ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Δημιουργία εφαρμογής κινητού τηλεφώνου με επεξεργασία εικόνας - edge detection και απτική αλληλεπίδραση για άτομα με προβλήματα όρασης»

Φοιτητής:
Ζερβόγλου
Δημήτριος Κωνσταντίνος
Αρ. Μητρώου: 2019045

Επιβλέπων:
Κοκκώνης Γεώργιος
Επίκουρος Καθηγητής

Θεσσαλονίκη 2026

Δημιουργία εφαρμογής κινητού τηλεφώνου με επεξεργασία εικόνας - edge detection και απτική αλληλεπίδραση για άτομα με προβλήματα όρασης

Κωδικός Δ.Ε. : 23347

Ονοματεπώνυμο φοιτητή: Ζερβόγλου Δημήτριος Κωνσταντίνος

Ονοματεπώνυμο εισηγητή : Κοκκώνης Γεώργιος

Ημερομηνία ανάληψης Δ.Ε. : 08/10/2024

Ημερομηνία περάτωσης Δ.Ε. : 30/05/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Στους αγαπημένους μου»

Πρόλογος

Η επιλογή του θέματος της παρούσας Δ.Ε. προήλθε από το ενδιαφέρον μου για την Τεχνητή Νοημοσύνη και την ανάπτυξη κινητών εφαρμογών. Μέσα από αυτά ήταν που εξοικειώθηκα περισσότερο με τους όρους του Computer Vision, Edge AI, και την προσβασιμότητα στις κινητές συσκευές. Η ενασχόληση με μοντέρνες αλλά και πολύπλοκες τεχνολογίες όπως το YOLO και TensorFlow, αποτέλεσε σημαντική πρόκληση, μέσα από την οποία εξέλιξα τις προγραμματιστικές μου ικανότητες.

Περίληψη

Η παρούσα Δ.Ε. επικεντρώνεται στον σχεδιασμό, την υλοποίηση και την αξιολόγηση μιας εφαρμογής Android για άτομα με προβλήματα όρασης. Η εφαρμογή χρησιμοποιεί την πίσω κάμερα του κινητού για την ανίχνευση αντικειμένων σε πραγματικό χρόνο μέσω του μοντέλου YOLOv26n-seg. Ο χρήστης εξερευνά τα αντικείμενα γύρω του σύροντας το δάχτυλό του στην οθόνη. Όταν το δάχτυλο πλησιάσει το περίγραμμα ενός ανιχνευμένου αντικειμένου, ενεργοποιείται η δόνηση. Ο εντοπισμός ακμών δεν βασίζεται στα πλαίσια οριοθέτησης που παράγει το μοντέλο, αλλά στις μάσκες τμηματοποίησης. Παράλληλα, εκφωνείται το όνομα του αντικειμένου με TTS. Επιπλέον, υπάρχει δυνατότητα φωνητικής αλληλεπίδρασης. Οι παράμετροι δόνησης είναι ρυθμιζόμενες, ώστε η εφαρμογή να προσαρμόζεται στις ανάγκες του χρήστη.

Έπειτα από αξιολόγηση της εφαρμογής με χρήση του UEQ ερωτηματολογίου, η εφαρμογή αξιολογήθηκε από τους είκοσι συμμετέχοντες θετικά ως προς τη σαφήνεια λειτουργίας και το ενδιαφέρον που προκαλεί, ενώ αναδείχθηκε η ανάγκη επέκτασης του αριθμού αναγνωρίσιμων αντικειμένων.

«Developing a Mobile Application using Image Processing (Edge Detection) and Haptic Feedback for the Visually Impaired.»

«Zervoglou Dimitrios Konstantinos»

Abstract

This thesis focuses on the design, implementation, and evaluation of an Android application for visually impaired users. The application uses the device's rear camera to detect objects in real time through the YOLOv26n-seg model. The user explores the surrounding objects by sliding a finger across the screen. When the finger approaches the contour of a detected object, vibration is activated. Edge detection is not based on the bounding boxes produced by the model, but on the segmentation masks. At the same time, the object's name is announced via TTS. Additionally, voice interaction is supported. The vibration parameters are configurable, allowing the application to adapt to the user's individual needs.

Following an evaluation of the application using the UEQ questionnaire, the twenty participants rated the application positively in terms of clarity of function and the engagement it generates, while the need for expanding the number of recognizable objects was identified.

Ευχαριστίες

Θέλω να ευχαριστήσω την οικογένεια μου για την συμπαράσταση κατά την συγγραφή της διπλωματικής εργασίας αλλά και σε όλη την διάρκεια των σπουδών μου. Επιπλέον, ευχαριστώ τον επιβλέπων καθηγητή για την καθοδήγηση και την υπομονή που έδειξε.

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Εικόνων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
TTS	Text-To-Speech
YOLO	You Only Look Once

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Τα τελευταία χρόνια, οι υποστηρικτικές τεχνολογίες για άτομα με προβλήματα όρασης έχουν εξελιχθεί ραγδαία, από απλά εργαλεία υλικού, σε προηγμένα συστήματα τεχνητής νοημοσύνης. Τα παραπάνω έχουν ως αποτέλεσμα μεγαλύτερη ανεξαρτησία, ασφάλεια, και ποιότητα ζωής[15]. Μεταξύ των υποστηρικτικών λύσεων, οι εφαρμογές για κινητά τηλέφωνα αναδύονται ως ιδιαίτερα προσιτή κατηγορία, καθώς παρέχουν κινητικότητα και ευκολία χρήσης[15]. Τα σύγχρονα smartphones διαθέτουν πλέον επαρκή υπολογιστική ισχύ για την εκτέλεση μοντέλων ανίχνευσης αντικειμένων σε πραγματικό χρόνο [17] και χωρίς εξάρτηση από σύνδεση στο διαδίκτυο[15][18].

Στην παρούσα Δ.Ε. παρουσιάζεται η ανάπτυξη μιας Android εφαρμογής για άτομα με προβλήματα όρασης, η οποία:

- αξιοποιεί το πλέον πρόσφατο μοντέλο **YOLO26** για ανίχνευση αντικειμένων σε πραγματικό χρόνο
- παρέχει απτική ανατροφοδότηση κατά την εξερεύνηση
- εκφωνεί με ελληνικό Text-To-Speech τα αντικείμενα
- λειτουργεί εκτός σύνδεσης

1.2 Περιγραφή του Προβλήματος

Τουλάχιστον 2.2 δισεκατομμύρια άνθρωποι στον κόσμο αντιμετωπίζουν προβλήματα όρασης κοντινής ή μακρινής απόστασης ή τύφλωσης [19]. Η όραση διαδραματίζει καθοριστικό ρόλο σε κάθε πτυχή και στάδιο της ζωής.
Σύμφωνα με τον παγκόσμιο οργανισμό υγείας, ενήλικες με προβλήματα όρασης παρουσιάζουν χαμηλότερα ποσοστά απασχόλησης και υψηλότερα ποσοστά κατάθλιψης και άγχους. Αντίστοιχα οι ηλικιωμένοι αντιμετωπίζουν κοινωνική απομόνωση, μεγαλύτερη δυσκολία στην κίνηση και κίνδυνο πτώσης[19].

Η ασφαλής πλοήγηση αποτελεί μία από τις μεγαλύτερες προκλήσεις που συναντά ένα άτομο με μερική ή ολική απώλεια όρασης[15]. Ωστόσο, η ανάπτυξη κατάλληλων υποστηρικτικών συσκευών και εφαρμογών μπορεί να ενισχύσει σημαντικά την αυτονομία τους στην καθημερινή ζωή[15].

Η οπτική αναπηρία εμφανίζεται όταν μια οφθαλμική πάθηση επηρεάζει το οπτικό σύστημα ή μία ή περισσότερες από τις λειτουργίες της όρασης. Ο όρος «αναπηρία» αναφέρεται στις δυσκολίες, τους περιορισμούς και τα εμπόδια που αντιμετωπίζει ένα άτομο με οφθαλμική πάθηση κατά την αλληλεπίδραση του με το περιβάλλον του. Είτε είναι το φυσικό είτε είναι το κοινωνικό ή συμπεριφορικό[19].
Η γήρανση αποτελεί τον κύριο παράγοντα κινδύνου για πολλές οφθαλμικές παθήσεις όπως: η πρεσβυωπία, ο καταρράκτης, το γλαύκωμα και η ηλικιακή εκφύλιση της ωχράς κηλίδας[19]. Σύμφωνα με πρόσφατη ανασκόπηση, κάθε άνθρωπος, εφόσον ζήσει αρκετά, θα αντιμετωπίσει τουλάχιστον μία οφθαλμική πάθηση κατά την διάρκεια της ζωής του[15][19].

Η εξασθένηση της όρασης έχει βαθιές συνέπειες από τις μικρές ηλικίες μέχρι και τις μεγάλες. Στις μικρές ηλικίες, μικρά παιδιά που ταλαιπωρούνται με πρώιμη έναρξη εξασθένησης της όρασης παρουσιάζουν καθυστέρηση στην ανάπτυξη των γλωσσικών, κοινωνικών και γνωστικών ικανοτήτων τους[19].

1.3 Σκοπός της εργασίας

Η παρούσα εργασία αποσκοπεί στην ανάπτυξη εφαρμογής Android που δίνει τη δυνατότητα σε άτομα με προβλήματα όρασης να αντιλαμβάνονται τα αντικείμενα γύρω τους μέσω της αφής και του ήχου. Το βασικό χαρακτηριστικό της εφαρμογής είναι ότι ο χρήστης εξερευνά τη σκηνή σύροντας το δάχτυλό του στην οθόνη. Όταν πλησιάζει το περίγραμμα ενός αντικειμένου, η συσκευή δονείται.

Παράλληλα, η εφαρμογή εκφωνεί στα ελληνικά το όνομα του αντικειμένου μέσω Text-To-Speech. Η ανίχνευση πραγματοποιείται σε πραγματικό χρόνο μέσω του μοντέλου **YOLO26** σε μορφή **TensorFlow Lite**, κάτι που δεν απαιτεί την σύνδεση στο διαδίκτυο.

Για την ενεργοποίηση των ρυθμίσεων, ο χρήστης εκτελεί τέσσερα διαδοχικά πατήματα. Αυτά ενεργοποιούν την φωνητική αναγνώριση φωνητικών εντολών. Η χειρονομία τεσσάρων πατημάτων επιλέχθηκε ώστε να αποφεύγεται η κατά λάθος ενεργοποίηση κατά την εξερεύνηση της οθόνης με το δάχτυλο.

1.4 Δομή Δ.Ε.

Η εργασία διαρθρώνεται σε έξι κεφάλαια.

Στο πρώτο κεφάλαιο παρουσιάζεται το πρόβλημα της οπτικής αναπηρίας, η αναγκαιότητα υποστηρικτικών τεχνολογιών και οι στόχοι της εργασίας.

Στο δεύτερο κεφάλαιο αναπτύσσεται το θεωρητικό πλαίσιο: τα συνελικτικά νευρωνικά δίκτυα, η αρχιτεκτονική YOLO, η τμηματοποίηση στιγμιοτύπου, η απτική αλληλεπίδραση καθώς και επισκόπηση σχετικών εργασιών.

Στο τρίτο κεφάλαιο περιγράφονται τα εργαλεία και οι τεχνολογίες που χρησιμοποιήθηκαν, όπως το **Android Studio**, η **Kotlin**, η **CameraX**, το **YOLO26** με **TensorFlow Lite**, το **Vibrator API**, και το **Text-To-Speech**.

Στο τέταρτο κεφάλαιο παρουσιάζεται αναλυτικά η υλοποίηση της εφαρμογής. Συγκεκριμένα, αναλύεται η εκκίνηση και αρχικοποίηση, η γραφική διεπαφή, η αλυσίδα επεξεργασίας βίντεο, ο αλγόριθμος εντοπισμού ακμών και απτικής ανατροφοδότησης, η φωνητική αλληλεπίδραση, η διαχείριση των ρυθμίσεων και συνολικά η δομή του κώδικα.

Στο πέμπτο κεφάλαιο περιγράφονται οι δοκιμές, τα αποτελέσματα, και η αξιολόγηση από τους χρήστες.

Στο έκτο κεφάλαιο διατυπώνονται τα συμπεράσματα και οι προοπτικές για μελλοντικές επεκτάσεις.

1.5 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκε το πρόβλημα της οπτικής αναπηρίας, η κλίμακα του σε παγκόσμιο επίπεδο και οι συνέπειές του στην καθημερινή ζωή. Διατυπώθηκαν οι στόχοι της εφαρμογής και η δομή της εργασίας. Στο επόμενο κεφάλαιο αναπτύσσεται το θεωρητικό υπόβαθρο των τεχνολογιών που χρησιμοποιούνται.

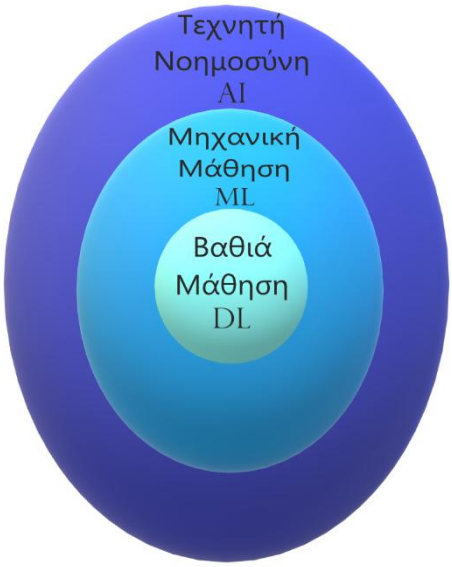
Κεφάλαιο 2ο: Θεωρητικό Πλαίσιο

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο στο οποίο στηρίζεται η εφαρμογή. Εξετάζονται οι βασικές έννοιες της τεχνητής νοημοσύνης, της υπολογιστικής όρασης, και η αρχιτεκτονική **YOLO** για ανίχνευση αντικειμένων και τμηματοποίηση. Τέλος, εξετάζεται η απτική αλληλεπίδραση και σχετικές εργασίες.

2.2 Τεχνητή Νοημοσύνη και Υπολογιστική Όραση

Η τεχνητή νοημοσύνη (Artificial Intelligence, AI) αναφέρεται σε ένα σύνολο εννοιών και αλγορίθμων που στοχεύουν στην αναπαραγωγή της λογικής του ανθρώπινου εγκεφάλου [34]. Οι κύριες εφαρμογές της AI είναι η μηχανική μάθηση (Machine Learning, ML) και η βαθιά μάθηση (Deep Learning, DL) [34]. Η μηχανική μάθηση αποτελεί υποσύνολο της AI και επιτρέπει στις εφαρμογές λογισμικού να προβλέπουν αποτελέσματα με μεγαλύτερη ακρίβεια, χωρίς να απαιτείται ρητός προγραμματισμός, βασιζόμενη σε ιστορικά δεδομένα ως είσοδο [34]. Οι υποκατηγορίες της είναι η εποπτευόμενη (Supervised Learning), η μη εποπτευόμενη (Unsupervised Learning) και η ημι-εποπτευόμενη μάθηση (Semi-Supervised Learning) [34].



Εικόνα 2.1

Η βαθιά μάθηση επιτρέπει σε υπολογιστικά μοντέλα πολλαπλών επιπέδων να μαθαίνουν και να αναπαριστούν δεδομένα με πολλαπλά επίπεδα αφαίρεσης, μιμούμενη τον τρόπο με τον οποίο ο ανθρώπινος εγκέφαλος αντιλαμβάνεται πληροφορίες [1]. Η βαθιά μάθηση έχει επιτύχει σημαντικά αποτελέσματα σε προβλήματα υπολογιστικής όρασης (Computer Vision), όπως η ανίχνευση αντικειμένων, η αναγνώριση προσώπων και η σημασιολογική τμηματοποίηση [1].

2.3 Συνελικτικά Νευρωνικά Δίκτυα (CNNs)

Τα Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks, CNNs) αποτελούν την κύρια αρχιτεκτονική βαθιάς μάθησης για την επεξεργασία εικόνων. Η δομή τους είναι εμπνευσμένη από την οργάνωση του οπτικού φλοιού των θηλαστικών, όπου μεμονωμένοι νευρώνες ανταποκρίνονται σε ερεθίσματα μικρών περιοχών του οπτικού πεδίου[2]. Ένα CNN εφαρμόζει μαθηματικές πράξεις συνέλιξης (convolution) σε μικρές περιοχές της εικόνας, εξάγοντας σταδιακά χαρακτηριστικά αυξανόμενης πολυπλοκότητας. Από απλές ακμές και υφές στα πρώτα επίπεδα, έως σύνθετα σχήματα και ολόκληρα αντικείμενα στα βαθύτερα[2]. Βασικό πλεονέκτημα της αρχιτεκτονικής είναι ο διαμοιρασμός βαρών (shared weights). Δηλαδή, το ίδιο φίλτρο εφαρμόζεται σε κάθε περιοχή της εικόνας, μειώνοντας δραστικά τον αριθμό παραμέτρων και καθιστώντας τα CNNs κατάλληλα για εκτέλεση σε συσκευές περιορισμένων πόρων, όπως τα κινητά τηλέφωνα[2].

Οι παραδοσιακές μέθοδοι ανίχνευσης αντικειμένων, όπως τα Deformable Parts Models (DPM) και το R-CNN, βασίζονταν σε πολυσταδιακές αρχιτεκτονικές. Πρώτα παράγονταν υποψήφιες περιοχές, στη συνέχεια εφαρμόζονταν ταξινομητές σε κάθε περιοχή ξεχωριστά, και τέλος γίνονταν διορθώσεις στα πλαίσια οριοθέτησης. Η πολυπλοκότητα αυτών των αρχιτεκτονικών τις καθιστούσε αργές και δύσκολες στη βελτιστοποίηση[3].

2.4 Ανίχνευση Αντικειμένων και η Αρχιτεκτονική YOLO

Το 2016, οι Redmon et al. παρουσίασαν μια διαφορετική προσέγγιση με την ονομασία YOLO (You Only Look Once). Σε αντίθεση με άλλες μεθόδους, το YOLO αντιμετωπίζει την ανίχνευση αντικειμένων ως έναν ενιαίο πρόβλημα παλινδρόμησης (regression). Πιο συγκεκριμένα, ένα μόνο συνελικτικό νευρωνικό δίκτυο επεξεργάζεται ολόκληρη την εικόνα σε ένα πέρασμα και προβλέπει ταυτόχρονα τα πλαίσια οριοθέτησης (bounding boxes) και τις πιθανότητες κλάσης[3]. Συγκεκριμένα, το YOLO διαιρεί την εικόνα εισόδου σε ένα πλέγμα S×S κελιών. Κάθε κελί είναι υπεύθυνο για την ανίχνευση αντικειμένων των οποίων το κέντρο βρίσκεται εντός του, προβλέποντας ταυτόχρονα τα πλαίσια οριοθέτησης και τις αντίστοιχες βαθμολογίες εμπιστοσύνης[3].
Το αρχικό μοντέλο YOLO επεξεργαζόταν εικόνες με ρυθμό 45 καρέ ανά δευτερόλεπτο (fps), ενώ η ελαφρύτερη εκδοχή του ξεπερνούσε τα 150 fps. Αυτές ήταν οι επιδόσεις που κατέστησαν εφικτή την ανίχνευση αντικειμένων σε πραγματικό χρόνο σε ροή βίντεο (<25ms latency)[3].

Η αρχιτεκτονική YOLO αποτελείται από τρία βασικά μέρη: τον κορμό (backbone) που εξάγει χαρακτηριστικά από την εικόνα σε πολλαπλές κλίμακες, τον λαιμό (neck) που συγκεντρώνει και ενισχύει τις αναπαραστάσεις χαρακτηριστικών, και την κεφαλή (head) που παράγει τις τελικές προβλέψεις θέσης και κλάσης[4].
Από το 2016 έως σήμερα, η αρχιτεκτονική YOLO έχει εξελιχθεί μέσα από πολλές διαδοχικές εκδόσεις, με κάθε επανάληψη να βελτιώνει την ακρίβεια, την ταχύτητα ή και τα δύο. Χαρακτηριστικό παράδειγμα αποτελεί η έκδοση YOLOv11, καθώς επιτυγχάνει υψηλότερη μέση ακρίβεια (mean Average Precision, mAP) στο σύνολο δεδομένων COCO με 22% λιγότερες παραμέτρους σε σύγκριση με το YOLOv8, ενώ παράλληλα προσφέρεται σε πέντε μεγέθη (nano έως extra large). Έτσι, γίνεται δυνατή η χρήση του ακόμη και σε συσκευές περιορισμένων πόρων[14].
Η εφαρμογή αξιοποιεί τη πιο πρόσφατη έκδοση YOLO26. Συγκεκριμένα, την παραλλαγή **YOLOv26n-seg**, δηλαδή τη nano έκδοση με υποστήριξη τμηματοποίησης στιγμιότυπου (instance segmentation). Η

επιλογή αυτή εξυπηρετεί δύο κρίσιμες απαιτήσεις. Το μέγεθος nano εξασφαλίζει εκτέλεση σε πραγματικό χρόνο σε κινητό τηλέφωνο και η τμηματοποίηση παρέχει μάσκες σε επίπεδο pixel αντί για απλά πλαίσια οριοθέτησης. Οι μάσκες τμηματοποίησης αποτελούν απαραίτητη προϋπόθεση για τον εντοπισμό άκρων που χρησιμοποιεί ο μηχανισμός απτικής ανατροφοδότησης.

2.5 Τμηματοποίηση Στιγμιότυπου

Η ανίχνευση αντικειμένων (object detection) εντοπίζει κάθε αντικείμενο μέσω ενός ορθογώνιου πλαισίου οριοθέτησης (bounding box)[5][22]. Η σημασιολογική τμηματοποίηση (semantic segmentation) ταξινομεί κάθε pixel της εικόνας σε μια κλάση, αλλά δεν διαχωρίζει μεμονωμένα αντικείμενα μεταξύ τους. Η τμηματοποίηση στιγμιότυπου (instance segmentation) συνδυάζει τα πλεονεκτήματα και των δύο: εκτελεί ταυτόχρονα ανίχνευση και τμηματοποίηση, παρέχοντας μάσκα σε επίπεδο pixel για κάθε αντικείμενο ξεχωριστά[5][22]. Επιπλέον, αποδίδει ένα ID για κάθε διαφορετικό αντικείμενο, ακόμη και αν ανήκουν στην ίδια κλάση.
Αυτή η διάκριση είναι ιδιαίτερα σημαντική σε σκηνές με αλληλεπικαλυπτόμενα αντικείμενα. Η απλή σημασιολογική τμηματοποίηση δεν μπορεί να τα διαχωρίσει, οπότε σε τέτοιες περιπτώσεις παράγει συγχωνευμένες μάσκες. Στην εφαρμογή κρίνεται απαραίτητη η ιδιότητα του instance segmentation να ξεχωρίζει αντικείμενα σε μοναδικές αντιστοιχίες.

Η τμηματοποίηση στιγμιότυπου χωρίζεται σε δύο στάδια:

- Πρώτο στάδιο: ανίχνευση αντικειμένου. Εντοπίζει κάθε αντικείμενο και παράγει ένα πλαίσιο οριοθέτησης (bounding box) γύρω από αυτό.
- Δεύτερο στάδιο: τμηματοποίηση. Για κάθε πλαίσιο που ανιχνεύει, εκτελείται δυαδική τμηματοποίηση (binary segmentation) στην περιοχή εντός του πλαισίου[5].

Η διαδοχική εκτέλεση αυτών των δύο σταδίων σημαίνει υψηλό υπολογιστικό κόστος, μεγάλο μέγεθος μοντέλου και αυξημένο αριθμό παραμέτρων. Το YOLO26 αντιμετωπίζει αυτό το πρόβλημα εκτελώντας ανίχνευση και τμηματοποίηση σε ένα πέρασμα από το νευρωνικό δίκτυο, αποφεύγοντας έτσι το υψηλό υπολογιστικό κόστος των παραδοσιακών προσεγγίσεων [14].

2.6 Απτική Αλληλεπίδραση

Η απτική αίσθηση (haptic sense) αποτελεί σημαντικό τρόπο αντίληψης για άτομα με προβλήματα όρασης. Υπάρχουν απτικά υποστηρικτικά εργαλεία που παρέχουν ανατροφοδότηση σε ποικίλες θέσεις του σώματος (δάκτυλα, χέρια, καρπούς, μέση κλπ)[6]. Η κατανομή των μηχανοϋποδοχέων δεν είναι ομοιόμορφη: τα δάκτυλα εμφανίζουν τον βέλτιστο δείκτη διαδοχικής διάκρισης δύο σημείων στα 0,3cm, ακολουθούμενα από την παλάμη (0,4cm) και τη ράχη του χεριού (0,9cm)[6]. Σε σύγκριση με την ηχητική ανατροφοδότηση, η απτική προσφέρει πιο διαισθητική κατανόηση γραφικής και χωρικής πληροφορίας, και για αυτό χρησιμοποιείται ολοένα και περισσότερο σε εφαρμογές για άτομα με προβλήματα όρασης[6].

Στο [6] περιγράφονται πέντε κύριοι τύποι απτικής ανατροφοδότησης:

- Πίεση (Pressure): Braille
- Κινητική (Kinesthetic feedback)

- Δόνηση (Vibration)
- Διάτμηση δέρματος (skin-stretch)
- Θερμική (Thermal feedback)

Από τα παραπάνω, καταλαβαίνει κανείς ότι η δόνηση αποτελεί πλέον διαδεδομένο τύπο σε εφαρμογές κινητών συσκευών, λόγω χαμηλής κατανάλωσης ισχύος και εύκολης ενσωμάτωσης[6].

Μάλιστα σε σχετική μελέτη, τυφλοί χρήστες αξιολόγησαν κατά περίπου 18% υψηλότερα την χρησιμότητα της εφαρμογής με απτική ανατροφοδότηση. Παράλληλα, ο δείκτης ικανοποίησης έφτασε στο 90% με δόνηση, έναντι 71% χωρίς δόνηση[7].

2.7 Σχετικές Εργασίες

Η ανάπτυξη εφαρμογών και γενικότερα υποστηρικτικών τεχνολογιών για άτομα με προβλήματα όρασης απαιτεί ταχύτητα και αξιοπιστία από το σύστημα. Αρκετές υφιστάμενες εφαρμογές εξαρτώνται από μοντέλα που βρίσκονται στο cloud. Αν και αποδοτικές, η παραπάνω υλοποίηση εξαρτάται από την σύνδεση στο διαδίκτυο. Ως αποτέλεσμα, οποιοδήποτε πρόβλημα σύνδεσης στο διαδίκτυο μεταφράζεται σε καθυστέρηση στην απόκριση του συστήματος[23]. Παράλληλα, αυτή η καθυστέρηση (latency) αποτελεί όχι μόνο θέμα ευχρηστίας, αλλά και κίνδυνο προς τον χρήστη. Σε περίπτωση που η πληροφορία φτάσει στην εφαρμογή με καθυστέρηση, τα δεδομένα που λαμβάνει ο χρήστης είναι ήδη παρωχημένα, γεγονός εξαιρετικά επικίνδυνο κατά την κίνηση του χρήστη στον χώρο[24]. Για την αποφυγή τέτοιων δυσλειτουργιών, υπάρχει η ανάγκη μετάβασης από το cloud σε μοντέλα που τρέχουν στη συσκευή. Αυτό εξασφαλίζει ταχύτητα, ιδιωτικότητα και αξιοπιστία[25].

