ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ: Τηλεπικοινωνιών και Τεχνολογίας Πληροφορίας (Τ. & Τ.Π.)

Διπλωματική Εργασία

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών της Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών

Μπότη Νικολάου του Ιωάννη
Αριθμός Μητρώου : 228426

Θέμα:

«Χαρτογράφηση Χώρου με Χρήση Κινούμενου Ρομποτικού Οχήματος»

Επιβλέπων

Δερματάς Ευάγγελος (Αναπληρωτής Καθηγητής)

Αριθμός Διπλωματικής Εργασίας:

Πάτρα, Οκτώβριος 2019

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με θέμα:

«Χαρτογράφηση Χώρου με Χρήση Κινούμενου Ρομποτικού Οχήματος»

Του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Μπότη Νικολάου του Ιωάννη
Αριθμός Μητρώου : 228426

Παρουσιάστηκε δημόσια και εξετάστηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις
……../……../……

Ο Επιβλέπων

Δερματάς Ευάγγελος
Αναπληρωτής Καθηγητής

Ο Διευθυντής του Τομέα

Μουρτζόπουλος Ιωάννης
Καθηγητής

Αριθμός Διπλωματικής Εργασίας:

Θέμα: «Χαρτογράφηση Χώρου με Χρήση Κινούμενου Ρομποτικού Οχήματος»

Φοιτητής: Επιβλέπων:

Μπότης Νικόλαος **Δερματάς Ευάγγελος**
Αναπληρωτής Καθηγητής

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία έχει ως αντικείμενο τον προγραμματισμό και τον έλεγχο ενός ρομποτικού οχήματος E-puck σε περιβάλλον προσομοίωσης. Στόχος είναι η χαρτογράφηση του περιβάλλοντος στο οποίο βρίσκεται, μέσα από την επεξεργασία των δεδομένων που λαμβάνει από τους αισθητήρες και τη χρήση κατάλληλων αλγορίθμων. Χρησιμοποιώντας τους 8 αισθητήρες απόστασης υπερύθρων που διαθέτει, λαμβάνονται πληροφορίες για το κοντινό του περιβάλλον και γίνεται εντοπισμός των εμποδίων που υπάρχουν σε αυτό. Για τη χαρτογράφηση του χώρου είναι απαραίτητο να γνωρίζουμε τη θέση του ρομποτικού οχήματος κάθε στιγμή, με όσο το δυνατόν μεγαλύτερη ακρίβεια, πράγμα που επιτυγχάνεται με τη χρήση των encoders που είναι ενσωματωμένοι στους τροχούς του. Για την υλοποίηση των αλγορίθμων Ταυτόχρονου Εντοπισμού Θέσης και Χαρτογράφησης (SLAM) χρησιμοποιήθηκε το περιβάλλον Robot Operating System (ROS), ενώ για την προσομοίωση του χώρου και της συμπεριφοράς του ρομπότ χρησιμποιήθηκε το πρόγραμμα προσομοίωσης Webots.

ABSTRACT

The subject of the present diploma thesis is the programming and control of an E-puck robot vehicle in a simulator environment. Our aim is the mapping of its environment, through the processing of data received from its sensors and the use of appropriate algorithms. By using its 8 infrared proximity sensors, information about its environment is obtained and localization of surrounding obstacles is achieved. In order for the map to be created, it is necessary that the pose of the robotic vehicle is known in every single moment, as precisely as possible, which is accomplished through the use of its embedded wheel encoders. The Simultaneous Localization and Mapping (SLAM) algorithms were implemeted in the Robot Operating System (ROS) framework, while the simulation of the environment and the robot's behaviour was done using the Webots simulator.

ΕΥΧΑΡΙΣΤΙΕΣ

Για την παρούσα εργασία θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή Ευάγγελο Δερματά, ο οποίος μου πρότεινε το θέμα και μου παρείχε χρήσιμες συμβουλές αφιερώνοντας πολύτιμο χρόνο, καθιστώντας έτσι δυνατή την ολοκλήρωσή της.

Επιπλέον, θα ήθελα να ευχαριστήσω τους φίλους και την οικογένειά μου για τη στήριξη που μου παρείχαν καθόλη τη διάρκεια των σπουδών μου.

Πίνακας Περιεχομένων

1

Εισαγωγή

Η ρομποτική αποτελεί έναν αυτοδύναμο τεχνολογικό κλάδο, με αντικείμενο τη μελέτη , το σχεδιασμό και τη λειτουργία των ρομπότ. Η ρομποτική σήμερα έχει εφαρμογές στην πλειονότητα των βιομηχανιών και βιοτεχνιών. Οι εφαρμογές της ρομποτικής συνεισφέρουν στη μείωση του κόστους, την αύξηση της παραγωγικότητας και τη βελτίωση της ποιότητας των παραγόμενων προϊόντων και επιπλέον απαλλάσσουν τον άνθρωπο από πολλές επικίνδυνες και ανθυγιεινές εργασίες.

Το ρομπότ είναι ένας προγραμματιζόμενος και πολυλειτουργικός μηχανισμός σχεδιασμένος να υλοποιεί συγκεκριμένες ενέργειες με κατάλληλα προγραμματιζόμενες κινήσεις που στοχεύουν στη βελτίωση της απόδοσης μιας σειράς εργασιών. Είναι ουσιαστικά οποιαδήποτε μηχανική συσκευή - ειδικά προγραμματιζόμενη από υπολογιστή - που μπορεί να υποκαθιστά τον άνθρωπο , σε διάφορες εργασίες. Ένα ρομπότ μπορεί να δράσει κάτω από τον απευθείας έλεγχο ενός ανθρώπου ή αυτόνομα κάτω από τον έλεγχο ενός προγραμματισμένου υπολογιστή. Τα ρομπότ χωρίζονται σε τρεις κατηγορίες. Το μεγαλύτερο

μέρος από αυτά είναι βιομηχανικά ρομπότ. Ένα μικρότερο αλλά σημαντικό μέρος καταλαμβάνουν τα ρομπότ που χρησιμοποιούνται σε στρατιωτικές εφαρμογές και τα κινητά ρομπότ (mobile robots).

Στην παρούσα εργασία θα ασχοληθούμε με την τελευταία κατηγορία, δηλαδή τα κινητά ρομπότ κι έναν συνεχώς αναπτυσσόμενο τομέα τους, τη χαρτογράφηση. Μερικοί τομείς που σχετίζονται άμεσα με τη χαρτογράφηση χώρου από αυτόνομα οχήματα είναι η μεταλλευτική βιομηχανία, η υποβρύχια εξερεύνηση, η εξερεύνηση άλλων πλανητών, ενώ βρίσκει εφαρμογή ακόμη και σε οικιακές συσκευές. Ένα τέτοιο παράδειγμα αποτελούν οι όλο και πιο διαδεδομένες πλέον αυτόνομες σκούπες, οι οποίες χαρτογραφούν έναν εσωτερικό χώρο με σκοπό αργότερα τον καθαρισμό του.

Πιο συκεκριμένα σκοπός της εργασίας είναι η εφαρμογή ενός αλγορίθμου ταυτόχρονου εντοπισμού θέσης και χαρτογράφησης (SLAM) σε ένα μικρής κλίμακας ρομποτικό όχημα (E-puck) που χρησιμοποιεί το Robot Operating System (ROS) σε περιβάλλον προσομοίωσης. Η επιλογή του αλγορίθμου SLAM που χρησιμοποιήθηκε έγινε με βάση μια συνοπτική σύγκριση μεταξύ τριών αλγορίθμων, του EKF, του FastSLAM και του GraphSLAM. Και οι τρεις είναι υλοποιημένοι σε πακέτα του ROS, όμως οι δύο τελευταίοι προτιμώνται γενικώς λόγω της υψηλότερης ευρωστίας και απόδοσής τους. Η υλοποίηση του FastSLAM στο πακέτο gmapping επιλέχθηκε τελικά καθώς είναι η πιο δημοφιλής και παρέχεταιι μεγαλύτερη υποστήριξη.

Ο αλγόριθμος θα πρέπει να δημιουργεί το χάρτη του περιβάλλοντος του ρομποτικού οχήματος και ταυτόχρονα να εντοπίζει τη θέση του οχήματος μέσα σε αυτό το χάρτη, ενώ θα πρέπει να παρέχεται κατάλληλος έλεγχος του οχήματος ώστε να κινείται στο περιβάλλον του.

2

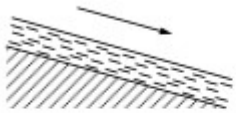
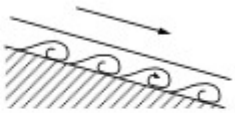
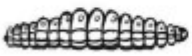
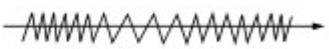
Κινούμενα Ρομποτικά Οχήματα

2.1 Εισαγωγή

Η ικανότητα της κίνησης συναντάται στα περισσότερα από τα είδη ζώων που υπάρχουν στη φύση, και ο τρόπος με τον οποίο επιτυγχάνεται συνήθως διαφέρει ανάλογα με το περιβάλλον στο οποίο ζει το κάθε είδος. Κάποια ζώα χρησιμοποιούν παθητικά συστήματα κίνησης (μετακινούνται χρησιμοποιώντας την κίνηση του νερού ή του αέρα), ενώ κάποια άλλα έχουν αναπτύξει πιο σύνθετους μηχανισμούς ενεργητικής κίνησης. Υπάρχουν είδη που πραγματοποιούν κίνηση στις 3 διαστάσεις του χώρου (όπως κολυμπώντας μεσα σε νερό, πετώντας στον αέρα ή κινούμενα μέσα στη γη), άλλα που κυρίως μετακινούνται σε

δισδιάστατες επιφάνειες του νερού ή του εδάφους και μερικά που μπορούν να συνδυάσουν διαφορετικούς από τους παραπάνω τρόπους κινησης.

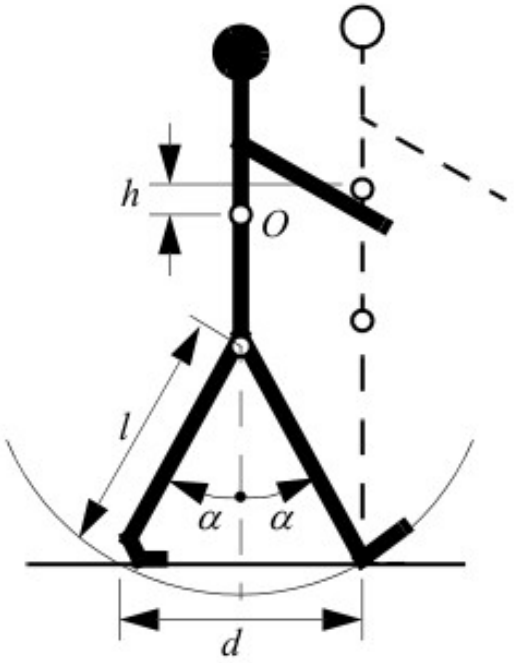
Όσον αφορά τα κινούμενα ρομπότ, ασχολούμαστε με συστήματα που κινούνται χρησιμοποιώντας δικούς τους μηχανισμούς κίνησης. Τις περισσότερες φορές αυτοί οι μηχανισμοί μιμούνται την κίνηση του ανθρώπου ή κάποιου ζώου. Κάποιοι από αυτούς φαίνονται παρακάτω στην εικόνα 2.1, υπάρχουν βέβαια πολλοί ακόμη, όπως για παράδειγμα η κίνηση με 6 πόδια που πραγματοποιούν τα έντομα κ.ά.

Type of motion	Resistance to motion	Basic kinematics of motion
Flow in a Channel 	Hydrodynamic forces	Eddies 
Crawl 	Friction forces	Longitudinal vibration 
Sliding 	Friction forces	Transverse vibration 
Running 	Loss of kinetic energy	Periodic bouncing on a spring 
Walking 	Loss of kinetic energy	Rolling of a polygon (see figure 2.2) 

Εικόνα 2.1 Μηχανισμοί κίνησης που χρησιμοποιούνται σε βιολογικά συστήματα (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

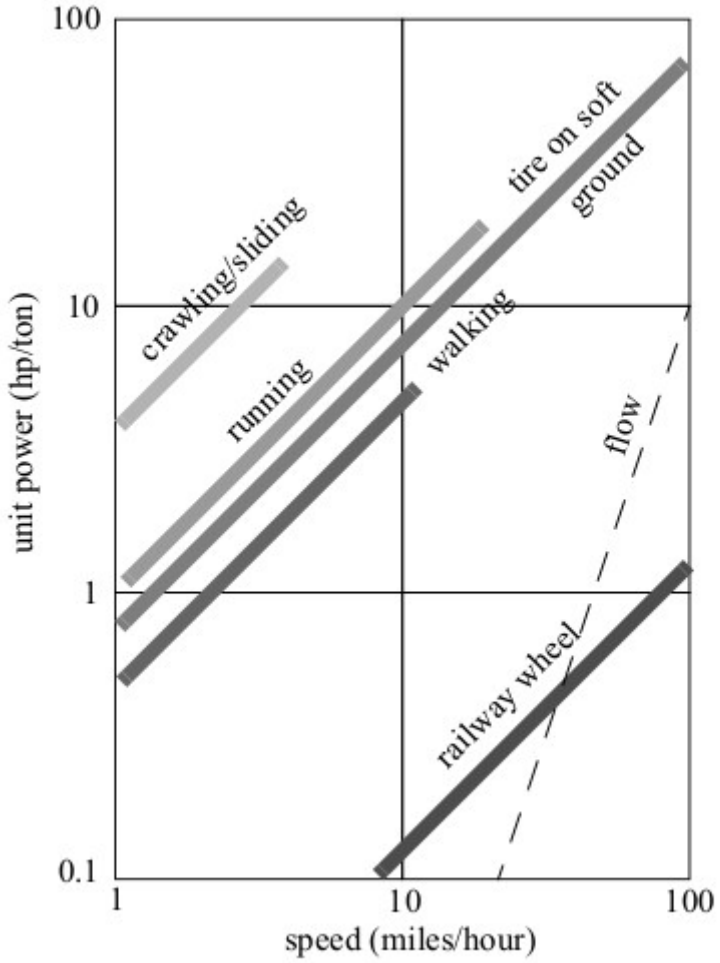
Οι μηχανισμοί των ρομπότ με πόδια αποτελούν ένα τέτοιο παράδειγμα και μάλιστα πολύ αποδοτικό σε ένα μεγάλο εύρος δυσπρόσιτων περιβάλλοντων. Η προσέγγιση όμως τέτοιων μηχανισμών έχει αποδειχθεί αρκετά δύσκολη για διάφορους λόγους, όπως είναι η μηχανική πολυπλοκότητα των ποδιών, η ευστάθεια και η κατανάλωση ενέργειας. Έτσι σε πολλά κινητά ρομπότ χρησιμοποιούνται τροχοί για την κίνησή τους.

Ο τροχός αποτελεί μια ανθρώπινη εφεύρεση που αποδίδει εξαιρετικά σε ομαλό έδαφος. Βέβαια ο μηχανισμός του τροχού δεν είναι τελείως ξένος στα βιολογικά συστήματα που συναντώνται στη φύση, καθώς το περπάτημα με δύο πόδια μπορεί να προσεγγιστεί από ένα κυλιόμενο πολύγωνο, πλευράς ίσης με το άνοιγμα του βήματος (Εικόνα 2.2). Η μείωση του μήκους του βήματος σημαίνει πως το πολύγωνο τείνει να γίνει κύκλος, πλησιάζοντας την κίνηση του τροχού.



Εικόνα 2.2 Προσέγγιση ανθρώπινου περπατήματος από κυλιόμενο πολύγωνο (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

Εκτός από την απλούστερη υλοποίησή τους, οι τροχοί είναι επίσης σχεδιασμένοι να αποδίδουν σε ομαλό έδαφος, όπως είναι το δάπεδο εσωτερικών χώρων. Όπως διαπιστώνεται από την Εικόνα 2.3, η κίνηση με τροχούς σε επίπεδο έδαφος είναι αρκετά αποδοτικότερη από την κίνηση με πόδια. Ο σιδηρόδρομος, για παράδειγμα, είναι ειδικά σχεδιασμένος για κίνηση με τροχούς, καθώς σε αυτόν η αντίσταση κύλισης ελαχιστοποιείται λόγω της σκληρής και επίπεδης επιφάνειας. Σε πιο μαλακές επιφάνειες όμως, η κίνηση με τροχούς γίνεται πιο δύσκολη και απαιτεί μεγαλύτερα ποσά ενέργειας, όπως φαίνεται στο Σχήμα 2.3.



Εικόνα 2.3 Απαιτούμενη ισχύς συναρτήσει της ταχύτητας για διάφορους μηχανισμούς κίνησης (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

2.2 Αυτόνομα Κινητά Συστήματα

Εκτός από την ικανότητα της κίνησης, ένα άλλο σημαντικό στοιχείο ενός κινητού συστήματος είναι η δυνατότητα απομακρυσμένης λειτουργίας από τον ανθρώπινο χειριστή. Αυτό σημαίνει πως το σύστημα χρειάζεται είτε να έχει κάποιο βαθμό αυτονομίας, δηλαδή να μπορεί να κινηθεί στο χώρο χωρίς τη βοήθεια του χειριστή, είτε να μπορεί να λειτουργήσει ως τηλεχειριζόμενο σύστημα, δηλαδή να δέχεται απευθείας εντολές από τον άνθρωπο-χειριστή.

Για να θεωρηθεί ένα σύστημα αυτόνομο, πρέπει να είναι ικανό να κινείται αυτόνομα στο περιβάλλον του. Ακόμη θα πρέπει να είναι αυτόνομο ως προς τους εξής δύο παράγοντες:

- ενέργεια: το ρομπότ θα πρέπει να φέρει κάποια πηγή ενέργειας
- λήψη αποφάσεων: το ρομπότ θα πρέπει να μπορεί να λαμβάνει αποφάσεις και να ενεργεί κατάλληλα

Στην πραγματικότητα αυτό σημαίνει πως το κινητό σύστημα λαμβάνει εντολές από τον άνθρωπο-χειριστή βασιζόμενο στο επίπεδο αυτονομίας που του παρέχεται. Το σύστημα ανταποκρίνεται προσπαθώντας να φέρει εις πέρας την εργασία που του ανατέθηκε, και αν δεν υπάρχουν απρόβλεπτες συνθήκες η εργασία ολοκληρώνεται επιτυχώς σε συγκεκριμένο χρονικό διάστημα. Ανάλογα και με το επίπεδο αυτονομίας του ρομπότ, οι παρακάτω τυπικές εντολές μπορούν να ληφθούν από το χειριστή:

- Επιθυμητή ταχύτητα τροχών
- Επιθυμητή γραμμική ταχύτητα και ταχύτητα περιστροφής
- Λειτουργία μέσα σε γνωστό περιβάλλον
- Λειτουργία σε άγνωστο περιβάλλον
- Επιθυμητή εργασία προς εκτέλεση

Τα κύρια μηχανικά και ηλεκτρονικά μέρη ενός αυτόνομου κινητού ρομπότ είναι:

- **Μηχανικά μέρη** Σταθερά και κινούμενα μέρη (σώμα, τροχοί, πόδια, κλπ.)
- **Επενεργητές** Ηλεκτρικοί κινητήρες (DC, βηματικοί, σερβοκινητήρες, κλπ)
- **Αισθητήρες** Encoders, Αισθητήρες προσέγγισης και απόστασης, Αδρανειακή μονάδα μέτρησης, GPS, κλπ.
- **Υπολογιστές** Μικροελεγκτές, φορητοί υπολογιστές, ενσωματωμένα συστήματα, κλπ
- **Μονάδα Ισχύος** Μπαταρίες, Φωτοβολταϊκά, κλπ
- **Ηλεκτρονικά** Μέτρηση αισθητήρων, διανομή ισχύος, ηλεκτρονικές διατάξεις για επικοινωνία

2.3 Κινητά Ρομπότ με Τροχούς

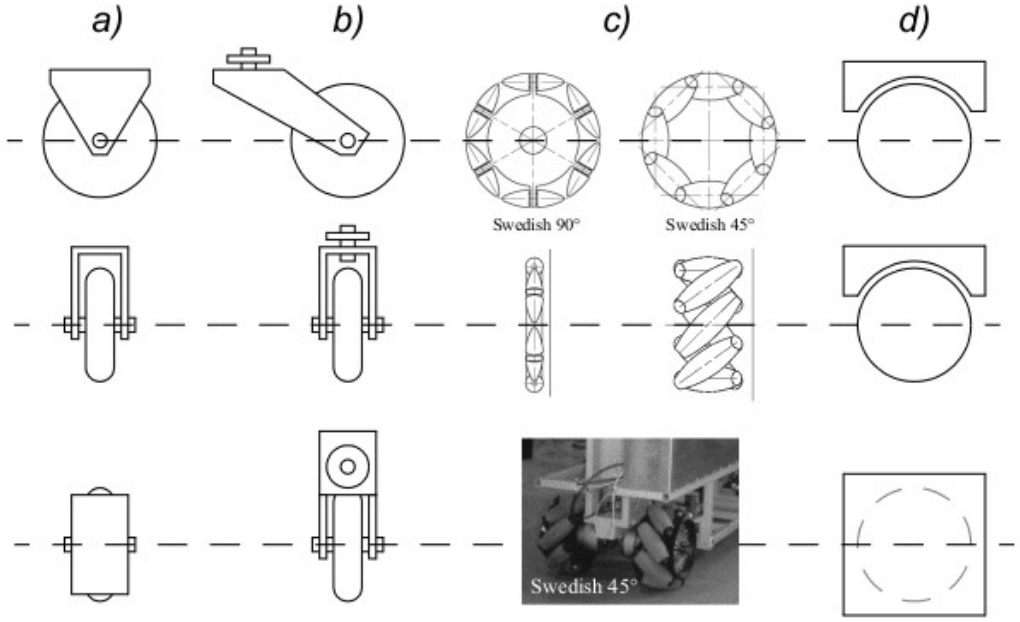
Εκτός από την αποδοτικότητα, ένας άλλος παράγοντας που συντέλεσε στο να γίνει ο τροχός ένας από τους πιο δημοφιλείς μηχανισμούς κίνησης είναι η ευστάθεια που εξασφαλίζει στο ρομπότ. Τα κύρια χαρακτηριστικά της ευστάθειας είναι ο αριθμός και η γεωμετρία των σημείων επαφής, το κέντρο βάρους του, η στατική ή δυναμική ευστάθεια και η κλίση του εδάφους.

Τα περισσότερα ρομπότ με τροχούς είναι σχεδιασμένα έτσι ώστε όλοι τους οι τροχοί να βρίσκονται συνεχώς σε επαφή με το έδαφος. Έτσι είναι δυνατό να εξασφαλίζεται η ισορροπία του συστήματος ακόμη και με τρεις τροχούς, αρκεί το κέντρο βάρους του ρομπότ να βρίσκεται πάνω από το νοητό τρίγωνο που σχηματίζεται με κορυφές τα σημεία επαφής των τροχών με το έδαφος. Σε συγκεκριμένες περιπτώσεις μπορεί ακόμη και με δύο τροχούς να επιτευχθεί ευστάθεια, ενώ όταν το ρομπότ διαθέτει από τέσσερις τροχούς και πάνω χρησιμοποιείται κατάλληλο σύστημα ανάρτησης για να εξασφαλίζεται η διατήρηση επαφής κάθε τροχού με το έδαφος ακόμη και όταν αυτό δεν είναι ομαλό.

2.3.1 Είδη Τροχών

Η μελέτη της κίνησης με τροχούς ξεκινάει από τον ίδιο τον τροχό. Οι κύριες κατηγορίες τροχών είναι τέσσερις, όπως φαίνεται στην Εικόνα 2.4

Στην Εικόνα 2.4.a) φαίνεται ο συμβατικός τροχός με δύο βαθμούς ελευθερίας, οι οποίοι είναι η περιστροφή γύρω από τον άξονα του τροχού και γύρω από την επαφή. Στην Εικόνα 2.4.b) φαίνεται ο τροχός τύπου castor, ο οποίος έχει επίσης δύο βαθμούς ελευθερίας, την περιστροφή γύρω από τον άξονα του τροχού και γύρω από τον μετατοπισμένο σύνδεσμο κατεύθυνσης. Στην Εικονα 2.4.c) δείχνεται ο Σουηδικός τροχός 45º και 90º, με τρεις βαθμούς ελευθερίας: περιστροφή γύρω από το σημείο επαφής, γύρω από τον άξονα του τροχού και γύρω από τα καρούλια. Στην Εικόνα 2.4.d) φαίνεται ο σφαιρικός τροχός, που είναι παγκατευθυντικός αλλά δύσκολος στην υλοποίησή του.

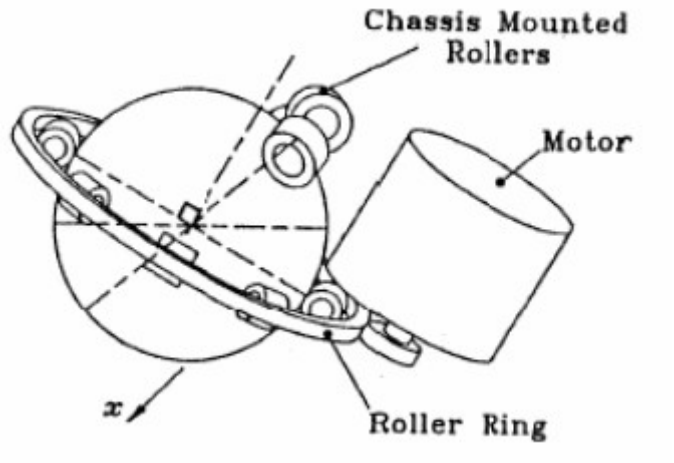


Εικόνα 2.4 Οι 4 κύριες κατηγορίες τροχών (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

Τα κύρια πλεονεκτήματα του συμβατικού και του τροχού castor είναι η εύκολη υλοποίηση, η ικανότητα να φέρουν μεγάλο φορτίο και η ανοχή σε ανώμαλο έδαφος. Όμως οι τροχοί αυτοί δεν είναι από μόνοι τους παγκατευθυντικοί, και για να μπορέσουμε να κατευθύνουμε ένα όχημα που τους χρησιμοποιεί, οι κατευθυνόμενοι τροχοί (εξαρτάται από το πως είναι σχεδιασμένο το όχημα) πρέπει πρώτα να καθοδηγούνται γύρω από έναν κατακόρυφο άξονα και μετά να περιστρέφονται γύρω από τον οριζόντιό τους άξονα. Συνεπώς, ειδικά σε βαριά οχήματα και όταν δεν κινούνται κατά την περιστροφή του τροχού γύρω από τον κατακόρυφο άξονα, αυτή η μέθοδος δημιουργεί μεγάλη τριβή που αυξάνει την κατανάλωση ενέργειας και μειώνει την ακρίβεια με την οποία μπορούμε να κατευθύνουμε το όχημα.

Ο Σουηδικός τροχός λειτουργεί όπως ένας κανονικός τροχός, με τη διαφορά ότι διαθέτει μικρούς παθητικούς κυλίνδρους (καρούλια) στην περιφέρειά του. Αυτοί οι κύλινδροι δημιουργούν μικρή αντίσταση προς τις άλλες κατευθύνσεις, αναλόγως τη γωνία την οποία σχηματίζουν, και με αυτόν τον τρόπο ο τροχός μπορεί να κυλήσει προς οποιαδήποτε κατεύθυνση. Ο άξονας του κυρίως τροχού είναι ο μόνος στον οποίο παρέχεται ισχύς, αλλά είναι δυνατό να σχεδιαστούν με χρήση αυτού του τροχού ολονομικά παγκατευθυντικά ρομπότ.

