ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

Ψηφιακή Επεξεργασία Εικόνας Αναφορά

Εργασία 2

Διακολουκάς Δημήτριος
ΑΕΜ 10642

Email: ddiakolou@ece.auth.gr

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή και περιγραφή εργασίας

Η παρούσα εργασία εντάσσεται στο πλαίσιο του μαθήματος **Ψηφιακή Επεξεργασία Εικόνας** και έχει ως κύριο αντικείμενο την ανίχνευση ακμών και κυκλικών περιγραμμάτων σε εικόνες, μέσω τεχνικών convolution και της μεθόδου Hough Transform.

Πιο συγκεκριμένα, αξιοποιούνται και υλοποιούνται τα παρακάτω βασικά εργαλεία:

- **FIR convolution**, που χρησιμοποιείται ως βασική λειτουργία για τη δημιουργία φίλτρων στην επεξεργασία εικόνας.
- **Ανίχνευση ακμών με Sobel**, για την εξαγωγή περιοχών έντονων μεταβολών φωτεινότητας με χρήση πρώτων παραγώγων.
- **Ανίχνευση ακμών με LoG (Laplacian of Gaussian)**, που εφαρμόζει φίλτρο Gaussian για εξομάλυνση και εντοπισμό ακμών μέσω της δεύτερης παραγώγου.
- **Ανίχνευση κύκλων με Hough Transform**, όπου ανιχνεύονται κυκλικά σχήματα σε δυαδικές εικόνες μέσω ψηφοφορίας σε έναν τρισδιάστατο πίνακα (κέντρο x, y, ακτίνα r).

Η εργασία αποτελείται από:

1. Συνάρτηση `fir_conv()` για συνέλιξη εικόνας με μάσκα.
2. Συνάρτηση `sobel_edge()` για ανίχνευση ακμών με Sobel.
3. Συνάρτηση `log_edge()` για εντοπισμό ακμών μέσω zero-crossings του φίλτρου LoG.
4. Συνάρτηση `circ_hough()` για εύρεση κύκλων σε εικόνες ακμών.
5. Το αρχείο `demo.py`, το οποίο συντονίζει την όλη ροή, εφαρμόζει τις παραπάνω συναρτήσεις στην εικόνα `basketball_large.png` η οποία παρουσιάζεται στο σχήμα παρακάτω και εξάγει τα τελικά αποτελέσματα.

Σχήμα 1.1: Εικόνα μπάλας μπάσκετ που δίνεται από τα δεδομένα.

Σκοπός της εργασίας είναι η υλοποίηση αυτών των τεχνικών από το μηδέν, χωρίς έτοιμες βιβλιοθήκες edge detection ή circle detection, παρά μόνο με χρήση βασικών πακέτων της Python όπως NumPy, SciPy, Matplotlib, PIL και OpenCV, όπου απαιτείται για την απεικόνιση.

Το τελικό αποτέλεσμα περιλαμβάνει τόσο τις υλοποιήσεις όσο και την απεικόνιση των ακμών και των κύκλων, μαζί με σχολιασμό για διαφορετικές παραμέτρους, όπως το κατώφλι ανίχνευσης, η ελάχιστη ψήφος για τον μετασχηματισμό Hough (`V_min`) και η ακτίνα κύκλων.

Κεφάλαιο 2

Συνέλιξη εικόνας με **FIR** φίλτρα

Στο πρώτο μέρος της εργασίας ζητείται η υλοποίηση της δισδιάστατης συνέλιξης εικόνας με FIR (Finite Impulse Response) φίλτρα. Τα FIR φίλτρα είναι μάσκες με πεπερασμένη υποστήριξη, δηλαδή μη μηδενικές τιμές εντός ενός πεπερασμένου ορθογωνίου χωρίου. Όταν εφαρμόζονται πάνω σε εικόνες, οδηγούν σε έξοδο μέσω της πράξης συνέλιξης.

Η έξοδος $g(n_1, n_2)$ από τη συνέλιξη μιας εικόνας εισόδου $f(n_1, n_2)$ με μάσκα $h(k_1, k_2)$ δίνεται από την εξίσωση:

$$g(n_1, n_2) = \sum_{k_1=N_{1,\min}}^{N_{1,\max}} \sum_{k_2=N_{2,\min}}^{N_{2,\max}} h(k_1, k_2) \cdot f(n_1 - k_1, n_2 - k_2)$$

Η πράξη αυτή πραγματοποιεί φιλτράρισμα στην εικόνα f, με τη μάσκα h να σκανάρει κάθε θέση της εικόνας και να υπολογίζει έναν νέο συνδυασμένο φωτεινότητας, βασισμένο στα γειτονικά πιξελς.

Στην παρούσα εργασία, η υλοποίηση πραγματοποιείται μέσω της συνάρτησης `fir_conv()`, η οποία λαμβάνει ως είσοδο:

- Την εικόνα (`in_img_array`) ως πίνακα NumPy 2 διαστάσεων.
- Τη μάσκα συνέλιξης (`h`), επίσης ως 2Dπίνακα.
- Προαιρετικά την αρχή (`origin`) της εικόνας και της μάσκας, ώστε να διατηρείται η σωστή αναφορά θέσης στο αποτέλεσμα.

