ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη αλγορίθμου πλοήγησης Robot σε εσωτερικό χώρο με χρήση κάμερας οροφής

Αλέξανδρος Φραγκότσης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος

Δελήμπασης Κωνσταντίνος – Επίκουρος Καθηγητής

Πλαγιανάκος Βασίλης – Αναπληρωτής Καθηγητής

Λαμία, 2014

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη αλγορίθμου πλοήγησης Robot σε εσωτερικό χώρο με χρήση κάμερας οροφής

Αλέξανδρος Φραγκότσης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος

Δελήμπασης Κωνσταντίνος – Επίκουρος Καθηγητής

Πλαγιανάκος Βασίλης – Αναπληρωτής Καθηγητής

Λαμία, 2014

Ανάπτυξη αλγορίθμου πλοήγησης Robot σε εσωτερικό χώρο με χρήση κάμερας οροφής

Αλέξανδρος Φραγκότσης

Τριμελής Επιτροπή:

Δελήμπασης Κωνσταντίνος – Επίκουρος Καθηγητής

Πλαγιανάκος Βασίλης – Αναπληρωτής Καθηγητής

Σανδαλίδης Χάρης – Επίκουρος Καθηγητής

Περίληψη

Καθώς ο κόσμος της τεχνολογίας εξελίσσεται η ιδέα του να υπάρχουν κάποια ρομπότ μέσα στο περιβάλλον μας φαντάζει όλο και πιο κοντινή. Η αλληλεπίδραση ανθρώπων και ρομπότ θα πρέπει να είναι φυσική και δεν θα πρέπει οι λειτουργίες ενός ρομπότ να επηρεάζουν την καθημερινότητα των ανθρώπων. Μέρος των καθημερινών λειτουργιών των ρομπότ μπορεί να είναι η υποβοήθηση των ανθρώπων στις καθημερινές τους δραστηριότητες.

Σκοπός αυτής της εργασίας είναι να φτιάξουμε ένα σύστημα το οποίο θα μπορεί να αναγνωρίσει την ανθρώπινη δραστηριότητα και στην συνέχεια μέσω ενός ρομπότ να ακολουθεί κάποιον άνθρωπο σε πραγματικό χρόνο χωρίς να χάνεται ή να μπερδεύεται.

Abstract

As the technological world evolves the idea of having personal robots inside our houses seems more and more probable. The human – robot interaction should be natural and the functions of a robot should not get in the way of the human's everyday life. Part of robot's everyday functions could be assisting humans with their everyday activities.

The purpose of this thesis is to develop a system that it could detect the human activity inside a house and send a robot follow a human in real-time and without getting lost or confused.

Λέξεις Κλειδιά - Keywords

Τεχνητή Νοημοσύνη , Ρομποτική , Τμηματοποίηση Εικόνας , Παρακολούθηση Ανθρώπου , Fisheye Camera , Υπολογιστική Όραση, Java , OpenCV, Human Tracking , Video Segmentation, Robot

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους γονείς και την οικογένεια μου που είναι δίπλα μου όλα αυτά τα χρόνια. Τον κ. Δελήμπαση και κ. Πλαγιανάκο που μου έδωσαν την ευκαιρία να ασχοληθώ με αυτήν την εργασία, καθώς και όλους τους καθηγητές μου. Ευχαριστώ τον Νίκο Γιαχούδη για την πολύτιμη βοήθεια του με το Ρομπότ. Τέλος να ευχαριστήσω και να αφιερώσω αυτήν την εργασία στην Ελευθερία που τόσα χρόνια με στηρίζει σε ότι κάνω.

Περιεχόμενα

Κεφάλαιο 1

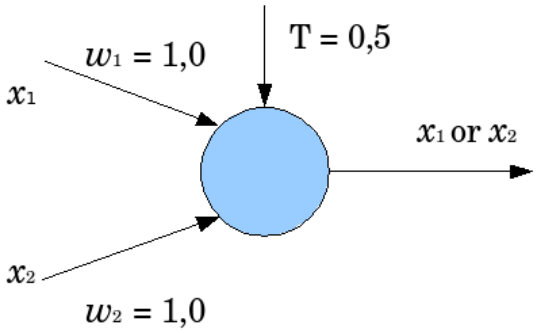
Εισαγωγή

1.1 Τεχνητή Νοημοσύνη

Με τον όρο Τεχνητή Νοημοσύνη – ΤΝ (Artificial Intelligence – AI) μπορούμε να χαρακτηρίσουμε την προσπάθεια που γίνεται από τους επιστήμονες να προσδώσουν στοιχεία νοημοσύνης σε μηχανές. Τέτοια στοιχεία μπορεί να είναι η μίμηση της ανθρώπινης συμπεριφοράς ως προς την μάθηση , προσαρμοστικότητα, εξαγωγή συμπερασμάτων, κατανόηση από τα συμφραζόμενα, επίλυση προβλημάτων κ.α. Η τεχνητή νοημοσύνη δεν αφορά μόνο τον κλάδο της πληροφορικής αλλά αποτελεί σημείο όπου κλάδοι όπως πληροφορική, μηχανική, ψυχολογία, φιλοσοφία, νευρολογία και γλωσσολογία συνεργάζονται ούτος ώστε να πετύχουν το καλύτερο δυνατό αποτέλεσμα. Οι συνηθέστεροι τρόποι για την ανάπτυξη τεχνητής νοημοσύνης είναι η ανάπτυξη αριθμητικών μοντέλων που προσομοιώνουν πραγματικές διαδικασίες.

Η τεχνητή Νοημοσύνη χρησιμοποιείται σε ένα ευρύ φάσμα εφαρμογών όπως επίλυση προβλημάτων, μηχανική μάθηση, ανακάλυψη γνώσης, καθώς προβλήματα μηχανικής όρασης, επεξεργασία φυσικής γλώσσας ή και ρομποτική. Κλασσικό παράδειγμα συνεργασίας των παραπάνω πεδίων αποτελεί το Turing Test το οποίο αναπτύχθηκε από τον Alan Turing και σύμφωνα με το οποίο ο εξεταστής κάνει ερωτήσεις ταυτόχρονα σε έναν άνθρωπο και μία μηχανή και παίρνει απαντήσεις στις ερωτήσεις του ταυτόχρονα χωρίς να γνωρίζει όμως από ποιον έρχονται οι απαντήσεις. Εάν τελικά ο εξεταστής δεν μπορεί να διακρίνει εάν η απάντηση έρχεται από τον άνθρωπο ή την μηχανή τότε θεωρείται ότι η μηχανή πέρασε το τεστ και θεωρείται ευφυής.

Παρόλο που η Τεχνητή Νοημοσύνη φαντάζει για μια επιστήμη του παρόντος αλλά πιο πολύ του μέλλοντος, επιστήμονες ασχολούνται με την ανάπτυξη ευφυών μηχανών εδώ και πολλά χρόνια και συγκεκριμένα κατά την δεκαετία του 1940 εμφανίστηκε η πρώτη μαθηματική περιγραφή τεχνητού νευρωνικού δικτύου(εικόνα 1), η οποία χρησιμοποιείται μέχρι και σήμερα για την επίλυση αριθμητικών προβλημάτων αν και έχει αρκετά περιορισμένες δυνατότητες. Στην συνέχεια το 1950 ο Alan Turing πρότεινε το Turing Test.[1] Το 1956 η τεχνητή νοημοσύνη θεμελιώθηκε σαν επιστημονικό πεδίο και αργότερα εκείνη την χρονιά παρουσιάστηκε το Logic Theorist ένα πρόγραμμα το οποίο στηριζόταν σε κανόνες λογικής για να αποδεικνύει μαθηματικά θεωρήματα.



Εικόνα 1: Διάγραμμα δομής και λειτουργίας ενός τεχνητού νευρώνα

Καθώς η Τεχνητή Νοημοσύνη άρχισε να αναπτύσσεται όλο και περισσότεροι επιστήμονες καταπιάνονταν και προσπαθούσαν να αναπτύξουν αλγορίθμους νοημοσύνης, σημαντικό ρόλο έπαιξε και η ανάπτυξη της γλώσσας LISP[2] το 1958 από τον Τζον Μακάρθι ,η οποία ήταν η πρώτη γλώσσα συναρτησιακού προγραμματισμού, σε αυτήν την γλώσσα αναπτύχθηκαν διάφορες εφαρμογές καθώς και γενετικοί αλγόριθμοι και εξελίξεις των νευρωνικών δικτύων όπως το δίκτυο perceptron.

Το 1966 ιδρύεται το πρώτο Εργαστήριο Μηχανικής Νοημοσύνης στο Ενδιμβούργο αλλά στο τέλος της δεκαετίας του 60 όμως το όλο θέμα με την Τεχνητή Νοημοσύνη άρχισε να ξεφουσκώνει διότι παρόλες τις προσπάθειες της επιστημονικής κοινότητας, δεν κατάφεραν να κάνουν τα νευρωνικά δίκτυα να λύσουν πιο πολύπλοκα προβλήματα και γενικά υπήρχε μια στασιμότητα μέχρι τα μέσα της δεκαετίες του 70 όπου με χρήση νέων μαθηματικών μεθόδων κατάφεραν να φτιάξουν μηχανές Τεχνητής Νοημοσύνης που αποθήκευαν γνώση για κάποιον συγκεκριμένο τομέα και είχαν την δυνατότητα να παράγουν λογικά συμπεράσματα πολύ γρήγορα. Εκείνη την εποχή εμφανίστηκε και η γλώσσα προγραμματισμού Prolog[3] η οποία αναπτύχθηκε από τον Alain Colmeraue και έδωσε νέα ώθηση στους επιστήμονες.

Κατά την δεκαετία του 90 η Τεχνητή Νοημοσύνη άνθιζε καθώς αναπτύσσονταν παράλληλα με μαθηματικά εργαλεία στατιστικής και μηχανικής όπως τα Hidden Markov Model και τα Κάλμαν φίλτρα. Αυτήν την δεκαετία μάλιστα με την άνοδο των ηλεκτρονικών υπολογιστών αλλά και του ίντερνετ η Τεχνητή Νοημοσύνη παίρνει ακόμα μεγαλύτερη φήμη τόσο από ταινίες του κινηματογράφου που φαντάζονται το μέλλον της Τεχνητής Νοημοσύνης αλλά τρανό παράδειγμα αποτελούν τα βιντεοπαιχνίδια τα οποία αποτελούν μια από τις σημαντικότερες εφαρμογές της τεχνητής νοημοσύνης μέχρι και σήμερα, προσομοιώνοντας τον αντίπαλο πολλές φορές τόσο δύσκολα ώστε ο χρήστης να μην μπορεί να κερδίσει. Ακόμα ένα παράδειγμα δημοσιότητας της Τεχνητής Νοημοσύνης είναι τα παιχνίδια σκάκι του υπερ-υπολογιστή Deep Blue της IBM το οποίο το 1997 κατάφερε να κερδίσει τον πρωταθλητή Γκάρι Κασπάροφ[4] η οποία αποτελεί την πρώτη φορά που υπολογιστής κατάφερε να κερδίσει παγκόσμιο πρωταθλητή. Η IBM για αυτό το γεγονός δημιούργησε μια σελίδα την οποία διατηρεί ακόμα και σήμερα.[5]

Παρόλο λοιπόν που ο τομέας της Τεχνητής Νοημοσύνης μετράει μόνο 50 χρόνια ζωής, έχουν γίνει τεράστια άλματα ως προς την ανάπτυξη του και

αποτελούν και σήμερα έναν από τους κύριους τομείς ανάπτυξης και έρευνας με τεράστια ποσά επιχορηγήσεων ανά τον κόσμο.

1.2 Ρομποτική

Η Ρομποτική είναι μια επιστήμη που συνδυάζει γνώσεις και έρευνα τόσο σε μηχανική και υλικό, όσο και σε προγραμματισμό και λογισμικό.[6] Έχει σαν σκοπό δηλαδή την δημιουργία μιας μηχανής, την συναρμολόγηση της αλλά και τον προγραμματισμό του για να εκτελεί μια συγκεκριμένη ή μια ομάδα εργασιών σε τέτοιο βαθμό ώστε να μπορεί να αντικαταστήσει τον άνθρωπο π.χ. η συναρμολόγηση μιας μητρικής κάρτας γίνεται από μηχανές – Ρομπότ και χωρίς την παρέμβαση ανθρώπων. Τέτοιες μηχανές συνήθως δεν αλληλοεπιδρούν με το περιβάλλον τους και απλά εκτελούν μια συγκεκριμένη εργασία με μεγάλη ταχύτητα και μικρό κόστος, γι' αυτό επιλέγονται και σε διάφορα εργοστάσια να αντικαταστήσουν το εργατικό δυναμικό. Υπάρχουν και τα πιο εξελιγμένα Ρομπότ τα οποία είναι εφοδιασμένα με αισθητήρες, είναι πιο κοντά στον άνθρωπο και έχουν σκοπό να βοηθάνε τους ανθρώπους στην καθημερινότητα τους.

Τα πιο εξελιγμένα Ρομπότ δέχονται δεδομένα από τους αισθητήρες τους και τα επεξεργάζονται, με αυτόν τον τρόπο μπορούν να βγάλουν συμπεράσματα για το περιβάλλον τους όπως την θέση που βρίσκονται, αν υπάρχουν άνθρωποι στο περιβάλλον τους κ.α. Εκτός βέβαια και από αυτού του είδους Ρομπότ, υπάρχουν και τα ρομποτικά μέλη τα οποία συνδέονται σε ανθρώπους που συνήθως έχουν χάσει κάποιο μέλος του σώματος τους και τα οποία ναι μεν είναι μηχανικά αλλά λειτουργούν με ερεθίσματα που δέχονται από τα συνδεδεμένα νεύρα του ανθρώπου. Τέτοια παραδείγματα είναι ρομποτικά χέρια και πόδια.

Εικόνα 2: (Αριστερά) Το ρομποτικό χέρι Shadow Hand (Δεξιά) Το ρομποτικό χέρι Bebionic3

Ο όρος Ρομπότ εμφανίζεται το 1921 όπου πρωτοεμφανίζεται σε ένα θεατρικό έργο επιστημονικής φαντασίας του Τσέχου συγγραφέα Κάρελ Τσάπεκ και προέρχεται από την σλαβική λέξη robota που σημαίνει εργάτης. [7] Ωστόσο 40 χρόνια αργότερα το 1961 τίθεται σε λειτουργία το πρώτο ρομπότ για βιομηχανική χρήση όπου χρησιμοποιήθηκε για να σηκώνει βαριές πλάκες μετάλλου και να τις στοιβάζει.

Η ιδέα βέβαια της κατασκευής μηχανών που δρουν μόνα τους ή βοηθάνε τον άνθρωπο προϋπήρχε από τα αρχαία χρόνια καθώς υπάρχουν περιγραφές “αυτομάτων” από τον 3ο αιώνα π.Χ. Έχουν σωθεί διάφορες περιγραφές όπως αυτή που χρονολογείται στο 420 π.Χ. όπου περιγράφει ένα μηχανικό πουλί που μπορεί να πετάει, 1494 σχέδια ενός ανθρωπόμορφου Ρομπότ, 1738 μια μηχανική πάπια που μπορεί να φάει και να κουνήσει τα φτερά της, 1738 Ο Νικολά Τέσλα επιδεικνύει ένα τηλεκατευθυνόμενο σκάφος. Αργότερα το 1956 φτιάχνεται το πρώτο εμπορικό ρομπότ από την εταιρία Unimation του George Devol και Joseph Engelberger βασισμένο στις πατέντες του Devol. Από εκείνη την εποχή και μετά έχουν φτιαχτεί διάφορα Ρομπότ με διάφορες μορφές για σκοπούς όπως χειρουργική, εξερεύνηση δύσβατων περιοχών καθώς και του διαστήματος, ιπτάμενα, με ρόδες ή με πόδια, για οικιακό, εμπορικό ή στρατιωτικό σκοπό.

Παρόλο που τα ρομπότ μπορεί να διαφέρουν πολύ ανάλογα με τον τομέα και τον σκοπό που χρησιμοποιούνται, όλα έχουν κάποια κοινά χαρακτηριστικά όπως αποτελούνται από κάποια μηχανικά μέρη τα οποία είναι αποτέλεσμα του προβλήματος που θέλουμε να επιλυθεί. Επίσης όλα τα ρομπότ έχουν κάποια ηλεκτρικά μέρη όπως καλώδια, αισθητήρες, κυκλώματα, μπαταρίες κλπ. Τέλος όλα τα Ρομπότ περιέχουν κάποιας μορφής προγραμματισμό (κώδικα). Ένα πρόγραμμα το οποίο περιγράφει πώς το Ρομπότ θα αποφασίσει πότε ή πώς να αντιμετωπίσει μια κατάσταση. Ακόμα και τα τηλεκατευθυνόμενα ρομπότ που τα χειρίζεται ο χρήστης με κάποιο χειριστήριο, μόλις πατηθεί κάποιο κουμπί τότε θα τρέξει το μέρος του προγράμματος που ανταποκρίνεται στο συγκεκριμένο κουμπί. Π.χ. όταν ο χρήστης πατήσει το κουμπί για να κατευθύνει το ρομπότ προς τα μπροστά τότε θα λειτουργήσει το μέρος του προγράμματος που δίνει ρεύμα στους κινητήρες και τους κάνει να κινήσουν το ρομπότ.

Υπάρχουν 3 είδη προγραμμάτων ρομπότ. Τα τηλεχειριζόμενα τα οποία έχουν προ εγκαταστημένο ένα σετ από εντολές οι οποίες θα τρέξουν μόνο όταν το ρομπότ λάβει το σήμα από ένα τηλεχειριστήριο το οποίο τις περισσότερες φορές το χειρίζεται κάποιος άνθρωπος. Το δεύτερο είδος είναι τα προγράμματα τεχνητής νοημοσύνης. Τα ρομπότ με τέτοιου είδους προγράμματα αλληλοεπιδρούν με το περιβάλλον τους και επιλύουν προβλήματα / ξεπερνάνε εμπόδια χρησιμοποιώντας το προϋπάρχουν πρόγραμμα για να αποφασίσουν, καταλάβουν, να μάθουν ή να δημιουργήσουν μια λύση. Τέλος υπάρχει το υβριδικό μοντέλο το οποίο χρησιμοποιεί και την τηλεχειριζόμενη αλλά και την λειτουργία της τεχνητής νοημοσύνης. Για παράδειγμα ένα ρομπότ προσπαθώντας να λύσει ένα πρόβλημα μπορεί να καταλήξει σε 2 απαντήσεις και να βασιστεί στον χρήστη για να αποφασίσει την σωστή απάντηση.

1.3 Κάμερα με ευρυγώνιο φακό

Ο fisheye φακός αποτελεί υποκατηγορία των ευρυγώνιων φακών και προσφέρει οπτικό πεδίο 180^{o}.Οι κάμερες με ευρυγώνιο φακό γενικά προσφέρουν ένα εξαιρετικά ευρύ οπτικό πεδίο με ισχυρή οπτική παραμόρφωση στις άκρες και οι οποίοι φακοί χρησιμοποιούνται για να αποτυπώσουν πανοραμικές ή ημισφαιρικές εικόνες (εικόνα 3).[8] Οι φακοί αυτοί καταφέρνουν να αποτυπώσουν τόσο μεγάλες γωνίες θέασης με το να μην αποτυπώνουν εικόνες με ευθείες γραμμές του ορίζοντα, αυτό μπορεί να γίνει αντιληπτό σε μια φωτογραφία π.χ. ενός κτηρίου, εάν στρέψουμε τον φακό προς τα επάνω θα δούμε ότι οι άκρες του κτηρίου συγκλίνουν προς το κέντρο.

Εικόνα 3: Στιγμιότυπο από fisheye κάμερα τοποθετημένη στο ταβάνι

Ο όρος fisheye επινοήθηκε το 1906 από τον Αμερικάνο φυσικό Robert W. Wood βασισμένος στο πώς θα έβλεπε ένα ψάρι κάτω από το νερό. Η πρώτη πρακτική χρήση των φακών αυτών ήταν το 1920 στην μετεωρολογία για την μελέτη του σχηματισμού των σύννεφων. Αργότερα οι φακοί αυτοί άρχισαν να παράγονται μαζικά και να χρησιμοποιούνται σε φωτογραφήσεις το 1960 και θεωρούνται ιδανικές για να αποτυπώνουν μεγάλο οπτικό πεδίο μέσα σε μικρούς χώρους όπως μέσα σε μικρά δωμάτια και σπίτια.

Αυτοί οι φακοί εκτός από εμπορική χρήση , χρησιμοποιούνται πολύ σε επιστημονικά πεδία όπως Αστρονόμοι τις χρησιμοποιούν για να καταγράψουν την κάλυψη των νεφών και για να μετρήσουν δεδομένα ρύπανσης του περιβάλλοντος, Ιατροί της χρησιμοποιούν σε κάψουλες κατάποσης για να διακρίνουν τυχών ανωμαλίες στο εσωτερικό του σώματος σε όλη την διαδρομή από τον λάρυγγα ως το παχύ έντερο. Τέλος για στρατιωτικό λόγο όπως σε εξομοιωτές πτήσεις και εξομοιωτές μάχης χρησιμοποιούνται fisheye

φακοί για να δημιουργήσουν ένα ρεαλιστικό περιβάλλον για πιλότους και ελεγκτές εναέριας κυκλοφορίας ώστε να εκπαιδευτούν με τον καλύτερο δυνατό τρόπο.

1.4 Η Πτυχιακή

Ο τομέας της αναγνώρισης της ανθρώπινης δραστηριότητας τυγχάνει μεγάλης προσοχής τα τελευταία χρόνια με σκοπό την δημιουργία ενός περιβάλλοντος ηλεκτρονικής υποβοήθησης. Έχουν γίνει αρκετές έρευνες για τον σκοπό αυτό οι οποίες βασίζονται σε ανθρώπινα μοντέλα 3D ή τοπικούς περιγραφείς εικόνας οι οποίοι χρησιμοποιούν τόσο χωρικά όσο και χρονικά δεδομένα. Στο κοντινό μέλλον φαίνεται πως διάφορων τύπων Ρομπότ θα υπάρχουν σε κάθε σπίτι και θα αλληλοεπιδρούν καθημερινά με τους ανθρώπους και το περιβάλλον γύρω τους καθώς και θα υποβοηθούν τους ανθρώπους σαν μέρος των καθημερινών λειτουργιών τους. Για να συμβεί αυτό άνθρωποι και Ρομπότ θα πρέπει να είναι σε κοντινή απόσταση μεταξύ τους, επιπλέον θα πρέπει οι αλληλεπιδράσεις τους να συμβαίνουν φυσικά με έναν φιλικό και εύχρηστο προς τον άνθρωπο τρόπο. Για να συμβεί αυτό τα Ρομπότ θα πρέπει να είναι σε θέση να αναγνωρίσουν την τοποθεσία του ανθρώπου , να ξεχωρίσουν ποιος είναι ο άνθρωπος ο οποίος θα πρέπει να αλληλοεπιδράσουν καθώς και να αναγνωρίσουν το περιβάλλον γύρω τους, αυτό μπορεί να γίνει χρησιμοποιώντας πολλούς αισθητήρες μέσα σε ένα «έξυπνο» περιβάλλον οι οποίο συλλέγουν και αναλύουν δεδομένα σε πραγματικό χρόνο και τροφοδοτούν με αυτά τα Ρομπότ.[9]

Υπάρχουν όμως και ομάδες ανθρώπων οι οποίοι δεν μπορούν να αντέξουν οικονομικά την εγκατάσταση ενός ολόκληρου «έξυπνου» περιβάλλοντος και θέλουν μια οικονομική και ταυτόχρονα αποδοτική λύση. Γι' αυτόν τον λόγο σε αυτή την εργασία δόθηκε περισσότερη βαρύτητα στην ανάπτυξη λογισμικού παρά στην αγορά ακριβού εξοπλισμού. Έτσι σαν εξωτερικοί αισθητήρες χρησιμοποιούμε μια fisheye κάμερα τοποθετημένη στο ταβάνι ενός χώρου καθώς και τους ενσωματωμένους αισθητήρες του Ρομπότ οι οποίοι περιγράφονται στα χαρακτηριστικά του Ρομπότ.

Σκοπός αυτής της πτυχιακής εργασίας είναι η ανάπτυξη αλγορίθμου για την αναγνώριση ανθρώπων από εικόνες που λήφθηκαν μέσω μιας fisheye κάμερας τοποθετημένης στην οροφή και παρακολούθηση του ανθρώπου από ένα ρομπότ. Η εικόνα βασίζεται σε κάμερες fisheye οι οποίες χρησιμοποιούν σφαιρικό φακό, οι παράμετροι της κάμερας ορίζονται μόνο μια φορά, κατά την αρχικοποίηση χρησιμοποιώντας τοποθεσίες που ορίζονται από τον χρήστη τόσο στον πραγματικό χώρο όσο και στα καρέ τα οποία συλλαμβάνονται από το βίντεο. Έτσι κάθε pixel της εικόνας χαρτογραφείται σε σχέση με τον πραγματικό χώρο και τα δεδομένα αποθηκεύονται σε lookup tables για γρήγορη πρόσβαση. Το κάθε καρέ χρησιμοποιείται για την τμηματοποίηση της εικόνας, για την αναγνώριση του ανθρώπου , διόρθωση της τμηματοποίησης με αφαίρεση της σκιάς , αναγνώριση της θέσης του ρομπότ ώστε να μην συμπεριληφθεί στον υπολογισμό της θέσης και τέλος

υπολογισμός της θέσης του ανθρώπου στον πραγματικό χώρο με την βοήθεια των lookup tables από την fisheye camera και αποστολή τους στο ρομπότ ώστε, το ρομπότ , να κινηθεί προς τον άνθρωπο.

Στο κεφάλαιο 2 περιγράφεται αναλυτικότερα το υλικό που χρησιμοποιήθηκε ενώ στο κεφάλαιο 3 οι αλγόριθμοι που αναπτύχθηκαν. Στο κεφάλαιο 4 περιγράφονται τα πειράματα που σχεδιάστηκαν για να αποδειχθεί ότι μέσω των αλγορίθμων το Ρομπότ είναι ικανό να ακολουθήσει κάποιον άνθρωπο και παρουσιάζονται τα αποτελέσματα ενώ στο παράρτημα φαίνεται όλος ο κώδικας που αναπτύχθηκε και τα στοιχεία του Ρομπότ.

Κεφάλαιο 2

Υλικό και Μέθοδοι

Το υλικό που χρησιμοποιήθηκε για την πτυχιακή αυτή είναι η fisheye camera η οποία έχει οπτικό πεδίο 180 μοίρες, ένα ρομπότ PeopleBot [10] της εταιρίας mobilerobots και τέλος οι αλγόριθμοι αναπτύχθηκαν στην γλώσσα προγραμματισμού Java [11] χρησιμοποιώντας την βιβλιοθήκη ανοιχτού λογισμικού OpenCV[12]. Στην παρούσα εργασία θεωρούμε δεδομένη την ύπαρξη των lookup tables του μοντέλου της κάμερας καθώς και το ότι η κίνηση του Ρομπότ γίνεται αποστέλλοντας του απλά τις συντεταγμένες.