Ο σφαιρικός τροχός είναι παγκατευθυντικός. Υπάρχουν αρκετές διαφορετικές υλοποιήσεις του, μία εκ των οποίων αναπτύχθηκε από τους West και Asada το 1997 και φαίνεται στην Εικόνα 2.5



Εικόνα 2.5 Μηχανισμός σφαιρικού τροχού (Πηγή: H. Yu, S. Dubowsky, A. Skwersky, Omni-Directional mobility using active split offset castors, Journal of mechanical design, vol. 126)

Σύμφωνα με το σχεδιασμό του σφαιρικού τροχού, η ισχύς μεταδίδεται μέσω γραναζιών από έναν κινητήρα σε έναν δακτύλιο κυλίνδρων και στη συνέχεια από τους κυλίνδρους στη σφαίρα μέσω της μεταξύ τους τριβής. Χάρη στους κυλίνδρους του δακτυλίου και σε αυτούς που είναι ενσωματωμένοι στο σασί, η σφαίρα μπορεί να κυλήσει προς οποιαδήποτε κατεύθυνση. Ένα ρομπότ χρειάζεται τουλάχιστον τρεις σφαιρικούς τροχούς για να κινηθεί.

2.3.2 Κύρια σημεία της κίνησης με τροχούς

Κατά τη σχεδίαση ενός τροχοφόρου ρομπότ, ο σχεδιαστής έχει να επιλέξει ανάμεσα σε πολλές διαφορετικές διατάξεις και είδη τροχών. Ο συνδυασμός του είδους και της διάταξης των τροχών είναι στενά συνδεδεμένος με την ευστάθεια, την ευελιξία και την ελεγξιμότητα του ρομπότ. Ένα τέτοιο παράδειγμα συνδυασμού είναι η διάταξη Ackermann στα αυτοκίνητα, με δύο κατευθυνόμενους τροχούς μπροστά και δύο μη κατευθυνόμενους πίσω, ενώ τουλάχιστον δύο από αυτούς, συνδεδεμένοι μεταξύ τους με έναν άξονα, συνδέονται στον κινητήρα. Σχεδόν σε όλα τα αυτοκίνητα χρησιμοποιείται η παραπάνω διάταξη, καθώς μεγιστοποιεί την ελεγξιμότητα, την ευστάθεια και την ευελιξία στο περιβάλλον του οδικού δικτύου.

Στην περίπτωση των ρομπότ, δεν υπάρχει μόνο ένα περιβάλλον για το οποίο είναι σχεδιασμένα όλα τα ρομπότ, καθώς διαφορετικά ρομπότ είναι σχεδιασμένα για εφαρμογές σε πολύ διαφορετικές καταστάσεις. Και καθώς δεν υπάρχει μια συγκεκριμένη διάταξη τροχών που να εξασφαλίζει μέγιστη ευστάθεια, ελεγξιμότητα και ευελιξία σε όλα τα περιβάλλοντα, πρέπει ο εκάστοτε σχεδιαστής να επιλέγει την καταλληλότερη ανάλογα με την εφαρμογή που θέλει να έχει το ρομπότ, και το πόσο η κάθε διάταξη επηρεάζει τους παραπάνω παράγοντες.

Ευστάθεια

Όπως έχει αναφερθεί προηγουμένως, ο ελάχιστος αριθμός τροχών που απαιτείται για να επιτευχθεί στατική ευστάθεια είναι δύο. Ένα δίτροχο ρομπότ διαφορικής κίνησης, μπορεί να θεωρηθεί ευσταθές, αν το κέντρο μάζας βρίσκεται κάτω από τον άξονα των τροχών ή αν υπάρχει ένα τρίτο σημείο επαφής με το έδαφος. Οι παραπάνω όμως είναι κάποιες πολύ ειδικές περιπτώσεις. Υπό κανονικές συνθήκες ένα ρομπότ χρειάζεται τουλάχιστον τρεις τροχούς σε επαφή με το έδαφος για να επιτευχθεί στατική ευστάθεια. Ακόμα αναγκαία συνθήκη είναι το κέντρο βάρους να βρίσκεται εντός του πολυγώνου που σχηματίζουν τα σημεία επαφής των τροχών με το έδαφος.

Ευελιξία

Η ευελιξία παίζει ιδιαίτερα σημαντικό ρόλο στο να είναι αποτελεσματικό ένα ρομπότ με τροχούς. Ένα ρομπότ που έχει την ικανότητα να κινηθεί προς οποιαδήποτε κατεύθυνση ενός επιπέδου x-y ονομάζεται παγκατευθυντικό. Για να επιτευχθεί αυτό απαιτούνται συνήθως τροχοί, οι οποίοι μπορούν να κινηθούν προς περισσότερες από μία κατευθύνσεις όπως οι σουηδικοί ή οι σφαιρικοί.

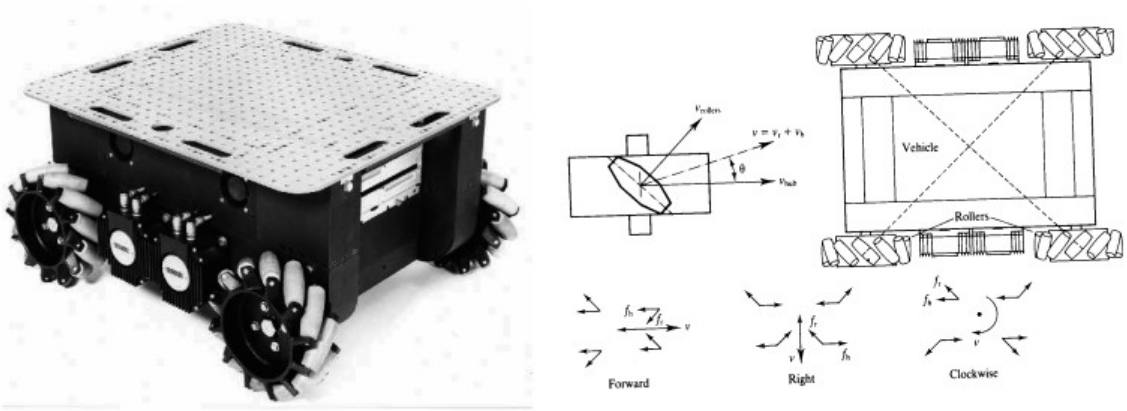
Η γεωμετρία Ackermann που χρησιμοποιείται στα αυτοκίνητα, δεν έχει εφαρμογή σε παγκατευθυντικά ρομπότ. Οχήματα που η κίνησή τους περιγράφεται από τη γεωμετρία Ackermann έχουν συνήθως ακτίνα στροφής μεγαλύτερη από τα ίδια τα οχήματα. Επίσης δεν είναι ικανά να κάνουν κινήσεις παράλληλες στον άξονα των τροχών τους. Έτσι για την επίτευξη ενός παρκαρίσματος απαιτούνται παραπάνω από μία πρόσθιες και οπίσθιες κινήσεις παράλληλα με αλλαγές κατεύθυνσης.

Ελεγξιμότητα

Το βασικό πλεονέκτημα ενός παγκατευθυντικού ρομπότ είναι η ευελιξία του. Παράλληλα όμως αυτή η ευελιξία καθιστά δύσκολο τον έλεγχο του ρομπότ. Για παράδειγμα για να κινηθεί σε ευθεία γραμμή ένα ρομπότ που χρησιμοποιεί τέσσερεις σουηδικούς τροχούς όπως το Carnige Mellon Uranus Robot της Εικόνας 2.6, όλοι οι τροχοί πρέπει να περιστρέφονται με την ίδια ακριβώς ταχύτητα. Ακόμη και μια πολύ μικρή παρέκκλισή στην ταχύτητα ενός τροχού θα έχει ως αποτέλεσμα την αλλαγή της κατεύθυνσής του.

Με το παραπάνω παράδειγμα γίνεται εύκολα αντιληπτό το πλεονέκτημα της γεωμετρίας Ackermann, αφού σε αυτή την περίπτωση, για να κινηθεί ένα όχημα σε ευθεία γραμμή, αρκεί όλοι οι τροχοί να τεθούν στην ίδια κατεύθυνση. Καθώς αυτοί ενώνονται από άξονες, περιστρέφονται με την ίδια ταχύτητα με τη χρήση ενός μόνο κινητήρα.

Τελικά μπορεί κανείς να υποστηρίξει πως η ευελιξία και η ελεγξιμότητα είναι αντιστρόφως ανάλογες έννοιες σε ότι αφορά την κίνηση ενός ρομπότ. Όσο πιο ευέλικτο είναι ένα ρομπότ, τόσο πιο δύσκολο είναι να ελεγχθεί η κίνησή του και το αντίθετο.



Εικόνα 2.6 Το ρομπότ Carnegie Mellon Uranus, ένα παγκατευθυντικό ρομπότ με 4 κινητήριους σουηδικούς τροχούς 45º. (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

Ολονομία

Η ολονομία ενός συστήματος αναφέρεται στη σχέση μεταξύ του ελέγξιμου και του συνολικού αριθμού των βαθμών ελευθερίας του ρομπότ. Στην περίπτωση που οι ελέγξιμοι βαθμοί ελευθερίας είναι ίσοι σε αριθμό με τους συνολικούς βαθμούς ελευθερίας του συστήματος, τότε το σύστημα είναι ολόνομο. Στην αντίθετη περίπτωση, όπου οι ελέγξιμοι βαθμοί ελευθερίας του ρομπότ είναι λιγότεροι σε αριθμό από τους συνολικούς βαθμούς ελευθερίας, το σύστημα χαρακτηρίζεται μη ολόνομο.

Το ρομπότ που χρησιμοποιήθηκε στην παρούσα διπλωματική εργασία εντάσσεται στην κατηγορία του μη ολόνομου συστήματος καθώς έχει τρεις βαθμούς ελευθερίας, τους δυο άξονες στο επίπεδο της κίνησης (x,y) και τον προσανατολισμό του ρομπότ (θ), ενώ οι καταστάσεις που είναι ελέγξιμες είναι η μεταφορική (εφαπτομενική) ταχύτητα του ρομπότ (u) και η περιστροφική (ω).

3

Μοντελοποίηση Κίνησης Ρομποτικού Οχήματος

3.1 Εισαγωγή

Στη ρομποτική χρειάζεται να κατανοήσουμε τη μηχανική συμπεριφορά του ρομπότ ώστε να μπορέσουμε να το σχεδιάσουμε κατάλληλα αλλά και ώστε να αναπτύξουμε κατάλληλο λογισμικό ελέγχου του υλικού του.

Η κινηματική είναι η βασικότερη μελέτη της μηχανικής συμπεριφοράς ενός συστήματος. Περιγράφει μαθηματικά την κίνηση του ρομπότ χωρίς να λαμβάνει υπόψη τις αιτίες της, όπως δυνάμεις ή ροπές. Περιλαμβάνει τη μελέτη του περιβάλλοντος εργασίας και μέσω

αυτής καθορίζεται το σύνολο των πιθανών θέσεων του ρομπότ μέσα σε αυτό, η ελεγξιμότητά του καθώς και οι πιθανές τροχιές που δύναται να ακολουθήσει.

Επιπλέον περιορισμοί εισέρχονται και με τη δυναμική μελέτη του συστήματος, που καθορίζει τη συμπεριφορά του όταν λαμβάνουμε υπόψη τις δυνάμεις και ροπές που ασκούνται σε αυτό.

Η μελέτη της κίνησης ενός ρομπότ ξεκινάει από τη μελέτη του πώς συμβάλλει ο κάθε τροχός ξεχωριστά στη συνολική του κίνηση. Κάθε τροχός παίζει ένα ρόλο στην ενεργοποίηση της κίνησης του οχήματος, ενώ επίσης εισάγει σε αυτή και κάποιους περιορισμούς, όπως για παράδειγμα εμποδίζει την πλευρική του ολίσθηση.

Η μεγάλη πρόκληση όσον αφορά την κινηματική είναι η δυνατότητα εκτίμησης της θέσης (pose estimation) του οχήματος ανά πάσα στιγμή. Τα κινούμενα ρομπότ κινούνται αυτόνομα μέσα στο περιβάλλον τους και για να υπολογιστεί η θέση τους μέσα στο χώρο, θα πρέπει να ολοκληρώνεται η κίνησή τους με το χρόνο. Συνυπολογίζοντας και τυχόν ανακρίβειες που προκύπτουν από ολίσθηση των τροχών, είναι εμφανές ότι ο ακριβής υπολογισμός της θέσης του κινούμενου ρομπότ μέσα στο χώρο είναι απαιτητική διαδικασία.

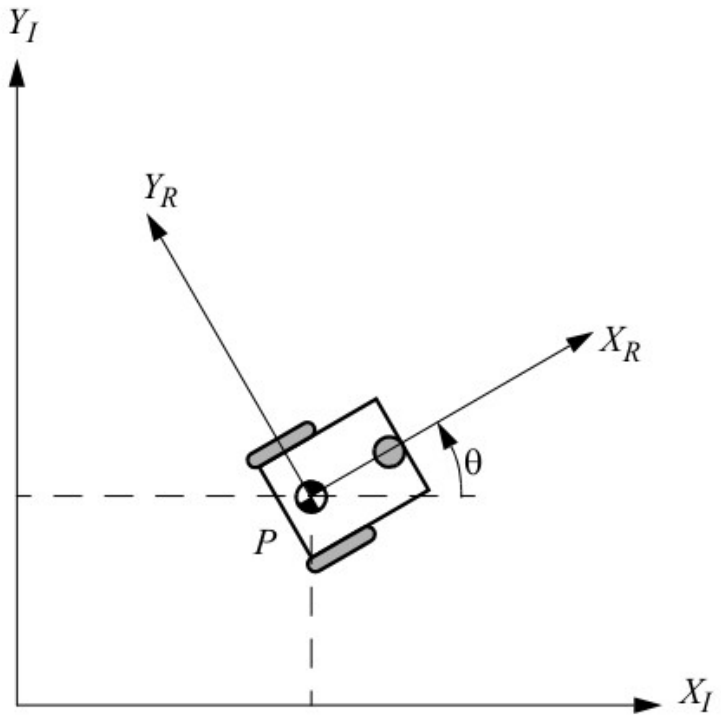
Στο κεφάλαιο αυτό θα ασχοληθούμε με το πώς εκφράζουμε την κίνηση του ρομπότ ως προς ένα σύστημα συντεταγμένων με εξωτερικό σημείο αναφοράς αλλά και ως προς το τοπικό σύστημα συντεταγμένων του ρομπότ.

3.2 Κινηματικά Μοντέλα και Περιορισμοί

Η εξαγωγή ενός κινηματικού μοντέλου για τη συνολική κίνηση του ρομπότ είναι μια διαδικασία που ξεκινά από κάτω προς τα πάνω. Κάθε τροχός συμβάλλει στην κίνηση του ρομπότ και ταυτόχρονα της επιβάλλει κάποιους περιορισμούς. Οι τροχοί είναι ενσωματωμένοι πάνω στο σασί ανάλογα με τη διάταξή τους, και συνεπώς οι περιορισμοί τους συνδυάζονται και σχηματίζουν περιορισμούς για όλο το σασί του ρομπότ. Όμως οι δυνάμεις και οι περιορισμοί των τροχών πρέπει να εκφράζονται ως προς ένα σαφές και σταθερό σύστημα αναφοράς. Αυτό έχει μεγάλη σημασία για τη μελέτη των κινουμένων ρομπότ εξαιτίας της αυτόνομης φύσης τους που καθιστά αναγκαίο να υπάρχει ξεκάθαρη αντιστοιχία μεταξύ τοπικών και αδρανειακών συντεταγμένων.

3.2.1 Αναπαράσταση της θέσης του ρομπότ

Σε όλη την ανάλυση που ακολυθεί θεωρούμε το ρομπότ ως στερεό σώμα πάνω σε τροχούς, που λειτουργεί στο οριζόντιο επίπεδο. Οι συνολικές διαστάσεις που χρησιμοποιούμε για το ρομπότ στο επίπεδο είναι τρεις: δύο για τη θέση του στο επίπεδο και μία για τον προσανατολισμό του ως προς τον κατακόρυφο άξονα, ο οποίος είναι κάθετος προς το επίπεδο. Φυσικά υπάρχουν και άλλοι βαθμοί ελευθερίας εξαιτίας των αξόνων των τροχών και των αρθρώσεων κατεύθυνσης τροχών. Όμως με τον όρο σασί αναφερόμαστε μόνο στο στερεό σώμα του ρομπότ αγνοώντας τις αρθρώσεις και τους εσωτερικούς βαθμούς ελευθερίας του ρομπότ και των τροχών του.



Εικόνα 3.1 Το αδρανειακό και το τοπικό σύστημα συντεταγμένων (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

Για να ορίσουμε τη θέση του ρομπότ στο επίπεδο, καθορίζουμε μια σχέση μεταξύ του αντικειμενικού και του τοπικού συστήματος αναφοράς, όπως φαίνεται στην Εικόνα 3.1. Οι άξονες X_I και Y_I ορίζουν ένα αυθαίρετο αδρανειακό σύστημα πάνω στο επίπεδο, το οποίο θα χρησιμοποιήσουμε σαν αντικειμενικό σύστημα αναφοράς, με σημείο αναφοράς O: {X_I , Y_I}. Για τον καθορισμό της θέσης του ρομπότ, επιλέγουμε ένα σημείο P πάνω στο σασί του. Η βάση {X_R , Y_R} ορίζει δύο άξονες σχετικούς ως προς το P και συνεπώς το τοπικό σύστημα αναφοράς. Η θέση του P στο αντικειμενικό σύστημα αναφοράς καθορίζεται από τις συντεταγμένες x και y, και η γωνιακή διαφορά μεταξύ αντικειμενικού και τοπικού συστήματος αναφοράς δίνεται από τη γωνία θ. Μπορούμε να περιγράψουμε τη θέση του ρομπότ σαν ένα διάνυσμα με τα τρία αυτά στοιχεία:

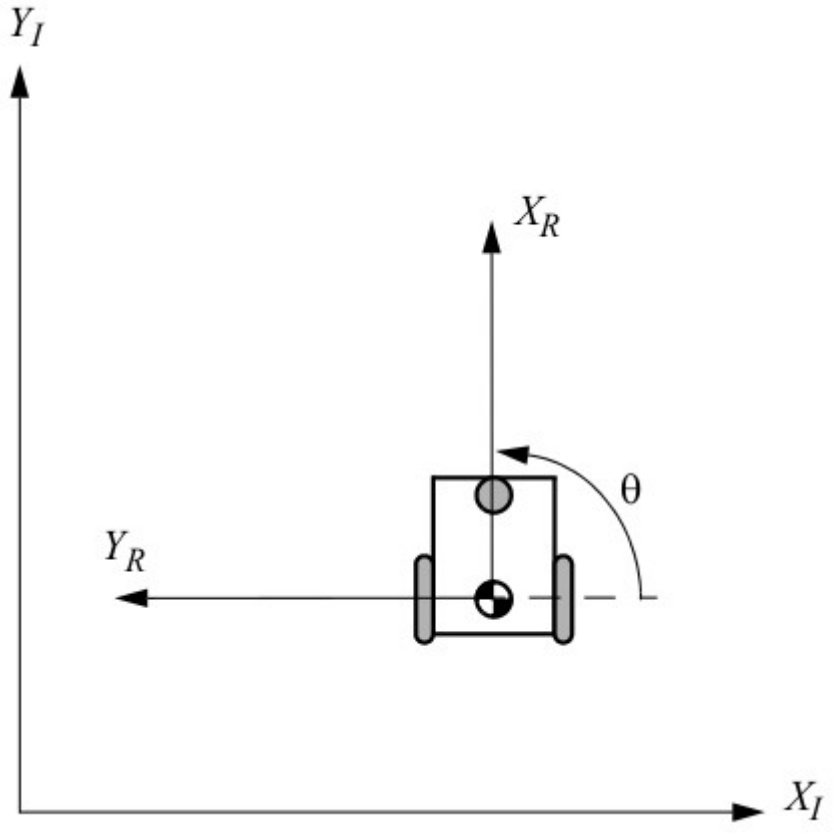
$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{3.1}$$

Για να περιγράψουμε την κίνηση του ρομπότ ως αποτέλεσμα σύνθετων κινήσεων, θα πρέπει να υπάρχει αντιστοιχία της κίνησης ως προς τους άξονες του αντικειμενικού συστήματος αναφοράς με την κίνηση ως προς τους άξονες του τοπικού συστήματος αναφοράς. Φυσικά, η αντιστοιχία είναι συνάρτηση της τρέχουσας θέσης του ρομπότ και επιτυγχάνεται μέσω της ορθογώνιας μήτρας περιστροφής:

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

Ο παραπάνω πίνακας μπορεί να χρησιμοποιηθεί για να αντιστοιχίσουμε την κίνηση στο αντικειμενικό σύστημα συντεταγμένων $\{X_I, Y_I\}$ σε κίνηση στο τοπικό σύστημα συντεταγμένων $\{X_R, Y_R\}$. Αυτή η διαδικασία γράφεται ως $R(\theta)\dot{\xi}_I$ καθώς ο υπολογισμός της εξαρτάται από την τιμή της γωνίας θ:

$$\dot{\xi}_R = R(\theta)\dot{\xi}_I \quad (3.3)$$



Εικόνα 3.2 Το ρομπότ ευθυγραμμισμένο με άξονα του αντικειμενικού συστήματος αναφοράς (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

Για παράδειγμα, αν θεωρήσουμε το ρομπότ της Εικόνας 3.2, ο στιγμιαίος πίνακας περιστροφής R είναι:

$$R(\frac{\pi}{2}) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Συνεπώς, δοσμένης της ταχύτητας $(\dot{x}, \dot{y}, \dot{\theta})$ στο αδρανειακό σύστημα συντεταγμένων, μπορούμε να υπολογίσουμε τις επιμέρους κινήσεις κατά μήκος των αξόνων του τοπικού συστήματος αναφοράς X_R και Y_R :

$$\dot{\xi}_R = R(\frac{\pi}{2})\dot{\xi}_I = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{bmatrix} \tag{3.5}$$

3.2.2 Ορθή Κινηματική

Στις πιο απλές περιπτώσεις, η αντιστοίχιση που περιγράφεται από την εξίσωση (3.3) είναι αρκετή για να δημιουργήσει έναν τύπο που αποτυπώνει την ορθή κινηματική του κινουμένου οχήματος: το πώς κινείται το ρομπότ δεδομένης της γεωμετρίας του και των ταχυτήτων των τροχών του.

Θεωρούμε για παράδειγμα το ρομπότ της Εικόνας 3.3. Πρόκειται για ένα ρομπότ διαφορικής κίνησης με δύο τροχούς, ο καθένας εκ των οποίων έχει ακτίνα r. Δεδομένου ενός σημείου P που βρίσκεται στο μέσο του κοινού άξονα των τροχών, κάθε τροχός απέχει απόσταση l από το P. Δεδομένων των r,l,θ και των ταχυτήτων περιστροφής των τροχών $\dot{\varphi}_1, \dot{\varphi}_2$, ένα μοντέλο ορθής κινηματικής θα προέβλεπε τη συνολική ταχύτητα του ρομπότ στο αντικειμενικό σύστημα αναφοράς:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \dot{\varphi}_1, \dot{\varphi}_2) \qquad (3.6)$$

Από την εξίσωση (3.3) γνωρίζουμε πως μπορούμε να υπολογίσουμε την κίνηση του ρομπότ στο αντικειμενικό σύστημα από την ταχύτητα στο τοπικό σύστημα αναφοράς:

$$\dot{\xi}_I = R(\theta)^{-1} \dot{\xi}_R \qquad (3.7)$$

Συνεπώς, πρώτα θα υπολογίσουμε τη συνεισφορά του κάθε τροχού στη συνολική κίνηση στο τοπικό σύστημα συντεταγμένων.

Θεωρούμε πως οι τοπικοί άξονες είναι τοποθετημένοι έτσι ώστε όταν το ρομπότ κινείται μπροστά να κινείτα προς τη θετική κατεύθυνση του άξονα $+X_R$ όπως στην Εικόνα 3.1. Πρώτα θα υπολογίσουμε τη συνεισφορά κάθε τροχού στη μεταφορική ταχύτητα του P κατά τον άξονα $+X_R$. Αν ο ένας τροχός περιστρέφεται εω ο άλλος μένει ακίνητος και δε συνεισφέρει τίποτα, και αφού το P βρίσκεται ακριβώς στο μέσο του άξονα, θα κινηθεί στιγμιαία με ταχύτητα: $\dot{x_{r1}} = \frac{1}{2} r \dot{\varphi}_1$ ή $\dot{x_{r2}} = \frac{1}{2} r \dot{\varphi}_2$. Σε ένα ρομπότ διαφορικής κίνησης αυτές οι δύο συνιστώσες μπορούν απλά να προστεθούν για να υπολογισθεί η ταχύτητα $\dot{x}_R$. Αν για παράδειγμα οι τροχοί κινηθούν με ίδια ταχύτητα προς αντίθετες κατευθύνσεις, το ρομπότ θα περιστρέφεται στην ίδια θέση. Η ταχύτητα $\dot{y}_R$ υπολογίζεται ακόμη πιο εύκολα. Κανένας από τους τροχούς δεν προκαλεί κίνηση προς τα πλάγια, οπότε το $\dot{y}_R$ ισούται πάντα με 0. Άρα μένει ο υπολογισμός της περιστροφικής συνιστώσας $\dot{\theta}_R$. Πάλι θα υπολογίσουμε ξεχωριστά την κίνηση που προκαλεί ο κάθε τροχός και θα τις προσθέσουμε. Θεωρούμε τον δεξί τροχό ως τον τροχό 1. Αν κινηθεί μπροστά προκαλεί στο P περιστροφή αντίθετα προς τη φορα του ρολογιού. Αν ο τροχός 2 (αριστερός) παραμένει ακίνητος, το

ρομπότ περιστρέφεται γύρω του. Ο τροχός 1 εκτελεί κυκλική κίνηση με ακτίνα 2l και σταθερή γωνιακή ταχύτητα ω_1 :

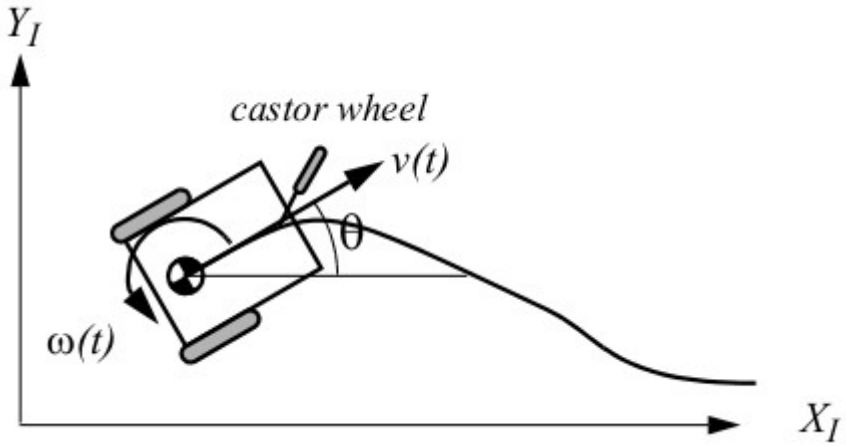
$$\omega_1 = \frac{r\dot{\varphi}_1}{2l} \quad . \tag{3.8}$$

Οι ίδιοι υπολογισμοί εκτελούνται για τον αριστερό τροχό, με τη διαφορά ότι προκαλεί περιστροφή ίδια με τη φορά του ρολογιού:

$$\omega_2 = \frac{-r\dot{\varphi}_2}{2l} \quad . \tag{3.9}$$

Συνδυάζοντας τους παραπάνω τύπους εξάγουμε το κινηματικό μοντέλο του δίτροχου ρομπότ διαφορικής κίνησης:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} + \frac{-r\dot{\varphi}_2}{2l} \end{bmatrix} = \begin{bmatrix} cos\theta \frac{r}{2}(\dot{\varphi}_1 + \dot{\varphi}_2) \\ sin\theta \frac{r}{2}(\dot{\varphi}_1 + \dot{\varphi}_2) \\ \frac{r}{2l}(\dot{\varphi}_1 - \dot{\varphi}_2) \end{bmatrix} \quad . \tag{3.10}$$



Εικόνα 3.3 Ρομπότ διαφορικής κίνησης (Πηγή: Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, Introduction to Autonomous Mobile Robots)

Αν ορίσουμε ως u τη στιγμιαία ταχύτητα του ρομπότ κατά τον άξονα $+X_R$, και ω τη στιγμιαία γωνιακή ταχύτητα του ρομπότ, τότε ισχύει:

$$u=\frac{r}{2}(\dot{\varphi}_1+\dot{\varphi}_2) \quad (3.11)$$

$$\omega=\dot{\theta} \quad (3.12)$$

Έτσι από τη σχέση (3.10) προκύπτει ότι:

$$\dot{x}=u\cdot cos\theta \quad (3.13)$$

$$\dot{y}=u\cdot sin\theta \quad (3.14)$$

Λύνοντας το σύστημα των δοαφορκιών εξισώσεων (3.13) και (3.14), για δεδομένα u(t) και ω(t) και για γνωστές αρχικές συνθήκες (x_0,y_0,θ_0), υπολογίζουμε τα x(t), y(t), θ(t) για κάθε στιγμή.

