

Ψηφιακή Επεξεργασία Εικόνας

Εργασία 2:

Ανίχνευση ακμών και κύκλων

Ρεπάνης Γεώργιος-Δημήτριος

AEM:10588

Εισαγωγή

Η παρούσα εργασία πραγματεύεται την επεξεργασία μίας δοθείσας εικόνας με έμφαση στην ανίχνευση ακμών και στον εντοπισμό κύκλων. Στόχος είναι η κατανόηση και υλοποίηση αλγορίθμων όπως η συνέλιξη (convolution), τα φίλτρα Sobel και Log, καθώς και ο μετασχηματισμός Hough για κύλους.

Η εικόνα εισόδου που χρησιμοποιούμε για να εφαρμόσουμε τα παραπάνω είναι η `basketball_large.png` και πάνω σε αυτή εφαρμόζονται διαδοχικά φίλτρα και αλγόριθμοι ώστε να εξάγουμε : Ανίχνευση ακμών με Sobel για διαφορετικά κατώφλια (thresholds), Ανίχνευση ακμών με Laplacian of Gaussian (LoG), Εφαρμογή Hough Transform για εντοπισμό κύκλων σε δυαδικές εικόνες ακμών.

Κατά την διάρκεια του προγράμματος γίνεται και αξιολόγηση των αποτελεσμάτων με χρήση γραφημάτων και εικόνων καθώς και όπου ήταν αναγκαίο έγινε από τον δημιουργό βελτιστοποίηση των υπολογισμών για ταχύτερη εκτέλεση του προγράμματος και αποφυγή καταστάσεων με αρκετό θόρυβο (όπως πχ περιορισμός στους ισχυρότερους κύκλους μέσω φίλτρων ή λιγότερες επαναλήψεις ή μικρότερες τιμές σε πίνακες όπως στο `hough_space` με πίνακα καστομαρισμένο σε χαμηλότερες διαστάσεις για την αποφυγή υπερβολικής χρήσης μνήμης `dim[50, 50, 20]` από `[100, 100, 20]`). Όλα περιγράφονται αναλυτικά στον κώδικα με σχόλια κατάλληλα που επεξηγούν.

Όλες οι συναρτήσεις είναι γενικής μορφής και μπορούν να χρησιμοποιηθούν και σε άλλες εικόνες πέραν της δοθείσας από την εκφώνηση.

Συναρτήσεις και περιγραφές λειτουργίας:

`fir_conv.py`

Η συνάρτηση υλοποιεί φίλτρο συνέλιξης (convolution filter) σε μία gray scale εικόνα, χρησιμοποιώντας δοθείσα μάσκα από την εκφώνηση. Λαμβάνει δηλαδή ως είσοδο μια εικόνα grayscale και μια συνεκτική μάσκα `h`. Η συνέλιξη εφαρμόζεται με έτοιμη βιβλιοθήκη (`scipy.signal.convolve2d`) για σταθερή έξοδο ίδιο μεγέθους και zero-padding (το ίδιο μέγεθος με την είσοδο `mode='same'`). Οι παράμετροι `in_origin` και `mask_origin` είναι προαιρετική αλλά προστέθηκαν στην ανάλυση για λόγους πληρότητας και δεν χρησιμοποιούνται στην πράξη αλλά επιστρέφονται όπως ζητείται για συμβατότητα.

Η `fir_conv` είναι βασική συνάρτηση για τα φίλτρα που παράγουν τις εικόνες Sobel edge detection στο `sobel_edge.py` και χρησιμοποιεί φίλτρα sobel για ανίχνευση οριζόντιων και κάθετων ακμών. Επίσης στο LoG πάλι για εντοπισμό ακμών.

Sobel_edge.py

Η συνάρτηση `sobel_edge` υλοποιεί έναν βασικό αλγόριθμο ανίχνευσης ακμών μέσω Sobel φίλτρων. Τα φίλτρα Sobel υπολογίζουν κατά προσέγγιση τις χωρικές παραγώγους της έντασης της εικόνας, εντοπίζοντας έτσι σημεία με έντονες μεταβολές — δηλαδή τις ακμές.

Χρησιμοποιούνται δύο πυρήνες Sobel, ένας για τον οριζόντιο και ένας για τον κατακόρυφο προσανατολισμό.

