ΨΗΦΙΑΚΗ ΕΠΕΞΕΡΓΑΣΙΑ ΕΙΚΟΝΑΣ

ΕΡΓΑΣΙΑ 2

ΕΠΙΜΕΛΕΙΑ : ΠΙΤΤΗΣ ΓΕΩΡΓΙΟΣ
ΑΕΜ : 10586
ΙΔΡΥΜΑΤΙΚΟ EMAIL :
gkpittis@ece.auth.gr

ΠΕΡΙΕΧΟΜΕΝΑ

2.1 Hough transform

- Ο μετασχηματισμός Hough είναι μια μέθοδος που χρησιμοποιούμε για την συνένωση των ακμών μιας binary εικόνας. Όλα τα pixel της binary εικόνας έχουν μαύρο χρώμα (τιμή 0) , εκτός από εκείνα που ενδέχεται να αντιστοιχούν σε ακμές της εικόνας τα οποία έχουν άσπρο χρώμα (τιμή 1). Πολλές φορές η binary εικόνα δεν έχει την αναμενόμενη ποιότητα , με αποτέλεσμα οι άσπρες ακμές που παρατηρώ είτε να είναι διακεκομμένες , είτε να είναι "σπασμένες"(δηλαδή δεν αντιστοιχούν σε αρκετά απότομη αλλαγή της φωτεινότητας), είτε ακόμη να αποτελούν θόρυβο. Ο μετασχηματισμός Hough επιλύει αυτό το πρόβλημα. Πιο συγκεκριμένα, δέχεται στην είσοδό του μια binary εικόνα και ψάχνει για άσπρα pixel(δηλαδή για ακμές) , τα οποία αν ενωθούν μεταξύ τους θα σχηματίσουν μια ευθεία. Αυτό το πετυχαίνει δημιουργώντας τον πίνακα H. Κάθε κελί του πίνακα H αντιστοιχεί σε μια μοναδική ευθεία ε (ρ, θ). Το ρ είναι η απόσταση της ευθείας ε από την αρχή των αξόνων . Το θ είναι η γωνία που σχηματίζει η ευθεία που είναι κάθετη στην ε με τον οριζόντιο άξονα. Τα άσπρα pixel της binary εικόνας μπορούν να ανήκουν σε καμία , σε μία ή και σε περισσότερες ευθείες (ρ, θ). Κάθε άσπρο pixel θα ρίξει μια ψήφο στα κελιά του πίνακα H , που αντιστοιχούν σε ευθείες που διέρχονται από αυτό. Επομένως, όλα τα συνευθειακά pixel θα ψηφίσουν όλα μαζί το κελί που αντιπροσωπεύει την ευθεία που τα ενώνει. Τέλος, βρίσκω τα κελιά με τις περισσότερες ψήφους, δηλαδή βρίσκω τα τοπικά μέγιστα του πίνακα H. Κάθε τοπικό μέγιστο του πίνακα H ταυτίζεται με μια ευθεία που υπάρχει στην εικόνα.

- **H , L , res = my_hough_transform(img_binary, d_rho, d_theta, n)**

  Η συνάρτηση λαμβάνει ως είσοδο τη binary εικόνα img_binary η οποία έχει προέλθει από κατωφλίωση της εξόδου του feature.canny edge detector 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. Επίσης, λαμβάνει ως είσοδο 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. Αρχικά, υπολογίζω τις διαστάσεις της εικόνας εισόδου (N_1, N_2) και στη συνέχεια θέτω το res ίσο με το συνολικό πλήθος των pixel της εικόνας. Έπειτα, βρίσκω την μέγιστη δυνατή απόσταση rho_max $= \sqrt{{N_1}^2 + {N_2}^2}$ κάθε ευθείας από την αρχή των αξόνων και εν συνεχεία καθορίζω το εύρος των τιμών του ρ (rhos). Ειδικότερα, το ρ παίρνει τιμές από $-$rho_max έως $+$rho_max. Επίσης, η γωνία θ (thetas) των ευθειών παίρνει τιμές από $-90°$ έως $+90°$. Ο πίνακας t_rad περιέχει τις τιμές των γωνιών θ σε rads. Δημιουργώ έναν κενό πίνακα H , ο οποίος έχει διαστάσεις RxT. Το R είναι το μήκος του πίνακα rhos και T είναι το μήκος του πίνακα thetas, δηλαδή ο πίνακας H έχει διαστάσεις ίσες με το πλήθος των bins των ρ και θ , αφού κάθε κελί του αντιπροσωπεύει ένα