3.2.3 Αντίστροφη Κινηματική

Το πρόβλημα της αντίστροφης κινηματικής ρομποτικού οχήματος έγκειται στον προσδιορισμό κατάλληλων εισόδων $u_r(t)$ και $\omega_r(t)$ για να επιτύχουμε μια επιθυμητή τροχιά αναφοράς ($x_r(t)$, $y_r(t)$, $\theta_r(t)$).

Επιλύοντας τις (3.13) και (3.14) ως προς $u_r(t)$ προκύπτει:

$$\tan(\theta_r(t)) = \frac{\dot{y}_r(t)}{\dot{x}_r(t)} \tag{3.15}$$

κια λύνοντας ως προς $\theta_r(t)$:

$$\theta_r(t) = \arctan\left(\frac{\dot{y}_r(t)}{\dot{x}_r(t)}\right) \tag{3.16}$$

Από τις (3.12) και (3.16) έχουμε:

$$\omega_r(t) = \frac{d}{dt}\left(\arctan\frac{\dot{y}_r(t)}{\dot{x}_r(t)}\right) = \frac{\ddot{y}_r(t)\,\dot{x}_r(t) - \dot{y}_r(t)\,\ddot{x}_r(t)}{\dot{x}_r(t)^2 + \dot{y}_r(t)^2} \tag{3.17}$$

Από πρόσθεση κατά μέλη των (3.13) και (3.14) προκύπτει:

$$u_r(t) = \pm\sqrt{\dot{x}_r(t)^2 + \dot{y}_r(t)^2} \tag{3.18}$$

όπου το + υποδηλώνει κίνηση προς τα εμπρός ενώ το – κίνηση προς τα πίσω.

Οι εξισώσεις (3.16),(3.17),(3.18) προσδιορίζουν τις κατάλληλες εισόδους του συστήματος για διάφορες τροχιές αναφοράς. Ωστόσο, το μοντέλο δεν είναι ιδανικό (θόρυβος, ολίσθηση τροχών, ευαισθησία στις αρχικές συνθήκες), με αποτέλεσμα ο feedforward έλεγχος να μην επαρκεί για σωστή παρακολούθηση της τροχιάς αναφοράς.

4

Αισθητήρες

4.1 Αισθητήρες Προσέγγισης

Ένας αισθητήρας προσέγγισης μετράει την απόσταση μεταξύ του ίδιου και ενός υλικού αντικειμένου που συνήθως ονομάζεται στόχος. Η έξοδος του αισθητήρα μπορεί να είναι αναλογική (τάση), σε μορφή σειριακών δεδομένων ή σήμα με διαμόρφωση εύρους παλμού και μπορεί να μεταδίδεται με χρήση προτοκόλλων σειριακής επικοινωνίας όπως SPI, TTL I2C

Εφαρμογές

Στη ρομποτική, ένας αισθητήρας προσέγγισης μπορεί να αποτρέψει τη σύγκρουση μεταξύ του ρομπότ και ενός κοντινού του εμποδίου. Κάποιοι πιο προηγμένοι αισθητήρες

προσέγγισης μπορούν να παρέχουν επαρκή δεδομένα στο κατάλληλο λογισμικό ώστε ναχαρτογραφήσει το περιβάλλον στο οποίο βρίσκεται το ρομπότ.

Μπορούν ακόμη να χρησιμοποιηθούν σε συστήματα συναγερμού, ή για εκτίμηση της στάθμης υγρών σε αποθηκευτικές δεξαμενές, ή σε αυτοκίνητα για να προειδοποιούν τον οδηγό όταν πλησιάζουν σε εμπόδιο (τελευταία χρησιμοποιούνται σε συνδυασμό με κάμερες οπισθοπορείας).

Σε φορητές συσκευές, αισθητήρες προσέγγισης χρησιμοποιούνται για να ανιχνεύσουν την παρουσία του χεριού ή του προσώπου του χρήστη, όπως για παράδειγμα όταν απενεργοποιείται η οθόνη του κινητού όταν ο χρήστης το φέρνει στο αυτί του.

Κατηγορίες

Οι κυριότερες κατηγορίες αισθητήρων προσέγγισης που χρησιμοποιούνται στη ρομποτική είναι:

Αισθητήρες Υπερήχων

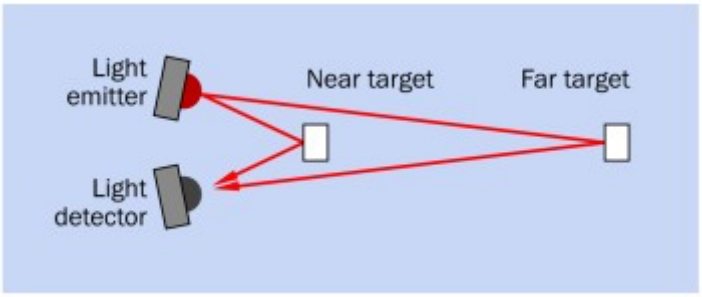
Ένας αισθητήρας προσέγγισης υπερήχων λειτουργεί εκπέμποντας μια μικρή ριπή ήχου και ακούγοντας την ηχώ που δημιουργείται από ανακλάσεις του ηχητικού κύματος σε εμπόδια του περιβάλλοντος.

Ο ήχος παράγεται από έναν πιεζοηλεκτρικό μορφοτροπέα σε μια συχνότητα μεταξύ των 30Hz και 50Hz, πολύ υψηλότερη από αυτές που μπορεί να ανιχεύσει το ανθρώπινο αυτί. Ο μορφοτροπέας μπορεί να εκτελεί διπλή λειτουργία, μία ως πομπός του ήχου και μία ως δέκτης του, ή μπορεί να χρησιμοποιούνται 2 διαφορετικοί μορφοτροπείς ενσωματωμένοι σε πολύ κοντινή απόσταση ο ένας απ τον άλλο.

Συνήθως η πλακέτα στην οποία ενσωματώνεται ο αισθητήρας περιλαμβάνει και έναν μικροελεγκτή που μετρά την καθυστέρηση μεταξύ της μετάδοσης ενός παλμού και της λήψης της ηχούς του. Η απόσταση από το αντικείμενο στο οποίο γίνεται η ανάκλαση υπολογίζεται σύμφωνα με την ταχύτητα του ήχου στον αέρα στο επίπεδο της θάλασσας, που είναι περίπου 340 m/s.

Αισθητήρες Υπερύθρων

Ένας αισθητήρας προσέγγισης υπερύθρων χρειάζεται για τη λειτουργία του μια ακτίνα υπέρυθρου φωτός από ένα LED που μπορεί να είναι ενσωματωμένο στην αισθητήρια μονάδα ή ξεχωριστά. Το φως ανακλάται στο στόχο και ανιχνεύεται από ένα φωτοτρανζίστορ ή μια φωτοδίοδο. Από τη γωνία του ανακλώμενης ακτίνας, μπορεί να υπολογισθεί η απόσταση από το στόχο, μέσω μιας διαδικασίας που ονομάζεται τριγωνισμός (Εικόνα 4.1).



Εικόνα 4.1 Ο αισθητήρας προσέγγισης υπερύθρων προσδιορίζει την απόσταση ενός αντικειμένου εκτιμώντας τη γωνία ανάκλασης (Πηγή: Charles Platt and Fredrik Jansson, Encyclopedia of Electronic Components Volume 3)

Για τη μείωση του ρίσκου ψευδοθετικών αποτελεσμάτων, το φως που εκπέμπεται από το LED περιέχει πολύ μικρό εύρος υπέρυθρων μηκών κύματος. Επίσης, διαμορφώνεται σε κατάλληλη συχνότητα που αναγνωρίζεται από τα κυκλώματα του αισθητήρα.

Χωρητικοί Αισθητήρες

Οι χωρητικοί αισθητήρες μετρούν την απόσταση ενός ηλεκτρικά αγώγιμου αντικειμένου. Σε αντίθεση με τους αισθητήρες των δύο προηγούμενων κατηγοριών, δε χρειάζεται επιπλέον πηγή φωτός, ήχου ή άλλης ακτινοβολίας. Η μέγιστη τυπική απόσταση εντοπισμού ενός τέτοιου αισθητήρα είναι περίπου 10mm, γι' αυτό και χρησιμοποιείται κυρίως στην παραγωγή μικρών συσκευών, οπότε δε θα ασχοληθούμε με την κατηγορία αυτή περαιτέρω.

Σχετικά Πλεονεκτήματα

Αισθητήρες υπερήχων

- Καταλληλότεροι για τον εντοπισμό αντικειμένων σε απόσταση μεγαλύτερη του ενός μέτρου
- Ανεπηρέαστοι από το ηλιακό φως, τους λαμπτήρες φθορισμού, και άλλες πηγές φωτός που προκαλούν παρεμβολές στους αισθητήρες υπερύθρων
- Μεγαλύτερη ακρίβεια, της τάξης των 5mm
- Ικανοί να μετρούν την απόσταση από υγρά και διάφανα αντικείμενα, που μπορεί να μην εντοπίζονται από αισθητήρες υπερύθρων

Αισθητήρες υπερύθρων

- Μικρότεροι σε μέγεθος
- Ικανοί να μετρούν αποστάσεις από μαλακά αντικείμενα, που ίσως δεν εντοπίζονται από αισθητήρες υπερήχων
- Καταλληλότεροι για εντοπισμό πολύ κοντινών αντικειμένων
- Πιο οικονομικοί

Εικόνα 4.2 Ο ασισθητήρας υπερήχω MaxSonar MB1003 εντοπίζει αντικείμενα σε απόσταση ως και 5 μέτρα (Πηγή: Charles Platt and Fredrik Jansson, Encyclopedia of Electronic Components Volume 3)

5

Ταυτόχρονος Εντοπισμός Θέσης και Χαρτογράφηση

5.1 Εισαγωγή

Ο ταυτόχρονος εντοπισμός θέσης και χαρτογράφηση (SLAM) είναι ένα πρόβλημα όπου ένα κινούμενο αντικείμενο χρειάζεται να κατασκευάσει το χάρτη ενός άγνωστου περιβάλλοντος, ενώ ταυτόχρονα εντοπίζει τη θέση του μέσα σε αυτόν το χάρτη. Υπάρχουν αρκετοί τομείς που θα μπορούσαν να επωφεληθούν από την εφαρμογή αλγορίθμων SLAM σε αυτόνομα οχήματα. Μερικά παραδείγματα θα ήταν η μεταλλευτική βιομηχανία, η υποβρύχια εξερεύνηση και η εξερεύνηση άλλων πλανητών. Το πρόβλημα SLAM εν γένει μπορεί να διατυπωθεί χρησιμοποιώντας μια συνάρτηση πυκνότητας πιθανότητας $p(x_t, m|z_{1:t}, u_{1:t})$,

όπου το x_t είναι η θέση του οχήματος, m ο χάρτης, $z_{1:t}$ το διάνυσμα όλων των παρατηρήσεων και $u_{1:t}$ είναι το διάνυσμα των σημάτων ελέγχου του οχήματος, που είναι είτε οι ίδιες οι εντολές ελέγχου είτε η οδομετρία, ανάλογα με την εφαρμογή.

5.1.1 Συσχέτιση δεδομένων

Η έννοια της συσχέτισης δεδομένων είναι ουσιαστικά η διερεύνηση της σχέσης μεταξύ παλιότερων και νεότερων δεδομένων που συλλέγονται. Στο πλαίσιο του SLAM, είναι αναγκαίο να συσχετίζουμε παλιότερες με νεότερες μετρήσεις. Αυτό επιτρέπει τον προσδιορισμό της θέσης των ορόσημων (ή σημείων αναφοράς, landmarks) στο περιβάλλον και έτσι, συνεπώς, παρέχονται και πληροφορίες σχετικά με τη θέση του ρομπότ μέσα στο χάρτη.

5.1.2 Κλείσιμο βρόχου

Η έννοια του κλεισίματος βρόχου στο πλαίσιο του προβλήματος SLAM είναι η ικανότητα του οχήματος να αναγνωρίζει ότι έχει ήδη επισκεφθεί μια τοποθεσία όταν την ξαναεπισκέπτεται. Με την εφαρμογή ενός αλγορίθμου κλεισίματος βρόχου, η ακρίβεια τόσο του χάρτη όσο και της θέσης του οχήματος μέσα σε αυτόν μπορούν να αυξηθούν. Ωστόσο, αυτό δεν είναι εύκολο να επιτευχθεί, λόγω του γεγονότος ότι το περιβάλλον λειτουργίας του οχήματος είναι πιθανό να περιέχει τοποθεσίες των οποίων τα χαρακτηριστικά μοιάζουν με αυτά άλλων τοποθεσιών στο ίδιο περιβάλλον. Σε αυτήν την περίπτωση αν ο αλγόριθμος κλεισίματος βρόχου λειτουργεί ανεπαρκώς, θα μπορούσε να οδηγήσει σεελαττωματικό κλείσιμο βρόχου, γεγονός το οποίο μπορεί να καταστρέψει το χάτη καθώς και τη θέση του οχήματος σε αυτόν.

5.1.3 Full SLAM και online SLAM

Υπάρχουν δύο διαφορετικά είδη προβλημάτων SLAM. Το πρώτο είναι το online SLAM, στο οποίο εκφράζονται μόνο η τρέχουσα θέση του ρομπότ x_t και ο χάρτης m, δεδομένης της εισόδου ελέγχου $u_{1:t}$ και των μετρήσεων $z_{1:t}$. Το δεύτερο είναι το πρόβλημα full SLAM που εκφράζει όλη την τροχιά του ρομπότ. Η συνάρτηση πυκνότητας πιθανότητας του full SLAM συμβολίζεται ως $p(x_{0:t}, m | z_{1:t}, u_{1:t})$, όπου λαμβάνονται υπόψη όλες οι θέσεις του ρομπότ συμπεριλαμβανομένης και της αρχικής του θέσης x_0.

5.1.4 Γενικά μοντέλα προβλημάτων SLAM

Ένα μοντέλο κίνησης για το πρόβλημα SLAM μπορεί να περιγραφεί από την εξίσωση:

$$x_t = g(u_t, x_{t-1}) + \delta_t \quad (5.1)$$

όπου η τρέχουσα θέση x_t εξαρτάται από την πιθανώς μη γραμμική συνάρτηση $g(u_t, x_{t-1})$, όπου u_t είναι η είσοδος ελχου, x_{t-1} η θέση στο προηγούμενο βήμα και δ_t μια τυχαία γκαουσιανή μεταβλητή με μηδενική μέση τιμή.

Ένα μοντέλο μετρήσεων για το πρόβλημα SLAM μπορεί να περιγραφεί από την εξίσωση:

$$z_t = h(x_t, m) + \varepsilon_t \quad (5.2)$$

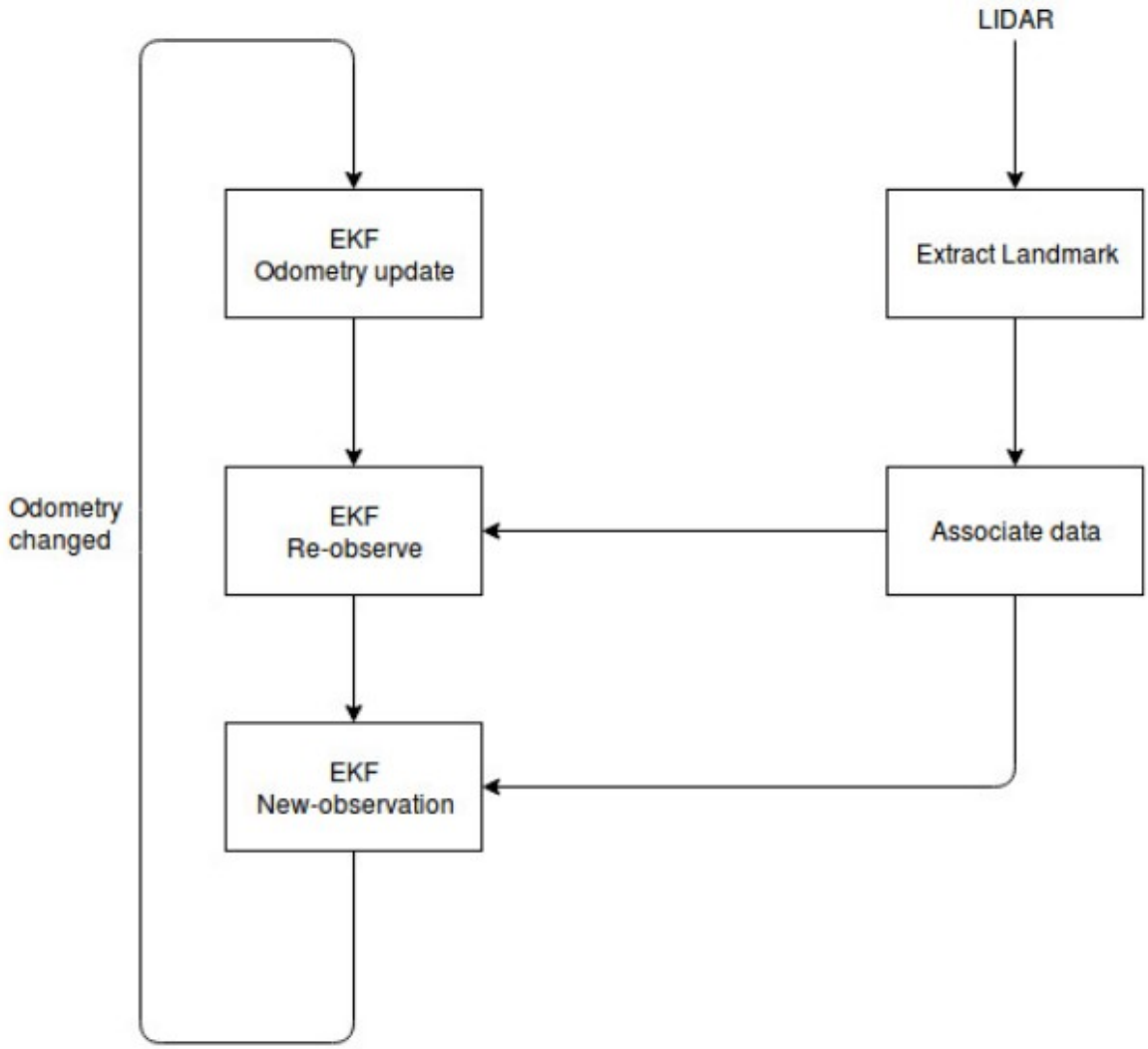
όπου η τρέχουσα μέτρηση z_t εξαρτάται από την πιθανώς μη γραμμική συνάρτηση $h(x_t, m)$, όπου το x_t συμβολίζει την τρέχουσα θέση, το m το χάρτη και το ε_t μια τυχαία γκαουσιανή μεταβλητή με μηδενική μέση τιμή.

5.2 SLAM με Εκτεταμένο Φίλτρο Kalman (Extended Kalman Filter SLAM)

Η προσέγγιση Extended Kalman Filter SLAM (EKF) για την επίλυση του προβλήματος SLAM γίνεται με χρήση δεδομένων αισθητήρων που συλλέγονται από την κίνηση και την περιστροφή του ρομπότ. Αυτό μπορεί να γίνει για παράδειγμα χρησιμοποιώντας κωδικοποιητές (encoders) τροχών και ένα γυροσκόπιο. Επιπλέον, είναι απαραίτητη η συγκέντρωση πληροφοριών για το περιβάλλον, για παράδειγμα με τη χρήση LIDAR (αισθητήρας φωτοεντοπισμού) ή αισθητήρων απόστασης όπως γίνεται στην εργασία αυτή. Με αυτά τα δεδομένα, ο αλγόριθμος παρακολουθεί τις πιθανές θέσεις του ρομπότ μέσα σε ένα χάρτη καθώς και τις θέσεις συγκεκριμένων ορόσημων που παρατηρούνται.

5.2.1 Αλγόριθμος

Όταν το ρομπότ είναι ενεργοποιημένο, οι αισθητήρες του ρομπότ όπως οι κωδικοποιητές τροχών και το γυροσκόπιο συλλέγουν πληροφορίες σχετικά με τη θέση του ρομπότ. Επιπλέον, τα ορόσημα του περιβάλλοντος εξάγονται βάσει νέων παρατηρήσεων από τους αισθητήρες απόστασης του ρομπότ (ή το LIDAR). Αυτές οι νέες παρατηρήσεις συσχετίζονται με προηγούμενες παρατηρήσεις και ενημερώνονται στον αλγόριθμο EKF. Ωστόσο, αν μια παρατήρηση ενός ορόσημου δεν μπορεί να συσχετιστεί με μια προηγούμενη, η ίδια η παρατήρησηπαρουσιάζεται στον αλγόριθμο ως νέα. Η διαδικασία αυτή απεικονίζεται στην Εικόνα 5.1



Εικόνα 5.1 Ο αλγόριθμος EKF SLAM (Πηγή: Johann Alexandersson, Olle Nordin, Implementation of SLAM algorithms in a small-scale vehicle using model-based development)

Πρώτα απ' όλα, το ρομπότ εντοπίζει τον ευατό του μέσα στο χάρτη που δημιουργείται χρησιμοποιώντας ήδη παρατηρημένα ορόσημα και πληροφορίες από τους αισθητήρες. Τα ορόσημα πρέπει κατά προτίμηση να είναι παρατηρήσιμα από πολλές γωνίες, καθώς και να βρίσκονται σε θέσεις τέτοιες ώστε να μπορούν να συσχετιστούν με άλλα ορόσημα. Αυτές οι προϋποθέσεις προσδίδουν στον αλγόριθμο EKF τη δυνατότητα να διακρίνει μεταξύ των οροσήμων σε μεταγενέστερο χρόνο. Επιπλέον, τα ορόσημα που χρησιμοποιούνται πρέπει να είναι στάσιμα.

Για να περιγράψουμε τη διαδικασία του EKF SLAM εν συντομία, μπορούμε να πούμε ότι αποτελείται από δύο στάδια:

- Βήμα πρόβλεψης

  Πρόβλεψε την τρέχουσα κατάσταση, η οποία εκφράζεται από τον προβλεπόμενο μέσο όρο και την προβλεπόμενη συνδιακύμανση. Αυτά υπολογίζονται με βάση την είσοδο ελέγχου, τους πίνακες κινηματικής του ρομπότ, την προηγούμενη συνδιακύμανση και μέσο όρο της θέσης.

- Βήμα διόρθωσης

  Πρώτα απ' όλα, η ιδέα πίσω από το βήμα διόρθωσης είναι η συσχέτιση δεδομένων. Η κύρια ιδέα πίσω από τη συσχέτιση δεδομένων στο πλαίσιο του SLAM είναι να γίνεται διάκριση μεταξύ ενός παλαιότερα παρατηρημένου ορόσημου και ενός πρόσφατα πατατηρούμενου. Όταν πραγματοποιείται μια νέα μέτρηση, η διαδικασία έχει ως εξής:

  - Αποθήκευσε τις θέσεις των προσφάτως παρατηρούμενων ορόσημων
  - Συσχέτισε τα παλιότερα ορόσημα με τα καινούρια

Επιπλέον υπολογίζεται το κέρδος διόρθωσης, γνωστό και ως κέρδος Kalman. Στην ουσία είναι ένας συντελεστής διόρθωσης που απαιτείται για την ενημέρωση του μέσου όρου και της συνδιακύμανσης της τρέχουσας κατάστασης. Ο τρέχων μέσος όρος και συνδιακύμανση δεν εξαρτώνται μόνο από το κέρδος Kalman αλλά λαμβάνονται υπόψη και οι προβλεπόμενες τιμές τους.

5.2.2 Υπολογιστική Πολυπλοκότητα

Το υπολογιστικό κόστος του αλγόριθμου EKF SLAM είναι αρκετά μεγάλο σε σύγκριση με άλλους αλγορίθμους SLAM. Όταν ανιχνεύονται νέα ορόσημα, προστίθενται στο διάνυσμα κατάστασης του φίλτρου και το μέγεθος του χάρτη με τα N αναγνωρισμένα ορόσημα αυξάνεται γραμμικά. Όσον αφορά τους αριθμούς, η χρήση μνήμης είναι $O(N^2)$. Το υπολογιστικό κόστος για ένα βήμα του αλγορίθμου είναι περίπου $O(N^2)$, το συνολικό κόστος για ολόκληρο τον αλγόριθμο είναι $O(N^3)$, εξαρτώμενο σε μεγάλο βαθμό από το μέγεθος του χάρτη.

5.2.3 Συσχέτιση Δεδομένων

Ένα πρόβλημα που εμφανίζει ο αλγόριθμος EKF SLAM είναι το υπολογιστικό κόστος που αυξάνεται όταν παρατηρούνται νέα ορόσημα. Αυτό οδηγεί στην προϋπόθεση ότι το περιβάλλον δεν πρέπει να περιέχει πάρα πολλά ορόσημα. Ωστόσο, δεδομένου ότι ο αριθμός των ορόσημων στο περιβάλλον είναι μικρός, καθίσταται πιο δύσκολος ο εντοπισμός στο χάρτη και η συσχέτιση των δεδομένων. Επιπλέον, η συσχέτιση γίνεται με υπολογισμό της μέγιστης πιθανότητας και σε περίπτωση ελαττωματικής συσχέτισης, τα μελλοντικά αποτελέσματα που βασίζονται σε προηγούμενες συσχετίσεις θα είναι επίσης ελαττωματικά.

5.3. FastSLAM

Ένας άλλος τρόπος επίλυσης του προβλήματος SLAM είναι με χρήση του αλγόριθμου FastSLAM, ο οποίος χρησιμοποιεί το Rao-Blackwellized φίλτρο σωματιδίων. Ο αλγόριθμος FastSLAM εκμεταλλεύεται κάτι που οι περισσότεροι αλγόριθμοι SLAM δεν εκμεταλλεύονται, και συγκεκριμένα το γεγονός ότι κάθε παρατήρηση αφορά μόνο ένα μικρό ποσοστό των μεταβλητών κατάστασης. Η θέση του ρομπότ εξαρτάται πιθανοτικά από την προηγούμενη θέση του, ενώ οι θέσεις των ορόσημων εξαρτώνται από τη θέση του ρομπότ. Αυτό δε λαμβάνεται υπόψη για παράδειγμα από τον αλγόριθμο EKF SLAM, ο οποίος με κάθε ενημέρωση πρέπει να υπολογίσει εκ νέου τον πίνακα συνδιακύμανσης, με αποτέλεσμα την κακή κλιμάκωση σε μεγάλους χάρτες. Αυτό δεν αποτελεί πρόβλημα στο FastSLAM, καθώς διαχωρίζει τους υπολογισμούς σε απλούστερα υποσυστήματα, όπως φαίνεται από την εξίσωση:

$$p(s^t, \theta | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t) \prod_{n=1}^{N} p(\theta_n | s^t, z^t, u^t, n^t) \quad (5.3)$$

στην οποία διαχωρίζεται η διαδρομή του ρομπότ s^t και το κάθε ένα από τα N ορόσημα θ_n. Μια σημαντική διαφορά μεταξύ του FastSLAM και του EKF SLAM είναι ότι εδώ υπολογίζεται όλη η διαδρομή s^t, και όχι μόνο η τρέχουσα θέση x_t. Αυτό σημαίνει ότι ο FastSLAM είναι αλγόριθμος full SLAM ενώ ο EKF SLAM είναι αλγόριθμος online SLAM.