Η συνάρτηση επιστρέφει:

- Τον πίνακα εξόδου (πλήρους συνέλιξης) με διαστάσεις `(M+P-1, N+Q-1)`.
- Τη θέση της αρχής $(0, 0)$ στον πίνακα εξόδου.

Ανάλυση υλοποίησης

Η υλοποίηση ακολουθεί πιστά τα μαθηματικά βήματα της συνέλιξης. Περιλαμβάνει αναστροφή της μάσκας, μηδενική επέκταση της εικόνας, και αθροιστικό πολλαπλασιασμό κατά στοιχείο μεταξύ μάσκας και τοπικής περιοχής της εικόνας.

Ανάλυση υλοποίησης `fir_conv()`

- Αρχικά γίνεται έλεγχος ότι τόσο η είσοδος όσο και η μάσκα είναι δισδιάστατοι πίνακες.
- Αν δεν δοθεί `in_origin`, θεωρείται ότι το $(0, 0)$ της εισόδου βρίσκεται στην πάνω αριστερή γωνία. Αντίστοιχα, αν δεν δοθεί `mask_origin`, ορίζεται στο κέντρο της μάσκας.
- Η μάσκα h αντιστρέφεται πλήρως και στους δύο άξονες (δηλαδή περιστρέφεται κατά 180 μοίρες), ώστε να ικανοποιεί τον μαθηματικό ορισμό της συνέλιξης.
- Η είσοδος f επεκτείνεται με μηδενικά γύρω από την εικόνα (zero-padding), ώστε να μπορεί να μετακινηθεί πλήρως η μάσκα χωρίς απώλειες στα όρια.
- Αρχικοποιείται πίνακας εξόδου διαστάσεων `(M + P - 1, N + Q - 1)` γεμάτος με μηδενικά.
- Για κάθε θέση (i, j) στην εικόνα εξόδου:
  - Επιλέγεται περιοχή από την επεκταμένη εικόνα, ίδιου μεγέθους με τη μάσκα.
  - Υπολογίζεται το γινόμενο κατά στοιχείο με τη μάσκα που έχει ήδη αντιστραφεί.
  - Αθροίζεται το αποτέλεσμα και αποθηκεύεται στη θέση (i, j).
- Τέλος, υπολογίζεται η νέα θέση του $(0, 0)$ στην εικόνα εξόδου ως `in_origin + mask_origin`.

Παρατηρήσεις

- Η υλοποίηση είναι πλήρους συνέλιξης (full convolution) και όχι έγκυρης (valid), δηλαδή επιστρέφει όλα τα πιθανά σημεία εφαρμογής της μάσκας, ακόμη και στα όρια της εικόνας.
- Η χρήση μηδενικού padding επηρεάζει τις ακραίες περιοχές, αλλά είναι απαραίτητη για τη σωστή ευθυγράμμιση της μάσκας σε όλο το φάσμα της εικόνας.
- Η προαιρετική χρήση μεταβλητών `in_origin` και `mask_origin` επιτρέπει διατήρηση της πληροφορίας θέσης, χρήσιμο σε πιο σύνθετες εφαρμογές (π.χ. Hough Transform).

Η συνάρτηση υλοποιείται με χρήση σχολίων στο αρχείο fir_conv.py και ονομάζεται fir_conv().

Κεφάλαιο 3

Ανίχνευση ακμών με τελεστή Sobel

Ο τελεστής Sobel είναι ένας κλασικός γραμμικός τελεστής για την ανίχνευση ακμών, ο οποίος προσεγγίζει τις παραγώγους πρώτης τάξης κατά τις δύο χωρικές κατευθύνσεις x_1 και x_2 μέσω συνέλιξης της εικόνας με δύο προκαθορισμένες μάσκες διαστάσεων 3×3.

Οι δύο μάσκες Sobel έχουν τη μορφή:

$$G_{x_1} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_{x_2} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Αυτές εφαρμόζονται στην εικόνα μέσω συνέλιξης για την προσέγγιση των παραγώγων κατά x_1 και x_2. Το τελικό μέτρο της κλίσης (ένταση της ακμής) υπολογίζεται ως:

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

όπου g_1 και g_2 είναι οι εικόνες που προκύπτουν από τη συνέλιξη με τις μάσκες G_{x_1} και G_{x_2} αντίστοιχα. Τα σημεία όπου η $g(n_1, n_2)$ είναι υψηλή αντιστοιχούν σε ακμές στην εικόνα.

Η υλοποίηση της μεθόδου Sobel πραγματοποιείται μέσω της συνάρτησης `sobel_edge()`, η οποία επιστρέφει ένα δυαδικό χάρτη ακμών για δεδομένο κατώφλι τιμής μεγέθους παραγώγου.

