ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΠΜΣ «ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ & ΥΠΗΡΕΣΙΕΣ» - ΜΕΓΑΛΑ ΔΕΔΟΜΕΝΑ & ΑΝΑΛΥΤΙΚΗ

«ΑΝΑΠΤΥΞΗ ΔΙΑΔΙΚΤΥΑΚΗΣ ΕΦΑΡΜΟΓΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ ΕΙΚΟΝΑΣ ΒΑΣΙΣΜΕΝΗ ΣΕ ΚΩΔΙΚΑ MATLAB»

ΠΑΝΤΕΛΗΣ ΣΒΕΝΤΖΟΥΡΗΣ – ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ : ΜΕ1614

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Δ. ΚΥΡΙΑΖΗΣ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1. ΕΙΣΑΓΩΓΗ

Στη σημερινή εποχή, η επεξεργασία ψηφιακών εικόνων, με τη βοήθεια της τεχνολογίας, έχει απλοποιηθεί σε σημαντικό βαθμό. Αποτέλεσμα αυτής της υλοποίησης είναι να υπάρχουν διάφορα πεδία εφαρμογής αυτής της τεχνικής επεξεργασίας. Κάποια από αυτά τα πεδία εφαρμογής μπορεί να είναι η Ιατρική, το φωτομοντάζ, ο κινηματογράφος, κ.α.

Ένα παράδειγμα μίας τέτοιας επεξεργασίας εικόνας θα μπορούσε να είναι η αφαίρεση μίας τρίχας από το δέρμα χωρίς να γίνει εμφανής η αλλαγή του δέρματος στο μάτι του θεατή. Για να επιτευχθεί αυτή η επεξεργασία θα πρέπει να υλοποιηθούν κάποιοι αλγόριθμοι επεξεργασίας εικόνας, οι οποίοι θα δέχονται ως είσοδο την εικόνα αλλά σε μορφή πίνακα (matrix) και εν συνεχεία με τη χρήση διάφορων πράξεων και μετασχηματισμών θα παράγουν ως έξοδο πάλι ένα διάνυσμα το οποίο μπορεί να μετατραπεί στη ζητούμενη εικόνα.

Επιπρόσθετα, στην εποχή μας όπου πλέον το Internet χρησιμοποιείται εκτεταμένα τόσο για βασικές ανάγκες όσο και για ανάγκες προστιθέμενης αξίας, κρίνεται σκόπιμη η μεταφορά εφαρμογών που μέχρι πρότινος χρησιμοποιούνταν μόνο σε κλειστές πλατφόρμες (π.χ. εφαρμογές γραμμένες αποκλειστικά σε περιβάλλοντα Microsoft Windows) σε περιβάλλον Internet (web apps) και μάλιστα σε τεχνολογίες που είναι ανοικτές (δηλαδή εκτέλεση του κώδικα των εφαρμογών σε διαφορετικά περιβάλλοντα και συνηθώς ανοικτά, όπως π.χ. Linux servers).

Ειδικά, για εφαρμογές που αναπτύσσονται σε κώδικα Matlab και αφορούν επεξεργασία εικόνας, πλέον υπάρχουν αντίστοιχες εναλλακτικές λύσεις ανοικτού κώδικα, όπως π.χ. Octave και το Open Computer Vision (OpenCV). Σε συνδυασμό ειδικά με τη μεγάλη διείσδυση του Internet, αλλά και την εξάπλωση της γλώσσας προγραμματισμού Python σε τεχνικές ανάλυσης δεδομένων, το OpenCV προκρίνεται ως η προτεινόμενη εναλλακτική για την ανάπτυξη μίας web εφαρμογής ισοδύναμης σε κώδικα Matlab.

Σε συνδυασμό μάλιστα με την δυνατότητα της Python να χρησιμοποιείται και για ανάπτυξη εφαρμογών (ρύθμιση web server, δημιουργία δυναμικού περιεχομένου ιστοσελίδων), προτείνεται η χρήση ενός λογισμικού που ονομάζεται Flask, το οποίο είναι μία lightweighted έκδοση ενός web server που έχει αναπτυχθεί σε γλώσσα προγραμματισμού Python.

Ακόμη, στα οφέλη χρήσης αυτών των εργαλείων λογισμικού προστίθεται και η διαλειτουργικότητα ανεξαρτήτως χρήσης λειτουργικού συστήματος (cross-platform interoperability), το οποίο σημαίνει πρακτικά ότι η συγκεκριμένη υλοποίηση που θα περιγραφεί στη συνέχεια, μπορεί να αναπτυχθεί σε οποιονδήποτε υπολογιστή ανεξαρτήτως λειτουργικού συστήματος (Windows, Linux, MacOS) καθώς και να χρησιμοποιηθεί από οποιαδήποτε συσκευή έχει πρόσβαση σε web εφαρμογές (PC, κινητά τηλέφωνα).

Σκοπός της παρούσας εργασίας είναι η συνοπτική παρουσίαση μίας εφαρμογής web η οποία θα αντικαταστήσει μέρος λειτουργικότητας μιας υφιστάμενης εφαρμογής υλοποιημένης σε κώδικα Matlab.

Στην Ενότητα 2 «Σχεδιασμός και Ανάπτυξη Εφαρμογής» παρουσιάζεται αναλυτικά η υφιστάμενη εφαρμογή σε Matlab καθώς και ποια τμήματα της θα μεταφερθούν στην web εφαρμογή, καθώς και η χρήση όλων των απαιτούμενων για αυτή τη διαιδκασία τεχνολογιών.

Στην Ενότητα 3 παρουσιάζεται αναλυτική παρουσίαση και εκτέλεση της web εφαρμογής καθώς και κάποια ενδεικτικά αποτελέσματά της. Επίσης παρουσιάζονται αναλυτικά και τα βήματα για την εγκατάσταση του αντίστοιχου περιβάλλοντος της web εφαρμογής σε οποιοδήποτε περιβάλλον.

Στην Ενότητα 4 παρουσιάζονται ποια θέματα δεν καλύπτονται καθώς και πιθανές μελλοντικές βελτιώσεις της web εφαρμογής, ως συμπεράσματα κατά την εκπόνηση της συγκεκριμένης εργασίας.

Στην Ενότητα 5 παρουσιάζονται αναλυτικά τα τμήματα του κώδικα που χρησιμοποιήθηκαν, τόσο σε Matlab της υφιστάμενης εφαρμογής, όσο και σε Python, HTML, JavaScript και CSS της νέας web εφαρμογής.

Στην Ενότητα 6 παρουσιάζεται, τέλος, η βιβλιογραφία που χρησιμοποιήθηκε τόσο για τη θεωρητική όσο και για την τεχνική θεμελίωση της παρούσας εργασίας.

2. ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ

2.1 Μεταφορά της υφιστάμενης Matlab εφαρμογής σε web app

Η συγκεκριμένη Matlab εφαρμογή ενώ έχει τα παρακάτω πλεονεκτήματα:

- Ταχύτητα εκτέλεσης
- Ποικιλία εμφάνισης γραφικών παραστάσεων
- Εύκολη παραμετροποίηση

Εντούτοις, έχει και δύο βασικά μειονεκτήματα τα οποία έρχεται να λύσει η μεταφορά της σε web εφαρμογή (web app):

- Εκτέλεση μόνο στο κλειστό περιβάλλον του Matlab (που συνεπάγεται αποκλειστικά χρήση PC με Microsoft Windows και αντίστοιχη άδεια χρήσης λογισμικού Matlab)
- Πρόσβαση στην εφαρμογή μόνο από PC και από καμία άλλη συσκευή.

Για αυτό λοιπόν, η μεταφορά της λειτουργικότητας της Matlab εφαρμογής σε web περιβάλλον έχει τα εξής πλεονεκτήματα:

- Πρόσβαση της εφαρμογής από κάθε συσκευή που έχει πρόσβαση σε web εφαρμογές (PC ανεξαρτήτως λειτουργικού συστήματος, Mac, κινητά τηλέφωνα, tablets)
- Η εφαρμογή είναι εγκατεστήμενη σε έναν server και είναι προσβάσιμη από σύνολο διαφορετικών συσκευών που έχουν συνδεθεί με τον server μέσω HTTP πρωτοκόλλου.
- Απουσία ανάγκης πληρωμών license fees για το Matlab.
- Επιπλέον διαλειτουργικότητα και συμβατότητα με άλλα προγράμματα (π.χ. προσθήκη νέας λειτουργικότητας ή ενός νέου αλγορίθμου)

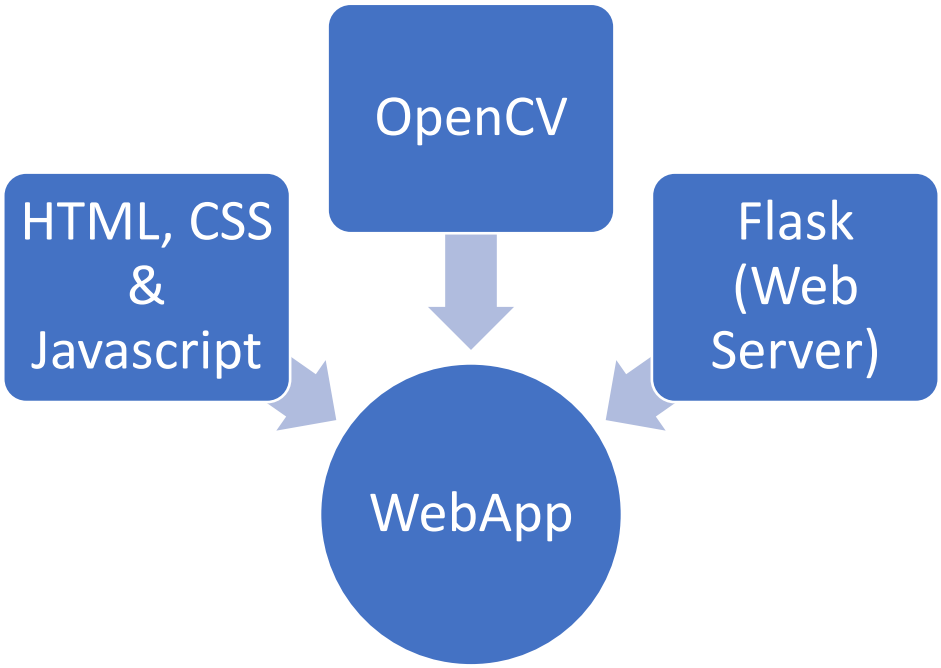
Η γλώσσα προγραμματισμού που επιλέχθηκε για την ανάπτυξη της νέας web εφαρμογής είναι η Python διότι περιέχει ένα μεγάλο σύνολο υφιστάμενων προγραμμάτων και βιβλιοθηκών για την ψηφιακή επεξεργασία εικόνων και επιπλέον μπορεί να χρησιμοποιηθεί και για την ανάπτυξη web εφαρμογών.

Πιο συγκεκριμένα, τα frameworks που θα χρησιμοποιηθούν είναι τα ακόλουθα:

- Open Comupter Vision (OpenCV): Είναι ένα σύνολο βιβλιοθηκών που υποστηρίζουν την πλειονότητα των λειτουργιών του Matlab όσον αφορά την ψηφιακής επεξεργασίας εικόνων και έχει αναπτυχθεί στις γλώσσες προγραμματισμού C++ και Python. Όλα τα επιμέρους στοιχεία της βιβλιοθήκης μπορούν να κληθούν ανεξάρτητα από κάθε εφαρμογή (εν προκειμένω υλοποιημένη σε Python).
- Flask: Είναι ένας lightweight web server υλοποιημένος σε Python όπου μπορεί να εξυπηρετήσει αιτήματα HTTP και να καλέσει προγράμματα γραμμένα σε Python.

Επίσης, η όλη εμφάνιση της εφαρμογής πλέον δεν αφορά υλοποίηση σε Matlab αλλά σε HTML, CSS και Javascript όπου πλέον είναι προσβάσιμη από πληθώρα συμβατών συσκευών.

Συνοπτικά στην παρακάτω εικόνα περιγράφεται η λειτουργικότητα της νέας εφαρμογής:



Εικόνα 1: Υλοποίηση εφαρμογής web με χρήση τεχνολογιών Python (OpenCV, Flask) και Web (HTML, CSS, JavaScript)

2.2 Χρήση της βιβλιοθήκης OpenCV

Η βιβλιοθήκη OpenCV προσφέρει μεγάλη ποικιλία λειτουργιών που χρειάζονται για την ψηφιακή επεξεργασία εικόνων (και όχι μόνο) . Μπορεί να χρησιμοποιηθεί με τις γλώσσες προγραμματισμού C++, Java, C#[i] και Python.

Για τις ανάγκες της παρούσας εργασίας, έχουν χρησιμοποιηθεί οι παρακάτω μέθοδοι του OpenCV:

- Εξομάλυνση (Smoothing)
- Μορφολογικοί μετασχηματισμοί (Morphological Transformations)
- Κλίση (Gradient)

Στη συνέχεια περιγράφονται συνοπτικά οι λειτουργίες που θα χρησιμοποιηθούν ανά κατηγορία:

2.2.1 Εξομάλυνση (smoothing)

Όσον αφορά την εξομάλυνση, μελετάμε 2 περιπτώσεις:

1) Η εφαρμογή φίλτρου ούτως ώστε να διαγραφεί ο επιπλέον θόρυβος της εικόνας. Το φίλτρο που εφαρμόζεται, βασίζεται σε έναν πυρήνα (kernel), όπου για τις ανάγκες της εργασίας επιλέχθηκε το φίλτρο με πυρήνα (πίνακας μορφής 3x3), το οποίο σημαίνει ότι γίνεται η πράξη της συνέλιξης (convolution) της αρχικής εικόνας μαζί με τον παρακάτω πυρήνα:

$$K = (1/9)\begin{matrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{matrix}$$

Η συγκεκριμένη πράξη ονομάζεται υπολογισμός κατά μέσο όρο (averaging), δηλαδή μετά την πράξη της συνέλιξης, βρίσκουμε το μέσο όρο των pixels και τα αντικαθιστούμε με το κεντρικό μέρος της εικόνας.

Στην Python, ο αντίστοιχος κώδικας είναι:

averaging_img = cv2.blur(initial_img,(3,3))

2) Η εφαρμογή φίλτρου για την αφαίρεση του προσθετικού Gaussian θορύβου (Gaussian Noise), με τη χρήση πυρήνα Gaussian[ii] , όπου αντιστοίχως με τα παραπάνω, ο αντίστοιχος κώδικας σε Python είναι:

Gaussian_averaging_img = cv2.GaussianBlur(initial_img,(3,3),0)

2.2.2 Μορφολογικοί μετασχηματισμοί (Morphological Transformations)

Όσον αφορά τους μορφολογικούς μετασχηματισμούς μιας εικόνας, εξετάζονται οι παρακάτω κατηγορίες:

1) Διάβρωση (erosion)

Η διάβρωση εικόνας προκύπτει μετά τη συνέλιξη της αρχικής εικόνας με το αντίστοιχο φίλτρο (όπως στις παραπάνω περιπτώσεις) . Στη συνέχεια ελέγχεται αν το κάθε pixel της αρχικής εικόνας και υπολογίζεται ως 1 αν όλα τα pixels του πυρήνα είναι 1 αλλιώς υπολογίζεται ως 0 (γίνεται δηλαδή «διάβρωση»).

Συνεπώς όλα τα pixels που βρίσκονται σε οριακές θέσεις, παραλείπονται, αναλόγως και τον πυρήνα που χρησιμοποιούμε κάθε φορά. Έτσι, η τελική εικόνα φαίνεται με τα λιγότερα δυνατά λευκά pixels.

Ο αντίστοιχος κώδικας σε Python είναι:

eroded_img = cv2.erode(initial_img,kernel,iterations=1)

όπου iterations ο αριθμός των επαναλήψεων του φίλτρου. (Εμπειρικά, μετά από πλήθος πάνω από 2 iterations η εικόνα τείνει να γίνει εντελώς μαύρη μιας και θα έχουν εξαλειφθεί όλα τα γειτονικά λευκά pixels)

2) Διεύρυνση (dilation)

Εδώ πρόκειται για το ακριβώς αντίθετο αποτέλεσμα από τη διάβρωση. Όσο αυξάνεται ο αριθμός των iterations, τόσο πιο λευκή γίνεται η εικόνα, διότι αντί να έχουμε αφαίρεση λευκών pixels, έχουμε προσθήκη.

