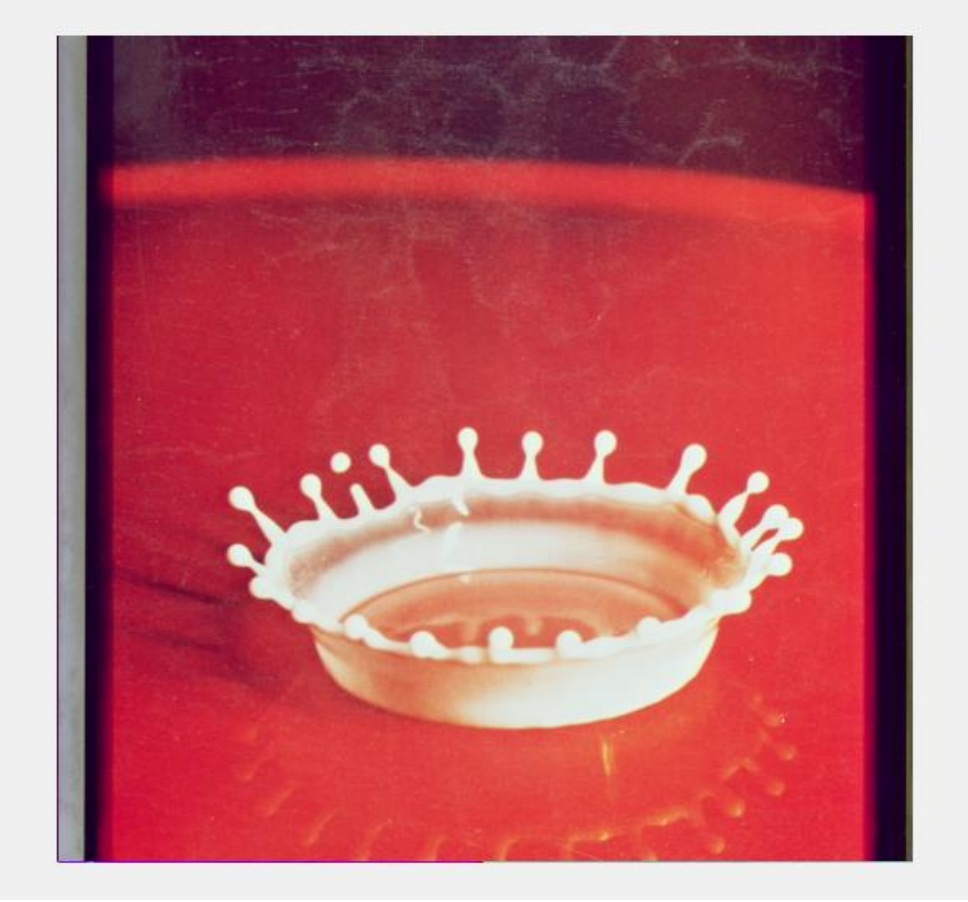
χρησιμοποιούμε μόνο τα τοπικά στοιχεία στον χάρτη ακμών ως θετικά ή αρνητικά χαρακτηριστικά, χωρίς ιδιαίτερη επεξεργασία εξομάλυνσης.

4 Υλοποίηση του ανιχνευτή περιγραμμάτων με βάση την πληροφορία ακμών

4.1 *Δημιουργία του προγράμματος*

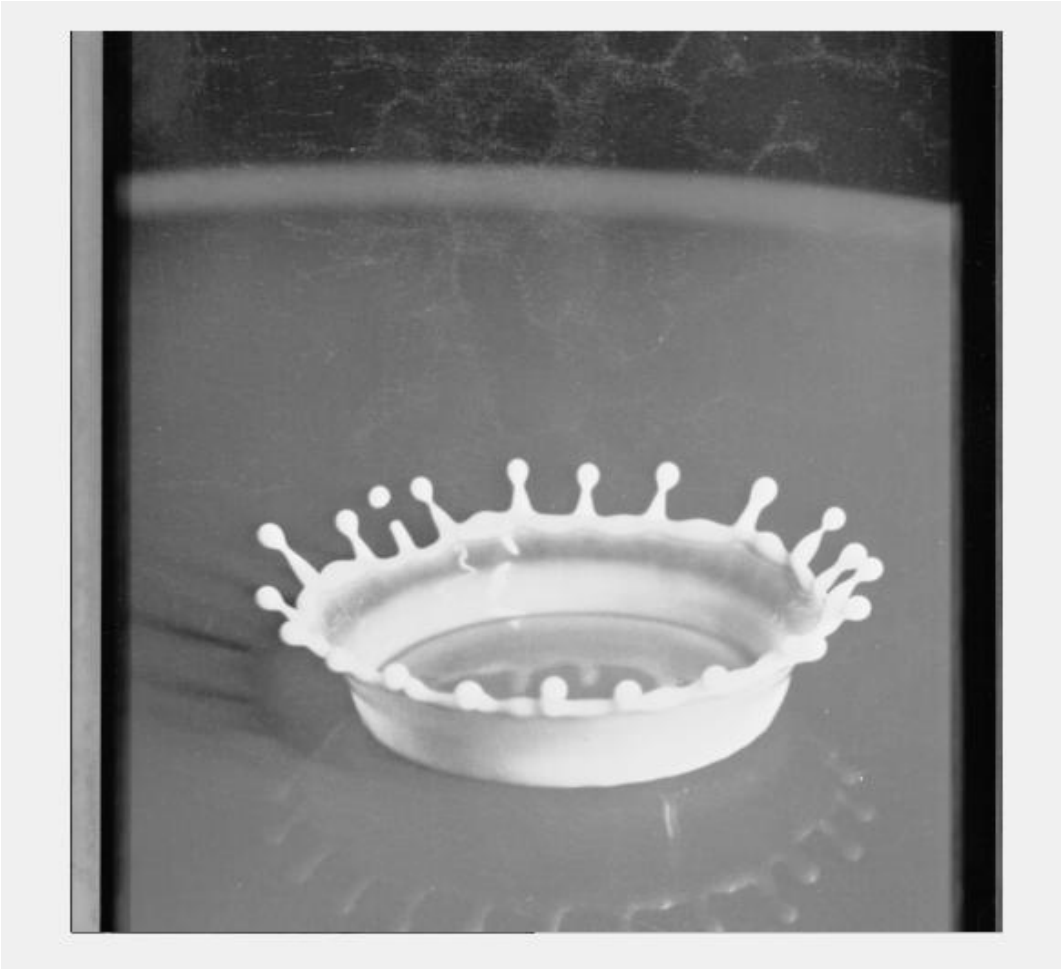
Σε προηγούμενη ενότητα αναφέρθηκε το διάγραμμα ροής του συστήματος καθώς και τα μέρη από τα οποία αποτετελείται. Σε αυτή την ενότητα ,θα αναλύσουμε περαιτέρω τα βήματα που ακολουθήσαμε προκειμένου να δημιουργήσουμε τον ανιχνευτή περιγραμμάτων με βάση την πληροφορία ακμών όσον αφορά τον κώδικα από τον οποίο αποτελείται και το πως λειτουργεί .

Σε πρώτο στάδιο , επιλέγουμε μια εικόνα με τις επιθυμητές διαστάσεις. Αυτή η εικόνα θα χρησιμοποιηθεί σαν όρισμα στο χρωματικό χώρο CIELab. Ο λόγος για τον οποίο επιλέγουμε το CIELab είναι ότι είναι σχεδιασμένο ώστε να προσεγγίζει την ανθρώπινη όραση σε αντίθεση με τα χρωματικά μοντέλα RGB και CMYK. Έτσι υπολογίζουμε τρία κανάλια L, a και b. Στο συγκεκριμένο παράδειγμα επιλέγουμε την εικόνα splash.tiff διαστάσεων 512x512.

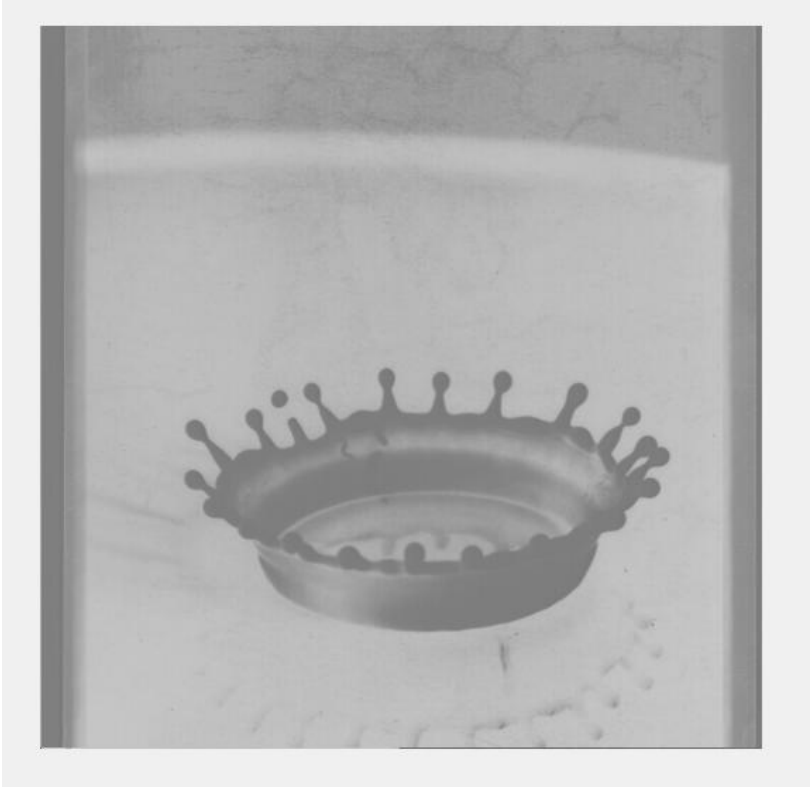


Εικόνα διαστάσεων 512x512 (1)

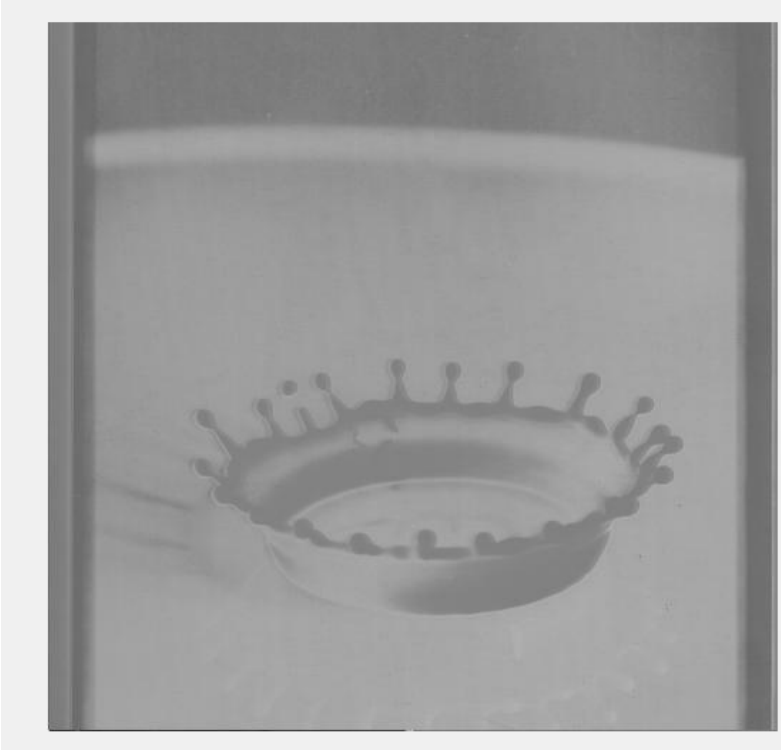
Αφού ολοκληρωθεί η επεξεργασία στο χρωματικό χώρο CIELab μπορούμε να εμφανίσουμε τα τρία κανάλια που έχουν δημιουργηθεί L, a και b. Το κανάλι ***L*** ταιριάζει περισσότερο με την ανθρώπινη αντίληψη της φωτεινότητας ενώ τα κανάλια ***a*** και ***b*** αντιπροσωπεύουν τη σχέση πράσινου-κόκκινου και μπλε-κίτρινου.



