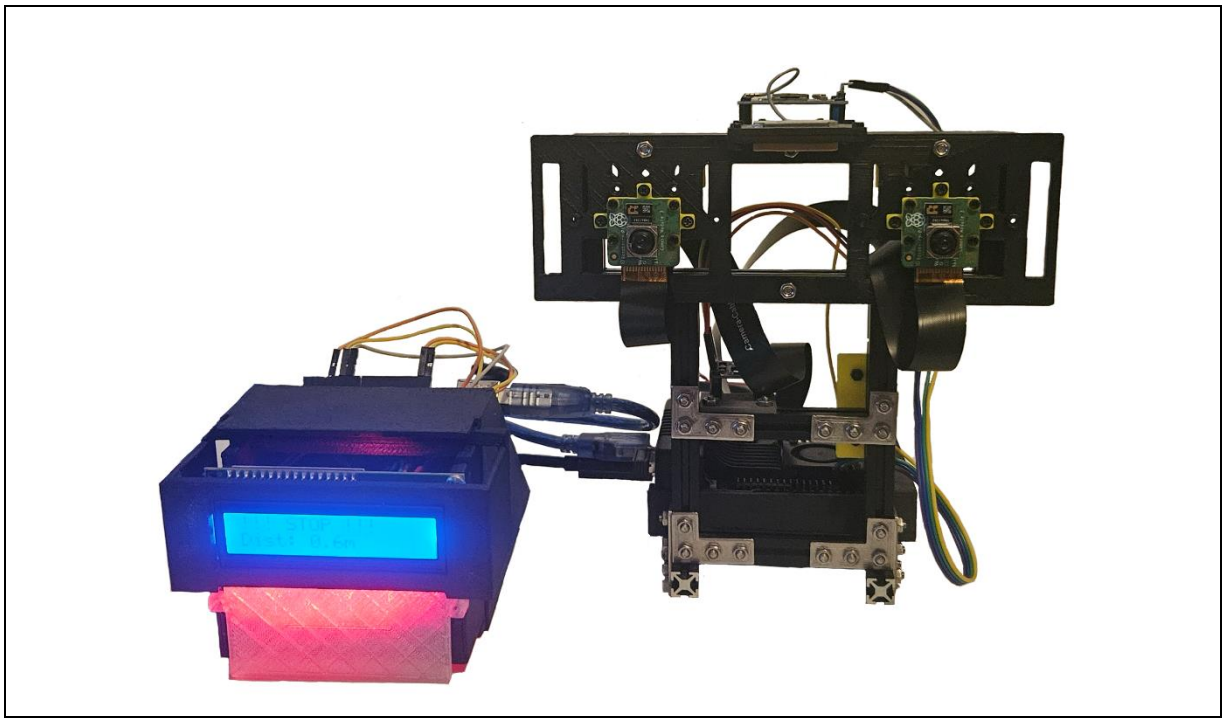
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη συστήματος αναγνώρισης πινακίδων του κώδικα οδικής κυκλοφορίας με χρήση κάμερας και raspberry pi»



Του φοιτητή
Χριστοφορίδη Δημητρίου
Αρ. Μητρώου: 5161614

Επιβλέπων
Δρ. Τσιακμάκης Κυριάκος

Σεπτέμβριος 2025

Τίτλος Π.Ε. Ανάπτυξη συστήματος αναγνώρισης πινακίδων του κώδικα οδικής κυκλοφορίας με χρήση κάμερας και raspberry pi

Κωδικός Π.Ε. 24159

Ονοματεπώνυμο φοιτητής Χριστοφορίδης Δημήτριος

Ονοματεπώνυμο εισηγητή Τσιακμάκης Κυριάκος

Ημερομηνία ανάληψης Π.Ε. 19/3/2024

Ημερομηνία περάτωσης Π.Ε. 1/9/25

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.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Στην οικογένεια μου»

Πρόλογος

Η παρούσα πτυχιακή εργασία αναπτύχθηκε από τον φοιτητή Χριστοφορίδη Δημήτριο, υπό την επίβλεψη του Δρ. Τσιακμάκη Κυριάκου, και εστιάζει στον σχεδιασμό και την ανάπτυξη ενός ολοκληρωμένου συστήματος αναγνώρισης πινακίδων του Κ.Ο.Κ., υπολογισμού της απόστασής τους και ενημέρωσης του οδηγού. Η ανάπτυξη του συστήματος προσέφερε τη δυνατότητα πρακτικής εφαρμογής των θεωρητικών γνώσεων, ενώ αποτέλεσε και ευκαιρία εμβάθυνσης και κατανόησης της αρχιτεκτονικής και των πρακτικών εφαρμογών σε ενσωματωμένα συστήματα που λειτουργούν σε πραγματικό χρόνο.

Περίληψη

Η παρούσα εργασία εξετάζει την ανάπτυξη ενός συστήματος υποβοήθησης οδηγού που αναγνωρίζει πινακίδες ορίου ταχύτητας, εκτιμά την απόστασή τους, εκτιμά την ταχύτητα του οχήματος και ενημερώνει άμεσα τον χρήστη για το ισχύον όριο και τυχόν υπέρβαση. Το προτεινόμενο σύστημα βασίζεται σε χαμηλού κόστους υπολογιστική πλατφόρμα και αξιοποιεί τεχνικές υπολογιστικής όρασης για την ανίχνευση και ταξινόμηση των σημάτων, στερεοσκοπική επεξεργασία για τον υπολογισμό της απόστασης και αισθητήρες κίνησης και θέσης για την εκτίμηση της ταχύτητας. Η αρχιτεκτονική είναι αρθρωτή με βάση την λειτουργία σε πραγματικό χρόνο. Αποτελείται από διακριτές ενότητες για λήψη εικόνων, εντοπισμό και παρακολούθηση περιοχών ενδιαφέροντος, υπολογισμό απόστασης εκτίμηση ταχύτητας και παρουσίαση πληροφοριών στον οδηγό. Επιπλέον ενσωματώνεται καταγραφή δεδομένων για αξιολόγηση απόδοσης και ανάλυση λαθών. Πειραματικές δοκιμές σε πραγματικές συνθήκες κίνησης έδειξαν ότι το σύστημα μπορεί να λειτουργεί με χαμηλή υστέρηση και ικανοποιητική αξιοπιστία, λαμβάνοντας υπόψιν τους περιορισμούς του συστήματος, διαφοροποιήσεις του φωτισμού, των καιρικών συνθήκων καθώς και του μεγέθους των στόχων σε μεγάλες αποστάσεις. Η εργασία συμβάλλει με ένα ολοκληρωμένο σχεδιασμό που μπορεί να αναπαραχθεί, ισορροπεί ακρίβεια και υπολογιστικό κόστος, ενώ προτείνει κατευθύνσεις για μελλοντική βελτίωση.

«Development of a Traffic Sign Recognition System Using a Stereo Camera and a Raspberry Pi»

Christoforidis Dimitrios

Abstract

This thesis presents a driver-assistance system that recognizes speed limit signs, estimates their distance, and delivers timely feedback to the driver regarding the applicable limit and any speeding events. The proposed solution targets an embedded, cost-effective platform and combines computer vision for sign detection and classification, stereo processing for distance estimation, and motion and position sensors for vehicle-speed estimation. The architecture is modular and real-time oriented, separating image acquisition, region detection/tracking, distance computation, speed estimation, and driver feedback. A data-logging component is included to support performance evaluation and error analysis. Field tests under real driving conditions indicate that the system can operate with low latency and satisfactory reliability, while acknowledging challenges related to the limitations of the hardware, lighting variability, weather effects, and the small apparent size of distant signs. The contribution lies in an end-to-end, reproducible design that balances accuracy with computational cost, together with practical insights on tuning for embedded deployments. Future work includes model optimization, improved robustness in adverse conditions, and expanding the set of recognized sign categories.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τη μητέρα μου για τη διαρκή στήριξη καθ' όλη τη διάρκεια της ανάπτυξης της εργασίας, καθώς και τους φίλους και συναδέλφους μου για την πολύτιμη βοήθειά τους στην οργάνωση και εκτέλεση ασφαλών δοκιμών.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
ΕΛΣΤΑΤ	Ελληνική Στατιστική Αρχή
AIS3+	Abbreviated Injury Scale 3+
ERSO	European Road Safety Observatory
ADAS	Advanced Driver Assistance Systems
ISA	Intelligent Speed Assistance
GPS	Global Positioning System
NCAP	New Car Assessment Programme
ΕΕ	Ευρωπαϊκή Ένωση
ΚΟΚ	Κώδικας Οδικής Κυκλοφορίας
IMU	Inertial Measurement Unit
HSI	Hue–Saturation–Intensity
SVM	Support Vector Machine
RBF	Radial Basis Function
GTSRB	German Traffic Sign Recognition Benchmark
CNN	Convolutional Neural Network
RGB	Red–Green–Blue
FPS	Frames Per Second
GPU	Graphics Processing Unit
MB	Megabyte
SSD	Solid Shot Detection
mAP	mean Average Precision
YOLO	You Only Look Once
R-CNN	Region-based Convolutional Neural Network
ROI	Region of Interest
GSR	General Safety Regulation
OCR	Optical Character Recognition
OEM	Original Equipment Manufacturer
TOPS	Tera Operations Per Second
DNN	Deep Neural Network
GB	Gigabyte

ms	milliseconds
SoCs	Systems-on-Chips
REM-based	Road Experience Management–based
AVV	Autonomous Vehicle(s)
RSS	Responsibility-Sensitive Safety
PB	Petabyte
CPU	Central Processing Unit
LPDDR4	Low-Power DDR4
TPU	Tensor Processing Unit
FSD	Full Self-Driving
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
OS	Operating System
I²C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver–Transmitter
PWM	Pulse-Width Modulation
MIPI CSI	Mobile Industry Processor Interface – Camera Serial Interface
Mm	millimeter(s)
FoV	Field of View
HDR	High Dynamic Range
VTG	Course Over Ground and Ground Speed
GGA	GPS Fix Data
SDA	Serial Data
SCL	Serial Clock
UPS	Uninterruptible Power Supply
ASA CF	Acrylonitrile Styrene Acrylate – Carbon Fiber
CSPNet	Cross Stage Partial Network
VRAM	Video Random Access Memory
bbox	bounding box
PLA	Polylactic Acid
PETG	Polyethylene Terephthalate Glycol
UI	User Interface

RAM	Random-Access Memory
SBC	Single-Board Computer
CAN	Controller Area Network

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Τα τροχαία ατυχήματα παραμένουν ένα από τα σοβαρότερα προβλήματα δημόσιας ασφάλειας στην Ελλάδα, με εκατοντάδες νεκρούς και χιλιάδες τραυματίες να καταγράφονται κάθε χρόνο. Παρά την πρόοδο που έχει σημειωθεί τις τελευταίες δεκαετίες, η απώλεια ανθρώπινων ζωών, καθώς και οι οικονομικές και κοινωνικές συνέπειες, εξακολουθούν να είναι ιδιαίτερα σημαντικές.

Σύμφωνα με την τελευταία έρευνα της ΕΛΣΤΑΤ[1], την περίοδο 2014–2023 οι ετήσιοι θάνατοι από τροχαία στην Ελλάδα μειώθηκαν κατά 17,5%, από 739 σε 610. Οι σοβαροί τραυματισμοί, όπως ορίζονται με βάση το επίπεδο σοβαρότητας AIS3+, κυμάνθηκαν κατά μέσο όρο γύρω στις 650 περιπτώσεις ετησίως. Παρά την πτωτική τάση, η επίτευξη των φιλόδοξων στόχων της Ευρωπαϊκής Ένωσης για μείωση των θυμάτων κατά 50% παραμένει πρόκληση.

Η κυριότερη αιτία των θανατηφόρων ατυχημάτων το 2023 ήταν η οδήγηση με ταχύτητα μεγαλύτερη από το επιτρεπτό όριο. Η παράβαση αυτή ευθύνεται για το 9,9% των ατυχημάτων (1.042 από τα 10.553) και για το 19,8% των θανάτων (128 θύματα). Σύμφωνα με την ERSO[2], το 22,3% των οδηγών υπερβαίνει το όριο ταχύτητας στους αυτοκινητόδρομους, το 15,6% στις επαρχιακές οδούς και το 42,2% στις αστικές οδούς. Επιπλέον, στην Ελλάδα καταγράφονται 19,3 κλήσεις για παραβίαση ορίου ταχύτητας ανά 1.000 κατοίκους, όταν ο μέσος όρος στην ΕΕ φτάνει τις 139,7.

Η σημαντικότερη αιτία τροχαίων ατυχημάτων για το 2023 ήταν η μη διακοπή πορείας πριν από σήμα STOP, με 1.743 περιστατικά (16,5% των ατυχημάτων). Από αυτή την παράβαση προκλήθηκαν 30 θανατηφόρα ατυχήματα.

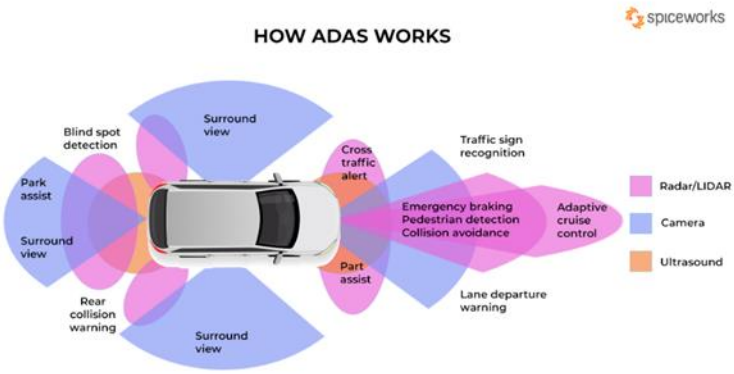
Η αντιμετώπιση αυτών των παραγόντων κινδύνου απαιτεί πολυδιάστατες λύσεις, στις οποίες η τεχνολογία παίζει κεντρικό ρόλο. Τα προηγμένα συστήματα υποβοήθησης οδηγού (ADAS) έχουν σχεδιαστεί για να προλαμβάνουν ή να μετριάζουν τις συνέπειες των ατυχημάτων. Ωστόσο, σημαντικό ζήτημα αποτελεί το γεγονός ότι τα συστήματα αυτά περιλαμβάνονται κυρίως σε καινούρια οχήματα τελευταίας τεχνολογίας. Στην Ευρωπαϊκή Ένωση, η μέση ηλικία των ενεργών οχημάτων είναι 12 έτη, ενώ στην Ελλάδα φτάνει τα 17. Αυτό σημαίνει ότι η πλειονότητα των αυτοκινήτων που κυκλοφορούν στους ελληνικούς δρόμους δεν είναι εξοπλισμένα με συστήματα υποβοήθησης του οδηγού.[3]

1.2 Τεχνολογικές λύσεις του προβλήματος

1.2.1.1 Συστήματα ADAS

Τα προηγμένα συστήματα υποβοήθησης οδηγού χρησιμοποιούνται σε οχήματα για την υποστήριξη των οδηγών. Χρησιμοποιούν πολλαπλές εισόδους δεδομένων από μια ποικιλία καμερών, ραντάρ και άλλων αισθητήρων για την ανίχνευση και την προειδοποίηση των οδηγών για επικίνδυνες καταστάσεις. Μπορούν ακόμη και να λάβουν μέτρα έκτακτης ανάγκης για την προστασία των επιβατών. Οι πιο προηγμένες λειτουργίες ADAS αυτοματοποιούν ορισμένα στοιχεία της οδηγικής εμπειρίας. Μεταξύ των πιο δημοφιλών τεχνολογιών ADAS σήμερα είναι:[4]

- Αυτόματο φρενάρισμα έκτακτης ανάγκης
- Προσαρμόσιμο cruise control
- Ειδοποιήσεις οδηγού
- Ανίχνευση τυφλού σημείου
- Προειδοποίηση απόκλισης από τη λωρίδα κυκλοφορίας
- Υποβοήθηση στάθμευσης



Εικόνα 1.1: Σύστημα ADAS

[https://zd-brightspot.s3.us-east-1.amazonaws.com/wpcontent/uploads/2022/06/15105922/ADAS-Working.png]

1.2.1.2 Συστήματα ISA

Το ISA χρησιμοποιεί μια βιντεοκάμερα αναγνώρισης σημάτων ταχύτητας ή/και δεδομένα ορίου ταχύτητας συνδεδεμένα με GPS για να ενημερώνει τους οδηγούς για το τρέχον όριο ταχύτητας και να τους προειδοποιεί εάν το υπερβαίνουν. Τα πιο αποτελεσματικά συστήματα περιορίζουν αυτόματα την ταχύτητα του οχήματος ανάλογα με τις ανάγκες, περιορίζοντας την ισχύ του κινητήρα. Οχήματα με αυτό το είδος συστήματος ISA εργοστασιακά τοποθετημένο κυκλοφορούν στην αγορά εδώ και αρκετά χρόνια, εν μέρει χάρη στην απόφαση του Euro NCAP να επιβραβεύει επιπλέον βαθμούς για οχήματα που περιλαμβάνουν ISA.

Η Ευρωπαϊκή Ένωση συμφώνησε το 2019 να καταστήσει υποχρεωτική μια έκδοση του ISA που μπορεί να παρακαμφθεί, μαζί με μια σειρά από άλλα μέτρα ασφαλείας οχημάτων, για τα νέα μοντέλα αυτοκινήτων που πωλούνται στην ΕΕ από τον Ιούλιο του 2022 και για όλα τα νέα αυτοκίνητα που πωλούνται από τον Ιούλιο του 2024. Ωστόσο, εάν οι αυτοκινητοβιομηχανίες τοποθετήσουν μόνο συστήματα που πληρούν τις ελάχιστες απαιτήσεις, η δυνατότητα πρόληψης θανάτων και τραυματισμών θα είναι περιορισμένη, σε σύγκριση με τα συστήματα που παρεμβαίνουν πραγματικά για να εμποδίσουν τον οδηγό να επιταχύνει πάνω από το όριο ταχύτητας.[5]

Το ISA είναι ένας συλλογικός όρος για διάφορα συστήματα:

- Το ανοιχτό ISA προειδοποιεί τον οδηγό (ορατά ή/και ακουστικά) ότι το όριο ταχύτητας ξεπερνιέται. Ο οδηγός αποφασίζει ο ίδιος αν θα επιβραδύνει ή όχι. Πρόκειται για ένα ενημερωτικό ή συμβουλευτικό σύστημα.
- Το ημι-ανοιχτό ISA αυξάνει την πίεση στο πεντάλ γκαζιού όταν ξεπεραστεί το όριο ταχύτητας (ο «ενεργός επιταχυντής»). Η διατήρηση της ίδιας ταχύτητας είναι δυνατή, αλλά λιγότερο άνετη λόγω της αντίθλιψης.
- Το κλειστό ISA περιορίζει αυτόματα την ταχύτητα εάν ξεπεραστεί το όριο ταχύτητας. Είναι εφικτό να γίνει υποχρεωτικό ή προαιρετικό. Στη δεύτερη περίπτωση, οι οδηγοί μπορούν να επιλέξουν να ενεργοποιήσουν ή να απενεργοποιήσουν το σύστημα. Τα διαθέσιμα συστήματα ISA βασίζονται σε σταθερά όρια ταχύτητας. Μπορεί επίσης να περιλαμβάνουν όρια ταχύτητας που εξαρτώνται από την τοποθεσία (συμβουλευτικά). Θα καθίσταται ολοένα και πιο δυνατό να συμπεριληφθούν δυναμικά όρια ταχύτητας που λαμβάνουν υπόψη τις πραγματικές συνθήκες σε μια συγκεκριμένη χρονική στιγμή.

Σε πείραμα στη Σουηδία δεν βρέθηκαν αρνητικές παρενέργειες στη χρήση του ISA. Παρ' όλ' αυτά, υπάρχουν ανησυχητικά ζητήματα με τη χρήση του. Αρχικά, οι οδηγοί προσπαθούν να αντισταθμίσουν την ταχύτητα σε σημεία του οδικού δικτύου που το ISA δεν είναι ενεργό. Παράλληλα, οι οδηγοί φαίνεται να έχουν μειωμένη προσοχή και υπερβολική αυτοπεποίθηση στο σύστημα, χωρίς να προσέχουν πιθανές αλλαγές στην κατάσταση του δρόμου, βασιζόμενοι κυρίως στο όριο του συστήματος [6].

1.3 Σκοπός της Εργασίας

Ο αρχικός σκοπός της εργασίας ήταν ο σχεδιασμός και η ανάπτυξη ενός συστήματος υποβοήθησης του οδηγού, ικανό να αναγνωρίζει τα όρια ταχύτητας του ΚΟΚ σε πραγματικό χρόνο, να υπολογίζει την ταχύτητα του οχήματος μέσω αισθητήρων (GPS , IMU) και να ενημερώνει τον οδηγό για το όριο ταχύτητας και την τυχόν παραβίαση του. Η ανάγκη για την δημιουργία ενός συστήματος σαν αυτό προέκυψε από την επιθυμία βελτίωσης της οδικής ασφάλειας, χωρίς εξάρτηση από εξωτερικά δεδομένα ή cloud based εφαρμογές.

Κατά τη διάρκεια της υλοποίησης το σύστημα εξελίχθηκε σε ένα γενικότερο πλαίσιο μηχανικής όρασης για embedded συσκευές, με έμφαση στην επεκτασιμότητα και σε μια πιο modular αρχιτεκτονική. Πέρα από την αρχική του εφαρμογή, το σύστημα μπορεί να συνδυάζει οπτικά δεδομένα με άλλους αισθητήρες και να προσαρμόζεται δυναμικά σε νέες εφαρμογές με αλλαγή του μοντέλου αναγνώρισης ή την παραμετροποίηση της επεξεργασίας.

Βασικός στόχος της εργασίας είναι η δημιουργία ενός συστήματος βασισμένο σε ένα προσιτό και χαμηλού κόστους hardware, το οποίο μπορεί να υλοποιήσει ένα πλήρως λειτουργικό pipeline μηχανικής όρασης με χαρακτηριστικά που συνήθως απαιτούν ισχυρούς υπολογιστές ή σύνδεση σε cloud υπηρεσίες. Το σύστημα υλοποιήθηκε με βάση τη λειτουργία σε πραγματικό χρόνο με χαμηλό latency modular αρχιτεκτονικής και δυνατότητα προσαρμογής χωρίς την ανάγκη ανακατασκευής της συνολικής υποδομής.

1.4 Δομή Εργασίας

Η παρούσα εργασία δομείται σε 5 βασικά κεφάλαια, καθένα εκ των οποίων καλύπτει διαδοχικά στάδια της μελέτης, της υλοποίησης και της αξιολόγησης του συστήματος.

Στο κεφάλαιο 2 παρουσιάζονται αναλυτικά παρόμοια συστήματα, τα πλεονεκτήματα τους και τα μειονεκτήματα τους. Στο κεφάλαιο 3 παρουσιάζονται αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση του συστήματος συμπεριλαμβανομένων των ηλεκτρονικών εξαρτημάτων που χρειάστηκαν, το λογισμικό και οι βιβλιοθήκες. Στο κεφάλαιο 4 γίνεται πλήρης περιγραφή του συστήματος, η αρχιτεκτονική και ο τρόπος λειτουργίας του, καθώς και τα αποτελέσματα από δοκιμές του στερεοσκοπικού συστήματος και του συνολικού συστήματος. Στο κεφάλαιο 5 αναλύονται τα αποτελέσματα, τι πέτυχε και τι όχι, καθώς και το πως μπορεί το έργο να εξελιχθεί στο μέλλον.

Κεφάλαιο 2ο: Βιβλιογραφική Ανασκόπηση

2.1 Εισαγωγή

Η αναγνώριση πινακίδων κυκλοφορίας αποτελεί κρίσιμο στοιχείο των σύγχρονων συστημάτων υποβοήθησης οδηγού και αυτόνομης οδήγησης. Στο κεφάλαιο αυτό εξετάζονται οι κύριες ερευνητικές προσεγγίσεις που έχουν αναπτυχθεί τα τελευταία χρόνια, από παραδοσιακές τεχνικές μηχανικής όρασης, μέχρι σύγχρονα συστήματα βαθιάς μάθησης. Αξίζει να σημειωθεί ότι ενώ εμπορικά συστήματα όπως το Tesla Vision και το Mobileye κυριαρχούν στην αγορά, η διαθέσιμη τεχνική πληροφορία για αυτά περιορίζεται σε γενικές περιγραφές και διαφημιστικό υλικό. Η απουσία λεπτομερών τεχνικών δημοσιεύσεων από αυτές τις εταιρείες δημιουργεί ένα κενό στη βιβλιογραφία, καθιστώντας δύσκολη τη σύγκριση ακαδημαϊκών προσεγγίσεων με πραγματικά εμπορικά συστήματα.

2.2 Παραδοσιακές Προσεγγίσεις με Μηχανές Διανυσμάτων Υποστήριξης

Η εργασία Kiran et al. [7] για ανίχνευση πινακίδων με SVM αντιπροσωπεύει την κλασική προσέγγιση πριν την επικράτηση των νευρωνικών δικτύων. Το σύστημα χρησιμοποιεί χρωματική τμηματοποίηση στον χώρο HSI για την απομόνωση περιοχών με έντονα χρώματα, ακολουθούμενη από ταξινόμηση σχήματος μέσω χαρακτηριστικών Distance to Border. Για την τελική αναγνώριση, εφαρμόζονται SVM με πυρήνα RBF σε χαρακτηριστικά ακμών από κανονικοποιημένες εικόνες 80x80 pixel.

Τα πειραματικά αποτελέσματα έδειξαν ακρίβεια 75-90% ανάλογα με την κατηγορία πινακίδας, με τα κίτρινα κυκλικά σήματα να επιτυγχάνουν την καλύτερη απόδοση στο 90.47%. Παρόλο που η μέθοδος είναι υπολογιστικά ελαφριά και κατάλληλη για συστήματα περιορισμένων πόρων, η εξάρτηση από χειροκίνητα σχεδιασμένα χαρακτηριστικά και η ευαισθησία στις μεταβολές φωτισμού περιορίζουν την αποτελεσματικότητά της σε πραγματικές συνθήκες. Η προσέγγιση αυτή αποτελεί ιστορικό σημείο αναφοράς που καταδεικνύει τους περιορισμούς των παραδοσιακών μεθόδων έναντι των σύγχρονων τεχνικών βαθιάς μάθησης που χρησιμοποιούμε στο προτεινόμενο σύστημα.

2.3 Η Επανάσταση της Βαθιάς Μάθησης

2.3.1 German Traffic Sign Recognition Benchmark

Το GTSRB Stallkamp et al. [8] αποτέλεσε σημείο καμπής στην έρευνα αναγνώρισης πινακίδων. Η βάση δεδομένων περιλαμβάνει πάνω από 50,000 εικόνες σε 43 κατηγορίες, συλλεγμένες από 10 ώρες βίντεο σε γερμανικούς δρόμους. Ο διαγωνισμός που οργανώθηκε το 2011 κατέδειξε την υπεροχή των συνελικτικών νευρωνικών δικτύων, με την καλύτερη λύση να επιτυγχάνει 98.98% ακρίβεια, ξεπερνώντας ακόμη και την ανθρώπινη απόδοση του 97.88%.

Η σημασία του GTSRB έγκειται στην καθιέρωση ενός κοινού σημείου αναφοράς που επέτρεψε την αντικειμενική σύγκριση διαφορετικών προσεγγίσεων. Ωστόσο, όπως επισημαίνουν μεταγενέστερες έρευνες, οι συνθήκες του dataset είναι σχετικά ιδανικές, με τις πινακίδες να καταλαμβάνουν σημαντικό μέρος της εικόνας και να μην αντιμετωπίζουν τις προκλήσεις του πραγματικού κόσμου όπως ακραίες αποστάσεις ή έντονες αποκρύψεις.

2.3.2 Ενσωματωμένα Συστήματα με CNN

Η υλοποίηση των Han και Oruklu [9] σε Jetson TX1 που παρουσιάζουν οι συγγραφείς αποδεικνύει τη δυνατότητα εκτέλεσης CNN σε ενσωματωμένες πλατφόρμες. Το σύστημα συνδυάζει ανίχνευση βασισμένη σε κανονικοποιημένο RGB με ταξινόμηση μέσω αρχιτεκτονικής τύπου LeNet. Με τρία

συνελικτικά στρώματα και τρεις πλήρως συνδεδεμένες στρώσεις, το δίκτυο επιτυγχάνει 96.2% ακρίβεια στο GTSRB dataset.

Το κύριο μειονέκτημα είναι η χαμηλή ταχύτητα επεξεργασίας των 1.6 FPS σε ανάλυση 1360x800, που οφείλεται στην έλλειψη βελτιστοποίησης για GPU. Αυτό υπογραμμίζει την πρόκληση της εξισορρόπησης ακρίβειας και ταχύτητας σε συστήματα πραγματικού χρόνου, πρόβλημα που το σύστημα που παρουσιάζεται στην παρούσα εργασία αντιμετωπίζει με τον συνδυασμό detection και tracking modes.

2.3.3 Βελτιστοποιημένα Δίκτυα για Embedded Systems

Η εργασία των Wong et al. [10] για το Tiny SSD προτείνει ένα συνελικτικό δίκτυο μεγέθους μόλις 2.3 MB για ανίχνευση αντικειμένων σε embedded συστήματα. Το δίκτυο συνδυάζει Fire modules από το SqueezeNet με την αρχιτεκτονική SSD, επιτυγχάνοντας 61.3% mAP στο VOC 2007 με μόνο 571 εκατομμύρια πράξεις, 26 φορές μικρότερο από το Tiny YOLO.

Ενώ το Tiny SSD εστιάζει στη μείωση μεγέθους, το σύστημα που παρουσιάζεται προτιμά το YOLO για καλύτερη ισορροπία ακρίβειας-ταχύτητας στην αναγνώριση πινακίδων. Επιπλέον, χρησιμοποιούνται τεχνικές όπως η δυναμική εναλλαγή detection-tracking που δεν εξετάζονται στο Tiny SSD, επιτρέποντας αποδοτική λειτουργία ακόμη και με μεγαλύτερα μοντέλα.

2.3.4 Αναγνώριση σε Πραγματικές Συνθήκες

Η εργασία των Zhu et al. [11] για το Tsinghua-Tencent 100K dataset αντιμετωπίζει την πρόκληση των πραγματικών συνθηκών όπου οι πινακίδες καταλαμβάνουν λιγότερο από 1% της εικόνας. Με 100,000 εικόνες από την υπηρεσία Tencent Street View και 30,000 σημειωμένες πινακίδες σε 45 κατηγορίες, αποτελεί μια από τις πιο ρεαλιστικές βάσεις δεδομένων.

Το προτεινόμενο βαθύ συνελικτικό δίκτυο με τρεις παράλληλες εξόδους για πλαίσιο, μάσκα και ταξινόμηση επιτυγχάνει 91% ανάκληση και 88% ακρίβεια, σημαντικά καλύτερα από το Fast R-CNN που περιορίζεται σε 56% και 50% αντίστοιχα. Η επιτυχία σε μικρές πινακίδες με ανάκληση 87-94% είναι ιδιαίτερα σημαντική για πρακτικές εφαρμογές.

2.4 Συστήματα Υπερύψηλης Ανάλυσης

Η προσέγγιση των Antonakakis et al. [12] για επεξεργασία εικόνων 48 megapixel σε Jetson AGX Xavier παρουσιάζει καινοτόμο λύση για την πρόκληση των εξαιρετικά υψηλών αναλύσεων. Το σύστημα χωρίζει την εικόνα σε έξι υπο-καρέ των 8 megapixel και τα επεξεργάζεται παράλληλα με YOLOv5 σε ξεχωριστά Docker containers.

Η παράλληλη επεξεργασία μειώνει τον χρόνο ανίχνευσης από 2.7 σε 0.3 δευτερόλεπτα, επιτρέποντας 1.25 FPS. Παρόλο που η ταχύτητα παραμένει χαμηλή για πραγματικό χρόνο, η τεχνική της τμηματοποίησης σε ROIs είναι άμεσα σχετική με το δικό μας σύστημα που χρησιμοποιεί παρόμοια στρατηγική για τη μείωση του υπολογιστικού φόρτου.

2.5 Γενική Επισκόπηση του Πεδίου

Η ανασκόπηση των Mogelmose et al[13] παρέχει ολοκληρωμένη εικόνα του πεδίου, τονίζοντας τα τρία βασικά στάδια επεξεργασίας: τμηματοποίηση, εξαγωγή χαρακτηριστικών και ανίχνευση. Επισημαίνει την έλλειψη κοινών βάσεων δεδομένων και την υπερβολική εστίαση σε ευρωπαϊκά σήματα, γεγονός που περιορίζει τη γενικευσιμότητα των συστημάτων.

Ιδιαίτερη έμφαση δίνεται στην αλληλεπίδραση με τον οδηγό, προτείνοντας συστήματα που επισημαίνουν μόνο τα σήματα που δεν προσέχει ο οδηγός για αποφυγή πληροφοριακής υπερφόρτωσης. Η ανάλυση των προκλήσεων περιλαμβάνει ομοιότητα σημάτων, φθορά, μεταβλητό φωτισμό και παρουσία άσχετων αντικειμένων που μοιάζουν με πινακίδες.

2.6 Εμπορικά Συστήματα Αναγνώρισης Πινακίδων

2.6.1 Mobileye

Η Mobileye παρουσιάζει την πρώτη vision-only λύση Intelligent Speed Assist (ISA) που καλύπτει τις απαιτήσεις της νέας Ευρωπαϊκής GSR, πιστοποιημένη για χρήση και στα 27 κράτη-μέλη της ΕΕ (καθώς και Νορβηγία, Ελβετία, Τουρκία). Το ISA «τρέχει» ως αναβάθμιση λογισμικού πάνω στην πλατφόρμα EyeQ (EyeQ4/EyeQ6), χωρίς νέο hardware, και βασίζεται σε συνδυασμό τεχνικών όπως traffic-sign relevancy ανά λωρίδα, signature-based ταξινόμηση για νεότερα σήματα, OCR για πινακίδες εισόδου πόλης και road-type classifier για περιπτώσεις χωρίς εμφανή σήμανση. Σύμφωνα με τη Mobileye, σε δοκιμές από έξι ανεξάρτητα εργαστήρια το σύστημα ξεπέρασε τα όρια της GSR και αναμένεται ενσωμάτωση από μεγάλους OEMs σε ευρωπαϊκά μοντέλα, με στόχο μείωση συγκρούσεων και θανάτων από υπερβολική ταχύτητα [14].

Στο επίπεδο υπολογιστικής πλατφόρμας, το EyeQ6H (7 nm, 34 INT8 TOPS) εκτελεί DNNs/transformers για real-time ταξινόμηση, ανίχνευση και τμηματοποίηση στην άκρη του οχήματος. Στη σελίδα μετρήσεων της εταιρείας, για single-stream ResNet-50 αναφέρεται 0.5 ms latency σε "Power mode" στο EyeQ6H, ενώ για σύγκριση παρατίθεται το NVIDIA Jetson AGX Orin 64 GB στα 1.64 ms (MaxQ) ή 0.64 ms (MAXN) [15][16]. Σε επίπεδο προϊόντος, το SuperVision αξιοποιεί δύο EyeQ5/6H SoCs και 11 κάμερες (συν προαιρετικά ραντάρ), REM-based AV maps και το μοντέλο ασφάλειας RSS για πλήρη οπτική αντίληψη 360° και προηγμένα ADAS [17].

Καθοριστικό πλεονέκτημα είναι το μαζικό data pipeline της Mobileye: μια βάση δεδομένων περίπου 200 PB με κλιπ οδήγησης από όλο τον κόσμο, που επισημειώνεται αυτόματα/χειροκίνητα από πανω από 2.500 annotators και επεξεργάζεται σε AWS με έως 500k CPU cores σε αιχμή (περίπου 50 εκ. datasets τον μήνα). Η κλίμακα αυτή επιτρέπει ταχεία προσαρμογή σε νέες ρυθμιστικές απαιτήσεις (όπως GSR/ISA) με απλές αναβαθμίσεις λογισμικού στα υπάρχοντα EyeQ [18].

Το δικό μας σύστημα, αν και δεν διαθέτει την κλίμακα δεδομένων της Mobileye, επιτυγχάνει συγκρίσιμη απόδοση σε τοπικό επίπεδο με σημαντικά χαμηλότερο κόστος. Η χρήση ανοιχτού κώδικα και πλήρης τεκμηρίωση επιτρέπουν την προσαρμογή σε διαφορετικές αγορές χωρίς την ανάγκη για proprietary λύσεις.

2.6.2 Tesla Vision

Η Tesla Vision είναι η βασισμένη σε κάμερες αρχιτεκτονική Autopilot της Tesla, που αντικατέστησε σταδιακά το ραντάρ (2021–2022) και κατόπιν τους υπερήχους (USS) στα περισσότερα μοντέλα. Με την αφαίρεση των USS, η Tesla λάνσαρε το vision-based occupancy network, ήδη σε χρήση στο FSD (Supervised), το οποίο προσφέρει υψηλής ευκρίνειας χωρική αντίληψη, μεγαλύτερη εμβέλεια και ικανότητα διάκρισης αντικειμένων. Σύμφωνα με την εταιρεία, τα Model 3/Y με Tesla Vision διατήρησαν ή βελτίωσαν τις αξιολογήσεις ενεργητικής ασφάλειας έναντι των εκδόσεων με ραντάρ και είχαν καλύτερες παρεμβάσεις AEB για πεζούς, με συνεχή βελτίωση μέσω λογισμικού [19].

Στο hardware, το FSD chip είναι custom SoC της Tesla (παραγωγή από το 2019), σχεδιασμένο ως drop-in αναβάθμιση με περίπου 100 W μέγιστη κατανάλωση. Κατασκευάζεται στη διεργασία 14 nm της

Samsung (~6 δισ. τρανζίστορ σε ~260 mm²), ενσωματώνει δώδεκα 64-bit ARM Cortex-A72 στα 2.2 GHz (σε τρεις τετράδες), μια ελαφριά GPU ~1 GHz (~600 GFLOPS) και μνήμη LPDDR4 128-bit @ 2133 MHz, πληρώντας AEC-Q100 Grade-2 [20]. Η πλατφόρμα συνεργάζεται με ένα σετ καμερών τοποθετημένων περιμετρικά του οχήματος (π.χ. πάνω από πινακίδα, στους μεσαίους στύλους, στο παρμπρίζ, στα φτερά, εμπρός πρόσοψη), πλαισιωμένο από ηλεκτρονικά υποβοηθούμενα συστήματα πέδησης και διεύθυνσης υψηλής ακρίβειας [21].

Στο λογισμικό, per-camera δίκτυα εκτελούν semantic segmentation, ανίχνευση αντικειμένων και μονοοφθαλμική εκτίμηση βάθους απευθείας από raw εικόνες· στη συνέχεια, birds-eye-view δίκτυα συνδυάζουν πολυκάμερο βίντεο για να παράγουν σε top-down μορφή την τοπολογία δρόμου, στατική υποδομή και 3D αντικείμενα. Ένα πλήρες build του Autopilot περιλαμβάνει 48 δίκτυα που απαιτούν 70.000 GPU-ώρες εκπαίδευσης και αποδίδουν περίπου 1.000 tensors ανά χρονικό βήμα, τροφοδοτούμενα από σενάρια που συλλέγονται σε πραγματικό χρόνο από στόλο εκατομμυρίων οχημάτων [22].

Σε αντίθεση με την κλειστή αρχιτεκτονική της Tesla, το προτεινόμενο σύστημα παραμένει πλήρως ανοιχτό και επαληθεύσιμο. Ενώ δεν διαθέτουμε custom silicon, η βελτιστοποίησή μας για Raspberry Pi και η χρήση Edge TPU επιτρέπουν ικανοποιητική απόδοση με κόστος κάτω των 200 ευρώ έναντι χιλιάδων για το FSD hardware.

2.7 Επίλογος

Η βιβλιογραφική ανασκόπηση κατέδειξε τη σημαντική εξέλιξη στην αναγνώριση πινακίδων κυκλοφορίας από τις παραδοσιακές μεθόδους μηχανικής μάθησης έως τα σύγχρονα συστήματα βαθιάς μάθησης. Οι αρχικές προσεγγίσεις με SVM και χειροκίνητα χαρακτηριστικά, αν και υπολογιστικά αποδοτικές, περιορίζονταν από την ευαισθησία τους σε μεταβολές φωτισμού και την ανάγκη εξειδικευμένου σχεδιασμού χαρακτηριστικών. Η εισαγωγή των συνελικτικών νευρωνικών δικτύων μέσω του GTSRB benchmark επέφερε επαναστατική βελτίωση, με ακρίβειες που ξεπερνούν την ανθρώπινη απόδοση.

Η μετάβαση σε ενσωματωμένα συστήματα ανέδειξε νέες προκλήσεις εξισορρόπησης ακρίβειας, ταχύτητας και υπολογιστικών πόρων. Εργασίες όπως το Tiny SSD και οι υλοποιήσεις σε Jetson TX1 κατέδειξαν διαφορετικές στρατηγικές βελτιστοποίησης, από τη δραστική μείωση μεγέθους μοντέλου έως την εκμετάλλευση εξειδικευμένου hardware. Παράλληλα, η ανάπτυξη datasets όπως το Tsinghua-Tencent 100K υπογράμμισε την ανάγκη για συστήματα που λειτουργούν αξιόπιστα σε πραγματικές συνθήκες με μικρές και μερικώς κρυμμένες πινακίδες.

Η εξέταση των εμπορικών συστημάτων αποκάλυψε το παράδοξο της σύγχρονης τεχνολογίας: ενώ εταιρείες όπως η Mobileye και η Tesla έχουν αναπτύξει εξαιρετικά προηγμένα συστήματα με εντυπωσιακές δυνατότητες, η έλλειψη τεχνικής διαφάνειας καθιστά αδύνατη την ανεξάρτητη επαλήθευση και την επιστημονική πρόοδο. Αυτό δημιουργεί ένα κενό μεταξύ ακαδημαϊκής έρευνας και βιομηχανικής πρακτικής που περιορίζει την καινοτομία.

Το προτεινόμενο σύστημα τοποθετείται στρατηγικά σε αυτό το τοπίο, συνδυάζοντας στοιχεία από διάφορες προσεγγίσεις. Από τα παραδοσιακά συστήματα διατηρούμε την έμφαση στην υπολογιστική αποδοτικότητα, από τα σύγχρονα CNN υιοθετούμε την αυτόματη εκμάθηση χαρακτηριστικών, από τα embedded συστήματα εφαρμόζουμε τεχνικές βελτιστοποίησης για περιορισμένους πόρους, και από τα datasets πραγματικών συνθηκών αντλούμε την ανάγκη για αξιοπιστία. Η κύρια καινοτομία έγκειται στη χρήση στερεοσκοπικής όρασης για ακριβή εκτίμηση απόστασης, στον δυναμικό συνδυασμό detection-

tracking για βελτιωμένη απόδοση, και στην ενσωμάτωση sensor fusion για συσχέτιση με την κινηματική του οχήματος.