ζευγάρι (ρ, θ). Στην συνέχεια, ο μετασχηματισμός Hough σκανάρει τη binary εικόνα εισόδου και εντοπίζει τα άσπρα pixel, δηλαδή εντοπίζει τις ενδεχόμενες ακμές της εικόνας. Για κάθε άσπρο pixel, ο μετασχηματισμός Hough βρίσκει όλες τις ευθείες που διέρχονται από αυτό το pixel. Δηλαδή για κάθε γωνία θ που περιέχεται στον πίνακα thetas , υπολογίζει την απόσταση $r = n1 * \cos(\theta) + n2 * \sin(\theta)$. Εξισώνει την απόσταση ρ της ευθείας από την αρχή των αξόνων με την τιμή του πίνακα rhos που βρίσκεται πιο κοντά στο r. Έτσι, για κάθε ευθεία που διέρχεται από το άσπρο pixel , υπολογίζει το ζευγάρι (ρ, θ) που την αντιπροσωπεύει και αυξάνει κατά 1 τα στοιχεία του πίνακα H που αντιστοιχούν στις συγκεκριμένες ευθείες. Η παραπάνω διαδικασία επαναλαμβάνεται για κάθε άσπρο pixel της binary εικόνας. Πλέον, ο πίνακας συσσώρευσης H έχει γεμίσει με τις ψήφους όλων των pixel της εικόνας. Ακόμη, χρησιμοποιώ την συνάρτηση maximum_filter του scipy.ndimage και δημιουργώ τον δυαδικό πίνακα local_max_binary. Ο local_max_binary έχει ίδιες διαστάσεις με τον πίνακα H και μάλιστα έχει τιμή True(τιμή 1) στις θέσεις εκείνες που στον πίνακα H αποτελούν τοπικά μέγιστα. Δηλαδή, αν η θέση H[i,j] αντιστοιχεί σε τοπικό μέγιστο , τότε το στοιχείο local_max_binary[i,j] έχει τιμή 1. Επιπλέον, ο local_max_binary έχει τιμή False(τιμή 0) στις υπόλοιπες θέσεις. Πολλαπλασιάζω τον H με τον local_max_binary και αποθηκεύω τα τοπικά μέγιστα στον πίνακα local_max. Έπειτα, βρίσκω τα n μεγαλύτερα τοπικά μέγιστα του πίνακα H και με αυτόν τον τρόπο προσδιορίζω τις n ισχυρότερες ευθείες της εικόνας. Κάθε φορά που βρίσκω το μεγαλύτερο τοπικό μέγιστο, δηλαδή κάθε φορά που βρίσκω το μεγαλύτερο στοιχείο του πίνακα local_max , τότε το αποθηκεύω στον πίνακα max_cells και στην συνέχεια το διαγράφω από τον πίνακα local_max. Αν η διαδικασία αυτή επαναληφθεί n φορές , τότε θα έχω ανιχνεύσει επιτυχώς τα n μεγαλύτερα τοπικά μέγιστα και θα τα έχω αποθηκεύσει στον πίνακα max_cells. Με την βοήθεια του max_cells παράγω τον πίνακα L. Επίσης, έχω ήδη μετρήσει το πλήθος των pixels που ανήκουν στις n ισχυρότερες ευθείες και το έχω αποθηκεύσει στην μεταβλητή count. Τέλος , αφαιρώ το count από το res και βρίσκω την τελική τιμή του res.