Εξίσου σημαντική παράμετρος σε σχετικές εφαρμογές είναι η χρήση της ηχητικής ανατροφοδότησης. Κάποιες εφαρμογές βασίζονται αποκλειστικά σε αυτό. Η προσέγγιση αυτή σύμφωνα με το [21] παρουσιάζει μειονεκτήματα. Αρχικά, παρεμποδίζει την ακοή. Η ακοή αποτελεί ίσως τον πιο κρίσιμο αισθητηριακό μηχανισμό που διαθέτουν τα άτομα με προβλήματα όρασης για την χαρτογράφηση του χώρου[21]. Επιπλέον, σε θορυβώδη εξωτερικά περιβάλλοντα, τα ηχητικά μηνύματα είναι πρακτικά αναποτελεσματικά ή δυσνόητα[26]. Εκτός απ' αυτό, η διαρκής επεξεργασία ηχητικών οδηγιών αυξάνει το γνωστικό φορτίο και τη διάσπαση προσοχής, οδηγώντας τον χρήστη σε παρατεταμένο στρες και νοητική κόπωση[27].

Ακόμα μία παράμετρος αποτελεί η **απτική ανατροφοδότηση** (haptic feedback). Ωστόσο, αρκετές εφαρμογές εξαρτώνται από τα πλαίσια οριοθέτησης (bounding boxes) σε αλγορίθμους Object Detection. Αν και χρειάζεται λιγότερη υπολογιστική ισχύ σε σχέση με άλλους αλγορίθμους, η χρήση αυτών των πλαισίων για την ενεργοποίηση των δονήσεων δημιουργεί ασάφεια και σύγχυση. Ειδικότερα όταν τα αντικείμενα είναι αλληλεπικαλυπτόμενα[25]. Τα πλαίσια αυτά ομαδοποιούν το πραγματικό αντικείμενο μαζί με διαφορετικά πράγματα στο παρασκήνιο (από πίσω του ή από δίπλα του). Έτσι ο χρήστης δεν καταλαβαίνει την ακριβή γεωμετρία του εμποδίου[24]. Αντιθέτως η τμηματοποίηση στιγμιότυπου (Instance Segmentation), επιτρέπει την παραγωγή μασκών απόλυτης ακρίβειας σε επίπεδο pixel για κάθε αντικείμενο. Γεγονός που καθιστά ένα τέτοιου είδους μοντέλο σαφώς πιο απαιτητικό σε υπολογιστική ισχύ, αλλά και πιο έμπιστο για χωρική αλληλεπίδραση[28].

Στην παρούσα εφαρμογή, χρησιμοποιείται το ακριβές περίγραμμα της μάσκας των αντικειμένων για απτική αλληλεπίδραση. Συγκεκριμένα, ενεργοποιεί μοτίβα δόνησης αποκλειστικά όταν το δάκτυλο του

χρήστη φτάσει σε μια προκαθορισμένη ακτίνα εγγύτητας από την ακμή της μάσκας. Έτσι, επιτρέπει την ψηλάφηση του γεωμετρικού σχήματος πάνω στην οθόνη. Η εφαρμογή διαθέτει ηχητική ανατροφοδότηση αλλά με μηχανισμό χρονικής καθυστέρησης μεταξύ των ηχητικών μηνυμάτων. Με αυτόν τον τρόπο αποτρέπει τη γνωστική υπερφόρτωση, διασφαλίζοντας παράλληλα ότι η ακουστική προσοχή του χρήστη παραμένει ανεμπόδιστη.

2.8 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκε το θεωρητικό υπόβαθρο πάνω στο οποίο βασίζεται η εφαρμογή. Αρχικά, έγινε εισαγωγή στις έννοιες της τεχνητής νοημοσύνης και της μηχανικής μάθησης, με έμφαση στη βαθιά μάθηση και τα Συνελικτικά Νευρωνικά Δίκτυα ως θεμέλιο της υπολογιστικής όρασης. Στη συνέχεια, αναλύθηκε η αρχιτεκτονική YOLO και η εξέλιξή της, με ιδιαίτερη αναφορά στην τμηματοποίηση στιγμιότυπου. Επίσης, εξετάστηκε η απτική ανατροφοδότηση μέσω δόνησης και η καταλληλότητά της για εφαρμογές υποβοήθησης ατόμων με προβλήματα όρασης. Τέλος, η ανασκόπηση σχετικών εργασιών ανέδειξε τους περιορισμούς των υφιστάμενων προσεγγίσεων, όπως η εξάρτηση από το cloud, η αποκλειστική χρήση ηχητικής ανατροφοδότησης και η χρήση απλών πλαισίων οριοθέτησης. Στο επόμενο κεφάλαιο παρουσιάζεται η αρχιτεκτονική και η υλοποίηση του συστήματος.

Κεφάλαιο 3ο: Εργαλεία και Τεχνολογίες Ανάπτυξης

3.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφονται τα εργαλεία και οι τεχνολογίες που επιλέχθηκαν για την υλοποίηση της εφαρμογής. Συγκεκριμένα, παρουσιάζεται το περιβάλλον ανάπτυξης, η γλώσσα προγραμματισμού, και ο μηχανισμός αποθήκευσης ρυθμίσεων. Έπειτα, αναλύεται η βιβλιοθήκη κάμερας, το μοντέλο ανίχνευσης YOLO, και τα εργαλεία για απτική και ηχητική ανάδραση.

3.2 Περιβάλλον Ανάπτυξης

Για την ανάπτυξη εφαρμογών Android χρησιμοποιείται το ολοκληρωμένο περιβάλλον ανάπτυξης (ή IDE) **Android Studio**. Το Android Studio της Google αποτελεί το μοναδικό πλήρως υποστηριζόμενο IDE για αυτόν τον σκοπό και επιλέχθηκε έναντι του Visual Studio Code, διότι είναι το μόνο εργαλείο που ενσωματώνει το Android SDK και επιτρέπει ταυτόχρονα την σύνταξη κώδικα, την δοκιμή σε πραγματική αλλά και εικονική συσκευή και Layout Editor[8].
Συγκεκριμένα ο Layout Editor διευκολύνει τον σχεδιασμό της διεπαφής χρήστη, καθώς αναπαριστά σε πραγματικό χρόνο τον κώδικα XML. Αυτή η λειτουργία αξιοποιήθηκε ιδιαιτέρως κατά τον σχεδιασμό του panel ρυθμίσεων.

Η **Kotlin** αποτελεί την επίσημη γλώσσα για εφαρμογές Android από το 2017. Επιλέχθηκε αντί της Java διότι προσφέρει ασφάλεια έναντι null τιμών σε επίπεδο μεταγλώττισης. Έτσι μειώνει τον κίνδυνο σφαλμάτων NullPointerException.
Επίσης ο κώδικας της είναι πιο συνοπτικός από τι στην Java. Ως συνέπεια, η ανάγνωση και συντήρηση του καθίσταται πιο εύκολη.
Βασικό χαρακτηριστικό της Kotlin που χρησιμοποιήθηκε στην εφαρμογή είναι οι coroutines, οι οποίες επιτρέπουν ασύγχρονες λειτουργίες χωρίς callbacks και χωρίς να μπλοκάρουν το κύριο thread[9][10].
Συγκεκριμένα χρησιμοποιήθηκαν για την ανάγνωση και εγγραφή ρυθμίσεων μέσω του DataStore, καθώς και για την αρχικοποίηση του μοντέλου TensorFlow Lite. Επιπλέον η γλώσσα υποστηρίζει data classes και χρησιμοποιήθηκαν για τις κλάσεις SegmentationResult και Output0 που μεταφέρουν τα αποτελέσματα εντοπισμού.

Για την αποθήκευση των ρυθμίσεων του χρήστη που πρέπει να παραμένουν μετά το κλείσιμο της εφαρμογής, απαιτείται κάποιος μηχανισμός μόνιμης αποθήκευσης. Η παραδοσιακή λύση στο Android είναι τα SharedPreferences, τα οποία όμως εκτελούν λειτουργίες εισόδου/εξόδου στο κύριο thread αυξάνοντας τον κίνδυνο παγώματος της εφαρμογής. Το Jetpack **DataStore** αποτελεί την σύγχρονη αντικατάσταση, το οποίο όπως προαναφέρθηκε εκτελεί με coroutines όλες τις λειτουργίες ασύγχρονα εκτός του κύριου thread[11].
Συγκεκριμένα, χρησιμοποιήθηκε η Preferences DataStore, η οποία αποθηκεύει ζεύγη κλειδιού-τιμής χωρίς να απαιτεί ορισμό σχήματος. Μέσω της κλάσης UserSettingsRepository, οι ρυθμίσεις (κατώφλι εμπιστοσύνης, preset δόνησης, διάρκεια παλμού, μέγιστη ακτίνα) εκτίθενται ως Flow<SettingsSnapshot> επιτρέποντας στη διεπαφή να ανανεώνεται αυτόματα όταν αλλάξει κάποια τιμή.

Για την αρχική φόρτωση κατά την εκκίνηση χρησιμοποιήθηκε ανάγνωση μέσω runBlocking, ώστε οι ρυθμίσεις να είναι διαθέσιμες πριν ξεκινήσει η κάμερα.

3.3 Εργαλεία Μηχανικής Όρασης

Για την πρόσβαση στην κάμερα του Android απαιτείται βιβλιοθήκη που να διαχειρίζεται τον κύκλο ζωής της συσκευής, τη ροή καρέ και τις διαφορές μεταξύ κατασκευαστών. Η επιλογή Camera2 API προσφέρει πλήρη έλεγχο αλλά είναι ιδιαίτερα πολύπλοκη στη χρήση. Αντιθέτως η **CameraX** απλοποιεί σημαντικά τον κώδικα και λειτουργεί με συνέπεια σε διαφορετικές συσκευές και εκδόσεις Android. Αυτός είναι ο λόγος για τον οποίο επιλέχθηκε.
Η **CameraX** χρησιμοποιήθηκε για το Preview και το ImageAnalysis. Στο Preview, εμφανίζεται η ζωντανή εικόνα της κάμερας στην οθόνη, και το ImageAnalysis τροφοδοτεί κάθε καρέ στο μοντέλο ανίχνευσης. Το ImageAnalysis ρυθμίστηκε έτσι ώστε αν η επεξεργασία αργεί, να απορρίπτεται το παλιό καρέ και να διατηρείται πάντα το πιο πρόσφατο. Έτσι η εφαρμογή ανταποκρίνεται ακόμα και σε παλιότερες συσκευές[12].

Για την ανίχνευση αντικειμένων σε πραγματικό χρόνο απαιτείται ένα μοντέλο μηχανικής μάθησης που να είναι ταυτόχρονα γρήγορο και ελαφρύ ώστε να τρέχει σε κινητή συσκευή. Η οικογένεια μοντέλων YOLO (You Only Look Once) επεξεργάζεται ολόκληρο το καρέ με ένα μόνο πέρασμα από το νευρωνικό δίκτυο, σε αντίθεση με παλαιότερες προσεγγίσεις που σάρωναν την εικόνα σε πολλαπλά βήματα[3].

Σε σχέση με προηγούμενες εκδόσεις, το YOLO26 απλοποιεί την εσωτερική αρχιτεκτονική αφαιρώντας στοιχεία που εισήγαγαν πολυπλοκότητα κατά την εξαγωγή για κινητές συσκευές, και παράγει τις ανιχνεύσεις απευθείας χωρίς επιπλέον βήμα μεταγενέστερης επεξεργασίας. Σύμφωνα με το [14], ο χρόνος εκτέλεσης σε CPU μειώθηκε κατά έως 43% σε σύγκριση με το YOLOv11 για το nano μοντέλο.

Το «n» (nano) υποδηλώνει την ελαφρύτερη έκδοση του μοντέλου, συνεπώς και την πιο γρήγορη. Αυτό το καθιστά κατάλληλο για κινητές συσκευές με περιορισμένους πόρους. Το «seg» υποδηλώνει ότι το μοντέλο παράγει μάσκες τμηματοποίησης (segmentation masks)[13]. Το **YOLO26n-seg** επιτυγχάνει mAP 39.6% για bounding boxes και 33.9% για μάσκες στο σύνολο δεδομένων COCO, με χρόνο εκτέλεσης 53.3ms σε CPU και 2.1ms σε GPU T4 [14]. Το μοντέλο είναι ήδη εκπαιδευμένο με 80 κατηγορίες αντικειμένων στο σύνολο δεδομένων COCO[14]. Οι μάσκες αυτές είναι απαραίτητες για τον αλγόριθμο εντοπισμού ακμών στην εφαρμογή, διότι χρησιμοποιούνται τα pixel της μάσκας για να υπολογιστεί η απόσταση του δακτύλου από το περίγραμμα του αντικειμένου.

Μια σημαντική τεχνική διαφορά του YOLO26 σε σχέση με παλαιότερες εκδόσεις αφορά τη μορφή του tensor εξόδου. Στις παραδοσιακές εκδόσεις YOLO ο tensor εξόδου έχει σχήμα [1, 116, 8400], όπου τα 8.400 υποψήφια bounding boxes απαιτούν μεταγενέστερο φιλτράρισμα μέσω NMS. Ενώ παλαιότερες εκδόσεις YOLO παρήγαγαν χιλιάδες υποψήφιες ανιχνεύσεις που έπρεπε να φιλτραριστούν εκ των υστέρων, το YOLO26 εξάγει 300 έτοιμες ανιχνεύσεις. Κάθε μία από αυτές συνοδεύεται από συντεταγμένες θέσης, βαθμολογία εμπιστοσύνης, κατηγορία αντικειμένου και τα δεδομένα μάσκας.

Στην εφαρμογή, για να διατηρηθεί συμβατότητα με μοντέλα παλαιότερης μορφής, η κλάση InstanceSegmentation ανιχνεύει αυτόματα τον τύπο εξόδου από το σχήμα του tensor κατά την αρχικοποίηση. Αυτό καθιστά την εφαρμογή συμβατή με YOLO26, αλλά και με παλαιότερες εκδόσεις (YOLOv11, YOLOv8).

Ένα μοντέλο μηχανικής μάθησης που εκπαιδεύτηκε σε υπολογιστή δεν μπορεί να τρέξει απευθείας σε κινητή συσκευή. Είναι πολύ μεγάλο και απαιτεί υπολογιστική ισχύ που δεν διαθέτει ένα smartphone. Το TensorFlow Lite της Google λύνει αυτό το πρόβλημα: μετατρέπει μοντέλα σε ελαφρύτερη μορφή κατάλληλη για κινητές συσκευές και παρέχει ένα API για την εκτέλεσή τους απευθείας στην συσκευή, χωρίς σύνδεση στο διαδίκτυο. Για να λειτουργεί σε Android, μετατράπηκε σε μορφή TensorFlow Lite με συμπίεση float16. Έτσι το αρχείο γίνεται περίπου μισό σε μέγεθος με ελάχιστη διαφορά στην ακρίβεια.

Αξίζει να σημειωθεί ότι οι μάσκες που παράγει το μοντέλο χρησιμοποιούνται απευθείας για τον εντοπισμό των ακμών των αντικειμένων, χωρίς εξωτερική βιβλιοθήκη επεξεργασίας εικόνας όπως το OpenCV. Η λογική αυτή αναλύεται στην ενότητα 4.5.

3.4 Εργαλεία Απτικής και Ηχητικής Ανάδρασης

Για να μεταδοθεί πληροφορία στον χρήστη μέσω αφής, το Android παρέχει το **Vibrator API** για τον έλεγχο της δόνησης της συσκευής.
Το API επιτρέπει τη δημιουργία προγραμματιστικών μοτίβων δόνησης μέσω της VibrationEffect.createWaveform()[29][30]. Στην εφαρμογή υπάρχουν τρία preset δόνησης: σύντομος παλμός, συνεχής δόνηση και προσαρμοσμένο μοτίβο. Η δόνηση ενεργοποιείται όταν το δάχτυλο βρίσκεται κοντά στην ακμή ενός αντικειμένου και σταματά όταν το δάχτυλο απομακρυνθεί.

Για να επικοινωνεί η εφαρμογή με τον χρήστη, απαιτείται μηχανισμός μετατροπής κειμένου σε ομιλία (Text-to-Speech). Το Android παρέχει εγγενώς το TextToSpeech API[31], το οποίο επιλέχθηκε έναντι εξωτερικών βιβλιοθηκών διότι υποστηρίζει ελληνικά και δεν απαιτεί σύνδεση στο διαδίκτυο. Στην εφαρμογή το TTS χρησιμοποιείται για την ανακοίνωση των αντικειμένων που εντοπίζονται στην κάμερα και για τις επιβεβαιώσεις φωνητικών εντολών. Για να αποφευχθεί η συνεχής επανάληψη της ίδιας ετικέτας, εφαρμόζεται χρονικό cooldown έτσι ώστε η ίδια λέξη να μην ανακοινώνεται ξανά αν δεν έχει περάσει αρκετός χρόνος από την προηγούμενη φορά.

Για να δέχεται η εφαρμογή εντολές μέσω φωνής, απαιτείται μηχανισμός αναγνώρισης ομιλίας. Το Android παρέχει εγγενώς το **SpeechRecognizer API**[32], το οποίο επιλέχθηκε ομοίως διότι υποστηρίζει ελληνικά και λειτουργεί χωρίς εξωτερική βιβλιοθήκη. Σε αντίθεση με το TTS που λειτουργεί τοπικά, το SpeechRecognizer προωθεί ηχητικά δεδομένα στους servers της Google[32], ως αποτέλεσμα να απαιτείται σύνδεση στο διαδίκτυο για να λειτουργήσει. Αυτό αποτελεί το μόνο μέρος της εφαρμογής που απαιτεί σύνδεση. Η ανίχνευση αντικειμένων, η δόνηση, και το TTS λειτουργούν τοπικά.
Στην εφαρμογή ενεργοποιείται με τέσσερα διαδοχικά πατήματα στην οθόνη, έτσι ώστε ο χρήστης να μην το ενεργοποιήσει καταλάθος. Μόλις ο χρήστης μιλήσει, το αποτέλεσμα περνά από την κλάση VoiceCommandRouter που αναγνωρίζει τις ελληνικές λέξεις-κλειδιά (π.χ. «ρυθμίσεις»). Τέλος, υπάρχει και ηχητική ένδειξη για να ενημερώσει τον χρήστη πότε αρχίζει και πότε τελειώνει η ακρόαση.

3.5 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκαν τα εργαλεία και οι τεχνολογίες που επιλέχθηκαν για την υλοποίηση της εφαρμογής. Περιγράφτηκε το περιβάλλον ανάπτυξης Android Studio και η γλώσσα Kotlin, καθώς και ο μηχανισμός μόνιμης αποθήκευσης ρυθμίσεων μέσω Jetpack DataStore. Στη συνέχεια, αναλύθηκαν τα εργαλεία μηχανικής όρασης, δηλαδή η CameraX για τη ροή καρέ, το μοντέλο YOLOv26n-seg για την τμηματοποίηση στιγμιότυπου και το TensorFlow Lite για την εκτέλεσή του στη συσκευή. Τέλος, παρουσιάστηκαν τα εργαλεία απτικής και ηχητικής ανάδρασης, συγκεκριμένα το Vibrator API, το TextToSpeech API και το SpeechRecognizer API. Στο επόμενο κεφάλαιο περιγράφεται η αρχιτεκτονική της εφαρμογής και η λεπτομερής υλοποίηση κάθε επιμέρους συστατικού.

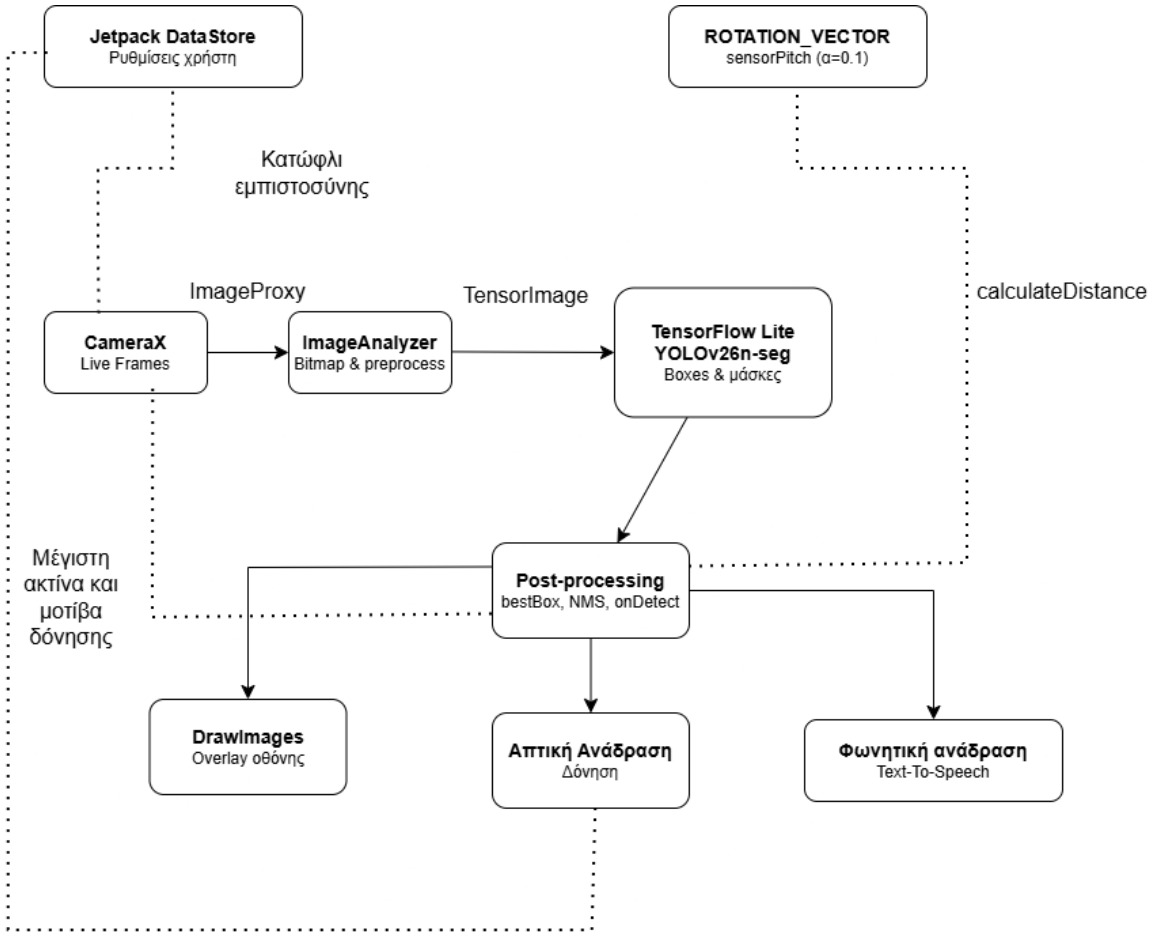
Κεφάλαιο 4ο: Αρχιτεκτονική και Υλοποίηση Εφαρμογής

4.1 Εισαγωγή

Στο επόμενο κεφάλαιο μπαίνουμε στην υλοποίηση και στον κώδικα της εφαρμογής στο Android Studio. Η εφαρμογή θέλει να βοηθήσει άτομα με προβλήματα όρασης και για αυτό συνδυάζει την ανίχνευση αντικειμένων με κάμερα, σε πραγματικό χρόνο με απτική και φωνητική αλληλεπίδραση.

Για την δημιουργία της εφαρμογής χρησιμοποιήθηκε η γλώσσα Kotlin για Android σε κινητά τηλέφωνα. Παράλληλα χρησιμοποιήθηκαν ορισμένες βιβλιοθήκες για την διαχείριση της κάμερας (CameraX), για την τοπική εκτέλεση του YOLO μοντέλου (TensorFlow Lite), και για την αποθήκευση των προτιμήσεων του χρήστη.

Πρώτα, το κείμενο αναλύει την εκκίνηση και αρχικοποίηση της εφαρμογής, και δείχνει την γραφική διεπαφή χρήστη. Στη συνέχεια, εξετάζεται σε βάθος ο τρόπος λειτουργίας στην αλυσίδα επεξεργασίας βίντεο, στον αλγόριθμο εντοπισμού ακμών και απτικής ανατροφοδότησης, στην φωνητική αλληλεπίδραση και τη διαχείριση ρυθμίσεων, και ολοκληρώνεται με αναλυτική περιγραφή κάθε κλάσης.



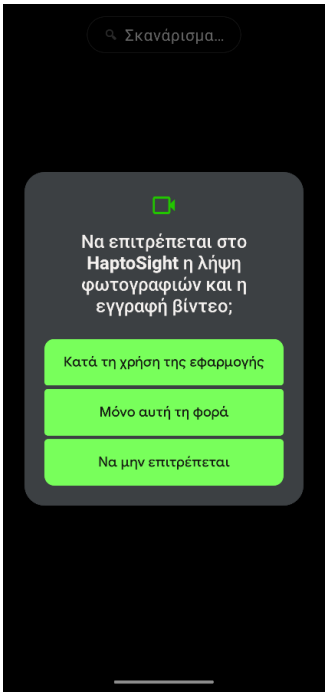
Εικόνα 4.1: Διάγραμμα ροής δεδομένων

Στην Εικόνα 4.1 παρουσιάζεται η ροή δεδομένων της εφαρμογής από την είσοδο της κάμερας έως τις εξόδους αλληλεπίδρασης με τον χρήστη. Η ροή ξεκινά από το CameraX, το οποίο παρέχει frames ως ImageProxy στο ImageAnalyzer. Στο ImageAnalyzer μετατρέπεται το frame σε TensorImage. Το TensorFlow Lite εκτελεί το μοντέλο YOLOv26n-seg και επιστρέφει πλαίσια (bounding boxes) και μάσκες pixel, τα οποία στέλνονται για post-processing. Εκεί φιλτράρονται με βάση το κατώφλι εμπιστοσύνης που αντλεί από το Jetpack Datastore (ρυθμίσεις χρήστη). Επιπλέον, ο αισθητήρας ROTATION_VECTOR τροφοδοτεί τη συνάρτηση calculateDistance() για τον υπολογισμό της απόστασης βάσει της κλίσης της κάμερας. Μετά το post-processing, ανανεώνεται ανάλογα η οπτική απεικόνιση (DrawImages), η δόνηση (με τις ανάλογες ρυθμίσεις του χρήστη), και η φωνητική ανάδραση μέσω Text-To-Speech.

Με άλλα λόγια:

1. **Κάμερα/Camera X** – Η κάμερα τραβά συνεχώς frames σε πραγματικό χρόνο
2. **ImageAnalyzer** – Κάθε εικόνα προετοιμάζεται ώστε να τη διαβάσει το μοντέλο (σωστές διαστάσεις, format)
3. **TensorFlow Lite/YOLOv26n-seg** – Το μοντέλο δέχεται την εικόνα και ανιχνεύει αντικείμενα δίνοντας την θέση και το σχήμα.
4. **Post-processing** – Φιλτράρονται τα αποτελέσματα με βάση το κατώφλι εμπιστοσύνης, και υπολογίζεται η απόσταση κάθε αντικειμένου μέσω calculateDistance() με την κλίση.
5. **Έξοδοι** – Σχεδιάζονται στην οθόνη τα πλαίσια γύρω από τα αντικείμενα. Ενεργοποιείται η δόνηση και η φωνητική λειτουργία.