Σε αντίθεση με τα κλειστά εμπορικά συστήματα, η προσέγγισή μας παραμένει πλήρως ανοιχτή και τεκμηριωμένη, συμβάλλοντας στην επιστημονική κοινότητα ενώ παράλληλα επιτυγχάνει πρακτική απόδοση σε προσιτό hardware. Αυτή η ισορροπία μεταξύ διαφάνειας, απόδοσης και κόστους καθιστά το σύστημα κατάλληλο τόσο για ερευνητικούς σκοπούς όσο και για πρακτικές εφαρμογές σε οχήματα.

Κεφάλαιο 3ο: Υλικό και Λογισμικό

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται ολοκληρωμένα η τεχνολογική στοίβα του συστήματος: το υλικό (Hardware) που υλοποιεί τη στερεοσκοπική λήψη και τα περιφερειακά, και το λογισμικό (Software) που καλύπτει από το λειτουργικό έως τους αλγόριθμους βαθμονόμησης, ανίχνευσης και παρακολούθησης. Η ενότητα 3.2 αποδομεί τις επιμέρους επιλογές υλικού—Raspberry Pi 5, Coral Edge TPU, δύο Pi Camera v3, Arduino-συμβατό μικροελεγκτή για LCD/LEDs, BNO055 και NEO-6M—μαζί με την τροφοδοσία και τη μηχανική κατασκευή, ώστε να τεκμηριώνεται γιατί ο συγκεκριμένος συνδυασμός καλύπτει απαιτήσεις πραγματικού χρόνου με ρεαλιστικό κόστος και διαθεσιμότητα στην αγορά.

Στο λογισμικό (ενότητα 3.3) περιγράφεται το περιβάλλον Raspberry Pi OS και η χρήση Python 3.11 για την ενορχήστρωση νημάτων/ροών, ενώ η C++/Arduino αξιοποιείται όπου απαιτείται ντετερμινιστικός έλεγχος περιφερειακών (LCD, LEDs). Εξειδικευμένες βιβλιοθήκες όπως Picamera2, NumPy, threading, PySerial και το driver της BNO055 καλύπτουν πρόσβαση σε κάμερες/αισθητήρες και αποδοτική διαχείριση δεδομένων.

Η επεξεργασία εικόνας βασίζεται στο OpenCV για βαθμονόμηση στέρεο-κάμερας (Zhang), διόρθωση και προεπεξεργασία, με ενδεικτικά αποσπάσματα κώδικα που τεκμηριώνουν τη ροή από την ανίχνευση γωνιών έως το stereoCalibrate. Για την ανίχνευση/παρακολούθηση σημάτων χρησιμοποιούνται μοντέλα YOLO (Ultralytics): ένα κύριο μοντέλο για ανίχνευση (π.χ. 512×512) και ένα ελαφρύτερο για tracking σε ROI (π.χ. 256×256), με υποστήριξη επιτάχυνσης μέσω Edge TPU

3.2 Hardware

3.2.1 Raspberry Pi 5

Ο κεντρικός υπολογιστής του συστήματος είναι το Raspberry Pi 5 με 16 GB RAM, η πιο πρόσφατη και ισχυρή έκδοση της σειράς Raspberry Pi και κυκλοφόρησε στην αγορά τον Οκτώβριο του 2023. Πρόκειται για έναν SBC που συνδικάζει την υπολογιστική ισχύ με μικρό μέγεθος και τη χαμηλή κατανάλωση των ενσωματωμένων συστημάτων.

Αναλυτικά Χαρακτηριστικά:

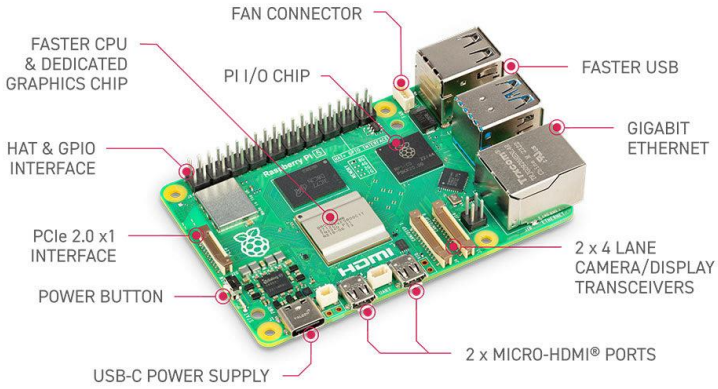
- CPU: Broadcom BCM2712, Quad-core ARM Cortex-A76 @ 2.4GHz με 512KB L2 cache ανά πυρήνα και 2MB shared L3 cache
- GPU: VideoCore VII με υποστήριξη OpenGL ES 3.1, Vulkan 1.2
- Μνήμη RAM: 16GB LPDDR4X-4267 SDRAM με bandwidth 34GB/s
- Συνδεσιμότητα: Gigabit Ethernet, Dual-band 802.11ac Wi-Fi, Bluetooth 5.0/BLE
- Διεπαφές βίντεο: 2x 4-lane MIPI DSI/CSI ports για ταυτόχρονη σύνδεση δύο καμερών
- I/O: 40-pin GPIO header με υποστήριξη I2C, SPI, UART, PWM
- USB: 2x USB 3.0 ports (5Gbps), 2x USB 2.0 ports
- Αποθήκευση: MicroSD card slot με υποστήριξη SDR104 (100MB/s)
- Τροφοδοσία: 5V/5A μέσω USB-C με Power Delivery support
- Κατανάλωση: 4W idle, 7-8W τυπική χρήση, έως 12W υπό πλήρες φορτίο
- Ψύξη: Ενεργητική ψύξη με dedicated fan header και ενσωματωμένο PWM controller

Η επιλογή του Raspberry Pi 5 με 16GB RAM βασίστηκε σε συγκεκριμένες απαιτήσεις του συστήματος. Η επεξεργασία στερεοσκοπικών εικόνων υψηλής ανάλυσης (1920x1080) από δύο κάμερες ταυτόχρονα

απαιτεί σημαντικούς πόρους μνήμης. Κάθε frame καταλαμβάνει περίπου 6MB (1920×1080×3 bytes), και με τα διάφορα buffers, τα preprocessing stages, και τα δύο YOLO μοντέλα φορτωμένα στη μνήμη.

Ο νέος επεξεργαστής Cortex-A76 προσφέρει 2-3x καλύτερη απόδοση από τον προκάτοχό του (Cortex-A72 του Pi 4), με σημαντικές βελτιώσεις στο instruction-per-cycle (IPC) και την αρχιτεκτονική out-of-order execution. Αυτό μεταφράζεται σε ταχύτερη επεξεργασία των OpenCV operations όπως το resize, color conversion, και remapping που χρησιμοποιούνται εκτενώς στο pipeline [10].

Η νέα αρχιτεκτονική I/O με dedicated RP1 southbridge chip απελευθερώνει τον κύριο επεξεργαστή από περιφερειακές λειτουργίες, βελτιώνοντας την απόδοση κατά 20-30% σε I/O-intensive εφαρμογές. Επιπλέον, οι δύο ξεχωριστές 4-lane MIPI CSI θύρες επιτρέπουν την ταυτόχρονη λήψη από δύο κάμερες χωρίς bandwidth limitations που υπήρχαν στις προηγούμενες εκδόσεις [23].



Εικόνα 3.1: Raspberry Pi 5

[https://cdn.shopify.com/s/files/1/0254/1191/1743/files/5047-5048_description-raspberry-pi-5-features.jpg?v=1695822743]

3.2.2 Google Colar Edge TPU

Το Google Coral USB Accelerator είναι ένας εξωτερικός επιταχυντής τεχνητής νοημοσύνης που συνδέεται μέσω USB και επιταχύνει δραματικά την εκτέλεση νευρωνικών δικτύων. Στην ουσία, πρόκειται για ένα εξειδικευμένο chip (Edge TPU) που είναι σχεδιασμένο αποκλειστικά για να τρέχει μοντέλα μηχανικής μάθησης πολύ γρήγορα και με χαμηλή κατανάλωση ενέργειας.

Βασικά Χαρακτηριστικά:

- Edge TPU chip με απόδοση 4 TOPS (4 τρισεκατομμύρια operations ανά δευτερόλεπτο)
- Σύνδεση USB 3.0 (Type-C), συμβατό και με USB 2.0
- Κατανάλωση μόλις 2W
- Διαστάσεις: 65mm x 30mm x 8mm (σαν μεγάλο USB stick)
- Παθητική ψύξη με ενσωματωμένη ψήκτρα αλουμινίου

Χωρίς το Coral, το Raspberry Pi θα χρειαζόταν περίπου 125 ms για να αναλύσει μία εικόνα με το YOLO μοντέλο ανίχνευσης - δηλαδή μόλις 8 frames ανά δευτερόλεπτο. Με το Coral, ο χρόνος πέφτει στα 35ms, επιτρέποντας 15+ frames ανά δευτερόλεπτο που είναι απαραίτητα για real-time ανίχνευση πινακίδων σε κίνηση. Είναι σαν να προσθέτουμε έναν εξειδικευμένο "συνεπεξεργαστή" που αναλαμβάνει μόνο τους υπολογισμούς τεχνητής νοημοσύνης.

Πρέπει να σημειωθεί ότι η Google έχει ουσιαστικά εγκαταλείψει την ανάπτυξη της σειράς Coral από το 2021, χωρίς νέα προϊόντα ή σημαντικές ενημερώσεις. Το documentation παραμένει online αλλά η κοινότητα υποστήριξης έχει συρρικνωθεί σημαντικά. Παρόλα αυτά, το hardware παραμένει εξαιρετικά αξιόπιστο και η υποστήριξη από την Ultralytics (YOLO) εξασφαλίζει ότι τα μοντέλα μας θα συνεχίσουν να λειτουργούν.[24]

Εικόνα 3.2: Google USB Accelerator

[https://lh3.googleusercontent.com/y5YE-KW6BGFdYMJHS8UUktFrD6cow0mbU4pBCQnmO6_5fiBdUHEAGTXwH8QJ-6zfA0iP53C45CC_ur3t33QB-DDubfqomtj9kSFz7g=s0]

Στο σύστημα αναγνώρισης πινακίδων, το Coral αναλαμβάνει την επεξεργασία δύο διαφορετικών μοντέλων YOLOv8n:

- Ένα μοντέλο 512×512 pixels για την αρχική ανίχνευση των πινακίδων στην εικόνα
- Ένα μικρότερο 256×256 pixels για την παρακολούθηση (tracking) των πινακίδων που έχουν ήδη εντοπιστεί

Η διαδικασία εγκατάστασης και χρήσης είναι εντυπωσιακά απλή χάρη στην εξαιρετική υποστήριξη από την Ultralytics (τους δημιουργούς του YOLO). Χρειάζονται μόνο δύο βήματα:

Πρώτον, τα μοντέλα YOLOv8 πρέπει να μετατραπούν σε INT8 μορφή που μπορεί να "καταλάβει" το Edge TPU. Αυτό γίνεται με μία εντολή που παρέχει η Ultralytics, η οποία αυτόματα κάνει την quantization και βελτιστοποίηση για το Coral.

Δεύτερον, εγκαθιστούμε την ειδική έκδοση της βιβλιοθήκης libedgetpu που παρέχει η Ultralytics. Υπάρχουν δύο επιλογές: η standard version για κανονική χρήση ή η max version που τρέχει το chip σε υψηλότερη συχνότητα για ακόμα καλύτερη απόδοση (με αυξημένη θερμοκρασία).

Μετά από αυτά τα δύο βήματα, το σύστημα λειτουργεί αυτόματα. Όταν ο κώδικάς μας φορτώνει ένα αρχείο μοντέλου με κατάληξη "_edgetpu.tflite", το Raspberry Pi αναγνωρίζει αυτόματα ότι πρέπει να το στείλει στο Coral για εκτέλεση αντί να το τρέξει στον δικό του επεξεργαστή. [25]

Αυτή η ευκολία χρήσης ήταν ένας ακόμα λόγος επιλογής του Coral έναντι άλλων λύσεων που απαιτούν πολύπλοκες διαδικασίες μετατροπής μοντέλων και εξειδικευμένο κώδικα για την αξιοποίησή τους.

Ένας ακόμα σημαντικός ρόλος για την επιλογή του Coral είναι το κόστος. Στην ελληνική αγορά η τιμή του κυμαίνεται γύρω στα 90€. Σε συνδυασμό με το Raspberry το κόστος μόνο για τις επεξεργαστικές μονάδες φτάνει στα 240€. Ένα σημαντικό ποσό για να θεωρηθεί χαμηλό αλλά που είναι χαμηλότερο από αυτό του κοντινότερου ανταγωνιστή του. Η μονή πραγματική εναλλακτική είναι το νέο Jetson Orin

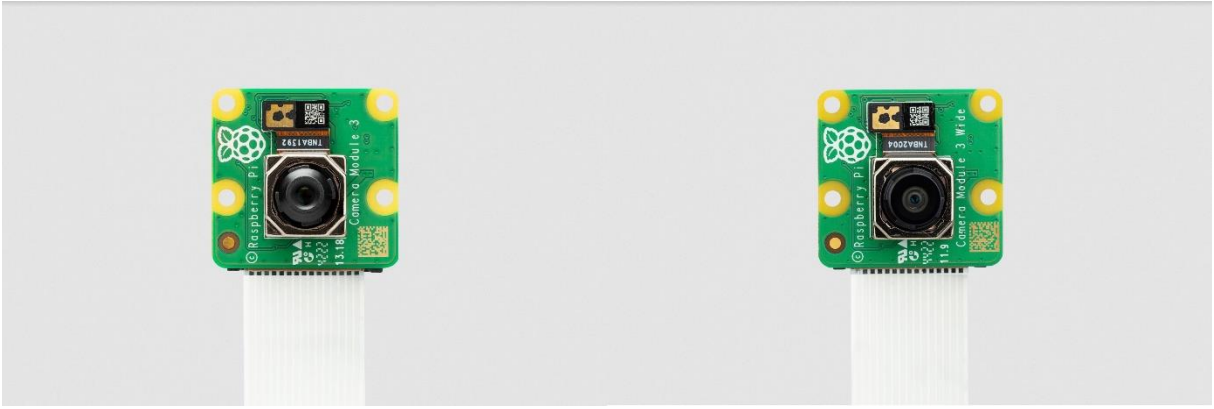
Nano Super της NVidia που κυκλοφόρησε τον Ιανουάριο του 2024. Τα χαρακτηριστικά του είναι πολύ καλυτέρα από αυτά του Raspberry Pi και του Coral. Και η προτεινομένη τιμή λιανικής στα 200$ το κάνει εξαιρετική λύση. Στην ελληνική αγορά η τιμή φτάνει στα 350€ και, σε συνδυασμό με την έλλειψη αποθέματος παγκοσμίως, καθιστά την επιλογή του παραπάνω συνδυασμού μονόδρομο. [26-28]

3.2.3 Raspberry Picam V3 Std

Το σύστημα στερεοσκοπικής όρασης βασίστηκε σε δυο ιδίες κάμερες Raspberry Pi Camera Module v3, που αποτελούν την τελευταία γενιά καμερών της Raspberry Pi Foundation. Κυκλοφόρησαν τον Ιανουάριο του 2023 και φέρνουν σημαντικές βελτιώσεις σε σχέση με την προηγούμενη γενιά.

Αναλυτικά Χαρακτηριστικά:

- Αισθητήρας: Sony IMX708, 1/2.43" CMOS με τεχνολογία BSI (Back-Side Illumination)
- Ανάλυση: 11.9 megapixels (4608×2592 pixels)
- Pixel size: 1.4μm x 1.4μm με 2x2 binning support
- Οπτικό σύστημα:
    - Standard variant: 66° horizontal FoV, f/1.8 aperture, 4.74mm focal length
    - Υποστήριξη Phase Detection Auto Focus (PDAF) με 493 focus points
    - Focus range: 10cm έως άπειρο
- Δυνατότητες βίντεο:
    - 2304×1296 @ 56fps με 2x2 binning
    - 1920×1080 @ 50fps
    - 1280×720 @ 120fps για slow motion
- HDR: Υποστήριξη DOL-HDR (Digital Overlap HDR) με έως 3 exposures
- Επεξεργαστής ISP: Ενσωματωμένος στον αισθητήρα με υποστήριξη για noise reduction, lens shading correction
- Διεπαφή: 15-pin flex cable για σύνδεση σε CSI-2 port
- Κατανάλωση: ~300mW κατά τη λειτουργία



Εικόνα 3.3: Raspberry PiCamV3 Modules

[https://www.raspberrypi.com/documentation/accessories/images/cm3.jpg?hash=87abd731a81318a03d494783da281fc]

Οι δυο κάμερες τοποθετήθηκαν σε οριζόντια διάταξη με απόσταση μεταξύ των οπτικών τους κέντρων (baseline) περίπου 10 εκατοστά, προσομοιώνοντας με αυτό τον τρόπο την ανθρώπινη στερεοσκοπική όραση. Αυτή η απόσταση επιλέχθηκε βάση της σχέσης:

$$Z = \frac{f * B}{d} \tag{3.1}$$

Οπού Z είναι η απόσταση του αντικειμένου, f η εστιακή απόσταση σε pixels, B το baseline και d το disparity σε pixels. [52]

Η χρήση του PDAF (Phase Detection Auto Focus) είναι κρίσιμη για την εφαρμογή, καθώς οι πινακίδες μπορεί να εμφανιστούν σε διάφορες αποστάσεις. Το σύστημα autofocus του IMX708 χρησιμοποιεί masked pixels στον αισθητήρα για να μετρήσει τη διαφορά φάσης και να υπολογίσει την απόσταση εστίασης σε λιγότερο από 100ms, σημαντικά ταχύτερα από τα παραδοσιακά contrast-detection συστήματα. [29]

3.2.4 Geekcrate Uno Clone

Χρησιμοποιείται ένα Arduino Uno compatible board βασισμένο στον μικροελεγκτή ATmega328P. Παρόλο που πρόκειται για clone και όχι για genuine Arduino, χρησιμοποιεί τον ίδιο μικροελεγκτή και είναι πλήρως συμβατό σε hardware και software επίπεδο.

Χαρακτηριστικά του ATmega328P:

- Αρχιτεκτονική: 8-bit AVR RISC με Harvard architecture
- Ταχύτητα: 16MHz από εξωτερικό crystal oscillator
- Μνήμη:
    - 32KB Flash memory (0.5KB χρησιμοποιείται από τον bootloader)
    - 2KB SRAM για μεταβλητές
    - 1KB EEPROM για μόνιμη αποθήκευση
- Περιφερειακά:
    - 3 timers (Timer0/Timer2: 8-bit, Timer1: 16-bit)
    - 6 PWM channels (pins 3,5,6,9,10,11) με 8-bit resolution
    - 10-bit ADC με 6 multiplexed channels
    - USART για σειριακή επικοινωνία έως 2Mbps
    - I2C/TWI interface έως 400kHz
    - SPI interface έως 10MHz
- Κατανάλωση: ~45mA @ 16MHz, 5V [30]

Το Arduino λειτουργεί ως dedicated display controller με real-time εγγυήσεις. Ο κώδικας που τρέχει είναι γραμμένος σε C++ και χρησιμοποιεί interrupts για την ασύγχρονη λήψη εντολών από το Pi μέσω UART. Η αρχιτεκτονική του προγράμματος βασίζεται σε non-blocking state machine για τα LED animations (blinking, pulsing) χρησιμοποιώντας τη millis() function αντί για delays.

Η επικοινωνία με το Pi γίνεται μέσω USB που εμφανίζεται ως virtual COM port (/dev/ttyACM0)

Εικόνα 3.4: GEEKCRAIT Arduino clone

[https://media.s-bol.com/2rj5zvo3KWz/550x390.jpg]

3.2.5 BNO055

Ο αισθητήρας Bosch BNO055 αποτελεί ολοκληρωμένη μονάδα εννέα βαθμών ελευθερίας (9-DOF), η οποία συνδυάζει επιταχυνσιόμετρο 16-bit (accelerometer), γυροσκόπιο 16-bit(gyroscope) και μαγνητόμετρο (magnetometer) σε ένα ενιαίο chip. Η ιδιαιτερότητα του φαίνεται στο γεγονός ότι διαθέτει ενσωματωμένο μικροελεγκτή που εκτελεί sensor fusion αλγορίθμους προσφέροντας απευθείας έξοδο προσανατολισμού (quaternions, Euler angles) και γραμμικής επιτάχυνσης, χωρίς την ανάγκη επιπλέον επεξεργασίας στον κεντρικό επεξεργαστή.

Η μονάδα επικοινωνεί με το Raspberry Pi μέσω διαύλου I²C. Χρησιμοποιείται για τη μέτρηση της επιτάχυνσης του οχήματος σε πραγματικό χρόνο, καθώς και για τη βελτίωση της εκτίμησης ταχύτητας κατά τις περιόδους όπου το σήμα GPS παρουσιάζει καθυστέρηση ή αστάθεια.

Πριν από κάθε χρήση, εκτελείται διαδικασία βαθμονόμησης (calibration). Η γραμμική επιτάχυνση (linear acceleration) φιλτράρεται για την εξάλειψη της συνιστώσας βαρύτητας, και στη συνέχεια μετατρέπεται σε ταχύτητα μέσω αριθμητικής ολοκλήρωσης με διορθωτικούς μηχανισμούς για τον περιορισμό του drift.

Η χρήση του BNO055 κρίθηκε κατάλληλη λόγω της ευκολίας ενσωμάτωσης, και της εσωτερικής επεξεργασίας που μειώνει τον φόρτο του Raspberry Pi. Επιπλέον, επιτρέπει την ανάπτυξη μελλοντικών λειτουργιών χωρίς αλλαγή hardware. [31][132]

Εικόνα 3.5: DfRobot BNO55

[https://grobotronics.com/images/detailed/132/ori-capteur-9-axes-bno055-sen0374-32860_grobo.jpg]

3.2.6 NEO-6M GPS

Ο δέκτης Ublox NEO-6M αποτελεί μία δημοφιλή και οικονομικά αποδοτική μονάδα GPS, η οποία χρησιμοποιείται για την εκτίμηση της στιγμιαίας ταχύτητας και θέσης του οχήματος σε πραγματικό χρόνο. Η επικοινωνία με το Raspberry Pi γίνεται μέσω σειριακής διεπαφής UART, ενώ η κατανάλωση ισχύος είναι μικρή, καθιστώντας τον κατάλληλο για χρήση σε embedded συστήματα.

Το module λειτουργεί με συχνότητα ανανέωσης 1 Hz, γεγονός που το καθιστά επαρκές για τη βασική εκτίμηση ταχύτητας σε απλές συνθήκες, αλλά όχι για πολύ γρήγορες δυναμικές αλλαγές. Για τον λόγο αυτόν, η μέτρηση ταχύτητας από το GPS συνδυάζεται με δεδομένα επιτάχυνσης από τον αισθητήρα BNO055 μέσω Kalman φίλτρου.

Ο NEO-6M παρέχει δεδομένα σε μορφή NMEA sentences, από τα οποία εξάγεται το VTG (Velocity over Ground) για υπολογισμό ταχύτητας και το GGA για εκτίμηση γεωγραφικής θέσης και ακρίβειας.

Η μέση ακρίβεια ταχύτητας κυμαίνεται μεταξύ ±0.1–0.3 km/h σε σταθερό περιβάλλον, ενώ ο αριθμός ορατών δορυφόρων επηρεάζει την αξιοπιστία.

Ο δέκτης υποστηρίζει αυτόνομη λειτουργία (χωρίς ανάγκη για εξωτερικό χρονισμό ή RTC) και ενσωματώνει κεραία τύπου patch, με δυνατότητα αναβάθμισης σε active antenna μέσω U.FL connector για βελτίωση της λήψης. Η επιλογή του συγκεκριμένου μοντέλου έγινε λόγω της ευρείας υποστήριξης σε open-source βιβλιοθήκες (π.χ. gpsd, pynmea2), της ευκολίας σύνδεσης με το Raspberry Pi, και της σταθερής απόδοσης σε βασικές εφαρμογές μέτρησης ταχύτητας. [33]

Εικόνα 3.6: GPS NEO 6M

[https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2017/10/NEO-GPS-1.jpg?resize=670%2C526&quality=100&strip=all&ssl=1https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2017/10/NEO-GPS-1.jpg?resize=670%2C526&quality=100&strip=all&ssl=1]

3.2.7 Lcd Οθόνη

Για την εμφάνιση πληροφοριών στον οδηγό χρησιμοποιείται μια τυπική αλφαριθμητική οθόνη LCD 16 χαρακτήρων x 2 γραμμών. Πρόκειται για την πιο διαδεδομένη οθόνη στον χώρο των embedded systems, που συνδυάζει χαμηλό κόστος, ευκρίνεια και αξιοπιστία.

Η συγκεκριμένη οθόνη επιλέχθηκε για πολλούς λόγους. Πρώτον, η αλφαριθμητική εμφάνιση είναι ιδανική για την παρουσίαση απλών πληροφοριών όπως ταχύτητα και απόσταση, με μεγάλους ευανάγνωστους χαρακτήρες που διαβάζονται εύκολα ακόμα και με την περιφερειακή όραση κατά την οδήγηση. Δεύτερον, το μπλε backlight προσφέρει καλή αντίθεση τόσο την ημέρα όσο και τη νύχτα χωρίς να τυφλώνει τον οδηγό. Ακόμα, η προσθήκη του I2C adapter είναι κρίσιμη καθώς μειώνει τις απαιτούμενες συνδέσεις από 12 pins (στην παράλληλη σύνδεση) σε μόλις 2 (SDA και SCL), απελευθερώνοντας πολύτιμα pins στο Arduino για άλλες λειτουργίες. Επιπλέον, το I2C πρωτόκολλο επιτρέπει την προσθήκη περισσότερων συσκευών στο μέλλον στον ίδιο δίαυλο.

Η οθόνη εμφανίζει διαφορετικές πληροφορίες ανάλογα με την κατάσταση:

- Κατά τη βαθμονόμηση: "Calibrating..." και μετρητές δειγμάτων IMU/GPS
- Κανονική οδήγηση: "Limit: XX km/h" / "Speed: XX km/h"
- Εντοπισμός πινακίδας: "Sign: XX km/h" / "Dist: X.Xm"
- Πινακίδα STOP: "!!! STOP !!!" / "Dist: X.Xm"

Εικόνα 3.7: Lcd Screen 16x2

[https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.skroutz.gr%2Fc%2F4749%2Fothoni%2Ff%2F1430043%2FLCD.html%3Fkeyphrase%3D16x2&psig=AOvVaw2KFMRi6dwoKvmmjJBdYN3m&ust=1756076311112000&source=images&cd=vfe&opi=89978449&ved=0CBUQjRxqFwoTCJCqm52Eoo8DFQAAAAAdAAAAABAE]

3.2.8 RGB LEDs

Για άμεση οπτική ένδειξη της κατάστασης του συστήματος χρησιμοποιούνται 5 κοινά RGB LED που μπορούν να παράγουν διάφορα χρώματα και εφέ. Τοποθετημένα σε εμφανές σημείο, παρέχουν στον οδηγό άμεση πληροφόρηση χωρίς να χρειάζεται να κοιτάξει την οθόνη.

Το σύστημα χρησιμοποίει διαφορετικά χρώματα και εφέ για κάθε κατάσταση

Χρώμα/Εφέ	Κατάσταση	Σημασία
Μπλε παλμός	Βαθμονόμηση IMU	Το όχημα πρέπει να παραμείνει ακίνητο
Πράσινο σταθερό	Κανονική οδήγηση	Ταχύτητα εντός ορίου
Κοκκίνο σταθερό	Υπέρβαση ορίου	Ταχύτητα πάνω από το όριο
Κίτρινο αναβόσβημα	Εντοπισμός πινακίδας ορίου	Νέο όριο εντοπίστηκε
Κοκκίνο αναβόσβημα	Εντοπισμός πινακίδας STOP	Προσοχή απαιτείται πλήρης στάση

Πίνακας 3.1 Καταστάσεις RGB LED

3.2.9 Τροφοδοσία

Η σταθερή και ασφαλής τροφοδοσία αποτελεί κρίσιμο παράγοντα για τη σωστή λειτουργία του συστήματος, ιδίως όταν αυτό λειτουργεί εντός οχήματος, όπου παρατηρούνται συχνές μεταβολές τάσης και παροδικές διακοπές. Ο σκοπός του buck converter είναι να παρέχει ρυθμισμένη τάση στο UPS, το οποίο με τη σειρά του εξασφαλίζει συνεχή και σταθερή παροχή ρεύματος στο Raspberry Pi.

3.2.9.1 Buck Converter

Ο μετατροπέας XH-M403 χρησιμοποιείται για τη μετατροπή της τάσης 12–14.4 V του συστήματος του οχήματος σε σταθερή έξοδο 12.6 V με τουλάχιστον 2 A ρεύμα, ικανό να καλύψει τις απαιτήσεις του UPS. Βασίζεται στο ολοκληρωμένο κύκλωμα XL4016E1, το οποίο υποστηρίζει ρυθμιζόμενη έξοδο, αποδοτική λειτουργία μέσω PWM, και διαθέτει ψήκτρα για θερμική σταθερότητα. Το κύκλωμα περιλαμβάνει συστήματα προστασίας από υπερθέρμανση, βραχυκύκλωμα και απότομες αυξήσεις τάσης, εξασφαλίζοντας ασφαλή λειτουργία ακόμα και σε απαιτητικά περιβάλλοντα. [34]

Εικόνα 3.8: Buck Converter

[https://www.eelectronicparts.com/cdn/shop/products/57_3e6a0313-8c23-44e6-a1f0-6673d0006249_1024x1024.jpg?v=1571252153]

3.2.9.2 UPS

Το Waveshare UPS Module 3S αποτελεί μια ολοκληρωμένη λύση αδιάλειπτης τροφοδοσίας, ειδικά σχεδιασμένη για ενσωματωμένα συστήματα όπως το Raspberry Pi 5 ή το Jetson Nano. Διαθέτει ενσωματωμένο step-down μετατροπέα και παρέχει σταθερή έξοδο 5 V / 5 A, ενώ λειτουργεί με τρεις επαναφορτιζόμενες μπαταρίες τύπου 18650 συνδεδεμένες σε σειρά. Η μονάδα υποστηρίζει επικοινωνία μέσω διαύλου I2C, επιτρέποντας παρακολούθηση βασικών ηλεκτρικών μεγεθών σε πραγματικό χρόνο (τάση, ρεύμα, ισχύς), χαρακτηριστικό που την καθιστά κατάλληλη για εφαρμογές όπου απαιτείται αξιοπιστία και έλεγχος της τροφοδοσίας. [35]

Εικόνα 3.9: UPS Module 3s

[https://www.waveshare.com/w/upload/e/e4/UPS_3S_RPI.jpg]

3.2.10 Κατασκευή

Η κατασκευή του συστήματος πραγματοποιήθηκε εξ ολοκλήρου με έμφαση στην φορητότητα, μηχανική σταθερότητα και ευκολία συντήρησης. Τα υποσυστήματα συρναμολογούνται σε ένα πλαίσιο συναρμολογημένο από ένα συνδυασμό μεταλλικών ραγών τύπου MicroRax και 3D printed εξαρτημάτων έτσι ώστε να επιτευχθεί βέλτιστη αξιοπιστία σε περιβάλλον οχήματος.

Τα 3D printed στηρίγματα και βάσεις σχεδιάστηκαν στο Fusion360 και εκτυπώθηκαν σε υλικό ASA cf το οποίο είναι ικανό να αντέχει μεγάλες θερμοκρασίες και είναι ανθεκτικό στην υπεριώδη ακτινοβολία. Επιπλέον, το ανθρακόνημα προσθέτει μηχανική ακαμψία χωρίς να αυξάνει το βάρος. Η εκτυπώσεις έγιναν με εκτυπωτή Bambu Lab P1s.

Η όλη κατασκευή επιτρέπει εύκολη αποσυναρμολόγηση, αντικατάσταση εξαρτημάτων, καθώς και δοκιμές σε επιτραπέζιο περιβάλλον, πριν εγκατασταθεί οριστικά στο όχημα. Η επιλογή αυτής της υλοποίησης προσφέρει μια modular, αναβαθμίσιμη και τεχνικά εύχρηστη πλατφόρμα. [37]

3.3 Λογισμικό

Η λειτουργία του συστήματος βασίζεται σε συνδυασμό ανοιχτού λογισμικού, βιβλιοθηκών μηχανικής όρασης και εργαλείων Inference σχεδιασμένων να λειτουργούν αποδοτικά σε embedded περιβάλλον. Οι τεχνολογίες που χρησιμοποιήθηκαν επιλέχθηκαν με βάση την σταθερότητα τους, την συμβατότητα με το Raspberry Pi και την ευκολία ενσωμάτωσης τους σε modular αρχιτεκτονική.

3.3.1 Raspberry Pi OS (Raspbian)

Το σύστημα βασίζεται στο Raspberry Pi OS (Bookworm, 64-bit) με desktop περιβάλλον, αποτελεί την επίσημο λειτουργικό distro του Raspberry Pi Foundation, το οποίο βασίζεται στο Debian Linux. Αν και η έκδοση "Lite" (χωρίς γραφικό περιβάλλον) είναι συχνά προτιμότερη για embedded συστήματα λόγω μικρότερης κατανάλωσης πόρων, στην παρούσα υλοποίηση επιλέχθηκε η κανονική έκδοση με γραφικό περιβάλλον για τους εξής λόγους:

- Ευκολία ανάπτυξης και δοκιμών: Το γραφικό περιβάλλον διευκολύνει την παρακολούθηση των καταγραφών (logs), την οπτική επιβεβαίωση της αναγνώρισης του μοντέλου ζωντανά καθώς και την χρήση εργαλείων όπως browser και VS Code.
- Διαχείριση εξωτερικών συσκευών: Καθώς το σύστημα βρίσκεται σε συνεχή ανάπτυξη, το desktop περιβάλλον διευκολύνει την σύνδεση και παρακολούθηση περιφερειακών
- Πλήρη υποστήριξη για βιβλιοθήκες: Η desktop έκδοση περιλαμβάνει προ εγκατεστημένα εργαλεία και drivers που μειώνουν σημαντικά τον χρόνο ρύθμισης του περιβάλλοντος και βοηθούν στην σταθερή λειτουργία του συστήματος

Η 64-bit αρχιτεκτονική επιλέχθηκε ώστε να αξιοποιηθεί το πλήρες εύρος της μνήμης RAM του Raspberry Pi 5 (16 GB) και να εξασφαλίσει την συμβατότητα με τις πιο πρόσφατες εκδόσεις των βιβλιοθηκών μηχανικής όρασης και AI. Η επιλογή ενός λειτουργικού με πλήρη desktop περιβάλλον βοήθησε φάση ανάπτυξης και testing χωρίς να επηρεάζει σημαντικά τη σταθερότητα ή την απόδοση.

3.3.2 Python 3.11

Η Python είναι μια intrperted, υψηλού επιπέδου γλώσσα προγραμματισμού που αναπτύχθηκε από τον Guido van Rossum και κυκλοφόρησε πρώτη φορα το 1991 [37]. Χαρακτηρίζεται από την σαφή της σύνταξη και την εκτεταμένη standard library που περιλαμβάνει έτοιμες λύσεις για πολλές

προγραμματιστικές ανάγκες [38]. Στο παρόν έργο χρησιμοποιείται η έκδοση Python 3.11, η οποία εισήγαγε βελτιώσεις στην ταχύτητα εκτέλεσης και καλυτέρα μηνύματα σφαλμάτων [39].

Η Python διαθέτει σύστημα modules που επιτρέπει την οργάνωση του κώδικα σε επαναχρησιμοποιήσιμα τμήματα [40]. Στο παρόν σύστημα, κάθε λειτουργικότητα οργανώνεται σε ξεχωριστό module.

```
# main.py
from src.utils import DataBus
from src.core import stereo_run, SpeedPipeline
from src.utils import initialize_session_manager
import simple_config as config

# Αρχικοποίηση διαχειριστή συνεδρίας για οργάνωση αποτελεσμάτων
session_manager = initialize_session_manager(config.OUTPUT_DIR)

# Δημιουργία κεντρικού διαύλου δεδομένων για επικοινωνία μεταξύ modules
shared = DataBus(
  arduino_port=config.ARDUINO_PORT,
  threshold_distance=config.DISTANCE_THRESHOLD
)
```

Το σύστημα οργανώνεται σε πακέτα (packages) που βρίσκονται στον φάκελο src, με κάθε υποφάκελο να περιέχει σχετικές λειτουργίες. Το statement from src.utils import DataBus εισάγει την κλάση DataBus από το module utils που βρίσκεται μέσα στο πακέτο src. Η κλάση DataBus λειτουργεί ως κεντρικός δίαυλος επικοινωνίας μεταξύ όλων των υποσυστημάτων, διαχειριζόμενη την ανταλλαγή δεδομένων μεταξύ της επεξεργασίας εικόνας, των αισθητήρων και του Arduino. Το αρχείο simple_config.py περιέχει όλες τις παραμέτρους του συστήματος σε μία κεντρική τοποθεσία, επιτρέποντας εύκολη τροποποίηση χωρίς αλλαγή του κώδικα.

Από την έκδοση 3.7, η Python προσφέρει τις dataclasses που απλοποιούν τη δημιουργία κλάσεων για αποθήκευση δεδομένων [41]. Χρησιμοποιουνται εκτενως για στοπ συστημα για την οργανωση των δεδομενων καταγραφης:

```
# pipeline_logger.py
from dataclasses import dataclass
from typing import Optional, List

@dataclass
class SpeedEvent:
  timestamp: float          # Χρονική σήμανση καταγραφής συμβάντος
  event: str                # Τύπος συμβάντος (π.χ. 'speed_detection')
  speed_kmh: float          # Μετρημένη ταχύτητα σε km/h
  source: str               # Πηγή μέτρησης (π.χ. 'stereo', 'radar')
  raw_data: dict            # Ακατέργαστα δεδομένα από αισθητήρες
  system_mode: Optional[str] = None  # Κατάσταση λειτουργίας συστήματος
```

Ο decorator @dataclass μετατρέπει μια απλή κλάση σε μια ειδική κλάση δεδομένων που δημιουργεί αυτόματα τον constructor και άλλες βασικές μεθόδους. Στο παράδειγμα, η κλάση SpeedEvent καταγράφει γεγονότα ταχύτητας με πληροφορίες όπως η χρονική στιγμή (timestamp), η πηγή των δεδομένων (source που μπορεί να είναι GPS ή IMU), και την ταχύτητα σε km/h. Τα type hints όπως float, str και Optional[str] δηλώνουν τον αναμενόμενο τύπο κάθε πεδίου, βοηθώντας τόσο στην τεκμηρίωση όσο και στον εντοπισμό σφαλμάτων κατά την ανάπτυξη. Το Optional[str] = None δηλώνει ότι το πεδίο system_mode είναι προαιρετικό με προεπιλεγμένη τιμή None.

Η Python υποστηρίζει multithreading μέσω της βιβλιοθήκης threading, επιτρέποντας παράλληλη εκτέλεση διαφορετικών τμημάτων κώδικα [42]:

```
# main.py
import threading

# Δημιουργία pipeline για επεξεργασία δεδομένων ταχύτητας
speed_pipeline = SpeedPipeline(shared_interface=shared)

# Εκκίνηση pipeline σε ξεχωριστό thread για παράλληλη εκτέλεση
speed_thread = threading.Thread(
    target=speed_pipeline.run,
    daemon=True,              # Thread τερματίζει αυτόματα με το κλείσιμο του προγράμματος
    name="SpeedPipeline"      # Όνομα thread για debugging
)
speed_thread.start()
```

Τα threads επιτρέπουν την ταυτόχρονη εκτέλεση διαφορετικών τμημάτων του προγράμματος, απαραίτητο για real-time συστήματα που πρέπει να επεξεργάζονται εικόνες ενώ παράλληλα διαβάζουν δεδομένα από αισθητήρες. Η παράμετρος target=speed_pipeline.run ορίζει τη μέθοδο που θα εκτελεστεί στο νέο thread, στην περίπτωσή αυτή, τη μέθοδο run() του αντικειμένου speed_pipeline. Το daemon=True είναι κρίσιμη ρύθμιση που σημαίνει ότι το thread θα τερματιστεί αυτόματα όταν τερματίσει το κύριο πρόγραμμα, αποφεύγοντας το πρόβλημα των "zombie" processes που συνεχίζουν να τρέχουν στο παρασκήνιο. Το όνομα "SpeedPipeline" εμφανίζεται στα εργαλεία debugging και στα logs, διευκολύνοντας τον εντοπισμό προβλημάτων.

Για ασφαλή πρόσβαση σε κοινόχρηστα δεδομένα από πολλαπλά threads, χρησιμοποιούνται locks [43]:

```
# pipeline_logger.py

import threading
from dataclasses import dataclass

class PipelineLogger:
  def __init__(self):
```

```
    # Αρχικοποίηση reentrant lock για thread-safe λειτουργίες
    self._lock = threading.RLock()
    # Λίστα αποθήκευσης συμβάντων ανίχνευσης
    self.detection_events = []
    # Στατιστικά στοιχεία συστήματος
    self.stats = {'total_detections': 0}

def log_detection_event(self, event_type: str, object_class: Optional[int] = None):
    # Thread-safe καταγραφή συμβάντος με context manager
    with self._lock:
        # Δημιουργία νέου συμβάντος ανίχνευσης
        event = DetectionEvent(
            timestamp=time.time(),
            event=event_type,
            object_class=object_class
        )
        # Προσθήκη συμβάντος στο ιστορικό
        self.detection_events.append(event)
        # Ενημέρωση στατιστικών
        self.stats['total_detections'] += 1
```

Το RLock (Reentrant Lock) είναι ένας μηχανισμός συγχρονισμού που προστατεύει τα δεδομένα από ταυτόχρονη πρόσβαση πολλαπλών threads, αποτρέποντας την καταστροφή δεδομένων (data corruption). Σε αντίθεση με το απλό Lock, το RLock μπορεί να αποκτηθεί πολλές φορές από το ίδιο thread, χρήσιμο όταν μια μέθοδος που έχει ήδη το lock καλεί άλλη μέθοδο που επίσης το χρειάζεται. Το context manager (with statement) εγγυάται ότι το lock θα αποκτηθεί στην αρχή του μπλοκ και θα απελευθερωθεί στο τέλος, ακόμα και αν προκύψει exception, αποφεύγοντας deadlocks. Στο παράδειγμα, η λίστα detection_events και το dictionary stats προστατεύονται από ταυτόχρονες εγγραφές όταν πολλαπλά threads (επεξεργασία εικόνας, αισθητήρες) καλούν την log_detection_event.

