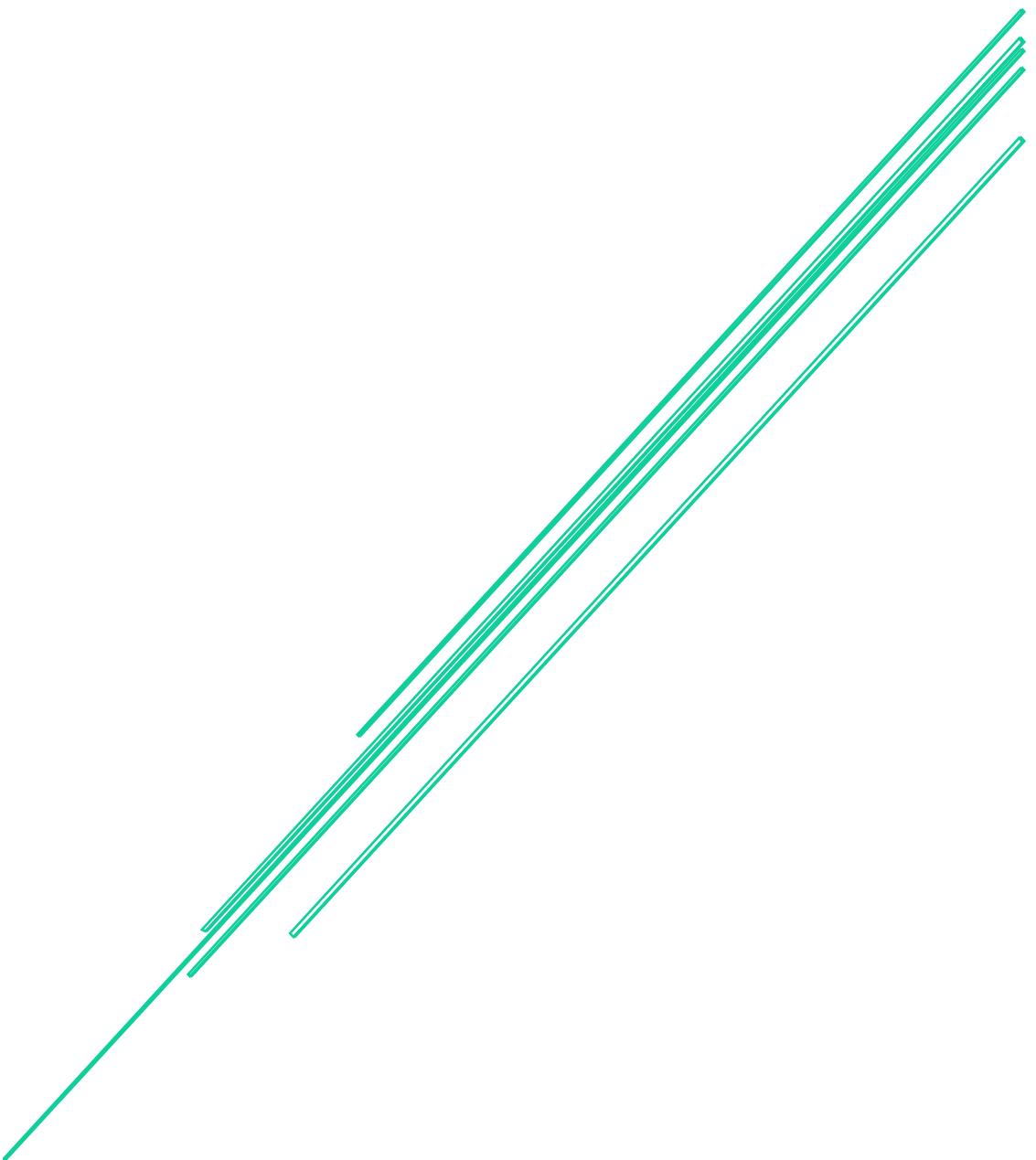


CAMERA VS LIDAR

ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ & Εφαρμογές 3Δ
Μοντελοποίησης – Εργασία 4



Ζωγράφου Μαρία Νίκη
AM: 1096060

ΠΕΡΙΕΧΟΜΕΝΑ

ΜΕΡΟΣ Α.....	2
A1 Έκδοση 1: Μία κάμερα.....	2
Ενδεικτικά επιτυχημένα αποτελέσματα:.....	4
Ανίχνευση Λωρίδων	6
A1 Έκδοση 2: Stereo Vision	6
Ενδεικτικά Αποτελέσματα:.....	8
A2: Computer Vision Approach	9
Ενδεικτικά Αποτελέσματα.....	9
A2: YOLO net	11
A3: Διάνυσμα Κίνησης.....	11
A4: Ανίχνευση Τοίχου.....	13
Σύντομες Οδηγίες χρήσης	14
ΜΕΡΟΣ Β.....	15
B1 – Ανίχνευση δρόμου	15
Ενδεικτικά αποτελέσματα:.....	15
B2 – Ανίχνευση εμποδίων	19
B3 – Διάνυσμα Κίνησης.....	21
Ενδεικτικά αποτελέσματα.....	22
B4 – Ανίχνευση τοίχου	23
Βιβλιογραφία	25

Github Repository: https://github.com/ManyaZ1/Camera_vs_LIDAR

Part_A1-2-3 video (stereo version): <https://www.youtube.com/watch?v=5cIPDILWHS>

Part_B1-2-3 video: <https://youtu.be/IFZvKo8ArFE>

ΜΕΡΟΣ Α

Στόχος του Μέρους Α είναι η εφαρμογή τεχνικών υπολογιστικής όρασης σε εικόνες του δρόμου, με σκοπό την ανίχνευση των ορίων του δρόμου, τον διαχωρισμό των λωρίδων κυκλοφορίας, την αναγνώριση εμποδίων και τον σχεδιασμό του διανύσματος κίνησης του οχήματος — λαμβάνοντας υπόψη την απόσταση από το πλησιέστερο εμπόδιο.

Για την ανίχνευση των ορίων του δρόμου (ερώτημα A1), υλοποιήθηκαν δύο διαφορετικές προσεγγίσεις: η πρώτη βασίζεται σε μία μόνο κάμερα, ενώ η δεύτερη αξιοποιεί στερεοσκοπική όραση (ζεύγος καμερών). Αντίστοιχα, για την ανίχνευση εμποδίων (ερώτημα A2), αναπτύχθηκαν δύο μέθοδοι: μία βασισμένη αποκλειστικά σε γεωμετρική επεξεργασία εικόνας και μία μέσω του νευρωνικού δικτύου YOLO.

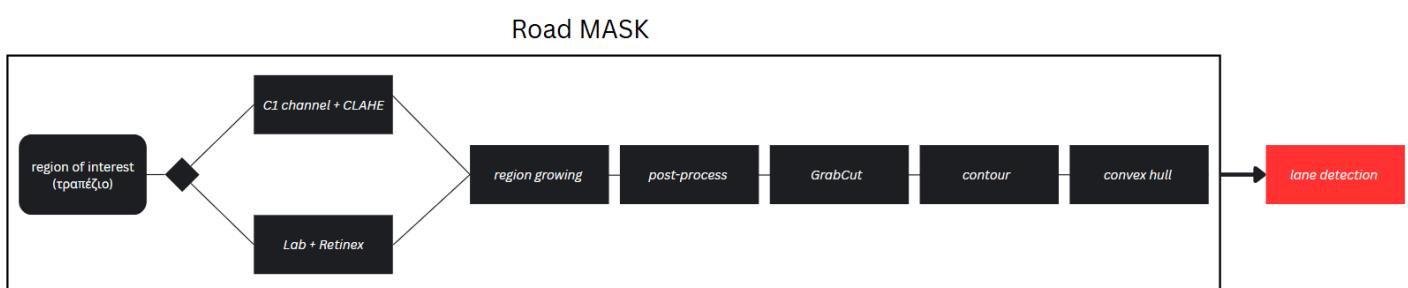
Λόγω της αυξημένης ταχύτητας της στερεοσκοπικής μεθόδου για την A1 και της μεγαλύτερης ακρίβειας του YOLO για την A2, οι επόμενες ενότητες (A3 και A4) βασίζονται στη συνδυαστική εκδοχή των δύο αυτών τεχνικών. Συνεπώς, στα ερωτήματα A1–A2 παρουσιάζονται και συγκρίνονται δύο εκδοχές: (α) η βασική υλοποίηση με μία μόνο κάμερα και χωρίς νευρωνικά δίκτυα και (β) η τελική, βελτιστοποιημένη εκδοχή με χρήση δύο καμερών και YOLO, η οποία επιλέχθηκε ως καταλληλότερη για τα επόμενα στάδια.

A1 Έκδοση 1: Μία κάμερα

Αρχείο κώδικα: [road_detector_A1.py](#)

Η αυτόματη ανίχνευση της οδικής επιφάνειας και ο διαχωρισμός των λωρίδων κυκλοφορίας από μεμονωμένες RGB εικόνες, εμφάνισε ιδιαίτερες δυσκολίες, ειδικά σε εικόνες με σκιές ή μη σταθερή φωτεινότητα και μη ξεκάθαρα πεζοδρόμια.

Για να αντιμετωπιστεί αυτό το πρόβλημα χρησιμοποιήθηκαν τεχνικές υπολογιστικής όρασης και επεξεργασίας χρωματικών καναλιών. Η ανίχνευση βασίζεται σε ένα pipeline που συνδυάζει ανάλυση χρωματικής πληροφορίας, region growing, GrabCut και μορφολογική επεξεργασία, ενώ η εξαγωγή των λωρίδων γίνεται μέσω Canny edge detection και Hough Transform.



```
#           __ C1 channel + CLAHE  \
#           /                               \
# ROI ---                                     -- region growing → post-process →
GrabCut → contour → convex hull
#           \__ Lab + Retinex ____/
```

Η περιπλοκότητα του pipeline καθιστά αυτή τη μέθοδο πιο αργή, δίνει όμως καλή ακρίβεια σε παράξενες εικόνες με σκιάσεις ή υπερβολική φωτεινότητα και σε εικόνες με έντονη βλάστηση. Παρουσιάζει επίσης επιτυχία σε ανισοεπίπεδους δρόμους (στους οποίους η 2^η μέθοδο με ransac δυσκολεύεται).

Βήματα επεξεργασίας εικόνας για εύρεση μάσκας δρόμου:

1. Κατασκευή Περιοχής Ενδιαφέροντος (ROI):

Ορίζεται μια τραπεζοειδής περιοχή ενδιαφέροντος στην εικόνα, με στόχο την αποκοπή του ουρανού και των άχρηστων περιοχών.

2. Προεπεξεργασία Καναλιού C1:

Υπολογίζεται το κανάλι C1 ως: $\text{arctangent}(R / \max(G, B))$.

Εφαρμόζεται CLAHE (Contrast Limited Adaptive Histogram Equalization) για ενίσχυση της αντίθεσης. Το **CLAHE** ενισχύει την αντίθεση (contrast) σε εικόνες, κάνοντας Histogram Equalization. Δηλαδή ανακατανέμει τις φωτεινότητες μιας εικόνας ώστε να καλύπτουν ομοιόμορφα το δυναμικό εύρος (0–255), ενισχύοντας την αντίθεση μεταξύ περιοχών διαφορετικής φωτεινότητας. Έτσι μπορεί να βελτιώσει τοπικά την ευκρίνεια και τις λεπτομέρειες, ειδικά σε εικόνες με χαμηλό φωτισμό ή χαμηλή αντίθεση.

Το κανάλι C1 είναι ανθεκτικό σε βλάστηση και γαλάζιο ουρανό.

