ΠΑΝΕΜΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία:

ΜΕΛΕΤΗ ΔΥΝΑΤΟΤΗΤΩΝ ΤΟΥ ΚΙΝΟΥΜΕΝΟΥ ΡΟΜΠΟΤ PIONEER P3-DX ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ

υπό

Αγαπουλάκη Βασίλη

Υπεβλήθη για την εκπλήρωση μέρους των απαιτήσεων για την απόκτηση του Διπλώματος Μηχανολόγου Μηχανικού, Οκτώβρης 2012.

Εγκρίθηκε από τα Μέλη της Τριμελούς Επιτροπής

Πρώτος Εξεταστής (Επιβλέπων):
Δρ Κωνσταντίνος Βλάχος Συμβασιούχος Διδάσκων (Π.Δ. 407/80), Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

Δεύτερος Εξεταστής:
Τάσος Σταματέλλος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

Τρίτος Εξεταστής:
Δρ Ερρίκος Σταμπουτζής, Αναπληρωτής Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Πρώτα απ'όλα θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ.Κωνσταντινο Βλάχο για την πολύτιμη βοήθεια του, την επιστημονική καθοδήγηση και την υπομονή του. Επίσης ευχαριστώ τα υπόλοιπα μέρη της επιστημονικής επιτροπής για την ανάγνωση την διόρθωση και τις υποδείξεις στην εργασία.

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου στην οποία και αφιερώνω την παρούσα εργασία για την αγάπη και την υποστήριξη τους.

Αγαπουλάκης Βασίλης

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

Η πτυχιακή εργασία πραγματοποιήθηκε πάνω στο κινητό ρομπότ Pioneer P3-DX (Βλέπε εικόνα 1.1.). Ο σκοπός αυτής της εργασίας ήταν αρχικά να μπει σε ισχύ το εν λόγω μηχάνημα, πραγματοποιώντας την απαραίτητη συνδεσμολογία. Έπειτα έγινε η πρώτη επαφή με το hardware και το software του ρομπότ, έχοντας πάντα ως οδηγό, τα συνοδευτικά του manual. Τέλος στο προγραμματιστικό μέρος της εργασίας γράφτηκαν σε γλώσσα προγραμματισμού C++, κώδικες οι οποίοι μέσα από έναν απομακρυσμένο υπολογιστή ή ακόμα και μέσα από τον ίδιο ενσωματωμένο στο ρομπότ υπολογιστή, είχαν στόχο να κάνουν το Pioneer να πραγματοποιήσει κάποιες εργασίες, όπως ανίχνευση του περιβάλλοντος με τους αισθητήρες ή με την ενσωματωμένη του κάμερα. Επίσης υλοποιήθηκε κώδικας προκειμένου να γίνει έλεγχος (control) της θέσης και της ταχύτητας του κινούμενου ρομπότ. Το σύνολο των κωδίκων περιλαμβάνεται στο συνοδευτικό με την εργασία cd.

Εικόνα 1.1. Το μοντέλο PIONEER P3-DX

Ένα κινητό ρομπότ είναι ένα αυτόματο μηχάνημα, που μπορεί να κινείται σε ένα δεδομένο περιβάλλον. Τα κινητά ρομπότ είναι στο επίκεντρο της τρέχουσας έρευνας και αρκετά πανεπιστήμια ανά τον κόσμο διαθέτουν εργαστήρια τα οποία επικεντρώνονται στην έρευνα αυτή.
Τα κινητά ρομπότ συναντώνται επίσης στη βιομηχανία, σε στρατιωτικές εγκαταστάσεις και σε αποθηκευτικούς χώρους. Μπορούν επίσης να εμφανιστούν ως καταναλωτικά προϊόντα για ψυχαγωγία ή για οικιακές εργασίες.

Τα ρομπότ μπορούν να ταξινομηθούν, ανάλογα με το περιβάλλον στο οποίο ταξιδεύουν:

- σε ρομπότ ξηράς
- σε εναέρια ρομπότ (UAVs-μη επανδρωμένα εναέρια οχήματα)
- σε υποβρύχια ρομπότ (AUVs-αυτόματα υποβρύχια οχήματα)
- σε πολικά ρομπότ (polar robots) σχεδιασμένα για να κινούνται σε περιβάλλοντα με πάγο.

Ανάλογα με τον τρόπο που χρησιμοποιούν για να κινούνται:

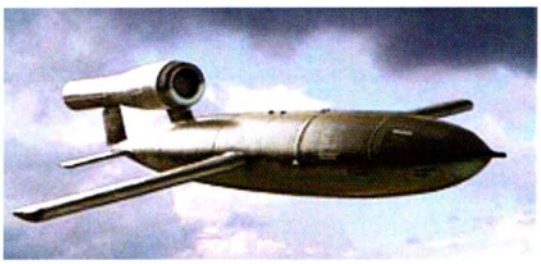
- ρομπότ με πόδια όπως του ανθρώπου
- ρομπότ με πόδια όπως των ζώων
- τροχοφόρα
- ρομπότ με ερπύστριες

Ανάλογα με τον τύπο πλοήγησης έχουμε:

- πλοήγηση από απόσταση αυτόνομη ή από το χρήστη (manual remote), όπου το ρομπότ ελέγχεται με ένα ασύρματο χειριστήριο ή μέσω ενός ασύρματου υπολογιστή.
- φυλασσόμενη πλοήγηση (guarded) στην οποία υπάρχει δυνατότητα ανίχνευσης και αποφυγής των εμποδίων κατά τη διάρκεια της πορείας.
- παρακολούθηση τροχιάς (line-fallowing). Αυτός ο τύπος πλοήγησης χρησιμοποιείται στα πολύ παλαιά μοντέλα κινητών ρομπότ τα οποία μπορούσαν να ακολουθήσουν μία σχεδιασμένη γραμμή στο δάπεδο ή στην οροφή χρησιμοποιώντας τους αισθητήρες τους, σταματώντας και περιμένοντας όμως κάθε φορά που ένα εμπόδιο μπλόκαρε την πορεία τους.
- τυχαία πλοήγηση (wander mode).
- αυτόματη πλοήγηση βάσει οδηγού (guide), όπου το ρομπότ γνωρίζει κάποιες πληροφορίες σχετικά με την τρέχουσα θέση του και το πώς θα προσεγγίσει διάφορους στόχους και σημεία κατά μήκος της διαδρομής. Στα σύγχρονα ρομπότ βέβαια έχουμε συνδυασμούς των παραπάνω ειδών πλοήγησης. [1]

ΚΕΦΑΛΑΙΟ 2. ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Τα πρώτα κινητά ρομπότ προέκυψαν ως αποτέλεσμα των τεχνικών εξελίξεων σε μια σειρά από σχετικά νέους τομείς, όπως η έρευνα της επιστήμης των υπολογιστών, κατά τη διάρκεια του δευτέρου παγκοσμίου πολέμου, με τη μορφή των έξυπνων βομβών, οι οποίες εκπυρσοκροτούν μόνο σε μια συγκεκριμένη περιοχή του στόχου (βλέπε εικόνα 2.1)

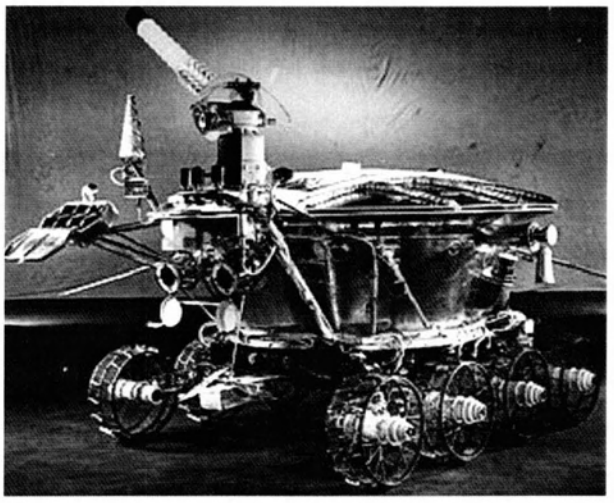


Εικόνα 2.1. Ιπτάμενη βόμβα τύπου V1

Αργότερα, το 1948, ο νευροψυχολόγος και ρομποτιστής William Grey Walter κατασκευάζει τα πρώτα ηλεκτρονικά αυτόνομα ρομπότ elmer και elsie, το καθένα εξοπλισμένο με ένα αισθητήρα φωτός. [2]

Το 1960 κατασκευάζεται το ρομπότ Johns Hopkins Beast. Το μηχάνημα αυτό διέθετε ένα κύκλωμα ελέγχου αποτελούμενο από χιλιάδες τρανζίστορ και είχε μία υποτυπώδη νοημοσύνη.

Το 1970 στο Stanford κατασκευάζεται το πρώτο ρομπότ, με το όνομα shakey, το οποίο μπορούσε να αναλύσει τις οδηγίες του χρήστη σε βασικά κομμάτια, προκειμένου να τις εκτελέσει. Την ίδια χρονιά η πρώην Σοβιετική ένωση κατασκευάζει το όχημα Lunokhod 1, το οποίο μέσω του διαστημικού σκάφους Luna 17, έγινε το πρώτο κατευθυνόμενο ρομπότ που προσγειώθηκε στο φεγγάρι (βλέπε φωτογραφία 2). [3], [4]



Εικόνα 2.2. Το διαστημικό όχημα LUNOKHOD

Στις αρχές της δεκαετίας του 1980 στο Μόναχο κατασκευάζεται το πρώτο ρομπότ αυτοκίνητο.

Το 1989 ο καναδός Mark Tilden αναπτύσσει τα BEAM robotics όπου χρησιμοποιούνται απλά αναλογικά κυκλώματα αντί για έναν μικροεπεξεργαστή με αποτέλεσμα μία ασυνήθιστα απλή σχεδίαση σε σχέση με τα παραδοσιακά κινούμενα ρομπότ. [5]

Το 1990 ο Joseph Engelberger, ο πατέρας του βιομηχανικού ρομποτικού βραχίονα, (βλέπε εικόνα 7), μαζί με συναδέλφους σχεδιάζει το πρώτο εμπορικά διαθέσιμο κινητό ρομπότ για νοσοκομεία. [6].

Το 1993 από το Carnegie Mellon University χρησιμοποιούνται ρομπότ για την εξερεύνηση ενεργών ηφαιστείων.

Το 1994 μέσα από οδηγίες υπολογιστή δύο παρόμοια ρομπότ οχήματα ταξιδεύουν αυτόνομα για πάνω από χίλια χιλιόμετρα στο Παρίσι φτάνοντας ταχύτητες της τάξεως των 130 χιλιομέτρων την ώρα ενώ έναν χρόνο αργότερα ο καθηγητής Ernst Dickmanns κατασκευάζει ένα αντίστοιχο όχημα που ταξιδεύει από το Μόναχο στην Κοπεγχάγη. [7]

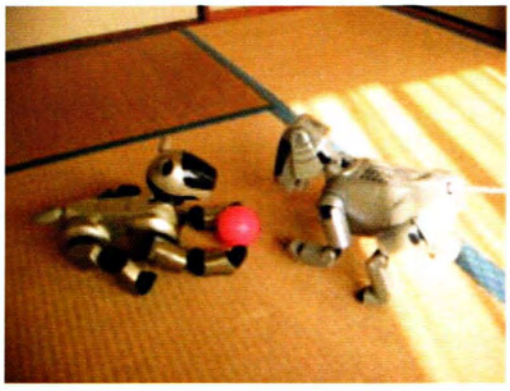
Το 1996 η NASA στέλνει το διαστημικό αεροσκάφος Mars Pathfinder στον Άρη το οποίο αποτελούνταν από ένα σκάφος προσεδάφισης κι ένα ελαφρύ τροχοφόρο ρόβερ. [8]

Το 1999 η Sony κατασκευάζει τον ρομποτικό σκύλο Aibo ικανό να περπατάει, να βλέπει να αλληλεπιδρά με το περιβάλλον του και να αντιδρά σε φωνητικές εντολές στα Αγγλικά και τα Ισπανικά (βλέπε εικόνα 2.3.). [9]

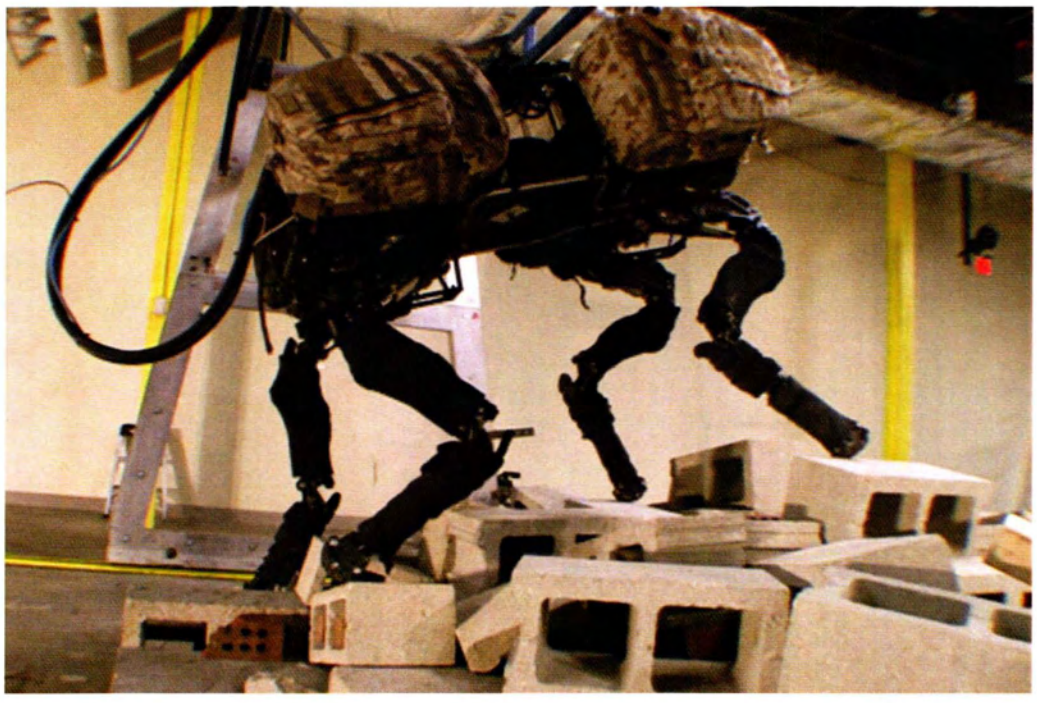
Το 2001 εισάγεται μια νέα προσέγγιση για τον συντονισμό συστημάτων με πολλαπλά ρομπότ (multirobot) με μικρά ρομπότ- έντομα που αποτελούν μια μικρογραφία σμήνους (Swarm). [10]

Το 2004 διοργανώνεται για πρώτη φορά ο αγώνας DARPA Grand Challenge, όπου διαγωνίζονται μόνο ρομπότ-οχήματα. [11]

Το 2005 η εταιρία Boston Dynamics δημιουργεί ένα τετράποδο ρομπότ, με ζωόμορφο σχεδιασμό, που προορίζεται να μεταφέρει βαριά φορτία σε πολύ τραχύ έδαφος (Big Dog). Το Big Dog της εικόνας 2.4., έχει ύψος 0,91m και ζυγίζει 110 kg. [12]



Εικόνα 2.3. Το ρομπότ σκύλος Aibo



Εικόνα 2.4. Το τετράποδο ρομπότ BIG DOG

Εικόνα 2.5. Ρομποτικός βραχίονας

ΚΕΦΑΛΑΙΟ 3. ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΥΠΑΡΧΟΝΤΟΣ ΣΥΣΤΗΜΑΤΟΣ.

3.1. Hardware του Pioneer P3-Dx.

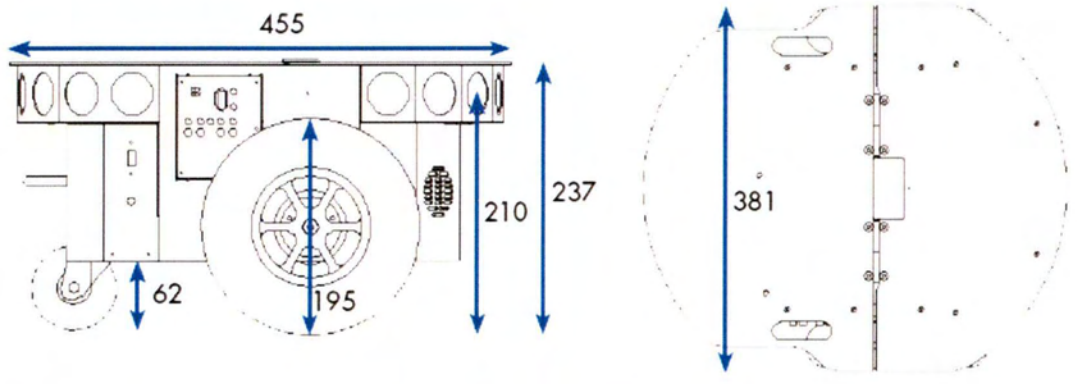
Με βάρος μόλις 9 κιλά (20 κιλά με μία μπαταρία), το ρομπότ Pioneer P3 DX είναι ελαφρύ, αλλά διαθέτει ένα ισχυρό σώμα από αλουμίνιο και στιβαρή κατασκευή, που το καθιστούν σχεδόν άφθαρτο.

Αυτά τα χαρακτηριστικά του επιτρέπουν να μεταφέρει αρκετά μεγάλα φορτία. Το συγκεκριμένο Pioneer μπορεί να μεταφέρει μέχρι 23 κιλά επιπλέον βάρος. Ωστόσο, λόγω του μικρού βάρους του και της ενσωματωμένης λαβής που διαθέτει, είναι εύκολο να μεταφερθεί.

Τα Pioneer ρομπότ αποτελείται από πολλά κύρια μέρη:

- deck (κατάστρωμα)
- πίνακας Ελέγχου Χρήστη
- κυρίως σώμα
- αισθητήρες Sonar, PTZ κάμερα, λέιζερ
- κινητήρας, ρόδες και κωδικοποιητές
- μπαταρίες
- ενσωματωμένος ηλεκτρονικός υπολογιστής κ.λ.π. [14]

Το μοντέλο P3-dx είναι μικρότερο από παρόμοια άλλα κινητά ρομπότ. Όπως φαίνεται στην παρακάτω εικόνα 3.1., το ύψος του Pioneer είναι 237mm, συνολικό μήκος 455mm, το πλάτος 381mm και η διάμετρος κάθε ρόδας 195mm. [15]

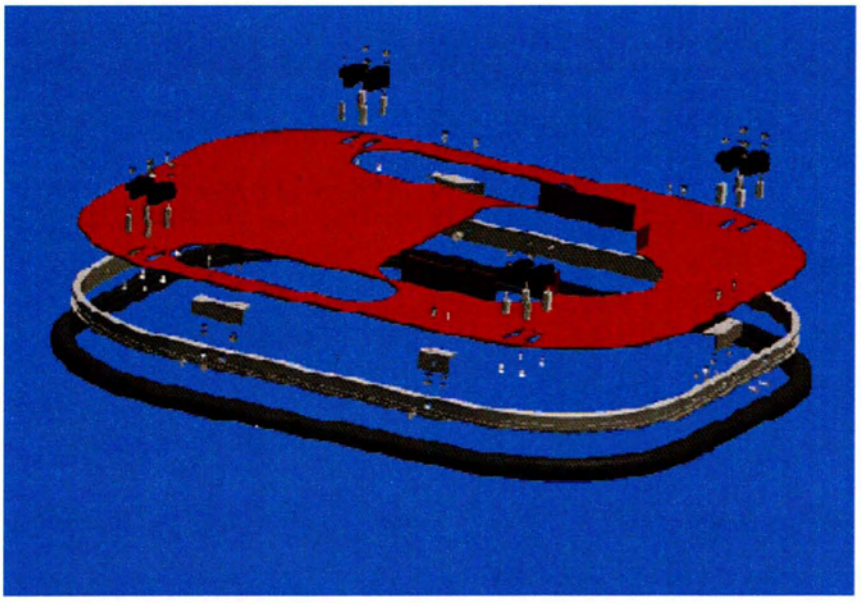


Εικόνα 3.1. Διαστάσεις του Pioneer P3-Dx.

3.1.1. Deck (Κατάστρωμα)

Το μοντέλο Pioneer P3-dx διαθέτει αρθρωτές πλάκες, οι οποίες δίνουν ευκολότερη πρόσβαση στο εσωτερικό του ρομπότ.

Το κατάστρωμα του ρομπότ είναι απλά η επίπεδη επιφάνεια, που χρησιμοποιείται για την τοποθέτηση της PTZ κάμερας και του λέιζερ. Μια αφαιρούμενη τάπα στη μέση του καταστρώματος σε όλα τα μοντέλα, δίνει εύκολη πρόσβαση στο εσωτερικό του ρομπότ. Το κατάστρωμα προστατεύεται με τους απαραίτητους προφυλακτήρες, μπορεί εύκολα να αφαιρεθεί, όπως φαίνεται στην εικόνα 3.2.



Εικόνα 3.2. Το κατάστρωμα του PIONEER P3-DX

3.1.2. Ενσωματωμένος ηλεκτρονικός υπολογιστής

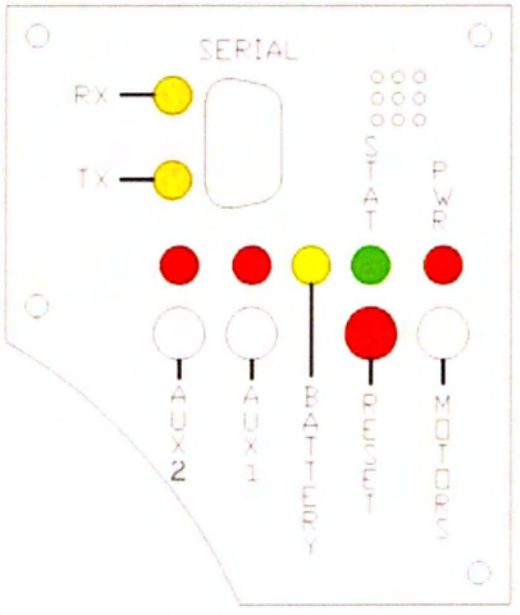
Αφαιρώντας τις βίδες στο κατάστρωμά του ρομπότ, μπορεί κάποιος να δει το εσωτερικό τμήμα του, δηλαδή τον ενσωματωμένο υπολογιστή και το υπόλοιπο hardware. Ο ενσωματωμένος ηλεκτρονικός υπολογιστής διαθέτει έως και τέσσερις σειριακές θύρες, θύρες σύνδεσης για 10/100Base-T Ethernet, οθόνη, πληκτρολόγιο και ποντίκι, δύο θύρες USB και υποστήριξη για δισκέτα, καθώς και IDE σκληρούς δίσκους. Επίσης ο ενσωματωμένος υπολογιστής δέχεται κάρτες PC104 και PC104-plus στη μητρική του και επικοινωνεί με τον μικροελεγκτή του ρομπότ μέσω της host σειριακής του θύρας.

Όλα τα ρομπότ Pioneer P3 DX, έχουν ένα κουμπί STOP στο πίσω μέρος του deck, ώστε να μπορεί ο χρήστης, να θέσει τον κινητήρα του ρομπότ εκτός λειτουργίας. Η παύση λειτουργίας μπορεί να οδηγήσει σε συνεχόμενες ηχητικές ενδείξεις από τα ενσωματωμένα ηχεία, τα οποία διαθέτει το ρομπότ.

.

3.1.3. Πίνακας ελέγχου

Από τον Πίνακα Ελέγχου Χρήστη (user control panel) ο χρήστης μπορεί να ρυθμίσει διάφορες παραμέτρους στον μικροελεγκτή του ρομπότ. Βρίσκεται στην αριστερή πλευρά του ρομπότ, αποτελείται από τα κουμπιά ελέγχου, LED που μας βοηθάνε να καταλάβουμε την κατάσταση στην οποία βρίσκεται το ρομπότ και μια σειριακή θύρα RS-232 (9-pin DSUB connector).



Εικόνα 3.3. Ο Πίνακας ελέγχου χρήστη (User Control Panel) του Pioneer P3-Dx.

Το κόκκινο PWR LED στον πίνακα ανάβει όταν η τάση εφαρμόζεται στο ρομπότ. Το πράσινο STAT LED εξαρτάται από τον τρόπο λειτουργίας και άλλες συνθήκες. Η λυχνία μπαταρίας εξαρτάται από την τάση της μπαταρίας του ρομπότ, δηλαδή πράσινο όταν φορτιστεί πλήρως (> 12.5V), πορτοκαλί όταν βρίσκεται σε ενδιάμεση κατάσταση και κόκκινο όταν η τάση πέφτει κάτω από 11.5V. Όταν το pioneer βρίσκεται σε κατάσταση συντήρησης, η ενδεικτική λυχνία μπαταρίας ανάβει με έντονο κόκκινο, ανεξάρτητα από την κατάσταση της μπαταρίας.

Τα ενσωματωμένα ηχεία δημιουργούν ακουστικές ενδείξεις για την κατάσταση του ρομπότ, την επιτυχή εκκίνηση του μικροελεγκτή και για οποιαδήποτε σύνδεση client.

Οι AUX1 και AUX2 διακόπτες στον πίνακα ελέγχου είναι πλήκτρα, τα οποία παρέχουν έως 5 και 12 VDC. Το κόκκινο πλήκτρο RESET,, χρησιμοποιείται για να επαναφέρει τον μικροελεγκτή, απενεργοποιώντας τυχόν ενεργές συνδέσεις ή τις συνδεδεμένες συσκευές συμπεριλαμβανομένων των κινητήρων.

3.1.4. Αισθητήρες

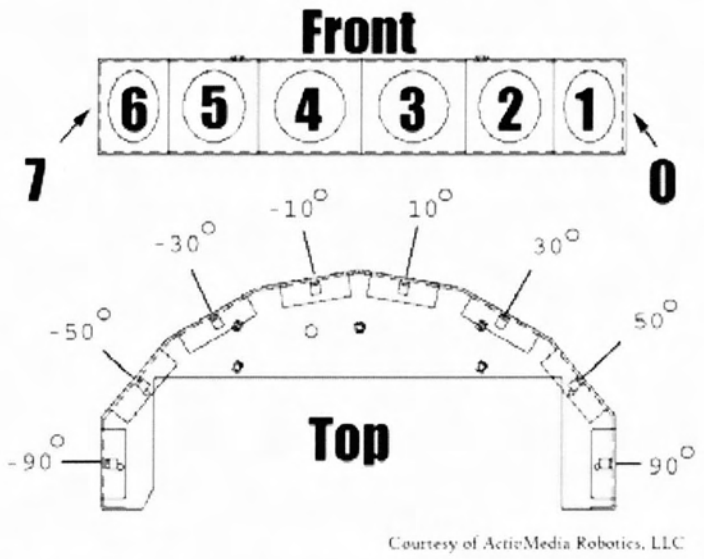
Το ρομπότ διαθέτει έως και οκτώ αισθητήρες, που εξασφαλίζουν την ανίχνευση αντικειμένων και πληροφοριών καθώς και για αποφυγή οποιασδήποτε σύγκρουσης, παρέχοντας δυνατότητες αναγνώρισης, εντοπισμού και πλοήγησης.

Τα σόναρ είναι έτσι διατεταγμένα ώστε να υπάρχει ένα σε κάθε πλευρά της πλατφόρμας και τα υπόλοιπα έξι σε διαστήματα 20^0, ώστε να παρέχεται κάλυψη 360^0 για την πλατφόρμα. Η ακτίνα των σόναρ είναι ρυθμιζόμενη και η συχνότητα στην οποία εκπέμπουν ορίζεται σε 25 Hz (40 χιλιοστά του δευτερολέπτου ανά αισθητήριο ανά συστοιχία). Η ευαισθησία τους κυμαίνεται από 10 εκατοστά (έξι ίντσες) έως πάνω από τέσσερα μέτρα.

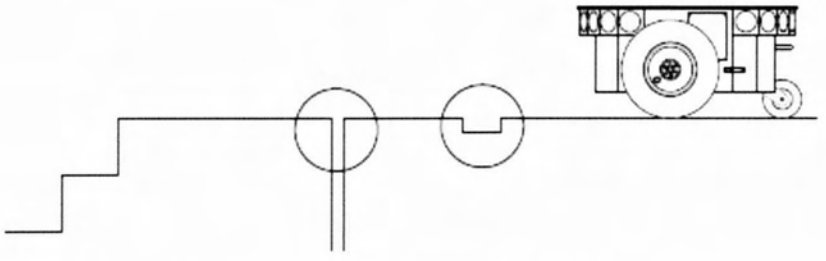
Το Pioneer διαθέτει επίσης λέιζερ (SICK Laser) πάνω από την σειριακή θύρα το οποίο εκτελεί σαρώσεις 180^0 στον χώρο

3.1.5. Κάμερα

Η Canon VC-C που διαθέτει το κινητό ρομπότ, είναι μια φωτογραφική μηχανή, που διαθέτει PTZ (οριζόντια περιστροφή – κατακόρυφη περιστροφή - εστίαση). Η κάμερα ενεργοποιείται πατώντας το πλήκτρο AUX1, που βρίσκεται πάνω στο ρομπότ, οπότε και θα ανάψει ένα πράσινο λαμπάκι. Οι εικόνες διαβάζονται από την ψηφιακή φωτογραφική μηχανή χρησιμοποιώντας μια συσκευή στον ενσωματωμένο στο Pioneer υπολογιστή.



Εικόνα 3.4. Τα ενσωματωμένα Sonars.



Εικόνα 3.5. Αποφυγή εμποδίων με χρήση των σόναρ

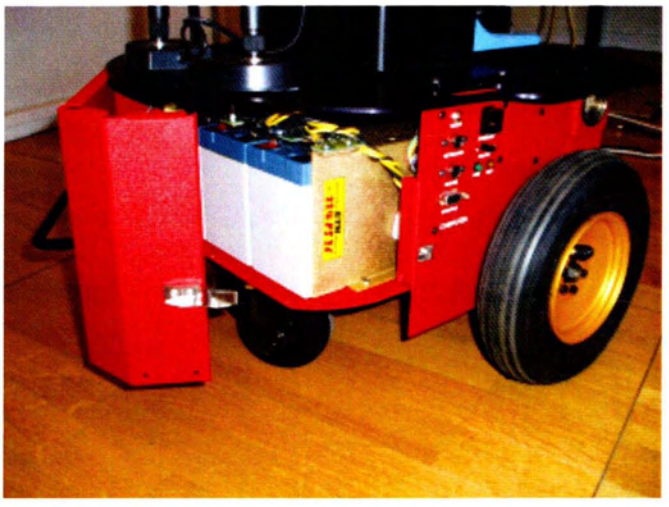
3.1.6. Μπαταρίες.

Το ρομπότ διαθέτει έως και τρεις μπαταρίες συνεχούς ρεύματος (VDC), έντασης 7 A και τάσης 12 V, όπως φαίνονται στην εικόνα 3.6. Το Pioneer DX έχει τη δυνατότητα σύνδεσης με φορτιστή.

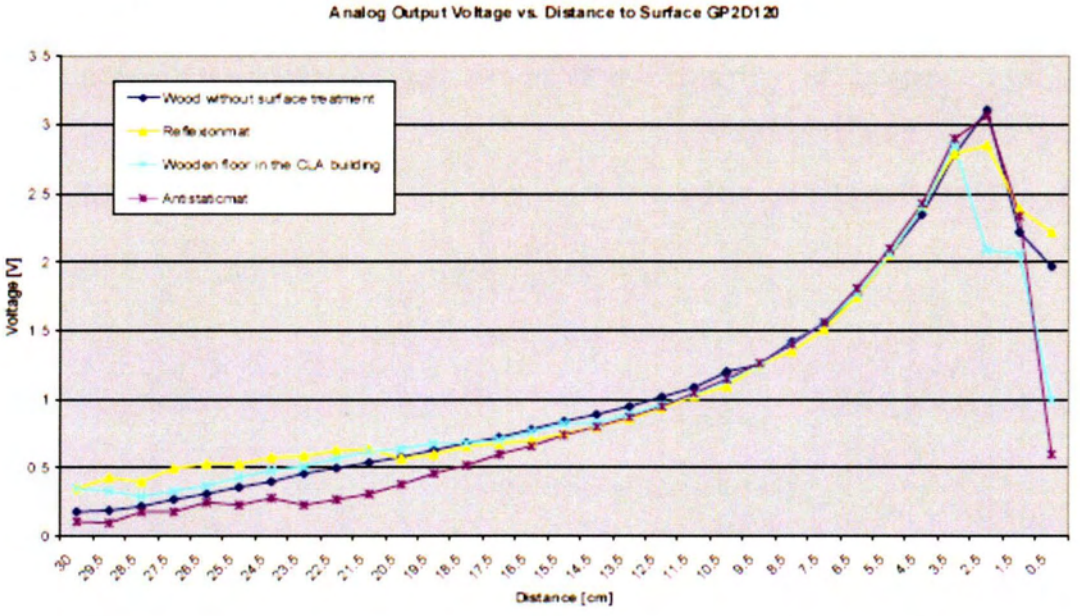
Η διάρκεια ζωής της μπαταρίας, εξαρτάται, φυσικά, από το πλήθος και το είδος των παρελκόμενων του αλλά και της κινητικής δραστηριότητας και διαρκεί έξι ή περισσότερες ώρες. Αν δεν χρησιμοποιείται ο κινητήρας η μπαταρία θα διαρκέσει για αρκετές ημέρες μετά την τελευταία φόρτιση, όπως φαίνεται και το σχήμα 3.7. Οι μπαταρίες έχουν σημαντική επίπτωση στην ισορροπία και τη λειτουργία του ρομπότ. Από τα ηχεία στον Πίνακα Ελέγχου Χρήστη, αν αυτά είναι ενεργά, θα ακουστεί ένας επαναλαμβανόμενος ήχος εάν η

τάση της μπαταρίας πέσει σταθερά κάτω από το προεπιλεγμένο επίπεδο π.χ. των 11,5 VDC και ο μικροελεγκτής κλείνει αυτόματα την σύνδεση του πελάτη και ειδοποιεί τον υπολογιστή να κλείσει ώστε προβλεφθεί η απώλεια δεδομένων.