Κανάλι L(2)



Κανάλι a (3)



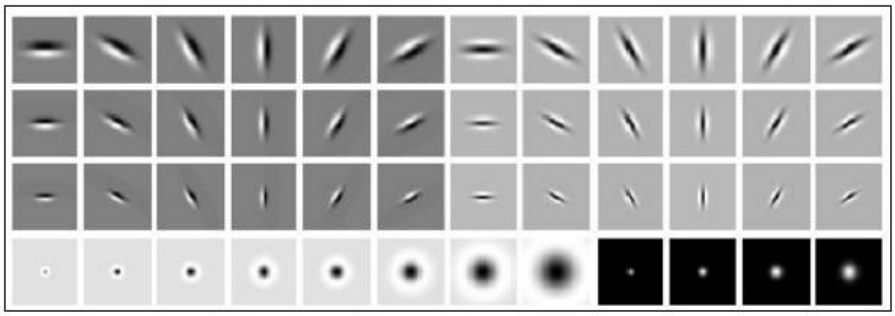
Κανάλι b (4)

Στην περίπτωση μας, προκειμένου να δημιουργήσουμε τον χάρτη των ακμών και τους προσανατολισμούς τους, χρησιμοποιούμε σαν όρισμα το ***L*** στον ανιχνευτή ακμών Canny επειδή στην πράξη η εξαγωγή πληροφορίας ακμών από το κανάλι έντασης της έγχρωμης εικόνας είναι αρκετά ικανοποιητική για περαιτέρω επεξεργασία, ειδικά για τις φυσικές εικόνες. Επίσης θα πρέπει να βρούμε τον προσανατολισμό που έχει η εκάστοτε ακμή στην εικόνα μας κάτι το οποίο θα επιταχύνει ραγδαία την επεξεργασία της εικόνας.

Πληροφορία ακμών του καναλιού l (5)

4.2 *Η συστοιχία φίλτρων Leung Malik*

Παράλληλα με την δημιουργία του χάρτη ακμών , θέλουμε και την δημιουργία των textons τα οποία θα βοηθήσουν στην ραγδαία μείωση του υπολογιστικού κόστους. Όπως αναφέραμε και σε προηγούμενη ενότητα θέλουμε να τοποθετήσουμε δείκτες textons ***μόνο*** στα εικονοστοιχεία τα οποία βρίσκονται μέσα στον δίσκο με ακτίνα «σ» στο κέντρο του κάθε εικονοστοιχείου. Για να φτάσουμε σε εκείνο το σημείο πρέπει να ακολουθήσουμε την εξής διαδικασία: Πρώτον, χρειαζόμαστε μια συστοιχία φίλτρων (filterbank). Εμείς επιλέγουμε την Leung Malik(LM) filter bank [17] η οποία αποτελείται από 48 φίλτρα. Είναι μια πολλαπλών κλιμάκων και προσανατολισμών συστοιχία φίλτρων η οποία αποτελείται από πρώτες και δεύτερες παραγώγους Gaussian σε 6 προσανατολισμούς και 3 κλίμακες -δηλαδή ένα σύνολο από 36- , 8 φίλτρα LaplacianofGaussian (LOG) και 4 Gaussian φίλτρα.



Τα 48 φίλτρα τα οποία αποτελούνται από πρώτες και δεύτερες παραγώγους Gaussian σε 6 προσανατολισμούς και 3 κλίμακες ,αθροίζοντας 36, 8 φίλτρα Laplacian of Gaussian (LOG) και 4 Gaussian φίλτρα. (6)

Εφόσον δημιουργήσουμε τη συστοιχία φίλτρων μεγέθους 49x49x48 κάνουμε συνέλιξη της filterbank με το κανάλι της φωτεινότητας που εξήγαμε πριν στο CIELab.Το επόμενο βήμα μας είναι να αλλάξουμε τις διαστάσεις της εικόνας μας που περιέχει τις ακμές δημιουργώντας αντίγραφα ώστε να ταιριάζουν με τα αποτελέσματα της συνέλιξης μεταξύ l2 και συστοιχίας φίλτρων (δηλαδή 512x 512 x 48),ώστε να συνδυαστεί η πληροφορία ακμών και οι προσανατολισμοί τους με το αποτέλεσμα της συνέλιξης μεταξύ του χρωματικού καναλιού *L* και της συστοιχίας φίλτρων.

Για να επιταχυνθεί η διαδικασία θέλουμε να επιλέξουμε όλα τα μη μηδενικά σημεία του προηγούμενου αποτελέσματος (συνδυασμός πληροφορίας ακμών και αποτέλεσμα συνέλιξης καναλιού *L* και filterbank) για να μειωθεί κατά πολύ το υπολογιστικό κόστος. Εξάγουμε λοιπόν αυτά τα σημεία τα οποία στην συγκεκριμένη εικόνα είναι 18.377 από ένα σύνολο των 262.144.

Προκειμένου να βρούμε τα textons πρέπει να κάνουμε χρήση του k-means clustering, δηλαδή μιας μεθόδου συσταδοποίησης.Πιο αναλυτικά είναι μια μέθοδος κβαντισμού διανυσμάτων, η οποία προέρχεται από την επεξεργασία σήματος και που είναι δημοφιλής όσον αφορά την ανάλυση συστάδων για εξαγωγή δεδομένων. Ο τρόπος με τον οποίο λειτουργεί είναι πως διαχωρίζει τις παρατηρήσεις σε ένα matrix n-by-p, σε k clusters και επιστρέφει ένα n-by-1 διάνυσμα (idx) το οποίο εμπεριέχει δείκτες συστάδων για κάθε παρατήρηση. Οι σειρές του x αντιστοιχούν σε σημεία και οι στήλες αντιστοιχούν σε μεταβλητές.

Επίσης, θα πρέπει να βρούμε τον προσανατολισμό που έχει η εκάστοτε ακμή στην εικόνα μας κάτι το οποίο όπως αναφέραμε σε προηγούμενη ενότητα θα επιταχύνει ραγδαία την επεξεργασία της εικόνας.

4.3 *Η κλίση των textons (TextonGradient)*

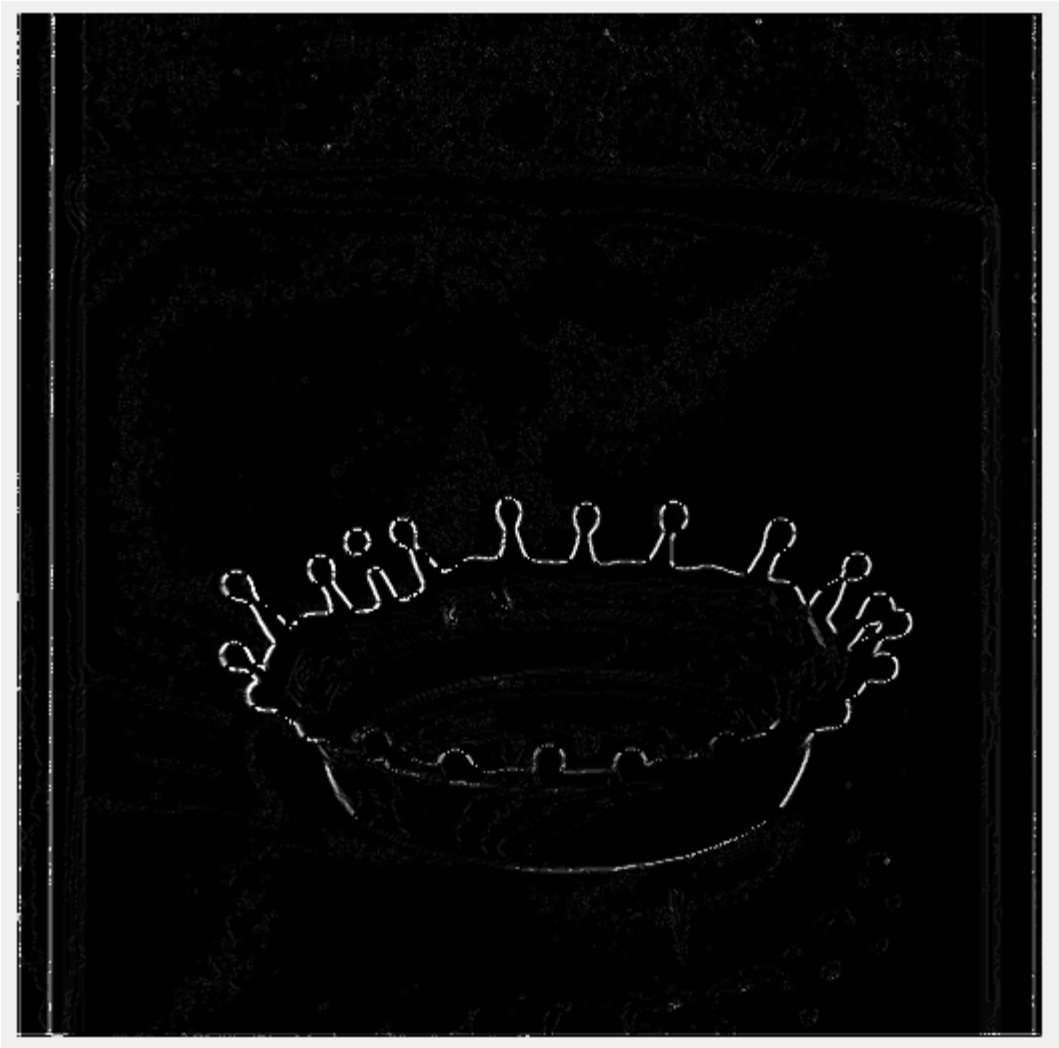
Έτσι, λοιπόν, δίνουμε σαν όρισμα στο k-means το **v** το οποίο έχει το σύνολο των 18.377 σημείων της εικόναςκαι τον αριθμό των συστάδων (clusters) που επιθυμούμε(εμείς επιλέγουμε 32 clusters).Αφού ολοκληρωθεί η διαδικασία του clustering μπορούμε πλέον να βρούμε το textongradient δηλαδή τις διαφορές γειτονικών τιμών όσον αφορά τα textons.Με τον όρο κλίση εικόνας (imagegradient) αναφερόμαστε στην αλλαγή κατεύθυνσης στην ένταση ή στο χρώμα μιας εικόνας και μπορεί να χρησιμοποιηθεί για την εξαγωγή πληροφοριών από την εικόνα.Για να βρούμε το textongradient χρησιμοποιούμε σαν όρισμα, το διάνυσμα που βρήκαμε προηγουμένως το

οποίο περιέχει τους δείκτες συστάδων, τον αριθμό των συστάδων (clusters) που θέλουμε, την ακτίνα δίσκου στην οποία θα εφαρμοστεί, και τον αριθμό των προσανατολισμών και τον πίνακα ομοιότητας των textons. Έπειτα, δημιουργούμε έναν πίνακα από μηδενικά διαστάσεων 512x512 πάνω στον οποίον τοποθετούμε στις σωστές συντεταγμένες και με τον σωστό προσανατολισμό της εκάστοτε ακμής, τα μη μηδενικά σημεία που έχουμε βρει στο textongradient. Το αποτέλεσμα είναι άκρως ικανοποιητικό και είναι το εξής:

Το Textongradient (7)

4.4 *Η κλίση χρώματος*

Στη συνέχεια θέλουμε να βρούμε το colorgradient (δηλαδή τις *χρωματικές* διαφορές γειτονικών τιμών).Στην περίπτωσή μας, χρησιμοποιούμε σαν ορίσματα την εικόνα l3,τον αριθμό των textons (32), την ακτίνα δίσκου στην οποία θα εφαρμοστεί καθώς και τον αριθμό των προσανατολισμών. Δημιουργούμε ένα πίνακα από μηδενικά διαστάσεων 512x512 και τοποθετούμε τα αποτελέσματα του colorgradient. Το αποτέλεσμα της επεξεργασίας είναι το εξής:



Το Colorgradient (8)

4.5 *Το τελικό στάδιο του ανιχνευτή*

Η συνάρτηση που υλοποιήσαμε ερμηνεύεται ως εξής: Το *Pb* διαστάσεων *x,y* είναι ένα σταθμισμένο άθροισμα σε όλα τα κανάλια (εμείς χρησιμοποιήσαμε μόνο το *L* γιατί ικανοποιεί τις ανάγκες του εγχειρήματός μας) των τελεστών κλίσης G_i για τον συγκεκριμένο προσανατολισμό θ_e της ακμής του σημείου (x, y):

$$Pb(x, y) = \sum_i \alpha_i G_i(x, y, \theta_e), \qquad (6)$$

Τα a_i είναι κάποιες σταθερές οι οποίες προκύπτουν μέσω της τεχνικής της λογιστικής παλινδρόμησης με είσοδο χαρακτηριστικά από ένα μεγάλο σύνολο εικόνων. Οπότε, αυτό που μένει πλέον είναι να συνδυάσουμε το αποτέλεσμα των textongradient και colorgradient ώστε να δούμε το τελικό αποτέλεσμα επεξεργασίας του ανιχνευτή μας.

