DEVELOPMENT AND EVALUATION OF AN OPEN SOURCE EDUCATIONAL ROBOTIC VEHICLE IN A ROS2 ENVIRONMENT

by

ANDREADAKIS GEORGIOS

B.A. Applied Informatics and Multimedia Engineering at
Technological Educational Institute of Crete, 2013

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

MARCH 2026

Approved by:

Associate Professor

PANAGIOTAKIS SPYRIDON

Abstract

The teaching of robotics is currently applied at all levels of education and the main tools used are robotics platforms which are divided into software and moving or non-moving robotic systems. These platforms can support teaching subjects ranging from the introduction to robotics in primary education to the software development for autonomous robotic vehicles in tertiary education and depending on their use they have different characteristics and costs.

The Free and Open Source Software Bot (FOSSBot) is an economical educational robotic vehicle, which has been designed by the Open Technologies Organization (GFOSS) and is suitable for students in primary and secondary education. However, it lags behind in the teaching of robotics in tertiary education as its equipment supports simple movement algorithms and not complex ones such as real-time mapping and navigation (SLAM). Robotic vehicles that support such algorithms, incorporate advanced sensors such as LiDAR and cameras and have a high cost and level of complexity, which prevents their use in education. For this reason, this thesis attempts to upgrade the hardware and software of FOSSBot to make it an economical educational tool suitable for all levels of the education.

The aim of this thesis is: a) The successful integration of the original and new sensors into the FOSSBot framework, ensuring their mounting and power supply, b) the development of software for integration of the new sensors into it and for the upgrading of its simulation model in the Gazebo environment, c) the implementation and testing, in a ROS2 environment, of SLAM algorithms using data from LiDAR, IMU and odometry and autonomous navigation algorithms based on the maps created by the first algorithms, d) the creation of instructions and usage scenarios for the use of FOSSBot by students in both real scenarios and simulations and e) the evaluation of the system's performance in terms of mapping accuracy and location detection.

Keywords: Educational robotic vehicles, autonomous navigation, FOSSBot, ROS2, Gazebo, SLAM, NAV2, LiDAR, IMU, odometry.

Περίληψη

Η διδασκαλία της ρομποτικής εφαρμόζεται σήμερα σε όλες τις βαθμίδες της εκπαίδευσης και τα βασικά εργαλεία που χρησιμοποιούνται σε αυτή είναι οι πλατφόρμες ρομποτικής που απαρτίζονται από το λογισμικό και τα κινούμενα ή μη ρομποτικά συστήματα. Οι πλατφόρμες αυτές μπορούν να υποστηρίξουν αντικείμενα διδασκαλίας από την εισαγωγή στη ρομποτική στην πρωτοβάθμια εκπαίδευση έως την ανάπτυξη λογισμικού για αυτόνομα ρομποτικά οχήματα στην τριτοβάθμια. Ανάλογα με τη βαθμίδα στην οποία χρησιμοποιούνται, έχουν διαφορετικά χαρακτηριστικά και κόστος.

Το Free and Open Source Software Bot (FOSSBot) είναι ένα οικονομικό εκπαιδευτικό ρομποτικό όχημα, το οποίο έχει σχεδιαστεί από τον Οργανισμό Ανοιχτών Τεχνολογιών (ΕΕΛ/ΛΑΚ) και είναι κατάλληλο για μαθητές της πρωτοβάθμιας και δευτεροβάθμιας εκπαίδευσης. Υστερεί όμως στη διδασκαλία της ρομποτικής στην τριτοβάθμια εκπαίδευση καθώς ο εξοπλισμός του υποστηρίζει απλούς αλγόριθμους κίνησης και όχι σύνθετους όπως η χαρτογράφηση και πλοήγηση σε πραγματικό χρόνο (SLAM). Τα ρομποτικά οχήματα που υποστηρίζουν τέτοιους αλγόριθμους, ενσωματώνουν προηγμένους αισθητήρες όπως LiDAR και κάμερα, και έχουν υψηλό κόστος και επίπεδο πολυπλοκότητας γεγονός που συχνά αποτρέπει τη μαζική χρησιμοποίηση τους στην εκπαίδευση. Για το λόγο αυτό, στην παρούσα εργασία γίνεται προσπάθεια να αναβαθμιστεί το υλικό και το λογισμικό του FOSSBot, ώστε να καταστεί ένα οικονομικό εκπαιδευτικό εργαλείο κατάλληλο για όλες τις βαθμίδες εκπαίδευσης.

Στόχος αυτής της διπλωματικής εργασίας είναι: α) Η επιτυχής ενσωμάτωση των αρχικών και των νέων αισθητήρων στο πλαίσιο του FOSSBot διασφαλίζοντας τη στερέωση και την τροφοδοσία τους, β) η ανάπτυξη λογισμικού για την ενσωμάτωση των νέων αισθητήρων στο FOSSBot και για την αναβάθμιση του μοντέλου προσομοίωσης του σε περιβάλλον Gazebo, γ) η υλοποίηση και η δοκιμή, σε περιβάλλον ROS2, αλγορίθμων SLAM χρησιμοποιώντας τα δεδομένα από τον LiDAR, την IMU και την οδομετρία, καθώς και αλγορίθμων αυτόνομης πλοήγησης βασιζόμενων στους χάρτες που δημιουργούνται από τους πρώτους αλγόριθμους, δ) η δημιουργία οδηγιών και σεναρίων χρήσης για τη χρησιμοποίηση του FOSSBot από φοιτητές τόσο σε πραγματικά σενάρια όσο και σε προσομοιώσεις και ε) η αξιολόγηση της απόδοσης του συστήματος όσον αφορά την ακρίβεια της χαρτογράφησης, του εντοπισμού θέσης και της πλοήγησης που επιτυγχάνει.

Λέξεις-κλειδιά: Εκπαιδευτικά ρομποτικά οχήματα, αυτόνομη πλοήγηση, FOSSBot, ROS2, Gazebo, SLAM, NAV2, LiDAR, IMU, οδομετρία.

Πίνακας περιεχομένων

Πίνακας περιεχομένων v
Λίστα Εικόνων ix
Λίστα Πινάκων xiii
Κεφάλαιο 1 - Εισαγωγή 1
1.1. Η Ρομποτική στην Εκπαίδευση 1
1.2. Εκπαιδευτικές Πλατφόρμες Ρομποτικής 2
1.3. Ορισμός του προβλήματος 3
1.4. Στόχος διατριβής 4
Κεφάλαιο 2 - Σχετικές Τεχνολογίες 5
2.1. Ρομποτικά Οχήματα 5
2.1.1 Εισαγωγή 5
2.1.2 Κατηγορίες 5
2.1.2.1 Ρομποτικά οχήματα για μαθητές ηλικίας μέχρι 12 χρονών 5
2.1.2.2 Ρομποτικά οχήματα για Πανεπιστήμια και έρευνα 6
2.1.2.3 Ρομποτικά οχήματα με χαμηλό κόστος και μεγάλη επεκτασιμότητα 9
2.1.2.4 Βασικά χαρακτηριστικά 10
2.1.3 Αξιολόγηση Ρομποτικών Οχημάτων 12
2.1.4 Προκλήσεις 14
2.1.4.1 Αντιμετώπιση προκλήσεων 15
2.2. FOSSBot 16
2.2.1 Υλικό μέρος του FOSSBot 16
2.2.1.1 Μικροϋπολογιστής Raspberry Pi Zero 2 W 17
2.2.1.2 Ενεργοποιητές 17
2.2.1.3 Αισθητήρες 18
2.2.1.4 Τροφοδοσία 21
2.2.2 Λογισμικό 22
2.2.2.1 Λογισμικό Ρομποτικού Οχήματος 22
2.2.2.2 Λογισμικό Προσομοίωσης 23
2.2.3 Ελλείψεις FOSSBot 24

2.2.3.1 Υλικού ... 24
2.2.3.2 Προσομοίωσης ... 24
2.3. Πρόσθετοι Αισθητήρες ... 25
2.3.1 Ανίχνευση και Μέτρηση Απόστασης μέσω Φωτός – LiDAR ... 25
2.3.2 Κάμερα ... 26
2.4. Λειτουργικό Σύστημα Ρομπότ 2η έκδοση - ROS2 ... 27
2.4.1 Αρχιτεκτονική του ROS2 ... 27
2.4.1.1 Υπηρεσία Διανομής Δεδομένων - DDS ... 27
2.4.1.2 Επίπεδα Αφαίρεσης ... 28
2.4.1.3 Κόμβοι ... 28
2.4.1.4 Υποστήριξη Διαφορετικών Περιβαλλόντων ... 29
2.4.2 Εργαλεία και πακέτα του ROS2 ... 29
2.4.2.1 Προσομοίωση - Gazebo ... 29
2.4.2.2 Μορφή Περιγραφής Προσομοίωσης – SDF / URDF ... 30
2.4.2.3 Τρισδιάστατα μοντέλα – DAE / STL ... 31
2.4.2.4 Βιβλιοθήκη μετασχηματισμών - TF2 ... 31
2.4.2.5 Οπτικοποίηση - RViz2 ... 32
2.4.3 Οδομετρία ... 32
2.4.4 Σύντηξη Αισθητήρων ... 34
2.4.5 Ταυτόχρονος Εντοπισμός και Χαρτογράφηση– SLAM ... 34
2.4.5.1 Η Διαδικασία του SLAM ... 35
2.4.5.2 Αλγόριθμοι SLAM ... 35
2.4.5.3 Προκλήσεις ... 38
2.4.6 Αυτόνομη Πλοήγηση – Nav2 ... 39
Κεφάλαιο 3 - Ανάπτυξη Πλατφόρμας Προσομοίωσης ... 41
3.1. Επιλογή Έκδοσης Λογισμικού ... 41
3.2. Εγκατάσταση Λογισμικού ... 42
3.2.1 Εγκατάσταση Ubuntu 22.04 LTS σε WSL ... 42
3.2.2 Εγκατάσταση ROS2 Humble Hawksbill Desktop ... 44
3.2.3 Εγκατάσταση Πακέτων του ROS2 ... 45
3.2.3.1 Εγκατάσταση Gazebo Fortress ... 45

3.2.3.2 Επικοινωνία ROS2 Humble με Gazebo Simulator 46
3.2.3.3 SlamToolbox 46
3.2.3.4 Nav2 47
3.2.3.5 RTAB-Map 47
3.3. Ανάπτυξη Λογισμικού Προσομοίωσης 47
3.3.1 Πακέτο Προσομοίωσης – FOSSBot_Sim 48
3.3.2 Μοντέλο FOSSBot 50
3.3.2.1 Αρχεία meshes 50
3.3.2.2 Αρχείο μοντέλου - model.sdf 50
3.3.3 Περιβάλλον Προσομοίωσης 63
3.3.4 Εκκίνηση Προσομοίωσης 66
3.3.4.1 Γέφυρες – Bridges 67
3.3.4.2 Εκδότης Κατάστασης Ρομπότ – Robot State Publisher 68
3.3.4.3 Topics ROS2 69
3.3.4.4 RViz2 69
3.3.5 Δημιουργία bash script εκκίνησης 70
3.4. Σενάρια Λειτουργίας 71
3.4.1 Τηλεχειρισμός – Teleoperation 71
3.4.2 Υλοποίηση Χαρτογράφησης με Laser 73
3.4.2.1 TF TREE 74
3.4.2.2 Αρχείο YAML 75
3.4.2.3 Διαδικασία Χαρτογράφησης και αποτέλεσμα 76
3.4.3 Υλοποίηση Χαρτογράφησης με Laser και κάμερα 78
3.4.3.1 Αρχείο εκκίνησης - rtabmap.py 78
3.4.3.2 Διαδικασία Χαρτογράφησης και αποτέλεσμα 80
3.4.4 Υλοποίηση Χαρτογράφησης με κάμερα 82
3.4.5 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης με LiDAR 85
3.4.5.1 Localization 86
3.4.5.2 Behavior Tree Navigator 87
3.4.5.3 Local Planner 88
3.4.5.4 Global Planner 91

3.4.5.5 Map Server ... 91
3.4.5.6 Behavior Server ... 91
3.4.5.7 Costmaps ... 92
3.4.5.8 Collision Monitor ... 93
3.4.5.9 Velocity Smoother ... 94
3.4.5.10 Διαδικασία Αυτόνομης Πλοήγησης ... 95
3.4.6 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης με κάμερα και LiDAR ... 96
3.4.6.1 Αρχείο εκκίνησης - nav2rtab.py ... 97
3.4.6.2 Αρχείο παραμέτρων - nav2rtab.yaml ... 98
3.4.6.3 Διαδικασία Αυτόνομης Πλοήγησης ... 98
3.4.7 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης με κάμερα ... 99
3.4.8 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης για Χαμηλά Εμπόδια ... 101
3.5. Αξιολόγηση Πλατφόρμας Προσομοίωσης ... 104
3.5.1 Αξιολόγηση Αισθητήρων ... 104
3.5.2 Σύγκριση χαρτών ... 107
3.5.3 Σύγκριση αυτόνομης πλοήγησης ... 110
Κεφάλαιο 4 - Συμπεράσματα - Μελλοντικές επεκτάσεις ... 115
4.1. Συμπεράσματα ... 115
4.1.1 Σύγκριση των Αισθητήρων ... 115
4.1.2 Ανίχνευση των Χαμηλών Εμποδίων ... 116
4.1.3 Βελτιστοποίηση της Nav2 ... 116
4.1.4 Το Περιβάλλον Προσομοίωσης ... 116
4.1.5 Εκπαιδευτική Αξιοποίηση ... 117
4.2. Μελλοντικές επεκτάσεις ... 117
4.2.1 Πραγματοποιημένες Ενέργειες Κατασκευής Φυσικού FOSSBot ... 117
4.2.1.1 Τρισδιάστατη Εκτύπωση Οχήματος ... 117
4.2.1.2 Συναρμολόγηση Ηλεκτρονικών Εξαρτημάτων ... 118
4.2.1.3 Ενσωμάτωση Προηγμένων Αισθητήρων ... 119
4.2.2 Μελλοντικές Ενέργειες ... 120
Συντομογραφίες ... 121
Βιβλιογραφία ... 123

Λίστα Εικόνων

Εικόνα 2-1. FOSSBot 16
Εικόνα 2-2. Raspberry Pi Zero 2 W 17
Εικόνα 2-3. H-Bridge L298N 18
Εικόνα 2-4. DC Gear Motor TT-120RPM 18
Εικόνα 2-5. MPU6050 19
Εικόνα 2-6. Αισθητήρας διακοπής δέσμης φωτός 19
Εικόνα 2-7. Κωδικοποιητής τροχών 19
Εικόνα 2-8. HC-SR04 20
Εικόνα 2-9. Φωτοαντίσταση KS0028 20
Εικόνα 2-10. Αισθητήρας Απόστασης Υπερύθρων 21
Εικόνα 2-11. Θήκη Μπαταριών 21
Εικόνα 2-12. HX-3S-01 22
Εικόνα 2-13. Δομή λογισμικού FOSSBot 22
Εικόνα 2-14. SLAMTEC RPLIDAR C1 26
Εικόνα 2-15. Raspberry Pi Camera Module V2 26
Εικόνα 3-1. ROS2 Humble Hawksbill 42
Εικόνα 3-2. Εντολή Εγκατάστασης Ubuntu-22.04 σε WSL2 43
Εικόνα 3-3. Αποτέλεσμα εντολής locale 44
Εικόνα 3-4 Δομή Πακέτου fossbot_sim 49
Εικόνα 3-5 Αρχείο SDF αρχή 51
Εικόνα 3-6 Inertial 51
Εικόνα 3-7 Base Collision 52
Εικόνα 3-8 Base Visual 52
Εικόνα 3-9 IMU Visual 53
Εικόνα 3-10 LIDAR Visual 53
Εικόνα 3-11 IMU Sensor 54
Εικόνα 3-12 LiDAR Sensor 55
Εικόνα 3-13 Ultrasonic Sensor 56
Εικόνα 3-14 Camera 57

Εικόνα 3-15 Right Wheel 58
Εικόνα 3-16 Right Wheel 2 59
Εικόνα 3-17 Caster Back 60
Εικόνα 3-18 Joints 61
Εικόνα 3-19 Plugins 62
Εικόνα 3-20 Αρχείο Obstacles World 64
Εικόνα 3-21 Έδαφος 65
Εικόνα 3-22 Εμπόδια 65
Εικόνα 3-23 Gazebo Simulation 66
Εικόνα 3-24 Αρχείο Urdf 68
Εικόνα 3-25 Λίστα ROS2 Topic 69
Εικόνα 3-26 RViz2 70
Εικόνα 3-27 Αρχείο sim.sh 71
Εικόνα 3-28 Ενεργοποίηση Πρόσθετου Teleop 72
Εικόνα 3-29 Teleop με βέλη 73
Εικόνα 3-30 Teleop με πλήκτρα 73
Εικόνα 3-31 SLAM Topics 74
Εικόνα 3-32 TF Tree RViz 74
Εικόνα 3-33 TF Tree PDF 75
Εικόνα 3-34 Εκτέλεση Χαρτογράφησης 77
Εικόνα 3-35 Χάρτης από δεδομένα LiDAR 77
Εικόνα 3-36 Συνάρτηση generate_launch_description 79
Εικόνα 3-37 Αντικείμενο LaunchDescription 80
Εικόνα 3-38 RTAB-Map, RViz2 στην χαρτογράφηση 81
Εικόνα 3-39 Χάρτης από δεδομένα LiDAR και κάμερας 82
Εικόνα 3-40 Παράμετροι αρχείου rtabcam.py 83
Εικόνα 3-41 Παράθυρα κατά την χαρτογράφηση 84
Εικόνα 3-42 Διορθωμένος χάρτης 84
Εικόνα 3-43 Χάρτης από δεδομένα κάμερας 85
Εικόνα 3-44 Παράμετροι amcl 87
Εικόνα 3-45 Παράμετροι bt_navigator 88

Εικόνα 3-46 Παράμετροι controller_server 90
Εικόνα 3-47 Παράμετροι planner_server 91
Εικόνα 3-48 Παράμετροι map_server 91
Εικόνα 3-49 Παράμετροι behavior_server 92
Εικόνα 3-50 local_costmap 93
Εικόνα 3-51 Παράμετροι collision_monitor 94
Εικόνα 3-52 Παράμετροι velocity_smoother 95
Εικόνα 3-53 Αυτόνομη Πλοήγηση με LiDAR 96
Εικόνα 3-54 Κόμβοι επεξεργασίας δεδομένων βάθους 97
Εικόνα 3-55 Πλοήγηση με κάμερα και LiDAR 98
Εικόνα 3-56 Κόμβοι Ανίχνευσης Εμποδίων με Κάμερα 99
Εικόνα 3-57 Πλοήγηση με κάμερα 100
Εικόνα 3-58 RViz2 με Ultrasonic 101
Εικόνα 3-59 Local Costmap με Ultrasonic 102
Εικόνα 3-60 Global Costmap με Ultrasonic 103
Εικόνα 3-61 Collision Monitor με Ultrasonic 103
Εικόνα 3-62 Δεδομένα LiDAR 104
Εικόνα 3-63 Nav2 με ανίχνευση χαμηλού εμποδίου από το Ultrasonic 105
Εικόνα 3-64 Nav2 χωρίς ανίχνευση χαμηλού εμποδίου 106
Εικόνα 3-65 Πρόγραμμα GIMP 107
Εικόνα 3-66 Χάρτης για σύγκριση από LiDAR 108
Εικόνα 3-67 Χάρτης για σύγκριση από LiDAR και κάμερα 108
Εικόνα 3-68 Χάρτης για σύγκριση από κάμερα 108
Εικόνα 3-69 Αποτελέσματα Σύγκρισης Χαρτών 110
Εικόνα 3-70 Σύγκριση Χαρτών σε σχέση με τον χάρτη του LiDAR 110
Εικόνα 3-71 Επιλογή στόχου Πλοήγησης στον LiDAR χάρτη 111
Εικόνα 3-72 Επιλογή στόχου Πλοήγησης στον Fusion χάρτη 111
Εικόνα 3-73 Διακύμανση Χρόνων Πλοήγησης 113
Εικόνα 3-74 Κόλλημα FOSSBot με LiDAR 113
Εικόνα 3-75 Λάθος Costmap με Κάμερα και LiDAR 114
Εικόνα 3-76 Λάθος Costmap με Κάμερα 114

Εικόνα 4-1 Ρυθμίσεις Εκτυπωτή 117
Εικόνα 4-2 Βάση FOSSBot 118
Εικόνα 4-3 Θήκες Κινητήρων 118
Εικόνα 4-4 Βάση Ηλεκτρονικών 118
Εικόνα 4-5 Ηλεκτρονικά Μέρη 119
Εικόνα 4-6 Συγκόλληση Εξαρτημάτων 119
Εικόνα 4-7 Καπάκι Βάσης FOSSBot..... 119

Λίστα Πινάκων

Πίνακας 2-1 Χαρακτηριστικά Ρομποτικών Οχημάτων 10
Πίνακας 3-1 FOSSBot Gazebo Topics 63
Πίνακας 3-2 Αποτελέσματα Σύγκρισης Χαρτών..... 109
Πίνακας 3-3 Χρόνοι Ολοκλήρωσης Πλοήγησης..... 111
Πίνακας 3-4 Επιτυχείς Πλοηγήσεις..... 112
Πίνακας 3-5 Λόγοι Αποτυχίας Πλοήγησης 112

Κεφάλαιο 1 - Εισαγωγή

1.1. Η Ρομποτική στην Εκπαίδευση

Η ρομποτική είναι ένας επιστημονικός τομέας που χρησιμοποιεί μηχανική, ηλεκτρονική, τεχνητή νοημοσύνη, αισθητήρες και συστήματα ελέγχου για τη δημιουργία αυτόνομων ή ημιαυτόνομων ρομπότ. Ένα ρομπότ είναι μια αυτόματη μηχανή που μπορεί να εκτελεί ενέργειες με άνεση και με νοημοσύνη παρόμοια με εκείνη ενός ανθρώπου [1]. Η ρομποτική υπάρχει σήμερα σε όλους τους τομείς της ζωής της ανθρωπότητας διευκολύνοντας και αναβαθμίζοντας όλες τις δραστηριότητες και τις υπηρεσίες στις οποίες χρησιμοποιείται. Ένας από τους τομείς που επωφελείται από την ρομποτική αρκετά χρόνια είναι αυτός της βιομηχανίας και της παραγωγής, όπου χρησιμοποιούνται συχνά ρομποτικοί βραχίονες για την εκτέλεση εργασιών με ακρίβεια και ταχύτητα και αυτοκινούμενα οχήματα (automated guided vehicles - AGVs) ή αυτόνομα κινητά ρομπότ (autonomous mobile robots - AMRs) για την μεταφορά υλικών μέσα στις εγκαταστάσεις [2]. Ένας άλλος τομέας στον οποίο έχει αποδειχθεί σημαντική η ρομποτική είναι η Ιατρική καθώς οι χειρουργοί με την βοήθεια ρομποτικών συστημάτων, όπως το Da Vinci, μπορούν να εκτελούν επεμβάσεις με μεγαλύτερη ακρίβεια [3] και η αποκατάσταση ασθενών που έχουν κινητικά προβλήματα μπορεί να επιταχυνθεί με την χρήση εξωσκελετικών ρομποτικών μηχανισμών [4]. Άλλα παραδείγματα εφαρμογών της ρομποτικής σε διάφορους τομείς της καθημερινής ζωής είναι τα μη επανδρωμένα αεροσκάφη τα οποία χρησιμοποιούνται στην άμυνα και στην γεωργία για την παρακολούθηση των συνόρων και των καλλιεργειών αντίστοιχα, τα ειδικά ρομπότ της αστυνομίας τα οποία βοηθούν στην αντιμετώπιση ύποπτων αντικειμένων όπως οι εκρηκτικοί μηχανισμοί, τα αυτόνομα οχήματα τα οποία είναι ευρέως διαδεδομένα σήμερα και οι ρομποτικές σκούπες οι οποίες υπάρχουν πλέον σε πολλά σπίτια βοηθώντας στο καθάρισμα του δαπέδου.

Η εκπαίδευση έχει επίσης ωφεληθεί από την ρομποτική καθώς διδάσκεται πλέον σε όλες τις βαθμίδες της και έχει αποδειχθεί ότι βοηθάει τους εκπαιδευόμενους να ενισχύσουν τις δεξιότητες τους στις επιστήμες, την τεχνολογία, την μηχανική και τα μαθηματικά (STEM) [5][6]. Επίσης σύμφωνα με το [5] η διδασκαλία της ρομποτικής στην πρωτοβάθμια και την δευτεροβάθμια εκπαίδευση μπορεί μέσω πρακτικών εφαρμογών και ασκήσεων να ενισχύσει την υπολογιστική σκέψη και την ικανότητα επίλυσης προβλημάτων των μαθητών, καθώς μαθαίνουν να αναλύουν ένα πρόβλημα, να σχεδιάζουν αλγόριθμους και να προγραμματίζουν ρομπότ. Ενώ

μέσω ομαδικών εργασιών στη ρομποτική οι μαθητές μαθαίνουν να συνεργάζονται, να μοιράζονται ρόλους και να λύνουν προβλήματα συλλογικά. Παρόμοια στην τριτοβάθμια εκπαίδευση η ρομποτική αναπτύσσει δεξιότητες όπως η ομαδικότητα, η επικοινωνία και η ικανότητα επίλυσης πολύπλοκων προβλημάτων, οι οποίες είναι αναγκαίες για την μελλοντική εργασία των φοιτητών και βοηθάει στην εκμάθηση του προγραμματισμού και των αλγορίθμων, που είναι απαραίτητοι για εργαζόμενους στον τομέα της πληροφορικής και της μηχανικής [7].

1.2. Εκπαιδευτικές Πλατφόρμες Ρομποτικής

Τα βασικά εργαλεία που χρησιμοποιούνται στη διδασκαλία της ρομποτικής είναι οι εκπαιδευτικές πλατφόρμες ρομποτικής, οι οποίες διαχωρίζονται στο λογισμικό και τα κινούμενα ή μη ρομποτικά συστήματα. Το λογισμικό διακρίνεται σε προγραμματιστικά περιβάλλοντα και σε εφαρμογές προσομοίωσης. Ενώ τα προγραμματιστικά περιβάλλοντα, όπως είναι το Scratch [8], το microbit [9] και το blockly [10], χρησιμοποιούνται στην πρωτοβάθμια και στην δευτεροβάθμια εκπαίδευση και ο μαθητής μπορεί να δημιουργήσει σε αυτά ένα πρόγραμμα συνδυάζοντας έτοιμα κομμάτια κώδικα σε μορφή γραφικών αντικειμένων (blocks) χωρίς να χρειάζεται να γράψει τον συγκεκριμένο κώδικα. Ενώ οι εφαρμογές προσομοίωσης, όπως είναι το Gazebo [11] και το CoppeliaSim [12], χρησιμοποιούνται στην τριτοβάθμια εκπαίδευση και ο φοιτητής έχει την δυνατότητα να δημιουργήσει, να χειριστεί και να προγραμματίσει ρομποτικά συστήματα σε ένα εικονικό περιβάλλον.

Τα σταθερά ρομποτικά συστήματα, ελέγχονται από μικροελεγκτές όπως το Arduino [13], και χρησιμοποιούνται στην εκπαίδευση για την μελέτη του ελέγχου αισθητήρων, ενώ τα κινούμενα, όπως το FOSSBot [14][15], το TurtleBot [16] και το LEGO Mindstorms [17], για την μελέτη της πλοήγησης ρομποτικών οχημάτων. Τα κινούμενα συστήματα έχουν κοινά χαρακτηριστικά, όπως είναι το ανοιχτό λογισμικό, η εύκολη επέκταση του υλικού τους και η υποστήριξη από κοινότητες όπως το GitHub. Ωστόσο υπάρχουν διαφορές στις δυνατότητες και το κόστος τους που τα κατατάσσουν σε δύο κατηγορίες:

- Τα εκπαιδευτικά ρομποτικά οχήματα τα οποία έχουν σχεδιαστεί για τα εισαγωγικά μαθήματα της ρομποτικής και του προγραμματισμού. Τα συγκεκριμένα είναι αρκετά οικονομικά καθώς διαθέτουν εξοπλισμό για απλές εφαρμογές πλοήγησης και ελέγχου αισθητήρων. Δύο παραδείγματα τέτοιων οχημάτων είναι το FOSSBot και το LEGO Mindstorms.

- Τα επαγγελματικά και ερευνητικά ρομποτικά οχήματα, όπως είναι το TurtleBot, το TugBot [18] και το MIT Racercar [19], τα οποία έχουν υψηλό κόστος και είναι κατάλληλα για διδασκαλία πάνω σε προχωρημένα θέματα ρομποτικής καθώς υποστηρίζουν SLAM (Simultaneous Localization and Mapping) [20] και αυτόνομη πλοήγηση.

1.3. Ορισμός του προβλήματος

Οι δύο παραπάνω κατηγορίες ρομποτικών οχημάτων διαφέρουν στο κόστος και τις δυνατότητες τους λόγω του διαφορετικού εξοπλισμού τους. Τα οχήματα της πρώτης κατηγορίας ενώ είναι πολύ οικονομικά δεν διαθέτουν εξελιγμένους αισθητήρες, επομένως δεν μπορούν να υποστηρίξουν προηγμένους αλγόριθμους, όπως το SLAM, και δεν υπάρχουν έτοιμες υλοποιήσεις κατάλληλες για μαθήματα σε πανεπιστημιακό επίπεδο. Μια τέτοια οικονομική υλοποίηση είναι το FOSSBot, το οποίο είναι ανοιχτού κώδικα και έχει σχεδιαστεί από την ΕΕΛΛΑΚ με σκοπό την εισαγωγή σε βασικές αρχές ρομποτικής και προγραμματισμού των μαθητών πρωτοβάθμιας και δευτεροβάθμιας εκπαίδευσης. Ο σχεδιασμός του προβλέπει την ενσωμάτωση αισθητήρων απόστασης υπερήχων, οδομετρίας, ανίχνευσης γραμμών, φωτός και IMU (Inertial Measurement Unit), οι οποίοι είναι κατάλληλοι για απλούς αλγόριθμους κίνησης.

Αντίθετα τα ερευνητικά ρομποτικά οχήματα διαθέτουν προηγμένους αισθητήρες, όπως το LiDAR (Light Detection and Ranging) και η κάμερα, καθιστώντας τα κατάλληλα για την εφαρμογή σύνθετων αλγορίθμων πλοήγησης. Επίσης λόγω του εξοπλισμού τους μπορούν να υποστηρίξουν την δοκιμή λογισμικού για αυτόνομα ρομποτικά οχήματα, το οποίο έχει συγγραφεί σε περιβάλλον ROS2 (Robotic Operating System version 2), όμως έχουν πολύ υψηλό κόστος και δεν είναι εφικτό για πολλά πανεπιστήμια να τα εντάξουν στα εκπαιδευτικά εργαλεία τους.

Συνεπώς όπως προκύπτει και από το [21] δεν υπάρχει ένα χαμηλού κόστους και ανοιχτού κώδικα ρομποτικό όχημα που να συνδυάζει αισθητήρες LiDAR, κάμερας, IMU και οδομετρίας ώστε να είναι εξίσου κατάλληλο για την εισαγωγή στη ρομποτική σε μαθητές και για την υλοποίηση αλγορίθμων SLAM και αυτόνομης πλοήγησης από φοιτητές. Το FOSSBot θα έχει τις προδιαγραφές και τις δυνατότητες μιας τέτοιας πλατφόρμας εφόσον επεκταθεί το υλικό του, αναβαθμιστεί το λογισμικό του, χρησιμοποιηθεί το περιβάλλον ROS2 και συνοδευτεί με εκπαιδευτικά σενάρια και λογισμικό προσομοίωσης έτοιμο για πανεπιστημιακή διδασκαλία.

1.4. Στόχος διατριβής

Στόχος αυτής της διατριβής είναι: α) Η επιτυχής ενσωμάτωση των αρχικών και των νέων αισθητήρων στο πλαίσιο του FOSSBot διασφαλίζοντας τη στερέωση τους και την τροφοδοσία τους, β) η ανάπτυξη λογισμικού για την ενσωμάτωση των νέων αισθητήρων σε αυτό και για την αναβάθμιση του μοντέλου προσομοίωσης του σε περιβάλλον Gazebo, γ) η υλοποίηση και η δοκιμή, σε περιβάλλον ROS2, αλγορίθμων SLAM χρησιμοποιώντας τα δεδομένα από τον LiDAR, το IMU και την οδομετρία και αλγορίθμων αυτόνομης πλοήγησης βασιζόμενων στους χάρτες που θα δημιουργηθούν από τους πρώτους αλγόριθμους, δ) η δημιουργία οδηγιών και σεναρίων χρήσης για τη χρησιμοποίηση του FOSSBOT από φοιτητές τόσο σε πραγματικά σενάρια όσο και σε προσομοιώσεις και ε) η αξιολόγηση της απόδοσης του συστήματος όσον αφορά την ακρίβεια της χαρτογράφησης και του εντοπισμού της θέσης του.

Τα αναμενόμενα αποτελέσματα της διατριβής είναι:

- Μια πλήρως λειτουργική και αναβαθμισμένη έκδοση του FOSSBOT με την ενσωμάτωση των αρχικών αισθητήρων, των αισθητήρων LiDAR και κάμερας και όλων των επιπλέον τροποποιήσεων που θα προκριθούν να γίνουν.
- Ένα λογισμικό ανοιχτού κώδικα βασισμένο στο ROS2 το οποίο θα υλοποιεί αλγόριθμους SLAM και πλοήγησης για το ρομπότ.
- Ένα σετ οδηγιών εγκατάστασης και σεναρίων χρήσης του λογισμικού, έτοιμο για χρήση σε εργαστήρια ρομποτικής των ΑΕΙ.

Κεφάλαιο 2 - Σχετικές Τεχνολογίες

2.1. Ρομποτικά Οχήματα

2.1.1 Εισαγωγή

Τα τελευταία χρόνια, έχει υλοποιηθεί και δοκιμαστεί ένας μεγάλος αριθμός από ρομποτικά οχήματα στη διδασκαλία STEM σε όλα τα επίπεδα της εκπαίδευσης βοηθώντας τον μαθητή να κατανοήσει ευκολότερα σύνθετες και αφηρημένες έννοιες [21]. Υπάρχει μεγάλη ποικιλία υλοποιήσεων οχημάτων με διαφορετικό κόστος και δυνατότητες, τα οποία μπορεί να είναι προ-συναρμολογημένα ή να είναι της φιλοσοφίας κάντο μόνος σου (DIY) ώστε οι μαθητές μέσα από την κατασκευή, την τροποποίηση και την δοκιμή να αποκτούν τις επιθυμητές γνώσεις με εφαρμοσμένο τρόπο [22]. Επίσης ο προγραμματισμός και ο έλεγχος τους μπορεί να γίνει είτε με την χρήση μπλοκ είτε με προχωρημένες γλώσσες προγραμματισμού ανάλογα με το επίπεδο των μαθητών. Στην παρούσα ενότητα μέσα από άρθρα, που υπάρχουν στην επίσημη ιστοσελίδα του IEEExplore και έχουν δημοσιευθεί από το 2020 και μετά, παρουσιάζονται οι κατηγορίες, τα χαρακτηριστικά, οι περιορισμοί και οι τρόποι αξιολόγησης δημοφιλών ρομποτικών οχημάτων τα οποία είναι κατάλληλα είτε για διδασκαλία σε προχωρημένα θέματα STEM είτε για τις πρώτες βαθμίδες της εκπαίδευσης. Τέλος αναλύονται τα κενά και οι προκλήσεις που παρατηρούνται στη ρομποτική και την αυτόνομη πλοήγηση, οι προσπάθειες αντιμετώπισης τους και οι προτάσεις για περαιτέρω έρευνα.

2.1.2 Κατηγορίες

2.1.2.1 Ρομποτικά οχήματα για μαθητές ηλικίας μέχρι 12 χρονών

Στην πρώτη κατηγορία ανήκουν τα ρομποτικά οχήματα ColourBot, SMARTmBOT και FOSSBot, τα οποία έχουν σχετικά χαμηλό κόστος και σκοπός τους είναι η εισαγωγή των μαθητών, της πρωτοβάθμιας και δευτεροβάθμιας εκπαίδευσης, σε βασικές έννοιες του προγραμματισμού και της ρομποτικής μέσω της διδασκαλίας STEM [23]. Τα δύο πρώτα παρουσιάζονται παρακάτω ενώ το FOSSBot θα αναλυθεί στο επόμενο κεφάλαιο.

- Το ColourBot, το οποίο παρουσιάζεται στο [23], είναι ένα αρκετά φθηνό ρομποτικό όχημα, ανοιχτού κώδικα, αρθρωτού σχεδιασμού και τα μέρη του πλαισίου του εκτυπώνονται με τρισδιάστατο εκτυπωτή και συναρμολογούνται

εύκολα μέσω εξωτερικών ελαστικών χειλιών. Διαθέτει μικροελεγκτή Arduino Nano, χρησιμοποιεί τυπωμένες πλακέτες κυκλωμάτων (PCB) σχεδιασμένες με το ελεύθερο λογισμικό σχεδίασης KiCad [24] και επιτρέπει την ταυτόχρονη σύνδεση μέχρι 13 εξωτερικών συσκευών μέσω Type-C υποδοχών. Το λογισμικό που χρησιμοποιεί είναι μια αναβαθμισμένη έκδοση του Ardublockly [25], το οποίο είναι σχεδιασμένο για μαθητές πρωτοβάθμιας και δευτεροβάθμιας εκπαίδευσης και τους επιτρέπει να δημιουργούν κώδικα Arduino με οπτικά μπλοκ. Για την τεκμηρίωση του αναπτύχθηκε ένα απλό πρόγραμμα το οποίο ανιχνεύει εμπόδια ,σε εύρος από 2 έως 40 cm, με την χρήση αισθητήρα σόναρ και σταματάει όταν η απόσταση γίνει μικρότερη από 8 cm.

- Το [26] αναφέρει ότι το SMARTmBOT είναι ρομποτικό όχημα ανοικτού κώδικα, το οποίο βασίζεται σε ROS2. Στον εξοπλισμό του περιλαμβάνει Raspberry Pi 4, 8 αισθητήρες ToF, 2 αισθητήρες γραμμής και μία κάμερα Pi, γεγονός που του επιτρέπει να εκτελεί αλγόριθμους κίνησης όπως η κατεύθυνση προς ένα στόχο (go-to-goal), η καταδίωξη (pure-pursuit), η αποφυγή εμποδίων, η ακολούθηση γραμμής και το σμήνος (swarming). Το SMARTmBOT προορίζεται για χρήση σε ηλικίες μέχρι 12 χρονών, όμως είναι αρκετά πολύπλοκο και διαθέτει υπολογιστή μίας πλακέτας (SBC) επομένως θα ήταν καταλληλότερο για την τριτοβάθμια εκπαίδευση.

2.1.2.2 Ρομποτικά οχήματα για Πανεπιστήμια και έρευνα

Η δεύτερη κατηγορία περιλαμβάνει τα ρομπότ LEGO Mindstorms, Duckiebot, AWS DeepRacer, F1TENTH, TurtleBot3, JetRacer, MIT RACECAR , Roomba και SROS. Τα ρομποτικά οχήματα αυτά σχεδιάζονται για να μπορούν χρησιμοποιηθούν στην έρευνα και τη διδασκαλία προηγμένων θεμάτων όπως η αυτόνομη οδήγηση, ο έλεγχος της κίνησης και η τεχνητή νοημοσύνη [27].

- Το LEGO Mindstorms, σύμφωνα με το [27] και του [28], επικοινωνεί μέσω Bluetooth με τον υπολογιστή και το λογισμικό του αναπτύσσεται μέσω Java σε γραφικό περιβάλλον (GUI). Διαθέτει αισθητήρα υπερήχων και κάμερα που του επιτρέπουν να πραγματοποιεί οπτική ανίχνευση και δημιουργία 2D χάρτη του περιβάλλοντος. Χρησιμοποιεί τον αλγόριθμο Monte Carlo για να δημιουργήσει

αυτόν τον δισδιάστατο χάρτη και να εντοπίσει την θέση του ενώ η κάμερα το βοηθάει να βελτιώσει την σωστή αντιστοίχιση της θέσης του στον χάρτη και να επιτύχει ποσοστό επιτυχία περίπου 80%.

- Στο [29] παρουσιάζεται το Duckiebot, ένα ρομποτικό όχημα χαμηλού κόστους και ανοιχτού κώδικα το οποίο χρησιμοποιεί Raspberry Pi 2 για να εκτελεί τους υπολογισμούς και μια μονοφθάλμια κάμερα (monocular camera) για να έχει αντίληψη του χώρου στον οποίο κινείται. Το Duckiebot έχει σχεδιαστεί για να υποστηρίξει τη διδασκαλία σε προπτυχιακά και μεταπτυχιακά μαθήματα πάνω στην αυτόνομη οδήγηση και να εκτελεί τις λειτουργίες της παρακολούθησης γραμμής, αποφυγής εμποδίων και εντοπισμού της θέσης του.
- Το AWS DeepRacer [30][31] της Amazon Web Services ενώ διαθέτει αισθητήρες LiDAR, IMU και δύο κάμερες, η χρήση του στην εκπαίδευση είναι πολύ δύσκολη γιατί έχει πολύ υψηλό κόστος. Το συγκεκριμένο όχημα χρησιμοποιεί για την εκτέλεση των λειτουργιών του το περιβάλλον λογισμικού ROS, για την προσομοίωση του το Gazebo και για την εκπαίδευση των πρακτόρων της ενισχυτικής μάθησης [32] τους αλγόριθμους Proximal Policy Optimization (PPO) [33] και Soft Action Critic (SAC) [34].
- Στο [35] παρουσιάζεται το F1TENTH το οποίο ένα αυτόνομο όχημα ανοικτής αρχιτεκτονικής κατάλληλο για τη διδασκαλία της ρομποτικής σε δευτεροβάθμια και τριτοβάθμια εκπαίδευση λόγω του αρθρωτού υλικού και λογισμικού του. Χρησιμοποιεί NVIDIA Jetson Xavier NX για την επεξεργασία, δισδιάστατο LiDAR και στερεοσκοπική κάμερα για να αντιλαμβάνεται το περιβάλλον του και δισδιάστατο προσομοιωτή για την γρήγορη δοκιμή του κώδικα. Επίσης ενσωματώνει το σύστημα διεύθυνσης Ackerman, το οποίο είναι μια γεωμετρική διάταξη συνδέσμων στο σύστημα διεύθυνσης ενός οχήματος που έχει σχεδιαστεί για να λύσει το πρόβλημα των τροχών στο εσωτερικό και στο εξωτερικό μιας στροφής, περιστρέφοντας τους σε διαφορετικές γωνίες, ώστε τελικά να περιστρέφονται γύρω από ένα κοινό κεντρικό σημείο κατά τη στροφή [36]. Το F1TENTH μπορεί να φτάσει ταχύτητα έως 60 km/h και να χρησιμοποιηθεί σε ανταγωνιστικό περιβάλλον "Autonomous Racing", όμως η τιμή του και η πολυπλοκότητα του υλικού και του λογισμικού του είναι πάρα πολύ υψηλές.

- Το JetRacer 4WS [37] έχει υψηλό κόστος και είναι βασισμένο στο JetRacer [38] με την προσθήκη τετραδιεύθυνσης (Four-Wheel Steering - 4WS), η οποία παρέχει βελτιωμένη ευελιξία και σταθερότητα. Το συγκεκριμένο όχημα μπορεί να χρησιμοποιηθεί σε εσωτερικούς και εξωτερικούς χώρους, διαθέτει υπολογιστή NVIDIA Jetson Nano και το σύστημα εντοπισμού θέσης Marvelmind, το οποίο βασίζεται σε υπερήχους και έχει υψηλή ακρίβεια σφάλματος έως ±2 cm [39]. Επίσης έχει σχεδιαστεί για την δοκιμή αλγορίθμων ελέγχου κίνησης, όπως η τεχνική event-triggered του MPC, η οποία χρησιμοποιεί σύστημα εντοπισμού θέσης (ground truth) αντί για αισθητήρες αντίληψης και μειώνει την υπολογιστική πολυπλοκότητα, έως 10%, σε σχέση με την προσομοίωση [40].
- Το MIT RACECAR σύμφωνα με το [41] μπορεί να χρησιμοποιηθεί για την εκπαίδευση μαθητών λυκείου σε προχωρημένα θέματα ρομποτικής, σε πανεπιστήμια για την ανάπτυξη αλγορίθμων αυτόνομης χαρτογράφησης και στην έρευνα για την σύγκριση διαφόρων συστημάτων SLAM. Το συγκεκριμένο όχημα είναι εξοπλισμένο με υπολογιστή Nvidia TX2, δισδιάστατο LiDAR, στερεοσκοπική κάμερα και IMU και λόγω του ανοιχτού κώδικα του είναι εύκολη η πρόσθεση περισσότερων αισθητήρων, όμως το υψηλό κόστος του περιορίζει την ευρεία χρήση του στην εκπαίδευση.
- Το Roomba είναι μια οικιακή ηλεκτρική σκούπα εξοπλισμένη με κωδικοποιητές τροχών και αισθητήρες πρόσκρουσης, η οποία είναι ιδανική για την έρευνα πάνω στη ρομποτική καθώς μπορεί να χρησιμοποιηθεί για την δοκιμή αλγορίθμων χωρικής συλλογιστικής όπως είναι ο Monte Carlo και ο FastSLAM. Όμως στα μειονεκτήματα της συγκαταλέγεται η σημαντική απόκλιση που έχει η οδομετρία της, ειδικά στις στροφές και ο μεγάλος κώνος ανίχνευσης που έχουν οι αισθητήρες πρόσκρουσης της και μπορεί να φτάσει έως τις 60°, σε χαμηλούς ρυθμούς δειγματοληψίας [42].
- Το SROS είναι σχεδιασμένο για την επίδειξη SLAM και αυτόνομης πλοήγησης σε περιβάλλον ROS χρησιμοποιώντας τους αλγόριθμους Gmapping και Particle Filter. Ο εξοπλισμός του περιέχει εγκέφαλο Raspberry Pi και 3D κάμερα βάθους Kinect, η οποία επιτρέπει τη λήψη 3D δεδομένων για τη χαρτογράφηση. Όμως η

χαρτογράφηση αυτή είναι αργή και η μεταφορά των δεδομένων στο Rviz έχει μεγάλη καθυστέρηση [43].

- Το TurtleBot3 [44] είναι ένα δημοφιλές ρομποτικό όχημα με αρθρωτό σχεδιασμό εξοπλισμένο με πλακέτα OpenCR, η οποία είναι βασισμένη σε Arduino Uno, με εγκέφαλο Raspberry Pi 3 και με αισθητήρες LiDAR 360°, IMU και κωδικοποιητές τροχών (encoders). Χρησιμοποιεί λογισμικό ROS και υποστηρίζει προηγμένους αλγόριθμους, όπως τους Gmapping και Cartographer, εκτελώντας ακριβή 2D χαρτογράφηση και εντοπισμό θέσης [45]. Είναι ιδανικό για πειραματισμό σε πραγματικές συνθήκες και σε προσομοίωση Gazebo [46], όμως έχει αρκετά υψηλό κόστος γεγονός που το εμποδίζει να χρησιμοποιείται αρκετά στην εκπαίδευση. Επίσης διατίθεται σε διάφορες εκδόσεις, ανάλογα με τις εφαρμογές που προορίζεται, όπως η Waffle Pi και η Burger.

2.1.2.3 Ρομποτικά οχήματα με χαμηλό κόστος και μεγάλη επεκτασιμότητα

Σε αυτή την κατηγορία συγκαταλέγονται ρομποτικά οχήματα που σχεδιάστηκαν ώστε να έχουν χαμηλή τιμή και πολυπλοκότητα με σκοπό να είναι κατάλληλα για την εκπαίδευση σε όλα τα επίπεδα της και για την έρευνα σε ένα ευρύ φάσμα εφαρμογών. Όμως τα συγκεκριμένα οχήματα δεν είναι πάντα ιδανικά για την εκπαίδευση καθώς σπάνια έχουν ολοκληρωμένη προσομοίωση και το εκπαιδευτικό υλικό τους δεν καλύπτει όλα τα επίπεδα της διδασκαλίας της ρομποτικής, παρουσιάζοντας κενά στην σύνδεση μεταξύ απλών και επαγγελματικών εφαρμογών. Μερικά δημοφιλή οχήματα της κατηγορίας αυτής είναι το MBot, το UiAbot και το Flower∞Bot.

