



Εργασία 2: Hough, Harris, Rot και αποκοπή εικόνων

Παλάσκα Αιμιλία 10453 (aimiliapm@ece.auth.gr)

Επιβλέποντες: Αναστάσιος Ντελόπουλος, Αλέτρας Δημήτριος

Λειτουργία συναρτήσεων

Τα προγράμματα μπορούν να κληθούν είτε από κάποιο IDE π.χ. VSCode είτε με την εντολή τερματικού `/bin/python3 demo.py`. Να σημειωθεί πως σε κάθε deliverable και στο my lazy scanner πρέπει να γίνει αλλαγή της μεταβλητής image_path στο κατάλληλο directory, δηλαδή αυτό του φακέλου images, και επίσης απαιτούνται μερικά λεπτά για να ολοκληρωθεί η εκτέλεσή τους. Επίσης, ο κώδικας μπορεί να βρεθεί και στο [GitHub](#). Στο πρόγραμμα συμπεριλαμβάνονται τα παρακάτω αρχεία και συναρτήσεις: (περισσότερα στοιχεία για την λειτουργία τους στα σχόλια του κώδικα)

Αρχείο image_preprocessing.py

- ❖ `image_preprocessing(img:Image, contrast:float, ratio:int, blur:int) -> np.ndarray:`
προ-επεξεργασία εικόνας μέσω μετατροπής σε ασπρόμαυρη, σμίκρυνσης, αύξησης αντίθεσης και εφαρμογής Gaussian blur.

Αρχείο plot_line_on_image.py

- ❖ `find_intersection(line1: tuple, line2:tuple):` εύρεση των σημείου τομής δύο ευθειών
- ❖ `plot_line_on_image(image:np.ndarray, L:np.ndarray, color:tuple) -> np.ndarray:`
χάραξη ευθειών με συγκεκριμένο χρώμα σε δοσμένη εικόνα.

Αρχείο my_hough_transform.py

- ❖ `my_hough_transform(binary_image: np.ndarray, d_rho: int, d_theta: float, n: int) -> tuple[np.ndarray, np.ndarray, int]:` εφαρμογή του μετασχηματισμού Hough σε δυαδική εικόνα ακμών και επιστροφή του accumulator, των συντελεστών (ρ, θ) ή σημαντικότερων γραμμών και του αριθμού ακμών που δεν συμπεριλαμβάνονται στις ευθείες.

Αρχείο my_harris.py

- ❖ `gaussian_kernel(size: int, sigma: float) -> np.ndarray:` δημιουργία Gaussian kernel για εφαρμογή του αλγορίθμου Harris
- ❖ `my_corner_harris(image:np.ndarray, k:float, sigma:float) -> np.ndarray:` επιστροφή του harris_response μιας εικόνας
- ❖ `my_corner_peaks(harris_response: np.ndarray, rel_threshold: float) -> np.ndarray:` εφαρμογή κατωφλίου στο harris response και επιστροφή των συντεταγμένων των κορυφών που εντοπίστηκαν

Αρχείο my_img_rotation.py

- ❖ `my_img_rotation(img: np.ndarray, angle: float) -> np.ndarray:` περιστροφή εικόνας ανθωρολογιακα κατά δοσμένη γωνία

Αρχείο resembles_parallelgram.py

- ❖ `perpendicular_intersection(L: np.ndarray) -> np.ndarray:` υπολογισμός σημείου τομής όσων ευθειών έχουν διαφορά γωνίας $\sim 90^\circ$ ή ισοδύναμα $\sim 270^\circ$



- ❖ `resembles_parallelgram(points:list) -> np.ndarray`: διάταξη σε τετράδες τους συνδυασμούς σημείων που τείνουν σε σχήμα παραλληλογράμμου

Αρχείο deliverable_1.py

- ❖ `main`: εισαγωγή εικόνας `im2.jpg`, προ-επεξεργασία, εφαρμογή μετασχηματισμού Hough, αποθήκευση τοπικά τόσο του μετασχηματισμού όσο και της αρχικής εικόνας με τις εντοπισμένες ευθείες και προβολή των συντελεστών των πιο σημαντικών γραμμών και του αριθμού των λοιπών σημείων.