Το τελικό αποτέλεσμα του ανιχνευτή περιγραμμάτων με βάση την πληροφορία ακμών (edge based contour detector). Αποτέλεσμα συνδυασμου του texton gradient (7) και του color gradient (8). (9)

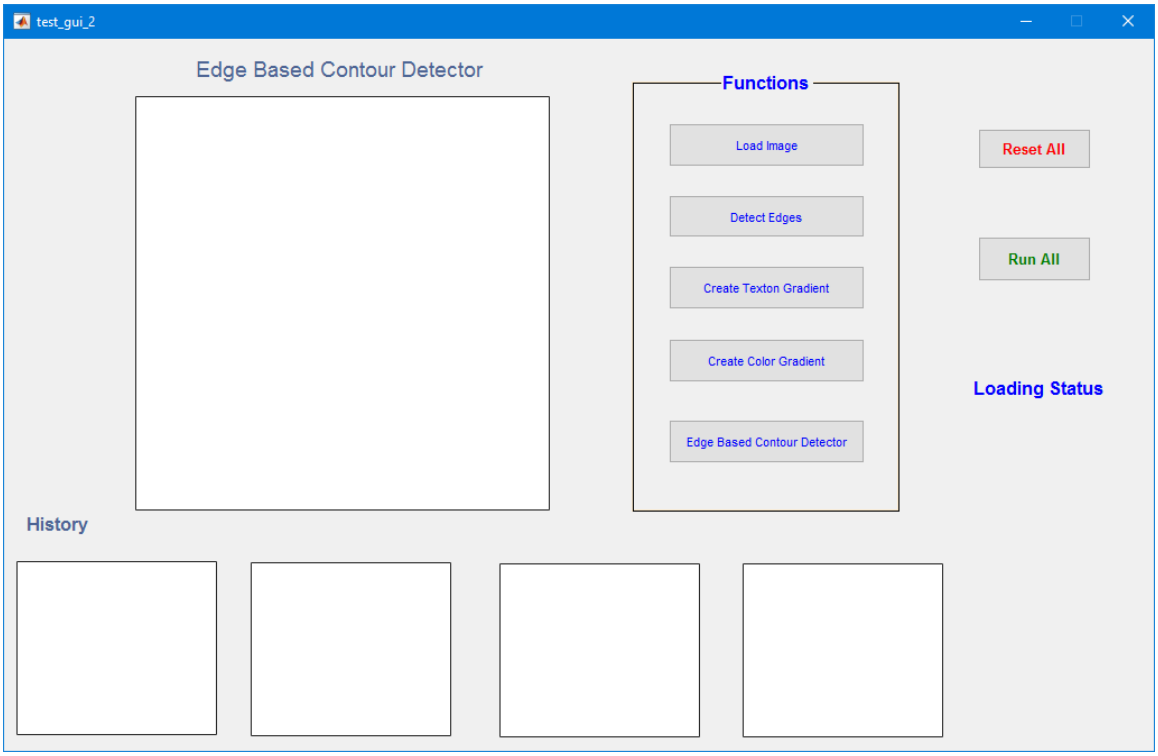
Στην εργασία υλοποιήσαμε έναν γρήγορο αλγόριθμο ανίχνευσης περιγραμμάτων ο οποίος συνδυάζει επί της ουσίας τον ανιχνευτή ακμών Canny και τον πρότυπο *Pb*. Τα αποτελέσματα μας δείχνουν πως ο προτεινόμενος ανιχνευτής λειτουργεί πιο γρήγορα από τον πρότυπο *Pb* διατηρώντας παράλληλα την επιθυμητή απόδοση. Έτσι, διαπιστώνουμε πως ο ανιχνευτής Canny είναι ικανός να ανιχνεύει τα περιγράμματα που προεξέχουν

στην εικόνα και κατά συνέπεια μπορεί να χρησιμοποιηθεί για να επιταχυνθεί ο *Pb* καθώς και άλλοι υπάρχοντες αλγόριθμοι ανίχνευσης περιγραμμάτων. Η τεχνική αυτή καταφέρνει να γεφυρώσει το κενό μεταξύ ανίχνευσης ακμών και ανίχνευσης περιγραμμάτων με την έννοια ότι ο ανιχνευτής ακμών παρέχει εύκολα πληροφορίες σχετικά με την θέση και τον προσανατολισμό για τον ανιχνευτή περιγραμμάτων. Η καλή απόδοση και υψηλή αποτελεσματικότητα του υλοποιούμενου ανιχνευτή, τον καθιστά ιδανικό προς εφαρμογή και πολλά υποσχόμενο για εφαρμογές μεγάλης κλίμακας.

5 Το γραφικό περιβάλλον χρήστη

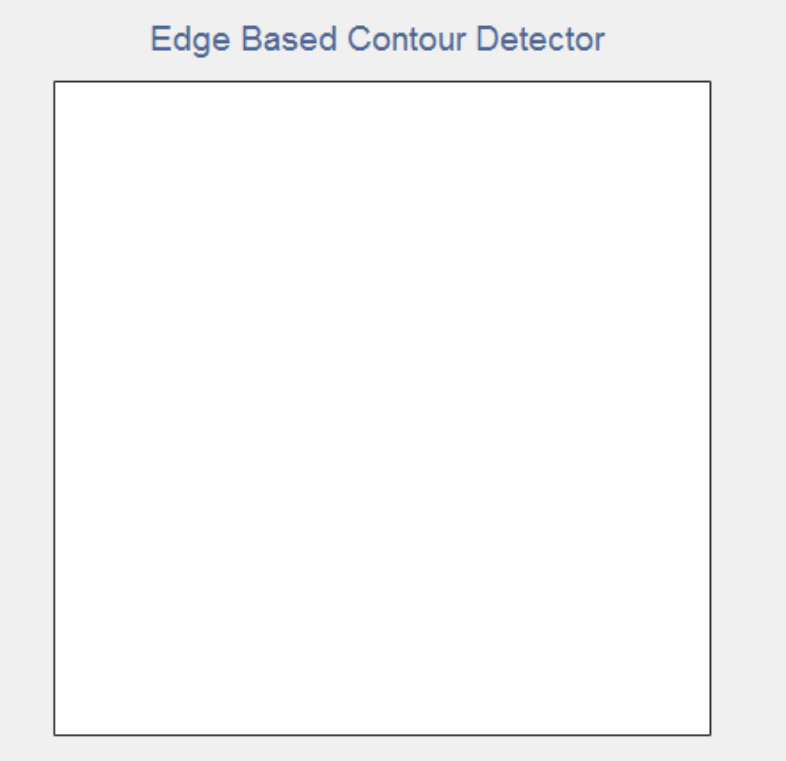
5.1 *Ανάλυση του γραφικού περιβάλλοντος χρήστη (GUI)*

Προκειμένου να ολοκληρωθεί ο ανιχνευτής περιγραμμάτων με βάση την πληροφορία ακμών , δημιουργήσαμε ένα πρακτικό , εύχρηστο και ελκυστικό γραφικό περιβάλλον. Για την δημιουργία του χρησιμοποιήσαμε το **GUIDE**(graphical user interface development environment) του Matlab το οποίο παρέχει πολλές δυνατότητες τροποποίησης και είναι εύχρηστο ακόμα και σε όχι ιδιαίτερα εξοικειωμένους χρήστες με το αντικείμενο. Παρακάτω θα ακολουθήσουν αναλυτικές οδηγίες χρήσης του γραφικού περιβάλλοντος που δημιουργήσαμε βήμα-βήμα .

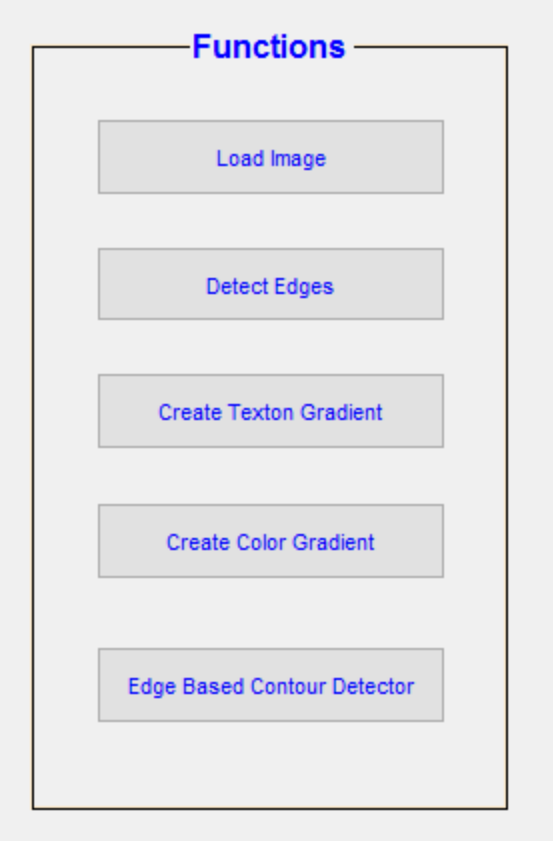


Το γραφικό περιβάλλον χρήστη (GUI)

Στα αριστερά,κάτω από την επιγραφή EdgeBasedContourDetector βρίσκεται ένα μεγάλο πλαίσιο στο οποίο εμφανίζεται το στάδιο επεξεργασίας στο οποίο βρίσκεται η εικόνα μας κάθε φορά. Στα δεξιά του , βρίσκεται ένα πλαίσιο με τις λειτουργίες του προγράμματος μας με την επιγραφή Functions. Συνεχίζοντας , έχουμε δύο κουμπιά με την επιγραφή Reset All(Επανέφερε όλα) και Run all (Τρέξε όλα),επίσης υπάρχει το Loading Status (Στάδιο φόρτωσης) στο οποίο γεμίζει η μπάρα ανάλογα με το στάδιο που βρίσκεται το πρόγραμμα . Τέλος , στο κάτω μέρος έχουμε μια σειρά από πλαίσια στα οποία θα εμφανίζονται τα διαφορετικά στάδια επεξεργασίας μέχρι το τελικό στάδιο που είναι και το ζητούμενο μας.



