Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών &
Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών

Διπλωματική Εργασία

Πλήρης κάλυψη γνωστού χώρου από επίγειο όχημα με χρήση στρατηγικών πλοήγησης και σημασιολογική ανάλυση του χάρτη

Εκπόνηση:
Μάλαμας Νικόλαος
ΑΕΜ: 8400

Επίβλεψη:
Αν. Καθ. Συμεωνίδης Ανδρέας
Δρ. Τσαρδούλιας Εμμανουήλ

Θεσσαλονίκη, Σεπτέμβριος 2019

Success is not final, failure is not fatal: it is the courage to continue that counts.
— Winston Churchill

The best way to predict the future is to invent it.
— Alan Kay

Ευχαριστιες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή κ. Ανδρέα Συμεωνίδη για την εμπιστοσύνη που μου έδειξε και την στήριξή του καθ' όλη τη διάρκεια της συνεργασίας μας στην εκπόνηση την διπλωματικής μου εργασίας. Θα ήθελα επίσης να απευθύνω τις ευχαριστίες μου στον Μεταδιδακτορικό Ερευνητή του τμήματος Δρ. Εμμανουήλ Τσαρδούλια, για την καθοδήγηση του, τις πολύτιμες συμβουλές και την εξαιρετική συνεργασία που είχαμε. Τέλος θα ήθελα να ευχαριστήσω τον ερευνητή του τμήματος Αλέξανδρο Φιλοθέου, για την βοήθεια και τη στήριξή του κατά τη διάρκεια εκπόνησης της εργασίας.

Το μεγαλύτερο ευχαριστώ το οφείλω στους γονείς μου Αριστοκλή και Ευφημία τις θυσίες και την αγάπη τους όλα αυτά τα χρόνια, δίνοντας μου κουράγιο να προχωρώ και να υπερπηδώ κάθε εμπόδιο για να φτάσω στον στόχο μου.

Τέλος, θα ήθελα να ευχαριστήσω συγγενείς, φίλους και γνωστούς για την όποια συνεισφορά τους τόσο σε προσωπικό όσο και σε επαγγελματικό επίπεδο.

Περίληψη

Ο κλάδος της ρομποτικής έχει εμφανίσει ραγδαία εξέλιξη τα τελευταία χρόνια. Η χρήση της αν και στην αρχή περιοριζόταν κυρίως σε στρατιωτικές εφαρμογές, σήμερα έχει γενικευτεί σε μια πληθώρα εφαρμογών τόσο επαγγελματικού όσο και καθημερινού περιεχομένου. Η σχέση των ανθρώπων με την ρομποτική συνεχώς αλλάζει και εξελίσσεται, σε μια προσπάθεια αυτή να βοηθήσει στην αποτελεσματικότερη αντιμετώπιση των διάφορων ανθρωπίνων προβλημάτων.

Ένας ολόκληρος τομέας των εφαρμογών της ρομποτικής αφορά την αυτόνομη πλοήγηση ρομποτικών οχημάτων σε γνωστούς ή και άγνωστους χώρους. Τέτοιες περιπτώσεις είναι τα αυτοκινούμενα αυτοκίνητα, ο αυτόματος καθαρισμός ενός χώρου από μια ρομποτική σκούπα, η αυτόνομη απογραφή προϊόντων σε μια αποθήκη, η χαρτογράφηση άγνωστων περιοχών κ.α.

Η παρούσα διπλωματική εργασία έχει ως στόχο την μελέτη και επίλυση του προβλήματος της αυτοματοποίησης της διαδικασίας απογραφής προϊόντων σε οποιονδήποτε γνωστό δισδιάστατο χώρο αποτελεσματικά. Το πρόβλημα αυτό περιλαμβάνει τρία υποπροβλήματα: α) τον διαχωρισμό του γνωστού χώρου σε υποχώρους, β) τον υπολογισμό της αλληλουχίας προσέγγισης των διάφορων υποχώρων, και γ) την εύρεση του μονοπατιού για την πλήρη κάλυψη του χώρου. Η κάλυψη αυτή, μάλιστα, πραγματοποιείται από αισθητήρες τα χαρακτηριστικά των οποίων δεν είναι πρότερα γνωστά.

Τα προβλήματα αυτά αντιμετωπίζονται κάνοντας χρήση του γνωστού δισδιάστατου χάρτη του χώρου. Αρχικά, υλοποιείται η τοπολογική ανάλυση του χώρου προκειμένου να εντοπιστούν τα διάφορα δωμάτια του περιβάλλοντος. Έπειτα, υπολογίζεται η βέλτιστη σειρά επίσκεψης αυτών των δωματίων. Τέλος, υπολογίζεται το βέλτιστο μονοπάτι κίνησης σε κάθε υποχώρο μέσω των οποίων το όχημα καλύπτει πλήρως το χάρτη. Τα κριτήρια αξιολόγησης της μελέτης είναι η καλύτερη δυνατή κάλυψη του χώρου και η ταχύτητα εκτέλεσης τόσο της διαδικασίας πλοήγησης όσο και των υπολογισμών. Επιπλέον, στην διπλωματική αυτή εργασία εξετάζονται δύο διαφορετικές στρατηγικές κάλυψης του χώρου που αφορούν τον τρόπο προσέγγισης του ρομποτικού οχήματος στα πιθανά σημεία εύρεσης προϊόντων.

Επιπρόσθετα, πραγματοποιήθηκαν πειράματα στα τρία τμήματα της διαδικασίας σε περιβάλλον προσομοίωσης τα αποτελέσματα των οποίων παρουσιάζονται. Χρησιμοποιήθηκαν χώροι με διαφορετική μορφολογία και αισθητήρες με διαφορετικά χαρακτηριστικά, με στόχο την ολοκληρωμένη αξιολόγηση της μελέτης.

Νικόλαος Μάλαμας
malamasn@ece.auth.gr
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών,
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, Ελλάδα
Σεπτέμβριος 2019

Title

Full Coverage of Known Area with Unmanned Ground Vehicle using Path Patterns and Semantic Map Annotation

Abstract

Over the last years a rapid growth of the robotics industry has been noticed. Despite the fact that they were primarily used for military applications, nowadays many robotic applications have emerged trying to help people to deal with both everyday and professional tasks.

An important field of robotics applications is the unmanned ground vehicle navigation in known or unknown environments. There is a vast variety of such systems that have already been developed, such as autonomous vehicles, automated house cleaning robots, autonomous real time inventorying, mapping unknown areas etc.

The present Diploma Thesis focuses on studying and solving the problem of the fast and optimal autonomous inventorying of any known 2D warehouse. This problem consists of three sub problems: a) the separation of the known area into subareas, b) the computation of the visiting sequence of these subareas, and c) the computation of the full coverage path in each subarea. The area coverage is accomplished using sensors with a priori unknown characteristics.

A 2D occupancy grid map representing the environment has been used, in order to face these problems. First, a topological analysis of the map is implemented to locate the area's different rooms, according to which the area is separated. Next, the optimal room sequence is computed. Then, a coverage path for each room is computed through many stages of optimization. The evaluation metrics of the process are the complete area coverage and the execution time of both the computations and the navigation. In addition, in the present Diploma Thesis two different navigation strategies have been developed and compared.

Finally, a series of experiments were carried out at each stage of the implementations in order to thoroughly test each part. Maps with different topologies and sensors with different configurations were used to obtain robust results and test the developed process. As for the experiments, they were solely executed in simulation environments.

Nikolaos Malamas
malamasn@ece.auth.gr
Electrical & Computer Engineering Department,
Aristotle University of Thessaloniki, Greece

September 2019

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος πινάκων

Κατάλογος Αλγορίθμων

Ακρωνύμια Εγγράφου

Παρακάτω παρατίθενται ορισμένα από τα πιο συχνά χρησιμοποιούμενα ακρωνύμια της παρούσας διπλωματικής εργασίας:

AMCL ⟶ Adaptive Monte Carlo Localization
GPS ⟶ Global Positioning System
GVD ⟶ Generalized Voronoi Diagram
HC ⟶ Hill Climbing
LiDAR ⟶ Light Detection And Ranging
LVQ ⟶ Learning Vector Quantization
OGM ⟶ Ocupancy Grid Map
RGBD ⟶ Red Green Blue Depth
ROS ⟶ Robot Operating System
RRHC ⟶ Random Restart Hill Climbing
SLAM ⟶ Simultaneous Localization And Mapping
SVM ⟶ Support Vector Machines
TSP ⟶ Traveling Salesman Problem

1 Εισαγωγή

Ο όρος *Ρομποτική* αναφέρεται στον κλάδο της επιστήμης που μελετά τις μηχανές εκείνες που μπορούν να αντικαταστήσουν τον άνθρωπο στην εκτέλεση μιας εργασίας, η οποία συνδυάζει τη φυσική δραστηριότητα με τη διαδικασία λήψης αποφάσεων. Οι μηχανές αυτές ονομάζονται ρομπότ και ένας ορισμός τους είναι ο επόμενος.

> Ρομπότ είναι ένας πράκτορας εφοδιασμένος με ένα πλήθος αισθητήρων και τελεστών, για να ερμηνεύει το περιβάλλον του και να κινείται μέσα σε αυτό, ενώ μπορεί να προγραμματιστεί για να παρουσιάσει νοημοσύνη.

Η ρομποτική, λοιπόν, έχει ώς στόχο τον προγραμματισμό αυτό των μηχανών προκειμένου να παρουσιάσουν αυτόνομα νοημοσύνη και να εκτελέσουν κάποια διεργασία χωρίς την ανθρώπινη εποπτεία.

Οι πρώτες προσπάθειες του ανθρώπου να κατασκευάσουν αυτόνομες μηχανές παρατηρούνται στην αρχαία Ελλάδα με πιο γνωστή την κατασκευή του μηχανισμού των Αντικυθήρων. Η ρομποτική, στη μορφή που έχει σήμερα, άρχισε να αναπτύσσεται στα μέσα του 20ού αιώνα με στόχο την αυτοματοποίηση της παραγωγικής διαδικασίας στις βιομηχανίες μαζικής παραγωγής. Όμως, δεν περιορίζεται πια μόνο σ' αυτόν τον τομέα. Πετυχημένα ρομποτικά συστήματα συναντά κανείς στην ιατρική, στις αγροτικές καλλιέργιες, στην εξερεύνηση του διαστήματος, στην διάσωση ανθρώπων από καταστροφές. Μάλιστα, γίνονται καθημερινά προσπάθειες χρήσης ρομποτικών συστημάτων σε όλο και περισσότερες και πιο πολύπλοκες διεργασίες σε μια προσπάθεια βελτίωσης της αποτελεσματικότητας τους.

Η καθολική χρήση της ρομποτικής εγείρει ορισμένα θέματα ηθικής. Ο Isaac Asimov διατύπωσε το 1942 τους τρεις νόμους της Ρομποτικής [1] οι οποίοι έθεσαν για πρώτη φορά ένα ηθικό πλαίσιο χρήσης της και βρίσκονται μέχρι και σήμερα σε συνεχή μελέτη και αναθεώρηση. Οι νόμοι είναι οι εξής:

- Ένα ρομπότ δεν πρέπει να βλάπτει ένα ανθρώπινο ον, ή να επιτρέψει σε ένα ανθρώπινο ον να τραυματιστεί μέσω αδράνειάς του.
- Ένα ρομπότ πρέπει να υπακούει σε εντολές που του δίνονται απο ανθρώπινα όντα, εκτός αν αυτές οι εντολές παραβαίνουν τον πρώτο νόμο.
- Ένα ρομπότ πρέπει να προστατεύεται, εκτός απο τις περιπτώσεις που παραβιάζεται ο πρώτος ή ο δεύτερος νόμος.

1.1 Περιγραφη του Προβληματος

Η χρήση των *Μη Επανδρομένων Επίγειων Οχημάτων* αυξάνεται συνεχώς με όλο και περισσότερες εφαρμογές τα τελευταία χρόνια. Κάποιες από αυτές είναι οι εξής:

- Εύρεση του χάρτη ενός δυσπρόσιτου χώρου (SLAM), όπως ένα ορυχείο.
- Αναζήτηση ανθρώπων σε χώρους που έχουν υποστεί φυσικές καταστροφές.
- Αυτόνομη μεταφορά και παράδοση δεμάτων.
- Αυτοματοποιημένη απογραφή προϊόντων σε αποθηκευτικούς χώρους.
- Αποτελεσματικός καθαρισμός του πατώματος ενός σπιτιού.
- Χρήση τους ως μέσα μεταφοράς χωρίς οδηγό.
- Χρήση τους ως ξεναγός ή βοηθός σε χώρους με πολύ κόσμο, όπως μουσεία, αεροδρόμια κ.α.

Αυτές οι εφαρμογές επιτυγχάνονται, διότι τα οχήματα αυτά φέρουν κατάλληλους αισθητές που είναι προσαρμοσμένοι στην κάθε περίπτωση, αλλά και γιατί έχουν αναπτυχθεί πολλοί αποτελεσματικοί αλγόριθμοι πλοήγησης. Έτσι, τα ρομπότ είναι σε θέση να δημιουργήσουν μια αντίληψη για τον περιβάλλοντα χώρο και να μάθουν να κινούνται μέσα σ' αύτον εκπληρώνοντας τον εκάστοτε στόχο τους.

Στην διπλωματική αυτή εργασία πραγματοποιείται μια αναλυτική μελέτη της αυτόνομης απογραφής των προϊόντων σε έναν οποιονδήποτε γνωστό χώρο. Η απογραφή προϊόντων είναι μια διαδικασία η οποία μπορεί να εκτελεστεί από ρομποτικούς πράκτορες, αντί για ανθρώπους. Αποτελεί μια καθορισμένη διαδικασία, η οποία δεν απαιτεί ανθρώπινες ικανότητες που δεν μπορούν να υλοποιηθούν από ένα ρομπότ, όπως είναι η λήψη πολύπλοκων αποφάσεων και η εκτέλεση σύνθετων κινήσεων. Έτσι, το ανθρώπινο δυναμικό θα έχει την δυνατότητα να ασχοληθεί με πιο πολύπλοκες και απαιτητικές διεργασίες την ώρα που ένα ρομποτικό όχημα μπορεί να φέρει εις πέρας την διαδικασία αυτή και να καταγράψει την θέση όλων των αποθηκευμένων προϊόντων με ακρίβεια μερικών εκατοστών.

Το πρόβλημα που μελετάται διακρίνεται σε τρία υποπροβλήματα: α) τον διαχωρισμό του χώρου σε υποχώρους (map annotation/decomposition), β) τον υπολογισμό της βέλτιστης ακολουθίας επίσκεψης τους και, γ) την εύρεση του βέλτιστου μονοπατιού σε κάθε υποχώρο (path planning). Ο διαχωρισμός του χώρου πραγματοποιείται προκειμένου να απλοποιηθεί το πρόβλημα υπολογισμού του βέλτιστου μονοπατιού σε πολλά μικρότερα, καθώς η προσπάθεια εύρεσης ενός συνολικού μονοπατιού θα

ήταν υπολογιστικά απαγορευτική. Επιπλέον, σε κάθε υποχώρο υπολογίζεται μια αλληλουχία σημείων τα οποία πρέπει να προσπελάσει το όχημα, ώστε να καλύψει πλήρως τον χώρο. Τα κριτήρια αξιολόγησης των διάφορων μεθόδων που μελετώνται είναι ο συνολικός χρόνος πλοήγησης και το τελικό ποσοστό κάλυψης του χώρου στο οποίο μπορούν να συναντηθούν προϊόντα.

Η προσέγγιση του προβλήματος είναι ανεξάρτητη τόσο του χώρου όσο και των αισθητήρων κάλυψης που φέρει το όχημα. Ο χώρος τον οποίο πρέπει να καλύψει το όχημα είναι γνωστός και βρίσκεται σε μορφή *χάρτη - πλέγμα πιθανοτικής κάλυψης* (Occupancy Grid Map - OGM) 3.1. Επίσης, οι αισθητήρες κάλυψης είναι RFID κεραίες και τα χαρακτηριστικά τους δίνονται από τον χρήστη. Όλοι οι υπολογισμοί των μονοπατιών είναι παραμετρικοί αυτών. Η ανεξαρτησία αυτή στοχεύει σε μια γενικευμένη λύση, δίχως εξαρτήσεις σε στοιχεία που εκ των πραγμάτων δεν είναι πρότερα γνωστά, στην περίπτωση της πραγματικής εφαρμογής της μεθόδου αυτής.

1.2 Σκοπος - Συνεισφορα της Διπλωματικης Εργασιας

Στόχος αυτής της διπλωματικής εργασίας είναι η παρουσίαση ενός ολοκληρωμένου συστήματος ανάλυσης του χάρτη του χώρου, υπολογισμού του βέλτιστου μονοπατιού πλοήγησης συναρτήσει των αισθητήρων που φέρει το ρομπότ και πλήρους κάλυψης του χώρου χρησιμοποιώντας ένα μη επανδρωμένο επίγειο όχημα.

Η ανάλυση του χώρου πραγματοποιείται βρίσκοντας την τοπολογία του χώρου, κάνοντας χρήση του χάρτη σε μορφή OGM. Έτσι, ο χώρος διαχωρίζεται σε επιμέρους τμήματα, τα οποία και μελετούνται στη συνέχεια ως ξεχωριστές οντότητες.

Η εύρεση του βέλτιστου μονοπατιού περιλαμβάνει τον υπολογισμό της βέλτιστης ακολουθίας κάλυψης των δωματίων. Αυτό επιτυγχάνεται χρησιμοποιώντας μια παραλλαγή του αλγορίθμου αναρρίχησης λόφων (Hill-Climbing).

Στη συνέχεια για κάθε δωμάτιο υπολογίζονται θέσεις στον χώρο απ' όπου το όχημα θα έχει πρόσβαση στα προϊόντα με ομοιόμορφη δειγματοληψία πολλαπλών βημάτων και υπολογίζεται η βέλτιστη ακολουθία τους. Για την εύρεση της βέλτιστης ακολουθίας χρησιμοποιούνται παραλλαγές του αλγορίθμου αναρρίχησης λόφων (Hill-Climbing) σε συνδιασμό με τον αλγόριθμο εύρεσης πλησιέστερου γείτονα (Nearest Neighbor). Μετά, υπολογίζονται οι καλύτεροι δυνατοί προσανατολισμοί του οχήματος σε κάθε θέση με στόχο την καταγραφή όσων περισσότερων προϊόντων γίνεται και ταυτόχρονα την ελαχιστοποίηση του μήκους του συνολικού μονοπατιού.

Έπειτα, πραγματοποιείται μια μείωση του πλήθους των σημείων τα οποία έχει να επισκεφθεί το όχημα, αφαιρώντας τα περιττά, προκειμένου να μειωθεί ο συνολικός χρόνος πλοήγησης, χωρίς όμως να επηρεάζει το συνολικό ποσοστό κάλυψης του χώρου.

Στην εργασία αυτή χρησιμοποιούνται δύο διαφορετικές κύριες στρατηγικές προσέγγισης των σημείων αυτών του χώρου. Η πρώτη αποτελεί μια προσέγγιση της στρατηγικής ακολουθίας των τοίχων του χώρου (wall following), καθώς σε αυτούς βρίσκονται τα προϊόντα. Στην δεύτερη δημιουργούνται κινήσεις ζιγκ-ζαγκ προκειμένου να σαρώνουν οι αισθητήρες τον χώρο περισσότερες φορές και από διαφορετικές γωνίες.

Τέλος, ο κώδικας που αναπτύχθηκε εκτελέσθηκε σε περιβάλλον προσομοίωσης

τόσο κατά τη διάρκεια σχεδιασμού των αλγορίθμων για την επιμέρους βελτίωση, όσο και συνολικά στο πέρας της μελέτης.

1.3 Διαρθρωση της Αναφορας

Η διάρθρωση της παρούσας διπλωματικής εργασίας είναι η εξής:

- **Κεφάλαιο 2**: Γίνεται ανασκόπηση της ερευνητικής περιοχής που αφορά την επιλογή κατάλληλων αισθητήρων, την τοπολογική ανάλυση ενός χώρου και την πλήρη κάλυψη του.
- **Κεφάλαιο 3**: Περιγράφονται βασικά θεωρητικά στοιχεία στα οποία βασίστηκαν οι υλοποιήσεις.
- **Κεφάλαιο 4**: Παρουσιάζονται τα εργαλεία που χρησιμοποιήθηκαν στις υλοποιήσεις.
- **Κεφάλαιο 5**: Αναλύονται οι υλοποιήσεις που αναπτύχθηκαν.
- **Κεφάλαιο 6**: Παρουσιάζεται αναλυτικά η μεθοδολογία των πειραμάτων και τα αποτελέσματα.
- **Κεφάλαιο 7**: Παρουσιάζονται τα τελικά συμπεράσματα.
- **Κεφάλαιο 8**: Αναφέρονται τα προβλήματα που προέκυψαν και προτείνονται θέματα για μελλοντική μελέτη, αλλαγές και επεκτάσεις.

2

Επισκόπηση της Ερευνητικής Περιοχής

Η επιστημονική κοινότητα μελετά σε βάθος την χρήση μη επανδρωμένων επίγειων οχημάτων για πληθώρα εφαρμογών. Μια από αυτές είναι η αυτοματοποίηση της απογραφής των προϊόντων στις αποθήκες. Υπάρχουν πολλές δημοσιεύσεις που επιλύουν τα προαναφερθέντα προβλήματα αυτής της εφαρμογής χρησιμοποιώντας διαφορετικά εργαλεία. Θα εξεταστούν στη συνέχεια προτάσεις άλλων ερευνητών όσον αφορά την επιλογή των αισθητήρων που θα φέρει το όχημα, τον διαχωρισμό ενός χώρου σε δωματια και την πλήρη κάλυψη του.

2.1 Επιλογη Αισθητηρων

Το πρώτο και σημαντικότερο πρόβλημα είναι η επιλογή των σωστών αισθητήρων για την εφαρμογή αυτη. Δίχως σωστούς αισθητήρες η συνολική διαδικασία της κάλυψης του χώρου θα περιέχει σφάλματα, ανεξάρτητα από την υπόλοιπη διαδικασία. Συνεπώς, η επιλογή κατάλληλων αισθητήρων χρίζει προσοχής.

Μια μέθοδος που έχει αναπτυχθεί στο παρελθόν είναι η χρήση ετικετών *barcode*. Η τεχνολογία αυτή μελετήθηκε από τον Emery [2] για τη βελτίωση του χρόνο και του κόστους της διαδικασίας της καταγραφής προϊόντων στις αποθήκες. Η ανάγνωση τους μπορεί να πραγματοποιηθεί είτε με την επεξεργασία της εικόνας μιας κάμερας είτε με τη χρήση laser ανάγνωσης barcodes (barcode scanner). Και οι δύο αυτοί αισθητήρες, όμως, βασίζονται στην οπτική επαφή τους με το εκάστοτε προϊόν, πράγμα που συχνά είναι αδύνατο.

Η χρήση *RFID κεραιών* για την καταγραφή αντικειμένων σε εσωτερικούς χώρους προτάθηκε από τον Tesoriero κ.α. [3] μαζί με ένα σύστημα εντοπισμού θέσης σε κλειστούς χώρους, σε μια προσπάθεια αντικατάστασης του GPS το οποίο δεν λειτουργεί σωστά σ' αυτές τις περιπτώσεις. Το σύστημα που προτείνουν συσχετίζει τον αριθμό της ετικέτας RFID που διαβάζει η κεραία με την τρέχουσα θέση του ρομπότ. Έτσι, μπορεί να αποθηκευτεί το σύνολο των αντικειμένων που συνά-

ντησε κατά την περιήγηση του το όχημα και να πραγματοποιηθεί μια ολοκληρωμένη απογραφή. Η μέθοδος αυτή είναι αρκετά ακριβής και οικονομικά φιλική για κάθε αποθήκη, ενώ δεν απαιτεί την οπτική επαφή μεταξύ κεραιών και προϊόντων.

2.2 Εξαγωγη Πληροφοριας Δωματιων και Διαδρομων

Ο διαχωρισμός ενός OGM ενός χώρου μπορεί να γίνει με διάφορες προσεγγίσεις όπως προτείνεται από τον Bormann κ.ά. [4]. Αυτές είναι ο μορφολογικός διαχωρισμός, ο διαχωρισμός με βάση τις αποστάσεις, ο διαχωρισμός εξάγοντας χαρακτηριστικά και ο διαχωρισμός βασιζόμενος στο διάγραμμα Voronoi (Generalized Voronoi Diagram - GVD). Σε κάθε περίπτωση χρησιμοποιούνται διαφορετικά στοιχεία από τον εκάστοτε χάρτη.

Πιο αναλυτικά, στον μορφολογικό διαχωρισμό ο OGM μετασχηματίζεται σε δυαδική εικόνα και ένας μορφολογικός τελεστής διάβρωσης περνάει επαναληπτικά από αυτήν. Περιοχές που κατά τη διαδικασία αυτή αποκόπτονται από την ενιαία περιοχή αποτελούν διαφορετικά δωμάτια. Στον διαχωρισμό βάση των αποστάσεων υπολογίζεται ένας χάρτης αποστάσεων του κάθε σημείου με το κοντινότερο του και ο OGM μετασχηματίζεται σε δυαδική εικόνα. Στη συνέχεια, επαναληπτικά χρησιμοποιείται ένα κατώφλι απόστασης, το οποίο σταδιακά αυξάνεται. Ο αλγόριθμος συγκρίνει κάθε σημείο με το κατώφλι, κρατάει όσα σημεία είναι μικρότερα του και αναζητά περιοχές οι οποίες αποκόπτονται από την ενιαία περιοχή του ελεύθερου χώρου που αποτελούν τα διάφορα δωμάτια. Ο διαχωρισμός εξάγοντας χαρακτηριστικά προσομοιώνει σε κάθε σημείο του χώρου μια μέτρηση ενός αισθητήρα απόστασης (laser) και ανάλογα με τις μετρήσεις που υπολογίζει κατηγοριοποιεί το σημείο ως τμήμα ενός διαδρόμου ή ενός δωματίου. Τέλος, στην περίπτωση χρήσης του GVD υπολογίζονται κρίσιμα σημεία τα οποία μπορεί να αποτελούν σημεία ύπαρξης πόρτας και, επομένως, διαχωρισμού του χώρου.

Μια άλλη προσέγγιση προτείνεται από τον Brown [5] και περιλαμβάνει την εύρεση σημείων στένωσης του διδιάστατου χώρου και προσπαθεί να χωρίσει τον χώρο βασιζόμενο στα σημεία αυτά. Τα σημεία αυτά προέρχονται από μια παραλλαγή του GVD, που όμως όλα τα τμήματα του διαγράμματος είναι ευθύγραμμα τμήματα. Έτσι, ξεκινάει από τα σημεία αυτά και απομακρύνεται προσπαθώντας να καλύψει ολόκληρο τον χώρο. Δημιουργούνται, δηλαδή, μέτωπα αναζήτησης από το κέντρο περίπου των δωματίων προς τις άκρες. Κάθε σύγκρουση των μετώπων αυτών σημαίνει ότι συναντώνται διαφορετικά δωμάτια, τα οποία και διαχωρίζονται.

Σε ορισμένες περιπτώσεις ταυτόχρονα με τον διαχωρισμό των δωματίων υλοποιείται και η κατηγοριοποίηση τους. Ο συνήθης διαχωρισμός είναι σε δωμάτια, διαδρόμους και σημεία πόρτας. Τα σημεία των πορτών είναι αυτά που διαχωρίζουν τελικά τον χώρο, ενώ η χρήση των ετικετών *διάδρομος* και *δωμάτιο* δίνει την δυνατότητα σε ένα ρομποτικό όχημα να μπορεί να έχει διαφορετική συμπεριφορά σε κάθε περίπτωση.

Ο Kaleci κ.ά. [6] και ο O. M. Mozos [7] κατηγοριοποιούν κάθε ελεύθερο σημείο του χώρου σε τρεις κλάσεις, δωμάτιο, διάδρομος και πόρτα. Το ρομπότ κατευθύνεται σε κάθε σημείο του χώρου και καταγράφει μια μέτρηση του LiDAR του και χρησιμοποιεί την μέτρηση ως είσοδο ενός ταξινομητή K-means ή Learning Vector

Quantization (LVQ). Ο K-means ταξινομεί με μεγάλη ακρίβεια τα δωμάτια και τους διαδρόμους, ενώ αποτυγχάνει να προβλέψει σωστά τις πόρτες. Αντίθετα, ο LVQ παρουσιάζει μια σημαντική βελτίωση στο ποσοστό εύρεσης σημείων πόρτας, χωρίς να μειώνει σημαντικά τα ποσοστά επιτυχίας των άλλων δύο κλάσεων.

Ο Filliat κ.ά. στο [8] χρησιμοποιούν μια RGBD κάμερα κατά την εξερεύνηση του χώρου. Χρησιμοποιούν την εικόνα βάθους προκειμένου να διαχωρίσουν τα διάφορα δωμάτια, να ανακαλύψουν τις συσχετίσεις μεταξύ τους και να εντοπίσουν αντικείμενα σε κάθε δωμάτιο. Αυτό επιτυγχάνεται με αλγορίθμους υπολογιστικής όρασης και μηχανικής μάθησης σε συνδιασμό και το αποτέλεσμα είναι ένας υψηλού επιπέδου χάρτης, παρόμοιος με αυτούς που θα δημιουργούσε νοητικά ένας άνθρωπός.

2.3 Πλήρης Κάλυψη Χώρου

Η πλήρης κάλυψη ενός χάρτη OGM, αλλά και ενός χώρου γενικότερα, συνεπάγεται την κάλυψη κάθε σημείου του χώρου του χάρτη αυτού. Όπως διατύπωσαν οι Galceran και Carreras [9] υπάρχουν έξι προϋποθέσεις για να καλυφθεί βέλτιστα ένας χώρος.

- Το ρομπότ πρέπει να περάσει απ' όλα τα σημεία του χώρου, καλύπτοντας τα πλήρως.
- Το ρομπότ πρέπει να καλύψει την περιοχή δίχως την ύπαρξη αλληλοεπικαλυπτόμενων περιοχών.
- Απαιτούνται συνεχείς και ακολουθιακές διεργασίες, δίχως επαναλήψεις μονοπατιών.
- Το ρομπότ πρέπει να αποφεύγει τα εμπόδια του περιβάλλοντος.
- Πρέπει να χρησιμοποιούνται όσο το δυνατόν πιο απλές τροχιές κίνησης, όπως ευθείες.
- Προτιμάται ένα βέλτιστο μονοπάτι, εαν αυτό υπάρχει.

Όμως, το πρόβλημα εύρεσης της βέλτιστης διαδρομής ενός συνόλου σημείων αποτελεί *το πρόβλημα του περιπλανώμενου πωλητή* (Traveling Salesman Problem - TSP) και είναι NP-hard πρόβλημα. Γι' αυτό έχουν αναπτυχθεί απόλυτες μέθοδοι που διαχωρίζουν τον χώρο σε μικρότερα χωρία και επιλύουν βέλτιστα το πρόβλημα της κάλυψης άνα χωρίο, αλλά και ευριστικές μέθοδοι που προσπαθούν να απλοποιήσουν σημαντικά το πρόβλημα και πολλές φορές δεν φτάνουν σε μια βέλτιστη λύση.

Μια από τις πρώτες προσεγγίσεις που δημοσιεύτηκαν είναι των Choset και Pignon [10]. Αυτοί πρότειναν την μέθοδο της αποσύνθεσης του χώρου με την κινήση boustrophedon, την κίνηση δηλαδή που κάνει ένα βόδι σε ένα χωράφι. Οι περιοχές οι οποίες περιλαμβάνουν εμπόδια δεν επιτρέπουν την ύπαρξη μιας συνεχόμενης διαδρομής. Γι' αυτό, ο χώρος χωρίζεται σε επιμέρους τμήματα και στο εσωτερικό τους πραγματοποιείται η συγκεκριμένη κίνηση. Έτσι, καλύπτονται τα σημεία του χώρου με απλές κινήσεις του ρομποτικού οχήματος προς τα μπροστά και προς τα πίσω. Μόλις καλυφθεί κάθε σημείο, αυτομάτως έχει καλυφθεί ολόκληρος ο χώρος.

Στο [5] χρησιμοποιείται μια παραλλαγή της boustrophedon μεθόδου. Εδώ σε κάθε χώρο σχηματίζεται μια σπειροειδής διαδρομή που ξεκινάει από το κέντρο του δωματίου και τερματίζει στην πόρτα του. Μετά συνδιάζονται όλες οι διαδρομές των δωματίων, ώστε το συνολικό μονοπάτι να είναι βέλτιστο.

Όπως αναφέρεται στο [9], υπάρχουν και άλλες τεχνικές διαχωρισμού του χώρου εκτός του boustrophedon. Υπάρχει ο διαχωρισμός Morse, ο τραπεζοειδής διαχωρισμός, ο διαχωρισμός Morse κάνοντας χρήση και του GVD κ.α.

Οι ευριστικές μέθοδοι χρησιμοποιούν διάφορα μοτίβα για να καλύψουν πλήρως τον χώρο. Οι Faud και Purwanto [11] χρησιμοποιώντας την εικόνα της κάμερας του ρομπότ το κατευθύνουν να ακουλουθεί τους τοίχους των διαδρόμων (wall following) διατηρώντας από αυτούς μια σταθερή απόσταση. Έτσι, πραγματοποιεί έμμεσα την πλήρη κάλυψη αυτών των χώρων.

3
Θεωρητικό Υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζονται ορισμένα βασικά θεωρητικά στοιχεία για την πλήρη κατανόηση της διπλωματικής αυτής εργασίας. Πρώτα, γίνεται αναφορά στα OGM και πώς αυτά αποθηκεύουν την πληροφορία του χώρου. Μετά αναλύεται το Γενικευμένο Διάγραμμα Voronoi. Τέλος, παρουσιάζονται οι αλγόριθμοι Brushfire, Dijkstra και Hill Climbing.

