Ψηφιακή Επεξεργασία Εικόνας
-Εργασία 2: Hough, Harris, Rot και αποκοπή εικόνων

Α. Ντελόπουλος

Άνοιξη 2024

1 Εισαγωγικά

Στην πρώτη ενότητα της εργασίας θα υλοποιήσετε μια σουίτα από ρουτίνες (Ενότητες 2.1, 2.2 και 2.3) τις οποίες θα πρέπει να χρησιμοποιήσετε για να δώσετε λύση στο τελικό, real life, πρόβλημα (Ενότητα 3). Πιο συγκεκριμένα, θα υλοποιήσετε:

1. Τον μετασχηματισμό Hough
2. Τον Harris corner detector
3. Την περιστροφή εικόνας

2 Η Εργαλειοθήκη

2.1 Hough transform

Η εργαλειοθήκη ξεκινάει με την κατασκευή μιας ρουτίνας που υλοποιεί τον μετασχηματισμό **Hough**. Πιο συγκεκριμένα, κατασκευάστε την συνάρτηση:

```
H: np.ndarray, L: np.ndarray, res: int = my_hough_transform(
    img_binary: np.ndarray,
    d_rho: int,
    d_theta: float,
    n: int
)
```

η οποία λαμβάνει ως είσοδο μία binary εικόνα `img_binary` η οποία έχει προέλθει από κατωφλίωση της εξόδου ενός edge detector[1] για μία grayscale εικόνα, την διακριτότητα `d_rho` στην διάσταση ρ στο πεδίο του Hough μετρούμενη σε pixels, την διακριτότητα `d_theta` του θ στο πεδίο του Hough μετρούμενη σε rads, καθώς και το ζητούμενο πλήθος των `n` ισχυρότερων ευθειών προς εντοπισμό στην εικόνα.

Η συνάρτηση επιστρέφει τον πίνακα μετασχηματισμού Hough `H` και τον πίνακα `L` με τις παραμέτρους `rho` και `theta` των n ισχυρότερων ευθειών, των ευθειών δηλαδή που αντιστοιχούν στα n μεγαλύτερα τοπικά μέγιστα του πίνακα `H`. Η συνάρτηση επιστρέφει επίσης το πλήθος `res` των σημείων της εικόνας εισόδου που δεν ανήκουν στις n ευθείες που έχουν εντοπιστεί.

[1] Μπορείτε να χρησιμοποιήσετε έναν από έτοιμους edge detectors που υπάρχουν διαθέσιμοι σε βιβλιοθήκες της Python, όπως `scikit-image, opencv`. Να δικαιολογήσετε την επιλογή σας στο report.

ρ	θ
ρ_1	θ_1
$\vdots$	$\vdots$
ρ_n	θ_n

Πίνακας 1: Ο πίνακας `L` με διαστάσεις $n \times 2$

2.1.1 Παραδοτέα

Να κατασκευάσετε το Python script `deliverable_1.py` στο οποίο θα δείχνετε:

1. Τον πίνακα μετασχηματισμού `H` [2] και να τονίσετε πάνω στην εικόνα τις κορυφές (πίνακας `L`) που εντοπίσατε (μπορείτε να χρησιμοποιήσετε την εντολή `plt.scatter()` του `matplotlib`, επιλέγοντας ευδιάκριτο μέγεθος markers)

2. Χρωματίστε τις ευθείες που εντοπίσατε πάνω στην αρχική εικόνα εισόδου.

   Και για τα δύο παραδοτέα της ενότητας να χρησιμοποιήσετε την εικόνα `im2.jpg` που σας δίνεται μαζί με την εκφώνηση σαν εικόνα εισόδου.

2.2 Harris corner detector

Για τον εντοπισμό σημείων ενδιαφέροντος θα υλοποιήσετε τον αλγόριθμο **Harris** corner detector που πρωτοπροτάθηκε στο το άρθρο [1]. Ακολουθεί η βασική ιδέα. Ας υποθέσουμε ότι $w(x_1, x_2)$ είναι μία διδιάστατη συνάρτηση που έχει μή μηδενικές τιμές κοντά στην αρχή των αξόνων και "πεθαίνει" καθώς το $(x_1,\ x_2)$ απομακρύνεται από το $(0,\ 0)$. Για παράδειγμα:

$$w(x_1, x_2) \ = \ \exp\left(-\frac{x_1^2 + x_2^2}{2\sigma^2}\right) \tag{1}$$

Αν $I(x_1, x_2)$ είναι η φωτεινότητα μιας gray scale εικόνας, τότε η συνάρτηση

$$E(x_1, x_2; p_1, p_2) \ = \ \sum_{u_1, u_2} w(u_1, u_2) \| I(p_1 + u_1 + x_1, p_2 + u_2 + x_2) - I(p_1 + u_1, p_2 + u_2) \|^2 \tag{2}$$