Το πλαίσιο στο οποίο εμφανίζεται κάθε στάδιο επεξεργασίας που περνάει η εικόνα μας και εν τέλει το τελικό αποτέλεσμα του Edge Based Contour Detector



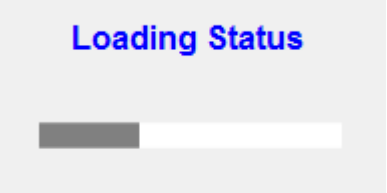
Το πλαίσιο Functions στο οποίο εμφανίζονται οι επιλογές που έχουμε για την επεξεργασία της εικόνας βήμα-βήμα

Στο πρόγραμμά μας έχουμε την επιλογή είτε να ορίσουμε την εικόνα για επεξεργασία και να «τρέξουμε» όλο το πρόγραμμα μας, είτε να «τρέξουμε» το πρόγραμμα βήμα-βήμα. Στο πλαίσιο που παρατίθεται πάνω με την επιγραφή Functions (Λειτουργίες) βλέπουμε τα βήματα τα οποία έχουμε ορίσει και αριθμούν επτά. Αρχικά έχουμε την επιλογή **LoadImage**(Φόρτωσε την εικόνα) το οποίο μας δίνει τη δυνατότητα να φορτώσουμε οποιαδήποτε εικόνα,δημιουργεί και εφαρμόζει τη συστοιχία φίλτρων στην αρχική εικόνα. Συνέχεια έχει το **DetectEdges** (Ανίχνευσε ακμές) το οποίο εφαρμόζει τον ανιχνευτή ακμών Canny στο χρωματικό κανάλι *L*, βρίσκει τον προσανατολισμό των ακμών και πολλαπλασιάζει τα σημεία των ακμών με τις αποκρίσεις των φίλτρων. Ακολουθούν τα βήματα **CreateTextonGradient** (Δημιούργησε το TextonGradient) στο οποίο βρίσκουμε τις μη μηδενικές τιμές μας και τις συντεταγμένες

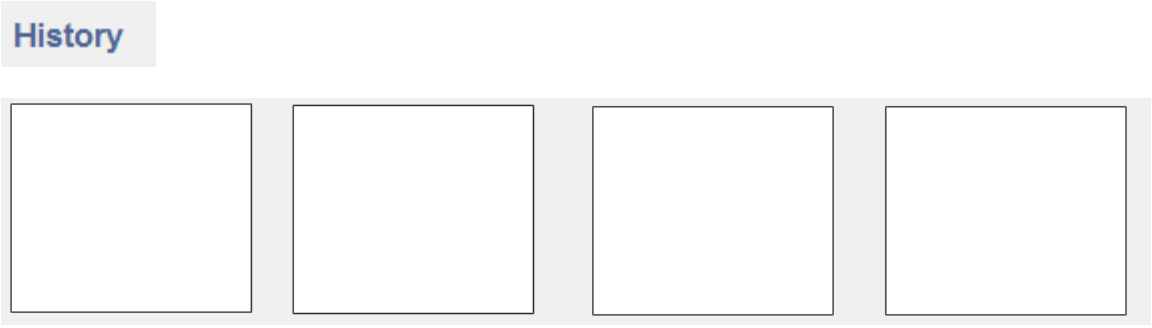
τους και δημιουργούμε την κλίση των textons και **CreateColorGradient** (Δημιούργησε το Colorgradient) στο οποίο δημιουργούμε την κλίση χρώματος. Στο τέλος έχουμε το **EdgeBasedContourDetector** (Ανιχνευτής περιγραμμάτων με χρήση πληροφορίας ακμών) το οποίο κάνει χρήση των ακμών, του Textongradient και Colorgradient και εμφανίζει στο μεγάλο πλαίσιο το τελικό αποτέλεσμα της επεξεργασίας.

Μπορούμε βέβαια με την επιλογή RunAll να ορίσουμε την εικόνα που θέλουμε και το πρόγραμμα να τρέξει όλες τις εντολές εμφανίζοντας το τελικό αποτέλεσμα στο μεγάλο πλαίσιο και τα στάδια επεξεργασίας που προηγήθηκαν στα πλαίσια του ιστορικού. Με την επιλογή ResetAll σβήνουμε τις εικόνες που έχουν εμφανιστεί και μπορούμε να ορίσουμε μία νέα εικόνα προς επεξεργασία.

Τα κουμπιά ResetAll και RunAll για την ολική επαναφορά του προγράμματος και το «τρέξιμο» όλου του προγράμματος αντίστοιχα

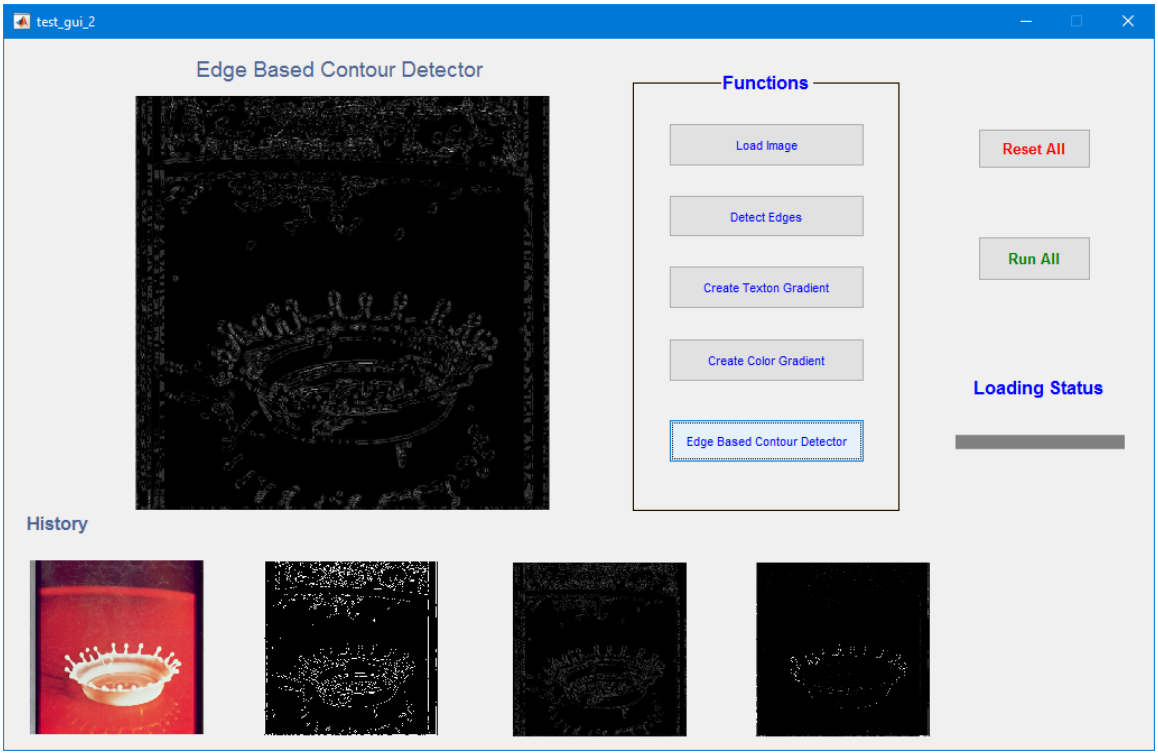


Το Loading Status (Στάδιο φόρτωσης) το οποίο αποτελείται απο μία μπάρα η οποία δείχνει το στάδιο στο οποίο βρίσκεται το πρόγραμμα.



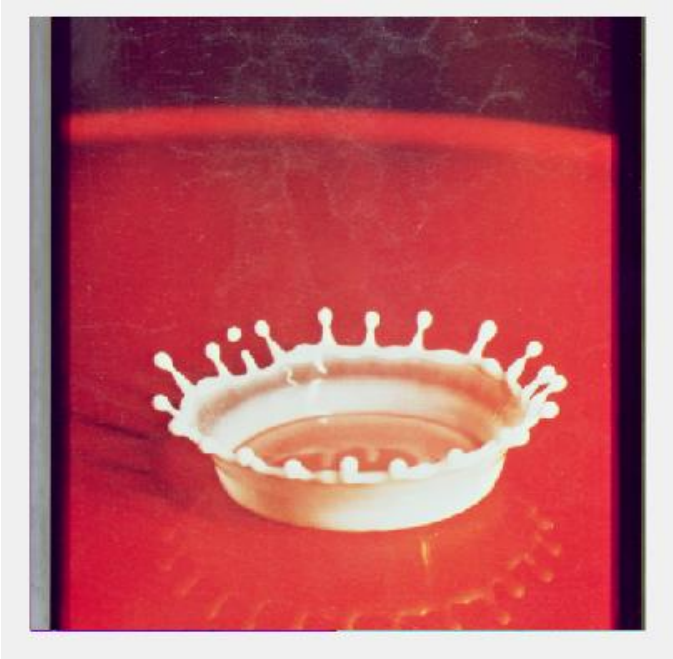
Το ιστορικό των διαφόρων σταδίων επεξεργασίας της εικόνας μας , κατά την ολοκλήρωση του εμφανίζεται σε κάθε πλαίσιο ένα από τα βήματα μέχρι το τελικό βήμα το ανιχνευτή μας

Το γραφικό περιβάλλον μας θα μοιάζει έτσι αφού ολοκληρωθεί η επεξεργασία της εικόνας .



5.2 *Παράθεση και σύγκριση αποτελεσμάτων επεξεργασίας στον Edge Based Contour Detector*

Για την εικόνα **splash**

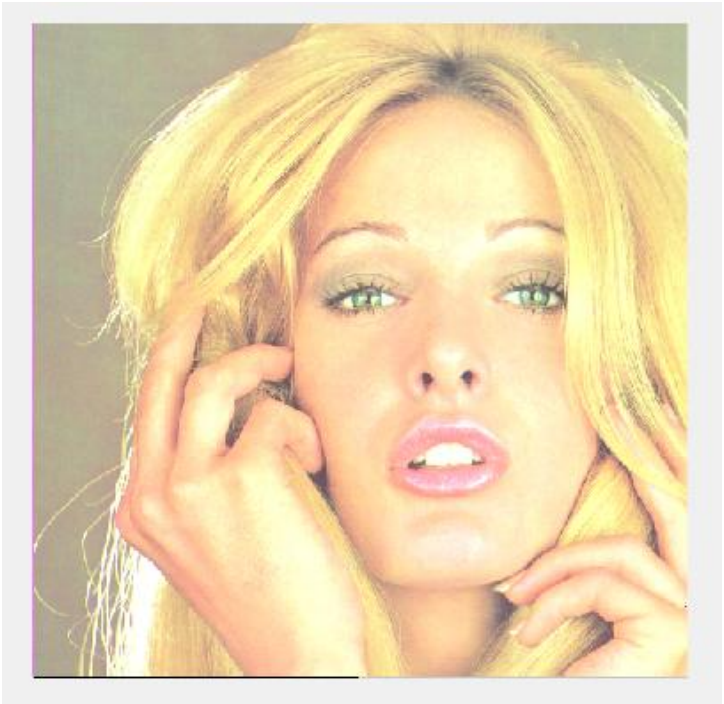


Αρχική εικόνα

Ακμές της αρχικής εικόνας

Αποτελέσματα του Edge based Contour Detector

Για την εικόνα **girl**

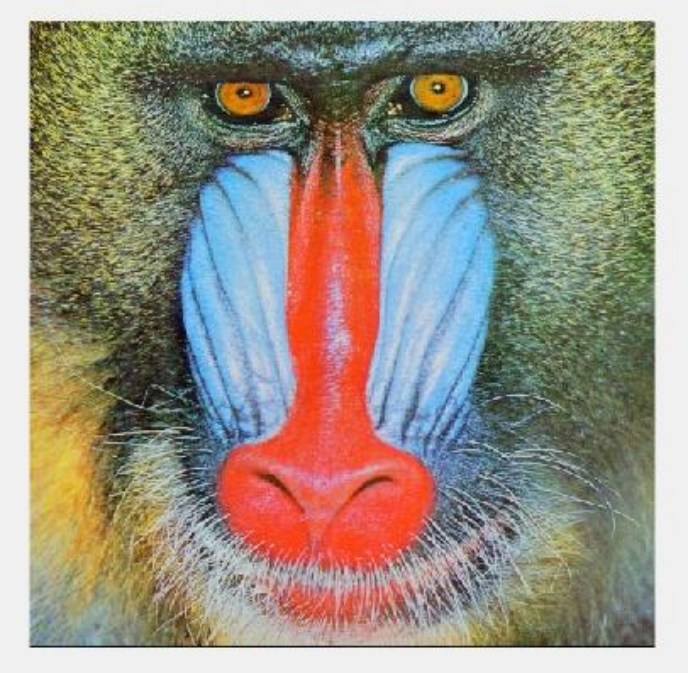


Αρχική εικόνα

Ακμές της αρχικής εικόνας

Αποτελέσματα του Edge based Contour Detector

Για την εικόνα **baboon**



Αρχική εικόνα

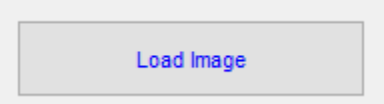
Ακμές της αρχικής εικόνας

Αποτελέσματα του Edge based Contour Detector

Παρατηρούμε λοιπόν πως ο ανιχνευτής περιγραμμάτων μας λειτουργεί διαφορετικά σε κάθε εικόνα και επιστρέφει πιο λεπτομερή περιγράμματα κάτι το οποίο σχετίζεται με το πλήθος των μη μηδενικών σημείων κάθε εικόνας. Στην εικόνα splash είναι 18.377 σημεία, στην εικόνα girl είναι 22.513 και στην εικόνα baboon είναι 47.945.