Ο τυπικός χρόνος επαναφόρτισης της μπαταρίας χρησιμοποιώντας τον έντασης 800 mA φορτιστή ποικίλλει ανάλογα με το επίπεδο φόρτισης και είναι περίπου ίσο με τρεις ώρες, ανά βολτ και ανά μπαταρία.



Εικόνα 3.6. Οι μπαταρίες του PIONEER P3-DX

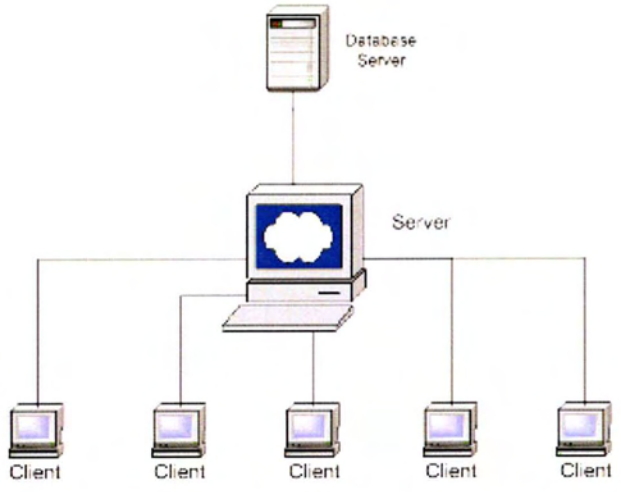


Εικόνα 3.7. Η κατάσταση μπαταρίας σε V, ανάλογα με την απόσταση που έχει διανυθεί από το ρομπότ και της ποιότητας της επιφανείας που κινείται.

3.2. Αρχιτεκτονική CLIENT-SERVER

Ο όρος Client-Server (βλέπε εικόνα 3.8) αναφέρεται σε μία αρχιτεκτονική δικτύων, που συναντάται και στην κατηγορία των κινούμενων ρομπότ, μεταξύ αυτών και το Pioneer P3dx. Η αρχιτεκτονική αυτή διαχωρίζει την εφαρμογή που τρέχει ο τελικός χρήστης (client) από την εφαρμογή που "σερβίρει" τα δεδομένα στο χρήστη (server). Ο client στέλνει αιτήσεις στον server, ζητώντας δεδομένα και ο server ικανοποιεί αυτές τις αιτήσεις, επιστρέφοντας τα δεδομένα στον client. Με αρχιτεκτονική client/server λειτουργεί η δημοφιλέστερη υπηρεσία του Internet, το World Wide Web ή Παγκόσμιος Ιστός. Όλες οι ιστοσελίδες είναι αποθηκευμένες σε έναν Web server. Όταν ο χρήστης περιηγείται στο ίντερνετ, με έναν browser, και πρόκειται να επισκεφθεί μία ιστοσελίδα, ο browser στέλνει αιτήσεις στον Web server, ζητώντας το κατέβασμα της ιστοσελίδας στον υπολογιστή. Με αρχιτεκτονική client/server είναι φτιαγμένες όλες οι σύγχρονες επιχειρησιακές εφαρμογές,.

Η εφαρμογή μισθοδοσίας ή διαχείρισης αποθεμάτων είναι εγκατεστημένη και τρέχει στον application server, ο οποίος με τη σειρά του τη "σερβίρει" στους υπολογιστές client που είναι συνδεδεμένοι στο δίκτυο, εξυπηρετώντας τις αιτήσεις των τελικών χρηστών. [13]



Εικόνα 3.8. Αρχιτεκτονική CLIENT SERVER

Σε μία εφαρμογή που δεν έχει σχεδιαστεί με αρχιτεκτονική client/server, η προβολή των δεδομένων και η λογική της εφαρμογής (οι αλγόριθμοι που χειρίζονται την επεξεργασία και τη ροή των πληροφοριών), λαμβάνουν χώρα εντός αυτής. Αλλά, στις εφαρμογές αρχιτεκτονικής client-server, η ροή πληροφοριών είναι πολύ διαφορετική.

Ο χρήστης τρέχει το client κομμάτι της εφαρμογής τοπικά. Όταν χρειαστεί πληροφορίες από τη βάση δεδομένων, ο client αναλαμβάνει να δημιουργήσει μία αίτηση ή ένα ερώτημα, το μορφοποιεί κατάλληλα και το στέλνει στον server. Ο server ελέγχει τα δικαιώματα πρόσβασης του χρήστη στις συγκεκριμένες πληροφορίες.

Ο server επεξεργάζεται το ερώτημα και επιστρέφει τα αποτελέσματα στον client ο οποίος με την σειρά του παραλαμβάνει τις πληροφορίες που ζήτησε ο χρήστης, τις μορφοποιεί κατάλληλα και τις παρουσιάζει στο χρήστη.
Ο διαχωρισμός της λογικής της εφαρμογής από τη βάση δεδομένων έχει πολλά πλεονεκτήματα. Η βασική επεξεργασία γίνεται στον server, το σύστημα στο οποίο τρέχει ο client, δεν χρειάζεται να είναι ιδιαίτερα ισχυρό και οι πόροι του αποδεσμεύονται μετά την αποστολή του ερωτήματος προς τον server. Μειώνεται ο φόρτος διακίνησης δεδομένων στο δίκτυο, διότι δεν υπάρχει ανάγκη να μεταβιβαστούν ολόκληρα αρχεία από και προς τον client. Η αποσυμφόρηση της "κυκλοφορίας" στο δίκτυο είναι πολύ σημαντική, ιδίως στα μεγάλα εταιρικά δίκτυα που εξυπηρετούν εκατοντάδες ή και χιλιάδες clients.
Οι clients μπορούν να τρέχουν σε οποιαδήποτε πλατφόρμα. Εφόσον είναι σε θέση να διατυπώνουν σωστές αιτήσεις SQL στον server, δεν έχει καμία σημασία για τον τελικό χρήστη η πλατφόρμα στην οποία δουλεύουν και η γραφική επιφάνεια εργασίας της εφαρμογής που χρησιμοποιούν. Είναι ελεύθεροι να επιλέξουν αυτή που προτιμούν.

Με τη διατήρηση της βάσης δεδομένων σε σύστημα διαφορετικό από αυτό στο οποίο τρέχουν οι clients, διασφαλίζεται στο μέγιστο δυνατό βαθμό η ακεραιότητα των πληροφοριών που φυλάσσονται εκεί. Είναι πολύ πιο εύκολο να ελεγχθεί, να συντηρηθεί (π.χ., με τη δημιουργία αντιγράφων ασφαλείας), να προστατευτεί και να ανακτηθεί (σε περίπτωση αστοχίας του αποθηκευτικού μέσου) μία βάση δεδομένων που είναι αποθηκευμένη.

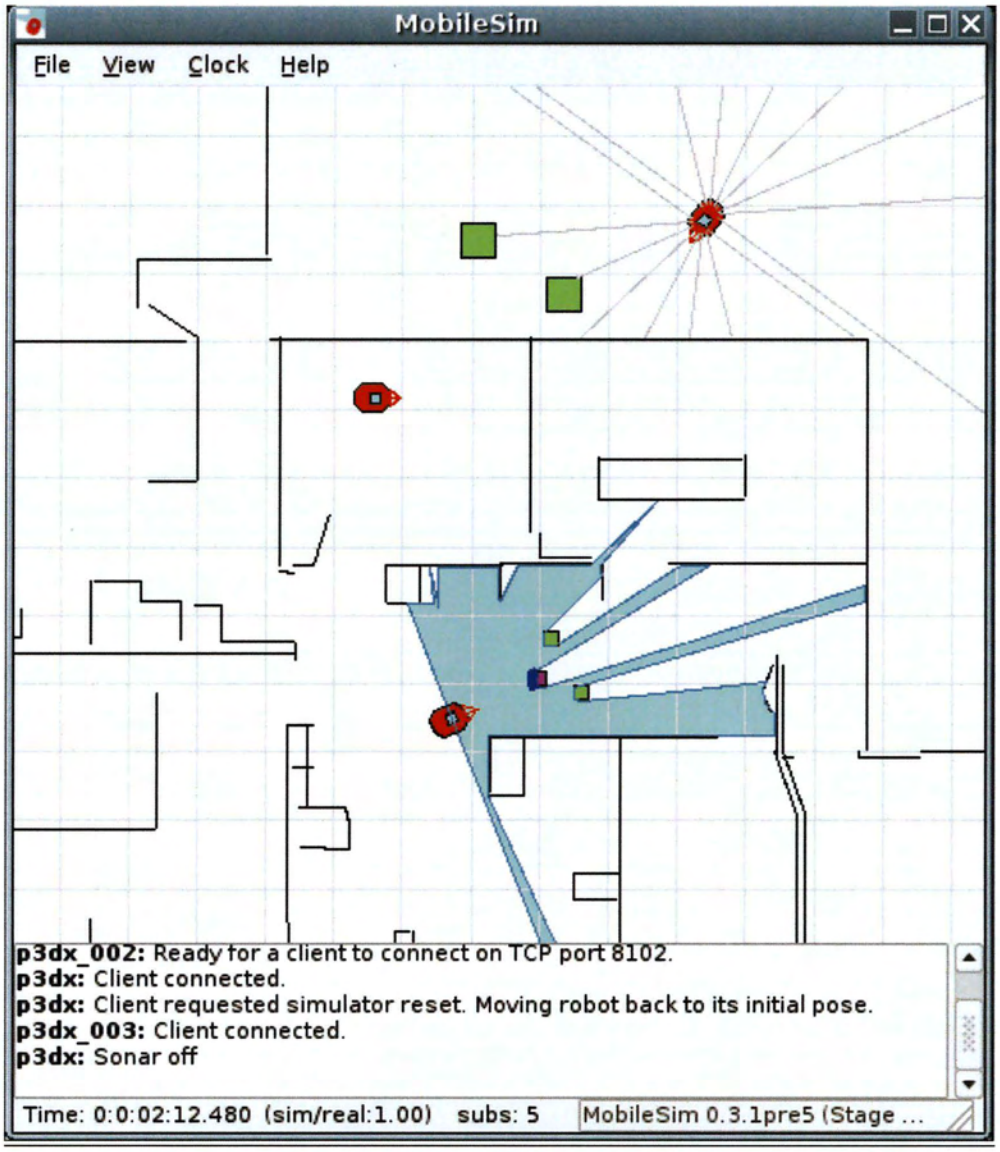
3.3: SOFTWARE TOY Pioneer P3-Dx.

3.3.1. ARIA

Η ARIA [16] (Mobile Robots' Advanced Robot Interface for Applications), είναι μια C++ βιβλιοθήκη (λογισμικού ανάπτυξης εργαλείων ή SDK), που χρησιμοποιείται για όλες τις MobileRobots/ActivMedia πλατφόρμες. Μέσω της ARIA, οι χρήστες των ρομπότ pioneer μπορούν να ελέγξουν δυναμικά την ταχύτητα του ρομπότ και άλλες παραμέτρους της κίνησης του είτε μέσω απλών χαμηλού επιπέδου εντολών είτε μέσω υψηλού επιπέδου υποδομή των δράσεών της. Μέσω της ARIA λαμβάνονται επίσης εκτιμήσεις για την θέση, τους ενσωματωμένους αισθητήρες και όλα τα άλλα τρέχοντα δεδομένα λειτουργίας που αποστέλλονται από την πλατφόρμα ρομπότ. Το ARIA παρέχει εργαλεία για την επαρκή επικοινωνία του ρομπότ με εξωτερικούς υπολογιστές και την διανομή υλικού ψηφιακής και αναλογικής μορφής και περιλαμβάνει επίσης μια βιβλιοθήκη, που ονομάζεται ArNetworking, η οποία είναι επεκτάσιμη και βοηθάει τον χρήστη στον εύκολο εξ΄ αποστάσεως έλεγχο των λειτουργιών του δικτύου για το ρομπότ. Ο client του ArNetworking έχει την δυνατότητα μέσα από ένα διακομιστή να συνδεθεί από έναν άλλο υπολογιστή στο δίκτυο για να πάρει δεδομένα και εντολές.

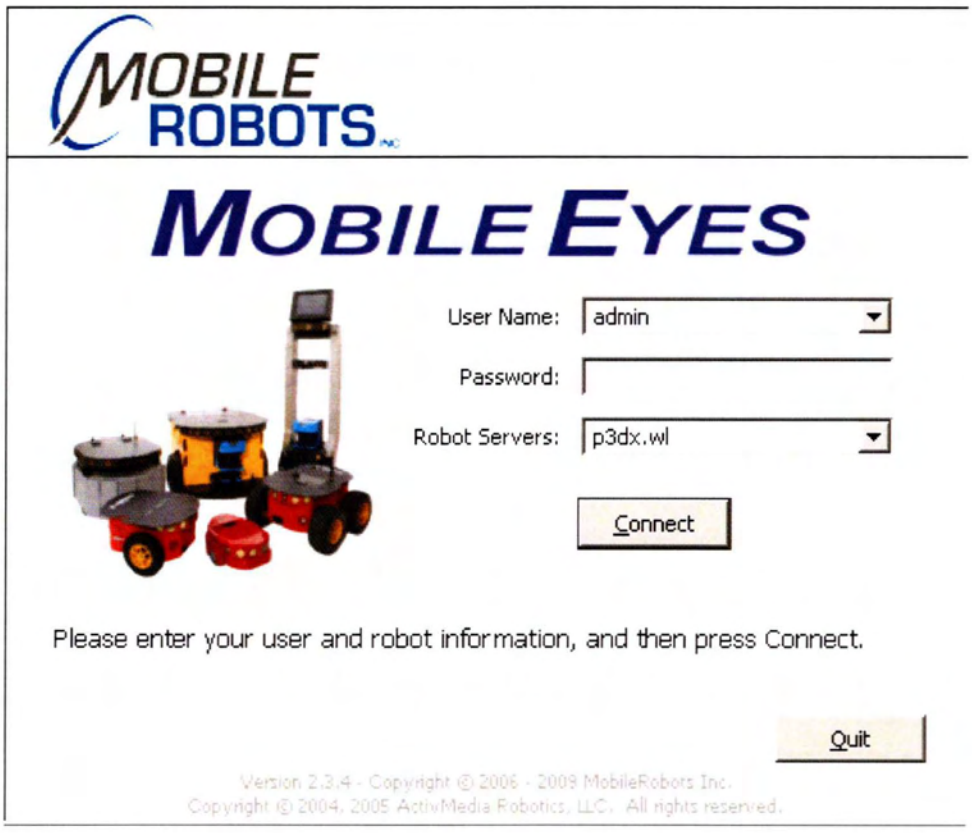
Μια ποικιλία από άλλα χρήσιμα εργαλεία για τη δημιουργία εφαρμογών στο ρομπότ περιλαμβάνονται στην ARIA ή διατίθενται ως χωριστές βιβλιοθήκες, συμπεριλαμβανομένης της σύνθεσης φωνής και αναγνώρισης ήχου. Αν και η βιβλιοθήκη ARIA είναι γραμμένη σε C++.,η πρόσβαση στις περισσότερες από τις εφαρμογές της ARIA είναι επίσης διαθέσιμη και σε άλλες γλώσσες προγραμματισμού όπως Java και Python.

3.3.2 MOBILE SIM



Εικόνα 3 Το λογισμικό MOBILE SIM

Το MobileSim [17] είναι το λογισμικό, που χρησιμοποιείται για την προσομοίωση του κινητό ρομπότ pioneer και του περιβάλλοντος του, καθώς και για τον εντοπισμό σφαλμάτων με βοήθεια της ARIA ή κάποιου άλλου λογισμικού, που υποστηρίζουν οι πλατφόρμες MobileRobots. Βασίζεται στην βιβλιοθήκη Stage, και είναι ένα ελεύθερο λογισμικό. Το πρόγραμμα μετατρέπει έναν χάρτη που δημιουργήθηκε μέσα από το Mapper3 ™, το Mapper3-Basic, ή με άλλα μέσα σε ένα περιβάλλον, και τοποθετεί το προσομοιωμένο μοντέλο ρομπότ στο περιβάλλον αυτό και στη συνέχεια παρέχει μια προσομοίωση σύνδεση ελέγχου, μέσω μιας θύρας TCP.



Εικόνα 3.10. Εικόνα έναρξης του λογισμικού MOBILE EYES

3.3.3. MobileEyes.

Το MobileEyes [18] είναι ένα λογισμικό με γραφικό περιβάλλον, που χρησιμοποιείτε από την MobileRobots, για τον απομακρυσμένο έλεγχο ενός ή περισσότερων ρομπότ. Μέσα από το πρόγραμμα ο χρήστης μπορεί να έχει θέαση του ρομπότ που ελέγχει καθώς και του γύρω περιβάλλοντος του.

3.3.4. Mapper 3

Το Mappper3 [19] είναι ένα interface με γραφικό περιβάλλον, το οποίο χρησιμοποιείται για το σχεδιασμό χαρτών, τον εντοπισμό στόχων, το μαρκάρισμα ανεπιθύμητων περιοχών κ.α. Χρησιμοποιώντας το πρόγραμμα ο χρήστης μπορεί να επεξεργαστεί τις παραπάνω λειτουργίες, ανάλογα με τις εκάστοτε ανάγκες. Τα αρχεία, που δημιουργούνται από το λέιζερ σάρωσης, είναι μορφής .2d και με το mapper3 μετατρέπονται σε χάρτες (.map).

3.3.5. Arcos

Το ARCOS [20] είναι το ενσωματωμένο λογισμικό (firmware), που τρέχει στον μικροελεγκτή του ρομπότ Pioneer 3-DX. Λογισμικό ARCOS υπάρχει

επίσης και στα υπόλοιπα ρομπότ της MobileRobots όπως τα PeopleBot, PowerBot και MapperBot SH2 προσαρμοσμένο στις απαιτήσεις του μικροελεγκτή τους. Στο ARCOS περιέχονται επίσης βοηθητικά προγράμματα λειτουργίας συντήρησης (ARCOSstub GDB interface) και ρυθμίσεων των παραμέτρων λειτουργίας της μνήμης του ρομπότ.

Όλες οι πλατφόρμες της MobileRobots χρησιμοποιούν μια client-server αρχιτεκτονική για τον κινητό έλεγχο του ρομπότ. Οι διακομιστές του ρομπότ λειτουργούν έτσι ώστε το ρομπότ να μπορεί να διαχειριστεί όλες τις χαμηλού επιπέδου λεπτομέρειες των συστημάτων. Σε αυτές τις λειτουργίες περιλαμβάνονται η λειτουργία του μοτέρ, η έναρξη του σόναρ και η συλλογή και η υποβολή πληροφοριών σχετικά με τους αισθητήρες, τα δεδομένα από τον κωδικοποιητή του τροχού και άλλες, μέσα από την υποβολή εκθέσεων σε μια εφαρμογή πελάτη, όπως το demo του ARIA.

3.3.6. Arnl

Το ARNL [21] είναι ένα σύνολο από πακέτα λογισμικού, που βασίζεται στην βιβλιοθήκη του ARIA, για έξυπνη πλοήγηση και για εντοπισμό. Λέγοντας "εντοπισμός" εννοούμε την ικανότητα στο πρόγραμμα να παρακολουθεί ο κάθε χρήστης τη θέση του ρομπότ και λέγοντας "πλοήγηση" την ικανότητα του προγράμματος να στείλει τις απαραίτητες οδηγίες στο ρομπότ, ώστε να φτάσει έναν συγκεκριμένο προορισμό. Οι βιβλιοθήκες λειτουργούν και συντονίζουν τα δύο αυτά καθήκοντα, και μέσω ενός απομακρυσμένου υπολογιστή-πελάτη, όπως το MobileEyes η μέσω του ArNetworking πρωτοκόλλου, γίνεται ο έλεγχος του ρομπότ.

Οι βιβλιοθήκες είναι γραμμένες σε C + + αλλά εφαρμογές υπάρχουν και σε υπόλοιπες γλώσσες προγραμματισμού όπως Python και Java. Τα διαθέσιμα πακέτα του ARNL περιλαμβάνουν τις κοινές βιβλιοθήκες / DLLs, αρχεία κεφαλής, εγχειρίδια αναφοράς και παράδειγμα προγραμμάτων με τον πηγαίο κώδικα. Το ARNL περιλαμβάνει το SONARNL, το οποίο είναι παρόμοιο με το ARNL, αλλά για χρήση των σόναρ και ενός χάρτη γραμμών.

3.3.7. Sav

Το SAV [22] είναι ένα λογισμικό πακέτο μετάδοσης AV. Ο διακομιστής του SAV χρησιμοποιείται για την αποστολή εικόνων, που ανακτώνται από μια

συσκευή βίντεο μέσω του δικτύου (με χρήση της βιβλιοθήκης ArNetworking που παρέχεται από το ARIA) σε έναν πελάτη, όπως το MobileEyes.

3.3.8 Acts

Ο πελάτης (client), με τον οποίο μπορεί να γίνει η σύνδεση για να υπάρξει εικόνα από το ρομπότ, λέγεται EZ-train. Σε όλες τις εκδόσεις του ACTS [23] , με την έναρξη της εφαρμογής, ανοίγουν δύο παράθυρα. Η EZ-train κονσόλα και ένα παράθυρο γραφικών με κάποιες πληροφορίες. Υπάρχει δυνατότητα για live video, όπου το παράθυρο κονσόλας εμφανίζεται μαζί με ένα άλλο παράθυρο, που περιέχει ζωντανά καρέ βίντεο από την PTZ κάμερα, που είναι ενσωματωμένη στο ρομπότ. Για λειτουργία στατικής εικόνας, μπορεί να επιλεγεί μια εικόνα μορφής .ppm, ώστε αυτή να εμφανίζεται στην οθόνη και να επεξεργάζεται από τον εκάστοτε χρήστη.

ΚΕΦΑΛΑΙΟ 4. ΜΕΛΕΤΗ ΤΩΝ ΔΥΝΑΤΟΤΗΤΩΝ ΤΟΥ ΚΙΝΟΥΜΕΝΟΥ ΡΟΜΠΟΤ.

Μετά την πρώτη επαφή με το Pioneer P3-dx στο εργαστήριο, το επόμενο βήμα ήταν η επισκόπηση των δυνατοτήτων του ρομπότ, μέσα από το υλικό και το διαθέσιμο λογισμικό του.

Αρχικά πρέπει να ειπωθεί ότι το λογισμικό σύστημα του ενσωματωμένου ηλεκτρονικού υπολογιστή στο ρομπότ είναι debian linux και επομένως όλα τα αναφερόμενα στο προηγούμενο κεφάλαιο προγράμματα τρέχουν σε αυτό το περιβάλλον, είτε από τα αντίστοιχα εικονίδια στην επιφάνεια εργασίας, είτε μέσα από το X-terminal που διαθέτουν τα linux. Το σύνολο σχεδόν των προγραμμάτων επίσης τρέχει και από κάποιον απομακρυσμένο υπολογιστή (ανεξαρτήτως του λειτουργικού συστήματος του) μέσω ενός απλού καλωδίου usb to serial, όπως και δοκιμάστηκε πολλές φορές επιτυχώς.

Στις παραγράφους που ακολουθούν αναλύεται πως ένας νέος χρήστης θα καταφέρει να συνδεθεί με το Pioneer και γίνεται μια σύντομη ανάλυση του ηλεκτρονικού υπολογιστή του ρομπότ. Περιγράφονται οι βασικές δυνατότητες του, όπως η ικανότητα του για ασφαλή πλοήγηση, για χαρτογράφηση και για ανίχνευση των αντικειμένων με τη βοήθεια των αισθητήρων και της ενσωματωμένης κάμερας του.

4.1. Αρχική σύνδεση.

Για την αρχική σύνδεση με το ρομπότ χρειάζεται να συνδεθεί το πληκτρολόγιο, η οθόνη και το ποντίκι, στις αντίστοιχες υποδοχές στον Πίνακα Ελέγχου και στη συνέχεια να τεθεί σε λειτουργία ο υπολογιστής.

Μετά την εκκίνηση, έχει γίνει πλέον σύνδεση με το σύστημα και υπάρχει η πρόσβαση στο λογισμικό του PC ως root (Linux) ή «διαχειριστής» (Windows).

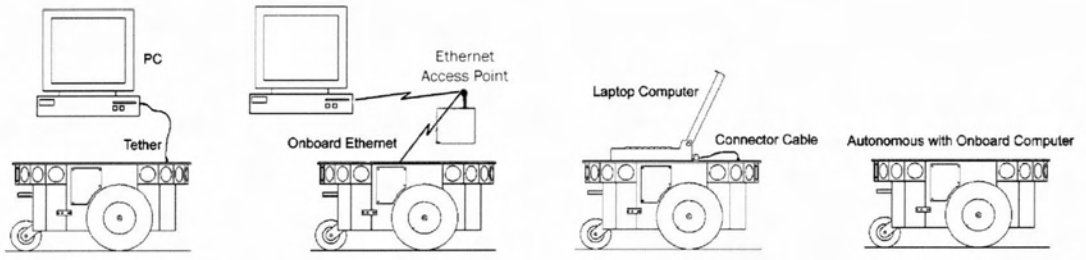
Σημειώνεται ότι για να αποκτήσει πρόσβαση ο χρήστης σαν root στα debian linux του υπολογιστή του ρομπότ το username είναι *root* και ο κωδικός πρόσβασης *toor*.

Όταν γίνει η σύνδεση ως root, μπορεί ο εκάστοτε χρήστης να κάνει τις ρυθμίσεις που επιθυμεί και να διαχειριστεί το σύστημα, όπως π.χ. να μπορεί να αλλάζει κωδικούς πρόσβασης, να υπάρχει δυνατότητα προσθήκης και άλλων χρηστών, να δημιουργηθεί δίκτυο κτλ.

Όταν ο χρήστης είναι συνδεδεμένος σε λειτουργικό σύστημα των Windows, εύκολα κάνοντας κλικ στο ποντίκι μπορεί να επιλέξει τα προγράμματα και τις εφαρμογές. Με το Linux, μπορεί να χρησιμοποιήσει την startx εντολή για την ενεργοποίηση επιφάνεια εργασίας με γραφικό περιβάλλον παρόμοιο με αυτό των Windows.

Όσον αφορά την εγκατάσταση διαδικτύου στο ρομπότ, θα πρέπει να υπάρχει ένα σημείο πρόσβασης (access point) στο τοπικό δίκτυο LAN. Ο τρόπος λειτουργίας που χρησιμοποιείται, είναι η client-server αρχιτεκτονική, που έχει περιγραφεί και παραπάνω.

Αν και δεν χρειάζεται να ασχοληθεί ο χρήστης με οδηγούς και χαμηλού επιπέδου ρυθμίσεις της συσκευής, πριν να μπορέσει να δημιουργήσει μια σύνδεση δικτύου με το onboard PC ακόμα και στην απλή περίπτωση ενός «cross-over" καλωδίου Ethernet σε ένα άλλο υπολογιστή, θα χρειαστεί να επαναριθμήσει τις ρυθμίσεις δικτύου στον υπολογιστή του ρομπότ.



Εικόνα 4.1. Τρόποι σύνδεσης ρομπότ με Η/Υ.

Εν συντομία, με τα Windows οι απαραίτητες ρυθμίσεις μπορούν να γίνουν στο μενού network, από τον Πίνακα Ελέγχου και από εκεί στις network and Dial-up Connections. Από εκεί στα properties μπορεί να γίνει η αλλαγή της διεύθυνσης IP και να ρυθμιστούν κι άλλες λεπτομέρειες της σύνδεσης. Στα debian Linux του ρομπότ υπάρχουν παρόμοια εργαλεία στα X-Windows, για να βοηθήσουν στις ρυθμίσεις του δικτύου, όπως η netcfg, αλλά, για να επιτευχθεί μια εύκολη σύνδεση, προτιμότερο είναι να γίνει από τις ρυθμίσεις δικτύου, που βρίσκονται στο / etc / network /interfaces.

4.2. Μέθοδοι πλοήγησης.

4.2.1. Πλοήγηση με το DEMO του ARIA.

Τα παραδείγματα στο ARIA είναι εφαρμογές που δεν περιλαμβάνουν γραφικό περιβάλλον, έτσι ώστε τα προγράμματά της δεν απαιτούν X-Windows πάνω από το Linux ή ειδικό λογισμικό σε απομακρυσμένο υπολογιστή πελάτη. Απαιτείται ειδική συσκευή Ethernet-to-serial η εναλλακτικά ειδικό λογισμικό που να βρίσκεται στο onboard computer και να μετατρέπει τα πακέτα ip σε σειριακά δεδομένα. Εναλλακτικά, τα δεδομένα μπορούν να περαστούν από ένα pc, το οποίο να συνδέεσαι στο ρομπότ, από μια σειριακή θύρα.

Αρχίζουμε με την περίπτωση της σειριακής, η οποία είναι και η απλούστερη. Για επιβεβαίωση της επικοινωνίας μεταξύ του ενσωματωμένου υπολογιστή του ρομπότ και ενός άλλου απομακρυσμένου υπολογιστή, μπορεί να χρησιμοποιηθεί η εντολή ping στο αντίστοιχο ip του ρομπότ.
Από προεπιλογή, το πρόγραμμα ARIA συνδέεται με το ρομπότ μέσω της σειριακής θύρας (π.χ. την COM1, ανάλογα με τον υπολογιστή στα Windows ή /dev/ttyS0 στα Linux).
Στα windows η σύνδεση μπορεί να γίνει μέσω command line, π.χ. C:\Program Files\MobileRobots\ARIA\bin\demo.exe -remoteHost 192.168.1.32 η -rp com1 (ανάλογα πάντα με την θύρα).

Σε περίπτωση που πετύχει η σύνδεση με το ρομπότ, το ARIA θα εμφανίσει στην οθόνη μερικά διαγνωστικά, θα ακουστεί ένα ηχητικό σύνθημα, που θα επιβεβαιώνει την επιτυχημένη σύνδεση, από τα σόναρ του ρομπότ θα ακουστεί ένα διακριτικό και επαναλαμβανόμενο κλικ και το LED, που σχετίζεται με τους κινητήρες, θα πρέπει να αναβοσβήνει πολύ γρήγορα (πιο πριν αναβόσβηνε αργά, εν αναμονή σύνδεσης).

Όταν συνδεθεί με το ARIA, το ρομπότ ανταποκρίνεται και μπορεί να κινείται με προσοχή. Αν και μπορεί να κατευθυνθεί προς ένα εμπόδιο, το ρομπότ δεν θα προσκρούσει πάνω του, επειδή το demo ARIA περιλαμβάνει σύστημα αποφυγής εμποδίων, που επιτρέπουν στο ρομπότ να ανιχνεύει τα εμπόδια και να αποφευχθούν ανεπιθύμητες συγκρούσεις.
Το demo του Aria [24] περιλαμβάνει διάφορα είδη λειτουργίας, τα οποία περιγράφονται στη συνέχεια.

- Teleoperation mode: Είναι η προεπιλεγμένη λειτουργία, που αρχίζει μετά την επιτυχή έναρξη του demo. Μέσω αυτής, ο χρήστης έχει την δυνατότητα να κατευθύνει το ρομπότ μέσω του πληκτρολογίου και πιο συγκεκριμένα με τα βέλη κατεύθυνσης ή εναλλακτικά με κάποιο ενσωματωμένο joystick. Επίσης στην Teleoperation mode, το ρομπότ έχει την ικανότητα να αποφεύγει τις συγκρούσεις με όποια εμπόδια συναντήσει στον χώρο. Ενεργοποιείται με το πλήκτρο t του πληκτρολογίου.

- Unguarded teleoperation mode: Στην περίπτωση της Unguarded teleoperation mode το ρομπότ δεν έχει την ικανότητα αποφυγής εμποδίων, οπότε απαιτείται λίγη περισσότερη προσοχή. Ο τρόπος οδήγησης είναι ο ίδιος με την Teleoperation mode. Ενεργοποιείται με το πλήκτρο u του πληκτρολογίου.

- Wander mode: Με το που θέσουμε σε ισχύ αυτήν την λειτουργία, το ρομπότ αρχίζει να κινείτε αυτόνομα και να περιφέρεται στον χώρο, αποφεύγοντας ταυτόχρονα τα εμπόδια που συναντάει. Ο τερματισμός της λειτουργίας γίνεται με το πλήκτρο escape του πληκτρολογίου. Ενεργοποιείται με το πλήκτρο w.