- Όπως αναφέρεται στο [47] το ρομποτικό όχημα MBot του πανεπιστημίου του Michigan έχει σχεδιαστεί με 4 διαφορετικούς συνδυασμούς υλικού, των οποίων η τιμή εξαρτάται από τις δυνατότητες τους. Οι εκδόσεις του συγκεκριμένου οχήματος είναι η Basic, η Classic, η Omni, και η Balance με τιμές που κυμαίνονται σε μερικές εκατοντάδες ευρώ. Στον εξοπλισμό του, ανάλογα την έκδοση, εμπεριέχεται ο μικροελεγκτής RP2040, ο εγκέφαλος Raspberry Pi ή ο Jetson Nano και οι αισθητήρες LiDAR και κάμερα. Το MBot προορίζεται για φοιτητές αρχάριους στον προγραμματισμό και για τον λόγο αυτό προσφέρει ένα απλοποιημένο API (Application Programming Interface) σε όλες τις εκδόσεις

του. Επίσης στις ακριβότερες εκδόσεις μπορεί να υποστηρίξει εφαρμογές αυτόνομης οδήγησης όπως το SLAM και ο εντοπισμός θέσης.

- Το UiAbot σχεδιάστηκε, υλοποιήθηκε και δοκιμάστηκε από μια ομάδα μεταπτυχιακών φοιτητών του Πανεπιστημίου του Agder στην Νορβηγία. Το συγκεκριμένο ρομποτικό όχημα έχει χαμηλό κόστος, κατασκευάζεται με τρισδιάστατη εκτύπωση και διαθέτει τον ελεγκτή ODrive V3.6, το RPLiDAR A1, την IMU BNO055 και συστοιχία μπαταριών Li-ion υψηλής χωρητικότητας. Ο συνδυασμός του υλικού και του λογισμικού του, το οποίο περιέχει το περιβάλλον λογισμικού ROS2 και το πλαίσιο πλοήγησης Nav2, το καθιστά κατάλληλο για SLAM και αυτόνομη πλοήγηση με έναν αρκετά ακριβή έλεγχο κίνησης [48].
- Το Flower∞Bots [49] διαθέτει 3 μοντέλα το Lily∞Bot, το Daisy∞Bot και το Rosie∞Bot που προορίζονται για διαφορετικά επίπεδα εκπαίδευσης. Το πλαίσιο τους μπορεί να εκτυπωθεί σε 3D εκτυπωτή, αν και για το μοντέλο Rosie που είναι το πιο μεγάλο υπάρχει δυσκολία σε μικρούς εκτυπωτές. Ο εγκέφαλος του μπορεί να είναι Arduino Uno υποστηρίζοντας έλεγχο κίνησης και αποφυγή εμποδίων ή κάποιος πιο εξελιγμένος υποστηρίζοντας SLAM και εντοπισμό θέσης.

2.1.2.4 Βασικά χαρακτηριστικά

Τα ρομποτικά οχήματα που αναλύονται στις προηγούμενες παραγράφους παρουσιάζουν μεγάλη ποικιλία αισθητήρων και διαφορετικές κεντρικές μονάδες επεξεργασίας. Στον Πίνακας 2-1 παρουσιάζονται αυτά τα χαρακτηριστικά για κάθε ρομπότ.

Πίνακας 2-1 Χαρακτηριστικά Ρομποτικών Οχημάτων

Όνομα	Αισθητήρες	Εγκέφαλος	3D Print	Κύριες Εφαρμογές	Τιμή €
ColourBot	Sonar, IR, Line tracking, Φωτοαντίσταση	Arduino Nano	Ναι	STEM σε Πρωτοβάθμια και Δευτεροβάθμια	~50
SMARTmBOT	8x ToF Laser (VL53L0X), 2x Line sensors, Pi Camera	Raspberry Pi 4	Ναι	Σμήνη (Swarms), HRI, Αυτόνομη Πλοήγηση	~210

LEGO Mindstorms	Ultrasonic, Camera	NXT	Όχι	Χαρτογράφηση, Εύρεση θέσης	~360
Duckiebot	Μονοεστιακή κάμερα	Raspberry Pi 2/3/4	Όχι	Line following, πλοήγηση σε σμήνη	150 – 450
AWS DeepRacer	Stereo Camera, LiDAR, IMU	Intel Processor	Όχι	Reinforcement Learning, Racing	250 – 400
F1TENTH	2D LiDAR, RGBD / Stereo Camera, IMU, Encoders	NVIDIA Jetson TX2 / Xavier NX	Ναι	Αυτόνομοι Αγώνες, Έρευνα	3000 – 4500
JetRacer 4WS	Marvelmind, Encoders	NVIDIA Jetson Nano	Ναι	Τετραδιεύθυνση (4WS), MPC	~850
MIT RACECAR	Hokuyo LiDAR (30m), ZED Stereo Camera, Encoders	NVIDIA Jetson TX1/TX2	Ναι	Agile Driving, SLAM, AI	2600 – 3000
Roomba	Bump sensors, IR Wall, Cliff sensors, Encoders	SCI API control	Όχι	Αλγόριθμοι κάλυψης, Έρευνα SLAM	~130 – 150
SROS	Microsoft Kinect (3D), Wheel Encoders	RPi3, Arduino Mega	Όχι	SLAM, Αυτόνομη Πλοήγηση, Εκπαίδευση	< 200
TurtleBot 3	360° LiDAR, IMU, Encoders, Κάμερα	Raspberry Pi 3/4	Ναι	SLAM (Gmapping), Πλοήγηση (Nav2)	586 – 1400
TurtleBot 4	OAK-D Lite camera, LiDAR, IMU	Raspberry Pi 4	Όχι	Αυτόνομη Πλοήγηση	1200
MBot (Basic)	7-seg Line sensor, Encoders	RP2040 (Pico)	Ναι	Εισαγωγή στον προγραμματισμό	100
MBot (Classic)	2D LiDAR, Camera, Encoders, IMU	RPi4/Nano & RP2040	Ναι	SLAM, Αυτόνομη Πλοήγηση	285 – 375

MBot (Balance)	IMU (9-axis), Encoders	RP2040	Ναι	Έλεγχος Κίνησης	~100+
UiAbot	RPLiDAR A1, IMU (BNO055), Encoders	NVIDIA Jetson Nano	Ναι	Nav2, Έρευνα, SLAM	-
Lily∞Bot	Sonar, IR, photo-resistor	Arduino Uno	Ναι	STEM σε 1βάθμια και 2βάθμια Εκπ.	~50
Daisy∞Bot	Προσθήκη έως 14 αισθητήρων	Arduino Mega	Ναι	Πανεπιστήμια	~50+
Rosie∞Bot	Προσθήκη έως 22 αισθητήρων	Arduino Mega /Expert	Ναι	Έρευνα, AI, Swarming	~50+

2.1.3 Αξιολόγηση Ρομποτικών Οχημάτων

Τα πιο εξελιγμένα από τα παραπάνω ρομποτικά οχήματα έχουν την δυνατότητα της χαρτογράφησης του χώρου στον οποίο κινούνται, τον εντοπισμό της θέσης τους και την αυτόνομη πλοήγηση μέσα στον δημιουργημένο χάρτη. Κάθε όχημα όμως έχει διαφορετική απόδοση στις εν λόγω εφαρμογές και είναι αναγκαίο να υπολογίζεται η ακρίβεια των υπολογισμών και το ποσοστό σφάλματος τους ώστε να είναι δυνατή η σύγκριση και η αξιολόγηση τους. Ο τρόπος υπολογισμού του σφάλματος εξαρτάται από την εφαρμογή και τους αισθητήρες που διαθέτει το κάθε όχημα. Οι κύριοι τρόποι για τον υπολογισμό αυτόν είναι:

- Αξιολόγηση Χαρτογράφησης

  Η επιτυχία των αλγορίθμων χαρτογράφησης στο ρομποτικό όχημα Roomba αξιολογείται από την ικανότητα ευθυγράμμισης των τοίχων σε σχέση με την πραγματική απόσταση μεταξύ τους [42]. Δηλαδή η ιδανική ευθυγράμμιση σημαίνει ότι η απόσταση μεταξύ δύο τοίχων που μετρά το ρομπότ είναι πολύ κοντά στην πραγματική φυσική απόσταση. Στο συγκεκριμένο όχημα η χρήση διορθωμένου μοντέλου οδομετρίας έδωσε μικρότερες αποκλίσεις από ό,τι βασική οδομετρία. Στο [28] παρουσιάζεται η δημιουργία 2D χάρτη με αισθητήρα υπερήχων από το LEGO Mindstorms NXT και η αξιολόγηση της χαρτογράφησης, συγκρίνοντας τις αποστάσεις και τις γωνίες στον συγκεκριμένο χάρτη με τις πραγματικές διαστάσεις του περιβάλλοντος. Στο ίδιο άρθρο αναφέρεται ότι τα σφάλματα στην

χαρτογράφηση μπορεί να προέρχονται από πολλές αιτίες όπως είναι ο θόρυβος των αισθητήρων και τα σφάλματα της οδομετρίας . Ενώ στο [50] επισημαίνεται η αρνητική επίπτωση των περιορισμών στη RAM και την επεξεργαστική ισχύ στην ακρίβεια των εκτιμήσεων λόγω της πιθανής καθυστέρησης που προκαλούν.

- Μέτρηση Σφάλματος Εντοπισμού Θέσης και Πλοήγησης

  Το συγκεκριμένο σφάλμα είναι η διαφορά ανάμεσα στην εκτιμώμενη και την πραγματική θέση του οχήματος. Για την καλύτερη αξιολόγηση χρησιμοποιείται συχνά μια εξωτερική πηγή δεδομένων που παρέχει την πραγματική θέση και γωνία του ρομπότ ως αληθινή στάση [37]. Στο LEGO Mindstorms μετρήθηκε το σφάλμα του αλγόριθμου εντοπισμού της θέσης MonteCarlo (MCL), συγκρίνοντας την θέση των σωματιδίων στον χάρτη και της πραγματικής θέσης, η οποία υποδεικνύεται από μια κάμερα [28]. Ενώ στο [51] για τον υπολογισμό του σφάλματος πραγματοποιούνται 15 επαναλήψεις της ίδιας διαδρομής μεταξύ σταθερών σημείων αναφοράς και στη συνέχεια σε διαγράμματα απεικονίζεται η απόκλιση των σημείων του οχήματος από τα σημεία στόχου. Η άφιξη στον στόχο θεωρήθηκε επιτυχής όταν το σφάλμα δεν ήταν πάνω από 10 cm. Παρόμοια στο [52] συγκρίνεται η επιθυμητή θέση με τα δεδομένα άφιξης και στη συνέχεια υπολογίζεται το σφάλμα λαμβάνοντας υπόψη την μέση ακρίβεια της θέσης σε εκατοστά και την μέση ακρίβεια γωνίας σε μοίρες. Στην αξιολόγηση της επίδοσης του JetRacer-4WS χρησιμοποιήθηκε το Root Mean Square Error (RMSE), το οποίο μετρά το μέσο μέγεθος των σφαλμάτων μεταξύ των προβλεπόμενων τιμών και των πραγματικών παρατηρούμενων τιμών. Επίσης λήφθηκε υπόψη το Μέγιστο Σφάλμα στην πλευρική απόκλιση το οποίο ορίζει τη μέγιστη αποδεκτή διαφορά μεταξύ της μετρούμενης και της αναμενόμενης θέσης [37]. Στο [53] παρουσιάζεται η αξιολόγηση του JetBot μέσα από την δοκιμή πλοήγησης σε διαφορετικά εσωτερικά περιβάλλοντα και χρησιμοποιώντας την Λειτουργία Αθροιστικής Κατανομής (CDF), όπου το διάγραμμα CDF απεικονίζει τα σφάλματα πλοήγησης μεταξύ της διαδρομής του ρομπότ και της πραγματικότητας του εδάφους. Τέλος σε κάποια άρθρα στην αξιολόγηση του εντοπισμού θέσης λαμβάνεται υπόψη και ο απαιτούμενος αριθμός επαναλήψεων για τον ακριβή εντοπισμό της θέσης [28].

2.1.4 Προκλήσεις

Στα ρομποτικά οχήματα υπάρχουν αρκετές προκλήσεις, οι οποίες διαφέρουν ανάλογα με την κατηγορία του οχήματος και συνήθως αφορούν ζητήματα όπως το κόστος, το υλικό και τις διαφορές ανάμεσα στην προσομοίωση και την πραγματικότητα. Στα άρθρα των ρομποτικών οχημάτων που παρουσιάζονται στην παράγραφο 2.1.2 αναφέρονται οι κυριότερες προκλήσεις και παρουσιάζονται παρακάτω:

- Τα ρομποτικά οχήματα που διαθέτουν προηγμένους αισθητήρες και δυνατότητες έχουν συνήθως υψηλό κόστος και επομένως είναι δύσκολο να χρησιμοποιούνται ευρέως στην εκπαίδευση και στην έρευνα. Για παράδειγμα το F1/10 και το MIT RACECAR είναι εξοπλισμένα με τέτοιους αισθητήρες, όπως είναι το LiDAR υψηλής ταχύτητας και εμβέλειας και στοιχίζουν μερικές χιλιάδες δολάρια.
- Επίσης τα οχήματα με μικρό κόστος έχουν συνήθως μικρή επεξεργαστική ισχύ με αποτέλεσμα να προκαλείται συχνά καθυστέρηση στην επεξεργασία των δεδομένων. Αυτή η καθυστέρηση επιφέρει μείωση στην ακρίβεια των αλγορίθμων πλοήγησης σε εφαρμογές με υψηλό φόρτο εργασίας όπως την ταυτόχρονη χαρτογράφηση και τον εντοπισμό της θέσης [54].
- Τα εκπαιδευτικά οχήματα, όπως το TurtleBot 3, χρειάζονται αναβάθμιση του υλικού τους ώστε να συμβαδίζουν με τη διδασκαλία της ρομποτικής και χειροκίνητη ενημέρωση του λογισμικού τους λόγω του ότι δεν γίνεται πάντα αυτόματα [44].
- Κρίσιμο ζήτημα στα οχήματα αυτά είναι ο σωστός εντοπισμός της θέσης τους χωρίς τη χρησιμοποίηση GPS. Σε ορισμένα οχήματα το σφάλμα είναι αρκετά μεγάλο και οφείλεται σε διάφορους παράγοντες όπως το σφάλμα των αισθητήρων οδομετρίας, η ολίσθηση των τροχών και η διαμόρφωση του εδάφους.
- Μια πρόκληση που αντιμετωπίζουν ακόμα και τα ακριβά οχήματα είναι η αδυναμία ανίχνευσης των χαμηλών εμποδίων. Αυτό οφείλεται στο σταθερό ύψος της σάρωσης των LiDAR, το οποίο είναι μεγαλύτερο από το ύψος αντικειμένων που μπορεί να βρίσκονται στο έδαφος όπως τα καλώδια του Ethernet και τα μικρά κουτιά [54].
- Μια άλλη αδυναμία των LiDAR είναι η ανακρίβεια που έχουν στις πολύ κοντινές αποστάσεις δηλαδή αποστάσεις κάτω των 20cm. Τα αποτελέσματα μιας ανακριβούς μέτρησης του LiDAR είναι η λάθος ανίχνευση σύγκρουσης και η μη δυνατότητα

εύρεσης εναλλακτικής διαδρομής, τα οποία μπορεί να προκαλέσουν το σταμάτημα του οχήματος.

- Τέλος κρίσιμο ζήτημα στην ανάπτυξη αλγορίθμων και την εκπαίδευση των ρομποτικών οχημάτων είναι η διαφορές που υπάρχουν στην απόδοση τους ανάμεσα στην προσομοίωση και στην πραγματικότητα [30]. Ένα μοντέλο το οποίο αποδίδει άριστα στην προσομοίωση δεν είναι πάντα το ίδιο αποδοτικό σε πραγματικές συνθήκες και αυτό οφείλεται σε φυσικούς παράγοντες όπως η αντανάκλαση του φωτός, η σύσταση του δαπέδου και η ολίσθηση των τροχών [31]. Επίσης ο θόρυβος των αισθητήρων που υπάρχει σε ένα φυσικό ρομποτικό όχημα μειώνει την αποτελεσματικότητα εξελιγμένων αλγορίθμων αυτόνομης οδήγησης που έχουν δοκιμαστεί σε προσομοίωση [37].

2.1.4.1 Αντιμετώπιση προκλήσεων

Για την αντιμετώπιση των παραπάνω προκλήσεων έχουν γίνει διάφορες προτάσεις και έχουν δοκιμαστεί ορισμένες λύσεις με ή χωρίς επιτυχία.

- Στο [54] προτείνεται η αναβάθμιση του επεξεργαστή σε ένα ρομποτικό όχημα, η οποία μπορεί να βελτιώσει την ακρίβεια του εντοπισμού της θέσης του και να επεκτείνει την διάρκεια βιωσιμότητας του, όμως θα αυξήσει ταυτόχρονα και το κόστος του.
- Για την μείωση του κόστους της κατασκευής των ρομποτικών οχημάτων προτείνονται λύσεις όπως η χρησιμοποίηση φθηνότερων υλικών και ο σχεδιασμός μικρότερου μεγέθους οχημάτων [29][35].
- Για να αντιμετωπιστεί ο θόρυβος των αισθητήρων και να γίνει έτσι πιο ακριβής ο εντοπισμός της θέσης του οχήματος χρησιμοποιούνται αλγόριθμοι οι οποίοι υλοποιούν ενισχυτική μάθηση [37].
- Μια λύση για να αντιμετωπιστούν τα σφάλματα της οδομετρίας και συνεπώς να βελτιωθεί η ακρίβεια του εντοπισμού της θέσης του οχήματος είναι ο συνδυασμός των δεδομένων από αισθητήρες όπως το IMU, οι κωδικοποιητές τροχών και το LiDAR. Για τον συνδυασμό των δεδομένων αυτών χρησιμοποιούνται διάφοροι αλγόριθμοι όπως είναι ο Extended Kalman Filter (EKF) [48].

- Η χρησιμοποίηση αισθητήρων υπερήχων και κάμερας για την ανίχνευση των χαμηλών εμποδίων και των πολύ κοντινών αποστάσεων, τα οποία δεν μπορεί να ανιχνεύσει το LiDAR. Οι αισθητήρες υπερήχων μπορούν να σταματήσουν ακαριαία το ρομποτικό όχημα σε περίπτωση ανίχνευσης εμποδίου ενώ η χρήση κάμερας, η οποία θα κοιτάει προς το έδαφος, μπορεί να αναγνωρίζει τα χαμηλά εμπόδια [54].

2.2. FOSSBot

Το FOSSBot είναι ένα οικονομικό ρομποτικό όχημα που μπορεί να χρησιμοποιηθεί στην πρωτοβάθμια και στην δευτεροβάθμια εκπαίδευση [14].Έχει σχεδιαστεί από την ΕΕΛΛΑΚ ακολουθώντας το πρότυπο «Ελεύθερο Λογισμικό και Λογισμικό Ανοικτού Κώδικα» (ΕΛ/ΛΑΚ) και επιτρέπει τη μελέτη, την τροποποίηση και τη βελτίωση του λογισμικού και του υλικού του, τα οποία είναι διαθέσιμα στο GitHub [15].



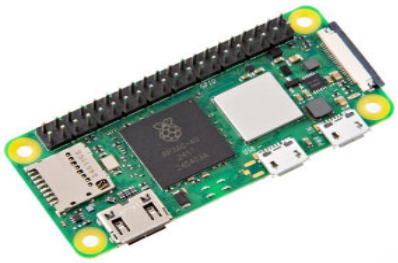
Εικόνα 2-1. FOSSBot

2.2.1 Υλικό μέρος του FOSSBot

Η κατασκευή του ακολουθεί την φιλοσοφία του Do It Yourself (DIY) και μπορεί να πραγματοποιηθεί είτε από γνώστες του αντικειμένου είτε από αρχάριους, π.χ. μαθητές, με στόχο την εκπαίδευση τους. Στην Εικόνα 2-1 φαίνονται τα πλαστικά του μέρη, τα οποία εκτός από τους τροχούς, δημιουργούνται από τρισδιάστατο εκτυπωτή με πολύ μικρό κόστος και σε περίπου 36 ώρες. Είναι σχεδιασμένο με τέτοιο τρόπο ώστε να συναρμολογείται εύκολα και να τοποθετούνται τα ηλεκτρονικά του εξαρτήματα σε συγκεκριμένα σημεία, κρατώντας τα σταθερά και εύκολα προσβάσιμα. Τα ηλεκτρονικά του εξαρτήματα είναι χαμηλού κόστους και αναφέρονται σε μια λίστα μέσα στις οδηγίες κατασκευής του στο [15]. Τα κυριότερα περιγράφονται παρακάτω.

2.2.1.1 Μικροϋπολογιστής Raspberry Pi Zero 2 W

Το Raspberry Pi Zero 2 W, το οποίο φαίνεται στην Εικόνα 2-2, είναι η τελευταία έκδοση της σειράς Zero των Raspberry Pi. Έχει χαμηλή τιμή, πολύ μικρό μέγεθος (65mm × 30mm) και είναι αρκετά αποδοτικός λόγω του τετραπύρηνου επεξεργαστή του 64-bit Arm Cortex-A53 με συχνότητα 1GHz και 512MB RAM. Προσφέρει μεγάλη ποικιλία συνδεσμολογίας και είναι ιδανικό για συσκευές Internet of Things (IOT), όπως οι ασύρματες κάμερες ασφαλείας και οι έξυπνες συσκευές ενός σπιτιού [55].



Εικόνα 2-2. Raspberry Pi Zero 2 W

Κύρια χαρακτηριστικά:

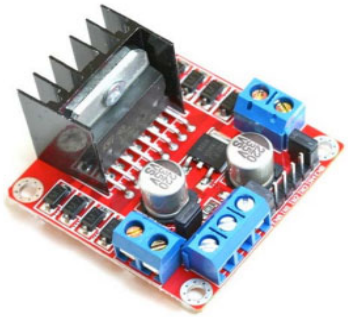
- Ασύρματη σύνδεση: 2.4GHz 802.11 b/g/n wireless LAN, Bluetooth 4.2 με υποστήριξη Low Energy (BLE).
- Θύρες: Μία USB 2.0 με λειτουργία On-The-Go (OTG), μία mini HDMI και υποδοχή κάμερας CSI-2.
- Επέκταση: Υποστηρίζει 40-ακίδα κεφαλίδα GPIO, η οποία είναι συμβατή με τα HAT αρθρώματα του Raspberry Pi και υποδοχή κάρτας microSD.
- Τάση τροφοδοσίας : 5V.

2.2.1.2 Ενεργοποιητές

2.2.1.2.1 H-Bridge L298N

Η γέφυρα L298N είναι ένας οδηγός για κινητήρες συνεχούς ρεύματος και υποστηρίζει έως δύο κινητήρες μέγιστης ισχύος 2A ανά κινητήρα. Είναι μικρή σε μέγεθος και ιδανική για την εφαρμογή πλήθους κατασκευών ενώ διαθέτει κλέμες για την σύνδεση των κινητήρων και του ρεύματος. Όπως φαίνεται στην Εικόνα 2-3 η L298N είναι εξοπλισμένη με 4 ακίδες εισόδου για τον έλεγχο των κινητήρων από τα ψηφιακά pins του μικροϋπολογιστή, τέσσερις εξόδους που

τροφοδοτούν τους κινητήρες, δύο ακίδες για τον έλεγχο της ταχύτητας των κινητήρων με τη χρήση PWM, δύο εισόδους τροφοδοσίας 12 Volt και 5 Volt και μία για τη γείωση [56].



Εικόνα 2-3. H-Bridge L298N

2.2.1.2.2 Κινητήρας DC

Ο κινητήρας DC Gear Motor TT-120RPM φαίνεται στην Εικόνα 2-4 και είναι ένας τύπος κινητήρα συνεχούς ρεύματος που διαθέτει ένα κιβώτιο ταχυτήτων. Το κιβώτιο ταχυτήτων μειώνει την ταχύτητα του κινητήρα και αυξάνει τη ροπή του. Έχει τάση λειτουργίας 3V DC και τροφοδοτείται είτε από 2 μπαταρίες AA είτε από το Raspberry Pi [57].



Εικόνα 2-4. DC Gear Motor TT-120RPM

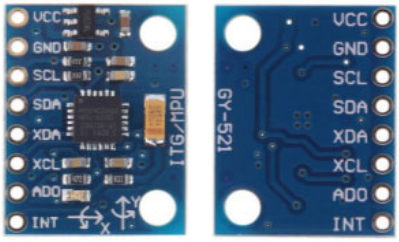
2.2.1.3 Αισθητήρες

2.2.1.3.1 Μονάδα Αδρανειακής Μέτρησης IMU - MPU6050

Η IMU κατά το [58] είναι ένας αισθητήρας που μετρά και αναφέρει την κίνηση και τον προσανατολισμό της συσκευής στο τρισδιάστατο χώρο συνδυάζοντας επιταχυνσιόμετρο, γυροσκόπιο και μερικές φορές μαγνητόμετρο. Στο σχεδιασμό του FOSSBot χρησιμοποιείται η μονάδα MPU-6050, η οποία φαίνεται στην Εικόνα 2-5 και σε ένα τσιπ συνδυάζει δύο αισθητήρες που χρησιμοποιούν έναν μετατροπέα από αναλογικό σε ψηφιακό 16-bit :

- Το γυροσκόπιο που μετράει την γωνιακή ταχύτητα δηλαδή την ταχύτητα που περιστρέφεται η μονάδα γύρω από τους άξονες X,Y,Z.

- Το επιταχυνσιόμετρο που μετράει την γραμμική επιτάχυνση δηλαδή την μεταβολή της ταχύτητας κατά μήκος των αξόνων X,Y,Z.



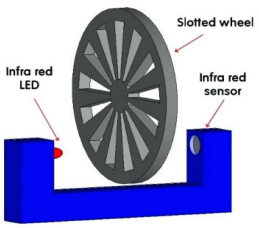
Εικόνα 2-5. MPU6050

2.2.1.3.2 Κωδικοποιητής τροχών

Ο κωδικοποιητής τροχών μετράει την ταχύτητα ενός οχήματος με την βοήθεια ενός αισθητήρα διακοπής δέσμης φωτός, ο οποίος φαίνεται στην Εικόνα 2-6 και ενός δίσκου με σχισμές (slotted wheel) ο οποίος είναι τοποθετημένος στον άξονα των τροχών του εν λόγω οχήματος. Συγκεκριμένα, όπως φαίνεται στην Εικόνα 2-7, στον αισθητήρα από την μια μεριά είναι ο πομπός (infra red LED), ο οποίος εκπέμπει υπέρυθρο φως και από την άλλη μεριά είναι ο δέκτης (infra red sensor), ο οποίος είτε ενεργοποιείται όταν υπάρχει σχισμή στον δίσκο, που βρίσκεται ανάμεσα, και δέχεται το φως είτε απενεργοποιείται όταν εμποδίζεται η δίοδος του φωτός από τον δίσκο. Με την εναλλαγή αυτή υπολογίζεται η ταχύτητα και η απόσταση που διανύει το όχημα. Για την καλύτερη ακρίβεια της μέτρησης χρησιμοποιείται ο μετατροπέας Schmidt, ο οποίος καθαρίζει το σήμα και στη συνέχεια το αποστέλλει στον μικροϋπολογιστή [59].



Εικόνα 2-6. Αισθητήρας διακοπής δέσμης φωτός



Εικόνα 2-7. Κωδικοποιητής τροχών

2.2.1.3.3 HC-SR04

Ο HC-SR04 είναι ένας υπερηχητικός αισθητήρας, ο οποίος χρησιμοποιεί σόναρ για να μετρήσει την απόσταση από ένα αντικείμενο. Παρέχει σταθερές μετρήσεις χωρίς επαφή με πολύ υψηλή ακρίβεια έως 3 χιλιοστά και σε αποστάσεις από 2cm έως 400 cm. Οι μετρήσεις αυτές δεν

επηρεάζονται από το ηλιακό φως ή το μαύρο υλικό. Τα βασικά μέρη του είναι το κύκλωμα ελέγχου, ο πομπός και ο δέκτης υπερήχων, οι οποίοι φαίνονται στην Εικόνα 2-8, και οι ακίδες τους από τις οποίες στέλνονται τα δεδομένα στον μικροϋπολογιστή [60].



Εικόνα 2-8. HC-SR04

2.2.1.3.4 Φωτοαντίσταση - KS0028

Η φωτοαντίσταση είναι αντιστάτες των οποίων η αντίσταση μεταβάλλεται αντιστρόφως ανάλογα με την ένταση του φωτός, δηλαδή όσο μεγαλύτερη είναι η φωτεινότητα τόσο μικρότερη είναι η αντίσταση [61]. Ο αισθητήρας KS0028 που φαίνεται στην Εικόνα 2-9 αποτελείται από μια φωτοαντίσταση και 3 ακροδέκτες, έναν για την τροφοδοσία, έναν για την γείωση και έναν αναλογικό, ο οποίος στέλνει την τιμή της φωτεινότητας στον μικροϋπολογιστή [62].

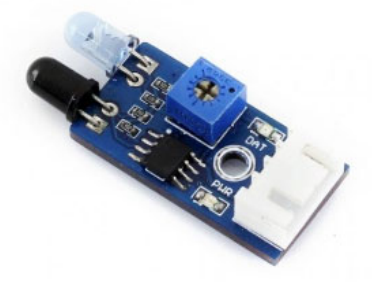


Εικόνα 2-9. Φωτοαντίσταση KS0028

2.2.1.3.5 Αισθητήρας απόστασης υπερύθρων

Ο αισθητήρας απόστασης υπερύθρων είναι ένας αισθητήρας εγγύτητας που εκπέμπει υπέρυθρο φως από τον πομπό και λαμβάνει στη συνέχεια στον δέκτη το ανακλώμενο υπέρυθρο φως ώστε να μπορεί να ενημερώσει σχετικά με την παρουσία εμποδίων μπροστά του. Αυτός πομπός και ο δέκτης διακρίνονται στην Εικόνα 2-10 με άσπρο και μαύρο χρώμα αντίστοιχα. Ο συγκεκριμένος αισθητήρας της waveshare διαθέτει συγκριτή τάσης ευρέος φάσματος LM393, ποτενσιόμετρο για ρυθμιζόμενη ευαισθησία, αναλογική έξοδο για μέτρηση απόστασης και ψηφιακή για ανίχνευση εμποδίου κάτω από ένα κατώφλι που ορίζουμε εμείς. Έχει εύρος

ανίχνευσης από 2 cm έως 30 cm, το οποίο όμως επηρεάζεται από το χρώμα του εμποδίου, λόγω της διαφοράς στην ανάκλαση του υπέρυθρου φωτός σε λευκό εμπόδιο, όπου το εύρος είναι το μέγιστο δυνατό και σε μαύρο όπου είναι κοντά στο ελάχιστο. Αυτό το χαρακτηριστικό των υπερύθρων είναι χρήσιμο για εφαρμογές ανίχνευσης γραμμής όπου χρησιμοποιείται συνήθως λευκού χρώματος πίστα στην οποία αντανακλάται πολύ το φως και μαύρου χρώματος γραμμή στην οποία αντανακλάται λίγο έως καθόλου. Τέλος επηρεάζεται η ακρίβεια του αισθητήρα από την ηλιακή ακτινοβολία και άλλες πηγές υπερύθρων [63].



Εικόνα 2-10. Αισθητήρας Απόστασης Υπερύθρων

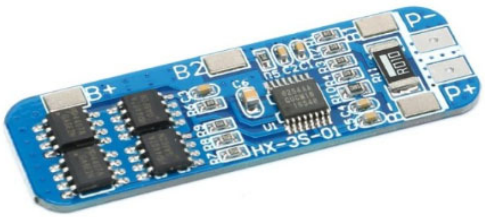
2.2.1.4 Τροφοδοσία

Το FOSSBot τροφοδοτείται από 3 επαναφορτιζόμενες μπαταρίες λιθίου-ιόντων τύπου 18650, οι οποίες έχουν τάση 3,6 V και υψηλή χωρητικότητα στα 3400mAh. Οι μπαταρίες αυτές είναι τοποθετημένες στην θήκη που φαίνεται στην Εικόνα 2-11, η οποία είναι εξοπλισμένη με δύο καλώδια 6" για τροφοδοσία και γείωση.



Εικόνα 2-11. Θήκη Μπαταριών

Η φόρτιση των μπαταριών αυτών γίνεται μέσω της μονάδας HX-3S-01, η οποία φαίνεται στην Εικόνα 2-12 και είναι μια πλακέτα που προστατεύει επίσης τις μπαταρίες από υπερφόρτιση και υπερένταση. Είναι κατάλληλη για τέτοιου τύπου μπαταρίες με 12V και 10A συνολικά [64].



Εικόνα 2-12. HX-3S-01

2.2.2 Λογισμικό

2.2.2.1 Λογισμικό Ρομποτικού Οχήματος

Το λογισμικό του ρομποτικού οχήματος FOSSBot είναι ανοιχτού κώδικα και έχει δομηθεί αρθρωτά χρησιμοποιώντας την λογική των μικρο-υπηρεσιών και την τεχνολογία Docker, η οποία προσφέρει γρήγορη εγκατάσταση και ενημερώσεις. Ο πυρήνας του λογισμικού είναι μια βιβλιοθήκη γραμμένη στην γλώσσα προγραμματισμού Python, η οποία ελέγχει την κίνηση και τους αισθητήρες του FOSSBot και μέσω αυτής μπορεί να δώσει κάποιος εντολές χωρίς να γνωρίζει τις τεχνικές λεπτομέρειες των ηλεκτρονικών. Το ρομποτικό όχημα τρέχει Raspbian, το οποίο είναι το λειτουργικό σύστημα του μικροϋπολογιστή Raspberry Pi και υποστηρίζει διάφορα εργαλεία χρήσιμα για την ρομποτική και τις εφαρμογές IOT [14]. Στην Εικόνα 2-13 φαίνεται η αρθρωτή δομή του λογισμικού του FOSSBot η οποία υποστηρίζει τους διαφόρους τρόπους λειτουργίας του και τον πλήρη έλεγχο του μέσω μιας διεπαφής διαχειριστή.



Εικόνα 2-13. Δομή λογισμικού FOSSBot

Το FOSSBot σύμφωνα με το [14] μπορεί να χρησιμοποιηθεί με τέσσερις διαφορετικούς τρόπους ανάλογα το επίπεδο του χρήστη. Ο πρώτος είναι κατάλληλος για παιδιά προσχολικής ηλικίας και χρησιμοποιεί μια διεπαφή με κουμπιά για την κίνηση του οχήματος μπροστά ή πίσω και για την εκτέλεση στροφής. Ο δεύτερος τρόπος είναι ο προγραμματισμός με την χρήση μπλοκ εντολών, ο οποίος είναι κατάλληλος για παιδιά δημοτικού και τους επιτρέπει να ελέγχουν την κίνηση, τους αισθητήρες και τις συσκευές αλληλεπίδρασης, όπως το LED και το ηχείο, του οχήματος με τον συνδυασμό των συγκεκριμένων μπλοκ. Ο τρίτος τρόπος προορίζεται για την δευτεροβάθμια εκπαίδευση και τη διδασκαλία των βασικών αρχών του προγραμματισμού Python. Σε αυτή την προσέγγιση ο μαθητής μπορεί να γράφει κώδικα σε Jupyter Notebook και να αναλύει τα δεδομένα των αισθητήρων από την οπτικοποίηση τους που γίνεται σε πραγματικό χρόνο. Τέλος, ο τέταρτος τρόπος λειτουργίας του FOSSBot είναι ο προγραμματισμός σε Python απευθείας στο κέλυφος του συστήματος του, μέσω σύνδεσης SSH. Με αυτόν τον τρόπο μπορούμε να τροποποιήσουμε την κεντρική βιβλιοθήκη του και να γράψουμε προγράμματα σε Python τα οποία θα εκτελούνται απευθείας στο ρομποτικό όχημα.

2.2.2.2 Λογισμικό Προσομοίωσης

Το λογισμικό προσομοίωσης είναι ένα πρόγραμμα που χρησιμοποιείται για την δημιουργία εικονικών κόσμων και αντικειμένων, όπως τα ρομποτικά οχήματα, ώστε να μιμηθεί την λειτουργία των αντίστοιχων πραγματικών οχημάτων. Τα λογισμικά αυτά είναι χρήσιμα ώστε ο καθένας να μπορεί να χρησιμοποιήσει και να δοκιμάσει το λογισμικό ενός ρομποτικού οχήματος και να πειραματιστεί με αυτό χωρίς κόστος [65].

Για το FOSSBot αναπτύχθηκε καταρχήν ένας προσομοιωτής, ο οποίος αποτελεί την βασική προσομοίωση και τρέχει απευθείας σε πρόγραμμα περιήγησης ιστού. Ο κώδικας του βρίσκεται στο βασικό αποθετήριο του FOSSBot στο GitHub [15] και δίνει τη δυνατότητα να προγραμματίσει κάποιος με τα ίδια εργαλεία και γλώσσα που χρησιμοποιούνται και για το πραγματικό ρομπότ. Αυτός ο προσομοιωτής είναι πολύ καλός για εκπαίδευση όμως δεν παρέχει επικοινωνία με το περιβάλλον λογισμικού ROS και δεν έχει αρκετά ρεαλιστική προσομοίωση με λειτουργίες όπως οι συγκρούσεις και η βαρύτητα. Για το λόγο αυτό αναπτύχθηκε στη συνέχεια ένα μοντέλο προσομοίωσης του FOSSBot για το λογισμικό CoppeliaSim (πρώην V-REP) [66], το οποίο υπάρχει στο GitHub στο [67]. Το μοντέλο αυτό προσφέρει έτοιμο σκηνικό που περιέχει το εικονικό FOSSBot με όλους τους αισθητήρες και τους κινητήρες του και την δυνατότητα να το ελέγξει κάποιος με ROS όπως με το πραγματικό όχημα. Το CoppeliaSim είναι κατάλληλο για

την εκπαίδευση και παρέχει μεταξύ άλλων σχεδιασμό κίνησης, έλεγχο συγκρούσεων, λειτουργίες κινηματικής αντίστροφης, μοντελοποίηση των αισθητήρων και δυνατότητα δέσμης ενεργειών. Τέλος υπάρχουν πολλές εκδόσεις του συγκεκριμένου λογισμικού από τις οποίες η έκδοση Edu είναι δωρεάν [65].

2.2.3 Ελλείψεις FOSSBot

2.2.3.1 Υλικού

Το FOSSBot υποστηρίζει έναν μεγάλο αριθμό από αισθητήρες με τους οποίους μπορούν να εκτελεστούν διάφορα εκπαιδευτικά σενάρια [14]. Οι βασικοί αισθητήρες που υπάρχουν στον σχεδιασμό του παρουσιάστηκαν στο κεφάλαιο 2.2.1.3 και σε αυτούς πρέπει να προστεθούν δύο εξελιγμένοι αισθητήρες που είναι απαραίτητοι για την εκτέλεση προχωρημένων εφαρμογών όπως οι αλγόριθμοι SLAM. Οι αισθητήρες αυτοί, οι οποίοι θα παρουσιαστούν στο κεφάλαιο 2.3, είναι η κάμερα και το LiDAR και χρησιμοποιούνται σε εξελιγμένα ρομποτικά οχήματα για την χαρτογράφηση του χώρου, τον εντοπισμό της θέσης τους και την πλοήγηση τους σε αυτόν.

2.2.3.2 Προσομοίωσης

Αν και το ρομποτικό όχημα FOSSBot συνοδεύεται από δύο διαφορετικά λογισμικά προσομοίωσης υπάρχουν μειονεκτήματα σε αυτά τα οποία αναδεικνύουν την ανάγκη για ένα πιο ολοκληρωμένο μοντέλο προσομοίωσης σε ισχυρότερο λογισμικό. Όπως παρουσιάστηκε προηγουμένως το βασικό λογισμικό προσομοίωσης του FOSSBot είναι κατάλληλο μόνο για την δοκιμή απλών λειτουργιών του, όπως ο έλεγχος της κίνησης και των βασικών αισθητήρων του. Ταυτόχρονα το μοντέλο προσομοίωσης στο λογισμικό CoppeliaSim προσφέρει μεγαλύτερη φυσική προσομοίωση και με περισσότερες λειτουργίες, όμως το λογισμικό αυτό είναι κλειστού κώδικα και δεν επιτρέπει την εύκολη μεταφορά του επαληθευμένου κώδικα από την προσομοίωση στο πραγματικό ρομπότ. Επίσης δεν υπάρχει μια βιβλιοθήκη εικονικών κόσμων όπως ένα γραφείο ή μια πίστα με εμπόδια ώστε να μπορεί να εκτελεστεί μια γρήγορη δοκιμή προηγμένων αλγορίθμων, όπως οι αλγόριθμοι χαρτογράφησης [65]. Για να καλυφθεί αυτό το κενό θα χρησιμοποιηθεί το λογισμικό Gazebo το οποίο είναι ιδανικό για πλήρη προσομοίωση ρομποτικής φυσικής και Αυτόνομα Κινητά Ρομπότ (AMR). Το Gazebo είναι ένας από τους πιο ικανούς προσομοιωτές ρομποτικής ανοιχτού κώδικα, ο οποίος προσφέρει περιβάλλοντα πολλαπλών ρομπότ, προηγμένους αισθητήρες όπως το LiDAR, πλοήγηση οχήματος αυτόματης

οδήγησης (AGV) και ενσωμάτωση σε ROS2 [65]. Τα χαρακτηριστικά και οι λειτουργίες του θα παρουσιαστούν λεπτομερώς στην παράγραφο 2.4.2.2.

2.3. Πρόσθετοι Αισθητήρες

2.3.1 Ανίχνευση και Μέτρηση Απόστασης μέσω Φωτός – LiDAR

Η τεχνολογία LiDAR (Light Detection and Ranging) χρησιμοποιείται για την συλλογή τρισδιάστατων δεδομένων που αφορούν την διαμόρφωση του χώρου. Ο υπολογισμός της απόστασης των αντικειμένων από την συσκευή είναι ακριβής και επιτρέπει την δημιουργία ψηφιακών μοντέλων του περιβάλλοντος. Οι μετρήσεις του LiDAR πραγματοποιούνται με την εκπομπή παλμού λέιζερ και στη συνέχεια, λαμβάνοντας υπόψη την κατεύθυνση εκπομπής του, μετριέται ο χρόνος επιστροφής του (time of flight- TOF) έχοντας ταξιδέψει από τον αισθητήρα στο αντικείμενο και πίσω, λαμβάνοντας υπόψη και την κατεύθυνση εκπομπής [68].

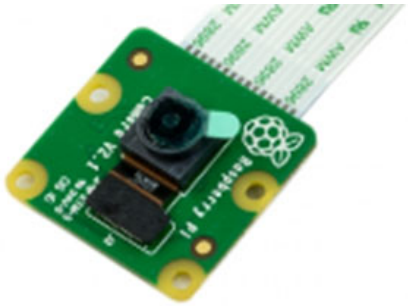
Ο αισθητήρας LiDAR της SLAMTEC που προτείνεται στην παρούσα εργασία για να τοποθετηθεί στο FOSSBot είναι ο δισδιάστατος αισθητήρας λέιζερ RPLIDAR C1 και φαίνεται στην Εικόνα 2-14. Ο συγκεκριμένος αισθητήρας προσφέρει κάλυψη 360 μοιρών, πραγματοποιεί 5.000 δειγματοληψίες το δευτερόλεπτο και λόγω της τεχνολογίας TOF που χρησιμοποιεί, διενεργεί ακριβείς μετρήσεις σε αποστάσεις έως και 12 μέτρα. Μεγάλο πλεονέκτημα του είναι η χαμηλή τυφλή ζώνη που διαθέτει, η οποία έχει τιμή 5 εκατοστά και επιτρέπει την ανίχνευση αντικειμένων σε πολύ κοντινή απόσταση. Ο σχεδιασμός του είναι συμπαγής με χαμηλά επίπεδα θορύβου, καθιστώντας τον ιδανικό για αυτόνομα οχήματα και συστήματα χαρτογράφησης εσωτερικών χώρων. Για την τροφοδοσία του χρειάζεται συνεχές ρεύμα 5 Volt, το οποίο λαμβάνει από τον ακροδέκτη VCC που υπάρχει στο 4-pin καλώδιο UART μαζί με τους ακροδέκτες GND (γείωση), TX (έξοδο) και RX (είσοδο). Τέλος κατά τη λειτουργία του καταναλώνει περίπου 230mA και χρειάζεται 4 βίδες τύπου M2.5 οι οποίες μπορούν να εισχωρήσουν στο εσωτερικό του FOSSBot μέχρι 4 χιλιοστά χωρίς να επηρεαστούν τα εσωτερικά συστατικά του αισθητήρα [69]. Ο συγκεκριμένος αισθητήρας επιλέχθηκε λόγω του συνδυασμού της χαμηλής τιμής του, σχετικά με άλλους παρόμοιους αισθητήρες, και της καλής απόδοσης του στις μετρήσεις.



Εικόνα 2-14. SLAMTEC RPLIDAR C1

2.3.2 Κάμερα

Η κάμερα είναι ένα όργανο που χρησιμοποιείται για τη λήψη και αποθήκευση εικόνων και βίντεο. Στα ρομποτικά οχήματα η κάμερα είναι χρήσιμη για εφαρμογές χαρτογράφησης, πλοήγησης και ανίχνευσης εμποδίων με την χρήση τεχνητής νοημοσύνης και μηχανικής μάθησης. Στο FOSSBot προτείνεται η χρήση της κάμερας Raspberry Pi Camera Module V2, η οποία είναι κατασκευασμένη από την Raspberry Pi Foundation και φαίνεται στην Εικόνα 2-15. Η συγκεκριμένη κάμερα διαθέτει τον αισθητήρα εικόνας Sony IMX219, ο οποίος έχει ανάλυση 8 megapixel και έναν φακό κάμερας σταθερής εστίασης. Μπορεί να κάνει λήψη φωτογραφιών ανάλυσης 3280 x 2464 pixel και υποστηρίζει βίντεο 1080p30, 720p60 και 640x480p90. Η σύνδεση της με το Raspberry γίνεται στην σειριακή διεπαφή κάμερας CSI, που διαθέτει ο μικροϋπολογιστής, με ένα καλώδιο Ribbon 15 ακίδων μήκους 150 mm [70].



Εικόνα 2-15. Raspberry Pi Camera Module V2

2.4. Λειτουργικό Σύστημα Ρομπότ 2η έκδοση - ROS2

Το Λειτουργικό Σύστημα Ρομπότ (ROS) είναι ένα λογισμικό ανοικτού κώδικα το οποίο χρησιμοποιείται για την ανάπτυξη ρομποτικών συστημάτων και περιέχει εργαλεία και βιβλιοθήκες που προσφέρουν τις απαραίτητες υπηρεσίες για τα συστήματα αυτά. Ορισμένες υπηρεσίες είναι η αφαίρεση υλικού, ο έλεγχος χαμηλού επίπεδου συσκευών, η διαχείριση πακέτων και η επικοινωνία μεταξύ διαφορετικών διεργασιών με τη χρήση μηνυμάτων. Οι διεργασίες αυτές παρουσιάζονται ως κόμβοι και τα μηνύματα ανταλλαγής είναι δεδομένα αισθητήρων, εντολές ελέγχου και πληροφορίες σχεδιασμού [71]. Το ROS χωρίζεται σε τρία μέρη, τα εργαλεία δόμησης και διανομής, τις βιβλιοθήκες του πελάτη όπως η Python και τα πακέτα που περιέχουν αλγόριθμους για χαρτογράφηση, πλοήγηση και αντίληψη του οχήματος. Το ROS διαδραμάτισε σημαντικό ρόλο πάνω στην έρευνα για την ρομποτική καθώς είχε σχεδιαστεί ως ερευνητική πλατφόρμα. Όμως αντιμετωπίζει πολλές προκλήσεις και περιορισμούς πάνω στην ασφάλεια, την αξιοπιστία του δικτύου και την υποστήριξη συστημάτων πραγματικού χρόνου. Με σκοπό να αντιμετωπιστούν οι προκλήσεις αυτές και ταυτόχρονα να είναι το ROS κατάλληλο για βιομηχανική παραγωγή, αποφασίστηκε η επόμενη έκδοση του, το ROS2, να σχεδιαστεί από την αρχή [17].

