Ατομική Διπλωματική Εργασία

ΠΡΟΣΟΜΟΙΩΣΗ ΑΛΓΟΡΙΘΜΟΥ ΣΥΓΚΕΝΤΡΩΣΗΣ ΑΥΤΟΝΟΜΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΟΝΤΟΤΗΤΩΝ ΣΤΟ ΕΠΙΠΕΔΟ

Νάταλη Τεμενέ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Προσομοίωση Αλγορίθμου Συγκέντρωσης Αυτόνομων Υπολογιστικών Οντοτήτων στο Επίπεδο

Νάταλη Τεμενέ

Επιβλέπων Καθηγητής
Χρύσης Γεωργίου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2013

Ευχαριστίες

Κατά αρχήν, θα ήθελα να ευχαριστήσω τον επιβλέπον καθηγητή κ. Χρύση Γεωργίου, για την άψογη συνεργασία που είχαμε κατά την διάρκεια αυτών των δύο εξαμήνων για την εκπόνηση αυτής της εργασίας.

Ένα μεγάλο ευχαριστώ οφείλω και στον Χρυσοβαλάντη Αγαθαγγέλου για την προθυμία και την πολύτιμη βοήθεια που μου πρόσφερε οποιαδήποτε στιγμή την χρειαζόμουν.

Ακόμη ένα μεγάλο ευχαριστώ οφείλω στον φίλο και συμφοιτητή Μάριο Μιχαήλ για την πολύτιμη βοήθεια του στην εγκατάσταση και κατανόηση κάποιων εννοιών για την βιβλιοθήκη pthreads.

Τελειώνοντας, θα ήθελα να ευχαριστήσω τους φίλους/συμφοιτητές και την οικογένεια μου για την στήριξη κατά την διάρκεια της εκπόνησης αυτής της εργασίας.

Περίληψη

Στα πλαίσια της ΑΔΕ μου κλήθηκα να υλοποιήσω τον αλγόριθμο για τη συσσώρευση ρομπότ σχήματος δίσκων στο επίπεδο, ο οποίος συντάχθηκε σε προηγούμενη εργασία. Ο αλγόριθμος αυτός επιλύει το πρόβλημα συγκέντρωσης ρομπότ στο επίπεδο, όπου μια συλλογή από n αυτόνομα ρομπότ από μια αρχική κατάσταση θα καταλήξουν σε μια τελική, η οποία θα έχει την μορφή κυρτού περιβλήματος, ώστε όλοι να βλέπουν όλους, και νε είναι συνδεδεμένα μεταξύ τους. Τα ρομπότ βρίσκονται αρχικά σκορπισμένα στο επίπεδο και με βάση τον αλγόριθμο συγκέντρωσης καταφέρνουν να βρουν μια θέση στο κυρτό περίβλημα με πλήρη ορατότητα σε σχέση με τα άλλα ρομπότ. Όταν ολοκληρωθεί αυτή η φάση, τότε τα ρομπότ εκτελούν την δεύτερη φάση, στην οποία πρέπει να αρχίσουν να συγκεντρώνονται με σκοπό να κοντέψουν όσο το δυνατό πιο κοντά γίνεται, χωρίς να χαλάσουν τα δυο χαρακτηριστικά που υλοποιήθηκαν στην πρώτη φάση.

Με την χρήση την γλώσσας προγραμματισμού C υλοποίησα τον αλγόριθμο αυτό σε περιβάλλον unix. Για τα γραφικά του προγράμματος έκανα χρήση της βιβλιοθήκης GP142 και για να είναι κάθε ρομπότ αυτόνομο και να κάνει τους υπολογισμούς του χωρίς να εξαρτάται από τα άλλα ρομπότ χρησιμοποίησα την βιβλιοθήκη pthreads, για να εισαχθεί ο παραλληλισμός στο πρόγραμμα και να υπάρξει μια μορφή ασυγχρονισμού.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο Διπλωματικής Εργασίας

Τα ρομπότ σήμερα αποτελούν ένα βασικό αντικείμενο έρευνας σε πολλά πεδία, όπως τεχνητή νοημοσύνη, νανοτεχνολογία, στρατός, και πολλά άλλα. Το τόσο μεγάλο ενδιαφέρον για το συγκεκριμένο αντικείμενο αποτελεί το γεγονός ότι με τα ρομπότ μπορεί να χρησιμεύσουν για δύσκολες και επικίνδυνες αποστολές που ο άνθρωπος να μην μπορεί να αντεπεξέλθει. Για τον λόγο αυτό τα ρομπότ πρέπει να είναι φτηνά και απλά με περιορισμένη υπολογιστική δύναμης, δηλαδή να μην κάνουν περίπλοκους υπολογισμούς.

Ένα θεμελιώδες ερευνητικό πρόβλημα είναι η συγκέντρωση, όπου αυτόνομες οντότητες πρέπει να συγκεντρωθούν σε συγκεκριμένο σημείο, περιοχή ή να δημιουργήσουν ένα συγκεκριμένο σχηματισμό στο επίπεδο. Αυτό το πρόβλημα έχει μελετηθεί αρκετά και έχουν υλοποιηθεί πολλοί θεωρητικοί αλγόριθμοι [2,3,4,5] με διαφορετική μοντελοποίηση. Όλοι οι αλγόριθμοι είχαν κάποια κοινά χαρακτηριστικά τα οποία περιελάμβαναν ότι: (α) το ρομπότ έχει όγκο, δηλαδή είναι αδιαφανές και μπορούν να εμποδίσει την θέα κάποιου άλλου ρομπότ, (β) έχει μια συσκευή όρασης,

δηλαδή μπορεί να δει γύρω του τι υπάρχει και (γ) εκτελεί τον κύκλο Βλέπω – Υπολογίζω – Μετακινούμαι, δηλαδή Βλέπει γύρω του, παίρνοντας μια εικόνα της σκηνής, με βάση αυτή την εικόνα εκτελεί κάποιους υπολογισμούς και παίρνει απόφαση αν θα μετακινηθεί σε ένα σημείο στο επίπεδο.

Ο αλγόριθμος, στον οποίο βασίστηκα, υλοποιεί το πρόβλημα συγκέντρωσης ρομπότ στο επίπεδο, προϋποθέτοντας ότι υπάρχουν n αυτόνομα ρομπότ, τα οποία δεν υπόκεινται σε σφάλματα, είναι ίδια μεταξύ τους - δηλαδή έχουν ίδια ακτίνα και μπορούν να κινηθούν σε ευθεία γραμμή στο επίπεδο. Το κλειδί του αλγορίθμου αυτού είναι ότι φέρνει τα ρομπότ σε μορφή πλήρης ορατότητας, δηλαδή όλοι βλέπουν όλους. Για να επιτευχθεί αυτό, όπου το γεγονός ότι τα ρομπότ είναι αδιαφανές το δυσκολεύει, απαιτείται από τα ρομπότ να σχηματίσουν ένα κυρτό περίβλημα, στο οποίο όλα τα ρομπότ θα είναι πάνω σ'αυτό. Κατά την διάρκεια των υπολογισμών, τα ρομπότ που βρίσκονται πάνω στο κυρτό περίβλημα δεν μετακινούνται και αυτά που βρίσκονται εντός του σχήματος βγαίνουν προς τα έξω ώστε να βρουν μια θέση πάνω στο σχήμα. Υπάρχουν δυο περιπτώσεις όπου μετακινούνται τα ρομπότ, τα οποία βρίσκονται πάνω στο κυρτό περίβλημα. Η πρώτη περίπτωση είναι όταν εμποδίζουν κάποιο ρομπότ να δει κάποιο άλλο πάνω στο κυρτό περίβλημα τότε μετακινούνται προς τα έξω ώστε να αποκαλύψει το κρυμμένο ρομπότ. Η δεύτερη περίπτωση είναι όταν αντιληφθεί ότι δεν έχει άλλο χώρο για άλλο ρομπότ πάνω στο κυρτό περίβλημα, για αυτά που βρίσκονται εντός του σχήματος, και μετακινηθεί προς τα έξω για να κάνει χώρο. Με βάση όλα αυτά, στο τέλος αυτών των υπολογισμών το κυρτό περίβλημα θα ανοίξει σε μεγάλο βαθμό, ώστε όλα τα ρομπότ να βρίσκονται πάνω και να έχουν πλήρη ορατότητα. Αυτό αποτελεί και την πρώτη φάση του αλγορίθμου αυτού. Ακολούθως, στην δεύτερη φάση αρχίζει να γίνεται η συγκέντρωση των ρομπότ, δηλαδή αρχίζουν να μπαίνουν προς τα μέσα, με μικρά βήματα κάθε φορά ώστε να μην χαλάσει το κυρτό περίβλημα και να μην χαθεί η πλήρης ορατότητα, ώστε να συνδεθούν και να τερματίσει ο αλγόριθμος.

Δεν έχουν υλοποιηθεί οι θεωρητικοί αλγόριθμοι ώστε να δούμε αν είναι εφικτοί πρακτικά να γίνουν και αν όντως επιλύουν το πρόβλημα της συγκέντρωσης των ρομπότ στο επίπεδο.

1.2 Σκοπός Διπλωματικής Εργασίας

Ο σκοπός αυτής της εργασίας ήταν η υλοποίηση του αλγορίθμου συγκέντρωσης αυτόνομων υπολογιστικών οντοτήτων στο επίπεδο[1,2], ο οποίος συντάχτηκε σε προηγούμενη εργασία. Η υλοποιήσει αυτή είχε σαν στόχο την απόδειξη ότι είναι εφικτά πρακτικά ο σκοπός του αλγορίθμου αυτού, δηλαδή ότι γίνεται η συγκέντρωση των ρομπότ.

Η υλοποίηση του αλγορίθμου έγινε με την γλώσσα προγραμματισμού C. Για την καλύτερη εμφάνιση του προγράμματος, έγινε χρήση της βιβλιοθήκης GP142 για το γραφικό περιβάλλον και για να είναι αυτόνομα τα ρομπότ έγινε χρήση της βιβλιοθήκης pthread, για να εισαχθεί ο παραλληλισμός και κάθε ρομπότ αποτελώντας ένα ξεχωριστό νήμα να εκτελεί μόνο του τους υπολογισμούς του.

1.3 Μεθοδολογία Υλοποίησης

Για την εκπόνηση αυτής της εργασίας ακολουθήθηκε η εξής μεθοδολογία. Αρχικά, μελέτη και κατανόηση του αλγορίθμου που μου δόθηκε[1,2], καθώς και η μελέτη και κατανόηση των εργαλείων που θα χρησιμοποιούσα στην εργασία αυτή, όπως η βιβλιοθήκη GP142 για τα γραφικά του προγράμματος και η βιβλιοθήκη pthread για την εκτέλεση κώδικα παράλληλα. Αφού μελετήθηκαν όλα αυτά άρχισε η συγγραφή του κώδικα για την υλοποίηση του συγκεκριμένου αλγορίθμου.

Αρχικά, υλοποιήθηκαν τα πέντε βασικά βήματα από τον κύκλο που εκτελεί το ρομπότ μέχρι να τερματίσει. Το βήμα υπολογίζω αποτελείται από 16 διαδικασίες, οι οποίες έπρεπε να μελετηθούν και να υλοποιηθούν. Για την υλοποίηση των διαδικασιών αυτών υπήρχε υλοποίηση άλλων γεωμετρικών συναρτήσεων, που αποτελούν βοηθητικές συναρτήσεις για τις συγκεκριμένες διαδικασίες. Επίσης, χρειάστηκε η υλοποίηση συναρτήσεων, βοηθητικές κυρίως για τις γεωμετρικές συναρτήσεις, οι οποίες είχαν σκοπό να κάνουν μαθηματικές πράξεις, όπως για παράδειγμα να βρει την εξίσωση της ευθείας από δύο σημεία, να βρει την απόσταση δυο σημείων και πολλά άλλα.

1.4 Δομή Διπλωματικής Εργασίας

Η δομή αυτής της εργασίας έχει την εξής μορφή. Στο Κεφάλαιο 2 δίδεται ένα υπόβαθρο σε σχέση με το πρόβλημα και τα εργαλεία που χρησιμοποιήθηκαν. Εδώ παρουσιάζεται σε γενικό επίπεδο το πρόβλημα που επιλύει ο αλγόριθμος που έχει υλοποιηθεί και μετά δίδεται μια επεξήγηση για τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση του αλγορίθμου, συγκεκριμένα για το γραφικό περιβάλλον με το οποίο εμπλουτίστηκε το πρόγραμμα και την εισαγωγή παραλληλισμού στην εκτέλεση των εντολών κώδικα στο πρόγραμμα. Στο Κεφάλαιο 3, γίνεται μια λεπτομερή περιγραφή για όλες τις συναρτήσεις και διαδικασίες που χρησιμοποιεί ο αλγόριθμος καθώς και λεπτομερή περιγραφή της υλοποίησης που έγινε για αυτές. Το Κεφάλαιο 4 παρουσιάζει στιγμιότυπα του προγράμματος να τρέχει και πώς φαίνεται η υλοποίηση του αλγορίθμου με τα γραφικά και τις κινήσεις των ρομπότ που γίνονται μέχρι να φτάσουν στην τελική τους κατάσταση. Παρουσιάζονται κάποια παραδείγματα εκτέλεσης του προγράμματος. Τέλος, στο Κεφάλαιο 5 εξάγουμε συμπέρασμα, αναφέρουμε τα προβλήματα που συναντήσαμε και πώς τα αντιμετωπίσαμε, καθώς και κάποιες μελλοντικές επεκτάσεις που μπορούν να γίνουν στο πρόγραμμα για την περαιτέρω βελτίωση του.

Κεφάλαιο 2

Υπόβαθρο

2.1 Ορισμός Προβλήματος

Ο αλγόριθμος προς υλοποίηση επιλύει το πρόβλημα συγκέντρωσης ρομπότ σχήματος δίσκων στο επίπεδο. Το πρόγραμμα παίρνει σαν είσοδο μια αρχική κατάσταση με n αυτόνομα ρομπότ σκορπισμένα στο επίπεδο και τερματίζει όταν αυτά σχηματίσουν ένα κυρτό περίβλημα, ώστε όλα τα ρομπότ να βλέπουν όλα τα άλλα – έχουν πλήρη ορατότητα και είναι συνδεδεμένα μεταξύ τους. Η εκτέλεση του προγράμματος, αρχικά θα αρχίσει να ανοίγουν τα ρομπότ στον χώρο σχηματίζοντας ένα κυρτό περίβλημα και όταν φτάσουν όλα να έχουν μια θέση πάνω στο σχήμα, τότε τα ρομπότ θα αρχίσουν με μικρά βήματα κάθε φορά να πηγαίνουν προς τα μέσα μέχρι να συνδεθούν, δηλαδή να εφάπτονται όλα μεταξύ τους.

Κάθε ρομπότ έχει τα ακόλουθα χαρακτηρίστηκα:

- περιέχει όγκο, δηλαδή είναι αδιαφανές - δεν μπορεί να περάσει πάνω από κάποιο ρομπότ παρά μόνο να συγκρουστεί μαζί του, με την έννοια ότι θα φτάσει σε σημείο που θα αγγίζει με το άλλο ρομπότ. Θεωρούμε ότι έχουν ακτίνα 1.

- δεν είναι επιρρεπής σε σφάλματα, δηλαδή δεν μπορούν να καταρρεύσουν κατά την διάρκεια της εκτέλεσης του προγράμματος.
- είναι αυτόνομο, δηλαδή δεν επηρεάζεται σε μεγάλο βαθμό από τις ενέργειες των άλλων ρομπότ.
- είναι ασύγχρονο σε σχέση με τα άλλα ρομπότ, δηλαδή δεν εκτελεί το ίδιο κομμάτι κώδικα αλλά βρίσκεται είτε στο ίδιο είτε σε διαφορετικό κομμάτι.
- έχουν συσκευή όρασης, δηλαδή ένα είδος κάμερας με την οποία μπορούν να δουν τι περιέχει γύρω τους και κυρίως ποια ρομπότ βρίσκονται στο οπτικό τους πεδίο. Η κάμερα αυτή έχει 360 μοίρες στροφή, άρα μπορεί να δει τι έχει μπροστά του ακόμη και τι έχει πίσω του.
- εκτελεί τον κύκλο Βλέπω – Υπολογίζω – Μετακινούμαι.

Κατά την διάρκεια εκτέλεσης του προγράμματος κάθε ρομπότ εκτελεί ένα κύκλο πέντε φάσεων, με σκοπό να φτάσει στην τερματική διαδικασία και να μπορέσει να τερματίσει. Οι πέντε φάσεις είναι οι ακόλουθες:

- Περιμένω: εδώ το ρομπότ κάνει μια μικρή παύση και παραμένει σε αδράνεια για ένα μικρό χρονικό διάστημα.
- Βλέπω: το ρομπότ ελέγχει ποια από τα άλλα ρομπότ βρίσκονται στο οπτικό του πεδίο.
- Υπολογίζω: εδώ εκτελείται ο αλγόριθμος συγκέντρωσης ρομπότ στο επίπεδο και παραμένει εδώ μέχρι να φτάσει σε μια τερματική διαδικασία που θα το μεταφέρει σε μια από τις δύο επόμενες φάσεις.
- Μετακινούμαι: εδώ το ρομπότ μετακινείται στην νέα θέση που υπολογίστηκε από την προηγούμενη φάση, ή παραμένει στην ίδια θέση αν αυτό θεώρησε ότι πρέπει να γίνει η προηγούμενη φάση.
- Τερματίζω: όταν το ρομπότ φτάσει εδώ, τότε αυτό σημαίνει ότι πληρεί όλα τα κριτήρια τερματισμού (βρίσκεται πάνω στο κυρτό περίβλημα, έχει πλήρη ορατότητα και είναι συνδεδεμένα όλα μεταξύ τους) και θα τερματίσει τον κύκλο, δηλαδή δεν θα εκτελέσει άλλους υπολογισμούς.

Με τον τερματισμό όλων των ρομπότ θα έχουμε όλα τα ρομπότ συγκεντρωμένα σε ένα συγκεκριμένο σημείο στην μορφή ενός κυρτού περιβλήματος.

2.2 Περιγραφή Εργαλείων

Για την υλοποίηση του αλγορίθμου αυτού χρησιμοποίησα δυο βασικά εργαλεία, το ένα για το γραφικό περιβάλλον και τον παραλληλισμό του προγράμματος. Τα γραφικά υλοποιήθηκαν με την βιβλιοθήκη GP142 [6] και ο παραλληλισμός με την χρήση της βιβλιοθήκης Pthread [7,8].

2.2.1 Γραφικά Προγράμματος

Τα γραφικά του προγράμματος έγιναν με την χρήση της βιβλιοθήκης GP142, η οποία κατά την έρευνα που έκανα για να βρω ποια βιβλιοθήκη να χρησιμοποιήσω αποδείχθηκε η πιο κατάλληλη για τον λόγο ότι είναι εύκολη στην κατανόηση και στην χρήση, αλλά και επαρκής για τον σκοπό που την ήθελα. Ένας ακόμη παράγοντας για την επιλογή αυτής της βιβλιοθήκης αποτέλεσε και το γεγονός ότι την έχω ξαναχρησιμοποιήσει στο μάθημα ΕΠΛ131 – Αρχές Προγραμματισμού Ι.

Για την χρήση αυτής της βιβλιοθήκης χρειάστηκε να προσθέσω τρία αρχεία, δύο βιβλιοθήκες «gp142.h» και «gp142lib.h» και ένα αρχείο «gp142.c». Στην κεντρική συνάρτηση απλά συμπεριέλαβα την βιβλιοθήκη «gp142.h» για να αναγνωρίζει τις συναρτήσεις που περιλαμβάνονται στο πρόγραμμα. Αρχικά, για να ξεκινήσει το παράθυρο των γραφικών χρειάζεται η συνάρτηση «GP142_open», η οποία πρέπει να μπει πάνω στον κώδικα και για να κλείσει το παράθυρο και να τερματιστεί η διαδικασία των γραφικών πρέπει στο τέλος να μπει η συνάρτηση «GP142_close()»,η οποία μπαίνει σαν τελευταία εντολή στον κώδικα. Για τον σχεδιασμό του μεγέθους του παραθύρου στο οποίο θα καθορίζονται οι παράμετροι του χρώματος του φόντου, τις συντεταγμένες του παράθυρου που θα ανοίξει και το πλάτο ς της γραμμής, χρησιμοποιήθηκε η συνάρτηση «GP142_rectangleXY(int color,int x1, int y1,int x2, int y2,int thickness)», η οποία σχεδιάζει ορθογώνια. Για τον σχεδιασμό των ρομπότ χρησιμοποιήθηκε η συνάρτηση «GP142_circleXY(int color,int x, int y,int radius)», η οποία σχεδιάζει κύκλους με παραμέτρους το χρώμα, τις συνιστώσες και την ακτίνα. Επειδή κάθε δευτερόλεπτο υπάρχει σχεδόν και κάτι καινούργιο να σχεδιάσει στην οθόνη και η βιβλιοθήκη αυτή δεν υποστηρίζει στιγμιαία επανασχεδίαση, για τον λόγο αυτό αν συνέβαινε και σχεδίαζε στιγμιαία τότε η οθόνη θα έτρεμε και όλο αυτό θα φαινόταν

άσχημα. Για όλους αυτούς τους λόγους χρειάστηκε να προ θέσω την εντολή «GP142_flush()», η οποία αναγκάζει την οθόνη να επανασχεδιαστεί μόνο σε σημεία που είναι αναγκαίο.

2.2.2 Παραλληλισμός Προγράμματος

Για να μπορεί κάθε ρομπότ να τρέχει ανεξάρτητα από τα άλλα και όχι σειριακά το κομμάτι του κώδικα που του αναλογεί, χρειάστηκε να χρησιμοποιήσω παραλληλισμό ώστε κάθε ρομπότ να τρέχει παράλληλα τον κώδικα του για να μην έχουμε σειριακή εκτέλεση και ούτε σύγχρονο μοντέλο. Για την χρήση παράλληλης εκτέλεσης χρησιμοποίησα την βιβλιοθήκη Pthread. Για να μπορώ να χρησιμοποιήσω τους τύπους και τις συναρτήσεις της βιβλιοθήκης αυτής έπρεπε να συμπεριλάβω την βιβλιοθήκη «pthread.h». Για να έχει κάθε ρομπότ το δικό του thread δημιούργησα ένα πίνακα τύπου «pthread_t» με μέγεθος ίσο με το μέγεθος της σταθεράς που ορίζει τον αριθμό n των ρομπότ του προγράμματος.

Κάθε thread δημιουργείται όταν καλείται η συνάρτηση «pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg)», η οποία παίρνει σαν παραμέτρους το όνομα του thread, τα χαρακτηριστικά του thread – αν υπάρχουν αλλιώς ορίζεται ως NULL, το όνομα της συνάρτησης που θα εκτελέσει το thread και οι παράμετροι της συνάρτησης, αν υπάρχουν. Επειδή πρέπει να συγχρονίσουμε τον τερματισμό των thread, ώστε να μην τερματίσει το πρόγραμμα μας όταν μια τερματίσει αλλά όταν τερματίσουν όλες, χρησιμοποιούμε την συνάρτηση «pthread_join(pthread_t thread, void **value_ptr)» που κάνει ακριβώς αυτή την λειτουργία. Αυτή η συνάρτηση παίρνει σαν παραμέτρους το όνομα του thread και ένα δείκτη για την επιστρεφόμενη τιμή της συνάρτησης που τρέχει το thread, αν δεν υπάρχει τότε το θέτουμε ως NULL.