Αρχείο deliverable_2.py

- ❖ `main`: εισαγωγή εικόνας `im2.jpg`, προ-επεξεργασία, εφαρμογή αλγορίθμου Harris, κατωφλίωση του `response` του και αποθήκευση τοπικά τόσο του `response` όσο και της αρχικής εικόνας με τις εντοπισμένες γωνίες

Αρχείο deliverable_3.py

- ❖ `main`: εισαγωγή εικόνας `im2.jpg`, περιστροφή κατά 54° και μέτα κατά 213°

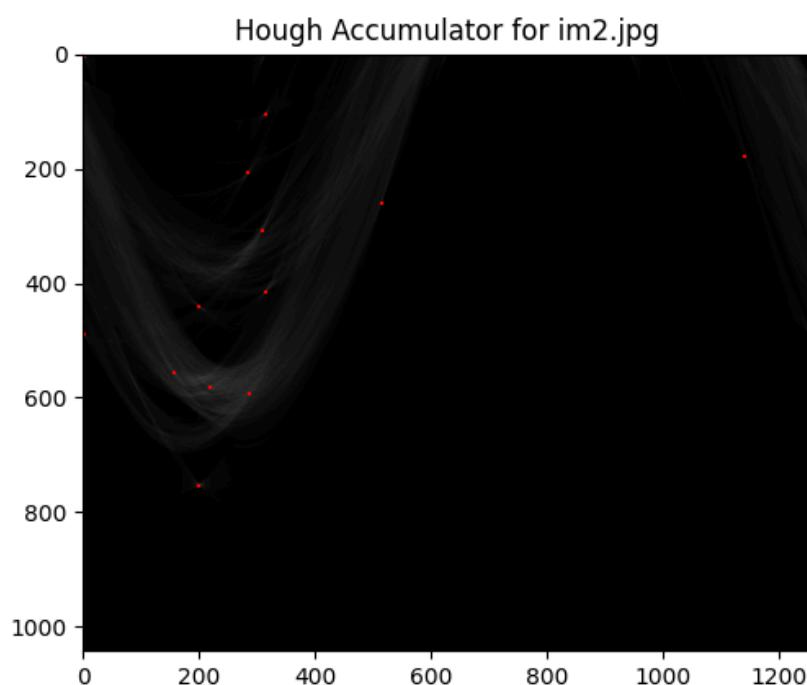
Αρχείο my_lazy_scanner.py

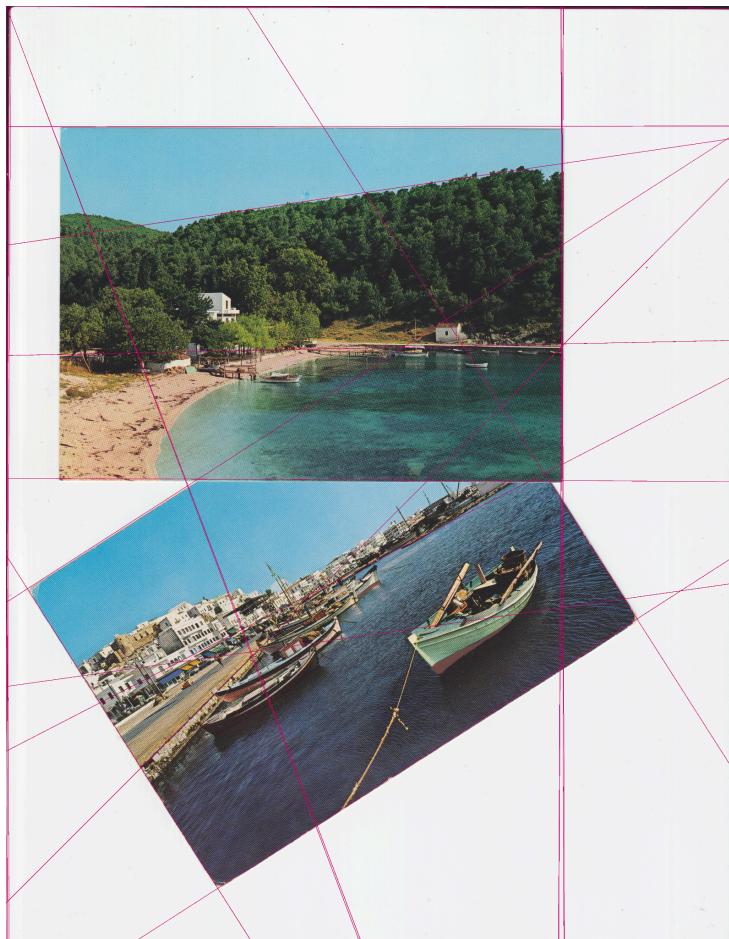
- ❖ `main`: εισαγωγή εικόνας `im1.jpg`, αναγνώριση κυρίαρχων γραμμών και γωνιών, εντοπισμός σημείων τομής κάθετων ευθειών και αντιστοίχιση σε αντίστοιχες γωνίες, περιστροφή και αποθήκευση τοπικά

