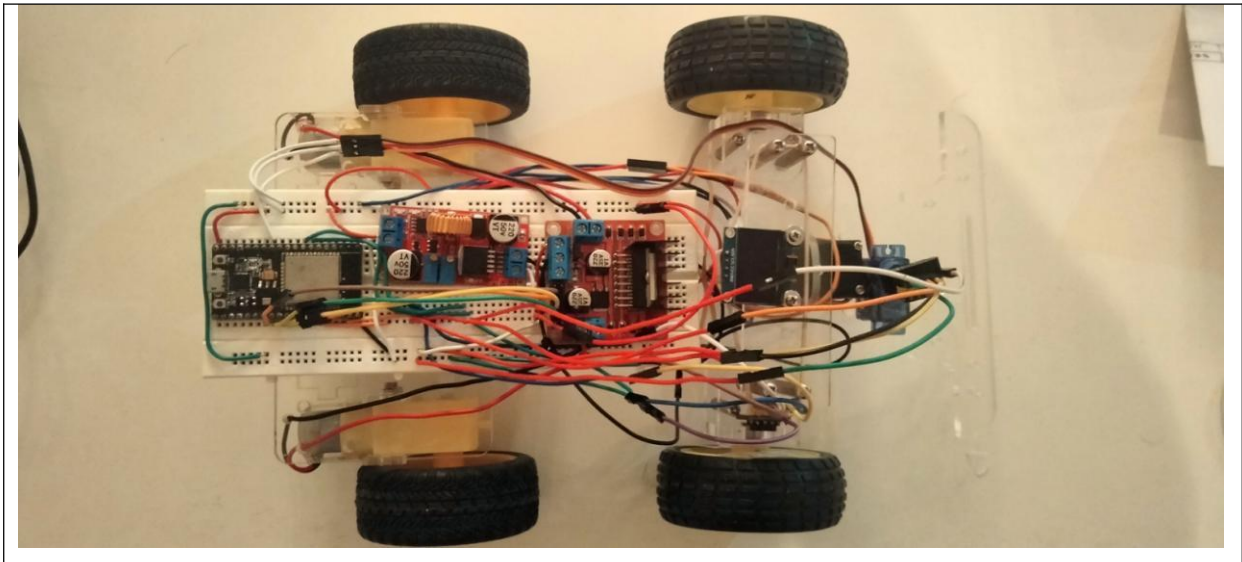


ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και υλοποίηση τροχοφόρου οχήματος το οποίο θα εκτελεί κίνηση σύμφωνα με προγεγραμμένη πορεία μέσω μιας διεπαφής



Του φοιτητή
Κυριακούλης Παναγιώτης
Αρ. Μητρώου: iee2019088

Επιβλέπων
Γιακουμής Άγγελος
Αναπληρωτής Καθηγητής

Θεσσαλονίκη Σεπτέμβριος 2026

Τίτλος Δ.Ε.: Σχεδίαση και υλοποίηση τροχοφόρου οχήματος το οποίο θα εκτελεί κίνηση σύμφωνα με προγεγραμμένη πορεία μέσω μιας διεπαφής

Κωδικός Δ.Ε. 25277

Ονοματεπώνυμο φοιτητή: Κυριακούλης Παναγιώτης

Ονοματεπώνυμο εισηγητή: Γιακουμής Άγγελος

Ημερομηνία ανάληψης Δ.Ε. : 27/06/2025

Ημερομηνία περάτωσης Δ.Ε. : 06/09/2026

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

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Κυριακούλη Παναγιώτη που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

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

«Στην οικογένεια μου»

Πρόλογος

Η ανάπτυξη και υλοποίηση αυτόνομων ρομποτικών συστημάτων είναι ένας από τους πιο ενδιαφέροντες τεχνολογικούς κλάδους της σύγχρονης επιστήμης. Στο πλαίσιο της παρούσας διπλωματικής εργασίας εξετάζεται η μελέτη και η υλοποίηση ενός αυτόνομου τροχοφόρου οχήματος το οποίο θα εκτελεί κίνηση σύμφωνα με προγεγραμμένη πορεία μέσω μιας διεπαφής.

Περίληψη

Στην παρούσα εργασία εξερευνάται η κατασκευή ενός αυτόνομου τροχοφόρου οχήματος το οποίο έχει τη δυνατότητα να εξερευνά τον χώρο και να διανύει μια πορεία μέσω μιας διεπαφής. Για την εξερεύνηση και ανίχνευση εμποδίων το όχημα είναι εξοπλισμένο με δύο αισθητήρες time of flight TOF400c από τους οποίους ο ένας είναι τοποθετημένος πάνω σε έναν motor servo SG90. Το όχημα διαθέτει δύο κινητήρες DC, οι οποίοι είναι τοποθετημένοι στους πίσω τροχούς και έναν οδηγό L298N motor driver ελέγχει τη ρύθμιση της λειτουργίας τους. Ένα δεύτερο motor servo SG90 είναι τοποθετημένο στον μπροστινό άξονα και είναι υπεύθυνο για τη σωστή κατεύθυνση του οχήματος. Το σύστημα τροφοδοτείται από δύο Li-ion μπαταρίες των 3.7V με έναν XL4015 step down converter να ρυθμίζει την τάση. Για την χαρτογράφηση του χώρου έχει δημιουργηθεί μια εφαρμογή με τη χρήση του Godot η οποία είναι υπεύθυνη για την μεταφορά δεδομένων από και προς τον μικροελεγκτή, την χαρτογράφηση του χώρου, την δημιουργία της διαδρομής, την εξομάλυνση διαδρομής και την δημιουργία μιας πορείας η οποία μπορεί να ακολουθήσει το όχημα.

Design and implementation of a vehicle that follows a predefined route through an interface

Kyriakoulis Panagiotis

Abstract

This thesis explores the design and construction of an autonomous wheeled vehicle capable of exploring its environment and following routes generated through a graphical interface. For environmental scanning, obstacle detection and odometry, the vehicle is equipped with two TOF400C time of flight sensors. One sensor is mounted on an SG90 servo motor, allowing it to scan the surrounding area, while the second sensor monitors the rotation of a rear wheel.

The vehicle is driven by two DC motors connected to the rear wheels and controlled through an L298N motor driver. A second SG90 servo motor is installed on the front axle and controls the steering direction. The system is powered by two 3.7 V Li-ion batteries, while an XL4015 step down converter regulates the supplied voltage. An ESP32 microcontroller is responsible for sensor acquisition, motor control, and wireless communication.

A dedicated application was developed using the Godot Engine to support environmental mapping and vehicle navigation. The application exchanges data with the ESP32, processes the sensor measurements, constructs a map of the environment, calculates and validates collision free paths and smooths the resulting routes. Finally the processed route is converted into a sequence of distance and steering commands that the vehicle can execute.

Ευχαριστίες

Πριν προχωρήσω στην παρουσίαση της διπλωματικής μου εργασίας, θα ήθελα να ευχαριστήσω αρχικά τον επιβλέποντα καθηγητή μου, κ. Γιακουμή Άγγελο, για την καθοδήγηση και την υποστήριξη του. Ευχαριστώ επίσης το διδακτικό προσωπικό και τους συναδέλφους του τμήματος του Διεθνούς Πανεπιστημίου της Ελλάδος που όλα αυτά τα χρόνια με τις γνώσεις που μας μετέδωσαν συνέβαλαν στη διεκπεραίωση της διπλωματικής αυτής εργασίας. Τέλος, θα ήθελα να ευχαριστήσω ιδιαίτερα τον φίλο μου, Γεώργιο Κουνατίδη, την υποστήριξη του και την πολύτιμη βοήθεια του στην επίλυση των αποριών μου.

Περιεχόμενα

Πρόλογος.....	v
Περίληψη.....	vi
Abstract.....	vii
Ευχαριστίες.....	viii
Περιεχόμενα.....	ix
Κατάλογος Σχημάτων.....	xii
Συντομογραφίες.....	xiii
Κεφάλαιο 1ο: Εισαγωγή.....	1
1.1 Εισαγωγή.....	1
1.2 Το πρόβλημα της υλοποίησης.....	2
1.3 Κίνητρο.....	2
1.4 Στόχος.....	3
Κεφάλαιο 2ο: Θεωρητικό υπόβαθρο.....	4
2.1 Κινητή Ρομποτική.....	4
2.2 Οδομετρία.....	5
2.3 Αισθητήρας Time-Of-Flight.....	6
2.4 S.L.A.M.....	8
2.5 Χάρτες Πλέγματος.....	9
2.6 Αλγόριθμος εύρεσης διαδρομής.....	10
2.7 Αλγόριθμος παρεμβολής Catmull-Rom.....	11
Κεφάλαιο 3ο: Απαιτήσεις και σχεδιασμός συστήματος.....	12
3.1 Λειτουργικές απαιτήσεις.....	12
3.1.1 Ανίχνευση χώρου.....	12
3.1.2 Κίνηση και Οδομετρία.....	12
3.1.3 Ασύρματη επικοινωνία.....	13
3.1.4 Χαρτογράφηση.....	13
3.1.5 Σχεδιασμός διαδρομής.....	13
3.1.6 Εξομάλυνση διαδρομής.....	14
3.2 Μη λειτουργικές απαιτήσεις.....	15
3.2.1 Λειτουργία σε πραγματικό χρόνο.....	15
3.2.2 Αξιοπιστία.....	16
3.2.3 Υπολογιστική αποδοτικότητα.....	16

3.2.4 Ασφάλεια	16
3.2.5 Υλοποίηση χαμηλού κόστους	16
3.3 Αρχιτεκτονική του συστήματος	17
3.3.1 Ανίχνευση και ενεργοποίηση αισθητήρων	17
3.3.2 Επίπεδο ελέγχου μικροεπεξεργαστή	18
3.3.3 Επίπεδο επικοινωνίας	19
3.3.4 Η συνολική ροή των δεδομένων	19
3.4 Διαμερισμός εργασίας	20
3.4.1 Ευθύνες του ESP32	21
3.4.2 Ευθύνες της εφαρμογής του υπολογιστή	21
Κεφάλαιο 4ο: Αρχιτεκτονική υλικού συστήματος	22
4.1 Επισκόπηση συστήματος	22
4.2 Μικροεπεξεργαστής ESP32	24
4.2.1 Επικοινωνία με αισθητήρες	25
4.2.2 Ασύρματη Επικοινωνία	26
4.3 Motor Driver	26
4.4 Αισθητήρας TOF400C	27
4.4.1 Διεπαφή αισθητήρα	28
4.4.2 Τοποθέτηση του αισθητήρα	29
4.5 Servo και κατεύθυνση	29
4.6 Σύστημα τροφοδοσίας	30
Κεφάλαιο 5ο: Αρχιτεκτονική Λογισμικού	31
5.1 Ενσωματωμένο λογισμικό	31
5.1.1 Αρχικοποίηση βιβλιοθήκης και υλικού	32
5.1.2 Αρχικοποίηση υλικού	32
5.1.3 Συνάρτηση αρχικοποίησης	33
5.1.4 Κύρια συνάρτηση επανάληψης	34
5.2 Λογισμικό αισθητήρων	34
5.2.1 Αρχικοποίηση VL53L1X	34
5.2.2 Μετρήσεις VL53L1X	35
5.2.3 Κατεύθυνση του αισθητήρα	35
5.3 Έλεγχος ηλεκτρικών μοτέρ	36
5.4 Εφαρμογή με χρήση Godot	38
Κεφάλαιο 6ο: S.L.A.M. και κατασκευή χάρτη	40

6.1 Αρχιτεκτονική συστήματος SLAM	40
6.2 Αναπαράσταση εικονικού χάρτη	40
6.3 Σύστημα συντεταγμένων	41
6.4 Ray casting	42
6.5 Επεξεργασία σάρωσης	43
6.6 Χαρτογράφηση χάρτη σε chunks	43
6.7 Ποινές Εμποδίων	45
Κεφάλαιο 7ο: Σχεδιασμός διαδρομής και πλοήγησης	46
7.1 Σχεδιασμός διαδρομής	47
7.1.1 Ο σχεδιασμός “Go to”	47
7.1.2 Εύρεση διαδρομής που σχεδιάζεται από τον χρήστη	47
7.2 Αλγόριθμος εύρεσης διαδρομής	48
7.2.1 Αναζήτηση γειτονικών κόμβων	48
7.2.2 Λειτουργία αξιολόγησης γειτονικών κόμβων	49
7.2.3 Διαδικασία αναζήτησης	50
Υπάρχουν άλλοι τρεις πίνακες	50
7.3 Επικύρωση διαδρομής	50
7.3.1 Τοπική επισκευή A^*	51
7.3.2 Πύκνωση διαδρομής και προετοιμασία για εξομάλυνση	51
7.4 Εξομάλυνση διαδρομής με Catmull-Rom	52
7.4.1 Περιορισμός αρχικής κατεύθυνσης	53
7.4.2 Υλοποίηση της centripetal Catmull-Rom	54
7.4.3 Δειγματοληψία καμπύλης	56
7.4.4 Διόρθωση γωνίας στροφής	57
7.5 Πλοήγηση	58
Κεφάλαιο 8ο: Συμπεράσματα και προτάσεις βελτίωσης	59
8.1 Μετρήσεις και αποτελέσματα	59
8.1.1 Μετρήσεις του αισθητήρα TOF400C	59
8.1.2 Μετρήσεις οδομετρίας	60
8.2 Συμπεράσματα	61
8.3 Τρόποι βελτίωσης	61
ΒΙΒΛΙΟΓΡΑΦΙΑ	63
ΠΑΡΑΡΤΗΜΑ Α : ΚΩΔΙΚΑΣ ESP32	64

Κατάλογος Σχημάτων

Εικόνα 2.1 τροχός οχήματος.....	5
Εικόνα 2.2 τροχός οχήματος οδομετρίας.....	6
Εικόνα 2.3 Datasheet αισθητήρα VL53L1X.....	7
Εικόνα 2.4 Υπόδειγμα χάρτη πλέγματος.....	9
Εικόνα 3.1 Διάγραμμα ροής εξομάλυνσης διαδρόμης.....	15
Εικόνα 3.2 Σχεδιάγραμμα ηλεκτρονικού κυκλώματος.....	18
Εικόνα 3.3 Διάγραμμα επικοινωνίας Websocket.....	19
Εικόνα 3.4 Διάγραμμα ροής συνολικού συστήματος.....	20
Εικόνα 4.1 Το όχημα.....	22
Εικόνα 4.2 Διάγραμμα ροής του ηλεκτρονικού συστήματος.....	24
Εικόνα 4.3 Διάταξη πινίων ESP32 WROOM-32.....	25
Εικόνα 4.4 Motor Driver L298N.....	27
Εικόνα 4.5 Αισθητήρας TOF400C.....	28
Εικόνα 4.6 Αισθητήρας TOF400C στη βάση σέρβο.....	29
Εικόνα 4.7 Ρυθμιστής τάσης XL4015.....	30
Εικόνα 5.1 Διάγραμμα ροής ελεγκτή ESP32.....	31
Εικόνα 6.1 Πορεία οχήματος χωρίς πίνακα ποινών (πριν την εξομάλυνση πορείας).....	46
Εικόνα 6.2 Πορεία οχήματος με πίνακα ποινών (πριν την εξομάλυνση πορείας).....	46
Εικόνα 7.1 Σχεδίαση πορείας από τον χρήστη πριν την εξομάλυνση πορείας.....	53
Εικόνα 7.2 Σχεδίαση πορείας από τον χρήστη πριν την εξομάλυνση πορείας.....	53
Εικόνα 7.3 Οι διάφοροι τρόποι παραμετροποίησης Catmull-Rom spline.....	54
Εικόνα 7.4 Η πυραμιδική διατύπωση των Barry και Goldman.....	56

Κατάλογος Πινάκων

Πίνακας 8.1 Μετρήσεις αισθητήρα TOF400C.....	59
Πίνακας 8.2 Υποθετικές μετρήσεις οδομετρίας.....	60
Πίνακας 8.3 Πραγματικές μετρήσεις οδομετρίας.....	61

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΙΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
SLAM	Simultaneous Localization and Mapping
LiDAR	Light Detection and Ranging
TOF	Time Of Flight
SPAD	Single Photon Avalanche Diode
FoV	Field of View
A*	A star
SDA	serial data
SCL	serial clock

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η ανάπτυξη αυτόνομων ρομποτικών οχημάτων έχει αναδειχθεί ως μια από τις πιο σημαντικές περιοχές στην έρευνα της ρομποτικής, των ενσωματωμένων συστημάτων, της πληροφορικής και της τεχνητής νοημοσύνης. Τα αυτόνομα συστήματα ρομποτικής έχουν σχεδιαστεί για να μπορούν να αντιληφθούν το περιβάλλον γύρω τους, να εκτιμήσουν τη δική τους θέση, να λαμβάνουν αποφάσεις και να μπορούν να πλοηγούνται σε έναν χώρο με περιορισμένη ή και καθόλου άμεση ανθρώπινη παρέμβαση. Αυτές οι δυνατότητες είναι πολλές φορές απαραίτητες σε τομείς όπως βιομηχανίες αυτοματισμού, μεγάλες αποθήκες, έρευνα και διάσωση, αποφυγή πυρκαγιών, γεωργία, μεταφορές, ρομπότ για οικιακή χρήση και εξερεύνηση.

Μια από τις πιο βασικές προϋποθέσεις ενός αυτόνομα κινούμενου ρομπότ είναι η ικανότητα να μπορεί να αντιληφθεί το περιβάλλον του ενώ ταυτόχρονα να μπορέσει να αναγνωρίσει την θέση του μέσα σε αυτό. Αυτό το πρόβλημα συχνά αναφέρεται ως S.L.A.M. (Simultaneous Localization and Mapping - Ταυτόχρονος Εντοπισμός και Χαρτογράφηση). Το SLAM περιγράφει μια διαδικασία κατά την οποία ένα ρομπότ δημιουργεί μια αναπαράσταση ενός αρχικά άγνωστου περιβάλλοντος ενώ ταυτόχρονα επιδιώκει να εκτιμήσει την θέση του σε αυτήν την αναπαράσταση.

Ένα συνηθισμένο SLAM σύστημα συνήθως συνδυάζει πληροφορίες από πολλαπλούς αισθητήρες. Ανάλογα με την χρήση του εκάστοτε συστήματος μπορεί να διαθέτει αισθητήρες όπως κάμερες, σαρωτές λέιζερ, LiDAR αισθητήρες, υπερηχητικούς αισθητήρες, αδρανειακές μονάδες μέτρησης (IMU), κωδικοποιητές τροχών ή και time of flight (TOF) αισθητήρες απόστασης. Κάθε αισθητήρας έχει τα δικά του πλεονεκτήματα και τους δικούς του περιορισμούς. Η επιλογή του κατάλληλου αισθητήρα για κάθε υλοποίηση SLAM είναι μια πολύ σημαντική επιλογή για τον σχεδιασμό του συστήματος, ειδικότερα όταν έχουμε περιορισμό στην υπολογιστική ισχύ, στο μέγεθος του χώρου και στο κόστος.

Αυτή η εργασία εστιάζει στην ανάπτυξη μιας μικρής φορητής ρομποτικής πλατφόρμας με την ικανότητα να εκτελεί βασικές περιβαλλοντικές αντιλήψεις, να χαρτογραφεί και να μπορεί να εντοπίζει και να πλοηγείται στον χώρο που έχει χαρτογραφηθεί. Ο χρήστης έχει, επιπλέον, τη δυνατότητα να μπορεί να σχεδιάσει τη διαδρομή του οχήματος με την πλατφόρμα να σχεδιάζει ένα που το όχημα θα μπορέσει να εκτελέσει. Το σύστημα συνδυάζει έναν μικροεπεξεργαστή με αισθητήρες απόστασης TOF. Τα δεδομένα αυτά μεταδίδονται σε μια εφαρμογή υπολογιστή, όπου το περιβάλλον αναπαριστάται ως ένας δισδιάστατος χάρτης και χρησιμοποιείται για την πλοήγηση και τον σχεδιασμό της τελικής διαδρομής του οχήματος.

Η εργασία ερευνά και την αλληλεπίδραση μεταξύ υλικού και λογισμικού σε ένα ρομποτικό σύστημα. Ο μικροεπεξεργαστής είναι υπεύθυνος για τις πιο χαμηλού επιπέδου δράσεις όπως η μέτρηση αισθητήρων, ο έλεγχος των μοτέρ, η κίνηση των μηχανισμών σέρβο και η επικοινωνία με την εφαρμογή ενώ η εφαρμογή στην πλατφόρμα είναι υπεύθυνη για πιο υψηλού επιπέδου εργασίες όπως η χαρτογράφηση, την αναπαράσταση και η τελική σχεδίαση της διαδρομής.

Το αποτέλεσμα αυτού του συστήματος παρέχει μια πρακτική εκτέλεση κατά την οποία διαφορετικά εξαρτήματα απαραίτητα για ένα μικρό αυτόνομο όχημα μπορούν να λειτουργήσουν ως ένα ολοκληρωμένο σύστημα αντί για διάφορα ανεξάρτητα υλικά και λογισμικά συστήματα.

1.2 Το πρόβλημα της υλοποίησης

Για ένα όχημα να μπορέσει να κινηθεί αυτόνομα πρέπει να μπορέσει να απαντήσει δύο βασικές ερωτήσεις : Πού είμαι; και Πώς μοιάζει το περιβάλλον γύρω μου; Η δυσκολία αυτών των ερωτήσεων κλιμακώνεται όταν το όχημα λειτουργεί σε ένα μέρος που δεν έχει προϋπάρχοντα χάρτη.

Ένα σύστημα ενσωματωμένο μόνο με αισθητήρες απόστασης μπορεί να αναγνωρίσει εμπόδια γύρω του αλλά αυτές οι μεμονωμένες μετρήσεις δεν μπορούν να προσφέρουν αρκετές πληροφορίες για μια ολοκληρωμένη εικόνα του χώρου. Όταν το όχημα κινείται, οι μετρήσεις πρέπει να σχετίζονται με τη διάφορα στην θέση του οχήματος και την κατεύθυνση έτσι ώστε να μπορεί να εκτιμηθεί μια αξιοπρεπής αναπαράσταση του χώρου.

Η θέση την οποία εκτιμά το ρομπότ δεν είναι απόλυτα ακριβής. Η κίνηση των τροχών μπορεί να επηρεαστεί από το υλικό του εδάφους, τη διαφορά της διαμέτρου των τροχών ακόμα και από ηλεκτρικό θόρυβο στα μοτέρ. Η απλή ενσωμάτωση της απόστασης που διανύουν οι τροχοί με την πάροδο του χρόνου μπορεί να οδηγήσει σε αυξανόμενο σφάλμα θέσης. Ακόμα και οι αισθητήρες απόστασης έχουν το ίδιο πρόβλημα για αυξανόμενα σφάλματα καθώς επηρεάζονται από την απόσταση της μέτρησης, το υλικό του εμποδίου ακόμα και από τον φωτισμό του χώρου.

Ένα ακόμα σημαντικό πρόβλημα για την υλοποίηση είναι η περιορισμένη υπολογιστική ισχύς που είναι διαθέσιμη για ένα τροχοφόρο όχημα αυτού του μεγέθους. Τα πιο ακριβά ρομποτικά συστήματα χρησιμοποιούν αισθητήρες όπως LiDAR, κάμερες βάθους, ισχυρούς επεξεργαστές και υλικά αποκλειστικά για χρήση SLAM σε πραγματικό χρόνο.

Το πρόβλημα που προσπαθεί να απαντήσει η εργασία είναι η σχεδιάσει και η υλοποίηση ενός χαμηλού κόστους, μικρού μεγέθους οχήματος που να μπορεί να αναγνωρίσει το περιβάλλον του, να εκτιμήσει μια πορεία, να δημιουργήσει έναν δισδιάστατο χάρτη που να αναπαριστά τον χώρο του και με αυτόν να μπορέσει να κινηθεί αυτόνομα.

1.3 Κίνητρο

Η κύρια παρακίνηση της εργασίας είναι η διερεύνηση μιας πρακτικής εφαρμογής για ένα αυτόνομο όχημα χρησιμοποιώντας προσιτό και εύκολα διαθέσιμο υλικό.

Το SLAM είναι ένα σημαντικό πρόβλημα της ρομποτικής γιατί η αυτόνομη λειτουργία σε έναν άγνωστο χώρο χρήζει περισσότερο από ένα απλό αλγόριθμο εντοπισμού εμποδίων. Το όχημα πρέπει να μπορεί να συνδυάσει παρατηρήσεις που έκανε σε διαφορετικές θέσεις και να καθορίσει πως αυτές οι παρατηρήσεις επηρεάζουν η μια την άλλη. Η εφαρμογή αυτής της εργασίας παραχωρεί την ευκαιρία να μελετηθούν αρκετοί σημαντικοί τομείς της μηχανικής όπως ενσωματωμένα συστήματα, διεπαφή αισθητήρων, ασύρματη επικοινωνία, χαρτογράφηση και σχεδιασμός πορείας.

Ένα ακόμα κίνητρο για την ανάληψη της εργασίας είναι η διερεύνηση των περιορισμών αισθητήρων χαμηλού κόστους. Συστήματα υψηλής απόδοσης όπως το LiDAR μπορούν να παραχωρήσουν πυκνές και ακριβείς μετρήσεις σε σχετικά μεγάλες αποστάσεις αλλά το κόστος τους και το μέγεθος τους μπορεί να τα κάνει ακατάλληλα για μικρά οχήματα όπως αυτό. Οι αισθητήρες TOF παρέχουν μια διαφορετική προσέγγιση μετρώντας απευθείας την απόσταση μεταξύ του αισθητήρα και ενός εμποδίου. Αν και έχουν περιορισμένες ικανότητες μπορούν να παρέχουν χρήσιμες πληροφορίες για μικρού μεγέθους χώρους.

1.4 Στόχος

Ο κύριος στόχος της εργασίας είναι η σχεδίαση και υλοποίηση ενός μικρού αυτόνομου οχήματος με την ικανότητα να αντιληφθεί και να χαρτογραφήσει ένα άγνωστο περιβάλλον και με τη χρήση αυτού το εικονικού χάρτη να μπορέσει να δημιουργήσει μια πορεία.

Ο πρώτος στόχος ήταν η υλοποίηση ενός φυσικού οχήματος. Αυτό περιλαμβάνει τον motor driver, μηχανισμού κατεύθυνσης, τον μικροεπεξεργαστή, αισθητήρων απόστασης και το σύστημα παροχής ενέργειας.

Ο δεύτερος στόχος είναι να εφαρμοστεί μια αξιόπιστη λήψη μετρήσεων απόστασης από τους αισθητήρες TOF. Τα δεδομένα από τους αισθητήρες θα πρέπει να λαμβάνονται σε κατάλληλο ρυθμό και να επεξεργάζονται έτσι ώστε οι μη κατάλληλες τιμές να μην επηρεάζουν τον χάρτη που δημιουργείται.

Ο τρίτος στόχος είναι η σωστή εκτίμηση της κίνησης του οχήματος. Υλοποιώντας έναν σωστό τρόπο να ελέγχεται η κίνηση των τροχών μπορούμε να αναγνωρίσουμε την απόσταση που κινήθηκε το ρομπότ, ενώ ο προσανατολισμός και η κίνηση του ενσωματώνονται στο σύστημα συντεταγμένων της χαρτογράφησης.

Ο τέταρτος στόχος είναι η δημιουργία ασύρματης επικοινωνίας μεταξύ του ενσωματωμένου ελεγκτή και της εφαρμογής του υπολογιστή. Το σύστημα επικοινωνίας επιτρέπει τη μεταφορά μετρήσεων αισθητήρων και πληροφοριών κατάστασης του ρομπότ στην εφαρμογή, ενώ παράλληλα επιτρέπει τη μετάδοση εντολών κίνησης και πληροφοριών πλοήγησης πίσω στο ρομπότ.

Ο πέμπτος στόχος είναι η δημιουργία ενός συστήματος χαρτογράφησης. Οι μετρήσεις των αισθητήρων μετασχηματίζονται από το τοπικό σύστημα συντεταγμένων του ρομπότ σε παγκόσμιες συντεταγμένες και ενσωματώνονται στην αναπαράσταση του χώρου. Ο χάρτης διακρίνει μεταξύ αγνώστων περιοχών, περιοχών που θεωρεί ελεύθερες, περιοχές που είναι ελεύθερες αλλά πολύ κοντά σε εμπόδια.