3.1 Occupancy Grid Map (OGM)

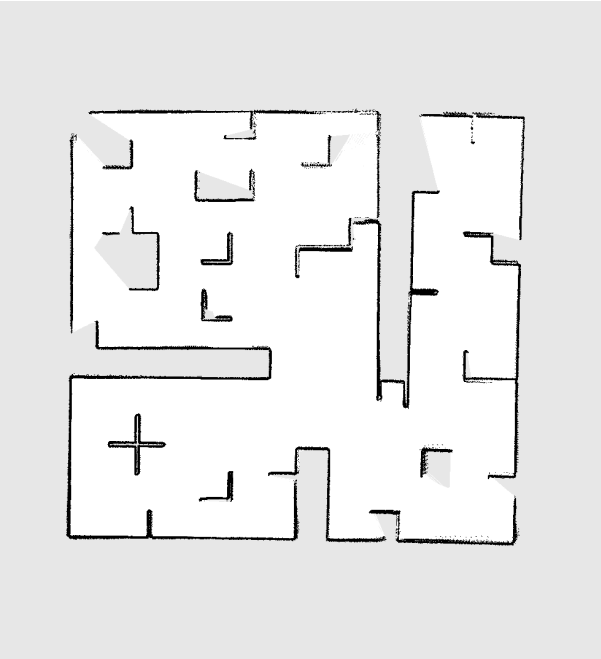
Η αναπαράσταση του χώρου μετά από την εφαρμογή του αλγορίθμου SLAM[1] πραγματοποιείται με την χρήση *χαρτών πλέγματος κατάληψης* (Occupancy Grid Maps) στους διδιάστατους χώρους, όπως περιγράφεται από τους Wolf και Sukhatme [12]. Αναπαρτιστά τον χώρο ως ένα σύνολο τετραγωνικών κελιών, στα οποία αποθηκεύεται πιθανότητα η κατάληψης τους. Το κάθε κελί μπορεί να πάρει τρεις διαφορετικές καταστάσεις που αντιστοιχούν στις παρακάτω τιμές:

- 100 όταν είναι κατειλημμένο (obstacle).
- 0 όταν είναι ελεύθερο (free space).
- -1 όταν είναι άγνωστο το περιεχόμενο του σημείου (unknown).

Κατά τη διαδικασία του SLAM διαβάζονται οι μετρήσεις του αισθητήρα που χρησιμοποιείται και όσο αυτός δείχνει ότι μια θέση είναι ελεύθερη τόσο η τιμή του κελιού της στο OGM θα μειώνεται έως και την τιμή 0, ενώ αντίστοιχα όσο θα δείχνει ότι μια θέση είναι κατειλημμένη, τόσο η τιμή στο κελί του OGM θα αυξάνεται μέχρι την τιμή 100. Με αυτόν τον τρόπο ορίζεται η πιθανότητα κατάληψης των κελιών.

Στο σχήμα 3.1 παρουσιάζεται ένα OGM ενός χώρου που προέκυψε μετά από προσομοίωση SLAM στο Gazebo. Τα εμπόδια έχουν μαύρο χρώμα, ο ελεύθερος χώρος άσπρο και ο άγνωστος γκρι.

[1] http://wiki.ros.org/gmapping



Σχήμα 3.1: Παράδειγμα OGM εικονικού χώρου

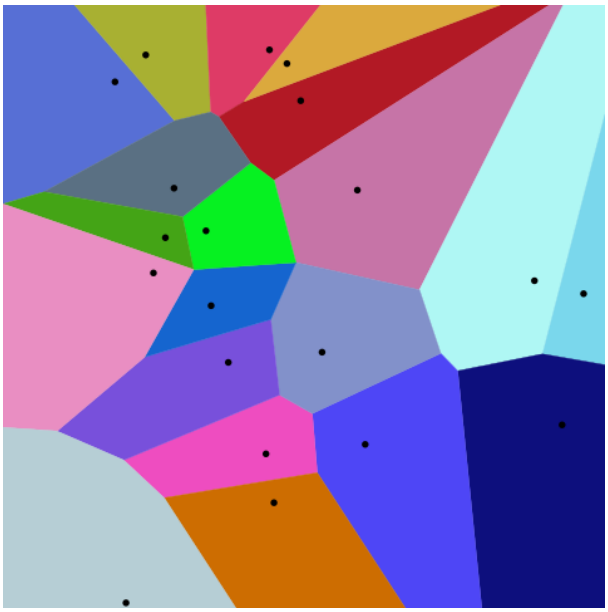
3.2 Generilized Voronoi Diagram (GVD)

Το *Γενικευμένο Διάγραμμα Voronoi* αποτελεί μια μέθοδο διαχωρισμού ενός χώρου σε επιμέρους περιοχές. Ο διαχωρισμός αυτός πραγματοποιείται με βάση τις αποστάσεις κάθε σημείου του χώρου από ένα υποσύνολο σημείων, σαφώς καθορισμένων από την αρχή της διαδικασίας. Οι τελικές περιοχές ονομάζονται κελιά Voronoi. Στο σχήμα 3.2 παρουσιάζεται ένα παράδειγμα ευκλείδιου διαχωρισμού του χώρου, στον οποίο τα όρια διαχωρισμού των περιοχών είναι ευθύγραμμα τμήματα.

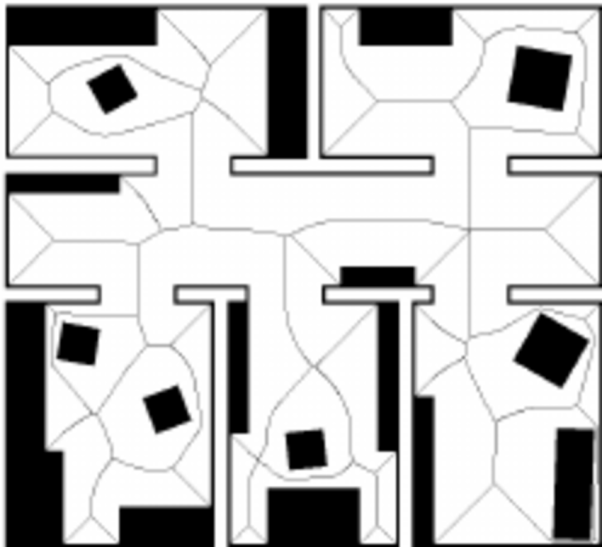
Το GVD είναι χρήσιμο στις ρομποτικές εφαρμογές, καθώς ο πραγματικός χώρος αναπαριστάται με μια διακριτή αναπαράσταση, τα OGM 3.1. Έτσι, μπορούν να υλοποιηθούν αλγόριθμοι εύρεσης του GVD του OGM ενός χώρου. Η πιο σημαντική περίπτωση είναι αυτή της χρήσης ως υποσύνολο σημείων όλα τα σημεία του OGM τα οποία αντιστοιχούν σε εμπόδια στον χώρο. Τότε σχηματίζεται ένα διάγραμμα που αποτελείται από τα πιο ασφαλή για την πλοήγηση του ρομποτικού οχήματος σημεία, καθώς αυτά απέχουν τις μέγιστες δυνατές αποστάσεις από τα γύρω εμπόδια. Ένα τέτοιο παράδειγμα είναι το σχήμα 3.3 [13]. Το GVD ενός χώρου μπορεί να χρησιμοποιηθεί σε διάφορες εφαρμογές, όπως η εύρεση των πορτών των δωματίων 5.1.2 ή η ταξινόμηση ενός υποχώρου σε δωμάτιο ή διάδρομο Α΄.

3.3 Αλγοριθμος Brushfire

Ο αλγόριθμος *Brushfire* ή *Wavefront* είναι μια μέθοδος υπολογισμού αποστάσεων μεταξύ σημείων σε έναν διακριτοποιημένο χώρο σε μορφή γράφου, όπως ένα OGM. Ο αλγόριθμος εκκινεί από ένα σύνολο σημείων του χώρου και βρίσκει τα αμέσως



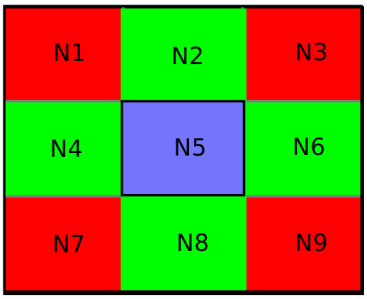
Σχήμα 3.2: Ευκλείδιο Διάγραμμα Voronoi
Πηγή: https://en.wikipedia.org/wiki/Voronoi_diagram



Σχήμα 3.3: GVD διδιάστατου χώρου

γειτονικά τους σημεία επαναληπτικά. Έτσι υπολογίζει την κοντινότερη απόσταση κάθε σημείου του ελεύθερου χώρου από το αρχικό υποσύνολο σημείων κρατώντας το πλήθος των επαναλήψεων που χρειάστηκαν για να φτάσει εως εκεί. Στην παρούσα διπλωματική η διαδικασία αυτή χρησιμοποιείται σε διάφορες παραλλαγές, ενώ σημειώνεται ότι σε όλες τις περιπτώσεις έχει υλοποιηθεί σε γλώσσα C για την βελτίωση της ταχύτητας της διαδικασίας των υπολογισμών.

Επιπλέον, ως γειτονικά σημεία κάθε σημείου του χώρου μπορούν να θεωρηθούν είτε μόνο τα τέσσερα σημεία (βόριο, νότιο, ανατολικό και δυτικό) που παρουσιάζονται στο σχήμα 3.4 με πράσινο χρώμα είτε και τα οκτώ σημεία που βρίσκονται γύρω του. Στην μελέτη αυτή χρησιμοποιήθηκαν και τα 8 σημεία ως γείτονες σε όλες τις περιπτώσεις, καθώς έτσι προσεγγίζεται η πραγματική απόσταση μεταξύ σημείων με μεγαλύτερη ακρίβεια.



Σχήμα 3.4: Πίνακας γειτόνων σημείων
Πηγή: http://roboscience.org/book/html/Planning/Brushfire.html

3.4 Αλγοριθμος Dijkstra

Ο αλγόριθμος *Dijkstra* [14] είναι ένας αλγόριθμος εύρεσης του ελάχιστου μονοπατιού μεταξύ κόμβων ενός γράφου. Δέχεται ως είσοδο έναν γράφο, τον κόμβο αφετηρία και τον κόμβο στόχο και εκτελεί μια επαναληπτική διαδικασία εύρεσης της βέλτιστης διαδρομής από την αφετηρία μέχρι τον στόχο. Κριτήριο βελτιστοποίησης είναι η ελαχιστοποίηση του συνολικού κόστους του μονοπατιού, που αποτελείται από το άθροισμα των βαρών των ακμών της κάθε διαδρομής.

Ένας γράφος, για παράδειγμα, μπορεί να αναπαριστά διάφορες πόλεις μιας χώρας, οι ακμές του τους δρόμους που ενώνουν πόλεις μεταξύ τους και τα βάρη των ακμών τις αποστάσεις τους. Ο αλγόριθμος αυτός μπορεί να βρει την κοντινότερη διαδρομή από μια πόλη προς όλες τις υπόλοιπες.

Η πολυπλοκότητα του αλγορίθμου αυτού είναι $O(|V|^2)$, όπου $|V|$ είναι ο αριθμός των κόμβων.

Αλγόριθμος 3.1 Dijkstra

```
1: function DIJKSTRA(Graph, source)
2:     for each vertex v in Graph do
3:         dist[v] = infinity
4:         previous[v] = undefined
5:     end for
6:     dist[source] = 0
7:     Q = the set of all nodes in Graph
8:     while Q is not empty do
9:         u = node in Q with smallest dist[]
10:        remove u from Q
11:        for each neighbor v of u do
12:            alt = dist[u] + dist_between(u, v)
13:            if alt < dist[v] then
14:                dist[v] = alt
15:                previous[v] = u
16:            end if
17:        end for
18:    end while
19:    return previous[]
```

3.5 Αλγοριθμος Hill Climbing

Ο αλγόριθμος *ανάβασης πλαγιάς* (Hill Climbing - HC)[2] είναι μια τεχνική βελτιστοποίησης που ανήκει στην κατηγορία της τοπικής αναζήτησης της αριθμητικής ανάλυσης. Είναι ένας επαναληπτικός αλγόριθμος που ξεκινάει με μία αυθαίρετη λύση στο πρόβλημα που μελετάται και προσπαθεί να βρει συνεχώς μια καλύτερη αλλάζοντας στοιχειωδώς την λύση. Εαν κάποια αλλαγή οδηγεί σε καλύτερο αποτέλεσμα, τότε η λύση αυτή κρατείται ως η τρέχουσα βέλτιστη και η διαδικασία συνεχίζεται με μια επόμενη αλλαγή. Η διαδικασία τερματίζεται όταν σταματήσει να υπάρχει κάποια αλλαγή που να βελτιώνει την τρέχουσα λύση.

Ο αλγόριθμος αυτός βρίσκει την καλύτερη δυνατή λύση του προβλήματος μέσα σε μια τοπική περιοχή, που εξαρτάται αποκλειστικά από την αρχική αυθαίρετη λύση. Αυτό σημαίνει ότι ο αλγόριθμος μπορεί να εγκλωβιστεί σε τοπικά μέγιστα, δίχως να μπορεί να προσεγγίσει τα ολικά μέγιστα. Ωστόσο, το γεγονός ότι είναι πολύ απλός και γρήγορος στην υλοποίηση του τον καθιστούν χρήσιμο σε προβλήματα που ο χρόνος υπολογισμού είναι σημαντικός ή σε προβλήματα που η ολική λύση δεν απέχει σημαντικά από τις τοπικές.

Ταυτόχρονα, έχουν δημιουργηθεί διάφορες παραλλαγές του, με στόχο την αποφυγή του σημαντικού αυτού μειονεκτήματος που έχει. Δύο τέτοιες παραλλαγές χρησιμοποιήθηκαν σε αυτή την διπλωματική εργασία. Στην πρώτη παραλλαγή χρησιμοποιούνται επανεκκινήσεις της διαδικασίας όταν αυτή κολλάει σε τοπικά μέγι-

[2] https://en.wikipedia.org/wiki/Hill_climbing

στα (Random Restart Hill Climbing - RRHC)[3]. Πιο αναλυτικά, εκτελείται ο απλός αλγόριθμος hill climbing μέχρι να φτάσει σε ένα τοπικό μέγιστο, όπου και η λύση αυτή κρατείται. Στη συνέχεια, εκτελείται ξανά ο hill climbing με διαφορετική αρχική λύση. Έτσι, εαν το νέο τοπικό μέγιστο αποτελεί βελτίωση του προηγούμενου, το αντικαθιστά. Η διαδικασία αυτή επαναλαμβάνεται για έναν συγκεκριμένο αριθμό επαναλήψεων που έχουν ορισθεί στην αρχή της διαδικασίας. Η δεύτερη παραλλαγή χρησιμοποιεί μια στοχαστική πιθανότητα για να αποδέχεται και αλλαγές που δεν βελτιώνουν την τρέχουσα λύση. Αυτή δίνει την δυνατότητα αποφυγής ορισμένων τοπικών ακρότατων, καθώς η λύση δεν περιορίζεται σε μια μικρή περιοχή τιμών. Μάλιστα, η πιθανότητα αυτή συνήθως με το πέρασμα των επαναλήψεων τείνει να μεταβάλλεται με τρόπο τέτοιο, ώστε ο αλγόριθμος να δέχεται όλο και περισσότερες αλλαγές που βελτιώνουν αυστηρά την λύση, προσεγγίζοντας σε άπειρο χρόνο την αρχική εκδοχή του hill climbing.

[3] https://en.wikipedia.org/wiki/Hill_climbingVariants

4
Εργαλεία

Στο κεφάλαιο αυτό παρουσιάζονται τα βασικά εργαλεία που χρησιμοποιήθηκαν σε όλη τη διάρκεια της παρούσας διπλωματικής εργασίας. Συγκεκριμένα, παρουσιάζεται το μεσολειτουργικό σύστημα ROS στο οποίο βασίστηκαν οι υλοποιήσεις των συστημάτων και τα πειράματα, το Gazebo στο οποίο πραγματοποιήθηκαν οι προσομοιώσεις των πειραμάτων, το Rviz με το οποίο οπτικοποιούνται τα δεδομένα κατά την εκτέλεση των αλγορίθων. Στη συνέχεια, παρουσιάζεται το Navigation Stack, που αποτελεί τη δομή η οποία συντονίζει την πλοήγηση του οχήματος στον χώρο. Τέλος, γίνεται μια σύντομη αναφορά στις γλώσσες προγραμματισμού που επιλέχθηκαν κατά την υλοποίηση των αλγορίθμων.

4.1 Robot Operating System (ROS)

Το Robot Operating System [15] αποτελεί το πιο διαδεδομένο σύστημα υλοποίησης ρομποτικών συστημάτων. Αποτελεί ένα μεσολειτουργικό (middleware) σύστημα, το οποίο διασυνδέει το λογισμικό (software) με το υλικό (hardware) με έναν τέτοιο τρόπο που η δημιουργία ρομποτικών συστημάτων και εφαρμογών είναι πιο απλή και γρήγορη. Περιλαμβάνει πολλά χαρακτηριστικά των λειτουργικών συστημάτων, όπως ο χαμηλού επιπέδου έλεγχος των συσκευών, η δημιουργία πλήθους διεργασιών και η παραλληλοποίηση των συστημάτων, η μετάδοση μηνυμάτων μεταξύ των διεργασιών αυτών, η διαχείρηση πακέτων. Επίσης, παρέχει χρήσιμα εργαλεία και βιβλιοθήκες για την ολοκληρωτική διαχείρηση κώδικα από πολλαπλούς υπολογιστές.

Το ROS είναι μια πλατφόρμα ανοικτού κώδικα που έχει ως κύριο στόχο την απλοποίηση της διαδικασίας δημιουργίας ρομποτικών εφαρμογών. Η δόμηση της είναι τέτοια που εξυπηρετεί αυτό στον σκοπό. Συγκεκριμένα, ο κώδικας μπορεί και είναι επαναχρησιμοποιήσιμος, καθώς η κάθε εφαρμογή αποτελείται από ένα πλήθος κατανεμημένων διεργασιών (Nodes) που δίνει την δυνατότητα ανεξάρτητης

ανάπτυξης των διαφόρων διεργασιών. Οι διεργασίες ομαδοποιούνται σε πακέτα (Packages) και στοίβες (Stacks), ώστε να είναι εύκολη η διανομή και ο έλεγχος τους μεταξύ των ερευνητών. Τέλος, μόνο κατά την συνολική εκτέλεση τους υπάρχει η ανάγκη συγκέντρωσης των διάφορων αυτών τμημάτων.

4.2 Εργαλεια Προσομοιωσης

Το Gazebo[4] αποτελεί ένα περιβάλλον προσομοίωσης ρομποτικών εφαρμογών σε τρισδιάστατα εικονικά περιβάλλοντα. Περιλαμβάνει όλους τους φυσικούς νόμους ενός πραγματικού περιβάλλοντος, κάτι που έχει ως αποτέλεσμα την προσομοίωση πραγματικών πειραμάτων σε ρεαλιστικές συνθήκες με μικρά σφάλματα σε σχέση με τα αντίστοιχα πραγματικά πειράματα. Επιπλέον, περιλαμβάνει ένα μεγάλο σύνολο από διαθέσιμα ρομπότ, αισθητήρες και περιβάλλοντα, κάνοντας πιο εύκολη την υλοποίηση των πειραμάτων.

Το ρομποτικό όχημα που χρησιμοποιείται στην εργασία αυτή είναι το turtlebot2[5] το οποίο είναι προμηθευμένο με ένα LiDAR τοποθετημένο στο πάνω μέρος του για την ασφαλή κίνηση και την ακριβή εκτίμηση της θέσης του στον χώρο.

Επίσης, σημειώνεται το ROS έχει την υποδομή της άμεσης σύνδεσης των υπολοίπων συστημάτων με το Gazebo, καλώντας το κατά την εκκίνηση του συνόλου των διεργασιών. Μάλιστα, το Gazebo λειτουργεί ως μια ανεξάρτητη διεργασία, η οποία επικοινωνεί συνεχώς με όλες τις υπόλοιπες.

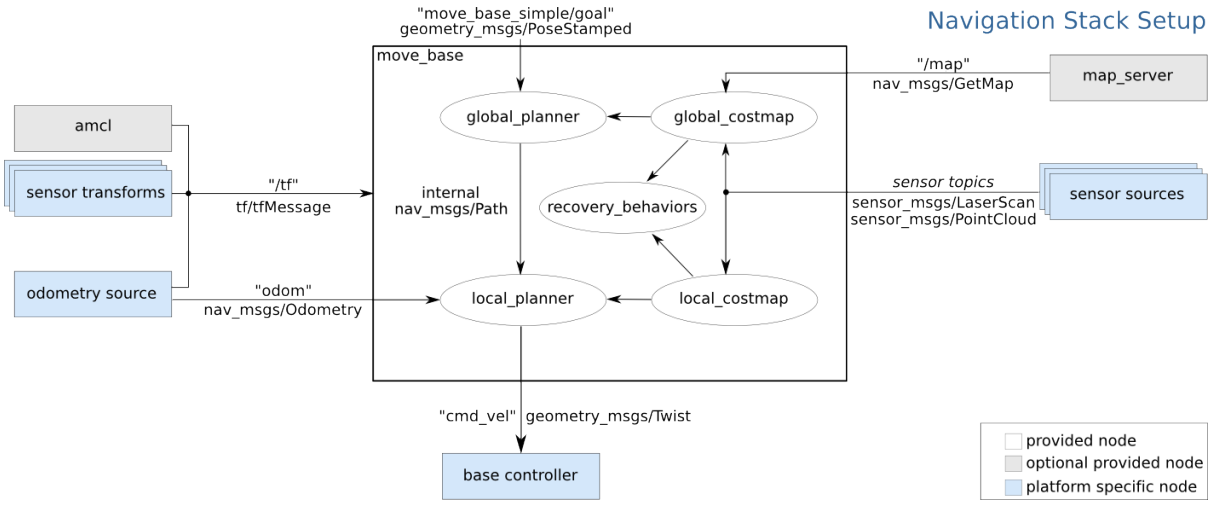
Το ROS visualiziation ή Rviz[6] χρησιμοποιείται για την τρισδιάστατη οπτικοποίηση όλων των δεδεμένων του συστήματος το οποίο τρέχει στον υπολογιστή. Αυτά τα δεδομένα μπορεί να αφορούν μετρήσεις από διάφορους αισθητήρες, πληροφορίες κατάστασης του ρομπότ και του περιβάλλοντα χώρου, ένα εικονικό μοντέλο του ρομπότ κ.α. Επίσης, μπορεί να παρουσιάσει πληροφορίες που προκύπτουν από τις διεργασίες που υλοποιούνται, όπως για παράδειγμα την επισήμανση του σημείου στόχου κατά την πλοήγηση του ρομπότ σε έναν χώρο.

4.3 Navigation Stack

Η πλοήγηση του ρομποτικού οχήματος υλοποιήθηκε με τη χρήση ενός stack πακέτων που προσφέρει το ROS, του navigation[7]. Αυτό περιέχει ένα σύνολο πακέτων τα οποία χρησιμοποιούν τα δεδομένα της οδομετρίας, των αισθητήρων και του χάρτη του χώρου. Δηλώνοντας ένα σημείο ως στόχο της πλοήγησης δημιουργεί το κατάλληλο σύνολο εντολών ταχύτητας τις οποίες πρέπει να ακολουθήσει το όχημα για να κατευθυνθεί στον στόχο αυτό, χωρίς να συγκρούεται με εμπόδια του περιβάλλοντα χώρου. Η συνολική δομή του stack παρουσιάζεται στο σχήμα 4.1.

Υπάρχουν, λοιπόν, δύο διαφορετικοί σχεδιαστές μονοπατιών, ο τοπικός (local planner) και ο καθολικός (global planner). Και οι δύο διαβάζουν την τρέχουσα

[4] http://gazebosim.org/
[5] https://www.turtlebot.com/turtlebot2/
[6] http://wiki.ros.org/rviz
[7] http://wiki.ros.org/navigation



Σχήμα 4.1: Δομή Navigation Stack
Πηγή: http://wiki.ros.org/navigation/Tutorials/RobotSetup

θέση του οχήματος από τη διεργασία amcl και τον χάρτη του χώρου από την διεργασία map_server και προσπαθούν να δημιουργήσουν μια ασφαλή πορεία μέχρι το σημείο στόχο. Ο global planner δημιουργεί ένα συνολικό μονοπάτι (global path) από την τρέχουσα θέση μέχρι τον στόχο, χωρίς να δίνει ιδιαίτερη έμφαση σε λεπτομερείς κινήσεις και μικρά εμπόδια στο περιβάλλον. Αντίθετα, ο local planner δημιουργεί ένα μονοπάτι με μικρή συνολική απόσταση (local path) που προσπαθεί να προσεγγίσει το global path ελέγχοντας με μεγαλύτερη ακρίβεια για κοντινά εμπόδια και δυναμικές αλλαγές του περιβάλλοντος προσαρμόζοντας τις κινήσεις του οχήματος. Στην εργασία αυτή χρησιμοποιήθηκαν οι εξής:

- teb_local_planner ,[8]
- global_planner[9]

Για την ασφαλή μετάβαση του οχήματος μέσα στον χώρο δημιουργούνται δύο χάρτες με τιμές κόστους (costmaps) που περιέχουν πληροφορίες για τα εμπόδια του χώρου. Ο ένας χάρτης χρησιμοποιείται από τον global planner και χρησιμοποιεί την δομή ολόκληρου του περιβάλλοντος για την δημιουργία του συνολικού μονοπατιού πλοήγησης, ενώ ο δεύτερος χρησιμοποιείται από τον local planner για τοπική αποφυγή εμποδίων.

4.4 Γλωσσες Προγραμματισμου

Οι ρομποτικές εφαρμογές που χρησιμοποιούν την υποδομή του ROS συνήθως αναπτύσσονται είτε σε Python[10] είτε σε C++[11]. Σε αυτή την διπλωματική χρησιμοποιήθηκε η Python, καθώς δίνει τη δυνατότητα εύκολης ανάπτυξης κώδικα και

[8] http://wiki.ros.org/teb$_{l}ocal_{p}lanner$
[9] http://wiki.ros.org/global$_{p}lanner$
[10] https://www.python.org/
[11] http://www.cplusplus.com/

γρήγορου ελέγχου. Έχει, όμως, ένα σημαντικό μειονέκτημα. Ο χρόνος εκτέλεσης κώδικα γραμμένο σε Python είναι μεγάλος. Για τον λόγο αυτό, σε ορισμένες περιπτώσεις που η ταχύτητα εκτέλεσης είναι μείζονος σημασίας, έχουν αναπτυχθεί αλγόριθμοι σε C και έχουν εισαχθεί στον κώδικα της Python με την χρήση της βιβλιοθήκης Cffi[12]. Τέλος, μια από τις πιο σημαντικές βιβλιοθήκες που χρησιμοποιείται κατά κόρον είναι η *numpy*[13] η οποία διαχειρίζεται τις μεταβλητές των προγραμμάτων και υλοποιεί γρήγορα πράξεις μεταξύ τους.

[12] https://cffi.readthedocs.io/en/latest/
[13] https://www.numpy.org/

5 Υλοποιήσεις

Η παρούσα διπλωματική εργασία ασχολείται με την ανάπτυξη ενός συστήματος για την αυτόνομη κάλυψη ενός γνωστού δισδιάστατου χώρου. Συγκεκριμένα, στοχεύει στην βέλτιστη κάλυψη των περιοχών αυτών που θα μπορούσαν να βρίσκονται προϊοντα, δηλαδή ετικέτες RFID. Τέτοια σημεία αποτελούν το σύνολο όλων των εμποδίων του χώρου, αφού μόνο εκεί μπορούν να υπάρχουν οι ετικέτες.

Ο χώρος στον οποίο γίνεται η μελέτη θεωρείται γνωστός και αναπαρίσταται σε μορφή OGM. Επίσης, οι αισθητήρες που χρησιμοποιούνται αποτελούν παράμετρο εισόδου του προγράμματος προκειμένου η διαδικασία να εκτελεστεί συναρτήσει αυτών. Τέλος, το ρομποτικό όχημα που χρησιμοποιείται είναι το Turtlebot2, όπως αναφέρθηκε και στο υποκεφάλαιο 4.2.

Οι υλοποιήσεις χωρίζονται σε 4 μέρη:

- Διαχωρισμός του χώρου σε δωμάτια
- Υπολογισμός βέλτιστης αλληλουχίας επίσκεψης των δωματίων
- Διαδικασία πλοήγησης και κάλυψης του χώρου
- Υπολογισμός βέλτιστου μονοπατιού κάλυψης σε κάθε δωμάτιο

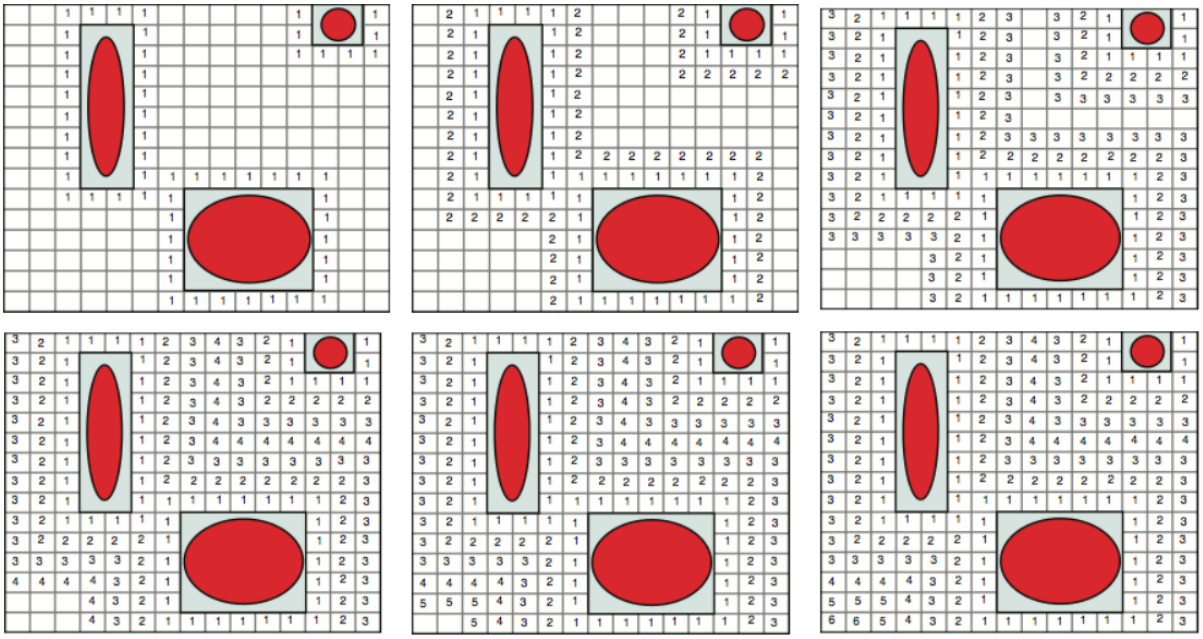
5.1 Τοπολογικη Αναλυση Χωρου

Στο πρώτο μέρος της εργασίας αυτής αναλύεται ο διδιάστατος χάρτης του χώρου με στόχο τον εντοπισμό των διάφορων δωματίων που αποτελούν τον χώρο αυτό. Για την επίτευξη αυτού χρησιμοποιούνται ορισμένοι αλγόριθμοι που παρουσιάζονται στην συνέχεια.

Αλγόριθμος Brushfire

Αρχικά, υλοποιήθηκε η πιο απλή αλλά και σημαντική εκδοχή του αλγορίθμου brushfire 3.3, η οποία είναι η εύρεση του πεδίου brushfire των εμποδίων του OGM.

Χρησιμοποιείται ως αρχικό υποσύνολο σημείων το σύνολο των εμποδίων του OGM και υπολογίζεται για κάθε σημείο του ελεύθερου χώρου η απόσταση του από το κοντινότερο του εμπόδιο. Η διαδικασία, λοιπόν, εκκινεί από τα εμπόδια και εξαπλώνεται συνεχώς σε όλα τα γειτονικά σημεία που δεν έχει διαδοθεί. Η επαναληπτική αυτή μέθοδος μπορεί να γίνει πιο κατανοητή από το σχήμα 5.1. Όπως φαίνεται, το κύμα ξεκινάει περιμετρικά των εμποδίων και εξαπλώνεται κατά ένα σημείο την φορά. Η διαδικασία συνεχίζεται μέχρι να μην μπορεί να πραγματοποιηθεί άλλη εξάπλωση στον χώρο, που σημαίνει ότι και κάλυψε ολόκληρο τον χώρο.



Σχήμα 5.1: Παράδειγμα Brushfire
Πηγή: http://roboscience.org/book/html/Planning/Brushfire.html

Ο brushfire εμποδίων παρουσιάζεται στο 5.1. Στις γραμμές 2-4 το πεδίο brushfire ορίζεται έτσι ώστε να έχει τις ίδιες διαστάσεις με το ogm, αρχικά με μηδενικές τιμές στον ελεύθερο χώρο, 1 στα εμπόδια και -1 στα άγνωστα σημεία. Για όσο συμβαίνει η εξάπλωση εξετάζονται όλα τα σημεία του brushfire. Εαν σε κάποιο ελεύθερο σημείο δεν έχει φτάσει ακόμα το κύμα αλλά έχει φτάσει σε έναν γείτονα του στην προηγούμενη επανάληψη με τιμή $step$, τότε και στο τρέχον σημείο ανανεώνει το brushfire με τιμή $step + 1$. Η διαδικασία επαναλαμβάνεται εως ότου δεν επέλθει καμία ανανέωση στον πίνακα $brushfire$, δηλαδή μέχρι την πλήρη εξάπλωση των τιμών.

Υπολογισμός GVD

Εξίσου χρήσιμο με το $brushfire$ του χάρτη είναι και το GVD του. Αυτό υπολογίζεται χρησιμοποιώντας τη μέθοδο skeletonize[14] της βιβλιοθήκης της skimage της Python. Το GVD έχει τις ίδιες διαστάσεις με το αντίστοιχο OGM και οι τελικές του τιμές θα είναι 1 στα σημεία που ανήκουν σ' αυτό και 0 σε όλα τα υπόλοιπα.