5.3.1 Δομή σωματιδίων

Το φίλτρο σωματιδίων που χρησιμοποιείται στο FastSLAM αποτελείται από M σωματίδια, όπου το κάθε σωματίδιο ορίζεται ως:

$$S_t^{[m]} = (s^{t[m]}, \mu_{1,t}, \Sigma_{1,t}, \ldots, \mu_{N,t}, \Sigma_{N,t}) \quad (5.4)$$

Κάθε σωματίδιο περιέχει N+1 μεταβλητές: μια διαδρομή του ρομπότ $s^{t\,[m]}$ και N ορόσημα, όπου το κάθε ένα από αυτά αντιπροσωπεύεται από ένα EKF με μέση τιμή $\mu_{n,t}$ και συνδιακύμανση $\Sigma_{n,t}$. Κατά τη διάρκεια μιας επανάληψης του αλγορίθμου, η διαδρομή του ρομπότ $s^{t\,[m]}$ ενημερώνεται μία φορά, όπως και κάθε φίλτρο ορόσημου $\mu_{n,t}$, $\Sigma_{n,t}$.

5.3.2 Αλγόριθμος

Κάθε επανάληψη του αλγορίθμου μπορεί να συνοψιστεί σε τέσσερα βήματα. Αυτά παρουσιάζονται παρακάτω.

1. Για κάθε ένα από τα M σωματίδια, δοκίμασε μια νέα διαδρομή s_t από την $p(s_t|s_{t-1}^{[m]}, u_t)$
2. Με τη νέα μέτρηση, ενημέρωσε τα φίλτρα των ορόσημων
3. Δώσε σε κάθε σωματίδιο έναν παράγοντα σημαντικότητας (βάρος)
4. Ξαναεπίλεξε τα σωματίδια, με εκείνα που έχουν μεγαλύτερο βάρος να είναι πιθανότερο να ξαναεπιλεγούν

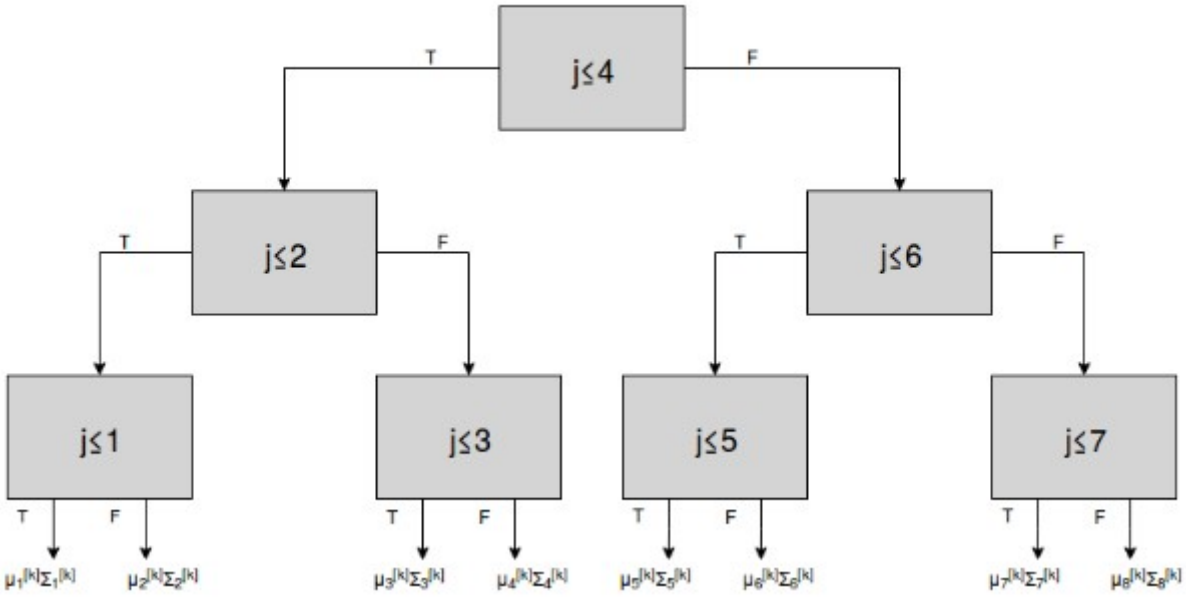
Στο βήμα 1, υπολογίζεται μια νέα διαδρομή για κάθε σωματίδιο. Η διαδρομή αυτή υπολογίζεται από το κινηματικό μοντέλο, με το σήμα ελέγχου σαν είσοδο. Στη συνέχεια, οι πιθναές θέσεις των ορόσημων ενημερώνονται, με βάση τις νέες μετρήσεις. Μετά από αυτό, σε κάθε σωματίδιο δίνεται ένας παράγοντας σημαντικότητας (βάρος), που στην ουσία είναι ένα μέτρο του πόσο ρεαλιστικές είναι οι τιμές των μεταβλητών στο σωματίδιο. Τέλος, τα σωματίδια ανασυντίθενται, ώστε να κρατηθούν μόνο αυτά που έχουν μεγαλύτερη πιθανότητα.

5.3.3 Υπολογιστική Πολυπλοκότητα

Εάν ο FastSLAM εφαρμόζεται όπως περιγράφεται παραπάνω, έχει O(M*N) υπολογιστική πολυπλοκότητα, όπου M είναι ο αριθμός των σωματιδίων και N ο αριθμός των ορόσημων. Η εφαρμογή του αλγόριθμου κατ' αυτόν τον τρόπο δε θα ήταν καλή ιδέα, καθώς το κόστος θα αυξανόταν γραμμικά με τον αριθμό των ορόσημων και θα καθιστούσε τον αλγόριθμο ακατάλληλο για μεγάλους χάρτες.

Για να αντιμετωπισθεί το ζήτημα αυτό, ο FastSLAM υλοποιείται χρησιμοποιώντας δυαδικά δέντρα αντί για τη δομή του πίνακα που φαίνεται στην Εικόνα 5.4. Έτσι ο αλγόριθμος έχει πολυπλοκότητα O(M*log(N)), όπως απεικονίζεται στην Εικόνα 5.2, όπου φαίνεται ένα τέτοιο δέντρο ενός σωματιδίου με N=8.

Η χρήση της προσέγγισης με δυαδικό δέντρο επίσης μειώνει σημαντικά το πόση μνήμη χρησιμοποιείται από τον αλγόριθμο.



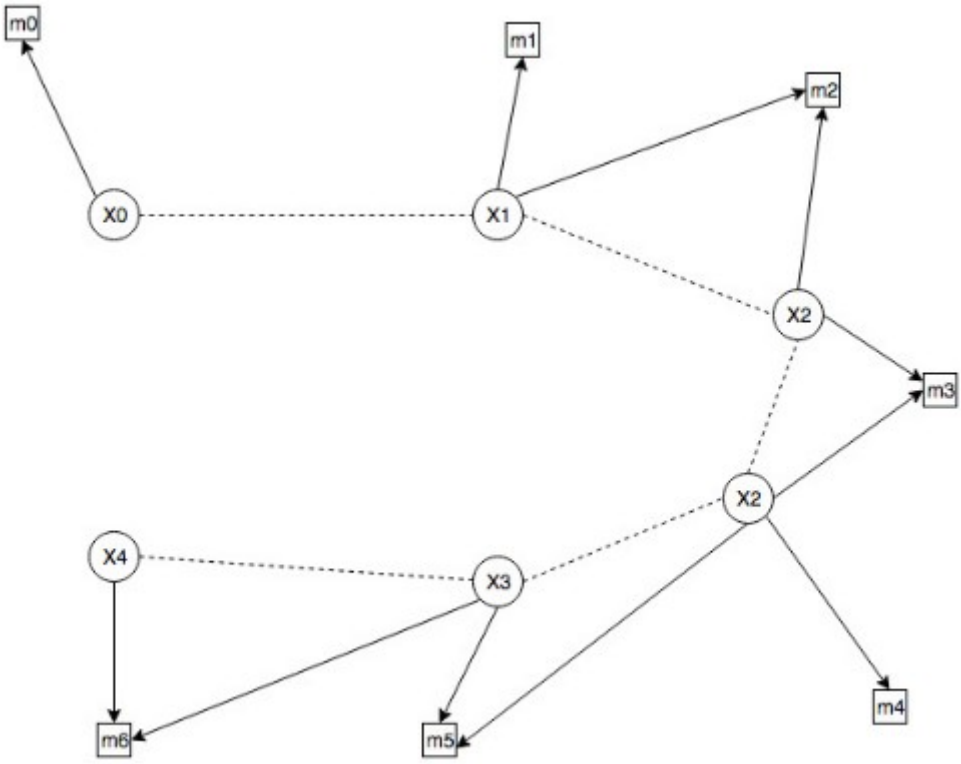
Εικόνα 5.1 Δομή δυαδικού δέντρου ενός σωματιδίου FastSLAM (Πηγή: Johann Alexandersson, Olle Nordin, Implementation of SLAM algorithms in a small-scale vehicle using model-based development)

5.3.4 Συσχέτιση Δεδομένων

Ένα άλλο πλεονέκτημα του αλγόριθμου FastSLAM είναι ότι η συσχέτιση δεδομένων γίνεται ανά σωματίδιο. Δηλαδή κάθε σωματίδιο έχει τη δική του υπόθεση για το ποιο ορόσημο σχετίζεται με τη μέτρηση. Εάν μια μέτρηση συσχετίζεται εσφαλμένα με ένα ορόσημο σε ένα σωματίδιο, τότε το σωματίδιο αυτό θα έχει χαμηλό παράγοντα σημαντικότητας, καθιστώντας την επιρροή αυτής της λανθασμένης συσχέτισης πολύ μικρή. Σε άλλους αλγόριθμους, όπως ο EKF SLAM, αν γίνει εσφαλμένη συσχέτιση δεδομένων, θα προκληθούν εσφαλμένα αποτελέσματα από αυτό το σημείο και μετά. Αυτό είναι ένα από τα βασικά πλεονεκτήματα του FastSLAM σε σύγκριση με άλλους αλγορίθμους και συμβάλλει στην υψηλή του ευρωστία.

5.4 GraphSLAM

Ο αλγόριθμος GraphSLAM αποτελεί μια άλλη προσέγγιση λύσης του προβλήματος SLAM. Η κύρια ιδέα πίσω από το GraphSLAM είναι η αντιμετώπιση του SLAM με χρήση γράφου. Ο γράφος περιέχει κόμβους που αντιπροσωπεύουν τις θέσεις του ρομπότ $x_0,\ldots,x_t$ καθώς και ορόσημα του χάρτη που συμβολίζονται ως $m_0,\ldots,m_t$. Ωστόσο, αυτό δεν αρκεί για να αντιμετωπισθεί το πρόβλημα SLAM, καθώς υπάρχουν και περιορισμοί μεταξύ των θέσεων x_t, $x_{t-1},x_{t-2},\ldots,x_{t-n}$ του ρομπότ και των ορόσημων $m_0,\ldots,m_t$. Αυτοί οι περιορισμοί αντιπροσωπεύουν την απόσταση μεταξύ γειτονικών θέσεων όπως x_t, x_{t-1} καθώς και την απόσταση μεταξύ των θέσεων αυτών και των ορόσημων $m_0,\ldots,m_t$. Οι περιορισμοί αυτοί κατασκευάζονται με βάση τις πληροφορίες οδομετρίας που λαμβάνονται. Ο γράφος που προαναφέρθηκε μπορεί να μετατραπεί σε αραιή μήτρα, που σε αντίθεση με μια πυκνή μήτρα μπορεί να χρησιμοποιηθεί πιο αποτελεσματικά. Η Εικόνα 5.3 απεικονίζει το πώς τα ορόσημα $m_0,\ldots,m_t$, οι θέσεις $x_0,\ldots,x_t$ και οι περιορισμοί συνδέονται μεταξύ τους.



Εικόνα 5.3 Οι θέσεις του ρομπότ απεικονίζονται ως κύκλοι, τα ορόσημα ως τετράγωνα, οι διακεκομμένες γραμμές αντιπροσωπεύουν την κίνηση του ρομπότ βάσει των εντολών ελέγχου και τα βέλη την απόσταση από τα ορόσημα βάσει των μετρήσεων. (Πηγή: Johann Alexandersson, Olle Nordin, Implementation of SLAM algorithms in a small-scale vehicle using model-based development)

5.4.1 Αλγόριθμος

Η συνάρτηση πυκνότητας πιθανότητας για το full SLAM πρόβλημα συμβολίζεται $p(x_{0:t}, m | z_{1:t}, u_{1:t})$. Για λόγους απλότητας, όλες οι θέσεις $x_{0:t}$ και το σύνολο των ορόσημων m (ο χάρτης) μπορούν να περιγραφούν από ένα διάνυσμα του χώρου κατάστασης (Εικόνα 5.4).

$$y = \begin{bmatrix} x_o \\ \cdot \\ \cdot \\ \cdot \\ x_t \\ m \end{bmatrix}$$

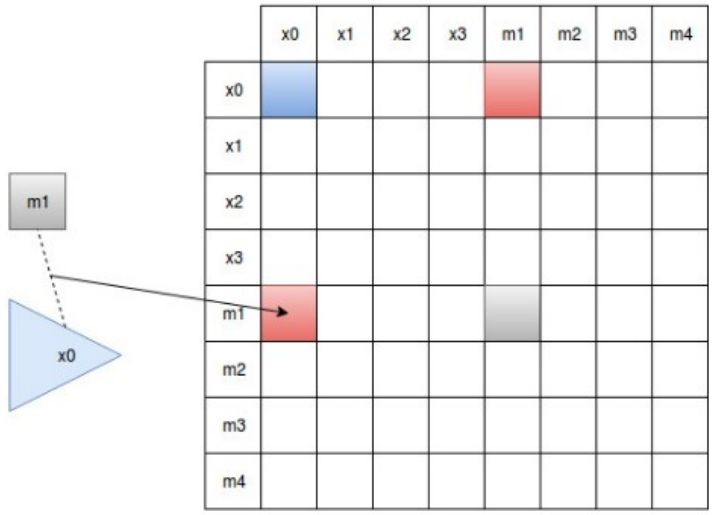
Εικόνα 5.4 Διάνυσμα του χώρου κατάστασης που περιέχει όλες τις θέσεις και τα ορόσημα

Στη συνέχεια προκειμένου να υπολογισθούν οι πιθανότερες θέσεις του ρομπότ και των ορόσημων στο χάρτη, πρέπει να χρησιμοποιηθεί ένας αλγόριθμος βελτιστοποίησης.

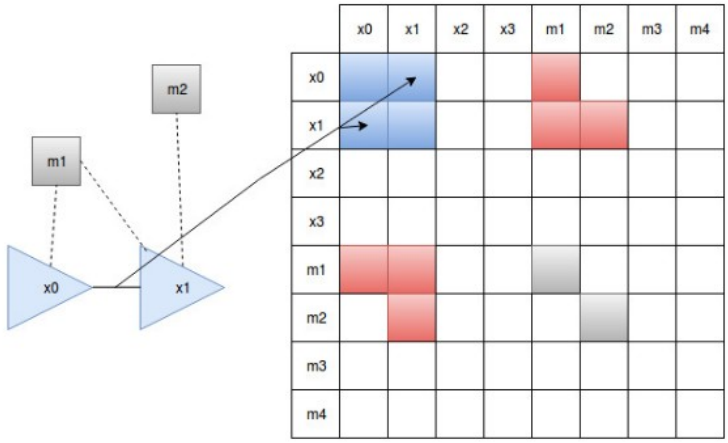
Επιπλέον, η συνάρτηση πυκνότητας πιθανότητας $p(y | z_{1:t}, u_{1:t})$ περιέχει τις ενδεχομένως μη γραμμικές συναρτήσεις $g(u_t, x_{t-1})$ και $h(x_t, m_i)$ όπου το m_i συμβολίζει το i-οστό ορόσημο που παρατηρείται τη στιγμή t. Ο αλγόριθμος βελτιστοποίησης δέχεται γραμμικές συναρτήσεις. Συνεπώς, ένα απαραίτητο βήμα είναι η γραμμικοποίηση των παραπάνω, πράγμα που μπορεί να επιτευχθεί με την εκτέλεση για παράδειγμα ενός αλγόριθμου γραμμικοποίησης Taylor.

Όπως αναφέρθηκε προηγουμένως, η προσέγγιση της λύσης του SLAM με τον αλγόριθμο GraphSLAM γίνεται με την εισαγωγή ενός γράφου. Αυτός ο γράφος μπορεί να μετατραπεί σε αραιή μήτρα. Στην ουσία, αυτή η αραιή μήτρα περιέχει όλες τις πληροφορίες που συλλέγει το ρομπότ καθώς κινείται στο περιβάλλον, όπως την κίνηση μέσα στο χάρτη και τα ορόσημα που παρατηρούνται από διαφορετικές θέσεις. Κατά συνέπεια, καθώς το ρομπότ κινείται η αραιή μήτρα αυξάνεται σε μέγεθος και και πληροφορία που περιέχει.

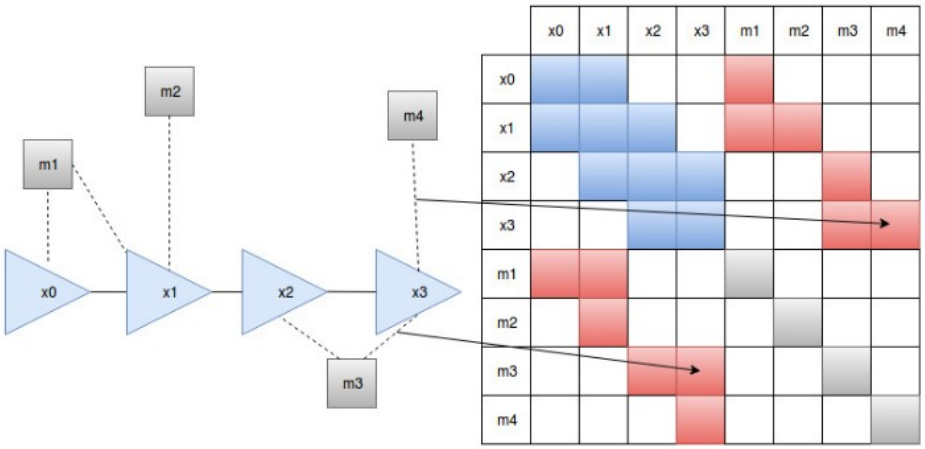
Στις Εικόνες που ακολουθούν 5.5, 5.6, 5.7, απεικονίζεται ο τρόπος με τον οποίο οι θέσεις του ρομπότ (μπλε τετράγωνα) και τα ορόσημα (κόκκινα τετράγωνα) εισάγονται στην αραιή μήτρα. Επιπλέον, τα μπλε τετράγωνα είναι γειτονικά μεταξύ τους λόγω της σχέσης εξάρτησης της κάθε θέσης από την προηγούμενη. Τα γκρίζα τετράγωνα της κύριας διαγωνίου δείχνουν ότι τα ορόσημα m_1, m_2, m_3, m_4 για κάθε σειρά είναι τα ίδια με τα ορόσημα της κάθε αντίστοιχης στήλης.



Εικόνα 5.5 Οι θέσεις του ρομπότ και τα παρατηρούμενα ορόσημα



Εικόνα 5.6 Οι θέσεις του ρομπότ και τα παρατηρούμενα ορόσημα



Εικόνα 5.7 Οι θέσεις του ρομπότ και τα παρατηρούμενα ορόσημα

5.4.2 Υπολογιστική Πολυπλοκότητα

Ορισμένα ενδιαφέροντα στατιστικά στοιχεία σχετικά με τον αλγόριθμο GraphSLAM είναι το υπολογιστικό του κόστος και η χρήση μνήμης. Σε σύγκριση με τον αλγόριθμο EKF SLAM του οποίου η χρήση μνήμης αυξάνεται με εξάρτηση $O(N^2)$ όταν εισάγονται N καινούρια ορόσημα και το υπολογιστικό κόστος με $O(N^3)$, η χρήση μνήμης του GraphSLAMαυξάνεται γραμμικά με την εισαγωγή N νέων ορόσημων. Όσο για το υπολογιστικό κόστος του, εξαρτάται από το χρόνο, συνεπώς, αν η διαδρομή του ρομπότ είναι μεγάλη όσον αφορά τη χρονική της διάρκεια, το κόστος μπορεί να γίνει πολύ υψηλό. Ωστόσο, γενικά τα χαρακτηριστικά του αλγορίθμου GraphSLAM είναι επιθυμητά όταν το περιβάλλον περιέχει πολλά ορόσημα, στην οποία περίπτωση ο EKF SLAM τείνει να αποτυγχάνει.

5.4.3 Συσχέτιση Δεδομένων

Το κομμάτι της συσχέτισης δεδομένων στον αλγόριθμο GraphSLAM πραγματοποιείται με τη χρήση ενός διανύσματος το οποίο περιέχει αντιστοιχίσεις μεταξύ των παρατηρημένων ορόσημων. Ο αλγόριθμος πραγματοποιεί δοκιμές αντιστοίχισης για να ελέγξει την πιθανότητα διαφορετικά ορόσημα του χάρτη να είναι στην πραγματικότητα το ίδιο ορόσημο. Αν το αποτέλεσμα αυτής της δοκιμής ξεπερνά μια ορισμένη τιμή, ο αλγόριθμος θα τα καθορίσει ως το ίδιο ορόσημο, διαφορετικά θα τα καθορίσει ως δύο ξεχωριστά. Σε σύγκριση με τον αλγόριθμο EKF SLAM όπου οι μελλοντικές συσχετίσεις δεδομένων μπορεί να είναι ελαττωματικές εξαιτίας παλιότερων συσχετίσεων, ο GraphSLAM μπορεί να επαναεξετάσει παλιότερες συσχετίσεις, μειώνοντας έτσι τον κίνδυνο εισαγωγής σφαλμάτων σε μελλοντικές. Ο αλγόριθμος GraphSLAM αποτελεί λύση του προβλήματος full SLAM, όπως και ο FastSLAM, και σε αντίθεση με τον EKF SLAM που λύνει μόνο το online SLAM πρόβλημα.

5.5 Σύγκριση αλγορίθμων

Υπάρχουν αρκετές παράμετροι που μας ενδιαφέρουν στη μελέτη των αλγορίθμων SLAM, οι πιο σημαντικές για την εργασία είναι: ευρωστία, απόδοση και συμβατότητα με το ROS. Αυτές οι παράμετροι αξιολογούνται και συγκρίνονται μεταξύ τους για κάθε αλγόριθμο παρακάτω.

- EKF
    - Ευρωστία

      Λαμβάνοντας υπόψη αυτά που αναφέρθηκαν στην παράγραφο 5.2.3, η ελαττωματική συσχέτιση δεδομένων οδηγεί σε εσφαλμένα μελλοντικά αποτελέσματα, συνεπώς ο EKF έχει μικρή ανοχή σε σφάλματα.

- Απόδοση

  Στην παράγραφο 5.2.2 αναφέρθηκε ότι το συνολικό υπολογιστικό κόστος εκτέλεσης του αλγορίθμου είναι $O(N^3)$, εξαρτώμενο σε μεγάλο βαθμό από το μέγεθος του χάρτη. Συμπερασματικά, ο EKF είναι καταλληλότερος για μικρότερους χάρτες με λίγα ορόσημα, εξαιτίας του υψηλού κόστους εκτέλεσης. Ωστόσο, όπως αναφέρθηκε στην ενότητα 5.2.3 ο εντοπισμός θέσης και η συσχέτιση δεδομένων γίνεται δυσκολότερη όσο μικραίνει ο αριθμός των ορόσημων.

- Συμβατότητα με ROS

  Έχει υλοποιηθεί στο πακέτο ROS mrpt_ekf_slam_2d.

- FastSLAM
  - Ευρωστία

    Δεδομένων όσων ειπώθηκαν στην ενότητα 5.3.5, ο αλγόριθμος FastSLAM είναι σημαντικά πιο εύρωστος από τον EKF λόγω του γεγονότος ότι οι συσχετίσεις γίνονται σε κάθε σωματίδιο μειώνοντας έτσι το πόσο επηρεάζει ένα σφάλμα τα μελλοντικά αποτελέσματα.

  - Απόδοση

    Υπολογίσαμε πως το συνολικό υπολογιστικό κόστος είναι O(M*log(N)), που σημαίνει ότι η πολυπλοκότητα εξαρτάται από τον αριθμό των σωματιδίων M και των ορόσημων N.

  - Συμβατότητα με ROS

    Έχει υλοποιηθεί στο πακέτο ROS gmapping.

- GraphSLAM
  - Ευρωστία

    Όπως προαναφέρθηκε, η ευρωστία του αλγόριθμου GraphSLAM είναι υψηλότερη από του EKF, εν μέρει λόγω του γεγονότος πως στον GraphSLAM μπορούν να επαναεξεταστούν παλιότερες συσχετίσεις δεδομένων αν έχει γίνει λάθος. Στην ουσία αυτό μειώνει τον κίνδυνο εμφάνισης εσφαλμένων μελλοντικών συσχετίσεων, αυξάνοντας έτσι την ευρωστία.

  - Απόδοση

    Αναφέρθηκε ήδη πως η χρήση της μνήμης εξαρτάται γραμμικά από τον αριθμό των ορόσημων N, ενώ στον EKF έχει εξάρτηση $O(N^2)$. Το υπολογιστικό κόστος εξαρτάται από το χρόνο εκτέλεσης του αλγορίθμου, συνεπώς αν η διαδρομή είναι μεγάλη η εκτέλεσή του μπορεί να είναι δαπανηρή υπολογιστικά.

  - Συμβατότητα με ROS

    Έχει υλοποιηθεί στο πακέτο slam_karto.

Πως φαίνεται από τα παραπάνω, ο EKF SLAM δεν είναι ιδιαίτερα εύρωστος, ενώ ο GraphSLAM καθώς και ο FastSLAM έχουν μεγαλύτερη ευρωστία. Επιπλέον, όσον αφορά την απόδοση ο EKF είναι αρκετά δαπανηρός. Δεδομένου ότι όλοι οι παραπάνω αλγόριθμοι έχουν υλοποιηθεί σε πακέτα του ROS, κάθε ένας από αυτούς θα αποτελούσε βιώσιμη επιλογή. Ωστόσο, αφού ο EKF στερείται ευρωστίας και αποτελεσματικότητας, θα ήταν προτιμότερο να επιλέξουμε έναν από τους δύο πιο αποδοτικούς αλγόριθμους. Αυτοί είναι σε θέση να αντιμετωπίζουν πιο αποτελεσματικά μεγαλύτερους χάρτες όσον αφορά είτε το καθαρό τους μέγεθος είτε τον αριθμό των ορόσημων που περιέχουν. Σε χάρτες με εκατοντάδες ορόσημα, όπως για παράδειγμα όταν χαρτογραφούμε εξωτερικούς χώρους, η υπολογιστική πολυπλοκότητα του αλγορίθμου EKF SLAM μπορεί να αποδειχθεί τεράστιο πρόβλημα.

6

Περιβάλλον Εργασίας και Λογισμικό

6.1 Λειτουργικό Σύστημα

Εικόνα 6.1 Εικονίδιο Ubuntu

Το λειτουργικό που επιλέχθηκε για τη διπλωματική αυτή είναι το Ubuntu Linux, ελεύθερο στην πρόσβαση και βασισμένο εξολοκλήρου στο λειτουργικό Linux. Έχει αναπτυχθεί από μια μεγάλη κοινότητα ανθρώπων και είναι μια διανομή βασισμένη στο Debian, μία από τις πιο αξιόπιστες διανομές Linux. Επιλέχθηκε γιατί το ROS, που χρησιμοποιείται για τον έλεγχο του ρομποτικού οχήματος, είναι συμβατό μόνο με unix-like λειτουργικά συστήματα και συγκεκριμένα η υποστήριξη που παρέχεται για Ubuntu Linux είναι καλύτερη απ ότι για οποιοδήποτε άλλο λειτουργικό σύστημα. Η έκδοση που χρησιμοποιήθηκε είναι η 18.04.3 LTS.

