# ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

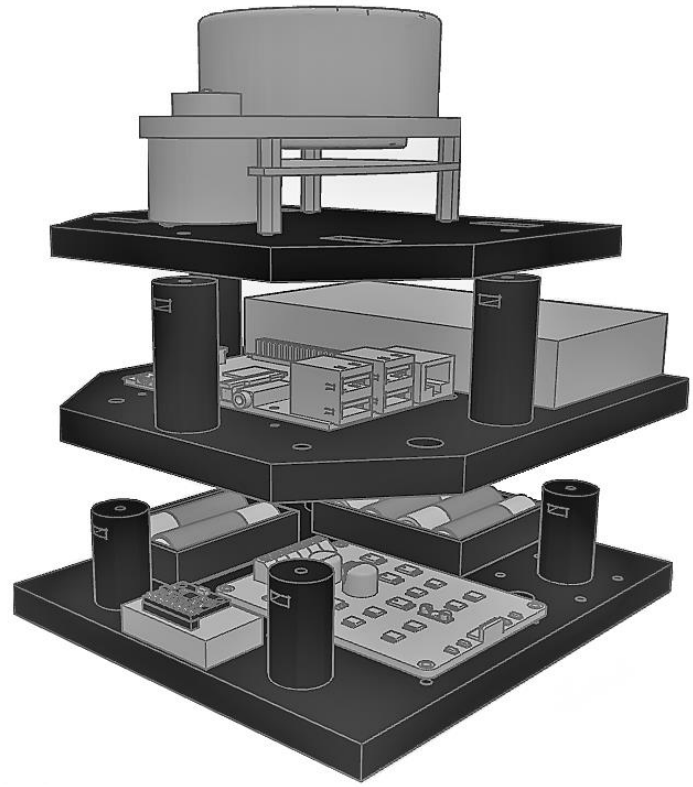
ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ


**Τεχνικές αυτόνομης πλοήγησης και χαρτογράφησης χώρου για ρομποτικές εφαρμογές**

Πτυχιακή εργασία


**Ελένη Νικόλα**


Αθήνα, 2022

# HAROKOPIO UNIVERSITY

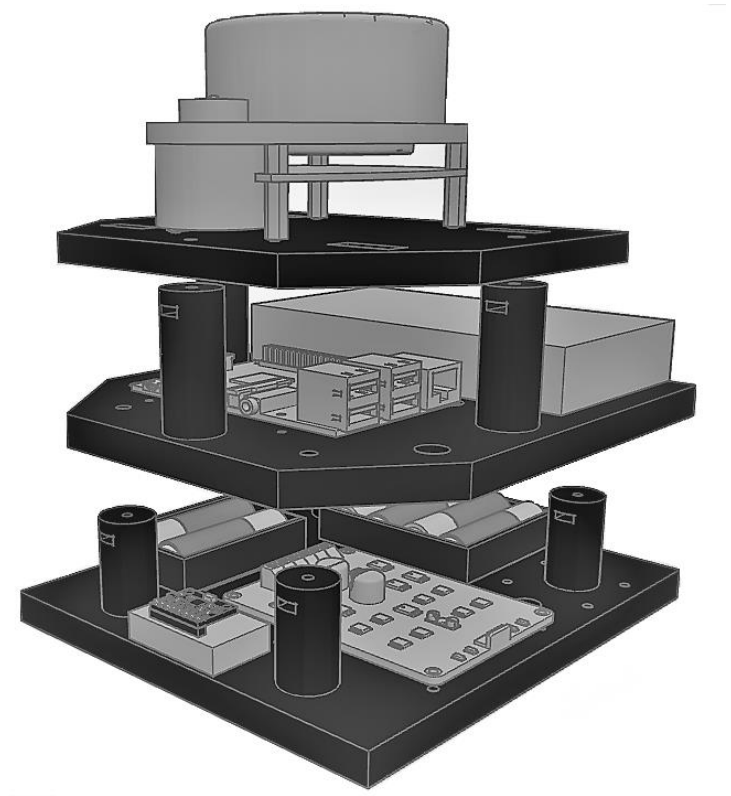
SCHOOL OF DIGITAL TECHNOLOGY

DEPARTMENT OF INFORMATICS AND TELEMATICS


**Self-navigation and mapping techniques for robotic applications**

Bachelor thesis


**Eleni Nikolla**


Athens, 2022

# ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

**Τριμελής Εξεταστική Επιτροπή**

**Θωμάς Καμαλάκης**
**Καθηγητής και Κοσμήτορας του Τμήματος, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο**

**Σωτήριος Ξύδης**
**Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο**

**Παναγιώτης Ριζομυλιώτης**
**Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο**

Η Ελένη Νικόλα

δηλώνω υπεύθυνα ότι:

**1)** Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

**2)** Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

**3)** Όπου υφίστανται δικαιώματα άλλων δημιουργών έχουν διασφαλιστεί όλες οι αναγκαίες άδειες χρήσης ενώ το αντίστοιχο υλικό είναι ευδιάκριτο στην υποβληθείσα εργασία.

# ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Θωμά Καμαλάκη, για την πολύτιμη καθοδήγηση και βοήθειά του, καθώς και για την άμεση ανταπόκριση του καθ' όλη τη διάρκεια υλοποίησης της πτυχιακής μου μελέτης.

Επιπλέον θα ήθελα να ευχαριστήσω τον κ. Σωτήριο Ξύδη και τον κ. Παναγιώτη Ριζομυλιώτη που δέχθηκαν να είναι μέλη της τριμελούς επιτροπής αξιολόγησης της πτυχιακής μου εργασίας.

Ευχαριστώ επίσης θερμά την οικογένεια μου και τους φίλους μου, και ιδιαίτερα τον Νίκο Πετραλίδη, την Κωνσταντίνα Σκρίκα και τον Κωνσταντίνο Παπανικολάου για την αδιάκοπη στήριξη και ενθάρρυνσή τους στην περίοδο συγγραφής της μελέτης.

Τέλος θα ήθελα να ευχαριστήσω τον συνάδελφο και υπεύθυνο μου, Elwin van der Hall, καθώς μου έδωσε τον απαραίτητο χρόνο για να εστιάσω και να εκπληρώσω την πτυχιακή μου εργασία.

## ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

# Περίληψη στα Ελληνικά

Αρχικά, στην παρακάτω έκθεση θα γίνει μια θεωρητική εισαγωγή και μελέτη της λειτουργίας των αλγορίθμων αυτόνομης χαρτογράφησης και εντοπισμού θέσης SLAM, του ρομποτικού συστήματος ROS2, της θεωρίας συγχώνευσης δεδομένων αισθητήρων σε συνδυασμό με το διευρυμένο φίλτρο Kalman και τις τεχνικές για την επίλυση του προβλήματος αυτόνομης εξερεύνησης βάσει της εύρεσης συνόρων.
Στη συνέχεια θα αναλυθούν διεξοδικά ο σχεδιασμός και τα βήματα που ακολουθήθηκαν για την υλοποίηση του συστήματος διαχωρίζοντας το στις εξής δύο κατηγορίες:

1. Στην κατασκευή του οχήματος, δηλαδή τον σχεδιασμό του σκελετού, την επιλογή των εξαρτημάτων, το είδος των αισθητήρων και των ενεργειακών αναγκών της ρομποτικής πλατφόρμας.
2. Στην αρχιτεκτονική του λογισμικού του συστήματος. Στο συγκεκριμένο κεφάλαιο γίνεται εκτενή αναφορά στον τρόπο χρήσης και ενσωμάτωσης του ROS2 λειτουργικού συστήματος, της διαχείρισης των αισθητήρων, των δεδομένων τους και του συνδυασμού τους για τον εντοπισμό της θέσης του συστήματος. Εξετάζονται οι αρχές της οδομετρίας και της διαφορικής κινηματικής καθώς και η μαθηματική ανάλυσή τους για τον έλεγχο της κίνησης του οχήματος. Επιπλέον μελετώνται οι αλγόριθμοι που χρησιμοποιήθηκαν, ο τρόπος επιλογής τους και η παραμετροποίηση τους για την κάλυψη των αναγκών της παρούσας εργασίας και την επιτυχή ενσωμάτωσης τους στο σύστημα.

Στα τελευταία κεφάλαια, αναφέρονται παραδείγματα χρήσης του ρομποτικού οχήματος σε διάφορους εσωτερικούς χώρους και των χαρτών που επιτυχώς παράχθηκαν. Μαζί με τα παραδείγματα θα εξεταστούν και οι αποδόσεις του συστήματος, οι δυσκολίες και τα προβλήματα που εντοπίστηκαν κατά την υλοποίηση της εργασίας καθώς και μελλοντικές μετατροπές και λύσεις που μπορούν να εφαρμοστούν για να βελτιωθεί ή να επεκταθεί το έργο.

**Λέξεις κλειδιά:** Ρομποτική, SLAM, ROS2, Εξερεύνηση Συνόρων

# Summary in English

The purpose of this thesis is to review the state of the art of technologies and scientific methods for the implementation of an autonomous, differential, small-scale vehicle. The goal of this robotic platform is to independently explore, navigate and dynamically map unknown interior spaces while using only its sensor readings. The following report will present a theoretical introduction and investigation on the available SLAM (autonomous mapping and positioning) algorithms, the robotic operating system (ROS2), the theory of sensor fusion combined with the extended Kalman filter, and the modern techniques for solving the autonomous exploration problem based on frontier search. Moreover, this paper will explain thoroughly the design and the steps followed to implement the robotic platform by dividing them into the following two categories:

1. The physical aspects of the vehicle, i.e., the design and construction of the system, the selection of the hardware components and sensors as well as the power requirements of the robotic platform.
2. The software architecture of the system. In this chapter an extensive description takes place on the usage and integration of the ROS2 operating system along with sensor control and fusion techniques with the aim of tracking the position and the state of the system. In addition to the above, there is a mathematical analysis of the odometry principles and differential kinematics for the purpose of controlling the vehicle motion. Furthermore, we are examining the algorithms that are being used, how they were selected, integrated, and parameterized to meet the needs of this project.

Finally, examples of the robotic vehicle exploring and mapping different indoor environments are presented along with their generated maps. On top of that in the last chapters the performance of the system, the difficulties, and problems that we encountered are presented along with proposed future modifications and solutions to improve and expand this project.

**Keywords:** Active SLAM, ROS2, Frontier Search

# Κατάλογος Εικόνων

# Κατάλογος Πινάκων

# Συντομογραφίες/Ακρωνύμια

| | |
|---|---|
| SLAM | Simultaneous Localization and Mapping |
| EKF | Extended Kalman Filter |
| ROS2 | Robot Operation System 2 |
| LIDAR | LIght Detection And Ranging of Laser Imaging Detection And Ranging |
| URDF | Unified Robotics Description Format |
| IMU | Inertial Measurement Unit |

# ΚΕΦ.1: ΕΙΣΑΓΩΓΗ

## 1.1. Σκοπός της πτυχιακής

Την τελευταία δεκαετία έχει παρατηρηθεί μια στροφή του τεχνολογικού τομέα προς την αναζήτηση λύσεων για την αυτοματοποίηση απλών καθημερινών εργασιών. Από τις αυτόνομες συσκευές καθαρισμού, τα μη επανδρωμένα εναέρια αεροσκάφη, τα αυτοκινούμενα ηλεκτρικά οχήματα μέχρι και την κατασκευή μικρών αυτό-κυβερνώμενων οχημάτων για τη διαστημική εξερεύνηση, είναι εμφανής η ανάγκη για τη δημιουργία ρομποτικών συστημάτων που μπορούν να εκτελούν με ασφάλεια και ακρίβεια περίπλοκες συνδυαστικές ενέργειες όπως είναι η αυτόνομη πλοήγηση. Ένα ρομποτικό σύστημα για να καταφέρει να πλοηγηθεί σε ένα οποιοδήποτε περιβάλλον εξαρτάται σε μεγάλο βαθμό από τους αισθητήρες του. Πρέπει να μπορεί να εντοπίζει τη θέση του στον χώρο αλλά και τα υπόλοιπα αντικείμενα που το περιτριγυρίζουν. Πρέπει επίσης να μπορεί να επιλέγει τον επόμενο προορισμό του και να σχεδιάζει την ταχύτερη και ασφαλέστερη διαδρομή σε αυτό, ενώ παράλληλα όσο εξερευνάει νέες περιοχές είναι απαραίτητο να ανανεώνει τα υπολογιστικά του μοντέλα και να έχει πάντα επίγνωση του περιβάλλοντος του, και όλα αυτά να τα εκτελεί σε πραγματικό χρόνο. Επομένως η αυτόνομη πλοήγηση αναδεικνύεται σε μία δύσκολη και σύνθετη αποστολή που μπορεί όμως να διαχωριστεί σε μικρότερα προβλήματα, που αφορούν την χαρτογράφηση του χώρου, της εξερεύνησης, εντοπισμού θέσης και πολλών άλλων.

Στόχος της πτυχιακής είναι η μελέτη των σύγχρονων τεχνολογιών και επιστημονικών μεθόδων που αφορούν την αυτόνομη πλοήγηση, εξερεύνηση και χαρτογράφηση του περιβάλλοντος και την πρακτική εφαρμογή τους μέσω της κατασκευής ενός αυτοκινούμενου διαφορικού οχήματος. Το τελικό ρομπότ που κατασκευάζεται σε αυτή την εργασία είναι ικανό να τοποθετείται σε εσωτερικούς χώρους για πρώτη

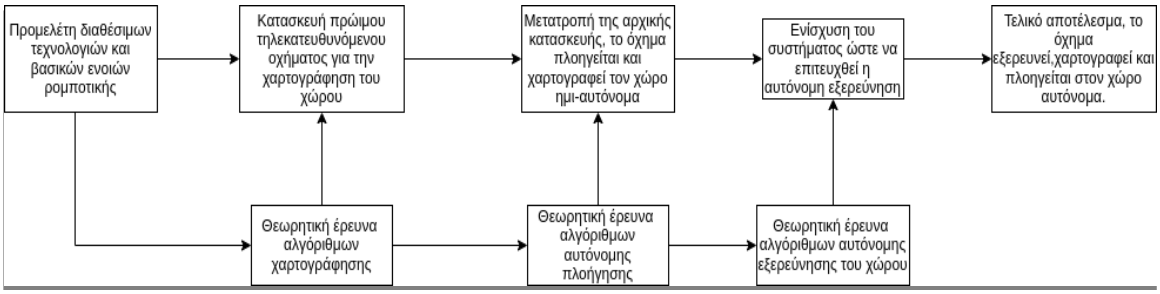
φορά και να πλοηγείται πλήρως αυτόνομα σε αυτούς χαρτογραφώντας ταυτόχρονα τις περιοχές που επισκέπτεται. Πέρα από την υλοποίηση του οχήματος, η παρούσα εργασία έχει στόχο να ερευνήσει και ποιοι σύγχρονοι μέθοδοι και αλγόριθμοι είναι διαθέσιμοι, αν παρέχεται η απαραίτητη υποστήριξη και αν μπορούν να ανταπεξέλθουν στις ραγδαία εξελισσόμενες απαιτήσεις και τεχνολογίες. Τέλος είναι σημαντικό να καταγραφούν τυχόν δυσκολίες και προβλήματα που μπορεί να προκύψουν κατά την υλοποίηση με σκοπό να βοηθήσουν τους μελλοντικούς αναγνώστες να αναπαράγουν ή να βελτιώσουν το συγκεκριμένο έργο.

## 1.2 Μεθοδολογία

Η παρακάτω εικόνα περιγράφει τη μεθοδολογία που χρησιμοποιήθηκε για την υλοποίηση της παρούσας πτυχιακής. Επειδή το αντικείμενο της εργασίας δεν περιορίζεται μόνο στη θεωρητική έρευνα αλλά επεκτείνεται και στην φυσική κατασκευή ενός αυτοκινούμενου ρομπότ, ήταν απαραίτητη η διάσπαση σε μικρότερους εφικτούς στόχους  για την επίτευξη του τελικού αποτελέσματος.

Σε πρώτη φάση ερευνήθηκαν οι βασικές έννοιες και αρχές για την κατασκευή αυτοκινούμενων οχημάτων όπως η κινηματική θεωρία ρομποτικής, το λειτουργικό σύστημα ρομποτικής ROS2 (Robot Operating System), τεχνικές εντοπισμού (localization), η χρήση κωδικοποιητών, γυροσκοπίων, οπτικών ραντάρ, κάμερας και άλλων αισθητήρων. Στη συνέχεια, ξεκίνησε η κατασκευή ενός δοκιμαστικού ρομπότ με πρώτο στόχο το όχημα να κινείται τηλεκατευθυνόμενα, με τον χρήστη να δίνει οδηγίες μέσω του πληκτρολογίου, αλλά να μπορεί να χαρτογραφεί τον χώρο που βρίσκεται. Αφού επιτεύχθηκε η χαρτογράφηση, σαν επόμενο βήμα επεκτείνεται η λειτουργικότητα του συστήματος και τίθεται σαν στόχος το όχημα να μπορεί να πλοηγείται και να χαρτογραφεί τον χώρο με τον χρήστη να επιλέγει μόνο τις συντεταγμένες της τελικής θέσης, καταργώντας έτσι τον μηχανισμό τηλεκατεύθυνσης.

Ταυτόχρονα, το ρομπότ παίρνει την οριστική του μορφή καθώς προστίθεται ο σκελετός και όλα τα απαραίτητα εξαρτήματα. Σε τελική φάση, επιλέγεται ο αλγόριθμος για την εξερεύνηση, που αντικαθιστά τον χρήστη πετυχαίνοντας την αυτόνομη κίνηση του οχήματος, καθώς πλέον ο αλγόριθμος επιλέγει τον προορισμό, τις συντεταγμένες και την κατεύθυνση πλοήγησης.




**Εικόνα 1: Μεθοδολογία Εργασίας**

# ΚΕΦ.2: ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

## 2.1 Λειτουργικό Σύστημα Ρομπότ – ROS2

Κατά την κατασκευή ρομποτικών συστημάτων είναι σύνηθες φαινόμενο να γίνεται χρήση πολλαπλών αισθητήρων και διαφορετικών τεχνολογιών που χρειάζεται να επικοινωνούν και να συνεργάζονται μεταξύ τους. Για να επιτευχθεί αυτό απαιτείται ένα κοινό πρωτόκολλο επικοινωνίας, ένας ενδιάμεσος παράγοντας που να μπορεί να καλύψει το κενό μεταξύ των διαφορετικών μεταβλητών του συστήματος. Αυτό το πρόβλημα καλείται να λύσει το λειτουργικό σύστημα ρομποτικής ROS [1], ένα ενδιάμεσο λογισμικό ανοιχτού κώδικα, που περιέχει πολλαπλές βιβλιοθήκες και εργαλεία σχεδιασμένα για την υλοποίηση, υποστήριξη και την ενίσχυση πολύπλοκων ρομποτικών κατασκευών.

Η ελεύθερη διανομή του κώδικα βοήθησε στη δημιουργία μιας τεράστιας κοινότητας προγραμματιστών που παρέχουν συνεχή υποστήριξη και προσθέτουν επιπλέον λειτουργικότητα, από την πρώτη έκδοση του ROS λογισμικού το 2007 έως και σήμερα. Παρόλες τις δυνατότητες και τις βελτιώσεις που επιδέχθηκε το ROS στην πάροδο του χρόνου, η ραγδαία εξέλιξη των αναγκών στον τομέα της ρομποτικής και η πολυπλοκότητα των εφαρμογών ώθησαν στην αναμόρφωση του λογισμικού, με αποτέλεσμα το 2017 να κοινοποιείται μία νέα και ανώτερη εκδοχή του, το ROS2. Σε αντίθεση με τον προκάτοχό του, το ROS2, επεκτείνει την λειτουργικότητα του σε πολλαπλές πλατφόρμες λειτουργικών συστημάτων όπως Windows, MacOS, Ubuntu, Linux Debian κ.λ.π. και εστιάζει στον έλεγχο του συστήματος σε πραγματικό χρόνο, στην ασφάλεια και στην αποδοτικότητα του λειτουργικού.

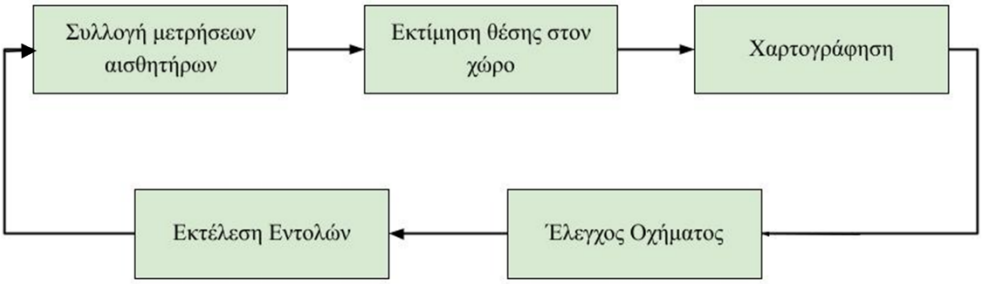
Δυστυχώς, οι μεγάλες διαφορές στη δομή και στη λειτουργικότητα μεταξύ των δύο εκδοχών καθιστούν όλα τα πακέτα λογισμικού που εκδόθηκαν κατά τα πρότυπα του ROS1 ασύμβατα με το ROS2, μια λεπτομέρεια που θα έχει καθοριστικό ρόλο στην επιλογή των τεχνολογιών για την υλοποίηση της εργασίας.

## 2.2 Τεχνικές Χαρτογράφησης και Εντοπισμού Θέσης – SLAM

Φανταστείτε έναν άνθρωπο να τοποθετείται σε ένα πολύπλοκο σκοτεινό λαβύρινθο, με μια άγνωστη αρχική θέση και στόχο να μετακινηθεί μόνος του προς την έξοδο. Χωρίς να γνωρίζει την κατεύθυνση της κίνησής του, τη θέση του σε σχέση με τον προορισμό του, τα πιθανά εμπόδια που μπορεί να βρίσκονται κοντά του αλλά και χωρίς να μπορεί να καταγράφει τις περιοχές που έχει ήδη επισκεφθεί, φαντάζει δύσκολο αν όχι ακατόρθωτο να εκτελέσει την αποστολή του, τουλάχιστον όχι με τον ταχύτερο τρόπο. Ακόμα και αν διέθετε έναν χάρτη του λαβύρινθου, χωρίς να γνωρίζει που βρίσκεται, δεν θα μπορούσε να βρει εύκολα το κοντινότερο μονοπάτι. Θα χρονοτριβούσε δοκιμάζοντας διαφορετικούς διαδρόμους μέχρι να αναγνωρίσει τη θέση του στον χάρτη, αυξάνοντας τον χρόνο και την υπολογιστική πολυπλοκότητα του τελικού μονοπατιού. Την ίδια δυσκολία θα αντιμετώπιζε αν ήξερε την αρχική του θέση σε σχέση με την έξοδο, αλλά δεν διέθετε τον χάρτη. Το παραπάνω σενάριο μπορεί να φαντάζει απίθανο, ωστόσο χρησιμοποιείται για να τονίσει το σύνολο των διαφορετικών πληροφοριών που χρησιμοποιούμε για να μετακινηθούμε από το ένα μέρος στο άλλο. Πολλές φορές δεν συνειδητοποιούμε την υπολογιστική πρόκληση που εμπεριέχουν ακόμα και οι απλές μετακινήσεις, καθώς ο συνδυασμός και η συλλογή των πληροφοριών από το περιβάλλον μας, γίνεται σχεδόν αβίαστα.

Όμοια με τον άνθρωπο στο παράδειγμά, είναι απαραίτητο για κάθε ρομποτικό σύστημα προκειμένου να μετακινηθεί στον χώρο, να γνωρίζει ανά πάσα στιγμή το κοντινό του περιβάλλον και την τοποθεσία του. Επιπλέον είναι αναγκαία και η

ικανότητα του να ανανεώνει και να επεκτείνει τον χάρτη του περιβάλλοντος του όσο κινείται και εξερευνά νέες περιοχές μέσα σε αυτό. Ως **SLAM (Simultaneous Localization And Mapping)** ονομάζουμε το υπολογιστικό πρόβλημα της κατασκευής ή της ενημέρωσης ενός χάρτη άγνωστου περιβάλλοντος, με τον ταυτόχρονο υπολογισμό της τοποθεσίας του πράκτορα μέσα σε αυτό. Η θέση του συστήματος εκφράζεται στο τρισδιάστατο καρτεσιανό σύστημα (άξονες x, y, z) και ο προσανατολισμός του σε Quaternion (τετραγωνικές) ή σε Euler γωνίες.




**Εικόνα 2: Ροή ενεργειών ενός αυτόνομου SLAM συστήματος**

Η δυσκολία του SLAM εντοπίζεται στο παράδοξο ότι για να δημιουργηθεί ένας χάρτης του περιβάλλοντος το σύστημα πρέπει να γνωρίζει την τοποθεσία του, και για να γνωρίζει την τοποθεσία του πρέπει να διαθέτει τον χάρτη του περιβάλλοντος. Επιπλέον, τα δεδομένα των αισθητήρων σε πραγματικές συνθήκες συνήθως εμπεριέχουν θόρυβο και ο όγκος των παραγόμενων μετρήσεων τους μπορεί να εξαντλήσει τις υπολογιστικές δυνατότητες του συστήματος. Ευτυχώς, οι διαρκείς έρευνες γύρω από την υλοποίηση SLAM συστημάτων έχουν αποδώσει πολυάριθμους αλγορίθμους που αναλαμβάνουν την επίλυση του συγκεκριμένου προβλήματος με διάφορες εφαρμογές στην καθημερινότητα, από τη δημιουργία αυτόματων καθαριστικών μηχανημάτων έως και τη διαστημική εξερεύνηση.