- Camera mode: Με αυτή η λειτουργία γίνεται ο χειρισμός και ο έλεγχος της PTZ (Pan-Tilt-Zoom) ενσωματωμένης στο ρομπότ κάμερας. Ενεργοποιείται με το πλήκτρο c του πληκτρολογίου.

- Position mode: Δείχνει στην οθόνη σε πραγματικό χρόνο τις συντεταγμένες της θέσης του ρομπότ στον χώρο, σχετικά με την αρχική του θέση (την 0,0). Ενεργοποιείται με το πλήκτρο p του πληκτρολογίου.

- Bumps mode: Ο χρήστης μπορεί να δει στην οθόνη κάποιες ενδείξεις σχετικά με την κατάσταση των προφυλακτήρων του ρομπότ. Ενεργοποιείται με το πλήκτρο b από το πληκτρολόγιο.

- Laser mode: Σε αυτήν την λειτουργία, ο χρήστης μπορεί να δει στην οθόνη τις ενδείξεις (τις πιο κοντινές και πιο μακρινές) από τα ενσωματωμένα laser του ρομπότ. Ενεργοποιείται με το πλήκτρο l από το πληκτρολόγιο.

- Sonar mode: Εμφανίζονται στην οθόνη οι ενδείξεις από τους αισθητήρες σόναρ, που διαθέτει το ρομπότ. Ενεργοποιείται με το πλήκτρο s του πληκτρολογίου.

- Gripper mode: Στην περίπτωση που έχουμε ενσωματωμένο βραχίονα στο ρομπότ, με αυτό το mode, ο χρήστης μπορεί να τον χειριστεί και να πάρει πληροφορίες, που αφορούν την κατάστασή του. Ενεργοποιείται με το πλήκτρο g του πληκτρολογίου.

- IO mode: Ο χρήστης εδώ μπορεί να δει στην οθόνη τις ψηφιακές και αναλογικές I/O θήρες. Ενεργοποιείται με το πλήκτρο I του πληκτρολογίου.

- Direct mode: Λειτουργία χειρισμού του ρομπότ με απευθείας εντολές. (command mode). Ενεργοποιείται με το πλήκτρο d του πληκτρολογίου.

Ο τερματισμός του demo και η αποσύνδεση του Aria client από τον υπολογιστή του ρομπότ γίνεται με το πλήκτρο escape. Οι κινητήρες και τα σόναρ του ρομπότ παύουν την λειτουργία τους και πλέον είναι ασφαλές ο χρήστης να κλείσει τον διακόπτη ισχύος - main power του ρομπότ.

4.2.2. Πλοήγηση με τον διακομιστή SONarnl και τον πελάτη MobileEyes

Ο διακομιστής SONarnl επιτρέπει τη σύνδεση ενός πελάτη-client [25] (π.χ. MobileEyes) με μια προσομοίωση του server του ρομπότ. Κατά συνέπεια μπορεί να χρησιμοποιήσει όλα τα sonars για ασφαλή πλοήγηση καθώς και πολλά άλλα χαρακτηριστικά γνωρίσματα του λογισμικού (ARIA, ARntworking).

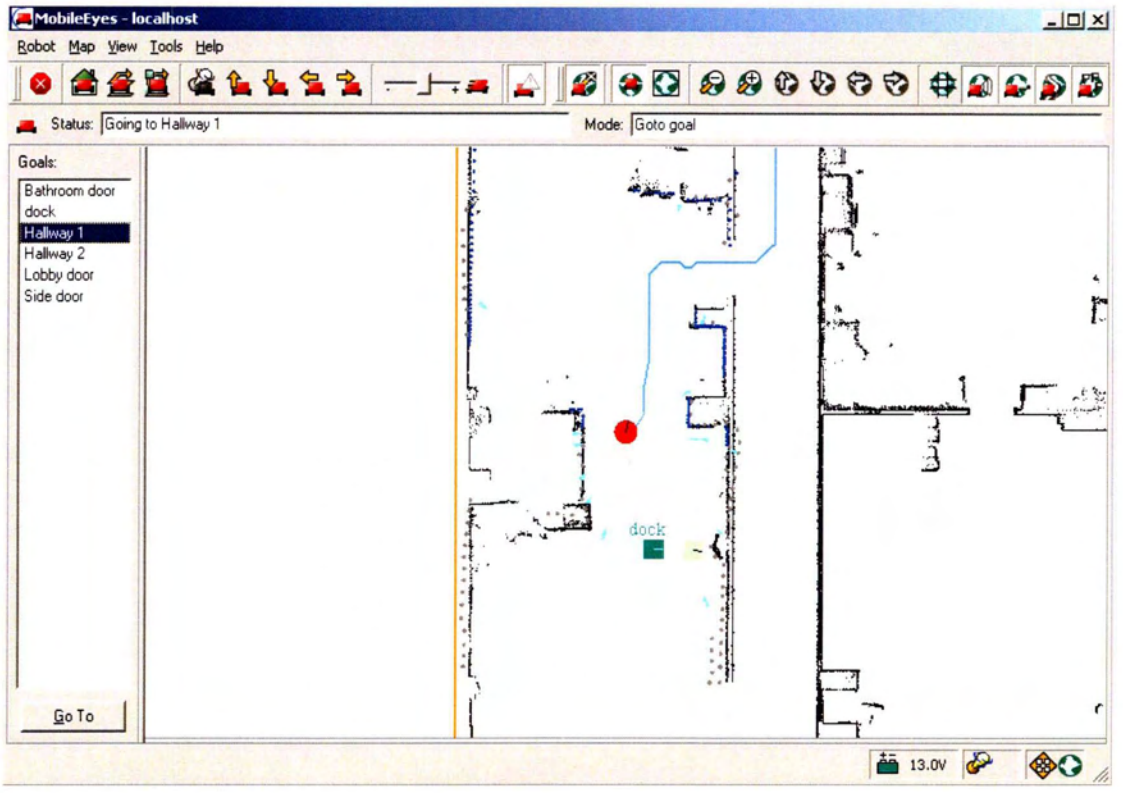
Στα εγκατεστημένα debian Linux του Pioneer P3-Dx το SONarnl δεν χρειάζεται εγκατάσταση και το αντίστοιχο εικονίδιο βρίσκεται στην επιφάνεια εργασίας. Σε αντίθετη περίπτωση για να εκτελεστεί το λογισμικό σε άλλο υπολογιστή, θα πρέπει να εγκατασταθούν τα Mobile Eyes, Mobile Sim και SONarnl. Ο SONarnl server χρησιμοποιεί αρχεία παραμέτρων, που καθορίζουν τα χαρακτηριστικά λειτουργίας, για επιτευχθεί η σύνδεση με το ρομπότ και τα εξαρτήματα του

Επίσης και το Mobile Eyes μπορεί να στείλει εντολές στο SONarnl για να αλλάξουν οι παράμετροι λειτουργίας του. Τέλος το SONarnl χρειάζεται ένα χάρτη στον οποίο θα καθορίζεται ο χώρος που θα κινείται το ρομπότ, αλλά και θα ορίζονται οι στόχοι και οι προορισμοί του.

Καθώς φορτώνει ο server στην οθόνη εμφανίζονται κάποιες χρήσιμες πληροφορίες, όπως ο τύπος του ρομπότ, πληροφορίες δικτύωσης κ.λ.π. Η έναρξη του Mobile Eyes γίνεται απλά από τα Windows με την εντολή start:programs:mobilerobots:mobileeyes:mobileeyes από ή από cd /usr/local/ mobileeyes/bin/ mobileeyes. Στα Linux γίνεται με χρήση τερματικού.

Μόλις αρχίσει το πρόγραμμα, ζητείται να καθοριστεί η διεύθυνση IP ή το όνομα του υπολογιστή, στον οποίο εκτελείται το SONarnl server. Συνήθως το όνομα που χρησιμοποιείται είναι local host, αν ο server εκτελείται στον ίδιο υπολογιστή με το Mobile Eyes. Στα Windows, με την έναρξη του προγράμματος το Mobile Eyes, υποδεικνύει ότι δεν έχει φορτωθεί χάρτης με τη σύνδεση του ρομπότ με τον server και δίνει τη δυνατότητα ο χρήστης να φορτώσει ένα χάρτη. Η επιλογή του χάρτη αποθηκεύεται σε ένα αρχείο, που ονομάζεται SONarnl.p για μελλοντικές χρήσεις.

Κατά την αρχική σύνδεση, ο SONarnl server στέλνει τον χάρτη που έχουμε επιλέξει στο Mobile Eyes, ο οποίος εμφανίζεται με μαύρες γραμμές, γύρω από ένα κόκκινο εικονίδιο, που αντιπροσωπεύει το ρομπότ και τη θέση του στο χάρτη. Οι διαστάσεις του ρομπότ και του περιβάλλοντος χάρτη στο Mobile Eyes είναι ανάλογες με τις πραγματικές (βλέπε εικόνα 4.2.). Άλλα χαρακτηριστικά του ρομπότ, όπως για παράδειγμα η μέγιστη ταχύτητα περιστροφής, βρίσκονται στο αρχείο P3dXP και από εκεί διαβάζονται από το Mobile Eyes καθώς και από το μενού του προγράμματος στο Tools (π.χ. μπορεί να γίνει αλλαγή χάρτη από Tools: Robot Configuration), ενώ τα μπλε τρίγωνα γύρω από το ρομπότ αντιπροσωπεύουν τα sonars.



Εικόνα 4.2. Το λογισμικό MobileEyes

4.3. Χαρτογράφηση.

Εφόσον το ρομπότ έχει περιηγηθεί στον χώρο, μπορεί με ευκολία να γίνει μια χαρτογράφηση (mapping), με την βοήθεια του mappper3 [26]

Το παράθυρο του Mapper3 αποτελείται από έναν χάρτη, ο οποίος είναι σκαναρισμένος από το ρομπότ, την γραμμή εργαλείων και περιέχει διάφορα χρήσιμα εργαλεία για την επαρκή επεξεργασία των χαρτών.

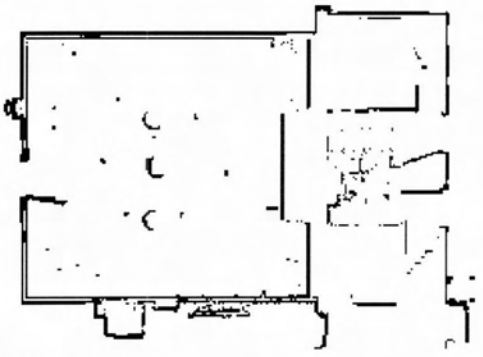
Για να ανοίξει ο χρήστης ένα αρχείο χάρτη, πρέπει να μεταβεί στο file open μενού ή από το αντίστοιχο κουμπί γραμμής εργαλείων και από εκεί να γίνει η επιλογή του, προς επεξεργασία, αρχείου. Το μενού File περιέχει επίσης στοιχεία για να γίνει η αποθήκευση του χάρτη καθώς και κατάλογο των πρόσφατα επεξεργασμένων χαρτών. Αντίστοιχη διαδικασία ακολουθείται και για τα αρχεία μορφής .2d, που παράγονται μετά από το σκανάρισμα κάποιου χώρου. Μεγέθυνση και σμίκρυνση του χάρτη για καλύτερη εποπτεία γίνεται περιστρέφοντας τον τροχό του ποντικιού.

Είναι δυνατή η εισαγωγή άλλων χαρτών στον τρέχοντα χάρτη. Αυτό γίνεται αν χρειάζεται να συνδυαστούν πολλαπλές σαρώσεις σε ένα μεγαλύτερο χάρτη, ή να αντικατασταθεί μια κακή σάρωση του χάρτη της περιοχής με καλύτερα δεδομένα. Αυτό γίνεται με την επιλογή του Insert Map από το μενού και ο νέος

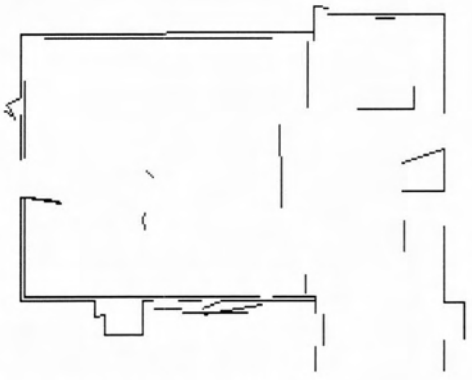
χάρτης θα εμφανιστεί πάνω στον παλιό, με ένα ημιδιαφανές μπλε φόντο, μαζί με μια γραμμή με εργαλεία.

Η μετατροπή μιας σάρωσης ξεκινάει με το άνοιγμα του σαρωμένου 2d αρχείου από το μενού open ... item και έπειτα στο file μενού, ή με στο κουμπί open, που βρίσκεται στην γραμμή εργαλείων. Με τον τρόπο αυτό θα εμφανιστούν τα εργαλεία σάρωσης καθώς και η γραμμή εργαλείων. Εάν η σάρωση δεν ξεκινήσει αυτόματα, μπορεί να γίνει και από το κουμπί start και έπειτα Scan Tools. Το Mapper επεξεργάζεται τα πρωτογενή δεδομένα σάρωσης από την πορεία του ρομπότ Όταν τελειώσει η διαδικασία, εμφανίζεται η save as επιλογή, η οποία επιτρέπει την αποθήκευση ενός νέου αρχείου χάρτη. Η διαδικασία παίρνει τέλος με το κουμπί "finish" στη γραμμή εργαλείων.

Οι περισσότερες λειτουργίες μπορούν να αναιρεθούν με την επιλογή Undo από το μενού Edit ή χρησιμοποιώντας το αντίστοιχο κουμπί στη γραμμή εργαλείων και κάθε στιγμή ο χρήστης μπορεί να παύσει και να συνεχίσει την επεξεργασία της σάρωσης με το κουμπί pause.



Εικόνα 4.3. Χάρτης με γραμμές.



Εικόνα 4 Χάρτης με σημεία.

4.4. Δυνατότητες με τη βοήθεια της κάμερας.

Με τη βοήθεια της ενσωματωμένης pan-tilt-zoom κάμερας, στο πάνω μέρος του ρομπότ και βεβαίως με χρήση ανάλογων προγραμμάτων, ο εκάστοτε χρήστης μπορεί να εκμεταλλευτεί της δυνατότητες που δίνει η όραση του pioneer αναλόγως των σκοπών της κάθε εφαρμογής.

Εικόνα 4.5. Η κάμερα του ρομπότ

4.4.1. Εικόνα μέσω του διακομιστή SAV

Για να μπορεί ο χρήστης να πάρει απλώς εικόνα από την ενσωματωμένη ptz κάμερα του ρομπότ στον απομακρυσμένο υπολογιστή του, τότε, με την προϋπόθεση ότι ο διακομιστής είναι εγκατεστημένος και στους δύο υπολογιστές, αρχίζοντας τον savClient πρέπει να βάλει στο hostname την διεύθυνση ip του ρομπότ.

4.4.2. Χειρισμός κάμερας μέσω του ARIA

Όπως προαναφέρθηκε στο demo του Aria, υπάρχει η ¨λειτουργία κάμερας¨ (camera mode), μέσα από την οποία ο χρήστης μπορεί να περιστρέψει την κάμερα, χρησιμοποιώντας τα βέλη του πληκτρολογίου του, από έναν απομακρυσμένο υπολογιστή.

4.4.3. Χειρισμός κάμερας μέσω του ACTS

Το λογισμικό ACTS, που βρίσκεται εγκατεστημένο στο ρομπότ, είναι το κύριο μέσο από το οποίο μπορεί το ρομπότ να αναγνωρίζει χρώματα και αντικείμενα και κατ΄ επέκταση ένα πολύ σημαντικό εργαλείο για να μπορούν να δημιουργηθούν κώδικες που εκμεταλλεύονται την όραση του Pioneer. Το

ACTS μέσα από την κονσόλα EZ-TRAIN έχει την δυνατότητα να εντοπίσει χρώματα από μια εικόνα ή από ένα ζωντανό βίντεο (και συγκεκριμένα αναλύοντας τα σε κουκίδες που ονομάζονται και blobs) και να τα αποθηκεύσει ως αρχεία κατάληξης .lut (look up table). Τα αρχεία αυτά αποθηκεύονται σε συγκεκριμένα ¨κανάλια¨ (chanels) από όπου μπορεί να υποστούν αργότερα επεξεργασία μέσω κατάλληλων ρυθμίσεων (configurations), σύμφωνα με τις επιθυμίες του χρήστη.

Παρακάτω, μέσα από έναν οδηγό εφτά (7) βημάτων, περιγράφεται η μεθοδολογία εντοπισμού και παρακολούθησης έγχρωμων αντικειμένων ή όπως αποκαλείται ¨εκπαίδευση¨ (training). [27]

Βήμα 1° : Εκκίνηση του ACTS.

Εκκίνηση του ACTS στην εγγενή του μορφή, στην οποία είναι συνδεδεμένη με τον πελάτη (client) του EZ-train με το live βίντεο, τις αρχικές ρυθμίσεις (configurations) και τα .LUT αρχεία.

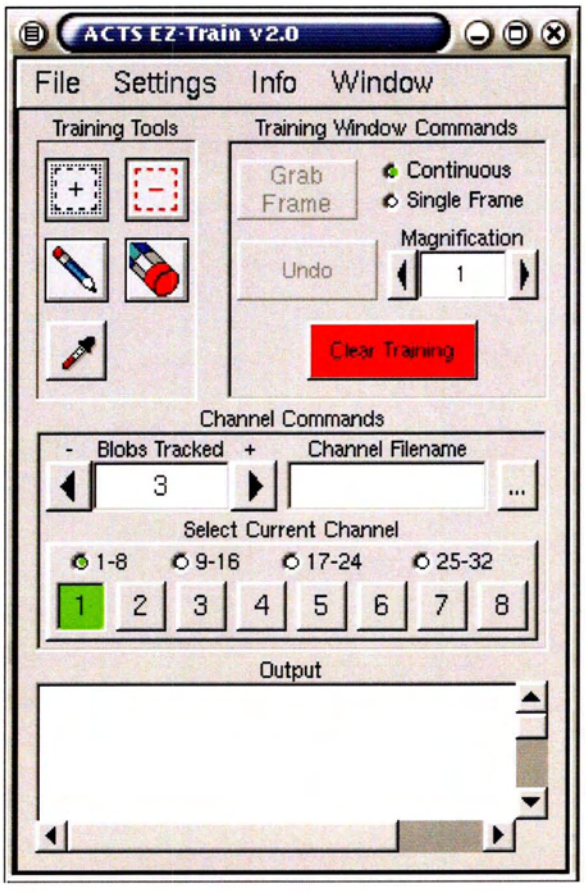
Για τα Linux εκκίνηση γίνεται από το terminal: cd/usr/local/ACTS/bin/acts

Για τα Windows με διπλό κλικ από το acts.exe στον προορισμό C:/Program Files/ActiveMedia Robotics/ACTS/bin.

Με την έναρξη του ACTS δύο παράθυρα θα ανοίξουν στην οθόνη, η κονσόλα EZ-train, ένα παράθυρο με εικόνα από την κάμερα (ή static image) και στην περίπτωση των Windows κι ένα τρίτο παράθυρο, που περιέχει μηνύματα συστήματος.

Βήμα 2° : Επιλογή μιας δοκιμαστικής εικόνας.

Αντί να χρησιμοποιηθεί βίντεο από την κάμερα για τα αντικείμενα στο χώρο, το οποίο δεν προσφέρει τόσο ευδιάκριτα χρώματα, μπορεί να χρησιμοποιηθεί μια στατική εικόνα, για να αρχίσει η διαδικασία εντοπισμού. Στα debian Linux του Robot PC, οι εικόνες αυτές είναι της μορφής .ppm. Η φόρτωση της εικόνας γίνεται από το μενού file της κονσόλας EZ-train. Η ανάλυση του παραθύρου είναι 160 οριζόντια και 120 κάθετα pixels αλλά με κατάλληλη μεγέθυνση το μέγεθος αυξάνεται έξι φορές.



Εικόνα 4.6. Η κονσόλα EZ-TPAIN

Βήμα 3° : Επιλογή καναλιού και αριθμός των blobs.

Ανάλογα με τα προς επιλογή αντικείμενα, επιλέγουμε και τον αριθμό των blobs, π.χ. αν σχεδιάζουμε να παρακολουθήσουμε δύο αντικείμενα αντίστοιχα και ο αριθμός των blobs πρέπει να είναι δύο. Η επιλογή του καναλιού (του χώρου στον οποίο αποθηκεύονται τα επιθυμητά χρώματα) γίνεται πατώντας το αντίστοιχο κουμπί του στην κονσόλα του EZ-train.

Βήμα 4° : Επιλογή αντικειμένου.

Με κλικ στο εικονίδιο προσθήκης (add) στην κονσόλα προστίθενται χρώματα για το επιλεγμένο κανάλι. Στο παράθυρο της εικόνας κρατώντας πατημένο το αριστερό κουμπί του ποντικιού σύρεται το ποντίκι μέχρι να δημιουργηθεί ένα ορθογώνιο παραλληλόγραμμο μέσα στο προς παρακολούθηση αντικείμενο, περικλείοντας όσο πιο πολλά pixels του αντικειμένου είναι δυνατόν. Πρέπει να δοθεί προσοχή ώστε να μην επιλεγούν pixels εκτός του αντικειμένου. Με την απελευθέρωση του ποντικιού τα επιλεγμένα pixels προστί-

θενται στα χρώματα του τρέχοντος καναλιού. Για ευκολότερη επιλογή μεμονωμένων pixels μπορεί να γίνει αλλαγή του μεγέθους της εικόνας.

Βήμα 5°: Αφαίρεση χρωμάτων, αναίρεση και μορφές προβολής των δεδομένων.

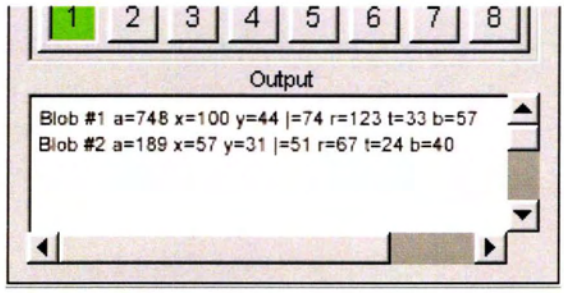
Αντίστοιχα με το εικονίδιο προσθήκης, στην κονσόλα με το εργαλείο αφαίρεσης, γίνεται επιλογή των χρωμάτων που δε θα συμπεριληφθούν στο κανάλι. Με το κουμπί της αναίρεσης (undo), η τελευταία πράξη που τροποποίησε τα δεδομένα στο κανάλι θα αντιστραφεί.

Χρησιμοποιώντας το δεξί πλήκτρο του ποντικιού μέσα στο παράθυρο το EZ-train (ή πατώντας το πλήκτρο 2, ενώ ο κέρσορας είναι μέσα στο παράθυρο), η μορφή προβολής αλλάζει και το πρόγραμμα εμφανίζει με λευκό τα ενδιαφέροντα pixels, ενώ τα υπόλοιπα εμφανίζονται με μαύρο χρώμα. Οι περιφέρειες των αντικειμένων τονίζονται με μοβ pixels. Πατώντας πάλι το ποντίκι ή το πλήκτρο 1 με τον κέρσορα μέσα στο παράθυρο, επανερχόμαστε στην αρχική εμφάνιση.

Εικόνα 4.7. THRESH MODE

Ενώ υπάρχουν κι άλλα ενδιαφέροντα blobs, εκτός από εκείνα στο προς παρακολούθηση αντικείμενο, το ACTS παρατηρεί μόνο τα blobs που αφορούν το αντικείμενο, αυτό διότι με την επιλογή των αντικειμένων (βήμα 4°), έχουν επιλεγεί τα συγκεκριμένα εικονοστοιχεία (pixels) και οι πολύ μικρές επιφάνειες

(με λιγότερα των δέκα εικονοστοιχείων) αγνοούνται από το ACTS. Στο πλαίσιο στο κάτω μέρος της κονσόλας υπάρχουν τα στατιστικά από τις παρακολουθούμενες επιφάνειες (π.χ. κέντρο μάζας).



Εικόνα 4.8. Στατιστικά στο κάτω μέρος της κονσόλας.

Βήμα 6° : Αποθήκευση δεδομένων καναλιού.

Με το πέρας της εκπαίδευσης (training) σε ένα συγκεκριμένο κανάλι, πρέπει να αποθηκευτεί το κανάλι (save) από το μενού file και να σωθεί το .LUT αρχείο (η προεπιλογή είναι channel1.LUT). Με την επιλογή save channel, επιλέγουμε διαφορετικό κατάλογο για την αποθήκευση.

Βήμα 7° : Δημιουργία ρυθμίσεων κατά την εκτέλεση.

Η επιλογή runtime configurations χρησιμοποιείται για να αποθηκευτεί το σύνολο των καναλιών και των παραμέτρων λειτουργίας τους. Συνήθως χρησιμοποιούνται για να καθορίσουν το λειτουργικό περιβάλλον για το ACTS με τον αντίστοιχο πελάτη-client. Στο μενού file υπάρχει επιλογή Save runtime configuration όπου, οι ρυθμίσεις που έγιναν παραπάνω, αποθηκεύονται στον υπολογιστή για μελλοντική χρήση.

ΚΕΦΑΛΑΙΟ 5. ΑΝΑΠΤΥΞΗ ΒΑΣΙΚΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ

5.1. Προγραμματισμός με ARIA

Αφού γραφτεί κάποιος κώδικας [28] (πάντα σε γλώσσα προγραμματισμού C++ και χρησιμοποιώντας κάποιο λογισμικό, όπως το Visual Studio της microsoft) μπορούν να παραχθούν εκτελέσιμα αρχεία. Αυτό επιτυγχάνεται αν γίνει compile ο κώδικας, μέσω της αντίστοιχης εντολής, και ο χρήστης μπορεί να βρει τα executables αρχεία στον φάκελο release (ή στο debug ανάλογα με τις ρυθμίσεις- configuration που έχουν γίνει).

Το Aria.dll [29] είναι η "βιβλιοθήκη" του SDK [30] (software development kit) του Aria, το οποίο είναι ένα σύνολο εργαλείων ανάπτυξης λογισμικού, που επιτρέπει την δημιουργία εφαρμογών. Ουσιαστικά, το .dll αυτό αρχείο, περιέχει τις μεθόδους επικοινωνίας με το ρομπότ. Έτσι για παράδειγμα, όταν θέλει ο χρήστης να δώσει εντολή στο ρομπότ να μηδενίσει την ταχύτητα του, δηλαδή Robot.SetVelocity(0), καλείται μια μέθοδος του Aria.dll. Η βιβλιοθήκη αυτή του Aria software δημιουργείται αυτόματα με την εγκατάσταση του ARIA-2.5.1 από το CD.

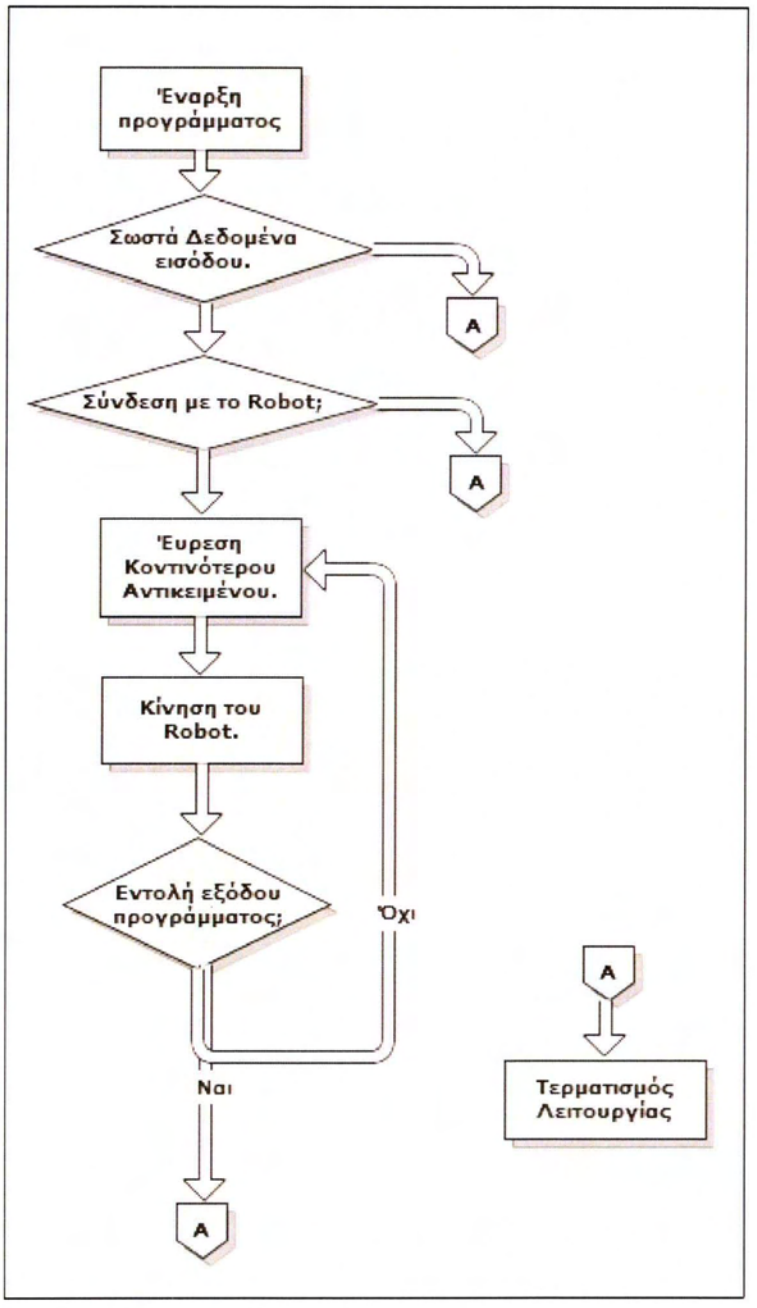
Έτσι, σε κάθε αρχείο κώδικα, είναι απαραίτητο από την αρχή να έχουμε καλέσει το Aria.dll με την εντολή #include Aria.h.

Όταν θελήσει ο χρήστης να τρέξει τα executables, είναι απαραίτητο το Arial.dll να βρίσκεται κάπου ώστε να το βλέπει η εφαρμογή, δηλαδή, είτε στον ίδιο φάκελο με την εφαρμογή, είτε σε κάποιον άλλον, όπου έχει όμως δοθεί στα windows εντολή να ψάχνουν γενικότερα.

5.2. Παραδείγματα δημιουργίας Κώδικα.

5.2.1. Κίνηση του ρομπότ.

Ο κώδικας που δημιουργήθηκε, είχε ως στόχο να δωθεί η εντολή στο ρομπότ να κινηθεί. Είναι εύκολο να κατανοηθεί η δομή του και οι επιμέρους εντολές του [31] με την χρήση του διαγράμματος ροής, που παρουσιάζεται στην ακόλουθη εικόνα 5.1. Το διάγραμμα ροής ακολουθεί ο κώδικας, που δίνει εντολή για την κίνηση του ρομπότ.



Εικόνα 5.1. Δομικό διάγραμματα για τον κώδικα κίνησης του ρομπότ

```
#include "Aria.h"
#include <stdio.h>
#include <iostream>
#include <time.h>
using namespace std;


int main(int argc, char **argv)
```

```
{
        ArRobot robot;
        ArKeyHandler keyHandler;
        ArSonarDevice sonarDev;
        ArSick sick;
        ArVCC4 ptz(&robot);
        Aria::init();
        bool ptzInitialized;
        ptzInitialized = ptz.init();
        ArUtil::sleep(4000);
        ArSimpleConnector connector(&argc, argv);
        if (!connector.parseArgs() || argc > 1)
        {
                connector.logOptions();

                return 1;
        }

        if (!connector.connectRobot(&robot))
        {
                printf("Could not connect to robot... exiting\n");
                Aria::shutdown();
                return 1;
        }

        Aria::setKeyHandler(&keyHandler);
        robot.attachKeyHandler(&keyHandler);
        robot.addRangeDevice(&sonarDev);
        robot.addRangeDevice(&sick);
         robot.runAsync(true);
        robot.enableMotors();
        int a = 0;

        robot.move(400);
```

```
  robot.setRotVel(-100);

  robot.waitForRunExit();
  Aria::shutdown();
return 0;
}
```

Η int main(int argc, char **argv) είναι η βασική συνάρτηση, από όπου ξεκινάει το πρόγραμμα. Έπειτα ακολουθεί η απαραίτητη 'αρχικοποίηση' των μεταβλητών, όπου ο χρήστης δηλώνει-ορίζει οποιεσδήποτε μεταβλητές χρησιμοποιεί. Για παράδειγμα η ArRobot robot; είναι η μεταβλητή που χρησιμοποιείται για να επιτευχθεί επικοινωνία με το ρομπότ και είναι απαραίτητη να περιεληφθεί σχεδόν σε κάθε κώδικα. Έπειτα με τις παρακάτω κλάσεις αντίστοιχα γίνονται οι παρακάτω ενέργειες:

ArKeyHandler keyHandler; διαβάζονται δεδομένα από το πληκτρολόγιο

ArSonarDevice sonarDev; σημειώνονται οι πρόσφατες αναγνώσεις από τα σόναρ που διαθέτει το ρομπότ

ArSick sick; γίνεται η σύνδεση με την συσκευή laser

ArVCC4 ptz(&robot); ; για να επιτευχθεί έλεγχος της κάμερας του ρομπότ

```
bool ptzInitialized;            ελέγχει αν η κάμερα έχει αρχικοποιηθεί
ptzInitialized = ptz.init();    την αρχικοποιεί κ.ο.κ.
```

Έπειτα η εντολή ArSimpleConnector connector(&argc, argv); πραγματοποιεί μια σύνδεση με το ρομπότ, είτε μέσω θύρας TCP (για τον προσομοιωτή ή για το ρομπότ) ή με απευθείας σύνδεση σειριακής θύρας.