Ψευδοκώδικας υλοποίησης

Ψευδοκώδικας sobel_edge()

```
1. Gx ← array [-1 0 +1; -2 0 +2; -1 0 +1]
   Gy ← array [+1 +2 +1; 0 0 0; -1 -2 -1]

2. mask_origin ← (1,1)
   in_origin ← (0,0)

3. [conv_x, origin_x]←fir_conv(in_img_array,Gx,in_origin,mask_origin)
   From conv_x extract:
     start_r, start_c ← origin_x
     end_r ← start_r + input_rows
     end_c ← start_c + input_cols
     gx ← conv_x[start_r:end_r, start_c:end_c]

4. Similarly for Gy:
   [conv_y, origin_y] ← fir_conv(...)
   gy ← cropped output as above

5. grad_mag ← sqrt(gx^2 + gy^2)

6. out_img_array ← same shape as input
   out_img_array ← (grad_mag ≥ thres)

7. return out_img_array
```

Ανάλυση υλοποίησης

- Οι μάσκες `Gx` και `Gy` εφαρμόζονται με τη συνάρτηση `fir_conv()` που αναλύσαμε παραπάνω για απόλυτο έλεγχο στη θέση της αρχής.
- Το αποτέλεσμα της συνέλιξης είναι εικόνες ίδιων διαστάσεων με την είσοδο, καθώς γίνεται κατάλληλη αποκοπή (cropping) από την πλήρη έξοδο, με βάση τη θέση της αρχής (`origin`).
- Η ένταση ακμής προκύπτει ως $\sqrt{g_x^2 + g_y^2}$.
- Τέλος, εφαρμόζεται κατώφλι (`thres`) για να παραχθεί δυαδική εικόνα: 1 για έντονες ακμές, 0 αλλιώς.

Παρατήρηση

Ο τελεστής Sobel είναι ιδιαίτερα χρήσιμος καθώς ενσωματώνει φιλτράρισμα τύπου Gaussian smoothing (μέσω των τιμών [1,2,1]) με αριθμητική παράγωγο. Έτσι, μειώνεται ο θόρυβος και ενισχύονται οι πραγματικές ακμές.

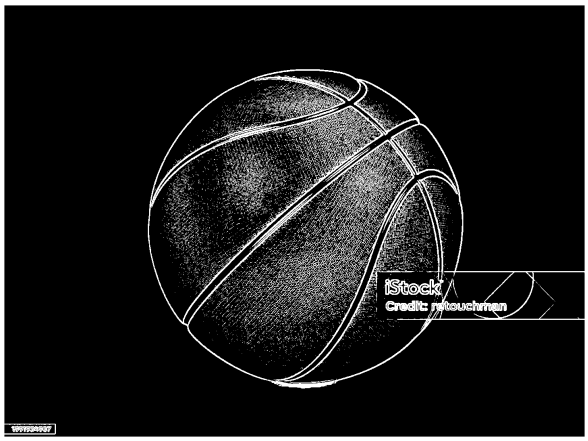
Η αντίστοιχη υλοποίηση με σχολιασμό βρίσκεται στο αρχείο sobel_edge.py και συγκεκριμένα η υλοποίηση στην συνάρτηση sobel_edge().

Αποτελέσματα και συμπεράσματα Sobel

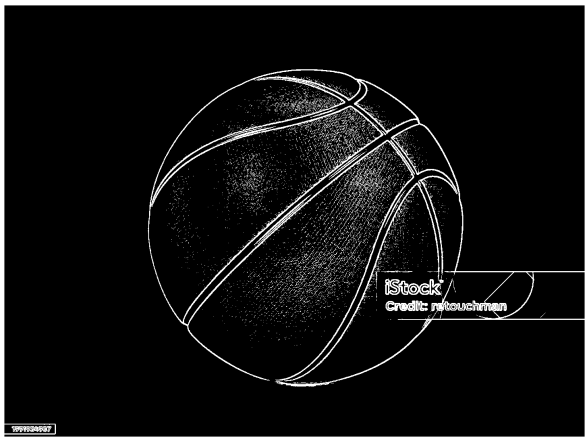
Στο αρχείο εκτέλεσης demo.py αφού γίνει η φόρτωση της εικόνας basketball_large.py και η μετατροπή της θεωρητικά σε float32 array σε [0,1] format (*np.ndarray*) ακολουθέι η κλήση της συνάρτησης sobel_edge(...) υπό καθορισμένες παραμέτρους κλήσεως threshold και βεβαίως *grayscale* εικόνας. Έπειτα τα αποτελέσματα ο κώδικας τα αποθηκεύει σε .png μορφή για οπτικοποίηση.

Output Sobel για διάφορα thresholds

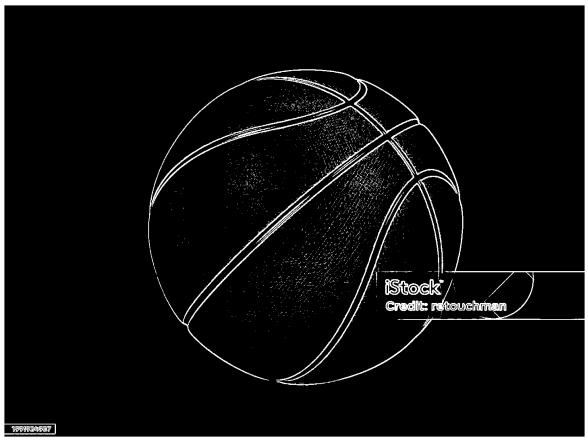
Παρακάτω παρατίθενται τα αντίστοιχα σχήματα που προέκυψαν από την κλήση της συνάρτησης sobel_edge().