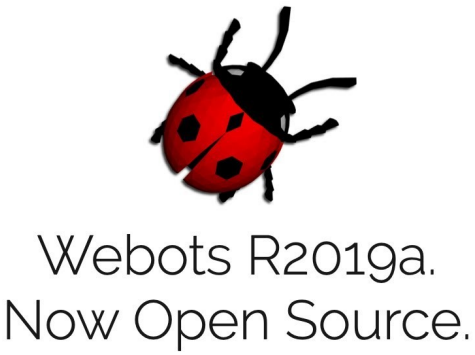
6.2 Γλώσσα προγραμματισμού

Εικόνα 6.2 Εικονίδιο C++

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για τη δημιουργία των βασικών αλγορίθμων του προγράμματος είναι η C++. Είναι μια μέσου επιπέδου γλώσσα, που εριλαμβάνει συνδυασμό χαρακτηριστικών από Γλώσσες υψηλού και χαμηλού επιπέδου. Η C++ κληρονόμησε το μεγαλύτερο μέρος της σύνταξης της C και τον προεπεξεργαστή της C. “Πατέρας” της γλώσσα είναι ο Μπιάρνε Στρούστρουπ, ο οποίος το 1979 στα εργαστήρια Bell της AT&T, βελτίωσε την ήδη υπάρχουσα γλώσσα προγραμματισμού C σε μια νέα γλώσσα, που αρχικά ονομάστηκε "C with Classes", δηλαδή C με Κλάσεις για να μετονομαστεί σε C++ το 1983. Αντιλαμβανόμαστε, λοιπόν, ότι η C++ είναι μια γλώσσα με αντικειμενοστραφή χαρακτηριστικά, η οποία σχεδιάστηκε για το προγραμματισμό διαφόρων συστημάτων, όπως είναι τόσο τα ενσωματωμένα όσο και συστήματα περιορισμένων πόρων(συστήματα με περιορισμένη CPU, μνήμη, ισχύ). Η επιλογή της C++ δίνει στους προγραμματιστές των προαναφερθέντων τύπων προβλημάτων βελτιωμένες επιδόσεις, επιθυμητή απόδοση και ευελιξία ως προς τη χρήση . Έτσι, η C++ έχει μεγάλο εύρος εφαρμογών, όπως ηλεκτρονικό εμπόριο, μηχανές αναζήτησης διαδικτύου, τηλεπικοινωνίες κ.α.

Ένας ακόμη λόγος που χρησιμοποιήθηκε η συγκεκριμένη γλώσσα είναι πως το ROS υποστηρίζει 3 γλώσσες προγραμματισμού, τη C++, την Python και τη Lisp, συνεπώς ήταν η μία από τις τρεις μόνο που μπορούσαν να χρησιμοποιηθούν.

6.3 Webots



Εικόνα 6.4 Εικονίδιο Webots

Το Webots αποτελεί ένα δωρεάν ανοικτού κώδικα πρόγραμμα 3D προσομοίωσης που χρησιμοποιείται στη βιομηχανία, την εκπαίδευση και την έρευνα. Ξεκίνησε το 1996 όταν αναπτύχθηκε αρχικά από τον Dr. Olivier Michel στην Ομοσπονδιακή Πολυτεχνική Σχολή της Λωζάνης στην Ελβετία και στη συνέχεια αγοράστηκε από τη Cyberbotics και κυκλοφορούσε ως λογισμικό ιδιόκτητης άδειας. Από το Δεκέμβριο του 2018 διανέμεται υπό τους όρους της δωρεάν και ανοικτού κώδικα άδειας Apache 2 License.

6.4 Robot Operating System (ROS)

Εικόνα 6.3 Εικονίδιο ROS

Το ROS είναι ένα ευέλικτο πλαίσιο που χρησιμοποιείται για την ανάπτυξη λογισμικού ρομπότ, που παρέχει σε ένα ετερογενές σύμπλεγμα υπολογιστών λειτουργίες που μοιάζουν με αυτές που παρέχει ένα λειτρουργικό σύστημα, παρότι δεν αποτελεί τέτοιο. Δημιουργήθηκε αρχικά το 2007 από το Εργαστήριο Τεχνητής Νοημοσύνης του Stanford. Από το 2008, η ανάπτυξή του συνεχίστηκε κυρίως από τη Willow Garage μέχρι και το 2013, όταν ο κύριος συντηρητής έγινε το Open Source Robotics Foundation (OSRF).

Πρόκειται για μια συλλογή εργαλείων, βιβλιοθηκών και συμβάσεων που αποσκοπούν στην απλούστευση του έργου δημιουργίας περίπλοκων και εύρωστων συμπεριφορών ρομπότ σε μια μεγάλη ποικιλία ρομποτικών πλατφόρμων. Βασίζεται σε μια αρχιτεκτονική γράφων, όπου η επεξεργασία πραγματοποιείται σε κόμβους που μπορούν να λαμβάνουν, ανα στέλνουν, και να πολυπλέκουν μηνύματα αισθητήρων, ελέγχου, κατάστασης, σχεδίασης, επενεργητών και άλλα.

Το ROS έχει δύο βασικές πλευρές. Την πλευρά των βασικών λειτουργιών (τύπου λειτουργικού συστήματος) όπως περιγράφονται παραπάνω, και την πλευρά των πακέτων ROS, μια σειρά πακέτων που έχουν συνεισφέρει οι χρήστες και υλοποιούν πολλές και χρήσιμες λειτουργίες και εφαρμογές. Κάποιες τέτοιες είναι προγράμματα οδήγησης υλικού, μοντέλα ρομπότ, τύπους δεδομένων, σχεδιασμός, αντίληψη, ταυτόχρονος εντοπισμός θέσης και χαρτογράφηση, εργαλεία προσομοίωσης και άλλοι αλγόριθμοι.

Τόσο τα ανεξάρτητα από τη γλώσσα εργαλεία όσο και οι κύριες βιβλιοθήκες πελατών (C ++, Python και Lisp) κυκλοφορούν υπό τους όρους της άδειας BSD και ως εκ τούτου αποτελούν λογισμικό ανοιχτού κώδικα και δωρεάν τόσο για εμπορική όσο και για ερευνητική χρήση. Η πλειοψηφία των άλλων πακέτων που κυκλοφορούν χρησιμοποιούν διάφορες άδειες ανοικτού κώδικα.

Οι κύριες βιβλιοθήκες πελατών του ROS απευθύνονται σε Unix-like συστήματα, κυρίως γιατί χρησιμοποιούν εξαρτήσεις (dependencies) λογισμικού ανοιχτού κώδικα.

6.4.1 Το πακέτο webots_ros

Το πακέτο webots_ros είναι ένα πακέτο ROS το οποίο περιλαμβάνει διάφορα services και εργαλεία τα οποία βοηθούν στη διασύνδεση του ROS με το πρόγραμμα προσομοίωσης Webots, παρέχοντας έλεγχο στους επενεργητές του ρομπότ ή πρόσβαση στα δεδομένα των αισθητήρων και άλλες χρήσιμες λειτουργίες που χρειάζονται για κάθε εφαρμογή.

6.4.2 Το πακέτο gmapping

Το gmapping είναι ένα πακέτο στο οποίο έχει υλοποιηθεί ο αλγόριθμος FastSLAM, δηλαδή μια προσέγγιση λύσης του SLAM με φίλτρο σωματιδίων Rao-Blackwellized, με βάση δεδομένα που λαμβάνονται από αισθητήρα LIDAR (φωτοεντοπισμού). Η προσέγγιση αυτή χρησιμοποιεί τα παραπάνω δεδομένα σε συνδυασμό με την οδομετρία του οχήματος, και είναι σχεδιασμένη να λειτουργεί με laser scanners μεγάης εμβέλειας. Γι' αυτό το λόγο χρειάζεται να κάνουμε μερικές μετατροπές, για να γίνει συμβατό το πακέτο με δεδομένα που προέρχονται από αισθητήρες προσέγγισης.

7

Προσομοίωση

7.1 Το ρομποτικό όχημα

Το ρομποτικό όχημα που χρησιμοποιήθηκε για την εργασία αυτή είναι το E-puck (Εικόνα 7.1). Το E-puck είναι ένα μικρών διαστάσεων κινούμενο ρομπότ, αρχικά σχεδιασμένο στην Ομοσπονδιακή Πολυτεχνική Σχολή της Λωζάνης από τους Michael Bonani και Francesco Mondada, στο Εργαστήριο Αυτόνομων Συστημάτων του καθηγητή Roland Siegwart. Πρόκειται για ένα ρομπότ ανοικτού υλικού και λογισμικού που κατασκευάζεται από διάφορες εταιρείες.

Εικόνα 7.1 Το κινούμενο ρομποτικό όχημα E-puck

Το E-puck είναι εφοδιασμένο με μεγάλο αριθμό συσκευών, που μαζί με τα διάφορα τεχνικά χαρακτηριστικά του δίνονται στον παρακάτω πίνακα (Εικόνα 7.2).

Feature	*Description*
Size	7.4 cm in diameter, 4.5 cm high
Weight	150 g
Battery	about 3 hours with the provided 5Wh LiION rechargeable battery
Processor	Microchip dsPIC 30F6014A @ 60MHz (about 15 MIPS)
Motors	2 stepper motors with 20 steps per revolution and a 50:1 reduction gear
IR sensors	8 infra-red sensors measuring ambient light and proximity of obstacles in a 4 cm range
Camera	color camera with a maximum resolution of 640x480 (typical use: 52x39 or 640x1)
Microphones	3 omni-directional microphones for sound localization
Accelerometer	3D accelerometer along the X, Y and Z axis
LEDs	8 red LEDs on the ring and one green LED on the body
Speaker	on-board speaker capable of playing WAV or tone sounds
Switch	16 position rotating switch
Bluetooth	Bluetooth for robot-computer and robot-robot wireless communication
Remote Control	infra-red LED for receiving standard remote control commands
Expansion bus	expansion bus to add new possibilities to your robot
Programming	C programming with the GNU GCC compiler system
Simulation	Webots facilitates the programming of e-puck with a powerful simulation, remote control and cross-compilation system

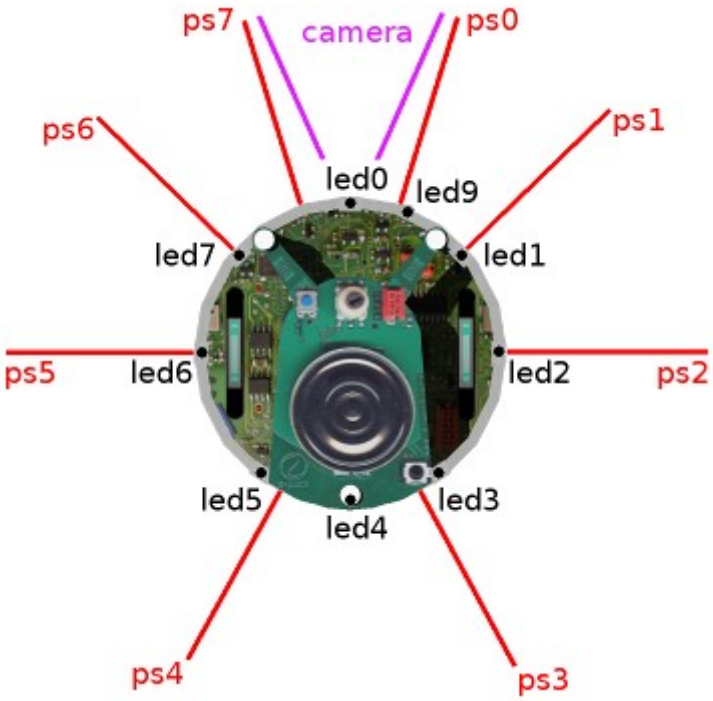
Εικόνα 7.2 Τα χαρακτηριστικά του E-puck

Ακόμη, από το επίσημο documentation (οδηγίες χρήσης) του ρομπότ δίνονται οι θέσεις και ο προσανατολισμός των 8 αισθητήρων προσέγγισης υπερύθρων πάνω στο σασί του ρομπότ.

Device	*x (m)*	*y (m)*	*z (m)*	*Orientation (rad)*
ps0	0.010	0.033	-0.030	1.27
ps1	0.025	0.033	-0.022	0.77
ps2	0.031	0.033	0.00	0.00
ps3	0.015	0.033	0.030	5.21
ps4	-0.015	0.033	0.030	4.21
ps5	-0.031	0.033	0.00	3.14159
ps6	-0.025	0.033	-0.022	2.37
ps7	-0.010	0.033	-0.030	1.87

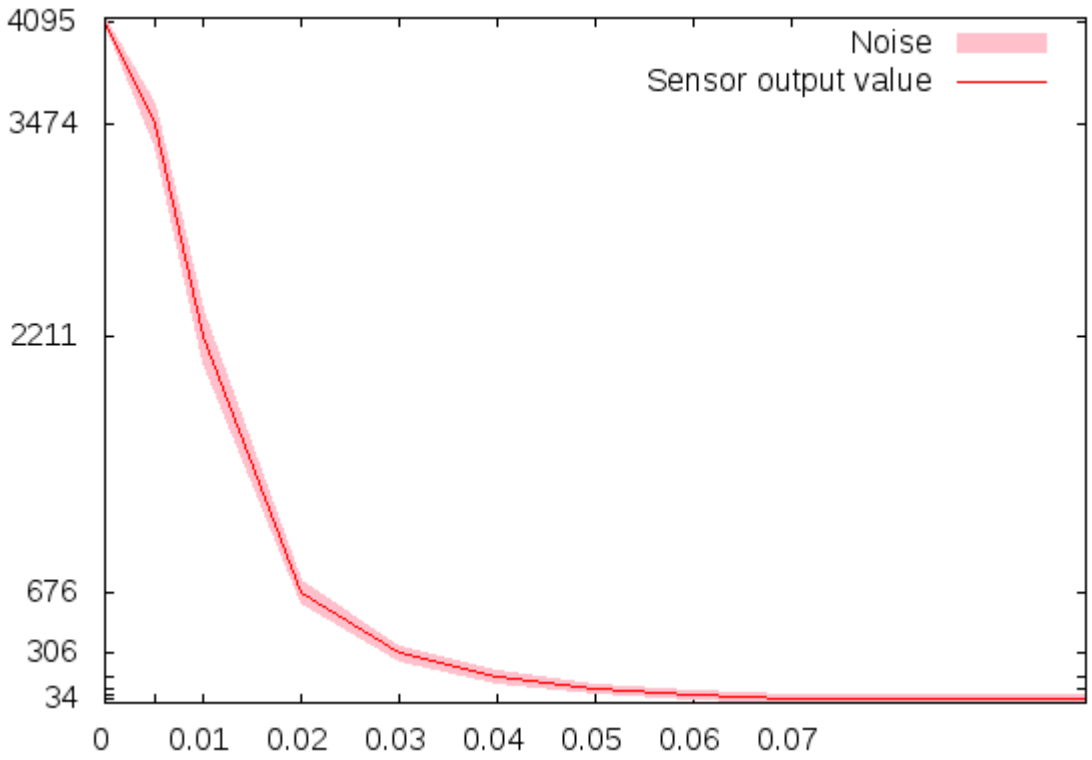
Εικόνα 7.3 Θέσεις και προσανατολισμός των αισθητήρων προσέγγισης

Να σημειωθεί ότι στον παραπάνω πίνακα, το σύστημα συντεταγμένων του ρομπότ είναι τέτοιο ώστε η μπροστινή του κατεύθυνση να δίνεται από τον αρνητικό άξονα z. Συνεπώς, οι θέσεις των αισθητήρων σχηματικά είναι οι εξής:



Εικόνα 7.4 Θέσεις των αισθητήρων προσέγγισης

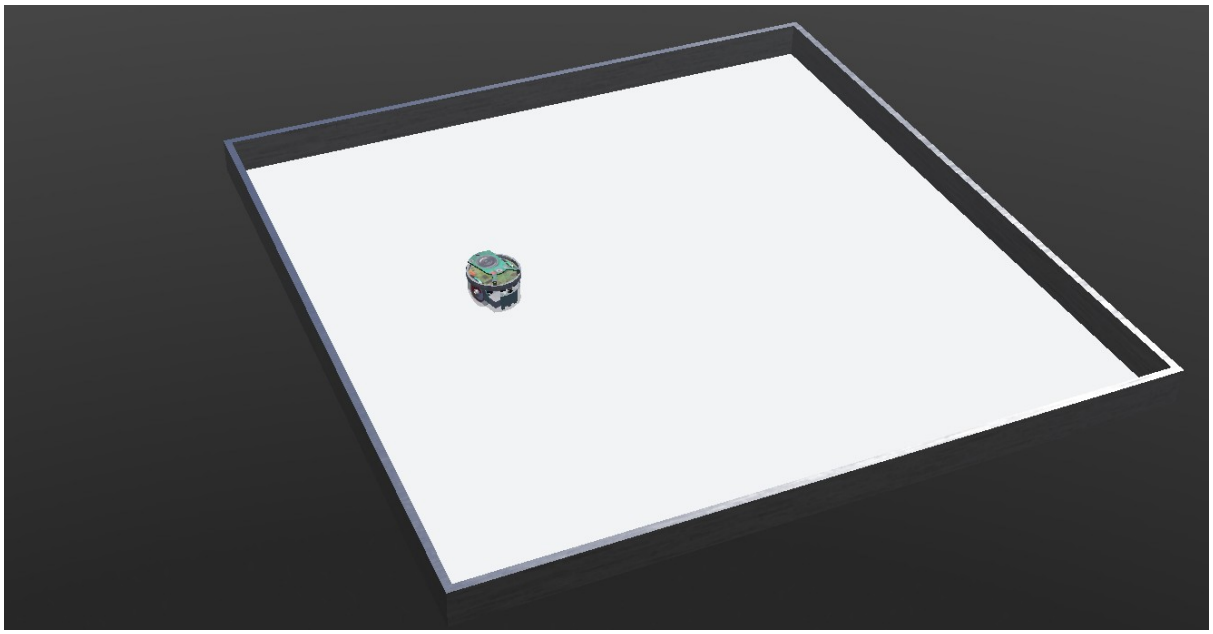
Σύμφωνα με το documentation πάλι του ρομπότ, η απόκριση των αισθητήρων για το πραγματικό ρομπότ συναρτήσει της απόστασης από το εντοπιζόμενο εμπόδιο έχει τη γραφική παράσταση:



Εικόνα 7.5 Απόκριση των αισθητήρων συναρτήσει της απόστασης σε m

7.2 Το προσομοιωμένο μοντέλο του ρομπότ

Τα χαρακτηριστικά που αναφέρονται στην προηγούμενη ενότητα αφορούν το πραγματικό όχημα E-puck. Ωστόσο, στην εργασία χρησιμοποιήθηκε το μοντέλο προσομοίωσής του, που παρέχεται από το Webots. Το προσομοιωμένο μοντέλο παρουσιάζει κάποιες διαφορές συγκριτικά με το πραγματικό μοντέλο, και για το λόγο αυτό χρειάστηκε μέσω μερικών απλών τεχνικών να μετρήσουμε τα αντίστοιχα χαρακτηριστικά του μοντέλου προσομοίωσης.



Εικόνα 7.6 Το E-puck σε περιβάλλον προσομοίωσης του Webots

Συγκεκριμένα, σύμφωνα με το documentation οι κινητήρες που διαθέτει το ρομπότ είναι βηματικοί με 20 βήματα ανά περιστροφή και μειωτήρα στροφών με λόγο 50:1. Συνεπώς, χρειάζονται 50 ολόκληρες περιστροφές του κινητήρα για μια περιστροφή του τροχού, δηλαδή 1000 βήματα για μια περιστροφή του τροχού. Επίσης, για το πραγματικό ρομπότ, η διάμετρος του τροχού είναι 41mm και το μήκοστου άξονα 52mm.

Θέτοντας τους δύο κινητήρες σε ίδια ταχύτητα και προχωρώντας την προσομοίωση βήμα βήμα, παρατηρούμε ότι όταν οι τροχοί ολοκληρώνουν 10 περιστροφές, το ρομπότ έχει μετακινηθεί σε ευθεία γραμμή για απόσταση 128.7cm. Μια περιστροφή του τροχού σημαίνει μετακίνηση του οχήματος κατά π*d, όπου d η διάμετρος του τροχού.

Άρα d=4.1cm. Δηλαδή η διάμετρος του τροχού είναι ίση με αυτή που δίνεται και για το πραγματικό μοντέλο.

Στη συνέχεια, θέτουμε πάλι τους τροχούς σε ίδια ταχύτητα μέχρι οι encoders τους να φτάσουν την τιμή 20. Παρατηρούμε πως το ρομπότ σταματά έχοντας διανύσει 40cm σε ευθεία γραμμή. Συνεπώς για κάθε βήμα που μετρούν οι encoders των τροχών το ρομπότ διανύει 2cm, που σημαίνει 0.155 περιστροφές τροχού σε κάθε βήμα, ή αλλιώς 6.44 βήματα για μια ολόκληρη περιστροφή του τροχού.

Τέλος, για να μετρήσουμε το μήκος του άξονα, θέτουμε τους κινητήρες σε ίδια ταχύτητα με αντίθετη φορά μέχρι να μετρήσουμε 10 βήματα στους encoders. Το ρομπότ περιστρέφεται στην ίδια θέση και σταματά αφού εκτελέσει περιστροφή κατά 7.0564 rad γύρω από τον εαυτό του. Αυτό σημαίνει 1.1231 περιστροφές. Ο κάθε τροχός σε 10 βήματα έχει μετακινηθεί κατά 20cm. Συνεπώς σε 1 περιστροφή του ρομπότ γύρω από τον εαυτό του , ο κάθε τροχός διανύει 20/1.1231=17.8079 cm.

Άρα η διάμετρος του κύκλου που εκτελούν τα σημεία επαφής των τροχών έχει μήκος 17.808/π=5.6686 cm.

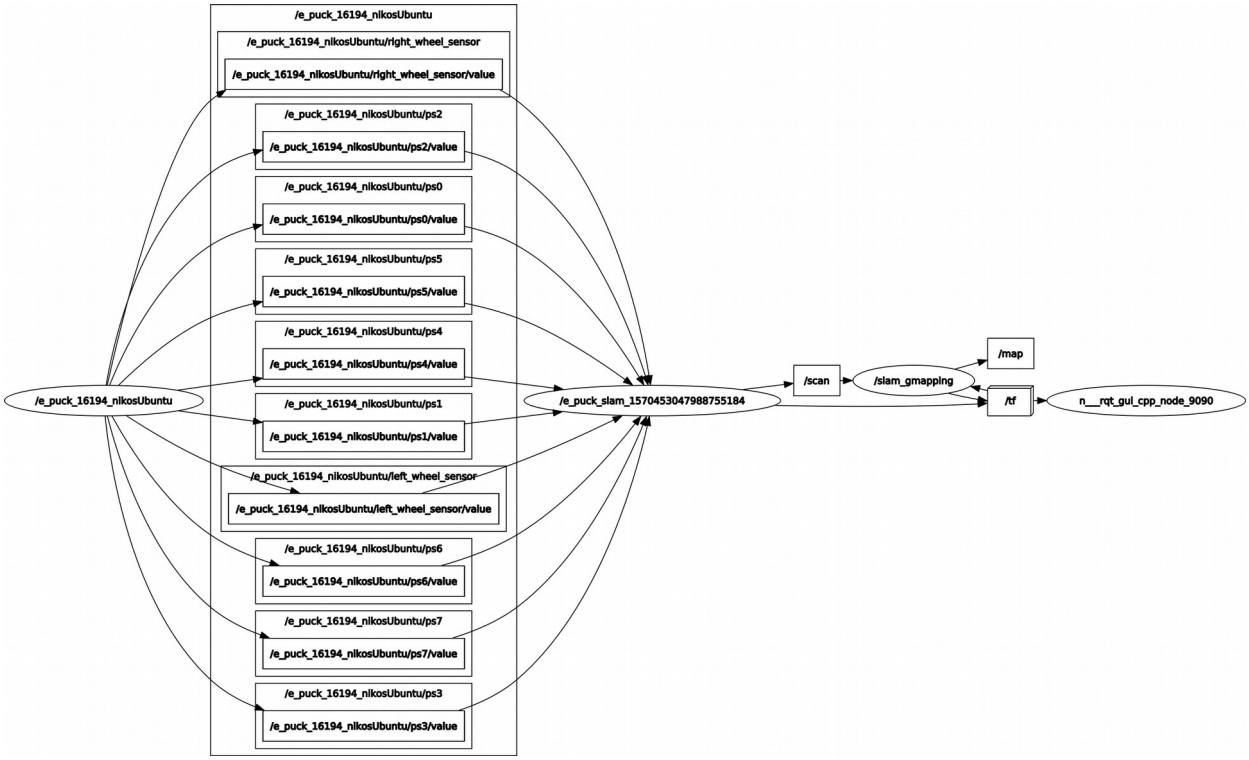
Τα παραπάνω χαρακτηριστικά που υπολογίσαμε είναι απαραίτητα για την εξαγωγή του κινηματικού μοντέλου του ρομπότ κατά την προσομοίωση και χρησιμοποιούνται για τον υπολογισμό της οδομετρίας σε κάθε βήμα εκτέλεσης του αλγορίθμου.

7.3 Κίνηση του ρομπότ

Για την κίνηση του ρομπότ χρησιμοποιήθηκε ένας αλγόριθμος που του επιτρέπει να αποφεύγει εμπόδια που συναντά (obstacle avoidance) καθώς και ένας ακόμη κατά την εκτέλεση του οποίου το ρομπότ ακολουθεί έναν τοίχο ή άλλο εμπόδιο. Επειδή ο συγκεκριμένος τρόπος όμως πολλές φορές οδηγεί το ρομπότ να ακολουθεί την ίδια διαδρομή συνεχώς χωρίς να επισκέπτεται αχαρτογράφητες περιοχές του περιβάλλοντός του, χρησιμοποιήθηκε και ένα πρόγραμμα που επιτρέπει στο χρήστη να ελέγξει την κίνηση του ρομπότ από το πληκτρολόγιο, ώστε να διορθώνει την κατεύθυνση του ρομπότ και να το κατευθύνει προς περιοχές του χάρτη που ακόμη είναι αχαρτογράφητες.

7.4 Υλοποίηση στο ROS

Για τον έλεγχο του οχήματος αλλά και τη συλλογή και επεξεργασία των δεδομένων των αισθητήρων του, υλοποιήθηκε ένα σύνολο nodes του ROS. Παρακάτω περιγράφεται η λειτουργία του κάθε ενός.



Εικόνα 7.7 Ο γράφος του ROS κατά την εκτέλεση του προγράμματος

7.4.1 e_puck_16194

Το node e_puck_16194 είναι ο κόμβος που διαχειρίζεται τον έλεγχο του ρομπότ καθώς και τα δεδομένα των αισθητήρων του. Δημιουργείται αυτόματα για κάθε ρομπότ της προσομοίωσης και έχει έναν μοναδικό αριθμό για να ξεχωρίζει από άλλα ρομπότ με ίδιο όνομα (στη συγκεκριμένη περίπτωση 16194). Η λειτουργία που εκτελεί είναι να διαβάζει συνεχώς τα δεδομένα των αισθητήρων και να τα στέλνει μέσω του κατάλληλου topic στο node e_puck_slam το οποίο τα διαχειρίζεται κατάλληλα. Ταυτόχρονα, δέχεται από τον κόμβο e_puck_slam εντολές που ρυθμίζουν την κίνηση των κινητήρων μέσω κατάλληλων ros services.

7.4.2 e_puck_slam

Ο κόμβος e_puck_slam λαμβάνει τις τιμές των αισθητήρων προσέγγισης από τον προηγούμενο κόμβο, οι οποίες έχουν τη μορφή μηνυμάτων τύπου Range. Η υλοποίηση που έχει γίνει για τον αλγόριθμο FastSLAM στο πακέτο ROS gmapping, απαιτεί είσοδο μηνύματος τύπου LaserScan. Η δουλειά του κόμβου αυτού λοιπόν είναι η μετατροπή των μηνυμάτων Range που δέχεται από τους αισθητήρες σε ένα μήνυμα τύπου LaserScan.