2.4.1 Αρχιτεκτονική του ROS2

Η αρχιτεκτονική του ROS2 προέκυψε με επανασχεδίαση του αρχικού λειτουργικού συστήματος και σε αντίθεση με το ROS 1, η δεύτερη έκδοση έχει σχεδιαστεί να είναι αρθρωτή και κλιμακωτή. Τα χαρακτηριστικά αυτής της αρχιτεκτονικής επιτρέπουν στο ROS2 να είναι κατάλληλο για πολλές εφαρμογές από την έρευνα έως την εμπορική παραγωγή. Τα κυριότερα από αυτά σύμφωνα με το [72] παρουσιάζονται παρακάτω.

2.4.1.1 Υπηρεσία Διανομής Δεδομένων - DDS

Το σημαντικότερο χαρακτηριστικό του ROS2 είναι το ανοιχτό πρότυπο Data Distribution Service (DDS), το οποίο χρησιμοποιείται ευρέως για την επικοινωνία σε κρίσιμα συστήματα όπως τα στρατιωτικά και τα οικονομικά καθώς επιτρέπει την ασφαλή ανταλλαγή δεδομένων σε πραγματικό χρόνο. Το DDS χρησιμοποιεί ομότιμη επικοινωνία και έτσι καταργείται ο κεντρικός διακομιστής που υπήρχε στο ROS και προκαλούσε συχνά αποτυχία στο σύστημα. Επίσης το πρωτόκολλο αυτό μεταβάλει τον τρόπο που κυκλοφορούν τα δεδομένα μέσα στο σύστημα προσαρμόζοντας τους περιορισμού του δικτύου. Τέλος προσφέρει αναβαθμισμένη ασφάλεια με

την βοήθεια της ενσωματωμένης κρυπτογράφησης και με τον έλεγχο της ταυτότητας και της πρόσβασης.

2.4.1.2 Επίπεδα Αφαίρεσης

Το ROS2 οργανώνεται σε επίπεδα και υπάρχει έτσι η δυνατότητα να αλλάζουν τα εξαρτήματα του συστήματος χωρίς να χρειάζεται να αλλάξει ο κώδικας της εφαρμογής. Τα επίπεδα αυτά κρύβουν τις περίπλοκες τεχνικές λεπτομέρειες της λειτουργίας του ROS2 και οι χρήστες αλληλεπιδρούν με τις βιβλιοθήκες πελάτες, όπως είναι η ROS Client Library (rcl). Η συγκεκριμένη βιβλιοθήκη είναι γραμμένη σε C και προσφέρει τις βασικές λειτουργίες των υπολοίπων βιβλιοθηκών πελάτη ώστε να έχουν όλες όμοια συμπεριφορά. Άλλες σημαντικές βιβλιοθήκες είναι η rclcpp για την C++ και η rclpy για την Python, οι οποίες χρησιμοποιούνται για την πρόσβαση στις διεπαφές προγραμματισμού της επικοινωνίας. Η βιβλιοθήκη ROS MiddleWare (rmw) είναι επίσης σημαντική καθώς παρέχει τις απαραίτητες διεπαφές επικοινωνίας και την δυνατότητα να μπορούν οι διαφορετικές υλοποιήσεις DDS να είναι εναλλάξιμες.

2.4.1.3 Κόμβοι

Η επεξεργασία στο ROS2 γίνεται σε κόμβους οι οποίοι είναι τα τελικά σημεία σε ένα γράφο. Μεταξύ των κόμβων χρησιμοποιείται ασύγχρονη επικοινωνία με topics και με τις λειτουργίες δημοσίευσης και εγγραφής (publish-subscribe). Επίσης εκτός από την ασύγχρονη διενεργείται και επικοινωνία με ερωτήσεις και αποκρίσεις (request-response), η οποία ονομάζεται υπηρεσίες (services) και χρησιμοποιείται για παράδειγμα όταν πρέπει να γίνει ενημέρωση αν μια εργασία ολοκληρώθηκε. Μια άλλη μορφή επικοινωνίας είναι η ενέργεια (action), η οποία χρησιμοποιείται σε εργασίες μεγάλης διάρκειας, όπως η αυτόνομη πλοήγηση, προσφέροντας ανατροφοδότηση και δυνατότητα ακύρωσης της εργασίας. Στο ROS2 υπάρχει η επίσης η δυνατότητα να αλλάζει η κατάσταση των κόμβων, μέσω μιας μηχανής καταστάσεων, σε καταστάσεις όπως μη διαμορφωμένο, ανενεργό, ενεργό και οριστικοποιημένο. Το χαρακτηριστικό αυτό είναι σημαντικό για τον συντονισμό των κατανεμημένων συστημάτων. Τέλος το ROS2 επιτρέπει να εκτελούνται πολλοί κόμβοι σε μία διεργασία και να διενεργείται επικοινωνία μεταξύ τους παρακάμπτοντας το δίκτυο και χρησιμοποιώντας pointers για την ανταλλαγή δεδομένων. Η λειτουργία αυτή μειώνει την καθυστέρηση και τη χρησιμοποίηση της CPU ειδικά σε μεγάλα μηνύματα όπως οι εικόνες.

2.4.1.4 Υποστήριξη Διαφορετικών Περιβαλλόντων

Αντίθετα με το ROS το οποίο υποστήριζε μόνο Linux, το ROS2 υποστηρίζει και άλλα λειτουργικά συστήματα όπως τα windows και τα macOS. Επίσης υπάρχει η δυνατότητα να χρησιμοποιηθεί το ROS2 σε ενσωματωμένα συστήματα όπως οι μικροελεγκτές με την έκδοση micro-ROS και να γίνει πιο εύκολη η επικοινωνία των αισθητήρων χαμηλού επιπέδου με τους κεντρικούς επεξεργαστές,

2.4.2 Εργαλεία και πακέτα του ROS2

Το λογισμικό ROS2 περιέχει διάφορα εργαλεία είτε στην γραμμή εντολών είτε σε γραφικές διεπαφές για τον έλεγχο και την διαμόρφωση των ρομποτικών συστημάτων. Επίσης είναι οργανωμένο σε πακέτα, τα οποία μπορεί να περιέχουν κόμβους, βιβλιοθήκες, δεδομένα ή αρχεία ρυθμίσεων.

2.4.2.1 Προσομοίωση - Gazebo

Το Gazebo είναι προσομοιωτής ρομποτικής τριών διαστάσεων ανοιχτού κώδικα και αποτελεί το κύριο εργαλείο προσομοίωσης στο ROS2. Σε αυτό είναι δυνατή η δοκιμή αλγορίθμων και ο έλεγχος λειτουργίας ρομποτικών συστημάτων σε εικονικά περιβάλλοντα πριν την υλοποίηση σε πραγματικά ρομπότ μειώνοντας έτσι το κόστος της ανάπτυξης και της δοκιμής των συστημάτων αυτών. Ο συγκεκριμένος προσομοιωτής χρησιμοποιείται στην έρευνα, στην Πανεπιστημιακή εκπαίδευση και στην βιομηχανία, σε μεγάλο αριθμό εφαρμογών. Για παράδειγμα η NASA το χρησιμοποιεί για την αποστολή VIPER προσομοιώνοντας τις συνθήκες φωτισμού και βαρύτητας της Σελήνης. Επίσης η Auterion πραγματοποίησε 22.000 ώρες πτήσης σε προσομοίωση το 2021, αντικαθιστώντας την εργασία 12 μηχανικών πλήρους απασχόλησης σε ζωντανές δοκιμές [72]. Η πιο πρόσφατη γενιά του Gazebo είναι η Ignition Gazebo, η οποία γράφτηκε από την αρχή και προσφέρει βελτιωμένη αρχιτεκτονική, καλύτερη απόδοση και ευκολότερη ενσωμάτωση με το ROS 2 [11].

Ο συγκεκριμένος προσομοιωτής σύμφωνα με το [11] έχει τα παρακάτω χαρακτηριστικά που τον καθιστούν ένα ισχυρό και απαραίτητο εργαλείο για την ρομποτική:

- Καταρχήν παρέχει πλούσια και λεπτομερή γραφική απεικόνιση, δίνοντας την δυνατότητα δημιουργίας ή εισαγωγής αναλυτικών περιβαλλόντων, όπως αποθήκες, γραφεία και πόλεις. Επίσης υποστηρίζει ρεαλιστικές αναπαραστάσεις όπως οι σκιές, ο φωτισμός, η ομίχλη και η αλλαγή του καιρού.

- Η αλληλεπίδραση των αντικειμένων με το περιβάλλον προσομοιώνεται ρεαλιστικά με την παρουσία χαρακτηριστικών όπως η τριβή, η βαρύτητα και η επαφή και με τη χρησιμοποίηση μηχανών φυσικής όπως το ODE και το SimBody.
- Το Gazebo περιέχει πολλά έτοιμα μοντέλα ρομπότ όπως είναι το Tugbot και το Turtlebot και διάφορα περιβάλλοντα όπως η αποθήκη του Tugbot και η πίστα με τα εμπόδια του Turtlebot, ταυτόχρονα δίνεται η δυνατότητα στον χρήστη να εισαγάγει δικά του μοντέλα σε μορφή αρχείων .sdf (Simulation Description Format) ή .urdf (Unified Robot Description Format).
- Ο έλεγχος των ρομπότ στο εικονικό περιβάλλον γίνεται μέσω κώδικα, ο οποίος εισάγεται στις διεπαφές προγραμματισμού που παρέχει το Gazebo για διάφορες γλώσσες όπως η Python και η C++.
- Τέλος το Gazebo μπορεί να υποστηρίξει μεγάλη ποικιλία αισθητήρων όπως το LiDAR, οι κάμερες και το IMU, οι οποίοι εξάγουν δεδομένα παρόμοια με τα πραγματικά και βοηθούν στην δοκιμή διαφόρων αλγορίθμων.

2.4.2.2 Μορφή Περιγραφής Προσομοίωσης – SDF / URDF

Η Μορφή Περιγραφής Προσομοίωσης (Simulation Description Format - SDF) είναι μια μορφή XML, η οποία περιγράφει αντικείμενα, αισθητήρες και περιβάλλοντα ώστε να χρησιμοποιηθούν σε εφαρμογές προσομοίωσης, οπτικοποίησης και έλεγχου ρομποτικών συστημάτων. Υπάρχει και η Ενοποιημένη Μορφή Περιγραφής Ρομπότ (Unified Robot Description Format – URDF) η οποία χρησιμοποιείται για την περιγραφή της κινηματικής δομής του ρομπότ ενώ το SDF προσφέρει περισσότερες δυνατότητες. Η SDF δημιουργήθηκε ως μέρος του Gazebo και έγινε μια ευρέως χρησιμοποιούμενη και επεκτάσιμη μορφή που περιγράφει στατικά και δυναμικά αντικείμενα, φωτισμό, έδαφος, ακόμη και φυσική [50]. Τα βασικά στοιχεία ενός ρομπότ που περιέχει ένα αρχείο SDF σύμφωνα με το [50] είναι:

- Οι αισθητήρες του, όπως το LiDAR, κάμερες και η IMU, όπου περιλαμβάνει τις παραμέτρους λειτουργίας τους και τις εντολές ώστε να στέλνουν μηνύματα συμβατά με το ROS.
- Οι ενεργοποιητές, όπως είναι οι ελεγκτές ροπής και οι μηχανισμοί διεύθυνσης στους οποίους λαμβάνονται υπόψη περιορισμοί όπως οι μέγιστες στροφές.

- Η θέση όλων των εξαρτημάτων σε σχέση με τον βασικό άξονα του ρομπότ που ονομάζεται base_link και των αρθρώσεων (joints) που τα συνδέουν.
- Τα οπτικά και τα γεωμετρικά χαρακτηριστικά με τα οποία καθορίζεται το σχήμα, το μέγεθος και το βάρος του πλαισίου του ρομπότ.

2.4.2.3 Τρισδιάστατα μοντέλα – DAE / STL

Συχνά σύμφωνα με το [50] για την απεικόνιση των γραφικών ενσωματώνονται στο αρχείο SDF δύο τύποι αρχείων 3D μοντέλων:

- Τα Standard Triangle Language (STL) τα οποία περιγράφουν τη γεωμετρία της επιφάνειας ενός τρισδιάστατου αντικειμένου μέσω τριγωνικών πλεγμάτων (meshes). Η ενσωμάτωση τους στο SDF επιτρέπει στο Gazebo να αναγνωρίσει το ρομπότ ως ένα φυσικό αντικείμενο με το οποίο μπορεί να αλληλεπιδράσει λαμβάνοντας υπόψη παραμέτρους όπως η τριβή και οι συγκρούσεις.
- Τα Digital Asset Exchange (DAE) τα οποία είναι μια πιο σύνθετη αναπαράσταση σε σχέση με το STL καθώς περιλαμβάνουν εκτός τη γεωμετρία και δεδομένα για τις υφές (textures), τα χρώματα και τον φωτισμό.

2.4.2.4 Βιβλιοθήκη μετασχηματισμών - TF2

Η TF2 είναι από τα βασικά εργαλεία του ROS2 και είναι απαραίτητη για την παρακολούθηση πολλαπλών συστημάτων συντεταγμένων τα οποία αλλάζουν με τον χρόνο [50]. Η βιβλιοθήκη αυτή στηρίζεται στην διατήρηση των σχέσεων μεταξύ των διαφόρων εξαρτημάτων του ρομπότ, των αισθητήρων του και του εξωτερικού περιβάλλοντος σε μια δενδρική δομή γνωστή ως tf-tree, στην οποία υπάρχει ιεραρχική οργάνωση των επιπέδων [46]. Τα κύρια επίπεδα σύμφωνα με το [48] είναι το map, το odom και το base_link. Το map είναι στατικό πλαίσιο αναφοράς και δημιουργείται κατά την εκτέλεση ενός αλγορίθμου χαρτογράφησης. Δείχνει τον χώρο μέσα από τα δεδομένα του αλγορίθμου και μέσα σε αυτό κινείται το ρομπότ. Το odom δείχνει, βάσει των υπολογισμών της οδομετρίας η οποία παρουσιάζεται παρακάτω, την θέση και τον προσανατολισμό του ρομπότ σε σχέση με την αρχική του θέση. Στον μετασχηματισμό αυτό η αρχική θέση είναι το πλαίσιο odom και η τρέχουσα το πλαίσιο base_link, το οποίο βρίσκεται στο κέντρο του ρομπότ και δείχνει την σωστή θέση και κίνηση του στο χώρο. Άλλα επίπεδα του tf-tree είναι οι αισθητήρες και τα εξαρτήματα τα οποία συνδέονται με σταθερούς ή δυναμικούς μετασχηματισμούς στο base_link.

Τα επίπεδα αυτά περιέχουν πλαίσια όπως είναι το scan για το LiDAR, το imu για την μονάδα αδρανειακής μέτρησης και για τους τροχούς τα right_wheel και left_wheel.

2.4.2.5 Οπτικοποίηση - RViz2

Το RViz2 χρησιμοποιείται για την τρισδιάστατη απεικόνιση των ρομποτικών συστημάτων και των δεδομένων των αισθητήρων τους σε ένα περιβάλλον. Τα δεδομένα αυτά τα λαμβάνει μέσω topics και δίνει την δυνατότητα να βλέπει ο χρήστης το ρομπότ και το περιβάλλον του σε πραγματικό χρόνο ώστε να μπορεί να διορθώσει τυχόν λάθη στους αισθητήρες. Επίσης προσφέρει την επιλογή να καταγράφει τα δεδομένα των αισθητήρων του σε νέφη σημείων ή μετασχηματισμούς συντεταγμένων [72]. Το RViz2 προσφέρει την δυνατότητα της οπτικής αναπαράστασης του δέντρου μετασχηματισμών (tf-tree) και της αυτόνομης πλοήγησης μέσω του καθορισμού της αρχικής θέσης του ρομπότ και του στόχου πλοήγησης πάνω στο χάρτη [48].

2.4.3 Οδομετρία

Η οδομετρία είναι η χρησιμοποίηση από ένα αυτόνομο σύστημα των δεδομένων που του παρέχουν ενσωματωμένοι αισθητήρες για τον υπολογισμό της μεταβολής της θέσης και του προσανατολισμού του κατά τη διάρκεια του χρόνου [73]. Στο ROS2 η οδομετρία αποτελεί τη γέφυρα μεταξύ του στατικού χάρτη (map) και του κινητού πλαισίου αναφοράς του ρομπότ (base_link). Υπάρχουν διάφοροι τύποι οδομετρίας οι οποίοι βασίζονται σε διαφορετικούς αισθητήρες και οι πέντε κυριότεροι σύμφωνα με το [73] είναι:

1. Η οπτική οδομετρία στην οποία χρησιμοποιούνται κάμερες για να γίνει η ανάλυση μιας ροής εικόνων και να υπολογιστεί η μετατόπιση των εικονοστοιχείων μεταξύ των πλαισίων των εικόνων με την πάροδο του χρόνου. Είναι πολύ ακριβής με πολύ χαμηλό σχετικό σφάλμα όμως χρειάζεται μεγάλη υπολογιστική ισχύ για την ανάλυση των εικόνων και επηρεάζεται από τον φωτισμό.
2. Η αδρανειακή οδομετρία η οποία βασίζεται στην IMU και χρησιμοποιεί το επιταχυνσιόμετρο και το γυροσκόπιο για την μέτρηση της ταχύτητας και τον υπολογισμό της θέσης και του προσανατολισμού του οχήματος. Λόγω του μικρού μεγέθους και της χαμηλής κατανάλωσης ενέργειας χρησιμοποιείται σε συστήματα με μικρή αυτονομία, όπως είναι τα drones, όμως αν και έχει μικρά

σφάλματα με την πάροδο του χρόνου συσσωρεύονται και δημιουργούνται μεγάλες αποκλίσεις στην θέση και τον προσανατολισμό του ρομπότ.

3. Η οδομετρία LiDAR στην οποία οι ακτίνες laser που αντανακλώνται από το περιβάλλον δημιουργούν μοτίβα και μέσω της παρακολούθησης των μοτίβων αυτών προσδιορίζεται η κίνηση. Οι αισθητήρες LiDAR δεν επηρεάζονται από τον φωτισμό και την υφή των επιφανειών όμως η διαδικασία υπολογισμού της κίνησης μέσω διαδοχικών σαρώσεων απαιτεί μεγάλη υπολογιστική ισχύ. Η μέθοδος αυτή δεν επηρεάζεται ούτε από τις καιρικές συνθήκες ούτε από τον φωτισμό και έχει μεγάλη εμβέλεια, όμως δεν έχει καλή ανάλυση ώστε να αναγνωρίζονται σωστά τα αντικείμενα.
4. Η οδομετρία με Radar η οποία για τον υπολογισμό της κίνησης του οχήματος χρησιμοποιεί ραδιοκύματα μετρώντας την ταχύτητα, την απόσταση και την γωνία των γύρω αντικειμένων.
5. Η οδομετρία των τροχών στην οποία υπολογίζεται η θέση και η ταχύτητα του οχήματος με βάση τον αριθμό των περιστροφών των τροχών με τους οποίους κινείται. Για την μέτρηση των περιστροφών αυτών χρησιμοποιούνται κωδικοποιητές οι οποίοι είναι τοποθετημένοι στους τροχούς και στην συνέχεια οι περιστροφές αντιστοιχίζονται σε γραμμική μετατόπιση. Η μέθοδος αυτή είναι η πιο απλή και χρησιμοποιείται συχνά σε επίγεια ρομποτικά οχήματα, όπως τα Mars Exploration Rovers της NASA, όμως οι υπολογισμοί της για την θέση του οχήματος παρουσιάζουν συχνά αποκλίσεις λόγω της ολίσθησης των τροχών στο έδαφος.

Όλες οι μέθοδοι οδομετρίας έχουν τα πλεονεκτήματα τους και τα μειονεκτήματα τους, τα οποία τις καθιστούν ιδανικές για διαφορετικές εφαρμογές την κάθε μία. Το κοινό πρόβλημα που αντιμετωπίζουν όμως είναι η ολίσθηση καθώς τα σφάλματα των μετρήσεων, όσο μικρά και αν είναι, συσσωρεύονται δημιουργώντας μεγάλες αποκλίσεις στον προσδιορισμό της θέσης του συστήματος. Τέλος η οδομετρία αν και εστιάζει στην τοπική κίνηση θεωρείται ως η πρώτη φάση του SLAM, καθώς οι αλγόριθμοι SLAM την χρησιμοποιούν για να δημιουργήσουν τον χάρτη με βάση το ιστορικό της στάσης του οχήματος. Στην συνέχεια μειώνουν την απόκλιση του υπολογισμού της τρέχουσας στάσης με την βοήθεια του χάρτη [73].

2.4.4 Σύντηξη Αισθητήρων

Για την επίλυση του φαινομένου της ολίσθησης προτείνεται στο [73] ο συνδυασμός δεδομένων από διαφορετικούς αισθητήρες ο οποίος γίνεται μέσω μιας διαδικασίας, γνωστής ως σύντηξη αισθητήρων, που μπορεί να βοηθήσει στην αύξηση της ακρίβειας των υπολογισμών σε ένα σύστημα αυτόνομης πλοήγησης. Για να αντιμετωπιστούν οι περιορισμοί των αισθητήρων μπορούν να γίνουν πολλοί διαφορετικοί συνδυασμοί, όπως η οπτικο-λέιζερ οδομετρία στην οποία συνδυάζονται η κάμερα και το LiDAR μειώνοντας την απόκλιση σε περιβάλλοντα με χαμηλή υφή και η ραντάρ-αδρανειακή οδομετρία η οποία χρησιμοποιείται για τον προσδιορισμό της ταχύτητας σε ολισθηρές επιφάνειες.

Ο συνδυασμός των δεδομένων μπορεί να γίνει είτε με χαλαρή σύνδεση είτε με ισχυρή σύνδεση. Στην χαλαρή σύνδεση τα δεδομένα από τους διαφορετικούς αισθητήρες συνδυάζονται στο τελικό στάδιο προσφέροντας χαμηλή υπολογιστική πολυπλοκότητα με μικρή όμως ακρίβεια, ενώ στην ισχυρή σύνδεση οι μετρήσεις των αισθητήρων συνδυάζονται σε μια κοινή συνάρτηση ή φίλτρο, προσφέροντας πολύ μεγάλη ακρίβεια με μεγάλες όμως απαιτήσεις σε υπολογιστική ισχύ. Υπάρχει επίσης μια ενδιάμεση λύση που προσπαθεί να ισορροπήσει την ακρίβεια με την υπολογιστική πολυπλοκότητα και ονομάζεται ημι-ισχυρή σύνδεση [73].

Τέλος υπάρχουν δυο βασικές προσεγγίσεις για την υλοποίηση της σύντηξης:

- Η υλοποίηση που βασίζεται στην κοινή βελτιστοποίηση οπτικών και αδρανειακών μετρήσεων και προσφέρει μεγάλη ακρίβεια. Η οποία όμως καταναλώνει πολλούς υπολογιστικούς πόρους αν και γίνεται προσπάθεια να μειωθεί ο φόρτος με τεχνικές όπως τα keyframes και τα sliding windows.
- Η υλοποίηση που βασίζεται σε φίλτρα και χρησιμοποιεί αλγόριθμους όπως το Extended Kalman Filter (EKF), το Multi-State Constraint Kalman Filter (MSCKF) και το Unscented Kalman Filter (UKF). Τα φίλτρα δεν καταναλώνουν πολλούς πόρους όμως μπορεί να παρουσιάσουν σφάλματα λόγω της γραμμικοποίησης [73].

2.4.5 Ταυτόχρονος Εντοπισμός και Χαρτογράφηση– SLAM

Η τεχνολογία SLAM επιτρέπει σε ένα αυτόνομο ρομποτικό όχημα να δημιουργεί μόνο του έναν χάρτη του άγνωστου περιβάλλοντος στο οποίο κινείται και ταυτόχρονα να προσδιορίζει την θέση και τον προσανατολισμό του μέσα σε αυτόν. Οι διαδικασίες της χαρτογράφησης και

του εντοπισμού δεν μπορούν να διαχωριστούν καθώς για δημιουργηθεί ένας καλός χάρτης πρέπει το ρομπότ να ξέρει την θέση του και για να γνωρίσει με ακρίβεια την θέση του πρέπει να έχει έναν καλό χάρτη. Η τεχνολογία αυτή είναι πολύ χρήσιμη σε περιβάλλοντα στα οποία δεν υπάρχει σήμα GPS, όπως είναι οι εσωτερικοί χώροι και οι ορεινές περιοχές [74].

2.4.5.1 Η Διαδικασία του SLAM

Το SLAM σύμφωνα με το [74] ακολουθεί μια επαναληπτική και αναδρομική διαδικασία χρησιμοποιώντας τα δεδομένα από έναν οι περισσότερους αισθητήρες που είναι ενσωματωμένοι στο ρομπότ ώστε να δημιουργηθεί ο χάρτης του χώρου. Η πρώτη φάση του SLAM είναι ο υπολογισμός της κίνησης του ρομπότ σε σχέση με την αρχική του θέση (Localization) χρησιμοποιώντας την οδομετρία με την βοήθεια αισθητήρων όπως η IMU, οι κάμερες, το ραντάρ και το LiDAR. Στη δεύτερη φάση, η οποία είναι η χαρτογράφηση (Mapping)., χρησιμοποιούνται οι αισθητήρες κάμερας και LiDAR για την σάρωση των αποστάσεων και της διαμόρφωσης των εμποδίων, στη συνέχεια τα δεδομένα που εξάγονται τροφοδοτούνται σε έναν αλγόριθμο ο οποίος δημιουργεί και διορθώνει συνεχώς έναν χάρτη από νέφη σημείων. Η οδομετρία σε αυτή την φάση ενημερώνει για την κίνηση του οχήματος, όμως με την πάροδο του χρόνου συσσωρεύοντας σφάλματα τα οποία ελαχιστοποιούνται από τον αλγόριθμο SLAM. Αυτό το καταφέρνει καθώς αναγνωρίζει αν μια περιοχή την έχει ήδη επισκεφθεί και χρησιμοποιεί το ιστορικό του χάρτη για να επαναφέρει την εκτίμηση της θέσης στην πραγματικότητα.

2.4.5.2 Αλγόριθμοι SLAM

Στα ρομποτικά οχήματα που αναλύθηκαν στο Κεφάλαιο 2 - χρησιμοποιήθηκαν διάφοροι αλγόριθμοι SLAM ανάλογα με τις δυνατότητες του οχήματος και τις απαιτήσεις της εφαρμογής που χρησιμοποιείται. Οι κυριότεροι αλγόριθμοι παρουσιάζονται παρακάτω:

- **Gmapping**

  Ο αλγόριθμος Grid-based FastSLAM (Gmapping) είναι ο πιο διαδεδομένος και αξιόπιστος αλγόριθμος SLAM για το ROS 1 και χρησιμοποιεί το φίλτρο σωματιδίων τύπου Rao-Blackwellized (RBPF) το οποίο δημιουργεί χάρτες υψηλής ακρίβειας [46]. Το φίλτρο αυτό διαχωρίζει την διαδικασία του προσδιορισμού της διαδρομής του ρομπότ από αυτήν της κατασκευής του χάρτη μειώνοντας έτσι το αριθμό των σωματιδίων που απαιτούνται για την δεύτερη διαδικασία. Ο Gmapping χρησιμοποιεί το LiDAR για την μέτρηση των

αποστάσεων των εμποδίων και την οδομετρία, μέσω της IMU ή και των κωδικοποιητών των τροχών, για την αρχική εκτίμηση της κίνησης του ρομπότ, την οποία διορθώνει στην συνέχεια [46]. Σύμφωνα με το [53] ο συγκεκριμένος αλγόριθμος είναι από τους πιο γρήγορους καθώς μέσα από δοκιμές αποδείχθηκε ότι μπορεί να ολοκληρώσει διαδικασίες χαρτογράφησης και πλοήγησης σε πολύ μικρότερο χρόνο από άλλους αλγόριθμους. Ταυτόχρονα σύμφωνα με το [45] ο αλγόριθμος αυτός διαχειρίζεται πολύ αποτελεσματικά τα δεδομένα που περιέχουν θόρυβο βοηθώντας στην δημιουργία λεπτομερών και με μεγάλη ακρίβεια χαρτών ενώ σε περιβάλλοντα προσομοίωσης όπως το Gazebo μπορεί να δημιουργήσει χάρτη με ποσοστό ακρίβειας πάνω από το 95%. Τα μειονεκτήματα του Gmapping είναι 1ον ότι η ανάπτυξή του έχει σταματήσει, 2ον ότι δυσκολεύεται σε χώρους με χαμηλή αντίθεση, όπως οι λείοι διάδρομοι, δημιουργώντας σφάλματα και 3ον όταν εκτελείται σε συστήματα με χαμηλή υπολογιστική ισχύ μπορεί να επηρεαστεί η απόδοση του σε πολύ περίπλοκους χώρους.

- **Karto SLAM**

  Ο αλγόριθμος Karto εκτελεί δισδιάστατη χαρτογράφηση και εντοπισμό θέσης και σε αντίθεση με τον Gmapping που χρησιμοποιεί φίλτρα σωματιδίων, ο συγκεκριμένος χρησιμοποιεί την βελτιστοποίηση γράφων (graph-based SLAM). Ο αλγόριθμος αυτός επεξεργάζεται τα δεδομένα του LiDAR και συγκρίνει τις σαρώσεις του περιβάλλοντος ενώ ταυτόχρονα χρησιμοποιεί τον μηχανισμό Sparse Posture Adjustment (SPA) για να διορθώσει τον χάρτη και την τροχιά του ρομπότ. Αυτός ο μηχανισμός απαιτεί χαμηλή υπολογιστική ισχύ καθώς αναπαριστά το πρόβλημα ως ένα γράφο όπου οι κόμβοι είναι οι θέσεις του ρομπότ και οι ακμές οι περιορισμοί που προκύπτουν από τις μετρήσεις. Ο Karto είναι αρκετά αποδοτικός σε χώρους με κινούμενα εμπόδια και λόγω του SPA μπορεί να εκτελεστεί σε ΚΜΕ που είναι ενσωματωμένες στο ρομπότ και έχουν χαμηλή υπολογιστική ισχύ χωρίς να μειωθεί η ποιότητα του χάρτη. Τα συγκεκριμένα πλεονεκτήματα αποδείχθηκαν σε πειράματα όπου ο Karto είχε την υψηλότερη ακρίβεια στη χαρτογράφηση και την πλοήγηση από τους άλλους αλγόριθμους, όπως ο Cartographer. Επίσης είχε πολύ καλή ισορροπία μεταξύ ακρίβειας και ταχύτητας επεξεργασίας όπου μόνο ο GMapping ήταν ταχύτερος

από αυτόν και βοήθησε στην επίτευξη χαμηλού μέσου σφάλματος παρακολούθησης και πλοήγησης το οποίο ήταν περίπου 18 εκατοστά [45][53].

- **Google Cartographer**

  Ο αλγόριθμος Cartographer της Google εκτελεί αξιόπιστο και ακριβές SLAM σε δύο και τρεις διαστάσεις με την χρήση της μεθόδου σύντηξης αισθητήρων [45]. Ο συγκεκριμένος αλγόριθμος συνδυάζει δεδομένα από πολλούς αισθητήρες όπως το LiDAR και η IMU ώστε να δημιουργήσει λεπτομερείς χάρτες. Ένα ισχυρό χαρακτηριστικό του είναι οι εξελιγμένες τεχνικές που χρησιμοποιεί για να αναγνωρίσει ήδη χαρτογραφημένες περιοχές και να διορθώσει τα σφάλματα της θέσης [53]. Σε πειράματα που έγιναν στο [53] ο Cartographer είχε μεγαλύτερα ποσοστά σφάλματος από τους αλγόριθμους Karto και GMapping και ειδικά σε χώρους με πολλά εμπόδια. Επίσης κατά το [53] ο αλγόριθμος αυτός είναι ο πιο αργός από τους προηγούμενους αλγόριθμους καθώς σε πειράματα με πολλά εμπόδια ο χρόνος πλοήγησης και χαρτογράφησης του ήταν υψηλότερος κατά ένα λεπτό από τους άλλους, γεγονός που δημιουργεί καθυστερήσεις και επηρεάζει την ασφάλεια της πλοήγησης.

- **Hector SLAM**

  Ο αλγόριθμος Hector SLAM μπορεί να εκτελεί χαρτογράφηση και εντοπισμό θέσης χωρίς την χρήση δεδομένων οδομετρίας χρησιμοποιώντας μόνο τους σύγχρονους και πολύ ακριβείς αισθητήρες LiDAR. Ο αλγόριθμος αυτός δημιουργεί τους χάρτες χρησιμοποιώντας την βελτιστοποίηση Gauss-Newton και στη συνέχεια υπολογίζει τη μετατόπιση του ρομπότ συγκρίνοντας την τρέχουσα σάρωση του laser με τον δημιουργημένο χάρτη. Επίσης επειδή δεν χρησιμοποιεί κωδικοποιητές τροχών δεν επηρεάζεται από τα σφάλματα που δημιουργεί η ολίσθηση των τροχών και επομένως είναι χρήσιμος σε ανώμαλα εδάφη ή εναέριες πλατφόρμες (UAVs). Όμως ο Hector SLAM μέσω πειραμάτων αποδείχθηκε ότι αντιμετωπίζει περισσότερα προβλήματα από άλλους αλγόριθμους σε χώρους με πολλά εμπόδια, όπως τα γραφεία, οι καρέκλες και οι κινούμενοι άνθρωποι. Επίσης σε ορισμένα πειράματα ο αλγόριθμος αυτός δεν δημιούργησε αξιόπιστους χάρτες σε χώρους με κινούμενα εμπόδια και χαμηλή αντίθεση υστερώντας σε απόδοση από τον Google Cartographer [41][45][53].

- **RTAB-Map**

  Ο Αλγόριθμος Real-Time Appearance-Based Mapping (RTAB-Map) χρησιμοποιείται για την πραγματοποίηση οπτικού SLAM (V-SLAM) με τον συνδυασμό δεδομένων από διαφορετικούς αισθητήρες [41]. Το χαρακτηριστικό που τον κάνει πιο αποτελεσματικό από άλλους αλγόριθμους στην διαδικασία SLAM σε μεγάλους χώρους είναι η καλή διαχείριση της μνήμης του συστήματος που έχει ως αποτέλεσμα ο χρόνος επεξεργασίας του να είναι μέσα στα όρια του πραγματικού χρόνου σε αντίθεση με άλλους αλγόριθμους. Ο αλγόριθμος αυτός χρησιμοποιεί RGB-D κάμερες αξιοποιώντας ταυτόχρονα την οπτική πληροφορία (RGB) και τα δεδομένα βάθους (Depth) που παρέχει ώστε να δημιουργεί δισδιάστατους και τρισδιάστατους χάρτες και να εντοπίζει τη θέσης του ρομπότ. Επίσης μπορεί να χρησιμοποιήσει και στερεοσκοπικές κάμερες, LiDAR και οδομετρία ώστε με την σύντηξη των αισθητήρων αυτών να αποφευχθούν οι συγκρούσεις και να είναι υψηλή η ακρίβεια του χάρτη [42]. Στα πειράματα στο [41] ο RTAB-Map υπερέχει από τον Cartographer και τον Gmapping όταν υπάρχει πολύ οπτική πληροφορία στον χώρο της κίνησης του ρομπότ, καθώς μπορεί να κατασκευάσει πολύ πιο λεπτομερείς χάρτες από ένα απλό LiDAR [41]. Ενώ σύμφωνα με το [53] η πιστότητα της χαρτογράφησης είναι καλύτερη από αλγόριθμους που βασίζονται μόνο σε LiDAR όπως ο Hector SLAM.

2.4.5.3 Προκλήσεις

Βάσει των δοκιμών που πραγματοποιήθηκαν στα ρομποτικά οχήματα που παρουσιάστηκαν στο Κεφάλαιο 2 - το SLAM αντιμετωπίζει πολλές προκλήσεις όταν η θεωρία των αλγορίθμων αντιμετωπίζει το φυσικό περιβάλλον και τους περιορισμούς της υπολογιστικής ισχύος [54]. Οι κύριες προκλήσεις είναι η διαχείριση των πόρων, τα σφάλματα των αισθητήρων, το χάσμα ανάμεσα στην προσομοίωση και την πραγματικότητα, η επικοινωνία και ο συντονισμός. Η εκτέλεση των αλγορίθμων SLAM απαιτεί μεγάλη μνήμη RAM και ταχύτητα επεξεργαστή και ορισμένες φορές δεν είναι αρκετή η επεξεργαστική ισχύς που έχουν συστήματα που χρησιμοποιούν για ΚΜΕ το Raspberry Pi ή το Jetson Nano. Το γεγονός αυτό έχει ως αποτέλεσμα την παρουσίαση καθυστερήσεων στην επεξεργασία των δεδομένων και αρνητικές επιπτώσεις στην ακρίβεια του εντοπισμού της θέσης σε πραγματικό χρόνο [54]. Ένα άλλο μεγάλο πρόβλημα είναι ο θόρυβος που παρουσιάζουν οι μετρήσεις από τους αισθητήρες και οι

παρεμβολές από το περιβάλλον καθιστώντας την σύντηξη των αισθητήρων αναγκαία [45]. Επίσης η διαμόρφωση του χώρου είναι σημαντικός παράγοντας για την επιτυχία του SLAM καθώς αντικείμενα χαμηλού ύψους δεν εντοπίζονται πάντα από τα LiDAR και απαιτείται η χρησιμοποίηση συμπληρωματικών αισθητήρων για την αναγνώριση τους [54].

Η αποδοτικότητα των αλγορίθμων SLAM εξαρτάται από τα όλα παραπάνω, όμως διαφέρει κατά πολύ όταν εκτελούνται σε πραγματικό ή εικονικό περιβάλλον [50]. Οι προσομοιωτές δεν είναι δυνατόν να αναπαραστήσουν ακριβώς τα χαρακτηριστικά του περιβάλλοντος όπως είναι η τριβή και οι ατέλειες του εδάφους και για αυτό το λόγο αλγόριθμοι που λειτουργούν τέλεια στην προσομοίωση πολλές φορές αποτυγχάνουν στο πραγματικό ρομποτικό όχημα [45]. Παράλληλα η ολίσθηση των τροχών που συμβαίνει σε πραγματικό έδαφος δημιουργεί σφάλματα στην οδομετρία, τα οποία συσσωρεύονται και αν δεν διορθωθούν έχει ως αποτέλεσμα το συνολικό σφάλμα να αυξάνεται απεριόριστα [48].

Τέλος σε ρομποτικά οχήματα που ελέγχονται απομακρυσμένα από ξεχωριστές συσκευές, όπως ένας προσωπικός υπολογιστής, η καλή επικοινωνία είναι σημαντική για την εκτέλεση αλγορίθμων SLAM. Συνήθως τα ασύρματα δίκτυα (WiFi) έχουν καθυστερήσεις και επηρεάζονται από άλλες ηλεκτρομαγνητικές πηγές με κίνδυνο να διακοπεί η σύνδεση του ρομπότ με τον υπολογιστή [75].

2.4.6 Αυτόνομη Πλοήγηση – Nav2

Η Nav2 είναι η νεότερη έκδοση του ROS Navigation Stack και αποτελεί τον εγκέφαλο της αυτόνομης πλοήγησης έχοντας κύρια λειτουργία του να σχεδιάζει την διαδρομή που θα ακολουθήσει το ρομπότ. Μπορεί να αλλάζει την ταχύτητα και την κατεύθυνση του οχήματος ώστε να αποφεύγει εμπόδια και χρησιμοποιεί δέντρα συμπεριφοράς για τη λήψη αποφάσεων, επιτρέποντας στο ρομπότ να εκτελεί σύνθετες αλληλουχίες ενεργειών και να ανακάμπτει από σφάλματα, όπως όταν εγκλωβίζεται το ρομπότ. Η Nav2 παίρνει αποφάσεις σε πραγματικό χρόνο χρησιμοποιώντας διάφορες πληροφορίες του ρομποτικού οχήματος. Καταρχήν πρέπει να γνωρίζει το tf-tree του οχήματος το οποίο καθορίζει τις συνδέσεις των πλαισίων αναφοράς του χάρτη και του ρομπότ. Επίσης η στοίβα χρησιμοποιεί τα δεδομένα από τους αισθητήρες απόστασης ώστε να επιβεβαιώνει την θέση του ρομπότ και να εντοπίζει τα εμπόδια στον χώρο, τα οποία τα αναπαριστά μαζί με τον ελεύθερο χώρο σε δισδιάστατους χάρτες κόστους [48].

Ο σχεδιασμός της διαδρομής εκτελείται από αλγόριθμους ελέγχου σε δύο επίπεδα οι οποίοι συνεργάζονται μεταξύ τους, αυτοί είναι ο καθολικός και ο τοπικός σχεδιαστής [45]. Ο καθολικός σχεδιαστής (Global Planner) καθορίζει την καλύτερη διαδρομή που μπορεί να εκτελέσει το όχημα από την θέση που βρίσκεται προς έναν στόχο χρησιμοποιώντας τον δημιουργημένο χάρτη. Στην συνέχεια όταν αρχίζει να κινείται το ρομπότ ο τοπικός σχεδιαστής (Local Planner) δίνει εντολές προσαρμόζοντας την ταχύτητα των κινητήρων του οχήματος ανάλογα με τα κινούμενα εμπόδια και άλλες παραμέτρους που δεν έχει υπολογίσει αρχικά ο Global Planner [48]. Οι πιο κοινοί Local Planners είναι ο DWB (Dynamic Window Approach variant) [45] και ο TEB (Timed Elastic Band) [50]. Αντίστοιχα με τους Planners υπάρχουν, Global και Local χάρτες κόστους (Costmaps), όπου οι πρώτοι χρησιμοποιούνται για μακροπρόθεσμο σχεδιασμό καλύπτοντας ολόκληρο το περιβάλλον και οι δεύτεροι χρησιμοποιούνται για την αποφυγή εμποδίων παρακολουθώντας ένα κυλιόμενο παράθυρο γύρω από το ρομπότ με δεδομένα από τους αισθητήρες σε πραγματικό χρόνο. Τέλος η Nav2 ενσωματώνει επίσης πακέτα με τα οποία γίνεται ο προσδιορισμός της στάσης του ρομπότ (Localization), όπως στην πρώτη φάση του SLAM, τέτοιο πακέτο είναι το Adaptive Monte Carlo Localization (AMCL), το οποίο συγκρίνει τα δεδομένα του LiDAR με τον χάρτη χρησιμοποιώντας φίλτρο σωματιδίων ώστε να διορθώσει τα συσσωρευμένα σφάλματα της οδομετρίας [48].

Η επιτυχία της πλοήγησης κατά το [48] εξαρτάται από την σωστή ρύθμιση των παραμέτρων στο αρχείο nav2_params.yaml. Μία βασική προϋπόθεση είναι να αντικατοπτριστεί το αποτύπωμα του ρομπότ το οποίο είναι αναγκαίο ώστε να είναι γνωστές οι φυσικές διαστάσεις του οχήματος. Στην συνέχεια πρέπει να καθοριστούν τα όρια επιτάχυνσης και ταχύτητας του οχήματος ώστε να αποφευχθούν απότομες κινήσεις που είναι πιθανόν να προκαλέσουν την ολίσθηση των τροχών με τις συνέπειες που έχουν αναφερθεί προηγουμένως. Επίσης σημαντικό είναι να καθοριστεί η απόσταση ασφαλείας που θα κρατηθεί από τα εμπόδια κατά την κίνηση του ρομπότ ώστε να αποφευχθούν οι συγκρούσεις. Τέλος κατά το [37] το Nav2 είναι ένα ισχυρό εργαλείο για τα αυτόνομα οχήματα, το οποίο τους επιτρέπει να κινούνται αυτόνομα σε περίπλοκα περιβάλλοντα χρησιμοποιώντας τους παραπάνω προηγμένους αλγόριθμους ελέγχου.

Κεφάλαιο 3 - Ανάπτυξη Πλατφόρμας Προσομοίωσης

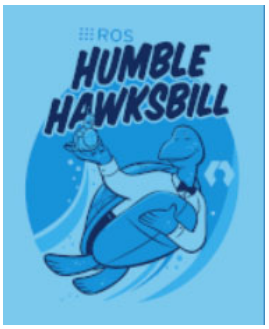
Στο 3^{ο} κεφάλαιο θα παρουσιαστούν τα βήματα που έγιναν για την πραγματοποίηση της παρούσας διατριβής, ξεκινώντας από την επιλογή της έκδοσης και την εγκατάσταση του λογισμικού της πλατφόρμας και καταλήγοντας στην αξιολόγηση της.

3.1. Επιλογή Έκδοσης Λογισμικού

Κρίσιμο ζήτημα για την ανάπτυξη μιας πλατφόρμας είναι η επιλογή του υλικού και του λογισμικού της. Στην περίπτωση ενός AMR το κύριο λογισμικό είναι το ROS2 και είναι σημαντικό να χρησιμοποιηθεί η κατάλληλη έκδοση του, όσον αφορά την συμβατότητα και την υποστήριξη του, ώστε να είναι η υλοποίηση του όσο το δυνατόν γρηγορότερη και πλήρης.

Σύμφωνα με το [76] υπάρχουν τρεις εκδόσεις του ROS2 που υποστηρίζονται επίσημα και δεν έχουν φτάσει ακόμα στο τέλος του κύκλου ζωής τους (end-of-life - EOL). Αυτές οι εκδόσεις είναι:

- Η Kilted Kaiju, η οποία είναι η πιο πρόσφατη και εξελιγμένη από τις άλλες εκδόσεις, όμως έχει μικρή διάρκεια ζωής με υποστήριξη έως τον Νοέμβριο του 2026.
- Η Jazzy Jalisco, η οποία είναι η πιο πρόσφατη Long-Term Support (LTS) έκδοση με έτος δημιουργίας το 2024 και υποστήριξη μέχρι το 2029. Όμως επειδή είναι τόσο νέα υπάρχει μεγάλη πιθανότητα ορισμένα πακέτα λογισμικού ή οδηγοί συσκευών να μην είναι ακόμα τόσο ώριμα για ARM.
- Και η Humble Hawksbill (LTS) της οποίας το λογότυπο φαίνεται στην Εικόνα 3-1 και έχει έτος δημιουργίας το 2022 και υποστήριξη μέχρι τα μέσα του 2027. Η έκδοση αυτή είναι η πιο κατάλληλη για ARM όσον αφορά την σταθερότητα και την υποστήριξη της. Αυτό οφείλεται στο γεγονός ότι υπάρχει μεγάλη κοινότητα ελεύθερου λογισμικού (Free Open Source Software - FOSS) με πολλά tutorials και πακέτα ήδη χτισμένα για ARM. Επίσης είναι απόλυτα συμβατή με το λειτουργικό σύστημα Ubuntu 22.04 (Jammy), το οποίο είναι το πιο ώριμο για το Raspberry Pi.



Εικόνα 3-1. ROS2 Humble Hawksbill

Λόγω των παραπάνω από τις συγκεκριμένες εκδόσεις του ROS2 επιλέχθηκε η Humble Hawksbill (LTS) [77], ενώ από τις εκδόσεις του λογισμικού Ubuntu που υπάρχουν, επιλέχθηκε η Ubuntu 22.04 (Jammy) [78] καθώς σύμφωνα με το [76] είναι η μοναδική έκδοση που υποστηρίζει την Humble Hawksbill. Όσον αφορά την εγκατάσταση του λογισμικού Ubuntu στον φορητό υπολογιστή υπάρχουν οι επιλογές της απευθείας εγκατάστασης του στον σκληρό δίσκο, της εικονικής μηχανής και του Windows Subsystem for Linux 2 (WSL2).