[14] https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html

Αλγόριθμος 5.1 Obstacle Brushfire

```
1: function obstacleBrushfire(ogm)
2:     brushfire = np.zeros(ogm.shape)
3:     brushfire[ogm > 49] = 1
4:     brushfire[ogm == −1] = −1
5:     step = 1
6:     changed = 1
7:     while changed == 1 do
8:         changed = 0
9:         for i = 1 to (brushfire.width - 1) do
10:            for j = 1 to (brushfire.height - 1) do
11:                if brushfire[i][j] == 0 and a neighbor's brushfire value is equal to step then
12:                    brushfire[i][j] = step + 1
13:                    changed = 1
14:                end if
15:            end for
16:        end for
17:        step+ = 1
18:    end while
19:    return brushfire
```

Στην μέθοδο skeletonize εισάγεται ως δυαδική εικόνα ο ελεύθερος χώρος του OGM. Όπως φαίνεται στην εντολή 3 του αλγορίθμου 5.2 χρησιμοποιούνται ως είσοδο τα σημεία που έχουν τιμή brushfire μεγαλύτερη από 5, δηλαδή όλα τα ελεύθερα σημεία εκτός από αυτά που είναι πάρα πολύ κοντά στα εμπόδια. Ο OGM πολλές φορές περιέχει ατέλειες πολύ κοντά στα εμπόδια λόγω αστοχίας της διαδικασίας του SLAM. Χαρακτηριστικό παράδειγμα είναι η περίπτωση να θεωρεί ως ελεύθερο χώρο σημεία πίσω από τον τοίχο, εκτός του χώρου, που κανονικά θα έπρεπε να θεωρούνται άγνωστα. Προκειμένου να μην επηρεάσουν τη διαδικασία υπολογισμού του GVD, λαμβάνονται υπόψη τα σημεία που έχουν μια στοιχειώδη απόσταση από τα εμπόδια.

Αλγόριθμος 5.2 GVD

```
1: function gvd(brushfire)
2:     voronoi = np.zeros(brushfire.shape)
3:     voronoi[brushfire >= 5] = 1
4:     voronoi = skimage.morphology.skeletonize(voronoi)
5:     return voronoi
```

5.1.1 Υπολογισμός τοπολογικού γράφου του χώρου

Το GVD ενός χάρτη περιέχει σημαντικές πληροφορίες για το περιβάλλον. Κρύβει μέσα του τόσο τα όρια των δωματίων όσο και την δομή τους. Επομένως, χρειάζεται

να εξαχθούν σημεία που περιέχουν αυτές τις πληροφορίες. Πιο σημαντικοί είναι οι κόμβοι που αντιστοιχούν στις πόρτες του χώρου, καθώς με αυτούς μπορεί να διαχωριστεί αποτελεσματικά ο χάρτης στα δωμάτια του. Η μέθοδος βασίστηκε σε μια αντίστοιχη μελέτη γνωστού χώρου που είχε πραγματοποιηθεί στο [16].

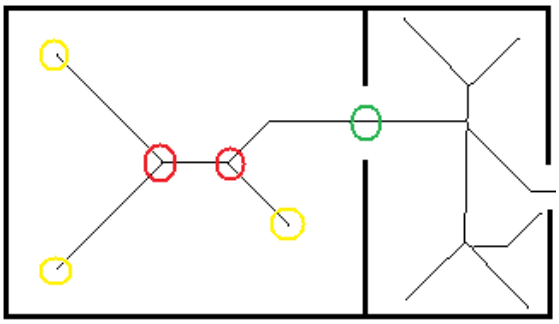
Η διαδικασία που υλοποιήθηκε αναλύεται στη συνέχεια. Καταρχάς, κάθε σημείο του GVD μπορεί να έχει από ένα έως τέσσερα γειτονικά σημεία τα οποία ανήκουν και αυτά στο GVD. Ανάλογα μάλιστα με το πλήθος των γειτόνων, μπορούν να εξαχθούν πληροφορίες για το κάθε σημείο. Έτσι, παρατηρώντας και το σχήμα 5.2 τα σημεία με έναν μόνο γείτονα βρίσκονται στις άκρες του GVD, δηλαδή σε περιοχές που πλησιάζουν σε εμπόδια, όπως οι γωνίες των δωματίων και επισημαίνονται με κίτρινους κύκλους. Αντίστοιχα, τα σημεία με τρεις ή τέσσερις γείτονες βρίσκονται σε πιο κεντρικά, όσον αφορά τις αποστάσεις από τα εμπόδια, σημεία ή σε σημεία διακλάδωσης, όπως η περιοχή μπροστά από μια πόρτα, όπου και το διάγραμα διακλαδίζεται και επισημαίνονται με κόκκινους κύκλους. Το σημείο που αντιστοιχεί σε πόρτα επισημαίνεται με πράσινο χρώμα και όπως φαίνεται έχει δύο γείτονες. Όλοι αυτοί οι κόμβοι πρέπει να εξαχθούν από το GVD για να δημιουργηθεί ο τοπολογικός γράφος του χώρου.

Πρώτα, σαρώνεται ο χώρος, ώστε να βρεθούν όλα τα σημεία του GVD με έναν, τρεις ή περισσότερους γείτονες. Αυτά αν και δεν χρειάζονται για την εύρεση των πορτών και τον διαχωρισμό των δωματίων, μπορούν να χρησιμοποιηθούν για άλλους λόγους, όπως παρουσιάζεται στο Α΄.

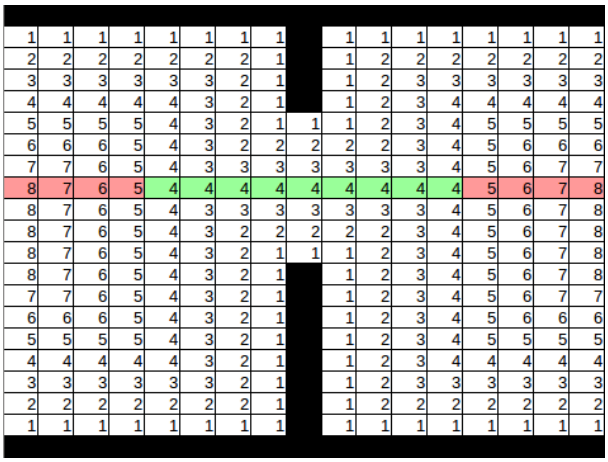
Έπειτα, πρέπει να μελετηθούν τα σημεία με δύο γείτονες για να εξαχθούν οι κόμβοι οι οποίοι είναι πιθανό να αποτελούν πόρτες στον χώρο. Για την επίτευξη αυτού θα χρησιμοποιηθεί η τιμή brushfire που έχουν αυτά τα σημεία. Συγκεκριμένα, τα σημεία πόρτες παρουσιάζουν τοπικό ακρότατο στην τιμή brushfire τους. Αυτό γίνεται εύκολα αντιληπτό εαν σκεφτεί κανείς πως για να περάσει το ρομποτικό όχημα, αλλά και ένας άνθρωπος, από μια πόρτα θα πλησιάσει τα εμπόδια, θα φτάσει σε μια ελάχιστη απόσταση ακριβώς στην ευθεία της πόρτας και μετά θα αρχίσει να απομακρύνεται από αυτά. Η ιδιότητα αυτή είναι καθοριστική, καθώς οι υποψήφιες πόρτες μπορούν να βρεθούν, εαν υπολογιστούν τα τοπικά ελάχιστα του brushfire στα σημεία που ανήκουν στο GVD και έχουν δύο γείτονες. Αυτό φαίνεται και στο σχήμα 5.3 όπου έχουν επισημανθεί τα σημεία που ανήκουν στο GVD, ενώ για κάθε σημείο έχει γραφεί το brushfire του. Πράγματι τα σημεία με πράσινο αποτελούν το τοπικό ελάχιστο του brushfire πάνω στο GVD.

Η διαδικασία που υλοποιήθηκε ελέγχει για κάθε σημείο του GVD με δύο γείτονες εαν έχει την ελάχιστη brushfire τιμή σε μια περιοχή 20x20 γύρω απ' αυτό, συγκρίνοντας μόνο σημειά που ανήκουν στο διάγραμμα Voronoi. Έτσι, βρίσκονται τα τοπικά ελάχιστα τα οποία, όμως, χρειάζονται περαιτέρω επεξεργασία. Όπως φαίνειται και στο 5.3 για κάθε πόρτα περισσότερα από ένα σημεία αποτελούν τοπικό ελάχιστο, καθώς ο αλγόριθμος brushfire εξαπλώνεται και στα οχτώ γειτονικά του σημεία. Για την αναγωγή κάθε τοπικού ελαχίστου στο πιο αντιπροσωπευτικό της πόρτας, ομαδοποιούνται τα σημεία ανά γειτονιές και κρατιέται το μεσαίο από κάθε ομάδα ως υποψήφιος κόμβος πόρτας.

Η διαδικασία παρουσιάζεται στον αλγόριθμο 5.3 και χρησιμοποιεί τα ήδη υπολογισμένα πεδία brushfire και GVD του χώρου. Στις εντολές 2-14 γίνεται η πρώτη εξαγωγή χρήσιμων κόμβων από το GVD. Στη συνέχεια στις εντολές 16-26 ομαδο-



Σχήμα 5.2: Κόμβοι τοπολογικού γράφου



Σχήμα 5.3: Τοπικό Ελάχιστο Brushfire στις πόρτες

ποιούνται οι κόμβοι με δύο γείτονες που βρίσκονται στην ίδια γειτονιά, δηλαδή που αντιστοιχούν στην ίδια πόρτα, και κρατιέται ο μεσαίος κόμβος ως υποψήφιος κόμβος πόρτας.

Τα ενδιάμεσα αποτελέσματα παρουσιάζονται και στα επόμενα σχήματα. Συγκεκριμένα, οι μικρές ευθείες των τοπικών ελαχίστων παρουσιάζονται στο 5.4, όπου με μπλε σημειώνονται τα σημεία που πράγματι είναι κοντά σε μια πόρτα και με πράσινο σημεία που ενώ είναι ακρότατα του χάρτη, δεν αντιστοιχούν σε κάποια πόρτα. Μάλιστα, αυτά τα σημεία αποτελούν τοπικά μέγιστα του χώρου. Στην πράξη, λοιπόν, αν και ο αλγόριθμος αναζητάει τοπικά ελάχιστα μόνο, είναι πολύ πιθανό να βρει και τοπικά μέγιστα, καθώς για κάθε σημείο του GVD η brushfire τιμή του συγκρίνεται με τις γύρω 20x20 τιμές. Αυτό σημαίνει ότι σε περιπτώσεις που παρουσιάζονται και τοπικά μέγιστα, εαν αυτά αποτελούνται από περισσότερα από 20 σειριακά σημεία, θα κρατηθούν ορισμένα από αυτά. Για να καταλάβει ο αλγόριθμος ότι πρόκειται πράγματι για μέγιστο και όχι ελάχιστο θα χρειαζόταν να ελέγξει μια μεγαλύτερη περιοχή από τα 20x20 σημεία που καλύπτει ήδη, κάτι που όμως δεν θα έλυνε αποτελεσματικά το πρόβλημα για κάθε χώρο. Πάντα θα υπάρχει ένα περιβάλλον που θα χρειάζεται μεγαλύτερο πεδίο σύγκρισης των τιμών brushfire με το τρέχον σημείο. Το πρόβλημα διαχωρισμού των πραγματικών πορτών, λοιπόν, θα αντιμετωπιστεί στην επόμενη υποενότητα με μια διαφορετική διαδικασία.

Αλγόριθμος 5.3 Topological Nodes

```
1: function topologicalNodes(brushfire, gvd)
2:     nodes = []
3:     for every point (x, y) of gvd, where gvd == 1 do
4:         count gvd neighbors of point (x, y)
5:         if number of neighbors is 1, 3 or 4 then
6:             nodes.append((x, y))
7:         end if
8:         if number of neighbors is 2 then
9:             Check if brushfire[x, y] is minimum in a 20x20 area
10:            if brushfire[x, y] is min then
11:                nodes.append((x, y))
12:            end if
13:        end if
14:    end for
15:     # Recheck nodes with 2 neighbors to keep one point of each neighborhood
16:    final_nodes = []
17:    add all nodes that don't have 2 neighbors to final_nodes
18:    Group nodes with 2 neighbors to lists of points of same line
19:    for each list of points do
20:        if number of nodes in list is odd then
21:            add middle point to final_nodes
22:        else
23:            add first of the two middle points to final_nodes
24:        end if
25:    end for
26:    return final_nodes
```

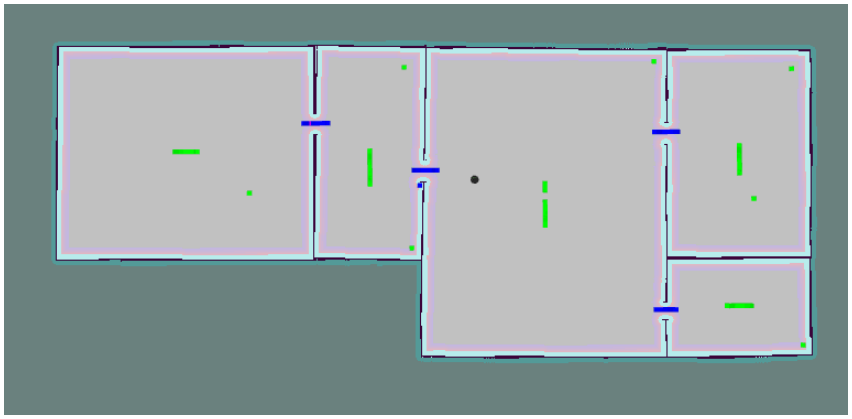
Στο σχήμα 5.5 παρουσιάζονται τα αντιπροσωπευτικά σημεία κάθε μικρού ευθύγραμμου τμήματος του προηγούμενου σχήματος με τα αντίστοιχα χρώματα. Οι κόμβοι αυτοί αποτελούν το σύνολο των υποψήφιων κόμβων πόρτας του χώρου.

Τέλος, στο σχήμα 5.6 παρουσιάζονται όλοι οι κόμβοι που επιστρέφει ο αλγόριθμος 5.3. Με μπλε εμφανίζονται οι τελικές πόρτες και με κόκκινο όλα τα υπόλοιπα σημεία στα εσωτερικά των δωματίων.

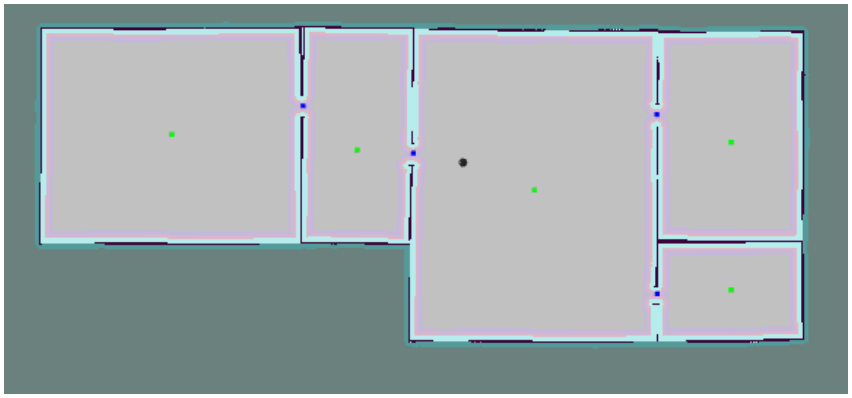
5.1.2 Ορισμός των Κόμβων Πόρτας

Το επόμενο βήμα της διαδικασίας είναι η εύρεση των πραγματικών πορτών του χώρου. Ως πιθανοί κόμβοι πόρτας λαμβάνονται όλοι οι κόμβοι της προηγούμενης διαδικασίας που έχουν δύο ακριβώς γείτονες στο GVD. Όπως αναλύθηκε νωρίτερα, υπάρχουν περισσότεροι υποψήφιοι κόμβοι από τις πραγματικές πόρτες ακόμη και σε περιβάλλοντα με απλή εσωτερική δομή. Φυσικά, όσο περισσότερα εμπόδια περιέχει ο χώρος, τόσα περισσότερα πιθανά σημεία πόρτας υπάρχουν.

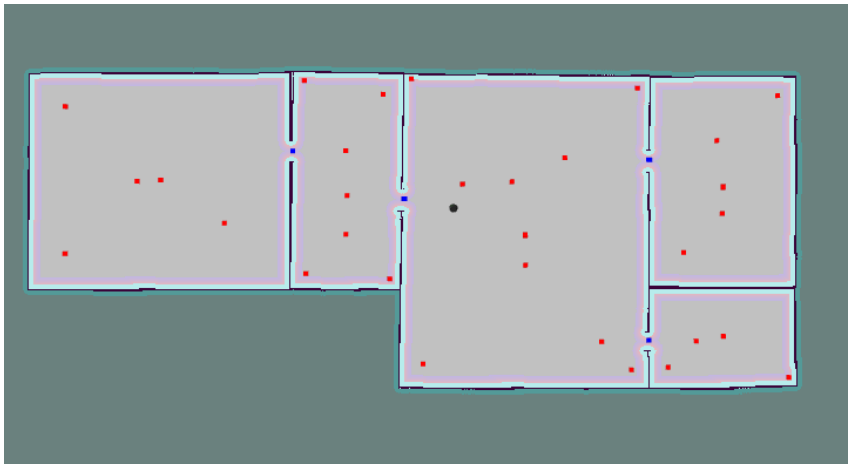
Η εύρεση των πραγματικών πορτών από το σύνολο των υποψήφιων βασίζεται σε μια σημαντική ιδιότητα κάθε πόρτας. Οι κόμβοι των πραγματικών πορτών βρί-



Σχήμα 5.4: Ευθείες με Κόμβους Τοπικών Ακροτάτων



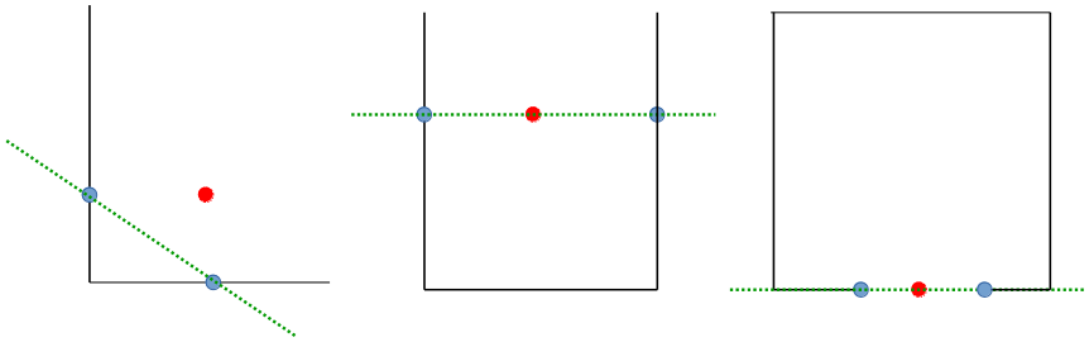
Σχήμα 5.5: Υποψήφιοι Κόμβοι Πόρτες



Σχήμα 5.6: Τελικοί Κόμβοι Τοπολογικού Γράφου του Χώρου

σκονται ενδιάμεσα δύο εμποδίων, των δύο τοιχών της πόρτας. Τα δύο αυτά σημεία σχηματίζουν, μάλιστα, μια ευθεία στον χώρο στην προέκταση της οποίας υπάρχει σημαντικό ποσοστό κατειλλημένου χώρου από την προέκταση των εμποδίων. Στόχος είναι να βρεθεί ένα κατάλληλο κατώφλι προσαρμοσμένο στον κάθε χάρτη, ώστε να συγκριθούν τα ποσοστά όλων των κόμβων με αυτό και να επιλεγούν μόνο αυτοί που έχουν μεγαλύτερο ποσοστό από το όριο αυτό.

Στο σχήμα 5.7 παρουσιάζονται τρεις διαφορετικές περιπτώσεις κόμβων που είναι υποψήφιοι να είναι κόμβοι πόρτες. Οι κόμβοι αυτοί έχουν κόκκινο χρώμα, ενώ τα σημεία που αποτελούν τα πιο κοντινά εμπόδια έχουν μπλε χρώμα. Η ευθεία των εμποδίων εμφανίζεται με διακεκομμένη πράσινη γραμμή. Στο πρώτο σχήμα, ο υποψήφιος κόμβος πόρτας βρίσκεται κοντά σε μια γωνία του δωματίου. Δεν αποτελεί σημείο πόρτας διότι δεν βρίσκεται ανάμεσα στα δύο σημεία των εμποδίων, δηλαδή δεν βρίσκεται πάνω στην πράσινη ευθεία. Στο δεύτερο σχήμα, ο υποψήφιος κόμβος αποτελεί σημείο της ευθείας αυτής. Όμως, τα σημεία που ανήκουν στην ευθεία κατά κύριο λόγο αποτελούν ελεύθερα σημεία του χώρου, πράγμα που σημαίνει ότι δεν υπάρχουν οι αναμενόμενοι τοίχοι στην προέκταση των εμποδίων. Τέλος, στο τρίτο σχήμα παρουσιάζεται η περίπτωση της πραγματικής πόρτας. Είναι εύκολα διακριτό το γεγονός ότι μεγάλο ποσοστό των σημείων που ανήκουν στην εν λόγω ευθεία είναι σημεία εμποδίων.



Σχήμα 5.7: Παραδείγματα Ελέγχου Κόμβων Πόρτας

Ο αλγόριθμος παρουσιάζεται σε αναλυτική μορφή στο 5.5. Η διαδικασία συνοψίζεται για κάθε υποψήφιο κόμβο στα παρακάτω βήματα:

- Εύρεση κοντινότερων εμποδίων του κόμβου
- Έλεγχος πλήθους αυτών των σημείων
- Εύρεση της ευθείας των σημείων αυτών και έλεγχος της εγγύτητας του κόμβου με αυτήν
- Μέτρηση ποσοστού κάλυψης στο OGM σε ένα πεπερασμένο μήκος της ευθείας αυτής
- Υπολογισμός κατωφλιού απόφασης
- Σύγκριση ποσοστού με κατώφλι αποδοχής κόμβου ως πόρτα

Το πρώτο βήμα κάνει χρήση της συνάρτησης 5.4 η οποία δέχεται ως είσοδο ένα OGM και τον τρέχοντα κόμβο και εκτελεί brushfire με αρχή το σημείο αυτό, μέχρι να

φτάσει η εξάπλωση στο πρώτο εμπόδιο. Τότε, η διαδικασία εκτελείται για άλλη μια επανάληψη και στη συνέχεια τερματίζεται. Η ύπαρξη της μιας παραπάνω φοράς είναι αναγκαία, καθώς σε περίπτωση που σε μια πόρτα μεσολαβεί άρτιος αριθμός ελεύθερων σημείων, ο κόμβος της πόρτας θα βρίσκεται στην μια πλευρά πιο κοντά κατά ένα pixel. Με τον τρόπο αυτό, όμως, θα εντοπιστούν και οι δύο πλευρές των εμποδίων. Στη συνέχεια, τα εμπόδια όπου έφτασε ο brushfire ομαδοποιούνται σε γειτονικά και επιλέγεται να επιστραφεί από κάθε ομάδα το σημείο που είναι πιο κοντά στον κόμβο εκκίνησης. Η εντολή αυτή καλείται στην γραμμή 4.

Έπειτα, πραγματοποιείται ο έλεγχος του πλήθους των σημείων των εμποδίων στην γραμμή 5. Το επόμενο βήμα υλοποιείται με τις εντολές 7-11. Μετά μετράται το ποσοστό κάλυψης των σημείων της ευθείας. Πολλές φορές όμως ένα OGM μπορεί να περιέχει ατέλειες. Γι αυτό τον λόγο ο αλγόριθμος χρησιμοποιεί σημεία που είναι γύρω από την ευθεία. Συγκεκριμένα, χρησιμοποιείται το ευθύγραμμο τμήμα των δύο σημείων των εμποδίων της εντολής 4 με 50 pixels επέκταση προς κάθε πλευρά. Επίσης, το πλάτος της ευθείας από 1 pixel αυξάνεται στα 5 pixel. Με τον τρόπο αυτό εξασφαλίζεται ότι οι γύρω τοίχοι θα βρίσκονται μέσα στην περιοχή που προσμετράται στο ποσοστό κάλυψης. Άλλωστε το πάχος των τοιχών στα συνήθη OGM είναι μεγαλύτερο του ενός pixel, συνεπώς οι επεκτάσεις θα οδηγήσουν σε ακόμα μεγαλύτερο ποσοστό κατάλληψης.

Το πιο σημαντικό βήμα της διαδικασίας αυτής είναι η επιλογή ενός σωστού κατωφλιού, ανάλογου του κάθε χάρτη. Θα ήταν λάθος η χρήση ενός σταθερού αριθμού, καθώς χάνεται η δυνατότητα γενίκευσης της διαδικασίας σε κάθε νέο χάρτη. Μελετήθηκαν οι τιμές των ποσοστών πορτών από διάφορους πολύπλοκους χώρους και προέκυψαν τα εξής δύο συμπεράσματα:

- Όλοι οι κόμβοι που δεν αποτελούν πόρτες έχουν ποσοστό μικρότερο από 20%.
- Οι κόμβοι που είναι πόρτες έχουν σημαντικά πολύ μεγαλύτερη τιμή της μετρικής αυτής.

Η διαφορά των τιμών είναι τόσο μεγάλη σε κάθε περίπτωση που το τελικό όριο μπορεί να βρεθεί εντοπίζοντας την μεγαλύτερη πρώτη διαφορά των ταξινομημένων τιμών. Για παράδειγμα, στην επόμενη λίστα παρουσιάζονται οι τιμές από το σχήμα 5.5:

[0.01126761, 0.01344538, 0.01587302, 0.12833333, 0.19310345, 0.24833333, 0.49579832]

Οι πρώτες τρεις τιμές που είναι κοντά στο 1% αφορούν σημεία που δεν αντιστοιχούν σε πόρτες. Αντίθετα, οι υπόλοιπες τέσσερις είναι από κόμβους πόρτας. Φαίνεται ότι ενώ οι μεν τιμές κυμαίνονται κοντά στο 1%, οι δε είναι αρκετά πάνω από το 10% με την μεγαλύτερη σχεδόν στο 50%. Επομένως, είναι αρκετά διακριτό το όριο σε κάθε χάρτη και εντοπίζεται στην μεγαλύτερη τιμή των πρώτων διαφορών των τιμών, για την οποία το όριο θα είναι μικρότερο του 20%. Στόχος είναι να τεθεί ως κατώφλι το μεγαλύτερο ποσοστό κάλυψης από τους κόμβους που δεν αποτελούν πόρτες. Έτσι, καθώς οι πραγματικές πόρτες θα έχουν μεγαλύτερο ποσοστό, θα επιστρέφονται ορθά από τον αλγόριθμο. Τέλος, σημειώνεται ότι προστίθεται και μια μηδενική τιμή στην λίστα, ώστε σε περίπτωση που όλοι οι κόμβοι αποτελούν πόρτες, να μπορεί να τεθεί το όριο στο μηδέν και να φέρει ο αλγόριθμος τα σωστά αποτελέσματα.

Το τελευταίο βήμα είναι η σύγκριση κάθε τιμής με το κατώφλι που υπολογίστηκε και η τελική απόφαση των κόμβων πόρτας κάθε χώρου.

Να σημειωθεί ότι η διαδικασία αυτή θεωρεί την ύπαρξη έστω μίας πόρτας στον χάρτη. Σε περίπτωση που δεν υπάρχει, θα θεωρήσει ως πόρτα το σημείο που προσεγγίζει με μεγαλύτερη ακρίβεια τα χαρακτηριστικά των πορτών που σχολιάστηκαν νωρίτερα.

Αλγόριθμος 5.4 Closest Obstacle Brushfire

```
1: function closestObstacleBrushfire(start, ogm)
2:     brushfire = np.zeros(ogm.shape)
3:     brushfire[ogm > 49] = 1
4:     brushfire[ogm == −1] = −1
5:     brushfire[start] = 2
6:     final_obstacles = []
7:     step = 2
8:     changed = 1
9:     last = 1
10:    found = 0
11:    while changed == 1 and last == 0 do
12:        changed = 0
13:        if found == 1 then
14:            last = 1
15:        end if
16:        for i = 1 to (brush.width - 1) do
17:            for j = 1 to (brush.height - 1) do
18:                if brushfire[i][j] == 0 and a neighbor's brushfire value is equal to
   step then
19:                    brushfire[i][j] = step + 1
20:                    changed = 1
21:                else if brushfire[i][j] == 1 and a neighbor's brushfire value is equal
   to step then
22:                    brushfire[i][j] = −2
23:                    found = 1
24:                end if
25:            end for
26:        end for
27:        step+ = 1
28:    end while
29:    Set obstacles equal to indexes where brushfire[][] == −2
30:    Group obstacles to neighborhoods
31:    for each neighborhood do
32:        Compute closest obstacle ob to start
33:        final_obstacles.append(ob)
34:    end for
35:    return final_obstacles
```

Αλγόριθμος 5.5 Find Door Nodes

```
1: function findDoorNodes(nodes, ogm, gvd)
2:     doorNodes = [], candidateDoors = [], perc = []
3:     for each node in nodes do
4:         ob = closestObstacleBrushfire(node, ogm)
5:         if length(ob) == 2 then
6:             candidateDoors.append(node)
7:             find line of 2 obstacles ob
8:                                     # Door nodes should be inside obstacles' line
9:             if node's distance from line is larger than 5 then
10:                Continue
11:            end if
12:            Collect indexes of points of line extended by 50 pixels at each direction
   outside the ob line segment and with 5 pixel width
13:            Compute occupancy percentage of extended line
14:            Append percentage to perc
15:        end if
16:    end for                                          # Compute right threshold
17:    perc.append(0.0)
18:    if length(perc) > 1 then
19:        perc_copy = numpy.array(perc).sort()
20:        diff = numpy.diff(perc_copy)
21:        max_value = max(diff), max_index = diff.argmax()
22:        threshold = perc_copy[max_index]
23:        while threshold > 0.20 do
24:            max = 0, current_index = −1
25:            for i in range(len(diff)) do
26:                if diff[i] > max and diff[i] not in max_values then
27:                    max = diff[i], current_index = i
28:                end if
29:            end for
30:            max_values.append(max)
31:            threshold = perc_copy[current_index]
32:        end while
33:    else if length(perc) == 1 then
34:        threshold = perc[0] − 0.001
35:    else
36:        return None
37:    end if
38:    for each node in candidateDoors do
39:        if perc of node > threshold then
40:            doorNodes.append(node)
41:        end if
42:    end for
43:    return doorNodes
```

5.1.3 Διαχωρισμός Κόμβων σε Δωμάτια

Το τελευταίο μέρος της μελέτης του χώρου αποτελείται από διαχωρισμό του συνόλου των κόμβων του χάρτη στα επιμέρους δωμάτια στα οποία ανήκουν. Ο αλγόριθμος 5.8 δέχεται ως είσοδο το GVD και το OGM του χώρου, το σύνολο των κόμβων και τους κόμβους πόρτες και επιστρέφει δύο λίστες, την $rooms$ που περιέχεί τους κόμβους διαχωρισμένους σε επιμέρους λίστες άνα δωμάτιο και την $roomDoors$ που περιέχει επιμέρους λίστες των πορτών κάθε δωματίου. Κατά την υλοποίηση του αλγορίθμου δημιουργήθηκε ένας ακόμη αλγόριθμος που στοχεύει στην εύρεση των γειτονικών κόμβων ενός κόμβου εκτελώντας brushfire πάνω στο GVD από τον αρχικό κόμβο μέχρι τους επόμενους γειτονικούς. Δημιουργήθηκαν, μάλιστα, δύο παραλλαγές, οι 5.6 και 5.7 προκειμένου να διαχωρίζονται οι γείτονες των κόμβων πόρτας σε δύο διαφορετικές λίστες στην δεύτερη περίπτωση αντί για μία, για την ευκολότερη διαχείρηση των δεδομένων.

Ο αλγόριθμος διαχωρισμού δωματίων 5.8 εκκινεί από κάθε πόρτα του χώρου και αναζητά γειτονικούς κόμβους κινούμενος πάνω στο GVD και τους κατατάσσει στα αντίστοιχα δωμάτια. Η διαδικασία πραγματοποιείται για κάθε πλευρά της πόρτας, δηλαδή για κάθε δωμάτιο και επαναληπτικά για όλες τις πόρτες του χάρτη, ώστε να καλυφθούν όλοι κόμβοι των επιμέρους χώρων. Ο αλγόριθμος αυτός είναι χρήσιμος και στην ταξινόμηση του κάθε χώρου σε δωμάτιο ή διάδρομο, όπως μελετήθηκε στο Α΄.