4.2 Εκκίνηση και Αρχικοποίηση Εφαρμογής



Εικόνα 4.2: Αίτηση άδειας κάμερας

Κατά την πρώτη εκκίνηση, η εφαρμογή ζητά άδεια πρόσβασης στην κάμερα και το μικρόφωνο της συσκευής. Οι δύο άδειες είναι απαραίτητες για τη λήψη βίντεο σε πραγματικό χρόνο, έτσι ώστε να το λάβει το μοντέλο ανίχνευσης, αλλά και για την φωνητική αναγνώριση εντολών.

Εικόνα 4.3: Αίτηση άδειας μικροφώνου

```
settingsRepo = UserSettingsRepository(applicationContext)
val savedSettings = settingsRepo.snapshotBlocking()
vibrationDuration = savedSettings.vibrationDurationMs
pulseGap = savedSettings.pulseGapMs
maxRadius = savedSettings.maxRadiusPx
confidenceThreshold = savedSettings.confidenceThreshold
currentPreset = when (savedSettings.preset) {
    "CONTINUOUS" -> VibrationPreset.CONTINUOUS
    "PULSE" -> VibrationPreset.PULSE
    else -> VibrationPreset.CUSTOM
```

Εικόνα 4.4: Φόρτωση ρυθμίσεων με snapshotBlocking()

Κατά την εκκίνηση της εφαρμογής φορτώνονται οι αποθηκευμένες ρυθμίσεις. Δημιουργείται το UserSettingsRepository με το applicationContext και καλείται η snapshotBlocking(), για συγχρονισμένη ανάγνωση από το DataStore. Αυτά γίνονται πριν την αρχικοποίηση του UI. Οι ρυθμίσεις που αποθηκεύονται για το επόμενο session είναι η διάρκεια της δόνησης(vibrationDuration), το κενό ανάμεσα στους παλμούς δόνησης(pulseGap), η μέγιστη ακτίνα αναζήτησης ακμής για ενεργοποίηση της δόνησης(maxRadius), και το κατώφλι εμπιστοσύνης(confidenceThreshold).

```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance( context = this)

    cameraProviderFuture.addListener( p0 = {
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()
        val aspectRatio = AspectRatio.RATIO_4_3


        val cameraManager = getSystemService( name = Context.CAMERA_SERVICE) as CameraManager
        try {

            val cameraId = cameraManager.cameraIdList.firstOrNull { id ->
                val characteristics = cameraManager.getCameraCharacteristics( cameraId = id)
                val lensFacing = characteristics.get(CameraCharacteristics.LENS_FACING)
                lensFacing == CameraCharacteristics.LENS_FACING_BACK
            }


            cameraId?.let {

                verticalFOV = getVerticalFOV(cameraManager, cameraId = it)
                Log.d( tag = "CameraFOV", msg = "Calculated True Vertical FOV = $verticalFOV degrees")
            } ?: run {
                Log.w( tag = "CameraFOV", msg = "No back camera found, using fallback FOV.")

                verticalFOV = 52.5
            }
        } catch (e: Exception) {
            Log.e( tag = "CameraFOV", msg = "Failed to calculate FOV, using fallback value", tr = e)

            verticalFOV = 52.5
        }
```

Εικόνα 4.5: startCamera()

Στην Εικόνα 4.5 παρουσιάζεται η μέθοδος startCamera(). Το πρώτο τμήμα της αφορά τον υπολογισμό του κατακόρυφου οπτικού πεδίου (verticalFOV). Μέσω του CameraManager, η συνάρτηση εντοπίζει τον πίσω φακό (LENS_FACING_BACK) και καλεί την getVerticalFOV() για να υπολογίσει την τιμή βάσει των φυσικών χαρακτηριστικών του αισθητήρα. Η τιμή αυτή είναι απαραίτητη για τη calculateDistance(), καθώς χρησιμοποιείται για τη μετατροπή pixel σε πραγματικές γωνίες.

Στο onResume() καταχωρείται ο αισθητήρας προσανατολισμού με συχνότητα ρυθμισμένη σε DELAY_NORMAL για τις ενημερώσεις κλίσης. Στο onPause() ο αισθητήρας αποδεσμεύεται για εξοικονόμηση μπαταρίας. Έπειτα αποθηκεύεται στο ViewModel η κατάσταση του panel ρυθμίσεων για αποκατάστασή του μετά από περιστροφή οθόνης.

```
override fun onPause() {
    super.onPause()
    sensorManager.unregisterListener(this)
    viewModel.bottomSheetWasExpanded = (bottomSheetBehavior.state == BottomSheetBehavior.STATE_EXPANDED)
}
```

Εικόνα 4.6: onPause()

```
val bottomSheet = binding.vibrationBottomSheet
bottomSheetBehavior = BottomSheetBehavior.from( view = bottomSheet)
bottomSheetBehavior.isHideable = true
bottomSheetBehavior.state = BottomSheetBehavior.STATE_HIDDEN
bottomSheetBehavior.addBottomSheetCallback(object : BottomSheetBehavior.BottomSheetCallback() {
    override fun onStateChanged(bottomSheet: View, newState: Int) {
        when (newState) {
            BottomSheetBehavior.STATE_HIDDEN -> {
                binding.vibrationPanelToggle.setImageResource(R.drawable.ic_expand)
                if (accessibilityManager.isTouchExplorationEnabled) {
                    binding.root.announceForAccessibility("Ρυθμίσεις κλειστές")
                }
            }
            BottomSheetBehavior.STATE_EXPANDED -> {
                binding.vibrationPanelToggle.setImageResource(R.drawable.ic_collapse)
                stopVibration()
                if (accessibilityManager.isTouchExplorationEnabled) {
                    binding.root.announceForAccessibility("Ρυθμίσεις ανοιχτές")
                }
            }
            BottomSheetBehavior.STATE_COLLAPSED -> {
                binding.vibrationPanelToggle.setImageResource(R.drawable.ic_expand)
            }
        }
    }
    override fun onSlide(bottomSheet: View, slideOffset: Float) { }
})
```

Εικόνα 4.7: Αρχικοποίηση BottomSheetBehavior

```
override fun onResume() {
    super.onResume()
    rotationVectorSensor?.let { sensor ->
        sensorManager.registerListener(this, sensor, samplingPeriodUs = SensorManager.SENSOR_DELAY_NORMAL)
    }
}
```

Εικόνα 4.8: onResume()

Στην Εικόνα 4.7 αρχικοποιείται το panel ρυθμίσεων ως BottomSheet και κατάσταση HIDDEN. Όταν αλλάζει η κατάσταση του panel, ενημερώνεται το εικονίδιο toggle και καλεί

announceForAccessibility(). Όταν ανοίγει το panel καλείται επίσης η μέθοδος stopVibration(), έτσι ώστε η δόνηση να μην συνεχίζεται ενώ ο χρήστης χρησιμοποιεί τις ρυθμίσεις.

```
soundPool = SoundPool.Builder().setMaxStreams(1).build()
listeningSoundId = soundPool.load( context = this,  resId = R.raw.voice_on,  priority = 1)
stopListeningSoundId = soundPool.load( context = this,  resId = R.raw.voice_off,  priority = 1)
```

Εικόνα 4.9: Αρχικοποίηση SoundPool

Στην 4.9 γίνεται η αρχικοποίηση του soundPool και φορτώνονται τα δύο ηχητικά που ενημερώνουν την ενεργοποίηση της αναγνώρισης φωνής.

```
tts = TextToSpeech( context = this) { status ->
    if (status == TextToSpeech.SUCCESS) {
        val result = tts.setLanguage(Locale( language = "el",  country = "GR"))
        if (result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.LANG_NOT_SUPPORTED) {
            Toast.makeText( context = this,  text = "Greek TTS not supported",  duration = Toast.LENGTH_SHORT).show()
        }
    } else {
        Toast.makeText( context = this,  text = "TTS initialization failed",  duration = Toast.LENGTH_SHORT).show()
    }
}
```

Εικόνα 4.10: Αρχικοποίηση TTS στα ελληνικά

Στην 4.10 αρχικοποιείται το TextToSpeech και ορίζεται ως γλώσσα η ελληνική. Σε περίπτωση αποτυχίας εμφανίζεται μήνυμα με το ανάλογο error.

4.3 Γραφική Διεπαφή Χρήστη

```
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.yolosegapp_smooth.MainActivity">
```

Εικόνα 4.11: activity_main - ρίζα CoordinatorLayout

Το αρχείο activity_main.xml χρησιμοποίει CoordinatorLayout ως root container. Αυτό επιτρέπει τα child views να αλληλεπιδρούν μεταξύ τους.

```
        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <androidx.camera.view.PreviewView
                android:id="@+id/previewView"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:importantForAccessibility="no" />

            <ImageView
                android:id="@+id/ivTop"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="centerCrop"
                android:contentDescription="@string/a11y_touch_canvas"
                android:importantForAccessibility="yes"
                android:focusable="true" />

            <!-- Floating Status Bar at Top -->
            <LinearLayout
                android:id="@+id/statusBar"
                android:layout_width="wrap_content"
                android:layout_height="44dp"
                android:layout_gravity="top|center_horizontal"
                android:layout_marginTop="48dp"
                android:background="@drawable/search_bar_bg"
                android:gravity="center"
                android:orientation="horizontal"
                android:paddingHorizontal="20dp"
                android:elevation="6dp">
```

Εικόνα 4.12: activity_main - PreviewView

Μέσα στο CoordinatorLayout ορίζεται ένα FrameLayout που καταλαμβάνει ολόκληρη την οθόνη και

περιέχει τα κύρια στρώματα της διεπαφής. Το live feed της κάμερας βρίσκεται στο PreviewView. Από πάνω του τοποθετείται το ImageView που εμφανίζει το overlay ανίχνευσης.

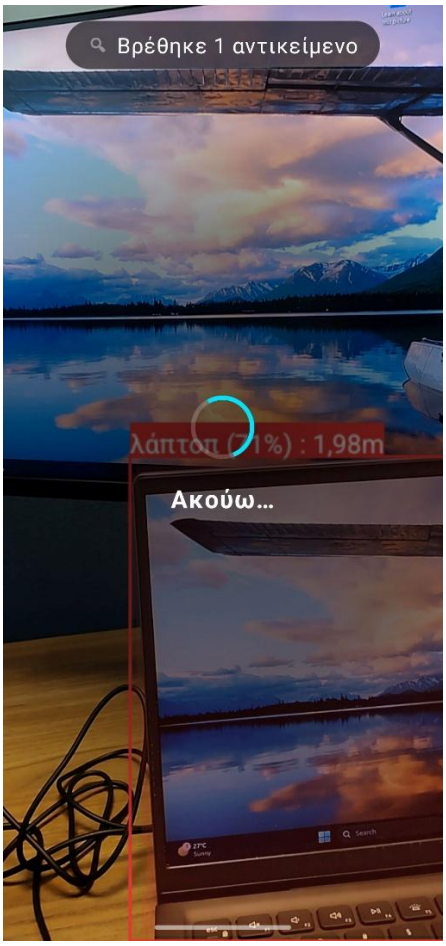
```
        <ImageView
            android:layout_width="18dp"
            android:layout_height="18dp"
            android:src="@android:drawable/ic_menu_search"
            android:layout_marginEnd="8dp"
            android:alpha="0.7"
            app:tint="@color/white"
            android:importantForAccessibility="no"
            android:contentDescription="Εικονίδιο αναζήτησης" />

        <TextView
            android:id="@+id/tvStatusBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Σκανάρισμα…"
            android:textColor="@color/white"
            android:textSize="17sp"
            android:letterSpacing="0.03"
            android:accessibilityLiveRegion="polite" />
    </LinearLayout>

    <!-- Listening overlay indicator -->
    <FrameLayout
        android:id="@+id/listeningIndicator"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/listening_overlay"
        android:visibility="gone"
        android:contentDescription="Ακούω φωνητική εντολή"
        android:accessibilityLiveRegion="assertive">
```

Εικόνα 4.13: activity_main - ορισμός γραμμής κατάστασης

```
74
75              <LinearLayout
76                  android:layout_width="wrap_content"
77                  android:layout_height="wrap_content"
78                  android:layout_gravity="center"
79                  android:orientation="vertical"
80                  android:gravity="center_horizontal">
81
82                  <com.google.android.material.progressindicator.CircularProgressIndicator
83                      android:layout_width="64dp"
84                      android:layout_height="64dp"
85                      android:indeterminate="true"
86                      app:indicatorColor="@color/accent_cyan"
87                      app:trackColor="#33FFFFFF"
88                      app:trackThickness="4dp"
89                      app:indicatorSize="56dp" />
90
91                  <TextView
92                      android:layout_width="wrap_content"
93                      android:layout_height="wrap_content"
94                      android:layout_marginTop="16dp"
95                      android:text="Ακούω…"
96                      android:textColor="@color/white"
97                      android:textSize="20sp"
98                      android:textStyle="bold"
99                      android:letterSpacing="0.05" />
100             </LinearLayout>
101         </FrameLayout>
102 </FrameLayout>
```

Εικόνα 4.14: activity_main – overlay φωνητικής ακρόασης

Στην Εικόνα 4.14 και 4.15 υπάρχει το overlay ακρόασης με ένα κυκλικό δείκτη προόδου και το κείμενο «Ακούω…».



Εικόνα 4.15: overlay φωνητικής ακρόασης

Για την αποτροπή τυχαίων ενεργοποιήσεων της φωνητικής αλληλεπίδρασης και το άνοιγμα των ρυθμίσεων, ο μηχανισμός ενεργοποιείται με τέσσερα διαδοχικά πατήματα. Όταν ενεργοποιηθεί, αναπαράγεται ηχητική ειδοποίηση για να ενημερώσει τον χρήστη ότι το μικρόφωνο καταγράφει.

Η εφαρμογή αναγνωρίζει τις παρακάτω φωνητικές εντολές:

- **«Τι βλέπεις»** : Ενεργοποιεί την φωνητική εκφώνηση όλων των αντικειμένων που ανιχνεύονται στο τρέχον καρέ της κάμερας μέσω TTS.
- **«Ρυθμίσεις», «Ναι», «Όχι», «Κλείσε»** : Φωνητικές εντολές ενεργοποίησης των ρυθμίσεων. Οι εντολές «Ναι» και «Όχι» απαντάνε στην αντίστοιχη ερώτηση επιβεβαίωσης «Θέλετε να ανοίξετε/κλείσετε τις ρυθμίσεις;».
- «Βοήθεια» : Ενεργοποιεί την φωνητική εκφώνηση όλων των διαθέσιμων φωνητικών εντολών.

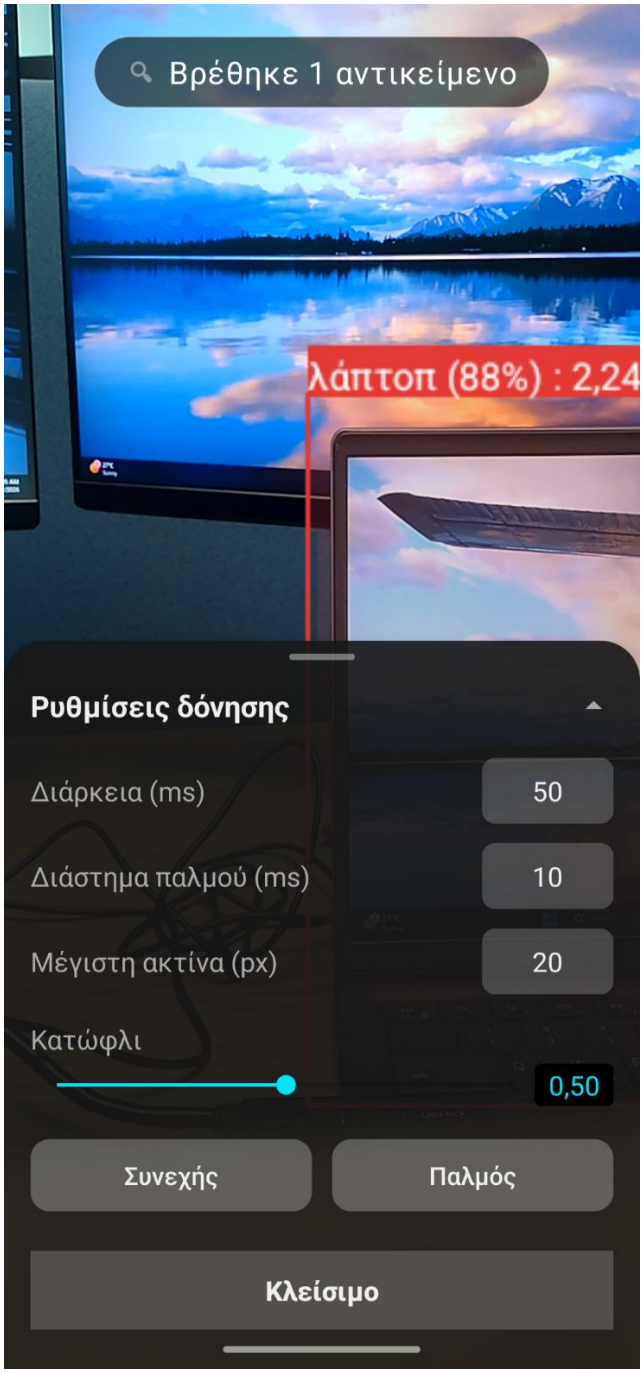
```
104     <!-- Layer 2: Bottom Sheet Settings Panel -->
105     <LinearLayout
106         android:id="@+id/vibration_bottom_sheet"
107         android:layout_width="match_parent"
108         android:layout_height="wrap_content"
109         android:orientation="vertical"
110         android:background="@drawable/panel_bg"
111         android:paddingTop="8dp"
112         android:elevation="16dp"
113         app:layout_behavior="com.google.android.material.bottomsheet.BottomSheetBehavior"
114         app:behavior_hideable="true"
115         app:behavior_peekHeight="56dp">
116
117         <!-- Drag Handle -->
118         <View
119             android:layout_width="40dp"
120             android:layout_height="4dp"
121             android:layout_gravity="center_horizontal"
122             android:layout_marginBottom="8dp"
123             android:background="@drawable/drag_handle" />
124
125         <!-- Panel Header -->
126         <LinearLayout
127             android:id="@+id/vibrationPanelHeader"
128             android:layout_width="match_parent"
129             android:layout_height="wrap_content"
130             android:orientation="horizontal"
131             android:gravity="center_vertical"
132             android:paddingHorizontal="16dp"
133             android:paddingVertical="8dp"
134             android:background="?android:attr/selectableItemBackground"
135             android:clickable="true"
136             android:focusable="true"
137             android:contentDescription="Ρυθμίσεις δόνησης. Πατήστε για άνοιγμα ή κλείσιμο.">
```

Εικόνα 4.16: κεφαλίδα πάνελ ρυθμίσεων (1/2)

```
138
139             <TextView
140                 android:layout_width="0dp"
141                 android:layout_height="wrap_content"
142                 android:layout_weight="1"
143                 android:text="Ρυθμίσεις δόνησης"
144                 android:textStyle="bold"
145                 android:textSize="16sp"
146                 android:textColor="@color/panel_text_primary" />
147
148             <ImageView
149                 android:id="@+id/vibrationPanelToggle"
150                 android:layout_width="24dp"
151                 android:layout_height="24dp"
152                 android:src="@drawable/ic_expand"
153                 app:tint="@color/panel_text_secondary"
154                 android:importantForAccessibility="no"
155                 android:contentDescription="Εναλλαγή πίνακα ρυθμίσεων" />
156         </LinearLayout>
157
158         <!-- Panel Content -->
159         <FrameLayout
160             android:id="@+id/vibrationPanelContent"
161             android:layout_width="match_parent"
162             android:layout_height="wrap_content"
163             android:visibility="visible" />
164     </LinearLayout>
```

Εικόνα 4.17: κεφαλίδα πάνελ ρυθμίσεων (2/2)

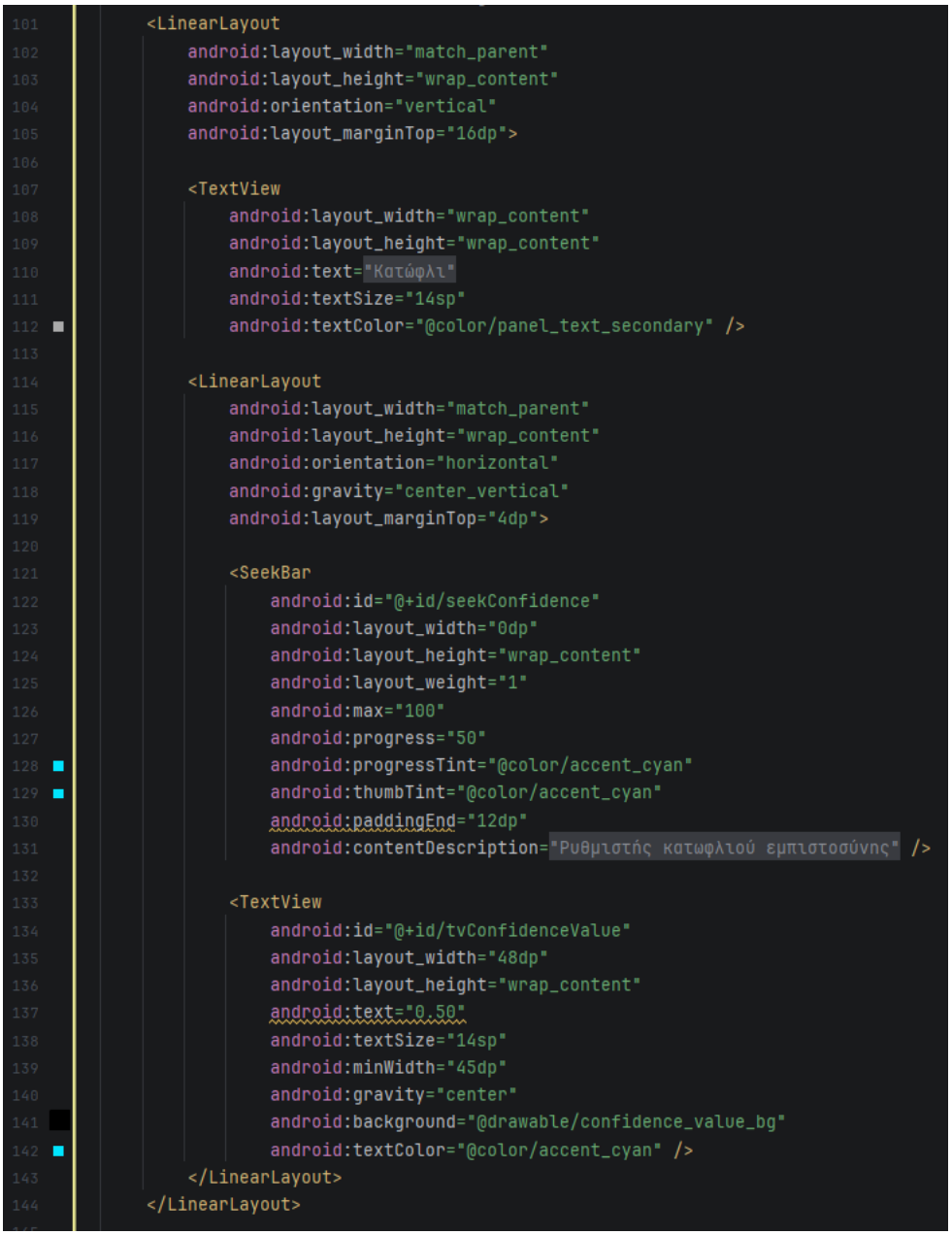
Οι εικόνες 4.16 και 4.17 παρουσιάζουν τη δομή του πάνελ ρυθμίσεων δόνησης. Με ρυθμισμένο το behavior του layout σε BottomSheet, το πάνελ μπορεί να εμφανίζεται μερικώς ορατό κάτω στην οθόνη και να ελέγχεται με σύρσιμο.



Εικόνα 4.18: Ανοιχτό πάνελ ρυθμίσεων

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:paddingHorizontal="16dp"
    android:paddingBottom="24dp"
    android:paddingTop="4dp">

    <!-- Duration Setting -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center_vertical"
        android:layout_marginTop="8dp">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Διάρκεια (ms)"
            android:textSize="14sp"
            android:textColor="@color/panel_text_secondary" />

        <EditText
            android:id="@+id/etDuration"
            android:layout_width="80dp"
            android:layout_height="40dp"
            android:inputType="number"
            android:text="50"
            android:textColor="@color/panel_text_primary"
            android:textSize="14sp"
            android:gravity="center"
            android:background="@drawable/edit_text_bg"
            android:minHeight="40dp"
            android:contentDescription="Πεδίο διάρκειας δόνησης σε χιλιοστά του δευτερολέπτου" />
    </LinearLayout>
```

Εικόνα 4.19: Ρυθμίσεις για Διάρκεια Δόνησης



```
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_marginTop="16dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Κατώφλι"
            android:textSize="14sp"
            android:textColor="@color/panel_text_secondary" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:gravity="center_vertical"
            android:layout_marginTop="4dp">

            <SeekBar
                android:id="@+id/seekConfidence"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:max="100"
                android:progress="50"
                android:progressTint="@color/accent_cyan"
                android:thumbTint="@color/accent_cyan"
                android:paddingEnd="12dp"
                android:contentDescription="Ρυθμιστής κατωφλιού εμπιστοσύνης" />

            <TextView
                android:id="@+id/tvConfidenceValue"
                android:layout_width="48dp"
                android:layout_height="wrap_content"
                android:text="0.50"
                android:textSize="14sp"
                android:minWidth="45dp"
                android:gravity="center"
                android:background="@drawable/confidence_value_bg"
                android:textColor="@color/accent_cyan" />
        </LinearLayout>
    </LinearLayout>
```

Εικόνα 4.20: Ρυθμίσεις για Κατώφλι Εμπιστοσύνης

Οι Εικόνες 4.19 και 4.20 παρουσιάζουν τις παραμέτρους ρύθμισης του πάνελ. Συγκεκριμένα, τα πεδία και την μπάρα ρύθμισης για τη διάρκεια δόνησης και το κατώφλι εμπιστοσύνης αντίστοιχα.

4.4 Αλυσίδα Επεξεργασίας Βίντεο και Ανίχνευση Αντικειμένων

```
inner class ImageAnalyzer : ImageAnalysis.Analyzer {
    override fun analyze(imageProxy: ImageProxy) {
        val bitmapBuffer =
            Bitmap.createBitmap(imageProxy.width, imageProxy.height, config = Bitmap.Config.ARGB_8888)
        imageProxy.use { bitmapBuffer.copyPixelsFromBuffer( src = imageProxy.planes[0].buffer) }

        val matrix = Matrix().apply {
            postRotate( degrees = imageProxy.imageInfo.rotationDegrees.toFloat())
        }

        val rotatedBitmap = Bitmap.createBitmap(
            source = bitmapBuffer, x = 0, y = 0, bitmapBuffer.width, bitmapBuffer.height,
            m = matrix, filter = true
        )



        imageProxy.close()
        instanceSegmentation.invoke( frame = rotatedBitmap)
    }
}
```

Εικόνα 4.21: ImageAnalyzer

```
private fun preProcess(frame: Bitmap): Array<ByteBuffer> {
    val resizedBitmap = Bitmap.createScaledBitmap( src = frame, dstWidth = tensorWidth, dstHeight = tensorHeight, filter = false)
    val tensorImage = TensorImage( dataType = INPUT_IMAGE_TYPE)
    tensorImage.load(resizedBitmap)
    val processedImage = imageProcessor.process(tensorImage)
    return arrayOf(processedImage.buffer)
}
```

Εικόνα 4.22: preprocess()

Η κλάση ImageAnalyzer στην Εικόνα 4.21 υλοποιεί το ImageAnalysis.Analyzer και αποτελεί το σημείο εισόδου κάθε frame στην αλυσίδα επεξεργασίας. Κάθε frame μετατρέπεται σε Bitmap μέσω copyPixelsFromBuffer. Έπειτα περιστρέφεται στον σωστό προσανατολισμό με βάση το rotationDegrees του imageProxy.