Ένα μήνυμα τύπου LaserScan περιέχει μετρήσεις αποστάσεων από μια προκαθορισμένη γωνία (angle_min) ως μια άλλη (angle_max). Η κάθε μέτρηση απέχει από την προηγούμενη σταθερή γωνία (angle_increment). Επειδή όμως οι αισθητήρες του ρομπότ δεν είναι τοποθετημένοι με σταθερή γωνία ο ένας από τον άλλο πρέπει να κάνουμε τους κατάλληλες μετατροπές. Συγκεκριμένα χρησιμοποιήθηκε ελάχιστη γωνία -90º, μέγιστη +90º και γωνία διαφοράς των μετρήσεων 10º, αν θεωρήσουμε πως η μπροστά κατεύθυνση του ρομπότ είναι ευθυγραμμισμένη προς τον άξονα x. Έτσι, για τις μετρήσεις στις γωνίες όπου δεν υπάρχει τοποθετημένος αισθητήρας, χρησιμοποιούνται οι μετρήσεις των γειτονικών αισθητήρων με κατάλληλα βάρη.

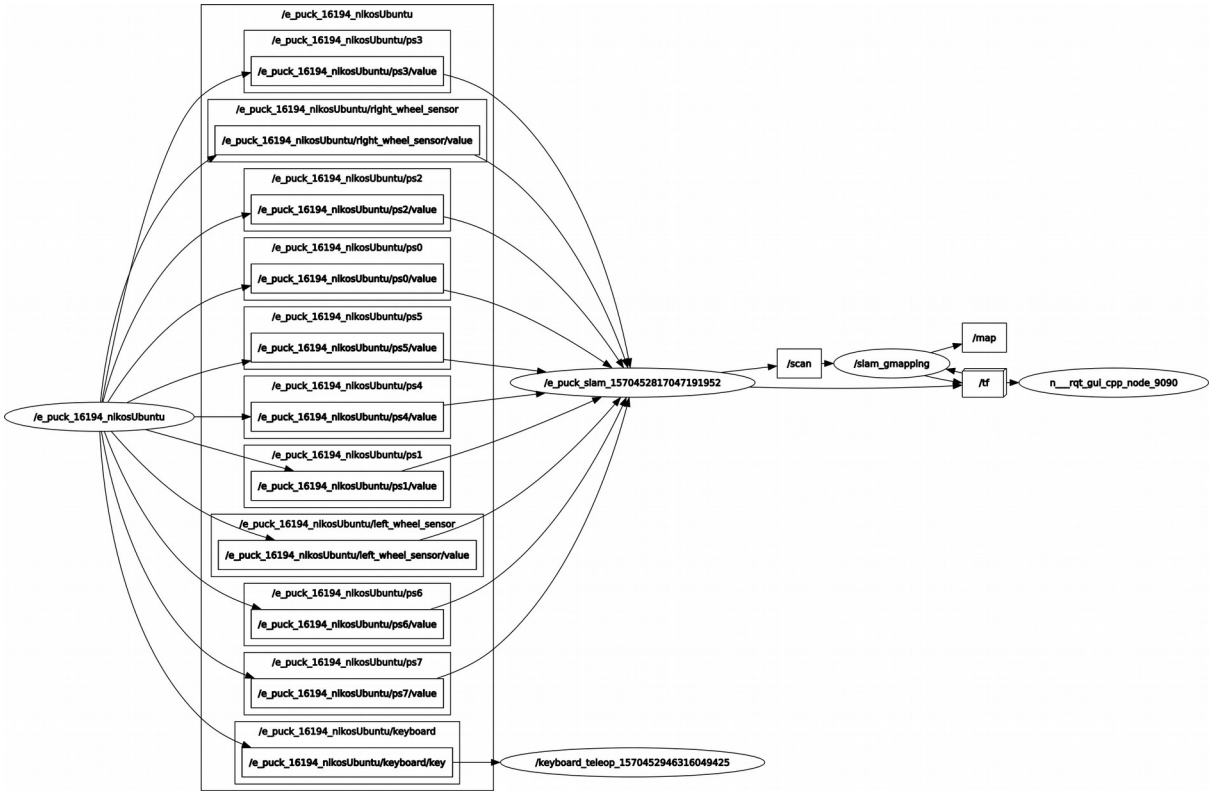
Επίσης δέχεται τα δεδομένα των encoders των τροχών και υπολογίζει την οδομετρία του οχήματος. Στη συνέχεια δημοσιεύει στο topic /scan το μήνυμα τύπου LaserScan για να το λάβει ο κόμβος /slam_gmapping και επίσης δημοσιεύει τον μετασχηματισμό (tf) του ρομπότ σε σχέση με την αρχική του θέση.

7.4.3 slam_gmapping

Ο κόμβος slam_gmapping λαμβάνει το μήνυμα τύπου LaserScan που δημιουργήσαμε στον προηγούμενο κόμβο καθώς και το μετασχηματισμό του συστήματος αναφοράς του ρομπότ από το αντικειμενικό σύστημα αναφοράς, το οποίο προκύπτει από την αρχική θέση του ρομπότ. Η λειτουργία του είναι να εκτελεί τον αλγόριθμο FastSLAM με παραμέτρους που θέτουμε εμείς, όπως το πλήθος των σωματιδίων, κάθε πότε να λαμβάνει καινούρια είσοδο από τους αισθητήρες κλπ. Ο κόμβος αυτός δημιουργεί με βάση τον αλγόριθμο αυτό το χάρτη του περιβάλλοντος, του οποίου η δημιουργία είναι και ο τελικός μας σκοπός, και τον δημοσιεύει στο topic /map.

7.4.4 keyboard_teleop

Επειδή κατά τη χαρτογράφηση το ρομπότ μπορεί να βρεθεί να εκτελεί συνεχώς την ίδια διαδρομή χωρίς να έχει χαρτογραφήσει κάποιες περιοχές του περιβάλλοντος, είτε να κολλήσει σε κάποιο εμπόδιο, χρησιμοποιούμε τον κόμβο keyboard_teleop για να το κατευθύνουμε προς τα εκεί που επιθυμούμε. Ο κόμβος αυτός επικοινωνεί με τον κόμβο e_puck_16194 και δίνει εντολές στους κινητήρες του ρομπότ.

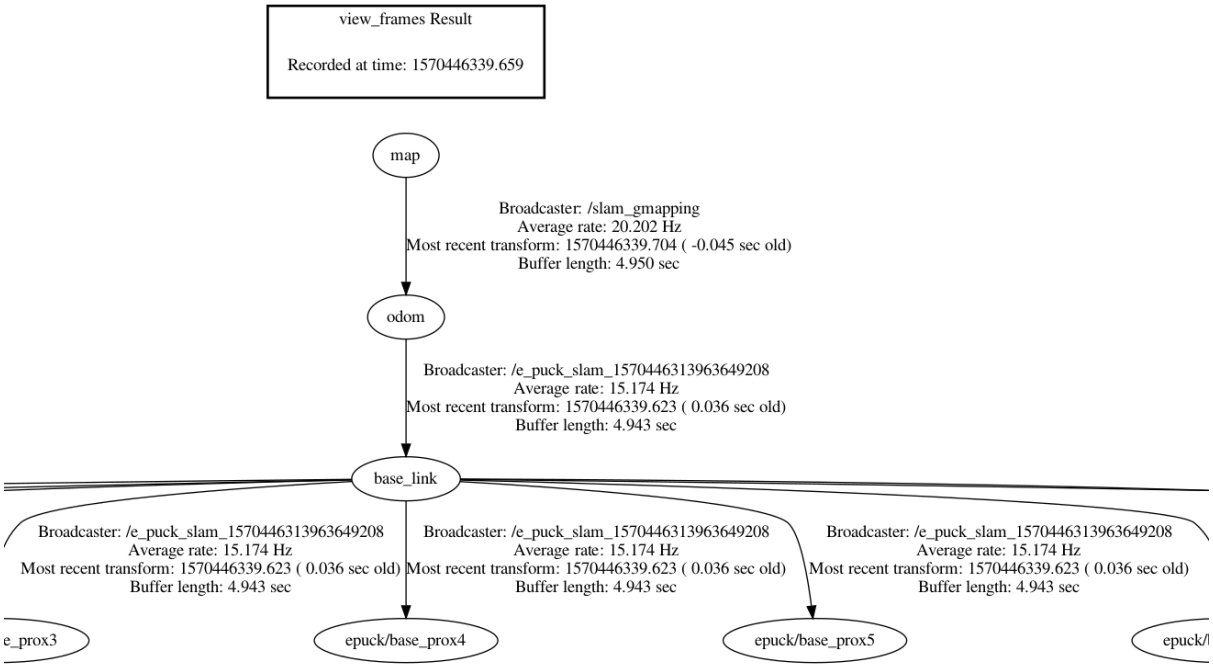


Εικόνα 7.8 Ο γράφος του ROS κατά την εκτέλεση του προγράμματος μαζί με τον κόμβο keyboard_teleop

7.4.5 Μετασχηματισμοί tf

Ένα πολύ βασικό κομμάτι του προγράμματός μας είναι ο σωστός μετασχηματισμός μεταφύ των συστημάτων αναφοράς που δημιουργούνται. Το σύστημα συντεταγμένων odom χρησιμοποιείται ως το αντικειμενικό σύστημα αναφοράς και καθορίζεται από την αρχική θέση και προσανατολισμό του ρομπότ. Το σύστημα base_link εκφράζει το τοπικό σύστημα αναφοράς του ρομπότ και η αρχή των αξόνων του βρίσκεται στο κέντρο του ρομπότ, ο άξονας x έχει την μπροστινή κατεύθυνση του ρομπότ και ο z είναι ο κατακόρυφος άξονας. Το σύστημα αναφοράς map εκφράζει τη θέση του χάρτη σε σχέση με την αρχική θέση του ρομπότ. Υπό ιδανικές συνθήκες το σύστημα αναφοράς του χάρτη και το αντικειμενικό σύστημα αναφοράς θα έπρεπε να ταυτίζονται, όμως λόγω σφαλμάτων στην οδομετρία ή στους αισθητήρες αυτό δε γίνεται κι έτσι πρέπει να εκτιμήσουμε τη σχετική τους θέση.

Το δέντρο των μετασχηματισμών tf που προκύπτει κατά την εκτέλεση του προγράμματος είναι το παρακάτω:

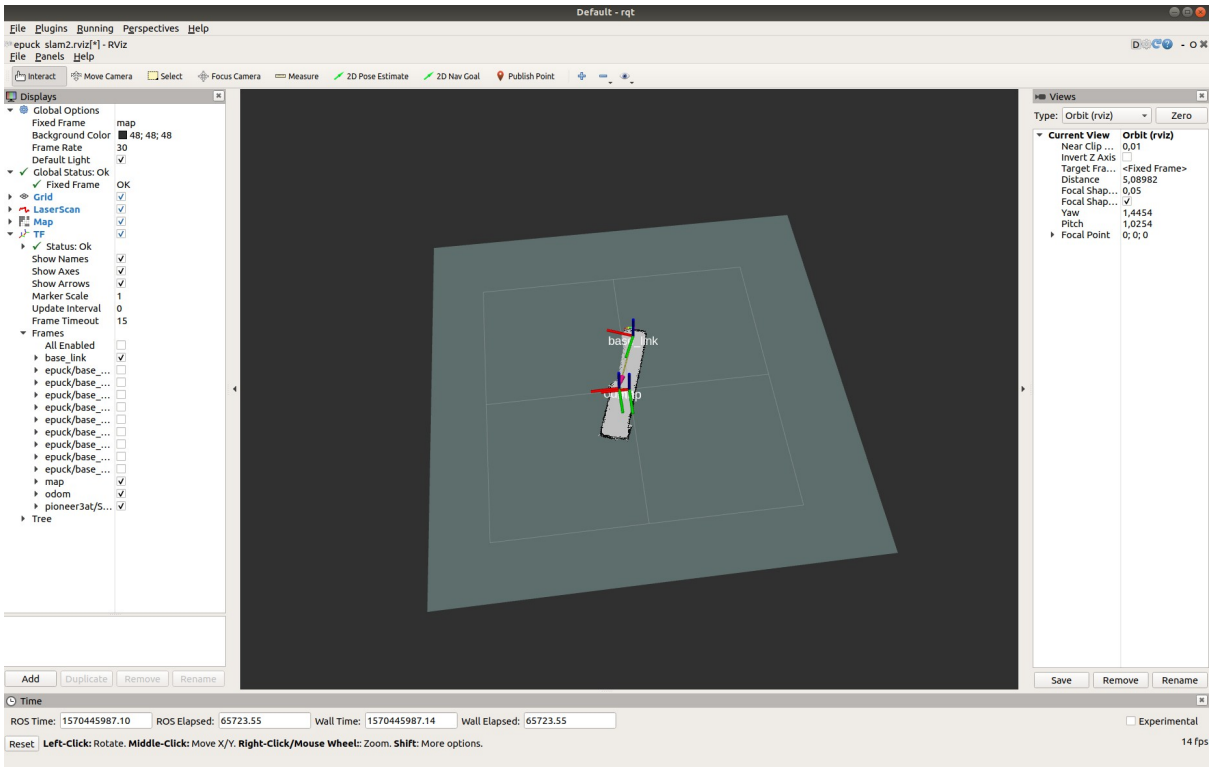


Εικόνα 7.9 Το δέντρο tf. Ο broadcaster είναι ο κόμβος είναι ο κόμβος που δημιουργεί τον μετασχηματισμό. Φαίνεται επίσης ο ρυθμός με τον οποίο υπολογίζεται ο κάθε μετασχηματισμός.

Στο κατώτερο επίπεδο βρίσκονται οι αισθητήρες προσέγγισης, για τον καθένα από τους οποίους υπολογίζεται ο μετασχηματισμός του από το κέντρο του ρομπότ. Δεν έχουν συμπεριληφθεί όλοι στην εικόνα, για να γίνεται πιο ευδιάκριτη.

7.4.6 rViz

Από το γράφο του ROS παραλείψαμε να αναφερθούμε στον κόμβο n_rqt_gui. Το rqt και πιο συγκεκριμένα το εργαλείο rViz του rqt είναι ένα εργαλείο οπτικοποίησης, το οποίο έχει τη δυνατότητα να παρέχει σε πραγματικό χρόνο το χάρτη που δημιουργείται από τον αλγόριθμο SLAM, καθώς και τη θέση του ρομπότ μέσα σε αυτόν. Το παράθυρο του rViz έχει την παρακάτω μορφή:



Εικόνα 7.10 Επισκόπηση του παραθύρου του rViz

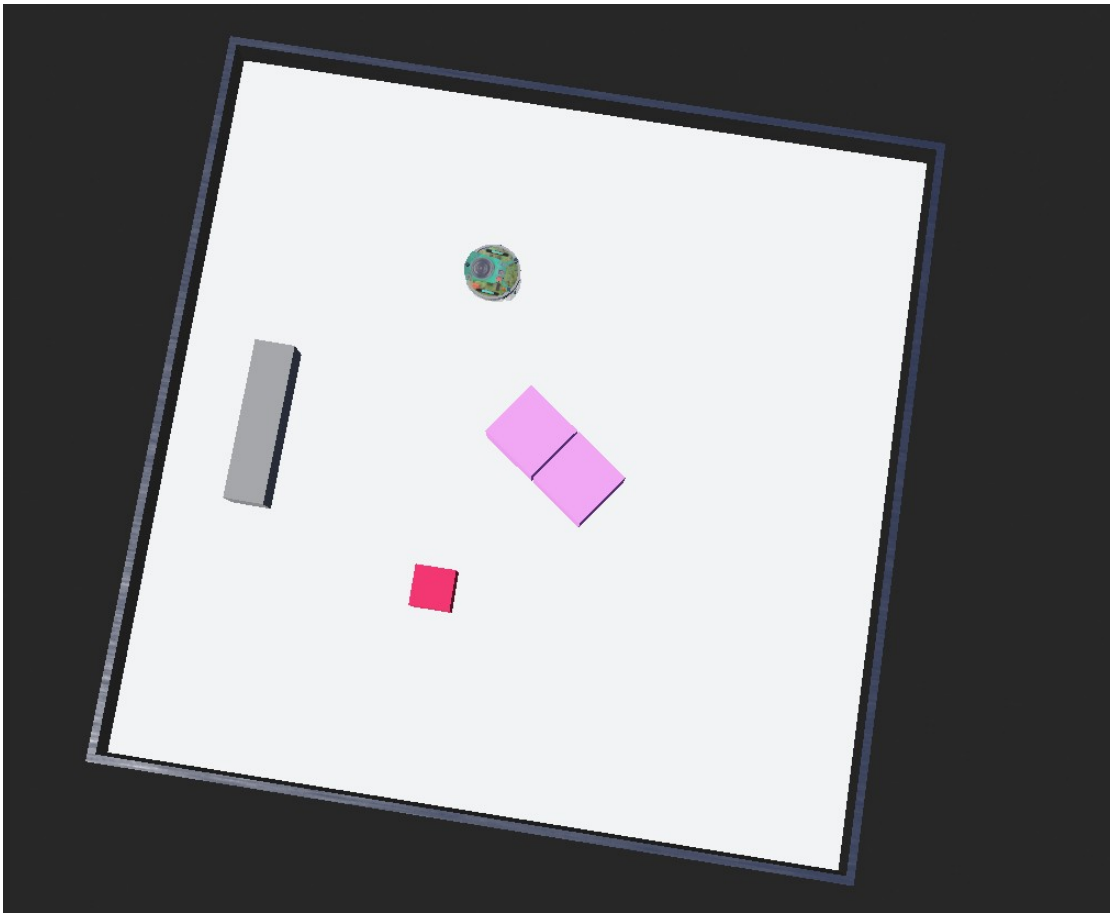
8

Αποτελέσματα

8.1 Περιβάλλοντα

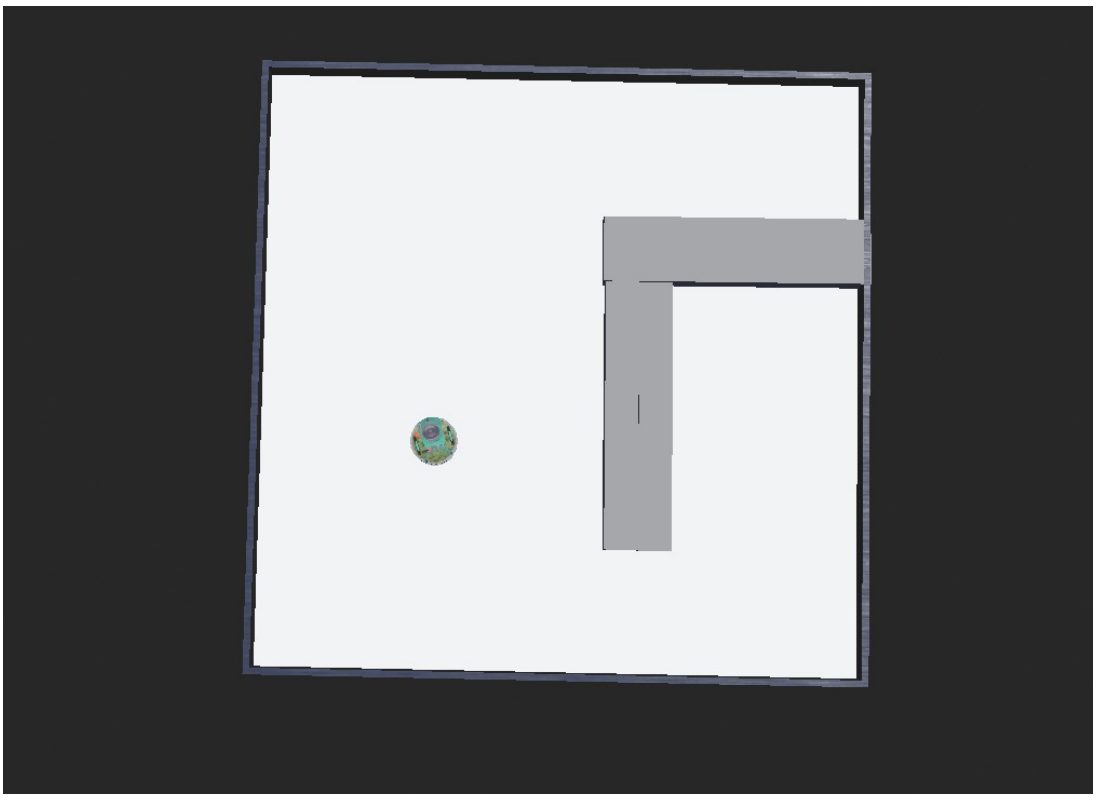
Για τη δοκιμή του προγράμματος δημιουργήθηκαν 3 περιβάλλοντα με εμπόδια και διαφορετικού σχεδιασμού, 2 σχετικά μικρού μεγέθους και ένα μεγαλύτερο. Τα περιβάλλοντα όπως δημιουργήθηκαν στο Webots παρουσιάζονται στις παρακάτω εικόνες:

Περιβάλλον 1



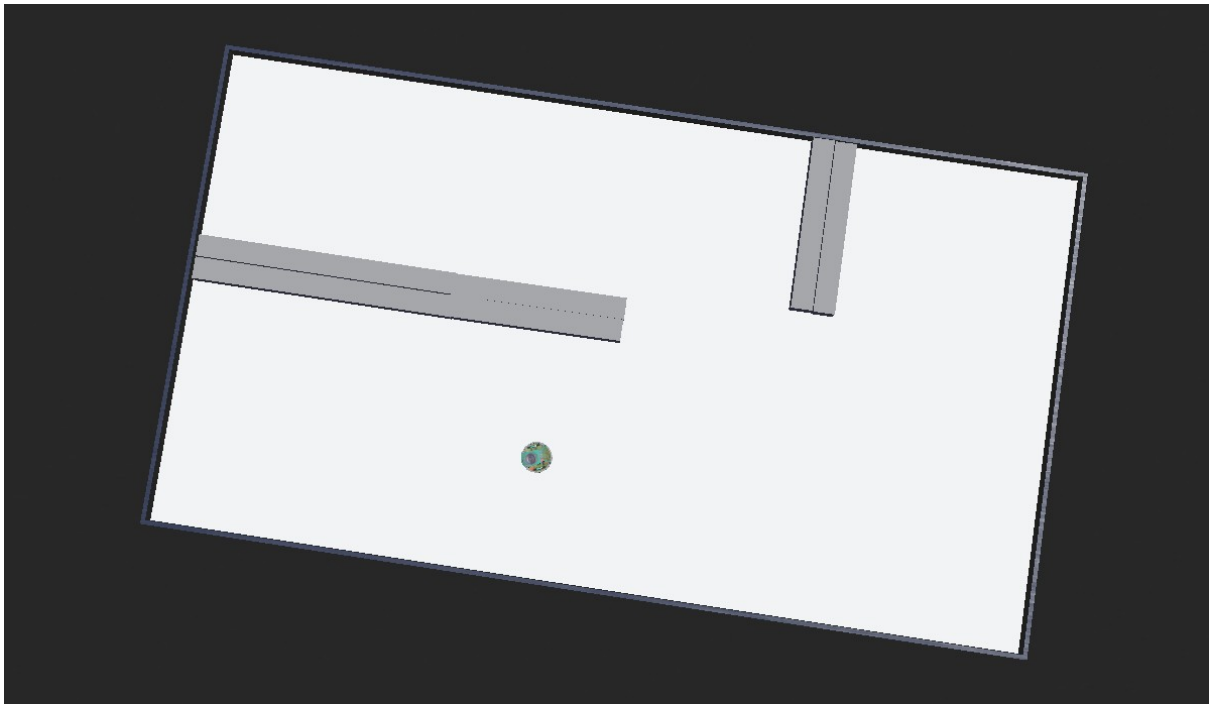
Εικόνα 8.1 Περιβάλλον 1 Στιγμιότυπο από Webots