## 2.2.1 Μαθηματικό Υπόβαθρο

Όπως αναφέρθηκε στο παραπάνω κεφάλαιο, κατά τη διάρκεια πλοήγησης SLAM συστημάτων, η θέση του οχήματος υπολογίζεται ταυτόχρονα με τη χαρτογράφηση του περιβάλλοντος του. Οι φυσικές παρεμβολές και ο θόρυβος που μπορεί να δημιουργηθεί στα δεδομένα των αισθητήρων δυσκολεύουν τη διαδικασία της οδομετρίας και υπολογισμού της ακριβής θέσης οδηγώντας σε εκτιμήσεις. Το SLAM μπορεί να θεωρηθεί κατά κύριο λόγο, ένα πολύπλοκο μαθηματικό πρόβλημα και σε αυτό το κεφάλαιο θα εξεταστούν οι μεταβλητές που το αποτελούν. Σαν δεδομένα μπορούμε να θεωρήσουμε τις μετρήσεις των αισθητήρων του συστήματος και τις εντολές ελέγχου ενώ σαν ζητούμενα, τον χάρτη του περιβάλλοντος και την κατάσταση (state) του οχήματος. Εάν θεωρήσουμε πως $x_t$ είναι η θέση του οχήματος, $u_t$ οι εντολές ελέγχου και $z_t$ οι μετρήσεις των αισθητήρων σε διάστημα χρόνου t τότε μπορούμε να εκφράσουμε μαθηματικά τις μεταβλητές ως [2]:

$$\mathrm{X\_\{t\} = \{x\_\{0\}, x\_\{1\}, x\_\{2\}, x\_\{3\}, ..., x\_\{t\}}$$
$$\mathrm{U\_\{t\} = \{u\_\{0\}, u\_\{1\}, u\_\{2\}, u\_\{3\}, ..., u\_\{t\}}$$
$$\mathrm{Z\_\{t\} = \{z\_\{0\}, z\_\{1\}, z\_\{2\}, z\_\{3\}, ..., z\_\{t\}}$$

Οι εντολές ελέγχου $u_t$ και οι μετρήσεις $z_t$ διαφέρουν ανάλογα με τις προδιαγραφές του κάθε συστήματος. Στην παρούσα εργασία όπως θα αναλυθεί και στα επόμενα κεφάλαια, οι εντολές ελέγχου αποτελούνται από το διάνυσμα κατάστασης (state vector) του οχήματος, δηλαδή την περιγραφή της θέσης του οχήματος στον χώρο καθώς και τον προσανατολισμό της κίνησης του (x, y, θ). Οι μετρήσεις Z προέρχονται από τα δεδομένα των αισθητήρων, στην συγκεκριμένα αποτελούνται από δείγματα του οπτικού ραντάρ LIDAR σε συνδυασμό με τις μετρήσεις του γυροσκοπίου/επιταχυνσιόμετρου. Λόγω της αβεβαιότητας των μετρήσεων οι αλγόριθμοι SLAM εστιάζουν στην πιθανολογική προσέγγιση της θέσης του οχήματος. Ο θόρυβος που εισάγουν τα δεδομένα είναι συχνά μη Gaussian και οι μεταβάσεις των μετρήσεων δεν ακολουθούν κάποιο γραμμικό μοντέλο. Η

πιθανοτική κατανομή που εκφράζει το πρόβλημα αβεβαιότητας μπορεί να εκφραστεί ως [3]:

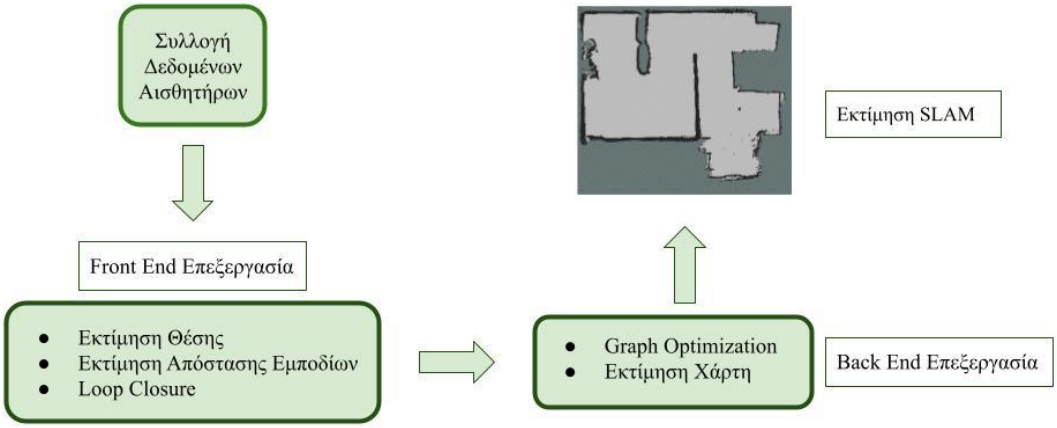
$$\mathrm{p}(s_{0:T}\ , \mathrm{m} \mid (z_{1:T}, u_{1:T})$$

Όπου $s_{0:T}$ εκφράζει την κατάσταση του συστήματος και m είναι ο εκτιμώμενος χάρτης. Η εκτίμηση της θέσης του ρομπότ και της θέσης των ορόσημων τη χρονική στιγμή (t, t-1), θα ενημερωθεί όταν το ρομπότ μετακινηθεί σε νέα θέση τη χρονική στιγμή t+1 με βάση τη νέα εκτίμηση. Αυτή η διαδικασία συνεχίζεται μέχρι το ρομπότ να ολοκληρώσει την εξερεύνηση του άγνωστου περιβάλλοντος.

## 2.2.2. Αρχιτεκτονική SLAM Αλγορίθμων

Η αρχιτεκτονική των SLAM αλγορίθμων μπορεί να διαχωριστεί σε διάφορα στάδια, ανάλογα με το είδος των μετρήσεων που δίνονται σαν είσοδο και με τον τρόπο επεξεργασίας τους. Παρόλες τις διαφορετικές υλοποιήσεις ανάμεσα τους, όλοι οι αλγόριθμοι βασίζονται σε πανομοιότυπη αρχιτεκτονική που παραμετροποιείται ανάλογα με τις ανάγκες τους, επιτρέποντας την ταξινόμησή της σε δύο μεγάλες γενικές κατηγορίες [4], frontend και backend μεθόδους.

Το frontend τμήμα είναι κυρίως υπεύθυνο για την επεξεργασία των δεδομένων που συλλέγονται από τους αισθητήρες του συστήματος. Ανάλογα με το είδος των αισθητήρων που χρησιμοποιούν οι αλγόριθμοι μπορούν να διαχωριστούν σε δύο μεγάλες κατηγορίες, στους οπτικούς (Visual) SLAM και σε αυτούς των οπτικών ραντάρ (LIDAR) SLAM.




**Εικόνα 3: Γενικευμένη αρχιτεκτονική SLAM αλγόριθμου**

Οι οπτικοί (Visual) SLAM ή αλλιώς vSLAM αλγόριθμοι χρησιμοποιούν οπτικούς αισθητήρες όπως κάμερες για να καταγράψουν το περιβάλλον τους. Αναγνωρίζουν συγκεκριμένα σημεία στον χώρο (tracking points) και συγκρίνοντας τα διαδοχικά καρέ που αποτυπώνονται κατά τη διάρκεια της κίνησης υπολογίζουν την τρισδιάστατη θέση τους στο περιβάλλον. Μπορούν να χρησιμοποιηθούν

διαφορετικά είδη κάμερας όπως στερεοφωνικές, RGB-D ακόμα και απλές διαδικτυακές κάμερες.

Για τα συστήματα που επανδρώνονται μόνο με μία κάμερα, εφαρμόζεται συγκεκριμένη κατηγορία αλγορίθμων, οι μονόφθαλμοι (Monocular) SLAM. Οι κάμερες έχουν περιορισμένο κάδρο όρασης συγκριτικά με τα περιστρεφόμενα οπτικά ραντάρ. Επιπλέον, είναι αρκετά δύσκολο να ερμηνευθεί το βάθος των αντικειμένων στην εικόνα, ώστε να υπολογιστεί η απόσταση των εμποδίων από το όχημα, καθώς χρειάζονται τουλάχιστον δύο οπτικές γωνίες. Σαν λύση, μπορούν να χρησιμοποιηθούν συγκεκριμένα σήματα στον χώρο όπως τα AR markers ώστε το όχημα να αναγνωρίζει πάντα ένα σταθερό σημείο (landmark) ή να χρησιμοποιηθούν στερεοφωνικές κάμερες. Οι στερεοφωνικές κάμερες ωστόσο προσθέτουν υπολογιστική πολυπλοκότητα στο σύστημα καθώς σε κάθε στιγμιότυπο εικόνας πρέπει να αντιστοιχηθούν και να συγχωνευθούν τα δύο διαφορετικά καρέ. Εκτενέστερη ανάλυση για τον τρόπο λειτουργίας των οπτικών SLAM αλγόριθμών και των αποδόσεων τους μπορούν να βρεθούν στην αναλυτική έρευνα των M. Barros, A., Michel, M., Moline, Y., Corre, G., & Carrel.[5]

Η δεύτερη κατηγορία αλγορίθμων, αυτή των οπτικών ραντάρ (LIDAR) SLAM, όπως προδίδει και το όνομά της, χρησιμοποιεί οπτικά ραντάρ ως βασικούς αισθητήρες για την χαρτογράφηση του χώρου. Οι τιμές εξόδου συνήθως είναι δεδομένα νέφους είτε δισδιάστατων σημείων 2D(x, y) είτε τρισδιάστατων 3D (x, y, z). Ο υψηλός ρυθμός καταγραφής δεδομένων καθιστά τα οπτικά ραντάρ ιδανικά για την χρήση τους σε συστήματα υψηλής ταχύτητας, όπως μη επανδρωμένα αεροσκάφη ή αυτοκινούμενα οχήματα. Τα περισσότερα οπτικά ραντάρ περιστρέφονται γύρω από τον άξονα τους και εκτοξεύουν ριπές λέιζερ από το κέντρο τους. Χρονομετρώντας τον χρόνο απόκρισης μπορούν να υπολογίσουν την απόσταση των αντικειμένων στον χώρο, η οποία δεν επηρεάζεται από τις συνθήκες φωτισμού σε αντίθεση με τις κάμερες, και διαθέτουν μεγαλύτερο εύρος και υψηλότερη ακρίβεια. Για

περισσότερες λεπτομέρειες σχετικά με τον τρόπο λειτουργίας των LIDAR SLAM αλγορίθμων μπορούν να βρεθούν στις αναφορές [6][7][8] και [9].

Η backend αρχιτεκτονική είναι υπεύθυνη για την κατασκευή του χάρτη και τον υπολογισμό της σχετικής κατάστασης του οχήματος στον χώρο. Χρησιμοποιεί τα δεδομένα που συλλέγονται από τα μοντέλα μέτρησης και κίνησης και συνήθως λειτουργεί συνδυαστικά με τις frontend μεθόδους χωρίς να είναι πάντα αναγκαίο. Επίσης οι backend μέθοδοι μπορούν να χωριστούν σε τρεις υποκατηγορίες ανάλογα με τον τρόπο επεξεργασία των δεδομένων[9].Στις μεθόδους που χρησιμοποιούν φίλτρα όπως το φίλτρο σωματιδίων (particle filters). Στις μεθόδους βελτιστοποίησης γράφου πόζας (pose graph optimization) όπου όλα τα δεδομένα που συλλέγονται από τους αισθητήρες, αποθηκεύονται με μορφή κορυφών γράφου. Εξαιτίας αυτής της ιδιαιτερότητας, οι συγκεκριμένοι αλγόριθμοι δεν μπορούν να εκτιμήσουν τους χάρτες σε πραγματικό χρόνο. Η τρίτη κατηγορία αναφορά κυρίως νεότερους SLAM αλγόριθμους, που βασίζονται στη μηχανική μάθηση και τα νευρωνικά δίκτυα. Τα τελευταία χρόνια έχουν βρει πρόσφορο έδαφος οι έρευνες στον τομέα των SLAM τεχνικών με τη χρήση μεθόδων τεχνητής νοημοσύνης με ιδιαίτερη έμφαση στη συνδυαστική χρήση τους με LIDAR αισθητήρες [11] [12] [13] [14] και στην πλοήγηση σε εσωτερικούς χώρους [15] [16] [17] [18]. Το κύριο πλεονέκτημα αλλά και μειονέκτημά τους είναι η ανάγκη για προκαταρκτική εκπαίδευση μοντέλων, που προσθέτει ένα σημαντικό υπολογιστικό φόρτο στο σύστημα, ενώ η διαθεσιμότητα ελεύθερων βάσεων δεδομένων για την εκπαίδευση μοντέλων για πλοήγηση σε εσωτερικούς χώρους είναι περιορισμένη.

## 2.3 Συγχώνευση Αισθητήρων - Sensor Fusion

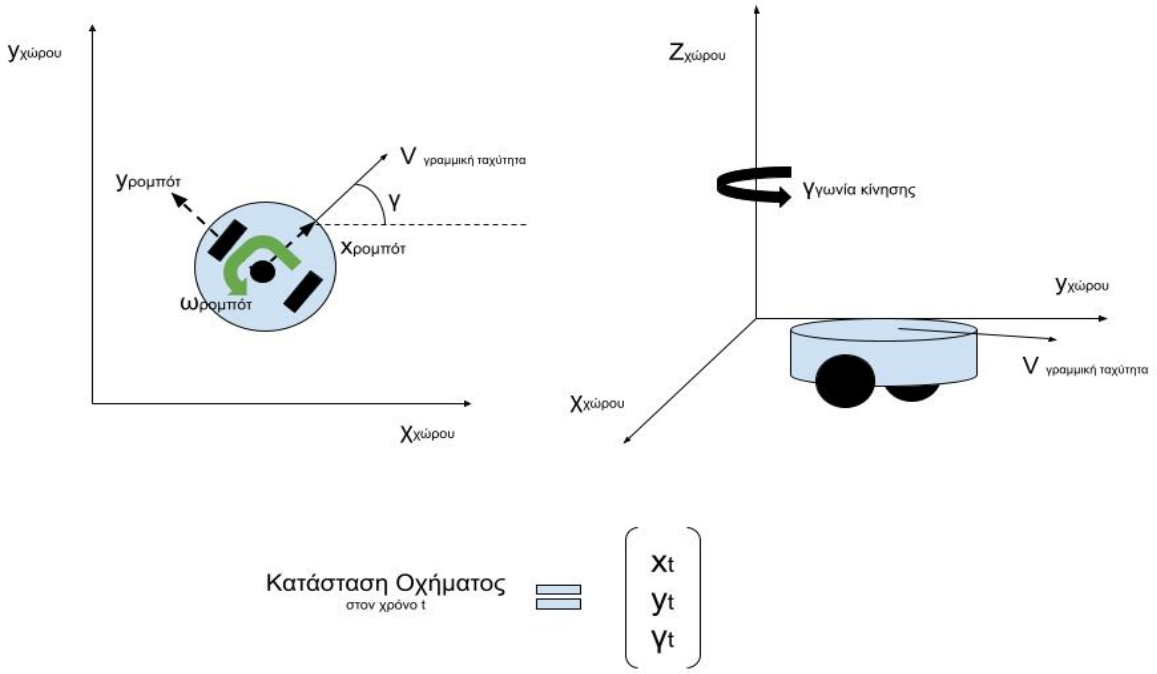
Ο έλεγχος ενός αυτοκινούμενου συστήματος με τη χρήση μόνο ενός αισθητήρα αποτελεί δύσκολη υπόθεση, και αυτό γιατί οι αισθητήρες όταν χρησιμοποιούνται σε πραγματικό χρόνο περιέχουν πολλές φορές θόρυβο στα δεδομένα τους, καθιστώντας τους αναξιόπιστους. Για παράδειγμα οι τροχοί μπορεί να ολισθήσουν και οι κωδικοποιητές να χάσουν τις μετρήσεις τους ή οι εικόνες από τις κάμερες να αλλοιωθούν από εξωτερικούς παράγοντες όπως το υπερβολικό φως, η βροχή ή το σκοτάδι. Η συγχώνευση των δεδομένων από τους διάφορους αισθητήρες (sensor fusion) αποσκοπεί στην εξομάλυνση των θορυβωδών μετρήσεων και στον υπολογισμό ενός ενιαίου μοντέλου, με στόχο την αποτύπωση μιας πιο ακριβούς εικόνας του περιβάλλοντος γύρω από το όχημα.

### 2.3.1 Διευρυμένο φίλτρο Kalman - Extended Kalman Filter

Ένα από τα πιο διάσημα φίλτρα που χρησιμοποιούνται για την εξομάλυνση των θορύβων είναι το φίλτρο Kalman και συγκεκριμένα η ενισχυμένη έκδοση του, το Extended Kalman Filter (EKF). Ουσιαστικά πρόκειται για έναν αλγόριθμο που χρησιμοποιεί πολλαπλούς αισθητήρες ως είσοδο, για την εκτίμηση της θέσης του οχήματος και τον υπολογισμό του προσανατολισμού της γραμμικής και γωνιακής ταχύτητας του συστήματος σε σχέση με τον χρόνο.

Έστω ότι έχουμε ένα διαφορικό όχημα που κινείται στον χώρο και προσπαθούμε να υπολογίσουμε την κατάσταση του οχήματος (state) σε χρονική στιγμή t. Η θέση και ο προσανατολισμός του, μπορούν να περιγραφούν από τη θέση του οχήματος στους χ, y και z άξονες. Ωστόσο, η δυσκολία ενός φαινομενικά απλού υπολογισμού, έγκειται στην ανακρίβεια των δεδομένων που συλλέγουν οι αισθητήρες του

οχήματος. Όταν οι μετρήσεις εμπεριέχουν θόρυβο, οι υπολογισμοί της θέσης του οχήματος ανά χρονική στιγμή t είναι ασταθής και δεν γίνεται να ακολουθήσουν κάποιο γραμμικό μοντέλο, με αποτέλεσμα το σύστημα να μην μπορεί να είναι




**Εικόνα 4: Κίνηση οχήματος στους καρτεσιανούς άξονες**

απόλυτα σίγουρο για την ακριβή του θέση και κατεύθυνση στον χώρο.

Το διευρυμένο φίλτρο Kalman προσεγγίζει το παραπάνω πρόβλημα αβεβαιότητας μη γραμμικών μοντέλων, παράγοντας εκτιμήσεις της κατάστασης του συστήματος με τη χρήση φίλτρων Bayes. Ο αλγόριθμος περιγράφει την πιθανότητα μετάβασης της κατάστασης του οχήματος και των μετρήσεων, με διαφορικές συναρτήσεις. Σαν έξοδος, παράγεται μια μαθηματική προσέγγιση των μετρήσεων, υπό μορφή Gaussian κατανομής, με εκτιμώμενη μέση τιμή και συνδιακύμανση. Η εκτιμώμενη θέση $\boldsymbol{x_t}$ και μετρήσεις του οχήματος δίνεται από τη σχέση [19][20]:

$$\mathrm{x(t) = g(u(t), x(t-1)) + w(t)}$$
$$\mathrm{z(t) = h(x(t)) + v(t)}$$

Τα $w_t$ και $v_t$ είναι οι θόρυβοι διεργασίας και παρατήρησης και ως $u_t$ ορίζουμε το διάνυσμα ελέγχου. Και οι δύο μεταβλητές υποθέτουμε ότι είναι μηδενικοί μέσοι μεταβλητοί θόρυβοι Gauss με συνδιακύμανση $\Sigma_t$**.** Παρακάτω παρατίθεται ο αλγόριθμος του διευρυμένου φίλτρου Kalman όπως τον διατύπωσε ο J.A.O. Nordin (2017), μέσω των S. Thrun, W. Burgard, and D. Fox (2005) [19][20]:

1: **function Extended_Kalman_filter($\mu_{t-1}$, $\Sigma_{t-1}$, $u_t$ , $z_t$ )**
2: $\overline{\mu_t} = g(u_t , \mu_{t-1})$
3: $\overline{\Sigma_t} = G_t \Sigma_{t-1} G^t{}_t + R_t$
4: $K_t = \overline{\Sigma_t} H^t{}_t (H_t \overline{\Sigma_t} H^t{}_t + Q_t)^{-1}$
5: $\mu_t = \overline{\mu_t} + K_t (z_t - h(\overline{\mu_t}))$
6: $\Sigma_t = (I - K_t H_t) \overline{\Sigma_t}$
7: **return** $\mu_t$, $\Sigma_t$
8: **end function**

Ο αλγόριθμος υπολογίζει τις προβλέψεις του μέσου όρου $\mu_t$ και της συνδιακύμανσης $\Sigma_t$ στη γραμμή 2 και 3. Αυτές οι προβλέψεις χρησιμοποιούνται στο βήμα διόρθωσης (γραμμές 4-6). Το πρώτο βήμα της διόρθωσης είναι ο υπολογισμός του κέρδους Kalman που συμβολίζεται $K_t$. Επιπλέον, ο υπολογισμός του μέσου όρου $\mu_t$ πραγματοποιείται σε σχέση με τον προβλεπόμενο μέσο όρο $\mu_t$ και της μέτρησης $z_t$. Τέλος, η νέα συνδιακύμανση $\Sigma_t$ υπολογίζεται όσον αφορά τον παράγοντα κέρδους Kalman καθώς και την προβλεπόμενη συνδιακύμανση Σ. [19][20]

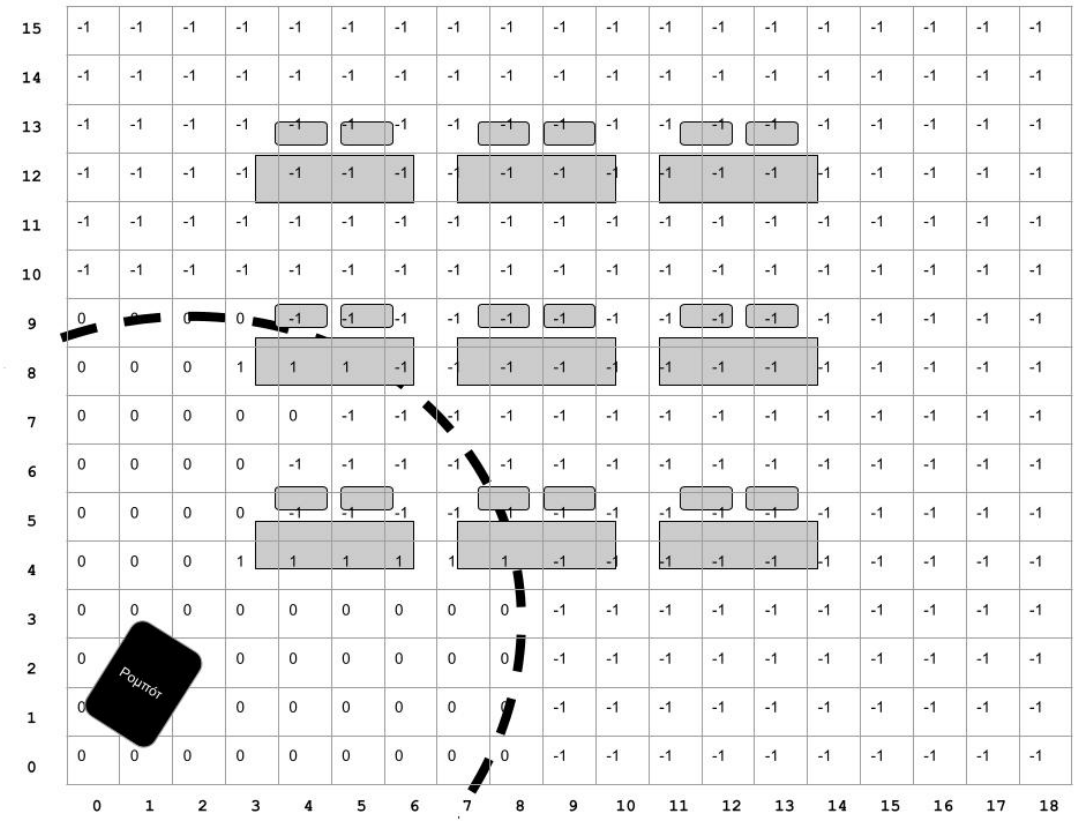
## 2.4 Ενεργό SLAM – Active SLAM

Για την επίτευξη μιας πλήρους αυτόνομης εξερεύνησης του περιβάλλοντος είναι απαραίτητο το σύστημα να γνωρίζει που βρίσκεται, ποιες περιοχές έχει επισκεφθεί και που πρέπει να πάει. Κατά την εκτέλεση SLAM οι αλγόριθμοι εστιάζουν στην καταγραφή του περιβάλλοντος και ταυτόχρονα στον εντοπισμό της θέσης, χωρίς όμως να αναλαμβάνουν τον σχεδιασμό της πορείας (path) που θα ακολουθήσει το όχημα. Συνήθως ο χρήστης δίνει οδηγίες μέσω κάποιου συστήματος τηλεκατεύθυνσης καθορίζοντας ακριβώς τη διαδρομή που πρέπει να εκτελεστεί. Εάν αφαιρέσουμε όμως τον χρήστη από το σενάριο, το όχημα πρέπει με κάποιον τρόπο να σχεδιάσει αυτόνομα την πορεία του ακόμα και για αχαρτογράφητες περιοχές που δεν γνωρίζει ακόμα. Με τον όρο ενεργό (Active) SLAM προσπαθούμε να περιγράψουμε το πρόβλημα ελέγχου της κίνησης ενός ρομποτικού συστήματος με στόχο τον υπολογισμό της βέλτιστης τροχιάς για την ελαχιστοποίηση της αβεβαιότητας του χάρτη και τον εντοπισμό του συστήματος.

Το ενεργό SLAM μπορεί να επίσης να χαρακτηριστεί και ως ένα πρόβλημα λήψης αποφάσεων καθώς κατά την εξερεύνηση του χώρου, το ρομποτικό σύστημα πρέπει να αποφασίσει μεταξύ της επίσκεψης νέων αχαρτογράφητων στόχων και της επανεπίσκεψης ήδη γνωστών θέσεων για τη μείωσης της ανακρίβειας του σχεδιασμένου χάρτη. Στα παρακάτω κεφάλαια θα αναλυθούν οι τεχνολογίες που επιλέχθηκαν για τη επίλυση του προβλήματος.

## 2.4.1 Πλέγμα Πληρότητας Χάρτη - Occupancy Grid Map

Τα πλέγματα πληρότητας χρησιμοποιούνται για να περιγράψουν τους χάρτες με τη μορφή πίνακα κελιών. Κάθε κελί περιέχει μια τιμή που αντιπροσωπεύει την παρουσίας ενός εμποδίου στην αντίστοιχη θέση στο περιβάλλον. Υπάρχουν δύο είδη πλεγμάτων ανάλογα με το είδος των τιμών που περιέχουν τα κελιά. Τα δυαδικά πλέγματα που παίρνουν τιμές 0 και 1 ανάλογα με το αν ο χώρος είναι ελεύθερος ή όχι αντίστοιχα, και στα πιθανοτικά πλέγματα, όπου κάθε κελί περιέχει την πιθανότητα ύπαρξης ενός εμποδίου στην εκάστοτε θέση. Για τον υπολογισμό ενός πλέγματος χάρτη είναι απαραίτητη η γνώση της κατάστασης του οχήματος αλλά και του περιβάλλοντος.