Ο έκτος στόχος είναι η εφαρμογή ενός συστήματος σχεδιασμού διαδρομής μεταξύ δύο θέσεων λαμβάνοντας υπόψη τον χάρτη. Η διαδρομή θα πρέπει να λαμβάνει υπόψη τα εμπόδια και τους φυσικούς περιορισμούς του οχήματος καθώς το όχημα έχει τη δυνατότητα να εκτελεί γωνίες μέχρι 45 μοίρες. Η εφαρμογή πρέπει να δημιουργεί ένα μονοπάτι που να μην διασχίζει εμπόδια, να μην κάνει γωνίες μεγαλύτερες των 45 μοιρών και να αφήνει αρκετό χώρο ελεύθερο από το εμπόδιο ώστε το όχημα να μην συγκρουστεί.

Κεφάλαιο 2ο: Θεωρητικό υπόβαθρο

2.1 Κινητή Ρομποτική

Η κινητή ρομποτική είναι ένας τομέας της ρομποτικής που ασχολείται με τον σχεδιασμό και τη λειτουργία ρομπότ που έχουν την ικανότητα να κινούνται σε έναν χώρο. Σε αντίθεση με τα στάσιμα ρομποτικά συστήματα τα ρομπότ που μπορούν να κινούνται πρέπει να καθορίζουν συνεχώς την θέση τους, να αλληλεπιδρούν με το περιβάλλον και να μπορούν να ελέγξουν την κίνηση τους για να μπορέσουν να εκτελέσουν μια εργασία. Τα αυτόνομα κινούμενα ρομπότ συνδυάζουν συνεπώς διάφορους τομείς της μηχανικής όπως μηχανικά συστήματα, ηλεκτρονικά, αισθητήρες και του λογισμικού.

Ένα κινητό ρομπότ μπορεί γενικά να θεωρηθεί ως ένα σύστημα που αποτελείται από τρία θεμελιώδη στοιχεία: την αντίληψη, τη λήψη αποφάσεων και την ενεργοποίηση. Το σύστημα αντίληψης λαμβάνει πληροφορίες σχετικά με το ρομπότ και το περιβάλλον του μέσω αισθητήρων. Το σύστημα λήψης αποφάσεων επεξεργάζεται αυτές τις πληροφορίες και καθορίζει ποιες ενέργειες πρέπει να εκτελεστούν. Το σύστημα ενεργοποίησης μετατρέπει αυτές τις αποφάσεις σε φυσική κίνηση μέσω κινητήρων.[1]

Το περιβάλλον όπου ένα όχημα μπορεί να λειτουργήσει αναπαριστάται χρησιμοποιώντας διαφορετικά συστήματα συντεταγμένων. Η αναπαράσταση που χρησιμοποιείται είναι $[X, Y, \Theta]$ όπου X, Y είναι η θέση του οχήματος στον χάρτη και Θ είναι η κατεύθυνση του άξονα του.

Η στάση του οχήματος είναι ιδιαίτερα σημαντική σε ένα σύστημα SLAM γιατί οι μετρήσεις που λαμβάνονται είναι σε σχέση με την θέση του ρομπότ. Για να ενσωματωθούν αυτές οι μετρήσεις σε έναν χάρτη οι μετρήσεις πρέπει πρώτα να μετασχηματιστούν από το τοπικό σύστημα συντεταγμένων στο παγκόσμιο σύστημα που χρησιμοποιεί η εφαρμογή.[3]

Τα κινούμενα ρομπότ μπορούν να χρησιμοποιήσουν μια ποικιλία μηχανισμών κίνησης. Μερικά παραδείγματα περιλαμβάνουν ρομπότ με σύστημα διεύθυνσης ολίσθησης, οχήματα με σύστημα διεύθυνσης Ackermann[2], ρομπότ ολικής κίνησης και ρομπότ με πόδια. Η επιλογή του μηχανισμού κίνησης καθορίζει το μαθηματικό μοντέλο που μπορεί να χρησιμοποιηθεί για την περιγραφή της κίνησης του ρομπότ.

Σε μια διαμόρφωση διαφορικής κίνησης, δύο ανεξάρτητα κινούμενοι τροχοί είναι τοποθετημένοι σε αντίθετες πλευρές του ρομπότ. Αλλάζοντας τη σχετική ταχύτητα των δύο τροχών, το ρομπότ μπορεί να κινηθεί προς τα εμπρός, να περιστραφεί ή να ακολουθήσει μια καμπύλη τροχιά. Αυτή η διαμόρφωση χρησιμοποιείται ευρέως λόγω της μηχανικής απλότητας και του σχετικά απλού μαθηματικού μοντέλου.[1]

Η κίνηση επομένως επηρεάζεται από την ταχύτητα των τροχών, την απόσταση μεταξύ των τροχών, τις διαστάσεις τους και την αλληλεπίδραση μεταξύ τροχών και εδάφους.

Επειδή κάθε αισθητήρας έχει περιορισμούς, τα αυτόνομα ρομπότ λειτουργούν με αβεβαιότητα. Οι μετρήσεις μπορεί να περιέχουν θόρυβο, οι αισθητήρες μπορεί να μην ανιχνεύουν αντικείμενα και η εκτιμώμενη κίνηση του ρομπότ μπορεί να διαφέρει από την πραγματική του κίνηση. Για αυτό, τα σύγχρονα ρομποτικά συστήματα χρησιμοποιούν συχνά μαθηματικά μοντέλα που λαμβάνουν ρητά υπόψη την αβεβαιότητα.

2.2 Οδομετρία

Η οδομετρία είναι η διαδικασία εκτίμησης της αλλαγής θέσης και του προσανατολισμού ενός ρομπότ, παρακολουθώντας την κίνηση του με την πάροδο του χρόνου. Στα ρομπότ με τροχούς η οδομετρία γίνεται συνήθως από κωδικοποιητές τροχών ή από άλλους αισθητήρες που μετράνε την περιστροφή των τροχών.[6]

Η βασική αρχή της είναι να καθοριστεί η απόσταση που διένυσε κάθε τροχός και να χρησιμοποιηθεί η διαφορά για να υπολογίσει την νέα θέση του οχήματος. Για ένα απλοποιημένο όχημα με διαφορετική κίνηση δύο τροχών, έστω (D_L) και (D_R) που αντιπροσωπεύουν τις αποστάσεις που διάνυσαν ο αριστερός και δεξιός τροχός αντίστοιχα. Η αλλαγή προσανατολισμού μπορεί να υπολογιστεί ως: $\Delta\theta = \frac{D_R - D_L}{L}$ όπου το L είναι η απόσταση από το κέντρο των δύο τροχών.

Η απόσταση που διανύει το όχημα μπορεί να υπολογιστεί προσεγγίστηκα: $\Delta s = \frac{D_R + D_L}{2}$

Αμα η προηγούμενη κατεύθυνση του οχήματος ήταν θ τότε η νέα θέση :

$$X_{t+1} = X_t + \Delta s \cdot \cos(\theta + \Delta\theta/2)$$

$$Y_{t+1} = Y_t + \Delta s \cdot \sin(\theta + \Delta\theta/2)$$

$$\theta_{t+1} = \theta_t + \Delta\theta$$

Για έναν τροχό με διάμετρο D η περιφέρεια του είναι : $C = \pi \cdot D$ και όταν ο τροχός ολοκληρώνει N περιστροφές η απόσταση που διάνυσε είναι : $d = N \cdot \pi \cdot D$



Εικόνα 2.1 τροχός οχήματος

Σε ένα πρακτικό σύστημα η απόσταση που υπολογίζεται από την περιστροφή του τροχού δεν είναι πάντα ίση με την πραγματική απόσταση που διανύει το όχημα. Πολλαπλοί παράγοντες μπορούν να προκαλέσουν σφάλματα όπως μικρές διαφορές στους τροχούς οι ανακριβείς μετρήσεις της απόστασης των τροχών, η ολίσθηση των τροχών, οι ανώμαλες επιφάνειες, οι

διάφορες στην ταχύτητα των μοτέρ και εσφαλμένη βαθμονόμηση[9]. Το μεγαλύτερο μειονέκτημα της οδομετρίας είναι ότι το σφάλμα είναι αθροιστικό. Ένα μικρό σφάλμα σε κάθε εκτίμηση κίνησης μπορεί να συσσωρευτεί όσο το όχημα ταξιδεύει περισσότερο. Η οδομετρία μπορεί να παρέχει μια χρήσιμη βραχυπρόθεσμη εκτίμηση της κίνησης ωστόσο για μακροπρόθεσμο εντοπισμό μπορεί να οδηγήσει σε σημαντική μετατόπιση θέσης.[6]

Για την ανάπτυξη αυτής της ρομποτικής πλατφόρμας η κίνηση του τροχού μετριέται χρησιμοποιώντας έναν TOF400C τοποθετημένο έτσι ώστε να εστιάζει στον τροχό. Στον τροχό έχουν τοποθετηθεί δύο μικρά χαρτιά σε αντίθετες πλευρές του έτσι ώστε να υπάρχουν δύο καταστάσεις: όταν ο αισθητήρας ανιχνεύει το χαρτί (απόσταση μικρότερη των 60mm) και όταν ο τροχός δεν ανιχνεύει το χαρτί (απόσταση μεγαλύτερη των 60mm). Άρα όταν έχει δύο ολοκληρωμένες αλλαγές στην κατάσταση το όχημα γνωρίζει πως ολοκλήρωσε μια ολόκληρη περιστροφή του τροχού που γνωρίζουμε πως είναι [6].



Εικόνα 2.2 τροχός οχήματος οδομετρίας

2.3 Αισθητήρας Time-Of-Flight

Η μέτρηση της απόστασης είναι η πιο σημαντική απαίτηση για ένα αυτόνομο όχημα. Ένα όχημα πρέπει να μπορεί να προσδιορίζει την απόσταση μεταξύ του εαυτού του και των αντικείμενων γύρω του έτσι ώστε να μπορεί να ανιχνεύσει εμπόδια και να κατασκευάσει έναν χάρτη.

Οι TOF αισθητήρες μπορούν να αναγνωρίσουν την απόσταση μετρώντας τον χρόνο που χρειάζεται για να ταξιδεύσει η εκπεμπόμενη ηλεκτρομαγνητική ακτινοβολία σε έναν στόχο και να επιστρέψει στον αισθητήρα. Για μια ιδανική μέτρηση χρόνου πτήσης η απόσταση μπορεί να εκφραστεί ως: $d = \frac{c \cdot t}{2}$

Όπου d είναι η απόσταση του στόχου, c η ταχύτητα διάδοσης του σήματος και t ο μετρούμενος χρόνος διαδρομής με επιστροφή. Η διαίρεση με το δύο πρέπει γιατί ο χρόνος αντιστοιχεί τόσο στην εξερχόμενη όσο και στην επιστροφή.

Τα TOF που χρησιμοποιούνται στην εργασία χρησιμοποιούν το VL53L1X αισθητήρα που κατασκευάζεται από την STMicroelectronics. Η συσκευή είναι αισθητήρας μέτρησης TOF που βασίζεται στην τεχνολογία FlightSense της ST. Σύμφωνα με τον κατασκευαστή ο αισθητήρας υποστηρίζει αποστάσεις μετρήσεως έως και 4 μέτρα υπό κατάλληλες συνθήκες.



VL53L1X
 Datasheet

A new generation, long distance ranging Time-of-Flight sensor based on ST's FlightSense technology



Product status link
[VL53L1X](#)

Features

Fully integrated miniature module

- 940 nm invisible laser emitter (Class 1)
- SPAD (single photon avalanche diode) receiving array with integrated lens
- Low power microcontroller running advanced digital firmware
- 4.9 x 2.5 x 1.56 mm

Fast, accurate distance ranging

- Fast and accurate long distance ranging
 - Up to 400 cm distance measurement
 - Up to 50 Hz ranging frequency
- Pin-to-pin compatible with the VL53L0X FlightSense ranging sensor
- 27° typical full field of view (FoV)
- Programmable region of interest (ROI) size on the receiving array allows reduction of the sensor FoV
- A programmable ROI position on the receiving array, provides multizone operation control from the host

Eyesafety

- Class 1 laser device compliant with latest standard IEC 60825-1:2014 - 3rd edition

Easy integration

- Software driver and code example for turnkey ranging
- Single reflowable component
- Single power supply 2v8
- Can be hidden behind many cover glass materials
- I2C interface (up to 400 kHz)
- Shutdown and interrupt pins

Application

- User detection (autonomous low-power mode) to power on/off and lock/unlock devices like personal computers/laptops and the Internet of Things (IoT)
- Service robots and vacuum cleaners (long distance and fast obstacle detection)
- Drones (landing assistance, hovering, ceiling detection)
- Smart shelves and vending machines (goods inventory monitoring)
- Sanitary (robust user detection whatever the target reflectance)
- Smart building and smart lighting (people detection, gesture control)
- 1 D gesture recognition
- Laser assisted autofocus (AF) enhances the camera AF system's speed and robustness, especially in difficult scenes such as low light and low contrast, and provides video focus tracking assistance

DS12385 - Rev 8 - August 2024
For further information contact your local STMicroelectronics sales office.
www.st.com

Εικόνα 2.3 Datasheet αισθητήρα VL53L1X

Το πλεονέκτημα ενός TOF σε αντίθεση με έναν αισθητήρα σόναρ είναι ότι ο TOF προσπαθεί να προσδιορίσει μια ακριβές απόσταση αντί απλώς να καθορίζει πως υπάρχει ένα εμπόδιο μπροστά. Ο VL53L1X ενσωματώνει επίσης ένα 940nm υπέρυθρο λέιζερ και μια διάταξη SPAD έχει επίσης οπτικό φίλτράρισμα και επεξεργασία που σκοπεύουν στην βελτίωση της απόδοσης μέτρησης εύρους ζώνης υπό διαφορετικές συνθήκες φωτισμού περιβάλλοντος. ένα ακόμα σχετικό χαρακτηριστικό είναι το οπτικό πεδίο FoV του αισθητήρα όπου καθορίζει την γωνιακή περιοχή από την οποία λαμβάνει χρήσιμες πληροφορίες. Ο αισθητήρας επιτρέπει την αύξηση, με αποτέλεσμα να διευκολύνει την εύρεση εμποδίων αλλά να οδήγησε σε μετρήσεις που δεν

αντιστοιχούν στην συγκεκριμένη κατεύθυνση , ή στην μείωση με αποτέλεσμα πιο σωστών μετρήσεων αλλά πιο δύσκολη εύρεση εμποδίων.

Άμα ένας TOF μετράει μια απόσταση r σε μια γωνία α σε σχέση με το όχημα οι τοπικές συντεταγμένες του μπορούν να αναπαρασταθούν ως: $X_s = r * \cos(\alpha)$ και $Y_s = r * \sin(\alpha)$ αυτές οι συντεταγμένες μπορούνε έπειτα να μετατραπούν στο καθολικό σύστημα συντεταγμένων με την εκτιμώμενη θέση του οχήματος.

Σε αυτήν την εργασία ο ένας αισθητήρας TOF χρησιμοποιείται ως κύρια πηγή για την περιβαλλοντική απόσταση. Ο δεύτερος χρησιμοποιείται για την οδομετρία των τροχών.

2.4 S.L.A.M.

Η ταυτόχρονη τοποθέτηση και χαρτογράφηση (SLAM) είναι ο τρόπος όπου ένα ρομπότ μπορεί να υποθέσει την θέση του ενώ ταυτόχρονα να κατασκευάσει έναν χάρτη ενός αγνώστου περιβάλλοντος.[3] Το πρόβλημα του SLAM είναι ένα βασικό για την αυτόνομη ρομποτική γιατί ένα ρομπότ σε άγνωστο χώρο δεν μπορεί να βασιστεί σε έναν προηγούμενος κατασκευασμένο χάρτη για τον εντοπισμό του. Ταυτόχρονα η δημιουργία του χάρτη θέλει δεδομένα που πρέπει να ληφθούν κατά την πορεία του οχήματος.

Επομένως, τα δύο προβλήματα εντοπισμός θέσης του ρομπότ και χαρτογράφηση εξαρτώνται το ένα από το άλλο. Εάν το ρομπότ ξέρει την θέση του μπορεί να τοποθέτηση σωστά τις παρατηρήσεις του στον χάρτη. Ενώ άμα έχει έναν σωστό χάρτη οι παρατηρήσεις του μπορούν να το βοηθήσουν για μια σωστή εκτίμηση της θέσης του.

Οι Durrant-Whyte και Bailey περιγράφουν το SLAM ως ένα πρόβλημα που περιλαμβάνει την εκτίμηση τόσο της τροχιάς του ρομπότ όσο και της δομής του περιβάλλοντος[3]. Η μαθηματική αναπαράσταση του είναι $X_{0,m}$ όπου X_0 είναι η αρχική θέση του ρομπότ από την αρχική στιγμή ενώ το m αντιπροσωπεύει τον χάρτη.

Το σύστημα λαμβάνει δεδομένα έλεγχου και παρατηρήσεις από τους αισθητήρες. Στόχος του είναι η εκτίμηση της πιο πιθανής τροχιάς, η μαθηματική σχέση του πιθανοτικού SLAM είναι

$P(X_{0,m} | Z_1, u_1)$ όπου :

- $P(..|...)$ η πιθανότητα να συμβούν τα αριστερά (η πορεία του) στοιχεία, δεδομένου ότι γνωρίζουμε τα δεξιά στοιχεία.
- X_0 αρχική θέση του οχήματος για $t=0$
- m ο χάρτης
- Z_1 η πρώτη μέτρηση του αισθητήρα TOF
- U_1 η πρώτη εντολή κίνησης ή μέτρησης της οδομετρίας.

[3]

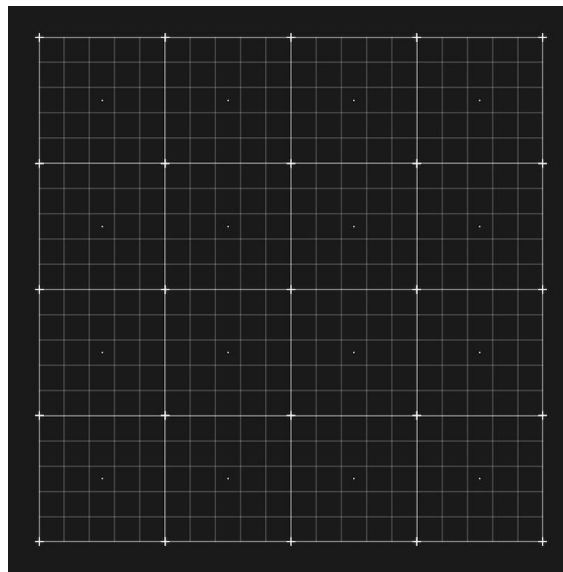
Για αυτήν την εργασία η υλοποίηση του SLAM χρησιμοποιεί μια δισδιάστατη αναπαράσταση χάρτη πλέγματος αντί για μια αναπαράσταση που βασίζεται σε ορόσημα. Η κίνηση του ρομπότ παρέχει μια εκτίμηση της θέσης του με την βοήθεια της οδομετρίας, ενώ οι μετρήσεις του TOF παρέχουν

παρατηρήσεις εμποδίων. Έπειτα αυτές οι παρατηρήσεις προβάλλονται στον χάρτη σύμφωνα με την εκτιμώμενη θέση του οχήματος.

Αυτή η προσέγγιση είναι κατάλληλη για την κλίμακα της εργασίας και για του στόχους που έθεσα για το σύστημα επειδή παρέχει μια άμεση αναπαράσταση των εμποδίων και του ελεύθερου χώρου του περιβάλλοντος.

2.5 Χάρτες Πλέγματος

Ένας χάρτης πλέγματος αναπαριστά έναν χώρο ως μια συλλογή από διακριτά κελιά. Κάθε κελί αντιστοιχεί σε μια συγκεκριμένη περιοχή του φυσικού χώρου και αποθηκεύει πληροφορίες σχετικά με το άμα είναι κατειλημμένο, ελεύθερο, άγνωστο ή την τιμή της ποινής του άμα είναι κοντά σε ένα εμπόδιο.



Εικόνα 2.4 υπόδειγμα χάρτη πλέγματος

Ο χάρτης πλέγματος εισήχθηκε ως μέθοδος για την αναπαράσταση πληροφοριές σε χώρο που λαμβάνεται από αισθητήρες εμβέλειας. Οι Moravec και Elfes έδειξαν πως πολλαπλές μετρήσεις σόναρ μπορούν να προβληθούν σε μια αναπαράσταση ενός περιβάλλοντος επιτρέποντας την ταξινόμηση περιοχών σε κατειλημμένο, ελεύθερο, άγνωστο .[4]

Κάθε κελί μπορεί να αναπαρασταθεί ως : $M(i,j) \in \{0,1,2\}$ όπου:

- 0 αναπαριστά άγνωστο κελί
- 1 αναπαριστά ελεύθερο κελί
- 2 αναπαριστά κατειλημμένο κελί

Ενώ στην προσέγγιση αυτού του συστήματος υπάρχει και ένας παράλληλος χάρτης όπου εκεί καταγράφεται η τιμή της ποινής του κελιού άμα είναι κοντά σε εμπόδιο $P(i,j) \in \{0-100\}$ όπου 0 δεν έχει και 100 πρόκειται για κελί ακριβώς δίπλα σε εμπόδιο.

Η αναπαράσταση αυτή είναι ιδιαίτερα χρήσιμη για αισθητήρες απόστασης γιατί έστω ένας αισθητήρας βρίσκεται σε θέση s και μετράει ένα εμπόδιο σε θέση r . μεταξύ της θέσης s μέχρι την θέση r μπορεί να θεωρηθεί ελεύθερο καθώς η μέτρηση δεν βρήκε κάποιο εμπόδιο στην θέση εκείνη.

Καθώς το όχημα κινείται στο περιβάλλον του οι διαφορετικές μετρήσεις από τις διαφορετικές θέσεις του ενσωματώνονται στο ίδιο πλέγμα. Οι επαναλαμβανόμενες παρατηρήσεις μπορούν ενδεχόμενος να ενισχύσουν τις τιμές των κελιών. Αυτή η προσέγγιση έχει πολλά πλεονεκτήματα για μια μικρή ρομποτική πλατφόρμα όπως αυτήν. Η αναπαράσταση είναι σχετικά απλή και μπορεί να οπτικοποιηθεί εύκολα ενώ μπορεί να χρησιμοποιηθεί εύκολα από πολλούς αλγόριθμους σχεδιασμού διαδρομής.

Η ανάλυση του πλέγματος πρέπει να επιλέγεται σωστά. Στην πλατφόρμα αυτή ο χάρτης χωρίζεται σε μεγάλα κομμάτια (chunks) όπου κάθε ένα από αυτά αποτελείται από 32 κελιά. Ο λόγος που χωρίζεται σε chunks είναι γιατί είναι πιο εύκολο να φορτώνουμε και να διαγράφουμε τα συγκεκριμένα κομμάτια του χάρτη που χρειαζόμαστε ή δεν χρειαζόμαστε και να διατηρούμε την μνήμη καθαρή. Κάθε κελί στην προσέγγιση αυτή αντιπροσωπεύει 1 εκ. έτσι μπορούμε να έχουμε μια καλή ανάλυση του χώρου χωρίς να χάνουμε λεπτομέρειες και χωρίς να επηρεάζουν τόσο τα σφάλματα του TOF.

Ο χάρτης στην συνέχεια μπορεί να χρησιμοποιηθεί από το σύστημα πλοήγησης για να προσδιορίσει ποιες περιοχές είναι ασφαλείς να διασχίσει και να δημιουργήσει μια διαδρομή.

2.6 Αλγόριθμος εύρεσης διαδρομής

Ο σχεδιασμός διαδρομής είναι η διαδικασία προσδιορισμού μιας ακολουθίας θέσεων ή κινήσεων που επιτρέπει σε ένα ρομπότ να ταξιδέψει από μια θέση εκκίνησης σε έναν επιθυμητό προορισμό, ικανοποιώντας παράλληλα περιορισμούς όπως η αποφυγή εμποδίων και οι περιορισμοί κίνησης του ρομπότ.

Ο σωστός σχεδιασμός διαδρομής είναι ένα από τα πιο σημαντικά κομμάτια της αυτόνομης πλοήγησης επειδή δεν αρκεί να ξέρει μόνο που είναι τα εμπόδια. Το ρομπότ πρέπει επίσης να καθορίσει πώς μπορεί να κινηθεί μέσα στον διαθέσιμο ελεύθερο χώρο.

Μια διαδρομή μπορεί να εκφραστεί ως ένα σύνολο σημείων $P = \{p_0, p_1, \dots, p_n\}$ όπου P το σύνολο σημείων και p_i το κάθε μεμονωμένο σημείο.

Οι αλγόριθμοι σχεδιασμού διαδρομής χωρίζονται σε διάφορες κατηγορίες. Αλγόριθμους αναζήτησης γραφημάτων, μεθόδους που βασίζονται σε δειγματοληψία, ευρετικές μεθόδους. Για χάρτες πλέγματος, όπως σε αυτήν την πλατφόρμα, οι αλγόριθμοι αναζήτησης γραφημάτων είναι οι πιο βολικοί γιατί τα κελιά μπορούν να ερμηνευτούν σαν γράφημα. Κάθε ελεύθερο μπορεί να θεωρηθεί ως Ένας κόμβος και οι συνδέσεις μεταξύ γειτονικών κελιών μπορούν να αναπαραστήσουν πιθανές κινήσεις.

Ένας από του πιο συχνούς αλγορίθμους αναζήτησης που χρησιμοποιείται και σε αυτήν την εργασία είναι ο A^* . Ο A^* συνδυάζει το κόστος όπου έχει ήδη προκύψει για να φτάσει σε έναν συγκεκριμένο κόμβο με το εκτιμώμενο κόστος να φτάσει στον στόχο του [7]. Ο τύπος αξιολόγησης του είναι

$f(n) = g(n) + h(n)$ όπου:

- $f(n)$ το συνολικό εκτιμώμενο κόστος μιας λύσης μέσω του κόμβου n
- $g(n)$ το κόστος για να φτάσει τον κόμβο n
- $h(n)$ το κόστος από τον κόμβο n μέχρι το τελικό αποτέλεσμα

[7]

Για την εύρεση της απόστασης καθώς το σύστημα μας επιτρέπει την διαγώνια κάθετη και οριζόντια κίνηση χρησιμοποιούμε την ευκλείδεια απόσταση. $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ όπου (x_2, y_2) είναι οι συντεταγμένες του σημείου $n+1$ και (x_1, y_1) οι συντεταγμένες του σημείου n .

Ο αλγόριθμος A^* δεν δίνει πάντα την πιο γρήγορη δυνατή διαδρομή αλλά δίνει πολύ γρήγορα μια σχετικά γρήγορη διαδρομή ιδιαίτερα σημαντικό όταν το όχημα αναγνωρίζει εμπόδια καθώς κινείται σε μια πορεία του και θέλει γρήγορα να τα αποφύγει. Ωστόσο ο αλγόριθμος A^* μπορεί να δημιουργήσει μια διαδρομή που ένα όχημα (όπως αυτό) δεν μπορεί να ακολουθήσει γιατί η πορεία που σχεδίασε έχει μια στροφή μεγαλύτερη των δυνατοτήτων του οχήματος (το όχημα έχει περιορισμό στροφής του άξονα 45 μοίρες) ή η πορεία ξεκινάει από μια πλευρά αντίρροπη της κατεύθυνσης του οχήματος.

