Όραση Υπολογιστών

Εργασία 1

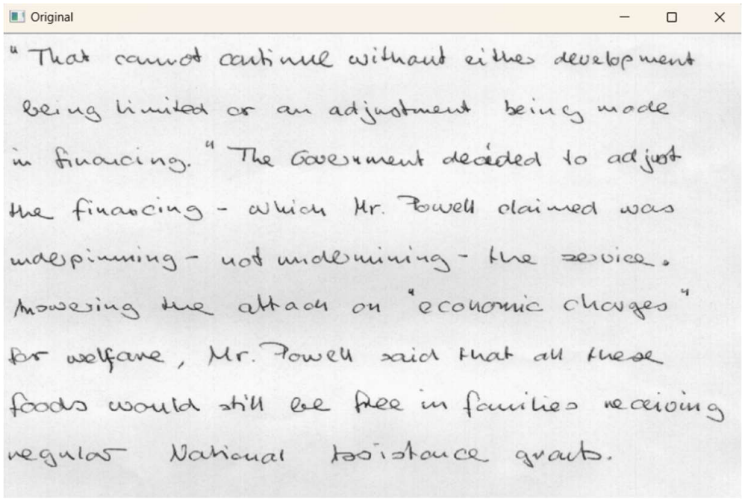
Ο παραδοτέος κώδικας βρίσκεται μέσα σε δύο αρχεία *.py*· το *line_detection.py*, μέσα στο οποίο βρίσκεται η "main" της εφαρμογής, και το *functions.py*, μέσα στο οποίο έχουν οριστεί επιπλέον συναρτήσεις που χρησιμοποιούνται στο κύριο πρόγραμμα. Επειδή το τελευταίο ψηφίο του ΑΕΜ είναι το "5", η εφαρμογή επεξεργάζεται την φωτογραφία *5.png*.

Ο κώδικας του *line_detection.py* χωρίζεται σε δύο μέρη· το κομμάτι επεξεργασίας της πρωτότυπης εικόνας (γραμμές 9-100) και το κομμάτι επεξεργασίας της αρχικής εικόνας αφού έχει εισαχθεί θόρυβος (γραμμές 102-207). Και στις δύο επεξεργασίες, η εικόνα έχει μετατραπεί σε *grayscale*.

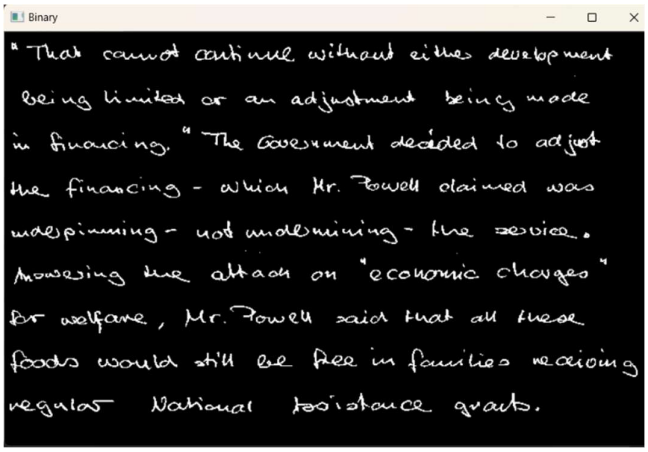
Επεξεργασία πρωτότυπης εικόνας

- **Thresholding**

Προκειμένου να γίνουν σωστά όλες οι διεργασίες, πρέπει πρώτα η εικόνα να μετατραπεί σε δυαδική εικόνα, δηλαδή όλα τα pixels να έχουν είτε την τιμή 0 (μαύρο) είτε την τιμή 255 (λευκό). Αυτό επιτυγχάνεται με την συνάρτηση *cv2.threshold()* (γραμμή 19), η οποία, εφόσον έχουμε τον τύπο *cv2.THRESH_BINARY_INV*, θέτει την τιμή 255 στα pixel με τιμές μεγαλύτερη του κατωφλιού 205 και την τιμή 0 σε κάθε άλλη περίπτωση. Το αποτέλεσμα αυτής της συνάρτησης στην αρχική εικόνα (*εικόνα 1.α*) φαίνεται στην παρακάτω *εικόνα 2*.α. Η μετατροπή της εικόνας σε δυαδική είναι απαραίτητη, καθώς δεν μας είναι χρήσιμο να γνωρίζουμε αν και που υπάρχουν διαβαθμίσεις του γκρι.