**Εικόνα 5: Παράδειγμα πλέγματος πληρότητας χάρτη.** Το ρομπότ έχει πάντα σχετική θέση (0,0) και με τον αισθητήρα του οπτικού ραντάρ ανιχνεύει το περιβάλλον του. Τα κελιά διατηρούν τιμή εύρους [0,1] ανάλογα με την πιθανότητα ύπαρξης εμποδίου, και για χάρη του παραδείγματος παίρνουν τιμή -1 αν είναι άγνωστα.

## 2.4.2 Τεχνική Εξερεύνησης Συνόρων – Frontier Search

Η τεχνική εξερεύνησης συνόρων προσφέρει μία λύση στο πρόβλημα της αυτόνομης εξερεύνησης. Ως σύνορα (frontiers) ορίζουμε τα όρια ανάμεσα στις άγνωστες και στις ήδη χαρτογραφημένες περιοχές. Για τον εντοπισμό frontiers χρησιμοποιούνται τα πλέγματα πληρότητας χάρτη. Ο αλγόριθμός που παρουσιάστηκε από τον B. Yamauchi[21] χρησιμοποιεί τις τιμές των κελιών για να ανιχνεύσει τα σύνορα και τον επόμενο προορισμό του οχήματος. Τα κελιά ανάλογα με την τιμή πιθανότητας που διαθέτουν κατηγοριοποιούνται σε τρεις κατηγορίες:

- Ανοιχτά: πιθανότητα κατάληψης < προηγούμενη πιθανότητα.
- Άγνωστα: πιθανότητα κατάληψης = προηγούμενη πιθανότητα.
- Κατειλημμένο: πιθανότητα κατάληψης > προηγούμενη πιθανότητα.

Με την προηγούμενη πιθανότητα, να ορίζεται η τιμή που είχαν τα κελιά στην αμέσως προηγούμενη εκτίμηση. Ο αλγόριθμος χαρακτηρίζει όλα τα ανοιχτά κελιά που γειτνιάζουν με άγνωστα κελιά ως ακμές συνόρου και στην συνέχεια τα ομαδοποιεί σε συνοριακές περιοχές. Οι συνοριακές περιοχές που έχουν το κατάλληλο μέγεθος, δηλαδή λίγο μεγαλύτερο από το μέγεθος του ρομποτικού οχήματος, θεωρούνται σύνορα. Το ρομπότ θα πλοηγηθεί προς το πλησιέστερο προ βάσιμο ανεξερεύνητο σύνορο. Ο διακομιστής σχεδιασμού μονοπατιού (planner server) εκτελεί Depth First αναζήτηση σε ολόκληρο το πλέγμα και προσπαθεί να επιλέξει το συντομότερο ελεύθερο μονοπάτι χωρίς εμπόδια για να φτάσει στον προορισμό του. Όσο κατευθύνεται προς το σύνορο, ειδικοί μηχανισμοί αποφυγής εμποδίων ενεργοποιούνται και χρησιμοποιούν τους αισθητήρες των συστήματος για να κατευθυνθεί γύρω από τα εμπόδια και αν ακολουθήσει την προκαθορισμένη διαδρομή του. Μόλις το σύστημα φτάσει σε ένα σύνορο, τότε αυτό αποθηκεύεται στη λίστα των συνόρων που είχε επισκεφθεί προηγουμένως. Το ρομπότ τότε ξανά εκτελεί σάρωση του περιβάλλοντος υπολογίζοντας το νέο πιθανολογικό πλέγμα χάρτη και διαλέγει το επόμενο σύνορο-προορισμό του επαναλαμβάνοντας όλη τη διαδικασία.

# ΚΕΦ.3: Σχεδίαση Συστήματος

## 3.1 Επιλογή Βιβλιοθηκών και Αλγορίθμων

Στα προηγούμενα κεφάλαια αναλύθηκε εκτενώς η πολυπλοκότητα που εμπεριέχει το φαινομενικά απλό έργο της πλοήγησης και εξερεύνησης. Εφόσον για την υλοποίηση του συστήματος είναι απαραίτητος ο συνδυασμός πολλών διαφορετικών τεχνολογιών και λογισμικών πακέτων, είναι απόλυτα κρίσιμο να σχεδιάσουμε και να διαχωρίσουμε την αρχιτεκτονική του συστήματος σε μικρότερα ξεκάθαρα στρώματα. Σε αυτή την ενότητα θα εξετάσουμε τις επιλογές των αλγορίθμων αλλά και την αρχιτεκτονική που υλοποιήθηκε στα πλαίσια του ROS.

### 3.1.1 Επιλογή λογισμικού

Από την έρευνα που διεξήχθη φαίνεται πως ο πιο σύγχρονος, δημοφιλής και αξιόπιστος τρόπος για να προγραμματιστεί η ρομποτική πλατφόρμα ήταν η χρήση του ρομποτικού λειτουργικού συστήματος ROS2. Δεν βρέθηκε κάποια άλλο λογισμικό πακέτο ανοιχτού κώδικα με την ίδια απήχηση στην κοινότητα ρομποτικής και με τόσα ελεύθερα διαθέσιμα έγγραφα ανάλυσης και υποστήριξης των προγραμματιστών, καθιστώντας το ROS2 την πρώτη επιλογή όταν πρόκειται για την ανάπτυξη ρομποτικών συστημάτων.

Ωστόσο το ROS2 εισάγει και μερικούς περιορισμούς καθώς όλοι οι αλγόριθμοι και οι βιβλιοθήκες που χρησιμοποιήθηκαν έπρεπε να είναι συμβατοί με την νέα έκδοσή του, και δεδομένου του σύντομου χρόνου κυκλοφορίας του, πολλά από τα πακέτα που είχαν αναπτυχθεί όλα αυτά τα χρόνια για την προηγούμενη έκδοσή του είναι

πλέον ασύμβατα και μη λειτουργικά. Γνωρίζοντας αυτό το πρόβλημα, μαζί με το ROS2 εκδόθηκε και ένα πακέτο υποστήριξης «ros_bridge» που ουσιαστικά γεφυρώνει το χάσμα μεταξύ των εκδόσεων μετατρέποντας όλα τα μηνύματα και τα πακέτα που έρχονται από το ROS2 σε μορφή συμβατή με το ROS1 και αντίστροφα. Όταν δοκιμάστηκε στο Raspberry Pi [39], παρατηρήθηκε αύξηση της κατανάλωσης των πόρων τους συστήματος και δεν φάνηκε χρήσιμο να προσθέσουμε τέτοια περιπλοκότητα και καθυστερήσεις στο επίπεδο επικοινωνίας μεταξύ των κόμβων, επομένως προτιμήθηκε η χρήση καθαρά ROS2 πακέτων.

#### 3.1.1.1 Έκδοση ROS2 και λειτουργικού συστήματος.

Υπάρχουν τρεις διαφορετικοί τύποι εκδόσεων του ROS2:

- Long Time Support (LTS)
- Stable
- Rolling

Οι LTS εκδόσεις περιέχουν τις περισσότερες ενημερώσεις και διορθώσεις, ανακοινώνονται ανά τουλάχιστον δύο χρόνια, και υποστηρίζονται από την επίσημη ομάδα ανάπτυξης για μεγαλύτερο διάστημα από όλες τις άλλες εκδόσεις. Οι stable ανακοινώνονται κατά τα μονά έτη και περιέχουν νέες λειτουργίες και μικρό-διορθώσεις σε προβλήματα που μπορεί να εμφανίστηκαν στις LTS εκδόσεις. Παρόλα αυτά, οι stable έχουν μικρό κύκλο ζωής καθώς παρέχεται περιορισμένη υποστήριξη για μικρό διάστημα και σε αντίθεση με το όνομα τους δεν είναι πάντα σταθερές. Καθώς περνάει ο χρόνος και πλησιάζουν προς το τέλος ζωής τους, όλο και περισσότερα προβλήματα εμφανίζονται που για να λυθούν, ο χρήστης καλείται να κάνει re-build τις προβληματικές βιβλιοθήκες με τα διορθωμένα κομμάτια κώδικα (εάν αυτά έχουν διανεμηθεί από την ομάδα ανάπτυξης), ότι θα έκανε δηλαδή για να λύσει και τα προβλήματα που εμφανίζονται και στις LTS εκδόσεις. Τέλος, υπάρχουν και τα Rolling Releases, που ανανεώνονται πολύ συχνά, περιέχουν τα περισσότερα hotfixes και είναι ασταθείς καθώς έχουν μεγάλες αλλαγές που συχνά επηρεάζουν την συμπεριφορά των APIs. Για την υλοποίηση της παρούσας εργασίας,

επιλέχθηκε η τρέχουσα LTS έκδοση του ROS2, η Foxy Fitzroy, καθώς όπως αναφέρθηκε παραπάνω, διαθέτει και την μεγαλύτερη υποστήριξη.

Σημαντικό κριτήριο για την επιλογή του λειτουργικού συστήματος που θα τρέξει στο Raspberry Pi, ήταν η συμβατότητα του με τις ROS2 βιβλιοθήκες. Κάθε LTS έκδοση του ROS2 είναι συμβατή με μία έκδοση των Ubuntu, ενώ για το προεπιλεγμένο λειτουργικό σύστημα των Raspberry Pi, Raspbian, δεν διατίθενται τα εκτελέσιμα αρχεία, αντιθέτως, πρέπει ο χρήστης να μεταγλωττίσει τον κώδικα εκ του μηδενός. Για το Foxy, η επίσημα υποστηριζόμενή έκδοση είναι η Ubuntu Linux 20.04 Focal Fossa. Επιπλέον, παρόλο που το Raspberry Pi είναι εντυπωσιακά ισχυρό για το μέγεθός του, τρέχοντας ολόκληρη την έκδοση του Ros2 λογισμικού και τον προγραμμάτων προσομοίωσης μπορεί να φανεί απαιτητικό έργο και να απασχολήσει παραπάνω πόρους από τον επεξεργαστή από ότι είναι αρεστό, για αυτό προτιμήθηκε η εγκατάσταση της Ubuntu Server έκδοσης, που δεν διαθέτει γραφικά στοιχεία. Για τους ίδιους λόγους χρησιμοποιήθηκε και η εκδοχή του Foxy Fitzroy χωρίς τα τις βιβλιοθήκες γραφικών. Για περισσότερες πληροφορίες σχετικά με την αρχιτεκτονική και πώς αργότερα χρησιμοποιήθηκαν προγράμματα απεικόνισης όπως το Rviz2 ανατρέξτε στο κεφάλαιο 3.2 [35]. Παρακάτω παρατίθενται οι εντολές που χρησιμοποιήθηκαν για την εγκατάσταση του Foxy Base και των απαιτούμενων βιβλιοθηκών:

```
1. sudo apt update && sudo apt install locales
2. sudo locale-gen en_US en_US.UTF-8
3. sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
4. export LANG=en_US.UTF-8
5. sudo apt update && sudo apt install curl gnupg2 lsb-release
```

```
6. sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key
   -o /usr/share/keyrings/ros-archive-keyring.gpg
7. echo "deb [arch=$(dpkg --print-architecture) signed-
   by=/usr/share/keyrings/ros-archive-keyring.gpg]
   http://packages.ros.org/ros2/ubuntu $(source /etc/os-release && echo
   $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list >
   /dev/null
8. sudo apt update
9. sudo apt install ros-foxy-ros-base
10. sudo apt install python3-colcon-common-extensions
11. sudo apt install python3-pip
12. pip3 install argcomplete
13. source /opt/ros/foxy/setup.bash
   /*Add this to your bashrc so you dont have to source ROS2 everytime*/
```

### 3.1.2 Βασικός Αισθητήρας SLAM – Οπτικό Ραντάρ ή Κάμερα

Η επιλογή της δεύτερης έκδοσης του ρομποτικού λειτουργικού συστήματος ελαχιστοποιεί τις εναλλακτικές των διαθέσιμων αλγορίθμων και αισθητήρων. Για να επιλεχθεί ο τελικός αλγόριθμος SLAM ήταν απαραίτητη πρώτα η επιλογή των αισθητήρων και για να διαλέξουμε τους αισθητήρες έπρεπε πρώτα να μελετήσουμε ποιοι SLAM αλγόριθμοι ήταν διαθέσιμοι στο ROS2.

Στην αρχή δοκιμάστηκε η χρήση κάμερας μοντέλου Raspberry PI Module 2, ανάλυσης 8 MP. Το γεγονός ότι επιλέχθηκε λογισμικό Ubuntu χωρίς γραφική διεπαφή χρήστη και χωρίς υποστήριξη για τις φυσικές (hardware) ιδιότητες του επεξεργαστή όπως οι i2c,spi κλπ. θύρες, δυσκόλεψε αρκετά την εγκατάσταση και την βαθμονόμηση της κάμερας. Ακόμα πιο δύσκολο ήταν να βρεθούν αξιόπιστοι αλγόριθμοι ανοιχτού κώδικα που να υποστηρίζουν την ROS2 Foxy έκδοση. Οι πιο γνωστοί αλγόριθμοι Visual SLAM, όπως οι ORB 2-SLAM, VISO2, OpenVSLAM δεν

διατίθενται από τις επίσημες ομάδες ανάπτυξης στο ROS2, με τον OpenVSLAM να έχει καταργηθεί τελείως. Ακόμα και να είχαν ενσωματώσει το ROS2 λογισμικό, η χρήση οπτικού SLAM και κάμερας θα προϋπέθετε και τη δυνατότητα επεξεργασίας της εικόνας για την εξαγωγή χαρακτηριστικών ή ακόμα και εκπαίδευση μοντέλων για την αναγνώριση σχημάτων και εμποδίων, ανάλογα με τον αλγόριθμο. Και οι δύο περιπτώσεις, απαιτούν υψηλή υπολογιστική δύναμη από ένα Raspberry Pi ήδη επιβαρυμένο να εκτελεί ταυτόχρονα SLAM, εξερεύνηση και ταυτόχρονα πλοήγηση. Επιπλέον, στην περίπτωση της αναγνώρισης εικόνων είναι απαραίτητη και η εύρεση βάσεων δεδομένων για την εκπαίδευση των αντίστοιχων μοντέλων ενώ είναι και αναγκαίος ο καθορισμός σαφών απαιτήσεων σχετικά με το είδος των αντικειμένων που πρέπει το μοντέλο να ανιχνεύει, αν θα έπρεπε για παράδειγμα να εντοπίζονται τοίχοι, καρέκλες, τραπέζια, βιβλιοθήκες, ρούχα κλπ. Δεδομένου ότι στην συγκεκριμένη πτυχιακή εργασία εστιάζουμε στην εξερεύνηση οποιουδήποτε εσωτερικού χώρου που μπορεί να περιέχει πληθώρα απρόβλεπτων αντικειμένων, δεν συμβαδίζει με τις απαιτήσεις της εργασίας να περιορίσουμε τα μοντέλα ανίχνευσης σε συγκεκριμένες περιπτώσεις μόνο. Επίσης οι ελεύθερα διαθέσιμες βάσεις δεδομένων που περιέχουν αντικείμενα εσωτερικού χώρου είναι σχεδόν ανύπαρκτες καθώς οι κάμερες και τα μοντέλα ανίχνευσης χρησιμοποιούνται κυρίως για την πλοήγηση σε εξωτερικούς χώρους όπως στα ηλεκτροκίνητα αυτοκινούμενα οχήματα. Τέλος, με την εξερεύνηση εσωτερικών χώρων σημαίνει ότι μερικές φορές οι συνθήκες φωτισμού μπορεί να μην είναι επαρκείς για τη χρήση καμερών. Για όλους τους παραπάνω λόγους αποφασίστηκε η χρήση οπτικού ραντάρ (LIDAR) έναντι κάμερας.

### 3.1.3 Αλγόριθμος SLAM

Η χρήση του οπτικού ραντάρ ως βασικού αισθητήρα χαρτογράφησης διευκολύνει αρκετά την τελική επιλογή του SLAM αλγόριθμου, και αυτό διότι για το ROS2 είναι

ελάχιστες οι εναλλακτικές. Σύμφωνα με Steve Macenski και Ivona Jambrecic (2021) στην έκθεση για το πακέτο slam_toolbox [22] αναφέρουν ότι από τους αλγόριθμους SLAM ανοιχτού κώδικα οπτικών ραντάρ που ήταν διαθέσιμοι στο λειτουργικό σύστημα ρομπότ και συγκεκριμένα τους GMapping, Karto, Cartographer και Hector, λίγοι από αυτούς μπορούν να δημιουργήσουν ακριβείς χάρτες μεγάλων χώρων. Ακόμη λιγότεροι μπορούν να το κάνουν σε πραγματικό χρόνο. Το μόνο πακέτο που μπορούσε να επιτύχει τα παραπάνω ήταν ο Cartographer. Ωστόσο, εγκαταλείφθηκε από την Google και δεν συντηρείται πλέον. Επίσης ο Karto, ο Hector και ο Gmapping δεν έχουν υποστηριχθεί ακόμα επισήμως στην ROS2 έκδοση, αφήνοντας μόνο μια επιλογή τον slam_toolbox. Επιπλέον αξίζει να αναφερθεί ότι το πακέτο που θα χρησιμοποιηθεί για την πλοήγηση, Navigation2, υποστηρίζει μόνο δύο αλγόριθμους εντοπισμού θέσης, τους slam_toolbox και AMCL (Adaptive Monte Carlo Localization), με τον AMCL να μην μπορεί να χρησιμοποιηθεί σε χώρους που δεν διαθέτουμε ήδη τους χάρτες τους. Επομένως, η μοναδική λύση που ικανοποιεί τις προϋποθέσεις της εργασίας είναι ο slam_toolbox.

Ενώ το πακέτο slam_toolbox θα χρησιμοποιηθεί για τη χαρτογράφηση, για τον εντοπισμό θέσης επιλέχθηκε το πακέτο robot_localization[26] που υλοποιεί τον διευρυμένο φίλτρο Kalman διαμορφωμένο για να υποστηρίζεται από το ROS.

### 3.1.4 Βιβλιοθήκης Πλοήγησης

Για την πλοήγηση του ρομποτικού συστήματος το μοναδικό πακέτο που βρέθηκε με διαθέσιμη υποστήριξη και αποδεδειγμένη λειτουργικότητα είναι το Nav2[28]. Αποτελεί την επίσημη συνέχιση της ROS1 βιβλιοθήκης, navigation_stack και αναλαμβάνει την πλοήγηση του ρομποτικού συστήματος από την αρχική του θέση έως τον προορισμό του. Αυτό περιλαμβάνει και τον υπολογισμό του βέλτιστου μονοπατιού, την αποφυγή εμποδίων και τον έλεγχο της ταχύτητας του οχήματος σε πραγματικό χρόνο κατά τη διάρκεια της πλοήγησης, με τη χρήση πάντα του χάρτη

που δημοσιεύεται από τον slam_toolbox και της οδομετρίας.

### 3.1.5 Βιβλιοθήκη Εξερεύνησης

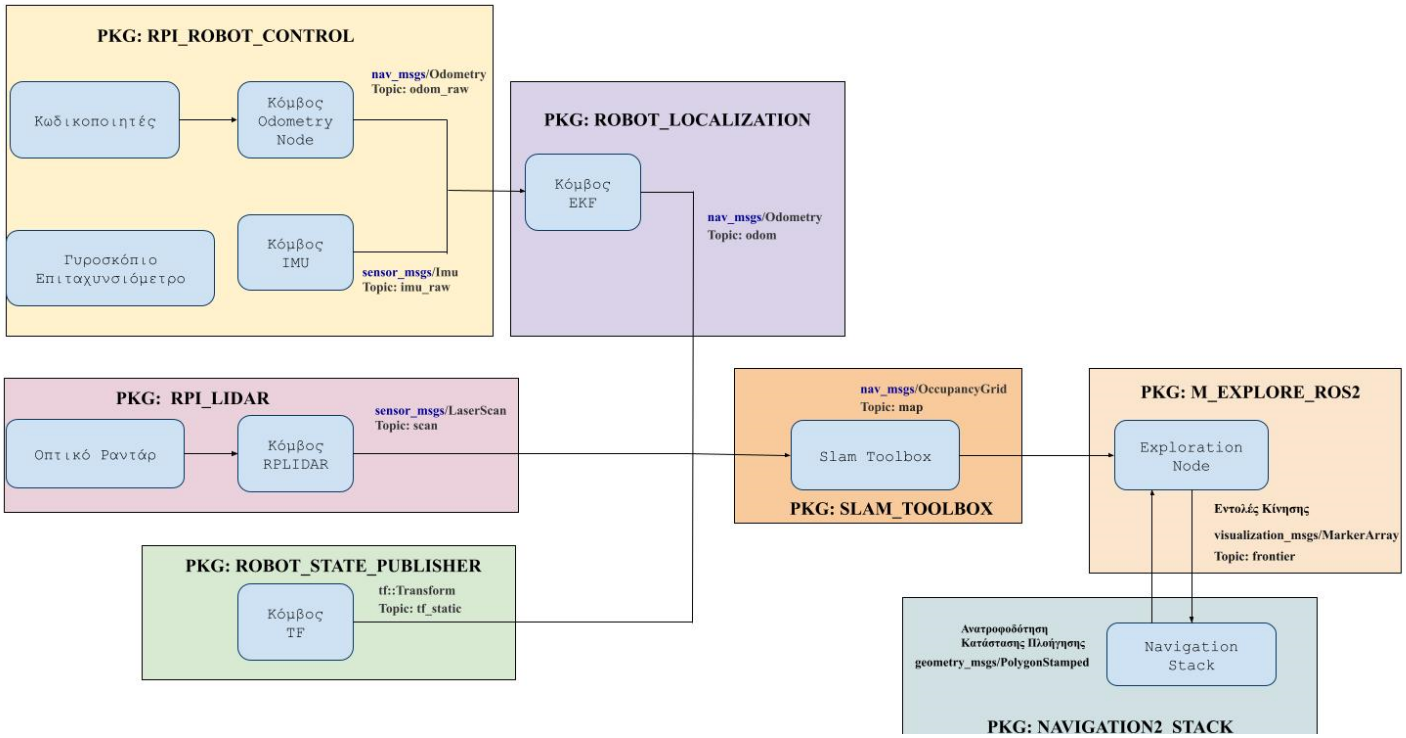
Για την εξερεύνηση προτιμήθηκαν βιβλιοθήκες που υποστηρίζουν αλγόριθμους εξερεύνησης συνόρων όπως αναφέρθηκε στο κεφάλαιο 2.4.2 [26]. Βρέθηκε μόνο ένα πακέτο που καλύπτει τις ανάγκες της συγκεκριμένης εργασίας, το m-explore-ros2[25] που βασίζεται στα explore_lite[23] και frontier_search[21][24] του ROS1. Το m-explore-ros2 εκτελεί αναζήτηση συνόρων στο πιθανοτικό πλέγμα χάρτη που δημοσιεύεται από το slam toolbox. Στη συνέχεια υπολογίζει τον κοντινότερο ανεξερεύνητο σύνορο και στέλνει τις συντεταγμένες του στην βιβλιοθήκη πλοήγησης η οποία αναλαμβάνει τον σχεδιασμό της βέλτιστης διαδρομής.

## 3.2. Αρχιτεκτονική Συστήματος

Ένα από τα βασικότερα χαρακτηριστικά του λειτουργικού συστήματος ρομπότ είναι η χρήση κόμβων (nodes) για την αναμετάδοση πληροφορίας. Κάθε κόμβος χρησιμοποιείται για να επεξεργαστεί την αντίστοιχη πληροφορία για την οποία έχει δημιουργηθεί και συνεργάζεται με τους υπόλοιπους κόμβους του συστήματος, δημοσιεύοντας στο τοπικό δίκτυο τα μηνύματά τους. Ουσιαστικά το ROS εισάγει τρεις πολύ βασικές έννοιες για την επικοινωνία μεταξύ των κόμβων, αυτές των θεμάτων (topic), εγγραφής (subscribe) και δημοσίευσης (publish). Κάθε κόμβος μπορεί να εγγράφεται ή να δημοσιεύει (ή και τα δύο) σε έναν ή σε πολλαπλά topic, προκαθορισμένης μορφής μηνύματα. Είναι πολύ σημαντικό τα μηνύματα των κόμβων να ακολουθούν τα προκαθορισμένα πρότυπα ανάλογα με το topic που θέλουν να χρησιμοποιήσουν γιατί αλλιώς υπάρχει κίνδυνος οι υπόλοιποι κόμβοι του δικτύου να μην μπορούν να καταλάβουν τα μηνύματα.