Με μια απλή εντολή if, διαπιστώνεται η σύνδεση ή όχι του ρομπότ με τον χρήστη, σε πιθανή αποτυχία σύνδεσης να βλέπει το μήνυμα Could not connect to robot... exiting στην οθόνη του.

```
if (!connector.parseArgs() || argc > 1)
  {
```

```
        connector.logOptions();


        return 1;
}

if (!connector.connectRobot(&robot))
{
        printf("Could not connect to robot... exiting\n");
        Aria::shutdown();
        return 1;
}
```

Η εντολή Aria::setKeyHandler(&keyHandler); είναι απαραίτητη για να υπάρξει και η εντολή ArKeyHandler keyHandler; ώστε να διαβάζονται τα δεδομένα από το πληκτρολόγιο και επομένως περιλαμβάνεται στον κώδικα. Αντίστοιχα απαραίτητη είναι και η robot.addRangeDevice, η οποία είναι βασική μεταβλητή ή αλλιώς τάξη – Class, όπως συναντάται συχνά, για όλες τις συσκευές ανίχνευσης (λέιζερ/σόναρ), που επιστρέφουν πληροφορίες από τη συσκευή.

Έπειτα με την κλάση robot.runAsync(true); αρχίζει να λειτουργεί το ρομπότ. Η παράμετρος true στην παρένθεση δηλώνει ότι, αν χαθεί η σύνδεση, ο βρόγχος τελειώνει.

Οι κινητήρες του ρομπότ τελικά ενεργοποιούνται (robot.enableMotors) και το ρομπότ κινείται με 400mm/sec και έχοντας γωνιακή ταχύτητα 100μμ/sec ανάλογα με την εντολή robot.move(400) ή robot.setRotVel(-100) αντίστοιχα. Τέλος, όταν ο χρήστης πατήσει escape, κλείνει το πρόγραμμα με τις παρακάτω εντολές:

```
robot.waitForRunExit();
Aria::shutdown();
```

5.2.2. Κίνηση της κάμερας.

Με την ίδια λογική, μπορεί να δημιουργηθεί ένα απλό πρόγραμμα, που έχει στόχο την κίνησης της ενσωματωμένης pan-tilt-zoom κάμερας στο ρομπότ και παρατίθεται παρακάτω.

```
#include "Aria.h"
#include <stdio.h>
#include <iostream>
#include <time.h>
using namespace std;


void ptzExercise(int inc, bool initPTZ, ArVCC4 *thisPTZ)
{
typedef enum
{
up_U,
down_D,
} VertDirection;
typedef enum
{
left_L,
right_R,
} HorizDirection;
int panAngle;
int tiltAngle;
int lowPan;
int lowTilt;
int highPan;
int highTilt;
HorizDirection hDir;
VertDirection vDir;

if (initPTZ == true)                                    {
panAngle = thisPTZ->getPan();
tiltAngle = thisPTZ->getTilt();
highPan = thisPTZ->getMaxPosPan();
lowPan = thisPTZ->getMaxNegPan();
```

```
highTilt = thisPTZ->getMaxPosTilt();
lowTilt = thisPTZ->getMaxNegTilt();
cout << "Pan = " << panAngle << " Tilt = " << tiltAngle << "\n";
(1)     if (panAngle == highPan && tiltAngle == highTilt)
{
cout << "Changing direction to L and D\n";
hDir = left_L;
vDir = down_D;
}
if (panAngle == lowPan && tiltAngle == lowTilt)
{
cout << "Changing direction to R and U\n";
hDir = right_R;
vDir = up_U;
}
if (hDir == right_R)
{
thisPTZ->panRel(inc);
}else
{
thisPTZ->panRel(-1*inc);
}
if (vDir == up_U)
{

thisPTZ->tiltRel(inc);
}else
{
thisPTZ->tiltRel(-1*inc);
}
}else
{
cout << "Cannot initialize camera\n";
}
```

```
}
int main(int argc, char **argv)
{
ArRobot robot;
ArKeyHandler keyHandler;
ArSonarDevice sonarDev;
ArSick sick;
ArVCC4 ptz(&robot);
Aria::init();
bool ptzInitialized;
ptzInitialized = ptz.init();
ArUtil::sleep(4000);
ArSimpleConnector connector(&argc, argv);
if (!connector.parseArgs() || argc > 1)
{
connector.logOptions();
return 1;
}
if (!connector.connectRobot(&robot))
{
printf("Could not connect to robot... exiting\n");
Aria::shutdown();
return 1;
}
Aria::setKeyHandler(&keyHandler);
robot.attachKeyHandler(&keyHandler);
robot.addRangeDevice(&sonarDev);
robot.addRangeDevice(&sick);
robot.runAsync(true);
robot.enableMotors();
int a = 0;
while(true){
ptzExercise( 5, ptzInitialized, &ptz);
}
```

```
robot.waitForRunExit();
Aria::shutdown();
```

Ο παραπάνω κώδικας, αν και πιο μακροσκελής από τον προηγούμενο, ακολουθεί παρόμοια δομή και αναλύοντας τον, μπορεί να γίνουν αντιληπτά τα επιμέρους τμήματα, από τα οποία αποτελείται.

Αρχικά με τις εντολές:

```
typedef enum
up_U,
down_D,
VertDirection;
```

καθορίζεται αν η κάμερα θα κινηθεί πάνω ή κατω. Οπότε αντίστοιχα για να καθορίστει αν η κάμερα θα κινηθεί δεξιά η αριστερά:

```
 typedef enum
{
left_L,
 right_R,
```

Με την εντολή int panAngle; ορίζεται η μεταβλητή της γωνίας pan της κάμερας την συγκεκριμένη χρονική στιγμή, ενώ με την int tiltAngle η γωνία tilt αντιστοιχα. Οι ενολές που ακολουθούν (int lowPan; int lowTilt;, int highPan; , int highTilt;) θέτουν τα πάνω και κάτω όρια στις αντίσοιχες γωνίες. Η HorizDirection hDir ορίζει την τωρινή οριζόντια διεύθηνση που βρίσκεται η κάμερα και η VertDirection vDir; την κάθετη τωρινή διεύθηνσή της.

Αν η κάμερα έχει αρχικοποιηθεί σωστά (if (initPTZ == true)) μέσω της panAngle = thisPTZ->getPan(); ορίζεται η pan angle ως η τωρινή γωνία πάν και η tiltAngle = thisPTZ->getTilt(); ορίζει την τωρινή γωνία tilt. Έπειτα οι α-κραίες (μέγιστες-ελάχιστες γωνίες) αποθηκεύονται με τις παρακάτω εντολές:

```
highPan = thisPTZ->getMaxPosPan();
lowPan = thisPTZ->getMaxNegPan();
highTilt = thisPTZ->getMaxPosTilt();
lowTilt = thisPTZ->getMaxNegTilt();
```

Οι κινήσεις της κάμερας και οι αλλαγές κατεύθυνσης που θα πραγμα-τοποιήσει, βρίσκονται μέσα στον μεγάλο βρόχο (1).

Φυσικά όπως διακρίνεται κάτω από την βασική συνάρτηση και σε αυτόν τον κώδικα η αρχικοποίηση είναι παρόμοια και απαραίτητη και για την επικοινωνία με το ρομπότ (ArRobot robot;) αλλά και για να ορισθούν οι μεταβλητές των αισθητήρων της κάμερας κ.ο.κ.

π.χ. ArKeyHandler keyHandler;

ArSonarDevice sonarDev;

ArSick sick;

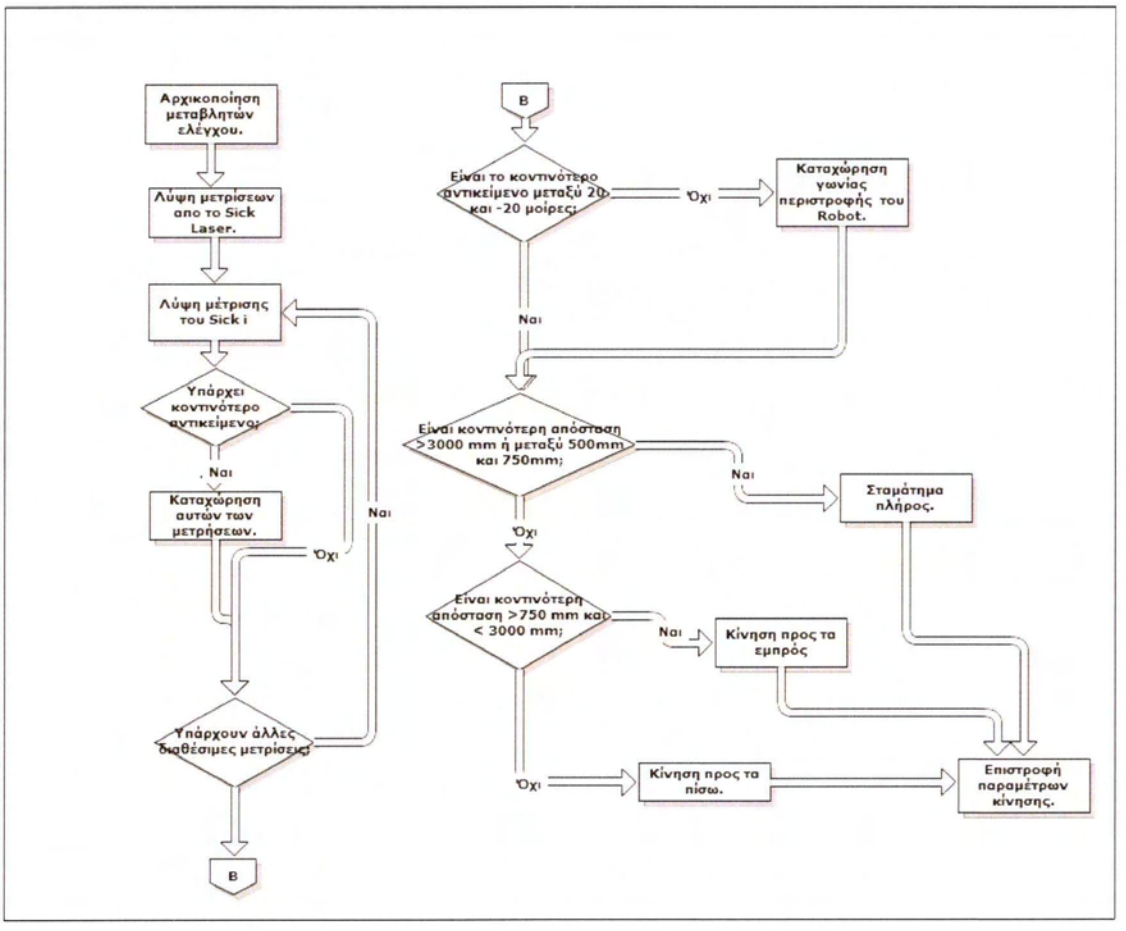
ArVCC4 ptz(&robot);

Επίσης παράμοια και απαραίτητη είναι η διάγνωση της σύνδεσης ή όχι με το ρομπότ που επιτυγχάνεται μέσα από τον παρακάτω βρόγχο:

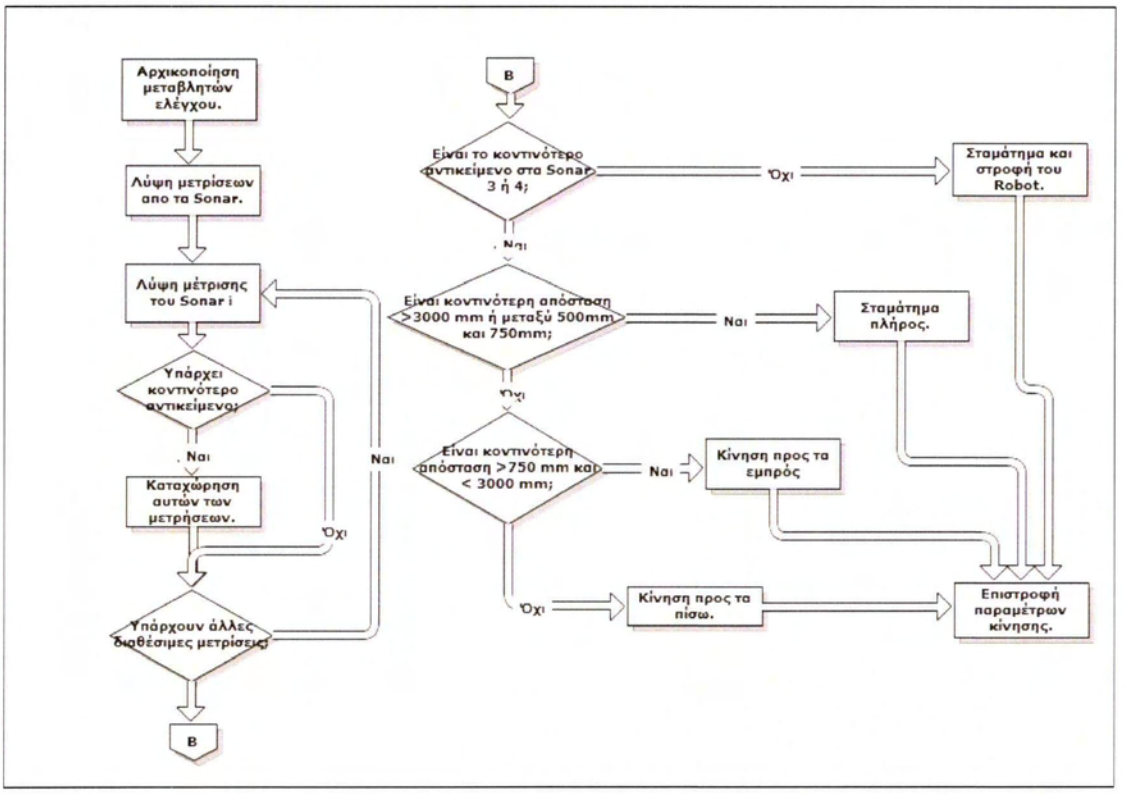
```
ArSimpleConnector connector(&argc, argv);
if (!connector.parseArgs() || argc > 1)
{
connector.logOptions();

return 1;
}

if (!connector.connectRobot(&robot))
{
printf("Could not connect to robot... exiting\n");
Aria::shutdown();
return 1;
}
]
```

5.3. Βασικοί κώδικες για εντοπισμό στόχου, μέσω των αισθητήρων της κάμερας.

5.3.1. Εντοπισμός στόχου και αποφυγή εμποδίων μέσω των αισθητήρων.

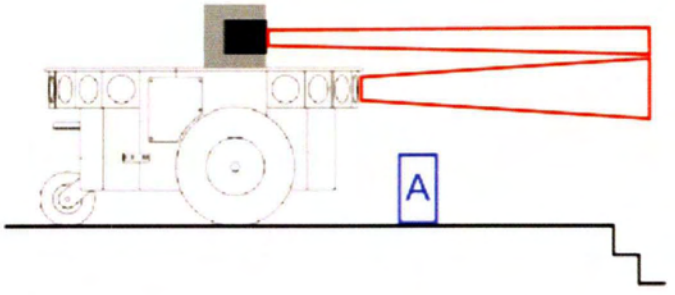


Εικόνα 5.2. Δομικό διάγραμμα για τον εντοπισμό στόχου μέσω λέιζερ.



Εικόνα 5.3. Δομικό διάγραμμα για εντοπισμό στόχου μέσω σόναρ

Αρχικά αναλύοντας τον πρώτο βρόγχο του κώδικα:



Εικόνα 5.4. Το κινητό ρομπότ μέσω των αισθητήρων του καταφέρνει να αποφύγει πιθανά εμπόδια.

```
void moveRobot(ArRobot* rob, int degrees , int move){
if(move==-1){
rob->setVel(-400);
}
else if(move==1){
rob->setVel(400);
```

```
}
else if (move==0){
rob->setVel(0);
}
if(degrees!=0){
rob->setDeltaHeading(degrees);
}
}
```

Η εντολή setVel λέει στο ρομπότ να προσαρμόσει την ταχύτητα του στην τιμη της παραμέτρου (-400, 400 και 0 αντίστοιχα, όπως φαίνεται παραπάνω) και ανάλογα να κινηθεί (move), ενώ η εντολή setDeltaHeading(degrees), προκαλεί περιστροφή του ρομπότ ανάλογα με τις δοθείσες μοίρες (γωνία).

Με το getdirection προσδιορίζεται ο προσανατολισμός του ρομπότ και αρχικοποιείται: int getdirection(ArRobot* rob,int &turn ,int &move), ενώ διατηρούνται και τα αποτελέσματα από τους αισθητήρες του ρομπότ:

```
ArSensorReading* sens;
```

Εδώ θα χρησιμοποιηθεί ένας φάκελος για τις μετρήσεις από τα σοναρ, ουσιαστικά ένα text αρχείο με αποτελέσματα μορφής, όπως και στον πίνακα που παρατίθεται στο τέλος του κώδικα. Στο εξαγόμενο αρχείο αναφέρονται η ώρα σε πραγματικό χρόνο (real time), οι αποστάσεις, που έχουν παρατηρηθεί από τα οχτώ σόναρ και τυπώνεται η μικρότερη τιμή της απόστασης από τα σόναρ:

```
 FILE *tmpfile = fopen("SonarTest.txt","a");
time_t rawtime = time(NULL);
fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));
int sonar = 0,range = 10000;
for(int i =0 ; i<8 ;i++){
sens = rob->getSonarReading(i);
int tmpRange = sens->getRange();
fprintf (tmpfile , "Sonar # %d Value %d \n",i,tmpRange);
if(tmpRange<range){
sonar = i;
```

```
range = tmpRange
fprintf (tmpfile , "Closest Sonar # %d Value %d \n",sonar,range);
```

Αντίστοιχα και για το laser:

```
int getsickDirections(ArSick *sic,int &turn ,int &move){
const std::list<ArSensorReading *>*readinglist;
std::list<ArSensorReading *>::const_iterator it;
int i =-1;
FILE *tmpfile = fopen("LaserTest.txt","a");
time_t rawtime = time(NULL);
fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));
int degrees = 0 , range =10000;
readinglist= sic->getRawReadings();
for (it = readinglist->begin(); it != readinglist->end(); it++)
{
i++;
int tmpRange = (*it)->getRange();
int tmpDegrees = (*it)->getSensorTh();
//fprintf (tmpfile , "Sonar # %d Value %d \n",i,tmpRange);
fprintf (tmpfile ,"%d LaserReading = %d Angle =
%d",i,tmpRange,tmpDegrees);
if(tmpRange<range){
degrees = tmpDegrees;
range = tmpRange;
```

Τώρα στον επόμενο μεγάλο βρόχο, ανάλογα με την απόσταση που έχει το ρομπότ μας από το στόχο (που εκφράζεται με την μεταβλητή range) υπάρχουν τρείς περιπτώσεις

A.

```
if(range==5000){
  turn = 0;
move=0;
}
```

```
        else {
      if(sonar==0){
            turn = 90;
            move = 0;
              ..... κ.ο.κ

              else{
B.          if((range>2000)&(range<3000)){
                  turn = 0;
                  move = 0;
            }
Γ.          else if(range<2000){
                  turn = 0;
                  move = -1;
            }
            else{
                  turn = 0;
                  move = 1;
            }
      }
}
      fclose(tmpfile);
return 0;
```

Η λογική είναι ότι, έχοντας ορίσει μια απόσταση ασφαλείας από 2000-3000 χιλιοστά, το ρομπότ εφ΄όσον απέχει από τον στόχο την παραπάνω απόσταση δεν θα κινηθεί καθόλου. Στην περίπτωση όπου η απόσταση είναι πάνω από 5000 μονάδες, το ρομπότ θα κινηθεί προς τα μπρος, με σκοπό να πλησιάσει το στόχο, ενώ για απόσταση μικρότερη των 2000 μονάδων το ρομπότ θα κατευθυνθεί προς τα πίσω, για να αποφευχθεί οποιαδήποτε πιθανή σύγκρουση. Οι μονάδες μέτρησεις, που χρησιμοποιήθηκαν στην συγκεκριμένη περίπτωση, είναι τα χιλιοστά.

Αν ο στόχος ξεφεύγει από τους μπροστινούς αισθητήρες για πάνω από είκοσι μοίρες δεξιά αριστερά, τότε το ρομπότ θα στρίψει για να τον ακολουθήσει:

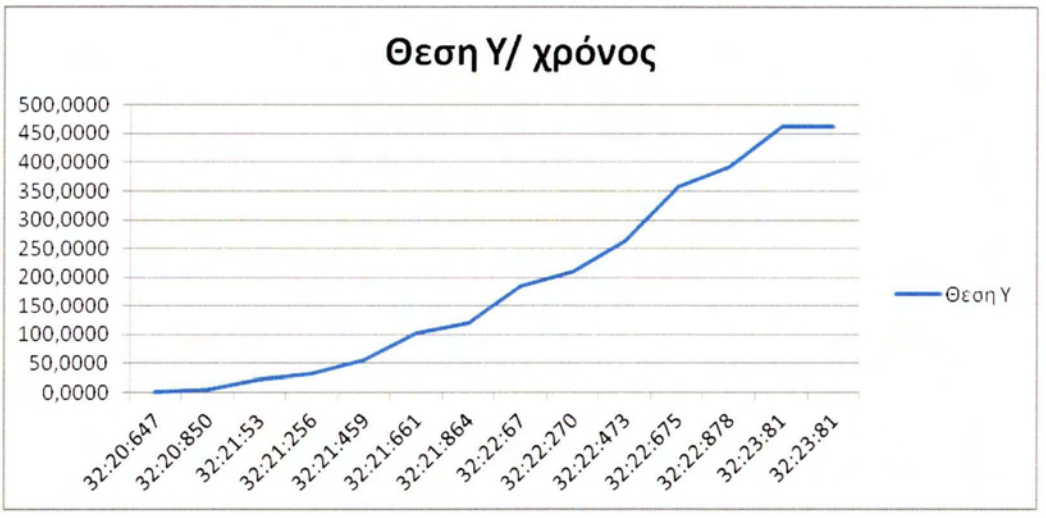
```
        if((degrees<-20)||(degrees>20)){
                turn = degrees;
        }
        else{
                turn = 0;
        }

        if((range>2000)&(range<3000)){
                move = 0 ;
        }
        else if(range<2000){
                move = -1;
        }
        else{
                move = 1;
        }
        fclose(tmpfile);
        return 0;
}
```

Το παραγόμενο από τον κώδικα αρχείο .txt εμφανίζει τις τιμές με τις αποστάσεις από τα οχτώ σόναρ του ρομπότ σε σχέση με τα πιο κοντινά αντικείμενα στο περιβάλλον. Τα αποτελέσματα έχουν την μορφή που παρουσιάζεται στον παρακάτω πίνακα.

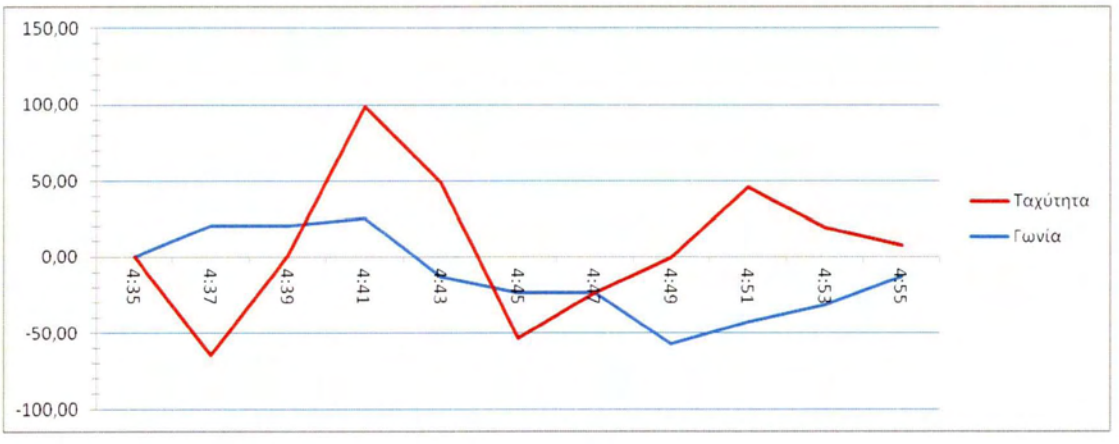
Sonar # 0 Value 1325
Sonar # 1 Value 786
Sonar # 2 Value 589
Sonar # 3 Value 499
Sonar # 4 Value 499
Sonar # 5 Value 497
Sonar # 6 Value 535

```
Sonar # 7 Value 683
Closest Sonar # 5 Value 497
Current Time Mon Jul 11 14:21:42 2011

Sonar # 0 Value 1363
Sonar # 1 Value 1083
Sonar # 2 Value 944
Sonar # 3 Value 935
Sonar # 4 Value 846
Sonar # 5 Value 1013
Sonar # 6 Value 787
Sonar # 7 Value 771
Closest Sonar # 7 Value 771
Current Time Mon Jul 11 14:21:43 2011

Sonar # 0 Value 1161
Sonar # 1 Value 1244
Sonar # 2 Value 1198
Sonar # 3 Value 1244
Sonar # 4 Value 977
Sonar # 5 Value 903
Sonar # 6 Value 964
Sonar # 7 Value 1833
Closest Sonar # 5 Value 903
Current Time Mon Jul 11 14:21:44 2011 κοκ...
```

Εικόνα 5.5. Μεταβολή της x θέσης ως προς τον χρόνο



Εικόνα 5.6. Μεταβολή της y θέσης ως προς τον χρόνο



Εικόνα 5.7. Πληροφορίες από τα sonar του ρομπότ

5.3.2. Εντοπισμός στόχου και αποφυγή εμποδίων μέσω κάμερας.

Το επόμενο πρόγραμμα που δημιουργήθηκε, έχει στόχο να εντοπίσει κάποιον υποψήφιο στόχο, μέσω τον εντοπισμό των χρωμάτων που αυτός αποτελείται και χρησιμοποιώντας την κάμερα, που διαθέτει η πλατφόρμα. Προϋπόθεση του κώδικα είναι το πρόγραμμα ACTS να είναι ανοιχτό στον ενσωματωμένο ηλεκτρονικό υπολογιστή ρομπότ, καθώς μέσω αυτού του προγράμματος θα γίνει η παρακολούθηση του στόχου. Ο κώδικας αυτός, όπως και ο προηγούμενος, υπάρχει αναλυτικά στο τελευταίο κεφάλαιο της εργασίας.

Η εντολή class Follow : public ArAction //Class Decleration φτιάχνει την συγκεκριμένη κλάση (ή αλλιώς class όπως αναφέρθηκε και παραπάνω) τύπου aractions, το οποίο χρησιμοποιείται από το λογισμικό ARIA για να μπορεί ο καθένας να φτιάξει τα δικά του classes και να δηλώσει την μορφή τους (declaration). Το ArAction συγκεκριμένα είναι η βασική κλάση που χρησιμοποιείται για εντολές κίνησης π.χ. εδώ έχουμε ArActionDesired *fire(ArActionDesired currentDesired); ένα event όπως λέγεται και σε όρους προγραμματισμού

```
{
public:
```

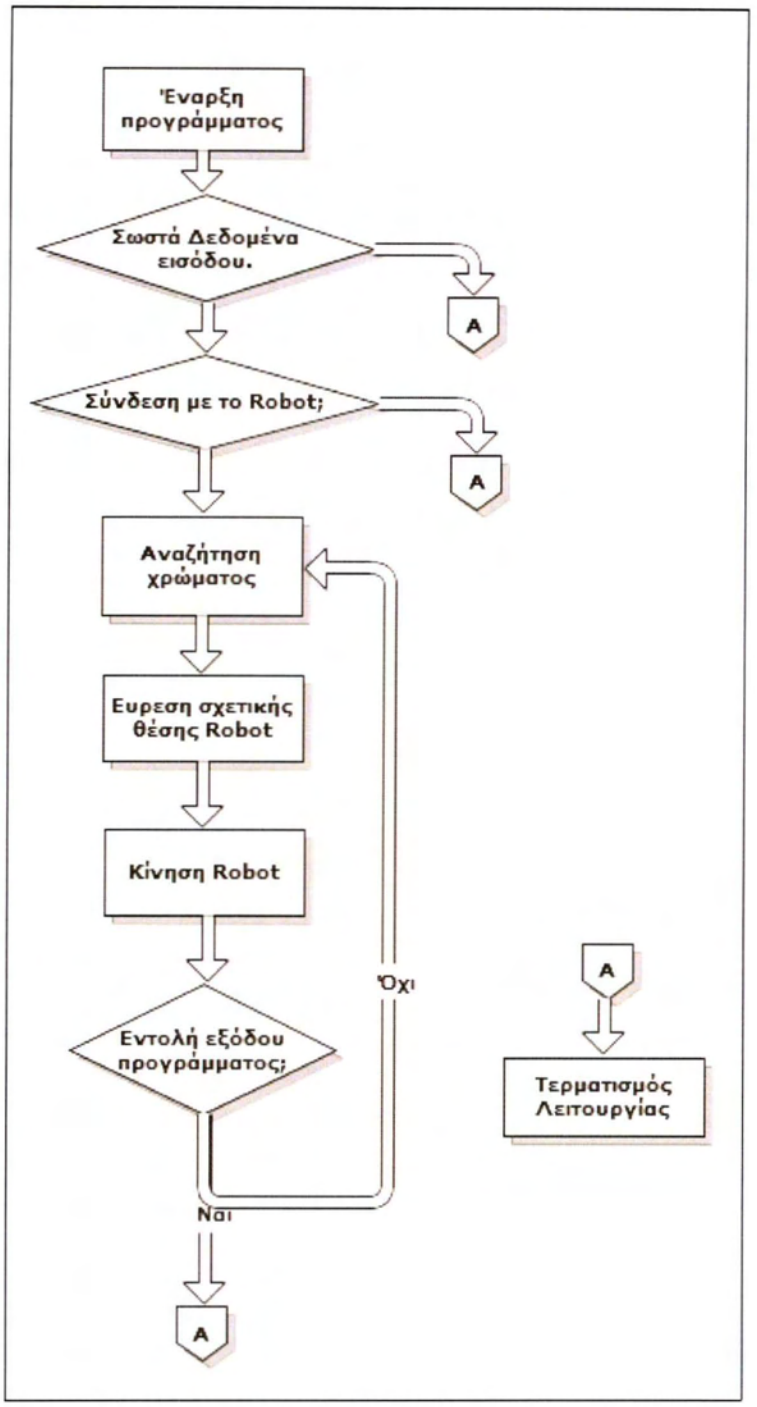
Η εντολή enum State αφορά την εσωτερική κατάσταση του ρομπότ

```
  NO_TARGET,
  TARGET,
 };
```

Η εντολή Follow(ArACTS_1_2 *acts, ArVCC4 *camera); λέγεται και constructor ενώ η ~Follow(void); destructor. Το ArACTS_1_2 *myActs; στην παρένθεση είναι ο driver για το ACTS το οποίο είναι απαραίτητο για τον κώδικα και ArVCC4 *myCamera; Αντίστοιχα για την κάμερα

Ορίζουμε το κανάλι bool setChannel(int channel); και οι διαστάσεις τις εικόνας ως εξής:

```
  WIDTH = 160,
  HEIGHT = 120;
```



Εικόνα 5.8. Διάγραμμα ροής για τον κώδικα εντοπισμού στόχου μέσω της κάμερας

Με άλλα λόγια με την εντολή Follow ωθείται το ρομπότ να παρακολουθήσει το μεγαλύτερο blob (κόκκος χρώματος) και επομένως εισάγουμε στο πρόγραμμα την εντολή Follow::Follow(ArACTS_1_2 *acts, ArVCC4 *camera). Μέσα στο follow υπάρχουν αυτά που είχαμε αρχικοποιήσει την πρώτη φορά και αφου χρειάζεται να αρχικοποιήσουμε επιπλέον και ένα πρώτο κανάλι για τον εντοπισμό του στόχου χρησιμοποιούμε την εντολή int myChannel.