Για την αποφυγή αυτή Ένας αλγόριθμος εξομάλυνσης της διαδρομής μπορεί να χρησιμοποιηθεί για τον μετασχηματισμό της πορείας σε μια πιο συνεχή τροχιάς. Μέθοδοι παρεμβολής όπως καμπύλες Bezier ή Catmull-Rom splines μπορούν να χρησιμοποιηθούν. Στην προκειμένη περίπτωση χρησιμοποιείται ο αλγόριθμος παρεμβολής Catmull-Rom splines.[5]

2.7 Αλγόριθμος παρεμβολής Catmull-Rom

Ένα spline είναι μια τμηματικά οριζόμενη μαθηματική καμπύλη που μπορεί χρησιμοποιηθεί για την παρεμβολή ή την προσέγγιση μιας ακολουθίας σημείων έλεγχου (control points) διατηρώντας ένα νεπιθυμητο βαθμό ομαλότητας.[5]

Μια spline Catmull-Rom είναι μια παρεμβαλλόμενη spline, που σημαίνει ότι η καμπύλη που δημιουργείται διέρχεται από τα σημεία έλεγχου. Αυτό το χαρακτηριστικό την καθιστά ιδιαίτερα χρήσιμη όταν μια διαδρομή που δημιουργείται από έναν αλγόριθμο σχεδιασμού διαδρομής πρέπει να εξομαλυνθεί χωρίς να αλλάξουν σημαντικά οι θέσεις των αρχικών σημείων αναφοράς

Για τέσσερα συνεχόμενα σημεία P_0, P_1, P_2, P_3 ο αλγόριθμος carmull-rom θα δημιουργήσει μια καμπύλη μεταξύ των σημείων P_1, P_2 :

$$P(t) = \frac{1}{2} [2P_1 + (-P_0+P_2)t + (2P_0 - 5P_1 + 4P_2 - P_3)t^2 + (-P_0 + 3P_1 - 3P_2 + P_3)t^3]$$

Οπου:

- $P(t)$ αντιπροσωπεύει ένα σημείο στην παρεμβαλλόμενη καμπύλη
- P_1 και P_2 είναι τα σημεία διαδρομής όταν $t=0$ η εξίσωση δίνει ακριβώς P_1 ενώ όταν $t=1$ δίνει P_2
- P_0 και P_3 είναι τα σημεία έλεγχου που χρησιμοποιούνται για να καθορίσουν την ταχύτητα και την καμπυλότητα (εφαπτόμενη) της εισόδου P_1 και εξόδου P_2 . Η καμπύλη δεν περνάει από αυτά
- t είναι η παράμετρος προόδου δείχνει το ποσοστό της διαδρομής μεταξύ P_1 και P_2

Μια σημαντική ιδιότητα της καμπύλης Catmull-Rom είναι ότι παρέχει ομαλές μεταβάσεις μεταξύ των τμημάτων της διαδρομής. Αυτό μπορεί να μειώσει τον αριθμό των απότομων αλλαγών κατεύθυνσης που υπάρχουν σε μια διαδρομή που βασίζεται στην διαδρομή του A^* . Στο σύστημα η

παρεμβολή Catmull-Rom χρησιμοποιείται ως βήμα επεξεργασίας μετά την εύρεση διαδρομής. Τα διακριτά σημεία αναφοράς που δημιουργούνται από τον αλγόριθμο A* χρησιμοποιούνται ως σημεία ελέγχου για το spline.

Κεφάλαιο 3ο: Απαιτήσεις και σχεδιασμός συστήματος

3.1 Λειτουργικές απαιτήσεις

Για το προτεινόμενο όχημα με χρήση SLAM, οι λειτουργικές απαιτήσεις προέκυψαν από τους στόχους της δημιουργίας μιας πλατφόρμας ικανής να ανιχνεύει το περιβάλλον του, να εκτιμά την κίνησή του, να κατασκευάζει έναν χάρτη, να επιτρέπει στην χρήση να σχεδιάζει την δικιά του πορεία και να πλοηγείται στο χαρτογραφημένο περιβάλλον.

Το σύστημα μπορεί να χωριστεί σε μερικές βασικές λειτουργίες: ανίχνευση χώρου, κίνηση και οδομετρία, ασύρματη επικοινωνία, χαρτογράφηση, σχεδιασμός διαδρομής και έλεγχος οχήματος.

3.1.1 Ανίχνευση χώρου

Το όχημα θα πρέπει να είναι ικανό να μετράει αποστάσεις από τα αντικείμενα στο χώρο του. Ο αισθητήρας TOF χρησιμοποιείται ως κυρία πηγή πληροφορίας απόστασης. Το σύστημα των αισθητήρων πρέπει να :

- Αρχικοποιεί τους αισθητήρες TOF και να ελέγχει άμα λειτουργούν σωστά.
- Να λαμβάνει μετρήσεις σε έναν προκαθορισμένο χρόνο (33ms).
- Να μπορεί να αναγνωρίζει λανθάνουσες μετρήσεις.
- Να συσχετίζει τις μετρήσεις με την αντίστοιχη κατεύθυνση.
- Να μεταδίδει έγκυρες μετρήσεις στην εφαρμογή.

3.1.2 Κίνηση και Οδομετρία

Το όχημα θα πρέπει να μπορεί να ελέγχει την κίνηση του στο χώρο γύρω και να γνωρίσει την απόσταση που διάνυσε. Ένας αισθητήρας TOF έχει τοποθετηθεί εστιασμένος σε έναν από τους τροχούς με δύο χαρτιά τοποθετημένα σε κάθε μεριά του τροχού έτσι ώστε να μπορεί να αναγνωρίζει τότε έγινε μια ολοκληρωτική περιστροφή του τροχού. Το σύστημα της οδομετρίας θα πρέπει να:

- Αναγνωρίζει τότε κινείται ο τροχός

- Να μετράει τις κινήσεις του τροχού
- Να μετατρέπει την απόσταση που πρέπει να διαμήνυση σε πόσες περιστροφές του τροχού πρέπει να γίνουν
- Να αναγνωρίζει πότε γίνεται αλλαγή στην θέση του οχήματος
- Μα συνυπολογίζει την αλλαγή την κατεύθυνσης του οχήματος στην τελική πορεία
- Να επιστρέφει πληροφορίες για το που βρίσκεται στην πορεία του στην εφαρμογή.

3.1.3 Ασύρματη επικοινωνία

Ο μικροεπεξεργαστής και η εφαρμογή θα πρέπει να επικοινωνούν ανέπαφα έχοντας μια αναδρομική σχέση. Η επικοινωνία θα πρέπει να μην σταματάει της βασικές εργασίες του ελεγκτή ούτε της εφαρμογής. Το όχημα θα πρέπει να στέλνει δεδομένα από τους αισθητήρες για την δημιουργία ενός χάρτη και τις οδομετρικές πληροφορίες του ενώ η εφαρμογή θα πρέπει να στέλνει την πορεία που πρέπει να διανύσει το όχημα.

3.1.4 Χαρτογράφηση

Η εφαρμογή θα μπορεί να δημιουργεί και να διατηρεί μια αναπαράσταση του χώρου χρησιμοποιώντας τις λαμβανόμενες μετρήσεις των αισθητήρων. Το σύστημα της χαρτογραφήσεις θα πρέπει

- Να λαμβάνει μετρήσεις αποστάσεων
- Να προσδιορίζει την θέση και τον προσανατολισμό του οχήματος
- Να μετατρέπει τις συντεταγμένες που λαμβάνει από τους αισθητήρες στις συντεταγμένες που χρησιμοποιεί ο χάρτης
- Να προσδιορίζει τα αντίστοιχα κελιά του χάρτη
- Να προσδιορίζει τον ελεύθερο χώρο του χάρτη
- Να προσδιορίζει τα εμπόδια

3.1.5 Σχεδιασμός διαδρομής

Το σύστημα πρέπει να είναι ικανό να δημιουργεί μια διαδρομή μεταξύ την θέση του οχήματος και ενός προορισμού.

- Να προσδιορίζει περιοχές που μπορεί να διασχίσει στον χάρτη
- Να αποφύγει τα εμπόδια
- Να λαμβάνει υπόψη της φυσικές διαστάσεις του οχήματος
- Να υπολογίζει μια διαδρομή ανάμεσα σε δύο σημεία

- Να επικυρώνει την τελική διαδρομή

Ο αλγόριθμος A* μπορεί να χρησιμοποιηθεί για να προσδιορίσει μια κατάλληλη διαδρομή μέσω ελεύθερου χώρου.

3.1.6 Εξομάλυνση διαδρομής

Η διαδρομή που σχεδιάζετε από το αλγόριθμο A* ενδέχεται να περιέχει απότομες αλλαγές κατεύθυνσης. Επομένως πρέπει να παρέχει ένα στάδιο εξομάλυνσης διαδρομής. Τα σημεία έλεγχου υποβάλλονται σε επεξεργασία χρησιμοποιώντας έναν αλγόριθμο παρεμβολής spline Catmull-Rom δημιουργώντας μια συνεχή καμπύλη.[5] Με αυτόν το τρόπο μπορούμε:

- Να γνωρίζουμε την απόσταση μεταξύ κάθε σημείο έλεγχου
- Να γνωρίζουμε την γωνία μεταξύ κάθε σημείο έλεγχου
- Να μπορούμε να επεξεργαστούμε την γωνία μεταξύ δύο σημείων έλεγχου για να υποστηρίξει τον περιορισμό που έχει το όχημα (μπορεί να κάνει γωνία μέχρι 45 μοίρες)
- Να σχεδιάσουμε εξολοκλήρου την διαδρομή του οχήματος ζωγραφίζοντας την πορεία
- Να ακολουθήσει την πορεία του αλγορίθμου εύρεσης διαδρομής όταν η εκκίνηση δεν έχει ίδια κατευθυντικότητα με το όχημα

Η διαδρομή που δημιουργείται πρέπει, ωστόσο, να ελέγχεται ξανά αν τηρεί τις προϋποθέσεις του αλγορίθμου εύρεσης διαδρομής. Η πλήρης διαδικασία πλοήγησης μπορεί να αναπαρασταθεί ως εξής:



Εικόνα 3.1 Διάγραμμα ροής εξομάλυνσης διαδρομής

3.2 Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις περιγράφουν χαρακτηριστικά που καθορίζουν τον τρόπο που θα πρέπει να λειτουργεί αλλά όχι συγκεκριμένες λειτουργίες που πρέπει να εκτελεί. Αυτές οι απαιτήσεις είναι ιδιαίτερα σημαντικές, επειδή το σύστημα μπορεί τεχνικά να εκτελεί τις απαιτούμενες λειτουργίες παρά όλα αυτά να εξακολουθεί να είναι ακατάλληλο για λειτουργία σε πραγματικό χρόνο.

3.2.1 Λειτουργία σε πραγματικό χρόνο

Το συνολικό σύστημα θα πρέπει να επεξεργάζεται μετρήσεις αισθητήρων και πληροφορίες κατάστασης του οχήματος με επαρκώς χαμηλή καθυστέρηση για να υποστηρίξει την πλοήγηση. Οι μετρήσεις απόστασης πρέπει να λαμβάνονται με κατάλληλη συχνότητα, ενώ η επικοινωνία μεταξύ του οχήματος και του υπολογιστή θα πρέπει να παρουσιάζει ελάχιστη καθυστέρηση.

Το σύστημα δεν απαιτεί μετρήσεις εξαιρετικά υψηλής συχνότητας, είναι ελεγχόμενο στα 33ms, καθώς το όχημα κινείται με μέγιστη ταχύτητα 33,4 εκα./δευτερ. Ωστόσο οι υπερβολικές καθυστερήσεις μεταξύ ανίχνευσης, χαρτογράφησης, εύρεσής διαδρομής και επικοινωνίας μπορούν να οδηγήσουν το όχημα να ανταποκρίνεται σε παλιές πληροφορίες.

3.2.2 Αξιοπιστία

Οι μετρήσεις των αισθητήρων πρέπει να ελέγχονται πριν χρησιμοποιηθούν από το σύστημα χαρτογράφησης. Οι μη έγκυρες μετρήσεις δεν θα πρέπει να ερμηνεύονται αυτόματα ως εμπόδια, καθώς αυτό θα μπορούσε να οδηγήσει σε εσφαλμένες τροποποιήσεις στον χάρτη. Θα πρέπει να εντοπίζονται βλάβες επικοινωνίας, ώστε το ρομπότ μπορεί να έχει μια ασφαλή κατάσταση αντί να συνεχίζει να εκτελεί παρωχημένες εντολές.

3.2.3 Υπολογιστική αποδοτικότητα

Το σύστημα έχει υπολογιστικούς περιορισμούς λόγω του επιλεγμένου υλικού για αυτό τον λόγο πρέπει οι δυνατότητες κάθε συστήματος να υλοποιηθούν επαρκώς.

Ο ενσωματωμένος μικροεπεξεργαστής είναι υπεύθυνος για χρονικά ευαίσθητες λειτουργίες και μέτρηση αισθητήρων όποτε δεν θα πρέπει να επιβαρύνεται με υπολογιστικά απαιτητικές λειτουργίες χαρτογράφησης ή σχεδιασμού διαδρομής όταν αυτές μπορούν να εκτελεστούν από τον υπολογιστή. Όχι μόνο κάποιες από αυτές τις δυνατότητες θα είναι υπερβολικά χρονοβόρες για αυτό το σύστημα αλλά και τέλειος αδύνατες να υλοποιηθούν.

Ταυτόχρονα οι αλγόριθμοι χαρτογράφησης και σχεδιασμού διαδρομής πρέπει να έχουν σχεδιαστεί για να επεξεργάζονται την επιλεγμένη ανάλυση του χάρτη εντός αποδεκτού χρονικού διαστήματος.

3.2.4 Ασφάλεια

Το όχημα θα πρέπει να μπορεί να σταματήσει όταν κάποιο σημαντικό σφάλμα έχει εμφανιστεί. Κάποια πιθανά σφάλματα είναι:

- Αποτυχία επικοινωνίας
- Εσφαλμένες εντολές κινήσεως
- Αποτυχία των αισθητήρων
- Άμα υπάρχει ένα εμπόδιο πολύ κοντά στο όχημα

Το σύστημα θα πρέπει να έχει μια καθορισμένη ασφαλή κατάσταση στην οποία οι κινητήρες σταματούν όταν η κίνηση δεν μπορεί να ελεγχθεί αξιόπιστα.

3.2.5 Υλοποίηση χαμηλού κόστους

Ένας επιπλέον στόχος σχεδιασμού είναι η διατήρηση ενός σχετικά χαμηλού κόστους υλικού. Κατά συνέπεια, τα άμεσα διαθέσιμα εξαρτήματα προτιμούνται αντί για τα πιο ακριβά εξειδικευμένα ρομποτικά υλικά. Εξαρτήματα όπως το ESP32, TOF400C, SG90 πληρούν τις απαιτήσεις που έχουν τεθεί με σχετικά χαμηλό ολικό κόστος.

Η χρήση αισθητήρων TOF χαμηλού κόστους και ενός ενσωματωμένου μικροεπεξεργαστή (ESP32 Wroom 32) παρέχουν έναν πρακτικό συμβιβασμό μεταξύ της ικανότητας ανίχνευσης, των υπολογιστικών απαιτήσεων και του συνολικού κόστους του συστήματος.

3.3 Αρχιτεκτονική του συστήματος

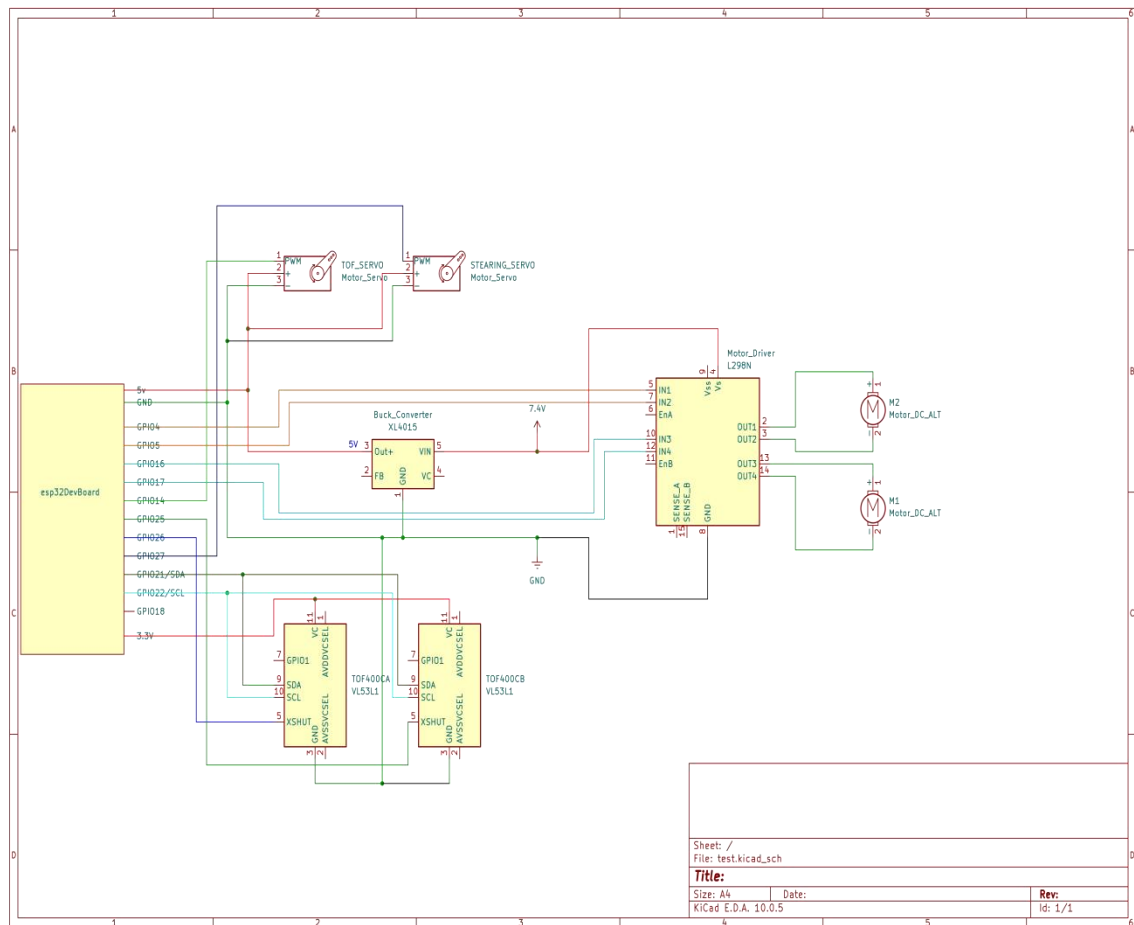
Το ολοκληρωμένο σύστημα ακολουθεί μια αρχιτεκτονική στην οποία ο έλεγχος χαμηλού επιπέδου εκτελείται από έναν μικροεπεξεργαστή, ενώ η επεξεργασία υψηλότερου επιπέδου εκτελείται από μια εφαρμογή στον υπολογιστή. Η αρχιτεκτονική μπορεί να χωριστεί σε τέσσερα κομμάτια:

- Ανίχνευση και ενεργοποίηση αισθητήρων
- Επίπεδο ελέγχου μικροεπεξεργαστή
- Επίπεδο επικοινωνίας
- χαρτογράφηση και πλοήγηση

3.3.1 Ανίχνευση και ενεργοποίηση αισθητήρων

Το χαμηλότερο επίπεδο του συστήματος αποτελείται από τα φυσικά υλικά του οχήματος. Τα εξαρτήματα ανίχνευσης περιλαμβάνουν τους αισθητήρες απόστασης TOF όπου ο ένας ασχολείται με την χαρτογράφηση του χώρου και ο δεύτερος με την κίνηση του τροχού για οδομετρία.

Τα εξαρτήματα ενεργοποίησης περιλαμβάνουν τους κινητήρες και τα δύο σέρβο ένα υπεύθυνο για την κατεύθυνση του οχήματος και ένα βοηθάει το TOF για την σωστή δειγματοληψία. Ένας οδηγός κινητήρα τοποθετείται μεταξύ του ενσωματωμένου επεξεργαστή και των κινητήρων, επειδή ο μικροελεγκτής δεν μπορεί να παρέχει άμεσα το ρεύμα που απαιτείται από τους κινητήρες. Ενώ ένας μετασχηματιστής ρεύματος buck converter είναι και αυτός τοποθετημένος στην πηγή ενέργειας, δύο μπαταρίες Li-ion 3,7 V σε σειρά, ώστε να παρέχει τη σωστή τάση των 5 V στον ελεγκτή και στα δύο σέρβο.



Εικόνα 3.2 Σχεδιάγραμμα ηλεκτρονικού κυκλώματος

3.3.2 Επίπεδο ελέγχου μικροεπεξεργαστή

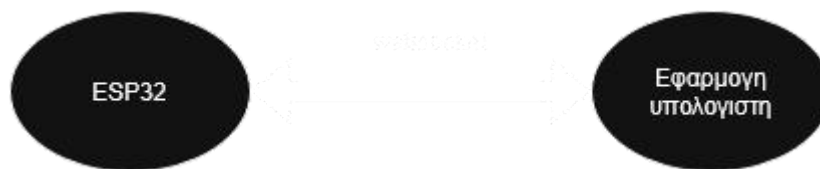
Ο ενσωματωμένος ελεγκτής λειτουργεί ως διεπαφή μεταξύ του φυσικού υλικού και της εφαρμογής του υπολογιστή. Οι βασικές του υποχρεώσεις είναι:

- Αρχικοποίηση αισθητήρων
- Λήψη μετρήσεων απόστασης
- Ανίχνευση κίνησης τροχού
- Έλεγχος κινητήρα
- Έλεγχος κατεύθυνσης
- Επεξεργασία βασικών πληροφοριών αισθητήρα
- Διατήρηση πληροφοριών κατάστασης ρομπότ
- Ασύρματη επικοινωνία

Ο ESP32 δεν χρειάζεται να διατηρεί τον χάρτη. Στέλνει τις απαιτούμενες πληροφορίες αισθητήρα και κατάστασης στον υπολογιστή, όπου μπορεί να γίνει μια πιο υπολογιστικά απαιτητική επεξεργασία για χαρτογράφηση και πλοήγηση.

3.3.3 Επίπεδο επικοινωνίας

Το επίπεδο επικοινωνίας συνδέει το όχημα με την εφαρμογή του υπολογιστή. Η ασύρματη σύνδεση παρέχει αμφίδρομη επικοινωνία μέσω μιας σύνδεσης websocket. Ο μικροελεγκτής δρα ως ο server στην επικοινωνία και η εφαρμογή ως client. Τα δεδομένα αισθητήρων αποστέλλονται από το ρομπότ στον υπολογιστή, ενώ οι εντολές κίνησης και πλοήγησης αποστέλλονται προς την αντίθετη κατεύθυνση.



Εικόνα 3.3 Διάγραμμα επικοινωνίας Websocket

Τα μεταδιδόμενα δεδομένα είναι σχεδιασμένα έτσι ώστε να μπορούν να στέλνουν ταυτόχρονα πολλαπλές τιμές. Οι μετρήσεις αισθητήρων, οι πληροφορίες θέσης, ο προσανατολισμός και τα δεδομένα που σχετίζονται με τον έλεγχο μπορούν επομένως να μεταφερθούν ως ένα ενιαίο λογικό μήνυμα. Ένα τυπικό μήνυμα έχει την μορφή JSON:

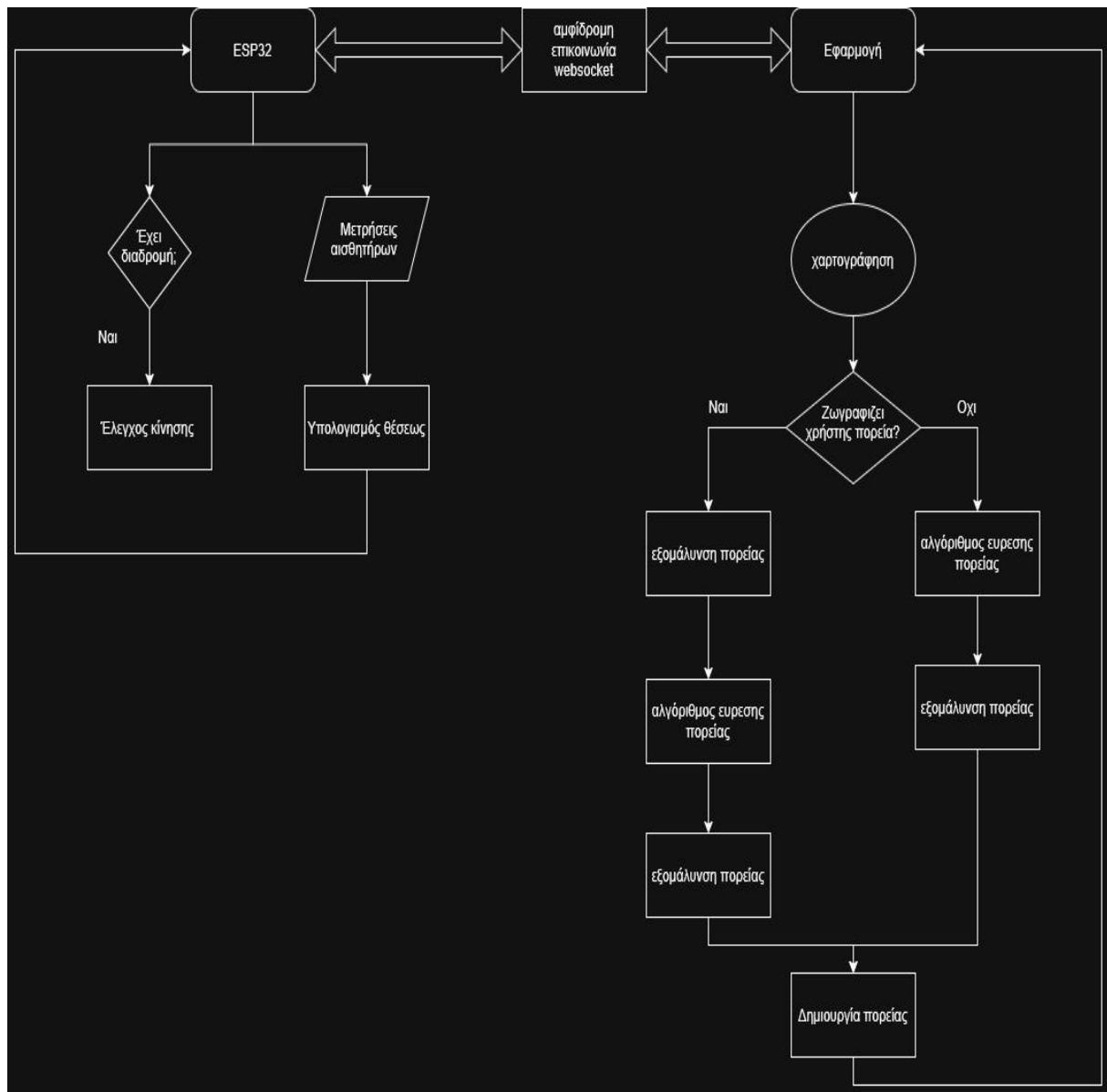
```
String json = "{";
json += "\"type\": \"sensor_data\", ";
json += "\"angleA\": " + String(totalAngleA) + ", ";
json += "\"distanceA\": " + String(lastDistA);
json += "}";
```

3.3.4 Η συνολική ροή των δεδομένων

Αρχικά το όχημα παρατηρεί το περιβάλλον του μέσω των αισθητήρων απόστασης. Οι μετρήσεις μεταδίδονται στον υπολογιστή. Ο υπολογιστής χρησιμοποιεί αυτές τις πληροφορίες για να εκτιμήσει πού βρίσκεται το ρομπότ και ενημερώνει τον χάρτη για ποια κελιά είναι ελεύθερα ή κατειλημμένα καθώς και τις τιμές των ποινών για τα κελιά κοντά στα εμπόδια για να γνωρίζει ο αλγόριθμος εύρεσης διαδρομής ποια κελιά να αποφύγει. Μόλις επιλεγεί ένας προορισμός το σύστημα πλοήγησης αναζητά στον χάρτη μια κατάλληλη διαδρομή. Η διαδρομή εξομαλύνεται και επικυρώνεται πριν από τη δημιουργία της πραγματικής πορείας. Αν ο χρήστης επιλέξει να ζωγραφίσει την πορεία του οχήματος

πρέπει πρώτα η διαδρομή να περάσει από τον αλγόριθμο εξομαλύνσεις και μετά από αλγόριθμο εύρεσης πορείας και ξανά από αλγόριθμο εξομαλύνσεις. Αυτές οι εντολές μεταδίδονται στον ενσωματωμένο ελεγκτή ο οποίος ελέγχει τους κινητήρες και τον μηχανισμό διεύθυνσης.

Καθώς το όχημα κινείται οι αισθητήρες του παράγουν νέες παρατηρήσεις. Αυτές οι παρατηρήσεις μεταδίδονται πίσω στον υπολογιστή και η διαδικασία επαναλαμβάνεται άμα χρειαστεί.



Εικόνα 3.4 Διάγραμμα ροής συνολικού συστήματος

3.4 Διαμερισμός εργασίας

Μια σημαντική απόφαση στην ανάπτυξη του συστήματος ήταν ποιες λειτουργίες θα έπρεπε να εκτελούνται στον ενσωματωμένο μικροεπεξεργαστή και ποιες από την εφαρμογή του υπολογιστή.

Αυτή η διαίρεση είναι απαραίτητη επειδή οι δύο πλατφόρμες έχουν διαφορετικά πλεονεκτήματα και μειονεκτήματα. Ο ενσωματωμένος ελεγκτής παρέχει άμεση πρόσβαση στους αισθητήρες και τους ενεργοποιητές του ρομπότ και έχει σχεδιαστεί για αλληλεπίδραση υλικού σε πραγματικό χρόνο. Ο υπολογιστής παρέχει σημαντικά μεγαλύτερους υπολογιστικούς πόρους και μνήμη και είναι πολύ καλύτερος στην χαρτογράφηση, εικονογράφηση και σχεδιασμό της διαδρομής.