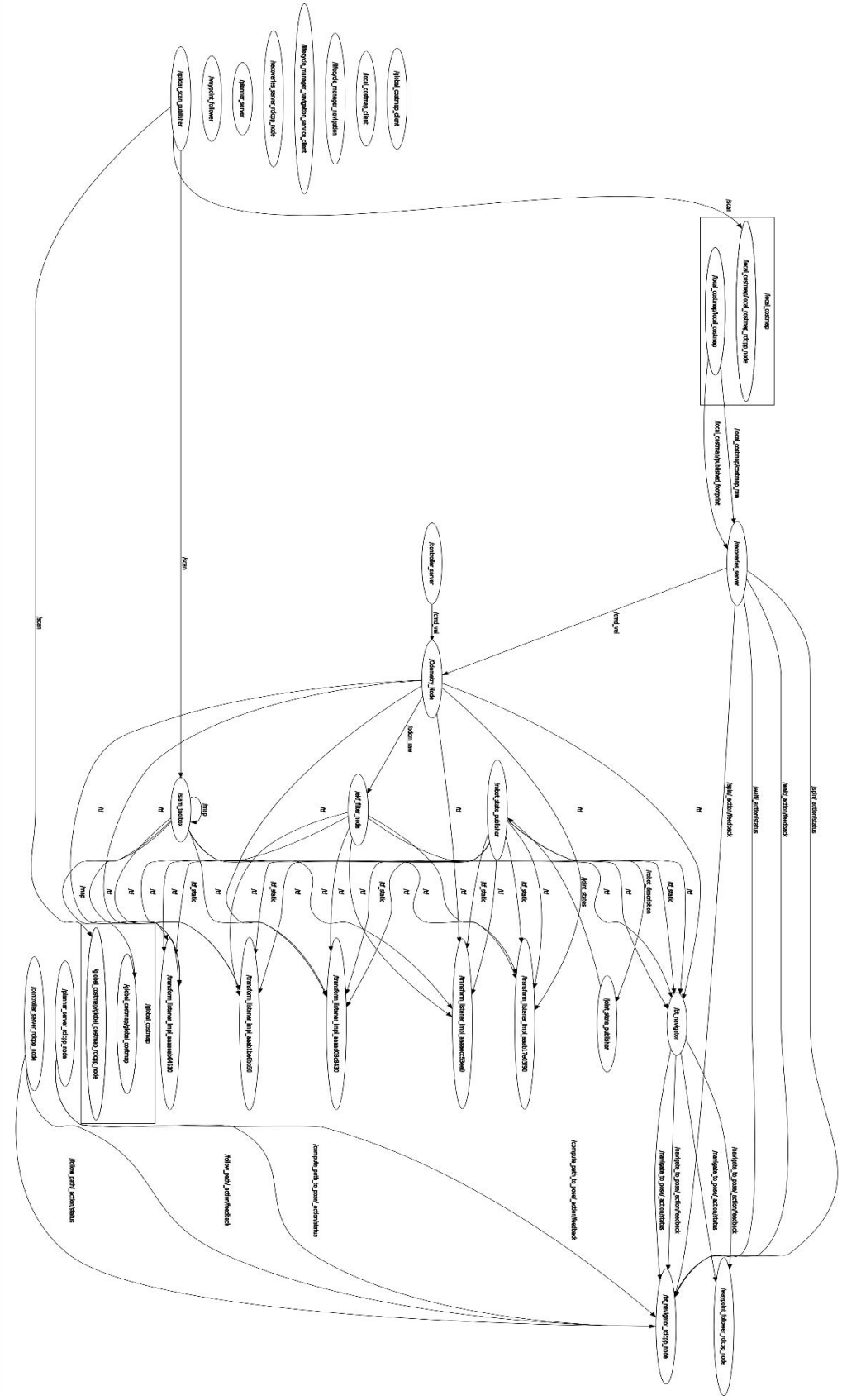
Για να εξηγηθεί η αρχιτεκτονική του συστήματος καλύτερα, θα χρησιμοποιηθεί η εικόνα 9. Αρχικά δημιουργήθηκε το πακέτο rpi_robot_bringup που περιέχει το urdf μοντέλο [47], όλα τα αρχεία παραμέτρων για τους αλγόριθμους και το βασικό αρχείο robot_state_publisher.launch.py. Το launch αρχείο είναι υπεύθυνο για την αρχικοποίηση όλων των υπόλοιπων πακέτων του έργου, έτσι ώστε ο χρήστης να εκτελεί μόνο μια εντολή όταν θέλει να τρέξει τον κώδικα, και όχι να αρχικοποιεί κάθε πακέτο ξεχωριστά. Στη συνέχεια δημιουργήθηκε το rpi_robot_control που συλλέγει δεδομένα από το γυροσκόπιο, επιταχυνσιόμετρο και τους κωδικοποιητές των τροχών, τα μετατρέπει σε αποδεκτή ROS2 μορφή μηνύματος και τα δημοσιεύει στο δίκτυο ROS. Το πακέτο διαθέτει δύο κόμβους που διαβάζουν το IMU και τους κωδικοποιητές αντίστοιχα. Ο κόμβος οδομετρίας μετατρέπει τα μηνύματα σε τύπου geometry_msgs/Twist και τα δημοσιεύει στο topic «odom_raw» και ο κόμβος IMU (γυροσκόπιο/επιταχυνσιόμετρο), στο topic «imu_raw», σε μορφή sensor_msgs/Imu.

Οι δύο κόμβοι τροφοδοτούν το διευρυμένο φίλτρο Kalman για να μπορέσει να υπολογίσει την θέση του συστήματος. Η επεξεργασία γίνεται στο πακέτο robot_localization που εγγράφεται στα δύο topics «imu_raw» και «odom_raw» καθώς ξέρει ότι θα δεχθεί τα αντίστοιχα μηνύματα. Κατά αυτόν τον τρόπο, ενώ το φίλτρο Kalman και οι κόμβοι των αισθητήρων προέρχονται από διαφορετικές βιβλιοθήκες μπορούν να επικοινωνούν επιτυχώς μεταξύ τους. Το αποτέλεσμα δημοσιεύεται στο topic «odom_filtered» τύπου nav_msgs/Odometry. Ο κόμβος του φίλτρου Kalman επίσης υπολογίζει και το transform του odom frame που θα αναλυθεί παρακάτω. Όμοια ο κόμβος που διαβάζει την πληροφορία του οπτικού ραντάρ δημοσιεύει στο topic «scan» ειδικά μηνύματα τύπου sensor_msgs/LaserScan. Το slam_toolbox ταυτόχρονα χρησιμοποιεί τα μηνύματα των RPLIDAR, tf και odom για να κατασκευάσει τον χάρτη και δημοσιεύει το πλέγμα πληρότητας στο δίκτυο για να χρησιμοποιηθεί από το exploration node. Τέλος, οι κόμβοι exploration node και navigation node αλληλεπικοινωνούν για να σχεδιάσουν το μονοπάτι πλοήγησης και να υπολογίσουν τα διαθέσιμα σύνορα ανταλλάσσοντας τα απαραίτητα μηνύματα όπως φαίνεται και στην εικόνα.

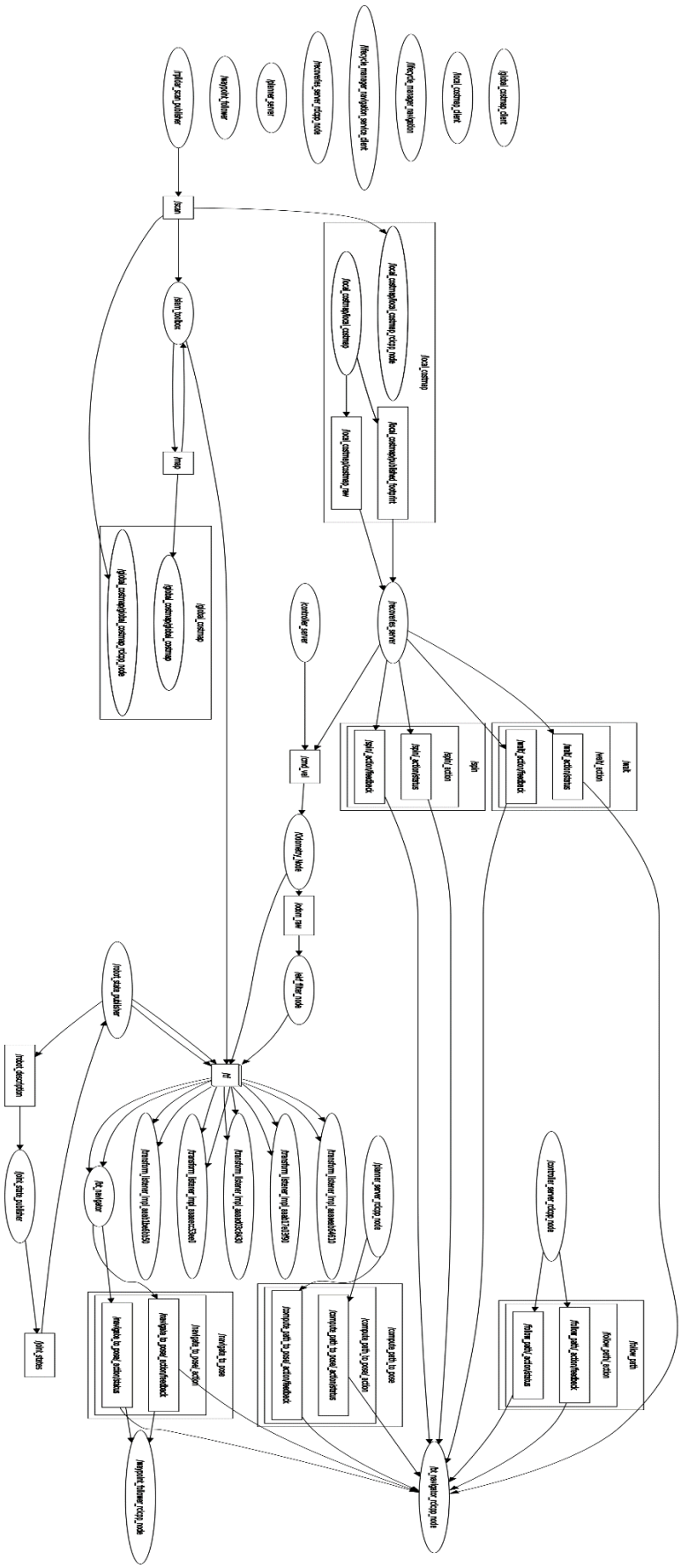



**Εικόνα 6: Αρχιτεκτονική Συστήματος**

Ακόμα πιο αναλυτικά μπορούμε να δούμε όλους τους κόμβους και τις λεπτομέρειες στις παρακάτω εικόνες:



**Εικόνα 7: Γράφημα topics**



Εικόνα 8: Γράφημα κόμβων

Επιπλέον αξίζει να αναφερθεί ότι ενώ όλο το σύστημα και οι βιβλιοθήκες «τρέχουν» στο Raspberry PI, επειδή το ίδιο δεν διαθέτει γραφικές διεπαφές για να δούμε τους χάρτες που δημιουργούνται σε πραγματικό χρόνο, τρέχουμε το πρόγραμμα RVIZ2 σε έναν δεύτερο υπολογιστή και συνδεόμαστε στο ROS δίκτυο. Έτσι μπορούμε να βλέπουμε και τα μηνύματα που δημοσιεύονται και τους χάρτες που απαιτούν GUI εργαλεία. Η συγκεκριμένη τεχνική μας δίνει τη δυνατότητα να παρακολουθούμε εξ αποστάσεως και τη λειτουργία του συστήματος.

## 3.3 Επιλογή Εξαρτημάτων

Βάσει της θεωρητικής έρευνας που προηγήθηκε [31] επιλέγουμε ως τον βασικό αισθητήρα για την χαρτογράφηση το LIDAR. Ως κεντρική μονάδα επεξεργασίας ορίζουμε το Raspberry Pi και προσθέτουμε ένα γυροσκόπιο για να μπορούμε να έχουμε πιο ακριβείς συντεταγμένες κατά τη διάρκεια της πλοήγησης.
Για την επιλογή των εξαρτημάτων και την σχεδίαση του συστήματος επίσης λήφθηκαν υπόψιν και τα περιορισμένα αποθέματα και οι καθυστερήσεις στον χρόνο παράδοσης λόγω της πανδημίας COVID19.

Αναλυτικά τα εξαρτήματα που χρησιμοποιήθηκαν:

| Hardware | Κατανάλωση Ρεύματος | |
|---|---|---|
| | Volts | Amps |
| Raspberry Pi 4 Model B 4GB RAM | 5 V | 3.3 A |
| RPLIDAR A1 360° | 5 V | 3 A |
| MPU 92/65 | 5 V | 3.3 A |
| Βοηθητικό Μεταλλικό Ροδάκι (Caster Wheel) | -- | -- |
| Μετασχηματιστής UART σε σειριακό USB | 5 V | 3 A |
| JGB37 Ρόδες **x 2** | 12 V (Max) | 2 A (Stall)<br>0.3 A (Load) |

| | | |
|---|---|---|
| MDD10A Dual Channel 10A DC Motor Driver | -- | -- |
| M3 Βίδες και Περικόχλια **x 19** | -- | -- |
| M2.5 Βίδες και Περικόχλια **x 8** | -- | -- |

**Πίνακας 1: Σύνολο εξαρτημάτων**

- Raspberry Pi Model B 4GB RAM

Για την επιλογή της επεξεργαστικής μονάδας τέθηκαν τα εξής κριτήρια:
Ο μικρό-υπολογιστής πρέπει να υποστηρίζει Linux Based λειτουργικά συστήματα, ακροδέκτες εισόδου εξόδου (GPIO) και USB 3.0 θύρες για την σύνδεση των αισθητήρων, σύνδεση στο δίκτυο για να μεταδίδονται τα δεδομένα των ROS κόμβων και να διαθέτει έναν ισχυρό επεξεργαστή που θα εκτελεί αβίαστα όλα τα απαραίτητα προγράμματα. Επιπλέον θα πρέπει να διαθέτει μικρό μέγεθος για να μην επιβαρύνει τον σκελετό.
Βάσει των παραπάνω, η επιλογή του Raspberry Pi ως κεντρικής μονάδας ήταν εύκολη. Συνδυάζοντας εντυπωσιακές επιδόσεις σε σχέση με το μικρό του μέγεθος και χαμηλό κόστος, το Raspberry έρχεται με ARMv8 64-bit επεξεργαστή υποστηρίζοντας Linux based λειτουργικά συστήματα, 4GB RAM, 40 GPIO (ακροδέκτες) και USB 3.0 για την σύνδεσή του με τους αισθητήρες και 2.4 GHz/ 5.0 GHz WiFi που καλύπτουν ικανοποιητικά τις απαιτήσεις.

- RPLIDAR A1 360°

Τα οπτικά ραντάρ (LIDARS) αποτελούν μια από τις πιο διάσημες επιλογές όσον αφορά αισθητήρες για την αποφυγή εμποδίων καθώς και για την χαρτογράφηση του χώρου. Τα LIDARS εκτοξεύουν ριπές φωτός προς μια επιφάνεια ή εμπόδιο, και μετρώντας τον χρόνο που θα χρειαστεί το φως να ανακλαστεί και να επιστρέψει στην πηγή, υπολογίζουν την απόσταση της επιφάνειας. Το συγκεκριμένο LIDAR διαθέτει αρκετά μεγάλο εύρος, στα 12

μέτρα, και περιστρέφεται 360° με μέγιστη συχνότητα 10Hz δίνοντας αποτελέσματα μεγάλης ακρίβειας. Συνδυάζοντας όλα τα παραπάνω, καθιστούν το RPLIDAR ως την καλύτερη επιλογή. Το RPLIDAR χρησιμοποιεί UART έξοδο ως προεπιλογή, για αυτό χρησιμοποιήθηκε μετασχηματιστής από UART σε σειριακή σύνδεση για να συνδεθεί απευθείας με την USB 3.0 έξοδο του Raspberry Pi.

- Τροχοί JGB37 12V

Μια από τις βασικότερες πηγές δεδομένων για τον υπολογισμό θέσης του οχήματος  είναι οι κωδικοποιητές των τροχών. Για αυτό ήταν σημαντικό να βρεθούν συνεχούς ρεύματος  ρόδες με μαγνητικούς κωδικοποιητές για μεγαλύτερη ακρίβεια, και μεγάλη ροπή για να μπορούν να κινούν ολόκληρο το όχημα παρόλο το βάρος. Υπάρχουν πολλά προϊόντα στην αγορά που πληρούν τις αναφερόμενες προϋποθέσεις, λαμβάνοντας όμως υπόψιν τη διαθεσιμότητα και τον χρόνο παράδοσης επιλέχθηκαν οι παραπάνω ρόδες.

- MDD10A Dual Channel 10A DC Motor Driver

Οι τροχοί συνεχόμενου ρεύματος συνήθως λειτουργούν με υψηλά ρεύματα και τάσεις, που οι GPIO έξοδοι του Raspberry δεν μπορούν να ικανοποιήσουν ( δίνουν περίπου 16mA μέγιστο ρεύμα). Για να μπορέσουν να λειτουργήσουν οι ρόδες εκτός από ξεχωριστή τροφοδοσία, χρησιμοποιήθηκε και ένας «καθοδηγητής» (motor driver). Η λειτουργία του δεν είναι μόνο να απομονώνει το κύκλωμα των τροχών/τροφοδοτικού αλλά και να βοηθάει στον έλεγχο της κατεύθυνσης και ταχύτητας των τροχών, μέσω PWM σήματος. Για το παρόν σύστημα, χρειαζόταν ένας καθοδηγητής που να μπορεί να ελέγχει ταυτόχρονα δυο ρόδες και να δέχεται σχετικά

υψηλά ρεύματα, για αυτό και επιλέχθηκε ο MDD10A.

- MPU 92/65

  Για να ενισχύσουμε την ακρίβεια της οδομετρίας από τους κωδικοποιητές των τροχών προτιμήθηκε η χρήση ενός γυροσκόπιού/επιταχυνσιόμετρου (IMU). Ένα από τα πιο εύχρηστα και αξιόπιστα IMUs είναι το MPU9265 που διαθέτει τρεις άξονες (x, y, z) για το γυροσκόπιο και τρεις για το επιταχυνσιόμετρο. Επιπλέον, το MPU9265 είναι εξοπλισμένο με ένα μαγνητόμετρο που δεν θα αξιοποιηθεί στην συγκεκριμένη εργασία.

## 3.4 Τροφοδοσία

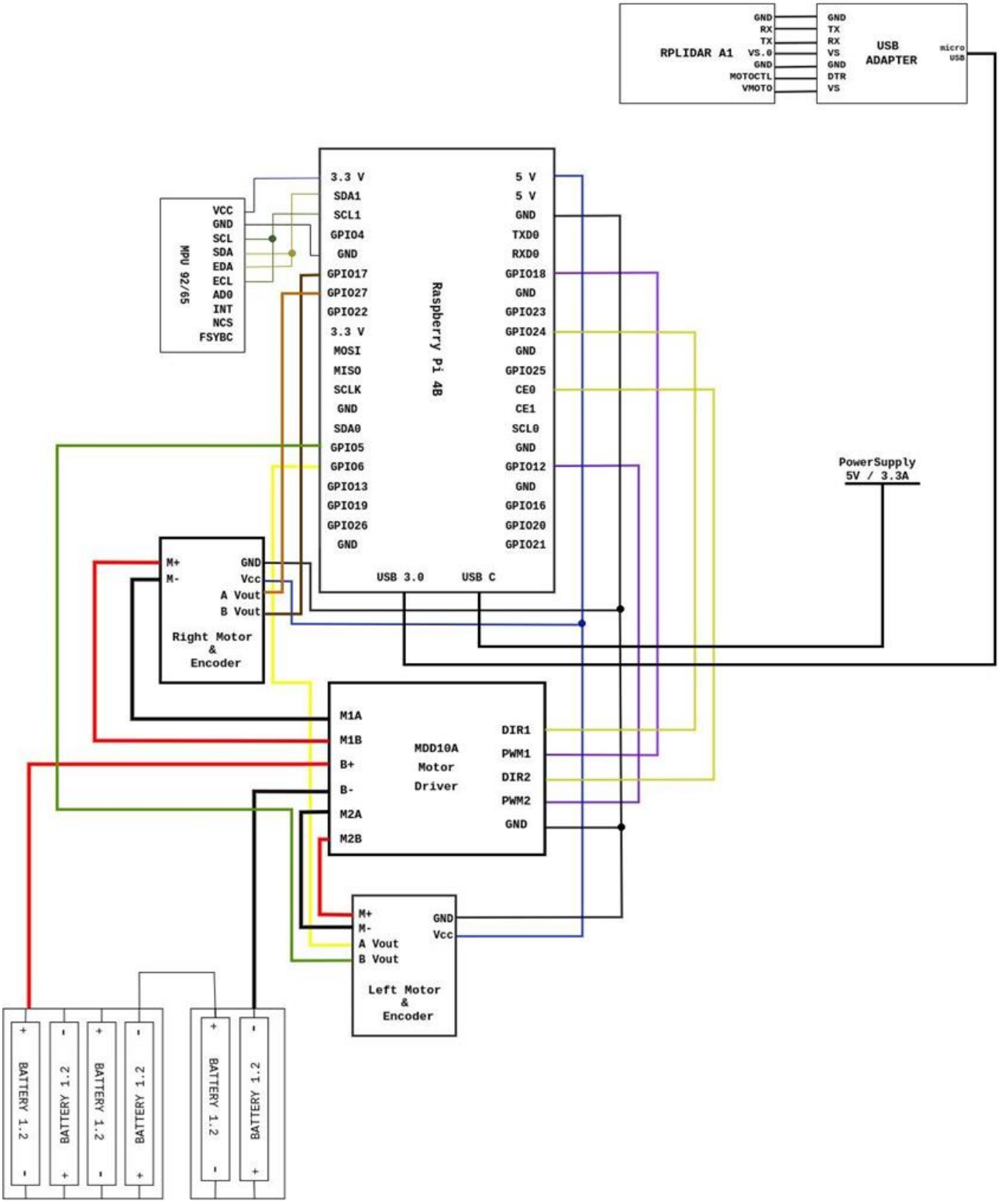
Βλέποντας τις ενεργειακές ανάγκες των εξαρτημάτων παρατηρούμε ότι μόνο οι ρόδες χρειάζονται υψηλότερο τάσης φορτίο για να λειτουργήσουν ενώ τα υπόλοιπα μέρη περιορίζονται στα 5V. Για τον λόγο αυτό, χρησιμοποιήθηκε ένα απλό powerbank με χωρητικότητα 10000mhA και έξοδο 5V/3.3A για να τροφοδοτήσει το Raspberry Pi και όλα τα περιφερειακά εξαρτήματα, ενώ για τους τροχούς συνδέθηκαν παράλληλα 6 επαναφορτιζόμενες μπαταρίες 1.2V δίνοντας συνολικά φορτίο 8.2V και 1900mhA.Κατά τη διάρκεια της ανάπτυξης και των δοκιμών του οχήματος χρησιμοποιήθηκε ένα απλό DC τροφοδοτικό, καθώς οι απλές αλκαλικές μπαταρίες αποφορτίζονται πολύ γρήγορα και είχαν απογοητευτική απόδοση. Για τον λόγο αυτό, στο τελικό σχέδιο χρησιμοποιήθηκαν επαναφορτιζόμενες μπαταρίες NiMh που ήταν άμεσα διαθέσιμες στο εμπόριο.

Αξίζει εδώ να αναφέρουμε πως ενώ οι τροχοί είναι σχεδιασμένοι για απόδοση έως 333 RPMs στα 12V, για τις ανάγκες του συγκεκριμένου συστήματος δεν χρειαζόμαστε υψηλές ταχύτητες για την πλοήγηση. Οι ρόδες μπορούν να

λειτουργήσουν μέχρι και με 1V ωστόσο για να αποδώσουν τις απαιτούμενες ταχύτητες μετρήσαμε εμπειρικά ότι από 6V και πάνω οποιοδήποτε τάση φορτίου είναι ικανοποιητική. Ένα άλλο σημείο αναφοράς είναι η χωρητικότητα του powerbank/τροφοδοτικών που είναι σημαντική για την διάρκεια λειτουργίας του οχήματος. Εφόσον η συγκεκριμένη εργασία εστιάζει στην εξερεύνηση εσωτερικών κλειστών χώρων θέτουμε κατά προσέγγιση περιορισμό ότι το όχημα δεν θα πλοηγείται για περισσότερο από μια ώρα, το οποίο από εμπειρικές δοκιμές ήταν παραπάνω από αρκετό.

## 3.5 Συνδεσμολογία

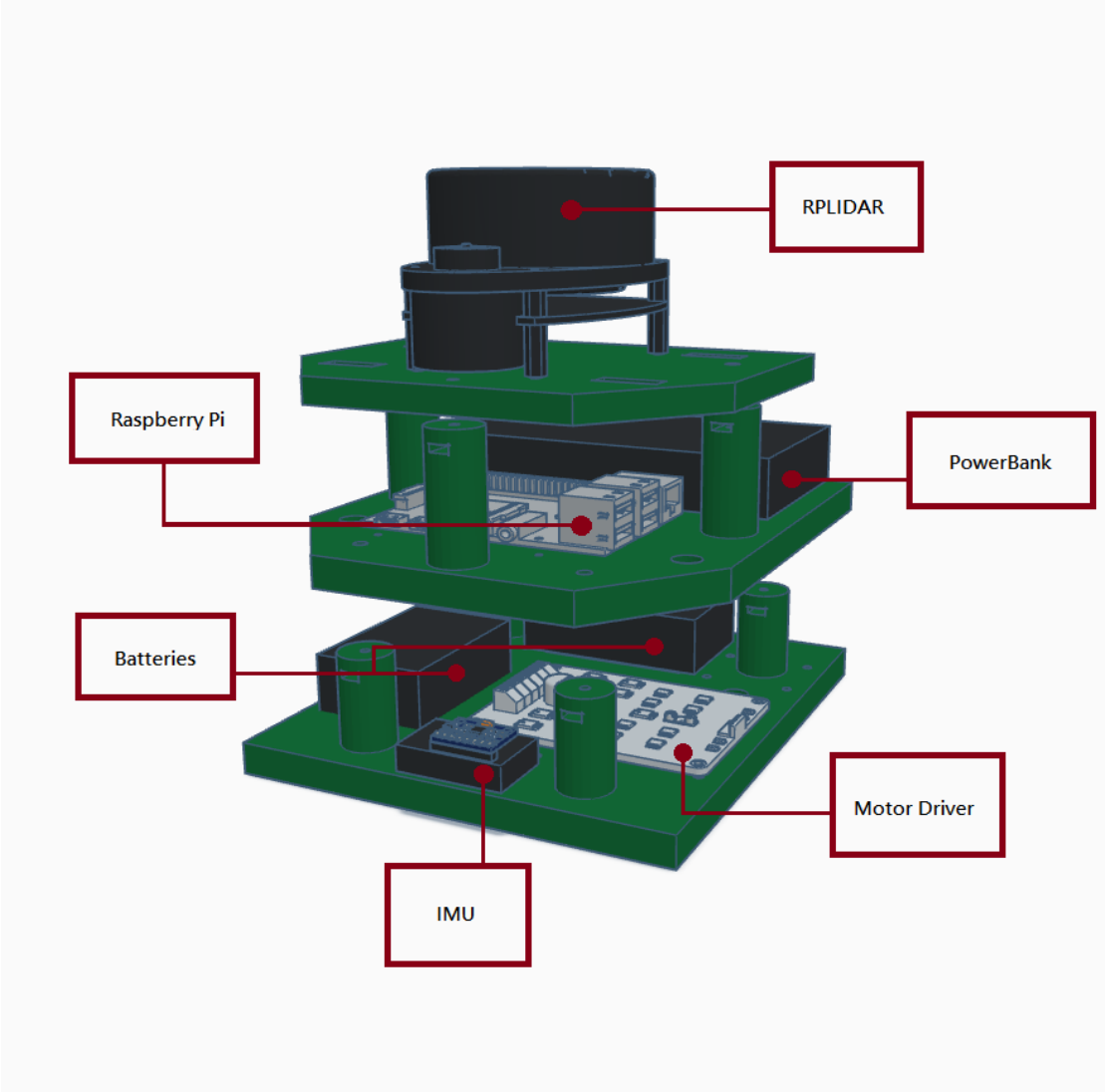
Το Raspberry Pi είναι η κεντρική υπολογιστή μονάδα του συστήματος που ελέγχει και παρέχει την απαραίτητη ενέργεια για την λειτουργία των αισθητήρων. Χρησιμοποιούμε τις I2C εξόδους για να επικοινωνήσουμε με το γυροσκόπιο, την usb 3.0 θύρα για το LIDAR και τα υπόλοιπα GPIO pins για να διαβάσουμε τους κωδικοποιητές των τροχών. Στο παρακάτω διάγραμμα απεικονίζονται αναλυτικά οι συνδέσεις μεταξύ των εξαρτημάτων.