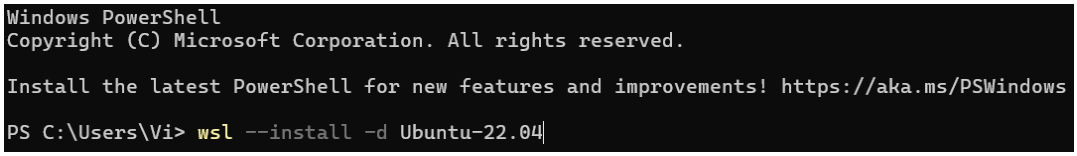
Από τα παραπάνω επιλέχθηκε το WSL2, όπου σύμφωνα με το [79], είναι ένα εργαλείο των Windows 11, με το οποίο υπάρχει η δυνατότητα της εκκίνησης ενός πλήρους περιβάλλοντος τερματικού Linux απευθείας στα Windows χωρίς τη χρησιμοποίηση εικονικής μηχανής. Η επιλογή αυτή έγινε λόγω της απόδοσης του WSL2, η οποία είναι πολύ κοντά στην native και της πρόσβασης που παρέχει, μέσα από τα windows, στο ROS2, το Gazebo και το SLAM. Επίσης προσφέρει πρόσβαση στους φακέλους του Linux μέσα από τον περιηγητή αρχείων των windows και καλύτερη κατανομή μνήμης και CPU σε σύγκριση με τις εικονικές μηχανές.

Τέλος όσον αφορά το λογισμικό προσομοίωσης επιλέχθηκε το Gazebo, το οποίο παρουσιάζεται στην ενότητα 2.4.2.1, και συγκεκριμένα η έκδοση Fortress καθώς είναι η πιο πρόσφατη και εξελίσσεται συνεχώς. Η έκδοση αυτή διαθέτει βελτιωμένες δυνατότητες φυσικής προσομοίωσης και ευκολότερη ενσωμάτωση με το ROS 2 μέσω της γέφυρας ros_gz.

3.2. Εγκατάσταση Λογισμικού

3.2.1 Εγκατάσταση Ubuntu 22.04 LTS σε WSL

Η εγκατάσταση του Ubuntu 22.04 σε WSL γίνεται μέσω της εφαρμογής γραμμής εντολών powershell και συγκεκριμένα με την εντολή που φαίνεται στην Εικόνα 3-2 [80].

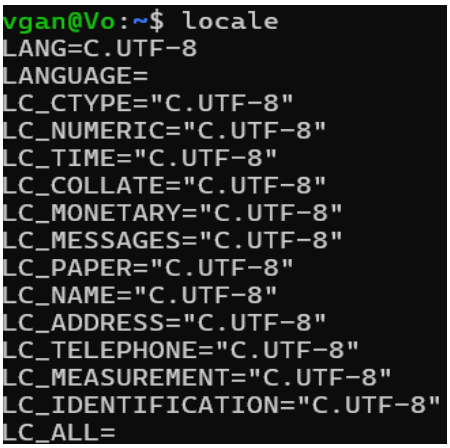


Εικόνα 3-2. Εντολή Εγκατάστασης Ubuntu-22.04 σε WSL2

Εκκίνηση: Για να εκκινήσουμε το περιβάλλον του λειτουργικού συστήματος Ubuntu ανοίγουμε από το Μενού Έναρξης την εφαρμογή "Ubuntu 22.04".

Μετά την εκκίνηση ελέγχουμε αν υπάρχει υποστήριξη για Αγγλικά (Η.Π.Α.) με κωδικοποίηση UTF-8 με την εντολή '**locale**' καθώς το ROS2 απαιτεί αυτήν την κωδικοποίηση για την σωστή εμφάνιση χαρακτήρων και μηνυμάτων λάθους. Συνεπώς εάν το αποτέλεσμα της εντολής αυτής δεν είναι όπως στην Εικόνα 3-3 εκτελούμε με την σειρά τις παρακάτω εντολές [81] :

- **sudo apt update && sudo apt upgrade,** που ενημερώνει τη λίστα των διαθέσιμων πακέτων από τα διαθέσιμα αποθετήρια και αναβαθμίζει τα εγκατεστημένα πακέτα στην τελευταία έκδοση. Κατά την εκτέλεση της εντολής θα εμφανιστούν τα πακέτα που πρόκειται να εγκατασταθούν, ο αριθμός των kB που θα ληφθούν γι' αυτά, ο χώρος που θα καταλάβουν στο δίσκο και τέλος θα μας ζητηθεί να επιβεβαιώσουμε την ενέργεια που πρόκειται να πραγματοποιηθεί πληκτρολογώντας τον χαρακτήρα "y".
- **sudo apt install locales -y,** που εγκαθιστά τα πακέτα για την διαχείριση των τοπικών ρυθμίσεων. Η ρύθμιση "**-y**" εκτελεί την εντολή χωρίς να χρειάζεται την δική μας επιβεβαίωση όπως προηγουμένως.
- **sudo locale-gen en_US en_US.UTF-8 ,** η οποία δημιουργεί τις τοπικές ρυθμίσεις για Αγγλικά ΗΠΑ σε UTF-8 κωδικοποίηση.
- **sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8 ,** η οποία ορίζει μόνιμα τις τοπικές ρυθμίσεις συστήματος σε UTF-8.
- **export LANG=en_US.UTF-8 ,** που ορίζει τις τοπικές ρυθμίσεις για το τρέχον τερματικό.



Εικόνα 3-3. Αποτέλεσμα εντολής locale

3.2.2 Εγκατάσταση ROS2 Humble Hawksbill Desktop

Το πακέτο ros-humble-desktop δεν υπάρχει στο προεπιλεγμένο αποθετήριο του λογισμικού Ubuntu. Για αυτό το λόγο πρέπει να προστεθεί το ROS2 αποθετήριο στο Ubuntu ώστε να μπορεί να εγκατασταθεί το πακέτο ROS2 Humble. Η πρόσθεση του αποθετηρίου και η εγκατάσταση του πακέτου γίνεται με τις παρακάτω εντολές [81]:

- **sudo apt update –y ,** η οποία ενημερώνει τη λίστα πακέτων μετά τις αλλαγές.
- **sudo apt install curl gnupg2 lsb-release software-properties-common**
  η οποία εγκαθιστά τέσσερα πακέτα ταυτόχρονα που χρειάζονται για την πρόσθεση αποθετηρίων τρίτων και την εγκατάσταση πακέτων με ασφάλεια. Συγκεκριμένα το πακέτο ***curl*** είναι εργαλείο για την λήψη δεδομένων, το **gnupg2** είναι απαραίτητο για την διαχείριση των δημόσιων κλειδιών (GPG κλειδιών), το **lsb-release** δίνει πληροφορίες για την έκδοση του λειτουργικού και το **software-properties-common** διαθέτει εργαλεία όπως το add-apt-repository που επιτρέπει την εύκολη προσθήκη νέων PPA αποθετηρίων.
- **sudo add-apt-repository universe ,** η οποία ενεργοποιεί το καθολικό αποθετήριο του Ubuntu που περιέχει μη επίσημα αλλά ευρέως χρησιμοποιούμενα πακέτα.
- **sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg,** η οποία κατεβάζει το GPG κλειδί που χρησιμοποιεί το ROS αποθετήριο για να υπογράφει τα πακέτα του και το

αποθηκεύει στο /usr/share/keyrings/ros-archive-keyring.gpg ώστε το apt να μπορεί να ελέγχει την αυθεντικότητα των πακέτων.

- **echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null,** η οποία προσθέτει τη διεύθυνση του αποθετηρίου του ROS2 στην λίστα με τις πηγές του apt.
- **sudo apt update –y ,** η οποία ενημερώνει την τοπική λίστα πακέτων με τα διαθέσιμα πακέτα από το αποθετήριο ROS2.
- **sudo apt install ros-humble-desktop –y ,** η οποία εγκαθιστά το ROS2 Humble Hawksbill έκδοσης Desktop που περιέχει τον πυρήνα του ROS2, το Rviz2, διάφορα demo και tutorials.
- **sudo apt install ros-dev-tools ,** η οποία εγκαθιστά τα εργαλεία δημιουργίας πακέτων ROS όπως είναι οι μεταγλωττιστές.

Στην συνέχεια γίνεται η ρύθμιση του περιβάλλοντος ROS2 στο τερματικό, με τις εντολές:

- **echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc ,** η οποία προσθέτει σε μια γραμμή στο αρχείο .bashrc του χρήστη την εντολή "**source"** ώστε να φορτώνεται αυτόματα το περιβάλλον ROS2 και οι ρυθμίσεις του κάθε φορά που ανοίγει νέο τερματικό. Συγκεκριμένα η εντολή source εκτελεί ένα αρχείο ".bash" σε ένα κέλυφος περιβάλλοντος ώστε οι μεταβλητές, οι συναρτήσεις και άλλες απαραίτητες ρυθμίσεις, που ορίζονται σε αυτό το αρχείο, να γίνονται διαθέσιμες στο τρέχον περιβάλλον.
- **source ~/.bashrc ,** η οποία φορτώνει το setup του ROS χωρίς να χρειαστεί να κλείσει και να ανοίξει νέο τερματικό.

3.2.3 Εγκατάσταση Πακέτων του ROS2

Το λογισμικό ROS2 έκδοσης Humble Hawksbill Desktop περιλαμβάνει το Rviz2, ενώ άλλα πακέτα που χρειάζονται πρέπει να εγκατασταθούν χειροκίνητα.

3.2.3.1 Εγκατάσταση Gazebo Fortress

Για την εγκατάσταση του προσομοιωτή Gazebo εκτελούνται με την σειρά οι εντολές:

- **sudo curl -sSL https://packages.osrfoundation.org/gazebo.gpg -o /usr/share/keyrings/gazebo-archive-keyring.gpg ,** η οποία κατεβάζει το GPG κλειδί του αποθετηρίου του Gazebo και το αποθηκεύει στο /usr/share/keyrings/gazebo-archive-keyring.gpg.
- **echo "deb [arch=amd64 signed-by=/usr/share/keyrings/gazebo-archive-keyring.gpg] http://packages.osrfoundation.org/gazebo/ubuntu-stable jammy main" | sudo tee /etc/apt/sources.list.d/gazebo-stable.list ,** η οποία προσθέτει τη διεύθυνση του παραπάνω αποθετηρίου στην λίστα με τις πηγές του apt.
- **sudo apt update –y ,** με την οποία γίνεται ενημέρωση νέων πακέτων
- **sudo apt install gz-fortress –y ,** όπου γίνεται η εγκατάσταση του Gazebo Fortress

Σημείωση : Όπως αναφέρεται στην αρχή του κεφαλαίου, για την εκτέλεση του λειτουργικού συστήματος Ubuntu, στο οποίο εκτελείται το Gazebo, χρησιμοποιούμε εικονική μηχανή WSL. Όμως οι εικονικές μηχανές δεν έχουν σωστή υποστήριξη OpenGL και επομένως είναι αναγκαίο να γίνεται η λειτουργία της OpenGL μέσω CPU αντί για GPU. Για να γίνει η ρύθμιση αυτή εκτελείται πριν την εκκίνηση του Gazebo η εντολή :

export LIBGL_ALWAYS_SOFTWARE=1.

Και στην συνέχεια εκτελείται η εκκίνηση του Gazebo Fortress με την εντολή :

ign gazebo

3.2.3.2 Επικοινωνία ROS2 Humble με Gazebo Simulator

Για να επιτευχθεί η σύνδεση των topic, services και actions του Gazebo με αυτά του ROS2 είναι απαραίτητο να εγκατασταθεί το πακέτο ros-humble-ros-gz. Το πακέτο αυτό είναι η επίσημη γέφυρα των δύο αυτών συστημάτων και μετατρέπει με εντολές bridge τα μηνύματα που ανταλλάσουν μεταξύ τους. Η εγκατάσταση του συγκεκριμένου πακέτου γίνεται με την εντολή:

sudo apt install ros-humble-ros-gz -y

3.2.3.3 SlamToolbox

Το πακέτο Slam Toolbox [81] είναι ένα σύγχρονο πακέτο SLAM για ROS 2, το οποίο μπορεί να συνδυάζει δεδομένα από LiDAR και μετασχηματισμούς TF ώστε να δημιουργεί έναν δισδιάστατο χάρτη ενός χώρου, να τον αποθηκεύει και να τον ξαναφορτώνει για να συνεχιστεί η χαρτογράφηση, ο εντοπισμός ή ο χειρισμός του με άλλο τρόπο. Επιτρέπει την εκτέλεση

αλγορίθμων SLAM με δύο διαφορετικούς τρόπους λειτουργίας, όπου η πρώτη είναι η σύγχρονη, στην οποία περιμένει να παραλάβει όλα τα δεδομένα πριν από την επεξεργασία και επίσης κάθε σάρωση επεξεργάζεται με τη σειρά, γεγονός που μπορεί να προκαλέσει καθυστέρηση αν υπάρχουν αργοί αισθητήρες. Η δεύτερη είναι η ασύγχρονη, όπου επεξεργάζονται τα δεδομένα μόλις φτάνουν παρέχοντας καλύτερη απόδοση και λιγότερες καθυστερήσεις. Το Slam Toolbox υποστηρίζει τους αλγόριθμους Karto SLAM που είναι ο προεπιλεγμένος, Gmapping για συμβατότητα με το ROS1, Google's Cartographer και Async Google's Cartographer. Η εγκατάσταση του γίνεται με την εντολή: **sudo apt install ros-humble-slam-toolbox –y**

3.2.3.4 Nav2

Το πακέτο Nav2 είναι το επίσημο Navigation Stack του ROS 2 που παρουσιάστηκε στην ενότητα Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε. και η εγκατάσταση του με τα planners και τα πρόσθετα γίνεται με την εντολή:

**sudo apt install ros-humble-navigation2 ros-humble-nav2-bringup **
**ros-humble-nav2-smac-planner ros-humble-nav2-theta-star-planner **
**ros-humble-nav2-constrained-smoother ros-humble-nav2-velocity-smoother **
**ros-humble-nav2-lifecycle-manager –y **
ros-humble-nav2-rviz-plugins –y

3.2.3.5 RTAB-Map

Το πακέτο RTAB-Map μπορεί να πραγματοποιεί μεταξύ άλλων και οπτικό SLAM συνδυάζοντας δεδομένα από LiDAR, κάμερα και μετασχηματισμούς TF. Κατά την εκτέλεση του δημιουργούνται ένας δισδιάστατος και ένας τρισδιάστατος χάρτης του χώρου, τους οποίους αποθηκεύει και ξαναφορτώνει για να γίνει ο εντοπισμός και η αυτόματη πλοήγηση του ρομπότ. Η εγκατάσταση του γίνεται με την εντολή:

sudo apt install ros-humble-rtabmap-ros ros-humble-rtabmap-viz

Και η εγκατάσταση των πακέτων για την τρισδιάστατη οπτικοποίηση γίνεται με την εντολή:

sudo apt install ros-humble-octomap-msgs ros-humble-octomap-rviz-plugins

3.3. Ανάπτυξη Λογισμικού Προσομοίωσης

Για την ανάπτυξη του λογισμικού της προσομοίωσης του εξελιγμένου FOSSBot δημιούργησα στην αρχή το πακέτο της προσομοίωσης και μέσα σε αυτό το ψηφιακό μοντέλο

του FOSSBot σε αρχείο SDF με την απαραίτητη βασική δομή, γραφικά, αισθητήρες και πρόσθετα ώστε να μοιάζει και να συμπεριφέρεται το ψηφιακό όπως το πραγματικό. Επίσης χρησιμοποιήθηκε αρχείο .WORLD για την προσομοίωση του περιβάλλοντος στο οποίο θα κινείται το ρομπότ και στην συνέχεια δημιουργήθηκε αρχείο .SH για την αυτόματη εκκίνηση των συστημάτων της προσομοίωσης και αρχεία για την αποθήκευση των ρυθμίσεων των εργαλείων του ROS2.

3.3.1 Πακέτο Προσομοίωσης – FOSSBot_Sim

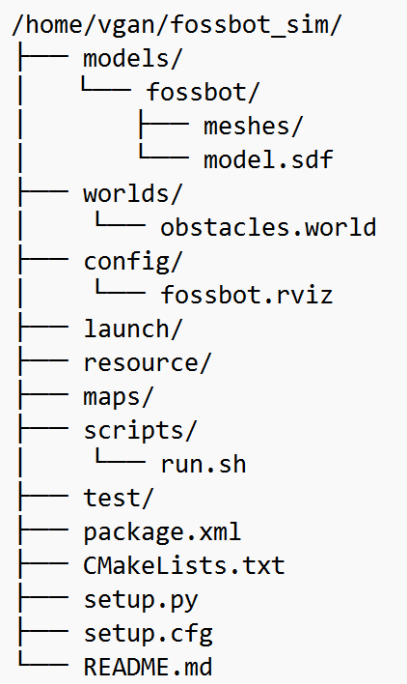
Η δημιουργία του ROS2 πακέτου στον γονικό φάκελο του χρήστη γίνεται με την εντολή:

ros2 pkg create --build-type ament_python fossbot_sim

Μετά την εκτέλεση της παραπάνω εντολής δημιουργούνται οι απαραίτητοι φάκελοι και αρχεία για την λειτουργία του πακέτου. Οι υπόλοιποι φάκελοι και αρχεία που χρειάζονται για την προσομοίωση δημιουργούνται με τις παρακάτω εντολές:

- Εντολή Δημιουργίας Φακέλων:
  mkdir -p fossbot_sim/{launch,config,models/fossbot/meshes,worlds,scripts,maps}
  με την οποία δημιουργούνται μέσα στο fossbot_sim οι φάκελοι launch, config, scripts, worlds, maps και models και οι υποφάκελοι fossbot και meshes.
- Εντολές Δημιουργίας Αρχείων (τα οποία θα αναλυθούν στην συνέχεια του κεφαλαίου):
  touch fossbot_sim/models/fossbot/model.sdf
  touch fossbot_sim/worlds/obstacles.world
  touch fossbot_sim/config/fossbot.rviz
  touch fossbot_sim/scripts/sim.sh
- Εντολή επεξεργασίας των αρχείων:
  nano fossbot_sim/models/fossbot/model.sdf

Μετά την ολοκλήρωση της δημιουργίας των παραπάνω φακέλων και αρχείων προκύπτει η δομή του πακέτου που φαίνεται στην Εικόνα 3-4.



Εικόνα 3-4 Δομή Πακέτου fossbot_sim

Σε αυτό το σημείο της ανάπτυξης όπου είναι έτοιμη η δομή του πακέτου και κάθε φορά που δημιουργείται ή προστίθεται στο πακέτο νέο αρχείο ή φάκελος εκτελούνται στον γονικό φάκελο του χρήστη οι παρακάτω εντολές:

- **colcon build --packages-select fossbot_sim**
  η οποία είναι το build system της ROS2, όπως το catkin_make της ROS1, και κατασκευάζει το ROS 2 πακέτο fossbot_sim. Μετά την εκτέλεση της εντολής αυτής είναι δυνατή η εκτέλεση των nodes και των launch αρχείων (.PY) και τα config files αντιγράφονται στον φάκελο share/fossbot_sim/ του ROS2.
- **source install/setup.bash**
  η οποία φορτώνει τις ρυθμίσεις στο τρέχον περιβάλλον από το αρχείο install/setup.bash που δημιουργεί η προηγούμενη εντολή.

3.3.2 Μοντέλο FOSSBot

3.3.2.1 Αρχεία meshes

Για την τρισδιάστατη απεικόνιση του πλαισίου του FOSSBot στο περιβάλλον του Gazebo χρησιμοποιήθηκαν τα αρχεία STL που υπάρχουν στο αποθετήριο [83]. Στο συγκεκριμένο αποθετήριο δεν υπάρχουν αρχεία για την αναπαράσταση των τροχών και γι' αυτό το λόγο έγινε λήψη από το αποθετήριο [84] των αρχείων της προσομοίωσης του αρχικού FOSSBot για την εφαρμογή CoppeliaSim. Στην συνέχεια μέσω της εφαρμογής copelliasim 4.10 έγινε εκκίνηση του αρχείου fossbot.ttt και από το μενού Modules-Exporters-Urdf exporter επιλέχθηκαν όλα τα στοιχεία και έγινε εξαγωγή του μοντέλου σε fossbot.urdf, συνοδευόμενο από τα DAE αρχεία της γραφικής αναπαράστασης του πλαισίου του ρομπότ. Από τα συγκεκριμένα αρχεία χρησιμοποιήθηκαν τα DAE αρχεία που αναπαριστούν τους τροχούς και μετατράπηκαν σε STL αρχεία με την εφαρμογή meshlab και τις εντολές:

```
for dae_file in ~/fossbot_sim/models/fossbot/meshes/*.dae; do
        stl_file="${dae_file%.dae}.stl"
        meshlabserver -i "$dae_file" -o "$stl_file"
        echo "Converted: $(basename $dae_file) to STL" done
```

ενώ είχε εγκατασταθεί η εφαρμογή MeshLab με την εντολή: **sudo apt install meshlab**

Τέλος για τον αισθητήρα LiDAR, τον οποίο δεν διαθέτει το αρχικό FOSSBot, χρησιμοποιήθηκαν τα DAE αρχεία από το TugBot [18], τα οποία υπάρχουν στο αποθετήριο [25] και αναπαριστούν τον αισθητήρα VLP-16 που χρησιμοποιεί το συγκεκριμένο ρομποτικό όχημα. Τα παραπάνω αρχεία τρισδιάστατων μοντέλων τοποθετήθηκαν στον φάκελο meshes ώστε να ενσωματωθούν στο αρχείο που περιγράφει το ψηφιακό μοντέλο του αναβαθμισμένου FOSSBot.

3.3.2.2 Αρχείο μοντέλου - model.sdf

Για την περιγραφή του μοντέλου του FOSSBot σε μορφή SDF χρησιμοποιήθηκαν στοιχεία από δύο άλλα παρόμοια ρομποτικά οχήματα, το TugBot με πηγή το αποθετήριο [85] και το TurtleBot 3 [86] με πηγή το αποθετήριο [87]. Η δομή του αρχείου model.sdf περιλαμβάνει με τη σειρά το κύριο σώμα του ρομπότ με τα γεωμετρικά χαρακτηριστικά του και το τρισδιάστατο μοντέλο του, τους αισθητήρες, δύο τροχούς μπροστά με διαφορικό σύστημα που κινούν το ρομπότ και έναν τροχό πίσω χωρίς κίνηση για ισορροπία. Στο τέλος υπάρχουν οι αρθρώσεις για την σύνδεση των παραπάνω στοιχείων και τα πρόσθετα για τον έλεγχο της

κίνησης και της δημοσίευσης των δεδομένων των αισθητήρων και των μετασχηματισμών των συντεταγμένων (transforms - tf) μεταξύ διαφορετικών εξαρτημάτων στο χώρο.

3.3.2.2.1 Κύριο σώμα

Στο κύριο σώμα του αρχείου όπως φαίνεται στην Εικόνα 3-5 περιγράφεται το όνομα, η θέση και ο προσανατολισμός του ρομπότ καθώς και το σημείο επαφής του με το έδαφος.

```
<model name="fossbot">
    <pose>0.0 0.0 0.0 0.0 0.0 0.0</pose>
    <link name="base_footprint"/>
```

Εικόνα 3-5 Αρχείο SDF αρχή

Στην συνέχεια ορίζονται οι ιδιότητες αδράνειας του κύριου σώματος, οι οποίες καθορίζουν πώς το ρομπότ επιταχύνει, περιστρέφεται και αντιδρά σε εξωτερικές δυνάμεις. Σε αυτές τις ιδιότητες, όπως φαίνεται στην Εικόνα 3-6, διακρίνουμε στην αρχή τη θέση του κέντρου μάζας και τον προσανατολισμό (pose) του σώματος σε σχέση με το πλαίσιο αναφοράς. Ενώ παρακάτω καθορίζεται η ροπή αδράνειας για τους άξονες και η μάζα (mass) του συγκεκριμένου συνδέσμου (link) σε κιλά.

```
<inertial>
    <pose>-0.032 0 0.070 0 0 0</pose>
    <inertia>
        <ixx>7.2397393e-01</ixx>
        <ixy>4.686399e-10</ixy>
        <ixz>-1.09525703e-08</ixz>
        <iyy>7.2397393e-01</iyy>
        <iyz>2.8582649e-09</iyz>
        <izz>6.53050163e-01</izz>
    </inertia>
    <mass>8.2573504e-01</mass>
</inertial>
```

Εικόνα 3-6 Inertial

Παρακάτω καθορίζεται η γεωμετρία σύγκρουσης για τον σύνδεσμο αυτό, η οποία χρησιμοποιείται για τη φυσική προσομοίωση δηλαδή την ανίχνευση των συγκρούσεων. Στην συγκεκριμένη περίπτωση το σχήμα που χρησιμοποιείται για τους υπολογισμούς αυτούς είναι ένα ορθογώνιο κουτί και οι διαστάσεις του φαίνονται στην Εικόνα 3-7.

```
<collision name="base_collision">
    <pose>-0.032 0 0.070 0 0 0</pose>
    <geometry>
        <box>
            <size>0.078 0.16 0.15</size>
        </box>
    </geometry>
</collision>
```

Εικόνα 3-7 Base Collision

Τέλος δημιουργείται η οπτική αναπαράσταση (visual) του συνδέσμου καθώς όπως φαίνεται στην Εικόνα 3-8 φορτώνεται η γεωμετρία (mess) από ένα εξωτερικό αρχείο τρισδιάστατου μοντέλου (STL) και ορίζονται οι ιδιότητες του υλικού (material), δηλαδή το χρώμα και το πώς αλληλεπιδρά με το φως.

```
<visual name="base_visual">
    <pose>-0.032 0 0 0 0 0</pose>
    <geometry>
        <mesh>
            <uri>model://fossbot/meshes/robot_base.stl</uri>
            <scale>1 1 1</scale>
        </mesh>
    </geometry>
    <material>
        <ambient>0.1 0.1 0.9 1.0</ambient>
        <diffuse>0.1 0.1 0.9 1.0</diffuse>
        <specular>0.1 0.1 0.8 1.0</specular>
    </material>
</visual>
```

Εικόνα 3-8 Base Visual

3.3.2.2.2 Αισθητήρες

Οι βασικές παράμετροι για τους αισθητήρες είναι, όπως και σε όλους τους συνδέσμους, η θέση, ο προσανατολισμός και η οπτική αναπαράσταση. Στην Εικόνα 3-9 φαίνεται η οπτική αναπαράσταση της IMU η οποία ορίζεται ως ένα κουτί με συγκεκριμένα χρώματα ενώ η οπτική αναπαράσταση του LiDAR όπως φαίνεται στην Εικόνα 3-10 ορίζεται από ένα αρχείο τύπου DAE.

```
<!-- IMU -->

<link name="imu">
    <pose>-0.05 0 0.05 0 0 0</pose>
    <visual name="imu_visual">
        <geometry>
            <box>
                <size>0.01 0.01 0.01</size>
            </box>
        </geometry>
        <material>
            <ambient>1.0 0.8 0.0 1.0</ambient>
            <diffuse>1.0 0.8 0.0 1.0</diffuse>
            <specular>0.1 0.1 0.1 1.0</specular>
        </material>
    </visual>
```

Εικόνα 3-9 IMU Visual

```
<link name="lidar">
    <pose>-0.04 0 0.105 0 0 0 </pose>
    <visual name="lidar_visual">
        <pose>0 0 0 0 0 0</pose>
        <geometry>
            <mesh>
                <uri>model://fossbot/meshes/VLP16_scan.dae</uri>
                <scale>0.4 0.4 0.4</scale>
            </mesh>
        </geometry>
    </visual>
```

Εικόνα 3-10 LIDAR Visual

- IMU

Στη συνέχεια όπως φαίνεται στην Εικόνα 3-11 ορίζονται το όνομα, ο τύπος και οι ρυθμίσεις του αισθητήρα καθώς και το όνομα του ROS2 topic στο οποίο δημοσιεύονται τα δεδομένα του αισθητήρα. Ο παρακάτω αισθητήρας IMU ρυθμίζεται να είναι πάντα ενεργός κατά τη διάρκεια της προσομοίωσης (always_on) και να δημοσιεύει τα δεδομένα του με ρυθμό 200Hz στο topic /imu. Ο IMU μετρά την γωνιακή ταχύτητα (angular velocity) δηλαδή την ταχύτητα περιστροφής γύρω από κάθε άξονα και την γραμμική επιτάχυνση (linear acceleration) κατά μήκος κάθε άξονα και για να είναι οι μετρήσεις αυτές πιο ρεαλιστικές τους προσθέτει χαμηλό Gaussian θόρυβο με απόκλιση 0.0002 rad/s και 0.017 m/s αντίστοιχα. Στη συνέχεια δημοσιεύει τα δεδομένα της γωνιακής ταχύτητας με μορφή x, y, z σε rad/s και της γραμμικής επιτάχυνσης με x, y, z σε m/s^2.

```
<sensor name="imu" type="imu">
    <always_on>true</always_on>
    <update_rate>200</update_rate>
    <topic>imu</topic>
    <imu>
        <angular_velocity>
            <x>
                <noise type="gaussian">
                    <mean>0.0</mean>
                    <stddev>2e-4</stddev>
                </noise>
            </x>
            <y>
                <noise type="gaussian">
                    <mean>0.0</mean>
                    <stddev>2e-4</stddev>
                </noise>
            </y>
            <z>
                <noise type="gaussian">
                    <mean>0.0</mean>
                    <stddev>2e-4</stddev>
                </noise>
            </z>
        </angular_velocity>
        <linear_acceleration>
            <x>
                <noise type="gaussian">
                    <mean>0.0</mean>
                    <stddev>1.7e-2</stddev>
                </noise>
            </x>
            <y>
                <noise type="gaussian">
                    <mean>0.0</mean>
                    <stddev>1.7e-2</stddev>
                </noise>
            </y>
            <z>
                <noise type="gaussian">
                    <mean>0.0</mean>
                    <stddev>1.7e-2</stddev>
                </noise>
            </z>
        </linear_acceleration>
    </imu>
</sensor>
```

Εικόνα 3-11 IMU Sensor

- LiDAR

Οι ρυθμίσεις του αισθητήρα LiDAR φαίνονται στην Εικόνα 3-12 στις οποίες ορίζεται ότι στο συγκεκριμένο μοντέλο είναι ένας παν-κατευθυντικός (omni) σαρωτής λέιζερ, ο οποίος χρησιμοποιεί την GPU (gpu_lidar) για τους υπολογισμούς επιταχύνοντας την απόδοση σε σχέση με τον CPU. Ο συγκεκριμένος αισθητήρας βρίσκεται 30 cm πάνω από το κέντρο του συνδέσμου, εκτελεί 10 πλήρεις σαρώσεις ανά

δευτερόλεπτο (update_rate) και δημοσιεύει τα δεδομένα στο ROS2 topic /scan. Από τις παραμέτρους καθορίζεται ότι εκτελεί μόνο οριζόντια σάρωση (horizontal) με ελάχιστη γωνία -180° και μέγιστη +180°, άρα είναι δισδιάστατος με πλήρη σάρωση 360 μοιρών. Τέλος ορίζεται το εύρος της μέτρησης του από 12 εκατοστά έως 12 μέτρα με ανάλυση απόστασης 1 εκατοστό. Τα δεδομένα που δημοσιεύει περιέχουν πίνακα 360 αποστάσεων σε μέτρα (ranges[ ]), γεωμετρία σάρωσης (angle_min, angle_max, angle_increment) και τεχνικά όρια (range_min, range_max).

```
<sensor name="scan_omni" type="gpu_lidar">
    <pose>0 0 0.3 0 0 0 </pose>
    <update_rate>10</update_rate>
    <topic>scan</topic>
    <lidar>
        <scan>
            <horizontal>
                <samples>360</samples>
                <resolution>1</resolution>
                <min_angle>-3.14159</min_angle>
                <max_angle>3.14159</max_angle>
            </horizontal>
        </scan>
        <range>
            <min>0.12</min>
            <max>12.0</max>
            <resolution>0.01</resolution>
        </range>
    </lidar>
</sensor>
```

Εικόνα 3-12 LiDAR Sensor

- Ultrasonic

Σε αυτό τον εικονικό αισθητήρα προσομοιώνεται ένας υπερηχητικός αισθητήρας χρησιμοποιώντας έναν laser με συγκεκριμένες ρυθμίσεις οι οποίες φαίνονται στην Εικόνα 3-13. Οι παράμετροι που κάνουν τον gpu-lidar να συμπεριφέρεται ως ultrasonic είναι η στενή γωνία σάρωσης, η οποία έχει όρια από -0.01 έως +0.01 ακτίνια δηλαδή σχεδόν 1 μοίρα και το ένα δείγμα ανά σάρωση που λαμβάνεται με την εκπομπή ακτίνας. Άλλες ρυθμίσεις που είναι όμοιες με αυτές του ultrasonic είναι η ελάχιστη απόσταση μέτρησης στα 2 εκατοστά, η μέγιστη στα 2 μέτρα και η ανάλυση στο 1 εκατοστό. Τέλος οι ακτίνες του LiDAR είναι ορατές στην προσομοίωση ως λεπτές γραμμές και τα δεδομένα της σάρωσης δημοσιεύονται στο topic /ultrasonic με ρυθμό ανανέωσης 10 Hz ενώ το ύψος του αισθητήρα από το έδαφος είναι όπως το πραγματικό ρομπότ 6 εκατοστά.

```
<link name="ultrasonic">
    <pose>0.018 0 0.06 0 0 0</pose>
    <sensor name="ultrasonic" type="gpu_lidar">
        <pose>0 0 0 0 0 0</pose>
        <visualize>true</visualize>
        <always_on>true</always_on>
        <update_rate>10</update_rate>
        <topic>ultrasonic</topic>
        <ray>
            <scan>
                <horizontal>
                    <samples>1</samples>
                    <resolution>1</resolution>
                    <min_angle>-0.01</min_angle>
                    <max_angle>0.01</max_angle>
                </horizontal>
            </scan>
            <range>
                <min>0.02</min>
                <max>2.0</max>
                <resolution>0.01</resolution>
            </range>
        </ray>
    </sensor>
```

Εικόνα 3-13 Ultrasonic Sensor

- Κάμερα

Ο αισθητήρας της κάμερας στη προσομοίωση του FOSSBot είναι μια κάμερα RGB-D, η οποία αποτελείται όπως φαίνεται στην Εικόνα 3-14 από μια RGB κάμερα και μια κάμερα που μετρά απόσταση ανά pixel. Η κάμερα RGB μεταδίδει 10 καρέ ανά δευτερόλεπτο στο topic /color_image με ανάλυση 848x480 δηλαδή περίπου 0.4 MegaPixels. Η κάμερα αυτή αποδίδει αντικείμενα που βρίσκονται σε απόσταση από 10 εκατοστά έως 50 μέτρα και έχει οριζόντιο πεδίο όρασης 1.2 ακτίνια δηλαδή περίπου 70 μοίρες. Η κάμερα Depth μεταδίδει 6 καρέ ανά δευτερόλεπτο στο topic /depth_image και κάθε pixel είναι μια μονής ακρίβειας δεκαδική τιμή (float) που αντιπροσωπεύει την απόσταση σε μέτρα. Αυτή η κάμερα έχει εύρος μικρότερο από την RGB που κυμαίνεται από 1 εκατοστό έως 10 μέτρα. Τέλος και οι δύο κάμερες έχουν παραμέτρους εσωτερικής διάταξης οι οποίες ορίζουν το μαθηματικό μοντέλο του φακού και αντιστοιχούν σε πραγματικές παραμέτρους κάμερας.

```
<sensor name="color" type="camera">
    <update_rate>10</update_rate>
    <always_on>0</always_on>
    <topic>color_image</topic>
    <camera>
        <horizontal_fov>1.20427718</horizontal_fov>
        <image>
            <width>848</width>
            <height>480</height>
        </image>
        <clip>
            <near>0.01</near>
            <far>50</far>
        </clip>
        <lens>
            <intrinsics>
                <fx>615.9603271484375</fx>
                <fy>616.227294921875</fy>
                <cx>419.83026123046875</cx>
                <cy>245.1431427001953</cy>
                <s>0</s>
            </intrinsics>
        </lens>
    </camera>
</sensor>

<sensor name="depth" type="depth_camera">
    <update_rate>6</update_rate>
    <always_on>0</always_on>
    <topic>depth_image</topic>
    <camera>
        <horizontal_fov>1.51843645</horizontal_fov>
        <image>
            <width>848</width>
            <height>480</height>
            <format>R_FLOAT32</format>
        </image>
        <clip>
            <near>0.01</near>
            <far>10</far>
        </clip>
        <lens>
            <intrinsics>
                <fx>421.61578369140625</fx>
                <fy>421.61578369140625</fy>
                <cx>422.2854309082031</cx>
                <cy>236.57237243652344</cy>
                <s>0</s>
            </intrinsics>
        </lens>
    </camera>
</sensor>
```

Εικόνα 3-14 Camera

3.3.2.2.3 Τροχοί

Οι τροχοί του εικονικού FOSSBot είναι τρεις, ο αριστερός και ο δεξιός μπροστά τροχός και ο πίσω πλαστικός τροχός. Οι μπροστά τροχοί μεταδίδουν την κίνηση του ρομπότ και έχουν τις ίδιες ρυθμίσεις με μόνη διαφορά την θέση τους σε σχέση με το κύριο σώμα του ρομπότ. Στις παραμέτρους τους ορίζεται καταρχήν η θέση τους και ο προσανατολισμός τους, ο οποίος είναι τέτοιος (-1.57 rad) ώστε ο τροχός να είναι κάθετα αντί οριζόντια. Επίσης ορίζεται η ροπή

αδράνειας γύρω από του άξονες x,y,z του τροχού όπου φαίνεται στην Εικόνα 3-15 ότι στον y είναι μεγαλύτερη καθώς αυτός είναι ο άξονας περιστροφής του τροχού.

```
<link name="wheel_right">
    <inertial>
        <pose>-0.01 -0.105 0.026 -1.57 0 0</pose>
        <inertia>
            <ixx>1.8158194e-03</ixx>
            <ixy>-9.3392e-12</ixy>
            <ixz>1.04909e-11</ixz>
            <iyy>3.2922126e-03</iyy>
            <iyz>5.75694e-11</iyz>
            <izz>1.8158194e-03</izz>
        </inertia>
        <mass>2.8498940e-02</mass>
    </inertial>
```

Εικόνα 3-15 Right Wheel

Η επόμενη ομάδα ρυθμίσεων φαίνεται στην Εικόνα 3-16 και αφορά την γεωμετρία σύγκρουσης, σε αυτήν οι πρώτες παράμετροι ορίζουν την ακτίνα του τροχού σε 3,3 εκατοστά και το πλάτος σε 1,8 εκατοστά. Οι επόμενες παράμετροι είναι αυτοί της επαφής (surface), οι οποίες είναι σημαντικές για να υπάρχει ρεαλιστική κίνηση. Σε αυτές υπάρχει ο συντελεστής τριβής που έχει οριστεί σε 1,2 προσφέροντας ρεαλιστικό και καλό κράτημα και η ολίσθηση που έχει οριστεί σε 0 τις εκατό επιτρέποντας στον τροχό να κινείται χωρίς να ολισθαίνει κατά την επιτάχυνση ή την στροφή. Η επόμενη ομάδα είναι αυτή της επαφής όπου καθορίζονται η στιβαρότητα του αντικειμένου (kp), η απορρόφηση των κραδασμών (kd), η διείσδυση στο έδαφος και η μέγιστη ταχύτητα διείσδυσης η οποία ελέγχει πόσο απότομη είναι η σύγκρουση. Τέλος στις ρυθμίσεις της οπτικής αναπαράστασης υπάρχει η διεύθυνση του αρχείου stl το οποίο είναι το τρισδιάστατο μοντέλο του τροχού και ο ορισμός του χρώματος, του μεγέθους, της θέσης και του προσανατολισμού του μοντέλου αυτού.

```
<collision name="wheel_right_collision">
    <pose>-0.01 -0.105 0.026 -1.57 0 0</pose>
    <geometry>
        <cylinder>
            <radius>0.033</radius>
            <length>0.018</length>
        </cylinder>
    </geometry>
    <surface>
        <friction>
            <ode>
                <mu>1.2</mu>
                <mu2>1.2</mu2>
                <fdir1>0 0 0</fdir1>
                <slip1>0</slip1>
                <slip2>0</slip2>
            </ode>
        </friction>
        <contact>
            <ode>
                <soft_cfm>0</soft_cfm>
                <soft_erp>0.2</soft_erp>
                <kp>1e+5</kp>
                <kd>1</kd>
                <max_vel>0.01</max_vel>
                <min_depth>0.001</min_depth>
            </ode>
        </contact>
    </surface>
</collision>

<visual name="wheel_right_visual">
    <pose>-0.01 -0.105 0.026 0 0 0</pose>
    <geometry>
        <mesh>
            <uri>model://fossbot/meshes/right_wheel.stl</uri>
            <scale>1 1 1</scale>
        </mesh>
    </geometry>
    <material>
        <ambient>1.0 0.8 0.0 1.0</ambient>
        <diffuse>1.0 0.8 0.0 1.0</diffuse>
        <specular>0.1 0.1 0.1 1.0</specular>
    </material>
</visual>
```

Εικόνα 3-16 Right Wheel 2

Ο πίσω πλαστικός τροχός (caster wheel) είναι ένας μικρός βοηθητικός τροχός που είναι ελεύθερος να περιστρέφεται σε όλες τις κατευθύνσεις χωρίς να μεταδίδει κίνηση αλλά χρησιμοποιείται μόνο για να είναι σταθερό το ρομπότ. Η θέση του όπως φαίνεται στην Εικόνα 3-17 είναι στο πίσω μέρος του ρομπότ, έχει μικρή μάζα και σχεδιάζεται ως μια μικρή σφαίρα ακτίνας 0.5 εκατοστών. Ο τροχός αυτός δεν έχει τριβή αλλά μόνο στοιχεία επαφής ίδια με τους άλλους τροχούς όπως και το χρώμα.

```
<link name="caster_back">
    <pose>0 0 0 0 0 0</pose>
    <inertial>
        <mass>0.005</mass>
        <inertia>
            <ixx>0.001</ixx>
            <ixy>0.000</ixy>
            <ixz>0.000</ixz>
            <iyy>0.001</iyy>
            <iyz>0.000</iyz>
            <izz>0.001</izz>
        </inertia>
    </inertial>
    <collision name="collision">
        <geometry>
            <sphere>
                <radius>0.005000</radius>
            </sphere>
        </geometry>
        <surface>
            <contact>
                <ode>
                    <soft_cfm>0</soft_cfm>
                    <soft_erp>0.2</soft_erp>
                    <kp>1e+5</kp>
                    <kd>1</kd>
                    <max_vel>0.01</max_vel>
                    <min_depth>0.001</min_depth>
                </ode>
            </contact>
        </surface>
    </collision>

</link>
```

Εικόνα 3-17 Caster Back

3.3.2.2.4 Αρθρώσεις

Σε αυτό το τμήμα του μοντέλου ορίζονται οι αρθρώσεις (joints) που συνδέουν τα μέρη του ρομπότ μεταξύ τους και φαίνονται στην Εικόνα 3-18. Η πρώτη άρθρωση συνδέει το εικονικό σημείο του εδάφους (base_footprint), το οποίο χρησιμοποιείται για την πλοήγηση και τον εντοπισμό της θέσης του ρομπότ, με το κύριο σώμα του ρομπότ ανυψωμένο κατά 1 εκατοστό. Στις επόμενες δύο αρθρώσεις τύπου revolute, συνδέονται οι μπροστινοί τροχοί με το σώμα ρομπότ και ορίζεται ως άξονας περιστροφής τους ο άξονας z. Στην συνέχεια η άρθρωση που συνδέει τον πίσω τροχό με το σώμα του ρομπότ είναι τύπου ball και έτσι καθορίζονται τρεις άξονες περιστροφής επιτρέποντας στον τροχό να περιστρέφεται ελεύθερα όπως ένας πραγματικός σφαιρικός τροχός. Τέλος στις σταθερές αρθρώσεις των αισθητήρων IMU, LiDAR, Ultrasonic και κάμερας με το κύριο μέρος του ρομπότ καθορίζεται η θέση και ο προσανατολισμός του αισθητήρα σε σχέση με τον κεντρικό άξονα του ρομπότ.

```
<joint name="base_joint" type="fixed">
    <parent>base_footprint</parent>
    <child>base_link</child>
    <pose>0.0 0.0 0.010 0 0 0</pose>
</joint>

<joint name="wheel_left_joint" type="revolute">
    <parent>base_link</parent>
    <child>wheel_left</child>
    <pose>-0.01 0.105 0.026 -1.57 0 0</pose>
    <axis>
        <xyz>0 0 1</xyz>
    </axis>
</joint>

<joint name="wheel_right_joint" type="revolute">
    <parent>base_link</parent>
    <child>wheel_right</child>
    <pose>-0.01 -0.105 0.026 -1.57 0 0</pose>
    <axis>
        <xyz>0 0 1</xyz>
    </axis>
</joint>

<joint name="caster_back_joint" type="ball">
    <parent>base_link</parent>
    <child>caster_back</child>
```

```
510  <joint name="imu_joint" type="fixed">
511      <parent>base_link</parent>
512      <child>imu</child>
513      <pose>0 0 0.1 0 0 0</pose>
514  </joint>
515
516  <joint name="lidar_joint" type="fixed">
517      <parent>base_link</parent>
518      <child>lidar</child>
519      <pose>-0.032 0 0.171 0 0 0</pose>
520  </joint>
521
522  <joint name="ultrasonic_joint" type="fixed">
523      <parent>base_link</parent>
524      <child>ultrasonic</child>
525      <pose>0.018 0 0.02 0 0 0</pose>
526  </joint>
527
528  <joint name="camera_joint" type="fixed">
529      <parent>base_link</parent>
530      <child>camera_front</child>
531      <pose>0.040 -0.000 0.110 0 0 0</pose>
532  </joint>
533
534
535
```

Εικόνα 3-18 Joints

3.3.2.2.5 Πρόσθετα – plugins

Τα πρόσθετα είναι πακέτα λογισμικού που δίνουν λειτουργικότητα στην προσομοίωση και ενσωματώνονται στο Ignition Gazebo για να ελέγχουν και να αλληλεπιδρούν με το μοντέλο. Τα πρόσθετα του FOSSBot φαίνονται στην Εικόνα 3-19 και το κάθε ένα δημοσιεύει ένα topic όπως ορίζεται στον Πίνακας 3-1.

- Πρόσθετο αισθητήρα IMU

Το πρόσθετο για τον IMU αισθητήρα δημιουργεί τις μετρήσεις που δίνει ένας πραγματικός αισθητήρας IMU δηλαδή την γωνιακή ταχύτητα και την γραμμική επιτάχυνση με βάση την πραγματική κίνηση του ρομπότ στο περιβάλλον προσομοίωσης. Σε αυτό το πρόσθετο δεν έχουν οριστεί παράμετροι και επομένως χρησιμοποιεί τις αρχικές ρυθμίσεις του αισθητήρα IMU και δημοσιεύει στο topic /imu.

- Πρόσθετο Αισθητήρων

Το ακόλουθο πρόσθετο ενεργοποιεί και διαχειρίζεται όλους τους υπόλοιπους αισθητήρες του μοντέλου όπως τον LiDAR, τον ultrasonic και την κάμερα με topics /scan, /ultrasonic

/color_image και /depth_image αντίστοιχα. Επίσης ορίζει ως μηχανή γραφικών την Ogre2, η οποία είναι η σύγχρονη μηχανή του Gazebo και αποδίδει τα οπτικά δεδομένα σε αισθητήρες όπως η κάμερα.

