Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ψηφιακή Επεξεργασία Εικόνας

Εργασία 2 - Ανίχνευση ακμών και κύκλων

Λώλος Ιωάννης 10674

19 Μαΐου 2025

Περιεχόμενα

Κατάλογος σχημάτων

Σκοπός της εργασίας

Στόχο της εργασίας αποτελεί η υλοποίηση αλγορίθμων που στοχεύουν στην ανίχνευση ακμών και κύκλων δεδομένης κάποιας εικόνας. Για την υλοποίηση του ζητουμένου αναπτύχθηκαν και χρησιμοποιήθηκαν οι συναρτήσεις:

- `fir_conv`
- `sobel_edge`
- `log_edge`
- `circ_rough`

και με την βοήθεια του script `demo`, έγινε επίδειξη τους, ώστε να διαπιστωθεί η ορθότητα τους. Οι συναρτήσεις έγιναν optimized όσο το δυνατόν περισσότερο, κυρίως μέσω της βιβλιοθήκης *numpy*, ώστε να αποφευχθούν υπερβολικά μεγάλοι χρόνοι εκτέλεσης.

Υλοποίηση των συναρτήσεων

2.1 Η συνάρτηση `fir_conv`

Ο αλγόριθμος `fir_conv` έχει ως στόχο την εφαρμογή συνέλιξης ενός kernel h σε μία grayscale εικόνα. Για να καλυφθούν τα edge cases στα όρια της εικόνας, προστίθεται ένα padding από μάυρα pixels γύρω απο αυτή, ανάλογα με τις διαστάσεις του kernel. Στη συνέχεια, χρησιμοποιώντας strides (μέσω της numpy), η εικόνα χωρίζεται σε windows, ώστε να αποφευχθούν τα αργά for loops της python. Τέλος χρησιμοποιώντας της συνάρτηση einsum της numpy, υπολογίζεται το εσωτερικό γινόμενο κάθε παραθύρου με το flipped kernel, ουσιαστικά πραγματοποιώντας την συνέλιξη.

2.2 Η συνάρτηση `sobel_edge`

Η συνάρτηση `sobel_edge` είναι εξαιρετικά απλή στην υλοποίηση της. Ουσιαστικά δημιουργούνται δύο πίνακες-kernels, ένας για κάθε διεύθυνση. Τα kernels της οριζόντιας και κάθετης διεύθυνσης αντίστοιχα είναι τα εξής:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.2}$$

.

Στη συνέχεια υπολογίζεται η συνέλιξη της εικόνας με κάθε ένα από αυτά τα kernels και ως τελικό αποτέλεσμα λαμβάνεται ο πίνακας που περιέχει το μέτρο των συνελίξεων για κάθε στοιχείο.

$$g(n_1, n_2) = \sqrt{g_1^2(n_1, n_2) + g_2^2(n_1, n_2)} \tag{2.3}$$

Τέλος εφαρμόζεται μια μάσκα. Τα σημεία για τα οποία $g(n_1, n_2) >= threshold$, όπου το threshold είναι ένα δοσμένο όρισμα στη συνάρτηση επιστρέφονται ως 1, ενώ τα υπόλοιπα 0

2.3 Η συνάρτηση `log_edge`

Όλη η υλοποίηση της συνάρτησης `log_edge` βασίζεται στην κατασκευή του LoG kernel, ανάλογα με την δεδομένη διασπορά σ. Το kernel είναι ένας πίνακας διαστάσεων $2k + 1 \times 2k + 1$, όπου $k = 3 \cdot \sigma$, με τιμή

$$LoG(x_1, x_2) = -\frac{1}{\pi\sigma^4}(1 - \frac{x_1^2 + x_2^2}{2\sigma^2})e^{\frac{x_1^2+x_2^2}{2\sigma^2}} \tag{2.4}$$

Τα x_1, x_2 στην (διακριτή) περίπτωση της εργασίας αποτελούν διανύσματα με τιμή $\big[1..3k\big]$. Έπειτα, το kernel συνελίσσεται με την εικόνα και με τη χρήση της συνάρτησης `np.roll` υπολογίζεται αν υπάρχει μεταβολή προσήμου μεταξύ γειτονικών pixels, συμπεριλαμβανομένων των διαγωνίων γειτόνων. Ωστόσο, αυτή η προσέγγιση δημιούργησε κάποιες φορές τεχνητές ακμές στις άκρες τις εικόνας. Για αυτόν τον λόγο στο τέλος του αλγορίθμου μηδενίζονται τα pixels στα όρια.