6 *Επεξήγηση του κώδικα*

Όπως αναφέραμε σε προηγούμενη ενότητα για την δημιουργία του γραφικού περιβάλλοντος χρησιμοποιήσαμε το GUIDE. Τοποθετήσαμε τα components που χρειαζόμαστε και γράψαμε τον κατάλληλο κώδικα για τις αντίστοιχες λειτουργίες Έχουμε την δυνατότητα να τρέξουμε το πρόγραμμα μας είτε βήμα-βήμα είτε όλο μαζί με το πάτημα ενός κουμπιού. Ας δούμε αρχικά τον κώδικα που χρησιμοποιήσαμε βήμα-βήμα.




```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%loading image
function [src, l2 ,l3, srcHeight, srcWidth, res] = test_image(handles)
 [path1,user_cance]=imgetfile();
if user_cance
    msgbox(sprintf('Error'),'Error','Error');
    return
end
src=imread(path1);
src_Size=size(src);

srcHeight = src_Size(1);
srcWidth = src_Size(2);
lab = rgb2lab(image); % devide to chromatic channels %figure,
l=lab(:,:,1);
l2=im2double(l);
l3=l2/255;

F=makeLMfilters;
for k=1:48,
 res(:,:,k)=conv2(l2,F(:,:,k),'same');
end
prog=[0.5 0.5 1 1 1 1 1 1 1 1 1 1];
axes(handles.axes2);
imshow(src);
axes(handles.axes1);
imshow(src);
axes(handles.axes8);
imshow(prog);
```

```
drawnow
```

Δηλώνουμε σαν global μεταβλητές την εικόνα src,το κανάλι l2 και τις διαστάσεις της εικόνας(`srcHeight,srcWidth`) .Μετατρέπουμε τις τιμές του l σε double γιατί είναι unint8, ώστε να μπορέσουμε να εργαστούμε. Μέσα στη συνάρτηση βρίσκεται η ακολουθία των εντολών.Ορίσαμε το path το οποίο δίνει τη δυνατότητα στο χρήστη να επιλέξει από τον υπολογιστή μια οποιαδήποτε εικόνα .Σε περίπτωση που δεν επιλέξει εμφανίζεται το "Error".Δημιουργείται η συστοιχία φίλτρων και γίνεται συνέλιξη του l2 με κάθε ένα από τα 48 φίλτρα. Τέλος, η εικόνα εμφανίζεται στο βασικό πλαίσιο που έχουμε ορίσει και έπειτα στο πρώτο πλαίσιο του ιστορικού.


```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%edge detection and thinning
function [res3, BW,grad,Gradient_direction_theta ] =
test_edgedetection(handles, l2, res,srcHeight, srcWidth)

 [BW,grad]=edge2(l2,'canny');

BW2=repmat(BW,[1,1,48]);

for k=1:48,
res3(:,:,k)=res(:,:,k).*BW2(:,:,k);
end

Gradient_direction=rad2deg(grad);

Gradient_direction_theta=ones(srcHeight,srcWidth);

for i=1:srcHeight %Analoga me to gradient kanoume map tis times[-
180,180]stous analogous prosanatolismous
    for j=1:srcWidth
        if Gradient_direction(i,j)>0 && Gradient_direction(i,j)<45
            Gradient_direction_theta(i,j)=1;
        elseif Gradient_direction(i,j)>=45 &&
Gradient_direction(i,j)<90
            Gradient_direction_theta(i,j)=2;
```

```
        elseif Gradient_direction(i,j)>=90 &&
Gradient_direction(i,j)<135
            Gradient_direction_theta(i,j)=3;
        elseif Gradient_direction(i,j)>=135 &&
Gradient_direction(i,j)<=180
            Gradient_direction_theta(i,j)=4;
        elseif Gradient_direction(i,j)<0 && Gradient_direction(i,j)>=-
45
            Gradient_direction_theta(i,j)=5;
        elseif Gradient_direction(i,j)<-45 && Gradient_direction(i,j)>-
90
            Gradient_direction_theta(i,j)=6;
        elseif Gradient_direction(i,j)<=-90 &&
Gradient_direction(i,j)>-135
            Gradient_direction_theta(i,j)=7;
        elseif Gradient_direction(i,j)<=-135 &&
Gradient_direction(i,j)>=-180
            Gradient_direction_theta(i,j)=8;
        end
    end
end
prog=[0.5 0.5 0.5 0.5 1 1 1 1 1 1 1 1];

axes(handles.axes4);
imshow(BW);
axes(handles.axes1);
imshow(BW);
axes(handles.axes8);
imshow(prog);
drawnow
```

Δηλώνουμε σαν global μεταβλητές τα l2,res3, res, BW,grad, Gradient_direction_theta και srcHeight, srcWidth.Χρησιμοποιούμε τον ανιχνευτή ακμών Canny για το χρωματικό κανάλι l2. Αλλάζουμε τις διαστάσεις της εικόνας σε τρεις με την εντολή repmat και στη συνέχεια κάνουμε πολλαπλασιασμό σημείο προς σημείο για κάθε ένα φίλτρο με το αποτέλεσμα της προηγούμενης συνέλιξης. Μέσω της τροποποιημένης edge2 βρίσκουμε τον προσανατολισμό των ακμών και μετατρέπουμε τα ακτίνια σε μοίρες.Έπειτα δημιουργούμε ένα νέο πίνακα τον Gradient_direction_theta μέσα στον οποίον, ανάλογα με το Gradient_direction,κάνουμε map τις τιμές από [-180,180] στους κατάλληλους προσανατολισμούς.Έπειτα εμφανίζουμε την εικόνα στο βασικό πλαίσιο και στο δεύτερο πλαίσιο του ιστορικού.


```
function
[out_tg,texton_gradient]=test_textongradient(handles,res3,Gradient_dire
ction_theta,srcHeight,srcWidth)
%global test2 idx2 v row col v2
% out_tg texton_gradient res3 Gradient_direction_theta srcHeight
srcWidth

for k=1:48,
[row(:,k),col(:,k),v(:,k)]=find(res3(:,:,k));
end
idx2=kmeans(v,32,'MaxIter',1000);

v=size(idx2);
v2=v(1);
out_tg=zeros(srcHeight,srcWidth);

flag=1;

for x=1:v2(1)
    out_tg(row(x),col(x))=idx2(x,1);% depending on the edge use the
gradient orientation matrix to choose correctly the t
    %third dimension of the tgmo output.
end

texton_gradient=tgmo(out_tg,32,3,8,0);

for x=1:srcHeight
    for y=1:srcWidth
    out_tg(x,y)=texton_gradient(x,y,Gradient_direction_theta(x,y)); %
depending on the edge use the gradient orientation matrix to choose
correctly the t
    %third dimension of the tgmo output.

    end
end
prog=[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 1 1 1 1];
axes(handles.axes5);
imshow(out_tg);
axes(handles.axes1);
imshow(out_tg);
axes(handles.axes8);
imshow(prog);
```

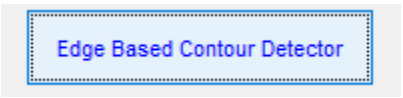
```
drawnow
```

Δηλώνουμε τις επιθυμητές μεταβλητές που χρειαζόμαστε σαν εξόδους,δηλαδή τα out_tg,texton_gradient, res3, Gradient_direction_theta, srcHeight και srcWidth. Χρησιμοποιούμε την εντολή find για να βρούμε τα μη μηδενικά σημεία για κάθε ένα από τα 48 φίλτρα μας καθώς και τους δείκτες τους.Έπειτα, κάνουμε χρήση της εντολής kmeans για να δημιουργήσουμε τα clusters που επιθυμούμε δίνοντας ορίσματα το v με το πλήθος των σημείων, τον αριθμό των clusters (32) και επίσης δίνουμε την παράμετρο MaxIter η οποία μας επιτρέπει να αυξήσουμε τον αριθμό των επαναλήψεων (η default είναι 100).Βρίσκουμε το μέγεθος του v και κάνουμε χρήση της συνάρτησης tgmo (textongradientmultipleorientations) για να βρούμε το textongradient. Τα ορίσματα που δίνουμε είναι το διάνυσμα που βρήκαμε προηγουμένως με την kmeans,ο αριθμός των textons (32), η ακτίνα σ (3), οι προσανατολισμοί (8)και επίσης δίνουμε την τιμή 0 στην παράμετρο tsim(Textonsimilaritymatrix). Δημιουργούμε έναν πίνακα με μηδενικά στις διαστάσεις της εικόνας μας και περνάμε τις τιμές που βρήκαμε στο textongradient πάνω στον πίνακα αξιοποιώντας την πληροφορία του Gradient_direction_theta σχετικά με τον προσανατολισμό της εκάστοτε ακμής ώστε να επιταχυνθεί η διαδικασία. Έπειτα, εμφανίζουμε τα αποτελέσματα στο βασικό πλαίσιο και στο τρίτο πλαίσιο του ιστορικού.


```
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%upologismos CG
function
[out_cg,color_gradient]=test_colorgradient(handles,Gradient_direction_t
heta,l3,BW,srcHeight,srcWidth)


color_gradient=cgmo(l3,32,1,8);
```

```
out_cg=zeros(srcHeight,srcWidth);

for i=1:srcHeight,
    for j=1:srcWidth,
       out_cg(i,j)=color_gradient(i,j,Gradient_direction_theta(i,j)); %
analoga tin akmh xrisimopoiw to gradient orientation matrix gia na
epileksw tin katallili eksodo  aptin triti diastasi tou tgmo output.
    end
end

out_cg=out_cg.*BW;
prog=[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 1 1];
axes(handles.axes6);
imshow(out_cg);
axes(handles.axes1);
imshow(out_cg);
axes(handles.axes8);
imshow(prog);
drawnow
```

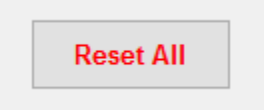
Δηλώνουμε τις μεταβλητές που χρειαζόμαστε και κάνουμε χρήση της συνάρτησης cgmo (colorgradientmultipleorientations),η οποία δέχεται σαν ορίσματα το l3, την ακτίνα $\sigma(1)$ και τους προσανατολισμούς (8).Δημιουργούμε ξανά έναν πίνακα με μηδενικά και περνάμε τις σωστές τιμές στις σωστές συντεταγμένες κάνοντας χρήση και του Gradient_direction_theta ώστε να πάρουμε τον σωστό προσανατολισμό. Τέλος,πολλαπλασιάζουμε σημείο προς σημείο με τον BW και εμφανίζουμε την εικόνα στο βασικό πλαίσιο και στο τέταρτο πλαίσιο του ιστορικού.




```
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%final step edgebased contour detector
function [Pb]=test_edgebasedcontour(handles,BW,out_tg,out_cg)

pres='gray';
[beta,fstd] =train_algo(pres);
```

```
Pb=pbCGTG(beta,fstd,BW,out_tg,out_cg);
Pb=(Pb-min(Pb(:)))/(max(Pb(:))-min(Pb(:)));
prog=[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5];
axes(handles.axes1);
imshow(Pb);
axes(handles.axes8);
imshow(prog);
```

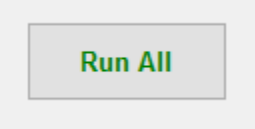
Δίνουμε σαν εισόδους τα BW, out_tg και out_cg. Αξιοποιώντας το datasetBSDS300 χρησιμοποιούμε έναν αλγόριθμο μέσω του οποίου κάνουμε μια δειγματοληψία από 200 εικόνες, οι οποίες δημιουργούν τα beta που αποτελούν τις σταθερές μας (*a* weights)και τα fstd(αποκλίσεις από την μέση τιμή). Αυτά, μαζί με τις εξόδους των textongradient και colorgradient, δίνονται ως ορίσματα στον τελικό *Pb* ανιχνευτή περιγραμμάτων μας. Εμφανίζουμε στο βασικό πλαίσιο το τελικό αποτέλεσμα του ανιχνευτή μας.




```
% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%resetAll
test_reset_All(handles);


function test_reset_All(handles)
global resetAll
resetAll=1;
axes(handles.axes1);
imshow(resetAll);
axes(handles.axes2);
imshow(resetAll);
axes(handles.axes4);
imshow(resetAll);
axes(handles.axes5);
imshow(resetAll);
axes(handles.axes6);
imshow(resetAll);
clearvars;
axes(handles.axes8);
imshow(resetAll);
```

```
clear global;
```

Δηλώνουμε τη μεταβλητή resetAll και τις δίνουμε την τιμή 1. Έτσι, «καθαρίζουμε» το βασικό πλαίσιο και τα πλαίσια του ιστορικού πατώντας το κουμπί ResetAll. Με τις clearvars και clearglobal«καθαρίζουμε» τη μνήμη ώστε να περαστούν οι νέες τιμές όταν φορτώσουμε μια νέα εικόνα.




```
% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%RunAll
global src l2 l3 srcHeight srcWidth res res3 BW grad
Gradient_direction_theta out_tg texton_gradient out_cg color_gradient
Pb
[src, l2, l3, srcHeight, srcWidth, res] = test_image(handles);
[res3, BW,grad,Gradient_direction_theta] = test_edgedetection(handles,
l2, res,srcHeight, srcWidth);
[out_tg,texton_gradient] =
test_textongradient(handles,res3,Gradient_direction_theta,srcHeight,
srcWidth );
[out_cg,color_gradient]=test_colorgradient(handles,l3,BW,Gradient_direc
tion_theta , srcHeight, srcWidth);
[Pb]=test_edgebasedcontour(handles,BW,out_tg,out_cg);
```

Δηλώνουμε όλες τις μεταβλητές που θα χρειαστούμε. Έπειτα καλούμε με τη σειρά τις συναρτήσεις .Ο κώδικας που χρησιμοποιούμε είναι ο ίδιος που χρησιμοποιήσαμε και στις εντολές βήμα-βήμα πέραν από μικρές διαφορές όπως την εντολή drawnow με την οποία εμφανίζεται το αποτέλεσμα κάθε βήματος στο ιστορικό μόλις διεκπεραιωθεί και όχι όλα μαζί στο τέλος.