- Πρόσθετο DiffDrive

Το πρόσθετο DiffDrive είναι το κύριο σύστημα ελέγχου του ρομπότ και το κάνει να κινείται όταν λαμβάνονται, στο ROS topic /cmd_vel, μηνύματα τύπου geometry_msgs/Twist τα οποία περιέχουν γραμμική και γωνιακή ταχύτητα. Η κίνηση επιτυγχάνεται υπολογίζοντας τις απαιτούμενες ταχύτητες περιστροφής για κάθε τροχό χρησιμοποιώντας τις κινηματικές εξισώσεις διαφορικού οχήματος και εφαρμόζοντας στη συνέχεια την αναγκαία ροπή στους τροχούς αυτούς μέσω των συνδέσμων wheel_left_joint και wheel_right_joint. Τέλος το πρόσθετο αυτό δημοσιεύει την οδομετρία (odometry) του οχήματος με βάση την περιστροφή των τροχών του. Η πρώτη παράμετρος που υπάρχει στο DiffDrive ορίζει την απόσταση μεταξύ των κέντρων των δύο τροχών (wheel_separation) σε 21 εκατοστά και η δεύτερη την ακτίνα των τροχών σε 3 εκατοστά , η οποία είναι σκόπιμα μικρότερη από την ακτίνα που ορίστηκε στις παραμέτρους σύγκρουσης ώστε να αντισταθμιστεί η συμπίεση του ελαστικού και η ολίσθηση. Η επόμενη παράμετρος (odom_publish_frequency) ορίζει την συχνότητα της δημοσίευσης της οδομετρίας σε 50 φορές το δευτερόλεπτο και η τελευταία το εισερχόμενο topic για τον έλεγχο της ταχύτητας σε /cmd_vel. Το πρόσθετο αυτό δημοσιεύει στο /model/fossbot/odometry την εκτίμηση βάσει των τροχών της θέσης του ρομπότ και στο /model/fossbot/tf τους μετασχηματισμούς του συστήματος αναφοράς.

```
<plugin filename="ignition-gazebo-imu-system" name="ignition::gazebo::systems::Imu"/>

<plugin filename="ignition-gazebo-sensors-system" name="ignition::gazebo::systems::Sensors">
    <render_engine>ogre2</render_engine>
</plugin>

<!-- TOPICS: /model/fossbot/odometry , /model/fossbot/tf -->
<plugin filename="ignition-gazebo-diff-drive-system" name="ignition::gazebo::systems::DiffDrive">
    <left_joint>wheel_left_joint</left_joint>
    <right_joint>wheel_right_joint</right_joint>
    <wheel_separation>0.210</wheel_separation>
    <wheel_radius>0.03</wheel_radius>
    <odom_publish_frequency>50</odom_publish_frequency>
    <topic>cmd_vel</topic>
</plugin>
```

Εικόνα 3-19 Plugins

Πίνακας 3-1 FOSSBot Gazebo Topics

Topic	Τύπος Μηνύματος	Προορισμός
/cmd_vel	geometry_msgs/Twist	Έλεγχος ταχύτητας του ρομπότ
/scan	sensor_msgs/LaserScan	2D LiDAR σαρώσεις
/color_image	sensor_msgs/msg/Image	RGB εικόνα
/depth_image	sensor_msgs/msg/Image	Βάθος εικόνας
/imu	sensor_msgs/msg/Imu	IMU μετρήσεις
/ultrasonic	sensor_msgs/msg/LaserScan	Υπερηχητικές μετρήσεις
/model/fossbot/odometry	nav_msgs/msg/Odometry	Οδομετρία
/model/fossbot/tf	tf2_msgs/TFMessage	Μετασχηματισμοί συστημάτων

3.3.3 Περιβάλλον Προσομοίωσης

Το μοντέλο προσομοίωσης του ρομποτικού οχήματος FOSSBot που περιγράψαμε παραπάνω χρειάζεται ένα περιβάλλον ώστε να τρέξει η προσομοίωση του. Αυτό το περιβάλλον ονομάζεται world και τα βασικά του στοιχεία είναι οι βασικές ρυθμίσεις φυσικής, φωτισμού και περιβάλλοντος οι οποίες φαίνονται στην Εικόνα 3-20, το έδαφος το οποίο φαίνεται στην Εικόνα 3-21 και οι απαραίτητοι τοίχοι και εμπόδια, οι οποίοι έχουν την μορφή όπως στην Εικόνα 3-22. Στις ρυθμίσεις φυσικής υπάρχει καταρχήν το βήμα χρόνου, το οποίο είναι 3 ms και ορίζει το χρονικό διάστημα μεταξύ δύο διαδοχικών υπολογισμών της φυσικής μηχανής. Στην συνέχεια καθορίζεται η προσομοίωση να τρέχει σε πραγματικό χρόνο (real_time_factor) με ρυθμό ανανέωσης (real_time_update_rate) 1000 φορές το δευτερόλεπτο και προσθέτονται τα plugins του συστήματος, τα οποία διαχειρίζονται την φυσική προσομοίωση, επιτρέπουν τον χειροκίνητο έλεγχο μέσω GUI, δημοσιεύουν πληροφορίες της σκηνής για οπτικοποίηση και διαχειρίζονται τις επαφές. Άλλες ρυθμίσεις είναι η βαρύτητα (gravity), η οποία έχει οριστεί σε 9.8 m/s^2 και το μαγνητικό πεδίο (magnetic_field) το οποίο προσομοιώνει την πυξίδα και το μαγνητόμετρο.

Όσον αφορά το οπτικό περιβάλλον υπάρχει ο φωτισμός στον οποίο ορίζεται η κατεύθυνση προέλευσης του (direction) και η ύπαρξη σκιών που αυξάνει τον ρεαλισμό, η

σύσταση της ατμόσφαιρας (atmosphere type) και οι ρυθμίσεις της σκηνής που περιλαμβάνουν το φόντο, το φωτισμό και τις σκιές. Μετά τις ρυθμίσεις ορίζονται τα αντικείμενα του περιβάλλοντος τα οποία είναι το έδαφος (ground plane) στο οποίο κινείται το ρομπότ, τα σύνορα (border) τα οποία δημιουργούν ένα κλειστό δωμάτιο στο οποίο περιορίζεται το ρομπότ ώστε να μην πέσει στα άκρα του εδάφους και τα εμπόδια (obstacles) τα οποία δημιουργούν ένα κόσμο κατάλληλο για την ρεαλιστική δοκιμή της πλοήγησης σε στενούς διαδρόμους, των αλγορίθμων SLAM και της αποφυγής εμποδίων.

```
<world name='default'>
    <physics name='1ms' type='ignored'>
        <max_step_size>0.003</max_step_size>
        <real_time_factor>1</real_time_factor>
        <real_time_update_rate>1000</real_time_update_rate>
    </physics>
    <plugin name='ignition::gazebo::systems::Physics' filename='ignition-gazebo-physics-system' />
    <plugin name='ignition::gazebo::systems::UserCommands' filename='ignition-gazebo-user-commands-system' />
    <plugin name='ignition::gazebo::systems::SceneBroadcaster' filename='ignition-gazebo-scene-broadcaster-system' />
    <plugin name='ignition::gazebo::systems::Contact' filename='ignition-gazebo-contact-system' />
    <light name='sun' type='directional'>
        <cast_shadows>1</cast_shadows>
        <pose>0 0 10 0 -0 0</pose>
        <diffuse>0.8 0.8 0.8 1</diffuse>
        <specular>0.2 0.2 0.2 1</specular>
        <attenuation>
            <range>1000</range>
            <constant>0.90000000000000002</constant>
            <linear>0.01</linear>
            <quadratic>0.001</quadratic>
        </attenuation>
        <direction>-0.5 0.1 -0.9</direction>
        <spot>
            <inner_angle>0</inner_angle>
            <outer_angle>0</outer_angle>
            <falloff>0</falloff>
        </spot>
    </light>
    <gravity>0 0 -9.8</gravity>
    <magnetic_field>6e-06 2.3e-05 -4.2e-05</magnetic_field>
    <atmosphere type='adiabatic' />
    <scene>
        <ambient>0.4 0.4 0.4 1</ambient>
        <background>0.7 0.7 0.7 1</background>
        <shadows>1</shadows>
    </scene>
    <model name='ground plane'>

    <model name='border'>

    <model name='obstacles'>
</world>
```

Εικόνα 3-20 Αρχείο Obstacles World

```
<model name='ground_plane'>
    <static>1</static>
    <link name='link'>
        <collision name='collision'>
            <geometry>
                <plane>
                    <normal>0 0 1</normal>
                    <size>100 100</size>
                </plane>
            </geometry>
            <surface>
                <friction>
                    <ode />
                </friction>
                <bounce />
                <contact />
            </surface>
        </collision>
        <visual name='visual'>
            <geometry>
                <plane>
                    <normal>0 0 1</normal>
                    <size>100 100</size>
                </plane>
            </geometry>
            <material>
                <ambient>0.8 0.8 0.8 1</ambient>
                <diffuse>0.8 0.8 0.8 1</diffuse>
                <specular>0.8 0.8 0.8 1</specular>
            </material>
            <plugin name='__default__' filename='__default__' />
        </visual>
    </link>
```

Εικόνα 3-21 Έδαφος

```
<link name='wall1'>
    <pose>-10.005 0 0 0 0 0</pose>
    <static>true</static>
    <collision name='wall1_collision'>
        <geometry>
            <box>
                <size>0.01 20 2</size>
            </box>
        </geometry>
    </collision>
    <visual name='wall1_visual'>
        <geometry>
            <box>
                <size>0.01 20 2</size>
            </box>
        </geometry>
        <material>
            <ambient>0.1 0.1 0.1 1</ambient>
            <diffuse>0 0.01 0.05 1</diffuse>
            <specular>0 0.01 0.05 1</specular>
        </material>
    </visual>
</link>
```

Εικόνα 3-22 Εμπόδια

3.3.4 Εκκίνηση Προσομοίωσης

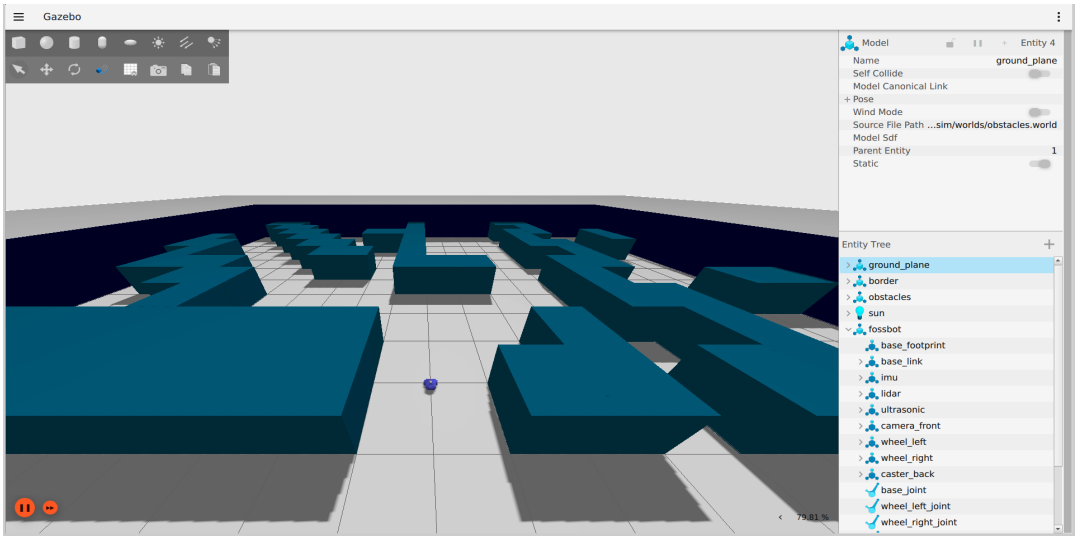
Μετά την δημιουργία του μοντέλου του FOSSBot, του περιβάλλοντος της προσομοίωσης και την εισαγωγή των τρισδιάστατων οπτικών αρχείων του ρομπότ στον φάκελο meshes, επόμενο βήμα είναι η εκτέλεση των εντολών για την εκκίνηση της προσομοίωσης. Οι εντολές με την σειρά που πρέπει να εκτελεστούν είναι οι εξής:

I. **export LIBGL_ALWAYS_SOFTWARE=1,** για την οποία αναφερθήκαμε στο 3.2.3.1

II. **ign gazebo -r ~/fossbot_sim/worlds/obstacles.world,** η οποία φορτώνει τον κόσμο obstacles που αναλύσαμε παραπάνω.

III. **ign service -s /world/default/create **
**--reqtype ignition.msgs.EntityFactory **
**--reptype ignition.msgs.Boolean **
**--timeout 5000 **
--req 'sdf_filename: "/home/vgan/fossbot_sim/models/fossbot/model.sdf"'
, η οποία δημιουργεί (spawn) το μοντέλο του FOSSBot μέσα στον κόσμο που έχουμε φορτώσει χρησιμοποιώντας το service “create” που ανήκει στον κόσμο default και στη συνέχεια επιστρέφει true ή false ανάλογα με το αν η δημιουργία ήταν επιτυχής μέσα σε 5 δευτερόλεπτα.

Μετά την εκτέλεση των εντολών I,II,III έχει ανοίξει το Gazebo με το περιβάλλον obstacles και το FOSSBot μέσα σε αυτό όπως φαίνεται στην Εικόνα 3-23 Gazebo Simulation.



Εικόνα 3-23 Gazebo Simulation

3.3.4.1 Γέφυρες – Bridges

Όπως αναφέρεται στην ενότητα 3.2.3.2 για να επικοινωνούν μεταξύ τους ο προσομοιωτής Gazebo και το λειτουργικό σύστημα ROS2 απαιτείται η ύπαρξη γεφυρών (bridges). Οι bridges μεταφέρουν τα topic από το Gazebo στο ROS2 και το αντίστροφο με την δυνατότητα μετονομασίας τους (remapping). Η σύνταξη της εντολής φαίνεται παρακάτω και σε αυτή διακρίνεται ότι χρησιμοποιείται είτε το σύμβολο "[" το οποίο ορίζει την κατεύθυνση της bridge από το Gazebo προς το ROS2 είτε το "]" το οποίο ορίζει την κατεύθυνση από το ROS2 προς το Gazebo. Επίσης στην παράμετρο –r γίνεται το remapping και στην –p ρυθμίζεται το ROS2 να χρησιμοποιεί το ρολόι προσομοίωσης και όχι το πραγματικό ρολόι.

ros2 run ros_gz_bridge parameter_bridge
/ignition_topic@<ros_message_type>[<ignition_message_type>
**/<ignition_topic>@<ros_message_type>]<ignition_message_type> **
**--ros-args **
**-r <ignition_topic>:/=<ros_topic> **
-p use_sim_time:=true

Η εντολή που χρησιμοποιείται ώστε να εκτελεστούν οι γέφυρες για τα topics του μοντέλου μας που φαίνονται στον Πίνακας 3-1 είναι η παρακάτω.

IV. **ros2 run ros_gz_bridge parameter_bridge **
**/imu@sensor_msgs/msg/Imu[ignition.msgs.IMU **
**/ultrasonic@sensor_msgs/msg/LaserScan[ignition.msgs.LaserScan **
**/color_image@sensor_msgs/msg/Image[ignition.msgs.Image **
**/depth_image@sensor_msgs/msg/Image[ignition.msgs.Image **
**/camera_info@sensor_msgs/msg/CameraInfo[ignition.msgs.CameraInfo **
**/cmd_vel@geometry_msgs/msg/Twist]ignition.msgs.Twist **
**/scan@sensor_msgs/msg/LaserScan[ignition.msgs.LaserScan **
**/model/fossbot/odometry@nav_msgs/msg/Odometry[ignition.msgs.Odometry **
**/model/fossbot/tf@tf2_msgs/msg/TFMessage[ignition.msgs.Pose_V **
**/clock@rosgraph_msgs/msg/Clock[ignition.msgs.Clock **
**--ros-args **
**-r /model/fossbot/tf:=/tf **
**-r /model/fossbot/odometry:=/odom **

-p use_sim_time:=true

3.3.4.2 Εκδότης Κατάστασης Ρομπότ – Robot State Publisher

Ο εκδότης της κατάστασης του ρομπότ (robot_state_publisher) είναι απαραίτητος ώστε να γνωρίζει το ROS2 τις συνδέσεις που υπάρχουν μεταξύ των εξαρτημάτων του ρομπότ και ώστε να οριστεί το tf-tree του μοντέλου. Ο εκδότης διαβάζει ένα αρχείο URDF, το οποίο όπως φαίνεται στην Εικόνα 3-24 περιέχει τις θέσεις των αρθρώσεων (/joint_states) του ρομπότ και βάσει αυτών υπολογίζει και δημοσιεύει τους TF μετασχηματισμούς μεταξύ όλων των συνδέσμων.

```
<?xml version="1.0"?>
<robot name="fossbot">
  <link name="base_footprint"/>
  <link name="base_link"/>

  <link name="imu"/>
  <link name="imu/imu"/>
  <link name="lidar"/>
  <link name="lidar/scan_omni"/>
  <link name="ultrasonic"/>
  <link name="ultrasonic/ultrasonic"/>
  <link name="camera_front"/>
  <link name="camera_front/color"/>
  <link name="camera_front/depth"/>
  <link name="caster_back"/>
  <link name="wheel_left"/>
  <link name="wheel_right"/>

  <!-- JOINTS -->
  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </joint>

  <joint name="imu_joint" type="fixed">
    <parent link="base_link"/>
    <child link="imu"/>
    <origin xyz="-0.05 0 0.05" rpy="0 0 0"/>
  </joint>
```

Εικόνα 3-24 Αρχείο Urdf

Η εντολή που εκκινεί το κόμβο robot_state_publisher για το μοντέλο μας είναι:

V. **ros2 run robot_state_publisher robot_state_publisher **

**--ros-args **

**-p robot_description:="$(cat /home/vgan/fossbot_sim/models/fossbot/model.urdf)" **

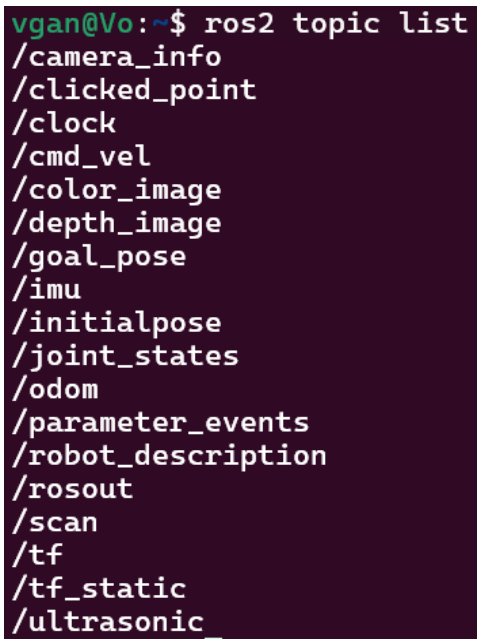
**-p use_sim_time:=true **

**-p publish_frequency:=30.0 **

-p frame_prefix:="fossbot/"

3.3.4.3 Topics ROS2

Μετά την εκτέλεση των παραπάνω εντολών τα topics που υπάρχουν στο ROS2 φαίνονται στην Εικόνα 3-25.



Εικόνα 3-25 Λίστα ROS2 Topic

3.3.4.4 RViz2

Για την εκκίνηση του Rviz2 χρησιμοποιούμε την εντολή «**rviz2**» και όταν ανοίξει πρέπει να ορίσουμε τις ρυθμίσεις της οπτικοποίησης ώστε να εμφανίζονται σε αυτή όλα τα στοιχεία του ρομπότ όπως είναι οι άξονες των εξαρτημάτων του, η κατεύθυνση του βάσει της οδομετρίας, η εικόνα της κάμερας του και τα σημεία των αντικειμένων που καταγράφει το Laser. Οι ρυθμίσεις για το μοντέλο του FOSSBot είναι οι εξής:

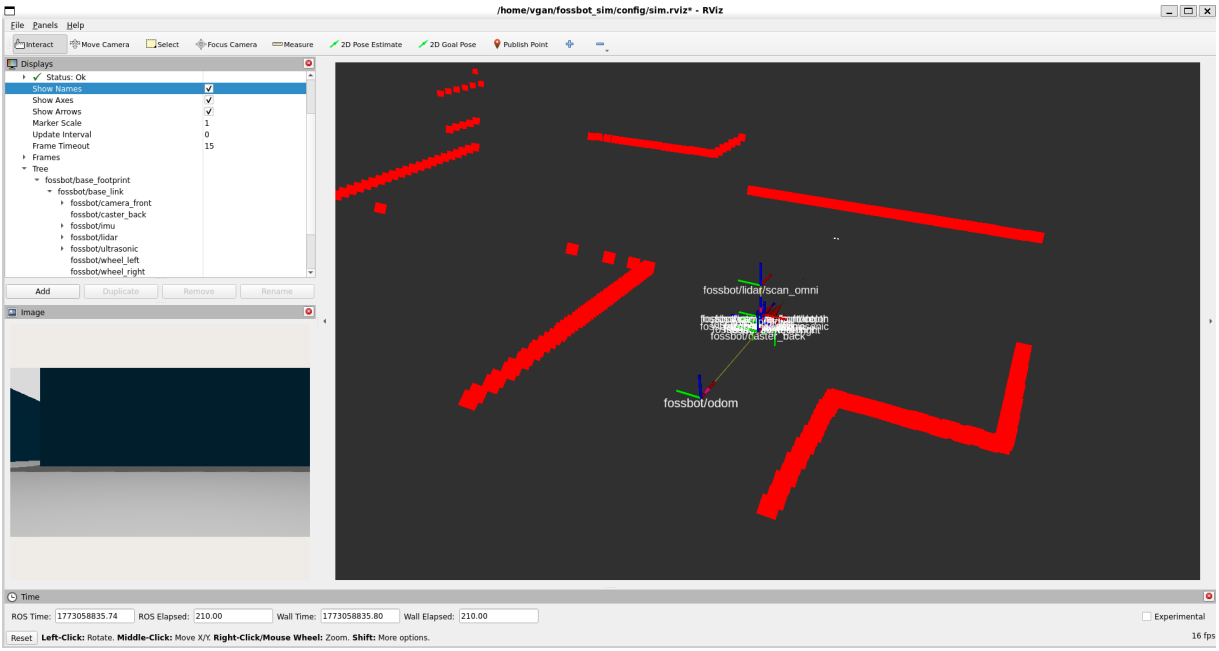
- Global Options -> Fixed Frame : fossbot/base_link (όταν εκτελούμε SLAM είναι /map)
- Add LaserScan -> Topic : /scan
- Add TF
- Add Odometry -> Topic: /odom
- Add Image -> Topic: /color_image
- Add Map -> Topic: /map

  Μετά την εισαγωγή των ρυθμίσεων τις αποθηκεύουμε στο μενού File -> Save Config As με Φάκελο: \home\vgan\fossbot_sim\config και File name: fossbot.rviz.

Μετά την αποθήκευση των ρυθμίσεων η εκκίνηση του Rviz2 γίνεται με την εντολή:

VI. **rviz2 -d ~/fossbot_sim/config/fossbot.rviz**

Μετά την εκτέλεση των εντολών IV,V,VI,VII έχει ανοίξει το RViz2 με απεικόνιση των joints του FOSSBot όπως φαίνεται στην Εικόνα 3-26.



Εικόνα 3-26 RViz2

3.3.5 Δημιουργία bash script εκκίνησης

Για την αυτοματοποίηση των διαδικασιών της εκκίνησης της προσομοίωσης δημιουργήθηκε ένα αρχείο SH, το οποίο είναι ένα σενάριο Bash με εντολές που εκτελούνται στο τερματικό. Στο αρχείο αυτό με όνομα SIM εισήχθησαν οι εντολές από την I έως την VI όπως φαίνεται στην Εικόνα 3-27.

Στην συνέχεια το αρχείο αυτό γίνεται εκτελέσιμο με την εντολή:

chmod +x ~/fossbot_sim/scripts/sim.sh,

Επίσης με σκοπό να γίνεται η εκτέλεση του με την εντολή **sim** μπήκε συντόμευση στο αρχείο ρυθμίσεων του κελύφους BASH με τις εντολές:

echo "alias sim='./fossbot_sim/scripts/sim.sh'" >> ~/.bashrc

source ~/.bashrc

```
#!/bin/bash
 export LIBGL_ALWAYS_SOFTWARE=1
 ign gazebo -r ~/fossbot_sim/worlds/obstacles.world &
 sleep 5

 ign service -s /world/default/create \
   --reqtype ignition.msgs.EntityFactory \
   --reptype ignition.msgs.Boolean \
   --timeout 5000 \
   --req 'sdf_filename: "/home/vgan/fossbot_sim/models/fossbot/model.sdf"' &

ros2 run ros_gz_bridge parameter_bridge \
  /imu@sensor_msgs/msg/Imu[ignition.msgs.IMU \
  /ultrasonic@sensor_msgs/msg/LaserScan[ignition.msgs.LaserScan \
  /color_image@sensor_msgs/msg/Image[ignition.msgs.Image \
  /depth_image@sensor_msgs/msg/Image[ignition.msgs.Image \
  /camera_info@sensor_msgs/msg/CameraInfo[ignition.msgs.CameraInfo \
  /cmd_vel@geometry_msgs/msg/Twist]ignition.msgs.Twist \
  /scan@sensor_msgs/msg/LaserScan[ignition.msgs.LaserScan \
  /model/fossbot/odometry@nav_msgs/msg/Odometry[ignition.msgs.Odometry \
  /clock@rosgraph_msgs/msg/Clock[ignition.msgs.Clock \
  /model/fossbot/tf@tf2_msgs/msg/TFMessage[ignition.msgs.Pose_V \
  --ros-args \
  -r /model/fossbot/odometry:=/odom \
  -r /model/fossbot/tf:=/tf \
  -p use_sim_time:=true &

ros2 run robot_state_publisher robot_state_publisher \
  --ros-args \
  -p robot_description:="$(cat /home/vgan/fossbot_sim/models/fossbot/model.urdf)" \
  -p use_sim_time:=true \
  -p publish_frequency:=30.0 \
  -p frame_prefix:="fossbot/" &

 rviz2 -d ~/fossbot_sim/config/fossbot.rviz &

 sleep 20
 echo "Για τηλεχειρισμό εκτέλεσε την εντολή δώσε τιμές στο x για κίνηση και στο y για στροφή"
 echo "ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0}, angular: {z: 0.0}}""

wait
```

Εικόνα 3-27 Αρχείο sim.sh

3.4. Σενάρια Λειτουργίας

3.4.1 Τηλεχειρισμός – Teleoperation

Η χειροκίνητη κίνηση του ρομπότ στο περιβάλλον γίνεται με τον έλεγχο της ταχύτητας του (Velocity) μέσω μηνυμάτων τύπου twist. Η εντολή για την αποστολή τέτοιων μηνυμάτων είναι της μορφής:

ros2 topic pub /cmd_vel geometry_msgs/msg/Twist

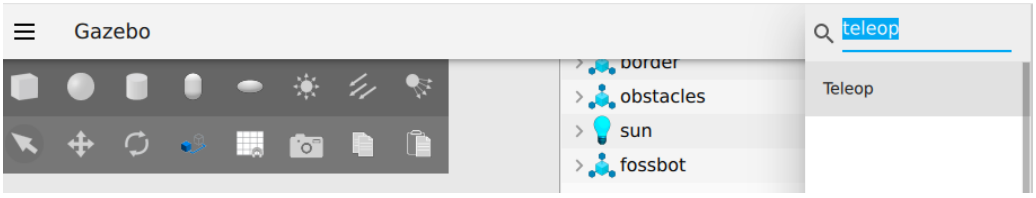
"{linear: {x: Επιθυμητή Ταχύτητα Μπροστά},

angular: {z: Επιθυμητή Ταχύτητα Στροφής}}"

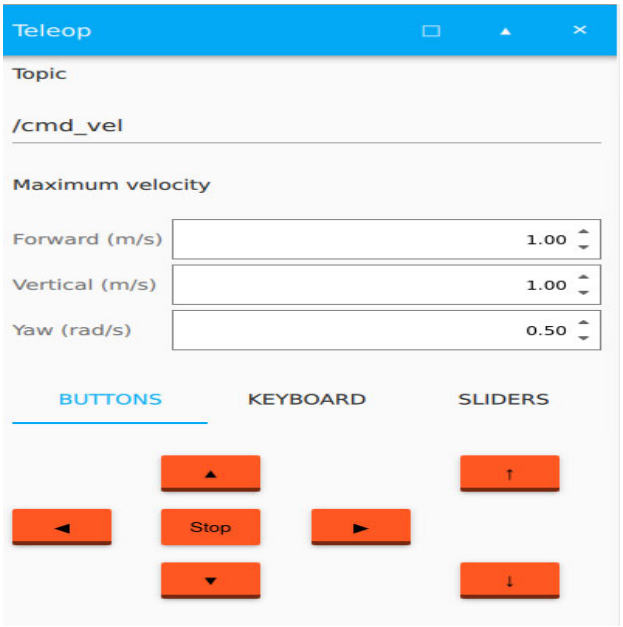
Παραδείγματα τέτοιων εντολών είναι:

- Κίνηση μπροστά: **ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.2}, angular: {z: 0.0}}"**
- Κίνηση προς τα πίσω: **ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: -0.2}, angular: {z: 0.0}}"**
- Στροφή αριστερά: **ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0}, angular: {z: 0.5}}"**
- Στροφή δεξιά: **ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0}, angular: {z: -0.5}}**

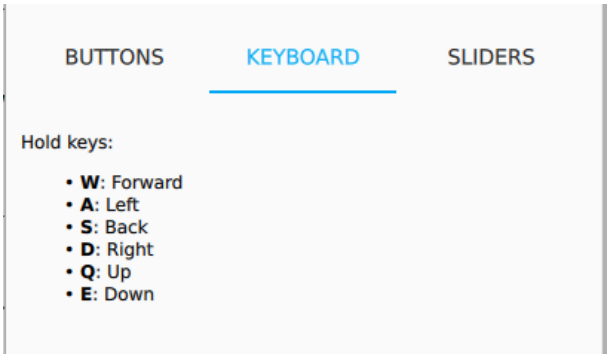
Ο έλεγχος της κίνησης του ρομπότ μπορεί να γίνει και με το πρόσθετο του Gazebo **"Teleop"**, το οποίο μπορεί να ενεργοποιηθεί από το μενού που βρίσκεται πάνω και δεξιά στο γραφικό περιβάλλον του Gazebo όπως φαίνεται στην Εικόνα 3-28. Στο teleop υπάρχει η επιλογή αποστολής μηνυμάτων πλοήγησης με βέλη, τα οποία φαίνονται στην Εικόνα 3-29, και με πλήκτρα που φαίνονται στην Εικόνα 3-30, σε συγκεκριμένο topic και με ρυθμιζόμενη ταχύτητα κίνησης και περιστροφής.



Εικόνα 3-28 Ενεργοποίηση Πρόσθετου Teleop



Εικόνα 3-29 Teleop με βέλη



Εικόνα 3-30 Teleop με πλήκτρα

3.4.2 Υλοποίηση Χαρτογράφησης με Laser

Η χαρτογράφηση του περιβάλλοντος, η οποία σύμφωνα με το 2.4.5.1 είναι η δεύτερη διαδικασία του SLAM, έγινε χρησιμοποιώντας το εγκατεστημένο ROS2 πακέτο "SLAM Toolbox" και τα δεδομένα από τον αισθητήρα LiDAR. Το πακέτο αυτό χρησιμοποιεί τον αλγόριθμο χαρτογράφησης Karto SLAM, ο οποίος όπως προκύπτει από το 2.4.5.2 είναι ο πιο αποδοτικός αλγόριθμος. Η χαρτογράφηση γίνεται με ασύγχρονο τρόπο λειτουργίας, με την φόρτωση των παραμέτρων από το αρχείο τύπου YAML "slam.yaml" στον φάκελο "config" του πακέτου "fossbot_sim" και με την εντολή:

VII. **ros2 run slam_toolbox async_slam_toolbox_node **
**--ros-args **
**--params-file ~/fossbot_sim/config/slam.yaml **
--remap /clock:=/clock

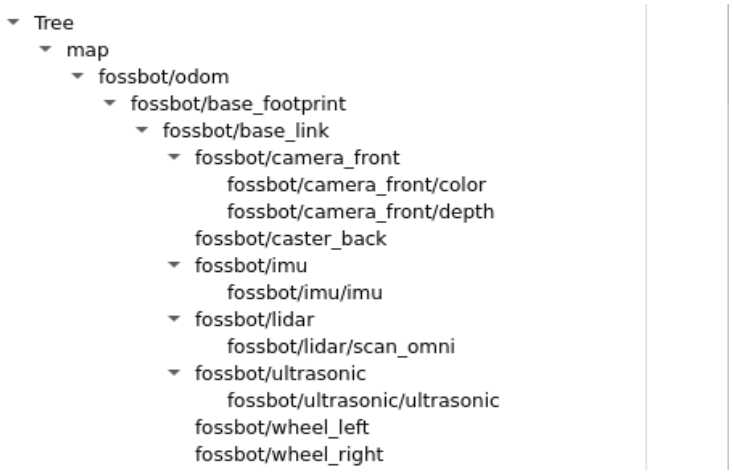
Κατά την εκτέλεση της χαρτογράφησης υπάρχουν στο ROS2 εκτός των topics από την ενότητα 3.3.4.3 και τα topics που φαίνονται στην Εικόνα 3-31.

```
/map
/map_metadata
/map_updates
/slam_toolbox/feedback
/slam_toolbox/graph_visualization
/slam_toolbox/scan_visualization
/slam_toolbox/update
```

Εικόνα 3-31 SLAM Topics

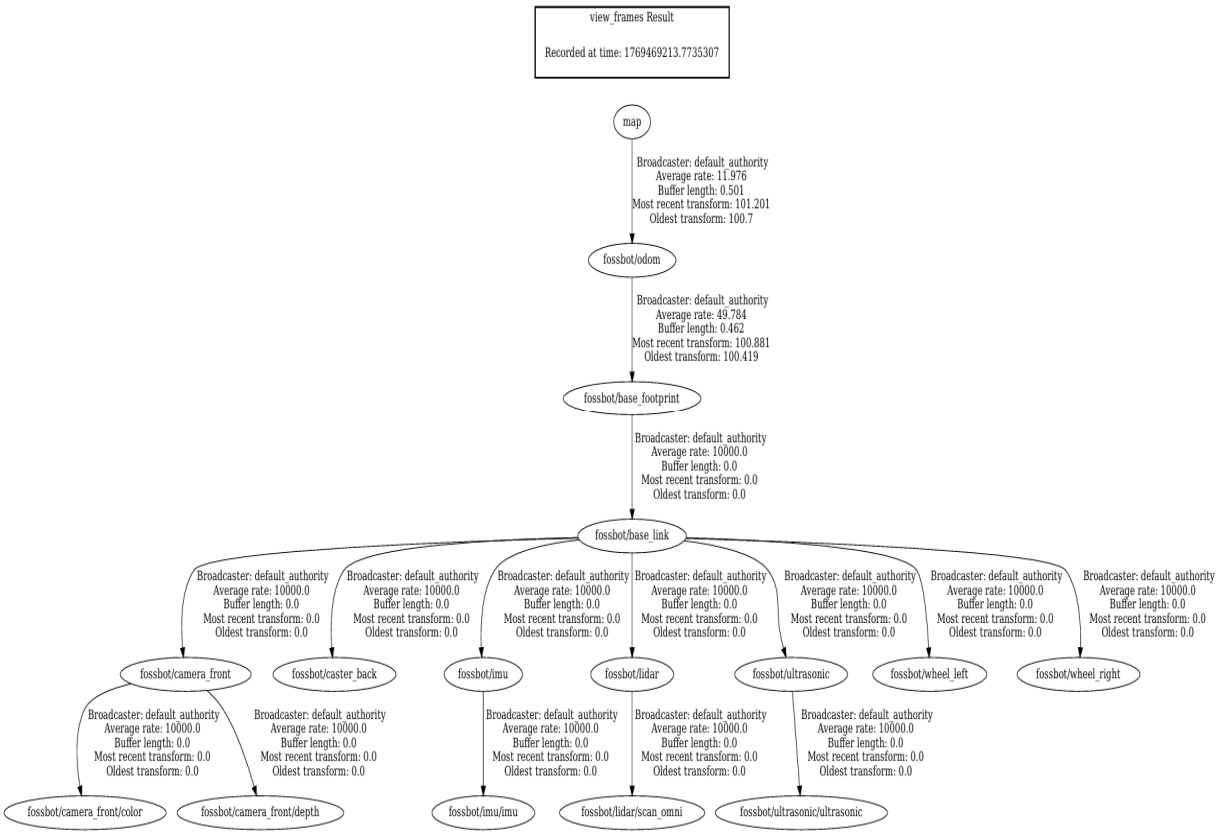
3.4.2.1 TF TREE

Επίσης στις ρυθμίσεις του RViz2 πρέπει να γίνει η αλλαγή του Global Options -> Fixed Frame από fossbot/base_link σε /map και τότε έχουμε το tf_tree το οποίο φαίνεται στην Εικόνα 3-32 από τις ρυθμίσεις του Rviz2.



Εικόνα 3-32 TF Tree RViz

Το συγκεκριμένο tf_tree φαίνεται επίσης στην Εικόνα 3-33 από το αρχείο τύπου pdf που δημιουργήθηκε στον γονικό φάκελο του χρήστη με την εντολή **ros2 run tf2_tools view_frames.**



Εικόνα 3-33 TF Tree PDF

3.4.2.2 Αρχείο YAML

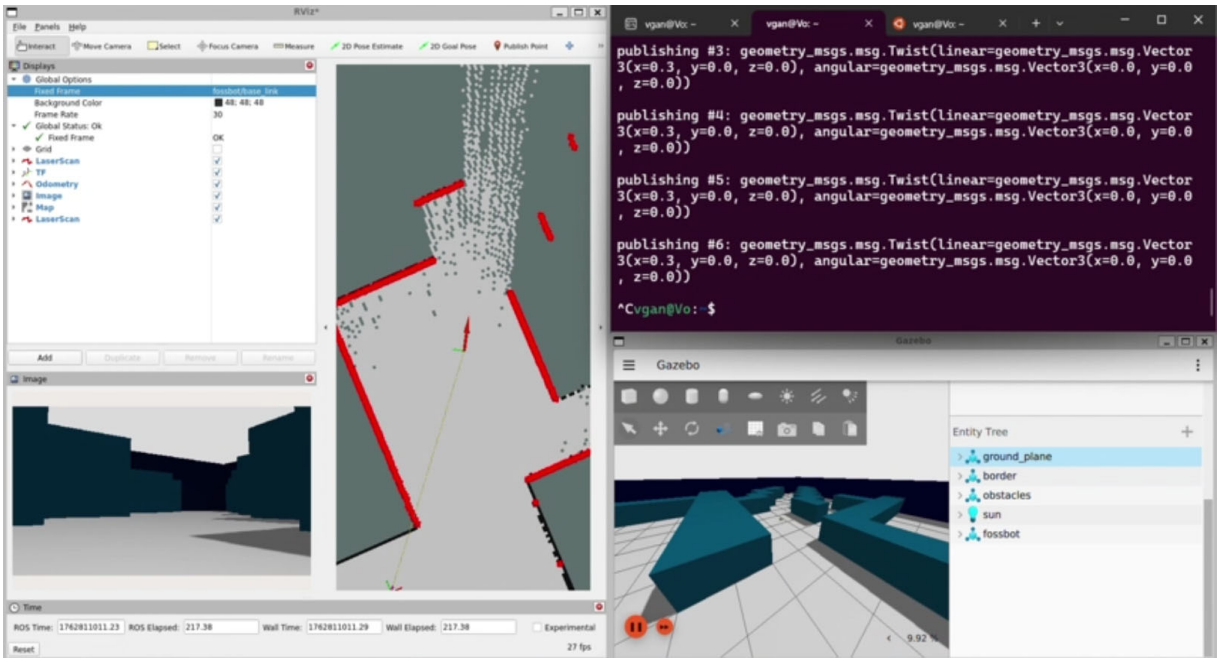
Το αρχείο slam.yaml περιέχει τους απαραίτητες παραμέτρους για την εκτέλεση χαρτογράφησης σύμφωνα με το [88], οι οποίοι είναι οι παρακάτω:

- Ορίζεται η χαρτογράφηση να χρησιμοποίει το ρολόι προσομοίωσης από το Gazebo με την παράμετρο : **use_sim_time: true**
- Τα ID των απαραίτητων frames για τον καθορισμό του tf_tree
  base_frame: "fossbot/base_footprint"
  odom_frame: "fossbot/odom"
  map_frame: "map"

- Τα ονόματα των topics: **scan_topic: "/scan"**
  odom_topic: "/odom"
  map_topic: "/map"
- **scan_queue_size: 10** # Το μέγεθος της ουράς της επεξεργασίας των σαρώσεων
- **throttle_scans: 1** # Ο ορισμός της χρησιμοποίησης κάθε σάρωσης
- **min_laser_range: 0.12** # Η ελάχιστη εμβέλεια του ραντάρ
- **max_laser_range: 12.0** # Η μέγιστη εμβέλεια του ραντάρ
- **resolution: 0.05** # Η ανάλυση του χάρτη σε εκατοστά ανά pixel
- **map_update_interval: 0.1** # Ο χρόνος ενημέρωσης του χάρτη σε δευτερόλεπτα
- **tf_buffer_duration: 5.0** # Ο χρόνος που κρατάει ιστορικό σε δευτερόλεπτα
- **transform_publish_period: 0.05** # Κάθε πότε θα δημοσιεύει TF σε δευτ/πτα
- **scan_buffer_size: 10** # Πόσες σαρώσεις να κρατάει για αντιστοίχιση
- **scan_buffer_maximum_scan_distance: 20.0** # Το όριο της διαφοράς των σαρώσεων για την σύγκριση τους
- **minimum_travel_distance: 0.1** # μέτρα
- **minimum_travel_heading: 0.1 #** rad
- **qos_overrides:**
  /scan:
  reliability: reliable # Αξιόπιστο για να μην χάνονται σαρώσεις
  durability: volatile # Χρησιμοποιούνται μόνο νέα δεδομένα
  depth: 50 # Πόσα μηνύματα να αποθηκεύει στην ουρά

3.4.2.3 Διαδικασία Χαρτογράφησης και αποτέλεσμα

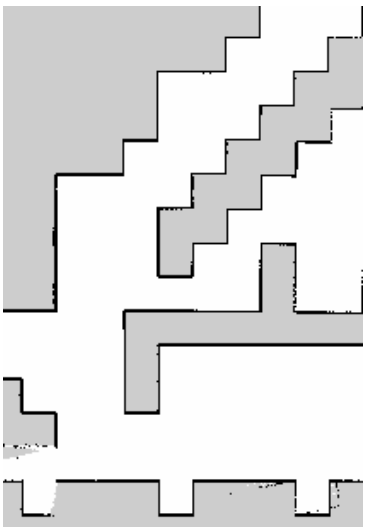
Κατά την εκτέλεση του κόμβου που πραγματοποιεί την χαρτογράφηση κινείται το FOSSBot μέσα στο περιβάλλον με τηλεχειρισμό από τον χρήστη και δημιουργείται ο χάρτης που φαίνεται στην Εικόνα 3-34.



Εικόνα 3-34 Εκτέλεση Χαρτογράφησης

Όταν έχει γίνει χαρτογράφηση στον επιθυμητό χώρο, πραγματοποιείται η αποθήκευση του χάρτη που φαίνεται στην Εικόνα 3-35 με την παρακάτω εντολή και αποθηκεύεται ο χάρτης στον φάκελο fossbot_sim/maps.

VIII. ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "{name: {data: './fossbot_sim/maps/map'}}"



Εικόνα 3-35 Χάρτης από δεδομένα LiDAR

3.4.3 Υλοποίηση Χαρτογράφησης με Laser και κάμερα

Για την πραγματοποίηση της χαρτογράφησης του περιβάλλοντος με την ταυτόχρονη χρήση των αισθητήρων της κάμερας και του LiDAR χρησιμοποιήθηκε το πακέτο RTAB-Map. Το RTAB-Map δημιουργεί δισδιάστατους και τρισδιάστατους χάρτες μέσω της σύντηξης των δεδομένων των παραπάνω αισθητήρων. Η πραγματοποίηση της χαρτογράφησης γίνεται με την παρακάτω εντολή, με την οποία εκκινείται το αρχείο τύπου launch με δύο παραμέτρους που ορίζουν την χρήση του χρόνου προσομοίωσης και τον τρόπο λειτουργίας του πακέτου RTAB-Map σε SLAM.

IX. **ros2 launch ~/fossbot_sim/launch/rtabmap.py **
**use_sim_time:=true **
localization:=false

3.4.3.1 Αρχείο εκκίνησης - rtabmap.py

Το αρχείο rtabmap.py είναι ένα αρχείο εκκίνησης γραμμένο σε γλώσσα Python το οποίο ρυθμίζει την εκκίνηση του συγκεκριμένου πακέτου. Στο αρχείο αυτό καλείται η συνάρτηση generate_launch_description, η οποία φαίνεται στην Εικόνα 3-36 και δημιουργεί και επιστρέφει ένα αντικείμενο LaunchDescription. Στην αρχή της συνάρτησης ορίζονται οι μεταβλητές των παραμέτρων της εκκίνησης, οι οποίες είναι η χρήση ή όχι του χρόνου προσομοίωσης και ο τρόπος λειτουργίας του RTAB-Map. Στην συνέχεια ορίζονται οι ρυθμίσεις του πακέτου, από τις οποίες οι κυριότερες είναι:

- Η διαδρομή αποθήκευσης των δεδομένων (database_path) του RTAB-Map σε μορφή βάσης δεδομένων με περιεχόμενα τον χάρτη, τα χαρακτηριστικά και τα γραφήματα (/home/vgan/fossbot_sim/maps/rtabmap.db).
- Τα frames που θα χρησιμοποιηθούν από το πακέτο όπως το frame_id.
- Η ενεργοποίηση ή όχι της λήψης δεδομένων συγκεκριμένους αισθητήρες όπως η κάμερα βάθους (subscribe_depth) και το LiDAR (subscribe_scan).
- Grid/CellSize: Το μέγεθος του κελιού.
- Grid/MaxGroundHeight: Το μέγιστο ύψος του εδάφους.
- Grid/MaxObstacleHeight: Το μέγιστο ύψος των εμποδίων.
- Grid/RangeMax: Η εμβέλεια του αισθητήρα.
- Rtabmap/DetectionRate: Ο ρυθμός ανανέωσης σε Hz.