Εικόνα 1.α: Η πρωτότυπη φωτογραφία (5.png)

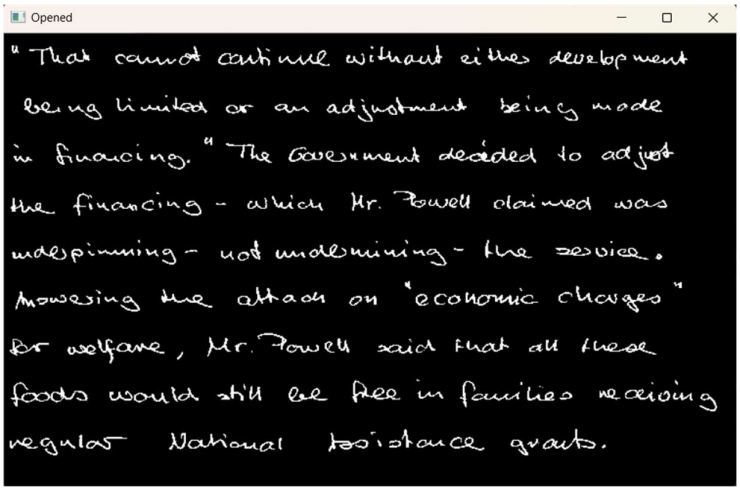


Εικόνα 2.α: Η πρωτότυπη εικόνα μετά από thresholding

Υπάρχει περίπτωση κάποια "αδέσποτα" pixels να απέκτησαν την τιμή 255 παρόλο που δεν ανήκουν στο κείμενο την αρχικής εικόνας. Αυτά τα pixels θα αποκτήσουν την τιμή 0 στο επόμενο βήμα.

- **Opening**

Όπως προαναφέρθηκε, στην δυαδική εικόνα μπορεί υπάρχουν κάποια ανεπιθύμητα λευκά pixels. Επιπλέον, παρόλο που μπορεί να μην φαίνεται εκ πρώτης όψεως, η εικόνα αυτή έχει ένα πολύ λεπτό λευκό περίγραμμα. Όλα αυτά τα επιπλέον λευκά εικονοστοιχεία θα δημιουργήσουν πρόβλημα στα επόμενα βήματα, εάν δεν επιλυθούν. Για την εξαφάνιση τους λοιπόν χρησιμοποιείται η μέθοδος Opening (γραμμές 25-29), δηλαδή ένα Erosion ακολουθούμενο από ένα Dilation. Λόγω της μικρής επιφάνειας των ανεπιθύμητων "σωμάτων", χρησιμοποιείται ένα μικρό τετραγωνικό kernel μεγέθους 3x3. Όπως φαίνεται και στην *εικόνα 3.α,* αυτή η μέθοδος εξαφάνισε πλήρως τα ανεπιθύμητα pixels. Υπάρχει περίπτωση να έχουν μαυρίσει και κάποια επιθυμητά άσπρα εικονοστοιχεία αλλά αυτά είναι ελάχιστα και η απόλυτη ακρίβεια δεν είναι απαραίτητη για τις επόμενες μεθόδους.