Αλγόριθμος 5.6 Gvd Neighbor Brushfire

1: **function** GVDNEIGHBORBRUSHFIRE(start, nodes, gvd)
2: $brushfire = np.zeros(gvd.shape)$
3: $brushfire[gvd == 1] = 1$
4: $brushfire[nodes] = -1$
5: $brushfire[start] = 2$
6: $step = 2$, $changed = 1$
7: **while** $changed == 1$ **do**
8: $changed = 0$
9: **for** i = 1 to ($brushfire.width$ - 1) **do**
10: **for** j = 1 to ($brushfire.height$ - 1) **do**
11: **if** $brushfire[i][j] == 1$ and a neighbor's brushfire value is equal to step **then**
12: $brushfire[i][j] = step + 1$, $changed = 1$
13: **else if** $brushfire[i][j] == -1$ and a neighbor's brushfire value is equal to step **then**
14: $brushfire[i][j] = -2$
15: **end if**
16: **end for**
17: **end for**
18: $step += 1$
19: **end while**
20: Set ind equal to indexes where $brushfire[][] == -2$
21: **return** ind

Αλγόριθμος 5.7 Gvd Neighbor Split Brushfire

```
1: function gvdNeighborSplitBrushfire(start, nodes, gvd)
2:     brushfire = np.zeros(gvd.shape)
3:     brushfire[gvd == 1] = 1
4:     brushfire[nodes] = −1
5:     brushfire[start] = 2
6:     Compute 2 gvd neighbors nn of start point
7:     brushfire[nn[0]] = 3
8:     step = 3, changed = 1
9:     while changed == 1 do
10:        changed = 0
11:        for i = 1 to (brushfire.width - 1) do
12:            for j = 1 to (brushfire.height - 1) do
13:                if brushfire[i][j] == 1 and a neighbor's brushfire value is equal to step then
14:                    brushfire[i][j] = step + 1, changed = 1
15:                else if brushfire[i][j] == −1 and a neighbor's brushfire value is equal to step then
16:                    brushfire[i][j] = −2
17:                end if
18:            end for
19:        end for
20:        step+ = 1
21:    end while
22:    Set ind[0] equal to indexes where brushfire[][] == −2
23:    brushfire[nn[1]] = 3
24:    step = 3, changed = 1
25:    while changed == 1 do
26:        changed = 0
27:        for i = 1 to (brushfire.width - 1) do
28:            for j = 1 to (brushfire.height - 1) do
29:                if brushfire[i][j] == 1 and a neighbor's brushfire value is equal to step then
30:                    brushfire[i][j] = step + 1, changed = 1
31:                else if brushfire[i][j] == −1 and a neighbor's brushfire value is equal to step then
32:                    brushfire[i][j] = −2
33:                end if
34:            end for
35:        end for
36:        step+ = 1
37:    end while
38:    Set ind[1] equal to indexes where brushfire[][] == −2
39:    return ind
```

Αλγόριθμος 5.8 Find Rooms

```
1: function findRooms(gvd, doors, nodes, ogm)
2:     rooms = [], roomDoors = [], visited = []
3:     for each door in doors do
4:         visited.append(door)
5:         nn = gvdNeighborSplitBrushfire(door, nodes, gvd)
6:         Discard nodes of nn that are in visited
7:         for i in range(2) do
8:             current_room = [], next = []
9:             if length(nn[i]) == 0 then
10:                Continue
11:            end if
12:            current_room.append(nn[i][0])                    # Start from first node
13:            next = nn[i][1 :]                    # Add rest to list for next iteration
14:            current = current_room[:], all_doors = []
15:            all_doors.append(door)
16:            while current! = [] do
17:                for node in current do
18:                    if node in doors and node! = door then
19:                        all_doors.append(node)
20:                    else if node not in doors then
21:                        visited.append(node)
22:                        node_nn = gvdNeighborBrushfire(node, nodes, gvd)
23:                        for each node n in node_nn do
24:                            if n in doors and n! = door and n not in all_doors then
25:                                all_doors.append(n)
26:                            else if n not in visited and n not in doors then
27:                                visited.append(n)
28:                                next.append(n)
29:                                current_room.append(n)
30:                            end if
31:                        end for
32:                    end if
33:                end for
34:                current = next, next = []
35:            end while
36:            roomDoors.append(all_doors)
37:            rooms.append(current_room)
38:        end for
39:    end for
40:    return rooms, roomDoors
```

5.2 Βελτιστοποιηση Ακολουθιας Επισκεψης των Δωματιων

Το επόμενο βήμα της μελέτης αυτής είναι ο υπολογισμός της βέλτιστης σειράς επίσκεψης των δωματίων από το ρομποτικό όχημα. Μετρική βελτίωσης της διαδρομής αποτελεί η συνολική απόσταση που χρειάζεται να διανύσει το όχημα. Το πρόβλημα αυτό αποτελεί το γνωστό *πρόβλημα του περιπλανώμενου πωλητή - TSP*. Η προσέγγιση που προτείνεται χρησιμοποιεί δύο παραλλαγές του Hill Climbing αλγόριθμου που ονομάζονται *RRHC* και *anneal HC*, η σύγκριση των οποίων θα πραγματοποιηθεί μέσω των πειραμάτων. Επιπλέον, για τον γρήγορο υπολογισμό των αποστάσεων ανάμεσα σε όλες τις πόρτες δημιουργείται ο αντίστοιχος γράφους του χώρου. Τέλος, είναι χρήσιμο να υπολογιστεί η πόρτα εισόδου και η πόρτα εξόδου για κάθε δωμάτιο σύμφωνα με την διαδρομή που έχει βρεθεί, κάτι που θα φανεί περισσότερο στην επόμενη ενότητα. Η διαδικασία, λοιπόν, αποτελείται από τα παρακάτω βήματα:

- δημιουργία γράφου με κόμβους τις πόρτες του χάρτη
- υπολογισμός όλων των αποστάσεων
- υπολογισμός βέλτιστης διαδρομής
- αντιστοιχία αλληλουχίας πορτών με αλληλουχία δωματίων
- υπολογισμός πόρτας εισόδου και εξόδου για κάθε δωμάτιο δεδομένης της διαδρομής που βρέθηκε

5.2.1 Δημιουργία Γράφου Δωματίων

Αρχικά, λοιπόν, δημιουργείται ένας γράφος, με σκοπό τον ακριβή υπολογισμό των αποστάσεων μεταξύ όλων των πορτών. Η χρήση του ενισχύει την ταχύτητα και την ακρίβεια της διαδικασίας. Ο χώρος μπορεί κάλιστα να προσεγγιστεί από έναν γράφο με την μορφή που αναλύεται στη συνέχεια. Οι κόμβοι πόρτας που έχουν υπολογιστεί αποτελούν τους κόμβους του γράφου, οι αποστάσεις τους αποτελούν τα βάρη των ακμών ενώ συνδεδεμένοι είναι οι κόμβοι που ανήκουν στο ίδιο δωμάτιο. Οι αποστάσεις μεταξύ των γειτονικών κόμβων υπολογίζονται με τη χρήση brushfire με αρχή το ένα σημείο και πέρας το δεύτερο 5.9, προσεγγίζοντας έτσι την πραγματική απόσταση μεταξύ τους στον ελεύθερο χώρο. Αντίθετα, για τους μη γειτονικούς κόμβους χρησιμοποιείται ο αλγόριθμος Dijkstra 3.1 για κάθε συνδιασμό κόμβων. Με τον τρόπο αυτό υπολογίζεται μια ικανοποιητικά ακριβής απόσταση μεταξύ όλων των κόμβων πόρτας του χώρου, ενώ αποφεύγεται η χρήση ευκλείδιων αποστάσεων που χάνουν σημαντικά σε ακρίβεια και η χρήση brushfire μεταξύ όλων των κόμβων που απαιτεί μεγαλύτερο χρόνο εκτέλεσης.

Η δημιουργία του γράφου βασίζεται στην ήδη υλοποιημένη δομή Dijkstar [17]. Αυτή περιέχει κώδικα γραμμένο σε *Python* με μεθόδους που δίνουν τη δυνατότητα εύκολης δημιουργίας ενός γράφου και εκτέλεσης του αλγορίθμου Dijkstra σ' αυτόν. Όπως φαίνεται και στον 5.10 γίνεται χρήση των παρακάτω μεθόδων της δομής Dijkstar.

- $Graph()$: κατασκευαστής (constructor) της κλάσης του γράφου

Αλγόριθμος 5.9 Point to Point Brushfire

```
1: function pointToPointBrushfire(start, finish, ogm)
2:     brushfire = np.zeros(ogm.shape)
3:     brushfire[ogm > 49] = 1
4:     brushfire[ogm == −1] = 1
5:     brushfire[start] = 2
6:     brushfire[finish] = −1
7:     step = 2
8:     changed = 1
9:     iters_made = 0
10:    found = 0
11:    while changed == 1 and not found do
12:        changed = 0
13:        for i = 1 to (brushfire.width - 1) do
14:            for j = 1 to (brushfire.height - 1) do
15:                if brushfire[i][j] <= 0 and a neighbor's brushfire value is equal to
   step then
16:                    if brushfire[i][j] == −1 then
17:                        found = 1
18:                    end if
19:                    brushfire[i][j] = step + 1
20:                    changed = 1
21:                end if
22:            end for
23:        end for
24:        step+ = 1
25:        iters_made+ = 1
26:    end while
27:    if brushfire[finish] <= 0 then
28:        iters_made = −1
29:    end if
30:    return iters_made
```

- $add\_edge(node\_1, node\_2, weight)$: μέθοδος δημιουργίας ακμής στον γράφο μεταξύ δύο σημείων και με βάρος ίσο με $weight$

- $find\_path(graph, node\_1, node\_2)$: μέθοδος υπολογισμού το συντομότερου μονοπατιού μεταξύ δύο κόμβων του γράφου με τον αλγόριθμο Dijkstra

Υπολογίζονται τελικά οι αποστάσεις κάθε σημείου ως προς κάθε άλλο και αποθηκεύονται σε έναν διδιάστατο πίνακα. Επιπλέον, ο αλγόριθμος επιστρέφει και την δημιουργημένη δομή του γράφου, ώστε να μπορεί να αξιοποιηθεί και στα επόμενα τμήματα της διαδικασίας.

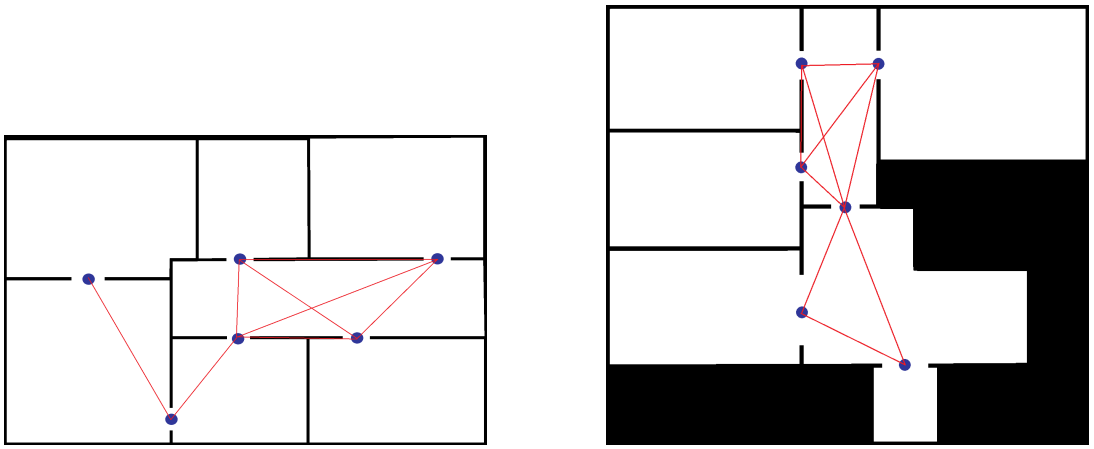
Αλγόριθμος 5.10 Compute Door Distances

```
1: function computeDoorDistances(door_nodes, room_doors)
2:     graph = Graph()                                          # Create graph
3:     for room in room_doors do
4:         if len(room) > 1 then
5:             for i in range(len(room)) do
6:                 for j in range(i + 1, len(room)) do
7:                     node_id_1 = door_nodes.index(room[i])
8:                     node_id_2 = door_nodes.index(room[j])
9:                     dist = pointToPointBrushfire(room[i], room[j], ogm)
10:                    if dist < 0 then
11:                        return
12:                    end if
13:                    graph.add_edge(node_id_1, node_id_2, dist)
14:                    graph.add_edge(node_id_2, node_id_1, dist)
15:                end for
16:            end for
17:        end if
18:    end for
19:                                  # Calculate graph's distances between all nodes
20:    doors_length = len(door_nodes)
21:    if doors_length >= 2 then
22:        distances = np.zeros((doors_length, doors_length))
23:        for i in range(doors_length) do
24:            for j in range(i + 1, doors_length) do
25:                _, _, _, dist = find_path(graph, i, j)
26:                distances[i][j] = dist
27:                distances[j][i] = distances[i][j]
28:            end for
29:        end for
30:    else
31:        distances[0][0] = 0
32:    end if
33:    return distances, graph
```

Δύο παραδείγματα δημιουργίας του εν λόγω γράφου πάνω στο OGM παρουσιάζονται στα σχήματα 5.8. Με μπλε έχουν σημειωθεί όλες οι πόρτες κάθε περιβάλλοντος, ενώ οι συνδεδεμένοι κόμβοι, δηλαδή οι κόμβοι που ανήκουν στο ίδιο δωμάτιο, έχουν ενωθεί με κόκκινη γραμμή.

5.2.2 Υπολογισμός Βέλτιστης Διαδρομής

Η βέλτιστη διαδρομή υπολογίζεται, αρχικά, με τον αλγόριθμο *anneal HC*. Αυτός αποτελεί μια παραλλαγή του vanilla HC στην οποία γίνονται αποδεχτές πιθανοτικά και μεταβολλές της λύσης που δεν βελτιώνουν το τελικό αποτέλεσμα 3.5. Η συνάρ-



Σχήμα 5.8: Παραδείγματα Γράφων πάνω στο OGM

τηση βελτιστοποίησης είναι η συνολική απόσταση μιας ολόκληρης κυκλικής διαδρομής. Επισημαίνεται ότι χρησιμοποιείται στο άθροισμα και η απόσταση μεταξύ του τελευταίου κόμβου προς τον πρώτο προκειμένου να βρεθεί μια γενικευμένη λύση η οποία δεν καθορίζεται από την αρχή της διαδρομής. Αυτό δίνει την δυνατότητα στο ρομποτικό όχημα να μπορεί να εκκινήσει από ένα οποιοδήποτε δωμάτιο και να εκτελέσει κυκλικά την διαδρομή μέχρι να τα επισκεφτεί όλα. Άλλωστε είναι θεμιτό να αρχίζει η επίσκεψη των δωματίων με πρώτο το δωμάτιο στο οποίο βρίσκεται αρχικά το όχημα.

Η υλοποίηση του αλγορίθμου βασίστηκε σε μια ήδη υπάρχουσα υλοποίηση [18] και παρουσιάζεται στο 5.11. Περιέχει τόσο την βάση της HC μεθόδου, που μεταβάλλει την λύση προς τα καλύτερα αποτελέσματα, όσο και την στοχαστικότητα της anneal τεχνικής. Αναλυτικότερα, η τιμή T επιλέγεται στο 1.0 και η τιμή $alpha$ στο 0.95. Η καλύτερη λύση $best$ αρχικοποιείται ως ο αύξων δείκτης των κόμβων και υπολογίζεται το κόστος της $best\_score$. Για κάθε βέλτιστη λύση υπολογίζονται όλοι οι πιθανοί συνδιασμοί αντιστροφής τμημάτων της. Για παράδειγμα στην αλληλουχία 1-2-3-4-5-6-7-8 δύο πιθανές αντιστροφές επιφέρουν τις εξής ακολουθίες 1-2-5-4-3-6-7-8 και 1-2-3-4-8-7-6-5. Αυτό έχει ως αποτέλεσμα να μεταβάλλεται η λύση προς κάθε κατεύθυνση και να εξετάζεται το κόστος της κάθε πιθανής διαδρομής. Η αποδοχή ή μη της κάθε μεταβολής εξαρτάται από τον πιθανοτικό παράγοντα p ο οποίος εισάγει την στοχαστικότητα στον αλγόριθμο αυτό μέσω της τιμής T. Αφότου εξεταστούν όλες οι πιθανές μεταβολές της αλληλουχίας ή βρεθεί κάποια νέα καλύτερη σειρά, η τιμή T μειώνεται ποσοστιαία κατά $1 - alpha$, έτσι ώστε με την πάροδο των επαναλήψεων να γίνονται δεκτές όλο και λιγότερες λύσεις που δυσχεραίνουν το κόστος της διαδρομής. Επιπλέον, η συνολική διαδικασία τερματίζεται είτε όταν συμπληρωθούν οι επαναλήψεις που ορίστηκαν στην μεταβλητή εισόδου $iterations$ είτε εαν καμία μεταβολή της λύσης δεν επιλεγεί αντί της τρέχουσας βέλιστης πράγμα που σημαίνει ότι ο αλγόριθμος έφτασε σε τοπικό ελάχιστο. Τέλος, επιστέφονται η βέλτιστη διαδρομή, το κόστος της και το πλήθος των επαναλήψεων που χρειάστηκαν.

Αλγόριθμος 5.11 Anneal Hill Climb

```
1: function ANNEAL(distances, iterations, start_temp, alpha)
2:     length = distances.shape[0]
3:     best = range(length)
4:     Set best_score equal to the length of route best
5:     iter = 0
6:     done = False
7:     T = start_temp
8:     while not done do
9:         Find all possible reversed sections of current route best
10:        move_made = False
11:        for each reversed route next do
12:            if iter >= iterations then
13:                done = True
14:                Break
15:            end if
16:            Compute length of next route and set it to next_score
17:            iter+ = 1
18:            if next_score > best_score then
19:                p = 1
20:            else
21:                p = math.exp(−abs(next_score − prev_score)/T)
22:            end if
23:                                        # probablistically accept this solution
24:            if random.random() < p then
25:                best = next
26:                best_score = next_score
27:                move_made = True
28:                break
29:            end if
30:        end for
31:        T = T ∗ alpha
32:        if not move_made then
33:            Break
34:        end if
35:    end while
36:    return door_route, cost, iter
```

Μετά υλοποιείται ο *RRHC* αλγόριθμος. Η ανάπτυξη του βασίστηκε και αυτή στην υλοποίση του [18]. Αυτός καλεί τον απλό HC μέχρι αυτός να φτάσει σε ένα τοπικό μέγιστο και κρατάει τη λύση του ως τελική λύση. Στη συνέχεια, επανεκκινεί τη διαδικασία μέχρι να συμπληρωθεί το πλήθος των επαναλήψεων που έχει οριστεί εξαρχής. Φυσικά, κάθε νέο τοπικό μέγιστο που αποτελεί βελτίωση του τρέχοντος καλύτερου, το αντικαθιστά. Ο RRHC αναλύεται στο 5.12.

Αλγόριθμος 5.12 Random Restard Hill Climb

```
1: function randomRestartHillClimb(distances, epochs, fixed_edges = False)
2:     best = None, best_score = 0, iter = 0
3:     while iter < epochs do
4:         iters_left = epochs − iter
5:         sequence, score, iters_made = hillClimb(distances, iters_left, fixed_edges)
6:         iter+ = iters_made
7:         if score < best_score or best is None then
8:             best_score = score, best = sequence
9:         end if
10:    end while
11:    return best, best_score, iter
```

Αλγόριθμος 5.13 Hill Climb

```
1: function HillClimb(distances, epochs, fixed_edges = False)
2:     length = distances.shape[0]
3:     best = range(length)
4:     Set best_score equal to the length of route best
5:     iter = 1
6:     while iter < epochs do
7:         move_made = False
8:         for each reversed route next do
9:             if iter >= epochs then
10:                break
11:            end if
12:            Set next_score equal to the length of route next
13:            iter + +
14:            if next_score < best_score then
15:                best = next, best_score = next_score, move_made = True
16:                break
17:            end if
18:        end for
19:        if not move_made then
20:            break
21:        end if
22:    end while
23:    return best, best_score, iter
```

5.2.3 Αντιστοιχία Κόμβων με Δωμάτια

Ένα απλό αλλά σημαντικό βήμα είναι η αντιστοιχία των κόμβων σε δωμάτια. Η αλληλουχία που έχει υπολογιστεί αφορά τις πόρτες του χώρου, όμως το όχημα χρειάζεται να καλύψει τα διάφορα δωμάτια. Έτσι, δημιουργείται η αλληλουχία δωματίων που προκύπτει από τις πόρτες παίρνοντας για κάθε μία τα δύο δωμάτια

που διαχωρίζει, το αριστερό και το δεξί με αυτή την σειρά, δίχως αυτή η θεώρηση να επηρεάζει συνολικά, αφού και τα δύο δωμάτια θα προσπελαστούν. Φυσικά, κάθε φορά ελέγχεται η περίπτωση ένα δωμάτιο να υπάρχει ήδη στην αλληλουχία από μια προηγούμενη πόρτα.

5.2.4 Υπολογισμός Πόρτας Εισόδου και Εξόδου

Στο τελευταίο βήμα εντοπίζονται οι πόρτες από τις οποίες το όχημα εισέρχεται σε κάθε δωμάτιο και αυτές από τις οποίες εξέρχεται. Αυτό επιτυγχάνεται προσομοιώνοντας την διαδικασία επίσκεψης των διάφορων δωματίων με την σειρά που έχει υπολογιστεί. Χρησιμοποιείται ο γράφος που έχει δημιουργηθεί στο πρώτο βήμα της μελέτης αυτής, καθώς ουσιαστικά κάθε δωμάτιο αποτελεί τις ακμές του γράφου αυτού. Έτσι, ελέγχονται οι κόμβοι που προσπελάσσονται πριν και μετά από την επίσκεψη των ακμών αυτών. Ο αλγόριθμος παρουσιάζεται στο 5.15.

Ο τελικός αλγόριθμος της ενότητας αυτής παρουσιάζεται στην συνέχεια στο 5.14. Εδώ παρουσιάζεται η περίπτωση της χρήσης του RRHC, ωστόσο για την χρήση του anneal αλγορίθμου απαιτείται αλλαγή μόνο στην 7η εντολή, χρησιμοποιώντας την εντολή:

$door\_route, cost, iter = anneal(distances, max\_iterations, 1.0, 0.95)$

Αλγόριθμος 5.14 Find Room Sequence

```
1: function findRoomSequence(door_nodes, room_doors)
2:     room_sequence = []
3:     doors_length = len(door_nodes)
4:     if doors_length >= 2 then
5:         distances, graph = computeDoorDistances(door_nodes, room_doors)
6:         max_iterations = 500 * doors_length
7:         door_route, cost, iter = rrhc(distances, max_iterations)
8:     else
9:         door_route = [0]
10:    end if
11:                                   # Correspond door route to room route
12:    for each door in door_route do
13:        for i in range(len(room_doors)) do
14:            if door_nodes[door] in room_doors[i] and i not in room_sequence then
15:                room_sequence.append(i)
16:            end if
17:        end for
18:    end for
19:    enter, leave = findEnteringLeavingDoors(graph, room_sequence,
   room_doors, door_route)
20:    return room_sequence, enter, leave
```

Αλγόριθμος 5.15 Find Entering Leaving Doors

```
1: function findEnteringLeavingDoors(graph, room_sequence, room_doors, door_route)
2:     enter = {}, leave = {}
3:     if doors_length >= 2 then
4:         room_i = 1                                   # Start from 2nd room
5:         room_idx = room_sequence[room_i]
6:         entered = False
7:         for i in range(len(door_route)) do
8:                                                      # Find path from current to next door
9:             door_idx = door_route[i]
10:            next_door_idx = door_route[(i + 1)%len(door_route)]
11:            path = find_path(graph, door_idx, next_door_idx)
12:            if entered and len(path.nodes) > 1 then  # if already inside room, first node of path is already used
13:                ii = 1
14:            else
15:                ii = 0
16:            end if
17:                    # Transverse through path and check for corresponding rooms
18:            while ii < len(path.nodes) do
19:                node_idx = path.nodes[ii], door = door_nodes[node_idx]
20:                if door in room_doors[room_idx] then
21:                    if not entered then
22:                        enter[room_idx] = door
23:                        entered = not entered
24:                        if len(room_doors[room_idx]) == 1 then
25:                            ii− = 1                  # To revisit this door
26:                        end if
27:                    else
28:                        leave[room_idx] = door
29:                        entered =not entered
30:                        room_i+ = 1
31:                        room_idx = room_sequence[room_i%len(room_sequence)]
32:                        ii− = 1                      # To revisit this door
33:                    end if
34:                end if
35:                ii+ = 1
36:            end while
37:        end for
38:    else
39:        for i in range(length(room_sequence)) do
40:            enter[i] = door_nodes[0], leave[i] = door_nodes[0]
41:        end for
42:    end if
43:    return enter, leave
```

5.3 Πλοηγηση Οχηματος και Καλυψη του Χωρου

Σημαντικό μέρος της εργασίας αποτελεί η δημιουργία του συστήματος πλοήγησης του ρομποτικού οχήματος στον χώρο και η συνεχής καταγραφή των περιοχών που καλύπτουν οι κεραίες που φέρει. Τα δύο αυτά συστήματα είναι η βάση όλης της μελέτης της παρούσας εργασίας. Η υλοποίηση τους παρουσιάζεται στη συνέχεια.

5.3.1 Υλοποίηση Πλοήγησης

Η πλοήγηση του ρομποτικού οχήματος στον εικονικό κόσμο επιτυγχάνεται με τη χρήση του Navigation Stack 4.3. Αυτό διευκολύνει σημαντικά τη διαδικασία, καθώς απαιτεί μόνο τις τρεις μεταβλητές του σημείου στόχου, δηλαδή τη θέση και τον προσανατολισμό του. Το stack παράγει την τροχιά που χρειάζεται να ακολουθήσει το όχημα για να βρεθεί στον ζητούμενο στόχο και του δίνει τις κατάλληλες εντολές ταχύτητας.

Η ανάπτυξη του συστήματος βασίστηκε στο [19] και παρουσιάζεται στους επόμενους δύο αλγορίθμους. Ο πρώτος 5.16 διαβάζει τη θέση και τον προσανατολισμό του σημείου στόχου, τον μετασχηματίζει στην κατάλληλη μορφή, δημιουργεί το μήνυμα *goal* και το στέλνει στο stack περιμένοντας την επιτυχή εκτέλεση του. Ο δεύτερος 5.17 αποτελεί τον βασικό αλγόριθμο της διαδικασίας και υπολογίζει το τρέχον δωμάτιο διαβάζοντας τη θέση του οχήματος στο OGM και για κάθε δωμάτιο σύμφωνα με την αλληλουχία που έχει υπολογιστεί ανακαλεί τα σημεία στόχους του. Κάθε σημείο διαχωρίζεται σε θέση και προσανατολισμό και καλείται η συνάρτηση *goToGoal*. Τέλος, μόλις το όχημα προσεγγίσει τον στόχο παραμένει εκεί για 0.1 δευτερόλεπτα, για την καλύτερη κάλυψη του χώρου, πριν συνεχίσει για το επόμενο σημείο.

Αλγόριθμος 5.16 Go To Goal

```
1:  function goToGoal(target, yaw)
2:      Set resolution equal to map's resolution
3:      Set origin equal to map's origin
4:      move_base_client.wait_for_server()
5:      goal = MoveBaseGoal()
6:      goal.target_pose.header.frame_id =' map'
7:      goal.target_pose.header.stamp = rospy.Time.now()
8:      goal.target_pose.pose.position.x = target[0] * resolution + origin['x']
9:      goal.target_pose.pose.position.y = target[1] * resolution + origin['y']
10:     quaternion = tf.transformations.quaternion_from_euler(0, 0, yaw)
11:     goal.target_pose.pose.orientation.x = quaternion[0]
12:     goal.target_pose.pose.orientation.y = quaternion[1]
13:     goal.target_pose.pose.orientation.z = quaternion[2]
14:     goal.target_pose.pose.orientation.w = quaternion[3]
15:     move_base_client.send_goal(goal)
16:     wait = move_base_client.wait_for_result()
17:     return move_base_client.get_result()
```

Αλγόριθμος 5.17 Navigation

```
1: function NAVIGATION(sequence, rooms)
2:     Read robot's current pose
3:     Compute current room
4:     for each room in sequence starting from current do
5:         for each pose in room do
6:             result = goToGoal(pose['position'], pose['yaw'])
7:             rospy.sleep(0.1)
8:         end for
9:     end for
10:    return
```

5.3.2 Υλοποίηση Κάλυψης Χώρου

Η δεύτερη υλοποίηση αφορά την συνεχή καταγραφή της κάλυψης του χώρου. Το σύστημα αυτό λειτουργεί παράλληλα με όλα τα υπόλοιπα χωρίς να τα επηρεάζει ως μια ξεχωριστή διεργασία.

Το όχημα φέρει ένα σύνολο αισθητήρων με χαρακτηριστικά τα οποία ορίζονται στην αρχή της διαδικασίας και είναι το μήκος της ακτίνας κάλυψης, το γωνιακό εύρος κάλυψης FOV και η γωνία τοποθέτησης της κεραίας ως προς το εμπρός μέρος του οχήματος. Το σχήμα του πεδίου κάλυψης είναι ουσιαστικά ένα τμήμα ενός κύκλου, όπως φαίνεται και στις δύο εικόνες 5.9, όπου με μπλε χρώμα είναι σκιασμένες οι περιοχές κάλυψης. Στην πρώτη το όχημα φέρει μια κεραία με ευρεία γωνία κάλυψης ενώ στην δεύτερη φέρει δύο, μικρότερου εύρους, τοποθετημένες συμμετρικά στις δύο πλευρές του.

Σχήμα 5.9: Παράδειγμα Κάλυψης Αισθητήρων

Στην διπλωματική αυτή εργασία γίνεται μελέτη τριών διαφορετικών στοιχείων της κάλυψης του χώρου κατά την πλοήγηση του οχήματος. Αυτά είναι τα εξής:

- Κάλυψη του χώρου
- Πλήθος διαφορετικών σαρώσεων κάθε σημείου
- Γωνία σάρωσης κάθε σημείου

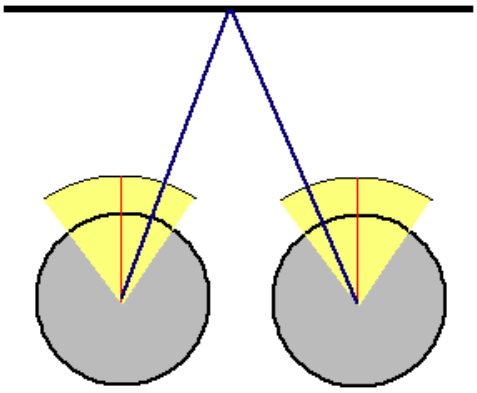
Συγκεκριμένα, το πρώτο αναφέρεται στην vanilla κάλυψη του χώρου, δηλαδή εάν οι αισθητήρες έχουν σαρώσει κάθε σημείο ξεχωριστά. Το δεύτερο αναφέρεται στον αριθμό σαρώσεων κάθε σημείου. Όσες περισσότερες φορές δει ο αισθητήρας κάποιο σημείο, τόσο πιο σίγουρος θα είναι για την ύπαρξη ή μη ετικετών RFID στο σημείο

αυτό. Το τελευταίο αναφέρεται στην γωνία υπό την οποία σάρωσε ο αισθητήρας ένα σημείο. Στην ιδανική περίπτωση στόχος είναι να σαρωθεί ένα εμπόδιο τόσο από το θετικό μισό τόξο του αισθητήρα, όσο και από το αρνητικό. Με τον τρόπο αυτό η ετικέτα RFID θα αναγνωσθεί από διαφορετική φάση, κάτι που οδηγεί σε αποτελεσματικότερο εντοπισμό της θέσης της. Ο εντοπισμός των ετικετών εξαρτάται κυρίως από την ισχύ του λαμβανόμενου σήματος, την απόσταση του από την κεραία και την φάση του σήματος. Συνεπώς, η ακρίβεια αυξάνεται σημαντικά χρησιμοποιώντας πληθώρα διαφορετικών μετρήσεων, όπως αναφέρουν και ο Nikitin κ.ά. [20] και ο Hekimian-Williams κ.ά. [21].

Η πρόταση αυτή γίνεται αντιληπτή στο σχήμα 5.10, όπου ένα σημείου του τοίχου σαρώνεται από τον αισθητήρα δύο φορές, κάθε μία από διαφορετικό μισό του FOV του αισθητήρα. Θεωρώντας την κεντρική ακτίνα του FOV ως τις 0 μοίρες το τόξο κάλυψης χωρίζεται σε δύο ίσα τμήματα, το αριστερό (αρνητικό) και το δεξιό (θετικό). Βέβαια, αυτή η μέτρηση αφορά σημεία τα οποία έχουν σαρωθεί τουλάχιστον μία φορά από τους αισθητήρες. Έτσι, μπορεί να χρησιμοποιηθεί η παρακάτω μετρική:

$$coverage\_angles = \frac{|left - right|}{left + right}$$

Όσο πιο ομοιόμορφη είναι η κατανομή σαρώσεων μεταξύ των δύο τμημάτων και όσο περισσότερες είναι οι σαρώσεις συνολικά, τόσο η μετρική θα προσεγγίζει το μηδέν, αλλιώς θα προσεγγίζει τη μονάδα.



Σχήμα 5.10: Παράδειγμα Κάλυψης Σημείου υπό Διαφορετικών Γωνιών

Συμπερασματικά, η κάθε κάλυψη μπορεί να ποσοτικοποιηθεί στις παρακάτω πέντε μετρικές, οι οποίες και χρησιμοποιήθηκαν σε όλα τα πειράματα κάλυψης χώρου.