Επειδή το κάθε ρομπότ επηρεαζόταν από τα αποτελέσματα του άλλου, θεώρησα σωστό ότι οι εναλλαγές στις συντεταγμένες κάθε ρομπότ πρέπει να αποτελέσει ένα είδος κρίσιμου σημείου, ώστε να διασφαλιστεί η ανάγνωση και η εγγραφή των συντεταγμένων χωρίς περιθώρια λαθών. Αυτό ήταν αναγκαίο να γίνει διότι κατά την εκτέλεση του αλγορίθμου παρατηρήθηκε ότι ο ταυτοχρονισμός της ανάγνωσης και της

εγγραφής στις συντεταγμένες των ρομπότ προκαλούσε κάποιο πρόβλημα, δηλαδή μπορεί να μην έπαιρνε την τελευταία εγγραφή ή να μην έπαιρνε την τελευταία ολοκληρωμένη εγγραφή, αλλά να έπαιρνε το μισό από την παλιά και το μισό από την τελευταία εγγραφή. Με το κλείδωμα αυτό των εντολών επιτυγχάναμε την σωστή ενέργεια για τον ταυτοχρονισμό της ανάγνωσης και της εγγραφής.

Για το σκοπό αυτό, η βιβλιοθήκη Pthread, έχει την μεταβλητή mutex η οποία κλειδώνει κάποιο κομμάτι κώδικα με σκοπό να το αποδεσμεύσει μόλις εκτελεστεί το κομμάτι αυτό, ώστε να μην μπορεί κανείς άλλος να εκτελέσει μια εργασία ανάγνωσης ή εγγραφής πάνω σε αυτό το κομμάτι. Χρειάστηκε η δήλωση μιας μεταβλητής τύπου pthread_mutex_t και η συνάρτηση int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr) για την αρχικοποίηση της μεταβλητής αυτής. Στο κομμάτι κώδικα που θέλαμε να προστατεύσουμε έπρεπε να δημιουργήσεις μια αίσθηση μπλοκ, αρχίζοντας το με την εντολή int pthread_mutex_lock(pthread_mutex_t *mutex) και τελειώνοντας το με την εντολή int pthread_mutex_unlock(pthread_mutex_t *mutex), όπου και οι δύο αυτές εντολές παίρνουν σαν παράμετρο το όνομα του mutex . Η πρώτη εντολή δίνει το σύνθημα για κλείδωμα της εντολής, είναι λες και υψώνεται μια σημαία και δεν αφήνει άλλο να κάνει κάτι πάνω σε αυτές τις εντολές μέχρι να κατεβεί, και η δεύτερη τερματίζει την λειτουργία του μπλοκ, δηλαδή κατεβάζει την σημαία.

Κεφάλαιο 3

Υλοποίηση Αλγορίθμου

Σε αυτό το κεφάλαιο θα δοθεί λεπτομερής περιγραφεί των συναρτήσεων, καθώς και των φάσεων που μπορεί να μεταβεί το ρομπότ και οι κύριες διαδικασίες του αλγορίθμου. Πρώτα όμως θα δοθεί λεπτομερή περιγραφή της δομής του ρομπότ, η οποία αποτελείται από ένα πίνακα, όπου κάθε του θέση αντιπροσωπεύει και ένα ρομπότ στο πεδίο και κάθε ρομπότ στον πίνακα έχει την πιο κάτω δομή:

```
typedef struct{
        float x, y, px, py;
        char state;
        int statec, numV, chNumV;
        float visible[MAX][2], ch[MAX][2], cc[MAX][5];
}robotnode;

typedef struct{
        robotnode c[MAX];
}Robot;
```

Τα τέσσερα πρώτα στοιχεία αντιπροσωπεύουν συντεταγμένες, πιο συγκεκριμένα οι δύο πρώτες είναι οι υφιστάμενες συντεταγμένες του κέντρου του ρομπότ και οι άλλες δύο είναι οι προηγούμενες συντεταγμένες του ρομπότ. Τα δύο επόμενα στοιχεία παρουσιάζουν σε ποία κατάσταση βρίσκεται το ρομπότ κάθε στιγμή. Πιο συγκεκριμένα το πρώτο αντιστοιχεί στις φάσεις που μπορεί να βρίσκεται ανά πάσα στιγμή το ρομπότ και το άλλο στοιχείο αντιστοιχεί στην κατάσταση που βρίσκεται το ρομπότ στην φάση υπολογίζω, για τις υπόλοιπες φάσεις το πεδίο αυτό είναι ίσο με -1. Οι επόμενοι τρεις πίνακες αποθηκεύουν διάφορες πληροφορίες που πρέπει να γνωρίζει το ρομπότ κατά διαστήματα μέσα στο πρόγραμμα. Ο πρώτος πίνακας αποθηκεύει τις συντεταγμένες των ρομπότ που βλέπει το ρομπότ στο πεδίο του. Ο δεύτερος πίνακας αποθηκεύει τις συντεταγμένες των ρομπότ που βλέπει το ρομπότ στο πεδίο του και βρίσκονται πάνω στο κυρτό περίβλημα. Ο τρίτος πίνακας αποθηκεύει τα στοιχεία για τις συγκεντρώσεις

που δημιουργούνται σε κάποια στιγμή μέσα στο πρόγραμμα. Τα τελευταία δυο πεδία αποτελούν τα μεγέθη των δύο πρώτων πινάκων, αντίστοιχα.

3.1 Γεωμετρικές Συναρτήσεις

Οι γεωμετρικές συναρτήσεις αποτελούν μια συλλογή από συναρτήσεις που υλοποιούν γεωμετρικούς υπολογισμούς και έχουν βοηθητικό σκοπό, αφού χρησιμοποιούνται ως υπόβαθρο για τις κύριες συναρτήσεις του αλγορίθμου που καλούν τα ρομπότ. Λεπτομερής κώδικας για τις συναρτήσεις αυτές δίδεται στο Παράρτημα Α.

3.1.1 Συνάρτηση ΕίμαιΠάνωΣτοΚυρτόΠερίβλημα

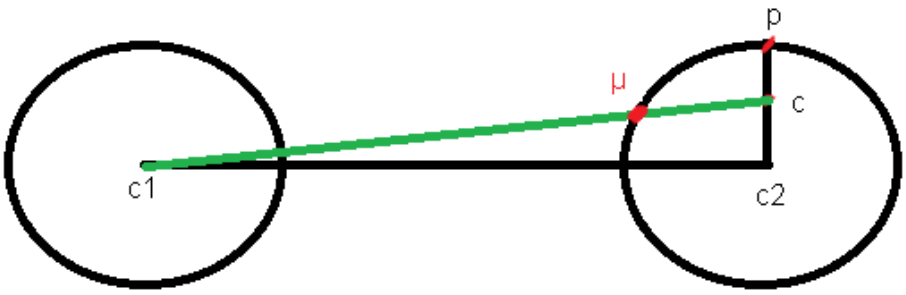
Η συνάρτηση αυτή έχει ως στόχο να δούμε αν το συγκεκριμένο ρομπότ που την καλεί βρίσκεται πάνω στο κυρτό περίβλημα. Για τον σκοπό αυτό, η συνάρτηση παίρνει σαν είσοδο τον πίνακα με τα ρομπότ που βλέπει το συγκεκριμένο ρομπότ, τις συντεταγμένες του που βρίσκεται αυτή την χρονική στιγμή και υπολογίζοντας τα σημεία πάνω στο κυρτό περίβλημα ελέγχει αν περιλαμβάνεται ο εαυτός του μέσα, αν ναι επιστρέφει NAI, είμαι, αλλιώς OXI, δεν είμαι.

Για τον υπολογισμό των σημείων που βρίσκονται στο κυρτό περίβλημα χρησιμοποίησα τον αλγόριθμο του Graham[6,7], ο οποίος βρίσκει το πιο μικρό σημείο σε σχέση με τον άξονα των y και γίνεται μια ταξινόμηση όλων των σημείων με βάση αυτό το σημείο σε σύγκριση με την πολική γωνία τους, αν υπάρχουν ισοπαλίες σπάζονται με τον έλεγχο για το μεγαλύτερο x. Για την εύρεση της σωστής σειράς στην ταξινόμηση απλά από το μικρότερο σημείο που βρήκα πριν προχωρούσα δεξιά και έλεγχα τον αριθμό που επέστρεφε η συνάρτηση «ccw(p1,p2,p3)», που κάνει μια μαθηματική πράξη, (p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x), η οποία με βάση το αποτέλεσμα της γνωρίζαμε αν το σημείο αυτό βρίσκεται δεξιά ή αριστερά της γραμμής που σχηματίζει το ρομπότ μας με το συγκεκριμένο σημείο. Έχοντας ταξινομήσει τα στοιχεία δημιούργησα μια στοίβα, στην οποία αρχικά αποθήκευσα τα δυο πρώτα στοιχεία του ταξινομημένου πίνακα και για κάθε επόμενο έλεγχα με την πιο πάνω συνάρτηση, που βρίσκεται το νέο σημείο με βάση την γραμμή των δύο πρώτων στοιχείων στην στοίβα. Ανάλογα με τον αριθμό που επιστρέφει η συνάρτηση ή έμπαινε το νέο στοιχείο στην

λίστα και έβγαινε το πρώτο που υπήρχε πριν μέσα ή έμεναν και τα δύο μέσα. Με το τέλος αυτής της διαδικασίας, τα στοιχεία που μένουν μέσα στην στοίβα αποτελούν και τα ρομπότ που είναι πάνω στο κυρτό περίβλημα.

3.1.2 Συνάρτηση ΜετακινήσουΣτοΣημείο

Η συνάρτηση αυτή παίρνει σαν είσοδο δύο κέντρα και βγάζει σαν έξοδο ένα σημείο στο οποίο θα μπορεί να μετακινηθεί το ρομπότ ώστε να επιτύχει επαφή με το άλλο ρομπότ. Όπως φαίνεται στο σχήμα πιο κάτω.



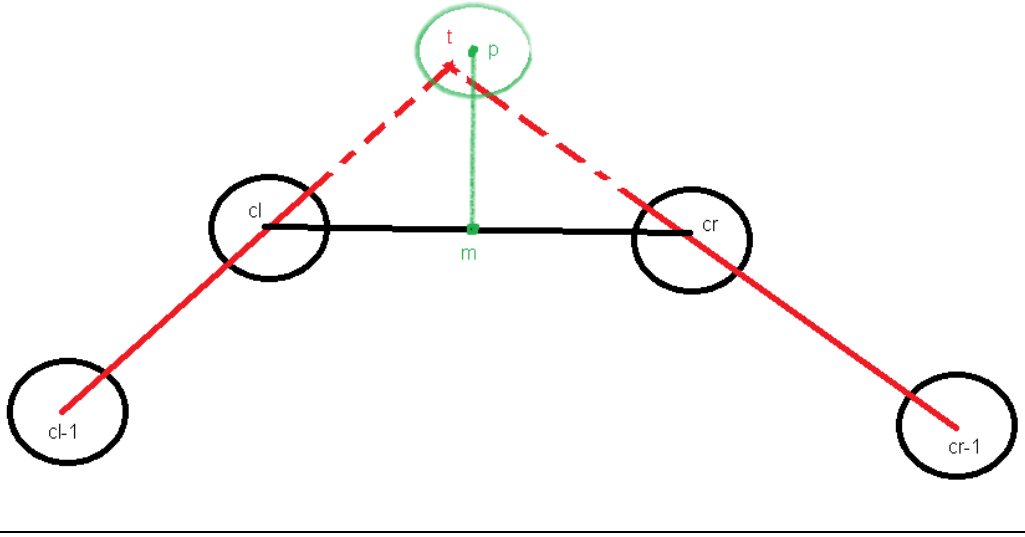
Σχήμα 3.1.1 - ο τρόπος υπολογισμός του σημείου μ

Για τον υπολογισμό του σημείου που θα μετακινηθεί το ρομπότ, με κέντρο c1, βρίσκω το σημείο p, το οποίο βρίσκεται πάνω στην περίμετρο του δεύτερου ρομπότ, με κέντρο c2. Για να το βρω αυτό βρίσκω την εξίσωση της ευθείας που περνά από τα δύο κέντρα των ρομπότ (μαύρη οριζόντια ευθεία), και αφού η ευθεία του σημείου p με το δεύτερο ρομπότ είναι κάθετη σε αυτή, ξέρω ότι η κλίση του p με το c2 είναι η αντίστροφος ανάλογη από την κλίση του c1c2 . Με βάση την κλίση που βρήκα και το σημείο της ευθείας που ξέρω, δηλαδή το c2, βρίσκω το σημείο στο οποίο τέμνει η ευθεία τον άξονα x. Μετά υπολογίζω την γωνία που δημιουργεί η ευθεία με τον άξονα x και με βάση την γωνία και την απόσταση που έχει από το σημείο c2, η οποία είναι ίση με την ακτίνα του κύκλου, δηλαδή 1, υπολογίζω το σημείο p πάνω στην περίμετρο του δεύτερου ρομπότ. Στην συνέχεια, βρίσκω το σημείο c που βρίσκεται επίσης πάνω στην ευθεία pc2, αφού ξέρω την γωνία και ξέρω την απόσταση (1/2m - ε) που έχει από το c2 το υπολογίζω. Για να βρω το τελικό σημείο μετακίνησης του ρομπότ, σημείο μ, λύνω σύστημα ευθείας με κύκλου. Παίρνω την ευθεία c1c και την εξίσωση του κύκλου με

κέντρο c2 και λύνω στο σύστημα. Με την λύση αυτή θα προκύψουν δύο σημεία, από αυτά τα δύο ελέγχω πιο από αυτό βρίσκεται σε κοντινότερη απόσταση από τον κύκλο με κέντρο c1 και το επιστρέφω.

3.1.3 Συνάρτηση ΒρέςΤαΣημεία

Η συνάρτηση αυτή βρίσκει σημεία πάνω στο κυρτό περίβλημα, τα οποία δεν θα επηρεάσουν τα υπόλοιπα σημεία πο υ είναι πάνω στο κυρτό περίβλημα, αλλά βρίσκονται μεταξύ αυτών. Η συνάρτηση αυτή παίρνει σαν είσοδο τα σημεία του κυρτού περιβλήματος και επιστρέφει τα σημεία που βρήκε. Όπως φαίνεται στο πιο κάτω σχήμα.



Σχήμα 3.1.2 – πως υπολογίζεται το σημείο p

Για την υλοποίηση αυτής της συνάρτησης δημιούργησα ένα πίνακα, στον οποίο θα αποθηκεύονται τα σημεία που θα βρίσκει η συνάρτηση αυτή. Στην συνέχεια, παίρνοντας τα σημεία πάνω στο κυρτό περίβλημα με βάση τη φορά το υ ρολογιού, δηλαδή δεξιόστροφα, ελέγχω κάθε δύο γειτονικά σημεία αν η απόσταση τους είναι μεγαλύτερη από 2. Αν ισχύει αυτό τότε, βρίσκω το κέντρο, m, της ευθείας τους, με την πράξη (x1+x2)/2 και ανάλογα για την συνιστώσα y, και βρίσκοντας την κάθετη που περνά από το σημείο του κέντρου m, επεκτείνω την ευθεία αυτή προς τα έξω το υ κυρτού περιβλήματος με απόσταση ίση με 1/n, και ονομάζω το σημείο αυτό με p. Στην συνέχεια, βρίσκω το σημείο που τέμνει τις δύο ευθείες, την ευθεία του αριστερο ύ

γείτονα με τον αριστερό του γείτονα και την ευθεία του δεξιού γείτονα με τον δεξιό του γείτονα. Για να δω αν το σημείο p που βρήκα είναι όντως το σωστό σημείο πρέπει να ελέγξω αν δεν έχω σημεία πάνω από το σημείο p και αν το σημείο p βρίσκεται σε απόσταση μεγαλύτερη ή ίση από απόσταση 1/n από τις ευθείες του σημείου που τέμνονται οι δύο ευθείες με τα γειτονικά σημεία των αρχικών γειτόνων, αν ισχύει αυτό τότε το σημείο αυτό προσθέτεται στον πίνακα, αλλιώς όχι. Όταν ελέγξουμε όλα τα σημεία πάνω στο κυρτό περίβλημα ο πίνακας με τα σημεία που βρήκα επιστρέφεται.

3.1.4 Συνάρτηση ΕπέστρεψεΤιςΣυγκεντρώσεις

Η συνάρτηση αυτή παίρνει σαν είσοδο το σύνολο των σημείων που βλέπει το ρομπότ και τις συνιστώσες του ρομπότ που καλεί την συνάρτηση. Επιστρέφει ένα πίνακα από στοιχεία τα οποία αποτελούν τις συγκεντρώσεις, δηλαδή είναι μια συλλογή από ρομπότ τα οποία είναι ενωμένα, έχουν μια συγκεκριμένη απόσταση μεταξύ τους. Τα στοιχεία που περιλαμβάνει ο πίνακας εξόδου της συνάρτησης είναι το δεξιότερο ρομπότ της συγκέντρωσης, το αριστερότερο ρομπότ και τον αριθμό των ρομπότ που περιέχονται στην συγκεκριμένη συγκέντρωση.

Ξεκινώντας από το σημείο του ρομπότ που καλεί την συνάρτηση αυτή, προχωρούμε προς τα δεξιά μέχρι να βρούμε ένα κενό, το ονομάζουμε κενό0. Αν το κενό0 είναι μικρότερο ή ίσο από 1/2m, όπου m είναι ο αριθμός των ρομπότ που βλέπει το συγκεκριμένο ρομπότ, τότε προχωρώ δεξιά μέχρι να βρω άλλο κενό. Αυτό συνεχίζεται μέχρι να προστεθούν τα σημεία στο σύνολο των συγκεντρώσεων και επαναλαμβάνεται το ίδιο με διαφορετική αρχική κατάσταση, η οποία ορίζεται από τον αλγόριθμο της συνάρτησης. Αν το κενό έχει μεγαλύτερο μήκος από αυτό που δώσαμε πιο πάνω τότε προχωρά προς τα αριστερά.

Για την υλοποίηση της συνάρτησης αυτής χρειάστηκε να δημιουργήσω μια βοηθητική συνάρτηση που να υπολογίζει την απόσταση δύο σημείων με βάση την ευκλείδεια απόσταση, $distance(p1,p2)=\sqrt{(p1.x-p2.x)^2+(p1.y-p2.y)^2}$. Για να μπορώ να αποθηκεύω τα ακραία σημεία της συγκέντρωσης αποθήκευα στην αρχή του μπλοκ κάθε κενού την θέση του αρχικού σημείου και έτσι ήξερα ποια ήταν τα δύο ακραία σημεία και οι έλεγχοι για την ύπαρξη ή όχι κενού υλοποιήθηκαν με συνθήκες ελέγχου.

3.1.5 Συνάρτηση ΠόσηΑπόστασηΑπέχει

Η συνάρτηση αυτή παίρνει σαν είσοδο το σύνολο των ρομπότ που βλέπει το ρομπότ που καλεί την συνάρτηση και τις συνιστώσες του συγκεκριμένου ρομπότ. Σαν έξοδο επιστρέφει ένα αριθμό από το σύνολο 1 μέχρι 3, μετρώντας την απόσταση που έχει κάθε συγκέντρωση με τον δεξιό της γείτονα. Αν επιστρέψει 1, τότε σημαίνει ότι είτε περιλαμβάνεται στην συγκέντρωση που έχει την μικρότερη απόσταση από τον δεξιό της γείτονα, είτε ισχύει το προηγούμενο αλλά έχει κι άλλες συγκεντρώσεις που έχουν την ίδια απόσταση με τον δεξιό τους γείτονα, όχι όλες όμως, και η δεξιά σου συγκέντρωση δεν έχει την μικρότερη απόσταση από τον δεξιά της γείτονα. Αν επιστρέψει 2, τότε σημαίνει όλες οι συγκεντρώσεις έχουν την ίδια απόσταση από την δεξιά γειτονική τους συγκέντρωση. Αν επιστρέψει 3, τότε είτε έχεις την μικρότερη απόσταση από τον δεξιό γείτονα σου αλλά έχουν κι άλλοι την μικρότερη απόσταση από τον δικό τους γείτονα και η δεξιά σου συγκέντρωση έχει κι αυτή την μικρότερη απόσταση από τον δικό της δεξιό γείτονα, είτε δεν ισχύει τίποτα από τους πιο πάνω περιορισμούς.

Για την υλοποίηση αυτής της συνάρτησης αυτής, ως πρώτο βήμα βρήκα την μικρότερη απόσταση που απέχει μια συγκέντρωση από την δεξιά της συγκέντρωση. Στην συνέχεια, έλεγξα πόσες συγκεντρώσεις έχουν αυτή την απόσταση με τον γείτονα τους, καθώς επίσης και αν από αυτές που την έχουν, είναι η συγκέντρωση με το ρομπότ που ελέγχουμε. Αν ο μετρητής για τις όμοιες σε απόσταση συγκεντρώσεις είναι ίσος με τον αριθμό των συγκεντρώσεων τότε επιστρέφει τον αριθμό 2. Αν ο μετρητής είναι ίσος με ένα και η σημαία για το αν το ρομπότ βρισκόταν σε αυτή την συγκέντρωση είναι θετική τότε επιστρέφει 1. Αν η σημαία είναι θετική αλλά ο μετρητής είναι μεγαλύτερος από ένα και μικρότερος από τον συνολικό αριθμό συγκεντρώσεων, τότε γίνεται έλεγχος για την δεξιά γειτονική συγκέντρωση, αν είναι στο σύνολο των συγκεντρώσεων που έχουν μικρότερη απόσταση τότε επιστρέφει 3, αλλιώς επιστρέφει 1. Αν δεν ισχύει τίποτα από όλα αυτά, καμιά συνθήκη, τότε επιστρέφει 3.

3.1.6 Συνάρτηση ΕίσαιΣτηνΜεγαλύτερηΣυγκέντρωση

Η συνάρτηση αυτή παίρνει σαν είσοδο το σύνολο των ρομπότ που βλέπει το ρομπότ που καλεί την συνάρτηση και τις συνιστώσες του συγκεκριμένου ρομπότ. Σαν έξοδο επιστρέφει ένα αριθμό από το σύνολο 1 μέχρι 3. Αν επιστρέψει 1 τότε σημαίνει ότι ανήκει στην μεγαλύτερη συγκέντρωση ή ανήκεις στην μεγαλύτερη συγκέντρωση αλλά έχει κι άλλες συγκεντρώσεις με τον ίδιο αριθμό ρομπότ, όχι όλες. Αν επιστρέψει 2 τότε σημαίνει ότι όλες οι συγκεντρώσεις έχουν τον ίδιο αριθμό ρομπότ στις συγκεντρώσεις τους. Διαφορετικά επιστρέφει 3.

Για την υλοποίηση αυτής της συνάρτησης, αρχικά έγινε μια επανάληψη για να βρεθεί ο μεγαλύτερος αριθμός συγκέντρωσης που υπάρχει την συγκεκριμένη στιγμή. Αφού βρεθεί ο αριθμός αυτός τότε γίνεται μια ακόμα επανάληψη, η οποία ελέγχει πόσες συγκεντρώσεις έχουν αυτό τον αριθμό και αν από αυτές τις συγκεντρώσεις ανήκει και η συγκέντρωση του ρομπότ μας που ελέγχουμε. Αν ο μετρητής είναι ίσος με τον συνολικό αριθμό των ρομπότ τότε επιστρέφει 2. Αν ο μετρητής είναι ίσος με ένα ή είναι μεγαλύτερος από ένα και μικρότερος από τον συνολικό αριθμό συγκεντρώσεων, τότε επιστρέφει 1. Αλλιώς, επιστρέφει 3.