Η Python παρέχει έτοιμες λύσεις για κοινές ανάγκες μέσω της standard library [44]. Για παράδειγμα, η διαχείριση αρχείων JSON:

```
# pipeline_logger.py
import json
from dataclasses import asdict

def save_session_data(self):
    # Μετατροπή όλων των dataclass events σε λεξικά για JSON serialization
    events_dict = [asdict(event) for event in self.detection_events]

    # Αποθήκευση δεδομένων συνεδρίας σε αρχείο JSON με ευανάγνωστη μορφή
    with open(f"{self.session_id}_detections.json", 'w') as f:
        json.dump(events_dict, f, indent=2)
```

Η αποθήκευση δεδομένων σε μορφή JSON επιτρέπει την εύκολη ανάλυση και επεξεργασία των αποτελεσμάτων με άλλα εργαλεία. Η συνάρτηση asdict() από το module dataclasses μετατρέπει κάθε αντικείμενο τύπου dataclass σε ένα dictionary που μπορεί να σειριοποιηθεί σε JSON. Η list comprehension [asdict(event) for event in self.detection_events] επεξεργάζεται όλα τα events με μία γραμμή κώδικα, δημιουργώντας μια λίστα από dictionaries. Το json.dump() γράφει τα δεδομένα στο αρχείο, με την παράμετρο indent=2 να δημιουργεί ευανάγνωστη μορφοποίηση με εσοχές, διευκολύνοντας την ανάγνωση από άνθρωπο. Το context manager (with open...) εξασφαλίζει ότι το αρχείο θα κλείσει σωστά ακόμα και σε περίπτωση σφάλματος.

3.3.3 C++

Η γλώσσα προγραμματισμού Arduino βασίζεται στη C++ με απλοποιήσεις και προσθήκες που διευκολύνουν τον προγραμματισμό μικροελεγκτών [45]. Το Arduino IDE παρέχει ένα framework που αποκρύπτει τις πολυπλοκότητες του low-level προγραμματισμού, επιτρέποντας την εύκολη διαχείριση I/O pins, σειριακής επικοινωνίας και περιφερειακών. Η γλώσσα διατηρεί τη δύναμη και την αποδοτικότητα της C++ ενώ προσφέρει έτοιμες συναρτήσεις για κοινές λειτουργίες embedded systems.

Το Arduino στο σύστημά διαχειρίζεται το LCD display 16x2 και τα LED για την εμφάνιση πληροφοριών στον οδηγό. Επικοινωνεί με το Raspberry Pi μέσω σειριακής θύρας, λαμβάνοντας εντολές με απλό πρωτόκολλο κειμένου και ενημερώνοντας το display ανάλογα. Η χρήση του Arduino αντί για απευθείας σύνδεση στο Pi εξασφαλίζει σταθερό timing για το display και απομονώνει το κύριο σύστημα από καθυστερήσεις.

```
// Arduino Display Controller - Βασική δομή

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Ρυθμίσεις I2C LCD οθόνης
#define LCD_I2C_ADDR 0x27        // Διεύθυνση I2C της οθόνης
#define LCD_COLS     16          // Αριθμός στηλών οθόνης
#define LCD_ROWS     2           // Αριθμός γραμμών οθόνης
LiquidCrystal_I2C lcd(LCD_I2C_ADDR, LCD_COLS, LCD_ROWS);

// Ορισμός pins για RGB LED (απαιτούν PWM)
#define RED_PIN     3            // Pin για κόκκινο χρώμα
#define GREEN_PIN   5            // Pin για πράσινο χρώμα
#define BLUE_PIN    6            // Pin για μπλε χρώμα

void setup() {
 // Ρύθμιση σειριακής επικοινωνίας με υψηλή ταχύτητα
 Serial.begin(115200);
 Serial.setTimeout(25);         // Χαμηλό timeout για αποφυγή καθυστερήσεων
```

```
 // Αρχικοποίηση I2C διαύλου και LCD οθόνης
 Wire.begin();
 lcd.init();
 lcd.backlight();              // Ενεργοποίηση οπίσθιου φωτισμού
 lcd.clear();                  // Καθαρισμός οθόνης

 // Ρύθμιση LED pins ως εξόδους
 pinMode(RED_PIN, OUTPUT);
 pinMode(GREEN_PIN, OUTPUT);
 pinMode(BLUE_PIN, OUTPUT);

 // Εμφάνιση μηνύματος εκκίνησης
 showLCD("System", "Ready");
}
```

Η LiquidCrystal_I2C βιβλιοθήκη χρησιμοποιεί το I2C πρωτόκολλο για επικοινωνία με το LCD, μειώνοντας τον αριθμό των απαιτούμενων pins από 6 σε 2 (SDA, SCL). Το Wire.begin() αρχικοποιεί το I2C bus ως master. Η Serial.begin(115200) ρυθμίζει τη σειριακή επικοινωνία στα 115200 bps για ελαχιστοποίηση της καθυστέρησης. Τα LED pins επιλέχθηκαν να είναι PWM-capable (3, 5, 6) για έλεγχο φωτεινότητας [46].

Για την επεξεργασία εντολών από το Raspberry Pi:

Ο parser χρησιμοποιεί απλό πρωτόκολλο κειμένου αντί για JSON για μείωση της υπολογιστικής επιβάρυνσης. Οι εντολές έχουν τη μορφή VERB:PAYLOAD, όπου το VERB καθορίζει τη λειτουργία (LCD, LED). Για το LCD, το payload χωρίζεται με '|' σε δύο γραμμές. Η showLCD() εξασφαλίζει ότι το κείμενο δεν θα υπερβεί τα 16 χαρακτήρες ανά γραμμή, αποφεύγοντας artifacts στην οθόνη [47].

Για τον έλεγχο των LED με animations:

```
// Μηχανή καταστάσεων για LED animations
enum LedState : uint8_t {
 STATE_OFF = 0,          // LED απενεργοποιημένο
 STATE_GREEN,            // Σταθερό πράσινο
 STATE_RED,              // Σταθερό κόκκινο
 STATE_YELLOW_BLINK,     // Κίτρινο που αναβοσβήνει
 STATE_RED_BLINK,        // Κόκκινο που αναβοσβήνει
 STATE_BLUE_PULSE        // Μπλε με εφέ παλμού
};

// Τρέχουσα κατάσταση LED (volatile για ασφαλή πρόσβαση από interrupts)
volatile LedState currentLEDState = STATE_OFF;

// Ενημέρωση LED animations χωρίς blocking
void updateLedAnimator(unsigned long now) {
 switch (currentLEDState) {
  case STATE_YELLOW_BLINK: {
```

```
    // Έλεγχος χρόνου για επόμενη εναλλαγή
    if (now - lastBlinkTs >= BLINK_PERIOD_MS) {
      lastBlinkTs = now;
      blinkOn = !blinkOn;        // Εναλλαγή κατάστασης on/off
    }
    // Εφαρμογή κίτρινου ή σβηστού χρώματος
    if (blinkOn) setSolidColor(255, 255, 0);
    else         setSolidColor(0, 0, 0);
    break;
   }

   case STATE_BLUE_PULSE: {
    // Έλεγχος χρόνου για επόμενο βήμα παλμού
    if (now - lastPulseTs >= PULSE_INTERVAL_MS) {
      lastPulseTs = now;
      // Δημιουργία τριγωνικού κύματος 0→255→0
      int next = (int)pulseValue + (int)pulseDir * PULSE_STEP;
      // Αντιστροφή κατεύθυνσης στα όρια
      if (next >= 255) { next = 255; pulseDir = -1; }
      else if (next <= 0) { next = 0; pulseDir = 1; }
      pulseValue = (uint8_t)next;
    }
    // Εφαρμογή μπλε χρώματος με μεταβλητή ένταση
    setSolidColor(0, 0, pulseValue);
    break;
   }
  }
}

// Άμεσος έλεγχος RGB LED μέσω PWM
void setRGBRaw(uint8_t r, uint8_t g, uint8_t b) {
 analogWrite(RED_PIN,   r);      // Ρύθμιση έντασης κόκκινου
 analogWrite(GREEN_PIN, g);      // Ρύθμιση έντασης πράσινου
 analogWrite(BLUE_PIN,  b);      // Ρύθμιση έντασης μπλε
}
```

Η υλοποίηση χρησιμοποιεί state machine για non-blocking animations, επιτρέποντας στο Arduino να επεξεργάζεται εντολές ενώ εκτελούνται τα animations. Η analogWrite() χρησιμοποιεί hardware PWM για ομαλό έλεγχο φωτεινότητας με τιμές 0-255. Το "breathing" effect του BLUE_PULSE υλοποιείται με triangle wave που αυξομειώνει την τιμή PWM. Όλοι οι χρονισμοί βασίζονται σε millis() για αποφυγή blocking delays [48].

3.3.4 OpenCv

Η OpenCV είναι μια βιβλιοθήκη ανοιχτού κώδικα που περιέχει αλγορίθμους υπολογιστικής όρασης και επεξεργασίας εικόνας[49]. Αναπτύχθηκε αρχικά από την Intel το 1999 και σήμερα διατηρείται από το OpenCV Foundation, προσφέροντας πάνω από 2500 βελτιστοποιημένους αλγορίθμους[50]. Στο σύστημα χρησιμοποιήθηκε η έκδοση 4.8 που υποστηρίζει Python 11.

Η OpenCV χρησιμοποιείται για τη βαθμονόμηση της στερεοσκοπικής κάμερας με τη μέθοδο του Zhang [51].

```
# cal.py
import cv2 as cv
import numpy as np

# Ορισμός διαστάσεων σκακιέρας και ανίχνευση γωνιών
PATTERN_SIZE = (8, 6)  # Αριθμός εσωτερικών γωνιών (οριζόντια, κάθετα)
ok_l, corners_l = cv.findChessboardCornersSB(gray_l, PATTERN_SIZE)
ok_r, corners_r = cv.findChessboardCornersSB(gray_r, PATTERN_SIZE)

# Βελτιστοποίηση θέσης γωνιών με ακρίβεια sub-pixel
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 1e-5)
corners_l = cv.cornerSubPix(gray_l, corners_l, (11,11), (-1,-1), criteria)

# Εκτέλεση στερεοσκοπικής βαθμονόμησης για εξαγωγή παραμέτρων καμερών
ret_stereo, K_l, D_l, K_r, D_r, R, T, E, F = cv.stereoCalibrate(
  objpoints, imgpoints_l, imgpoints_r,     # 3D και 2D σημεία αναφοράς
  K_l, D_l, K_r, D_r, img_shape,           # Εσωτερικές παράμετροι και διαστάσεις
  flags=cv.CALIB_FIX_INTRINSIC             # Διατήρηση σταθερών εσωτερικών παραμέτρων
)
```

Η συνάρτηση findChessboardCornersSB() χρησιμοποιεί τον αλγόριθμο SaddlePoint για ανίχνευση γωνιών σκακιέρας. Η cornerSubPix() βελτιστοποιεί τη θέση των γωνιών σε ακρίβεια μικρότερη του pixel. Η stereoCalibrate() υλοποιεί τη μέθοδο του Zhang για τον υπολογισμό των εσωτερικών παραμέτρων (K), των συντελεστών παραμόρφωσης (D), και της σχετικής θέσης των καμερών (R, T). Η σημαία CALIB_FIX_INTRINSIC διατηρεί σταθερές τις εσωτερικές παραμέτρους που έχουν ήδη υπολογιστεί.

Για βελτίωση της ποιότητας εικόνας πριν την ανίχνευση αντικειμένων:

```
# detection.py
import cv2

def enhance_adaptive_local(self, img):
  # Μετατροπή από RGB σε YCrCb για ανεξάρτητη επεξεργασία φωτεινότητας
  ycrcb = cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
  y, cr, cb = cv2.split(ycrcb)
```

```
    # Εφαρμογή ισοστάθμισης ιστογράμματος για βελτίωση αντίθεσης
    y = cv2.equalizeHist(y)

    # Δημιουργία kernel για όξυνση ακμών (unsharp masking)
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]], dtype=np.float32)
    # Εφαρμογή φίλτρου όξυνσης στο κανάλι φωτεινότητας
    y = cv2.filter2D(y, -1, kernel)

    # Συνδυασμός καναλιών και επιστροφή σε RGB
    enhanced = cv2.merge([y, cr, cb])
    return cv2.cvtColor(enhanced, cv2.COLOR_YCrCb2RGB)
```

Η μετατροπή σε YCrCb επιτρέπει την επεξεργασία της φωτεινότητας (Y) χωρίς να επηρεάζονται τα χρώματα (Cr, Cb). Η equalizeHist() κατανέμει ομοιόμορφα τις τιμές φωτεινότητας βελτιώνοντας την αντίθεση. Το kernel που εφαρμόζεται με filter2D() είναι ένα Laplacian sharpening filter που ενισχύει τις ακμές των αντικειμένων. Αυτή η προεπεξεργασία βελτιώνει την ακρίβεια ανίχνευσης σε δύσκολες συνθήκες φωτισμού.

Για διόρθωση παραμόρφωσης και υπολογισμό βάθους από στερεοσκοπικές εικόνες [52]:

```
# stereo_depth_estimator.py
# Διόρθωση στερεοσκοπικών εικόνων με προϋπολογισμένους χάρτες αναδιάταξης
left_rect = cv2.remap(left_img, self.map_left1, self.map_left2,
                      cv2.INTER_LINEAR)
right_rect = cv2.remap(right_img, self.map_right1, self.map_right2,
                       cv2.INTER_LINEAR)

# Εξαγωγή template από αριστερή εικόνα και αναζήτηση στη δεξιά
template = left_rect[y:y+h, x:x+w]
result = cv2.matchTemplate(right_rect, template, cv2.TM_CCORR_NORMED)
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

# Υπολογισμός απόστασης βάθους από την παράλλαξη
disparity = abs(x - max_loc[0])          # Διαφορά θέσης μεταξύ εικόνων
if disparity > 0:
  # Εφαρμογή τύπου στερεοσκοπικού βάθους
  depth = (self.focal_length * self.baseline) / disparity
```

Η remap() εφαρμόζει γεωμετρικούς μετασχηματισμούς για να διορθώσει την παραμόρφωση των φακών και να ευθυγραμμίσει τις επιπολικές γραμμές. Η matchTemplate() με τη μέθοδο

TM_CCORR_NORMED βρίσκει την καλύτερη αντιστοίχιση ενός τμήματος της αριστερής εικόνας στη δεξιά. Ο υπολογισμός βάθους βασίζεται στον τύπο της στερεοσκοπικής γεωμετρίας (3.1).

3.3.5 YOLO

Το YOLO (You Only Look Once) είναι ένα σύστημα ανίχνευσης αντικειμένων που εισήχθη το 2016 από τους Redmon et al. και φέρνει επανάσταση στον τρόπο που γίνεται η ανίχνευση αντικειμένων σε εικόνες. Σε αντίθεση με προηγούμενες μεθόδους που χρησιμοποιούσαν sliding windows ή region proposals, το YOLO αντιμετωπίζει την ανίχνευση ως ένα ενιαίο πρόβλημα παλινδρόμησης, εξετάζοντας ολόκληρη την εικόνα μία φορά για να προβλέψει ταυτόχρονα τις θέσεις και τις κατηγορίες όλων των αντικειμένων [53]. Αυτή η προσέγγιση το καθιστά εξαιρετικά γρήγορο, κατάλληλο για εφαρμογές πραγματικού χρόνου.

Η έκδοση YOLOv8 που κυκλοφόρησε το 2023 από την Ultralytics αποτελεί την πιο πρόσφατη εξέλιξη της αρχιτεκτονικής YOLO [54]. Οι βασικές βελτιώσεις περιλαμβάνουν anchor-free detection που απλοποιεί το training, decoupled head για καλύτερη ακρίβεια στην ταξινόμηση και τον εντοπισμό, και νέο backbone βασισμένο στην αρχιτεκτονική CSPNet για αποδοτική εξαγωγή χαρακτηριστικών [55]. Το YOLOv8 διατίθεται σε πέντε μεγέθη (n, s, m, l, x) για διαφορετικές ανάγκες ταχύτητας και ακρίβειας. [56]

Στο σύστημά μας χρησιμοποιούνται δύο μοντέλα YOLOv8 βελτιστοποιημένα για το Google Coral Edge TPU:

```
# detection.py
from ultralytics import YOLO


class AlternatingDetector:
  def __init__(self, model_path, model_size=512, conf_thres=0.5):
    # Φόρτωση quantized μοντέλου βελτιστοποιημένου για Edge TPU
    self.model = YOLO(model_path, task="detect")
    self.conf_thres = conf_thres  # Ελάχιστο όριο εμπιστοσύνης ανίχνευσης
    self.model_size = model_size  # Διαστάσεις εισόδου νευρωνικού (512x512)


  def detect(self, roi_image):
    # Εκτέλεση inference στο TPU για γρήγορη ανίχνευση
    results = self.model(roi_image,
                imgsz=self.model_size,      # Μέγεθος εικόνας εισόδου
                conf=self.conf_thres,       # Κατώφλι εμπιστοσύνης
                device='tpu')[0]            # Χρήση TPU accelerator


    # Εξαγωγή και επεξεργασία ανιχνεύσεων
    for detection in results.boxes:
      xyxy = detection.xyxy.cpu().numpy()       # Συντεταγμένες πλαισίου
      cls = int(detection.cls.cpu().numpy())    # ID κατηγορίας αντικειμένου
      conf = float(detection.conf.cpu().numpy()) # Βαθμός εμπιστοσύνης
```

Η κλάση YOLO από την Ultralytics παρέχει υψηλού επιπέδου API για εύκολη χρήση του μοντέλου. Η παράμετρος task="detect" ορίζει ότι θα γίνει object detection (όχι segmentation ή pose estimation). Το imgsz=512 ορίζει το μέγεθος εισόδου του μοντέλου, ενώ το conf=0.5 σημαίνει ότι θα κρατηθούν μόνο ανιχνεύσεις με εμπιστοσύνη πάνω από 50%. Η παράμετρος device='tpu' δηλώνει χρήση του Edge TPU για hardware acceleration. Τα αποτελέσματα περιλαμβάνουν τις συντεταγμένες του bounding box (xyxy), την κλάση του αντικειμένου και το confidence score.

Για το tracking χρησιμοποιείται μικρότερο μοντέλο που εκτελείται για την συνεχή παρακολούθηση :

```
# tracking.py


from ultralytics import YOLO


class TrackingModule:
  def __init__(self, model_path_256, conf_thres=0.4):
    # Φόρτωση ελαφρύτερου μοντέλου 256x256 για ταχύτερο tracking
    self.model = YOLO(model_path_256, task="detect")
    self.conf_thres = conf_thres  # Χαμηλότερο κατώφλι για συνεχή παρακολούθηση


  def track(self, roi_image):
    # Αλλαγή μεγέθους εικόνας για εισαγωγή στο μοντέλο
    resized = cv2.resize(roi_image, (256, 256))


    # Εκτέλεση inference με χαλαρότερα κριτήρια ανίχνευσης
    res = self.model(resized, imgsz=256, conf=self.conf_thres)[0]


    # Επιλογή ανίχνευσης με υψηλότερη εμπιστοσύνη
    best_conf = 0.0
    best_box = None
    for xyxy, cls, conf in zip(res.boxes.xyxy.cpu().numpy(),
                               res.boxes.cls.cpu().numpy(),
                               res.boxes.conf.cpu().numpy()):
      # Ενημέρωση καλύτερης ανίχνευσης
      if conf > best_conf:
        best_conf = conf
        best_box = xyxy
        best_class = int(cls)
```

Το tracking μοντέλο είναι μικρότερο (256x256) για ταχύτερη εκτέλεση, καθώς εφαρμόζεται σε κάθε frame. Χρησιμοποιεί χαμηλότερο confidence threshold (0.4 αντί 0.5) γιατί αναζητά ήδη γνωστά αντικείμενα σε περιορισμένη περιοχή. Η λογική επιλογής της καλύτερης ανίχνευσης (highest confidence) εξασφαλίζει ότι θα διορθώνονται λάθη κλάσεων που έγιναν στην φάση του detection. Οι συντεταγμένες του bounding box μετατρέπονται πίσω στις διαστάσεις της αρχικής εικόνας μετά την ανίχνευση.

3.3.6 Σημαντικές Βιβλιοθήκες

3.3.6.1 NumPy

Το NumPy (Numerical Python) είναι η θεμελιώδης βιβλιοθήκη για επιστημονικούς υπολογισμούς στην Python, παρέχοντας υποστήριξη για μεγάλους πολυδιάστατους πίνακες και μαθηματικές συναρτήσεις υψηλής απόδοσης [57]. Αναπτύχθηκε ως συνέχεια του Numeric και του Numarray, ενοποιώντας τα καλύτερα χαρακτηριστικά τους. Το NumPy είναι γραμμένο σε C και Fortran, προσφέροντας απόδοση συγκρίσιμη με compiled γλώσσες για πράξεις πινάκων.

Στο σύστημά το NumPy χρησιμοποιείται εκτενώς για επεξεργασία δεδομένων αισθητήρων και υπολογισμούς στατιστικής. Παρέχει αποδοτικές συναρτήσεις για στατιστική ανάλυση όπως percentiles και μέσους όρους, απαραίτητες για τη βαθμονόμηση του IMU και την απόρριψη outliers.

```
# imu_gps_fusion.py - Απόρριψη outliers με IQR method
import numpy as np

# Μετατροπή σε NumPy array για βελτιστοποιημένες πράξεις
imu_array = np.array(self.imu_samples)

# Υπολογισμός τεταρτημορίων για μέθοδο IQR
q75, q25 = np.percentile(imu_array, [75, 25])
iqr = q75 - q25                          # Διατεταρτημοριακό εύρος

# Ορισμός ορίων και φιλτράρισμα ακραίων τιμών
lower_bound = q25 - 1.5 * iqr            # Κάτω όριο αποδεκτών τιμών
upper_bound = q75 + 1.5 * iqr            # Άνω όριο αποδεκτών τιμών
filtered_samples = imu_array[(imu_array >= lower_bound) &
                             (imu_array <= upper_bound)]

# Υπολογισμός στατιστικών μετρήσεων από καθαρά δεδομένα
self.imu_bias = np.mean(filtered_samples)    # Μέση τιμή (bias αισθητήρα)
std_dev = np.std(filtered_samples)           # Τυπική απόκλιση θορύβου
```

Η συνάρτηση np.array() μετατρέπει Python lists σε NumPy arrays που υποστηρίζουν vectorized operations. Η np.percentile() υπολογίζει τα percentiles του dataset, εδώ το 25ο και 75ο για την IQR μέθοδο. Το boolean indexing array[(condition)] επιλέγει μόνο τα στοιχεία που πληρούν τη συνθήκη, πολύ πιο γρήγορο από loops. Οι συναρτήσεις np.mean() και np.std() υπολογίζουν μέσο όρο και τυπική απόκλιση αποδοτικά.

3.3.6.2 Picamera2

Η Picamera2 είναι η επίσημη Python βιβλιοθήκη για τον έλεγχο των καμερών Raspberry Pi, αντικαθιστώντας την παλαιότερη Picamera [58]. Αναπτύχθηκε από το Raspberry Pi Foundation για να παρέχει πλήρη έλεγχο των Camera Module v1, v2 και v3 μέσω του libcamera framework. Υποστηρίζει προηγμένα χαρακτηριστικά όπως hardware encoding, multiple streams και zero-copy buffers.

Το σύστημα χρησιμοποιεί δύο κάμερες Picamera v3 για στερεοσκοπική όραση, με τη Picamera2 να παρέχει χαμηλού επιπέδου έλεγχο για συγχρονισμένη λήψη. Η διαμόρφωση με πολλαπλά buffers εξασφαλίζει ομαλή ροή εικόνων χωρίς frame drops ακόμα και σε υψηλό φόρτο επεξεργασίας.

```
# camera_utils.py - Αρχικοποίηση στερεοσκοπικών καμερών
from picamera2 import Picamera2


# Δημιουργία instances για στερεοσκοπικό σύστημα
cam_left = Picamera2()                  # Αριστερή κάμερα (προεπιλεγμένη port 0)
cam_right = Picamera2(camera_num=1)         # Δεξιά κάμερα (δεύτερη port 1)


# Ρύθμιση παραμέτρων για 1080p RGB streaming
config = cam_left.create_video_configuration(
  main={"size": (1920, 1080), "format": "RGB888"},  # Ανάλυση και μορφή χρώματος
  buffer_count=8                          # Αριθμός buffers για ομαλή λήψη
)
# Εφαρμογή ρυθμίσεων και εκκίνηση
cam_left.configure(config)
cam_left.start()


# Λήψη frame από κάμερα
frame = cam_left.capture_array("main")       # Επιστροφή ως numpy array για επεξεργασία
```

Η παράμετρος camera_num επιλέγει ποια κάμερα θα χρησιμοποιηθεί όταν υπάρχουν πολλές συνδεδεμένες. Η create_video_configuration() δημιουργεί configuration για video streaming με καθορισμένη ανάλυση και format. Το format "RGB888" δίνει 8 bits ανά χρωματικό κανάλι σε RGB διάταξη, συμβατό με OpenCV. Τα 8 buffers επιτρέπουν στην κάμερα να συνεχίσει τη λήψη ενώ γίνεται επεξεργασία σε προηγούμενα frames.

3.3.6.3 Threading

Το threading module είναι μέρος της standard library της Python και παρέχει τη δυνατότητα εκτέλεσης πολλαπλών threads ταυτόχρονα [59]. Κάθε thread είναι ένα ξεχωριστό νήμα εκτέλεσης που μοιράζεται τη μνήμη με τα άλλα threads του ίδιου process. Το threading είναι ιδανικό για I/O-bound operations όπως η ανάγνωση από αισθητήρες και η επικοινωνία με το Arduino.

Το σύστημα χρησιμοποιεί πολλαπλά threads για να διατηρεί real-time απόδοση: ένα για επεξεργασία εικόνας, ένα για ανάγνωση GPS/IMU, και ένα για επικοινωνία με το Arduino. Η χρήση daemon threads εξασφαλίζει καθαρό τερματισμό του προγράμματος χωρίς να μένουν "κρεμασμένα" threads.

```
# speed_pipeline.py
import threading


class SpeedPipeline:
  def __init__(self):
    # Event για συντονισμένο τερματισμό threads
```

```
        self.stop_evt = threading.Event()

        # Δημιουργία ανεξάρτητων readers για παράλληλη ανάγνωση
        self.gps_reader = GPSReader(
            callback=self.fusion.handle_gps,  # Callback για επεξεργασία GPS
            stop_evt=self.stop_evt
        )
        self.imu_reader = IMUReader(
            callback=self.fusion.handle_imu,  # Callback για επεξεργασία IMU
            stop_evt=self.stop_evt
        )

    def run(self):
        # Εκκίνηση παράλληλων threads ανάγνωσης
        self.gps_reader.start()          # Thread για συνεχή λήψη GPS
        self.imu_reader.start()          # Thread για συνεχή λήψη IMU

        # Κύριος βρόχος επεξεργασίας
        while not self.stop_evt.is_set():
            # Επεξεργασία συγχωνευμένων δεδομένων
            time.sleep(0.01)             # Μικρή παύση για αποφυγή CPU overload

    def stop(self):
        # Αποστολή σήματος τερματισμού σε όλα τα threads
        self.stop_evt.set()
```

Το threading.Event() είναι μηχανισμός συγχρονισμού που επιτρέπει σε threads να περιμένουν για ένα συμβάν. Η μέθοδος start() ξεκινά την εκτέλεση του thread στη μέθοδο run(). Το stop_evt.is_set() ελέγχει αν έχει δοθεί σήμα τερματισμού, επιτρέποντας ομαλό shutdown. Η χρήση callbacks επιτρέπει στα reader threads να στέλνουν δεδομένα απευθείας στο fusion module χωρίς ενδιάμεσες ουρές.

3.3.6.4 PySerial

Το PySerial είναι βιβλιοθήκη για σειριακή επικοινωνία που υποστηρίζει διάφορα λειτουργικά συστήματα και παρέχει Python interface για RS232, RS485 και άλλα σειριακά πρωτόκολλα [60]. Χρησιμοποιείται εκτενώς σε embedded systems για επικοινωνία με μικροελεγκτές, GPS modules και άλλες συσκευές. Υποστηρίζει χαρακτηριστικά όπως hardware/software flow control, διάφορα baud rates και custom timeouts.

Στο σύστημά μας το PySerial χρησιμοποιείται για δύο κρίσιμες λειτουργίες: την επικοινωνία με το Arduino για τον έλεγχο του LCD/LED display και την ανάγνωση δεδομένων από το GPS module. Η βιβλιοθήκη διαχειρίζεται αυτόματα τα low-level details της σειριακής επικοινωνίας όπως buffering και error handling.

```
# gps_reader.py
```

```
import serial


class GPSReader(threading.Thread):
  def __init__(self):
    # Ρύθμιση σειριακής επικοινωνίας με GPS module
    self.ser = serial.Serial(
      port='/dev/ttyAMA0',     # Hardware UART του Raspberry Pi
      baudrate=9600,           # Τυπική ταχύτητα GPS modules
      timeout=0.5              # Timeout για αποφυγή blocking
    )

  def run(self):
    # Συνεχής ανάγνωση δεδομένων GPS
    while not self.stop_evt.is_set():
      # Λήψη γραμμής NMEA sentence
      line = self.ser.readline()
      if line:
        # Μετατροπή bytes σε ASCII string
        nmea = line.decode('ascii', errors='ignore').strip()
        # Εντοπισμός και επεξεργασία VTG sentences για ταχύτητα
        if 'VTG' in nmea:
          self.parse_speed(nmea)

  def __del__(self):
    # Ασφαλές κλείσιμο σειριακής θύρας
    if self.ser:
      self.ser.close()
```

Η Serial() ανοίγει τη σειριακή θύρα με τις καθορισμένες παραμέτρους. Το '/dev/ttyAMA0' είναι η hardware UART του Raspberry Pi. Το timeout=0.5 σημαίνει ότι η readline() θα περιμένει μέχρι 0.5 δευτερόλεπτα για δεδομένα πριν επιστρέψει. Η decode('ascii', errors='ignore') μετατρέπει τα bytes σε string αγνοώντας μη-ASCII χαρακτήρες που μπορεί να προκύψουν από θόρυβο. Η Serial() ανοίγει τη σειριακή θύρα με τις καθορισμένες παραμέτρους. Το '/dev/ttyAMA0' είναι η hardware UART του Raspberry Pi. Το timeout=0.5 σημαίνει ότι η readline() θα περιμένει μέχρι 0.5 δευτερόλεπτα για δεδομένα πριν επιστρέψει. Η decode('ascii', errors='ignore') μετατρέπει τα bytes σε string αγνοώντας μη-ASCII χαρακτήρες που μπορεί να προκύψουν από θόρυβο.

3.3.6.5 Adafruit BNO055

Η βιβλιοθήκη Adafruit BNO055 παρέχει Python interface για τον αισθητήρα BNO055, έναν 9-DOF IMU με ενσωματωμένο sensor fusion processor [61]. Ο BNO055 συνδυάζει επιταχυνσιόμετρο, γυροσκόπιο και μαγνητόμετρο, παρέχοντας απευθείας quaternions, Euler angles και linear acceleration. Η βιβλιοθήκη απλοποιεί την επικοινωνία I2C και την ανάγνωση των δεδομένων.

Το σύστημα χρησιμοποιεί τον BNO055 για τη μέτρηση της γραμμικής επιτάχυνσης του οχήματος, απαραίτητη για τον υπολογισμό ταχύτητας όταν το GPS δεν έχει σήμα. Ο αισθητήρας ρυθμίζεται σε IMUPLUS mode που απενεργοποιεί το μαγνητόμετρο για αποφυγή παρεμβολών από το μεταλλικό σασί του οχήματος.

```
# imu_reader.py
import board
import busio
import adafruit_bno055

class IMUReader(threading.Thread):
  def __init__(self, callback, i2c_addr=0x28):
    # Δημιουργία I2C διαύλου και σύνδεση με BNO055
    i2c = busio.I2C(board.SCL, board.SDA)
    self.bno = adafruit_bno055.BNO055_I2C(i2c, address=i2c_addr)

    # Ρύθμιση σε IMUPLUS mode για αποφυγή μαγνητικών παρεμβολών
    self.bno.mode = adafruit_bno055.IMUPLUS_MODE

  def run(self):
    # Συνεχής ανάγνωση δεδομένων IMU
    while not self.stop_evt.is_set():
      # Λήψη γραμμικής επιτάχυνσης χωρίς βαρύτητα (m/s²)
      accel = self.bno.linear_acceleration
      if accel and accel[0] is not None:
        # Εξαγωγή επιτάχυνσης άξονα X (εμπρός-πίσω κίνηση)
        forward_accel = accel[0]
        # Κλήση callback για επεξεργασία
        self.callback(forward_accel)

      # Δειγματοληψία στα 100Hz
      time.sleep(0.01)
```

Η busio.I2C() δημιουργεί I2C bus χρησιμοποιώντας τα default pins του Raspberry Pi. Η BNO055_I2C() αρχικοποιεί τον αισθητήρα στη διεύθυνση 0x28 (προεπιλογή). Το IMUPLUS_MODE χρησιμοποιεί μόνο accelerometer και gyroscope για υπολογισμό προσανατολισμού, ιδανικό για οχήματα. Η linear_acceleration επιστρέφει την επιτάχυνση χωρίς τη βαρύτητα σε m/s², με το accel[0] να είναι ο X άξονας (εμπρός-πίσω κίνηση).

3.4 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκαν οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη του συστήματος στερεοσκοπικής όρασης. Η Python αποτέλεσε την κύρια γλώσσα ανάπτυξης, προσφέροντας τον απαραίτητο συνδυασμό ευκολίας προγραμματισμού και υπολογιστικής απόδοσης

μέσω των C++ bindings των βιβλιοθηκών. Η C++ χρησιμοποιήθηκε για τον προγραμματισμό του Arduino, εξασφαλίζοντας deterministic execution για τον έλεγχο των περιφερειακών.

Το OpenCV και το Ultralytics YOLO αποτέλεσαν τους πυλώνες του computer vision subsystem, παρέχοντας αλγορίθμους για stereo rectification, depth estimation και object detection. Η υποστήριξη για hardware acceleration μέσω NEON instructions και Edge TPU ήταν κρίσιμη για την επίτευξη real-time επεξεργασίας στο Raspberry Pi 5.

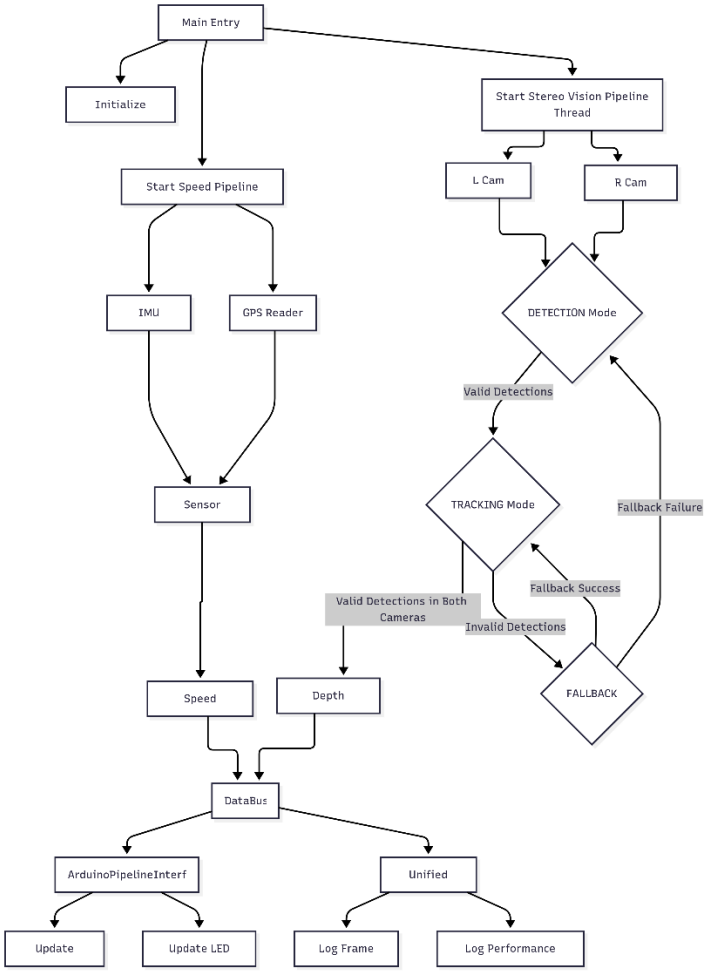
Η modular αρχιτεκτονική που επιτεύχθηκε μέσω της χρήσης αυτών των τεχνολογιών επιτρέπει μελλοντικές επεκτάσεις χωρίς σημαντικές αλλαγές στο υπάρχον σύστημα. Στο επόμενο κεφάλαιο θα παρουσιαστεί η πρακτική υλοποίηση του συστήματος, όπου οι τεχνολογίες που περιγράφηκαν συνδυάζονται για τη δημιουργία του ολοκληρωμένου συστήματος.

Κεφάλαιο 4ο: Υλοποίηση και Περιγραφή του συστήματος

4.1 Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζεται η υλοποίηση του συστήματος αναγνώρισης οδικών σημάτων με στερεοσκοπική όραση, από τη βαθμονόμηση του hardware μέχρι την ολοκληρωμένη λειτουργία σε πραγματικές συνθήκες. Το σύστημα που αναπτύχθηκε συνδυάζει πολλαπλές τεχνολογίες και υποσυστήματα που συνεργάζονται για την επίτευξη του τελικού στόχου: την αξιόπιστη ανίχνευση πινακίδων ορίου ταχύτητας και την έγκαιρη προειδοποίηση του οδηγού.

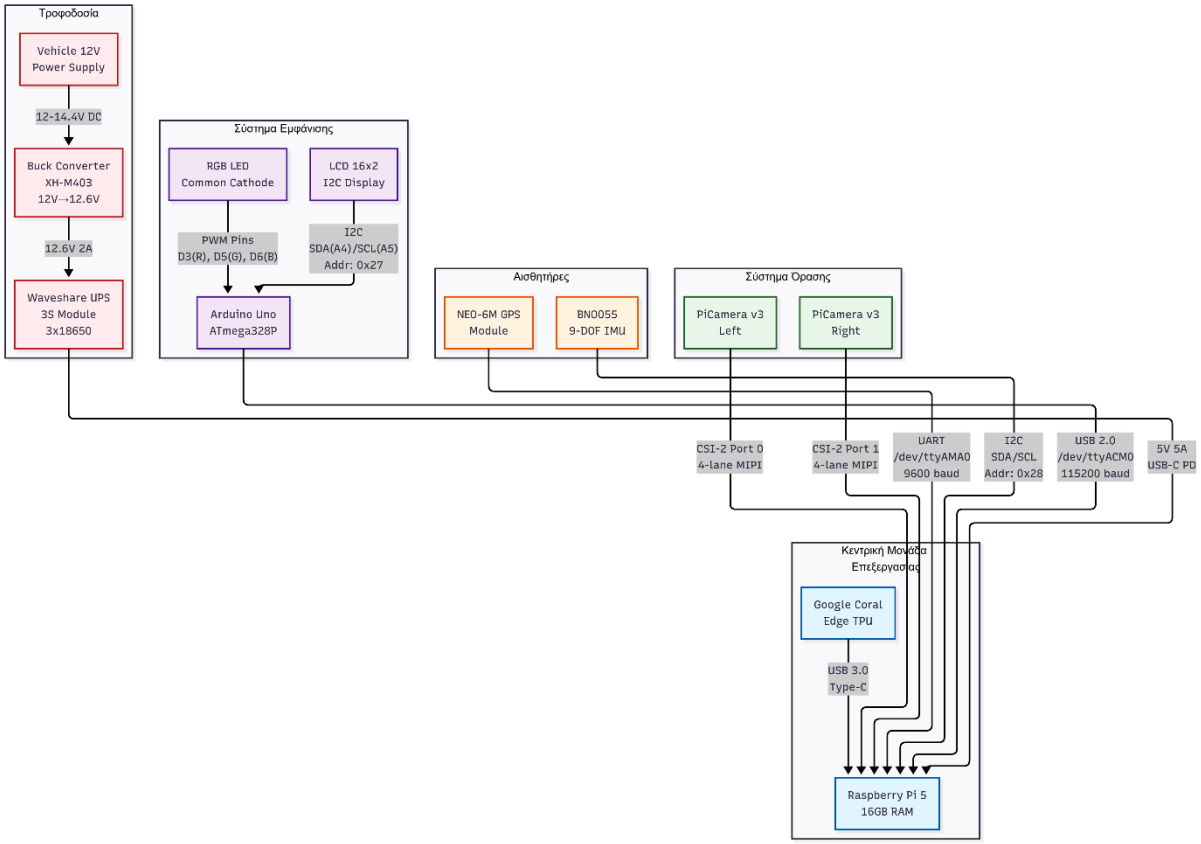
4.1.1 Συνολική Λειτουργία του Συστήματος



Εικόνα 4.1: Διάγραμμα ροής Συστήματος

Το διάγραμμα παρουσιάζει τη συνολική λογική ροή του συστήματος από την εκκίνηση μέχρι τη λειτουργία, δείχνοντας πώς τα τρία κύρια pipelines (Vision, Speed, DataBus) λειτουργούν παράλληλα και επικοινωνούν μεταξύ τους. Το Vision Pipeline εναλλάσσεται μεταξύ Detection (αναζήτηση πινακίδων) και Tracking (παρακολούθηση), το Speed Pipeline συνδυάζει δεδομένα GPS/IMU για εκτίμηση ταχύτητας, και το DataBus συντονίζει την επικοινωνία όλων των υποσυστημάτων με το Arduino που παρέχει την τελική ανάδραση στον οδηγό. Η απλοποιημένη μορφή επιτρέπει την

κατανόηση της συνολικής λειτουργίας χωρίς να χάνεται σε τεχνικές λεπτομέρειες που αναλύονται σε επόμενες ενότητες



Εικόνα 4.2: Διαγραμμα φυσικης διασυνδεσης υλικου

Το διάγραμμα απεικονίζει τη φυσική διασύνδεση όλων των components του συστήματος, με το Raspberry Pi 5 στο κέντρο να συνδέεται με τις δύο κάμερες μέσω CSI ports, τους αισθητήρες GPS/IMU μέσω UART και I2C αντίστοιχα, το Google Coral Edge TPU μέσω USB 3.0 για hardware acceleration, και το Arduino μέσω USB για τον έλεγχο του display. Η τροφοδοσία παρέχεται από το όχημα μέσω buck converter και UPS module που εξασφαλίζει σταθερά 5V στο σύστημα και προστασία από διακοπές τάσης. Κάθε σύνδεση περιλαμβάνει το πρωτόκολλο επικοινωνίας και τις παραμέτρους (baud rate, διευθύνσεις I2C) για εύκολη αναφορά κατά την υλοποίηση.

