ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΕΦΑΡΜΟΓΕΣ ΠΛΟΗΓΗΣΗΣ ΣΕ ΡΟΜΠΟΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

(ROBOTIC APPLICATION OF NAVIGATION)

ΑΛΕΠΗΣ ΑΝΑΣΤΑΣΙΟΣ

(Α.Μ. 41509)

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

Δρ. ΝΙΚΟΛΑΟΥ ΓΡΗΓΟΡΙΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΙΓΑΛΕΩ ΣΕΠΤΕΜΒΡΙΟΣ 2018

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Αλέπης Αναστάσιος, του Στέλιου, φοιτητής του Τμήματος **Μηχανικών Αυτοματισμού Τ.Ε,** του Πανεπιστημίου Δυτικής Αττικής, πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

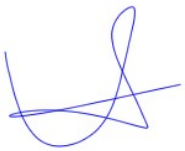
«Η Πτυχιακή Εργασία (Π.Ε) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε, ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα, σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασή της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση Π.Ε με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε πρέπει να ολοκληρώσει εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσής της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18. παρ.5 του ισχύοντος Εσωτερικού Κανονισμού».

Ο Δηλών **Ημερομηνία**

Αλέπης Αναστάσιος 17/9/2018



Περίληψη

Ο σκοπός της πτυχιακής εργασίας είναι η δημιουργία αλγόριθμού, ο οποίος θα χρησιμοποιηθεί για την πλοήγηση ενός ρομποτικού συστήματος μέσα σε ένα άγνωστο περιβάλλον. Για τον σχεδιασμό του αλγορίθμου χρησιμοποιήθηκε το λογισμικό Matlab και οι δοκιμές έγιναν, με την βοήθεια του εξομοιωτή V-Rep, στο ρομποτικό σύστημα Youbot της KUKA.

Ο αλγόριθμός καταφέρνει χρησιμοποιώντας μετρήσεις από τους αισθητήρες του ρομποτικού συστήματος να εξερευνήσει τον άγνωστο χώρο, δημιουργώντας σταδιακά έναν χάρτη του περιβάλλοντος. Με την βοήθεια του χάρτη σχεδιάζει εφικτές διαδρομές μέσα στο περιβάλλον καθιστώντας την πλοήγηση εφικτή.

Για την δημιουργία του αλγορίθμου χρησιμοποιήθηκαν τεχνικές SLAM και αναζήτησης βέλτιστης διαδρομής, οι οποίες αποτελούν το θεωρητικό υπόβαθρο αυτής της πτυχιακής εργασίας και παρουσιάζονται αναλυτικότερα στα παρακάτω κεφάλαια.

Abstract

The purpose of this thesis is creating an algorithm, who will be used to navigate a robotic system in an unknown environment. For the design of this algorithm, the Matlab software was used and the tests have been done with the assistance of the V-Rep simulator, in the robotic system of KUKA Youbot.

The algorithm manages to explore the unknown environment, using the calculations provided from the robotic system, to effectively mapping down the unknown area. With guideness of the map, it can then draw possible routes in the specific environment, making the navigation feasible.

For the creation of the algorithm, SLAM techniques were being used as well as search techniques for the best route possible, which altogether are the theoretical background of this thesis and are detailed introduced in the following chapters.

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω την οικογένεια μου που μου συμπαραστάθηκε και έκανε υπομονή όλο το διάστημα που χρειάστηκε για τη δημιουργία αυτής της εργασίας. Το συμφοιτητή και φίλο Δημήτριο Λεβέντη για την βοήθειά του και τέλος τον καθηγητή μου Γρηγόριο Νικολάου που πίστεψε σε εμάς από την αρχή.

Περιεχόμενα

Κεφάλαιο 1 - Ρομποτική

Η ρομποτική είναι κομμάτι της μηχανικής και διαφόρων άλλων επιστημών, όπως της μηχανολογίας, της ηλεκτρολογίας και των υπολογιστών. Ασχολείται με το σχεδιασμό, την κατασκευή, την λειτουργιά και την χρήση των ρομπότ.

Τα ρομπότ χρησιμοποιούνται για την αυτοματοποίηση διαφόρων εργασιών όπως την κατασκευή προϊόντων σε εργοστάσια, την εξερεύνηση δυσπρόσιτων σημείων, την ιατρική και το στρατό.

1.1 Βιομηχανικά ρομπότ

Τα βιομηχανικά ρομπότ χρησιμοποιούνται στην παραγωγή προϊόντων. Είναι αυτόματα, προγραμματιζόμενα και έχουν ελευθερία κίνησης σε δύο ή παραπάνω άξονες. Οι πιο συνήθεις εφαρμογές τους είναι η συγκόλληση, βαφή, συναρμολόγηση, τοποθέτηση, συσκευασία και έλεγχος ποιότητας.

Τα πιο συχνά χρησιμοποιημένα ρομπότ είναι τα αρθρωτά ρομπότ, τα SCARA ρομπότ και τα Cartesian coordinate ρομπότ, (γνωστά και ως ρομπότ xyz). Κάποια από τα ρομπότ είναι προγραμματισμένα να ακολουθούν πιστά συγκεκριμένες κινήσεις, επαναλαμβανόμενα, χωρίς διαφοροποιήσεις και μεγάλη ακρίβεια. Αυτές οι κινήσεις περιγράφονται από προγράμματα που καθορίζουν την κατεύθυνση, την ταχύτητα και την απόσταση τους.

Άλλα ρομπότ είναι πιο ευέλικτα στις κινήσεις τους ανάλογα με αντικείμενο στο οποίο εργάζονται. Για παράδειγμα ένα ρομπότ χρησιμοποιώντας machine vision μπορεί να αλλάξει το τρόπο συμπεριφοράς του ανάλογα με αυτά που «βλέπει».

Εικόνα 1.1 – Γραμμή παραγωγής αυτοκινήτων.

–Πηγή: https://blog.robotiq.com/imagining-a-world-without-robotics-in-manufacturing (16/07/2018)

1.2 Ψυχαγωγικά ρομπότ

Τα ψυχαγωγικά ρομπότ όπως εννοείται και από το όνομα τους, στοχεύουν στην αναψυχή. Οι εφαρμογές τους μπορεί να είναι από πολύ απλές, όπως ένα παιχνίδι για παιδιά, έως πολύ σύνθετες, όπως αξιοθέατα σε πάρκα, animatronics και αυτοματισμοί για την παραγωγή ειδικών εφέ σε ταινίες.

Εικόνα 2.2 – Ψυχαγωγικά ρομπότ.

–Πηγή: http://archive.boston.com/bigpicture/2009/03/robots.html (16/07/2018)

1.3 Οικιακά ρομπότ

Τα οικιακά ρομπότ ασχολούνται με τις δουλειές του σπιτιού. Είναι ευρέως διαθέσιμα, προσιτά και έχουν διάφορες σύνθετες τεχνολογίες όπως το SLAM. Κάποια από αυτά είναι οι ρομποτικές σκούπες, οι καθαριστές πισίνας και ρομπότ για την κοπή του χορταριού.

Εικόνα 3.3 – Ρομποτική σκούπα.

–Πηγή: https://bgr.com/2018/07/16/prime-day-deals-robot-vacuum-deebot-n79s/ (16/07/2018)

1.4 Στρατιωτικά ρομπότ

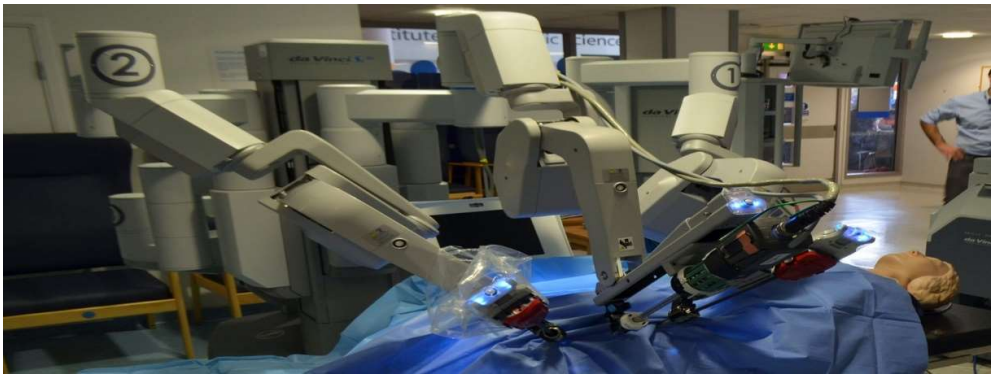
Τα στρατιωτικά ρομπότ αποτελούν το μεγαλύτερο κομμάτι έρευνας και ανάπτυξης της πολεμικής τεχνολογίας. Είναι πηγή συζητήσεων και απασχολεί των κόσμο, λόγω των επιπτώσεων που μπορούν να έχουν. Το πιο γνωστό και χαρακτηριστικό παράδειγμα, είναι το UAV ή αλλιώς Unmanned Aerial Vehicle. Πρόκειται για αεροσκάφη ή drones τα οποία δεν έχουν επιβάτες και είτε είναι πλήρης αυτόνομα, είτε ελέγχονται από απόσταση.

Εικόνα 4.4 – Unmanned Aerial Vehicle.

–Πηγή: http://gpsworld.com/uav-industry-demonstrates-innovation/ (16/07/2018)

1.5 Ιατρικά ρομπότ

Μία από τις πιο σημαντικές εφαρμογές της ρομποτικής είναι στην ιατρική. Χρησιμοποιείται στην χειρουργική επιτρέποντας στους ιατρούς να πραγματοποιήσουν επεμβάσεις με πολύ μεγάλη ακρίβεια, μειώνοντας έτσι τον κίνδυνο για τους ασθενείς. Επίσης τα τελευταία χρόνια παρατηρείται αύξηση στην έρευνα ρομποτικών προσθετικών μελών, για την αντικατάσταση άκρων ή ακόμα και οργάνων όπως η καρδιά, τα μάτια και τα αυτιά.

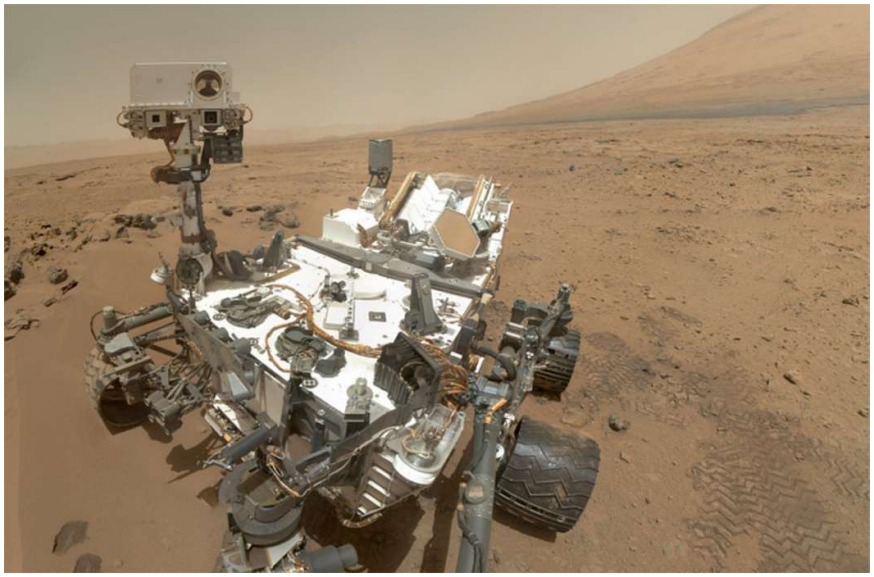


Εικόνα 5.5 – Ρομποτικοί βραχίονες στην ιατρική.

–Πηγή: http://lifemag.org/article/medical-robots-the-future-of-surgery (16/07/2018)

1.6 Εξερευνητικά ρομπότ

Η εξερεύνηση είναι μια από τις κυριότερες μεθόδους του ανθρώπου για την κατανόηση του κόσμου. Με την βοήθεια της ρομποτικής δίνεται πλέον η πιθανότητα εξερεύνησης δυσπρόσιτων περιοχών που ο άνθρωπος από μόνος του δεν θα έφτανε ποτέ. Ρομπότ όπως το υποβρύχιο «Οδύσσεια ΙΙβ», το οποίο αναπτύχθηκε από τους επιστήμονες του M.I.T, βοηθάει την εξερεύνηση των ωκεανών. Τα mars rovers με τελευταίο το curiosity επέτρεψαν στον άνθρωπο να δει τον πλανήτη Άρη.



Εικόνα 6.5 – Mars Rover.

–Πηγή: https://www.jpl.nasa.gov/spaceimages/details.php?id=PIA16239 (16/07/2018)

Κεφάλαιο 2 – Εντοπισμός θέσης

Η αναζήτηση της θέσης μας μέσα στο χώρο είναι ένα πολύ παλιό πρόβλημα και έχει πολλές λύσεις. Χαρακτηριστικό παράδειγμα είναι ο προσδιορισμός της θέσης ενός πλοίου στη θάλασσα.

Σήμερα οι ναυτικοί μπορούν εύκολα να βρουν τη θέση χρησιμοποιώντας GPS. Πριν το GPS οι ναυτικοί υπολόγιζαν τη θέση τους με βάση την ταχύτητα του πλοίου και την ένδειξη της πυξίδας τους. Η τεχνική αυτή ονομάζεται dead reckoning και όπως αναφέρθηκε παραπάνω χρησιμοποιεί την ταχύτητα, την κατεύθυνση και τον χρόνο.

Πηγαίνοντας ακόμα πιο πίσω, πριν την ανακάλυψη της πυξίδας, η εύρεση της θέσης ήταν ακόμα πιο δύσκολη. Ο υπολογισμός θέσης γινόταν πάλι με βάση την ταχύτητα και την κατεύθυνση αλλά με μεγάλα σφάλματα. Για την διόρθωση της θέσης, οι άνθρωποι εκείνης της εποχής χρησιμοποιούσαν σημεία ενδιαφέροντος (landmarks) όπως ένα νησί, ένας φάρος ή τα αστέρια τα οποία σύγκριναν με χάρτες ή χρησιμοποιούσαν για την δημιουργία ενός χάρτη.

Τα ίδια προβλήματα και τους ίδιους τρόπους λύσης έχουν σήμερα τα ρομποτικά συστήματα.

2.1 Χωρίς χάρτη

Η μέθοδος αυτή ονομάζεται dead reckoning. Το πρώτο βήμα είναι η εκτίμηση της θέσης του ρομπότ ως μία συνάρτηση f(), που περιγράφει τις αλλαγές από την μία στιγμή στην άλλη. Ένα μοντέλο οχήματος περιγράφει την εξέλιξη της κατάστασης του ρομπότ σε συνάρτηση με τις εισόδους ελέγχου. Στα ρομποτικά συστήματα γίνεται χρήση οδομετρίας. Ένας αισθητήρας οδομετρίας επιστρέφει την απόσταση που διανύθηκε από το ρομπότ, συνήθως μετρώντας την γωνιακή ταχύτητα των τροχών. Για την κατεύθυνση της κίνησης χρησιμοποιούνται ηλεκτρονικές πυξίδες, γυροσκόπια η διαφορική οδομετρία. Η διαφορική οδομετρία υπολογίζεται συγκρίνοντας την ταχύτητα του δεξιού τροχού με την ταχύτητα του αριστερού τροχού.

Γράφουμε λοιπόν ένα μοντέλο διακριτού χρόνου βασισμένο στην οδομετρία δ(k) = (δ_d, δ_θ) είναι η απόσταση και η κατεύθυνση. Η αρχική θέση περιγράφεται:

$$\xi_k \sim \begin{pmatrix} cos\theta(k) & -sin\theta(k) & x(k) \\ sin\theta(k) & cos\theta(k) & y(k) \\ 0 & 0 & 1 \end{pmatrix}$$

Επιλέγουμε να κινηθούμε στον άξονα x κατά δ_d και μετά να περιστραφούμε κατά δ_θ δίνοντας την νέα θέση

$$\xi_{(k+1)}\sim\begin{pmatrix} cos\theta(k) & -sin\theta(k) & x(k) \\ sin\theta(k) & cos\theta(k) & y(k) \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & \delta_d \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} cos\delta_\theta & -sin\delta_\theta & 0 \\ sin\delta_\theta & cos\delta_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\sim\begin{pmatrix} \cos(\theta(k)+\delta_\theta) & -sin(\theta(k)+\delta_\theta) & x(k)+\delta_d cos\theta(k) \\ sin(\theta(k)+\delta_\theta) & cos(\theta(k)+\delta_\theta) & y(k)+\delta_d sin\theta(k) \\ 0 & 0 & 1 \end{pmatrix}$$

$$\xi_{(k+1)}\sim\begin{pmatrix} x(k)+\delta_d(k)\cos(\theta(k)+\delta_\theta) \\ y(k)+\delta_d(k)\sin(\theta(k)+\delta_\theta) \\ \theta(k)+\delta_\theta \end{pmatrix}$$

Η τελευταία εξίσωση μας δίνει την νέα κατάσταση σε σχέση με την προηγούμενη και την οδομετρία. Η παραπάνω σχέση θεωρεί ότι η οδομετρία είναι τέλεια. Για να εισάγουμε σφάλμα στην οδομετρία προσθέτουμε συνεχής τυχαίες μεταβλητές v_d και v_θ στα δ_d και δ_θ αντίστοιχα.

$$\xi_{(k+1)}\sim\begin{pmatrix} x(k)+(\delta_d(k)+v_d)\cos(\theta(k)+\delta_\theta+v_\theta) \\ y(k)+(\delta_d(k)+v_d)\sin(\theta(k)+\delta_\theta+v_\theta) \\ \theta(k)+\delta_\theta+v_\theta \end{pmatrix} (2.1)$$

Τέλος έχουμε την εξίσωση f()

$$x(k+1)=f(x(k),\delta(k),v(k)) \ (2.2)$$

Όπου k είναι η επόμενη χρονική στιγμή, δ(k) οι μετρήσεις της οδομετρίας και v(k) ο τυχαίος θόρυβος.

Το πρόβλημά μας είναι πώς θα κάνουμε την καλύτερη εκτίμηση της θέσης μας με βάση την προηγούμενη θέση και την οδομετρία με θόρυβο. Θα χρησιμοποιήσουμε το Kalman filter το οποίο θα μας επιστρέψει την καλύτερη εκτίμηση για ένα σύστημα με θόρυβο Gaussian κατανομής. Το φίλτρο είναι ένας επαναλαμβανόμενος αλγόριθμος, ο οποίος ανανεώνει σε κάθε χρονική στιγμή την καλύτερη εκτίμηση της θέσης και πόσο σίγουρη είναι.

Το Kalman filter είναι φτιαγμένο για γραμμικά συστήματα για τον λόγο αυτό κάνουμε γραμμηκοποίηση της εξίσωσης x(k)

$$\hat{x}(k+1) = \hat{x}(k) + F_x\big(x(k) - \hat{x}(k)\big) + F_v(k))$$

Όπου F_x και F_v είναι Jacobians οι οποίες γράφονται και ως $\partial f/\partial x$ και $\partial f/\partial v$ αντίστοιχα. Αυτή η προσέγγιση ενός μη γραμμικού συστήματος ονομάζεται Extended Kalman filter ή EKF. Οι Jacobians βρίσκονται διαφοροποιώντας την εξίσωση (2.1) και εξετάζοντας για v=0

$$F_x = \frac{\partial f}{\partial x}\Big|_{v=0} = \begin{pmatrix} 1 & 0 & -\delta_d(k) - \sin(\theta(k) + \delta_\theta) \\ 0 & 1 & \delta_d(k)\cos(\theta(k) + \delta_\theta) \\ 0 & 0 & 1 \end{pmatrix} \text{(2.3)}$$

$$F_v = \frac{\partial f}{\partial v}\Big|_{v=0} = \begin{pmatrix} \cos(\theta(k) + \delta_\theta) & -\delta_d(k)\sin(\theta(k) + \delta_\theta) \\ \sin(\theta(k) + \delta_\theta) & \delta_d(k)\cos(\theta(k) + \delta_\theta) \\ 0 & 1 \end{pmatrix} \text{(2.4)}$$

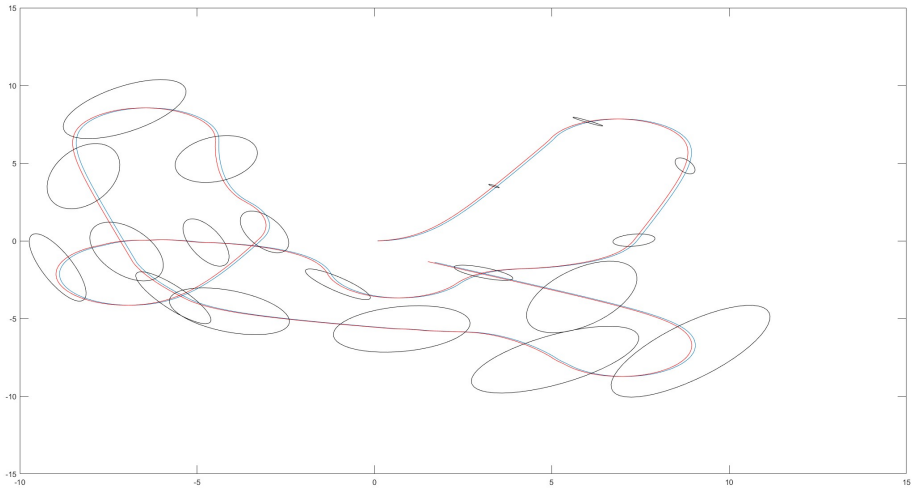
Τώρα μπορούμε να γράψουμε τις εξισώσεις εκτίμησης του EKF

$$\hat{x}(k+1|k) = f(\hat{x}(k), \delta(k), 0) \text{ (2.5)}$$

$$\hat{P}(k+1|k) = F_x(k)\hat{P}(k|k)F_x(k)^T + F_v(k)\hat{V}F_v(k)^T \text{ (2.6)}$$

Οι οποίες περιγράφουν την κατάσταση και την εξέλιξη της κατανομής με τον χρόνο.

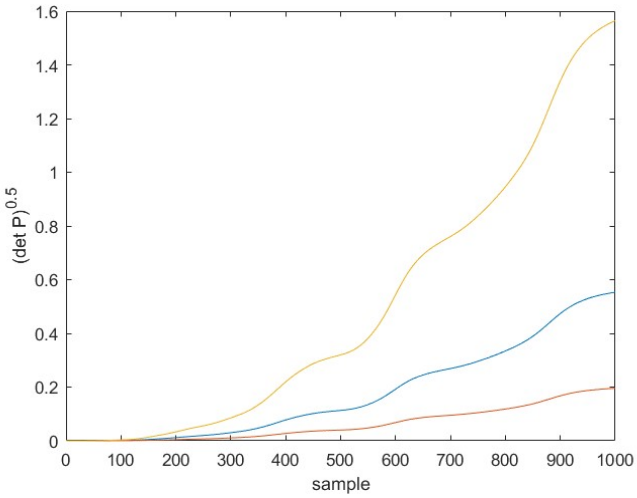
Με την βοήθεια του Robotics toolbox από τον Peter Corke μπορούμε να δούμε ένα παράδειγμα-εξομοίωση της συμπεριφοράς αυτής της μεθόδου. Η εξομοίωση είναι ένα όχημα το οποίο εκτελεί μια τυχαία κίνηση. Χρησιμοποιούμε μια μικρή κατανομή θορύβου με την υπόθεση ότι το όχημα έχει μια καλή ιδέα για το πού βρίσκεται αρχικά.



Εικόνα 2.1 – Dead reckoning με χρήση EKF.

Όπως βλέπουμε στο παραπάνω γράφημα της κίνησης του ρομπότ, η μπλε γραμμή είναι η πραγματική τροχιά, η κόκκινη είναι η εκτίμηση του EKF και τέλος οι ελλείψεις μας δείχνουν την αβεβαιότητα στο σημείο (όσο μεγαλύτερες τόσο μεγαλύτερη η αβεβαιότητα).

Στο παρακάτω γράφημα βλέπουμε την μεταβολή της αβεβαιότητας με την πάροδο του χρόνου, από την αρχή της εξομοίωσης (time stop 0) μέχρι και το τέλος της (time step 1000). Επίσης βλέπουμε την διαφορά που δημιουργεί η αύξηση της κατανομής του θορύβου. Έχουμε με μπλε την κανονική τιμή της κατανομής, με κόκκινη το ½ της τιμής και με κίτρινο την διπλάσια τιμή της τιμής.



Εικόνα 2.2 – Μεταβολή αβεβαιότητας (χωρίς χρήση χάρτη).