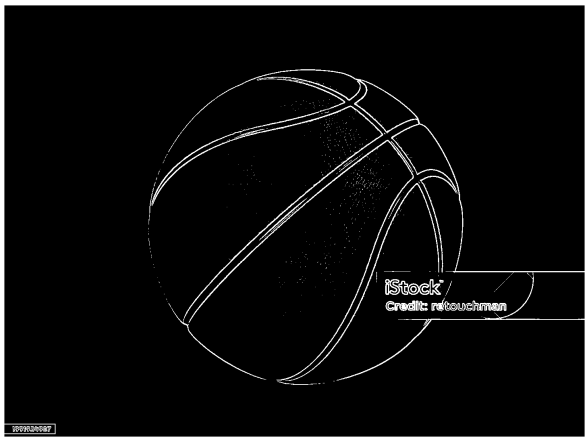
Σχήμα 3.1: Εικόνα μπάλας μπάσκετ με εφαρμογή sobel και threshold 0.25.



Σχήμα 3.2: Εικόνα μπάλας μπάσκετ με εφαρμογή sobel και threshold 0.33.



Σχήμα 3.3: Εικόνα μπάλας μπάσκετ με εφαρμογή sobel και threshold 0.4.



Σχήμα 3.4: Εικόνα μπάλας μπάσκετ με εφαρμογή sobel και threshold 0.5.

Συμπεράσματα για διαφορετικές τιμές `threshold` στον τελεστή Sobel

Για την αξιολόγηση της ευαισθησίας του τελεστή Sobel ως προς την παράμετρο κατωφλίου, εφαρμόστηκε η συνάρτηση `sobel_edge()` σε διαφορετικές τιμές του `threshold`. Η έξοδος είναι ένας δυαδικός χάρτης ακμών που εξαρτάται άμεσα από το πόσο έντονη πρέπει να είναι η κλίση ώστε να θεωρηθεί ακμή.

Παρατηρήσεις από τα αποτελέσματα:

- Για **sobel_thres = 0.25**, οι ακμές είναι πιο πυκνές και ανιχνεύονται ακόμα και περιοχές με μέτριες μεταβολές στη φωτεινότητα. Αυτό μπορεί να περιλαμβάνει και θόρυβο ή ανεπιθύμητες λεπτομέρειες.
- Για **sobel_thres = 0.33**, η εικόνα παρουσιάζει λιγότερες ακμές, καθώς απορρίπτονται πιο αδύναμες κλίσεις. Η περιγραφή του κυρίου αντικειμένου (η μπάλα) παραμένει ικανοποιητική.
- Με τιμές όπως **sobel_thres = 0.4** και **0.5**, η ανίχνευση επικεντρώνεται πλέον στις ισχυρότερες ακμές της εικόνας, κυρίως στο περίγραμμα της μπάλας και στις εσωτερικές καμπύλες. Ωστόσο, λεπτομέρειες μικρής έντασης παραλείπονται.

Η επιλογή της τιμής `threshold` επηρεάζει σημαντικά την πληρότητα και την ακρίβεια της ανίχνευσης. Χαμηλές τιμές οδηγούν σε υπερανίχνευση, ενώ υψηλές ενδέχεται να παραλείψουν χρήσιμες πληροφορίες. Η κατάλληλη ρύθμιση εξαρτάται από την εφαρμογή.

Κεφάλαιο 4

Ανίχνευση ακμών με τελεστή Laplacian of Gaussian (LoG)

Ο τελεστής Laplacian of Gaussian (LoG) χρησιμοποιείται για την ανίχνευση ακμών, εντοπίζοντας σημεία της εικόνας όπου η δεύτερη παράγωγος της έντασης αλλάζει πρόσημο — τα λεγόμενα zero crossings. Η διαδικασία περιλαμβάνει δύο στάδια:

- Εξομάλυνση της εικόνας με δισδιάστατο Gaussian πυρήνα (για απομάκρυνση θορύβου).
- Εφαρμογή της Laplacian στον εξομαλυμένο πυρήνα, ώστε να εντοπιστούν περιοχές υψηλής καμπυλότητας.

Μαθηματικό υπόβαθρο

Ο πυρήνας LoG προκύπτει από την εξίσωση:

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

Η εικόνα υφίσταται συνέλιξη με αυτό τον πυρήνα και οι ακμές ανιχνεύονται εντοπίζοντας αλλαγές πρόσημου (δηλαδή σημεία όπου η τιμή αλλάζει από θετική σε αρνητική ή το αντίστροφο).

Υλοποίηση `_generate_log_kernel()`

Η συνάρτηση δημιουργεί διακριτό LoG πυρήνα με βάση το σ και την εξίσωση που παρέθεσα πιο πάνω. Η περιοχή υπολογισμού είναι $[-3\sigma, 3\sigma]$, που καλύπτει την πλειοψηφία της Γκαουσιανής.

Ψευδοκώδικας _generate_log_kernel()

```
1. k ← ceil(3 * sigma)
2. x1, x2 ← [-k, ..., k]
3. X1, X2 ← meshgrid(x1, x2)

4. r_squared ← X1^2 + X2^2
5. factor ← -1 / (pi * sigma^4)
6. log_kernel ← factor * (1 - r_squared / (2*sigma^2)) *
                            exp(-r_squared / (2*sigma^2))

7. Return log_kernel and origin = (k, k)
```

Υλοποίηση _zero_crossings()

Η συνάρτηση ελέγχει σε κάθε σημείο (εκτός ορίων) αν η τοπική περιοχή 3×3 περιέχει τιμές τόσο θετικές όσο και αρνητικές — ένδειξη μηδενικής διέλευσης.