- **deliverable_1.py** :

  - Κάνω resize την εικόνα im2.jpg με scale_factor = 0,2 για να αποφύγω υπερβολικά μεγάλους χρόνους εκτέλεσης.

  - plt.figure(1, figsize=(img_resized.shape[1] / 100, img_resized.shape[0] / 100))
    Προσαρμόζω τις διαστάσεις του σχήματος figure 1 , για να χωράει ακριβώς σε αυτό η αρχική εικόνα εισόδου με τις n κόκκινες ισχυρότερες ευθείες.

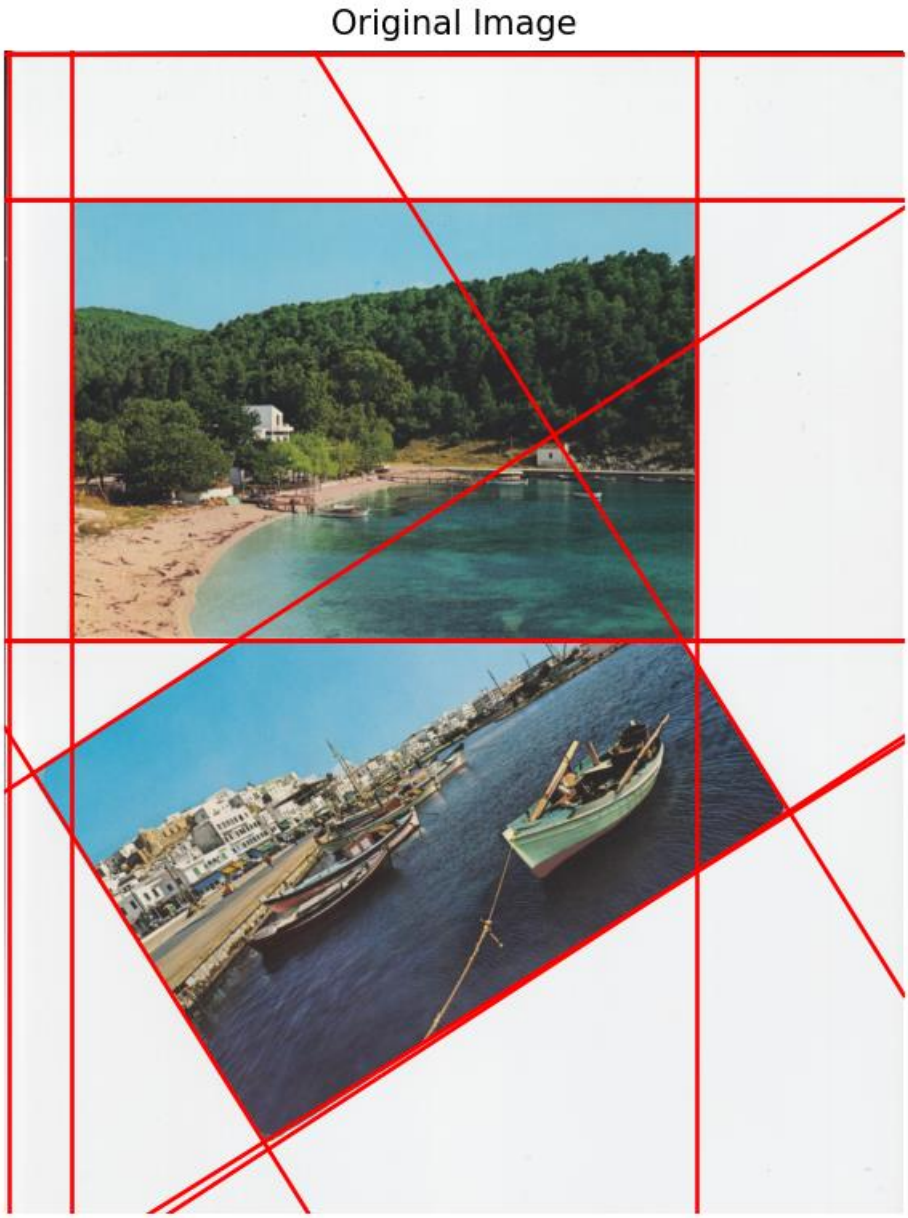
  - I1 = gaussian_filter(I1, sigma=4)
    Χρησιμοποιώ το φίλτρο εξομάλυνσης Gauss για να μειώσω τον θόρυβο στην εικόνα.