Ο αντίστοιχος κώδικας σε Python είναι:

dilated_img = cv2.dilate(initial_img,kernel,iterations=1)

3) Άνοιγμα (opening)

Το άνοιγμα είναι το αποτέλεσμα αν εκτελέσουμε διαδοχικά διάβρωση και διεύρυνση μιας εικόνας. Είναι μία χρήσιμη λειτουργία για αφαίρεση του θορύβου.

Ο αντίστοιχος κώδικας σε Python είναι:

opening_img = cv2.morphologyEx(initial_img, cv2.MORPH_OPEN, kernel)

4) Κλείσιμο (closing)

Το κλείσιμο είναι ακριβώς το αντίστροφο από το άνοιγμα, δηλαδή το αποτέλεσμα αν εκτελέσουμε διαδοχικά διεύρυνση και διάβρωση της εικόνας. Είναι χρήσιμη λειτουργία για αφαίρεση θορύβου μέσα στο λευκό περίγραμμα της εικόνας.

Ο αντίστοιχος κώδικας σε Python είναι:

closing_img = cv2.morphologyEx(initial_img, cv2.MORPH_CLOSE, kernel)

5) Top Hat

Ο Top Hat μετασχηματισμός είναι η διαφορά μεταξύ της αρχικής εικόνας και του ανοίγματός της. Ο αντίστοιχος κώδικας σε Python είναι:

tophat_img = cv2.morphologyEx(initial_img, cv2.MORPH_TOPHAT, kernel)

6) Black Hat ή Bot Hat

Ο Black Hat ή Bot Hat μετασχηματισμός είναι η διαφορά μεταξύ της αρχικής εικόνας και του κλεισίματός της. Ο αντίστοιχος κώδικας σε Python είναι:

blackhat_img = cv2.morphologyEx(initial_img, cv2.MORPH_BLACKHAT, kernel)

2.2.3 Κλίση (Gradient)

Η βιβλιοθήκη του OpenCV παρέχει τριών ειδών φίλτρα (υψιπερατά ή highpass) για κλίση εικόνων (image gradient):

- Το Sobel[iii]
- Το Laplacian[iv]
- Και το Scharr[v]

Στην παρούσα εργασία θα ασχοληθούμε μόνο με τα φίλτρα Sobel και Laplacian

2.3 Αντιστοίχιση λειτουργιών OpenCV με Matlab

Η υπάρχουσα εφαρμογή που είναι υλοποιημένη σε κώδικα Matlab, έχει πολλά κοινά στοιχεία με την υλοποίηση σε OpenCV. Παρακάτω στον Πίνακα 1 μπορούμε να δούμε την αντιστοίχιση των λειτουργιών που υπάρχουν τόσο στο Matlab όσο και στο OpenCV. Κάποιες λειτουργίες του Matlab δεν είναι διαθέσιμες στο OpenCV και συνεπώς θα πρέπει είτε να αναπτυχθεί αντίστοιχος ισοδύναμος κώδικας σε Python και να προσαρτηθεί στην web εφαρμογή, είτε θα πρέπει να παραλειφθεί από τη νέα υλοποίηση της web εφαρμογής. Σημειώνεται ότι το πρόθεμα cv2 στη στήλη OPENCV(PYTHON) αφορά το package του OpenCV στην Python.

MATLAB	OPENCV (PYTHON)
imwrite()	cv2.imread()
strel('line')	Δεν υπάρχει ισοδύναμη μέθοδος
imbothat()	cv2.morphologyEx(initial_img, cv2.MORPH_BLACKHAT, kernel)
imadd(img1,img2)	img1 + img2
im2bw()	cv2.threshold()
imdilate()	cv2.dilate()
rgbimage(:,:,1)	img[:,:,0]
roifill()	roi.setTo(cv2.Scalar(blue,green,red))
rgb2gray(img)	cv2.imread(img_filename,cv2.IMREAD_GRAYSCALE)
wiener2()	Δεν υπάρχει ισοδύναμη μέθοδος. Υπάρχει όμως εναλλακτική υλοποίηση σε Python που περιγράφεται στη συνέχεια.
bwmorph()	Υπάρχουν ισοδύναμες μέθοδοι στην κατηγορία Morphological Transformations (Υποενότητα 2.3.2)
imclose()	cv2.morphologyEx(initial_img, cv2.MORPH_CLOSE, kernel)
imerode()	cv2.erode()
edge(img,'log')	cv2.Canny() με ελαφρώς διαφορετικά αποτελέσματα

2.4 Χρήση του λογισμικού Flask

Για να αναπτυχθεί web εφαρμογή σε Python και μάλιστα με απλή σχετικά εγκατάσταση web server επιλέχθηκε το πρόγραμμα Flask[vi], το οποίο είναι ένας απλός στην εγκατάσταση και συνάμα δυνατός web server για εξυπηρέτηση http requests και επιπλέον έχει πλήρη συμβατότητα με το OpenCV ούτως ώστε όλα τα αποτελέσματα εκτέλεσης του OpenCV να επιστρέφονται ως http responses, είτε ως στατικές σελίδες είτε ως api calls.

Ένα ακόμα πλεονέκτημα είναι η χρήση του σε Linux server και όχι μόνο σε Windows server, καθιστώντας το έτσι cross-plarform compatible.

2.5 Τεχνικοί περιορισμοί στην web εφαρμογή

Η μεταφορά της εφαρμογής Matlab σε webapp κατέστη δυνατή με τη μεταφορά της βασικής λειτουργικότητάς της, παρόλα αυτά υπάρχουν κάποιοι τεχνικοί περιορισμοί οι οποίοι περιγράφονται συνοπτικά παρακάτω:

- Η χρήση του Flask server σε συστήματα παραγωγής δεν προτείνεται αν πρόκειται για επεξεργασία μεγάλων σε μέγεθος εικόνων και παράλληλη χρήση από πολλούς χρήστες, διότι απαιτεί μεγαλύτερο ποσό μνήμης και επεξεργαστικής ισχύος, λόγω της χρήσης της βιβλιοθήκης OpenCV σε Python. Η ταχύτητα εκτέλεσης μπορεί να βελτιωθεί αν χρησιμοποιηθεί η αντίστοιχη έκδοση OpenCV σε C++ από κάποια native εφαρμογή και όχι web εφαρμογή.
- Για τις ανάγκες της παρούσας εργασίας ο Flask server έχει παραμετροποιηθεί για περιβάλλον ανάπτυξης (dev environment) και όχι για παραγωγικό, συνεπώς θα υπάρξει επιπλέον overhead.
- Η συγκεκριμένη εφαρμογή σε κάθε βήμα εκτέλεσης των αλγορίθμων αποθηκεύει στον σκληρό δίσκο τις ενδιάμεσες εικόνες, με αποτέλεσμα να υπάρχει επιπλέον I/O.
- Επειδή η αρχική σελίδα (entry point) της εφαρμογής είναι μία στατική html σελίδα, οι εικόνες που φορτώνονται ίσως αποθηκευτούν προσωρινά στην cache του browser, οπότε καλό είναι μετά την εμφάνιση των αποτελεσμάτων των αλγορίθμων να κλείνει και να ανοίγει ξανά το tab της σελίδας στον browser.
- Η εικόνα που εισάγεται στην εφαρμογή θα πρέπει να είναι σε συμπιεσμένη μορφή JPEG ή PNG, με αποτέλεσμα να χάνεται ήδη πληροφορία στην αρχή της επεξεργασίας.

3. ΕΚΤΕΛΕΣΗ ΕΦΑΡΜΟΓΗΣ – ΕΝΔΕΙΚΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

3.1 Οδηγίες εγκατάστασης περιβάλλοντος εφαρμογής

Όπως προαναφέρθηκε, για τη δημιουργία και εγκατάσταση της εφαρμογής θα χρησιμοποιηθούν τα εξής προγράμματα λογισμικού:

- OpenCV
- Flask

Στη συνέχεια παρατίθενται συνοπτικά οι οδηγίες εγκατάστασης σε περιβάλλον Ubuntu 20.04 LTS με επεξεργαστή Intel i7 x64, 8GB RAM:

1) Εγκαθιστούμε την Python3
2) Εγκαθιστούμε το pip3: *sudo apt install -y python3-pip*
3) Δημιουργούμε ένα εικονικό περιβάλλον εκτέλεσης (virtual environment - venv), μία λειτουργικότητα που προσφέρει η Python3 ούτως ώστε να εκτελούμε τα προγράμματα από έναν συγκεκριμένο φάκελο και να έχουμε όλα τα απαιτούμενα dependencies της εφαρμογής:
   sudo apt install python3-venv
   source venv/bin/activate
4) Εγκαθιστούμε το Flask μέσα στο εικονικό περιβάλλον εκτέλεσης: *pip install Flask*
5) Εγκαθιστούμε το OpenCV μέσα στο εικονικό περιβάλλον εκτέλεσης: *sudo apt-get install python-opencv*
6) Τέλος, εγκαθιστούμε τη βιβλιοθήκη numpy: *pip install numpy*

Πλέον είμαστε έτοιμοι με την αρχική εγκατάσταση του περιβάλλοντος που θα χρησιμοποιήσουμε για την υλοποίηση και εκτέλεση της νέας μας εφαρμογής.

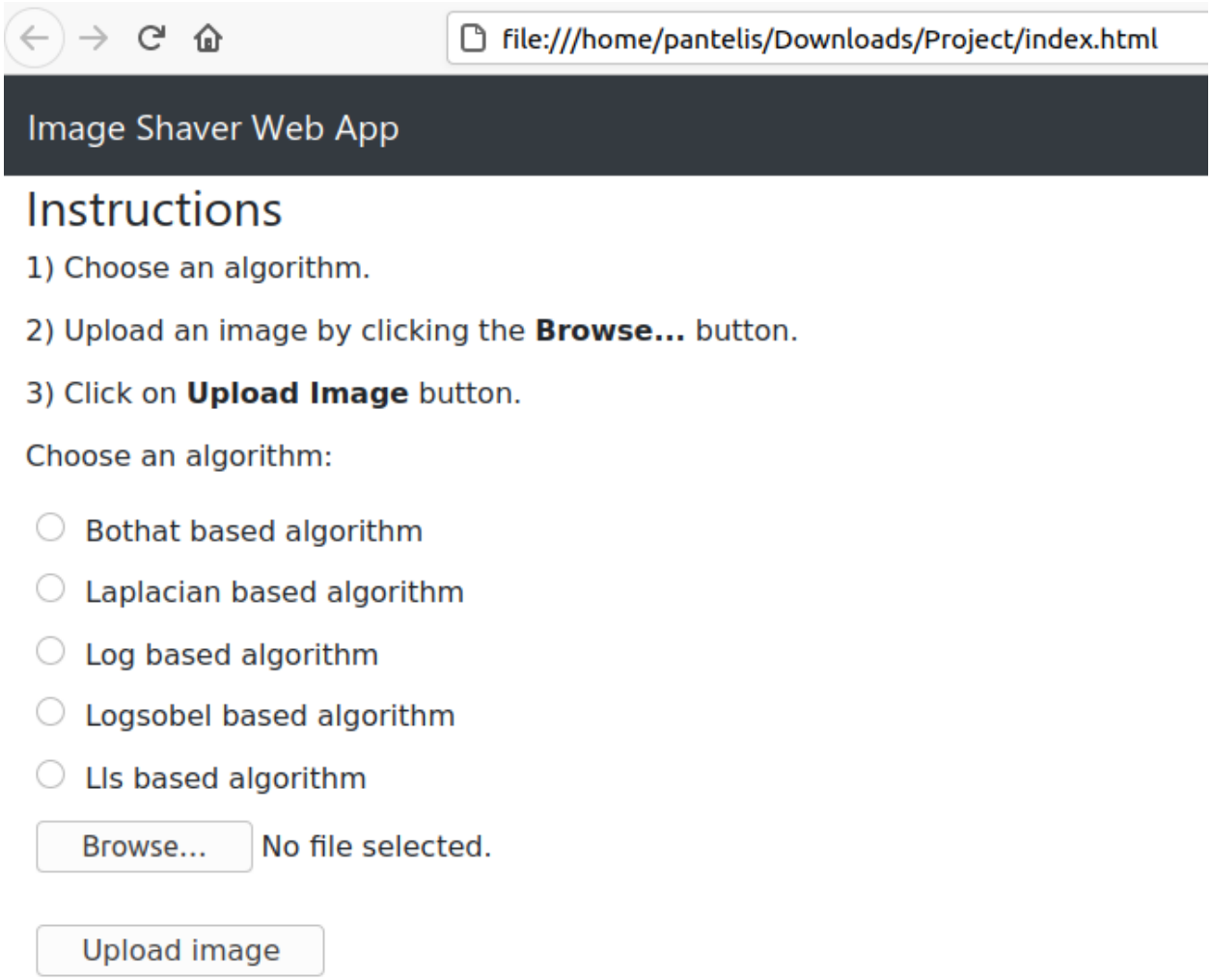
Στη συνέχεια, θα παραμετροποιήσουμε το Flask προκειμένου να φιλοξενήσει την εφαρμογή μας αλλά συγχρόνως και να «συνδέσει» τη λειτουργικότητα του OpenCV με την web εφαρμογή μας.

1) Μέσα στο φάκελο του εικονικού περιβάλλοντος που βρισκόμαστε, δημιουργούμε ένα νέο αρχείο με όνομα webapp.py. Ο κώδικας του αρχείου παρατίθεται στην ενότητα «Παράρτημα».
2) Αντιστοίχως στον ίδιο φάκελο δημιουργούμε ένα αρχείο index.html το οποίο αποτελεί και το entry point της εφαρμογής μας για να επιλέξουμε κάποιον απ' τους 5 αλγόριθμους και να ανεβάσουμε, καθώς και το user interface της εφαρμογής.
3) Στον ίδιο φάκελο μπορούμε να προσθέσουμε επιπλέον logic της εφαρμογής που θα χρειαστεί (όπως για παράδειγμα για να υλοποιήσουμε κάποια μέθοδο που δεν περιλαμβάνεται στο OpenCV). Εν προκειμένω, έχει δημιουργηθεί ένα αρχείο με όνομα cv_filters.py το οποίο περιλαμβάνει επιπλέον λειτουργικότητα που χρειάζονται οι αλγόριθμοι και δεν υπάρχει στο OpenCV (η χρήση του Wiener φίλτρου αφαίρεσης θορύβου).