```
private val imageProcessor = ImageProcessor.Builder()
    .add(NormalizeOp(INPUT_MEAN, stddev = INPUT_STANDARD_DEVIATION))
    .add(CastOp( destinationType = INPUT_IMAGE_TYPE))
    .build()
```

Εικόνα 4.23: imageProcessor

Στην μέθοδο preProcess() (Εικόνα 4.22) κλιμακώνεται το bitmap στις διαστάσεις που περιμένει το

μοντέλο (tensorWidth x tensorHeight) και φορτώνεται σε TensorImage.

```
init {
    val options = Interpreter.Options()
    options.setNumThreads(4)

    val model = FileUtil.loadMappedFile(context, filePath = modelPath)
    interpreter = Interpreter( byteBuffer = model, options)

    labels.addAll( elements = extractNamesFromMetadata(model))
    if (labels.isEmpty()) {
        if (labelPath == null) {
            message("Model not contains metadata, provide LABELS_PATH in Constants.kt")
            labels.addAll( elements = MetaData.TEMP_CLASSES)
        } else {
            labels.addAll( elements = extractNamesFromLabelFile(context, labelPath))
        }
    }

    val inputShape = interpreter.getInputTensor( inputIndex = 0)?.shape()
    val outputShape0 = interpreter.getOutputTensor( outputIndex = 0)?.shape()
    val outputShape1 = interpreter.getOutputTensor( outputIndex = 1)?.shape()

    if (inputShape != null) {
        tensorWidth = inputShape[1]
        tensorHeight = inputShape[2]


        if (inputShape[1] == 3) {
            tensorWidth = inputShape[2]
            tensorHeight = inputShape[3]
        }
    }

    if (outputShape0 != null) {
        if (outputShape0[1] > outputShape0[2]) {
            isEndToEnd = true
            numDetections = outputShape0[1]
            numFields = outputShape0[2]
        } else {
            isEndToEnd = false
            numChannel = outputShape0[1]
            numElements = outputShape0[2]
        }
    }

    if (outputShape1 != null) {
        if (outputShape1[1] == 32) {
            masksNum = outputShape1[1]
            xPoints = outputShape1[2]
            yPoints = outputShape1[3]
        } else {
            xPoints = outputShape1[1]
            yPoints = outputShape1[2]
            masksNum = outputShape1[3]
        }
    }
}
```

Εικόνα 4.24: init

Ο imageProcessor (Εικόνα 4.23) εφαρμόζει κανονικοποίηση και τα μετατρέπει σε FLOAT32 μέσω CastOp, πριν επιστρέψει το buffer για είσοδο στο μοντέλο.

Στο init block (Εικόνα 4.24) της InstanceSegmentation φορτώνεται το μοντέλο TFLite από το αρχείο assets με 4 νήματα επεξεργασίας. Οι ετικέτες κλάσεων εξάγονται πρώτα από τα metadata του μοντέλου. Στη συνέχεια διαβάζονται τα shapes των τριών tensors(είσοδος, έξοδος 0, έξοδος 1). Για την είσοδο, ο έλεγχος inputShape[1]==3 ανιχνεύει αν το μοντέλο χρησιμοποιεί channel-first διάταξη και αντιστρέφει ανάλογα τις διαστάσεις. Για την έξοδο 1(μάσκες), ελέγχεται αν οι μάσκες βρίσκονται στη δεύτερη ή τελευταία διάσταση. Διαφορετικές εκδόσεις YOLO εξάγουν τα ίδια δεδομένα σε διαφορετική σειρά.

Για την έξοδο 0 (ανιχνεύσεις), συγκρίνονται οι δύο διαστάσεις: αν outputShape0[1] > outputShape0[2], το μοντέλο λειτουργεί σε end-to-end (NMS-free) μορφή και αποθηκεύονται numDetections και numFields. Διαφορετικά, χρησιμοποιείται η παραδοσιακή μορφή. Έτσι ο ίδιος κώδικα υποστηρίζει και τις δύο εκδόσεις εξαγωγής.

```
fun invoke(frame: Bitmap) {
    if (tensorWidth == 0 || tensorHeight == 0
        || xPoints == 0 || yPoints == 0 || masksNum == 0) {
        instanceSegmentationListener.onError("Interpreter not initialized properly")
        return
    }
    if (!isEndToEnd && (numChannel == 0 || numElements == 0)) {
        instanceSegmentationListener.onError("Interpreter not initialized properly")
        return
    }
    if (isEndToEnd && (numDetections == 0 || numFields == 0)) {
        instanceSegmentationListener.onError("Interpreter not initialized properly")
        return
    }

    var preProcessTime = SystemClock.uptimeMillis()

    val imageBuffer = preProcess(frame)

    val coordinatesBuffer = if (isEndToEnd) {
        TensorBuffer.createFixedSize(
            shape = intArrayOf(1, numDetections, numFields),
            dataType = OUTPUT_IMAGE_TYPE
        )
    } else {
        TensorBuffer.createFixedSize(
            shape = intArrayOf(1, numChannel, numElements),
            dataType = OUTPUT_IMAGE_TYPE
        )
    }

    val maskProtoBuffer = TensorBuffer.createFixedSize(
        shape = intArrayOf(1, xPoints, yPoints, masksNum),
        dataType = OUTPUT_IMAGE_TYPE
    )

    val outputBuffer = mapOf<Int, Any>(
        0 to coordinatesBuffer.buffer.rewind(),
        1 to maskProtoBuffer.buffer.rewind()
    )
```

Εικόνα 4.25: invoke() (1/2)

Η invoke() (Εικόνα 4.25 και 4.26) δέχεται ένα Bitmap frame και τρέχει την ανίχνευση. Αρχικά, ελέγχει ότι το μοντέλο έχει φορτωθεί σωστά — αν λείπουν οι διαστάσεις των tensors, επιστρέφει σφάλμα. Ο έλεγχος αυτός είναι διαφορετικός για τα δύο formats: τα παραδοσιακά μοντέλα χρειάζονται numChannel και numElements (διαστάσεις του πίνακα 8400 κελιών), ενώ τα end-to-end χρειάζονται numDetections και numFields(διαστάσεις του πίνακα 300 ανιχνεύσεων). Στη συνέχεια, δημιουργούνται δύο κενοί TensorBuffer που θα γεμίσουν με τα αποτελέσματα του μοντέλου: ο coordinatesBuffer για τις ανιχνεύσεις (bounding boxes, confidence, κλάση) και ο maskProtoBuffer για τις μάσκες. Το shape του coordinatesBuffer εξαρτάται από τη μορφή του μοντέλου — (1, 300, 38) για end-to-end ή (1, 116, 8400) για παραδοσιακή. Οι δύο buffers συνδέονται σε mapOf με δείκτη 0 και 1 και παραδίδονται στον interpreter για να γεμίσουν. Μετά την εκτέλεση, ανάλογα με τη σημαία isEndToEnd, καλείται είτε η bestBoxE2E() είτε η bestBox() για να φιλτράρει τα αποτελέσματα.

```
    preProcessTime = SystemClock.uptimeMillis() - preProcessTime

    var interfaceTime = SystemClock.uptimeMillis()

    interpreter.runForMultipleInputsOutputs( inputs = imageBuffer,  outputs = outputBuffer)

    interfaceTime = SystemClock.uptimeMillis() - interfaceTime

    var postProcessTime = SystemClock.uptimeMillis()

    val bestBoxes = if (isEndToEnd) {
        bestBoxE2E(coordinatesBuffer.floatArray)
    } else {
        bestBox(coordinatesBuffer.floatArray)
    } ?: run {
        instanceSegmentationListener.onEmpty()
        return
    }

    val maskProto = reshapeMaskOutput(maskProtoBuffer.floatArray)

    val segmentationResults = bestBoxes.map {
        SegmentationResult(
            box = it,
            mask = getFinalMask(frame.width, frame.height,  output0 = it,  output1 = maskProto)
        )
    }

    postProcessTime = SystemClock.uptimeMillis() - postProcessTime

    instanceSegmentationListener.onDetect(
        preProcessTime = preProcessTime,
        interfaceTime = interfaceTime,
        postProcessTime = postProcessTime,
        results = segmentationResults,
        bitmap_width = frame.width,
        bitmap_height = frame.height
    )
}
```

Εικόνα 4.26: invoke() (2/2)

Η runForMultipleInputsOutputs() εκτελεί το μοντέλο και γεμίζει τους δύο buffers εξόδου. Ανάλογα με τη μορφή του μοντέλου, καλείται είτε η bestBox() είτε η bestBoxE2E() για να φιλτράρει τα αποτελέσματα βάσει confidence. Η reshapeMaskOutput() μετατρέπει τον επίπεδο πίνακα μασκών σε τρισδιάστατη δομή και στην συνέχεια η getFinalMask() παράγει την τελική μάσκα για κάθε αντικείμενο.

```
206     private fun bestBox(array: FloatArray) : List<Output0>? {
207
208         val output0List = mutableListOf<Output0>()
209
210         for (c in 0 ≤ until < numElements) {
211             var maxConf = confidenceThreshold
212             var maxIdx = -1
213             var currentInd = 4
214             var arrayIdx = c + numElements * currentInd
215
216             while (currentInd < (numChannel - masksNum)){
217                 if (array[arrayIdx] > maxConf) {
218                     maxConf = array[arrayIdx]
219                     maxIdx = currentInd - 4
220                 }
221                 currentInd++
222                 arrayIdx += numElements
223             }
224
225             if (maxConf > confidenceThreshold) {
226                 val clsName = labels[maxIdx]
227                 val cx = array[c]
228                 val cy = array[c + numElements]
229                 val w = array[c + numElements * 2]
230                 val h = array[c + numElements * 3]
231                 val x1 = cx - (w/2F)
232                 val y1 = cy - (h/2F)
233                 val x2 = cx + (w/2F)
234                 val y2 = cy + (h/2F)
235                 if (x1 < 0F || x1 > 1F) continue
236                 if (y1 < 0F || y1 > 1F) continue
237                 if (x2 < 0F || x2 > 1F) continue
238                 if (y2 < 0F || y2 > 1F) continue
239
240                 val maskWeight = mutableListOf<Float>()
241                 while (currentInd < numChannel){
242                     maskWeight.add(array[arrayIdx])
243                     currentInd++
244                     arrayIdx += numElements
245                 }
```

Εικόνα 4.27: bestBox (1/2)

Η bestBox() (Εικόνα 4.27 και 4.28) σαρώνει τον πίνακα εξόδου του μοντέλου σε column-major διάταξη. Ο δείκτης arrayIdx προχωρά κατά στήλη αντί κατά γραμμή. Το YOLOv11 αποθηκεύει τα δεδομένα σε μορφή channels x elements. Για κάθε ένα από τα numElements υποψήφια bounding boxes, σαρώνονται οι κλάσεις και κρατείται αυτή με το υψηλότερο confidence. Αν ξεπερνά το κατώφλι τότε υπολογίζονται οι συντεταγμένες σε κανονικοποιημένη κλίμακα [0,1]. Τα bounding boxes εκτός ορίων απορρίπτονται και εξάγονται τα mask weights που θα χρησιμοποιηθούν αργότερα για τη σύνθεση της μάσκας.

```
245                 }
246
247                 output0List.add(
248                     Output0(
249                         x1 = x1, y1 = y1, x2 = x2, y2 = y2,
250                         cx = cx, cy = cy, w = w, h = h,
251                         cnf = maxConf, cls = maxIdx, clsName = clsName,
252                         maskWeight = maskWeight
253                     )
254                 )
255             }
256         }
257
258         if (output0List.isEmpty()) return null
259
260         return applyNMS(output0List)
261     }
262
```

Εικόνα 4.28: bestBox() (2/2)

```
private fun applyNMS(output0List: List<Output0>) : MutableList<Output0> {
    val sortedBoxes = output0List.sortedByDescending { it.cnf }.toMutableList()
    val selectedBoxes = mutableListOf<Output0>()

    while(sortedBoxes.isNotEmpty()) {
        val first = sortedBoxes.first()
        selectedBoxes.add(first)
        sortedBoxes.remove( element = first)

        val iterator = sortedBoxes.iterator()
        while (iterator.hasNext()) {
            val nextBox = iterator.next()
            val iou = calculateIoU( box1 = first,  box2 = nextBox)
            if (iou >= IOU_THRESHOLD) {
                iterator.remove()
            }
        }
    }

    return selectedBoxes
}
```

Εικόνα 4.29: applyNMS()

Κάθε αντικείμενο που ξεπερνά το κατώφλι αποθηκεύεται ως Output0 με τις συντεταγμένες, το confidence, την κλάση και τα mask weights.
Τα αποτελέσματα περνούν στην applyNMS (Εικόνα 4.29) στην οποία εξαλείφονται τα επικαλυπτόμενα bounding boxes. Ταξινομεί τα boxes κατά φθίνον confidence. Έπειτα κρατά το καλύτερο και αφαιρεί όσα έχουν μεγαλύτερο ή ίσο IoU με αυτό.

```
private fun bestBoxE2E(array: FloatArray): List<Output0>? {
    val output0List = mutableListOf<Output0>()
    val maskStartIdx = 6

    for (d in 0 ≤ until < numDetections) {
        val rowOffset = d * numFields
        val x1Raw = array[rowOffset]
        val y1Raw = array[rowOffset + 1]
        val x2Raw = array[rowOffset + 2]
        val y2Raw = array[rowOffset + 3]
        val conf = array[rowOffset + 4]
        val clsId = array[rowOffset + 5].toInt()

        if (conf < confidenceThreshold) continue
        if (clsId < 0 || clsId >= labels.size) continue

        val x1 = x1Raw.coerceIn(0F, 1F)
        val y1 = y1Raw.coerceIn(0F, 1F)
        val x2 = x2Raw.coerceIn(0F, 1F)
        val y2 = y2Raw.coerceIn(0F, 1F)

        if (x2 <= x1 || y2 <= y1) continue

        val cx = (x1 + x2) / 2F
        val cy = (y1 + y2) / 2F
        val w = x2 - x1
        val h = y2 - y1

        val maskWeight = mutableListOf<Float>()
        for (m in maskStartIdx ≤ until < numFields) {
            maskWeight.add(array[rowOffset + m])
        }

        output0List.add(
            Output0(
                x1 = x1, y1 = y1, x2 = x2, y2 = y2,
                cx = cx, cy = cy, w = w, h = h,
                cnf = conf, cls = clsId, clsName = labels[clsId],
                maskWeight = maskWeight
            )
        )
    }

    return output0List.ifEmpty { null }
}
```

Εικόνα 4.30: bestBoxE2E()

Η bestBoxE2E() (Εικόνα 4.30) εξυπηρετεί τα end-to-end μοντέλα. Αντί να σαρώνει 8400 κελιά, διαβάζει απευθείας τις 300 ανιχνεύσεις που παράγει το μοντέλο εσωτερικά. Οι συντεταγμένες είναι ήδη κανονικοποιημένες στο [0,1]. Αν το confidence περνά το κατώφλι και η κλάση είναι έγκυρη, η ανίχνευση αποθηκεύεται ως Output0. Δεν καλείται applyNMS() γιατί το μοντέλο έχει ήδη εφαρμόσει NMS εσωτερικά.

```
private fun calculateIoU(box1: Output0, box2: Output0): Float {
    val x1 = maxOf( a = box1.x1,  b = box2.x1)
    val y1 = maxOf( a = box1.y1,  b = box2.y1)
    val x2 = minOf( a = box1.x2,  b = box2.x2)
    val y2 = minOf( a = box1.y2,  b = box2.y2)
    val intersectionArea = maxOf( a = 0F,  b = x2 - x1) * maxOf( a = 0F,  b = y2 - y1)
    val box1Area = box1.w * box1.h
    val box2Area = box2.w * box2.h
    return intersectionArea / (box1Area + box2Area - intersectionArea)
}
```

Εικόνα 4.31: calculateIoU()

Η μέθοδος calculateIoU (Εικόνα 4.31) μετρά πόσο επικαλύπτονται δύο bounding boxes. Υπολογίζει την κοινή τους επιφάνεια (τομή) και τη διαιρεί με τη συνολική τους επιφάνεια (ένωση). Αν το αποτέλεσμα είναι κοντά στο 1, τα δύο boxes δείχνουν σχεδόν το ίδιο αντικείμενο και το ένα πρέπει να αφαιρεθεί.

```
private fun reshapeMaskOutput(floatArray: FloatArray): List<Array<FloatArray>> {
    return List( size = masksNum) { mask ->
        Array( size = xPoints) { r ->
            FloatArray( size = yPoints) { c ->
                floatArray[masksNum * yPoints * r + masksNum * c + mask]
            }
        }
    }
}
```

Εικόνα 4.32: reshapeMaskOutput()

Στην 4.32 μετατρέπει τον επίπεδο πίνακα που επιστρέφει το μοντέλο σε λίστα δισδιάστατων μασκών.

```
private fun getFinalMask(
    width: Int,
    height: Int,
    output0: Output0,
    output1: List<Array<FloatArray>>
): Array<FloatArray> {
    val output1Copy = output1.clone()
    val relX1 = output0.x1 * xPoints
    val relY1 = output0.y1 * yPoints
    val relX2 = output0.x2 * xPoints
    val relY2 = output0.y2 * yPoints

    val zero: Array<FloatArray> = Array( size = yPoints) { FloatArray( size = xPoints) { 0F } }
    for ((index, proto) in output1Copy.withIndex()) {
        for (y in 0 ≤ until < yPoints) {
            for (x in 0 ≤ until < xPoints) {
                proto[y][x] *= output0.maskWeight[index]
                if (x + 1  > relX1 && x + 1 < relX2 && y + 1 > relY1 && y + 1 < relY2) {
                    zero[y][x] += proto[y][x]
                }
            }
        }
    }

    return zero.scaleMask( targetWidth = 450 , targetHeight = 600)
}
```

Εικόνα 4.33: getFinalMask()

Η getFinalMask() (Εικόνα 4.33) συνδυάζει αυτές τις μάσκες σε μία τελική μάσκα για κάθε αντικείμενο. Κάθε μάσκα πολλαπλασιάζεται με το αντίστοιχο maskWeight που προέκυψε από το bestBox. Στη συνέχεια αθροίζεται μόνο μέσα στην περιοχή του bounding box. Τα pixel εκτός ορίων αγνοούνται. Το αποτέλεσμα κλιμακώνεται σε 450x600 pixel μέσω scaleMask για την τελική εμφάνιση στην οθόνη.

```
fun Array<FloatArray>.scaleMask(targetWidth: Int, targetHeight: Int): Array<FloatArray> {
    val originalHeight = this.size
    val originalWidth = this[0].size

    val xRatio = (originalWidth shl 16) / targetWidth
    val yRatio = (originalHeight shl 16) / targetHeight

    val output = Array( size = targetHeight) { FloatArray( size = targetWidth) }

    for (y in 0 ≤ until < targetHeight) {
        val origY = (y * yRatio) ushr 16
        for (x in 0 ≤ until < targetWidth) {
            val origX = (x * xRatio) ushr 16
            output[y][x] = this[origY][origX]
        }
    }

    return output
}
```

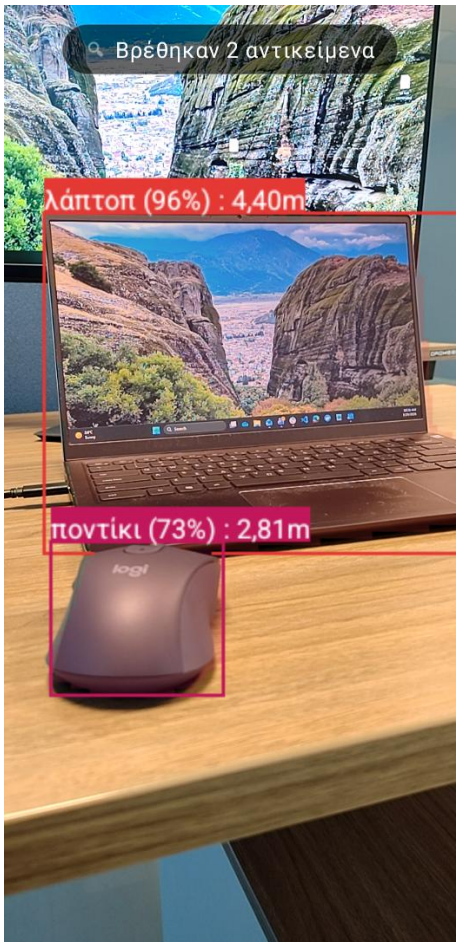
Εικόνα 4.34: scaleMask()

Η scaleMask (Εικόνα 4.34) μεγεθύνει τη μάσκα από τις μικρές διαστάσεις του μοντέλου στις διαστάσεις εμφάνισης (450x600) χρησιμοποιώντας nearest-neighbor παρεμβολή.

```
override fun onDetect(
    interfaceTime: Long,
    results: List<SegmentationResult>,
    preProcessTime: Long,
    postProcessTime: Long,
    bitmap_width: Int,
    bitmap_height: Int
) {

    val resultsWithDistance = results.map { result ->
        val distance = calculateDistance(result.box, bitmapHeight = bitmap_height)

        val newBox = result.box.copy(
            cnf = distance?.toFloat() ?: -1.0f,
            originalCnf = result.box.cnf
        )
        result.copy(box = newBox)
    }

    // Update state
    lastResults = resultsWithDistance

    // Draw
    val image = drawImages.invoke(resultsWithDistance)
    runOnUiThread {
        binding.ivTop.setImageBitmap(image)
        updateStatusBar( count = resultsWithDistance.size)
    }
}
```

Εικόνα 4.35: onDetect()

Η onDetect (Εικόνα 4.35) καλείται όταν ολοκληρωθεί η ανίχνευση. Για κάθε αποτέλεσμα υπολογίζεται η απόσταση μέσω calculateDistance και αποθηκεύεται στο πεδίο cnf του Output0. Η αρχική τιμή confidence διατηρείται στο originalCnf. Τέλος τα αποτελέσματα αποθηκεύονται στο lastResults και η drawImages.invoke δημιουργεί το overlay.



Εικόνα 4.36: Κύρια οθόνη εφαρμογής

Στην κύρια οθόνη εμφανίζεται η ζωντανή ροή της κάμερας με τα αποτελέσματα ανίχνευσης. Στο πάνω μέρος της οθόνης εμφανίζεται ο αριθμός των ανιχνευμένων αντικειμένων. Κάθε αντικείμενο περιβάλλεται από πλαίσιο οριοθέτησης με την ελληνική ετικέτα, το ποσοστό εμπιστοσύνης, τη μάσκα τμηματοποίησης, και την απόσταση. Αξίζει να σημειωθεί, πως ο υπολογισμός της απόστασης θεωρείται έγκυρος μόνο για αντικείμενα που βρίσκονται στο δάπεδο.

```kotlin
private fun calculateDistance(box: Output0, bitmapHeight: Int): Double? {
    if (verticalFOV == 0.0 || bitmapHeight == 0) return null

    val effectiveFOV = verticalFOV * FOV_CORRECTION_FACTOR

    val yBottomPixels = box.y2 * bitmapHeight

    val screenCenterY = bitmapHeight / 2.0
    val pixelsFromCenter = yBottomPixels - screenCenterY

    val focalLengthPixels = (bitmapHeight / 2.0) / kotlin.math.tan( x = Math.toRadians(effectiveFOV / 2.0))

    val angleFromOpticalAxisRadians = kotlin.math.atan( x = pixelsFromCenter / focalLengthPixels)
    val angleFromOpticalAxisDegrees = Math.toDegrees(angleFromOpticalAxisRadians)

    val calibratedPitch = sensorPitch + PITCH_OFFSET_DEG
    val totalAngleDegrees = calibratedPitch + angleFromOpticalAxisDegrees

    if (totalAngleDegrees <= 1.0 || totalAngleDegrees >= 89.0) return null
    val totalAngleRadians = Math.toRadians(totalAngleDegrees)

    return cameraHeightMeters / kotlin.math.tan( x = totalAngleRadians)
}
```

Εικόνα 4.37: calculateDistance

Η calculateDistance() (Εικόνα 4.37) υπολογίζει την οριζόντια απόσταση ενός αντικειμένου από τη συσκευή, μέσω της θέσης του αντικειμένου στην οθόνη και την κλίση της συσκευής. Το κάτω μέρος του bounding box (box.y2) χρησιμοποιείται για την προσέγγιση με το σημείο επαφής του αντικειμένου με το έδαφος.

Η διαδικασία υπολογισμού εκτυλίσσεται σε τρία βήματα. Πρώτον, η κατακόρυφη γωνία οπτικού πεδίου (vertical FOV), την οποία λαμβάνει μέσω CameraCharacteristics, πολλαπλασιάζεται με τον εμπειρικό συντελεστή FOV_CORRECTION_FACTOR = 0.92 για αντιστάθμιση αποκλίσεων μεταξύ θεωρητικής και πραγματικής οπτικής. Από αυτήν υπολογίζεται η εστιακή απόσταση σε pixel (focal length) μέσω της σχέσης f = (bitmapHeight / 2) / tan(effectiveFOV / 2). Δεύτερον, η κατακόρυφη απόσταση του κάτω άκρου από το κέντρο της εικόνας (frame) μετατρέπεται σε γωνία ως προς τον οπτικό άξονα: θ = atan(pixelsFromCenter / f). Τρίτον, στη γωνία αυτή προστίθεται η τρέχουσα κλίση της συσκευής (sensorPitch), η οποία προέρχεται από τον αισθητήρα TYPE_ROTATION_VECTOR και εξομαλύνεται μέσω φίλτρου χαμηλών συχνοτήτων (α = 0.1), καθώς και ένα σταθερό offset (PITCH_OFFSET_DEG = −4°) για αντιστάθμιση σφάλματος. Το άθροισμα αυτών αποτελεί η totalAngle, δηλαδή η γωνία μεταξύ της οπτικής ακτίνας προς το αντικείμενο και του οριζόντιου επιπέδου.

Η τελική απόσταση υπολογίζεται από τη σχέση:

d = cameraHeightMeters / tan(totalAngle)

όπου cameraHeightMeters = 1.2 m αντιστοιχεί στο ύψος κράτησης της συσκευής που θεωρούμε (ύψος στήθους). Η συνάρτηση επιστρέφει null, όταν η totalAngle βρίσκεται εκτός του διαστήματος (1°, 89°), διότι σε ακραίες γωνίες η τριγωνομετρική σχέση αποκλίνει σημαντικά από την πραγματικότητα.

4.5 Αλγόριθμος Εντοπισμού Ακμών και Απτική Ανατροφοδότηση

```
private fun installMainTouchListener() {
    binding.ivTop.setOnTouchListener { _, event ->
        when (event.action) {
            MotionEvent.ACTION_DOWN -> {
                if (bottomSheetBehavior.state == BottomSheetBehavior.STATE_EXPANDED) return@setOnTouchListener true
                val labelFromVibration = handleVibration( x = event.x,  y = event.y)
                if (labelFromVibration != null) {
                    speakGreekWithCooldown(labelFromVibration)
                } else {
                    val label = getLabelUnderFinger( x = event.x,  y = event.y)
                    if (label != null) speakGreekWithCooldown(label)
                }
                if (!isPressing) {
                    isPressing = true
                }
            }
            MotionEvent.ACTION_MOVE -> {
                if (bottomSheetBehavior.state == BottomSheetBehavior.STATE_EXPANDED) return@setOnTouchListener true
                val labelFromVibration = handleVibration( x = event.x,  y = event.y)
                if (labelFromVibration != null) {
                    speakGreekWithCooldown(labelFromVibration)
                } else {
                    val label = getLabelUnderFinger( x = event.x,  y = event.y)
                    if (label != null) speakGreekWithCooldown(label)
                }
            }
            MotionEvent.ACTION_UP, MotionEvent.ACTION_CANCEL -> {
                stopVibration()
                isPressing = false
                lastSpokenLabel = null
                lastSpokenTime = 0
                detectConsecutiveTaps()
            }
        }
        true
    }
}
```

Εικόνα 4.38: installMainTouchListener()

Η installMainTouchListener (Εικόνα 4.38) διαχειρίζεται τρεις τύπους αφής. Στο ACTION_DOWN πάτημα και ACTION_MOVE σύρσιμο και σε περίπτωση που το panel ρυθμίσεων είναι ανοιχτό, η αφή αγνοείται.

Διαφορετικά καλείται η handleVibration που επιστρέφει την ετικέτα του αντικειμένου κοντά στην ακμή. Αν βρεθεί εκφωνείται μέσω speakGreekWithCooldown. Αν το δάχτυλο βρίσκεται κοντά σε ακμή, η getLabelUnderFinger ελέγχει αν βρίσκεται πάνω σε αντικείμενο και εκφωνεί την ετικέτα χωρίς δόνηση. Στο ACTION_UP σήκωμα δακτύλου σταματά η δόνηση, μηδενίζεται ο χρόνος εκφώνησης, και καλείται η detectConsecutiveTaps, έτσι ώστε να μπορέσει να ενεργοποιηθεί η αναγνώριση φωνής στα 4 διαδοχικά πατήματα.