  - binary_image = feature.canny(I1, sigma=1)
    Ο edge detector Canny είναι ένας δημοφιλής αλγόριθμος ανίχνευσης ακμών. Είναι γνωστός για την ικανότητά του να ανιχνεύει πραγματικές

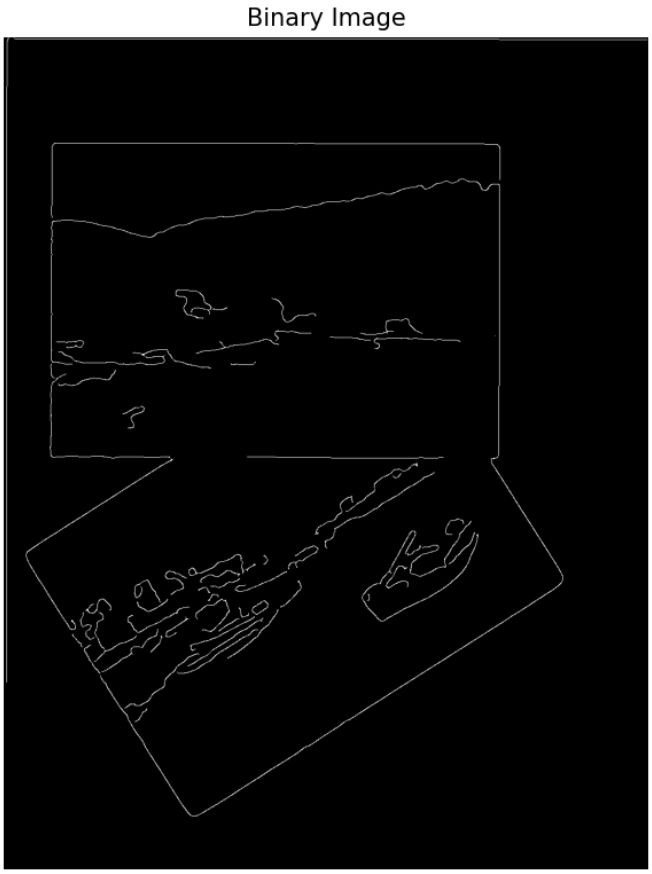
ακμές και να μειώνει την πιθανότητα ανίχνευσης θορύβου ως ακμές. Ο edge detector Canny παράγει την binary εικόνα που θα πάρει ως είσοδο η συνάρτηση my_hough_transform.

- Η ζητούμενη εικόνα έχει προκύψει για τις τιμές ρ και θ που αναφέρθηκαν παραπάνω και n=14.
  Επιλέχθηκαν διακριτότητες d_rho = 1 pixel και d_theta = $\frac{\pi}{180°}$ rads .