3.4.1 Ευθύνες του ESP32

Ο ενσωματωμένος ελεγκτής είναι υπεύθυνος κυρίως για λειτουργίες χαμηλού επιπέδου όπως:

- Επικοινωνία με αισθητήρες TOF
- Διαμόρφωση αισθητήρων
- Μετρήσεις αποστάσεων
- Ανίχνευση κίνησης τροχών
- Έλεγχος του μοτέρ
- Έλεγχος της κατεύθυνσης
- Λήψη εντολών κίνησης
- Μετάδοση δεδομένων
- Διατήρηση της κατάστασης του οχήματος

Αυτές οι λειτουργίες απαιτούν άμεση πρόσβαση στα GPIO του ESP32 και στις διεπαφές επικοινωνίας του μικροελεγκτή. Η λήψη δεδομένων από αισθητήρες είναι κατάλληλη για τον ενσωματωμένο ελεγκτή επειδή οι αισθητήρες είναι συνδεδεμένοι με αυτόν. Η επεξεργασία των ακατέργαστων μετρήσεων κατευθείαν στον ελεγκτή μειώνει επίσης την απαιτούμενη επικοινωνία χαμηλού επιπέδου μεταξύ του ρομπότ και του υπολογιστή.

3.4.2 Ευθύνες της εφαρμογής του υπολογιστή

Ο υπολογιστής είναι υπεύθυνος για λειτουργίες που απαιτούν περισσότερους υπολογιστικούς πόρους ή χρειάζονται μια γραφική διεπαφή όπως :

- Λήψη πληροφοριών αισθητήρα
- Διατήρηση του χάρτη
- Μετατροπή μετρήσεων αισθητήρα σε συντεταγμένες
- Οπτικοποίηση του οχήματος και του περιβάλλοντος
- Ανίχνευση εμποδίων
- Δημιουργία διαδρομής
- Εφαρμογή εξομάλυνσης διαδρομής

- Επικύρωση διαδρομής
- Εμφάνιση πληροφοριών πλοήγησης
- Επιτρέπει στον χρήστη να διαλέγει την πορεία του οχήματος

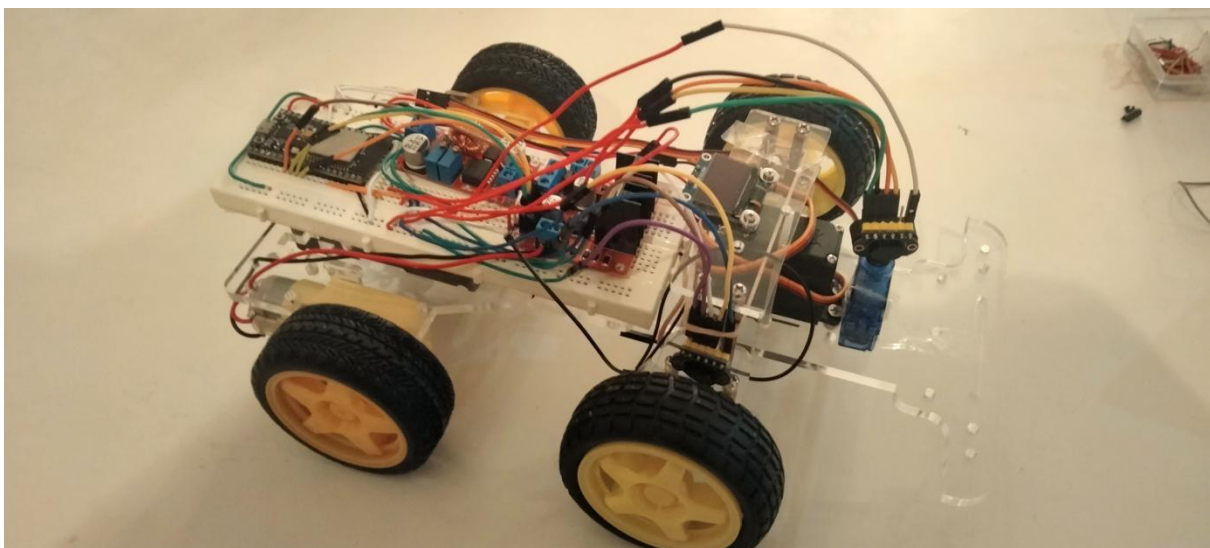
Επομένως η εφαρμογή του υπολογιστή λειτουργεί ως η κύρια πλατφόρμα επεξεργασίας και εικονοποίησης για το SLAM και το σύστημα πλοήγησης.

Κεφάλαιο 4ο: Αρχιτεκτονική υλικού συστήματος

4.1 Επισκόπηση συστήματος

Η πλατφόρμα του οχήματος αποτελεί τη μηχανική και ηλεκτρική βάση του αυτόνομου οχήματος. Η πλατφόρμα είναι υπεύθυνη για τη μεταφορά των εξαρτημάτων ανίχνευσης, επεξεργασίας, ενεργοποίησης και ισχύος, παρέχοντας παράλληλα την κίνηση που απαιτείται για την πλοήγηση και τη χαρτογράφηση. Δεδομένου ότι ο στόχος του έργου είναι η εφαρμογή ενός δισδιάστατου συστήματος ταυτόχρονου εντοπισμού και χαρτογράφησης SLAM, το όχημα σχεδιάστηκε ως μια κινητή πλατφόρμα ικανή για ελεγχόμενη επίπεδη κίνηση και μέτρηση απόστασης.

Το όχημα χρησιμοποιεί μια διαμόρφωση κινητής πλαστικής πλατφόρμας με τέσσερις τροχούς. Η μηχανική του δομή αποτελείται από ένα πλαίσιο, κινητήριους τροχούς, έναν μηχανισμό διεύθυνσης, δύο κινητήρες, σημεία στήριξης αισθητήρων και ηλεκτρονικά εξαρτήματα. Συγκεκριμένα, η απόσταση μεταξύ των τροχών, η διάμετρος του τροχού, η θέση των αισθητήρων σε σχέση με το σημείο αναφοράς του ρομπότ και ο προσανατολισμός των αισθητήρων πρέπει να είναι γνωστά, προκειμένου να μετατραπούν σωστά οι μετρήσεις των αισθητήρων στο παγκόσμιο σύστημα συντεταγμένων του χάρτη.



Εικόνα 4.1 Το όχημα

Το όχημα κατασκευάστηκε σε μια πλαστική πλατφόρμα και έχει πλάτος περίπου 20 εκ. και μήκος 35 εκ. ενώ οι κινητήριои τροχοί έχουν διάμετρο περίπου 68 mm. Η διάμετρος του τροχού είναι ιδιαίτερα σημαντική για την οδομετρία γιατί η διανυθείσα απόσταση μπορεί να εκτιμηθεί από τη γωνιακή μετατόπιση του τροχού $d = \frac{N}{N_{rev}} \pi * D$

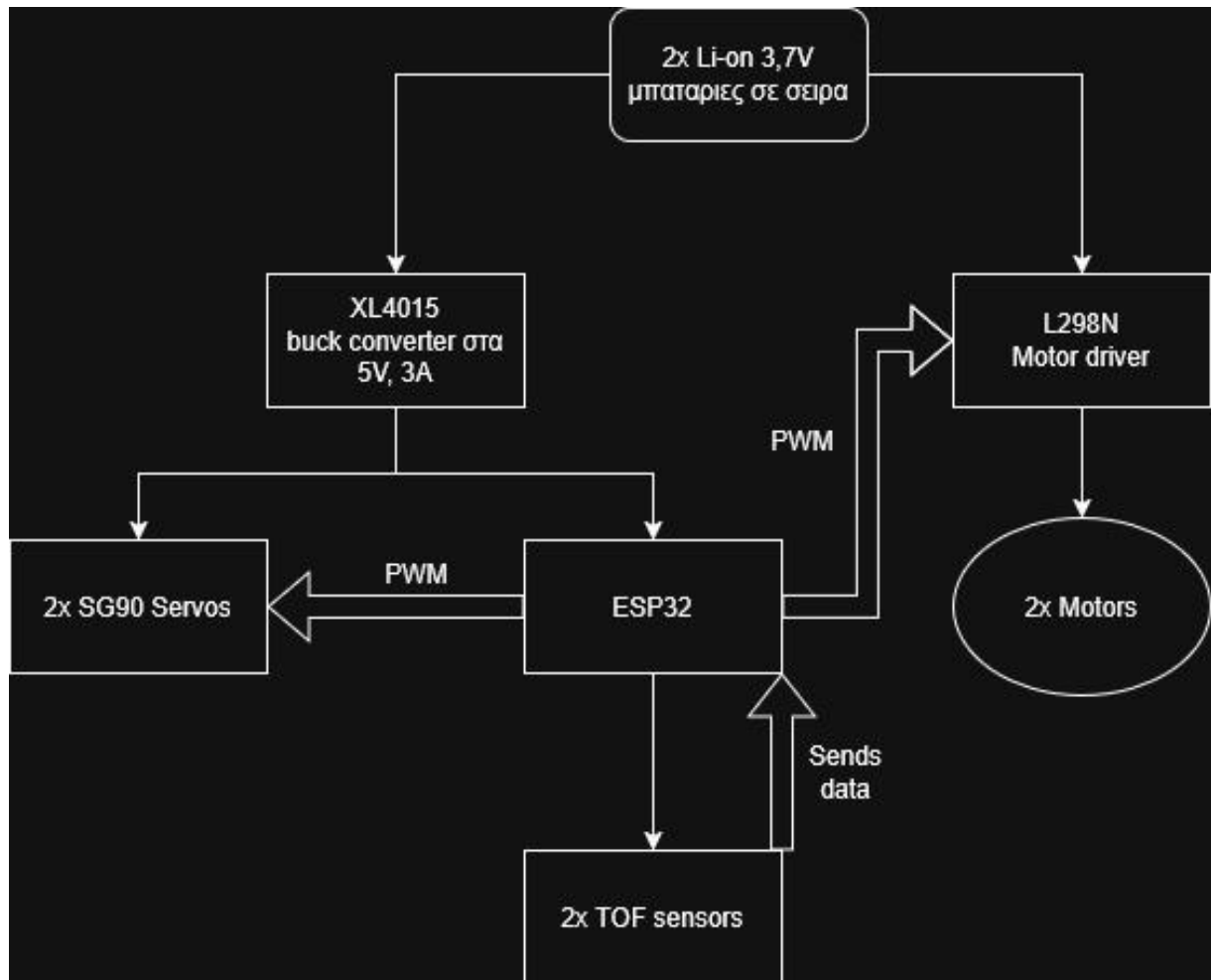
όπου d είναι η απόσταση κίνησης, N είναι ο μετρούμενος αριθμός παλμών που ανιχνεύει το TOF, N_{rev} είναι ο αριθμός παλμών για μια πλήρη περιστροφή του τροχού και D είναι η διάμετρος τροχού. Για έναν τροχό με διάμετρο 68mm η περιφέρεια είναι $C = \pi * (0,068) = 0.2136 \mu$.

Επομένως μια πλήρης περιστροφή του τροχού αντιστοιχεί σε περίπου 21,36 εκ. της διαδρομής υποθέτοντας πως δεν υπάρχει ολίσθηση του τροχού.

Ο μηχανικός σχεδιασμός έχει επίσης επίδραση στην απόδοση του SLAM. Ιδανικά η θέση των αισθητήρων απόστασης θα πρέπει να είναι σε σταθερή σχέση με το σώμα του ρομπότ. Οποιαδήποτε κίνηση ή δόνηση των αισθητήρων μπορεί να προκαλέσει σφάλματα επειδή ο αλγόριθμος SLAM υποθέτει ότι οι μετρήσεις των αισθητήρων προέρχονται από γνωστή σταθερή θέση και κατεύθυνση. Η δομή στήριξης του αισθητήρα πρέπει να παραμένει επαρκώς άκαμπτη κατά τη λειτουργία.

Η πλατφόρμα μπορεί να χωριστεί σε τέσσερα κύρια υποσυστήματα:

- Υπολογιστικό σύστημα του μικροελεγκτή ESP32
- Σύστημα ανίχνευσης που αποτελείται κυρίως από δύο αισθητήρες απόστασης VL53L1X έναν για απόσταση και έναν για τη θέσης τροχού
- Σύστημα ενεργοποιητών που αποτελείται από δύο κινητήρες συνεχούς ρεύματος, έναν οδηγό κινητήρα L298N και δύο σερβοκινητήρες SG90
- Σύστημα ισχύος που αποτελείται από την μπαταρία και το κύκλωμα ρύθμισης τάσης buck converter



Εικόνα 4.2 Διάγραμμα ροής του ηλεκτρονικού συστήματος

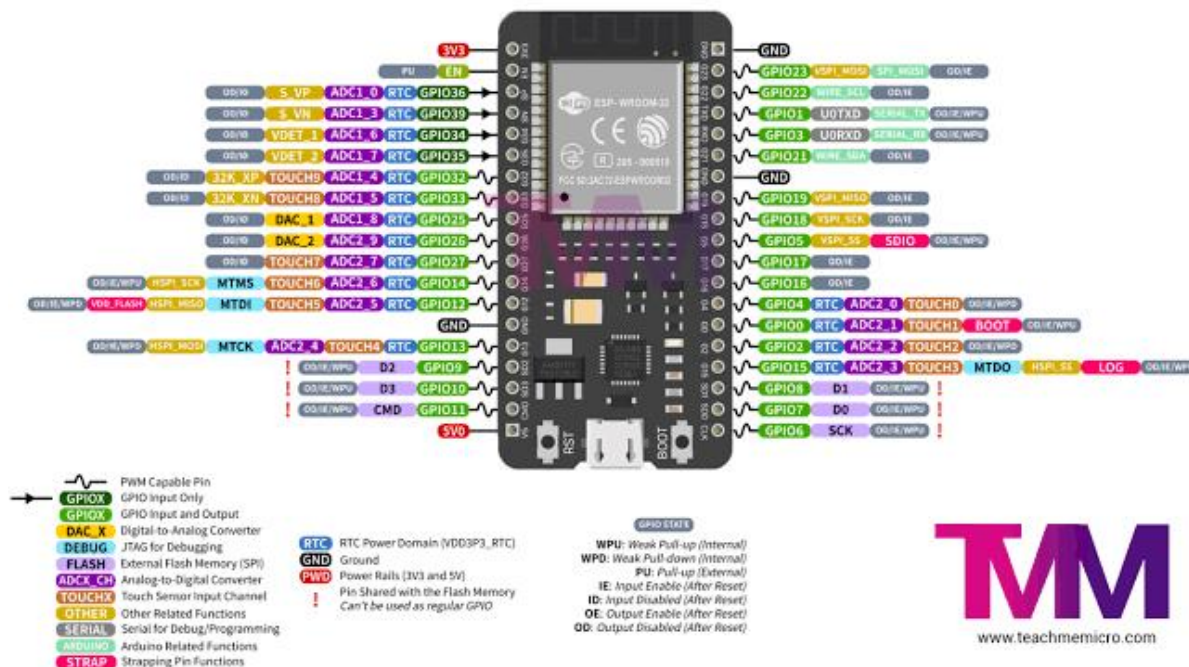
Το ESP32 διαχειρίζεται αισθητήρες, επικοινωνία, λογική ελέγχου κινητήρα και επεξεργασία αισθητήρων, ενώ ο οδηγός του κινητήρα χειρίζεται την ηλεκτρική ισχύ που απαιτείται από τους κινητήρες. Αυτός ο διαχωρισμός είναι σημαντικός επειδή οι κινητήρες συνεχούς ρεύματος μπορούν να παράγουν ηλεκτρικό θόρυβο και σημαντικές διακυμάνσεις ρεύματος, ιδιαίτερα κατά την επιτάχυνση, το φρενάρισμα και τις αλλαγές κατεύθυνσης.

4.2 Μικροεπεξεργαστής ESP32

Το ESP32 χρησιμεύει ως ελεγκτής του οχήματος. Επιλέχθηκε επειδή συνδυάζει επαρκή υπολογιστική απόδοση με σχετικά μεγάλο αριθμό ψηφιακών επαφών και ενσωματωμένη ασύρματη επικοινωνία. Η οικογένεια ESP32 περιλαμβάνει έναν επεξεργαστή 32-bit που βασίζεται σε Xtensa, διεπαφές GPIO, περιφερειακά PWM, λειτουργικότητα I2C, SPI, UART, ADC και ενσωματωμένες δυνατότητες WiFi και Bluetooth 2,4 GHz.[8]

ESP32 PINOUT DIAGRAM

ESP32-WROOM-32



Εικόνα 4.3 Διάταξη πινών ESP32 WROOM-32

Αυτά τα χαρακτηριστικά καθιστούν το ESP32 κατάλληλο για μια πλατφόρμα στην οποία ο μικροελεγκτής πρέπει ταυτόχρονα να επικοινωνεί με αισθητήρες, να ελέγχει κινητήρες και σύστημα διεύθυνσης, να επεξεργάζεται πληροφορίες από τους τροχούς και να ανταλλάσσει δεδομένα με μια εξωτερική εφαρμογή υπολογιστή. Το ESP32 παρέχει την κεντρική διεπαφή μεταξύ του υλικού και του λογισμικού του οχήματος.

Το ESP32 περιέχει δύο διεπαφές I2C και ειδικά περιφερειακά PWM, τα οποία καθιστούν δυνατή τη σύνδεση των αισθητήρων απόστασης και των ενεργοποιητών ελέγχου χωρίς να απαιτείται πρόσθετος μικροελεγκτής. Το φύλλο δεδομένων ESP32 καθορίζει επίσης την ειδική λειτουργικότητα PWM ελέγχου κινητήρα η οποία είναι χρήσιμη για τον έλεγχο του κινητήρα.

4.2.1 Επικοινωνία με αισθητήρες

Οι αισθητήρες VL53L1X επικοινωνούν με το ESP32 χρησιμοποιώντας το πρωτόκολλο I2C. Το I2C απαιτεί δύο κύριες γραμμές σήματος:

- SDA serial data GPIO21
- SCL serial clock GPIO22

Πολλαπλές συσκευές μπορούν να μοιράζονται τον ίδιο δίαυλο I2C με την προϋπόθεση ότι έχουν διαφορετικές διευθύνσεις. Αυτό είναι ιδιαίτερα σημαντικό για το όχημα, επειδή χρησιμοποιούνται δύο αισθητήρες TOF400C.

Το TOF400C παρέχει μια προγραμματιζόμενη διεύθυνση I2C και περιλαμβάνει μια είσοδο XSHUT που μπορεί να χρησιμοποιηθεί για να θέσει τη συσκευή σε λειτουργία απενεργοποίησης. Η STMicroelectronics προτείνει συγκεκριμένα τη χρήση του σήματος XSHUT όταν χρησιμοποιούνται πολλαπλοί αισθητήρες στον ίδιο δίαυλο I2C.[8]

Κατά την ενεργοποίηση οι δύο αισθητήρες μπορούν επομένως να ενεργοποιηθούν ξεχωριστά. Ο πρώτος αισθητήρας ενεργοποιείται και του αντιστοιχίζεται μια νέα διεύθυνση I2C. Στη συνέχεια, ο δεύτερος αισθητήρας ενεργοποιείται και του αντιστοιχίζεται μια άλλη διεύθυνση. Μόλις ολοκληρωθεί αυτή η αρχικοποίηση, και οι δύο αισθητήρες μπορούν να λειτουργούν στον ίδιο δίαυλο.

Αυτή η διάταξη βοηθάει στην αποφυγή της ανάγκης για ξεχωριστό δίαυλο I2C για κάθε αισθητήρα και μειώνει τον αριθμό των απαιτούμενων συνδέσεων μεταξύ του ESP32 και του υποσυστήματος ανίχνευσης.

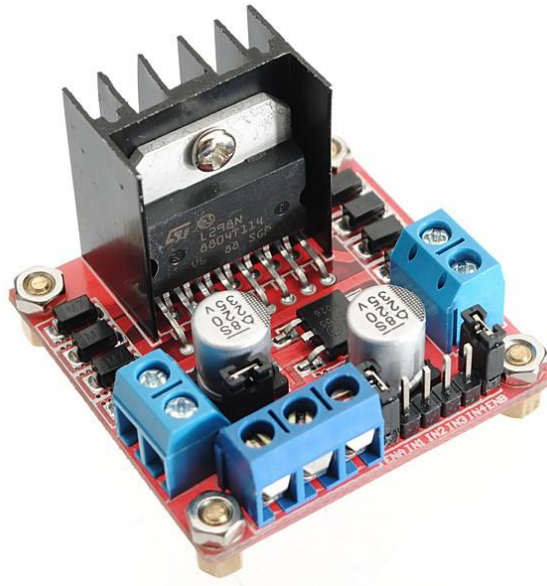
4.2.2 Ασύρματη Επικοινωνία

Ένα από τα χαρακτηριστικά του ESP32 για αυτό το έργο είναι το ενσωματωμένο WiFi. Αντί να εκτελεί τον πλήρη αλγόριθμο SLAM στον μικροεπεξεργαστή το ESP32 μπορεί να λειτουργήσει ως ενσωματωμένη μονάδα συλλογής δεδομένων και ελέγχου κίνησης ενώ ένας πιο ισχυρός υπολογιστής εκτελεί χαρτογράφηση, σχεδιασμό διαδρομής και επεξεργασία υψηλότερου επιπέδου για τον χρήστη.

Στην αρχιτεκτονική του συστήματος οι πληροφορίες για τους αισθητήρες και την κατάσταση του ρομπότ μπορούν επομένως να μεταδοθούν από το ESP32 στην εξωτερική εφαρμογή. Αντίθετα, οι εντολές πλοήγησης που υπολογίζονται από την εξωτερική εφαρμογή μπορούν να σταλθούν πίσω στο ESP32.

Η Espressif καθορίζει την υποστήριξη WiFi για το ESP32 χρησιμοποιώντας το πρότυπο IEEE 802.11b/g/n στη ζώνη των 2,4GHz, με ρυθμούς δεδομένων 802.11n έως και 150Mbps στο φυσικό επίπεδο[8]. Οι απαιτήσεις της εφαρμογής, ωστόσο, είναι πολύ μικρότερες από το θεωρητικό ρυθμό του WiFi καθώς μεταδίδουν κυρίως μικρά πακέτα json με μετρήσεις και εντολές. Ωστόσο η επεξεργασία λογισμικού και το ασύρματο περιβάλλον έχουν ως αποτέλεσμα χαμηλότερη αποτελεσματική απόδοση.

4.3 Motor Driver



Εικόνα 4.4 Motor Driver L298N

Οι κινητήρες DC δεν μπορούν να συνδεθούν απευθείας στο ESP32 επειδή οι έξοδοι GPIO του μικροελεγκτή προορίζονται για σήματα ελέγχου λογικού επιπέδου και όχι για την παροχή του ρεύματος που απαιτείται από τους κινητήρες (περίπου 1A σε πλήρη χρήση). Απαιτείται ένας motor driver μεταξύ του ESP32 και των κινητήρων.

Η πλατφόρμα χρησιμοποιεί έναν motor driver διπλής γέφυρας H (H bridge) που βασίζεται στο L298N. Η συσκευή L298N είναι ένας οδηγός διπλής πλήρους γέφυρας που έχει σχεδιαστεί για τον έλεγχο φορτίων συμπεριλαμβανομένων των κινητήρων συνεχούς ρεύματος. Παρέχει δύο ανεξάρτητα κανάλια γέφυρας και ξεχωριστές εισόδους ενεργοποίησης, επιτρέποντας τον ανεξάρτητο έλεγχο των δύο καναλιών κινητήρα.[9]

Η βασική λειτουργία του motor driver είναι να λειτουργεί ως γέφυρα μεταξύ των λογικών σημάτων χαμηλής ισχύος που παράγονται από το ESP32 (PWM) και των ηλεκτρικών φορτίων υψηλότερου ρεύματος που χρειάζονται οι κινητήρες. Η κατεύθυνση ενός κινητήρα ελέγχεται μέσω των σημάτων εισόδου της αντίστοιχης γέφυρας. Αλλάζοντας την πολικότητα που εφαρμόζεται στον κινητήρα, ο κινητήρας μπορεί να περιστρέφεται προς οποιαδήποτε κατεύθυνση.

Για την πλατφόρμα αυτή το όχημα έχει δύο καταστάσεις:

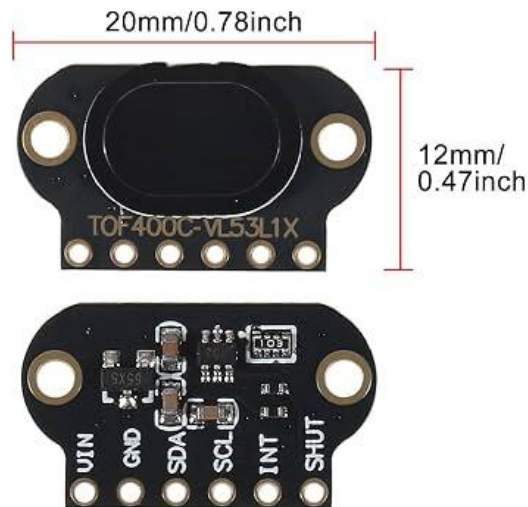
- Forward
- Stop

Η κίνηση αριστερά και δεξιά ελέγχεται από το σέρβο.

4.4 Αισθητήρα TOF400C

Η πλατφόρμα χρησιμοποιεί δύο αισθητήρες απόστασης VL53L1X Χρόνου Πτήσης (TOF) που κατασκευάζονται από την STMicroelectronics. Ο VL53L1X ανήκει στην οικογένεια FlightSense της

ST και χρησιμοποιεί έναν αόρατο πομπό λέιζερ Κλάσης 1 940 nm μαζί με μια συστοιχία λήψης SPAD για την εκτέλεση μετρήσεων απόστασης.[11]



Εικόνα 4.5 Αισθητήρας TOF400C

Σε αντίθεση με τους συμβατικούς αισθητήρες έντασης υπέρυθρων ο TOF400C μετράει την απόσταση χρησιμοποιώντας τον χρόνο που απαιτείται για να ταξιδέψει το φως που εκπέμπει σε έναν στόχο και να επιστρέψει στον δέκτη. Αυτό καθιστά τη μέτρηση λιγότερο εξαρτώμενη από τη φωτεινότητα ενός αντικειμένου σε σχέση με τους παραδοσιακούς ανακλαστικούς αισθητήρες υπέρυθρων.

Η ST καθορίζει εμβέλεια έως περίπου 4 m υπό κατάλληλες συνθήκες και συχνότητα εμβέλειας έως 50 Hz. Ο αισθητήρας παρέχει επίσης μια προγραμματιζόμενη περιοχή ενδιαφέροντος (FOV), επιτρέποντας τη μείωση του ενεργού οπτικού πεδίου.[11]

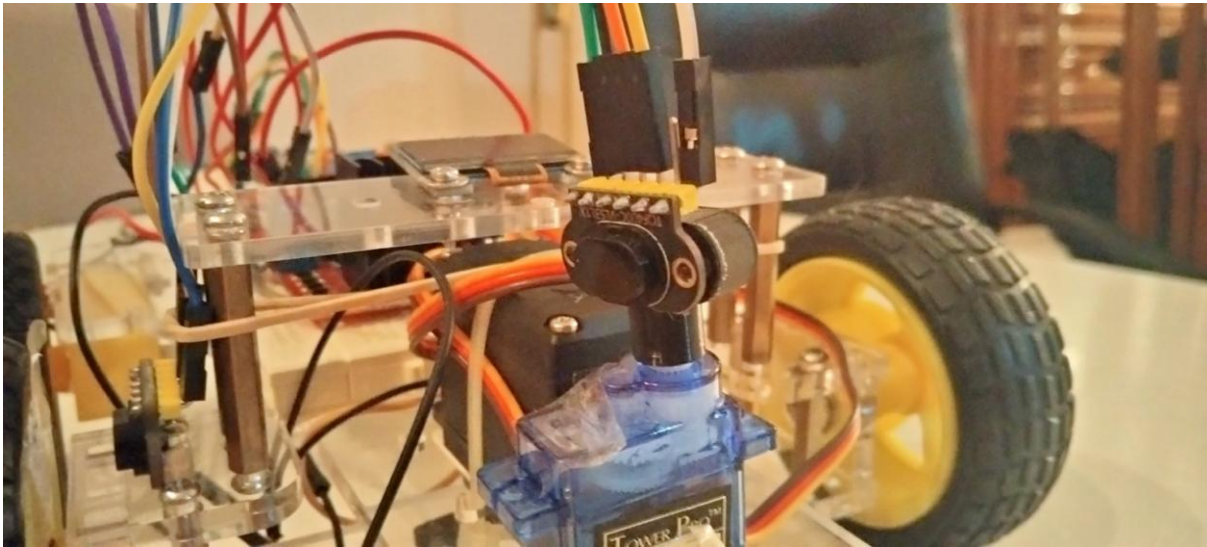
4.4.1 Διεπαφή αισθητήρα

Τα σχετικά σήματα του αισθητήρα TOF400C είναι:

- VIN συνδέεται στην 3,3V έξοδο του ESP32
- Ground συνδέεται με την αρνητική έξοδο της μπαταρίας
- Η γραμμή SDA είναι κοινή και για τους δύο αισθητήρες και συνδέεται στο GPIO21
- Η γραμμή SCL είναι κοινή και για τους δύο αισθητήρες και συνδέεται στο GPIO22
- XSHUT κάθε αισθητήρας έχει μια διεύθυνση. Ο αισθητήρας 1 στη GPIO 33 και αισθητήρας 2 στη GPIO 32

4.4.2 Τοποθέτηση του αισθητήρα

Για τη σωστή γωνιακή κάλυψη ο πρώτος αισθητήρας έχει τοποθετηθεί πάνω σε ένα σέρβο SG90. Το σέρβο έχει γωνία ελέγχου 90 μοίρες (-45, 45) και κέντρο τις 0 μοίρες στον άξονα της κατεύθυνσης του οχήματος. Έτσι του επιτρέπει να έχει μια ολόκληρη κάλυψη του χώρου κατά την πορεία του οχήματος. Ο δεύτερος αισθητήρας έχει τοποθετηθεί στον μπροστινό δεξιό τροχό έτσι ώστε να ελέγχει την κίνηση του και να βοηθάει στην οδομετρία.