Κώδικας παραδείγματος:

```
% Αρχικοποίηση οχήματος εξομοίωσης.
% Διασπορά σφάλματος μετρήσεων του οχήματος. Σε αυτό το παράδειγμα έχουμε για απόσταση 2cm και γωνία 0.5º.
V = diag([0.005, 0.5*pi/180].^2);

% Χαρακτηριστικά οχήματος.
veh = Vehicle(V);

% Ενσωμάτωση οδηγού στο όχημα με τυχαία διαδρομή.
veh.add_driver(RandomPath(10));

% Αρχική διασπορά σφάλματος αρχικής θέσης. Υποθέτουμε ότι γνωρίζουμε αρκετά καλά που βρισκόμαστε οπότε η τιμή της είναι μικρή.
P0 = diag([0.005, 0.005, 0.001].^2);

% Δημιουργία και εξομοίωση του EKF.
ekf = EKF(veh, V, P0);
ekf.run(1000);

% Γράφημα της κίνησης του οχήματος. (Εικόνα 2.1)
veh.plot_xy();
hold on

% Γράφημα της εκτιμώμενης θέσης από το EKF. (Εικόνα 2.1)
ekf.plot_xy('r');
hold on

% Γράφημα της αβεβαιότητας του EKF για κάθε θέση (x,y) με μορφή ελλείψεων. (Εικόνα 2.1)
ekf.plot_ellipse([], 'g');
hold off;


% Δημιουργία γραφήματος με την συνολική αβεβαιότητα (x,y,θ) για διαφορετικές τιμές του πίνακα διασποράς V. (V = a*V). (Εικόνα 2.2)
i=1;
figure;
for i = 1:3
   V = diag([0.005, 0.5*pi/180].^2);
   if i == 1
      a = 2;
      V = a*V;
   elseif i == 2
      a = 1;
      V = a*V;
   elseif i == 3;
      a = 0.5;
      V = a*V;
   end
```

```
    ekf = EKF(veh, V, P0);
    ekf.run(1000);
    ekf.plot_P();
    hold on;
end
legend('a = 2','a = 1','a = 0.5');
hold off;
```

2.2 Με χάρτη

Όπως είδαμε στο προηγούμενο κεφάλαιο χωρίς την χρήση κάποιου χάρτη η αβεβαιότητα συνέχιζε να αυξάνεται. Σε αυτό το κεφάλαιο θα δούμε την ίδια εξομοίωση χρησιμοποιώντας αυτήν τη φορά έναν χάρτη.

Αρχικά χρειαζόμαστε ένα μοντέλο αισθητήρα που θα βλέπει τα ορόσημα (landmarks) μέσα στον χάρτη

$$z = h(x_v, x_f, w) \quad (2.7)$$

Όπου x_v είναι η κατάσταση του οχήματος, x_f είναι η γνωστή θέση του παρατηρηθέντος ορόσημου σε συντεταγμένες του κόσμου και w είναι το μοντέλο των σφαλμάτων του αισθητήρα.

Έστω λοιπόν ότι ο αισθητήρας μας είναι ένα laser το οποίο επιστρέφει την απόσταση και είναι τοποθετημένο επάνω στο όχημα. Η μέτρηση για την i-οστη παρατήρηση $x_{fi} = (x_i, y_i)$ είναι

$$z = \begin{pmatrix} \sqrt{(y_i - y_v)/(x_i - x_v)} \\ tan^{-1}(y_i - y_v)/(x_i - x_v) - \theta_v \end{pmatrix} + \begin{pmatrix} w_r \\ w_\beta \end{pmatrix} \quad (2.8)$$

Όπου $z = (r, \beta)^T$ και r είναι η απόσταση, β είναι γωνία, και ο θόρυβος της μέτρησης είναι

$$\begin{pmatrix} w_r \\ w_\beta \end{pmatrix} \sim N(0, W), \quad W = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\beta^2 \end{pmatrix}$$

Ο διαγώνιος πίνακας κατανομής μας δείχνει ότι τα σφάλματα απόστασης και γωνίας είναι ανεξάρτητα μεταξύ τους. Στη δικιά μας εξομοίωση θα ορίσουμε τις τιμές $\sigma_r = 0.1$ και $\sigma_\beta = 1^o$.

Χρησιμοποιώντας την (2.8) το όχημα μπορεί να κάνει μια εκτίμηση για την απόσταση και την γωνία του ορόσημου με βάση την δικιά του εκτιμημένη θέση και να την συγκρίνει με την πραγματική θέση του ορόσημου από τον χάρτη. Η διαφορά που προκύπτει δείχνει το σφάλμα στην εκτίμηση της θέσης του οχήματος. Αυτή η διαφορά

$$v(k+1) = z(k+1) - h(\hat{x}(k+1|k), x_f, 0)$$

είναι απαραίτητη για την λειτουργία του Kalman filter και ονομάζεται innovation.

Το Kalman filter χρησιμοποιεί την innovation για να διορθώσει την εκτίμηση και να ανανεώσει την αβεβαιότητα.

Όπως και στο προηγούμενο κεφάλαιο για την χρήση του Kalman filter πρέπει να γραμμικοποιήσουμε την εξίσωση (2.7)

$$z(k) = \hat{h} + H_z\big(x(k) - \hat{x}(k)\big) + H_w w(k) \qquad (2.9)$$

Όπου διαφοροποιώντας την (2.8) βρίσκουμε τις Jacobians

$$H_{x_i} = \frac{\partial h}{\partial x_v}\Big|_{w=0} = \begin{pmatrix} -\frac{x_i - x_v(k)}{r} & -\frac{y_i - y_v(k)}{r} & 0 \\ \frac{x_i - x_v(k)}{r^2} & -\frac{y_i - y_v(k)}{r^2} & -1 \end{pmatrix} \quad (2.10)$$

$$H_w = \frac{\partial h}{\partial w}\Big|_{w=0} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (2.11)$$

Χρησιμοποιούμε την innovation για να ανανεώσουμε την εκτιμώμενη κατάσταση των (2.5) και (2.6)

$$\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + K(k+1)v(k+1) \quad (2.12)$$

$$\hat{P}(k+1|k+1) = \hat{P}(k+1|k)F_x(k)^T - K(k+1)H_x(k+1)\hat{P}(k+1|k) \quad (2.13)$$

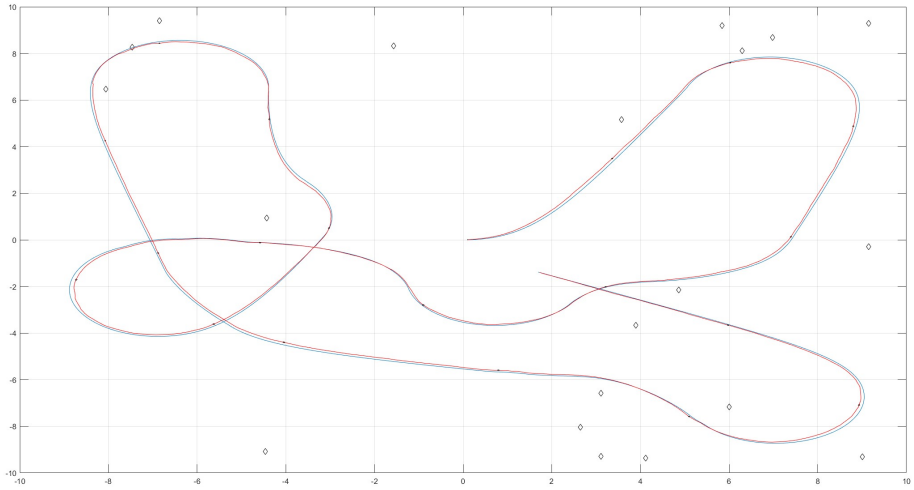
Οι παραπάνω σχέσεις είναι οι εξισώσεις ανανέωσης του Kalman filter. Παίρνουν τις εκτιμώμενες τιμές της επόμενης χρονικής στιγμής k+1|k και προσθέτουν τις πληροφορίες από την χρονική στιγμή k+1 για να υπολογίσουν τις τιμές της k+1|k+1. Η innovation έχει προστεθεί στην εκτιμώμενη κατάσταση μετά το πολλαπλασιασμό με το Kalman gain (K) το οποίο ορίζεται ως

$$S(k+1) = H_x(k+1)\hat{P}(k+1|k)H_x(k+1)^T + H_w(k+1)\widehat{W}(k+1)H_w(k+1)^T \quad (2.14)$$

$$K(k+1) = \hat{P}(k+1|k)H_x(k+1)^T S(k+1)^{-1} \quad (2.15)$$

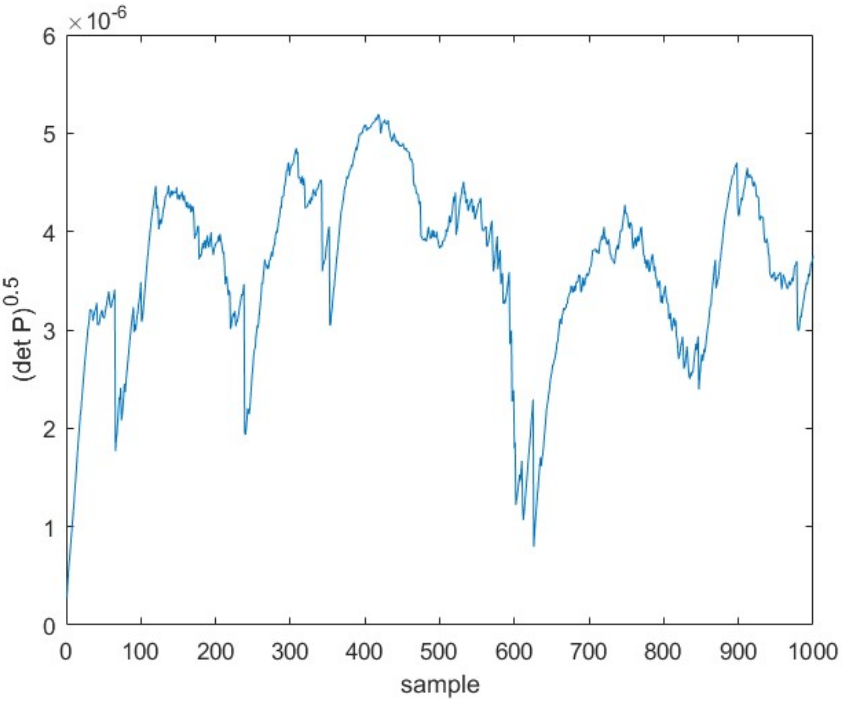
Όπου $\widehat{W}$ είναι η εκτιμώμενη κατανομή θορύβου του αισθητήρα.

Όπως και στο προηγούμενο κεφάλαιο το παρακάτω γράφημα μας δείχνει την πραγματική τροχιά (μπλε) του οχήματος, την εκτιμώμενη τροχιά (κόκκινο) και αυτή την φορά τα landmarks του χάρτη (ρόμβοι).



Εικόνα 2.3 – Εντοπισμός θέσης με χρήση EKF, οδομετρίας και landmarks.

Η εμφανέστερη διαφορά φαίνεται στο γράφημα της αβεβαιότητας η οποία με την βοήθεια πλέον του χάρτη σταματάει να αυξάνεται συνεχώς αλλά παραμένει μέσα σε ένα σταθερό εύρος τιμών.



Εικόνα 2.4 – Μεταβολή αβεβαιότητας (με χρήση χάρτη).

Κώδικας παραδείγματος:

```
% Δημιουργία χάρτη με 20 σημεία αναφοράς (landmarks).
map = Map(20);

% Διασπορά σφάλματος μετρήσεων αισθητήρα.
W = diag([0.1, 1*pi/180].^2);

%Αρχικοποίηση οχήματος. (Ιδιο με το προηγούμενο παράδειγμα.)
V = diag([0.005, 0.5*pi/180].^2);
veh = Vehicle(V);
veh.add_driver(RandomPath(map.dim));

% Δημιουργία του αισθητήρα.
sensor = RangeBearingSensor(veh, map, W);

% Αρχική διασπορά σφάλματος αρχικής θέσης. Υποθέτουμε ότι γνωρίζουμε αρκετά καλά
που βρισκόμαστε οπότε η τιμή της είναι μικρή.
P0 = diag([0.005, 0.005, 0.001].^2);

% Δημιουργία και εξομοίωση του EKF.
ekf = EKF(veh, V, P0, sensor, W, map);
ekf.run(1000);

% Γράφημα του χάρτη. (Εικόνα 2.3)
map.plot();
hold on;

% Γράφημα της κίνησης του οχήματος. (Εικόνα 2.3)
veh.plot_xy();
hold on;

% Γράφημα της εκτιμώμενης θέσης από το EKF. (Εικόνα 2.3)
ekf.plot_xy('r');
hold on;

% Γράφημα της αβεβαιότητας του EKF για κάθε θέση (x,y) με μορφή ελλείψεων. (Εικόνα
2.3)
ekf.plot_ellipse([], 'g');
hold off;

% Γράφημα της συνολικής αβεβαιότητας (x,y,θ). (Εικόνα 2.4)
figure;
ekf.plot_P();
```

Κεφάλαιο 3 - Ταυτόχρονος εντοπισμός θέσης και χαρτογράφηση

Ένα πρόβλημα των αυτόνομων ρομποτικών συστημάτων είναι η χαρτογράφηση και πλοήγηση άγνωστων χώρων. Η λύση στο πρόβλημα αυτό είναι το Simultaneous localization and mapping ή εν συντομία SLAM. Το SLAM είναι η διαδικασία χαρτογράφησης ενός χώρου ενώ ταυτόχρονα γίνεται ο υπολογισμός της θέσης του ρομπότ.

Η λογική ενός αλγορίθμου Slam είναι η εξής. Αρχικά το ρομπότ ψάχνει γύρω του ώστε να ανακαλύψει έναν αριθμό από σημεία αναφοράς (landmarks), τα οποία έπειτα χρησιμοποιεί για να καθορίσει τη θέση του μέσα στο χώρο. Στη συνέχεια, κινείται μέσα στο χώρο ανακαλύπτοντας περισσότερα landmarks, δημιουργώντας με αυτό το τρόπο έναν χάρτη. Το μεγαλύτερο πρόβλημα σε αυτή τη διαδικασία είναι ο θόρυβος των μετρήσεων, δηλαδή κατά πόσο οι μετρήσεις του ρομποτικού συστήματος αντιπροσωπεύουν την πραγματικότητα.

Η υλοποίηση του μπορεί να γίνει με διάφορους τρόπους αναλόγως τις ανάγκες τις εφαρμογής και τις δυνατότητες του ρομποτικού συστήματος. Οι πιο συνήθεις τρόποι, για την ανάλυση των δεδομένων από τους διάφορους αισθητήρες, είναι τα Kalman Filters (KF) και τα Particle Filters (PF) ή αλλιώς Monte Carlo localization (MCL). Για την χαρτογράφηση γίνεται συνήθως χρήση των grid maps ή των landmarks.

3.1 Μαθηματικό μοντέλο του SLAM

Έστω ένα κινούμενο ρομπότ σε έναν χώρο, το οποίο παρατηρεί σημεία αναφοράς (landmarks) με τη βοήθεια κάποιου αισθητήρα (laser scanner). Έχουμε, λοιπόν, τις εξής πληροφορίες σε μία χρονική στιγμή t:

x_t : Διάνυσμα κατάστασης. Έχει πληροφορίες για την θέση και τον προσανατολισμό του ρομπότ (αγγλικός όρος – pose).

u_t : Διάνυσμα ελέγχου. Εφαρμόζεται τη στιγμή k-1 για την μετάβαση του ρομπότ από την κατάσταση k-1 στην κατάσταση k.

m_i : Οι θέσεις των landmarks.

z_{it} : Η παρατήρηση των θέσεων των landmarks τη στιγμή k.

Δημιουργούνται τα εξής σύνολα των πληροφοριών μέχρι την χρονική στιγμή t:

$X_{0:t} = \{x_0, x_1, \ldots, x_\kappa\} = \{X_{0:\kappa-1}, x_\kappa\}$: Σύνολο όλων των poses.

$U_{0:t} = \{u_1, u_2, \ldots, u_\kappa\} = \{U_{0:\kappa-1}, u_\kappa\}$: Σύνολο όλων των εισόδων ελέγχου.

$M = \{m_1, m_2, \ldots, m_n\}$: Σύνολο όλων των landmarks.

$Z_{0:t} = \{z_1, u_2, \ldots, z_\kappa\} = \{Z_{0:\kappa-1}, z_\kappa\}$: Σύνολο όλων των παρατηρήσεων.

Το σύνολο $X_{0:t}$ αποτελείται από όλες τις θέσεις του ρομπότ, σκιαγραφώντας την διαδρομή που ακολούθησε. Το σύνολο $U_{0:t}$ είναι η οδομετρία (odometry). Είναι η κίνηση που προκάλεσε τη μετάβαση από την μία κατάσταση στην άλλη. Η οδομετρία υπολογίζεται συνήθως από τους encoders των τροχών (wheel odometry), την σχετική κίνηση μίας κάμερας (visual odometry), ή από τις εντολές ελέγχου προς τους κινητήρες.

3.2 Πιθανολογικό SLAM

Οι μετρήσεις που χρησιμοποιούνται για την λύση του SLAM δεν είναι ποτέ τέλειες και δεν έχουν ποτέ απόλυτη ακρίβεια αλλά περιέχουν αβεβαιότητες. Για τον λόγο αυτό περιγράφονται από κατανομές πιθανοτήτων. Έτσι, το πρόβλημα του SLAM έγκειται στον υπολογισμό της εκ των υστέρων (posterior) πιθανότητας :

$$P(X_{0:t}, m \mid Z_{0:t}, U_{0:t}) \qquad (1)$$

Ο ορισμός αυτός λέγεται full SLAM. Καταγράφει το χάρτη του περιβάλλοντος με τις πραγματικές θέσεις των landmarks και όλα τα poses που πραγματοποίησε το ρομπότ. Άλλος ένας τρόπος που χρησιμοποιείται από πολλούς αλγορίθμους είναι το online SLAM που υπολογίζει μόνο το παρόν pose και ορίζεται από την πιθανότητα:

$$P(x_t, m \mid Z_{0:t}, U_{0:t}, x_0) \qquad (2)$$

Αυτή η πιθανολογική κατανομή περιγράφει την συνολική μεταγενέστερη κατάσταση των θέσεων των landmarks και της κατάστασης του ρομπότ (για χρονική στιγμή t) με βάση τις προηγούμενες καταγεγραμμένες μετρήσεις και εισόδους ελέγχου μέχρι και την χρονική στιγμή t μαζί με την αρχική κατάσταση του ρομπότ. Γενικά, μια αναδρομική λύση του προβλήματος SLAM είναι επιθυμητή. Ξεκινώντας με μία εκτίμηση της κατανομής

$$P(x_{t-1}, m \mid Z_{0:t-1}, U_{0:t-1})$$

σε χρόνο t-1 και χρησιμοποιώντας την είσοδο ελέγχου u_k και την παρατήρηση z_t, υπολογίζεται η μεταγενέστερη κατάσταση με τη χρήση της θεωρίας του Bayes. Αυτός ο υπολογισμός απαιτεί το καθορισμό μοντέλων της αλλαγής κατάστασης και της παρατήρησης, τα οποία θα περιγράφουν την επίδραση των εισόδων ελέγχου και των παρατηρήσεων αντίστοιχα.

Το μοντέλο παρατήρησης περιγράφει την πιθανότητα πραγματοποίησης μιας παρατήρησης z_t, όταν οι θέσεις του ρομπότ και των landmarks είναι γνωστές.

$$P(z_t \mid x_t, m). \qquad (3)$$

Υποθέτουμε πως όταν η θέση του ρομπότ και ο χάρτης καθοριστούν, οι παρατηρήσεις είναι ανεξάρτητες με βάση τον χάρτη και τη κατάσταση του ρομπότ.

Το μοντέλο κίνησης του ρομπότ περιγράφεται με την μορφή

$$P(x_t \mid x_{t-1}, u_t) \qquad (4)$$

Υπό την προϋπόθεση ότι η αλλαγή κατάστασης είναι μια διαδικασία Markov στην οποία η επόμενη κατάσταση x_t εξαρτάται μόνο από την αμέσως προηγούμενη κατάσταση x_{t-1} και την είσοδο ελέγχου u_t και είναι ανεξάρτητη από τις παρατηρήσεις και τον χάρτη.

Ο αλγόριθμος SLAM μπορεί πλέον να αναπαρασταθεί με τα εξής 2 βήματα:

1. Πρόβλεψη (prediction)(Markov)

$$P(x_t, m \mid Z_{0:t-1}, U_{0:t}, x_0) = \int P(x_t \mid x_{t-1}, u_t)\, P(x_{t-1}, m \mid Z_{0:t-1}, U_{0:t-1}, x_0) dx_{t-1} \qquad (5)$$

2. Διόρθωση (correction)(Bayes)

$$P(x_t, m \mid Z_{0:t}, U_{0:t}, x_0) = P(z_t \mid x_t, m) P(x_t, m \mid Z_{0:t-1}, U_{0:t}, x_0) / P(z_t \mid Z_{0:t-1}, U_{0:t}) \qquad (6)$$

3.3 Χαρτογράφηση

Εκτός από την εύρεση θέσης του ρομπότ η μέθοδος SLAM ασχολείται και με την χαρτογράφηση του περιβάλλοντος.

Οι μετρικοί χάρτες αναπαριστούν το περιβάλλον με όρους γεωμετρικών σχέσεων μεταξύ των αντικειμένων και ενός σταθερού συστήματος αναφοράς. Χωρίζονται σε δύο κατηγορίες, τους χάρτες χαρακτηριστικών σημείων και τα πλέγματα πληρότητας.

Οι χάρτες χαρακτηριστικών σημείων (feature maps) χρησιμοποιούν οπτική οδομετρία για την αναπαράσταση του περιβάλλοντος μέσω των γεωμετρικών ιδιοτήτων των ανιχνευμένων χαρακτηριστικών σημείων. Το μεγαλύτερο πλεονέκτημά τους είναι η μικρή κατανάλωση υπολογιστικών πόρων, καθώς οι χάρτες είναι συνήθως αραιοί (sparse), με την ανίχνευση των σημείων να έχει γίνει ήδη στο βήμα της οδομετρίας. Μειονέκτημα, αποτελεί η επιρρέπεια σε λανθασμένες συσχετίσεις δεδομένων (false data association) όπου η παρατήρηση αντιστοιχίζεται λάθος σε ένα ήδη καταχωρημένο feature.

Οι χάρτες πλεγμάτων πληρότητας (occupancy grids), σε αντίθεση με τους χάρτες χαρακτηριστικών σημείων, ασχολούνται με το αν ο χώρος είναι κατειλημμένος ή όχι. Με αυτή τη μέθοδο ο χώρος χωρίζεται σε ένα πλέγμα και αναπαρίσταται από έναν πίνακα, του οποίου οι τιμές δείχνουν τη πιθανότητα αν η αντίστοιχη περιοχή να είναι κατειλημμένη. Κάθε στοιχείο του πίνακα παίρνει τιμές (0,1,-1) αναλόγως με την κατάσταση της περιοχής που περιγράφει. Συνήθως αυτοί οι χάρτες προβάλλουν το χώρο σε δισδιάστατο επίπεδο x,y (2D occupancy grid). Το μεγαλύτερο πλεονέκτημά τους είναι η άμεση συσχέτιση και η εύκολη χρήση τους σε προβλήματα σχεδιασμού διαδρομής (path planning) και εξερεύνησης (exploration) χώρου. Μειονέκτημά τους είναι η πολυπλοκότητα του υπολογισμού τους, η οποία αυξάνεται ακόμα περισσότερο σε μεγάλους χώρους.

Κεφάλαιο 4 – Χρήση φίλτρων για SLAM

Βασική ανάγκη για την λύση του προβλήματος του πιθανολογικού SLAM είναι η σωστή αναπαράσταση των μοντέλων παρατήρησης (3) και κίνησης (4), η οποία επιτρέπει τον υπολογισμό των εξισώσεων (5, 6) με μεγαλύτερη αποτελεσματικότητα και συνοχή. Η πιο διαδεδομένη αναπαράσταση έχει την μορφή ενός μοντέλου χώρου κατάστασης μαζί με Gaussian θόρυβο, οδηγώντας στην χρήση του Extended Kalman Filter (EKF). Ακόμη, μία διαφορετική προσέγγιση είναι η αναπαράσταση του μοντέλου κίνησης (4) ως μία ομάδα μετρήσεων χωρίς Gaussian κατανομή πιθανοτήτων. Αυτό οδήγησε στην χρήση του Rao-Blackwellised particle filter.

4.1 Extended Kalman Filter

Η χρήση του φίλτρου Kalman είναι ο πρώτος ιστορικά τρόπος λύσης του προβλήματος SLAM και έγινε πηγή έμπνευσης για τους επόμενους. Το Extended Kalman Filter προτάθηκε το 1990 από τους Smith, Self και Cheesman.

Όπως αναφέρθηκε και παραπάνω η βασική υπόθεση του EKF είναι πως ο θόρυβος των μετρήσεων από τους αισθητήρες περιγράφεται από μια Gaussian κατανομή, πιο συγκεκριμένα Gaussian θόρυβο μηδενικής μέσης τιμής και ασυσχέτιστων τιμών (zero-mean uncorrelated Gaussian noise).