Παρουσίαση και σχολιασμός αποτελεσμάτων

Deliverable 1

Παρατίθενται τα αποτελέσματα από προ-επεξεργασία της εικόνας `im2.jpg` (`ratio = width // 600, contrast = 3, blur kernel = 5`), εφαρμογή του edge detection Canny (thresholds 100 και 300) και εκτέλεση του μετασχηματισμού Hough (`d_rho = 1, d_theta = 0.005, n = 20`)





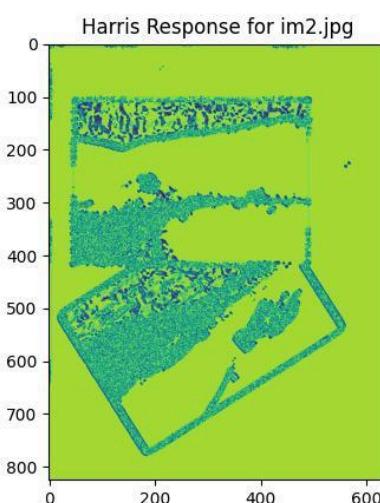
Η επιλογή του Canny edge detector έγινε μετά από διάφορα πειράματα και κρίθηκε πως λειτουργεί ιδανικά μετά από προ-επεξεργασία της εικόνας, δηλαδή σμίκρυνση των διαστάσεων, αύξηση της αντίθεσης και εφαρμογή θαμπώματος.

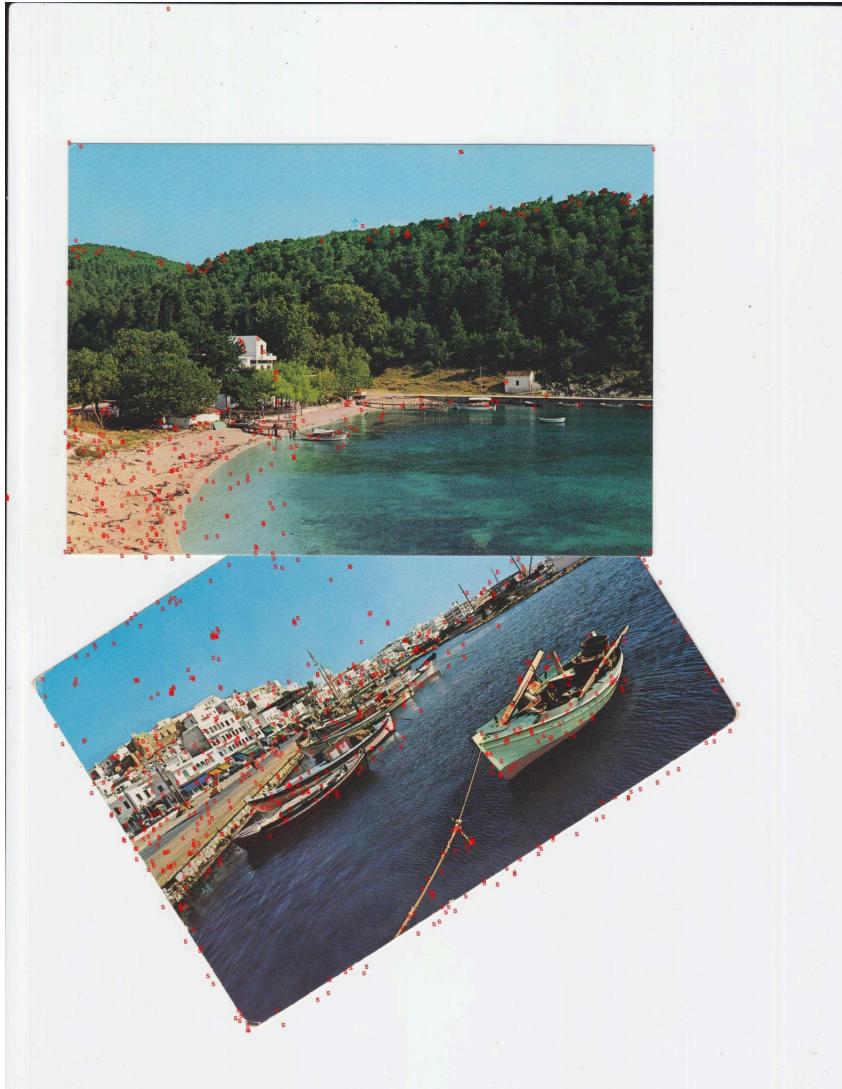
Στα αποτελέσματα είναι φανερό ότι τα όρια των επιμέρους εικόνων εντοπίστηκαν σωστά. Παρ' όλα αυτά, ο αλγόριθμος εντόπισε κάποιες γραμμές στα όρια ή στον ορίζοντα που φαινομενικά βγάζει νόημα, γιατί υπάρχουν γραμμές σε αυτές τις περιοχές.