2.4 Η συνάρτηση `circ_hough`

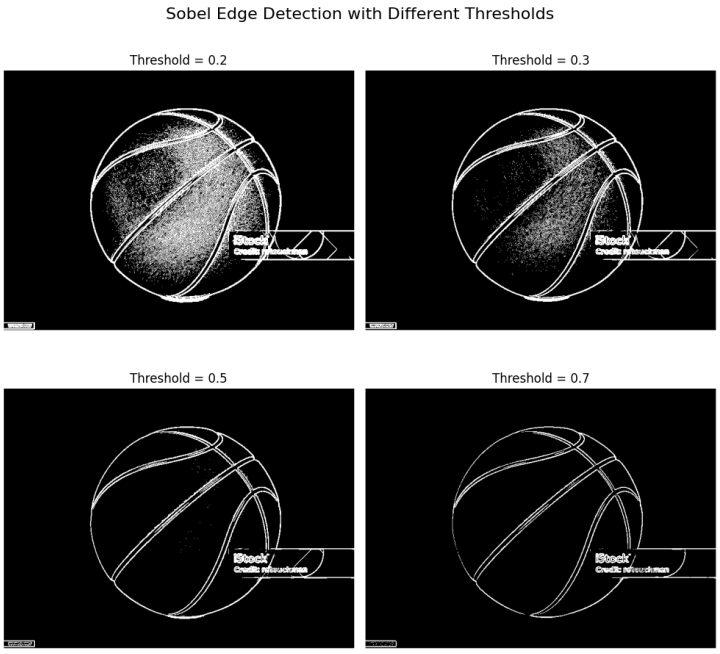
Για την εφαρμογή του αλγορίθμου Hough σε μία εικόνα χρειάζεται να δοθεί ένας πίνακας ψηφοφορίας, ο οποίος χωρίζει σε bins όλες τις δυνατές τιμές x, y, r που μπορεί να έχει ένας κύκλος κέντρου (x, y) και ακτίνας r στην εικόνα. Επίσης απαιτείται ο αριθμός των διακριτών γωνιών για τις οποίες ο αλγόριθμος θα αναζητήσει σημεία γύρω από το κέντρο. Για λόγους ταχύτητας, το ημίτονο και συνημίτονο των γωνιών αυτών υπολογίζεται μια φορά στην αρχή του αλγορίθμου και επαναχρησιμοποιούνται στη συνέχεια. Πρακτικά ο αλγόριθμος για κάθε ακτίνα στον πίνακα ψηφοφορίας ψάχνει bins περιμετρικά, στις γωνίες που αναφέρθηκαν νωρίτερα, τα οποία περιέχουν ακμές. Εφόσον κάποιο κατάλληλο bin βρεθεί προστίθεται στον πίνακα ψηφοφορίας. Στη συνέχεια ο πίνακας ψηφοφορίας χωρίζεται σε $3 \times 3 \times 3$ τμήματα και αναζητείται το τοπικό μέγιστο σε καθένα από αυτά. Τα μέγιστα που θα βρεθούν και είναι πάνω από ένα threshold V_{min}, αφού γίνουν scale πίσω σε κανονικές συντεταγμένες (και όχι bins) αποτελούν τα στοιχεία (κέντρο και ακτίνα) των κύκλων που αναγνωρίστηκαν. Τέλος, προστέθηκαν στον αλγόριθμο δύο επιπλέον features για καλύτερη αντοχή στον θόρυβο. Αρχικά προστέθηκε πέραν της μέγιστης ακτίνας, για την οποία αναζητούνται κύκλοι, και μία ελάχιστη ακτίνα. Επομένως αν βρεθούν κύκλοι ακτίνας