3.2 Εκτέλεση εφαρμογής

Πλέον είμαστε στο στάδιο για την εκτέλεση της εφαρμογής. Ακολουθούμε τα παρακάτω βήματα:

1) Δεν χρειάζεται να ανοίξουμε νέο terminal διότι πρέπει να είμαστε στο ίδιο εικονικό περιβάλλον που περιγράφηκε παραπάνω.
2) Εκτελούμε την εντολή: export FLASK_APP=webapp.py
3) Εκτελούμε την εντολή: *flask run -host=0.0.0.0*
4) Πλέον ο web server του Flask είναι σε λειτουργία και φιλοξενεί την εφαρμογή μας (η εφαρμογή είναι προσβάσιμη στο link: http://localhost:5000/uploader
5) Ανοίγοντας το αρχείο index.html με οποιονδήποτε web browser, εμφανίζεται η παρακάτω εικόνα:



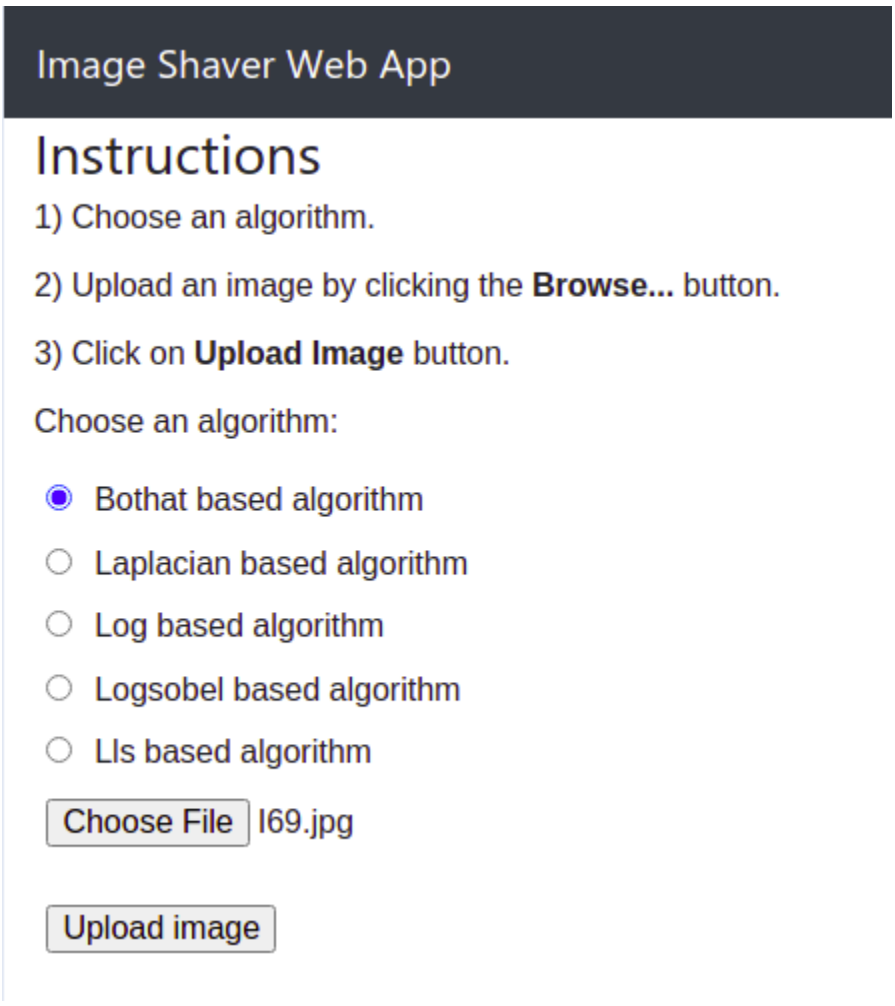
Εικόνα 2: Αρχική σελίδα εφαρμογής

3.3 Εκτέλεση της εφαρμογής σε Virtual Machine – Ενδεικτικά αποτελέσματα

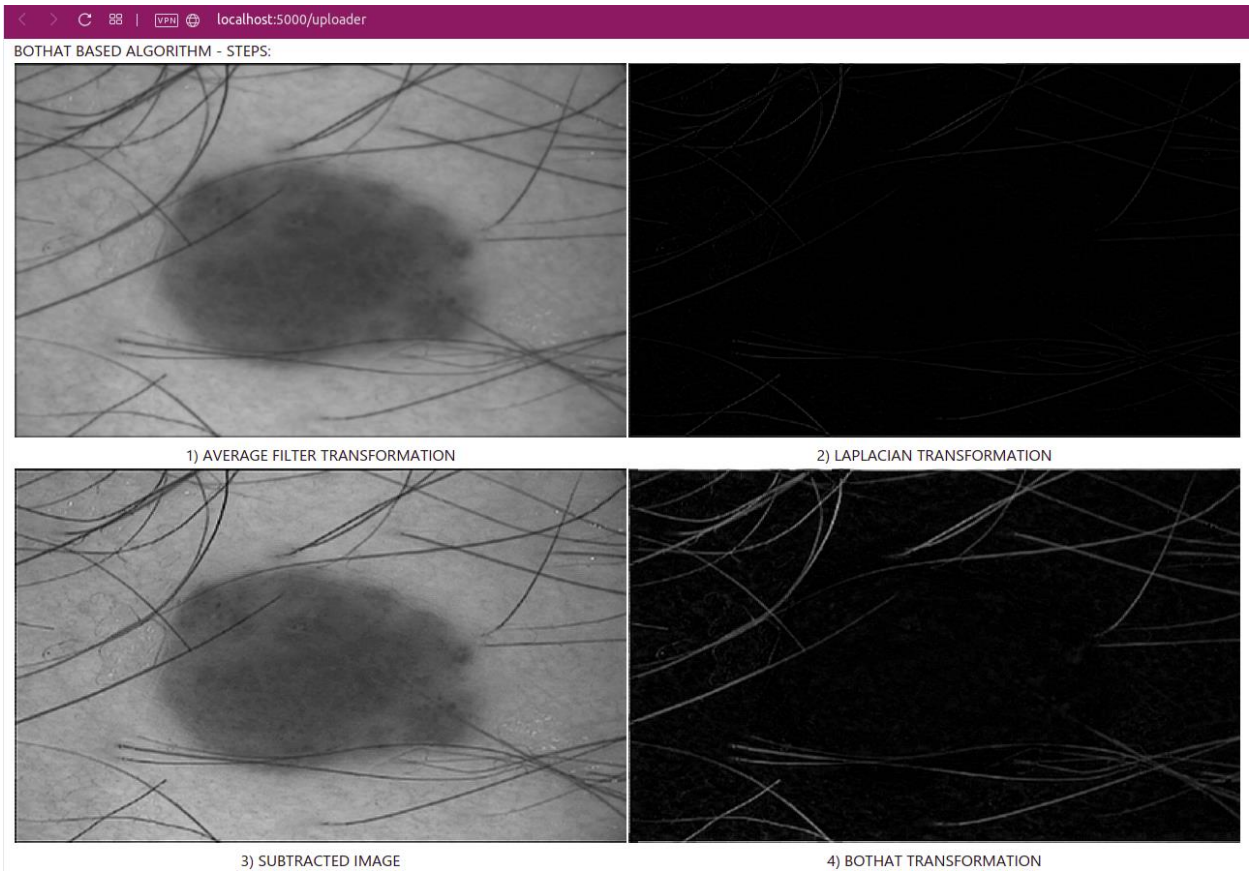
Οι ενότητες 3.1 και 3.2 μπορούν να παραλειφθούν, αν εκτελεστεί απευθείας η εφαρμογή από κάποιο virtual machine στο οποίο θα έχουν εκτελεστεί τα παραπάνω βήματα και πλέον θα είμαστε σε θέση απλά να εισέλθουμε στο url http://telephone.ddns.net:5000/home και να δοκιμάσουμε την web εφαρμογή.

Στη συνέχεια έχουμε κάποια ενδεικτικά αποτελέσματα που παρουσιάζονται κατά την εκτέλεση των αλγορίθμων. Οι αλγόριθμοι που εξετάζονται είναι οι εξής:

- Bothat

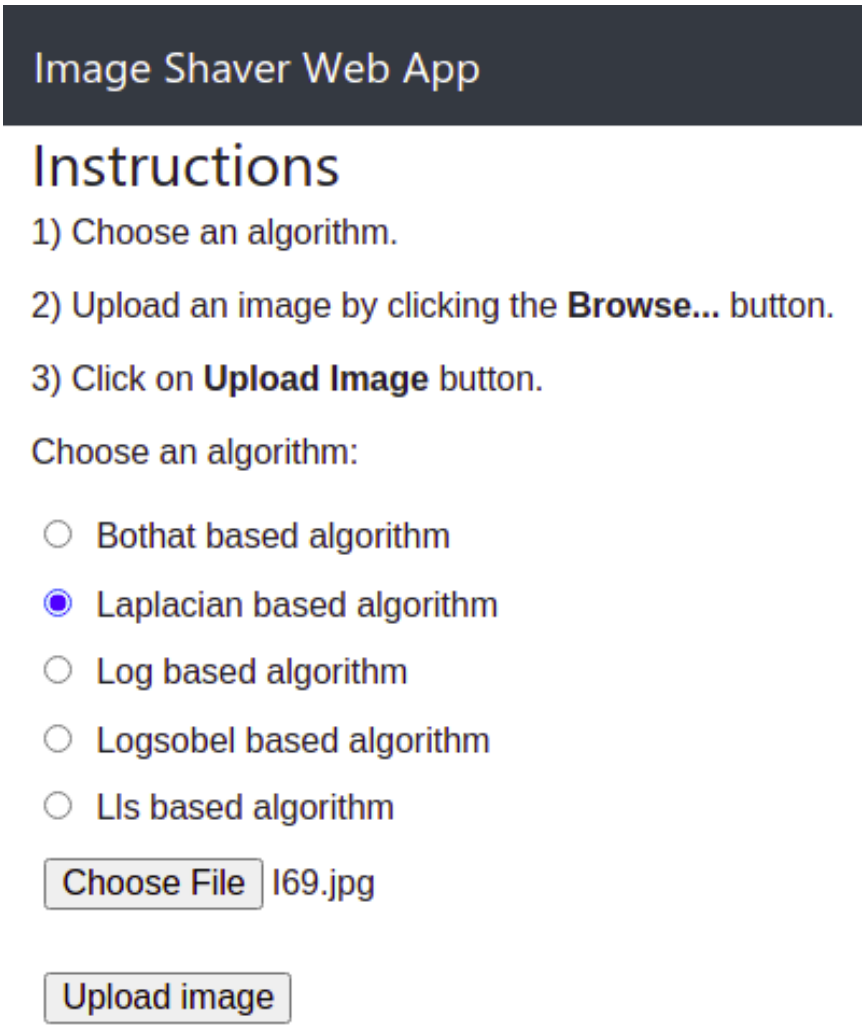


Εικόνα 3: Επιλογή αλγορίθμου BotHat

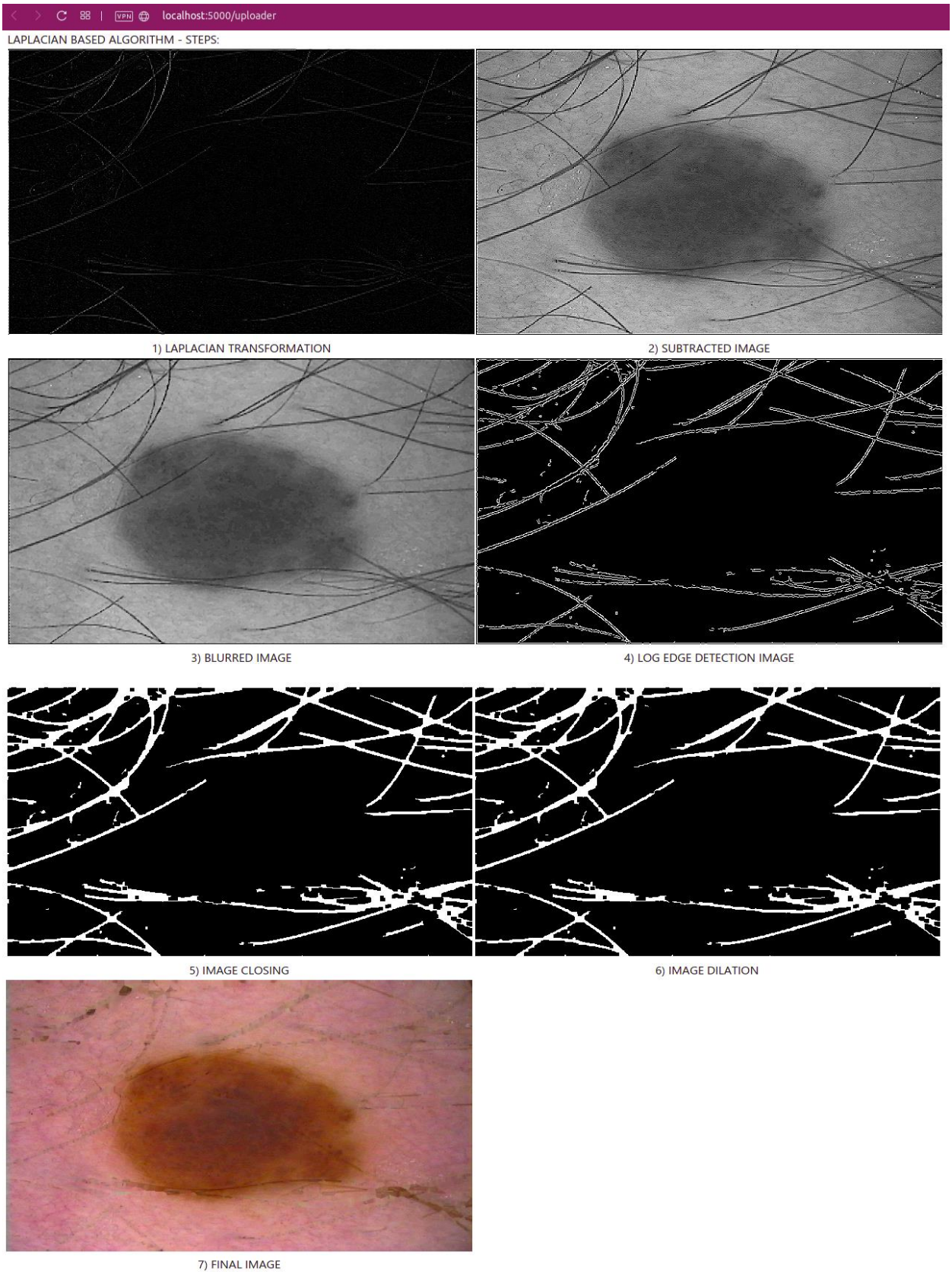


Εικονα 4: Βήματα εκτέλεσης BotHat αλγόριθμου – Ενδεικτικά αποτελέσματα

- Laplacian

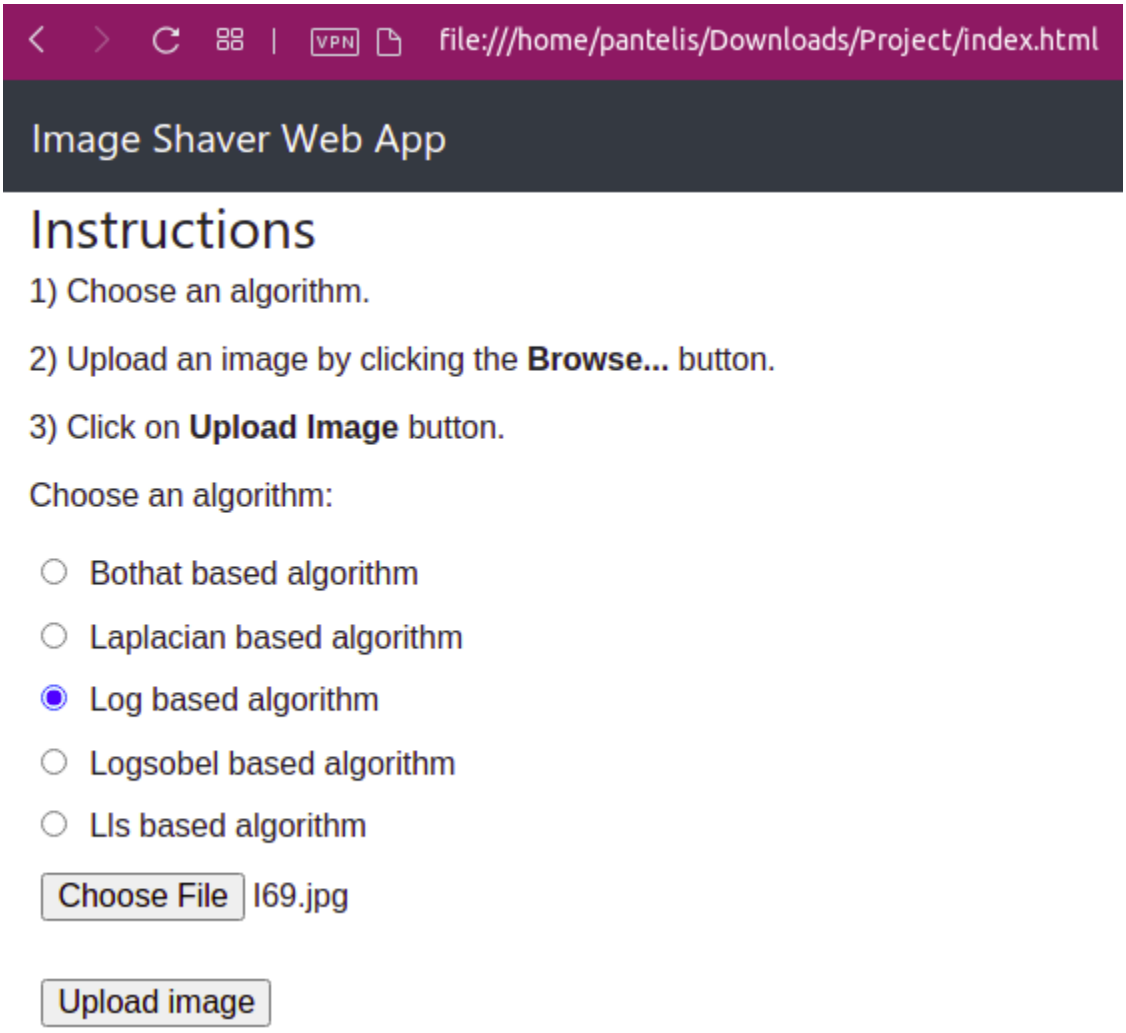


Εικόνα 5: Επιλογή αλγορίθμου Laplacian

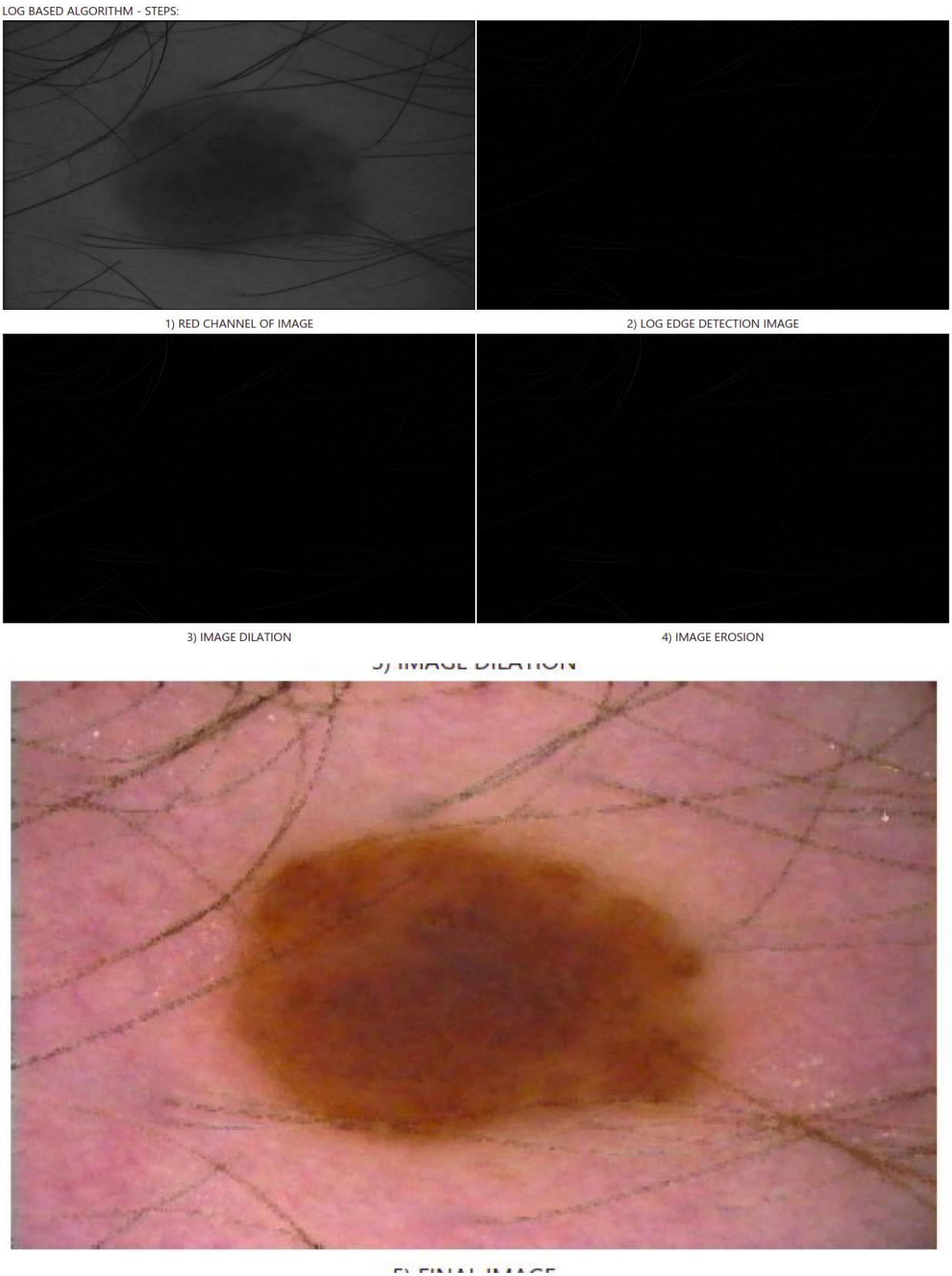


Εικόνα 6: Βήματα εκτέλεσης Laplacian αλγόριθμου – Ενδεικτικά αποτελέσματα

- Log

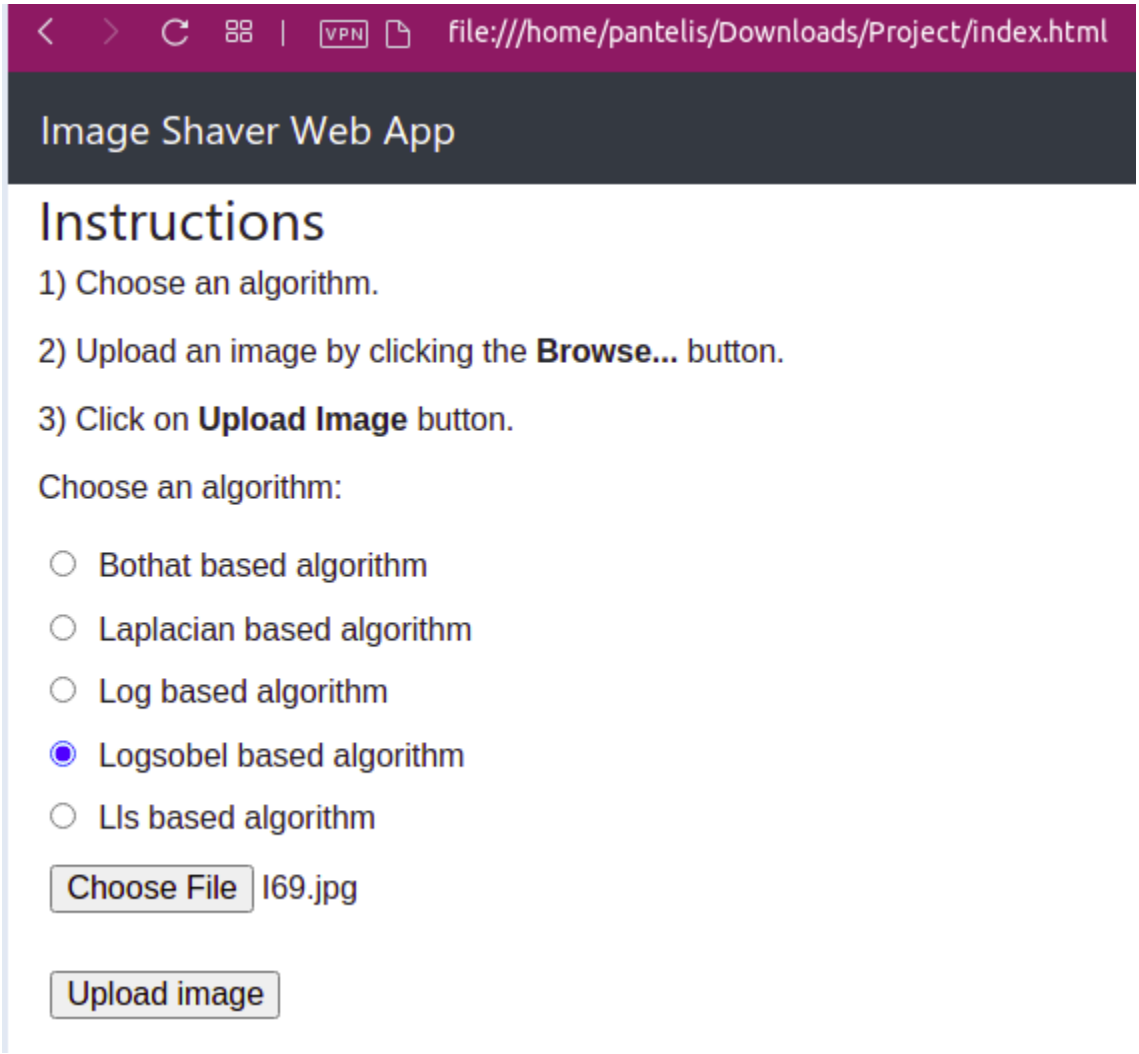


Εικόνα 7: Επιλογή αλγορίθμου LoG (Laplacian Of Gaussian)