**Εικόνα 9: Συνδεσμολογία**

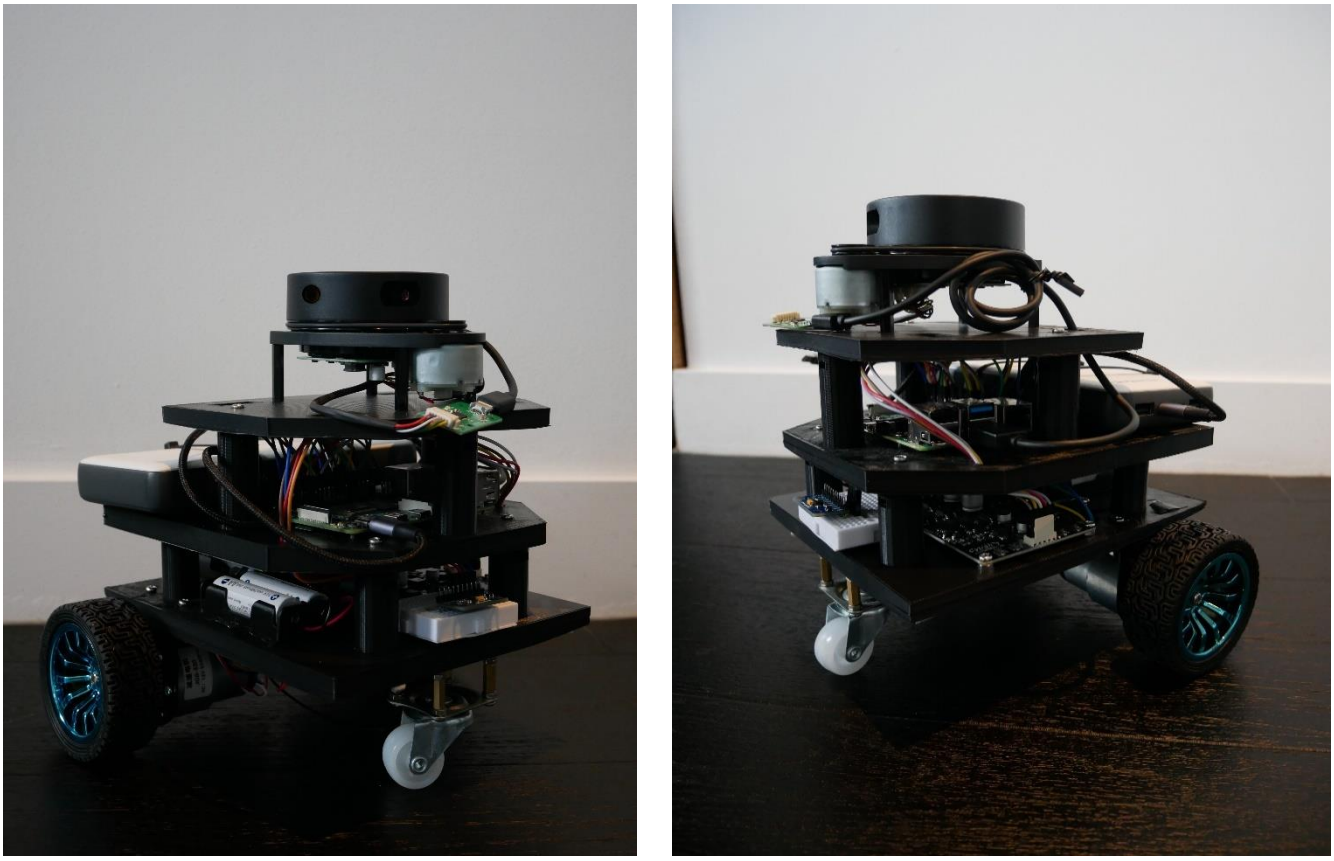
## 3.6 Κατασκευή Σκελετού

Για την κατασκευή του σκελετού του αυτοκινούμενου οχήματος χρησιμοποιήθηκε τρισδιάστατος εκτυπωτής Ender 3 - Pro με γυάλινη βάση για ακριβέστερα αποτελέσματα εκτύπωσης. Χωρίζουμε το όχημα σε τρία επίπεδα, καθώς στόχος είναι να έχουμε ένα μικρό συγκροτημένο διαφορικό όχημα που θα μπορεί να πλοηγείται σε κλειστούς χώρους με εμπόδια αλλά και να μεταφέρεται εύκολα. Παρατίθεται παρακάτω το σχέδιο:




**Εικόνα 10: Σχέδιο για τρισδιάστατη εκτύπωση**

Στο κατώτερο επίπεδο, στο πίσω μέρος της βάσης, τοποθετούμε τους δυο DC τροχούς και προσθέτουμε ένα βοηθητικό μεταλλικό ροδάκι στο μπροστινό μέρος για καλύτερη στήριξη και έλεγχο, όταν το όχημα θα περιστρέφεται. Στο ίδιο επίπεδο τοποθετούμε τις επαναφορτιζόμενες μπαταρίες, τον Motor Driver και το γυροσκόπιο. Στο επόμενο επίπεδο βάζουμε το Raspberry Pi με το powerbank. Το LIDAR τοποθετείται στην κορυφή ώστε να έχει ελεύθερο πεδίο για τις ριπές laser, χωρίς να παρεμποδίζεται από τα υπόλοιπα εξαρτήματα. Επειδή το συγκεκριμένο powerbank ζυγίζει σχεδόν το τριπλάσιο από το LIDAR, φροντίζουμε να μοιράσουμε ομοιόμορφα το βάρος, τοποθετώντας το στο πίσω μέρος του δεύτερου επιπέδου ενώ το τρίτο επίπεδο στερεώνεται αντί διαμετρικά.



**Εικόνα 11: Ολοκληρωμένη κατασκευή ρομπότ**

# ΚΕΦ.4: Υλοποίηση

## 4.1 Περιγραφή Ρομπότ στον Τρισδιάστατο Χώρο

Όσο σημαντική και αν είναι η συλλογή και η εγκυρότητα των δεδομένων από τους αισθητήρες , δεν μπορούν να αξιοποιηθούν σωστά εάν το ρομπότ δεν γνωρίζει που βρίσκεται το ίδιο στον χώρο αλλά και που βρίσκονται τα υπόλοιπα εξαρτήματά του σε σχέση με αυτό. Για παράδειγμα έστω μία ρομποτική πλατφόρμα εξοπλισμένη με έναν LIDAR αισθητήρα. Ο αισθητήρας ανιχνεύει τον χώρο και εντοπίζει έναν τοίχο σε απόσταση δύο μέτρων, αλλά αυτή η πληροφορία δεν είναι αρκετή ώστε το σύστημα να σχεδιάσει μία νέα τροχιά για να αποφύγει το εμπόδιο. Και αυτό διότι θα μπορούσε η πλατφόρμα να έχει μήκος ενάμιση μέτρο και ο αισθητήρας να είχε τοποθετηθεί στο πίσω μέρος της, δίνοντας στο ρομπότ μόνο μισό μέτρο ελεύθερου χώρου για να στρίψει. Όσο περισσότεροι αισθητήρες, ρομποτικοί βραχίονες και εξαρτήματα τοποθετούνται στο σύστημα τόσο πιο περίπλοκη γίνεται η μαθηματική περιγραφή του σε έναν τρισδιάστατο χώρο. Το λογισμικό ROS2 είναι συμβατό με μία συλλογή βιβλιοθηκών, τα robot_state_publisher, joint_state_publisher και tf2, που έχουν σχεδιαστεί για να διευκολύνουν την λύση του παραπάνω προβλήματος.

### 4.1.1 Βραχίονες και Κόμβοι Ρομποτικού Μοντέλου URDF

Αρχικά, είναι σημαντικό να γίνει διαχωρισμός της κατασκευής σε δύο κατηγορίες, τους βραχίονες (links) και τους κόμβους ή άκρα (joints). Κάθε βραχίονας συνδέει τα άκρα του συστήματος μεταξύ τους και με τη βάση του σκελετού, ενώ οι κόμβοι ανάλογα με την ιδιότητα τους, περιστρεφόμενοι ή στατικοί, καθορίζουν την κίνηση των βραχιόνων. Για την καλύτερη κατανόηση των εννοιών, μπορούμε να παρομοιάσουμε την χρήση των βραχιόνων και κόμβων με την ανθρώπινη ανατομία. Στο ανθρώπινο σώμα ο βραχίονας είναι το μπράτσο, δηλαδή το τμήμα του

ανώτερου άκρου που συνδέει δύο κόμβους, αυτούς του ώμου και του αρθρωτού συνδέσμου.

Πέρα από τους βραχίονες και τους κόμβους του συστήματος είναι σημαντικό να προσδιοριστούν και τα πλαίσια συντεταγμένων (coordinate frames) για κάθε εξάρτημα και η ιεραρχία τους. Τα πλαίσια συντεταγμένων είναι ένας τρόπος να περιγράψουμε μαθηματικά στον τρισδιάστατο χώρο την θέση όλων των εξαρτημάτων του συστήματος σε σχέση με τη βάση του. Στο προηγούμενο παράδειγμα της ρομποτικής πλατφόρμας με τον αισθητήρα LIDAR, η βάση του ρομπότ είναι το κεντρικό frame, ενώ το LIDAR είναι ο κόμβος παιδί (child frame) που βρίσκεται πενήντα εκατοστά πίσω από το κέντρο της πλατφόρμας.

Για να περιγράψουμε τις διαστάσεις, τους βραχίονες, τους κόμβους και άλλες φυσικές ιδιότητες του συστήματος όπως π.χ. η μάζα, χρησιμοποιούνται ειδικά αρχεία τύπου URDF (Unified Robot Description Format) σε XML μορφοποίηση. Το πακέτο Joint State Publisher που παρέχεται από το ROS λογισμικό είναι υπεύθυνο για την αναμετάδοση της θέσης και της ταχύτητας των κόμβων στους αντίστοιχους ROS κόμβους (topics), συγκεκριμένα στο sensor_msgs/JointState. Στη συνέχεια η βιβλιοθήκη robot state publisher χρησιμοποιεί τα δεδομένα που δημοσιεύονται στο sensor_msgs/JointState και τα urdf αρχεία για να υπολογίσει τη θέση και τον προσανατολισμό των βραχιόνων και κόμβων σε σχέση με το πλαίσιο συντεταγμένων του ρομπότ, τοποθετώντας τους δηλαδή σε σύγκριση με τους άξονες x, y, z του χώρου. Τέλος το πακέτο tf2 (transform library) χρησιμοποιώντας τα δεδομένα από το Robot State Publisher, διατηρεί τη σχέση μεταξύ των πλαισίων συντεταγμένων σε μια δενδρική δομή κατά τη διάρκεια του χρόνου και επιτρέπει στο χρήστη να μετασχηματίζει σημεία, διανύσματα κ.λπ. μεταξύ δύο οποιωνδήποτε πλαισίων συντεταγμένων σε οποιαδήποτε επιθυμητή χρονική στιγμή. Είναι σημαντικό η δενδρική tf2 δομή να είναι αντιπροσωπευτική της πραγματικής κατασκευής, καθώς η φυσική περιγραφή του συστήματος παίζει καθοριστικό ρόλο στην χρήση των βιβλιοθηκών πλοήγησης και SLAM. Επίσης είναι σημαντικό να αναφερθεί ότι στο URDF αρχείο εκτός από τις φυσικές διαστάσεις του συστήματος

έχουν προστεθεί και οι μετρήσεις μάζας και αδράνειας που βοηθούν την βιβλιοθήκη πλοήγησης να υπολογίσει την επιθυμητή ταχύτητα. Για το σύστημα που περιγράφεται στην παρούσα εργασία έχουμε ορίσει τα παρακάτω πλαίσια συντεταγμένων και βραχίονες/κόμβους:

**Map Frame (πλαίσιο χάρτη):**

Το συγκεκριμένο πλαίσιο συντεταγμένων είναι ένα παγκόσμιο σταθερό πλαίσιο, με τον άξονα Z να δείχνει πάντα προς τα πάνω και δημοσιεύεται/ανανεώνεται από τις βιβλιοθήκες χαρτογράφησης όπως το slam toolbox. Οι βιβλιοθήκες χαρτογράφησης είναι επίσης υπεύθυνες για την δημοσίευση της συσχέτισης map και odom πλαισίων.

**Odom Frame (πλαίσιο οδομετρίας):**

Το πλαίσιο odom είναι και αυτό ένα παγκόσμιο σταθερό πλαίσιο στον χώρο. Οι συντεταγμένες του είναι ίδιες με της θέσης που βρίσκεται το ρομπότ όταν αρχικοποιείται. Δημοσιεύεται και ανανεώνεται από την βιβλιοθήκη εντοπισμού θέσης (localization) και υπολογίζεται βάσει των δεδομένων που συλλέγονται από τους τροχούς και το γυροσκόπιο/επιταχυνσιόμετρο.

**Base Footprint Frame:**

Πλαίσιο συντεταγμένων που βρίσκεται ακριβώς κάτω από το κέντρο της ρομποτικής πλατφόρμας. Περιγράφει την δισδιάστατη θέση του ρομπότ στο έδαφος (z = 0) και κινείται στον χώρο καθώς κινείται το ρομπότ.

**Base Link Frame:**

Το πλαίσιο συντεταγμένων base_link αντιπροσωπεύει την βάση του ρομπότ και αρχικοποιείται ακριβώς στο σημείο περιστροφής ή αλλιώς στο κέντρο του ρομπότ. Το πλαίσιο συντεταγμένων κινείται καθώς κινείται το ρομπότ.

**Lidar Link Frame:**

Το laser_link έχει την αφετηρία του στο κέντρο του αισθητήρα LIDAR. Αυτό το πλαίσιο συντεταγμένων παραμένει στατικό σε σχέση με το base_link και συνδέεται

μαζί του μέσω του βραχίονα lidar_joint που περιγράφει την απόσταση του από το κέντρο του ρομπότ.

**Imu Link Frame:**

Το imu_link έχει την αφετηρία του στο κέντρο του αισθητήρα γυροσκοπίου/επιταχυνσιόμετρου. Αυτό το πλαίσιο συντεταγμένων παραμένει στατικό σε σχέση με το base_link και συνδέεται μαζί του μέσω του βραχίονα imu_joint που περιγράφει την απόσταση του από το κέντρο του ρομπότ.

**Wheel Link:**

Οι περιστρεφόμενοι κόμβοι drivewhl_l_link και drivewhl_r_link περιγράφουν τα άκρα των δύο τροχών τοποθετημένους στις δύο πλευρές του οχήματος. Η σύνδεση τους με την βάση του ρομπότ, base_link, καθορίζεται από τους βραχίονες drivewhl_l/drivewhl_r_joint αντίστοιχα για κάθε τροχό.

**Front Caster Link:**

Ο περιστρεφόμενος κόμβος front_caster που περιγράφει την μεταλλική υποστηρικτική ρόδα του ρομπότ. Διαθέτει ξεχωριστό βραχίονα caster_joint που καθορίζει την σχέση του με το κέντρο της ρομποτικής βάσης base_link.

Για να χρησιμοποιηθούν οι βιβλιοθήκες που αναφέρθηκαν πρέπει πρώτα να εγκατασταθούν στο Ubuntu λειτουργικό σύστημα. Επειδή το Raspberry Pi έχει προσαρμοστεί ως διακομιστής, δηλαδή χωρίς περιβάλλον διεπαφής χρήστη, πρέπει να εγκατασταθούν οι εκδόσεις των λειτουργικών χωρίς γραφικά στοιχεία τρέχοντας:

```
sudo apt-get install ros-foxy-robot-state-publisher

sudo apt install ros-foxy-joint-state-publisher
```

Ενώ για να κληθούν οι βιβλιοθήκες όταν ξεκινήσει και η ρομποτική πλατφόρμα πρέπει να τοποθετηθούν στο launch αρχείο:

**Πίνακας 2: Αρχικοποίηση URDF**

```
default_model_path = os.path.join(pkg_share, 'description/rpi_robot.urdf')
robot_name_in_urdf = 'rpi_robot'
default_rviz_config_path = os.path.join(pkg_share,'rviz/urdf_config.rviz')

robot_state_publisher_node = launch_ros.actions.Node(,

package='robot_state_publisher',

executable='robot_state_publisher',

parameters=[{'robot_description': Command(

['xacro ', LaunchConfiguration('model')])}],

remappings=remappings)

joint_state_publisher_node = launch_ros.actions.Node(

package='joint_state_publisher',

executable='joint_state_publisher',

name='joint_state_publisher')
```

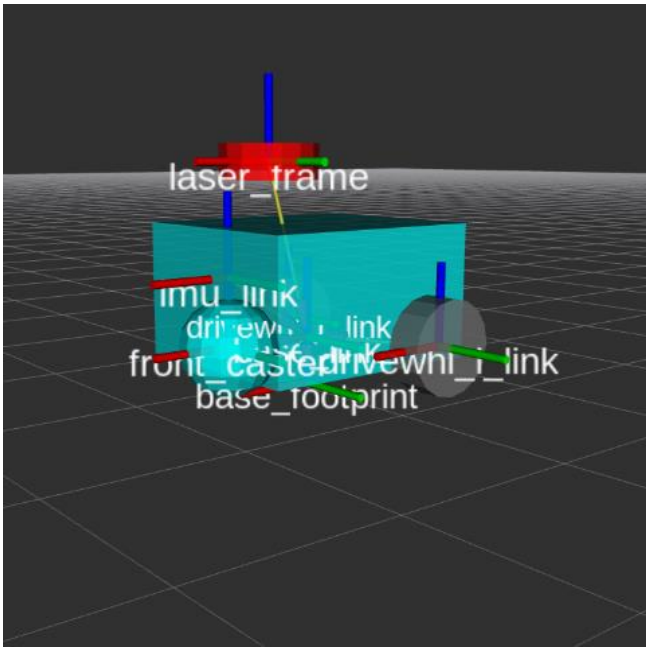
Παρακάτω παρατίθεται και ο κώδικας του URDF μοντέλου:

```
<?xml version="1.0"?>

<robot name="rpi_robot" xmlns:xacro="http://ros.org/wiki/xacro">

  <!-- ****************** ROBOT CONSTANTS ******************************  -->

  <!-- Define the size of the robot's main chassis in meters -->

  <xacro:property name="base_width" value="0.15" />

  <xacro:property name="base_length" value="0.17" />

  <xacro:property name="base_height" value="0.10" />

  <!-- Define the shape of the robot's two back wheels in meters -->

  <xacro:property name="wheel_radius" value="0.0355" />

  <xacro:property name="wheel_width" value="0.026" />

  <xacro:property name="wheel_ygap" value="0.01" />

  <xacro:property name="wheel_zoff" value="0" />

  <xacro:property name="wheel_xoff" value="0.05" />

  <!-- Position the caster wheel along the x-axis -->

  <xacro:property name="caster_xoff" value="0.07" />

  <xacro:property name="caster_radius" value="0.0355" />

  <!-- Define some commonly used intertial properties  -->

  <xacro:macro name="box_inertia" params="m w h d">

    <inertial>

      <origin xyz="0 0 0" rpy="${pi/2} 0 ${pi/2}" />

      <mass value="${m}" />

      <inertia ixx="${(m/12) * (h*h + d*d)}" ixy="0.0" ixz="0.0" iyy="${(m/12) * (w*w + d*d)}"
iyz="0.0" izz="${(m/12) * (w*w + h*h)}" />

    </inertial>

  </xacro:macro>

  <xacro:macro name="cylinder_inertia" params="m r h">

    <inertial>

      <origin xyz="0 0 0" rpy="${pi/2} 0 0" />

      <mass value="${m}" />

      <inertia ixx="${(m/12) * (3*r*r + h*h)}" ixy="0" ixz="0" iyy="${(m/12) * (3*r*r + h*h)}"
iyz="0" izz="${(m/2) * (r*r)}" />

    </inertial>

  </xacro:macro>

  <xacro:macro name="sphere_inertia" params="m r">

    <inertial>

      <mass value="${m}" />

      <inertia ixx="${(2/5) * m * (r*r)}" ixy="0.0" ixz="0.0" iyy="${(2/5) * m * (r*r)}"
iyz="0.0" izz="${(2/5) * m * (r*r)}" />

    </inertial>

  </xacro:macro>

  <!-- ****************** ROBOT BASE FOOTPRINT **************************  -->

  <!-- Define the center of the main robot chassis projected on the ground -->
```

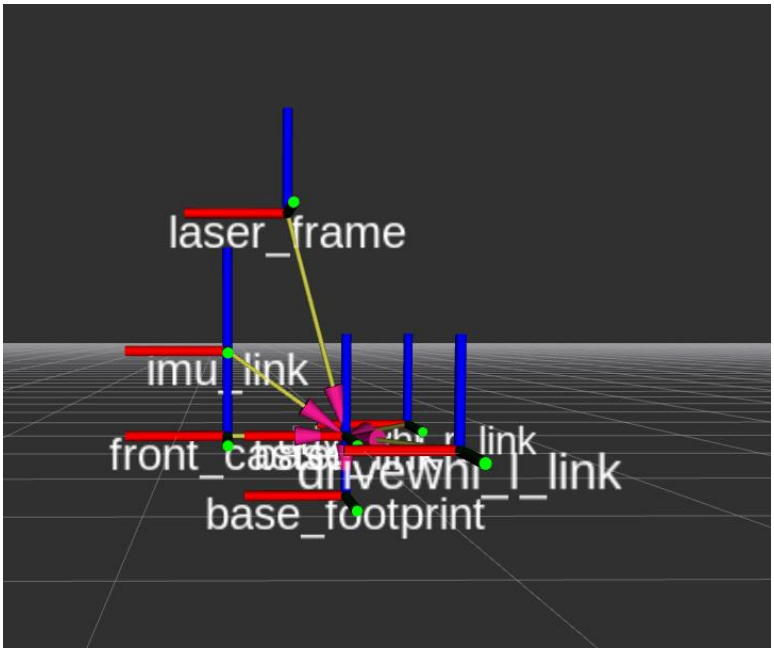
**Πίνακας 3: Urdf Μοντέλο**

## 4.1.2 Απεικόνιση Μοντέλων και RVIZ2

Εάν τρέξουμε τα παραπάνω κομμάτια κώδικα θα δούμε άμεσα τα αποτελέσματα στο RVIZ2 όπως φαίνονται και στις παρακάτω εικόνες. Μπορούμε να δούμε τους βραχίονες «γυμνούς» που είναι εμφανείς (Εικόνα 13) και αν προσθέσουμε και το topic urdf model στο RVIZ2 θα δούμε και την προσομοίωση της φυσικής υπόστασης του ρομπότ (Εικόνα 12). Είναι πολύ σημαντικό σε κάθε πρόγραμμα ρομποτικής να καθορίζονται σωστά οι κομβοι/άκρα και οι βραχίονες, διότι ακόμα και αν είχαμε ιδανικά δεδομένα χωρίς κανέναν απολύτως θόρυβο και τον τέλειο αλάθητο αλγόριθμο να τα ελέγχει, αν ο σκελετός δεν είναι σωστά ορισμένος δεν θα καταφέρουμε να εκτελέσουμε SLAM με επιτυχία.