- Ποσοστό συνολικής κάλυψης των εμποδίων του χώρου
- Μέση τιμή πλήθους σαρώσεων κάθε σημείου

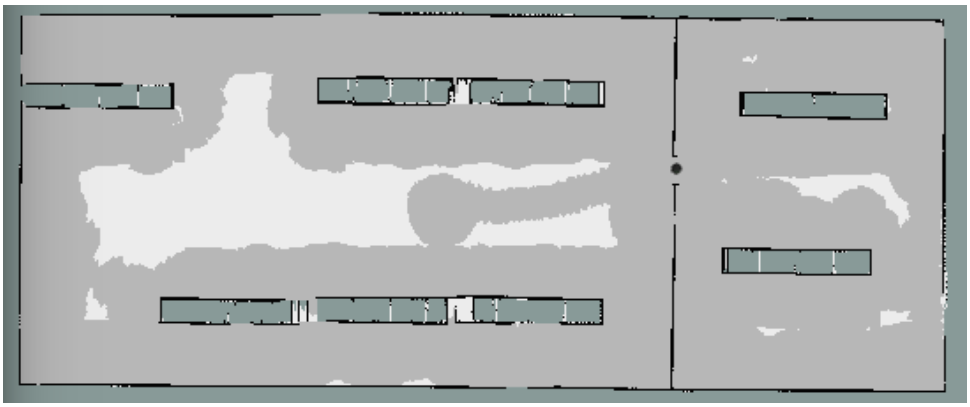
- Διακύμανση πλήθους σαρώσεων κάθε σημείου
- Μέση τιμή μετρικής γωνίας σάρωσης κάθε σημείου που έχει σαρωθεί
- Διακύμανση μετρικής γωνίας σάρωσης κάθε σημείου που έχει σαρωθεί

Η υλοποίηση του συστήματος αυτού περιλαμβάνει δύο συναρτήσεις. Η πρώτη 5.18 είναι αυτή που βρίσκει τα σημεία του OGM που καλύπτονται κάθε φορά από έναν αισθητήρα του ρομποτικού οχήματος. Για δεδομένα χαρακτηριστικά αισθητήρα και πόζα του οχήματος υπολογίζονται τα σημεία με τη χρήση του brushfire αλγορίθμου και της ray casting τεχνικής. Συγκεκριμένα, δημιουργείται μια ακτίνα από την κεραία προς τα έξω, πάνω στην οποία εκτελείται ο αλγόριθμος brushfire μέχρι να συναντήσει εμπόδιο ή να ξεπεράσει την ακτίνα κάλυψης του αισθητήρα. Όλα τα ελεύθερα σημεία στα οποία διαδίδεται ο brushfire είναι αυτά που καλύπτει ο αισθητήρας. Η διαδικασία αυτή εκτελείται επαναληπτικά για όλες τις ακτίνες μέσα στο FOV του αισθητήρα με πολύ μικρό βήμα, ίσο με 0.5 μοίρες. Επίσης, υπάρχει η επιλογή μέσω της μεταβλητής εισόδου *return_obstacles* ο αλγόριθμος να επιστρέψει είτε τα σημεία του ελεύθερου χώρου είτε το σύνολο των εμποδίων που σαρώνονται.

Η μέθοδος αυτή φέρει σημαντικά πλεονεκτήματα. Συγκεκριμένα, με τη χρήση του brushfire, η εξάπλωση περιορίζεται αποκλειστικά στον γύρω ελεύθερο χώρο και δεν σαρώνονται τα σημεία εμποδίων. Πράγματι, η εξάπλωση σταματάει για κάθε ακτίνα μόλις εντοπιστεί εμπόδιο και παραμένει στο ίδιο δωμάτιο. Συνεπώς, δεν σαρώνονται σημεία πίσω από τοίχους, κάτι που ρεαλιστικά οι αισθητήρες αδυνατούν να κάνουν. Επιπλέον, η χρήση της ray casting τεχνικής προσομοιώνει την πραγματική λειτουργία του αισθητήρα, καθώς αυτός σαρώνει σε συγκεκριμένο πεδίο ενός κύκλου το οποίο μπορεί να προσεγγιστεί ικανοποιητικά χρησιμοποιώντας ένα μεγάλο πλήθος ακτίνων.

Ο αλγόριθμος 5.19 συντονίζει τη διαδικασία κάλυψης. Διαβάζει, αρχικά, την τρέχουσα θέση του οχήματος και ελέγχει εαν αυτό έχει κινηθεί, συγκρίνοντας την με την προηγούμενη μέτρηση. Αυτός ο έλεγχος γίνεται για εξοικονόμηση υπολογιστικής ισχύος, καθώς όταν το όχημα παραμένει ακίνητο οι αισθητήρες εντοπίζουν τα ίδια σήματα. Εαν το ρομπότ έχει κινηθεί ως προς οποιαδήποτε από τις τρεις μεταβλητές υπολογίζονται τα σημεία που καλύπτει το σύνολο των αισθητήρων του. Μετά, για κάθε σημείο ανανεώνεται η τιμή κάλυψης του στον πίνακα *coverage*, ο οποίος έχει διαστάσεις ίσες με το OGM του χώρου και αποθηκεύει την κάλυψη ή όχι του κάθε διακριτού σημείου με τιμές 100 και 0, αντίστοιχα. Επίσης, ανανεώνεται η τιμή πλήθους σαρώσεων στον *coverage_number*, αλλά και του πλήθους σαρώσεων κάθε τμήματος γωνιών στον *coverage_angles*, έναν τρισδιάστατο πίνακα με τρίτη διάσταση ίση με 2.

Ένα παράδειγμα του πίνακα *coverage* παρουσιάζεται στο 5.11, όπου φαίνεται το OGM του περιβάλλοντος και με σκούρο γκρι χρώμα είναι σημειωμένα όλα τα σημεία τα οποία καλύφθηκαν από τους αισθητήρες κατά την πλοήγηση του στον χώρο.



Σχήμα 5.11: Παράδειγμα Πεδίου Κάλυψης

Αλγόριθμος 5.18 Circular Ray Cast Coverage

```
1: function circularRayCastCoverage(start, ogm, cover_range, fov, theta, direction,
   return_obstacles = False)
2:     brushfire = numpy.zeros(ogm.shape, np.dtype('int32'))
3:     brushfire[ogm > 49] = 1
4:     brushfire[ogm == -1] = -1
5:     brushfire[start] = 2
6:     th = -fov/2
7:     while th <= fov/2 do
8:         angle = (th + direction + theta)* π /180.0;
9:         step = 2, iters = 1
10:        while iters <= cover_length do
11:            xx = (int)round(start[0] + cos(angle) * iters)
12:            yy = (int)round(start[1] + sin(angle) * iters)
13:            if brushfire[xx][yy] == 0 then
14:                brushfire[xx][yy] = step
15:                step + +
16:            else if brushfire[xx][yy] == 1 then
17:                brushfire[xx][yy] = -2
18:                break
19:            else if brushfire[xx][yy] < 0 then
20:                break
21:            end if
22:            iters + +
23:        end while
24:        th+ = 0.5
25:    end while
26:    if return_obstacles then
27:        indexes = zip(*numpy.where(brushfire == -2))
28:    else
29:        indexes = zip(*numpy.where(brushfire > 1))
30:    end if
31:    return indexes
```

Αλγόριθμος 5.19 Update Cover

```
1: function updateCover(previous_robot_pose, ogm, coverage, coverage_number,
   coverage_angles, resolution, sensor_range, sensor_fov, sensor_direction)
2:     Read robot's pose xx, yy, th
3:     Check if robot has been moved from previous pose
4:     if robot moved then
5:         for each sensor s do
6:             cover_length = int(sensor_range[s]/resolution)
7:             indexes = circularRayCastCoverage((xx, yy), ogm, cover_length,
           sensor_fov[s], th, sensor_direction[s])
8:             for x, y in indexes do
9:                 coverage[x, y] = 100
10:                coverage_number[x, y]+ = 1
11:                angle = th + sensor_direction[s] + math.degrees(math.atan2(yy −
   y, xx − x))
12:                if angle < 0 then
13:                    coverage_angles[x, y][0]+ = 1
14:                else
15:                    coverage_angles[x, y][1]+ = 1
16:                end if
17:            end for
18:        end for
19:    end if
20:    Update previous_robot_pose with current pose
21:    return coverage, coverage_number, coverage_angles, previous_robot_pose
```

5.4 Βελτιστοποιηση Μονοπατιου Καλυψης Δωματιου

Το τελευταίο και μεγαλύτερο τμήμα της εργασίας αυτής είναι η πλήρης κάλυψη κάθε δωματίου. Για να επιτευχθεί αυτό χρειάζεται να υπολογιστούν σημεία στα οποία θα κατευθυνθεί το ρομποτικό όχημα σαρώνοντας, έτσι, έμμεσα τον χώρο. Όλοι οι υπολογισμοί πραγματοποιούνται συναρτήσει των αισθητήρων. Υλοποιήθηκαν δύο διαφορετικές στρατηγικές, η στρατηγική ακολουθίας τοίχων και η στρατηγική ζιγκ ζαγκ. Η διαδικασία αυτή αποτελείται από τα παρακάτω βήματα:

- Δειγματοληπτικός υπολογισμός σημείων κάλυψης
- Υπολογισμός βέλτιστης αλληλουχίας επίσκεψης των σημείων
- Υπολογισμός βέλτιστου προσανατολισμού σε κάθε σημείο
- Προσομοίωση κάλυψης του χώρου και διαγραφή περιττών σημείων
- Δημιουργία αλληλουχίας κόμβων ζιγκ-ζαγκ

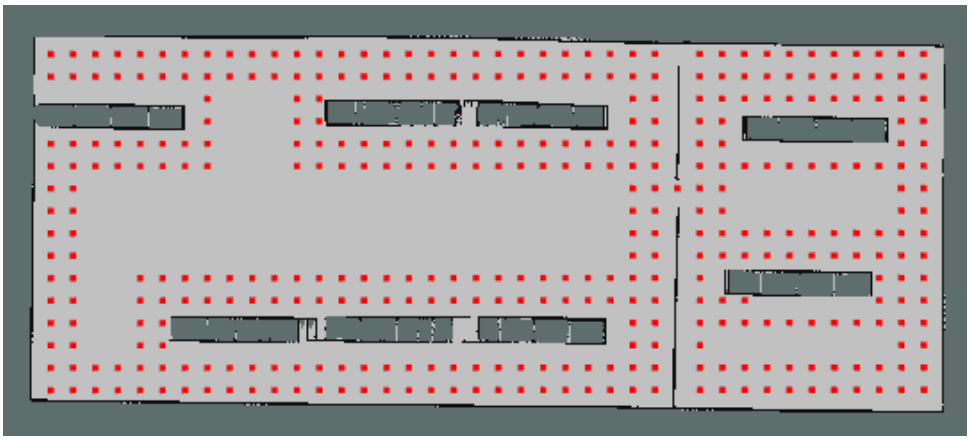
5.4.1 Δειγματοληπτικός υπολογισμός σημείων κάλυψης

Αρχικά, πρέπει να εξαχθεί ένα σύνολο σημείων στα οποία πηγαίνοντας το όχημα θα έχει την δυνατότητα να καταγράψει προϊόντα, δηλαδή σημεία που βρίσκονται

κοντά σε εμπόδια. Το πόσο κοντά, φυσικά, εξαρτάται από την ακτίνα των κεραιών RFID που φέρει το ρομπότ.

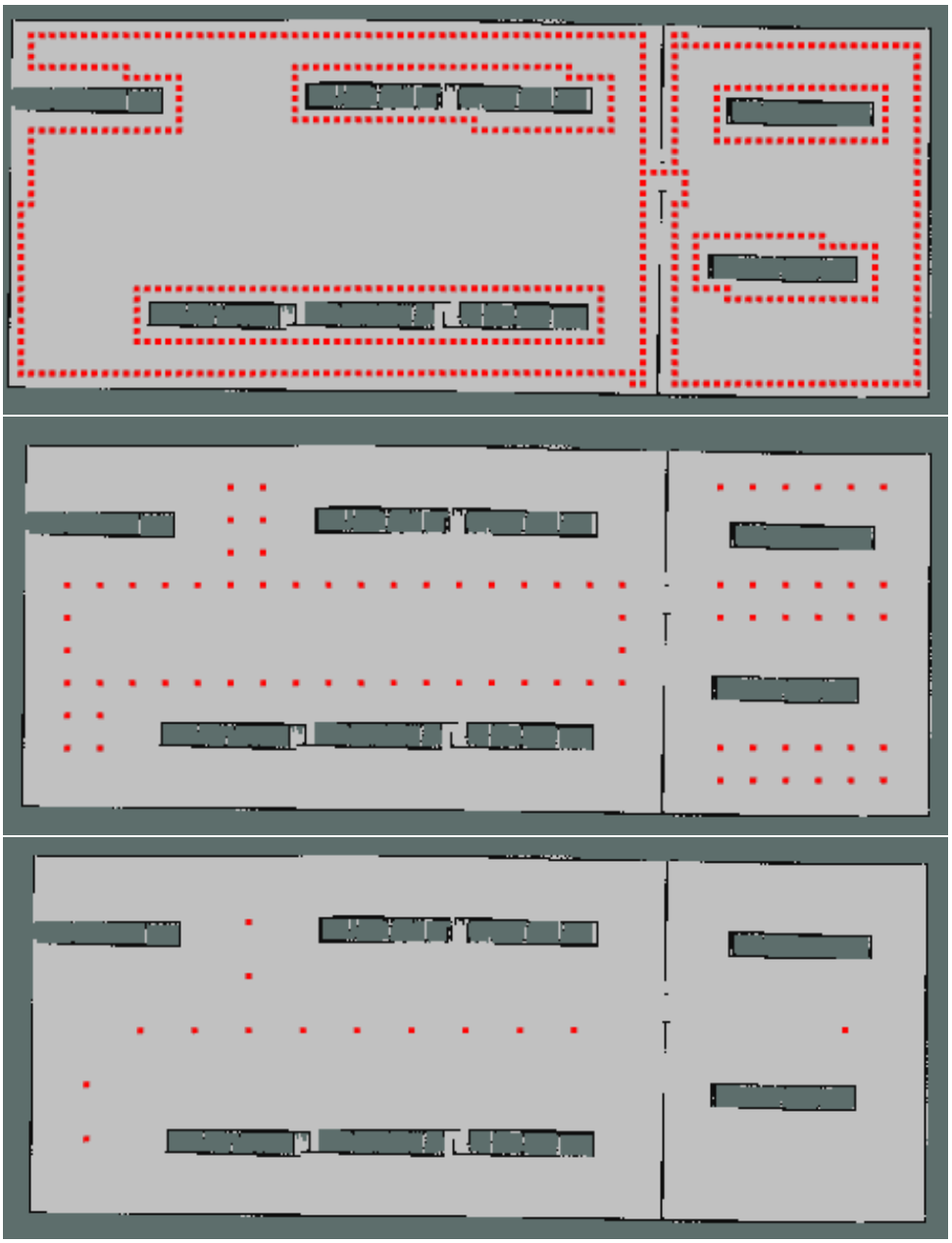
Η μέθοδος που υλοποιήθηκε εφαρμόζει ομοιόμορφη δειγματοληψία επαναληπτικά μεταβάλλοντας το βήμα της και αποθηκεύει όλα τα σημεία τα οποία βρίσκονται σε κοντινή απόσταση από τα γύρω εμπόδια. Το κέρδος με την χρήση πολλών διαφορετικών βημάτων είναι ότι αποθηκεύονται σημεία σε όλο τον ελεύθερο χώρο που μπορεί να επισκευτεί το ρομπότ ανεξαρτήτως της τοπολογίας του.

Πρώτα υπολογίζονται η ελάχιστη και η μέγιστη τιμή των ακτίνων των διάφορων κεραιών που φέρει το ρομπότ. Μετά, επαναληπτικά σαρώνεται ολόκληρος ο χάρτης με κάποιο βήμα και κρατούνται όλα εκείνα τα σημεία που βρίσκονται στον ελεύθερο χώρο και απέχουν απόσταση από τα γύρω εμπόδια στο διάστημα $(step, 2 * step]$. Θέτοντας αυτόν τον περιορισμό, σε κάθε επανάληψη σχηματίζεται μόλις μία σειρά σημείων εσωτερικά των εμποδίων και όχι περισσότερες, κάτι που οδηγεί σε μεγάλο πλεόνασμα σημείων μέσα στην ίδια επανάληψη. Το σχήμα 5.12 αποτελεί μια περίπτωση περισσότερων της μίας σειράς σημείων. Φυσικά, για να γίνει δεκτό ένα σημείο πρέπει οι κεραίες να είναι επαρκώς κοντά σε κάποιο εμπόδιο.



Σχήμα 5.12: Παράδειγμα Αποτυχίας Εύρεσης Σημείων

Το βήμα της δειγματοληψίας αρχίζει από τιμή ίση με μιάμιση φορά την ακτίνα του οχήματος, όσο δηλαδή και η ελάχιστη απόσταση που μπορεί να φτάσει το όχημα από τα εμπόδια για λόγους ασφαλείας και διπλασιάζεται σε κάθε επανάληψη μέχρι να γίνει μεγαλύτερο από την μέγιστη ακτίνα των κεραιών. Ένα τέτοιο παράδειγμα για τα σημεία με διαφορετικό βήμα δειγματοληψίας παρουσιάζεται στο 5.13. Έτσι, όμως, θα υπάρχουν πολλά σημεία απ' όπου το ρομπότ θα σαρώσει τα ίδια εμπόδια, είναι δηλαδή περιττά. Επομένως, πρέπει να γίνει ένας τελικός έλεγχος εαν κάθε σημείο προσφέρει όντως στην κάλυψη, δίνοντας προτεραιότητα στα σημεία που προέκυψαν από την δειγματοληψία με τα μεγαλύτερα βήματα. Για κάθε σημείο υπολογίζονται τα εμπόδια που μπορούν να σαρωθούν από εκεί και εαν περισσότερα από το 90% τους έχουν ήδη καλυφθεί από τα προηγούμενα, τότε αυτό απορρίπτεται. Σημειώνεται ότι τα σημεία που προέκυψαν από το μεγαλύτερο βήμα δειγματοληψίας κρατούνται όλα. Με τη μέθοδο αυτή εξασφαλίζεται ότι θα αποθηκευτούν τελικά τα λιγότερα δυνατά σημεία που καλύπτουν ολόκληρο τον χώρο. Ο αλγόριθμος περιγράφεται αναλυτικά στο 5.20.

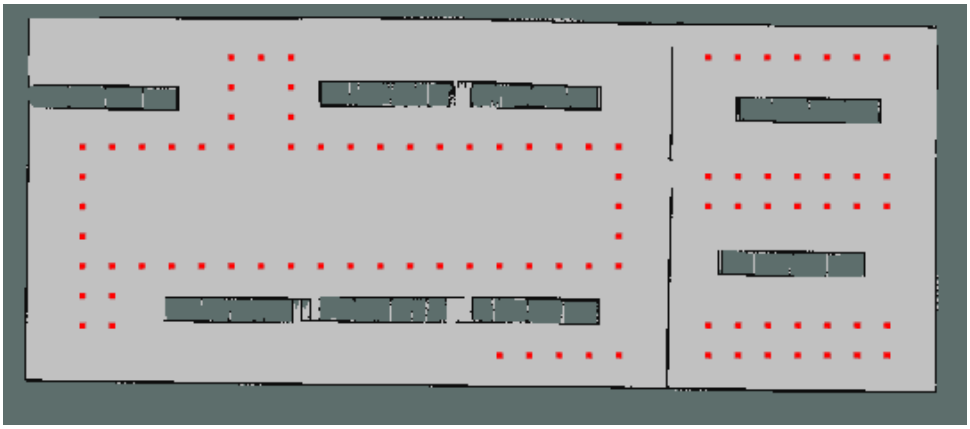


Σχήμα 5.13: Δειγματοληψία με διάφορα βήματα

Αλγόριθμος 5.20 Uniform Sampling

```
1: function uniformSampling(brush, sensor_range, resolution, robot_radius)
2:     nodes = [], step_list = []
3:     min_range = int(min(sensor_range)/resolution)
4:     max_range = int(max(sensor_range)/resolution)
5:     step = int(1.5 * robot_radius/resolution)
6:     while step <= max_range do
7:         temp_nodes = []
8:         step_list.append(step)
9:         indexes = np.where((brush > step) & (brush <= 2 * step) & (brush <=
   max_range))
10:        indexes = zip(*indexes)
11:        for each (x, y) in indexes do
12:            if not x % step and not y % step then
13:                temp_nodes.append((x, y))
14:            end if
15:        end for
16:        step* = 2
17:        nodes.append(temp_nodes)
18:    end while
19:    obstacles = numpy.zeros(brush.shape), final_nodes = [], i = len(nodes) − 1
20:    while i >= 0 do
21:        temp_nodes = nodes[i]
22:                    # Check if new nodes override previous ones and delete them
23:        for point in temp_nodes do
24:            indexes = circularRayCastCoverage(point, ogm, max_range,
          360, 0, 0, True)
25:            if not len(indexes) then
26:                continue
27:            else if i == len(nodes) − 1 then
28:                obstacles[zip(*indexes)] = 1
29:                final_nodes.append(point)
30:            else
31:                temp = obstacles[zip(*indexes)]
32:                p = len(np.where(temp > 0)[0])/len(indexes)
33:                if p < 0.9 then
34:                    final_nodes.append((x, y))
35:                    obstacles[zip(*indexes)] = 1
36:                end if
37:            end if
38:        end for
39:        i− = 1
40:    end while
41:    return final_nodes, step_list
```

Αντίθετα, εαν γίνει χρήση σταθερού βήματος δειγματοληψίας είναι πολύ πιθανό να προκύψουν περιοχές χωρίς σημεία πράγμα που σημαίνει αποτυχία πλήρους κάλυψης του χώρου, όπως φαίνεται και στο σχήμα 5.14.



Σχήμα 5.14: Παράδειγμα Αποτυχίας Εύρεσης Σημείων

5.4.2 Υπολογισμός βέλτιστης αλληλουχίας επίσκεψης των σημείων

Στην συνέχεια, τα σημεία αυτά χωρίζονται στα διάφορα δωμάτια και υπολογίζεται η βέλτιστη αλληλουχία των σημείων ανά δωμάτιο. Η διαδικασία αυτή έχει τα παρακάτω βήματα.

- Διαχωρισμός σημείων σε δωμάτια
- Hill Climbing σε συνδιασμό με τον αλγόριθμο κοντινότερου γείτονα
- Προσθήκη των πορτών εισόδου και εξόδου του δωματίου στην αλληλουχία
- RRHC με τις πόρτες σταθερές στις άκρες της αλληλουχίας

Ο διαχωρισμός των κόμβων σε δωμάτια πραγματοποιείται χωρίζοντας το OGM σε ασύνδετους χώρους, δηλαδή προσθέτοντας τοίχους εκεί που υπάρχουν οι πόρτες και χρησιμοποιώντας μια παραλλαγή του αλγορίθμου brushfire, την *pointBrushfire*.

Οι πόρτες του χώρου είναι ήδη γνωστές ενώ τα κοντινότερα εμπόδια τους, δηλαδή οι δύο τοίχοι υπολογίζονται από τον αλγόριθμο *closestObstacleBrushfire* 5.4. Στη συνέχεια, υπολογίζεται η νοητή ευθεία του τοίχου, δηλαδή η ευθεία της κάθε πόρτας στο OGM και προστίθεται τοίχος, δηλαδή τα σημεία αυτά τίθονται ίσα με 100 στο νέο OGM. Η συνάρτηση αυτή παρουσιάζεται στο 5.21.

Έπειτα, με την χρήση brushfire με αρχή τους κόμβους κάθε δωματίου από την προηγούμενη διαδικασία εξαγωγής χρήσιμων σημείων από το GVD 5.1.3 και το διαχωρισμένο σε δωμάτια OGM χωρίζονται οι κόμβοι προσπέλασης σε δωμάτια.

Στη συνέχεια, για κάθε δωμάτιο υπολογίζεται μια βέλτιστη αλληλουχία. Τα σημεία ταξινομούνται σύμφωνα με τις συντεταγμένες τους (x, y) και στη συνέχεια χρησιμοποιείται μια παραλλαγή του HC που αξιοποιεί το γεγονός ότι η δειγματοληψία των σημείων έχει πραγματοποιηθεί με συγκεκριμένα βήματα. Αυτό σημαίνει ότι τα σημεία που είναι πολύ κοντά μεταξύ τους, θα έχουν απόσταση ίση με ένα

Αλγόριθμος 5.21 Door Closure

```
1: function doorClosure(doors, ogm)
2:     filled_ogm = numpy.copy(ogm)
3:     for door in doors do
4:         ob = closestObstacleBrushfire(door, ogm)
5:         Find obstacle points' line
6:         Set points of line equal to 100 in filled_ogm
7:     end for
8:     return filled_ogm
```

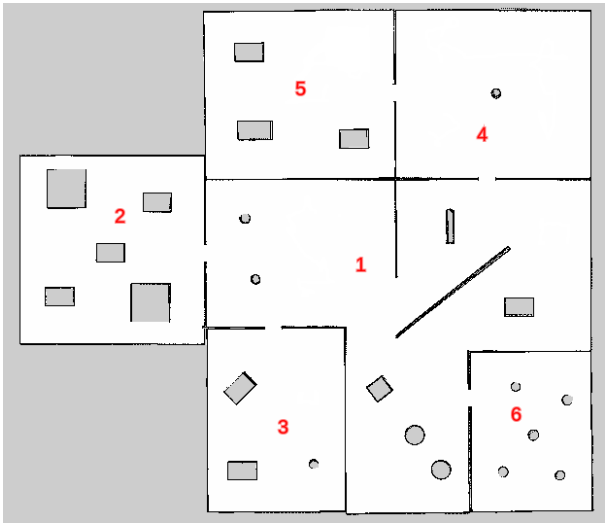
από τα διάφορα βήματα, τα οποία έχουν αποθηκευτεί στο *step_list* στον αλγόριθμο 5.20. Έτσι, υπολογίζεται μια αλληλουχία με βάση εαν τα σειριακά σημεία έχουν μεταξύ τους απόσταση ίση με τα βήματα της δειγματοληψίας. Εαν δεν υπάρχει τέτοιο σημείο, τότε προστίθεται στην σειρά το κοντινότερο δυνατό, χρησιμοποιείται, δηλαδή, και η προσέγγιση του κοντινότερο γείτονα. Ο αλγόριθμος που περιγράφει αυτή την διαδικασία παρουσιάζεται στο 5.22.

Αλγόριθμος 5.22 Step Hill Climb

```
1: function stepHillClimb(distances, epochs, steps)
2:     length = distances.shape[0]
3:     best = range(length)
4:     Set best_score equal to the length of route best
5:     iter = 0
6:     for i in range(length − 2) do
7:         if iter >= epochs then
8:             break
9:         end if
10:        minimum = distances[best[i]][best[i + 1]]
11:        if minimum <= min(steps) then
12:            continue
13:        else
14:            Find point index closest to point i
15:            best[i + 1], best[index] = best[index], best[i + 1]
16:            Set best_score equal to the length of route best
17:        end if
18:    end for
19:    return best, best_score, iter
```

Στη συνέχεια, προστίθενται και οι πόρτες εισόδου και εξόδου στην αρχή και στο τέλος αντίστοιχα της υπολογισμένης αλληλουχίας. Μάλιστα, για την περαιτέρω βελτίωση της, πραγματοποιείται μια εσωτερική κυκλική περιστροφή των σημείων, έτσι ώστε το δεύτερο σημείο να είναι το κοντινότερο από την πόρτα εισόδου που είναι το πρώτο σημείο. Αυτή είναι μια λογική μεταβολλή η οποία βελτιώνει την συνολική απόσταση του μονοπατιού χωρίς την χρήση σύνθετων αλγορίθμων.

Τέλος, εφαρμόζεται ο RRHC αλγόριθμος με σταθερά άκρα τις πόρτες εισόδου



Σχήμα 5.15: Περιβάλλον ελέγχου διαδικασίας υπολογισμού μονοπατιού

και εξόδου στο δωμάτιο. Μετά από πειραματισμούς στο πλήθος των επαναλήψεων του RRHC με τιμές 1000, 2000 και 5000 φορές επί το πλήθος των σημείων του δωματίου, προέκυψε ότι ιδανικότερος αριθμός είναι οι 2000 επί πλήθος των σημείων του δωματίου. Είναι ένας αριθμός που φέρει πολύ καλά αποτελέσματα χωρίς να απαιτεί σημαντικό χρόνο υπολογισμών. Ο RRHC αναλύεται στο 5.12.

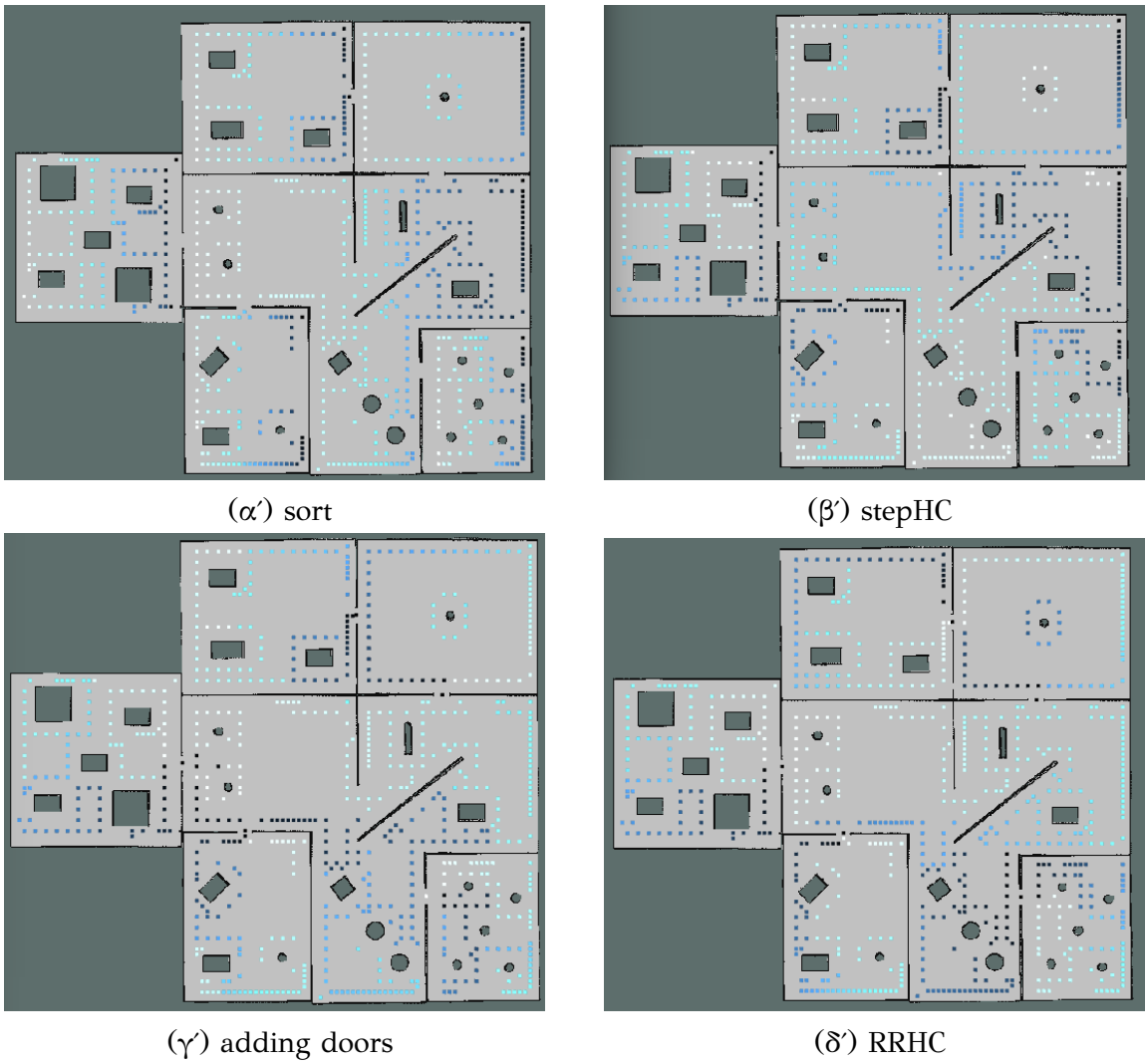
Ένα παράδειγμα της προόδου του μήκους της αλληλουχίας των διάφορων δωματίων ενός χάρτη σε κάθε ένα από τα τέσσερα στάδια επεξεργασίας παρουσιάζεται στον επόμενο πίνακα 5.1 και στα σχήματα 5.16. Η μελέτη πραγματοποιήθηκε στο το OGM του σχήματος 5.15, πάνω στο οποίο εμφανίζεται και με κόκκινους αριθμούς η σειρά των δωματίων. Η σειρά επίσκεψης των σημείων κάθε δωματίου οπτικοποιείται με την εναλλαγή χρωμάτων ανά δωμάτιο. Το όχημα θα εκκινεί από τα σκούρα σημεία και θα κατευθύνεται στα πιο ανοικτόχρωμα, καταλήγοντας στα άσπρα.

Πίνακας 5.1: Βελτίωση μήκους διαδρομής ανά στάδιο επεξεργασίας

Stages	room 1	room 2	room 3	room 4	room 5	room 6
sort	14719	4711	5398	4699	4443	2633
stepHC	3951	1494	1049	945	1275	1024
adding doors	4554	1524	1060	1212	1283	1056
RRHC	3761	1344	935	1032	1131	944

5.4.3 Υπολογισμός βέλτιστου προσανατολισμού σε κάθε σημείο

Αφότου βρεθεί το σύνολο των σημείων για την πλήρη κάλυψη του χώρου, χρειάζεται να υπολογιστεί και η βέλτιστη γωνία του ρομπότ σε κάθε θέση. Άλλωστε, η ακριβής θέση του οχήματος στον διδιάστατο χώρο αποτελείται από τρεις μεταβλητές, δύο για την θέση και μία για τον προσανατολισμό. Τα κριτήρια επιλογής του προσανατολισμού είναι δύο. Το πρώτο είναι οι αισθητήρες να καλύπτουν όσο το



(α΄) sort (β΄) stepHC

(γ΄) adding doors (δ΄) RRHC

Σχήμα 5.16: Στάδια βελτίωσης μονοπατιού

δυνατόν περισσότερη επιφάνεια εμποδίων και το δεύτερο η γωνιακή μετατόπιση ανάμεσα στον τρέχοντα προσανατολισμό και το επόμενο σημείο να είναι η ελάχιστη δυνατή. Αυτά μας εξασφαλίζουν όσο το δυνατόν μεγαλύτερη κάλυψη του χώρου και ελαχιστοποίηση της χρονικής διάρκειας της διαδικασίας αντίστοιχα. Για τον λόγο αυτό δημιουργήθηκε μια συνδιαστική συνάρτηση των δύο αυτών παραμέτρων με την μέθοδο motor schema, όπου λαμβάνονται υπόψη και οι δύο παράμετροι με κάποιο βάρος. Η μέθοδος υπολογισμού των βαρών αυτών παρουσιάζεται στο τέλος της υποενότητας.