παρουσιάζει την εξής συμπεριφορά:

- Αν το σημείο (p_1, p_2) βρίσκεται σε μία σχετικά ομαλή περιοχή της εικόνας τότε $E(x_1, x_2; p_1, p_2) \approx 0$.
- Αν το (p_1, p_2) βρίσκεται πάνω ή πολύ κοντά σε μία ακμή η οποία - σε σχέση με το μέγεθος της μη μηδενικής περιοχής του παραθύρου $w(x_1, x_2)$ - είναι ευθύγραμμη, τότε $E(x_1, x_2; p_1, p_2) \approx 0$ όταν η ολίσθηση $(x_1,\ x_2)$ είναι (σχεδόν) παράλληλη με την ακμή ενώ θα παίρνει τιμές $>> 0$ όταν η ολίσθηση $(x_1,\ x_2)$ είναι προς την κάθετη στην ακμή κατεύθυνση.
- Αν το (p_1, p_2) βρίσκεται πάνω ή πολύ κοντά σε μία γωνία τότε $E(x_1, x_2; p_1, p_2) >> 0$ για οποιαδήποτε κατεύθυνση της ολίσθησης $(x_1,\ x_2)$.

Συνεπώς η μελέτη της $E(x_1, x_2; p_1, p_2)$ ως συνάρτηση των $(x_1,\ x_2)$ μας επιτρέπει να αποφασίζουμε αν το σημείο $(p_1,\ p_2)$ βρίσκεται σε περιοχή ομαλής φωτεινότητας, σε ακμή ή σε γωνία. Χρησιμοποιώντας ανάπτυγμα Taylor γύρω από το σημείο $(p_1 + u_1, p_2 + u_2)$ (ως προς τις μεταβλητές (x_1, x_2), η $E(x_1, x_2; p_1, p_2)$ γράφεται

$$E(x_1, x_2; p_1, p_2) \approx \sum_{u_1, u_2} w(u_1, u_2) \| x_1 I_1(p_1 + u_1, p_2 + u_2) + x_2 I_2(p_1 + u_1, p_2 + u_2) \|^2 \tag{3}$$

όπου $I_1(x_1, x_2) = \partial I(x_1, x_2)/\partial x_1$, $I_2(x_1, x_2) = \partial I(x_1, x_2)/\partial x_2$ και έχουμε παραλήψει τους όρους υψηλότερης τάξης. Εναλλακτικά,

$$E(x_1, x_2; p_1, p_2) \approx [x_1,\ x_2]\, \mathbf{M}(p_1, p_2) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4}$$

[2] Μπορείτε να χρησιμοποιήστε το `matplotlib` και την εντολή **plt.imshow(H, cmap="gray")**

όπου

$$\mathbf{M}(p_1, p_2) = \sum_{u_1, u_2} w(u_1, u_2)\mathbf{A}(u_1, u_2; p_1, p_2) \tag{5}$$

και

$$\mathbf{A}(u_1, u_2; p_1, p_2) = \begin{bmatrix} I_1(p_1 + u_1, p_2 + u_2)^2 & I_1(p_1 + u_1, p_2 + u_2)I_2(p_1 + u_1, p_2 + u_2) \\ I_1(p_1 + u_1, p_2 + u_2)I_2(p_1 + u_1, p_2 + u_2) & I_2(p_1 + u_1, p_2 + u_2)^2 \end{bmatrix} \tag{6}$$

Πρακτικά οι μερικές παράγωγοι που συμμετέχουν στις παραπάνω εκφράσεις υπολογίζονται με τη χρήση κατάλληλων συνελικτικών μασκών όπως έχουμε εξηγήσει στη θεωρία. Ο υπολογισμός τους μάλιστα γίνεται μια κι έξω για όλη την εικόνα. Οι τρείς εναλλακτικές συμπεριφορές της συνάρτησης $E(x_1, x_2; p_1, p_2)$ που είδαμε παραπάνω αντιστοιχούν σε τρεις αντίστοιχες διαφορετικές εκδοχές για τις ιδιοτιμές του πίνακα $\mathbf{M}(p_1, p_2)$:

- Αν το σημείο (p_1, p_2) βρίσκεται σε μία σχετικά ομαλή περιοχή της εικόνας τότε $\lambda_1 \approx 0$ και $\lambda_1 \approx 0$.
- Αν το (p_1, p_2) βρίσκεται πάνω ή πολύ κοντά σε μία ακμή τότε $\lambda_1 >> \lambda_2 \approx 0$.
- Αν το (p_1, p_2) βρίσκεται πάνω ή πολύ κοντά σε μία γωνία τότε $\lambda_1 \approx \lambda_2 >> 0$.