Πιθανόν με κατάλληλη μεταβολή τόσο της προ-επεργασίας όσο και των παραμέτρων αυτό να βελτιωνόταν σε έναν βαθμό.

Deliverable 2

Παρατίθενται τα αποτελέσματα από προ-επεξεργασία της εικόνας im2.jpg (ratio = width // 600, contrast = 5, blur kernel = 15), εφαρμογή του αλγορίθμου Harris (k = 0.2, sigma = 0.8) και κατωφλίωσή του (rel_threshold = 0.4) με τις τελικές γωνίες να εμφανίζονται με το σύμβολο 's' και κόκκινο χρώμα.





Είναι φανερό ότι εντοπίστηκαν οι ζητούμενες γωνίες, δηλαδή αυτές των εγγράφων, μαζί με άλλες λιγότερο σχετικές στην παρούσα υλοποίηση. Η μεγαλύτερη κατωφλίωση οδηγούσε σε μη-θεώρηση των κορυφών οπότε προτιμήθηκαν αυτές οι παράμετροι, χωρίς να είναι απόλυτα ιδανικές.

Deliverable 3

Παρατίθενται τα αποτελέσματα από εισαγωγή της εικόνας im2.jpg και περιστροφής της πρώτα κατά 54° και έπειτα κατά 213°





My Lazy Scanner

Τονίζεται ότι η υλοποίηση αυτή δεν παράγει τα θεμιτά αποτελέσματα, αλλά αποτελεί περισσότερο μια προσπάθεια επίλυσης του προβλήματος. Στα πλαίσια της εξαγωγής πολλαπλών εγγράφων από μία εικόνα, υλοποιήθηκαν δύο βοηθητικές συναρτήσεις για εντοπισμό κάθετων διασταυρώσεων και σημείων σε σχήμα παραλληλογράμμου. Επίσης δοκιμάστηκε να εντοπιστεί έγγραφο με μόνο 4 ευθείες αλλά δυστυχώς ούτε αυτή η προσέγγιση ήταν λειτουργική, καθώς οι 4 ισχυρότερες ευθείες δεν είναι πάντα αυτές που πλαισίωναν την μία επιμέρους φωτογραφία. Τέλος, αν εκτελεστεί το αρχείο `my_lazy_scanner.py` παράγονται 200-300 κομμένες και περιστραμμένες φωτογραφίες για να επιλέξει ο χρήστης ποια είναι αντιπροσωπευτική.

Πηγές

- ❖ Γλωσσικό μοντέλο [ChatGPT 3.5](#)
- ❖ [Canny Edge Detection](#) της βιβλιοθήκης OpenCV
- ❖ [Visual Studio Code](#) - Community Edition
- ❖ [Python](#) - Version 3.10.12