3. Region Growing στο C1:

Ξεκινά από 3 σημεία εκκίνησης (seed points) κοντά στη βάση της εικόνας.

Region Growing (RG): Μέθοδος τμηματοποίησης εικόνας που ξεκινά από επιλεγμένα σημεία (seed points) και "μεγαλώνει" την περιοχή, προσθέτοντας γειτονικά pixel που έχουν παρόμοια τιμή φωτεινότητας ή χρώματος.

4. Προεπεξεργασία στο Χρωματικό Χώρο Lab:

Αρχικά εφαρμόζεται η βελτίωση Retinex για εξομάλυνση του φωτισμού. Η Retinex είναι αλγόριθμος εξομάλυνσης του φωτισμού — διορθώνει περιπτώσεις όπου η εικόνα έχει σκιές ή ανομοιομορφίες φωτεινότητας. Δίνει μια «πιο ουδέτερη» εικόνα

Η εικόνα μετατρέπεται σε Lab: $L^* = \text{Lightness}$ (φωτεινότητα), $a^* = \text{Πράσινο - Κόκκινο}$, $b^* = \text{Μπλε - Κίτρινο}$ και εξάγεται το κανάλι b^* . Ακολουθεί histogram equalization σε global επίπεδο (όχι τοπικά όπως το CLAHE). Αυτό χρειάζεται καθώς το Retinex εξομαλύνει τον φωτισμό, αλλά δεν ενισχύει απαραίτητα τη χρωματική αντίθεση στο κανάλι b^* . Η αντίθεση αυτή (μεταξύ μπλε και κίτρινου) κάνει την περιοχή του δρόμου να ξεχωρίζει πιο καθαρά.

5. Ανάπτυξη Περιοχής στο Lab b^* :

Χρησιμοποιούνται τα ίδια σημεία εκκίνησης για το region growing όπως και προηγουμένως.

Γιατί Retinex για Lab και CLAHE για C1:

Το Lab μοντέλο προορίζεται για αντίληψη χρώματος υπό φυσικό φωτισμό → άρα θέλει σταθεροποιημένο φωτισμό → προ επεξεργασία με Retinex. Ενώ το C1 βασίζεται σε σχετικές τιμές καναλιών RGB → άρα χρειάζεται τοπική ενίσχυση → προ επεξεργασία με CLAHE.

6. Επιλογή Καναλιού:

Μετά από πειράματα δημιούργησα έναν εμπειρικό κανόνα στον οποίο η καλύτερη μάσκα επιλέγεται βάσει στατιστικών του μέσου χρώματος:

Προτιμάται το Lab από προεπιλογή.

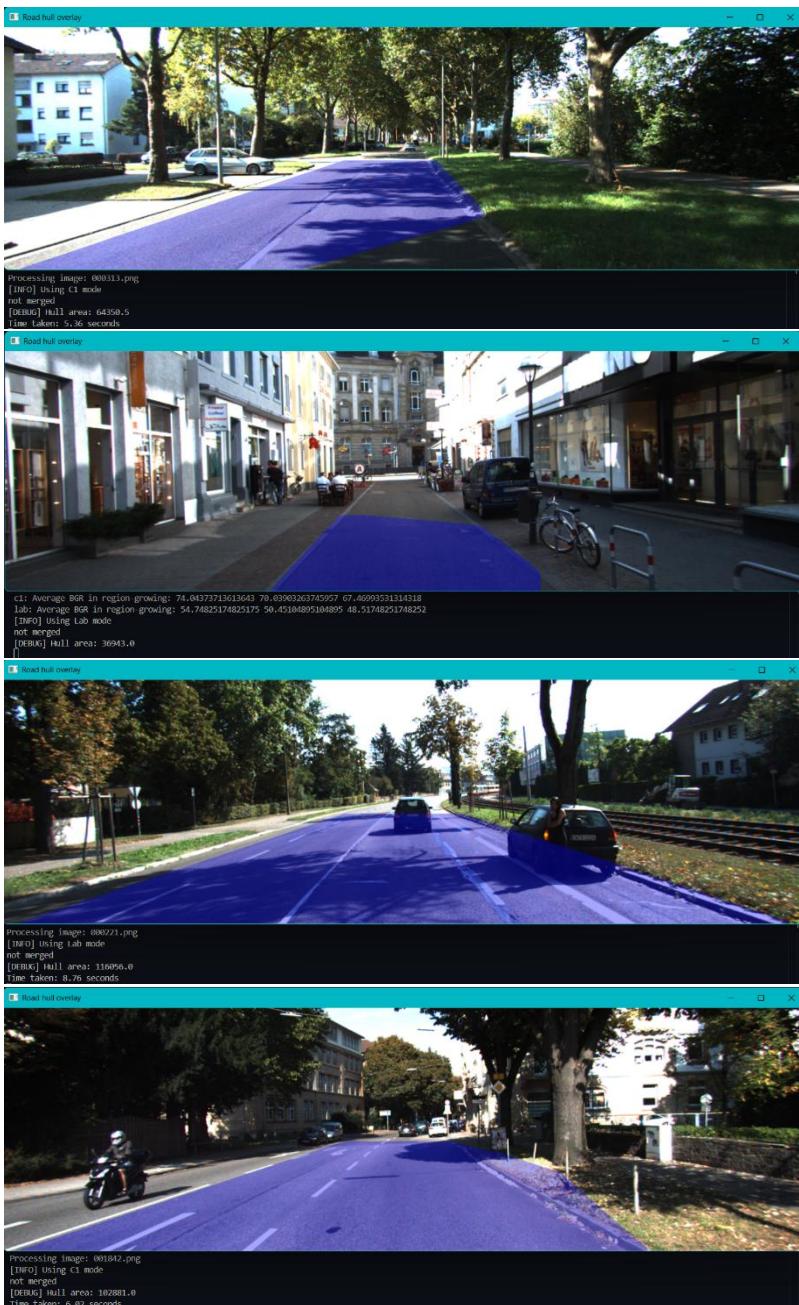
Το πρόγραμμα βασίζεται στο χρωματικό περιεχόμενο των δύο υποψήφιων μασκών (c1_rg, lab_rg), μετρώντας τον μέσο όρο των τιμών BGR της εικόνας μέσα στη μάσκα που εξήχθη από το region growing.

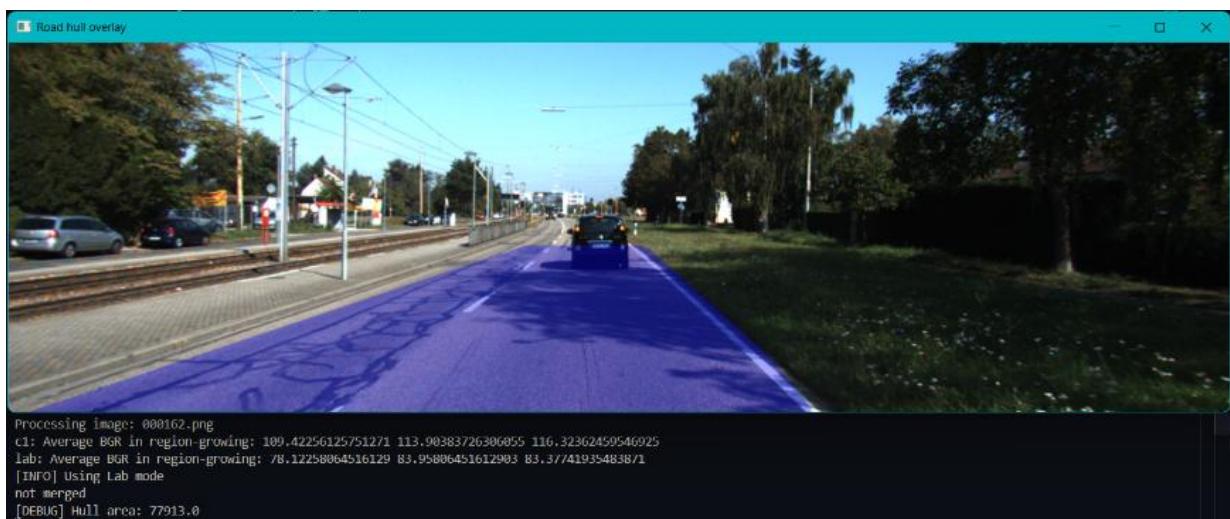
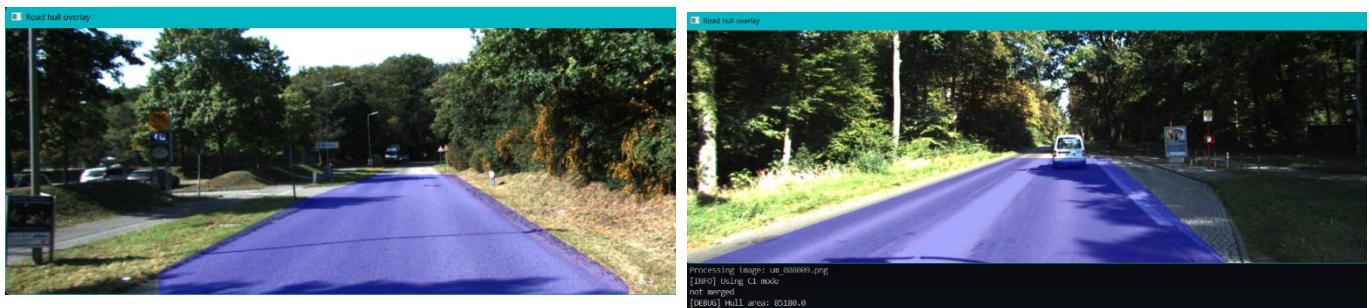
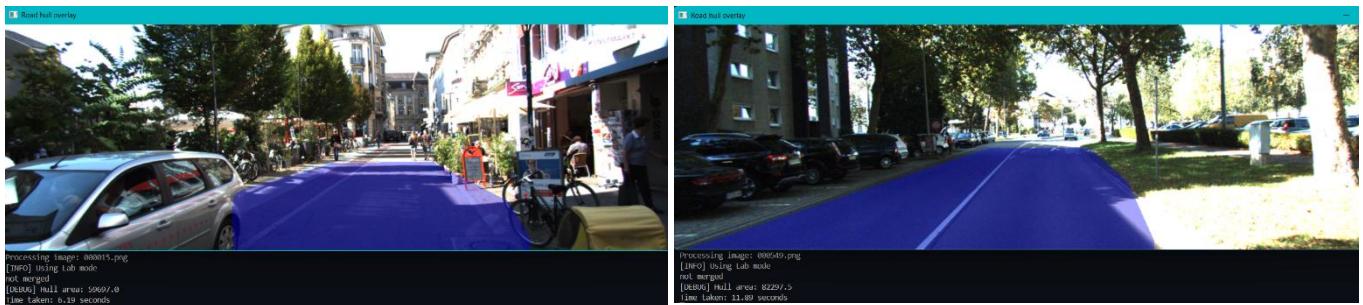
Αν η μάσκα του Lab περιέχει πράσινες ή κοκκινωπές περιοχές (ένδειξη βλάστησης ή πλακόστρωτου), γίνεται εναλλαγή σε C1. Η επιλογή του "καλύτερου" καναλιού βασίζεται στο τι περιέχει η μάσκα που προκύπτει από το region growing — όχι απλώς στο ίδιο το κανάλι.