Συνεπώς η απόφαση για το αν η περιοχή του σημείου (p_1, p_2) είναι ομαλή, βρίσκεται σε ακμή ή σε γωνία μπορεί να ληφθεί από τη μελέτη των δύο ιδιοτιμών του πίνακα $\mathbf{M}(p_1, p_2)$. Για λόγους υπολογιστική πολυπλοκότητας - καθόσον ο υπολογισμός ιδιοτιμών απαιτεί υπολογισμό τετραγωνικών ριζών - χρησιμοποιούμε την παρακάτω μετρική γωνιότητας:

$$R(p_1, p_2) = det(\mathbf{M}(p_1, p_2)) - kTrace(\mathbf{M}(p_1, p_2))^2 \tag{7}$$

όπου $k > 0$ μια οριζόμενη από εμάς παράμετρος. Επειδή

$$R(p_1, p_2) = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{8}$$

η τιμή R θα είναι θετική όταν το σημείο (p_1, p_2) είναι κοντά σε γωνίες, αρνητική στη γειτονιά ακμών και κοντά στο μηδέν σε ομοιόμορφες περιοχές. Κατάλληλα κατώφλια μπορούν να χρησιμοποιηθούν για να επιλεγούν οι πλέον ευδιάκριτες γωνίες.

Κατασκευάστε την συνάρτηση:

```
harris_response: np.ndarray = my_corner_harris(
    img: np.ndarray,
    k: float,
    sigma: float
    )
```

η οποία υλοποιεί τον αλγόριθμο, όπως περιγράφεται παραπάνω. Η μεταβλητή `img` είναι ένας πίνακας 2 διαστάσεων και περιέχει μία εικόνα σε gray scale, με τιμές πραγματικούς αριθμούς στο διάστημα $[0, 1]$. Το όρισμα `k` αναφέρεται στην αντίστοιχη παράμετρο της Εξίσωσης 7, ενώ το `sigma` παριστά την τυπική απόκλιση στη σχέση 1. Επιπλέον, το μήκος της πλευράς του τετράγωνου Gaussian παραθύρου, ισούται με `round(4 * sigma)`

Η έξοδος `harris_response` είναι μια εικόνα ίδιων διαστάσεων με αυτές της εισόδου, η τιμή της οποίας στη θέση (p_1, p_2) ισούται με $R(p_1, p_2)$ σύμφωνα με τη σχέση 7.

Σε συνέχεια της προηγούμενης, να υλοποιηθεί η συνάρτηση:

```
corner_locations: np.ndarray = my_corner_peaks(
    harris_response: np.ndarray,
    rel_threshold: float
    )
```

η οποία δέχεται ως είσοδο την εικόνα εξόδου R της προηγούμενης διαδικασίας, καθώς και την παράμετρο `rel_threshold`. Η τελευταία, μέσω του κατωφλίου `R.max() * rel_threshold` ορίζει ποια δείγματα της απόκρισης `R` θα θεωρούνται οι τελικές θέσεις γωνιών της αρχικής εικόνας. Η έξοδος `corner_locations` είναι ένας 2-D `numpy array` 2 στηλών, κάθε γραμμή του οποίου περιλαμβάνει τις 2 συντεταγμένες των θέσεων των γωνιών, που έχουν εντοπιστεί σαν αποτέλεσμα της διαδικασίας αυτής.

Σημειώνεται, ότι στη βιβλιοθήκη `scikit-image` υπάρχουν οι υλοποιήσεις των παραπάνω διαδικασιών, ως `corner_harris()`, `corner_peaks()`. Περισσότερες πληροφορίες υπάρχουν στη σελίδα του επίσημου documentation[3].