μικρότερης της $R_{R\text{min}}$ απορρίπτονται, καθώς θεωρούνται οτί προέρχονται από θόρυβο. Επίσης, κύκλοι με κοντινά στοιχεία γίνονται merge, ώστε να μην υπάρχουν παρόμοιοι κύκλοι σε πολύ μικρή απόσταση μεταξύ τους.

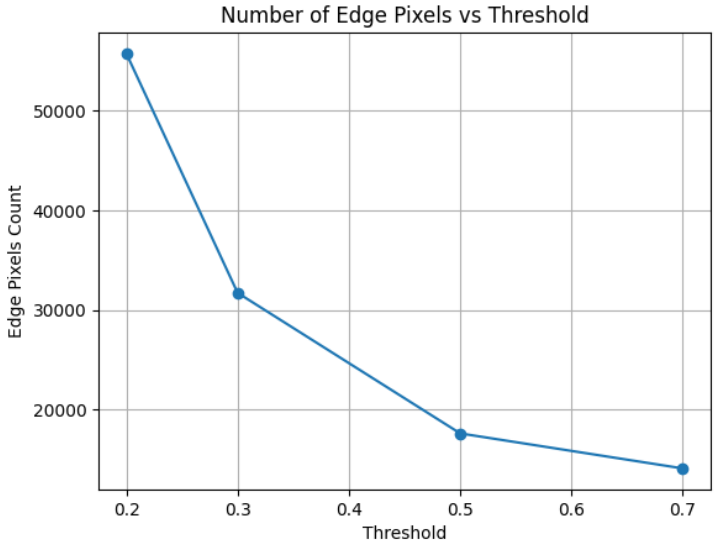
Αποτελέσματα

3.1 Ανίχνευση ακμών με Sobel

Για την δεδομένη εικόνα ο αλγόριθμος παρήγαγε αξιόπιστα αποτελέσματα για threshold μεγαλύτερο του 0.5. Σε χαμηλότερα επίπεδα η εικόνα που επιστρέφεται είναι υπερβολικά θορυβώδης για ουσιαστική επεξεργασία. Ο χρόνος που χρειάστηκε για την επεξεργασία ενός κατα 50% scaled down version της εικόνας(1070 x 1440 pixels) δεν ξεπέρασε σε καμία περίπτωση τα 0.035 δευτερόλεπτα.



Σχήμα 3.1: Οι αναγνωρισμένες ακμές από τον αλγόριθμο Sobel συναρτήσει του threshold



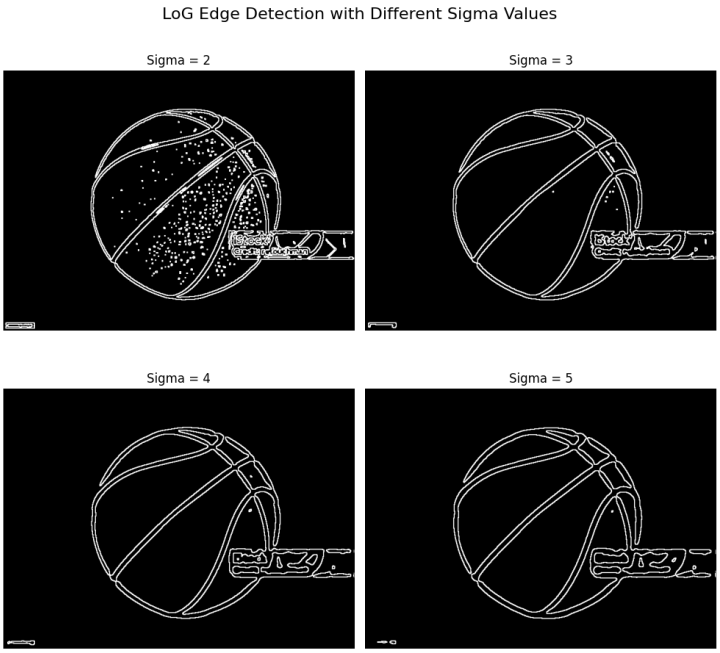
Σχήμα 3.2: Ο αριθμός των αναγνωρισμένων ακμών από τον αλγόριθμο Sobel συναρτήσει του threshold