- Kp/MaxFeatures: Ο μέγιστος αριθμός χαρακτηριστικών ανά εικόνα.
- Reg/Force3DoF: Η κίνηση του ρομπότ σε 2D κίνηση.
- Grid/3D: Η δημιουργία 3D ή 2D χάρτη.
- Οι αντιστοιχίες των ονομάτων των topics του RTAB-Map με τα αντίστοιχα του FOSSBot.

```
def generate_launch_description():

    use_sim_time = LaunchConfiguration('use_sim_time')
    localization = LaunchConfiguration('localization')

    database_path = '/home/vgan/fossbot_sim/maps/rtabmap.db'

    parameters={
          'frame_id': 'fossbot/base_footprint',
          'odom_frame_id': 'fossbot/odom',
          'map_frame_id': 'map',
          'use_sim_time': use_sim_time,
          'subscribe_depth': True,
          'subscribe_scan': True,
          'use_action_for_goal': False,
          'Reg/Force3DoF': 'true',
          'Grid/RayTracing': 'true',
          'Grid/3D': 'false',
          'Grid/RangeMax': '3.0',
          'Grid/FromDepth': 'true',
          'Grid/CellSize': '0.05',
          'Grid/NormalsSegmentation': 'false',
          'Grid/MaxGroundHeight': '0.02',
          'Grid/MaxObstacleHeight': '0.4',
          'Optimizer/GravitySigma': '0',
          'Kp/MaxFeatures': '1000',
          'Vis/FeatureType': '6',
          'Mem/IncrementalMemory': 'true',
          'Mem/InitWMWithAllNodes': 'false',
          'Rtabmap/DetectionRate': '3.0',
          'Rtabmap/CreateIntermediateNodes': 'false',
          'database_path': database_path
    }

    remappings=[
          ('rgb/image', '/color_image'),
          ('rgb/camera_info', '/camera_info'),
          ('depth/image', '/depth_image'),
          ('scan', '/scan'),
          ('odom', '/odom')
    ]
```

Εικόνα 3-36 Συνάρτηση generate_launch_description

Στο αντικείμενο LaunchDescription όπως φαίνεται στην Εικόνα 3-37 υπάρχει η εντολή για την εκκίνηση του κόμβου του RATB-Map σε λειτουργία χαρτογράφησης η οποία εκτελείται αν η μεταβλητή localization είναι false. Στην εντολή αυτή ορίζονται το πακέτο που περιέχει τον κόμβο, το εκτελέσιμο αρχείο, η χρησιμοποίηση των προαναφερθέντων ρυθμίσεων και η διαγραφή ή μη του προϋπάρχοντος χάρτη (arguments=['-d']). Αν η μεταβλητή localization είναι true τότε εκτελείται η εντολή που ξεκινάει τον κόμβο του πακέτου RATB-Map σε localization mode αλλάζοντας δύο παραμέτρους από τους αρχικούς. Τέλος και στις δύο περιπτώσεις

εκκινείται ο κόμβος οπτικοποίησης ο οποίος δείχνει διάφορα γραφικά όπως τον τρέχοντα χάρτη, τη θέση του ρομπότ και τα τρισδιάστατα νέφη των σημείων που ορίζουν τα εμπόδια.

```
return LaunchDescription([

    DeclareLaunchArgument(
        'use_sim_time', default_value='true',
        description='Use simulation (Gazebo) clock if true'),

    DeclareLaunchArgument(
        'localization', default_value='false',
        description='Launch in localization mode.'),

    Node(
        condition=UnlessCondition(localization),
        package='rtabmap_slam', executable='rtabmap', output='screen',
        parameters=[parameters],
        remappings=remappings,
        arguments=['-d']),

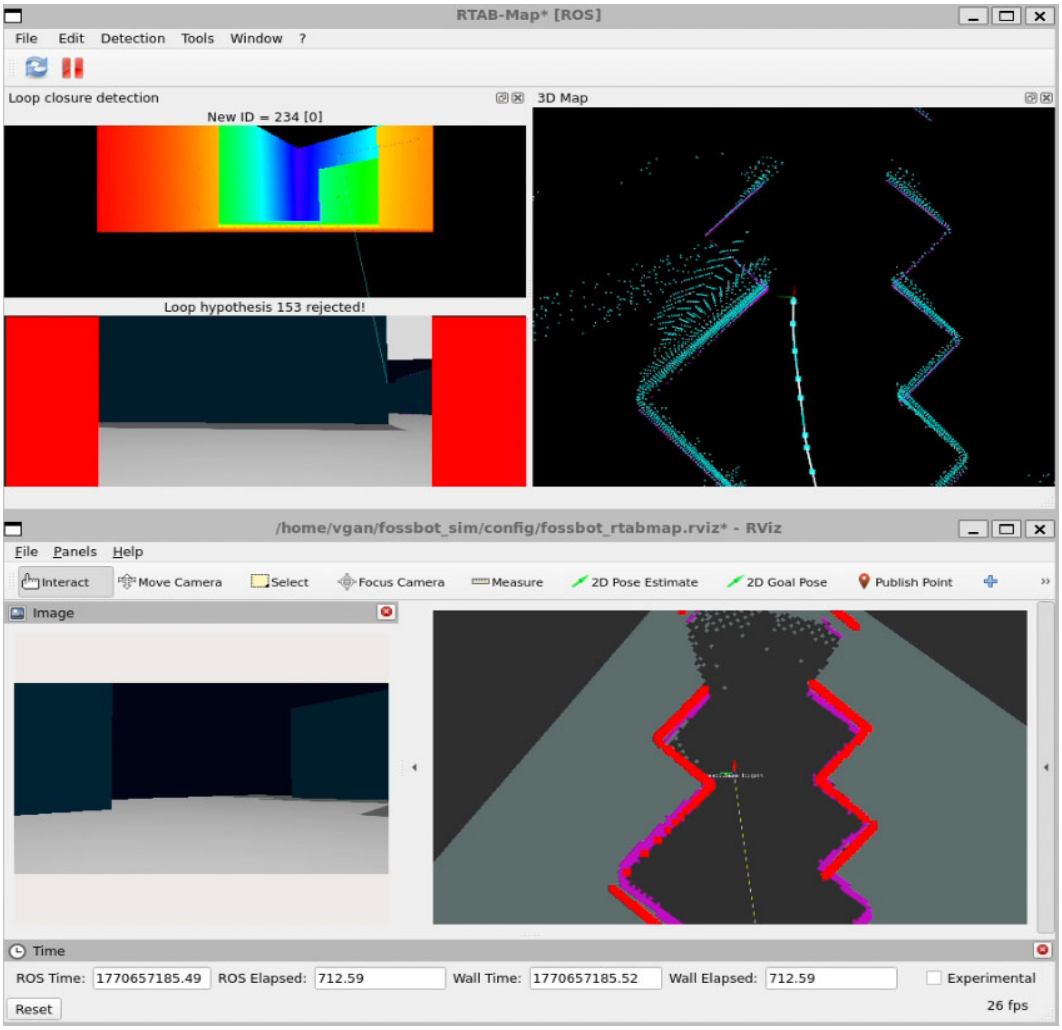
    Node(
        condition=IfCondition(localization),
        package='rtabmap_slam', executable='rtabmap', output='screen',
        parameters=[parameters,
          {'Mem/IncrementalMemory': 'False',
           'Mem/InitWMWithAllNodes': 'True'}],
        remappings=remappings),

    Node(
        package='rtabmap_viz', executable='rtabmap_viz', output='screen',
        parameters=[parameters],
        remappings=remappings),
```

Εικόνα 3-37 Αντικείμενο LaunchDescription

3.4.3.2 Διαδικασία Χαρτογράφησης και αποτέλεσμα

Κατά την εκτέλεση της εντολής IX και έχοντας εκτελέσει πρώτα το bash script εκκίνησης sim.sh (3.3.5), εμφανίζονται στην οθόνη όπως φαίνεται στην Εικόνα 3-38 η εφαρμογή Gazebo με το περιβάλλον προσομοίωσης και το ρομπότ, η εφαρμογή Rviz2 και η εφαρμογή RTAB-Map. Στην συνέχεια με χειροκίνητη πλοήγηση στον χώρο χαρτογραφείται το επιθυμητό περιβάλλον, όπως φαίνεται στην παρακάτω εικόνα, με την εφαρμογή RTAB-Map που εμφανίζει την κάμερα βάθους, την εικόνα πρόβλεψης που απορρίφθηκε, τον 3D χάρτη από την κάμερα και την διαδρομή που ακολουθεί το ρομπότ και με την εφαρμογή Rviz2 όπου εμφανίζονται ο χάρτης του περιβάλλοντος που δημιουργείται τη συγκεκριμένη στιγμή, τα δεδομένα του LiDAR (κόκκινη γραμμή) και η εικόνα της κάμερας χρώματος.



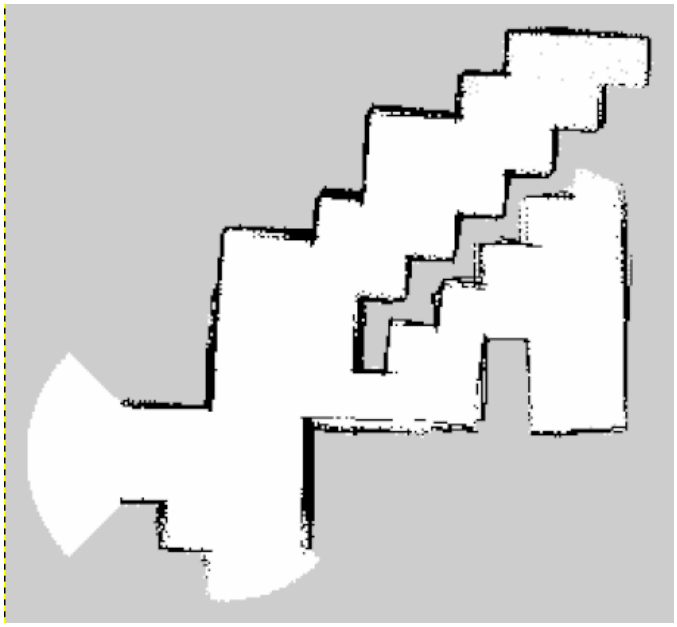
Εικόνα 3-38 RTAB-Map, RViz2 στην χαρτογράφηση

Μετά την ολοκλήρωση της διαδικασίας οι πληροφορίες και ο χάρτης από την χαρτογράφηση αποθηκεύονται στο αρχείο **rtabmap.db** στον φάκελο **/home/vgan/fossbot_sim/maps** και λόγω του ότι κατά την εκκίνηση νέας χαρτογράφησης θα διαγραφεί αυτό το αρχείο πρέπει να γίνει η αντιγραφή του με την παρακάτω εντολή, η οποία προσθέτει στο όνομα την ημερομηνία και την ώρα της αποθήκευσης.

cp ~/fossbot_sim/maps/rtabmap.db ~/fossbot_sim/maps/rtabmap_map_$(date +%Y%m%d_%H%M%S).db

Στη συνέχεια ανοίγοντας το αρχείο αυτό με την εφαρμογή rtabmap-databaseViewer εκτελώντας την παρακάτω εντολή μπορούμε να δούμε τον χάρτη που φαίνεται στην Εικόνα 3-39 και να τον εξάγουμε στην μορφή που επιθυμούμε, όπως τα αρχεία rtabmap.pgm και rtabmap.yaml που θα χρησιμοποιήσουμε για την αυτόνομη πλοήγηση.

rtabmap-databaseViewer ~/fossbot_sim/maps/rtabmap.db



Εικόνα 3-39 Χάρτης από δεδομένα LiDAR και κάμερας

3.4.4 Υλοποίηση Χαρτογράφησης με κάμερα

Για την πραγματοποίηση της χαρτογράφησης του περιβάλλοντος με την χρήση μόνο της κάμερας έγιναν αλλαγές στο αρχείο rtabmap.py από την ενότητα 3.4.3.1, αποθηκεύθηκαν στο αρχείο rtabcam.py και έγινε η εκτέλεση της παρακάτω εντολής. Οι αλλαγές που έγιναν στο αρχείο rtabmap.py φαίνονται στην Εικόνα 3-40 και είναι η αλλαγή του subscribe_scan σε false ώστε να απενεργοποιηθεί το LiDAR, η αφαίρεση του remapping του scan ('scan', '/scan') καθώς δεν χρειάζεται πλέον, το NormalsSegmentation σε true για να διαχωριστεί το έδαφος από τα εμπόδια και το MaxGroundHeight αυξήθηκε ώστε να μην ερμηνεύεται το πάτωμα ως εμπόδιο. Επίσης προστέθηκαν και άλλες παράμετροι ώστε να γίνει καλύτερη επεξεργασία της εικόνας βάθους και να εξαλειφθεί ο θόρυβος των δεδομένων, όπως είναι η

FlatObstacleDetectedMaxGroundAngle που είναι η μέγιστη γωνία για τα επίπεδα εμπόδια, η ProjMaxGroundAngle η οποία είναι η μέγιστη γωνία προβολής του εδάφους, η NoiseFilteringRadius που είναι η ακτίνα φιλτραρίσματος των εμποδίων και η NoiseFilteringMinNeighbors όπου είναι οι ελάχιστοι γείτονες για να είναι ένα σημείο έγκυρο.

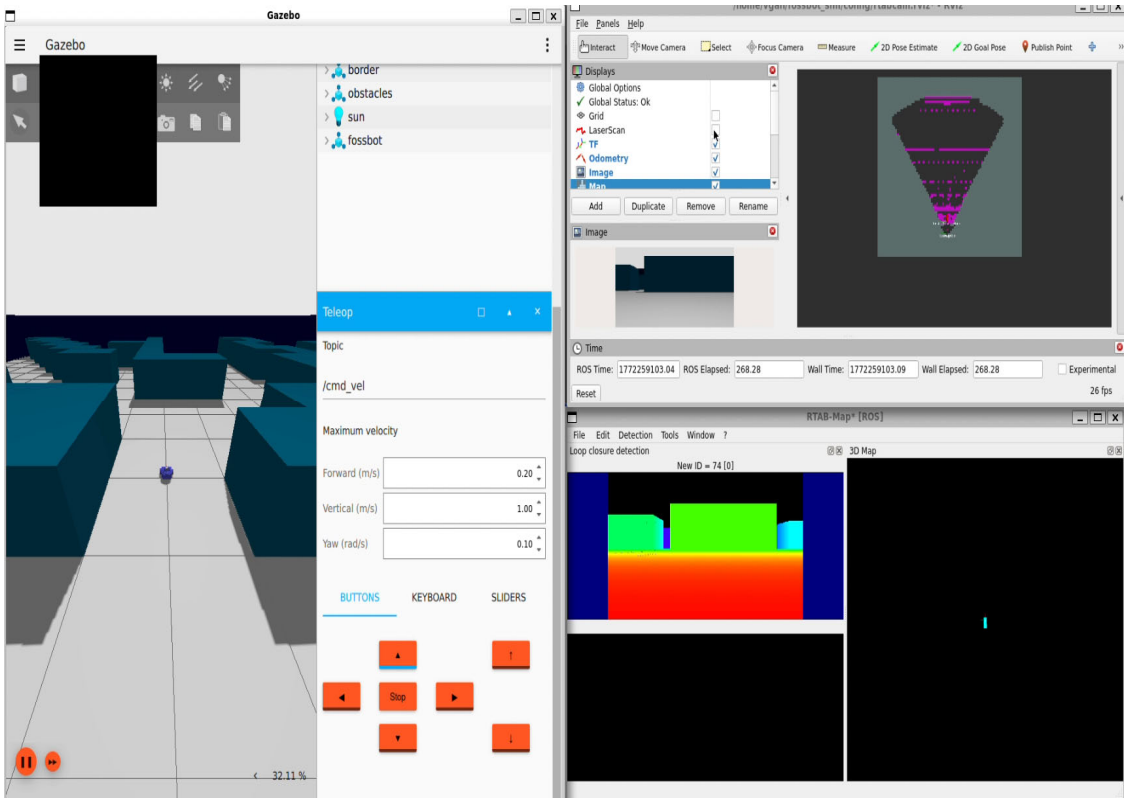
```
parameters={
      'frame_id':'fossbot/base_footprint',
      'map_frame_id': 'map',
      'use_sim_time':use_sim_time,
      'subscribe_depth':True,
      'subscribe_scan':False,
      'use_action_for_goal':False,
      'Reg/Force3DoF':'true',
      'Grid/RayTracing':'true',
      'Grid/3D':'false',
      'Grid/RangeMax':'3.0',
      'Grid/FromDepth':'true',
      'Grid/CellSize': '0.05',
      'Grid/NormalsSegmentation':'true',
      'Grid/MaxGroundHeight':'0.20',
      'proj_max_ground_angle': '45',
      'Grid/MaxObstacleHeight':'2.0',
      'Optimizer/GravitySigma':'0',
      'Grid/NoiseFilteringRadius': '0.05',
      'Grid/NoiseFilteringMinNeighbors': '2',
      'Grid/FlatObstacleDetectedMaxGroundAngle': '15',
      'Grid/ProjMaxGroundAngle': '15',
      'database_path': database_path
}
```

Εικόνα 3-40 Παράμετροι αρχείου rtabcam.py

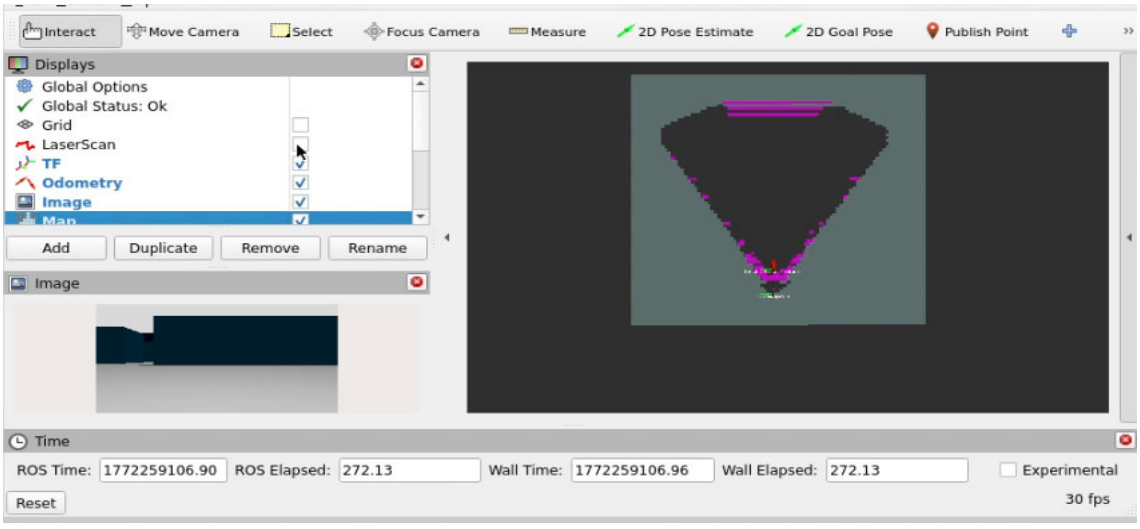
Η εντολή εκκίνησης της χαρτογράφησης είναι η παρακάτω:

X. **ros2 launch ~/fossbot_sim/launch/rtabcam.py **
**use_sim_time:=true **
localization:=false

Κατά την χαρτογράφηση με κάμερα έχουμε τα παράθυρα που φαίνονται στην Εικόνα 3-41, όπου διακρίνεται ότι στην αρχή της χαρτογράφησης εμφανίζεται το έδαφος ως εμπόδιο μέχρι να το αναγνωρίσει ο αλγόριθμος και στην συνέχεια διορθώνει αυτά τα λάθος σημεία όπως φαίνεται στην Εικόνα 3-42.

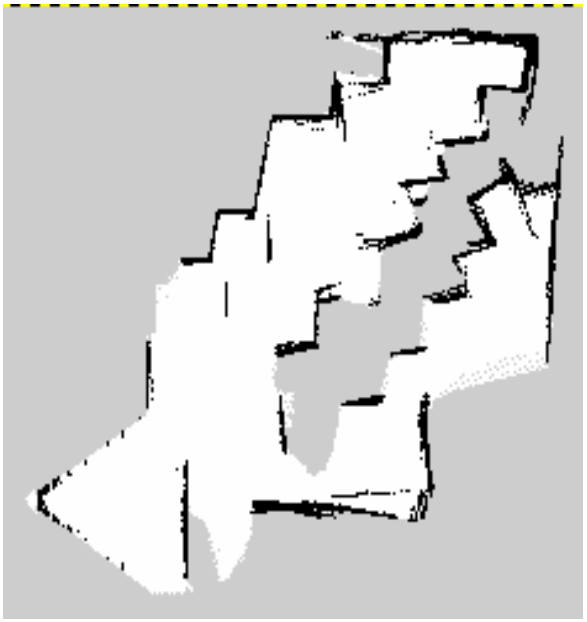


Εικόνα 3-41 Παράθυρα κατά την χαρτογράφηση



Εικόνα 3-42 Διορθωμένος χάρτης

Μετά την ολοκλήρωση της χαρτογράφησης κάνουμε εξαγωγή από την βάση δεδομένων **rtabcam.db** του χάρτη **rtabcam.pgm** που φαίνεται στην Εικόνα 3-43.



Εικόνα 3-43 Χάρτης από δεδομένα κάμερας

3.4.5 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης με LiDAR

Όπως αναλύεται στο 3.2.3.4 τα στοιχεία της αυτόνομης πλοήγησης είναι ο αλγόριθμος εντοπισμού της θέσης του ρομπότ σε σχέση με την αρχική του (Localization), ο καθολικός σχεδιαστής (Global Planner), ο τοπικός σχεδιαστής (Local Planner) και ο αλγόριθμους ελέγχου. Η εντολή εκκίνησης της αυτόνομης πλοήγησης είναι η XI:

XI. **ros2 launch nav2_bringup bringup_launch.py **
**use_sim_time:="true" **
**localization:=true **
**map:=/home/vgan/fossbot_sim/maps/map.yaml **
params_file:=/home/vgan/fossbot_sim/params/nav2lidar.yaml

XII. ros2 topic pub /initialpose geometry_msgs/msg/PoseWithCovarianceStamped

```
"{ header: {frame_id: 'map'},
        pose: {
                pose: {
                        position: {x: 0.0, y: 0.0, z: 0.0},
                        orientation: {z: 0.0, w: 1.0}
                        }
                }
 }"
```

Με την εντολή XI εκκινείται ο κόμβος nav2_bringup με χρόνο προσομοίωσης και ταυτόχρονη εκτέλεση Localization. Το αρχείο **nav2lidar.yaml** περιέχει όλους τους παραμέτρους που χρειάζονται οι αλγόριθμοι του Nav2 ώστε να εκτελεστεί η αυτόνομη πλοήγηση χρησιμοποιώντας τον χάρτη που ορίζεται στο αρχείο map.yaml. Μετά την εκκίνηση του Nav2 εκτελείται η εντολή XII που δημοσιεύει την αρχική θέση του ρομπότ στον χάρτη ώστε να γίνει η αρχικοποίηση του αλγόριθμου που κάνει localization. Παρακάτω παρουσιάζονται οι παράμετροι του αρχείου YAML για κάθε στοιχείο του.

3.4.5.1 Localization

Όπως αναφέρεται στο 2.4.5.1 το Localization είναι η πρώτη διαδικασία του SLAM και εκτελείται στην αρχή του Nav2 χρησιμοποιώντας αλγόριθμους όπως ο Adaptive Monte Carlo Localization (AMCL). Οι παράμετροι του αρχείου nav2lidar.yaml για το Localization φαίνονται στην Εικόνα 3-44.

```yaml
amcl:
  ros__parameters:
    use_sim_time: true
    alpha1: 0.2
    alpha2: 0.2
    alpha3: 0.2
    alpha4: 0.2
    alpha5: 0.2
    base_frame_id: "fossbot/base_footprint"
    beam_skip_distance: 0.5
    beam_skip_error_threshold: 0.9
    beam_skip_threshold: 0.3
    do_beamskip: false
    global_frame_id: "map"
    lambda_short: 0.1
    laser_max_range: 20.0
    laser_min_range: -1.0
    laser_model_type: "likelihood_field"
    max_beams: 60
    max_particles: 2000
    min_particles: 500
    odom_frame_id: "fossbot/odom"
    pf_err: 0.05
    pf_z: 0.99
    recovery_alpha_fast: 0.0
    recovery_alpha_slow: 0.0
    resample_interval: 1
    robot_model_type: "nav2_amcl::DifferentialMotionModel"
    save_pose_rate: 0.5
    sigma_hit: 0.2
    tf_broadcast: true
    transform_tolerance: 0.5
    update_min_a: 0.2
    update_min_d: 0.25
    z_hit: 0.5
    z_max: 0.05
    z_rand: 0.5
    z_short: 0.05
    scan_topic: scan
    map_topic: map
    set_initial_pose: false
    always_reset_initial_pose: false
    first_map_only: false
    initial_pose:
      x: 0.0
      y: 0.0
      z: 0.0
      yaw: 0.0
```

Εικόνα 3-44 Παράμετροι amcl

3.4.5.2 Behavior Tree Navigator

Ο bt_navigator χρησιμοποιεί δέντρα εργασιών (Behavior Trees) για να οργανώσει τις ενέργειες που πρέπει να γίνουν ώστε να εκτελεστεί η πλοήγηση σε ένα στόχο, συνδέει τα άλλα στοιχεία του Nav2, διαχειρίζεται τα σφάλματα τους και αντιμετωπίζει προβλήματα που μπορεί να προκύψουν. Οι βασικοί παράμετροι του στο αρχείο nav2.yaml φαίνονται στην Εικόνα 3-45 και είναι οι εξής:

- global_frame, robot_base_frame : Τα ονόματα των frames του χάρτη και του ρομπότ.
- transform_tolerance : Ο μέγιστος χρόνος για τον οποίο θεωρείται ένας μετασχηματισμός συντεταγμένων έγκυρος για χρήση.

- Τα Behavior Trees για την πλοήγηση σε ένα στόχο ή μέσω πολλαπλών στόχων.
- Η φόρτωση των Behavior Trees αρχείων ή όχι κάθε φορά.
- Ο ορισμός στην κοινή μνήμη των κόμβων (Blackboard) των κλειδιών για μεμονωμένο στόχο, για λίστα στόχων και για την τροχιά προς αυτούς.
- Ο καθορισμός των ενεργών τύπων navigator (navigators).
- Τα πρόσθετα για την πλοήγηση σε έναν στόχο ή μέσω μιας σειρά από στόχους (navigate_to_pose, navigate_through_poses).
- Ο ορισμός των προθεμάτων για τα ονόματα των σφαλμάτων που προκύπτουν από τα πρόσθετα (error_code_name_prefixes).

```
bt_navigator:
  ros__parameters:
    use_sim_time: true
    global_frame: map
    robot_base_frame: fossbot/base_footprint
    transform_tolerance: 0.5
    filter_duration: 0.3
    default_nav_to_pose_bt_xml: "$(find-pkg-share nav2_bt_navigator)
    default_nav_through_poses_bt_xml: "$(find-pkg-share nav2_bt_navi
    always_reload_bt_xml: false
    goal_blackboard_id: goal
    goals_blackboard_id: goals
    path_blackboard_id: path
    navigators: ['navigate_to_pose', 'navigate_through_poses']
    navigate_to_pose:
      plugin: "nav2_bt_navigator::NavigateToPoseNavigator"
    navigate_through_poses:
      plugin: "nav2_bt_navigator::NavigateThroughPosesNavigator"
    error_code_name_prefixes:
      - assisted_teleop
      - backup
      - compute_path
      - drive_on_heading
      - follow_path
      - nav_thru_poses
      - nav_to_pose
      - spin
      - route
      - wait
```

Εικόνα 3-45 Παράμετροι bt_navigator

3.4.5.3 Local Planner

Ο τοπικός σχεδιαστής αναφέρεται στο αρχείο των παραμέτρων ως Controller Server και μετατρέπει τις υπολογισμένες διαδρομές (paths) σε εντολές κίνησης (cmd_vel) για το σύστημα κίνησης του ρομπότ. Οι βασικές ρυθμίσεις του φαίνονται στην Εικόνα 3-46 και είναι οι εξής:

- Η συχνότητα ενημέρωσης του ελεγκτή (controller_frequency).
- Το ποσοστό της επιτρεπόμενης αποτυχίας (failure_tolerance).

- Η ελάχιστη ταχύτητα για την οποία το ρομπότ θεωρείται ότι κινείται (velocity_threshold).
- Τα πρόσθετα που ελέγχουν αν το ρομπότ προχωράει, αν έφτασε στο στόχο και αν ακολουθεί την διαδρομή (checker_plugins).
- Οι παράμετροι βάσει των οποίων λειτουργούν τα παραπάνω πρόσθετα, οι οποίοι είναι για το πρώτο πόσο πρέπει να προχωρήσει το ρομπότ για να θεωρηθεί ότι κινείται (required_movement_radius) και ο μέγιστος χρόνος που επιτρέπεται να μην κινείται. Ενώ για το δεύτερο είναι ο μέγιστη διαφορά της απόστασης και του προσανατολισμού από το στόχο για να θεωρηθεί ότι επετεύχθη.
- Ο ορισμός του πρόσθετου του local planner που στην προκειμένη περίπτωση είναι ο ελεγκτής Dynamic Window Approach (DWB) και των ρυθμίσεων του. Στις ρυθμίσεις περιλαμβάνονται τα μέγιστα και τα ελάχιστα όρια της ταχύτητας, της επιτάχυνσης και την επιβράδυνσης. Επίσης οι παράμετροι που καθορίζουν τους συνδυασμούς ταχυτήτων που προσομοιώνει ο ελεγκτής (samples) για ορισμένο χρόνο (sim_time) και το βήμα για την γραμμική και την γωνιακή κίνηση (granularity).
- Τέλος ορίζονται ποια critics θα αξιολογούν τις τροχιές που δημιουργεί ο DWB βάσει διαφόρων παραγόντων. Οι παράγοντες αυτοί είναι ποινές και οι επιβραβεύσεις για την τροχιά και στην συγκεκριμένη περίπτωση έχουμε ποινή για μια τροχιά που χτυπάει ένα εμπόδιο (BaseObstacle), επιβράβευση για μια τροχιά που ευθυγραμμίζεται με τη διαδρομή (PathAlign) και προτίμηση τροχιάς που είναι κοντά στη διαδρομή (PathDist) ή που πλησιάζει το στόχο (GoalDist).

```yaml
controller_server:
  ros__parameters:
    use_sim_time: true
    controller_frequency: 10.0
    min_x_velocity_threshold: 0.001
    min_y_velocity_threshold: 0.001
    min_theta_velocity_threshold: 0.001
    failure_tolerance: 0.3
    progress_checker_plugins: ["progress_checker"]
    goal_checker_plugins: ["goal_checker"]
    controller_plugins: ["FollowPath"]
    progress_checker:
      plugin: "nav2_controller::SimpleProgressChecker"
      required_movement_radius: 0.1
      movement_time_allowance: 10.0
    goal_checker:
      stateful: true
      plugin: "nav2_controller::SimpleGoalChecker"
      xy_goal_tolerance: 0.25
      yaw_goal_tolerance: 0.25
    FollowPath:
      plugin: "dwb_core::DWBLocalPlanner"
      debug_trajectory_details: true
      min_vel_x: 0.0
      min_vel_y: 0.0
      max_vel_x: 0.3
      max_vel_y: 0.0
      max_vel_theta: 1.0
      min_speed_xy: 0.0
      max_speed_xy: 0.3
      min_speed_theta: 0.0
      acc_lim_x: 3.0
      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: 0
      vtheta_samples: 40
      sim_time: 1.5
      linear_granularity: 0.05
      angular_granularity: 0.025
      transform_tolerance: 0.2
      xy_goal_tolerance: 0.05
      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
    enable_stamped_cmd_vel: true
```

Εικόνα 3-46 Παράμετροι controller_server

3.4.5.4 Global Planner

Ο καθολικός σχεδιαστής αναφέρεται στο αρχείο των παραμέτρων ως Planner Server όπως φαίνεται στην Εικόνα 3-47 και υπολογίζει την βέλτιστη διαδρομή μεταξύ μιας αρχικής και μιας τελικής θέσης αποφεύγοντας τα εμπόδια. Στην παρούσα εργασία χρησιμοποιείται ο grid-based σχεδιαστής Navigation Function Planner (plugin: "nav2_navfn_planner/NavfnPlanner") με τον αλγόριθμο Dijkstra αντί του A* (use_astar: false). Επίσης ορίζεται η αναμενόμενη συχνότητα υπολογισμού των διαδρομών (expected_planner_frequency), η μέγιστη απόσταση (tolerance) από το στόχο για τον ορισμό της διαδρομής αν αυτός είναι κοντά σε εμπόδιο και οι διαδρομές σε άγνωστες περιοχές (allow_unknown).

```
planner_server:
  ros__parameters:
    use_sim_time: true
    expected_planner_frequency: 10.0
    planner_plugins: ["GridBased"]
    GridBased:
      plugin: "nav2_navfn_planner/NavfnPlanner"
      tolerance: 0.5
      use_astar: false
      allow_unknown: true
```

Εικόνα 3-47 Παράμετροι planner_server

3.4.5.5 Map Server

Ο map_server που φαίνεται στην Εικόνα 3-48 φορτώνει και δημοσιεύει τον υπάρχον χάρτη του περιβάλλοντος μέσω του αρχείου map.yaml.

```
map_server:
  ros__parameters:
    use_sim_time: true
    yaml_filename: "/home/vgan/fossbot_sim/maps/map.yaml"
```

Εικόνα 3-48 Παράμετροι map_server

3.4.5.6 Behavior Server

Ο behavior_server εκτελεί ενέργειες αποκατάστασης όταν το ρομπότ κολλήσει και όπως φαίνεται στην Εικόνα 3-49 περιέχει παραμέτρους για τον ορισμό των αναγκαίων topics και frames και για την μέγιστη και ελάχιστη ταχύτητα περιστροφής. Επίσης σε αυτές ορίζεται η λίστα με τις ενέργειες αποκατάστασης και τα αντίστοιχα πρόσθετα τους και ορισμένες ρυθμίσεις ασφαλείας όπως ο χρόνος προσομοίωσης πριν την εκτέλεση της κίνησης (simulated_ahead_time) και ο μέγιστος χρόνος αναμονής λήψης transforms πριν την αποτυχία.

```yaml
behavior_server:
  ros__parameters:
    use_sim_time: true
    local_costmap_topic: local_costmap/costmap_raw
    local_footprint_topic: local_costmap/published_footprint
    global_costmap_topic: global_costmap/costmap_raw
    global_footprint_topic: global_costmap/published_footprint
    cycle_frequency: 10.0
    behavior_plugins: ["spin", "backup", "drive_on_heading", "wait", "assisted_teleop"]
    spin:
      plugin: "nav2_behaviors/Spin"
    backup:
      plugin: "nav2_behaviors/BackUp"
    drive_on_heading:
      plugin: "nav2_behaviors/DriveOnHeading"
    wait:
      plugin: "nav2_behaviors/Wait"
    assisted_teleop:
      plugin: "nav2_behaviors/AssistedTeleop"
    local_frame: fossbot/odom
    global_frame: map
    robot_base_frame: fossbot/base_footprint
    transform_timeout: 0.1
    simulate_ahead_time: 2.0
    max_rotational_vel: 1.0
    min_rotational_vel: 0.4
    rotational_acc_lim: 3.2
    enable_stamped_cmd_vel: true
```

Εικόνα 3-49 Παράμετροι behavior_server

3.4.5.7 Costmaps

Τα costmaps χρησιμοποιούνται για την αποφυγή εμποδίων (local costmap) και την σχεδίαση της διαδρομής (global costmap) που θα ακολουθήσει το ρομπότ για να φτάσει στο στόχο. Η αποδοτικότητα των παραπάνω βασίζεται στις κοινές παραμέτρους των costmaps που φαίνονται στην Εικόνα 3-50, όπως είναι η συχνότητα ανανέωσης, το topic του αισθητήρα, η ζώνη κινδύνου (inflation_layer) και η ακτίνα του ρομπότ (robot_radius). Ενώ διαφέρουν στο frame που χειρίζονται, στην συχνότητα, την κίνηση τους μαζί με το ρομπότ και τον τύπο του χάρτη. Επίσης ορίζονται πρόσθετα για τα στρώματα του χάρτη τα οποία παρέχουν το κάθε ένα πληροφορίες και όλα μαζί τον πλήρη χάρτη. Σημαντική σημείωση για τον local χάρτη είναι ότι υπάρχει ρύθμιση που ορίζει το στρώμα που ανιχνεύει τα εμπόδια, το οποίο εδώ είναι το LiDAR, και έχει συγκεκριμένη εμβέλεια στο ύψος των εμποδίων.

```yaml
local_costmap:
  ros__parameters:
    use_sim_time: true
    update_frequency: 5.0
    publish_frequency: 2.0
    global_frame: fossbot/odom
    robot_base_frame: fossbot/base_footprint
    rolling_window: true
    width: 9
    height: 9
    resolution: 0.05
    robot_radius: 0.1
    plugins: ["obstacle_layer", "voxel_layer", "inflation_layer"]
    inflation_layer:
      plugin: "nav2_costmap_2d::InflationLayer"
      inflation_radius: 0.5
      cost_scaling_factor: 5.0
    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"
    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"
        raytrace_max_range: 3.0
        raytrace_min_range: 0.0
        obstacle_max_range: 2.5
        obstacle_min_range: 0.0
    static_layer:
      map_subscribe_transient_local: true
    always_send_full_costmap: true
```

Εικόνα 3-50 local_costmap

3.4.5.8 Collision Monitor

Ο Collision Monitor προστατεύει το ρομπότ από την σύγκρουση, ακόμα και αν όλα τα συστήματα κάνουν λάθος, παρακολουθώντας τον χώρο γύρω από το ρομπότ με τον αισθητήρα που έχουμε ορίσει στην παράμετρο observation_sources όπως φαίνεται στην Εικόνα 3-51. Για να το επιτύχει αυτό χρησιμοποιεί ζώνες ασφαλείας, οι οποίες ορίζονται ως πολύγωνα και έχει η κάθε μια συγκεκριμένες ρυθμίσεις. Η πρώτη η PolygonStop είναι κύκλος γύρω από το ρομπότ με ακτίνα 10 εκατοστά στον οποίο αν εμφανιστεί ένα εμπόδιο σταματάει το ρομπότ. Η PolygonSlow είναι τετράγωνο και αν εμφανιστεί εμπόδιο επιβραδύνει το ρομπότ κατά 30 % της

ταχύτητας του. Και τελευταία είναι η FootprintApproach, η οποία χρησιμοποιεί το πραγματικό σχήμα του ρομπότ από το topic "/local_costmap/published_footprint" για να προβλέψει την πορεία του και να υπολογίσει αν υπάρχει πιθανότητα να συγκρουστεί το ρομπότ στα επόμενα 2 δευτερόλεπτα.

```
collision_monitor:
  ros__parameters:
    use_sim_time: true
    base_frame_id: "fossbot/base_footprint"
    odom_frame_id: "fossbot/odom"
    cmd_vel_in_topic: "cmd_vel_smoothed"
    cmd_vel_out_topic: "cmd_vel"
    state_topic: "collision_monitor_state"
    transform_tolerance: 0.5
    source_timeout: 5.0
    base_shift_correction: true
    stop_pub_timeout: 2.0
    enable_stamped_cmd_vel: true
    use_realtime_priority: false
    polygons: ["PolygonStop", "PolygonSlow", "FootprintApproach"]
    PolygonStop:
      type: "circle"
      radius: 0.1
      action_type: "stop"
      min_points: 4
      visualize: true
      polygon_pub_topic: "polygon_stop"
      enabled: true
    PolygonSlow:
      type: "polygon"
      points: "[[0.1, 0.1], [0.1, -0.1], [-0.1, -0.1], [-0.1, 0.1]]"
      action_type: "slowdown"
      min_points: 4
      slowdown_ratio: 0.3
      visualize: true
      polygon_pub_topic: "polygon_slowdown"
      enabled: true
    FootprintApproach:
      type: "polygon"
      action_type: "approach"
      footprint_topic: "/local_costmap/published_footprint"
      time_before_collision: 2.0
      simulation_time_step: 0.02
      min_points: 6
      visualize: False
      enabled: true
    observation_sources: ["scan"]
    scan:
      source_timeout: 0.2
      type: "scan"
      topic: "/scan"
      enabled: true
```

Εικόνα 3-51 Παράμετροι collision_monitor

3.4.5.9 Velocity Smoother

Ο Velocity Smoother κάνει τις εντολές ταχύτητας πιο ομαλές ώστε να κινείται το ρομπότ χωρίς απότομες κινήσεις και τραντάγματα. Οι βασικοί παράμετροι του φαίνονται στην Εικόνα 3-52 και είναι οι εξής:

- smoothing_frequency: Η συχνότητα υπολογισμού βελτιωμένων εντολών.

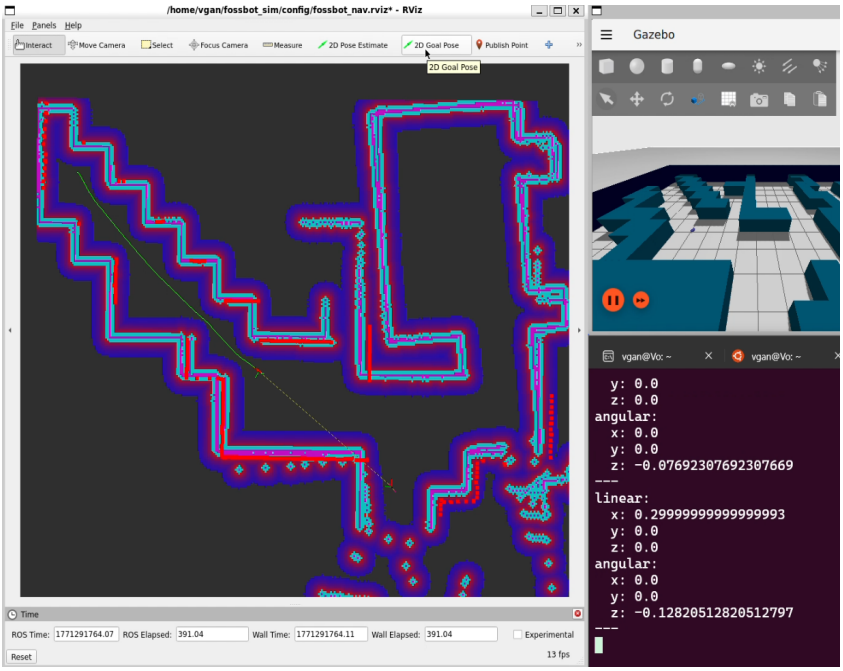
- feedback: Η επιλογή για ανατροφοδότηση από την οδομετρία.
- max_velocity: Μέγιστη ταχύτητα κίνησης μπροστά και στροφής.
- min_velocity: Ελάχιστη ταχύτητα κίνησης μπροστά και στροφής.
- max_accel: Όριο αύξησης της ταχύτητας.
- max_decel: Όριο μείωσης της ταχύτητας.
- odom_topic: Το όνομα του topic της οδομετρίας
- use_realtime_priority: Η χρησιμοποίηση ή όχι προτεραιότητας για την επεξεργασία σε σχέση με άλλες διεργασίες ώστε να μην καθυστερεί ο συγκεκριμένος κόμβος.

```yaml
velocity_smoother:
  ros__parameters:
    use_sim_time: true
    smoothing_frequency: 20.0
    scale_velocities: false
    feedback: "OPEN_LOOP"
    max_velocity: [0.5, 0.0, 2.5]
    min_velocity: [-0.5, 0.0, -2.5]
    deadband_velocity: [0.0, 0.0, 0.0]
    velocity_timeout: 1.0
    max_accel: [2.5, 0.0, 3.2]
    max_decel: [-2.5, 0.0, -3.2]
    odom_topic: "/odom"
    odom_duration: 0.1
    use_realtime_priority: false
    enable_stamped_cmd_vel: true
```

Εικόνα 3-52 Παράμετροι velocity_smoother

3.4.5.10 Διαδικασία Αυτόνομης Πλοήγησης

Μετά την εκτέλεση των εντολών XI και XII, ενώ πρώτα έχει φορτώσει το gazebo με το sim.sh, ανοίγει το rviz2 και εμφανίζεται ο χάρτης που δημιουργήθηκε στην ενότητα 3.4.2.3. Στη συνέχεια επιλέγοντας ένα σημείο στον χάρτη με το κουμπί 2D goal pose δημιουργείται και εμφανίζεται στο χάρτη το μονοπάτι που θα ακολουθήσει το ρομπότ μέχρι το στόχο. Έπειτα αρχίζουν να στέλνονται εντολές ελέγχου κίνησης στο ρομπότ και να κινείται προς το στόχο μέχρι να φτάσει σε αυτόν. Στην Εικόνα 3-53 φαίνεται το μονοπάτι, ο χάρτης του περιβάλλοντος και οι εντολές που δημιουργούνται και στέλνονται από το nav2.



Εικόνα 3-53 Αυτόνομη Πλοήγηση με LiDAR

3.4.6 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης με κάμερα και LiDAR

Για την πραγματοποίηση της αυτόνομης πλοήγησης στο περιβάλλον με την χρήση του Εικόνα 3-39 και την ταυτόχρονη χρήση των αισθητήρων της κάμερας και του LiDAR χρησιμοποιήθηκε, όπως και στην χαρτογράφηση, το πακέτο RTAB-Map. Αντίθετα με την πλοήγηση με τον αλγόριθμο AMCL στην ενότητα 3.4.4, με τον RTAB-Map εκτελούμε πρώτα την εντολή XII για την εκκίνηση του αλγόριθμου για το Localization και έπειτα την εντολή XIII για την πλοήγηση.

Στην εντολή XII εκκινείται το αρχείο τύπου launch nav2rtab.py με δύο παραμέτρους που ορίζουν την χρήση του χρόνου προσομοίωσης και τον τρόπο λειτουργίας του πακέτου RTAB-Map σε Localization. Ενώ στην εντολή XIII εκκινείται ο κόμβος nav2_bringup με χρόνο προσομοίωσης χρησιμοποιώντας το αρχείο nav2rtab.yaml που περιέχει τους παραμέτρους για την αυτόνομη πλοήγηση και τον χάρτη που ορίζεται στο αρχείο rtabmap.yaml.

**XIII. ros2 launch ~/fossbot_sim/launch/nav2rtab.py **
**use_sim_time:=true **
localization:=true

**XIV. ros2 launch nav2_bringup bringup_launch.py **

**use_sim_time:=true **

**map:=/home/vgan/fossbot_sim/maps/rtabmap.yaml **

**params_file:=/home/vgan/fossbot_sim/params/nav2rtab.yaml **

autostart:=true

3.4.6.1 Αρχείο εκκίνησης - nav2rtab.py

Το nav2rtab.py είναι ίδιο με το αρχείο rtabmap.py που χρησιμοποιούμε στην χαρτογράφηση με την προσθήκη δυο επιπλέον κόμβων οι οποίοι φαίνονται στην Εικόνα 3-54. Ο πρώτος μετατρέπει τα δεδομένα από την κάμερα βάθους (depth_image) σε νέφος σημείων (camera_cloud) στον τρισδιάστατο χώρο και το δημοσιεύει. Στη συνέχεια λαμβάνει ο δεύτερος το νέφος σημείων και το αναλύει για να εντοπίσει τα εμπόδια (/camera/obstacles) και το έδαφος (/camera/ground).

```
Node(
    package='rtabmap_util',
    executable='point_cloud_xyz',
    name='point_cloud_xyz_node',
    output='screen',
    parameters=[{
        'decimation': 2,
        'max_depth': 3.0,
        'voxel_size': 0.02,
        'use_sim_time': use_sim_time
    }],
    remappings=[
        ('depth/image', '/depth_image'),
        ('depth/camera_info', '/camera_info'),
        ('cloud', '/camera/cloud')
    ]),

Node(
    package='rtabmap_util',
    executable='obstacles_detection',
    name='obstacles_detection_node',
    output='screen',
    parameters=[parameters],
    remappings=[
        ('cloud', '/camera/cloud'),
        ('obstacles', '/camera/obstacles'),
        ('ground', '/camera/ground')
    ]),
```

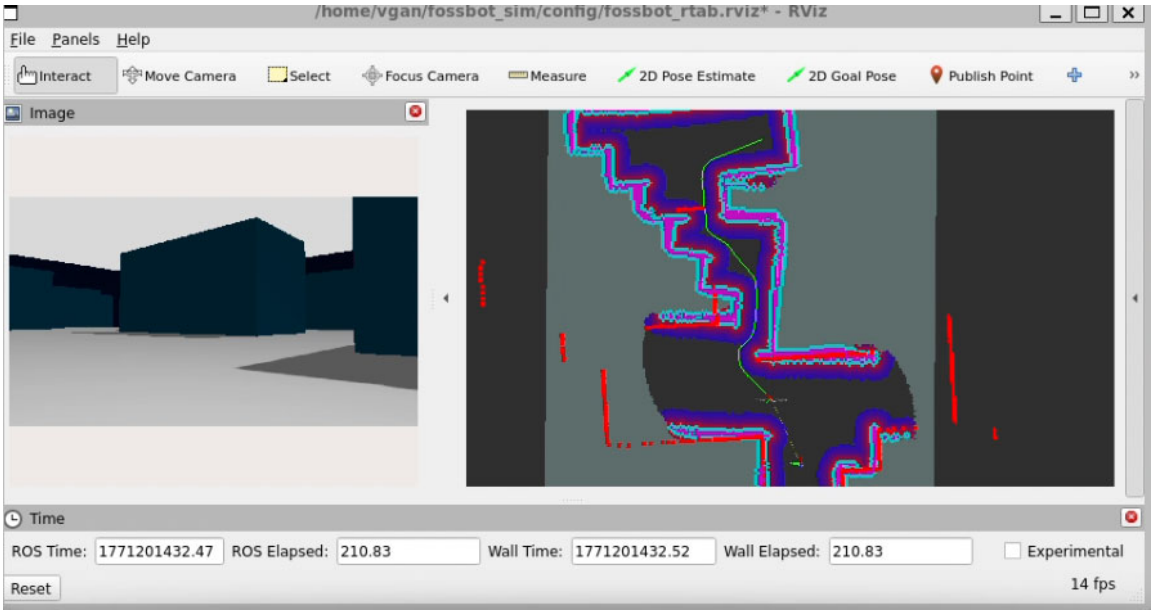
Εικόνα 3-54 Κόμβοι επεξεργασίας δεδομένων βάθους

3.4.6.2 Αρχείο παραμέτρων - nav2rtab.yaml

Το αρχείο nav2rtab.yaml είναι παρόμοιο με το nav2lidar.yaml από την ενότητα 3.4.4 με ορισμένες διαφορές ώστε να χρησιμοποιούνται και οι πληροφορίες της κάμερας στην όλη διαδικασία. Συγκεκριμένα στο Local Costmap και στο Global Costmap έχουν προστεθεί στο voxel_layer δύο ακόμα observation_sources, η ground που παίρνει δεδομένα από το topic "/camera/ground" και η obstacles που παίρνει δεδομένα από το topic "/camera/obstacles" ώστε να λαμβάνονται υπόψη και τα εμπόδια που βλέπει η κάμερα. Επίσης στο collision_monitor προστέθηκε στα observation_sources το camera που παίρνει δεδομένα από το topic "/camera/obstacles" ώστε να παρακολουθεί και τα δεδομένα της κάμερας παρέχοντας ένα επιπλέον επίπεδο ασφαλείας.