Το μοντέλο κίνησης περιγράφεται από την σχέση:

$$P(x_t \mid x_{t-1}, u_t) \Longleftrightarrow x_t = f(x_{t-1}, u_t) + w_t \qquad (4.1)$$

Όπου το f() είναι το μοντέλο κινηματικής του ρομπότ και το w_k είναι η Gaussian κατανομή θορύβου με μέση τιμή μηδέν και συνδιασπορά Q_t.

Το μοντέλο παρατήρησης περιγράφεται από τη ν σχέση:

$$P(z_t \mid x_t, m) \Longleftrightarrow z(t) = h(x_t, m) + v_t \qquad (4.2)$$

Όπου το h() περιγράφει την γεωμετρία των παρατηρήσεων και το v_t είναι η Gaussian κατανομή θορύβου με μέση τιμή μηδέν και συνδιασπορά R_t.

Σημαντικό για την μέθοδο EKF SLAM είναι η χρήση ενός επαυξημένου διανύσματος κατάστασης για την αναπαράσταση του pose του ρομπότ μαζί με τις θέσεις των landmarks,

$$y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} = \begin{pmatrix} x\ y\ \theta\ m_{1,x}\ m_{1,y}\ m_{2,x}\ m_{2,y}\ \dots\ m_{N,x}\ m_{N,y} \end{pmatrix} \quad (4.3)$$

όπου x, y, θ αποτελούν το pose του ρομπότ και $m_{\iota,x}\ m_{\iota,y}$ οι συντεταγμένες των landmarks στο χώρο. Το επαυξημένο διάνυσμα έχει διαστάσεις 3+2N.

Με την βοήθεια της σχέσης (2.1) υπολογίζεται η μέση τιμή μ. Ο πίνακας συνδιασποράς $\Sigma_{(3+2N)(3+2N)}$ του επαυξημένου διανύσματος υπολογίζεται με την παρακάτω διαδικασία πρόβλεψης και διόρθωσης.

Αρχικά, το ρομπότ βρίσκεται στην αρχική του κατάσταση στην αρχή του συστήματος συντεταγμένων χωρίς να παρατηρεί κανένα landmark. Οπότε έχουμε αρχικά την παρακάτω μέση τιμή και πίνακα συνδιασποράς:

$$\mu_0 = (0\ 0\ 0\ \dots\ 0)^T \quad (4.4)$$

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & \infty & \dots & \infty \\ 0 & 0 & 0 & \infty & \dots & \infty \\ 0 & 0 & 0 & \infty & \dots & \infty \\ \infty & \infty & \infty & \infty & \dots & \infty \\ \dots & \dots & \dots & \dots & \dots & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix} = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xm} \\ \Sigma_{mx} & \Sigma_{mm} \end{pmatrix} \quad (4.5)$$

Το ρομπότ κινείται σύμφωνα με το μοντέλο ταχυτήτων στο χώρο κατάστασης:

$$y_t = y_{t-1} + \begin{pmatrix} -\frac{u_t}{\omega_t} sin\theta + \frac{u_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{u_t}{\omega_t} cos\theta - \frac{u_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t + \gamma_t \Delta t \\ 0 \\ \dots \\ 0 \end{pmatrix} \quad (4.6)$$

Στην παραπάνω σχέση εισάγεται ο Gaussian θόρυβος, και αφού το μοντέλο κίνησης δεν επηρεάζει τη θέση των landmarks, γράφεται ως:

$$y_t = y_{t-1} + F_x^T \begin{pmatrix} -\frac{u_t}{\omega_t} sin\theta + \frac{u_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{u_t}{\omega_t} cos\theta - \frac{u_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t + \gamma_t \Delta t \end{pmatrix} + N(0, F_x^T R_t F_x) \qquad (4.7)$$

όπου

$$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \end{pmatrix}_{3\times(3+2N)} \qquad (4.8)$$

Στο EKF το μοντέλο κίνησης προσεγγίζεται από ένα ανάπτυγμα Taylor πρώτης τάξης:

$$g(u_t, y_{t-1}) \approx g(u_t, \mu_{t-1}) + G(y_{t-1} - \mu_{t-1}) \qquad (4.9)$$

Όπου η G_t είναι η Jacobian της g στο u_t μετασχηματισμένη σε (3+2N)×(3+2N) διαστάσεις:

$$G_t = g'(u_t, \mu_{t-1}) = I + F_x^T g_t F_x \qquad (4.10)$$

όπου

$$g_t = \begin{pmatrix} 0 & 0 & \frac{u_t}{\omega_t}\cos(\mu_{t-1,\theta}) - \frac{u_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & \frac{u_t}{\omega_t} sin(\mu_{t-1,\theta}) - \frac{u_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} \qquad (4.11)$$

Οι παραπάνω σχέσεις αποτελούν το μέρος πρόβλεψης του EKF SLAM (4.6-4.11)

και καταλήγουν στην εκτίμηση (μέσης τιμής και συνδιασποράς) του pose μετά την εφαρμογή του ελέγχου u_t:

$$\bar{\mu}_t = g(u_t, \mu_{t-1}) \qquad (4.12)$$

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \qquad (4.13)$$

Στη συνέχεια είναι η παρατήρηση των landmarks. Αν οι αντιστοιχίες των παρατηρήσεων με τα landmarks είναι γνωστές τότε υπάρχει και μια τρίτη μεταβλητή, η υπογραφή (signature) s. Στην περίπτωση των άγνωστων αντιστοιχιών εκτελείται μια διαδικασία εκλογής, βάζοντας στην αρχή το παρατηρηθέν landmark με νέο δείκτη N+1 (όπου N ο αριθμός των προηγούμενων landmarks που έχουν βρεθεί) και ελέγχοντας αν η απόσταση του από τα υπόλοιπα N landmarks είναι εντός του ορίου ανοχής a. Το όριο αυτό ορίζεται ως:

$$D_M = \sqrt{(x-y)^T C^{-1} (x-y)} \qquad (4.14)$$

Όπου C είναι ο πίνακας συνδιασποράς των ανεξάρτητων μεταβλητών x, y. Σε περίπτωση που δεν είναι καινούριο landmark, επιλέγεται η αντιστοιχία με τη μεγαλύτερη πιθανότητα.

Επειδή στη πλειοψηφία των περιπτώσεων χρησιμοποιούνται μετρήσεις από αισθητήρες απόστασης (laser), οι παρατηρήσεις ορίζονται με βάση την απόστασή τους από τον αισθητήρα, r, και τη bearing γωνία μεταξύ των δύο, φ. Με δεδομένο τα παραπάνω, για το μοντέλο κίνησης (4.2) μπορεί να εφαρμοστεί το μοντέλο:

$$z_i^t = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - x)^2)} \\ atan2\big(m_{j,y} - y, m_{j,x} - x\big) - \theta \\ m_{j,s} \end{pmatrix} + N(0, \begin{pmatrix} \sigma_T & 0 & 0 \\ 0 & \sigma_\varphi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}) \qquad (4.15)$$

Δηλαδή για:

$$h(y_t, j) = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - x)^2)} \\ atan2\big(m_{j,y} - y, m_{j,x} - x\big) - \theta \\ m_{j,s} \end{pmatrix} \qquad (4.16)$$

ο θόρυβος έχει την συνδιασπορά Q_t. Όπως κάναμε και για το μοντέλο κίνησης, η h αναπτύσσεται με Taylor στο σημείο $\bar{\mu}_t$:

$$h(y_t, j) \approx h(\bar{\mu}_t, j) + H_t^i (y_t - \bar{\mu}_t) = h(\bar{\mu}_t, j) + h_i^t F_{x,j} \qquad (4.17)$$

όπου h_i^t είναι η Jacobian της h ως προς x_t και m_j στο $\bar{\mu}_t$ την οποία ο πίνακας $F_{x,j}$ μετασχηματίζεται σε (3N+3)×3 διαστάσεις για την εισαγωγή του στο μοντέλο παρατήρησης:

$$h_t^i = \begin{pmatrix} \frac{m_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q_t}} & \frac{y_t-\bar{\mu}_{t,y}}{\sqrt{q_t}} & 0 & \frac{\bar{\mu}_{t,x}-m_{j,x}}{\sqrt{q_t}} & \frac{\bar{\mu}_{t,y}-y_t}{\sqrt{q_t}} & 0 \\ \frac{\bar{\mu}_{t,y}-y_t}{q_t} & \frac{m_{j,x}-\bar{\mu}_{t,x}}{q_t} & -1 & \frac{y_t-\bar{\mu}_{t,y}}{q_t} & \frac{\bar{\mu}_{t,x}-m_{j,x}}{q_t} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4.18)$$

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{pmatrix} \qquad (4.19)$$

Όπου

$$q_t = (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2 \qquad (4.20)$$

Το landmark που δεν έχει βρεθεί στο παρελθόν, δεν αρχικοποιείται στο σημείο $(0\ 0\ 0)^T$, όπως ορίζει η παραπάνω γραμμηκοποίηση αλλά του δίνεται η αναμενόμενη θέση του από την παρούσα εκτίμηση για το pose και τη μέτρηση του αισθητήρα:

$$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r^i \begin{pmatrix} \cos(\varphi_i^t + \bar{\mu}_{t,\theta}) \\ \sin(\varphi_i^t + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix} \qquad (4.21)$$

Τέλος, με κάθε νέα παρατήρηση γίνεται διόρθωση της πρόβλεψης του μοντέλου κίνησης για το pose. Ο βαθμός με τον οποίο θα επηρεάσει η νέα παρατήρηση το pose ορίζεται από το κέρδος Kalman (Kalman Gain).

$$K_t^i = \bar{\Sigma}_t H_t^{iT} (H_t^i \bar{\Sigma}_t H_t^{iT} + Q_t)^{-1} \qquad (4.22)$$

$$\mu_t = \bar{\mu}_\tau + \sum_i K_t^i (z_t^i - \hat{z}_t^i) \qquad (4.23)$$

$$\Sigma_t = (I - \sum_i K_t^i H_t^i)\bar{\Sigma}_t \qquad (4.24)$$

Το μεγαλύτερο μειονέκτημα του EKF SLAM είναι η εκθετική αύξηση του χρόνου υπολογισμού και της απαιτούμενης μνήμης όσο μεγαλώνει ο αριθμός των landmarks. Για τον λόγο αυτό δεν συνιστάται η χρήση του για την χαρτογράφηση μεγάλων χώρων.

4.2 Particle filter

Το 2002 προτάθηκε μία νέα λύση στο πρόβλημα του SLAM με όνομα FastSLAM από τον Michael Montemerlo. Ο αλγόριθμος βασίζεται στην ακολουθιακή δειγματοληψία Monte Carlo ή όπως αλλιώς είναι γνωστό particle filtering. Το κύριο πλεονέκτημά του είναι η δυνατότητα αντιμετώπισης περιπτώσεων στις οποίες το μοντέλο κίνησης δεν περιγράφεται από Gaussian κατανομές ή/και έχει μη γραμμικότητες στις οποίες δεν είναι δυνατό το ανάπτυγμα Taylor. Όσον αφορά το μοντέλο παρατήρησης συνεχίζει να γραμμικοποιείται καθώς δίνει ήδη μία καλή εκτίμηση.

Οι μεταβλητές σε αυτήν την μέθοδο όπως αναφέρθηκε και παραπάνω δεν περιγράφονται από Gaussian κατανομές αλλά από τυχαία δείγματα που προσεγγίζουν αυτές τις κατανομές, με μεγαλύτερη πυκνότητα κοντά στη μέση τιμή και αντίστοιχα μικρότερη μακριά της.

Η μεταγενέστερη (posterior) πιθανότητα της κατάστασης του ρομπότ δεν αναπαρίσταται πλέον από μία συνεχή κατανομή όπως στο EKF αλλά από ένα σύνολο δειγμάτων (particles). Κάθε δείγμα (particle) περιέχει την πιθανή κατάσταση του ρομπότ για μία χρονική στιγμή. Για την επιλογή των δειγμάτων από μία αυθαίρετη κατανομή που δεν μπορεί να γίνει μοντελοποίηση (στόχος) f, χρησιμοποιείται μία γνωστή κατανομή (πρόταση) g μέσω της δειγματοληψίας με σημαντικότητα (Importance Sampling Principle). Στα δείγματα της συνάρτησης (πρότασης) g δίνεται ένα βάρος σημαντικότητας (importance weight) με σκοπό η νέα δειγματοληψία να γίνεται από τη g.

Το βάρος του κάθε δείγματος δίνεται από τη σχέση:

$$w(x) = \frac{στόχος}{πρόταση} = \frac{f(x)}{g(x)}$$

Η μέθοδος αυτή βασίζεται σε μεγάλο βαθμό στο θεώρημα των Rao-Blackwell, με το οποίο παραγοντοποιεί τη ζητούμενη μεταγενέστερη πιθανότητα ως εξής:

$$p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) = p(x_{0:t} \mid z_{1:t}, u_{1:t}) p(m_{1:M} \mid x_{0:t}, z_{1:t})$$

Χρησιμοποιώντας λοιπόν το θεώρημα των Rao-Blackwell με την παραπάνω παραγοντοποίηση, ο αλγόριθμος εκτιμά πρώτα την διαδρομή που ακολούθησε το ρομπότ $p(x_{0:t} \mid z_{1:t}, u_{1:t})$ και έπειτα το χάρτη δεδομένων των poses $(m_{1:M} \mid x_{0:t}, z_{1:t})$. Αυτό γίνεται για κάθε δείγμα ξεχωριστά, θεωρώντας πως τα landmarks είναι ανεξάρτητα μεταξύ τους.

Παρακάτω βλέπουμε συνοπτικά τα βήματα που ακολουθεί ο αλγόριθμος για την λύση του προβλήματος.

1. Αρχικοποίηση των K αριθμό δειγμάτων (particles):

$$x_t^{[i]}, \mu_{t,1}^{[i]}, \mu_{t,2}^{[i]}, \dots, \mu_{t,N}^{[i]}, \Sigma_{t,1}^{[i]}, \Sigma_{t,2}^{[i]}, \dots, \Sigma_{t,N}^{[i]}$$

μαζί με το αρχικό pose και τις μηδενικές μέσες τιμές, συνδιασπορές για τα N landmarks.

2. Σε κάθε δείγμα υπολογίζεται μία προτεινόμενη συνάρτηση, με βάση την προϊστορία του και το μοντέλο κίνησης του ρομπότ.

$$x_t^{[i]} \sim \pi(x_t \mid X_{1:t-1}^{[i]}, Z_{1:k}, u_t)$$

3. Το βάρος για κάθε particle υπολογίζεται με βάση την προτεινόμενη συνάρτηση και με στόχο την μεταγενέστερη πιθανότητα του SLAM:

$$w_t^{[k]} = \frac{p(X_{1:t}^{[i]}, Z_{1:t}, U_{1:t})}{\pi(x_t \mid X_{1:t-1}^{[i]}, Z_{1:k}, u_t)} = \frac{p(X_{1:t}^{[i]}, Z_{1:t}, U_{1:t})}{p(x_t^{[i]} \mid x_{t-1}, u_t) p(X_{1:t-1}^{[i]}, Z_{1:t-1}, U_{1:t-1})}$$

Με την χρήση των θεωρημάτων Bayes και Rao-Blackwell το βάρος προκύπτει:

$$w^{[k]} = |2\pi Q|^{-1/2} \exp\{-\frac{1}{2}\left(z_t - \hat{z}^{[i]}\right)^T Q^{-1} \left(z_t - \hat{z}^{[i]}\right)\}$$

Όπου $\hat{z}$ η αναμενόμενη παρατήρηση και Q ο πίνακας συνδιασποράς της μέτρησης. Ο αλγόριθμος λοιπόν αυξάνει το βάρος των δειγμάτων που θεωρεί πιο αντιπροσωπευτικά με την πραγματικότητα.

4. Χρησιμοποιώντας τα καινούρια βάρη επιλέγεται μια ομάδα νέων δειγμάτων από τα ήδη υπάρχοντα με αντικατάσταση. Δηλαδή τα δείγματα με μεγάλο βάρος αντικαθιστούν αυτά με το μικρότερο.

5. Τέλος ενημερώνεται η θέση των landmarks σε κάθε δείγμα. Η μεταγενέστερη πιθανότητα που προκύπτει μετά την παραγοντοποίηση είναι:

$$P(X_{0:t}, m \mid Z_{1:t}, U_{1:t}) = P(X_{0:t} \mid Z_{1:t}, U_{1:t}) \prod_{k=1}^{N} P(m_i \mid X_{0:t}, Z_{1:t})$$

Η κατανομή για κάθε landmark μπορεί να εκτιμηθεί με τη χρήση ενός δισδιάστατου EKF εφόσον τα poses έχουν θεωρηθεί γνωστά. Για τον λόγο αυτό η πολυπλοκότητα του αλγορίθμου αυξάνεται γραμμικά με το μέγεθος του χάρτη. Με βάση λοιπόν την καλύτερη απόδοση του και την δυνατότητα αντιμετώπισης μη γραμμικών μοντέλων, χωρίς Gaussian κατανομές, θεωρείτε ως μία καλύτερη λύση σε σχέση με το EKF SLAM.

Κεφάλαιο 5 – Πλοήγηση

Η πλοήγηση ενός ρομποτικού συστήματος αφορά το πρόβλημα της καθοδήγησης του ρομπότ προς έναν στόχο. Ο πρώτος τρόπος λύσης που μας έρχεται στο μυαλό είναι η χρήση ενός χάρτη και πινακίδων καθώς αυτοί είναι και οι τρόποι που χρησιμοποιούμε και συναντάμε ως άνθρωποι στη καθημερινότητα μας. Τα ρομπότ από την άλλη είναι ικανά να προηγηθούν μέσα στο χώρο χωρίς τη χρήση κάποιου χάρτη χρησιμοποιώντας τεχνικές που αναφέρονται ως «πλοήγηση αντίδρασης» (reactive navigation). Για παράδειγμα ένα ρομπότ μπορεί να κινείται προς το φως, να ακολουθεί μία γραμμή στο πάτωμα, να κινείται μέσα σε έναν λαβύρινθο ακολουθώντας το τοίχο, ή να κάνει απλά τυχαίες κινήσεις. Το ρομπότ ανταποκρίνεται απευθείας με το περιβάλλον του, την ένταση του φωτός, τη θέση σε σχέση με την γραμμή στο πάτωμα ή την επαφή του με το τοίχο. Σήμερα πολλές από τις αυτόματες σκούπες κάνουν τυχαίες κινήσεις και καταλαβαίνουν μόνο ότι έχουν έρθει σε επαφή με κάποιο εμπόδιο.

Η χρήση χαρτών γίνεται σε πιο περίπλοκα ρομπότ. Οι τεχνικές αυτές υποστηρίζουν πιο σύνθετες εργασίες άλλα είναι και αυτές πιο σύνθετες. Απαιτούν έναν αριθμό προ απαιτούμενων, όπως έναν χάρτη του περιβάλλοντος και την θέση του ρομπότ μέσα σε αυτόν για κάθε χρονική στιγμή.

5.1 Πλοήγηση αντίδρασης

Τα συστήματα αντίδρασης (reactive systems) είναι μια σχετικά καινούρια εξέλιξη στο τομέα της ρομποτικής. Τα συστήματα αυτά βασίζονται στο συνδυασμό των αισθητήρων των μηχανισμών τους με μια ομάδα χαμηλού επιπέδου συμπεριφορών, οι οποίες σχεδιάζονται εξολοκλήρου ή εμπνέονται από την νευρονική και την γνωστική ψυχολογία. Το πλήθος τον ενεργειών έρχεται από την αλληλεπίδραση μεταξύ των διαφορετικών συμπεριφορών, τις ενδείξεις των αισθητήρων και το κόσμο. Τα συστήματα αντίδραση έχουν μεγαλύτερο ενδιαφέρον σε δυναμικά περιβάλλοντα τα οποία αλλάζουν συνεχώς και η χρήση ενός χάρτη δεν είναι εύκολη. Έχουν δοκιμαστεί με επιτυχία σε διάφορα ρομποτικά συστήματα, όπως ρομπότ με πολλά πόδια, ερπετά, πλατφόρμες εξωτερικού χώρου και σμήνος ρομπότ.

Μερικά από τα κυριότερα χαρακτηριστικά των ρομποτικών συστημάτων αντίδρασης περιλαμβάνου:

1. Οι συμπεριφορές είναι βασικά κομμάτια του σχεδιασμού.

Μία συμπεριφορά σε αυτά τα συστήματα είναι συνήθως ένας συνδυασμός του αισθητήρα και του κινητήρα, όπου οι μετρήσεις του αισθητήρα δίνουν τις απαραίτητες πληροφορίες για μια χαμηλού επιπέδου αντίδραση από τον κινητήρα, για την αποφυγή ενός εμποδίου ή την κατεύθυνση προς κάποιον στόχο.

2. Η γενική γνώση του περιβάλλοντος αποφεύγεται.

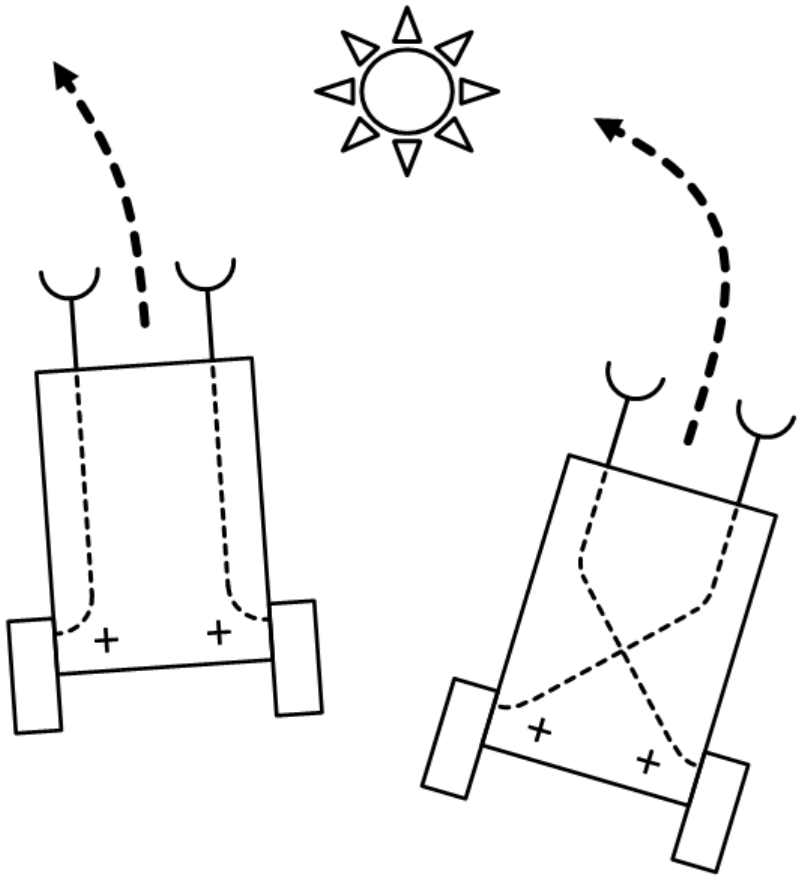
Η δημιουργία και η συντήρηση μίας περιγραφής του κόσμου είναι μια χρονοβόρα διαδικασία και είναι επιρρεπής σε λάθη. Τα συστήματα αντίδρασης δεν κρατάνε κάποιο μοντέλο για το κόσμο, αντιθέτως αντιδρούν σε ερεθίσματα του κόσμου. Αυτό είναι χρήσιμο για δυναμικούς και επικίνδυνους κόσμους, όπου το περιβάλλον είναι απρόβλεπτο.

3. Συνήθως τα συστήματα βασίζονται σε μοντέλα συμπεριφοράς ζώων.

Μοντέλα από νευρονική και την γνωστική ψυχολογία χρησιμοποιούνται για την αναπαράσταση συμπεριφορών που είναι απαραίτητες για την ασφαλή αλληλεπίδραση ενός ρομπότ με τον κόσμο.

5.1.1 Οχήματα Braitenberg

Τα οχήματα Braitenberg θεωρούνται πειράματα για την κατανόηση της κίνησης των φυτών και των ζώων προς ή μακριά από ένα ερέθισμα. Τα πιο ενδιαφέροντα οχήματα είναι συμμετρικές συσκευές με δύο εμπρόσθιους αισθητήρες μαζί με έναν ελεύθερο τροχό και δύο τροχούς πίσω συνδεδεμένους αντίστοιχα σε δύο μοτέρ. Το όχημα ελέγχεται από μία συνδεσμολογία η οποία συνδέει τον αριστερό αισθητήρα με το δεξιό μοτέρ και το δεξί αισθητήρα με το αριστερό μοτέρ. Όταν ενεργοποιείται ο αριστερός αισθητήρας, το δεξιό μοτέρ περιστρέφεται και το όχημα κινείται αριστερά. Αντιθέτως όταν ο δεξιός αισθητήρας ενεργοποιείται το αριστερό μοτέρ περιστρέφεται και το όχημα κινείται δεξιά. Η διαδικασία επαναλαμβάνεται μέχρι το όχημα να φτάσει στην πηγή του ερεθίσματος. Ένας άλλος τρόπος συνδεσμολογίας είναι οι αισθητήρες να συνδεθούν με το μοτέρ της ίδιας πλευράς (δεξιός αισθητήρας – δεξιό μοτέρ, αριστερός αισθητήρας – αριστερό μοτέρ), σε αυτή τη περίπτωση το όχημα θα απομακρύνεται από την πηγή του ερεθίσματος.