2.1 Η κάμερα

Ο φακός της κάμερας είναι τεχνολογίας fisheye με οπτικό πεδίο 180 μοίρες και είναι τοποθετημένη στο εσωτερικό χώρου σε ύψος 3.5 μέτρων από το δάπεδο. Από αυτήν την κάμερα λαμβάνεται ζωντανό βίντεο ή και εικόνες για την τμηματοποίηση και την αναγνώριση των ανθρώπων όπως περιγράφεται στην ενότητα 3. Η μεθοδολογία υπολογισμού της θέσης των ανθρώπων βασίζεται σε ένα παραμετρικό μοντέλο της εικόνας σύμφωνα με το οποίο κάθε σημείο του χώρου (x,y,z) αντιστοιχεί σε ένα pixel της εικόνας (i,j). Επιπρόσθετα κάθε pixel(i,j) μπορεί να συσχετιστεί με την κατεύθυνση του πεδίου το οποίο ορίζεται από 2 γωνίες: Το αζιμούθιο θ και το ύψος φ. Οι παράμετροι αυτοί του μοντέλου της fisheye κάμερας ορίζονται μόνο μία φορά κατά την λειτουργία της βαθμονόμησης. Το αποτέλεσμα των συσχετίσεων των pixel με το Αζιμούθιο και το ύψος αποθηκεύονται σε lookup tables για γρήγορη αναφορά κατά τον υπολογισμό των συντεταγμένων του ανθρώπου. Το μοντέλο της κάμερας μπορεί να περιγραφεί από την γενική μορφή ως εξής:

$$(j, i) = M(x, y, z) \quad (1)$$

Όπου τα (j,i) είναι τα pixel του frame της εικόνας και τα (x,y,z) οι πραγματικές συντεταγμένες του χώρου. Παρακάτω θα περιγραφεί ο αντίστροφος μετασχηματισμός δηλαδή έχοντας κάποια pixel (i,j) από το frame του βίντεο υπολογίζεται η κατεύθυνση της εικόνας η οποία ορίζεται από δύο γωνίες: Το αζιμούθιο θ και το ύψος φ

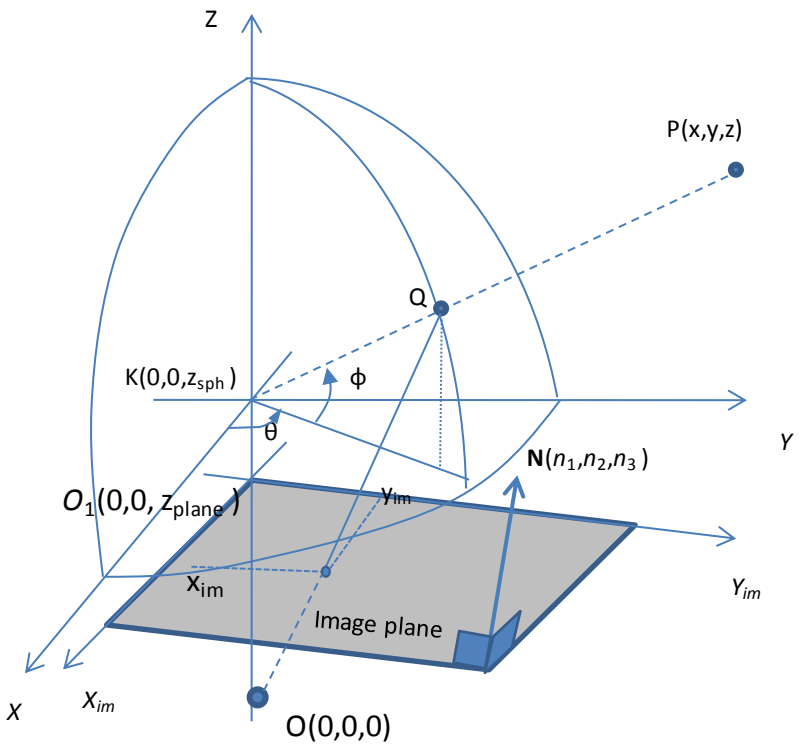
$$(\theta, \varphi) = M_1(i, j) \quad (2)$$

Ο ορισμός του μοντέλου της fisheye κάμερας βασίζεται την φυσική του σχηματισμού της εικόνας όπως περιγράφεται στα [13][14] και παρουσιάζεται [15]. Το μοντέλο αποτελείται από:

- Ένα σφαιρικό στοιχείο με ακτίνα R_0 με το κέντρο της $K(0,0,Z_{sph})$
- Το επίπεδο του αισθητήρα CMOS που ορίζεται καθώς περνάει από το $(0,0,Z_{plane})$ και από το μοναδιαίου μήκους κάθετο διάνυσμα n.

Για κάθε σημείο P με πραγματικές συντεταγμένες (x,y,z) ορίζουμε την διατομή της ευθείας KP με το σφαιρικό στοιχείο της fisheye κάμερας. Το σημείο P προβάλλεται στο κεντρικό σημείο προβολής (x_m,y_m) του Q στην εικόνα χρησιμοποιώντας το O(0,0,0) σαν κέντρο της προβολής, υποθέτοντας ότι η τοποθέτηση της κάμερας είναι τέτοια όπου το σχέδιο της εικόνας (δηλ. ο φακός της κάμερας) είναι κάθετος και ο άξονας του σφαιρικού φακού είναι σωστά ευθυγραμμισμένος. Επομένως είναι προφανές ότι όλα τα σημεία του πραγματικού χώρου που είναι πάνω στην ευθεία KP προβάλλονται στο ίδιο σημείο (x_m,y_m) στην εικόνα. Η ευθεία KP ορίζεται μοναδικά από τις γωνίες Αζιμούθιου θ και Ύψους φ της. Η ιδέα της fisheye κάμερας παρουσιάζεται στην εικόνα 1.

Η κάμερα δεν έχει κινούμενα μέρη επομένως η σχέση μεταξύ Z_{sph} και Z_{plane} ορίζει τον σχηματισμό της εικόνας. Αν σκεφτούμε το Z_{plane} σαν έναν αυθαίρετο αριθμό μικρότερο του R0 και να ορίσουμε $Z_{sph} = pZ_{plane}$ όπου το p είναι η κύρια παράμετρος του μοντέλου fisheye. Για να συνυπολογίσουμε την πιθανή μη σωστή ευθυγράμμιση του φακού καθώς και τον αισθητήρα της κάμερας που μπορεί να παρουσιάσει παραμορφώσεις στην εικόνα, προσθέτουμε δύο επιπλέων παραμέτρους την X και Y θέση του κέντρου του σφαιρικού φακού K(X_{sph},Y_{sph},Z_{sph}) σε σχέση με τον οπτικό άξονα της κάμερας. Τώρα οι παράμετροι της εικόνας της κάμερας αποτελούνται από n, p , X_{sph} και Y_{sph} . Η εικόνα 1 δείχνει την γεωμετρία του μοντέλου της fisheye κάμερας για X_{sph} = 0 και Y_{sph} = 0



Εικόνα 4: Η γεωμετρία του προτεινόμενου μοντέλου fisheye κάμερας

Η θέση κάθε σημείου της ευθείας KP, καθώς και το Q δίνονται από την σχέση

$$(Q_x, Q_y, Q_z) = (\lambda(x-x_{sph}), \lambda(y-y_{sph}), \lambda(z-z_{sph})) \quad (3)$$

όπου το λ είναι μια παράμετρος στο διάστημα [0,1]. Αν εισάγουμε την (3) στην εξίσωση του σφαιρικού οπτικού στοιχείου θα έχουμε:

$$(\lambda(x - x_{\text{sph}}) - x_{\text{sph}})^2 + (\lambda(y - y_{\text{sph}}) - y_{\text{sph}})^2 + (\lambda(z - z_{\text{sph}}) - z_{\text{sph}})^2 - R_0{}^2 = 0 \quad (4)$$

Η παράμετρος λ που ορίζει την θέση του Q μπορεί να βρεθεί λύνοντας την εξίσωση 4 και παίρνοντας ως λύση το αποτέλεσμα στο διάστημα [0,1]. Για να γενικοποιήσουμε το fisheye μοντέλο υποθέτουμε ότι το επίπεδο του αισθητήρα δεν είναι το XY επίπεδο αλλά ορίζεται ως το σημείο που διέρχεται από το $(0,0,Z_{plane})$ και είναι κανονικοποιημένο ως προς το διάνυσμα n = (n1,n2,n3). Τα στοιχεία του n συμπεριλαμβάνονται σαν παράμετροι της fisheye κάμερας όπου έχει περιγραφεί στο [16]. Στην συνέχεια μπορούμε να υπολογίσουμε το κεντρικό σημείο προβολής (x_m,y_m,z_m) του Q στην εικόνα ως εξής:

$$(mQx, zQy, mQz) \quad (5)$$

Όπου η παράμετρος m δίνεται από τον τύπο: $m = \dfrac{z_{plane} n_3}{Q_x n_1 + Q_y n_2 + Q_z n_3}$

Τέλος οι συντεταγμένες του σημείου προβολής (x_m,y_m,z_m) μπορούν να υπολογιστούν μετασχηματίζοντας γεωμετρικά το κεντρικό σημείο προβολής, με ομογενείς συντεταγμένες, έτσι ώστε το διάνυσμα n που είναι κανονικοποιημένο ως προς τον αισθητήρα, να γίνει παράλληλο στον Z άξονα.

$$\left(x_{im}, y_{im}, z_{im}, 1\right)^T = \mathbf{A}\left(mQ_x, mQ_y, mQ_z, 1\right)^T \quad (6)$$

όπου A είναι ο πίνακας που μετασχηματίζει το διάνυσμα (n_1,n_2,n_3) στο ∠ άξονα.

$$\mathbf{A} = \begin{pmatrix} \dfrac{\lambda}{|\mathbf{n}|} & \dfrac{-n_1 n_2}{\lambda|\mathbf{n}|} & \dfrac{-n_1 n_2}{\lambda|\mathbf{n}|} & 0 \\ 0 & \dfrac{\lambda}{n_3} & \dfrac{-n_2}{n_3} & 0 \\ \dfrac{n_1}{|\mathbf{n}|} & \dfrac{n_2}{|\mathbf{n}|} & \dfrac{n_3}{|\mathbf{n}|} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \lambda = \sqrt{n_2^2 + n_3^2}$$

Όταν το $x \to \pm\infty$ τότε το $Q_x \to R_0 \pm x_{sph}$ (το ίδιο ισχύει και για την συντεταγμένη y). Επομένως κάθε σημείο P με πραγματικές συντεταγμένες του χώρου $z>z_{sph}$ θα προβληθεί στην εικόνα στην θέση (x_m,y_m) η οποία είναι φραγμένη ως εξής:

$$x_{\text{im\_min}} \le x_{\text{im}} \le x_{\text{im\_max}} \text{ και } y_{\text{im\_min}} \le y_{\text{im}} \le y_{\text{im\_max}} \quad (7)$$

Όπου

$$\begin{aligned} x_{im\_\max} &= \lambda_+ \left(x_{sph} + R_0\right) \\ x_{im\_\min} &= \lambda_- \left(x_{sph} - R_0\right) \end{aligned}, \lambda_\pm = \frac{n_3 z_{pl}}{\left(x_{sph} \pm R_0\right) n_1 + \left(y_{sph} \pm R_0\right) n_2 + z_{sph} n_3}$$

Το θέση του pixel της εικόνας (i,j) που αντιστοιχεί στην προβολή της εικόνας (x_{im},y_{im}) υπολογίζεται από τον γραμμικό μετασχηματισμό

$$j = x_{im}\frac{R_{FoV}}{f_1} + CoD_x,\ f_1 = \frac{x_{im\_\max} - x_{im\_\min}}{2}$$
$$i = y_{im}\frac{R_{FoV}}{f_2} + CoD_y,\ f_2 = \frac{y_{im\_\max} - y_{im\_\min}}{2} \quad (8)$$

Όπου το (CoD_x,CoD_y) είναι το κέντρο της στρέβλωσης pixel που αντιστοιχεί στο ύψος φ=π/2 και R_{FoV} είναι η ακτίνα του κυκλικού οπτικού πεδίου (Field of View) όπως περιγράφεται στην επόμενη παράγραφο.

Στην περίπτωση ενός fisheye φακού, το [17] προτείνει το κέντρο ση) να είναι τοποθετημένο στο κέντρο του την περίπτωση μας είναι έτσι τοποθετημένο για ου προβάλλεται σχεδόν όλο το οπτικό πεδίο. Έτσι anny edge detector [18], χρησιμοποιώντας τυπική για να βρούμε δυνατές άκρες στην εικόνα οι οποίες είναι του κυκλικού οπτικού πεδίου. Έπειτα χρησιμοποιείται η η ελάχιστων τετραγώνων για να υπολογιστεί το κέντρο ς (CoD) και η ακτίνα του οπτικού πεδίου. Τα αποτελέσματα στην εικόνα 2. Αυτή η διαδικασία εφαρμόστηκε μόνο μία φορά μετά τάσταση της κάμερας.

Εικόνα 5: Ένα frame της fisheye κάμερας. Το κέντρο στρέβλωσης (CoD) και η ακτίνα του οπτικού πεδίου έχουν σημειωθεί για να χρησιμοποιηθούν στους μετέπειτα υπολογισμούς

2.2 Το Ρομπότ

Το ρομπότ που χρησιμοποιήθηκε για την εργασία αυτή είναι το pleBot [10] της εταιρίας mobilerobots. Χρησιμοποιεί λειτουργικό Linux, όλο που έχει την δυνατότητα να πλοηγηθεί με την ενσωματωμένη του α καθώς και με joystick στην παρούσα πτυχιακή τίποτα από αυτά δεν οποιήθηκε. Διαθέτει ενσωματωμένους αισθητήρες sonar [19] για την ιση και αποφυγή εμποδίων, τα προγράμματα που τρέχουν στο αι γραμμένα σε c και c++ γλώσσα προγραμματισμού. Λειτουργεί με α 12V. Περισσότερες λεπτομέρειες για τα χαρακτηριστικά του ανται στο παράρτημα.

Στο ρομπότ έχει εισαχθεί ο χάρτης του χώρου μία φορά κατά την αρχική του εγκατάσταση και στην αρχή κάθε πειράματος εισάγεται από τον χρήση η τρέχουσα θέση του ρομπότ. Η θέση εισάγεται μόνο μία φορά στην αρχή και στην συνέχεια το Ρομπότ ανανεώνει την θέση του σύμφωνα με την οδομετρία του καθώς και με τις ενδείξεις που παίρνει από τους αισθητήρες sonar καθώς προχωράει. Για την εργασία αυτή χρησιμοποιήθηκε το λογισμικό του Ρομπότ για την κίνηση του αλλαγμένο όμως σε κάποια σημεία ώστε να δέχεται ένα HTTP Connection, από το λογισμικό που αναπτύχθηκε. Μόλις γίνει η σύνδεση, ο αλγόριθμος αφού αναγνωρίσει την θέση του ανθρώπου στέλνει τις συντεταγμένες στο Ρομπότ και με την σειρά του αυτό κινείται προς αυτήν την κατεύθυνση. Σε αυτό το σημείο αξίζει να διευκρινιστεί ότι το Ρομπότ κινείται αυτόνομα χωρίς την παρεμβολή των αλγορίθμων. Μόνο του το λογισμικό του Ρομπότ αποφασίζει ποια πορεία πρέπει να ακολουθήσει για να φτάσει τον στόχο που του έχει σταλεί καθώς και την αποφυγή εμποδίων που θα αναγνωριστούν από τους αισθητήρες sonar και θα πρέπει να αποφευχθούν.

2.3 Το περιβάλλον ανάπτυξης

2.3.1 Προγραμματισμός του αλγορίθμου

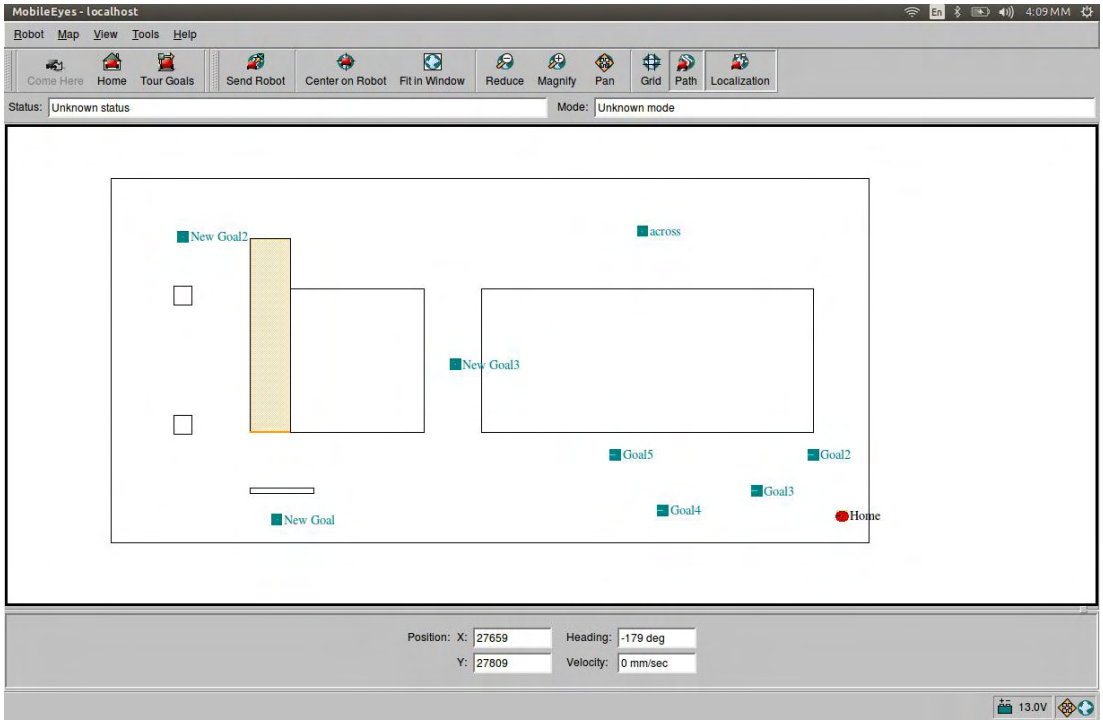
Για να αναπτυχθούν οι παρακάτω αλγόριθμοι χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java στο περιβάλλον Eclipse Kepler Service Release 2 [20] χρησιμοποιώντας το Java SE Development Kit 7u65. Για τον χειρισμό και επεξεργασία των εικόνων και βίντεο που λήφθηκαν από την κάμερα χρησιμοποιήθηκαν τόσο η Java όσο και η βιβλιοθήκη ανοιχτού λογισμικού OpenCV 2.4.8 [12]. Ο υπολογιστής που δοκιμάστηκαν οι παρακάτω αλγόριθμοι είναι ένα Macbook Pro early 2011 με λειτουργικό σύστημα OS X Mavericks (έκδοση 10.9) καθώς και σε Ubuntu 14.04 [21].

2.3.2 Χειρισμός και επικοινωνία με το Ρομπότ

Όπως προαναφέρθηκε αυτή η εργασία θεωρεί ως δεδομένο τον τρόπο υπολογισμού της πορείας και κίνησης του Ρομπότ και επομένως χρησιμοποιήθηκαν έτοιμες βιβλιοθήκες και προγράμματα από τον πάροχο του Ρομπότ.

Για την εξομοίωση της κίνησης του Ρομπότ στον υπολογιστή για την διεξαγωγή πειραμάτων χρησιμοποιήθηκε η πλατφόρμα ARIA (Advanced Robot Interface for Applications) η οποία έχει δημιουργηθεί από την ομάδα Adept MobileRobots. Ο στόχος της εταιρίας αυτής είναι να κατασκευάζει ρομποτικά συστήματα για ερευνητικό αλλά και για εμπορικό σκοπό. Η πλατφόρμα ARIA στην ουσία είναι μια βιβλιοθήκη γραμμένη στην γλώσσα προγραμματισμού C++ με την οποία μπορεί κάποιος να ελέγξει την ταχύτητα,

την περιστροφή, την πορεία, τη σχετική πορεία και άλλες μεταβλητές κίνησης ενός ρομπότ. Εκτός όμως από αυτήν την βιβλιοθήκη υπάρχει και η ArNetworking μέσω της οποίας μπορεί κάποιος να προγραμματίσει μια επικοινωνία πελάτη-εξυπηρετητή που τρέχει τοπικά στο Ρομπότ. Με τον τρόπο αυτό επιτυγχάνεται απομακρυσμένος έλεγχος και κίνηση του Ρομπότ. Από την εταιρία MobileRobots δίνονται κάποια έτοιμα προγράμματα πελάτη και εξυπηρετητή όπως το MobileEyes το οποίο είναι μια εφαρμογή πελάτη για τον χειρισμό του Ρομπότ το οποίο είναι απλό και εύκολο στην χρήση. Για αυτήν την εργασία όμως χρησιμοποιήθηκε μια αλλαγμένη έκδοση του εξυπηρετητή που τρέχει στο Ρομπότ το οποίο δημιουργήθηκε για τον σκοπό μιας άλλης πτυχιακής εργασίας [22] στο τμήμα αυτό.

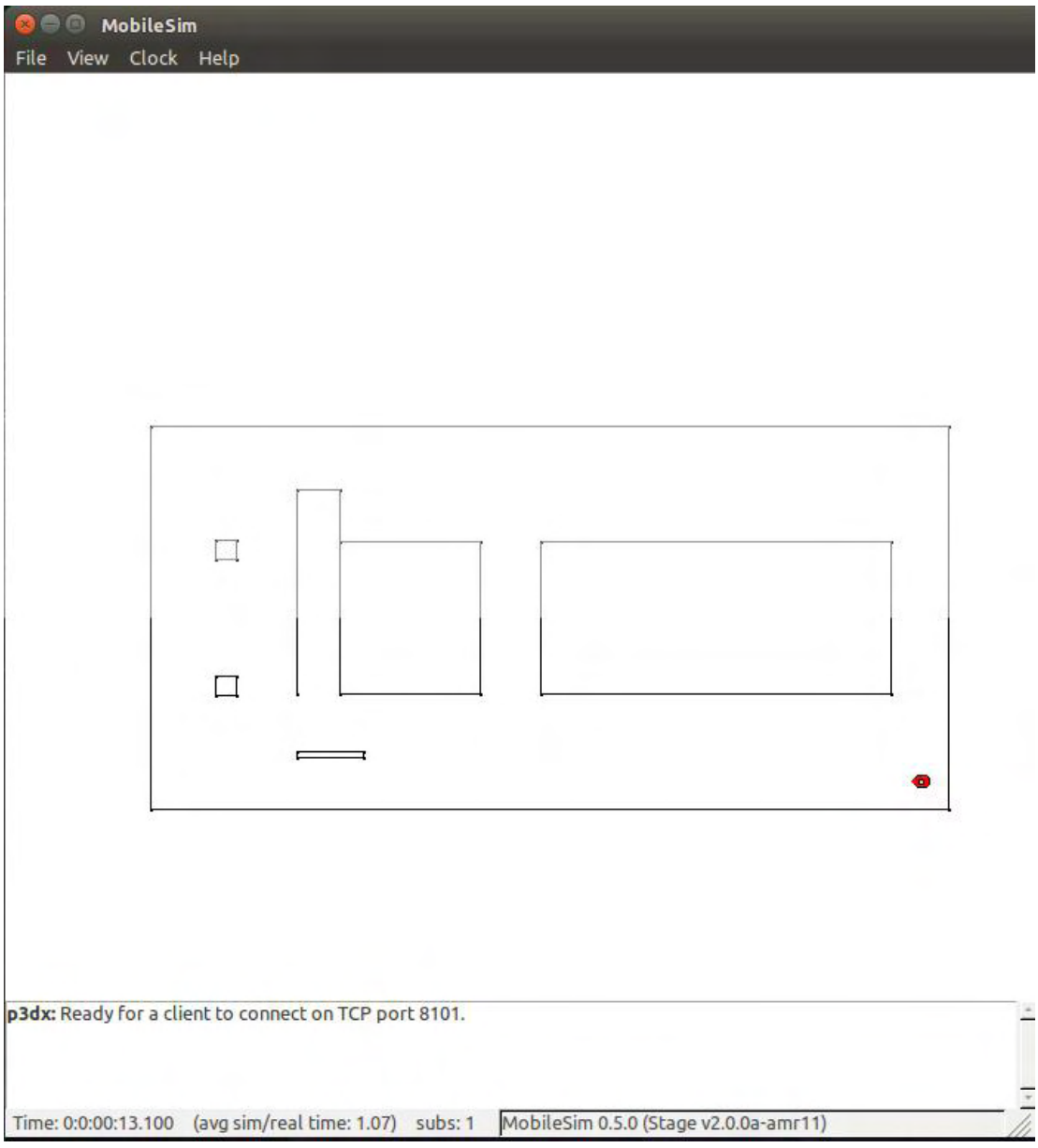


Εικόνα 6: Στιγμιότυπο λειτουργείας του προγράμματος MobileEyes

Σε αυτήν την έκδοση του εξυπηρετητή το Ρομπότ χρησιμοποιείται έλεγχος κινήσεων και πορείας, εκτίμηση θέσης του Ρομπότ μέσο οδομετρίας, γίνεται ανάγνωση πληροφοριών μέσω αισθητήριων οργάνων όπως sonar , λέιζερ, προφυλακτήρων και υπέρυθρων. Η ιδιαιτερότητα που παρουσιάζει αυτή η έκδοση είναι ότι τρέχει έναν εξυπηρετητή ο οποίος αφού αρχικοποιηθεί περιμένει να λάβει πληροφορίες από κάποιον πελάτη ώστε να κινηθεί. Οι πληροφορίες αυτές είναι συνήθως συντεταγμένες τις οποίες μόλις τις λάβει το ρομπότ, χρησιμοποιεί τις ενσωματωμένες βιβλιοθήκες καθώς και πληροφορίες που έχει συλλέξει (όπως η θέση του) για να σχεδιάσει την πορεία που θα ακολουθήσει προς αυτές τις συντεταγμένες και τέλος κινείται προς εκεί. Εάν σε κάποιο σημείο και ενώ δεν έχει φτάσει στον προορισμό λάβει ξανά από τον πελάτη άλλες συντεταγμένες, τότε ξανά-υπολογίζει την πορεία που πρέπει να ακολουθήσει και αλλάζει την κατεύθυνση του ώστε να

κινηθεί προς τις νέες συντεταγμένες. Σε αυτό το σημείο πρέπει να αναφερθεί ότι σύμφωνα με τις πληροφορίες που έχει συλλέξει από τα αισθητήρια όργανα του το ρομπότ πραγματοποιεί λειτουργείες αποφυγής εμποδίων.