3.4.6.3 Διαδικασία Αυτόνομης Πλοήγησης

Μετά την εκτέλεση του sim.sh και των εντολών XIII και XIV εκτελείται η ίδια διαδικασία με την ενότητα 3.4.5.10 με την διαφορά ότι για την πλοήγηση χρησιμοποιείται ο χάρτης και η βάση δεδομένων που δημιουργήθηκαν στην ενότητα 3.4.3.2. Ο χάρτης, η διαδρομή και η κάμερα φαίνονται στην Εικόνα 3-55.



Εικόνα 3-55 Πλοήγηση με κάμερα και LiDAR

3.4.7 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης με κάμερα

Για την πραγματοποίηση της αυτόνομης πλοήγησης στο περιβάλλον με την χρήση της κάμερας χρησιμοποιήθηκε όπως και στην ενότητα 3.4.6 το RTAB-Map με τις εξής εντολές:

XV. **ros2 launch ~/fossbot_sim/launch/nav2cam.py **
**use_sim_time:=true **
localization:=true

XVI. **ros2 launch nav2_bringup bringup_launch.py **
**use_sim_time:=true **
**map:=/home/vgan/fossbot_sim/maps/rtabmap.yaml **
**params_file:=/home/vgan/fossbot_sim/params/nav2cam.yaml **
autostart:=true

Η εντολή XV χρησιμοποιείται για την εκκίνηση του αλγόριθμου για το Localization όπου εκκινείται το αρχείο τύπου launch nav2cam.py με λειτουργία του RTAB-Map σε Localization. Έπειτα με την εντολή XVI εκκινείται ο κόμβος nav2_bringup με χρόνο προσομοίωσης χρησιμοποιώντας το αρχείο nav2cam.yaml που περιέχει τους παραμέτρους για την αυτόνομη πλοήγηση και τον χάρτη που ορίζεται στο αρχείο rtabcam.yaml. Το αρχείο nav2cam.py δημιουργήθηκε από το αρχείο rtabcam.py από την ενότητα 3.4.4 με την πρόσθεση των κόμβων που φαίνονται στην Εικόνα 3-56 για την ανίχνευσης εμποδίων με την κάμερα.

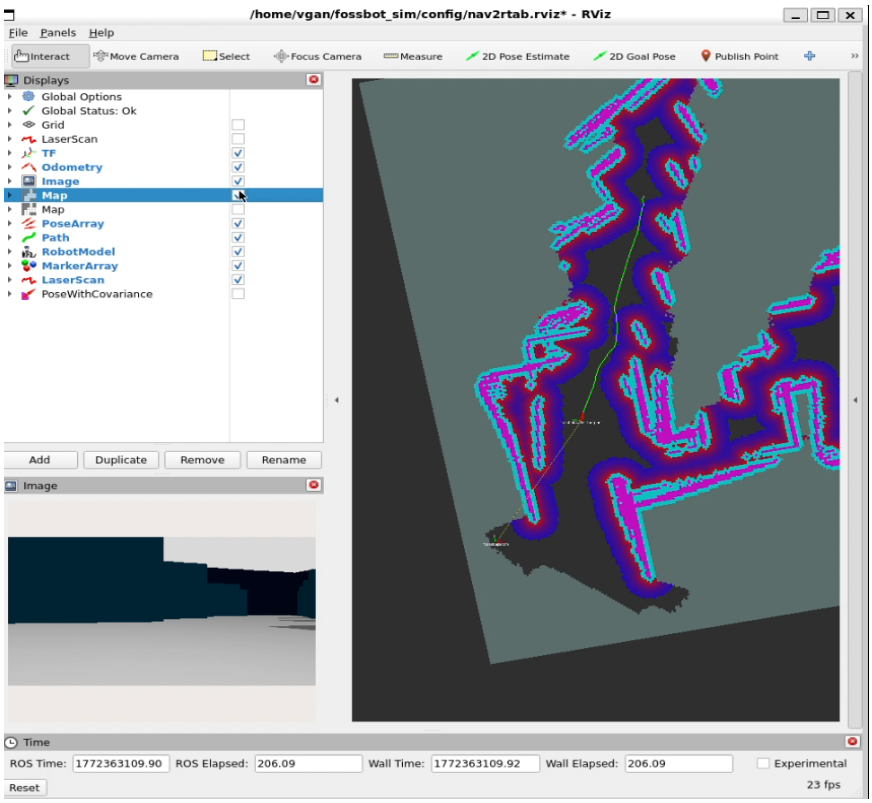
```
Node(
    package='rtabmap_util',
    executable='point_cloud_xyz',
    name='point_cloud_xyz_node',
    output='screen',
    parameters=[{
        'decimation': 2,
        'max_depth': 3.0,
        'voxel_size': 0.02,
        'use_sim_time': use_sim_time
    }],
    remappings=[
        ('depth/image', '/depth_image'),
        ('depth/camera_info', '/camera_info'),
        ('cloud', '/camera/cloud')
    ]),

Node(
    package='rtabmap_util',
    executable='obstacles_detection',
    name='obstacles_detection_node',
    output='screen',
    parameters=[parameters],
    remappings=[
        ('cloud', '/camera/cloud'),
        ('obstacles', '/camera/obstacles'),
        ('ground', '/camera/ground')
    ]),
```

Εικόνα 3-56 Κόμβοι Ανίχνευσης Εμποδίων με Κάμερα

Το αρχείο nav2cam.yaml προέκυψε από το αντίστοιχο αρχείο της ενότητας 3.4.6 nav2rtab.yaml με ορισμένες αλλαγές ώστε να μην χρησιμοποιείται το LiDAR στην ανίχνευση εμποδίων παρά μόνο η κάμερα. Οι αλλαγές αυτές είναι ότι αφαιρέθηκαν όλες οι αναφορές στο topic '/scan' από την λίστα observation_sources του Local Costmap, του Global Costmap και του Collision Monitor.

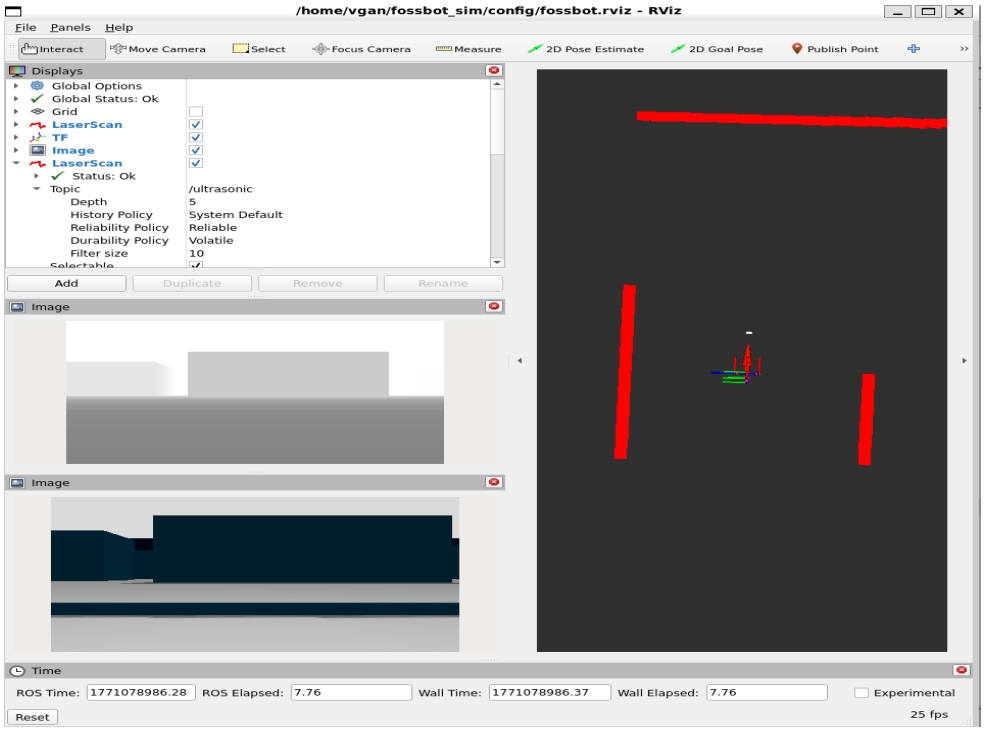
Παρόμοια με την ενότητα 3.4.6, μετά την εκτέλεση του sim.sh και των εντολών XV και XVI εκτελείται η ίδια διαδικασία με την ενότητα 3.4.5.10 με την διαφορά ότι για την πλοήγηση χρησιμοποιείται ο χάρτης και η βάση δεδομένων που δημιουργήθηκαν στην ενότητα 3.4.4. Ο χάρτης, η διαδρομή και η κάμερα φαίνονται στην Εικόνα 3-57.



Εικόνα 3-57 Πλοήγηση με κάμερα

3.4.8 Υλοποίηση Αλγορίθμου Αυτόνομης Πλοήγησης για Χαμηλά Εμπόδια

Μια σημαντική πρόκληση που αντιμετωπίζουν τα αυτόνομα ρομποτικά οχήματα είναι τα χαμηλού ύψους εμπόδια όπου το 2D LiDAR δεν μπορεί να τα ανιχνεύσει. Ο λόγος της αδυναμίας του αυτής είναι ότι το LiDAR βρίσκεται πάνω στο ρομπότ και πραγματοποιεί σαρώσεις σε ένα οριζόντιο επίπεδο στο οποίο αν δεν βρίσκεται το εμπόδιο δεν θα το ανιχνεύσει. Οι αισθητήρες που δοκιμάστηκαν για να λυθεί το πρόβλημα αυτό στο FOSSBot είναι η κάμερα και το ultrasonic. Για την δοκιμή αυτή προστέθηκε στον κόσμο obstacles ένα χαμηλού ύψους εμπόδιο 4 εκατοστών και στην συνέχεια έγινε η εκκίνηση του RTAB-Map. Όπως φαίνεται στην Εικόνα 3-58 ο LiDAR (κόκκινες γραμμές) δεν ανιχνεύει το εμπόδιο ενώ ο ultrasonic (άσπρη τελεία) μπορεί να το εντοπίσει καθώς εκπέμπει έναν κώνο ήχου και ανιχνεύει μια περιοχή μπροστά του.



Εικόνα 3-58 RViz2 με Ultrasonic

Επίσης δοκιμάστηκε και η κάμερα για την ανίχνευση του χαμηλού εμποδίου και ενώ το εμφανίζει στην κάμερα χρώματος δεν μπορεί να το ανιχνεύσει στην κάμερα βάθους λόγω του μικρού πάχους του. Δοκιμαστήκαν διαφορετικές ρυθμίσεις στην κάμερα και παράμετροι στο

RTAB-Map ώστε να επιτευχθεί η ανίχνευση του χαμηλού εμποδίου όμως χωρίς επιτυχία. Επομένως επιλέχθηκε η προσέγγιση με το ultrasonic και έγιναν οι απαραίτητες προσθήκες στα αρχεία .yaml με τις παραμέτρους του RTAB-Map ώστε να χρησιμοποιείται και ο ultrasonic αισθητήρας στην ανίχνευση εμποδίων για την αποφυγή συγκρούσεων. Οι προσθήκες αυτές έγιναν στον Local Costmap, στον Global Costmap και στο Collision Monitor όπως φαίνεται στην Εικόνα 3-59, στην Εικόνα 3-60 και στην Εικόνα 3-61 αντίστοιχα.

```yaml
local_costmap:
  local_costmap:
    ros__parameters:
      use_sim_time: true
      update_frequency: 5.0
      publish_frequency: 2.0
      global_frame: fossbot/odom
      robot_base_frame: fossbot/base_footprint
      rolling_window: true
      width: 9
      height: 9
      resolution: 0.05
      robot_radius: 0.1
      plugins: ["obstacle_layer", "voxel_layer", "inflation_layer"]
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        inflation_radius: 0.5
        cost_scaling_factor: 5.0
      obstacle_layer:
        plugin: "nav2_costmap_2d::ObstacleLayer"
        enabled: true
        observation_sources: scan ultrasonic
        scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: true
          marking: true
          data_type: "LaserScan"
        ultrasonic:
          topic: /ultrasonic
          marking: true
          clearing: false
          max_obstacle_height: 0.15
          min_obstacle_height: 0.02
```

Εικόνα 3-59 Local Costmap με Ultrasonic

```yaml
global_costmap:
  global_costmap:
    ros__parameters:
      use_sim_time: true
      update_frequency: 1.0
      publish_frequency: 1.0
      global_frame: map
      robot_base_frame: fossbot/base_footprint
      transform_tolerance: 0.5
      robot_radius: 0.1
      resolution: 0.05
      track_unknown_space: true
      plugins: ["static_layer", "obstacle_layer", "voxel_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"
          raytrace_max_range: 3.0
          raytrace_min_range: 0.0
          obstacle_max_range: 2.5
          obstacle_min_range: 0.0
      ultrasonic:
          topic: /ultrasonic
          marking: true
          clearing: true
          max_obstacle_height: 0.15
          min_obstacle_height: 0.02
          data_type: "LaserScan"
          obstacle_max_range: 2.0
          obstacle_min_range: 0.02
          raytrace_max_range: 2.0
          raytrace_min_range: 0.02
```

Εικόνα 3-60 Global Costmap με Ultrasonic

```yaml
collision_monitor:
  ros__parameters:
    use_sim_time: true
    base_frame_id: "fossbot/base_footprint"
    odom_frame_id: "fossbot/odom"
    cmd_vel_in_topic: "cmd_vel_smoothed"
    cmd_vel_out_topic: "cmd_vel"
    state_topic: "collision_monitor_state"
    transform_tolerance: 0.5
    source_timeout: 5.0
    base_shift_correction: true
    stop_pub_timeout: 2.0
    enable_stamped_cmd_vel: true
    use_realtime_priority: false
    polygons: ["UltrasonicStop", "UltrasonicSlow", "PolygonStop", "PolygonSlow", "FootprintApproach"]
    UltrasonicStop:
      type: "circle"
      radius: 0.30
      action_type: "stop"
      enabled: true
      visualize: true
      observation_sources: ["ultrasonic"]
    UltrasonicSlow:
      type: "circle"
      radius: 0.40
      action_type: "slowdown"
      slowdown_ratio: 0.3
      enabled: true
      observation_sources: ["ultrasonic"]
    PolygonStop:
      type: "circle"
      radius: 0.1
      action_type: "stop"
      min_points: 4
      visualize: true
      polygon_pub_topic: "polygon_stop"
      enabled: true
```

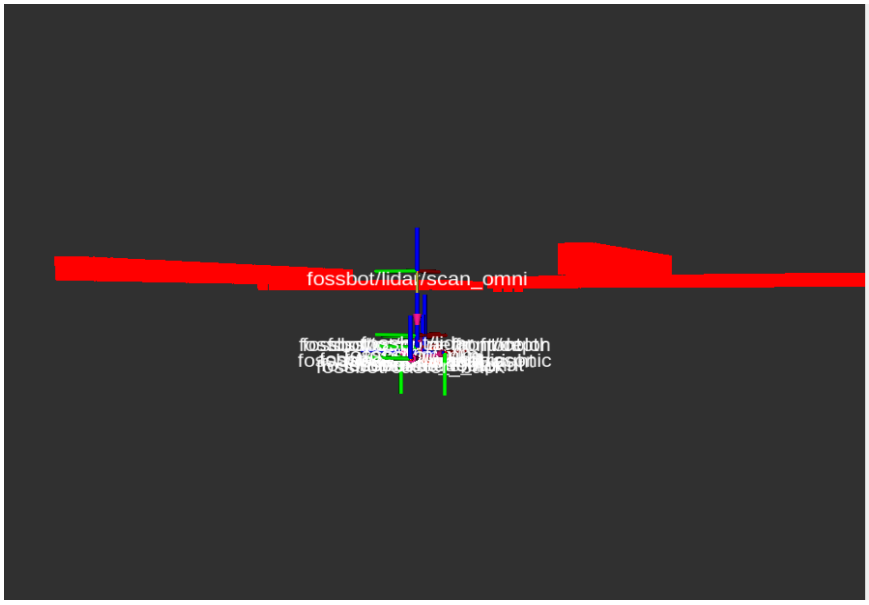
Εικόνα 3-61 Collision Monitor με Ultrasonic

3.5. Αξιολόγηση Πλατφόρμας Προσομοίωσης

Με τα σενάρια λειτουργίας που εκτελέσαμε στο προηγούμενο κεφάλαιο μπορέσαμε να προσομοιώσουμε τις κυρίες λειτουργίες του αναβαθμισμένου FOSSBot και να παρατηρήσουμε τα χαρακτηριστικά και τις προκλήσεις που αντιμετωπίζουν οι αισθητήρες σε μια τέτοια πλατφόρμα.

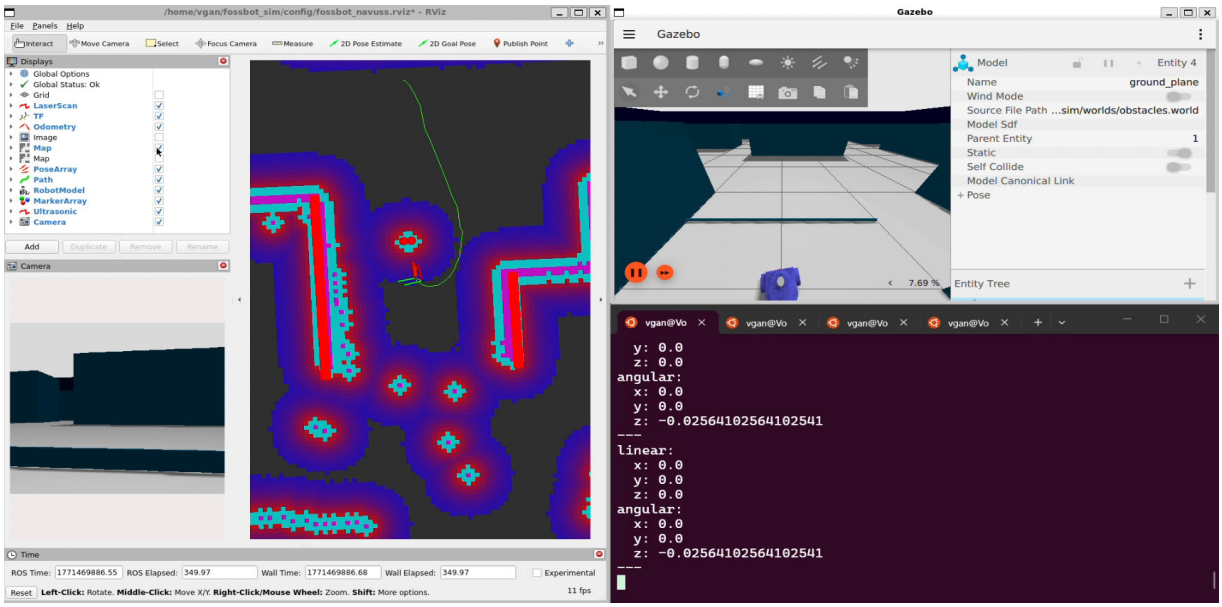
3.5.1 Αξιολόγηση Αισθητήρων

Το LiDAR στην ενότητα 3.4.2.3 χρησιμοποιήθηκε για την χαρτογράφηση του χώρου με πολύ ικανοποιητικά αποτελέσματα όπως φαίνεται στην Εικόνα 3-35 Χάρτης , όπου αποθηκεύτηκαν τα εμπόδια που υπάρχουν στο ύψος που βρίσκεται το LiDAR. Κατά την χαρτογράφηση η ταχύτητα της κίνησης και της περιστροφής δεν επηρέαζε την ορθή δημιουργία του χάρτη όμως σε αυτόν δεν βρίσκονται αντικείμενα με ύψος κάτω από το ύψος που βρίσκεται το LiDAR, όπως φαίνεται στην Εικόνα 3-62. Η αυτόνομη πλοήγηση είναι πολύ αξιόπιστη με το LiDAR σε περιβάλλον με ψηλά εμπόδια όμως λόγω των περιορισμών του αισθητήρα αυτού σε περιβάλλον με χαμηλού ύψους εμπόδια υπάρχει μεγάλη πιθανότητα σύγκρουσης του ρομπότ με ένα από αυτά.



Εικόνα 3-62 Δεδομένα LiDAR

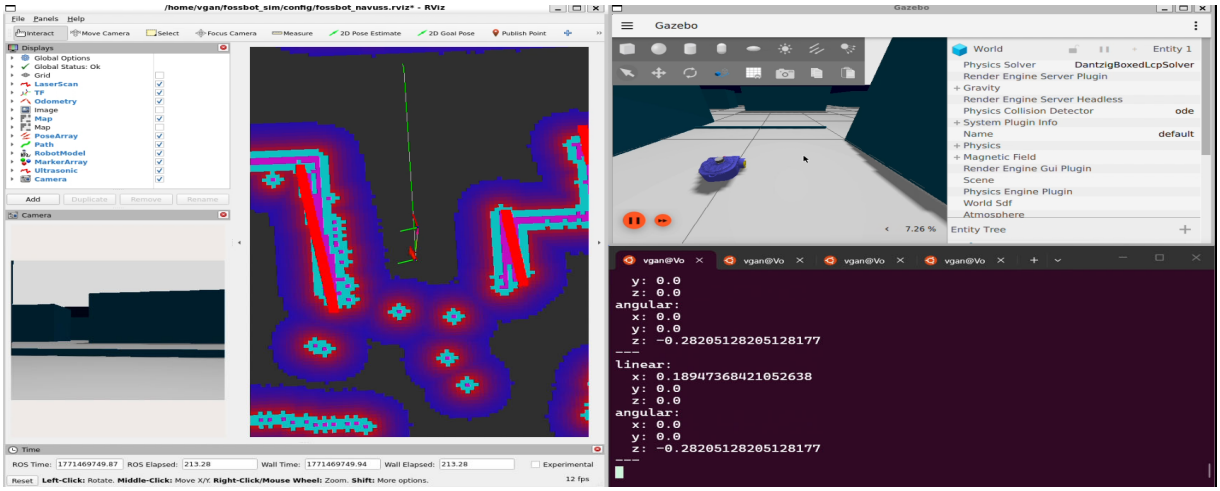
Όσον αφορά το Ultrasonic, δεν ήταν εφικτό να χρησιμοποιηθεί στην χαρτογράφηση λόγω της στενής δέσμης ακτινών που εκπέμπει και τα περιορισμένα δείγματα που λαμβάνει. Όμως χρησιμοποιήθηκε επιτυχώς για την αυτόνομη πλοήγηση ώστε να ανιχνεύει τα χαμηλού ύψους εμπόδια που δεν μπορεί το LiDAR. Η πρόσθεση του ως πηγή ανίχνευσης εμποδίων στις παραμέτρους του Nav2 βοήθησε να εκτελείται η αυτόνομη πλοήγηση χωρίς προβλήματα ακολουθώντας μια έγκυρη και σύντομη πορεία που θα αποφεύγει χαμηλά εμπόδια όπως φαίνεται στην Εικόνα 3-63.



Εικόνα 3-63 Nav2 με ανίχνευση χαμηλού εμποδίου από το Ultrasonic

Κατά την αυτόνομη πλοήγηση και την επιλογή ενός στόχου υπολογίζεται η διαδρομή για τον στόχο αυτό όπως φαίνεται στην Εικόνα 3-64 και στην συνέχεια κατά την κίνηση του ρομπότ αν ανιχνευτεί από το ultrasonic ή το lidar ένα εμπόδιο που δεν είναι στο χάρτη, όπως ένα χαμηλού ύψους εμπόδιο, επαναϋπολογίζεται η διαδρομή αυτή όπως φαίνεται στην Εικόνα 3-63 και το όχημα αποφεύγει το εμπόδιο αυτό. Σε περίπτωση όμως που το εμπόδιο εμφανιστεί πολύ κοντά στο όχημα και δεν έχει το χρόνο το navigator να αλλάξει τη διαδρομή τότε το collision_monitor σταματάει το όχημα πριν αυτό συγκρουστεί με το εμπόδιο. Τέλος η ικανότητα του αισθητήρα αυτού να ανιχνεύει τα χαμηλά εμπόδια εξαρτάται από το ύψος της τοποθέτησης του στο ρομπότ όπου στο FOSSBot η θέση αυτή είναι στα 6 εκατοστά με αποτέλεσμα να ανιχνεύονται εμπόδια από 2 εκατοστό και πάνω. Τέλος το ultrasonic παρουσιάζει μεγάλη

ευαισθησία στην ταλάντωση του FOSSBot καθώς αυτό κινείται, και συγκεκριμένα κατά την κίνηση του οχήματος προς τα μπροστά αν ανασηκωθεί λίγο το μπροστινό του μέρος το ultrasonic θα “κοιτάει” πάνω από το χαμηλό εμπόδιο και δεν θα το βλέπει, ενώ κατά την κίνηση του προς τα πίσω αν ανασηκωθεί το πίσω μέρος το ultrasonic θα “βλέπει” το έδαφος.



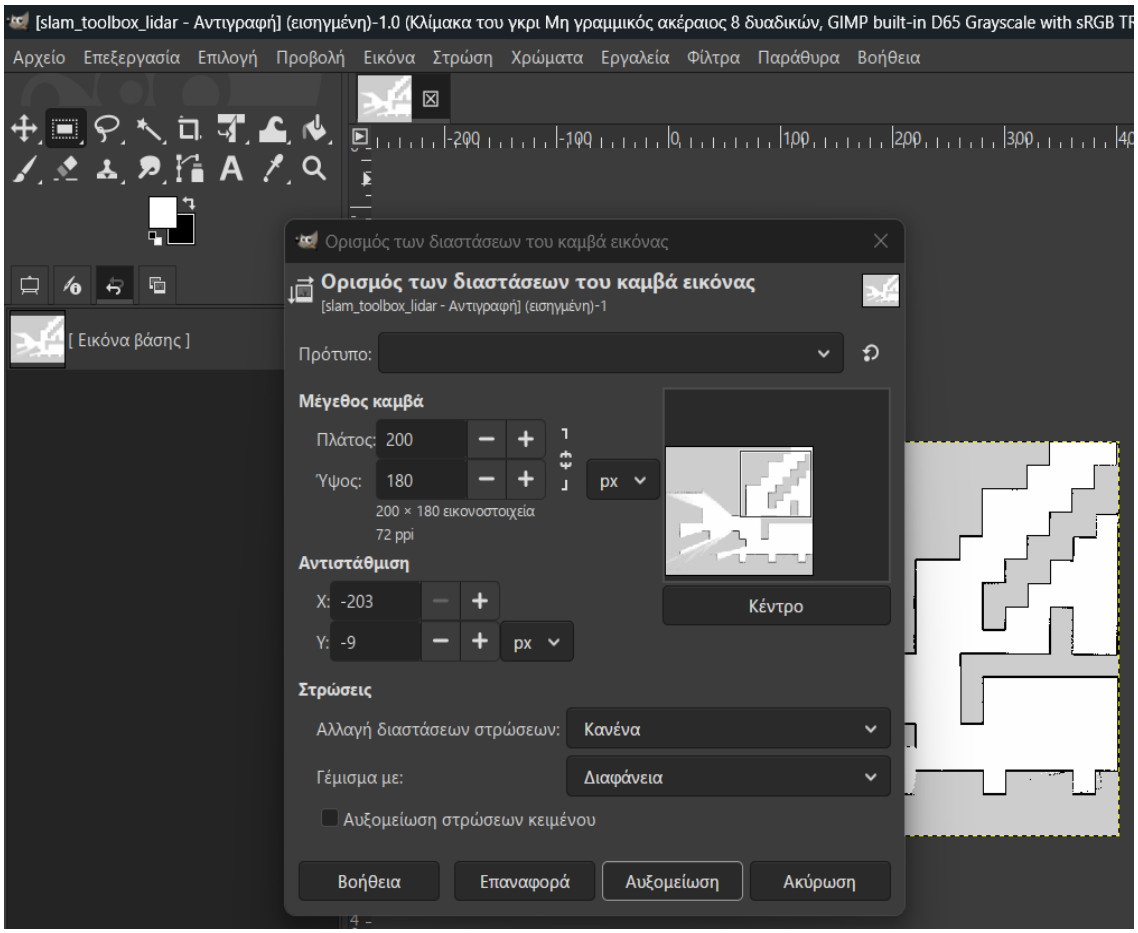
Εικόνα 3-64 Nav2 χωρίς ανίχνευση χαμηλού εμποδίου

Τέλος η κάμερα χρησιμοποιήθηκε μαζί με το LiDAR τόσο σε λειτουργία χαρτογράφησης όσο και σε λειτουργία πλοήγησης με ικανοποιητικά αποτελέσματα. Στην χαρτογράφηση με κάμερα αποθηκεύονται περισσότερες πληροφορίες για το περιβάλλον από ότι με το LiDAR και επίσης αναπαριστώνται τρισδιάστατα ολόκληρα τα αντικείμενα ανεξάρτητα το ύψος τους. Αντίθετα όμως από το LiDAR, το οποίο καταγράφει τα εμπόδια σε 360 μοίρες, η κάμερα χρειάζεται άμεση οπτική επαφή με τα εμπόδια και χαμηλές ταχύτητες κίνησης και περιστροφής ώστε να καταγράψει το Nav2 όλες τις λεπτομέρειες των αντικειμένων που βρίσκονται στον χώρο με αξιοπιστία. Επίσης τα δεδομένα της χαρτογράφησης λόγω της λεπτομέρειας τους καταλαμβάνουν πολύ περισσότερο αποθηκευτικό χώρο και χρειάζονται επεξεργασία ώστε να χρησιμοποιηθούν στην αυτόνομη πλοήγηση. Ο χάρτης που δημιουργείται από τα δεδομένα μόνο της κάμερας δεν είναι πάντα ακριβής και είναι απαραίτητη η χρήση δεδομένων και από το LiDAR ώστε να είναι αξιόπιστος και έγκυρος. Κατά την αυτόνομη πλοήγηση η κάμερα παρέχει παραπάνω πληροφορίες από ότι ο LIDAR, όμως λόγω του μεγάλου όγκου των δεδομένων απαιτείται μεγάλη υπολογιστική ισχύς. Όταν υπερφορτωθεί το σύστημα

προκαλείται καθυστέρηση στην επεξεργασία που εκτελεί το Nav2 και στην αποστολή των εντολών ελέγχου της κίνησης του ρομπότ με ότι αυτό συνεπάγεται.

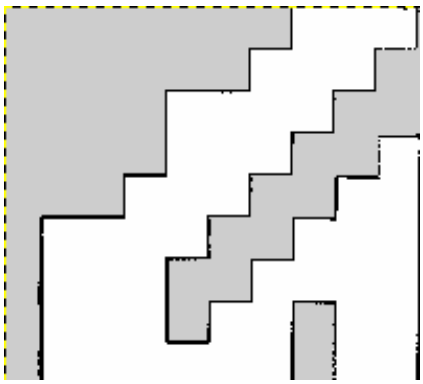
3.5.2 Σύγκριση χαρτών

Σε αυτή την ενότητα θα παρουσιαστεί η διαδικασία για την σύγκριση των χαρτών που παράχθηκαν από την χαρτογράφηση με τους διαφορετικούς συνδυασμούς αισθητήρων. Συγκεκριμένα στις ενότητες 3.4.2, 3.4.3, 3.4.4 πραγματοποιήθηκε χαρτογράφηση από την εφαρμογή SlamToolbox με LiDAR, και από την RTABMap με συνδυασμό κάμερας και LiDAR και με μόνο κάμερα αντίστοιχα. Οι χάρτες που παράχθηκαν, οι οποίοι φαίνονται αντίστοιχα στην Εικόνα 3-35, στην Εικόνα 3-39 και στην Εικόνα 3-43 επεξεργάστηκαν με το πρόγραμμα επεξεργασίας εικόνων "GIMP 3.0.4" ώστε να εξαχθούν χάρτες με τις ίδιες διαστάσεις από ένα συγκεκριμένο μέρος τους, όπως φαίνεται στην Εικόνα 3-65.



Εικόνα 3-65 Πρόγραμμα GIMP

Οι τελικοί χάρτες για τη πραγματοποίηση της σύγκρισης είναι οι χάρτες που φαίνονται στην Εικόνα 3-66, στην Εικόνα 3-67 και στην Εικόνα 3-68.



Εικόνα 3-66 Χάρτης για σύγκριση από LiDAR



Εικόνα 3-67 Χάρτης για σύγκριση από LiDAR και κάμερα



Εικόνα 3-68 Χάρτης για σύγκριση από κάμερα

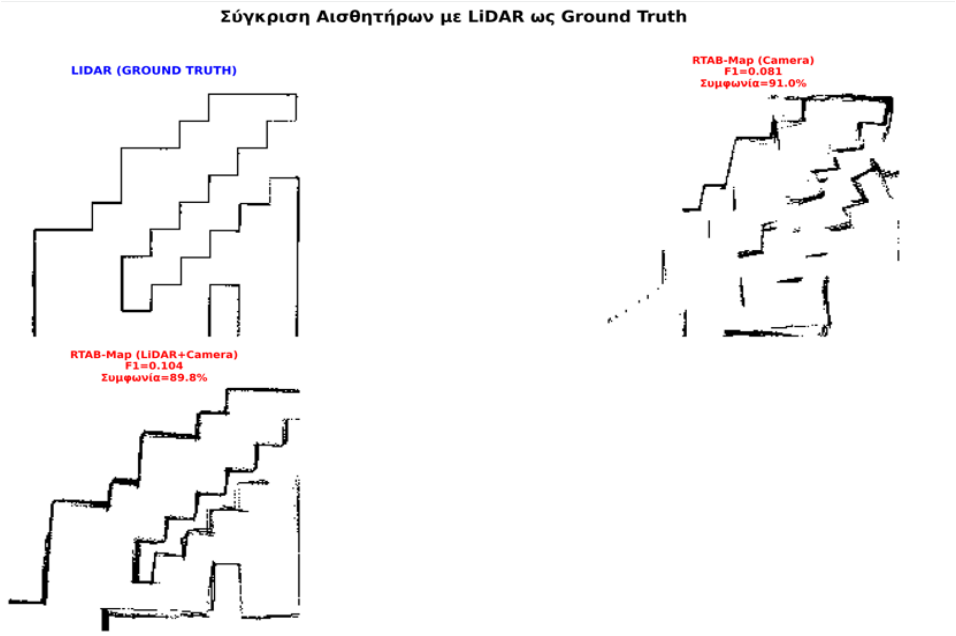
Παρατηρήθηκε ότι ο χάρτης από το SlamToolbox με το LiDAR είναι ο πιο ακριβής για το λόγο αυτό χρησιμοποιήθηκε ως ο πραγματικός χάρτης του περιβάλλοντος για την εκτέλεση σύγκρισης των χαρτών με script γραμμένο σε python. Το συγκεκριμένο script συγκρίνει τους χάρτες pixel προς pixel και βγάζει αποτελέσματα όπως είναι η ακρίβεια (Accuracy) και το F1-score ενώ χρησιμοποιείται επίσης το Σφάλμα Ρίζας Μέσου Τετραγώνου (RMSE) το οποίο μετράει την απόκλιση μεταξύ της εκτιμώμενης θέσης των εμποδίων και της πραγματικής τους θέσης. Μετά την σύγκριση των παραπάνω χαρτών είχαμε τα αποτελέσματα που φαίνονται στον Πίνακας 2-1, στον οποίο υπάρχουν:

- Η ομοιότητα η οποία δείχνει το ποσοστό των pixels που συμφωνούν μεταξύ δύο χαρτών.
- Η correlation που μετράει αν οι δύο χάρτες συσχετίζονται με τιμές από -1 μέχρι 1.
- Η Structural Similarity Index (SSIM) που μετράει πόσο μοιάζει η δομή των δύο χαρτών σε σχέση με τα μοτίβα, τα σχήματα και την υφή. Η SSIM παίρνει τιμές από 0 μέχρι 1 και λαμβάνει υπόψη την φωτεινότητα, την αντίθεση και την δομή.
- Το Jaccard με τιμές από 0 μέχρι 1, μετράει πόσο συμφωνούν οι δύο χάρτες στους τοίχους και όχι στον ελεύθερο χώρο.
- Το RMSE που όπως αναφέρθηκε προηγουμένως είναι το σφάλμα θέσης τοίχων σε μέτρα.

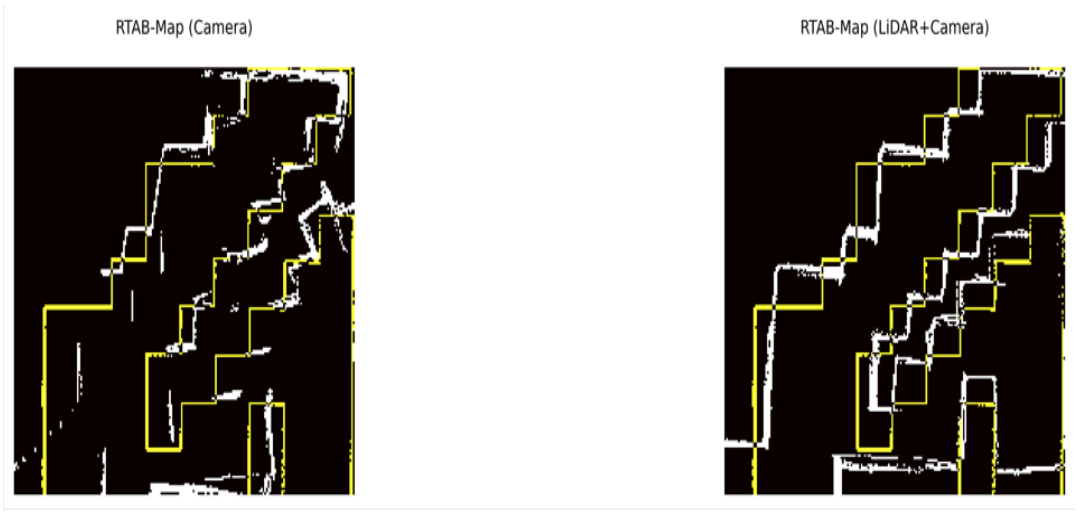
Πίνακας 3-2 Αποτελέσματα Σύγκρισης Χαρτών

Σύγκριση	Ομοιότητα %	Συσχέτιση	SSIM	Jaccard	RMSE (m)
LiDAR με κάμερα	0.9100	0.0359	0.6481	0.0423	0.5717
LiDAR με Fusion	0.8978	0.0575	0.6107	0.0550	0.3151
Κάμερα με Fusion	0.8861	0.0705	0.5860	0.0694	0.3569

Από τις τιμές στον πίνακα συμπεραίνουμε ότι οι χάρτες έχουν χαμηλή συσχέτιση και ότι συμφωνούν μόνο το 4% με 6 % των τοίχων και ότι η απόκλιση είναι μεγάλη όμως έχουν μεγάλη ομοιότητα μεταξύ τους. Επίσης ο χάρτης από το συνδυασμό των αισθητήρων LiDAR και κάμερας έχει καλύτερη συνολική απόδοση (F1) και μικρότερο σφάλμα θέσης σε σχέση με τον χάρτη από το LiDAR, επομένως ταιριάζουν περισσότερο αυτοί οι δύο χάρτες. Επίσης εξήχθησαν δύο εικόνες, η Εικόνα 3-69 και η Εικόνα 3-70 στις οποίες παρατηρούνται οι διαφορές των χαρτών σε σχέση με τον χάρτη από το LiDAR.



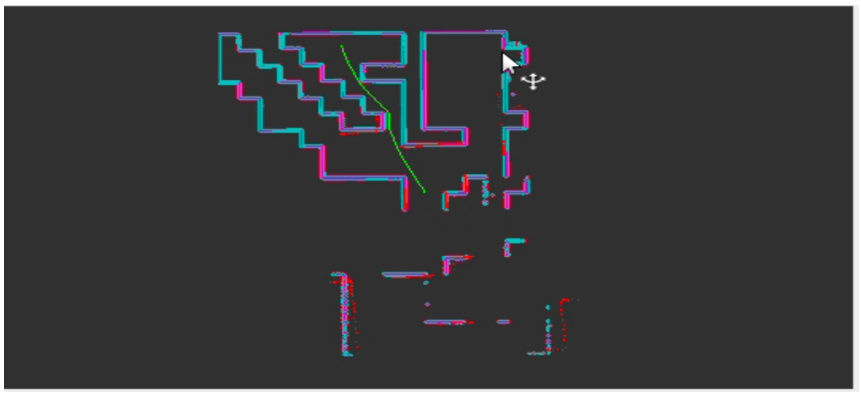
Εικόνα 3-69 Αποτελέσματα Σύγκρισης Χαρτών



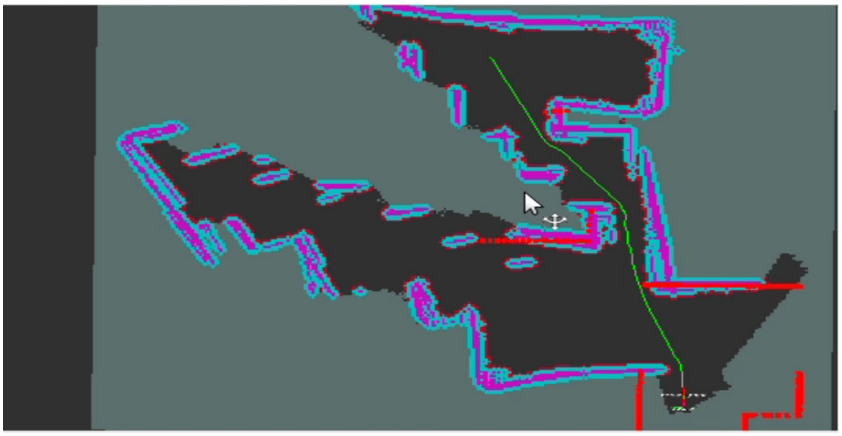
Εικόνα 3-70 Σύγκριση Χαρτών σε σχέση με τον χάρτη του LiDAR

3.5.3 Σύγκριση αυτόνομης πλοήγησης

Σε αυτή την ενότητα θα παρουσιαστεί η διαδικασία για την σύγκριση της αυτόνομης πλοήγησης με τους διαφορετικούς αισθητήρες. Για την σύγκριση αυτή έγιναν για κάθε αισθητήρα 20 εκτελέσεις αυτόνομης επιλογής με τον ίδιο στόχο πλοήγησης για όλους όπως στην Εικόνα 3-71 και στην Εικόνα 3-72.



Εικόνα 3-71 Επιλογή στόχου Πλοήγησης στον LiDAR χάρτη



Εικόνα 3-72 Επιλογή στόχου Πλοήγησης στον Fusion χάρτη

Για την σύγκριση των πλοηγήσεων καταγράφηκε ο χρόνος που χρειάστηκε το ρομπότ για να φτάσει στον στόχο πλοήγησης και η επίτευξη ή όχι του στόχου. Τα δεδομένα αυτά φαίνονται στον Πίνακας 3-3, στον Πίνακας 3-4, στον Πίνακας 3-5 και στο διάγραμμα στην Εικόνα 3-73.