3.1.7 Συνάρτηση ΕίσαιΣτηνΜικρότερηΣυγκέντρωση

Η συνάρτηση αυτή είναι ακριβώς η ίδια με την πιο πάνω συνάρτηση με μόνη διαφορά ότι εδώ συγκρίνουμε για την μικρότερη συγκέντρωση. Παίρνει σαν είσοδο το σύνολο των ρομπότ που βλέπει το ρομπότ που καλεί την συνάρτηση και τις συνιστώσες του συγκεκριμένου ρομπότ. Σαν έξοδο επιστρέφει ένα αριθμό από το σύνολο 1 μέχρι 3. Αν επιστρέψει 1 τότε σημαίνει ότι ανήκει στην μικρότερη συγκέντρωση ή ανήκει στην μικρότερη συγκέντρωση αλλά ανήκουν κι άλλες, όχι όλες, και η δεξιά της συγκέντρωση δεν ανήκει στο σύνολο των συγκεντρώσεων που έχουν τον μικρότερο αριθμό ρομπότ. Αν επιστρέψει 2 τότε σημαίνει ότι όλες οι συγκεντρώσεις έχουν τον ίδιο αριθμό ρομπότ. Και επιστρέφει 3, αν ανήκει στην μικρότερη συγκέντρωση και δεν είναι η μόνη συγκέντρωση και η δεξιά της συγκέντρωση ανήκει στο σύνολο με τις συγκεντρώσεις που έχουν τον λιγότερο αριθμό ρομπότ. Επίσης, επιστρέφει 3, αν δεν ισχύει καμία από τις πιο πάνω συνθήκες.

Για την υλοποίηση αυτής της συνάρτησης, αρχικά έγινε μια επανάληψη για να βρεθεί ο μικρότερος αριθμός συγκέντρωσης που υπάρχει την συγκεκριμένη στιγμή. Αφού βρεθεί ο αριθμός αυτός τότε γίνεται μια ακόμα επανάληψη, η οποία ελέγχει πόσες συγκεντρώσεις έχουν αυτό τον αριθμό και αν από αυτές τις συγκεντρώσεις ανήκει και η συγκέντρωση του ρομπότ μας που ελέγχουμε. Αν ο μετρητής είναι ίσος με τον συνολικό αριθμό των ρομπότ τότε επιστρέφει 2. Αν ο μετρητής είναι ίσος με ένα ή είναι μεγαλύτερος από ένα και μικρότερος από τον συνολικό αριθμό συγκεντρώσεων και η δεξιά σου συγκέντρωση δεν ανήκει στο σύνολο αυτό, τότε επιστρέφει 1. Αλλιώς, επιστρέφει 3, αν ανήκεις στην μικρότερη συγκέντρωση αλλά ανήκουν κι άλλες συγκεντρώσεις και η δεξιά σου συγκέντρωση ανήκει στο σύνολο ή αν δεν ισχύει τίποτα από τα πιο πάνω.

3.1.8 Συνάρτηση ΣτηνΊδιαΕυθεία2

Η συνάρτηση αυτή παίρνει σαν είσοδο τρία κέντρα ρομπότ και επιστρέφει αν ανήκουν στην ίδια ευθεία. Για να δω αν ισχύει αυτό βρήκα την εξίσωση της ευθείας για τα δύο ακρινά σημεία και στην συνέχεια έλυσα την εξίσωση με βάση τις συνιστώσες του κεντρικού σημείου, αν η λύση βγάζει μηδέν τότε σημαίνει ότι το σημείο είναι πάνω στην ευθεία, αλλιώς δεν είναι.

3.2 Συναρτήσεις Αλγορίθμου

Σε αυτό το υποκεφάλαιο θα γίνει περιγραφή των βασικών διαδικασιών του αλγόριθμου και τις φάσεις που αλλάζει ένα ρομπότ κατά την διάρκεια εκτέλεσης του συγκεκριμένου αλγορίθμου.

3.2.1 Φάσεις Αλγορίθμου

Κατά την διάρκεια εκτέλεσης του προγράμματος το ρομπότ αλλάζει πέντε φάσεις, οι οποίες είναι αλληλένδετες μεταξύ τους αφού η μία μεταβαίνει στην άλλη. Όλο αυτό συνεχίζεται μέχρι το ρομπότ να πληρεί τα κριτήρια τερματισμού και να μπορέσει να μπει στην φάση τερματισμού και να τερματίσει. Η υλοποίηση αυτού του κομματιού

κώδικα έγινε με την χρήση του switch και των cases, ο οποίος αποτελεί τον πιο εύκολο στην χρήση.

Φάση Περιμένω: Αυτή η φάση αποτελεί το σημείο που χάνεται ο συγχρονισμός στο πρόγραμμα διότι κάθε ρομπότ περιμένει διαφορετικό χρόνο από τα άλλα. Αυτό επιτυγχάνεται με την χρήση της συνάρτησης rand() της βιβλιοθήκης «math.h», η οποία επιστρέφει τυχαίους αριθμούς. Με την χρήση της συνάρτησης sleep() [8,9] της βιβλιοθήκης «time.h» το ρομπότ κοιμάται για ένα χρονικό διάστημα, δηλαδή δεν κάνει τίποτα, είναι σε πλήρης αδράνεια. Για το χρονικό διάστημα επέλεξα μέγιστη τιμή το 1 λεπτό, διότι πιο μεγάλοι αριθμοί δημιουργούσαν αρκετή καθυστέρηση στο πρόγραμμα.

Φάση Βλέπω: Στην φάση βλέπω το ρομπότ ελέγχει το πεδίο και αναγνωρίζει ποια από τα ρομπότ στο πεδίο βρίσκονται στο οπτικό του πεδίο, δηλαδή μπορεί να τα δει χωρίς να υπάρχει κάποιο εμπόδιο μπροστά. Για να διευκολυνθεί η υλοποίηση στους υπόλοιπους υπολογισμούς το ρομπότ προσθέτει και τον εαυτό του στα ρομπότ που βλέπει.

Στην υλοποίηση αυτό έγινε με τον υπολογισμό της ευθείας που δημιουργεί το ρομπότ με άλλο από το πεδίο και τον έλεγχο αν υπάρχει κάποιο άλλο ρομπότ που να βρίσκεται πάνω σε αυτή την ευθεία, δηλαδή που να εμποδίζει τα δύο αρχικά ρομπότ να δει το ένα το άλλο. Αυτό συνεχίζεται μέχρι να ελέγξει όλα τα ρομπότ που βρίσκονται στο πεδίο. Αν στην ευθεία δεν βρει άλλα που να είναι πάνω, τότε το προσθέτει στον πίνακα, αλλιώς προσθέτει το πιο κοντινό του από τα ρομπότ που βρίσκονται στην ευθεία αυτή. Αν το ρομπότ, το οποίο θα ελέγξει την συγκεκριμένη στιγμή ,αποτελεί τον εαυτό του, τότε δεν κάνει τις πιο πάνω ενέργειες αλλά προσθέτει τις συντεταγμένες του στον πίνακα κατευθείαν.

Φάση Υπολογίζω: Η φάση αυτή αποτελεί την πιο κρίσιμη φάση που εκτελεί το ρομπότ καθώς είναι αυτή που ορίζει τι κινήσεις θα γίνουν και πως θα προχωρήσει το ρομπότ. Σε αυτή την φάση έχουμε δεκαεπτά καταστάσεις τις οποίες μπορεί να επισκεφτεί το ρομπότ. Οι καταστάσεις αυτές θα δοθούν με λεπτομερή περιγραφή στο επόμενο υποκεφάλαιο. Και πάλι εδώ έγινε η χρήση του switch για την εύκολη εναλλαγή των καταστάσεων. Με το τέλος της φάσης αυτής το ρομπότ αλλάζει φάση και μπαίνει είτε

στην φάση Μετακινούμαι είτε στην φάση τερματίζω, αυτό εξαρτάται από την κατάσταση στην οποία θα τερματίσει το ρομπότ. Στο σύνολο υπάρχουν μια τερματική, οκτώ εσωτερικές μεταβιβάσεις και οκτώ που υπολογίζουν σημείο μετακίνησης του ρομπότ.

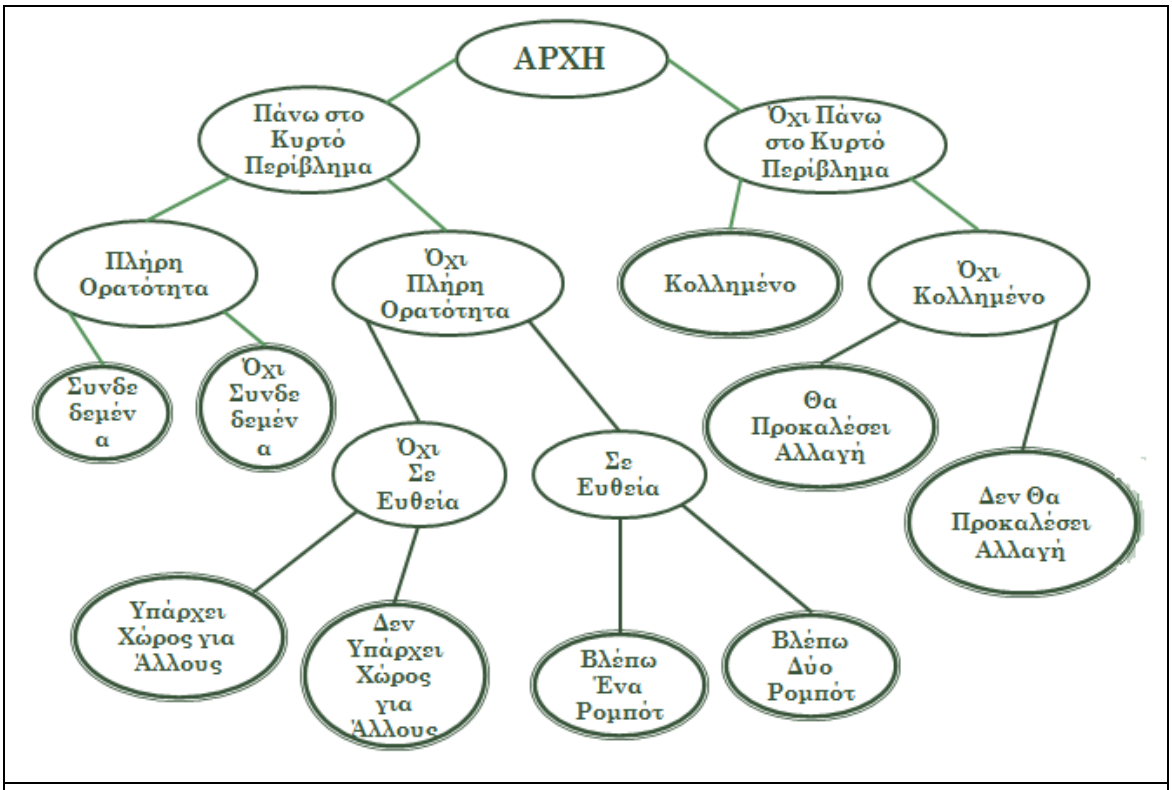
Φάση Μετακινούμαι: Στην φάση αυτή το ρομπότ έχοντας μεταβεί εδώ από την φάση υπολογίζω, στην οποία έχουν υπολογιστεί οι συνιστώσες στην οποία πρέπει να μετακινηθεί το ρομπότ, γίνεται η μετακίνηση στις νέες συνιστώσες ή αν οι συνιστώσες που επέστρεψε η προηγούμενη φάση είναι οι ίδιες με τις ήδη υπάρχουσες τότε μένει στην θέση του. Με την μετακίνηση το ρομπότ, η κατάσταση του αλλάζει σε Περιμένω.

Φάση Τερματίζω: Η φάση αυτή είναι ο τελικός σταθμός κάθε ρομπότ, καθώς αποτελεί την τελευταία φάση που θα επισκεφτεί. Εδώ το ρομπότ τερματίζει και δεν ξανακάνει άλλους υπολογισμούς, δηλαδή έχει φτάσει σε κατάσταση που το ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα, βλέπει όλα τα ρομπότ που υπάρχουν στον χώρο και είναι όλα συνδεδεμένα μεταξύ τους.

3.2.2 Διαδικασίες Αλγορίθμου

Η φάση υπολογίζω αποτελείται από δεκαεπτά διαδικασίες, τις οποίες επισκέπτεται το ρομπότ ανάλογα με τα δεδομένα που έχει και σε ποίες αποστέλλεται από αυτά. Κάποιες από αυτές τις διαδικασίες, οι οποίες είναι τερματικές, παίρνουν το ρομπότ στην φάση Μετακινούμαι ή στην φάση τερματίζω. Λεπτομερής κώδικας για τις διαδικασίες αυτές δίδεται στο Παράρτημα Β.

Γενικά, η ιδέα του αλγορίθμου είναι αρχικά να επιτευχθεί ο στόχος, δηλαδή τα ρομπότ να σχηματίσουν ένα κυρτό περίβλημα και όλοι να μπορούν να βλέπουν όλους. Μόλις επιτευχθεί αυτό αρχίζουν να συγκεντρώνονται, ώστε να φτάσουν όσο πιο κοντά γίνεται χωρίς να χαλάσουν τον αρχικό στόχο. Όταν ισχύουν όλα τα προαναφερθέντα τότε κάθε ρομπότ τερματίζει και το πρόβλημα συγκέντρωσης έχει λυθεί. Οι συνολικές διαδικασίες του αλγορίθμου φαίνονται στο πιο κάτω σχήμα.



Σχήμα 3.2.1 – οι διαδικασίες του αλγορίθμου

3.2.2.1 Διαδικασία Αρχή

Η αρχική διαδικασία που καλείται πάντα την πρώτη φορά που θα εκτελεστεί η κατάσταση Υπολογίζω, ελέγχει κατά πόσο το ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα. Αυτό γίνεται με το κάλεσμα της συνάρτησης ΠάνωΣτοΚυρτόΠερίβλημα, από το προηγούμενο υποκεφάλαιο. Αν επιστρέψει ότι το ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα τότε μεταβαίνει στην κατάσταση 2, διαδικασία ΠάνωΣτοΚυρτόΠερίβλημα, αλλιώς στην κατάσταση 13, διαδικασία ΌχιΠάνωΣτοΚυρτόΠερίβλημα.

Ψευδοκώδικας:

- Κάλεσε την συνάρτηση ΠάνωΣτοΚυρτόΠερίβλημα με είσοδο τα ρομπότ που βλέπεις και το κέντρο σου.
- Αν η συνάρτηση επιστρέψει ΝΑΙ, τότε μετακινήσου στην κατάσταση ΠάνωΣτοΚυρτόΠερίβλημα. Αλλιώς, μετακινήσου στην κατάσταση Όχι ΠάνωΣτοΚυρτόΠερίβλημα.

3. 2.2.2 Διαδικασία ΠάνωΣτοΚυρτόΠερίβλημα

Η διαδικασία αυτή καλείται όταν το ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα και ελέγχει αν το κυρτό περίβλημα περιλαμβάνει όλα τα ρομπότ ή όχι, άρα με το τέλος της διαδικασίας αυτής ξέρουμε κατά προσέγγιση πόσα ρομπότ είναι πάνω στο κυρτό περίβλημα.

Ψευδοκώδικας:

- Εάν μπορείς να δεις όλα τα υπόλοιπα ν-1 ρομπότ και όλα όσα βλέπεις είναι πάνω στο κυρτό περίβλημα, τότε για κάθε ένα ρομπότ που βλέπεις κάλεσε την συνάρτηση ΣτηνΊδιαΕυθεία2, με είσοδο τους δύο γείτονες του ρομπότ αυτού μαζί με τις δικές του συνιστώσες. Αν η συνάρτηση επιστρέψει ΝΑΙ τότε μετακινήσου στην κατάσταση ΌχιΠλήρηΟρατότητα, διαφορετίκα μετακινήσου στην κατάσταση ΠλήρηΟρατότητα.
- Αν δεν βλέπεις όλα τα υπόλοιπα ρομπότ τότε μετακινήσου στην κατάσταση ΌχιΠλήρηΟρατότητα.

Για την υλοποίηση, αρχικά ελέγχουμε αν το ρομπότ που καλεί την συγκεκριμένη διαδικασία βλέπει όλα τα ρομπότ και αν περιλαμβάνονται όλα πάνω στο κυρτό περίβλημα, αν ισχύει αυτό τότε ελέγχει για κάθε ρομπότ στο οπτικό του πεδίο αν βρίσκεται σε ευθεία γραμμή με τα γειτονικά του. Για τον σκοπό αυτό καλείτε η συνάρτηση ΣτηνΊδιαΕυθεία2, από το προηγούμενο υποκεφάλαιο. Αν σε κάποιο σημείο αυτή η συνάρτηση επιστρέψει ότι βρίσκονται σε ευθεία γραμμή τότε τερματίζει και μεταβαίνει στην κατάσταση 6, διαδικασία ΌχιΠλήρηςΟρατότητα. Αν τερματίσει ο πιο πάνω έλεγχος χωρίς να βρεθούν ρομπότ στην ίδια ευθεία τότε το ρομπότ μεταβαίνει στην κατάσταση 3, διαδικασία ΠλήρηςΟρατότητα. Αν από την αρχή δεν ισχύει ο περιορισμός, δηλαδή δεν βλέπει όλα τα ρομπότ ή όσα βλέπει δεν είναι πάνω στο κυρτό περίβλημα τότε τερματίζει η διαδικασία αυτή και μεταβαίνει στην κατάσταση 6, διαδικασία ΌχιΠλήρηςΟρατότητα.

3. 2.2.3 Διαδικασία ΠλήρηςΟρατότητα

Η διαδικασία αυτή εκτελείται όταν το ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα και όταν όλα τα ρομπότ βρίσκονται και αυτά πάνω και όλο αυτό συμβαίνει αφού το ρομπότ βλέπει όλα τα άλλα ρομπότ.

Ψευδοκώδικας:

- Διάλεξε τυχαία ένα ρομπότ από αυτά που βλέπεις και θέσε το σύνολο συγκέντρωση ίσο με αυτό.
- Μέχρι το σύνολο να μην αλλάζει κάνε:
    - Για κάθε στοιχείο στο σύνολο κάνε:
        - Έλεγξε κάθε στοιχείο από το οπτικό πεδίο του αρχικού ρομπότ, εκτός από τα στοιχεία στο σύνολο, αν εφάπτονται με το στοιχείο που το ελέγχει. Αν ισχύει αυτό τότε πρόσθεσε αυτό το ρομπότ στο σύνολο.
- Αν ο αριθμός του συνόλου είναι ίσο με τον αριθμό των ρομπότ, τότε μετακινήσου στην κατάσταση Συνδεδεμένα, διαφορετικά, μετακινήσου στην κατάσταση ΌχιΣυνδεδεμένα.

Για την υλοποίηση της διαδικασίας αυτής υλοποίησα δύο βοηθητικές συναρτήσεις, όπου η μια ελέγχει αν το συγκεκριμένο σημείο που θα ελέγξω είναι μέσα στο σύνολο των ήδη ελεγχόμενων στοιχείων και η άλλη ελέγχει αν δύο ρομπότ, με βάση τα κέντρα τους, εφάπτονται. Αρχικά προσθέτει στο σύνολο το πρώτο σημείο από το σύνολο των ρομπότ που βλέπει και με βάση αυτό ελέγχει στο σύνολο των ρομπότ που βλέπει αν αυτό που ελέγχει εφάπτεται με το ρομπότ στο σύνολο, αν ναι τότε το προσθέτει στο σύνολο. Αν με το τέλος αυτής της διαδικασίας το σύνολο των στοιχείων είναι ίσο με τον αριθμό των ρομπότ που βλέπει τότε μεταβαίνει στην κατάσταση 4, διαδικασία Συνδεδεμένα, αλλιώς μεταβαίνει στην κατάσταση 5, διαδικασία ΌχιΣυνδεδεμένα.

3. 2.2.4 Διαδικασία Συνδεδεμένα

Η διαδικασία αυτή εκτελείται όταν όλα τα ρομπότ βρίσκονται στο οπτικό πεδίο του συγκεκριμένου ρομπότ και όλα βρίσκονται πάνω στο κυρτό περίβλημα και είναι ενωμένα μεταξύ τους. Αυτή η διαδικασία τερματίζει την φάση υπολογίζω και παίρνει το ρομπότ στην φάση τερματίζω, δηλαδή τερματίζει το switch της φάσης υπολογίζω και επιλέγει την φάση τερματίζω.

Ψευδοκώδικας:

- Τερματίζει το ρομπότ. Θα μετακινηθεί στην φάση Τερματίζω.

3. 2.2.5 Διαδικασία ΌχιΣυνδεδεμένα

Η διαδικασία αυτή εκτελείται όταν όλα τα ρομπότ που βλέπει το συγκεκριμένο ρομπότ και το ίδιο είναι πάνω στο κυρτό περίβλημα. Αυτή η διαδικασία με το τέλος της τερματίζει την φάση υπολογίζω και παίρνει το ρομπότ στην φάση μετακινούμαι, ώστε να μεταβεί στις νέες συνιστώσες που θα υπολογιστούν από αυτή την διαδικασία.

Εδώ έχουμε πέντε βασικές περιπτώσεις που θα τερματίσουν την διαδικασία αυτή έχοντας υπολογίσει τις νέες συνιστώσες που θα μεταβεί το ρομπότ.

• Η πρώτη βασική περίπτωση είναι έχοντας τρία ρομπότ, όπου το αριστερό είναι το συγκεκριμένο ρομπότ που εκτελεί τις διαδικασίες, το μεσαίο ο αριστερός του γείτονας και το δεξί ο αριστερός γείτονας του γείτονα του, και η απόσταση μεταξύ της ευθείας του αριστερού και δεξιού ρομπότ με το μεσαίο ρομπότ είναι μικρότερη από απόσταση 1/n. Για αυτό τον υπολογισμό υλοποίησα μια συνάρτηση ΑπόστασηΣημείουΑποΕυθεία, η οποία βρίσκει την απόσταση του σημείου από την ευθεία, γνωρίζοντας την εξίσωση της. Στην συνέχεια, αν η απόσταση που επέστρεψε η πιο πάνω συνάρτηση είναι μικρότερη από 1/n τότε υπολογίζω το x, ως την μέγιστη απόσταση που μπορεί να κινηθεί το ρομπότ χωρίς να κρύψει κάποιο ρομπότ. Το x το υπολογίζω ως την απόσταση που έχει το ρομπότ με την ευθεία των δύο γειτονικών του

ρομπότ, αλλά προσθέτοντας ένα πολύ μικρό αριθμό ώστε να μην μπλοκάρει την ορατότητα των γειτονικών ρομπότ μεταξύ τους. Αυτός ο αριθμός, για το x, προκύπτει με την εύρεση της εξίσωσης των δύο γειτονικών ρομπότ, την εύρεση του σημείου που τέμνει η ευθεία που περνά το κέντρο του ρομπότ με την ευθεία των δύο γειτονικών ρομπότ και την εύρεση της απόστασης του σημείου αυτού με το κέντρο του συγκεκριμένου ρομπότ, με την τοπολογία το συγκεκριμένο ρομπότ να είναι είτε το μεσαίο ρομπότ είτε το δεξί ρομπότ μιας τριάδας από γειτονικά ρομπότ. Αφαιρώντας ένα μικρό αριθμό από αυτή την απόσταση έχουμε υπολογίσει το x. Αν το x αποτελεί απόσταση που δεν επηρεάζει την ορατότητα κάποιου ρομπότ και με οποιοδήποτε συνδυασμό, έχοντας το συγκεκριμένο ρομπότ σε τριάδα με άλλα δύο – είτε αριστερό, είτε δεξί, είτε στο κέντρο, η απόσταση του μεσαίου ρομπότ με την ευθεία των δύο ακρινών ρομπότ να είναι μικρότερη της απόστασης 1/2n, τότε επιλέγω ως νέα απόσταση x’ την απόσταση που είναι μικρότερη μεταξύ των τιμών x και 1/2n. Αφού έχω υπολογίσει την απόσταση που θα μετακινηθεί το ρομπότ μου τώρα βρίσκω την γωνιά που κάνει η ευθεία των δυο γειτονικών ρομπότ με την ευθεία που περνά κάθετα από την ευθεία τους και από το κέντρο του ρομπότ. Έχοντας την απόσταση και την γωνία βρίσκω το νέο σημείο στο οποίο θα μετακινηθεί το συγκεκριμένο ρομπότ.