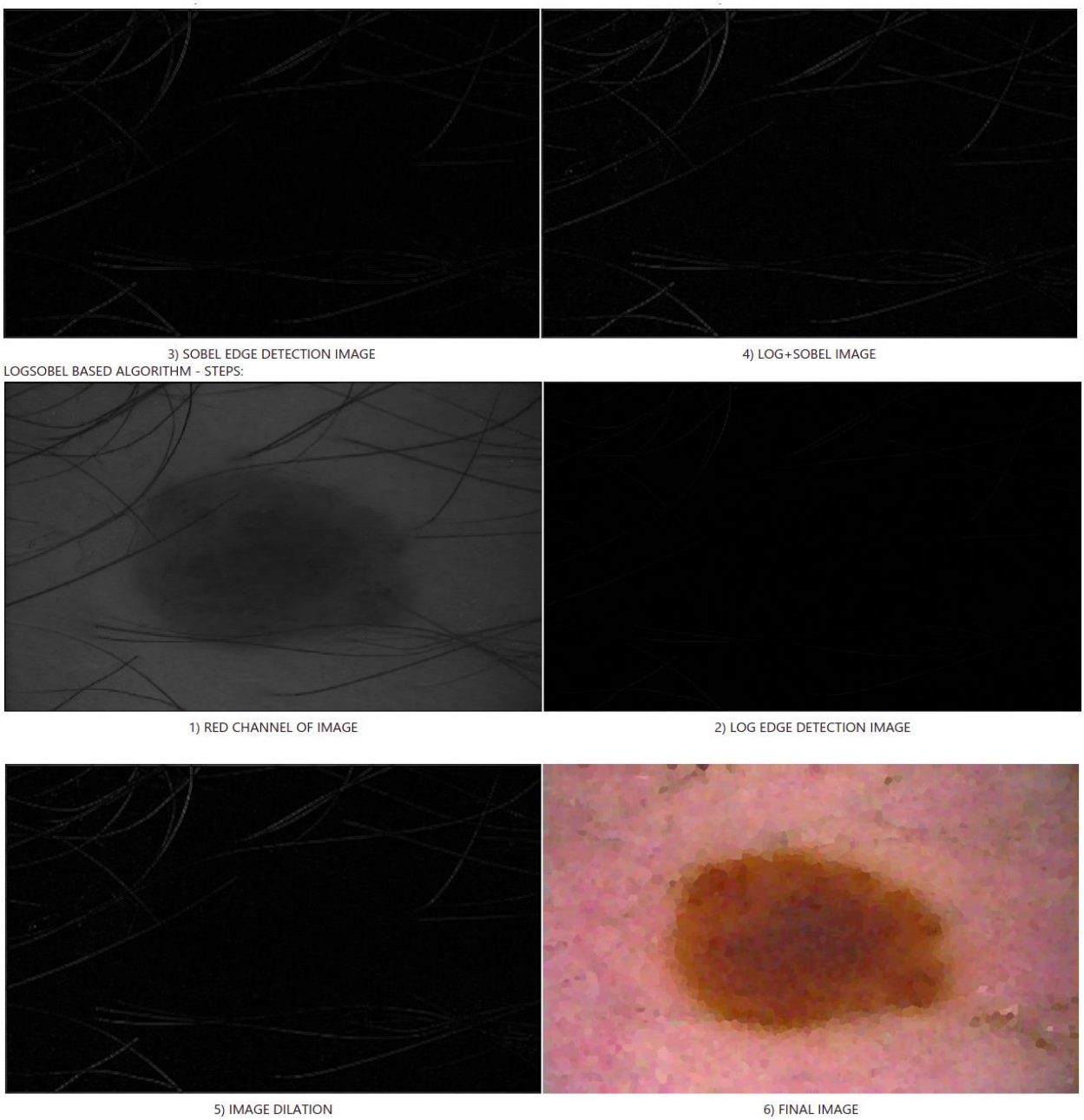


Εικόνα 8: Βήματα εκτέλεσης LoG αλγόριθμου – Ενδεικτικά αποτελέσματα

- Logsobel

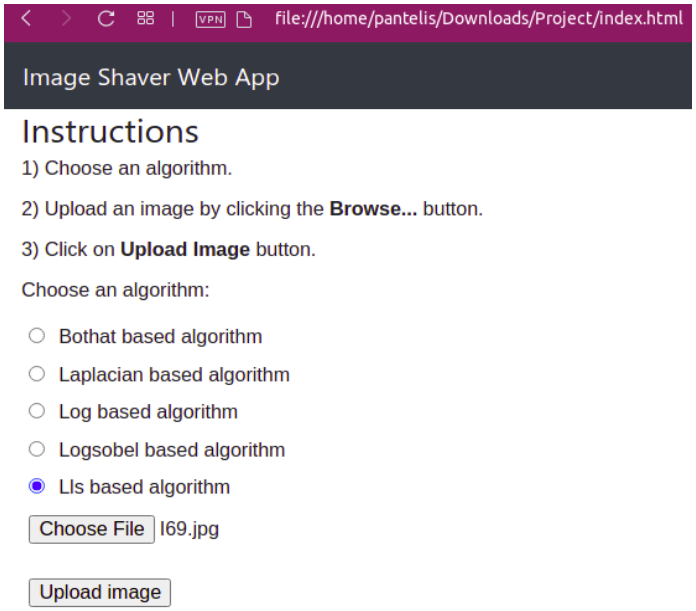


Εικόνα 9: Επιλογή αλγορίθμου LoG-Sobel

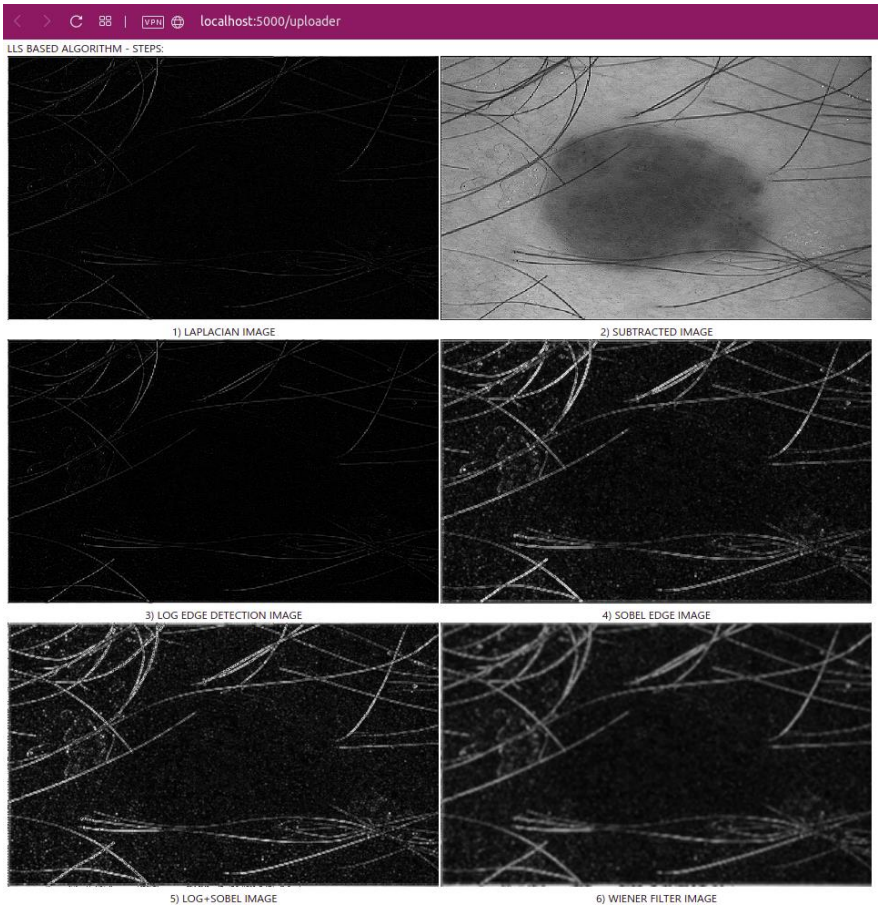


Εικόνα 10: Επιλογή αλγορίθμου LoG-Sobel

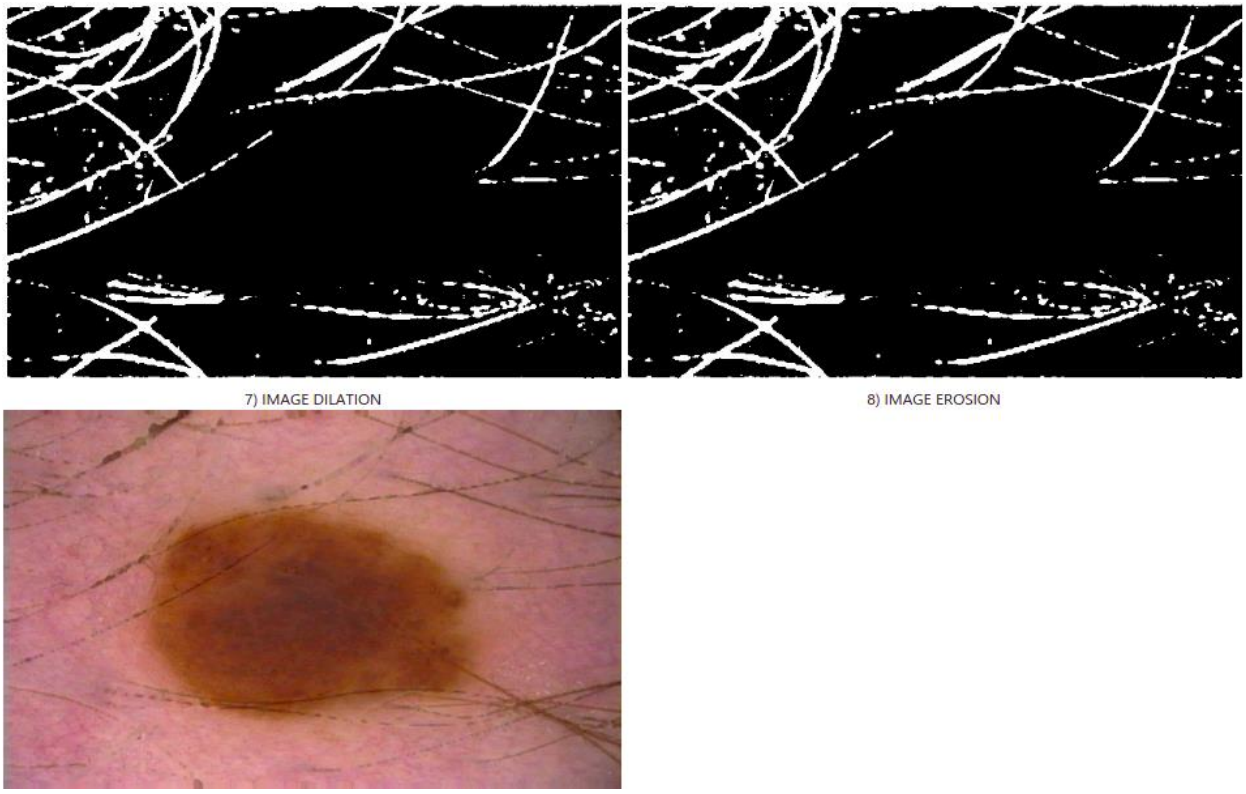
- Lls



Εικόνα 11: Επιλογή αλγορίθμου Lls



Εικόνα 12 (1/2): Βήματα εκτέλεσης Lls αλγόριθμου – Ενδεικτικά αποτελέσματα



Εικόνα 13 (2/2): Βήματα εκτέλεσης LIs αλγόριθμου – Ενδεικτικά αποτελέσματα

3.4 Περιγραφή κώδικα εφαρμογής

Η συγκεκριμένη εφαρμογή έχει αναπτυχθεί όσον αφορά το backend κομμάτι σε Python (Flask web app) και σε html, css και javascript στο front end κομμάτι.

Σε ότι αφορά τον κώδικα Python, ακολουθεί σύντομη περιγραφή του κώδικα που αναπτύχθηκε:

webapp.py

```
from flask import Flask, render_template, request
from cv_filters import WienerFilter
import cv2 as cv
import numpy as np
```

Αρχικά, εισάγουμε τις κλάσεις Flask και request καθώς και τη μέθοδ *render_template()* που χρειάζονται για τη διαχείριση των αιτημάτων http requests που θα δέχεται η εφαρμογή και αναλόγως του αλγορίθμου που θ α επιλεχθεί, να χρησιμοποιηθεί και η αντίστοιχη μέθοδος και στο τέλος να εμφανίσει τα βήματα με τα αντίστοιχα αποτελέσματα.

Στη συνέχεια εισάγουμε τη βιβλιοθήκη OpenCV η οποία στην Python δηλώνεται ως cv2, καθώς και τη βιβλιοθήκη NumPy[vii] που χρειάζονται για τις μαθηματικές πράξεις των εικόνων και των φίλτρων που θα χρησιμοποιηθούν σε κάθε αλγόριθμο.

Τέλος έχουμε και τη βοηθητική κλάση WienerFilter που περιγράφεται και στο Παράρτημα 5 για να υλοποιήσουμε το Wiener φίλτρο που δεν υπάρχει άμεσα στην βιβλιοθήκη OpenCV.

```
app = Flask(__name__)
```

Εδώ δηλώνουμε ένα νέο instance της κλάσης Flask όπου θα μπορέσουμε να χρησιμοποιήσουμε στη συνέχεια για την web εφαρμογή.

```
@app.route('/uploader', methods = ['GET', 'POST'])
```

Εδώ δηλώνουμε έναν κανόνα για να καλέσουμε την web εφαρμογή μας (/uploader) και μπορούμε να την καλέσουμε είτε με HTTP GET είτε με HTTP POST μέθοδο. Στην παρούσα εργασία θα την καλέσουμε με POST μέθοδο.

```
def upload_file():
  if request.method == 'GET':
    return 'GET Method not allowed'
```

Εδώ δηλώνουμε τη μέθοδο upload_file που θα κληθεί από την web φόρμα του client και επιτρέπουμε μόνο HTTP POST κλήση.

```
if request.method == 'POST':
    f = request.files['file']
```

Εδώ παίρνουμε το αρχείο της εικόνας που εστάλη από τον client.

```
if request.form['algorithm'] == 'bothat':
      init_img = cv.imread(f.filename)
      img = cv.imread(f.filename, cv.IMREAD_GRAYSCALE)
      avg_img = cv.blur(img, (3,3))
      cv.imwrite('static/img/bothat/avg_img_'+f.filename,avg_img)
      avg_img_fname = 'static/img/bothat/avg_img_'+ f.filename
      laplacian_img = cv.Laplacian(avg_img,cv.CV_64F)
      cv.imwrite('static/img/bothat/laplacian_img_'+f.filename,laplacian_img)
      laplacian_img_fname = 'static/img/bothat/laplacian_img_' + f.filename
      subtracted_img = avg_img - laplacian_img
      cv.imwrite('static/img/bothat/subtracted_img_'+f.filename,subtracted_img)
      subtracted_img_fname = 'static/img/bothat/subtracted_img_' + f.filename
      kernel = cv.getStructuringElement(1, (17, 17))
      bothat_img = cv.morphologyEx(subtracted_img, cv.MORPH_BLACKHAT, kernel)
      cv.imwrite('static/img/bothat/bothat_img_'+f.filename,bothat_img)
      bothat_img2 = cv.imread('static/img/bothat/bothat_img_'+f.filename, 0)
      bothat_img_fname = 'static/img/bothat/bothat_img_' + f.filename
      #adj_img = imadjust(bothat_img)
      ret,thresh2 = cv.threshold(bothat_img2, 10, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
      kernel = np.ones((1,1),np.uint8)
      dilated_img = cv.dilate(thresh2,kernel,iterations = 1)
      cv.imwrite('static/img/bothat/dilated_img_'+f.filename,dilated_img)
      dilated_img_fname = 'static/img/bothat/dilated_img_' + f.filename
      res_img = cv.inpaint(init_img,thresh2,1,cv.INPAINT_TELEA)
      cv.imwrite('static/img/bothat/res_img_'+f.filename,res_img)
      res_img_fname = 'static/img/bothat/res_img_' + f.filename
      return render_template('bothat.html', avg_img = avg_img_fname, laplacian_img = laplacian_img_fname, subtracted_img =
subtracted_img_fname, bothat_img = bothat_img_fname, dilated_img = dilated_img_fname, res_img = res_img_fname)
```

Εδώ περιγράφεται ο αλγόριθμος bothat όπου αρχικά μετατρέπουμε την εικόνα σε γκρι εκδοχή της. Στη συνέχεια δημιουργούμε ένα φίλτρο blur 3x3 και στη συνέχεια εφαρμόζουμε τον τελεστή Laplacian. Στη συνέχεια θα αφαιρέσουμε τις δύο εικόνες και στο αποτέλεσμα θα εφαρμόσουμε τον αλγόριθμο BotHat

(ή BlackHat όπως υπάρχει στο OpenCV). Στη συνέχεια, θα εφαρμόσουμε binary mask με τη μέθοδο Otsu προκειμένου οι τρίχες (εν προκειμένω) να εμφανιστούν σαν άσπρες περιοχές και το υπόλοιπο δέρμα μία μαύρη περιοχή. Στη συνέχεια εφαρμόζουμε dilation προκειμένου να ενισχυθεί το πάχος των τριχών της προηγούμενης εικόνας. Στη συνέχεια, καλούμε τη μέθοδο *cv.inpaint()* προκειμένου να εφαρμόσουμε αφαίρεση των τριχών στην τελική εικόνα. Τέλος, καλούμε την render_template μέθοδο του Flask προκειμένου να επιστρέψουμε τα δεδομένα στη σελίδα.

```
if request.form['algorithm'] == 'laplacian':
    init_img = cv.imread(f.filename)
    img = cv.imread(f.filename, cv.IMREAD_GRAYSCALE)
    laplacian_img = cv.Laplacian(img,cv.CV_64F)
    cv.imwrite('static/img/laplacian/laplacian_img_'+f.filename,laplacian_img)
    laplacian_img_fname = 'static/img/laplacian/laplacian_img_'+ f.filename
    subtracted_img = img - laplacian_img
    cv.imwrite('static/img/laplacian/subtracted_img_'+f.filename,subtracted_img)
    subtracted_img_fname = 'static/img/laplacian/subtracted_img_' + f.filename
    subtracted_img = cv.imread('static/img/laplacian/subtracted_img_'+f.filename, cv.IMREAD_GRAYSCALE)
    blurred_img = cv.medianBlur(subtracted_img,3)
    cv.imwrite('static/img/laplacian/blurred_img_'+f.filename,blurred_img)
    blurred_img_fname = 'static/img/laplacian/blurred_img_' + f.filename
    log_edge_detection_img = cv.Canny(blurred_img, 100, 200)
    cv.imwrite('static/img/laplacian/log_edge_detection_img_'+f.filename,log_edge_detection_img)
    log_edge_detection_img_fname = 'static/img/laplacian/log_edge_detection_img_' + f.filename
    kernel = np.ones((5,5),np.uint8)
    close_img = cv.morphologyEx(log_edge_detection_img, cv.MORPH_CLOSE, kernel)
    cv.imwrite('static/img/laplacian/close_img_'+f.filename,close_img)
    close_img_fname = 'static/img/laplacian/close_img_'+ f.filename
    ret,thresh2 = cv.threshold(close_img, 10, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
    kernel = np.ones((1,1),np.uint8)
    dilated_img = cv.dilate(thresh2,kernel,iterations = 1)
    cv.imwrite('static/img/laplacian/dilated_img_'+f.filename,dilated_img)
    dilated_img_fname = 'static/img/laplacian/dilated_img_' + f.filename
    res_img = cv.inpaint(init_img,thresh2,1,cv.INPAINT_TELEA)
    cv.imwrite('static/img/laplacian/res_img_'+f.filename,res_img)
    res_img_fname = 'static/img/laplacian/res_img_' + f.filename
    return render_template('laplacian.html', laplacian_img = laplacian_img_fname, subtracted_img = subtracted_img_fname, blurred_img
= blurred_img_fname, log_edge_detection_img = log_edge_detection_img_fname, close_img = close_img_fname, dilated_img =
dilated_img_fname, res_img = res_img_fname)
```

Εδώ περιγράφεται ο αλγόριθμος Laplacian ο οποίος είναι παρόμοιος με τον BotHat απλά γίνονται κάποια διαφορετικά βήματα μετά τον τελεστή Laplacian και πιο συγκεκριμένα , χρησιμοποιείται φίλτρο MedianBlur, στη συνέχεια εφαρμόζεται το LoG Edge Detection μέσω της μεθόδου Canny[viii] του OpenCV . Στη συνέχεια εφαρμόζεται ο μορφολογικός μετασχηματισμός του κλεισίματος όπως αναφέρθηκε και στην προηγούμενη ενότητα, μετά dilation και μετά πάλι ομοίως με τον παραπάνω αλγόριθμο εφαρμόζουμε το φίλτρο για να αφαιρεθούν οι τρίχες από το δέρμα. Τέλος, επιστρέφουμε τα δεδομένα στη σελίδα.

```
if request.form['algorithm'] == 'log':
    init_img = cv.imread(f.filename)
    red_img = init_img.copy()
    red_img[:, :, 0] = 0
    red_img[:, :, 1] = 0
    cv.imwrite('static/img/log/red_img_'+f.filename,red_img)
    red_img_fname = 'static/img/log/red_img_'+ f.filename
    red_img = cv.imread(red_img_fname, cv.IMREAD_GRAYSCALE)
    cv.imwrite('static/img/log/red_img_gray_'+f.filename,red_img)
    red_img_fname = 'static/img/log/red_img_gray_'+ f.filename
    blur =  cv.GaussianBlur(red_img,(3,3),0)
    log_edge_detection_img = cv.Laplacian(blur,cv.CV_64F)
    cv.imwrite('static/img/log/log_edge_detection_img_'+f.filename,log_edge_detection_img)
    log_edge_detection_img = cv.imread('static/img/log/log_edge_detection_img_'+f.filename, cv.IMREAD_GRAYSCALE)
```

```
    log_edge_detection_img_fname = 'static/img/log/log_edge_detection_img_' + f.filename
    kernel = np.ones((1,1),np.uint8)
    dilated_img = cv.dilate(log_edge_detection_img,kernel,iterations = 1)
    cv.imwrite('static/img/log/dilated_img_'+f.filename,dilated_img)
    dilated_img_fname = 'static/img/log/dilated_img_' + f.filename
    eroded_img = cv.erode(dilated_img, kernel, iterations =1)
    cv.imwrite('static/img/log/eroded_img_'+f.filename,eroded_img)
    eroded_img_fname = 'static/img/log/eroded_img_' + f.filename
    res_img = cv.inpaint(init_img,eroded_img,1,cv.INPAINT_TELEA)
    cv.imwrite('static/img/log/res_img_'+f.filename,res_img)
    res_img_fname = 'static/img/log/res_img_' + f.filename
    return render_template('log.html', red_img = red_img_fname, log_edge_detection_img = log_edge_detection_img_fname, dilated_img
= dilated_img_fname, eroded_img = eroded_img_fname, res_img = res_img_fname)
```

Εδώ εφαρμόζουμε τον αλγόριθμο LoG, όπου αρχικά παίρνουμε την κόκκινη εκδοχή της εικόνας, την μετατρέπουμε σε ισοδύναμη γκρι εκδοχή της, εφαρμόζεται ο Gaussian αλγόριθμος για το blurring της εικόνας και αμέσως μετά ο αντίστοιχος Laplacian, στη συνέχεια εφαρμόζονται οι μορφολογικοί μετασχηματισμοί dilation και erosion και στο τέλος πάλι εφαρμόζεται το αντίστοιχο φίλτρο για να αφαιρεθούν οι τρίχες και αποστέλλονται τα δεδομένα στη σελίδα.

```
    if request.form['algorithm'] == 'logsobel':
    init_img = cv.imread(f.filename)
    red_img = init_img.copy()
    red_img[:, :, 0] = 0
    red_img[:, :, 1] = 0
    cv.imwrite('static/img/logsobel/red_img_'+f.filename,red_img)
    red_img_fname = 'static/img/logsobel/red_img_'+ f.filename
    red_img = cv.imread(red_img_fname, cv.IMREAD_GRAYSCALE)
    cv.imwrite('static/img/logsobel/red_img_gray_'+f.filename,red_img)
    red_img_fname = 'static/img/logsobel/red_img_gray_'+ f.filename
    blur =  cv.GaussianBlur(red_img,(3,3),0)
    log_edge_detection_img = cv.Laplacian(blur,cv.CV_64F)
    cv.imwrite('static/img/logsobel/log_edge_detection_img_'+f.filename,log_edge_detection_img)
    log_edge_detection_img = cv.imread('static/img/logsobel/log_edge_detection_img_'+f.filename, cv.IMREAD_GRAYSCALE)
    log_edge_detection_img_fname = 'static/img/logsobel/log_edge_detection_img_' + f.filename
    grad_x = cv.Sobel(log_edge_detection_img, cv.CV_16S, 1, 0, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT)
    grad_y = cv.Sobel(log_edge_detection_img, cv.CV_16S, 0, 1, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT)
    abs_grad_x = cv.convertScaleAbs(grad_x)
    abs_grad_y = cv.convertScaleAbs(grad_y)
    sobel_edge_detection_img = cv.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
    cv.imwrite('static/img/logsobel/sobel_edge_detection_img_'+f.filename,sobel_edge_detection_img)
    sobel_edge_detection_img_fname = 'static/img/logsobel/sobel_edge_detection_img_'+ f.filename
    add_binary_img = log_edge_detection_img + sobel_edge_detection_img
    cv.imwrite('static/img/logsobel/add_binary_img_'+f.filename,add_binary_img)
    add_binary_img_fname = 'static/img/logsobel/add_binary_img_'+ f.filename
    kernel = np.ones((1,1),np.uint8)
    dilated_img = cv.dilate(add_binary_img,kernel,iterations = 1)
    cv.imwrite('static/img/logsobel/dilated_img_'+f.filename,dilated_img)
    dilated_img_fname = 'static/img/logsobel/dilated_img_' + f.filename
    res_img = cv.inpaint(init_img,dilated_img,1,cv.INPAINT_TELEA)
    cv.imwrite('static/img/logsobel/res_img_'+f.filename,res_img)
    res_img_fname = 'static/img/logsobel/res_img_' + f.filename
    return render_template('logsobel.html', red_img = red_img_fname, log_edge_detection_img = log_edge_detection_img_fname,
sobel_edge_detection_img = sobel_edge_detection_img_fname, add_binary_img = add_binary_img_fname, dilated_img =
dilated_img_fname, res_img = res_img_fname)
```