Το MobileSim είναι ένα περιβάλλον προσομοίωσης για αποσφαλμάτωση και πειραματισμό για προγράμματα που χρησιμοποιούν την βιβλιοθήκη ARIA και χρησιμοποιήθηκε και αυτό για την παρακολούθηση της κίνησης κατά την προσομοίωση εκτέλεσης των αλγορίθμων στον υπολογιστή.

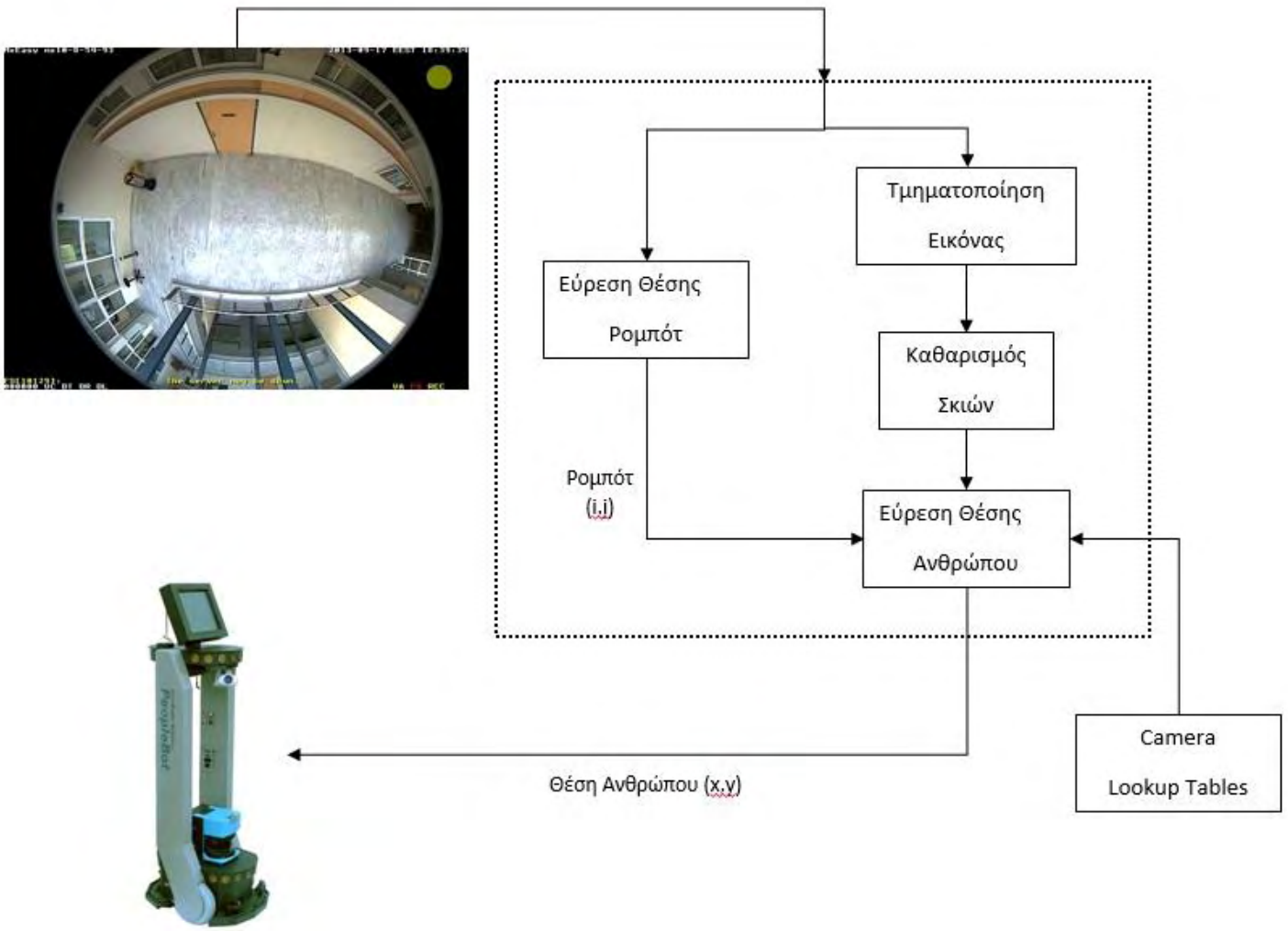


Εικόνα 7: Στιγμιότυπο λειτουργείας του προγράμματος MobileSim

Κεφάλαιο 3

Οι Αλγόριθμοι

Για τον σκοπό αυτής της πτυχιακής αναπτύχθηκαν και δοκιμάστηκαν δύο αλγόριθμοι τμηματοποίησης βίντεο, ένας αλγόριθμος αναγνώρισης της θέσης του ρομπότ και ένας αλγόριθμος υπολογισμού της θέσης του ανθρώπου οι οποίοι περιγράφονται στην συνέχεια. Και οι δύο αλγόριθμοι τμηματοποίησης λαμβάνουν σαν είσοδο βίντεο ή περιοδικές εικόνες από την fisheye camera καθώς και τους lookup tables που έχουν δημιουργηθεί σύμφωνα με το μοντέλο της κάμερας, στην συνέχεια τμηματοποιούνται οι εικόνες έτσι ώστε τα λευκά pixel να εμφανίζουν τις περιοχές όπου υπάρχει κίνηση και τα μαύρα pixel τις περιοχές του υποβάθρου. Αφού αναγνωριστεί ο άνθρωπος στην συνέχεια σύμφωνα με τα pixel (i,j) όπου βρίσκεται , υπολογίζεται ο αντίστροφος μετασχηματισμός σύμφωνα με το μοντέλο της fisheye κάμερας για να υπολογιστούν οι πραγματικές συντεταγμένες (x,y) του ανθρώπου. Τέλος οι συντεταγμένες αυτές αποστέλλονται στο ρομπότ και το ρομπότ κινείται προς την κατεύθυνση αυτή. Η διαδικασία αυτή επαναλαμβάνεται για κάθε frame και ανά τακτά χρονικά διαστήματα στέλνονται οι συντεταγμένες του ανθρώπου που έχουν αναγνωριστεί στο ρομπότ. Παρακάτω φαίνεται το μπλοκ διάγραμμα της λειτουργίας.



Εικόνα 8: Μπλοκ διάγραμμα του αλγορίθμου που αναπτύχθηκε

3.1 Αλγόριθμος εύρεσης Ρομπότ

Το πρώτο πρόβλημα εμφανίζεται αμέσως μετά την τμηματοποίηση των εικόνων και είναι ότι στην τμηματοποιημένη εικόνα θα εμφανιστεί και το Ρομπότ καθώς ο αλγόριθμος δεν έχει κάποιον τρόπο για να αναγνωρίσει εάν η κίνηση στην εικόνα προήλθε από άνθρωπο ή από το Ρομπότ. Έτσι αναπτύχθηκε ο συγκεκριμένος αλγόριθμος προκειμένου να αναγνωριστεί η θέση του Ρομπότ και να αφαιρεθεί από τον αλγόριθμο υπολογισμού θέσης. Για να λειτουργήσει ο αλγόριθμος ζητάει κατά το πρώτο frame από τον χρήστη να εισάγει τις συντεταγμένες (pixel) του Ρομπότ. Επίσης για να ξεχωρίσουμε την σιλουέτα του Ρομπότ από τις υπόλοιπες έχουμε τοποθετήσει ένα συγκεκριμένο χρώμα επάνω στο Ρομπότ και σύμφωνα με το χρώμα η εικόνα τμηματοποιείται σύμφωνα με τον εξής κανόνα.

$$R(i,j) > 1.8 \cdot G(i,j) > 1.2 \cdot B(i,j) \tag{9}$$

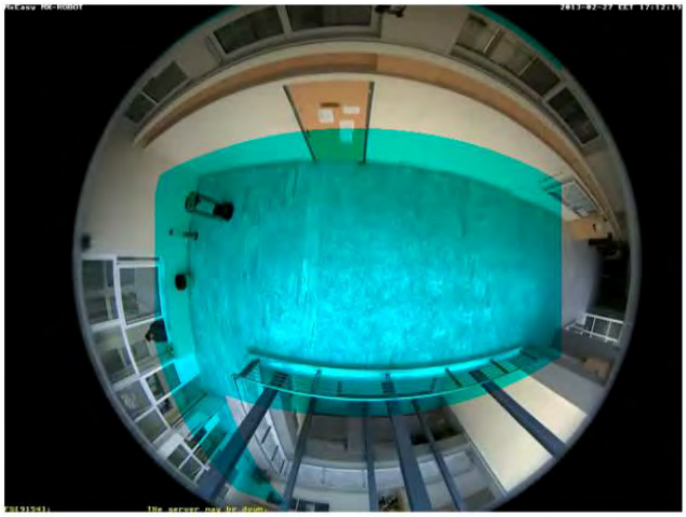
Με αυτόν τον τρόπο καταφέρνουμε να πάρουμε με μεγάλη ταχύτητα και ακρίβεια την τμηματοποιημένη εικόνα επιτρέποντας μας να τρέξουμε αυτόν τον αλγόριθμο σε πραγματικό χρόνο.

Εικόνα 9:(α) Πραγματική εικόνα του Ρομπότ, (β) αποτέλεσμα του αλγορίθμου αναγνώρισης του Ρομπότ

Αφού πάρουμε την τμηματοποιημένη κατά το πρώτο frame υπολογίζουμε την απόσταση (pixel) του βαρύκεντρου κάθε ομάδας pixel που θα εμφανιστεί από τις συντεταγμένες που έχει δώσει ο χρήστης και για όποια ομάδα είναι πιο κοντά εξάγουμε τις συντεταγμένες (i_robot,j_robot) των pixel. Αυτές τις συντεταγμένες όπως περιγράφουμε παρακάτω θα χρησιμοποιηθούν για να αφαιρέσουμε την σιλουέτα του Ρομπότ από τον αλγόριθμο υπολογισμού θέσης. Τέλος αυτές οι συντεταγμένες αποθηκεύονται για να χρησιμοποιηθούν στην επόμενη επανάληψη.

Γνωρίζοντας μάλιστα τα χαρακτηριστικά του Ρομπότ αφού έχει σταθερό ύψος, πλάτος και μήκος μπορούμε να εξαιρέσουμε και άλλες περιοχές της εικόνας, περιοχές που δεν μπορεί να βρεθεί το Ρομπότ. Έτσι χρησιμοποιούμε και με δυαδική μάσκα όπου τα επιτρεπόμενα pixel έχουν την

τιμή 1 μόνο στις περιοχές όπου μπορεί να πάει το ρομπότ όπως φαίνεται στο σκιασμένο μέρος στην εικόνα.



Εικόνα 10: Ένα τυπικό frame, η σκιασμένη περιοχή είναι τα pixel όπου υπάρχει πιθανότητα να εμφανιστεί το Ρομπότ

3.2 Τμηματοποίηση Σιλουετών

Κατά το γενικό μοντέλο της τμηματοποίησης η εικόνα περνάει από μια επεξεργασία και το τελικό αποτέλεσμα είναι μια binary εικόνα όπου με άσπρο χρώμα θα είναι τα μέρη της εικόνας όπου παρουσιάζουν μια κίνηση και με μαύρο θα είναι το υπόβαθρο. Στην αρχή λοιπόν παίρνουμε το τρέχων frame από την κάμερα, το περνάμε από τον αλγόριθμο τμηματοποίησης, στην συνέχεια αφαιρούμε τις μικρές ομάδες λευκών pixel (κάτω από 20) επειδή είναι θόρυβος , καθαρίζουμε όσο το δυνατόν καλύτερα γίνεται την τμηματοποιημένη εικόνα έτσι ώστε να ξεχωρίσουν οι σιλουέτες και στην συνέχεια το αποτέλεσμα της τμηματοποίησης εισάγεται σαν είσοδο στους επόμενους κατά σειρά αλγορίθμους.

Αυτή η διαδικασία μπορεί να γίνει είτε με το κανονικό έγχρωμο (RGB) frame από την κάμερα είτε με grayscale frame (αποχρώσεις του γκρι). Εάν επιλέξουμε να κάνουμε τμηματοποίηση σε grayscale frame το πλεονέκτημα είναι ότι είναι αρκετά γρήγορη η εκτέλεση του αλγορίθμου αλλά τα μειονεκτήματα είναι ότι η τμηματοποίηση δεν είναι πολύ καλή και το ότι δεν μπορούμε να εκτελέσουμε και τον αλγόριθμο καθαρισμού των σκιών. Εάν τώρα επιλέξουμε να κάνουμε τμηματοποίηση με το έγχρωμο frame τότε ο αλγόριθμος της τμηματοποίησης θα πρέπει να τρέξει 3 φορές (μία για κάθε χρώμα RGB) και στην συνέχεια με κάποιον τρόπο να συνδυάσουμε τις 3 τμηματοποιημένες εικόνες που θα έχουμε σαν αποτέλεσμα σε μία για να πάρουμε την τελική τμηματοποίηση. Όπως είναι φυσικό αυτή η εκτέλεση είναι χρονοβόρα αλλά μας δίνει αρκετά καλύτερα αποτελέσματα και μας επιτρέπει να εκτελέσουμε τον αλγόριθμο καθαρισμού σκιών για ακόμα μεγαλύτερο καθαρισμό της εικόνας. Για τον σκοπό αυτής της πτυχιακής εργασίας χρησιμοποιήθηκε ο αλγόριθμος τμηματοποίησης illumination-sensitive–IS

όπως περιγράφεται [23] και τροποποιειται [24] κάνοντας τμηματοποίηση στην έγχρωμη RGB εικόνα.

3.4 Καθαρισμός Σκιών

Όπως αναφέραμε και παραπάνω εάν ο αλγόριθμος τρέχει στον RGB χώρο είναι δυνατόν να χρησιμοποιήσουμε και τον αλγόριθμο καθαρισμού σκιάς ώστε να έχουμε πιο ακριβή αναγνώριση του ανθρώπου. Ο αλγόριθμος καθαρισμού σκιάς παίρνει σαν είσοδο το τρέχων frame της εικόνας καθώς και το μοντέλο του υποβάθρου που δημιουργείται μετά από κάθε επανάληψη του αλγορίθμου Illumination Sensitive.

Ο αλγόριθμος αυτός [26] αν και αρκετά απλός χρειάζεται υπολογιστικούς πόρους καθώς πραγματοποιεί αρκετές μαθηματικές πράξεις. Καταρχήν μετατρέπεται οι εικόνα του τρέχοντος frame και του υποβάθρου σε HSV[25] εικόνα και για κάθε pixel υπολογίζεται

- Το ratio της φωτεινότητας (Value) του τρέχοντος frame ως προς το μοντέλο του παρασκηνίου
- Η απόλυτη τιμή της διαφοράς του κορεσμού (Saturation) του τρέχοντος frame σε σχέση με το μοντέλο του παρασκηνίου
- Η απόλυτή τιμή τις διαφοράς της απόχρωσης (Hue) του τρέχοντος frame σε σχέση με το μοντέλο του παρασκηνίου.

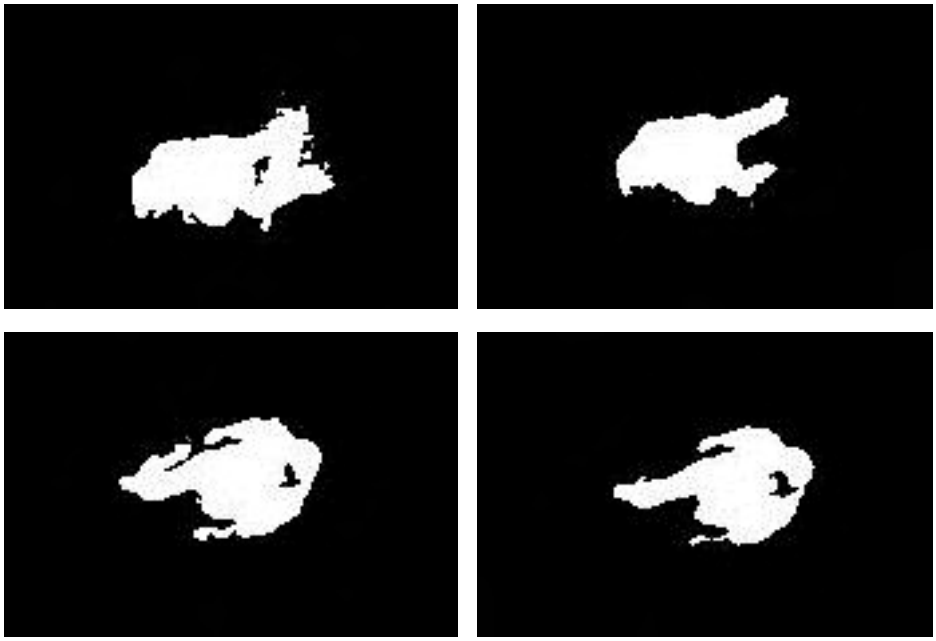
Και αυτό διότι όταν υπάρχει σκιά μέσα στην εικόνα μας οι τιμές της απόχρωσης (Hue) της εικόνας αλλάζουν αλλά πάντα μέσα σε κάποια όρια. Επίσης και οι τιμές του κορεσμού (Saturation) έχει αποδειχθεί ότι αλλάζουν μέσα σε συγκεκριμένα όρια. Η διαφορά του κορεσμού θα πρέπει να είναι σε απόλυτη τιμή ενώ η διαφορά της απόχρωσης θα πρέπει να υπολογιστεί σαν γωνιακή διαφορά όπως φαίνεται στον παρακάτω τύπο. Έτσι λοιπόν ορίζουμε για κάθε pixel εάν πρόκειται για σκιά εάν ισχύει το παρακάτω

$$\alpha \leq \frac{frame(i,j).V}{BGmodel(i,j).V} \leq \beta \wedge |frame(i,j).S - BGmodel(i,j).S| \leq Ts \wedge \mathrm{Dh} \leq \mathrm{Th} \qquad (10)$$

Όπου $\alpha \in [0,1]$, $\beta \in [0,1]$ και

$$Dh = \min(|frame(i,j).H - BGmodel(i,j).H| \, , \, 360 - |frame(i,j).H - BGmodel(i,j).H|)$$

Όπου H,S,V είναι οι τιμές της απόχρωσης (Hue) , του κορεσμού (Saturation) και της φωτεινότητας (Value) αντίστοιχα. [25]



Εικόνα 11:(αριστερά) Τμηματοποιημένη εικόνα χωρίς αφαίρεση σκιάς, (δεξιά) με αφαίρεση σκιάς

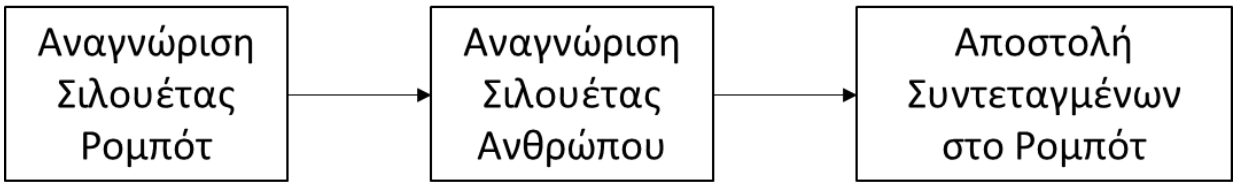
3.5 Αναγνώριση Σιλουετών

Σε αυτό το κομμάτι του αλγορίθμου λαμβάνεται σαν είσοδος

- Η τμηματοποιημένη εικόνα που προέρχεται από τον αλγόριθμο τμηματοποίησης
- Οι συντεταγμένες του Ρομπότ που προέρχονται από τον αλγόριθμο εύρεσης του Ρομπότ (παράγραφος 3.1)
- Οι συντεταγμένες (pixel) του ανθρώπου (των ανθρώπων) που εισάγονται από τον χρήστη κατά το πρώτο frame

Επίσης από εδώ και πέρα σαν ομάδα pixel αναφέρεται μια ομάδα από λευκά pixel τα οποία έχουν προέλθει από τον αλγόριθμο τμηματοποίησης. Αυτή η ομάδα μπορεί να ανήκει στον άνθρωπο, στο ρομπότ ή και σε κομμάτι του ανθρώπου. Σαν σιλουέτα (ανθρώπου ή ρομπότ) αναφέρεται ένα σύνολο από ομάδες pixel που έχει όμως αναγνωριστεί ότι ανήκει στον άνθρωπο ή στο ρομπότ. Μια σιλουέτα μπορεί να αποτελείται από πολλές ομάδες pixel. Τέλος οι αρχικές συντεταγμένες του ανθρώπου και του Ρομπότ εισάγονται από τον χρήστη, μόνο μια φορά κατά πρώτο frame και στην συνέχεια υπολογίζονται από τον αλγόριθμο 3.1 για το ρομπότ και 3.6 για τον άνθρωπο.

Η τμηματοποιημένη εικόνα που λαμβάνεται σε αυτό το κομμάτι του αλγορίθμου περιέχει με λευκά pixel τον άνθρωπο και το Ρομπότ (εφόσον έχουν κινηθεί). Έτσι θα πρέπει να αναγνωρίσουμε ποιες ομάδες pixel ανήκουν στον άνθρωπο και ποιες στο Ρομπότ. Η γενική λειτουργεία του αλγορίθμου φαίνεται στο παρακάτω σχήμα.



Εικόνα 12: Γενικό μοντέλο λειτουργείας του αλγορίθμου

3.5.1 Αναγνώριση Σιλουέτας Ρομπότ

Σε αυτό το σημείο πρέπει να αναφέρουμε ότι ο αλγόριθμος εύρεσης του Ρομπότ (παράγραφος 3.1) θα εξάγει τις συντεταγμένες (pixel) του Ρομπότ ανεξάρτητα με το αν αυτό κουνηθεί η όχι. Επομένως πρώτα ελέγχουμε εάν κάποια από τις ομάδες pixel της εικόνας ανήκει στο Ρομπότ έτσι ώστε να την αφαιρέσουμε.

Η αναγνώριση αυτή γίνεται με τον εξής τρόπο. Πρώτα υπολογίζουμε την απόσταση (σε pixel) του βαρύκεντρου κάθε ομάδας pixel από τις συντεταγμένες του ρομπότ που έχουμε λάβει σαν είσοδο. Στην συνέχεια εάν το βαρύκεντρο της πιο κοντινής ομάδα pixel βρίσκεται σε απόσταση μικρότερη από 20 pixel από τις συντεταγμένες του ρομπότ, όπως έχουν υπολογιστεί από τον αλγόριθμο 3.1, τότε αυτή η ομάδα pixel αναγνωρίζεται ως η σιλουέτα του Ρομπότ και αφαιρείται.

3.5.2 Αναγνώριση Σιλουέτας Ενός Ανθρώπου

Η διαδικασία αναγνώρισης της σιλουέτας του ανθρώπου είναι πιο πολύπλοκη από την αναγνώριση της σιλουέτας του Ρομπότ. Αυτό συμβαίνει γιατί ο άνθρωπος πολλές φορές μπορεί να εμφανιστεί στην τμηματοποιημένη εικόνα σαν “σπασμένος” σε 2 ή και περισσότερες ομάδες pixel όπως φαίνεται στην εικόνα.

Εικόνα 13: Αποτέλεσμα τμηματοποίησης μιας ανθρώπινης σιλουέτας

Όπως αναφέραμε και παραπάνω στο πρώτο frame της εκτέλεσης του αλγορίθμου ζητείται από τον χρήστη να δώσει τις συντεταγμένες του ανθρώπου που πρόκειται να παρακολουθηθεί. Με αυτές τις συντεταγμένες λοιπόν και με την τμηματοποιημένη εικόνα καλούμαστε να αναγνωρίσουμε ποια σιλουέτα ανήκει σε κάθε άνθρωπο.

Κατά την πρώτη επανάληψη υπολογίζεται η απόσταση του βαρύκεντρου κάθε ομάδας pixel από τις συντεταγμένες του ανθρώπου που έχει δώσει ο χρήστης. Η πιο κοντινή ομάδα pixel αναγνωρίζεται σαν μια ομάδα pixel που ανήκουν στον άνθρωπο. Για να σχηματίσουμε τώρα ολόκληρη την σιλουέτα του ανθρώπου υπολογίζουμε την απόσταση αυτής της ομάδας pixel από όλες τις υπόλοιπες. Όσες ομάδες pixel βρίσκονται σε απόσταση μικρότερη από 20 pixel τότε συνενώνονται για να σχηματίσουν την ολοκληρωμένη πλέων σιλουέτα του ανθρώπου. Η τελική σιλουέτα του ανθρώπου μπορεί να αποτελείται από 1 ή και παραπάνω ομάδες pixel.