Πρώτα αναλύεται η διαδικασία επιλογής κάθε γωνίας στροφής. Για κάθε σημείο στόχο εξετάζονται όλες οι γωνίες μία προς μία με βήμα 10 μοιρών και κρατείται αυτή που δίνει την βέλτιστη τιμή της συνάρτησης του motor schema, η οποία είναι η παρακάτω.

$$eval = obstacle\_weight * \frac{len(covered\_obstacles)}{len(obstacles)} + rotation\_weight * \frac{180 - next\_rotation}{180}$$

Ο πρώτος όρος περιέχει το πλήθος των εμποδίων που καλύπτονται από την εν λόγω γωνία και ο δεύτερος περιέχει τη γωνιακή στροφή προς το επόμενο σημείο στόχο. Και οι δύο είναι κανονικοποιημένοι, έτσι ώστε οι τιμές να περιορίζονται στο εύρος $[0, 1]$ και το 1 να αντιστοιχεί στο βέλτιστο επιμέρους αποτέλεσμα.

Σε κάθε σημείο, όμως, δεν αποθηκεύεται μόνο μια γωνία αλλά μπορούν και περισσότερες μέχρι να έχει συνολικά καλυφθεί το περισσότερο τμήμα των γύρω εμποδίων, δηλαδή ένα ποσοστό μεγαλύτερο του 50%. Μετά από ορισμένες δοκιμές βρέθηκε ότι το όριο του 60% προσφέρει καλή συνολική κάλυψη του χώρου. Μεγαλύτερα όρια, όπως τα 75% και 80%, οδηγούν σε αποθήκευση πολλών περιττών θέσεων - στόχων χωρίς βελτίωση στην συνολική τελική κάλυψη του χώρου, ενώ το μικρότερο όριο 50% οδηγεί σε μείωση της κάλυψης. Συνεπώς, για κάθε σημείο αποθηκεύονται οι καλύτερες γωνίες εως ότου έχει καλυφθεί τουλάχιστον το 60% των κοντινών εμποδίων. Το ποσοστό είναι φαινομενικά μικρό, ωστόσο εξαιτίας της κοντινής απόστασης μεταξύ των σημείων στόχων οδηγεί σε θεμιτά αποτελέσματα.

Αυτό το κατώφλι χρησιμοποιήθηκε για τις περιπτώσεις που υπάρχουν περισσότερα του ενός εμπόδια στην γύρω περιοχή και το όχημα φέρει μικρό αριθμό αισθητήρων, για παράδειγμα έναν. Τότε εαν αποθηκευτεί ένας μόνο στόχος, είναι πολύ πιθανό να μην σαρωθεί σημαντικό τμήμα των εμποδίων. Με την πολλαπλή αποθήκευση γωνιών σε κάθε σημείο εξασφαλίζεται η καλύτερη δυνατή κάλυψη της κάθε περιοχής. Ο αλγόριθμος αυτός παρουσιάζεται αναλυτικά στο 5.23.

Τέλος, η διαδικασία αυτή υλοποιήθηκε σε έναν συγκεκριμένο χάρτη με πέντε διαφορετικούς συνδιασμούς βαρών, ώστε να μελετηθεί η επιρροή τους και να επέλθει η τελική μέθοδος επιλογής τους στην προηγούμενη συνάρτηση. Μελετήθηκαν η εκτίμηση της κάλυψης του χώρου, η πραγματική κάλυψη του και ο χρόνος της και τα αποτελέσματα παρουσιάζονται στους επόμενους δύο πίνακες 5.2, 5.3 για διαφορετικούς αισθητήρες. Στην πρώτη περίπτωση οι κεραίες είναι τοποθετημένες στα πλάγια του οχήματος, ενώ στην δεύτερη στο μπροστά και το πίσω μέρος του.

Obstacle W.	Rotation W.	Εκτίμηση	Πραγματική Κάλυψη	Χρόνος
4	1	85.49 %	87.43 %	71.6 min
2	1	85.80 %	85.95 %	71.2 min
1	1	85.04 %	86.78 %	66 min
1	2	83.16 %	86.20 %	59.5 min
1	4	83.16 %	86.74 %	76.6 min

Πίνακας 5.2: Εκτίμηση Βαρών Motor Schema με Κεραίες στα Πλάγια

Obstacle W.	Rotation W.	Εκτίμηση	Πραγματική Κάλυψη	Χρόνος
4	1	85.95 %	83.03 %	76 min
2	1	85.34 %	83.61 %	73.7 min
1	1	84.85 %	80.58 %	66.4 min
1	2	83.29 %	84.38 %	98.18 min
1	4	82.81 %	85.41 %	122.1 min

Πίνακας 5.3: Εκτίμηση Βαρών Motor Schema με Κεραίες Μπροστά και Πίσω

Αλγόριθμος 5.23 Find Best Yaw

```
1: function findBestYaw(node, next_node, obstacle_weight, rotation_weight, sensor_range, resolution)
2:     yaw = []
3:     yaw_between_nodes = math.degrees(math.atan2(next_node[1] − node[1], next_node[0] − node[0]))
4:     steps = int(max(sensor_range)/resolution)
5:     Fill obstacles with indexes of nearby obstacles at range steps
6:     found = 0
7:     while found < 0.6 * len(obstacles) and len(obstacles) do
8:         best_evaluation = 0, best_yaw = 0, best_found = 0, best_covered_obstacles = []
9:         for each angle angle in [−180, 180] do
10:            Check covered nearby obstacles covered_obstacles
11:            next_rotation = abs(angle − yaw_between_nodes)
12:            Evaluate candidate angle
13:            if angle is better than current best then
14:                best_evaluation = angle's evaluation
15:                best_yaw = angle
16:                best_found = len(covered_obstacles)
17:                best_covered_obstacles = covered_obstacles
18:            end if
19:        end for
20:        found+ = best_found
21:        Discard best_covered_obstacles from obstacles
22:    end while
23:    return best_yaw
```

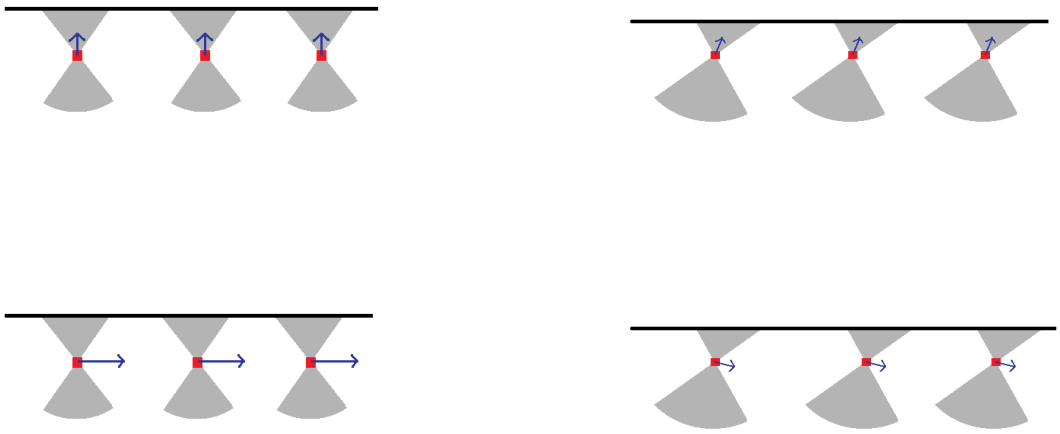
Τα συμπεράσματα που προκύπτουν είναι ότι ο χρόνος ελαχιστοποιείται όταν το βάρος της γωνιακής κίνησης είναι λίγο μεγαλύτερο από το βάρος της κάλυψης, αλλά όταν γίνει πολύ μεγαλύτερο, με αναλογία πάνω από δύο προς ένα, τότε δημιουργούνται πολλά περιττά σημεία προσπέλασης αυξάνοντας ριζικά τον χρόνο. Αντίθετα, η χρήση μεγαλύτερου βάρους στο ποσοστό κάλυψης δεν βελτιώνει την συνολική κάλυψη του χώρου, αλλά αυξάνει τον συνολικό χρόνο της διαδικασίας. Συνεπώς, τα βάρη και αυτά με την σειρά τους εξαρτώνται άμεσα από την θέση των κεραιών και, μάλιστα, όσο πιο κεντρικά είναι αυτές, τόσο μεγαλύτερο βάρος πρέπει να δίνεται στο ποσοστό κάλυψης, ενώ όσο πιο πλάγια είναι τόσο πρέπει να αυξάνεται το βάρος της περιστροφής. Στην προκειμένη περίπτωση κέντρο είναι η διάμετρος του οχήματος που διέρχεται από το μπροστινό του μέρος. Οι συντελεστές θα κυμαίνονται στο πεδίο $[1, 2]$, όπου και βελτιστοποιείται η διαδικασία. Οι ακριβείς συναρτήσεις που χρησιμοποιήθηκαν παρουσιάζονται στην συνέχεια.

$$obstacle\_weight = 1 + \frac{90 - theta}{90}$$

$$rotation\_weight = 2 - \frac{90 - theta}{90}$$

Όπου $theta = mean(sensor\_direction)$. Στόχος είναι, δηλαδή, η κεραία να κοιτάει όσο πιο κεντρικά γίνεται το κάθε εμπόδιο, εξασφαλίζοντας την καλύτερη δυνατή κάλυψη. Σημειώνεται ότι αρχικά η γωνία κατεύθυνσης κάθε αισθητήρα μετασχηματίζεται στο πεδίο $[0, 90]$ για την ευκολότερη ανάλυση των δεδομένων.

Τα συμπεράσματα αυτά μπορούν να γίνουν κατανοητά και από τις επόμενες εικόνες 5.17. Στις δύο επάνω εικόνες οι κεραίες είναι τοποθετημένες στο μπροστά και το πίσω μέρος του οχήματος. Η εμπρός κατεύθυνση του οχήματος φαίνεται από το μπλε βέλος. Είναι προτιμότερο το όχημα να μην κοιτάει απευθείας τα εμπόδια, αλλά να έχει μια ελαφριά κλίση προς τον επόμενο στόχο, καλύπτοντας όμως σημαντικό τμήμα του τοίχου. Έτσι, μετά τη σάρωση θα χρειαστεί να πραγματοποιήσει μικρότερη περιστροφική κίνηση για να φτάσει στον επόμενο στόχο. Αντίθετα, στις δύο κάτω εικόνες οι κεραίες είναι τοποθετημένες στα πλάγια μέρη του οχήματος. Στην περίπτωση αυτή είναι προτιμότερο το όχημα να κοιτάει ακριβώς προς τον επόμενο στόχο. Έτσι, καλύπτει σταθερά τα εμπόδια κατά την μετακίνηση του, χωρίς να αποκλίνει από το μονοπάτι πλοήγησης του. Αυτή μάλιστα είναι η καλύτερη δυνατή τοποθέτηση των κεραιών, διότι το όχημα κινείται συνεχώς πάνω στην ευθεία μεταξύ των διαδοχικών στόχων. Με τη μέθοδο αυτή, λοιπόν, αποφεύγονται οι περιττές περιστροφικές κινήσεις και εξοικονομείται σημαντικός χρόνος και ενέργεια.



Σχήμα 5.17: Παραδείγματα διαφορετικών προσανατολισμών

5.4.4 Προσομοίωση κάλυψης του χώρου και διαγραφή περιττών σημείων

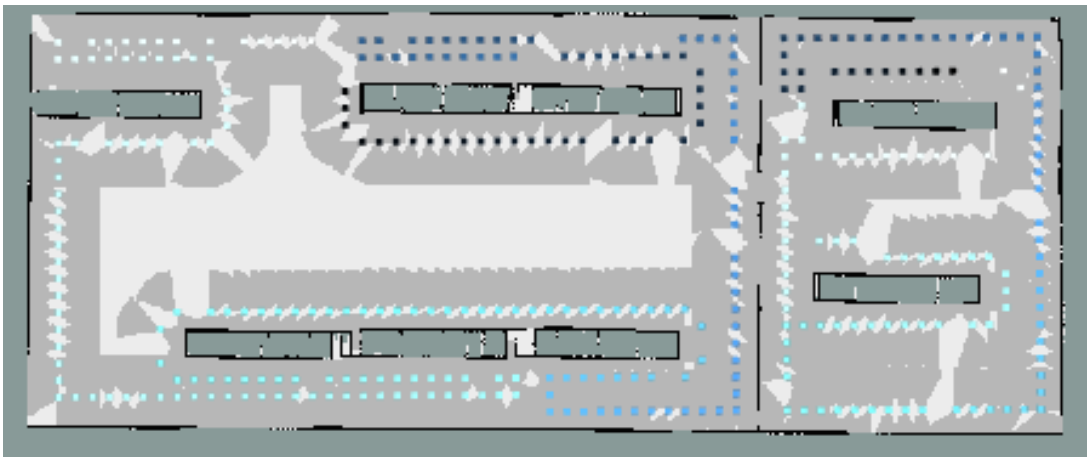
Τελευταίο βήμα της δημιουργίας της πρώτης στρατηγικής πλοήγησης είναι η εκ των προτέρων εκτίμηση της κάλυψης του χώρου και η διαγραφή των στόχων που είναι περιττοί στην όλη διαδικασία. Η εκτίμηση της κάλυψης πραγματοποιείται προσομοιώνοντας τη μέτρηση των αισθητήρων του οχήματος για κάθε στόχο της ακολουθίας. Με τον τρόπο αυτό αποθηκεύονται σε ένα OGM τα σημεία τα οποία

Αλγόριθμος 5.24 Find Weights

```
1: function findWeights(sensor_number, sensor_direction, sensor_fov)
2:     angles = []
3:     for s in range(sensor_number) do
4:         theta = sensor_direction[s]
5:         Transform theta to range [0, 90]
6:         angles.append((90 − theta)/90)
7:     end for
8:     obstacle_weight = 1 + mean(angles)
9:     rotation_weight = 2 − mean(angles)
10:    return obstacle_weight, rotation_weight
```

πρόκειται να καλυφθούν. Αυτή είναι η απλή περίπτωση με την οποία μπορεί να υπολογιστεί και το ποσοστό κάλυψης των εμποδίων του χώρου, όπως και έγινε στα πειράματα των πινάκων 5.2 και 5.3. Σημειώνεται ότι η κάλυψη αυτή είναι μια εκτίμηση, καθώς η πλοήγηση του οχήματος δεν περιλαμβάνεται στο σημείο αυτό με αποτέλεσμα να μην λαμβάνονται υπόψη οι ενδιάμεσες κινήσης μετάβασης από το κάθε σημείο στο επόμενο. Μάλιστα, η τελική πραγματική κάλυψη θα είναι, θεωρητικά, μεγαλύτερη της εκτίμησης.

Ένα παράδειγμα εκτίμησης κάλυψης παρουσιάζεται στο σχήμα 5.18, όπου φαίνονται με διάφορες αποχρώσεις του μπλε τους στόχους και με σκούρο γκρι την περιοχή που εκτιμάται ότι θα καλύψει το όχημα από όλα τα σημεία. Επίσης, οπτικοποιείται και η σειρά προσπέλασης, η οποία εκκινεί για κάθε δωμάτιο από τα πιο σκούρα σημεία και κατευθύνεται στα πιο ανοικτόχρωμα, καταλήγοντας στα λευκά.



Σχήμα 5.18: Παράδειγμα Πεδίου Εκτίμησης Κάλυψης

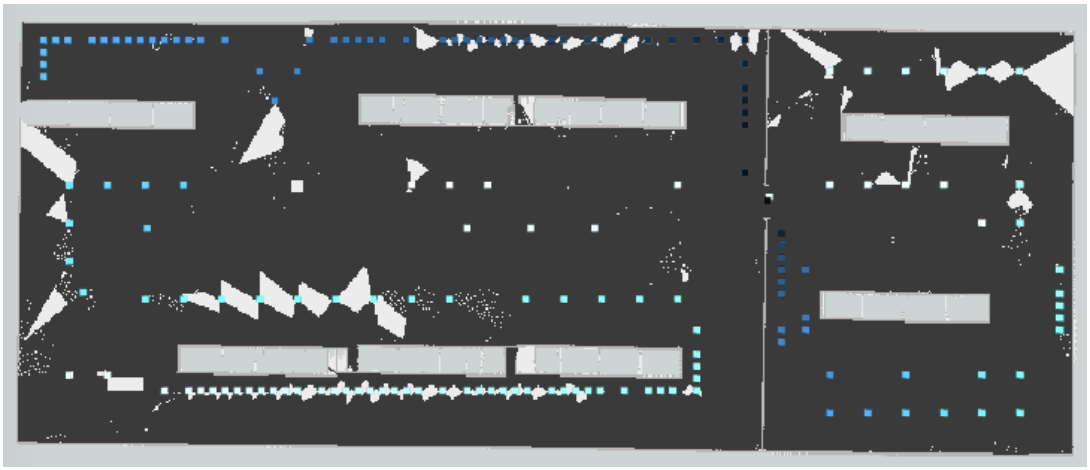
Αντίθετα, η διαγραφή σημείων προϋποθέτει τον έλεγχο κάθε φορά των εμποδίων που πρόκειται να καλυφθούν. Εαν το μεγαλύτερο ποσοστό τους είναι ήδη καλυμμένο, τότε το σημείο αυτό είναι περιττό για την μελέτη αυτή. Στόχος είναι να καλυφθεί όσο το δυνατό μεγαλύτερη επιφάνεια εμποδίων, δηλαδή ένα ποσοστό που πλησιάζει το 100%. Μετά από ορισμένες δοκιμές βρέθηκε ότι το όριο του 85% προσφέρει καλή συνολική κάλυψη του χώρου. Η χρήση μεγαλύτερων ορίων,

όπως τα 90% και 95%, οδηγεί σε αποθήκευση πολλών περιττών θέσεων - στόχων χωρίς βελτίωση στην συνολική τελική κάλυψη του χώρου, ενώ το μικρότερο όριο 80% οδηγεί σε μείωση της κάλυψης. Συνεπώς, για κάθε σημείο υπολογίζονται τα εμπόδια που πρόκειται να σαρώσουν οι αισθητήρες και εφόσον περισσότερο από το 85% των σημείων είναι ήδη καλυμμένα, τότε ο στόχος αυτός διαγράφεται από την ακολουθία. Τονίζεται ότι τα σημεία εισόδου και εξόδου κάθε χώρου, δηλαδή οι κόμβοι πόρτες, είναι απαραίτητοι για την ομάλη μετάβαση του οχήματος στον χώρο και, επομένως, δεν επιδέχονται διαγραφής.

Ο 5.26 υπολογίζει τα σημεία που σαρώνουν οι αισθητήρες για μια δεδομένη πόζα του οχήματος και στη συνέχεια υπολογίζει πόσα από αυτά έχουν ήδη σαρωθεί. Εαν το ποσοστό είναι μεγαλύτερο από το *threshold* τότε η πόζα αυτή απορρίπτεται, ειδάλλως προστίθεται κανονικά στην τελική ακολουθία.

Ο 5.25 αποτελεί τον συγκεντρωτικό αλγόριθμο εκτέλεσης της εκ των προτέρων κάλυψης του χώρου. Ο αλγόριθμος δέχεται ως είσοδο και μια λογική μεταβλητή, την *eliminate*, η οποία καθορίζει εαν θα πραγματοποιηθεί εξάλειψη των περιττών σημείων. Αν αυτή είναι Ψευδής τότε η διαδικασία αποτελεί την απλή εκτίμηση κάλυψης του χώρου, το τελικό OGM της οποίας χρησιμοποιείται για να υπολογιστεί το ποσοστό κάλυψης. Αντίθετα, εαν είναι Αληθής, τότε στόχος είναι η μείωση του μεγέθους της ακολουθίας. Επιπλέον, η εντολή ελέγχου στην όγδοη σειρά είναι αυτή που καθορίζει ποια από τις δύο περιπτώσεις θα κληθεί. Στην εντολή αυτή φαίνεται και το γεγονός ότι ο πρώτος και ο τελευταίος στόχος κάθε δωματίου κρατείται στην τελική ακολουθία για τους λόγους που σχολιάστηκαν νωρίτερα. Τέλος, η διαδικασία πραγματοποιείται για κάθε δωμάτιο και τελικά επιστρέφεται η τελική ακολουθία του πρώτου πρότυπου πλοήγησης.

Στο σχήμα 5.18 παρουσιάστηκε η περίπτωση η απλή περίπτωση εκτίμησης κάλυψης. Στο 5.19 παρουσιάζεται η δεύτερη περίπτωση, όπου όσοι στόχοι είναι περιττοί έχουν διαγραφεί. Επίσης, η δειγματοληψία σ αυτό το παράδειγμα έχει πολλά διαφορετικά βήματα, ενώ οι αισθητήρες έχουν μεγαλύτερη ακτίνα λήψης. Τέλος, η μαύρη περιοχή αποτελεί την εκτίμηση της περιοχής που θα σαρωθεί από τους αισθητήρες.



Σχήμα 5.19: Παράδειγμα Πεδίου Εκτίμησης Κάλυψης με Διαγραφή Περιττών Σημείων

Αλγόριθμος 5.25 A Priori Coverage

```
1: function aPrioriCoverage(ogm, brushfire, nodes, eliminate, resolution, sensor_range, sensor_fov, sensor_direction)
2:     coverage = numpy.zeros(ogm.shape)
3:     new_nodes = []
4:     for room in nodes do
5:         new_room = []
6:         for i in range(len(room)) do
7:             x, y = room[i]['position'], yaw = room[i]['yaw']
8:             if not i or i == len(room) − 1 or not eliminate then
9:                 coverage, previous_robot_pose = updateCover(
                   room[i − 1]['position'], ogm, coverage, resolution, sensor_range,
                   sensor_fov, sensor_direction)
10:                updated =True
11:            else
12:                updated, coverage = checkAndUpdateCover(brushire, 0.85, ogm, coverage,
                   resolution, sensor_range, sensor_fov, sensor_direction)
13:            end if
14:            if updated then
15:                new_room.append(room[i])
16:            end if
17:        end for
18:        new_nodes.append(new_room)
19:        i + +
20:    end for
21:    return new_nodes
```

Αλγόριθμος 5.26 Check And Update Cover

```
1: function checkAndUpdateCover(brushfire, threshold, ogm, coverage, resolution,
   sensor_range, sensor_fov, sensor_direction)
2:     Read robot's pose xx, yy, th
3:     indexes = []
4:     for each sensor s do
5:         cover_length = int(sensor_range[s]/resolution)
6:         temp = circularRayCastCoverage((xx, yy), ogm, cover_length, sensor_fov[s],
             th, sensor_direction[s])
7:         indexes.extend(temp)
8:     end for
9:     Compute percentage perc of already covered points of indexes
10:    updated =False
11:    if perc < threshold then
12:        updated =True
13:        for x, y in indexes do
14:            coverage[x, y] = 100
15:        end for
16:    end if
17:    return updated, coverage
```

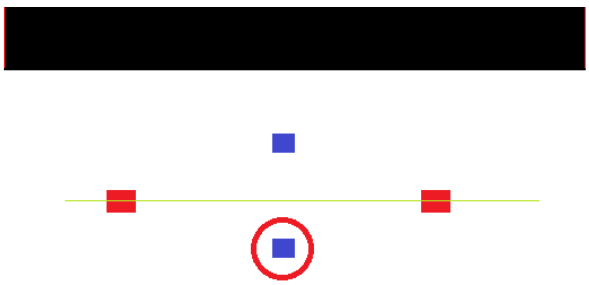
5.4.5 Δημιουργία Αλληλουχίας Κόμβων Ζιγκ-Ζαγκ

Το τελευταίο μέρος της υλοποίησης του συστήματος περιέχει την δημιουργία της δεύτερης στρατηγικής πλοήγησης στον χώρο. Αυτή είναι η πλοήγηση με κινήσεις σε μορφή ζιγκ ζαγκ. Η κίνηση του οχήματος χρησιμοποιώντας αυτή την στρατηγική έχει δύο πλεονεκτήματα. Πρώτον, κάθε εμπόδιο θα σαρώνεται περισσότερες φορές και, δεύτερον, οι σαρώσεις θα πραγματοποιούνται με διαφορετικές γωνίες μεταξύ αισθήρα και εμποδίου, προσφέροντας πιο αναλυτική και ομοιόμορφη κάλυψη. Στην ιδανική περίπτωση στόχος είναι να σαρωθεί ένα εμπόδιο τόσο από το θετικό μισό τόξο του αισθητήρα, όσο και από το αρνητικό 5.10. Βέβαια, όσες περισσότερες φορές και υπό διαφορετικές γωνίες σαρώσουν οι αισθητήρες ένα σημείο, τόσο πιο βέβαιο θα είναι το αποτέλεσμα. Το σημαντικό μειονέκτημα, όμως, της περίπτωσης αυτής είναι η αύξηση του συνόλου των στόχων που έχει να προσεγγίσει το όχημα και, επομένως, η αύξηση του συνολικού χρόνου κάλυψης του χώρου.

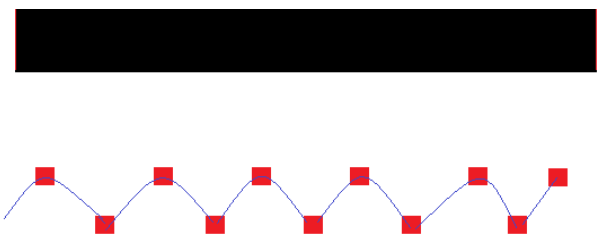
Ένας πολύ απλός και ταυτόχρονα αποτελεσματικός τρόπος παραγωγής μιας τέτοιας πορείας είναι η προσθήκη σημείων στόχων στην ήδη υπάρχουσα βέλτιστη ακολουθία, με στόχο την δημιουργία ζιγκ ζαγκ κινήσεων. Η προσθήκη τέτοιων στόχων μεταξύ των συνεχώμενων γειτονικών σημείων με μια μικρή απόκλιση από την μεταξύ τους ευθεία θα επιφέρει το επιθυμητό αποτέλεσμα. Ο αλγόριθμος αυτός παρουσιάζεται στο 5.27 και αναλύεται στη συνέχεια. Η ακολουθία κάθε δωματίου εξετάζεται χωριστά. Τα ήδη υπολογισμένα σημεία διατηρούνται ανέπαφα και ενδιάμεσα τους προστίθενται νέα, όπου κρίνεται απαραίτητο. Η διαδικασία εκμεταλλεύεται το γεγονός ότι η δειγματοληψία που έχει υλοποιηθεί έχει σταθερά βήματα, συνεπώς είναι εύκολα διακριτό εαν δύο σημεία βρίσκονται πολύ κοντά και

χρίζουν προσθήκης νέων ενδιάμεσων στόχων. Θα έχουν μια από τις δύο μεταβλητές θέσης ίσες, πράγμα δεδομένο λόγω της ομοιόμορφης δειγματοληψίας σε κάθε βήμα. Εαν ισχύει αυτό και τα δύο σημεία είναι αρκετά κοντά, αλλά δεν ταυτίζονται, τότε προστίθεται ένας ακόμη στόχος. Υπολογίζεται η μέση των δύο σημείων και η απόσταση του νέου σημείου από την ευθεία τους η οποία είναι η μισή από την μεταξύ τους απόσταση, ώστε η τροχιά του οχήματος να είναι όσο πιο ομαλή γίνεται. Από τις δύο υποψήφιες θέσεις επιλέγεται αυτή με μεγαλύτερη τιμή brushfire, καθώς θα απέχει μεγαλύτερη απόσταση από τα εμπόδια. Έπειτα, καθώς κάθε στόχος περιέχει και τον προσανατολισμό του οχήματος, τίθεται ως τιμή στροφής του οχήματος η γωνία της ευθείας μεταξύ του τρέχοντος σημείου και του νέου, κάτι που δεν θα επηρεάζει, δηλαδή, τη συνολική κίνηση του οχήματος. Τέλος προκειμένου, να προστεθεί στην αλληλουχία το νέο σημείο πρέπει να ανήκει στο εν λόγω δωμάτιο, γι αυτό και πραγματοποιείται ο αντίστοιχος έλεγχος.

Η διαδικασία οπτικοποιείται στο 5.20. Τα κόκκινα σημεία έχουν βρεθεί μέσω της δειγματοληψίας και είναι πολύ κοντά, συνεπώς μπορεί να προστεθεί ένας ακόμη στόχος ενδιάμεσα τους για να προκληθεί η ζιγκ ζαγκ κίνηση. Η ευθεία των σημείων σημειώνεται με πράσινο. Τα μπλε σημεία είναι τα δύο υποψήφια να προστεθούν στην ακολουθία, τα οποία βρίσκονται πάνω στην μεσοκάθετο του ευθύγραμμου τμήματος των δύο σημείων. Επιλέγεται, τελικά, αυτό που απέχει την μεγαλύτερη απόσταση από τα γύρω εμπόδια, δηλαδή το κάτω σημείο, το οποίο και έχει κυκλωθεί. Ένα παράδειγμα ζιγκ ζαγκ κίνησης μπροστά από ένα εμπόδιο είναι το 5.21.



Σχήμα 5.20: Παράδειγμα Επιλογής Ζιγκ Ζαγκ Σημείου



Σχήμα 5.21: Παράδειγμα Κίνησης Ζιγκ Ζαγκ παράλληλα με ένα εμπόδιο

Αλγόριθμος 5.27 Add Zig Zag Nodes

```
1: function addZigZagNodes(sequence, brushfire, ogm, sensor_range, resolution)
2:     zig_zag_sequence = []
3:     for each room in sequence do
4:         temp_sequence = []
5:         for i in range(len(room)) do
6:             if not i then
7:                 temp_sequence.append(room[i])
8:                 continue
9:             end if
10:            node = room[i]['position'], previous_node = room[i − 1]['position']
11:            x_final, y_final = 0, 0
12:            if previous_node[0] == node[0] and abs(previous_node[1] − node[1]) and
    abs(previous_node[1] − node[1]) <= max(sensor_range)/resolution then
13:                dif = abs(previous_node[1] − node[1])
14:                x1 = int(node[0] − dif/2), x2 = int(node[0] + dif/2)
15:                y = int(min([previous_node[1], node[1]]) + dif/2)
16:                if brushfire[x1, y] >= brushfire[x2, y] then
17:                    x_final = x1, y_final = y, yaw = math.degrees(math.atan2(y −
    node[1], x1 − node[0]))
18:                else
19:                    x_final = x2, y_final = y, yaw = math.degrees(math.atan2(y −
    node[1], x2 − node[0]))
20:                end if
21:            else if previous_node[1] == node[1] and abs(previous_node[0] − node[0])
    and abs(previous_node[0] − node[0]) <= max(sensor_range)/resolution then
22:                dif = abs(previous_node[0] − node[0])
23:                y1 = int(node[1] − dif/2), y2 = int(node[1] + dif/2)
24:                y = int(min([previous_node[0], node[0]]) + dif/2)
25:                if brushfire[x, y1] >= brushfire[x, y2] then
26:                    x_final = x, y_final = y1, yaw = math.degrees(math.atan2(y1 −
    node[1], x − node[0]))
27:                else
28:                    x_final = x, y_final = y2, yaw = math.degrees(math.atan2(y2 −
    node[1], x − node[0]))
29:                end if
30:            end if
31:            if (x_final, y_final) is point of room then
32:                temp_sequence.append('position' : (x_final, y_final),' yaw' : yaw)
33:            end if
34:            temp_sequence.append(room[i])
35:        end for
36:        zig_zag_sequence.append(temp_sequence)
37:    end for
38:    return zig_zag_sequence
```

6

Πειράματα - Αποτελέσματα

Τα πειράματα χωρίζονται στις τρεις φάσεις της υλοποίησης. Οι φάσεις είναι οι εξής:

- επιτυχία εντοπισμού κάθε πόρτας του χώρου
- επιλογή βέλτιστης σειράς επίσκεψης των δωματίων
- πλήρης κάλυψη του χάρτη

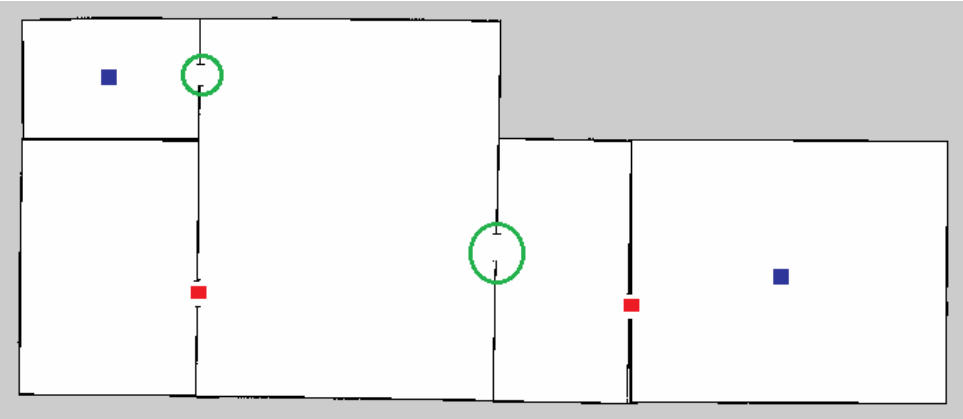
6.1 Πειραματα Ευρεσης Δωματιων

Στο πρώτο μέρος των πειραμάτων χρησιμοποιούνται 80 διαφορετικοί χάρτες με σκοπό την μελέτη της ακρίβειας εύρεσης των διάφορων πορτών του χώρου. Οι 74 συλλέχθηκαν από το [22], ενώ για τους υπόλοιπους 6 δημιουργήθηκε το αντίστοιχο περιβάλλον στο Gazebo και μετά από την διαδικασία SLAM προέκυψαν τα αντίστοιχα OGM.