Εδώ εφαρμόζεται ο αλγόριθμος LoG – Sobel, όπου όμοια με τον παραπάνω αλγόριθμο LoG παίρνουμε την κόκκινη και στη συνέχεια τη γκρι εκδοχή της εικόνας, ακολουθούμε όλα τα βήματα όπως και στον LoG αλγόριθμο, αλλά παράλληλα εφαρμόζουμε και τον αλγόριθμο Sobel[ix] και στη συνέχεια εφαρμόζουμε dilation και αντίστοιχα γίνεται αφαίρεση των τριχών και αποστέλλονται τα δεδομένα στη σελίδα.

```
if request.form['algorithm'] == 'lls':
    init_img = cv.imread(f.filename)
```

```
    img = cv.imread(f.filename, cv.IMREAD_GRAYSCALE)
    laplacian_img = cv.Laplacian(img,cv.CV_64F)
    cv.imwrite('static/img/lls/laplacian_img_'+f.filename,laplacian_img)
    laplacian_img_fname = 'static/img/lls/laplacian_img_'+ f.filename
    subtracted_img = img - laplacian_img
    cv.imwrite('static/img/lls/subtracted_img_'+f.filename,subtracted_img)
    subtracted_img_fname = 'static/img/lls/subtracted_img_' + f.filename
    blur =  cv.GaussianBlur(subtracted_img,(3,3),0)
    log_edge_detection_img = cv.Laplacian(blur,cv.CV_64F)
    cv.imwrite('static/img/lls/log_edge_detection_img_'+f.filename,log_edge_detection_img)
    log_edge_detection_img = cv.imread('static/img/lls/log_edge_detection_img_'+f.filename, cv.IMREAD_GRAYSCALE)
    log_edge_detection_img_fname = 'static/img/lls/log_edge_detection_img_' + f.filename
    grad_x = cv.Sobel(log_edge_detection_img, cv.CV_16S, 1, 0, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT)
    grad_y = cv.Sobel(log_edge_detection_img, cv.CV_16S, 0, 1, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT)
    abs_grad_x = cv.convertScaleAbs(grad_x)
    abs_grad_y = cv.convertScaleAbs(grad_y)
    sobel_edge_detection_img = cv.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
    cv.imwrite('static/img/lls/sobel_edge_detection_img_'+f.filename,sobel_edge_detection_img)
    sobel_edge_detection_img_fname = 'static/img/lls/sobel_edge_detection_img_'+ f.filename
    add_binary_img = log_edge_detection_img + sobel_edge_detection_img
    cv.imwrite('static/img/lls/add_binary_img_'+f.filename,add_binary_img)
    add_binary_img_fname = 'static/img/lls/add_binary_img_'+ f.filename
    input_img = cv.imread(add_binary_img_fname, cv.IMREAD_GRAYSCALE)
    wiener_filter = WienerFilter(input_img, (5,5))
    wiener_output_img = wiener_filter.estimateOutput()
    cv.imwrite('static/img/lls/wiener_output_img_'+f.filename,wiener_output_img)
    wiener_output_img_fname = 'static/img/lls/wiener_output_img_'+ f.filename
    wiener_output_img = cv.imread(wiener_output_img_fname, cv.IMREAD_GRAYSCALE)
    ret,thresh2 = cv.threshold(wiener_output_img, 10, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
    kernel = np.ones((1,1),np.uint8)
    dilated_img = cv.dilate(thresh2,kernel,iterations = 1)
    cv.imwrite('static/img/lls/dilated_img_'+f.filename,dilated_img)
    dilated_img_fname = 'static/img/lls/dilated_img_' + f.filename
    eroded_img = cv.erode(dilated_img, kernel, iterations =1)
    cv.imwrite('static/img/lls/eroded_img_'+f.filename,eroded_img)
    eroded_img_fname = 'static/img/lls/eroded_img_' + f.filename
    res_img = cv.inpaint(init_img,eroded_img,1,cv.INPAINT_TELEA)
    cv.imwrite('static/img/lls/res_img_'+f.filename,res_img)
    res_img_fname = 'static/img/lls/res_img_' + f.filename
    return render_template('lls.html', laplacian_img = laplacian_img_fname, subtracted_img = subtracted_img_fname,
log_edge_detection_img = log_edge_detection_img_fname, sobel_edge_detection_img = sobel_edge_detection_img_fname,
add_binary_img = add_binary_img_fname, wiener_output_img = wiener_output_img_fname, dilated_img = dilated_img_fname,
eroded_img = eroded_img_fname, res_img = res_img_fname)
```

Εδώ εφαρμόζεται ο αλγόριθμος Lls ο οποίος είναι παρόμοιος με τον παραπάνω LoG – Sobel, εδώ όμως υπάρχει η διαφορά ότι εφαρμόζεται και η τεχνική φίλτρου Wiener, το οποίο βρίσκεται στο αρχείο cv_filters.py, όπως περιγράφεται παρακάτω:

cv_filters.py

```
import cv2 as cv
import numpy as np

class WienerFilter():
  def __init__(self, input, filter_size):
    self.input = input
    self.filter_size = filter_size
    if(input.ndim != 2):
      print("Image not 2d")
    if(filter_size[0]%2 != 1 or filter_size[1]%2 != 1 or filter_size[0]<=1 or filter_size[1]<=1):
      print("Invalid filter dimension")

  def estimateOutput(self):
    means = cv.boxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size), anchor=(-1,1), normalize=True,
borderType=cv.BORDER_REPLICATE)
```

```
   square_means = cv.sqrBoxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size), anchor=(-1,1), normalize=True,
borderType=cv.BORDER_REPLICATE)

   means2 = cv.multiply(means, means)

   #calculating the variance matrix
   variances = square_means - means2

   # estimating noise variance by finding projections across length and width
   avgVarianceMat = cv.reduce(variances, 1, cv.REDUCE_SUM, -1)
   avgVarianceMat = cv.reduce(avgVarianceMat, 0, cv.REDUCE_SUM, -1)

   noiseVar = np.asscalar(avgVarianceMat/self.input.shape[0] * self.input.shape[1])

   y = np.zeros(self.input.shape)
   for row in range(self.input.shape[0]):
     for col in range(self.input.shape[1]):
       y[row][col] = means[row][col] + max(0, variances[row][col] - noiseVar) * (self.input[row][col]-
means[row][col])/max(variances[row][col], noiseVar)

   return y

 def estimateOutputColorised(self):
   # get all local means in
   means = cv.boxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size), anchor=(-1,1), normalize=True,
borderType=cv.BORDER_REPLICATE)
   square_means = cv.sqrBoxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size), anchor=(-1,1), normalize=True,
borderType=cv.BORDER_REPLICATE)

   means2 = cv.multiply(means, means)
   variances = square_means - means2

   avgVariance = cv.reduce(variances, 0, cv.REDUCE_SUM, -1)
   avgVariance = cv.reduce(avgVariance, 1, cv.REDUCE_SUM, -1)

   noiseVar = avgVariance/self.input.shape[0] * self.input.shape[1] * self.input.shape[2]

   noiseVar = np.reshape(noiseVar, (1,3))

   y = np.zeros(self.input.shape)

   for row in range(self.input.shape[0]):
     for col in range(self.input.shape[1]):
       y[row][col] = self.saturate_cast(means[row][col] + np.maximum(np.zeros(noiseVar[0].shape), variances[row][col] - noiseVar[0]) *
(self.input[row][col]-means[row][col])/np.maximum(variances[row][col], noiseVar[0]))

   return y

 def saturate_cast(self, val):

   if val.any() < 0:
     return np.array([255, 255, 255])
   elif val.any() >255:
     return np.array([255,255, 255])
   return val
```

όπου μπορεί να χρησιμοποιηθεί είτε για έγχρωμη εικόνα (RGB) είτε για γκρι (grayscale) που χρησιμοποιείται στην παρούσα εργασία.

4. ΣΥΜΠΕΡΑΣΜΑΤΑ – ΕΠΟΜΕΝΕΣ ΕΝΕΡΓΕΙΕΣ

4.1 Πλεονεκτήματα

Τα πλεονεκτήματα που υπάρχουν από την ανάπτυξη της εφαρμογής συνοπτικά είναι τα ακόλουθα:

- Με τη χρήση της βιβλιοθήκης OpenCV και τη χρήση της γλώσσας Python, μπορούμε να μετατρέψουμε σχεδόν όλες τις λειτουργικότητες ψηφιακής επεξεργασίας εικόνας του Matlab σε web εφαρμογές, κάνοντας έτσι προσβάσιμες από πολλές κινητές συσκευές και υπολογιστές ανεξαρτήτως λειτουργικού συστήματος.
- Ο λόγος χρησιμοποίησης του Flask web server ήταν επειδή μπορούμε με τη γνώση μόνο μίας γλώσσας προγραμματισμού (Python εν προκειμένω) να αναπτύξουμε web εφαρμογή και παράλληλα να αξιοποιήσουμε τις δυνατότητες του OpenCV.
- Η λειτουργικότητα του OpenCV πλέον μπορεί να καταναλωθεί σε πιο μοντέρνες τεχνολογίες web (π.χ. χρήση web services, web APIs, κτλ.).

4.2 Μειονεκτήματα

Αντίστοιχα με τα πλεονεκτήματα που προαναφέρθηκαν, υπάρχουν αντίστοιχα και τα παρακάτω μειονεκτήματα:

- Η λειτουργικότητα της συγκεκριμένης εφαρμογής επιδέχεται βελτίωσης τόσο σε επίπεδο κώδικα (π.χ. χρησιμοποίηση περισσότερων OOP[x] λειτουργιών αλλά και χρησιμοποίηση λιγότερων I/O λειτουργιών προκειμένου να βελτιωθεί η ταχύτητα εκτέλεσης.
- Η συγκεκριμένη εφαρμογή επιτρέπει την εισαγωγή μόνο μίας εικόνας. Μελλοντικά θα μπορούσε να εισάγεται πλήθος εικόνων και να γίνεται παράλληλη και distributed επεξεργασία τους.
- Η υλοποίηση των αλγορίθμων έγινε προσεγγιστικά στα αποτελέσματα του ισοδύναμου Matlab κώδικα και προφανώς επιδέχεται βελτίωσης (για παράδειγμα χρήση binary mask που εμφανίζουν τις τρίχες της εικόνας ακόμα πιο έντονα για την ομαλότερη διαγραφή τους από την τελική εικόνα).
- Ειδικά για αυτή την εργασία, το περιβάλλον του web server μπορεί να φανεί ασταθές μετά την εκτέλεση πολλών επεξεργασιών με αποτέλεσμα να εμφανίζει cached data ή κάποιες φωτογραφίες να μην τις αναγνωρίζει σαν έγκυρα δεδομένα. Αυτό μπορεί να επιλυθεί σε επόμενη έκδοση της εφαρμογής ή με τροποποίηση του configuration του συστήματος που φιλοξενεί την εφαρμογή. Μέχρι στιγμής φιλοξενείται σεvirtual development environment (Python venv[xi]) που προκρίνεται ως η καλύτερη επιλογή για development και όχι production περιβάλλοντα.

4.3 Εναλλακτικές επιλογές

Πέραν της υλοποίησης της web εφαρμογής σε OpenCV και Flask υπάρχουν οι εξής εναλλακτικές επιλογές:

- Χρήση του Django server αντί για το Flask: Πιο σταθερός και αξιόπιστος web server με περισσότερες επιλογές αλλά και μεγαλύτερο βαθμό δυσκολίας configuration συγκριτικά με το Flask. Είναι επίσης υλοποιημένος σε Python.
- Χρήση μίας εκδοχής του OpenCV σε ASP.NET C#/VB.NET περιβάλλον (https://www.emgu.com) που έχει τη δυνατότητα χρήσης του σε ASP.NET εφαρμογές, μόνο που είναι διαθέσιμο μόνο για Windows servers.
- Χρήση του OpenCV.js (https://docs.opencv.org/3.4/d5/d10/tutorial_js_root.html) που είναι συμβατό με τον Node Js server και είναι συμβατό με όλες τις πλατφόρμες.

4.4 Διδάγματα (Lessons Learned)

Κάποια βασικά σημεία που θα πρέπει να δοθεί προσοχή κατά την εκτέλεση της εφαρμογής, κάποια από τα οποία έχουν ήδη αναφερθεί και θα συμπεριληφθούν παρακάτω είναι τα εξής:

- Οι οδηγίες εκτέλεσης της εφαρμογής βρίσκονται στις υποενότητες 3.1 και 3.2.
- Τα αρχεία και η δομή της εφαρμογής περιγράφονται στην ενότητα 5.1 του Παραρτήματος.
- Η εφαρμογή μπορεί να εγκατασταθεί σε οποιοδήποτε περιβάλλον Windows, Linux, MacOS με εγκατεστημένη την Python 3.x (μπορεί να εγκατασταθεί και σε Python 2.x, απλά κάποια dependencies του OpenCV και του NumPy μπορεί να μη λειτουργήσουν σωστά. Επιπλέον με την Python 3.x η εγκατάσταση επιπλέον packages είναι πολύ πιο εύκολη και γρήγορη από το pip[xii]).
- Η εφαρμογή δέχεται ως είσοδο μία εικόνα, οπότε για μεγαλύτερο πλήθος εικόνων, θα πρέπει να εκτελεστούν ξανά τα ίδια βήματα που περιγράφονται στην υποενότητα 3.3 για κάθε εικόνα.
- Οι μεγάλες εικόνες (είτε μεγαλύτερες από 500x500 pixels, είτε μεγέθους μεγαλύτερου των 500KB) ή αυτές που δεν είναι της μορφής JPEG/PNG μπορεί να προκαλέσουν exceptions κατά την εκτέλεση της εφαρμογής.
- Το I/O της υφιστάμενης εφαρμογής γενικά δεν είναι βέλτιστο κι αυτό γιατί για κάθε βήμα του αλγορίθμου υπάρχει αποθήκευση της εικόνας στο filesystem και στη συνέχεια άνοιγμα του αρχείου για το επόμενο βήμα με αποτέλεσμα όταν υπάρχει παράλληλη χρήση της εφαρμογής από διαφορετικούς clients να υπάρξει είτε καθυστέρηση είτε deadlock.

4.5 Προτάσεις βελτίωσης της υπάρχουσας εφαρμογής

Η συγκεκριμένη εφαρμογή μπορεί να βελτιωθεί σε επόμενες εκδόσεις της σε κάποια θέματα όπως:

- Τα βήματα των αλγορίθμων να μην αποθηκεύουν κάθε εικόνα στο filesystem σε κάθε βήμα εκτέλεσης, αλλά στην μνήμη RAM.
- Ο κάθε αλγόριθμος να έχει ως αποτέλεσμα ένα web api και όχι να γεμίζει με δεδομένα τοδικό του view όπως γίνεται τώρα. Αυτό μπορεί να βοηθήσει την εφαρμογή να χρησιμοποιηθεί με άλλες διαδικτυακές εφαρμογές (api consumers), όπως επίσης και να μην υπάρχει το αντίστοιχο caching που υπάρχει σε αυτή την έκδοση.
- Χρήση του OMPC[xiii] για την εκτέλεση Matlab scripts από περιβάλλον Python, χωρίς την ανάγκη χρήσης του OpenCV. Σε αυτή την περίπτωση ο server θα πρέπει να είναι σε περιβάλλον Windows. Για κάποια scripts, μπορεί να χρησιμοποιηθεί και το λογισμικό Octave[xiv] που είναι cross-platform compatible.