```
private fun handleVibration(x: Float, y: Float): String? {
    if (lastResults.isEmpty()) {
        stopVibration()
        return null
    }

    val imageView = binding.ivTop
    val drawable = imageView.drawable ?: run {
        stopVibration()
        return null
    }
    val bmp = (drawable as? android.graphics.drawable.BitmapDrawable)?.bitmap ?: run {
        stopVibration()
        return null
    }

    val scaleX = bmp.width.toFloat() / imageView.width
    val scaleY = bmp.height.toFloat() / imageView.height

    val imgX = (x * scaleX).toInt()
    val imgY = (y * scaleY).toInt()

    var nearPerimeter = false
    var detectedDistance: Int? = null
    var detectedLabel: String? = null

    lastResults.forEach { result ->
        // Check if the point is within the mask bounds
        if (imgY in result.mask.indices && imgX in result.mask[imgY].indices) {
            val distancePixel = distanceToPerimeter(result.mask, x = imgX, y = imgY, maxRadius)
            if (distancePixel != null) {
                nearPerimeter = true
                detectedDistance = distancePixel
                detectedLabel = result.box.clsName
            }
        }
    }

    if (nearPerimeter && detectedDistance != null) {
        triggerVibration()
        return detectedLabel
    } else {
        stopVibration()
        return null
    }
}
```

Εικόνα 4.39: handleVibration()

Η handleVibration (Εικόνα 4.39) δέχεται τις συντεταγμένες αφής του χρήστη και ελέγχει αν βρίσκονται κοντά στην ακμή κάποιου αντικειμένου. Αρχικά, αν δεν υπάρχουν αποτελέσματα ανίχνευσης ή δεν είναι διαθέσιμη η εικόνα, σταματά η δόνηση και η συνάρτηση επιστρέφει null. Στη συνέχεια, οι συντεταγμένες της οθόνης μετατρέπονται σε συντεταγμένες εικόνας μέσω αναλογικής κλιμάκωσης (scaleX, scaleY). Για κάθε αποτέλεσμα ανίχνευσης, αν το σημείο αφής βρίσκεται εντός της μάσκας του αντικειμένου, καλείται η distanceToPerimeter που υπολογίζει την απόσταση σε pixel από την πλησιέστερη ακμή.
Αν βρεθεί ακμή εντός του maxRadius ενεργοποιείται η δόνηση μέσω της μεθόδου triggerVibration. Αν δεν βρεθεί κοντινή ακμή η δόνηση σταματά.

```
private fun distanceToPerimeter(mask: Array<FloatArray>, x: Int, y: Int, maxRadius: Int): Int? {
    val h = mask.size
    val w = mask[0].size
    if (x < 0 || x >= w || y < 0 || y >= h) return null

    val centerInside = mask[y][x] > 0.5f

    for (r in 1 ≤ .. ≤ maxRadius) { // increasing radius
        for (dy in -r ≤ .. ≤ r) {
            val yy = y + dy
            if (yy !in 0 ≤ until < h) continue
            for (dx in -r ≤ .. ≤ r) {
                // Skip the center point (0,0) because we already know it
                if (dx == 0 && dy == 0) continue
                val xx = x + dx
                if (xx !in 0 ≤ until < w) continue
                val neighborInside = mask[yy][xx] > 0.5f
                if (neighborInside != centerInside) {
                    return r // distance to perimeter
                }
            }
        }
    }
    return null
}
```

Εικόνα 4.40: distanceToPerimeter()

Στην Εικόνα 4.40 υπολογίζεται η απόσταση του σημείου αφής από την πλησιέστερη ακμή ενός αντικειμένου. Αρχικά ελέγχει αν το σημείο βρίσκεται εντός ή εκτός της μάσκας. Έπειτα σαρώνει ομόκεντρους δακτυλίους με αυξανόμενη ακτίνα(r=1 έως maxRadius). Σε κάθε δακτύλιο, αν βρεθεί γειτονικό pixel με διαφορετική κατάσταση (εντός/εκτός) από το κεντρικό, σημαίνει ότι εντοπίστηκε ακμή. Έτσι η συνάρτηση επιστρέφει την ακτίνα r ως απόσταση σε pixel. Αν δεν βρεθεί ακμή εντός του maxRadius, επιστρέφει null.

```
private fun triggerVibration() {
    if (vibrating) return
    val nowNs = System.nanoTime()
    if (nowNs - lastHapticNanos < hapticMinIntervalNs) return
    lastHapticNanos = nowNs
    vibrating = true

    vibrator?.let { v ->
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
            val effect = when (currentPreset) {
                VibrationPreset.PULSE -> VibrationEffect.createWaveform(
                    timings = longArrayOf(0, vibrationDuration, pulseGap),
                    amplitudes = intArrayOf(0, 1, 0), repeat = 0
                )
                VibrationPreset.CONTINUOUS -> VibrationEffect.createWaveform(
                    timings = longArrayOf(0, vibrationDuration, 10),
                    amplitudes = intArrayOf(0, 1, 0), repeat = 0
                )
                VibrationPreset.CUSTOM -> VibrationEffect.createWaveform(
                    timings = longArrayOf(0, vibrationDuration, pulseGap),
                    amplitudes = intArrayOf(0, 1, 0), repeat = 0
                )
            }
            v.vibrate( vibe = effect)
        } else {
            @Suppress( ...names = "DEPRECATION")
            val pattern = when (currentPreset) {
                VibrationPreset.PULSE -> longArrayOf(0, vibrationDuration, pulseGap, vibrationDuration, pulseGap)
                VibrationPreset.CONTINUOUS -> longArrayOf(0, vibrationDuration, 10, vibrationDuration, 10)
                VibrationPreset.CUSTOM -> longArrayOf(0, vibrationDuration, pulseGap, vibrationDuration, pulseGap)
            }
            @Suppress( ...names = "DEPRECATION")
            v.vibrate(pattern, repeat = 0)
        }
    }
}
```

Εικόνα 4.41: triggerVibration()

Στην μέθοδο της Εικόνας 4.41 ενεργοποιείται η δόνηση. Πρώτα ελέγχεται αν η δόνηση είναι ήδη ενεργή και αν έχει περάσει αρκετός χρόνος από την τελευταία ώστε να αποφεύγονται υπερβολικά συχνές δονήσεις. Ανάλογα με το επιλεγμένο preset (PULSE, CONTINUOUS, CUSTOM), δημιουργεί διαφορετικό μοτίβο δόνησης.

```
private fun speakGreekWithCooldown(label: String) {
    val currentTime = System.currentTimeMillis()

    // Check if enough time has passed since the last announcement
    if (lastSpokenLabel != label || currentTime - lastSpokenTime > ttsCooldown) {
        speakGreek(label)
        lastSpokenLabel = label
        lastSpokenTime = currentTime
    }
}
```

Εικόνα 4.42: speakGreekWithCooldown()

Η speakGreekWithCooldown (Εικόνα 4.42) εκφωνεί την ετικέτα του αντικειμένου στα ελληνικά. Αυτό γίνεται μόνο αν η ετικέτα είναι διαφορετική από την τελευταία που εκφωνήθηκε ή αν έχει περάσει αρκετός χρόνος από το cooldown. Με αυτόν τον τρόπο αποφεύγεται η επαναλαμβανόμενη εκφώνηση του ίδιου αντικειμένου όσο ο χρήστης κρατά το δάχτυλο σταθερό.

```
private fun getLabelUnderFinger(x: Float, y: Float): String? {
    if (lastResults.isEmpty()) return null

    val imageView = binding.ivTop
    val drawable = imageView.drawable ?: return null
    val bmp = (drawable as? android.graphics.drawable.BitmapDrawable)?.bitmap ?: return null

    val scaleX = bmp.width.toFloat() / imageView.width
    val scaleY = bmp.height.toFloat() / imageView.height

    val imgX = (x * scaleX).toInt()
    val imgY = (y * scaleY).toInt()

    lastResults.forEach { result ->
        if (imgY in result.mask.indices && imgX in result.mask[imgY].indices) {
            if (result.mask[imgY][imgX] > 0.5f) {
                return result.box.clsName
            }
        }
    }
    return null
}
```

Εικόνα 4.43: getLabelUnderFinger()

Η getLabelUnderFinger (Εικόνα 4.43) λειτουργεί συμπληρωματικά στη handleVibration. Όταν το δάχτυλο του χρήστη δεν βρίσκεται κοντά σε ακμή, η συνάρτηση αυτή ελέγχει αν βρίσκεται απευθείας πάνω σε αναγνωρισμένο αντικείμενο. Μετατρέπει τις συντεταγμένες οθόνης σε συντεταγμένες εικόνας και ελέγχει αν η τιμή της μάσκας σε αντίστοιχο pixel είναι μεγαλύτερη από 0.5. Αν είναι, τότε επιστρέφει την ετικέτα του αντικειμένου.
Η διαφορά είναι ότι η getLabelUnderFinger δεν ενεργοποιεί την δόνηση, απλώς αναγνωρίζει το αντικείμενο κάτω από το δάχτυλο.

4.6 Φωνητική Αλληλεπίδραση

```kotlin
package com.example.yolosegapp_smooth.voice

import java.text.Normalizer

16 Usages
enum class VoiceCommand {
    2 Usages
    WHAT_DO_YOU_SEE,
    2 Usages
    OPEN_SETTINGS,
    2 Usages
    CLOSE_SETTINGS,
    2 Usages
    HELP,
    2 Usages
    YES,
    2 Usages
    NO,
    2 Usages
    UNKNOWN
}

2 Usages
object VoiceCommandRouter {

    1 Usage
    fun normalize(text: String): String {
        val nfd = Normalizer.normalize( src = text.lowercase(), form = Normalizer.Form.NFD)
        return nfd.replace(Regex( pattern = "\\p{InCombiningDiacriticalMarks}+"), replacement = "")
    }

    1 Usage
    fun route(spoken: String): VoiceCommand {
        val n = normalize( text = spoken)
        return when {
            n.contains( other = "βλεπεις") || n.contains( other = "βλεπει") -> VoiceCommand.WHAT_DO_YOU_SEE
            n.contains( other = "ανοιξε ρυθμισεις") || n.contains( other = "ρυθμισεις") -> VoiceCommand.OPEN_SETTINGS
            n.contains( other = "κλεισε") -> VoiceCommand.CLOSE_SETTINGS
            n.contains( other = "βοηθεια") || n.contains( other = "βοηθ") -> VoiceCommand.HELP
            n.contains( other = "ναι") || n.contains( other = "εντ") -> VoiceCommand.YES
            n.contains( other = "οχι") -> VoiceCommand.NO
            else -> VoiceCommand.UNKNOWN
        }
    }
}
```

Εικόνα 4.44: VoiceCommandRouter

Η κλάση VoiceCommandRouter της Εικόνας 4.44 διαχειρίζεται την αναγνώριση φωνητικών εντολών. Ορίζεται ένα enum VoiceCommand με τις διαθέσιμες εντολές.

Η συνάρτηση normalize προετοιμάζει το κείμενο που επιστρέφει η αναγνώριση ομιλίας. Το μετατρέπει σε πεζά, εφαρμόζει κανονικοποίηση NFD και αφαιρεί τους τόνους. Αυτό διασφαλίζει ότι η εντολή αναγνωρίζεται ανεξάρτητα από τη χρήση τόνων.

Η route αντιστοιχίζει το κανονικοποιημένο κείμενο στην κατάλληλη εντολή.

```
private fun detectConsecutiveTaps() {
    val currentTime = System.currentTimeMillis()

    if (currentTime - lastTapTime > tapTimeout) {
        tapCount = 0
    }

    tapCount++
    lastTapTime = currentTime

    if (tapCount == 4) {
        tapCount = 0

        // Check if the panel is currently visible or hidden
        if (bottomSheetBehavior.state != BottomSheetBehavior.STATE_HIDDEN) {
            // Panel is open, ask to close it
            startVoiceCloseConfirmationFlow()
        } else {
            // Panel is hidden, ask to open it
            startVoiceConfirmationFlow()
        }
    }
}
```

Εικόνα 4.45: detectConsecutiveTaps()

Η detectConsecutiveTaps (Εικόνα 4.45) μετρά τα διαδοχικά πατήματα του χρήστη. Αν ο χρόνος μεταξύ δύο πατημάτων ξεπεράσει το tapTimeout, ο μετρητής μηδενίζεται. Όταν φτάσει στα 4 πατήματα, ελέγχεται η κατάσταση του panel ρυθμίσεων και ξεκινά η ανάλογη μέθοδος φωνητικών εντολών.

```
private fun startVoiceCloseConfirmationFlow() {
    if (ActivityCompat.checkSelfPermission( context = this, permission = Manifest.permission.RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED) {
        Toast.makeText( context = this, text = "Microphone permission is required for voice commands", duration = Toast.LENGTH_LONG).show()
        return
    }
    stopVibration()

    val prompt = "Θέλετε να κλείσετε τις ρυθμίσεις; Πείτε ναι ή όχι."
    speakGreekText(prompt, utteranceId = "prompt_close_settings")

    tts.setOnUtteranceProgressListener(object : UtteranceProgressListener() {
        override fun onStart(utteranceId: String?) {}
        override fun onDone(utteranceId: String?) {
            if (utteranceId == "prompt_close_settings") {
                // --- ADD A SMALL DELAY ---
                Handler(Looper.getMainLooper()).postDelayed( r = {
                    runOnUiThread {
                        startListening()
                    }
                }, delayMillis = 500) // 500 millisecond (half-second) delay
            }
        }
        override fun onError(utteranceId: String?) {}
    })
}
```

Εικόνα 4.46: startVoiceCloseConfirmationFlow()

Η startVoiceCloseConfirmationFlow (Εικόνα 4.46) ελέγχει πρώτα αν υπάρχει άδεια μικροφώνου. Στη συνέχεια σταματά τη δόνηση και εκφωνεί το μήνυμα «Θέλετε να κλείσετε τις ρυθμίσεις; Πείτε ναι ή όχι.» Μόλις ολοκληρωθεί η εκφώνηση ενεργοποιείται η αναγνώριση ομιλίας.

```
403     private fun startListening() {
404         if (isListening) return
405
406         // Destroy any previous instance to be safe
407         speechRecognizer?.destroy()
408
409         // Create a new instance
410         speechRecognizer = SpeechRecognizer.createSpeechRecognizer( context = this)
411         val recognizer = speechRecognizer ?: return
412
413         val intent = Intent( action = RecognizerIntent.ACTION_RECOGNIZE_SPEECH).apply {
414             putExtra( name = RecognizerIntent.EXTRA_LANGUAGE_MODEL, value = RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
415             putExtra( name = RecognizerIntent.EXTRA_LANGUAGE, value = "el-GR")
416             putExtra( name = RecognizerIntent.EXTRA_PARTIAL_RESULTS, value = false)
417         }
418
419         recognizer.setRecognitionListener(object : RecognitionListener {
420             override fun onReadyForSpeech(params: Bundle?) {
421                 isListening = true
422                 runOnUiThread { findViewById<View>(R.id.listeningIndicator).visibility = View.VISIBLE }
423                 soundPool.play(listeningSoundId, leftVolume = 1.0f, rightVolume = 1.0f, priority = 0, loop = 0, rate = 1.0f)
424             }
425
426             override fun onBeginningOfSpeech() {}
427             override fun onRmsChanged(rmsdB: Float) {}
428             override fun onBufferReceived(buffer: ByteArray?) {}
429             override fun onEndOfSpeech() { isListening = false }
430             override fun onPartialResults(partialResults: Bundle?) {}
431
                5 Usages
432             private fun finishListening() {
433                 isListening = false
434                 recognizer.destroy()
435                 speechRecognizer = null
436                 runOnUiThread { findViewById<View>(R.id.listeningIndicator).visibility = View.GONE }
437                 soundPool.play(stopListeningSoundId, leftVolume = 1.0f, rightVolume = 1.0f, priority = 0, loop = 0, rate = 1.0f)
438             }
439
```

Εικόνα 4.47: startListening()

Στην μέθοδο της Εικόνας 4.47 ξεκινά η αναγνώριση ομιλίας. Αν είναι ήδη ενεργή, αγνοείται. Δημιουργεί νέο SpeechRecognizer και ρυθμίζει ως γλώσσα την ελληνική. Στο onReadyForSpeech εμφανίζεται ο δείκτης ακρόασης και αναπαράγεται ένας ήχος, για να καταλάβει ο χρήστης ότι η εφαρμογή περιμένει φωνητική εντολή.

Στο finishListening τερματίζει την αναγνώριση. Κρύβει τον δείκτη ακρόασης και αναπαράγει έναν δεύτερο ήχο για να σηματοδοτήσει το τέλος της ακρόασης.

```
private fun handleVoiceCommand(command: String) {
    when (VoiceCommandRouter.route( spoken = command)) {
        VoiceCommand.WHAT_DO_YOU_SEE -> announceDetectedObjects()
        VoiceCommand.OPEN_SETTINGS -> {
            if (bottomSheetBehavior.state == BottomSheetBehavior.STATE_HIDDEN) {
                speakGreekWithDelay( text = "Άνοιγμα ρυθμίσεων.")
                toggleBottomSheet()
            }
        }
        VoiceCommand.CLOSE_SETTINGS -> {
            if (bottomSheetBehavior.state != BottomSheetBehavior.STATE_HIDDEN) {
                speakGreekWithDelay( text = "Κλείσιμο ρυθμίσεων.")
                toggleBottomSheet()
            }
        }
        VoiceCommand.HELP -> speakGreekText(
            "Διαθέσιμες εντολές: τι βλέπεις, ρυθμίσεις, κλείσε, βοήθεια."
        )
        VoiceCommand.YES -> {
            if (bottomSheetBehavior.state != BottomSheetBehavior.STATE_HIDDEN) {
                speakGreekWithDelay( text = "Κλείσιμο ρυθμίσεων.")
                toggleBottomSheet()
            } else {
                speakGreekWithDelay( text = "Άνοιγμα ρυθμίσεων.")
                toggleBottomSheet()
            }
        }
        VoiceCommand.NO -> { /* User declined, do nothing */ }
        VoiceCommand.UNKNOWN -> speakGreekText("Δεν κατάλαβα. Πείτε βοήθεια για λίστα εντολών.")
    }
}
```

Εικόνα 4.48: handleVoiceCommand()

Η handleVoiceCommand (Εικόνα 4.48) εκτελεί την κατάλληλη ενέργεια ανάλογα με την εντολή που αναγνωρίστηκε. Η εντολή «τι βλέπεις» καλεί την announceDetectedObjects, οι εντολές «ρυθμίσεις» και «κλείσε» ανοίγουν ή κλείνουν το panel ρυθμίσεων, η «βοήθεια» εκφωνεί τη λίστα διαθέσιμων εντολών, το «ναι» λειτουργεί ως επιβεβαίωση στη ροή ερώτησης και το «όχι» αγνοείται. Τέλος, οποιαδήποτε μη αναγνωρίσιμη εντολή προτρέπει τον χρήστη να χρησιμοποιήσει την εντολή «βοήθεια» για να θυμηθεί όλες τις διαθέσιμες εντολές.

```kotlin
private fun announceDetectedObjects() {
    if (lastResults.isEmpty()) {
        speakGreekText(getString( resId = R.string.tts_nothing_detected))
        return
    }

    val sb = StringBuilder(getString( resId = R.string.tts_i_see) + " ")
    lastResults.forEachIndexed { index, result ->
        val greekLabel = CocoGreek.CocoTranslated.translations[result.box.clsName.lowercase()] ?: result.box.clsName
        val distance = result.box.cnf
        if (distance > 0) {
            val distStr = String.format("%.1f", distance)
            sb.append("$greekLabel ${getString( resId = R.string.tts_at_distance)} $distStr ${getString( resId = R.string.tts_meters)}")
        } else {
            sb.append(greekLabel)
        }
        if (index < lastResults.size - 1) {
            sb.append(", ")
        }
    }
    speakGreekText(sb.toString())
}
```

Εικόνα 4.49: announceDetectedObjects()

Η announceDetectedObjects (Εικόνα 4.49) εκφωνεί τα αντικείμενα που βλέπει η κάμερα. Αν δεν υπάρχουν αποτελέσματα, εκφωνεί ότι δεν εντοπίστηκε τίποτα. Αλλιώς, για κάθε αντικείμενο χτίζει μια πρόταση με την ελληνική ετικέτα. Αν υπάρχει και διαθέσιμη εκτίμηση απόστασης, προσθέτει και την απόσταση σε μέτρα. Τα αντικείμενα χωρίζονται με κόμμα και εκφωνούνται ως ενιαία πρόταση.

4.7 Διαχείριση Ρυθμίσεων και Επιβίωση Διαμόρφωσης

```kotlin
private val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = "yolo_anti_settings")

3 Usages
data class SettingsSnapshot(
    val vibrationDurationMs: Long = 50L,
    val pulseGapMs: Long = 20L,
    val maxRadiusPx: Int = 10,
    val confidenceThreshold: Float = 0.5f,
    val preset: String = "CUSTOM"
)

3 Usages
class UserSettingsRepository(private val context: Context) {

    10 Usages
    private object Keys {
        2 Usages
        val VIBRATION_DURATION = longPreferencesKey( name = "vibration_duration_ms")
        2 Usages
        val PULSE_GAP = longPreferencesKey( name = "pulse_gap_ms")
        2 Usages
        val MAX_RADIUS = intPreferencesKey( name = "max_radius_px")
        2 Usages
        val CONFIDENCE = floatPreferencesKey( name = "confidence_threshold")
        2 Usages
        val PRESET = stringPreferencesKey( name = "preset")
    }

    1 Usage
    val settingsFlow: Flow<SettingsSnapshot> = context.dataStore.data.map { prefs ->
        SettingsSnapshot(
            vibrationDurationMs = prefs[Keys.VIBRATION_DURATION] ?: 50L,
            pulseGapMs = prefs[Keys.PULSE_GAP] ?: 20L,
            maxRadiusPx = prefs[Keys.MAX_RADIUS] ?: 10,
            confidenceThreshold = prefs[Keys.CONFIDENCE] ?: 0.5f,
            preset = prefs[Keys.PRESET] ?: "CUSTOM"
        )
    }
```

Εικόνα 4.50: UserSettingsRepository

Η κλάση UserSettingsRepository της Εικόνας 4.50 διαχειρίζεται την αποθήκευση και ανάκτηση των ρυθμίσεων του χρήστη μέσω Jetpack Datastore. Οι ρυθμίσεις αποθηκεύονται σε αρχείο. Η SettingsSnapshot είναι ένα data class που ομαδοποιεί όλες τις παραμέτρους: διάρκεια δόνησης, κενό μεταξύ παλμών, μέγιστη ακτίνα αναζήτησης ακμής, κατώφλι εμπιστοσύνης, και preset δόνησης. Κάθε παράμετρος έχει μια προκαθορισμένη τιμή.

```
settingsRepo = UserSettingsRepository(applicationContext)
val savedSettings = settingsRepo.snapshotBlocking()
vibrationDuration = savedSettings.vibrationDurationMs
pulseGap = savedSettings.pulseGapMs
maxRadius = savedSettings.maxRadiusPx
confidenceThreshold = savedSettings.confidenceThreshold
currentPreset = when (savedSettings.preset) {
    "CONTINUOUS" -> VibrationPreset.CONTINUOUS
    "PULSE" -> VibrationPreset.PULSE
    else -> VibrationPreset.CUSTOM
}
if (viewModel.bottomSheetWasExpanded) {
    Handler(Looper.getMainLooper()).post {
        bottomSheetBehavior.state = BottomSheetBehavior.STATE_EXPANDED
    }
}
```

Εικόνα 4.51: snapshotBlocking()

Κατά την εκκίνηση, οι αποθηκευμένες ρυθμίσεις φορτώνονται μέσω snapshotBlocking και εφαρμόζονται στις αντίστοιχες μεταβλητές.

```
private val saveDebounceHandler = Handler(Looper.getMainLooper())
2 Usages
private val saveDurationRunnable = Runnable { lifecycleScope.launch { settingsRepo.saveVibrationDuration(vibrationDuration) } }
2 Usages
private val saveGapRunnable = Runnable { lifecycleScope.launch { settingsRepo.savePulseGap(pulseGap) } }
2 Usages
private val saveRadiusRunnable = Runnable { lifecycleScope.launch { settingsRepo.saveMaxRadius(maxRadius) } }
```

Εικόνα 4.52: saveDebounceHandler

Όταν ο χρήστης αλλάξει μια τιμή στο πεδίο, η εφαρμογή δεν την αποθηκεύει αμέσως. Περιμένει 300ms σε περίπτωση αλλαγής της τιμής, αποφεύγοντας έτσι περιττές εγγραφές.