Σε όλους τους χάρτες δοκιμάστηκε η υλοποίηση του προηγουμένου κεφαλαίου 5.1 και μετρήθηκαν 3 συνολικά στοιχεία:

- True Positives: σημεία που είναι πόρτες και εντοπίστηκαν σωστά
- False Negatives: σημεία που είναι πόρτες αλλά δεν εντοπίστηκαν
- False Positives: σημεία που δεν είναι πόρτες αλλά χαρακτηρίστηκαν ως τέτοιες

Στο σχήμα 6.1 παρουσιάζονται από δύο παραδείγματα για κάθε περίπτωση. Έστω ότι στον χάρτη αυτό προέκυψαν 4 τελικοί κόμβοι πόρτας (2 μπλε και 2 κόκκινοι). Οι δύο που σημειώνονται με κόκκινο χρώμα πράγματι βρίσκονται σε πόρτες του δωματίου, δηλαδή είναι True Positives. Οι δύο μπλε κόμβοι δεν αποτελούν σημείο πόρτας αν και έχουν χαρακτηριστεί έτσι, επομένως είναι False Positive. Τέλος, στις περιπτώσεις που σημειώνονται με πράσινους κύκλους η διαδικασία απέτυχε



Σχήμα 6.1: Παραδείγματα μετρήσεων πειράματος

να εντοπίσει τις δύο πόρτες, δηλαδή πρόκειται για περιπτώσεις Fasle Negative αποτελεσμάτων.

Τα τελικά αποτελέσματα είναι:

Πίνακας 6.1: Μετρήσεις Εντοπισμού Πορτών

True Positive	501
False Negative	40
False Positive	149

Το θετικό συμπέρασμα που προκύπτει είναι ότι η διαδικασία έχει πολύ υψηλό ποσοστό εντοπισμού μιας πόρτας $accuracy = 92.606\%$. Το αρνητικό είναι ότι εμφανίζονται αρκετά συχνά πόρτες σε σημεία που δεν υπάρχουν. Ωστόσο, όπως θα φανεί και από κάποια παραδείγματα δεν είναι τόσο σημαντικό πρόβλημα για δύο λόγους. Πρώτον, πολλές φορές εμφανίζονται παραπάνω πόρτες σε σημεία όπως η μέση ενός διαδρόμου, όπου δεν επηρεάζει αρνητικά την διαδικασία η θεώρηση ενός ακόμη δωματίου. Δεύτερον και σημαντικότερον, η ύπαρξη αυτή προκύπτει εξαιτίας της αναπαράστασης του χώρου. Η συλλογή [22] περιέχει χάρτες σε μορφή εικόνας png και όχι σε μορφή OGM. Αυτό έχει ως αποτέλεσμα τα σημεία που είναι εκτός των δωματίων, αντί να αντιστοιχούν σε τιμές αγνώστου, αντιστοιχούν σε τιμές εμποδίων. Έτσι, ο αλγόριθμος όταν ελέγχει το ποσοστό κάλυψης της προέκτασης της ευθείας των κοντινότερων εμποδίων δίνει μεγάλες τιμές, κάτι που με κανονική OGM αναπαράσταση δεν θα συνέβαινε. Το επιχείρημα αυτό βασίζεται στο γεγονός ότι και οι έξι OGM χάρτες έχουν 100% ακρίβεια στον εντοπισμό των πορτών, χωρίς την ύπαρξη λανθασμένων υποδείξεων.

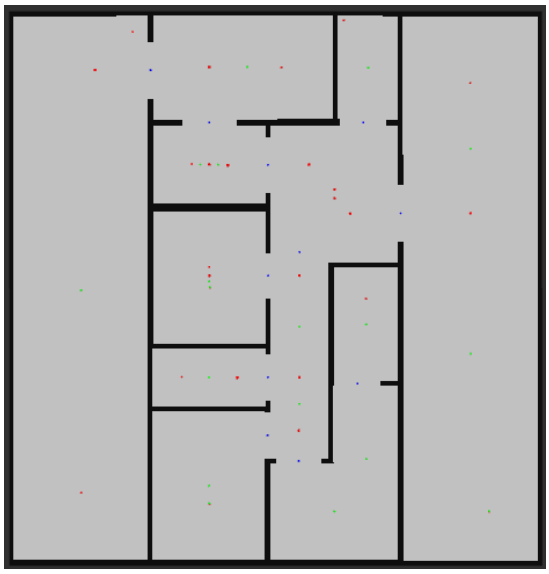
Επομένως, τα δωμάτια ενός χώρου συνδέονται άμεσα με τα χαρακτηριστικά του αντίστοιχου χάρτη. Ο εντοπισμός και διαχωρισμός τους μπορεί να πραγματοποιηθεί με μεγάλη επιτυχία με το αναπτυγμένο σύστημα που εξαρτάται αποκλειστικά από τον εκάστοτε χάρτη.

Στην συνέχεια παρουσιάζονται ορισμένα παραδείγματα χαρτών με τους κόμβους. Οι μπλε κόμβοι αποτελούν τα εκτιμώμενα σημεία πόρτας, τα πράσινα τους

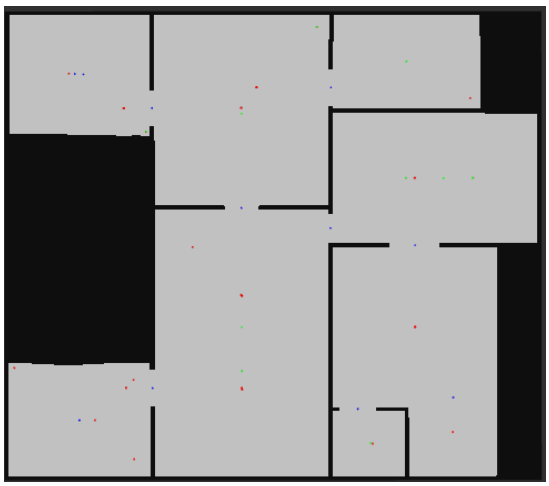
υποψήφιους κόμβους πόρτες που απορρίφθηκαν και τα κόκκινα τα υπόλοιπα σημεία του τοπολογικού χάρτη του χώρου.

Στο 6.2α΄ επιτεύχθηκε σωστός εντοπισμός των δωματίων. Στο 6.2β΄ παρατηρούνται τρεις περιττοί κόμβοι, δύο πάνω αριστερά και ένας κάτω αριστερά. Και οι τρεις προκύπτουν από το γεγονός ότι στο αριστερό μέρος του χώρου υπάρχει ένα τεράστιο εμπόδιο, κάτι που υπό φυσιολογικές συνθήκες θα ήταν άγνωστη περιοχή ή ένα άλλο δωμάτιο. Στο 6.2γ΄ ο χώρος προσομοιώνει μια πραγματική αποθήκη και η διαδικασία εντοπίζει και τις δύο πόρτες του χώρου. Τέλος, στο 6.2δ΄ συναντάται ένας πολύ σύνθετος χώρος. Ωστόσο, η διαδικασία εντοπίζει με επιτυχία και τις έξι πόρτες.

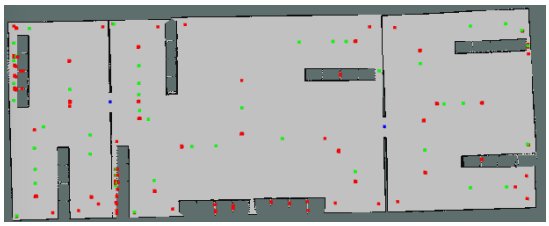
Παρατηρώντας τα παραδείγματα αυτά φαίνεται ότι πράγματι η διαδικασία λειτουργεί με ικανοποιητικά αποτελέσματα σε πολλούς και σύνθετους χώρους όπως είναι μια αποθήκη.



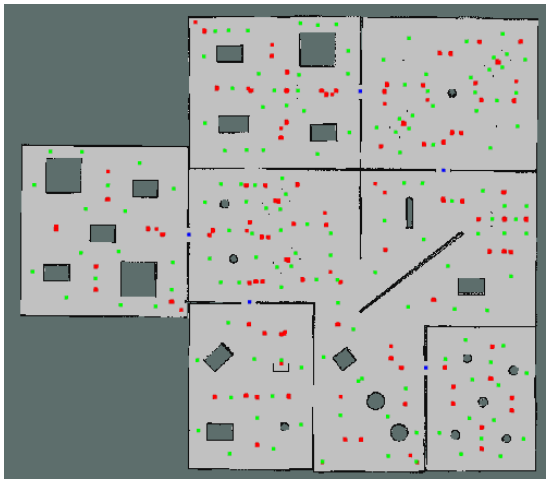
(α΄) Παράδειγμα 1



(β΄) Παράδειγμα 2



(γ΄) Παράδειγμα 3



(δ΄) Παράδειγμα 4

6.2 Πειραματα Ακολουθιας Δωματιων

Στο δέυτερο μέρος των πειραμάτων χρησιμοποιούνται 70 χάρτες από αυτούς που χρησιμοποιήθηκαν και στο πρώτο μέρος των πειραμάτων με σκοπό τη μελέτη του μήκους της διαδρομής επίσκεψης των δωματίων κάθε χώρου. Συγκρίθηκαν 3 διαφορετικοί αλγόριθμοι, οι:

- Επιλογή Κοντινότερου Επόμενου Δωματίου (Κοντινότερος Γείτονας)
- Anneal Hill Climb
- Random Restart Hill Climb

Ο πρώτος αποτελεί μια άπληστη μέθοδο επιλογής, όπου κάθε φορά επιλέγεται ο τρέχων καλύτερος επόμενος κόμβος στην ακολουθία, δηλαδή ο κοντινότερος. Είναι μια απλή και γρήγορη αλλά υποβέλτιστη μέθοδος, εφαρομογή ουσιαστικά του αλγορίθμου επιλογής κοντινότερου γείτονα. Οι δύο άλλοι αναπτύχθηκαν στην ενότητα 5.2.2 και η σύγκριση τους με την άπληστη μέθοδο καταδεικνύει την αποτελεσματικότητα τους. Επίσης, επισημαίνεται ότι οι δύο HC αλγόριθμοι εκτελέστηκαν για πλήθος επαναλήψεων ίσο με 500 φορές επί το πλήθος των πορτών του χάρτη. Σημαντική παρατήρηση είναι ότι ο χρόνος εκτέλεσης και των τριών αλγορίθμων είναι παρόμοιος και αμελητέος.

Ορισμένα τελικά αποτελέσματα μήκους της συνολικής ακολουθίας επίσκεψης των πορτών σε τιμές brushfire, δηλαδή σε μήκος πάνω στο εκάστοτε OGM είναι:

Πίνακας 6.2: Επιλεγμένα Αποτελέσματα Μήκους Ακολουθίας Δωματίων

Κοντινότερος Γείτονας	Anneal HC	RRHC	Ποσοστό Βελτίωσης με RRHC
2809	5201	2809	0%
3338	6870	2979	10.6%
7928	11990	7698	2.9%
2414	2766	2318	3.9%
2769	4087	2519	9.0%
2834	3766	2834	0%
11466	13762	10331	9.8%
996	996	849	14.7%
558	558	558	0%
1149	1149	849	26.1%

Από τα πειράματα προέκυψαν τα παρακάτω ευρήματα. Αρχικά, ο αλγόριθμος anneal HC τις περισσότερες φορές δίνει αποτέλεσμα σημαντικά μεγαλύτερο από το απλό εξαιτίας της στοχαστικότητας του, κάτι που τελικά τον θέτει μη ικανοποιητικό. Μόλις σε 12 χάρτες, στις πιο απλές περιπτώσεις, οι τιμές του anneal HC ήταν ίσες με τις τιμές του κοντινότερου γείτονα.

Επιπλέον, ο RRHC δίνει σταθερά ίδια ή καλύτερα αποτελέσματα από τον αλγόριθμο κοντινότερου γείτονα, επομένως, αποτελεί μια σημαντική βελτιωμένη διαδικασία. Σε απλές περιπτώσεις ή περιπτώσεις με μικρό αριθμό δωματίων τα αποτε-

λέσματα των δύο αλγορίθμων είναι συνήθως ίσα, ωστόσο όσο αυξάνεται το πλήθος των δωματίων και μια βέλτιστη λύση γίνεται πιο σύνθετη, τόσο σημαντιότερη είναι η διαφορά των αποτελεσμάτων τον δύο αλγορίθμων. Συγκεκριμένα, σε 31 χάρτες παρατηρήθηκε βελτίωση στο μήκος της διαδρομής, ενώ στους υπόλοιπους 39 τα αποτελέσματα ήταν ίσα. Κατά μέσο όρο το ποσοστό βελτίωσης του μήκους της διαδρομής είναι 7.597% στους χάρτες που πράγματι παρουσιάζεται βελτίωση και 3.256% συνολικά, ενώ τα τρία μεγαλύτερα ποσοστά είναι 26.1%, 15.9% και 14.7%.

Τα συνολικά αποτελέσματα είναι:

Πίνακας 6.3: Μετρήσεις Μήκους Ακολουθίας Δωματίων

Χάρτες που παρουσίασαν βελίωση	31 / 70
Μέση Τιμή βελτίωσης	3.256%
Τυπική απόκλιση βελτίωσης	5.166%

Συμπερασματικά, ο anneal HC δεν προσφέρει κάποια βελτίωση, ενώ αντίθετα ο RRHC οδηγεί σε ικανοποιητικά αποτελέσματα και από τη στιγμή που δεν επιφέρει χρονική καθυστέρηση συγκριτικά με απλούστερες μεθόδους, όπως του κοντινότερου γείτονα, μπορεί να χρησιμοποιηθεί για τον υπολογισμό της βέλτιστης ακολουθίας δωματίων κάθε χάρτη.

6.3 Πληρης Καλυψη Χαρτη

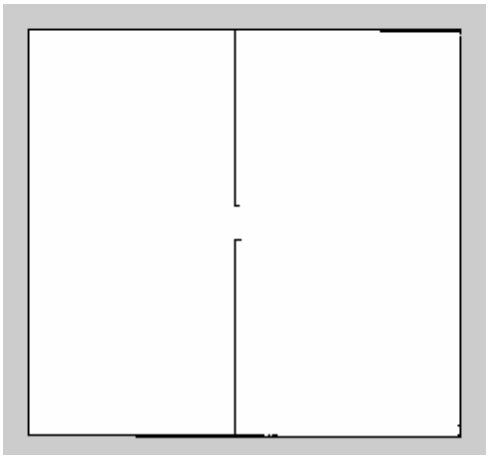
Το τρίτο και μεγαλύτερο μέρος των πειραμάτων περιέχει την πλήρη κάλυψη ενός χώρου. Στα πλαίσια της μελέτης αυτής χρησιμοποιήθηκαν τρεις διαφορετικοί χάρτες με διαφορετικό επίπεδο πολυπλοκότητας περιβάλλοντος, όπως φαίνεται στα αντίστοιχα OGM 6.3. Ο πρώτος χάρτης αποτελεί ένα πολύ απλό περιβάλλον δύο μικρών δωματίων, ο δεύτερος ένα άδειο περιβάλλον με 5 δωμάτια και ο τρίτος ένα πολύπλοκο περιβάλλον με 6 δωμάτια και πολλά εμπόδια.

Στην μελέτη χρησιμοποιήθηκαν 4 διαφορετικά σετ αισθητήρων για την πλήρη προσομοίωση όλων των πιθανών περιπτώσεων. Σε όλες τις περιπτώσεις τοποθετήθηκαν δύο κεραίες στα πλάγια μέρη του οχήματος. Αυτά είναι:

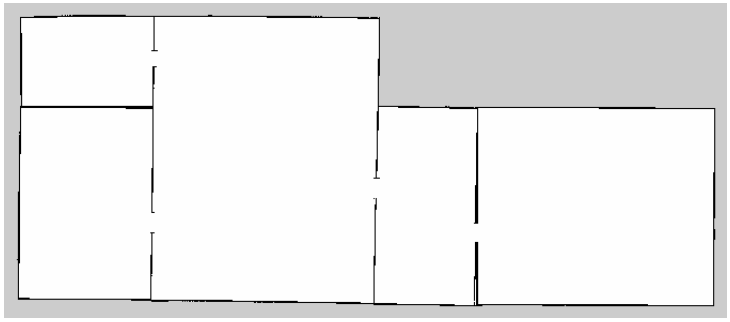
Πίνακας 6.4: Παράμετροι αισθητήρων RFID

Τύπος	FOV	Εύρος
Ευρύ FOV - Μικρή Ακτίνα	70	1 m
Ευρύ FOV - Μεγάλη Ακτίνα	70	3 m
Στενό FOV - Μικρή Ακτίνα	35	1 m
Στενό FOV - Μεγάλη Ακτίνα	35	3 m

Τέλος, χρησιμοποιήθηκαν 4 διαφορετικές στρατηγικές δημιουργίας των αντίστοιχων μονοπατιών κάλυψης του κάθε χώρου, οι οποίες είναι:



(α΄) Χάρτης map_a



(β΄) Χάρτης rooms_3

(γ΄) Χάρτης indoors_with_features

Σχήμα 6.3: Περιβάλλοντα που χρησιμοποιήθηκαν στα πειράματα κάλυψης χώρου

- Στρατηγική Ακολουθίας Τοίχων (Wall Follow)
- Στρατηγική Ζιγκ Ζαγκ
- Απλή Στρατηγική με Δειγματοληψία μικρού σταθερού βήματος
- Απλή Στρατηγική με Δειγματοληψία μεγάλου σταθερού βήματος

Οι δύο πρώτες αποτελούν τις δύο στρατηγικές που υλοποιήθηκαν στον κορμό της παρούσας διπλωματικής εργασία. Οι δύο τελευταίες αποτελούν μια απλοποιημένη εκδοχή των προηγούμενων μεθόδων, ώστε να μελετηθεί εαν η πολυπλοκότητα των υπολογισμών πράγματι βοηθάει στο αποτέλεσμα.

Οι απλές στρατηγικές αποτελούνται από μια ομοιόμορφη δειγματοληψία σημείων σταθερού βήματος, RRHC στην αλληλουχία κάθε δωματίου με σταθερά άκρα, τις πόρτες εισόδου και εξόδου του δωματίου και, τέλος, επιλογή του καλύτερου προσανατολισμού σε κάθε σημείο με τη χρήση του motor schema που υλοποιήθηκε με ίσα βάρη. Το μικρό βήμα δειγματοληψίας τέθηκε ίσο με 0.5 μέτρα και το μεγάλο με 2 μέτρα.

Σε κάθε κάλυψη συλλέγονται οι παρακάτω μετρήσεις:

- Πλήθος poses διαδρομής
- Μήκος συνολικής διαδρομής
- Χρόνος πλήρους κάλυψης
- Ποσοστό συνολικής κάλυψης των εμποδίων του χώρου
- Μέση τιμή πλήθους σαρώσεων κάθε σημείου (mean of scans)
- Διακύμανση πλήθους σαρώσεων κάθε σημείου (std of scans)
- Μέση τιμή μετρικής γωνίας σάρωσης κάθε σημείου που έχει σαρωθεί (mean of scans' angles)
- Διακύμανση μετρικής γωνίας σάρωσης κάθε σημείου που έχει σαρωθεί (std of scans' angles)

Οι επόμενοι πίνακες παρουσιάζουν τα αποτελέσματα από το σύνολο των πειραμάτων χωρισμένα ανά χάρτη και για όλα τα σετ αισθητήρων.

Πίνακας 6.5: Αποτελέσματα στον map_a με Ευρύ FOV - Μικρή Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	150	233	126	10
Total path length	1287.97	1576.87	1169.17	452.03
Total time (secs)	1335	1910	773	97
Coverage percentage	100%	99.84%	99.29%	6.07%
Mean of scans	17.7743	30.6754	10.5984	0.5719
Std of scans	11.5988	17.6976	6.5329	2.605
Mean of scans' angles	0.43401	0.35269	0.2936	0.1451
Std of scans' angles	0.49562	0.4778	0.4554	0.3522

Πίνακας 6.6: Αποτελέσματα στον map_a με Ευρύ FOV - Μεγάλη Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	20	43	126	10
Total path length	809.612	971.98	1169.17	452.03
Total time (secs)	310	501	855	101
Coverage percentage	99.7%	99.688%	100%	37.74%
Mean of scans	19.7494	30.0677	29.21167	4.9159
Std of scans	9.8338	15.84533	20.5956	7.8433
Mean of scans' angles	0.376	0.35031	0.2711	0.2284
Std of scans' angles	0.4843	0.47706	0.4453	0.4198

Πίνακας 6.7: Αποτελέσματα στον map_a με Στενό FOV - Μικρή Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	215	297	126	10
Total path length	1268.06	1557.15	1169.17	452.03
Total time (secs)	1620	2898	789	106
Coverage percentage	100%	99.53%	99.221%	5.83%
Mean of scans	14.5	21.17665	6.01634	0.298
Std of scans	9.4428	11.9857	4.5871	1.393
Mean of scans' angles	0.69623	0.59139	0.62365	0.5636
Std of scans' angles	0.45988	0.49157	0.48446	0.4959

Πίνακας 6.8: Αποτελέσματα στον map_a με Στενό FOV - Μεγάλη Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	46	60	126	10
Total path length	809.612	971.98	1169.17	452.03
Total time (secs)	517	618	833	96
Coverage percentage	99.92%	99.61%	100%	37.66%
Mean of scans	15.3517	19.4023	15.5097	2.785
Std of scans	6.5877	9.2672	12.0028	4.744
Mean of scans' angles	0.5591	0.3301	0.53505	0.3198
Std of scans' angles	0.4964	0.47025	0.49876	0.4664

Πίνακας 6.9: Αποτελέσματα στον rooms_3 με Ευρύ FOV - Μικρή Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	370	572	286	30
Total path length	3875.882	4537.929	3490.03	1627.56
Total time (secs)	3580	5524	2808	408
Coverage percentage	98.91%	98.396%	97.881%	12.25%
Mean of scans	14.38819	19.8414	10.6211	0.8323
Std of scans	10.0359	14.5949	6.6085	2.7883
Mean of scans' angles	0.43892	0.42213	0.44157	0.47
Std of scans' angles	0.49599	0.4939	0.49657	0.4991

Πίνακας 6.10: Αποτελέσματα στον rooms_3 με Ευρύ FOV - Μεγάλη Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	81	114	286	30
Total path length	2562.367	2968.296	3490.03	1627.56
Total time (secs)	1095	1350	2550	405
Coverage percentage	98.577%	98.638%	99%	59.45%
Mean of scans	21.6901	28.3939	31.47261	6.146
Std of scans	13.7954	16.8131	13.2472	7.338
Mean of scans' angles	0.35177	0.304012	0.29845	0.4266
Std of scans' angles	0.47752	0.45998	0.457581	0.4945

Πίνακας 6.11: Αποτελέσματα στον rooms_3 με Στενό FOV - Μικρή Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	473	692	286	30
Total path length	3972.408	4687.32	3490.03	1627.56
Total time (secs)	4125	8682	3157	352
Coverage percentage	98.15%	95.34%	95.521%	11.49%
Mean of scans	9.729	10.0599	5.43721	0.4127
Std of scans	7.051	9.1189	3.96784	1.458
Mean of scans' angles	0.7326	0.63012	0.72391	0.796
Std of scans' angles	0.4425	0.48277	0.44705	0.4029

Πίνακας 6.12: Αποτελέσματα στον rooms_3 με Στενό FOV - Μεγάλη Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	126	160	286	30
Total path length	2560.6	3024.52	3490.06	1627.56
Total time (secs)	1301	1978	3133	411
Coverage percentage	98.033%	97.881%	99.031%	55.91%
Mean of scans	14.2375	14.8172	15.4	3.138
Std of scans	8.5554	8.9399	7.8128	4.059
Mean of scans' angles	0.43052	0.43306	0.61159	0.4821
Std of scans' angles	0.49514	0.4954	0.4873	0.4996

Πίνακας 6.13: Αποτελέσματα στον indoors_with_features με Ευρύ FOV - Μικρή Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	608	944	743	46
Total path length	8393.4909	9919.85	10007.63	2883.37
Total time (secs)	9047	10715	7808	998
Coverage percentage	92.74%	94.628%	91.9%	14.31%
Mean of scans	8.2309	10.917	8.94	0.827
Std of scans	8.9038	10.3473	7.569	2.55
Mean of scans' angles	0.487172	0.42139	0.3862	0.5881
Std of scans' angles	0.49983	0.49377	0.4868	0.4921

Πίνακας 6.14: Αποτελέσματα στον indoors_with_features με Ευρύ FOV - Μεγάλη Ακτίνα

Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	245	316	743	46
Total path length	5241.9	5972.02	10007.63	2883.37
Total time (secs)	3390	4619	7939	972
Coverage percentage	96.901%	96.801%	96.49%	60.21%
Mean of scans	27.6053	32.8307	41.6034	7.385
Std of scans	10.4321	21.0289	28.9465	8.907
Mean of scans' angles	0.283	0.3033	0.2055	0.5
Std of scans' angles	0.45048	0.4597	0.4041	0.49

Πίνακας 6.15: Αποτελέσματα στον indoors_with_features με Στενό FOV - Μικρή Ακτίνα

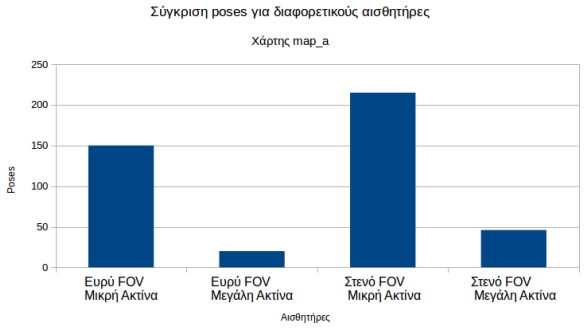
Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	820	1179	743	46
Total path length	8479.88	10105.16	10007.63	2883.37
Total time (secs)	10651	15268	7898	977
Coverage percentage	92.916%	91.267%	96.72%	56.35%
Mean of scans	6.42698	6.25858	21.4439	3.823
Std of scans	5.8097	6.14225	16.4325	5.1012
Mean of scans' angles	0.64802	0.6669	0.4615	0.607
Std of scans' angles	0.47758	0.47131	0.4985	0.4484

Πίνακας 6.16: Αποτελέσματα στον indoors_with_features με Στενό FOV - Μεγάλη Ακτίνα

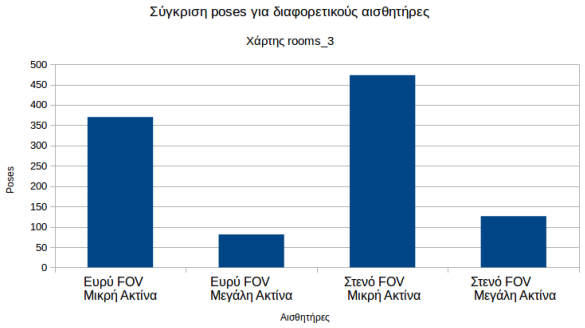
Στρατηγική	Wall Follow	Zig Zag	Simple small	Simple Large
Number of Poses	344	419	743	46
Total path length	5097.26	5841.74	10007.63	2883.37
Total time (secs)	4172	5720	7751	942
Coverage percentage	96.327%	96.664%	86.95%	13.7%
Mean of scans	16.4983	18.7782	4.6193	0.4473
Std of scans	10.6672	12.91	4.21	1.447
Mean of scans' angles	0.45	0.5121	0.6484	0.7263
Std of scans' angles	0.49749	0.4998	0.4774	0.4458

Αρχικά, παρατηρείται ότι η χρήση κεραιών RFID με μεγάλη ακτίνα και ευρύ πεδίο όρασης είναι η καλύτερη δυνατή, διότι οδηγεί σε πιο ικανοποιητικά αποτελέσματα. Πρώτον, μειώνει σημαντικά το χρόνο της συνολική κάλυψης του χώρου, καθώς απαιτούνται πολύ λιγότεροι στόχοι. Δεύτερον αυξάνει το μέσο όρο σαρώσεων ανά σημείο συγκριτικά με τις υπολοίπες κεραίες, διότι σαρώνονται συνεχώς περισσότερα τμήματα του χώρου. Τρίτον, στις δύο πρώτες στρατηγικές μειώνει τη μετρική γωνίας σάρωσης κάθε σημείου, δηλαδή η κάλυψη των εμποδίων γίνεται πιο ομοιόμορφα από τα δύο εύρη γωνιών. Όλες οι συγκρίσεις παρουσιάζονται και στα επόμενα γραφήματα 6.4. Με μπλε είναι η σύγκριση του αριθμού των στόχων, με κόκκινο του χρόνου πλοήγησης σε κάθε χάρτη, με πορτοκαλί της μέσης τιμής του αριθμού σαρώσεων των σημείων και με πράσινο της μέσης τιμής της μετρικής της γωνίας σάρωσης. Η σύγκριση γίνεται μεταξύ των τεσσάρων σετ αισθητήρων σε κάθε χάρτη και σε όλες τις περιπτώσεις παρατηρείται μικρότερος αριθμός στόχων, χρόνου και μετρικής γωνίας και μεγαλύτερη μέση τιμή σαρώσεων όταν χρησιμοποιούνται κεραίες με ευρύ FOV και μεγάλη ακτίνα. Σημειώνεται ότι τα αποτελέσματα αυτά αφορούν τη στρατηγική ακολουθίας τοίχων, η οποία είναι και η βασική στρατηγική της μελέτης αυτής.

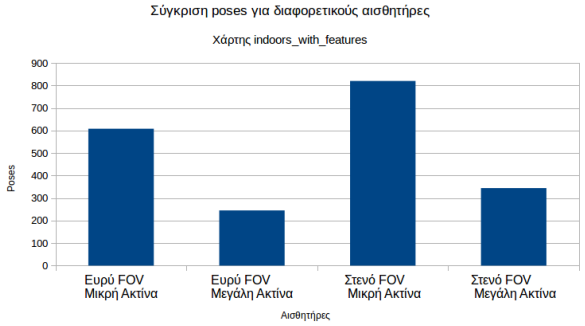
Επιπλέον, στο σχήμα 6.5 παρουσιάζεται η τελική πλήρης κάλυψη του χάρτη από τα δύο διαφορετικά σετ αισθητήρων χρησιμοποιώντας τη στρατηγική ακολουθίας τοίχων.