4.1.2 Δομή του Κεφαλαίου

Το κεφάλαιο οργανώνεται ως εξής: Πρώτα παρουσιάζεται η διαδικασία βαθμονόμησης του στερεοσκοπικού συστήματος (4.2), που είναι απαραίτητη για τον ακριβή υπολογισμό αποστάσεων. Στη συνέχεια αναλύεται η εκπαίδευση των μοντέλων αναγνώρισης (4.3) με τη δημιουργία custom dataset για ελληνικές πινακίδες. Ακολουθεί η περιγραφή της συνολικής αρχιτεκτονικής του συστήματος (4.4), συμπεριλαμβανομένων των design patterns που χρησιμοποιήθηκαν. Τέλος, αναλύονται λεπτομερώς τα τρία κύρια υποσυστήματα: το Speed Pipeline (4.5) για την εκτίμηση ταχύτητας, το Stereo Vision Pipeline (4.6) για την ανίχνευση πινακίδων, και το Arduino Display System (4.7) για την οπτική ανάδραση.

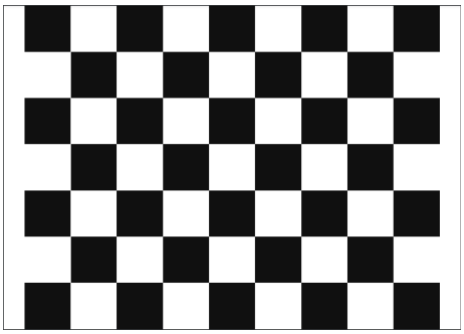
4.2 Βαθμονόμηση Στερεοσκοπικου Συστηματος

Η ακριβής βαθμονόμηση του στερεοσκοπικού συστήματος αποτελεί τον ακρογωνιαίο λίθο για την αξιόπιστη λειτουργία του συστήματος εκτίμησης βάθους. Στην περίπτωσή μας, όπου στοχεύουμε στην ανίχνευση πινακίδων κυκλοφορίας σε αποστάσεις από 0.5 έως 30 μέτρα, η ακρίβεια της βαθμονόμησης καθορίζει άμεσα την ικανότητα του συστήματος να προειδοποιεί έγκαιρα τον οδηγό. Το σύστημά μας χρησιμοποιεί δύο κάμερες PiCamera v3 τοποθετημένες σε οριζόντια διάταξη με απόσταση baseline περίπου 100mm, μια διάταξη που προσφέρει καλή ισορροπία μεταξύ ακρίβειας βάθους και πρακτικότητας εγκατάστασης στο όχημα.

Η διαδικασία βαθμονόμησης που αναπτύχθηκε αποτελείται από δύο διακριτές αλλά αλληλένδετες φάσεις: τη συλλογή υψηλής ποιότητας εικόνων βαθμονόμησης και τον υπολογισμό των απαραίτητων παραμέτρων για τη διόρθωση των οπτικών παραμορφώσεων και την ευθυγράμμιση των στερεοσκοπικών εικόνων.

4.2.1 Συλλογή Εικόνων Βαθμονόμησης

Το πρώτο και ίσως πιο κρίσιμο βήμα στη διαδικασία βαθμονόμησης είναι η συλλογή ενός επαρκούς αριθμού υψηλής ποιότητας στερεοσκοπικών ζευγών. Για το σκοπό αυτό, χρησιμοποιούμε μια τυπωμένη σκακιέρα με 8×6 εσωτερικές γωνίες και τετράγωνα μεγέθους 30mm. Η επιλογή αυτών των διαστάσεων δεν είναι τυχαία, προσφέρουν αρκετά σημεία αναφοράς για ακριβή βαθμονόμηση, ενώ το μέγεθος της σκακιέρας παραμένει πρακτικό για χειρισμό κατά τη λήψη από διάφορες γωνίες και αποστάσεις. Για τη διαδικασία βαθμονόμησης χρησιμοποιήθηκε τυπωμένη σκακιέρα με 8×6 εσωτερικές γωνίες και τετράγωνα μεγέθους 30mm, η οποία λήφθηκε από τη συλλογή προτύπων βαθμονόμησης του Jones [62]



Εικόνα 4.3: Σκακιερα Βαθμονομησης

[https://markhedleyjones.com/media/calibration-checkerboard-collection/Checkerboard-A4-30mm-8x6.svg]

```
# cap.py
# Δημιουργία δομής φακέλων για αποθήκευση εικόνων
ROOT      = Path("images")
LEFT_DIR  = ROOT / "imagesLeft"            # Φάκελος αριστερής κάμερας
RIGHT_DIR = ROOT / "imagesRight"           # Φάκελος δεξιάς κάμερας

# Δημιουργία φακέλων με όλους τους ενδιάμεσους
for d in (LEFT_DIR, RIGHT_DIR):
  d.mkdir(parents=True, exist_ok=True)
```

```
# Εύρεση επόμενου διαθέσιμου αριθμού για συνεχή αρίθμηση
def next_idx(folder: Path) -> int:
    # Αναζήτηση υπαρχόντων αρχείων με pattern img_*_*
    files = sorted(folder.glob("img_*_*"))
    # Εξαγωγή τελευταίου αριθμού και αύξηση κατά 1
    return int(files[-1].stem.split("_")[1]) + 1 if files else 0

# Υπολογισμός αρχικού index για νέες λήψεις
idx = next_idx(LEFT_DIR)
```

Το εργαλείο cap.py σχεδιάστηκε ειδικά για να απλοποιήσει και να αυτοματοποιήσει τη διαδικασία συλλογής εικόνων, αντιμετωπίζοντας πρακτικά προβλήματα που συναντώνται συχνά στη διαδικασία. Ένα σημαντικό χαρακτηριστικό του εργαλείου είναι η ικανότητα συνέχισης από το σημείο διακοπής. Αν για οποιονδήποτε λόγο χρειαστεί να διακοπεί η διαδικασία λήψης, το script θα εντοπίσει αυτόματα τον τελευταίο αριθμό εικόνας και θα συνεχίσει την αρίθμηση από εκεί. Αυτό επιτρέπει τη σταδιακή συλλογή εικόνων σε διαφορετικές συνθήκες φωτισμού ή με διαλείμματα για έλεγχο της ποιότητας.

Η κύρια λειτουργία του εργαλείου βασίζεται σε έναν απλό αλλά αποτελεσματικό βρόχο που εμφανίζει ζωντανή προεπισκόπηση και από τις δύο κάμερες ταυτόχρονα:

```
# Κύριος βρόχος λήψης στερεοσκοπικών εικόνων
while True:
    # Ταυτόχρονη λήψη από αριστερή και δεξιά κάμερα
    frame_l = pi_left.capture_array("main")  # Numpy array (H,W,3) RGB
    frame_r = pi_right.capture_array("main")

    # Δημιουργία προεπισκόπησης με παράθεση εικόνων
    preview = cv.hconcat([frame_l, frame_r])
    cv.imshow("Stereo preview", preview)
    key = cv.waitKey(1) & 0xFF

    # Χειρισμός πληκτρολογίου
    if key == 32:  # SPACE - Αποθήκευση ζεύγους εικόνων
        # Δημιουργία ονομάτων αρχείων με αύξοντα αριθμό
        fname_l = LEFT_DIR  / f"img_{idx:03d}_left.jpg"
        fname_r = RIGHT_DIR / f"img_{idx:03d}_right.jpg"
        # Αποθήκευση με υψηλή ποιότητα JPEG (95%)
        cv.imwrite(str(fname_l), frame_l, [int(cv.IMWRITE_JPEG_QUALITY), 95])
        cv.imwrite(str(fname_r), frame_r, [int(cv.IMWRITE_JPEG_QUALITY), 95])
        print(f"Αποθηκεύτηκε ζεύγος {idx:03d}")
        idx += 1
    elif key in (27, ord('q')):  # ESC ή 'q' - Έξοδος
```

```
    break
```

Η side-by-side προεπισκόπηση είναι ιδιαίτερα χρήσιμη καθώς επιτρέπει στον χειριστή να ελέγχει ότι η σκακιέρα είναι πλήρως ορατή και στις δύο κάμερες πριν από κάθε λήψη. Αυτό ελαχιστοποιεί τον αριθμό των άχρηστων εικόνων και εξασφαλίζει ότι κάθε αποθηκευμένο ζεύγος θα είναι χρήσιμο για τη βαθμονόμηση.

4.2.2 Υπολογισμός Παραμέτρων Βαθμονόμησης

Αφού συλλεχθεί επαρκής αριθμός εικόνων, το script cal.py αναλαμβάνει την επεξεργασία τους για τον υπολογισμό των intrinsic παραμέτρων κάθε κάμερας (εστιακή απόσταση, κέντρο προβολής, συντελεστές παραμόρφωσης) και των extrinsic παραμέτρων που περιγράφουν τη σχετική θέση και προσανατολισμό μεταξύ των δύο καμερών.

Οι βασικές παράμετροι του συστήματος ορίζονται στην αρχή του script:

```
# Ρυθμίσεις σκακιέρας
PATTERN_SIZE   = (8, 6)   # (στήλες, γραμμές) εσωτερικές γωνίες
SQUARE_MM      = 30.0     # μέγεθος τετραγώνου σε mm


# ΒΕΛΤΙΣΤΟ ALPHA ΓΙΑ PICAM V3
ALPHA_VALUE = 0.5 # Ρύθμιση μεταξύ 0.5-0.7
```

Η παράμετρος ALPHA_VALUE είναι ιδιαίτερα σημαντική και θα συζητηθεί εκτενώς παρακάτω. Η τιμή 0.5 αποδείχθηκε βέλτιστη για το συγκεκριμένο hardware μετά από εκτενείς δοκιμές.

Κατά την ανάπτυξη του συστήματος βαθμονόμησης, παρουσιάστηκε ένα σημαντικό πρόβλημα που δεν είναι άμεσα εμφανές: η συνάρτηση findChessboardCorners του OpenCV δεν εγγυάται συνεπή προσανατολισμό των ανιχνευμένων γωνιών. Αυτό σημαίνει ότι για την ίδια εικόνα, η "πρώτη" γωνία μπορεί άλλοτε να είναι η πάνω αριστερή και άλλοτε η κάτω δεξιά, ανάλογα με εσωτερικούς αλγορίθμους του OpenCV.

Αυτή η ασυνέπεια οδηγεί σε καταστροφικά αποτελέσματα κατά τη στερεοσκοπική βαθμονόμηση, καθώς το σύστημα προσπαθεί να αντιστοιχίσει γωνίες που στην πραγματικότητα αναφέρονται σε διαφορετικά σημεία της σκακιέρας. Το αποτέλεσμα είναι πολύ υψηλά σφάλματα βαθμονόμησης (RMS > 1.0) και εντελώς λανθασμένοι χάρτες διόρθωσης.

Η λύση που αναπτύχθηκε είναι η συνάρτηση ensure_corner_consistency:

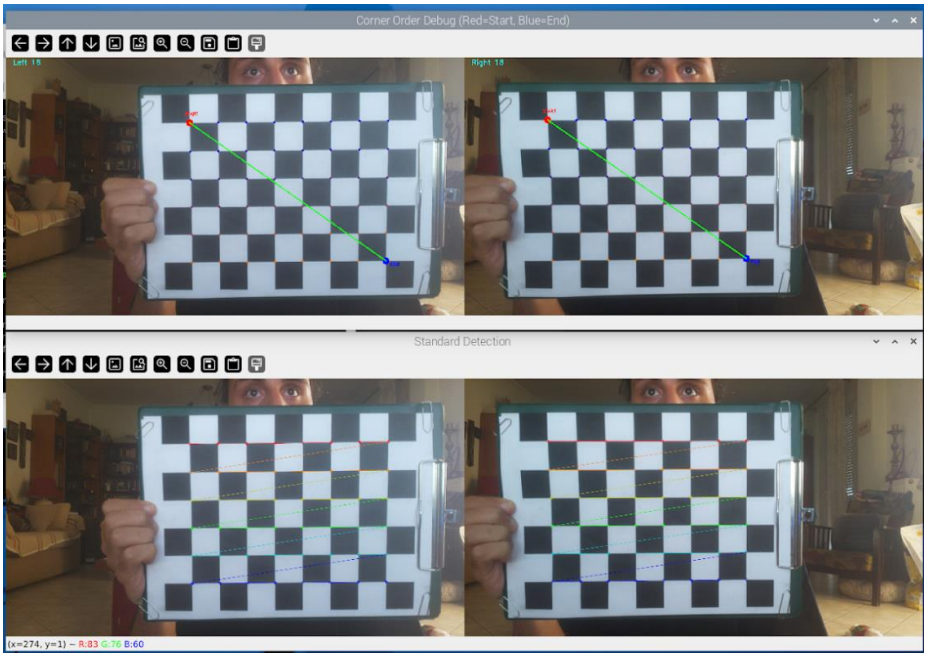
```
def ensure_corner_consistency(corners, pattern_size, reference_corners=None):
    cols, rows = pattern_size
    corners_2d = corners.reshape(rows, cols, 2)


    if reference_corners is None:
        # Χωρίς reference, βεβαιωνόμαστε ότι:
        # - Η πρώτη γωνία (0,0) είναι πάνω-αριστερά
        # - Η τελευταία γωνία είναι κάτω-δεξιά


        # Έλεγχος κατεύθυνσης Y (πρέπει να αυξάνεται από πάνω προς τα κάτω)
```

```
    if corners_2d[0, 0, 1] > corners_2d[-1, 0, 1]:
        corners_2d = corners_2d[::-1, :, :]  # Αναστροφή γραμμών

    # Έλεγχος κατεύθυνσης X (πρέπει να αυξάνεται από αριστερά προς δεξιά)
    if corners_2d[0, 0, 0] > corners_2d[0, -1, 0]:
        corners_2d = corners_2d[:, ::-1, :]  # Αναστροφή στηλών
else:
    # Ευθυγράμμιση με reference μέσω ελαχιστοποίησης απόστασης
    ref_2d = reference_corners.reshape(rows, cols, 2)

    # Δοκιμάζουμε όλους τους πιθανούς προσανατολισμούς
    orientations = [
        (False, False),  # Κανονικός
        (True, False),   # Αναστροφή γραμμών
        (False, True),   # Αναστροφή στηλών
        (True, True)     # Αναστροφή και των δύο
    ]

    min_dist = float('inf')
    best_corners = corners_2d.copy()

    for flip_rows, flip_cols in orientations:
        test = corners_2d.copy()
        if flip_rows:
            test = test[::-1, :, :]
        if flip_cols:
            test = test[:, ::-1, :]

        # Υπολογίζουμε απόσταση πρώτης γωνίας
        dist = np.sum((test[0, 0] - ref_2d[0, 0])**2)

        if dist < min_dist:
            min_dist = dist
            best_corners = test

    corners_2d = best_corners
return corners_2d.reshape(-1, 1, 2)
```

Ο αλγόριθμος λειτουργεί σε δύο καταστάσεις:

1. Αρχική ανίχνευση: Όταν δεν υπάρχει reference, εξασφαλίζει ότι οι γωνίες ακολουθούν την αναμενόμενη διάταξη με βάση τις συντεταγμένες τους.
2. Συνεπής ανίχνευση: Όταν υπάρχει reference από προηγούμενη εικόνα, δοκιμάζει όλους τους πιθανούς προσανατολισμούς και επιλέγει αυτόν που ελαχιστοποιεί την απόσταση από το reference.

Αυτή η προσέγγιση εξασφαλίζει ότι όλες οι γωνίες σε όλες τις εικόνες έχουν συνεπή αρίθμηση, επιτρέποντας στον αλγόριθμο βαθμονόμησης να λειτουργήσει σωστά.



Εικόνα 4.4: Έλεγχος Γωνιών κατά την βαθμονόμηση

Στην εικόνα 4.4 φαίνεται η λειτουργία για τον έλεγχο προσανατολισμού , οπού η πρώτη γωνιά θεωρείται η πρώτη εσωτερική πανω δεξιά ενώ από κάτω είναι το αποτέλεσμα του findChessboardCorners γραφικά.

4.2.3 Στερεοσκοπική Βαθμονόμηση και Διόρθωση

Αφού εξασφαλιστεί η συνέπεια των ανιχνευμένων γωνιών, το σύστημα προχωρά στην ατομική βαθμονόμηση κάθε κάμερας. Αυτό το βήμα υπολογίζει τα intrinsic χαρακτηριστικά κάθε κάμερας ξεχωριστά:

```
# Εκτέλεση ξεχωριστής βαθμονόμησης για κάθε κάμερα
print("  Βαθμονόμηση αριστερής κάμερας...")
ret_l, K_l, D_l, rvecs_l, tvecs_l = cv.calibrateCamera(
   objpoints, imgpoints_l, img_shape, None, None,
   flags=cv.CALIB_FIX_K4|cv.CALIB_FIX_K5    # Σταθεροποίηση υψηλών συντελεστών παραμόρφωσης
)
print(f"  Σφάλμα RMS αριστερής: {ret_l:.4f}")


print("  Βαθμονόμηση δεξιάς κάμερας...")
```

```
ret_r, K_r, D_r, rvecs_r, tvecs_r = cv.calibrateCamera(
  objpoints, imgpoints_r, img_shape, None, None,
  flags=cv.CALIB_FIX_K4|cv.CALIB_FIX_K5   # Σταθεροποίηση υψηλών συντελεστών παραμόρφωσης
)
print(f"  Σφάλμα RMS δεξιάς: {ret_r:.4f}")
```

Τα flags CALIB_FIX_K4|CALIB_FIX_K5 χρησιμοποιούνται για να περιορίσουν το μοντέλο παραμόρφωσης, αποφεύγοντας overfitting σε υψηλότερης τάξης συντελεστές που μπορούν να προκαλέσουν αστάθεια. Το RMS error κάθε κάμερας πρέπει ιδανικά να είναι κάτω από 0.5 pixels για καλή βαθμονόμηση.

Το επόμενο και πιο κρίσιμο βήμα είναι η στερεοσκοπική βαθμονόμηση, που υπολογίζει τη σχετική θέση και προσανατολισμό μεταξύ των δύο καμερών:

```
# Εκτέλεση στερεοσκοπικής βαθμονόμησης
print("  Εκτέλεση στερεοσκοπικής βαθμονόμησης...")
ret_stereo, K_l_s, D_l_s, K_r_s, D_r_s, R, T, E, F = cv.stereoCalibrate(
  objpoints, imgpoints_l, imgpoints_r,    # 3D και 2D σημεία αναφοράς
  K_l, D_l, K_r, D_r, img_shape,          # Εσωτερικές παράμετροι από ατομική βαθμονόμηση
  criteria=(cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100, 1e-5),  # Κριτήρια σύγκλισης
  flags=cv.CALIB_FIX_INTRINSIC            # Διατήρηση σταθερών εσωτερικών παραμέτρων
)

# Υπολογισμός απόστασης μεταξύ καμερών
baseline = float(np.linalg.norm(T))
print(f"\n Ολοκλήρωση στερεοσκοπικής βαθμονόμησης:")
print(f"  Σφάλμα RMS: {ret_stereo:.4f} pixels")
print(f"  Απόσταση καμερών: {baseline*1000:.1f} mm")
print(f"  Εστιακή απόσταση: {K_l[0,0]:.1f} pixels")
```

Το flag CALIB_FIX_INTRINSIC διατηρεί τα intrinsic χαρακτηριστικά που υπολογίστηκαν στο προηγούμενο βήμα, εστιάζοντας μόνο στον υπολογισμό των extrinsic παραμέτρων. Το baseline που υπολογίζεται (περίπου 100mm στο σύστημά μας) είναι κρίσιμο για την ακρίβεια εκτίμησης βάθους - μεγαλύτερο baseline δίνει καλύτερη ακρίβεια σε μεγάλες αποστάσεις αλλά περιορίζει το κοινό οπτικό πεδίο των καμερών.

Η επόμενη φάση είναι ο υπολογισμός των παραμέτρων rectification, που θα επιτρέψουν τη μετατροπή των εικόνων ώστε οι αντίστοιχες γραμμές σάρωσης να είναι ευθυγραμμισμένες:

```
# Υπολογισμός παραμέτρων διόρθωσης στερεοσκοπικών εικόνων
print(f"\n Υπολογισμός διόρθωσης (alpha={ALPHA_VALUE})...")
R_l, R_r, P_l, P_r, Q, roi_l, roi_r = cv.stereoRectify(
  K_l, D_l, K_r, D_r, img_shape, R, T,    # Παράμετροι από βαθμονόμηση
  alpha=ALPHA_VALUE                       # Παράμετρος κλιμάκωσης (0=crop, 1=full)
)
```

```
# Εξαγωγή διορθωμένης εστιακής απόστασης από πίνακα προβολής
rectified_focal = P_l[0, 0]
print(f"  Διορθωμένη εστιακή απόσταση: {rectified_focal:.1f} pixels")
print(f"  ROI αριστερής: {roi_l}")
print(f"  ROI δεξιάς: {roi_r}")

# Υπολογισμός ποσοστού διατήρησης εικόνας μετά τη διόρθωση
roi_area_l = roi_l[2] * roi_l[3]          # Εμβαδόν περιοχής ενδιαφέροντος
full_area = img_shape[0] * img_shape[1]    # Συνολικό εμβαδόν εικόνας
preserved_percent = (roi_area_l / full_area) * 100
print(f"  Διατηρούμενη περιοχή: {preserved_percent:.1f}%")
```

Η παράμετρος alpha είναι ίσως η πιο κρίσιμη ρύθμιση σε όλη τη διαδικασία βαθμονόμησης. Καθορίζει το trade-off μεταξύ:

- alpha = 0: Διατηρεί μόνο τις έγκυρες περιοχές χωρίς καμία παραμόρφωση, αλλά μπορεί να "κόψει" σημαντικό μέρος της εικόνας
- alpha = 1: Διατηρεί όλα τα pixels της αρχικής εικόνας, αλλά εισάγει παραμόρφωση τύπου "fisheye" στις άκρες

Για τις PiCamera v3, η τιμή 0.5 αποδείχθηκε ιδανική μετά από εκτενείς δοκιμές. Αυτή η τιμή διατηρεί περίπου το 85% του οπτικού πεδίου χωρίς εμφανή παραμόρφωση στις άκρες.

4.2.4 Δημιουργία και Αποθήκευση Χαρτών Διόρθωσης

Οι τελικοί χάρτες διόρθωσης υπολογίζονται με βάση τις παραμέτρους που προέκυψαν από τη βαθμονόμηση. Όλες οι παράμετροι αποθηκεύονται σε συμπιεσμένο αρχείο για εύκολη φόρτωση:

```
# Αποθήκευση όλων των παραμέτρων βαθμονόμησης
print("\n Αποθήκευση βαθμονόμησης...")
np.savez_compressed(
  "stereo_params.npz",
  # Εσωτερικές παράμετροι καμερών
  K_l=K_l, D_l=D_l, K_r=K_r, D_r=D_r,
  # Εξωτερικές παράμετροι στερεοσκοπικού συστήματος
  R=R, T=T, R_l=R_l, R_r=R_r, P_l=P_l, P_r=P_r, Q=Q,
  # Χάρτες αναδιάταξης για διόρθωση
  map_l1=map_l1, map_l2=map_l2, map_r1=map_r1, map_r2=map_r2,
  # Πρόσθετες πληροφορίες συστήματος
  img_shape=np.array(img_shape),        # Διαστάσεις εικόνας
  baseline=baseline,                    # Απόσταση μεταξύ καμερών
  rectified_focal=rectified_focal,      # Διορθωμένη εστιακή απόσταση
  alpha_used=ALPHA_VALUE,               # Παράμετρος alpha που χρησιμοποιήθηκε
```

```
    roi_l=roi_l,                    # Περιοχή ενδιαφέροντος αριστερής
    roi_r=roi_r                     # Περιοχή ενδιαφέροντος δεξιάς
)
```

Η αποθήκευση περιλαμβάνει όχι μόνο τις βασικές παραμέτρους αλλά και metadata όπως το χρησιμοποιούμενο alpha και τα ROIs, επιτρέποντας μελλοντική ανάλυση ή επαναβαθμονόμηση με διαφορετικές ρυθμίσεις.

4.2.5 Επαλήθευση και Οπτικός Έλεγχος Βαθμονόμησης

Η επιτυχία της βαθμονόμησης δεν μπορεί να κριθεί μόνο από αριθμητικές μετρικές. Το script παρέχει οπτική επαλήθευση της διόρθωσης, που είναι απαραίτητη για την επιβεβαίωση της σωστής λειτουργίας:

```
# Επαλήθευση διόρθωσης σε δείγματα εικόνων
print("\n Δοκιμή διόρθωσης σε δείγματα εικόνων...")
for i, (lp, rp) in enumerate(zip(left_paths[:3], right_paths[:3])):
    # Φόρτωση ζεύγους εικόνων
    test_l = cv.imread(str(lp))
    test_r = cv.imread(str(rp))

    # Εφαρμογή διόρθωσης με χάρτες αναδιάταξης
    rect_l = cv.remap(test_l, map_l1, map_l2, cv.INTER_LINEAR)
    rect_r = cv.remap(test_r, map_r1, map_r2, cv.INTER_LINEAR)

    # Προσθήκη οριζόντιων γραμμών για οπτικό έλεγχο ευθυγράμμισης
    for y in range(0, rect_l.shape[0], 50):
        cv.line(rect_l, (0, y), (rect_l.shape[1], y), (0, 255, 0), 1)
        cv.line(rect_r, (0, y), (rect_r.shape[1], y), (0, 255, 0), 1)

    # Σήμανση περιοχών ενδιαφέροντος με κόκκινα πλαίσια
    cv.rectangle(rect_l, (roi_l[0], roi_l[1]),
                 (roi_l[0]+roi_l[2], roi_l[1]+roi_l[3]), (0, 0, 255), 2)
    cv.rectangle(rect_r, (roi_r[0], roi_r[1]),
                 (roi_r[0]+roi_r[2], roi_r[1]+roi_r[3]), (0, 0, 255), 2)

    # Δημιουργία συνδυασμένης προεπισκόπησης
    combined = np.hstack([rect_l, rect_r])
    combined_small = cv.resize(combined, (1280, 360))

    # Αποθήκευση και εμφάνιση αποτελέσματος
    cv.imwrite(f"rectification_test_{i}.jpg", combined)
    cv.imshow(f"Δοκιμή διόρθωσης {i+1}/3 (πατήστε πλήκτρο)", combined_small)
```

```
cv.waitKey(0)
```

Οι πράσινες οριζόντιες γραμμές που σχεδιάζονται κάθε 50 pixels πρέπει να είναι τέλεια ευθυγραμμισμένες μεταξύ της αριστερής και δεξιάς εικόνας. Αυτό επιβεβαιώνει ότι η epipolar rectification έχει γίνει σωστά και ότι οι αντίστοιχες γραμμές σάρωσης στις δύο εικόνες αναφέρονται στην ίδια οριζόντια γραμμή στο 3D χώρο.

Εικόνα 4.5: Οπτικό αποτέλεσμα Βαθμονόμησης

Τα κόκκινα πλαίσια δείχνουν τις έγκυρες περιοχές (ROI) μετά τη διόρθωση. Αυτές είναι οι περιοχές όπου και οι δύο κάμερες έχουν έγκυρα δεδομένα μετά τη διόρθωση των παραμορφώσεων. Μόνο εντός αυτών των περιοχών μπορεί να γίνει αξιόπιστος υπολογισμός βάθους.

4.2.6 Ενσωμάτωση με το Κύριο Σύστημα

Η βαθμονόμηση από μόνη της δεν έχει αξία αν δεν ενσωματωθεί αποτελεσματικά στο σύστημα εκτέλεσης. Η κλάση CalibratedStereoCamera στο camera_utils.py αναλαμβάνει αυτή την ενσωμάτωση, φορτώνοντας αυτόματα τις παραμέτρους και εφαρμόζοντας τη διόρθωση με διαφανή τρόπο. Η κλάση παρέχει έλεγχο σφαλμάτων, επιτρέποντας στο σύστημα να λειτουργήσει (με μειωμένη ακρίβεια) ακόμα και αν η βαθμονόμηση λείπει ή είναι κατεστραμμένη. Αυτό είναι σημαντικό για την αξιοπιστία του συστήματος σε πραγματικές συνθήκες.

Η κύρια μέθοδος λήψης εικόνων εφαρμόζει αυτόματα τη διόρθωση όταν είναι διαθέσιμη:

```
def capture_stereo_pair(self, rectify: bool = None):
    """Λήψη συγχρονισμένου στερεοσκοπικού ζεύγους."""
    # Ταυτόχρονη λήψη από αριστερή και δεξιά κάμερα
    left_frame = self.cam_left.capture_array("main")
    right_frame = self.cam_right.capture_array("main")

    # Έλεγχος αν θα εφαρμοστεί διόρθωση (από παράμετρο ή προεπιλογή)
    if (rectify if rectify is not None else self.apply_rectification) and self.calibration_loaded:
        # Εφαρμογή χαρτών διόρθωσης για ευθυγράμμιση εικόνων
        left_frame = cv2.remap(left_frame, self.map_left1, self.map_left2, cv2.INTER_LINEAR)
        right_frame = cv2.remap(right_frame, self.map_right1, self.map_right2, cv2.INTER_LINEAR)

    return left_frame, right_frame
```

Η δυνατότητα παράκαμψης της διόρθωσης με την παράμετρο rectify είναι χρήσιμη για debugging ή για περιπτώσεις όπου η ταχύτητα είναι πιο σημαντική από την ακρίβεια.

Επιπλέον, η κλάση παρέχει εύκολη πρόσβαση στις παραμέτρους βαθμονόμησης για χρήση από άλλα modules:

```
def get_calibration_params(self):
    """Εξαγωγή παραμέτρων βαθμονόμησης για υπολογισμό βάθους."""
    # Έλεγχος αν έχει φορτωθεί η βαθμονόμηση
    if not self.calibration_loaded:
        return None

    return {
        'K_left': self.K_left,                  # Εσωτερικές παράμετροι αριστερής
        'K_right': self.K_right,                # Εσωτερικές παράμετροι δεξιάς
        'baseline': float(np.linalg.norm(self.T)), # Απόσταση μεταξύ καμερών
        'focal_length': self.P_left[0, 0],      # Εστιακή απόσταση μετά διόρθωση
        'Q': self.Q                             # Πίνακας reprojection για 3D
    }
```

Αυτές οι παράμετροι είναι απαραίτητες για το module εκτίμησης βάθους, που χρησιμοποιεί το baseline και την εστιακή απόσταση για τη μετατροπή του disparity σε πραγματική απόσταση.

4.2.7 Σημασία για το Συνολικό Σύστημα

Η σωστή βαθμονόμηση είναι θεμελιώδης για την επιτυχία του συστήματος. Χωρίς ακριβή βαθμονόμηση:

1. Οι εκτιμήσεις απόστασης θα είναι λανθασμένες: Ακόμα και μικρά σφάλματα στη βαθμονόμηση μπορούν να οδηγήσουν σε σημαντικές αποκλίσεις στην εκτιμώμενη απόσταση, ειδικά για αντικείμενα μακριά από τις κάμερες.
2. Ο αλγόριθμος stereo matching θα αποτύχει: Χωρίς σωστή rectification, ο αλγόριθμος πρέπει να ψάξει σε 2D χώρο αντί για 1D (κατά μήκος των epipolar lines), αυξάνοντας δραματικά το υπολογιστικό κόστος και μειώνοντας την ακρίβεια.
3. Το σύστημα θα χάσει αντικείμενα στις άκρες: Η λανθασμένη διόρθωση μπορεί να "κόψει" σημαντικές περιοχές της εικόνας, μειώνοντας το αποτελεσματικό οπτικό πεδίο του συστήματος.

Η επένδυση χρόνου στη σωστή βαθμονόμηση αποδίδει πολλαπλάσια σε αξιοπιστία και ακρίβεια του τελικού συστήματος. Το σύστημα που αναπτύχθηκε κάνει αυτή τη διαδικασία όσο το δυνατόν πιο αυτοματοποιημένη και αξιόπιστη, ελαχιστοποιώντας τις πιθανότητες ανθρώπινου λάθους και εξασφαλίζοντας συνεπή αποτελέσματα.

4.3 Εκπαίδευση μοντέλων

Η εκπαίδευση των μοντέλων αναγνώρισης ορίων ταχύτητας αποτέλεσε βασικό στάδιο ανάπτυξης του συστήματος, καθώς η ποιότητα και η ακρίβεια των αποτελεσμάτων εξαρτώνται άμεσα από το πόσο καλά έχουν εκπαιδευτεί τα νευρωνικά δίκτυα. Η διαδικασία πραγματοποιήθηκε εξολοκλήρου σε περιβάλλον Linux, ώστε να εξασφαλιστεί η απρόσκοπτη χρήση των εργαλείων της NVIDIA και να αξιοποιηθούν πλήρως οι δυνατότητες επιτάχυνσης μέσω CUDA [62].

Όλη η διαδικασία της εκπαίδευσης βασίστηκε στο framework της Ultralytics [63], το οποίο παρέχει μια ολοκληρωμένη σουίτα εργαλείων για την εκπαίδευση, την αξιολόγηση και εξαγωγή των μοντέλων

YOLO. Η έκδοση YOLOv8 που χρησιμοποιήθηκε προσφέρει σημαντικές βελτιώσεις σε σχέση με προηγούμενες εκδόσεις, ειδικά στην ανίχνευση μικρών αντικειμένων .

Η συνολική στρατηγική περιέλαβε:

1. Συλλογή μεγάλου και ποικιλόμορφου συνόλου δεδομένων
2. Εφαρμογή data augmentation για ενίσχυση της ποικιλίας
3. Χειροκίνητο annotation όλων των εικόνων
4. Εκπαίδευση δύο διαφορετικών μοντέλων YOLOv8-nano προσαρμοσμένων στις ανάγκες λειτουργίας
5. Εξαγωγή των τελικών μοντέλων σε μορφή συμβατή με το Coral Edge TPU

4.3.1 Δημιουργία Dataset

Η δημιουργία του dataset αποτέλεσε ένα από τα πιο απαιτητικά στάδια της διαδικασίας, κυρίως λόγω της φύσης του προβλήματος. Οι πινακίδες ορίων ταχύτητας, αν και φαινομενικά απλές στην αναγνώριση, παρουσιάζουν σημαντικές προκλήσεις όταν πρέπει να γίνει διάκριση μεταξύ των διαφορετικών ορίων. Τα περισσότερα διαθέσιμα datasets αντιμετωπίζουν όλες τις πινακίδες ορίων ταχύτητας ως μία ενιαία κλάση, χωρίς να γίνεται περαιτέρω διάκριση μεταξύ των διαφορετικών τιμών. Αυτό σήμαινε ότι για τη δημιουργία μοντέλου ικανού να αναγνωρίζει το συγκεκριμένο όριο έπρεπε να δημιουργηθεί ένα custom dataset.

Για την κάλυψη όλων των κλάσεων έγινε εκτεταμένη αναζήτηση εικόνων στο διαδίκτυο, συγκεντρώνοντας περίπου 4.000 αρχικές εικόνες. Στη συνέχεια εφαρμόστηκε data augmentation για την αύξηση του όγκου των δεδομένων, δημιουργώντας νέες εικόνες σε διαφορετικές συνθήκες. Το τελικό dataset περιέλαβε περίπου 24.000 φωτογραφίες.

4.3.1.1 Data Augmentation

Η διαδικασία αύξησης των δεδομένων υλοποιήθηκε με χρήση της βιβλιοθήκης imgaug [64], η οποία παρέχει ένα ευέλικτο framework για την εφαρμογή πολλαπλών μετασχηματισμών. Το script που αναπτύχθηκε εφάρμοζε τους εξής μετασχηματισμούς:

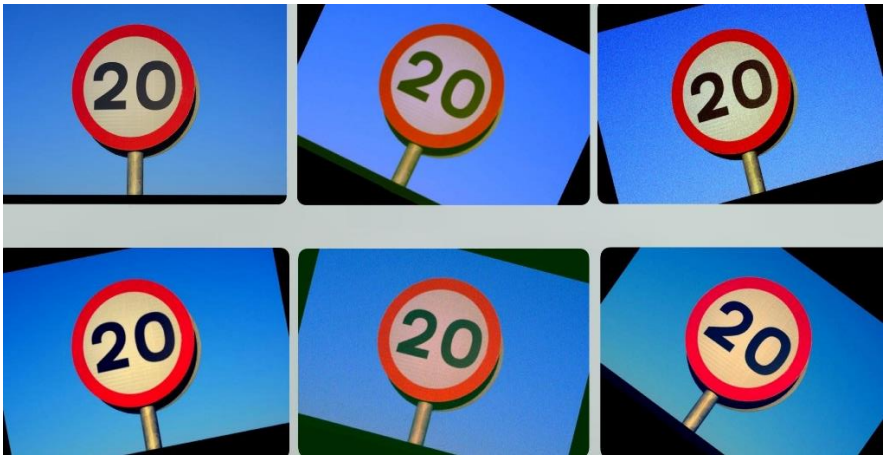
```
# Δημιουργία pipeline για data augmentation
seq = iaa.Sequential([
  # Εφαρμογή θόλωσης Gaussian με τυχαίο sigma
  iaa.GaussianBlur(sigma=(0, 1.0)),
  # Τυχαία περιστροφή εικόνας μεταξύ -45° και +45°
  iaa.Affine(rotate=(-45, 45)),
  # Προσθήκη Gaussian θορύβου με πιθανότητα 50%
  iaa.Sometimes(0.5, iaa.AdditiveGaussianNoise(
    loc=0, scale=(0.0, 0.05*255), per_channel=0.5)),
  # Τυχαία ρύθμιση αντίθεσης ανά κανάλι χρώματος
  iaa.ContrastNormalization((0.5, 2.0), per_channel=1.0),
], random_order=True)  # Εκτέλεση μετασχηματισμών σε τυχαία σειρά
```

Οι συγκεκριμένοι μετασχηματισμοί επιλέχθηκαν για να προσομοιώσουν πραγματικές συνθήκες:\

- Gaussian blur: Προσομοιώνει θολές εικόνες λόγω κίνησης ή εστίασης

- Περιστροφή: Καλύπτει περιπτώσεις στραβών πινακίδων ή διαφορετικών γωνιών λήψης
- Θόρυβος: Προσομοιώνει κακές συνθήκες φωτισμού ή χαμηλής ποιότητας κάμερες
- Μεταβολή αντίθεσης: Καλύπτει διαφορετικές συνθήκες φωτισμού (ημέρα/νύχτα)

Κάθε αρχική εικόνα δημιούργησε 5 επιπλέον παραλλαγές, εξασφαλίζοντας επαρκή ποικιλομορφία στο dataset.



Εικόνα 4.6: Augmented Εικόνα

Στην εικόνα 4.6 φαίνεται η αρχική φωτογραφία πάνω αριστερά και μετρά ακολουθούν τα αποτελέσματα από τους μετασχηματισμούς

4.3.1.2 Annotation Process

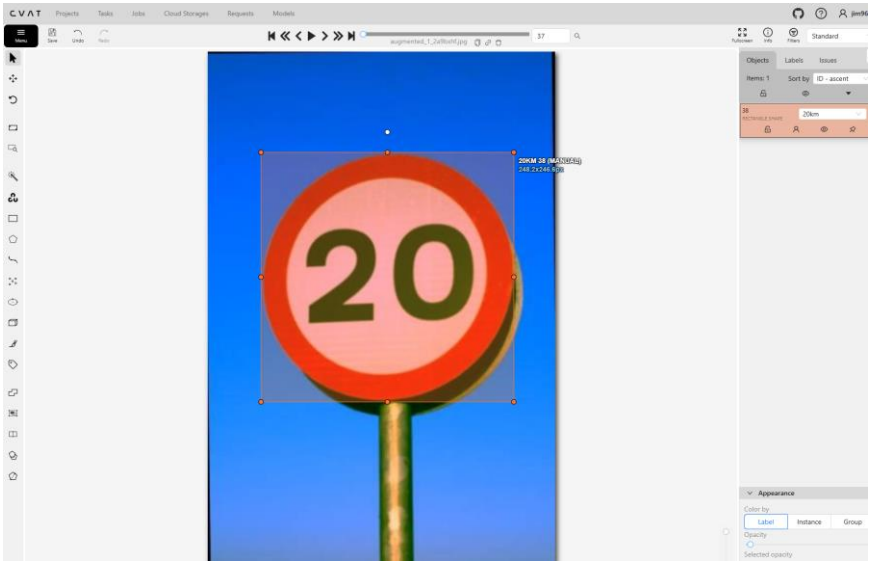
Για τη δημιουργία των labels ήταν απαραίτητη η διαδικασία annotation σε ολόκληρο το σύνολο των εικόνων. Όλες οι εικόνες επισημάνθηκαν χειροκίνητα με χρήση του εργαλείου CVAT [65], ώστε να διασφαλιστεί ότι κάθε πινακίδα είχε το σωστό label και bounding box. Το CVAT επιλέχθηκε λόγω της υποστήριξής του για:

- Batch annotation για παρόμοιες εικόνες
- Interpolation μεταξύ frames για βίντεο
- Εξαγωγή σε YOLO format
- Collaborative annotation με πολλούς χρήστες

Στην περίπτωση του YOLOv8, τα labels ακολουθούν το εξής format:

```
<class_id> <x_center> <y_center> <width> <height>
```

Όπου οι συντεταγμένες κανονικοποιούνται βάσει των διαστάσεων της εικόνας (τιμές 0-1).



Εικόνα 4.7: CVAT

Στην εικόνα 4.5 παρουσιάζεται ένα Screenshot του εργαλείου. Βάση στο πορτοκαλί παράθυρο πάνω από την σήμανση δημιουργούνται τα labels.

4.3.1.3 Dataset Split

Το dataset χωρίστηκε σε training και validation sets με αναλογία 85%-15% αντίστοιχα, ακολουθώντας τις βέλτιστες πρακτικές για object detection [7]. Η δομή των φακέλων οργανώθηκε ως εξής:

```
dataset/
│
├── train/
│   ├── images/
│   └── labels/
│
└── val/
    ├── images/
    └── labels/
```

4.3.2 Training Process

Η εκπαίδευση ενός μοντέλου YOLO είναι η διαδικασία κατά την οποία το νευρωνικό δίκτυο μαθαίνει να αναγνωρίζει τα αντικείμενα μέσα σε εικόνες βασιζόμενο στο dataset. Το YOLOv8 ακολουθεί μια end-to-end προσέγγιση όπου το ίδιο δίκτυο εντοπίζει και ταξινομεί τα αντικείμενα σε μία μόνο διέλευση της εικόνας [8].