Ακολουθεί ο κώδικας των συναρτήσεων που χρησιμοποιήσαμε για την μετατροπή της εικόνας στο χρωματικό χώρο του CIELab rgb2lab ,για την δημιουργία της συστοιχίας φίλτρων , του texton gradient ,του color gradient και του τελικού ανιχνευτή.

- Για την συνάρτηση rgb2lab

```
function [l,a,b]=rgb2labv4(src)
%load rgb image

%convert to lab
labTransformation = makecform('srgb2lab');
labI = applycform(src,labTransformation);

%seperate l,a,b
l = labI(:,:,1);
a = labI(:,:,2);
b = labI(:,:,3);
```

- Για την δημιουργία του filter bank

```
function F=makeLMfilters
% Returns the LML filter bank of size 49x49x48 in F. To convolve an
% image I with the filter bank you can either use the matlab function
% conv2, i.e. responses(:,:,i)=conv2(I,F(:,:,i),'valid'), or use the
% Fourier transform.

  SUP=49;                 % Support of the largest filter (must be odd)
  SCALEX=sqrt(2).^[1:3];  % Sigma_{x} for the oriented filters
  NORIENT=6;              % Number of orientations

  NROTINV=12;
  NBAR=length(SCALEX)*NORIENT;
  NEDGE=length(SCALEX)*NORIENT;
  NF=NBAR+NEDGE+NROTINV;
  F=zeros(SUP,SUP,NF);
  hsup=(SUP-1)/2;
  [x,y]=meshgrid([-hsup:hsup],[hsup:-1:-hsup]);
  orgpts=[x(:) y(:)]';

  count=1;
  for scale=1:length(SCALEX),
```

```
    for orient=0:NORIENT-1,
      angle=pi*orient/NORIENT;  % Not 2pi as filters have symmetry
      c=cos(angle);s=sin(angle);
      rotpts=[c -s;s c]*orgpts;
      F(:,:,count)=makefilter(SCALEX(scale),0,1,rotpts,SUP);
      F(:,:,count+NEDGE)=makefilter(SCALEX(scale),0,2,rotpts,SUP);
      count=count+1;
    end;
  end;

  count=NBAR+NEDGE+1;
  SCALES=sqrt(2).^[1:4];
  for i=1:length(SCALES),
    F(:,:,count)=normalise(fspecial('gaussian',SUP,SCALES(i)));
    F(:,:,count+1)=normalise(fspecial('log',SUP,SCALES(i)));
    F(:,:,count+2)=normalise(fspecial('log',SUP,3*SCALES(i)));
    count=count+3;
  end;
return

function f=makefilter(scale,phasex,phasey,pts,sup)
  gx=gauss1d(3*scale,0,pts(1,:),phasex);
  gy=gauss1d(scale,0,pts(2,:),phasey);
  f=normalise(reshape(gx.*gy,sup,sup));
return

function g=gauss1d(sigma,mean,x,ord)
% Function to compute gaussian derivatives of order 0 <= ord < 3
% evaluated at x.

  x=x-mean;num=x.*x;
  variance=sigma^2;
  denom=2*variance;
  g=exp(-num/denom)/(pi*denom)^0.5;
  switch ord,
    case 1, g=-g.*(x/variance);
    case 2, g=g.*((num-variance)/(variance^2));
  end;
return

function f=normalise(f), f=f-mean(f(:)); f=f/sum(abs(f(:))); return
```

- Για την δημιουργία του texton gradient

```
function [tg] = tgso(tmap,ntex,radius,theta,tsim)

% Compute the texture gradient at a single orientation and scale.
%
% INPUT
```

```
%   tmap        Texton map, values in [1,ntex].
%   ntex        Number of textons.
%   radius      Radius of disc for tg.
%   theta       Orientation orthogonal to tg.
%   'smooth'    Smoothing method, one of
%           {'gaussian','savgol','none'}, default 'none'.
%   'sigma'     Sigma for smoothing, default to radius.
%   'tsim'      Texton similarity matrix.  If not
%           provided, then use chi-squared.
%
% OUTPUT
%   tg      The tg image.

    smooth  = 'none';

    sigma   = radius;

    usechi2 = true;

    radius  = max(1,radius);

    wr      = floor(radius);

    [u,v] = meshgrid(-wr:wr,-wr:wr);

    gamma = atan2(v,u);

    mask = (u.^2 + v.^2 <= radius^2);

    mask(wr+1,wr+1) = 0; % mask out center pixel to remove bias

    count = sum(mask(:));
    side = 1 + (mod(gamma-theta,2*pi) < pi);

    side = side .* mask;

    lmask = (side==1)/count*2;

    rmask = (side==2)/count*2;

    flag=1;
    if tsim==0
        usechi2=true;
    else
        usechi2=false;
    end

    if usechi2 && flag==1
        tg = zeros(size(tmap));
        for i = 1:ntex,
            im = double(tmap==i);
            tgL = conv2(im,lmask,'same');
            tgR = conv2(im,rmask,'same');
            tg =tg+ sum((tgL-tgR).^2./(tgL+tgR+eps),3);
```

```
        end
    tg = 0.5 * tg;
    else
        fprintf('IN\n')
        [h,w] = size(tmap);
        d = zeros(h*w,ntex);
        for i = 1:ntex,
            im = double(tmap==i);
            tgL = conv2(im,lmask,'same');
            tgR = conv2(im,rmask,'same');
            d(:,i) = reshape(abs(tgL-tgR),h*w,1);
        end
        tg = sum((d*tsim).*d,2);
        tg = reshape(tg,h,w);
    end
end


function [tg] = tgmo(tmap,ntex,radius,norient,tsim)
% function [tg,theta] = tgmo(tmap,ntex,radius,norient,...)
%
% Compute the texture gradient at a single scale and multiple
% orientations.
%
% INPUT
%   tmap        Texton map, values in [1,ntex].
%   ntex        Number of textons.
%   radius      Radius of disc for texture gradient.
%   norient     Number of orientation at which to compute
%           the texture gradient.
%   'smooth'    Smoothing method, one of
%           {'gaussian','savgol','none'}, default 'none'.
%   'sigma'     Sigma for smoothing, default to radius.
%   'tsim'      Texton similarity matrix.  If not
%           provided, then use chi-squared.
%
% OUTPUT
%   tg      Size [h w norient] array of tg images.
%   theta       Vector of disc orientations (which are
%           orthogonal to the texture gradient).

% process options
    smooth  = 'none';

    sigma   = radius;

    usechi2 = true;

    radius  = max(1,radius);

    norient = max(1,norient);

    theta   = (0:norient-1)/norient*pi;

    [h,w] = size(tmap);
```

```
    tg = zeros(h,w,norient);
    fwrite(2,'[');
    for i = 1:norient,
        fwrite(2,'.');
        tg(:,:,i) = tgso(tmap,ntex,radius,theta(i),tsim);
    end
        fwrite(2,sprintf(']\n'));
end
```

- Για την δημιουργία του color gradient

```
function [cg,theta] = cgso(im,radius,theta,varargin)
% function [cg] = cgso(im,radius,theta,...)
%
% Compute the color gradient at a single scale and multiple
% orientations.
%
% INPUT
%   im      Grayscale or RGB image, values in [0,1].
%   radius      Radius of disc for cg.
%   theta       Orientation orthogonal to cg.
%   'nbins'     Number of bins; should be > 1/sigmaSim.
%   'sigmaSim'  For color similarity function.
%   'gamma'     Gamma correction for LAB [2.5].
%   'smooth'    Smoothing method, one of
%           {'gaussian','savgol','none'}, default 'none'.
%   'sigmaSmo'  Sigma for smoothing, default to radius.
%
% OUTPUT
%   cg      The cg image(s), same size as im.
%
% The input parameters {radius,nbins,sigmaSim,sigmaSmo} should be
% scalars when the input image is grayscale, and can be either scalars
% or 3-element vectors when the image is RGB.
%
% See also cgmo.
%
% David R. Martin <dmartin@eecs.berkeley.edu>
% April 2003

% process options
nbins = 32;
sigmaSim = 0.1;
gamma = 2.5;
smooth = 'none';
sigmaSmo = radius;
for i = 1:2:numel(varargin),
  opt = varargin{i};
  if ~ischar(opt), error('option names not a string'); end
  if i==numel(varargin), error(sprintf('option ''%s'' has no
value',opt)); end
```

```
  val = varargin{i+1};
  switch opt,
   case 'nbins', nbins=val;
   case 'sigmaSim', sigmaSim=val;
   case 'gamma', gamma=val;
   case 'smooth',
    switch val,
     case {'none','gaussian','savgol'}, smooth=val;
     otherwise, error(sprintf('invalid option smooth=''%s''',val));
    end
   case 'sigmaSmo', sigmaSmo=val;
   otherwise, error(sprintf('invalid option ''%s''',opt));
  end
end

% check arguments
if ndims(im)==2, % grayscale image
  if numel(radius)~=1, error('radius should have 1 element'); end
  if numel(nbins)~=1, error('nbins should have 1 element'); end
  if numel(sigmaSim)~=1, error('sigmaSim should have 1 element'); end
  if numel(sigmaSmo)~=1, error('sigmaSim should have 1 element'); end
elseif ndims(im)==3, % RGB image
  if numel(radius)==1, radius = radius*ones(3,1); end
  if numel(nbins)==1, nbins = nbins*ones(3,1); end
  if numel(sigmaSim)==1, sigmaSim = sigmaSim*ones(3,1); end
  if numel(sigmaSmo)==1, sigmaSmo = sigmaSmo*ones(3,1); end
  if numel(radius)~=3, error('radius should have 1 or 3 elements'); end
  if numel(nbins)~=3, error('nbins should have 1 or 3 elements'); end
  if numel(sigmaSim)~=3, error('sigmaSim should have 1 or 3 elements');
end
  if numel(sigmaSmo)~=3, error('sigmaSmo should have 1 or 3 elements');
end
  radius = radius(:);
  nbins = nbins(:);
  sigmaSim = sigmaSim(:);
  sigmaSmo = sigmaSmo(:);
else
  error('image not of valid dimension');
end
nbins = max(1,nbins);

% min and max values for a,b channels of LAB
% used to scale values into the unit interval
abmin = -73;
abmax = 95;

% make sure nbins is large enough with respect to sigmaSim
if any( nbins < 1./sigmaSim ),
  warning('nbins < 1/sigmaSim is suspect');
end

% check pixel valies
if min(im(:)) < 0 | max(im(:))>1,
  error('pixel values out of range [0,1]');
end
```

```
if ndims(im)==2, % grayscale image

  % compute cg from gray values
  cmap = max(1,ceil(im*nbins));
  csim = colorsim(nbins,sigmaSim);
  cg = tgso(...
      cmap,nbins,radius,theta,...
      'tsim',csim,'smooth',smooth,'sigma',sigmaSmo);

else, % RGB image

  % convert gamma-corrected image to LAB and scale values into [0,1]
  lab = RGB2Lab(im.^gamma);
  lab(:,:,1) = lab(:,:,1) ./ 100;
  lab(:,:,2) = (lab(:,:,2) - abmin) ./ (abmax-abmin);
  lab(:,:,3) = (lab(:,:,3) - abmin) ./ (abmax-abmin);
  lab(:,:,2) = max(0,min(1,lab(:,:,2)));
  lab(:,:,3) = max(0,min(1,lab(:,:,3)));

  % compute cg from LAB values
  cg = zeros(size(im));
  for i = 1:3,
    cmap = max(1,ceil(lab(:,:,i)*nbins(i)));
    csim = colorsim(nbins(i),sigmaSim(i));
    cg(:,:,i) = tgso(...
        cmap,nbins(i),radius(i),theta,...
        'tsim',csim,'smooth',smooth,'sigma',sigmaSmo(i));
  end

end

% compute color similarity matrix assuming colors are in [0,1]
function m = colorsim(nbins,sigma)
bc = ((1:nbins)-0.5)/nbins; % bin centers
[x,y] = meshgrid(bc,bc);
m = 1.0 - exp(-abs(x-y).^2./(2*sigma.^2));
```

```
function cg = cgmo(im,nbins,radius,norient)


%
% Compute the color gradient at a single scale and multiple
% orientations.
%
% INPUT
%   im      Grayscale or RGB image, values in [0,1].
%   radius      Radius of disc for cg.
%   norient     Number of orientations for cg.
```

```
%   'nbins'     Number of bins; should be > 1/sigmaSim.
%   'sigmaSim'  For color similarity function.
%   'gamma'     Gamma correction for LAB [2.5].
%   'smooth'    Smoothing method, one of
%           {'gaussian','savgol','none'}, default 'none'.
%   'sigmaSmo'  Sigma for smoothing, default to radius.
%
% OUTPUT
%   cg      Size [h,w,d,norient] array of cg images,
%           where d is the dimensionality of the image.
%
% The input parameters {radius,nbins,sigmaSim,sigmaSmo} should be
% scalars when the input image is grayscale, and can be either scalars
% or 3-element vectors when the image is RGB.
%

% process options
    sigmaSim = 0.1;
    gamma    = 2.5;
    smooth   = 'none';
    sigmaSmo = radius;

    norient = max(1,norient);

    nbins = max(1,nbins);

    % compute cg from gray values
    cmap = max(1,ceil(im*nbins));
    csim = colorsim(nbins,sigmaSim);
    cg = tgmo(cmap,nbins,radius,norient,csim);
end
```