• Η δεύτερη περίπτωση ελέγχει αν το συγκεκριμένο ρομπότ περιέχεται σε ένα σύνολο από εφαπτόμενα ρομπότ όπου ισχύει ο αρχικός περιορισμός της πιο πάνε περίπτωσης και το πιο δεξί ρομπότ δεν μπορεί να μετακινηθεί εκτελείτε η ίδια διαδικασία με την πιο πάνω περίπτωση με μόνη διαφορά τον συνδυασμό της τριάδας των γειτονικών ρομπότ, όπου το συγκεκριμένο ρομπότ είναι αντί το δεξί , το αριστερό ρομπότ. Για την εύρεση αν ένα ρομπότ εφάπτεται με το γειτονικό του, υλοποίησα μια συνάρτηση Εφάπτονται, η οποία παίρνει σαν είσοδο τις δύο συνιστώσες των ρομπότ και υπολογίζοντας την ευκλείδεια απόσταση, αν η απόσταση είναι ίση με μηδέν τότε επιστρέφει ότι εφάπτονται αλλιώς ότι δεν εφάπτονται.

• Η τρίτη περίπτωση ελέγχει κατά πόσο υπάρχει ένας συνδυασμός τριών γειτονικών ρομπότ που η απόσταση του μεσαίου ρομπότ με την ευθεία των ακραίων ρομπότ είναι μικρότερη από 1/n. Αυτό υπολογίζεται με μια επανάληψη στον πίνακα των ρομπότ στο σύνολο των ρομπότ πάνω στο κυρτό περίβλημα και το κάλεσμα της συνάρτησης ΑπόστασηΣημείουΑποΕυθεία, η οποία υπολογίζει την απόσταση που ζητάμε. Αν βρει

έστω και ένα συνδυασμό που να έχουν απόσταση περισσότερη από την ζητούμενη τότε τερματίζει και επιστρέφει τις υπάρχουσες συνιστώσες, δηλαδή το ρομπότ δεν θα μετακινηθεί.

- Η τέταρτη περίπτωση ελέγχει κατά πόσο τα δύο γειτονικά ρομπότ του συγκεκριμένου ρομπότ εφάπτονται με αυτόν. Αυτό έγινε με την εύρεση των δύο γειτονικών ρομπότ του συγκεκριμένου ρομπότ και το κάλεσμα της συνάρτησης Εφάπτονται και για τις δύο περιπτώσεις. Αν και οι δυο επιστρέψουν ότι εφάπτονται τότε τερματίζει επιστρέφοντας τις ήδη υπάρχουσες συνιστώσες για νέες, δηλαδή δεν θα μετακινηθεί σε νέα θέση.

- Η πέμπτη περίπτωση ελέγχει αν το ρομπότ εμπεριέχεται σε μια συγκέντρωση ρομπότ διαφορετικά καλεί τις συναρτήσεις ΕίσαιΣτηνΜεγαλύτερηΣυγκέντρωση και ΕίσαιΣτηνΜικρότερηΣυγκέντρωση. Στην υλοποίηση αυτής της διαδικασίας αρχικά καλείτε η συνάρτηση ΕπέστρεψεΤιςΣυγκεντρώσεις και ελέγχουμε αν ο αριθμός των συγκεντρώσεων που δημιουργηθήκαν είναι ένα, τότε έχοντας την τριάδα γειτονικών ρομπότ, δηλαδή το συγκεκριμένο ρομπότ να είναι στην μέση και τα άλλα δύο ο αριστερός και δεξιός γείτονας του ρομπότ, βρίσκω την κάθετη γραμμή που περνά από το συγκεκριμένο ρομπότ από την ευθεία των άλλων δυο ρομπότ. Την κάθετη την βρίσκω, έχοντας τα δύο σημεία από την άλλη ευθεία, βρίσκω την κλίση της και της κάθετης είναι η αντιστρόφως ανάλογη. Βρίσκω το σημείο που τέμνει η κάθετη γραμμή τον άξονα τον x, με την συνάρτηση ΈρευσηΣημείουΠουΤέμνειΤονΆξοναΧ, η οποία το υπολογίζει με την λύση της εξίσωσης της ευθείας με την κλίση της ευθείας που βρήκαμε πριν και τις συνιστώσες του ρομπότ, ως προς b και μετά βρήκαμε το σημείο x στον άξονα με την πράξη –a/b για το σημείο που τέμνει τον άξονα x. Αφού βρήκαμε αυτό το σημείο, βρήκαμε την γωνία, με την υλοποίηση της συνάρτησης ΓωνίαΜεταξυΔύοΓραμμών, η οποία επιστρέφει την γωνιά δύο γραμμών, που κάνει η κάθετη ευθεία με τον άξονα x και έχοντας την απόσταση που πρέπει να έχει το νέο σημείο με το ρομπότ και την γωνία βρήκαμε το νέο σημείο, που αποτελεί τις νέες συνιστώσες που θα μετακινηθεί το συγκεκριμένο ρομπότ. Αν η συνάρτηση ΕπέστρεψεΤιςΣυγκεντρώσεις επιστρέψει περισσότερα από μια συγκέντρωση τότε καλείτε η συνάρτηση ΕίσαιΣτηνΜεγαλύτερηΣυγκέντρωση με είσοδο τις συνιστώσες του συγκεκριμένο υ ρομπότ και το σύνολο των ρομπότ πο υ βλέπει. Αν αυτή η συνάρτηση επιστρέψει 1, τότε επιστρέφει τις υπάρχουσες συνιστώσες, δηλαδή το ρομπότ δεν θα μετακινηθεί. Αν επιστρέψει 2 τότε καλείτε η συνάρτηση

ΕίσαιΣτηνΜικρότερηΣυγκέντρωση. Αν η συνάρτηση αυτή επιστρέψει 1, τότε καλείτε η συνάρτηση ΜετακηνίσουΣτοΣημείο με είσοδο τις συνιστώσες του συγκεκριμένου ρομπότ και του δεξιού του γείτονα πάνω στο κυρτό περίβλημα και επιστρέφει το σημείο που θα επιστρέψει αυτή η συνάρτηση ως τις νέες συνιστώσες του συγκεκριμένου ρομπότ. Αν η συνάρτηση επιστρέψει 2 , τότε καλείτε η συνάρτηση ΕπέστρεψεΤιςΣυγκεντρώσεις με είσοδο τις συνιστώσες του συγκεκριμένου ρομπότ και του δεξιού του γείτονα πάνω στο κυρτό περίβλημα και με τις συγκεντρώσεις που θα επιστρέψει βρες αυτή που περιλαμβάνει το συγκριμένο ρομπότ. Δεδομένου των δύο ακραίων ρομπότ αυτής της συγκέντρωσης, δημιουργούμε την ευθεία των δύο αυτών σημείων και έχοντας την κλίση της ευθείας τους δημιουργούμε την παράλληλη γραμμή που περνά από το σημείο του κέντρου του συγκεκριμένου ρομπότ. Η εξίσωση της παράλληλης προκύπτει από την κλίση της άλλης ευθείας, αφού είναι παράλληλες τότε έχουν την ίδια κλίση, και λύνουμε την εξίσωση με βάση τις συνιστώσες του συγκεκριμένου ρο μπότ. Στην συνέχεια δημιουργούμε την κάθετη γραμμή που περνά από το συγκεκριμένο ρομπότ με την παράλληλη ευθεία που περνά από το ρομπότ. Αυτή προκύπτει από την αντιστρόφως ανάλογη κλίση της παράλληλης και την λύση της εξίσωσης με τις συνιστώσες του συγκεκριμένου ρομπότ. Βρίσκω το σημείο που τέμνει την κάθετη με τον άξονα x, όπως περιγράφηκε πιο πάνω και με τον ίδιο τρόπο όπως και πριν βρίσκω την γωνία της ευθείας με τον άξονα x. Έχοντας την γωνία και την απόσταση που θέλω να είναι το νέο σημείο, το υπολογίζω και το ορίζω ως τις νέες συνιστώσες πο υ θα μετακινηθεί το συγκεκριμένο ρομπότ. Αλλιώς, αν δηλαδή η συνάρτηση ΕίσαιΣτηνΜικρότερηΣυγκέντρωση επιστέψει 3, επιστρέφει τις ήδη υπάρχουσες συνιστώσες, δηλαδή το ρομπότ δεν θα μετακινηθεί. Αν τώρα η συνάρτηση ΕίσαιΣτηνΜεγαλύτερηΣυγκέντρωση επιστρέψει 3, τότε καλείτε η συνάρτηση ΕίσαιΣτηνΜικρότερηΣυγκέντρωση με είσοδο τις συνιστώσες του συγκεκριμένου ρομπότ και το σύνολο των ρομπότ που βλέπει. Αν αυτή η συνάρτηση επιστρέψει 1, τότε καλείται η συνάρτηση ΜετακηνίσουΣτοΣημείο με είσοδο τις συνιστώσες του συγκεκριμένου ρομπότ και του δεξιο ύ του γείτονα πάνω στο κυρτό περίβλημα και επιστρέφει το σημείο που θα επιστρέψει αυτή η συνάρτηση ως τις νέες συνιστώσες του συγκεκριμένου ρομπότ. Αν επιστρέψει 2, τότε καλείται η συνάρτηση ΠόσηΑπόστασηΑπέχει. Αν η συγκεκριμένη συνάρτηση επιστρέψει 1, τότε καλείται η συνάρτηση ΜετακηνίσουΣτοΣημείο με είσοδο τις συνιστώσες του συγκεκριμένου ρομπότ και του δεξιού του γείτονα πάνω στο κυρτό περίβλημα και επιστρέφει το

σημείο που θα επιστρέψει αυτή η συνάρτηση ως τις νέες συνιστώσες του συγκεκριμένου ρομπότ. Αν επιστρέψει 2, τότε καλείτε η συνάρτηση ΕπέστρεψεΤιςΣυγκεντρώσεις με είσοδο τις συνιστώσες του συγκεκριμένου ρομπότ και του δεξιού του γείτονα πάνω στο κυρτό περίβλημα και με τις συγκεντρώσεις που θα επιστρέψει βρες αυτή που περιλαμβάνει το συγκριμένο ρομπότ. Δεδομένου των δύο ακραίων ρομπότ αυτής της συγκέντρωσης, δημιουργούμε την ευθεία των δύο αυτών σημείων και έχοντας την κλίση της ευθείας τους δημιουργούμε την παράλληλη γραμμή που περνά από το σημείο του κέντρου του συγκεκριμένου ρομπότ. Η εξίσωση της παράλληλης προκύπτει από την κλίση της άλλης ευθείας, αφού είναι παράλληλες τότε έχουν την ίδια κλίση, και λύνουμε την εξίσωση με βάση τις συνιστώσες του συγκεκριμένου ρομπότ. Στην συνέχεια δημιουργούμε την κάθετη γραμμή που περνά από το συγκεκριμένο ρομπότ με την παράλληλη ευθεία που περνά από το ρομπότ. Αυτή προκύπτει από την αντιστρόφως ανάλογη κλίση της παράλληλης και την λύση της εξίσωσης με τις συνιστώσες του συγκεκριμένου ρομπότ. Βρίσκω το σημείο που τέμνει την κάθετη με τον άξονα x, όπως περιγράφηκε πιο πάνω και με τον ίδιο τρόπο όπως και πριν βρίσκω την γωνία της ευθείας με τον άξονα x. Έχοντας την γωνία και την απόσταση που θέλω να είναι το νέο σημείο, το υπολογίζω και το ορίζω ως τις νέες συνιστώσες που θα μετακινηθεί το συγκεκριμένο ρομπότ. Αλλιώς, δηλαδή η συνάρτηση ΠόσηΑπόστασηΑπέχει επέστεψε 3, επιστρέφει τις ήδη υπάρχουσες συνιστώσες του ρομπότ, δηλαδή δεν θα μετακινηθεί. Αν η προηγούμενη συνάρτηση, δηλαδή η συνάρτηση ΕίσαιΣτηνΜικρότερηΣυγκέντρωση, επιστέψει 3, τότε επιστρέφει τις υπάρχουσες συνιστώσες και το ρομπότ δεν μετακινείται.

3. 2.2.6 Διαδικασία ΌχιΠλήρηςΟρατότητα

Η διαδικασία αυτή εκτελείται όταν το συγκεκριμένο ρομπότ που την καλεί είναι πάνω στο κυρτό περίβλημα και από το σύνολο των ρομπότ που βλέπει δεν είναι όλα πάνω.

Ψευδοκώδικας:

- Έχουμε μια τριάδα γειτονικών ρομπότ, cl-cr-cm που βρίσκονται πάνω στο κυρτό περίβλημα.
- Ζωγράφισε ένα ορθογώνιο με βάση τα δύο ακρινά ρομπότ από την τριάδα, cl και cr.
- Θεώρησε και τις τρείς περιπτώσεις θέσεων του ρομπότ μας.
- Αν σε κάποια από τις τρείς περιπτώσεις το μεσαίο σημείο είναι εντός του ορθογωνίου, τότε μετακινήσου στην κατάσταση ΣεΕυθεία, διαφορετικά, στην κατάσταση ΌχιΣεΕυθεία.

Αρχικά έχω τρείς περιπτώσεις που πρέπει να ελέγξω. Έχοντας μια τριάδα από γειτονικά ρομπότ που και τα τρία βρίσκονται πάνω στο κυρτό περίβλημα, σε κάθε περίπτωση το συγκεκριμένο ρομπότ βρίσκεται στην μεσαία, αριστερή ή δεξιά θέση. Βρίσκω την ευθεία (clcr) που περνά από το αριστερό και δεξί ρομπότ, με την βοήθεια της συνάρτησης ΕύρεσηΕυθείας, η οποία με είσοδο τις συνιστώσες των δύο σημείων της ευθείας επιστρέφει το a και b της εξίσωσης την ευθείας τους. Γνωρίζοντας την κλίση της ευθείας clcr, μπορώ να βρω την κάθετη που περνά από το σημείο cl, της οποίας η κλίση θα είναι αντιστρόφως ανάλογη της κλίσης της ευθείας clcr. Θα έχουμε μια ευθεία από το cl μέχρι το σημείο A, προς τα πάνω, που έχει απόσταση 1/n και την ευθεία από το cl στο B, προς τα κάτω, με ίδια απόσταση. Το ίδιο ισχύει και για το σημείο cr, θα σχηματιστεί μια ευθεία CD. Για όλες τις περιπτώσεις των θέσεων που μπορεί να έχει το συγκεκριμένο ρομπότ στην τριάδα γειτονικών ρομπότ, ελέγχω αν βρίσκεται εντός του ορθογωνίου που σχηματίζεται από τις δύο ευθείες, δηλαδή το ABCD. Για την εύρεση αν το σημείο είναι εντός του παραλληλόγραμμου έχω υλοποιήσει μια συνάρτηση ΠάνωΣτοΠολύγωνο, η οποία υπολογίζει μαθηματικά αν το σημείο βρίσκεται εντός του πολύγωνου. Παίρνοντας σαν είσοδο το σύνολο των ακμών του πολυγώνου, τις συνιστώσες του ρομπότ που θέλουμε να βρούμε αν είναι εντός και τον αριθμό των ακμών, ελέγχουμε για κάθε ακμή του πολύγωνου αν οι δύο συνιστώσες είναι μικρότερες από αυτές. Αν ισχύει αυτό τότε είναι μέσα στο ορθογώνιο, αλλιώς όχι. Αν οποιαδήποτε από τις τρείς περιπτώσεις θέσεων του ρομπότ είναι εντός του ορθογωνίου τότε μεταβαίνει στην κατάσταση 10, διαδικασία ΣεΕυθεία, αλλιώς στην κατάσταση 7, διαδικασία ΔενΕίναιΣεΕυθεία.

3. 2.2.7 Διαδικασία ΔενΕίναιΣεΕυθεία

Η διαδικασία αυτή καλείται όταν το ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα αλλά δεν έχει πλήρη ορατότητα. Με λίγα λόγια το ρομπότ αυτό δεν βλέπει όλα τα ρομπότ και αυτό δεν οφείλετε στο γεγονός ότι είναι σε ευθεία με κάποιο άλλο ώστε αυτό να του κόβει την ορατότητα, κρύβοντας κάποια από τα ρομπότ στο χώρο.

Ο αλγόριθμος αρχικά ελέγχει αν πάνω στο κυρτό περίβλημα υπάρχουν όλα τα ρομπότ, αν ισχύει αυτό τότε μεταβαίνει στην κατάσταση 8, διαδικασία ΥπάρχειΧώροςΓιαΆλλους. Αν δεν ισχύει αυτό προχωράει, εδώ ελέγχει αν το ρομπότ βλέπει όλα τα ρομπότ, αν ισχύει αυτό, τότε ελέγχει αν στο κυρτό περίβλημα υπάρχει πλευρά που έχει μήκος τουλάχιστον 2. Αυτό υλοποιήθηκε με μια επανάληψη, όπου για κάθε ρομπότ που βλέπει το συγκεκριμένο ρομπότ ελέγχει την απόσταση που έχει με τον γείτονα του, εδώ χρησιμοποιήθηκε η βοηθητική συνάρτηση ΑπόστασηΑποΔύοΣημεία, στην οποία γίνεται χρήση της ευκλείδειας απόστασης. Αν βρει μια πλευρά που επαληθεύει την συνθήκη, τότε το ρομπότ μεταβαίνει στην κατάσταση 8, διαδικασία ΥπάρχειΧώροςΓιαΆλλους. Αλλιώς μεταβαίνει στην κατάσταση 9, διαδικασία ΔενΥπάρχειΧώροςΓιαΆλλους. Στην περίπτωση που το ρομπότ δεν έχει πλήρη ορατότητα τότε για κάθε ρομπότ που ανήκει στα ρομπότ που βλέπει και είναι πάνω στο κυρτό περίβλημα το αντιγράφει σε άλλη αναπαράσταση του κυρτού περιβλήματος. Αυτό προγραμματιστικά έγινε με την δημιουργία ενός όμοιου πίνακα με τον αρχικό και την αντιγραφεί όλων των στοιχείων στον νέο πίνακα. Στην συνέχεια, για κάθε ρομπότ που ανήκει στο οπτικό του πεδίο αλλά δεν είναι πάνω στο κυρτό περίβλημα ζωγραφίζουμε την ευθεία από το ρομπότ μας μέχρι ένα σημείο χ, το οποίο είναι πάνω στο κυρτό περίβλημα και το ρομπότ ανήκει πάνω σε αυτή την ευθεία, βάζουμε το σημείο χ στο αντίγραφο του κυρτού περιβλήματος. Αυτό προγραμματιστικά έγινε, με τον έλεγχο στα ρομπότ που βλέπει το ρομπότ μας κα την εύρεση ποίων δεν ανήκουν πάνω στο κυρτό περίβλημα. Για το καθένα από αυτό, με την βοήθεια της επανάληψης, βρίσκω το γείτονα του σε πιο σημείο βρίσκεται πάνω στο κυρτό περίβλημα. Όταν τον βρω έχοντας τον γείτονα του και τον γείτονα του γείτονα του μεταξύ των οποίων θα μπει ανάμεσα, βρίσκω το σημείο που τέμνονται οι δύο ευθείες, των δυο γειτονικών ρομπότ πάνω στο κυρτό περίβλημα και του ρομπότ μας με του ρομπότ που υ

επεξεργαζόμαστε την συγκεκριμένη στιγμή. Για να προστεθεί αυτό το σημείο στο όμοιο πίνακα του κυρτού περιβλήματος απλά Μετακινούμαι τα σημεία μια θέση πάνω και προσθέτουμε το σημείο. Αφού το κάνουμε για όλα τα ρομπότ που βλέπει αλλά δεν είναι πάνω στο κυρτό περίβλημα τότε κάνουμε και πάλι τον έλεγχο για να βρούμε αν υπάρχει πλευρά που να είναι τουλάχιστον 2 για τον νέο πίνακα, με τον ίδιο τρόπο όπως και πιο πάνω. Αν βρούμε πλευρά που είναι τουλάχιστον 2 τότε το ρομπότ μεταβαίνει στην κατάσταση 8, διαδικασία ΥπάρχειΧώροςΓιαΆλλους, αλλιώς στην κατάσταση 9, διαδικασία ΔενΥπάρχειΧώροςΓιαΆλλους.

3. 2.2.8 Διαδικασία ΥπάρχειΧώροςΓιαΆλλους

Η διαδικασία αυτή καλείται όταν το συγκεκριμένο ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα, δεν έχει πλήρη ορατότητα και δεν βρίσκεται σε ευθεία με κάποιο άλλο ρομπότ ώστε να του περιορίζει την ορατότητα. Αυτή η διαδικασία με το τέλος της τερματίζει την φάση υπολογίζω και παίρνει το ρομπότ στην φάση μετακινούμαι, ώστε να μεταβεί στις νέες συνιστώσες που θα υπολογιστούν από αυτή την διαδικασία.

Ψευδοκώδικας:

- Αν το ρομπότ εφάπτεται με ένα άλλο ρομπότ, με το οποίο δεν είναι γείτονες πάνω στο κυρτό περίβλημα, τότε βρες την κάθετη γραμμή που περνά από το ρομπότ, από την ευθεία των δύο γειτονικών σου ρομπότ και βρές το σημείο στην κάθετη που έχει απόσταση 1/2n-ε. Επέστρεψε αυτό το σημείο.
- Διαφορετικά, επέστρεψε τις συνιστώσες που είχες.

Αν το συγκεκριμένο ρομπότ είναι εφαπτόμενο με κάποιο άλλο ρομπότ, το οποίο βρίσκεται πάνω στο κυρτό περίβλημα και δεν είναι γείτονες στο κυρτό περίβλημα τότε, βρίσκουμε τους δύο γείτονες του ρομπότ μας, αριστερά και δεξιά του στο κυρτό περίβλημα, και ζωγραφίζουμε μια ευθεία που περνά από το ρομπότ μας και είναι κάθετη στην ευθεία των γειτονικών ρομπότ και έχει μήκος 1/2n-ε. Το σημείο τερματισμού της ευθείας είναι το νέο σημείο στο οποίο θα μετακινηθεί το ρομπότ μας, αν δεν εφάπτεται με κανένα ρομπότ τότε επιστρέφει τις ήδη υπάρχουσες συνιστώσες του, άρα δεν θα μετακινηθεί. Προγραμματιστικά αυτό υλοποιήθηκε με μια επανάληψη

αρχικά για την εύρεση αν υπάρχει κάποιο ρομπότ πάνω στο κυρτό περίβλημα με το οποίο να εφάπτεται. Για την εύρεση αν εφάπτεται ή όχι, υλοποίησα μια συνάρτηση, Εφάπτονται, στην οποία υπολογίζεται με την τετραγωνική ρίζα της διαφοράς των δύο συνιστωσών του άξονα x και την τετραγωνική ρίζα της διαφοράς των δύο συνιστωσών του άξονα ψ, αν το αποτέλεσμα της πράξης αυτής είναι μηδέν τότε εφάπτονται αλλιώς όχι. Στην συνέχεια βρίσκουμε τους δυο γείτονες του, αφαιρώντας και προσθέτοντας ένα στην θέση που βρίσκεται το ρομπότ μας, κάνοντας τους απαραίτητους υπολογισμούς για τις ακραίες συνθήκες. Αφού βρούμε τους γείτονες, βρίσκουμε την κλίση της ευθείας των δύο ρομπότ και την κλίση της καθέτους τους, αφού είναι η αντιστρόφως ανάλογη της κλίσης τους. Με βάση την κλίση της καθέτου και το σημείο που περνά η κάθετος βρίσκουμε το σημείο στο οποίο τέμνει τον άξονα x, με την χρήση της βοηθητικής συνάρτησης ΈρευσηΣημείουΠουΤέμνειΤονΆξοναΧ, και στην συνέχεια την εύρεση της γωνίας μεταξύ της καθέτου και του άξονα x, με την χρήση της βοηθητικής συνάρτησης ΓωνίαΜεταξυΔύοΓραμμών. Έχοντας την γωνία και γνωρίζοντας την απόσταση του σημείου που ψάχνουμε από το ρομπότ μας, βρίσκουμε το σημείο αυτό, με τον πρόσθεση της συνιστώσας με τον πολλαπλασιασμό της απόστασης με το ημίτονο, για την x-συνιστώσα, και συνημίτονο για την y-συνιστώσα. Αυτό το σημείο που βρήκαμε αποτελεί της νέες συνιστώσες στις οποίες θα μετακινηθεί το ρομπότ μας, αφού με τον τερματισμό αυτής της διαδικασίας θα μεταβεί στην φάση Μετακινήσου. Αν από την αρχή δεν βρει ρομπότ με το οποίο να εφάπτονται τότε αφήνει τις ήδη υπάρχουσες συνιστώσες, δηλαδή δεν μετακινείται.