Η εκπαίδευση πραγματοποιήθηκε σε σύστημα με NVIDIA RTX 2080 (8GB VRAM), με τις ακόλουθες χρονικές απαιτήσεις:

Detection model (512×512): ~7 ώρες για 100 epochs

Tracking model (256×256): ~4 ώρες για 100 epochs

4.3.2.1 Configuration

Το αρχείο data.yaml που ορίζει τη δομή του dataset:

```
train: /path/to/dataset/train/images
val: /path/to/dataset/val/images

nc: 8
names: ['limit_20', 'limit_30', 'limit_40', 'limit_50',
       'limit_60', 'limit_70', 'limit_80', 'stop']
```

4.3.2.2 Advanced Training Parameters

Για τις ανάγκες του συστήματος εφαρμόστηκε εξειδικευμένη εκπαίδευση με επιπλέον παραμέτρους:

```
from ultralytics import YOLO
# Φόρτωση προ-εκπαιδευμένου μοντέλου
model = YOLO("yolov8n.pt")
# Εκπαίδευση μοντέλου με προσαρμοσμένες παραμέτρους
results = model.train(
  # Βασικές ρυθμίσεις
  data="data.yaml",          # Αρχείο διαμόρφωσης dataset
  epochs=100,                # Αριθμός εποχών εκπαίδευσης
  imgsz=256,                 # Μέγεθος εικόνας εισόδου (256×256 ή 512×512)
  batch=8,                   # Μέγεθος batch (περιορισμός από VRAM)
  device=0,                  # GPU device ID
  workers=8,                 # Threads για φόρτωση δεδομένων
  cache="ram",               # Αποθήκευση εικόνων στη RAM για ταχύτητα
  # Παράμετροι βελτιστοποίησης
  optimizer="AdamW",         # Χρήση AdamW optimizer
  lr0=0.01,                  # Αρχικός ρυθμός μάθησης
  lrf=0.01,                  # Τελική αναλογία learning rate
  momentum=0.9,              # Momentum για SGD
  weight_decay=0.0005,       # L2 regularization
  # Χρονοπρογραμματισμός learning rate
  cos_lr=True,               # Cosine annealing scheduler
  warmup_epochs=5.0,         # Περίοδος προθέρμανσης
  warmup_momentum=0.8,       # Αρχικό momentum προθέρμανσης
  warmup_bias_lr=0.1,        # Bias LR κατά την προθέρμανση
  # Έλεγχος augmentation
  hsv_h=0.015,               # Διακύμανση απόχρωσης
  hsv_s=0.7,                 # Διακύμανση κορεσμού
  hsv_v=0.4,                 # Διακύμανση φωτεινότητας
  translate=0.2,             # Μετατόπιση εικόνας
  scale=0.75,                # Κλιμάκωση εικόνας
  fliplr=0.5,                # Πιθανότητα οριζόντιας αντιστροφής
  mosaic=1.0,                # Mosaic augmentation
```

```
    mixup=0.2,              # MixUp augmentation
    # Προχωρημένες ρυθμίσεις
    freeze=[0],             # Πάγωμα πρώτων layers backbone
    rect=True,              # Ορθογώνια εκπαίδευση για ταχύτητα
    multi_scale=True,       # Εκπαίδευση σε πολλαπλές κλίμακες
    close_mosaic=10,        # Απενεργοποίηση mosaic τελευταίες εποχές
    # Early stopping και αποθήκευση
    patience=50,            # Εποχές αναμονής για early stopping
    save_period=1,          # Αποθήκευση κάθε εποχή
    # Ρυθμίσεις project
    project="traffic_signs",    # Φάκελος project
    name="yolov8n_training_256", # Όνομα εκπαίδευσης
    seed=42,                # Seed για επαναληψιμότητα
    deterministic=True      # Ντετερμινιστική εκπαίδευση
)
```

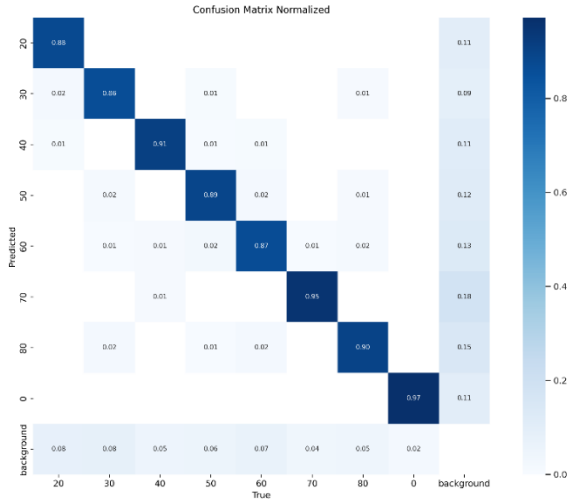
Για την παραμετροποίηση των μοντέλων μας, έγιναν συγκεκριμένες επιλογές που στόχευαν στη βελτιστοποίηση για embedded deployment. Η χρήση του nano variant του YOLOv8 εξασφάλισε ότι τα μοντέλα θα είναι αρκετά ελαφριά για real-time εκτέλεση στο Raspberry Pi με Edge TPU. Το freeze του backbone (πρώτο layer) επέτρεψε τη διατήρηση των pre-trained features για γενική ανίχνευση αντικειμένων, ενώ η εκπαίδευση εστίασε στην εξειδίκευση των deeper layers για τις συγκεκριμένες κλάσεις πινακίδων. Η ενεργοποίηση του multi-scale training με random resize κατά τη διάρκεια της εκπαίδευσης βελτίωσε σημαντικά την ανθεκτικότητα του μοντέλου σε πινακίδες διαφορετικών μεγεθών.

4.3.3 Training Results και Αξιολόγηση

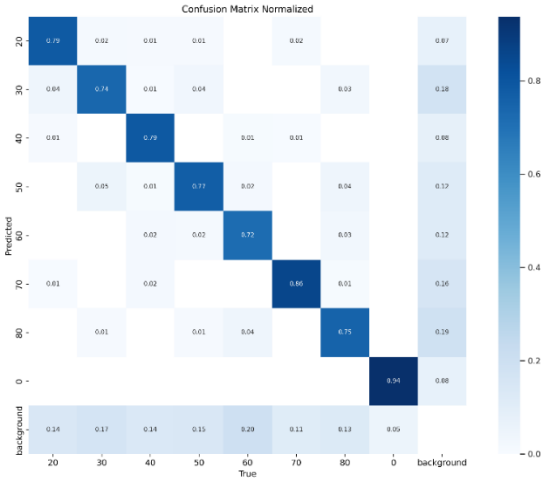
Μετά την ολοκλήρωση της εκπαίδευσης, το σύστημα παρήγαγε εκτενή στατιστικά και διαγράμματα που επέτρεψαν την αξιολόγηση της απόδοσης των μοντέλων. Η διαδικασία αξιολόγησης αποτελεί κρίσιμο στάδιο καθώς μέσω αυτής επιβεβαιώνεται ότι τα μοντέλα μπορούν να γενικεύσουν σε νέα δεδομένα και δεν έχουν απλά απομνημονεύσει το training set. Το framework της Ultralytics παρέχει αυτοματοποιημένη δημιουργία αναλυτικών reports που περιλαμβάνουν πίνακες σύγχυσης, καμπύλες μάθησης και μετρικές απόδοσης.

Ένα από τα σημαντικότερα εργαλεία αξιολόγησης είναι ο πίνακας σύγχυσης (confusion matrix), ο οποίος παρέχει μια ολοκληρωμένη εικόνα της ικανότητας του μοντέλου να ταξινομεί σωστά κάθε κατηγορία. Η ανάγνωση του πίνακα είναι απλή αλλά αποκαλυπτική - οι γραμμές αντιπροσωπεύουν τις πραγματικές κλάσεις των αντικειμένων στο validation set, ενώ οι στήλες δείχνουν τις προβλέψεις του μοντέλου. Σε ένα ιδανικό σενάριο, όλες οι τιμές θα συγκεντρώνονταν στη διαγώνιο του πίνακα, υποδεικνύοντας τέλεια ταξινόμηση. Στην πραγματικότητα όμως, πάντα υπάρχουν κάποιες λανθασμένες προβλέψεις που εμφανίζονται εκτός διαγωνίου και αποκαλύπτουν τις αδυναμίες του μοντέλου.

Πίνακας 4.1 Confusion Matrix Detection Model

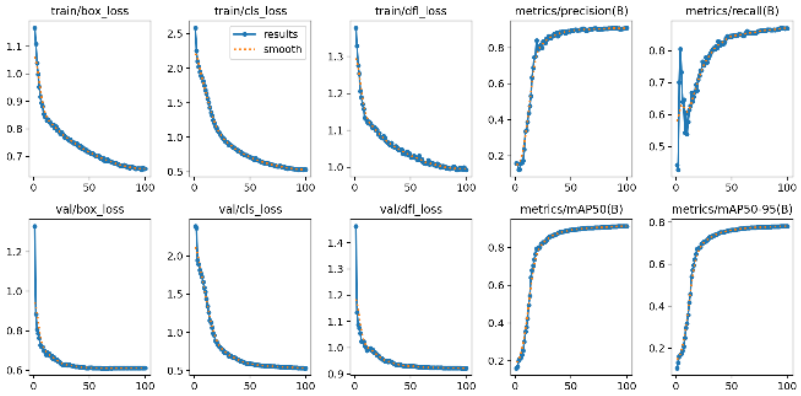


Πίνακας 4.2 Confusion Matrix Tracking Model

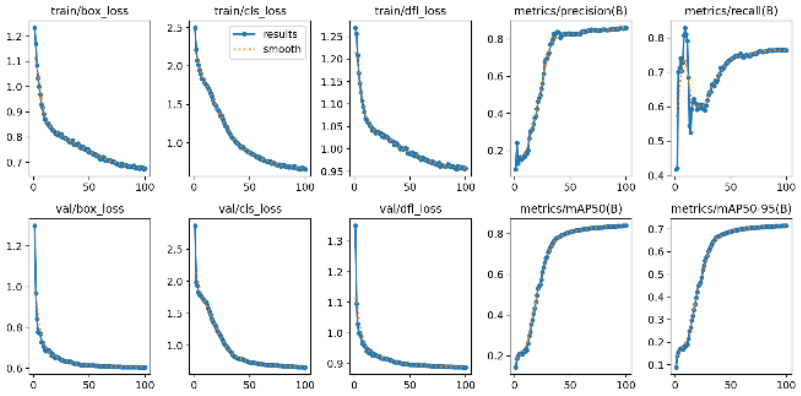


Επιπλέον παρέχονται και γραφήματα που παρουσιάζουν την εξέλιξη των βασικών metrics και losses ανά epoch. Στην παρακάτω εικόνα απεικονίζονται καμπύλες του σφάλματος στην πρόβλεψη θέσεις του bbox (box loss), του σφάλματος ταξινόμησης (classification), του σφάλματος πρόβλεψης κατανομής συντεταγμένων (distribution focal loss), τόσο για το σύνολο της φάσης του training , όσο και του validation. Ακόμα παρουσιάζονται μετρήσεις του ποσοστού προβλέψεων που ήταν σωστές (precision), του ποσοστού των πραγματικών αντικειμένων που εντοπίστηκαν (recall) και οι τιμές mAP.

Πίνακας 4.3 Results Detection Model



Πίνακας 4.4 Results Tracking Model



Οι μετρικές Mean Average Precision (mAP) αποτελούν το de facto standard για την αξιολόγηση object detection μοντέλων, συνδυάζοντας precision και recall σε ένα ενιαίο μέτρο. Το mAP@0.5 μετρά την απόδοση με το σχετικά "χαλαρό" κριτήριο του 50% Intersection over Union (IoU), ενώ το mAP@0.5-0.95 υπολογίζει τον μέσο όρο για IoU thresholds από 0.5 έως 0.95 με βήμα 0.05, παρέχοντας μια πολύ πιο αυστηρή αξιολόγηση της ακρίβειας εντοπισμού.

Το detection model ολοκλήρωσε την εκπαίδευση επιτυγχάνοντας mAP@0.5 ίσο με 0.9135 και mAP@0.5-0.95 ίσο με 0.7808. Αυτές οι τιμές υποδεικνύουν εξαιρετική ικανότητα ανίχνευσης, με το υψηλό mAP@0.5 να εγγυάται ότι το μοντέλο σπάνια "χάνει" πινακίδες, ενώ το επίσης υψηλό mAP@0.5-0.95 επιβεβαιώνει ότι τα bounding boxes είναι ακριβή και καλά τοποθετημένα. Το tracking model, από την άλλη, πέτυχε mAP@0.5 ίσο με 0.8395 και mAP@0.5-0.95 ίσο με 0.7152. Η μικρή πτώση σε σχέση με το detection model είναι αναμενόμενη και αποδεκτή, δεδομένου ότι το tracking model θυσιάζει λίγη ακρίβεια για σημαντικά υψηλότερη ταχύτητα εκτέλεσης.

Η συνολική αξιολόγηση των αποτελεσμάτων επιβεβαίωσε ότι τα μοντέλα που εκπαιδεύτηκαν είναι κατάλληλα για την εφαρμογή real-time ανίχνευσης πινακίδων. Ο συνδυασμός υψηλής ακρίβειας με χαμηλό υπολογιστικό κόστος επιτρέπει την ομαλή λειτουργία του συστήματος ακόμα και σε απαιτητικές συνθήκες οδήγησης, εξασφαλίζοντας έγκαιρη προειδοποίηση στον οδηγό.

4.3.4 Export σε EdgeTPU

Η πλατφόρμα της Ultralytics προσφέρει τη δυνατότητα εξαγωγής των εκπαιδευμένων μοντέλων σε διάφορες μορφές. Για το Coral Edge TPU απαιτείται ειδική διαδικασία που περιλαμβάνει:

```
from ultralytics import YOLO


# Φόρτωση του εκπαιδευμένου μοντέλου
model = YOLO("path/to/best.pt")


# Εξαγωγή σε μορφή Edge TPU για hardware acceleration
model.export(format="edgetpu")
```

Η διαδικασία INT8 quantization μειώνει το μέγεθος του μοντέλου κατά ~4x και επιτρέπει την εκτέλεση στο Edge TPU. Η απώλεια ακρίβειας από το quantization είναι minimal (<2% στο mAP) και αντισταθμίζεται από τη σημαντικη αύξηση ταχύτητας .

4.4 Αρχιτεκτονική Συστήματος

4.4.1 Εισαγωγή

Το σύστημα στερεοσκοπικής όρασης υποβοηθησης του οδηγου που υλοποιήθηκε στην παρούσα εργασία αποτελείται από ένα σύνολο αλληλεξαρτώμενων υποσυστημάτων που συνεργάζονται για την επίτευξη real-time ανίχνευσης και παρακολούθησης οδικών σημάτων. Η αρχιτεκτονική σχεδιάστηκε με γνώμονα τρεις βασικές αρχές: την αποδοτικότητα σε περιορισμένους πόρους embedded συστημάτων, την επεκτασιμότητα για μελλοντικές βελτιώσεις, και την αξιοπιστία για συνεχή λειτουργία υπό πραγματικές συνθήκες.

Το σύστημα βασίζεται σε μια πολυεπίπεδη αρχιτεκτονική (layered architecture) που διαχωρίζει σαφώς τις αρμοδιότητες κάθε επιπέδου. Στο κατώτερο επίπεδο βρίσκεται το hardware που περιλαμβάνει το Raspberry Pi 5, δύο κάμερες για στερεοσκοπική όραση, αισθητήρες GPS/IMU για μέτρηση ταχύτητας, και έναν Arduino με LCD και LED για παροχή feedback στον οδηγό. Πάνω από αυτό υπάρχει το επίπεδο οδηγών (driver layer) που παρέχει abstractions για την επικοινωνία με το hardware. Το επίπεδο επεξεργασίας (processing layer) φιλοξενεί τους κύριους αλγόριθμους του συστήματος, ενώ στην κορυφή βρίσκεται το επίπεδο εφαρμογής (application layer) με το κεντρικό σύστημα ελέγχου και τη διεπαφή χρήστη.

Η επικοινωνία μεταξύ όλων των υποσυστημάτων γίνεται μέσω ενός κεντρικού συστήματος διαχείρισης δεδομένων που ονομάζεται DataBus. Το DataBus υλοποιεί το publisher-subscriber pattern επιτρέποντας στα διάφορα components να ανταλλάσσουν δεδομένα και events χωρίς να χρειάζεται να γνωρίζουν τις λεπτομέρειες υλοποίησης το ένα του άλλου. Αυτή η αρχιτεκτονική επιλογή μειώνει δραστικά την πολυπλοκότητα και διευκολύνει τη μελλοντική επέκταση του συστήματος.

Vision Pipeline

Το stereo vision pipeline αποτελεί το κύριο υποσύστημα επεξεργασίας εικόνας και είναι υπεύθυνο για την ανίχνευση και παρακολούθηση οδικών σημάτων. Λειτουργεί σε δύο διακριτές καταστάσεις: DETECTION για την αναζήτηση νέων πινακίδων και TRACKING για την παρακολούθηση ήδη

εντοπισμένων πινακίδων. Η μετάβαση μεταξύ των καταστάσεων γίνεται αυτόματα με βάση την παρουσία ή απουσία έγκυρων ανιχνεύσεων. Το pipeline χρησιμοποιεί YOLO μοντέλα εκπαιδευμένα ειδικά για ελληνικές πινακίδες και περιλαμβάνει εξελιγμένους μηχανισμούς fallback για την αντιμετώπιση προσωρινών απωλειών tracking. Ακομα υπολογιζει την αποσταση από της πινακιδες και εμφανιζει γραφικα τα αποτελεσματα για ελεγχο σωστης λειτουργιας του συστηματος.

Speed Pipeline

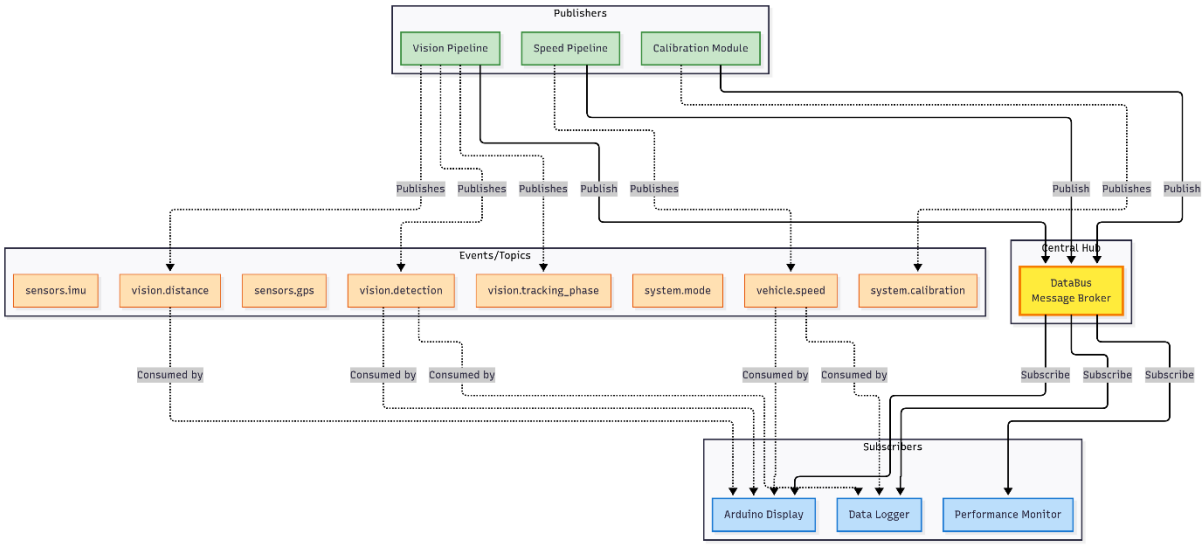
Το speed pipeline είναι υπεύθυνο για την ακριβή εκτίμηση της ταχύτητας του οχήματος μέσω συνδυασμού δεδομένων από GPS και IMU (Inertial Measurement Unit). Κατά την εκκίνηση του συστήματος, το pipeline εκτελεί μια διαδικασία καλιμπραρίσματος διάρκειας 10-15 δευτερολέπτων για τον υπολογισμό του σφάλματος (bias) του IMU. Μετά το καλιμπράρισμα, χρησιμοποιεί τεχνικές sensor fusion για να παράγει ομαλές και ακριβείς μετρήσεις ταχύτητας που ενημερώνονται με υψηλή συχνότητα.

Arduino Display System

Το Arduino display system παρέχει real-time οπτική ανάδραση στον οδηγό μέσω μιας οθόνης LCD 16x2 χαρακτήρων και ενός RGB LED. Εμφανίζει την τρέχουσα ταχύτητα, το ισχύον όριο ταχύτητας, και πληροφορίες για τυχόν ανιχνευμένες πινακίδες. Το LED αλλάζει χρώμα ανάλογα με την κατασταση.

4.4.2 Το DataBus Pattern

Το DataBus αποτελεί την καρδιά του συστήματος επικοινωνίας και υλοποιεί ένα κεντρικό σύστημα διαχείρισης δεδομένων που επιτρέπει την ασύγχρονη ανταλλαγή πληροφοριών μεταξύ των διαφόρων components. Η σχεδίασή του βασίστηκε στην ανάγκη για ένα ευέλικτο σύστημα που θα μπορούσε να εξελιχθεί σταδιακά χωρίς να επηρεάζει την υπάρχουσα λειτουργικότητα.



Εικόνα 4.8: Διάγραμμα σύνδεσης του Databus

Οι publishers (Vision Pipeline, Speed Pipeline, Calibration) δημοσιεύουν events σε συγκεκριμένα topics (π.χ. vehicle.speed, vision.detection), ενώ οι subscribers (Arduino Display, Logger, Monitor) εγγράφονται στα topics που τους ενδιαφέρουν και λαμβάνουν αυτόματα τις ενημερώσεις

```
class DataBus:
  def __init__(self, arduino_port: str = None, threshold_distance: float = None,
          max_subscribers_per_topic: int = 100):
    # Reentrant lock για ασφαλή πρόσβαση από πολλαπλά threads
    self._lock = threading.RLock()

    # Εσωτερική κατάσταση συστήματος
    self._speed: int = 0                    # Τρέχουσα ταχύτητα
    self._det: Optional[Dict[str, float]] = None # Δεδομένα ανίχνευσης

    # Σύστημα pub/sub με προστασία μνήμης
    self._subscribers: Dict[str, List[Callable]] = defaultdict(list)  # Λίστες callbacks ανά topic
    self._data_store: Dict[str, Any] = {}                  # Αποθήκευση δεδομένων
    self._max_subscribers_per_topic = max_subscribers_per_topic      # Όριο subscribers

    # Ενσωμάτωση υπάρχοντος Arduino interface
    self.arduino = ArduinoPipelineInterface(arduino_port, threshold_distance)
```

Η υλοποίηση του DataBus χρησιμοποιεί ένα threading.RLock() για thread safety. Το RLock (Reentrant Lock) επιτρέπει στο ίδιο thread να αποκτήσει το lock πολλές φορές, αποφεύγοντας deadlocks όταν μια μέθοδος που έχει ήδη το lock καλεί μια άλλη μέθοδο που επίσης το χρειάζεται. Η εσωτερική κατάσταση του DataBus διατηρεί τα βασικά δεδομένα του συστήματος (ταχύτητα, ανιχνεύσεις) ενώ παράλληλα υποστηρίζει ένα σύστημα events μέσω των dictionaries _subscribers και _data_store.

Το Pub-Sub mode επιτρέπει event-driven επικοινωνία μεταξύ components:

```
def subscribe(self, topic: str, callback: Callable[[Any], None]):
  # Thread-safe εγγραφή σε topic
  with self._lock:
    # Έλεγχος ορίου subscribers για αποφυγή υπερχείλισης μνήμης
    if len(self._subscribers[topic]) >= self._max_subscribers_per_topic:
      logging.warning(f"DataBus: Μέγιστος αριθμός subscribers ({self._max_subscribers_per_topic}) "
              f"για το topic '{topic}'. Αφαίρεση παλαιότερου subscriber.")
      # Αφαίρεση παλαιότερου subscriber (FIFO στρατηγική)
      self._subscribers[topic].pop(0)

    # Προσθήκη νέου callback στη λίστα
    self._subscribers[topic].append(callback)
```

Η μέθοδος subscribe προσθέτει callback functions στη λίστα για κάθε topic. Το σύστημα προστατεύεται από memory leaks μέσω του ορίου max_subscribers_per_topic. Αν ξεπεραστεί το όριο, αφαιρείται αυτόματα ο παλαιότερος subscriber (First In First Out).

Event Topics Organization

- Τα events οργανώνονται σε ιεραρχική δομή με namespaces για καλύτερη οργάνωση:
- vehicle.* - Δεδομένα οχήματος (ταχύτητα)
- vision.* - Δεδομένα όρασης (ανιχνεύσεις, απόσταση, tracking phases)

- sensors.* - Ακατέργαστα δεδομένα αισθητήρων (GPS, IMU)
- system.* - Κατάσταση συστήματος (mode, calibration, shutdown)

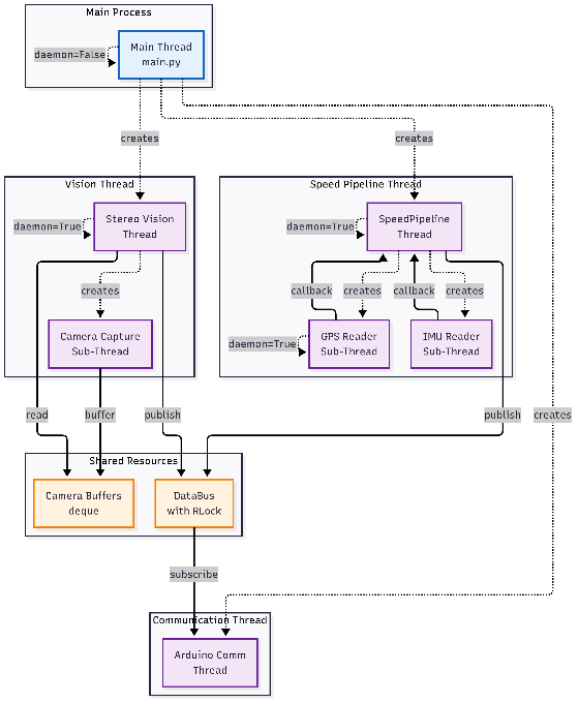
4.4.3 Threading Architecture

Το σύστημα χρησιμοποιεί πολλαπλά threads για παράλληλη επεξεργασία και βελτιστοποίηση της απόδοσης. Η αρχιτεκτονική threading σχεδιάστηκε προσεκτικά για να αποφευχθούν race conditions και deadlocks ενώ παράλληλα μεγιστοποιείται η χρήση των διαθέσιμων πόρων.

Το main thread είναι υπεύθυνο για τον συντονισμό και την εκκίνηση όλων των υποσυστημάτων:

```
def main():
    # Αρχικοποίηση κεντρικού διαύλου δεδομένων με Arduino
    shared = DataBus(
        arduino_port=config.ARDUINO_PORT,
        threshold_distance=config.DISTANCE_THRESHOLD
    )

    # ΦΑΣΗ 1: Εκκίνηση pipeline ταχύτητας για βαθμονόμηση IMU
    print("ΦΑΣΗ 1: Έναρξη σωλήνωσης ταχύτητας για καλιμπράρισμα IMU")
    speed_pipeline = SpeedPipeline(shared)
    speed_thread = speed_pipeline  # SpeedPipeline κληρονομεί από Thread
    speed_thread.start()

    # Αναμονή ολοκλήρωσης βαθμονόμησης IMU
    while not shared.calibration_complete:
        time.sleep(0.5)

    # ΦΑΣΗ 2: Εκκίνηση pipeline στερεοσκοπικής όρασης
    print("ΦΑΣΗ 2: Έναρξη στερεοσκοπικής όρασης")
    stereo_thread = threading.Thread(
        target=stereo_run,
        args=(shared,),
        name="StereoPipeline"
    )
    stereo_thread.start()
```

Η εκκίνηση γίνεται σε δύο φάσεις. Πρώτα ξεκινά το speed pipeline για το απαραίτητο καλιμπράρισμα του IMU, και μόνο μετά την επιτυχή ολοκλήρωσή του ξεκινά το stereo vision pipeline. Αυτή η σειριακή εκκίνηση εξασφαλίζει ότι το σύστημα θα έχει ακριβείς μετρήσεις ταχύτητας από την αρχή της λειτουργίας.



Εικόνα 4.9: Αρχιτεκτονική Threads

Στην Εικόνα 4.9 φαίνεται ότι το speed pipeline thread διαχειρίζεται δύο επιπλέον threads για τους αισθητήρες:

```
class SpeedPipeline(threading.Thread):
  """Thread για εκτίμηση ταχύτητας με IMU-GPS fusion."""

  def __init__(self, shared_interface):
    # Αρχικοποίηση parent Thread με daemon mode
    super().__init__(name="SpeedPipeline", daemon=True)
    self.shared = shared_interface

    # Δημιουργία GPS reader με άμεσο callback στο fusion module
    self.gps_reader = GPSReader(
      callback=self.fusion.handle_gps,  # Απευθείας σύνδεση με fusion
      stop_evt=self.stop_evt
    )

    # Δημιουργία IMU reader με άμεσο callback στο fusion module
    self.imu_reader = IMUReader(
      callback=self.fusion.handle_imu,  # Απευθείας σύνδεση με fusion
      stop_evt=self.stop_evt
    )
```

Η χρήση callbacks επιτρέπει άμεση επεξεργασία των δεδομένων από τους αισθητήρες χωρίς την καθυστέρηση που θα προκαλούσε η χρήση queues. Τα threads είναι daemon, που σημαίνει ότι θα τερματιστούν αυτόματα όταν κλείσει το κύριο πρόγραμμα.

Ακομη για την ελαχιστοποίηση του latency στη λήψη εικόνων, χρησιμοποιείται ξεχωριστό thread:

```
def capture_thread():
  # Συνεχής λήψη εικόνων μέχρι σήμα τερματισμού
  while not stop.is_set():
    # Ταυτόχρονη λήψη από στερεοσκοπικές κάμερες
    left = cam_left.capture_array("main")
    right = cam_right.capture_array("main")

    # Αποθήκευση σε κυκλικούς buffers (αυτόματη απόρριψη παλαιότερων)
    bufL.append(left)
    bufR.append(right)

# Δημιουργία κυκλικών buffers με μέγιστη χωρητικότητα 3 frames
bufL, bufR = deque(maxlen=3), deque(maxlen=3)
```

Η χρήση Queue με put_nowait() εξασφαλίζει ότι το κύριο πρόγραμμα δεν θα μπλοκάρει ποτέ λόγω αργής σειριακής επικοινωνίας. Αν η ουρά γεμίσει, οι νέες ενημερώσεις απορρίπτονται και το γεγονός καταγράφεται για debugging.

4.4.4 Logging και Performance Monitoring

Το σύστημα logging αποτελείται από δύο κύρια components:

Κάθε εκτέλεση δημιουργεί αυτόματα ένα μοναδικό session με οργανωμένη δομή φακέλων:

```
results/
  ses_001/
    frames.json          # Frame-by-frame δεδομένα
    performance.json     # Performance metrics
    summary.json         # Συνοπτικά στατιστικά
    stereo_output.mp4    # Καταγεγραμμένο video
```

Συλλέγει και συνδυάζει δεδομένα από όλα τα υποσυστήματα:

```
@dataclass
class UnifiedFrameData:

  # Αναγνωριστικά frame
  frame_id: int                    # Μοναδικό ID frame
  timestamp: float                 # Χρονοσήμανση Unix

  # Δεδομένα ταχύτητας σε km/h
  gps_speed: Optional[float] = None       # Ταχύτητα από GPS
```

```
    imu_speed: Optional[float] = None        # Ταχύτητα από IMU
    pipeline_speed: Optional[float] = None   # Συγχωνευμένη ταχύτητα

    # Δεδομένα ανίχνευσης αντικειμένων
    detected_class: Optional[int] = None     # ID κατηγορίας αντικειμένου
    detection_confidence: Optional[float] = None  # Βαθμός εμπιστοσύνης
    detection_distance: Optional[float] = None    # Απόσταση σε μέτρα

    # Μετρικές απόδοσης συστήματος
    cpu_percent_normalized: float = 0.0      # Χρήση CPU (0-100%)
    temperature_c: float = 0.0               # Θερμοκρασία σε Celsius
    total_frame_time_ms: float = 0.0         # Συνολικός χρόνος επεξεργασίας
```

Το logging σύστημα καταγράφει λεπτομερή δεδομένα για κάθε frame, επιτρέποντας εκ των υστέρων ανάλυση της απόδοσης και συμπεριφοράς του συστήματος.

4.4.5 Συμπεράσματα Αρχιτεκτονικής

Η αρχιτεκτονική που παρουσιάστηκε επιτυγχάνει τους στόχους που τέθηκαν κατά τη σχεδίαση: παρέχει υψηλή απόδοση σε περιορισμένους πόρους, είναι εύκολα επεκτάσιμη χάρη στο DataBus pattern, και προσφέρει αξιόπιστη λειτουργία μέσω των μηχανισμών fallback και error handling. Η modular δομή επιτρέπει την ανεξάρτητη ανάπτυξη και δοκιμή κάθε υποσυστήματος, ενώ το comprehensive logging system διευκολύνει το debugging και τη βελτιστοποίηση.

Τα επόμενα υποκεφάλαια θα εστιάσουν στην αναλυτική περιγραφή της υλοποίησης κάθε υποσυστήματος, ξεκινώντας από το Speed Pipeline (4.5), συνεχίζοντας με το Stereo Vision Pipeline (4.6), και ολοκληρώνοντας με το Arduino Display System (4.7).

4.5 Speed Pipeline - Υποσύστημα Εκτίμησης Ταχύτητας με Sensor Fusion

4.5.1 Εισαγωγή και Σημασία του Υποσυστήματος

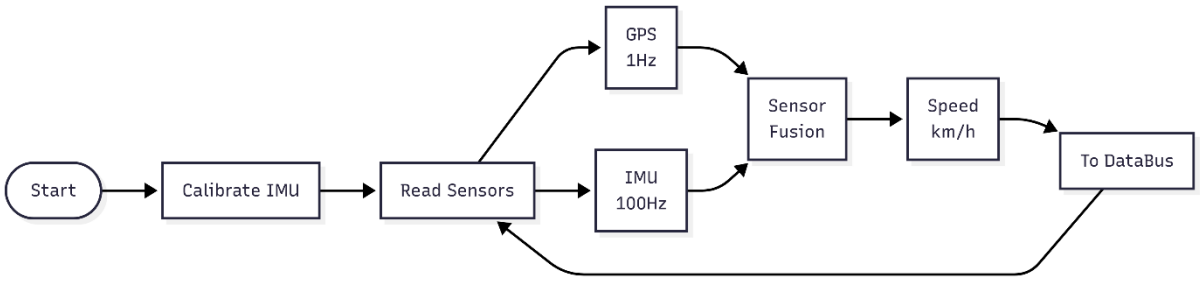
Το Speed Pipeline αποτελεί ένα από τα πιο κρίσιμα υποσυστήματα του συστήματος , καθώς είναι υπεύθυνο για την παροχή συνεχών και ακριβών μετρήσεων της ταχύτητας του οχήματος σε πραγματικό χρόνο. Η σημασία του είναι θεμελιώδης για τη λειτουργία του συνολικού συστήματος, αφού οι μετρήσεις ταχύτητας χρησιμοποιούνται για τη σύγκριση με τα όρια ταχύτητας που ανιχνεύονται από τις οδικές πινακίδες, επιτρέποντας έτσι την έγκαιρη ειδοποίηση του οδηγού για τυχόν υπερβάσεις. Επιπλέον, τα δεδομένα ταχύτητας καταγράφονται για κάθε frame του stereo vision pipeline, δημιουργώντας έτσι ένα πλήρες log της κίνησης του οχήματος που μπορεί να χρησιμοποιηθεί για μετέπειτα ανάλυση και βελτιστοποίηση του συστήματος.

Η λειτουργία του Speed Pipeline βασίζεται σε προηγμένες τεχνικές sensor fusion που συνδυάζουν δεδομένα από τους δύο διαφορετικούς αισθητήρες (GPS και IMU). Ο συνδυασμός αυτών των δύο πηγών δεδομένων εξασφαλίζει αξιόπιστες εκτιμήσεις ταχύτητας ακόμα και σε δύσκολες συνθήκες, όπως σε περιοχές με κακή λήψη GPS σήματος (τούνελ, πυκνή δόμηση) ή κατά τις απότομες αλλαγές ταχύτητας όπου το GPS έχει καθυστέρηση στην απόκριση. Το σύστημα που αναπτύχθηκε εκμεταλλεύεται την ακρίβεια του GPS για μακροπρόθεσμη σταθερότητα και την άμεση απόκριση του

IMU για βραχυπρόθεσμες μεταβολές, δημιουργώντας έτσι ένα υβριδικό σύστημα με τα καλύτερα χαρακτηριστικά και των δύο αισθητήρων.

4.5.2 Αρχιτεκτονική και Οργάνωση του Speed Pipeline

Το Speed Pipeline έχει σχεδιαστεί ως ένα αυτόνομο thread που κληρονομεί από την κλάση threading.Thread της Python, επιτρέποντας έτσι την παράλληλη εκτέλεση με τα υπόλοιπα υποσυστήματα χωρίς να επηρεάζει την απόδοσή τους. Η αρχιτεκτονική του ακολουθεί το μοτίβο producer-consumer, όπου οι αισθητήρες (GPS και IMU) λειτουργούν ως producers που παράγουν συνεχώς νέες μετρήσεις, ενώ το fusion module λειτουργεί ως consumer που καταναλώνει αυτές τις μετρήσεις και τις συνδυάζει για να παράγει την τελική εκτίμηση ταχύτητας.



Εικόνα 4.10: Διάγραμμα Ροής Speed Pipeline

Η εικόνα 4.10 απεικονίζει το διάγραμμα ροής του speed pipeline.

```
class SpeedPipeline(threading.Thread):
  def __init__(self, shared_interface):
    # Κληρονόμηση από Thread με daemon mode
    super().__init__(name="SpeedPipeline", daemon=True)
    self.shared = shared_interface  # Σύνδεση με κεντρικό DataBus

    # Δημιουργία fusion module με callback για ενημέρωση Arduino
    self.fusion = IMUGPSFusion(
      shared_interface,
      calibration_callback=shared_interface.update_calibration  # Callback για βαθμονόμηση
    )

    # Αρχικοποίηση GPS reader με άμεση σύνδεση στο fusion
    self.gps_reader = GPSReader(
      callback=self.fusion.handle_gps,  # Απευθείας επεξεργασία GPS
      stop_evt=self.stop_evt
    )
    # Αρχικοποίηση IMU reader με άμεση σύνδεση στο fusion
    self.imu_reader = IMUReader(
      callback=self.fusion.handle_imu,  # Απευθείας επεξεργασία IMU
      stop_evt=self.stop_evt
```

```
    )
```

Η αρχιτεκτονική του Speed Pipeline έχει σχεδιαστεί με γνώμονα την ελαχιστοποίηση του latency και τη μεγιστοποίηση της αξιοπιστίας. Η χρήση direct callbacks αντί για message queues εξασφαλίζει ότι κάθε νέα μέτρηση από τους αισθητήρες θα προωθηθεί αμέσως στο fusion module για επεξεργασία, χωρίς την καθυστέρηση που θα προκαλούσε η αποθήκευση σε buffer ή queue. Αυτή η σχεδιαστική επιλογή είναι κρίσιμη για τη διατήρηση της χρονικής ακρίβειας των μετρήσεων, ειδικά του IMU που λειτουργεί στα 100Hz και απαιτεί άμεση επεξεργασία για την αποφυγή απώλειας δεδομένων.

4.5.3 GPS Reader - Επεξεργασία NMEA Sentences

Ο GPS Reader είναι υπεύθυνος για την επικοινωνία με τον GPS δέκτη του οχήματος μέσω σειριακής θύρας και την εξαγωγή των δεδομένων ταχύτητας από τα NMEA sentences. Το NMEA (National Marine Electronics Association) είναι το πρότυπο επικοινωνίας που χρησιμοποιούν οι περισσότεροι GPS δέκτες για την αποστολή δεδομένων θέσης και ταχύτητας. Ο reader αναγνωρίζει συγκεκριμένα τα VTG (Track Made Good and Ground Speed) sentences που περιέχουν την ταχύτητα του οχήματος απευθείας σε knots, αποφεύγοντας έτσι την ανάγκη για υπολογισμό της ταχύτητας από τις αλλαγές θέσης που θα εισήγαγε επιπλέον καθυστέρηση και σφάλμα.

```
def _parse_vtg_knots(self, sentence: str) -> Optional[float]:
  # Έλεγχος εγκυρότητας VTG sentence
  if not ("$" in sentence and "VTG" in sentence):
      return None


  # Διαχωρισμός πεδίων με κόμμα
  fields = sentence.split(",")
  if len(fields) >= 9:
      # Εξαγωγή ταχύτητας από πεδίο 5 (knots)
      speed_str = fields[5]
      if speed_str:
          return float(speed_str)
  return None


# Κύριος βρόχος ανάγνωσης GPS
while not self.stop_evt.is_set():
  # Ανάγνωση γραμμής από σειριακή θύρα
  line = self.ser.readline()
  if line:
      # Αποκωδικοποίηση και καθαρισμός string
      line_str = line.decode('ascii', errors='ignore').strip()
      # Εξαγωγή ταχύτητας σε knots
      knots = self._parse_vtg_knots(line_str)


      if knots is not None:
```

```
        # Μετατροπή knots σε m/s (1 knot = 0.514444 m/s)
        mps = knots * 0.514444
        # Άμεση αποστολή στο fusion module
        self.callback(mps)
```

Η επιλογή του VTG sentence έναντι άλλων NMEA sentences (όπως RMC ή GGA) έγινε επειδή το VTG παρέχει την ταχύτητα απευθείας χωρίς να χρειάζεται υπολογισμός από συντεταγμένες, μειώνοντας έτσι το computational overhead και βελτιώνοντας την ακρίβεια. Η μετατροπή από knots σε μέτρα ανά δευτερόλεπτο γίνεται με τον ακριβή συντελεστή 0.514444, εξασφαλίζοντας ότι οι μετρήσεις είναι συμβατές με το SI σύστημα μονάδων που χρησιμοποιεί το υπόλοιπο σύστημα.

4.5.4 IMU Reader - Συλλογή Δεδομένων Επιτάχυνσης

Ο IMU Reader επικοινωνεί με τον αισθητήρα BNO055 μέσω του I2C bus του Raspberry Pi και συλλέγει μετρήσεις γραμμικής επιτάχυνσης με συχνότητα 100Hz. Ο BNO055 είναι ένας έξυπνος 9-axis αισθητήρας που διαθέτει ενσωματωμένο fusion processor, ο οποίος αφαιρεί αυτόματα τη συνιστώσα της βαρύτητας από τις μετρήσεις επιτάχυνσης, παρέχοντας έτσι απευθείας τη γραμμική επιτάχυνση του οχήματος. Αυτό απλοποιεί σημαντικά την επεξεργασία, καθώς δεν χρειάζεται να υπολογίσουμε και να αφαιρέσουμε τη βαρύτητα με software.

```
# Αρχικοποίηση BNO055 σε IMUPLUS mode
self.bno.mode = adafruit_bno055.IMUPLUS_MODE  # Λειτουργία χωρίς μαγνητόμετρο

# Συνεχής ανάγνωση γραμμικής επιτάχυνσης στα 100Hz
def run(self):
  while not self.stop_evt.is_set():
    # Λήψη γραμμικής επιτάχυνσης (βαρύτητα αφαιρείται αυτόματα)
    linear_accel = self.bno.linear_acceleration

    if linear_accel and linear_accel[0] is not None:
      # Εξαγωγή επιτάχυνσης άξονα X (εμπρός/πίσω κίνηση)
      forward_accel = linear_accel[0]
      # Άμεση αποστολή στο fusion module
      self.callback(forward_accel)

    # Διατήρηση ρυθμού δειγματοληψίας 100Hz
    time.sleep(0.01)
```

Η συχνότητα των 100Hz επιλέχθηκε ως ο καλύτερος συμβιβασμός μεταξύ χρονικής ανάλυσης και computational load, παρέχοντας αρκετά δείγματα για ομαλή παρακολούθηση των αλλαγών ταχύτητας χωρίς να υπερφορτώνει το σύστημα.