Περιβάλλον 2



Εικόνα 8.2 Περιβάλλον 2 Στιγμιότυπο από Webots

Περιβάλλον 3

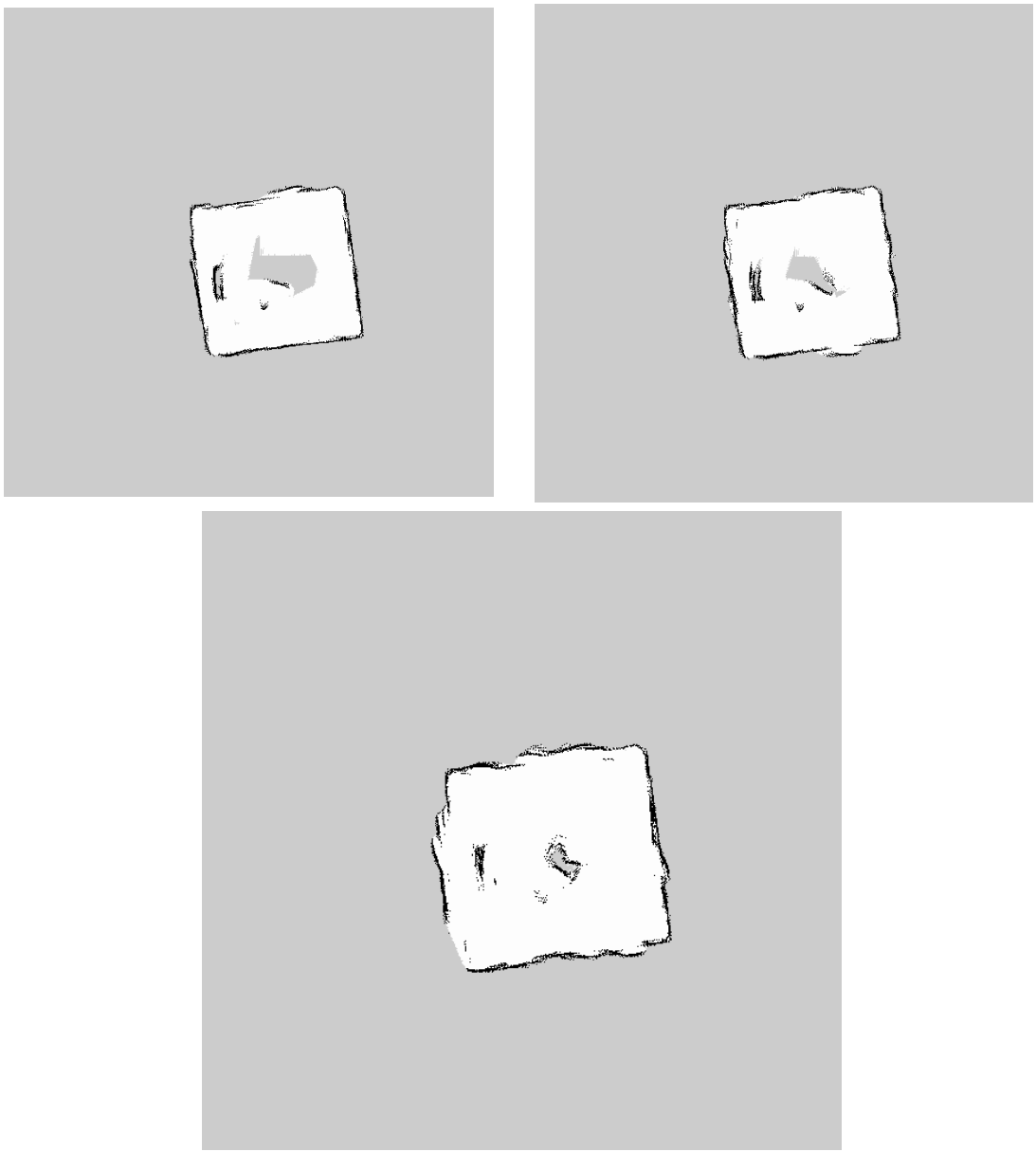


Εικόνα 8.3 Περιβάλλον 3 Στιγμιότυπο από Webots

8.2 Χαρτογράφηση

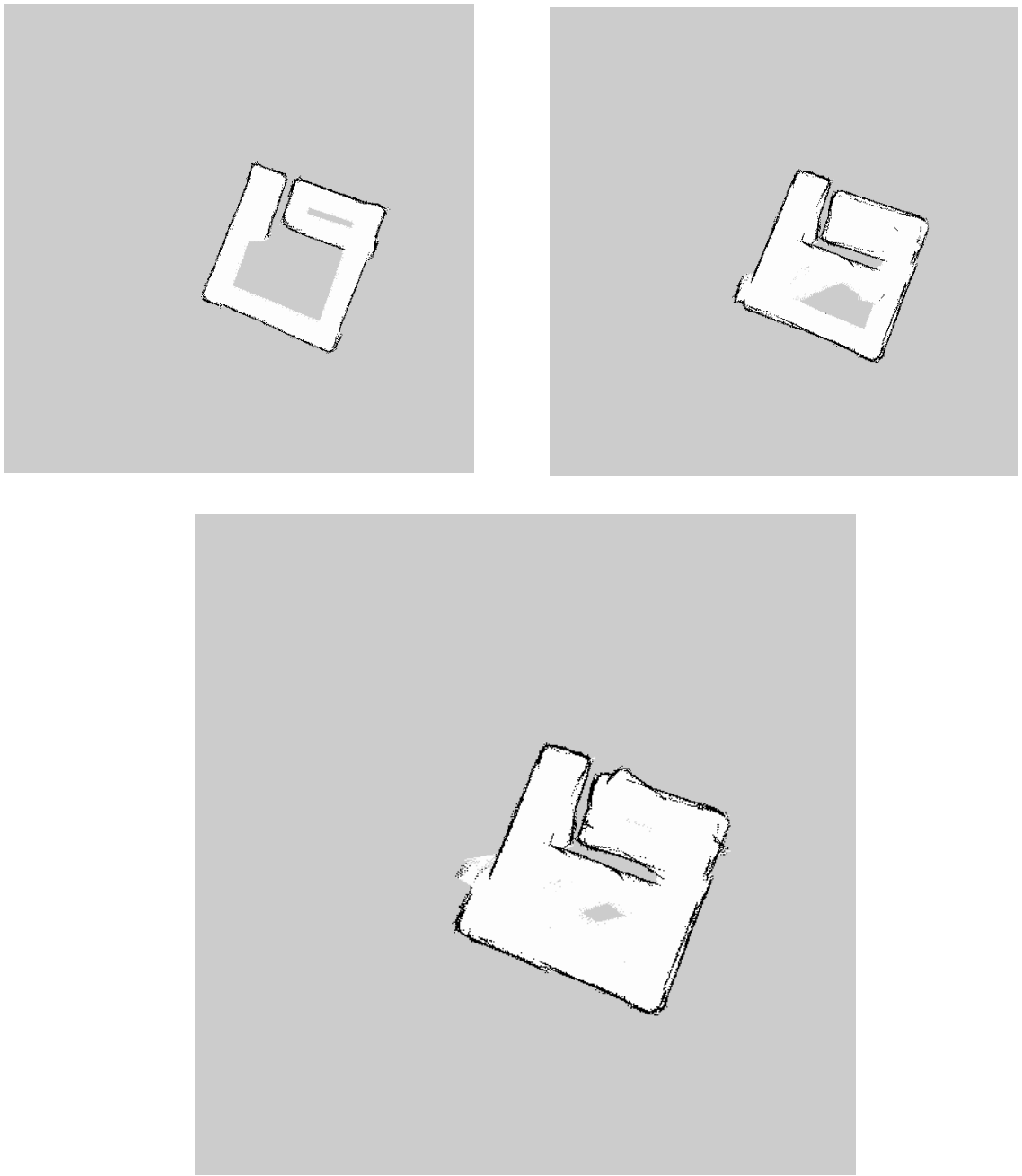
Με την εκτέλεση του προγράμματος και του εργαλείου map_server του ROS, αποθηκεύεται ο χάρτης του περιβάλλοντος που έχει δημιουργήσει το ρομπότ, τον οποίο μπορούμε άλλωστε να βλέπουμε σε πραγματικό χρόνο μέσω του rViz όπως προαναφέρθηκε. Οι χάρτες που λάβαμε δεν ήταν πάντα ιδανικοί για όλες τις επαναλήψεις της εκτέλεσης του προγράμματος. Ωστόσο, ε κάποιες προσπάθεις καταφέραμε να δημιουργήσουμε μια αρκετά καλή απεικόνιση για κάθε ένα από τα περιβάλλοντα που δημιουργήσαμε:

Περιβάλλον 1



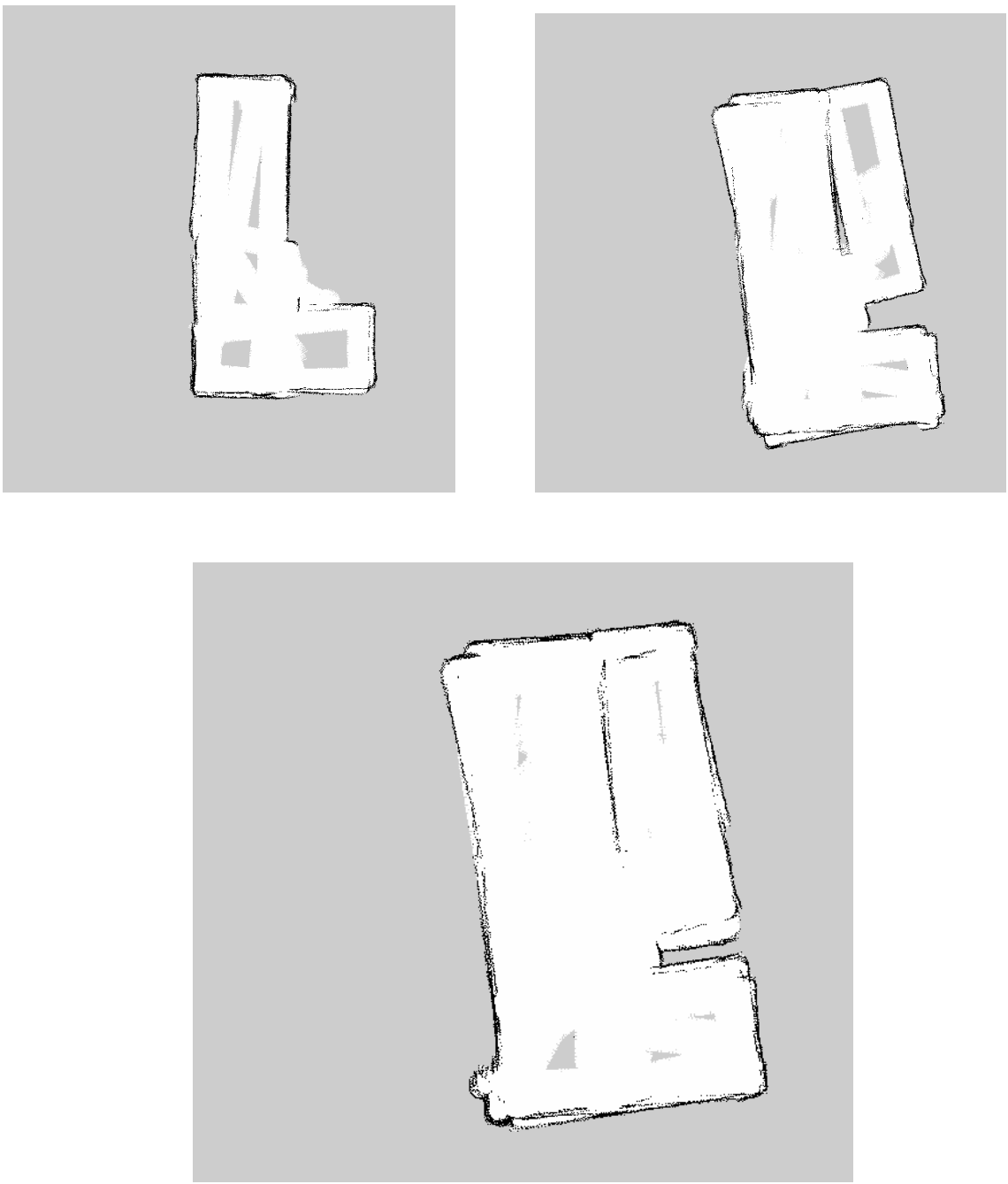
Εικόνα 8.4 Στιγμιότυπα του χάρτη που δημιουργήθηκε για το Περιβάλλον 1

Περιβάλλον 2



Εικόνα 8.5 Στιγμιότυπα του χάρτη που δημιουργήθηκε για το Περιβάλλον 2

Περιβάλλον 3



Εικόνα 8.6 Στιγμιότυπα του χάρτη που δημιουργήθηκε για το Περιβάλλον 3

Παρατηρούμε πως στους παραπάνω χάρτες αλλά και σε άλλους που προέκυψαν κατά την εκτέλεση του προγράμματος, υπάρχουν σφάλματα στην απεικόνιση του χώρου. Αυτά οφείλονται κυρίως σε εσφαλμένες συσχετίσεις δεδομένων, όπως για παράδειγμα μια λάθος εκτίμηση της θέσης ενός εμποδίου, οι οποίες μπορεί να έχουν ως αποτέλεσμα την πρόκληση σφαλμάτων και σε μελλοντικές συσχετίσεις δεδομένων. Ένας ακόμη σημαντικός παράγοντας είναι σφάλματα που συμβαίνουν στον υπολογισμό της οδομετρίας, που έχουν ως αποτέλεσμα λάθη στην εκτίμηση της θέσης του οχήματος. Επιπλέον, ο θόρυβος στους αισθητήρες απόστασης δεν τους καθιστά ικανούς να μας δώσουν ιδιαίτερα έμπιστα δεδομένα. Ο μικρός τους αριθμός και η μικρή τους εμβέλεια σημαίνουν ότι οι μετρήσεις που παίρνουμε δεν είναι επαρκείς για να μπορούμε να εκτιμήσουμε τη θέση των ορόσημων του χάρτη. Συγκεκριμένα δεν επαρκούν ώστε να μπορούμε στην ίδια παρατήρηση να έχουμε εκτίμηση για μεγάλο αριθμό ορόσημων, γεγονός που θα διευκόλυνε τον υπολογισμό των μεταξύ τους αποστάσεων. Αυτό έχει ως αποτέλεσμα οι παρατηρήσεις διαφορετικών σημείων του περιβάλλοντος να μοιάζουν μεταξύ τους (perceptual aliasing), οδηγώντας τον αλγόριθμο σε λάθος εκτίμηση της θέσης του ρομπότ.

9

Συμπεράσματα και Μελλοντική Έρευνα

9.1 Συμπεράσματα

Στόχος της παρούσας διπλωματικής εργασίας ήταν η χαρτογράφηση εσωτερικού χώρου σε περιβάλλον προσομοίωσης, με το εκπαιδευτικό κινούμενο ρομπότ E-puck, με χρήση του Robot Operating System (ROS). Διαπιστώθηκε ότι το ρομπότ είναι ικανό μέσα στο περιβάλλον προσομοίωσης:

- Να εκτελέσει κίνηση αποφεύγοντας εμπόδια που συναντά αλλά και να εκτελέσει εντολές κίνησης που του δίνονται από το χρήστη μέσω του πληκτρολογίου

- Να υπολογίσει σε πραγματικό χρόνο τη θέση του σχετικά με το σημείο εκκίνησής του
- Να χαρτογραφήσει το περιβάλλον του δημιουργώντας μια δισδιάστατη απεικόνιση του χώρου με τις θέσεις των αντικειμένων που εντοπίζει

Τα αποτελέσματα ήταν αρκετά ικανοποιητικά, κυρίως για τους δύο πρώτους από τους παραπάνω στόχους. Τα μεγαλύτερα προβλήματα εμφανίστηκαν στη χαρτογράφηση, και οφείλονται κυρίως στη μικρή εμβέλεια των αισθητήρων απόστασης του ρομπότ που οδηγούν σε λανθασμένες συσχετίσεις δεδομένων.

9.2 Μελλοντική Έρευνα

Μεγάλος αριθμός από αλλαγές ή προσθήκες μπορεί να πραγματοποιηθεί στην παρούσα διπλωματική εργασία, με σκοπό τη βελτίωση των αποτελεσμάτων και της συμπεριφοράς του συστήματος γενικότερα.

Αυτό που καθόρισε σε μεγάλο βαθμό τα αποτελέσματα είναι το μοντέλο του ρομπότ που χρησιμοποιήθηκε και το υλικό του. Η απόδοση του συστήματος μπορεί να αυξηθεί χρησιμοποιώντας ένα ρομποτικό όχημα με αισθητήρες μεγαλύτερης εμβέλειας και ακρίβειας, που να προσφέρουν μεγαλύτερο αριθμό παρατηρήσεων ανά βήμα. Ακόμη, μεγαλύτερη ακρίβεια στους encoders των τροχών μπορεί να παρέχει καλύτερο υπολογισμο της θέσης του ρομπότ.

Η μεγαλύτερη βέβαια προσθήκη που θα μπορούσε να πραγματοποιηθεί είναι η υλοποίηση σε πραγματικό ρομπότ. Η προσομοίωση μας δίνει μια ιδέα για τη συμπεριφορά του οχήματος σε ένα περιβάλλον, αλλά μόνο η εφαρμογή στην πραγματικότητα μπορεί να δώσει πραγματική εικόνα για το αν είναι εφικτός ο έλεγχος του ρομπότ και η χαρτογράφηση του περιβάλλοντός του.

Βιβλιογραφία

[1] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza, (2011), Introduction to Autonomous Mobile Robots, Second Edition

[2] Roland Siegwart, Margarita Chli, Nick Lawrance , (2019), Mobile Robot Kinematics,Autonomous Mobile Robots

https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/asl-dam/documents/lectures/autonomous_mobile_robots/spring-2019/3%20ETH%20Lecture%20%20Mobile%20Robots%20Kinematics%20add%20ons%202019%20RS.pdf

[3] Guy Campion, Georges Bastin, and Brigitte D' AndrCa-Novel, Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 12, NO. 1, FEBRUARY 1996

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.575.8055&rep=rep1&type=pdf

[4] Sven Böttcher, Principles of robot locomotion, Seminar 'Human robot interaction'

http://users.dimi.uniud.it/~antonio.dangelo/Robotica/2012/helper/leggedRobot/RobotLocomotion.pdf

[5] Matthew T. Mason, (2001), Mechanics of Robotic Manipulation

[6] Gerald Cook, (2011), Mobile Robots, Navigation, Control and Remote Sensing

[7] Charles Platt, and Fredrik Jansson, (2016), Encyclopedia of Electronic Components, Volume 3, Sensors

[8] Shuzhi Sam Ge, Frank L. Lewis, (2006), Autonomous Mobile Robots, Sensing, Control, Decision Making and Applications

https://doc.lagout.org/science/0_Computer Science/8_Electronics %26 Robotics/Autonomous Mobile Robots - Shuzhi Sam Ge%2C Frank L Lewis.pdf

[9] Johan Alexandersson, Olle Nordin, (2017), Implementation of SLAM algorithms in a small-scale vehicle using model-based development, Master of Science Thesis, Linköping University

https://liu.diva-portal.org/smash/get/diva2:1218791/FULLTEXT01.pdf

[10] Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard, Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling

http://people.ee.duke.edu/~lcarin/Lihan9.4.06b.pdf

[11] Cyrill Stachniss, (2012), Robot mapping EKF SLAM

http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam04-ekf-slam.pdf

[12] Παπαδημητροπούλου Β., Κυριακάκης Ε., (2017), Κατασκευή Συστήματος Στερεοσκοπικής Όρασης για Lego Mindstorms, Διπλωματική Εργασία, Πανεπιστήμιο Πατρών

https://nemertes.lis.upatras.gr/jspui/bitstream/10889/10344/3/Nemertes_Kyriakakis-Papadimitropoulou%28ele%29.pdf#%5B%7B%22num%22%3A526%2C%22gen%22%3A0%7D%2C%7B%22name%22%3A%22XYZ%22%7D%2C74%2C376%2C0%5D

[13] Μαγκαφάς Μ., Φιλτισένιος Ι., (2019), Προγραμματισμός και Έλεγχος Ρομποτικών Οχημάτων σε Ανταγωνιστικά Παιχνίδια, Διπλωματική Εργασία, Πανεπιστήμιο Πατρών

[14] Ζαφειράκης Γ., (2018), Οδομετρία Κινούμενου Ρομποτικού Συστήματος με Ψηφιακή Επεξεργασία Σημάτων Video, Διπλωματική Εργασία, Πανεπιστήμιο Πατρών

https://nemertes.lis.upatras.gr/jspui/bitstream/10889/10344/3/Nemertes_Kyriakakis-Papadimitropoulou(ele).pdf#%5B{"num"%3A526%2C"gen"%3A0}%2C{"name"%3A"XYZ"}%2C74%2C376%2C0%5D

Σύνδεσμοι

[1] http://wiki.ros.org/ROS/Tutorials

[2] http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData

[3] https://www.gctronic.com/doc/index.php/E-Puck

[4] http://wiki.ros.org/rviz_plugin_tutorials

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

[6] http://wiki.ros.org/tf/Tutorials

[7] http://docs.ros.org/melodic/api/sensor_msgs/html/index-msg.html

[8] https://cyberbotics.com/doc/guide/epuck

[9] http://www-inst.eecs.berkeley.edu/~ee290t/fa18/readings/Slam-for-dummies-mit-tutorial.pdf