3. 2.2.9 Διαδικασία ΔενΥπάρχειΧώροςΓιαΆλλους

Η διαδικασία αυτή καλείται όταν το συγκεκριμένο ρομπότ είναι πάνω στο κυρτό περίβλημα, δεν έχει πλήρη ορατότητα και δεν βρίσκεται σε ευθεία με κάποιο άλλο ρομπότ ώστε να μην μπορεί να δει κάποια ρομπότ τα οποία καλύπτει το αυτό το ρομπότ. Με το τέλος αυτής της διαδικασίας, το ρομπότ έχει υπολογίσει τις νέες συνιστώσες στις οποίες θα μετακινηθεί και μεταβαίνει από την φάση Υπολογίζω στην φάση Μετακινούμαι.

Ψευδοκώδικας:

- Βρές το μεσαίο σημείο της ευθείας των δύο γειτονικών σου ρομπότ.
- Υπολόγισε το σημείο π, που βρίσκεται πάνω στην κάθετη ευθεία από την ευθεία των δυο γειτόνων σου και περνά από το μεσαίο σημείο και έχει απόσταση 1/2n-ε. Υπολόγισε και επέστρεψε το σημείο πάνω στην ευθεία p και το κέντρο σου, που μπορείς να μετακινηθείς στο μέγιστο χωρίς να αλλάξει το κυρτό περίβλημα.

Αρχικά, βρίσκουμε τον αριστερό και δεξιό γείτονα του ρομπότ μας πάνω στο κυρτό περίβλημα, αυτό γίνεται με ένα πέρασμα από όλα τα στοιχεία, αν χρειαστεί, για να βρούμε την θέση του ρομπότ μας στο κυρτό περίβλημα και μετά αφαιρώντας και προσθέτοντας ένα στην θέση του βρίσκουμε τους δύο γείτονες του, προσθέτουμε και τις ακραίες περιπτώσεις, δηλαδή το ρομπότ μας να είναι στην πρώτη θέση άρα ο αριστερός γείτονας είναι στην τελευταία θέση και την περίπτωση που βρίσκεται στην τελευταία θέση άρα ο δεξιός γείτονας είναι στην πρώτη θέση. Μετά, ζωγραφίζουμε την ευθεία που περνά από τους δύο γείτονες και βρίσκουμε το μεσαίο σημείο, υπολογίζοντας το με την πρόσθεση των δύο συνιστωσών των δυο ρομπότ, για κάθε άξονα, και διαιρώντας το δια 2. Στην συνέχεια, υπολογίζουμε την κάθετη, όπως περιγράψαμε και στην προηγούμενη διαδικασία, δηλαδή υπολογίζοντας την κλίση, την γωνία την ευθείας με τον άξονα x και την εύρεση της γωνιάς. Έχοντας υπολογίσει την γωνία και γνωρίζοντας το μήκος της βρίσκουμε το σημείο p. Για την εύρεση του τελικού σημείου μετακίνησης, το οποίο βρίσκεται ανάμεσα στο μεσαίο σημείο και το σημείο p, απλά στο μήκος της απόστασης πρόσθεσα μια μικρότερη απόσταση από αυτή που προσθέτει για την εύρεση του σημείου p. Αφού έχουμε τις νέες συνιστώσες του ρομπότ μας, η διαδικασία τερματίζει.

3. 2.2.10 Διαδικασία ΣεΕυθεία

Η διαδικασία αυτή καλείται από το συγκεκριμένο ρομπότ όταν είναι πάνω στο κυρτό περίβλημα και δεν έχει πλήρη ορατότητα. Ουσιαστικά εδώ γνωρίζουμε γιατί το συγκεκριμένο ρομπότ δεν έχει πλήρη ορατότητα, αφού βρίσκεται σε ευθεία με άλλο ρομπότ.

Ψευδοκώδικας:

- Οι ίδιες ρυθμίσεις όπως και στην κατάσταση ΌχιΠλήρηςΟρατότητα, με την μόνη περίπτωση το ρομπότ να βρίσκεται στην μεσαία θέση.
- Αν το μεσαίο βρίσκεται μέσα στο ορθογώνιο, τότε μετακινήσου στην κατάσταση ΒλέπωΔύοΡομπότ. Αλλιώς, μετακινήσου στην κατάσταση ΒλέπωΈναΡομπότ.

Σε αυτή την διαδικασία έχουμε να υλοποιήσουμε την μία περίπτωση από τις τρείς που υλοποιήθηκαν στην διαδικασία ΌχιΠλήρηΟρατότητα, την περίπτωση που το ρομπότ μας βρίσκεται στην μεσαία θέση. Βρίσκουμε τους δύο γείτονές του και όπως και στην άλλη διαδικασία δημιουργούμε το ορθογώνιο ΑΒΨΔ, η υλοποίηση και η εύρεση του ορθογωνίου ABCD γίνεται με τον ίδιο τρόπο όπως και στην προαναφερόμενη διαδικασία, και ελέγχουμε αν το ρομπότ μας βρίσκεται εντός του ορθογωνίου, αν αυτό ισχύει τότε το ρομπότ μεταβαίνει στην κατάσταση 12, διαδικασία ΒλέπωΔύοΡομπότ, αλλιώς μεταβαίνει στην κατάσταση 11, διαδικασία ΒλέπωΈναΡομπότ.

3. 2.2.11 Διαδικασία ΒλέπωΈναΡομπότ

Η διαδικασία αυτή καλείται όταν το συγκεκριμένο ρομπότ βρίσκεται πάνω στο κυρτό περίβλημα, δεν έχει πλήρη ορατότητα και είναι σε ευθεία με κάποιο άλλο ρομπότ, σε αυτό το γεγονός οφείλεται και η μη πλήρης ορατότητα του συγκεκριμένο υ ρομπότ. Αυτή η διαδικασία τερματίζει αυτή την φάση και το ρομπότ μεταβαίνει στην φάση μετακινούμαι.

Ψευδοκώδικας:

- Επέστρεψε τις συνιστώσες που ήδη έχεις.

Σε αυτή την διαδικασία ο μόνος υπολογισμός πο υ γίνεται είναι η επιστροφή των ήδη υπαρχουσών συνιστωσών του ρομπότ και η μεταβίβαση του από την φάση Υπολογίζω στην φάση Μετακινούμαι.

3. 2.2.12 Διαδικασία ΒλέπωΔύοΡομπότ

Η διαδικασία αυτή εκτελείται από το συγκεκριμένο ρομπότ όταν βρίσκεται πάνω στο κυρτό περίβλημα, δεν έχει πλήρη ορατότητα και αυτό οφείλεται στο γεγονός ότι βρίσκεται σε ευθεία με κάποιο άλλο ρομπότ. Όταν τερματίσει αυτή η διαδικασία τότε το ρομπότ πηγαίνει από την φάση Υπολογίζω στην φάση Μετακινούμαι, με τις νέες συνιστώσες που θα υπολογιστούν από την διαδικασία.

Η υλοποίηση αυτής της διαδικασία είναι παρόμοια με την υλοποίηση της διαδικασία ΌχιΠλήρηΟρατότητα, με την χρήση μόνο της περίπτωσης όπου το ρομπότ μας είναι το μεσαίο από μια τριάδα από γειτονικών ρομπότ. Στην συνέχεια, ζωγραφίζω την κάθετη, πάνω στην ευθεία των δύο ακρινών γειτονικών του ρομπότ, ευθεία που ξεκινά από το μεσαίο ρομπότ και τερματίζει σε απόσταση 1/2n-ε. Το σημείο στο οποίο τερματίζει η ευθεία αυτή, το ορίζω ως το σημείο π. Ο υπολογισμός του σημείου αυτού είναι ακριβώς ο ίδιος με την διαδικασία ΣεΕυθεία, δηλαδή βρίσκουμε την κλίση της ευθείας των δυο ακρινών ρομπότ, έτσι ξέρουμε και την κλίση της κάθετης, μετά βρίσκουμε που τέμνει η κάθετη τον άξονα x και με βάση αυτό το σημείο βρίσκουμε την γωνία που σχηματίζουν. Έχοντας όλα αυτά σαν δεδομένα υπολογίζουμε το σημείο p. Μετά, υπολογίζουμε το σημείο που τέμνονται οι δύο ευθείες, η κάθετη με την ευθεία των δύο γειτόνων, με την χρήση της βοηθητικής συνάρτησης ΕύρεσηΣημείουΠουΤέμνονταιΔύοΕυθείες, η οποία με μαθηματικούς υπολογισμούς υπολογίζει αυτό το σημείο. Έχοντας αυτό το σημείο , βρίσκουμε την γωνία που σχηματίζει η γωνία μεταξύ της ευθείας των δυο γειτονικών ρομπότ με την ευθεία του ρομπότ μας με το καινούργιο σημείο που υπολογίσαμε. Με βάση την γωνία και γνωρίζοντας την απόσταση που θέλουμε να είναι το νέο μας σημείο, απόσταση ίση με 1/n, βρίσκουμε αυτό το σημείο και το ονομάζουμε σημείο p΄. Μετέπειτα, υπολογίζουμε τις αποστάσεις μεταξύ των δύο νέων σημείων, p και p΄, με τις συνιστώσες του ρομπότ μας και ελέγχουμε ποια είναι η πιο μικρή. Το σημείο που βρίσκεται πιο κοντά στο ρομπότ μας είναι και το σημείο το οποίο θα επιστραφεί ως οι νέες συνιστώσες του ρομπότ.

3. 2.2.13 Διαδικασία ΌχιΠάνωΣτοΚυρτόΠερίβλημα

Η διαδικασία αυτή καλείται όταν το ρομπότ δεν βρίσκεται πάνω στο κυρτό περίβλημα. Σε αυτή την διαδικασία θα ελεγχθεί κατά πόσο το συγκεκριμένο ρομπότ εφάπτεται με κάποιο άλλο ρομπότ.

Ψευδοκώδικας:

- Έλεγξε αν υπάρχει ρομπότ που εφάπτεται μαζί σου.
- Αν υπάρχει, μετακινήσου στην κατάσταση Κολλημένα.
- Αλλιώς, μετακινήσου στην κατάσταση ΌχιΚολλημένα.

Για την υλοποίηση αυτής της διαδικασία ελέγχουμε αρχικά αν υπάρχει κάποιο ρομπότ από το οπτικό πεδίο του ρομπότ μας με το οποίο να εφάπτονται. Αυτό υλοποιήθηκε με μια επανάληψη, η οποία περνά όλα τα ρομπότ που βλέπει το ρομπότ μας και καλεί την βοηθητική συνάρτηση Εφάπτονται με είσοδο τις δυο συνιστώσες των δυο ρομπότ. Αν η συνάρτηση αυτή επιστρέψει ότι εφάπτεται με κάποιο ρομπότ, τότε το ρομπότ μεταβαίνει στην κατάσταση 14, διαδικασία Κολλημένο. Αλλιώς μεταβαίνει στην κατάσταση 13, διαδικασία ΌχιΚολλημένο.

3. 2.2.14 Διαδικασία Κολλημένο

Η διαδικασία αυτή καλείται από το συγκεκριμένο ρομπότ όταν αυτό δεν βρίσκεται πάνω στο κυρτό περίβλημα. Με το τέλος της διαδικασίας αυτής το ρομπότ μεταβαίνει από την φάση Υπολογίζω στην φάση Μετακινούμαι.

Η διαδικασία αυτή καλεί την γεωμετρική συνάρτηση ΒρεςΣημεία με είσοδο τα ρομπότ από το οπτικό πεδίου του συγκεκριμένο υ ρομπότ που βρίσκονται πάνω στο κυρτό περίβλημα. Αν η πιο πάνω συνάρτηση επιστρέψει τουλάχιστον 1 σημείο, τότε επιλέγω το κο ντινότερο σημείο από αυτά από το ρομπότ μου. Αυτό υλοποιήθηκε με την επανάληψη και τον έλεγχο των αποστάσεων των σημείων με το ρομπότ, με την βοήθεια της συνάρτησης ΕυρεσήΑπόστασηςΑποΔύοΣημεία, η οποία την υπολογίζει με βάση την ευκλείδεια απόσταση. Στην συνέχεια πρέπει να ελεγχθεί αν υπάρχουν ρομπότ τα

οποία είναι κολλημένα με το ρομπότ μας, τα οποία βρίσκονται σε πιο κοντινή απόσταση από το σημείο που βρήκαμε πιο πάνω. Για τον σκοπό αυτό, καλείται η βοηθητική συνάρτηση ΕύρεσηΚολλημένωνΣημείων, η οποία με είσοδο τα ρομπότ που βλέπει το ρομπότ μας επιστρέφει αυτά με τα οποία είναι κολλημένο. Μετά από αυτά, για κάθε ένα από αυτά τα ρομπότ ελέγχει την απόσταση με το ρομπότ μας, βρίσκοντας την πιο μικρή. Με βάση τις δυο αποστάσεις που έχουμε, βρίσκουμε ποια είναι η πιο μικρή. Αν η πιο μικρή είναι από το ρομπότ με το οποίο είναι κολλημένο, τότε επιστρέφει τις ήδη υπάρχουσες συνιστώσες, δηλαδή δεν θα μετακινηθεί. Αλλιώς, αν υπάρχουνε ρομπότ από αυτά που είναι κολλημένα που έχουν την ίδια απόσταση με το αρχικό σημείο που βρήκαμε, τότε ελέγχουμε αν το ρομπότ μας είναι το δεξιότερο από το σύνολο των κολλημένων ρομπότ, με την βοήθεια της συνάρτησης ΕύρεσηςΔεξιότερουΚολλημένουΣημείου. Αν είναι το δεξιότερο ρομπότ του συνόλου, τότε βρίσκουμε τους δύο του γείτονες και ελέγχουμε αν κάποιος από τους δύο είναι το σημείο που βρήκε η συνάρτηση πριν. Αν δεν είναι, τότε βρίσκει το σημείο στο οποίο τέμνονται οι δύο ευθείες, των δυο γειτόνων με την ευθεία του σημείου με το ρομπότ μας, και με μια μικρή απόσταση πιο πάνω από αυτό το σημείο τομής θα επιλεχτεί το νέο σημείο στο οποίο θα μετακινηθεί το ρομπότ μας. Αυτό με την μικρή απόσταση γίνεται για να μην μπει το ρομπότ σε ευθεία με τα γειτονικά του ρομπότ. Αν είναι ένα από αυτά, τότε απλά βρίσκουμε ένα σημείο λίγο πιο πάνω από το σημείο αυτό, και πάλι έχουμε μικρή απόσταση πιο πάνω για τον ίδιο λόγο όπως και πριν, στο οποίο θα μετακινηθεί το ρομπότ. Αλλιώς, αν δηλαδή δεν επιστρέψει σημεία η συνάρτηση, επιλέγονται οι δύο γείτονες του ρομπότ με απόσταση τουλάχιστον 2 από το αυτόν και είναι πάνω στο κυρτό περίβλημα. Αν δεν βρεθεί κάποιος γείτονας με την προαναφερόμενη απόσταση τότε επιστρέφει τις ήδη υπάρχουσες συνιστώσες. Οι δύο αυτοί γείτονες βρίσκονται με την βοήθεια της συνάρτησης ΕύρεσηΔύοΚοντινότερωνΣημείων, που βρίσκει τα δύο σημεία που είναι πιο κοντά στο ρομπότ μας. Μετά, ελέγχουμε αν από τα ρομπότ με τα οποία είναι κολλημένο το συγκεκριμένο ρομπότ έχουν πιο κοντινή απόσταση αυτά τα δύο ρομπότ από ότι έχουν με το ρομπότ μας. Αυτό υλοποιήθηκε με την εύρεση των ρομπότ με τα οποία είναι κολλημένο το ρομπότ μας, την βοηθητική συνάρτηση ΕύρεσηΚολλημένωνΣημείων, και μετά τον έλεγχο για καθένα από αυτά τις απόσταση που έχει με τα δύο σημεία με την απόσταση του ρομπότ μας με αυτά. Αν έχουν μικρότερη απόσταση από αυτή που έχει το συγκεκριμένο ρομπότ, τότε επιστρέφονται οι υφιστάμενες συνιστώσες και το ρομπότ

δεν μετακινείται. Αν όμως, ένα ή περισσότερα από τα κολλημένα ρομπότ έχουν ίδια απόσταση με την απόσταση του συγκεκριμένου ρομπότ με αυτά τα δύο σημεία, τότε ελέγχουμε αν το ρομπότ μας είναι το δεξιότερο από τα κολλημένα ρομπότ και αν ισχύει αυτό τότε βρίσκουμε το μεσαίο σημείο της ευθείας των δύο αρχικών γειτονικών ρομπότ, αυτό υπολογίζεται με την πρόσθεση των δυο συνιστωσών και την διαίρεση δια δύο. Αν δεν ισχύει αυτό, τότε επιστρέφονται οι ήδη υφιστάμενες συνιστώσες και το ρομπότ δεν μετακινείται. Αυτό υλοποιήθηκε, με την βοήθεια της συνάρτησης ΕύρεσηςΔεξιότερουΚολλημένουΣημείου, η οποία ελέγχει τις συντεταγμένες για τον άξονα x και ελέγχει αν το μεγαλύτερο χ το έχει το ρομπότ μας. Αλλιώς, αν δηλαδή μόνο το συγκεκριμένο ρομπότ έχει αυτή την απόσταση από τα δύο ρομπότ, τότε βρίσκουμε το μεσαίο σημείο της ευθείας των δύο αυτών ρομπότ και το επιστρέφουμε ως τις νέες συνιστώσες του ρομπότ μας.

3. 2.2.15 Διαδικασία ΌχιΚολλημένο

Η διαδικασία αυτή καλείται από το συγκεκριμένο ρομπότ όταν αυτό δεν βρίσκεται πάνω στο κυρτό περίβλημα.

Ψευδοκώδικας:

- Κάλεσε την συνάρτηση ΒρέςΣημεία με είσοδο τα σημεία πάνω στο κυρτό περίβλημα.
- Αν επιστρέψει έστω και ένα σημείο, τότε μετακινήσου στην κατάσταση ΔενΘαΠροκαλέσειΑλλαγή.
- Διαφορετικά, μετακινήσου στην κατάσταση ΘαΠροκαλέσειΑλλαγή.

Η διαδικασία αυτή αρχικά καλεί την γεωμετρική συνάρτηση ΒρέςΣημεία, η οποία περιγράφεται στο υποκεφάλαιο 3.1, με είσοδο τα ρομπότ που βρίσκονται στο οπτικό πεδίο του ρομπότ μας. Όταν η συνάρτηση επιστρέψει τα σημεία που βρήκε τότε ελέγχουμε πόσα σημεία επέστρεψε με ένα απλό έλεγχο. Αν επέστρεψε τουλάχιστον 1 σημείο τότε το ρομπότ μεταβαίνει στην κατάσταση 17, διαδικασία ΔενΘαΠροκαλέσειΑλλαγή. Αλλιώς το ρομπότ μεταβαίνει στην κατάσταση 16, διαδικασία ΘαΠροκαλέσειΑλλαγή.

3. 2.2.16 Διαδικασία ΘαΠροκαλέσειΑλλαγή

Η διαδικασία αυτή καλείται όταν το συγκεκριμένο ρομπότ δεν βρίσκεται πάνω στο κυρτό περίβλημα και δεν είναι κολλημένο με άλλο ρομπότ. Με το τέλος αυτής της διαδικασίας το ρομπότ μεταβαίνει από την φάση Υπολογίζω στην φάση Μετακινούμαι.

Ψευδοκώδικας:

- Αν δεν υπάρχουν γειτονικά ρομπότ πάνω στο κυρτό περίβλημα με τουλάχιστον απόσταση 2, τότε επέστρεψε τις συνιστώσες που ήδη έχεις.
- Αλλιώς, επέλεξε τους δύο πιο κοντινούς και βρες το κέντρο της ευθείας που σχηματίζουν.

Αρχικά βρίσουμε όλα τα ρομπότ πάνω στο κυρτό περίβλημα από το οπτικό πεδίο του ρομπότ μας που απέχουν απόσταση δύο από τον γείτονα τους. Αυτό υλοποιήθηκε με μια επανάληψη που ελέγχει για κάθε ρομπότ πάνω στο κυρτό περίβλημα αν με τον γείτονα του έχουν την προαναφερόμενη απόσταση με την βοήθεια της συνάρτησης ΕυρεσήΑπόστασηςΑποΔύοΣημεία, η οποία υλοποιεί την ευκλείδεια απόσταση. Αν δεν υπάρχουν γειτονικά ρομπότ που να αληθεύουν αυτή την συνθήκη τότε επιστρέφει τις ήδη υπάρχουσες συνιστώσες του ρομπότ μας, δηλαδή δεν θα μετακινηθεί. Αλλιώς, επιλέγουμε το ζεύγος που βρίσκεται πιο κοντά από το ρομπότ μας, δηλαδή με μια επανάληψη για τα ζεύγη που επιλέχθηκαν ελέγχουμε την απόσταση τους με το ρομπότ μας, με την βοήθεια της συνάρτησης ΕύρεσηΚοντινότερωνΣημείων, η οποία απλά περνά από κάθε ρομπότ στον πίνακα και ελέγχει την απόσταση του με το ρομπότ μας, επιλέγεται αυτό που έχει την μικρότερη απόσταση από όλους. Έχοντας τα δυο κοντινότερα ρομπότ βρίσκουμε το μεσαίο σημείο της ευθείας τους, το οποίο υπολογίζουμε με το άθροισμα των δυο συνιστωσών, για κάθε άξονα ξεχωριστά, και διαιρούμαι δια δύο. Αυτό το σημείο που θα βρούμε είναι και οι νέες συνιστώσες που θα μετακινηθεί το ρομπότ με το τέλος αυτής της διαδικασίας.

3. 2.2.17 Διαδικασία ΔενΘαΠροκαλέσειΑλλαγή

Η διαδικασία αυτή καλείται όταν το συγκεκριμένο ρομπότ δεν βρίσκεται πάνω στο κυρτό περίβλημα και δεν είναι κολλημένο με άλλο ρομπότ. Με το τέλος αυτής της διαδικασίας το ρομπότ μεταβαίνει από την φάση Υπολογίζω στην φάση Μετακινούμαι.

Ψευδοκώδικας:

- Κάλεσε την συνάρτηση ΒρέςΣημεία με είσοδο τα ρομπότ πάνω στο κυρτό περίβλημα.
- Επέλεξε το πιο κοντινό σημείο από αυτά που θα επιστρέψει η συνάρτηση.
- Βρες και επέστρεψε το σημείο πάνω στο κυρτό περίβλημα που είναι πάνω στην γραμμή μεταξύ εσένα και του κοντινού σημείου.