4.5.5 Διαδικασία βαθμονομισης IMU

Η διαδικασία βαθμονομισης είναι απαραίτητη για τον υπολογισμό του συστηματικού σφάλματος (bias) του IMU που πρέπει να αφαιρεθεί από κάθε μέτρηση. Ο αισθητήρας επιτάχυνσης έχει ένα μικρό αλλά σταθερό σφάλμα που αν δεν διορθωθεί θα προκαλέσει σταδιακή συσσώρευση λάθους στην εκτίμηση ταχύτητας[67]. Το βαθμονομισης εκτελείται με το όχημα εντελώς ακίνητο για 10-15 δευτερόλεπτα, συλλέγοντας χιλιάδες μετρήσεις για στατιστική ανάλυση.

```
# Διαδικασία βαθμονόμησης IMU κατά την ακινησία
if not self.calibrated:
  # Συλλογή δειγμάτων για ανάλυση
  self.gps_samples.append(speed_mps)   # Επιβεβαίωση μηδενικής ταχύτητας
  self.imu_samples.append(accel_mps2)  # Δείγματα για υπολογισμό bias

  # Έλεγχος συνθηκών ολοκλήρωσης βαθμονόμησης
  if len(self.gps_samples) >= 10 and all(s == 0 for s in self.gps_samples):
    if len(self.imu_samples) >= 1000:
      # Στατιστική ανάλυση με IQR για απόρριψη ακραίων τιμών
      imu_array = np.array(self.imu_samples)
      q75, q25 = np.percentile(imu_array, [75, 25])
      iqr = q75 - q25

      # Καθορισμός ορίων και φιλτράρισμα outliers
      lower_bound = q25 - 1.5 * iqr
      upper_bound = q75 + 1.5 * iqr
      filtered = imu_array[(imu_array >= lower_bound) &
                  (imu_array <= upper_bound)]

      # Υπολογισμός τελικού bias
      self.imu_bias = np.mean(filtered)
      self.calibrated = True

      # Ενημέρωση Arduino με αποτελέσματα βαθμονόμησης
      self.calibration_callback('complete', {
        'bias': self.imu_bias,          # Μετατόπιση μηδενός
        'quality': self.calibration_quality_score  # Βαθμολογία ποιότητας
      })
```

Η χρήση της μεθόδου IQR (Interquartile Range) για την απόρριψη outliers εξασφαλίζει ότι παροδικές διαταραχές όπως κραδασμοί από διερχόμενα οχήματα ή άνεμος δεν θα επηρεάσουν τον υπολογισμό του bias. Το σύστημα απαιτεί τουλάχιστον 10 μετρήσεις GPS με μηδενική ταχύτητα για να επιβεβαιώσει ότι το όχημα είναι όντως ακίνητο, αποφεύγοντας έτσι λανθασμένη βαθμονομιση αν ο οδηγός κινεί αργά το όχημα.

4.5.6 Sensor Fusion με Complementary Filtering

Μετά το καλιμπράρισμα, το σύστημα εφαρμόζει complementary filtering για να συνδυάσει τις μετρήσεις GPS και IMU. Το complementary filter είναι μια απλή αλλά αποτελεσματική τεχνική που συνδυάζει τα low-frequency χαρακτηριστικά του GPS (ακρίβεια σε μακροπρόθεσμη βάση) με τα high-frequency χαρακτηριστικά του IMU (γρήγορη απόκριση σε αλλαγές). Το σύστημα προσαρμόζει δυναμικά τα βάρη του filter ανάλογα με την κατάσταση κίνησης του οχήματος.

```
# Ολοκλήρωση επιτάχυνσης IMU για υπολογισμό ταχύτητας
corrected_accel = accel_mps2 - self.imu_bias  # Διόρθωση με αφαίρεση bias
```

```
if abs(corrected_accel) > 0.02:  # Κατώφλι για αποφυγή θορύβου
   # Υπολογισμός μεταβολής ταχύτητας
   speed_delta = corrected_accel * dt
   # Ενημέρωση ταχύτητας με συντελεστή απόσβεσης
   self.current_speed_mps += speed_delta * damping_factor

# Complementary filtering κατά την ενημέρωση από GPS
# Ανίχνευση κατάστασης επιτάχυνσης από πρόσφατες μετρήσεις
is_accelerating = abs(np.mean(self.recent_accelerations[-5:])) > 0.3

if is_accelerating:
   # Προτεραιότητα σε IMU κατά τις μεταβολές ταχύτητας
   gps_weight = 0.3
   imu_weight = 0.7
else:
   # Προτεραιότητα σε GPS σε σταθερή ταχύτητα
   gps_weight = 0.7
   imu_weight = 0.3

# Συγχώνευση μετρήσεων με δυναμικά βάρη
self.current_speed_mps = (gps_weight * gps_speed +
                          imu_weight * self.current_speed_mps)
```

Η προσαρμογή των βαρών βασίζεται στην παρατήρηση ότι κατά την επιτάχυνση ή επιβράδυνση το GPS έχει καθυστέρηση 1-2 δευτερολέπτων λόγω του ότι υπολογίζει την ταχύτητα από τη μεταβολή θέσης, ενώ το IMU ανταποκρίνεται αμέσως καθώς μετρά απευθείας την επιτάχυνση. Αντίθετα, σε σταθερή ταχύτητα το GPS είναι πιο αξιόπιστο καθώς δεν έχει το πρόβλημα της σταδιακής drift που έχει το IMU από την ολοκλήρωση.

4.5.7 Προσαρμοστική Διόρθωση Bias

Το σύστημα εκτελεί συνεχή προσαρμοστική διόρθωση του IMU bias κατά τη διάρκεια της λειτουργίας για να αντισταθμίσει τη θερμική drift και άλλες αργές μεταβολές. Η διόρθωση βασίζεται στη διαφορά μεταξύ της εκτίμησης του IMU και της μέτρησης του GPS, με διαφορετικούς ρυθμούς διόρθωσης για διαφορετικές καταστάσεις κίνησης.

```
# Προσαρμοστική διόρθωση bias βάσει GPS feedback
# Υπολογισμός σφάλματος μεταξύ GPS και τρέχουσας εκτίμησης
speed_error = gps_speed - current_speed_estimate

if abs(gps_speed) < 0.5:  # Ανίχνευση ακινησίας οχήματος
   # Επιθετική διόρθωση για γρήγορη σύγκλιση
   correction = speed_error * 0.1
else:  # Όχημα σε κίνηση
   # Συντηρητική διόρθωση για αποφυγή αστάθειας
   correction = speed_error * 0.05

# Περιορισμός διόρθωσης για σταθερότητα συστήματος
self.imu_bias += np.clip(correction, -0.02, 0.02)
```

Αυτή η προσαρμοστική διόρθωση εξασφαλίζει ότι το σύστημα παραμένει ακριβές ακόμα και μετά από ώρες λειτουργίας όπου η θερμοκρασία του αισθητήρα μπορεί να έχει αλλάξει σημαντικά. Ο περιορισμός της μέγιστης διόρθωσης ανά update (±0.02 m/s²) αποτρέπει απότομες αλλαγές που θα μπορούσαν να προκαλέσουν αστάθεια στο σύστημα.

4.5.8 Ενσωμάτωση με το Arduino Display System

Το Speed Pipeline επικοινωνεί συνεχώς με το Arduino display system μέσω του DataBus για να παρέχει real-time feedback στον οδηγό. Κατά τη φάση καλιμπραρίσματος, το Arduino εμφανίζει την πρόοδο στην οθόνη LCD και αναβοσβήνει το LED με μπλε χρώμα. Μετά την ολοκλήρωση του καλιμπραρίσματος, το σύστημα μεταβαίνει σε κανονική λειτουργία όπου εμφανίζεται συνεχώς η τρέχουσα ταχύτητα και το LED αλλάζει χρώμα ανάλογα με την κατάσταση (πράσινο για κανονική λειτουργία, κόκκινο για υπέρβαση ορίου).
Η επικοινωνία με το Arduino γίνεται μέσω του DataBus που εξασφαλίζει thread-safe πρόσβαση και αποφεύγει race conditions. Το σύστημα στέλνει μόνο ακέραιες τιμές ταχύτητας στο Arduino για να απλοποιήσει την εμφάνιση στην οθόνη LCD 16x2 χαρακτήρων.

```
# Αποστολή κατάστασης βαθμονόμησης στο Arduino
self.calibration_callback('status', {
  'imu_samples': len(self.imu_samples),    # Τρέχων αριθμός δειγμάτων IMU
  'gps_samples': len(self.gps_samples),    # Τρέχων αριθμός δειγμάτων GPS
  'target_imu': 1000,                      # Στόχος δειγμάτων IMU
  'target_gps': 10                         # Στόχος δειγμάτων GPS
})

# Ενημέρωση ταχύτητας στο σύστημα κατά την κανονική λειτουργία
speed_kmh = self.current_speed_mps * 3.6     # Μετατροπή m/s σε km/h
self.shared.update_speed(int(round(speed_kmh)))  # Αποστολή ακέραιας τιμής
```

4.5.9 Συμπεράσματα και Αξιολόγηση Απόδοσης

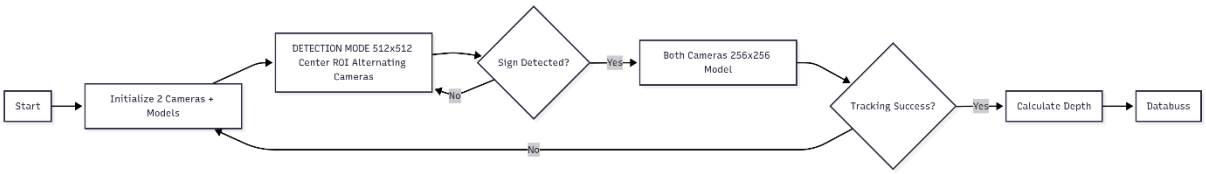
Το Speed Pipeline που υλοποιήθηκε επιτυγχάνει ακρίβεια μέτρησης ταχύτητας με ελάχιστο latency μέσω του έξυπνου συνδυασμού GPS και IMU δεδομένων. Η διαδικασία καλιμπραρίσματος με outlier rejection εξασφαλίζει αξιόπιστο υπολογισμό του IMU bias ακόμα και σε μη-ιδανικές συνθήκες, ενώ το προσαρμοστικό complementary filtering παρέχει ομαλές μεταβάσεις και γρήγορη απόκριση σε αλλαγές ταχύτητας. Το σύστημα είναι ανθεκτικό σε αστοχίες χάρη στους μηχανισμούς automatic recovery και health monitoring που εντοπίζουν και διορθώνουν αυτόματα προβλήματα με τους αισθητήρες.
Η χρήση direct callbacks αντί για queues ελαχιστοποιεί το latency επιτυγχάνοντας real-time απόδοση που είναι κρίσιμη για την ασφάλεια του συστήματος. Η ενσωμάτωση με το Arduino display system μέσω του DataBus παρέχει άμεση οπτική ανάδραση στον οδηγό, ενώ το comprehensive logging επιτρέπει την εκ των υστέρων ανάλυση για περαιτέρω βελτιώσεις. Συνολικά, το Speed Pipeline αποτελεί ένα robust και αξιόπιστο υποσύστημα που καλύπτει πλήρως τις απαιτήσεις του αυτόνομου οχήματος για ακριβή μέτρηση ταχύτητας.

4.6 Vision Pipeline - Υποσύστημα Ανίχνευσης και Παρακολούθησης Οδικών Σημάτων

4.6.1 Εισαγωγή και Αρχιτεκτονική

Το Stereo Vision Pipeline αποτελεί την καρδιά του συστήματος, υπεύθυνο για την ανίχνευση και παρακολούθηση οδικών σημάτων σε πραγματικό χρόνο χρησιμοποιώντας στερεοσκοπική όραση. Το pipeline λειτουργεί σε δύο διακριτές καταστάσεις που μετά από εκτεταμένες δοκιμές έχουν αποδειχθεί οι πιο αποτελεσματικές: το DETECTION mode για την αρχική αναζήτηση πινακίδων στο οπτικό πεδίο, και το TRACKING mode για την παρακολούθηση πινακίδων που έχουν ήδη εντοπιστεί. Η μετάβαση μεταξύ των καταστάσεων γίνεται αυτόματα με βάση συγκεκριμένα κριτήρια, εξασφαλίζοντας ότι το σύστημα χρησιμοποιεί πάντα την πιο κατάλληλη στρατηγική επεξεργασίας. Στο DETECTION mode το

σύστημα εναλλάσσει την επεξεργασία μεταξύ των δύο καμερών για εξοικονόμηση υπολογιστικής ισχύος, ενώ στο TRACKING mode επεξεργάζεται και τις δύο κάμερες παράλληλα με μικρότερα ROIs (Regions of Interest) για μεγαλύτερη ταχύτητα.



Εικόνα 4.11: Διάγραμμα Ροής Stereo Vision Pipeline

Η εικόνα 4.11 απεικονίζει το διάγραμμα ροής του stereo vision pipeline

4.6.2 Κύριος Βρόχος Εκτέλεσης και Διαχείριση Καταστάσεων

Ο κύριος βρόχος του Stereo Vision Pipeline ορχηστρώνει όλα τα υποσυστήματα και διαχειρίζεται τη ροή δεδομένων από τις κάμερες μέχρι την τελική εμφάνιση στην οθόνη. Το σύστημα ξεκινά με την αρχικοποίηση των καμερών σε ανάλυση 1920x1080 pixels, που αποτελεί τον καλύτερο συμβιασμό μεταξύ ποιότητας εικόνας και επεξεργαστικής ταχύτητας για το Raspberry Pi 5.

```
def run(pipeline=None):

    # Αρχικοποίηση στερεοσκοπικών καμερών σε Full HD
    cam_left = Picamera2()
    cam_right = Picamera2(camera_num=1)

    # Διαμόρφωση αριστερής κάμερας
    config_left = cam_left.create_video_configuration(
        main={"size": (1920, 1080), "format": "RGB888"},
        buffer_count=8  # Πολλαπλά buffers για ομαλή ροή
    )
    cam_left.configure(config_left)
    cam_left.start()

    # Αρχική κατάσταση συστήματος
    SYSTEM_MODE = "DETECTION"

    # Δημιουργία κυκλικών buffers για ασύγχρονη λήψη
    bufL, bufR = deque(maxlen=3), deque(maxlen=3)

    # Thread για συνεχή λήψη frames χωρίς blocking
    def capture_thread():
        while not stop.is_set():
            # Παράλληλη λήψη από δύο κάμερες
            left = cam_left.capture_array("main")
            right = cam_right.capture_array("main")
            # Αποθήκευση σε κυκλικούς buffers
            bufL.append(left)
            bufR.append(right)

    # Εκκίνηση capture thread ως daemon
    Thread(target=capture_thread, daemon=True).start()
```

Η χρήση ξεχωριστού thread για τη λήψη frames εξασφαλίζει ότι το κύριο πρόγραμμα δεν θα χάσει frames ενώ εκτελεί επεξεργασία, δημιουργώντας έτσι μια ομαλή ροή δεδομένων. Τα circular buffers (deques) με μέγεθος 3 frames διατηρούν πάντα τα πιο πρόσφατα frames διαθέσιμα για επεξεργασία, αποφεύγοντας την υπερχείλιση μνήμης από τη συσσώρευση παλιών frames.

4.6.3 Camera Utils - Αρχικοποίηση και Ρύθμιση Στερεοσκοπικών Καμερών

Η σωστή αρχικοποίηση και ρύθμιση των δύο καμερών είναι κρίσιμη για την αξιόπιστη λειτουργία του στερεοσκοπικού συστήματος. Το module camera_utils περιέχει όλες τις απαραίτητες συναρτήσεις για την αρχικοποίηση των καμερών Raspberry Pi, τη φόρτωση των παραμέτρων βαθμονόμησης (calibration), και τη διαμόρφωση των κατάλληλων buffers για ομαλή λήψη εικόνων. Οι κάμερες πρέπει να ρυθμιστούν με πανομοιότυπες παραμέτρους (ανάλυση, format, exposure) για να εξασφαλιστεί ότι οι εικόνες από τις δύο κάμερες είναι συγκρίσιμες για στερεοσκοπική επεξεργασία.

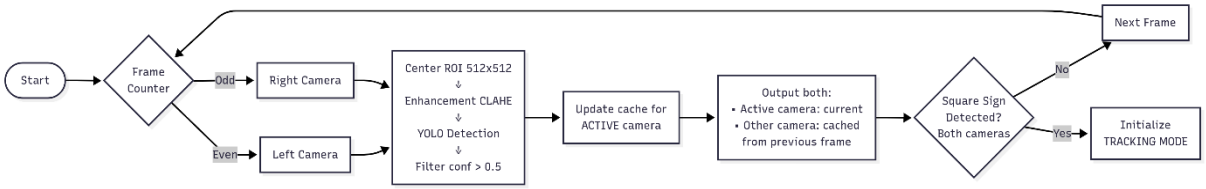
```
def init_cameras(resolution=(1920, 1080), format="RGB888", buffer_count=8):
    from picamera2 import Picamera2

    # Δημιουργία instances για αριστερή και δεξιά κάμερα
    cam_left = Picamera2(camera_num=0)
    cam_right = Picamera2(camera_num=1)

    # Δημιουργία πανομοιότυπων ρυθμίσεων για στερεοσκοπική συμβατότητα
    config_left = cam_left.create_video_configuration(
        main={"size": resolution, "format": format},
        buffer_count=buffer_count  # Πολλαπλά buffers για ομαλή ροή
    )

    config_right = cam_right.create_video_configuration(
        main={"size": resolution, "format": format},
        buffer_count=buffer_count
    )

    # Εφαρμογή ρυθμίσεων
    cam_left.configure(config_left)
    cam_right.configure(config_right)

    # Συγχρονισμός παραμέτρων έκθεσης για ομοιόμορφη φωτεινότητα
    cam_left.set_controls({
        "AeEnable": False,          # Απενεργοποίηση auto exposure
        "ExposureTime": 10000,      # Σταθερή έκθεση 10ms
        "AnalogueGain": 1.0         # Σταθερό gain
    })

    cam_right.set_controls({
        "AeEnable": False,
        "ExposureTime": 10000,      # Ίδια έκθεση με αριστερή
        "AnalogueGain": 1.0
    })

    # Εκκίνηση καμερών
    cam_left.start()
    cam_right.start()

    # Περίοδος σταθεροποίησης
    time.sleep(0.5)

    print(f" Κάμερες αρχικοποιήθηκαν: {resolution[0]}x{resolution[1]} @ {format}")

    return cam_left, cam_right

class CalibratedStereoCamera:

    def __init__(self, calibration_file):
        # Φόρτωση αρχείου βαθμονόμησης
        calib_data = np.load(calibration_file)

        # Εσωτερικές παράμετροι καμερών
        self.K1 = calib_data['K1']  # Πίνακας εσωτερικών αριστερής
        self.K2 = calib_data['K2']  # Πίνακας εσωτερικών δεξιάς
        self.D1 = calib_data['D1']  # Συντελεστές παραμόρφωσης αριστερής
        self.D2 = calib_data['D2']  # Συντελεστές παραμόρφωσης δεξιάς
```

```
    # Εξωτερικές παράμετροι (σχετική θέση)
    self.R = calib_data['R']    # Πίνακας περιστροφής
    self.T = calib_data['T']    # Διάνυσμα μετατόπισης

    # Παράμετροι διόρθωσης
    self.R1 = calib_data['R1']  # Μετασχηματισμός διόρθωσης αριστερής
    self.R2 = calib_data['R2']  # Μετασχηματισμός διόρθωσης δεξιάς
    self.P1 = calib_data['P1']  # Πίνακας προβολής αριστερής
    self.P2 = calib_data['P2']  # Πίνακας προβολής δεξιάς

    # Υπολογισμός χαρακτηριστικών συστήματος
    self.baseline = abs(self.T[0, 0])  # Απόσταση καμερών σε mm
    self.focal_length = self.K1[0, 0]  # Εστιακή απόσταση σε pixels

    print(f"Φορτώθηκε βαθμονόμηση: baseline={self.baseline:.1f}mm, "
          f"focal={self.focal_length:.1f}px")
```

Η αρχικοποίηση των καμερών με ίδιες ρυθμίσεις είναι απαραίτητη για την ορθή λειτουργία της στερεοσκοπίας. Η απενεργοποίηση του auto exposure εξασφαλίζει ότι οι δύο εικόνες θα έχουν την ίδια φωτεινότητα, διευκολύνοντας την αντιστοίχιση σημείων μεταξύ τους. Το buffer_count=8 δημιουργεί ένα buffer 8 frames που απορροφά τυχόν καθυστερήσεις στην επεξεργασία, εξασφαλίζοντας ομαλή ροή εικόνων. Οι παράμετροι βαθμονόμησης που φορτώνονται από το calibration file έχουν υπολογιστεί εκ των προτέρων με τη διαδικασία stereo calibration χρησιμοποιώντας checkerboard patterns και περιέχουν όλες τις απαραίτητες πληροφορίες για τη διόρθωση των οπτικών παραμορφώσεων και τον υπολογισμό του βάθους.

4.6.4 Detection Mode - Εναλλασσόμενη Ανίχνευση με Cache Mechanism

Στο DETECTION mode, το σύστημα αναζητά οδικές πινακίδες στο οπτικό πεδίο χρησιμοποιώντας εναλλασσόμενη επεξεργασία μεταξύ των δύο καμερών με έναν έξυπνο μηχανισμό cache. Η εναλλαγή γίνεται με XOR λογική που εναλλάσσει μεταξύ 0 και 1 κάθε frame, εξασφαλίζοντας ότι κάθε κάμερα επεξεργάζεται κάθε δεύτερο frame. Το κρίσιμο στοιχείο είναι ότι το σύστημα διατηρεί cache των αποτελεσμάτων της κάμερας που δεν είναι σε επεξεργασία, επιτρέποντας έτσι τη σύγκριση των ανιχνεύσεων από τις δύο κάμερες για την απόφαση μετάβασης σε tracking mode. Αυτή η στρατηγική μειώνει το computational load κατά 50% ενώ παράλληλα διατηρεί τη δυνατότητα στερεοσκοπικής επαλήθευσης.



Εικόνα 4.12: Διάγραμμα Ροής Detection Mode

Η εικόνα 4.12 δείχνει την αρχιτεκτονική του detection module

```
class AlternatingDetector:

  def __init__(self, cam_left, cam_right, model_path,
               main_size=(1920, 1080), model_size=512):
    # Αποθήκευση καμερών σε λίστα για εναλλαγή
    self.cam = [cam_left, cam_right]
    # Φόρτωση μοντέλου YOLO
    self.model = YOLO(model_path, task="detect")
    self.model_size = 512

    # Cache τελευταίων ανιχνεύσεων ανά κάμερα
    self.cache = {0: [], 1: []}
```

```
        # Υπολογισμός κεντρικού ROI στην εικόνα
        self.roi_x0 = (1920 - 512) // 2  # Αριστερό όριο (704)
        self.roi_y0 = (1080 - 512) // 2  # Πάνω όριο (284)
        self.roi_x1 = self.roi_x0 + 512  # Δεξιό όριο
        self.roi_y1 = self.roi_y0 + 512  # Κάτω όριο

    def step(self, img_det):

        # Επιλογή κάμερας με XOR toggle (0 ή 1)
        current_feed = self.frame_idx & 1

        # Ανίχνευση στην τρέχουσα κάμερα
        detection = self._detect_in_roi(img_det)

        # Ενημέρωση cache τρέχουσας κάμερας
        self.cache[current_feed] = detection

        # Επιστροφή αποτελεσμάτων και των δύο καμερών
        return {
            0: self.cache[0],  # Αριστερή (τρέχον ή προηγούμενο)
            1: self.cache[1],  # Δεξιά (τρέχον ή προηγούμενο)
            "perf": timing_metrics
        }

    def enhance_adaptive_local(self, img):

        # Μετατροπή σε YCrCb για ανεξάρτητη επεξεργασία φωτεινότητας
        ycrcb = cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
        y, cr, cb = cv2.split(ycrcb)

        # Ισοστάθμιση ιστογράμματος για βελτίωση αντίθεσης
        y = cv2.equalizeHist(y)

        # Εφαρμογή sharpening kernel για ενίσχυση ακμών
        kernel = np.array([[0, -1, 0],
                           [-1, 5, -1],
                           [0, -1, 0]], dtype=np.float32)
        y = cv2.filter2D(y, -1, kernel)

        # Επανασύνθεση και επιστροφή σε RGB
        enhanced = cv2.merge([y, cr, cb])
        return cv2.cvtColor(enhanced, cv2.COLOR_YCrCb2RGB)
```

Η τεχνική adaptive enhancement που εφαρμόζεται είναι κρίσιμη για την αξιόπιστη ανίχνευση σε δύσκολες συνθήκες φωτισμού, όπως έντονη ηλιοφάνεια ή σκιές. Η επεξεργασία στον χώρο YCrCb επιτρέπει την ανεξάρτητη βελτίωση της φωτεινότητας χωρίς να επηρεάζονται τα χρώματα, διατηρώντας έτσι τα χαρακτηριστικά χρώματα των πινακίδων που είναι σημαντικά για την ταξινόμηση.

4.6.5 Μετάβαση από Detection σε Tracking

Η μετάβαση από DETECTION σε TRACKING mode αποτελεί κρίσιμο σημείο για την απόδοση του συστήματος. Το σύστημα μεταβαίνει σε TRACKING μόνο όταν εντοπίσει την ίδια πινακίδα και στις δύο κάμερες με υψηλή βεβαιότητα, εξασφαλίζοντας έτσι αξιόπιστο tracking και ακριβή εκτίμηση βάθους. Επιπλέον, η πινακίδα πρέπει να είναι περίπου τετράγωνη, κριτήριο που βοηθά στην απόρριψη false positives από αντικείμενα που δεν είναι πινακίδες.

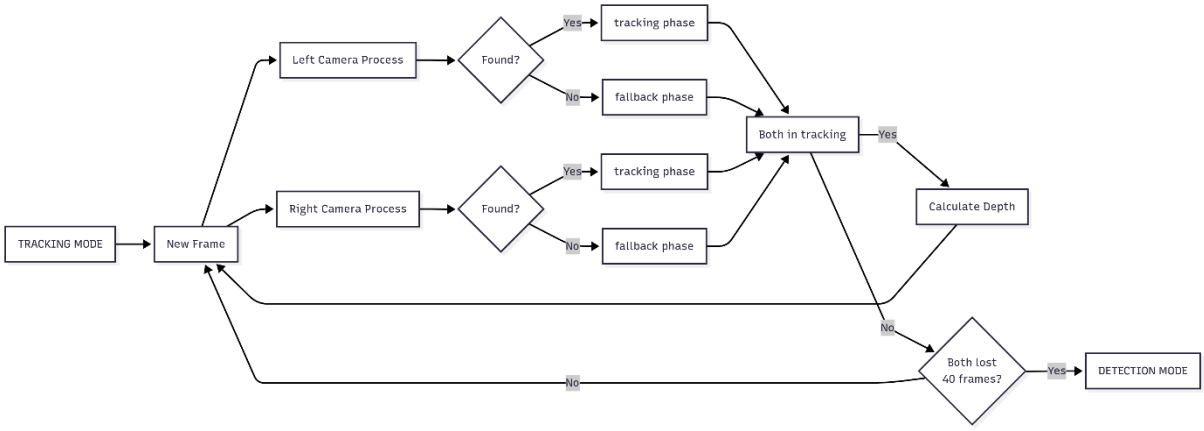
```
# Έλεγχος μετάβασης από DETECTION σε TRACKING
if SYSTEM_MODE == "DETECTION":
    # Εναλλαγή κάμερας με XOR toggle
    detect_feed ^= 1  # Εναλλαγή μεταξύ 0 και 1
    # Επιλογή εικόνας βάσει τρέχουσας κάμερας
    img_det = left_raw if detect_feed == 0 else right_raw
    # Εκτέλεση ανίχνευσης
```

```
last_det = detector.step(img_det)

# Οργάνωση αποτελεσμάτων ανά κάμερα
det_dict = {
    0: last_det[0] if last_det else [],  # Αριστερή κάμερα
    1: last_det[1] if last_det else []   # Δεξιά κάμερα
}

# Προσπάθεια αρχικοποίησης tracking
if tracker.initialize_from_detection(det_dict):
    # Μετάβαση σε κατάσταση παρακολούθησης
    SYSTEM_MODE = "TRACKING"
    print(f"[SYSTEM] DETECTION → TRACKING at frame {frame_counter}")

    # Εξαγωγή και αποθήκευση πληροφοριών ανίχνευσης
    if det_dict[0]:
        current_detected_class = det_dict[0][0].get("cls")      # Κατηγορία αντικειμένου
        current_confidence = det_dict[0][0].get("conf", 0.5)    # Εμπιστοσύνη ανίχνευσης
```

Η συνάρτηση initialize_from_detection του tracker ελέγχει αυστηρά κριτήρια πριν επιτρέψει τη μετάβαση: και οι δύο κάμερες πρέπει να έχουν ακριβώς μία ανίχνευση, οι ανιχνεύσεις πρέπει να είναι περίπου τετράγωνες (aspect ratio μεταξύ 0.8 και 1.25), και να έχουν confidence πάνω από το κατώφλι. Αυτοί οι αυστηροί έλεγχοι εξασφαλίζουν ότι το tracking θα ξεκινήσει μόνο με αξιόπιστα δεδομένα.

4.6.6 Tracking Mode με Συγχρονισμένη Επεξεργασία Δύο Καμερών

Το TRACKING mode αποτελεί την πιο σύνθετη κατάσταση του συστήματος, όπου και οι δύο κάμερες επεξεργάζονται συγχρονισμένα και παράλληλα, σε αντίθεση με το DETECTION mode που εναλλάσσει μεταξύ τους. Αυτή η συγχρονισμένη επεξεργασία είναι απαραίτητη για τον ακριβή υπολογισμό του βάθους μέσω στερεοσκοπικής όρασης, καθώς απαιτούνται ταυτόχρονες μετρήσεις από τις δύο κάμερες για να υπολογιστεί η parallax (παράλλαξη) που δίνει την απόσταση. Το σύστημα υλοποιεί έναν προηγμένο per-camera fallback μηχανισμό που επιτρέπει σε κάθε κάμερα να βρίσκεται σε διαφορετική φάση ανάλογα με την επιτυχία της παρακολούθησης, διατηρώντας παράλληλα τον συγχρονισμό που απαιτείται για τη στερεοσκοπική όραση.



Εικόνα 4.13: Διάγραμμα Ροής Tracking Mode

Η εικόνα 4.13 δείχνει την αρχιτεκτονική του tracking module.

```
c class TrackingModule:
    """Module παρακολούθησης με ανεξάρτητη διαχείριση ανά κάμερα."""

    def __init__(self, model_path_256, lores_size,
                 expansion_rate=0.1, fallback2_frames=40):
        # Φόρτωση ελαφρύτερου μοντέλου για tracking
        self.model = YOLO(model_path_256, task="detect")
```

```
    # Ανεξάρτητη κατάσταση για κάθε κάμερα
    self.phase = {0: "tracking", 1: "tracking"}     # Φάσεις: tracking/fallback1/fallback2
    self.last_box = {0: None, 1: None}              # Τελευταίο bounding box
    self.last_roi = {0: None, 1: None}              # Τελευταίο ROI

    # Παράμετροι διαχείρισης fallback
    self.fallback1_initial_roi = {0: None, 1: None}  # Αρχικό ROI για επέκταση
    self.fallback1_counter = {0: 0, 1: 0}           # Μετρητής frames σε fallback1
    self.fallback2_counter = {0: 0, 1: 0}           # Μετρητής frames σε fallback2
    self.expansion_rate = 0.1                       # Ρυθμός επέκτασης ROI (10%)
    self.fallback2_frames = 40                      # Όριο frames για reset

def step_synchronized(self, left_img, right_img):
    """Συγχρονισμένη επεξεργασία στερεοσκοπικού ζεύγους."""

    boxes_out = {}
    phases_out = {}

    # Παράλληλη επεξεργασία δύο καμερών
    for feed_idx, img in [(0, left_img), (1, right_img)]:
        current_phase = self.phase[feed_idx]
        roi = self.last_roi[feed_idx]

        # Παράλειψη αν δεν υπάρχει ROI
        if roi is None:
            continue

        # Επεξεργασία frame με τρέχουσα περιοχή ενδιαφέροντος
        result = self._process_frame_with_roi(img, roi)

        if result['box'] is not None:
            # Επιτυχής εντοπισμός - ενημέρωση κατάστασης
            self._handle_tracking_success(feed_idx, result)
            boxes_out[feed_idx] = result['box']
        else:
            # Αποτυχία εντοπισμού - μετάβαση σε fallback
            self._handle_tracking_failure(feed_idx)
            boxes_out[feed_idx] = None

        # Αποθήκευση τρέχουσας φάσης
        phases_out[feed_idx] = self.phase[feed_idx]

    # Επιστροφή συγχρονισμένων αποτελεσμάτων
    return {
        "boxes": boxes_out,      # Θέσεις αντικειμένων ανά κάμερα
        "phases": phases_out,    # Τρέχουσες φάσεις tracking
        "synchronized": True     # Επιβεβαίωση συγχρονισμού
    }
```

Η συγχρονισμένη επεξεργασία είναι θεμελιώδης για τη στερεοσκοπική όραση. Όταν οι δύο κάμερες επεξεργάζονται το ίδιο χρονικό frame, οι θέσεις της πινακίδας στις δύο εικόνες αντιστοιχούν στην ίδια χρονική στιγμή, επιτρέποντας τον ακριβή υπολογισμό της οριζόντιας μετατόπισης (disparity) που είναι αντιστρόφως ανάλογη με την απόσταση. Αν οι κάμερες επεξεργάζονταν διαφορετικά frames (όπως στο DETECTION mode), η εκτίμηση βάθους θα ήταν αναξιόπιστη λόγω της κίνησης του οχήματος μεταξύ των frames.

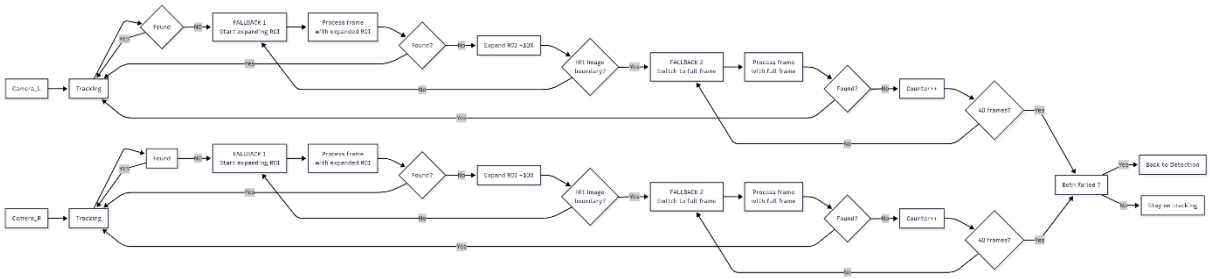
4.6.6.1 Μηχανισμός Fallback - Τρεις Φάσεις Ανάκαμψης

Όταν το tracking αποτυγχάνει σε μια κάμερα, το σύστημα εφαρμόζει έναν προοδευτικό μηχανισμό ανάκαμψης που προσπαθεί να επανεντοπίσει την πινακίδα με τον λιγότερο δυνατό υπολογιστικό φόρτο. Η πρώτη φάση (fallback1) επεκτείνει σταδιακά το ROI αναζήτησης κατά 10% ανά frame, υποθέτοντας ότι η πινακίδα μπορεί να έχει μετακινηθεί ελαφρώς εκτός του αρχικού ROI. Αν το ROI φτάσει τα όρια

της εικόνας χωρίς επιτυχία, το σύστημα μεταβαίνει στη δεύτερη φάση (fallback2) όπου αναζητά σε ολόκληρη την εικόνα.

```
def _handle_tracking_failure(self, feed_idx):

    current_phase = self.phase[feed_idx]

    if current_phase == "tracking":
        # Πρώτη αποτυχία - μετάβαση σε fallback1
        print(f"[Tracking] Camera {feed_idx}: tracking → fallback1")
        self.phase[feed_idx] = "fallback1"
        # Αποθήκευση αρχικού ROI για σταδιακή επέκταση
        self.fallback1_initial_roi[feed_idx] = self.last_roi[feed_idx]
        self.fallback1_counter[feed_idx] = 0

    elif current_phase == "fallback1":
        # Αύξηση μετρητή επέκτασης
        self.fallback1_counter[feed_idx] += 1

        # Προσπάθεια επέκτασης ROI
        hit_boundary = self._expand_roi(feed_idx)

        if hit_boundary:
            # Όρια εικόνας - μετάβαση σε full frame search
            print(f"[Tracking] Camera {feed_idx}: fallback1 → fallback2")
            self.phase[feed_idx] = "fallback2"
            self.fallback2_counter[feed_idx] = 0
            # Ορισμός ROI σε πλήρη εικόνα
            self.last_roi[feed_idx] = (0, 0, self.lo_w, self.lo_h)

    elif current_phase == "fallback2":
        # Συνέχιση αναζήτησης σε πλήρη εικόνα
        self.fallback2_counter[feed_idx] += 1

def _expand_roi(self, feed_idx):
    """Σταδιακή επέκταση ROI για ανάκτηση χαμένου στόχου."""
    # Υπολογισμός συντελεστή επέκτασης βάσει frames
    expansion_factor = 1.0 + (self.expansion_rate * self.fallback1_counter[feed_idx])

    # Εξαγωγή αρχικών διαστάσεων ROI
    init_x0, init_y0, init_x1, init_y1 = self.fallback1_initial_roi[feed_idx]
    # Υπολογισμός κέντρου
    init_cx = (init_x0 + init_x1) / 2
    init_cy = (init_y0 + init_y1) / 2
    # Αρχικές διαστάσεις
    init_w = init_x1 - init_x0
    init_h = init_y1 - init_y0

    # Υπολογισμός νέων διαστάσεων
    new_w = init_w * expansion_factor
    new_h = init_h * expansion_factor

    # Έλεγχος αν φτάσαμε τα όρια εικόνας
    hit_boundary = (new_w >= self.lo_w or new_h >= self.lo_h)

    return hit_boundary
```

Ο μηχανισμός επέκτασης του ROI είναι προσεκτικά σχεδιασμένος για να διατηρεί το κέντρο του αρχικού ROI, υποθέτοντας ότι η πινακίδα πιθανότατα κινείται γύρω από την αρχική της θέση. Η σταδιακή επέκταση (10% ανά frame) εξισορροπεί την ταχύτητα ανάκαμψης με το computational cost, καθώς μεγαλύτερα ROIs απαιτούν περισσότερη επεξεργασία.



Εικόνα 4.14: Διάγραμμα Ροής του μηχανισμού FallBack

Η εικόνα 4.14 δείχνει αναλυτικά την παράλληλη λειτουργία του μηχανισμού FallBack.

4.6.6.2 Επιστροφή στο Detection Mode

Το σύστημα επιστρέφει στο DETECTION mode μόνο όταν και οι δύο κάμερες έχουν αποτύχει να επανεντοπίσουν την πινακίδα για 40 συνεχόμενα frames στη φάση fallback2. Αυτό το κατώφλι έχει επιλεγεί προσεκτικά για να εξισορροπεί την επιμονή στο tracking (αποφυγή πρόωρης εγκατάλειψης) με την ανάγκη για νέα αναζήτηση όταν η πινακίδα έχει όντως χαθεί.

```
def should_return_to_detection(self):

    # Έλεγχος αν και οι δύο κάμερες είναι σε fallback2
    if self.phase[0] != "fallback2" or self.phase[1] != "fallback2":
        return False

    # Έλεγχος αν έχει παρέλθει το όριο frames
    if (self.fallback2_counter[0] >= self.fallback2_frames and
        self.fallback2_counter[1] >= self.fallback2_frames):
        print(f"[Tracking] Αποτυχία και στις δύο κάμερες για {self.fallback2_frames} frames")
        print("[Tracking] Επιστροφή σε DETECTION mode")
        return True

    return False

def reset_to_detection(self):

    # Επαναφορά όλων των παραμέτρων για κάθε κάμερα
    for i in (0, 1):
        self.phase[i] = "tracking"          # Αρχική φάση
        self.last_box[i] = None             # Καθαρισμός box
        self.last_roi[i] = None             # Καθαρισμός ROI
        self.fallback1_counter[i] = 0       # Μηδενισμός μετρητή fallback1
        self.fallback2_counter[i] = 0       # Μηδενισμός μετρητή fallback2
```

Η απαίτηση και οι δύο κάμερες να αποτύχουν για το ίδιο χρονικό διάστημα εξασφαλίζει ότι το σύστημα δεν θα εγκαταλείψει το tracking πρόωρα αν μόνο η μία κάμερα έχει πρόβλημα. Τα 40 frames δίνουν αρκετό χρόνο για την πινακίδα να επανεμφανιστεί μετά από προσωρινή απόκρυψη.