Πιο συγκεκριμένα η επιλογή του C1 γίνεται με τα εξής κριτήρια:

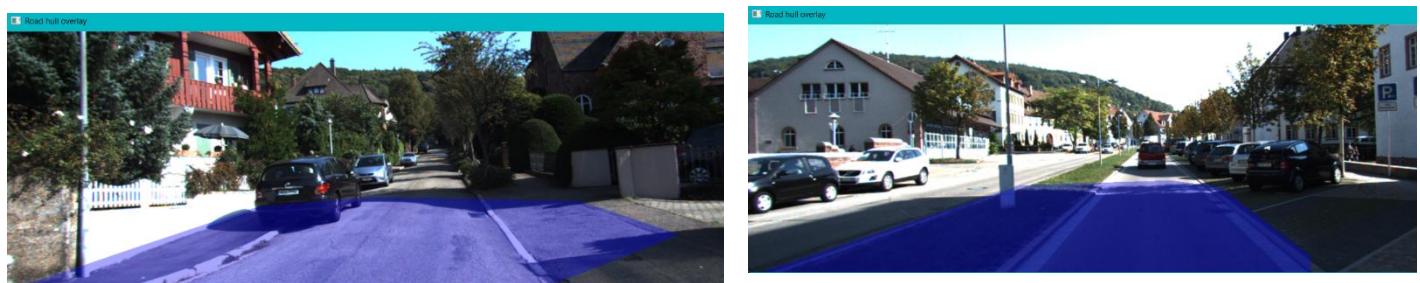
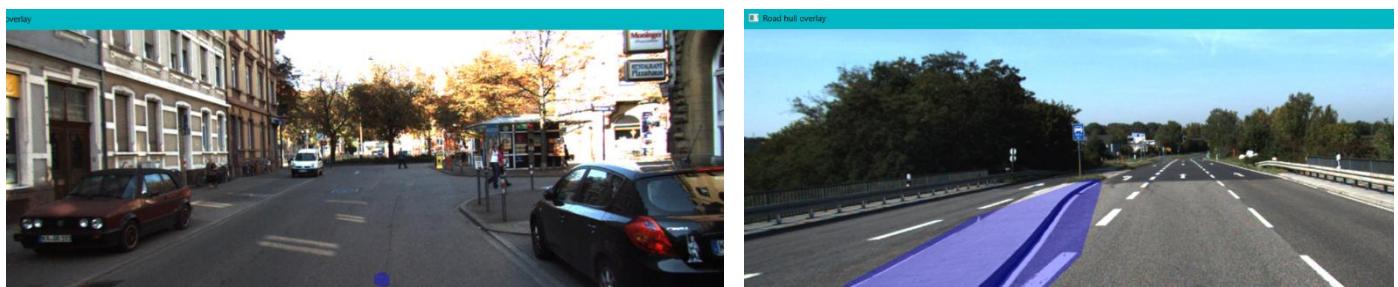
Αν η μέση πράσινη τιμή (green) της Lab μάσκας είναι τουλάχιστον 6 μονάδες μεγαλύτερη από το μπλε υποδεικνύει πιθανή βλάστηση οπότε επιλέγεται το C1. Αν η κόκκινη τιμή (red) της Lab είναι εμφανώς μεγαλύτερη από το μπλε τότε υπάρχουν πιθανές κόκκινες περιοχές (πεζοδρόμια, εμπόδια) στη μάσκα άρα επιλέγεται το C1. Τέλος αν το πράσινο είναι σαφώς μεγαλύτερο από το κόκκινο ίσως έχουμε φυτά ή θάμνους στη μάσκα άρα γυρνάμε σε C1.

Ενδεικτικά επιτυχημένα αποτελέσματα:





Η μέθοδος **αποτυγχάνει** όμως σε άλλες περιπτώσεις (συμπερίληψη πεζοδρομίων/γρασιδιού ή μη εύρεση μάσκας δρόμου καθόλου!):



Ανίχνευση Λωρίδων: Η ανίχνευση λωρίδων γίνεται και στις δύο εκδόσεις με τον ίδιο τρόπο οπότε η λογική της θα αναλυθεί παρακάτω. Ακολουθούν ενδεικτικά παραδειγματα Lane detection αυτής της μεθόδου:



Λόγος απόρριψης: Οι μεγάλοι χρόνοι υπολογισμού που κυμαίνονται μεταξύ των 5 και των 7 δευτερολέπτων, καθώς και η περιορισμένη αξιοπιστία αυτής της μεθόδου με οδήγησε να την απορρίψω και να χρησιμοποιήσω την μέθοδο που βασίζεται στις δύο κάμερες.

A1' Έκδοση 2: Stereo Vision

Αρχείο κώδικα: [camera_complete.py](#) → Υλοποιεί A1-A2-A3-A4.

Βήματα επεξεργασίας εικόνας για εύρεση μάσκας δρόμου:

1. Ανίχνευση εμποδίων με YOLO (σε όλη την αριστερή εικόνα): Το YOLO εφαρμόζεται για την εντοπισμό αντικειμένων. Η έξοδος είναι ένα δυαδικό εμπόδιο–mask που αποκόπτει τα εμπόδια από την επεξεργασία βάθους.
2. Εξαγωγή περιοχής ενδιαφέροντος (κάτω μισό εικόνας): Κόβεται το κάτω μισό της εικόνας για υπολογισμό βάθους και εντοπισμό του δρόμου.
3. Υπολογισμός χάρτη διαφοράς (disparity map): Γίνεται με το StereoSGBM (Semi-Global Block Matching) μεταξύ αριστερής και δεξιάς εικόνας.
4. Μετατροπή disparity σε 3D point cloud (εξαιρώντας εμπόδια): Χρησιμοποιώντας την μήτρα Q (από το calibration), παράγεται το νέφος σημείων του οδοστρώματος.
5. Εύρεση επιπέδου δρόμου με RANSAC: Εφαρμόζεται τμηματοποίηση επιπέδου (plane segmentation) στο point cloud για να εντοπιστούν inliers που ανήκουν στην επιφάνεια του δρόμου. Σημείωση ο ransac σε αυτή την υλοποίηση ψάχνει για ένα και μόνο επίπεδο (plane) μέσα στο point cloud.
6. Χαρτογράφηση των inliers σε μάσκα εικόνας: Οι 3D inlier πόντοι επαναπροβάλλονται στην εικόνα, σχηματίζοντας τη μάσκα δρόμου.
7. Φίλτραρισμα με βάση το κέντρο και καθαρισμός: Κρατιούνται μόνο οι περιοχές κοντά στο οπτικό κέντρο της εικόνας και εφαρμόζονται μορφολογικά φίλτρα.

8. Εύρεση κυρτών περιγραμμάτων (convex hulls): Οι μεγαλύτερες συνδεδεμένες περιοχές ενώνονται ή διατηρούνται, σχηματίζοντας το τελικό σχήμα του δρόμου.



Εικόνα από αρχικές εκδόσεις (χωρίς lanes):



Παρκάτω μετά την εξήγηση της ανίχνευσης λωριδων θα υπάρξουν περισσότερα ενδεικτικά αποτελέμσατα.

Βήματα ανίχνευσης λωρίδων κυκλοφορίας:

1. Ανίχνευση ακμών (Canny): Η αρχική έγχρωμη εικόνα μετατρέπεται σε grayscale, εφαρμόζεται Gaussian blur και στη συνέχεια Canny edge detector, για να βρεθούν οι έντονες ακμές – πιθανές γραμμές λωρίδων.
2. Μάσκα Περιοχής Ενδιαφέροντος (ROI): Οι ακμές περιορίζονται μόνο εντός της μάσκας δρόμου (road_hull_mask) — δηλαδή κρατάμε μόνο ότι βρίσκεται πάνω στον δρόμο.
3. Ανίχνευση ευθειών με Hough Transform: Χρησιμοποιείται η μέθοδος Hough για να εντοπιστούν γραμμές εντός των ακμών, οι οποίες πιθανόν αντιστοιχούν σε λωρίδες.
4. Επιλογή καλύτερης κεντρικής γραμμής (center line): Από τις ανιχνευμένες γραμμές, επιλέγεται εκείνη που:
 - a. Είναι σχεδόν κάθετη (αρκετά μεγάλη κλίση)
 - b. Βρίσκεται πιο κοντά στο κέντρο της μάσκας δρόμου

Αυτή θεωρείται η νοητή διαχωριστική γραμμή ανάμεσα στις δύο λωρίδες.

5. Διαχωρισμός σε αριστερή/δεξιά λωρίδα: Για κάθε pixel μέσα στη μάσκα του δρόμου, υπολογίζεται αν βρίσκεται αριστερά ή δεξιά της γραμμής (μέσω γινομένου διανυσμάτων) και δημιουργούνται δύο μάσκες: left_mask και right_mask
6. Οπτική αποτύπωση (overlay): Η κάθε λωρίδα χρωματίζεται:
 - a. Μπλε για δεξιά λωρίδα
 - b. Μωβ για αριστερή

Και προβάλλεται πάνω στην αρχική εικόνα.

Αν δεν βρεθεί valid κεντρική γραμμή, επιστρέφεται ολόκληρη η μάσκα του δρόμου χωρίς λωρίδες.

Ενδεικτικά Αποτελέσματα:



Συμπέρασμα: Παρατηρούμε γενικώς πιο αξιόπιστη ανίχνευση δρόμου πολύ γρηγορότερα, καθώς οι χρόνοι κυμαίνονται γύρω στο 1 δευτερόλεπτο. Αυτή η μέθοδος επιλέχθηκε τελικά.

A2: Computer Vision Approach

Η συγκεκριμένη έκδοση αποτελεί συνέχεια της έκδοσης του A1 με όραση από μία μόνο κάμερα και εγκαταλείφθηκε μετά την χρήση του YOLO. Εδώ είναι μια σύντομη περιγραφή της:

Για τον εντοπισμό εμποδίων, δεν γινόταν να χρησιμοποιηθεί μία μόνο εικόνα. Η μάσκα του δρόμου προκύπτει από την μία μόνο εικόνα, αλλά η ανίχνευση εμποδίων βασίζεται σε RANSAC και stereo vision.

Οπότε ακολουθήθηκαν τα παραάτω βήματα όπως και προηγουμένως, αλλά με κάποιες προσθήκες:

1. Υπολογισμός χάρτη διαφοράς (disparity map)
2. Δημιουργία 3D point cloud
3. RANSAC για ανίχνευση επιπέδου εδάφους
4. Αφαίρεση του εδάφους
5. Αποκοπή πολύ μακρινών σημείων
6. Εντοπισμός περιοχής δρόμου(με χρήση του A1 με colorspaces)
7. Ομαδοποίηση σημείων με DBSCAN
8. Φίλτραρισμα ψευδών clusters

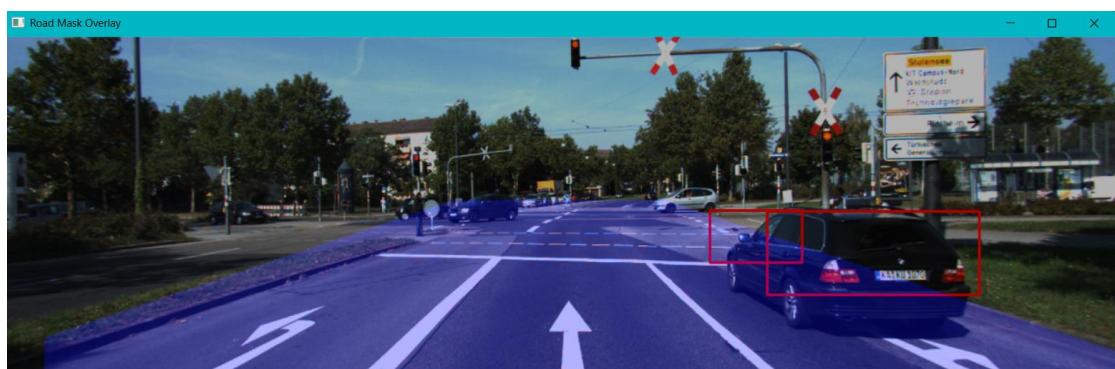
Κρατούνται μόνο τα clusters που: έχουν επαρκές μέγεθος ΚΑΙ επικαλύπτουν τη μάσκα δρόμου (π.χ. βρίσκονται “πάνω στον δρόμο”)

9. Προβολή Bounding Boxes: Υπολογίζονται τα 2D bounding boxes για κάθε cluster. Τα επικαλυπτόμενα boxes συγχωνεύονται (non-maximum suppression). Και εμφανίζονται επάνω στην εικόνα.