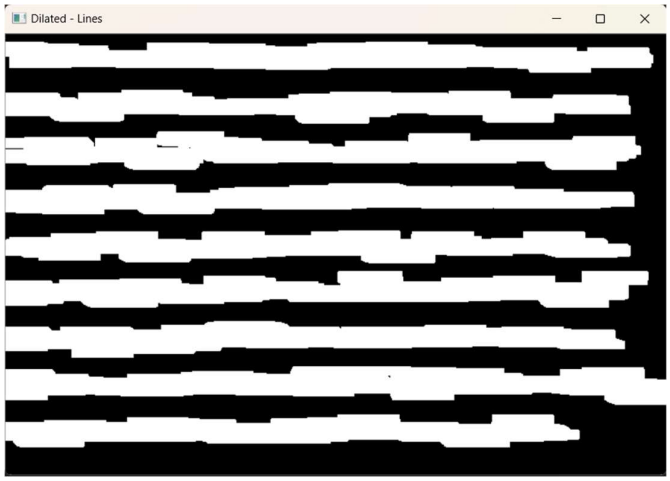


Εικόνα 3.α: Η δυαδική εικόνα μετά από εφαρμογή Opening

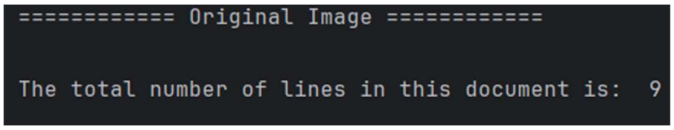
- **Line Detection**

Για την ανίχνευση των γραμμών, χρησιμοποιείται η συνάρτηση *cv2.connectedComponents()* (γραμμές 46-47). Αυτή συνάρτηση αποθηκεύει το πλήθος όλων των ενιαίων σωμάτων της εικόνας στην μεταβλητή *total_lines*. Επομένως, για να υπολογιστεί το πλήθος των γραμμών στις εικόνα, θα πρέπει όλες οι λέξεις και οι χαρακτήρες τις ίδιας γραμμής να ενωθούν μεταξύ τους, χωρίς να συγχωνευτούν με αυτές των άλλων γραμμών. Αυτό επιτυγχάνεται εφαρμόζοντας Dilation με ορθογώνιο παραλληλόγραμμο kernel (γραμμές 37-40). Προκειμένου να έχουμε ένα component ανά γραμμή, χωρίς να παρεμβάλλονται τα components διαφορετικών γραμμών μεταξύ τους, πρέπει ο πίνακας του δομικού στοιχείου να έχει πολύ μεγάλο αριθμό στηλών αλλά σχετικά μικρό αριθμό γραμμών (γραμμή 39). Τηρώντας αυτές τις προδιαγραφές παράγεται η *εικόνα 4.α*, η οποία αν εισαχθεί στην *cv2.connectedComponents()*, μας δίνει με ακρίβεια των αριθμό των γραμμών, όπως φαίνεται από το αποτέλεσμα της εκτύπωσης της γραμμής 50 (*εικόνα 5.α*).

Σχόλιο: Με την γραμμή 50 δεν εκτυπώνεται η *total_lines* αλλά η *total_lines – 1*. Ο λόγος που επιλέχθηκε να εκτυπωθεί αυτή η τιμή είναι επειδή η *cv2.connectedComponents()* θεωρεί ως ενιαίο αντικείμενο και το background της εικόνας. Άρα η *total_lines* περιλαμβάνει όχι μόνο τις γραμμές αλλά και το background.



Εικόνα 4.α: Ενοποίηση λέξεων γραμμών μέσω Dilation



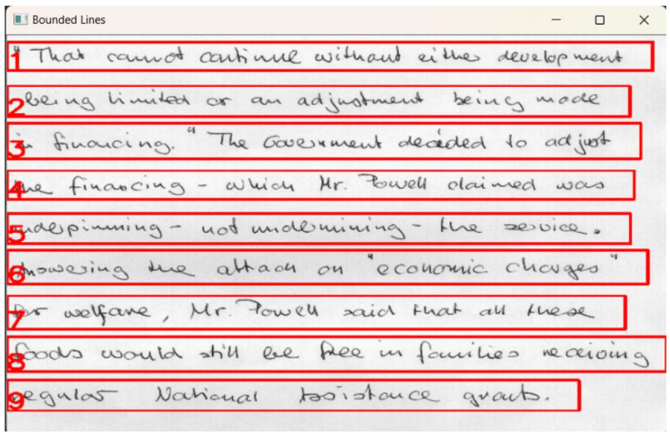
Εικόνα 5.α: Το αποτέλεσμα της εκτύπωσης total_lines – 1 (γραμμή 50)

- **Bounding Box**

Χρησιμοποιώντας τα συσσωματώματα/γραμμές της *εικόνας 4.α* μπορούμε να σχεδιάσουμε περιβάλλοντα κουτιά γύρω από κάθε γραμμή. Αυτό επιτυγχάνεται στις γραμμές 53-61. Πρώτα υπολογίζουμε όλα τα περιγράμματα της εικόνας μέσω της συνάρτησης *cv2.findContours()* (γραμμή 54) και τα αποθηκεύουμε στην μεταβλητή

contours. Ο τύπος *cv2.RETR_EXTERNAL* σημαίνει ότι για τον υπολογισμό των περιγραμμάτων χρησιμοποιούνται οι ακραίες/εξωτερικές συντεταγμένες. Έπειτα, μέσω της for loop (γραμμές 57-61), διατρέχουμε τόσα στοιχεία της *contours*, δηλαδή περιγράμματα, όσες είναι οι γραμμές του εγγράφου και για καθένα από αυτά σχεδιάζουμε ένα bounding box (γραμμή 59) και έναν μοναδικό αύξων αριθμό (γραμμές 60-61) πάνω στην αρχική εικόνα αξιοποιώντας τις συντεταγμένες της αντίστοιχης *contour* (γραμμή 58). Το αποτέλεσμα αυτής της διαδικασίας φαίνεται στην *εικόνα 6.α*.

Σχόλιο: Η *contour* αποθηκεύει τα περιγράμματα των γραμμών με την αντίθετη από την κανονική σειρά. Δηλαδή το *contour[0]* αναφέρεται στην τελευταία γραμμή του εγγράφου, το *contour[1]* στην προ-τελευταία, κ.ο.κ. Για αυτό και βλέπουμε στην γραμμή 58 το *contours[total_lines - 2 – cntr]*, έτσι ώστε η loop να διατρέχει τα περιγράμματα με την ίδια σειρά που εμφανίζονται οι γραμμές στο έγγραφο.