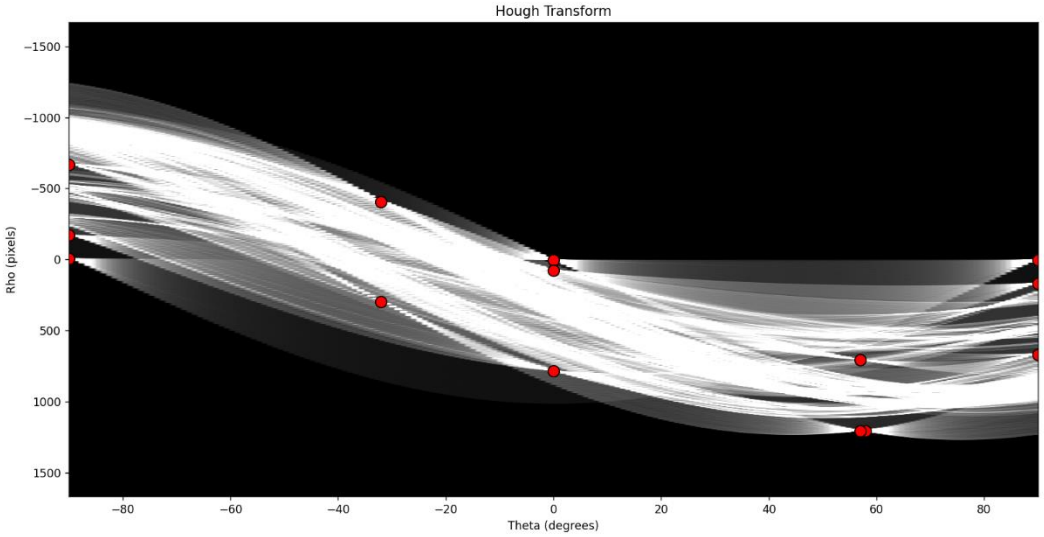
2.1.1 ΑΠΟΤΕΛΕΣΜΑΤΑ



Εικόνα 1 : Η αρχική εικόνα με τις n ισχυρότερες ευθείες που βρέθηκαν από τον μετασχηματισμό Hough



Εικόνα 2 : Η binary εικόνα που παράχθηκε από τον edge detector Canny



Εικόνα 3 : Ο πίνακας μετασχηματισμού Η και οι κορυφές(πίνακας L) που εντοπίστηκαν

2.2 Harris corner detector

- Ο Harris Corner Detector καθορίζει αν ένα pixel της εικόνας βρίσκεται σε γωνία, σε ακμή ή σε ομοιόμορφη περιοχή. Στην παρούσα εργασία με ενδιαφέρει μόνο ο εντοπισμός των γωνιών της αρχικής εικόνας εισόδου. Στο pdf της εκφώνησης γίνεται αναλυτική περιγραφή του αλγορίθμου.