Εικόνα 5.1 – Οχήματα Braitenberg

–Πηγή: https://en.wikipedia.org/wiki/Braitenberg_vehicle (16/07/2018)

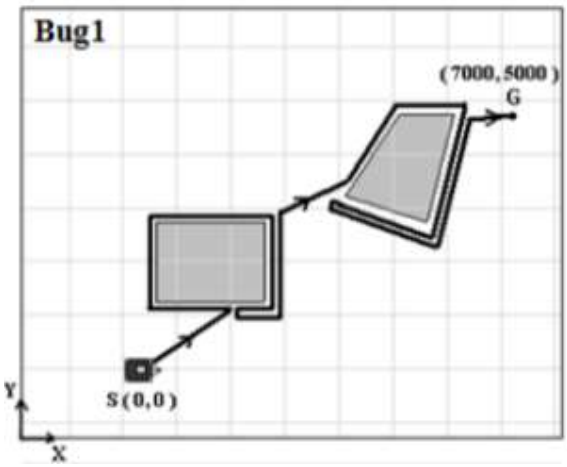
5.1.2 Bug

Οι αλγόριθμοι Bug είναι απλοί σχεδιαστές. Όταν συναντούν ένα άγνωστο εμπόδιο, μπορούν να δημιουργήσουν το δικό τους μονοπάτι γύρω από το αντικείμενο σε δισδιάστατο επίπεδο αν υπάρχει μια διαδρομή προς το στόχο. Σκοπός είναι η δημιουργία μίας διαδρομής χωρίς συγκρούσεις. Η οικογένεια των Bug έχει τις εξής υποθέσεις: ότι το ρομπότ είναι ένα σημείο, ότι έχει τέλειο εντοπισμό θέσης και ότι οι αισθητήρες του είναι ακριβείς. Λόγω των παραπάνω υποθέσεων η εφαρμογή τους σε πραγματικά ρομπότ και περιβάλλοντα είναι πολύ δύσκολη. Αντιθέτως σε περιβάλλον εξομοίωσης η εφαρμογή της είναι πολύ πιο εύκολη.

1. **Αλγόριθμοι Bug1 και Bug2**

Οι αλγόριθμοι Bug1 και Bug2, από τους Lumelsky και Stepanove, είναι από τους πρώτους αλγόριθμους σχεδίασης με χρήση αισθητήρων και προσφέρουν πολύ μικρή κατανάλωση μνήμης. Συνήθως βασίζονται σε αισθητήρες επαφής.

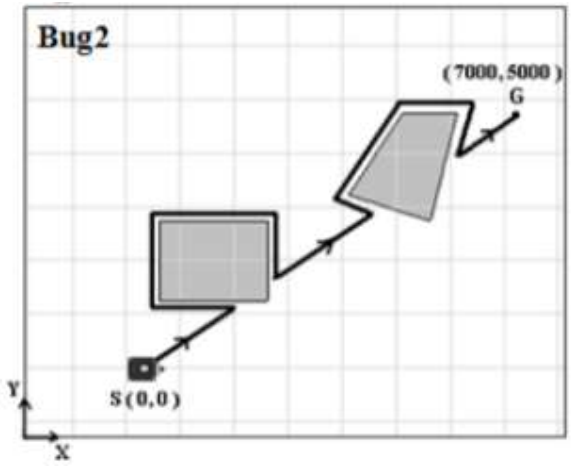
Ο αλγόριθμος Bug1 κινεί το ρομπότ σε μια ευθεία γραμμή από το σημείο S προς το σημείο G εκτός αν συναντήσει κάποιο εμπόδιο. Σε αυτή τη περίπτωση το ρομπότ αρχικά εξερευνά την περίμετρο του εμποδίου καλύπτοντας την όλη, κινούμενο με την φορά του ρολογιού. Έπειτα βρίσκει πιο σημείο της περιμέτρου είναι πιο κοντά προς τον στόχο G (σημείο αναχώρησης) και πηγαίνει προς αυτό συνεχίζοντας την πορεία του ή γυρνώντας προς τα πίσω αναλόγως ποια διαδρομή είναι συντομότερη. Όταν φτάσει στο σημείο αναχώρησης κινείται σε μία καινούρια ευθεία γραμμή προς το σημείο G. Αν συναντήσει κάποιο άλλο εμπόδιο η παραπάνω διαδικασία επαναλαμβάνεται. Αυτή η μέθοδος δεν είναι αποδοτική αλλά εγγυάται ότι το ρομπότ θα φτάσει στον στόχο.



Εικόνα 5.2 – Παράδειγμα αλγόριθμου Bug1

–Πηγή: "Performance comparison of bug navigation algorithms.", Βιβλιογραφία [16].

Στον αλγόριθμο Bug2 το ρομπότ προσπαθεί να ακολουθήσει την ευθεία γραμμή μεταξύ των σημείων S και G. Στην περίπτωση που συναντήσει κάποιο εμπόδιο τότε κινείται περιμετρικά από αυτό με την φορά του ρολογιού μέχρι να βρεθεί ξανά επάνω στην ευθεία όπου και την ξανά ακολουθεί. Σε αυτή την μέθοδο οι διαδρομές είναι συνήθως μικρότερες όμως υπάρχουν και περιπτώσεις που μπορεί να είναι μεγαλύτερες από τον Bug1.

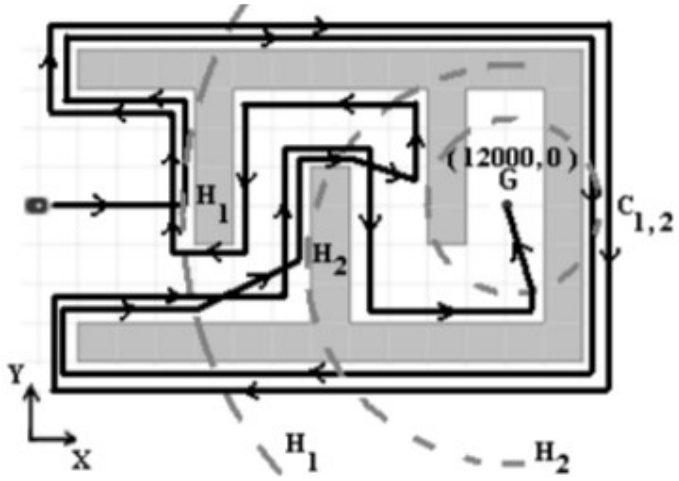


Εικόνα 5.3 – Παράδειγμα αλγόριθμου Bug2

–Πηγή: "Performance comparison of bug navigation algorithms.", Βιβλιογραφία [16].

2. Αλγόριθμος Class1

Ο αλγόριθμος Class1 από τον Noborio, ακολουθεί την περίμετρο του εμποδίου μέχρι η απόσταση προς τον στόχο G να είναι μεγαλύτερη από την μικρότερη απόσταση, της οποίας το σημείο C_i είναι αποθηκευμένο στο ρομπότ. Ο αλγόριθμος περιορίζει την περιοχή αναζήτησής του σε έναν κύκλο με κέντρο το σημείο G και ακτίνα την απόσταση μεταξύ G και C. Επίσης περιορίζει την περιοχή αναζήτησης χρησιμοποιώντας σημεία χτυπήματος H_i. Με την χρήση των σημείων C_i και H_i, ο αλγόριθμος δημιουργεί κυκλικούς βρόγχους για την περιοχή αναζήτησης, όπως φαίνεται στην παρακάτω εικόνα, με σκοπό να εγκαταλείψει το εμπόδιο προς το σημείο G περνώντας από τον εξωτερικό βρόγχο προς τον εσωτερικό.

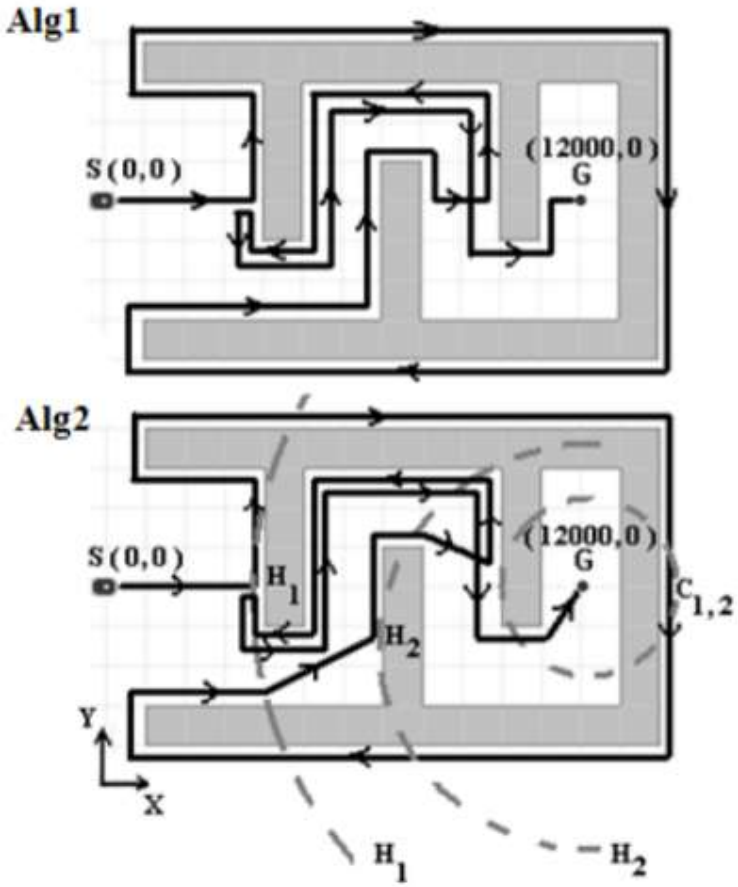


Εικόνα 5.4 – Παράδειγμα αλγόριθμου Class1

–Πηγή: "Performance comparison of bug navigation algorithms.", Βιβλιογραφία [16].

3. Αλγόριθμοι Alg1 και Alg2

Οι αλγόριθμοι Alg1 και Alg2 των Sankaranarayanan και Vidyasagar είναι αναθεωρημένες εκδοχές των Bug2 και Class1 αντίστοιχα. Και οι δύο στοχεύουν την σμίκρυνση της διαδρομής που πραγματοποιεί το ρομπότ. Η λειτουργία τους έχει ως εξής: αν το ρομπότ περάσει από ένα σημείο χτυπήματος $H_κ$, ή ένα σημείο εγκατάλειψης, L_k, που έχει ξανά συναντήσει κατά τη διάρκεια της κίνησής του στην περίμετρο, τότε συγκρίνει την απόσταση από το σημείοο H_k ή L_k με το τελευταίο σημείο χτυπήματος H_i ή το τελευταίο σημείο εγκατάλειψης L_i, με την φορά του ρολογιού και αντίθετα από την φορά του ρολογιού (k<i). Με βάση την μικρότερη απόσταση διαδρομής προς το H_i ή L_i προσδιορίζεται η κατεύθυνση με την οποία το ρομπότ θα ακολουθήσει την περίμετρο και θα κινηθεί προς το H_i ή L_i χωρίς να καταγράφει την απόσταση της διαδρομής. Η καταγραφή της απόστασης διαδρομής θα ξεκινήσει ξανά όταν το ρομπότ φτάσει στο H_i ή L_i. Τότε το ρομπότ αναζητά νέες εξωτερικές γραμμές του εμποδίου και συνεχίζει να ακολουθεί την περίμετρο μέχρι να βρει ένα καινούριο σημείο εγκατάλειψης προς το G.



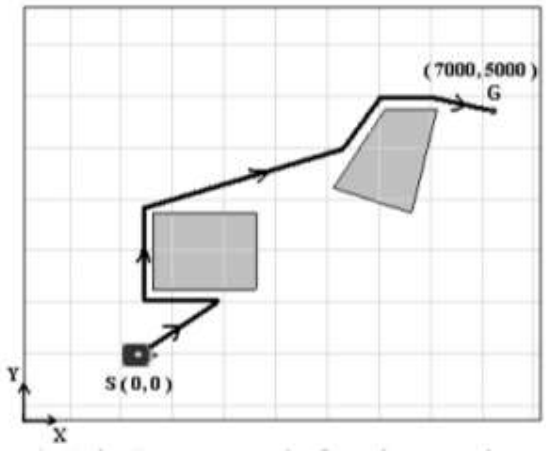
Εικόνα 5.5 – Παράδειγμα αλγόριθμων Alg1, Alg2.

–Πηγή: "Performance comparison of bug navigation algorithms.", Βιβλιογραφία [16].

4. Αλγόριθμος DistBug

Ο αλγόριθμος DistBug από τους Kamon και Rivlin προέρχονται από τους Alg1 και Alg2. Χρησιμοποιεί διαφορετικές δομές δεδομένων για την αποθήκευση των σημείων χτυπήματος και εγκατάλειψης μαζί με χρήσιμες πληροφορίες για την διαδρομή που ακολουθήθηκε.

Το ρομπότ έχει μια μέγιστη περιοχή ανίχνευσης με εμβέλεια R και έχει δύο βασικές συμπεριφορές την ακολουθία της περιμέτρου και την κίνηση προς τον στόχο. Αρχικά το ρομπότ κινείται προς το G μέχρι να συναντήσει κάποιο εμπόδιο. Τότε ακολουθεί την περίμετρο με την φορά του ρολογιού. Κατά τη διάρκεια της ακολουθίας το ρομπότ καταγράφει την ελάχιστη απόσταση $d_{min}(G)$ προς το σημείο G που έχει επιτευχθεί από το τελευταίο σημείο χτυπήματος. Επίσης το ρομπότ καταλαβαίνει την ελεύθερη απόσταση F μεταξύ της θέσης του X και του επόμενου εμποδίου που βρίσκεται ανάμεσα σε αυτό και τον στόχο G. Αν δεν υπάρχει εμπόδιο τότε η τιμή του F είναι R. Το ρομπότ εγκαταλείπει την περίμετρο του εμποδίου μόνο όταν η διαδρομή προς το G είναι ελεύθερη ή όταν ικανοποιείται η εξίσωση, $d(X, G) – F \leq d_{min}(G) – Step$, όπου d(X, G) είναι η απόσταση από το X προς το G και Step μια προκαθορισμένη σταθερά.



Εικόνα 5.6 – Παράδειγμα αλγόριθμου DistBug.

–Πηγή: "Performance comparison of bug navigation algorithms.", Βιβλιογραφία [16].

5.2 Πλοήγηση με χάρτη

Ο καλύτερος τρόπος εύρεσης μίας διαδρομής μεταξύ δύο σημείων είναι με την χρήση ενός χάρτη. Συνήθως αυτό σημαίνει η μικρότερη δυνατή διαδρομή ή η ευκολότερη διαδρομή με βάση τη δυσκολία του εδάφους. Ένας ακόμα πολυπλοκότερος σχεδιαστής μπορεί να λαμβάνει υπόψιν του την κινηματική και δυναμική ενός οχήματος ώστε να αποφεύγει διαδρομές με στροφές που δεν μπορεί να πραγματοποιήσει το όχημα.

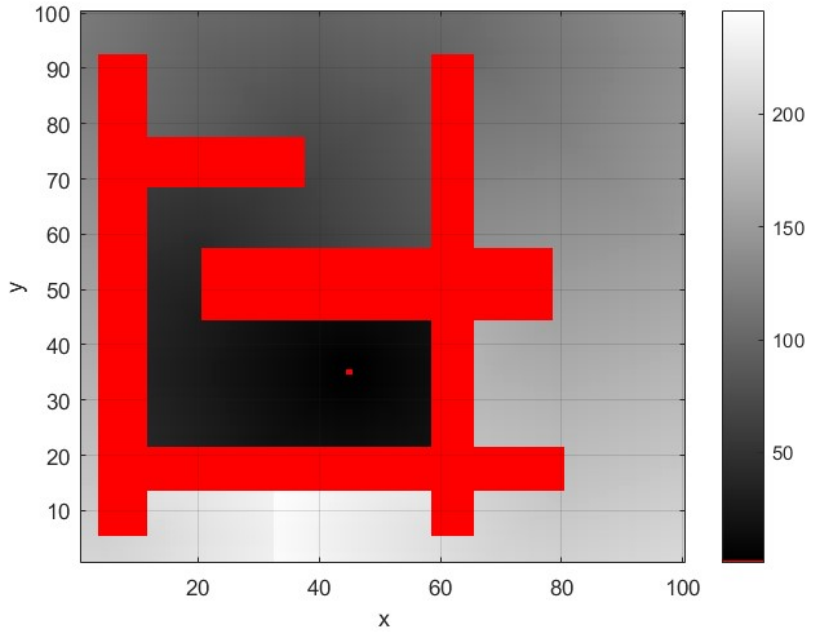
Υπάρχουν πολλοί τρόποι απόδοσης ενός χάρτη και της θέσης του οχήματος μέσα σε αυτόν. Μία προσέγγιση είναι η απόδοση της θέσης του οχήματος σε $(x,y)\in R^2$ και οι ελεύθερες περιοχές ή τα εμπόδια σε πολύγωνα, με το καθένα να έχει μια λίστα από σημεία ή άκρες. Με αυτή τη μέθοδο ο υπολογισμός των συγκρούσεων μεταξύ του ρομπότ και των εμποδίων θα πρέπει να γίνεται με δοκιμές με μεγάλες λίστες ακρών.

Μία πιο απλή απόδοση είναι το πλέγμα πληρότητας (occupancy grid). Όπως λέει και το όνομα του ο κόσμος χωρίζεται σε ένα πλέγμα από κελιά, με κάθε κελί να μαρκάρεται ως ελεύθερο ή κατειλημμένο. Με μηδέν έχουμε τα ελεύθερα κελιά στα οποία μπορεί να οδήγηση το ρομπότ και με ένα τα κατειλημμένα. Το μέγεθος των κελιών εξαρτάται από την εφαρμογή. Η μνήμη που απαιτείται για τις πληροφορίες του πλέγματος πληρότητας μεγαλώνει ανάλογα με την περιοχή κάλυψης και αντιθέτως ανάλογα με το μέγεθος των κελιών.

5.2.1 Distance Transform

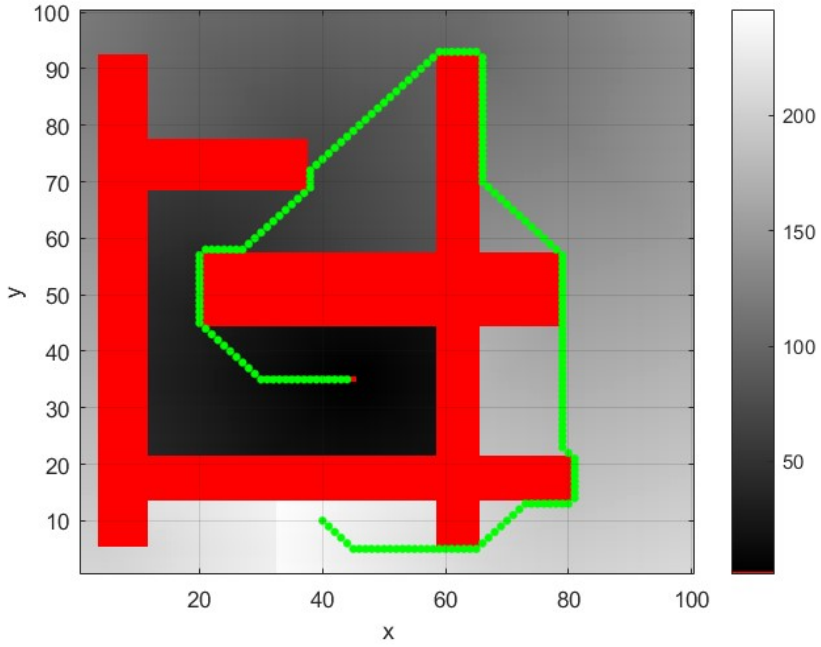
Ένας μετασχηματισμός απόστασης (distance transform) μετατρέπει μια δυαδική ψηφιακή εικόνα, αποτελούμενη από ένα χαρακτηριστικό και μη χαρακτηριστικά εικονοκύτταρα (pixels), σε μία εικόνα όπου όλα τα μη χαρακτηριστικά εικονοκύτταρα έχουν μια τιμή απόστασης από το κοντινότερο χαρακτηριστικό εικονοκύτταρο. Ο υπολογισμός των αποστάσεων κανονικά είναι μια καθολική (global) εργασία. Αυτές οι εργασίες είναι απαγορευτικά δαπανηρές. Για τον λόγο αυτό οι αλγόριθμοι που ψάχνουν μόνο μικρές ομάδες, αλλά συνεχίζουν να δίνουν καλές εκτιμήσεις της Ευκλείδειας απόστασης, είναι απαραίτητοι.

Έστω λοιπόν ότι έχουμε ένα ρομπότ με διαστάσεις ενός pixel, με αρχική θέση το σημείο (40,10) και στόχο το σημείο (45,35), και για χάρτη την παρακάτω εικόνα. Με κόκκινο χρώμα βλέπουμε τα εμπόδια και με την βαθμίδα μαύρο-άσπρο την απόσταση κάθε pixel από τον στόχο.



Εικόνα 5.7 – Χάρτης περιβάλλοντος μαζί με βάρος απόστασης distance transform.

Η εύρεση μίας διαδρομής είναι εύκολη. Ο αλγόριθμος διαλέγει σε κάθε βήμα το αμέσως επόμενο γειτονικό pixel το οποίο βρίσκεται πιο κοντά στο στόχο, έως ότου φτάσει στο στόχο.



Εικόνα 5.8 – Χάρτης περιβάλλοντος μαζί με βάρος απόστασης distance transform και διαδρομή μεταξύ start-goal.

Αυτή η μέθοδος λόγο του υψηλού κόστους δεν είναι πρακτική για μεγάλα πλέγματα πληρότητας.

Κώδικας παραδείγματος:

```
%% Καθορισμός αρχικής και τελικής θέσης.
goal = [45 35];
start = [40 10];

%% Είσαγωγή χάρτη.
load map1;

%% Δημιουργία distance transform για τον χάρτη.
dx = DXform(map);

%% Εμφάνιση του χάρτη μαζί με distance transform (Εικόνα 5.7)
dx.visualize();

%% Σχεδιασμός διαδρομής μεταξύ αρχικού και τελικού σημείου.
dx.plan(goal);
p = dx.path(start);

%% %% Εμφάνιση του χάρτη μαζί με distance transform και διαδρομή (Εικόνα 5.8)
dx.visualize (p);
```

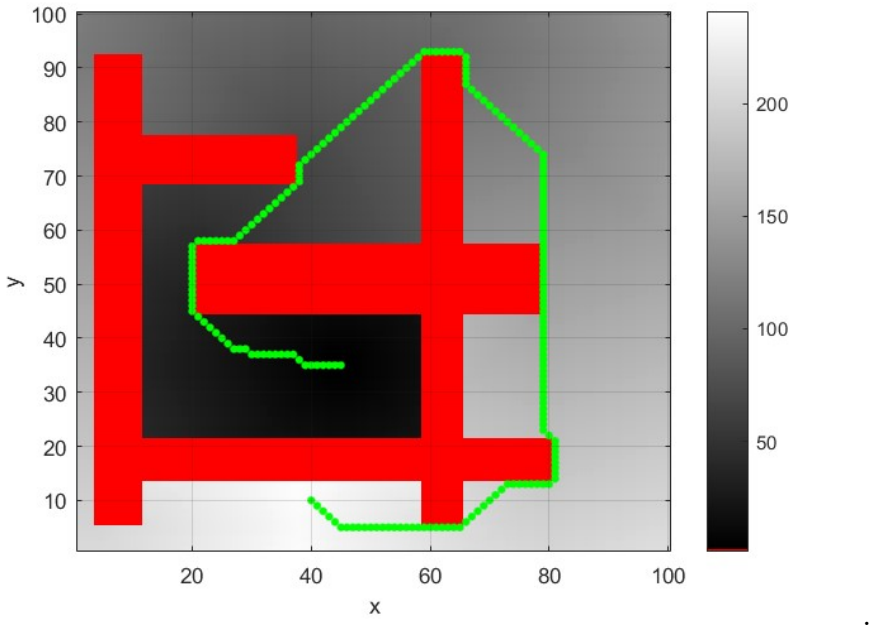
5.2.2 D*

Ο αλγόριθμο D* είναι πολύ δημοφιλής στην δημιουργία διαδρομών για ρομπότ, και έχει πολλά χαρακτηριστικά που είναι χρήσιμα σε πραγματικές εφαρμογές. Ο αλγόριθμος γενικεύει το πλέγμα πληρότητας σε έναν χάρτη κόστους το οποίο αποδίδει το κόστος $c \in R$, $c > 0$ της διάσχισης κάθε κελιού σε οριζόντια ή κάθετη κατεύθυνση. Το κόστος διάσχισης των κελιών διαγώνια είναι $c\sqrt{2}$. Για κελιά που αντιστοιχούν σε εμπόδιο έχουμε $c = \infty$.