Ψευδοκώδικας _zero_crossings()

```
1. Initialize output as zero array (same shape as input)

2. For each pixel (r, c) excluding borders:
   - patch ← 3x3 region centered at (r, c)
   - If patch.min() < 0 and patch.max() > 0:
       → output[r, c] ← 1 (edge detected)

3. Return output
```

Υλοποίηση log_edge()

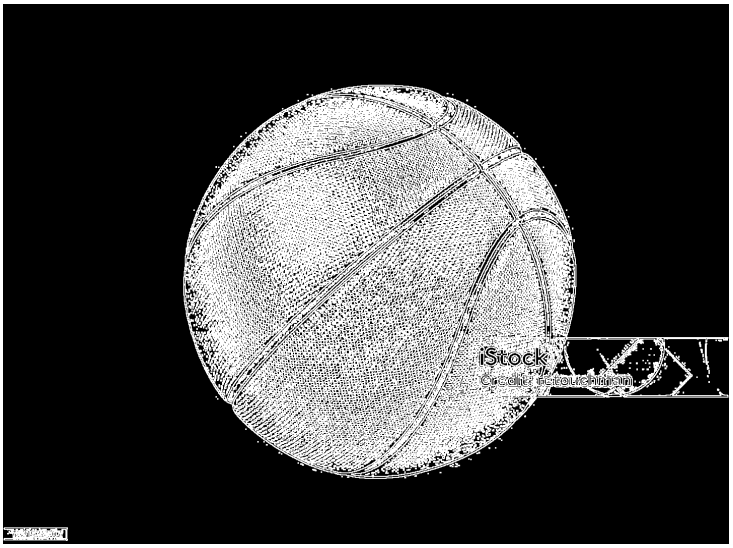
Αυτή είναι η κύρια συνάρτηση. Παράγει τον LoG πυρήνα, εφαρμόζει συνέλιξη μέσω `fir_conv()`, κάνει κατάλληλο `cropping` και εντοπίζει ακμές μέσω `zero-crossings`.

Ψευδοκώδικας log_edge()

```
1. h_log, mask_origin ← _generate_log_kernel(sigma)

2. conv_full, out_origin ← fir_conv(
       in_img_array, h_log,
       in_origin = (0,0),
       mask_origin = mask_origin
   )

3. Crop result from conv_full using out_origin:
   log_img ← conv_full[r0:r1, c0:c1]

4. result ← _zero_crossings(log_img)

5. Return result
```

Παραγόμενη εικόνα εξόδου LoG και ανάλυση αποτελεσμάτων

Με παρόμοιο τρόπο με την μέθοδο Sobel έτσι και στην μέθοδο LoG κλήθηκε η συνάρτηση log_edge(...) υπό καθορισμένες παραμέτρους κλήσεως ($s = 1$) και βεβαίως *grayscale* εικόνας. Έπειτα τα αποτελέσματα ο κώδικας τα αποθηκεύει σε .png μορφή για οπτικοποίηση.



Σχήμα 4.1: Ανίχνευση ακμών με τελεστή LoG για $\sigma = 1.0$. Οι ακμές εντοπίζονται στα σημεία μηδενικής διέλευσης.

Παρατηρήσεις και συμπεράσματα LoG και σύγκριση με Sobel

Έτσι, λοιπόν, παραθέτω παρακάτω και ορισμένες από τις διαπιστώσεις μου για την εφαρμογή της μεθόδου LoG συγκριτικά και με την μέθοδο Sobel .

- Η έξοδος του LoG είναι πιο συνεχής σε σχέση με τον Sobel, αναδεικνύοντας περιοχές με καμπυλότητες και διαφορές φωτεινότητας, ακόμη κι αν δεν αποτελούν έντονες ακμές.
- Το αποτέλεσμα περιλαμβάνει πολλές λεπτές γραμμές, καθώς εντοπίζονται μηδενικές διελεύσεις ακόμη και σε ήπιες μεταβολές — χαρακτηριστικό της δεύτερης παραγώγου.
- Παρότι η εξομάλυνση με $\sigma = 1.0$ μειώνει τον θόρυβο, εξακολουθούν να εμφανίζονται εσωτερικές υφές και δομές στην επιφάνεια της μπάλας.
- Σε σύγκριση με τον Sobel, το LoG αποφεύγει την ανάγκη ορισμού κατωφλίου, ωστόσο παράγει πιο πολύπλοκη έξοδο που περιλαμβάνει και ακμές «χαμηλής έντασης» (εντός του αντικειμένου).
- Η μέθοδος είναι χρήσιμη όταν επιθυμούμε πλήρη ανάδειξη δομών και όχι μόνο των πιο ισχυρών ακμών (όπως κάνει ο Sobel με κατάλληλο threshold).

Ο αντίστοιχος κώδικας για την υλοποίηση της μεθόδου LoG βρίσκεται στο αρχείο log_edge.py και η συνάρτηση κλήσεως είναι η log_edge().