```
etDuration?.addTextChangedListener( watcher = object : TextWatcher {
    override fun afterTextChanged(s: Editable?) {
        vibrationDuration = s.toString().toLongOrNull()?.coerceAtLeast( minimumValue = 1L) ?: vibrationDuration
        if (!isProgrammaticChange && currentPreset != VibrationPreset.CUSTOM) {
            currentPreset = VibrationPreset.CUSTOM
            Toast.makeText( context = this@MainActivity, resId = R.string.custom_mode, duration = Toast.LENGTH_SHORT).show()
        }
        if (!isProgrammaticChange) {
            saveDebounceHandler.removeCallbacks( r = saveDurationRunnable)
            saveDebounceHandler.postDelayed( r = saveDurationRunnable, delayMillis = 300)
        }
    }
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {}
})
```

Εικόνα 4.53: TextWatcher

Όταν ο χρήστης τροποποιήσει χειροκίνητα μια παράμετρο ενώ χρησιμοποιεί κάποιο έτοιμο preset, το preset αλλάζει αυτόματα σε CUSTOM και εμφανίζεται ειδοποίηση ότι πλέον χρησιμοποιούνται προσαρμοσμένες ρυθμίσεις.

4.8 Δομή Κώδικα και Περιγραφή Βασικών Κλάσεων

4.8.1 MainActivity

```
class MainActivity : AppCompatActivity(), InstanceSegmentation.InstanceSegmentationListener, SensorEventListener {

    20 Usages
    private lateinit var binding: ActivityMainBinding
    4 Usages
    private lateinit var instanceSegmentation: InstanceSegmentation
    2 Usages
    private lateinit var drawImages: DrawImages
    2 Usages
    private lateinit var previewView: PreviewView
    11 Usages
    private lateinit var tts: TextToSpeech
    3 Usages
    private var isPressing = false
    8 Usages
    @Volatile private var lastResults: List<SegmentationResult> = emptyList()
    // 1. FOV CORRECTION (Scale Factor)
// Most cameras crop 5-10% of the image for the video buffer.
// If distance is TOO SHORT, REDUCE this number (try 0.85 to 0.95).
// If distance is TOO LONG, INCREASE this number (up to 1.0).
    1 Usage
    private val FOV_CORRECTION_FACTOR = 0.92

    1 Usage
    private val PITCH_OFFSET_DEG = -4.0
    8 Usages
    private var currentVibrationPanel: View? = null
    5 Usages
    private var etDuration: EditText? = null
    5 Usages
    private var etPulseGap: EditText? = null
    3 Usages
    private var etMaxRadius: EditText? = null
    // Add these new variables for TTS cooldown
    3 Usages
    private var lastSpokenLabel: String? = null
    3 Usages
    private var lastSpokenTime: Long = 0
    1 Usage
    private val ttsCooldown: Long = 2000 // 2 second cooldown between TTS announcements

    3 Usages
    private var seekConfidence: SeekBar? = null
    3 Usages
    private var tvConfidenceValue: TextView? = null
```

Εικόνα 4.54: MainActivity δηλώσεις (1/2)

Η MainActivity κληρονομεί από AppCompatActivity και υλοποιεί δύο interfaces: InstanceSegmentationListener για τα αποτελέσματα ανίχνευσης, SensorEventListener για τον αισθητήρα κλίσης. Δηλώνονται οι βασικές μεταβλητές: το binding για πρόσβαση στα views, ο InstanceSegmentation για το μοντέλο ανίχνευσης, ο DrawImages για τη σχεδίαση αποτελεσμάτων, η PreviewView για την κάμερα, και το TextToSpeech για την εκφώνηση.

Η lastResults αποθηκεύει τη λίστα αντικειμένων που εντόπισε το τελευταίο καρέ. Ο FOV_CORRECTION_FACTOR διορθώνει το οπτικό πεδίο επειδή οι περισσότερες κάμερες περικόπτουν 5-10% της εικόνας. Ο PITCH_OFFSET_DEG αντισταθμίζει την τυπική κλίση του τηλεφώνου στο χέρι.

```
7 Usages
83          private var confidenceThreshold: Float = 0.5f
12 Usages
84          private var isProgrammaticChange = false  // Add this flag
1 Usage
85          private val cameraHeightMeters = 1.2f
6 Usages
86          @Volatile private var verticalFOV: Double = 0.0
5 Usages
87          private var cameraExecutor: ExecutorService? = null
4 Usages
88          private lateinit var sensorManager: SensorManager
89
90
2 Usages
91          private var rotationVectorSensor: Sensor? = null
16 Usages
92          private enum class VibrationPreset { CONTINUOUS, PULSE, CUSTOM }
9 Usages
93          private var currentPreset = VibrationPreset.CUSTOM
94
14 Usages
95          private var vibrationDuration = 50L //ms
11 Usages
96          private var pulseGap = 20L //ms
6 Usages
97          private var maxRadius = 10 // (px) max radius to search for edge
4 Usages
98          private var tapCount = 0
2 Usages
99          private var lastTapTime = 0L
1 Usage
100         private val tapTimeout = 400L // max gap between taps to count as consecutive
101
7 Usages
102         private lateinit var soundPool: SoundPool
2 Usages
103         private var listeningSoundId: Int = 0
6 Usages
104         private var speechRecognizer: SpeechRecognizer? = null
2 Usages
105         private var stopListeningSoundId: Int = 0
6 Usages
106         private var isListening = false
16 Usages
107         private lateinit var bottomSheetBehavior: BottomSheetBehavior<LinearLayout>
8 Usages
108         private lateinit var settingsRepo: UserSettingsRepository
6 Usages
109         private val saveDebounceHandler = Handler(Looper.getMainLooper())
2 Usages
110         private val saveDurationRunnable = Runnable { lifecycleScope.launch { settingsRepo.saveVibrationDuration(vibrationDuration) } }
2 Usages
111         private val saveGapRunnable = Runnable { lifecycleScope.launch { settingsRepo.savePulseGap(pulseGap) } }
2 Usages
112         private val saveRadiusRunnable = Runnable { lifecycleScope.launch { settingsRepo.saveMaxRadius(maxRadius) } }
5 Usages
113         private lateinit var accessibilityManager: AccessibilityManager
2 Usages
114         private val viewModel: MainViewModel by viewModels()
```

Εικόνα 4.55: MainActivity δηλώσεις (2/2)

Συνεχίζοντας τις δηλώσεις μεταβλητών: το confidenceThreshold ορίζει το ελάχιστο κατώφλι εμπιστοσύνης για αποδοχή ανιχνεύσεων. Το cameraHeightMeters(1.2m) χρησιμοποιείται στον υπολογισμό απόστασης αντικειμένων.
Οι μεταβλητές verticalFOV, cameraExecutor και sensorManager σχετίζονται με την κάμερα και τον αισθητήρα κλίσης. Το enum VibrationPreset ορίζει τα τρία preset μοτίβα δόνησης με τις παραμέτρους vibrationDuration, pulseGap και maxRadius να ρυθμίζονται από τον χρήστη.
Για τη φωνητική αλληλεπίδραση δηλώνονται ο SoundPool, ο SpeechRecognizer, το flag isListening και οι μεταβλητές tapCount, lastTapTime, tapTimeout για την ανίχνευση 4 διαδοχικών πατημάτων.

```
override fun onSensorChanged(event: SensorEvent?) {
    if (event?.sensor?.type == Sensor.TYPE_ROTATION_VECTOR) {
        val rotationMatrix = FloatArray( size = 9)
        val adjustedRotationMatrix = FloatArray( size = 9)
        val orientationAngles = FloatArray( size = 3)

        SensorManager.getRotationMatrixFromVector( R = rotationMatrix, rotationVector = event.values)


        SensorManager.remapCoordinateSystem(
            inR = rotationMatrix,
            X = SensorManager.AXIS_X,
            Y = SensorManager.AXIS_Z,
            outR = adjustedRotationMatrix
        )

        SensorManager.getOrientation( R = adjustedRotationMatrix, values = orientationAngles)


        val rawPitch = Math.toDegrees(orientationAngles[1].toDouble())


        val alpha = 0.1
        sensorPitch = alpha * rawPitch + (1 - alpha) * sensorPitch
    }
}

override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
    // We don't need to do anything here
}
```

Εικόνα 4.56: onSensorChanged()

Στο onSensorChanged υπολογίζεται η κλίση της συσκευής χρησιμοποιώντας τον αισθητήρα ROTATION_VECTOR. Τα δεδομένα του αισθητήρα μετατρέπονται σε πίνακα περιστροφής, ο οποίος προσαρμόζεται στο σύστημα συντεταγμένων της κάμερας μέσω remapCoordinateSystem. Από τον προσαρμοσμένο πίνακα εξάγεται η γωνία pitch σε μοίρες.
Για να αποφευχθεί ο θόρυβος του αισθητήρα, εφαρμόζεται εκθετική εξομάλυνση με συντελεστή alpha=0.1. Δηλαδή μόνο το 10% της νέας μέτρησης επηρεάζει την τιμή sensorPitch, ενώ το 90% προέρχεται από την προηγούμενη τιμή. Ως αποτέλεσμα, η κλίση που χρησιμοποιείται στον υπολογισμό απόστασης αντικειμένων παραμένει πιο σταθερή χωρίς απότομες μεταβολές.

```
inner class ImageAnalyzer : ImageAnalysis.Analyzer {
    override fun analyze(imageProxy: ImageProxy) {
        val bitmapBuffer =
            Bitmap.createBitmap(imageProxy.width, imageProxy.height, config = Bitmap.Config.ARGB_8888)
        imageProxy.use { bitmapBuffer.copyPixelsFromBuffer( src = imageProxy.planes[0].buffer) }

        val matrix = Matrix().apply {
            postRotate( degrees = imageProxy.imageInfo.rotationDegrees.toFloat())
        }

        val rotatedBitmap = Bitmap.createBitmap(
            source = bitmapBuffer, x = 0, y = 0, bitmapBuffer.width, bitmapBuffer.height,
            m = matrix, filter = true
        )



        imageProxy.close()
        instanceSegmentation.invoke( frame = rotatedBitmap)
    }
}
```

Εικόνα 4.57: ImageAnalyzer

Η εσωτερική κλάση ImageAnalyzer υλοποιεί το ImageAnalysis.Analyzer και αποτελεί τον σύνδεσμο μεταξύ της κάμερας και του μοντέλου ανίχνευσης. Σε κάθε καρέ αντιγράφει τα pixel από τον buffer της κάμερας σε ένα Bitmap μορφής ARGB_8888. Στη συνέχεια, εφαρμόζει περιστροφή στην εικόνα κατά τις μοίρες που υποδεικνύει ο αισθητήρας προσανατολισμού ,ώστε η εικόνα να είναι πάντα σωστά προσανατολισμένη ανεξάρτητα από τη θέση της συσκευής.

```
override fun onDestroy() {
    super.onDestroy()
    instanceSegmentation.close()
    if (::tts.isInitialized) {
        tts.stop()
        tts.shutdown()
    }
    if (isListening) {
        speechRecognizer?.destroy()
    }
    if (::soundPool.isInitialized) {
        soundPool.release()
    }
    cameraExecutor?.shutdown()
    cameraExecutor = null
}
```

Εικόνα 4.58: onDestroy()

Η onDestroy απελευθερώνει όλους τους πόρους κατά τον τερματισμό της Activity. Κλείνει το μοντέλο ανίχνευσης, σταματά και τερματίζει το TextToSpeech και καταστρέφει τον speech recognizer αν είναι ενεργός. Τέλος απελευθερώνει το SoundPool και τερματίζει τον executor της κάμερας.

```
private fun updateStatusBar(count: Int) {
    val statusText = when (count) {
        0 -> "Σκανάρισμα…"
        1 -> "Βρέθηκε 1 αντικείμενο"
        else -> "Βρέθηκαν {count} αντικείμενα"
    }
    binding.tvStatusBar.text = statusText
    if (accessibilityManager.isEnabled && accessibilityManager.isTouchExplorationEnabled) {
        binding.tvStatusBar.announceForAccessibility(statusText)
    }
}
```

Εικόνα 4.59: updateStatusBar()

Η updateStatusBar ενημερώνει τη γραμμή κατάστασης ανάλογα με τον αριθμό αντικειμένων που εντοπίστηκαν. Αν δεν βρέθηκε κανένα είναι «Σκανάρισμα», αν βρέθηκε 1 είναι «Βρέθηκε 1 αντικείμενο», και «Βρέθηκαν x αντικείμενα» για περισσότερα.

4.8.2 InstanceSegmentation

```
class InstanceSegmentation(
    context: Context,
    modelPath: String,
    labelPath: String?,
    private val instanceSegmentationListener: InstanceSegmentationListener,
    private val message: (String) -> Unit
) {
```

Εικόνα 4.60: InstanceSegmentation - Listener

Η κλάση InstanceSegmentation δέχεται ως παραμέτρους το context, τη διαδρομή του μοντέλου, τη διαδρομή των ετικετών, έναν listener για τα αποτελέσματα ανίχνευσης, και μια συνάρτηση για μηνύματα σφαλμάτων.

```
fun setConfidenceThreshold(threshold: Float) {
    confidenceThreshold = threshold
}
```

Εικόνα 4.61: setConfidenceThreshold()

Η setConfidenceThreshold επιτρέπει την αλλαγή του κατωφλίου εμπιστοσύνης κατά τη διάρκεια εκτέλεσης.

```
private val imageProcessor = ImageProcessor.Builder()
    .add(NormalizeOp(INPUT_MEAN, stddev = INPUT_STANDARD_DEVIATION))
    .add(CastOp( destinationType = INPUT_IMAGE_TYPE))
    .build()
```

Εικόνα 4.62: imageProcessor()

Στον imageProcessor προετοιμάζεται η εικόνα πριν φτάσει στο μοντέλο. Πρώτα κανονικοποιούνται οι τιμές pixel αφαιρώντας τον μέσο όρο και διαιρώντας με την τυπική απόκλιση. Στη συνέχεια μετατρέπει τον τύπο δεδομένων στη μορφή που το απαιτεί το μοντέλο.

```
init {
    val options = Interpreter.Options()
    options.setNumThreads(4)

    val model = FileUtil.loadMappedFile(context, filePath = modelPath)
    interpreter = Interpreter( byteBuffer = model, options)

    labels.addAll( elements = extractNamesFromMetadata(model))
    if (labels.isEmpty()) {
        if (labelPath == null) {
            message("Model not contains metadata, provide LABELS_PATH in Constants.kt")
            labels.addAll( elements = MetaData.TEMP_CLASSES)
        } else {
            labels.addAll( elements = extractNamesFromLabelFile(context, labelPath))
        }
    }

    val inputShape = interpreter.getInputTensor( inputIndex = 0)?.shape()
    val outputShape0 = interpreter.getOutputTensor( outputIndex = 0)?.shape()
    val outputShape1 = interpreter.getOutputTensor( outputIndex = 1)?.shape()

    if (inputShape != null) {
        tensorWidth = inputShape[1]
        tensorHeight = inputShape[2]


        if (inputShape[1] == 3) {
            tensorWidth = inputShape[2]
            tensorHeight = inputShape[3]
        }
    }

    if (outputShape0 != null) {
        if (outputShape0[1] > outputShape0[2]) {
            isEndToEnd = true
            numDetections = outputShape0[1]
            numFields = outputShape0[2]
        } else {
            isEndToEnd = false
            numChannel = outputShape0[1]
            numElements = outputShape0[2]
        }
    }

    if (outputShape1 != null) {
        if (outputShape1[1] == 32) {
            masksNum = outputShape1[1]
            xPoints = outputShape1[2]
            yPoints = outputShape1[3]
        } else {
            xPoints = outputShape1[1]
            yPoints = outputShape1[2]
            masksNum = outputShape1[3]
        }
    }
}
```

Εικόνα 4.63: init

Στο init φορτώνεται το μοντέλο με 4 threads. Οι ετικέτες αντικειμένων εξάγονται πρώτα από τα metadata του μοντέλου.
Στη συνέχεια διαβάζονται οι διαστάσεις των tensors εισόδου και εξόδου. Για την είσοδο αν η δεύτερη διάσταση είναι 3 (κανάλια χρώματος), σημαίνει ότι το μοντέλο χρησιμοποιεί μορφή channel-first(CHW) αντί channel-last(HWC), και οι διαστάσεις πλάτους/ύψους προσαρμόζονται αντίστοιχα.

Για τον πρώτο tensor εξόδου (ανιχνεύσεις), συγκρίνονται οι δύο διαστάσεις: αν outputShape0[1] > outputShape0[2], το μοντέλο λειτουργεί σε end-to-end μορφή και αποθηκεύονται numDetections και numFields. Διαφορετικά, χρησιμοποιείται η παραδοσιακή μορφή και αποθηκεύονται numChannel και numElements. Με απλούστερους όρους, αν ο αριθμός των γραμμών είναι μεγαλύτερος από τον αριθμό των στηλών (300 ανιχνεύσεις x 38 πεδία), πρόκειται για end-to-end μοντέλο. Αλλιώς, πρόκειται για προγενέστερο μοντέλο (π.χ. 116 x 8400).

Για τον δεύτερο tensor εξόδου(μάσκες), ελέγχεται αν η δεύτερη διάσταση είναι 32. Αν ναι, τότε η διάταξη είναι [1,32,H,W], αλλιώς είναι [1,H,W,32]. Αυτή η αυτόματη ανίχνευση επιτρέπει στον κώδικα να λειτουργεί σωστά με διαφορετικές εκδόσεις μοντέλων.

```
fun invoke(frame: Bitmap) {
    if (tensorWidth == 0 || tensorHeight == 0
        || xPoints == 0 || yPoints == 0 || masksNum == 0) {
        instanceSegmentationListener.onError("Interpreter not initialized properly")
        return
    }
    if (!isEndToEnd && (numChannel == 0 || numElements == 0)) {
        instanceSegmentationListener.onError("Interpreter not initialized properly")
        return
    }
    if (isEndToEnd && (numDetections == 0 || numFields == 0)) {
        instanceSegmentationListener.onError("Interpreter not initialized properly")
        return
    }

    var preProcessTime = SystemClock.uptimeMillis()

    val imageBuffer = preProcess(frame)

    val coordinatesBuffer = if (isEndToEnd) {
        TensorBuffer.createFixedSize(
            shape = intArrayOf(1, numDetections, numFields),
            dataType = OUTPUT_IMAGE_TYPE
        )
    } else {
        TensorBuffer.createFixedSize(
            shape = intArrayOf(1, numChannel, numElements),
            dataType = OUTPUT_IMAGE_TYPE
        )
    }

    val maskProtoBuffer = TensorBuffer.createFixedSize(
        shape = intArrayOf(1, xPoints, yPoints, masksNum),
        dataType = OUTPUT_IMAGE_TYPE
    )

    val outputBuffer = mapOf<Int, Any>(
        0 to coordinatesBuffer.buffer.rewind(),
        1 to maskProtoBuffer.buffer.rewind()
    )

    preProcessTime = SystemClock.uptimeMillis() - preProcessTime

    var interfaceTime = SystemClock.uptimeMillis()

    interpreter.runForMultipleInputsOutputs( inputs = imageBuffer, outputs = outputBuffer)

    interfaceTime = SystemClock.uptimeMillis() - interfaceTime

    var postProcessTime = SystemClock.uptimeMillis()

    val bestBoxes = if (isEndToEnd) {
        bestBoxE2E(coordinatesBuffer.floatArray)
    } else {
        bestBox(coordinatesBuffer.floatArray)
    } ?: run {
        instanceSegmentationListener.onEmpty()
        return
    }

    val maskProto = reshapeMaskOutput(maskProtoBuffer.floatArray)

    val segmentationResults = bestBoxes.map {
        SegmentationResult(
            box = it,
            mask = getFinalMask(frame.width, frame.height, output0 = it, output1 = maskProto)
        )
    }

    postProcessTime = SystemClock.uptimeMillis() - postProcessTime

    instanceSegmentationListener.onDetect(
        preProcessTime = preProcessTime,
        interfaceTime = interfaceTime,
        postProcessTime = postProcessTime,
        results = segmentationResults,
        bitmap_width = frame.width,
        bitmap_height = frame.height
    )
}
```

Εικόνα 4.64: invoke()

Η invoke είναι η κεντρική συνάρτηση που εκτελεί την ανίχνευση σε κάθε καρέ. Αρχικά ελέγχει ότι ο interpreter έχει αρχικοποιηθεί σωστά. Στη συνέχεια, η εικόνα προεπεξεργάζεται μέσω preprocess και δημιουργούνται δύο buffers εξόδου: ένας για τις συντεταγμένες των bounding boxes και ένας για τα mask prototypes.

Ο interpreter εκτελεί το μοντέλο και μετριέται ο χρόνος εκτέλεσης. Από το πρώτο buffer εξάγονται τα καλύτερα boxes μέσω bestBox() ή bestBoxE2E(), ανάλογα με τη μορφή του μοντέλου. Για κάθε box δημιουργείται ένα SegmentationResult που περιλαμβάνει το box και την τελική μάσκα μέσω getFinalMask. Τέλος, τα αποτελέσματα μαζί με τους χρόνους επεξεργασίας στέλνονται στον listener μέσω onDetect.

```
private fun bestBox(array: FloatArray) : List<Output0>? {

    val output0List = mutableListOf<Output0>()

    for (c in 0 .. until .. < numElements) {
        var maxConf = confidenceThreshold
        var maxIdx = -1
        var currentInd = 4
        var arrayIdx = c + numElements * currentInd

        while (currentInd < (numChannel - masksNum)){
            if (array[arrayIdx] > maxConf) {
                maxConf = array[arrayIdx]
                maxIdx = currentInd - 4
            }
            currentInd++
            arrayIdx += numElements
        }

        if (maxConf > confidenceThreshold) {
            val clsName = labels[maxIdx]
            val cx = array[c]
            val cy = array[c + numElements]
            val w = array[c + numElements * 2]
            val h = array[c + numElements * 3]
            val x1 = cx - (w/2F)
            val y1 = cy - (h/2F)
            val x2 = cx + (w/2F)
            val y2 = cy + (h/2F)
            if (x1 < 0F || x1 > 1F) continue
            if (y1 < 0F || y1 > 1F) continue
            if (x2 < 0F || x2 > 1F) continue
            if (y2 < 0F || y2 > 1F) continue

            val maskWeight = mutableListOf<Float>()
            while (currentInd < numChannel){
                maskWeight.add(array[arrayIdx])
                currentInd++
                arrayIdx += numElements
            }

            output0List.add(
                Output0(
                    x1 = x1, y1 = y1, x2 = x2, y2 = y2,
                    cx = cx, cy = cy, w = w, h = h,
                    cnf = maxConf, cls = maxIdx, clsName = clsName,
                    maskWeight = maskWeight
                )
            )
        }
    }

    if (output0List.isEmpty()) return null

    return applyNMS(output0List)
}
```

Εικόνα 4.65: bestBox()

Η bestBox εξάγει τα αντικείμενα από τον tensor εξόδου του μοντέλου. Για κάθε υποψήφιο αντικείμενο, σαρώνονται οι κλάσεις και βρίσκεται η κλάση με τη μεγαλύτερη εμπιστοσύνη. Αν ξεπερνά το confidenceThreshold, εξάγονται οι συντεταγμένες κέντρου και διαστάσεις, και υπολογίζονται οι γωνίες του box.
Τέλος συλλέγονται τα mask weights του αντικειμένου και δημιουργείται ένα Output0 με όλα τα στοιχεία. Η λίστα περνά από applyNMS για αφαίρεση διπλότυπων ανιχνεύσεων.

```
private fun applyNMS(output0List: List<Output0>) : MutableList<Output0> {
    val sortedBoxes = output0List.sortedByDescending { it.cnf }.toMutableList()
    val selectedBoxes = mutableListOf<Output0>()

    while(sortedBoxes.isNotEmpty()) {
        val first = sortedBoxes.first()
        selectedBoxes.add(first)
        sortedBoxes.remove( element = first)

        val iterator = sortedBoxes.iterator()
        while (iterator.hasNext()) {
            val nextBox = iterator.next()
            val iou = calculateIoU( box1 = first,  box2 = nextBox)
            if (iou >= IOU_THRESHOLD) {
                iterator.remove()
            }
        }
    }

    return selectedBoxes
}
```

Εικόνα 4.66: applyNMS()

Η applyNMS αφαιρεί τις διπλότυπες ανιχνεύσεις. Ταξινομεί τα αντικείμενα κατά φθίνουσα εμπιστοσύνη και επιλέγει αυτό με την υψηλότερη. Στη συνέχεια, για κάθε εναπομείναν αντικείμενο υπολογίζει την επικάλυψη (IoU) με το επιλεγμένο. Αν ξεπερνά το IOU κατώφλι, σημαίνει ότι αφορούν το ίδιο αντικείμενο και το λιγότερο σίγουρο αφαιρείται. Η διαδικασία επαναλαμβάνεται μέχρι να εξεταστούν όλα.

```
private fun calculateIoU(box1: Output0, box2: Output0): Float {
    val x1 = maxOf( a = box1.x1,  b = box2.x1)
    val y1 = maxOf( a = box1.y1,  b = box2.y1)
    val x2 = minOf( a = box1.x2,  b = box2.x2)
    val y2 = minOf( a = box1.y2,  b = box2.y2)
    val intersectionArea = maxOf( a = 0F,  b = x2 - x1) * maxOf( a = 0F,  b = y2 - y1)
    val box1Area = box1.w * box1.h
    val box2Area = box2.w * box2.h
    return intersectionArea / (box1Area + box2Area - intersectionArea)
}
```

Εικόνα 4.67: calculateIoU()

Η calculateIoU υπολογίζει το ποσοστό επικάλυψης δύο boxes(Intersection over Union). Βρίσκει το ορθογώνιο τομής μέσω maxOf/minOf στις συντεταγμένες, υπολογίζει τα εμβαδά τομής και ένωσης, και επιστρέφει τον λόγο τους. Τιμή κοντά στο 1 σημαίνει σχεδόν πλήρη επικάλυψη, ενώ τιμή 0 σημαίνει ότι δεν υπάρχει κοινό σημείο.

```
private fun getFinalMask(
    width: Int,
    height: Int,
    output0: Output0,
    output1: List<Array<FloatArray>>
): Array<FloatArray> {
    val output1Copy = output1.clone()
    val relX1 = output0.x1 * xPoints
    val relY1 = output0.y1 * yPoints
    val relX2 = output0.x2 * xPoints
    val relY2 = output0.y2 * yPoints

    val zero: Array<FloatArray> = Array( size = yPoints) { FloatArray( size = xPoints) { 0F } }
    for ((index, proto) in output1Copy.withIndex()) {
        for (y in 0 ≤ until < yPoints) {
            for (x in 0 ≤ until < xPoints) {
                proto[y][x] *= output0.maskWeight[index]
                if (x + 1  > relX1 && x + 1 < relX2 && y + 1 > relY1 && y + 1 < relY2) {
                    zero[y][x] += proto[y][x]
                }
            }
        }
    }

    return zero.scaleMask( targetWidth = 450 ,  targetHeight = 600)
}
```

Εικόνα 4.68: getFinalMask()

Η getFinalMask δημιουργεί την τελική μάσκα κατάτμησης για κάθε αντικείμενο. Συνδυάζει τα mask prototypes (γενικά σχήματα που παράγει το μοντέλο) με τα mask weights του συγκεκριμένου αντικειμένου. Κάθε prototype πολλαπλασιάζεται με το αντίστοιχο βάρος και τα αποτελέσματα αθροίζονται. Η μάσκα περιορίζεται μόνο εντός του bounding box, ώστε τα pixel εκτός ορίων να μηδενίζονται.