Πίνακας 3-3 Χρόνοι Ολοκλήρωσης Πλοήγησης

Αριθμός Εκτέλεσης	Χρόνοι LiDAR (λεπτά)	Χρόνοι LiDAR και Κάμερας (λεπτά)	Χρόνοι Κάμερας (λεπτά)
1	5,8	7,2	9,5
2	6,1	8,1	11,2
3	5,9	7,8	10,8
4	6,3	8,4	12,5

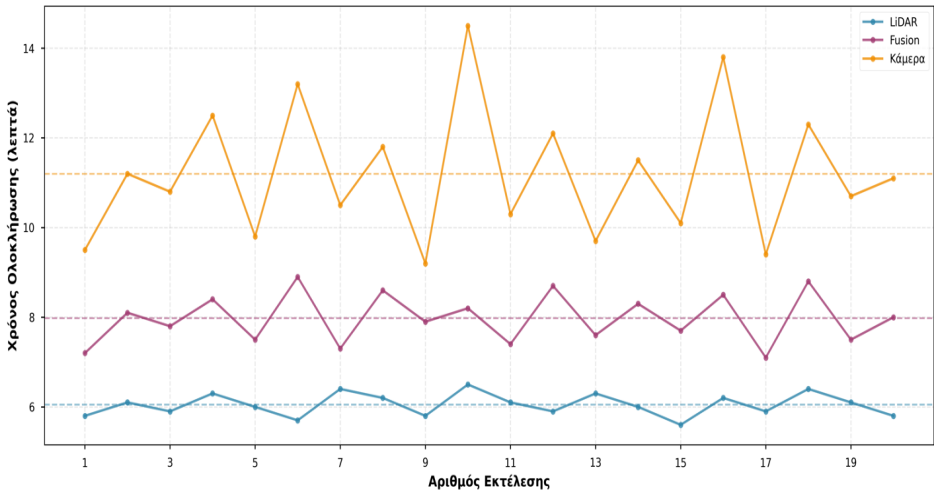
5	6,0	7,5	9,8
6	5,7	8,9	13,2
7	6,4	7,3	10,5
8	6,2	8,6	11,8
9	5,8	7,9	9,2
10	6,5	8,2	14,5
11	6,1	7,4	10,3
12	5,9	8,7	12,1
13	6,3	7,6	9,7
14	6,0	8,3	11,5
15	5,6	7,7	10,1
16	6,2	8,5	13,8
17	5,9	7,1	9,4
18	6,4	8,8	12,3
19	6,1	7,5	10,7
20	5,8	8,0	11,1
Μέσος Όρος	**6,06**	**8,01**	**11,22**
Ελάχιστος Χρόνος	**5,6**	**7,1**	**9,2**
Μέγιστος Χρόνος	**6,5**	**8,9**	**14,5**

Πίνακας 3-4 Επιτυχείς Πλοηγήσεις

Αισθητήρας	Επιτυχείς Δοκιμές	Ποσοστό Επιτυχίας
LiDAR	18	90%
Fusion (LiDAR+RGB-D)	14	70%
RGB-D Κάμερα	9	45%

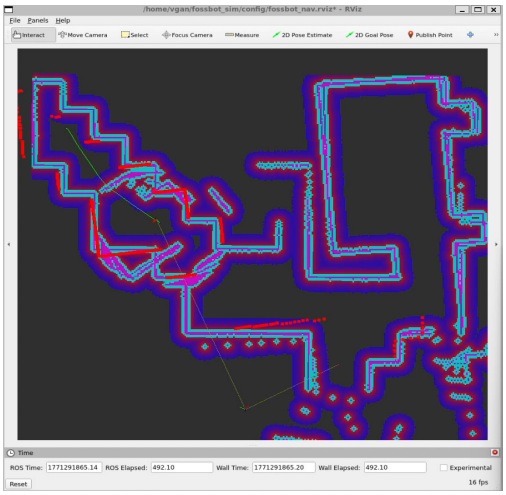
Πίνακας 3-5 Λόγοι Αποτυχίας Πλοήγησης

Λόγοι Αποτυχίας	LiDAR	LiDAR και Κάμερας	Κάμερας
Σύγκρουση	1	1	1
Κόλλημα	1	3	4
Λάθος Costmaps	0	2	6

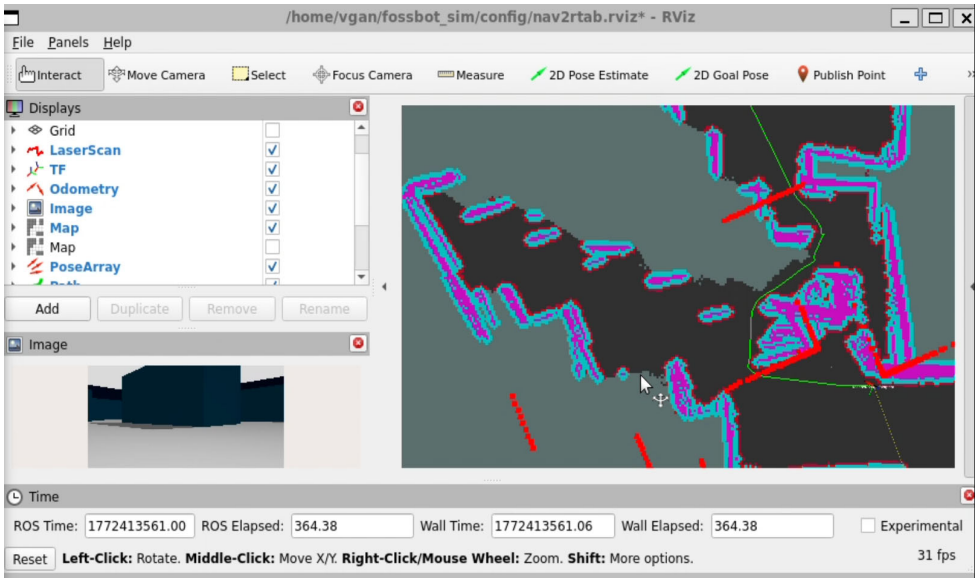


Εικόνα 3-73 Διακύμανση Χρόνων Πλοήγησης

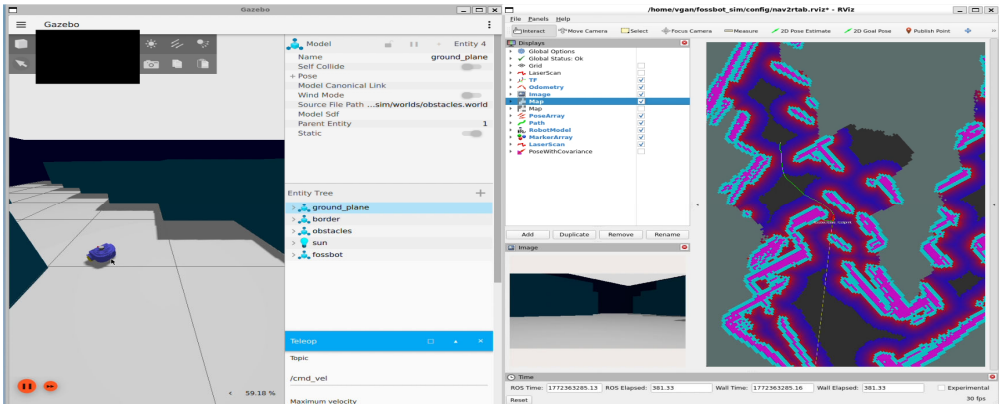
Από τους παραπάνω πίνακες προκύπτει ότι το LiDAR μόνο του είναι πιο αποδοτικό σε σχέση με τις άλλες προσεγγίσεις και όσο αφορά τον χρόνο πλοήγησης και όσον αφορά την ολοκλήρωση της. Ο συνδυασμός του LiDAR με την κάμερα, την βοηθάει να αποδώσει καλύτερα καθώς μόνη της έχει την χειρότερη απόδοση και τις περισσότερες συγκρούσεις. Παρακάτω στην Εικόνα 3-74, στην Εικόνα 3-75 και στην Εικόνα 3-76 φαίνονται οι συγκρούσεις και τα κολλήματα του ρομπότ κατά την εκτέλεση των πλοηγήσεων οι οποίες προκλήθηκαν από την ολίσθηση των τροχών.



Εικόνα 3-74 Κόλλημα FOSSBot με LiDAR



Εικόνα 3-75 Λάθος Costmap με Κάμερα και LiDAR



Εικόνα 3-76 Λάθος Costmap με Κάμερα

Κεφάλαιο 4 - Συμπεράσματα - Μελλοντικές επεκτάσεις

4.1. Συμπεράσματα

Ο κύριος στόχος της παρούσας εργασίας ήταν η αναβάθμιση του FOSSBot ώστε από ένα εκπαιδευτικό ρομπότ που χρησιμοποιείται για την εισαγωγή στις βασικές έννοιες της ρομποτικής και του προγραμματισμού να γίνει μια προηγμένη πλατφόρμα αυτόνομης πλοήγησης και έρευνας. Με την χρήση του περιβάλλοντος λογισμικού ROS2 Humble και του προσομοιωτή Ignition Gazebo Fortress ο στόχος αυτός επετεύχθη καθώς έγινε υλοποίηση σύνθετων αλγορίθμων χαρτογράφησης και πλοήγησης, οι οποίοι δεν ήταν εφικτό να εκτελεστούν με τον αρχικό εξοπλισμό του FOSSBot. Κατά την διαδικασία ανάπτυξης του λογισμικού και της υλοποίησης των αλγορίθμων προέκυψαν τα παρακάτω συμπεράσματα.

4.1.1 Σύγκριση των Αισθητήρων

1. Η χρησιμοποίηση του αισθητήρα LiDAR με το πακέτο λογισμικού SlamToolbox και τον προεπιλεγμένο αλγόριθμο Karto SLAM ήταν η πιο αποδοτική μέθοδος για την πραγματοποίηση της δισδιάστατης χαρτογράφησης του περιβάλλοντος. Επίσης με τη χρησιμοποίηση του παραγόμενου χάρτη και του αλγόριθμου AMCL το ποσοστό επιτυχίας στην αυτόνομη πλοήγηση ήταν 90% και οι χρόνοι ολοκλήρωσης της διαδρομής ήταν οι μικρότεροι σε σχέση με τις άλλες μεθόδους.
2. Η σύντηξη των δεδομένων από τους αισθητήρες LiDAR και κάμερας και η χρησιμοποίηση του πακέτου λογισμικού RTAB-Map έχει ως αποτέλεσμα οι χάρτες που παράγονται να είναι πιο λεπτομερείς οπτικά, να αυξάνεται όμως η κατανάλωση των υπολογιστικών πόρων και να μειώνεται το ποσοστό επιτυχίας στην πλοήγηση στο 70%.
3. Η χρήση της κάμερας βάθους ως μοναδικό αισθητήρα αντίληψης του περιβάλλοντος στην χαρτογράφηση και στην αυτόνομη πλοήγηση είχε τα μειονεκτήματα της δημιουργίας χαμηλής ποιότητας χαρτών και της ανάγκης να κινείται το ρομπότ με χαμηλή ταχύτητα ώστε να αποφευχθεί ο θόρυβος στα δεδομένα της κάμερας. Επίσης απαιτείται προσεκτικός καθορισμός των παραμέτρων εκτέλεσης του πακέτου RTAB-Map ώστε να μπορεί ο αλγόριθμος να διαχωρίζει το έδαφος από τα εμπόδια.

4.1.2 Ανίχνευση των Χαμηλών Εμποδίων

Σχετικά με την ανίχνευση των χαμηλών εμποδίων:

1. Αποδείχθηκε ότι ο αισθητήρας LiDAR δεν μπορεί να ανιχνεύσει εμπόδια με ύψος χαμηλότερο από το ύψος σάρωσης του.
2. Ενσωματώθηκε ο αισθητήρας υπερήχων ως πηγή δεδομένων στον υπολογισμό των χαρτών κόστους που χρησιμοποιούνται στην πλοήγηση με το Nav2 ώστε να δημιουργείται μια διαδρομή που θα αποφεύγει τα χαμηλά εμπόδια. Παράλληλα χρησιμοποιήθηκε και στον ελεγκτή συγκρούσεων ώστε το ρομπότ να αποφεύγει τα χαμηλά εμπόδια σε πραγματικό χρόνο και να ελαχιστοποιούνται έτσι οι συγκρούσεις με αυτά.
3. Τέλος παρατηρήθηκε ότι η ακρίβεια εντοπισμού των εμποδίων από τον αισθητήρα υπερήχων επηρεάζεται από την ταλάντωση του ρομπότ όταν αυξομειώνεται η ταχύτητα του, γεγονός που πρέπει να ληφθεί υπόψη για τον σχεδιασμό των αλγορίθμων ελέγχου της αυτόνομης πλοήγησης του.

4.1.3 Βελτιστοποίηση της Nav2

Παρατηρήθηκε ότι ο συνδυασμός του Global Planner που χρησιμοποιεί τον αλγόριθμο Dijkstra και του Local Planner που χρησιμοποιεί τον ελεγκτή DWB είχε ως αποτέλεσμα τον υπολογισμό αποδοτικής διαδρομής, όμως οι χάρτες κόστους που παράγουν επηρεάζονται αρκετά από την ολίσθηση των τροχών με αποτέλεσμα να κολλάει ή να συγκρούεται με εμπόδια το ρομπότ. Για την ελαχιστοποίηση της ολίσθησης των τροχών χρησιμοποιείται ο Velocity Smoother περιορίζοντας την ταχύτητα και την επιτάχυνση του ρομπότ.

4.1.4 Το Περιβάλλον Προσομοίωσης

Παρατηρήθηκε ότι ο συνδυασμός των λογισμικών Windows 11, WSL2, Ubuntu 22.04, ROS2 Humble και Ignition Gazebo Fortress λειτούργησε χωρίς ιδιαίτερα προβλήματα, με καλή χρησιμοποίηση των πόρων του συστήματος και με συμπεριφορά παρόμοια με native εγκατάσταση του λογισμικού Ubuntu. Επίσης προσέφερε εύκολη διαχείριση των φακέλων και των αρχείων του συστήματος μέσω του Εξερευνητή Αρχείων (File Explorer) των Windows. Όμως υπάρχει ο περιορισμός της χρησιμοποίησης του συνδυασμού μόνο στις εκδόσεις 10 και 11 των Microsoft Windows οι οποίες ενσωματώνουν το WSL.

4.1.5 Εκπαιδευτική Αξιοποίηση

Αποδείχθηκε ότι το αναβαθμισμένο FOSSBot μπορεί να υποστηρίξει τα εκπαιδευτικά και ερευνητικά σενάρια του πολύ πιο ακριβού ρομποτικού οχήματος TurtleBot και ως εκ τούτου είναι μια εξαιρετική και οικονομική λύση για πανεπιστημιακά μαθήματα που διδάσκουν SLAM, σύντηξη αισθητήρων και αυτόνομη πλοήγηση.

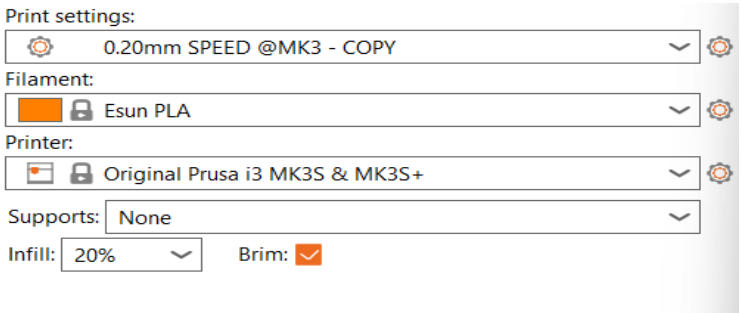
4.2. Μελλοντικές επεκτάσεις

Η παρούσα εργασία επικεντρώθηκε στον σχεδιασμό, την υλοποίηση και την αξιολόγηση της προσομοίωσης του αναβαθμισμένου FOSSBot όμως έχει ξεκινήσει και η κατασκευή του φυσικού ρομποτικού οχήματος ώστε να γίνει η μεταφορά του λογισμικού σε αυτό. Οι ενέργειες που έχουν γίνει και μπορούν να χρησιμοποιηθούν ως βάση για μελλοντική έρευνα παρουσιάζονται παρακάτω.

4.2.1 Πραγματοποιημένες Ενέργειες Κατασκευής Φυσικού FOSSBot

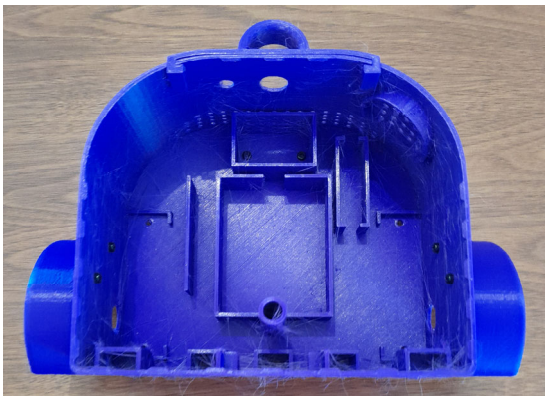
4.2.1.1 Τρισδιάστατη Εκτύπωση Οχήματος

Η πρώτη ενέργεια για την κατασκευή του φυσικού FOSSBot που πραγματοποιήθηκε είναι η τρισδιάστατη εκτύπωση των πλαστικών μερών του ρομπότ. Τα αρχεία STL που χρησιμοποιήθηκαν για την συγκεκριμένη εκτύπωση υπάρχουν στο αποθετήριο GitHub [83] και επεξεργάστηκαν με την εφαρμογή PrusaSlicer έκδοση 2.9.4. Στην εφαρμογή αυτή έγινε επιλογή του τρισδιάστατου εκτυπωτή «Prusa MK3S+» με NOZZLE «0.4» και τις ρυθμίσεις που φαίνονται στην Εικόνα 4-1.

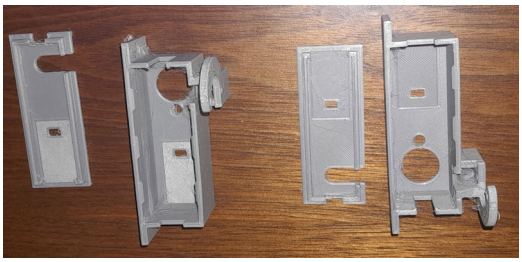


Εικόνα 4-1 Ρυθμίσεις Εκτυπωτή

Στην συνέχεια έγινε slice σε κάθε αρχείο και εξαγωγή σε αρχεία .gcode αποθηκεύοντας τα στην SD Card του εκτυπωτή. Ο συνολικός χρόνος εκτύπωσης ήταν 36 ώρες και από το υλικό μέρος του FOSSBot φαίνεται η βάση του, οι θήκες των κινητήρων και η βάση στήριξης των ηλεκτρονικών εξαρτημάτων του στην Εικόνα 4-2, στην Εικόνα 4-3 και στην Εικόνα 4-4 αντίστοιχα.



Εικόνα 4-2 Βάση FOSSBot



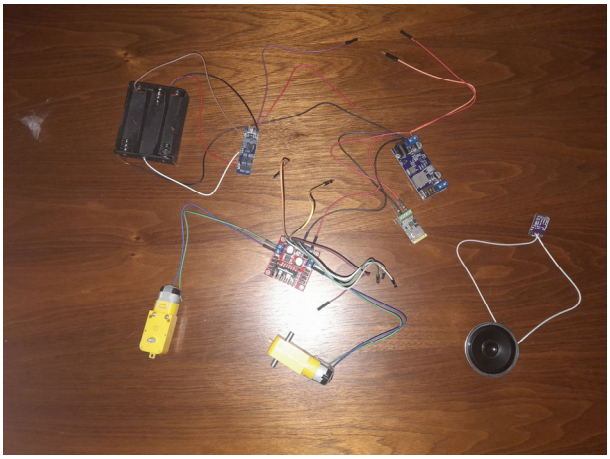
Εικόνα 4-3 Θήκες Κινητήρων



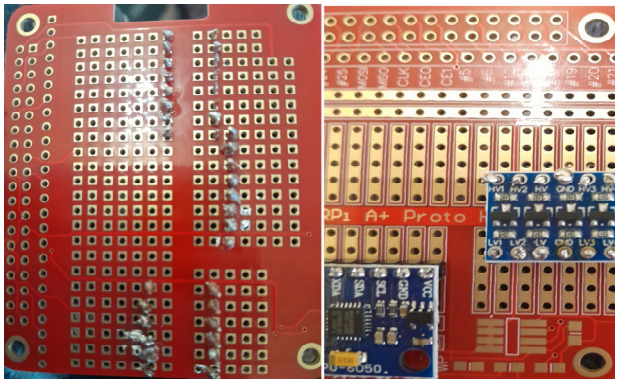
Εικόνα 4-4 Βάση Ηλεκτρονικών

4.2.1.2 Συναρμολόγηση Ηλεκτρονικών Εξαρτημάτων

Έχει πραγματοποιηθεί βάσει των οδηγιών του αρχείου [89] η συναρμολόγηση ενός μέρους των ηλεκτρονικών εξαρτημάτων όπως φαίνεται στην Εικόνα 4-5 και η συγκόλληση των περισσοτέρων επαφών και καλωδίων αυτών όπως φαίνεται στην Εικόνα 4-6.



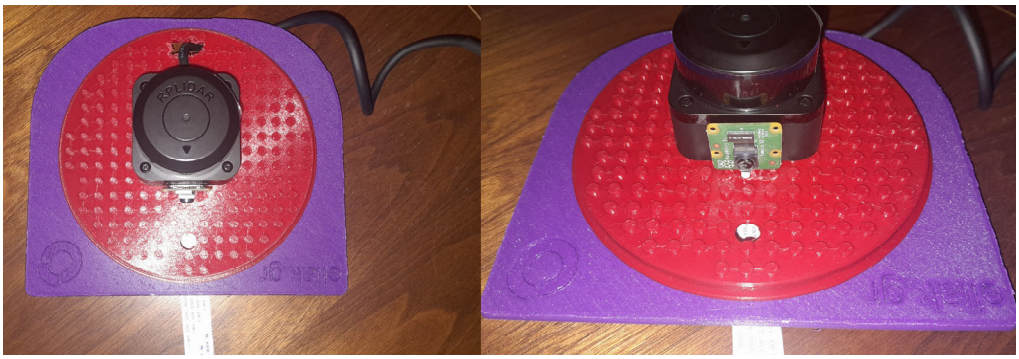
Εικόνα 4-5 Ηλεκτρονικά Μέρη



Εικόνα 4-6 Συγκόλληση Εξαρτημάτων

4.2.1.3 Ενσωμάτωση Προηγμένων Αισθητήρων

Επίσης ενσωματώθηκαν οι πρόσθετοι αισθητήρες **“RPLIDAR C1”** και **“Raspberry Pi Camera Module V2”** στο καπάκι της βάσης του ρομπότ όπως φαίνεται στην Εικόνα 4-7.



Εικόνα 4-7 Καπάκι Βάσης FOSSBot

4.2.2 Μελλοντικές Ενέργειες

Σε συνέχεια των παραπάνω ενεργειών οι μελλοντικές επεκτάσεις που μπορούν να γίνουν είναι οι παρακάτω.

1. Η ολοκλήρωση της συναρμολόγησης των ηλεκτρονικών εξαρτημάτων, της καλωδίωσης και της τροφοδοσίας όλων των αισθητήρων σύμφωνα με το [89].
2. Η εγκατάσταση του λογισμικού στο Raspberry Pi με τις εξής επιμέρους ενέργειες:
    a. Την εγκατάσταση του Ubuntu 22.04 το οποίο είναι απαραίτητο για την υποστήριξη του ROS2 Humble.
    b. Την εγκατάσταση του ROS2 Humble Hawksbill (LTS).
    c. Την ενσωμάτωση του λογισμικού του FOSSBot μέσα στο Raspberry, επιτρέποντας την απομονωμένη λειτουργία των υπηρεσιών και την εύκολη ενημέρωση του.
    d. Την εγκατάσταση των πακέτων των πρόσθετων αισθητήρων LiDAR και κάμερας και την ρύθμιση τους.
    e. Τη πραγματοποίηση σύνδεσης SSH για την εκτέλεση εντολών και για τον προγραμματισμό του ρομπότ.
    f. Την ρύθμιση των topics που θα δημοσιεύει το φυσικό FOSSBot ώστε να ταιριάζουν με αυτά που χρησιμοποιήθηκαν στην προσομοίωση του.
3. Η δοκιμή των σεναρίων χρήσης της χαρτογράφησης και της αυτόνομης πλοήγησης με τους διαφορετικούς αισθητήρες και αλγόριθμους που έγιναν στην προσομοίωση του FOSSBot στο φυσικό αυτή τη φορά ρομποτικό όχημα. Με αυτό τον τρόπο θα αξιολογηθούν οι παράγοντες που δεν προσομοιώνονται πλήρως όπως είναι η ολίσθηση των τροχών, ο θόρυβος των αισθητήρων και η ακρίβεια της οδομετρίας. Επίσης θα αξιολογηθεί η αποτελεσματικότητα των αλγορίθμων αυτών σε πραγματικές συνθήκες περιβάλλοντος και θα γίνει η προσαρμογή των παραμέτρων τους ώστε να ανταποκρίνονται στις δυνατότητες του Raspberry Pi Zero 2 W και στις καθυστερήσεις της επικοινωνίας και της επεξεργασίας των δεδομένων SLAM.

Συντομογραφίες

- **AGV** - Αυτοκινούμενο όχημα (Automated Guided Vehicle)
- **AMCL** - Προσαρμοστικός εντοπισμός θέσης Monte Carlo (Adaptive Monte Carlo Localization)
- **AMR** - Αυτόνομο κινητό ρομπότ (Autonomous Mobile Robot)
- **API** - Διεπαφή προγραμματισμού εφαρμογών (Application Programming Interface)
- **BT** - Δέντρο συμπεριφοράς (Behavior Tree)
- **CSI** - Σειριακή διεπαφή κάμερας (Camera Serial Interface)
- **DAE** - Μορφή αρχείου 3D (Digital Asset Exchange)
- **DDS** - Υπηρεσία διανομής δεδομένων (Data Distribution Service)
- **DIY** - Κάντο μόνος σου (Do It Yourself)
- **DWB -** Τοπικός Σχεδιαστής (Dynamic Window Approach variant)
- **EKF** - Εκτεταμένο φίλτρο Kalman (Extended Kalman Filter)
- **EOL** - Τέλος κύκλου ζωής (End-of-Life)
- **FOSS** - Ελεύθερο λογισμικό ανοικτού κώδικα (Free Open Source Software)
- **FOSSBot** - Εκπαιδευτικό ρομποτικό όχημα ανοικτού κώδικα (Free and Open Source Software Bot)
- **GPS** - Παγκόσμιο σύστημα εντοπισμού θέσης (Global Positioning System)
- **GUI** - Γραφική διεπαφή χρήστη (Graphical User Interface)
- **IMU** - Μονάδα αδρανειακής μέτρησης (Inertial Measurement Unit)
- **IoT** - Διαδίκτυο των πραγμάτων (Internet of Things)
- **LiDAR** - Ανίχνευση και μέτρηση απόστασης μέσω φωτός (Light Detection and Ranging)
- **LTS** - Μακροχρόνια υποστήριξη (Long-Term Support)
- **MCL -** Εντοπισμός θέσης Monte Carlo (Monte Carlo Localization)
- **MPC** - Μοντέλο προβλεπτικού ελέγχου (Model Predictive Control)
- **MSCKF** - Φίλτρο Kalman με περιορισμούς πολλαπλών καταστάσεων (Multi-State Constraint Kalman Filter)
- **Nav2** - Πακέτο αυτόνομης πλοήγησης (Navigation 2)
- **ODE** - Ανοιχτή Μηχανή Δυναμικής (Open Dynamics Engine)
- **PCB** - Τυπωμένη πλακέτα κυκλωμάτων (Printed Circuit Board)

- **PPA** - Προσωπικό αρχείο πακέτων (Personal Package Archive)
- **PWM** - Διαμόρφωση εύρους παλμού (Pulse Width Modulation)
- **RBPF** - Φίλτρο σωματιδίων τύπου Rao-Blackwellized
- **RGB-D** - Τύπος Κάμερας (Red-Green-Blue-Depth)
- **RMSE** - Σφάλμα ρίζας μέσου τετραγώνου (Root Mean Square Error)
- **ROS** - Λειτουργικό σύστημα ρομπότ (Robotic Operating System)
- **ROS2** - Λειτουργικό σύστημα ρομπότ 2η έκδοση (Robotic Operating System version 2)
- **SBC** - Υπολογιστής μίας πλακέτας (Single Board Computer)
- **SDF** - Μορφή αρχείου προσομοίωσης (Simulation Description Format)
- **SLAM** - Ταυτόχρονος εντοπισμός και χαρτογράφηση (Simultaneous Localization and Mapping)
- **SSH –** Τύπος απομακρυσμένης σύνδεσης (Secure Shell)
- **SSIM** - Δείκτης δομικής ομοιότητας (Structural Similarity Index)
- **STEM** - Επιστήμες, τεχνολογία, μηχανική και μαθηματικά (Science, Technology, Engineering, and Mathematics)
- **STL** - Μορφή αρχείου 3D (Standard Triangle Language)
- **TF2** - Βιβλιοθήκη μετασχηματισμών 2η έκδοση (Transform Library 2)
- **TOF** - Χρόνος πτήσης (Time of Flight)
- **UART** – Γενικός ασύγχρονος δέκτης-πομπός (Universal Asynchronous Receiver-Transmitter)
- **UAV** - Μη επανδρωμένο εναέριο όχημα (Unmanned Aerial Vehicle)
- **URDF** - Ενοποιημένη μορφή περιγραφής ρομπότ (Unified Robot Description Format)
- **V-SLAM** - Οπτικό SLAM (Visual SLAM)
- **WSL** - Υποσύστημα των Windows για Linux (Windows Subsystem for Linux)
- **WSL2** - Υποσύστημα των Windows για Linux 2η έκδοση
- **XML** - Επεκτάσιμη γλώσσα σήμανσης (Extensible Markup Language)
- **YAML** - YAML Ain't Markup Language (μορφή αρχείου ρυθμίσεων)
- **4WS** - Τετραδιεύθυνση (Four-Wheel Steering)

Βιβλιογραφία

[1] G. Yang and S. Hu, "Review of Robotics Technologies and Its Applications," 2023 International Conference on Advanced Robotics and Mechatronics (ICARM), Sanya, China, 2023, pp. 322-329, doi: 10.1109/ICARM58088.2023.10218815.

[2] G. Fragapane, R. de Koster, F. Sgarbossa, and J. O. Strandhagen, “Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda,” Eur. J. Oper. Res., vol. 294, no. 2, pp. 405–426, Apr. 2021, doi: 10.1016/j.ejor.2021.01.019.

[3] Y. Rivero-Moreno et al., “Robotic surgery: A comprehensive review of the literature and current trends,” Cureus, vol. 15, 2023, Art. no. e42370, doi: 10.7759/cureus.42370.

[4] P. Nepomuceno, W. H. Souza, M. Pakosh, K. E. Musselman, and B. C. Craven, “Exoskeleton-based exercises for overground gait and balance rehabilitation in spinal cord injury: A systematic review of dose and dosage parameters,” J. NeuroEngineering Rehabil., vol. 21, no. 1, p. 73, May 2024, doi: 10.1186/s12984-024-01365-2.

[5] S. Evripidou, K. Georgiou, L. Doitsidis, A. A. Amanatiadis, Z. Zinonos and S. A. Chatzichristofis, "Educational Robotics: Platforms, Competitions and Expected Learning Outcomes," in IEEE Access, vol. 8, pp. 219534-219562, 2020, doi: 10.1109/ACCESS.2020.3042555.

[6] A. Alam, "Educational Robotics and Computer Programming in Early Childhood Education: A Conceptual Framework for Assessing Elementary School Students’ Computational Thinking for Designing Powerful Educational Scenarios," 2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN), Villupuram, India, 2022, pp. 1-7, doi: 10.1109/ICSTSN53084.2022.9761354.

[7] G. C. Nugent, B. S. Barker, and N. Grandgenett, “The impact of educational robotics on student STEM learning, attitudes, and workplace skills,” in Robotics: Concepts, Methodologies, Tools, and Applications, I. Management Association, Ed. Hershey, PA, USA: IGI Global, 2014, pp. 1442–1459, doi: 10.4018/978-1-4666-4607-0.ch070.

[8] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, “The Scratch programming language and environment,” ACM Trans. Comput. Educ., vol. 10, no. 4, pp. 1–15, Nov. 2010, doi: 10.1145/1868358.1868363

[9] T. Ball, P. H. Hart, K. E. Stickler, S. D. Utting, and C. L. N. A. S. A. Project, “The BBC micro:bit – from the UK to the world,” *Commun. ACM*, vol. 62, no. 8, pp. 62–69, Jul. 2019, doi: 10.1145/3336759.

[10] E. S. Blockly, Google, “Blockly: A library for building visual programming editors,” 2020. [Online]. Available: https://developers.google.com/blockly. (Accessed: September 21, 2025).

[11] Open Robotics, "Gazebo Simulator," Gazebo. [Online]. Available: https://gazebosim.org/. [Accessed: 1 Sept 2025].

[12] Coppelia Robotics, "CoppeliaSim User Manual," Coppelia Robotics AG, 2024. [Online]. Available: https://manual.coppeliarobotics.com. (Accessed: September 21, 2025).

[13] M. Banzi and M. Shiloh, *Getting Started with Arduino*, 4th ed. Sebastopol, CA, USA: Maker Media, Inc., 2022.

[14] C. Chronis and I. Varlamis, “FOSSBot: An Open Source and Open Design Educational Robot,” Electronics, vol. 11, no. 16, article 2606, 2022. doi:10.3390/electronics11162606.

[15] EELLAK, “eellak/fossbot: An open source educational robot,” GitHub repository, 2025. Available: https://github.com/eellak/fossbot (Accessed: 1 Sept 2025)

[16] R. Amsters και P. Slaets, “Turtlebot 3 as a Robotics Education Platform,” Robotics in Education, Advances in Intelligent Systems and Computing, vol. 1132, pp. 170-181, 2020. doi:10.1007/978-3-030-26945-6_16

[17] LEGO Education, “LEGO Mindstorms EV3 Education Kit,” 2020. [Online]. Available: https://education.lego.com [Accessed: 1 Sept 2025]

[18] Robotnik Automation, “TugBot – Autonomous Mobile Robot for Indoor Logistics,” Tugbot.com, 2025. Available: https://tugbot.com [Accessed: 1 Sept 2025].

[19] MIT CSAIL, “MIT RACECAR: An Open-Source Autonomous Vehicle Platform for Education,” 2020. Available: https://mit-racecar.github.io [Accessed: 1 Sept 2025].

[20] H. Durrant-Whyte and T. Bailey, “Simultaneous Localization and Mapping (SLAM): Part I,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, June 2006, doi:10.1109/MRA.2006.1638022.

[21] M. Kalaitzidou and T. P. Pachidis, “Recent Robots in STEAM Education,” *Education Sciences*, vol. 13, no. 3, p. 272, 2023, doi: 10.3390/educsci13030272.

[22] E. R. Kazakoff, A. Sullivan, and M. U. Bers, "The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood," Early Childhood Educ. J., vol. 41, no. 4, pp. 245–255, 2013.

[23] C. Giannoulaki, A. Sioutis, G. Kypraios, G. Tsinarakis and L. Doitsidis, "ColourBot: A New Modular Tool for STEM Education," 2024 32nd Mediterranean Conference on Control and Automation (MED), Chania - Crete, Greece, 2024, pp. 400-405, doi: 10.1109/MED61351.2024.10566228.

[24] KiCad Developers. (2023). KiCad EDA. [Online]. Available: https://www.kicad.org. [Accessed: 1 Nov 2025].

[25] C. Pereira and The Ardublockly Contributors, “Ardublockly: A Visual Programming Editor for Arduino,” Embedded Log, 2024. [Online]. Available: https://ardublockly.embeddedlog.com. [Accessed: 1 Nov 2025].

[26] R. K. Senthilkumaran, "SMARTmBOT: A ROS2-based low-cost and open-source mobile robot platform," arXiv, 2022, doi: 10.48550/arXiv.2203.08903.

[27] N. Nakamoto and H. Kobayashi, "Development of an Open-source Educational and Research Platform for Autonomous Cars," IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, Lisbon, Portugal, 2019, pp. 6871-6876, doi: 10.1109/IECON.2019.8926794.

[28] J. F. M. Lee and J. A. Buitrago, "Map generation and localization for a LEGO NXT robot," 2015 IEEE 2nd Colombian Conference on Automatic Control (CCAC), Manizales, Colombia, 2015, pp. 1-5, doi: 10.1109/CCAC.2015.7345229.

[29] L. Paull et al., "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 1497-1504, doi: 10.1109/ICRA.2017.7989179.

[30] J. McCalip, M. Pradhan and K. Yang, "Reinforcement Learning Approaches for Racing and Object Avoidance on AWS DeepRacer," 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Torino, Italy, 2023, pp. 958-961, doi: 10.1109/COMPSAC57700.2023.00129.

[31] J. L. Cota, J. A. T. Rodríguez, B. G. Alonso and C. V. Hurtado, "Roadmap for development of skills in Artificial Intelligence by means of a Reinforcement Learning model using a DeepRacer autonomous vehicle," 2022 IEEE Global Engineering Education Conference (EDUCON), Tunis, Tunisia, 2022, pp. 1355-1364, doi: 10.1109/EDUCON52537.2022.9766659.

[32] Intel Corporation, "Reinforcement Learning Coach," Intel AI GitHub Pages, 2023. [Online]. Available: https://intellabs.github.io/coach/. [Accessed: 1 Nov 2025].

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv, 2017, doi: 10.48550/arXiv.1707.06347.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in Proc. 35th Int. Conf. Mach. Learn. (ICML), vol. 80, 2018, pp. 1856–1865, doi: 10.48550/arXiv.1801.01290.

[35] J. Betz et al., "F1TENTH: Enhancing Autonomous Systems Education Through Hands-On Learning and Competition," in IEEE Transactions on Intelligent Vehicles, vol. 10, no. 10, pp. 4770-4783, Oct. 2025, doi: 10.1109/TIV.2024.3495227.

[36] J. Vogel, "Tech explained: Ackermann steering geometry," Racecar Engineering, Apr. 6, 2021. [Online]. Available: Racecar Engineering. [Accessed: 1 Nov 2025].

[37] C. Rother, Z. Zhou and J. Chen, "Development of a Four-Wheel Steering Scale Vehicle for Research and Education on Autonomous Vehicle Motion Control," in IEEE Robotics and Automation Letters, vol. 8, no. 8, pp. 5015-5022, Aug. 2023, doi: 10.1109/LRA.2023.3291916.

[38] NVIDIA, “An autonomous AI racecar using NVIDIA Jetson Nano,” 2019. [Online]. Available: https://github.com/NVIDIA-AI-IOT/jetracer. [Accessed: 1 Nov 2025].

[39] Marvelmind Robotics, "Home - Marvelmind Robotics," Marvelmind. [Online]. Available: https://marvelmind.com/. [Accessed: 1 Dec 2025].

[40] F. D. Brunner, W. P. M. H. Heemels and F. Allgöwer, "Robust Event-Triggered MPC With Guaranteed Asymptotic Bound and Average Sampling Rate," in IEEE Transactions on Automatic Control, vol. 62, no. 11, pp. 5694-5709, Nov. 2017, doi: 10.1109/TAC.2017.2702646.

[41] J. S. De La Trinidad-Rendon, J. A. Reyes-Avendaño and H. G. Gonzalez-Hernandez, "A Comparative Study of Scaled Autonomous Vehicle Platforms for Research and Education," 2025 IEEE Mexican Humanitarian Technology Conference (MHTC), Puebla, Mexico, 2025, pp. 123-129, doi: 10.1109/MHTC65808.2025.11086476.

[42] B. Tribelhorn and Z. Dodds, "Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education," Proceedings 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 2007, pp. 1393-1399, doi: 10.1109/ROBOT.2007.363179.

[43] S. Kuzin and G. Sziebig, "SROS: Educational, Low-cost Autonomous Mobile Robot Design Based on ROS," 2020 IEEE/SICE International Symposium on System Integration (SII), Honolulu, HI, USA, 2020, pp. 1052-1057, doi: 10.1109/SII46433.2020.9025974.

[44] R. Amsters and P. Slaets, "Turtlebot 3 as a Robotics Education Platform," in Robotics in Education, vol. 1023, M. Merdan et al., Eds. Cham, Switzerland: Springer, 2020, pp. 170-181, doi: 10.1007/978-3-030-26945-6_16.

[45] M. T. Tamang, D. Maheriya, M. S. Sharif and T. Sutharssan, "Autonomous Navigation for TurtleBot3 Robots in Gazebo Simulation Environment," 2024 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Sakhir, Bahrain, 2024, pp. 568-574, doi: 10.1109/3ict64318.2024.10824668.

[46] G. Varghese et al., "Multi-Robot System for Mapping and Localization," 2023 8th International Conference on Robotics and Automation Engineering (ICRAE), Singapore, Singapore, 2023, pp. 79-84, doi: 10.1109/ICRAE59816.2023.10458504.

[47] P. Gaskell, J. Pavlasek, T. Gao, A. Narula, S. Lewis and O. C. Jenkins, "MBot: A Modular Ecosystem for Scalable Robotics Education," 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 2024, pp. 18294-18300, doi: 10.1109/ICRA57147.2024.10610587.

[48] D. Hagen, M. Mæland, T. Skotterud, M. Dahlseng Hermansen, H. Haij and M. Sauar Wad, "UiAbot: A Versatile Autonomous Mobile Robot Platform for Research and Education," 2024 IEEE 19th Conference on Industrial Electronics and Applications (ICIEA), Kristiansand, Norway, 2024, pp. 1-8, doi: 10.1109/ICIEA61579.2024.10664856.

[49] C. A. Berry, A. M. Larregola, K. Collins and J. McGee, "Robotics for the Streets: Open-Source Robotics for Academics," 2023 IEEE Frontiers in Education Conference (FIE), College Station, TX, USA, 2023, pp. 1-5, doi: 10.1109/FIE58773.2023.10343348.

[50] V. S. Babu and M. Behl, "f1tenth.dev - An Open-source ROS based F1/10 Autonomous Racing Simulator," 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), Hong Kong, China, 2020, pp. 1614-1620, doi: 10.1109/CASE48305.2020.9216949.

[51] E. Riazati, A. Hajizadeh, and S. Esmaeilzadeh, "Design and Implementation of a Modular ROS-Based Mobile Robot with Hierarchical Control," in Proc. 32nd Int. Conf. Electr. Eng. (ICEE), 2025, pp. 1038-1043, doi: 10.1109/ICEE67339.2025.11213761.

[52] Z. Jin and W. Zhou, "Design and Implementation of MobileRobot Navigation System Based on ROS Platform," 2022 International Conference on Automation, Robotics and Computer Engineering (ICARCE), Wuhan, China, 2022, pp. 1-4, doi: 10.1109/ICARCE55724.2022.10046597.

[53] T. Dupga, A. Deshwal, S. Navik, and A. Pal, "Integrated UWB and LIDAR Assisted Localization and Navigation System for Automated Platooning," in Proc. IEEE Int. Conf. Wireless Mob. Comput. Netw. Commun. (WiMob), 2025, pp. 1-6, doi: 10.1109/WiMob66857.2025.11257436.

[54] F. Chetouane, "Autonomous Mobile Robot Design and Testing for Data Center Monitoring Mission," in IEEE Access, vol. 13, pp. 162571-162587, 2025, doi: 10.1109/ACCESS.2025.3610346.

[55] Raspberry Pi Foundation, “Raspberry Pi Zero 2 W,” Raspberry Pi, 2021. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/. [Accessed: 1 Dec 2025].

[56] D. Nedelkovski, "L298N Motor Driver – Arduino Interface, How It Works, Codes, Schematics," HowToMechatronics, Aug. 8, 2017. [Online]. Available: https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/. [Accessed: 1 Dec 2025].

[57] DFRobot, "Micro DC Motor with Encoder-SJ01 SKU: FIT0450," DFRobot Product Wiki, 2019. [Online]. Available: https://wiki.dfrobot.com/Micro_DC_Motor_with_Encoder-SJ01_SKU__FIT0450. [Accessed: 1 Dec 2025].

[58] RucksikaaR, "MPU-6050 Sensor Module: A Friendly Introduction," Arduino Project Hub, Jun. 28, 2021. [Online]. Available: https://projecthub.arduino.cc/RucksikaaR/mpu-6050-sensor-module-a-friendly-introduction-7b4d4b. [Accessed: 1 Dec 2025].

[59] Waveshare, "Photo Interrupter Sensor," Waveshare Wiki. [Online]. Available: https://www.waveshare.com/wiki/Photo_Interrupter_Sensor. [Accessed: 1 Dec 2025].

[60] Ardumotive, "Ultrasonic Sensor HC-SR04 Arduino Tutorial," Ardumotive.com, Jul. 23, 2015. [Online]. Available: https://www.ardumotive.com/how-to-use-the-hc-sr04-ultrasonic-sensor-en.html. [Accessed: 1 Dec 2025].

[61] Wikipedia contributors, "Photoresistor," Wikipedia, The Free Encyclopedia, Oct. 3, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Photoresistor. [Accessed: 1 Dec 2025].

[62] Keyestudio, "Ks0028 keyestudio Photocell Sensor," Keyestudio Wiki. [Online]. Available: https://wiki.keyestudio.com/Ks0028_keyestudio_Photocell_Sensor. [Accessed: 1 Dec 2025].

[63] Waveshare, "Infrared Proximity Sensor," Waveshare Wiki. [Online]. Available: https://www.waveshare.com/wiki/Infrared_Proximity_Sensor. [Accessed: 1 Dec 2025].

[64] Grobotronics, "DATASHEET_ROBU.IN_.pdf," Grobotronics. [Online]. Available: https://grobotronics.com/images/companies/1/datasheets/DATASHEET_ROBU.IN_.pdf?1549637822086. [Accessed: 1 Dec 2025].

[65] MDCplus, "Best Free & Open Source Robotics Simulation and Path Planning Tools," MDCplus Blog, Dec. 11, 2025. [Online]. Available: https://mdcplus.fi/blog/top-free-robotic-simulation-path-planning-software/. [Accessed: 1 Dec 2025].

[66] Coppelia Robotics, "CoppeliaSim: The robot simulator for everyone," Coppelia Robotics AG, 2023. [Online]. Available: https://www.coppeliarobotics.com/. [Accessed: 1 Dec 2025].

[67] C. Chronis, "fossbot_simulation: A simulation of the FOSSbot educational robot for the Webots simulator," GitHub, 2023. [Online]. Available: https://github.com/chronis10/fossbot_simulation. [Accessed: 1 Dec 2025].

[68] Βικιπαίδεια, "LiDAR," Βικιπαίδεια, Η Ελεύθερη Εγκυκλοπαίδεια, Dec. 22, 2024. [Online]. Available: https://el.wikipedia.org/wiki/LiDAR. [Accessed: 1 Dec 2025].

[69] Slamtec, "RPLIDAR C1 360 Degree Laser Range Scanner with Fusion Technology Introduction and Datasheet (Model: C1M1)," Datasheet, 2024. [Online]. Available: https://grobotronics.com/images/SLAMTEC_rplidar_datasheet_C1_v1.1_en.pdf?1758286975117. [Accessed: 1 Dec 2025].

[70] Raspberry Pi Foundation, "Camera," Raspberry Pi Documentation, 2024. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html. [Accessed: 1 Dec 2025].

[71] Wikipedia contributors, "Robot Operating System" Wikipedia, The Free Encyclopedia, Jan. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System. [Accessed: 1 Dec 2025].

[72] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, Architecture, and Uses In The Wild," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2022, pp. 7326-7331, doi: 10.1109/ICRA46639.2022.9811702.

[73] S. A. S. Mohamed, M. -H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen and J. Plosila, "A Survey on Odometry for Autonomous Navigation Systems," in IEEE Access, vol. 7, pp. 97466-97486, 2019, doi: 10.1109/ACCESS.2019.2929133.

[74] J. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," IEEE Trans. Robot., vol. 39, no. 6, pp. 4333–4352, Dec. 2023, doi: 10.1109/TRO.2023.3248510.

[75] S. A. Memon et al., "Tracking Multiple Autonomous Ground Vehicles Using Motion Capture System Operating in a Wireless Network," in IEEE Access, vol. 12, pp. 61780-61794, 2024, doi: 10.1109/ACCESS.2024.3394536.

[76] Open Robotics, "ROS 2 Documentation," ROS.org. [Online]. Available: https://docs.ros.org/. [Accessed: 1 Dec 2025].

[77] Open Robotics, "ROS 2 Documentation: Humble Hawksbill," ROS.org. [Online]. Available: https://docs.ros.org/en/humble/index.html. [Accessed: 1 Dec 2025].

[78] Canonical Ltd., "Ubuntu 22.04.4 LTS (Jammy Jellyfish)," Ubuntu Releases. [Online]. Available: https://releases.ubuntu.com/jammy/. [Accessed: 1 Dec 2025].

[79] Canonical Ltd., “Ubuntu on WSL,” Ubuntu.com. [Online]. Available: https://ubuntu.com/desktop/wsl. [Accessed: 1 Dec 2025].

[80] Canonical Ltd., “How to install Ubuntu on WSL2,” *Ubuntu Documentation*. [Online]. Available: https://documentation.ubuntu.com/wsl/stable/howto/install-ubuntu-wsl2/. [Accessed: 1 Dec 2025].

[81] Open Robotics, “Install ROS 2 on Ubuntu Linux from Debian packages,” *ROS 2 Documentation*. [Online]. Available: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html. [Accessed: 1 Dec 2025].

[82] S. Macenski, “slam_toolbox,” *ROS Wiki*. [Online]. Available: https://wiki.ros.org/slam_toolbox. [Accessed: 1 Dec 2025].

[83] C. Apostolakis, "fossbot-hardware/designs," *GitHub repository*, commit e4feb35. [Online]. Available: https://github.com/chronis10/fossbot-hardware/tree/e4feb3521844e5f51492749b7b54cdb3e1aea9ac/designs. [Accessed: 1 Sept 2025].

[84] C. Apostolakis, "fossbot_simulation," *GitHub repository*. [Online]. Available: https://github.com/chronis10/fossbot_simulation. [Accessed: 1 Sept 2025].

[85] A. Porizou, "tugbot_ros2_pkgs/tugbot_description/meshes," *GitHub repository*, branch master. [Online]. Available: https://github.com/porizou/tugbot_ros2_pkgs/tree/master/tugbot_description/meshes. [Accessed: 1 Sept 2025].

[86] ROBOTIS, "TurtleBot3 Overview," *ROBOTIS e-Manual*. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Accessed: 1 Dec 2025].

[87] ROBOTIS, "turtlebot3_simulations," *GitHub repository*. [Online]. Available: https://github.com/ROBOTIS-GIT/turtlebot3_simulations. [Accessed: 1 Dec 2025].

[88] S. Macenski, "slam_toolbox/README.md," *GitHub repository*, branch ros2. [Online]. Available: https://github.com/SteveMacenski/slam_toolbox/blob/ros2/README.md. [Accessed: 1 Dec 2025].

[89] C. Chronis, "FOSSBOT Electronics Assembly Guide," *GitHub repository*, 2022. [Online]. Available: https://github.com/chronis10/fossbot-hardware/blob/35c8e0286c2f6f8589beedcc259d35103167e49e/electronics_instructions/draft_instructions.pdf. [Accessed: 1 Dec 2025].