Αρχικά καλείται η γεωμετρική συνάρτηση ΒρέςΣημεία, η οποία περιγράφεται στο υποκεφάλαιο 3.1, με είσοδο τα ρομπότ που βρίσκονται πάνω στο κυρτό περίβλημα. Από τα σημεία που θα επιστραφούν επιλέγουμε το πιο κοντινό στις συντεταγμένες του ρομπότ μας. Αυτό υλοποιήθηκε με την βοήθεια της συνάρτησης ΕυρεσήΑπόστασηςΑποΔύοΣημεία το οποίο υπολογιζόταν σε κάθε επανάληψη πάνω στα σημεία που επέστρεψε η συνάρτηση. Στην συνέχεια, βρίσκουμε τους δύο γείτονες του πιο κοντινού σημείου και βρίσκουμε το μεσαίο σημείο της ευθείας τους. Αυτό το σημείο αποτελεί το σημείο στο οποίο αν μετακινηθεί το ρομπότ μας θα βρίσκεται σε ευθεία με τα δύο γειτονικά του ρομπότ, άρα το σημείο στο οποίο πρέπει να μετακινηθεί είναι λίγο πιο πάνω από αυτό το σημείο που μόλις βρήκαμε. Υπολογίζουμε αυτό το σημείο με μια μικρή απόσταση από το μεσαίο σημείο και έτσι έχουμε τις νέες συνιστώσες στις οποίες θα μετακινηθεί το ρομπότ μας με το τέλος της διαδικασίας.

Αυτό που ουσιαστικά έχουμε είναι ένα κατανεμημένο αλγόριθμο που αποτελείται από τις ανεξάρτητες εκτελέσεις του τοπικού αλγορίθμου από κάθε ρομπότ. εκτελείται σε ασύγχρονο περιβάλλον, όπου κάθε ρομπότ εκτελεί κάθε βήμα ανεξάρτητα με την κατάσταση που βρίσκονται τα άλλα ρομπότ. Αυτό επιτυγχάνεται με την χρήση των pthreads και του βήματος Περιμένων, που δημιουργούν ασυγχρονισμό ώστε κάθε ρομπότ να εκτελεί διαφορετικό βήμα στο πρόγραμμα.

Κεφάλαιο 4

Στιγμιότυπα Προγράμματος

Για την εκτέλεση του προγράμματος χρησιμοποιήθηκε υπολογιστής με λειτουργικό σύστημα Centos 6.3, με CPU Intel(R) Core(TM)2 Duo και με RAM 4GB.

4.1 Διαμόρφωση Διαδραστικού Περιβάλλοντος

Ο αριθμός των ρομπότ εισαγάγετε στο πρόγραμμα μέσα στα αρχεία επικεφαλίδας (.h αρχεία) όπου είναι δηλωμένη η σταθερά MAX που αντιπροσωπεύει τον συγκεκριμένο αριθμό. Όταν θέλουμε να αλλάξουμε αυτό τον αριθμό , τότε αλλάζουμε μόνο τις γραμμές που βρίσκεται η δήλωση της σταθεράς αυτής.

Οι αρχικές θέσεις των ρομπότ βρίσκονται με την χρήση της συνάρτησης rand της «math.h». Το όριο για τους αριθμούς των δυο συνιστωσών που θα πρέπει να επιστρέψει τυχαία η συνάρτηση αυτή πρέπει να είναι αριθμοί μέχρι των αριθμό που αντιστοιχεί στα όρια του παραθύρου. Επειδή αυτή η συνάρτηση θέλουμε να μας επιστρέφει κάθε φορά και διαφορετικές τιμές ώστε να μην έχουμε σύγκρουση κάποιων ρομπότ, έγινε χρήση της συνάρτηση «srand(time(NULL))», η οποία για τον χρόνο πο υ της δίνεις σαν παράμετρο μόλις τον ξεπεράσει βρίσκει νέες τιμές η συνάρτηση rand.

4.2 Παράδειγμα Εκτέλεσης

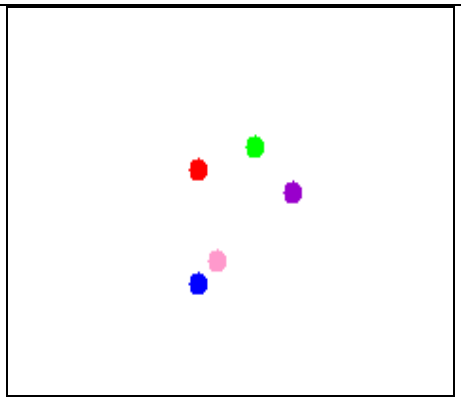
Σε αυτό το υποκεφάλαιο θα παρουσιαστούν στιγμιότυπα από την λειτουργία του προγράμματος, κάποια παραδείγματα, και τα διάφορα στάδια, από το αρχικό μέχρι το τελικό, που περνούν τα ρομπότ μέχρι να ολοκληρώσουν και να τερματίσουν όλα.

Αρχικά το πρόγραμμα ξεκινά με το πάτημα του πλήκτρου “S” και μόλις τελειώσει το πρόγραμμα βγαίνει μήνυμα ώστε ο χρήστης να πατήσει το “Q” και να τερματίσει το πρόγραμμα. Για διευκόλυνση της αναγνώρισης κάθε ρομπότ έχει προστεθεί διαφορετικό χρώμα στο καθένα.

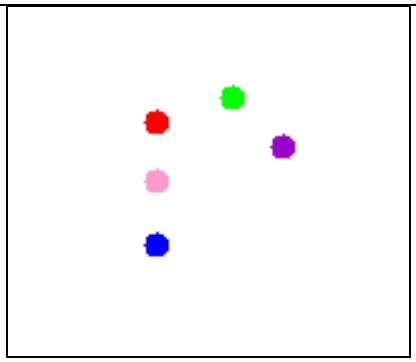
Εικόνα 1 – Αρχική εικόνα προγράμματος

Η εικόνα 1 αποτελεί την εικόνα της εισόδου στο πρόγραμμα, στην οποία ζητείται από τον χρήστη να πατήσει το πλήκτρο “S” για να ξεκινήσει το πρόγραμμα την λειτουργία του. Με το πάτημα αυτού το υ πλήκτρου, στην οθόνη θα εμφανιστή η αρχική κατάσταση του σεναρίου που θα τρέξει.

Πρώτο Παράδειγμα εκτέλεσης αλγορίθμου:



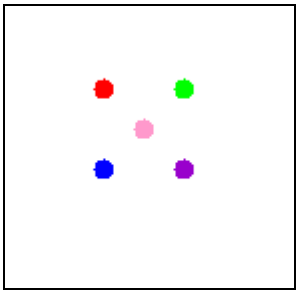
Εικόνα 4.2.1.α – Αρχική κατάσταση προγράμματος



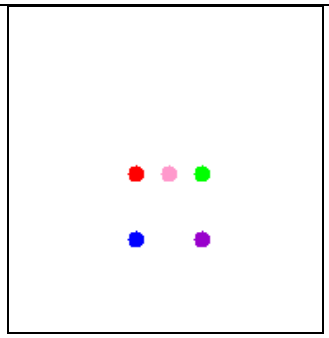
Εικόνα 4.2.1.β – Τελική κατάσταση προγράμματος για φάση 1

Η εικόνα 2.1 δείχνε την αρχική κατάσταση των ρομπότ στον χώρο και η εικόνα 2.2 δείχνει την τελική κατάσταση. Το ροζ ρομπότ εμποδίζει την πλήρη όραση του μπλε, έτσι με την μετακίνηση του στα δεξιά, δίνει την δυνατότητα για πλήρη ορατότητα σε όλα τα ρομπότ. Μπορεί στην εικόνα να φαίνεται ότι τα τρία ρομπότ, ροζ-μπλε-κόκκινο, είναι σε ευθεία αλλά δεν είναι και το ροζ δεν τους εμποδίζει να δει ο ένας τον άλλο, διότι με βάση τους υπολογισμούς το ροζ δεν βρίσκεται στην ευθεία που σχηματίζει το μπλε με το κόκκινο.

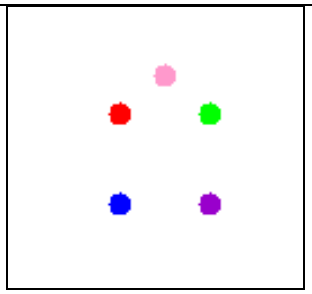
Δεύτερο Παράδειγμα εκτέλεσης αλγορίθμου:



Εικόνα 2.2.α – Αρχική κατάσταση



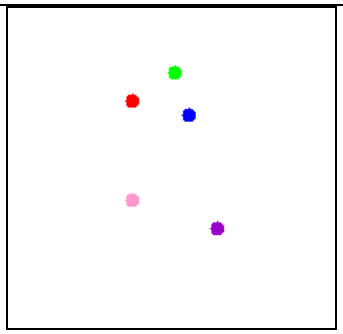
Εικόνα 2.2.β – Ενδιάμεση κατάσταση



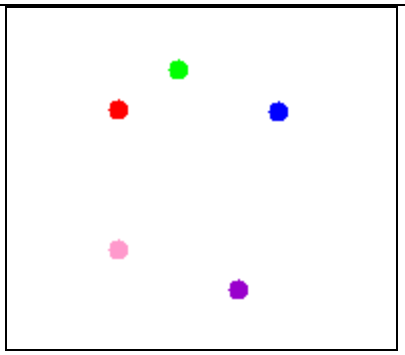
Εικόνα 2.2.γ – Τελική κατάσταση

Στο δεύτερο παράδειγμα μπορούμε να δούμε από την εικόνα 2 2.α την αρχική κατάσταση, από την οποία είναι εμφανές πως το μεσαίο, ροζ, ρομπότ εμποδίζει κάθε ακραίο ρομπότ από το να δει το ρομπότ ακριβώς διαγώνια του. Για τον σκοπό αυτό, και όπως φαίνεται στην εικόνα 2.2 β, το ροζ ρομπότ μετακινείται προς τα πάνω. Με την κίνηση του αυτή θέτει σε ευθεία γραμμή το κόκκινο και πράσινο ρομπότ. Με τους κατάλληλους υπολογισμούς κινείται ακόμη λίγο προς τα πάνω ώστε να μπορέσουν τα δύο προαναφερόμενα ρομπότ να δει ο ένας τον άλλο. Με αυτές τις θέσεις τα ρομπότ μπορούν δουν όλα τα ρομπότ όλα τα υπόλοιπα και όλα βρίσκονται πάνω στο κυρτό περίβλημα.

Τρίτο Παράδειγμα εκτέλεσης αλγορίθμου:



Εικόνα 2.3.α – Αρχική κατάσταση



Εικόνα 2.3.β – Τελική κατάσταση

Στο τρίτο παράδειγμα μπορούμε να δούμε ότι το μπλε ρομπότ δεν επιτρέπει στο πράσινο να δει το μωβ, έτσι με τους κατάλληλους υπολογισμούς μετακινείται προς τα πάνω και μπαίνει και αυτό με την σειρά του σε τερματική κατάσταση.

Κεφάλαιο 5

Συμπεράσματα

5.1 Συμπεράσματα

Το κυριότερο συμπέρασμα που βγήκε από αυτή την Ατομική Διπλωματική Εργασία είναι το γεγονός ότι ο αλγόριθμος συγκέντρωσης ρομπότ σχήματος δίσκων στο επίπεδο μπόρεσε να υλοποιηθεί σε πρόγραμμα και να δουλέψει.

Έχοντας τρέξει τον αλγόριθμο για διαφορετικό αριθμό ρομπότ σαν είσοδο κάθε φορά, έχω συμπεράνει ότι ο μέγιστος αριθμός ρομπότ που μπορεί να ανεχτεί το πρόγραμμα αυτό και να βγάλει σωστά αποτελέσματα είναι για 50 ρομπότ. Για μεγαλύτερο αριθμό από αυτό ο αλγόριθμος τρέχει αλλά επειδή έχει πολλά νήματα, αφού τα νήματα έχουν ίσο αριθμό με τον συνολικό αριθμό νημάτων, παρουσιάζεται πρόβλημα στους υπολογισμούς. Αυτό οφείλετε στο ότι τα νήματα μοιράζονται την ίδια μνήμη και αυτό πρέπει να έχει κάποιο συγχρονισμό, για τον λόγο αυτό έγινε και η πρόσθεση του mutex για κλείδωμα κάποιων κομματιών στον κώδικα. Αυτός ο συγχρονισμός που πρέπει να έχουν τα νήματα προκαλεί δύο μειονεκτήματα. Το ένα είναι να αργεί το πρόγραμμα στην λειτουργία του και αυτό θα οφείλετε στο γεγονός ότι ουσιαστικά δεν θα υπάρχει παραλληλισμός. Το άλλο μειονέκτημα είναι η δυσκολία στην υλοποίηση του.

Σε συνέχεια με το προηγούμενο συμπέρασμα προκύπτει και το ακόλουθο. Ο χρόνος εκτέλεσης και ολοκλήρωσης του προγράμματος οφείλεται σε μεγάλο βαθμό από την μέγιστη χρονική διάρκεια που μπορεί να περιμένει ένα ρομπότ στην φάση Περιμένω

και από τον συγχρονισμό των νημάτων. Έχω παρατηρήσει ότι, για μεγάλο αριθμό ρομπότ στο πρόγραμμα πρέπει ο μέγιστος αυτός χρόνος να μην είναι πολύ μικρός καθώς θα υπάρχει σενάριο όπου τα ρομπότ θα τρέχουν συγχρονισμένα, αλλά ούτε πολύ μεγάλος γιατί τότε θα υπάρχει μεγάλη καθυστέρηση στο πρόγραμμα. Αυτό που ισχύει είναι ότι για μικρό αριθμό ρομπότ ο μέγιστος χρόνος δεν πρέπει να ξεπερνά τα 15 δευτερόλεπτα γιατί μετά αργεί το πρόγραμμα χωρίς να υπάρχει λόγος.

5.2 Προβλήματα και Πώς Αντιμετωπίστηκαν

Αποσφαλμάτωση τελικού προγράμματος: ένα από τα κυριότερα προβλήματα που χρειάστηκε να επιλύσω ήταν η αποσφαλμάτωση του τελικού προγράμματος, δηλαδή με την εισαγωγή των νημάτων για τον παραλληλισμό. Με την χρήση του εργαλείου gdb για αποσφαλμάτωση του προγράμματος ήταν δύσκολο να ασχοληθείς συγκεκριμένα με ένα νήμα, επειδή κατά την διάρκεια εκτέλεσης του προγράμματος, αν ένα άλλο νήμα έμπαινε στο σημείο διακοπής για έλεγχο, τότε άλλαζε το νήμα και έπρεπε να μεταφερθείς στο νήμα στο οποίο θέλεις. Αυτό προκαλούσε μεγάλη σύγχυση και δυσκολία στην εύρεση σφάλματος. Για την αντιμετώπιση αυτού του προβλήματος δούλευα σε ξεχωριστό αρχείο, όσο μπορούσα και ανάλογα με το σφάλμα που είχα να επιλύσω, ώστε να μην επηρεάζεται με τα νήματα και να μπορώ να χρησιμοποιήσω το εργαλείο αποσφαλμάτωσης. Αυτό όμως δυσκόλεψε κατά πολύ την εύρεση σφάλματος για την συνολική εικόνα του προγράμματος διότι στο περιβάλλον χωρίς παραλληλισμό έλειπε η αντίδραση των άλλων ρομπότ και δεν υπήρχαν τα ανάλογα ερεθίσματα ώστε το συγκεκριμένο ρομπότ να προχωρήσει στην σωστή κατάσταση.

Συγχρονισμός νημάτων: ένα ακόμη πρόβλημα που χρειάστηκε να αντιμετωπίσω ήταν ο ταυτοχρονισμός της ανάγνωσης και της γραφής των συντεταγμένων των ρομπότ, καθώς στην φάση Βλέπω το ρομπότ χρειαζόταν να διαβάσει τις συντεταγμένες όλων των άλλων ρομπότ. Κατά την εκτέλεση του προγράμματος εντοπίστηκε ότι υπήρχε περίπτωση όπου ένα ρομπότ βρισκόταν στην φάση Μετακινούμαι και άλλαζε τις συντεταγμένες του, κάποιο άλλο ρομπότ να βρισκόταν στην φάση Βλέπω και να διάβαζε αυτές τις συντεταγμένες. Αυτό που συνέβαινε ήταν είτε να έπαιρνε τις παλιές συντεταγμένες, είτε τις καινούργιες συντεταγμένες, είτε να έπαιρνε μια την καινούργια και την άλλη την παλιά. Για τον σκοπό αυτό προστέθηκαν τα locks από τα pthreads για

να κλειδώνουν την συγκεκριμένη εντολή και να την ξεκλειδώσουν μόλις γινόταν η αλλαγή. Έτσι δεν υπήρχε περίπτωση να έπαιρνε ενδιάμεσα την αλλαγή, και πάντα θα είχε τις καινούργιες είτε τις παλιές, και στις δύο περιπτώσεις αυτό ήταν σωστό.

5.3 Μελλοντικές επεκτάσεις

Στο μέλλον, θα μπορούσε να βελτιωθεί το γραφικό περιβάλλον του προγράμματος. Δηλαδή να δοθεί περισσότερη έμφαση στα γραφικά και να γίνει πιο ευπαρουσίαστη η εμφάνιση του. Αυτό μπορεί να επιτευχθεί με την αντικατάσταση των σημείων, τα οποία αναπαριστούν τα ρομπότ, με ένα πιο ρεαλιστικό σύμβολο, ακόμη και ρομπότ. Καθώς και με την δημιουργία κουμπιών, για να γίνεται έναρξη και τερματισμός. Ακόμη με την βοήθεια των γραφικών μπορεί να δοθεί η επιλογή στον χρήστη είτε να γίνεται αυτόματα από το σύστημα ή να γίνεται χειρονακτικά, δηλαδή να τοποθετεί ο χρήστης με την βοήθεια του ποντικιού τα ρομπότ στις διάφορες θέσεις που επιθυμεί.

Επίσης, θα μπορούσε να εξελιχθεί σε μορφή παιχνιδιού με την εμπλοκή του χρήστη. Δηλαδή, θα μπορούσε να κάνει ο χρήστης την εισαγωγή των σημείων-ρομπότ στα σημεία που θέλει και στην συνέχεια να ξεκινά τον αλγόριθμο και να βλέπει το πρόγραμμα να εκτελείται μέχρι να ολοκληρωθεί. Ο χρήστης θα μπορούσε να δίνει σαν είσοδο και την ακτίνα που θέλει να έχουν τα ρομπότ καθώς και τον μέγιστο αριθμό που θα υπάρχουν στο πρόγραμμα κάθε φορά.

Βιβλιογραφία

[1] Χρυσοβαλάντης Αγαθαγγέλου (2010). Αλγόριθμος για τη Συσσώρευση Ρομπότ Σχήματος Δίσκων στο Επίπεδο, Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

[2] H. Ando, Y. Oasa, I. Suzuki ___ M. Yamashita, "Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility", IEEE Transactions on Robotics and Automation, Τόμος 15, Τεύχος 5, σελ. 818-828, 1999.

[3] N. Agmon και D. Peleg, "Fault Tolerant Gathering Algorithms for Autonomous Mobile Robots", Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, Ιανουάριος 2003.

[4] P. Flocchini, G. Prencipe, N. Santoro ___ P. Widmayer, "Gathering of Asynchronous Robots With Limited Visibility", *Theoretical Computer Science*, Τόμος 337, Τεύχη 1-3, σελ. 147-168, 2005.

[5] S. Souissi, X. Défago ___ M. Yamashita, "Gathering Asynchronous Mobile Robots with Inaccurate Compasses", *Proceding of the 10th International Conference on Principles of Distributed Systems*, Lecture Notes in Computer Science, Τόμος 4305, σελ. 333-349, Δεκέμβριος 2006.

[6] Chrysovalandis Agathangelou, Chryssis Georgiou and Marios Mavronicolas, A Distributed Algorithm for Gathering Many Fat Mobile Robots in the Plane, in Proc. of the 32nd ACM Symposium on Principles of Distributed Computing (PODC 2013), accepted, Montreal, Canada, 2013. [pdf]

[7] Βοήθημα για την βιβλιοθήκη γραφικών gp142 από το μάθημα ΕΠΛ131 – Αρχές Προγραμματισμού Ι (χειμερινό εξάμηνο 2009), «graphicserg6.pdf».

[8] Ιστοσελίδα που περιγράφει τις συναρτήσεις του pthreads και περιέχει και παράδειγμα. http://timmurphy.org/2010/05/04/pthreads-in-c-a-minimal-working-example/

[9] Βοήθημα για τα pthreads από το φροντιστήριο του μαθήματος ΕΠΛ372 – Παράλληλη Επεξεργασία του Κωνσταντίνου Χριστοφή, «pthreads.pdf».

[10] Μια ιστοσελίδα που περιγράφει τον αλγόριθμο του Graham για την εύρεση των σημείων ενός κυρτού περιβλήματος. (Graham Scan Algorithm) http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/ConvexHull/GrahamScan/grahamScan.htm

[11] Από την ιστοσελίδα της Βικιπαίδεια για τον αλγόριθμο του Graham για την εύρεση των σημείων ενός κυρτού περιβλήματος. (Graham Scan Algorithm) http://en.wikipedia.org/wiki/Graham_scan

[12] Για την χρήση της συνάρτησης sleep, για τον ασυγχρονισμό στις κινήσεις των ρομπότ. http://pubs.opengroup.org/onlinepubs/009695399/functions/sleep.html

[13] Ιστοσελίδα για την συνάρτηση sleep. http://www.gnu.org/software/libc/manual/html_node/Sleeping.html

[14] Ιστοσελίδα με τρόπους για εύρεση της εξίσωσης μιας ευθείας. http://www.wikihow.com/Find-the-Equation-of-a-Line

[15] Ιστοσελίδα με τρόπο να βρεις το σημείο τομής δύο ευθειών. http://flassari.is/2008/11/line-line-intersection-in-cplusplus/

[16] Ιστοσελίδα για την εύρεση αν ένα στοιχείο βρίσκεται μέσα σε ένα πολύγωνο ή όχι. http://www.codeproject.com/Tips/84226/Is-a-Point-inside-a-Polygon

[17] Ιστοσελίδα για την εύρεση της γωνίας μεταξύ δύο ευθειών. http://stackoverflow.com/questions/3365171/calculating-the-angle-between-two-lines-without-having-to-calculate-the-slope

[18] Ιστοσελίδα για την λύση συστήματος κύκλου και ευθείας. http://www.analyzemath.com/CircleEq/circle_line_intersection.html

Παράρτημα Α

Στο παράρτημα αυτό θα παρουσιαστούν αναλυτικά ο κώδικας των γεωμετρικών συναρτήσεων του προγράμματος που περιγράφονται στο κεφάλαιο 3, συγκεκριμένα στο υποκεφάλαιο 3.1.

Α.1 Συνάρτηση ΕίμαιΠάνωΣτοΚυρτόΠερίβλημα

```
bool onConvexHull(float v[MAX][2],int num,float x,float y,float ch[MAX][2],int
&chNum){
	int i,pos;

	pos=findConvexHull(v,num,ch);
	for(i=0;i<=pos;i++)
		if(x==ch[i][0] && y==ch[i][1])
			return true;
	chNum=pos;
	return false;
}
```

Η συνάρτηση αυτή καλεί και την συνάρτηση ΒρέςΤοΚυρτόΠερίβλημα και αυτή με την σειρά της την συνάρτηση CCW, που βοηθά στην εύρεση του κυρτού περιβλήματος.