```
private fun reshapeMaskOutput(floatArray: FloatArray): List<Array<FloatArray>> {
    return List( size = masksNum) { mask ->
        Array( size = xPoints) { r ->
            FloatArray( size = yPoints) { c ->
                floatArray[masksNum * yPoints * r + masksNum * c + mask]
            }
        }
    }
}
```

Εικόνα 4.69: reshapeMaskOutput()

Η reshapeMaskOutput μετατρέπει τον μονοδιάστατο πίνακα εξόδου του interpreter σε τρισδιάστατη δομή.

4.8.3 DrawImages

```
class DrawImages(private val context: Context) {

    private val boxColors = listOf(
        R.color.overlay_orange,
        R.color.overlay_blue,
        R.color.overlay_green,
        R.color.overlay_red,
        R.color.overlay_pink,
        R.color.overlay_cyan,
        R.color.overlay_purple,
        R.color.overlay_gray,
        R.color.overlay_teal,
        R.color.overlay_yellow,
    )
```

Εικόνα 4.70: DrawImages – χρώματα overlay

Η κλάση DrawImages αναλαμβάνει τη σχεδίαση των αποτελεσμάτων κατάτμησης πάνω στην εικόνα. Ορίζει μια λίστα 10 χρωμάτων. Κάθε αντικείμενο παίρνει διαφορετικό χρώμα με βάση την κλάση για να ξεχωρίζουν οπτικά.

```
fun invoke(results: List<SegmentationResult>) : Bitmap {
    val width = results.first().mask[0].size
    val height = results.first().mask.size
    val combined = Bitmap.createBitmap(width, height, config = Bitmap.Config.ARGB_8888)

    results.forEach { result ->
        val colorResId = boxColors[result.box.cls % 10]
        applyTransparentOverlay(context, overlay = combined, segmentationResult = result, overlayColorResId = colorResId)
    }
    return combined
}
```

Εικόνα 4.71: DrawImages.invoke()

Η invoke δημιουργεί ένα κενό bitmap στις διαστάσεις της μάσκας και για κάθε αποτέλεσμα καλεί την applyTransparentOverlay, που σχεδιάζει ημιδιαφανή επικάλυψη στα pixel του αντικείμενο με το αντίστοιχο χρώμα. Το τελικό bitmap επιστρέφεται με όλα τα αντικείμενα χρωματισμένα.

```
38      private fun applyTransparentOverlay(context: Context, overlay: Bitmap, segmentationResult: SegmentationResult, overlayColorResId: Int) {
39          val width = overlay.width
40          val height = overlay.height
41
42          val overlayColor = ContextCompat.getColor(context, overlayColorResId)
43
44          for (y in 0 ≤ until < height) {
45              for (x in 0 ≤ until < width) {
46                  val maskValue = segmentationResult.mask[y][x]
47                  if (maskValue > 0) {
48                      overlay.setPixel(x, y, color = applyTransparentOverlayColor(overlayColor))
49                  }
50              }
51          }
52
53          val canvas = Canvas( bitmap = overlay)
54
55          val boxPaint = Paint().apply {
56              color = ContextCompat.getColor(context, overlayColorResId)
57              strokeWidth = 2F
58              style = Paint.Style.STROKE
59          }
60
61          val box = segmentationResult.box
62
63          val left = (box.x1 * width).toInt()
64          val top = (box.y1 * height).toInt()
65          val right = (box.x2 * width).toInt()
66          val bottom = (box.y2 * height).toInt()
67
68          canvas.drawRect(left.toFloat(), top.toFloat(), right.toFloat(), bottom.toFloat(), boxPaint)
```

Εικόνα 4.72: applyTransparentOverlay() (1/2)

Η applyTransparentOverlay σχεδιάζει την οπτική απεικόνιση κάθε αντικειμένου. Πρώτα, για κάθε pixel της μάσκας με θετική τιμή, εφαρμόζει ημιδιαφανές χρώμα στο bitmap. Το χρώμα παρουσιάζεται ως ημιδιαφανές για να μην κρύβεται η εικόνα από κάτω.

Στη συνέχεια σχεδιάζεται το bounding box ως ορθογώνιο περίγραμμα γύρω από το αντικείμενο. Οι συντεταγμένες του μοντέλου μετατρέπονται σε pixel πολλαπλασιάζοντας με τις διαστάσεις της εικόνας. Πάνω από το bounding box εμφανίζεται μια ετικέτα με το ελληνικό όνομα του αντικειμένου, το ποσοστό εμπιστοσύνης και την εκτιμώμενη απόσταση σε μέτρα. Η ετικέτα σχεδιάζεται ως λευκό κείμενο πάνω στο χρωματιστό φόντο για να είναι ευανάγνωστη.

```
        canvas.drawRect(left.toFloat(), top.toFloat(), right.toFloat(), bottom.toFloat(), boxPaint)

        val textBackgroundPaint = Paint().apply {
            color = ContextCompat.getColor(context, overlayColorResId)
            style = Paint.Style.FILL
        }

        val textPaint = Paint().apply {
            color = Color.WHITE
            style = Paint.Style.FILL
            textSize = 16f
        }

        val greekLabel = CocoGreek.CocoTranslated.translations[box.clsName.lowercase()] ?: box.clsName

        val confidence = segmentationResult.box.originalCnf
        val distance = segmentationResult.box.cnf
        val confidencePercent = (confidence * 100).toInt()

        val distanceString = String.format("%.2f", distance)
        val finalLabel = "$greekLabel ($confidencePercent%) : ${distanceString}m"


        val bounds = android.graphics.Rect()
        textPaint.getTextBounds( text = finalLabel,  start = 0,  end = finalLabel.length, bounds)

        val textWidth = textPaint.measureText(finalLabel)
        val textHeight = bounds.height()
        val padding = 2f  // Use float for consistency

        canvas.drawRect(
            left.toFloat(),
            top.toFloat() - textHeight - 2 * padding,
            right = left + textWidth,  // No extra padding on the right
            bottom = top.toFloat(),
            textBackgroundPaint
        )

        canvas.drawText(finalLabel,  x = left.toFloat(),  y = top.toFloat() - padding, textPaint)
    }
```

Εικόνα 4.73: applyTransparentOverlay() (2/2)

4.8.4 CocoGreek

```
class CocoGreek {
    3 Usages
    object CocoTranslated {
        3 Usages
        val translations = mapOf(
            "person" to "άτομο",
            "bicycle" to "ποδήλατο",
            "car" to "αυτοκίνητο",
            "motorcycle" to "μοτοσικλέτα",
            "airplane" to "αεροπλάνο",
            "bus" to "λεωφορείο",
            "train" to "τρένο",
            "truck" to "φορτηγό",
            "boat" to "βάρκα",
            "traffic light" to "φανάρι",
            "fire hydrant" to "πυροσβεστικός κρούνος",
            "stop sign" to "πινακίδα στοπ",
            "parking meter" to "παρκόμετρο",
            "bench" to "παγκάκι",
            "bird" to "πουλί",
            "cat" to "γάτα",
            "dog" to "σκύλος",
            "horse" to "άλογο",
            "sheep" to "πρόβατο",
            "cow" to "αγελάδα",
            "elephant" to "ελέφαντας",
            "bear" to "αρκούδα",
            "zebra" to "ζέβρα",
            "giraffe" to "καμηλοπάρδαλη",
            "backpack" to "σακίδιο πλάτης",
            "umbrella" to "ομπρέλα",
            "handbag" to "τσάντα",
            "tie" to "γραβάτα",
            "suitcase" to "βαλίτσα",
            "frisbee" to "φρίσμπι",
            "skis" to "πέδιλα του σκι",
            "snowboard" to "σανίδα σνόουμπορντ",
            "sports ball" to "μπάλα",
            "kite" to "χαρταετός",
            "baseball bat" to "ρόπαλο μπέιζμπολ",
            "baseball glove" to "γάντι του μπέιζμπολ",
            "skateboard" to "σκέιτμπορντ",
            "surfboard" to "σανίδα του σερφ",
            "tennis racket" to "ρακέτα του τένις",
            "bottle" to "μπουκάλι",
```

Εικόνα 4.74: CocoGreek

Στην κλάση CocoGreek περιέχεται ένας χάρτης μετάφραση που αντιστοιχίζει τις 80 ετικέτες του dataset COCO στα ελληνικά. Ο χάρτης χρησιμοποιείται για το TextToSpeech και τις ετικέτες πάνω από τα bounding boxes.

4.8.5 MainViewModel

```
class MainViewModel : ViewModel() {
    var currentPresetName: String = "CUSTOM"
    2 Usages
    var bottomSheetWasExpanded: Boolean = false
}
```

Εικόνα 4.75: MainViewModel

Η κλάση MainViewModel κληρονομεί από ViewModel και διατηρεί δύο τιμές που πρέπει να επιβιώσουν. Οι τιμές δεν χάνονται όταν η Activity δημιουργείται ξανά.

4.8.6 Output0 και SegmentationResult

```
data class Output0(
    val x1: Float,
    val y1: Float,
    val x2: Float,
    val y2: Float,
    val cx: Float,
    val cy: Float,
    val w: Float,
    val h: Float,
    val cnf: Float,
    val cls: Int,
    val clsName: String,
    val maskWeight: List<Float>,
    val originalCnf: Float = cnf
)
```

Εικόνα 4.76: Output0

Η Output0 είναι ένα data class που αναπαριστά ένα ανιχνευμένο αντικείμενο. Περιέχει τις συντεταγμένες του bounding box, το κέντρο, τις διαστάσεις, την εμπιστοσύνη, τον δείκτη κλάσης, το όνομα, τα βάρη μάσκας, και την αρχική εμπιστοσύνη.

```
data class SegmentationResult(
    val box: Output0,
    val mask: Array<FloatArray>
) {
    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (javaClass != other?.javaClass) return false