Τέλος για την σιλουέτα του ανθρώπου αποθηκεύουμε κάποιες τιμές για να μπορέσουμε να κάνουμε την αναγνώριση στην επόμενη επανάληψη. Οι τιμές που αποθηκεύονται είναι:

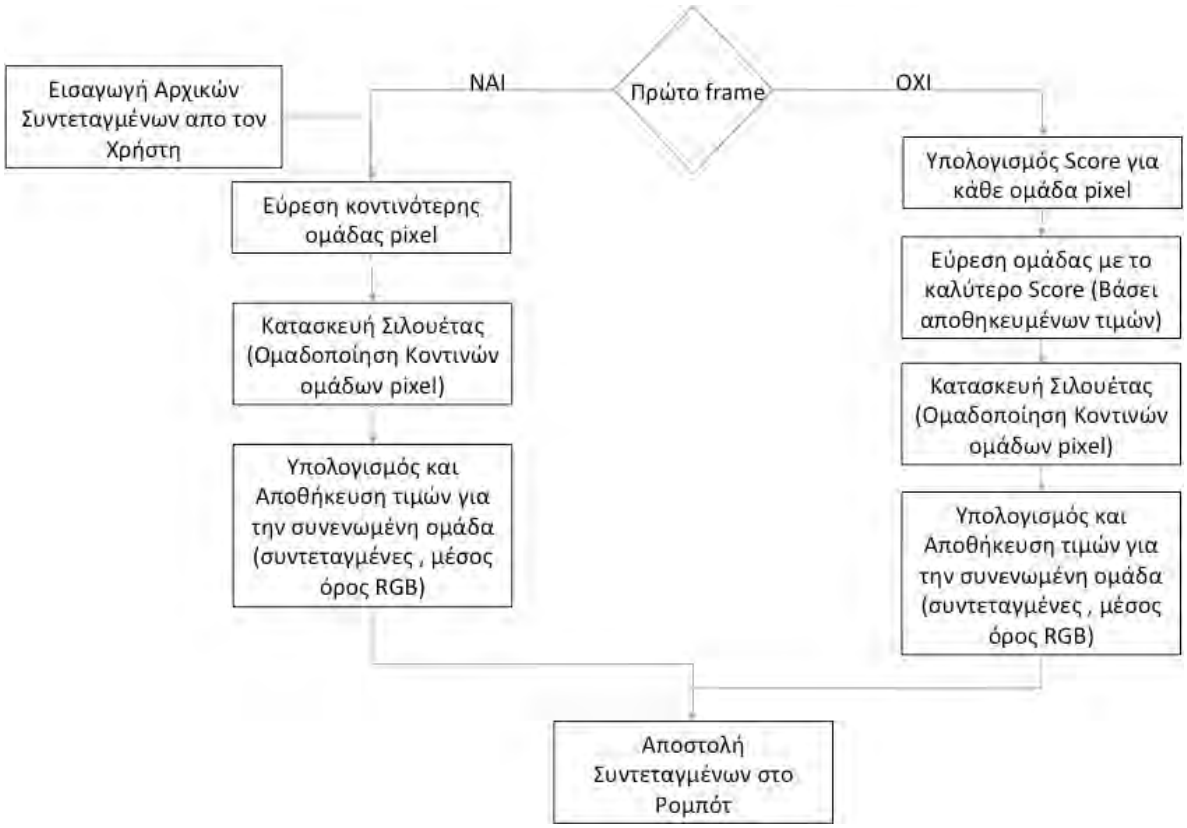
- Μέσος όρος των τιμών RGB
- Συντεταγμένες (pixel) του βαρύκεντρου

Κατά τις επόμενες επαναλήψεις τώρα επαναλαμβάνεται η διαδικασία χρησιμοποιώντας κάποια επιπλέων βήματα για την αναγνώριση της σιλουέτας. Μόλις λάβουμε την νέα τμηματοποιημένη εικόνα υπολογίζουμε για κάθε ομάδα pixel το βαρύκεντρο και τον μέσο όρο των τιμών RGB. Στην συνέχεια υπολογίζουμε την διαφορά των τιμών που υπολογίσαμε για κάθε ομάδα pixel από τις τιμές που είχαμε αποθηκεύσει από την προηγούμενη επανάληψη. Έτσι σύμφωνα με τον παρακάτω τύπο, υπολογίζουμε ένα score για κάθε ομάδα pixel.

$$score = 0.5 * distance + 0.3 * rgb\_mean\_difference - 0.2 * size \quad (11)$$

Όπου $distance$ είναι η απόσταση κάθε ομάδας pixel από τις αποθηκευμένες συντεταγμένες (αποστάσεις pixel) , $rgb\_mean\_difference$ είναι η διαφορά του μέσου όρου των τιμών rgb της ομάδας pixel από την αποθηκευμένη τιμή και τέλος το $size$ είναι ο αριθμός των pixel που αποτελούν αυτήν την ομάδα. Από τον τύπο φαίνεται ότι όσο πιο μικρή είναι αυτή η τιμή τόσο μεγαλύτερη είναι η πιθανότητα αυτή η ομάδα pixel να ανήκει στον άνθρωπο που θέλουμε να παρακολουθήσουμε. Οπότε αφού υπολογιστεί αυτό το score για όλες τις ομάδες pixel της αναγνωρισμένης εικόνας επιλέγεται η ομάδα με το μικρότερο score. Στην συνέχεια αφού επιλεγεί αυτή η ομάδα pixel υπολογίζεται πάλι η απόσταση του βαρύκεντρου αυτής της ομάδας από όλες τις υπόλοιπες ομάδες (όπως περιγράφηκε παραπάνω) και συνενώνονται όσες ομάδες pixel βρίσκονται σε απόσταση μικρότερη από 20 pixel για να σχηματίσουν την σιλουέτα του ανθρώπου. Τέλος για την σιλουέτα αυτή υπολογίζεται το νέο βαρύκεντρο και η μέσος όρος των τιμών RGB και αποθηκεύονται για να χρησιμοποιηθούν για την επόμενη επανάληψη.

.



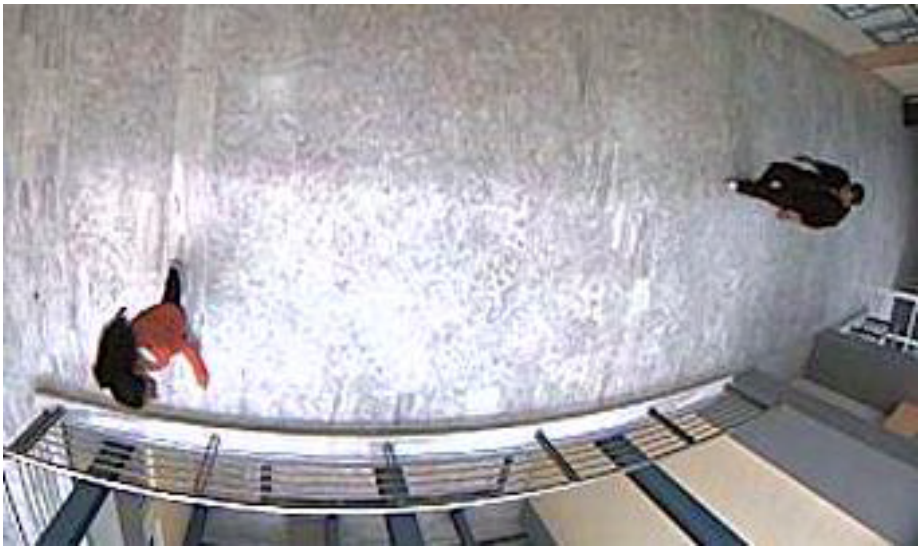
Εικόνα 14: Λογικό διάγραμμα του αλγορίθμου

```
human_x , human_y //είσοδος απο τον χρήστη

ομαδες_pixel //οι ομάδες με τα λευκά pixel

ΑΝ( εινααιΠρώτοFrame)
    //Εύρεση κοντινότερης ομάδας pixel
    min_position = 1;
    min_distance =
      ΑΠΟΣΤΑΣΗ(human_x,human_y,ομαδες_pixel(1)_x,ομαδες_pixel(1)_y)

    ΓΙΑ i ΑΠΟ 2 ΕΩΣ ΜΕΓΕΘΟΣ(ομαδες_pixel)
      distance =
        ΑΠΟΣΤΑΣΗ(human_x,human_y,ομαδες_pixel(i)_x,ομαδες_pixel(i)_y)
      ΑΝ distance < min_distance
            min_distance = distance
            min_position = i
      ΤΕΛΟΣ
    //Ομαδοποίηση Κοντινών ομάδων pixel
    ΓΙΑ i ΑΠΟ 1 ΕΩΣ ΜΕΓΕΘΟΣ(ομαδες_pixel)
      distance =
       ΑΠΟΣΤΑΣΗ(ομαδες_pixel(min_position)_x,ομαδες_pixel(min_position)_y,
                ομαδες_pixel(i)_x,ομαδες_pixel(i)_y)
      ΑΝ distance < 20
            σιλουετα_ανθρώπου = σιλουετα_ανθρώπου + ομαδες_pixel(i)
      ΤΕΛΟΣ
    ΤΕΛΟΣ

    //Υπολογισμός και αποθήκευση τιμών για την συνενωμένη ομάδα
    rgb_mean = ΜΕΣΟΣ_ΟΡΟΣ_RGB(σιλουετα_ανθρώπου)//Μέσος Όρος RGB
    [human_x,human_y] = ΒΑΡΥΚΕΝΤΡΟ(σιλουετα_ανθρώπου) //Συντεταγμένες
```

```
ΑΛΛΙΩΣ
    //Υπολογισμός score για κάθε ομάδα pixel
    ΓΙΑ i ΑΠΟ 1 ΕΩΣ ΜΕΓΕΘΟΣ(ομαδες_pixel)
     rgb_diff =
       ΑΠΟΛΥΤΗ_ΤΙΜΗ(ΜΕΣΟΣ_ΟΡΟΣ_RGB(ομαδες_pixel(i)) - rgb_mean)
     distance =
       ΑΠΟΣΤΑΣΗ(human_x,human_y,ομαδες_pixel(i)_x,ομαδες_pixel(i)_y)
     score(i) =
       0.5 * distance + 0.3 * rgb_diff - 0.2 * ομαδες_pixel(i)_size
    ΤΕΛΟΣ

    //Εύρεση ομάδας με το καλύτερο score
    best_silouete_position = ΕΛΑΧΙΣΤΟ(score)
    [best_silouete_x,best_silouete_y] =
      ΒΑΡΥΚΕΝΤΡΟ(ομαδες_pixel(best_silouete_position));

    //Ομαδοποίηση κοντινών ομάδων pixel
    ΓΙΑ i ΑΠΟ 1 ΕΩΣ ΜΕΓΕΘΟΣ(ομαδες_pixel)
      diff =
        ΑΠΟΣΤΑΣΗ(best_silouete_x,best_silouete_y,
                 ομαδες_pixel(i)_x,ομαδες_pixel(i)_y)
        ΑΝ(diff < 20)
            σιλουετα_ανθρώπου = σιλουετα_ανθρώπου + ομαδες_pixel(i)
        ΤΕΛΟΣ
    ΤΕΛΟΣ

    //Υπολογισμός και αποθήκευση τιμών για την συνενωμένη ομάδα
    rgb_mean = ΜΕΣΟΣ_ΟΡΟΣ_RGB(σιλουετα_ανθρώπου)//Μέσος Όρος RGB
    [human_x,human_y] = ΒΑΡΥΚΕΝΤΡΟ(σιλουετα_ανθρώπου)//Συντεταγμένες
ΤΕΛΟΣ
```

3.5.2 Αναγνώριση Σιλουέτας Περισσότερων Ανθρώπων

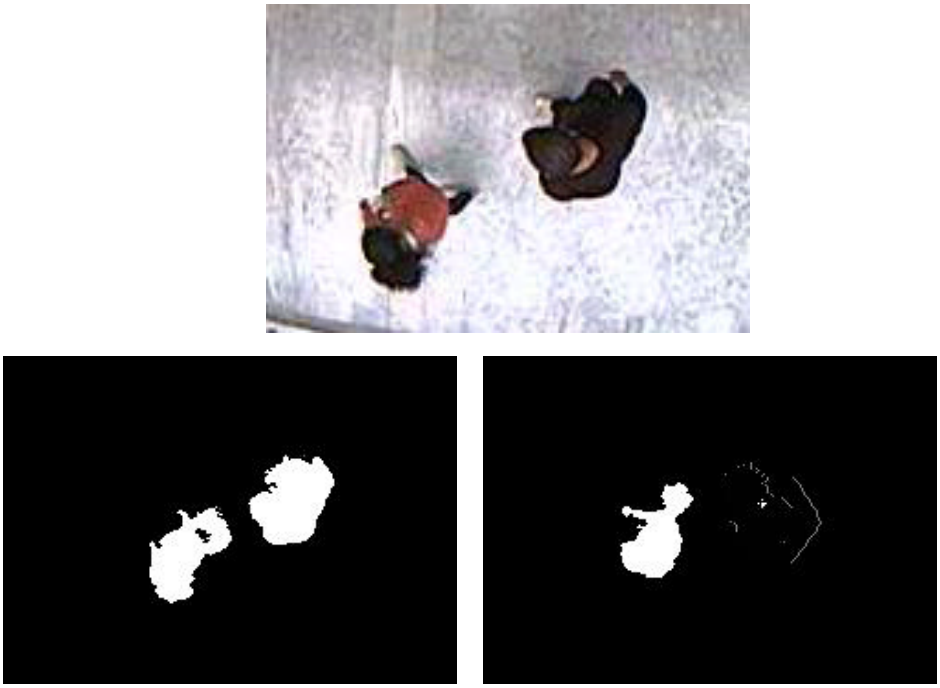
Στον πραγματικό κόσμο βέβαια θα τύχει να υπάρξουν και παραπάνω από ένας άνθρωποι στο δωμάτιο όπως φαίνεται στην εικόνα.



Εικόνα 15: Εικόνα και αποτέλεσμα τμηματοποίησης 2 ανθρώπων

Εάν οι άνθρωποι κατά την κίνηση τους δεν υπερκαλύψουν ο ένας τον άλλον τότε και μόνο ο υπολογισμός της απόστασης των βαρύκεντρων από τις προηγούμενες συντεταγμένες θα ήταν αρκετός για να παρακολουθηθεί κάποιος από τους 2 ανθρώπους. Στον πραγματικό κόσμο όμως αυτό είναι αδύνατον και γι' αυτόν τον λόγο ζητάμε από τον χρήστη κατά το πρώτο frame να δώσει και τις συντεταγμένες του δεύτερου ανθρώπου, αυτού που δεν θα παρακολουθηθεί.

Έτσι ο αλγόριθμος της αναγνώρισης της σιλουέτας θα τρέξει 2 φορές. Την 1η φορά θα υπολογίσει το score για τις συντεταγμένες του 1ου ανθρώπου (χρησιμοποιώντας τις συντεταγμένες που έχουν υπολογιστεί από τον αλγόριθμο 3.1 για την υπό-παρακολούθηση σιλουέτα) για να αναγνωριστεί ο 1ος άνθρωπος. Ο αλγόριθμος θα επαναληφθεί την 2η φορά για τις συντεταγμένες του 2ου ανθρώπου για να αναγνωριστεί ο 2ος ανθρώπου (αυτός που δεν θέλουμε να παρακολουθηθεί), χωρίς να συμμετέχει στον υπολογισμό του ελάχιστου score η σιλουέτα που αναγνωρίστηκε από το προηγούμενο βήμα. Έτσι λοιπόν επιπρόσθετα σε κάθε επανάληψη θα αποθηκεύονται τιμές (συντεταγμένες, μέσος όρος τιμών RGB) και για τον 2ο άνθρωπο. Με αυτόν τον τρόπο καταφέρνουμε να αναγνωρίζουμε και να παρακολουθούμε μόνο τον 1 από τους 2 ανθρώπους χωρίς να μπερδεύεται ο αλγόριθμος.



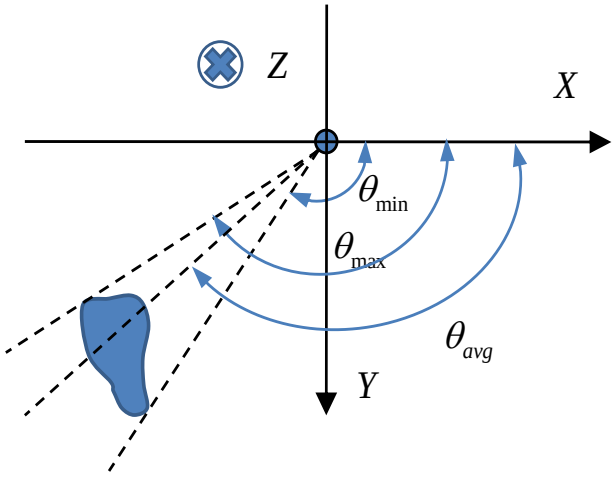
Εικόνα 16: (α) Frame 2 ανθρώπων , (β) Αποτέλεσμα τμηματοποίησης 2 ανθρώπων, (γ) αποτέλεσμα αναγνώρισης του ανθρώπου που πρέπει να παρακολουθηθεί (αριστερά) και απόρριψη του 2ου ανθρώπου (δεξιά)

Επίσης κατά την επικάλυψη των δύο ανθρώπων το αποτέλεσμα της τμηματοποίησης είναι μια μεγάλη ομάδα pixel, το πλήθος των οποίων θα υπερβαίνει ένα προκαθορισμένο κατώφλι στο οποίο έχει δοθεί η τιμή 180 (βάσει πειραματισμού). Σε εκείνη την επανάληψη ο αλγόριθμος δεν θα κάνει κανέναν υπολογισμό και θα προχωρήσει στα επόμενα frame μέχρι οι σιλουέτες να διαχωριστούν ώστε να υπολογίσει τα score και να αποφασίσει ποιόν από τους δύο θα πρέπει να ακολουθήσει. Τέλος αφού απορριφθεί ο ένας άνθρωπος υπολογίζεται η πραγματική θέση του άλλου όπως περιγράφεται στην παράγραφο 3.6 και οι συντεταγμένες αυτές αποστέλλονται στο Ρομπότ.

3.6 Υπολογισμός και Αποστολή Συντεταγμένων

Σε αυτό το σημείο για τον υπολογισμό των συντεταγμένων χρησιμοποιείται η ομαδοποιημένη σιλουέτα του ανθρώπου και τα lookup tables του μοντέλου της κάμερας για να υπολογιστούν οι τιμές θ_{min}, θ_{max}, και φ_{min}, φ_{max}, φ_{avg} όπως περιγράφονται παρακάτω

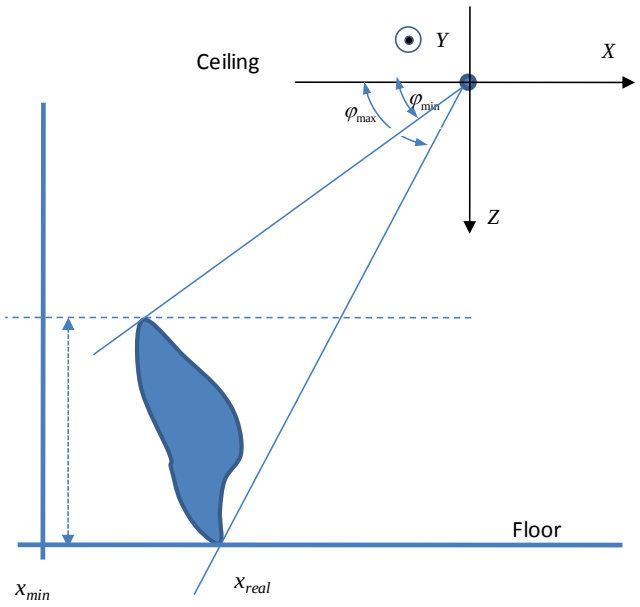
Έτσι μπορεί εύκολα να υπολογιστεί η ελάχιστη, η μέγιστη και η μέση τιμή του Αζιμούθιου καθώς και η ελάχιστη και η μέγιστη τιμή του ύψους σύμφωνα με τα lookup tables όπως φαίνεται και στις παρακάτω εικόνες



View from above

.

Εικόνα 17: Σε αυτήν την εικόνα είναι η όψη βλέποντας από το ταβάνι προς τα κάτω. Φαίνεται πως υπολογίζεται η ελάχιστη και μέγιστη τιμή του Αζιμούθιου



Εικόνα 18: Στην κάτω εικόνα είναι η πλαϊνή όψη και φαίνεται πως υπολογίζεται η ελάχιστη και η μέγιστη τιμή του ύψους.

Υποθέτοντας ότι το τμηματοποιημένο αντικείμενο είναι ένας άνθρωπος που πατάει στο έδαφος, η θέση του στο πάτωμα (x_{real}, y_{real}) υπολογίζονται όπως παρακάτω.

$$x_{real} = \frac{z_{\max}}{\sin\varphi_{\max}} \cos\varphi_{\max} \cos\theta_{avg},$$
$$y_{real} = \frac{z_{\max}}{\sin\varphi_{\max}} \cos\varphi_{\max} \sin\theta_{avg}. \quad (12)$$

Όπου φ_{max} και θ_{avg} είναι το μέγιστο ύψος και ο μέσος όρος του Αζιμούθιου των pixel της τμηματοποιημένης εικόνας που αντιστοιχούν στον άνθρωπο (Εικόνα 5, Εικόνα 6)

Αν υποθέσουμε ότι ο άνθρωπος στέκεται όρθιος και δεν είναι ακριβώς κάτω από την κάμερα ($\varphi_{min} < \frac{\pi}{2}$) τότε το ύψος και το πλάτος του ανθρώπου / αντικειμένου μπορούν να υπολογιστούν

$$height = z_{\max} - \tan(\varphi_{\min})\sqrt{x_{real}^2 + y_{real}^2},$$
$$width = 2\sqrt{x_{real}^2 + y_{real}^2}\tan\left(\frac{|\theta_{\max} - \theta_{\min}|}{2}\right). \quad (13)$$

Όπου Z_{max} είναι η Z συντεταγμένη του πατώματος σύμφωνα με το σύστημα συντεταγμένων, η τομή της ευθείας που ξεκινάει από την κάμερα και ορίζεται από το (Θ_{avg}, φ_{max}) με το πάτωμα ($Z=Z_{max}$).

Για την αποστολή των συντεταγμένων καθ' όλη την διάρκεια της εκτέλεσης του προγράμματος, ο αλγόριθμος είναι συνδεδεμένος με το ρομπότ μέσω ενός HTTP Connection. Παρόλο που ο υπολογισμός των συντεταγμένων του ανθρώπου γίνεται πολλές φορές μέσα σε ένα δευτερόλεπτο, επιλέξαμε να στέλνουμε τις συντεταγμένες στο Ρομπότ κάθε 5 δευτερόλεπτα, δίνοντας του χρόνο να κινηθεί προς τον άνθρωπο πριν του αλλάξουμε την πορεία. Ο χρόνος αυτός επιλέχθηκε διότι το συγκεκριμένο Ρομπότ κάθε φορά που του στέλνουμε κάποιες συντεταγμένες με την εντολή να κινηθεί προς τα εκεί, σταματάει, υπολογίζει την πορεία που πρέπει να ακολουθήσει και στην συνέχεια κινείται προς τα εκεί. Εάν εμείς του στέλναμε τις συντεταγμένες πολύ συχνά τότε ο χρόνος που θα ήταν σταματημένος θα ήταν αρκετά μεγάλος.

Κεφάλαιο 4

Αποτελέσματα

4.1 Αποτελέσματα Προσομοιώσεων

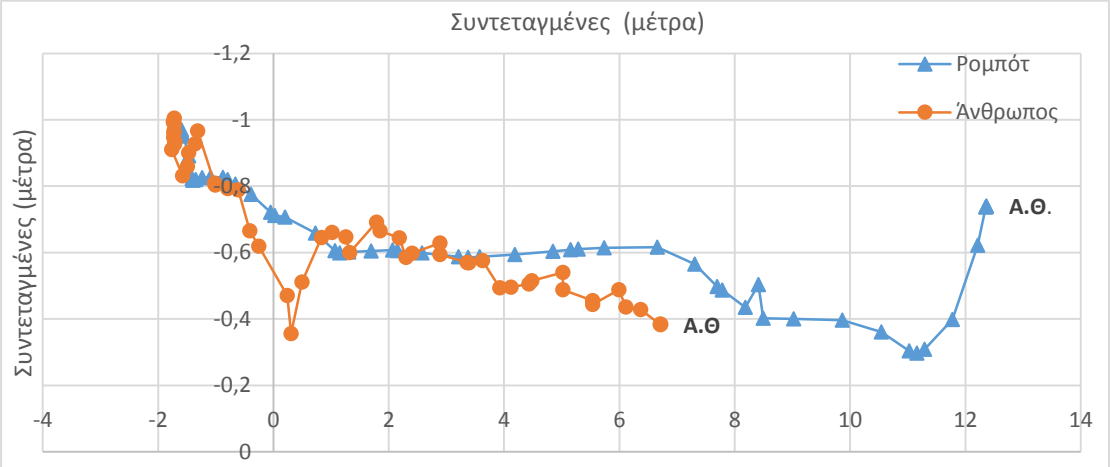
Η πειραματική διάταξη που χρησιμοποιήθηκε στην περίπτωση της προσομοιωμένης παρακολούθησης ανθρώπου περιγράφεται ως εξής. Αντί για video πραγματικού χρόνου (live) χρησιμοποιήθηκε offline βίντεο με κίνηση ανθρώπων που συλλέχθηκαν χωρίς την παρουσία του ρομπότ. Ο αλγόριθμος εκτελείται όπως έχει περιγραφεί, αλλά οι εντολές κίνησης του ρομπότ αποστέλλονται μέσω ενός HTTP Connectionn στο περιβάλλον προσομοίωσης του Ρομπότ API[A [28], το οποίο εκτελεί το ίδιο πρόγραμμα πλοήγησης.

Για να δοκιμαστεί η αποτελεσματικότητα του αλγορίθμου δοκιμάστηκαν διάφορες θέσεις εκκίνησης του Ρομπότ (κοντά – μακριά από τον άνθρωπο) καθώς και θέσεις όπου ο άνθρωπος κινούνταν σε αντίθετη και ίδια κατεύθυνση με το ρομπότ, κάθετα και τέλος κίνηση όπου ο άνθρωπος δεν κινούταν πάνω σε κάποια ευθεία αλλά έκανε μία πιο πολύπλοκη , κυκλική κίνηση. Συγκεκριμένα τα πειράματα ήταν

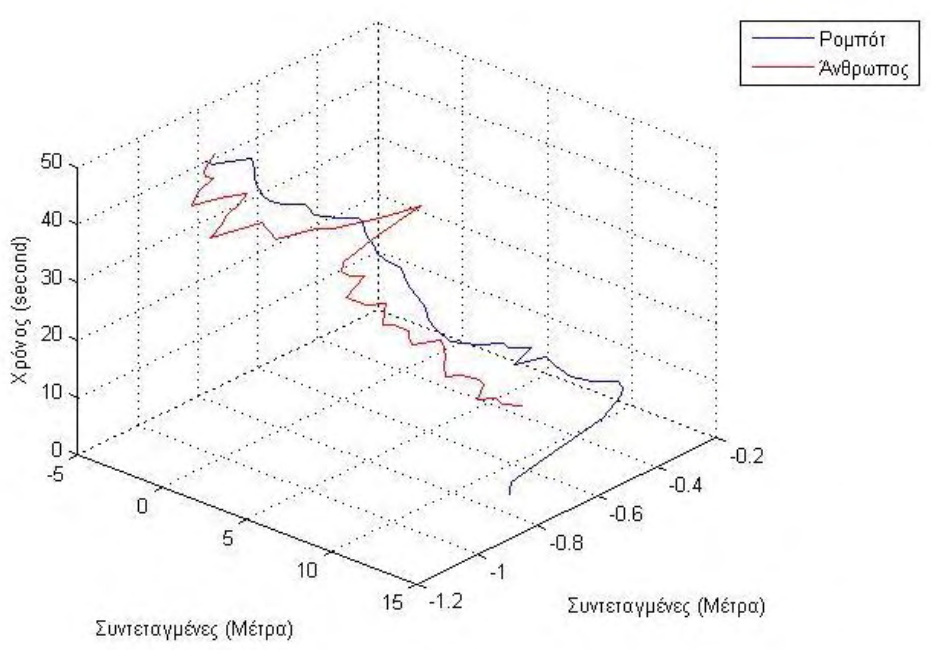
1. Αρχική θέση του Ρομπότ μακριά από αυτήν του ανθρώπου και κίνηση προς την ίδια κατεύθυνση
2. Αρχική θέση του Ρομπότ κοντά από αυτήν του ανθρώπου και κίνηση προς την ίδια κατεύθυνση
3. Αρχική θέση του Ρομπότ μακριά από αυτήν του ανθρώπου και κίνηση προς την αντίθετη κατεύθυνση (2 πειράματα)
4. Αρχική θέση του Ρομπότ κοντά σε αυτήν του ανθρώπου και κάθετα προς την κίνηση του
5. Βίντεο με 2 ανθρώπους , με πολύπλοκη (κυκλική κίνηση) όπου το ρομπότ θα έπρεπε αντίστοιχα να ακολουθήσει τον πρώτο και στην συνέχεια ($2^{ο}$ πείραμα) τον δεύτερο άνθρωπο.