- **harris_response = my_corner_harris(img, k, sigma)**

  Η συνάρτηση υλοποιεί τον αλγόριθμο του Harris corner detector. Παίρνει ως είσοδο την grayscale εικόνα img , τα pixel της οποίας παίρνουν τιμές πραγματικούς αριθμούς στο διάστημα [0, 1]. Επίσης, δέχεται στην είσοδο το όρισμα k που αναφέρεται στην αντίστοιχη παράμετρο της Εξίσωσης 7 , καθώς και το όρισμα sigma που αποτελεί την τυπική απόκλιση στη σχέση 1. Επιστρέφει τον πίνακα harris_response , ο οποίος ταυτίζεται με τον πίνακα R των εξισώσεων 7 και 8. Αρχικά, θέτω το μήκος της πλευράς του τετράγωνου Gaussian παραθύρου(filter_size) , ίσο με round(4 * sigma). Στη συνέχεια, δημιουργώ το πλέγμα συντεταγμένων x και y για το Gaussian παράθυρο. Υπολογίζω την συνάρτηση w(x, y) της σχέσης 1 (πίνακας g) , η οποία έχει μη μηδενικές τιμές κοντά στην αρχή των αξόνων και "πεθαίνει" καθώς το (x, y) απομακρύνεται από το (0, 0). Η συνάρτηση w(x, y) και κατ' επέκταση ο πίνακας g , αποτελεί ουσιαστικά ένα φίλτρο εξομάλυνσης Gauss. Ορίζω τις μάσκες Sobel για την οριζόντια και για την κατακόρυφη κατεύθυνση (H_horizontal και H_vertical αντίστοιχα). Πραγματοποιώ συνέλιξη της εικόνας εισόδου με την μάσκα H_horizontal για να υπολογίσω την μερική παράγωγο I_xτης εικόνας στην οριζόντια κατεύθυνση. Πραγματοποιώ συνέλιξη της εικόνας εισόδου με την μάσκα H_vertical για να υπολογίσω την μερική παράγωγο I_yτης εικόνας στην κατακόρυφη κατεύθυνση. Για τις παραπάνω συνελίξεις χρησιμοποιώ την έτοιμη συνάρτηση convolve2d. Υπολογίζω τα στοιχεία του πίνακα M της εξίσωσης 5, δηλαδή υπολογίζω τα $I^2{}_x$, $I^2{}_y$, I_{xy} . Για να υπολογίσω τα στοιχεία του πίνακα M, κάνω συνέλιξη των στοιχείων Ix*Ix , Iy*Iy , Ix*Iy με τον πίνακα g που υπολόγισα παραπάνω. Για να πραγματοποιήσω τις συνελίξεις αυτές χρησιμοποίησα την έτοιμη συνάρτηση convolve. Η ορίζουσα του M είναι : $I^2{}_x \cdot I^2{}_y - I^2{}_{xy}$ και το ίχνος του M είναι : $I^2{}_x + I^2{}_y$. Κάνω χρήση της εξίσωσης 7 και υπολογίζω τον πίνακα harris_response. Ο πίνακας harris_response είναι ουσιαστικά ο detector και το πρόσημο του καθορίζει αν έχω γωνία , ακμή ή ομοιόμορφη περιοχή.

- **corner_locations = my_corner_peaks(harris_response, rel_threshold)**

  Η συνάρτηση δέχεται ως είσοδο την εικόνα εξόδου harris_response της προηγούμενης διαδικασίας, καθώς και την παράμετρο rel_threshold. Η τελευταία, μέσω του κατωφλίου harris_response.max() * rel_threshold ορίζει ποια δείγματα της απόκρισης harris_response θα θεωρούνται οι τελικές θέσεις γωνιών της αρχικής εικόνας. Επιστρέφει τον πίνακα corner_locations, κάθε γραμμή του οποίου περιλαμβάνει τις 2 συντεταγμένες των θέσεων των γωνιών, που έχουν εντοπιστεί σαν αποτέλεσμα της διαδικασίας αυτής. Αρχικά εντοπίζει τα τοπικά μέγιστα του πίνακα harris_response και τα αποθηκεύει