        other as SegmentationResult

        return mask.contentDeepEquals(other.mask)
    }

    override fun hashCode(): Int {
        return mask.contentDeepHashCode()
    }
}
```

Εικόνα 4.77: SegmentationResult

Η SegmentationResult συνδυάζει ένα Output0 με την μάσκα κατάτμησης.

4.9 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε αναλυτικά η αρχιτεκτονική και η υλοποίηση της εφαρμογής. Αρχικά, έγινε η περιγραφή της εκκίνησης και αρχικοποίησης, η γραφική διεπαφή χρήστη και η αλυσίδα επεξεργασίας βίντεο από τα frames μέσω CameraX. Έπειτα αναλύθηκε η εκτέλεση του μοντέλου YOLOv26n-seg και ο υπολογισμός της απόστασης. Τέλος, εξετάστηκε ο αλγόριθμος εντοπισμού ακμών και απτικής ανατροφοδότησης, η φωνητική αλληλεπίδραση μέσω TTS και η διαχείριση ρυθμίσεων μέσω Jetpack Datastore. Το κεφάλαιο ολοκληρώθηκε με περιγραφή των βασικών κλάσεων. Στο επόμενο κεφάλαιο παρουσιάζονται οι δοκιμές και η αξιολόγηση της εφαρμογής.

Κεφάλαιο 5ο: Δοκιμές και Αξιολόγηση

5.1 Εισαγωγή

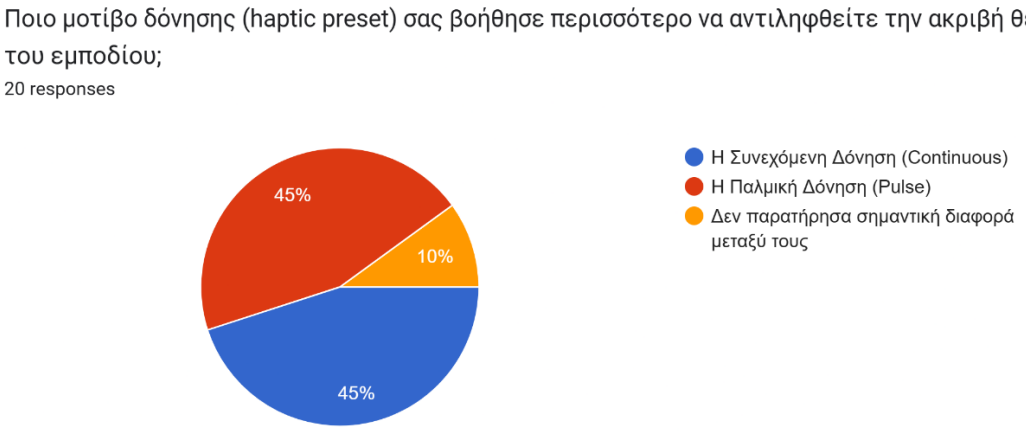
Στο παρόν κεφάλαιο αναλύεται η διαδικασία και τα αποτελέσματα των δοκιμών της εφαρμογής. Σκοπός των δοκιμών είναι η εύρεση του βαθμού στον οποίο η εφαρμογή ανταποκρίνεται στις απαιτήσεις που τέθηκαν κατά τον σχεδιασμό της. Επιπλέον, να διαπιστωθεί η χρηστικότητα της εφαρμογής ως υποστηρικτικό εργαλείο για άτομα με προβλήματα όρασης.

5.2 Μεθοδολογία και Σενάρια Χρήσης

Οι είκοσι συμμετέχοντες κλήθηκαν να δοκιμάσουν την εφαρμογή με κλειστά μάτια για να προσομοιωθεί η εμπειρία ενός χρήστη με οπτική αναπηρία. Η δοκιμή πραγματοποιήθηκε με συγκεκριμένα αντικείμενα (λάπτοπ, πληκτρολόγιο, άτομο). Στόχος ήταν τα αντικείμενα να εμπεριέχονται στα ογδόντα αντικείμενα-κλάσεις του COCO, στα οποία είναι εκπαιδευμένο το μοντέλο YOLO26-seg. Αξίζει να σημειωθεί ιδιαίτερα πως χρησιμοποιήθηκαν δύο διαφορετικά preset δόνησης για κάθε συμμετέχοντα, για να διερευνηθεί η αποτελεσματικότητά τους με βάση την προτίμηση των χρηστών. Οι χρήστες κλήθηκαν να αναγνωρίσουν το αντικείμενο μέσω αφής. Πρώτον, η αφή ενεργοποιεί την ηχητική ανακοίνωση και αναγνωρίζουν το αντικείμενο. Δεύτερον, μέσω της δόνησης να καταλάβουν το σχήμα και την σχετική θέση του στον χώρο.
Τέλος, χρησιμοποιήθηκε μια σταθερή τιμή μέγιστης ακτίνας δόνησης (maxRadius) σε όλα τα αντικείμενα. Στη συνέχεια ζητήθηκε η αξιολόγηση της.

5.3 Αξιολόγηση Τεχνικών Παραμέτρων

Τα δύο διαφορετικά μοτίβα δόνησης που χρησιμοποιήθηκαν είναι η Συνεχόμενη και η Παλμική δόνηση. Η συνεχόμενη δόνηση διαθέτει Διάστημα Παλμού 10 ms, ενώ η παλμική 100ms. Αυτή είναι η μόνη παράμετρος στην οποία διαφέρουν, καθώς η Διάρκεια είναι 50 ms, η Μέγιστη ακτίνα είναι 20 pixel, και το κατώφλι εμπιστοσύνης 0,5 (50%).



Εικόνα 5.1: Ερώτηση για το μοτίβο δόνησης

Τα αποτελέσματα της Εικόνας 5.1 δείχνουν ότι υπάρχει ισορροπημένη κατανομή μεταξύ των δύο μοτίβων δόνησης. Το 45% των συμμετεχόντων προτίμησε τη Συνεχόμενη δόνηση, και αντίστοιχα το άλλο 45% επέλεξε την Παλμική. Το υπόλοιπο 10% δεν παρατήρησε διαφορά μεταξύ τους. Όπως προαναφέρθηκε, τα δύο μοτίβα διαφέρουν μόνο ως προς το Διάστημα Παλμού. Η αλλαγή μόνο μιας παραμέτρου επηρεάζει σημαντικά την εμπειρία του χρήστη. Το εύρημα αυτό αναδεικνύει ότι η προτίμηση απτικής ανατροφοδότησης στην εφαρμογή είναι σε μεγάλο βαθμό υποκειμενική. Το παραπάνω δικαιολογεί την απόφαση να παρέχονται πολλαπλά προκαθορισμένα μοτίβα (presets) καθώς και η δυνατότητα προσαρμοσμένων ρυθμίσεων (custom) αντί για ένα σταθερό μοτίβο δόνησης.

Εικόνα 5.2: Ερώτηση για τη μέγιστη ακτίνα

Η τελευταία ερώτηση αφορούσε την αντιληπτή ζώνη δόνησης γύρω από τα αντικείμενα κατά την εξερεύνηση. Αυτή η ζώνη δόνησης αντιστοιχεί στην παράμετρο μέγιστη ακτίνα δόνησης (maxRadius) του αλγορίθμου. Με ρυθμισμένη την μέγιστη ακτίνα στα 20 pixel, το 85% των συμμετεχόντων αξιολόγησε τη ζώνη δόνησης ως κατάλληλη. Δηλαδή, η πλειονότητα μπορούσε να καταλάβει ξεκάθαρα τη θέση και το σχήμα των αντικειμένων. Το 10% χαρακτήρισε τη ζώνη δόνησης υπερβολικά μεγάλη, με αποτέλεσμα να δυσκολεύεται να εντοπίσει την ακριβή θέση. Το υπόλοιπο 5% βρήκε τη ζώνη δόνησης υπερβολικά μικρή.

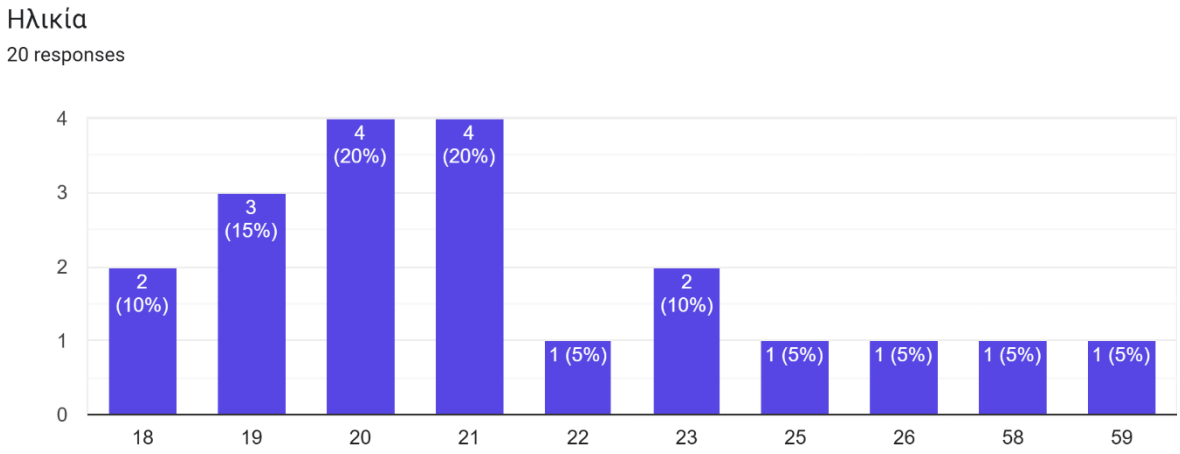
Τα παραπάνω επιβεβαιώνουν πως η προεπιλεγμένη τιμή μέγιστης ακτίνας 20 pixel αποτελεί ισορροπημένη επιλογή. Αντιθέτως, το υπόλοιπο 15% δηλώνει ότι η παράμετρος θέλει αυξομείωση. Άρα δικαιολογεί την απόφαση να είναι ρυθμιζόμενη από τις ρυθμίσεις.

5.4 Αποτελέσματα Εμπειρίας Χρήστη (UEQ-Short)

Για την αξιολόγηση της εφαρμογής χρησιμοποιήθηκε η σύντομη έκδοση του ερωτηματολογίου UEQ (User Experience Questionnaire) [33] μεταφρασμένη στα ελληνικά. Το συγκεκριμένο ερωτηματολόγιο παρέχει οχτώ ζεύγη αντωνυμιών, όπου σε κάθε ζεύγος υπάρχει η επιλογή 1 έως 7. Ο χρήστης αξιολογεί ποια λέξη ταιριάζει περισσότερο στην εμπειρία του.

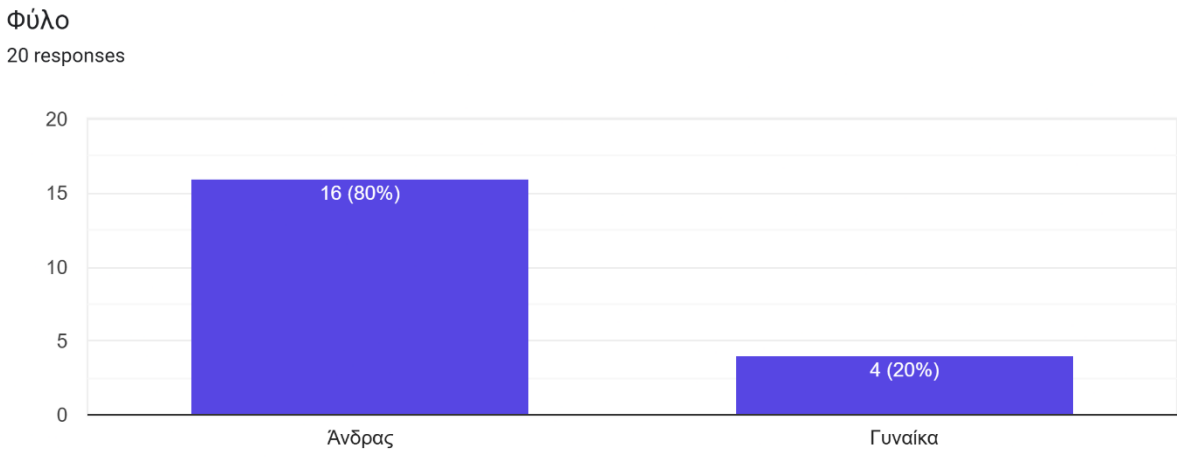
	1	2	3	4	5	6	7	
παρελκυστικό	○	○	○	○	○	○	○	υποστηρικτικό
περίπλοκο	○	○	○	○	○	○	○	εύκολο
ανεπαρκές	○	○	○	○	○	○	○	επαρκές
μπερδεμένο	○	○	○	○	○	○	○	σαφές
βαρετό	○	○	○	○	○	○	○	συναρπαστικό
αδιάφορο	○	○	○	○	○	○	○	ενδιαφέρον
συμβατικό	○	○	○	○	○	○	○	εφευρετικό
συνηθισμένο	○	○	○	○	○	○	○	πρωτοπόρο

Εικόνα 5.3: UEQ ερωτηματολόγιο

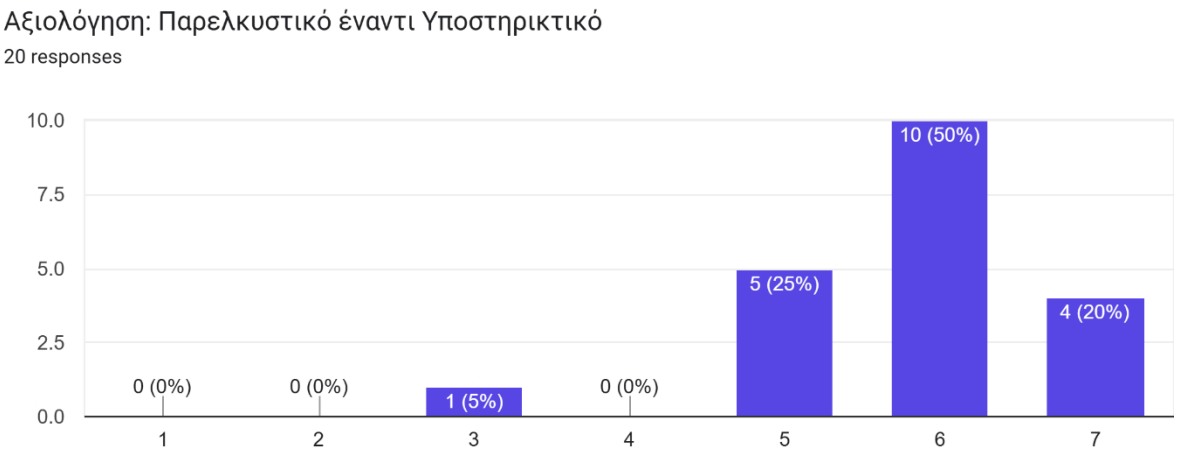


Εικόνα 5.4: Ηλικία συμμετεχόντων

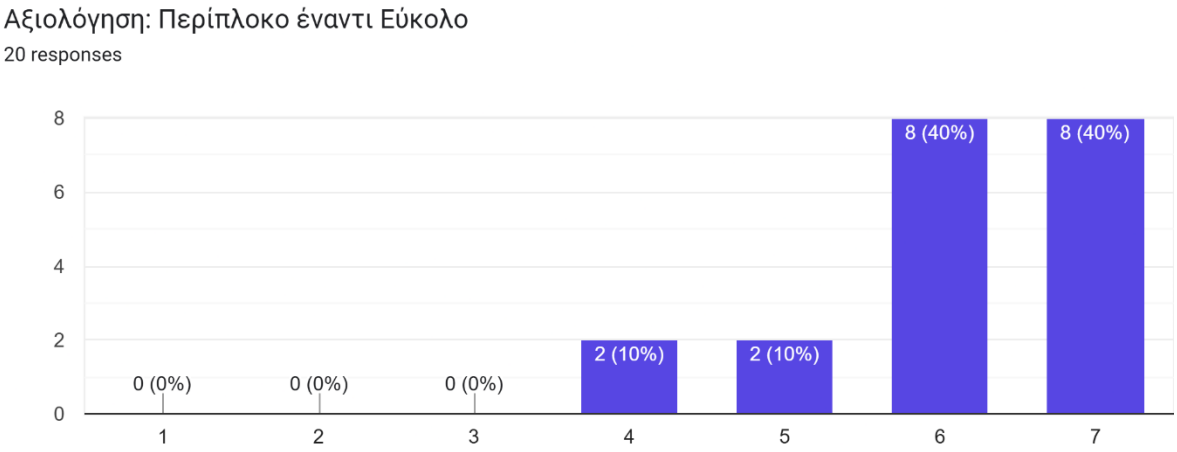
Οι συμμετέχοντες ήταν συνολικά 20 άτομα, εκ των οποίων το 80% ήταν άνδρες και το 20% γυναίκες. Οι ηλικίες κυμαίνονται κυρίως μεταξύ 19 και 21 ετών.



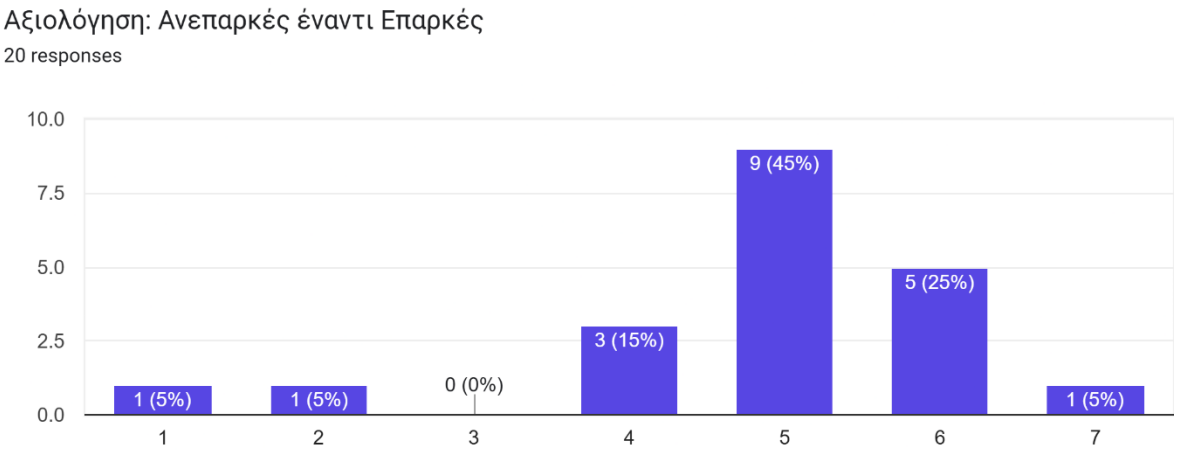
Εικόνα 5.5: Φύλο συμμετεχόντων



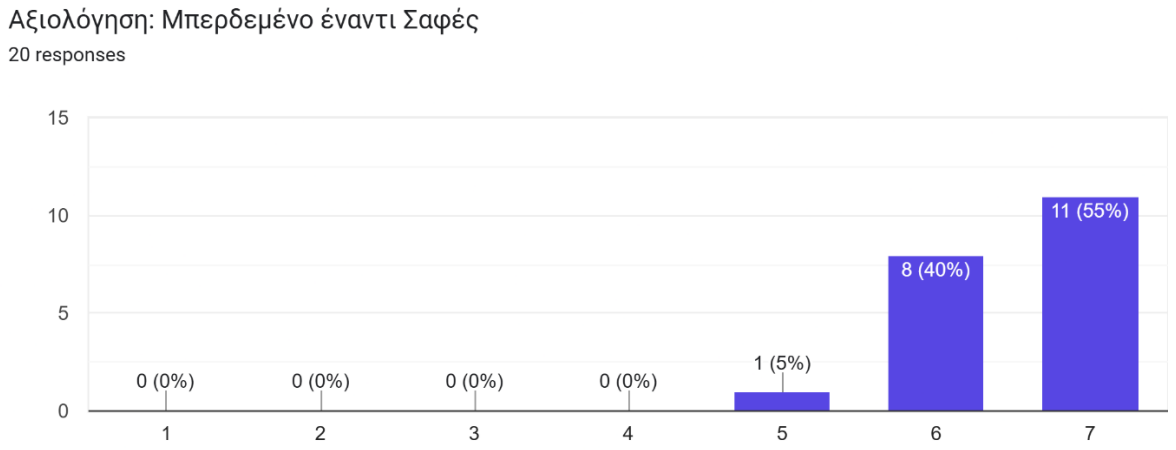
Εικόνα 5.6: Διάγραμμα με τιμή 1 για Παρελκυστικό και τιμή 7 για Υποστηρικτικό



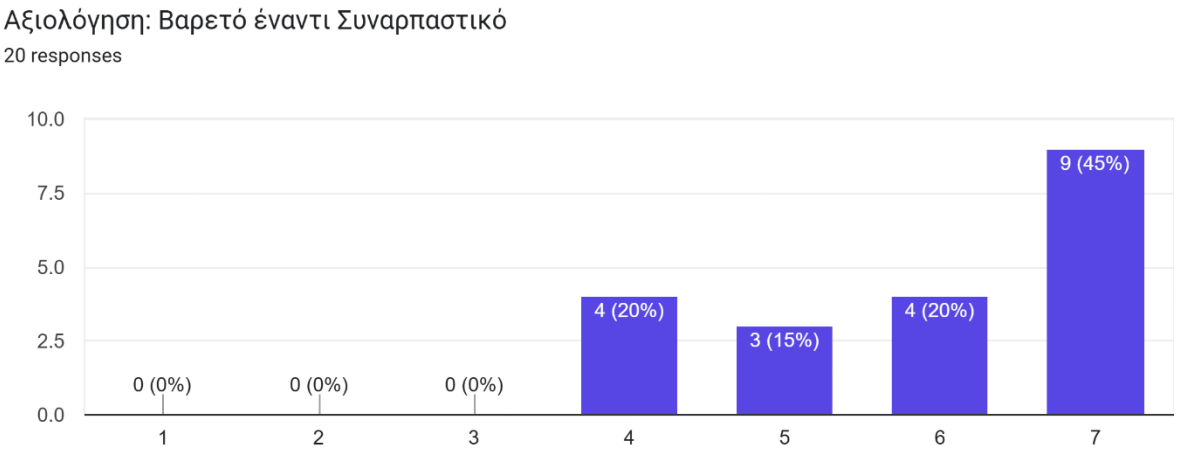
Εικόνα 5.7: Διάγραμμα με τιμή 1 για Περίπλοκο και τιμή 7 για Εύκολο



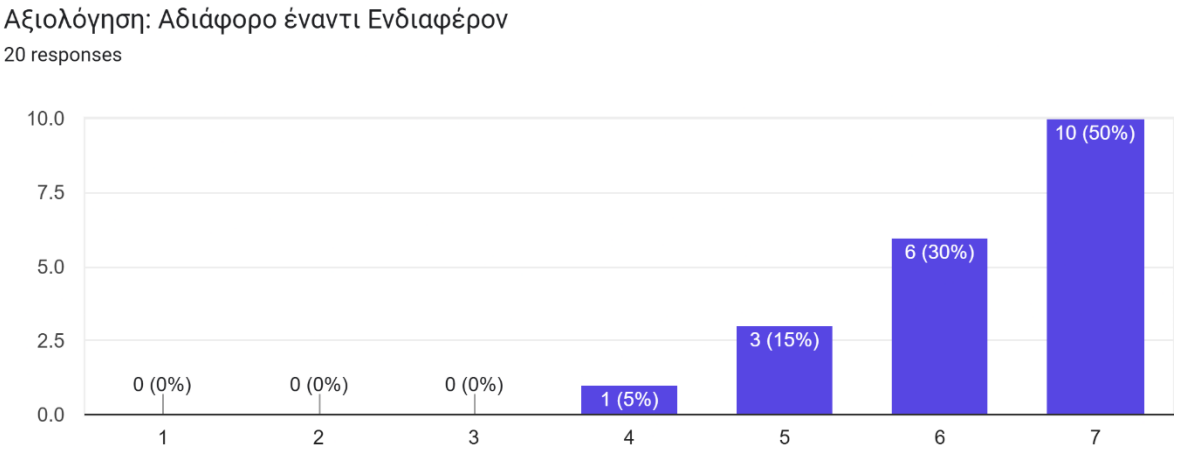
Εικόνα 5.8: Διάγραμμα για τιμή 1 για Ανεπαρκές και τιμή 7 για Επαρκές



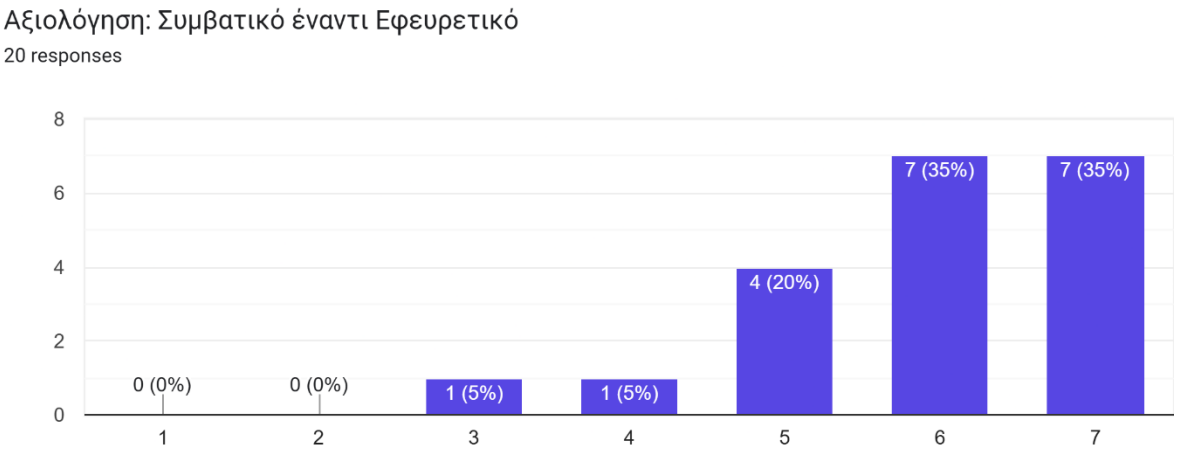
Εικόνα 5.9: Διάγραμμα με τιμή 1 για Μπερδεμένο και τιμή 7 για Σαφές



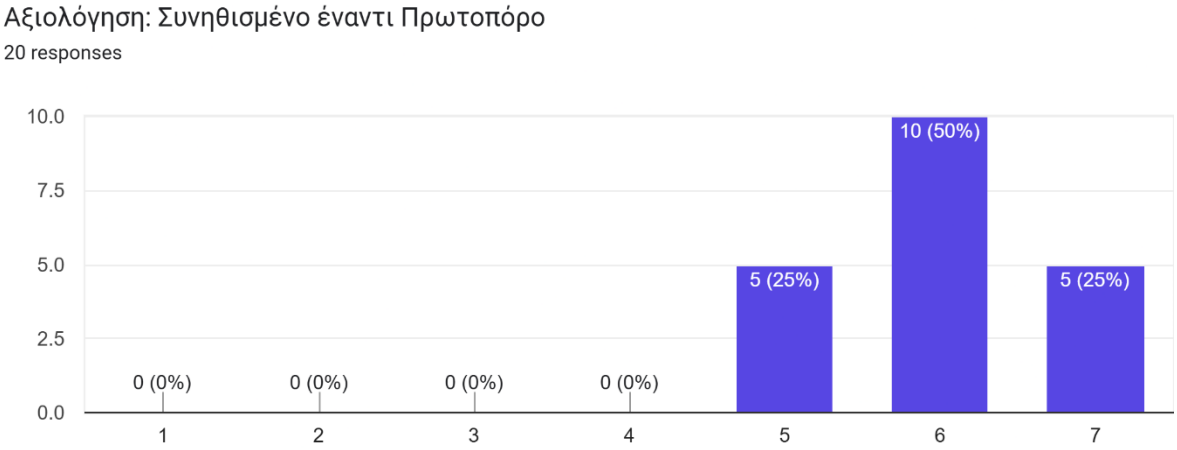
Εικόνα 5.10: Διάγραμμα με τιμή 1 για Βαρετό και τιμή 7 για Συναρπαστικό



Εικόνα 5.11: Διάγραμμα με τιμή 1 για Αδιάφορο και τιμή 7 για Ενδιαφέρον



Εικόνα 5.12: Διάγραμμα με τιμή 1 για Συμβατικό και τιμή 7 για Εφευρετικό



Εικόνα 5.13: Διάγραμμα με τιμή 1 για Συνηθισμένο και τιμή 7 για Πρωτοπόρο

Ζεύγος Αξιολόγησης	Μέσος Όρος
Παρελκυστικό → Υποστηρικτικό	**5,8 / 7**
Περίπλοκο → Εύκολο	**6,1 / 7**
Ανεπαρκές → Επαρκές	**4,85 / 7**
Μπερδεμένο → Σαφές	**6,5 / 7**
Βαρετό → Συναρπαστικό	**5,9 / 7**
Αδιάφορο → Ενδιαφέρον	**6,25 / 7**
Συμβατικό → Εφευρετικό	**5,9 / 7**
Συνηθισμένο → Πρωτοπόρο	**6,0 / 7**

Πίνακας 5.1: Μέσοι όροι αξιολόγησης

Ιδιαίτερο ενδιαφέρον παρουσιάζουν οι ακραίες τιμές μέσου όρου στον Πίνακα 5.1. Η υψηλότερη βαθμολογία καταγράφηκε στο ζεύγος «Μπερδεμένο → Σαφές» με μέσο όρο 6,5/7. Το παραπάνω υποδηλώνει ότι η πλειονότητα των συμμετεχόντων κατάλαβε με σαφήνεια τον σκοπό και τον τρόπο λειτουργίας της εφαρμογής. Η ένδειξη αυτή είναι σημαντική για εφαρμογές προσβασιμότητας, διότι δείχνει ότι ο χρήστης κατανοεί με ευκολία τον στόχο και την λειτουργία της εφαρμογής.

Αντιθέτως, η χαμηλότερη βαθμολογία εντοπίστηκε στο ζεύγος «Ανεπαρκές → Επαρκές» με μέσο όρο 4,85/7. Μετά από συζήτηση με κάποιους από τους συμμετέχοντες σχετικά με τα σχόλια και τους προβληματισμούς τους, διαπιστώθηκε ότι η κύρια αιτία είναι ο αριθμός των αντικειμένων που εντοπίζει η εφαρμογή. Το μοντέλο χρησιμοποιεί 80 κατηγορίες αντικειμένων του συνόλου δεδομένων COCO, με αποτέλεσμα πολλά αντικείμενα με τα οποία αλληλεπιδρά ο χρήστης στην καθημερινότητά του να μην ανιχνεύονται. Αυτό αναδεικνύει την ανάγκη για μελλοντική επέκταση του μοντέλου σε ευρύτερο σύνολο αντικειμένων.

5.5 Επίλογος

Συνοψίζοντας, στο παρόν κεφάλαιο παρουσιάστηκαν οι δοκιμές και τα αποτελέσματα της αξιολόγησης της εφαρμογής. Με το ερωτηματολόγιο UEQ (short version), η εφαρμογή αξιολογήθηκε με υψηλή βαθμολογία στη σαφήνεια λειτουργίας, ενώ η χαμηλότερη βαθμολογία αναδεικνύει την ανάγκη επέκτασης των αντικειμένων που εντοπίζει. Επίσης, αξίζει να σημειωθεί ότι οι επιπλέον ερωτήσεις επιβεβαίωσαν την αξία της παροχής ρυθμιζόμενων παραμέτρων στον χρήστη. Τα παραπάνω ευρήματα αποτελούν τη βάση για τα συμπεράσματα και τις μελλοντικές βελτιώσεις που παρουσιάζονται στο επόμενο κεφάλαιο.

Κεφάλαιο 6ο: Συμπεράσματα και Μελλοντικές Επεκτάσεις

6.1 Συμπεράσματα

Στην παρούσα Δ.Ε. παρουσιάστηκε ο σχεδιασμός, η υλοποίηση και η αξιολόγηση μιας εφαρμογής Android με σκοπό την υποστήριξη ατόμων με προβλήματα όρασης. Η εφαρμογή συνδυάζει ανίχνευση αντικειμένων σε πραγματικό χρόνο μέσω του μοντέλου YOLOv26n-seg, απτική αλληλεπίδραση μέσω δόνησης και φωνητική αλληλεπίδραση μέσω TTS και αναγνώριση ομιλίας. Όλα τα παραπάνω εκτελούνται τοπικά, με εξαίρεση την αναγνώριση φωνής.

Επιπλέον, αξίζει να σημειωθεί ότι χρησιμοποιήθηκε τμηματοποίηση στιγμιότυπου, αντί απλής ανίχνευσης αντικειμένων. Οι μάσκες που δημιουργεί η τμηματοποίηση είναι αυτές που επέτρεψαν τον σχεδιασμό αλγορίθμου εντοπισμού ακμών, ο οποίος καθορίζει πότε ενεργοποιείται η δόνηση.

Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα αποτελέσματα της αξιολόγησης, τόσο μέσω του UEQ ερωτηματολογίου, όσο και στις δύο ερωτήσεις σχετικά με τις τεχνικές παραμέτρους. Αρχικά, οι μέσοι όροι ζευγών κυμαίνονται μεταξύ 5,8 και 6,5 στα 7. Εξαίρεση αποτελεί η χαμηλότερη βαθμολογία μέσου όρου (4,85/7) στο ζεύγος «Ανεπαρκές → Επαρκές», η οποία οφείλεται κατά κύριο λόγο στα περιορισμένα αντικείμενα που ανιχνεύονται, καθώς το μοντέλο είναι εκπαιδευμένο μόνο στα βασικά 80 αντικείμενα του COCO. Αυτός είναι και ο σημαντικότερος περιοριστικός παράγοντας της εφαρμογής. Η υψηλότερη βαθμολογία μέσου όρου (6,5/7) στο ζεύγος «Μπερδεμένο → Σαφές» επιβεβαιώνει ότι η εφαρμογή γίνεται κατανοητή στους περισσότερους.

Επιπροσθέτως, τα αποτελέσματα σχετικά με τις προτιμήσεις μοτίβων δόνησης υποδεικνύουν ότι η επιλογή είναι υποκειμενική (45% Συνεχόμενη, 45% Παλμική), γεγονός που επιβεβαιώνει την αξία της παροχής προσαρμοσμένων ρυθμίσεων στον χρήστη. Τέλος, το 85% των συμμετεχόντων αξιολόγησε την προεπιλεγμένη τιμή μέγιστης ακτίνας (maxRadius=20 pixel) ως κατάλληλη.

6.2 Μελλοντικές Βελτιώσεις

Η παράμετρος μέγιστης ακτίνας δόνησης (maxRadius) ορίζεται σε pixel και η ερμηνεία της εξαρτάται από το μέγεθος που καταλαμβάνει το αντικείμενο στην οθόνη, το οποίο επηρεάζεται τόσο από την απόσταση του αντικειμένου όσο και από το φυσικό του μέγεθος. Μια σταθερή τιμή μέγιστης ακτίνας συμπεριφέρεται διαφορετικά ανάλογα με το αν το αντικείμενο βρίσκεται κοντά ή μακριά από τη συσκευή. Για αντικείμενα που καταλαμβάνουν μικρό τμήμα της οθόνης, η ζώνη δόνησης μπορεί να καλύπτει δυσανάλογα μεγάλο ποσοστό, ενώ για αντικείμενα που καταλαμβάνουν μεγάλο τμήμα, η ζώνη ενδέχεται να είναι ανεπαρκής για την αντίληψη του περιγράμματος.

Επομένως, προτείνεται ως μελλοντική βελτίωση η προσθήκη δυναμικής αυξομείωσης της μέγιστης ακτίνας, ανάλογα με το μέγεθος του bounding box κάθε αντικειμένου στην οθόνη.

Επιπλέον, η αξιολόγηση ανέδειξε ότι ο αριθμός των αντικειμένων που ανιχνεύεται είναι μικρός. Η επέκταση του μοντέλου σε μεγαλύτερο εύρος αντικειμένων θα βελτίωνε κατά πολύ την εφαρμογή.

Επιπροσθέτως, η αναγνώριση φωνής αποτελεί το μοναδικό τμήμα της εφαρμογής που απαιτεί σύνδεση στο διαδίκτυο, λόγω της επικοινωνίας του SpeechRecognizer API με τους servers της Google[32]. Η

εύρεση και αντικατάσταση του με ένα τοπικό μοντέλο αναγνώρισης ομιλίας θα καθιστούσε την εφαρμογή πλήρως λειτουργική εκτός σύνδεσης.

Τέλος, η εκτίμηση απόστασης βασίζεται σε παραδοχές που δεν ισχύουν πάντα στην πράξη, όπως το ύψος κράτησης της συσκευής που θεωρείται σταθερό στα 1,2 μέτρα. Η βελτίωση της ακρίβειας εκτίμησης απόστασης αποτελεί πεδίο για μελλοντική έρευνα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep Learning for Computer Vision: a Brief Review,” *Computational Intelligence and Neuroscience*, vol. 2018, no. 1, pp. 1–13, Feb. 2018, doi: https://doi.org/10.1155/2018/7068349.

[2] R. Archana and P. S. E. Jeevaraj, “Deep learning models for digital image processing: a review,” Artif Intell Rev, vol. 57, no. 1, Jan. 2024, doi: 10.1007/s10462-023-10631-z.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, pp. 779–788, Jun. 2016. doi: 10.1109/cvpr.2016.91.

[4] R. Khanam and M. Hussain, “YOLOv11: An Overview of the Key Architectural Enhancements,” 2024, arXiv. doi: 10.48550/ARXIV.2410.17725.

[5] E. Mohamed, A. Shaker, A. El-Sallab, and M. Hadhoud, “INSTA-YOLO: Real-Time Instance Segmentation,” 2021, arXiv. doi: 10.48550/ARXIV.2102.06777.

[6] C. Jiang, E. Kuang, and M. Fan, “How Can Haptic Feedback Assist People with Blind and Low Vision (BLV): A Systematic Literature Review,” *ACM Transactions on Accessible Computing*, Jan. 2025, doi: https://doi.org/10.1145/3711931.

[7] G. Voutsakelis, I. Dimkaros, N. Tzimos, G. Kokkonis, and S. Kontogiannis, “Development and Evaluation of a Tool for Blind Users Utilizing AI Object Detection and Haptic Feedback,” Machines, vol. 13, no. 5, p. 398, May 2025, doi: 10.3390/machines13050398.

[8] Google, "Meet Android Studio," *Android Developers*. [Online]. Available: https://developer.android.com/studio/intro.

[9] JetBrains, "Kotlin Programming Language," *kotlinlang.org*. [Online]. Available: https://kotlinlang.org.

[10] JetBrains, "Coroutines guide," *Kotlin Documentation*. [Online]. Available: https://kotlinlang.org/docs/coroutines-guide.html.

[11] Google, "DataStore," *Android Developers*. [Online]. Available: https://developer.android.com/topic/libraries/architecture/datastore.

[12] Google, "CameraX overview," *Android Developers*. [Online]. Available: https://developer.android.com/media/camera/camerax.

[13] Ultralytics, "YOLO26," *Ultralytics Documentation*, 2026. [Online]. Available: https://docs.ultralytics.com/models/yolo26.

[14] R. Sapkota, R. H. Cheppally, A. Sharda, and M. Karkee, "YOLO26: Key Architectural Enhancements and Performance Benchmarking for Real-Time Object Detection," *arXiv:2509.25164v5*, Mar. 2026. doi: 10.48550/arXiv.2509.25164.

[15] F. Cheraghpour Samavati and A. Rahimi Ghasem Abadi, “Assistive Technologies for the Visually Impaired: A Review,” *Cureus Journal of Computer Science*, Oct. 2025, doi: https://doi.org/10.7759/s44389-025-06891-1.

[16] A. Lavric, C. Beguni, E. Zadobrischi, A.-M. Căilean, and S.-A. Avătămăniței, “A Comprehensive Survey on Emerging Assistive Technologies for Visually Impaired Persons: Lighting the Path with Visible Light Communications and Artificial Intelligence Innovations,” Sensors, vol. 24, no. 15, p. 4834, Jul. 2024, doi: 10.3390/s24154834.

[17] M. Satyanarayanan, Q. Dong, J. Xu, and P. Pillai, "Towards Mobile AI That Is Accurate and Fast," in *Proc. IEEE 7th Int. Conf. on Cognitive Machine Intelligence (CogMI)*, 2025, pp. 425–432, doi: 10.1109/CogMI67134.2025.00056.

[18] A. Tyagi, "Deploying Real-Time Object Detection and Obstacle Avoidance For Smart Mobility Using Edge-AI," *TechRxiv*, Apr. 2025, doi: 10.36227/techrxiv.174440189.93590848/v1.

[19] World Health Organization, "World report on vision," WHO, 2019. [Online]. Available: https://www.who.int/publications/i/item/world-report-on-vision.

[20] R. Sun et al., “Training Indoor and Scene-Specific Semantic Segmentation Models to Assist Blind and Low Vision Users in Activities of Daily Living,” IEEE Open J. Eng. Med. Biol., vol. 6, pp. 533–539, 2025, doi: 10.1109/ojemb.2025.3607816.

[21] I. Dede and A. Gumus, “Vis-Assist: computer vision and haptic feedback-based wearable assistive device for visually impaired,” J Multimodal User Interfaces, vol. 19, no. 2, pp. 217–234, May 2025, doi: 10.1007/s12193-025-00452-5.

[22] Voxel51, "Why are image segmentation maps superior to bounding boxes?," Voxel51 Blog. [Online]. Available: https://voxel51.com/blog/why-are-image-segmentation-maps-superior-to-bounding-boxes.

[23] A. Boussihmed, K. El Makkaoui, I. Ouahbi, Y. Maleh, and A. Chetouani, “A TinyML model for sidewalk obstacle detection: aiding the blind and visually impaired people,” Multimed Tools Appl, vol. 84, no. 22, pp. 25837–25864, Sep. 2024, doi: 10.1007/s11042-024-20070-9.

[24] A. Kumar et al., “Navigating beyond sight: a real-time 3D audio-enhanced object detection system for empowering visually impaired spatial awareness,” SIViP, vol. 19, no. 12, Sep. 2025, doi: 10.1007/s11760-025-04614-6.

[25] R. Shrestha and N. Amatya, “A Gesture-Based Mobile Interface for Enhanced Image Accessibility for Visually Impaired,” in 2025 3rd International Conference on Foundation and Large Language Models (FLLM). IEEE, pp. 888–894, Nov. 25, 2025. doi: 10.1109/fllm67465.2025.11391056.

[26] B. Leporini, M. Rosellini, and N. Forgione, “Haptic Wearable System to Assist Visually-Impaired People in Obstacle Detection,” in Proceedings of the 15th International Conference on PErvasive Technologies Related to Assistive Environments. ACM, pp. 269–272, Jun. 29, 2022. doi: 10.1145/3529190.3529217.

[27] X. Chen, Y. Gu, C. García-Cena, and P. Sareh, “WanderPal: An intelligent wearable to enhance mobility for people with severe visual impairments,” Device, vol. 3, no. 10, p. 100924, Oct. 2025, doi: 10.1016/j.device.2025.100924.

[28] M. Arsalan, A. Imran, A. Arif, K. Khan, and M. R. Siddiqi, “Deep Learning-Based Assistive Robotic Grasping System for Visually Impaired Individuals,” IEEE Access, vol. 14, pp. 60205–60228, 2026, doi: 10.1109/access.2026.3684195.

[29] Android Developers, "Create custom haptic effects," *Android Developers*, Feb. 2026. [Online]. Available: https://developer.android.com/develop/ui/views/haptics/custom-haptic-effects. [Accessed: May 2026].

[30] Android Developers, "Android haptics API reference," *Android Developers*, Feb. 2026. [Online]. Available: https://developer.android.com/develop/ui/views/haptics/haptics-apis. [Accessed: May 2026].

[31] Android Developers, "TextToSpeech," *Android Developers API Reference*. [Online]. Available: https://developer.android.com/reference/android/speech/tts/TextToSpeech. [Accessed: May 2026].

[32] Android Developers, "SpeechRecognizer," *Android Developers API Reference*. [Online]. Available: https://developer.android.com/reference/android/speech/SpeechRecognizer. [Accessed: May 2026].

[33] M. Schrepp, A. Hinderks, and J. Thomaschewski, “Design and Evaluation of a Short Version of the User Experience Questionnaire (UEQ-S),” IJIMAI, vol. 4, no. 6, pp. 103–108, Dec. 2017, doi: 10.9781/ijimai.2017.09.001.

[34] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *J Big Data*, vol. 8, no. 1, p. 53, Mar. 2021, doi: 10.1186/s40537-021-00444-8.