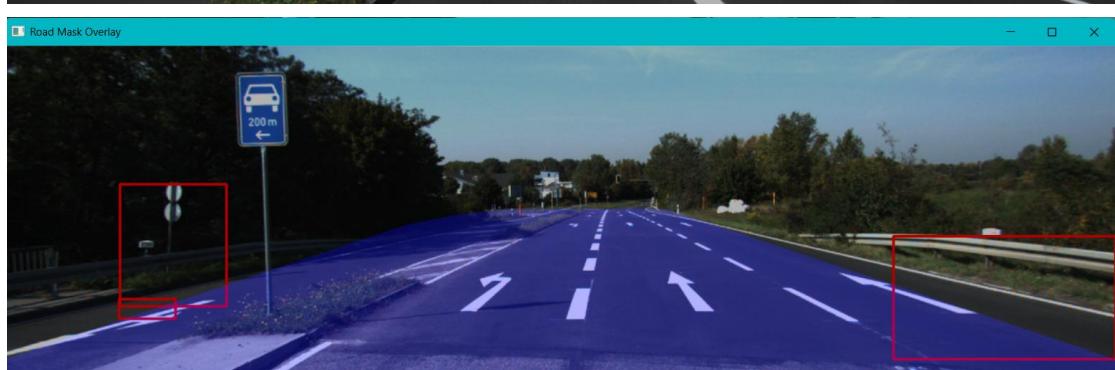
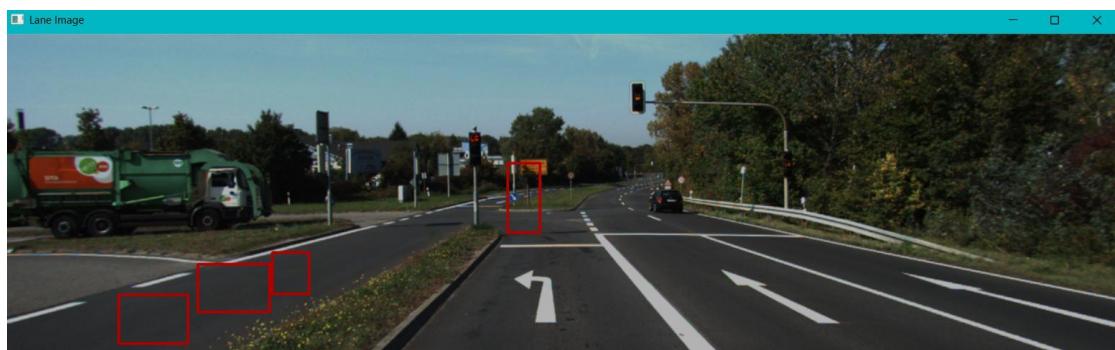
Ενδεικτικά Αποτελέσματα

Παρατηρήθηκαν επιτυχημένες ανιχνεύσεις:



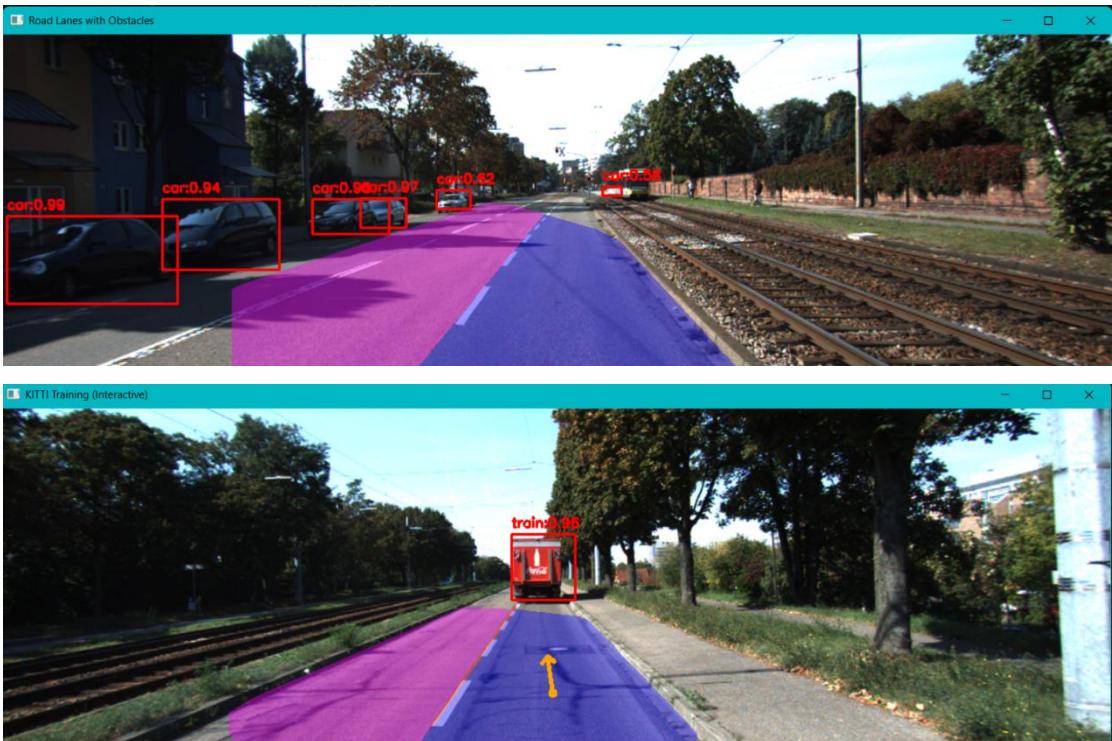


Όμως αυτή η μέθοδος παρουσίασε κιόλας αρκετά false positives, δηλαδή να εμφανίζονται bounding boxes σε σημεία χωρίς αντικείμενα, όπως παρακάτω, οπότε προτιμήθηκε τη μέθοδος με το νευρωνικό δίκτυο.



A2: YOLO net

Αρχείο κώδικα: [camera_complete.py](#) → Υλοποιεί A1-A2-A3-A4.



Το νευρωνικό γείτονα είναι πολύ γρηγορότερο και ακριβέστερο στην ανίχνευση εμποδίων.

A3: Διάνυσμα Κίνησης

Αρχείο κώδικα: [camera_complete.py](#) → Υλοποιεί A1-A2-A3-A4.

Για την εκτίμηση κατεύθυνσης κίνησης, χρησιμοποιείται ένα απλοποιημένο μοντέλο στο οποίο το διάνυσμα κατεύθυνσης ταυτίζεται με τη γραμμή θέασης της κάμερας, δηλαδή "όπου κοιτάει η κάμερα" θεωρείται και η κατεύθυνση της κίνησης. Αυτό γίνεται με χρήση του calibration matrix (P2), που επιτρέπει την προβολή 3D σημείων στην εικόνα 2D.

Συγκεκριμένα: Ορίζονται δύο σημεία στο 3D χώρο (σε KITTI συντεταγμένες), σε απόσταση arrow_length το ένα από το άλλο, πάνω στον άξονα Z (μπροστά). Προβάλλονται στην εικόνα με χρήση της κάμερας (μέσω του P2) και σχεδιάζεται το αντίστοιχο βέλος στην εικόνα, για να δείξει την "προτεινόμενη" κατεύθυνση.

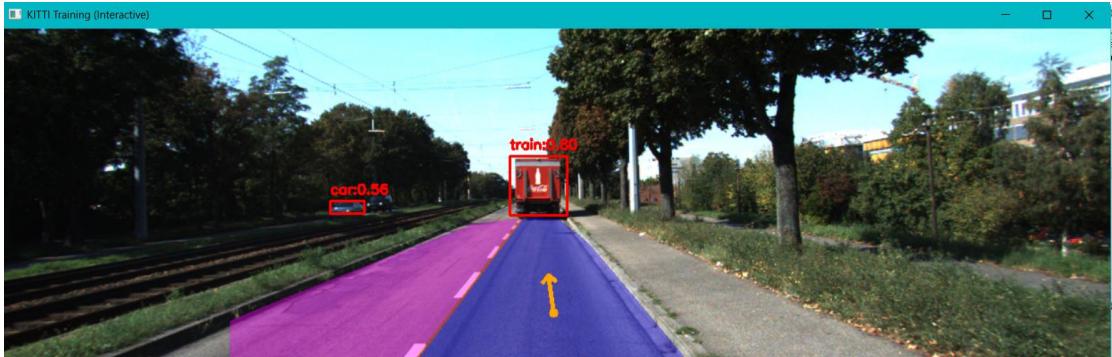
Εκτίμηση Απόστασης Εμποδίου: Για να προσαρμοστεί το μήκος του βέλους ανάλογα με την ύπαρξη εμποδίων μπροστά εξετάζονται τα YOLO boxes (ανιχνευμένα εμπόδια). Επιλέγεται το κοντινότερο εμπόδιο εντός κώνου $\pm 5^\circ$ από το κέντρο της εικόνας (μπροστά). Η θέση (κάθετο pixel y) του κάτω μέρους του box χρησιμοποιείται για χοντρική εκτίμηση της απόστασης: όσο πιο κάτω το pixel (στο image space), τόσο πιο κοντά θεωρείται το εμπόδιο. Η απόσταση μετατρέπεται σε εκτίμηση μήκους του βέλους με γραμμική παρεμβολή ($3.5m - 11m$).

Χρωματισμός Βέλους: Ο χρωματισμός του βέλους δηλώνει το πόσο "ασφαλής" είναι η πορεία. **Κόκκινο** αν η εκτιμώμενη απόσταση εμποδίου είναι πολύ μικρή (βέλος κοντό).

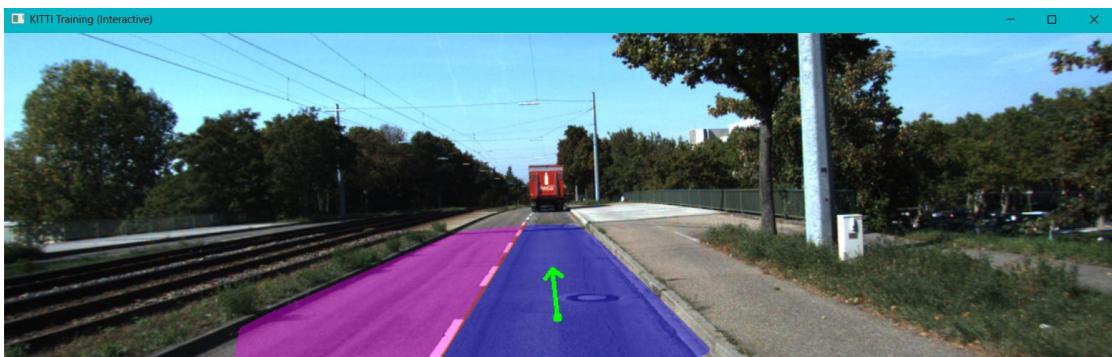
Πορτοκαλί για μέτριες αποστάσεις (μέσο μήκος). **Πράσινο** όταν δεν εντοπίζεται εμπόδιο κοντά (μέγιστο βέλος).