Επίσης η εντολή ArTime myLastSeen αφορά το πότε το ρομπότ είδε τελευταία φορά κάποιο αντικείμενο, καθώς απαρσίτητος είναι κι ο μέγιστος χρόνος από τότε που το είδε (η αντίσοιχη εντολή όπως φαίνεται και στον κώδικα του παραρτήματος είναι int myMaxTime)

Όταν τελειώσει η διαδικασία το ρομπότ καλείτε να μην κάνει απολύτως τίποτα και χρησιμοποιείται ένας destructor, όλες οι classes πρέπει να έχουν έναν destructor (εντολή Follow::~Follow(void) {}//Destructor)

Ακολουθεί το κύριο μέρος του κώδικα όπου εισάγονται ο αριθμος των επιθημητών κουκίδων χρωμάτων (blobs) η προς παρακολούθηση περιοχή blob area οι άξονες κτλ:

```
ArActionDesired *Follow::fire(ArActionDesired currentDesired)/{
 ArACTSBlob blob;
 ArACTSBlob largestBlob;
 bool flag = false;
 int numberOfBlobs;
 int blobArea = 10;
 double xRel, yRel;
 myDesired.reset();
```

Επίσης χρειάζεται η εντολή numberOfBlobs = myActs->getNumBlobs(myChannel); όπου βάση του ACTS πόσα blobs έχουν βρεθεί στο συγκεκριμένο κανάλι

Έτσι αν ο αριθμός των blobs είναι 0 (if(numberOfBlobs != 0)), τότε για κάθε blob από το 0 ως τον μεγαλύτερο αριθμό, το πρόγραμμα ξεκινάει και παίρνει το πρώτο blob και ο βρόχος που ακολουθεί [if(blob.getArea() > blobArea)] blob area χρειάζεται για να μην παίρνει το πρόγραμμα μικρά αντικείμενα μέσα στην περιοχή. Στη συνέχεια το πρόγραμμα κρατάει τον μεγαλύτερο κόκκο χρώματος (largestBlob = blob;) και με την εντολή myLastSeen.setToNow(); την στιγμή που βλέπει το robot το μεγαλύτερο blob.

Στην συνέχεια ακολουθει έταιρος βρόχος που αφορά καθαρά τον στόχο.

Αν έχει περάσει πάνω από ένα δευτερόλεπτο έχουμε αντίστοιχα:

Σε περίπτωση που χάθηκε ο στόχος τότε το διαγνωστικό μύνημα"Target Lost" θα εμφανιστεί στην οθόνη:

```
if(myState != NO_TARGET) ArLog::log(ArLog::Normal, "Target Lost");
    myState = NO_TARGET;
```

Ενώ, σε περίπτωση που ο στόχος είναι στο πεδίο της κάμερας, ο χρήστης ενημερώνεται αντίστοιχα, όπως φαίνεται παρακάτω :

```
if(myState != TARGET) ArLog::log(ArLog::Normal, "Target Aquired");
myState = TARGET;
```

Στην περίπτωση ενός καινούργιου στόχου το πρόγραμμα ξεκινάει πάλι υπολογισμούς για την σχετική του θέση, με συντεταγμενες xRel για το πλάτος και yRel για το ύψος.

Στον επόμενο βρόχο ξεκαθαρίζεται αν θα γυρίσει η κάμερα πάνω κάτω ως εξής:

```
if(!(ArMath::fabs(yRel) < .20))
  if (-yRel > 0)
    myCamera->tiltRel(1);
  else
    myCamera->tiltRel(-1);
}
```

```
if (ArMath::fabs(xRel) < .10)
    {
```

το myDesired.setDeltaHeading(0); deltaheading αφορά την κατεύθυνση βάση της παρούσας θέσης, πάντα έχοντας υπόψη ότι το setheading είναι βάση της αρχικής θέσης

if (ArMath::fabs(-xRel * 10) <= 10) για μικρότερο των δέκα μοιρών στρίβει λίγο

```
myDesired.setDeltaHeading(-xRel * 10);
```

else if (-xRel > 0) αν θα πάει δεξιά ή αριστερά και στην παρένθεση οι μοίρες

```
            myDesired.setDeltaHeading(10);
          else
            myDesired.setDeltaHeading(-10);

        }

    myDesired.setVel(200); θέτει ταχύτητα στο 200
    return &myDesired;
   }
  else   (σε αυτήν την περίπτωση δεν υπάρχει στόχος άρα σταματάει).
   {
    myDesired.setVel(0);
    myDesired.setDeltaHeading(0);
    return &myDesired;
   }
}

bool Follow::setChannel(int channel)
{
 if (channel >= 1 && channel <= ArACTS_1_2::NUM_CHANNELS)
 {
  myChannel = channel;
  return true;
 }
 else
  return false;
}
```

Ακολουθεί η βασική συνάρτηση από όπου ξεκινάει το πρόγραμμα

```
int main(int argc, char** argv)
```

με την απαραίτητη αρχικοποίηση των σόναρ του ACTS κτλ

```
 Aria::init();
 ArRobot robot;
 ArKeyHandler keyHandler;
```

```
ArSonarDevice sonar;
ArVCC4 vcc4 (&robot);
ArACTS_1_2 acts;
ArArgumentParser argParser(&argc, argv);
argParser.loadDefaultArguments();
```

Η προσπάθεια για σύνδεση γίνεται με τον ίδιο τρόπο όπως έχει αναφερθεί και παραπάνω στα απλούστερα παραδείγματα:

```
ArSimpleConnector simpleConnector(&argParser)   si;
   if (!Aria::parseArgs())
{
  Aria::logOptions();
  keyHandler.restore();
  Aria::shutdown();
  return 1;
}
```

Ενώ με την εντολή aractionlimiter περιορίζεται το ρομπότ σχετικά με την κίνηση, όπως φαίνεται και στις παρακάτω εντολές που χρησιμοποιήθηκαν, ArActionLimiterForwards limiter("speed limiter near", 300, 600, 250); κόβει ταχύτητα στα 30 εκ και σταματάει στα 25 κ.λ.π.

Η μεταβλητή ArActionConstantVelocity stop("stop", 0); αφορά την ταχύτητα σταματήματος, η ArActionConstantVelocity backup("backup", -200); ταχύτητα για όπισθεν κ.ο.κ.

Υπόψιν ότι το acts δουλεύει εσωτερικά με κάποια ports και ανοίγουμε το port για να πάρουμε πληροφορίες μέσω της εντολής acts.openPort(&robot);
ενώ με την vcc4.init(); γίνεται initialization στην κάμερα.

Το ArUtil::sleep(1000); ;; είναι ένας τυπικός χρόνος καθυστέρησης, όπου το ρομπότ περιμένει για να μην γίνει κανένα λάθος, γιατί δεν έχει προλάβει να ξεκινήσει.

Όριζεται η μέγιστη ταχύτητα στα 400 ως εξής:

```
robot.setAbsoluteMaxTransVel(400);
```

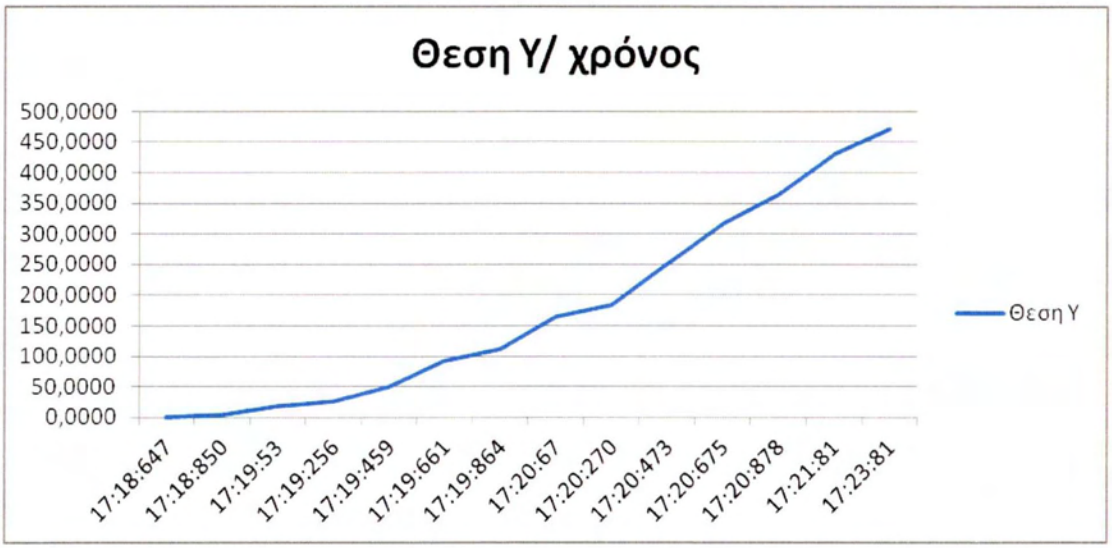
και προσθέτονται και τα απαραίτητα για την κίνηση actions ως εξής:

```
robot.addAction(&limiter, 100);
robot.addAction(&limiterFar, 99);
robot.addAction(&backwardsLimiter, 98); κοκ
```

Τέλος όσον αφορά τα εξαγώγιμα αρχεία, όπως βλέπουμε παρακάτω μέσα από την εντολή FILE *tmpfile = fopen("AriaCameraPositionTest.txt","a"); φτιάχνεται το output αρχείο και ξεκινάει να καταγράφεται η θέση η ταχύτητα και ο χρόνος του ρομπότ όσο τρέχει ο κώδικας.

```
while(true){
        double x = robot.getX();
        double y = robot.getY();
        double vel = robot.getVel();
        time_t rawtime;
        time(&rawtime);
        fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));
        fprintf (tmpfile , "Current Position X: %f Y: %f Velocity: %f \n\n",x,y,vel);
        int mseconds = 1000;
        clock_t goal = mseconds + clock();
        while (goal > clock());
}
fclose(tmpfile);
```

Εικόνα 5.9. Μεταβολή της x θέσης σε σχέση με τον χρόνο



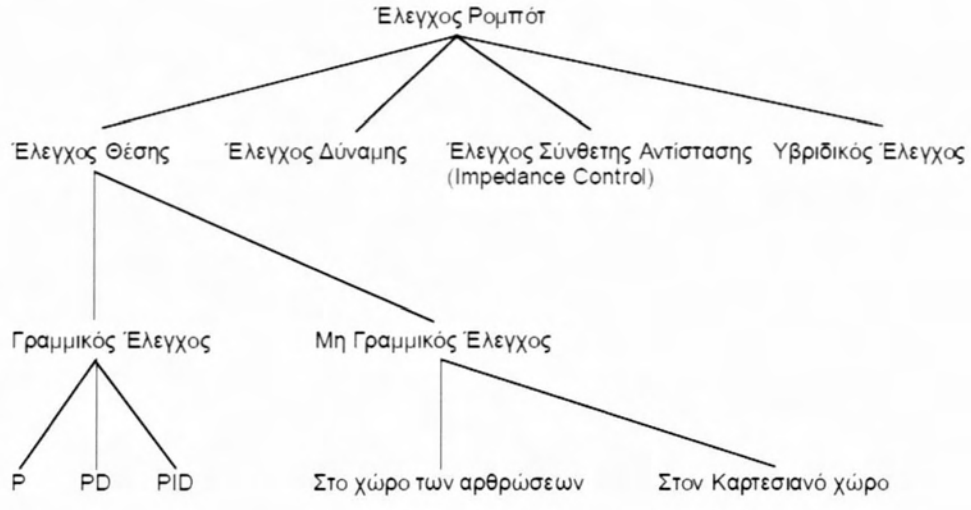
Εικόνα 5.10. Μεταβολή της y θέσης σε σχέση με τον χρόνο

Εικόνα 5.11. Μεταβολή της ταχύτητας σε σχέση με τον χρόνο

ΚΕΦΑΛΑΙΟ 6. ΑΥΤΟΜΑΤΟΣ ΕΛΕΓΧΟΣ ΡΟΜΠΟΤ

6.1. Γενικές παρατηρήσεις.

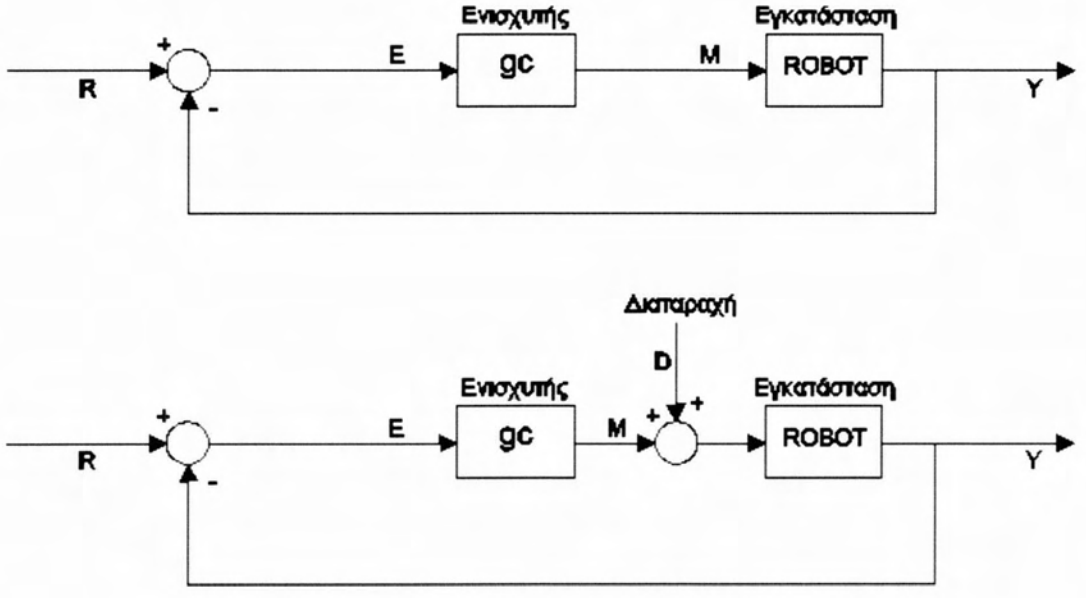
Ένα ρομπότ είναι γενικά ένα πολυμεταβλητό, μη γραμμικό και χρονικά μεταβαλλόμενο σύστημα. Νόμοι ελέγχου, που είναι κατάλληλοι για γραμμικά συστήματα, ή μικρές μεταβολές γύρω από λειτουργικά σημεία, δεν προσφέρονται για τον έλεγχο ενός συστήματος στο οποίο η αδράνεια μπορεί να αλλάξει με μεγάλους λόγους τάξεως 1:100. Επιπλέον, το τι θέλουμε να ελέγξουμε σε ένα ρομπότ δεν είναι πάντοτε το ίδιο και όπως φαίνεται στο παρακάτω σχεδιάγραμμα, ο έλεγχος για ρομπότ μπορεί να έχει ως στόχο τον έλεγχο της θέσης, τον έλεγχο της δύναμης αλληλεπίδρασης με το περιβάλλον, τον έλεγχο της σχέσης μεταξύ θέσης (ταχύτητας) και δύναμης (Έλεγχος Σύνθετης Αντίστασης ή Εμπέδησης, Impedance Control), ή μπορεί να ελέγχεται το ρομπότ ως προς τη θέση σε κάποιες διευθύνσεις και ως προς τη δύναμη σε κάποιες άλλες (ο λεγόμενος και Υβριδικός Έλεγχος). Ο έλεγχος θέσης χωρίζεται σε δυο κατηγορίες. Μπορεί να είναι Γραμμικός (P, PD, PID, κλπ). ή μη Γραμμικός. Ο γραμμικός έλεγχος γενικά αγνοεί τη μη γραμμική δυναμική του ρομπότ και μπορεί να εφαρμοσθεί με καλά αποτελέσματα όταν το ρομπότ έχει αρθρώσεις. Ο μη Γραμμικός Έλεγχος είναι απαραίτητος σε ρομπότ υψηλής απόδοσης με κατευθείαν σύνδεση του κινητήρα με την αδράνεια που επιταχύνει (λέγονται και direct drive robot)και μπορεί να έχει ως στόχο την ελαχιστοποίηση συγκεκριμένων σφαλμάτων(ρομποτικές αρθρώσεις, σφάλματα στον χώρο). [32]



Εικόνα 6.1. Ταξινόμηση μεθόδου ελέγχων.

6.2.1. Σύστημα ελέγχου θέσης και ταχύτητας.

Ο τελευταίος κώδικάς που δημιουργήθηκε, έχει σκοπό να μετρήσει το σχετικό σφάλμα μεταξύ της επιθυμητής γωνίας που θέλουμε να στρίψει το ρομπότ και της πραγματικής γωνίας που στρίβει και αντίστοιχα το σχετικό σφάλμα μεταξύ επιθυμητής και πραγματικής ταχύτητας. Έχουμε να κάνουμε λοιπόν με ένα απλό γραμμικό σύστημα μιας εισόδου και μιας εξόδου [33] με ανάδραση (βλέπε Εικόνα 6.1.). Το σύστημα περιλαμβάνει μια είσοδο (την επιθυμητή γωνία στροφής π.χ. 90^0), έναν απλό ενισχυτή (π.χ. gc ο οποίος έχει μια σταθερά ενίσχυσης kp η οποία ενισχύει το σφάλμα και το στέλνει κατόπιν στο σύστημα, το σύστημα-ρομπότ, την ανάδραση (η οποία εδώ είναι μονάδα) και την γωνία εξόδου. Επίσης όπως φαίνεται και στο δεύτερο σχήμα μπορούμε να έχουμε και μια εξωτερική διαταραχή (αυτό μπορεί να γίνει και με άλλους τρόπους π.χ. παρεμβάλλοντας ένα εμπόδιο στην πορεία του ρομπότ).



Εικόνα 6.2. Απλά συστήματα αυτομάτου ελέγχου με και χωρίς εξωτερική διαταραχή.

Η δυσκολία στην μέτρηση του σφάλματος είναι ότι, χωρίς την χρήση ενός εξωτερικού μηχανισμού που να θεωρείται αξιόπιστος, δεν υπάρχει κάτι ώστε να αντιπαρατεθεί με της μετρήσεις του ρομπότ, δηλαδή κάθε φορά νομίζοντας ότι η εργασία ολοκληρώθηκε σωστά τότε το output που θα πάρει ο

χρήστης είναι όσες μοίρες δόθηκαν ως εντολή για να στρίψει. Εδώ μπορεί να γίνει εισαγωγή μιας πλασματικής γωνίας λάθους. [34]

Παρακάτω περιγράφεται λοιπόν η διαδικασία που ακολουθείται για την θέση του ρομπότ και αντίστοιχα με μια απλή παραγώγιση dθ/dt, παρόμοια διαδικασία ακολουθείται και για την ταχύτητα μέσα από ένα απλό παράδειγμα.

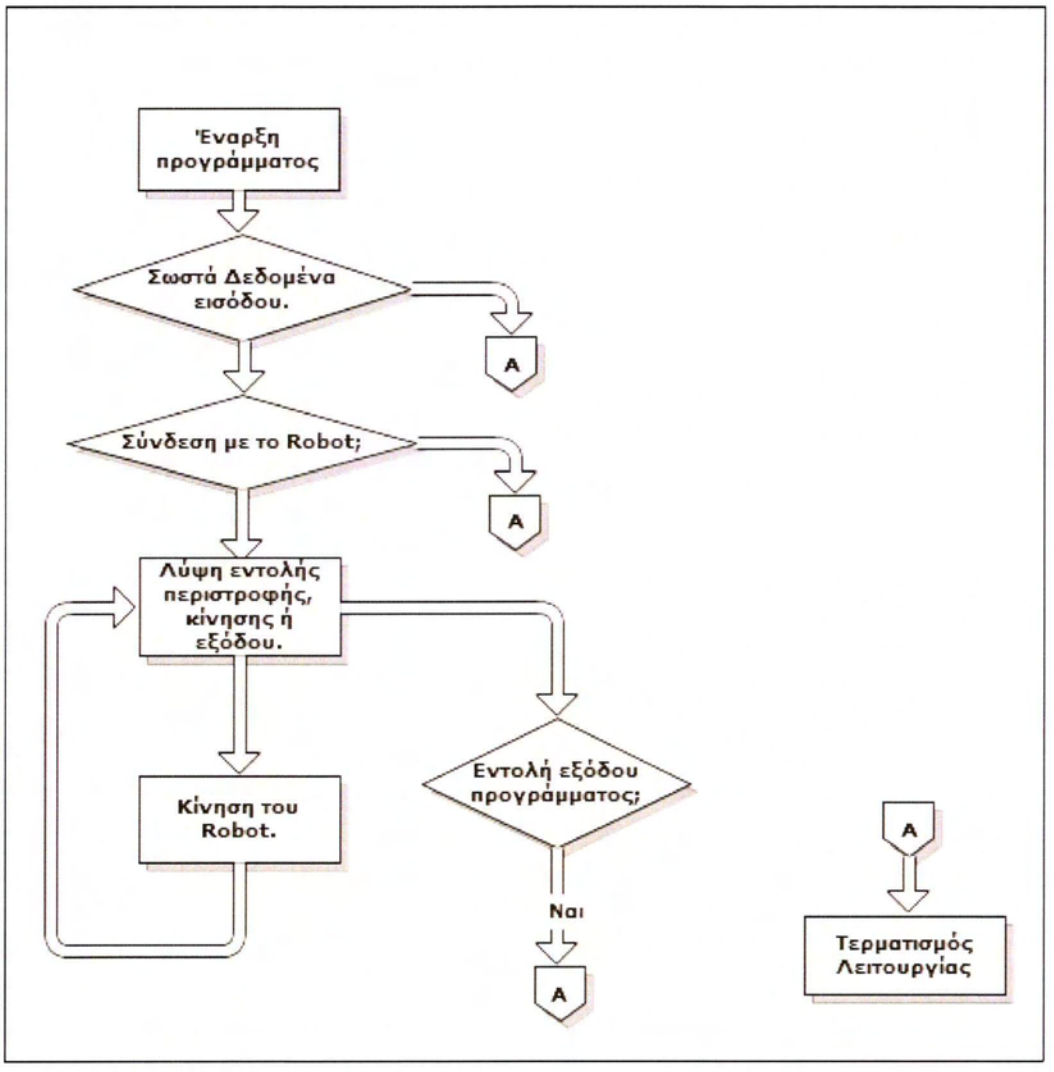
Παραδείγματος χάριν έστω ότι η επιθυμητή γωνία που θέλουμε να στρίψει το ρομπότ είναι 50^0, ο ενισχυτής σφάλματος έχει σταθερά 1 και η πλασματική έστω γωνία λάθους 5^0.

Η μόνη μεταβλητή που θα αλλάζει θα είναι η έξοδος από το mobile robot αρχίζοντας από 0^0 και φτάνοντας ως την επιθυμητή γωνία συν την γωνία λάθους που έχουμε εισάγει (δηλαδή στην περίπτωση μας $50^0+5^0= 55^0$) Επομένως κάθε φορά στο σύστημα γίνεται η εξής αλληλουχία πράξεων (επιθυμητή γωνία - (έξοδος του mobile robot –πλασματική γωνία λάθους)) επί την σταθερά ενίσχυσης (kp) και προσαρμόζοντας την στο παράδειγμα οι πράξεις που πραγματοποιούνται φαίνονται παρακάτω:

(50 - (0-5))*1 = 55
(50 - (1-5))*1 = 54
(50 - (2-5))*1 = 53
(50 - (3-5))*1 = 52
....
....
(50 - (55-5))*1 = 0
και εφόσον πλέον δεν υπάρχει σφάλμα το πρόγραμμα θα σταματήσει.
Στις παρακάτω σελίδες επεξηγούνται τα κομμάτια του αντίστοιχου κώδικα.

Ο κώδικας αρχίζει κατά τα γνωστά με την αρχικοποίηση των αισθητήρων του ρομπότ, της κάμερας κτλ

```
Aria::init();
ArRobot robot;
ArKeyHandler keyHandler;  κ.ο.κ.
```

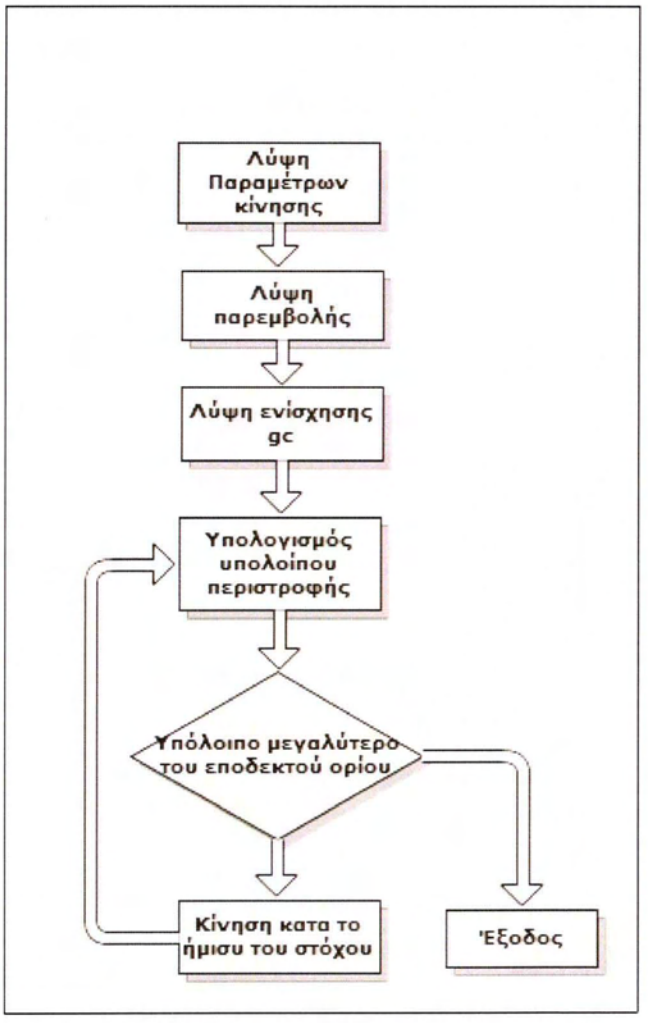


Εικόνα 6.3. Δομικό διάγραμμα για τον αυτόματο έλεγχο του ρομπότ

Ακολουθεί η σύνδεση με το ρομπότ:

```
 ArSimpleConnector simpleConnector(&argParser)    si;
   //Connection  if (!Aria::parseArgs())
{
  Aria::logOptions();
  keyHandler.restore();
  Aria::shutdown();
  return 1;
} if (!simpleConnector.connectRobot(&robot))
{
  printf("Could not connect to robot... exiting\n");
  keyHandler.restore();
```

```
  Aria::shutdown();
  return 1;
}
```



Εικόνα 6.4. Δομικό διάγραμμα για τον αυτόματο έλεγχο θέσης και ταχύτητας του ρομπότ.

Κατόπιν με τις εντολές Araction (Aractionlimiter)

για τους περιορισμούς της κίνησης:

πχ. ArActionLimiterForwards limiter("speed limiter near", 300, 600, 250);

Και επίσης:

```
robot.addAction(&limiter, 100);
robot.addAction(&limiterFar, 99);
robot.addAction(&backwardsLimiter, 98);
robot.addAction(&Follow, 77);
robot.addAction(&backup, 50);
```

```
robot.addAction(&stop, 30);
Follow Follow(&acts, &vcc4);
Aria::setKeyHandler(&keyHandler);
robot.attachKeyHandler(&keyHandler);
robot.addRangeDevice(&sonar);
acts.openPort(&robot);
vcc4.init();
ArUtil::sleep(1000);
robot.setAbsoluteMaxTransVel(400);

robot.comInt(ArCommands::ENABLE, 1);
robot.comInt(ArCommands::SOUNDTOG, 0);
ArUtil::sleep(200);
```

Ακολουθεί η εντολή ώστε το ρομπότ να αρχίζει να κινείται:

```
robot.runAsync(true);
```

Το πρόγραμμα θα χρειαστεί προφανώς να δανειστεί μεθόδους από την βιβλιοθήκη του Aria:

```
#include "Aria.h"
#include <stdio.h>
#include <iostream>
#include <time.h>
#include <fstream>
#include <sstream>
#include <limits>
using namespace std;
```

Το επόμενο μέρος είναι απαραίτητο ώστε να παραχθεί το output αρχείο με τα αποτελέσματα θέσης και ταχύτητας (που τυπώνονται με τις εντολές fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime)); fprintf (tmpfile , "Current Position X: %f Y: %f Velocity: %f \n\n",x,y,vel);) το οποίο ο αναγνώστης μπορεί να βρεί ολόκληρο και στο παράρτημα της εργασίας.

```
FILE *tmpfile = fopen("AriaCameraPositionTest.txt","a");
```

```
while(true){
        double x = robot.getX();
        double y = robot.getY();
        double vel = robot.getVel();
        time_t rawtime;
        time(&rawtime);
        fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));
        fprintf (tmpfile , "Current Position X: %f Y: %f Velocity: %f
\n\n",x,y,vel);
        int mseconds = 1000;
        clock_t goal = mseconds + clock();
        while (goal > clock());
 }
fclose(tmpfile);
robot.waitForRunExit();
Aria::shutdown();
return 0;
}
```

Παρακάτω το πρόγραμμα βάση των προαναφερθέντων πράξεων ((επιθυμητή γωνία - (έξοδος του mobile robot –πλασματική γωνία λάθους) επί την σταθερά ενίσχυσης kp)) θα πραγματοποιήσει το υπολογιστικό μέρος του κώδικα. Έτσι για την περιστροφή:

```
void rotate(ArRobot* rob, int degrees,int distor,int gccoef,Log* mylog){
        Log myLog = Log();
        myLog.MessageChange(degrees,distor,gccoef,0);
```

Και το pioneer θα προσαρμόσει την περιστροφή του και θα αυξήσει τις μοίρες σύμφωνα με την θεωρία:

```
    rob->setDeltaHeading(degrees+distor*gccoef);

        while(rob->isHeadingDone()==false){
```

```
                ArUtil::sleep(200);
                myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),0);
        }
        myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),888);
        //myLog.Message(*rob,0);
}
```

Και αντίστοιχα για την κίνηση:

```
void moveRob(ArRobot* rob, int distance,Log* mylog){
        rob->move(distance);
        Log myLog = Log();
        myLog.MessageChange(0,0,0,distance);
        while(rob->isMoveDone()==false){
                ArUtil::sleep(200);
                myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),0);
        }
        myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),999);
}

int checkDigit(string inpstr){
        const char * csdf = inpstr.c_str();
        if(strspn(csdf,"-0123456789")==inpstr.length()){
                return 1;
        }
        else{
                return 0;
        }
}
```

Από εδώ και κάτω αρχίζει η βασική συνάρτηση του κώδικα και οι απαραίτητες αρχικοποιήσεις :

```
int main(int argc, char** argv)
{

Log myLog = Log();
Aria::init();
ArRobot robot;
ArKeyHandler keyHandler;
ArArgumentParser argParser(&argc, argv);
argParser.loadDefaultArguments();
ArSimpleConnector simpleConnector(&argParser);
```

Και η προσπάθεια για σύνδεση ή όχι του ρομπότ:

```
 if (!Aria::parseArgs())
 {
   Aria::logOptions();
   keyHandler.restore();
   Aria::shutdown();
   return 1;
 }

   if (!simpleConnector.connectRobot(&robot))
 {
   printf("Could not connect to robot... exiting\n");
   //keyHandler.restore();
   Aria::shutdown();
   return 1;
 }

 ArUtil::sleep(1000);
 robot.runAsync(true);
```

```
robot.enableMotors();
bool check = true;
```

Όπως ειπώθηκε είναι απαραίτητες τρεις μεταβλητές, έτσι ώστε ο κώδικας να λειτουργήσει σωστά την επιθυμητή γωνία περιστροφής, την γωνία σφάλματος απόκλισης και τον συντελεστή ενίσχυσης gc.