Εικόνα 6.α: Η αρχική εικόνα με τα bounding boxes και τους αύξοντες αριθμούς

Από την παραπάνω εικόνα φαίνεται ότι αυτή η μέθοδος επιτυγχάνει αποτελεσματικά να προσδιορίσει τις γραμμές του εγγράφου. Ωστόσο, όπως φαίνεται από τα κενά στην δεξιά μεριά του κάθε πλαισίου, δεν σχεδιάζει αυστηρά τα όρια της κάθε γραμμής. Αυτό συμβαίνει λόγω του μεγάλου αριθμού στηλών του structuring element. Οι επιλεγμένες διαστάσεις του μας εξασφαλίζουν ότι όλες οι λέξεις και σύμβολα της κάθε γραμμής θα συγχωνευθούν και θα πλαισιωθούν με επιτυχία, αλλά καταλαμβάνοντας περισσότερο από τον πραγματικό χώρο στην x διάσταση.

- **Word Detection**

Για την ανίχνευση των λέξεων σε κάθε γραμμή (γραμμές 63-73) χρησιμοποιείται η ίδια διαδικασία που εφαρμόστηκε στην ανίχνευση γραμμών αλλά με δύο διαφορές. Η πρώτη είναι ότι απομονώνουμε την κάθε γραμμή σε μία νέα εικόνα *img_line* (γραμμή 65), έτσι ώστε να μπορούμε να κάνουμε τους υπολογισμούς χωρίς να επηρεαζόμαστε από τις υπόλοιπες γραμμές. Οι συντεταγμένες των γραμμών υπολογίζονται όπως είδαμε στην γραμμή 58. Η δεύτερη είναι ότι στο Dilation (γραμμές 69-70) το δομικό στοιχείο είναι πάλι ορθογώνιο παραλληλόγραμμο, αλλά οι γραμμές του είναι περισσότερες από τις στήλες του και οι στήλες του πολύ λιγότερες σε σχέση εκείνες του kernel του line detection. Αυτές οι διαστάσεις του δομικού στοιχείου επιλέχθηκαν

ώστε να απομονώνεται η κάθε λέξη, αλλά ταυτόχρονα και για να συγχωνεύεται με τα σημεία στίξης που την περιβάλλουν. Το αποτέλεσμα της ανίχνευσης λέξεων κάθε γραμμής εκτυπώνεται μέσω της γραμμής 89.

- **Area Calculation**

Ένας επιπλέον υπολογισμός που πρέπει να υλοποιηθεί για κάθε γραμμή είναι ο αριθμός των εικονοστοιχείων που ανήκουν σε γράμμα και όχι σε φύλλο, δηλαδή η επιφάνεια της γραμμής. Ο υπολογισμός αυτός πραγματοποιείται μέσω ενός white pixel counter (γραμμές 75-82) για την αντίστοιχη απομονωμένη γραμμή (*img_line*). Αυτός μετρητής σκανάρει διαδοχικά όλα τα pixel της εικόνας και αυξάνεται κατά 1 κάθε φορά που διαβάζει ένα λευκό pixel. Η τιμή του μετρητή στο τέλος της υλοποίησης, αποτελεί την επιφάνεια της γραμμής (σε pixels) και τυπώνεται μέσω της γραμμής 87.

Αν και αυτή η μέθοδος είναι αποτελεσματική, υπάρχουν οι εξής προβληματισμοί. Το μέγεθος της εικόνας της γραμμής που πρέπει να σαρωθεί είναι σχετικά μικρό, οπότε ο υπολογισμός δεν καταλαμβάνει πολύ χρόνο. Ωστόσο, για μια μεγαλύτερη γραμμή μπορεί ο χρόνος υλοποίησης να ήταν μεγάλος, καθώς ο κώδικας σαρώνει διαδοχικά όλα τα pixels της εικόνας, κάτι που μπορεί να καθιστά την μέθοδο, αν και αποτελεσματική, μη αποδοτική. Επιπλέον, η εικόνα η οποία σαρώνεται είναι η αποκοπή της εικόνας *img_opened*. Σε αυτή την εικόνα έχουν μαυρίσει τα ανεπιθύμητα εικονοστοιχεία αλλά υπάρχει περίπτωση να έχουν μαυρίσει και κάποια επιθυμητά. Εκτός από αυτό, προηγουμένως αυτή η εικόνα είχε υποστεί thresholding, οπότε κάποια pixel γραμμάτων μπορεί να ήταν κάτω από το κατώφλι και έτσι να μην πήραν την τιμή 255. Επομένως, αυτή η μέθοδος μπορεί να μην είναι ακριβής, λόγω της απώλειας pixel γραμμάτων κατά τις προηγούμενες επεξεργασίες της εικόνας.