[10] https://www.youtube.com/watch?v=xRvFJDuhx3Y

[11] https://www.youtube.com/watch?v=MIh-1EACaJc

[12] http://users.softlab.ntua.gr/~ktzaf/Courses/epuck-robotica2009.pdf

Παράρτημα

e_puck_slam.cpp

```
#include <sensor_msgs/Image.h>
#include <sensor_msgs/Imu.h>
#include <sensor_msgs/Range.h>
#include <sensor_msgs/NavSatFix.h>
#include <signal.h>
#include <std_msgs/String.h>
#include <tf/transform_broadcaster.h>
#include "ros/ros.h"
#include <webots_ros/set_float.h>
#include <webots_ros/set_int.h>
#include <std_msgs/Float64MultiArray.h>
#include <std_msgs/UInt16.h>
#include <std_msgs/UInt8MultiArray.h>
#include <cstdlib>
#include <geometry_msgs/Twist.h>
#include <std_msgs/Float64.h>
#include <std_msgs/UInt8MultiArray.h>
#include <sensor_msgs/LaserScan.h>
#include <nav_msgs/Odometry.h>
#include <sstream>
#include <webots_ros/get_int.h>
#include <webots_ros/get_float.h>
#include "webots_ros/Float64Stamped.h"
#include <webots_ros/Int32Stamped.h>


#define TIME_STEP 64
#define NMOTORS 2
#define NSENSORS 2
#define MAX_SPEED 6.4
#define OBSTACLE_THRESHOLD 0.1
#define DECREASE_FACTOR 0.9
#define BACK_SLOWDOWN 0.9

#define TRUE 1
#define FALSE 0
#define NO_SIDE -1
#define LEFT 0
#define RIGHT 1
#define WHITE 0
#define BLACK 1
#define SIMULATION 0  // for wb_robot_get_mode() function
#define REALITY 2     // for wb_robot_get_mode() function
```

```
#define WHEEL_DISTANCE 0.0566863     // Distance between wheels in meters (axis length); it's the same value as "WHEEL_SEPARATION" but expressed in meters.
#define WHEEL_DIAMETER 4.1         // cm.
#define WHEEL_CIRCUMFERENCE ((WHEEL_DIAMETER*M_PI)/100.0)     // Wheel circumference (meters).
#define MOT_STEP_DIST 0.02    // Distance for each motor step (meters); a complete turn is 1000 steps (0.000125 meters per step (m/steps)).
#define ROBOT_RADIUS 0.035 // meters.

#define DEBUG_ROS_PARAMS 1
#define DEBUG_UPDATE_SENSORS_DATA 0
#define DEBUG_ODOMETRY 0

std::string epuckname;
ros::Time currentTimeMap, lastTimeMap;

// 8 IR proximity sensors
#define NB_DIST_SENS 8
#define PS_RIGHT_00 0
#define PS_RIGHT_45 1
#define PS_RIGHT_90 2
#define PS_RIGHT_REAR 3
#define PS_LEFT_REAR 4
#define PS_LEFT_90 5
#define PS_LEFT_45 6
#define PS_LEFT_00 7


int psValue[NB_DIST_SENS] = {0, 0, 0, 0, 0, 0, 0, 0};
const int PS_OFFSET_SIMULATION[NB_DIST_SENS] = {300, 300, 300, 300, 300, 300, 300, 300};
const int PS_OFFSET_REALITY[NB_DIST_SENS] = {480, 170, 320, 500, 600, 680, 210, 640};

int proxData[8];
ros::Publisher proxPublisher[8];
sensor_msgs::Range proxMsg[8];
ros::Publisher laserPublisher;
sensor_msgs::LaserScan laserMsg;


// Encoders
double encValues[2]={0,0};
ros::Publisher odomPublisher;
nav_msgs::Odometry odomMsg;
double xPos, yPos, theta;
double deltaSteps, deltaTheta;
double leftStepsDiff = 0, rightStepsDiff = 0;
double leftStepsPrev = 0, rightStepsPrev = 0;
ros::Time currentTime, lastTime;


//------------------------------------------------------------------------------
//
//    ROS CALLBACKS
//
//------------------------------------------------------------------------------


static char modelList[10][100];
static int count = 0;
```

```
static int countDist = NB_DIST_SENS;
static int countEnc=2;
static int step=TIME_STEP;


static int controllerCount;
static std::vector<std::string> controllerList;

ros::ServiceClient setTimeStepClient;
webots_ros::set_int setTimeStepSrv;


void updateRosInfo() {

     static tf::TransformBroadcaster br;

     int i = 0;

     // Proximity handling.

     if(DEBUG_UPDATE_SENSORS_DATA)std::cout << "[" << epuckname << "] " << "prox: " << proxData[0] << "," << proxData[1] << "," << proxData[2] << "," << proxData[3] << "," << proxData[4] << "," << proxData[5] << "," << proxData[6] << "," << proxData[7] << std::endl;

     for(i=0; i<8; i++) {
          if(proxData[i] > 0) {
          proxMsg[i].range = 0.5/sqrt(proxData[i]);    // Transform the analog value to a distance value in meters (given from field tests).
          } else {
               proxMsg[i].range = proxMsg[i].max_range;
          }
          if(proxMsg[i].range > proxMsg[i].max_range) {
               proxMsg[i].range = proxMsg[i].max_range;
          }
          if(proxMsg[i].range < proxMsg[i].min_range) {
               proxMsg[i].range = proxMsg[i].min_range;
          }
          proxMsg[i].header.stamp = ros::Time::now();
          proxPublisher[i].publish(proxMsg[i]);
     }

     // e-puck proximity positions (cm), x pointing forward, y pointing left
     //           P7(3.5, 1.0)   P0(3.5, -1.0)
     //       P6(2.5, 2.5)           P1(2.5, -2.5)
     //   P5(0.0, 3.0)                   P2(0.0, -3.0)
     //       P4(-3.5, 2.0)          P3(-3.5, -2.0)
     //
     // e-puck proximity orentations (degrees)
     //           P7(10)   P0(350)
     //       P6(40)           P1(320)
     //   P5(90)                   P2(270)
     //       P4(160)          P3(200)
     std::stringstream parent;
     std::stringstream child;
     tf::Transform transform;
     tf::Quaternion q;

     transform.setOrigin( tf::Vector3(0.035, -0.010, 0.034) );
```

```
    q.setRPY(0, 0, 6.11);
    transform.setRotation(q);
    parent << epuckname << "/base_prox0";
    child << "base_link";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

    transform.setOrigin( tf::Vector3(0.025, -0.025, 0.034) );
    q.setRPY(0, 0, 5.59);
    transform.setRotation(q);
    parent.str("");
    parent << epuckname << "/base_prox1";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

    transform.setOrigin( tf::Vector3(0.000, -0.030, 0.034) );
    q.setRPY(0, 0, 4.71);
    transform.setRotation(q);
    parent.str("");
    parent << epuckname << "/base_prox2";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

    transform.setOrigin( tf::Vector3(-0.035, -0.020, 0.034) );
    q.setRPY(0, 0, 3.49);
    transform.setRotation(q);
    parent.str("");
    parent << epuckname << "/base_prox3";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

    transform.setOrigin( tf::Vector3(-0.035, 0.020, 0.034) );
    q.setRPY(0, 0, 2.8);
    transform.setRotation(q);
    parent.str("");
    parent << epuckname << "/base_prox4";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

    transform.setOrigin( tf::Vector3(0.000, 0.030, 0.034) );
    q.setRPY(0, 0, 1.57);
    transform.setRotation(q);
    parent.str("");
    parent << epuckname << "/base_prox5";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

    transform.setOrigin( tf::Vector3(0.025, 0.025, 0.034) );
    q.setRPY(0, 0, 0.70);
    transform.setRotation(q);
    parent.str("");
    parent << epuckname << "/base_prox6";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

    transform.setOrigin( tf::Vector3(0.035, 0.010, 0.034) );
    q.setRPY(0, 0, 0.17);
    transform.setRotation(q);
    parent.str("");
```

```
    parent << epuckname << "/base_prox7";
    br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));


    currentTimeMap = ros::Time::now();
    parent.str("");
    parent << epuckname << "/base_laser";
    //populate the LaserScan message
    laserMsg.header.stamp = ros::Time::now();
    laserMsg.header.frame_id = parent.str();
    laserMsg.angle_min = -M_PI/2.0;
    laserMsg.angle_max = M_PI/2.0;
    laserMsg.angle_increment = M_PI/18.0; // 10 degrees.
    //laserMsg.time_increment  =  (currentTimeMap-lastTimeMap).toSec()/180;  //0.003;  //(1  /
laser_frequency) / (num_readings);
    //laserMsg.scan_time = (currentTimeMap-lastTimeMap).toSec();
    // The laser is placed in the center of the robot, but the proximity sensors are placed
around the robot thus add "ROBOT_RADIUS" to get correct values.
    laserMsg.range_min = 0.005+ROBOT_RADIUS; // 0.5 cm + ROBOT_RADIUS.
    laserMsg.range_max = 0.05+ROBOT_RADIUS; // 5 cm + ROBOT_RADIUS.
    laserMsg.ranges.resize(19);
    laserMsg.intensities.resize(19);
    lastTimeMap = ros::Time::now();

    // We use the information from the 6 proximity sensors on the front side of the robot to
get 19 laser scan points. The interpolation used is the following:
    // -90 degrees: P2
    // -80 degrees: 4/5*P2 + 1/5*P1
    // -70 degrees: 3/5*P2 + 2/5*P1
    // -60 degrees: 2/5*P2 + 3/5*P1
    // -50 degrees: 1/5*P2 + 4/5*P1
    // -40 degrees: P1
    // -30 degrees: 2/3*P1 + 1/3*P0
    // -20 degrees: 1/3*P1 + 2/3*P0
    // -10 degrees: P0
    // 0 degrees: 1/2*P0 + 1/2*P7
    // 10 degrees: P7
    // 20 degrees: 1/3*P6 + 2/3*P7
    // 30 degrees: 2/3*P6 + 1/3*P7
    // 40 degrees: P6
    // 50 degrees: 1/5*P5 + 4/5*P6
    // 60 degrees: 2/5*P5 + 3/5*P6
    // 70 degrees: 3/5*P5 + 2/5*P6
    // 80 degrees: 4/5*P5 + 1/5*P6
    // 90 degrees: P5

    float tempProx;
    tempProx = proxData[2];
    if(tempProx > 0) {
        laserMsg.ranges[0]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
        laserMsg.intensities[0] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
        laserMsg.ranges[0] = laserMsg.range_max;
        laserMsg.intensities[0] = 0;
    }
```

```
     tempProx = proxData[2]*4/5 + proxData[1]*1/5;
     if(tempProx > 0) {
          laserMsg.ranges[1]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
          laserMsg.intensities[1] = tempProx;
     } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
          laserMsg.ranges[1] = laserMsg.range_max;
          laserMsg.intensities[1] = 0;
     }

     tempProx = proxData[2]*3/5 + proxData[1]*2/5;
     if(tempProx > 0) {
          laserMsg.ranges[2]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
          laserMsg.intensities[2] = tempProx;
     } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
          laserMsg.ranges[2] = laserMsg.range_max;
          laserMsg.intensities[2] = 0;
     }

     tempProx = proxData[2]*2/5 + proxData[1]*3/5;
     if(tempProx > 0) {
          laserMsg.ranges[3]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
          laserMsg.intensities[3] = tempProx;
     } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
          laserMsg.ranges[3] = laserMsg.range_max;
          laserMsg.intensities[3] = 0;
     }

     tempProx = proxData[2]*1/5 + proxData[1]*4/5;
     if(tempProx > 0) {
          laserMsg.ranges[4]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
          laserMsg.intensities[4] = tempProx;
     } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
          laserMsg.ranges[4] = laserMsg.range_max;
          laserMsg.intensities[4] = 0;
     }

     tempProx = proxData[1];
     if(tempProx > 0) {
          laserMsg.ranges[5]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
          laserMsg.intensities[5] = tempProx;
     } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
          laserMsg.ranges[5] = laserMsg.range_max;
          laserMsg.intensities[5] = 0;
     }

     tempProx = proxData[1]*2/3 + proxData[0]*1/3;
     if(tempProx > 0) {
          laserMsg.ranges[6]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
          laserMsg.intensities[6] = tempProx;
```

```
    } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
        laserMsg.ranges[6] = laserMsg.range_max;
        laserMsg.intensities[6] = 0;
    }

    tempProx = proxData[1]*1/3 + proxData[0]*2/3;
    if(tempProx > 0) {
        laserMsg.ranges[7]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
        laserMsg.intensities[7] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
        laserMsg.ranges[7] = laserMsg.range_max;
        laserMsg.intensities[7] = 0;
    }

    tempProx = proxData[0];
    if(tempProx > 0) {
        laserMsg.ranges[8]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
        laserMsg.intensities[8] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
        laserMsg.ranges[8] = laserMsg.range_max;
        laserMsg.intensities[8] = 0;
    }

    tempProx = (proxData[0]+proxData[7])>>1;
    if(tempProx > 0) {
        laserMsg.ranges[9]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
        laserMsg.intensities[9] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
        laserMsg.ranges[9] = laserMsg.range_max;
        laserMsg.intensities[9] = 0;
    }

    tempProx = proxData[7];
    if(tempProx > 0) {
        laserMsg.ranges[10]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
        laserMsg.intensities[10] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
        laserMsg.ranges[10] = laserMsg.range_max;
        laserMsg.intensities[10] = 0;
    }

    tempProx = proxData[7]*2/3 + proxData[6]*1/3;
    if(tempProx > 0) {
        laserMsg.ranges[11]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  //  Transform  the  analog
value to a distance value in meters (given from field tests).
        laserMsg.intensities[11] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
        laserMsg.ranges[11] = laserMsg.range_max;
        laserMsg.intensities[11] = 0;
    }
```

```
    tempProx = proxData[7]*1/3 + proxData[6]*2/3;
    if(tempProx > 0) {
        laserMsg.ranges[12]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  // Transform the analog value to a distance value in meters (given from field tests).
        laserMsg.intensities[12] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there is no obstacles.
        laserMsg.ranges[12] = laserMsg.range_max;
        laserMsg.intensities[12] = 0;
    }

    tempProx = proxData[6];
    if(tempProx > 0) {
        laserMsg.ranges[13]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  // Transform the analog value to a distance value in meters (given from field tests).
        laserMsg.intensities[13] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there is no obstacles.
        laserMsg.ranges[13] = laserMsg.range_max;
        laserMsg.intensities[13] = 0;
    }

    tempProx = proxData[6]*4/5 + proxData[5]*1/5;
    if(tempProx > 0) {
        laserMsg.ranges[14]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  // Transform the analog value to a distance value in meters (given from field tests).
        laserMsg.intensities[14] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there is no obstacles.
        laserMsg.ranges[14] = laserMsg.range_max;
        laserMsg.intensities[14] = 0;
    }

    tempProx = proxData[6]*3/5 + proxData[5]*2/5;
    if(tempProx > 0) {
        laserMsg.ranges[15]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  // Transform the analog value to a distance value in meters (given from field tests).
        laserMsg.intensities[15] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there is no obstacles.
        laserMsg.ranges[15] = laserMsg.range_max;
        laserMsg.intensities[15] = 0;
    }

    tempProx = proxData[6]*2/5 + proxData[5]*3/5;
    if(tempProx > 0) {
        laserMsg.ranges[16]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  // Transform the analog value to a distance value in meters (given from field tests).
        laserMsg.intensities[16] = tempProx;
    } else { // Sometimes the values could be negative due to the calibration, it means there is no obstacles.
        laserMsg.ranges[16] = laserMsg.range_max;
        laserMsg.intensities[16] = 0;
    }

    tempProx = proxData[6]*1/5 + proxData[5]*4/5;
    if(tempProx > 0) {
        laserMsg.ranges[17]  =  (0.5/sqrt(tempProx))+ROBOT_RADIUS;  // Transform the analog value to a distance value in meters (given from field tests).
```

```
            laserMsg.intensities[17] = tempProx;
        } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
            laserMsg.ranges[17] = laserMsg.range_max;
            laserMsg.intensities[17] = 0;
        }

        tempProx = proxData[5];
        if(tempProx > 0) {
            laserMsg.ranges[18] = (0.5/sqrt(tempProx))+ROBOT_RADIUS; // Transform the analog
value to a distance value in meters (given from field tests).
            laserMsg.intensities[18] = tempProx;
        } else { // Sometimes the values could be negative due to the calibration, it means there
is no obstacles.
            laserMsg.ranges[18] = laserMsg.range_max;
            laserMsg.intensities[18] = 0;
        }

        for(i=0; i<19; i++) {
            if(laserMsg.ranges[i] > laserMsg.range_max) {
                laserMsg.ranges[i] = laserMsg.range_max;
            }
            if(laserMsg.ranges[i] < laserMsg.range_min) {
                laserMsg.ranges[i] = laserMsg.range_min;
            }
        }

        transform.setOrigin( tf::Vector3(0.0, 0.0, 0.034) );
        q.setRPY(0, 0, 0);
        transform.setRotation(q);
        parent.str("");
        child.str("");
        parent << epuckname << "/base_laser";
        child << "base_link";
        br.sendTransform(tf::StampedTransform(transform,      ros::Time::now(),      child.str(),
parent.str()));

        laserPublisher.publish(laserMsg);


        // Odometry handling.
        if(DEBUG_ODOMETRY) std::cout << "[" << epuckname << "] " << "enc left: " << encValues[0]
<< ", enc right: " << encValues[1] << std::endl;

        // Compute odometry.
        leftStepsDiff = encValues[0]*MOT_STEP_DIST - leftStepsPrev; // Expressed in meters.
        rightStepsDiff = encValues[1]*MOT_STEP_DIST - rightStepsPrev;   // Expressed in meters.
        if(DEBUG_ODOMETRY)std::cout << "[" << epuckname << "] " << "left, right steps diff: " <<
leftStepsDiff << ", " << rightStepsDiff << std::endl;

        deltaTheta = (rightStepsDiff - leftStepsDiff)/WHEEL_DISTANCE;   // Expressed in radiant.
        deltaSteps = (rightStepsDiff + leftStepsDiff)/2;        // Expressed in meters.
        if(DEBUG_ODOMETRY)std::cout << "[" << epuckname << "] " << "delta theta, steps: " <<
deltaTheta << ", " << deltaSteps << std::endl;

        xPos += deltaSteps*cos(theta + deltaTheta/2);   // Expressed in meters.
        yPos += deltaSteps*sin(theta + deltaTheta/2);   // Expressed in meters.
        theta += deltaTheta;    // Expressed in radiant.
```

```
    if(DEBUG_ODOMETRY)std::cout << "[" << epuckname << "] " << "x, y, theta: " << xPos << ",
" << yPos << ", " << theta << std::endl;

    leftStepsPrev = encValues[0]*MOT_STEP_DIST;     // Expressed in meters.
    rightStepsPrev = encValues[1]*MOT_STEP_DIST;    // Expressed in meters.

    // Publish the odometry message over ROS.
    odomMsg.header.stamp = ros::Time::now();
    odomMsg.header.frame_id = "odom";
    std::stringstream ss;

    ss << "base_link";
    odomMsg.child_frame_id = ss.str();
    odomMsg.pose.pose.position.x = xPos;
    odomMsg.pose.pose.position.y = yPos;
    odomMsg.pose.pose.position.z = 0;
    // Since all odometry is 6DOF we'll need a quaternion created from yaw.
    geometry_msgs::Quaternion odomQuat = tf::createQuaternionMsgFromYaw(theta);
    odomMsg.pose.pose.orientation = odomQuat;
    currentTime = ros::Time::now();
    odomMsg.twist.twist.linear.x  =  deltaSteps  /  ((currentTime-lastTime).toSec());      //
"deltaSteps" is the linear distance covered in meters from the last update (delta distance);

            // the time from the last update is measured in seconds thus to get m/s we
multiply them.
    odomMsg.twist.twist.angular.z  =  deltaTheta  /  ((currentTime-lastTime).toSec());    //
"deltaTheta" is the angular distance covered in radiant from the last update (delta angle);

            // the time from the last update is measured in seconds thus to get rad/s we
multiply them.
    if(DEBUG_ODOMETRY)std::cout  <<  "["  <<  epuckname  <<  "]  "  <<  "time  elapsed  =  "  <<
(currentTime-lastTime).toSec() << " seconds" << std::endl;
    lastTime = ros::Time::now();

    odomPublisher.publish(odomMsg);


    // Publish the transform over tf.
    geometry_msgs::TransformStamped odomTrans;
    odomTrans.header.stamp = odomMsg.header.stamp;
    odomTrans.header.frame_id = odomMsg.header.frame_id;
    odomTrans.child_frame_id = odomMsg.child_frame_id;
    odomTrans.transform.translation.x = xPos;
    odomTrans.transform.translation.y = yPos;
    odomTrans.transform.translation.z = 0.0;
    odomTrans.transform.rotation = odomQuat;
    br.sendTransform(odomTrans);

}

void encodersCallback(const webots_ros::Float64Stamped::ConstPtr& value) {
  if (countEnc < NSENSORS) {
    encValues[countEnc] = value->data - 0.40192;
  }
  countEnc++;
}

void psCallback(const sensor_msgs::Range::ConstPtr& value)
{
```

```
  if (countDist < NB_DIST_SENS) {
          proxData[countDist] = value->range;
     psValue[countDist]=((value->range - PS_OFFSET_SIMULATION[countDist]) < 0) ? 0 : (value-
>range - PS_OFFSET_SIMULATION[countDist]);
  }
  countDist++;
}

void modelNameCallback(const std_msgs::String::ConstPtr &name) {
  count++;
  strcpy(modelList[count], name->data.c_str());
  ROS_INFO("Model #%d: %s.", count, name->data.c_str());
}

void quit(int sig) {
  setTimeStepSrv.request.value = 0;
  setTimeStepClient.call(setTimeStepSrv);
  ROS_INFO("User stopped the 'e_puck_map_test' node.");
  ros::shutdown();
  exit(0);
}


//////////////////////////////////////////
// OAM - Obstacle Avoidance Module
//
// The OAM routine first detects obstacles in front of the robot, then records
// their side in "oam_side" and avoid the detected obstacle by
// turning away according to very simple weighted connections between
// proximity sensors and motors. "oam_active" becomes active when as soon as
// an object is detected and "oam_reset" inactivates the module and set
// "oam_side" to NO_SIDE. Output speeds are in oam_speed[LEFT] and oam_speed[RIGHT].

int oam_active, oam_reset;
int oam_speed[2];
int oam_side = NO_SIDE;

//#define OAM_OBST_THRESHOLD 100
#define OAM_OBST_THRESHOLD 5   //40

//change speed arxikh 150
//#define OAM_FORWARD_SPEED 150
#define OAM_FORWARD_SPEED 20
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////
// #define OAM_K_PS_90 0.2
// #define OAM_K_PS_45 0.9
// #define OAM_K_PS_00 1.2

//#define OAM_K_PS_90 0.1*0.1
#define OAM_K_PS_90 0.2*0.1
#define OAM_K_PS_45 0.9*0.1
#define OAM_K_PS_00 1.2*0.1

#define OAM_K_MAX_DELTAS 600
```

```
////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
void wallFollowingModule(void) {
  int max_ds_value, i;
  int Activation[] = {0, 0};

  // Module RESET
  if (oam_reset) {
    oam_active = FALSE;
    oam_side = NO_SIDE;
  }
  oam_reset = 0;

  // Determine the presence and the side of an obstacle
  max_ds_value = 0;
  for (i = PS_RIGHT_00; i <= PS_RIGHT_45; i++) {
    if (max_ds_value < psValue[i])
      max_ds_value = psValue[i];
    Activation[RIGHT] += psValue[i];
  }
  for (i = PS_LEFT_45; i <= PS_LEFT_00; i++) {
    if (max_ds_value < psValue[i])
      max_ds_value = psValue[i];
    Activation[LEFT] += psValue[i];
  }
  if (max_ds_value > OAM_OBST_THRESHOLD)
    oam_active = TRUE;

   if (oam_active && oam_side == NO_SIDE) {   // check for side of obstacle only when not
already detected
    if (Activation[RIGHT] > Activation[LEFT])
      oam_side = RIGHT;
    else
      oam_side = LEFT;
  }

  // Forward speed
  oam_speed[LEFT] = OAM_FORWARD_SPEED;
  oam_speed[RIGHT] = OAM_FORWARD_SPEED;

  // Go away from obstacle
  if (oam_active) {

    int DeltaS = 0;
    // The rotation of the robot is determined by the location and the side of the obstacle
    if (oam_side == LEFT) {
      //(((psValue[PS_LEFT_90]-PS_OFFSET)<0)?0:(psValue[PS_LEFT_90]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_90 * psValue[PS_LEFT_90]);
      //(((psValue[PS_LEFT_45]-PS_OFFSET)<0)?0:(psValue[PS_LEFT_45]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_45 * psValue[PS_LEFT_45]);
      //(((psValue[PS_LEFT_00]-PS_OFFSET)<0)?0:(psValue[PS_LEFT_00]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_00 * psValue[PS_LEFT_00]);
    } else {  // oam_side == RIGHT
      //(((psValue[PS_RIGHT_90]-PS_OFFSET)<0)?0:(psValue[PS_RIGHT_90]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_90 * psValue[PS_RIGHT_90]);
      //(((psValue[PS_RIGHT_45]-PS_OFFSET)<0)?0:(psValue[PS_RIGHT_45]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_45 * psValue[PS_RIGHT_45]);
      //(((psValue[PS_RIGHT_00]-PS_OFFSET)<0)?0:(psValue[PS_RIGHT_00]-PS_OFFSET)));
```

```
      DeltaS += (int)(OAM_K_PS_00 * psValue[PS_RIGHT_00]);
    }
    if (DeltaS > OAM_K_MAX_DELTAS)
      DeltaS = OAM_K_MAX_DELTAS;
    if (DeltaS < -OAM_K_MAX_DELTAS)
      DeltaS = -OAM_K_MAX_DELTAS;

    oam_speed[LEFT] -= DeltaS;
    oam_speed[RIGHT] += DeltaS;
  }
}


////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////

void ObstacleAvoidanceModule2(void) {
  int max_ds_value, i;
  int Activation[] = {0, 0};

  // Module RESET
  if (oam_reset) {
    oam_active = FALSE;
    oam_side = NO_SIDE;
  }
  oam_reset = 0;

  // Determine the presence and the side of an obstacle
  max_ds_value = 0;
   for (i = PS_RIGHT_00; i <= PS_RIGHT_REAR; i++) {
    if (max_ds_value < psValue[i])
      max_ds_value = psValue[i];
    Activation[RIGHT] += psValue[i];
  }
  for (i = PS_LEFT_REAR; i <= PS_LEFT_00; i++) {
    if (max_ds_value < psValue[i])
      max_ds_value = psValue[i];
    Activation[LEFT] += psValue[i];
  }
  if (max_ds_value > OAM_OBST_THRESHOLD)
    oam_active = TRUE;

   if (oam_active && oam_side == NO_SIDE) {   // check for side of obstacle only when not
already detected
    if (Activation[RIGHT] > Activation[LEFT])
      oam_side = RIGHT;
    else
      oam_side = LEFT;
  }

  // Forward speed
  oam_speed[LEFT] = OAM_FORWARD_SPEED;
  oam_speed[RIGHT] = OAM_FORWARD_SPEED;

  // Go away from obstacle
  if (oam_active) {

    int DeltaS = 0;
```

```
    // The rotation of the robot is determined by the location and the side of the obstacle
    if (oam_side == LEFT) {
      //(((psValue[PS_LEFT_90]-PS_OFFSET)<0)?0:(psValue[PS_LEFT_90]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_90 * psValue[PS_LEFT_90]);
      //(((psValue[PS_LEFT_45]-PS_OFFSET)<0)?0:(psValue[PS_LEFT_45]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_45 * psValue[PS_LEFT_45]);
      //(((psValue[PS_LEFT_00]-PS_OFFSET)<0)?0:(psValue[PS_LEFT_00]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_00 * psValue[PS_LEFT_00]);
    } else {  // oam_side == RIGHT
      //(((psValue[PS_RIGHT_90]-PS_OFFSET)<0)?0:(psValue[PS_RIGHT_90]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_90 * psValue[PS_RIGHT_90]);
      //(((psValue[PS_RIGHT_45]-PS_OFFSET)<0)?0:(psValue[PS_RIGHT_45]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_45 * psValue[PS_RIGHT_45]);
      //(((psValue[PS_RIGHT_00]-PS_OFFSET)<0)?0:(psValue[PS_RIGHT_00]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_00 * psValue[PS_RIGHT_00]);
    }
    if (DeltaS > OAM_K_MAX_DELTAS)
      DeltaS = OAM_K_MAX_DELTAS;
    if (DeltaS < -OAM_K_MAX_DELTAS)
      DeltaS = -OAM_K_MAX_DELTAS;

    oam_speed[LEFT] -= DeltaS;
    oam_speed[RIGHT] += DeltaS;
  }
}
```

```
int main(int argc, char **argv) {
  int i;
  int oamOfmSpeed[2];
  double speed[2];

  int stepMax = 900000 / TIME_STEP;
  int nStep = 0;

  double init_xpos, init_ypos, init_theta;

  ROS_INFO("This is just a test");

  ros::init(argc, argv, "e_puck_slam", ros::init_options::AnonymousName);
  ros::NodeHandle np("~"); // Private.
  ros::NodeHandle n; // Public.

  signal(SIGINT, quit);

  np.param<std::string>("epuck_name", epuckname, "epuck");
  np.param("xpos", init_xpos, 0.0);
  np.param("ypos", init_ypos, 0.0);
  np.param("theta", init_theta, 0.0);

  if(DEBUG_ROS_PARAMS) {
        std::cout << "[" << epuckname << "] " << "epuck name: " << epuckname << std::endl;
          std::cout << "[" << epuckname << "] " << "init pose: " << init_xpos << ", " <<
init_ypos << ", " << init_theta << std::endl;
```

```
  }


  std::string modelName;
  geometry_msgs::Twist command;


  // get the name of the robot
  ros::Subscriber nameSub = n.subscribe("model_name", 100, modelNameCallback);
  while (count == 0 || count < nameSub.getNumPublishers()) {
    ros::spinOnce();
    ros::spinOnce();
    ros::spinOnce();
  }
  ros::spinOnce();
  if (count == 1)
    modelName = modelList[1];
  else {
    int wantedModel = 0;
    std::cout << "Choose the # of the model you want to use:\n";
    std::cin >> wantedModel;
    if (1 <= wantedModel && wantedModel <= count)
      modelName = modelList[wantedModel];
    else {
      ROS_ERROR("Invalid choice.");
      return 1;
    }
  }
  nameSub.shutdown();
  count = 0;


  // send robot time step to webots
  setTimeStepClient = n.serviceClient<webots_ros::set_int>(modelName + "/robot/time_step");
  setTimeStepSrv.request.value = step;

  std::vector<ros::ServiceClient> enableDistSensorClient;
  webots_ros::set_int enableDistSensorSrv;

  std::vector<ros::ServiceClient> enableEncoderClient;
  webots_ros::set_int enableEncoderSrv;


  // enable encoders
  char sensorsList[2][20] = {"left_wheel_sensor","right_wheel_sensor"};
  char deviceName[20];
  ros::Subscriber SubEncoder[NSENSORS];
  for (i = 0; i < NSENSORS; i++) {
     sprintf(deviceName, "%s", sensorsList[i]);
          enableEncoderClient.push_back(n.serviceClient<webots_ros::set_int>(modelName+'/'   +
deviceName + "/enable"));
    enableEncoderSrv.request.value = step;
     if (enableEncoderClient[i].call(enableEncoderSrv) && enableEncoderSrv.response.success ==
1)  {
      ROS_INFO("Device %s enabled", deviceName);
       std::ostringstream s;
       s << step;
```

```
          SubEncoder[i]   =   n.subscribe(modelName   +   '/'   +   deviceName   +   "/value",   1,
encodersCallback);
       while (SubEncoder[i].getNumPublishers() == 0);
    } else {
       if (enableEncoderSrv.response.success == 2)
         ROS_ERROR("Sampling period is not valid");
       ROS_ERROR("Failed to enable encoders");
       return 1;
    }
  }


  // enable ir proximity sensors
  ros::Subscriber SubDistIr[NB_DIST_SENS];
  for (i = 0; i < NB_DIST_SENS; i++) {
    sprintf(deviceName, "ps%d", i);
     enableDistSensorClient.push_back(n.serviceClient<webots_ros::set_int>(modelName  +  '/'  +
deviceName + "/enable"));
    enableDistSensorSrv.request.value = step;
                          if       (enableDistSensorClient[i].call(enableDistSensorSrv)       &&
enableDistSensorSrv.response.success) {
      ROS_INFO("Device %s enabled.", deviceName);
      std::ostringstream s;
      s << step;
      SubDistIr[i] = n.subscribe(modelName + '/' + deviceName + "/value", 1, psCallback);
      while (SubDistIr[i].getNumPublishers() == 0) {
      }
    } else {
      if (!enableDistSensorSrv.response.success)
        ROS_ERROR("Sampling period is not valid.");
      ROS_ERROR("Failed to enable %s.", deviceName);
      return 1;
    }
  }


  for(i=0; i<8; i++) {
     std::stringstream ss;
     ss.str("");
     ss << "proximity" << i;
     proxPublisher[i] = n.advertise<sensor_msgs::Range>(ss.str(), 10);
     //proxMsg[i] = new sensor_msgs::Range();
     proxMsg[i].radiation_type = sensor_msgs::Range::INFRARED;
     ss.str("");
     ss << epuckname << "/base_prox" << i;
     proxMsg[i].header.frame_id =  ss.str();
     proxMsg[i].field_of_view = 0.26;     // About 15 degrees...to be checked!
     proxMsg[i].min_range = 0.005;        // 0.5 cm.
     proxMsg[i].max_range = 0.05;         // 5 cm.
  }

  laserPublisher = n.advertise<sensor_msgs::LaserScan>("scan", 10);


  webots_ros::set_float wheelSrv;
  webots_ros::set_float leftWheelSrv;
  webots_ros::set_float rightWheelSrv;
  ////////////////////////////////////////////////////////////////////////////
```

```
  wheelSrv.request.value = INFINITY;
  webots_ros::set_float wheelVelSrv;
  wheelVelSrv.request.value = 0;
  ros::ServiceClient leftWheelPositionClient =
    n.serviceClient<webots_ros::set_float>(modelName + "/left_wheel_motor/set_position");
  leftWheelPositionClient.call(wheelSrv);
  ros::ServiceClient rightWheelPositionClient =
    n.serviceClient<webots_ros::set_float>(modelName + "/right_wheel_motor/set_position");
  rightWheelPositionClient.call(wheelSrv);
  ros::ServiceClient leftWheelVelocityClient =
    n.serviceClient<webots_ros::set_float>(modelName + "/left_wheel_motor/set_velocity");
  leftWheelVelocityClient.call(leftWheelSrv);
  ros::ServiceClient rightWheelVelocityClient =
    n.serviceClient<webots_ros::set_float>(modelName + "/right_wheel_motor/set_velocity");
  rightWheelVelocityClient.call(rightWheelSrv);
```

```
  // Init odometry publisher.
  odomPublisher = n.advertise<nav_msgs::Odometry>("odom", 10);
  currentTime = ros::Time::now();
  lastTime = ros::Time::now();
  theta = init_theta;
  xPos = init_xpos;
  yPos = init_ypos;


  oam_reset = TRUE;
  oam_active = FALSE;

  if (setTimeStepClient.call(setTimeStepSrv) && setTimeStepSrv.response.success)
    nStep++;
  else
    ROS_ERROR("Failed to call service time_step to update robot's time step.");


  ROS_INFO("You can now start the creation of the map using 'rosrun gmapping slam_gmapping "
     "scan:=/scan     _xmax:=1.5     _xmin:=-1.5     _ymax:=1.5     _ymin:=-1.5     _delta:=0.005
_linearUpdate:=0.007 "
     "_angularUpdate:=0.05    _resampleThreshold=0.5    _maxUrange:=0.085    _maxRange:=0.0851
_sigma:=0 _particles:=60'.");
  ROS_INFO("You can now visualize the sensors output in rqt using 'rqt'.");
```

```
  // Main loop
  while (1) {
    if (setTimeStepClient.call(setTimeStepSrv) && setTimeStepSrv.response.success)
      nStep++;
    else
```

```
      ROS_ERROR("Failed to call service time_step to update robot's time step.");

    // get sensors value
    if (nStep % 1 == 0) {
      countDist = 0;
      countEnc = 0;
      while (countDist < NB_DIST_SENS || countEnc < NSENSORS)
        ros::spinOnce();
    }


    updateRosInfo();


    // Speed initialization
    speed[LEFT] = 350;
    speed[RIGHT] = 350;


     // if (nStep<4000)
    //      ObstacleAvoidanceModule();
    //    else
    //      ObstacleAvoidanceModule2();

    ObstacleAvoidanceModule2();
    ////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    oam_reset = TRUE;

    // Sum A
    oamOfmSpeed[LEFT] = oam_speed[LEFT];
    oamOfmSpeed[RIGHT] = oam_speed[RIGHT];

    // Suppression A
    if (oam_active) {
      speed[LEFT] = oamOfmSpeed[LEFT];
      speed[RIGHT] = oamOfmSpeed[RIGHT];
    }

    if (!oam_active) {

    }
    // convert speed to rad/s and send command to the motors
    wheelSrv.request.value = (M_PI / 1000.0) * speed[LEFT];
    leftWheelVelocityClient.call(wheelSrv);
    wheelSrv.request.value = (M_PI / 1000.0) * speed[RIGHT];
    rightWheelVelocityClient.call(wheelSrv);
  }


  //disable ir proximity sensors
  for (i = 0; i < NB_DIST_SENS; i++) {
    SubDistIr[i].shutdown();
    ROS_INFO("ds%d disabled.", i);
  }
```

```
  // tells Webots this node will stop using time_step service
  setTimeStepSrv.request.value = 0;
  if (setTimeStepClient.call(setTimeStepSrv) && setTimeStepSrv.response.success)
    ROS_INFO("Robot's time step updated to end simulation.");
  else
    ROS_ERROR("Failed to call service time_step to end simulation.");

////////////////////////////////////////////////////////////////////////////
 /* wheelSrv.request.value = 0;
  leftWheelVelocityClient.call(wheelSrv);
  wheelSrv.request.value = 0;
  rightWheelVelocityClient.call(wheelSrv);*/


  ros::shutdown();

  return 0;
}
```