Εκτύπωση στο Τερματικό: Κατά την εκτέλεση τυπώνεται η εκτιμώμενη απόσταση του κοντινότερου εμποδίου ([INFO] Estimated obstacle distance: X.Xm) καθώς και το αντίστοιχο μήκος του βέλους ([INFO] Adaptive arrow length: X.Xm)

Παράδειγμα ανίχνευσης κοντινού εμποδίου (πορτοκαλί βέλος):



Το φορτηγό απομακρύνθηκε και δεν ανιχνεύεται πια σαν εμπόδιο (πράσινο βέλος):

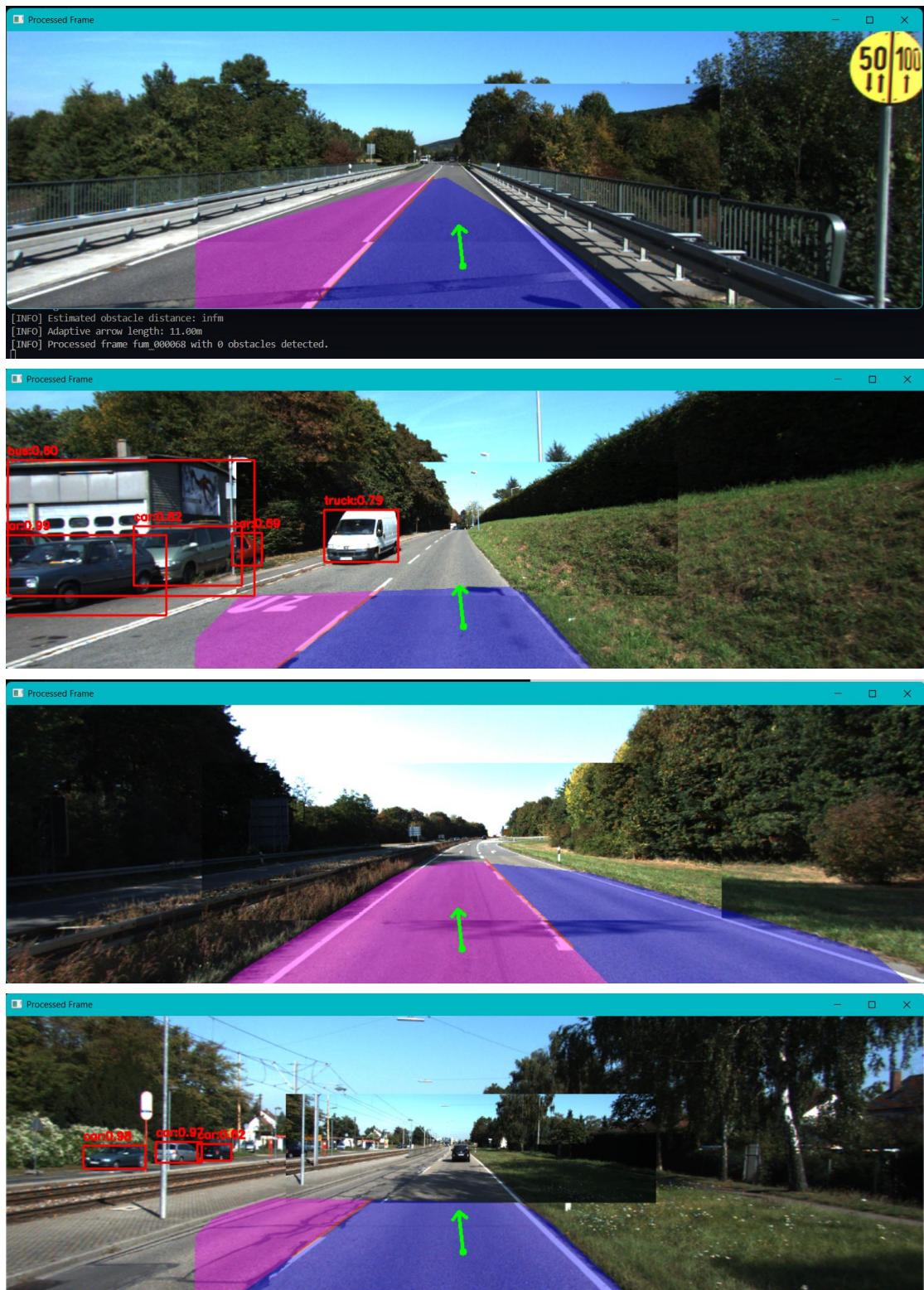


Part_A1-2-3 video (stereo version): <https://www.youtube.com/watch?v=5cIPDILlWHz>

A4: Ανίχνευση Τοίχου

Αρχείο κώδικα: [camera_complete.py](#) → Υλοποιεί A1-A2-A3-A4.

Για την προσομοίωση τοίχου έγινε συρραφή συνεχόμενων εικόνων του KITTI. Αυτό έγινε και για την δεξιά και για την αριστερή εικόνα με ίδιες διαστάσεις. Η δαδικασία πραγματοποιήθηκε για 4 εικόνες.



Σε καμία από τις 4 εικόνες δεν βρέθηκε «εμπόδιο», ενώ μάλιστα στις δύο από αυτές ο RANSAC θεώρησε την επιφάνεια του τοίχου μέρος του επιπέδου του δρόμου. Το διάνυσμα κίνησης δείχνει ότι το όχημα θα συγκρουστεί με τον τοίχο, καθώς η κάμερά δεν μπορεί να εντοπίσει τέτοιο εμπόδιο.

Σύντομες Οδηγίες χρήσης

A1-2-3-4 ΤΕΛΙΚΗ ΈΚΔΟΣΗ:

Camera_vs_LIDAR\PART_A\stereoVersion\camera_complete.py:

Ζητούνται command line arguments για τα paths του KITTI dataset και του YOLO(έχει χρησιμοποιηθεί το YOLO v3). Τα default τους είναι τα Path που βόλευαν εμένα.

Αν δεν δοθεί κάποιο arg, τρέχει για όλο το kitti training dataset αν το βρει By default.

Αν δοθεί --testing, τα directories αλλάζουν από treining σε testing.

Αν δοθεί --video, παύει να είναι σε interactive mode το πρόγραμμα και τρέχει σαν live video, ενώ αποθηκεύει αρχείο .avi στο ίδιο directory με το script. (Μπορεί να γίνει και για testing dataset αυτό με --testing --video).

Με --index=.., δίνεται η επιλογή να εμφανιστεί μόνο μία εικόνα, αυτή με index ...

Με -wall τρέχει το ερώτημα B4 με τα fake directories μα τις πειραγμένες εικόνες (περνούνται τα ορίσματα των left_dir, calib_dir, right_dir μέσα σε μια έτοιμη συνάρτηση για προσωπική ευκολία.)

\Camera_vs_LIDAR\PART_A\oneCamera\obstacle_detection_cv_A2.py ανίχνευση εμποδίων με μόνο υπολογιστική όραση:

Τρέχει στο KITTI training dataset ολόκληρο

\Camera_vs_LIDAR\PART_A\oneCamera\road_detector_A1.py ανίχνευση δρόμου με μόνο μία κάμερα:

Επιλογές στο χρήστη με Input για ποιο group εικόνων θέλει να τεστάρει

ΜΕΡΟΣ Β

Camera_vs_LIDAR\PART_B\lidar_complete.py

B1 – Ανίχνευση δρόμου

Βήματα ανίχνευσης δρόμου

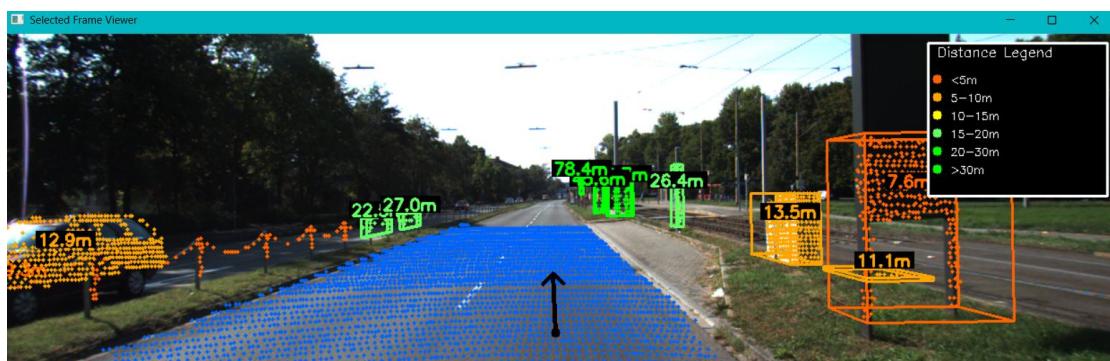
- Φόρτωση και προεπεξεργασία point cloud: Φορτώνεται το .bin αρχείο με το point cloud (load_bin). Γίνεται spatial crop για να διατηρηθούν μόνο τα σημεία εμπρός από το όχημα ($x > 0, |y| < 10$). Φίλτραρονται ύψος: μόνο σημεία εντός $[-3, 2]$ μέτρων στον άξονα z. Εφαρμόζεται voxel downsampling (0.08m) για μείωση της πυκνότητας. Σημείωση: το down sampling γίνεται σε Ordered σημεία για να είναι ντετερμινιστικό το πρόγραμμα και να μην “μπαίνει” τυχαιότητα από το down sampling.
- Ανίχνευση δρόμου με Multi-plane RANSAC: Εκτελείται επαναληπτικό RANSAC για να εντοπιστούν έως 3 επίπεδα (πιθανά επίπεδα δρόμου). Για κάθε επίπεδο, επιλέγονται τα inliers (κοντινά σημεία στο επίπεδο). Όλα τα inliers από τα επίπεδα συνδυάζονται.
- Προσαρμοστικό φίλτραρισμα ύψους (Adaptive Height Filtering): Το cloud χωρίζεται σε 2D grid. Για κάθε cell, υπολογίζεται το τοπικό επίπεδο εδάφους (percentile z). Κρατούνται μόνο σημεία που είναι max 10cm πάνω από το τοπικό έδαφος.
- Στόχος: φίλτραρισμα θορύβου και μη-οδοστρώματος.
- Αφαίρεση πεζοδρομίων με PCA στα normals: Εκτελείται clustering (DBSCAN) για να βρεθούν τα κύρια τμήματα δρόμου. Επιλέγεται το μεγαλύτερο cluster ως "κύριος δρόμος". Ύστερα γίνεται PCA analysis σε τοπικά patches. Αν το τρίτο eigenvalue είναι μεγάλο \rightarrow υπάρχει απότομη μεταβολή σε z (πιθανόν πεζοδρόμιο). Άρα σημεία με υψηλή κατακόρυφη τραχύτητα (high Z variance) χαρακτηρίζονται ως rough και αφαιρούνται. Υπολογίζονται όρια αριστερού/δεξιού πεζοδρομίου και αφαιρούνται τα σημεία έξω από αυτά (percentile).
- Τελική έξοδος: Το main_road περιέχει τα τελικά σημεία δρόμου, χωρίς πεζοδρόμια.