2.2.1 Παραδοτέα

Να κατασκευάσετε το Python script `deliverable_2.py` στο οποίο θα δείχνετε:

1. Τις γωνίες που εντοπίσατε πάνω στην grayscale εικόνα εισόδου δημιουργώντας ένα κόκκινο marker (χρησιμοποιώντας `marker = "s"` στη `plt.scatter()`) με κέντρο την κάθε εντοπισμένη κορυφή.

Για το παραδοτέο της ενότητας να χρησιμοποιήσετε την εικόνα `im2.jpg` που σας δίνεται μαζί με την εκφώνηση σαν εικόνα εισόδου.

2.3 Rotation

Κατασκευάστε την συνάρτηση:

```
rot_img: np.ndarray = my_img_rotation(img: np.ndarray, angle: float)
```

η οποία θα λαμβάνει σαν είσοδο μία εικόνα `img` και θα την περιστρέφει αντίστροφα από την φορά του ρολογιού κατά γωνία `angle` σε rads. Η συνάρτηση σας θα πρέπει να λειτουργεί ανεξάρτητα του αριθμού των καναλιών της εικόνας εισόδου (π.χ. RGB ή grayscale). Η εικόνα εξόδου `rot_img` θα πρέπει να έχει τις κατάλληλες διαστάσεις για να χωράει ολόκληρη την εικόνα εισόδου μετά την περιστροφή της. Υποθέστε ότι το background είναι μαύρο.

Η ιδέα πίσω από την περιστροφή της εικόνας είναι να βρείτε την αντιστοίχιση του κάθε pixel στην μετασχηματισμένη εικόνα ως προς την αρχική εικόνα εισόδου. Για παράδειγμα, αντί να πείτε πως το pixel με συντεταγμένες (1,1) στην αρχική εικόνα είναι το pixel με συντεταγμένες (3.4, 6.1) στην μετασχηματισμένη, θα πρέπει να κάνετε το αντίθετο (πως το pixel με συντεταγένες (1,1) στην μετασχηματισμένη εικόνα είναι το pixel (.4, 1.4) της αρχικής. Επιπλέον, θεωρήστε `bilinear interpolation` (Παράδειγμα στον πίνακα 2) για να υπολογίσετε την τιμή του pixel. Οποιαδήποτε στρογγυλοποίηση συντεταγμένων pixel προκύψει να είναι προς τα κάτω. Τέλος, για τις οριακές περιπτώσεις μπορείτε να θεωρήσετε πως η εικόνα εισόδου επεκτείνεται στο άπειρο με μαύρα pixels.

p_1	p_2	p_3
p_4	$\mathbf{p_5} = ?$	p_6
p_7	p_8	p_9

Πίνακας 2: Παράδειγμα bilinear interpolation. Η τιμή του pixel p_5 είναι το average των γειτονικών pixels, $p_5 = \frac{(p_2+p_4+p_6+p_8)}{4}$

2.3.1 Παραδοτέα

Να κατασκευάσετε το Python script `deliverable_3.py` στο οποίο θα:

1. Πραγματοποιήστε περιστροφή της εικόνας εισόδου κατά $54^\circ \times \pi/180^\circ$ rads, δείξτε το αποτέλεσμα.

2. Πραγματοποιήστε περιστροφή της εικόνας εισόδου κατά $213^\circ \times \pi/180^\circ$ rads, δείξτε το αποτέλεσμα.

Και για τα δύο παραδοτέα της ενότητας να χρησιμοποιήσετε την εικόνα `im2.jpg` που σας δίνεται μαζί με την εκφώνηση σαν εικόνα εισόδου. Οι περιστροφές είναι ανεξάρτητες η μία από την άλλη.

[3] `https://scikit-image.org/docs/stable/api/skimage.feature.html#skimage.feature.corner_harris`

3 Το πρόβλημα

Σε αυτή την ενότητα θα χρησιμοποιήσετε τις τρομερές γνώσεις που έχετε αποκτήσει μέχρι τώρα στην ψηφιακή επεξεργασία εικόνας καθώς και τις συναρτήσεις που έχετε φτιάξει σε αυτή την εργασία για να λύσετε το παρακάτω πρόβλημα.