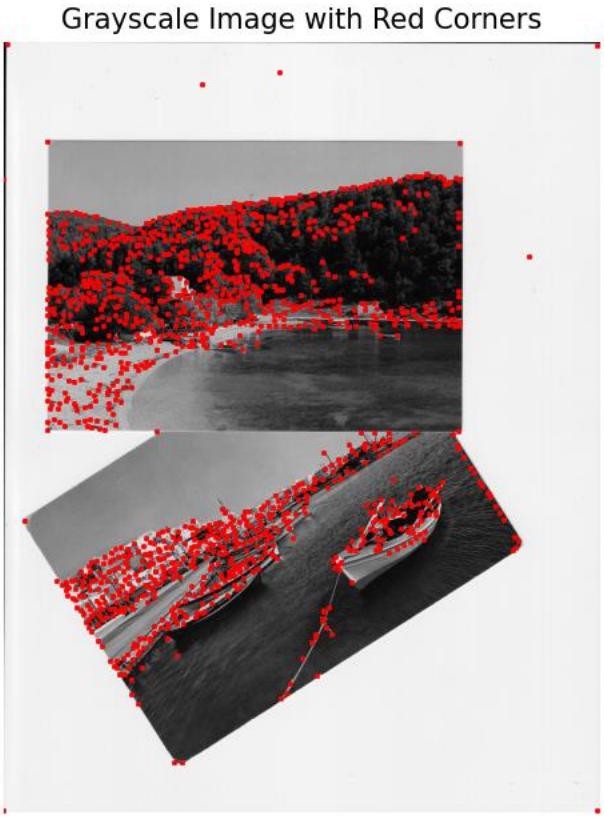
στον δυαδικό πίνακα local_max. Δηλαδή, αν το harris_response[i, j] είναι τοπικό μέγιστο, τότε local_max[i, j] = True. Διαφορετικά, αν το harris_response[i, j] δεν είναι τοπικό μέγιστο, τότε local_max[i, j] = False. Βρίσκω τα τοπικά μέγιστα του harris_response , διότι είναι τα σημεία που με ενδιαφέρουν όταν ψάχνω για γωνίες. Βρίσκω το threshold από την πράξη harris_response.max() * rel_threshold. Οι ζητούμενες γωνίες που θα αποθηκεύσω στον πίνακα corner_locations , ταυτίζονται με τα τοπικά μέγιστα που έχουν μεγαλύτερη τιμή από το threshold. Στην εργασία αυτό που μας ενδιαφέρει είναι να εντοπίζονται ικανοποιητικά οι 4 γωνίες του περιγράμματος κάθε φωτογραφίας. Ωστόσο, δεν μπορεί να αποφευχθεί και ο εντοπισμός γωνιών στο εσωτερικό των φωτογραφιών.

- **deliverable_2.py** :

  - Κάνω resize την εικόνα im2.jpg με scale_factor = 0,2 για να αποφύγω μεγάλους χρόνους εκτέλεσης.

  - Χρησιμοποιώ παραμέτρους :
    k = 0,05
    sigma = 2,5
    rel_threshold = 0.00355029

2.2.1 ΑΠΟΤΕΛΕΣΜΑΤΑ



Εικόνα 4 : Οι γωνίες που εντόπισε ο Harris Corner Detector στην grayscale εκδοχή της αρχικής εικόνας

Εικόνα 5 : Οι γωνίες που εντόπισε ο Harris Corner Detector στην αρχική εικόνα

2.3 Rotation

- **rot_img = my_img_rotation(img, angle) :**

Η συνάρτηση λαμβάνει σαν είσοδο μία εικόνα img και την περιστρέφει αντίστροφα από την φορά του ρολογιού κατά γωνία angle σε rads. Η συνάρτηση λειτουργεί ανεξάρτητα του αριθμού των καναλιών της εικόνας εισόδου (π.χ. RGB ή grayscale). Επιστρέφει την εικόνα rot_img , η οποία θα πρέπει να έχει τις κατάλληλες διαστάσεις για να χωράει ολόκληρη την εικόνα εισόδου μετά την περιστροφή της. Αρχικά, ελέγχω αν η εικόνα εισόδου είναι RGB ή grayscale. Αν είναι grayscale , τότε προσθέτω σε αυτή μια τρίτη διάσταση στην οποία θα υπάρχει η τιμή του κάθε pixel της εικόνας. Έτσι, η grayscale εικόνα πλέον θα έχει διάσταση (img.shape[0], img.shape[1], 1). Αν η εικόνα εισόδου είναι RGB τότε θα έχει 3 διαστάσεις και μάλιστα στην τρίτη διάσταση θα περιέχει , για κάθε pixel , ένα 1x3 διάνυσμα χρώματος. Δηλαδή, η RGB εικόνα έχει σχήμα (img.shape[0],