Σε αρχικό στάδιο θέτουμε την επιθυμητή γωνία περιστροφής, την γωνία σφάλματος απόκλισης, τον συντελεστή ενίσχυσης gc, την αρχική γωνία που βρίσκεται το ρομπότ και την απόσταση που πρόκειται να διανύσει ίσον με μηδέν.

```
while(check)
{
    string in="",dummy="";
    int angle=0,distortion=0,gccoef=0,turnangle=0,disttance=0;
```

Έπειτα εμφανίζεται το ακόλουθο μήνυμα στην οθόνη του χρήστη όπου υπάρχει η επιλογή για κίνηση ή περιστροφή του ρομπότ:

```
    cout << "Please Choose action. m for move , r for rotate , q for quit: ";

    string mystr1;
    getline (cin,mystr1);
    stringstream(mystr1)>> in;
    string mystr;
    if(in.compare("m")!=0&&in.compare("r")!=0&&in.compare("q")!=0){
        continue;
    }
    else if(in.compare("r")==0)
    {

        system("cls");
```

Τώρα ο χρήστης καλείτε να πληκτρολογήσει τις τρεις προαναφερθείσες μεταβλητές (οι οποίες πρέπει να είναι θετικές, αλλιώς εμφανίζεται το μήνυμα Not a valid number στην οθόνη).

```
cout << "Enter Rotation Angle :";
getline (cin,mystr);
if(checkDigit(mystr)>0){
        stringstream(mystr) >> angle;
}
else{
        cout<<"Not a valid number";
        cout<<endl;
        continue;
}

cout << "Enter Distortion Angle :";
getline (cin,mystr);
if(checkDigit(mystr)>0){
        stringstream(mystr) >> distortion;
}
else{
        cout<<"Not a valid number";
        cout<<endl;
        continue;
}
cout << "Enter gc Coeficient :";
getline (cin,mystr);
if(checkDigit(mystr)>0){
        stringstream(mystr) >> gccoef;
}
else{
        cout<<"Not a valid number";
        cout<<endl;
        continue;
```

```
                }

            rotate(&robot,angle,distortion,gccoef,&myLog);

        myLog.MessageNew(robot.getX(),robot.getY(),robot.getVel(),robot.

getCompass(),robot.getTh(),111);
                system("cls");
                continue;
        }
        else if(in.compare("m")==0){
                system("cls");
                cout << "Enter Distance :";
                getline (cin,mystr);

                if(checkDigit(mystr)>0){
                        stringstream(mystr) >> disttance;
                        cout<<"Starting Moving for "<<disttance;
                        cout<<endl;
                }
                else{
                        cout<<"Not a valid number";
                        cout<<endl;
                }
                moveRob(&robot, disttance,&myLog);

        //myLog.MessageNew(robot.getX(),robot.getY(),robot.getVel(),robot.get
Compass(),robot.getTh(),0);
                continue;
        }
        else{
                check = false;
        }
 }
```

Και τέλος ο κώδικας κλείνει απλά με το πλήκτρο escape, το ρομπότ αποσυνδέεται και το aria τερματίζει την λειτουργία του:

```
 robot.disconnect();
   Aria::shutdown();
   return 0;
}
```

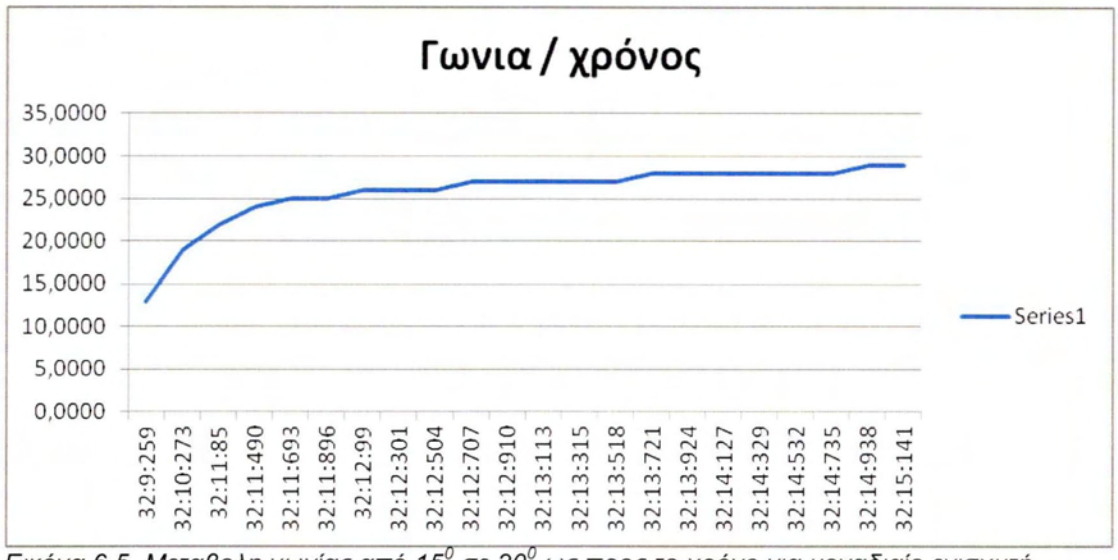
Συμπερασματικά το κομμάτι του αυτομάτου ελέγχου είναι ένα απαραίτητο εργαλείο ώστε να μετρηθούν τα σχετικά σφάλματα που προκύπτουν σε εντολές αλλαγής θέσης και ταχύτητας. Μέσω απλών γραμμικών συστημάτων είναι δυνατή η ελαχιστοποίηση τους πράγμα πολύ σημαντικό για την υψηλή απόδοση των κινητών ρομπότ και την αξιοπιστία τους. Επίσης, σε μελλοντικές εργασίες προκειμένου να γίνουν ακόμα πιο ρεαλιστικες προσομοιώσεις μπορεί να γίνει όπως προαναφέρθηκε η χρήση ενός εξωτερικού μηχανισμού.

6.3. Εξαγώγιμα αρχεία του κώδικα αυτομάτου ελέγχου.

Τα παραγόμενα αρχεία θα είναι της μορφής AriaErrorLog.xml και η μορφή τους φαίνεται παρακάτω.

```
<State>
<Rotation Angle>0</Rotation Angle>
<Distortion>0</Distortion>
<gccoef>0</gccoef>
<Distance>-400</Distance>
</State>

<State>
<Time>
Tue Jan 31 16:57:37 2012
</Time>
<X>-1.94</X>
```

```
<Y>0</Y>
<Velocity>-24</Velocity>
<Compass>0</Compass>
<Th>0</Th>
<NewDHeading>0</NewDHeading>
</State>

<State>
<Time>
Tue Jan 31 16:57:37 2012
</Time>
<X>-15.52</X>
<Y>0</Y>
<Velocity>-83</Velocity>
<Compass>0</Compass>
<Th>0</Th>
<NewDHeading>0</NewDHeading>
</State>

<State>
<Time>
Tue Jan 31 16:57:37 2012
</Time>
<X>-40.74</X>
<Y>0</Y>
<Velocity>-140</Velocity>
<Compass>0</Compass>
<Th>0</Th>
<NewDHeading>0</NewDHeading>
</State>                    KOK...
```



Εικόνα 6.5. Μεταβολη γωνίας από 15^0 σε 30^0 ως προς το χρόνο για μοναδιαίο ενισχυτή.

Εικόνα 6.6. Μεταβολή της ταχύτητας ως προς το χρόνο

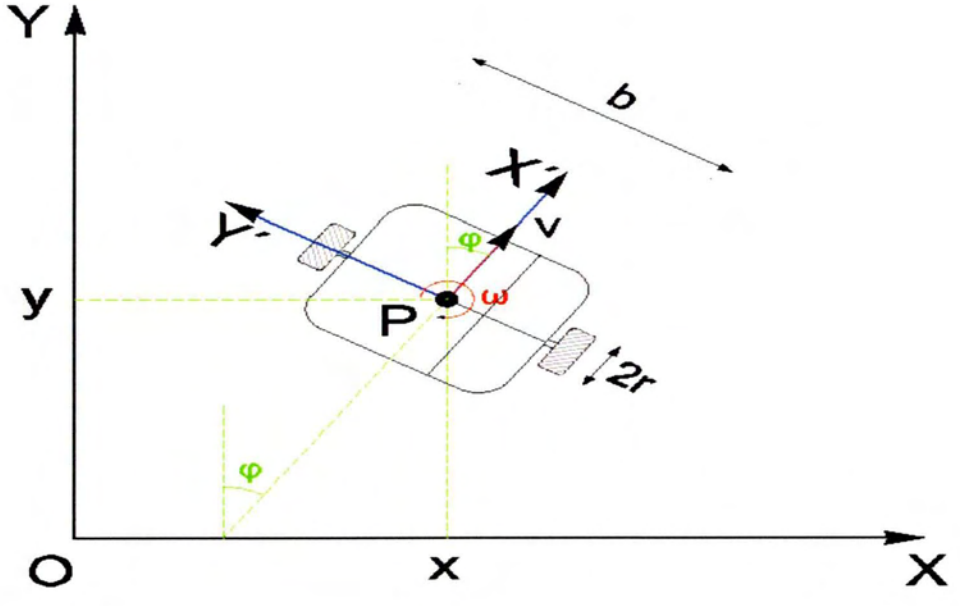
ΚΕΦΑΛΑΙΟ 7. ΚΙΝΗΜΑΤΙΚΟ ΜΟΝΤΕΛΟ ΤΟΥ PIONEER P3-DX

7.1. Κινηματικό μοντέλο.

7.1.1 Εισαγωγή

Στο κεφάλαιο αυτό εξάγονται οι κινηματικές εξισώσεις που περιγράφουν το σύστημα που θα μελετηθεί στη συνέχεια. Οι εξισώσεις αυτές είναι ιδιαίτερα σημαντικές, διότι παρέχουν σχέσεις μεταξύ των εμπλεκόμενων μεταβλητών σε γεωμετρικό και κινηματικό επίπεδο, οι οποίες είναι άμεσα συνδεδεμένες με κάθε προσπάθεια σχεδιασμού ελεγκτών κίνησης. [35]

Το προς μελέτη όχημα φέρει δύο τροχούς, οι οποίοι είναι υπεύθυνοι για την πρόωση και τον προσανατολισμό του οχήματος. Οι τροχοί αυτοί κινούνται ανεξάρτητα μεταξύ τους, ενώ ένας μικρότερος παθητικός τροχός προστίθεται για λόγους ευστάθειας. Στη συνέχεια θεωρείται το όχημα που απεικονίζεται στο σχήμα, όπου ο παθητικός τροχός δεν έχει σχεδιαστεί και δεν λαμβάνεται υπ' όψη μιας και δεν έχει καμία συμμετοχή στην οδήγηση του οχήματος. Η κίνηση του οχήματος θεωρείται στο επίπεδο.



Εικόνα 7.1. Μεταβλητές κινηματικού μοντέλου.

7.1.2 Ευθεία κινηματική

Οι βασικές διαστάσεις του οχήματος δίνονται στην εικόνα 7.1.

Οι βασικές συνθήκες για τη δημιουργία του μαθηματικού μοντέλου είναι: [38]

- Το όχημα θεωρείται ότι κινείται σε οριζόντιο επίπεδο.
- Υπάρχει σημειακή επαφή των τροχών
- Οι τροχοί είναι μη παραμορφώσιμοι
- Οι τροχοί δεν ολισθαίνουν (V=0 στο σημείο επαφής τους με το δάπεδο)
- Δεν υπάρχει τριβή στο σημείο επαφής των τροχών με το δάπεδο
- Οι τροχοί στηρίζονται σε σταθερό σκελετό (σασί)

Είναι σαφές ότι για την πλήρη γνώση της συμπεριφοράς του οχήματος απαιτούνται τρείς μεταβλητές για την γνώση της θέσης του σε κάθε χρονική στιγμή οι συντεταγμένες (x,y) του σημείου P (βλέπε σχ. 7.1.) και η γωνία προσανατολισμού φ.

Στο σχήμα 7.1 απεικονίζονται:

- OX, OY αδρανιακό σύστημα συντεταγμένων
- OX', OY' σωματόδετο σύστημα συντεταγμένων (ΟΧ΄ ο κύριος άξονας του κινητού)
- V η γραμμική ταχύτητα
- ω η γωνιακή (περιστροφική) ταχύτητα
- x, y, θ η γενικευμένες συντεταγμένες του κινητού
- b η απόσταση μεταξύ των τροχών του κινητού
- $2r$ η διάμετρος του τροχού κινήσεως του κινητού

Κάνοντας την παραδοχή ότι το όχημα κινείται με χαμηλές ταχύτητες, μπορεί να υποτεθεί ότι η πλευρική ολίσθηση των τροχών είναι αμελητέα και συνεπώς η ταχύτητα V της πλατφόρμας είναι παράλληλη προς τον κύριο άξονα του οχήματος.

Η ταχύτητα V του κινητού αναλύεται στις συνιστώσες τις στους άξονες OX, OY ως εξής: $x' = V\sin\phi \quad y' = V\cos\phi$. Επίσης ισχύει $\phi' = \omega$. (1)

όπου x' είναι η συνιστώσα της ταχύτητας κατά τον άξονα x, y' είναι η συνιστώσα της ταχύτητας κατά τον άξονα y και $\phi' = \omega$ είναι η γωνιακή ταχύτητα περιστροφής της πλατφόρμας.

Οι εξισώσεις (1) γραμμένες υπό μορφή πινάκων γίνονται:

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} \sin\phi & 0 \\ \cos\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix} \qquad (2)$$

Όπως προκύπτει από την εικόνα 7.1

$$V = \frac{V_R + V_L}{2} = \frac{\omega_R + \omega_L}{2} r \qquad (3)$$

$$\omega = \frac{V_R - V_L}{b} = \frac{\omega_R - \omega_L}{b} r \qquad (4)$$

Όπου b η απόσταση μεταξύ των τροχών , r η ακτίνα των τροχών κίνησης, ω_R, ω_L οι γωνιακές ταχύτητες του δεξιού και αριστερού τροχού αντίστοιχα του ρομπότ.

Αντικαθιστώντας τις σχέσεις (3) και (4) στην σχέση (2) έχουμε,

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} \frac{r(\sin\phi)}{2} & \frac{r(\sin\phi)}{2} \\ \frac{r(\cos\phi)}{2} & \frac{r(\cos\phi)}{2} \\ \frac{-r}{b} & \frac{r}{b} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \qquad (5)$$

Και θεωρώντας $\omega_i = 2\pi n_i$, $i = L, R$ όπου n_R και n_L οι στροφές δεξιού και αριστερού τροχού του ρομπότ αντίστοιχα έχουμε,

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} \frac{r(\sin\phi)}{2} & \frac{r(\sin\phi)}{2} \\ \frac{r(\cos\phi)}{2} & \frac{r(\cos\phi)}{2} \\ \frac{-r}{b} & \frac{r}{b} \end{bmatrix} \begin{bmatrix} 2\pi n_L \\ 2\pi n_R \end{bmatrix} \qquad (6)$$

Ο τελικός τύπος υπολογισμού μετά από πράξεις στη σχέση (6) είναι

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} \pi r \cdot (\sin\phi) & \pi r \cdot (\sin\phi) \\ \pi r \cdot (\cos\phi) & \pi r \cdot (\cos\phi) \\ \frac{-2\pi r}{b} & \frac{2\pi r}{b} \end{bmatrix} \begin{bmatrix} n_L \\ n_R \end{bmatrix} \qquad (7)$$

7.1.2 Αντίστροφη κινηματική

Στην αντίστροφη κινηματική για την μετάβαση του κινητού, από την αρχική θέση στην τελική θέση, υπολογίζονται οι συνδυασμοί των στροφών των τροχών ώστε να επιτευχθεί αυτή η μετάβαση. Η λύση αυτού του προβλήματος είναι αρκετά πολύπλοκη. Η αιτία της δυσκολίας έγκειται στο ότι υπάρχουν πολλές πιθανές λύσεις του προβλήματος, δηλαδή συνδυασμοί στροφών των τροχών. Μεταξύ των πιθανών λύσεων μπορούν όμως με περιορισμούς, όπως διαδρομή μικρότερης κατανάλωσης ενέργειας ή ομαλότερης πορείας να αποκλειστούν μερικές λύσεις και να παραμείνουν οι πλέον λογικές. [35]

7.2. Αναλυτική περιγραφή υπολογισμών θέσης και ταχύτητας Robot

Η κίνηση του Robot εξαρτάται από την ταχύτητα περιστροφής των τροχών του. Ίδια ταχύτητα περιστροφής και στους δύο τροχούς έχει σαν αποτέλεσμα την ευθύγραμμη κίνηση του ενώ διαφορές στην ταχύτητα περιστροφής μεταξύ των τροχών θα οδηγήσει σε καμπυλόγραμμη κίνηση.
Για να πραγματοποιηθούν, λοιπόν, υπολογισμοί της θέσης και της ταχύτητας του Robot πρέπει απαραίτητα να γνωρίζουμε τις ταχύτητες περιστροφής των τροχών σαν συναρτήσεις του χρόνου. Για την πληρέστερη αντιμετώπιση του προβλήματος θεωρήσαμε γραμμική μεταβολή των συχνοτήτων περιστροφής των τροχών. Συγκεκριμένα θεωρήθηκαν οι σχέσεις μεταβολής:

$n_r = a_1 + b_1 t$ (1)

$n_l = a_2 + b_2 t$ (2)

Όπου n_r και n_l είναι οι συχνότητες περιστροφής δεξιού και αριστερού τροχού αντίστοιχα.

Ο καθορισμός σταθερής μορφής εξισώσεων στο εσωτερικό του προγράμματος υπολογισμού θα περιόριζε σημαντικά τη δυνατότητα μελέτης μεγάλου αριθμού κινήσεων και έτσι επιλέχθηκε η λύση να καθορίζονται εξωτερικά από τον χρήστη οι συντελεστές a και b. Ο χρήστης έχει τη δυνατότητα, επίσης, να καθορίζει εξωτερικά τόσο το χρονικό βήμα όπως και των αριθμό των βημάτων για τους οποίους θα γίνονται οι υπολογισμοί.

Με το ξεκίνημα του προγράμματος ενημερώνονται εσωτερικά μεταβλητές που αφορούν τα φυσικά χαρακτηριστικά του Robot που είναι η ακτίνα τροχών, η απόσταση των τροχών και μέγιστη ταχύτητα περιστροφής των τροχών. Στη συνέχεια αφού ανοιχθεί το αρχείο στο οποίο θα καταγραφούν τα αποτελέσματα ζητείται από τον χρήστη να καθορίσει τις μεταβλητές a και b τόσο για τον δεξιό όσο και για τον αριστερό τροχό καθώς και το χρονικό βήμα υπολογισμών και τον αριθμό των βημάτων.

Το λογικό διάγραμμα και ο κώδικας του προγράμματος επισυνάπτονται στο τέλος του κεφαλαίου. Ο κώδικας του προγράμματος αναπτύχθηκε σε γλώσσα προγραμματισμού FORTRAN.

Στη συνέχεια παρουσιάζονται τρία σενάρια διαφορετικών κινήσεων του robot.

7.2.1. Ευθύγραμμη επιταχυνόμενη κίνηση.

Για τις ταχύτητες περιστροφής των τροχών του robot θεωρήθηκαν οι παρακάτω σχέσεις:

$n_r = 0.1 + 0.1t$ (3)

$n_l = 0.1 + 0.1t$ (4)

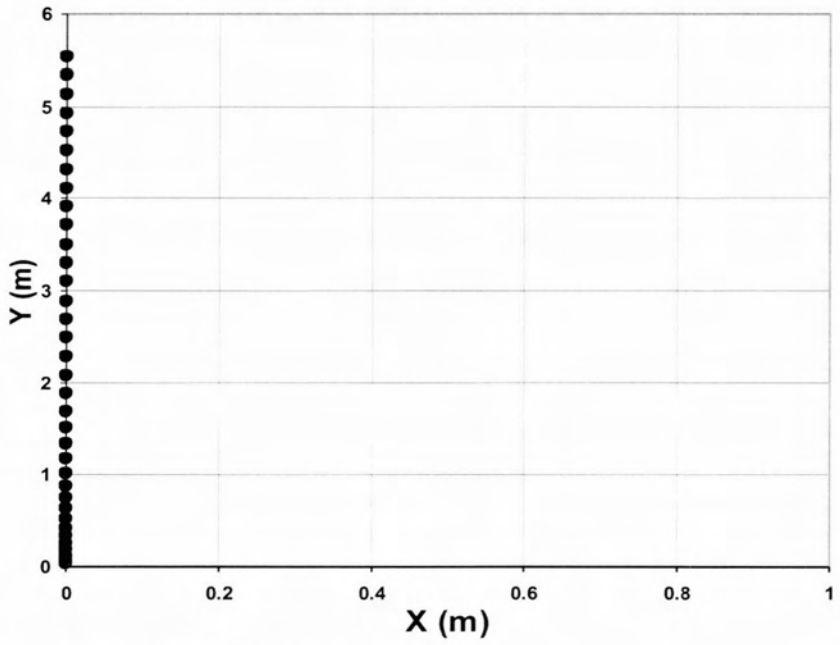
Από τις (3) και (4) φαίνεται ότι και οι δύο τροχοί εκτελούν επιταχυνόμενη κίνηση, ενώ πάντα έχουν την ίδια ταχύτητα περιστροφής, με αποτέλεσμα και η κίνηση του robot να είναι ευθύγραμμη και επιταχυνόμενη.

Στον πίνακα 7.1 φαίνονται μερικές τιμές των παραμέτρων που υπολογίστηκαν και συγκεκριμένα η ταχύτητα περιστροφής κάθε τροχού, οι συντεταγμένες της θέσης του robot καθώς και το μέτρο και η διεύθυνση της ταχύτητας του.

Πίνακας 7.1. Τιμές στροφών δεξιού και αριστερού τροχού , συντεταγμένων (X,Y), διεύθυνσης και μέτρου της ταχύτητας (φ,V) του robot

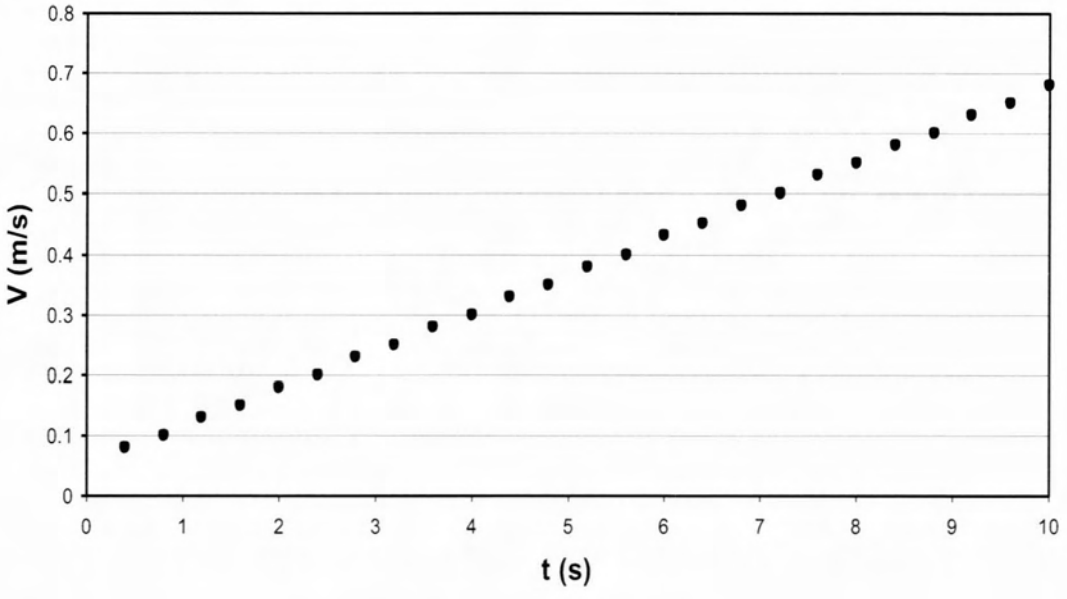
Time Step	Time	Nr	Nl	X	Y	φ	V
1	0.4	0.12	0.12	0	0.03	0	0.08
2	0.8	0.16	0.16	0	0.07	0	0.1
3	1.2	0.2	0.2	0	0.12	0	0.13
4	1.6	0.24	0.24	0	0.18	0	0.15
5	2	0.28	0.28	0	0.25	0	0.18
6	2.4	0.32	0.32	0	0.33	0	0.2
7	2.8	0.36	0.36	0	0.42	0	0.23
8	3.2	0.4	0.4	0	0.52	0	0.25
9	3.6	0.44	0.44	0	0.63	0	0.28
10	4	0.48	0.48	0	0.75	0	0.3
11	4.4	0.52	0.52	0	0.88	0	0.33
12	4.8	0.56	0.56	0	1.02	0	0.35
13	5.2	0.6	0.6	0	1.18	0	0.38
14	5.6	0.64	0.64	0	1.34	0	0.4
15	6	0.68	0.68	0	1.51	0	0.43
16	6.4	0.72	0.72	0	1.69	0	0.45
17	6.8	0.76	0.76	0	1.88	0	0.48
18	7.2	0.8	0.8	0	2.08	0	0.5
19	7.6	0.84	0.84	0	2.29	0	0.53
20	8	0.88	0.88	0	2.51	0	0.55
21	8.4	0.92	0.92	0	2.74	0	0.58
22	8.8	0.96	0.96	0	2.98	0	0.6
23	9.2	1	1	0	3.24	0	0.63
24	9.6	1.04	1.04	0	3.5	0	0.65

Στο σχήμα 7.2 φαίνεται η θέση του robot στο καρτεσιανό επίπεδο. Παρατηρούμε ότι η κίνηση του είναι ευθύγραμμη κατά τον άξονα Y και ταυτόχρονα επιταχυνόμενη, όπως διαπιστώνεται από τις διαδοχικές θέσεις του που αντιστοιχούν σε ίσα χρονικά διαστήματα.



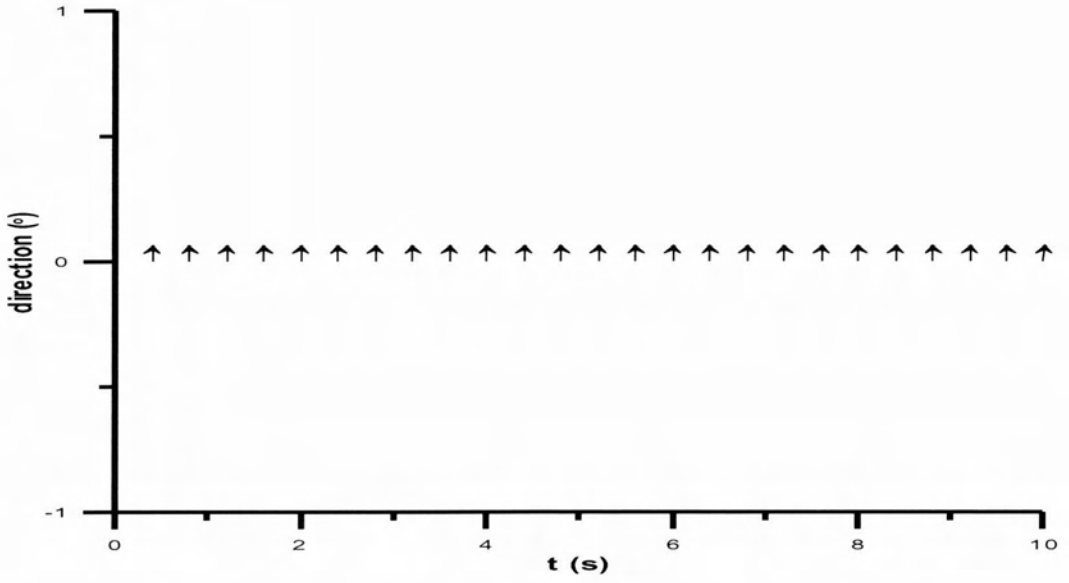
Εικόνα 7.2. Θέση του robot στο επίπεδο
($n_r = 0.1 + 0.1t, n_l = 0.1 + 0.1t$)

Στο σχήμα 7.2 φαίνεται το μέτρο της ταχύτητας του robot, όπου και εδώ παρατηρούμε ότι η κίνηση είναι ομαλά επιταχυνόμενη, σε συμφωνία με την επιταχυνόμενη κίνηση περιστροφής των τροχών, σχέσεις (3) και (4).



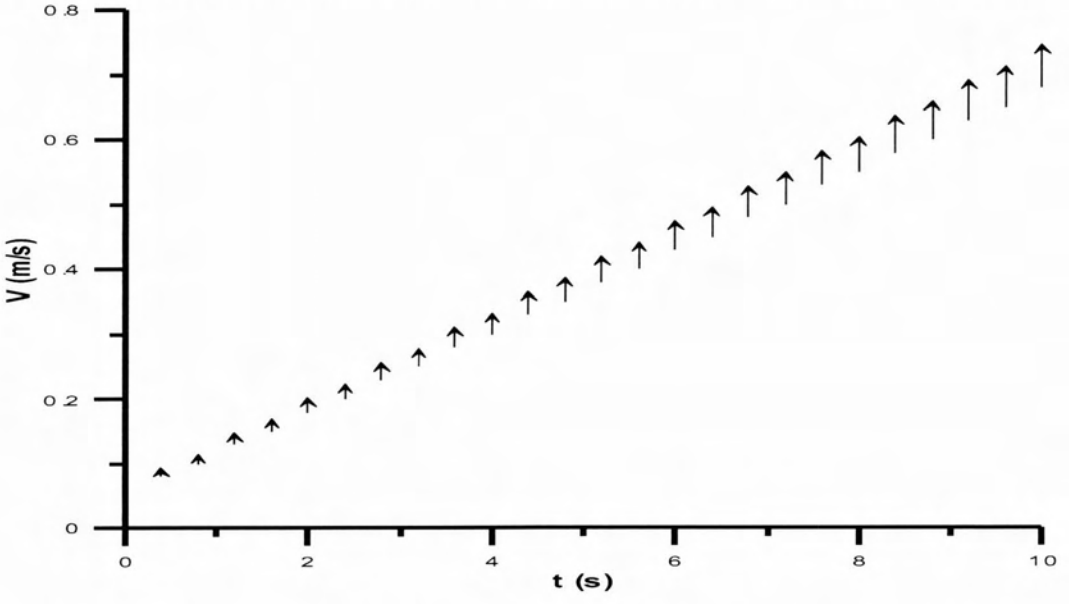
Εικόνα 7.3. Ταχύτητα κίνησης του robot σε σχέση με τον χρόνο
($n_r = 0.1 + 0.1t, n_l = 0.1 + 0.1t$)

Στην εικόνα 7.4 φαίνεται η διεύθυνση της ταχύτητας του robot. Παρατηρούμε ότι είναι σταθερή σε συμφωνία με την εικόνα 7.2.



Εικόνα 7.4. Διεύθυνση της ταχύτητα του robot σε σχέση με τον χρόνο ($n_r = 0.1 + 0.1t$, $n_l = 0.1 + 0.1t$)

Στην εικόνα 7.5 αποτελεί συνδυασμό των σχημάτων 2 και 3 όπου φαίνεται τόσο το μέτρο όσο και η διεύθυνση της ταχύτητας σε σχέση με τον χρόνο.



Εικόνα 7.5. Μέτρο και διεύθυνση του διανύσματος της ταχύτητας του robot ($n_r = 0.1 + 0.1t$, $n_l = 0.1 + 0.1t$)

7.2.2. Καμπυλόγραμμη επιταχυνόμενη κίνηση.

Για τις ταχύτητες περιστροφής των τροχών του robot θεωρήθηκαν οι παρακάτω σχέσεις

$n_r = 0.1 + 0.05t$ (5)

$n_l = 0.1 + 0.02t$ (6)

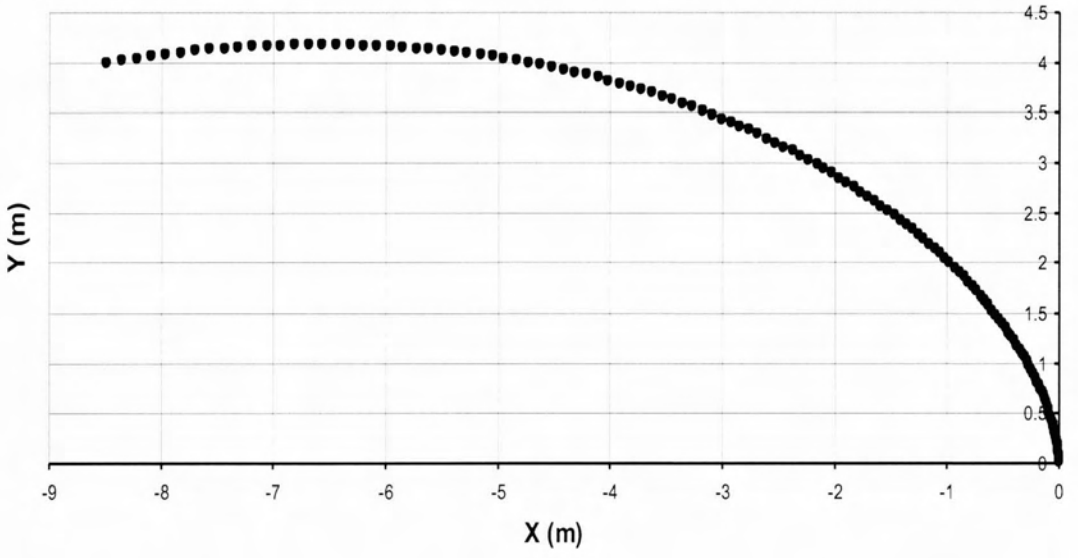
Από τις (5) και (6) φαίνεται ότι και οι δύο τροχοί εκτελούν επιταχυνόμενη κίνηση με διαφορετική ταχύτητα περιστροφής. Ο δεξιός τροχός έχει μεγαλύτερη επιτάχυνση και επομένως μεγαλύτερη ταχύτητα περιστροφής με αποτέλεσμα το robot να εκτελέσει καμπυλόγραμμη κίνηση προς τα αριστερά. Στον πίνακα 1 φαίνονται μερικές τιμές των παραμέτρων που υπολογίστηκαν και συγκεκριμένα η ταχύτητα περιστροφής κάθε τροχού, οι συντεταγμένες της θέσης του robot καθώς και το μέτρο και η διεύθυνση της ταχύτητας του.

Πίνακας 7.2. Τιμές στροφών δεξιού και αριστερού τροχού , συντεταγμένων (X,Y), διεύθυνσης και μέτρου της ταχύτητας (φ,V) του robot.