Θεωρήστε πως η εικόνα εισόδου που σας δόθηκε δημιουργήθηκε από μια άτακτη τοποθέτηση εγγράφων πάνω στην επιφάνεια ενός scanner. Παρά την αδιανόητη τεμπελιά μας θα θέλαμε να έχουμε μια ψηφιοποιήμενη έκδοση κάθε εγγράφου που τοποθετήσαμε στο scanner. Για να γίνει αυτό, ο αλγόριθμος σας θα πρέπει να εντοπίζει κάθε ένα από τα έγγραφα τις εικόνας (ή όσα περισσότερα μπορείτε), να τα απομονώνει, στην συνέχεια να τα περιστρέφει κατάλληλα και τέλος να τα αποθηκεύει σαν ανεξάρτητες εικόνες στο ίδιο format με την εικόνα εισόδου και πρόσθετη κατάληξη _1, _2 κ.λπ.

Φτιάξτε το Python script με όνομα `my_lazy_scanner.py` το οποίο θα φορτώνει την εικόνα εισόδου με στόχο την ανεξάρτητη ψηφιοποίηση των εγγράφων της. Το script θα πρέπει να ζεί στο ίδιο directory με τις συναρτήσεις σας. Το script πρέπει να μας πληροφορεί πόσες εικόνες εντόπισε και πόσες τελικά πέτυχε να γράψει στο δίσκο σαν αυτόνομα αρχεία.

Να δείξετε την λειτουργία της μεθόδού σας και για τις πέντε εικόνες που σας δίνονται μαζί με την εκφώνηση. Τέλος, είναι απαραίτητο το script σας να λειτουργεί χωρίς a-priori γνώση της εικόνας εισόδου και να λειτουργεί με τις **ίδιες** παραμέτρους **για όλες** τις εικόνες εισόδου (one-script-fits-all λογική).

4 Για την υποβολή της εργασίας

Παραδώστε μία αναφορά με τις περιγραφές και τα συμπεράσματα που σας ζητούνται στην εκφώνηση. Η αναφορά θα πρέπει να επιδεικνύει την ορθή λειτουργία του κώδικά σας στην εικόνα που σας δίνεται και να παρουσιάζει και σχολιάζει τις εικόνες και τα ιστογράμματα που παράγονται από το πρόγραμμα επίδειξης.

Ο κώδικας θα πρέπει να είναι σχολιασμένος ώστε να είναι κατανοητό τι ακριβώς λειτουργία επιτελεί (σε θεωρητικό επίπεδο, όχι σε επίπεδο κλίσης συναρτήσεων). Επίσης, ο κώδικας θα πρέπει να εκτελείται και να υπολογίζει τα σωστά αποτελέσματα για *οποιαδήποτε* είσοδο πληροί τις υποθέσεις της εκφώνησης, και όχι μόνο για την εικόνα που σας δίνεται.

Απαραίτητες προϋποθέσεις για την βαθμολόγηση της εργασίας σας είναι ο κώδικας να εκτελείται χωρίς σφάλμα (μόνο demos που εκτελούνται επιτυχώς θα βαθμολογηθούν), καθώς και να τηρούνται τα ακόλουθα:

- Υποβάλετε ένα και μόνο αρχείο, τύπου zip.
- Το όνομα του αρχείου πρέπει να είναι `AEM.zip`, όπου `AEM` είναι τα τέσσερα ψηφία του Α.Ε.Μ. του φοιτητή της ομάδας.
- Το προς υποβολή αρχείο πρέπει να περιέχει τα αρχεία κώδικα Python και το αρχείο `report.pdf` το οποίο θα είναι η αναφορά της εργασίας.
- Η αναφορά πρέπει να είναι ένα αρχείο τύπου PDF, και να έχει όνομα `report.pdf`.
- Όλα τα αρχεία κώδικα πρέπει να είναι αρχεία κειμένου τύπου UTF-8, και να έχουν κατάληξη `py`.
- Το αρχείο τύπου zip που θα υποβάλετε δεν πρέπει να περιέχει κανέναν φάκελο.
- Μην υποβάλετε τις εικόνες που σας δίνονται για πειραματισμό.
- Μην υποβάλετε αρχεία που δεν χρειάζονται για την λειτουργία του κώδικά σας, ή φακέλους/αρχεία που δημιουργεί το λειτουργικό σας, πχ “Thumbs.db”, “.DS_Store”, “.directory”.
- Για την ονομασία των αρχείων που περιέχονται στο προς υποβολή αρχείο, χρησιμοποιείτε μόνο αγγλικούς χαρακτήρες, και όχι ελληνικούς ή άλλα σύμβολα, πχ “#”, “$”, “%” κλπ.

Αναφορές

[1] C. Harris and M. Stephens, *A Combined Corner and Edge Detector*, Proceedings of the 4th Alvey Vision Conference, August 1988, pp. 147-151