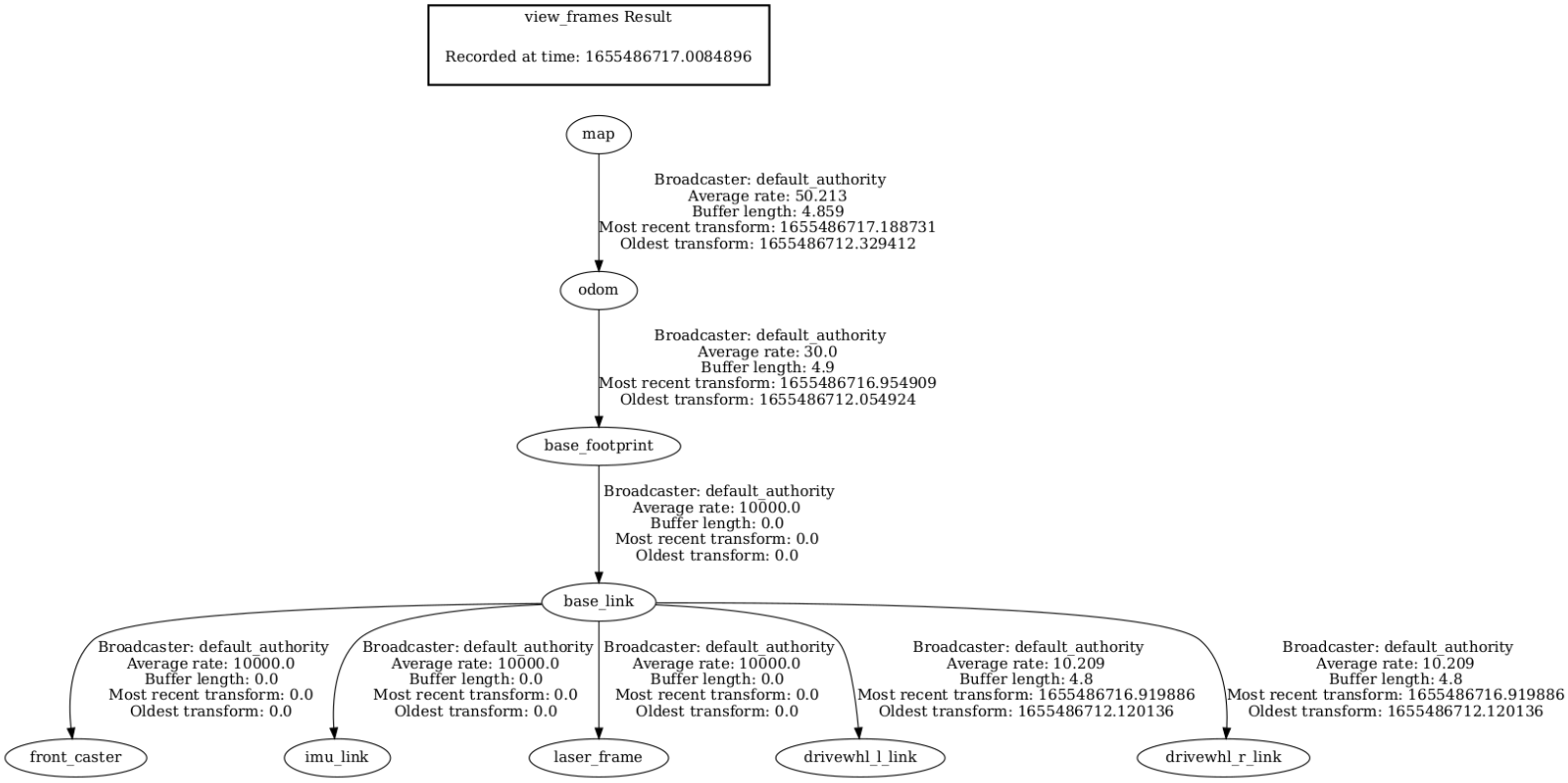

**Εικόνα 12: Μοντέλο προσομοίωσης**




**Εικόνα 13: Βραχίονες και άκρα του ρομποτικού συστήματος**

Άλλο ένα χρήσιμο εργαλείο για την απεικόνιση και την επαλήθευση των πλαισίων συντεταγμένων και των μετασχηματισμών τους είναι η tf_tools βιβλιοθήκη. Εκτελώντας το tf2_view μπορούμε να δούμε όλα τα links/tf που δημοσιεύονται και την ιεραρχία τους. Στη συγκεκριμένη εργασία όταν τρέξουμε όλα τα πακέτα

λογισμικού το tf view παράγει το παρακάτω ιεραρχικό δέντρο που έρχεται σε απόλυτη συμφωνία με το urdf μοντέλο που ορίσαμε παραπάνω καθώς και με την αρχιτεκτονική που




σχεδιάσαμε στο κεφάλαιο 3.2 [35].

**Εικόνα 14: Ιεραρχικό δέντρο μετασχηματισμών**

## 4.2 Οδομετρία - Odometry

Ένα από τα βασικότερα προβλήματα σε οποιοδήποτε σύστημα αυτόνομης πλοήγησης, είναι η δυνατότητα αυτοπροσδιορισμού της θέσης του οχήματος στον χώρο. Εάν το ρομπότ δεν διαθέτει την απαραίτητη πληροφορία μπορεί να κινείται στον χώρο χωρίς να γνωρίζει τον τελικό προορισμό ή την κατεύθυνση. Ορίζουμε ως οδομετρία την ικανότητα του οχήματος να χρησιμοποιεί τα δεδομένα των αισθητήρων του ώστε να υπολογίζει την απόσταση που έχει διανύσει από την αρχική του θέση αλλά και τον προσανατολισμό του. Στα πλαίσια αυτή της εργασίας δυο βασικοί αισθητήρες συγχωνεύονται για τον υπολογισμό θέσης, το γυροσκόπιο-επιταχυνσιόμετρο και οι τροχοί. Συλλέγοντας τα δεδομένα από τους δύο αισθητήρες, το όχημα μπορεί να εντοπίσει την ακριβή του τοποθεσία στον χώρο και να ανανεώσει τον χάρτη όσο κινείται. Στα παρακάτω κεφάλαια αναλύονται οι τεχνολογίες και οι μέθοδοι που χρησιμοποιήθηκαν για την υλοποίηση του οχήματος.

### 4.2.1 Κινητήρες Συνεχούς Ρεύματος – DC Motors

Οι κινητήρες συνεχούς ρεύματος είναι ηλεκτρικές μηχανές που μετατρέπουν την ηλεκτρική ενέργεια σε μηχανική, δημιουργώντας ένα μαγνητικό πεδίο που τροφοδοτείται από συνεχές ρεύμα. Βασικό πλεονέκτημα τους αποτελεί ο εύκολος και ακριβής έλεγχος της ροπής και της ταχύτητάς τους σε ένα μεγάλο εύρος τιμών. Οι κινητήρες συνεχούς ρεύματος έχουν την ικανότητα να σταματούν, να ξεκινούν, να αλλάζουν την κατεύθυνση τους άμεσα, καθιστώντας τους ιδανικούς για το σύστημα που περιγράφεται στην παρούσα εργασία. Παρόλα αυτά, η χρήση απλών κινητήρων για το αυτοκινούμενο όχημα δεν είναι αρκετή. Το σύστημα πρέπει να γνωρίζει πόση απόσταση έχει διανύσει σε διάρκεια χρόνου, εάν περιστρέφεται, την

ταχύτητα κίνησης κ.α. Για να επιτευχθεί αυτό είναι σημαντικό να επιλέξουμε τους κατάλληλους κωδικοποιητές (encoders).

### 4.2.2 Κωδικοποιητές

Οι κωδικοποιητές είναι αισθητήρες που είναι προσαρτημένοι πάνω στον τροχό και ανιχνεύουν τον αριθμό των περιστροφών. Υπάρχουν πολλοί διαφορετικοί κωδικοποιητές, για ποικιλία διαφόρων χρήσεων αλλά συνηθίζεται να κατηγοριοποιούνται σε τρεις μεγάλες ομάδες ανάλογα με το είδος κίνησης σε:

- Γραμμικούς (Rotary Encoders)
  Οι γραμμικοί κωδικοποιητές χρησιμοποιούνται σε κατασκευές που κινούνται κατά μήκος μιας διαδρομής ή γραμμής. Χρησιμοποιούν μετατροπείς για να μετρήσουν την απόσταση δύο σημείων, με τη βοήθεια καλωδίων ή ράβδων.

- Περιστροφικούς (Rotary Encoders)
  Οι περιστροφικοί κωδικοποιητές χρησιμοποιούνται για την παροχή πληροφορίας σχετικά με την κίνηση περιστρεφόμενων αντικειμένων ή αξόνων. Ο περιστροφικός κωδικοποιητής μετατρέπει τη γωνιακή θέση ενός κινούμενου άξονα σε αναλογικό ή ψηφιακό σήμα εξόδου.

- Γωνίας (Angle Encoders)
  Οι κωδικοποιητές γωνίας μοιάζουν στην χρήση τους με τους περιστροφικούς αλλά εστιάζουν πιο πολύ στην μέτρηση της γωνίας που σχηματίζει ο περιστρεφόμενος άξονας. Συνήθως είναι οπτικοί.

Οι γραμμικοί και οι περιστρεφόμενοι κωδικοποιητές μπορούν επίσης να χωριστούν ανάλογα με τον τύπο εξόδου σε δύο ακόμα κατηγορίες:

- Επαυξητικός (incremental)

  Οι επαυξητικοί κωδικοποιητές χρησιμοποιούν παλμούς για να υπολογίσουν την θέση του τροχού. Κάθε περιστροφή παράγει έναν καθορισμένο αριθμό παλμών.

- Απόλυτους (Absolute)

  Στους απόλυτους κωδικοποιητές, το σήμα εξόδου που παράγεται από τη συσκευή καταλήγει σε ένα μοναδικό σύνολο δυαδικών ψηφίων που αντιστοιχούν σε μια συγκεκριμένη θέση του αντικειμένου που μετριέται.

Και τέλος, ανάλογα με τον σχεδιασμό τους οι κωδικοποιητές διακρίνονται σε:

- Οπτικοί (Optical)

  Ένας περιστρεφόμενος οπτικός κωδικοποιητής αποτελείται από μια πηγή φωτός, και έναν περιστρεφόμενο δίσκο που φέρει μια σειρά γραμμών και εναλλασσόμενων ημιδιαφανών σχισμών. Καθώς το φως διέρχεται μέσα από τον περιστρεφόμενο δίσκο, παράγει ένα ηλεκτρικό σήμα που ανιχνεύεται από τις σχισμές στον δίσκο.

- Μαγνητικοί (Magnetic)

  Οι μαγνητικοί κωδικοποιητές βασίζονται στην ανίχνευση της μεταβολής της μαγνητικής ροής για τον προσδιορισμό της κίνησης και της θέσης ενός αντικειμένου. Ο αισθητήρας που χρησιμοποιείται σε αυτούς τους κωδικοποιητές μπορεί είτε να κάνει χρήση του φαινομένου Hall, το οποίο ανιχνεύει μια αλλαγή στην τάση, είτε να είναι ένας μαγνητοαντιστατικός αισθητήρας που μπορεί να ανιχνεύσει απευθείας την αλλαγή στο μαγνητικό πεδίο.

- Χωρητικότητας (Capacitive)

  Οι κωδικοποιητές χωρητικότητας βασίζονται στην ανίχνευση της μεταβολής της «χωρητικότητας» με τη χρήση ενός σήματος υψηλής συχνότητας. Συγκεκριμένα διαθέτουν έναν κινητήρα που είναι χαραγμένος με ημιτονοειδές μοτίβο, και καθώς αυτός κινείται, το σήμα υψηλής συχνότητας που το χτυπάει διαμορφώνει ένα νέο σήμα με προβλέψιμες διαμορφώσεις.

Ο δέκτης διαβάζει τις διαμορφώσεις και τα ενσωματωμένα ηλεκτρονικά συστήματα τις μεταφράζουν σε βήματα περιστροφικής κίνησης.

### 4.2.3 Επιλογή Κωδικοποιητών

Αποκλείστηκαν οι γραμμικοί κωδικοποιητές εφόσον το όχημα πρέπει κινείται ελεύθερα χωρίς να ακολουθεί κάποια αυστηρή γραμμή και οι κωδικοποιητές γωνίας μιας και δεν θέλουμε να εστιάσουμε στον έλεγχο της γωνίας του άξονα κατά την κίνηση του. Για τον ίδιο λόγο απορρίπτονται οι απόλυτοι κωδικοποιητές, μιας και χρησιμοποιούνται πιο συχνά για τον ακριβή προσδιορισμό της γωνίας του τροχού ενώ για την υλοποίηση του συστήματος μεγαλύτερη σημασία έχει ο υπολογισμός της απόστασης που έχει διανείμει το όχημα. Επίσης, αποκλείουμε τους κωδικοποιητές χωρητικότητας εφόσον η τεχνολογία σχεδιασμού τους είναι αρκετά καινούρια επομένως βρίσκονται σε περιορισμένη διαθεσιμότητα κόστος και διαθέτουν υψηλό κόστος. Όσον αφορά τους οπτικούς κωδικοποιητές, είναι αρκετά ευαίσθητοι σε κραδασμούς και εφόσον πρόκειται να χρησιμοποιηθούν σε αυτοκινούμενο όχημα, θα προτιμηθεί μια πιο σταθερή λύση, αυτή των μαγνητικών κωδικοποιητών. Εν τέλει, οι δύο κινητήρες που επιλέχθηκαν για το όχημα διαθέτουν περιστρεφόμενους επαυξανόμενους κωδικοποιητές με δύο μαγνητικούς Hall αισθητήρες ο καθένας, που παράγουν έντεκα σήματα για κάθε περιστροφή.

### 4.2.4 Υπολογισμός Απόστασης με τη Χρήση Κωδικοποιητών

Η απόσταση που έχει διανείμει το όχημα μπορεί να υπολογισθεί συγκρίνοντας τον συνολικό αριθμό σημάτων που παράγει ο κινητήρας σε μία ολόκληρη περιστροφή με το άθροισμα των σημάτων που έχουν μετρηθεί σε ένα χρονικό διάστημα t. Αξίζει

επίσης εδώ να σημειωθεί ότι οι επιλεγμένοι τροχοί διαθέτουν και γρανάζια με αναλογία 1:30, που σημαίνει ότι όταν ολοκληρώσει μια ολόκληρη περιστροφή ο τροχός, ο άξονας του κωδικοποιητή έχει ολοκληρώσει ήδη 30, και αν κάθε Hall αισθητήρας παράγει έντεκα σήματα για κάθε περιστροφή αυτό σημαίνει ότι έχουμε 30* 11 = 330 παραγόμενα σήματα από κάθε Hall αισθητήρα (330 * 2 = 660 από κάθε τροχό) για μία πλήρη περιστροφή του κινητήρα.

Εάν για παράδειγμα μετράμε μόνο τον ένα από τους δύο αισθητήρες και υπολογίσουμε ότι παράχθηκαν 2640 σήματα τότε μπορούμε να πούμε ότι ο τροχός έκανε 2460 / 330 = 8 περιστροφές. Και αν σε μία περιστροφή ο τροχός έχει διανύσει 2π απόσταση, τότε μπορούμε να πούμε ότι σε αυτό το παράδειγμα διένυσε 16π.

```
encoder_diff = encoder_pulse_new - encoder_pulse_last;
nr_of_revolutions = encoder_diff / pulses_per_revolution;
distance_traveled = nr_of_revolutions * circumference;
```

### 4.2.5 Τεχνική Διαμόρφωσης Εύρους – Pulse Width Modulation

Η τεχνική διαμόρφωση εύρους είναι μια μέθοδος που χρησιμοποιείται για τον έλεγχο ισχύος ενός σήματος, αλλάζοντας το εύρος του κύματος ενώ η συχνότητα διατηρείται σταθερή. Η τεχνική αυτή μας δίνει την δυνατότητα να ελέγξουμε την ταχύτητα των τροχών με τη βοήθεια του οδηγού κινητήρα. Ο οδηγός κινητήρα διαθέτει δύο ακροδέκτες, για κάθε ρόδα, έναν που παίρνει σαν είσοδο το PWM σήμα και ελέγχει την ταχύτητα του τροχού και έναν που δέχεται δυαδικό σήμα και καθορίζει την κατεύθυνση του οχήματος (θετικό σήμα για να κινείται το όχημα με την κατεύθυνση του ρολογιού, αρνητικό σήμα για να κινείται αντίθετα). Και τα δύο σήματα, παράγονται από το Raspberry Pi και μεταδίδονται μέσω των ακροδεκτών

GPIO που έχει. Το PWM σήμα που στέλνεται στον οδηγό καθορίζει την μέση τιμή της τάσης που φτάνει στις ρόδες. Η τιμή εξαρτάται από τον κύκλο λειτουργίας (duty cycle) του σήματος, δηλαδή για πόσο χρόνο το σήμα βρίσκεται στην θετική του φάση αναφορικά με την περίοδό του. Όσο μεγαλύτερος είναι ο κύκλος ζωής τόσο μεγαλύτερη είναι η τάση που τροφοδοτείται στους τροχούς και τόσο μεγαλύτερη είναι η ταχύτητα που αναπτύσσεται (η ταχύτητα είναι αναλογική της τάσης που δίνεται στο σύστημα). Εάν ο κύκλος λειτουργίας είναι μηδενικός, τότε η τάση μηδενίζεται και το όχημα σταματάει.

Θα χρησιμοποιηθεί το παρακάτω παράδειγμα για να εξηγήσουμε την τεχνική που χρησιμοποιήθηκε αναλυτικότερα. Έστω ότι έχουμε πηγή που τροφοδοτεί τον οδηγό κινητήρα και κατ' επέκταση τους τροχούς με 12V. Εάν δώσουμε σήμα με κύκλο λειτουργίας 50% στον οδηγό κινητήρα αυτό σημαίνει ότι για περίοδο T, το σήμα είναι θετικό για T/2 και μηδενικό για T/2. Πρακτικά για τον οδηγό κινητήρα αυτό σημαίνει ότι πρέπει να δώσει στους τροχούς το 50% της εισόδου ρεύματος, δηλαδή φορτίο τάσης 6V. Εάν θέλαμε να δώσουμε στους τροχούς 9V τότε το Raspberry Pi θα πρέπει να στείλει στον οδηγό κινητήρα σήμα με κύκλο λειτουργίας 75%.Για τον υπολογισμό του επιθυμητού κύκλου λειτουργίας, σημαντικό ρόλο παίζει και τι ταχύτητα θέλουμε να αναπτυχθεί στους τροχούς. Εάν για παράδειγμα ξέρουμε ότι στα 12V οι ρόδες κινούνται με 0.6m/s για να αλλάξουμε την ταχύτητα σε 0.2m/s θα πρέπει να δώσουμε PWM σήμα κύκλου λειτουργίας 30% δηλαδή 4V.

## 4.2.6 Ρομποτική Κίνηση και Διαφορική Κινηματική - Differential Kinematics

Η ρομποτική κινηματική εφαρμόζει αρχές της γεωμετρίας για τη μελέτη των κινηματικών αλυσίδων πολλών βαθμών ελευθερίας ρομποτικών συστημάτων. Στην παρούσα ενότητα αναλύεται το διαφορικό κινηματικό μοντέλο που εφαρμόστηκε κατά την ανάπτυξη του συστήματος. Η διαφορική κίνηση είναι ένα σύστημα κίνησης βασισμένο στην λειτουργία δύο αυτόνομων τροχών με δικούς τους κινητήρες που βρίσκονται ευθυγραμμισμένοι στον ίδιο άξονα. Η ονομασία αναφέρεται στο γεγονός ότι το διάνυσμα κίνησης του ρομπότ είναι το άθροισμα των ανεξάρτητων κινήσεων των τροχών. Για ευκολία στην διατήρηση της ισορροπίας σε πολλά διαφορικά συστήματα κίνησης προστίθενται επιπλέον τροχοί. Στο παρόν σύστημα χρησιμοποιήθηκε ένας επιπλέον μεταλλικός τροχός χωρίς κινητήρα που κινείται ελεύθερα για επιπλέον υποστήριξη. Σε ένα διαφορικό σύστημα οδήγησης οι δύο βασικοί τροχοί κίνησης μπορούν να κινούνται ανεξάρτητα είτε προς τα μπροστά είτε προς τα πίσω, με διαφορετικές ή ίσες ταχύτητες, δίνοντας την δυνατότητα στο όχημα είτε να κινείται σε ευθεία γραμμή ή να περιστρέφεται γύρω από ένα σημείο που βρίσκεται κατά μήκος του κοινού αριστερού και δεξιού άξονα των τροχών. Το σημείο που το ρομπότ περιστρέφεται γύρω από αυτό είναι γνωστό ως Κέντρο Στιγμιαίας Καμπύλωσης ή ICC Instantaneous Center of Curvature.[27]

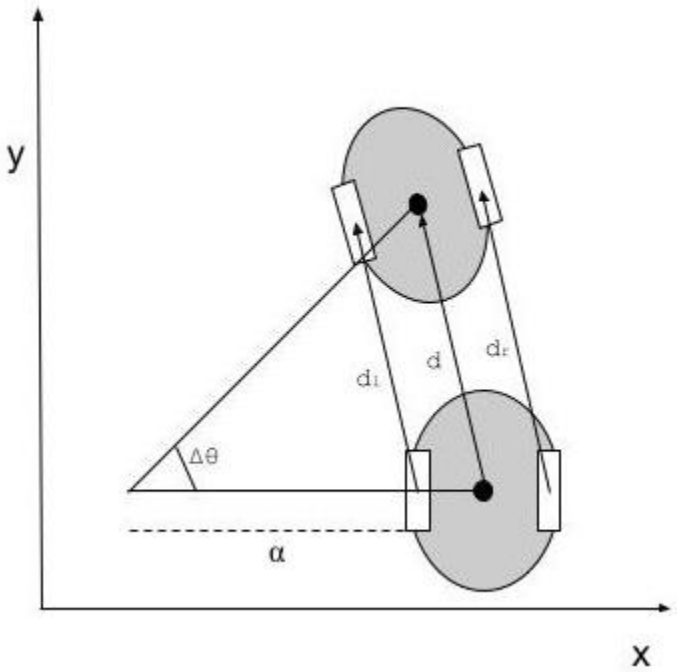
Μεταβάλλοντας τις ταχύτητες των δύο τροχών, μπορούμε να μεταβάλλουμε τις τροχιές που ακολουθεί το ρομπότ. Η κίνηση προς τα εμπρός επιτυγχάνεται όταν και οι δύο οι τροχοί κινούνται με τον ίδιο ρυθμό, η στροφή προς τα δεξιά επιτυγχάνεται με την κίνηση του αριστερού τροχού με μεγαλύτερη ταχύτητα από ό,τι του δεξιού τροχού και αντίστροφα για την αριστερή στροφή. Αυτός ο τύπος κινητού ρομπότ μπορεί να στρίψει επιτόπου οδηγώντας τον τροχό προς τα εμπρός και τον δεύτερο τροχό προς την αντίθετη κατεύθυνση με τον ίδιο ρυθμό. Τα

παραπάνω μεταφράζονται σε μαθηματικές εξισώσεις στις παρακάτω ενότητες.

## 4.2.7 Υπολογισμός απόστασης και ταχύτητας

Καθώς το όχημα κινείται στον χώρο διανύει απόσταση d από το σημείο A στο σημείο B και ταυτόχρονα οι κωδικοποιητές σε κάθε ρόδα συλλέγουν πληροφορίες, υπολογίζοντας την απόσταση που κάθε ρόδα διέσχισε. Έστω ότι ο δεξιός τροχός διέσχισε dr απόσταση και ο αριστερός dl ενώ ορίζουμε ως Rw την απόσταση των τροχών από το ICC σημείο.

Μπορούμε να εκφράσουμε την διαφορά γωνίας Δθ σε συνάρτηση με την απόσταση των τροχών από το κέντρο του οχήματος (R) ως:




$$dr = (\alpha + 2R)\Delta\theta$$

$$d = (\alpha + R)\Delta\theta$$

ή

(1) $\Delta\theta = (dr - dl)/\ 2R$

$$d = (\alpha + R)\Delta\theta$$

$$d = \alpha\Delta\theta + R\Delta\theta$$

$$d = dl + (dr - dl)/2$$

$$d = dl/2 + dr/2$$

(2) $d = (dl+dr)\ /\ 2$

**Εικόνα 15: Υπολογισμός ταχύτητας διαφορικού οχήματος**

Χρησιμοποιώντας τους τύπους **(1)** και **(2)** μπορούμε να εκφράσουμε τις συντεταγμένες και των δύο θέσεων σε ένα καρτεσιανό σύστημα. Οι συντεταγμένες εκτός από τις τιμές x και y θα πρέπει να περιγράφουν και τη διαφορά γωνίας, δηλαδή την κατεύθυνση της κίνησης Δθ. Εάν η αρχική θέση A εκφράζεται ως A = [ x y θ], τότε η θέση B = A + [dcos(θ) dsin(θ) Δθ] εφόσον μπορούμε να πούμε πάλι μέσω τριγωνομετρίας Δx=dcos(θ) και Δy=dsin(θ).

Με τον ίδιο τρόπο μπορούμε να υπολογίσουμε και την ταχύτητα του συστήματος. Καθώς το όχημα κινείται στο χώρο αναπτύσσει μια γραμμική και μια γωνιακή ταχύτητα. Θεωρώντας ότι το όχημα ξεκίνησε από την αρχική θέση A και κατέληξε στη B με νέες συντεταγμένες και προσανατολισμό μπορούμε να εκφράσουμε την ταχύτητα v ως:

$$v = (vl + vr)/2$$
$$\theta' = r/2Rw(vr - vl)$$

Επομένως εκφράζοντας την νέα θέση του οχήματος ως:

$$[\,x'\ y'\ \theta] = [v\cos\theta \quad v\sin\theta \quad r(vr\text{-}vl)/2Rw]$$

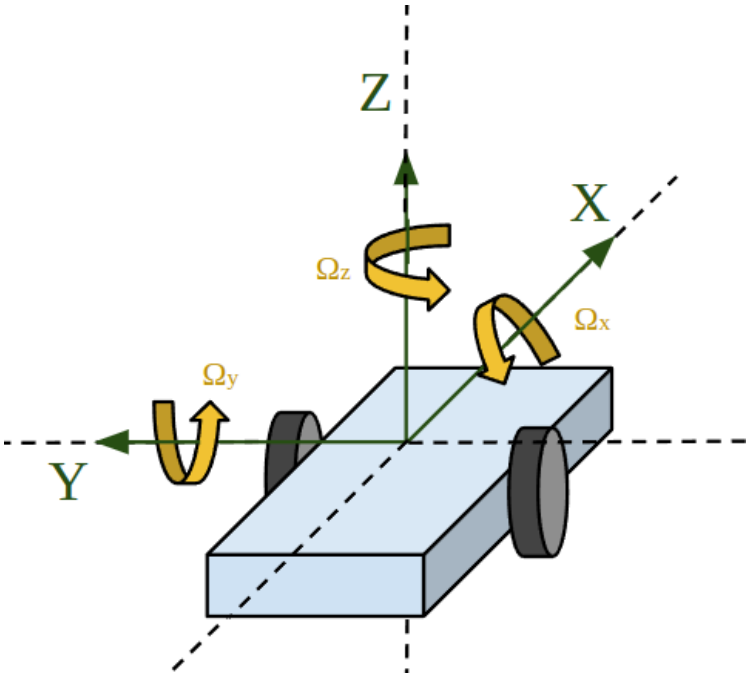
ή

$$[\,x'\ y'\ \theta] = [(vl+vr)\cos\theta/2 \quad (vl+vr)\sin\theta/2 \quad r(vr\text{-}vl)/2Rw]$$

Λαμβάνοντας τις μετρήσεις από τους κωδικοποιητές μπορούμε να υπολογίσουμε την απόσταση που έχει διανύσει το όχημα σε χρόνο t και την ταχύτητα του, και χρησιμοποιώντας τους παραπάνω τύπους να εκφράσουμε την θέση του στον χώρο και να υπολογίσουμε την οδομετρία του.