- **Line Specs**

Στις γραμμές 84-90 εκτυπώνονται όλα τα χαρακτηριστικά τις κάθε γραμμές που αναζητούμε. Πιο συγκεκριμένα, όπως προαναφέρθηκε, οι γραμμές 87 και 89 τυπώνουν την επιφάνεια της γραμμής και τον αριθμό των λέξεων αντίστοιχα, ενώ η γραμμή 88 υπολογίζει και τυπώνει την επιφάνεια του αντίστοιχου bounding box, χρησιμοποιώντας τις συντεταγμένες εξόδου της συνάρτησης *cv2.boundingRect()* (γραμμή 58).

Μετά την ολοκλήρωση της εκτέλεσης της for-loop, θα έχουν τυπωθεί τα αποτελέσματα της *εικόνας 7.α*. Παρατηρούμε ότι έχει υπολογιστεί μια παραπάνω και μία λιγότερη λέξη για τα *Region 6* και *7* αντίστοιχα. Αυτό πιθανόν δηλώνει ότι, στην πρώτη περίπτωση, ένα σημείο στίξης παρέμεινε μόνο του, ενώ στην δεύτερη, δύο λέξεις δεν κατάφεραν να απομονωθούν. Αυτή η αστοχία πηγάζει, κατά ένα βαθμό από την επιλογή των διαστάσεων του δομικού στοιχείου (γραμμή 69), αλλά κατά ένα μεγαλύτερο βαθμό από το μεταβλητό κενό ανάμεσα στις λέξεις και στα σημεία στίξης στην πρωτότυπη εικόνα. Συμπεραίνουμε λοιπόν ότι για την συγκεκριμένη εφαρμογή, η μέθοδος για την ανίχνευση των λέξεων δεν είναι τόσο αποτελεσματική.

<u>Σχόλιο</u>: Οι διαστάσεις του δομικού στοιχείου επιλέχθηκαν μέσω δοκιμής. Αν και δεν οδήγησαν στον ακριβή εντοπισμό λέξεων, ήταν πιο κοντά στο επιθυμητό αποτέλεσμα, για αυτό και επιλέχθηκαν.

```
========= Region 1 =========
Area (px):  22476
Bounding Box Area(px):  244680
Number of Words:  6

========= Region 2 =========
Area (px):  18800
Bounding Box Area(px):  255450
Number of Words:  7

========= Region 3 =========
Area (px):  21986
Bounding Box Area(px):  298000
Number of Words:  7

========= Region 4 =========
Area (px):  18034
Bounding Box Area(px):  239459
Number of Words:  7

========= Region 5 =========
Area (px):  19364
Bounding Box Area(px):  247842
Number of Words:  5

========= Region 6 =========
Area (px):  20845
Bounding Box Area(px):  291168
Number of Words:  7

========= Region 7 =========
Area (px):  19722
Bounding Box Area(px):  274950
Number of Words:  7

========= Region 8 =========
Area (px):  23063
Bounding Box Area(px):  308136
Number of Words:  8

========= Region 9 =========
Area (px):  18530
Bounding Box Area(px):  232845
Number of Words:  4
```

Εικόνα 7.α: Τα υπολογισμένα στοιχεία της κάθε γραμμής

Τέλος μέσω της γραμμής 98 αποθηκεύεται η *img_bounded* (*εικόνα 6.α*) ως *5_bounded_og.png*.

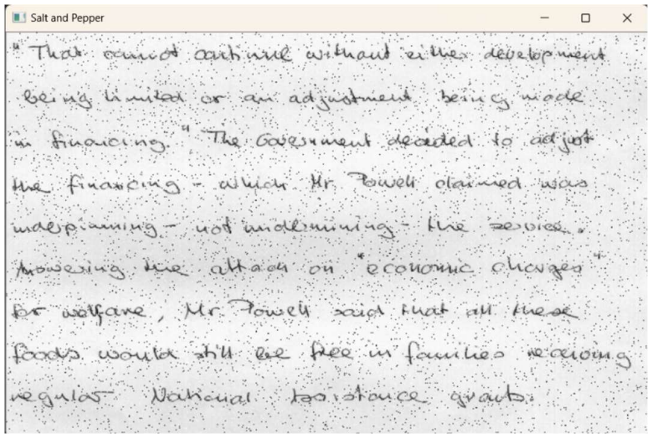
<u>Επεξεργασία εικόνας με θόρυβο</u>

Η μεθοδολογία για την εύρεση των ζητούμενων για την εικόνα με θόρυβο (γραμμές 102-207) είναι ακριβώς η ίδια με την αντίστοιχη χωρίς θόρυβο. Υπάρχουν μόνο δύο επιπλέον αρχικές διαδικασίες· η διαδικασία εισαγωγής salt and pepper θορύβου και η διαδικασία denoising μέσω median φίλτρου.