Ο D* βρίσκει μία διαδρομή με το κατώτερο δυνατό κόστος. Στην περίπτωση που θέλουμε το λιγότερο δυνατό χρόνο άφιξης στον στόχο τότε το κόστος είναι ο χρόνος διάσχισης κάθε κελιού. Στην περίπτωση που θέλουμε την λιγότερη ζημιά στο όχημα ή μεγαλύτερη άνεση για τον επιβάτη, τότε το κόστος μπορεί να είναι η τραχύτητα εδάφους του κάθε κελιού. Το κόστος κάθε κελιού εξαρτάται και από τα χαρακτηριστικά και τις δυνατότητες του οχήματος.

Το κύριο χαρακτηριστικό του D* είναι η υποστήριξη της σταδιακής επανασχεδίασης. Αυτό είναι σημαντικό όταν, ενώ κινούμαστε, ανακαλύπτουμε ότι ο κόσμος είναι διαφορετικός από τον χάρτη. Αν βρούμε ότι μία διαδρομή έχει τελικά μεγαλύτερο κόστος από αυτό που περιμέναμε ή υπάρχει κάποιο εμπόδιο τότε μπορούμε σταδιακά να ξανά σχεδιάσουμε μια καλύτερη διαδρομή. Αυτή η σταδιακή επανασχεδίαση έχει μικρότερο υπολογιστικό κόστος από ότι να σχεδιάζαμε από την αρχή.

Έστω λοιπόν ότι έχουμε τις ίδιες συνθήκες όπως και στο παράδειγμα του distance transform. Μπορούμε αρχικά να δούμε την διαδρομή που σχεδιάστικε.

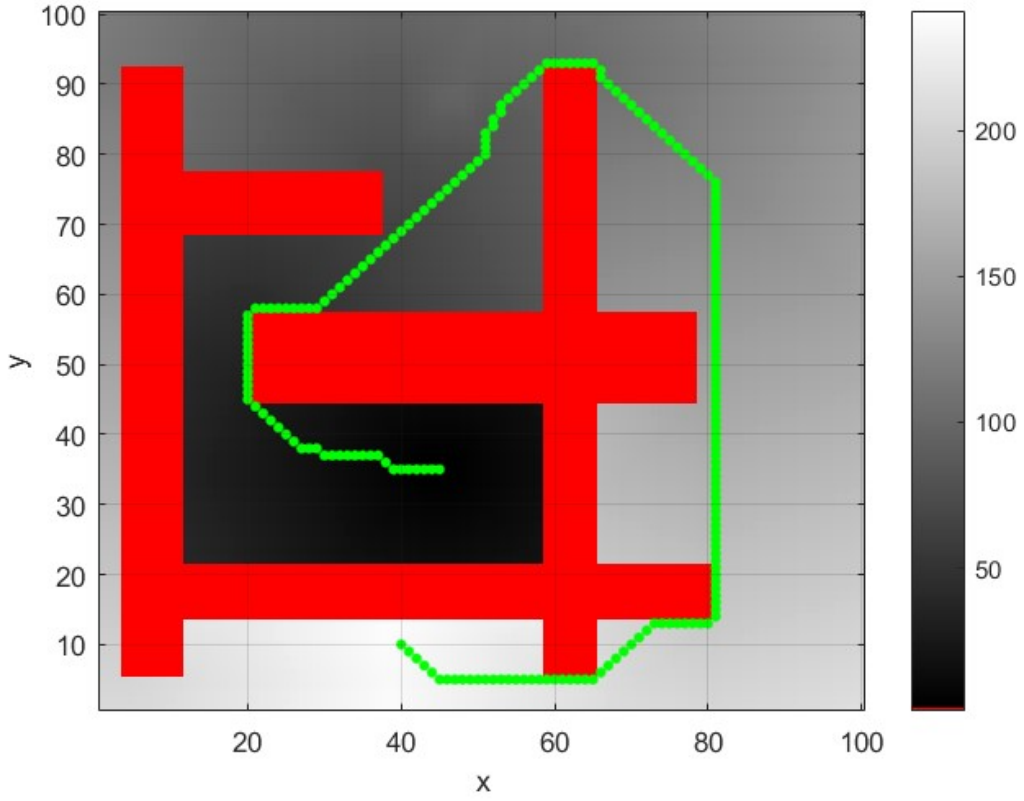


Εικόνα 5.9 – Χάρτης περιβάλλοντος μαζί με βάρος απόστασης D και διαδρομή μεταξύ start-goal.*

Παρατηρούμε το ίδιο αποτέλεσμα με τον distance transform καθώς το κόστος υπολογίστικε με βάση την μικρότερη δυνατή διαδρομή. Επίσης βλέπουμε ότι χρειάστηκαν 10556

προσπάθειες για την δημιουργία της διαδρομής. Εδώ βρίσκεται και η μεγάλη διαφορά και το πλεονέκτημα του D*.

Αν αλλάξουμε το κόστος στα σημεία x(45-60) και y(80-90) σε δύο, τότε η διαδρομή μας θα επανασχεδιαστεί όπως φαίνεται παρακάτω έχοντας χρησιμοποιήσει μόλις 4096 προσπάθειες.



Εικόνα 5.10 – Χάρτης περιβάλλοντος μαζί με αλλαγμένο βάρος D και νέα διαδρομή μεταξύ start-goal.*

Κώδικας παραδείγματος:

```
%% Καθορισμός αρχικής και τελικής θέσης.
goal = [45 35];
start = [40 10];

%% Είσαγωγή χάρτη.
load map1;

%% Δημιουργία D* για τον χάρτη.
ds = Dstar(map);

%% Σχεδιασμός διαδρομής μεταξύ αρχικού και τελικού σημείου.
ds.plan(goal);
```

```
%% Εμφάνιση χάρτη και διαδρομής. (Εικόνα 5.9)
ds.path(start);

%% Αριθμός προσπαθειών για την δημιουργία της διαδρομής. (10556)
ds.niter

%% Μεταβολή του βάρους στα σημεία του χάρτη y(80-90) και x(40-50)
for y=80:90
   for x=40:50
      ds.modify_cost([x,y], 2);
   end
end

%% Ανανέωση σχεδιασμού διαδρομής.
ds.plan();

%% Εμφάνιση χάρτη και ανανεωμένης διαδρομής. (Εικόνα 5.10)
ds.path(start);

%% Αριθμός προσπαθειών για την δημιουργία της διαδρομής. (4096)
ds.niter
```

5.2.3 Roadmaps

Στη σχεδίαση διαδρομών η δημιουργία ενός σχεδίου ονομάζεται φάση σχεδίασης. Η φάση αναζήτησης χρησιμοποιεί τα αποτελέσματα της φάσης σχεδίασης για να βρει μια διαδρομή μεταξύ δύο σημείων. Οι δύο προηγούμενες μέθοδοι, distance transform και D*, απαιτούν μεγάλους πόρους συστήματος για την φάση σχεδίασης και πολύ λίγους για την φάση αναζήτησης. Ωστόσο το σχέδιο βασίζεται στο στόχο. Αν ο στόχος αλλάξει η φάση σχεδίασης πρέπει να υπολογιστεί ξανά. Παρόλο που ο D* επιτρέπει την επανασχεδίαση καινούριας διαδρομής, δεν υποστηρίζει την αλλαγή του στόχου.

Οι παραπάνω λόγοι οδήγησαν στην ανάπτυξη μεθόδων roadmap όπου η φάση αναζήτησης μπορεί να συμπεριλαμβάνει και την αρχή και το στόχο. Η φάση σχεδίασης μας δίνει μια ανάλυση του χώρου που επιτρέπει την αλλαγή των σημείων αρχής και στόχου. Μία καλή αναλογία είναι τα μέσα μαζικής μεταφοράς όπως το μετρό. Βρίσκουμε την διαδρομή μας μέχρι τον σταθμό, ταξιδεύουμε με το τρένο, κατεβαίνουμε κοντά στο προορισμό μας και τέλος από εκεί βρίσκουμε μια διαδρομή προς τον στόχο μας. Η διαδρομή του τρένου είναι πάντα ίδια. Η σχεδίαση των διαδρομών, προς τον αρχικό σταθμό και από το τελικό σταθμό προς το στόχο, είναι σχετικά εύκολη καθώς είναι, ιδανικά, μικρές. Με αυτή τη μέθοδο το πρόβλημα της πλοήγησης γίνεται η δημιουργία ενός δικτύου ελευθέρων διαδρομών μέσα στο περιβάλλον που λειτουργούν όπως και το μετρό. Αυτό το δίκτυο διαδρομών ονομάζεται roadmap (χάρτης δρόμων). Ο roadmap υπολογίζεται μόνο μία φορά και των χρησιμοποιούμε για την αναζήτηση μιας διαδρομής μεταξύ δύο οποιονδήποτε σημείων αρχής και στόχου.

5.2.4 Voronoi Diagram

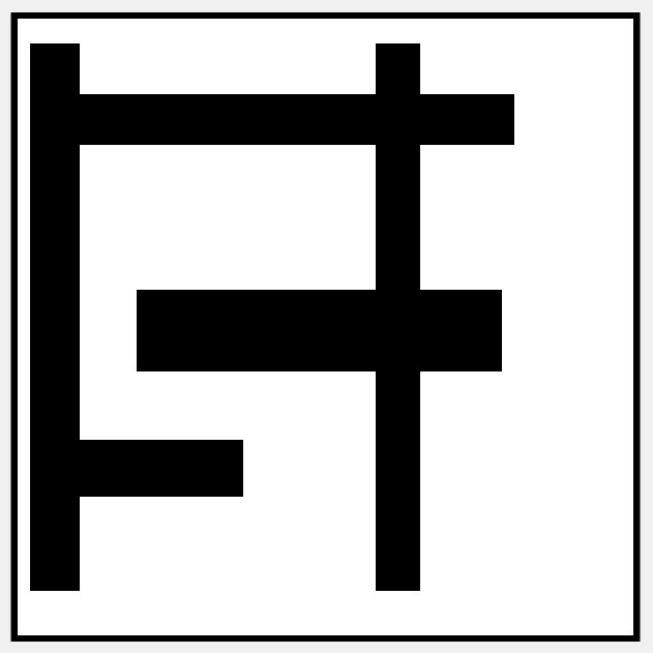
Το διάγραμμα Voronoi ορίζεται ως μία ομάδα σημείων τα οποία απέχουν εξίσου το ίδιο από δύο ή περισσότερα χαρακτηριστικά ενός αντικειμένου. Το διάγραμμα Voronoi χωρίζει τον χώρο σε περιοχές, με κάθε περιοχή να περιλαμβάνει ένα χαρακτηριστικό. Για κάθε σημείο στην περιοχή αυτό το χαρακτηριστικό είναι και το πιο κοντινό από όλα τα χαρακτηριστικά.

Σε αυτή τη μέθοδο υπάρχουν μόνο O(n) άκρες και κατασκευάζονται σε χρόνο Ω(n log n), όπου n είναι ο αριθμός των χαρακτηριστικών. Ένα πλεονέκτημα αυτής της μεθόδου είναι ότι η αρχική σύνδεση του αντικειμένου μεταφέρεται στο διάγραμμα, σε αντίθεση με άλλες μεθόδους που χρειάζεται να δημιουργήσουν την σύνδεση εκ νέου σε κάποιο άλλο βήμα.

Στην πράξη τα διαγράμματα Voronoi δεν είναι αποδοτικά σε πολλές διαστάσεις και χρειάζονται πολύ περίπλοκες δομές δεδομένων. Επίσης, η μορφή του διαγράμματος αλλάζει όταν διαφορετικά χαρακτηριστικά ενός αντικειμένου λαμβάνονται ως τοποθεσίες, όπως κορυφές, άκρες, ή ακόμα και όλο το αντικείμενο.

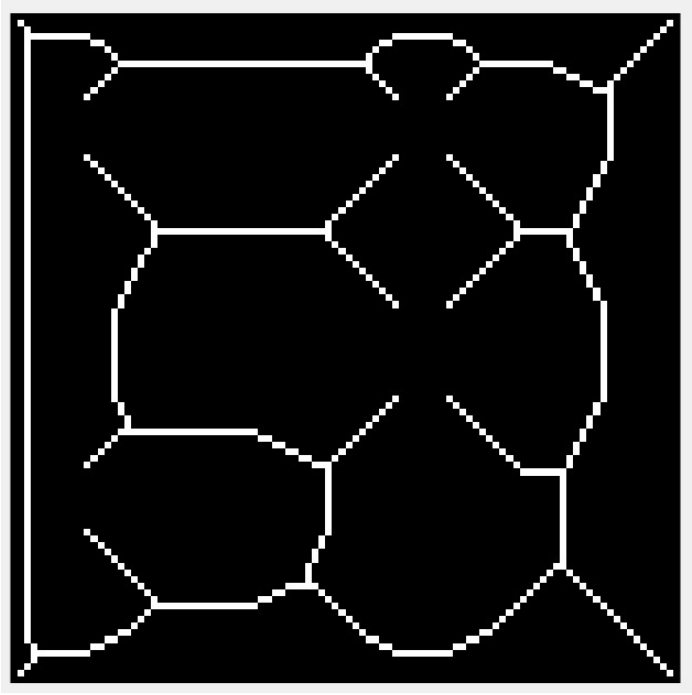
Χρησιμοποιώντας τον ίδιο χάρτη (πλέγμα πληρότητας) που είχαμε και στα προηγούμενα παραδείγματα μπορούμε να δούμε ένα Voronoi roadmap.

Αρχικά έχουμε το πλέγμα πληρότητας όπου με άσπρο είναι ο ελεύθερος χώρος και με μαύρο τα εμπόδια.



Εικόνα 5.11 – Πλέγμα πληρότητας χάρτη.

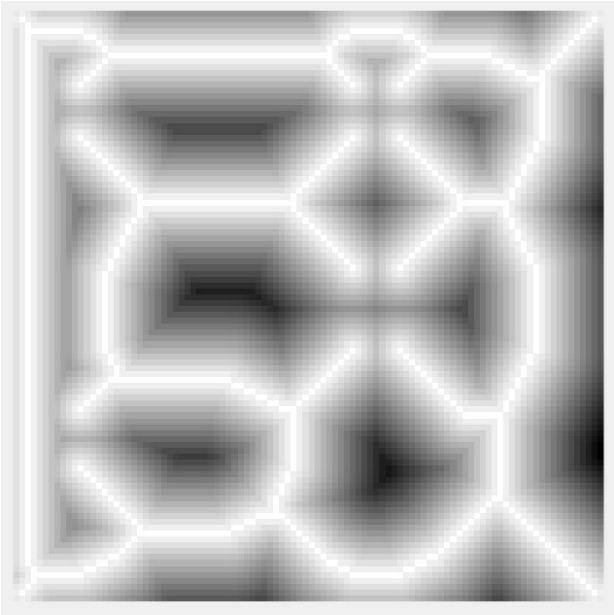
Ο τοπολογικός σκελετός της ελεύθερης περιοχής υπολογίζεται με έναν μορφολογικό αλγόριθμο επεξεργασίας εικόνας του Matlab, γνωστή ως thinning (function ithin()).



Εικόνα 5.12 – Πλέγμα πληρότητας χάρτη μετά το thining.

Βλέπουμε ότι η περιοχή των εμποδίων (μαύρο) έχει μεγαλώσει και η ελεύθερη περιοχή (άσπρο) έχει γίνει ένα λεπτό δίκτυο ενωμένων άσπρων κελιών που ισαπέχουν από τα όρια των αρχικών εμποδίων..

Τέλος χρησιμοποιώντας την μέθοδο distance transform μπορούμε να δούμε τις τιμές απόστασης των pixels από το δίκτυο.



Εικόνα 5.13 – Βάρος απόστασης distance transform προς το δίκτυο Voronoi.

Κώδικας παραδείγματος:

```
% Εισαγωγή χάρτη.
load map1;

% Επεξεργασία του χάρτη για την χρήση του thinning.
free = 1-map;
free(1,:) = 0; free(100,:) = 0;
free(:,1) = 0; free(:,100) = 0;

% Χρηση thinning
skeleton = ithin(free);

% Distance transform
dis_tran = bwdist(skeleton,'euclidean');
distance = repmat(mat2gray(dis_tran), [1 1 3]);
distance = 1-distance;

% Δημιουργία γραφημάτων.
imshow(distance); (Εικόνα 5.13)
figure;
imshow(skeleton); (Εικόνα 5.12)
figure;
imshow(free); (Εικόνα 5.11)
```

5.2.5 Probalistic Roadmap

Το μεγάλο κόστος των distance transform και Voronoi τα κάνει ακατόρθωτα σε μεγάλους χάρτες και οδήγησε στην ανάπτυξη των πιθανολογικών (probabilistic) μεθόδων. Αυτές οι μέθοδοι παίρνουν αραιά δείγματα από το κόσμο και η πιο γνωστή μέθοδος είναι η Probalistic Road Map.

Το PRM (Probalistic Road Map) είναι μια μέθοδος υπολογισμού ελεύθερων διαδρομών χωρίς συγκρούσεις σε περιβάλλοντα με στατικά εμπόδια. Χωρίζεται σε δύο φάσεις, την φάση της εκμάθησης και την φάση της αναζήτησης.

Στην φάση της εκμάθησης δημιουργείται ένας χάρτης πιθανών διαδρομών επιλέγοντας τυχαία ελεύθερα σημεία στον χάρτη (nodes) και ενώνοντάς τα με γραμμές που δεν διέρχονται από εμπόδια (edges).

Στην συνέχεια γίνεται η φάση της αναζήτησης. Στην αναζήτηση ψάχνουμε μια διαδρομή μεταξύ δύο σημείων στον χάρτη. Αρχικά η μέθοδος ενώνει τα δύο σημεία του χάρτη (αρχή και τέλος) με τα, αντίστοιχα δύο πιο κοντινά σε αυτά, nodes. Έπειτα γίνεται ανεύρεση μιας αλληλουχίας γραμμών (edges) που ενώνουν τα δύο nodes. Η αλληλουχία αυτή είναι και η ελεύθερη διαδρομή που ψάχναμε.

Αρχικά κατασκευάζεται ένα διάγραμμα χάρτη διαδρομών

$$R = (N,E)$$

όπου N είναι nodes (τυχαία επιλεγμένα ελεύθερα σημεία) και E απλές διαδρομές (edges). Ένα edge(a,b) περιγράφει μία εφικτή διαδρομή μεταξύ των nodes a και b. Αυτές οι διαδρομές ονομάζονται τοπικές διαδρομές (local paths) και υπολογίζονται πολύ γρήγορα μέσω ενός σχεδιαστή (planner) που ονομάζουμε τοπικό σχεδιαστή (local planner). Δεν είναι απαραίτητο να αποθηκεύονται στον χάρτη διαδρομών καθώς ο υπολογισμός τους είναι πολύ εύκολος, σώζοντας έτσι πολύ χώρο.

Φάση εκμάθησης

Η φάση εκμάθησης αποτελείται από δύο βήματα, την κατασκευή και την επέκταση. Στο βήμα της κατασκευής δημιουργείται ένα συνδεδεμένο διάγραμμα με αρκετά nodes ώστε να καλύπτει όλο τον ελεύθερο χώρο (C-space) και ακόμα και τα πιο «δύσκολα» σημεία του χάρτη να έχουν κάποια nodes. Το δεύτερο βήμα προσπαθεί να βελτιώσει τη συνδεσιμότητα του χάρτη. Διαλέγει nodes στα «δύσκολα» σημεία του χάρτη και επεκτείνει το διάγραμμα γύρω τους δημιουργώντας καινούρια nodes κοντά τους. Για τον λόγο αυτό ο τελικός χάρτης διαδρομών δεν είναι ομοιόμορφος αλλά εξαρτάται από την πολυπλοκότητα του χώρου.

1. Βήμα κατασκευής

Αρχικά το διάγραμμα R=(N,E) είναι άδειο. Έπειτα γίνεται η τυχαία επιλογή των nodes N. Για κάθε καινούριο node (c), επιλέγεται ένας αριθμός nodes από το παρόν N. Με την βοήθεια του local planner προσπαθούμε να ενώσουμε το καινούριο node με τα υπόλοιπα επιλεγμένα nodes. Κάθε φορά που ο local planner καταφέρνει να ενώσει το node (c) με κάποιο επιλεγμένο node (n), τότε το edge(c,n) προστίθεται στο E.

Η επιλογή των nodes για τον έλεγχο ένωσης με το καινούριο node (c) γίνεται ως εξής: Πρώτα, επιλέγεται μια ομάδα N_c γειτονικών nodes από το παρόν N. Τα γειτονικά αυτά nodes απέχουν μια συγκεκριμένη απόσταση D από το c. Τέλος, προσπαθούμε να ενώσουμε το node c με κάθε ένα από τα επιλεγμένα nodes.

Κάθε φορά που το local planner καταφέρνει να ενώσει δυο nodes το διάγραμμα R ανανεώνεται δυναμικά. Για το λόγο αυτό δεν είναι απαραίτητο να ψάχνουμε το διάγραμμα για ήδη ενωμένα nodes, όταν διαλέγουμε τα N_c.

2. Βήμα επέκτασης

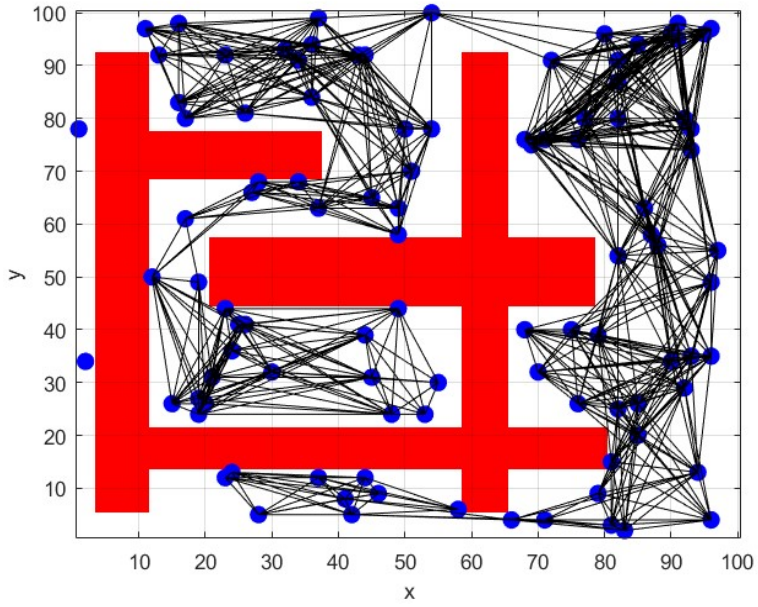
Όταν ο αριθμός των nodes είναι μεγάλος τότε η ομάδα N έχει μια αρκετά ομοιόμορφη κάλυψη του ελεύθερου χώρου (C-space). Σε εύκολους χώρους το διάγραμμα R είναι καλά συνδεδεμένο. Σε πιο δύσκολους χώρους με στενά περάσματα το διάγραμμα R αποτελείται από περιοχές με πολλά nodes και περιοχές με λίγα. Αυτό κάνει τη συνδεσιμότητα πιο δύσκολη.

Το βήμα επέκτασης προσπαθεί να βελτίωση τη συνδεσιμότητα του διαγράμματος R. Η λογική αυτού του βήματος, είναι η επιλογή nodes τα οποία βρίσκονται σε αυτά τα δύσκολα σημεία του χώρου και να τα «επεκτείνει». Η επέκταση γίνεται δημιουργώντας γειτονικά nodes γύρω από τα επιλεγμένα, αυξάνοντας έτσι των αριθμό των nodes στις δύσκολες περιοχές.

Φάση αναζήτησης

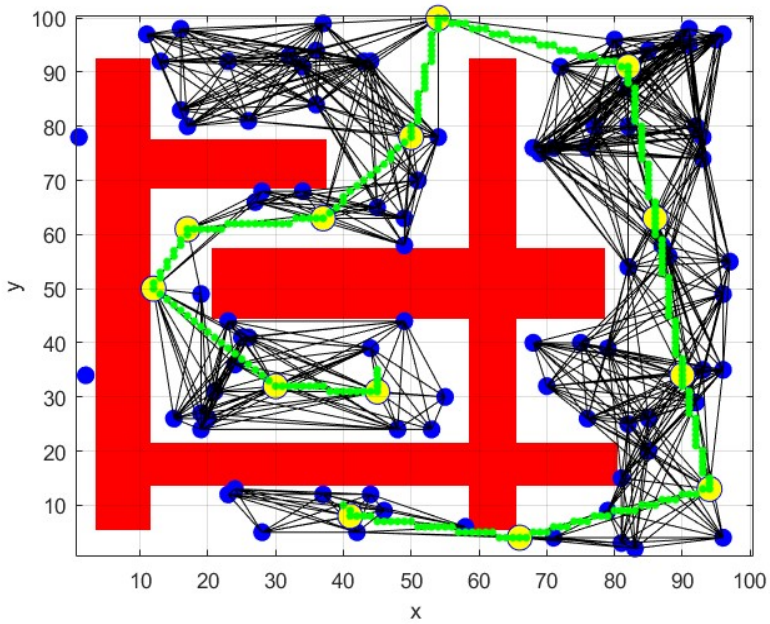
Στη φάση αναζήτησης, ψάχνουμε διαδρομές μεταξύ κάποιας αρχής (start) και κάποιου στόχου (goal), χρησιμοποιώντας το χάρτη διαδρομών (roadmap) που κατασκευάσαμε στην φάση εκμάθησης. Η αναζήτηση αποτελείται από μια αρχή s και ένα στόχο g. Προσπαθούμε να ενώσουμε τα s,g σε δύο nodes του R, s' και g' με διαδρομές P_s και P_g. Έπειτα υπολογίζουμε μια διαδρομή P στο R που ενώνει τα s' και g'. Αν τα παραπάνω βήματα δεν αποτύχουν τότε θα έχουμε μια διαδρομή που θα ενώνει την αρχή και το στόχο.