Εικόνα 4.6 Αισθητήρας TOF400C στην βάση σέρβο

4.5 Servo και κατεύθυνση

Το όχημα χρησιμοποιεί ένα μικρό σέρβο SG90 για τον έλεγχο του μηχανισμού διεύθυνσης και έναν ακόμα για την χαρτογράφηση. Επιλέχθηκε σέρβο αντί να κινεί απευθείας τον μηχανισμό διεύθυνσης με κινητήρα συνεχούς ρεύματος γιατί ένας σέρβο παρέχει ακριβή έλεγχο της θέσης ενός οχήματος.

Ο TowerPro SG90 είναι ένας μικρός σέρβο περίπου 9 g που προορίζεται για εφαρμογές που απαιτούν ελεγχόμενη γωνιακή τοποθέτηση. Οι προδιαγραφές του κατασκευαστή για το SG90 περιλαμβάνουν τάση λειτουργίας 4,8 V και ροπή στρέψης περίπου 1,8 kg*cm για την αναλογική έκδοση.[10]

Το σέρβο διεύθυνσης παρέχει την γωνιακή εντολή που είναι απαραίτητη για την αλλαγή του προσανατολισμού των μπροστινών τροχών του ρομπότ. Αυτή η σχέση μπορεί να αναπαρασταθεί ως εξής: $\Theta_s = f(PWM)$

όπου Θ_s η γωνία κατεύθυνσης και PWM είναι ο παλμός ελέγχου που εφαρμόζεται στον σέρβο.

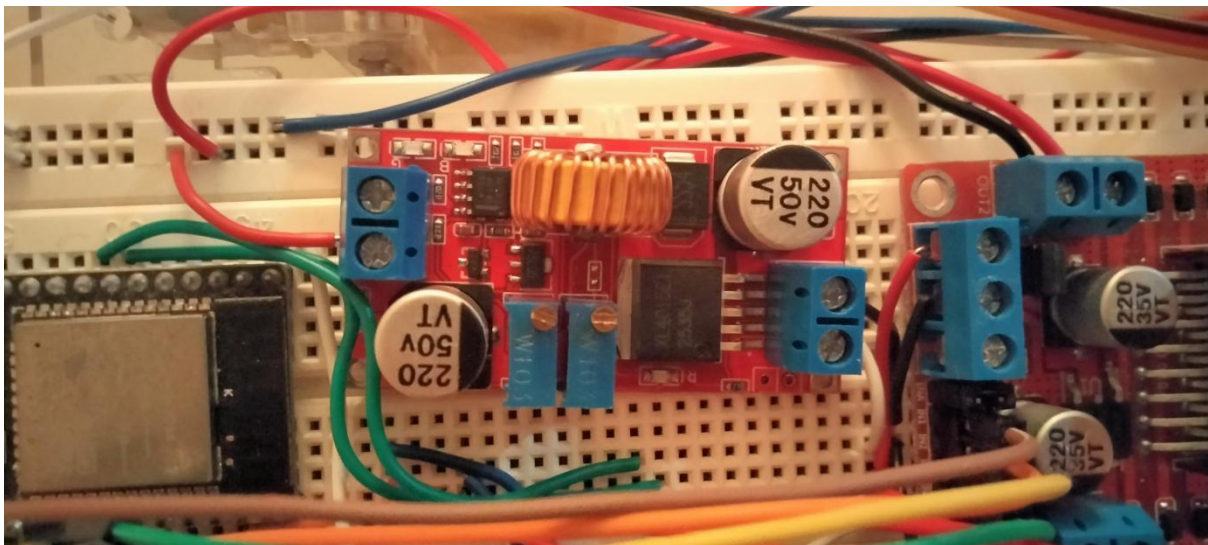
Το σέρβο λαμβάνει ένα σήμα ελέγχου του οποίου το πλάτος καθορίζει την επιθυμητή γωνιακή θέση. Η κεντρική θέση χρησιμοποιείται ως βασική θέση διεύθυνσης ενώ η αύξηση ή η μείωση της εντολής κινεί τον μηχανισμό διεύθυνσης προς την αντίστοιχη κατεύθυνση.

4.6 Σύστημα τροφοδοσίας

Το σύστημα τροφοδοσίας παρέχει την ηλεκτρική ενέργεια που απαιτείται από όλα τα υλικά του οχήματος. Τα διαφορετικά εξαρτήματα απαιτούν διαφορετικά επίπεδα τάσης και έχουν σημαντικά διαφορετικές απαιτήσεις ρεύματος. Το όχημα έχει τρεις διαφορετικές απαιτήσεις ενέργειας:

- Motor driver 6-12V μπορεί να φτάσει μέχρι και 2A σε αιχμές
- Μικροελεγκτής και σέρβο 5V το ESP32 μπορεί να χρειαστεί έως και 500mA κατά την επικοινωνία πολλαπλών δεδομένων με την εφαρμογή και το κάθε σέρβο μπορεί να φτάσει στο 1A
- Αισθητήρες 3,3V το ρεύμα δεν υπερβαίνει τα 50mA

Για αυτόν τον λόγο έχει προστεθεί ένας XL4015 ρυθμιστής τάσης στα 5V. Το motor driver και XL4015 συνδέεται κατευθείαν με την μπαταρία, ο μικροελεγκτής και σέρβο με το XL4015 και οι TOF400C με τον ελεγχτή.



Εικόνα 4.7 Ρυθμιστής τάσης XL4015

Η συνολική ηλεκτρική ισχύς του συστήματος μπορεί να υπολογιστεί ως :

$$P_{total} = P_{motor} + P_{esp32} + P_{sensors} + P_{servos}$$

Κεφάλαιο 5ο: Αρχιτεκτονική Λογισμικού

Το λογισμικό για την πλατφόρμα χωρίζεται σε δύο κύρια περιβάλλοντα: το ενσωματωμένο λογισμικό που εκτελείται στον μικροελεγκτή ESP32 και την εφαρμογή (C# και C++) που εκτελείται στη μηχανή παιχνιδιών Godot. Αυτή η διαίρεση ακολουθεί την διαμέριση υλικού και λογισμικού που περιγράφεται στο Κεφάλαιο 3 και επιτρέπει τον διαχωρισμό των λειτουργιών υλικού χαμηλού επιπέδου από τις υπολογιστικά πιο απαιτητικές λειτουργίες χαρτογράφησης και πλοήγησης.

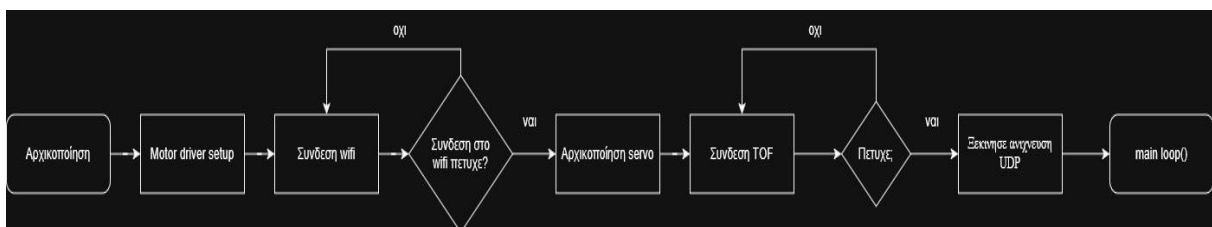
Το ενσωματωμένο λογισμικό είναι υπεύθυνο για την άμεση αλληλεπίδραση με το φυσικό υλικό του οχήματος. Αρχικοποιεί και επικοινωνεί με τους αισθητήρες VL53L1X, ανιχνεύει την κίνηση των τροχών χρησιμοποιώντας τον αισθητήρα VL53L1X, ελέγχει τους κινητήρες και το σέρβο διεύθυνσης και διαχειρίζεται την ασύρματη επικοινωνία με τον υπολογιστή.

Η εφαρμογή Godot λειτουργεί σε υψηλότερο επίπεδο. Λαμβάνει τις πληροφορίες που παράγονται από τον ενσωματωμένο ελεγκτή και τις χρησιμοποιεί για να διατηρεί μια εκτίμηση της θέσης του ρομπότ, να ενημερώνει τον χάρτη, να εικονοποιεί τις μετρήσεις των αισθητήρων, να δημιουργεί διαδρομές πλοήγησης και να στέλνει εντολές κίνησης πίσω στο ρομπότ.

Το ολοκληρωμένο σύστημα σχηματίζει έναν συνέχει κλειστό βρόχο ανατροφοδότησης. Οι πληροφορίες των αισθητήρων λαμβάνονται από το ρομπότ και μεταδίδονται στην εφαρμογή όπου χρησιμοποιούνται για την ενημέρωση της αναπαράστασης του χώρου. Η πορεία κίνησης στη συνέχεια μεταδίδεται πίσω στο όχημα και μετατρέπονται σε φυσική κίνηση.

5.1 Ενσωματωμένο λογισμικό

Το λογισμικό είναι οργανωμένο γύρω από την τυπική δομή ενσωματωμένου προγραμματισμού που αποτελείται από ένα στάδιο αρχικοποίησης και έναν συνεχώς εκτελούμενο κύριο βρόχο. Κατά την αρχικοποίηση διαμορφώνονται οι απαιτούμενες επικοινωνίες και τα περιφερειακά υλικού. Οι αισθητήρες αρχικοποιούνται, οι διευθύνσεις τους διαμορφώνονται όπου είναι απαραίτητο, προετοιμάζονται οι ακροδέκτες ελέγχου κινητήρα και ξεκινά το υποσύστημα ασύρματης επικοινωνίας.



Εικόνα 5.1 Διάγραμμα ροής ελεγκτή ESP32

5.1.1 Αρχικοποίηση βιβλιοθήκης και υλικού

Το πρόγραμμα του ESP32 χρησιμοποιεί διάφορες βιβλιοθήκες για να παρέχει πρόσβαση στο απαιτούμενο υλικό και τις λειτουργίες επικοινωνίας. Το λογισμικό που αναπτύχθηκε για το έργο χρησιμοποιεί τη λειτουργικότητα WiFi του ESP32 μαζί με την υλοποίηση του διακομιστή WebSocket, την επικοινωνία I2C για τους αισθητήρες TOF, τη βιβλιοθήκη VL53L1X για τη μέτρηση απόστασης και το ArduinoJson για τη μετάδοση δομημένων δεδομένων καθώς και τη βιβλιοθήκη το UDP ώστε να μπορεί ο χρήστης να συνδέεται εύκολα από την εφαρμογή.

```
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <Wire.h>
#include <VL53L1X.h>
#include <ArduinoJson.h>
#include <WiFiUdp.h>
```

5.1.2 Αρχικοποίηση υλικού

Το ενσωματωμένο πρόγραμμα ορίζει τις ακίδες και τις παραμέτρους υλικού που χρησιμοποιούνται από το ρομπότ. Αυτοί οι ορισμοί περιλαμβάνουν τη διεπαφή I2C, τις ακίδες απενεργοποίησης αισθητήρα, τις ακίδες ελέγχου κινητήρα και της ακίδες των σέρβο.

Η διατήρηση αυτών των τιμών ως σταθερές ή σαν ορισμοί επιτρέπει την τροποποίηση της διαμόρφωσης υλικού χωρίς να αλλάξει η κύρια λογική του προγράμματος.

```
#define SDA 21
#define SCL 22
#define SERVO_PIN 27
#define PAN_SERVO_PIN 14
//===== MOTOR DRIVER =====
#define IN1 4
#define IN2 5
#define IN3 16
#define IN4 17

//ToF
const uint8_t XSHUT_A = 33;
const uint8_t XSHUT_B = 32;
```

5.1.3 Συνάρτηση αρχικοποίησης

Η συνάρτηση `setup()` είναι υπεύθυνη για την προετοιμασία του οχήματος για σωστή λειτουργία. Η ακολουθία αρχικοποίησης είναι ιδιαίτερα σημαντική για τους αισθητήρες VL53L1X επειδή πολλές συσκευές μοιράζονται τον ίδιο δίαυλο I2C οπότε πρέπει να αρχικοποιείται το ένα μετά το άλλο με σωστό χρόνο μεσολάβησης.

Όταν πολλοί αισθητήρες VL53L1X είναι συνδεδεμένοι στον ίδιο δίαυλο I2C, οι προεπιλεγμένες διευθύνσεις τους πρέπει να διαχειρίζονται έτσι ώστε κάθε αισθητήρας να μπορεί να συνδέεται ανεξάρτητα. Οι είσοδοι τερματισμού λειτουργίας χρησιμοποιούνται επομένως κατά την αρχικοποίηση για την ενεργοποίηση των αισθητήρων ξεχωριστά και την εκχώρηση των απαιτούμενων διευθύνσεων I2C τους.

```
void setup()
{
    Serial.begin(115200);
    //motor driver
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    motorStop();
    //websocket
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print("-");
    }
    Serial.println();
    Serial.print("connected with IP : \n");
    Serial.println(WiFi.localIP());
    websocket.begin();
    websocket.onEvent(webSocketEvent);
    //servos
    ledcAttach(SERVO_PIN, 50, 16);
    ledcAttach(PAN_SERVO_PIN, 50, 16);
    servoWriteAngle(0);
    panServoWriteAngle(0);
    //ToF
    Wire.begin(SDA, SCL);
    setupToFSensors();
}
```

```

udp.begin(UDP_PORT);
Serial.println("UDP discovery started");
}

```

5.1.4 Κύρια συνάρτηση επανάληψης

Μετά την αρχικοποίηση το ESP32 εκτελεί επανειλημμένα τον κύριο βρόχο προγράμματος. Ο κύριος βρόχος συντονίζει τις διαφορετικές εργασίες του ρομπότ. Αυτές οι εργασίες θα πρέπει να οργανώνονται έτσι ώστε η λήψη, η επικοινωνία και ο έλεγχος κίνησης των αισθητήρων να μην μπλοκάρουν άσκοπα η μία την άλλη.

5.2 Λογισμικό αισθητήρων

Το λογισμικό του αισθητήρα είναι υπεύθυνο για τη λήψη πληροφοριών σχετικά με το φυσικό περιβάλλον και τη μετατροπή των μετρήσεων του υλικού σε δεδομένα που μπορούν να υποβληθούν σε επεξεργασία από το σύστημα χαρτογράφησης.

5.2.1 Αρχικοποίηση VL53L1X

```

void setupTofSensors()
{
    pinMode(XSHUT_A, OUTPUT);
    pinMode(XSHUT_B, OUTPUT);
    digitalWrite(XSHUT_A, LOW);
    digitalWrite(XSHUT_B, LOW);
    delay(10);
    digitalWrite(XSHUT_A, HIGH);
    delay(10);
    initTofSensor(sensorA, ADDR_A, 33000, 33);
    digitalWrite(XSHUT_B, HIGH);
    delay(10);
    initTofSensor(sensorB, ADDR_B, 20000, 20, true);
    Serial.println("Both TOF sensors initialized.");
}

```

Αυτή είναι η συνάρτηση αρχικοποίησης για τους αισθητήρες VL53L1X καθένας έχει δικό του σκοπό οπότε πρέπει να έχει και διαφορετική αρχικοποίηση. Ο πρώτος που είναι για τη χαρτογράφηση του περιβάλλοντος οπότε πρέπει να είναι πιο ακριβής. Σύμφωνα με το datasheet του κατασκευαστή η καλύτερη λειτουργία για πιο ακριβείς μετρήσεις είναι στα 20ms-50ms οπότε 33ms είναι πιο σίγουρη μέτρηση. Ο δεύτερος που μετράει τις περιστροφές του τροχού πρέπει να είναι όσο πιο γρήγορος γίνεται οπότε έχει τεθεί στα 20ms το κατώτατο όριο.

5.2.2 Μετρήσεις VL53L1X

Μόλις οι αισθητήρες λειτουργούν το ESP32 λαμβάνει περιοδικά την πιο πρόσφατη μέτρηση από κάθε συσκευή. Οι τιμές απόστασης αρχικά αναπαρίστανται σε χιλιοστά. Αυτές οι τιμές στη συνέχεια συσχετίζονται με τον αντίστοιχο προσανατολισμό τοποθέτησης του αισθητήρα. Αυτή η διάκριση είναι σημαντική επειδή δύο πανομοιότυπες μετρήσεις απόστασης που λαμβάνονται από αισθητήρες με διαφορετικό προσανατολισμό δεν αναπαριστούν την ίδια κατεύθυνση στο σύστημα συντεταγμένων του ρομπότ.

```
int readValidDistance(VL53L1X &sensor, bool SensorB=false) {
    if (!sensor.dataReady())
        return -2;
    int dist = sensor.read(false);
    uint8_t status = sensor.ranging_data.range_status;
    if (status != 0) return -1;
    if (dist <= 0 || dist > MAX_VALID_RANGE_MM )
        return -1;
    return dist;
}
```

Στην κύρια συνάρτηση του προγράμματος loop() για κάθε αισθητήρα καλείται η συνάρτηση readValidDistance όπου ανιχνεύει άμα ο αισθητήρας είναι έτοιμος να παρέχει κάποια μέτρηση και, αν δώσει μέτρηση, αυτή να είναι έγκυρη που να μην υπερβαίνει το ανώτατο όριο.

5.2.3 Κατεύθυνση του αισθητήρα

Ο κύριος αισθητήρας απόστασης που είναι υπεύθυνος για την χαρτογράφηση είναι τοποθετημένος σε ένα σερβοκινητήρα όπου κινείται προς μια από τις κατευθύνσεις από -45 έως 45 μοίρες. Ωστόσο κατά την οδήγηση του οχήματος αυτή η εμβέλεια μικραίνει στα -15 με 15 μοίρες με άξονα την κατεύθυνση του οχήματος.

```
void updatePanSweep()
{
    unsigned long now = millis();
    if (now - lastPanUpdate < PAN_UPDATE_INTERVAL_MS)
        return;
    lastPanUpdate = now;
    int sweepLimit = (driveState == DRIVING) ? 15 : 45;
    int nextOffset = currentPanAngle + PAN_STEP_DEG * panDirection;
    if (nextOffset >= sweepLimit)
    {
```

```

    nextOffset = sweepLimit;
    panDirection = -1;
}
else if (nextOffset <= -sweepLimit)
{
    nextOffset = -sweepLimit;
    panDirection = 1;
}
int physicalAngle = -(currentSteerAngle + nextOffset);
physicalAngle = constrain(physicalAngle, -90, 90);
int servoAngle = PAN_CENTER + physicalAngle;
int pulseWidth = map(servoAngle, 0, 180, 500, 2500);
int duty = (pulseWidth * 65535L) / 20000L;
ledcWrite(PAN_SERVO_PIN, duty);
currentPanAngle = nextOffset;
}

```

Κάθε 30 ms η θέση του σέρβο αυξάνεται προς την κατεύθυνση που κινούνταν μέχρι να φτάσει στη μέγιστη όπου αρχίζει να κινείται προς την αντίθετη κατεύθυνση .

5.3 Έλεγχος ηλεκτρικών μοτέρ

Η κατεύθυνση ενός κινητήρα συνεχούς ρεύματος μπορεί να ελεγχθεί από τον συνδυασμό σημάτων στις ακίδες εισόδου του οδηγού του κινητήρα. Επομένως το ενσωματωμένο λογισμικό παρέχει λειτουργίες που περιγράφουν τον χειρισμό των ακίδων χαμηλού επιπέδου. Για την εύκολη λειτουργία του οχήματος έχει δύο καταστάσεις: όταν είναι ακίνητο IDLE και όταν κινείται DRIVING. Το πρόγραμμα έχει δύο συναρτήσεις οι οποίες ελέγχουν τις ακίδες για να κινηθεί το όχημα ή να σταματήσει. Οι συναρτήσεις καλούνται από το πρόγραμμα όταν μια νέα διαδρομή έχει σταλθεί σε μορφή JSON. Η μορφή είναι ένας πίνακας {path:[{Θ[n], D[n]}]} όπου Θ είναι ένας πίνακας με τις γωνίες και D είναι Ένας πίνακας με τις αποστάσεις (σε εκατοστά) των σημείων ελέγχου της διαδρομής.

Μόλις δημιουργηθούν οι δύο πίνακες με τις γωνίες και τις αποστάσεις καλείται η συνάρτηση updateDriving() όπου είναι υπεύθυνη για την κίνηση σε κάθε σημείο έλεγχου της πορείας.

```

void updateDriving()
{
    if (driveState != DRIVING)
        return;

    bool sensorLookingForward = abs(currentPanAngle) <= ObstacleLimit;
    if (sensorLookingForward && lastDistA > 0 && lastDistA <= ObstacleStopDis)
    {

```

```

    emergencyStop();
    return;
}

if (pathQueueIndex >= pathQueueCount)
{
    driveState = IDLE;
    motorStop();
    return;
}

PathPoint &pt = pathQueue[pathQueueIndex];
updateSteering(pt.angle);
float traveledThisSegment = getDistanceTraveledCm() - segmentStartDistanceCm;
bool distanceReached = traveledThisSegment >= pt.distance;
bool stalled = (millis() - segmentStartTime) >= (segmentDurationMs * 3);
if (distanceReached || stalled)
{
    int completedIndex = pathQueueIndex;
    pathQueueIndex++;
    bool pathComplete = (pathQueueIndex >= pathQueueCount);
    sendSegmentComplete(completedIndex, pathComplete);
    if (pathComplete)
    {
        driveState = IDLE;
        motorStop();
        Serial.println("Path complete.");
        return;
    }
    startNextPathPoint();
}
}

```

Η κατεύθυνση του οχήματος ελέγχεται χρησιμοποιώντας έναν μηχανισμό διεύθυνσης που βασίζεται σε σέρβο. Το ESP32 παράγει το κατάλληλο σήμα ελέγχου με βάση την εντολή πλοήγησης. Μόλις αρχίσει να κινείται σε ένα τμήμα το όχημα αρχίζει να κινείται προς τα εμπρός και η γωνία στόχος διεύθυνσης που σχετίζεται με αυτό το τμήμα μεταβιβάζεται στον ελεγκτή διεύθυνσης. Η γωνία διεύθυνσης καθορίζεται επομένως από την σχεδιασμένη τροχιά αντί να ελέγχεται απευθείας από τους κινητήρες κίνησης. Στη συνέχεια, καθορίζει τη μέγιστη γωνιακή αλλαγή που επιτρέπεται κατά τη διάρκεια του τρέχοντος διαστήματος ελέγχου από τον χρόνο που έχει παρέλθει και έναν προκαθορισμένο ρυθμό διεύθυνσης (160). Εάν η απαιτούμενη διόρθωση είναι μικρότερη από αυτό το επιτρεπόμενο βήμα το σέρβο μετακινείται απευθείας στον στόχο.

```

void updateSteering(float targetAngle)
{
    unsigned long now = millis();
    if (lastSteeringUpdate == 0)
    {
        lastSteeringUpdate = now;
        return;
    }
    float dt = (now - lastSteeringUpdate) / 1000.0f;
    lastSteeringUpdate = now;
    targetAngle = constrain(targetAngle, -40.0f, 40.0f);
    float difference = targetAngle - currentSteerAngle;
    float maxStep = STEERING_SMOOTH_SPEED * dt;
    if (abs(difference) <= maxStep)
    {
        servoWriteAngle((int)targetAngle);
    }
    else if (difference > 0)
    {
        servoWriteAngle(currentSteerAngle + (int)maxStep);
    }
    else
    {
        servoWriteAngle(currentSteerAngle - (int)maxStep);
    }
}

```

5.4 Εφαρμογή με χρήση Godot

Το λογισμικό από την πλευρά του υπολογιστή υλοποιείται χρησιμοποιώντας τη μηχανή παιχνιδιών Godot, C# και C++. Το Godot παρέχει το γραφικό περιβάλλον που απαιτείται για την απεικόνιση του ρομπότ και του περιβάλλοντός του, ενώ η C# χρησιμοποιείται για την υλοποίηση της λογικής ελέγχου και επεξεργασίας της εφαρμογής και επιτρέπει τη δημιουργία C++ επεκτάσεων για κομμάτια του προγράμματος που είναι ιδιαίτερα απαιτητικά στους υπολογιστικούς πόρους όπως η εύρεση διαδρομής και εξομάλυνσης πορείας. Το Godot επιλέχτηκε γιατί ουσιαστικά δημιουργείται μια διαδραστική οπτικοποίηση σε πραγματικό χρόνο, κάτι που ακριβώς έχει σχεδιαστεί για να κάνει μια μηχανή παιχνιδιών. Προσφέρει επίσης δισδιάστατη απόδοση, χειριστήρια UI, εισαγωγή με ποντίκι και πληκτρολόγιο, δικτύωση, υποστήριξη C#, επεκτάσεις C++ και ανάπτυξη για διάφορες πλατφόρμες όπως εφαρμογή για Windows, Android και browser.

Η εφαρμογή είναι υπεύθυνη για:

- Να έχει επικοινωνία με το ESP32
- Λήψη μετρήσεων αισθητήρα
- Ερμηνεία πληροφοριών κατάστασης οχήματος
- Μετατροπή μετρήσεων αισθητήρα σε συντεταγμένες του χάρτη
- Ενημέρωση του χάρτη για εμπόδια
- Δημιουργία διαδρομών πλοήγησης
- Εξομάλυνση διαδρομών
- Μετάδοση εντολών κίνησης

Το Godot είναι οργανωμένο σε κόμβους και σενάρια, με μεμονωμένα στοιχεία υπεύθυνα για συγκεκριμένη λειτουργικότητα. Το Godot επιτρέπει όμως την επικοινωνία μεταξύ σεναρίων με τη χρήση σημάτων (signals) τα οποία δρουν ως χειριστές συμβάντων για την εφαρμογή.

Η επικοινωνία μεταξύ του Godot και του ESP32 γίνεται από έναν ειδικό διαχειριστή αντί να κατανέμεται σε όλο τον κώδικα χαρτογράφησης και πλοήγησης. Αυτός ο σχεδιασμός δίνει πρόσβαση στην επικοινωνία μόνο σε συγκεκριμένα σενάρια της εφαρμογής που χρειάζονται επικοινωνία με τον ελεγκτή. Ο διαχειριστής επικοινωνίας είναι υπεύθυνος για να στέλνει μηνύματα αλλά και για να δέχεται μηνύματα από το ESP32 και να αποφασίζει σε ποιο μέρος της εφαρμογής πρέπει να προωθούνται τα δεδομένα. Με αυτόν τον τρόπο δεν χρειάζεται ποτέ η κεντρική ροή του προγράμματος να σταματήσει για την επικοινωνία.