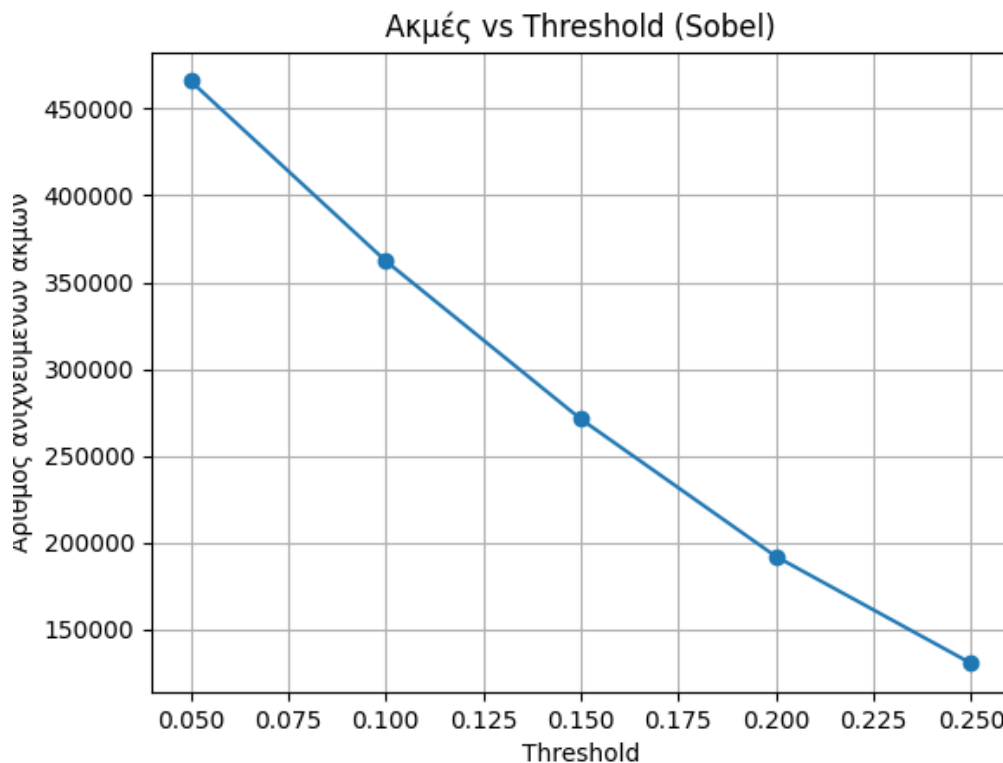
Το μέτρο του βαθμωτού μεγέθους του διανύσματος κλίσης υπολογίζεται ως:

$$magnitude = \sqrt{G_x^2 + G_y^2}$$

Στη συνέχεια εφαρμόζεται κατώφλι (*threshold*) ώστε να δημιουργηθεί η τελική δυαδική εικόνα ακμών (τιμές 0 και 1)

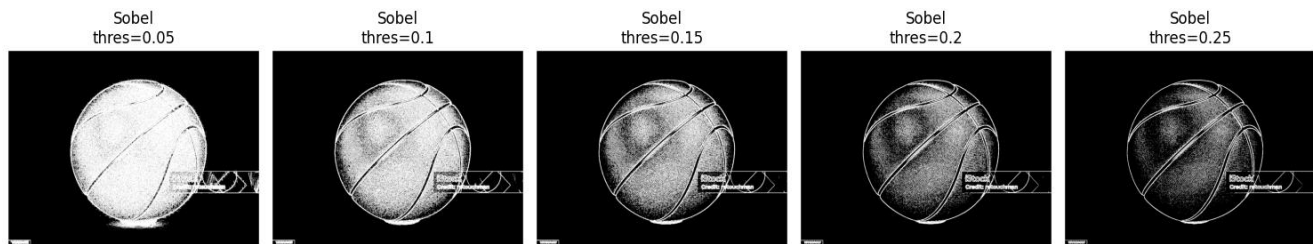
Στο `demo.py` καλείται η `sobel_edge` για 5 διαφορετικές τιμές *threshold* [0.05, 0.1, 0.15, 0.2, 0.25]. Οι εικόνες στην έξοδο δείχνουν τις ανιχνευμένες ακμές για κάθε κατώφλι και ένα γράφημα εμφανίζει τον αριθμό ακμών που ανιχνεύθηκαν συναρτήσει του *threshold*.

Sobel_threshold_plot.png



Το γράφημα `sobel_threshold_plot.png` δείχνει ξεκάθαρα ότι όσο αυξάνεται το `threshold` τόσο μειώνεται ο αριθμός των ανιχνευμένων ακμών.