Χρησιμοποιώντας την μέθοδο PRM στο χάρτη μας βλέπουμε τα nodes και τις διαδρομές μεταξύ αυτών που κατασκεύασε ο αλγόριθμος.



Εικόνα 5.14 – Χάρτης περιβάλλοντος μαζί με δίκτυο από nodes (PRM).

Τέλος μπορούμε να δούμε και την διαδρομή μεταξύ των σημείων start (40,10) και goal (45,35).



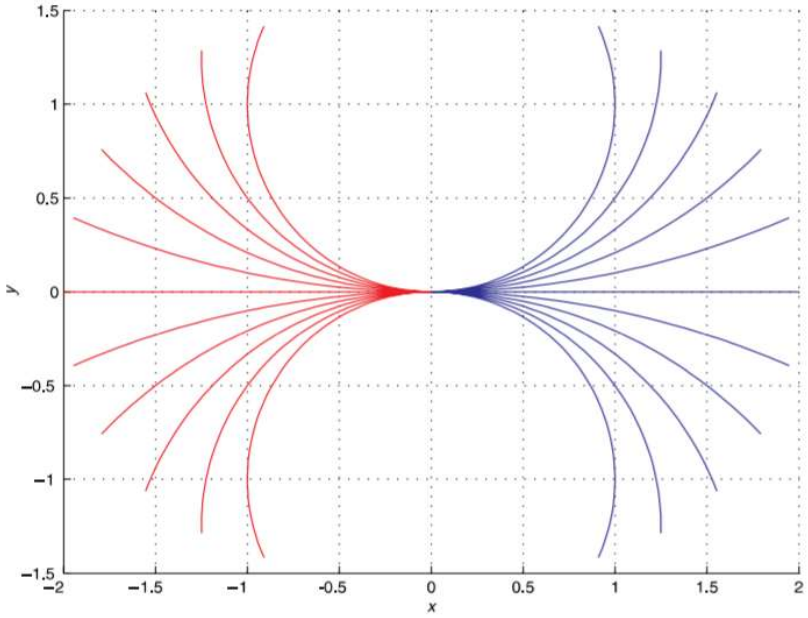
Εικόνα 5.15 – Χάρτης περιβάλλοντος μαζί με δίκτυο από nodes (PRM) και διαδρομή μεταξύ start-goal.

Κώδικας παραδείγματος:

```
% Αρχικοποίηση σημείων αρχικής θέσης και στόχου.
goal = [45 35];
start = [40 10];

% Εισαγωγή χάρτη.
load map1;

% Δημιουργία PRM με βάση το χάρτη.
prm = PRM(map);
prm.plan();

% Γράφημα του PRM. (Εικόνα 5.14)
prm.visualize();
figure;

% Γράφημα του PRM μαζί με την διαδρομή. (Εικόνα 5.15)
prm.path(start, goal);
```

5.2.6 Rapidly-exploring Random Tree (RRT)

Η μέθοδος σχεδίασης RRT λαμβάνει υπόψιν της το κινηματικό μοντέλο του οχήματος. Αρχικά ας πάρουμε για παράδειγμα το μοντέλο κίνησης ενός ποδηλάτου, με την παρακάτω εικόνα να μας δείχνει μια ομάδα από κινήσεις που μπορούν να γίνουν από το ποδήλατο για διακριτές τιμές της ταχύτητας, προς τα μπροστά και προς τα πίσω, και της γωνίας του τιμονιού για ένα συγκεκριμένο χρονικό διάστημα. Βλέπουμε ένα υποσύνολο πιθανών θέσεων που μπορεί να πάει το όχημά μας με βάση την αρχική του θέση. Στο συγκεκριμένο παράδειγμα υπολογίστηκαν από την αρχική θέση 22 πιθανές στάσεις του οχήματος. Για κάθε μία από αυτές θα μπορούσαμε να υπολογίσουμε άλλες 22 πιθανές στάσεις μετά από δυο χρονικά διαστήματα, και ούτω καθεξής. Μετά από μερικά χρονικά διαστήματα θα είχαμε έναν μεγάλο αριθμό πιθανών στάσεων.

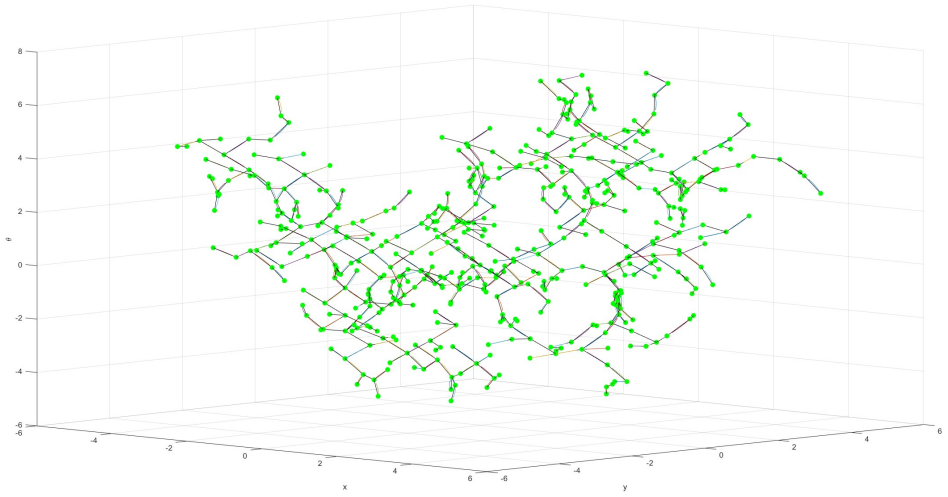


Εικόνα 5.16 – Ομάδα πιθανών κινήσεων μοντέλου ποδηλάτου για χρονικό διάστημα 2 sec.

Πηγή: "Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised. .", Βιβλιογραφία [14].

Όπως και το PRM η μέθοδος αυτή είναι ένας πιθανολογικός αλγόριθμος και τα κύρια βήματά του είναι τα εξής. Ένα γράφημα των διαμορφώσεων του ρομπότ διατηρείται και κάθε node είναι μία διαμόρφωση $\xi \in SE(2)$, η οποία παριστάνεται από ένα 3-διάνυσμα $\xi \sim (x, y, \theta)$. Το πρώτο node στο γράφημα είναι κάποια αρχική διαμόρφωση του ρομπότ. Μία τυχαία διαμόρφωση ξ_{rand} επιλέγεται και το node με τη πιο κοντινή διαμόρφωση ξ_{near} βρίσκεται. Έπειτα υπολογίζεται ένας έλεγχος που κινεί το ρομπότ από το σημείο ξ_{near} στο σημείο ξ_{rand} σε ένα συγκεκριμένο χρονικό διάστημα. Το νέο σημείο που βρίσκεται το ρομπότ ονομάζεται ξ_{new} και προστίθεται στο γράφημα.

Με την βοήθεια του Robotics toolbox μπορούμε να δούμε ένα RRT roadmap για ένα περιβάλλον χωρίς εμπόδιο και με ένα βασικό μοντέλο κινηματικής ενός ποδηλάτου.



Εικόνα 5.17 – Ένα RRT βασισμένο στο μοντέλο ποδηλάτου.

Βλέπουμε ότι οι διαδρομές καλύπτουν αρκετά τον ελεύθερο χώρο όχι μόνο στις κατευθύνσεις x και y αλλά και στο προσανατολισμό, για αυτό το λόγο ο αλγόριθμος ονομάζεται και rapidly exploring.

Κώδικας παραδείγματος:

```
% Δημιουργία RRT.
rrt = RRT();
rrt.plan();

% Γράφημα «δέντρου» RTT. (Εικόνα 5.17)
rrt.visualize();
```

Κεφάλαιο 6 - Πρακτικό μέρος

6.1 Λογισμικό

1. Matlab

Το Matlab είναι μια πλατφόρμα προγραμματισμού σχεδιασμένη για μηχανικούς και επιστήμονες. Χρησιμοποιεί την δικιά του γλώσσα η οποία βασίζεται σε πίνακες επιτρέποντας μια πιο φυσική έκφραση των μαθηματικών υπολογισμού.

Με το Matlab μπορούμε να:

- Αναλύσουμε δεδομένα
- Αναπτύξουμε αλγόριθμους
- Δημιουργήσουμε μοντέλα και εφαρμογές

Η γλώσσα, οι εφαρμογές, και οι ενσωματωμένες μαθηματικές λειτουργίες επιτρέπουν την γρήγορη αναζήτηση πολλαπλών προσεγγίσεων για τη λύση ενός προβλήματος. Με το Matlab μπορούμε να πάρουμε τις ιδέες μας από την έρευνα στην παραγωγή με την ανάπτυξη υψηλού επιπέδου εφαρμογών και ενσωματωμένων συσκευών, όπως επίσης και με την χρήση του Simulink και του Model-Based Design.

Χρησιμοποιείται από εκατομμύρια μηχανικών και επιστημόνων στην βιομηχανία και την εκπαίδευση. Μπορεί να χρησιμοποιηθεί για ένα εύρος εφαρμογών όπως deep learning και machine learning, επεξεργασία σήματος και τηλεπικοινωνιών, επεξεργασία εικόνας και βίντεο, συστήματα ελέγχου, δοκιμές και μετρήσεις, οικονομικούς υπολογισμούς, και υπολογιστική βιολογία.

2. V-Rep

Το V-Rep είναι ένας εξομοιωτής ρομποτικών συστημάτων, με ενσωματωμένο περιβάλλον ανάπτυξης και βασίζεται σε μία διαμερισμένου ελέγχου αρχιτεκτονική. Κάθε αντικείμενο ή μοντέλο μπορεί μεμονωμένα να ελεγχθεί από έναν ενσωματωμένο κώδικα, plugin, κόμβους ROS, κόμβους BlueZero, API απομακρυσμένου ελέγχου, ή κάποια άλλη προσαρμοσμένη λύση. Αυτό κάνει το V-Rep πολύ ευπροσάρμοστο και ιδανικό για εφαρμογές πολλαπλών ρομποτικών συστημάτων. Οι ελεγκτές μπορούν να γραφτούν σε C/C++, Python, Java, Lua, Matlab, Octave ή Urbi.

Το V-rep μπορεί να χρησιμοποιηθεί σε εφαρμογές όπως:

- Εξομοίωση συστημάτων αυτοματισμού ενός εργοστασίου.
- Απομακρυσμένη παρακολούθηση.
- Έλεγχος υλικού.
- Γρήγορος σχεδιασμός πρωτοτύπων και επαλήθευση.
- Παρακολούθηση ασφάλειας.
- Γρήγορη ανάπτυξη αλγόριθμων.
- Εκπαίδευση ρομποτικής.
- Παρουσίαση προϊόντων.

Το V-Rep μπορεί να σταθεί ως αυτόνομη εφαρμογή ή να ενσωματωθεί πολύ εύκολα σε μία άλλη. Το μικρό του ίχνος και το περίτεχνο API κάνει το V-Rep ιδανικό για ενσωμάτωση σε υψηλού επιπέδου εφαρμογές. Ο ενσωματωμένος Lua interpreter κάνει το V-Rep μία εξαιρετικά ευπροσάρμοστη εφαρμογή, αφήνοντας την ελευθερία στο χρήστη να συνδυάσει χαμηλές και υψηλές λειτουργίες για την δημιουργία νέων υψηλών λειτουργιών.

6.2 Υλικό

Το πρακτικό μέρος της πτυχιακής εργασίας έγινε σε εξομοίωση χρησιμοποιώντας εντός αυτής ένα πραγματικό ρομποτικό σύστημα και αισθητήρες. Παρακάτω θα δούμε κάποιες λεπτομέρειες και τεχνικά χαρακτηριστικά των όσων χρησιμοποιήθηκαν στην εξομοίωση.

1. Youbot - KUKA

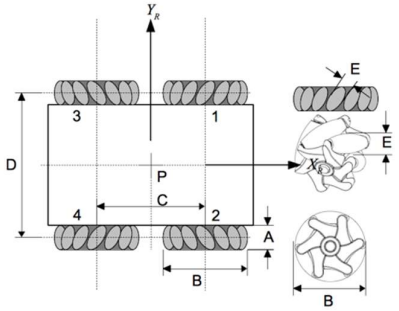
Αρχικά ας δούμε τον πρωταγωνιστή της εξομοίωσης, το ρομποτικό σύστημα Youbot από την KUKA. Η βάση του χρησιμοποιεί ένα σύστημα οδήγησης προς κάθε κατεύθυνση με τέσσερεις τροχούς mecanum. Σε αντίθεση με τους κανονικούς τροχούς, οι τροχοί mecanum αποτελούνται από μια σειρά κυλίνδρων τοποθετημένοι σε γωνία 45^{o}. Αυτό επιτρέπει στο ρομπότ κίνηση προς κάθε κατεύθυνση, συμπεριλαμβανομένης της κίνησης πλαγίως, καθιστώντας το ρομπότ πολύ πιο ευκίνητο σε δύσκολες περιοχές.

Συνδεδεμένος στη βάση του βρίσκεται ένας ρομποτικός βραχίονας με πέντε βαθμούς ελευθερίας. Ο βραχίονας αποτελείται από 5 συνδέσμους με πλήρη περιστροφική κίνηση. Είναι στερεός και δυνατός, όπως και οι μεγαλύτεροι βιομηχανικοί ρομποτικοί βραχίονες της KUKA.

Αναλυτικά χαρακτηριστικά.

Βάση:

- Συνολικό βάρος: ~20 kg
- Επιτρεπόμενο βάρος φορτίου: 20 kg
- Συνολικό μήκος: 580 mm
- Συνολικό πλάτος: 380 mm
- Συνολικό ύψος: 140 mm
- Απόσταση από το έδαφος: 20 mm
- Ελάχιστη ταχύτητα: 0.01 m/s
- Μέγιστη ταχύτητα: 0.8 m/s
- Τροφοδοτικό: 24 V
- Επικοινωνία: EtherCat



Εικόνα 6.1 – Κάτοψη βάσης youbot.

–Πηγή: http://www.youbot-store.com/wiki/index.php/YouBot_Detailed_Specifications (16/07/2018)

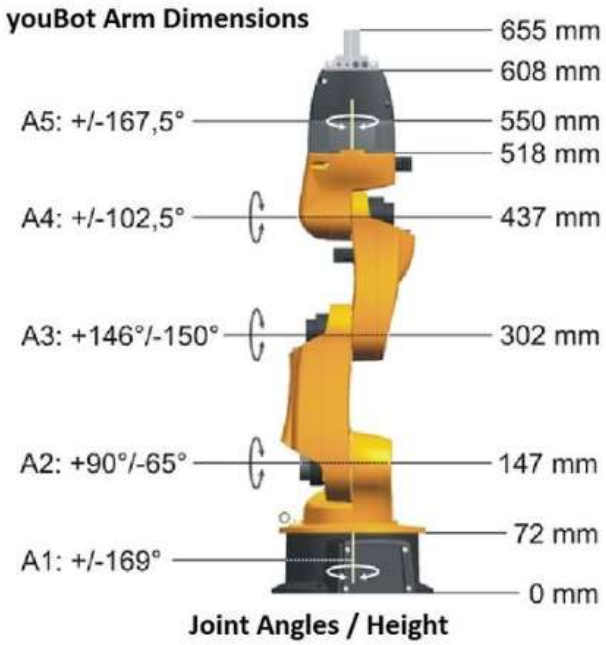
A = 74,87 mm B = 100 mm C = 471 mm D = 300,46 mm E = 28 mm (μέγιστη διάμετρος περιστροφής)

Τροχοί:

- Είδος: Omni-directional
- Αριθμός τροχών: 4
- Διάμετρος: 47.5 mm

Βραχίονας

- Αριθμός αξόνων: 5
- Ύψος: 655 mm
- Περιοχή εργασίας: 0.513 m^2
- Βάρος: 5.3 kg
- Βάρος φορτίου: 0.5 kg
- Κατασκευή: Μαγνήσιο
- Επαναληψιμότητα θέσης: 1 mm
- Επικοινωνία: EtherCat
- Τροφοδοσία: 24 V DC
- Μέγιστη ισχύ: 80 W
- Ταχύτητα αξόνων: 90 deg/s
- Όριο Encoders: 1000 Hz



Εικόνα 6.2 – Διαστάσεις ρομποτικού βραχίονα.

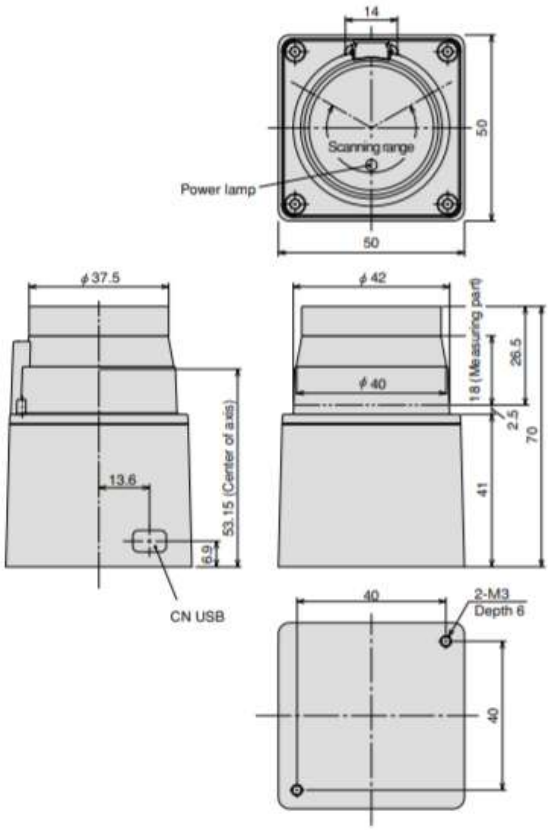
–Πηγή: http://www.youbot-store.com/wiki/index.php/YouBot_Detailed_Specifications (16/07/2018)

2. Αισθητήρας Hokuyo urg-04lx-ug01

Ο αισθητήρας πρόκειται για ένα laser εύρεσης απόστασης με μέγιστη απόσταση τέσσερα μέτρα και 240^{o} γωνία κάλυψης. Εντός της εξομοίωσης στο V-rep έχουμε μέγιστη απόσταση πέντε μέτρα.

Πιο αναλυτικά:

- Τροφοδοσία: 5 V DC ±5% (USB Bus power)
- Κατανάλωση: 500 mA ή λιγότερο
- Πηγή φωτός: Ημιαγώγιμη δίοδος laser λ=785nm
- Περιοχή ανίχνευσης: 0.06 μέχρι 4 μέτρα
- Ακρίβεια: 0.06 μέχρι 1m: ±30mm, 1 μέχρι 4m: 3% της μετρούμενης απόστασης
- Επαναληψιμότητα: 0.06 μέχρι 1m: ±30mm
- Γωνία ανίχνευσης: 204^{o}
- Ανάλυση: Περίπου 1 mm
- Διάμετρος ακτίνας: Περίπου 40 mm (στα 4 μέτρα)
- Χρόνος ανίχνευσης: 100msec/scan
- Διεπαφή: USB2.0 [Mini B] (Full Speed)
- Βάρος: Περίπου 160g



Εικόνα 6.3 – Διαστάσεις αισθητήρα Hokuyo urg-04lx-ug01.

–Πηγή: https://www.hokuyo-usa.com/application/files/5414/7196/1896/URG-04LX-UG01_Specifications_Catalog.pdf (16/07/2018)

3. Υπολογιστής εξομοίωσης

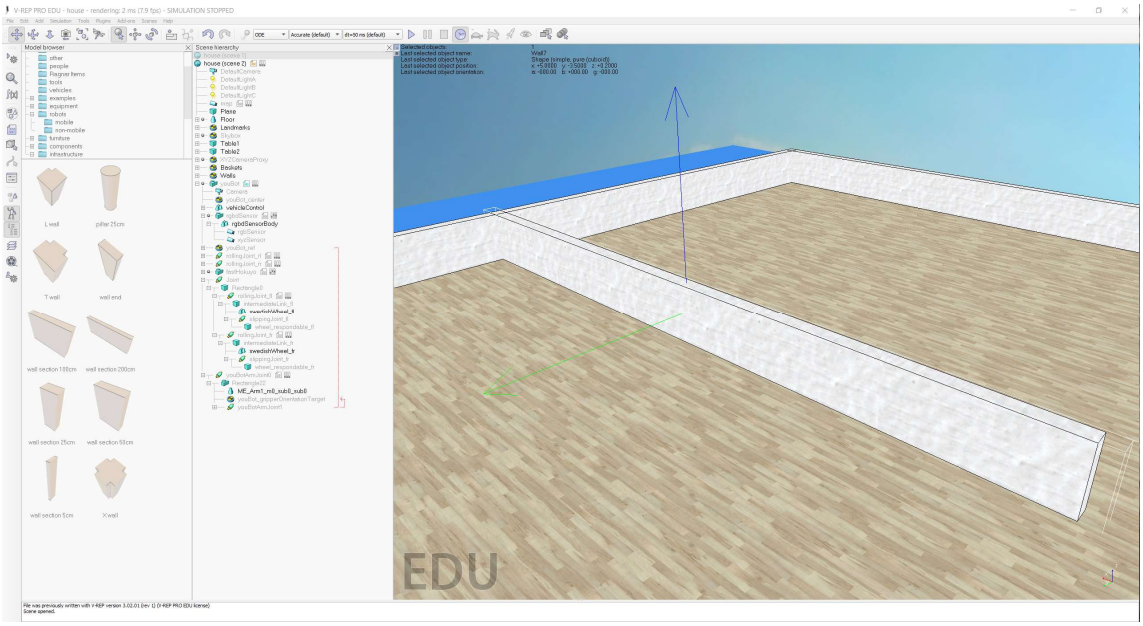
Η ανάπτυξη του κώδικα (Matlab) και η εξομοίωση V-Rep έγιναν σε υπολογιστή με λειτουργικό σύστημα Windows 10 και η χαρτογράφηση χρειάστηκε περίπου 29 εικονικά λεπτά και περίπου 32 πραγματικά λεπτά.

Τεχνικά χαρακτηριστικά υπολογιστή:

- CPU: Intel i5 4670k (@4.2GHz)
- RAM: 16GB
- GPU: Nvidia GTX 1070 8GB
- SSD 250GB
- PSU: 650 Watt

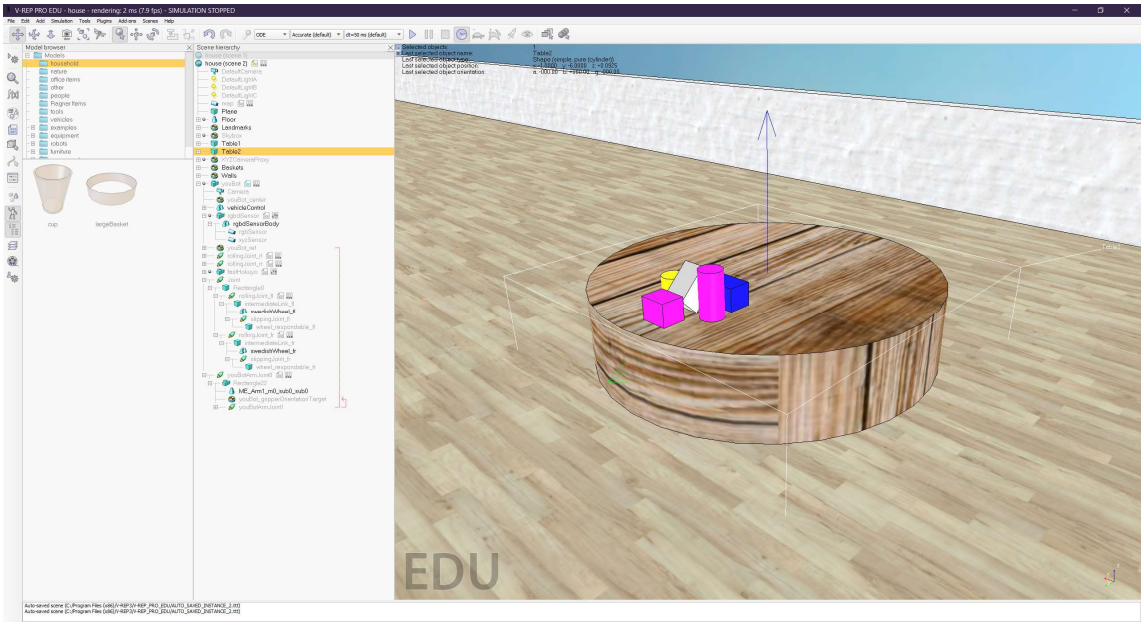
6.3 Δημιουργία περιβάλλοντος εξομοίωσης

Το περιβάλλον δημιουργήθηκε εξολοκλήρου εντός του προγράμματος V-Rep χρησιμοποιώντας τα υπάρχοντα στοιχεία. Αρχικά δημιουργήθηκε το δωμάτιο χρησιμοποιώντας τα στοιχεία πατώματος και τοίχων.