## 4.3 Αδρανειακή Μονάδα - IMU (Inertial Measurement Unit)

Οι αδρανειακές μονάδες είναι ηλεκτρονικές συσκευές που συνήθως συνδυάζουν γυροσκόπια, επιταχυνσιόμετρα και μαγνητόμετρα για να μετρούν την γωνιακή ταχύτητα, την δυναμική επιτάχυνση και τον προσανατολισμό του συστήματος. Στα πλαίσια αυτής της πτυχιακής χρησιμοποιήθηκε μια αδρανειακή μονάδα εξοπλισμένη με γυροσκόπιο και επιταχυνσιόμετρο. Το επιταχυνσιόμετρο είναι απαραίτητο για τη μέτρηση των δυναμικών και στατικών επιταχύνσεων. Οι δυναμικές επιταχύνσεις είναι ανομοιόμορφες και προκαλούνται από αλλαγές στην ταχύτητα ή στην κατεύθυνση του οχήματος ή ακόμα και από κραδασμούς και δονήσεις. Οι στατικές επιταχύνσεις είναι σταθερές δυνάμεις που ασκούνται στο σώμα όπως είναι η βαρυτική ή η τριβή. Το γυροσκόπιο είναι υπεύθυνο για τον υπολογισμό της γωνιακής ταχύτητας και του προσανατολισμού κίνησης του οχήματος. Στην παρακάτω εικόνα απεικονίζονται οι δυνάμεις που ασκούνται γύρω από τους άξονες στο όχημα. Οι drivers του συστήματος συλλέγουν τα δεδομένα του IMU και τα δημοσιεύουν στο topic imu_raw μηνύματα τύπου sensor_msgs/Imu.




**Εικόνα 16: Δυναμική και στατική επιτάχυνση**

## 4.4 Οπτικό Ραντάρ – LIDAR

Το οπτικό ραντάρ RPLIDAR A1 συνδέεται σειριακά με το Raspberry Pi. Για να μπορούμε να διαβάσουμε τα δεδομένα πρέπει πρώτα να του δώσουμε τα κατάλληλα «δικαιώματα» στη θύρα εισόδου. Αυτό μπορεί να επιτευχθεί με την εντολή:

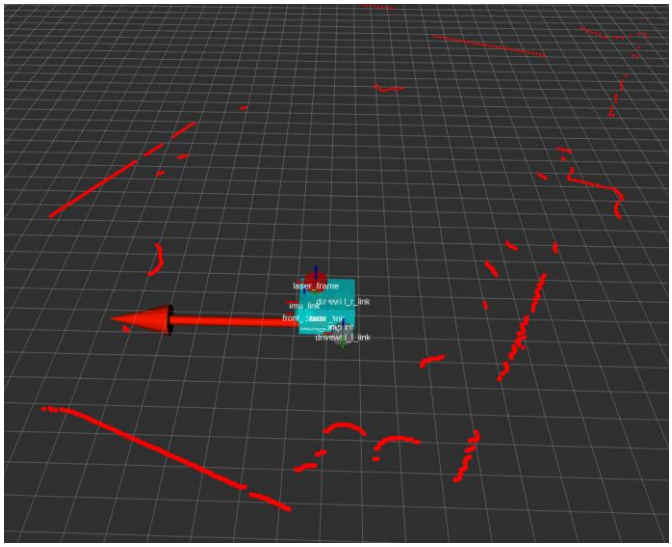
```
sudo chmod a+rw /dev/ttyUSB0
```

Το RPLIDAR έχει τη δυνατότητα να περιστρέφεται 360$^{o}$ μοίρες εκτοξεύοντας ριπές λέιζερ για τον υπολογισμό της απόστασης. Έρχεται με δικό του έτοιμο ros2 node που μετατρέπει τα δεδομένα σε τύπου sensor_msgs/LaserScan για το topic «scan». Για να καλέσουμε το πακέτο του πρέπει να εισάγουμε στο launch αρχείο το παρακάτω κομμάτι κώδικα:

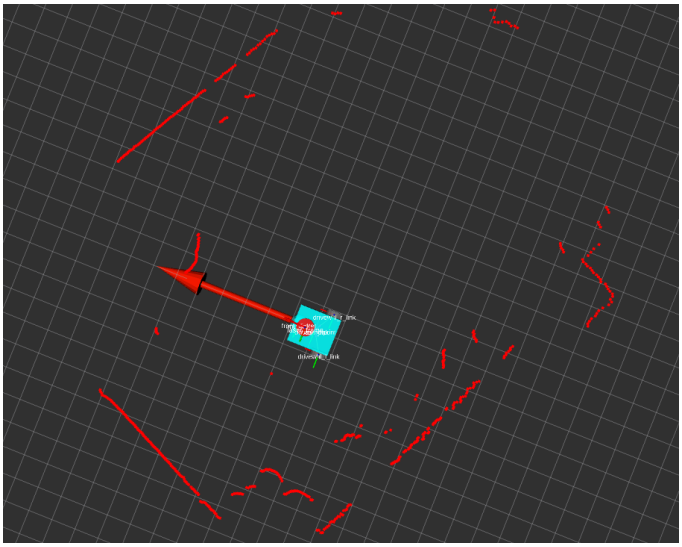
```
rpi_lidar_node = launch_ros.actions.Node(
        package='rplidar_ros2',
        executable='rplidar_scan_publisher',
        name='rplidar_scan_publisher',
        parameters=[{'serial_port': serial_port,
                     'serial_baudrate': serial_baudrate,
                     'frame_id': frame_id,
                     'inverted': inverted}],
        output='screen'
    )
```

**Πίνακας 4: Αρχικοποίηση LIDAR**

Με το serial_baudrate: 115200, serial_port: /dev/ttyUSB0, frame_id: laser_frame και inverted: false. Το frame_id πρέπει να είναι ίδιο με το πλαίσιο που ορίζεται στο urdf μοντέλο [66]. Επίσης είναι σημαντικό ο θετικός άξονας x του LIDAR να ευθυγραμμίζεται με τον αντίστοιχο του οχήματος ώστε να ταιριάζουν οι κατευθύνσεις των δύο εξαρτημάτων. Τα δεδομένα από το οπτικό ραντάρ εμφανίζονται στο RVIZ2:



Εικόνα 17: Απεικόνιση δεδομένων οπτικού ραντάρ 1



Εικόνα 18: Απεικόνιση δεδομένων οπτικού ραντάρ 2

## 4.5 Εντοπισμός Θέσης – Διευρυμένο Φίλτρο Kalman

Το πακέτο robot_localization[26] χρησιμοποιείται για την υλοποίηση του φίλτρου Kalman. Για να κληθεί πρέπει να μπει στο launch αρχείο το παρακάτω κομμάτι κώδικα:

```
robot_localization_node = launch_ros.actions.Node(
        package='robot_localization',
        executable='ekf_node',
        name='ekf_filter_node',
        output='screen',
        parameters=[{'params_file': efk_config,
                     'use_sim_time': False}]
    )
```

**Πίνακας 2: Κώδικας για την αρχικοποίηση του robot_localization**

Όπου «efk_config» η τοποθεσία του αρχείου παραμέτρων. Για να γνωρίζει η βιβλιοθήκη από που να αντλήσει την πληροφορία των αισθητήρων πρέπει να

ορισθεί το αρχείο παραμέτρων τύπου yaml. Στο αρχείο διευκρινίζεται το όνομα των topics και το είδος της πληροφορίας, καθώς και ποια είδη αισθητήρων πρέπει να συνδυαστούν για να υπολογιστεί η τελική οδομετρία. Παρακάτω παρατίθεται το yaml αρχείο που χρησιμοποιήθηκε για να συνδυάσει τους κωδικοποιητές και τον IMU.

```
### ekf config file ###
ekf_filter_node:
ros__parameters:
use_sim_time: false
frequency: 30.0
sensor_timeout: 0.5
two_d_mode: false
publish_acceleration: false
```

**Πίνακας 3: Παραμετροποίηση EFK**

```
publish_tf: true
map_frame: map
odom_frame: odom # Defaults to "odom" if unspecified
base_link_frame: base_footprint # Defaults to "base_link" if unspecified
world_frame: odom
odom0: odom_raw
odom0_config: [
            true, #x
            true, #y
            true, #z
            false, #roll
            false, #pitch
            true, #yaw
            true, #vx
            true, #vy
            true, #vz
            false, #vroll
            false, #vpitch
            true, #vyaw
            false, #ax
            false, #ay
            false, #az
                        ]
odom0_differential: false
odom0_queue_size: 5
odom0_pose_rejection_threshold: 5.0
odom0_twist_rejection_threshold: 1.0

imu0: imu_raw
imu0_config: [
false, #x
false, #y
false, #z
false, #roll
false, #pitch
true, #yaw
```

```
false, #vx

false, #vy

false, #vz

false, #vroll

false, #vpitch

true, #vyaw

true, #ax

true, #ay

true, #az

         ]

imu0_queue_size: 5

imu0_pose_rejection_threshold: 0.8 # Note the difference in parameter names

imu0_twist_rejection_threshold: 0.8 #

imu0_linear_acceleration_rejection_threshold: 0.8 #


process_noise_covariance: [0.05, 0.0,    0.0,    0.0,    0.0,    0.0,    0.0,     0.0,
0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.05, 0.0,    0.0,    0.0,    0.0,    0.0,     0.0,
0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.06, 0.0,    0.0,    0.0,    0.0,     0.0,
0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.03, 0.0,    0.0,    0.0,     0.0,
0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.03, 0.0,    0.0,     0.0,
0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.06, 0.0,     0.0,
0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.025, 0.0,
0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.025, 0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,     0.04, 0.0,    0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,     0.0,    0.01, 0.0,    0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,     0.0,    0.0,    0.01, 0.0,    0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,     0.0,    0.0,    0.0,    0.02, 0.0,    0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,     0.0,    0.0,    0.0,    0.0,    0.01, 0.0,    0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,     0.0,    0.0,    0.0,    0.0,    0.0,    0.01, 0.0,

                                  0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,     0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.015]
```

```
initial_estimate_covariance: [ 1e-9, 0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    1e-9, 0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    1e-9, 0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    1e-9, 0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    1e-9, 0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    1e-9, 0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    1e-9,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
1e-9, 0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    1e-9, 0.0,     0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    1e-9, 0.0,     0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     1e-9, 0.0,     0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     1e-9,  0.0,    0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     1e-9, 0.0,    0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    1e-9, 0.0,

                                    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,    0.0,
0.0,    0.0,    0.0,     0.0,     0.0,     0.0,    0.0,    1e-9
]
```

## 4.6 Χαρτογράφηση

Όπως αναφέρθηκε και προηγουμένως στο [32] για την χαρτογράφηση του χώρου επιλέχθηκε η βιβλιοθήκη slam_toolbox [22][29]. Το slam_toolbox συλλέγει τα δεδομένα του LIDAR αισθητήρα που δημοσιεύονται στο ROS2 topic «/scan» και τον tf2 μετασχηματισμό μεταξύ οδομετρίας και base_link, για την παραγωγή ενός δισδιάστατου χάρτη. Επιπλέον καθώς το σύστημα πλοηγείται, έχει την δυνατότητα να ενημερώνει τυχόν ασαφή σημεία στον χώρο. Η βιβλιοθήκη παράγει τα πλέγματα πληρότητας καθώς και τον μετασχηματισμό του πλαισίου συντεταγμένων χάρτη και οδομετρίας (map => odom), ενώ δημοσιεύει τα δεδομένα του χάρτη στο ROS2 topic με τη δομή «nav_msgs/OccupancyGrid» που είναι απαραίτητα για την λειτουργία του λογισμικού πλοήγησης.

Η βιβλιοθήκη προσφέρει πολλές επιλογές παραμετροποίησης για να προσαρμοστεί στις ανάγκες κάθε έργου. Στον παρακάτω πίνακα αναφέρονται οι σημαντικότερες μεταβλητές που προστέθηκαν στο προεπιλεγμένο αρχείο παραμετροποίησης:

| Παράμετρος | Τιμή | Περιγραφή |
|---|---|---|
| odom_frame | odom | Το πλαίσιο συντεταγμένων που χρησιμοποιείται για την οδομετρία. |
| map_frame | map | Το πλαίσιο συντεταγμένων που θα χρησιμοποιηθεί η βιβλιοθήκη για να |

| | | |
|---|---|---|
| | | δημοσιευθεί ο μετασχηματισμός του χάρτη. |
| base_frame | Base_footprint | Το πλαίσιο συντεταγμένων που χρησιμοποιείται για την βάση. |
| scan_topic | scan | Το ROS2 topic που δημοσιεύει ο αισθητήρας LIDAR τα δεδομένα του. |
| mode | mapping | Θέτουμε την βιβλιοθήκη σε λειτουργία χαρτογράφησης, εφόσον θα χρησιμοποιηθεί μόνο για τη δημιουργία του χάρτη και όχι για τον προσδιορισμό της θέσης (localization). |
| map_file_name | Αφαιρέθηκε | Δεν έχουμε προκατασκευασμένο χάρτη. |
| map_update_interval | 5.0 | Χρονικό διάστημα για την ενημέρωση του δισδιάστατου πλέγματος σε δευτερόλεπτα. |
| resolution | 0.05 | Ανάλυση χάρτη σε μέτρα/πίξελ. |
| transform_publish_period | 0.02 | Η περίοδος αναδημοσίευσης των μετασχηματισμών |

| | | |
|---|---|---|
| | | πλαισίων χάρτη και οδομετρίας (map -> odom) σε δευτερόλεπτα. |
| max_laser_range | 12 | Η μέγιστη εμβέλεια του LIDAR αισθητήρα, σε μέτρα. |
| transform_timeout | 0.5 | Ελάχιστη περίοδος σε δευτερόλεπτα για την αναζήτηση των απαραίτητων tf2 μετασχηματισμών. |
| tf_buffer_duration | 30 | Διάρκεια χρόνου για την αποθήκευση των tf2 μηνυμάτων μετασχηματισμού σε δευτερόλεπτα. |
| minimum_travel_distance | 0.5 | Ελάχιστη απόσταση σε μέτρα που πρέπει να διανύσει το όχημα πριν αρχίσει η βιβλιοθήκη να επεξεργάζεται τα νέα δεδομένα του LIDAR αισθητήρα. |

**Πίνακας 4: Αναλυτική παραμετροποίηση SLAM αλγορίθμου**

Δεν χρειάστηκε κάποια αλλαγή στις υπόλοιπες παραμέτρους του αρχείου, που αφορούν πιο περίπλοκα τεχνικά ζητήματα για την λειτουργία της βιβλιοθήκης. Οι προεπιλεγμένες τους τιμές εξασφαλίζουν την ομαλή λειτουργία του λογισμικού.

Η slam_toolbox βιβλιοθήκη μπορεί να εγκατασταθεί εύκολα στα Ubuntu συστήματα με την εντολή:

```
sudo apt install ros-foxy-slam-toolbox
```

Για να εκτελεστεί χρειάζεται το yaml αρχείο παραμέτρων και να κληθεί από το launch αρχείο του συστήματος όπως θα καλούσαμε και έναν οποιοδήποτε ros node. Αξίζει να αναφερθεί πως υπάρχουν διαφορετικές εκδόσεις της βιβλιοθήκης για κάθε έκδοση του ROS2. Επομένως ενδέχεται να αλλάζουν και οι μεταβλητές που χρησιμοποιούνται όσο θα ανανεώνονται και οι εκδόσεις των πακέτων. Παρακάτω παρατίθεται όλο το αρχείο με τις μεταβλητές που χρησιμοποιήθηκαν για την Foxy διανομή του ROS2:

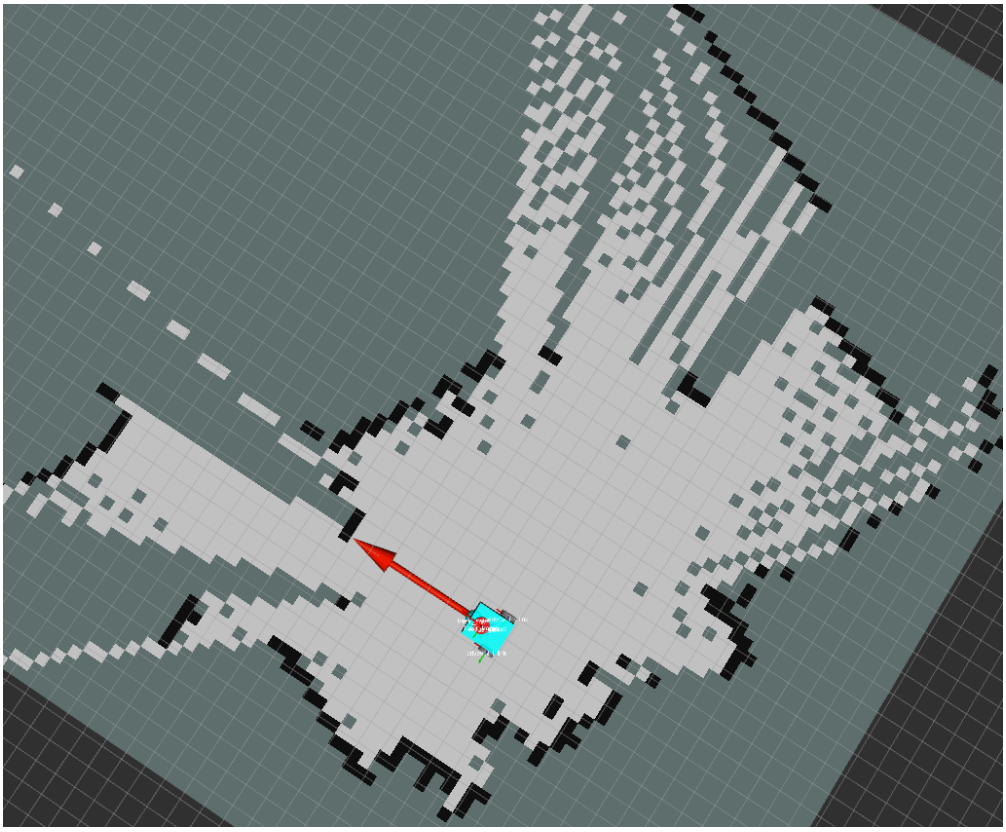
```
slam_toolbox:
  ros__parameters:
    # Plugin params
    solver_plugin: solver_plugins::CeresSolver
    ceres_linear_solver: SPARSE_NORMAL_CHOLESKY
    ceres_preconditioner: SCHUR_JACOBI
    ceres_trust_strategy: LEVENBERG_MARQUARDT
    ceres_dogleg_type: TRADITIONAL_DOGLEG
    ceres_loss_function: None

    # ROS Parameters
    odom_frame: odom
    map_frame: map
    base_frame: base_footprint
    scan_topic: /scan
    mode: mapping #localization
```

```
    debug_logging: false

    throttle_scans: 1

    transform_publish_period: 0.02 #if 0 never publishes odometry

    map_update_interval: 5.0

    resolution: 0.05

    max_laser_range: 12.0 #for rastering images

    minimum_time_interval: 0.5

    transform_timeout: 0.2

    tf_buffer_duration: 30.

    stack_size_to_use: 40000000 #// program needs a larger stack size to serialize
large maps

    enable_interactive_mode: true


    # General Parameters

    use_scan_matching: true

    use_scan_barycenter: true

    minimum_travel_distance: 0.5

    minimum_travel_heading: 0.5

    scan_buffer_size: 10

    scan_buffer_maximum_scan_distance: 10.0

    link_match_minimum_response_fine: 0.1

    link_scan_maximum_distance: 1.5

    loop_search_maximum_distance: 3.0

    do_loop_closing: true

    loop_match_minimum_chain_size: 10

    loop_match_maximum_variance_coarse: 3.0

    loop_match_minimum_response_coarse: 0.35

    loop_match_minimum_response_fine: 0.45
```

```
# Correlation Parameters - Correlation Parameters
correlation_search_space_dimension: 0.5
correlation_search_space_resolution: 0.01
correlation_search_space_smear_deviation: 0.1

# Correlation Parameters - Loop Closure Parameters
loop_search_space_dimension: 8.0
loop_search_space_resolution: 0.05
loop_search_space_smear_deviation: 0.03

# Scan Matcher Parameters
distance_variance_penalty: 0.5
angle_variance_penalty: 1.0

fine_search_angle_offset: 0.00349
coarse_search_angle_offset: 0.349
coarse_angle_resolution: 0.0349
minimum_angle_penalty: 0.9
minimum_distance_penalty: 0.5
use_response_expansion: true
```

**Πίνακας 5: SLAM Παράμετροι**

Μόλις ενεργοποιηθεί το ρομπότ, δίνει μια πρώτη εκτίμηση του τοπικού χάρτη, που μπορεί να φανεί στο RVIZ2:



**Εικόνα 19: Απεικόνιση χάρτη στο RVIZ2**

## 4.7 Πλοήγηση

Για την πλοήγηση του συστήματος επιλέχθηκε η βιβλιοθήκη Navigation2 ή αλλιώς Nav2[28]. Χρησιμοποιεί τον χάρτη, τα transformations, τους αισθητήρες και το goal pose από την βιβλιοθήκη εξερεύνησης συνόρων για να σχεδιάσει το μονοπάτι προς τον στόχο αλλά και να στείλει τις σωστές εντολές κίνησης στο ρομπότ. Διαθέτει πολλές επιλογές παραμετροποίησης από τις οποίες επιλέχθηκαν 3 βασικές υπηρεσίες:

- **Planner Server (Διακομιστές σχεδιασμού):**
  Δέχεται σαν είσοδο το goal pose, υπολογίζει έναν χάρτη κόστους και τρέχει Dijkstra για να βρει την καλύτερη διαδρομή προς τον προορισμό του. Μόλις υπολογισθεί δημοσιεύει το global path στο σύστημα.

- **Controller Server (Διακομιστές ελέγχου):**
  Ελέγχουν συνέχεια αν το όχημα ακολουθεί επιτυχώς την προδιαγεγραμμένη πορεία και αν έφτασε στον στόχο του. Επίσης με τον DWB local planner, υπολογίζει τις ταχύτητες που χρειάζονται για να μετακινηθεί το ρομπότ προς τον στόχο του με τη χρήση ενός τοπικού χάρτη κόστους και λαμβάνοντας υπόψιν και τα εμπόδια που βρίσκονται κοντά. Οι drivers του οχήματος εγγράφονται στο topic της ταχύτητας και την μεταφράζουν σε pwm για κάθε τροχό.

- **Behavior Tree:**
  Αναλαμβάνει τα actions δηλαδή τις δράσεις της βιβλιοθήκης σε διάφορες καταστάσεις όπως για παράδειγμα σε περίπτωση που η πλοήγηση δεν ολοκληρωθεί ή εάν δεν λάβει feed back από το όχημα ή έρθουν timeout errors να επανεκκινήσει τις αντίστοιχες υπηρεσίες.

Το stack μόλις ολοκληρώσει τον στόχο του ενημερώνει την βιβλιοθήκη εξερεύνησης για να επαναλάβει την αναζήτηση συνόρων και να ξανά ξεκινήσει η διαδικασία μέχρι να μην υπάρχουν ανεξερεύνητα σύνορα στον χάρτη.