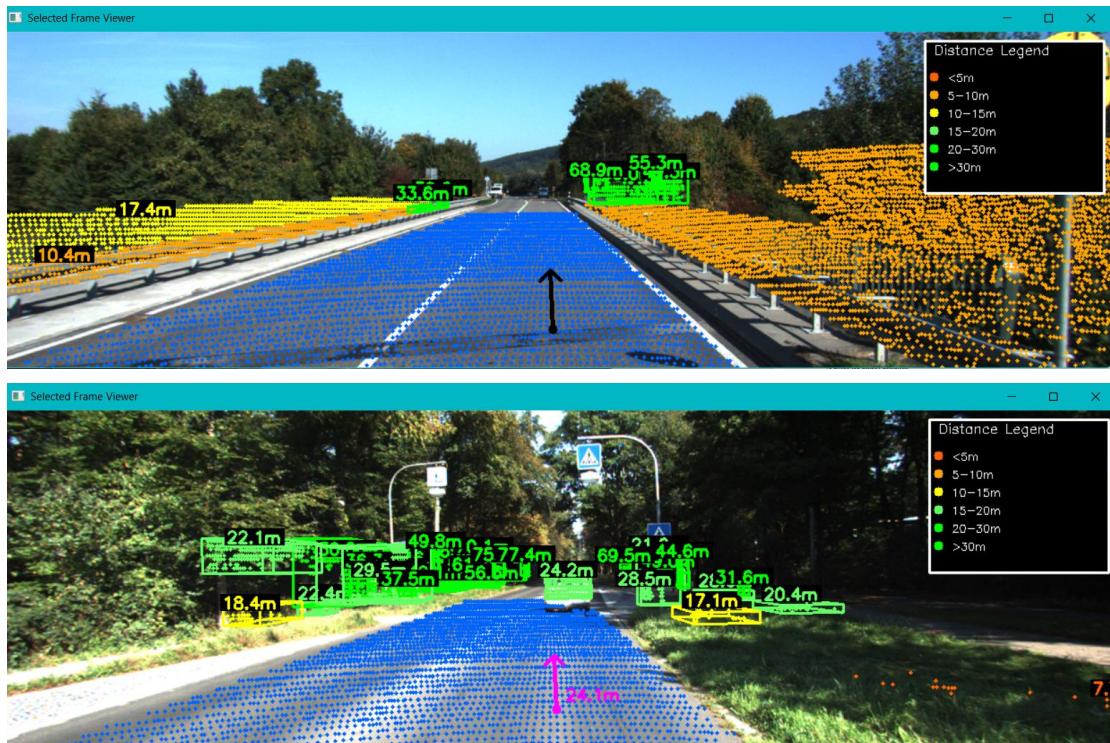
Το σύστημα μπορεί να προβάλλει αυτά τα σημεία πάνω στην εικόνα (χρώμα: μπλε).

Προαιρετικά, τα πεζοδρόμια προβάλλονται ως Αριστερά: κίτρινο, Δεξιά: πράσινο (στην εικόνα του KITTI προβάλλονται ανάποδα από ότι στον 3D χώρο)

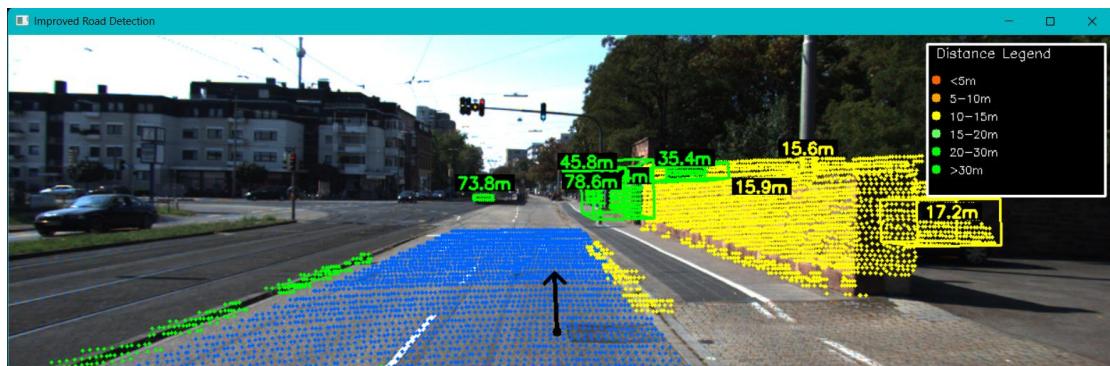
Ενδεικτικά αποτελέσματα:

Μάσκα Δρόμου χωρίς πεζοδρόμια:





Στην παρακάτω εικόνα φαίνεται η άκρη του πεζοδρομίου που χρησιμοποιήθηκε για την αφαίρεση του πλακόστρωτου κομματιού που θεωρούταν δρόμος (cmd arg --curbs):



Σημείωση: Στις εκδοχές με κάμερα, η συγκεκριμένη εικόνα ήταν κλασσικό πρόβλημα, καθώς το πλακόστρωτο πεζοδρόμιο, ενωνόταν με τον δρόμο:

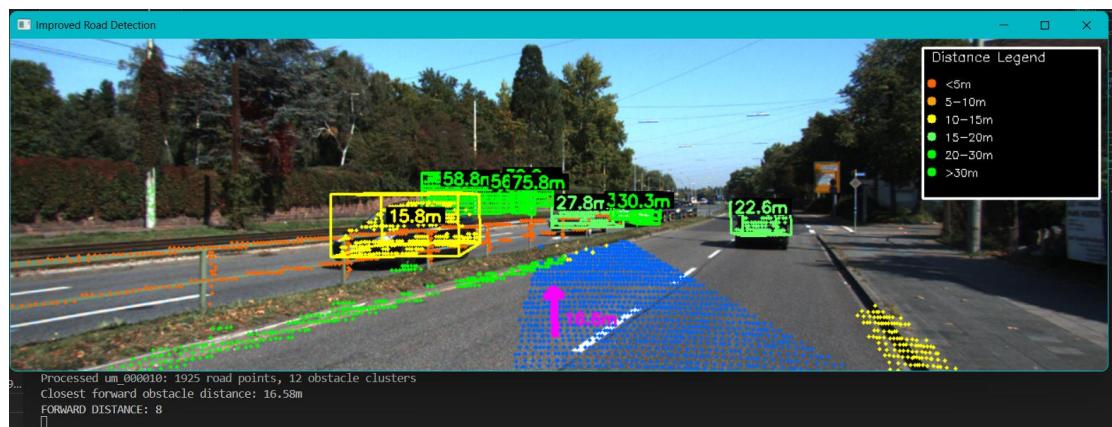
Από την τελική έκδοση `camera_complete.py`



Συμπέρασμα: Καταλήγουμε ότι το LIDAR είναι πιο ανθεκτικό από την κάμερα σε αντίξοες/απρόσμενες συνθήκες. Η προσέγγιση αυτή παρουσιάζει όμως και κάποια προβλήματα.

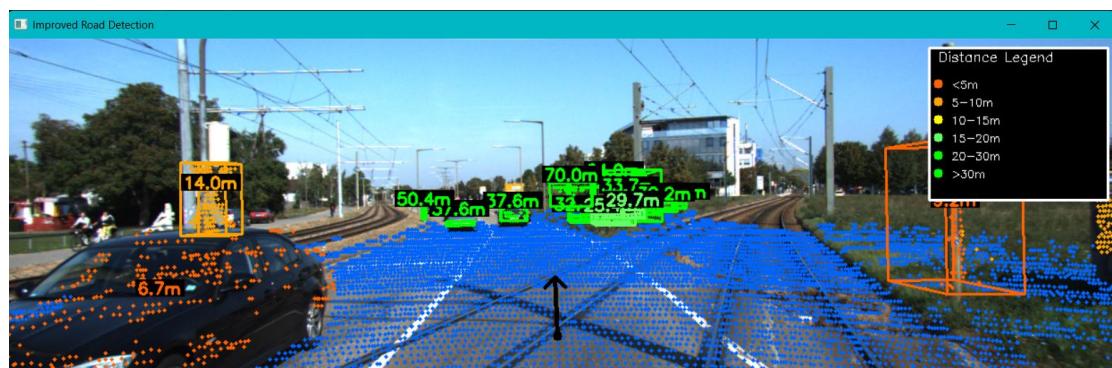
Προβλήματα από τον διαχωρισμό πεζοδρομίων:

Συγκεκριμένα στο στάδιο αφαίρεσης πεζοδρομίων, γίνεται διαχωρισμός μεταξύ αριστερού και δεξιού curb χρησιμοποιώντας τον άξονα για τον LIDAR και το μέσο του κύριου δρόμου ως αναφορά (δηλ. median y-value). Τα σημεία που βρίσκονται αριστερά του κέντρου ταξινομούνται ως αριστερά πεζοδρόμια, και αντίστοιχα για τα δεξιά. Αυτό λειτουργεί καλά όταν το όχημα κινείται σε ευθεία και η σκηνή είναι συμμετρική. Όμως στην παρακάτω στροφή αυτή η λογική αποτυγχάνει με αποτέλεσμα να κόβει σωστά κομμάτια της μάσκας του δρόμου:

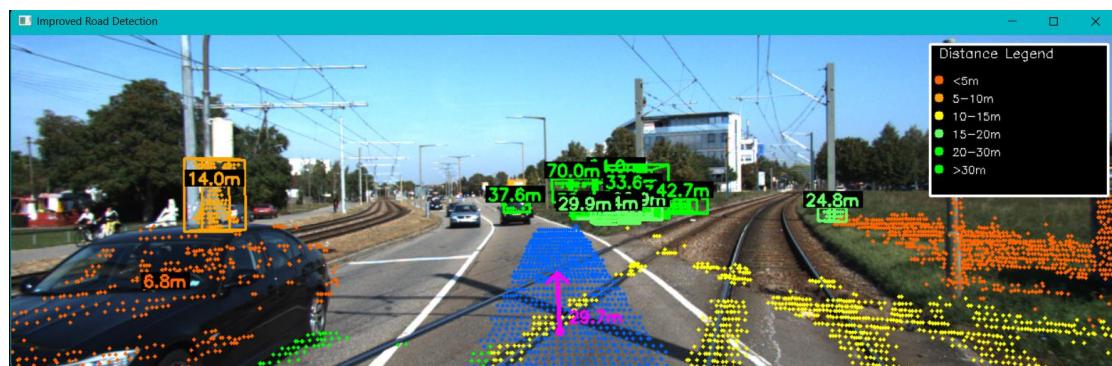


Επιπλέον υπάρχει η περίπτωση των σιδηροδρομικών ραγών.

Αρχική μάσκα:



Μάσκα με αφαίρεση πεζοδρομίου:



Οι ράγες, όπως αναμενόμενο, έχουν και αυτές ψηλό z variance και ο δρόμος αριστερά και δεξιά τους αφαιρείται! Βέβαια η λωρίδα που απομένει στην δεύτερη φωτογραφία είναι το μόνο ασφαλές σημείο να κατευθυνθεί το αμάξι, αλλά και πάλι υπάρχουν λάθη στον σωστό σχηματισμό της μάσκας του δρόμου.

B2 – Ανίχνευση εμποδίων

Η ανίχνευση εμποδίων βασίζεται στην γεωμετρική ανάλυση των σημείων LIDAR, με στόχο να εντοπιστούν αντικείμενα που προεξέχουν σημαντικά πάνω από το επίπεδο του δρόμου. Το σύστημα λειτουργεί αποκλειστικά με 3D πληροφορία (χωρίς YOLO ή PointNet) και υλοποιείται με τα εξής στάδια:

Obstacle Detection Pipeline

1. Προετοιμασία δεδομένων:

Έπειτα από την ανίχνευση και καθαρισμό του δρόμου (B1), εξάγονται τα σημεία του κύριου οδοστρώματος (main_road). Το πλήρες point cloud (xyz) περιέχει όλα τα σημεία του LIDAR για το frame, ενώ το main_road χρησιμοποιείται ως αναφορά για να υπολογιστεί τοπικά το ύψος του εδάφους. Για την επιτάχυνση της διαδικασίας, δημιουργείται δένδρο cKDTree βασισμένο στις x και y συντεταγμένες των σημείων του δρόμου. Η χρήση μόνο των x–y αξόνων (οριζόντιο επίπεδο) επιτρέπει ταχύτερη αναζήτηση κοντινών σημείων, αδιαφορώντας προσωρινά για μικρές αποκλίσεις στο z.