Κεφάλαιο 6ο: S.L.A.M. και κατασκευή χάρτη

Αυτό το κεφάλαιο παρουσιάζει την αρχιτεκτονική και την υλοποίηση του συστήματος χαρτογράφησης που αναπτύχθηκε για το όχημα. Το σύστημα λαμβάνει μετρήσεις γωνίας και απόστασης από τον ελεγκτή ESP32 και μετατρέπει κάθε μέτρηση στο πλαίσιο του χάρτη εφαρμογής, σημειώνει τον χώρο ως ελεύθερο ή κατειλημμένο, εισάγει εμπόδια και αποθηκεύει τον προκύπτοντα χάρτη σε τμήματα *chunks*. Η ίδια αναπαράσταση χρησιμοποιείται στη συνέχεια για οπτικοποίηση, εξερεύνηση, σχεδιασμό διαδρομής με αποφυγή εμποδίων και επανασχεδιασμό διαδρομής.

6.1 Αρχιτεκτονική συστήματος SLAM

Το λογισμικό χωρίζεται σε στοιχεία ανίχνευσης, επικοινωνίας, μετασχηματισμού συντεταγμένων, αποθήκευσης χάρτη και πλοήγησης. Το ESP32 μεταδίδει μηνύματα αισθητήρων στην εφαρμογή Godot μέσω μιας σύνδεσης WebSocket. Ένα μήνυμα αισθητήρα έχει τον τύπο "sensorData" και περιέχει δύο κύρια πεδία: τη γωνία αισθητήρα (*angleA*) και την απόσταση (*distanceA*). Το *WebSocketManager* αναλύει το πακέτο JSON και εκπέμπει ένα σήμα *ESP32Data*. Η κύρια σκηνή λαμβάνει αυτό το σήμα και συνδυάζει τη γωνία αισθητήρα με την τρέχουσα κατεύθυνση του οχήματος, μετατρέπει την μετρούμενη απόσταση από χιλιοστά σε κελιά πλέγματος (η αναλογία είναι ένα κελί 1εκ.) και υπολογίζει το τελικό σημείο.

Η κύρια σκηνή πλοήγησης του προγράμματος είναι η *DrawingAppGrid*. Δουλεύει σαν μια κλάση όπου μέσα της έχει τις απαραίτητες συναρτήσεις για επιλογή της πορείας.

Ο ίδιος ο χάρτης ανήκει στο *ChunkContainer* κόμβο. Αυτός ο κόμβος αποθηκεύει κελιά και ποινές πλοήγησης σε δύο *Lists*. Μετατρέπει επίσης τις καθολικές συντεταγμένες κελιών σε *chunk* και τοπικές συντεταγμένες, δημιουργεί *chunk* και επεκτείνει τα όρια χαρτογράφησης.

6.2 Αναπαράσταση εικονικού χάρτη

Ο χάρτης του χώρου αναπαρίσταται από ένα δισδιάστατο πλέγμα. Στον υλοποιημένο χάρτη κάθε κελί έχει μια από τις τρεις καταστάσεις: 0 είναι άγνωστο, 1 είναι ελεύθερο, 2 είναι εμπόδιο. Ο χάρτης του πλέγματος αποθηκεύει και μια ακόμα τιμή, ξεχωριστά από την κατάσταση του, με την τιμή της ποινής του κελιού.

Η υλοποίηση διατηρεί ένα δεύτερο επίπεδο πλέγματος που περιέχει ποινές εμποδίων. Είναι ένας χάρτης κόστους πλοήγησης που προέρχεται από κατειλημμένα κελιά. Ο διαχωρισμός του πίνακα πληρότητας από τον πίνακα κόστους επιτρέπει στο σύστημα να διατηρήσει τη θέση του εμποδίου ενώ παράλληλα προτιμά διαδρομές που διατηρούν επιπλέον απόσταση.

Δυο διαφορετικές κλίμακες υπάρχουν στο σύστημα.

MM_PER_CELL είναι η φυσική ανάλυση του πλέγματος και είναι 1εκ.

CELL_SIZE είναι ανάλογο με την κλίμακα του παραθύρου του προγράμματος και είναι 5pixels.

Ένα chunk έχει 32 κελιά άρα ένα chunk καλύπτει $32*32 = 1024$ ή $0,1024$ μέτρα² Η ακτίνα του TOF400C φτάνει μέχρι τα 1,3 μέτρα ή 130 κελιά.

6.3 Σύστημα συντεταγμένων

Η εφαρμογή χρησιμοποιεί τέσσερις τύπους συντεταγμένων.

- Συντεταγμένες αισθητήρα στις οποίες μια μέτρηση αναπαρίσταται από μια σχετική γωνία σάρωσης Θ και μια εμβέλεια r .
- Συντεταγμένες θέσεως οχήματος των οποίων η αρχή είναι η αρχική θέση του οχήματος και ο προσανατολισμός του είναι η κατεύθυνση του οχήματος
- Συντεταγμένες του χάρτη σε κελιά πλέγματος και επιτρέπεται να λαμβάνουν θετικές ή αρνητικές ακέραιες τιμές.
- Συντεταγμένες οθόνης σε pixel σε σχέση με το κέντρο του παραθύρου προβολής Godot και την τρέχουσα θέση της κάμερας.

Η σχετική γωνία της ακτίνας του αισθητήρα μετατρέπεται στο πλαίσιο του χάρτη ως εξής:

Μετασχηματισμός γωνίας από τοπικό σε ολικό: $\Theta_t = Y_t + at$ όπου

- Y_t Προσανατολισμός κατεύθυνσης του οχήματος τη χρονική στιγμή t
- A_t η σχετική γωνία του αισθητήρα
- Θ_t Απόλυτη κατεύθυνση

Η συνολική απόσταση μετατροπή από μετρική κλίμακα σε κελιά της ακτίνας $dt = \frac{rt}{q}$

- rt η απόσταση του αισθητήρα σε mm
- $q = 10\text{mm/κελί}$
- dt η συνολική απόσταση σε κελιά

Τελική θέση του της ακτίνας του TOF400C $pt = \begin{bmatrix} x_t + dt * \cos\theta_t \\ y_t + dt * \sin\theta_t \end{bmatrix}$

- x_t και y_t η τωρινή θέση του οχήματος στον χάρτη
- $dt*\cos\theta_t$ Οριζόντια μετατόπιση σε κελιά κατά μήκος της κατεύθυνσης της ακτίνας
- $dt*\sin\theta_t$ Καθετη μετατόπιση σε κελιά κατά μήκος της κατεύθυνσης της ακτίνας

Έτσι υπολογίζει που η ακτίνα δείχνει στο σύστημα συντεταγμένων. Η υλοποίηση στρογγυλοποιεί και τα δύο στοιχεία τελικού σημείου στο πλησιέστερο ακέραιο κελί πριν από τη δήλωση του εμποδίου.

Οι συντεταγμένες του Godot είναι αντίστροφες ως προς τον άξονα Y: το θετικό x δείχνει δεξιά και το θετικό y δείχνει προς τα κάτω. Μια μαθηματικά θετική γωνία που χρησιμοποιείται απευθείας με το $\cos \theta$, $\sin \theta$ εμφανίζεται δεξιόστροφα στην οθόνη. Το ESP32 και η εφαρμογή Godot πρέπει επομένως να χρησιμοποιούν το ίδιο πρόσημο. Εάν το υλικό ορίζει τις θετικές γωνίες σάρωσης ως αριστερόστροφες στο καρτεσιανό πλαίσιο, η γωνία πρέπει να αναιρείται κατά τον μετασχηματισμό.

Ο μετασχηματισμός από χάρτη σε συντεταγμένες της οθόνης $s = m + (g - v) * p$ όπου:

- $S(x,y)$ είναι οι συντεταγμένες της οθόνης
- $g(x,y)$ είναι οι συντεταγμένες στον χώρο πλέγματος(grid)
- $v(x,y)$ είναι το κέντρο της οθόνης σε μονάδες του πλέγματος
- $m(x,y)$ είναι το πραγματικό κέντρο της οθόνης

Με αυτόν τον τρόπο ο χρήστης μπορεί να χειρίζεται την κάμερα ελεύθερα.

6.4 Ray casting

Στην κατασκευή του χάρτη για τη δημιουργία ακτίνας χρησιμοποιείται ως σημείο ελέγχου την τιμή που επιστρέφει ο αισθητήρας. Ο αισθητήρας παρέχει το τελικό σημείο και ο αλγόριθμος καθορίζει ποια κελιά πλέγματος βρίσκονται μεταξύ του ρομπότ και αυτού του τελικού σημείου. Τα ενδιαμέσα κελιά αντιπροσωπεύουν τον ορατό ελεύθερο χώρο ενώ ένα έγκυρο σημείο τέλους αντιπροσωπεύει ένα εμπόδιο. Αμα ο αισθητήρας δεν επιστρέψει κάποιο σημείο τότε η συνολική απόσταση 1,3μ θεωρείται πως είναι ελεύθερη. Η μέθοδος που γίνεται είναι η ScanAtAngle:

1. Διάνυσμα μετατόπισης d

$$d = p_t - p_{rt}$$

αρχικά υπολογίζει το διάνυσμα μετατόπισης από την τοποθεσία του ρομπότ p_{rt} στο τελικό σημείο της ακτίνας p_t

2. Μέγεθος ακτίνας

$$L = \sqrt{dx^2 + dy^2}$$

Υπολογίζει την απόσταση της ακτίνας

3. Μετατροπή σε τιμή κελιών

$$dis = \frac{d}{L}$$

Μειώνει την κλίμακα του διανύσματος σε διάνυσμα μήκους 1 κελιού

4. σημεία ελέγχου

$$P_i = p_d + i \cdot dis \text{ με } i = 0, 1, \dots, L$$

Κινείται κατά μήκος της ακτίνας

- $i = 0$ με αρχική θέση την θέση του οχήματος
- $i = 1, 2, \dots, L-1$ τα ενδιάμεσα σημεία κατά μήκος της διαδρομής (αυτά τα κελιά είναι ελεύθερα)
- $i = L$ τελευταίο σημείο ελέγχου είναι κοντά στην τελική θέση της ακτίνας

Για κάθε δείγμα ελέγχεται το αντίστοιχο κελί. Εάν είναι ήδη κατειλημμένο η διάσχιση τερματίζεται αλλιώς χαρακτηρίζεται ως ελεύθερο. Δεδομένου ότι το OnESP32Data εισάγει ένα έγκυρο τελικό σημείο ως κατειλημμένο πριν από την επόμενη διάσχιση, η ακτίνα κανονικά καθαρίζει τα κελιά που οδηγούν στο τελικό σημείο και σταματά όταν φτάσει στο εμπόδιο.

6.5 Επεξεργασία σάρωσης

Ξεκινάει στο επίπεδο επικοινωνίας όπου κάθε διαθέσιμο πακέτο websocket αποκωδικοποιείται σε μορφή JSON file. Έπειτα καθορίζεται εάν το πακέτο περιέχει δεδομένα αισθητήρα, μήνυμα θέσης οχήματος ή διακοπή έκτακτης- ανάγκης.

Η επεξεργασία μέτρησης είναι:

- Πρόσθεση της γωνίας του αισθητήρα στην τρέχουσα κατεύθυνση του οχήματος.
- Μετατροπή της γωνίας που προκύπτει από μοίρες σε γωνία
- Έλεγχος επικαιρότητας απόστασης.
- Μετατροπή της απόστασης από χιλιοστά σε κελιά.
- Στρογγυλοποίηση των συντεταγμένων σε τιμή του τελικού σημείου.
- Σημειώνει το τελικό σημείο ως κατειλημμένο (άμα επιστρέφει έγκυρη τιμή απόστασης που δεν είναι αρνητική)
- Άμα επιστρέψει μη έγκυρη τιμή (αρνητική) διαμορφώνει μέγιστη εμβέλεια στα 1,3 μετρα.
- Θέτει τον ελεύθερο χώρο που προκύπτει από το όχημα μέχρι το τελικό σημείο της μέτρησης.

Με αυτήν την επεξεργασία το σύστημα αναγνωρίζει ότι όταν μια τιμή απόστασης είναι αρνητική ο αισθητήρας δεν έχει εντοπίσει ένα εμπόδιο στην μεγίστη απόσταση που έχει τεθεί. Η τρέχουσα υλοποίηση αποθηκεύει το τελευταίο τελικό σημείο στην παγκόσμια μεταβλητή laserEnd και η εντολή ScanAtAngle καλείται από τη διαδρομή σχεδίασης.

6.6 Χαρτογράφηση χάρτη σε chunks

Ο χάρτης χωρίζεται σε ένα πλέγμα από κατακερματισμένα τετράγωνα το ένα δίπλα στο άλλο (chunks). Κάθε κομμάτι διαιρείται σε 32 μικρότερα ίδιου μεγέθους ορθογώνια κομμάτια, τα κελιά

(cells). Τα chunks δημιουργούνται μόνο όταν πραγματοποιείται κάποια εγγραφή μέσα σε αυτά και αποθηκεύονται ως μεταβλητή τύπου Dictionary :

```
Dictionary<Vector2I, int[,]> chunks = new Dictionary<Vector2I, int[,]>();
```

Όπου το κλειδί του λεξικού είναι η δισδιάστατη συντεταγμένη του τμήματος και η τιμή είναι ο τοπικός πίνακας κελιών.

Για την αντιστοίχιση από παγκόσμια συντεταγμένη ενός κελιού στην θέση του σε ένα chunk πρέπει:

$$C = \left(\left\lfloor \frac{i}{S} \right\rfloor, \left\lfloor \frac{j}{S} \right\rfloor \right)$$

όπου

- (i,j) είναι η παγκόσμια συντεταγμένη ενός κελιού στο σύστημα του χάρτη.
- S είναι το μέγεθος ενός chunk
- C το ακέραιο κλειδί που προκύπτει για την εύρεση του κατακερματισμένου τετράγωνου chunk στο λεξικό (dictionary).

Η στρογγυλοποίηση στον πλησιέστερο μικρότερο ακέραιο αριθμό ($\lfloor \cdot \rfloor$) γίνεται γιατί μπορούμε να έχουμε μόνο ακέραιες συντεταγμένες.

Ο υπολογισμός για την τοπική μετατόπιση του κελιού μέσα στο chunk γίνεται με

$$L = (\text{mod}(i), \text{mod}(j))$$

Όπου η τοπική τιμή μπορεί να είναι $L(x,y)$ υποσύνολο $[0,31]^2$.

Ο λόγος επιλογής υλοποίησης ενός συστήματος από ένα πλέγμα από κατακερματισμένα τετράγωνα είναι η δυνατότητα δημιουργίας ενός δυναμικά επεκτεινόμενου χάρτη. Καθώς το σύστημα μπορεί να επεκτείνεται απεριόριστα κατά μήκος του πλάτους και του ύψους αν δεν έχει τεθεί ένα όριο. Η αναζήτηση στο dictionary προσθέτει μια μικρή επιβάρυνση αλλά κάθε εγγραφή κελιού παραμένει ,κατά μέσο όρο σταθερού χρόνου.

Η εντολή GetCell δεν γραφεί στην μνήμη για ένα κομμάτι που δεν υπάρχει απλώς επιστρέφει την κατάσταση 0, άγνωστο. Η εντολή SetCell καλεί την GetOrCreateChunk που δημιουργεί το κομμάτι στην πρώτη εγγραφή. Κατά την προβολή του χάρτη η εφαρμογή μετατρέπει τα τρέχοντα όρια που είναι ορατά από την οθόνη σε ελάχιστες και μέγιστες συντεταγμένες chunks και δείχνει μόνο τα κομμάτια όπου πρακτικά φαίνονται στον χρήστη. Τα κομμάτια συνεχίζουν να λειτουργούν λογικά στην εύρεση διαδρομής αλλά η εφαρμογή δεν τα κάνει render αποφεύγοντας έτσι άσκοπες διεργασίες.

Τα όρια του λογικού χάρτη επεκτείνονται όταν το όχημα ή η ακτίνα του TOF400C πλησιάζουν στα άρια του υπάρχοντος χάρτη. Η επέκταση γίνεται πολλαπλασιάζοντας τα ακριανά chunks. Η παράμετρος maxChunks περιορίζει την επέκταση σε έναν αριθμό που προκαθόρισε ο χρήστης (100 τεμάχια από προεπιλογή).

6.7 Ποινές Εμποδίων

Σε κάθε έγκυρη μέτρηση απόστασης το τελικό σημείο αντιστοιχίζεται η τιμή πληρότητας 2 που δηλώνει πως είναι εμπόδιο. Η διέλευση του αλγορίθμου εύρεσης διαδρομής στον ελεύθερο χώρο σταματά όταν φτάσει σε ένα κατειλημμένο κελί. Ωστόσο αυτό δεν σημαίνει πως η διαδρομή που βρέθηκε είναι κατάλληλη γιατί πολλές φορές η πορεία που επιστρέφει είναι υπερβολικά κοντά στο εμπόδιο ή περνά ανάμεσα από εμπόδια όπου το όχημα δεν χωράει. Αυτό έχει ως αποτέλεσμα το όχημα να ακουμπά τον τοίχο και να μην μπορεί να ολοκληρώσει την πορεία του. Για την αποφυγή αυτού του προβλήματος κάθε chunk διατηρεί και ένα λεξικό με την τιμή της ποινής κάθε κελιού. Μια ενημέρωση εμποδίου καλεί επίσης την εντολή addPenalty, η οποία γεμίζει μια κυκλική ζώνη με την αυξανόμενη τιμή της ποινής.

Η μεταβλητή PenaltyRadius έχει τεθεί στα 40 που σημαίνει πως για μια ακτίνα 40 κελιών γύρω από το εμπόδιο θα έχουν μια τιμή ποινής που ξεκινάει από (με baseValue = 100) το εμπόδιο και ανάλογα με την απόσταση από αυτό μειώνεται.

$$\text{Έστω } d = \sqrt{(dx^2 + dy^2)}$$

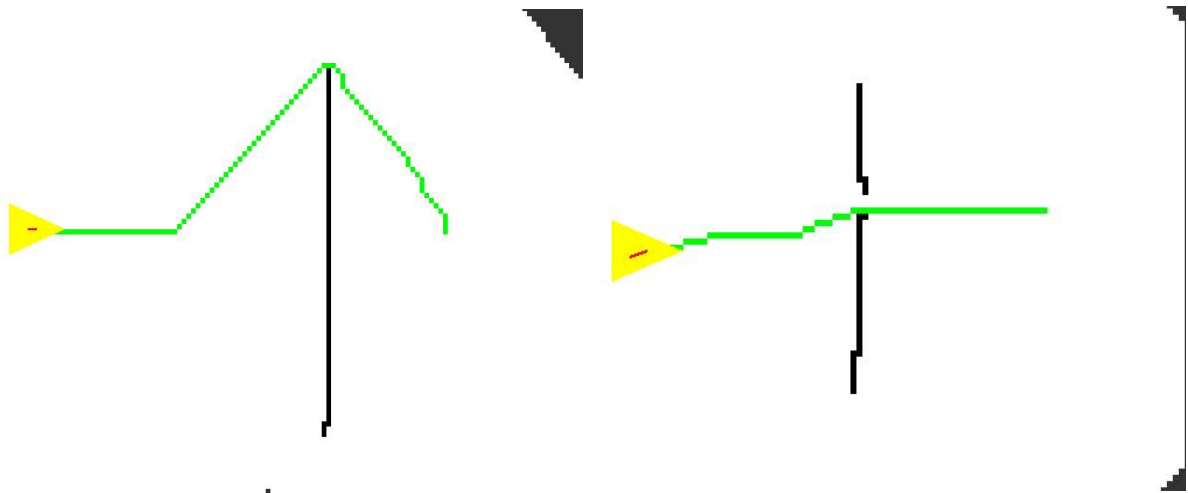
Η απόσταση του κελιού από το εμπόδιο με dx και dy την ακέραια μετατόπιση πάνω στην ακτίνα ο υπολογισμός του εμποδίου γίνεται :

$$p = \text{baseValue}/d.$$

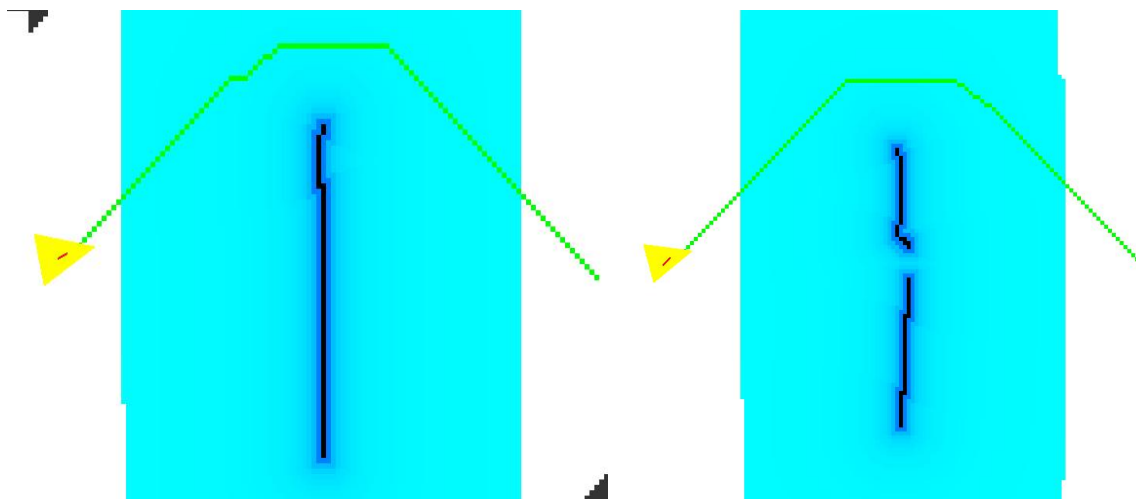
Με d να είναι ένας ακέραιος αριθμός από $1 \leq d \leq R$ με R την ακτίνα. Όταν οι κυκλικές ζώνες ποινής από πολλά κοντινά εμπόδια καλύπτουν το ένα το άλλο μόνο η μέγιστη τιμή της ποινής καταγράφεται και δεν προσθέτεται στην υπάρχουσα τιμή. Η χρήση του μέγιστου εμποδίζει την αύξηση του κόστους αποκλειστικά και μόνο επειδή παρατηρήθηκε ένα εμπόδιο πολλές φορές.

Με την προεπιλεγμένη τιμή baseValue = 100 το κόστος μειώνεται σταδιακά καθώς αυξάνεται η απόσταση.

Ο λόγος που η ακτίνα έχει τεθεί στα 40 είναι επειδή το όχημα έχει μέγεθος 25x35



Εικόνα 6.1 Πορεία οχήματος χωρίς πίνακα ποινών (πριν την εξομάλυνση πορείας)



Εικόνα 6.2 Πορεία οχήματος με πίνακα ποινών (πριν την εξομάλυνση πορείας)

Κεφάλαιο 7ο: Σχεδιασμός διαδρομής και πλοήγησης

Το σύστημα μετατρέπει είτε έναν προορισμό που επιλέγεται από τον χρήστη είτε μια πλήρη διαδρομή που σχεδιάζεται από τον χρήστη σε μια ομαλή διαδρομή με αποφυγή εμποδίων η οποία μπορεί να μεταδοθεί στον ελεγκτή ESP32. Συνδυάζει την αναζήτηση A* που βασίζεται στον χάρτη κελιών και τον χάρτη κόστους, την επικύρωση και επισκευή διαδρομής, την εξομάλυνση του Catmull-Rom και τη συνεχή παρακολούθηση διαδρομής από τους αισθητήρες.

Η υλοποίηση χωρίζεται σε κλάσεις C# και επεκτάσεις Godot C++. Η αλληλεπίδραση χρήστη, η επικύρωση, η οπτικοποίηση, η επικοινωνία και η διαχείριση της κατάστασης του οχήματος υλοποιούνται στην DrawingAppGrid.cs και DrawAppMainGrid.cs. Η αναζήτηση A* υλοποιείται στην κλάση PathFinding.cpp, ενώ η κατασκευή spline και η επισκευή των διαδρόμων υλοποιούνται στο CatmullRom.cpp και οι δύο λειτουργίες σχεδιασμού χρησιμοποιούν τους ίδιους χάρτες

κατακερματισμού και ποινής, επιτρέποντας στην πλοήγηση να ανταποκρίνεται άμεσα στο περιβάλλον που ανακατασκευάζεται από το σύστημα χαρτογράφησης.

7.1 Σχεδιασμός διαδρομής

Υλοποιούνται δύο προσεγγίσεις σχεδιασμού διαδρομής. Μοιράζονται τον ίδιο χάρτη, τον σχεδιαστή A^* , το στοιχείο εξομάλυνσης και το επίπεδο επικοινωνίας ρομπότ, αλλά διαφέρουν ως προς τον έλεγχο που δίνεται στον χρήστη.

7.1.1 Ο σχεδιασμός “Go to”

Επιλέγοντας την επιλογή “go to” στο μενού επιλογής ο χρήστης επιλέγει έναν προορισμό κάνοντας δεξί κλικ στον εμφανιζόμενο χάρτη. Η τοποθεσία στην οθόνη μετατρέπεται σε συντεταγμένες πλέγματος και δεσμεύεται στα τρέχοντα όρια χαρτογράφησης. Ένας προορισμός γίνεται αποδεκτός μόνο όταν η ποινή εμποδίου του είναι μικρότερη του 10. Διαφορετικά εμφανίζεται ένα μήνυμα σφάλματος. Αυτή η συνθήκη διεπαφής χρήστη είναι για να αποφύγει το όχημα να κολλήσει σε ένα εμπόδιο και να μην μπορεί να γυρίσει. Αυτή η προσέγγιση υλοποιείται στην κλάση DrawingAppGrid.

Το σύστημα εκτελεί την ακόλουθη ακολουθία:

- Χρησιμοποιήστε το τρέχον κελί όπου βρίσκεται το όχημα ως κόμβο έναρξης και το κελί στο οποίο έγινε κλικ ως κόμβο στόχου.
- Εξάγονται τα chunks πληρότητας και ποινής στον εγγενή σχεδιαστή διαδρομής.
- Εκτελέστε αναζήτηση A^* για να λάβετε μια διαδρομή πλέγματος χωρίς εμπόδια.
- Περνά τη διαδρομή μέσω την συνάρτηση εξομάλυνσης Catmull-Rom.
- Εξάγονται οι αποστάσεις των σημείων ελέγχου της catmull-Rom και οι γωνίες στροφής ανάμεσα στα σημεία.
- Μεταδίδονται οι αποστάσεις και οι γωνίες στο ESP32 μέσω websocket
- Παρακολουθείται η διαδρομή για νέα εμπόδια και επανασχεδιάζεται όταν είναι απαραίτητο

Αυτή η λειτουργία είναι προσανατολισμένη στον στόχο, ο χρήστης καθορίζει που πρέπει να πάει το όχημα ενώ η εφαρμογή αποφασίζει πως πρέπει να φτάσει σε αυτόν τον στόχο. Είναι κατάλληλη όταν προτιμάται η συντομότερη διαδρομή χαμηλού κόστους και ο χρήστης δεν χρειάζεται να ελέγχει το σχήμα της διαδρομής.

7.1.2 Εύρεση διαδρομής που σχεδιάζεται από τον χρήστη

Η δεύτερη προσέγγιση υλοποιείται στο DrawAppMainGrid. Αντί να επιλέξει μόνο έναν τελικό προορισμό ο χρήστης κρατάει πατημένο το δεξί κουμπί του ποντικιού και σχεδιάζει την επιθυμητή διαδρομή απευθείας στον χάρτη. Η σχεδιασμένη γραμμή χρησιμοποιείται όταν ο χρήστης θέλει πιο άμεσο σχεδιασμό της πορείας του οχήματος. Για παράδειγμα το ρομπότ πρέπει να περάσει από έναν συγκεκριμένο διάδρομο, να πλησιάσει ένα αντικείμενο από τη μία πλευρά ή να αποφύγει μια περιοχή.

Το σχέδιο μετατρέπεται από συντεταγμένες οθόνης σε συντεταγμένες του πλέγματος. Για να αποφευχθεί η αποθήκευση του ιδίου σημείου για κάθε κίνηση του ποντικιού, ένα σημείο γίνεται αποδεκτό μόνο όταν απέχει τουλάχιστον τρία κελιά από το προηγούμενο αποδεκτό σημείο.