Α.1.1 Συνάρτηση ΒρέςΤοΚυρτόΠερίβλημα

```
	int findConvexHull(float v[MAX][2],int numV,float ch[MAX][2]){
		int i,j=0,k=0;
		int min[MAX];
		min[j]=0;
		for(i=1;i<numV;i++){
			if(v[i][1]<v[min[j]][1])
				min[j]=i;
			else if(v[i][1]==v[min[j]][1]){
				j++;
				min[j]=i;
			}
		}
		//exo perisotera apo 1 simio me to idio mikrotero y
		if(j>0){
			k=1;
			int max=min[0];
			while(j>0){
				if(v[max][0]<v[min[j]][0])
					max=j;
				j--;
			}
		}

		int pos,pos1,pos0=min[0];
```

```
float n;
float newt[MAX+1][2];
i=0;
while(i<numV+1){
	if(i==0){
		newt[i][0]=v[pos0][0];
		newt[i][1]=v[pos0][1];
		i++;
	}
	else{
		pos=pos0+i;
		pos1=pos+1;

		if(pos==numV){
			pos=numV-(pos0+i);
			pos1=pos+1;
		}
		else if(pos1==numV){
			pos1=numV-(pos+1);
		}

		n=ccw(pos0,pos,pos1,v);

		if(n==0.0){
			if(v[pos][0]<v[pos1][0]){
				newt[i][0]=v[pos][0];
				newt[i][1]=v[pos][1];
				i++;
				newt[i][0]=v[pos1][0];
				newt[i][1]=v[pos1][1];
			}
			else{
				newt[i][0]=v[pos1][0];
				newt[i][1]=v[pos1][1];
				i++;
				newt[i][0]=v[pos][0];
				newt[i][1]=v[pos][1];
				i++;
			}
		}
		else if(n>0.0){ //left
			newt[i][0]=v[pos][0];
			newt[i][1]=v[pos][1];
			i++;
			newt[i][0]=v[pos1][0];
			newt[i][1]=v[pos1][1];
		}
		else{ //right
			newt[i][0]=v[pos1][0];
			newt[i][1]=v[pos1][1];
			i++;
			newt[i][0]=v[pos][0];
			newt[i][1]=v[pos][1];
			i++;
		}
	}
}
//newt[i][0]=v[pos0][0];
//newt[i][1]=v[pos0][1];
```

```
	stack S;
	S.length=0;

	Push(newt[0],&S);
	Push(newt[1],&S);

	i=2;
	while(i<=numV){
		n=cww1(S.list[S.length-2],S.list[S.length-1],newt[i],newt);
		if(n==0.0){ //collinear
			Push(newt[i],&S);
		}
		else if(n>0.0){ //left
			Push(newt[i],&S);
		}
		else{  //right

			do{
				Pop(&S);
				n=cww1(S.list[S.length-2],S.list[S.length-
1],newt[i],newt);
				if(n==0.0){
					Push(newt[i],&S);
					break;
				}
			}while(n<0.0);
			Push(newt[i],&S);
		}
		i++;
	}


	int length=S.length;
	int h;
	for(h=0;h<length;h++){
		PopAndCopy(&S,ch,h);
	}
	return h;
}
```

A.1.2 Συνάρτηση CCW

```
float ccw(int pos0,int pos,int pos1,float v[MAX][2]){
	float n=0.0,x1=0.0,x2=0.0,x3=0.0,x4=0.0;
	x1= v[pos][0]-v[pos0][0];
	x2= v[pos1][1]-v[pos0][1];
	x3= v[pos][1]-v[pos0][1];
	x4= v[pos1][0]-v[pos0][0];

	n=(x1*x2)-(x3*x4);
	return n;

}
```

A.2 Συνάρτηση ΜετακινήσουΣτοΣημείο

```
float moveToPoint(float x1,float y1,float x2,float y2,int m,int t){
```

```
	float p[2],n[2],y,c1[2],c2[2];
	c1[0]=x1;	c1[1]=y1;
	c2[0]=x2;	c2[1]=y2;
	float a=findLine(c1,c2,0);
	float a1;
	if(a==-1)
		a1=0;
	else
		a1=-(1.0/a);
	float x=findPointIntersectsTheXaxis(a1,c2);
	float c3[2],c4[2];
	c3[0]=x;	c3[1]=0.0;
	c4[0]=0.0;	c4[1]=0.0;
	double angle=getAngleofTwoLines(c2,c3,c3,c4);

	if(c2[0]>0)
		p[0]=c2[0]-(1*cos(angle));
	else
		p[0]=c2[0]+(1*cos(angle));

	if(c2[1]>0)
		p[1]=c2[1]-(1*sin(angle));
	else
		p[1]=c2[1]+(1*sin(angle));

	angle=getAngleofTwoLines(p,c1,c3,c4);

	float c[2];
	if(c2[0]>0)
		c[0]=c2[0]-(((1/(2*m))-0.0002)*cos(angle));
	else
		c[0]=c2[0]+(((1/(2*m))-0.0002)*cos(angle));

	if(c2[1]>0)
		c[1]=c2[0]-(((1/(2*m))-0.0002)*sin(angle));
	else
		c[1]=c2[0]+(((1/(2*m))-0.0002)*sin(angle));

	float a3,b3,a4,b4,b5;
	a3=findLine(c,c1,0);
	if(a3==-1)
		a3=0;
	b3=findLine(c,c1,1);

	a4=c2[0];
	b4=c2[1];
	b5=b3-b4;

	float x4,x5,y4,y5;
	double sq=((2*a3*b5)-(2*a4))-(4*(1+pow(a3,2))*(pow(a4,2)+pow(b5,2)-
1));
	if(sq<0)
		sq=sq*(-1);

	x4=((-((2*a3*b5)-(2*a4)))+sqrt(sq))/(2*(1+pow(a3,2)));
	x5=(-((2*a3*b5)-(2*a4))-sqrt(sq))/(2*(1+pow(a3,2)));
	double sq1=1-pow(x4-a4,2);
	if(sq1<0)
		sq1=sq1*(-1);
	y4=sqrt(sq1)+b4;
	double sq2=1-pow(x5-a4,2);
```

```
		if(sq2<0)
			sq2=sq2*(-1);
		y5=sqrt(sq2)+b4;

		double din1,din2;
		float cd[2];
		cd[0]=x4;	cd[1]=y4;
		din1=findDistanceOfTwoPoints(cd, c2);
		cd[0]=x5;	cd[1]=y5;
		din2=findDistanceOfTwoPoints(cd, c2);

		if(din1<din2){
			n[0]=x4;
			n[1]=c2[1];
			return n[t];
		}
		else{
			n[0]=x5;
			n[1]=c2[1];
			return n[t];
		}
		return p[t];
	}
```

Α.3 Συνάρτηση ΒρέςΤαΣημεία

```
int findPoint(float ch[MAX][2],float points[MAX][2],int numCH){
	float m[2],p[2],t[2],pc[2];
	double d;
	int i=0,j,k,count=0;

	while(i<MAX){
		d= sqrt(pow(ch[i+1][0]-ch[i][0],2)+pow(ch[i+1][1]-ch[i][1],2));

		if(d>=2){
			m[0]=(ch[i+1][0]+ch[i][0])/2;
			m[1]=(ch[i+1][1]+ch[i][1])/2;

			float x,y,c1[2],c2[2],c3[2];
			c1[0]=ch[i+1][0];	c1[1]=ch[i+1][1];
			c2[0]=ch[i][0];		c2[1]=ch[i][1];
			float a=findLine(c1,c2,0);
			float a1=-(1.0/a);
			float x=findPointIntersectsTheXaxis(a1,m);
			c3[0]=x;		c3[1]=0.0;
			double angle=getAngleofTwoLines(c1,c2,m,c3);
			//ena pou einai to mikos tis aktinos tu robot
			p[0]=m[0]+(1*cos(angle));
			p[1]=m[1]+(1*sin(angle));

			if(i==0){
				t[0]=intersectionPoint(ch[i],ch[i+1],ch[numCH-
1],ch[i+2],0);
				t[1]=intersectionPoint(ch[i],ch[i+1],ch[numCH-
1],ch[i+2],1);
			}
			else{
				t[0]=intersectionPoint(ch[i],ch[i+1],ch[i-
1],ch[i+2],0);
```

```
                        t[1]=intersectionPoint(ch[i],ch[i+1],ch[i-
1],ch[i+2],1);
                }

                j=0;
                for(k=0;k<MAX;k++){
                        if(ch[k][1]>p[1])
                                j++;
                }

                if(i==0){
                        pc[0]=foundDistanceofPointFromLine(ch[numCH-1],t,p);
                        pc[1]=foundDistanceofPointFromLine(t,ch[i+1],p);
                }
                else{
                        pc[0]=foundDistanceofPointFromLine(ch[i-1],t,p);
                        pc[1]=foundDistanceofPointFromLine(t,ch[i+1],p);
                }
                if(j==0 || (pc[0]>= (1.0/MAX) && pc[1] >= (1.0/MAX))){
                        points[count][0]=p[0];
                        points[count][1]=p[1];
                        count++;
                }
                i++;
        }//if
    }//while

    return count-1;
}
```

Α.4 Συνάρτηση ΕπέστρεψεΤιςΣυγκεντρώσεις

```
int connectedComponent(float cc[MAX][5], float v[MAX][2], int limit, float
c[2],int count,int k,int *pos){
    int i=0,j=0,t0=0,t1=0,t2=0,t3=0,t4=0,a0=0,a1=0,a2=0,a3=0,a4=0;
    double
d0=0.0,d1=0.0,d2=0.0,d3=0.0,d4=0.0,space0=0.0,space1=0.0,space2=0.0,space3=0.0,sp
ace4=0.0;
    bool flag=false;
    float initial[2];
    initial[0]=c[0];
    initial[1]=c[1];

    a0=-1;
    for(i=0;i<limit;i++){
        if(i==0){
            d0=findDistanceOfTwoPoints(initial,v[i]);
            t0=i;
        }
        else{
            d0=findDistanceOfTwoPoints(v[i],v[i++]);
            t0=i;
        }

        if(d0>2){
            space0=d0;
            break;
        }
    }
```

```
if(space0<= (1/(2*limit))){
	a1=t0+1;
	for(j=a1;j<limit;j++){
		d1=findDistanceOfTwoPoints(v[j],v[j++]);
		t1=j;


		if(d1>2){
			space1=d1;
			break;
		}
	}


	if(space1 <= (1/(2*limit))){
		a2=t1+1;
		for(j=a1;j<limit;j++){
			d2=findDistanceOfTwoPoints(v[j],v[j++]);
			t2=j;

			if(d2>2){
				space2=d2;
				break;
			}
		}

		if(space2 <= (1/(2*limit))){
			a3=t2+1;
			for(j=a1;j<limit;j++){
				d3=findDistanceOfTwoPoints(v[j],v[j++]);
				t3=j;

				if(d3>2){
					space3=d3;
					break;
				}
			}

			cc[k][0]=v[a3][0];
			cc[k][1]=v[a3][1];
			cc[k][2]=v[t3][0];
			cc[k][3]=v[t3][1];
			cc[k][4]=t3-a3+1.0;
			k++;
			cc[k][0]=v[a2][0];
			cc[k][1]=v[a2][1];
			cc[k][2]=v[t2][0];
			cc[k][3]=v[t2][1];
			cc[k][4]=t2-a2+1.0;
			k++;

			a4=-1;
			for(j=limit-1;j>0;j--){
				d4=findDistanceOfTwoPoints(v[j],v[j--]);
				t4=j;

				if(d4>2){
					space4=d4;
					break;
				}
```

```
                        }

                        cc[k][0]=v[t4][0];
                        cc[k][1]=v[t4][1];
                        cc[k][2]=v[t0][0];
                        cc[k][3]=v[t0][1];
                        cc[k][4]=(limit-t4+1.0)+t0+2.0;
                        *pos=k;
                        k++;
                        flag=true;
                        cc[k][0]=v[a1][0];
                        cc[k][1]=v[a1][1];
                        cc[k][2]=v[t1][0];
                        cc[k][3]=v[t1][1];
                        cc[k][4]=t1-a1+1.0;
                        k++;

                        initial[0]=v[t3+1][0];
                        initial[1]=v[t3+1][1];
                        if(flag==true && count>0){
                               //afero tis epanalipsis
                               int l;
                               for(i=0;i<k;i++){
                                      for(j=0;j<k;j++){
                                             if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                    for(l=j;j<k;j++){

       cc[j][0]=cc[j+1][0];

       cc[j][1]=cc[j+1][1];

       cc[j][2]=cc[j+1][2];

       cc[j][3]=cc[j+1][3];

       cc[j][4]=cc[j+1][4];
                                                    }
                                                    k=k-1;
                                             }
                                      }
                               }
                               //kai termatizw
                               return k-1;
                        }
                        else{

       k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                               return k;
                        }
                        flag=false;

                } // telioni to if t space2
                else{
                        a3=-1; //apotelei to initial
                        for(j=limit-1;j>0;j--){
                               if(i==0){

       d3=findDistanceOfTwoPoints(initial,v[j]);
                                      t3=j;
```

```
                                        }
                                        else{
                                                d3=findDistanceOfTwoPoints(v[j],v[j--]);
                                                t3=j;
                                        }

                                        if(d3>2){
                                                space3=d3;
                                                break;
                                        }
                                }

                                if(space3 > (1/(2*limit))){
                                        cc[k][0]=v[t3][0];
                                        cc[k][1]=v[t3][1];
                                        cc[k][2]=v[t2][0];
                                        cc[k][3]=v[t2][1];
                                        cc[k][4]=(limit-t3+1.0)+(t2+2.0);
                                        flag=true;
                                        k++;

                                        initial[0]=v[t2+1][0];
                                        initial[1]=v[t2+1][1];
                                        if(flag==true && count>0){
                                                //afero tis epanalipsis
                                                int l;
                                                for(i=0;i<k;i++){
                                                        for(j=0;j<k;j++){
                                                                if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                                        for(l=j;j<k;j++){

        cc[j][0]=cc[j+1][0];

        cc[j][1]=cc[j+1][1];

        cc[j][2]=cc[j+1][2];

        cc[j][3]=cc[j+1][3];

        cc[j][4]=cc[j+1][4];
                                                                        }
                                                                        k=k-1;
                                                                }
                                                        }
                                                }
                                                //kai termatizw
                                                return k-1;
                                        }
                                        else{

        k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                                return k;
                                        }
                                        flag=false;
                                }// telioni to if t space3
                                else{
                                        cc[k][0]=v[a1][0];
                                        cc[k][1]=v[a1][1];
                                        cc[k][2]=v[t1][0];
```

```
                                        cc[k][3]=v[t1][1];
                                        cc[k][4]=t1-a1+1.0;
                                        k++;
                                        cc[k][0]=v[a2][0];
                                        cc[k][1]=v[a2][1];
                                        cc[k][2]=v[t2][0];
                                        cc[k][3]=v[t2][1];
                                        cc[k][4]=t2-a2+1.0;
                                        k++;

                                        a4=t3-1; //apotelei to initial
                                        for(j=a4;j>0;j--){

        d4=findDistanceOfTwoPoints(initial,v[j]);
                                                t4=j;

                                                if(d4>2){
                                                        space4=d4;
                                                        break;
                                                }
                                        }

                                        cc[k][0]=v[t4][0];
                                        cc[k][1]=v[t4][1];
                                        cc[k][2]=v[a4][0];
                                        cc[k][3]=v[a4][1];
                                        cc[k][4]=t4-a4+1.0;
                                        k++;
                                        cc[k][0]=v[t3][0];
                                        cc[k][1]=v[t3][1];
                                        cc[k][2]=v[t0][0];
                                        cc[k][3]=v[t0][1];
                                        cc[k][4]=(limit-t3+1.0)+t0+2.0;
                                        *pos=k;
                                        flag=true;
                                        k++;

                                        initial[0]=v[t2+1][0];
                                        initial[1]=v[t2+1][1];
                                        if(flag==true && count>0){
                                                //afero tis epanalipsis
                                                int l;
                                                for(i=0;i<k;i++){
                                                        for(j=0;j<k;j++){
                                                                if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                                        for(l=j;j<k;j++){

        cc[j][0]=cc[j+1][0];

        cc[j][1]=cc[j+1][1];

        cc[j][2]=cc[j+1][2];

        cc[j][3]=cc[j+1][3];

        cc[j][4]=cc[j+1][4];
                                                                        }
                                                                        k=k-1;
                                                                }
```

```
                                        }
                                }
                                //kai termatizw
                                return k-1;
                        }
                        else{

        k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                return k;
                        }
                        flag=false;
                }// telioni to else t space3
                flag=false;
        } // telioni to else t space2
        flag=false;
}// telioni to if t space1
else{
        a2=-1;
        for(i=limit-1;i>=0;i--){
                if(i==limit-1){
                        d2=findDistanceOfTwoPoints(initial,v[i]);
                        t2=i;
                }
                else{
                        d2=findDistanceOfTwoPoints(v[i],v[i--]);
                        t2=i;
                }

                if(d2>2){
                        space2=d2;
                        break;
                }
        }

        if(space2 > (1/(2*limit))){
                cc[k][0]=v[t2][0];
                cc[k][1]=v[t2][1];
                cc[k][2]=v[t1][0];
                cc[k][3]=v[t1][1];
                cc[k][4]=(t2-limit+1.0)+(t1+2.0);
                flag=true;
                k++;

                initial[0]=v[t1+1][0];
                initial[1]=v[t1+1][1];
                if(flag==true && count>0){
                        //afero tis epanalipsis
                        int l;
                        for(i=0;i<k;i++){
                                for(j=0;j<k;j++){
                                        if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                for(l=j;j<k;j++){

        cc[j][0]=cc[j+1][0];

        cc[j][1]=cc[j+1][1];

        cc[j][2]=cc[j+1][2];
```

```
	cc[j][3]=cc[j+1][3];

	cc[j][4]=cc[j+1][4];
                                                        }
                                                        k=k-1;
                                                }
                                        }
                                }
                                //kai termatizw
                                return k-1;
                        }
                        else{

	k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                return k;
                        }
                        flag=false;
                } //telioni to if t space2
                else{
                        a3=t2-1;
                        for(i=a3;i>=0;i--){
                                d3=findDistanceOfTwoPoints(v[i],v[i--]);
                                t3=i;

                        if(d3>2){
                                space3=d3;
                                break;
                        }
                        }

                        if(space3 > (1/(2*limit))){
                        cc[k][0]=v[t3][0];
                        cc[k][1]=v[t3][1];
                        cc[k][2]=v[t1][0];
                        cc[k][3]=v[t1][1];
                        cc[k][4]=(limit-t3+1.0)+t1+2.0;
                        flag=true;
                        k++;

                        initial[0]=v[t1+1][0];
                        initial[1]=v[t1+1][1];
                        if(flag==true && count>0){
                                //afero tis epanalipsis
                                int l;
                                for(i=0;i<k;i++){
                                        for(j=0;j<k;j++){
                                                if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                                for(l=j;j<k;j++){

	cc[j][0]=cc[j+1][0];

	cc[j][1]=cc[j+1][1];

	cc[j][2]=cc[j+1][2];

	cc[j][3]=cc[j+1][3];

	cc[j][4]=cc[j+1][4];
```

```
                                                }
                                                k=k-1;
                                          }
                                    }
                              }
                              //kai termatizw
                              return k-1;
                        }
                        else{

k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                              return k;
                        }
                        flag=false;
                  }//telioni to if t space3
                  else{
                        a4=t3-1;
                        for(i=a4;i>=0;i--){

d4=findDistanceOfTwoPoints(v[i],v[i--]);
                                    t4=i;

                              if(d4>2){
                                    space4=d4;
                                    break;
                              }
                        }

                        cc[k][0]=v[a1][0];
                        cc[k][1]=v[a1][1];
                        cc[k][2]=v[t1][0];
                        cc[k][3]=v[t1][1];
                        cc[k][4]=t1-a1+1.0;
                        k++;
                        cc[k][0]=v[t2][0];
                        cc[k][1]=v[t2][1];
                        cc[k][2]=v[t0][0];
                        cc[k][3]=v[t0][1];
                        cc[k][4]=(limit-t2+1.0)+t0+2.0;
                        *pos=k;
                        flag=true;
                        k++;
                        cc[k][0]=v[t3][0];
                        cc[k][1]=v[t3][1];
                        cc[k][2]=v[a3][0];
                        cc[k][3]=v[a3][1];
                        cc[k][4]=a3-t3+1.0;
                        k++;
                        cc[k][0]=v[t4][0];
                        cc[k][1]=v[t4][1];
                        cc[k][2]=v[a4][0];
                        cc[k][3]=v[a4][1];
                        cc[k][4]=43-t4+1.0;
                        k++;

                        initial[0]=v[t1+1][0];
                        initial[1]=v[t1+1][1];
                        if(flag==true && count>0){
                              //afero tis epanalipsis
                              int l;
                              for(i=0;i<k;i++){
```

```
                                                for(j=0;j<k;j++){
                                                      if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                                  for(l=j;j<k;j++){

        cc[j][0]=cc[j+1][0];

        cc[j][1]=cc[j+1][1];

        cc[j][2]=cc[j+1][2];

        cc[j][3]=cc[j+1][3];

        cc[j][4]=cc[j+1][4];
                                                                  }
                                                                  k=k-1;
                                                            }
                                                      }
                                                }
                                                //kai termatizw
                                                return k-1;
                                          }
                                          else{

        k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                                return k;
                                          }
                                          flag=false;
                                    }//telioni to else t space3
                                    flag=false;
                              }//telioni to else to space2
                              flag=false;
                        } // telioni to else t space1
                        flag=false;
        } // telioni to if t space0
        else{
                        a1=-1;
                        for(i=limit-1;i>=0;i--){
                              if(i==limit-1){
                                    d1=findDistanceOfTwoPoints(initial,v[i]);
                                    t1=i;
                              }
                              else{
                                    d1=findDistanceOfTwoPoints(v[i],v[i--]);
                                    t1=i;
                              }

                              if(d1>2){
                                    space1=d1;
                                    break;
                              }
                        }

                        if(space1 > (1/(2*limit))){
                              cc[k][0]=v[t1][0];
                              cc[k][1]=v[t1][1];
                              cc[k][2]=v[t0][0];
                              cc[k][3]=v[t0][1];
                              cc[k][4]=(limit-t1+1.0)+t0+2.0;
                              *pos=k;
```

```
                        flag=true;
                        k++;

                        initial[0]=v[t0+1][0];
                        initial[1]=v[t0+1][1];
                        if(flag==true && count>0){
                                    //afero tis epanalipsis
                                    int l;
                                    for(i=0;i<k;i++){
                                          for(j=0;j<k;j++){
                                                if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                      for(l=j;j<k;j++){

       cc[j][0]=cc[j+1][0];

       cc[j][1]=cc[j+1][1];

       cc[j][2]=cc[j+1][2];

       cc[j][3]=cc[j+1][3];

       cc[j][4]=cc[j+1][4];
                                                      }
                                                      k=k-1;
                                                }
                                          }
                                    }
                                    //kai termatizw
                                    return k-1;
                              }
                              else{

       k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                    return k;
                              }
                        flag=false;
                  }//telioni to if t space1
                  else{
                        a2=t1-1;
                        for(i=a2;i>=0;i--){
                              d2=findDistanceOfTwoPoints(v[i],v[i--]);
                              t2=i;

                              if(d2>2){
                                    space2=d2;
                                    break;
                              }
                        }

                        if(space2 > (1/(2*limit))){
                              cc[k][0]=v[t2][0];
                              cc[k][1]=v[t2][1];
                              cc[k][2]=v[t0][0];
                              cc[k][3]=v[t0][1];
                              cc[k][4]=(limit-t2+1.0)+t0+2.0;
                              *pos=k;
                              flag=true;
                              k++;
```

```
                        initial[0]=v[t0+1][0];
                        initial[1]=v[t0+1][1];
                        if(flag==true && count>0){
                                //afero tis epanalipsis
                                int l;
                                for(i=0;i<k;i++){
                                        for(j=0;j<k;j++){
                                                if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                        for(l=j;j<k;j++){

        cc[j][0]=cc[j+1][0];

        cc[j][1]=cc[j+1][1];

        cc[j][2]=cc[j+1][2];

        cc[j][3]=cc[j+1][3];

        cc[j][4]=cc[j+1][4];
                                                        }
                                                        k=k-1;
                                                }
                                        }
                                }
                                //kai termatizw
                                return k-1;
                        }
                        else{

        k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                return k;
                        }
                        flag=false;
                }//telioni to if t space2
                else{
                        a3=t2-1;
                        for(i=a3;i>=0;i--){
                                d3=findDistanceOfTwoPoints(v[i],v[i--]);
                                t3=i;

                                if(d3>2){
                                        space3=d3;
                                        break;
                                }
                        }

                        if(space3 > (1/(2*limit))){
                                cc[k][0]=v[t3][0];
                                cc[k][1]=v[t3][1];
                                cc[k][2]=v[t0][0];
                                cc[k][3]=v[t0][1];
                                cc[k][4]=(limit-t3+1.0)+t0+2.0;
                                *pos=k;
                                flag=true;
                                k++;

                                initial[0]=v[t0+1][0];
                                initial[1]=v[t0+1][1];
                                if(flag==true && count>0){
```

```
                                        //afero tis epanalipsis
                                        int l;
                                        for(i=0;i<k;i++){
                                               for(j=0;j<k;j++){
                                                      if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                                    for(l=j;j<k;j++){

       cc[j][0]=cc[j+1][0];

       cc[j][1]=cc[j+1][1];

       cc[j][2]=cc[j+1][2];

       cc[j][3]=cc[j+1][3];

       cc[j][4]=cc[j+1][4];
                                                                    }
                                                                    k=k-1;
                                                             }
                                               }
                                        }
                                        //kai termatizw
                                        return k-1;
                                 }
                                 else{

       k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                        return k;
                                 }
                                 flag=false;
                          }//telioni to if t space3
                          else{
                                 a4=t3-1;
                                 for(i=a4;i>=0;i--){
                                        d4=findDistanceOfTwoPoints(v[i],v[i--]);
                                        t4=i;

                                        if(d4>2){
                                               space4=d4;
                                               break;
                                        }
                                 }

                                 cc[k][0]=v[t1][0];
                                 cc[k][1]=v[t1][1];
                                 cc[k][2]=v[t0][0];
                                 cc[k][3]=v[t0][1];
                                 cc[k][4]=(limit-t1+1.0)+t0+2.0;
                                 *pos=k;
                                 flag=true;
                                 k++;
                                 cc[k][0]=v[t2][0];
                                 cc[k][1]=v[t2][1];
                                 cc[k][2]=v[a2][0];
                                 cc[k][3]=v[a2][1];
                                 cc[k][4]=t2-a2+1.0;
                                 k++;
                                 cc[k][0]=v[t3][0];
                                 cc[k][1]=v[t3][1];
```

```
                                    cc[k][2]=v[a3][0];
                                    cc[k][3]=v[a3][1];
                                    cc[k][4]=t3-a3+1.0;
                                    k++;
                                    cc[k][0]=v[t4][0];
                                    cc[k][1]=v[t4][1];
                                    cc[k][2]=v[a4][0];
                                    cc[k][3]=v[a4][1];
                                    cc[k][4]=t4-a4+1.0;
                                    k++;

                                    initial[0]=v[t0+1][0];
                                    initial[1]=v[t0+1][1];
                                    if(flag==true && count>0){
                                           //afero tis epanalipsis
                                           int l;
                                           for(i=0;i<k;i++){
                                                  for(j=0;j<k;j++){
                                                         if(cc[i][0]==cc[j][0] &&
cc[i][1]==cc[j][1] && cc[i][2]==cc[j][2] && cc[i][3]==cc[j][3] &&
cc[i][4]==cc[j][4] && i!=j){
                                                                for(l=j;j<k;j++){

       cc[j][0]=cc[j+1][0];

       cc[j][1]=cc[j+1][1];

       cc[j][2]=cc[j+1][2];

       cc[j][3]=cc[j+1][3];

       cc[j][4]=cc[j+1][4];
                                                                }
                                                                k=k-1;
                                                         }
                                                  }
                                           }
                                           //kai termatizw
                                           return k-1;
                                    }
                                    else{

       k=connectedComponent(cc,v,limit,c,count+1,k,&(*pos));
                                           return k;
                                    }
                                    flag=false;
                             }//telioni to else t space3
                             flag=false;
                      }//telioni to else t space2
                      flag=false;
               }//telioni to else t space1
               flag=false;
       }// telioni to else t space0


       return k-1;
}
```