img.shape[1], 3). Η συνάρτηση λοιπόν μπορεί να επεξεργαστεί 3-D πίνακες που αναπαριστούν εικόνες. Στην συνέχεια υπολογίζω τον πίνακα περιστροφής t_rot με βάση την γωνία εισόδου angle : t_rot $= \begin{bmatrix} \cos(angle) & -\sin(angle) & 0 \\ \sin(angle) & \cos(angle) & 0 \\ 0 & 0 & 1 \end{bmatrix}$.
Επίσης, υπολογίζω τον πίνακα t_center που χρησιμοποιώ για να μετατοπίσω το κέντρο της εικόνας στην αρχή των αξόνων :
t_center $= \begin{bmatrix} 1 & 0 & -center[0] \\ 0 & 1 & -center[1] \\ 0 & 0 & 1 \end{bmatrix}$. Το διάνυσμα center = (center[0], center[1]) = $\left(\frac{dims[0]}{2}, \frac{dims[1]}{2}\right)$, περιέχει τις συντεταγμένες του κέντρου της εικόνας εισόδου, αφού εξαρχής στην μεταβλητή dims αποθήκευσα τις διαστάσεις της εικόνας εισόδου (dims = img.shape). Δημιουργώ τους πίνακες center_rot_x και center_rot_y για να αποθηκεύσω τις νέες συντεταγμένες x και y αντίστοιχα των pixel μετά την περιστροφή της εικόνας. Βρίσκω τις ομογενείς συντεταγμένες των pixel της εικόνας προσθέτοντας έναν άσσο $\left(\begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \right)$ και εκτελώ την πράξη :

t_rot $\cdot$ t_center $\cdot$ $\begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$, για να μετατοπίσω αρχικά το κέντρο της εικόνας στην αρχή των αξόνων και στην συνέχεια για να περιστρέψω την εικόνα αντιωρολογιακά κατά γωνία angle. Βρίσκω τις μέγιστες και ελάχιστες x και y συντεταγμένες της περιστραμμένης εικόνας (min_x, min_y, max_x, max_y) , προκειμένου να φτιάξω την τελική εικόνα rot_img με κατάλληλες διαστάσεις ώστε να χωράει ολόκληρη την περιστραμμένη εικόνα. Έπειτα, χρησιμοποιώντας τις συντεταγμένες (i, j) της τελικής εικόνας rot_img , βρίσκω τις αντίστοιχες συντεταγμένες (x, y) της αρχικής εικόνας img με βάση τους τύπους :

$$x = \left(i - \left(\frac{\text{rot\_img. shape[0]}}{2}\right)\right) \cdot \cos(angle) + \left(j - \left(\frac{\text{rot\_img. shape[1]}}{2}\right)\right) \cdot \sin(angle)$$

$$y = -\left(i - \left(\frac{\text{rot\_img. shape[0]}}{2}\right)\right) \cdot \sin(angle) + \left(j - \left(\frac{\text{rot\_img. shape[1]}}{2}\right)\right) \cdot \cos(angle)$$

όπου $\left(\frac{\text{rot\_img.shape[0]}}{2}, \frac{\text{rot\_img.shape[1]}}{2}\right)$ είναι οι συντεταγμένες του κέντρου της περιστραμμένης εικόνας.
Ακόμη, επαναφέρω τις συντεταγμένες που βρήκα στο σύστημα συντεταγμένων της αρχικής εικόνας : x = int(round(x) + center[0])
y = int(round(y) + center[1])
και τις μετατρέπω σε ακέραιους αριθμούς διότι θα αποτελέσουν τους δείκτες της εικόνας img. Τέλος, εφαρμόζω bilinear interpolation και υπολογίζω τις τιμές των pixel στην τελική εικόνα.

- **deliverable_3.py** :
  - Κάνω resize την εικόνα im2.jpg με scale_factor = 0,2 για να αποφύγω μεγάλους χρόνους εκτέλεσης.

2.3.1 ΑΠΟΤΕΛΕΣΜΑΤΑ

Εικόνα 6 : Η τελική εικόνα που προέκυψε από περιστροφή της εικόνας εισόδου κατά 54◦ x π/180◦ rads

Εικόνα 7 : Η τελική εικόνα που προέκυψε από περιστροφή της εικόνας εισόδου κατά

213◦ x π/180◦ rads