Sobel_outputs.png



Η εικόνα `sobel_outputs.png` δείχνει τις εξόδους της `sobel_edge` για διάφορες τιμές κατωφλίου. Συγκεκριμένα για μικρό `thres` (π.χ. 0.05): Η εικόνα είναι πολύ φωτεινή, σχεδόν "θολή" από τις πολλές λευκές κουκκίδες — δηλαδή ανιχνεύονται πάρα πολλές ακμές και θόρυβος. Για μεγάλο `thres` (π.χ. 0.25): Η εικόνα είναι πιο καθαρή και σκοτεινή, με λίγες μόνο αλλά καθαρές ακμές, κυρίως στις περιοχές των γραμμών της μπάλας.

Αυτό οφείλεται στο γεγονός ότι μόνο τα σημεία με ισχυρή μεταβολή (ισχυρό *gradient*) διατηρούνται ως ακμές, ενώ οι ασθενείς και ο θόρυβος απορρίπτονται. Έτσι μειώνεται ο αριθμός των λευκών *pixels* και γίνεται καλύτερη απομόνωση των σημαντικών ορίων.

Log_edge.py

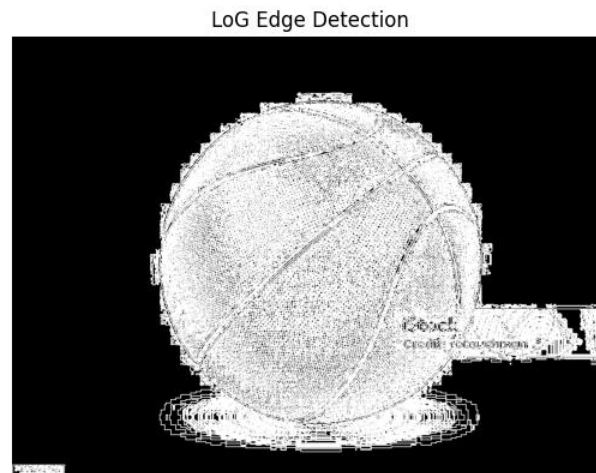
Η `log_edge()` εφαρμόζει τον τελεστή LoG (Laplacian of Gaussian) σε μία εικόνα εισόδου και εντοπίζει ακμές μέσω zero-crossings, δηλαδή θέσεις όπου η τιμή αλλάζει πρόσημο. Επιστρέφει μια δυαδική εικόνα με τιμές 0 και 1, όπου οι ακμές έχουν τιμή 1.

Είσοδος: γκρι εικόνα με τιμές $[0,1]$

Έξοδος: δυαδική εικόνα με 1 όπου εντοπίστηκε ακμή και 0 αλλιώς.

Περιγραφή κώδικα: Ορίζεται η μάσκα LoG μεγέθους 5x5 σύμφωνα με την εκφώνηση. Χρησιμοποιείται για να εφαρμόσουμε φιλτράρισμα τύπου Laplacian of Gaussian, που συνδυάζει εξομάλυνση (Gaussian) και δεύτερη παράγωγο (Laplacian). Στη συνέχεια γίνεται εφαρμογή συνέλιξης της εισόδου με τη LoG μάσκα, μέσω της ήδη υλοποιημένης `fir_convn()` και τέλος γίνεται έλεγχος κάθε εικονοστοιχείου (εκτός περιγράμματος) για αλλαγή πρόσημου στην 3x3 γειτονιά του. Αν εντοπιστεί τέτοια αλλαγή, σημαίνει πιθανή ακμή.

Μέσω του demo καλείται η συνάρτηση και δίνει:



Σχολιασμός:

Η εικόνα `log_output` απεικονίζει τα σημεία της εισόδου όπου εντοπίστηκαν ακμές μέσω της μεθόδου Laplacian of Gaussian (LoG). Οι λευκές περιοχές αντιστοιχούν σε θέσεις όπου η συνάρτηση εντόπισε ακμές (zero-crossings). Οι μαύρες περιοχές υποδεικνύουν περιοχές χωρίς ανίχνευση ακμών.