Time Step	Time	Nr	Nl	X	Y	φ	V
1	0.2	0.11	0.1	0	0.01	0.36	0.06
2	0.4	0.12	0.11	0	0.03	1.08	0.07
3	0.6	0.13	0.11	0	0.04	1.8	0.07
4	0.8	0.14	0.11	0	0.06	2.52	0.08
5	1	0.14	0.12	0	0.07	3.24	0.08
6	1.2	0.16	0.12	0	0.09	3.96	0.09
7	1.4	0.17	0.13	-0.01	0.11	4.68	0.09
8	1.6	0.17	0.13	-0.01	0.13	5.4	0.1
9	1.8	0.19	0.13	-0.01	0.15	6.12	0.1
10	2	0.2	0.14	-0.01	0.17	6.84	0.1
11	2.2	0.2	0.14	-0.01	0.19	7.56	0.11
12	2.4	0.22	0.15	-0.02	0.21	8.28	0.11
13	2.6	0.23	0.15	-0.02	0.24	9	0.12
14	2.8	0.23	0.15	-0.03	0.26	9.72	0.12
15	3	0.25	0.16	-0.03	0.29	10.44	0.13
16	3.2	0.25	0.16	-0.04	0.31	11.16	0.13
17	3.4	0.27	0.17	-0.04	0.34	11.88	0.14
18	3.6	0.28	0.17	-0.05	0.36	12.6	0.14
19	3.8	0.28	0.17	-0.05	0.39	13.32	0.14

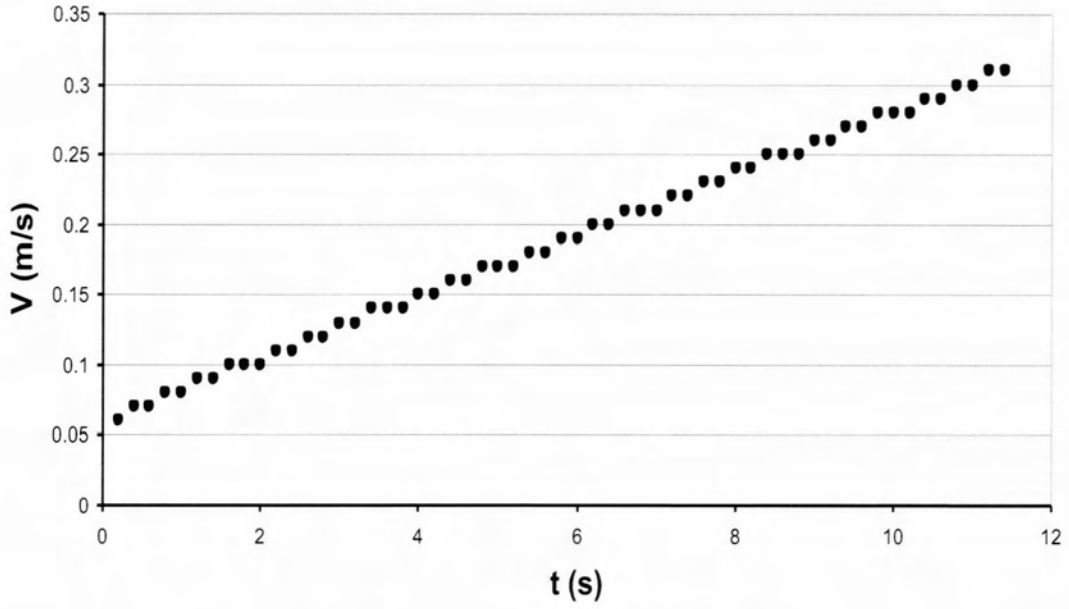
20	4	0.3	0.18	-0.06	0.42	14.04	0.15
21	4.2	0.31	0.18	-0.07	0.45	14.76	0.15
22	4.4	0.31	0.19	-0.08	0.48	15.48	0.16
23	4.6	0.33	0.19	-0.09	0.51	16.2	0.16
24	4.8	0.34	0.19	-0.1	0.54	16.92	0.17

Στην εικόνα 7.6 φαίνεται η θέση του robot στο καρτεσιανό επίπεδο. Παρατηρούμε ότι η κίνηση του είναι επιταχυνόμενη και καμπυλόγραμμη προς τα αριστερά σε συμφωνία με τις εξισώσεις (5) και (6).



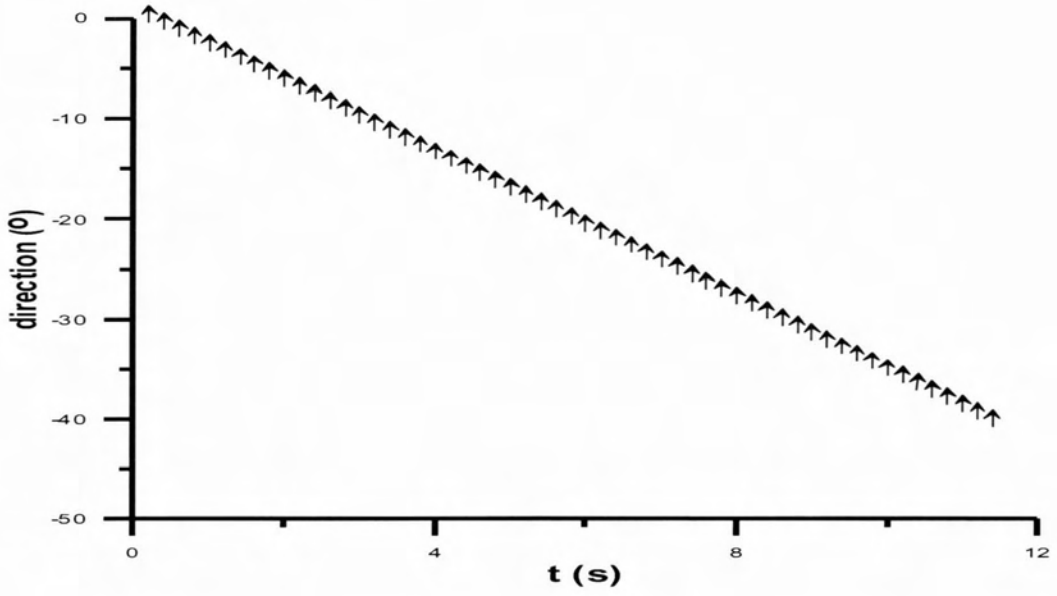
Εικόνα 7.6 . Θέση του robot στο επίπεδο ($n_r = 0.1 + 0.05t$, $n_l = 0.1 + 0.02t$)

Στην εικόνα 7.7 φαίνεται το μέτρο της ταχύτητας του robot για τα πρώτα 12s όπου και εδώ παρατηρούμε ότι η κίνηση είναι επιταχυνόμενη σε συμφωνία με την επιταχυνόμενη κίνηση περιστροφής των τροχών, σχέσεις (5) και (6).



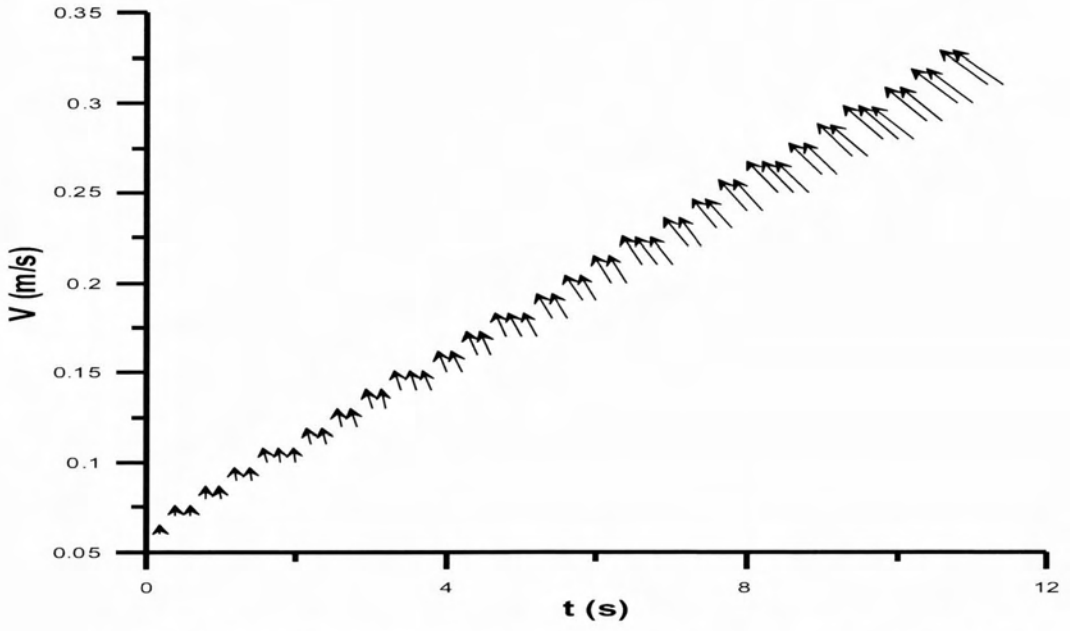
Εικόνα 7.7. Ταχύτητα κίνησης του robot σε σχέση με τον χρόνο $(n_r = 0.1 + 0.05t, n_l = 0.1 + 0.02t)$

Στην εικόνα 7.8. φαίνεται η διεύθυνση της ταχύτητας του robot για τα πρώτα 12s. Παρατηρούμε μεταβάλλεται από 0 έως -40° σε συμφωνία με το σχήμα 7.6.



Εικόνα 7.8. Διεύθυνση της ταχύτητας του robot σε σχέση με τον χρόνο $(n_r = 0.1 + 0.05t, n_l = 0.1 + 0.02t)$

Η εικόνα σχήμα 7.9. αποτελεί συνδυασμό των εικόνων 7.7. και 7.8., όπου φαίνεται τόσο το μέτρο όσο και η διεύθυνση της ταχύτητας σε σχέση με τον χρόνο.



Εικόνα 7.9. Μέτρο και διεύθυνση του διανύσματος της ταχύτητας του robot ($n_r = 0.1 + 0.05t$, $n_l = 0.1 + 0.02t$)

7.2.3. Σπειροειδής κίνηση.

Για τις ταχύτητες περιστροφής των τροχών του robot θεωρήθηκαν οι παρακάτω σχέσεις

$n_r = 0.1$ (7)

$n_l = 0.08 + 0.1t$ (8)

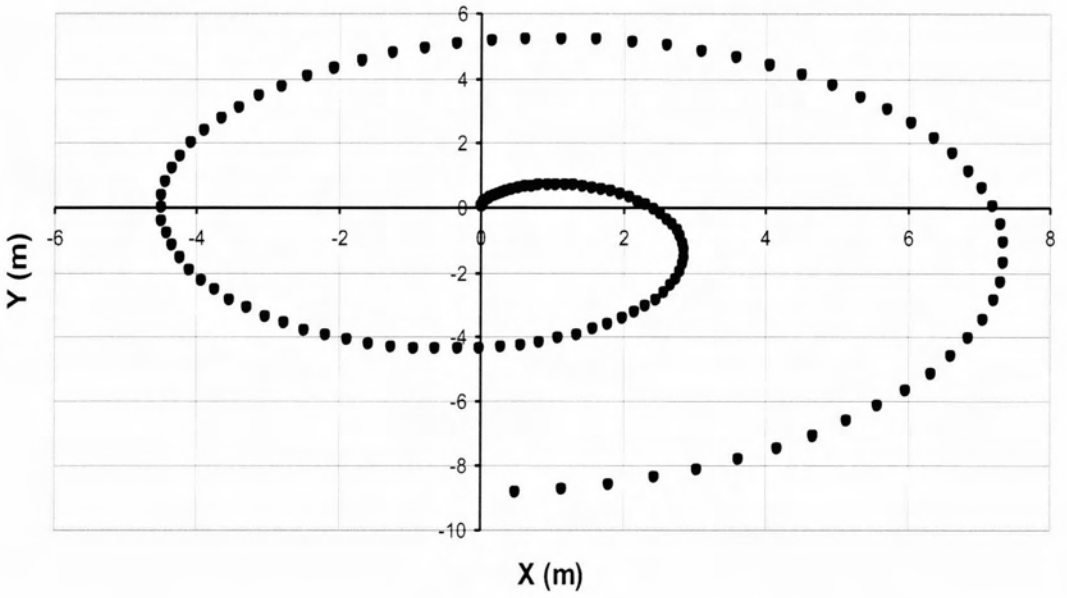
Από τις σχέσεις (7) και (8) φαίνεται ότι και ο δεξιός τροχός περιστρέφεται με σταθερή ταχύτητα ενώ ο αριστερός εκτελεί επιταχυνόμενη κίνηση περιστροφής, με αποτέλεσμα το robot να κινηθεί προς τα δεξιά με έντονα καμπυλόγραμμη κίνηση.

Στον πίνακα 7.3 φαίνονται μερικές τιμές των παραμέτρων που υπολογίστηκαν και συγκεκριμένα η ταχύτητα περιστροφής κάθε τροχού, οι συντεταγμένες της θέσης του robot καθώς και το μέτρο και η διεύθυνση της ταχύτητας του.

Πίνακας 7.3. Τιμές στροφών δεξιού και αριστερού τροχού, συντεταγμένων (X,Y), διεύθυνσης και μέτρου της ταχύτητας (φ,V) του robot.

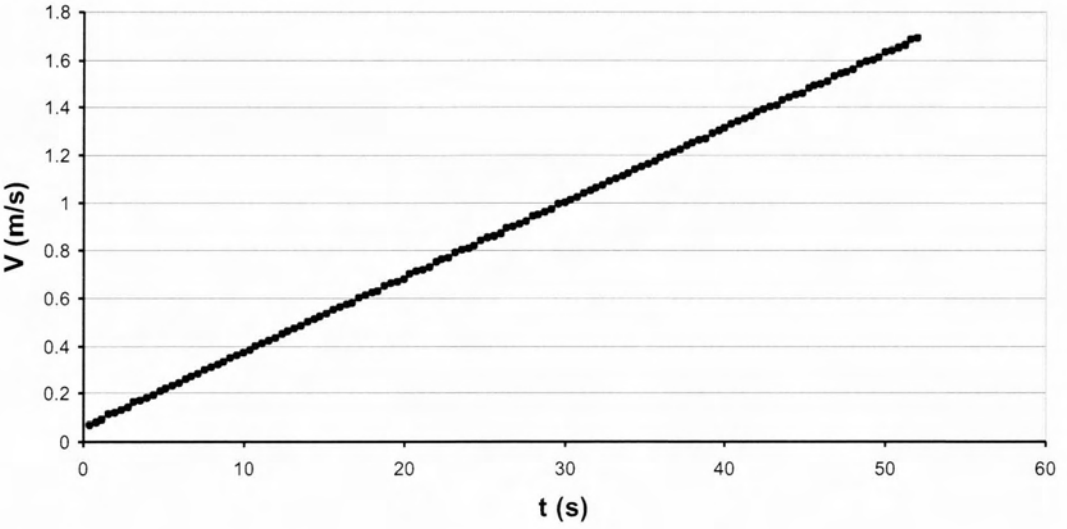
Time Step	Time	Nr	Nl	X	Y	φ	V
1	0.4	0.1	0.12	0	0.03	-2.4	0.07
2	0.8	0.1	0.16	0.01	0.06	-7.2	0.08
3	1.2	0.1	0.2	0.01	0.1	-12	0.09
4	1.6	0.1	0.24	0.03	0.14	-16.8	0.11
5	2	0.1	0.28	0.04	0.18	-21.6	0.12
6	2.4	0.1	0.32	0.07	0.23	-26.4	0.13
7	2.8	0.1	0.36	0.1	0.28	-31.2	0.14
8	3.2	0.1	0.4	0.13	0.33	-36	0.16
9	3.6	0.1	0.44	0.18	0.38	-40.8	0.17
10	4	0.1	0.48	0.23	0.43	-45.6	0.18
11	4.4	0.1	0.52	0.29	0.48	-50.4	0.19
12	4.8	0.1	0.56	0.36	0.53	-55.2	0.21
13	5.2	0.1	0.6	0.43	0.57	-60	0.22
14	5.6	0.1	0.64	0.52	0.61	-64.8	0.23
15	6	0.1	0.68	0.61	0.65	-69.6	0.24
16	6.4	0.1	0.72	0.71	0.67	-74.4	0.26
17	6.8	0.1	0.76	0.81	0.69	-79.2	0.27
18	7.2	0.1	0.8	0.93	0.71	-84	0.28
19	7.6	0.1	0.84	1.05	0.71	-88.8	0.3
20	8	0.1	0.88	1.17	0.7	-93.6	0.31
21	8.4	0.1	0.92	1.29	0.68	-98.4	0.32
22	8.8	0.1	0.96	1.42	0.65	-103.2	0.33
23	9.2	0.1	1	1.56	0.61	-108	0.35
24	9.6	0.1	1.04	1.69	0.55	-112.8	0.36

Στην εικόνα 7.10 φαίνεται η θέση του robot στο καρτεσιανό επίπεδο. Παρατηρούμε ότι η κίνηση του είναι επιταχυνόμενη σπειροειδής προς τα δεξιά σε συμφωνία με τις εξισώσεις (7) και (8).



Εικόνα 7.10. Θέση του robot στο επίπεδο ($n_r = 0.1t, n_l = 0.08 + 0.1t$)

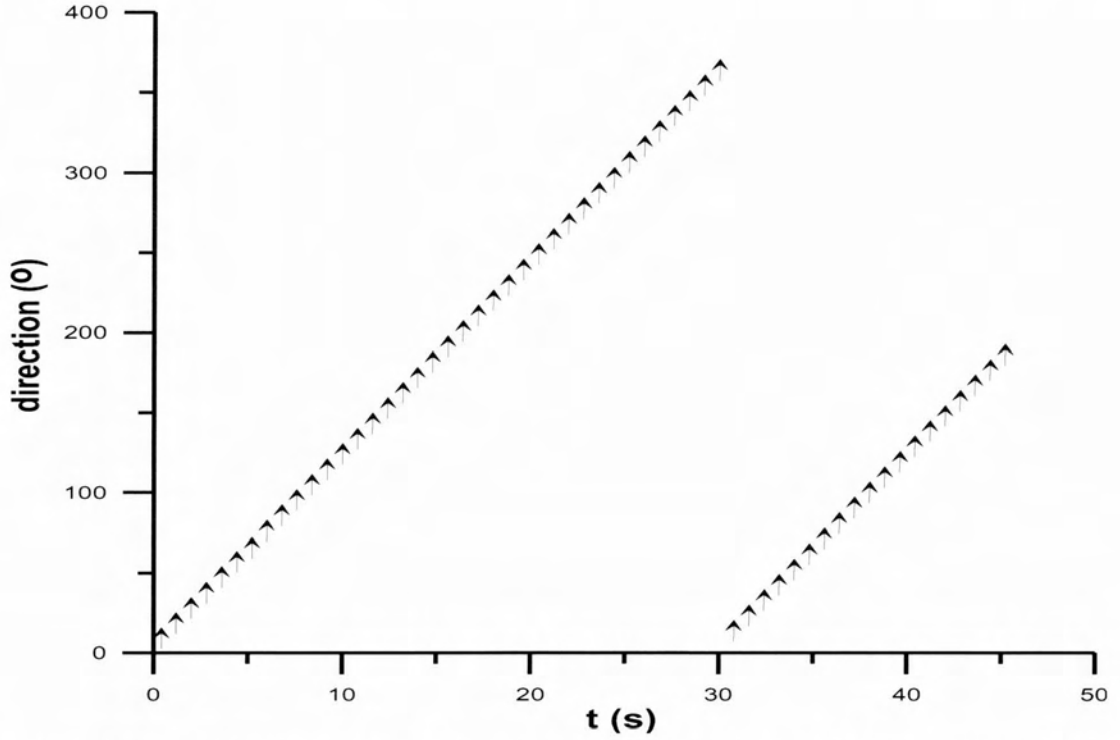
Στην εικόνα 7.11. φαίνεται το μέτρο της ταχύτητας του robot όπου και εδώ παρατηρούμε ότι η κίνηση είναι επιταχυνόμενη.



Εικόνα 7.11. Ταχύτητα κίνησης του robot σε σχέση με τον χρόνο ($n_r = 0.1t, n_l = 0.08 + 0.1t$)

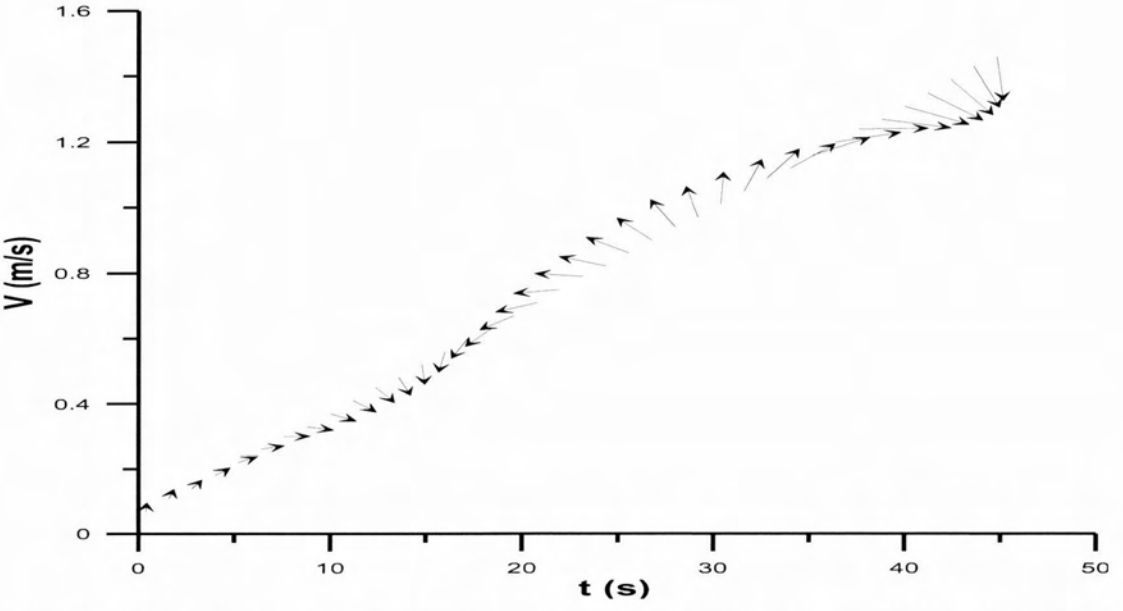
Στην εικόνα 7.12. φαίνεται η διεύθυνση της ταχύτητας του robot. Παρατηρούμε ότι βρίσκεται σε συμφωνία με το σχήμα 7 όπου φαίνεται ότι στα πρώτα

30s εκτελεί μια ολόκληρη (0-360°) περιστροφή και στα επόμενα 20s μισή (0-180°) ακόμη.



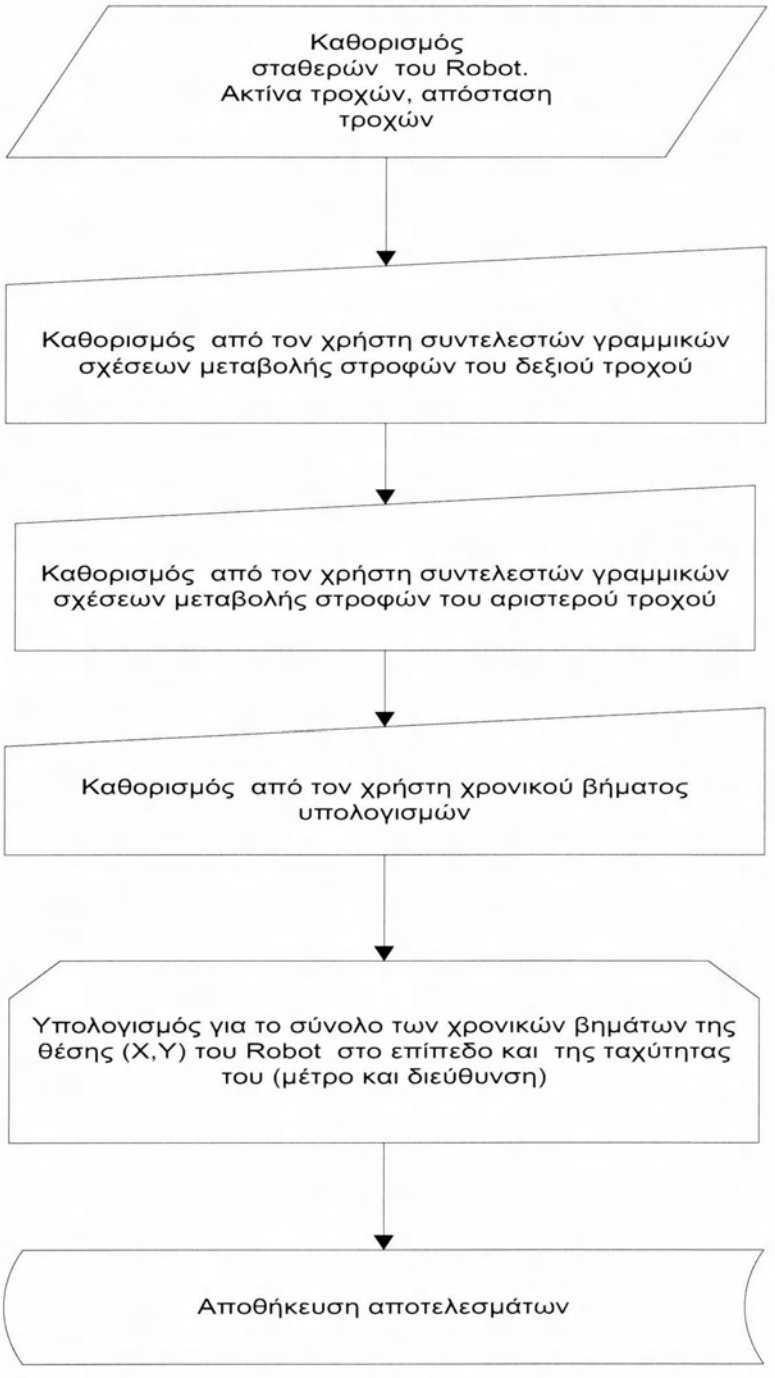
Εικόνα 7.12. Διεύθυνση της ταχύτητας του robot σε σχέση με τον χρόνο $(n_r = 0.1t, n_l = 0.08 + 0.1t)$

Η εικόνα 7.13 αποτελεί συνδυασμό των εικόνων 7.11 και 7.12, όπου φαίνεται τόσο το μέτρο όσο και η διεύθυνση της ταχύτητας σε σχέση με τον χρόνο.



Εικόνα 7.13. Μέτρο και διεύθυνση του διανύσματος της ταχύτητας του robot $(n_r = 0.1t, n_l = 0.08 + 0.1t)$

7.3. Λογικό διάγραμμα.



Εικόνα 7.14. Διάγραμμα ροής προγράμματος υπολογισμών θέσης και ταχύτητας.

Κώδικας σε γλώσσα FORTRAN

```
!     Program Robot ipologizei tis sintetagmenes kai th dieuthinsi kinisis
!
```

```
      program Robot_Thesis



      Real TimeStep,Fi,Meso,Nr,Nl,F1,F2,X,Y,X1,Y1,X2,Y2
      Real NlMax,NrMax
        integer Steps
        real Pi, R_trxn, D_trxn



        Pi=3.14

c-------Statheres tou Robot-------------------------------------
c     Diamteros troxwn(m)
       R_trxn=0.1
c     Apostasi troxwn (m)
       D_trxn=0.3
c     Max strofes aristerou troxou /min
       NlMax=0.81
c     Max strofes dexiou troxou /min
       NrMax=0.81
c---------------------------------------------------------------


c      open output file
      open(10,FILE='c:\robot\out')


c----------Kathorismos sinartisewn peristrofis troxwn-------------
c     Thewrite oti i taxitita peristrofis kathe troxou akolouthei tin
c     synartisi n(r)=a1+b1t dexios kai n(l)=a2+b2t aristeros troxos
c     o xristis toy programmatos kathorizei tis parametrous  a kai b
     print*,'-----------------------------------------------'
        print*, 'dwse parametrous a1 kai b1 gia ton dexio troxo '
        read(*,*) a1,b1

        print*, 'dwse parametrous a2 kai b2 gia ton aristero troxo'
```

```
      read(*,*) a2,b2

c-------------------------------------------------------------

c-------------kathorismos xronikou bimatos ypologismwn-----------
c    o xristis mporei na kathorisei kathe posa seconds tha
c    ektimate i thesi tou Robot. kathws kai posa
c    vimata tha periexei to output file.
c    Megisti timi vimatvn 1000
c    Shmeiwnetai oti gia tin oloklirwsi xrisimipoiite
c    xroniko vima 0.1 toy vimatos autou
 1      print*, 'dose xroniko vima ypologismwn (sec) kai arithmo vimatwn'
        read(*,*) TimeStep,Steps
        print*,'-----------------------------------------------'
c-------------------------------------------------------------------
c    Ypologismoi thesi Robot
c    Arxikes theseis (Fi, X,Y) theorountai 0

    Fi=0
       X=0
       Y=0
c        Gia ton synolo tvn apaitoumenvn vimatwn
       Do 2 i=1,STEPS,1
         i1=i
      Fi=0
         X=0
         Y=0
C     Ypologse me oloklirwsi ta Fi,X,Y
         Do 3 j=1,i1,1

c    Gia tin oloklirwsi theorite to embadon tou
c    paralilogramou.
c    Etsi prvta ypologizete to meso tis xronikis periodou
           Meso=(j-1)*TimeStep+(j*TimeStep-((j-1)*TimeStep))/2.
```

```
c          Sti sinexia i timi tis parametroy gia auto to simeio inai
             Nr=a1+b1*Meso
c     elegxow=s gia megisto arithmo strofwn
          if(Nr.gt.NrMax) then
           Nr=NrMax
          endif
          Nl=a2+b2*Meso
          if(Nl.gt.NlMax) then
           Nl=NlMax
          endif

             F1=(-2.)*Pi*R_trxn*Nl/D_trxn
          F2=2.*Pi*R_trxn*Nr/D_trxn

c     Telos ta embada atrizontai gia ton ypologismo
c     tou sinilikou embadou diladi to oloklirwma
             Fi=Fi+ (j*TimeStep-(j-1)*TimeStep)*(F1+F2)
c       print*, meso, F1+f2
c     Ypologise X kai Y  omoios me to Fi
          X1=(-1)*Pi*R_trxn*sin(F1+F2)*Nl
          X2=(-1)*Pi*R_trxn*sin(F1+F2)*Nr
          X=X+(j*TimeStep-(j-1)*TimeStep)*(X1+X2)

          Y1=Pi*R_trxn*cos(F1+F2)*Nl
          Y2=Pi*R_trxn*cos(F1+F2)*Nr
          Y=Y+(j*TimeStep-(j-1)*TimeStep)*(Y1+Y2)

 3          continue
c       Metaprepse ta rad se moires
          Fi=(Fi*360)/(2*Pi)
          if(abs(Fi).gt.360) then
           iFi=Fi/360
           Fi=Fi-(360*ifi)
```

```
          endif
          F=((F1+f2)*360)/(2*Pi)
          if(abs(F).gt.360) then
            iF=F/360
            F=F-(360*if)
          endif
c     grapse ta apotelesmata sto output arxeio
          write(10,4) i,i*TimeStep,Nr,Nl,X,Y,F,
     *    sqrt((X1+X2)**2+(Y1+Y2)**2)
4         format(i4,7(',',F10.2))
2       continue
      close(10)

          end
```

ΚΕΦΑΛΑΙΟ 8. ΣΥΜΠΕΡΑΣΜΑΤΑ

Με το πέρας τις παρούσας εργασίας είχα αποκτήσει πλέον τις πρώτες μου εμπειρίες με τα κινητά ρομπότ και τις εφαρμογές τους, τα οποία πλέον αποτελούν έναν σημαντικό τομέα έρευνας και ανάπτυξης και βρίσκουν εφαρμογές σε πολλούς τομείς τις βιομηχανίας και τις εκπαίδευσης. Πιο συγκεκριμένα το συγκεκριμένο μοντέλο που χρησιμοποιήθηκε (Pioneer p3 -dx) πιστεύω ήταν ιδανικό ώστε να κατανοήσω καλύτερα τις ιδιότητες των κινητών ρομπότ. Η πλατφόρμα αυτή έρευνας και ανάπτυξης έχει την απαραίτητη αρχιτεκτονική, την κατασκευή και λογισμικό και περιλαμβάνει όλα τα βασικά συστατικά για την ασφαλή πλοήγηση σε ένα πραγματικό περιβάλλον. Σε αρχικό στάδιο, όπως περιγράφεται και αναλυτικά στα παραπάνω κεφάλαια, το πρώτο βήμα ήταν να κατανοηθεί το hardware και το software του pioneer το οποίο περιλαμβάνει το σκελετό, το σύστημα οδήγησης, το μοτέρ και το σύστημα ελέγχου, του κωδικοποιητές της κίνησης, την μπαταρία και τον έλεγχο όλων αυτών σε ένα microcontroller πάνω στο ρομπότ και τον υπολογιστή (server) που χρησιμοποιείται.

Ακολούθησε η απαραίτητη συνδεσμολογία για να τεθεί σε λειτουργία η πλατφόρμα. Για αυτό το σκοπό χρησιμοποιήθηκαν μια οθόνη, ένα ποντίκι κι ένα πληκτρολόγιο, τα οποία και συνδέθηκαν στο πάνελ πρόσβασης στην δεξιά πλευρά του πίνακα ελέγχου για τον χειρισμό του ενσωματωμένου ηλεκτρονικού υπολογιστή του ρομπότ και ένα καλώδιο Ethernet για την σύνδεση του ηλεκτρονικού αυτού υπολογιστή με το διαδίκτυο. Επίσης για τον απομακρυσμένο έλεγχο χρησιμοποιήθηκε ένας άλλος φορητός ηλεκτρονικός υπολογιστής και ένα ειδικό καλώδιο για την σύνδεση του με το κινητό ρομπότ.