Κεφάλαιο 5

Ανίχνευση κύκλων με μετασχηματισμό **Hough**

Ο μετασχηματισμός Hough αποτελεί μία κλασική παραμετρική μέθοδο για την ανίχνευση καμπυλών, όπως ευθείες, κύκλοι και έλλειψης. Στην παρούσα εργασία εστιάζουμε στην ανίχνευση κυκλικών περιγραμμάτων (π.χ. μπάλας) μέσω ενός τρισδιάστατου παραμετρικού χώρου (x, y, r), όπου x, y είναι τα κέντρα και r η ακτίνα κάθε υποψήφιου κύκλου.

Αρχή μεθόδου

Για κάθε στοιχείο ακμής (x_i, y_i) που προκύπτει από τεχνικές ανίχνευσης ακμών (Sobel ή LoG), η μέθοδος:

- Εξετάζει όλους τους δυνατούς κύκλους που θα μπορούσαν να περνούν από το σημείο αυτό.
- Υποθέτει διαφορετικές ακτίνες και γωνίες, και «ψηφίζει» για τα κέντρα αυτών των κύκλων στον χώρο παραμέτρων.
- Οι κύκλοι με υψηλό πλήθος ψήφων θεωρούνται ανιχνευμένοι.

Η ψήφιση γίνεται σε έναν **accumulator** και οι κύκλοι εξάγονται είτε μέσω ορίου ψήφων (V_min) είτε μέσω ταξινόμησης κατά ψήφους.

Βελτιστοποίηση μνήμης και απόδοσης

Η πλήρης σάρωση σε εικόνα υψηλής ανάλυσης μπορεί να οδηγήσει σε:

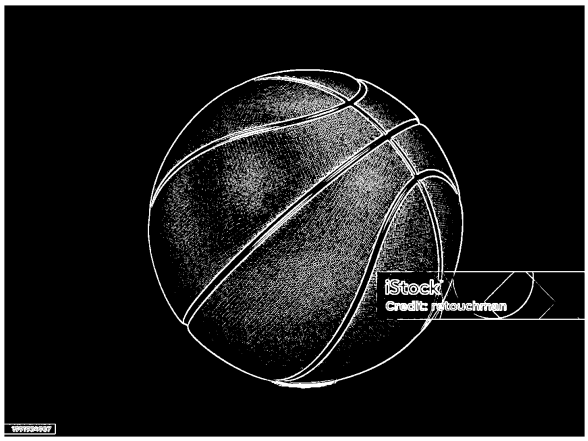
- **Υπερβολική κατανάλωση μνήμης**, λόγω του πλήθους των πιθανών συνδυασμών (x, y, r) (Message **killed** στο terminal).
- **Χρόνο εκτέλεσης** απαγορευτικό για διαδραστικές εφαρμογές και διαρκές *testing* για επιδίωξη βέλτιστων αποτελεσμάτων.

Για τον λόγο αυτό χρησιμοποιήθηκε η βοηθητική συνάρτηση `resize_custom_func()`, η οποία μειώνει το μέγεθος της εικόνας κατά 4 φορές, περιορίζοντας έτσι σημαντικά τις απαιτήσεις.

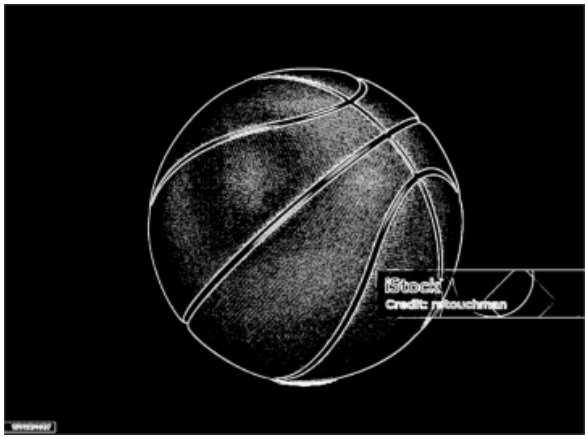
Ψευδοκώδικας resize_custom_func()

```
1. Load image from "name.png"
2. Compute new shape ← (H/4, W/4)
3. Apply resizing with anti_aliasing and preserve range
4. Save new image as "name_small.png"
```

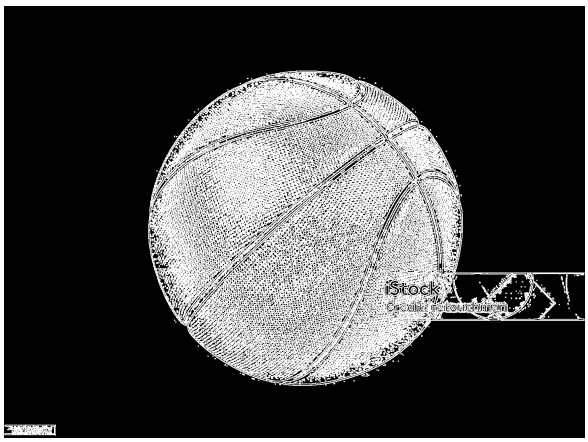
Παρακάτω παρατίθενται και οι εικόνες μειωμένου μεγέθους pixels πριν και μετά την χρήση της συνάρτησης για να δειχθεί ακριβώς τι εικόνα χρησιμοποιήθηκε για την συνάρτηση circ_hough() .