2. Εντοπισμός υπερυψωμένων σημείων

Αρχικά, για κάθε σημείο στο πλήρες point cloud αναζητούνται τα 5 κοντινότερα σημεία από το main_road (μέσω k-NN αναζήτησης στο 2D επίπεδο xy). Υπολογίζεται το μέσο ύψος (z) του τοπικού δρόμου σε εκείνη τη θέση. Αν το σημείο βρίσκεται τουλάχιστον 0.5 μέτρα πάνω από το τοπικό ύψος του δρόμου, θεωρείται ως πιθανό εμπόδιο. Τα σημεία που πληρούν το κριτήριο αυτό αποθηκεύονται στο σύνολο obstacle_points.

3. Ομαδοποίηση σημείων σε αντικείμενα

Τα obstacle_points οργανώνονται σε ανεξάρτητα clusters με χρήση του αλγορίθμου DBSCAN. Εδώ, εφαρμόζεται το clustering σε πλήρεις 3D συντεταγμένες (x, y, z). Η επιλογή του 3D χώρου, αντί για μόνο το xy επίπεδο, επιτρέπει την ακρίβεια στον διαχωρισμό αντικειμένων διαφορετικού ύψους, αποφεύγοντας την ενοποίηση ανεξάρτητων εμποδίων που βρίσκονται κάθετα το ένα πάνω στο άλλο (π.χ. πινακίδα και κολώνα). Ταυτόχρονα, μικρές διαφορές ύψους μέσα στο ίδιο αντικείμενο (π.χ. οροφή και καπό αυτοκινήτου) δεν οδηγούν σε τεχνητό διαχωρισμό. Κρατούνται μόνο τα clusters με τουλάχιστον 5 σημεία, ώστε να αποφεύγονται false positives από θόρυβο. Κάθε cluster αντιστοιχεί σε ένα φυσικό αντικείμενο: π.χ. σταθμευμένο όχημα, κάδος, κώνος, πεζός.

4. Προβολή εμποδίων στην εικόνα

Για κάθε εμπόδιο cluster υπολογίζεται το γεωμετρικό κέντρο και η Ευκλείδεια απόσταση από το όχημα. Τα σημεία του προβάλλονται στην κάμερα (μέσω του πίνακα P). Σχεδιάζονται τα σημεία του εμποδίου με χρώμα που εξαρτάται από την απόσταση και περιβάλλονται από 3D Bounding Box. (Αν έστω και μία γωνία του κουτιού είναι εκτός ορίων της εικόνας δεν θα εμφανιστεί, αλλά το cluster θα συνεχίσει να είναι εμφανές). Στην εικόνα εμφανίζεται επίσης κείμενο που δείχνει την απόσταση από το κέντρο του εμποδίου (π.χ. “12.3m”).

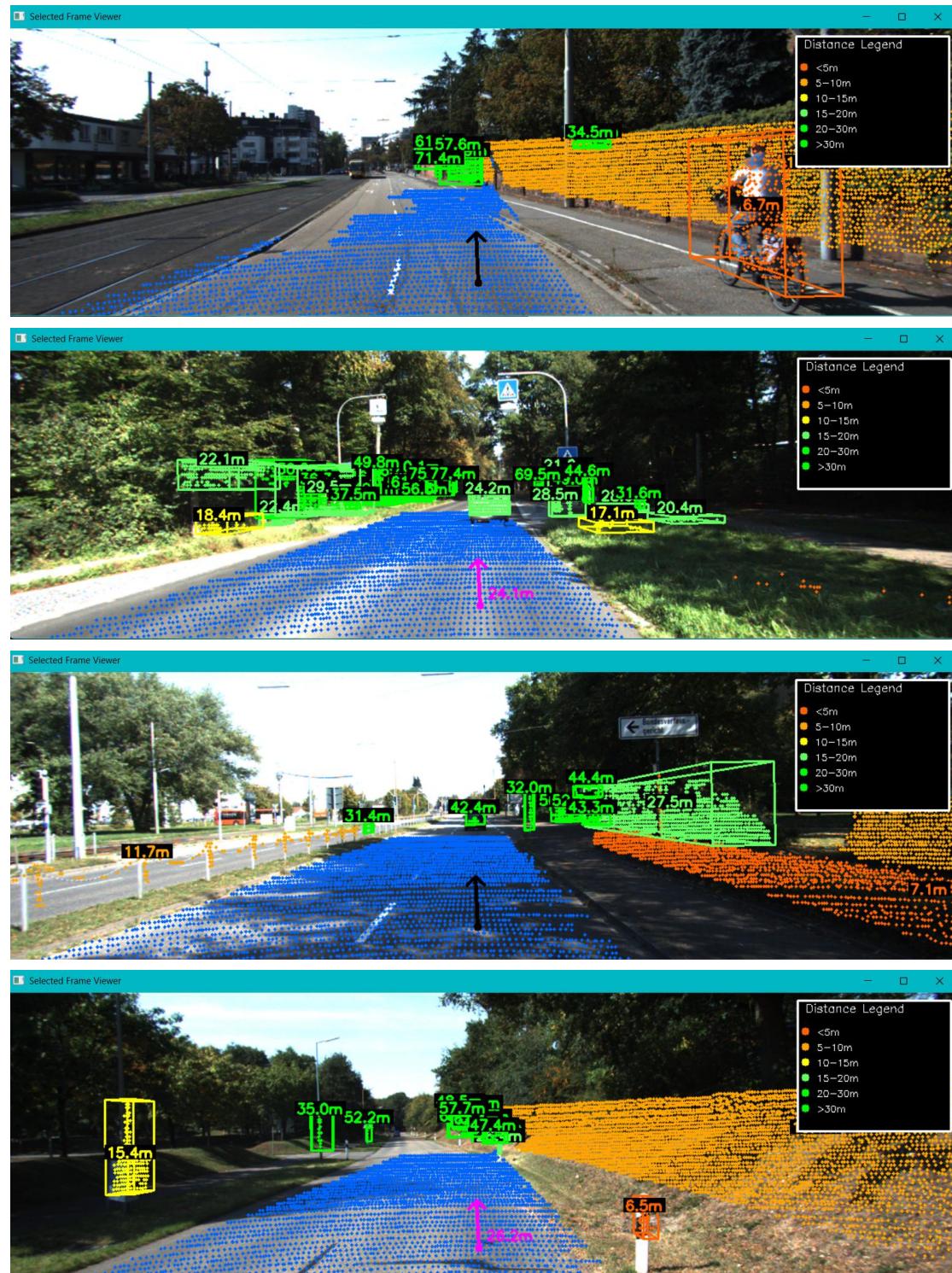
Η χρωματική κωδικοποίηση υποδεικνύει την εγγύτητα:

- Κόκκινο: πολύ κοντά (<5m)
- Πορτοκαλί: 5–10m
- Κίτρινο: 10–20m
- Ανοιχτό πράσινο: 20–30m
- Πράσινο: >30m 5.

Παρατηρήσεις:

Η χρήση του τοπικού επιπέδου του δρόμου (και όχι ενός καθολικού flat plane) για την εκτίμηση του σχετικού ύψους είναι πιο αξιόπιστη σε πραγματικά σενάρια με κλίσεις, στροφές ή αλλαγές υψομέτρου.

Ενδεικτικά αποτελέσματα:



B3 – Διάνυσμα Κίνησης

Το βέλος πορείας προβάλλεται πάνω στο οδόστρωμα και υποδεικνύει την κατεύθυνση και της προτεινόμενης πορείας. Το μήκος και το χρώμα του διανύσματος ρυθμίζονται δυναμικά, ανάλογα με την εγγύτητα εμποδίων στην κατεύθυνση κίνησης.

Για τη δημιουργία του ακολουθούνται τα εξής βήματα:

1. Ανάκτηση Calibration:

Αρχικά φορτώνονται τα calibration δεδομένα (Tr_velo_to_cam, R0_rect, P2) από αρχείο KITTI. Υπολογίζεται ο πίνακας προβολής. (πίνακας για να κάνουμε μετατροπή σημείων από το 3D LIDAR σύστημα στο 2D επίπεδο της κάμερας, με διόρθωση παραμόρφωσης.)

2. Υπολογισμός δυναμικού μήκους διανύσματος

Καλείται η calculate_adaptive_arrow_length() με είσοδο τα εμπόδια που έχουν ήδη εντοπιστεί και ομαδοποιηθεί (obstacle_clusters). Αν δεν υπάρχουν εμπόδια μπροστά από το αυτοκίνητο, το βέλος παίρνει το μέγιστο μήκος (π.χ. 6m). Αν υπάρχει εμπόδιο εντός της "κωνικής περιοχής πορείας" $\pm 2.5^\circ$, τότε το μήκος μειώνεται αναλογικά ως προς την απόσταση του εμποδίου. Για αποστάσεις κάτω από safe_distance (π.χ. 15m), εφαρμόζεται γραμμική παρεμβολή μεταξύ min_length (1m) και max_length (6m), οπότε το μήκος του βέλους μειώνεται.

3. Εύρεση κοντινότερου εμποδίου στην πορεία

Η get_closest_forward_obstacle_distance() σαρώνει όλα τα σημεία των clusters και αγνοεί σημεία πίσω από το όχημα ($x < 0.5$). Κρατά μόνο σημεία εντός $\pm 2.5^\circ$ (μετρώντας γωνία σε σχέση με τον άξονα X) και επιστρέφει την μικρότερη απόσταση προς ένα εμπόδιο που βρίσκεται ακριβώς στην πορεία του οχήματος. Αυτό το εμπόδιο επηρεάζει άμεσα το μήκος του διανύσματος. Στην εικόνα θα εμφανιστεί στο τέλος η απόσταση του παρατηρητή/οδηγού από το κοντινότερο σημείο του εμποδίου.

4. Προβολή και σχεδίαση του βέλους

Το βέλος τοποθετείται σε σταθερό ύψος ($z \approx -1.7m$, δηλαδή στο επίπεδο του δρόμου). Επιλέγεται ένα εύρος πιθανών αποστάσεων για το σημείο εκκίνησης του βέλους (π.χ. 8, 10, 12, 15, 20m) και επιχειρείται να προβληθεί το τμήμα [start → end] στο 2D επίπεδο εικόνας. Στο KITTI dataset για 8 μέτρα το βέλος φαίνεται με ευκρίνεια.

5. Χρώμα Βέλους

Το χρώμα είναι ανάλογο του μήκους (κόκκινο αν το βέλος είναι κοντό ⇒ κίνδυνος μπροστά). Το πάχος επίσης είναι ανάλογο του κινδύνου (παχύτερο = πιο κρίσιμη κατάσταση). Αν υπάρχει κοντινό εμπόδιο, εμφανίζεται πάνω από το βέλος η αντίστοιχη απόσταση σε μέτρα.