Επίσης μέσα από μια διαθέσιμη συλλογή από βιβλιοθήκες και εφαρμογές όπως Aria, ArNetworking, Mobile Eyes, Mapper3 Basic, Mobile Sim το μοντέλο δίνει τις δυνατότητες ασφαλής πλοήγησης και χαρτογράφησης, οι οποίες και πραγματοποιήθηκαν μέσα από μια σειρά πειραμάτων στο εργαστήριο, χρησιμοποιώντας πάντα με την βοήθεια των αντίστοιχων προγραμμάτων.

Το προγραμματιστικό κομμάτι είναι και από τα πιο βασικά και χρησιμοποιώντας την διαθέσιμη βιβλιοθήκη του κινητού ρομπότ (Aria.dll) δημιουργήθηκαν κώδικες που επιτρέπουν στο Pioneer να εκτελέσει σωστά κάποιες

εργασίες μέσω των αντίστοιχων εντολών. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε, για την σύνθεση των κωδικών, είναι η C++ και έγινε μέσα από το πακέτο λογισμικού Microsoft Visual C++, αν και υπάρχει η δυνατότητα ο εκάστοτε χρήσης να προγραμματίσει σε Java ή Python.

Οι κώδικες αυτοί περιλαμβάνονται στο πέμπτο και έκτο κεφάλαιο και στο παράρτημα τις εργασίας. Οι αρχικοί κώδικες που δημιουργήθηκαν προκαλούσαν απλώς την κίνηση του ρομπότ ή της ενσωματωμένης κάμερας που διέθετε, πάντα μέσω από μιας σειράς εντολών της διαθέσιμης βιβλιοθήκης. Έπειτα, έχοντας πλέον εξοικειωθεί καλύτερα με τον προγραμματισμό δημιουργήθηκαν οι πιο σύνθετοι και μακροσκελείς κώδικες που βρίσκονται στο παράρτημα τις εργασίας. Με παρόμοιο τρόπο μέσα από μια αλληλουχία εντολών το ρομπότ θα μπορούσε να εκτελέσει πιο περίπλοκες εργασίες. Αρχικά το ζήτημα ήταν να εντοπιστεί κάποιος πιθανός στόχος με τους ενσωματωμένους αισθητήρες (τα οχτώ σόναρ και το λέιζερ) που διαθέτει το συγκεκριμένο μοντέλο και έπειτα ο εντοπισμός να επικεντρώνεται σε συγκεκριμένες χρωματικές αποχρώσεις κι έτσι έγινε χρήση της κάμερας του ρομπότ και του αντίστοιχου προγράμματος χειρισμού της (ACTS).

Στη συνέχεια με βάση τις αρχές του αυτόματου ελέγχου, το επόμενο βήμα ήταν να δημιουργηθεί ένα πρόγραμμα που θα έλεγχε την θέση και την ταχύτητα του ρομπότ και θα πραγματοποιούσε τους απαραίτητους υπολογισμούς των σχετικών σφαλμάτων.

Οι κώδικες αυτοί κατόπιν εφαρμόστηκαν στην πράξη μέσα από πειράματα και βρίσκονται στο συνοδευτικό με την εργασία CD, για να μπορέσει, ο κάθε νέος χρήστης που επιθυμεί να προγραμματίσει τα δικά του προγράμματα, να έχει κάποιο σημείο αναφοράς.

Εν κατακλείδι, η εμπειρία έρευνας πάνω στα κινητά ρομπότ αποδείχτηκε άκρως ενδιαφέρουσα και όπως διαπιστώθηκε στην πορεία της εργασίας οι δυνατότητες και εφαρμογές τους πραγματικά αναρίθμητες. Επίσης έγινε και η πρώτη επαφή με τις βασικότερες δυνατότητες των εν λόγω πλατφόρμων, όπως η χαρτογράφηση, ο εντοπισμός, η αποφυγή εμποδίων, η ασφαλής πλοήγηση κ.ο.κ. και της ανάπτυξης περαιτέρω εφαρμογών, μέσω του κατάλληλου προγραμματισμού.

ΚΕΦΑΛΑΙΟ 9. ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Τα σύγχρονα ρομπότ μπορούν να αναπτύξουν μηχανικές και νοητικές ικανότητες, που στα παλαιότερα χρόνια θεωρούντο ότι ανήκαν στη σφαίρα επιστημονικής φαντασίας. Η ανάπτυξή τους θα συνεχίσει να επεκτείνεται με ολοένα νέα είδη ρομπότ κατάλληλα για τη βιομηχανία, την επιστήμη και την καθημερινή ζωή του ανθρώπου. Ήδη σήμερα υπάρχουν ιπτάμενα μη επανδρωμένα οχήματα-ρομπότ, ρομποτικοί οδηγοί (ρομποτικά μπαστούνια) τυφλών, ρομπότ ποδοσφαιριστές, πολύποδα ρομπότ ανίχνευσης ηφαιστείων, σμήνη συνεργαζόμενων ρομπότ, ρομποτικά έντομα, κοκ. Η έρευνα και ανάπτυξη προς την κατεύθυνση αυτή συνεχίζεται αδιάκοπα, έχοντας ως βασική προτεραιότητα την ασφάλεια και άνεση του ανθρώπου και την απαλλαγή του από δύσκολες, επίπονες και επικίνδυνες εργασίες.

Ο διαρκώς αναπτυσσόμενος κλάδος των κινητών ρομπότ έχει βρει εφαρμογές τόσο στην βιομηχανία όσο και στην έρευνα. Ένα βιομηχανικό ρομπότ [38] καθορίζεται ως ένας αυτόματα ελεγχόμενος, επαναπρογραμματιζόμενος, πολλαπλός βραχίονας (βλέπε εικόνα 8.2) κατασκευασμένος με τρεις ή περισσότερους άξονες. Τυπικές εφαρμογές της ρομποτικής [39] είναι η συγκόλληση, οι βαφές, η συναρμολόγηση, η τοποθέτηση (όπως π.χ. οι συσκευασίες), ο έλεγχος προϊόντων, και οι δοκιμές. Και όλα αυτά με υψηλή αντοχή, ταχύτητα και ακρίβεια. Τα πιο συχνά χρησιμοποιούμενα ρομπότ είναι τα αρθρωτά και τα ρομπότ που χρησιμοποιούν τις καρτεσιανές συντεταγμένες

Μερικά ρομπότ προγραμματίζονται για την πιστή εκτέλεση συγκεκριμένων ενεργειών ξανά και ξανά (επαναλαμβανόμενες πράξεις) χωρίς μεταβολές και με υψηλό βαθμό ακρίβειας. Οι δράσεις αυτές καθορίζονται από προγραμματισμένες ρουτίνες που καθορίζουν την κατεύθυνση, την επιτάχυνση, την ταχύτητα, την επιβράδυνση, και την απόσταση από μια σειρά συντονισμένων κινήσεων.

Άλλα ρομπότ είναι πολύ πιο ευέλικτα ως προς τον προσδιορισμό του αντικειμένου με το οποίο ενασχολούνται, ή ακόμα και την εργασία που πρέπει να εκτελεστεί στο ίδιο το αντικείμενο, το οποίο μπορεί ακόμα να χρειαστεί να προσδιοριστεί από το ίδιο το ρομπότ. Για παράδειγμα, για πιο ακριβή καθοδήγηση, τα ρομπότ συχνά περιέχουν υποσυστήματα μηχανικής όρασης που ενεργούν ως "μάτια", συνδεόμενα με ισχυρούς υπολογιστές ή ελεγκτές

(controllers). Η τεχνητή νοημοσύνη, ή ό,τι μοιάζει με αυτή, γίνεται όλο και πιο σημαντικός παράγοντας στα σύγχρονα βιομηχανικά ρομπότ.

Στον τομέα της έρευνας πολλά κινητά ρομπότ γίνονται αντικείμενο μελέτης από επιστήμονες φοιτητές αλλά και απλούς ανθρώπους προκειμένου να μελετηθούν οι δυνατότητες που διαθέτουν αλλά και οι ευρύτερες εφαρμογές τους.

Πιο συγκεκριμένα στην περίπτωση του pioneer p3-dx, το κινητό ρομπότ το οποίο μελετήθηκε στο εργαστήριο, διαθέτει δυνατότητες που το καθιστούν ένα ιδιαιτέρως χρήσιμο εργαλείο στους προαναφερθέντες τομείς τόσο της βιομηχανίας όσο και της έρευνας. Με την ικανότητα να αντιλαμβάνεται τον χώρο χαρτογραφώντας και να κινείτε σε αυτόν με προσοχή, αποφεύγοντας πιθανά εμπόδια μπορεί να χρησιμεύσει σε χώρους της βιομηχανίας σε αποθήκες σε στρατιωτικές βάσεις κ.λ.π.

Βασκά πλεονετήματα του είναι, ότι μέσω της κάμερας, μπορεί να αντιλαμβάνεται χρωματικές αποχρώσεις και να αναλύει έτσι το κάθε αντικείμενο που βλέπει. Επίσης όπως φαίνεται και στις παρακάτω εικόνες το Pioneer P3-DX επιδέχεται εκτός από κάμερα και βραχίονα, που είναι άκρως σημαντικός στην συναρμολόγηση, τοποθέτηση και άλλες εργασίες, όπως ειπώθηκε και στις παραπάνω παραγράφους, άλλα μπορεί επιπρόσθετα να αναλύει και να αναπαράγει ήχους.

Εικόνα 9.1. Ικανότητα όρασης.

Εικόνα 9.2. Ικανότητα αφής με ενσωματωμένο βραχίονα.

Εικόνα 9.3. Ικανότητα ήχου και ομιλίας.

Όσον αφορά τον τομέα του προγραμματισμού και εκεί οι δυνατότητες είναι απεριόριστες, αφού η βιβλιοθήκη που διαθέτει το κινητό ρομπότ είναι πλήρης και περιλαμβάνει όλες τις απαραίτητες εντολές. Επαφίεται λοιπόν στις ικανότητες και την φαντασία του κάθε προγραμματιστή να δημιουργήσει προγράμματα, που να εκμεταλλεύονται στο έπακρο τις ικανότητες της πλατφόρμας. Επίσημαίνεται ότι πρόσθετο λογισμικό και αναβαθμίσεις των υπαρχουσών εκδόσεων υπάρχει και στο διαδίκτυο, έτσι ώστε να επαυξάνονται διαρκώς οι προγραμματιστικές δυνατότητες.

ΚΕΦΑΛΑΙΟ 10. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] http://en.wikipedia.org/wiki/Mobile_robot

[2] http://www.ias.uwe.ac.uk/Robots/gwonline/gwonline.html

[3] http://www.planetary.org/blog/article/00002395/

[4] http://en.wikipedia.org/wiki/Lunokhod_programme

[5] Vehicles: experiments in synthetic psychology MIT press

[6] http://www.societyofrobots.com/robot_arm_tutorial.shtml

[7] http://en.wikipedia.org/wiki/Ernst_Dickmanns

[8] http://marsprogram.jpl.nasa.gov/MPF/mpf/edl/edl1.html

[9] http://aiboplus.sourceforge.net/

[10] http://www.swarm-bots.org/

[11] http://www.darpagrandchallenge.com/

[12] http://www.bostondynamics.com/robot_bigdog.html

[13] http://www.it.uom.gr/project/client_server/theoria1.htm

[14] Pioneer 3 operations manual

[15] Pioneer P3-DX datasheet

[16] http://robots.mobilerobots.com/wiki/ARIA

[17] http://robots.mobilerobots.com/wiki/MobileSim

[18]http://www.ing.unibs.it/~arl/docs/documentation/Aria%20documentation/Current/MobileEyes/doc/README.txt

[19] http://www.mobilerobots.com/software/mapper3.aspx

[20] http://robots.mobilerobots.com/wiki/ARCOS

[21] http://robots.mobilerobots.com/wiki/ARNL,_SONARNL_and_MOGS

[22] http://robots.mobilerobots.com/wiki/SAV

[23] ACTS user manual

[24] Pioneer 3 operations manual

[25],[26] SonARNL WITH MobileEyes installation and operation manual

[27] ACTS user manual

[28] Controlling the pioneer p3 dx robots at Csil by Amanda Whitbrook

[29] http://www.mobilerobots.com/software/aria.aspx

[30] http://en.wikipedia.org/wiki/Software_development_kit

[31] Aria reference

[32] http://courseware.mech.ntua.gr/ml23419/robotics_pdf/Ch7.pdf

[33] Ν.Ι. Κρικέλης, ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΑΥΤΟΜΑΤΟ ΕΛΕΓΧΟ

[34] K. Ogata, MODERN CONTROL ENGINEERING

[35] Mobile Robots, Herman Bruyninckx

[36] Marcello Restelli Dipartimento di electronica e informazione Politecnico di Milano

[37] Computational Intelligence and Robotics Laboratory- Course Manual EL5206

[38] http://www.societyofrobots.com/robot_arm_tutorial.shtml και Ρομπότ και άνθρωπος: Μισός αιώνας συμβίωσης Σπύρος Τζαφέστας, Ομότιμος καθηγητής ΣΗΜΜΥ- Ε.Μ.Πολυτεχνείου

[39] http://el.m.wikipedia.org/wiki/Βιομηχανικά_ρομπότ

ΠΑΡΑΡΤΗΜΑ. ΚΩΔΙΚΕΣ

Στις παρακάτω σελίδες περιγράφεται ο πρώτος κώδικάς που δημιουργήθηκε, ώστε το ρομπότ (pioneer p3 dx) να μπορεί, χρησιμοποιώντας τα sonars που διαθέτει, να εντοπίζει έναν κινητό στόχο και να μπορεί να κατευθύνεται προς αυτόν, προσαρμόζοντας ταυτόχρονα και την ταχύτητα του έτσι ώστε να αποφεύγονται οποιεσδήποτε συγκρούσεις.

```
#include "Aria.h"
#include <stdio.h>

#include <time.h>



void moveRobot(ArRobot* rob, int degrees , int move){
        if(move==-1){
                rob->setVel(-400);
        }
        else if(move==1){
                rob->setVel(400);
        }
        else if (move==0){
                rob->setVel(0);
        }
        if(degrees!=0){
                rob->setDeltaHeading(degrees);
        }
}
int getdirection(ArRobot* rob,int &turn ,int &move){
         ArSensorReading* sens;
         FILE *tmpfile = fopen("SonarTest.txt","a");
         time_t rawtime = time(NULL);
         fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));
```

```
int sonar = 0,range = 10000;
for(int i =0 ; i<8 ;i++){
        sens = rob->getSonarReading(i);
        int tmpRange = sens->getRange();
        fprintf (tmpfile , "Sonar # %d Value %d \n",i,tmpRange);
        if(tmpRange<range){
                sonar = i;
                range = tmpRange;
        }
}



        if(range==5000){
        turn = 0;
        move=0;
}



        else {
        if(sonar==0){
                turn = 90;
                move = 0;
        }
        else if(sonar==1){
                turn = 50;
                move = 0;
        }

        else if(sonar==2){
                turn = 30;
                move = 0;
        }
```

```
  else if(sonar==5){
        turn = -30;
        move = 0;
}


  else if(sonar==6){
        turn = -50;
        move = 0;
}


 else if(sonar==7){
        turn = -90;
        move = 0;
}


        else{
        if((range>2000)&(range<3000)){
              turn = 0;
              move = 0;
        }
        else if(range<2000){
              turn = 0;
              move = -1;
        }
        else{
              turn = 0;
              move = 1;
        }
}
```

```
        }
        fprintf (tmpfile , "Closest Sonar # %d Value %d \n",sonar,range);
        fclose(tmpfile);
        return 0;
}
int getsickDirections(ArSick *sic,int &turn ,int &move){
        const std::list<ArSensorReading *>*readinglist;
        std::list<ArSensorReading *>::const_iterator it;
        int i =-1;
        FILE *tmpfile = fopen("LaserTest.txt","a");
        time_t rawtime = time(NULL);
        fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));

        int degrees = 0 , range =10000;
        readinglist= sic->getRawReadings();



for (it = readinglist->begin(); it != readinglist->end(); it++)
        {
                i++;
                int tmpRange = (*it)->getRange();
                int tmpDegrees = (*it)->getSensorTh();
                //fprintf (tmpfile , "Sonar # %d Value %d \n",i,tmpRange);
                fprintf (tmpfile ,"%d LaserReading = %d Angle =
%d",i,tmpRange,tmpDegrees);
                if(tmpRange<range){
                        degrees = tmpDegrees;
                        range = tmpRange;
                }
        }

        if((degrees<-20)||(degrees>20)){
                turn = degrees;
```

```
        }
        else{
                turn = 0;
        }

        if((range>2000)&(range<3000)){
                move = 0 ;
        }
        else if(range<2000){
                move = -1;
        }
        else{
                move = 1;
        }
        fclose(tmpfile);
        return 0;
}
int main(int argc, char **argv)
{
        ArRobot robot;
        ArKeyHandler keyHandler;
        ArSonarDevice sonarDev;
        ArSick sick;
        Aria::init();

        ArSimpleConnector connector(&argc, argv);
        if (!connector.parseArgs() || argc > 1)
        {
                connector.logOptions();

                return 1;
        }

        if (!connector.connectRobot(&robot))
```

```
        {
            printf("Could not connect to robot... exiting\n");
            Aria::shutdown();
            return 1;
        }




        Aria::setKeyHandler(&keyHandler);
        robot.attachKeyHandler(&keyHandler);
        robot.addRangeDevice(&sonarDev);
        robot.addRangeDevice(&sick);

        robot.runAsync(true);
        robot.enableMotors();
        int a = 0;

        //robot.setVel(400);
        while(true){
            //-----Sonar test
            int degrees,vel;
            int i = getdirection(&robot,degrees,vel);
            //getsickDirections(&sick,degrees,vel);
            FILE *tmpfile = fopen("AriaTest.txt","a");
            time_t rawtime = time(NULL);
            fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));
            fprintf (tmpfile , "Returned Degrees %d Vel %d
\n\n",degrees,vel);
            fclose(tmpfile);
            moveRobot(&robot,degrees,vel);



            int mseconds = 1000;
```

```
            clock_t goal = mseconds + clock();
            while (goal > clock());
            //a++;
        }



        robot.waitForRunExit();
        Aria::shutdown();
      return 0;
}
```

Παρακάτω περιγράφεται ο κώδικας που δημιουργήθηκε, ώστε το ρομπότ να εντοπίζει μέσω της ενσωματωμένης ptz κάμερας και του προγράμματος ACTS, ένα συγκεκριμένο χρώμα και το ακολουθεί προσαρμόζοντας την ταχύτητα του, ανάλογα, ώστε να αποφευχθούν οποιεσδήποτε συγκρούσεις.

```
#include "Aria.h"
#include <stdio.h>
#include <iostream>
#include <time.h>
using namespace std;


// Follow is an action that moves the robot toward the largest blob.
// this is a ArAction class that has an event that is fired periodicaly

class Follow : public ArAction //Class Decleration {

public:


 enum State {
  NO_TARGET,
```

```
    TARGET,
  };
  Follow(ArACTS_1_2 *acts, ArVCC4 *camera);
  ~Follow(void);

  ArActionDesired *fire(ArActionDesired currentDesired);
  bool setChannel(int channel);
  State getState(void) { return myState; }
  enum {
    WIDTH = 160,
    HEIGHT = 120
  };

protected:
  ArActionDesired myDesired;

  ArACTS_1_2 *myActs;
  ArVCC4 *myCamera;
  ArTime myLastSeen;
  State myState;
  int myChannel;
  int myMaxTime;
};

//Follow Constractor
Follow::Follow(ArACTS_1_2 *acts, ArVCC4 *camera) :

    ArAction("Follow", "Follows the largest blob.")
{
  myActs = acts;
  myCamera = camera;
  myChannel = 0;
  myState = NO_TARGET;
  setChannel(1);
```

```
  myLastSeen.setToNow();
  myMaxTime = 1000;
}

Follow::~Follow(void) {}//Destractor




ArActionDesired *Follow::fire(ArActionDesired currentDesired)// Actual action

{
  ArACTSBlob blob;
  ArACTSBlob largestBlob;
  bool flag = false;
  int numberOfBlobs;
  int blobArea = 10;
  double xRel, yRel;
  myDesired.reset();
  numberOfBlobs = myActs->getNumBlobs(myChannel);  //Get the number of
Blobs found

  if(numberOfBlobs != 0)   //Get the largest blob and the current time

    {
      for(int i = 0; i < numberOfBlobs; i++)



          {
            myActs->getBlob(myChannel, i + 1, &blob);
            if(blob.getArea() > blobArea) blob area
```

```
            {
              flag = true;


              blobArea = blob.getArea();
            largestBlob = blob;
            }
          }
      myLastSeen.setToNow();
    }



 if (myLastSeen.mSecSince() > myMaxTime) // if more than 1 sec have
passed
    {
      if(myState != NO_TARGET) ArLog::log(ArLog::Normal, "Target Lost");

      myState = NO_TARGET;
    }
  else
    {
      if(myState != TARGET) ArLog::log(ArLog::Normal, "Target Aquired");


      myState = TARGET;
    }



if(TARGET && flag == true)   // If we have a target go to it.
```

```
{
  xRel = (double)(largestBlob.getXCG() - WIDTH/2.0)  / (double)WIDTH;


  yRel = (double)(largestBlob.getYCG() - HEIGHT/2.0) / (double)HEIGHT;

  if(!(ArMath::fabs(yRel) < .20))


     {
      if (-yRel > 0)
        myCamera->tiltRel(1);
      else
        myCamera->tiltRel(-1);
     }

  if (ArMath::fabs(xRel) < .10)

     {
      myDesired.setDeltaHeading(0);  deltaheading

     }
  else
     {
      if (ArMath::fabs(-xRel * 10) <= 10)



        myDesired.setDeltaHeading(-xRel * 10);
      else if (-xRel > 0)
        myDesired.setDeltaHeading(10);
      else
        myDesired.setDeltaHeading(-10);
```

```
        }

    myDesired.setVel(200);
    return &myDesired;
   }
  else // No target so stop.
   {
    myDesired.setVel(0);
    myDesired.setDeltaHeading(0);
    return &myDesired;
   }
}


bool Follow::setChannel(int channel)
{
 if (channel >= 1 && channel <= ArACTS_1_2::NUM_CHANNELS)
 {
  myChannel = channel;
  return true;
 }
 else
  return false;
}




// Main function
int main(int argc, char** argv)
{
        //Initialisation
```

Σύστημα ελέγχου θέσης και ταχύτητας

Ο τελευταίος κώδικάς που δημιουργήθηκε έχει σκοπό να μετρήσει το σχετικό σφάλμα μεταξύ της επιθυμητής γωνίας που θέλουμε να στρίψει το ρομπότ και της πραγματικής γωνίας που στρίβει και αντίστοιχα το σχετικό σφάλμα μεταξύ επιθυμητής και πραγματικής ταχύτητας. Έχουμε να κάνουμε λοιπόν με ένα απλό γραμμικό σύστημα αυτομάτου ελέγχου μιας εισόδου και μιας εξόδου.

```
Aria::init();
ArRobot robot;
ArKeyHandler keyHandler;
ArSonarDevice sonar;
ArVCC4 vcc4 (&robot);
ArACTS_1_2 acts;
ArArgumentParser argParser(&argc, argv);
argParser.loadDefaultArguments();
ArSimpleConnector simpleConnector(&argParser)    si;

//Connection  if (!Aria::parseArgs())
{
  Aria::logOptions();
  keyHandler.restore();
  Aria::shutdown();
  return 1;
}
aractionlimiter to


ArActionLimiterForwards limiter("speed limiter near", 300, 600, 250);
ArActionLimiterForwards limiterFar("speed limiter far", 300, 1100, 400);
ArActionLimiterBackwards backwardsLimiter;
ArActionConstantVelocity stop("stop", 0);
ArActionConstantVelocity backup("backup", -200);
```

```
Follow Follow(&acts, &vcc4);


Aria::setKeyHandler(&keyHandler);

robot.attachKeyHandler(&keyHandler);

robot.addRangeDevice(&sonar);

if (!simpleConnector.connectRobot(&robot))


{
  printf("Could not connect to robot... exiting\n");
  keyHandler.restore();
  Aria::shutdown();
  return 1;
}

acts.openPort(&robot);



vcc4.init();

ArUtil::sleep(1000);

robot.setAbsoluteMaxTransVel(400);

robot.comInt(ArCommands::ENABLE, 1);

robot.comInt(ArCommands::SOUNDTOG, 0);
```

```
 ArUtil::sleep(200);



 robot.addAction(&limiter, 100);
 robot.addAction(&limiterFar, 99);
 robot.addAction(&backwardsLimiter, 98);
 robot.addAction(&Follow, 77);
 robot.addAction(&backup, 50);
 robot.addAction(&stop, 30);

 robot.runAsync(true);  //Run the robot asyncronus




//Start Logging
 FILE *tmpfile = fopen("AriaCameraPositionTest.txt","a");



while(true){
        double x = robot.getX();
        double y = robot.getY();
        double vel = robot.getVel();
        time_t rawtime;
        time(&rawtime);
        fprintf (tmpfile , "Current Time %s \n",ctime(&rawtime));
        fprintf (tmpfile , "Current Position X: %f Y: %f Velocity: %f
\n\n",x,y,vel);
        int mseconds = 1000;
        clock_t goal = mseconds + clock();
        while (goal > clock());
 }
```

```
  fclose(tmpfile);
  robot.waitForRunExit();
  Aria::shutdown();
   return 0;
}



#include "Aria.h"
#include <stdio.h>
#include <iostream>
#include <time.h>
#include <fstream>
#include <sstream>
#include <limits>
using namespace std;

class Log
{
public:
        static void Message(ArRobot robot,int newDelta);
        static void MessageDelta(int newDelta);
        static void MessageNew(double x,double y,double vel,double com-
pass,double th,int newDelta);
        static void MessageChange(int angle,int distor,int gccoef,int dist);

};
void Log::MessageChange(int angle,int distor,int gccoef,int dist){
        string my_string;
        std::ostringstream ss;
        ss << "<State>"<<endl<<"<Rotation Angle>"<<angle<<"</Rotation



Angle>"<<endl
```

```
                <<"<Distortion>"<<distor<<"</Distortion>"<<endl
                <<"<gccoef>"<<gccoef<<"</gccoef>"<<endl
                <<"<Dis-
tance>"<<dist<<"</Distance>"<<endl<<"</State>"<<endl;
        ofstream file;
        file.open("AriaErrorLog.xml",ios_base::out | ios_base::app);
        my_string = ss.str();
        file << my_string << endl;
        file.close();
}
void Log::Message(ArRobot robot,int newDelta)
{

        double x = robot.getX();
        double y = robot.getY();
        double vel = robot.getVel();
        double compass = robot.getCompass();
        double th = robot.getTh();
        time_t rawtime;
        time(&rawtime);




string my_string;
        std::ostringstream ss;
        ss                                                              <<
"<State>"<<endl<<"<Time>"<<endl<<ctime(&rawtime)<<"</Time>"<<endl<<"
<X>"<<x<<"</X>"<<endl<<"<Y>"<<y<<"</Y>"<<endl<<"<Velocity>"<<vel<<"<
/Velocity>"<<endl;
        ss                              <<                      "<Com-
pass>"<<compass<<"</Compass>"<<endl<<"<Th>"<<th<<"</Th>"<<endl<<"
```

```
<NewDHead-
ing>"<<newDelta<<"</NewDHeading>"<<endl<<"</State>"<<endl;
        ofstream file;
        file.open("AriaErrorLog.xml",ios_base::out | ios_base::app);
        my_string = ss.str();
        file << my_string << endl;
        file.close();
}
void Log::MessageNew(double x,double y,double vel,double compass,double
th,int newDelta)
{

        char* msg;
        time_t rawtime;
        time(&rawtime);
        string my_string;
        std::ostringstream ss;
        ss                                                              <<
"<State>"<<endl<<"<Time>"<<endl<<ctime(&rawtime)<<"</Time>"<<endl<<"
<X>"<<x<<"</X>"<<endl<<"<Y>"<<y<<"</Y>"<<endl<<"<Velocity>"<<vel<<"<
/Velocity>"<<endl;
        ss                              <<                      "<Com-
pass>"<<compass<<"</Compass>"<<endl<<"<Th>"<<th<<"</Th>"<<endl<<"
<NewDHead-
ing>"<<newDelta<<"</NewDHeading>"<<endl<<"</State>"<<endl;
        ofstream file;
        file.open("AriaErrorLog.xml",ios_base::out | ios_base::app);
        my_string = ss.str();
        file << my_string << endl;
        file.close();
}
void Log::MessageDelta(int newDelta){
        time_t rawtime;
        time(&rawtime);
```

```
        string my_string;
        std::ostringstream ss;
        ss                                                                    <<
"<State>"<<endl<<"<Time>"<<endl<<ctime(&rawtime)<<"</Time>"<<endl<<"
<NewDHead-
ing>"<<newDelta<<"</NewDHeading>"<<endl<<"</State>"<<endl;
        ofstream file;
        file.open("AriaErrorLog.xml",ios_base::out | ios_base::app);
        my_string = ss.str();
        file << my_string << endl;
        file.close();
}


void rotate(ArRobot* rob, int degrees,int distor,int gccoef,Log* mylog){

        Log myLog = Log();
        myLog.MessageChange(degrees,distor,gccoef,0);

        rob->setDeltaHeading(degrees+distor*gccoef);



        while(rob->isHeadingDone()==false){
                ArUtil::sleep(200);
                myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),0);
        }
        myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),888);
        //myLog.Message(*rob,0);
}
```

```
void moveRob(ArRobot* rob, int distance,Log* mylog){
	rob->move(distance);
	Log myLog = Log();
	myLog.MessageChange(0,0,0,distance);
	while(rob->isMoveDone()==false){
		ArUtil::sleep(200);
		myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),0);
	}
	myLog.MessageNew(rob->getX(),rob->getY(),rob->getVel(),rob-
>getCompass(),rob->getTh(),999);
}

int checkDigit(string inpstr){
	const char * csdf = inpstr.c_str();
	if(strspn(csdf,"-0123456789")==inpstr.length()){
		return 1;
	}
	else{
		return 0;
	}
}




int main(int argc, char** argv)
{

 Log myLog = Log();
 Aria::init();
 ArRobot robot;
 ArKeyHandler keyHandler;
```

```
ArArgumentParser argParser(&argc, argv);
argParser.loadDefaultArguments();
ArSimpleConnector simpleConnector(&argParser);




if (!Aria::parseArgs())
{
  Aria::logOptions();
  keyHandler.restore();
  Aria::shutdown();
  return 1;
}

  if (!simpleConnector.connectRobot(&robot))
{
  printf("Could not connect to robot... exiting\n");
  //keyHandler.restore();
  Aria::shutdown();
  return 1;
}

ArUtil::sleep(1000);
robot.runAsync(true);
robot.enableMotors();
bool check = true;


while(check)
{
        string in="",dummy="";
        int angle=0,distortion=0,gccoef=0,turnangle=0,disttance=0;
```

```
cout << "Please Choose action. m for move , r for rotate , q for quit: ";
string mystr1;
getline (cin,mystr1);
stringstream(mystr1)>> in;
string mystr;
if(in.compare("m")!=0&&in.compare("r")!=0&&in.compare("q")!=0){
        continue;
}
else if(in.compare("r")==0)
{

        system("cls");

        cout << "Enter Rotation Angle :";
        getline (cin,mystr);
        if(checkDigit(mystr)>0){
                stringstream(mystr) >> angle;
        }
        else{
                cout<<"Not a valid number";
                cout<<endl;
                continue;
        }

        cout << "Enter Distortion Angle :";
        getline (cin,mystr);
        if(checkDigit(mystr)>0){
                stringstream(mystr) >> distortion;
        }
        else{
                cout<<"Not a valid number";
                cout<<endl;
                continue;
        }
```

```
                cout << "Enter gc Coeficient :";
                getline (cin,mystr);
                if(checkDigit(mystr)>0){
                        stringstream(mystr) >> gccoef;
                }
                else{
                        cout<<"Not a valid number";
                        cout<<endl;
                        continue;
                }

                rotate(&robot,angle,distortion,gccoef,&myLog);

        myLog.MessageNew(robot.getX(),robot.getY(),robot.getVel(),robot.


getCompass(),robot.getTh(),111);
                system("cls");
                continue;
        }
        else if(in.compare("m")==0){
                system("cls");
                cout << "Enter Distance :";
                getline (cin,mystr);

                if(checkDigit(mystr)>0){
                        stringstream(mystr) >> disttance;
                        cout<<"Starting Moving for "<<disttance;
                        cout<<endl;
                }
                else{
                        cout<<"Not a valid number";
                        cout<<endl;
                }
```

```
                moveRob(&robot, disttance,&myLog);


        //myLog.MessageNew(robot.getX(),robot.getY(),robot.getVel(),robot.get
Compass(),robot.getTh(),0);
                continue;
        }
        else{
                check = false;
        }
 }
  robot.disconnect();
  Aria::shutdown();
  return 0;

}
```