- **Salt and Pepper noise**

Η εισαγωγή του θορύβου γίνεται στην γραμμή 108 αξιοποιώντας μια από τις δύο συναρτήσεις που ορίστηκαν στο αρχείο *functions.py*, την *salt_pepper_noise()* (*functions.py*: γραμμές 6-35). Η συνάρτηση αυτή επιλέγει έναν τυχαίο αριθμό pixel (*functions.py*: γραμμή 14) και για καθένα από αυτά επιλέγει ένα τυχαίο pixel στην εικόνα (*line_detection.py*: γραμμές 16-21) στο οποίο δίνει την τιμή 255 (λευκό) για την εισαγωγή salt noise. Η ίδια ακριβώς διαδικασία εκτελείται μία ακόμη φορά αλλά, αντί για 255, δίνει την τιμή 0 (μαύρο) για την εισαγωγή pepper noise.

Εισάγοντας την αρχική εικόνα στην συνάρτηση προκύπτει η *img_noise*:



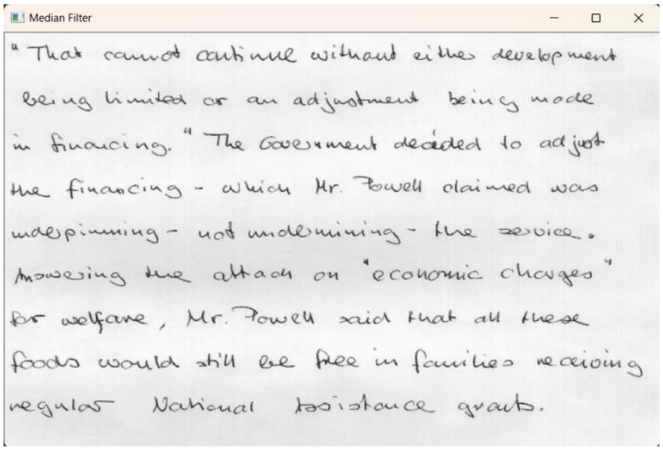
Εικόνα 1.β: Η αρχική εικόνα μετά την εισαγωγή Salt and Pepper noise

- **Median Filter Denoising**

Πριν κάνουμε όλους τους γνωστούς υπολογισμούς πρέπει η *img_noise* να απαλλαχθεί από τον θόρυβο εισάγοντας την σε ένα φίλτρο. Το φίλτρο που δρα πολύ αποτελεσματικά έναντι του salt and pepper θορύβου είναι το Median φίλτρο. Η εισαγωγή αυτή υλοποιείται στην γραμμή 119 μέσω της συνάρτησης *median_filter()*, η οποία είναι η δεύτερη από τις δύο συναρτήσεις που ορίζονται στο *functions.py* αρχείο (*functions.py*: γραμμές 38-65).

Η *median_filter()* σαρώνει διαδοχικά όλα τα μη εξωτερικά εικονοστοιχεία της εικόνας εισόδου και αντικαθιστά το καθένα από αυτά με την median τιμή από όλες τις γειτονικές του, συμπεριλαμβάνοντας και το ίδιο (3x3 παράθυρο).

Εισάγωντας την *img_noise* στην συνάρτηση προκύπτει η *img_denoised:*



Εικόνα 2.β: Η denoised μέσω median φίλτρου εικόνα

Παρατηρούμε ότι η *img_denoised* είναι πανομοιότυπη με την αρχική εικόνα. Εδώ φανερώνεται η αποτελεσματικότητα του Median φίλτρου, όχι μόνο στην περίπτωση του salt and pepper θορύβου, αλλά πόσο μάλλον σε μια εικόνα χωρίς πολλές λεπτομέρειες και τεράστιες χρωματικές διαφορές (μεταξύ των γραμμάτων). Το μόνο ελάττωμα είναι ο υπολογιστικός φόρτος. Η συνάρτηση διατρέχει σχεδόν όλα τα pixel