Η διαδρομή δεν στέλνεται απευθείας στο ρομπότ. Αρχικά επικυρώνεται σε σχέση με τους χάρτες εμποδίων και ποινής. Τα μη έγκυρα σημεία αναφοράς αφαιρούνται, τα σημεία αυτά είναι όλα τα σημεία που προκύπτουν μέσα σε κελί εμποδίου ή κελιά με τιμή ποινής μεγαλύτερη του 10. Αφότου όλα τα μη έγκυρα σημεία διαγράφονται, καλείται ο αλγόριθμος εύρεσης διαδρομής A* με αρχή το τελευταίο κατάλληλο σημείο και τέλος το επόμενο έγκυρο σημείο που δεν έχει διαγραφεί. Ουσιαστικά ο αλγόριθμος λειτουργεί σαν μια γέφυρα ανάμεσα στα έγκυρα σημεία. Η Catmull-Rom παράγει στη συνέχεια μια ομαλή διαδρομή και τις εντολές τμήματος που απαιτούνται από το ρομπότ. Η επικυρωμένη διαδρομή εμφανίζεται με πράσινο χρώμα και μεταδίδεται μόνο αφού ο χρήστης πατήσει το κουμπί Έναρξη.

7.2 Αλγόριθμος εύρεσης διαδρομής

Ο αλγόριθμος A* είναι ένας αλγόριθμος αναζήτησης γραφημάτων που ιεραρχεί τους κόμβους σύμφωνα με το άθροισμα του γνωστού κόστους από την αρχή και μια εκτίμηση του συνολικού κόστους μέχρι τον στόχο [13]. Το γράφημα κατασκευάζεται έμμεσα από το πλέγμα κατακερματισμού. Κάθε κελί πλέγματος λειτουργεί σαν να είναι ένας κόμβος και οι ακμές συνδέουν το κελί με τους οριζόντιους, κάθετους και διαγώνιους γείτονές του.

7.2.1 Αναζήτηση γειτονικών κόμβων

Το όχημα μπορεί να κινείται προς κάθε κατεύθυνση στον δισδιάστατο χώρο για αυτό όταν ο αλγόριθμος ελέγχει όλα τα σημεία γύρω από τον κόμβο .

```
Vector2i dirs[8] = {
    Vector2i(1,0), Vector2i(1,1), Vector2i(0,1), Vector2i(-1,1),
    Vector2i(-1,0), Vector2i(-1,-1), Vector2i(0,-1), Vector2i(1,-1)
};
```

Οι οριζόντιες και κάθετες κινήσεις έχουν κόστος 10, ενώ οι διαγώνιες κινήσεις έχουν κόστος 14 γιατί η διαγώνια τιμή προσεγγίζει την ευκλείδεια ευρετική τιμή της οριζόντιας/κάθετης κίνησης $10 \cdot \sqrt{2} \sim 14$.

Ένα κατειλημμένο κελί δεν εισάγεται ποτέ ως επόμενο σημείο. Αποτρέπεται επίσης η διαγώνια γωνία αν διέρχεται ανάμεσα από δύο κατειλημμένα κελιά. Πριν από την επέκταση του διαγώνιου κόμβου ο

σχεδιαστής ελέγχει τα δύο βασικά κελιά που είναι δίπλα στη διαγώνιο. Εάν οποιοδήποτε γειτονικό κελί είναι κατειλημμένο η διαγώνια απορρίπτεται.

7.2.2 Λειτουργία αξιολόγησης γειτονικών κόμβων

Για κάθε υποψήφιο επόμενο κόμβο n ο αλγόριθμος A^* αξιολογεί:

$$f(n) = g(n) + h(n)$$

όπου $g(n)$ το κόστος διαδρομής από την αρχή και $h(n)$ είναι το κόστος μέχρι τον στόχο. Στην υλοποίηση το κόστος μετάβασης στο επόμενο κόμβο είναι

$$g(n1) = g(n0) + C_{move}(n0,n1) + P(n1)$$

- όπου $n1$ ο επόμενος κόμβος $n0$ ο τωρινός κόμβος.
- $C_{move}(n0,n1)$ το γεωμετρικό κόστος μετακίνησης σε γειτονικό κόμβο (οριζόντιες και κάθετες κινήσεις 10, ενώ οι διαγώνιες 14).
- $P(n1)$ η ποινή του κόμβου από το χάρτη κόστους.

Έτσι μεταξύ διαδρομών παρόμοιου γεωμετρικού μήκους ο αλγόριθμος προτιμά τη διαδρομή με μεγαλύτερη απόσταση από εμπόδια.

Για να βρούμε την απόσταση από τον κόμβο μέχρι τον στόχο μας χρησιμοποιούμε την ευρετική απόσταση. Για κόμβο (x,y) και στόχο (x_g, y_g) :

$$dx = |x_g - x| \text{ και } dy = |y_g - y|$$

$$D_{max} = \max(dx, dy), \quad d_{min} = \min(dx, dy)$$

$$h(n) = 10 * d_{max} + 4 * d_{min} \Rightarrow 14d_{min} + 10(d_{max} - d_{min})$$

- d_{min} ο αριθμός των διαγώνιων βημάτων
- $d_{max} - d_{min}$ πόσα κάθετα/οριζόντια βήματα έχουν μείνει

Επειδή οι ποινές εμποδίων είναι μη αρνητικές η ευρετική δεν υπερεκτιμά το υπόλοιπο κόστος και επομένως είναι κατάλληλη για αναζήτηση A^* .

7.2.3 Διαδικασία αναζήτησης

Το σετ των κόμβων που ελέγχονται υλοποιείται ως μια ουρά ελάχιστης προτεραιότητας ταξινομημένη με βάση το κόστος $f(n)$ του κόμβου.

```
std::priority_queue<Map, std::vector<Map>, std::greater<Map>>> open;
```

Map είναι η μεταβλητή του κόμβου .

```
struct Map{
    Vector2i pos;
    int goal, cost, startHeading;
    int score() const {
        return goal + cost;
    }
    bool operator>(const Map &other) const {
        return score() > other.score();
    }
    Map(Vector2i p_pos, int p_goal, int p_cost, int p_startHeading) : pos(p_pos), goal(p_goal),
cost(p_cost), startHeading(p_startHeading) {}
    Map(Vector2i p_pos, int p_goal, int p_cost) : pos(p_pos), goal(p_goal), cost(p_cost) {}
};
```

Υπάρχουν άλλοι τρεις πίνακες

```
std::unordered_map<Vector2i, Vector2i> cameFrom;
std::unordered_map<Vector2i, int> cost;
std::unordered_set<Vector2i> closed;
```

- Ο cameFrom που περιέχει τους προηγούμενους κόμβους στην τελική διαδρομή
- Ο cost περιέχει τη χαμηλότερη τιμή που βρέθηκε για κάθε κόμβο
- Ο closed περιέχει κόμβους που έχουν ήδη επεκταθεί.

Ο αρχικός κόμβος εισάγεται με $g=0$. Σε κάθε επανάληψη ο κόμβος με το μικρότερο εκτιμώμενο συνολικό κόστος αφαιρείται από την ουρά. Κάθε νόμιμος γειτονικός κόμβος αξιολογείται και ο προηγούμενος του και το συσσωρευμένο κόστος ενημερώνονται όταν βρεθεί μια φθηνότερη διαδρομή. Η αναζήτηση σταματά όταν ο στόχος αφαιρεθεί για επέκταση, η ανοιχτή ουρά αδειάσει ή άμα ξεπεραστεί το όριο των 100000 επαναλήψεων. Η τελική διαδρομή ανακατασκευάζεται προς τα πίσω από τον στόχο μέσω του cameFrom και επιστρέφεται με σειρά από την αρχή στον στόχο.

7.3 Επικύρωση διαδρομής

Η επικύρωση διαδρομής είναι ιδιαίτερα σημαντική στη λειτουργία σχεδίασης διαδρομής από τον χρήστη επειδή μια γραμμή που σχεδιάστηκε μπορεί να περάσει μέσα από εμπόδια, να τα πλησιάσει πολύ κοντά ή να ξεκινήσει με μια κατεύθυνση που το όχημα δεν μπορεί να ακολουθήσει ομαλά. Το σύστημα εφαρμόζει μια διαδικασία επικύρωσης προσανατολισμένη στην επισκευή αντί να αποδέχεται ή να απορρίπτει απλώς ολόκληρη τη γραμμή.

Η ανεπεξέργαστη διαδρομή πρέπει να περιέχει τουλάχιστον δύο σημεία αναφοράς δειγματοληψίας. Κάθε σημείο αναφοράς w_i ταξινομείται ως μη έγκυρο όταν $M(w_i) = 2$ (εμπόδιο) , $P(w_i) > 10$ στον χάρτη κόστους.

Πρέπει να υπάρχουν τουλάχιστον δύο έγκυρα σημεία αναφοράς. Η τρέχουσα θέση του οχήματος εισάγεται ως το πρώτο σημείο της διορθωμένης διαδρομής, διασφαλίζοντας ότι η τελική διαδρομή ξεκινά από το ρομπότ και όχι από το πρώτο σημείο της σχεδίασης.

7.3.1 Τοπική επισκευή A^*

Ο αλγόριθμος σαρώνει τον πίνακα ταξινόμησης με τη σειρά σχεδίασης. Προστίθενται έγκυρα σημεία ενώ σημεία που εμφανίζονται δύο φορές διαγράφονται. Όταν εντοπιστεί μια μη έγκυρη διαδρομή διακρίνονται δύο περιπτώσεις:

Εάν η μη έγκυρη διαδρομή συμβεί στο τέλος της διαδρομής, αφαιρείται επειδή δεν υπάρχει επόμενο έγκυρο σημείο αναφοράς για επανασύνδεση.

Εάν υπάρχει μεταγενέστερο έγκυρο σημείο αναφοράς ο A^* αναζητά μια διαδρομή από το τελευταίο αποδεκτό σημείο στο επόμενο έγκυρο σημείο αναφοράς.

Το πρώτο σημείο της γέφυρας παραλείπεται επειδή αντιγράφει το προηγούμενο αποδεκτό σημείο ενώ τα υπόλοιπα σημεία γέφυρας προστίθενται με τη σειρά. Η επικύρωση αποτυγχάνει όταν η A^* δεν μπορεί να παράγει μια διαδρομή που να περιέχει τουλάχιστον έναν κόμβο αρχής και έναν κόμβο τέλους.

Αυτή η υλοποίηση τοπικής επισκευής διατηρεί όσο το δυνατόν μεγαλύτερο μέρος της διαδρομής του χρήστη. Μια πλήρης αντικατάσταση από τον A^* θα απέρριπτε τη σχεδιασμένη γεωμετρία και θα μείωνε την δεύτερη διεπαφή, όπου ο χρήστης σχεδιάζει την διαδρομή του, σε έναν άλλο σχεδιαστή τύπου go to. Επιδιορθώνοντας μόνο μη ασφαλή τμήματα η υλοποίηση συνδυάζει την καθοδήγηση του χρήστη με την αυτόματη αποφυγή εμποδίων.

7.3.2 Πύκνωση διαδρομής και προετοιμασία για εξομάλυνση

Μετά την επιδιόρθωση κάθε τμήμα δύο σημείων A,B υποδιαιρείται σύμφωνα με

$$N = \max(1, \left\lceil \frac{d}{L_{max}} \right\rceil)$$

όπου

- d είναι η απόσταση των σημείων A,B.

- $L_{\max} = 3$ και είναι το μέγιστο μήκος.
- $\lceil \cdot \rceil$ στρογγυλοποίηση στον πλησιέστερο μεγαλύτερο ακέραιο αριθμό
- $\text{Max}(1, \cdot)$ Εξασφαλίζει ότι υπάρχει τουλάχιστον 1 τμήμα και αποτρέπει τη διαίρεση με το μηδέν όταν $A=B$
- N είναι ο αριθμός υποδιαίρεσεων που χρειάζεται έτσι ώστε κανένα κομμάτι να μην είναι μεγαλύτερο του $L_{\max} = 3$

Έπειτα δημιουργεί ομοιόμορφα κατανεμημένες παραμέτρους παρεμβολής.

$$ts = \frac{s}{N}, s = 1, 2, \dots, N \text{ με } ts \text{ ανήκει στο } (0, 1]$$

s είναι ο δείκτης ακέραιων βημάτων που ξεκινάει από 1 έως N

Μετά τα σημεία αναφοράς εισάγονται

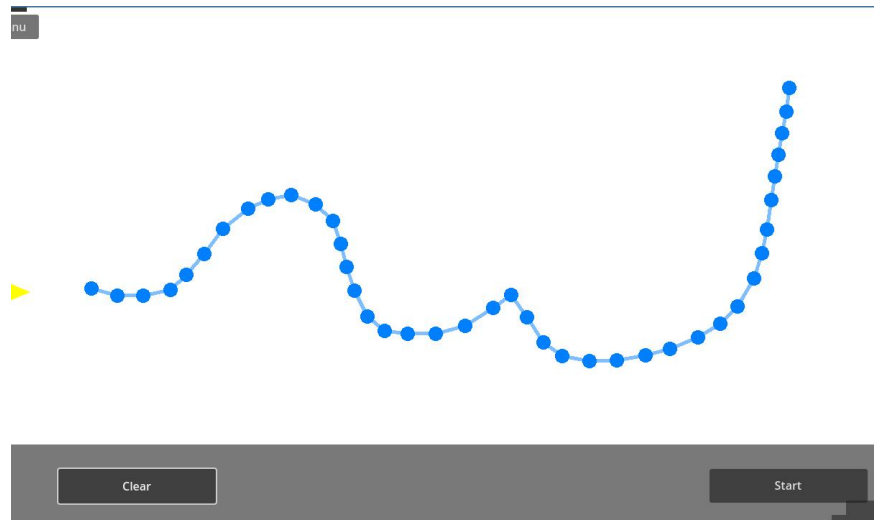
$$P_s = (1 - ts) * A + ts * B$$

Υπολογίζει τις συντεταγμένες του s -οστού εισαγόμενου σημείου χρησιμοποιώντας γραμμική παρεμβολή.

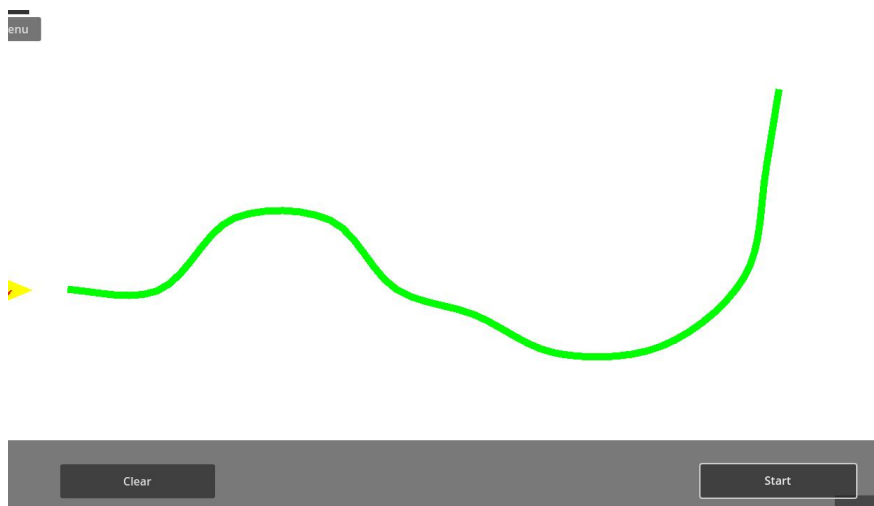
Η πύκνωση βοηθάει τον αλγόριθμο spline να έχει μια οριοθετημένη απόσταση εισόδου και κάνει την έξοδο της λιγότερο ευαίσθητη σε μεγάλα κενά μεταξύ των σημείων αναφοράς.

7.4 Εξομάλυνση διαδρομής με Catmull-Rom

Η διαδρομή που επιστρέφεται από τον αλγόριθμο A^* αποτελείται από οριζόντιες, κάθετες και διαγώνιες κινήσεις του χάρτη κελιών. Μια τέτοια διαδρομή περιέχει απότομες αλλαγές κατεύθυνσης και δεν είναι κατάλληλη για ένα όχημα με περιορισμένο σύστημα διεύθυνσης. Ομοίως μια διαδρομή που σχεδιάζεται από τον χρήστη μπορεί να περιέχει ανομοιόμορφες αποστάσεις και θορυβώδεις τοπικές αλλαγές κατεύθυνσης. Επομένως το έργο χρησιμοποιεί την παρεμβολή Catmull-Rom, μια τοπική παρεμβαλλόμενη spline που εισήχθη αρχικά από τους Catmull και Rom [6]. Σε αντίθεση με μια προσεγγιστική spline μια καμπύλη Catmull-Rom διέρχεται από τα σημεία ελέγχου της αυτό βοηθά στη διατήρηση τόσο της διαδρομής A^* όσο και της διαδρομής που σχεδίασε ο χρήστης. Το σύστημα καλεί την μέθοδο `createSplineWithHeading` η οποία είναι υπεύθυνη για την εξομάλυνση διαδρομής και την διόρθωση γωνίας στροφής.



Εικόνα 7.1 Σχεδίαση πορείας από τον χρήστη πριν την εξομάλυνση πορείας



Εικόνα 7.2 Σχεδίαση πορείας από τον χρήστη μετά την εξομάλυνση πορείας

7.4.1 Περιορισμός αρχικής κατεύθυνσης

Η μέθοδος `createSplineWithHeading`, η οποία είναι η πρώτη μέθοδος που επικαλείται για την εξομάλυνση διαδρομής χρησιμοποιεί την τρέχουσα κατεύθυνση του οχήματος `startHeadingDeg` και μιας διαμορφωμένη απόσταση `d=20` για να κατασκευάσει ένα αρχικό διάνυσμα πριν από την αρχή της διαδρομής:

```
double headingRad = (double)startHeadingDeg * M_PI / 180.0;
vector2f headingDir = { std::cos(headingRad), std::sin(headingRad) };
vector2f phantom = {
    carPoint.x - headingDir.x * controlDistance,
    carPoint.y - headingDir.y * controlDistance
}
```

```
};
vector2f phantom0 = {
    phantom.x - headingDir.x * controlDistance ,
    phantom.y - headingDir.y * controlDistance
};
```

Χωρίς αυτό το διάνυσμα η πρώτη εφαπτομένη της καμπύλης θα καθοριζόταν μόνο από τον χρήστη ή τα σημεία A^* και θα μπορούσε να απαιτήσει μια άμεση στροφή που δεν γίνεται με τον τρέχοντα προσανατολισμό του οχήματος. Αυτή η προσέγγιση προτιμήθηκε αντί για τη γεωμετρική προσέγγιση του Dubins path που θα έβαζε άσκοπα μεγάλη περιπλοκότητα καθώς ο αλγόριθμος ήδη έχει σύστημα διόρθωσης γωνιάς.

7.4.2 Υλοποίηση της centripetal Catmull-Rom

Η υλοποίηση C++ χρησιμοποιεί τέσσερα διαδοχικά σημεία : p_0, p_1, p_2, p_3

και κατασκευάζει την καμπύλη μεταξύ p_1 και p_2 . Οι τιμές κόμβου του είναι:

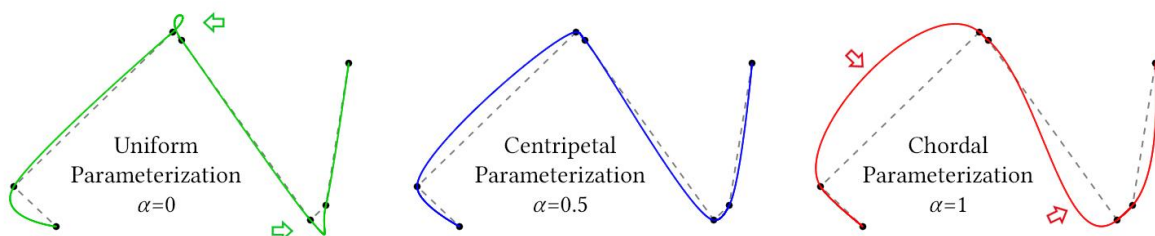
```
float t0 = 0.0f;
float t1 = getT( t0, spl.p0, spl.p1 , alpha);
float t2 = getT( t1, spl.p1, spl.p2 , alpha);
float t3 = getT( t2, spl.p2, spl.p3 , alpha);
```

Με $\alpha = 0.5$

```
inline float getT(float t, const vector2f& p0, const vector2f& p1, float alpha) {
    float a = std::pow(distance(p0, p1), alpha);
    return a + t;
}
```

Η `distance()` είναι η ευκλείδεια απόσταση των δύο σημείων.

Επιλέγεται $\alpha = 0.5$ (Centripetal) γιατί τα διαστήματα των κόμβων αυξάνονται ανάλογα με την τετραγωνική ρίζα της ευκλείδειας απόστασης. Αυτό αποτρέπει την υπερβολή όταν τα σημεία ελέγχου είναι άνισα κατανομημένα.[5]



Εικόνα 7.3 Οι διάφοροι τρόποι παραμετροποίησης Catmull-Rom spline

Για t στο εύρος $[t_1, t_2]$ η υλοποίηση αξιολογεί την spline αναδρομικά:

```
t = lerp( t1, t2, t );
double lerp(double a, double b, double t) {
    return a + t * (b - a);
}
```

Στο πρώτο επίπεδο για κάθε ζευγάρι διπλανών σημείων πραγματοποιεί γραμμική παρεμβολή :

```
vector2f A1 =
    {( t1-t )/( t1-t0 )*spl.p0.x + ( t-t0 )/( t1-t0 )*spl.p1.x ,
      ( t1-t )/( t1-t0 )*spl.p0.y + ( t-t0 )/( t1-t0 )*spl.p1.y };

vector2f A2 =
    {( t2-t )/( t2-t1 )*spl.p1.x + ( t-t1 )/( t2-t1 )*spl.p2.x ,
      ( t2-t )/( t2-t1 )*spl.p1.y + ( t-t1 )/( t2-t1 )*spl.p2.y };

vector2f A3 =
    {( t3-t )/( t3-t2 )*spl.p2.x + ( t-t2 )/( t3-t2 )*spl.p3.x ,
      ( t3-t )/( t3-t2 )*spl.p2.y + ( t-t2 )/( t3-t2 )*spl.p3.y };
```

$A1 = (p_0, p_1)$ $A2 = (p_1, p_2)$ $A3 = (p_2, p_3)$

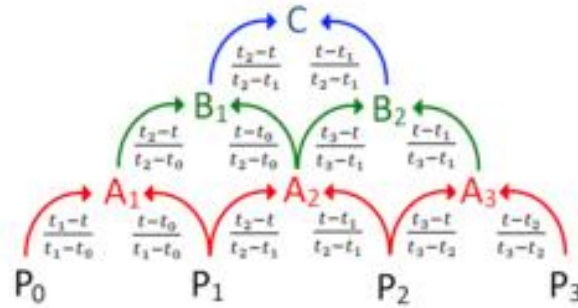
Στο δεύτερο επίπεδο συνδυάζει τα σημεία του πρώτου επιπέδου σε διαστήματα κόμβων από t_0 σε t_2 και t_1 σε t_3

```
vector2f B1 =
    {( t2-t )/( t2-t0 )*A1.x + ( t-t0 )/( t2-t0 )*A2.x ,
      ( t2-t )/( t2-t0 )*A1.y + ( t-t0 )/( t2-t0 )*A2.y };

vector2f B2 =
    {( t3-t )/( t3-t1 )*A2.x + ( t-t1 )/( t3-t1 )*A3.x ,
      ( t3-t )/( t3-t1 )*A2.y + ( t-t1 )/( t3-t1 )*A3.y };
```

Και στο τρίτο επίπεδο εκτελεί γραμμική παρεμβολή στο διάστημα t_1, t_2 παράγοντας το ακριβές σημείο της καμπύλης

```
vector2f C =
    {( t2-t )/( t2-t1 )*B1.x + ( t-t1 )/( t2-t1 )*B2.x ,
      ( t2-t )/( t2-t1 )*B1.y + ( t-t1 )/( t2-t1 )*B2.y };
```



Εικόνα 7.4 Η πυραμδική διατύπωση των Barry και Goldman

Επομένως:

- Πρώτο επίπεδο (A1, A2, A3) εκτελεί μια γραμμική παρεμβολή μεταξύ γειτονικών σημείων ελέγχου.
- Δεύτερο επίπεδο (B1, B2) συνδυάζει τα σημεία του πρώτου επιπέδου σε διαστήματα κόμβων t_0-t_2 και t_1-t_3 . Αυτό το βήμα ενσωματώνει έμμεσα τις εφαπτόμενες στα σημεία ελέγχου P1 και P2 με βάση την εφαπτόμενη των άλλων σημείων ελέγχου P0 και P3.
- Τρίτο επίπεδο εκτελεί μια τελική γραμμική παρεμβολή μεταξύ των σημείων B1 και B2 που βρίσκονται πάνω στην εφαπτόμενη κατά του μήκους του διαστήματος $[t_1, t_2]$ παράγοντας τη συντεταγμένη του spline point.

7.4.3 Δειγματοληψία καμπύλης

Για κάθε spline τεσσάρων σημείων ο αριθμός των δειγμάτων βασίζεται στην απόσταση μεταξύ των δύο κεντρικών σημείων:

```
density = floor(distance(pathPoints[i+1], pathPoints[i+2])/2.0f);
if(density < 1)
    density = 1;
splinePoint spl = createSplinePoint(pathPoints[i], pathPoints[i+1], pathPoints[i+2],
pathPoints[i+3]);
drawSpline(spl, density);
```

Επομένως, τα μεγαλύτερα τμήματα λαμβάνουν περισσότερα δείγματα από τα μικρά τμήματα. Αυτό παράγει μια πυκνότερη γραμμή όπου η καμπύλη καλύπτει μεγαλύτερη απόσταση χάρτη, αποφεύγοντας άσκοπα μεγάλο αριθμό δειγμάτων για κάθε τμήμα.

7.4.4 Διόρθωση γωνίας στροφής

Αφού έχει δημιουργηθεί η spline η καμπύλη επαναδειγματίζεται με σημεία ελέγχου σε απόσταση περίπου 15 κελιών το ένα από το άλλο. Για τρία διαδοχικά σημεία επαναδειγματισμού τα εισερχόμενα και εξερχόμενα διανύσματα είναι

```
vector2f from = {samplePathPoints[i].x - samplePathPoints[i-1].x,
                  samplePathPoints[i].y - samplePathPoints[i-1].y};
vector2f to   = {samplePathPoints[i+1].x - samplePathPoints[i].x,
                  samplePathPoints[i+1].y - samplePathPoints[i].y};
```

Και για να βρει την γωνία ανάμεσα στα δύο αυτά σημεία:

```
float angle = findAngle(from, to);
double findAngle(const vector2f& from, const vector2f& to) {
    double magn1 = std::sqrt(from.x * from.x + from.y * from.y);
    double magn2 = std::sqrt(to.x * to.x + to.y * to.y);
    const double EPS = 1e-9;
    if (magn1 < EPS || magn2 < EPS)
        return 0.0;
    double dot = from.x * to.x + from.y * to.y;
    double cosAngle = dot / (magn1 * magn2);
    cosAngle = std::max(-1.0, std::min(1.0, cosAngle));
    double angle = std::acos(cosAngle);
    return angle * (180.0 / M_PI);
}
```

αρχικά βρίσκει το μέγεθος των διανυσμάτων

```
double magn1 = std::sqrt(from.x * from.x + from.y * from.y);
double magn2 = std::sqrt(to.x * to.x + to.y * to.y);
```

Μετά ελέγχει για τυχόν διαίρεση με μηδέν χρησιμοποιώντας το EPS που είναι ένας πολύ μικρός αριθμός

```
const double EPS = 1e-9;
if (magn1 < EPS || magn2 < EPS)
    return 0.0;
```

Στη συνέχεια βρίσκει το εσωτερικό γινόμενο και λύνει για το συνημίτονο

```
double dot = from.x * to.x + from.y * to.y;
double cosAngle = dot / (magn1 * magn2);
```

Για την αποφυγή λαθών στην στρογγυλοποίηση, μειώνει το εύρος σε

```
cosAngle = std::max(-1.0, std::min(1.0, cosAngle));
```

Και τέλος μετατρέπει σε μοίρες

```
double angle = std::acos(cosAngle);
return angle * (180.0 / M_PI);
```

Αμα η γωνία που επιστρέφει είναι μεγαλύτερη των 45 μοιρών (η μέγιστη γωνία που μπορεί να στρίψει το όχημα) το σύστημα ελέγχει την κατεύθυνση της γωνίας άμα είναι δεξιόστροφη ή αριστερόστροφη.

```
bool checkClockwise(const vector2f& from, const vector2f& to) {
    double cross = from.x * to.y - from.y * to.x;
    return cross < 0;
}
```

Και δημιουργεί ένα νέο βέλτιστο διάνυσμα προς αυτήν κατεύθυνση.

```
vector2f createOptimalVectorClock(const vector2f& vector){
    vector2f vector2;
    vector2.x = (std::sqrt(2)/2) * vector.x - (std::sqrt(2)/2) * vector.y;
    vector2.y = (std::sqrt(2)/2) * vector.x + (std::sqrt(2)/2) * vector.y;
    return vector2;
}
```

Τα διορθωμένα σημεία περνούν από τη γεννήτρια spline για δεύτερη φορά. Η διαδικασία περιορίζει τις μεγάλες αλλαγές κατεύθυνσης μεταξύ των τμημάτων ελέγχου που έχουν υποστεί νέα δειγματοληψία.