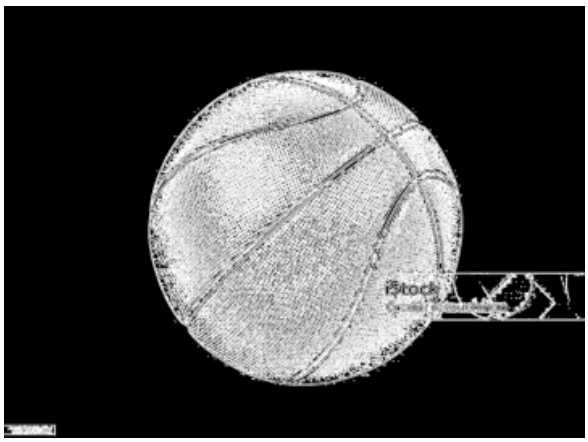
Σχήμα 5.1: Εικόνα μπάλας μπάσκετ με εφαρμογή sobel και threshold 0.25.



Σχήμα 5.2: Εικόνα μπάλας μπάσκετ με εφαρμογή sobel και threshold 0.25 μειωμένης διάστασης pixel.



Σχήμα 5.3: Εικόνα μπάλας μπάσκετ με εφαρμογή LoG .



Σχήμα 5.4: Εικόνα μπάλας μπάσκετ με εφαρμογή LoG μειωμένης διάστασης pixel.

Κύρια συνάρτηση circ_hough()

Η συνάρτηση `circ_hough()` ενσωματώνει την πλήρη διαδικασία εντοπισμού κύκλων:

Ψευδοκώδικας circ_hough()

```
1. Set parameters: R_min, delta_r, num_thetas
   bin_threshold ← V_min / num_thetas

2. Convert image to grayscale if needed
3. Apply Canny edge detector

4. Call find_hough_circles(...) to extract circles

5. Save result with circles drawn
6. Extract and return centers and radii of circles
```

Η έξοδος αποθηκεύεται είτε ως `circles_img_sobel.png` είτε `circles_img_log.png`, ανάλογα με τον χάρτη ακμών που χρησιμοποιήθηκε.

Ανίχνευση κύκλων: Συνάρτηση find_hough_circles()

Αυτή είναι η «καρδιά» της μεθόδου. Πραγματοποιεί την ψήφιση για κύκλους μέσω μιας βελτιστοποιημένης λίστας πιθανών παραμέτρων.

Ψευδοκώδικας find_hough_circles()

```
1. Prepare theta and radius ranges
2. For each edge pixel (x, y):
   - For each candidate (r, dx, dy):
       - Compute center (xc, yc) = (x - dx, y - dy)
       - Accumulate votes in (xc, yc, r)

3. Collect all entries with votes ≥ bin_threshold
4. Optionally sort by vote count and keep top_k

5. Post-process to eliminate near-duplicate circles
6. Draw circles on output image
7. Return image and list of circles
```

Ανάλυση εξόδου και σύγκριση

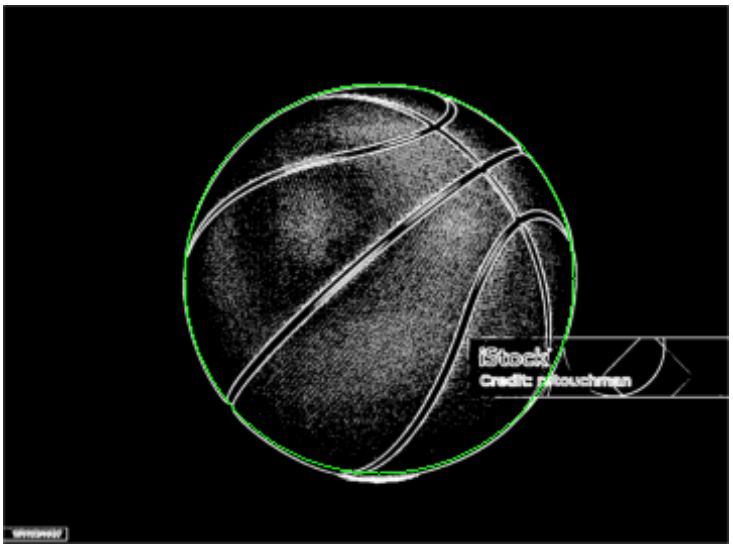
Η συνάρτηση `circ_hough()` εφαρμόστηκε σε δύο χάρτες ακμών:

- **Sobel edge map:** Απλός και γρήγορος, αλλά ευαίσθητος σε θόρυβο.
- **LoG edge map:** Παρέχει πιο σαφείς και ομαλές ακμές, βελτιώνοντας την ακρίβεια του εντοπισμού.

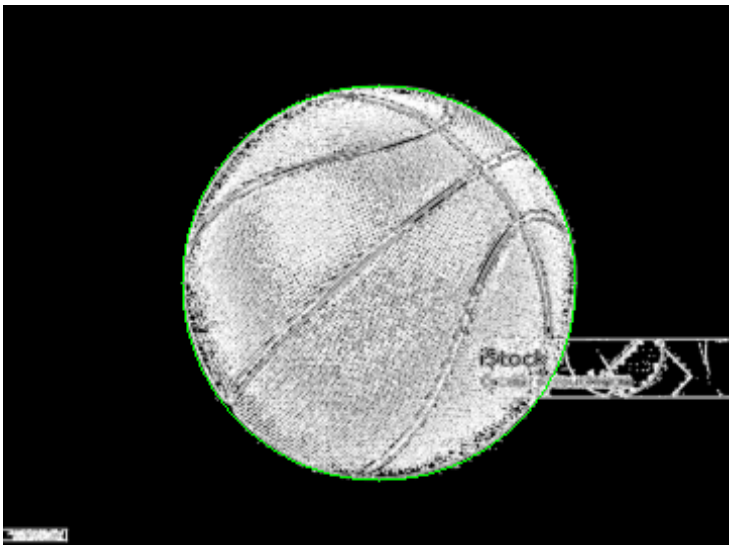
Αναλόγως με την παράμετρο `V_min`, ο αλγόριθμος ανίχνευσε είτε πολύ λίγους είτε πολλούς κύκλους. Η τελική επιλογή στο `V_min = 40` απέδωσε το περίγραμμα της μπάλας με ικανοποιητική ακρίβεια.