Στην επόμενη ενότητα παρατίθεται ο κώδικας Python όσο και html/css/javascript.

5. ΠΑΡΑΡΤΗΜΑ

5.1 Κώδικας σε Python

Ο κώδικας της εφαρμογής είναι δομημένος ως εξής:

Στην αρχική τοποθεσία του φακέλου υπάρχουν τα παρακάτω αρχεία:

- **webapp.py** : αποτελεί το βασικό αρχείο που υπάρχει όλο το logic σε Python και περιγράφει την web εφαρμογή. Ανάλογα με τον αλγόριθμο που θα επιλεχθεί, εκτελείται ο αντίστοιχος αλγόριθμος σε OpenCV, αποθηκεύει τα αποτελέσματα του κάθε βήματος στο directory static/img και στο τέλος εμφανίζει το αντίστοιχο view του κάθε αλγορίθμου μαζί με τις εικόνες κάθε βήματος που εκτελεί.
- **cv_filters.py:** είναι μία helper κλάση για να εφαρμοστεί το φίλτρο αφαίρεσης θορύβου Wiener που χρησιμοποιείται στον Lls αλγόριθμο.
- **Index.html:** αποτελεί το entry point της εφαρμογής.

και οι παρακάτω φάκελοι:

- **static**: Περιέχει όλα τα αρχεία css, javascript καθώς και τις παραγόμενες εικόνες κατά την εκτέλεση των αλγορίθμων.
- **templates:** περιέχονται τα views για κάθε αλγόριθμο που εμφανίζει όλες τις εικόνες και τα βήματα εκτέλεσης των αλγορίθμων.

Στη συνέχεια παρατίθεται ο κώδικας των αρχείων:

Index.html

```
<!doctype html>
<html class="no-js" lang="">

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="apple-touch-icon" href="icon.png">
  <link rel="stylesheet" href="static/css/normalize.css">
  <link rel="stylesheet" href="static/css/main.css">
  <link rel="stylesheet" href="static/css/bootstrap.min.css">
  <meta name="theme-color" content="#fafafa">
  <style>
  p,
label {
    font: 1rem 'Fira Sans', sans-serif;
```

```
}

input {
   margin: .4rem;
}
 </style>
</head>
<body>
<form action="http://localhost:5000/uploader" enctype="multipart/form-data" method="POST"
target="_blank">
<nav class="navbar navbar-expand-md navbar-dark bg-dark">
<a class="navbar-brand" href="#">Image Shaver Web App</a>
</nav>
<div class="col-xs-12" style="padding-left:15px;">
<h3>Instructions</h3>
<p>1) Choose an algorithm.</p>
<p>2) Upload an image by clicking the <b>Browse...</b> button.</p>
<p>3) Click on <b>Upload Image</b> button.</p>
</div>
<div class="col-xs-12" style="padding-left:15px;">
<p>Choose an algorithm:</p>
<div>
 <input type="radio" id="bothat" name="algorithm" value="bothat">
 <label for="bothat">Bothat based algorithm</label>
</div>
<div>
 <input type="radio" id="laplacian" name="algorithm" value="laplacian">
 <label for="laplacian">Laplacian based algorithm</label>
</div>
<div>
 <input type="radio" id="log" name="algorithm" value="log">
 <label for="log">Log based algorithm</label>
</div>
<div>
 <input type="radio" id="logsobel" name="algorithm" value="logsobel">
 <label for="logsobel">Logsobel based algorithm</label>
</div>
<div>
 <input type="radio" id="lls" name="algorithm" value="lls">
 <label for="lls">Lls based algorithm</label>
</div>
<div>
<p><input type="file" name="file"></p>
<p><input type="submit" value="Upload image"></p>
</div>
</div>
</form>
 <script src="js/vendor/modernizr-3.11.2.min.js"></script>
```

```
  <script src="js/plugins.js"></script>
  <script src="js/main.js"></script>
  <script src="js/bootstrap.min.js"></script>
  <script src="js/jquery.min.js"></script>
</body>
</html>
```

webapp.py

```
from flask import Flask, render_template, request
from cv_filters import WienerFilter

import cv2 as cv
import numpy as np

app = Flask(__name__)

@app.route('/uploader', methods = ['GET', 'POST'])
def upload_file():
   #return send_file('test.jpg', mimetype='image/jpg')
  if request.method == 'GET':
     return 'GET Method not allowed'

  if request.method == 'POST':
     f = request.files['file']
     if request.form['algorithm'] == 'bothat':
        init_img = cv.imread(f.filename)
        img = cv.imread(f.filename, cv.IMREAD_GRAYSCALE)
        avg_img = cv.blur(img, (3,3))
        cv.imwrite('static/img/bothat/avg_img_'+f.filename,avg_img)
        avg_img_fname = 'static/img/bothat/avg_img_'+ f.filename
        laplacian_img = cv.Laplacian(avg_img,cv.CV_64F)
        cv.imwrite('static/img/bothat/laplacian_img_'+f.filename,laplacian_img)
        laplacian_img_fname = 'static/img/bothat/laplacian_img_' + f.filename
        subtracted_img = avg_img - laplacian_img
        cv.imwrite('static/img/bothat/subtracted_img_'+f.filename,subtracted_img)
        subtracted_img_fname = 'static/img/bothat/subtracted_img_' + f.filename
        kernel = cv.getStructuringElement(1, (17, 17))
        bothat_img = cv.morphologyEx(subtracted_img, cv.MORPH_BLACKHAT, kernel)
        cv.imwrite('static/img/bothat/bothat_img_'+f.filename,bothat_img)
        bothat_img2 = cv.imread('static/img/bothat/bothat_img_'+f.filename, 0)
        bothat_img_fname = 'static/img/bothat/bothat_img_' + f.filename
        #adj_img = imadjust(bothat_img)
        ret,thresh2 = cv.threshold(bothat_img2, 10, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
        kernel = np.ones((1,1),np.uint8)
        dilated_img = cv.dilate(thresh2,kernel,iterations = 1)
        cv.imwrite('static/img/bothat/dilated_img_'+f.filename,dilated_img)
```

```
        dilated_img_fname = 'static/img/bothat/dilated_img_' + f.filename
        res_img = cv.inpaint(init_img,thresh2,1,cv.INPAINT_TELEA)
        cv.imwrite('static/img/bothat/res_img_'+f.filename,res_img)
        res_img_fname = 'static/img/bothat/res_img_' + f.filename
        return render_template('bothat.html', avg_img = avg_img_fname, laplacian_img =
laplacian_img_fname, subtracted_img = subtracted_img_fname, bothat_img = bothat_img_fname,
dilated_img = dilated_img_fname, res_img = res_img_fname)

    if request.form['algorithm'] == 'laplacian':
        init_img = cv.imread(f.filename)
        img = cv.imread(f.filename, cv.IMREAD_GRAYSCALE)
        laplacian_img = cv.Laplacian(img,cv.CV_64F)
        cv.imwrite('static/img/laplacian/laplacian_img_'+f.filename,laplacian_img)
        laplacian_img_fname = 'static/img/laplacian/laplacian_img_'+ f.filename
        subtracted_img = img - laplacian_img
        cv.imwrite('static/img/laplacian/subtracted_img_'+f.filename,subtracted_img)
        subtracted_img_fname = 'static/img/laplacian/subtracted_img_' + f.filename
        subtracted_img = cv.imread('static/img/laplacian/subtracted_img_'+f.filename,
cv.IMREAD_GRAYSCALE)
        blurred_img = cv.medianBlur(subtracted_img,3)
        cv.imwrite('static/img/laplacian/blurred_img_'+f.filename,blurred_img)
        blurred_img_fname = 'static/img/laplacian/blurred_img_' + f.filename
        log_edge_detection_img = cv.Canny(blurred_img, 100, 200)
        cv.imwrite('static/img/laplacian/log_edge_detection_img_'+f.filename,log_edge_detection_img)
        log_edge_detection_img_fname = 'static/img/laplacian/log_edge_detection_img_' + f.filename
        kernel = np.ones((5,5),np.uint8)
        close_img = cv.morphologyEx(log_edge_detection_img, cv.MORPH_CLOSE, kernel)
        cv.imwrite('static/img/laplacian/close_img_'+f.filename,close_img)
        close_img_fname = 'static/img/laplacian/close_img_'+ f.filename
        ret,thresh2 = cv.threshold(close_img, 10, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
        kernel = np.ones((1,1),np.uint8)
        dilated_img = cv.dilate(thresh2,kernel,iterations = 1)
        cv.imwrite('static/img/laplacian/dilated_img_'+f.filename,dilated_img)
        dilated_img_fname = 'static/img/laplacian/dilated_img_' + f.filename
        res_img = cv.inpaint(init_img,thresh2,1,cv.INPAINT_TELEA)
        cv.imwrite('static/img/laplacian/res_img_'+f.filename,res_img)
        res_img_fname = 'static/img/laplacian/res_img_' + f.filename
        return render_template('laplacian.html', laplacian_img = laplacian_img_fname, subtracted_img =
subtracted_img_fname, blurred_img = blurred_img_fname, log_edge_detection_img =
log_edge_detection_img_fname, close_img = close_img_fname, dilated_img = dilated_img_fname,
res_img = res_img_fname)

    if request.form['algorithm'] == 'log':
        init_img = cv.imread(f.filename)
        red_img = init_img.copy()
        red_img[:, :, 0] = 0
        red_img[:, :, 1] = 0
        cv.imwrite('static/img/log/red_img_'+f.filename,red_img)
```

```
        red_img_fname = 'static/img/log/red_img_'+ f.filename
        red_img = cv.imread(red_img_fname, cv.IMREAD_GRAYSCALE)
        cv.imwrite('static/img/log/red_img_gray_'+f.filename,red_img)
        red_img_fname = 'static/img/log/red_img_gray_'+ f.filename
        blur =  cv.GaussianBlur(red_img,(3,3),0)
        log_edge_detection_img = cv.Laplacian(blur,cv.CV_64F)
        cv.imwrite('static/img/log/log_edge_detection_img_'+f.filename,log_edge_detection_img)
        log_edge_detection_img = cv.imread('static/img/log/log_edge_detection_img_'+f.filename,
cv.IMREAD_GRAYSCALE)
        log_edge_detection_img_fname = 'static/img/log/log_edge_detection_img_' + f.filename
        kernel = np.ones((1,1),np.uint8)
        dilated_img = cv.dilate(log_edge_detection_img,kernel,iterations = 1)
        cv.imwrite('static/img/log/dilated_img_'+f.filename,dilated_img)
        dilated_img_fname = 'static/img/log/dilated_img_' + f.filename
        eroded_img = cv.erode(dilated_img, kernel, iterations =1)
        cv.imwrite('static/img/log/eroded_img_'+f.filename,eroded_img)
        eroded_img_fname = 'static/img/log/eroded_img_' + f.filename
        res_img = cv.inpaint(init_img,eroded_img,1,cv.INPAINT_TELEA)
        cv.imwrite('static/img/log/res_img_'+f.filename,res_img)
        res_img_fname = 'static/img/log/res_img_' + f.filename
        return render_template('log.html', red_img = red_img_fname, log_edge_detection_img =
log_edge_detection_img_fname, dilated_img = dilated_img_fname, eroded_img = eroded_img_fname,
res_img = res_img_fname)

    if request.form['algorithm'] == 'logsobel':
        init_img = cv.imread(f.filename)
        red_img = init_img.copy()
        red_img[:, :, 0] = 0
        red_img[:, :, 1] = 0
        cv.imwrite('static/img/logsobel/red_img_'+f.filename,red_img)
        red_img_fname = 'static/img/logsobel/red_img_'+ f.filename
        red_img = cv.imread(red_img_fname, cv.IMREAD_GRAYSCALE)
        cv.imwrite('static/img/logsobel/red_img_gray_'+f.filename,red_img)
        red_img_fname = 'static/img/logsobel/red_img_gray_'+ f.filename
        blur =  cv.GaussianBlur(red_img,(3,3),0)
        log_edge_detection_img = cv.Laplacian(blur,cv.CV_64F)
        cv.imwrite('static/img/logsobel/log_edge_detection_img_'+f.filename,log_edge_detection_img)
        log_edge_detection_img = cv.imread('static/img/logsobel/log_edge_detection_img_'+f.filename,
cv.IMREAD_GRAYSCALE)
        log_edge_detection_img_fname = 'static/img/logsobel/log_edge_detection_img_' + f.filename
        grad_x = cv.Sobel(log_edge_detection_img, cv.CV_16S, 1, 0, ksize=3, scale=1, delta=0,
borderType=cv.BORDER_DEFAULT)
        grad_y = cv.Sobel(log_edge_detection_img, cv.CV_16S, 0, 1, ksize=3, scale=1, delta=0,
borderType=cv.BORDER_DEFAULT)
        abs_grad_x = cv.convertScaleAbs(grad_x)
        abs_grad_y = cv.convertScaleAbs(grad_y)
        sobel_edge_detection_img = cv.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
```

```
cv.imwrite('static/img/logsobel/sobel_edge_detection_img_'+f.filename,sobel_edge_detection_img)
        sobel_edge_detection_img_fname = 'static/img/logsobel/sobel_edge_detection_img_'+ f.filename
        add_binary_img = log_edge_detection_img + sobel_edge_detection_img
        cv.imwrite('static/img/logsobel/add_binary_img_'+f.filename,add_binary_img)
        add_binary_img_fname = 'static/img/logsobel/add_binary_img_'+ f.filename
        kernel = np.ones((1,1),np.uint8)
        dilated_img = cv.dilate(add_binary_img,kernel,iterations = 1)
        cv.imwrite('static/img/logsobel/dilated_img_'+f.filename,dilated_img)
        dilated_img_fname = 'static/img/logsobel/dilated_img_' + f.filename
        res_img = cv.inpaint(init_img,dilated_img,1,cv.INPAINT_TELEA)
        cv.imwrite('static/img/logsobel/res_img_'+f.filename,res_img)
        res_img_fname = 'static/img/logsobel/res_img_' + f.filename
        return render_template('logsobel.html', red_img = red_img_fname, log_edge_detection_img =
log_edge_detection_img_fname, sobel_edge_detection_img = sobel_edge_detection_img_fname,
add_binary_img = add_binary_img_fname, dilated_img = dilated_img_fname, res_img =
res_img_fname)


    if request.form['algorithm'] == 'lls':
        init_img = cv.imread(f.filename)
        img = cv.imread(f.filename, cv.IMREAD_GRAYSCALE)
        laplacian_img = cv.Laplacian(img,cv.CV_64F)
        cv.imwrite('static/img/lls/laplacian_img_'+f.filename,laplacian_img)
        laplacian_img_fname = 'static/img/lls/laplacian_img_'+ f.filename
        subtracted_img = img - laplacian_img
        cv.imwrite('static/img/lls/subtracted_img_'+f.filename,subtracted_img)
        subtracted_img_fname = 'static/img/lls/subtracted_img_' + f.filename
        blur =  cv.GaussianBlur(subtracted_img,(3,3),0)
        log_edge_detection_img = cv.Laplacian(blur,cv.CV_64F)
        cv.imwrite('static/img/lls/log_edge_detection_img_'+f.filename,log_edge_detection_img)
        log_edge_detection_img = cv.imread('static/img/lls/log_edge_detection_img_'+f.filename,
cv.IMREAD_GRAYSCALE)
        log_edge_detection_img_fname = 'static/img/lls/log_edge_detection_img_' + f.filename
        grad_x = cv.Sobel(log_edge_detection_img, cv.CV_16S, 1, 0, ksize=3, scale=1, delta=0,
borderType=cv.BORDER_DEFAULT)
        grad_y = cv.Sobel(log_edge_detection_img, cv.CV_16S, 0, 1, ksize=3, scale=1, delta=0,
borderType=cv.BORDER_DEFAULT)
        abs_grad_x = cv.convertScaleAbs(grad_x)
        abs_grad_y = cv.convertScaleAbs(grad_y)
        sobel_edge_detection_img = cv.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
        cv.imwrite('static/img/lls/sobel_edge_detection_img_'+f.filename,sobel_edge_detection_img)
        sobel_edge_detection_img_fname = 'static/img/lls/sobel_edge_detection_img_'+ f.filename
        add_binary_img = log_edge_detection_img + sobel_edge_detection_img
        cv.imwrite('static/img/lls/add_binary_img_'+f.filename,add_binary_img)
        add_binary_img_fname = 'static/img/lls/add_binary_img_'+ f.filename
        input_img = cv.imread(add_binary_img_fname, cv.IMREAD_GRAYSCALE)
        wiener_filter = WienerFilter(input_img, (5,5))
```

```
        wiener_output_img = wiener_filter.estimateOutput()
        cv.imwrite('static/img/lls/wiener_output_img_'+f.filename,wiener_output_img)
        wiener_output_img_fname = 'static/img/lls/wiener_output_img_'+ f.filename
        wiener_output_img = cv.imread(wiener_output_img_fname, cv.IMREAD_GRAYSCALE)
        ret,thresh2 = cv.threshold(wiener_output_img, 10, 255, cv.THRESH_BINARY+cv.THRESH_OTSU)
        kernel = np.ones((1,1),np.uint8)
        dilated_img = cv.dilate(thresh2,kernel,iterations = 1)
        cv.imwrite('static/img/lls/dilated_img_'+f.filename,dilated_img)
        dilated_img_fname = 'static/img/lls/dilated_img_' + f.filename
        eroded_img = cv.erode(dilated_img, kernel, iterations =1)
        cv.imwrite('static/img/lls/eroded_img_'+f.filename,eroded_img)
        eroded_img_fname = 'static/img/lls/eroded_img_' + f.filename
        res_img = cv.inpaint(init_img,eroded_img,1,cv.INPAINT_TELEA)
        cv.imwrite('static/img/lls/res_img_'+f.filename,res_img)
        res_img_fname = 'static/img/lls/res_img_' + f.filename
        return render_template('lls.html', laplacian_img = laplacian_img_fname, subtracted_img =
subtracted_img_fname, log_edge_detection_img = log_edge_detection_img_fname,
sobel_edge_detection_img = sobel_edge_detection_img_fname, add_binary_img =
add_binary_img_fname, wiener_output_img = wiener_output_img_fname, dilated_img =
dilated_img_fname, eroded_img = eroded_img_fname, res_img = res_img_fname)

    return '<script>alert("The selected algorithm is not yet implemented");</script>'

if __name__ == '__main__':
  app.run(debug = True, host='0.0.0.0')
```