Όπως και με τη βιβλιοθήκη χαρτογράφησης, ο χρήστης μπορεί να επιλέξει και τροποποιήσει τη βιβλιοθήκη του Nav2 με την χρήση yaml αρχείων. Παρατηρήθηκε μια αστάθεια στη λειτουργία του bt_navigator με πολλές φορές να τερματίζεται το πρόγραμμα χωρίς κάποια σαφή αιτία. Επιπλέον κάποιες από τις προεπιλεγμένες βιβλιοθήκες που του δίνονται σαν παράμετροι δεν υποστηρίζονται πια στην Foxy έκδοση του ROS2 επομένως η κατασκευή του yaml αρχείου πρέπει να γίνεται με ιδιαίτερη προσοχή.
Παρακάτω παρατίθεται το αρχείο με όλες τις παραμέτρους που δοκιμάστηκαν (με επιτυχία) και για να τρέξουν τον Nav2. Ιδιαίτερη προσοχή σε δύο σημεία, πρώτον εφόσον εστιάζουμε σε εξερεύνηση σε πραγματικό χρόνο όλα τα nodes του navigation 2 πρέπει να έχουν ορισμένο την use_sim_time σε false τιμή. Δεύτερον εφόσον χρησιμοποιούμε τον ekf node για τον εντοπισμό θέσης πρέπει να αφαιρεθεί η προεπιλογή του amcl.

```
controller_server:
  ros__parameters:
    use_sim_time: False
    controller_frequency: 20.0
    min_x_velocity_threshold: 0.001
    min_y_velocity_threshold: 0.5
    min_theta_velocity_threshold: 0.001
    progress_checker_plugin: "progress_checker"
    goal_checker_plugin: "goal_checker"
    controller_plugins: ["FollowPath"]

      # Add high threshold velocity for turtlebot 3 issue.
      # https://github.com/ROBOTIS-GIT/turtlebot3_simulations/issues/75
      acc_lim_x: 2.5
      acc_lim_y: 0.0
      acc_lim_theta: 3.2
      decel_lim_x: -2.5
      decel_lim_y: 0.0
      decel_lim_theta: -3.2
      vx_samples: 20
      vy_samples: 5
      vtheta_samples: 20
```

```
      sim_time: 1.7
      linear_granularity: 0.05
      angular_granularity: 0.025
      transform_tolerance: 1.0
      xy_goal_tolerance: 0.25
      trans_stopped_velocity: 0.25
      short_circuit_trajectory_evaluation: True
      stateful: True
      critics:
        [
          "RotateToGoal",
          "Oscillation",
          "BaseObstacle",
          "GoalAlign",
          "PathAlign",
          "PathDist",
          "GoalDist",
        ]
      BaseObstacle.scale: 0.02
      PathAlign.scale: 32.0
      PathAlign.forward_point_distance: 0.1
      GoalAlign.scale: 24.0
      GoalAlign.forward_point_distance: 0.1
      PathDist.scale: 32.0
      GoalDist.scale: 24.0
      RotateToGoal.scale: 32.0
      RotateToGoal.slowing_factor: 5.0
      RotateToGoal.lookahead_time: -1.0
```

```
xy_goal_tolerance: 0.25
      trans_stopped_velocity: 0.25
      short_circuit_trajectory_evaluation: True
      stateful: True
      critics:
        [
          "RotateToGoal",
          "Oscillation",
          "BaseObstacle",
          "GoalAlign",
          "PathAlign",
          "PathDist",
          "GoalDist",
        ]
      BaseObstacle.scale: 0.02
      PathAlign.scale: 32.0
      PathAlign.forward_point_distance: 0.1
      GoalAlign.scale: 24.0
      GoalAlign.forward_point_distance: 0.1
      PathDist.scale: 32.0
      GoalDist.scale: 24.0
      RotateToGoal.scale: 32.0
      RotateToGoal.slowing_factor: 5.0
      RotateToGoal.lookahead_time: -1.0

controller_server_rclcpp_node:
  ros__parameters:
    use_sim_time: False

planner_server:
  ros__parameters:
    expected_planner_frequency: 5.0 #20.0
    use_sim_time: False
```

```
    planner_plugins: ["GridBased"]
    GridBased:
      plugin: "nav2_navfn_planner/NavfnPlanner"
      tolerance: 0.5
      use_astar: false
      allow_unknown: true

planner_server_rclcpp_node:
  ros__parameters:
    use_sim_time: False

recoveries_server:
  ros__parameters:
    recovery_plugins: ["spin", "backup", "wait"]
    spin:
      plugin: "nav2_recoveries/Spin"
    backup:
      plugin: "nav2_recoveries/BackUp"
    wait:
      plugin: "nav2_recoveries/Wait"

bt_navigator:
  ros__parameters:
    use_sim_time: False
    global_frame: map
    robot_base_frame: base_footprint
    odom_topic: /odom_filtered
    enable_groot_monitoring: False
    groot_zmq_publisher_port: 1666
    groot_zmq_server_port: 1667
    default_bt_xml_filename: "navigate_w_replanning_and_recovery.xml"
    plugin_lib_names:
      - nav2_compute_path_to_pose_action_bt_node
      - nav2_follow_path_action_bt_node
      - nav2_back_up_action_bt_node
      - nav2_spin_action_bt_node
      - nav2_wait_action_bt_node
      - nav2_clear_costmap_service_bt_node
      - nav2_is_stuck_condition_bt_node
```

```
- nav2_is_stuck_condition_bt_node
      - nav2_goal_reached_condition_bt_node
      - nav2_goal_updated_condition_bt_node
      - nav2_initial_pose_received_condition_bt_node
      - nav2_reinitialize_global_localization_service_bt_node
      - nav2_rate_controller_bt_node
      - nav2_distance_controller_bt_node
      - nav2_speed_controller_bt_node
      - nav2_truncate_path_action_bt_node
      - nav2_goal_updater_node_bt_node
      - nav2_recovery_node_bt_node
      - nav2_pipeline_sequence_bt_node
      - nav2_round_robin_node_bt_node
      - nav2_transform_available_condition_bt_node
      - nav2_time_expired_condition_bt_node
      - nav2_distance_traveled_condition_bt_node
bt_navigator_rclcpp_node:
  ros__parameters:
    use_sim_time: False
global_costmap:
  global_costmap:
    ros__parameters:
      update_frequency: 1.0
      publish_frequency: 1.0
      global_frame: map
   robot_base_frame: base_footprint
      use_sim_time: False
      robot_radius: 0.18
      resolution: 0.01
      track_unknown_space: true
      plugins: ["static_layer", "obstacle_layer", "inflation_layer"]
      obstacle_layer:
plugin: "nav2_costmap_2d::ObstacleLayer"
        enabled: True
        observation_sources: scan
```

```
scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
      static_layer:
        plugin: "nav2_costmap_2d::StaticLayer"
        map_subscribe_transient_local: True
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.95
      always_send_full_costmap: True
  global_costmap_client:
    ros__parameters:
      use_sim_time: False
  global_costmap_rclcpp_node:
    ros__parameters:
      use_sim_time: False

local_costmap:
  local_costmap:
    ros__parameters:
      update_frequency: 5.0
      publish_frequency: 2.0
      global_frame: odom
      robot_base_frame: base_footprint
      use_sim_time: False
      rolling_window: true
      width: 10 #3
      height: 10 #3
      resolution: 0.01
      robot_radius: 0.2
      #footprint: "[ [0.6, 0.62], [0.6, -0.62], [-0.6, -0.62], [-0.6, 0.62] ]"
      plugins: ["voxel_layer", "inflation_layer"]
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.75 #0.55
      voxel_layer:
```

```
plugin: "nav2_costmap_2d::VoxelLayer"
        enabled: True
        publish_voxel_map: True
        origin_z: 0.0
        z_resolution: 0.05
        z_voxels: 16
        max_obstacle_height: 2.0
        mark_threshold: 0
        observation_sources: scan
        scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
      static_layer:
        map_subscribe_transient_local: True
      always_send_full_costmap: True
  local_costmap_client:
    ros__parameters:
      use_sim_time: False

  local_costmap_rclcpp_node:
    ros__parameters:
      use_sim_time: False
```

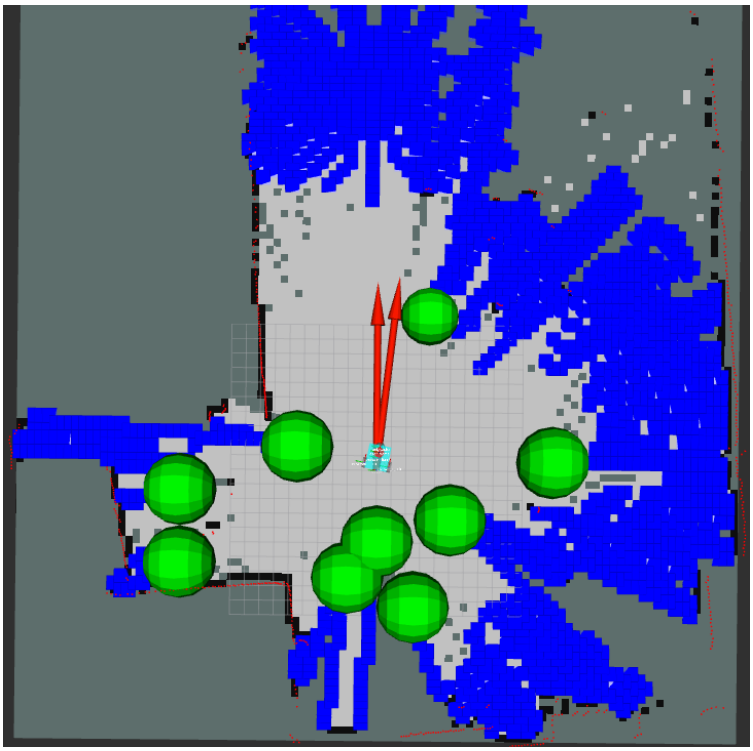
**Πίνακας 6: Nav2 Παράμετροι**

## 4.8 Εξερεύνηση

Το πακέτο m2-explore[25] χρησιμοποιήθηκε για την εξερεύνηση των διαθέσιμων συνόρων όπως έχει αναφερθεί και στο κεφάλαιο 3.1.5 [34]. Η χρήση του είναι αρκετά απλή αφού μπορεί να καλεστεί ανεξάρτητα με την εντολή:

```
ros2 run explore_lite explore --ros-args --params-file <path_to_ros_ws>/m-explore/explore/config/params.yaml
```

Όπως φαίνεται και στην Εικόνα 18 το όχημα μόλις έχει ενεργοποιηθεί, έχει δημιουργήσει έναν τοπικό χάρτη για το περιβάλλον του και το πακέτο explore lite αφού κάνει subscribe στον πλέγμα χάρτη του slam toolbox εκτελεί bfs για να βρει τα ποθητά σύνορα. Στην εικόνα βλέπουμε τον αλγόριθμο στην πράξη να έχει υπολογίσει ήδη τους διαθέσιμους frontiers (πράσινη μπάλα) και να κατευθύνεται προς τον κοντινότερο διαθέσιμο.



**Εικόνα 20: Εξερεύνηση Συνόρων**

# ΚΕΦ.5: Εφαρμογή

## 5.1 Σενάρια Χρήσης

Ο κύριος στόχος του αυτοκινούμενου οχήματος είναι να τοποθετείται σε έναν καινούριο χώρο για τον οποίο το σύστημα δεν διαθέτει καμία πληροφορία και να μπορεί να τον εξερευνήσει, χαρτογραφήσει και να πλοηγηθεί χωρίς καμία υπόδειξη από τον χρήστη. Αφού ενεργοποιηθούν τα πακέτα λογισμικού, το όχημα χρησιμοποιεί τους αισθητήρες οπτικού ραντάρ και το γυροσκόπιο / επιταχυνσιόμετρο και τους κωδικοποιητές για να εντοπίσει τη θέση του στον χώρο και να κατασκευάσει έναν πρώιμο τοπικό χάρτη. Στην συνέχεια το ρομποτικό σύστημα μέσω του πλέγματος πληρότητας χάρτη θα εντοπίσει τα διαθέσιμα κοντινά του σύνορα και θα διαλέξει το πιο προσβάσιμο σύνορο να επισκεφθεί. Όσο πλοηγείται προς τον προορισμό του, θα ανανεώνει, διορθώνει και επεκτείνει τον πρώιμο χάρτη ανανεώνοντας και το πλέγμα πληρότητας. Όταν επισκεφθεί τον προορισμό του, θα επαναλάβει ολόκληρη τη διαδικασία μέχρι να μην υπάρχουν άλλα ανεξερεύνητα σύνορα στον χάρτη. Μόλις τελειώσει η διαδικασία θα εμφανιστεί στο τερματικό ένα μήνυμα «Exploration Stopped» και ο χάρτης θα είναι έτοιμος. Ο χρήστης το μόνο που έχει να αποθηκεύσει τον χάρτη μέσω του RVIZ2 ή της εντολής:

```
ros2 run nav2_map_server map_saver_cli -f my_map --ros-
args -p save_map_timeout:=10000
```

όπου «my_map»: το όνομα του χάρτη που θέλουμε.

Ο χρήστης επίσης μπορεί να συνεχίσει την πλοήγηση του οχήματος αν θέλει. Χρησιμοποιώντας το πρόσθετο της βιβλιοθήκης πλοήγησης Nav2 στο RVIZ2, μπορεί να επιλέξει πάνω στον χάρτη που έχει δημιουργηθεί το σημείο που θέλει να φτάσει το όχημα, και εκείνο χρησιμοποιώντας το πακέτο Nav2 να ολοκληρώσει την

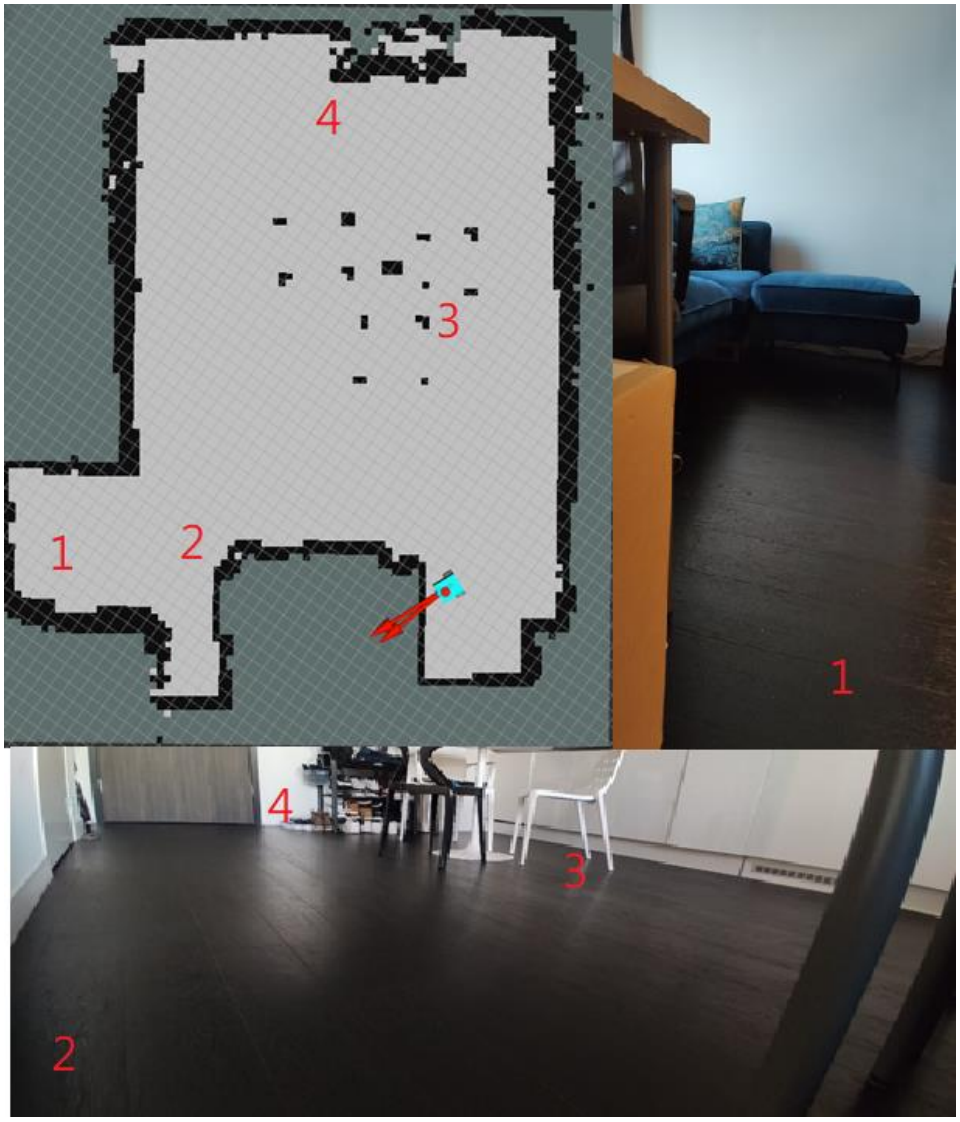
πλοήγηση του. Σε αυτό το σενάριο το exploration node δεν χρησιμοποιείται πια μιας και δεν υπάρχουν άλλα ανεξερεύνητα σημεία. Ωστόσο όσο ο αλγόριθμος SLAM συνεχίζει να λειτουργεί, συνεχίζει να διορθώνει (εάν χρειάζεται) τον χάρτη του.

## 5.2 Αποτελέσματα

Παρατίθενται δύο χάρτες διαμερισμάτων που δημιουργήθηκαν μετά την επιτυχή εκτέλεση αυτόνομης εξερεύνησης και χαρτογράφησης από το ρομποτικό όχημα. Μπορούμε να παρατηρήσουμε στην πρώτη εικόνα αρκετό “θόρυβο”. Λόγω του οπτικού ραντάρ οι ριπές λέιζερ μπορούν να ανιχνεύσουν οποιοδήποτε κενό στον χώρο. Στην εικόνα 19 για παράδειγμα με μια πρώτη ματιά νομίζουμε ότι ο χάρτης έχει παραμορφωμένα τα όρια του και ότι εμπεριέχει ανακρίβειες αλλά είναι επειδή το LIDAR έχει καταφέρει να καταγράψει ακόμα και κρυφές εσοχές εξαερισμού κάτω από τα έπιπλα όπως στο σημείο 1, ή τις αναδιπλώσεις ανάμεσα στις κουρτίνες (σημείο 2) ή και τα ράφια της βιβλιοθήκης (σημείο 6).


**Εικόνα 21: Χάρτης πρώτου διαμερίσματος**

Το ίδιο φαινόμενο παρατηρείται και στο δεύτερο διαμέρισμα όπου το οπτικό ραντάρ και πάλι καταφέρνει να καταγράψει μικρά κενά όπως αυτά ανάμεσα στην παπουτσοθήκη στο σημείο 4 της Εικόνας 20 και φαίνονται στον χάρτη σαν παραμορφώσεις ή στο σημείο 3 όπου έχει καταφέρει να αποτυπώσει σωστά τη θέση των ποδιών από τις καρέκλες.




**Εικόνα 22: Χάρτης 2ου διαμερίσματος**

Οι δοκιμές μπορούν να χαρακτηριστούν ως επιτυχείς εφόσον η ρομποτική πλατφόρμα κατάφερε να φέρει εις πέρας τον στόχο για τον οποίο κατασκευάστηκε.

## 5.3 Συμπεράσματα

Κατά την υλοποίηση της παρούσας πτυχιακής αντιμετωπίστηκαν κάποιες δυσκολίες που αφορούν κυρίως τις λειτουργίες του λογισμικού. Αρχικά, ένας από τους μεγαλύτερους περιορισμούς ήταν ο αριθμός των πακέτων και αλγορίθμων που υποστηρίζονται από το ρομποτικό λειτουργικό σύστημα ROS2 όπως αναφέρονται και στο κεφάλαιο [31]. Είναι λογικό, όταν η προηγούμενη έκδοση του ROS, αναπτύχθηκε στις κοινότητες ανοιχτού κώδικα από απεριόριστο αριθμό προγραμματιστών για περίπου μια δεκαετία να παρέχει τεράστια υποστήριξη και επιπλέον λειτουργικότητα σε σχέση με το πρόσφατο ROS2. Παρόλο τον περιορισμένο αριθμό πακέτων όμως ο στόχος της εργασίας επιτεύχθηκε αλλά με κάποιες δυσκολίες. Συγκεκριμένα το πακέτο πλοήγησης Nav2 ήταν αυτό που παρουσίασε τα περισσότερα προβλήματα. Τα μηνύματα λάθους:

```
[bt_navigator]: Action server failed while executing
action callback: "send_goal failed"
[bt_navigator]: [navigate_to_pose] [ActionServer]
Aborting handle.
```

εμφανιζόντουσαν αρκετά συχνά τερματίζοντας την πλοήγηση χωρίς κάποια εξήγηση. Προέρχονται από τον κόμβο behavior trees του Nav2 και φαίνονται να συσχετίζονται με την Foxy έκδοση του ROS2, καθώς από ότι φαίνεται έχει διορθωθεί στην Galactic. Επίσης, υπήρξε δυσκολία στην αποθήκευση του χάρτη καθώς ο υπεύθυνος διακομιστής για την συγκεκριμένη λειτουργία συνέχιζε να εκτυπώνει μηνύματα λάθους εξάντλησης χρόνου (time out) και δεν ανταποκρινόταν. Λύνεται με την εντολή αποθήκευσης που αναφέρεται στο κεφάλαιο 5.1. [87], χωρίς βέβαια αυτό να αποτελεί μόνιμη λύση.
Τέλος για την tf2 βιβλιοθήκη για να γίνει σωστά η ερμηνεία των ονομάτων, λόγω κάποιου προβλήματος του πακέτου, είναι απαραίτητη η χρήση της εξής παραμέτρου στο launch αρχείο.

```
remappings = [('/tf', 'tf'), ('/tf_static', 'tf_static')]
```

Η περιορισμένη λειτουργικότητα αναμένεται τα επόμενα χρόνια να βελτιωθεί καθώς καθημερινά όλο και περισσότεροι χρήστες συνεισφέρουν στο αποθετήριο των ROS2 πακέτων.

## 5.4 Μελλοντικές Επεκτάσεις

Για μεγαλύτερη ακρίβεια στον έλεγχο του ρομποτικού οχήματος θα ήταν χρήσιμο να υλοποιηθεί ένα σύστημα διαχείρισης επιπέδων μπαταρίας (Battery Management System) καθώς η τάση της μπαταρίας είναι ανάλογη με την ταχύτητα που μπορούν να φτάσουν οι τροχοί του οχήματος. Επομένως θα ήταν ωφέλιμο να υπάρχει ένα σύστημα που θα ανιχνεύει το υπόλοιπο φορτίο της μπαταρίας και θα ρυθμίζει τις μέγιστες τιμές της ταχύτητας και του κύκλου λειτουργίας (duty cycle). Επίσης θα ήταν ενδιαφέρον αν μπορούσαν να προστεθούν σόναρ αισθητήρες στο κατώτερο μέρος του οχήματος για να ανιχνεύουν διαφορές στο επίπεδο π.χ. αν υπάρχουν σκάλες. Άλλη μια λειτουργικότητα που θα ήταν χρήσιμη είναι η δημιουργία ενός αναλογικού-ολοκληρωτικού-παραγωγικού ελεγκτή (PID – Proportional Integral Derivative) για να μπορεί το όχημα να διανύει μεγάλες αποστάσεις διατηρώντας όσο περισσότερο γίνεται ευθεία πορεία. Τέλος, θα είχε ενδιαφέρον να επανεξεταστούν οι διαθέσιμες επιλογές σε μια νέα μελλοντική έκδοση LTS ROS2 και να συγκριθούν οι αποδώσεις ακόμα και χρησιμοποιώντας ανώτερες επεξεργαστικές μονάδες ή hardware accelerators.

# ΒΙΒΛΙΟΓΡΑΦΙΑ


[1]. Macenski S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics, 7(66). https://doi.org/10.1126/scirobotics.abm6074

[2]. H. Durrant-Whyte and T. Bailey,(June 2006), "Simultaneous localization and mapping: part I," in IEEE Robotics & Automation Magazine, vol. 13, no. 2, pp. 99-110, σελ.2-3, doi:10.1109/MRA.2006.1638022

[3]. Folmer, H., & Appel, R. (2016, December 12). Analysis, optimization, and design of a SLAM solution for an implementation on reconfigurable hardware (FPGA) using CλaSH, σελ 9-8, https://essay.utwente.nl/71550/3/Appel_MA_EEMCS.pdf.

[4]. Mahmoud, Doaa & Salem, Mohammed Abdel-Megeed Mohammed & Ramadan, Hassan & Roushdy, Mohamed. (2014). 3D Graph-Based Vision-SLAM Registration and Optimization. International Journal of Circuits, Systems and Signal Processing. 8. 123.

[5]. Macario Barros, A., Michel, M., Moline, Y., Corre, G., & Carrel, F. (2022). A Comprehensive Survey of Visual SLAM Algorithms. Robotics, 11(1), 24. https://doi.org/10.3390/robotics11010024

[6]. Yang J, Li Y, Cao L, Jiang Y, Sun L, Xie Q. A Survey of SLAM Research based on LiDAR Sensors. Int J Sens. 2019; 1(1): 1003.

[7]. Valentin, C.B. (2020). Evaluation and comparison of 3D lidar based SLAM algorithms.

[8]. Khan, Misha Urooj & Zaidi, Syed Azhar Ali & Ishtiaq, Arslan & Bukhari, Syeda & Samer, Sana & Farman, Ayesha. (2021). A Comparative Survey of LiDAR-SLAM and LiDAR based Sensor Technologies. 10.1109/MAJICC53071.2021.9526266.

[9]. Debeunne, C., & Vivet, D. (2020). A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping. Sensors, 20(7), 2068. https://doi.org/10.3390/s20072068

[10]. Kuzmin, (2018), "Review. Classification and Comparison of the Existing SLAM Methods for Groups of Robots," 2018 22nd Conference of Open Innovations Association (FRUCT), pp. 115-120, σελ. 116
doi: 10.23919/FRUCT.2018.8468281.

[11]. Y. Wenhui and Y. Fan, (2017), "Lidar Image Classification Based on Convolutional Neural Networks," 2017 International Conference on Computer Network, Electronic and Automation (ICCNEA), pp. 221-225, doi: 10.1109/ICCNEA.2017.37.

[12]. Y. Li et al., (August 2021), "Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review," in IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 8, pp. 3412-3432, doi: 10.1109/TNNLS.2020.3015992.

[13]. V. Vaquero, A. Sanfeliu and F. Moreno-Noguer, (2018), "Deep Lidar CNN to Understand the Dynamics of Moving Vehicles," 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 4504-4509, doi: 10.1109/ICRA.2018.8460554.

[14]. Capellier, E., Davoine, F., Berge-Cherfaoui, V., & Li, Y. (2021). Fusion of neural networks, for LIDAR-based evidential road mapping. *Journal of Field Robotics, 38*, 727 - 758.

[15]. G. Sepulveda, J. C. Niebles and A. Soto, (2018), "A Deep Learning Based Behavioral Approach to Indoor Autonomous Navigation," 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 4646-4653, doi: 10.1109/ICRA.2018.8460646.

[16]. Kim, D.K., & Chen, T. (2015). Deep Neural Network for Real-Time Autonomous Indoor Navigation. *ArXiv, abs/1511.04668*.

[17]. Rodrigo Eduardo Arevalo Ancona, Leonel Germán Corona Ramírez and Oscar Octavio Gutiérrez Frías, (2021), “Indoor Localization and Navigation based on Deep Learning using a Monocular Visual System” International Journal of Advanced Computer Science and Applications (IJACSA), 12(6), http://dx.doi.org/10.14569/IJACSA.2021.0120611

[18]. Hamieh, I., Myers, R., Nimri, H., Rahman, T. et al., (2020), "LiDAR and Camera-Based Convolutional Neural Network Detection for Autonomous Driving," SAE Technical Paper 2020-01-0136, https://doi.org/10.4271/2020-01-0136.

[19]. Alexandersson och Olle Nordin, (2017), Implementation of SLAM algorithms in a small-scale vehicle using model-based development Johan, σελ 9, (Master of Science Thesis) in Datorteknik, Fordonssystem Department of Electrical Engineering, Linköping University.

[20]. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabillistic Robotics. Σελ. 20 έως 29. Cambridge, Mass.: MIT Press, 1 edition, 2005

[21]. B. Yamauchi, "A frontier-based approach for autonomous exploration," Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation', 1997, pp. 146-151, doi: 10.1109/CIRA.1997.613851.

[22]. Macenski et al., (2021). SLAM Toolbox: SLAM for the dynamic world. Journal of Open-Source Software, 6(61), 2783, https://doi.org/10.21105/joss.02783

[23]. Jiří Hörner, ( 2016). Map-merging for multi-robot system, Bachelor's thesis, Charles University in Prague, Faculty of Mathematics and Physics, Prague. https://is.cuni.cz/webapps/zzp/detail/174125/, Github: https://github.com/hrnr/m-explore

[24]. Paul Bovbe (2014), Frontier Search, Github https://github.com/paulbovbel/frontier_exploration

[25]. Carlos Andrés Álvarez Restrepo, (2021), m-explore-ros2, Github: https://github.com/robo-friends/m-explore-ros2

[26]. T. Moore and D. Stouc (July 2014), A Generalized Extended Kalman Filter Implementation for the Robot Operating System, Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13), Springer

[27]. CS W4733 NOTES - Differential Drive Robots, Columbia, CS Department http://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf

[28]. S. Macenski, F. Martín, R. White, J. Clavero. The Marathon 2: A Navigation System. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[29]. Macenski, S., "On Use of SLAM Toolbox, A fresh(er) look at mapping and localization for the dynamic world", ROSCon 2019.