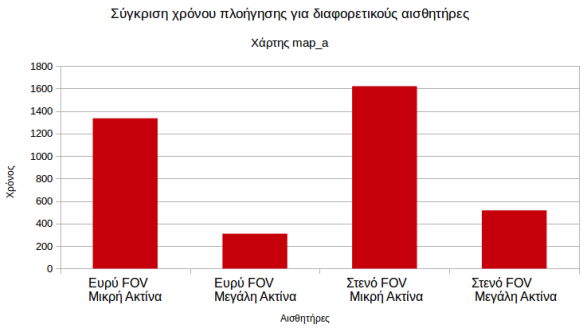
(α΄) Σύγκριση αριθμών poses στον map_a



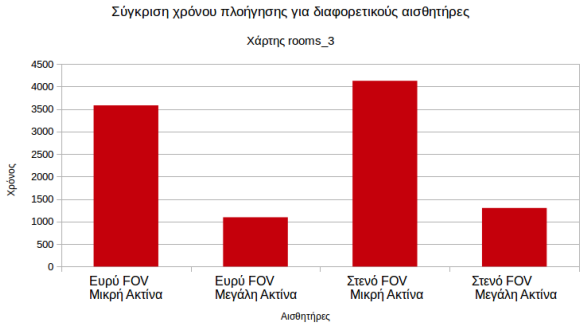
(β΄) Σύγκριση αριθμών poses στον rooms_3



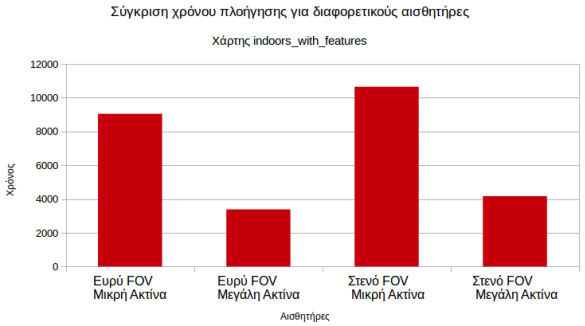
(γ΄) Σύγκριση αριθμών poses στον indoors_with_features



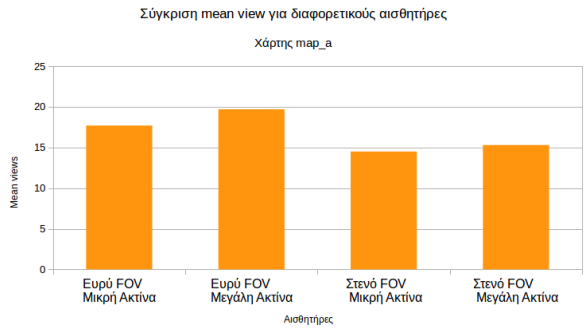
(δ΄) Σύγκριση χρόνου πλοήγησης στον map_a



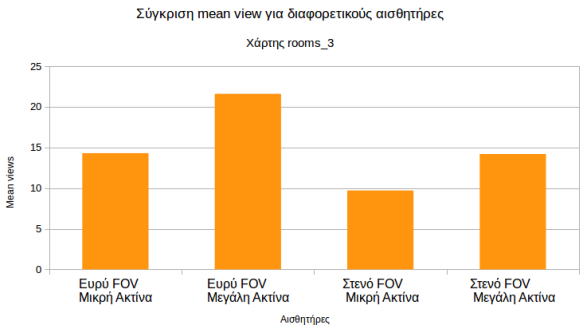
(ε΄) Σύγκριση χρόνου πλοήγησης στον rooms_3



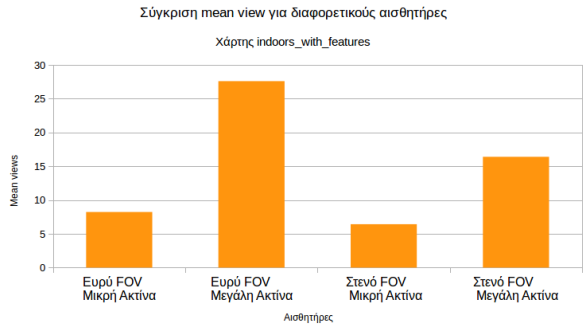
(στ΄) Σύγκριση χρόνου πλοήγησης στον indoors_with_features



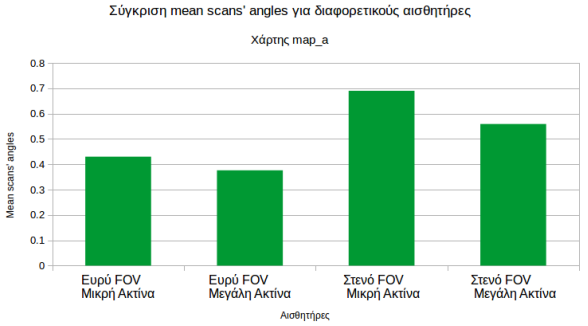
(ζ΄) Σύγκριση mean views στον map_a



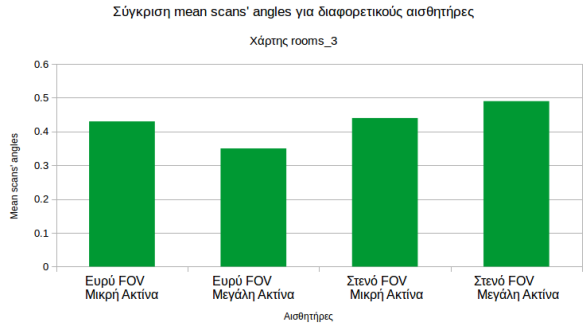
(η΄) Σύγκριση mean views στον rooms_3



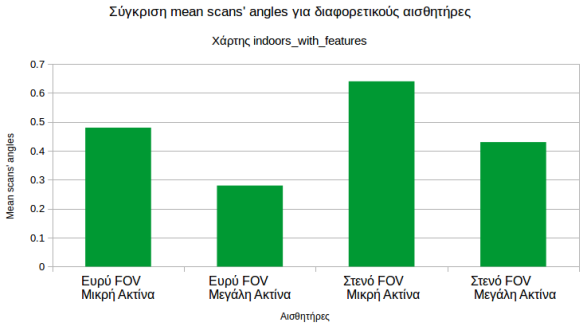
(θ΄) Σύγκριση mean views στον indoors_with_features



(ι΄) Σύγκριση mean scans' angles στον map_a

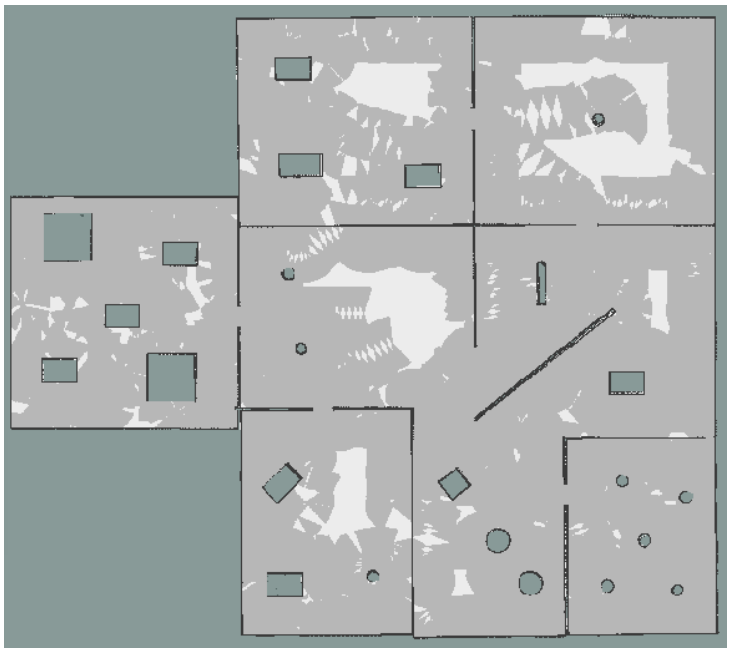


(ια΄) Σύγκριση mean scans' angles στον rooms_3



(ιβ΄) Σύγκριση mean scans' angles στον indoors_with_features

Σχήμα 6.4: Σύγκριση αποτελεσμάτων για διαφορετικούς αισθητήρες με στρατηγική ακολουθίας τοίχων



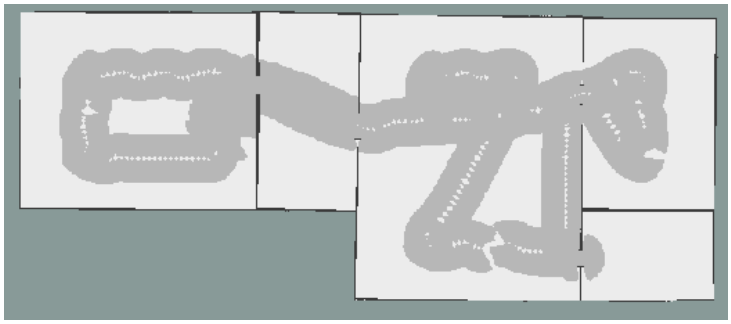
(α΄) Στενό FOV - Μικρή Ακτίνα

(β΄) Ευρύ FOV - Μεγάλη Ακτίνα

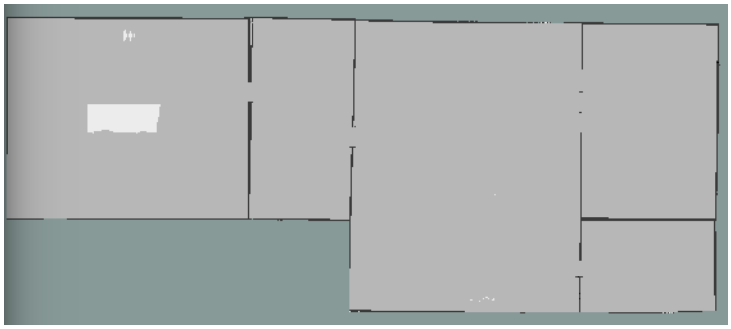
Σχήμα 6.5: Πλήρης κάλυψη χάρτη indoors_with_features με στρατηγική ακολουθίας τοίχων για διαφορετικούς αισθητήρες

Έπειτα, η χρήση σταθερού βήματος δειγματοληψίας, ανεξάρτητου από τα χαρακτηριστικά των αισθητήρων, έχει σημαντικές αρνητικές επιπτώσεις στη συνολική κάλυψη του χώρου. Στην περίπτωση που το βήμα είναι πολύ μεγαλύτερο από την ακτίνα των αισθητήρων η διαδικασία κάλυψης αποτυγχάνει πλήρως, όπως στο σχήμα 6.6α΄. Αντίθετα, στην περίπτωση που το βήμα δειγματοληψίας είναι πολύ μικρότερο από την ακτίνα των αισθητήρων ο συνολικός χρόνος κάλυψης αυξάνεται σημαντικά, φυσικά με εξαιρετικά αποτελέσματα κάλυψης, όπως φαίνονται και στο

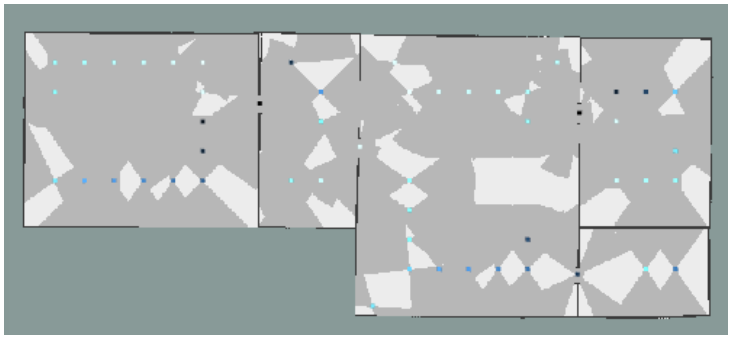
σχήμα 6.6β΄. Συνεπώς, η αναπτυγμένη μέθοδος δειγματοληψίας με πολλαπλά βήματα με εξάρτηση από τα χαρακτηριστικά των αισθητήρων, οδηγεί σε μια συνολική βελτίωση της διαδικασίας ως προς την χρονική διάρκεια και την επιτυχία κάλυψης, όπως φαίνεται και στην πρόβλεψη κάλυψης 6.6γ΄, όπου και παρουσιάζονται τα σημεία - στόχοι, αλλά και στην τελική κάλυψη του χώρου 6.6δ΄.



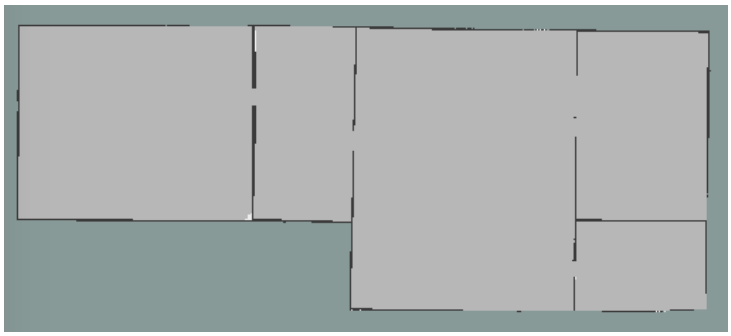
(α΄) Ευρύ FOV - Μικρή Ακτίνα με Απλή Στρατηγική μεγάλου βήματος δειγματοληψίας



(β΄) Στενό FOV - Μεγάλη Ακτίνα με Απλή Στρατηγική μικρού βήματος δειγματοληψίας



(γ΄) Ευρύ FOV - Μεγάλη Ακτίνα με στρατηγική ακολουθίας τοίχων, Πρόβλεψη Κάλυψης

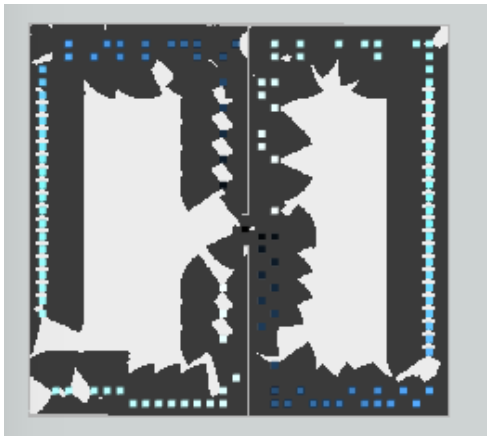


(δ΄) Ευρύ FOV - Μεγάλη Ακτίνα με στρατηγική ακολουθίας τοίχων

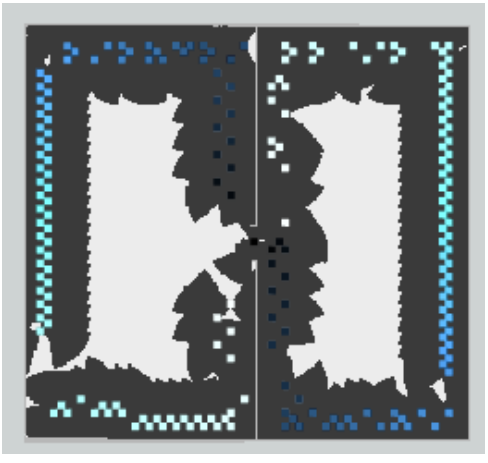
Σχήμα 6.6: Πλήρης κάλυψη χάρτη rooms_3 με τις διάφορες στρατηγικές και διαφορετικούς αισθητήρες

Τέλος, η στρατηγική zig zag συγκριτικά με τη wall follow αυξάνει μεν το πλήθος των σημείων και άρα τον συνολικό απαιτούμενο χρόνο, όμως αυξάνει κατά μέσο όρο και το πλήθος σαρώσεων των εμποδίων, ενώ στα περισσότερα πειράματα οι σαρώσεις είναι πιο ομοιόμορφες, ως αποτέλεσμα της μείωσης της μετρικής γωνίας σάρωσης. Συνεπώς, εαν η διαδικασία κάλυψης είναι ανάγκη να εκτελεστεί άμεσα και γρήγορα, μπορεί να επιλεγεί η στρατηγική ακολουθίας τοίχων, ενώ εαν είναι θεμιτό να καλυφθεί ο χώρος όσο το δυνατόν πιο αναλυτικά τότε μπορεί να επιλεγεί η στρατηγική ζιγκ ζαγκ.

Στο σχήμα 6.7 παρουσιάζονται οι κόμβοι του χάρτη map_a για τις δύο αυτές στρατηγικές. Σημειώνεται και η ακολουθία τους για κάθε δωμάτιο, η οποία εκκινεί από τα σκούρα σημεία και καταλήγει στα ανοικτόχρωμα, ενώ με μαύρο έχει τονιστεί η περιοχή που εκτιμάται ότι θα καλυφθεί από τα σημεία αυτά.



(α΄) Wall Follow Στρατηγική



(β΄) Ζιγκ Ζαγκ Στρατηγική

Σχήμα 6.7: Στόχοι και εκτίμηση κάλυψης του χάρτη map_a για δύο στρατηγικές με Ευρύ FOV - Μικρή Ακτίνα

7 Συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάζονται συνοπτικά τα συμπεράσματα που προέκυψαν από τα αποτελέσματα του συνόλου των πειραμάτων. Στη συνέχεια αναφέρονται προβλήματα που παρουσιάστηκαν κατά τη διάρκεια των υλοποιήσεων και των πειραμάτων.

7.1 Γενικα Συμπερασματα

Στα πλαίσια της παρούσας διπλωματικής εργασίας αναπτύχθηκε μια διαδικασία διαχωρισμού ενός χάρτη OGM στα επιμέρους δωμάτια του και ο υπολογισμός της βέλτιστης αλληλουχίας επίσκεψης τους. Επιπλέον, δημιουργήθηκε ένα μονοπάτι πλοήγησης μέσα σε κάθε δωμάτιο με στόχο την πλήρη κάλυψη των εμποδίων του χώρου, προκειμένου να πραγματοποιηθεί η απογραφή προϊόντων που είναι αποθηκευμένα στα δωμάτια με τη βοήθεια κεραιών RFID που φέρει το ρομποτικό όχημα. Το μονοπάτι αυτό δημιουργείται κάθε φορά συναρτήσει του εκάστοτε OGM και των χαρακτηριστικών των κεραιών και στοχεύει στην όσο το δυνατό μεγαλύτερη κάλυψη του χώρου και ταυτόχρονα την ελαχιστοποίηση του μήκους της διαδρομής.

Με βάση τα αποτελέσματα των πειραμάτων που αναλύθηκαν στο κεφάλαιο 6 και αφορούν τον εντοπισμό των δωματίων του χώρου, τον υπολογισμό της ακολουθίας επίσκεψης τους και την πλήρη κάλυψη του χάρτη, παρατηρήθηκαν τα εξής:

- Ο διαχωρισμός του χώρου σε δωμάτια μπορεί να πραγματοποιηθεί με εξαιρετικά αποτελέσματα όταν λαμβάνονται υπόψη τα χαρακτηριστικά ολόκληρου του χάρτη.
- Ο RRHC αλγόριθμος βελτιώνει το συνολικό μήκος κάθε ακολουθίας δωματίων, ενώ απαιτεί αμελητέο χρόνο εκτέλεσης συγκριτικά με απλούστερους αλγορίθμους.

- Η δειγματοληψία σημείων με πολλαπλά βήματα επιτυγχάνει να εντοπιστούν σημεία από ολόκληρο τον χάρτη που οδηγούν σε υψηλά ποσοστά κάλυψης των εμποδίων.
- Όσο πιο σύνθετο είναι ένα περιβάλλον, τόσο δυσκολότερη είναι η πλήρης κάλυψη του.
- Η χρήση κεραιών RFID με μεγάλη ακτίνα και ευρύ FOV βελτιώνουν σημαντικά τον χρόνο πλοήγησης και το συνολικό ποσοστό κάλυψης του χώρου.
- Το σύστημα κάλυψης των εμποδίων είναι εύρωστο και για τις δύο στρατηγικές που αναπτύχθηκαν, καθώς τα εμπόδια σαρώνονται αρκετές φορές και με διαφορετικές γωνίες, κάτι που οδηγεί σε πολύ μεγάλη πιθανότητα σωστής απογραφής των προϊόντων.
- Σε περιπτώσεις που είναι σημαντική η γρήγορη κάλυψη του χώρου μπορεί να χρησιμοποιηθεί η στρατηγική ακολουθίας τοίχων.
- Σε περιπτώσεις που απαιτείται η αναλυτική κάλυψη του χώρου χωρίς χρονικό περιορισμό μπορεί να χρησιμοποιηθεί η ζιγκ ζαγκ στρατηγική. Αυτή αυξάνει κατά μέσο όρο τον αριθμό σαρώσεων κάθε σημείου, οι οποίες, μάλιστα, πραγματοποιούνται και με πιο ομοιόμορφη κατανομή των φάσεων σάρωσης.

7.2 Προβληματα

Ένα βασικό πρόβλημα που διαπιστώθηκε σε ορισμένες περιπτώσεις αποτελεί η αστοχία εντοπισμού της θέσης του οχήματος κατά την πλοήγηση του στον χώρο. Συγκεκριμένα, σε σύνθετα περιβάλλοντα που απαιτούν υψηλότερη υπολογιστική ισχύ ή σε περιπτώσεις κάλυψης μεγάλης χρονικής διάρκειας αυξάνεται η πιθανότητα το όχημα να εκτιμήσει λανθασμένα την τρέχουσα θέση του. Εαν πράγματι το όχημα αποπροσανατολιστεί, η όλη διαδικασία απογραφής θα περιέχει σημαντικά λάθη. Συνεπώς, είναι αναγκαία η μελέτη του AMCL και η περαιτέρω βελτίωση του.

Ένα άλλο πρόβλημα που παρατηρήθηκε είναι η ταχύτητα επεξεργασίας και ανανέωσης της πληροφορίας κάλυψης. Αν και η διαδικασία αυτή δεν έχει υψηλές απαιτήσεις υπολογιστικής ισχύος, ο συνδιασμός της με τα συστήματα προσομοίωσης του χώρου και πλοήγησης οδηγεί σε περιορισμό των διαθέσιμων υπολογιστικών πόρων. Σε σύνθετα περιβάλλοντα παρατηρήθηκε η αδυναμία ανανέωσης του συστήματος κάλυψης σε πραγματικό χρόνο με αποτέλεσμα τη δημιουργία ορισμένων ασυνεχιών στο πεδίο κάλυψης. Συμπεραίνεται ότι η διαδικασία αυτή σε πραγματικούς πολύπλοκους χώρους για να λειτουργεί σωστά έχει ανάγκη από ένα σύστημα υψηλής υπολογιστικής δύναμης.

Τέλος, αποτέλεσε πρόβλημα η αναπαράσταση των διάφορων χαρτών OGM. Όπως αναφέρθηκε και στο υποκεφάλαιο 6.1, είναι σημαντικό ο χάρτης να αντιπροσωπεύει πλήρως το περιβάλλον, δίχως κενά ή λάθη. Το σύνολο των χαρτών που χρησιμοποιήθηκε στα πειράματα εντοπισμού δωματίων περιέχει χάρτες όπου ο άγνωστος χώρος αναπαριστάται ως εμπόδιο, πράγμα που οδηγεί σε λανθασμένα συμπεράσματα. Αντίστοιχα προβλήματα μπορούν να προκύψουν και από χάρτες πραγματικού SLAM, οι οποίοι περιέχουν ατέλειες και ανακρίβειες σε περιοχές που δεν έχουν χαρτογραφηθεί πλήρως. Συνεπώς, είναι σημαντικό προτού εκκινήσει η

ανάλυση του εκάστοτε περιβάλλοντος να έχει εξασφαλιστεί η ύπαρξη ενός άρτιου χάρτη.

8

Μελλοντικές επεκτάσεις

Αρχικά, μια σημαντική βελτίωση του συστήματος που δεν μελετήθηκε στην παρούσα διπλωματική είναι η βελτίωση του συστήματος εντοπισμού της θέσης του οχήματος, δηλαδή του AMCL. Από το σύνολο των πειραμάτων παρατηρήθηκε ότι το όχημα αποπροσανατολίζεται σε περιπτώσεις πλοήγησης μεγάλης χρονικής διάρκειας και σε σύνθετους χώρους που η απαιτούμενη υπολογιστική ισχύς είναι αρκετά αυξημένη. Ο ακριβής εντοπισμός της θέσης του είναι καίριας σημασίας, διότι τυχόν αποκλίσεις θα οδηγήσουν σε λανθασμένες εκτιμήσεις της θέσης των προϊόντων, κάτι που πρέπει να αποφευχθεί.

Επιπλέον, είναι σημαντικό η υπάρχουσα υλοποίηση να εξεταστεί και σε πραγματικές καταστάσεις για να ελεχθεί κατά πόσο η διαδικασία μπορεί να χρησιμοποιηθεί σε κανονικές συνθήκες. Οι προσομοιώσεις μπορεί να προσεγγίζουν ικανοποιητικά την πραγματικότητα, ωστόσο υπάρχουν λεπτομέρειες που μπορούν να επηρεάσουν σημαντικά. Τα OGM που προκύπτουν από διαδικασία SLAM πραγματικού χώρου δεν είναι τόσο ακριβή όσο τα OGM από προσομοιώσεις, ενώ η πλοήγηση ενός πραγματικού οχήματος είναι πιθανό να επιφέρει αποκλίσεις στα αποτελέσματα τα οποία δεν μπορούν να προβλεφθούν. Επομένως, πρέπει να ελεχθούν τα προβλήματα ή οι αστοχίες που είναι πιθανό να προκύψουν στην τοπολογική ανάλυση και στην πλήρη κάλυψη ενός πραγματικού χώρου.

Μια ακόμη επέκταση του συστήματος είναι η χρήση πολλαπλών ρομποτικών οχημάτων. Η κατανομή του χώρου κάλυψης σε περισσότερους πράκτορες, η επικοινωνία μεταξύ τους και ο συγχρονισμός τους απαιτεί μια πιο περίπλοκη υλοποίηση, ωστόσο επιφέρει σημαντική βελτίωση στον χρόνο πλήρους κάλυψης του χώρου και πιθανώς και στην ακρίβεια εντοπισμού των προϊόντων.

Ο εντοπισμός δωματίων μπορεί να αξιοποιηθεί ακόμη καλύτερα με την ταξινόμηση τους σε διάφορες κατηγορίες, όπως για παράδειγμα σε δωμάτια και διαδρόμους. Μια μέθοδος ταξινόμησης υλοποιήθηκε στο A′ και μπορεί να χρησιμοποιηθεί σε περιπτώσεις μη ομοιόμορφης κατανομής των προϊόντων στους διάφορους χώρους. Σε ένα ρεαλιστικό παράδειγμα αποθήκης είναι πιθανό προϊόντα να έχουν

τοποθετηθεί μόνο στους διαδρόμους. Επομένως, το όχημα θα πρέπει να προσαρμόσει τη συμπεριφορά του στην εκάστοτε κατάσταση. Η υλοποίηση ενός τέτοιου πλήρως προσαρμοστικού συστήματος θα γλυτώσει το όχημα από περιττές διαδρομές και θα εξοικονομίσει χρόνο και ενέργεια.

Τέλος, όσον αφορά το μονοπάτι σημείων κάλυψης υπάρχουν ορισμένες επεκτάσεις που μπορούν να μελετηθούν. Μια σημαντική και σύνθετη μέθοδος αποτελεί η υλοποίηση μιας δυναμικής αναπροσαρμογής της αλληλουχίας σημείων κάθε δωματίου. Στη μέθοδο που αναπτύχθηκε στο υποκεφάλαιο 5.4 η σειρά προσπέλασης των σημείων έχει εξαρχής οριστεί, ενώ η σειρά επίσκεψης των δωματίων προσαρμόζεται στο δωμάτιο που βρίσκεται το όχημα κατά την εκκίνηση της διαδικασίας. Αντίστοιχη προσαρμογή θα μπορούσε να μελετηθεί και για τα σημεία κάθε δωματίου. Η αρχική θέση είναι ένα στοιχείο που θα μπορούσε να χρησιμοποιηθεί για να βελτιώσει το συνολικό μήκος κίνησης σε πραγματικό χρόνο. Επίσης, ένα τέτοιο σύστημα θα μπορούσε να αποφασίζει να αγνοήσει σημεία ή χώρους οι οποίοι έχουν ήδη καλυφθεί, μειώνοντας το συνολικό χρόνο της διαδικασίας. Μια άλλη μέθοδος είναι η αντιμετώπιση του μονοπατιού και ως ένα ενιαίο, αντί αποκλειστικά ανά δωμάτιο. Με τον τρόπο αυτό μπορεί ένα κεντρικό δωμάτιο να καλύπτεται τμηματικά και ενδιάμεσα του να παρεμβάλλονται άλλα δωμάτια, μειώνοντας το συνολικό μήκος της διαδρομής.

Παραρτήματα

Ταξινόμηση Δωματίων

Στην ενότητα 5.1 υλοποιήθηκε η μελέτη ενός γνωστού χώρου και ο διαχωρισμός του στα επιμέρους δωμάτια. Κάτι που δεν μελετήθηκε στον κορμό της διπλωματικής αυτής, αλλά φέρει επιστημονικού ενδιαφέροντος, είναι η ταξινόμηση των επιμέρους χώρων ανάλογα με το είδος τους. Ο διαχωρισμός που μελετήθηκε περιέχει δύο κλάσεις, τους διαδρόμους και τα δωμάτια. Αυτός μπορεί να χρησιμοποιηθεί σε περιπτώσεις που το ρομποτικό όχημα είναι επιθυμητό να έχει διαφορετική συμπεριφορά ανάλογα με τον χώρο στον οποίο βρίσκεται. Για παράδειγμα, μπορεί να είναι γνωστό ότι δεν υπάρχουν προϊόντα στους διαδρόμους, επομένως δεν μελετάται η κάλυψη των τοίχων των διαδρόμων.

Στόχος είναι ο χαρακτηρισμός κάθε χώρου χρησιμοποιώντας τα λιγότερα δυνατα στοιχεία. Η μέθοδος που υλοποιήθηκε εξάγει στοιχεία από το ήδη υπολογισμένο GVD, το οποίο άλλωστε περιέχει την δομή του χώρου. Μετά την διαδικασία διαχωρισμού των κόμβων του χώρου σε δωμάτια 5.1.3, κάθε χώρος αποτελείται από τους κόμβους με έναν, τρεις ή περισσότερους γείτονες στο GVD όπως είχαν εντοπιστεί εξ αρχής, καθώς και από τους κόμβους με δύο γείτονες που όμως δεν αποτελούν σημεία πόρτας. Εξάγονται, λοιπόν, ορισμένα χαρακτηριστικά από αυτούς τους κόμβους τα οποία είναι τα εξής:

- αριθμός κόμβων του δωματίου
- μέσος όρος αριθμού γειτόνων των κόμβων
- μέση απόσταση μεταξύ των κόμβων μετασχηματισμένη σε μέτρα
- μέση τιμή της τιμής brushfire των κόμβων μετασχηματισμένη σε μέτρα
- διακύμανση της τιμής brushfire των κόμβων μετασχηματισμένη σε μέτρα
- μέση τιμή της απόσταση κάθε κόμβου με τον κοντινότερο του μετασχηματισμένη σε μέτρα

Σημειώνεται ότι οι τιμές brushfire είναι υπολογισμένες σε pixel του OGM. Η απόσταση ενός pixel αντιστοιχεί σε μια σταθερή πραγματική απόσταση σε κάθε χάρτη,

που ονομάζεται ανάλυση (resolution) του χάρτη και προκύπτει κατά το SLAM του χώρου. Έτσι, τα δεδομένα μετασχηματίζονται σε πραγματικές αποστάσεις, ώστε να είναι γενικευμένα και να μην παρουσιάζουν πρόβλημα ακόμη και για τον ίδιο χώρο με διαφορετική ανάλυση.

Συλλέχθηκαν 492 δείγματα από 70 χάρτες του [22] για τον σχηματισμό ενός συνόλου δεδομένων. Για την εκπαίδευση του μοντέλου έγινε χρήση ενός γραμμικού SVM[15]. Η επιλογή αυτή έγινε, καθώς όπως φάνηκε από τα αποτελέσματα τα δεδομένα είναι γραμμικώς διαχωρίσιμα. Το μοντέλο χωρίστηκε σε 70% - 30% αντίστοιχα για το σύνολο εκπαίδευσης και το σύνολο ελέγχου. Σε 10 πειράματα εκπαίδευσης το αποτέλεσμα είναι ποσοστό ακρίβειας 89,523 % κατά μέσο όρο, με το μικρότερο ποσοστό να βρίσκεται στο 85,13 % και το υψηλότερο στο 93,24 %.

Το συμπέρασμα της μελέτης είναι ότι κάθε χώρος μπορεί να ταξινομηθεί με μεγάλη ακρίβεια σε δωμάτιο ή διάδρομο εξάγοντας δεδομένα από τα σημαντικά σημεία του GVD. Η μέθοδος αυτή είναι αρκετά απλή στην υλοποίηση και γρήγορη στην εκτέλεση, καθώς χρησιμοποιεί στοιχεία που είναι ήδη γνωστά. Τέλος, οι χώροι είναι γραμμικώς διαχωρίσιμοι χάρη στην χρήση της μέσης τιμής και της διακύμανσης των τιμών brushfire στους κόμβους κάθε δωματίου. Τα δύο αυτά χαρακτηριστικά αποτελούν τα πλέον καθοριστικά, διότι η μορφή ενός διαδρόμου είναι αρκετά ομοιόμορφη. Αυτό έχει ως αποτέλεσμα μικρές τιμές διακύμανσης των brushfire, αλλά και μικρές τιμές brushfire, συγκριτικά με τα δωμάτια που έχουν συνήθως πιο σύνθετη δομή.

[15] hrefhttps://scikit-learn.org/stable/modules/svm.htmlhttps://scikit-learn.org/stable/modules/svm.html

Βιβλιογραφία

[1] Lee McCauley. "*AI Armageddon and the Three Laws of Robotics*". Ethics and Information Technology, 9:153–164, 07 2007.

[2] Charles D. Emery. "*The Use of Portable Barcode Scanners in Collections Inventory*". Collection Management, 13(4):1–17, 1991.

[3] Ricardo Tesoriero, Jose A. Gallud, María Lozano, and Victor Penichet. "*Tracking Autonomous Entities using RFID Technology*". Consumer Electronics, IEEE Transactions on, 55:650 – 655, 06 2009.

[4] Richard Bormann, Florian Jordan, Wenzhe Li, Joshua Hampp, and Martin Hagele. "*Room segmentation: Survey, implementation, and analysis*". pages 1019–1026, 05 2016.

[5] Stanley Brown. "*Coverage Path Planning and Room Segmentation in Indoor Environments using the Constriction Decomposition Method*", 2016.

[6] B. Kaleci, Ç. M. Şenler, H. Dutağacı, and O. Parlaktuna. "*A probabilistic approach for semantic classification using laser range data in indoor environments*". In "*2015 International Conference on Advanced Robotics (ICAR)*", pages 375–381, July 2015.

[7] Oscar Mozos. "*Semantic Labeling of Places with Mobile Robots*". 61, 01 2008.

[8] D. Filliat, E. Battesti, S. Bazeille, G. Duceux, A. Gepperth, L. Harrath, I. Jebari, R. Pereira, A. Tapus, C. Meyer, S. Ieng, R. Benosman, E. Cizeron, J. Mamanna, and B. Pothier. "*RGBD object recognition and visual texture classification for indoor semantic mapping*". In "*2012 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*", pages 127–132, April 2012.

[9] Enric Galceran and Marc Carreras. "*A survey on coverage path planning for robotics*". Robotics and Autonomous Systems, 61:1258–1276, 2013.

[10] Howie Choset and Philippe Pignon. "*Coverage Path Planning: The Boustrophedon Cellular Decomposition*". 1998.

[11] Djoko Purwanto Muhammad Fuad. "*Wall-Following Using a Kinect Sensor for Corridor Coverage Navigation*". 2014.

[12] Denis Wolf and Gaurav S. Sukhatme. "*Mobile Robot Simultaneous Localization and Mapping in Dynamic Environments*". Auton. Robots, 19:53–65, 07 2005.

[13] Jan Oliver Wallgrün. "*Exploiting Qualitative Spatial Constraints for Multi-hypothesis Topological Map Learning*". pages 141–158, 09 2009.

[14] E. W. Dijkstra. "*A note on two problems in connexion with graphs*". Numerische Mathematik, 1(1):269–271, Dec 1959. ISSN 0945-3245.

[15] Morgan Quigley, Ken Conley, Brian P Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. "*ROS: an open-source Robot Operating System*". volume 3, 01 2009.

[16] Συκούδη Αμανατίδου Πελαγία. "*Πλοήγηση ρομποτικού οχήματος με χρήση δυναμικών πεδίων και νευρωνικών δικτύων σε ένα γνωστό περιβάλλον και ανάπτυξη μεθόδων αποδοτικής πλοήγησης*", 2013.

[17] Wyatt Baldwin. "*Dijkstar*". https://github.com/wylee/Dijkstar, 2018.

[18] John Montgomery. "*Tackling the travelling salesman problem: hill-climbing*". http://www.psychicorigami.com/2007/05/12/tackling-the-travelling-salesman-problem-hill-climbing/, 2007.

[19] Fiorella Sibona. "*Sending Goals to the Navigation Stack - Python ROS node version*". https://hotblackrobotics.github.io/en/blog/2018/01/29/action-client-py/, 2018.

[20] P. V. Nikitin, R. Martinez, S. Ramamurthy, H. Leland, G. Spiess, and K. V. S. Rao. "*Phase based spatial identification of UHF RFID tags*". In "*2010 IEEE International Conference on RFID (IEEE RFID 2010)*", pages 102–109, April 2010.

[21] C. Hekimian-Williams, B. Grant, Xiuwen Liu, Zhenghao Zhang, and P. Kumar. "*Accurate localization of RFID tags using phase difference*". In "*2010 IEEE International Conference on RFID (IEEE RFID 2010)*", pages 89–96, April 2010.

[22] Li Tingguang, Ho Danny, Li Chenming, Zhu Delong, Wang Chaoqun, and Max Q.-H. Meng. "*HouseExpo: A Large-scale 2D Indoor Layout Dataset for Learning-based Algorithms on Mobile Robots*". arXiv preprint arXiv:1903.09845, 2019.