cv_filters.py[xv]

```
import cv2 as cv
import numpy as np

class WienerFilter():
  def __init__(self, input, filter_size):
    self.input = input
    self.filter_size = filter_size
    if(input.ndim != 2):
      print("Image not 2d")
    if(filter_size[0]%2 != 1 or filter_size[1]%2 != 1 or filter_size[0]<=1 or filter_size[1]<=1):
      print("Invalid filter dimension")

  def estimateOutput(self):
    means = cv.boxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size), anchor=(-1,1),
normalize=True, borderType=cv.BORDER_REPLICATE)
    square_means = cv.sqrBoxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size),
anchor=(-1,1), normalize=True, borderType=cv.BORDER_REPLICATE)

    means2 = cv.multiply(means, means)
```

```
    variances = square_means - means2

    avgVarianceMat = cv.reduce(variances, 1, cv.REDUCE_SUM, -1)
    avgVarianceMat = cv.reduce(avgVarianceMat, 0, cv.REDUCE_SUM, -1)

    noiseVar = np.asscalar(avgVarianceMat/self.input.shape[0] * self.input.shape[1])

    y = np.zeros(self.input.shape)
    for row in range(self.input.shape[0]):
      for col in range(self.input.shape[1]):
        y[row][col] = means[row][col] + max(0, variances[row][col] - noiseVar) * (self.input[row][col]-
means[row][col])/max(variances[row][col], noiseVar)

    return y

  def estimateOutputColorised(self):
        means = cv.boxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size), anchor=(-1,1),
normalize=True, borderType=cv.BORDER_REPLICATE)
    square_means = cv.sqrBoxFilter(src=self.input, ddepth=cv.CV_64F, ksize=tuple(self.filter_size),
anchor=(-1,1), normalize=True, borderType=cv.BORDER_REPLICATE)

    means2 = cv.multiply(means, means)
    variances = square_means - means2

    avgVariance = cv.reduce(variances, 0, cv.REDUCE_SUM, -1)
    avgVariance = cv.reduce(avgVariance, 1, cv.REDUCE_SUM, -1)

    noiseVar = avgVariance/self.input.shape[0] * self.input.shape[1] * self.input.shape[2]

    noiseVar = np.reshape(noiseVar, (1,3))

    y = np.zeros(self.input.shape)

    for row in range(self.input.shape[0]):
      for col in range(self.input.shape[1]):
        y[row][col] = self.saturate_cast(means[row][col] + np.maximum(np.zeros(noiseVar[0].shape),
variances[row][col] - noiseVar[0]) * (self.input[row][col]-
means[row][col])/np.maximum(variances[row][col], noiseVar[0]))

    return y

  def saturate_cast(self, val):

    if val.any() < 0:
      return np.array([255, 255, 255])
    elif val.any() >255:
      return np.array([255,255, 255])
    return val
```

bothat.html

```
<!doctype html>
<html class="no-js" lang="">

<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="apple-touch-icon" href="icon.png">
  <link rel="stylesheet" href="./static/css/normalize.css">
  <link rel="stylesheet" href="./static/css/main.css">
  <link rel="stylesheet" href="./static/css/bootstrap.min.css">
  <meta name="theme-color" content="#fafafa">
  <style>
  p,
label {
    font: 1rem 'Fira Sans', sans-serif;
}

input {
    margin: .4rem;
}
  </style>
</head>

<body>
<div style="padding-left:10px;">BOTHAT BASED ALGORITHM - STEPS:<br /></div>
<div style="padding-left:10px;">
<table style="border:none;">
<tr>
<td><img src="{{  avg_img  }}" alt="Avg Image" /></td>
<td><img src="{{  laplacian_img  }}" alt="Laplacian Image" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">1) AVERAGE FILTER TRANSFORMATION</td>
<td style="padding-top:5px; text-align:center;">2) LAPLACIAN TRANSFORMATION</td>
</tr>
<tr>
<td><img src="{{  subtracted_img  }}" alt="Subtracted Image" /></td>
<td><img src="{{  bothat_img  }}" alt="bothat_img Image" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">3) SUBTRACTED IMAGE</td>
<td style="padding-top:5px; text-align:center;">4) BOTHAT TRANSFORMATION</td>
</tr>
<tr>
<td><img src="{{  dilated_img  }}" alt="dilated_img Image" /></td>
```

```
<td><img src="{{  res_img  }}" alt="Final Image" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">5) DILATED IMAGE</td>
<td style="padding-top:5px; text-align:center;">6) FINAL IMAGE</td>
</tr>
</table>
</div>
</body>
</html>
```

laplacian.html

```
<!doctype html>
<html class="no-js" lang="">
<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="apple-touch-icon" href="icon.png">
  <link rel="stylesheet" href="./static/css/normalize.css">
  <link rel="stylesheet" href="./static/css/main.css">
  <link rel="stylesheet" href="./static/css/bootstrap.min.css">
  <meta name="theme-color" content="#fafafa">
  <style>
  p,
label {
   font: 1rem 'Fira Sans', sans-serif;
}

input {
   margin: .4rem;
}
  </style>
</head>

<body>
<div style="padding-left:10px;">LAPLACIAN BASED ALGORITHM - STEPS:<br /></div>
<div style="padding-left:10px;">
<table style="border:none;">
<tr>
<td><img src="{{  laplacian_img  }}" alt="Laplacian Image" /></td>
<td><img src="{{  subtracted_img  }}" alt="" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">1) LAPLACIAN TRANSFORMATION</td>
<td style="padding-top:5px; text-align:center;">2) SUBTRACTED IMAGE</td>
</tr>
```

```
<tr>
<td><img src="{{  blurred_img  }}" alt="Subtracted Image" /></td>
<td><img src="{{  log_edge_detection_img  }}" alt="LOG EDGE DETECTION IMAGE" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">3) BLURRED IMAGE</td>
<td style="padding-top:5px; text-align:center;">4) LOG EDGE DETECTION IMAGE</td>
</tr>
<tr>
<td><img src="{{  close_img  }}" alt="Image Closing" /></td>
<td><img src="{{  dilated_img  }}" alt="Image Dilation" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">5) IMAGE CLOSING</td>
<td style="padding-top:5px; text-align:center;">6) IMAGE DILATION</td>
</tr>
<tr>
<td><img src="{{  res_img  }}" alt="dilated_img Image" /></td>
<td></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">7) FINAL IMAGE</td>
<td style="padding-top:5px; text-align:center;"></td>
</tr>
</table>
</div>
</body>
</html>
```

lls.html

```
<!doctype html>
<html class="no-js" lang="">
<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="apple-touch-icon" href="icon.png">
  <link rel="stylesheet" href="./static/css/normalize.css">
  <link rel="stylesheet" href="./static/css/main.css">
  <link rel="stylesheet" href="./static/css/bootstrap.min.css">
  <meta name="theme-color" content="#fafafa">
  <style>
  p,
label {
    font: 1rem 'Fira Sans', sans-serif;
}
```

```
input {
   margin: .4rem;
}
  </style>
</head>
<body>
<div style="padding-left:10px;">LLS BASED ALGORITHM - STEPS:<br /></div>
<div style="padding-left:10px;">
<table style="border:none;">
<tr>
<td><img src="{{  laplacian_img  }}" alt="LAPLACIAN IMAGE" /></td>
<td><img src="{{  subtracted_img  }}" alt="SUBTRACTED IMAGE" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">1) LAPLACIAN IMAGE</td>
<td style="padding-top:5px; text-align:center;">2) SUBTRACTED IMAGE</td>
</tr>
<tr>
<td><img src="{{  log_edge_detection_img  }}" alt="LOG EDGE DETECTION IMAGE" /></td>
<td><img src="{{  sobel_edge_detection_img  }}" alt="SOBEL IMAGE" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">3) LOG EDGE DETECTION IMAGE</td>
<td style="padding-top:5px; text-align:center;">4) SOBEL EDGE IMAGE</td>
</tr>
<tr>
<td><img src="{{  add_binary_img  }}" alt="LOG+SOBEL IMAGE" /></td>
<td><img src="{{  wiener_output_img  }}" alt="WIENER FILTER IMAGE" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">5) LOG+SOBEL IMAGE</td>
<td style="padding-top:5px; text-align:center;">6) WIENER FILTER IMAGE</td>
</tr>
<tr>
<td><img src="{{  dilated_img  }}" alt="IMAGE DILATION" /></td>
<td><img src="{{  eroded_img  }}" alt="IMAGE EROSION" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">7) IMAGE DILATION</td>
<td style="padding-top:5px; text-align:center;">8) IMAGE EROSION</td>
</tr>
<tr>
<td><img src="{{  res_img  }}" alt="IMAGE DILATION" /></td>
<td></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">9) FINAL IMAGE</td>
<td style="padding-top:5px; text-align:center;"></td>
```

```
</tr>
</table>
</div>
</body>
</html>
```

log.html

```
<!doctype html>
<html class="no-js" lang="">
<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="apple-touch-icon" href="icon.png">
  <link rel="stylesheet" href="./static/css/normalize.css">
  <link rel="stylesheet" href="./static/css/main.css">
  <link rel="stylesheet" href="./static/css/bootstrap.min.css">
  <meta name="theme-color" content="#fafafa">
  <style>
  p,
label {
    font: 1rem 'Fira Sans', sans-serif;
}

input {
    margin: .4rem;
}
  </style>
</head>
<body>
<div style="padding-left:10px;">LOG BASED ALGORITHM - STEPS:<br /></div>
<div style="padding-left:10px;">
<table style="border:none;">
<tr>
<td><img src="{{  red_img  }}" alt="RED CHANNEL OF IMAGE" /></td>
<td><img src="{{  log_edge_detection_img  }}" alt="LOG EDGE DETECTION IMAGE" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">1) RED CHANNEL OF IMAGE</td>
<td style="padding-top:5px; text-align:center;">2) LOG EDGE DETECTION IMAGE</td>
</tr>
<tr>
<td><img src="{{  dilated_img  }}" alt="IMAGE DILATION" /></td>
<td><img src="{{  eroded_img  }}" alt="IMAGE EROSION" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">3) IMAGE DILATION</td>
```

```
<td style="padding-top:5px; text-align:center;">4) IMAGE EROSION</td>
</tr>
<tr>
<td><img src="{{  res_img  }}" alt="FINAL IMAGE" /></td>
<td></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">5) FINAL IMAGE</td>
<td style="padding-top:5px; text-align:center;"></td>
</tr>
</table>
</div>
</body>
</html>
```

logsobel.html

```
<!doctype html>
<html class="no-js" lang="">
<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="apple-touch-icon" href="icon.png">
  <link rel="stylesheet" href="./static/css/normalize.css">
  <link rel="stylesheet" href="./static/css/main.css">
  <link rel="stylesheet" href="./static/css/bootstrap.min.css">
  <meta name="theme-color" content="#fafafa">
  <style>
  p,
label {
    font: 1rem 'Fira Sans', sans-serif;
}

input {
    margin: .4rem;
}
  </style>
</head>
<body>
<div style="padding-left:10px;">LOGSOBEL BASED ALGORITHM - STEPS:<br /></div>
<div style="padding-left:10px;">
<table style="border:none;">
<tr>
<td><img src="{{  red_img  }}" alt="RED CHANNEL OF IMAGE" /></td>
<td><img src="{{  log_edge_detection_img  }}" alt="LOG EDGE DETECTION IMAGE" /></td>
</tr>
<tr>
```

```
<td style="padding-top:5px; text-align:center;">1) RED CHANNEL OF IMAGE</td>
<td style="padding-top:5px; text-align:center;">2) LOG EDGE DETECTION IMAGE</td>
</tr>
<tr>
<td><img src="{{  sobel_edge_detection_img  }}" alt="SOBEL EDGE DETECTION IMAGE" /></td>
<td><img src="{{  add_binary_img  }}" alt="LOG+SOBEL IMAGE" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">3) SOBEL EDGE DETECTION IMAGE</td>
<td style="padding-top:5px; text-align:center;">4) LOG+SOBEL IMAGE</td>
</tr>
<tr>
<td><img src="{{  dilated_img  }}" alt="IMAGE DILATION" /></td>
<td><img src="{{  res_img  }}" alt="FINAL IMAGE" /></td>
</tr>
<tr>
<td style="padding-top:5px; text-align:center;">5) IMAGE DILATION</td>
<td style="padding-top:5px; text-align:center;">6) FINAL IMAGE</td>
</tr>
</table>
</div>
</body>
</html>
```

6. ΒΙΒΛΙΟΓΡΑΦΙΑ – ΠΑΡΑΠΟΜΠΕΣ

1) Learning OpenCV, 2nd Edition, Adrian Kaehler & Gary Bradski, O'Reilly
2) https://www.researchgate.net/publication/301590571_OpenCV_for_Computer_Vision_Applications
3) https://www.cs.ccu.edu.tw/~damon/photo/,OpenCV/,Mastering_OpenCV.pdf
4) https://docs.opencv.org/master/d7/da8/tutorial_table_of_content_imgproc.html
5) https://flask.palletsprojects.com/en/1.1.x/
6) https://exploreflask.com/en/latest/

[i] Emgu Framework: http://www.emgu.com/
[ii] http://pages.stat.wisc.edu/~mchung/teaching/MIA/reading/diffusion.gaussian.kernel.pdf.pdf
[iii] https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html
[iv] https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html
[v] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_gradients/py_gradients.html
[vi] https://flask.palletsprojects.com/en/1.1.x/tutorial/#tutorial
[vii] NumPy Library https://numpy.org/doc/stable/user/tutorials_index.html
[viii] OpenCV – Canny Edge Detector https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html
[ix] OpenCV – Sobel Derivatives https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html
[x] Βασικές αρχιτεκτονικές προγραμματισμού σε αντικειμενοστρεφές περιβάλλον – Object Oriented Programming - Python (OOP in Python) - https://home.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/li.pdf
[xi] Flask installation - https://flask.palletsprojects.com/en/1.1.x/installation/
[xii] Python PIP Package Installation Management - https://pip.pypa.io/en/stable/installing/
[xiii] OMPC – Open-Source MATLAB-to-Python Compiler - http://ompc.juricap.com/
[xiv] GNU Octave - https://octave.org/doc/v6.1.0/
[xv] https://github.com/rishiraj824/adaptive_wiener_filters