Τέλος, επιστρέφεται ένας πίνακας για κάθε σημείου ελέγχου πλοήγησης με τιμές {x, y, z} όπου x, y οι συντεταγμένες του σημείου και z η γωνία του.

7.5 Πλοήγηση

Το στάδιο πλοήγησης μετατρέπει την σχεδιασμένη γεωμετρική διαδρομή σε μια διαδρομή που μπορεί να εκτελέσει το φυσικό όχημα. Η ίδια διαδρομή χρησιμοποιείται επίσης από την εφαρμογή Godot για οπτικοποίηση και παρακολούθηση της κατάστασης της κίνησης.

Ενώ μια διαδρομή είναι ενεργή, η εφαρμογή ελέγχει έναν αριθμό επερχόμενων σημείων spline. Η μεταβλητή LookAhead = 45 δείχνει πόσα σημεία ελέγχου μπροστά από το τωρινό σημείο του

οχήματος ελέγχει για εμπόδια. Μια διαδρομή θεωρείται αποκλεισμένη όταν ένα επερχόμενο σημείο είναι κατειλημμένο ή έχει ποινή χάρτη κόστους μεγαλύτερη από 10.

Όταν εντοπιστεί εμπόδιο ο στόχος διατηρείται, η κίνηση διακόπτεται προσωρινά και δημιουργείται μια νέα διαδρομή A^* από την τρέχουσα τοποθεσία του οχήματος. Ένας χρόνος αναμονής 0,5 δευτερολέπτων περιορίζει τη συχνότητα ενεργοποίησης του επανασχεδιασμού γιατί μπορεί να εμφανιστούν πολλά εμπόδια στο οπτικό πεδίο του οχήματος και κάθε ένα από αυτά θα καλούσε επανεξέταση. Η διαδρομή εξομαλύνεται, μετατρέπεται σε εντολές ελέγχου και μεταδίδεται στο ρομπότ.

Κεφάλαιο 8ο: Συμπεράσματα και προτάσεις βελτίωσης

8.1 Μετρήσεις και αποτελέσματα

Για την τελική αξιολόγηση εξέτασα τις μετρήσεις απόστασης από τον αισθητήρα TOF400C και δοκίμασα την οδομετρία του οχήματος σε πραγματικό φυσικό χώρο.

8.1.1 Μετρήσεις του αισθητήρα TOF400C

Για τις μετρήσεις χρησιμοποίησα έναν απλό αλγόριθμο που κάθε 1000ms επιστρέφει την τιμή του μέσω σειριακής επικοινωνίας. Επειδή ο αισθητήρας που χρησιμοποιείται για την χαρτογράφηση έχει τεθεί στο μέγιστο 1300mm οι μετρήσεις που χρησιμοποίησα είναι μέχρι τα 1300mm.

Τιμή ελέγχου	Δείγματα	Μέσος όρος	Λάθος	Έγκυρες επιστροφές
100mm	10	104.6	4.6	100%
500mm	10	505	5	100%
1000mm	10	1002.2	2.2	100%
1300mm	10	1308.7	8.7	100%

Πίνακας 8.1 Μετρήσεις αισθητήρα TOF400C

Για αυτές τις τιμές έλεγχου η τυπική απόκλιση σύμφωνα με τον κατασκευαστή είναι σε $\pm 20\text{mm}$ για εσωτερικούς χώρους. Για μετρήσεις μεγαλύτερης εμβέλειας, έως 4000mm υπάρχει μεγαλύτερος κίνδυνος να επιστρέφει άκυρες τιμές ή να μην επιστρέφει καθόλου καθώς και ο χρόνος απόκρισης

αυξάνεται σε τουλάχιστον >140ms κάτι που το κάνει ακατάλληλο για μια εφαρμογή που χρειάζεται γρήγορες μετρήσεις.

8.1.2 Μετρήσεις οδομετρίας

Για τις μετρήσεις στελνόταν μέσω της διεπαφής οι επιθυμητές αποστάσεις και στροφές. Η μέτρηση ήταν η απόσταση που διένυσε το μπροστινό μέρος του οχήματος. Το όχημα έχει τροχούς με περιφέρεια 21,3 εκ. και έχουν τοποθετηθεί 2 χαρτιά σε κάθε μεριά του τροχού. Επομένως για μια πλήρη περιστροφή πρέπει να περάσουν δύο αλλαγές κατάστασης από χαρτί σε όχι χαρτί.

Άρα ισχύει πως $21,3/2 = 10,65$ εκ είναι η απόσταση που κινείται το όχημα πριν γίνει η αλλαγή κατάστασης. Ο αλγόριθμος Catmull-Rom επιστρέφει σημεία ελέγχου για κάθε 15εκ και με ταχύτητα 33,4 εκ/δευτ. Προκύπτει αυτός ο υποθετικός πίνακας.

Απόσταση σε εκατοστά	Απαιτούμενες αλλαγές κατάστασης	Υποθετική απόσταση	Περισσευούμενη απόσταση	Ποσοστό λάθους %
10	1	10,65	0,65	6,5
15	2	21,30	6,3	42
20	2	21,30	1,3	6,5
50	5	53,25	3,25	6,5
100	10	106,5	6,5	6,5
200	19	202,35	2,35	1,18

Πίνακας 8.2 Υποθετικές μετρήσεις οδομετρίας

Ωστόσο στην πραγματική χρήση στον φυσικό χώρο προκύπτει :

Απόσταση σε εκατοστά	Απαιτούμενες αλλαγές κατάστασης	Πραγματική απόσταση	Περισευούμενη απόσταση	Ποσοστό λάθους %
10	1	14	4	40
15	2	30	15	100
20	2	30	10	50
50	5	68	18	36
100	10	118	18	18
200	19	229	29	14,5

Πίνακας 8.3 Πραγματικές μετρήσεις οδομετρίας

Από τον πίνακα προκύπτει ότι οι μικρότερες αποστάσεις έχουν τον μεγαλύτερο απόλυτο σφάλμα. Το πραγματικό τελικό σημείο εξαρτάται από τη φάση του τροχού κατά την έναρξη ενός τμήματος, την απόσταση ακινητοποίησης του κινητήρα, την ολίσθηση του τροχού, το πλάτος του χαρτιού. Στις μικρότερες αποστάσεις το όχημα έχει μεγαλύτερη ώθηση κατά την εκκίνηση με αποτέλεσμα να έχει μεγαλύτερο σφάλμα λόγω ολίσθησης.

8.2 Συμπεράσματα

Ο στόχος μου με αυτό το έργο ήταν η ανάπτυξη μιας ολοκληρωμένης πλατφόρμας ικανής να κατασκευάσει έναν χάρτη σε ένα άγνωστο περιβάλλον και να χρησιμοποιήσει αυτόν τον χάρτη για αυτόνομη πλοήγηση. Το υλοποιημένο όχημα συνδυάζει ενσωματωμένη ανίχνευση και έλεγχο με μια γραφική και υπολογιστική εφαρμογή βασισμένη στο Godot.

Στο πλαίσιο της εργασίας δόθηκε έμφαση στη χαρτογράφηση του περιβάλλοντος με βάση εκτιμώμενη θέση του οχήματος καθώς και στον σχεδιασμό της διαδρομής από τον χρήστη και στην αυτόνομη πορεία του οχήματος κατά μήκος της. Η προσέγγιση αυτή αποτελεί μια ισχυρή βάση για τη μελλοντική ανάπτυξη ενός ολοκληρωμένου συστήματος SLAM στο οποίο η θέση και ο προσανατολισμός του οχήματος θα μπορούν να εκτιμώνται και να διορθώνονται ταυτόχρονα μέσω παρατηρήσεων από τους αισθητήρες όπως ένας LiDAR.

8.3 Τρόποι βελτίωσης

Πάντα υπάρχει περιθώριο βελτίωσης, μια από αυτές θα μπορούσε να ήταν η χρήση ανώτερων εξαρτημάτων όπως αισθητήρων LiDAR ή κάποιου αισθητήρα Hall effect για την καλύτερη υλοποίηση οδομετρίας. Μια ακόμα βελτίωση θα ήταν η υλοποίηση ενός πραγματικού συστήματος ταυτόχρονου εντοπισμού θέσης με σάρωση κάτι το οποίο θα χρειαζόταν καλύτερο εξοπλισμό και υπολογιστική

δύναμη. Ο χάρτης πληρότητας θα μπορούσε να αντικατασταθεί από ένα πλέγμα πληρότητας με λογαριθμικές πιθανότητες. Αν τα κελιά παρατηρούνταν επανειλημμένα ως ελεύθερα ή κατειλημμένα θα αύξαναν ή θα μείωναν την εμπιστοσύνη αντί να αντικαθιστούν αμέσως μια κατάσταση.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

- [1] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, MIT Press, 2005.
- [2]Jonathan, Vogel "Tech Explained: Ackermann Steering Geometry". Racecar Engineering. Retrieved January 24, 2025.
- [3]H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," IEEE Robotics & Automation Magazine, vol. 13, no. 2, pp. 99–110, 2006.
- [4]H. P. Moravec and A. E. Elfes, "High Resolution Maps from Wide Angle Sonar," Proceedings of the IEEE International Conference on Robotics and Automation, pp. 116–121, 1985.
- [5]E. Catmull and R. Rom, A Class of Local Interpolating Splines, in Computer Aided Geometric Design, 1974.
- [6]"Using a PID-based Technique For Competitive Odometry and Dead-Reckoning". Seattle Robotics. 2016.
- [7]T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," Robotics and Autonomous Systems, 2016,
- [13]P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

Data Sheet

- [8]Espressif Systems, ESP32 Series Datasheet, Espressif Systems.
- [9]STMicroelectronics, L298 Dual Full-Bridge Driver, Datasheet DS0218.
- [10]TowerPro, SG90 Analog Servo.
- [11]STMicroelectronics, "VL53L1X: Time-of-Flight ranging sensor based on FlightSense technology."2024.
- [12]XLSEMI, "5A 180KHz 36V buck DC-DC converter," XL4015 datasheet

Κώδικας

- [14] https://github.com/panostp/My_thesis.git

ΠΑΡΑΡΤΗΜΑ Α : ΚΩΔΙΚΑΣ ESP32

```
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <Wire.h>
#include <VL53L1X.h>
#include <ArduinoJson.h>
#include <WiFiUdp.h>

//web
const char* ssid = "LAPTOPTTEST";
const char* password = "12345678";
WebSocketsServer webSocket = WebSocketsServer(81);
unsigned long lastSend = 0;
const unsigned long sendInterval = 50;
bool discoveryEnabled = true;

//UDP search
WiFiUDP udp;
const int UDP_PORT = 4210;
//UDP search-
//web-
#define SDA 21
#define SCL 22
#define SERVO_PIN 27
#define PAN_SERVO_PIN 14
bool connected = false;
//ToF
const uint8_t XSHUT_A = 33;
const uint8_t XSHUT_B = 32;
const uint8_t ADDR_A = 0x2A;
const uint8_t ADDR_B = 0x2B;
VL53L1X sensorA;
VL53L1X sensorB;
const int MOUNT_ANGLE_A = -30;
const int MAX_VALID_RANGE_MM = 4000;
int lastDistA = -1;
int lastDistB = -1;
//ToF==
//===== WHEEL ODOMETRY =====
const int SLITS_PER_REV = 2;
```

```

const float WHEEL_CIRCUMFERENCE_CM = 21.3f;
const int SLIT_FAR_MM = 65;
const int SLIT_NEAR_MM = 60;
bool inSlit = false;
volatile unsigned long slitCount = 0;

```

Ενημερώνει τον μετρητή για την αλλαγή κατάστασης του τροχού με βάση τις μετρήσεις του αισθητήρα TOF400C

```

void updateSlitCounter()
{
    if (!sensorB.dataReady())
        return;
    int dist = sensorB.read(false);
    if (sensorB.ranging_data.range_status != 0)
        return;
    if (!inSlit && dist >= SLIT_FAR_MM)
    {
        inSlit = true;
        slitCount++;
    }
    else if (inSlit && dist <= SLIT_NEAR_MM)
    {
        inSlit = false;
    }
}

```

Υπολογίζει τη συνολική απόσταση που έχει διανύσει το όχημα σε εκατοστά

```

float getDistanceTraveledCm()
{
    return (slitCount / (float)SLITS_PER_REV) * WHEEL_CIRCUMFERENCE_CM;
}

//===== WHEEL ODOMETRY =====
//===== SERVOS =====

int currentSteerAngle = 0;
int currentPanAngle = 0;

const int STEER_DUTY_MIN = 5100;
const int STEER_DUTY_CENTER = 6200;
const int STEER_DUTY_MAX = 7300;
const int PAN_CENTER = 90;
const int PAN_MIN = 0;
const int PAN_MAX = 180;

```

```
//=====
int panDirection = 1;
unsigned long lastPanUpdate = 0;
const unsigned long PAN_UPDATE_INTERVAL_MS = 30;
const int PAN_STEP_DEG = 2;
//=====
//===== MOTOR DRIVER =====
#define IN1 4
#define IN2 5
#define IN3 16
#define IN4 17
#define MOTOR_SPEED 200
//=====
//===== PATH QUEUE =====
struct PathPoint {
    float distance;
    float angle;
};
const int ObstacleStopDis = 200;
const int ObstacleLimit = 15;
bool obstacleStopped = false;
const int MAX_PATH_POINTS = 64;
PathPoint pathQueue[MAX_PATH_POINTS];
int pathQueueCount = 0;
int pathQueueIndex = 0;
enum DriveState {IDLE, DRIVING};
DriveState driveState = IDLE;
unsigned long segmentStartTime = 0;
unsigned long segmentDurationMs = 0;
float segmentStartDistanceCm = 0;
const float STEERING_SMOOTH_SPEED = 160.0f;
const float CM_PER_SECOND = 33.4f;
//=====
```

Αρχικοποιεί τον αισθητήρα TOF και ρυθμίζει την διεύθυνση και τις παραμέτρους λειτουργίας του. Ο πρώτος χρησιμοποιείται για χαρτογράφηση και ο δεύτερος για οδομετρία.

```
void initTofSensor(VL53L1X &sensor, uint8_t newAddress, unsigned long timingBudgetUs, unsigned long
continuousPeriodMs, bool isB = false)
{
    sensor.setTimeout(500);
    if (!sensor.init())
```

```

{
    Serial.print("Failed to init TOF sensor at address ");
    Serial.println(newAddress, HEX);
    while (1) delay(10);
}

sensor.setAddress(newAddress);
sensor.setDistanceMode(VL53L1X::Short);
if(isB)
    sensor.setROISize(4, 4);
sensor.setMeasurementTimingBudget(timingBudgetUs);
sensor.startContinuous(continuousPeriodMs);
}

```

Εκκινεί διαδοχικά τους δύο αισθητήρες TOF και τους αποδίδει διαφορετικές διευθύνσεις I2C

```

void setupTofSensors()
{
    pinMode(XSHUT_A, OUTPUT);
    pinMode(XSHUT_B, OUTPUT);
    digitalWrite(XSHUT_A, LOW);
    digitalWrite(XSHUT_B, LOW);
    delay(10);
    digitalWrite(XSHUT_A, HIGH);
    delay(10);
    initTofSensor(sensorA, ADDR_A, 33000, 33);
    digitalWrite(XSHUT_B, HIGH);
    delay(10);
    initTofSensor(sensorB, ADDR_B, 20000, 20, true);
    Serial.println("Both TOF sensors initialized.");
}

```

Διαβάζει και επιστρέφει μια έγκυρη μέτρηση απόστασης από τον επιλεγμένο αισθητήρα TOF

```

int readValidDistance(VL53L1X &sensor, bool SensorB=false) {
    if (!sensor.dataReady())
        return -2;
    int dist = sensor.read(false);
    uint8_t status = sensor.ranging_data.range_status;
    if (status != 0) return -1;
    if (dist <= 0 || dist > MAX_VALID_RANGE_MM )
        return -1;
    return dist;
}

```

Μετατρέπει την επιθυμητή γωνία διεύθυνσης στην κατάλληλη τιμή PWM για τον σερβοκινητήρα.

```
void writeServoAngle(int pin, int angle, int dutyMin, int dutyCenter, int dutyMax) {  
    angle = constrain(angle, -45, 45);  
    int duty;  
    if (angle >= 0) {  
        duty = map(angle, 0, 45, dutyCenter, dutyMax);  
    } else {  
        duty = map(angle, -45, 0, dutyMin, dutyCenter);  
    }  
    ledcWrite(pin, duty);  
}
```

Ορίζει τη γωνία του σέρβο διεύθυνσης μέσα στα επιτρεπτά όρια.

```
void servoWriteAngle(int angle) {  
    currentSteerAngle = constrain(angle, -45, 45);  
    int corrected = currentSteerAngle;  
    corrected = constrain(corrected, -45, 45);  
    writeServoAngle(SERVO_PIN, corrected, STEER_DUTY_MIN, STEER_DUTY_CENTER, STEER_DUTY_MAX);  
}
```

Ορίζει τη γωνία του σέρβο που περιστρέφει τον αισθητήρα απόστασης.

```
void panServoWriteAngle(int relativeAngle)  
{  
    relativeAngle = constrain(relativeAngle, -90, 90);  
    int physicalAngle = PAN_CENTER + relativeAngle;  
    physicalAngle = constrain(physicalAngle, PAN_MIN, PAN_MAX);  
    int pulseWidth = map(physicalAngle, 0, 180, 500, 2500);  
    int duty = (pulseWidth * 65535L) / 20000L;  
    ledcWrite(PAN_SERVO_PIN, duty);  
    currentPanAngle = relativeAngle;  
}
```

Ενημερώνει περιοδικά τη κίνηση του σέρβο ανάλογα με την κατάσταση κίνησης του οχήματος.

```
void updatePanSweep()  
{  
    unsigned long now = millis();  
    if (now - lastPanUpdate < PAN_UPDATE_INTERVAL_MS)  
        return;  
    lastPanUpdate = now;  
    int sweepLimit = (driveState == DRIVING) ? 15 : 45;  
    int nextOffset = currentPanAngle + PAN_STEP_DEG * panDirection;
```

```

    if (nextOffset >= sweepLimit)
    {
        nextOffset = sweepLimit;
        panDirection = -1;
    }
    else if (nextOffset <= -sweepLimit)
    {
        nextOffset = -sweepLimit;
        panDirection = 1;
    }
    int physicalAngle = -(currentSteerAngle + nextOffset);
    physicalAngle = constrain(physicalAngle, -90, 90);
    int servoAngle = PAN_CENTER + physicalAngle;
    int pulseWidth = map(servoAngle, 0, 180, 500, 2500);
    int duty = (pulseWidth * 65535L) / 20000L;
    ledcWrite(PAN_SERVO_PIN, duty);
    currentPanAngle = nextOffset;
}
//===== MOTOR CONTROL =====

```

Σταματάει το όχημα

```

void motorStop() {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
}

```

Ξεκινάει το όχημα

```

void motorForward(int speed) {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH);
    digitalWrite(IN4, LOW);
}

```

Ενημερώνει την εφαρμογή μέσω Websocket για την ολοκλήρωση ενός τμήματος της διαδρομής.

```

void sendSegmentComplete(int completedIndex, bool pathComplete)
{
    String json = "{";
    json += "\"type\":\"segment_complete\",";
    json += "\"index\":" + String(completedIndex) + ",";
}

```

```

    json += "\"pathComplete\": \"" + String(pathComplete ? "true" : "false");
    json += "\"";
    websocket.broadcastTXT(json);
}

```

Προετοιμάζει και ξεκινά την εκτέλεση του επόμενου τμήματος της διαδρομής.

```

void startNextPathPoint()
{
    if (pathQueueIndex >= pathQueueCount)
    {
        driveState = IDLE;
        motorStop();
        Serial.println("Path complete.");
        return;
    }
    PathPoint &pt = pathQueue[pathQueueIndex];
    Serial.printf(
        "Segment %d: distance=%.2f cm, target angle=%.2f\n",
        pathQueueIndex,
        pt.distance,
        pt.angle
    );
    segmentStartDistanceCm = getDistanceTraveledCm();
    segmentDurationMs =
        (unsigned long)((pt.distance / CM_PER_SECOND) * 1000.0f);
    segmentStartTime = millis();
    driveState = DRIVING;
    motorForward(MOTOR_SPEED);
}

```

Ελέγχει την πορεία του οχήματος, την αποφυγή και την ολοκλήρωση του τρέχοντος τμήματος.

```

void updateDriving()
{
    if (driveState != DRIVING)
        return;
    bool sensorLookingForward = abs(currentPanAngle) <= ObstacleLimit;
    if (sensorLookingForward && lastDistA > 0 && lastDistA <= ObstacleStopDis)
    {
        emergencyStop();
        return;
    }
}

```

```

if (pathQueueIndex >= pathQueueCount)
{
    driveState = IDLE;
    motorStop();
    return;
}
PathPoint &pt = pathQueue[pathQueueIndex];
updateSteering(pt.angle);
float traveledThisSegment = getDistanceTraveledCm() - segmentStartDistanceCm;
bool distanceReached = traveledThisSegment >= pt.distance;
bool stalled = (millis() - segmentStartTime) >= (segmentDurationMs * 3);
if (distanceReached || stalled)
{
    if (stalled && !distanceReached)
        Serial.println("Segment timed out ");
    int completedIndex = pathQueueIndex;
    pathQueueIndex++;
    bool pathComplete = (pathQueueIndex >= pathQueueCount);
    sendSegmentComplete(completedIndex, pathComplete);
    if (pathComplete)
    {
        driveState = IDLE;
        motorStop();
        Serial.println("Path compl");
        return;
    }
    startNextPathPoint();
}
}

```

Εκτελεί διακοπή πορείας όταν εντοπιστεί εμπόδιο μπροστά σε απόσταση λιγότερη των 20 εκ.

```

void emergencyStop()
{
    motorStop();
    driveState = IDLE;
    obstacleStopped = true;
    Serial.println("OBSTACLE! Car stopped");
    websocket.broadcastTXT("{\"type\":\"emergency_stop\"}");
}
unsigned long lastSteeringUpdate = 0;

```

Προσαρμόζει τη γωνία διεύθυνσης προς την επιθυμητή τιμή για ομαλότερη κίνηση.

```
void updateSteering(float targetAngle)
{
    unsigned long now = millis();
    if (lastSteeringUpdate == 0)
    {
        lastSteeringUpdate = now;
        return;
    }
    float dt = (now - lastSteeringUpdate) / 1000.0f;
    lastSteeringUpdate = now;
    targetAngle = constrain(targetAngle, -40.0f, 40.0f);
    float difference = targetAngle - currentSteerAngle;
    float maxStep = STEERING_SMOOTH_SPEED * dt;
    if (abs(difference) <= maxStep)
    {
        servoWriteAngle((int)targetAngle);
    }
    else if (difference > 0)
    {
        servoWriteAngle(currentSteerAngle + (int)maxStep);
    }
    else
    {
        servoWriteAngle(currentSteerAngle - (int)maxStep);
    }
}
```

Στέλνει αίτημα εντοπισμού μέσω UDP και αποστέλλει τη διεύθυνση IP του ESP32 στην εφαρμογή.

```
void ConnectUDP(){
    int packetSize = udp.parsePacket();
    if (packetSize)
    {
        char packet[256];
        int len = udp.read(packet, sizeof(packet) - 1);
        if (len > 0)
            packet[len] = '\0';
        if (strcmp(packet, "DISCOVER") == 0)
        {
            String response = "ESP32_CAR|" + WiFi.localIP().toString();
        }
    }
}
```

```

        udp.beginPacket(udp.remoteIP(),udp.remotePort());

        udp.print(response);

        udp.endPacket();

        Serial.println("Discovery response");
    }
}
}

```

Επεξεργάζεται τη διαδρομή που λαμβάνεται σε μορφή JSON και αποθηκεύει τα σημεία της στην ουρά.

```

void handlePathMessage(JsonDocument &doc) {
    JsonArray points = doc["points"].as<JsonArray>();
    motorStop();
    driveState = IDLE;
    obstacleStopped = false;
    pathQueueCount = 0;
    for (JsonVariant p : points)
    {
        if (pathQueueCount >= MAX_PATH_POINTS)
        {
            Serial.println("Path too long ");
            break;
        }
        JsonArray pair = p.as<JsonArray>();
        if (pair.size() < 2) continue;
        pathQueue[pathQueueCount].distance = pair[0].as<float>();
        pathQueue[pathQueueCount].angle = pair[1].as<float>();
        pathQueueCount++;
    }
    pathQueueIndex = 0;
    if (pathQueueCount > 0)
    {
        startNextPathPoint();
    }
}

```

Αποκωδικοποιεί τα εισερχόμενα μηνύματα JSON και προωθεί τις εντολές διαδρομής για επεξεργασία.

```

void Movement(uint8_t * payload, size_t length) {
    JsonDocument doc;

    DeserializationError err = deserializeJson(doc, payload, length);

    if (err) {

```

```

    Serial.print("JSON parse failed: ");
    Serial.println(err.c_str());
    return;
}

const char* type = doc["type"];
if (type == nullptr) {
    return;
}
if (strcmp(type, "path") == 0) {
    handlePathMessage(doc);
}
else {
    Serial.printf("Unknown message type: %s\n", type);
}
}

```

// Διαχειρίζεται την σύνδεσης, αποσύνδεσης και λήψης δεδομένων του WebSocket.

```

void websocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length)
{
    switch (type)
    {
        case WStype_DISCONNECTED:
            Serial.printf("[%u] Disconnected\n", num);
            discoveryEnabled=true;
            break;
        case WStype_CONNECTED:
            {
                IPAddress ip = websocket.remoteIP(num);
                Serial.printf("[%u] Connected from %d.%d.%d.%d\n", num, ip[0], ip[1], ip[2], ip[3]);
                connected = true;
                discoveryEnabled=false;
            }
            break;
        case WStype_TEXT:
            Serial.printf("[%u] Received: %s\n", num, payload);
            Movement(payload, length);
            break;
        default:
            break;
    }
}

```

Στέλνει στην εφαρμογή τη γωνία σάρωσης και την μέτρηση απόστασης του αισθητήρα.

```
void sendSensorData() {  
    int totalAngleA =currentSteerAngle + currentPanAngle;  
    String json = "{";  
    json += "\"type\":\"sensor_data\",";  
    json += "\"angleA\":\"" + String(totalAngleA) + ",";  
    json += "\"distanceA\":\"" + String(lastDistA);  
    json += "\"}";  
    websocket.broadcastTXT(json);  
}
```

Αρχικοποιεί τη σειριακή επικοινωνία, το δίκτυο, τους κινητήρες, τους σέρβο και τους αισθητήρες.

```
void setup()  
{  
    Serial.begin(115200);  
    //motor driver  
    pinMode(IN1, OUTPUT);  
    pinMode(IN2, OUTPUT);  
    pinMode(IN3, OUTPUT);  
    pinMode(IN4, OUTPUT);  
    motorStop();  
    //websocket  
    WiFi.begin(ssid, password);  
    while (WiFi.status() != WL_CONNECTED)  
    {  
        delay(500);  
        Serial.print("-");  
    }  
    Serial.println();  
    Serial.print("connected with IP : \n");  
    Serial.println(WiFi.localIP());  
    websocket.begin();  
    websocket.onEvent(webSocketEvent);  
    //servos  
    ledcAttach(SERVO_PIN, 50, 16);  
    ledcAttach(PAN_SERVO_PIN, 50, 16);  
    servoWriteAngle(0);  
    panServoWriteAngle(0);  
    connected = true;  
    //ToF
```

```

Wire.begin(SDA, SCL);
setupTofSensors();
udp.begin(UDP_PORT);
Serial.println("UDP discovery started");
}
void loop()
{
  websocket.loop();
  if(discoveryEnabled)
    ConnectUDP();
  updateDriving();
  updatePanSweep();
  int distA = readValidDistance(sensorA);
  if (distA != -2)
    lastDistA = distA;
  updateSlitCounter();
  unsigned long now = millis();
  if (now - lastSend >= sendInterval)
  {
    lastSend = now;
    sendSensorData();
  }
}

```