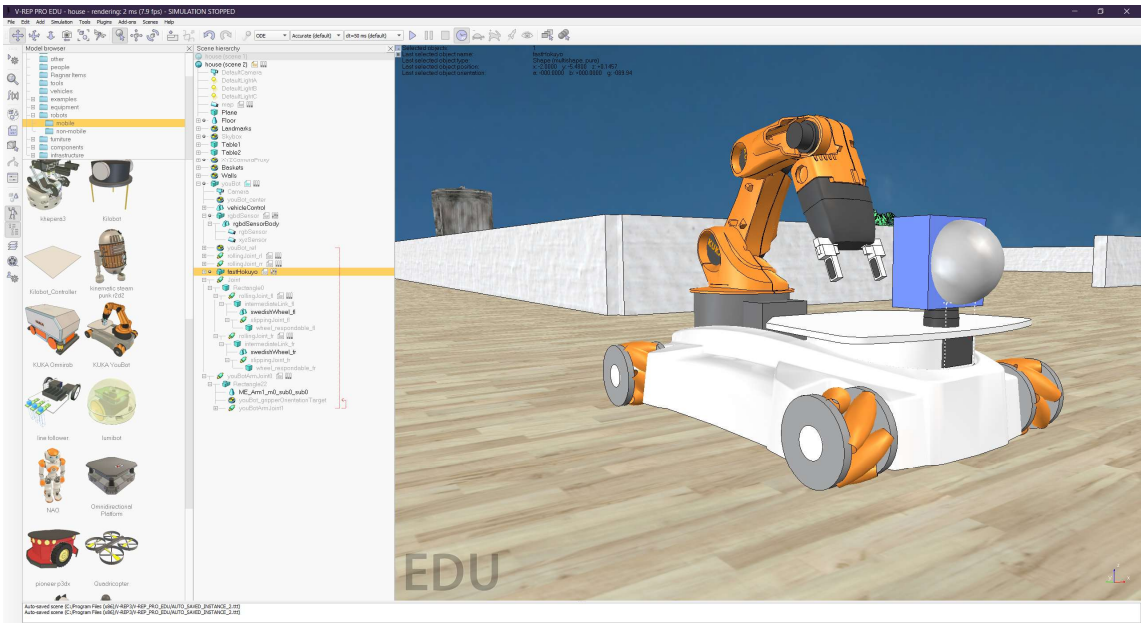
Εικόνα 6.4 – Περιβάλλον προγράμματος V-Rep. Κατασκευή δωματίου.

Στη συνέχεια προστέθηκαν διάφορα αντικείμενα όπως τα καλάθια ως εμπόδια στο χώρο.



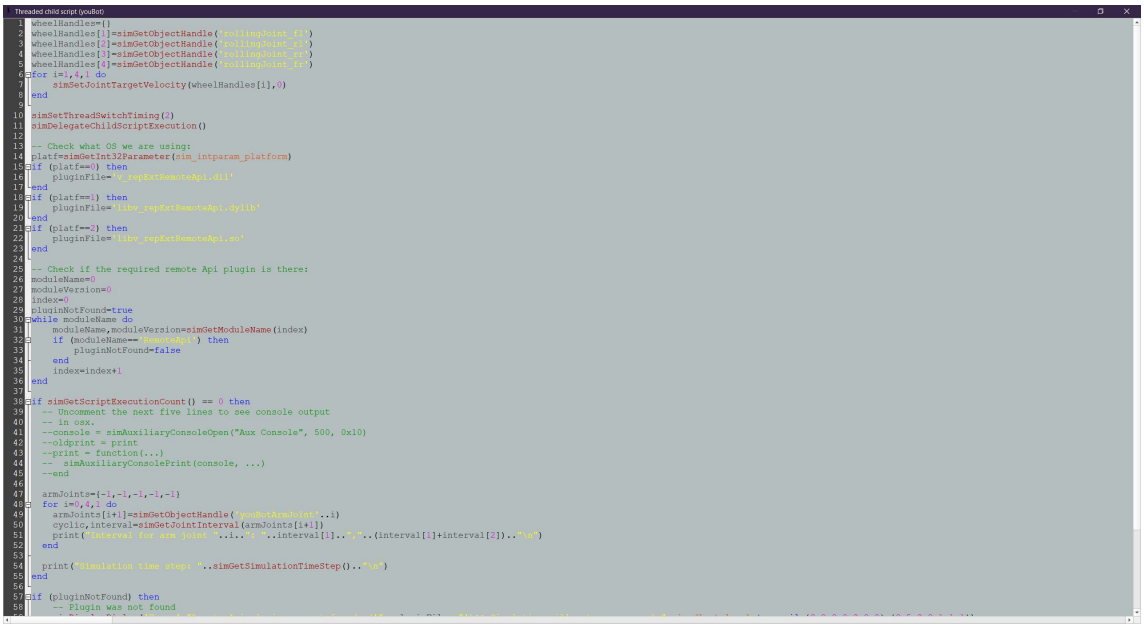
Εικόνα 6.5 – Περιβάλλον προγράμματος V-Rep. Τοποθέτηση εμποδίων.

Τέλος προστέθηκαν το ρομποτικό σύστημα Youbot και ο αισθητήρας.



Εικόνα 6.6 – Περιβάλλον προγράμματος V-Rep. Εισαγωγή Youbot και Hokuyo.

Μεγάλη σημασία έχει ο ενσωματωμένος κώδικας που έρχεται μαζί με τα ρομποτικά συστήματα και τους αισθητήρες. Είναι απαραίτητος για την λειτουργία τους εντός του εξομοιωτή και μας δίνει τις επίσης απαραίτητες πληροφορίες για τα handles του κάθε αντικειμένου. Τα handles βοηθάνε στην επικοινωνία και στη αναγνώριση των επιμέρους τμημάτων ενός αντικειμένου από τον κώδικα εντός του V-Rep ή τον κώδικα ενός τρίτου προγράμματος όπως το Matlab.

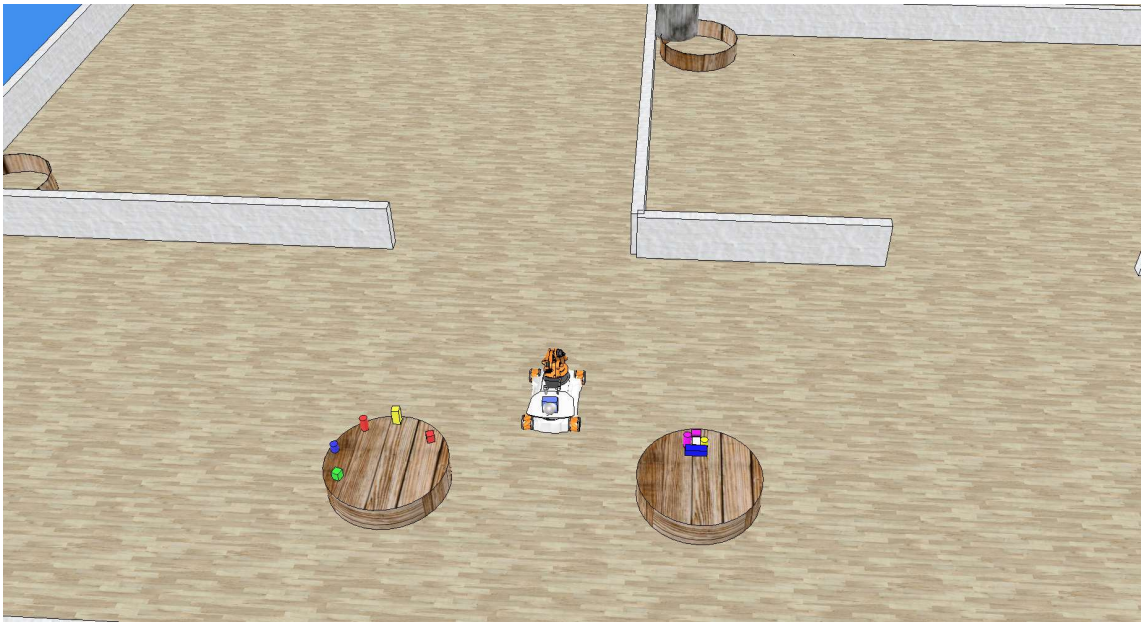


```
wheelHandles={}
wheelHandles[1]=simGetObjectHandle('rollingJoint_fl')
wheelHandles[2]=simGetObjectHandle('rollingJoint_rl')
wheelHandles[3]=simGetObjectHandle('rollingJoint_rr')
wheelHandles[4]=simGetObjectHandle('rollingJoint_fr')
for i=1,4,1 do
    simSetJointTargetVelocity(wheelHandles[i],0)
end

simSetThreadSwitchTiming(2)
simDelegateChildScriptExecution()

-- Check what OS we are using:
platf=simGetInt32Parameter(sim_intparam_platform)
if (platf==0) then
    pluginFile='v_repExtRemoteApi.dll'
end
if (platf==1) then
    pluginFile='libv_repExtRemoteApi.dylib'
end
if (platf==2) then
    pluginFile='libv_repExtRemoteApi.so'
end

-- Check if the required remote Api plugin is there:
moduleName=0
moduleVersion=0
index=0
pluginNotFound=true
while moduleName do
    moduleName,moduleVersion=simGetModuleName(index)
    if (moduleName=='RemoteApi') then
        pluginNotFound=false
    end
    index=index+1
end

if simGetScriptExecutionCount() == 0 then
  -- Uncomment the next five lines to see console output
  -- in osx.
  --console = simAuxiliaryConsoleOpen("Aux Console", 500, 0x10)
  --oldprint = print
  --print = function(...)
  --  simAuxiliaryConsolePrint(console, ...)
  --end

  armJoints={-1,-1,-1,-1,-1}
  for i=0,4,1 do
    armJoints[i+1]=simGetObjectHandle('youBotArmJoint'..i)
    cyclic,interval=simGetJointInterval(armJoints[i+1])
    print("Interval for arm joint "..i..": "..interval[1]..".."..(interval[1]+interval[2]).."\n")
  end

  print("Simulation time step: "..simGetSimulationTimeStep().."\n")
end

if (pluginNotFound) then
    -- Plugin was not found
```

Εικόνα 6.7 – Περιβάλλον προγράμματος V-Rep. Ενσωματωμένος κώδικας Youbot.

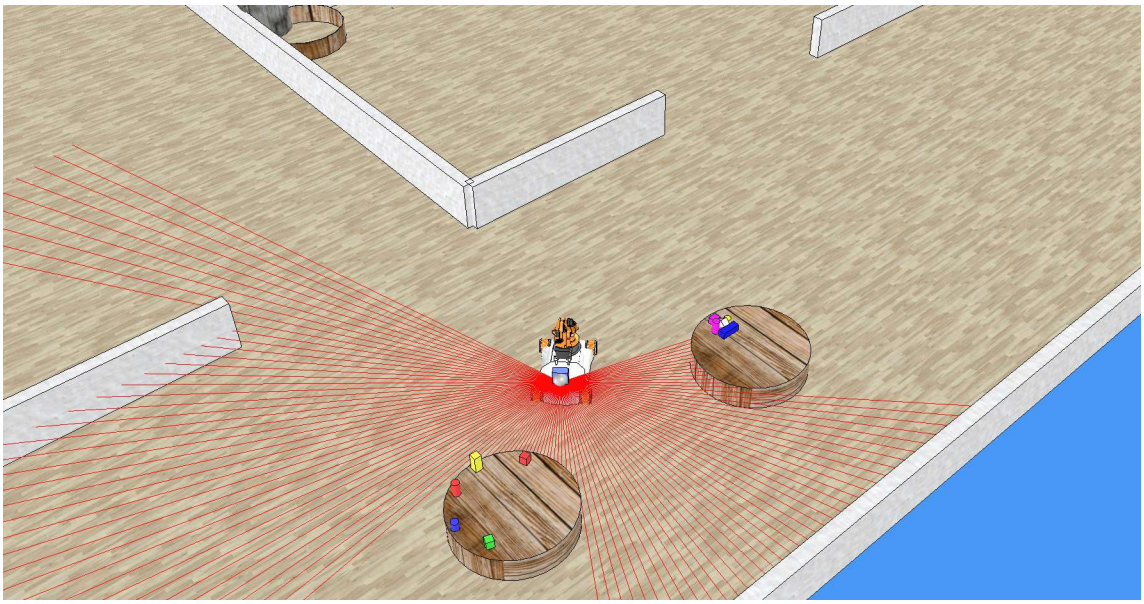
6.4 Παρουσίαση

Το ρομπότ ξεκινάει στην αρχική του θέση κάπου μέσα στον, άγνωστο για αυτό, χώρο.



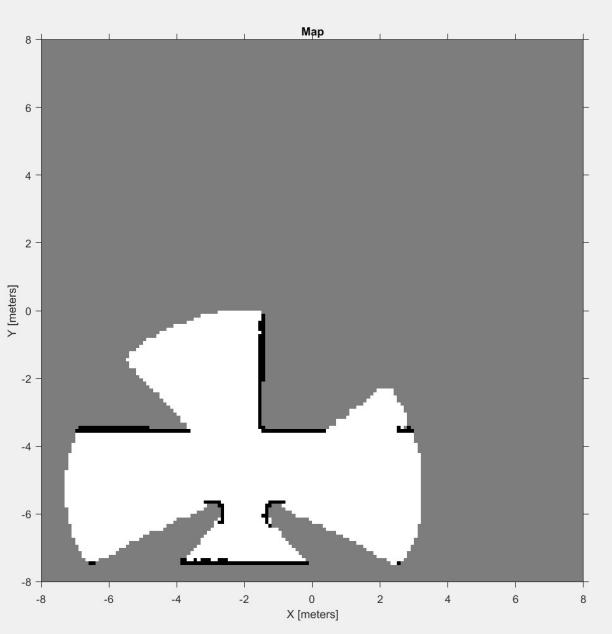
Εικόνα 6.8 – Περιβάλλον προγράμματος V-Rep. Αρχική κατάσταση ρομπότ.

Αρχικά πραγματοποιεί μια περιστροφή 360^{o} μοιρών καταγράφοντας τις μετρήσεις από τους αισθητήρες laser (hokuyo).



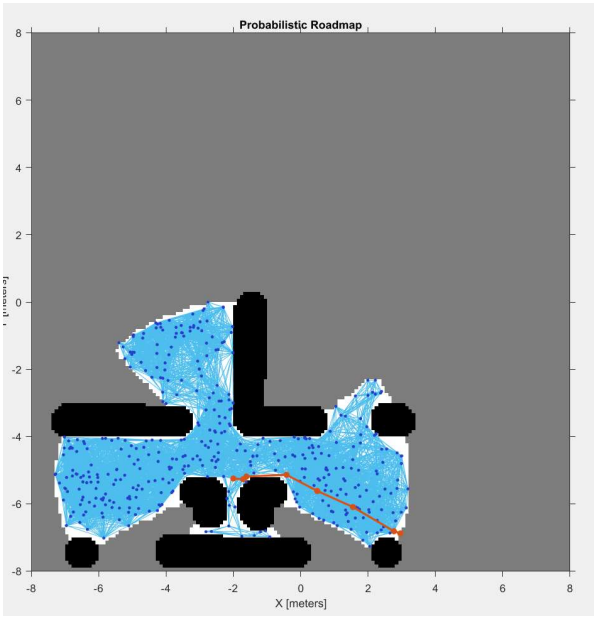
Εικόνα 6.9 – Περιβάλλον προγράμματος V-Rep. Περιστροφική κίνηση ρομπότ.

Στη συνέχεια χρησιμοποιώντας τις μετρήσεις δημιουργεί έναν χάρτη πλέγματος πληρότητας (occupancy grid).



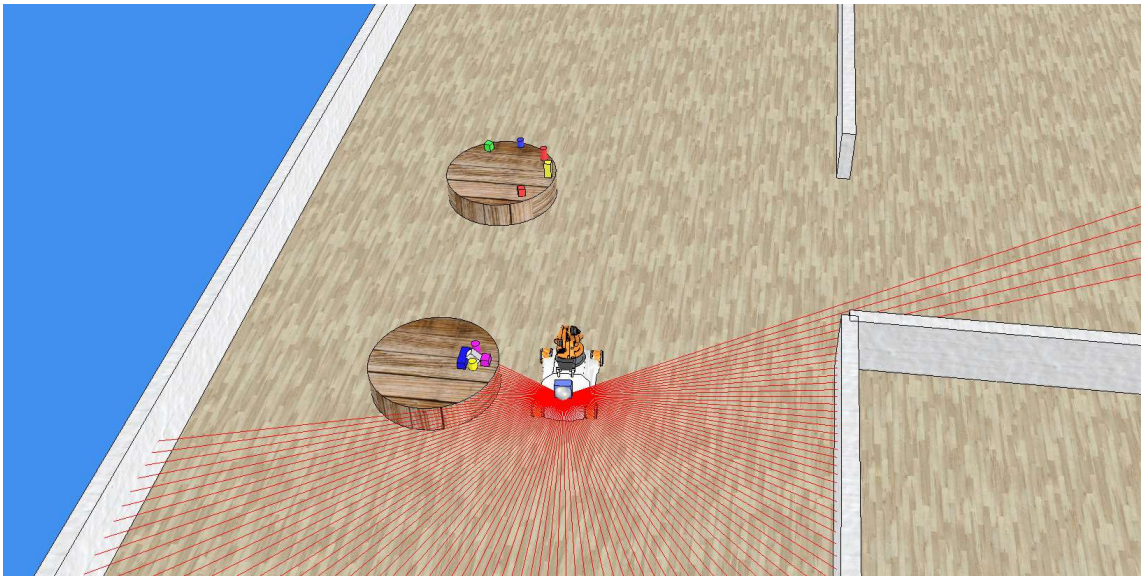
Εικόνα 6.10 – Πρώτη εκδοχή πλέγματος πληρότητας.

Έπειτα διαλέγει ένα από τα ελεύθερα (άσπρα) σημεία του χάρτη, που έχουν δίπλα τους άγνωστα σημεία (γκρι), για στόχο (goal) και χρησιμοποιώντας την μέθοδο PRM βρίσκει μια εφικτή διαδρομή προς αυτό.



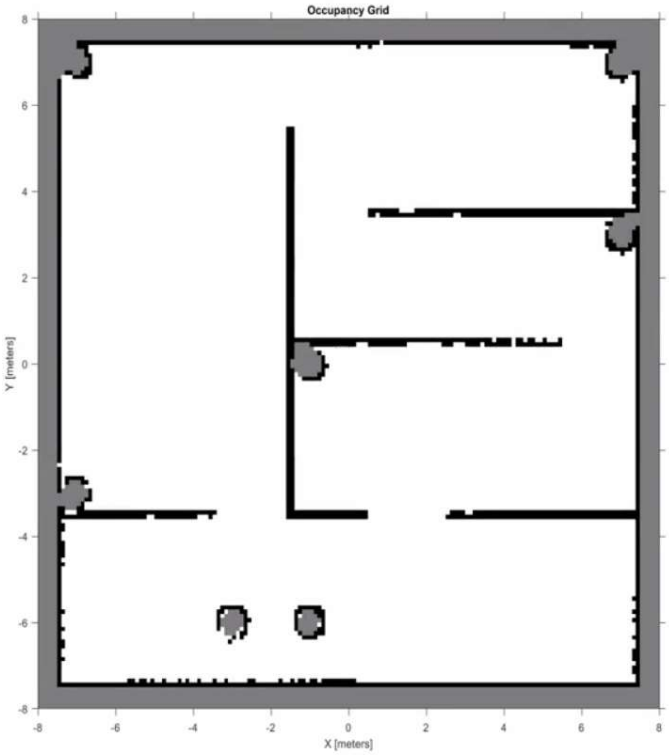
Εικόνα 6.11 – Πρώτη εκδοχή πλέγματος πληρότητας με δίκτυο διαδρομών PRM.

Έχοντας πλέον τις πληροφορίες που χρειάζεται για την διαδρομή το ρομπότ ξεκινάει την κίνηση του προς το στόχο.



Εικόνα 6.12 – Κίνηση ρομπότ προς σημείο εξερεύνησης.

Αφού φτάσει στο στόχο πραγματοποιεί άλλη μια περιστροφή 360º μοιρών καταγράφοντας το χώρο γύρω του και ανανεώνοντας τον χάρτη. Τέλος, τα παραπάνω βήματα επαναλαμβάνονται έως ότου, δεν θα υπάρχουν άλλα προσβάσιμα ανεξερεύνητα σημεία. Τότε ο αλγόριθμος σταματά και έχουμε την τελική μορφή του χάρτη.



Εικόνα 6.13 – Πλήρης εκδοχή πλέγματος πληρότητας.

6.5 Ανάλυση κύριων λειτουργιών κώδικα

1. Δημιουργία σύνδεσης Matlab/V-rep. Εκκίνηση και συγχρονισμός εξομοίωσης.

Αρχικά για την σύνδεση χρειαζόμαστε τα αρχεία remApi.m και remoteApi.dll τα οποία παρέχονται από το V-Rep και πρέπει να βρίσκονται εντός του φακέλου του project. Εντός του προγράμματος μας στο Matlab, χρησιμοποιούμε την εντολή vrep.simxStart για την εκκίνηση της σύνδεσης με το V-Rep.

Για την εκκίνηση της εξομοίωσης χρησιμοποιούμε την εντολή vrep.simxStartSimulation η οποία έχει διάφορες επιλογές συγχρονισμού μεταξύ Matlab και V-Rep. Στη δικιά μας εφαρμογή διαλέξαμε vrep.simx_opmode_oneshot_wait και σε συνδυασμό με την εντολή vrep.simxSynchronousTrigger αναγκάζουμε την εξομοίωση να περιμένει το Matlab. Αυτή η επιλογή έγινε διότι το Matlab απαιτεί περισσότερο χρόνο για τους υπολογισμούς από ότι ένας κύκλος εξομοίωσης.

2. Θέση του ρομπότ

Η θέση του ρομπότ στην εφαρμογή μας λαμβάνεται απευθείας από το V-Rep. Για την απόκτηση της γίνεται χρήση δύο εντολών.

- vrep.simxGetObjectPosition | Μας επιστρέφει την θέση του ρομπότ σε συντεταγμένες x,y,z στο παγκόσμιο επίπεδο (world frame).
- vrep.simxGetObjectOrientation | Μας επιστρέφει τον προσανατολισμό του ρομπότ.

3. Πληροφορίες αισθητήρα

Ο αισθητήρας μας όπως είδαμε και στα τεχνικά του χαρακτηριστικά μας επιστρέφει την απόσταση των αντικειμένων μέσα στο χώρο σε 240^{o} και μέγιστη απόσταση πέντε μέτρα.

Με την εντολή youbot_hokuyo παίρνουμε δύο πίνακες. Έναν πίνακα pts 3x684 και έναν contacts 1x684. Στον πίνακα pts έχουμε τις συντεταγμένες 684 σημείων σε x,y,z στο παγκόσμιο επίπεδο (world frame). Στον πίνακα contacts έχουμε την πληροφορία για το ποια από αυτά τα 684 σημεία βρίσκονται σε απόσταση μικρότερη από πέντε μετρά και κατ' επέκταση αντιστοιχούν σε κάποιο αντικείμενο.

Στη συνέχεια ο κώδικας δημιουργεί έναν τρίτο πίνακα 3xN, όπου N είναι το πλήθος των σημείων που βρίσκονται σε απόσταση μικρότερη από πέντε μέτρα (contacts(i=1:684)=1), με τις συντεταγμένες των σημείων του πίνακα pts.

4. Επιλογή της ελεύθερης περιοχής εντός της περιοχής αναζήτησης του αισθητήρα

Για την επιλογή της ελεύθερης περιοχής που είδε το ρομπότ χρησιμοποιήθηκε αρχικά η εντολή raycast. Με αυτή την εντολή ο κώδικας διάλεγε όλα τα σημεία που βρίσκονταν επάνω σε μία ευθεία που ένωνε δύο σημεία. Στη δικιά μας εφαρμογή αυτή η διαδικασία έπρεπε να γίνει για 684 διαφορετικά σημεία και απαιτούσε πάρα πολύ χρόνο.