Circ_hough.py

Η συνάρτηση `circ_hough` έχει ως σκοπό την ανίχνευση κύκλων σε μια δυαδική εικόνα, χρησιμοποιώντας τον μετασχηματισμό Hough. Δέχεται ως είσοδο εικόνα από άκρες (π.χ. από Sobel ή LoG), καθώς και παραμέτρους σχετικά με τις επιτρεπτές ακτίνες και τα όρια ψηφοφορίας. Επιστρέφει τα κέντρα και τις ακτίνες των εντοπισμένων κύκλων, καθώς και τον 3D πίνακα Hough space.

Περιγραφή κώδικα: Για κάθε pixel άκρης (δηλ. pixel με τιμή 1), και για κάθε πιθανή ακτίνα, διαγράφεται ένας κύκλος με χρήση τριγωνομετρικών συναρτήσεων (\cos , \sin) σε 18 γωνίες (ανά 20°). Η μετατροπή των (x,y,r) σε δείκτες `a_idx`, `b_idx`, `r_idx` στον πίνακα `hough_space` γίνεται με αναλογική κλιμάκωση για σωστή αντιστοίχιση με τις διαστάσεις του.

Προστέθηκε όριο `max_points = 3000` ώστε να γίνεται δειγματοληψία από τα σημεία ακμών εφόσον είναι υπερβολικά πολλά για να περιοριστεί ο υπερβολικός χρόνος εκτέλεσης και η κατανάλωση μνήμης σε περιπτώσεις εικόνων με πολλές ακμές.

Βασικό script

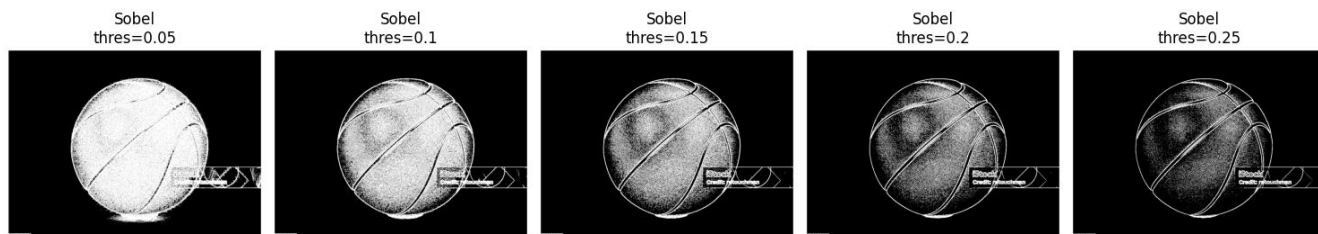
Demo.py

Το `demo.py` αποτελεί το βασικό σενάριο επίδειξης της λειτουργίας όλων των συναρτήσεων που αναπτύχθηκαν σε αυτή την εργασία. Πραγματοποιεί όλα τα βήματα της επεξεργασίας εικόνας. Όλες οι εικόνες και τα διαγράμματα εξόδου που περιλαμβάνονται στην αναφορά παράγονται από αυτό το script.

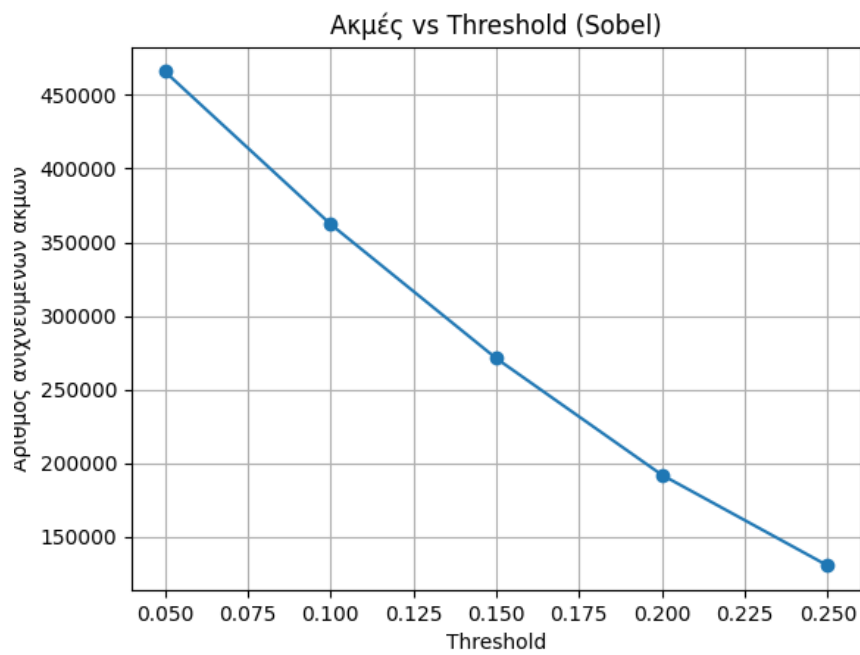
Ανάγνωση και προεπεξεργασία εικόνας: Η έγχρωμη εικόνα `basketball_large.png` μετατρέπεται σε `grayscale` (με τιμές $[0,1]$) για επεξεργασία.