Στα παρακάτω σχήματα φαίνονται οι πραγματικές θέσεις του ανθρώπου και του ρομπότ καταγράφοντας την θέση τους κάθε 1 δευτερόλεπτο, με Α.Θ. σημειώνεται η Αρχική Θέση του ανθρώπου και του Ρομπότ. Όλες οι συντεταγμένες είναι σε μέτρα από το σημείο (0,0) όπου βρίσκεται και η κάμερα.

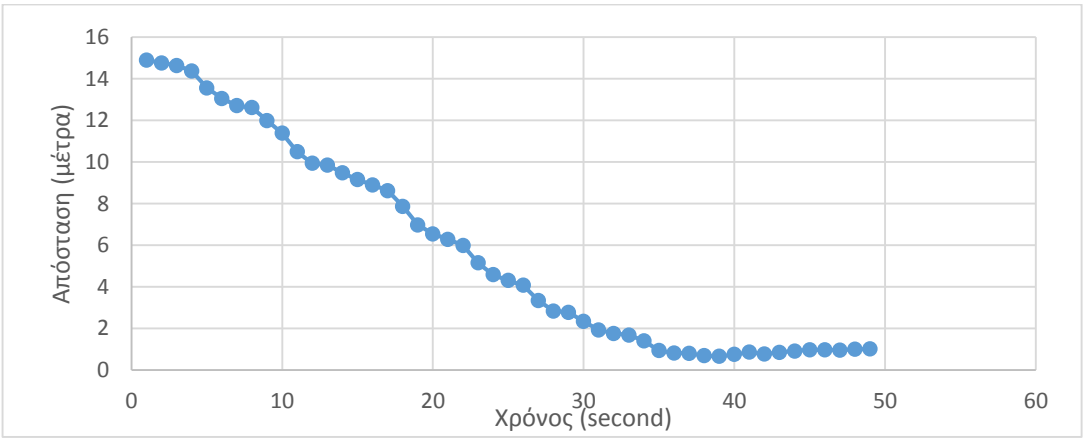
Πείραμα 1: Αρχική θέση του Ρομπότ μακριά από αυτήν του ανθρώπου και κίνηση προς την ίδια κατεύθυνση



Διάγραμμα 1: Θέση Ρομπότ και ανθρώπου

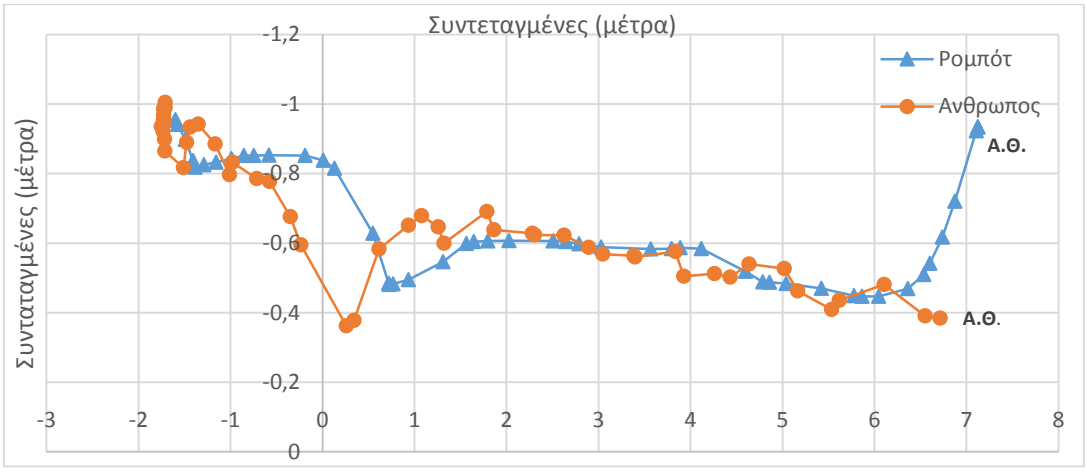


Διάγραμμα 2: Θέση Ρομπότ και Ανθρώπου στον χρόνο

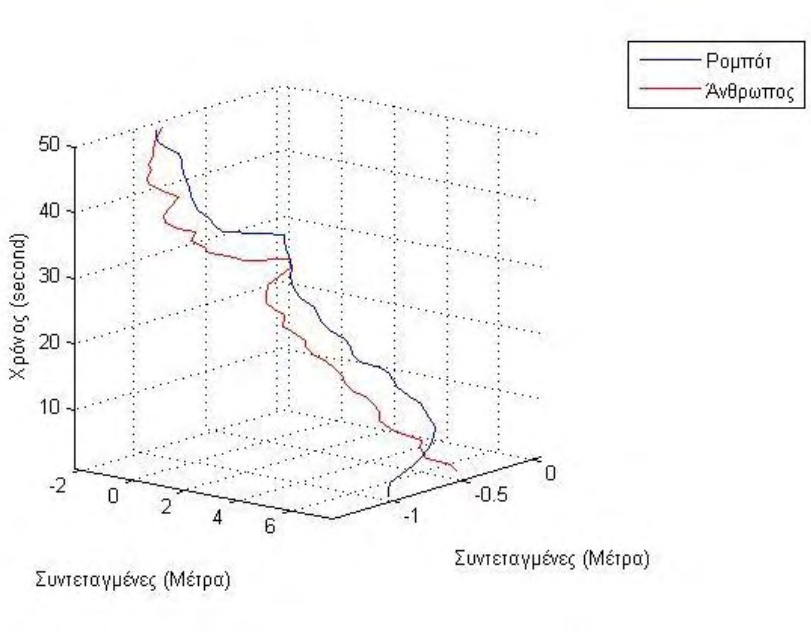


Διάγραμμα 3: Απόσταση Ρομπότ και Ανθρώπου στον Χρόνο

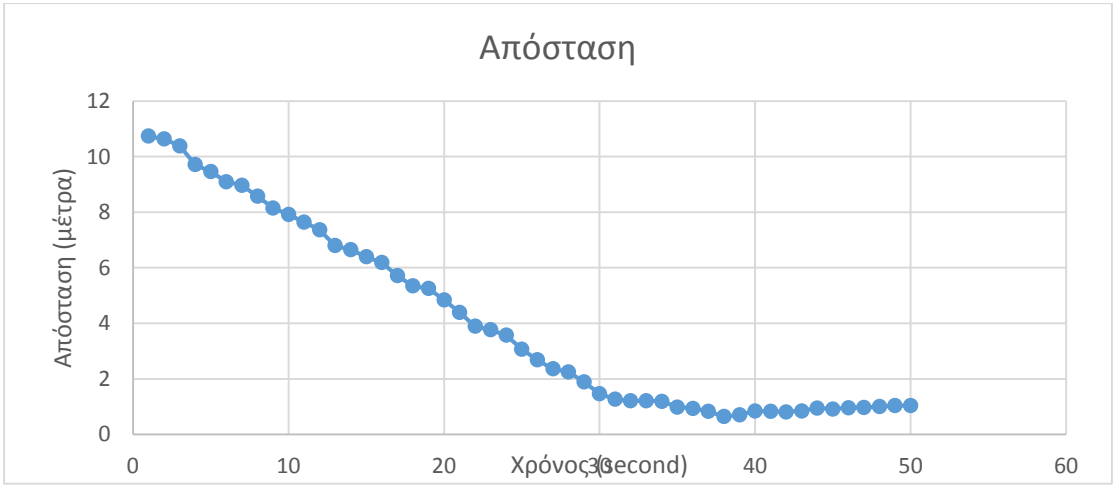
Πείραμα 2: Αρχική θέση του Ρομπότ κοντά από αυτήν του ανθρώπου και κίνηση προς την ίδια κατεύθυνση



Διάγραμμα 4: Θέση Ρομπότ και Ανθρώπου

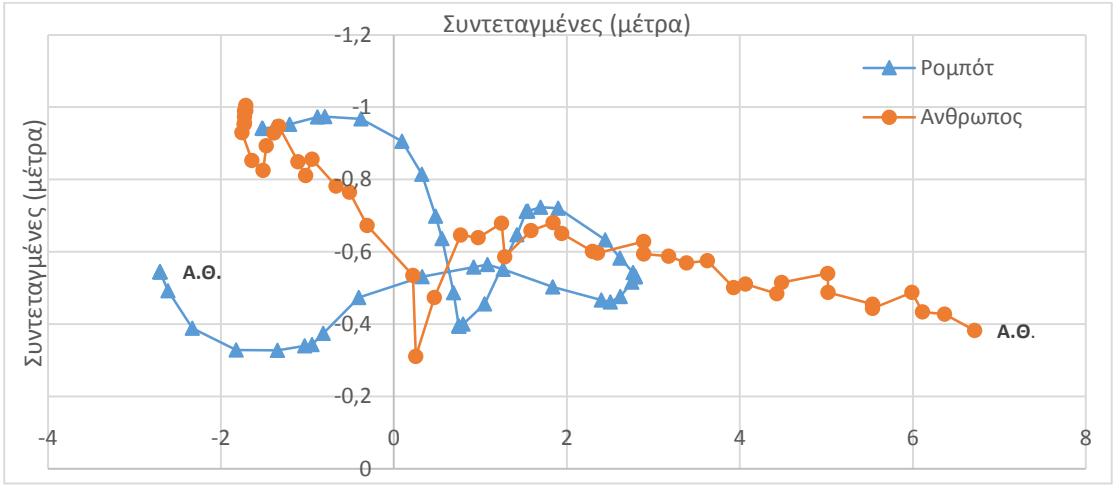


Διάγραμμα 5: Θέση Ρομπότ και Ανθρώπου στον Χρόνο

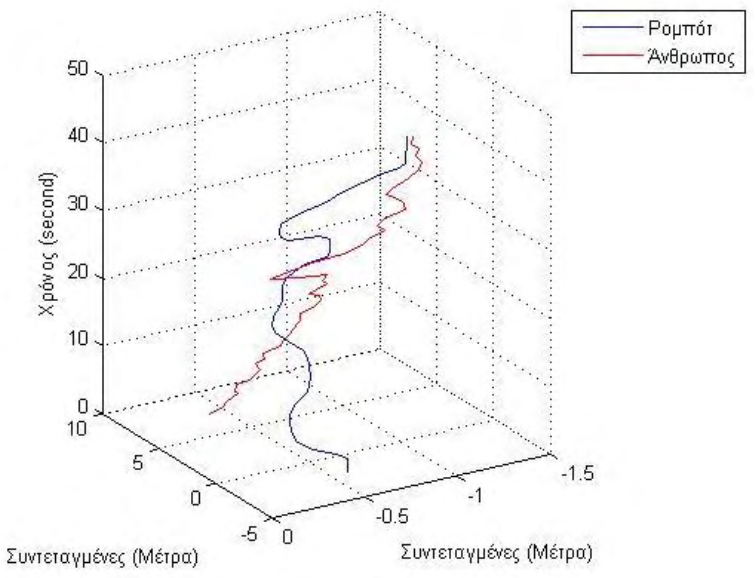


Διάγραμμα 6: Απόσταση Ρομπότ και Ανθρώπου στον Χρόνο

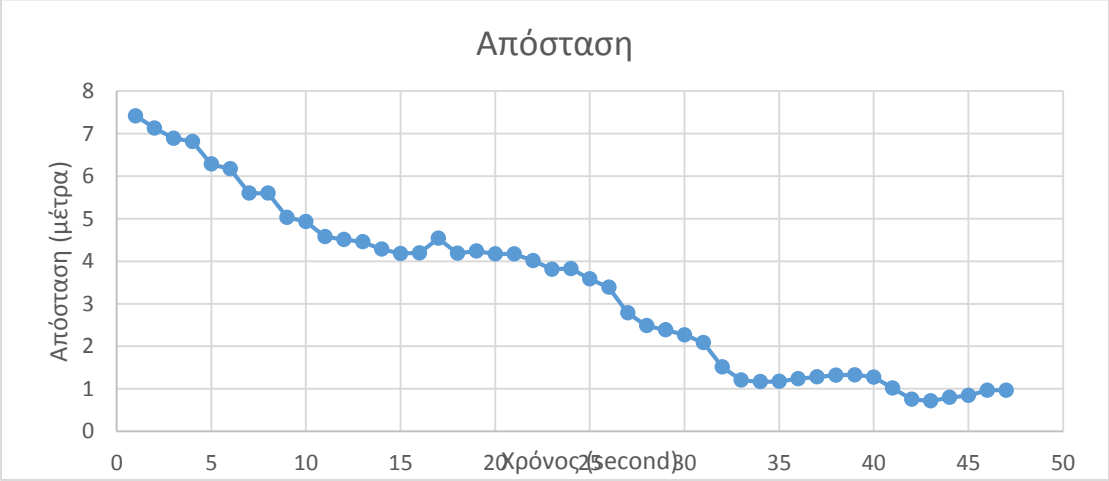
Πείραμα 3(α): Αρχική θέση του Ρομπότ μακριά από αυτήν του ανθρώπου και κίνηση προς την αντίθετη κατεύθυνση (1)



Διάγραμμα 7: Θέση Ρομπότ και Ανθρώπου

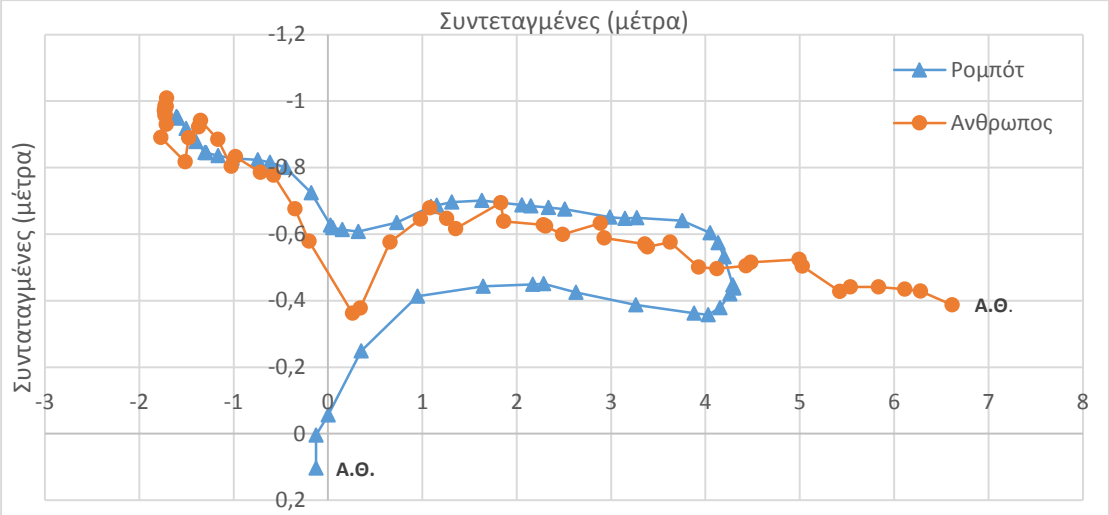


Διάγραμμα 8: Θέση Ρομπότ και Ανθρώπου στον Χρόνο

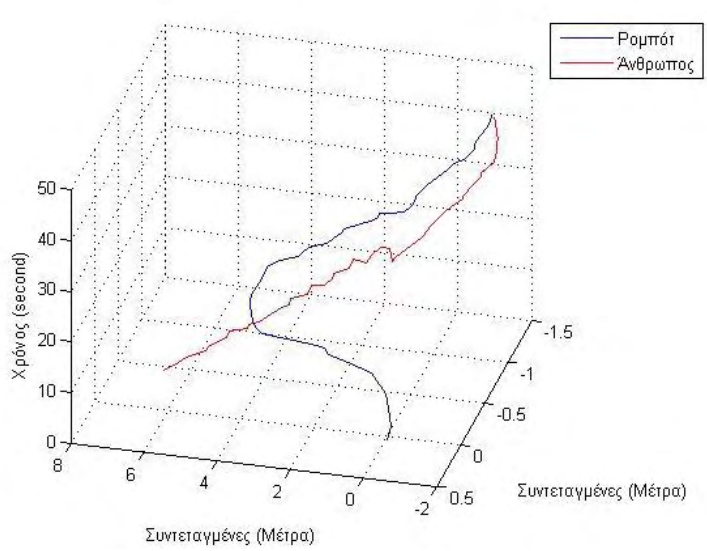


Διάγραμμα 9: Απόσταση Ρομπότ και Ανθρώπου στον Χρόνο

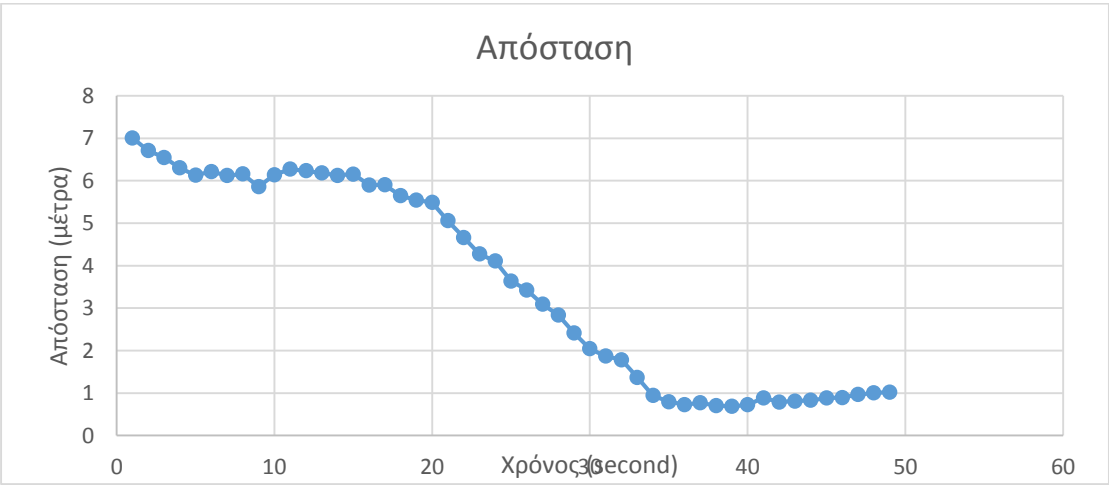
Πείραμα 3(β): Αρχική θέση του Ρομπότ μακριά από αυτήν του ανθρώπου και κίνηση προς την αντίθετη κατεύθυνση (2)



Διάγραμμα 10: Θέση Ρομπότ και Ανθρώπου

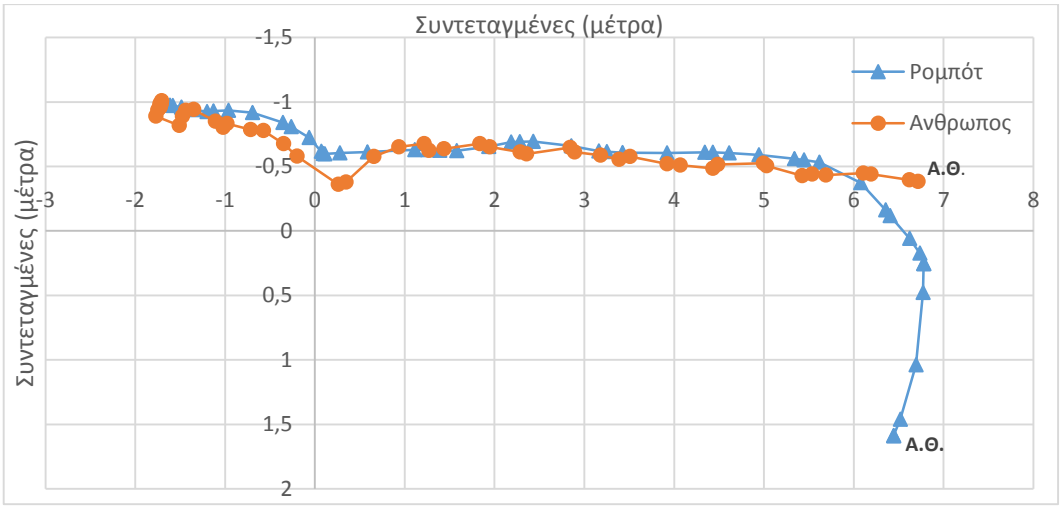


Διάγραμμα 11: Θέση Ρομπότ και Ανθρώπου στον Χρόνο

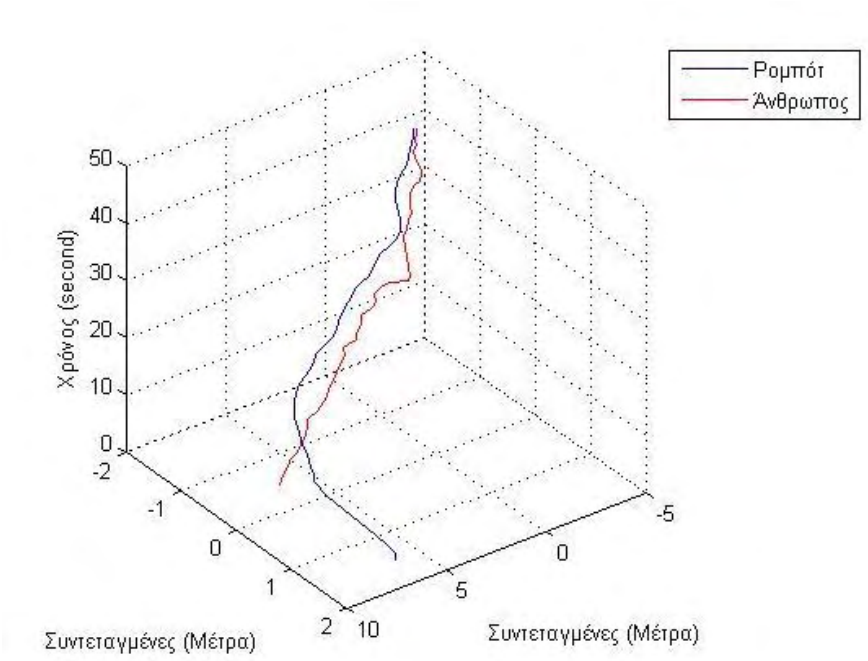


Διάγραμμα 12: Απόσταση Ρομπότ και Ανθρώπου στον Χρόνο

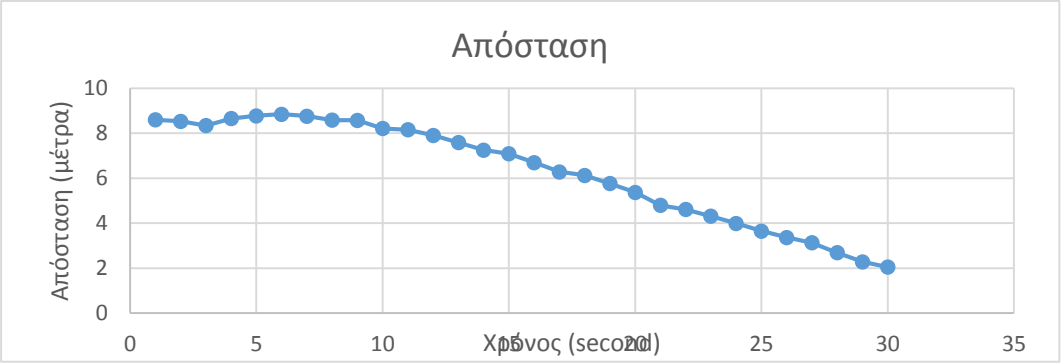
Πείραμα 4: Αρχική θέση του Ρομπότ κοντά σε αυτήν του ανθρώπου και κάθετα προς την κίνηση του



Διάγραμμα 13: Θέση Ρομπότ και Ανθρώπου

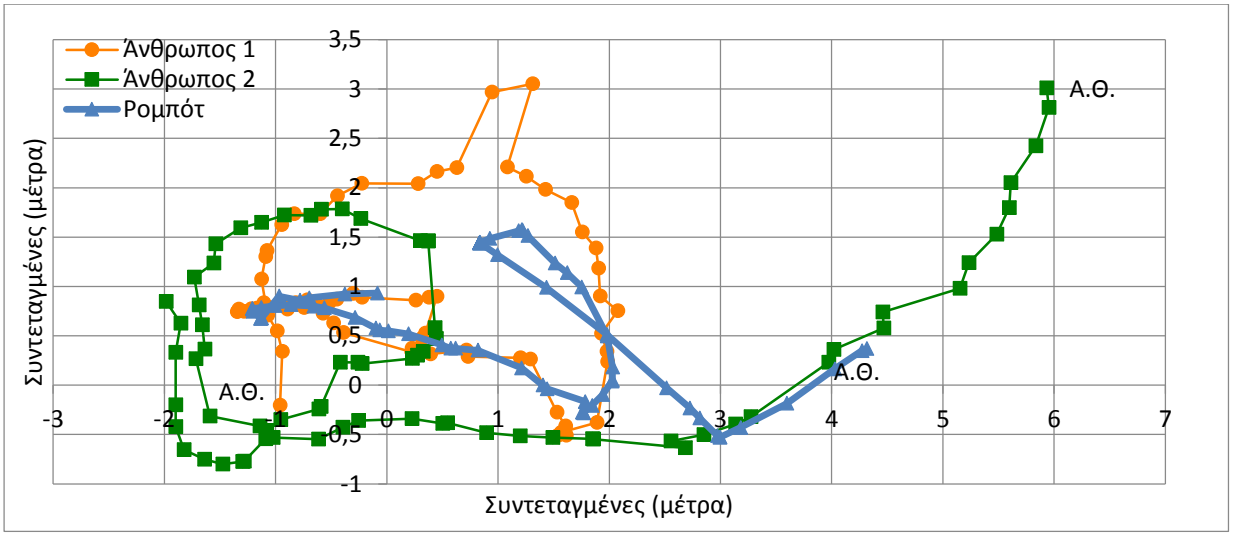


Διάγραμμα 14: Θέση Ρομπότ και Ανθρώπου στον Χρόνο

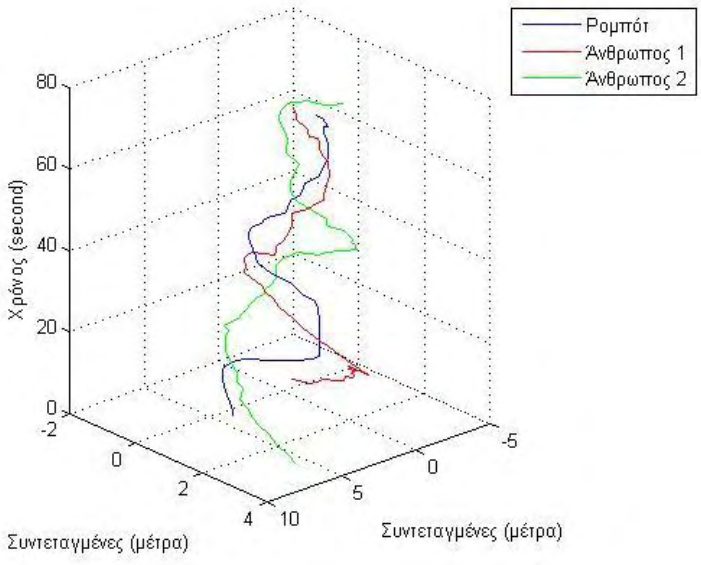


Διάγραμμα 15: Απόσταση Ρομπότ και Ανθρώπου στον Χρόνο

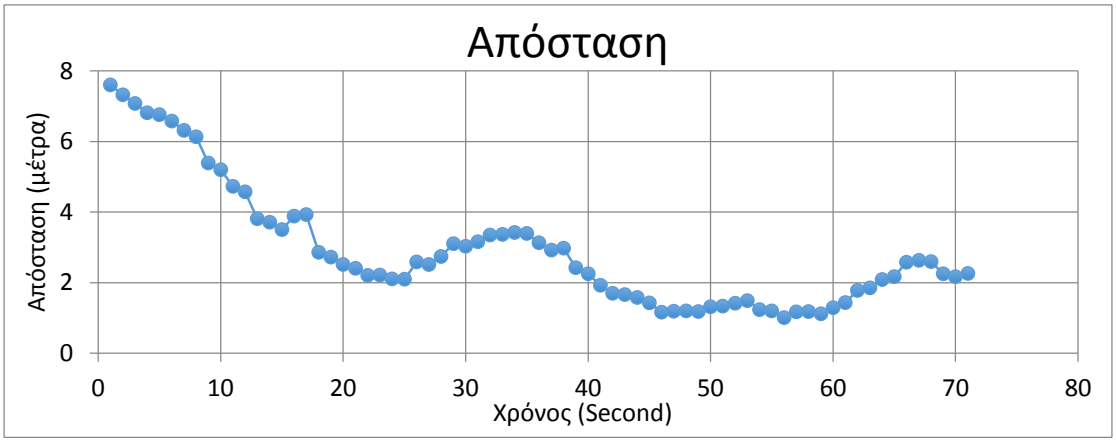
Πείραμα 5(α): Βίντεο με 2 ανθρώπους , με πολύπλοκη (κυκλική κίνηση) όπου το ακολουθάει τον 1 c