Εντοπισμένοι κύκλοι σε εικόνες log_small, sobel_small

Παρακάτω παρατίθενται τα αποτελέσματα που έγιναν demonstrate στο αρχείο demo.py υπό την κλήση της circ_hough() και αποθηκεύονται στον κώδικα σε μορφή .png.



Σχήμα 5.5: Ανίχνευση κύκλων με χρήση χάρτη ακμών Sobel . Ο εξωτερικός κύκλος εντοπίστηκε με επιτυχία.



Σχήμα 5.6: Ανίχνευση κύκλων με χρήση χάρτη ακμών LoG . Εντοπίζονται πολλαπλά επίπεδα περιγραμμάτων.

Συμπεράσματα και σχόλια

Σε αυτό το σημείο θεωρώ θεμιτώ να κάνω μία επίδειξη των παραμέτρων που υπέστη tuning προκειμένου να έχουμε τα βέλτιστα αποτελέσματα και για τις δύο περιπτώσεις όπως αναδείχθηκαν και στο παραπάνω section.

Συγκεκριμένα:

Πίνακας 5.1: Παράμετροι για τη συνάρτηση `circ_hough()`

Παράμετρος	**Τιμή**	**Περιγραφή**
`R_min`	95	Ελάχιστη ακτίνα κύκλου προς ανίχνευση
`R_max`	200	Μέγιστη ακτίνα κύκλου προς ανίχνευση
`delta_r`	1	Βήμα αύξησης της ακτίνας ανά κύκλο
`num_thetas`	100	Αριθμός διακριτών γωνιών ανά κύκλο (360° / 100 = 3.6° ανά βήμα)
`V_min`	40	Ελάχιστος αριθμός ψήφων (votes) που πρέπει να λάβει ένας κύκλος για να θεωρηθεί έγκυρος
`bin_threshold`	0.40	Κατώφλι ψήφων ανά αριθμό γωνιών: V_{min}/num_thetas
`min_edge_threshold`	100	Κατώτερο όριο για edge detection με `Canny`
`max_edge_threshold`	200	Ανώτερο όριο για `Canny edge detector`
`resize scale`	1/4	Κλίμακα σμίκρυνσης της εικόνας πριν την εφαρμογή του Hough για μείωση του κόστους
`top_k`	20	Μέγιστος αριθμός κύκλων που θα διατηρηθούν βάσει ψήφων

Τέλος, έχοντας και τα αποτελέσματα οπτικοποίησης εύκολα απορρέει το συμπέρασμα:

- Η εφαρμογή της `circ_hough()` στον Sobel edge map οδήγησε σε επιτυχή, αλλά περιορισμένη ανίχνευση κύκλων (λόγω και του `R_min` που έθεσα και του V_min) — εντοπίστηκε μόνο ο εξωτερικός κύκλος.

- Η εφαρμογή της ίδιας συνάρτησης στον LoG edge map παρείχε περισσότερους κύκλους, περιγράφοντας με μεγαλύτερη λεπτομέρεια τα εσωτερικά χαρακτηριστικά του αντικειμένου ωστόσο και πάλι με κατάλληλη επιλογή παραμέτρων καταφέραμε να αναδείξουμε τον μοναδικό κύκλο που περιβάλλει την μπάλα.

- Η χρήση LoG συνεπώς προσφέρει περισσότερες πληροφορίες, αλλά ενδέχεται να προσθέσει και παραπανίσιες ανιχνεύσεις, ανάλογα με τις παραμέτρους που θα οριστούν για μεγαλύτερη ελευθερία ανίχνευσης. Αντίθετα, ο Sobel είναι πιο στοχευμένος και φιλτραρισμένος — κατάλληλος για ανίχνευση κυρίως έντονων ορίων. Ωστόσο, με τις παραμέτρους που έθεσα, εντοπίζεται σωστά η σφαίρα ως μοναδικός κύκλος.

- Η ποιότητα της εξαγόμενης πληροφορίας εξαρτάται τόσο από την επιλογή του edge detector όσο και από την παράμετρο `V_min` που φιλτράρει τους υποψήφιους κύκλους.

Ο κώδικας βρίσκεται στο `circ_hough.py` και η υλοποίηση του κυκλικού μετασχηματισμού στο `find_hough_circles()`.

Βιβλιογραφία

[1] https://numpy.org/doc/

[2] https://docs.opencv.org/4.x/index.html

[3] https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html

[4] https://en.wikipedia.org/wiki/Circle_Hough_Transform