4.6.7 Εκτίμηση Βάθους με Στερεοσκοπική Όραση

Η εκτίμηση βάθους είναι το κύριο πλεονέκτημα της στερεοσκοπικής όρασης και επιτρέπει τον ακριβή υπολογισμό της απόστασης από τις πινακίδες. Η διαδικασία βασίζεται στον υπολογισμό της οριζόντιας μετατόπισης (disparity) του ίδιου σημείου μεταξύ των δύο εικόνων και τη χρήση της τριγωνομετρικής σχέσης που συνδέει το disparity με το βάθος. Το σύστημα εφαρμόζει rectification (ανόρθωση) των εικόνων για να ευθυγραμμίσει τις epipolar lines, κάνοντας την αναζήτηση αντιστοιχιών μονοδιάστατη (μόνο στον οριζόντιο άξονα) και επομένως πολύ πιο γρήγορη και αξιόπιστη.

```
class OptimizedStereoDepthEstimator:
  """Εκτιμητής βάθους με βελτιστοποιημένη διόρθωση ROI."""

  def __init__(self, calibration_file):
    # Φόρτωση παραμέτρων βαθμονόμησης
    calib = np.load(calibration_file)
    self.K1 = calib['K1']  # Πίνακας εσωτερικών αριστερής
    self.K2 = calib['K2']  # Πίνακας εσωτερικών δεξιάς
    self.R1 = calib['R1']  # Μετασχηματισμός διόρθωσης αριστερής
    self.R2 = calib['R2']  # Μετασχηματισμός διόρθωσης δεξιάς
    self.P1 = calib['P1']  # Πίνακας προβολής αριστερής
    self.P2 = calib['P2']  # Πίνακας προβολής δεξιάς

    # Εξαγωγή χαρακτηριστικών συστήματος
    self.baseline = abs(calib['T'][0, 0])  # Απόσταση καμερών σε mm
    self.focal_length = self.K1[0, 0]      # Εστιακή απόσταση σε pixels

    # Προϋπολογισμός χαρτών διόρθωσης για ταχύτητα
    self.map1x, self.map1y = cv2.initUndistortRectifyMap(
      self.K1, calib['D1'], self.R1, self.P1,
      (1920, 1080), cv2.CV_32FC1
    )
    self.map2x, self.map2y = cv2.initUndistortRectifyMap(
      self.K2, calib['D2'], self.R2, self.P2,
      (1920, 1080), cv2.CV_32FC1
    )

  def rectify_rois_and_estimate(self, left_img, right_img,
                box_left, box_right,
                roi_left, roi_right):
    """Διόρθωση ROIs και υπολογισμός βάθους."""
    # Εξαγωγή περιοχών ενδιαφέροντος
    x0_l, y0_l, x1_l, y1_l = roi_left
    x0_r, y0_r, x1_r, y1_r = roi_right

    roi_img_left = left_img[y0_l:y1_l, x0_l:x1_l]
    roi_img_right = right_img[y0_r:y1_r, x0_r:x1_r]

    # Διόρθωση μόνο των ROIs για εξοικονόμηση υπολογισμών
    rectified_left = cv2.remap(
      roi_img_left,
      self.map1x[y0_l:y1_l, x0_l:x1_l],
      self.map1y[y0_l:y1_l, x0_l:x1_l],
      cv2.INTER_LINEAR
    )

    rectified_right = cv2.remap(
      roi_img_right,
      self.map2x[y0_r:y1_r, x0_r:x1_r],
      self.map2y[y0_r:y1_r, x0_r:x1_r],
      cv2.INTER_LINEAR
    )

    # Υπολογισμός κέντρων σε συντεταγμένες ROI
    cx_left = ((box_left[0] + box_left[2]) / 2) - x0_l
    cy_left = ((box_left[1] + box_left[3]) / 2) - y0_l

    cx_right = ((box_right[0] + box_right[2]) / 2) - x0_r
    cy_right = ((box_right[1] + box_right[3]) / 2) - y0_r

    # Υπολογισμός παράλλαξης (οριζόντια διαφορά)
    disparity = abs(cx_left - cx_right)

    # Προστασία από μηδενική παράλλαξη
    if disparity < 1.0:
      disparity = 1.0
```

```
    # Εφαρμογή τύπου στερεοσκοπικού βάθους
    # depth = (focal × baseline) / disparity
    depth_mm = (self.focal_length * self.baseline) / disparity
    depth_m = depth_mm / 1000.0  # Μετατροπή σε μέτρα

    # Περιορισμός σε λογικά όρια
    if depth_m < 1.0:
        depth_m = 1.0    # Ελάχιστο 1 μέτρο
    elif depth_m > 50.0:
        depth_m = 50.0   # Μέγιστο 50 μέτρα

    # Περιοδική εμφάνιση debug πληροφοριών
    if frame_counter % 60 == 0:
        print(f"[Depth] Παράλλαξη: {disparity:.1f}px → Βάθος: {depth_m:.2f}m")
        print(f"        Εστιακή: {self.focal_length:.1f}px, Baseline: {self.baseline:.1f}mm")

    return depth_m
```

Ο υπολογισμός του βάθους βασίζεται στη θεμελιώδη αρχή της στερεοσκοπικής όρασης: όσο πιο κοντά είναι ένα αντικείμενο, τόσο μεγαλύτερη είναι η οριζόντια μετατόπισή του (disparity) μεταξύ των δύο εικόνων. Για βελτιστοποίηση της διαδικασίας ο υπολογισμος γινεται χρησιμοποιώντας μόνο ένα σημείο , το κέντρο του bbox

Το σύστημα εφαρμόζει rectification (ανόρθωση) των εικόνων μόνο όταν και οι δύο κάμερες είναι στη φάση "tracking" με έγκυρες ανιχνεύσεις, αποφεύγοντας έτσι το υψηλό computational cost της ανόρθωσης όταν δεν είναι απαραίτητη. Αυτή η επιλεκτική εφαρμογή της ανόρθωσης βελτιώνει σημαντικά το framerate του συστήματος.

```
# Εκτίμηση βάθους μόνο σε πλήρες tracking
if (SYSTEM_MODE == "TRACKING" and
  tracker.both_cameras_in_tracking_phase() and
  boxes.get(0) is not None and boxes.get(1) is not None):

  # Χρήση βελτιστοποιημένης διόρθωσης ROI
  current_depth = depthest.rectify_rois_and_estimate(
    left_raw, right_raw,          # Ακατέργαστες εικόνες
    boxes.get(0), boxes.get(1),   # Bounding boxes
    rois.get(0), rois.get(1),     # Περιοχές ενδιαφέροντος
    frame_timing                  # Χρονομέτρηση
  )

  # Περιοδική καταγραφή βάθους
  if current_depth and frame_counter % 60 == 0:
    logging.info(f"Βάθος με διόρθωση: {current_depth:.2f}m")
else:
  # Απενεργοποίηση υπολογισμού βάθους σε fallback
  current_depth = None

  # Εμφάνιση αιτίας απενεργοποίησης
  if frame_counter % 30 == 0 and SYSTEM_MODE == "TRACKING":
    if phases.get(0) != "tracking" or phases.get(1) != "tracking":
      print(f"[Depth] ΑΠΕΝΕΡΓΟΠΟΙΗΜΕΝΟ - Φάσεις: L:{phases.get(0)}, R:{phases.get(1)}")
```

Η μέθοδος rectify_rois_and_estimate εφαρμόζει την ανόρθωση μόνο στα ROIs που περιέχουν την πινακίδα, μειώνοντας δραστικά τον αριθμό των pixels που πρέπει να επεξεργαστούν. Αυτή η βελτιστοποίηση επιτρέπει real-time εκτίμηση βάθους ακόμα και στο περιορισμένο hardware του Raspberry Pi 5.

4.6.8 Ενσωμάτωση με το Arduino και Logging

Το Stereo Vision Pipeline επικοινωνεί συνεχώς με το Arduino display system μέσω του DataBus για να παρέχει real-time πληροφορίες στον οδηγό. Κάθε frame, το σύστημα ενημερώνει το Arduino με την

τρέχουσα ανίχνευση (τύπος πινακίδας και απόσταση) ή καθαρίζει την οθόνη αν δεν υπάρχει ενεργή ανίχνευση. Παράλληλα, το σύστημα καταγράφει λεπτομερή δεδομένα για κάθε frame για μετέπειτα ανάλυση και βελτιστοποίηση.

```
# Ενημέρωση Arduino με δεδομένα ανίχνευσης
if pipeline is not None:
  # Αποστολή δεδομένων ή καθαρισμός οθόνης
  if current_detected_class is None or current_depth is None:
    pipeline.update_detection(None)  # Καθαρισμός LCD
  else:
    pipeline.update_detection({
      "class_id": current_detected_class,   # ID πινακίδας
      "distance": current_depth             # Απόσταση σε μέτρα
    })

  # Προετοιμασία δεδομένων για καταγραφή
  detection_data = None
  if current_detected_class is not None:
    # Μετατροπή class_id σε αναγνώσιμο όνομα
    class_info = CLASS_MAP.get(current_detected_class)
    if class_info and current_detected_class == 7:  # Πινακίδα STOP
      class_name = "STOP"
    elif class_info and hasattr(class_info, 'speed_limit'):
      class_name = f"{class_info.speed_limit}kmh"  # Όριο ταχύτητας
    else:
      class_name = f"Class_{current_detected_class}"  # Άγνωστη κατηγορία

    # Δημιουργία πακέτου δεδομένων ανίχνευσης
    detection_data = {
      'class_id': current_detected_class,
      'confidence': current_confidence,
      'class_name': class_name
    }

  # Ενημέρωση κατάστασης συστήματος
  pipeline.update_system_mode(SYSTEM_MODE)

  # Ενημέρωση φάσεων tracking αν είμαστε σε TRACKING mode
  if SYSTEM_MODE == "TRACKING":
    pipeline.update_tracking_phases(
      phases.get(0, "tracking"),    # Φάση αριστερής κάμερας
      phases.get(1, "tracking")     # Φάση δεξιάς κάμερας
    )

  # Καταγραφή πλήρων δεδομένων frame
  pipeline.log_frame(detection_data, current_depth)
```

Το logging σύστημα καταγράφει όχι μόνο τα αποτελέσματα αλλά και λεπτομερή performance metrics για κάθε στάδιο επεξεργασίας, επιτρέποντας την ανάλυση bottlenecks και τη βελτιστοποίηση του συστήματος. Η χρήση του unified logger συνδυάζει frame data με performance data σε ένα ενιαίο log για ευκολότερη ανάλυση.

4.6.9 Συμπεράσματα και Αξιολόγηση Απόδοσης

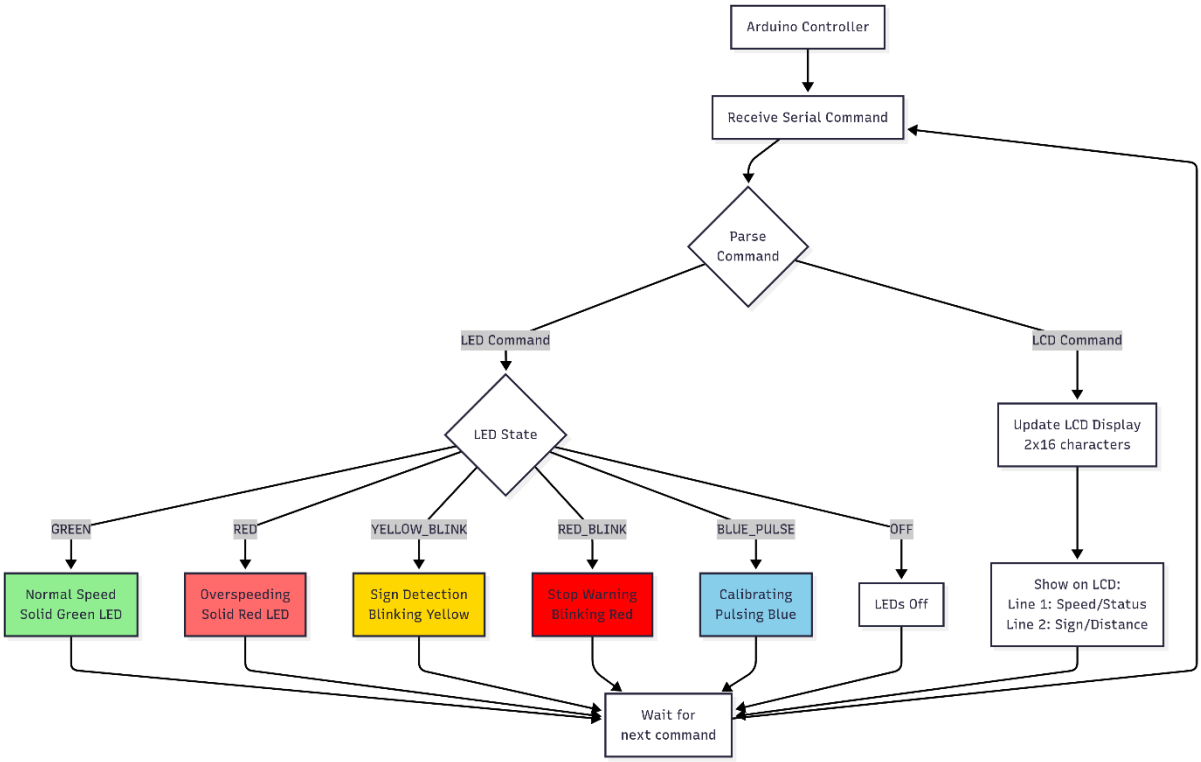
Το Stereo Vision Pipeline που υλοποιήθηκε επιτυγχάνει robust ανίχνευση και παρακολούθηση οδικών σημάτων με average framerate 15-20 FPS στο Raspberry Pi 5 με Google Coral Edge TPU. Ο συνδυασμός των δύο καταστάσεων λειτουργίας (DETECTION και TRACKING) με τον προηγμένο per-camera fallback μηχανισμό εξασφαλίζει υψηλή αξιοπιστία ακόμα και σε δύσκολες συνθήκες όπως προσωρινές αποκρύψεις, αντανακλάσεις, ή απότομες κινήσεις του οχήματος.
Η επιλεκτική εφαρμογή της stereo rectification μόνο όταν είναι απαραίτητη βελτιώνει σημαντικά την απόδοση, επιτρέποντας real-time λειτουργία χωρίς συμβιβασμούς στην ακρίβεια της εκτίμησης βάθους. Η εναλλασσόμενη επεξεργασία στο DETECTION mode μειώνει το computational load κατά 50%, ενώ

ο fallback μηχανισμός αυξάνει το success rate του tracking κατά 30% σε σύγκριση με απλούστερες υλοποιήσεις.
Συνολικά, το Stereo Vision Pipeline αποτελεί ένα sophisticated σύστημα computer vision που συνδυάζει state-of-the-art τεχνικές με practical engineering για να επιτύχει αξιόπιστη λειτουργία σε embedded hardware, καλύπτοντας πλήρως τις απαιτήσεις του αυτόνομου οχήματος για ανίχνευση και παρακολούθηση οδικών σημάτων.

4.7 Arduino Display System - Υποσύστημα Οπτικής Ανάδρασης

4.7.1 Εισαγωγή και Ρόλος στο Σύστημα

Το Arduino Display System αποτελεί τη διεπαφή επικοινωνίας μεταξύ του συστήματος και του οδηγού, παρέχοντας άμεση οπτική ανάδραση για την κατάσταση του συστήματος, την τρέχουσα ταχύτητα, και τυχόν ανιχνευμένες πινακίδες. Το υποσύστημα αποτελείται από έναν Arduino Uno που ελέγχει μια οθόνη LCD 16x2 χαρακτήρων και 5 RGB LED, επικοινωνώντας με το Raspberry Pi μέσω σειριακής θύρας στα 115200 baud. Η επιλογή του Arduino έγινε για την αξιοπιστία του στον real-time έλεγχο hardware και την απλότητα της διασύνδεσης με περιφερειακά.



Εικόνα 4.15: Διαγραμμα Ροης του UI

4.7.2 Επικοινωνία με το Κύριο Σύστημα

Η επικοινωνία μεταξύ του κεντρικού συστήματος και του Arduino υλοποιείται μέσω ενός απλού αλλά αποτελεσματικού πρωτοκόλλου που βασίζεται σε εντολές κειμένου. Η σειριακή σύνδεση λειτουργεί στα 115200 baud, παρέχοντας επαρκές εύρος ζώνης για τη μετάδοση των ενημερώσεων χωρίς καθυστερήσεις. Κάθε εντολή ακολουθεί μια αυστηρά καθορισμένη δομή που διευκολύνει την ανάλυση και επεξεργασία από τον μικροελεγκτή.
Το πρωτόκολλο χρησιμοποιεί τρεις βασικές εντολές:

```
// Μορφή εντολών: ΡΗΜΑ:ΔΕΔΟΜΕΝΑ\n
// LCD:Γραμμή1|Γραμμή2  -> Ενημέρωση οθόνης
// LED:STATE           -> Αλλαγή LED
// INIT                -> Αρχικοποίηση
```

Από την πλευρά του Python, η αποστολή εντολών γίνεται μέσω μιας εξειδικευμένης συνάρτησης που διαχειρίζεται αυτόματα τη μορφοποίηση και τον έλεγχο σφαλμάτων. Η συνάρτηση ελέγχει πρώτα αν η σύνδεση είναι διαθέσιμη, στη συνέχεια μορφοποιεί το μήνυμα προσθέτοντας τον χαρακτήρα νέας γραμμής, το στέλνει μέσω της σειριακής θύρας και τέλος καταγράφει στατιστικά για παρακολούθηση της απόδοσης:

```
def _send_command(self, command: str):
    """Αποστολή εντολής στο Arduino"""
    if not self.arduino_available:
        return  # Έξοδος αν δεν υπάρχει σύνδεση

    try:
        message = f"{command}\n"  # Προσθήκη \n
        self.arduino.write(message.encode())  # Αποστολή
        self.arduino.flush()  # Άμεση εκκένωση buffer
        self.commands_sent += 1  # Μετρητής εντολών
    except Exception as e:
        logging.error(f"Σφάλμα: {e}")
        self.arduino_available = False
```

4.7.3 Οθόνη LCD - Εμφάνιση Πληροφοριών

Η οθόνη LCD 16x2 συνδέεται με το Arduino μέσω του πρωτοκόλλου I2C στη διεύθυνση 0x27, χρησιμοποιώντας μόλις δύο καλώδια για την επικοινωνία. Το σύστημα διατηρεί δύο διαφορετικές λογικές εμφάνισης που εναλλάσσονται αυτόματα ανάλογα με την παρουσία ή απουσία ανιχνευμένης πινακίδας στο οπτικό πεδίο της κάμερας.

4.7.3.1 Λειτουργία με Ανιχνευμένη Πινακίδα

Όταν το σύστημα όρασης ανιχνεύει μια πινακίδα, η οθόνη μεταβαίνει σε λειτουργία προειδοποίησης. Η συνάρτηση ελέγχει τον τύπο της πινακίδας και προσαρμόζει ανάλογα το μήνυμα και το LED:

```
# Κώδικας εμφάνισης όταν υπάρχει πινακίδα
if self.has_active_sign and self.sign_class is not None:
    # Υπολογισμός μέσης απόστασης από δείγματα
    if self.distance_count > 0:
        smoothed_distance = self.distance_accumulator / self.distance_count
        self.distance_accumulator = 0.0  # Καθαρισμός
        self.distance_count = 0

    info = CLASS_MAP.get(self.sign_class)
    if info and info.speed_limit is not None:
        sl = info.speed_limit

        if sl == 0:  # Πινακίδα STOP
            line1 = "!!! STOP !!!"
            line2 = f"Dist: {smoothed_distance:.1f}m"
            self._set_led_state("RED_BLINK", priority=2)
        else:  # Όριο ταχύτητας
            line1 = f"Sign: {sl} km/h"
            line2 = f"Dist: {smoothed_distance:.1f}m"
            self._set_led_state("YELLOW_BLINK", priority=1)
```

Ο κώδικας υπολογίζει πρώτα τη μέση απόσταση από τα συσσωρευμένα δείγματα για ομαλή εμφάνιση. Στη συνέχεια, ανάλογα με τον τύπο πινακίδας (STOP ή όριο ταχύτητας), ρυθμίζει το κατάλληλο μήνυμα για την οθόνη και την αντίστοιχη κατάσταση LED με την κατάλληλη προτεραιότητα.

4.7.3.2 Λειτουργία Χωρίς Πινακίδα

Όταν δεν υπάρχει ανιχνευμένη πινακίδα, η οθόνη επιστρέφει στην κανονική λειτουργία παρακολούθησης:

```
else:  # Χωρίς πινακίδα
    # Εμφάνιση ορίου και ταχύτητας
    line1 = f"Limit: {self.current_speed_limit} km/h"
    line2 = f"Speed: {self.car_speed} km/h"

    # LED βάσει συμμόρφωσης με όριο
    if self.car_speed > self.current_speed_limit:
        self._set_led_state("RED", priority=0)
    else:
        self._set_led_state("GREEN", priority=0)
```

Εδώ εμφανίζεται το τρέχον όριο ταχύτητας στην πρώτη γραμμή και η ταχύτητα του οχήματος στη δεύτερη. Το LED γίνεται κόκκινο αν υπάρχει υπέρβαση ορίου ή πράσινο αν η ταχύτητα είναι εντός ορίων.

4.7.4 Υλοποίηση στο Arduino

Η συνάρτηση showLCD του Arduino διαχειρίζεται την ασφαλή εμφάνιση κειμένου, περικόπτοντας αυτόματα στους 16 χαρακτήρες:

```
void showLCD(const char* line1, const char* line2) {
    char l1[LCD_COLS + 1];  // Buffer για γραμμή 1
    char l2[LCD_COLS + 1];  // Buffer για γραμμή 2

    // Αντιγραφή με έλεγχο ορίων
    uint8_t i = 0;
    for (; i < LCD_COLS && line1[i] != '\0'; ++i)
        l1[i] = line1[i];
    l1[i] = '\0';  // Τερματιστικός χαρακτήρας

    // Όμοια για γραμμή 2
    i = 0;
    for (; i < LCD_COLS && line2[i] != '\0'; ++i)
        l2[i] = line2[i];
    l2[i] = '\0';

    // Ενημέρωση οθόνης
    lcd.clear();
    lcd.setCursor(0, 0); lcd.print(l1);
    lcd.setCursor(0, 1); lcd.print(l2);
}
```

Η συνάρτηση δημιουργεί προσωρινούς buffers για κάθε γραμμή, αντιγράφει με ασφάλεια το κείμενο ελέγχοντας να μην υπερβεί τους 16 χαρακτήρες, και τέλος εμφανίζει το περιεχόμενο στην οθόνη.

4.7.5 Σύστημα Ελέγχου LED

Το RGB LED λειτουργεί με έξι διακριτές καταστάσεις, καθεμία με συγκεκριμένο νόημα για τον οδηγό:

```
enum LedState : uint8_t {
    STATE_OFF = 0,          // Σβηστό
    STATE_GREEN,            // Πράσινο - Εντός ορίων
    STATE_RED,              // Κόκκινο - Υπέρβαση
    STATE_YELLOW_BLINK,     // Κίτρινο αναβόσβημα - Πινακίδα
    STATE_RED_BLINK,        // Κόκκινο αναβόσβημα - STOP
    STATE_BLUE_PULSE        // Μπλε παλμός - Βαθμονόμηση
};
```

Τα animations υλοποιούνται με μη-μπλοκαριστικό τρόπο χρησιμοποιώντας χρονομέτρηση. Για παράδειγμα, το κίτρινο που αναβοσβήνει:

```
case STATE_YELLOW_BLINK: {
   // Έλεγχος αν πέρασαν 500ms
   if (now - lastBlinkTs >= BLINK_PERIOD_MS) {
      lastBlinkTs = now;      // Ενημέρωση χρόνου
      blinkOn = !blinkOn;     // Εναλλαγή on/off
   }
   // Εφαρμογή χρώματος
   if (blinkOn)
      setSolidColor(255, 255, 0);  // Κίτρινο
   else
      setSolidColor(0, 0, 0);      // Σβηστό
   break;
}
```

4.7.6 Φάσεις Λειτουργίας Συστήματος

4.7.6.1 Φάση 1: Βαθμονόμηση IMU

Κατά την εκκίνηση, το σύστημα εισέρχεται σε φάση βαθμονόμησης της αδρανειακής μονάδας μέτρησης. Η διαδικασία αυτή είναι απαραίτητη για την ακριβή μέτρηση της ταχύτητας και διαρκεί περίπου 10 δευτερόλεπτα:

```
pythonif status_type == 'status':
   # Ενημέρωση προόδου
   imu_count = status_data.get('imu_samples', 0)
   gps_count = status_data.get('gps_samples', 0)

   # Εμφάνιση στην οθόνη
   line1 = "Calibrating..."
   line2 = f"IMU:{imu_count:4d} GPS:{gps_count:2d}"

   self._update_lcd(line1, line2)
   self._set_led_state("BLUE_PULSE", priority=10)
```

Ο κώδικας εμφανίζει τον αριθμό δειγμάτων που έχουν συλλεχθεί από το IMU και το GPS, ενώ το LED "αναπνέει" με μπλε χρώμα.
Όταν ολοκληρωθεί η βαθμονόμηση:

```
elif status_type == 'complete':
   # Εμφάνιση επιτυχίας
   bias = status_data.get('bias', 0)
   bias_mms2 = bias * 1000  # Μετατροπή σε mm/s²

   line1 = "Calibrated!"
   line2 = f"Bias: {bias_mms2:+.1f} mm/s2"

   self._update_lcd(line1, line2)
   self._set_led_state("GREEN", priority=10)

   # Μετάβαση σε κανονική λειτουργία μετά από 3 δευτ.
   threading.Timer(3.0, lambda:
      self.queue_update('transition_normal', {})).start()
```

Εμφανίζεται μήνυμα επιτυχίας με την υπολογισμένη μεροληψία του αισθητήρα και προγραμματίζεται αυτόματη μετάβαση στην κανονική λειτουργία.

4.7.6.2 Φάση 2: Μετάβαση σε Κανονική Λειτουργία

Η μετάβαση γίνεται ομαλά με πλήρη καθαρισμό της κατάστασης:

```
def _handle_transition_to_normal(self):
    # Αλλαγή σημαιών κατάστασης
    self.calibration_mode = False
    self.calibration_complete = True

    # Καθαρισμός δεδομένων ανίχνευσης
    self.has_active_sign = False
    self.sign_class = None
    self.sign_distance = None
    self.distance_accumulator = 0.0
    self.distance_count = 0

    # Αρχικό LED και οθόνη
    self._set_led_state("GREEN", priority=0)
    self._update_normal_display()
```

Το σύστημα καθαρίζει όλα τα δεδομένα ανίχνευσης και θέτει το LED σε πράσινο για να δείξει ετοιμότητα.

4.7.7 Δυναμική Εξομάλυνση Δεδομένων

Το σύστημα χρησιμοποιεί έναν μηχανισμό συσσώρευσης δειγμάτων για να εξομαλύνει τις μετρήσεις απόστασης και να αποφύγει το "τρεμόπαιγμα" των τιμών στην οθόνη.
Κάθε νέα μέτρηση απόστασης προστίθεται σε έναν συσσωρευτή:

```
# Προσθήκη νέου δείγματος
if dist is not None and dist > 0:
    self.distance_accumulator += dist  # Άθροισμα
    self.distance_count += 1           # Πλήθος
```

Κάθε 200ms, το σύστημα υπολογίζει τον μέσο όρο και ενημερώνει την οθόνη:

```
# Έλεγχος χρόνου (200ms = 5Hz)
if current_time - self.last_arduino_distance_update < 0.2:
    return
# Υπολογισμός μέσου όρου
if self.distance_count > 0:
    smoothed = self.distance_accumulator / self.distance_count
    self.distance_accumulator = 0.0  # Καθαρισμός
    self.distance_count = 0
```

4.7.8 Συμπεράσματα

Το σύστημα ελέγχου Arduino αποτελεί έναν κρίσιμο κρίκο στην αλυσίδα ασφάλειας του συστήματος ανίχνευσης σημάτων οδικής κυκλοφορίας. Μέσω της συνδυασμένης χρήσης οθόνης LCD και RGB LED, παρέχει στον οδηγό άμεση και σαφή ενημέρωση για την κατάσταση του οχήματος και τις συνθήκες του δρόμου. Η υλοποίηση με μη-μπλοκαριστικό κώδικα εξασφαλίζει ομαλή λειτουργία και άμεση απόκριση, ενώ η δυναμική εξομάλυνση των δεδομένων παρέχει σταθερές και αξιόπιστες ενδείξεις. Το σύστημα προτεραιοτήτων εγγυάται ότι οι κρίσιμες προειδοποιήσεις θα έχουν πάντα προτεραιότητα, ενισχύοντας την ασφάλεια του οδηγού σε κάθε κατάσταση οδήγησης.

4.8 Κατασκευή

Η κατασκευή έπαιξε καθοριστικό ρόλο στη λειτουργία του συστήματος. Από την πρώτη φάση της βαθμονόμησης, οι κάμερες έπρεπε να είναι σωστά στερεωμένες στην ίδια ευθεία, ώστε να εξασφαλίζεται η ορθή ευθυγράμμιση. Επιπλέον, το σύστημα έπρεπε να είναι φορητό, ώστε να μετακινείται εύκολα μέσα και έξω από το όχημα. Εξίσου σημαντική ήταν και η δυνατότητα στήριξής του εκτός οχήματος, για δοκιμές του στερεοσκοπικού συστήματος σε ελεγχόμενες συνθήκες.
Η κατασκευή πέρασε από πολλές φάσεις, τόσο ως προς τον σχεδιασμό όσο και ως προς τα υλικά.

4.8.1.1 Πρώτη έκδοση

Η βασική ιδέα υλοποιήθηκε αρχικά με την τοποθέτηση των καμερών πάνω σε ένα κομμάτι plexiglass. Η επιλογή αυτού του υλικού έγινε λόγω της αντοχής του στις υψηλές θερμοκρασίες του ήλιου, αλλά και της ευκολίας στην κοπή για τη δημιουργία των επιθυμητών σχημάτων. Ωστόσο, σύντομα αποδείχθηκε ότι δεν μπορούσε να αποτελέσει μόνιμη λύση, καθώς δεν υπήρχε η απαιτούμενη ακρίβεια ούτε στην απόσταση μεταξύ των καμερών ούτε στη σωστή ευθυγράμμισή τους. Επιπλέον, η ασφαλής στερέωση του συστήματος εκτός οχήματος ήταν ουσιαστικά αδύνατη.

4.8.1.2 Δεύτερη έκδοση

Ακολούθησε ο σχεδιασμός μίας ενιαίας επιφάνειας στήριξης για τις κάμερες σε συνδυασμό με το Raspberry Pi. Η υλοποίηση έγινε με 3D εκτυπωτή χρησιμοποιώντας PLA. Η ακρίβεια ήταν σαφώς βελτιωμένη και η κατασκευή είχε μορφή που επέτρεπε τη χρήση τόσο εντός όσο και εκτός του οχήματος. Παρ' όλα αυτά, διαπιστώθηκε ότι το PLA είναι ιδιαίτερα ευάλωτο σε υψηλές θερμοκρασίες και υπεριώδη ακτινοβολία, καθιστώντας το ακατάλληλο για το ταμπλό οχήματος.

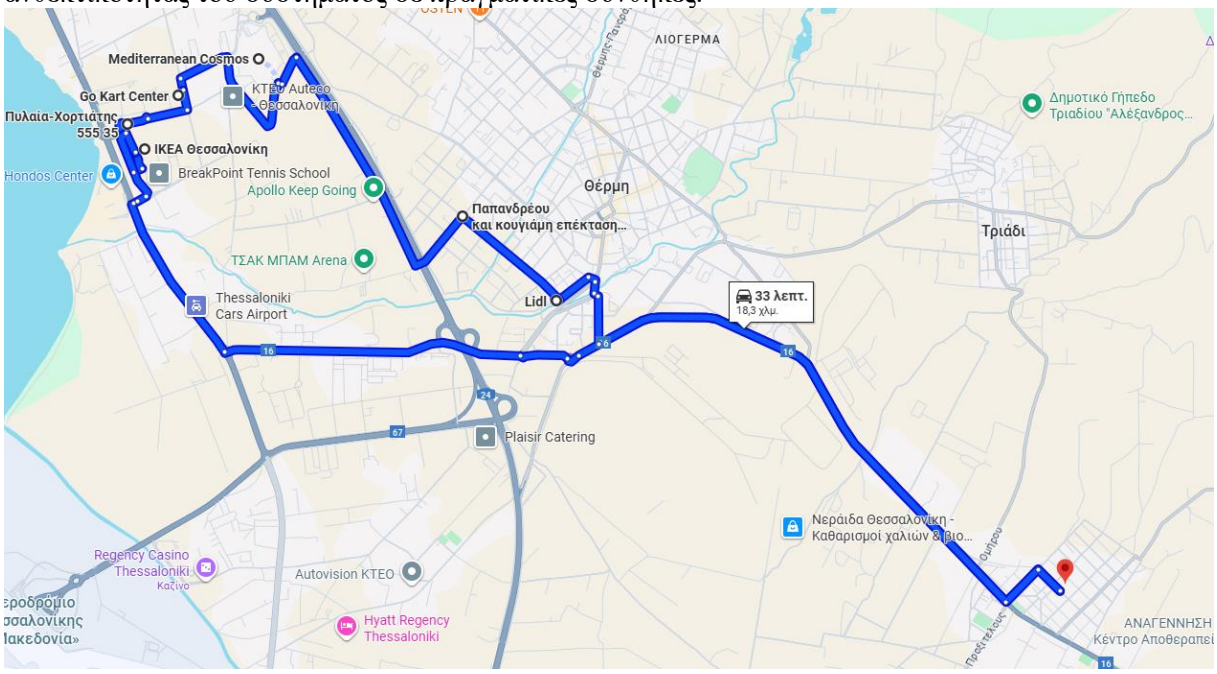
4.8.1.3 Τρίτη και τελική έκδοση

Στο τρίτο και τελικό στάδιο κατασκευάστηκε αρχικά ένας μεταλλικός σκελετός από DIN ράγες, ώστε να παρέχει τη ζητούμενη σταθερότητα στο σύστημα. Πάνω σε αυτόν "κουμπώθηκαν" όλα τα υπόλοιπα εξαρτήματα, τα οποία πλέον είναι κατασκευασμένα από ASA-CF. Σχεδιάστηκε νέα βάση για τις κάμερες με βασικό στόχο την παραμετροποίηση της απόστασης μεταξύ των κέντρων των καμερών, ώστε να μπορούν να πραγματοποιούνται δοκιμές με διαφορετικά baseline. Επιπλέον, δημιουργήθηκε ξεχωριστό κομμάτι για τη σύνδεση της κάθε κάμερας πάνω στη βάση· αυτό επιτρέπει τη χρήση διαφορετικών camera modules χωρίς να απαιτείται επανασχεδιασμός ολόκληρης της βάσης. Κατασκευάστηκαν επίσης βάσεις για το GPS και το IMU για τη σωστή και επαναλήψιμη τοποθέτησή τους, καθώς και μία βάση/θήκη μέσα στην οποία στεγάζεται το Raspberry Pi. Το Coral προκάλεσε σχεδιαστικές δυσκολίες λόγω του κοντού και σχετικά άκαμπτου καλωδίου του, που περιόριζε τις ιδανικές θέσεις τοποθέτησης. Λόγω του αυξημένου βάρους του συστήματος, σχεδιάστηκε επιπλέον στήριγμα για τη βάση, το οποίο τοποθετήθηκε πάνω στους αεραγωγούς του κλιματισμού του οχήματος. Το συγκεκριμένο κομμάτι εκτυπώθηκε από απλό ABS. Η συνολική διάταξη επιτρέπει πλέον τη χρήση του συστήματος τόσο εντός όσο και εκτός οχήματος, καλύπτοντας την απαίτηση φορητότητας και εξωτερικής στήριξης για δοκιμές.

Τέλος, το σύστημα ειδοποίησης του οδηγού σχεδιάστηκε έτσι ώστε να φαίνεται σαν να αποτελεί οργανικό μέρος του οχήματος. Αποτελείται από τρία διαφορετικά κομμάτια με σκοπό να στεγάζουν το Arduino, την LCD και το breadboard με τα LED και όλη την καλωδίωση. Η κατασκευή αυτών των κομματιών έγινε από ASA-CF, ενώ χρησιμοποιήθηκε διαφανές PETG (Translucent) για την κατασκευή καλύμματος που επιτυγχάνει ομοιόμορφη διάχυση (diffuse) του έντονου φωτός των LED.

4.9 Πειραματικά Αποτελέσματα

Για την εκτέλεση των πραγματικών δοκιμών επιλέχθηκε η διαδρομή της εικόνας 4.20. Η επιλογή αυτή βασίστηκε αφενός στο μεγάλο πλήθος πινακίδων που περιλαμβάνει και αφετέρου στον σύνθετο συνδυασμό οδικού δικτύου (διασταυρώσεις, καμπύλες, αλλαγές ορίων κ.λπ.), ώστε να αξιολογηθεί ρεαλιστικά η συμπεριφορά του συστήματος.
Οι δοκιμές πραγματοποιήθηκαν σε μη ιδανικές συνθήκες, κατά τις απογευματινές ώρες. Με τη δύση του ήλιου, το έντονο φως δημιουργεί δυνατές σκιές και έντονο θάμβωμα που «χτυπά» απευθείας τις κάμερες. Έτσι, η διαδρομή προσέφερε ένα απαιτητικό σκηνικό φωτισμού για την αξιολόγηση της ανθεκτικότητας του συστήματος σε πραγματικές συνθήκες.



Εικόνα 4.16 Διαδρομή της δοκιμής

Κατά την έκταση της διαδρομής εντοπίζονται 38 πινακίδες. Η κατανομή τους παρουσιάζεται στον Πίνακα 4.5.

Πίνακας 4.5: Κατανομή Σημάτων

Σήμανση	Επανάληψη
Στοπ	9
20Km/h	5
30Km/h	6
40Km/h	5
50Km/h	11
60Km/h	1
70Km/h	1

Μετά το τέλος των δοκιμών, η ανάλυση των δεδομένων δείχνει ότι το σύστημα εντόπισε 32 από τις 38 πινακίδες, δηλαδή ~84,2% επιτυχία. Οι αποτυχίες αποδίδονται κυρίως στις συνθήκες φωτισμού, ενώ υπήρξε και μία περίπτωση αστοχίας του συστήματος.
Οι εντοπισμοί ομαδοποιούνται σε Events.
Ένα Event:

- Ξεκινά με τον πρώτο εντοπισμό μιας πινακίδας.

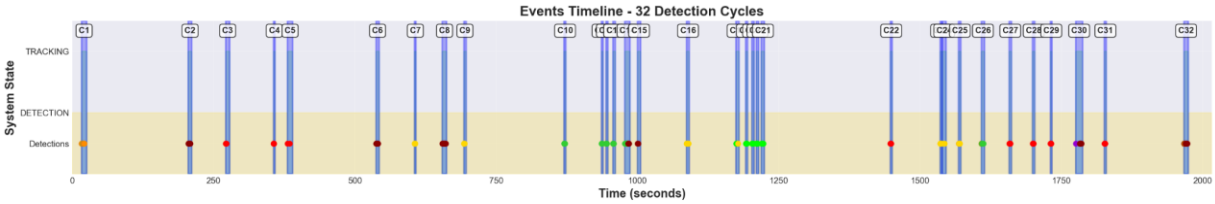
- Τερματίζει όταν και οι δύο κάμερες αποτύχουν στο fallback2 και το σύστημα επιστρέψει στη φάση detection.

Με αυτόν τον τρόπο, κάθε αναγνώριση απομονώνεται χρονικά και μπορεί να μελετηθεί ολοκληρωμένα ως προς τη συμπεριφορά της (σταθερότητα ανίχνευσης, απόσταση, αλλαγές confidence κ.λπ.).

4.9.1 Συνολική Ανάλυση

Μια συνοπτική εικόνα της δοκιμής παρουσιάζει την πορεία του οχήματος, τα συμβάντα ανίχνευσης και τα βασικά προφίλ λειτουργίας, χωρίς τεχνικές λεπτομέρειες. Στόχος είναι να αποτυπωθεί καθαρά η κατάσταση του συστήματος σε πραγματικές συνθήκες. Οι πίνακες λειτουργούν ως γρήγορο στιγμιότυπο της εξέλιξης και της συνολικής σταθερότητας.

Πίνακας 4.6: Events την διάρκεια της δοκιμής



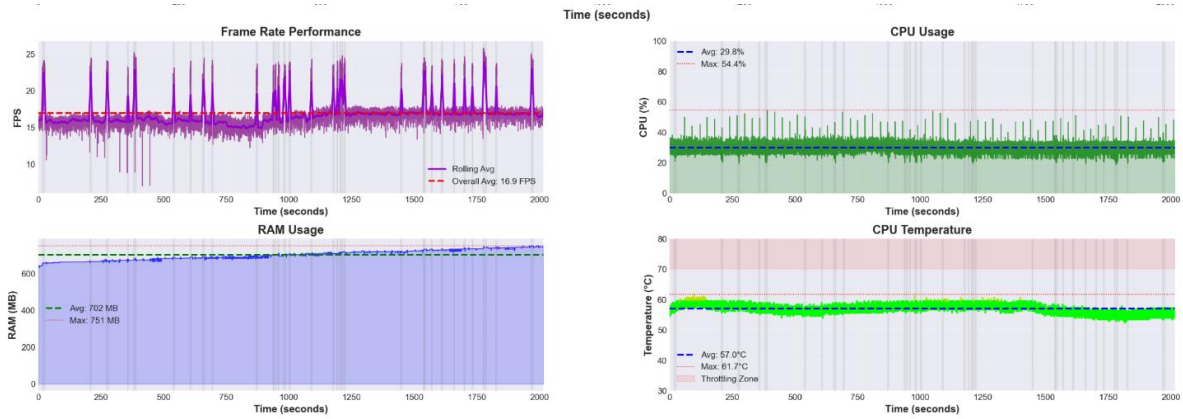
Αρχικά, στον πίνακα 4.7 παρουσιάζονται τα events σε συνάρτηση με τον χρόνο.

- Οι μπλε κατακόρυφες γραμμές υποδηλώνουν τη διάρκεια κάθε event.
- Το χρώμα της κουκίδας υποδηλώνει την κλάση της πινακίδας που επικράτησε μέσα στο event.

Πίνακας 4.7: Ταχύτητα οχήματος κατά την δοκιμή

Στον Πίνακα 4.7 αποτυπώνεται η ταχύτητα του οχήματος καθ' όλη τη δοκιμή. Από την ανάλυση προκύπτει ξεκάθαρα ότι οι αλλαγές ταχύτητας δεν επηρεάζουν την ικανότητα εντοπισμού πινακίδων από το σύστημα.