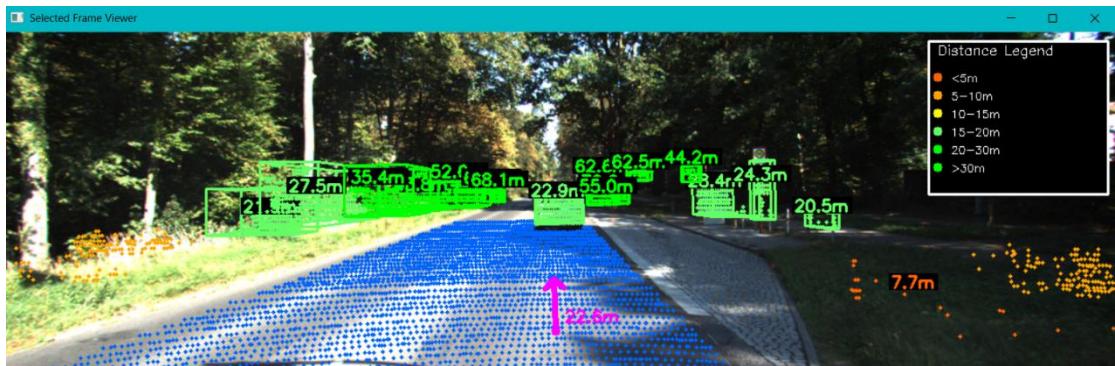
Όχι εμπόδιο μπροστά → Βέλος μήκους 6m, χρώμα μαύρο → Ελεύθερη πορεία

Εμπόδιο μακριά (>10m) → Βέλος μήκους 4–5.5m, χρώμα μωβ → Αρκετά ασφαλές

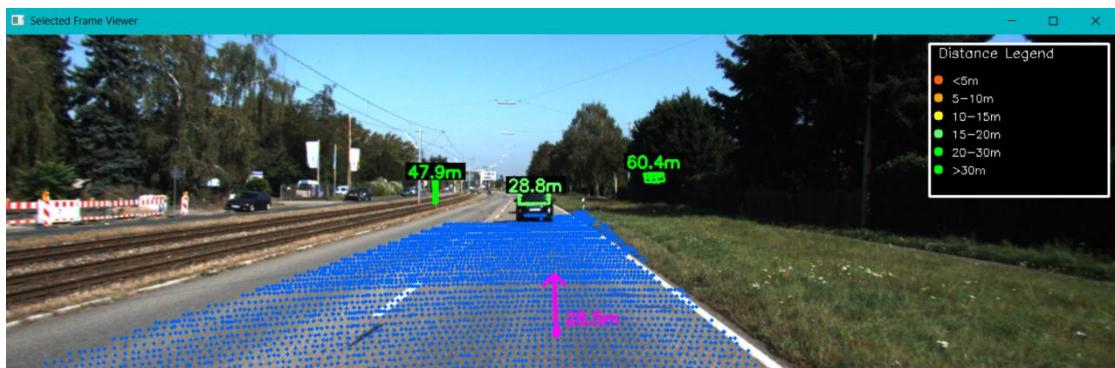
Εμπόδιο κοντά (5–10m) → Βέλος μήκους 2.5–4m, χρώμα πορτοκαλί → Μέτριος κίνδυνος

Εμπόδιο πολύ κοντά (<5m) → Βέλος μήκους 1–2.5m, χρώμα κόκκινο → Άμεσος κίνδυνος

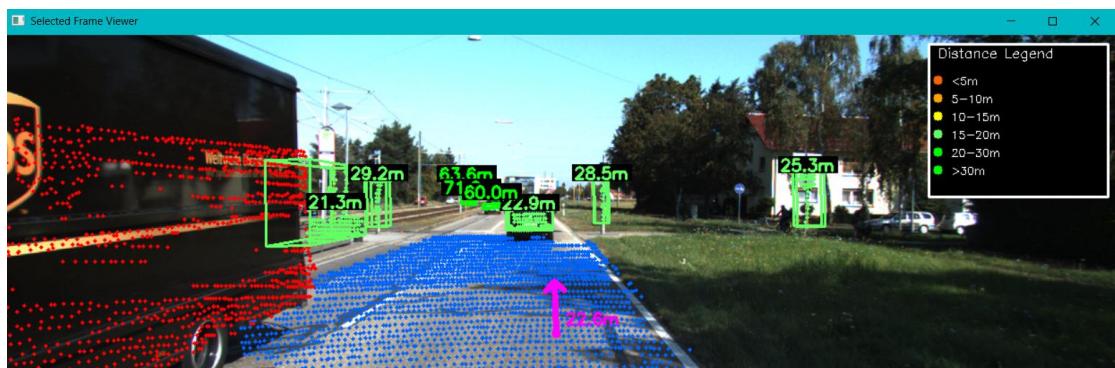
Ενδεικτικά αποτελέσματα



Ο λόγος που δίπλα στο βέλος αναγράφεται διαφορετική απόσταση από ότι στο αμάξι είναι ότι το κέντρο του αμαξιού απέχει 22.9m αλλά το κοντινότερο σημείο του απέχει 22.6m



Παρατηρούμε επίσης ότι χάρη στην μικρή κωνική γωνία τα εμπόδια στην αριστερή λωρίδα δεν παίζουν ρόλο στο χρώμα και μήκος του βέλους, μόνο τα μπροστινά εμπόδια μας επηρεάζουν:

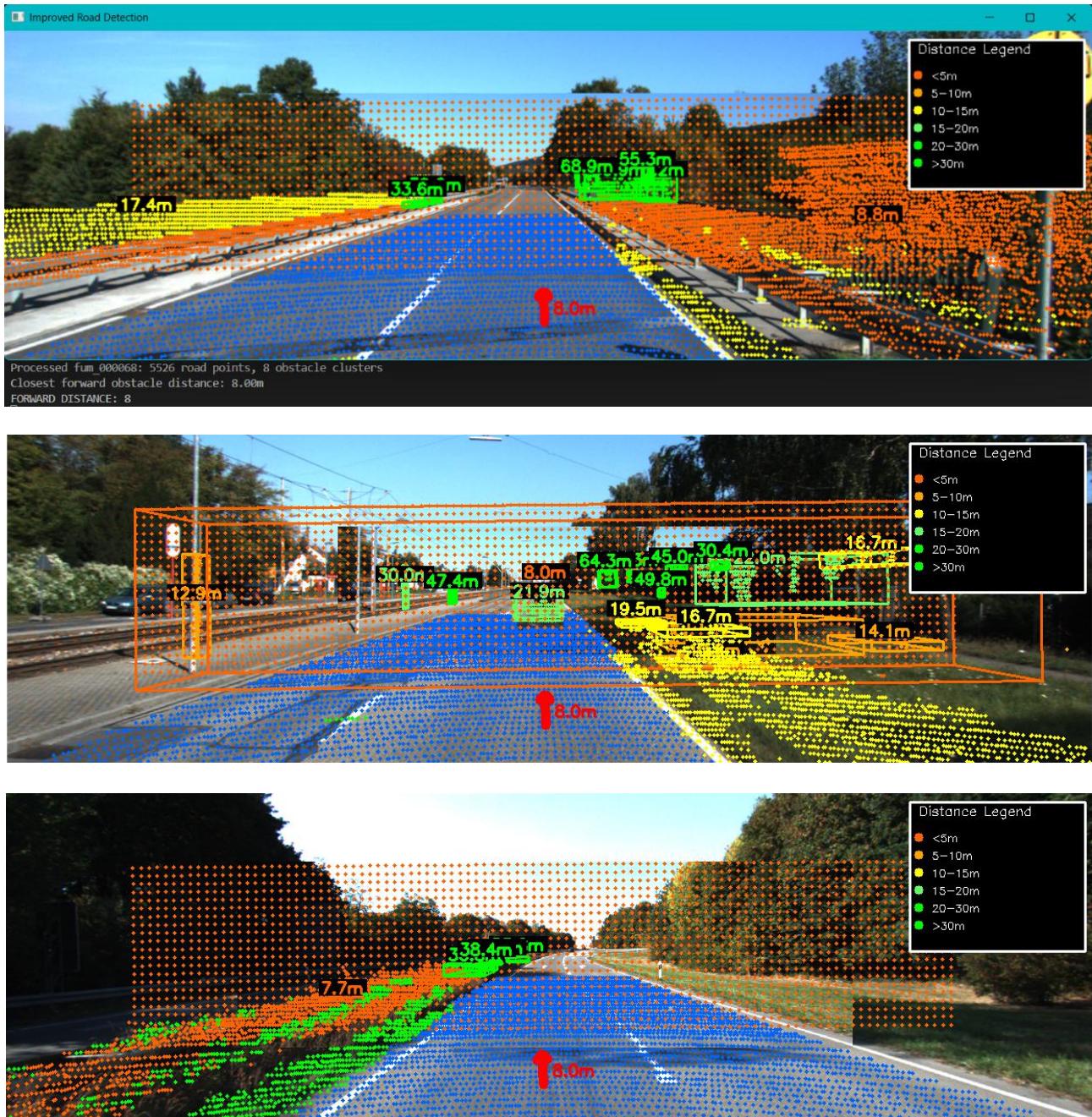


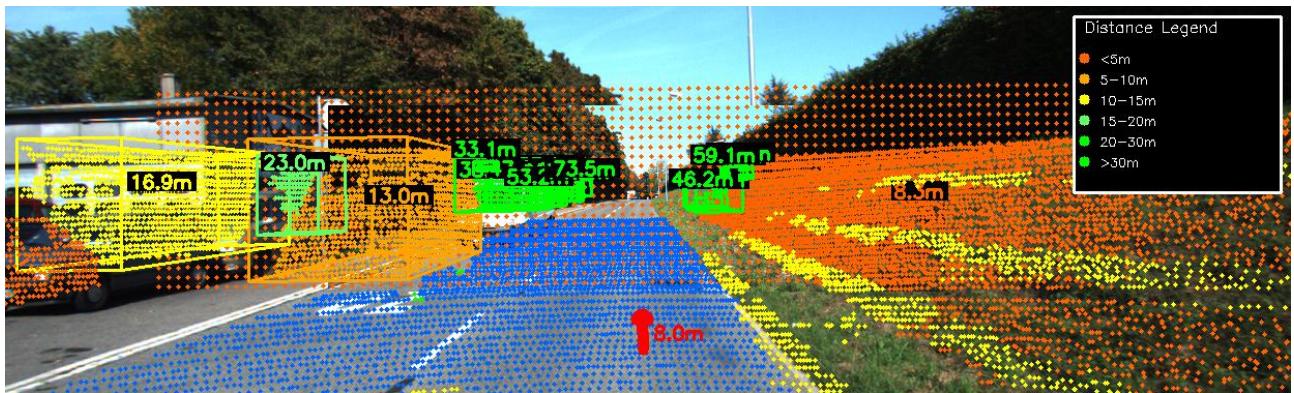
B4 – Ανίχνευση τοίχου

Με το αρχείο Camera_vs_LIDAR\PART_B\B4_wallcreation.py:

Έγινε προσθήκη τοίχου συγκεκριμένων διαστάσεων στο point cloud. Ύστερα αυτό αποθηκεύτηκε σε νέο αρχείο bin. Όπως και στα προηγούμενα ερωτήματα, αυτό θα προβληθεί στην εικόνα.

Με το Camera_vs_LIDAR\PART_B\lidar_complete.py --wall μπορούμε να δούμε αν το pipeline θα ανιχνεύσει τον τοίχο. Χρησιμοποιήθηκαν οι ίδιες 4 φωτογραφίες με το μέρος Α:





Και στις 4 περιπτώσεις το πρόγραμμα εντοπίζει κοντινότερο εμπόδιο στα 8 μέτρα, εκεί που τοποθετήθηκε ο τοίχος. Το βέλος γίνεται κόκκινο, γεγονός που δείχνει ότι βρισκόμαστε επικίνδυνα κοντά σε εμπόδιο. Επομένως ο τοίχος ανιχνεύτηκε επιτυχώς από το radar.

Βιβλιογραφία

1. <https://pages.cs.wisc.edu/~kponto/publications/HPE.pdf>
2. <https://core.ac.uk/download/pdf/77049476.pdf>
3. <https://pjreddie.com/darknet/yolo/>
4. <https://arxiv.org/pdf/2501.07245v1>
5. <https://pages.cs.wisc.edu/~kponto/publications/HPE.pdf>
6. https://pointclouds.org/documentation/tutorials/normal_estimation.html
7. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7877966>
8. <https://ieeexplore.ieee.org/document/10311086>