Μία πιο ικανοποιητική προσέγγιση είναι η εντολή inpolygon. Σε αυτή την εντολή ο κώδικας ενώνει το σημείο του αισθητήρα με τα υπόλοιπα 684, δημιουργώντας ένα μεγάλο πολύγωνο. Χρησιμοποιώντας αυτό το πολύγωνο διαλέγουμε όλα τα σημεία που βρίσκονται εντός του, έχοντας έτσι την ελεύθερη περιοχή.

5. Δημιουργία πλέγματος πληρότητας

Αρχικά δημιουργούμε με την εντολή robotics.OccupancyGrid(16,16,10) ένα πλέγμα πληρότητας διαστάσεων 16x16 μέτρα και ανάλυση 10 δηλαδή 10 κελιά ανά 1 μέτρο ή 10x10 ανά 1 τετραγωνικό μέτρο. Οι συντεταγμένες του χάρτη ξεκινάνε από το (-8,-8). Τα κελιά παίρνουν τις τιμές: -1 άγνωστο, 0 ελεύθερο, 1 κατειλημμένο.

Έπειτα γεμίζουμε το πλέγμα με τις πληροφορίες που έχουμε από τον αισθητήρα μας με την εντολή setOccupancy.

6. Κίνηση ανίχνευσης

Η καταγραφή και ανανέωση του χάρτη είναι χρονοβόρες και έχουν υψηλό υπολογιστικό κόστος στην εφαρμογή μας. Για τον λόγο αυτό γίνονται μόνο κατά την κίνηση ανίχνευσης και όχι συνέχεια. Αυτή η κίνηση γίνεται όταν το ρομπότ μας βρίσκεται στη θέση εξερεύνησης. Κατά την κίνηση το ρομπότ μας κάνει μια πλήρη περιστροφή στο ίδιο σημείο καταγράφοντας τις αλλαγές στο πλέγμα πληρότητας.

7. Επιλογή σημείου εξερεύνησης

Η επιλογή σημείου εξερεύνησης είναι μια σύνθετη διαδικασία στην εφαρμογή μας. Αρχικά ο κώδικας βρίσκει όλα τα σημεία στον χάρτη που πληρούν τις παρακάτω προϋποθέσεις:

- το σημείο αντιστοιχεί σε κελί με τιμή 0 δηλαδή είναι ελεύθερο
- το αμέσως επόμενο ή προηγούμενο κελί έχει τιμή -1 δηλαδή είναι άγνωστο

Ο κώδικας στη συνέχεια διαλέγει τυχαία 10 από αυτά και τέλος από αυτά τα 10 διαλέγει το πιο κοντινό στο ρομπότ μας και το καθορίζει ως σημείο εξερεύνησης. Αν για τον οποιοδήποτε λόγο δεν μπορεί να βρεθεί διαδρομή προς αυτό το σημείο τότε η παραπάνω διαδικασία επαναλαμβάνεται.

8. Εύρεση διαδρομής

Για την εύρεση της διαδρομής χρησιμοποιούμε την μέθοδο PRM που υπάρχει στο Robotics toolbox του Matlab. Αρχικά φτιάχνουμε ένα αντικείμενο prm με την εντολή robotics.PRM και καθορίζουμε ότι θα έχουμε 400 nodes και μέγιστη απόσταση διαδρομής μεταξύ των nodes 1.5. Έπειτα τροφοδοτούμε τον αλγόριθμο με την θέση του ρομπότ μας και το σημείο εξερεύνησης που επιλέξαμε προηγουμένως. Η διαδικασία αυτή δημιουργεί έναν πίνακα με τις συντεταγμένες κάθε node της διαδρομής, τις οποίες χρησιμοποιούμε στην κίνηση του ρομπότ.

9. Περιστροφή ρομπότ

Στην περιστροφή του ρομπότ χρησιμοποιούμε κυρίως την επιθυμητή γωνία προσανατολισμού και την πραγματική γωνία προσανατολισμού του ρομπότ μας. Η σύγκριση αυτών των γωνιών γίνεται συνέχεια κατά την περιστροφή και η διαφορά τους επηρεάζει την ταχύτητά της. Όταν η διαφορά τους είναι πλέον μηδέν τότε η περιστροφή σταματά.

10. Κίνηση ρομπότ

Στη κίνηση του ρομπότ χρησιμοποιείται κυρίως η θέση του και η επιθυμητή θέση. Επίσης χρησιμοποιούνται ακριβώς με τον ίδιο τρόπο όπως και στην κίνηση της περιστροφής οι γωνίες προσανατολισμού για τυχόν διορθώσεις στο προσανατολισμό του ρομπότ κατά την κίνηση.

Η ταχύτητα του ρομπότ εξαρτάται από την απόσταση μεταξύ της θέσης του ρομπότ και της επιθυμητής θέσης. Η κίνηση τελειώνει όταν η πραγματική και η επιθυμητή θέση είναι ίδιες.

11. Αποστολή εντολών ταχυτήτων στο ρομπότ

Καθ' όλη τη διάρκεια λειτουργίας του κώδικα, οι όποιες αλλαγές στις ταχύτητες, γίνονται αλλάζοντας τις μεταβλητές τους forwBackVel, leftRightVel και rotVel οι οποίες αφορούν τις κινήσεις μπροστά-πίσω, στο πλάι αριστερά-δεξιά και περιστροφή αντίστοιχα.

Στο τέλος κάθε κύκλου του κώδικα η εντολή youbot_drive χρησιμοποιεί αυτές τις μεταβλητές για να στείλει τις πληροφορίες στο V-Rep και από εκεί στο youbot.

Τέλος σημειώνεται ότι οι ταχύτητες έχουν μέγιστες τιμές, 12 για την ταχύτητα κίνησης και 4 για την ταχύτητα περιστροφής. Αυτά τα όρια δεν ξεπερνούνται ακόμα και όταν μέσα στο πρόγραμμα ο υπολογισμός της ταχύτητας επιστρέψει κάποια μεγαλύτερη τιμή από την μέγιστη.

12. Τέλος προγράμματος

Το πρόγραμμά μας τελειώνει όταν αποτύχει να βρεθεί άγνωστο σημείο στον χάρτη που να είναι προσβάσιμο για πάνω από δέκα προσπάθειες.

6.6 Κώδικας

Σε αυτή την ενότητα παρουσιάζεται ο κυρίως κορμός του κώδικα χωρίς περαιτέρω λεπτομέρειες για τα function που χρησιμοποιήθηκαν μέσα σε αυτόν.

Σύνδεση Matlab με V-Rep και έναρξη εξομοίωσης.

```
clear all
clc

disp('Program started');
vrep = remApi('remoteApi');
vrep.simxFinish(-1);
id = vrep.simxStart('127.0.0.1', 19997, true, true, 2000, 5);


if id < 0
    disp('Failed connecting to remote API server. Exiting.');
    vrep.delete();
    return;
end
fprintf('Connection %d to remote API server open.\n', id);

cleanupObj = onCleanup(@() cleanup_vrep(vrep, id));


vrep.simxStartSimulation(id, vrep.simx_opmode_oneshot_wait);
```

Ανάκτηση των handles του youbot από το V-Rep με την βοήθεια των factions youbot_init και youbot_hokuyo_init.

```
h = youbot_init(vrep, id);
h = youbot_hokuyo_init(vrep, h);
```

Η εξομοίωση τρέχει σε συγχρονισμό με το Matlab και περιμένει πότε θα τελειώσει ο αλγόριθμος για να συνεχίσει.

```
pause(2);
vrep.simxSynchronous(id,true);
```

Παράμετροι για τον έλεγχο της κίνησης του youbot.

```
forwBackVel = 0;
leftRightVel = 0;
rotVel = 0;
prevOri = 0;
```

Βοηθητικοί παράμετροι για τον έλεγχο των διαφόρων σταδίων του κώδικα.

```
fsm = 'scan';
scan_stage = 0;
path_stage = 1;
finish = 0;
```

Αρχικοποίηση του χάρτη (occupancy grid). Ο χάρτης έχει διαστάσεις 16x16 και οι συντεταγμένες του ξεκινάνε από το -8,-8.

```
map = robotics.OccupancyGrid(16,16,10);
map.GridLocationInWorld = [-8 -8];
```

Δημιουργία meshgrid για την βοήθεια καθορισμού της εξερευνημένης περιοχής με χρήση του function inpolygon_for_gpu.

```
 [XX, YY] = meshgrid(-8:.08:8, -8:.08:8);
XX = reshape(XX, 1, []);
YY = reshape(YY, 1, []);
XY = [XX; YY]';
```

Ο κυρίως κορμός του κώδικα τρέχει μέσα σε ένα while loop. Το πρώτο στάδιο του κώδικα είναι το scan. Σε αυτό αρχικά παίρνουμε την θέση και τον προσανατολισμό του youbot. Το function vrchk μας βοηθάει στο debugging σε τυχόν σφάλματα του V-Rep.

```
while true
if strcmp(fsm, 'scan')
[res, youbotPos] = vrep.simxGetObjectPosition(id, h.ref, -1, vrep.simx_opmode_buffer);
vrchk(vrep, res, true);
[res, youbotEuler] = vrep.simxGetObjectOrientation(id, h.ref, -1,
vrep.simx_opmode_buffer);
vrchk(vrep, res, true);
```

Καθορισμός της θέσης του αισθητήρα Hokuyo σε global συντεταγμένες (world reference frame).

```
trf = transl(youbotPos) * trotx(youbotEuler(1)) * troty(youbotEuler(2)) *
trotz(youbotEuler(3));
```

Εισαγωγή των δεδομένων του αισθητήρα με χρήση του function youbot_hokuyo. Ο πίνακας pts έχει τις συντεταγμένες όλων των σημείων που βλέπει ο αισθητήρας. Ο πίνακας contacts περιέχει πληροφορίες για ποια από αυτά τα σημεία βρίσκονται σε απόσταση μικρότερη από 5μ, δηλαδή ποια σημεία είναι εμπόδια.

```
 [pts, contacts] = youbot_hokuyo(vrep, h, vrep.simx_opmode_buffer, trf);
```

Μετατροπή των παραπάνω δεδομένων για χρήση τους στο occupancy grid.

```
size_contacts = 0;
for i=1:684
   if contacts(i) == 1
      size_contacts = size_contacts+1;
   end
end
x = zeros(2,size_contacts);
j = 1;
for l=1:684
   if contacts(l) == 1
      x(1,j) = pts(1, l);
      x(2,j) = pts(2, l);
      j = j + 1;
   end
end
X = x';
```

Δημιουργία πίνακα με τα σημεία που εξερευνήθηκαν με την βοήθεια του function inpolygon_for_gpu και του mesh grid. Πιο αναλυτικά το inpolygon_for_gpu δημιουργεί ένα πολύγωνο ενώνοντας την θέση του youbot και τα σημεία που είδε ο αισθητήρας Hokuyo. Έπειτα φτιάχνουμε έναν πίνακα με τις συντεταγμένες των σημείων του mesh grid που βρίσκονται μέσα στο πολύγωνο.

```
in = inpolygon_for_gpu(XX, YY, [youbotPos(1), pts(1, :), youbotPos(1)],[youbotPos(2),
pts(2, :), youbotPos(2)]);
in_logical = logical(in);
XY_in = [XX(in_logical)' YY(in_logical)'];
```

Δημιουργία του χάρτη (occupancy grid) με την βοήθεια των δεδομένων που επεξεργασθήκαμε παραπάνω (εμπόδια 1, ελεύθερος χώρος 0, άγνωστο -1).

```
setOccupancy(map, XY_in, 0);
setOccupancy(map, X, 1);
```

Περιστροφή 360 μοιρών. Η ταχύτητα περιστροφής ανανεώνεται συνεχώς με βάση την διαφορά της πραγματικής γωνίας προσανατολισμού του youbot και την επιθυμητή.

```
if scan_stage == 0
temp_angle1 = youbotEuler(3) + pi;
scan_stage = 1;
end
if scan_stage == 1

rotVel = angdiff(temp_angle1, youbotEuler(3));

if (abs(angdiff(temp_angle1, youbotEuler(3))) < .1 / 180 * pi) && ...
(abs(angdiff(prevOri, youbotEuler(3))) < .01 / 180 * pi)
rotVel = 0;
temp_angle2 = youbotEuler(3)+pi;
```

```
scan_stage = 2;
end
prevOri = youbotEuler(3);
end
if scan_stage == 2
rotVel = angdiff(temp_angle2, youbotEuler(3));

if (abs(angdiff(temp_angle2, youbotEuler(3))) < .1 / 180 * pi) && ...
(abs(angdiff(prevOri, youbotEuler(3))) < .01 / 180 * pi)
rotVel = 0;
fsm = 'target';
scan_stage = 3;
end
prevOri = youbotEuler(3);
end
end
```

Το επόμενο στάδιο του κώδικα είναι η επιλογή του σημείου προς εξερεύνηση. Αρχικά μεγαλώνουμε τα εμπόδια στον χάρτη για αποφυγή επιλογής σημείου πολύ κοντά σε αυτά.

```
    if strcmp(fsm, 'target')
        cd_point = 0;
        inflated_map = copy(map);
        inflate(inflated_map,0.2);
```

Δημιουργία πίνακα με τις τιμές κατάστασης κάθε σημείου του χάρτη.

```
iOccval = checkOccupancy(inflated_map,XY);
```

Καταμέτρηση των ελεύθερων σημείων του χάρτη (δηλαδή με τιμή μηδέν) που έχουν δίπλα τους σημεία με άγνωστη κατάσταση (δηλαδή με τιμή μείον ένα).

```
for i=2:length(iOccval)
   if iOccval(i) == 0 && (iOccval(i+1) < 0 || iOccval(i-1)<0)
      cd_point = cd_point + 1;
   end
end
```

Αν υπάρχουν σημεία που πληρούν τις παραπάνω προϋποθέσεις τότε αρχικά φτιάχνουμε έναν πίνακα με τις συντεταγμένες αυτών.

```
if cd_point>0
  candidate = zeros(cd_point,2);
  cd_pos = 1;
  for i=2:length(iOccval)
```

```
   if iOccval(i) == 0 && (iOccval(i+1) < 0 || iOccval(i-1)<0)
     candidate(cd_pos,:) = XY(i,:);
     cd_pos = cd_pos + 1;
   end
  end
```

Προεπιλογή του target με το πρώτο σημείο του πίνακα και υπολογισμός απόστασης του από το ρομπότ.

```
cd_pos = cd_pos - 1;
target = candidate(1,:);
dist_cand = pdist([youbotPos(1), youbotPos(2); candidate(1,1), candidate(1,2)]);
```

Επιλογή 10 τυχαίων σημείων. Σύγκριση της απόστασης μεταξύ όλων των επιλεγμένων σημείων και επιλογή του κοντινότερου για target. Η επιλογή τυχαίων σημείων και ο έλεγχος της μεταβλητής finish γίνονται για αποφυγή σφαλμάτων παρακάτω στον κώδικα.

```
if finish<6
for i=1:10
rnd = randi([1 cd_pos],1,1);
if dist_cand > pdist([youbotPos(1), youbotPos(2);
candidate(rnd,1), candidate(rnd,2)])
target = candidate(rnd,:);
end
end
else
rnd = randi([1 cd_pos],1,1);
target = candidate(rnd,:);
end
fsm = 'pathfind';
```

Σε περίπτωση που δεν βρεθούν άλλα σημεία το πρόγραμμα σταματάει και εμφανίζεται το ανάλογο μήνυμα.

```
else
show(map);
fprintf('Unable to find target, mapping ended. \n');
break;
end
end
```

Στάδιο pathfinding. Αρχικά δημιουργούμε το prm δίνοντάς του ένα χάρτη, το πλήθος των nodes, την μέγιστη απόσταση σύνδεσης μεταξύ των nodes, και τέλος την αρχική και τελική θέση.

```
   if strcmp(fsm, 'pathfind')
      prm = robotics.PRM;
      inflated_map = copy(map);
```

```
inflate(inflated_map,0.4);
prm.Map = inflated_map;
prm.NumNodes = 400;
prm.ConnectionDistance = 1.5;
startLocation = double([youbotPos(1) youbotPos(2)]);
endLocation = target;
```

Αν η επιλογή του target είναι καλή και δεν έχουμε κάποιο σφάλμα τότε εμφανίζουμε τον χάρτη και το PRM και προχωράμε στο επόμενο στάδιο. Αν η επιλογή του target μας επιστρέψει σφάλμα πάνω από 10 φορές τότε η χαρτογράφηση τελειώνει.

```
try
        path = findpath(prm, startLocation, endLocation);
        subplot(1,2,1);
        show(map); (Εικόνα 6.10)
        title('Map');
        subplot(1,2,2);
        show(prm); (Εικόνα 6.11)
        drawnow;
        %figure;
        path_stages = length(path);
        path_stage = 2;
        scan_stage = 0;
        fsm = 'path_check';
        finish  = 0;
    catch
        fsm = 'target';
        finish = finish + 1;
        if finish>10
            inflated_map_finished = copy(map);
            inflate(inflated_map_finished,0.1);
            show(inflated_map_finished);
            title('Finished Map'); (Εικόνα 6.13)
            fprintf('Map finished. \n');
            break;
        end
    end
end
```

Βοηθητικό στάδιο για την επιλογή των rotate και scan.

```
if strcmp(fsm, 'path_check')
if path_stage<=path_stages-2
theta = atan2(path(path_stage,2)-youbotPos(2),path(path_stage,1)-youbotPos(1))+pi/2;
fsm = 'rotate';
else
fsm = 'scan';
end
end
```

Σε αυτό το κομμάτι του κώδικα γίνεται ο έλεγχος της περιστροφής του ρομπότ μας. Γίνεται σύγκριση της επιθυμητής γωνίας και της πραγματικής γωνίας του ρομπότ και βάση αυτής υπολογίζεται η ταχύτητα περιστροφής. Όταν η διαφορά τους είναι πολύ μικρή (σχεδόν μηδέν) τότε η περιστροφή του ρομπότ σταματάει.

```
if strcmp(fsm, 'rotate')
      [res, youbotPos] = vrep.simxGetObjectPosition(id, h.ref, -1,
vrep.simx_opmode_buffer);
      vrchk(vrep, res, true);
      [res, youbotEuler] = vrep.simxGetObjectOrientation(id, h.ref, -1,
vrep.simx_opmode_buffer);
      vrchk(vrep, res, true);
      rotVel = angdiff(theta, youbotEuler(3));


      if (abs(angdiff(theta, youbotEuler(3))) < .1 / 180 * pi) && ...
            (abs(angdiff(prevOri, youbotEuler(3))) < .01 / 180 * pi)
         rotVel = 0;
         fsm = 'forward';
      end

      prevOri = youbotEuler(3);
   end
```

Έλεγχος της κίνησης του ρομπότ. Η κίνηση ξεκινάει μόνο όταν η ταχύτητα περιστροφής είναι πολύ μικρότερη της ταχύτητας κίνησης. Ο υπολογισμός της ταχύτητας κίνησης γίνεται με την απόσταση μεταξύ της πραγματικής και της επιθυμητής θέσης του ρομπότ.

```
if strcmp(fsm, 'forward')
      [res, youbotPos] = vrep.simxGetObjectPosition(id, h.ref, -1,
vrep.simx_opmode_buffer);
      vrchk(vrep, res, true);
      [res, youbotEuler] = vrep.simxGetObjectOrientation(id, h.ref, -1,
vrep.simx_opmode_buffer);
      vrchk(vrep, res, true);
      theta = atan2(path(path_stage,2)-youbotPos(2),path(path_stage,1)-youbotPos(1))+pi/2;

      rotVel = angdiff(theta, youbotEuler(3));
      if 100*rotVel < 0.6*pdist([youbotPos(1), youbotPos(2); path(path_stage,1),
path(path_stage,2)])
         forwBackVel = -0.6*pdist([youbotPos(1), youbotPos(2); path(path_stage,1),
path(path_stage,2)]);
      else
         forwBackVel = 0;
      end

      if path_stage == path_stages - 2
         if abs(forwBackVel) < 0.001
            forwBackVel = 0;
            rotVel = 0;
```

```
            path_stage = path_stage + 1;
            fsm = 'path_check';
        end
    else
        if pdist([youbotPos(1), youbotPos(2); path(path_stage,1), path(path_stage,2)])<0.05
            path_stage = path_stage + 1;
            fsm = 'path_check';
        end
    end
end
```

Στο τέλος του προγράμματος καλούμε το function youbot_drive και του δίνουμε της ταχύτητες περιστροφής και κίνησης. Η εντολή vrep.simxSynchronousTrigger(id); συγχρονίζει το Matlab με το V-Rep.

```
    h = youbot_drive(vrep, h, forwBackVel, leftRightVel, rotVel);


    vrep.simxSynchronousTrigger(id);
    if strcmp(fsm, 'end')
        break;
    end
end % main function
```

Συμπέρασμα

Σε αυτήν την εργασία είδαμε τα προβλήματα της πλοήγησης αυτόνομων ρομποτικών συστημάτων σε άγνωστο περιβάλλον και τις λύσεις τους όπως η χρήση του SLAM, των φίλτρων και των μεθόδων αναζήτησης διαδρομής.

Εφαρμόσαμε κάποιες από αυτές τις λύσεις επιτυγχάνοντας την αυτόνομη πλοήγηση του ρομποτικού συστήματος youbot από την Kuka στο εικονικό περιβάλλον του V-Rep. Αυξάνοντας με αυτό το τρόπο τις γνώσεις και την κατανόησή μας τόσο του προβλήματος SLAM όσο και του προγράμματος Matlab.

Το επόμενο βήμα για την περαιτέρω εξέλιξη του συστήματος που αναπτύχθηκε είναι η βελτίωση του κώδικα, η χρήση οδομετρίας και κάποιου φίλτρου για τον υπολογισμό της θέσης του ρομπότ.

Βιβλιογραφία

[1] Timeline of Robotics. http://www.thocp.net/reference/robotics/robotics.html (προσπελάστηκε 16/07/2018)

[2] Industrial robot. http://en.wikipedia.org/wiki/Industrial_robot (προσπελάστηκε 16/07/2018)

[3] Domestic or household robots. http://www.allonrobots.com/household-robots.html (προσπελάστηκε 16/07/2018)

[4] Medical robots. http://www.allonrobots.com/medical-robots.html (προσπελάστηκε 16/07/2018)

[5] Military robots. http://www.allonrobots.com/military-robots.html (προσπελάστηκε 16/07/2018)

[6] Entertainment robots. http://www.allonrobots.com/types-of-robots.html (προσπελάστηκε 16/07/2018)

[7] Exploration robots. http://i-heart-robots.blogspot.com/2005/10/robots-forexploration.html (προσπελάστηκε 16/07/2018)

[8] H. Durrant-Whyte, T. Bailey, Simultaneous localization and mapping (SLAM): Part I. The essential algorithms, Robot. Automat. Mag., vol. 13, no. 2, pp. 99-110, 2006.

[9] H. Durrant-Whyte, T. Bailey, Simultaneous localization and mapping (SLAM): Part II. State of the art, Robot. Automat. Mag., vol. 13, no. 3, pp. 108-117, 2006.

[10] Michael Montemerlo , Sebastian Thrun , Daphne Koller , Ben Wegbreit, FastSLAM: a factored solution to the simultaneous localization and mapping problem, Eighteenth national conference on Artificial intelligence, p.593-598, 2002.

[11] L. Kavraki, P. Svestka, J.-C. Latombe, M. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, pp. 7-16, 1994

[12] Arkin, Ronald C. "Reactive robotic systems." 1995

[13] Urdiales, Cristina, et al. "A purely reactive navigation scheme for dynamic environments using Case-Based Reasoning." Autonomous Robots 21.1 2006

[14] Corke, Peter. Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised. Vol. 118. Springer, 2017.

[15] González-Nalda, Pablo, and Blanca Cases. "Topos: generalized Braitenberg vehicles that recognize complex real sounds as landmarks." Proceedings of the 10th international conference on the simulation and synthesis of living systems. Cambridge, MA: Mit Press, 2006.

[16] Ng, James, and Thomas Bräunl. "Performance comparison of bug navigation algorithms." Journal of Intelligent and Robotic Systems 50.1 (2007): 73-84.

[17] Borgefors, Gunilla. "Distance transformations in digital images." Computer vision, graphics, and image processing 34.3 (1986): 344-371.

[18] Stentz, Anthony. The D* Algorithm for Real-Time Planning of Optimal Traverses. No. CMU-RI-TR-94-37. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1994.

[19] Masehian, Ellips, and M. R. Amin-Naseri. "A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning." Journal of Robotic Systems 21.6 (2004): 275-300.

[20] LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998).

[21] http://www.coppeliarobotics.com/helpFiles/ (προσπελάστηκε 16/07/2018)

[22] https://www.mathworks.com/discovery/what-is-matlab.html (προσπελάστηκε 16/07/2018)

[23] http://www.youbot-store.com/wiki/index.php/YouBot_Detailed_Specifications (προσπελάστηκε 16/07/2018)

[24] https://www.hokuyo-usa.com/application/files/5414/7196/1896/URG-04LX-UG01_Specifications_Catalog.pdf (προσπελάστηκε 16/07/2018)