Διάγραμμα 16: Θέση Ρομπότ και Ανθρώπου

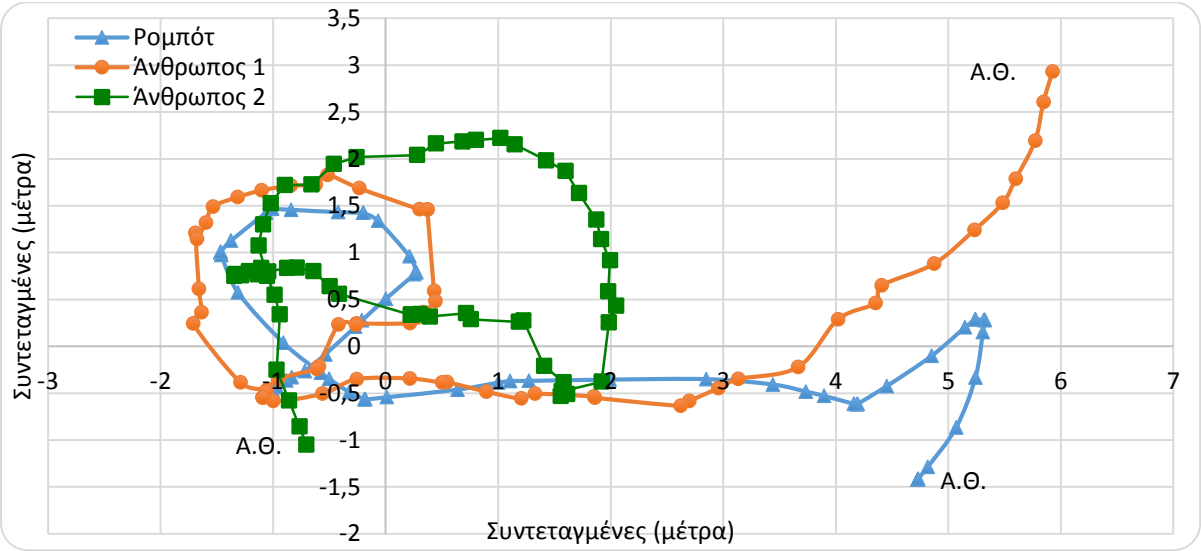


Διάγραμμα 17: Θέση Ρομπότ και Ανθρώπων στον Χρόνο

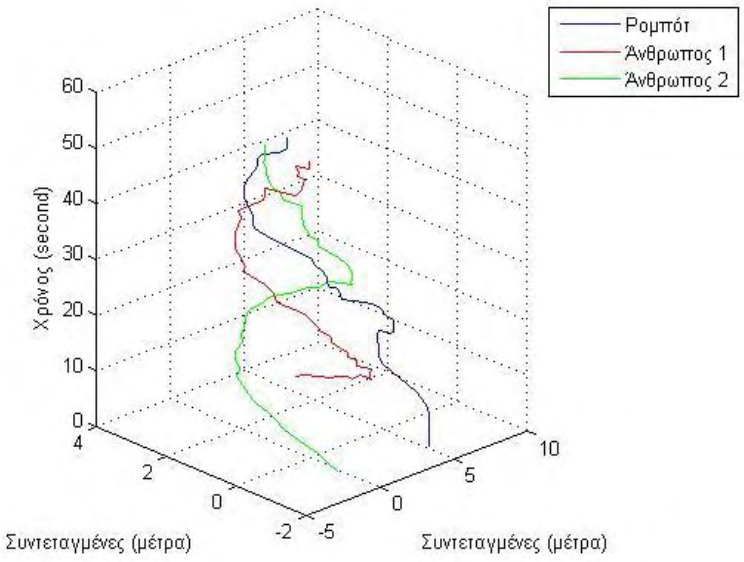


Διάγραμμα 18: Απόσταση Ρομπότ και Ανθρώπου 1 στον Χρόνο

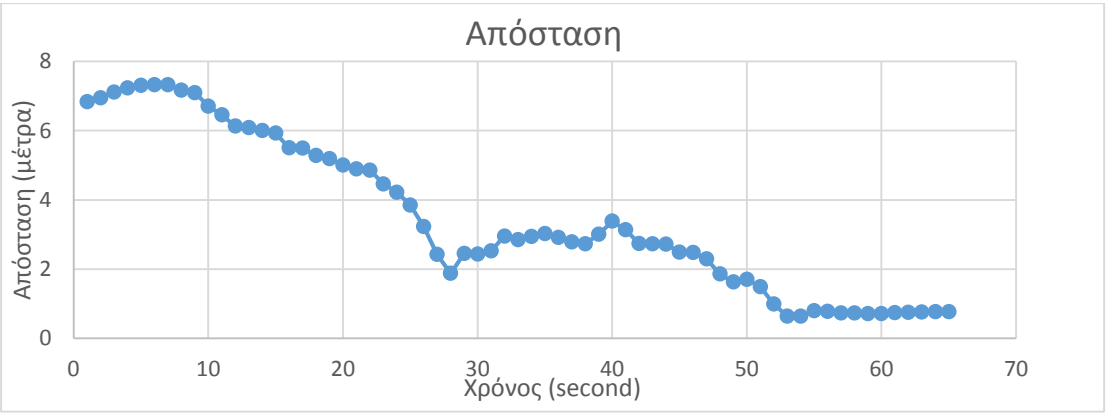
Πείραμα 5(β): Βίντεο με 2 ανθρώπους , με πολύπλοκη (κυκλική κίνηση) όπου το Ρομπότ ακολουθάει τον 1 (Άνθρωπος 1).



Διάγραμμα 19: Θέση Ρομπότ και Ανθρώπου



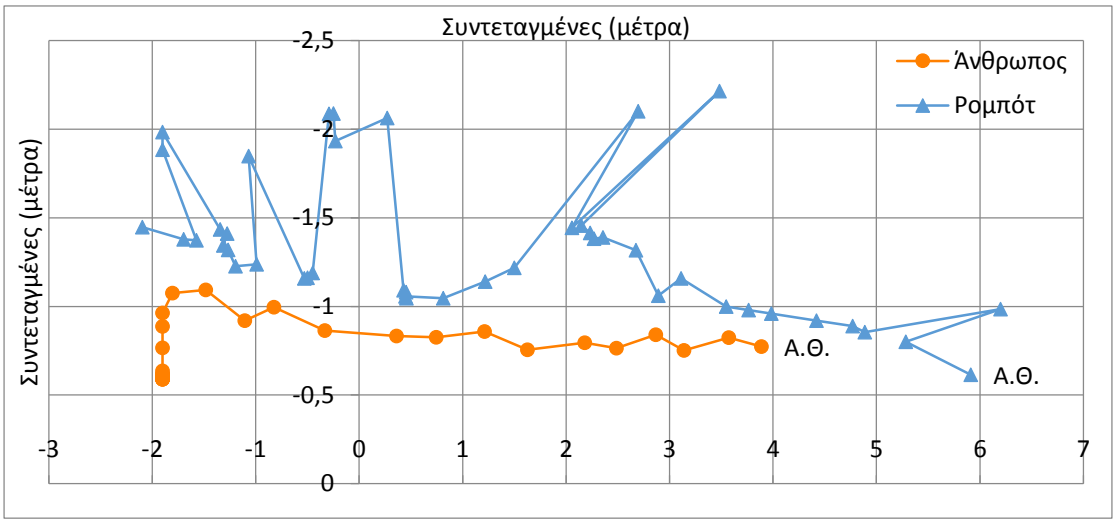
Διάγραμμα 20: Θέση Ρομπότ και Ανθρώπων στον Χρόνο



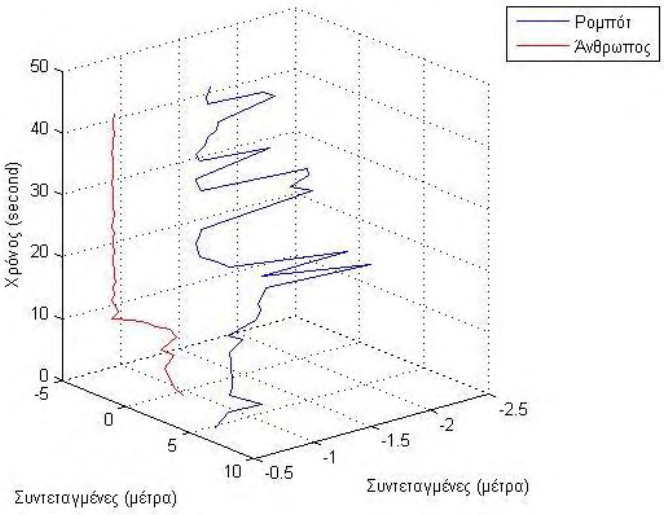
Διάγραμμα 21: Απόσταση Ρομπότ και Ανθρώπου 1 στον Χρόνο

4.2 Αποτελέσματα Πραγματικών Πειραμάτων

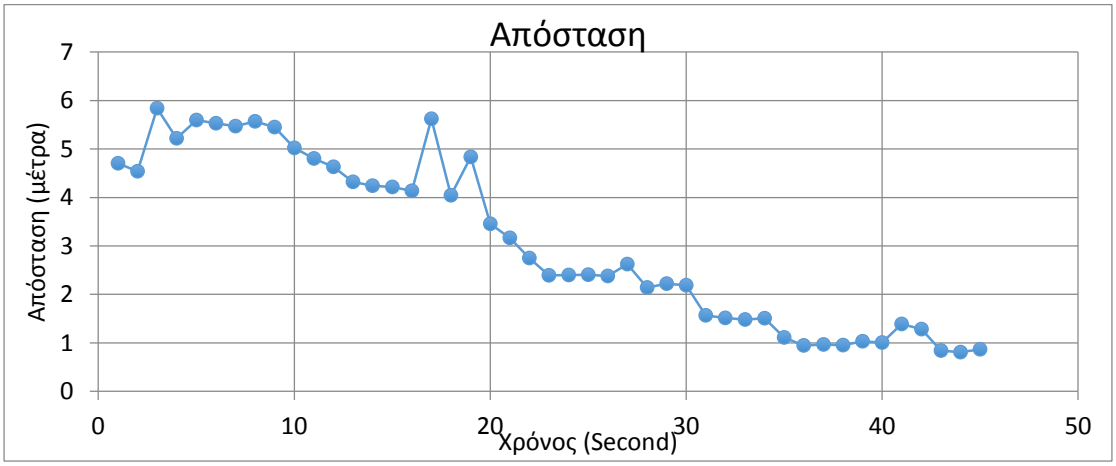
Πείραμα 1: Αρχική θέση του Ρομπότ κοντά από αυτήν του ανθρώπου και κίνηση προς την ίδια κατεύθυνση



Διάγραμμα 22: ***Θέση Ρομπότ και Ανθρώπου***

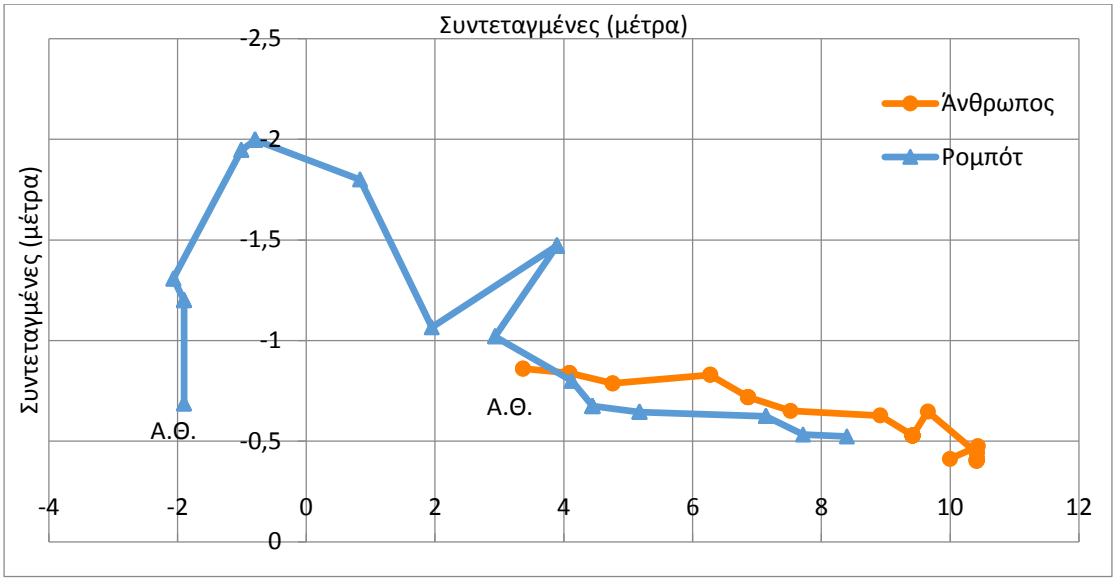


Διάγραμμα 23: : ***Θέση Ρομπότ και Ανθρώπου στον Χρόνο***

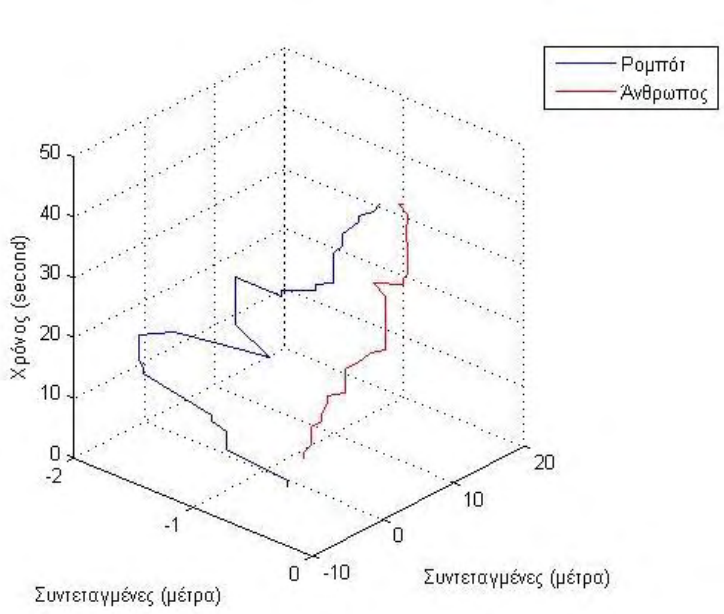


Διάγραμμα 24: Απόσταση Ρομπότ και Ανθρώπου στον Χρόνο

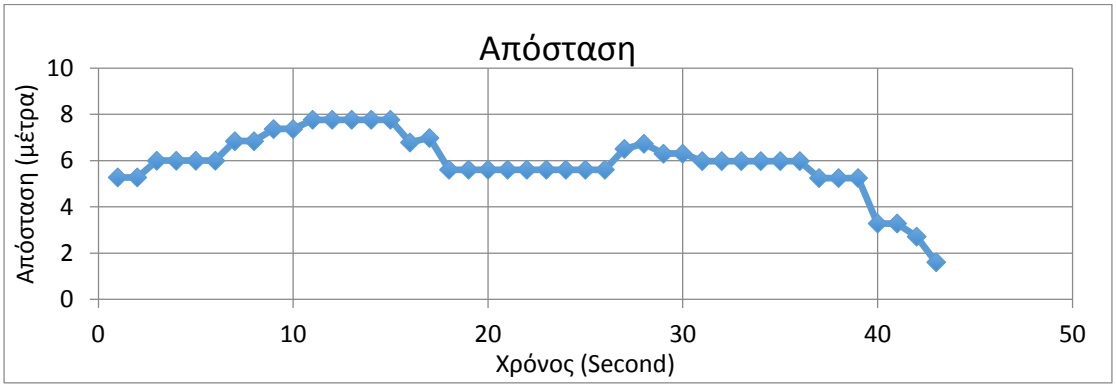
Πείραμα 2: Αρχική θέση του Ρομπότ κοντά από αυτήν του ανθρώπου και κίνηση προς την ίδια κατεύθυνση



Διάγραμμα 25: Θέση Ρομπότ και Ανθρώπου



Διάγραμμα 26: Θέση Ρομπότ και Ανθρώπου στον Χρόνο



Διάγραμμα 27: Απόσταση Ρομπότ και Ανθρώπου στον Χρόνο

Κεφάλαιο 5

Συμπεράσματα

Σκοπός της παρούσας πτυχιακής εργασίας ήταν η πλοήγηση ενός ρομπότ σε εσωτερικό χώρο με χρήση μιας fisheye κάμερας οροφής και πιο συγκεκριμένα η αναγνώριση της ανθρώπινης δραστηριότητας και η παρακολούθηση ενός ανθρώπου από ένα Ρομπότ. Για τον λόγο αυτόν αναπτύχθηκαν οι αλγόριθμοι που παρουσιάστηκαν στην ενότητα 3.

Όπως φαίνεται και στην ενότητα 4 με τα αποτελέσματα, το Ρομπότ μέσω των αλγορίθμων που αναπτύχθηκαν είναι σε θέση μέσα σε μερικά δευτερόλεπτα να φτάσει σε κοντινή απόσταση από τον άνθρωπο που πρέπει να παρακολουθήσει και να συνεχίσει να τον παρακολουθεί. Στα σχεδιαγράμματα φαίνεται ότι η ελάχιστη απόσταση που φτάνει το Ρομπότ από τον άνθρωπο είναι 1 – 2 μέτρα. Αυτή θεωρείται και η ιδανική απόσταση λόγω του ότι δεν επιθυμούμε το Ρομπότ να «κολλήσει» πάνω στον άνθρωπο αλλά θέλουμε να υπάρχει ένα περιθώριο για περεταίρω κινήσεις. Επίσης όπως βλέπουμε στα πειράματα 5(α) και 5(β) ο αλγόριθμος είναι σε θέση να διακρίνει ανάμεσα από 2 ανθρώπους που βρίσκονται στο δωμάτιο ποιόν πρέπει να παρακολουθήσει .

Όσον αφορά το ρομπότ δεν χρησιμοποιήθηκε κανένας από τους αισθητήρες του για την αναγνώριση της ανθρώπινης δραστηριότητας. Για τον λόγο αυτό μπορεί να χρησιμοποιηθεί και κάποιο πολύ φθηνότερο με λιγότερες δυνατότητες Ρομπότ και να έχουμε τα ίδιο ή παρόμοια αποτελέσματα.

Πιθανή χρήση των αλγορίθμων που αναπτύχθηκαν είναι η χρήση ενός ρομπότ στα πλαίσια ενός έξυπνου σπιτιού για υποβοήθηση των ανθρώπων που διαμένουν στο σπίτι.

Για την μελλοντική ανάπτυξη των αλγορίθμων θα μπορούσε να χρησιμοποιηθεί κάποιος πιο γρήγορος αλγόριθμος τμηματοποίησης για ακόμα καλύτερα αποτελέσματα καθώς και η περαιτέρω εξέλιξη του αλγορίθμου που διακρίνει το αντικείμενο υπό παρακολούθηση από τα υπόλοιπα αντικείμενα ώστε να μπορεί να διακρίνει τον άνθρωπο μέσα από μια ομάδα ανθρώπων χωρίς να μπερδεύεται. Τέλος θα μπορούσε να χρησιμοποιηθεί η θέση και η ταχύτητα του ανθρώπου που παρακολουθείται ώστε να προβλεφθεί μια μελλοντική θέση του ανθρώπου και να σταλεί αυτή στο Ρομπότ.

Κεφάλαιο 6

Βιβλιογραφία

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

[2] http://en.wikipedia.org/wiki/Lisp_(programming_language)

[3] http://en.wikipedia.org/wiki/Prolog

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

[5] http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[6] http://en.wikipedia.org/wiki/Robotics

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

[8] http://en.wikipedia.org/wiki/Fisheye_lens

[9] Morioka, K, Joo-Ho Lee ; Hashimoto, H., Human-following mobile robot in a distributed intelligent sensor network, Industrial Electronics, IEEE Transactions on (Volume:51 , Issue: 1), Feb. 2004

[10] http://www.mobilerobots.com/ResearchRobots/PeopleBot.aspx

[11] https://java.com/en/download/index.jsp

[12] http://opencv.org/

[13] N. Max 1983, Computer Graphics Distortion for IMAX and OMNIMAX Projection, *Proc Nicograph*, 83, Dec 1983 pp 137.

[14] Greene N. 1986, Environment Mapping and Other Applications of World Projections, *IEEE Computer Graphics and Applications*, November 1986, vol. 6(11), pp 21.

[15] http://paulbourke.net/dome/fisheye/

[16] KK Delibasis, T Goudas, VP Plagianakos, I Maglogiannis, Fisheye camera modeling for human segmentation refinement in indoor videos, Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments.

[17] B. Micusik and T. Pajdla, "Structure from Motion with Wide Circular Field of View Cameras", IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI 28(7), 2006, pp. 1-15.

[18] J. Canny, "A Computational Approach To Edge Detection", IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 8(6), 1986, pp. 679–698.

[19] http://en.wikipedia.org/wiki/Sonar

[20] https://www.eclipse.org/

[21] http://www.ubuntu.com

[22] Γιαχούδης Νικόλαος , Ευφυής έλεγχος και πλοήγηση ρομπότ μέσω της πλατφόρμας ARIA, Πτυχιακή Εργασία Πανεπιστήμιο Θεσσαλίας Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρκή, Λαμία 2014

[23] Cheng F.C., Huang S.C. and Ruan S.J. 2011, Implementation of Illumination-Sensitive Background Modeling Approach for Accurate Moving Object Detection, *IEEE Trans. onBoardcasting*, vol. 57, no. 4, pp.794-801, 2011.

[24] Christodoulidis A., Delibasis K. K., Maglogiannis I. 2012, Near real-time human silhouette and movement detection in indoor environments using fixed cameras, in *the 5th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, Heraklion, Crete, Greece, 2012.