3.2 Ανίχνευση ακμών με LoG

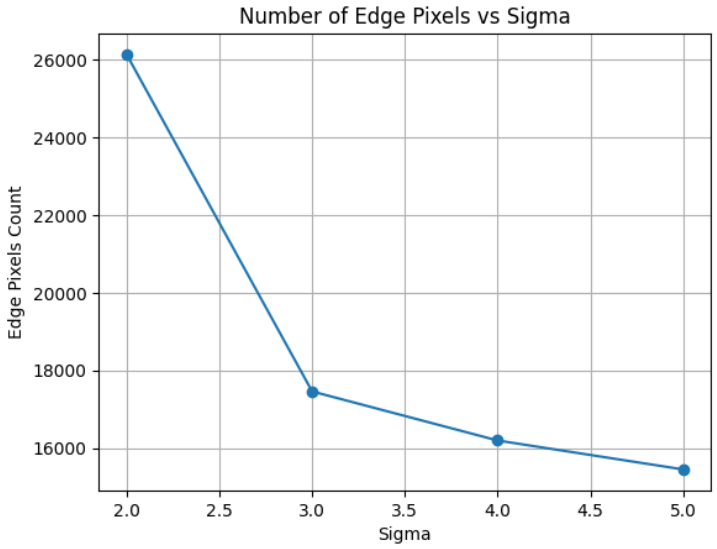
Αντίστοιχα ο Laplacian of Gaussian παρήγαγε αξιόπιστα αποτελέσματα για διακύμανση μεγαλύτερη από 3. Σε αντίθεση με τον Sobel, η αύξηση της διακύμανσης εδώ οδηγεί σε αύξηση του χρόνου επεξεργασίας. Ενδεικτικά οι χρόνοι παρουσιάζονται παρακάτω:

Variance (σ)	**Processing Time**
2	0.063
3	0.104
4	0.167
5	0.239

Πίνακας 3.1: Χρόνος επεξεργασίας του αλγορίθμου LoG συναρτήσει του Variance



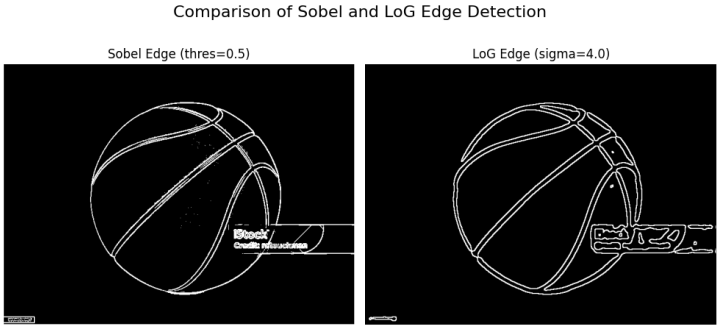
Σχήμα 3.3: Οι αναγνωρισμένες ακμές από τον αλγόριθμο LoG συναρτήσει της διακύμανσης(σ)



Σχήμα 3.4: Ο αριθμός των αναγνωρισμένων ακμών από τον αλγόριθμο LoG συναρτήσει της διακύμανσης(σ)