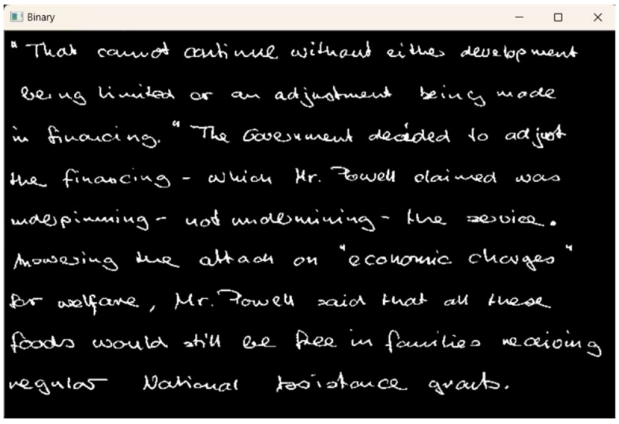
της εικόνας και για το καθένα από αυτά διαβάζει 9 τιμές και τις κάνει sort (*functions.py*: γραμμή 60). Για αυτόν τον λόγο και χρειάζεται να περάσουν αρκετά δευτερόλεπτα (10-15 δευτερόλεπτα) προτού παραχθεί η νέα εικόνα από την στιγμή που καλείται η συνάρτηση. Παρόλα αυτά, η αποτελεσματικότητα του αλγορίθμου σε συνδυασμό με το γεγονός ότι δεν υπάρχει φανερή αλλοίωση της εικόνας εισόδου υπερνικούν τον αυξημένο χρόνο υλοποίησης.

- **Αποτελέσματα εκτέλεσης**

Μετά από την εφαρμογή των 2 επιπλέον συναρτήσεων πάνω στην αρχική εικόνα, η μεθοδολογία είναι ακριβώς η ίδια με εκείνη της εικόνας χωρίς θόρυβο και οδηγεί στα παρακάτω αποτελέσματα.

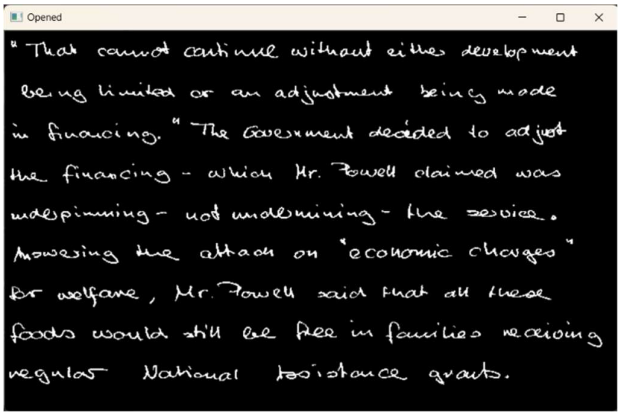
Σχόλιο: Όποια από τα αποτελέσματα είναι ίδια/σχεδόν ίδια με εκείνα της προηγούμενης περίπτωσης δεν σχολιάζονται. Σχολιάζεται μόνο οτιδήποτε διαφορετικό.

Thresholding:



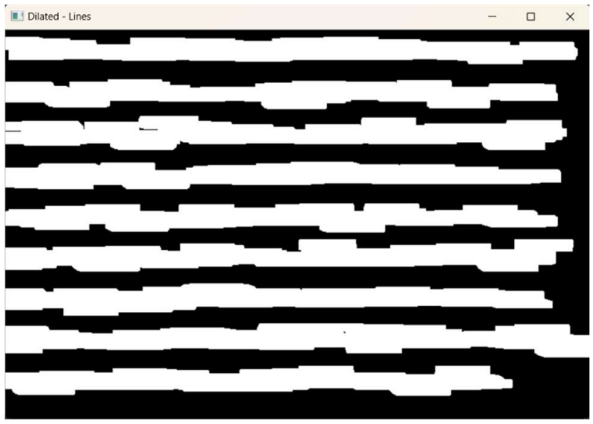
Εικόνα 3.β: Η denoised εικόνα μετά από thresholding

Opening:



Εικόνα 4.β: : Η δυαδική εικόνα μετά από εφαρμογή Opening

Line detection:



Εικόνα 5.β: Ενοποίηση λέξεων γραμμών μέσω Dilation

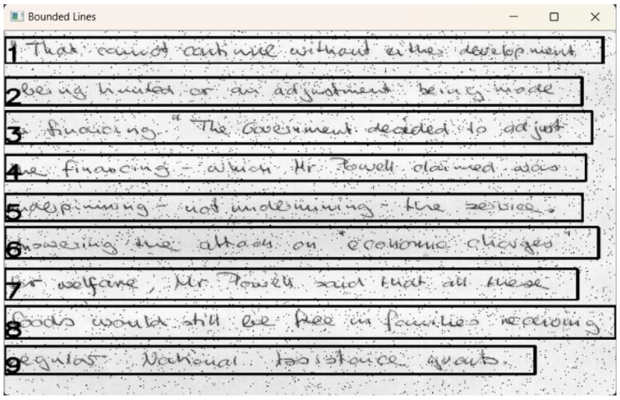
```
============ Noisy Image ============

Filtering...

The total number of lines in this document is:  9
```