[25] http://en.wikipedia.org/wiki/HSL_and_HSV

[26] Cucchiara, R., Grana, C., Piccardi, M., and Prati A. 2003, Detecting moving objects, ghosts, and shadows in video streams. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 10, (2003), 1337-1442

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

Κεφάλαιο 7

Παράρτημα

Α΄ Κώδικας

```
public class Connection {

    private String host;
    private int port;
    private Socket client = null;
    private BufferedReader in = null;
    private PrintWriter out = null;
    private double x;
    private double y;

    public Connection() {
        this.host = "localhost";
        this.port = 30000;
        this.x = 0;
        this.y = 0;
        connection();
    }

    public Connection(String hostName, int portNumber) {
        setHost(hostName);
        setPort(portNumber);
        connection();
    }

    public void setHost(String hostName) {
        this.host = hostName;
    }

    public void setPort(int portNumber) {
        this.port = portNumber;
    }

    public void setXY(double x, double y) {
        double t[] = Tools.getRobotCoords(x, y);
        this.x = t[0];
        this.y = t[1];
    }

    public void connection() {
        try {
            connectToServer();
            getStreams(); // get Streams from server
            processConnection();
        } catch (IOException ioe) {
            System.err.println(ioe);
        } // connect to server

    }

    public void runClient() {
            processConnection();
```

```
private void connectToServer() throws IOException {
        System.out.println("Attempting connection...");
        client = new Socket(host, port);
        System.out.println("Connected to " +
client.getInetAddress().getHostAddress() + " on port " + client.getPort());
    }

    private void getStreams() throws IOException {

        out = new PrintWriter(client.getOutputStream(), true);

        in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
    }

    private void processConnection() {
        try {
            ScheduledExecutorService ses =
Executors.newSingleThreadScheduledExecutor();

            ses.scheduleAtFixedRate(new Runnable() {

                @Override
                public void run() {
                    out.print(-8000 + ";" + Connection.this.x + ";" +
Connection.this.y);
                    out.flush();
                }
            }, 0, 5, TimeUnit.SECONDS); // every 5 seconds
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    private void closeStreams() throws IOException {
        if (out != null) {
            out.close();
        }
        if (in != null) {
            in.close();
        }
    }

    public void closeConnection() throws IOException {

        closeStreams();

        if (client != null) {
            client.close();
        }
    }

}
```

Connection.java

```
public class geom_proc1_all_v2 {

    private static double phi_tresh = Math.PI / 10;
    private static double theta_tresh0 = Math.PI / 20;
    private static double xgon = -4.29;
    private static double ygon = -1.9;
    private static double zmin = 1.3;
    private static double zmax = 3.5;

    public static double[] geom_proc1_all_v2(Mat L, double[][] Theta,
double[][] Phi, int[][] Mask, MatOfPoint person) {

        ArrayList<CustomPoints> phi_points = new ArrayList<CustomPoints>();
        ArrayList<CustomPoints> theta_points = new
ArrayList<CustomPoints>();

        List<Point> t = person.toList();

        for (int i = 0; i < t.size(); i++) {
            int pointx = (int) t.get(i).x;
            int pointy = (int) t.get(i).y;
            phi_points.add(new CustomPoints(Phi[pointy][pointx], t.get(i)));
            theta_points.add(new CustomPoints(Theta[pointy][pointx],
t.get(i)));

        }

        CustomPoints phi_min = Collections.min(phi_points);
        CustomPoints phi_max = Collections.max(phi_points);
        CustomPoints theta_min = Collections.min(theta_points);
        CustomPoints theta_max = Collections.max(theta_points);

        if ((theta_max.getValue() > Math.PI / 2) && (theta_min.getValue() <
-Math.PI / 2)) {
            theta_min.setValue(theta_min.getValue() + 2 * Math.PI);
            for (int i = 0; i < theta_points.size(); i++) {
                if (theta_points.get(i).getValue() < 0) {

theta_points.get(i).setValue(theta_points.get(i).getValue() + 2 * Math.PI);
                }
            }
        }

        double theta_avg = calculateAverage(theta_points);

        double RR = zmax / Math.sin(phi_max.getValue());
        Point real = sph2cart(theta_avg, phi_max.getValue(), RR);
        Point real_phimax = new Point(real.x, real.y);
        double sum = 0;
        for (int i = 0; i < Mask.length; i++) {
            for (int j = 0; j < Mask[0].length; j++) {
                sum = sum + Mask[i][j] * L.get(i, j)[0];
            }
        }
        if (sum > (100 * 255)) {
            double RR0 = zmax / Math.sin(phi_min.getValue());
            Point centroid = getCentroid(person);
            theta_avg = Theta[(int) centroid.y][(int) centroid.x];
            Point real0 = sph2cart(theta_avg, phi_min.getValue(), RR0);
            real.x = (real.x + real0.x) / 2;
            real.y = (real.y + real0.y) / 2;
```

```
            real = sph2cart(theta_avg, calculateAverage(phi_points), RR0);

        } else {
            real.x = real.x + 0.2 * Math.signum(real.x);
            real.y = real.y + 0.2 * Math.signum(real.y);
        }

        if (real_phimax.x < xgon) {
            double lambda = xgon / real.x;
            real.x = xgon;
            real.y = real.y * lambda;
        }
        if (real_phimax.y < ygon) {
            double lambda = ygon / real.y;
            real.y = ygon;
            real.x = real.x * lambda;
        }
        return new double[] {
                real.x, real.y
        };

    }

}
```

geom_proc_all_v2.java

```
package vid_seg;

public class indexes implements Comparable<indexes> {
    public final double dist;
    public final double mean;
    public final int pos;
    public final long size;

    public indexes(double dist, double mean, int pos, long size) {
        this.dist = dist;
        this.mean = mean;
        this.pos = pos;
        this.size = size;
    }

    @Override
    public int compareTo(indexes other) {
        return Double.compare(dist, other.dist);
    }

    @Override
    public String toString() {
        return String.format("dist=" + dist + "\tmean=" + mean + "\tpos=" +
pos + "\tsize=" + size);
    }
}
```

indexes.java

```java
package vid_seg;

public class graphics {
    private int fps = 0;
    private int robot_i = -1;
    private int robot_j = -1;
    private int human_i = -1;
    private int human_j = -1;
    private int sec_human_i = -1;
    private int sec_human_j = -1;
    private final Object LOCK = new Object(); // just something to lock on
    private final String startActionCommand = "StartActionCommand";
    private final String stopActionCommand = "StoptActionCommand";
    private final String setHuman1ActionCommand = "setHuman1ActionCommand";
    private final String setHuman2ActionCommand = "setHuman2ActionCommand";
    private final String setRobotActionCommand = "setRobotActionCommand";
    private final String getScreenshot = "getScreenshot";

    JButton setHuman1;
    JButton setHuman2;
    JButton setRobot;
    int current_frame = 0;

    static JTextArea frame_n;
    static JTextArea fps_n;

    static BufferedImage img;
    static JPanel superContainer;
    Video vid;
    private boolean isRunning = false;
    JLabel thumb;
    ImageIcon icon;
    JFrame Jframe;

    public graphics() {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

        Jframe = new JFrame("Image");
        Jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        icon = new ImageIcon();

        thumb = new JLabel();
        thumb.addMouseListener(new MouseListener() {

            @Override
            public void mouseReleased(MouseEvent arg0) {
                // TODO Auto-generated method stub
            }

            @Override
            public void mousePressed(MouseEvent arg0) {
                // TODO Auto-generated method stub
            }

            @Override
            public void mouseExited(MouseEvent arg0) {
                // TODO Auto-generated method stub
            }

            @Override
            public void mouseEntered(MouseEvent arg0) {
                // TODO Auto-generated method stub

            }
```

```
      @Override
            public void mouseClicked(MouseEvent e) {
                if (!setRobot.isEnabled()) {
                    System.out.println("robot x = " + e.getX() + " y = " +
e.getY());
                    robot_i = e.getX();
                    robot_j = e.getY();
                    setRobot.setEnabled(true);
                } else if (!setHuman1.isEnabled()) {
                    System.out.println("human x = " + e.getX() + " y = " +
e.getY());
                    human_i = e.getX();
                    human_j = e.getY();
                    setHuman1.setEnabled(true);
                } else if (!setHuman2.isEnabled()) {
                    System.out.println("sec human x = " + e.getX() + " y = "
+ e.getY());
                    sec_human_i = e.getX();
                    sec_human_j = e.getY();
                    setHuman2.setEnabled(true);
                } else {
                    System.out.println(" x = " + e.getX() + " y = " +
e.getY());
                }

            }
        });

        superContainer = new JPanel();
        superContainer.setLayout(new BorderLayout());

        frame_n = new JTextArea();
        frame_n.setText("Frame: " + 0);
        fps_n = new JTextArea();
        fps_n.setText("Fps: " + 0);

        superContainer.add(thumb, BorderLayout.CENTER);
        ButtonHandler handler = new ButtonHandler();

        JButton start = new JButton();
        start.setActionCommand(startActionCommand);
        start.setText("Connect and start");
        start.addActionListener(handler);

        JButton stop = new JButton();
        stop.setActionCommand(stopActionCommand);
        stop.setText("Stop");
        stop.addActionListener(handler);

        setHuman1 = new JButton();
        setHuman1.setText("Set Human1");
        setHuman1.setActionCommand(setHuman1ActionCommand);
        setHuman1.addActionListener(handler);

        setHuman2 = new JButton();
        setHuman2.setText("Set Human2");
        setHuman2.setActionCommand(setHuman2ActionCommand);
        setHuman2.addActionListener(handler);
```

```
        setRobot = new JButton();
        setRobot.setText("Set Robot");
        setRobot.setActionCommand(setRobotActionCommand);
        setRobot.addActionListener(handler);

        JButton screenshot = new JButton();
        screenshot.setText("Screenshot");
        screenshot.setActionCommand(getScreenshot);
        screenshot.addActionListener(handler);

        JPanel connection_panel = new JPanel();
        connection_panel.setLayout(new GridLayout(8, 1, 2, 2));
        connection_panel.add(start);
        connection_panel.add(stop);
        connection_panel.add(setHuman1);
        connection_panel.add(setHuman2);
        connection_panel.add(setRobot);
        connection_panel.add(frame_n);
        connection_panel.add(fps_n);
        connection_panel.add(screenshot);

        superContainer.add(connection_panel, BorderLayout.EAST);

        ScheduledExecutorService ses =
Executors.newSingleThreadScheduledExecutor();
        ses.scheduleAtFixedRate(new Runnable() {

            @Override
            public void run() {
                fps_n.setText("Fps: " + fps);
                fps = 0;
            }
        }, 0, 1, TimeUnit.SECONDS);
        //

        run();

    }

    private class ButtonHandler implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            if (startActionCommand.equals(e.getActionCommand())) {
                vid.setHumanCoords(human_i, human_j);
                vid.setRobotCoords(robot_i, robot_j);
                vid.setSecHumanCoords(sec_human_i, sec_human_j);
                isRunning = true;
                synchronized (LOCK) {
                    LOCK.notifyAll();
                }
            } else if (stopActionCommand.equals(e.getActionCommand())) {
                isRunning = false;
            } else if (setHuman1ActionCommand.equals(e.getActionCommand())) {
                setHuman1.setEnabled(false);
                setHuman2.setEnabled(true);
                setRobot.setEnabled(true);
            } else if (setHuman2ActionCommand.equals(e.getActionCommand())) {
                setHuman1.setEnabled(true);
                setHuman2.setEnabled(false);
                setRobot.setEnabled(true);
```

```
            } else if (setRobotActionCommand.equals(e.getActionCommand())) {
                setHuman1.setEnabled(true);
                setHuman2.setEnabled(true);
                setRobot.setEnabled(false);
            } else if (getScreenshot.equals(e.getActionCommand())) {
                Calendar now = Calendar.getInstance();
                SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMdd
hh mm ss a");

                try {
                    ImageIO.write(img, "JPG", new
File(formatter.format(now.getTime()) + " f-" + current_frame + ".jpg"));
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }

        }
    }

    private void run() {
        while (true) {
            vid = new Video(Functions.FUNCTION_SEG_SOM);
            Jframe.setSize(vid.getVideoWidth() + 200, vid.getVideoHeight() +
frame_n.getHeight());
            Jframe.setVisible(true);
            Jframe.setContentPane(superContainer);
            img = vid.getFirstFrame();
            icon.setImage(img);
            thumb.setIcon(icon);
            long t = 0;
            synchronized (LOCK) {
                while (!isRunning) {
                    try {
                        LOCK.wait();
                    } catch (InterruptedException e) {
                        break;
                    }
                }
            }
            while (isRunning) {
                long time = System.currentTimeMillis();
                img = vid.run();
                t += (System.currentTimeMillis() - time);
                double[] xy = vid.getHumanCoords();
                frame_n.setText("Frame: " + current_frame++ + "\n x=" +
(int) xy[0] + " y=" + (int) xy[1]);
                fps++;

                icon.setImage(img);
                thumb.repaint();

            }
            System.out.println("Avg time = " + t / current_frame);
        }
    }

}
```

graphics.java

```
package vid_seg;

import com.jmatio.io.MatFileReader;
import com.jmatio.types.MLArray;
import com.jmatio.types.MLDouble;

public class Matlab_loader {

    private static String PHI_ARRAY = "PHI";
    private static String THETA_ARRAY = "THETA";
    private static String DEFAULT_PATH = "path_to_mat_file"

    private double[][] phi;
    private double[][] theta;

    public Matlab_loader(String path) {
        MatFileReader mfr = null;
        try
            mfr = new MatFileReader(path);
        } catch (java.io.IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
        phi = null;
        theta = null;
        if (mfr != null) {
            MLArray field = mfr.getMLArray(THETA_ARRAY);
            theta = ((MLDouble) field).getArray();
            MLArray field2 = mfr.getMLArray(PHI_ARRAY);
            phi = ((MLDouble) field2).getArray();
        } else {
            System.out.println("null");
        }

    }

    public Matlab_loader() {
        this(DEFAULT_PATH);
    }

    public double[][] getPhi() {
        return phi;
    }

    public double[][] getTheta() {
        return theta;
    }
```

```
    public int[][] getMask() {
        // maska Mc
        int[][] mc = new int[theta.length][theta[0].length];

        for (int i = 0; i < theta.length; i++) {
            for (int j = 0; j < theta[0].length; j++) {
                int x = j + 1;
                int y = i + 1;

                if (Tools.getEucDistanceFromCenter(x, y) <= 50) {
                    mc[i][j] = 1;
                } else {
                    mc[i][j] = 0;
                }

            }
        }

        return mc;
    }

}
```

Matlab_loader.java

```
package vid_seg;

public class seg_deriv {

    float[] mask = { -0.0833f, 0.6667f, 0f, -0.6667f, 0.0833f };

    private final double[][] C;
    private final int[][] B;
    private final ArrayList<Mat> frames;
    private final ArrayList<Mat> frames_grayscale;
    private final int VIDEO_HEIGHT;
    private final int VIDEO_WIDTH;
    private VideoCapture camera;
    private final double[][] PHI;
    private final double[][] THETA;
    private final int[][] Mc;
    private final Connection client;
    private boolean is_video = false;
    private String image_path = "";

    public seg_deriv(String source_path, boolean video) {
        frames = new ArrayList<Mat>();
        frames_grayscale = new ArrayList<Mat>();
        is_video = video;
        if (is_video) {
            camera = new VideoCapture(source_path);
            if (!camera.isOpened()) {
                System.out.println("Error");
            } else {
                for (int i = 0; i < mask.length; i++) {
                    frames.add(new Mat());
                    camera.read(frames.get(i));
                    frames_grayscale.add(Tools.toGrayscale(frames.get(i)));
                }
```

```
            }
        } else {
            image_path = source_path;
            for (int i = 0; i < mask.length; i++) {

frames.add(Tools.matify(Tools.getBitmapFromURL(image_path)));
                frames_grayscale.add(Tools.toGrayscale(frames.get(i)));
            }
        }

        VIDEO_HEIGHT = frames.get(0).height();
        VIDEO_WIDTH = frames.get(0).width();

        C = new double[VIDEO_HEIGHT][VIDEO_WIDTH];
        for (int i = 0; i < VIDEO_HEIGHT; i++) {
            for (int j = 0; j < VIDEO_WIDTH; j++) {
                C[i][j] = Tools.COLOR_BLACK;
            }
        }

        B = new int[VIDEO_HEIGHT][VIDEO_WIDTH];

        Matlab_loader ml = new Matlab_loader();
        PHI = ml.getPhi();
        THETA = ml.getTheta();
        Mc = ml.getMask();
        client = new Connection();
    }

    public BufferedImage run_seg_deriv() {
        frames.remove(0);
        frames_grayscale.remove(0);
        if (is_video) {
            frames.add(new Mat());
            camera.read(frames.get(frames.size() - 1));
        } else {
            frames.add(Tools.matify(Tools.getBitmapFromURL(image_path)));
        }
        frames_grayscale.add(Tools.toGrayscale(frames.get(frames.size() -
1)));

        Mat out = applyFilter();
        BufferedImage i = Tools.toBufferedImage(out);

        return i;
    }
```

```
public Mat applyFilter() {
        Mat Fseg = Mat.zeros(VIDEO_HEIGHT, VIDEO_WIDTH, CvType.CV_8U);

        for (int i = 0; i < VIDEO_HEIGHT; i++) {
            for (int j = 0; j < VIDEO_WIDTH; j++) {
                double bit_number = 0;
                for (int k = 0; k < mask.length; k++) {
                    double temp[] = frames_grayscale.get(k).get(i, j);
                    bit_number = bit_number + mask[k] * temp[0];
                }
                C[i][j] = C[i][j] + bit_number;

                if (Math.abs(C[i][j]) > 40) {
                    B[i][j] = Tools.COLOR_WHITE;
                } else {
                    B[i][j] = Tools.COLOR_BLACK;
                }

                Fseg.put(i, j, B[i][j]);
            }
        }

        List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
        List<MatOfPoint> contours_to_del = new ArrayList<MatOfPoint>();
        Imgproc.findContours(Fseg, contours, new Mat(), Imgproc.RETR_LIST,
Imgproc.CHAIN_APPROX_SIMPLE);
        for (int i = 0; i < contours.size(); i++) {
            if (contours.get(i).total() < 50) {
                contours_to_del.add(contours.get(i));
            }
        }
        contours.removeAll(contours_to_del);
        if (contours.size() > 0) {
            // show only the largest contour
            int max_cont_idx = 0;
            for (int i = 0; i < contours.size(); i++) {
                if (contours.get(i).total() >
contours.get(max_cont_idx).total()) {
                    max_cont_idx = i;
                }
            }
            MatOfPoint cont = contours.get(max_cont_idx);
            Imgproc.drawContours(Fseg, contours, max_cont_idx, new
Scalar(255), -1);
            double[] xy = geom_proc1_all_v2.geom_proc1_all_v2(frames.get(0),
THETA, PHI, Mc, cont); // (x,y)
            client.setXY(xy[0], xy[1]);
        }
        return Fseg;
    }

    public int getVideoWidth() {
        return VIDEO_WIDTH;
    }

    public int getVideoHeight() {
        return VIDEO_HEIGHT;
    }
}
```