- Για τον υπολογισμό του Edge Based Contour Detector

```
function pb = pbCGTG(beta,fstd,im,tg,cg)

%
% Compute probability of boundary using CG and TG.

%fstd
beta = beta' ./ fstd;
beta([1 2 3])


% compute oriented pb
  [h,w] = size(im);
  pb = zeros(h,w);
  l = cg(:,:);
  l = l(:);
  t = tg(:,:);
  t = t(:);
```

```
  x = [ones(size(l)) l t];
  pbi = 1 ./ (1 + (exp(-x*beta')));
  pb(:,:) = reshape(pbi,[h w]);



end
```

- Για την δημιουργία του ανιχνευτή

```
function [beta,fstd]=train_algo(pres)

% get features and labels

[f,y] = sampleDetector(@detector,pres);
f=f'; y=y';
% normalize features to unit variance
fstd = std(f);
fstd = fstd + (fstd==0);
f = f ./ repmat(fstd,size(f,1),1);
% fit the model
fprintf(2,'Fitting model...\n');
beta = logist2(y,f);
% save the result
save(sprintf('beta_cgtg_%s.txt',pres),'fstd','beta','-ascii');
end

function [f] = detector(im)
[cg,tg] = detCGTG(im);
l = max(squeeze(cg(:,:,1,:)),[],3);
t = max(tg,[],3);
l = l(:);
t = t(:);
f = [ ones(size(b)) l t ]';
end



function [filename] = segFilename(present,uid,iid)


filename =
fullfile(bsdsRoot,'human',present,sprintf('%d',uid),sprintf('%d.seg',ii
d));

end

function [bmap] = seg2bmap(seg,width,height)


if nargin<3,
  [height,width] = size(seg);
end
[h,w] = size(seg);
```

```
% check width and height
ar1 = width / height;
ar2 = w / h;
if width>w | height>h | abs(ar1-ar2)>0.01,
  error(sprintf('Can''t convert %dx%d seg to %dx%d
bmap.',w,h,width,height));
end

e = zeros(size(seg));
s = zeros(size(seg));
se = zeros(size(seg));

e(:,1:end-1) = seg(:,2:end);
s(1:end-1,:) = seg(2:end,:);
se(1:end-1,1:end-1) = seg(2:end,2:end);

b = (seg~=e | seg~=s | seg~=se);
b(end,:) = (seg(end,:)~=e(end,:));
b(:,end) = (seg(:,end)~=s(:,end));
b(end,end) = 0;

if w==width & h==height,

  bmap = b;

else

  bmap = zeros(height,width);
  for x = 1:w,
    for y = 1:h,
      if b(y,x),
        j = 1+floor((y-1)*height/h);
        i = 1+floor((x-1)*width/w);
        bmap(j,i) = 1;
      end
    end
  end

end
end


function [f,y] = sampleDetector(detector,pres,n,buffer)
% function [f,y] = sampleDetector(detector,pres,n,buffer)
%
% Sample on and off-boundary pixels fromthe BSDS training images,
% returning 0|1 class labels in y with the associated feature
% vectors in y.  The feature vectors are computed by the function
% provided by the argument detector.
%


if nargin<3, n=1000000; end
```

```
if nargin<4, buffer=2; end

% list of images
iids = imgList('train');

% number of samples per image
nPer = ceil(n/numel(iids));

y = zeros(1,0);
f = [];

for i = 1:numel(iids),
  tic;
  % read the image
  iid = iids(i);
 % fprintf(2,'Processing image %d/%d (iid=%d)...\n',i,numel(iids),iid);
  im = imgRead(iid);
  % run the detector to get feature vectors
  fprintf(2,'  Running detector...\n');
  features = feval(detector,im);
  % load the segmentations and union the boundary maps
  fprintf(2,'  Loading segmentations...\n');
  segs = readSegs(pres,iid);
  bmap = zeros(size(segs{1}));
  for j = 1:numel(segs),
    bmap = bmap | seg2bmap(segs{j});
  end
  dmap = bwdist(bmap);
  % sample
  fprintf(2,'  Sampling...\n');
  onidx = find(bmap)';
  offidx = find(dmap>buffer)';
  ind = [ onidx offidx ];
  cnt = numel(ind);
  idx = randperm(cnt);
  idx = idx(1:min(cnt,nPer));
  y = [ y bmap(ind(idx)) ];
  f = [ f features(:,ind(idx)) ];
  fprintf(2,'  %d samples.\n',numel(idx));
  toc;
end
end


function [segs,uids] = readSegs(present,iid)
% function [segs,uids] = readSegs(present,iid)
%
% Return a cell array of segmentations of an image
% and the associated UIDs.
%
% David Martin <dmartin@eecs.berkeley.edu>
% January 2003

files = dir(fullfile(bsdsRoot,'human',present,'*'));
segs = {};
```

```
uids = {};
n = 0;
for i = 1:length(files),
  if ~files(i).isdir, continue; end
  if strcmp(files(i).name,'.'), continue; end
  if strcmp(files(i).name,'..'), continue; end
  uid = sscanf(files(i).name,'%d',1);
  file = segFilename(present,uid,iid);
  if length(dir(file))==0, continue; end
  n = n + 1;
  segs{n} = readSeg(file);
  uids{n} = uid;
end
end
function [seg] = readSeg(filename)


fid = fopen(filename,'r');
if fid==-1,
  error(sprintf('Could not open file %s for reading.',filename));
end

% parse header
width = 0;
height = 0;
while 1,
  line = fgetl(fid);
  if ~ischar(line), error('Premature EOF.'); end
  if strcmp(line,'data'), break; end;

  [a,count] = sscanf(line,'width %d');
  if count==1, width=a; continue; end

  [a,count] = sscanf(line,'height %d');
  if count==1, height=a; continue; end
end

% read data
vals = fscanf(fid,'%d %d %d %d');
fclose(fid);

% parse data
seg = zeros(width,height);
vals = reshape(vals,4,length(vals)/4);
vals = vals + 1;
for i = 1:size(vals,2),
  s = vals(1,i);
  r = vals(2,i);
  c1 = vals(3,i);
  c2 = vals(4,i);
  seg(c1:c2,r) = s;
end
seg = seg';

% validate data
```

```
if min(seg(:)) < 1,
  error('Some pixel is not assigned a segment.');
end
if length(unique(seg(:))) ~= max(seg(:)),
  error('Some segment IDs are missing.');
end
end

function [impad] = padReflect(im,r)
% function [impad] = padReflect(im,r)
%
% Pad an image with a border of size r, and reflect the image into
% the border.


impad = zeros(size(im)+2*r);
impad(r+1:end-r,r+1:end-r) = im; % middle
impad(1:r,r+1:end-r) = flipud(im(1:r,:)); % top
impad(end-r+1:end,r+1:end-r) = flipud(im(end-r+1:end,:)); % bottom
impad(r+1:end-r,1:r) = fliplr(im(:,1:r)); % left
impad(r+1:end-r,end-r+1:end) = fliplr(im(:,end-r+1:end)); % right
impad(1:r,1:r) = flipud(fliplr(im(1:r,1:r))); % top-left
impad(1:r,end-r+1:end) = flipud(fliplr(im(1:r,end-r+1:end))); % top-
right
impad(end-r+1:end,1:r) = flipud(fliplr(im(end-r+1:end,1:r))); % bottom-
left
impad(end-r+1:end,end-r+1:end) = flipud(fliplr(im(end-r+1:end,end-
r+1:end))); % bottom-right
end

function [im] = nonmax(im,theta)


if numel(theta)==1,
  theta = theta .* ones(size(im));
end

% Do non-max suppression orthogonal to theta.
theta = mod(theta+pi/2,pi);

% The following diagram depicts the 8 cases for non-max suppression.
% Theta is valued in [0,pi), measured clockwise from the positive x
% axis.  The 'o' marks the pixel of interest, and the eight
% neighboring pixels are marked with '.'.  The orientation is divided
% into 8 45-degree blocks.  Within each block, we interpolate the
% image value between the two neighboring pixels.
%
%        .66.77.
%        5\  |  /8
%        5 \|/ 8
%        .--o--.-----> x-axis
%        4 /|\ 1
%        4/  |  \1
%        .33.22.
%           |
```

```
%           |
%           v
%         y-axis
%
% In the code below, d is always the distance from A, so the distance
% to B is (1-d).  A and B are the two neighboring pixels of interest
% in each of the 8 cases.  Note that the clockwise ordering of A and B
% changes from case to case in order to make it easier to compute d.

% Determine which pixels belong to which cases.
mask15 = ( theta>=0 & theta<pi/4 );
mask26 = ( theta>=pi/4 & theta<pi/2 );
mask37 = ( theta>=pi/2 & theta<pi*3/4 );
mask48 = ( theta>=pi*3/4 & theta<pi );

mask = ones(size(im));
[h,w] = size(im);
[ix,iy] = meshgrid(1:w,1:h);

% case 1
idx = find( mask15 & ix<w & iy<h);
idxA = idx + h;
idxB = idx + h + 1;
d = tan(theta(idx));
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% case 5
idx = find( mask15 & ix>1 & iy>1);
idxA = idx - h;
idxB = idx - h - 1;
d = tan(theta(idx));
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% case 2
idx = find( mask26 & ix<w & iy<h );
idxA = idx + 1;
idxB = idx + h + 1;
d = tan(pi/2-theta(idx));
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% case 6
idx = find( mask26 & ix>1 & iy>1 );
idxA = idx - 1;
idxB = idx - h - 1;
d = tan(pi/2-theta(idx));
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% case 3
idx = find( mask37 & ix>1 & iy<h );
idxA = idx + 1;
idxB = idx - h + 1;
```

```
d = tan(theta(idx)-pi/2);
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% case 7
idx = find( mask37 & ix<w & iy>1 );
idxA = idx - 1;
idxB = idx + h - 1;
d = tan(theta(idx)-pi/2);
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% case 4
idx = find( mask48 & ix>1 & iy<h );
idxA = idx - h;
idxB = idx - h + 1;
d = tan(pi-theta(idx));
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% case 8
idx = find( mask48 & ix<w & iy>1 );
idxA = idx + h;
idxB = idx + h - 1;
d = tan(pi-theta(idx));
imI = im(idxA).*(1-d) + im(idxB).*d;
mask(idx(find(im(idx)<imI))) = 0;

% apply mask
im = im .* mask;
end


function [beta,p,lli] = logist2(y,x,w)


error(nargchk(2,3,nargin));

% check inputs
if size(y,2) ~= 1,
  error('Input y not a column vector.');
end
if size(y,1) ~= size(x,1),
  error('Input x,y sizes mismatched.');
end

% get sizes
[N,k] = size(x);

% if sample weights weren't specified, set them to 1
if nargin < 3,
  w = 1;
end
```

```
% normalize sample weights so max is 1
w = w / max(w);

% initial guess for beta: all zeros
beta = zeros(k,1);

% Newton-Raphson via IRLS,
% taken from Hastie/Tibshirani/Friedman Section 4.4.
iter = 0;
lli = 0;
while 1==1,
  iter = iter + 1;

  % fitted probabilities
  p = 1 ./ (1 + exp(-x*beta));

  % log likelihood
  lli_prev = lli;
  lli = sum( w .* (y.*log(p+eps) + (1-y).*log(1-p+eps)) );

  % least-squares weights
  wt = w .* p .* (1-p);

  % derivatives of likelihood w.r.t. beta
  deriv = x'*(w.*(y-p));

  % Hessian of likelihood w.r.t. beta
  % hessian = x'Wx, where W=diag(w)
  % Do it this way to be memory efficient and fast.
  hess = zeros(k,k);
  for i = 1:k,
    wxi = wt .* x(:,i);
    for j = i:k,
      hij = wxi' * x(:,j);
      hess(i,j) = -hij;
      hess(j,i) = -hij;
    end
  end

  % make sure Hessian is well conditioned
  if (rcond(hess) < eps),
    error(['Stopped at iteration ' num2str(iter) ...
           ' because Hessian is poorly conditioned.']);
    break;
  end;

  % Newton-Raphson update step
  step = hess\deriv;
  beta = beta - step;

  % termination criterion based on derivatives
  tol = 1e-6;
  if abs(deriv'*step/k) < tol, break; end;

  % termination criterion based on log likelihood
```

```
%   tol = 1e-4;
%   if abs((lli-lli_prev)/(lli+lli_prev)) < 0.5*tol, break; end;
end;
end

function [im] = imgRead(iid,format)
% function [im] = imgRead(iid,format)

if nargin<2, format='color'; end

im = double(imread(imgFilename(iid))) / 255;

if strcmp(format,'gray'),
  im = rgb2gray(im);
end

end
function [iids] = imgList(type)


if nargin<1, type='all'; end

iids_train = load(fullfile(bsdsRoot,'iids_train.txt'));
%iids_test = load(fullfile(bsdsRoot,'iids_test.txt'));

switch type,
 %case 'all', iids = [ iids_train ; iids_test ];
 case 'train', iids = iids_train;
 %case 'test', iids = iids_test;
 otherwise, error(sprintf('type=%s is invalid',type));
end

% return a row vector
iids = iids(:)';
end

function [filename] = imgFilename(iid)


filename = fullfile(bsdsRoot,'images','train',sprintf('%d.jpg',iid));
if length(dir(filename))==1, return; end

error(sprintf('Could not find image %d in %s/images/{train,test}.', ...
              iid,bsdsRoot));
end

function [a,b,c] = fitparab(z,ra,rb,theta)
% function [a,b,c] = fitparab(z,ra,rb,theta)
%
% Fit cylindrical parabolas to elliptical patches of z at each
% pixel.
%
% INPUT
%   z   Values to fit.
```

```
%   ra,rb   Radius of elliptical neighborhood, ra=major axis.
%   theta   Orientation of fit (i.e. of minor axis).
%
% OUTPUT
%   a,b,c   Coefficients of fit: a + bx + cx^2


ra = max(1.5,ra);
rb = max(1.5,rb);
ira2 = 1 / ra^2;
irb2 = 1 / rb^2;
wr = floor(max(ra,rb));
sint = sin(theta);
cost = cos(theta);

% compute the interior quickly with convolutions
a = savgol(z,2,1,ra,rb,theta);
if nargout>1, b = savgol(z,2,2,ra,rb,theta); end
if nargout>2, c = savgol(z,2,3,ra,rb,theta); end

% re-compute the border, since the convolution screws it up
[h,w] = size(z);
for x = 1:w,
  for y = 1:h,
    if x>wr & x<=w-wr & y>wr & y<=h-wr, continue; end
    d0=0; d1=0; d2=0; d3=0; d4=0;
    v0=0; v1=0; v2=0;
    for u = -wr:wr,
      xi = x + u;
      if xi<1 | xi>w, continue; end
      for v = -wr:wr,
        yi = y + v;
        if yi<1 | yi>h, continue; end
        di = -u*sint + v*cost; % distance along major axis
        ei = u*cost + v*sint; % distance along minor axis (at theta)
        if di*di*ira2 + ei*ei*irb2 > 1, continue; end
        zi = z(yi,xi);
        di2 = di*di;
        d0 = d0 + 1;
        d1 = d1 + di;
        d2 = d2 + di2;
        d3 = d3 + di*di2;
        d4 = d4 + di2*di2;
        v0 = v0 + zi;
        v1 = v1 + zi*di;
        v2 = v2 + zi*di2;
      end
    end

    % much faster to do 3x3 matrix inverse manually
    detA = -d2*d2*d2 + 2*d1*d2*d3 - d0*d3*d3 - d1*d1*d4 + d0*d2*d4;
    invA = [-d3*d3+d2*d4  d2*d3-d1*d4 -d2*d2+d1*d3 ; ...
            d2*d3-d1*d4 -d2*d2+d0*d4  d1*d2-d0*d3 ; ...
            -d2*d2+d1*d3  d1*d2-d0*d3 -d1*d1+d0*d2 ] / (detA + eps);
    param = invA * [ v0 ; v1 ; v2 ];
```

```
    a(y,x) = param(1);
    if nargout>1, b(y,x) = param(2); end
    if nargout>2, c(y,x) = param(3); end
  end
end
end


function [fim] = fbRun(fb,im)
% function [fim] = fbRun(fb,im)
%
% Run a filterbank on an image with reflected boundary conditions.
%


% find the max filter size
maxsz = max(size(fb{1}));
for i = 1:numel(fb),
  maxsz = max(maxsz,max(size(fb{i})));
end

% pad the image
r = floor(maxsz/2);
impad = padReflect(im,r);

% run the filterbank on the padded image, and crop the result back
% to the original image size
fim = cell(size(fb));
for i = 1:numel(fb),
  if size(fb{i},1)<50,
    fim{i} = conv2(impad,fb{i},'same');
  else
    fim{i} = fftconv2(impad,fb{i});
  end
  fim{i} = fim{i}(r+1:end-r,r+1:end-r);
end
end

function z = distSqr(x,y)


if size(x,1)~=size(y,1),
  error('size(x,1)~=size(y,1)');
end

[d,n] = size(x);
[d,m] = size(y);

% z = repmat(sum(x.^2)',1,m) ...
%     + repmat(sum(y.^2),n,1) ...
%     - 2*x'*y;

z = x'*y;
x2 = sum(x.^2)';
```

```
y2 = sum(y.^2);
for i = 1:m,
  z(:,i) = x2 + y2(i) - 2*z(:,i);
end

end

function [cg,tg,theta] = detCGTG(im,radius,norient)


if nargin<2, radius=[0.01 0.02 0.02 0.02]; end
if nargin<3, norient=8; end

if numel(radius)==1, radius = radius*ones(4,1); end

[h,w,unused] = size(im);
idiag = norm([h w]);

% compute color gradient
[cg,theta] = cgmo_orig(im,idiag*radius(1:3),norient,...
                  'smooth','savgol','sigmaSmo',idiag*radius(1:3));

% compute texture gradient
no = 6;
ss = 1;
ns = 2;
sc = sqrt(2);
el = 2;
k = 64;
fname = sprintf( ...
    'unitex_%.2g_%.2g_%.2g_%.2g_%.2g_%d.mat',no,ss,ns,sc,el,k);
textonData = load(fname); % defines fb,tex,tsim
tmap = assignTextons(fbRun(textonData.fb,rgb2gray(im)),textonData.tex);
[tg,theta] = tgmo_orig(tmap,k,idiag*radius(4),norient,...
                  'smooth','savgol','sigma',idiag*radius(4));
end


function [map] = assignTextons(fim,textons)
% function [map] = assignTextons(fim,textons)

d = numel(fim);
n = numel(fim{1});
data = zeros(d,n);
for i = 1:d,
  data(i,:) = fim{i}(:)';
end

d2 = distSqr(data,textons);
[y,map] = min(d2,[],2);
[w,h] = size(fim{1});
map = reshape(map,w,h);
end
```