Sobel Edge για διάφορα thresholds: Υπολογίζεται η εικόνα ακμών για 5 διαφορετικές τιμές κατωφλίου ($\text{thres} = 0.05$ έως 0.25). Επίσης παρουσιάζεται η αντίστοιχη εικόνα για κάθε threshold.

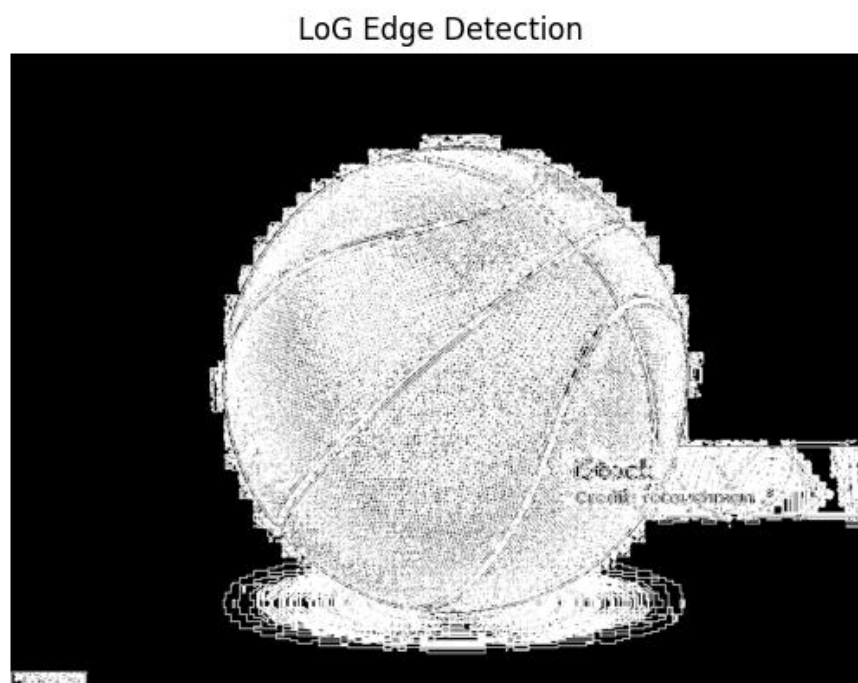
Η εικόνα που εξάγεται είναι:



Παράγεται γράφημα που δείχνει την επίδραση του *threshold* στον αριθμό ακμών.



LoG Edge Detection: Εκτελείται ανίχνευση ακμών με τον τελεστή Laplacian of Gaussian και παρουσιάζεται η έξοδος.



Ανίχνευση κύκλων με Hough Transform:

Χρησιμοποιείται ως είσοδος η δυαδική εικόνα ακμών από τον Sobel (π.χ. με threshold 0.1). Εξετάζονται 5 διαφορετικές τιμές του παραμέτρου v_{min} , η οποία καθορίζει το κατώφλι ψήφων για να θεωρηθεί έγκυρος ένας κύκλος. Προβάλλονται οι κύκλοι στην αρχική εικόνα για κάθε v_{min} .

Hough (Sobel) $v_{min}=2$



Hough (Sobel) $v_{min}=4$



Hough (Sobel) $v_{\min}=8$



Hough (Sobel) $v_{\min}=6$



Hough (Sobel) $v_{\min}=10$



Οδηγίες χρήσης του προγράμματος:

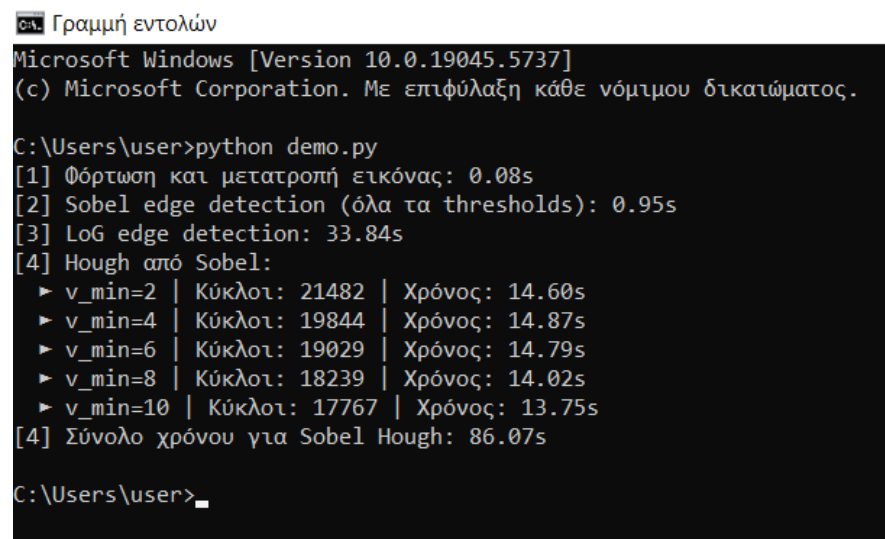
Το παραπάνω πρόγραμμα εκτελείται από την εντολή `python demo.py` στο τερματικό του υπολογιστή υπό την προϋπόθεση ότι όλα τα αρχεία δηλαδή οι συναρτήσεις και το script έχουν μπει στον ίδιο φάκελο. Έτσι δεν είναι αναγκαία η χρήση κάποιου VS για python.

Επίσης το πρόγραμμα καλεί συγκεκριμένες βιβλιοθήκες οι οποίες πρέπει να είναι προεγκατεστημένες. Αυτές είναι *NumPy*, *Matplotlib (Pyplot)*, *OpenCV (cv2)*, *Time* (προαιρετικό αλλά χρήσιμο).

Μετά την βασική εντολή στο τερματικό (`python demo.py`) ο χρήστης θα αναμένει μερικά δευτερόλεπτα μέχρι να του εμφανιστεί η πρώτη προσομοίωση. Για σκοπούς πληρότητας έχει προστεθεί στον κώδικα ο χρόνος αναμονής για την εκτέλεση κάθε λειτουργίας.

Μετά την παρουσίαση κάθε σταδίου το πρόγραμμα σταματά και ο χρήστης κλείνει την εκάστοτε προσομοίωση για να συνεχιστεί η λειτουργία του.

Ενδεικτικά το τερματικό μετά το πέρας του προγράμματος θα εμφανίζει κάτι σαν και αυτό:



```
Γραμμή εντολών
Microsoft Windows [Version 10.0.19045.5737]
(c) Microsoft Corporation. Με επιφύλαξη κάθε νόμιμου δικαιώματος.

C:\Users\user>python demo.py
[1] Φόρτωση και μετατροπή εικόνας: 0.08s
[2] Sobel edge detection (όλα τα thresholds): 0.95s
[3] LoG edge detection: 33.84s
[4] Hough από Sobel:
    ▶ v_min=2 | Κύκλοι: 21482 | Χρόνος: 14.60s
    ▶ v_min=4 | Κύκλοι: 19844 | Χρόνος: 14.87s
    ▶ v_min=6 | Κύκλοι: 19029 | Χρόνος: 14.79s
    ▶ v_min=8 | Κύκλοι: 18239 | Χρόνος: 14.02s
    ▶ v_min=10 | Κύκλοι: 17767 | Χρόνος: 13.75s
[4] Σύνολο χρόνου για Sobel Hough: 86.07s

C:\Users\user>_
```

Λειτουργικό Σύστημα: Windows 10 Pro 64-bit

Επεξεργαστής (CPU): Intel Core i5-10210U @ 1.60GHz, 4 cores, 8 threads

Μνήμη RAM: 8 GB

Python Έκδοση: Python 3.x (π.χ. 3.10)

IDE/Εκτέλεση: VSCode + cmd

Ανάλυση εικόνας: `basketball_large.png` (μεσαίας ανάλυσης)

Σχόλιο:

Οι χρόνοι εκτέλεσης μπορεί να διαφέρουν ανάλογα με τη CPU, τη μνήμη και τον αριθμό πυρήνων του συστήματος. Οι πιο κοστοβόρες συναρτήσεις (LoG και Hough) με

υπολογιστικά έντονη χρήση μνήμης. Ο αλγόριθμος είναι πλήρως λειτουργικός σε υπολογιστές με παρόμοιες ή καλύτερες προδιαγραφές.