seg_deriv.java

```
package vid_seg;

import static vid_seg.Tools.aktina;
import static vid_seg.Tools.kentro;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.highgui.VideoCapture;
import org.opencv.imgproc.Imgproc;

public class seg_som_my {

    private static int bg_model_length = 5;
    private static double a_value = 0.1;
    private static double tt_value = 2.8; // 3 for color
    // FRAME IS BLUE , GREEN ,RED (BGR) NOT (RGB) !!!!!!!!!!!!!!
    private static int RED = 2;
    private static int GREEN = 1;
    private static int BLUE = 0;

    private static int[][][] bg_modelR;
    private static int[][][] bg_modelG;
    private static int[][][] bg_modelB;
    private static int[][][] bg_model_grayscale;
    private static int[][][] bg_rgb;

    private final boolean is_grayscale = false;
    private final boolean is_video;
    private final boolean two_people = false;
    private final boolean is_robot = true;
    private static int VIDEO_HEIGHT;
    private static int VIDEO_WIDTH;
    private VideoCapture camera = null;
    private String image_path = "";

    private int last_robot_i = -1;
    private int last_robot_j = -1;
    private int last_human_i = -1;
    private int last_human_j = -1;
    private int sec_human_i = -1;
    private int sec_human_j = -1;
    private double mean1 = -1;
    private double mean2 = -1;
    private boolean isfirst = true;
    private boolean isfirst2 = true;

    private final double[][] pixels_allowed;
    private final double[][] THETA;
    private final double[][] PHI;
    private final int[][] Mc;
    private Mat frame;
    private Mat F_hsv;
    private float[][][] BG_hsv;
    private int[][] found_segm;
```

```
private static Object LOCK = new Object();
    private static Object TRISEGM_LOCK = new Object();
    private static Object SHADOW_LOCK = new Object();
    private boolean shouldRunTrisegm = false;
    private boolean shouldRunShadow = false;
    private CountDownLatch latch;
    private Connection con;

    double[] humancoords = new double[] { -1, -1 };

    int count = 0;

    public seg_som_my(String source_path, boolean is_video) {
        if (is_video) {
            camera = new VideoCapture(source_path);
            for (int i = 0; i < 160; i++) {
                camera.read(new Mat());
            }
        }
        this.is_video = is_video;
        getBgModel(source_path, camera);
        image_path = source_path;
        Matlab_loader ml = new Matlab_loader();
        PHI = ml.getPhi();
        THETA = ml.getTheta();
        Mc = ml.getMask();
        pixels_allowed = Tools.getPixels_allowed_robot(THETA, PHI, -4.29, -
1.29, 3.5, 271.3681, 1.2);
        frame = new Mat();

        ExecutorService executor = Executors.newFixedThreadPool(3);
        Runnable r = new tri_segm();
        Runnable s1 = new prakseis(0, VIDEO_WIDTH / 2);
        Runnable s2 = new prakseis(VIDEO_WIDTH / 2, VIDEO_WIDTH);
        executor.execute(r);
        executor.execute(s1);
        executor.execute(s2);
        con = new Connection();
    }

    public void getBgModel(String image_path, VideoCapture camera) {
        Mat frame = null;
        if (!is_grayscale) {

            bg_modelR = new int[480][640][bg_model_length];
            bg_modelG = new int[480][640][bg_model_length];
            bg_modelB = new int[480][640][bg_model_length];
            bg_rgb = new int[480][640][3];

            for (int k = 0; k < bg_model_length; k++) {
                if (is_video) {
                    frame = new Mat();
                    camera.read(frame);
                } else {
                    frame =
Tools.matify(Tools.getBitmapFromURL(image_path));
                }
```

```
for (int i = 0; i < frame.height(); i++) {
                    for (int j = 0; j < frame.width(); j++) {
                        double[] color = frame.get(i, j);
                        bg_modelR[i][j][k] = (int) color[RED];
                        bg_modelG[i][j][k] = (int) color[GREEN];
                        bg_modelB[i][j][k] = (int) color[BLUE];
                    }
                }
            }

            for (int i = 0; i < frame.height(); i++) {
                for (int j = 0; j < frame.width(); j++) {
                    Arrays.sort(bg_modelR[i][j]);
                    Arrays.sort(bg_modelG[i][j]);
                    Arrays.sort(bg_modelB[i][j]);
                }
            }
        } else {
            bg_model_grayscale = new int[480][640][bg_model_length];
            for (int k = 0; k < bg_model_length; k++) {
                if (is_video) {
                    frame = new Mat();
                    camera.read(frame);
                } else {
                    frame =
Tools.matify(Tools.getBitmapFromURL(image_path));
                }
                frame = Tools.toGrayscale(frame);

                for (int i = 0; i < frame.height(); i++) {
                    for (int j = 0; j < frame.width(); j++) {
                        bg_model_grayscale[i][j][k] = (int) frame.get(i,
j)[0];
                    }
                }

            }
            for (int i = 0; i < frame.height(); i++) {
                for (int j = 0; j < frame.width(); j++) {
                    Arrays.sort(bg_model_grayscale[i][j]);
                }
            }

        }
        VIDEO_WIDTH = frame.width();
        VIDEO_HEIGHT = frame.height();

    }

    public int[][] applyFilter(byte[] frame, int[][][] bg_model, double a,
double tt, int COLOR) {

        int[][] t1 = new int[VIDEO_HEIGHT][VIDEO_WIDTH];

        for (int i = 0; i < VIDEO_HEIGHT; i++) {
            for (int j = 0; j < VIDEO_WIDTH; j++) {
                double thresh = tt * Math.sqrt(bg_model[i][j][0]);
                if (Math.abs(((0x000000FF) & frame[i * VIDEO_WIDTH + j]) -
bg_model[i][j][0]) < thresh) {
                    t1[i][j] = 1;
                }
```

```
            }
        }

        t1 = Tools.convolones(t1);
        int[][] Fseg = new int[VIDEO_HEIGHT][VIDEO_WIDTH];
        for (int i = 0; i < VIDEO_HEIGHT; i++) {
            for (int j = 0; j < VIDEO_WIDTH; j++) {
                if (Tools.getDistance(kentro[1], kentro[0], i, j) > aktina)
{

                    Fseg[i][j] = 0;
                } else if (t1[i][j] > 5) {
                    int t = (int) ((1 - a) * bg_model[i][j][0] + a *
((0x000000FF) & frame[i * VIDEO_WIDTH + j]));
                    int k = 0;
                    while (k < bg_model[0][0].length - 1 && t <
bg_model[i][j][k]) {
                        k++;
                    }
                    bg_model[i][j][k] = t;
                    if (!is_grayscale) {
                        bg_rgb[i][j][COLOR] = t;
                    }
                    Fseg[i][j] = Tools.COLOR_BLACK;
                } else {
                    if (is_grayscale) {
                        Fseg[i][j] = Tools.COLOR_WHITE;
                    } else {
                        Fseg[i][j] = Tools.COLOR_BINARY_WHITE;
                        bg_rgb[i][j][COLOR] = bg_model[i][j][0];
                    }

                }
            }
        }

        return Fseg;
    }

      public BufferedImage run_seg_som() {
        if (is_grayscale) {
            // Wait for tri_segm to finish
            synchronized (TRISEGM_LOCK) {
                while (shouldRunTrisegm) {
                    try {
                        TRISEGM_LOCK.wait();
                    } catch (InterruptedException e) {
                        break;
                    }
                }
            }
        }
            if (is_video) {
                  camera.read(frame);
            } else {
                  frame = Tools.matify(Tools.getBitmapFromURL(image_path));
            }
            shouldRunTrisegm = true;
            synchronized (LOCK) {
                  LOCK.notify();
            }
```

```
found_segm = new int[frame.height()][frame.width()];
        if (!is_grayscale) {

            byte[] red_frame = new byte[VIDEO_HEIGHT * VIDEO_WIDTH];
            byte[] green_frame = new byte[VIDEO_HEIGHT * VIDEO_WIDTH];
            byte[] blue_frame = new byte[VIDEO_HEIGHT * VIDEO_WIDTH];

            List<Mat> to = new ArrayList<Mat>();
            Core.split(frame, to);
            to.get(RED).get(0, 0, red_frame);
            to.get(GREEN).get(0, 0, green_frame);
            to.get(BLUE).get(0, 0, blue_frame);

            int[][] outr = applyFilter(red_frame, bg_modelR, a_value,
tt_value, RED);
            int[][] outg = applyFilter(green_frame, bg_modelG, a_value,
tt_value, GREEN);
            int[][] outb = applyFilter(blue_frame, bg_modelB, a_value,
tt_value, BLUE);
            for (int i = 0; i < VIDEO_HEIGHT; i++) {
                for (int j = 0; j < VIDEO_WIDTH; j++) {

                    // (^1) xor with 1 to flip the values 0->1
                    if ((outr[i][j] ^ 1) + (outg[i][j] ^ 1) + (outb[i][j] ^
1) > 0) {
                        found_segm[i][j] = Tools.COLOR_BLACK;
                    } else {
                        found_segm[i][j] = Tools.COLOR_BINARY_WHITE;
                    }
                }
            }

            clearShadow();

        } else {

            Mat frame_grayscale = Tools.toGrayscale(frame);
            byte[] fr = new byte[VIDEO_HEIGHT * VIDEO_WIDTH];
            frame_grayscale.get(0, 0, fr);
            found_segm = applyFilter(fr, bg_model_grayscale, a_value,
tt_value, 0);

        }

        Mat fs = Tools.int2mat(found_segm, Tools.IMAGE_TYPE_GRAYSCALE);
        geom_proc(fs);

        BufferedImage i = Tools.toBufferedImage(fs);
        return i;
    }

    public class tri_segm implements Runnable {

        @Override
        public void run() {
            Mat BW1 = new Mat(VIDEO_HEIGHT, VIDEO_WIDTH,
Tools.IMAGE_TYPE_GRAYSCALE);
            while (true) {
                synchronized (LOCK) {
                    while (!shouldRunTrisegm) {
                        try {
                            LOCK.wait();
```

```
                        } catch (InterruptedException e) {
                            break;
                        }
                    }
                }
                for (int i = 0; i < VIDEO_HEIGHT; i++) {
                    for (int j = 0; j < VIDEO_WIDTH; j++) {
                        double[] t = frame.get(i, j);
                        if (t[RED] > 1.2 * t[GREEN] && t[RED] > 1.2 *
t[BLUE] && t[RED] > 120 && pixels_allowed[i][j] > 0) {
                            BW1.put(i, j, 1);
                        } else {
                            BW1.put(i, j, 0);
                        }
                    }
                }

                Mat mask = new Mat().ones(3, 3, CvType.CV_32F);
                Imgproc.filter2D(BW1, BW1, -1, mask);

                for (int i = 0; i < VIDEO_HEIGHT; i++) {
                    for (int j = 0; j < VIDEO_WIDTH; j++) {
                        if (BW1.get(i, j)[0] > 5) {
                            BW1.put(i, j, 255);
                        } else {
                            BW1.put(i, j, 0);
                        }
                    }
                }

                List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
                Imgproc.findContours(BW1, contours, new Mat(),
Imgproc.RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);
                Tools.removeSmallContours(contours, 5);

                if (contours.size() > 0) {
                    Point centroid = Tools.getCentroid(contours.get(0));
                    int min_pos = 0;
                    double min_value = Tools.getDistance(centroid.x,
centroid.y, last_robot_i, last_robot_j);
                    for (int i = 1; i < contours.size(); i++) {
                        centroid = Tools.getCentroid(contours.get(i));
                        if (Tools.getDistance(centroid.x, centroid.y,
last_robot_i, last_robot_j) < min_value) {
                            min_pos = i;
                        }
                    }
                    centroid = Tools.getCentroid(contours.get(min_pos));
                    last_robot_i = (int) centroid.x;
                    last_robot_j = (int) centroid.y;

                }
                shouldRunTrisegm = false;
                if (is_grayscale) {
                    synchronized (TRISEGM_LOCK) {
                        TRISEGM_LOCK.notify();
                    }
                }
            }
        }

    }
```

```
public BufferedImage getFirstFrame() {
        Mat frame = new Mat();
        if (is_video) {
            camera.read(frame);
        } else {
            frame = Tools.matify(Tools.getBitmapFromURL(image_path));
        }
        return Tools.toBufferedImage(frame);
    }

    private class prakseis implements Runnable {

        int j_start;
        int j_end;

        double alpha = 0.7;
        double beta = 0.85;
        double Ts = 255 * 0.5;
        double Th = 255 * 0.8;

        public prakseis(int j_start, int j_end) {
            this.j_start = j_start;
            this.j_end = j_end;
        }

        @Override
        public void run() {
            while (true) {
                synchronized (SHADOW_LOCK) {
                    while (!shouldRunShadow) {
                        try {
                            SHADOW_LOCK.wait();
                        } catch (InterruptedException e) {
                            // treat interrupt as exit request
                            break;
                        }
                    }
                }
                for (int i = 0; i < VIDEO_HEIGHT; i++) {
                    for (int j = j_start; j < j_end; j++) {
                        double ratio = F_hsv.get(i, j)[RED] /
BG_hsv[i][j][RED];
                        double diff1 = Math.abs(F_hsv.get(i, j)[GREEN] -
BG_hsv[i][j][GREEN]);
                        double diff2 = Math.min(Math.abs(F_hsv.get(i,
j)[BLUE] - BG_hsv[i][j][BLUE]), 360 - Math.abs(F_hsv.get(i, j)[BLUE] -
BG_hsv[i][j][BLUE]));

                        if (ratio >= alpha && ratio <= beta && diff1 <= Ts
&& diff2 <= Th) {
                            found_segm[i][j] = 0;
                        }
                    }
                }
                shouldRunShadow = false;
                latch.countDown();
            }
        }

    }
```

```
public void clearShadow() {
        F_hsv = Tools.toHSV(frame);
        BG_hsv = new float[VIDEO_HEIGHT][VIDEO_WIDTH][3];
        for (int i = 0; i < frame.height(); i++) {
            for (int j = 0; j < frame.width(); j++) {
                float[] hsv = new float[3];
                Color.RGBtoHSB(bg_rgb[i][j][RED], bg_rgb[i][j][GREEN],
bg_rgb[i][j][BLUE], hsv);
                hsv[0] = hsv[0] * 255;
                hsv[1] = hsv[1] * 255;
                hsv[2] = hsv[2] * 255;
                BG_hsv[i][j] = hsv;
            }
        }

        latch = new CountDownLatch(2);
        shouldRunShadow = true;
        synchronized (SHADOW_LOCK) {
            SHADOW_LOCK.notifyAll();
        }

        try {
            latch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        found_segm = Tools.convolones(found_segm);

        for (int i = 0; i < frame.height(); i++) {
            for (int j = 0; j < frame.width(); j++) {
                if (found_segm[i][j] > 5) {
                    found_segm[i][j] = Tools.COLOR_WHITE;
                } else {
                    found_segm[i][j] = Tools.COLOR_BLACK;
                }
            }
        }

    }

    private void geom_proc(Mat Fseg) {
        count++;
        List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
        List<MatOfPoint> contours2 = new ArrayList<MatOfPoint>();

        Imgproc.findContours(Fseg, contours, new Mat(), Imgproc.RETR_LIST,
Imgproc.CHAIN_APPROX_SIMPLE);
        Tools.removeSmallContours(contours, 10);
```

```
if (contours.size() > 0) {
            if (is_robot) {
                int idx = 0;
                Point centroid = Tools.getCentroid(contours.get(0));
                double value = Tools.getDistance(centroid.x, centroid.y,
last_robot_i, last_robot_j);
                for (int i = 0; i < contours.size(); i++) {
                    centroid = Tools.getCentroid(contours.get(i));
                    if (Tools.getDistance(centroid.x, centroid.y,
last_robot_i, last_robot_j) < value) {
                        // find the closest contour to last robot i,j
                        idx = i;
                    }
                }
                centroid = Tools.getCentroid(contours.get(idx));
                if (Tools.getDistance(centroid.x, centroid.y, last_robot_i,
last_robot_j) < 50) {
                    // only if found robot i,j is les than 50 pixels away
delete it and update robot i,j

                    contours.remove(idx);
                }
            }
            if (contours.size() > 0) {
                contours2.addAll(contours);

                ArrayList<indexes> itex = new ArrayList<indexes>();
                for (int i = 0; i < contours.size(); i++) {
                    Point human = Tools.getCentroid(contours.get(i));
                    double dist = Tools.getDistance(human.x, human.y,
last_human_i, last_human_j);
                    Point[] points = contours.get(i).toArray();
                    double mean = getMean(points);
                    itex.add(new indexes(dist, mean, i,
contours.get(i).total()));
                }
                Collections.sort(itex);

                int pos = 0;
                int it_itex = 0;
                if (isfirst) {
                    mean1 = itex.get(0).mean;
                    isfirst = false;
                    pos = itex.get(0).pos;
                } else {
                    double diff = Math.abs(mean1 - itex.get(0).mean);
                    Point human = Tools.getCentroid(contours.get(0));
                    double distance = Tools.getDistance(human.x, human.y,
last_human_i, last_human_j);
                    double rank = 0.5 * distance + 0.3 * diff - 0.2 *
itex.get(0).size;
                    for (int i = 0; i < itex.size(); i++) {
                        diff = Math.abs(mean1 - itex.get(i).mean);
                        human =
Tools.getCentroid(contours.get(itex.get(i).pos));
                        distance = Tools.getDistance(human.x, human.y,
last_human_i, last_human_j);
                        double trank = 0.5 * distance + 0.3 * diff - 0.2 *
itex.get(i).size;
```

```
 if (trank < rank) {

                            rank = trank;
                            pos = itex.get(i).pos;
                            it_itex = i;
                        }
                    }
                }

                // merge ta 40 kontinotera
                List<MatOfPoint> figure = new ArrayList<MatOfPoint>();
                List<Point> c = new ArrayList<Point>();
                Point human_recognized =
Tools.getCentroid(contours.get(pos));
                for (int i = 0; i < contours.size(); i++) {
                    Point human = Tools.getCentroid(contours.get(i));
                    double cur_val = Tools.getDistance(human.x, human.y,
human_recognized.x, human_recognized.y);
                    if (cur_val < 40) {
                        figure.add(contours.get(i));
                        c.addAll(contours.get(i).toList());
                    }
                }
                if (two_people) {
                    contours2.removeAll(figure);
                    // FIND 2nd human
                    ArrayList<indexes> itex2 = new ArrayList<indexes>();
                    for (int i = 0; i < contours2.size(); i++) {
                        Point human = Tools.getCentroid(contours2.get(i));
                        double dist = Tools.getDistance(human.x, human.y,
sec_human_i, sec_human_j);
                        Point[] points = contours2.get(i).toArray();
                        double mean = getMean(points);
                        itex2.add(new indexes(dist, mean, i,
contours2.get(i).total()));
                    }
                    Collections.sort(itex2);
                    int pos2 = 0;
                    int it_itex2 = 0;
                    if (itex2.size() > 0) {
                        if (isfirst2) {
                            mean2 = itex2.get(0).mean;
                            isfirst2 = false;
                            pos2 = itex2.get(0).pos;
                        } else {

                            double diff = Math.abs(mean2 -
itex2.get(0).mean);
                            Point human =
Tools.getCentroid(contours2.get(0));
                            double distance = Tools.getDistance(human.x,
human.y, sec_human_i, sec_human_j);
                            double rank = 0.5 * distance + 0.3 * diff - 0.2
* itex2.get(0).size;

                            for (int i = 0; i < itex2.size(); i++) {
                                diff = Math.abs(mean2 - itex2.get(i).mean);
                                human =
Tools.getCentroid(contours2.get(itex2.get(i).pos));
                                distance = Tools.getDistance(human.x,
human.y, sec_human_i, sec_human_j);
                                double trank = 0.5 * distance + 0.3 * diff -
0.2 * itex2.get(i).size;
```

```
 if (trank < rank) {

                                    rank = trank;
                                    pos2 = itex2.get(i).pos;
                                    it_itex2 = i;
                                }
                            }
                        }

                        // merge ta 40 kontinotera
                        List<MatOfPoint> figure2 = new
ArrayList<MatOfPoint>();
                        List<Point> c2 = new ArrayList<Point>();
                        Point human_recognized2 =
Tools.getCentroid(contours2.get(pos2));
                        for (int i = 0; i < contours2.size(); i++) {
                            Point human =
Tools.getCentroid(contours2.get(i));
                            double cur_val = Tools.getDistance(human.x,
human.y, human_recognized2.x, human_recognized2.y);
                            if (cur_val < 40) {
                                figure2.add(contours2.get(i));
                                c2.addAll(contours2.get(i).toList());
                            }
                        }

                        if (contours.get(pos).total() < 180) {
                            Point human =
Tools.getCentroid(contours2.get(pos2));
                            sec_human_i = (int) human.x;
                            sec_human_j = (int) human.y;

                            mean2 = itex2.get(it_itex2).mean;
                        }

                    }
                }
                if (contours.get(pos).total() < 180) {
                    MatOfPoint humanMerged = new MatOfPoint();
                    humanMerged.fromList(c);
                    double[] xy = geom_proc1_all_v2.geom_proc1_all_v2(Fseg,
THETA, PHI, Mc, humanMerged);
                    humancoords = Tools.getRobotCoords(xy[0], xy[1]);
                    con.setXY(xy[0], xy[1]);
                    Point human = Tools.getCentroid(humanMerged);
                    last_human_i = (int) human.x;
                    last_human_j = (int) human.y;
                    Imgproc.drawContours(Fseg, figure, -1, new Scalar(255),
-1);
                    mean1 = itex.get(it_itex).mean;
                }
            }

        }
```

```
public double getMean(Point[] points) {
        double mr = 0;
        double mg = 0;
        double mb = 0;
        for (int i = 0; i < points.length; i++) {
            int x = (int) points[i].x;
            int y = (int) points[i].y;
            mr += frame.get(y, x)[RED];
            mg += frame.get(y, x)[GREEN];
            mg += frame.get(y, x)[BLUE];
        }
        mr = mr / points.length;
        mg = mg / points.length;
        mb = mb / points.length;
        return (mr + mg + mb) / 3;
    }

    public double[] getHumanCoords() {
        return humancoords;
    }

    public void setPersonCoords(int i, int j) {
        last_human_i = i;
        last_human_j = j;
    }

    public void setSecPersonCoords(int i, int j) {
        sec_human_i = i;
        sec_human_j = j;
    }

    public void setRobotCoords(int i, int j) {
        last_robot_i = i;
        last_robot_j = j;
    }

    public int getVideoWidth() {
        return VIDEO_WIDTH;
    }

    public int getVideoHeight() {
        return VIDEO_HEIGHT;
    }

}
```

seg_som_my.java

```
package vid_seg;

import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.imgproc.Imgproc;

import sun.misc.BASE64Encoder;

public class Tools {

    public static double[] kentro = new double[] { 334.5957, 243.5654 };
    public static double aktina = 271.3681;
    public static int COLOR_WHITE = 255;
    public static int COLOR_BLACK = 0;
    public static int COLOR_BINARY_WHITE = 1;
    public static int IMAGE_TYPE_GRAYSCALE = CvType.CV_8U;

    /**
     * @param m
     *            a Mat array
     * @return Transforms a Mat array to BufferedImage
     */
    public static BufferedImage toBufferedImage(Mat m) {
        int type = BufferedImage.TYPE_BYTE_GRAY;
        if (m.channels() > 1) {
            type = BufferedImage.TYPE_3BYTE_BGR;
        }
        int bufferSize = m.channels() * m.cols() * m.rows();
        byte[] b = new byte[bufferSize];
        m.get(0, 0, b); // get all the pixels

        BufferedImage image = new BufferedImage(m.cols(), m.rows(), type);
        final byte[] targetPixels = ((DataBufferByte)
image.getRaster().getDataBuffer()).getData();
        System.arraycopy(b, 0, targetPixels, 0, b.length);

        return image;

    }
/**
     * @param m
     *            an integer array
     * @return Transforms an integer array to BufferedImage
     */
    public static BufferedImage toBufferedImage(int[][] m) {
        BufferedImage image = new BufferedImage(m[0].length, m.length,
BufferedImage.TYPE_BYTE_BINARY);
        int[] newArray = new int[m.length * m[0].length];
        for (int n = 0; n < m.length; n++) {
            System.arraycopy(m[n], 0, newArray, n * m[0].length,
m[0].length);
        }
        WritableRaster raster2 = image.getRaster();
        raster2.setPixels(0, 0, m[0].length, m.length, newArray);

        image.setData(raster2);
        return image;
    }
```

```
/**
     * @param b
     *            the BGR array
     * @return Transforms BGR to Grayscale array
     */
    public static Mat toGrayscale(Mat b) {
        Mat temp = new Mat(b.height(), b.width(), CvType.CV_8UC1);
        Imgproc.cvtColor(b, temp, Imgproc.COLOR_BGR2GRAY);
        return temp;
    }

    /**
     * @param b
     *            the BGR array
     * @return Transforms BGR to HSV array
     */
    public static Mat toHSV(Mat b) {
        Mat temp = new Mat(b.height(), b.width(), b.type());
        Imgproc.cvtColor(b, temp, Imgproc.COLOR_BGR2HSV);
        return temp;
    }

    /**
     * @param x
     * @param y
     * @return Transforms Cartesian x,y to Robot x,y
     */
    public static double[] getRobotCoords(double x, double y) {
        double y_new = 1000 * x + (26826 + 1900);
        double x_new = -1000 * y + (28702 - 4290);
        return new double[] { x_new, y_new };
    }

    /**
     * @param link
     *            : The Image link
     * @return Downloads and return the image from the given link
     */
    public static BufferedImage getBitmapFromURL(String link) {
        try {
            String credentials = "admin" + ":" + "meinsm";
            BASE64Encoder enc = new sun.misc.BASE64Encoder();
            String encoding = enc.encode(credentials.getBytes("UTF-8"));
            URL url = new URL(link);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setDoInput(true);
            connection.setRequestProperty("Authorization",
String.format("Basic %s", encoding));
            connection.connect();
            InputStream input = connection.getInputStream();
            BufferedImage myBitmap = new BufferedImage(640, 480,
BufferedImage.TYPE_3BYTE_BGR);
            myBitmap = ImageIO.read(input);
            return myBitmap;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

```
/**
     * @param im
     *            : the Image to transform
     * @return Transforms a BufferedImage to Mat array
     */
    public static Mat matify(BufferedImage im) {
        // Convert bufferedimage to byte array
        byte[] pixels = ((DataBufferByte)
im.getRaster().getDataBuffer()).getData();
        // Create a Matrix the same size of image
        Mat image = new Mat(im.getHeight(), im.getWidth(), CvType.CV_8UC3);
        // Fill Matrix with image values
        image.put(0, 0, pixels);
        return image;

    }

    /**
     * @param m
     *            : the array to transform
     * @return Transforms a Mat array to integer array
     */
    public static int[][] mat2int(Mat m) {
        int[][] frame = new int[m.height()][m.width()];
        for (int i = 0; i < m.height(); i++) {
            for (int j = 0; j < m.width(); j++) {
                frame[i][j] = (int) m.get(i, j)[0];
            }
        }
        return frame;
    }

    /**
     * @param m
     *            : the array to transform
     * @param IMG_TYPE
     *            : Result Image type
     * @return Transforms a double array to Mat array
     */
    public static Mat double2mat(double[][] m, int IMG_TYPE) {
        Mat frame = new Mat(m.length, m[0].length, IMG_TYPE);
        for (int i = 0; i < m.length; i++) {
            frame.put(i, 0, m[i]);
        }
        return frame;
    }

    /**
     * @param m
     *            : the array to transform
     * @param IMG_TYPE
     *            : Result Image type
     * @return Transforms an integer array to Mat array
     */
    public static Mat int2mat(int[][] m, int IMG_TYPE) {

        Mat frame = new Mat(m.length, m[0].length, IMG_TYPE);

        for (int i = 0; i < m.length; i++) {
```

```
            for (int j = 0; j < m[0].length; j++) {
                frame.put(i, j, m[i][j]);
            }
        }

        return frame;
    }

    /**
     * @param marks
     * @return The average for the given list of CustomPoints
     */
    public static double calculateAverage(List<CustomPoints> marks) {
        Double sum = 0.0;
        if (!marks.isEmpty()) {
            for (int i = 0; i < marks.size(); i++) {
                sum += marks.get(i).getValue();
            }
            return sum / marks.size();
        }
        return sum;
    }

    /**
     * @param t
     * @param f
     * @param r
     * @return The Cartesian coordinates for the give values
     */
    public static Point sph2cart(double t, double f, double r) {
        double X = r * Math.sin(t) * Math.cos(f);
        double Y = r * Math.cos(f) * Math.cos(t);
        double Z = r * Math.sin(f);
        return new Point(X, Y);
    }

    /**
     * @param contour
     * @return The centroid of given contour
     */
    public static Point getCentroid(MatOfPoint contour) {
        List<Point> points = contour.toList();
        double x = 0;
        double y = 0;

        for (int i = 0; i < points.size(); i++) {
            x += points.get(i).x;
            y += points.get(i).y;
        }

        return new Point(x / points.size(), y / points.size());
    }
```

```
public static double[][] getPixels_allowed_robot(double[][] THETA, double[][]
PHI, double xgon, double ygon, double zmax, double aktina, double hh) {
        final double[] kentro = new double[] { 334.5957, 243.5654 };
        double[][] pixels_allowed = new
double[THETA.length][THETA[0].length];
        for (int i = 0; i < THETA.length; i++) {
            for (int j = 0; j < THETA[0].length; j++) {
                if (Math.pow(i - kentro[0], 2) + Math.pow(i - kentro[0], 2)
<= Math.pow(aktina, 2)) {
                    double t = THETA[i][j];
                    double p = PHI[i][j];
                    double z0 = zmax - hh;
                    double r = z0 / Math.tan(p);
                    double x0 = r * Math.cos(t);
                    double y0 = r * Math.sin(t);

                    boolean synthiki1 = (x0 > xgon) && (x0 < 14) && (y0 >=
ygon) && (y0 < (ygon + 4.158));
                    boolean synthiki2 = (x0 > xgon) && (x0 <= (ygon + 2.044))
&& (y0 >= ygon) && (y0 < 8);
                    if (synthiki1 || synthiki2) {
                        pixels_allowed[i][j] = 1;
                    } else {
                        pixels_allowed[i][j] = 0;
                    }
                }
            }
        }

        return pixels_allowed;
    }

    /**
     * @param x
     * @param y
     * @param i
     * @param j
     * @return returns the Math.abs(x - i) + Math.abs(y - j)
     */
    public static double getDistance(double x, double y, double i, double j)
{
        return Math.abs(x - i) + Math.abs(y - j);
    }

    /**
     * @param x
     * @param y
     * @return Math.sqrt(Math.pow(x - kentro[0], 2) + Math.pow(y - kentro[1],
2))
     */
    public static double getEucDistanceFromCenter(double x, double y) {
        return getEucDistance(x, y, kentro[0], kentro[1]);

    }

    public static double getEucDistance(double x, double y, double i, double
j) {
        return Math.sqrt(Math.pow(x - i, 2) + Math.pow(y - j, 2));
    }
```

```
/**
     * @param A
     *            : Array to do convolution
     * @return The convolution with a mask ones(3,3)
     */
    public static int[][] convolones(int[][] A) {
        int[][] B = new int[A.length][A[0].length];
        for (int i = 1; i < A.length - 1; i++) {
            for (int j = 1; j < A[0].length - 1; j++) {
                B[i][j] = A[i - 1][j - 1] + A[i - 1][j] + A[i - 1][j + 1] +
A[i][j - 1] + A[i][j] + A[i][j + 1] + A[i + 1][j - 1] + A[i][j] + A[i][j +
1];
            }
        }
        return B;
    }

    public static void removeSmallContours(List<MatOfPoint> contours, long
value) {
        List<MatOfPoint> contours_todel = new ArrayList<MatOfPoint>();
        for (int i = 0; i < contours.size(); i++) {
            if (contours.get(i).total() < value) {
                contours_todel.add(contours.get(i));
            }
        }
        contours.removeAll(contours_todel);
    }

}
```

Tools.java

```
package vid_seg;

import java.awt.image.BufferedImage;

public class Video {

    public enum Functions {
        FUNCTION_SEG_DERIV, FUNCTION_SEG_SOM
    };

    private final boolean is_video = true;
    private String video_path = "path_to_video_file.avi"
    private final String image_path = "camera_ip";

    private final Functions function;
    private seg_deriv s;
    private seg_som_my sgm;

    public Video(Functions function) {
        if (function == Functions.FUNCTION_SEG_DERIV) {
            s = new seg_deriv(is_video ? video_path : image_path, is_video);
        } else if (function == Functions.FUNCTION_SEG_SOM) {
            sgm = new seg_som_my(is_video ? video_path : image_path,
is_video);
        }
        this.function = function;
    }
```

```
    public Video(Functions function, String path) {
        this(function);
        this.video_path = path;

    }

    public BufferedImage run() {
        if (function == Functions.FUNCTION_SEG_DERIV) {
            return s.run_seg_deriv();
        } else {
            return sgm.run_seg_som();
        }
    }

    public void setHumanCoords(int i, int j) {
        sgm.setPersonCoords(i, j);
    }

    public void setRobotCoords(int i, int j) {
        sgm.setRobotCoords(i, j);
    }

    public void setSecHumanCoords(int i, int j) {
        sgm.setSecPersonCoords(i, j);
    }

    public BufferedImage getFirstFrame() {
        return sgm.getFirstFrame();
    }

    public int getVideoWidth() {
        if (function == Functions.FUNCTION_SEG_DERIV) {
            return s.getVideoWidth();
        } else {
            return sgm.getVideoWidth();
        }
    }

    public int getVideoHeight() {
        if (function == Functions.FUNCTION_SEG_DERIV) {
            return s.getVideoHeight();
        } else {
            return sgm.getVideoHeight();
        }
    }

    public double[] getHumanCoords() {
        return sgm.getHumanCoords();
    }

}
```

Video.java

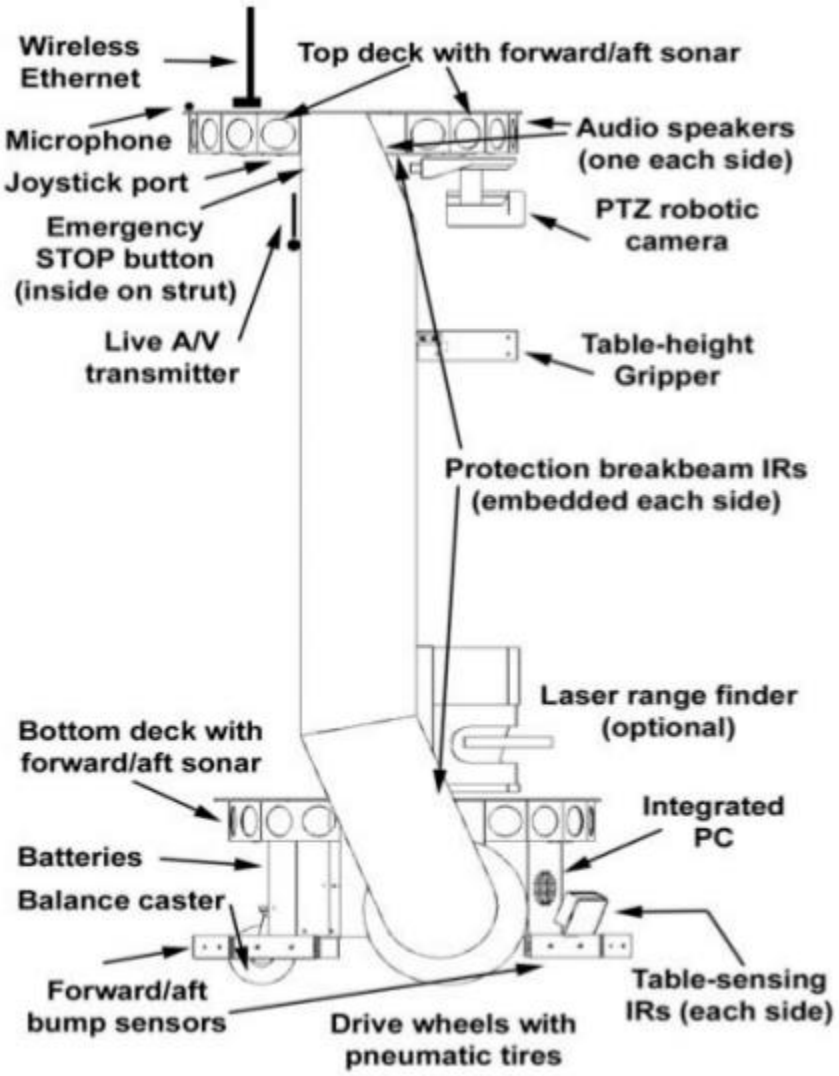
```
package vid_seg;

public class main {

    public static void main(String[] args) {
        graphics g = new graphics();
    }

}
```

main.java

Β’ Χαρακτηριστικά του PeopleBot

Φυσικά χαρακτηριστικά	
Μήκος (cm)	47
Πλάτος (cm)	38
Ύψος (cm)	124
Βάρος (kg)	21
Φορτίο (kg)	11
Ισχύς	
Μπαταρίες 12VDC	3
Φόρτιση (watt ανά ώρα)	252
Χρόνος λειτουργίας (ώρες)	8 - 10
με Η/Υ (ώρες)	3 - 4
Χρόνος φόρτισης ώρες/μπατ.	6
Κινητικότητα	
Τροχοί	2 με αφρό
Πέλμα	τρακτεροτό
Διάμετρος (mm)	195.3
Πλάτος (mm)	50
Οδήγηση	διαφορική
Λόγος ταχύτητας	38.3:1
Μέγιστη ταχύτητα (mm/sec)	900
Μέγιστη ταχύτητα περιστροφική (deg/sec)	150
Διασχίσιμο κενό (mm)	50
Διασχίσιμη κλίση (ποσοστό)	11%
Επιτρεπτό έδαφος	ανάλογο με καροτσάκι

Πίνακας 1: Τα φυσικά χαρακτηριστικά του PeopleBot



Σχήμα 1: Τα κομμάτια του PeopleBot