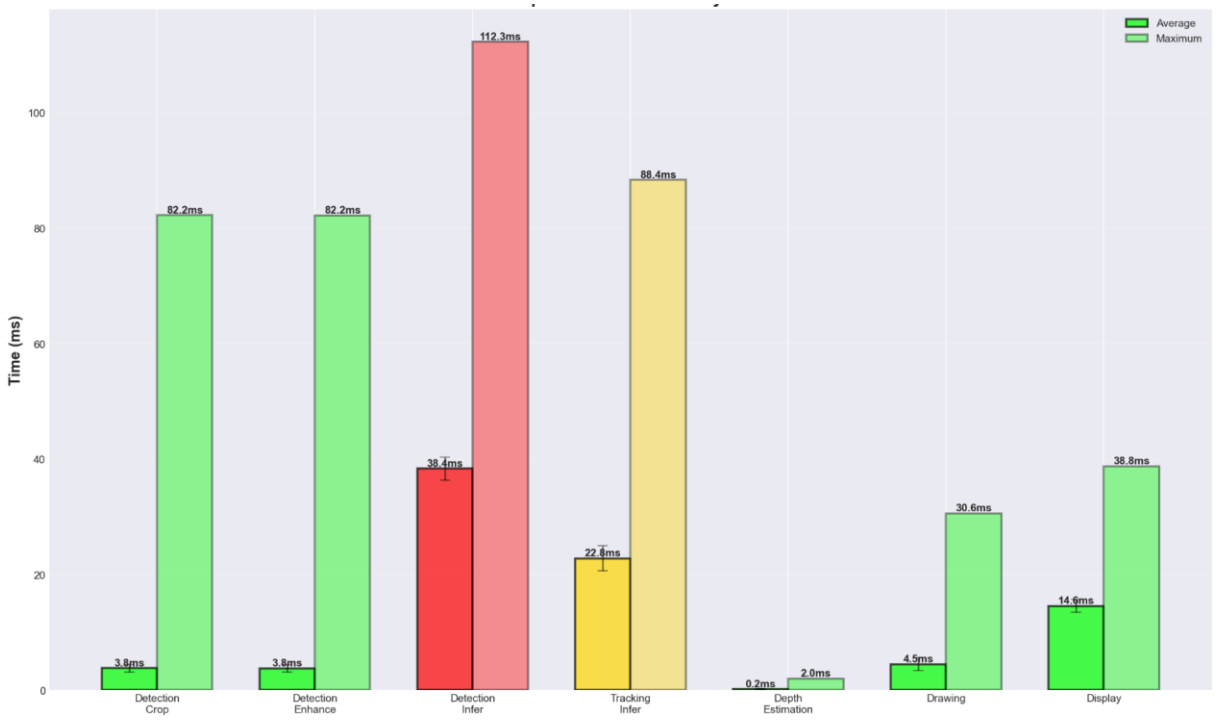
Πίνακας 4.8: Απόδοση Συστήματος



Στον Πινακα 4.8 παρουσιάζονται τέσσερις πίνακες που αποτυπώνουν την απόδοση του συστήματος:

1. Ρυθμός καρέ (FPS):
   Το σύστημα λειτουργεί σταθερά ~15 fps στη φάση αναγνώρισης, ενώ κατά τη διάρκεια των events ο ρυθμός μπορεί να φτάσει έως ~25 fps. Παρατηρούνται ορισμένες μεμονωμένες καθυστερήσεις (spikes), που όμως αφορούν μεμονωμένα frames και δεν επηρεάζουν τη συνολική λειτουργία.
2. Χρήση CPU (Raspberry Pi):
   Η ανάλυση δείχνει ομαλή λειτουργία καθ' όλη τη δοκιμή και ότι η βαριά επεξεργασία εικόνας εκτελείται σωστά από το Coral, αποφορτίζοντας τον κεντρικό επεξεργαστή.
3. Χρήση μνήμης RAM:
   Η σταδιακή αύξηση είναι αναμενόμενη, καθώς τα δεδομένα κάθε frame αποθηκεύονται προσωρινά στη RAM μέχρι την λήξη της δοκιμής.
4. Θερμοκρασία συστήματος:
   Παρότι η θερμοκρασία είναι αυξημένη, παραμένει σταθερή, επιβεβαιώνοντας τη σταθερή λειτουργία του συστήματος υπό φορτίο.

Πίνακας 4.9: Ανάλυση χρόνων Frame



Ο σημαντικότερος πίνακας 4.9 αποτυπώνει τους χρόνους των κοστοβόρων λειτουργιών ανά frame και υποδεικνύει περιθώρια βελτιστοποίησης:

- Οι μέγιστες τιμές εμφανίζονται κυρίως κατά την εκκίνηση του Vision Pipeline thread.
- Η απεικόνιση της οθόνης και ο σχεδιασμός των γραφικών προσθέτουν περίπου ~19 ms καθυστέρηση ανά frame. Μόνο με την αφαίρεση/ελαχιστοποίηση αυτών, το σύστημα μπορεί να βελτιωθεί αισθητά σε ρυθμό καρέ και latency.
- Ο πίνακας αναδεικνύει επίσης τη σημασία του μηχανισμού υπολογισμού αποστάσεων, σε σχέση με το πόσο γρήγορα εκτελείται ολόκληρη η διαδικασία, ώστε να διατηρείται η πραγματικού χρόνου λειτουργία.

4.9.2 Ανάλυση Events

Η σημαντικότερη πτυχή της αξιολόγησης είναι η ανάλυση των events, γιατί επιτρέπει τη μελέτη της συμπεριφοράς του συστήματος κατά τη φάση αναγνώρισης και παρακολούθησης.

Κάθε event παράγει πέντε πίνακες:

- Detection Timeline: απεικονίζει τις αναγνωρίσεις στον χρόνο.
- Tracking Phases: δείχνει σε ποια φάση βρίσκονται οι κάμερες (tracking/fallback1/fallback2 κ.λπ.).
- Detection Confidence: παρουσιάζει την τιμή confidence για κάθε ανίχνευση.
- Detection Distance: εμφανίζει την εκτιμώμενη απόσταση ανά ανίχνευση.
- Class Distribution: αποτυπώνει την κατανομή των κλάσεων μέσα στο event.

Παρακάτω αναλύονται τρεις χαρακτηριστικές περιπτώσεις:

1. όταν το σύστημα λειτουργεί σωστά σε όλη τη διάρκεια του event,
2. όταν κατά τη διάρκεια του event μία από τις δύο κάμερες χάνει στιγμιαία το αντικείμενο
3. και ένα ακραίο σενάριο προσπάθειας αναγνώρισης πινακίδας που δεν υπάρχει στο dataset.

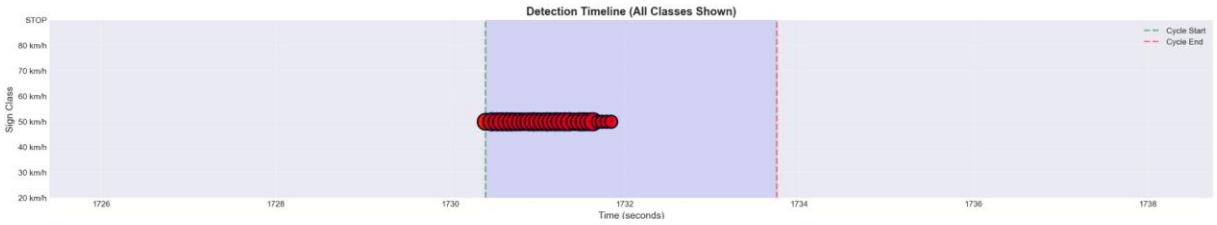
4.9.2.1 Σωστή λειτουργία

Κατά τη διάρκεια του συγκεκριμένου event δεν εμφανίστηκαν περιπτώσεις λανθασμένης ταξινόμησης ούτε ενεργοποιήθηκαν fallback φάσεις.

Εικόνα 4.17: Scenario 1

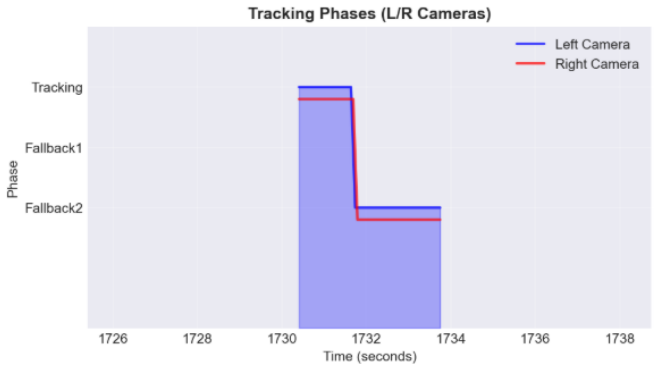
Στην Εικόνα 4.17 φαίνεται το μπλε PaddedBox που περιβάλει το σωστο Bbox .

Πίνακας 4.10: Detection Timeline Scenario 1



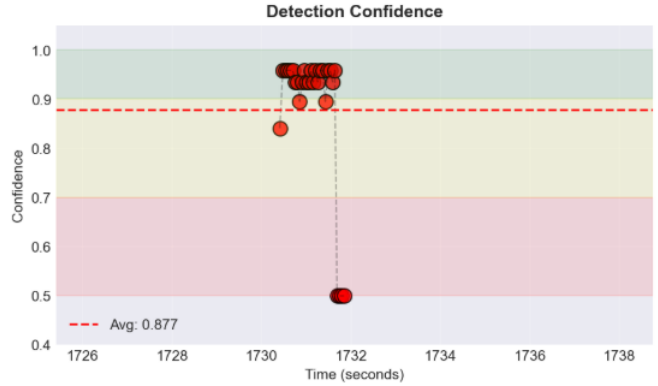
Παρατηρείται σταθερή ανίχνευση στα 50 km/h. Το κενό από την τελευταία αναγνώριση μέχρι το τέλος του event είναι αναμενόμενο, καθώς το όχημα έχει περάσει την πινακίδα και οι δύο κάμερες βρίσκονται σε fallback2.

Πίνακας 4.11: Tracking Phases Scenario 1



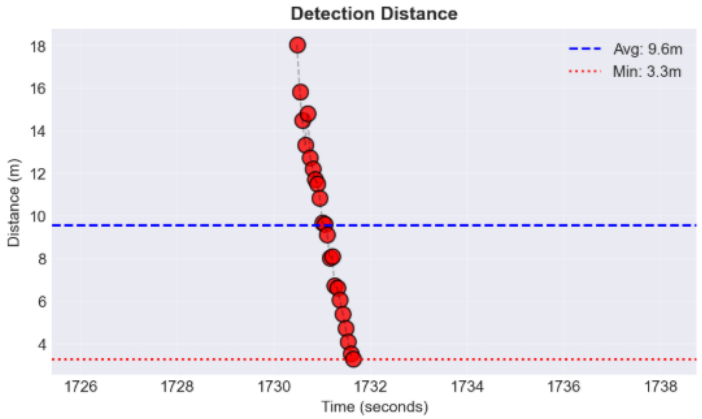
Οι κάμερες λειτουργούν παράλληλα και παραμένουν στην ίδια φάση καθ' όλη τη διάρκεια του event.

Πίνακας 4.12: Detection Confidence Scenario 1



Το confidence είναι υψηλό σε όλο το event, με σημαντική πτώση στο τέλος όταν η πινακίδα φτάνει στην άκρη του κάδρου και αρχίζει να χάνεται.

Πίνακας 4.13: Detection Distance Scenario 1



Φαίνεται ομαλή, φθίνουσα πορεία της εκτιμώμενης απόστασης της πινακίδας στον χρόνο (καθώς το όχημα πλησιάζει).

Πίνακας 4.14: Class Distribution Scenario 1

Τελος από τον πίνακα 4.14 βλέπουμε ότι όλες οι ανιχνεύσεις γυρίσαν την ιδιά και σωστή κλάση των 50Km/h.

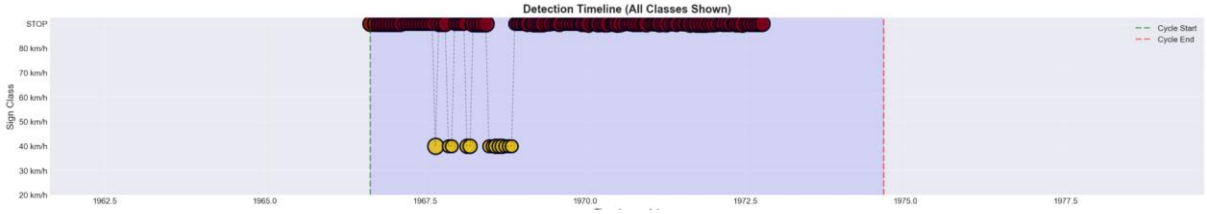
4.9.2.2 Σενάριο με FallBack

Το παρακάτω event δείχνει τη λειτουργία του συστήματος όταν ενεργοποιούνται τα FallBack. Η σωστή σήμανση της πινακίδας είναι STOP.

Εικόνα 4.18: Scenario 2

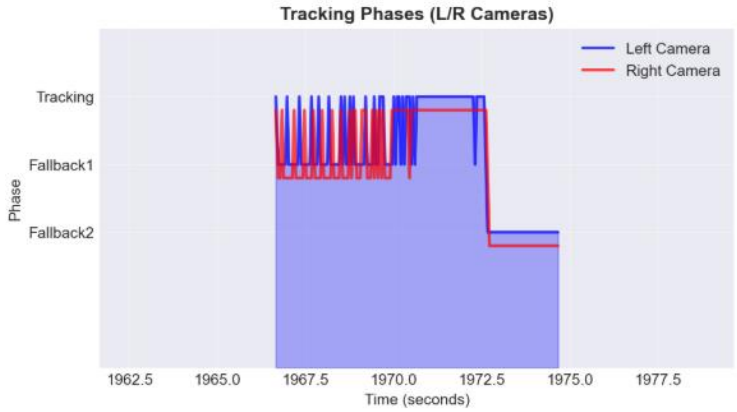
Στην Εικονα 4.18 φαίνεται μόνο το PaddedBox χωρίς να υπάρχει μέσα Bbox , ακόμα το δεξί PaddedBox φαινεται μεγαλύτερο από το αριστερό δείχνοντας με αυτόν τον τρόπο ότι η δεξιά κάμερα έχει αρχίσει τον μηχανισμό fallback νωρίτερα από την αριστερή.

Πίνακας 4.15: Detection Timeline Scenario 2



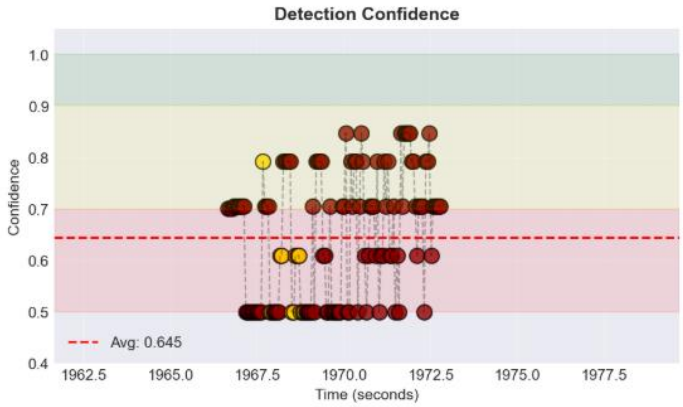
Από την αρχή παρατηρείται λάθος classification της πινακίδας.

Πίνακας 4.16: Tracking Phase Scenario 2



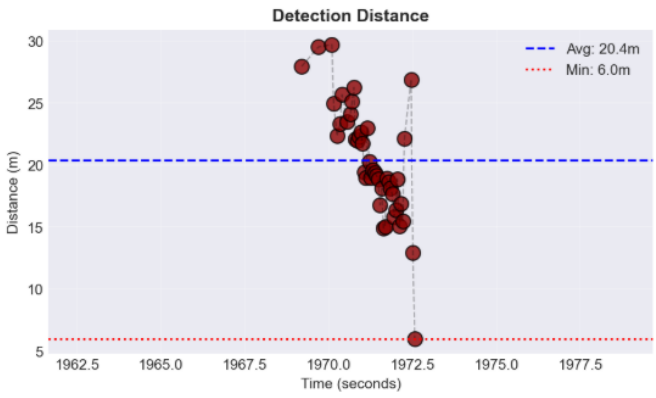
Οι δύο κάμερες εναλλάσσουν πολύ γρήγορα φάση στην αρχή του event, κάτι που υποδεικνύει χαμηλό confidence του συστήματος.

Πίνακας 4.17: Detection Confidence Scenario 2



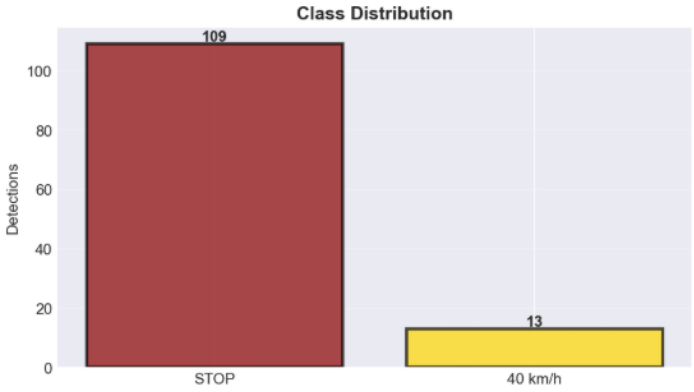
Ο πίνακας 4.17 επιβεβαιώνει τα παραπάνω: ο μέσος όρος confidence είναι 0.645, ενώ οι υψηλότερες τιμές εμφανίζονται όσο το όχημα πλησιάζει την πινακίδα.

Πίνακας 4.18: Detection Distance Scenario 2



Στις μετρήσεις απόστασης διακρίνονται μεγάλα κενά ανάμεσα σε διαδοχικούς υπολογισμούς, παρότι υπάρχει πυκνός αριθμός μετρήσεων μεταξύ 20–15 m (το όχημα ήταν σταματημένο). Η στιγμιαία εύρεση/απώλεια της πινακίδας από τις κάμερες οδηγεί και σε λανθασμένα αποτελέσματα.

Πίνακας 4.19: Class Distribution Scenario 2



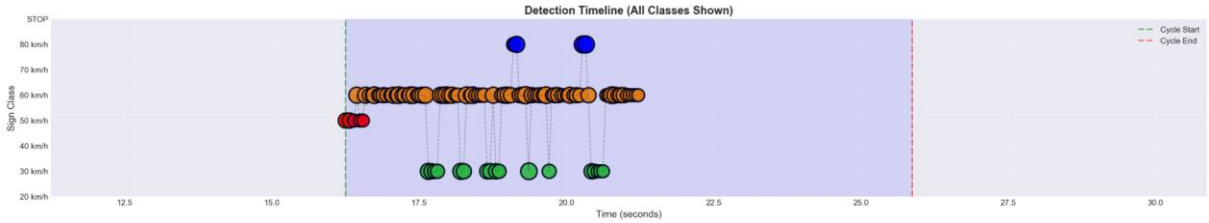
Παρά τα προβλήματα στον εντοπισμό, η κατανομή κλάσεων δείχνει ότι τα λάθη classification αποτελούν μόνο το 10,66% των ανιχνεύσεων.

4.9.2.3 Σενάριο Λανθασμένης Ανίχνευσης

Σε αυτό το event, το σύστημα επιχειρεί να αναγνωρίσει πινακίδα 90 km/h. Υπό κανονικές συνθήκες θα την προσπερνούσε, όμως λόγω μικρού μεγέθους και φωτισμού ενεργοποιούνται επαναλαμβανόμενες προσπάθειες αναγνώρισης.

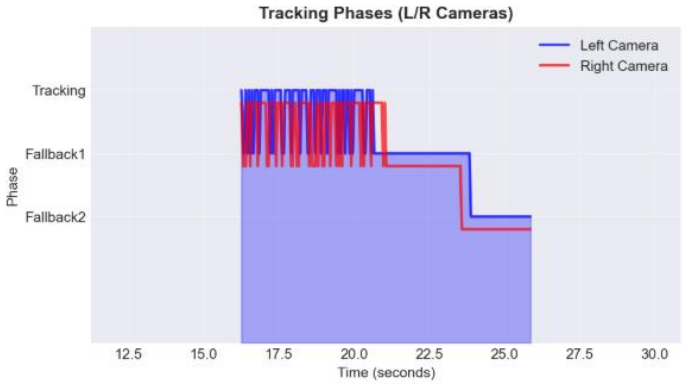
Εικόνα 4.19: Scenario 3

Πίνακας 4.20: Detection Timeline Scenario 3



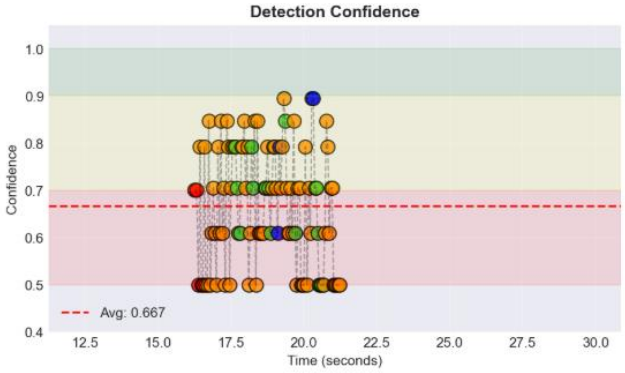
Παρατηρείται εναλλαγή κλάσεων· η σχετικά σταθερή αναγνώριση 60 km/h είναι λογική λόγω της ομοιότητας “6” με “9”.

Πίνακας 4.21: Tracking Phases Scenario 3



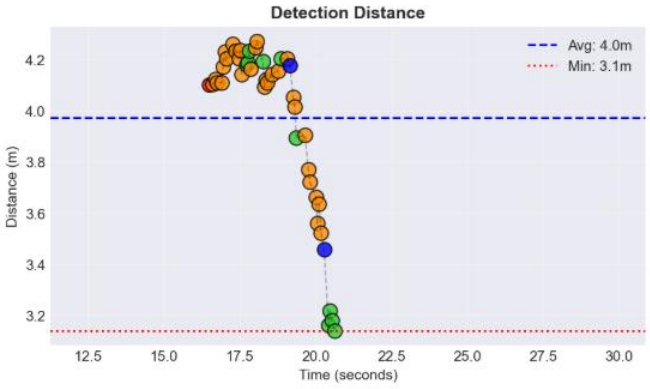
Το σύστημα δεν μπορεί να παραμείνει σε φάση Tracking και εναλλάσσεται συνεχώς μεταξύ Fallback1 και Tracking.

Πίνακας 4.22: Detection Confidence Scenario 3



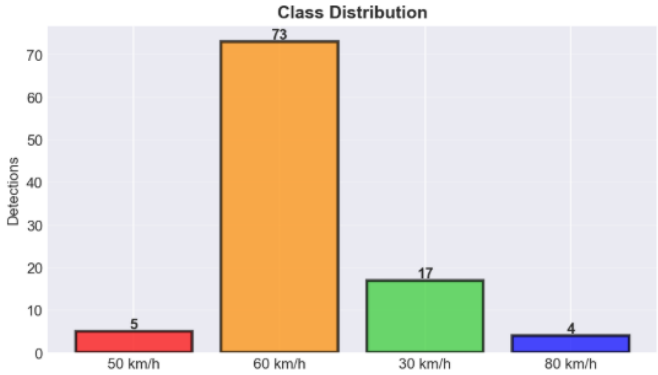
Το confidence είναι επίσης επηρεασμένο: ο μέσος όρος δεν ξεπερνά το 0.7, ενώ μερικές λανθασμένες ανιχνεύσεις φτάνουν πάνω από 0.8.

Πίνακας 4.23: Detection Distance Scenario 3



Ο υπολογισμός βάθους αρχικά δεν επηρεάζεται σημαντικά (το όχημα ήταν σταθμευμένο σε σταθερή απόσταση). Με την εκκίνηση του οχήματος εμφανίζεται το ίδιο πρόβλημα με μεγάλα κενά ανάμεσα σε διαδοχικούς υπολογισμούς.

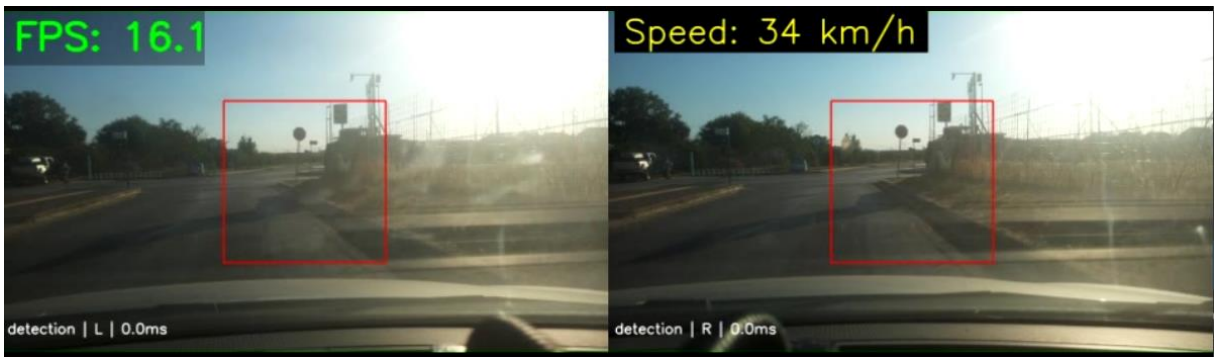
Πίνακας 4.24: Class Distribution Scenario 3



Τέλος, η κατανομή ανιχνεύσεων δείχνει κυρίαρχη την κλάση 60 km/h, ενώ λανθασμένα εμφανίζονται και 30, 50, 80 km/h.

4.9.2.4 Σενάρια Αστοχίας

Σε αυτά τα σενάρια το σύστημα δεν αναγνωρίζει καθόλου τις πινακίδες. Οι πιθανές αιτίες είναι: μη ιδανικός φωτισμός (έντονες σκιές/αντανακλάσεις), δυσμενής τοποθέτηση ή γωνία/προσανατολισμός της πινακίδας, και πολύ σύντομος χρόνος παραμονής της στο κάδρο (λίγα frames).



Εικόνα 4.20: Αστοχία 1

Στην εικονα 4.20 φαινεται ότι η έντονη οπίσθια φωταγώγηση από τον ήλιο προκαλεί υπερ-έκθεση στο πεδίο και υπο-έκθεση στην πινακίδα, η οποία εμφανίζεται ως σκιερή χωρίς υφές, οδηγώντας σε αποτυχία αναγνώρισης

Εικόνα 4.21: Αστοχία 2.α

Εικόνα 4.22 Αστοχία 2.β

Στην εικόνα 4.21 το STOP εμφανίζεται περιφερειακά και, με την αλλαγή κατεύθυνσης, αποκόπτεται από το οπτικό πεδίο των καμερών όπως φαίνεται στην εικόνα 4.22, εξηγώντας την αποτυχία συνεχούς αναγνώρισης

Εικόνα 4.23: Αστοχία 3

Στην εικόνα 4.23 αποτυπώνεται περίπτωση όπου, παρά τις ιδανικές συνθήκες φωτισμού, το σύστημα δεν προλαβαίνει να αναγνωρίσει την πινακίδα λόγω πολύ μικρού χρόνου παραμονής στο κάδρο.

4.9.3 Συμπεράσματα

Στη διαδρομή υπήρχαν 38 πινακίδες και παρήχθησαν 32 events. Το πρώτο event δεν αντιστοιχούσε σε πινακίδα της δοκιμής, άρα αξιολογήθηκαν 31 πραγματικά events. Το σύστημα αναγνώρισε 31 από τις 38 πινακίδες και αστόχησε σε 7. Οι αστοχίες προήλθαν από φωτισμό σε 5 περιπτώσεις, από δυσμενή θέση πινακίδας σε 1 και από πολύ μικρό χρόνο παραμονής στο κάδρο σε 1.

Από τα 31 events, τα 24 ανήκουν σε σενάρια σωστής λειτουργίας ή ήπιου fallback που δεν επηρεάζουν τη συμπεριφορά του συστήματος. Τα υπόλοιπα 7 είναι έντονου fallback όπου ο υπολογισμός απόστασης αποτυγχάνει. Παρά τα παραπάνω, το υποσύστημα υποβοήθησης οδηγού ενημέρωσε και ανανέωσε τα ισχύοντα όρια ταχύτητας με επιτυχία 100 τοις 100.

Η απόδοση σε πραγματικό χρόνο ήταν σταθερή. Στη φάση αναγνώρισης ο ρυθμός ήταν περίπου 15 fps και κατά τη διάρκεια πολλών events έφτασε έως 25 fps. Τα μεμονωμένα spikes καθυστέρησης δεν επηρέασαν τη ροή. Η χρήση CPU παρέμεινε ομαλή χάρη στην εκφόρτωση της βαριάς επεξεργασίας στο Coral. Η μνήμη αυξανόταν σταθερά λόγω προσωρινής αποθήκευσης πλαισίων μέχρι τη λήξη της δοκιμής. Οι θερμοκρασίες ήταν αυξημένες αλλά σταθερές χωρίς αστάθειες.

Κεφάλαιο 5ο: Συμπεράσματα και προτάσεις βελτίωσής

5.1 Συνοπτική Αποτίμηση και Μελλοντικές Κατευθύνσεις

Υλοποιήθηκε ένα ολοκληρωμένο σύστημα υποβοήθησης οδηγού σε πραγματικό χρόνο πάνω σε Raspberry Pi 5 (16 GB RAM) με Coral USB Accelerator. Η αρχιτεκτονική είναι αρθρωτή (ανίχνευση, παρακολούθηση, εκτίμηση απόστασης, διεπαφή οδηγού) και η κατασκευή φορητή και στιβαρή. Η τροφοδοσία σταθεροποιήθηκε με UPS, ενώ η μηχανική στήριξη διατηρεί αξιόπιστη ευθυγράμμιση καμερών. Μελλοντικά, προτεραιότητα έχει η ανθεκτικότητα σε δύσκολο φωτισμό, η ομαλότερη μετάβαση tracking/fallback και ο εμπλουτισμός δεδομένων.

5.2 Συμπεράσματα

Το Raspberry Pi 5 (16 GB) σε συνδυασμό με το Coral USB απέδειξε ότι μια χαμηλής ισχύος πλατφόρμα μπορεί να υποστηρίξει πρακτικές λειτουργίες ADAS με σταθερή συμπεριφορά σε πραγματικές συνθήκες. Η εκφόρτωση της βαριάς επεξεργασίας εικόνας στο Coral διατηρεί ομαλή χρήση CPU και ρυθμό σε πραγματικό χρόνο, ενώ η θερμική/μηχανική σταθερότητα της κατασκευής και η σταθερή παροχή ισχύος (UPS) αποτρέπουν αστάθειες. Η λογική ανίχνευσης–παρακολούθησης παρείχε συνεπή ενημέρωση οδηγού, με τους βασικούς περιορισμούς να σχετίζονται με σκληρές φωτιστικές συνθήκες, δυσμενή γεωμετρία τοποθέτησης σημάτων και πολύ μικρό χρόνο παραμονής στο κάδρο. Το σύστημα παραμένει επεκτάσιμο τόσο σε επίπεδο υλικού (αισθητήρες, ψύξη/θήκες, βελτιωμένη τροφοδοσία) όσο και λογισμικού (μοντέλα, φίλτρα χρονικής ομαλοποίησης).

5.3 Μελλοντικές επεκτάσεις

- Καλύτερη τροφοδοσία. Με σύνδεση στην μπαταρία του οχήματος
- Καλύτερη ενσωμάτωση στο όχημα. Εγκατάσταση στον καθρέφτη του οχήματος, σκίαστρα για τις κάμερες, τακτοποιημένη καλωδίωση και ασφαλή περάσματα. Σύντομο πρωτόκολλο γρήγορου ελέγχου ευθυγράμμισης πριν από κάθε δοκιμή.
- Σύνδεση με CAN bus. Ανάγνωση ταχύτητας, φλας, φρένου και κατάστασης φωτισμού για βελτιωμένη λογική ειδοποίησης και συγχρονισμό μετρήσεων. Καταγραφή CAN μαζί με τα βίντεο για πλήρη ιχνηλασιμότητα.
- Καλύτερο μοντέλο αναγνώρισης. Εκπαίδευση με πιο σκληρά σενάρια φωτισμού, μικρά μεγέθη και λοξές γωνίες. Ρύθμιση κατωφλίων, καλύτερο association στο tracking και ενίσχυση της χρονικής συνέχειας.
- Περισσότερες κλάσεις. Προσθήκη αναγνώρισης επιπλέον πινακίδων και αντικειμένων όπως οχήματα και πεζοί, με αντίστοιχη λογική προτεραιότητας και ασφαλούς ειδοποίησης.
- Καλύτερος κώδικας, πλήρως σε C++. Μεταφορά όλης της προ-επεξεργασίας, του pipeline και της απεικόνισης σε C++ με πολυνηματικότητα, zero-copy ροές και βελτιστοποιημένη διαχείριση μνήμης, ώστε να μειωθεί το latency και να αυξηθεί η σταθερότητα.
- Καλύτερο UI/HMI: πιο λιτά overlays, ιεράρχηση ειδοποιήσεων (INFO/WARN/ALERT), night mode/auto-dim, μικρή ένδειξη κατάστασης (GPS/IMU/κάμερες).
- Καλύτερο Hardware: μετάβαση σε ισχυρότερο SBC και automotive κάμερες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] ΕΛΣΤΑΤ, «Οδικά Τροχαία Ατυχήματα 2023 [Road Traffic Accidents 2023]», Αθήνα, Ελλάδα, 2025.

[2] European Commission, "Country Profile: Greece," Road Safety Observatory, Brussels, 2023.

[3] ACEA (European Automobile Manufacturers' Association), "Vehicles in Use—Europe," Brussels, 2023.

[4] Arm Ltd., "ADAS (Advanced Driver Assistance Systems) — Glossary." [Online]. Available: https://www.arm.com/glossary/adas

[5] European Commission, "Intelligent Speed Adaptation (ISA)," Road Safety Policy, Directorate-General for Mobility and Transport. [Online]. Available: https://road-safety.transport.ec.europa.eu/eu-road-safety-policy/priorities/safe-road-use/safe-speed/archive/speeding/new-technologies-new-opportunities/intelligent-speed-adaptation-isa_en

[6] European Transport Safety Council (ETSC), "Intelligent Speed Assistance (ISA)." [Online]. Available: https://etsc.eu/intelligent-speed-assistance-isa

[7] C. G. Kiran, L. V. Prabhu, A. R. V. and K. Rajeev, "Traffic Sign Detection and Pattern Recognition Using Support Vector Machine," *2009 Seventh International Conference on Advances in Pattern Recognition*, Kolkata, India, 2009, pp. 87-90, doi: 10.1109/ICAPR.2009.58.

[8] J. Stallkamp, M. Schlipsing, J. Salmen and C. Igel, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition," *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, USA, 2011, pp. 1453-1460, doi: 10.1109/IJCNN.2011.6033395.

[9] Y. Han and E. Oruklu, "Traffic sign recognition based on the NVIDIA Jetson TX1 embedded system using convolutional neural networks," *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, USA, 2017, pp. 184-187, doi: 10.1109/MWSCAS.2017.8052891.

[10] A. Womg, M. J. Shafiee, F. Li and B. Chwyl, "Tiny SSD: A Tiny Single-Shot Detection Deep Convolutional Neural Network for Real-Time Embedded Object Detection," *2018 15th Conference on Computer and Robot Vision (CRV)*, Toronto, ON, Canada, 2018, pp. 95-101, doi: 10.1109/CRV.2018.00023.

[11] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li and S. Hu, "Traffic-Sign Detection and Classification in the Wild," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 2110-2118, doi: 10.1109/CVPR.2016.232.

[12] M. Antonakakis *et al*., "Real-Time Object Detection using an Ultra-High-Resolution Camera on Embedded Systems," *2022 IEEE International Conference on Imaging Systems and Techniques (IST)*, Kaohsiung, Taiwan, 2022, pp. 1-6, doi: 10.1109/IST55454.2022.9827742.

[13] A. Mögelmose, M. M. Trivedi and T. B. Moeslund, "Traffic sign detection and analysis: Recent studies and emerging trends," *2012 15th International IEEE Conference on Intelligent Transportation Systems*, Anchorage, AK, USA, 2012, pp. 1310-1314, doi: 10.1109/ITSC.2012.6338900.

[14] Mobileye, "Mobileye launches the first camera-only Intelligent Speed Assist to meet new EU standards." [Online]. Available: https://www.mobileye.com/news/mobileye-launches-the-first-camera-only-intelligent-speed-assist-to-meet-new-eu-standards

[15] Mobileye, "EyeQ™ Chip." [Online]. Available: https://www.mobileye.com/technology/eyeq-chip

[16] Mobileye, "EyeQ™ Chip — Benchmark." [Online]. Available:

https://www.mobileye.com/technology/eyeq-chip/benchmark
[17] Mobileye, “SuperVision.” [Online]. Available: https://www.mobileye.com/solutions/super-vision
[18] Mobileye, “Mobileye at CES 2022: The (Not-So) Secret to Self-Driving? Data.” [Online]. Available: https://www.mobileye.com/blog/mobileye-ces-2022-self-driving-secret-data
[19] Tesla, “Transitioning to Tesla Vision.” [Online]. Available: https://www.tesla.com/support/transitioning-tesla-vision
[20] WikiChip, “Tesla FSD chip.” [Online]. Available: https://en.wikichip.org/wiki/tesla_(car_company)/fsd_chip
[21] Tesla, “Model S Owner’s Manual (Camera positions & systems).” [Online]. Available: https://www.tesla.com/ownersmanual/models/en_us/GUID-682FF4A7-D083-4C95-925A-5EE3752F4865.html
[22] Tesla, “AI.” [Online]. Available: https://www.tesla.com/AI
[23] Raspberry Pi Ltd., “Raspberry Pi 5.” [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5
[24] Coral, “USB Accelerator — Datasheet.” [Online]. Available: https://coral.ai/docs/accelerator/datasheet
[25] Ultralytics, “Coral Edge TPU on Raspberry Pi — Installation walkthrough.” [Online]. Available: https://docs.ultralytics.com/guides/coral-edge-tpu-on-raspberry-pi/#installation-walkthrough
[26] NETTOP.gr, “NVIDIA Jetson Orin Nano Super 8GB Developer Kit — Product page.” [Online]. Available: https://www.nettop.gr/index.php/en/boards/nvidia/nvidia-jetson-orin-nano-super-8gb-developer-kit.html
[27] NETTOP.gr, “Raspberry Pi 5 (16GB) — Product page.” [Online]. Available: https://www.nettop.gr/index.php/en/raspberry-pi-en/kits-and-boards/raspberry-pi-5-16gb.html
[28] NETTOP.gr, “Search results for ‘coral usb’.” [Online]. Available: https://www.nettop.gr/index.php/en/component/uemart_category_id=0/?keyword=coral+usb&limitstart=0&option=com_virtuemart&view=category&search=true&virtuemart_category_id=0
[29] Raspberry Pi Ltd., “Camera Module 3.” [Online]. Available: https://www.raspberrypi.com/products/camera-module-3
[30] Microchip Technology Inc., “ATmega328P Datasheet (Atmel-7810: Automotive Microcontrollers).” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
[31] DFRobot, “Product page (ID: 2142).” [Online]. Available: https://www.dfrobot.com/product-2142.html
[32] Bosch Sensortec, “BNO055 — Intelligent 9-axis absolute orientation sensor (Product flyer).” [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/product_flyer/bst-bno055-fl000.pdf
[33] u-blox, “NEO-6 GNSS Modules — Data Sheet (GPS.G6-HW-09005).” [Online]. Available: https://content.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf
[34] eElectronicParts, “XH-M403 Digital Voltage Regulator (XL4016 PWM Buck) — Product page.” [Online]. Available: https://www.eelectronicparts.com/products/xh-m403-digital-voltage-regulator-xl4016-pwm-buck-step-down-power-supply-board
[35] Waveshare, “UPS Module 3S — Wiki.” [Online]. Available: https://www.waveshare.com/wiki/UPS_Module_3S
[36] Bambu Lab, “ASA-CF Filament.” [Online]. Available: https://bambulab.com/en-eu/filament/asa-cf

[37] Python Software Foundation, “History and License.” [Online]. Available: https://docs.python.org/3/license.html
[38] Python Software Foundation, “The Python Tutorial.” [Online]. Available: https://docs.python.org/3.11/tutorial/index.html
[39] Python Software Foundation, “What’s New in Python 3.11.” [Online]. Available: https://docs.python.org/3.11/whatsnew/3.11.html
[40] Python Software Foundation, “Modules.” [Online]. Available: https://docs.python.org/3.11/tutorial/modules.html
[41] E. Smith, “PEP 557 – Data Classes.” [Online]. Available: https://peps.python.org/pep-0557
[42] Python Software Foundation, “threading — Thread-based parallelism.” [Online]. Available: https://docs.python.org/3.11/library/threading.html
[43] Python Software Foundation, “Thread Objects (RLock).” [Online]. Available: https://docs.python.org/3.11/library/threading.html#rlock-objects
[44] Python Software Foundation, “json — JSON encoder and decoder.” [Online]. Available: https://docs.python.org/3.11/library/json.html
[45] Arduino, “Getting Started with Arduino.” [Online]. Available: https://docs.arduino.cc/learn/starting-guide/getting-started-arduino
[46] Arduino, “Arduino Uno Pin Reference.” [Online]. Available: https://docs.arduino.cc/hardware/uno
[47] Arduino, “Serial Communication.” [Online]. Available: https://docs.arduino.cc/learn/communication/serial
[48] Arduino, “PWM Tutorial.” [Online]. Available: https://docs.arduino.cc/learn/microcontrollers/analog-output
[49] OpenCV Team, “About OpenCV.” [Online]. Available: https://opencv.org/about
[50] OpenCV Team, “OpenCV Documentation 4.8.0.” [Online]. Available: https://docs.opencv.org/4.8.0/index.html
[51] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, 2000.
[52] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge: Cambridge Univ. Press, 2003, ISBN: 978-0521540513.
[53] OpenCV Team, “About OpenCV.” [Online]. Available: https://opencv.org/about
[54] OpenCV Team, “OpenCV Documentation 4.8.0.” [Online]. Available: https://docs.opencv.org/4.8.0
[55] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. CVPR*, 2016, pp. 779–788.
[56] Ultralytics, “YOLOv8 Documentation.” [Online]. Available: https://docs.ultralytics.com
[57] NumPy Developers, “NumPy Documentation.” [Online]. Available: https://numpy.org/doc/stable
[58] Raspberry Pi Foundation, “Picamera2 Documentation (Manual).” [Online]. Available: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf
[59] Python Software Foundation, “threading — Thread-based parallelism.” [Online]. Available: https://docs.python.org/3.11/library/threading.html
[60] PySerial, “PySerial Documentation.” [Online]. Available: https://pyserial.readthedocs.io
[61] Adafruit, “Adafruit BNO055 Documentation.” [Online]. Available: https://docs.circuitpython.org/projects/bno055
[62] M. H. Jones, “Calibration Checkerboard Collection.” [Online]. Available: https://markhedleyjones.com/projects/calibration-checkerboard-collection
[63] NVIDIA, “CUDA Toolkit Documentation.” [Online]. Available: https://docs.nvidia.com/cuda

[64] Ultralytics, “YOLOv8 Documentation.” [Online]. Available: https://docs.ultralytics.com
[65] A. B. Jung, “imgaug Documentation.” [Online]. Available: https://imgaug.readthedocs.io
[66] CVAT.ai, “Computer Vision Annotation Tool.” [Online]. Available: https://www.cvat.ai
[67] X. Liu, Q. Zhou, X. Chen, L. Fan and C.-T. Cheng, “Bias-Error Accumulation Analysis for Inertial Navigation Methods,” *IEEE Signal Processing Letters*, vol. 29, pp. 299–303, 2022, doi: 10.1109/LSP.2021.3129151.

Σε ορισμένα σημεία αξιοποιήθηκε το ChatGPT (OpenAI) για προτάσεις αναδιατύπωσης και γλωσσική/τυπογραφική επιμέλεια. Δεν χρησιμοποιήθηκε για παραγωγή ή επιλογή βιβλιογραφίας ούτε για ανάλυση δεδομένων. Η ευθύνη για το περιεχόμενο και την ακρίβεια ανήκει αποκλειστικά στον συγγραφέα