Εικόνα 6.β: Το αποτέλεσμα της εκτύπωσης total_lines – 1 (γραμμή 158)

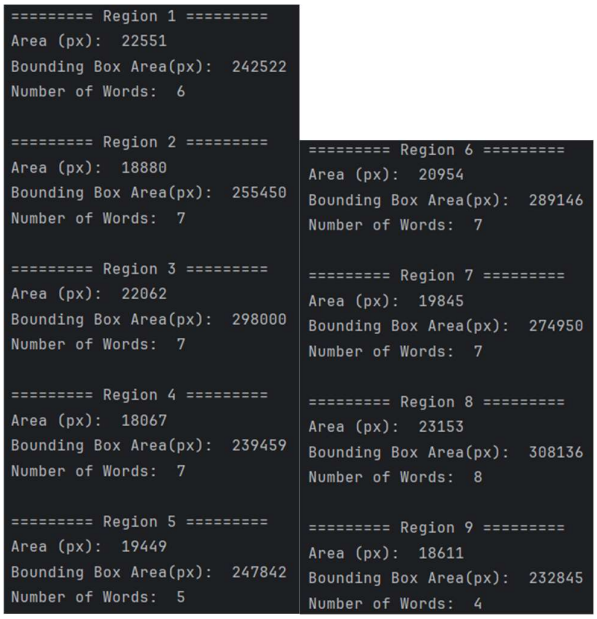
Bounding Box:



Εικόνα 7.β: Η noisy εικόνα με τα bounding boxes και τους αύξοντες αριθμούς

Παρατηρούμε στην *εικόνα 7.β* ότι τα περιβάλλοντα κουτιά και οι αύξοντες αριθμοί έχουν μαύρο χρώμα αντί για κόκκινο όπως ήταν στην πρώτη περίπτωση. Αυτό συμβαίνει, διότι η αρχική εικόνα με τον θόρυβο είναι grayscaled, οπότε προβάλλονται μόνο διαβαθμίσεις του γκρι.

Τα αποτελέσματα των μετρήσεων στις *εικόνες 7.α* και *8.β* αποκλίνουν από πολύ λίγο εώς και καθόλου. Αυτή η μικρή απόκλιση οφείλεται στην ελάχιστη αλλοίωση (η οποία δεν φαίνεται με γυμνό μάτι) της αρχικής εικόνας λόγω της εισαγωγής και της απομάκρυνσης (μέσω median φίλτρου) του θορύβου.



Εικόνα 8.β: Τα υπολογισμένα στοιχεία της κάθε γραμμής

Τέλος μέσω της γραμμής 205 αποθηκεύεται η *img_bounded* (*εικόνα 7.β*) ως *5_bounded_noise.png*.

Σύγκριση επιδόσεων εφαρμογής με και χωρίς θόρυβο

Συγκρίνοντας όλα τα προαναφερθέντα αποτελέσματα παρατηρούμε ότι η ύπαρξη θορύβου (salt and pepper) στην εικόνα δεν επηρεάζει σημαντικά την επίδοση της εφαρμογής. Ως προς την ανίχνευση των γραμμών και των λέξεων προκύπτουν τα ίδια αποτελέσματα, ενώ τα άλλα μεγέθη αποκλίνουν από λίγο εώς καθόλου, κάτι το οποίο είναι αναμενόμενο σε αυτές τις περιπτώσεις. Οι παραγόμενες εικόνες σε όλα τα βήματα είναι πανομοιότυπες με διαφορές που δεν φαίνονται, αν όχι με γυμνό μάτι, τουλάχιστον όχι με μεγάλη ευκολία, ενώ η πρωτότυπη και η denoised εικόνα είναι πρακτικά οι ίδιες. Η μόνη συγκρίσιμη διαφορά μεταξύ των δύο περιπτώσεων είναι ο χρόνος υλοποίησης. Η απομάκρυνση του θορύβου με median filter αυξάνει τον χρόνο υλοποίησης με αποτέλεσμα να ολοκληρώνει η εφαρμογή τις λειτουργείες της σε διαφορετικούς χρόνους στην κάθε περίπτωση. Ωστόσο αυτή η χρονική διαφορά δεν είναι σημαντικά μεγάλη και μάλιστα είναι αναμενόμενη, καθώς εκτελούνται επιπλέον διεργασίες στην περίπτωση του θορύβου.

Πηγές

- *salt_pepper_noise()*: Add a "salt and pepper" noise to an image with Python - GeeksforGeeks
- *median_filter()*: Spatial Filters - Averaging filter and Median filter in Image Processing - GeeksforGeeks