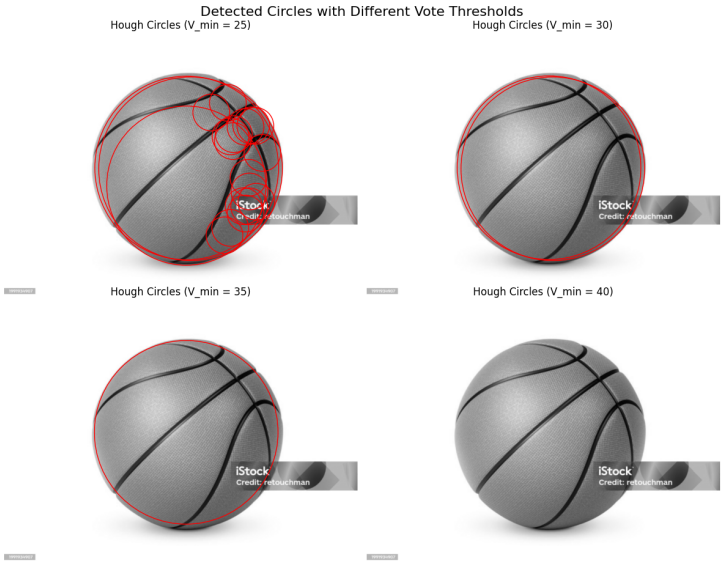
3.3 Σύγκριση Sobel - LoG

Από τις παραγόμενες εικόνες γίνεται αντιληπτό ότι η μέθοδος LoG παράγει ομαλότερες και πιο καθαρές γραμμές χωρίς θόρυβο, ωστόσο αυτό γίνεται εις βάρος της λεπτομέρειας, όπως γίνεται αντιληπτό από τα γράμματα μπροστά στην μπάλα. Η επιλογή της κατάλληλης μεθόδου απορρέει από τις ανάγκες ταχύτητας, ανθεκτικότητας σε θόρυβο και λεπτομέριας σε κάθε περίπτωση



Σχήμα 3.5: Σύγκριση των αποτελεσμάτων μεταξύ Sobel και LoG για παρόμοια ποιότητα ανίχνευσης ακμών

3.4 Αναγνώριση κύκλων με Hough



Σχήμα 3.6: Οι κύκλοι που αναγνωρίστηκαν από τον αλγόριθμο Hough συναρτήσει του V_{min}

Ο αλγόριθμος Hough κατάφερε να αναγνωρίσει κύκλους με μικτά αποτελέσματα αναλόγως του V_{min}. Για V_{min}: (25, 30, 35, 40) ο αλγόριθμος είχε σταθερή ταχύτητα γύρω στα 2.35 δευτερόλεπτα, ωστόσο για πολύ μικρές τιμές του ο χρόνος αυξήθηκε εκθετικά.

Συμπεράσματα

4.1 Σύγκριση μεταξύ Sobel - LoG

Για την χρήση τους στο πλαίσιο της εργασίας, δηλαδή ως preprocessing σε pipeline ανίχνευσης κύκλων, ο Laplacian of Gaussian (LoG) υπερείχε σημαντικά, καθώς παρήγαγε καθαρότερες και πιο καμπυλώτές γραμμές, οι οποίες βοήθησαν στην επεξεργασία, σε σχέση με τον Hough. Ο αλγόριθμος Sobel ενδεχομένως αποτελεί ελκυστικότερη επιλογή σε εφαρμογές με υψηλές απαιτήσεις σε ταχύτητα ή περιορισμένους υπολογιστικούς πόρους, καθώς υπερείχε σταθερά σε ταχύτητα συγκριτικά με τον LoG.

4.2 Εκτίμηση του αλγορίθμου Hough

Ο αλγόριθμος Hough, παρότι κατάφερε να παράξει ορθά αποτελέσματα, παρουσίασε ιδιαίτερη ευαισθησία στην μεταβολή των παραμέτρων του και ιδιαίτερα υψηλούς χρόνους εκτέλεσης, απαγορευτικούς παράγοντες για εφαρμογή σε real-time συστήματα με μεταβαλλόμενες συνθήκες περιβάλλοντος. Στην βιβλιογραφία προτείνεται το gradient-based Hough transform ως βελτίωση επί του αρχικού αλγορίθμου.