Βιβλιογραφία

[1] J. Malik, S. Belongie, T. Leung, and J. Shi, “Contour and texture analysis for image segmentation,” IJCV, vol.43, no. 1, pp. 7–27, 2001.
[2] P. Arbel´aez, M. Maire, C. Fowlkes, and J. Malik, “From contours to regions: An empirical evaluation,” in Conference on Computer Vision and Pattern Recognition. IEEE, 2009, pp. 2294–2301.
[3] Y. Cao, C. Wang, L. Zhang, and L. Zhang, “Edgel index for large-scale sketch-based image search,” in Conference on Computer Vision and Pattern Recognition. IEEE, 2011, pp. 761–768.
[4] M.Maire, S.X. Yu, and P. Perona, “Object detection and segmentation from joint embedding of parts and pixels,”in International Conference on Computer Vision. IEEE, 2011, pp. 2142–2149.
[5] S. Belongie, J. Malik, and J. Puzicha, “Shape matching and object recognition using shape contexts,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 4, pp. 509–522, 2002.
[6] D.R. Martin, C.C. Fowlkes, and J. Malik, “Learning to detect natural image boundaries using local brightness,color, and texture cues,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 5, pp.530–549, 2004.
[7] D.Martin, C. Fowlkes, D. Tal, and J.Malik, “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,” in International Conference on Computer Vision, 2001.
[8] P. Dollar, Z. Tu, and S. Belongie, “Supervised learning of edges and object boundaries,” in Conference on Computer Vision and Pattern Recognition. IEEE, 2006, vol. 2, pp. 1964–1971.
[9] P. Arbelaez, “Boundary extraction in natural images using ultrametric contour maps,” in Proceedings 5th IEEE Workshop on Perceptual Organization in Computer Vision (POCV’06), 2006.
[10] X. Ren, “Multi-scale improves boundary detection in natural images,” European Conference on Computer Vision, pp. 533–545, 2008.
[11] P. Felzenszwalb and D. McAllester, “A min-cover approach for finding salient curves,” in IEEE CVPR workshop. IEEE, 2006.
[12] M.Maire, P. Arbel´aez, C. Fowlkes, and J.Malik, “Using contours to detect and localize junctions in natural images,”in Conference on Computer Vision and Pattern Recognition, 2008.
[13] B. Catanzaro, B.Y. Su, N. Sundaram, Y. Lee, M. Murphy,and K. Keutzer, “Efficient, high-quality image contour detection,” in International Conference on Computer Vision. IEEE, 2009, pp. 2381–2388.
[14] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,”IEEE Transactions on Pattern Analysis and Machine Intelligence, , no. 99, pp. 1–1, 2011.
[15] J. Canny, “A computational approach to edge detection,”IEEE Transactions on Pattern Analysis and Machine Intelligence, , no. 6, pp. 679–698, 1986.
[16] J.E. Bresenham, “Algorithm for computer control of a digital plotter,” IBM Systems journal, vol. 4, no. 1, pp.25–30, 1965

[17] https://en.wikipedia.org/wiki/Nonlinear_eigenproblem

Για την μετατροπή rgb2lab
http://wildinformatics.blogspot.gr/2010/10/rgb-to-lab-color-transformation-in.html

Για το φίλτρο που χρησιμοποιήσαμε
http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html

Για τον υπολογισμό texton gradient και color gradient
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/code/lib/matlab/

Για τις εικόνες που χρησιμοποιήσαμε
http://sipi.usc.edu/database/database.php?volume=misc
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300/images/

Για την δημιουργία του Edge Based Contour Detector
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/code/