Α.5 Συνάρτηση ΠόσηΑπόστασηΑπέχει

```
int howMuchDistance(float v[MAX][2], int limit, float c[2],float cc[MAX][6],int
k,int pos){
	int i,pos1,p1=0,p2=0,same=0;
	bool flag=false;
	float c1[2],c2[2];
	double d,min;
	c1[0]=cc[0][2];		c1[1]=cc[0][3];
	c2[0]=cc[1][0];		c2[1]=cc[1][1];
	min=findDistanceOfTwoPoints(c1,c2);
	for(i=1;i<k;i++){
		p1=i%limit;
		p2=(i+1)%limit;
		c1[0]=cc[p1][2];
		c1[1]=cc[p1][3];
		c2[0]=cc[p2][0];
		c2[1]=cc[p2][1];
		d=findDistanceOfTwoPoints(c1,c2);
		if(min>d)
			min=d;
	}

	for(i=0;i<k;i++){
		p1=i%limit;
		p2=(i+1)%limit;
		c1[0]=cc[p1][2];
		c1[1]=cc[p1][3];
		c2[0]=cc[p2][0];
		c2[1]=cc[p2][1];
		d=findDistanceOfTwoPoints(c1,c2);

		if(d==min){
			same++;
			if(cc[i][5]==1.0){
				flag=true;
				pos1=i;
			}
		}

	}

	if(same==k)
		return 2;
	else if(flag==true && same==1)
		return 1;
	else if(flag==true && (same<k && same>1)){
		if(pos==MAX-1){
			c1[0]=cc[0][2];
			c1[1]=cc[0][3];
			c2[0]=cc[1][0];
			c2[1]=cc[1][1];
			d=findDistanceOfTwoPoints(c1,c2);

			if(d==min)
				return 3;
			else
				return 1;
		}
```

```
		else{
			c1[0]=cc[pos1+1][2];
			c1[1]=cc[pos1+1][3];
			c2[0]=cc[pos1+2][0];
			c2[1]=cc[pos1+2][1];
			d=findDistanceOfTwoPoints(c1,c2);

			if(d==min)
				return 3;
			else
				return 1;
		}
	}
	else
		return 3;

}
```

Α.6 Συνάρτηση ΕίσαιΣτηνΜεγαλύτερηΣυγκέντρωση

```
int inLargestComponent(float v[MAX][2], int limit, float c[2],float
cc[MAX][6],int k,int pos){
	int i,j,same=0;
	bool flag=false;

	float max=cc[0][4];
	for(i=1;i<k;i++){
		if(max<cc[i][4]){
			max=cc[i][4];
		}
	}

	for(i=0;i<k;i++){
		if(max==cc[i][4]){
			same++;
			if(cc[i][5]==1.0)
				flag=true;
		}

	}

	if(same == k)
		return 2;
	else if(flag==true && (same<k && same>1))
		return 1;
	else
		return 3;
}
```

Α.7 Συνάρτηση ΕίσαιΣτηνΜικρότερηΣυγκέντρωση

```
int inSmallestComponent(float v[MAX][2], int limit, float c[2],float
cc[MAX][6],int k,int pos){
	int i,j,pos1,same=0;
	bool flag=false;
```

```
	float min=cc[0][4];
	for(i=1;i<k;i++){
		if(min<cc[i][4])
			min=cc[i][4];
	}

	for(i=0;i<k;i++){
		if(min==cc[i][4]){
			same++;
			if(cc[i][5]==1.0 && flag==false){
				flag=true;
				pos1=i;
			}
		}
	}


	if(same==k)
		return 2;
	else if(flag==true && same==1)
		return 1;
	else if(flag==true && same>1){
		if(pos1==MAX-1){
			if(cc[0][4]==min)
				return 3;
			else
				return 1;
		}
		else{
			if(cc[pos1+1][4]==min)
				return 3;
			else
				return 1;
		}
	}
	else
		return 3;


}
```

Α.8 Συνάρτηση ΣτηνΙδιαΕυθεία2

```
bool onStraightLine2(float cl[2], float c[2], float cr[2]){
	float a=findLine(cl,cr,0);
	float b=findLine(cl,cr,1);

	bool answer=isOnLine(a,b,c);

	if(answer==true)
		return true;
	else
		return false;
}
```

Παράρτημα Β

Στο παράρτημα αυτό θα παρουσιαστούν αναλυτικά ο κώδικας των συναρτήσεων του αλγορίθμου του προγράμματος που περιγράφονται στο κεφάλαιο 3, συγκεκριμένα στο υποκεφάλαιο 3.2. Αρχικά θα δοθεί το κομμάτι του κώδικα για την εναλλαγή των φάσεων του ρομπότ και στην συνέχεια οι διαδικασίες του ρομπότ στην φάση Υπολογίζω.

Β.1 Φάσεις Αλγορίθμου

```
switch(r->c[i].state){
                    // to robot einai sto state WAIT
                    case 'w':
                                int waitSec;
                                waitSec= rand() % 10;
                                printf("robot %d waits for: %d
sec\n",i,waitSec);
                                sleep(waitSec);
                                for(j=0;j<MAX;j++){
                                       r->c[i].visible[j][0]=-50;
                                       r->c[i].visible[j][1]=-50;
                                       r->c[i].ch[j][0]=-50;
                                       r->c[i].ch[j][1]=-50;
                                }
                                r->c[i].numV=0;
                                r->c[i].chNumV=0;
                                r->c[i].state='l';
                                r->c[i].statec=-1;
                                break;
                    // to robot einai sto state LOOK
                    case 'l':
                                int count;
                                bool in;
                                float c[2],c1[2];
                                c[0]=r->c[i].x;     c[1]=r->c[i].y;
                                printf("robot %d is looking around!!\n",i);

                                count=0;
                                for(j=0;j<MAX;j++){
                                       c1[0]=r->c[j].x;    c1[1]=r->c[j].y;
                                       in=isNotInVisible(r-
>c[i].visible,c1,count,0);
                                       if((in==true) && j!=i){

                                              float a=findLine(c,c1,0);
                                              float b=findLine(c,c1,1);
                                              float pOnLine[MAX][2];
                                              int k,count1=0;
                                              float c2[2];
                                              for(k=0;k<MAX;k++){
                                                     if(j!=k && !(r-
>c[k].x==c[0] && r->c[k].y==c[1])){
```

```
                                                        c2[0]=r->c[k].x;
        c2[1]=r->c[k].y;
                                                        bool
is=isOnLine(a,b,c2);
                                                        if(is==true){
                                                                double
d1,d2;
        d1=findDistanceOfTwoPoints(c,c2);
        d2=findDistanceOfTwoPoints(c1,c2);
        if(!(d2>d1)){
        pOnLine[count1][0]=c2[0];
        pOnLine[count1][1]=c2[1];
        count1++;
                                                                }
                                                        }
                                                }
                                                //an isxei afto tote brikame
stixia pano stin euthia ara to j den tha mpi tha mpei ena stixio
                                                //apo ton pinaka ponline.
                                                if(count1>0){
                                                        float d;
                                                        double
mind=findDistanceOfTwoPoints(c,pOnLine[0]);
                                                        int minp=0;
                                                        for(k=1;k<count1;k++){
        d=findDistanceOfTwoPoints(c,pOnLine[k]);
                                                                if(mind>d){
                                                                        mind=d;
                                                                        minp=k;
                                                                }
                                                        }
                                                        in=isNotInVisible(r-
>c[i].visible,pOnLine[minp],count,1);
                                                        if(in==true){
                                                                r-
>c[i].visible[count][0]=pOnLine[minp][0];
                                                                r-
>c[i].visible[count][1]=pOnLine[minp][1];
                                                                count++;
                                                        }
                                                }
                                                else{
                                                        //alios blepi sigura to
robot j
                                                        r-
>c[i].visible[count][0]=c1[0];
                                                        r-
>c[i].visible[count][1]=c1[1];
                                                        count++;
                                                }
                                        }//klini to if
```

```
                                        else if(in==true && j==i){ //eimai o
eaftos m kai mpeni katefthian
                                              r->c[i].visible[count][0]=c1[0];
                                              r->c[i].visible[count][1]=c1[1];
                                              count++;
                                        }
                                  }//arxiko for

                                  r->c[i].numV=count;
                                  r->c[i].state='c';
                                  r->c[i].statec=0;
                                  break;}
                        // to robot einai sto state COMPUTE
                        case 'c':
                              bool flagc;
                              flagc=true;
                              while(flagc==true){
                                    printf("stateC: %d \n",r->c[i].statec);
                                    switch(r->c[i].statec){
                                          //code for the compute procedures
                                    }

                                    break;
                        // to robot einai sto state MOVE
                        case 'm':
                                    r->c[i].px=r->c[i].x;      //krata tin
proigumeni thesi tu x
                                    r->c[i].py=r->c[i].y;      //krata tin
proigumeni thesi tu y
                                    r->c[i].x=p[0];
                                    r->c[i].y=p[1];
                                    r->c[i].state='w'; //pigainei sto state wait
                                    return 1;
                              break;
                        // to robot einai sto state TERMINATE
                        case 't':
                                    //termatizete to while ara termatizontai oi
ektelesis tu sigkekrimenu robot
                                    return -1;
                              break;
                  }
```

B.2 Διαδικασίες Αλγορίθμου

B.2.1 Διαδικασία Αρχή

```
bool answer;
r->c[i].chNumV=findConvexHull(r->c[i].visible,r->c[i].numV,r-
>c[i].ch);
int k,a;
      for(a=0;a<r->c[i].chNumV-1;a++){
      for(k=a+1;k<r->c[i].chNumV;k++){
            if((r->c[i].ch[a][0]==r->c[i].ch[k][0] && r-
>c[i].ch[a][1]==r->c[i].ch[k][1])){
                  int n=k;
                  while(n<r->c[i].chNumV){
                        r->c[i].ch[n][0]==r->c[i].ch[n+1][0];
```

```
                        r->c[i].ch[n][1]==r->c[i].ch[n+1][1];
                        n++;
                  }
                  r->c[i].chNumV=r->c[i].chNumV-1;
            }
      }
}
printf("Visible for robot %d with x=%f kai y=%f\n",i,r-
>c[i].x,r->c[i].y);
int j;
for(j=0;j<r->c[i].numV;j++)
      printf("robot me sintetagmenes %f & %f are visible for
robot %d\n",r->c[i].visible[j][0],r->c[i].visible[j][1],i);
printf("Convex Hull for robot %d\n",i);
for(j=0;j<r->c[i].chNumV;j++)
      printf("robot me sintetagmenes %f & %f on CH of robot
%d\n",r->c[i].ch[j][0],r->c[i].ch[j][1],i);


answer=onConvexHull(r->c[i].x ,r->c[i].y,r->c[i].ch,r-
>c[i].chNumV);

if(answer==true){
      printf("Convex Hull for robot %d and the answer is
true\n",i);
      r->c[i].statec=1;
}else{
      printf("Convex Hull for robot %d and the answer is
false\n",i);
      r->c[i].statec=12;
}
break;}
```

B.2.2 Διαδικασία ΠάνωΣτοΚυρτόΠερίβλημα

```
if(r->c[i].numV==MAX && r->c[i].chNumV==MAX){
      int j;
      bool is=false;
      for(j=1;j<MAX;j++){
            if(j==1)
                  is=onStraightLine2(r->c[i].visible[MAX-1],r-
>c[i].visible[j],r->c[i].visible[j+1]);
            else
                  is=onStraightLine2(r->c[i].visible[j-1],r-
>c[i].visible[j],r->c[i].visible[j+1]);

            if(is==true)
                  break;

      }
      if(is==true)
            r->c[i].statec=5;
      else
            r->c[i].statec=2;
}
else
      r->c[i].statec=5;
break;
```

Β.2.3 Διαδικασία ΠλήρηςΟρατότητα

```
float components[MAX][2];
int j,count;
int k,btrue=0;
count=1;
bool flag;
flag=true;

components[0][0]=r->c[i].visible[0][0];
components[0][1]=r->c[i].visible[0][1];

while(flag){
      for(j=0;j<count;j++){
            for(k=0;k<r->c[i].numV;k++){
                  bool is=isNotInComponent(r-
>c[i].visible[k],components,count);
                  if(is==true){
                        bool have=findTangentLine(r-
>c[i].visible[k],components[j],1);
                        if(have==true){
                              components[count][0]=r-
>c[i].visible[k][0];
                              components[count][1]=r-
>c[i].visible[k][1];
                              count++;
                              btrue++;
                        }
                  }
            }
      }

      if(btrue>0)
            btrue=0;
      else
            flag=false;
} // tu while

if(count==MAX)
      r->c[i].statec=3;
else
      r->c[i].statec=4;

break;
```

Β.2.4 Διαδικασία Συνδεδεμένα

```
r->c[i].statec=-1;
r->c[i].state='t';
flagc=false;
break;
```

Β.2.5 Διαδικασία ΌχιΣυνδεδεμένα

```
bool exitF=false;
for(j=0;j<r->c[i].chNumV;j++){
      if(r->c[i].ch[j][0]==r->c[i].x && r->c[i].ch[j][1]==r-
>c[i].y)
            break;
```

```
}
float c[2],cm[2],cl[2],cr[2],cr1[2];
if(j==0){
      c[0]=r->c[i].x;   c[1]=r->c[i].y;
      cm[0]=r->c[i].ch[MAX-1][0];   cm[1]=r->c[i].ch[MAX-1][1];
      cl[0]=r->c[i].ch[MAX-2][0];   cl[1]=r->c[i].ch[MAX-2][1];
      cr[0]=r->c[i].ch[j+1][0];     cr[1]=r->c[i].ch[j+1][1];
      cr1[0]=r->c[i].ch[j+2][0];    cr1[1]=r->c[i].ch[j+2][1];
}
else{
      c[0]=r->c[i].x;   c[1]=r->c[i].y;
      cm[0]=r->c[i].ch[j-1][0];     cm[1]=r->c[i].ch[j-1][1];
      cl[0]=r->c[i].ch[j-2][0];     cl[1]=r->c[i].ch[j-2][1];
      if(j==MAX-1){
            cr[0]=r->c[i].ch[0][0]; cr[1]=r->c[i].ch[0][1];
            cr1[0]=r->c[i].ch[0][0];      cr1[1]=r-
>c[i].ch[0][1];
      }
      else{
            cr[0]=r->c[i].ch[j+1][0];     cr[1]=r-
>c[i].ch[j+1][1];
            cr1[0]=r->c[i].ch[j+2][0];    cr1[1]=r-
>c[i].ch[j+2][1];
      }
}

double d;
d=foundDistanceofPointFromLine(c,cl,cm);

if(d<(1.0/MAX)){
      //claculate x... max distance ri can move
      float l=findLine(c,cm,0);
      float a;
      if(l==0)
            a=0;
      else
            a=-(1.0/l);
      float b=r->c[i].y-(l*r->c[i].x);
      float x1,y;
      x1=1;
      y=(a*x1)+b;
      float in[2],c2[2];
      c2[0]=x1; c2[1]=y;
      in[0]=intersectionPoint(cm,cl,c,c2,0);
      in[1]=intersectionPoint(cm,cl,c,c2,1);

      double din=findDistanceOfTwoPoints(c,in);
      float x=din+0.002;
      double angle=getAngleofTwoLines(c,in,cm,cr);
      float newp[2];
      newp[0]=x*cos(angle); newp[1]=x*sin(angle);

      bool flag1=false,flag2=false,flag3=false;
      double d1,d2,d3;
      //case1: ci in the middle
      d1=foundDistanceofPointFromLine(cm,c,cr);
      if(d1< (1.0/(2*MAX)))
            flag1=true;

      //case2: ci is the left
```

```
	d2=foundDistanceofPointFromLine(c,cr,cr1);
	if(d2< (1.0/(2*MAX)))
		flag2=true;

	//case3: ci in the right
	d3=foundDistanceofPointFromLine(cl,cm,c);
	if(d3< (1.0/(2*MAX)))
		flag1=true;


	if(flag1==true || flag2==true || flag3==true){
		continue;
	}
	else{
		double length=1.0/((2*MAX)-0.0002);
		double x1;
		if(x<length){
			x1=x;
		}
		else{
			x1=length;
		}

		p[0]=x1*cos(angle);
		p[1]=x1*sin(angle);
		exitF=true; //tha termatisi
	}


}
if(exitF==true)
	break;

bool  flag1,flag2;
flag1=areTangent(r->c[i].x,r->c[i].y,cm);
flag2=areTangent(r->c[i].x,r->c[i].y,cr);

if(flag1==false && flag2==false)
	continue;
else{
	double d=foundDistanceofPointFromLine(c,cl,cm);
	if(d<(1.0/MAX)){
		//claculate x... max distance ri can move
		float l=findLine(c,cm,0);
		float a;
		if(l==0)
			a=0;
		else
			a=-(1.0/l);

		float b=r->c[i].y-(l*r->c[i].x);
		float x1,y;
		x1=1;
		y=(a*x1)+b;
		float in[2],c2[2];
		c2[0]=x1; c2[1]=y;
		in[0]=intersectionPoint(cm,cl,c,c2,0);
		in[1]=intersectionPoint(cm,cl,c,c2,1);

		double din=findDistanceOfTwoPoints(c,in);
```

```
            float x=din-0.002;
            double angle=getAngleofTwoLines(c,in,cm,cr);
            float newp[2];
            newp[0]=x*cos(angle); newp[1]=x*sin(angle);

            bool flag1=false,flag2=false,flag3=false;
            //case1: ci in the middle
            double d1=foundDistanceofPointFromLine(cm,c,cr);
            if(d1< (1.0/(2*MAX)))
                  flag1=true;

            //case2: ci is the left
            double d2=foundDistanceofPointFromLine(c,cr,cr1);
            if(d2< (1.0/(2*MAX)))
                  flag2=true;

            //case3: ci in the right
            double d3=foundDistanceofPointFromLine(cl,cm,c);
            if(d3< (1.0/(2*MAX)))
                  flag1=true;

            if(flag1==true || flag2==true || flag3==true){
                  continue;
            }
            else{
                  double length=1.0/((2*MAX)-0.0002);
                  double x1;
                  if(x<length){
                        x1=x;
                  }
                  else{
                        x1=length;
                  }

                  p[0]=x1*cos(angle);
                  p[1]=x1*sin(angle);
                  exitF=true;
            }
      }
}

if(exitF==true)
      break;

int z;
bool indistance;
indistance=false;
for(z=0;z<r->c[i].chNumV;z++){
      if(z==MAX-1)
            d=foundDistanceofPointFromLine(r->c[i].ch[z],r-
>c[i].ch[1],r->c[i].ch[0]);
      else
            d=foundDistanceofPointFromLine(r->c[i].ch[z],r-
>c[i].ch[z+2],r->c[i].ch[z+1]);

      if(d< (1.0/MAX)){
            indistance=true;
            break;
      }
}
```

```
if(indistance==true){
	pthread_mutex_lock(&lock);
	p[0]=r->c[i].x;
	p[1]=r->c[i].y;
	pthread_mutex_unlock(&lock);
	exitF=true; //tha termatisi...
}

if(exitF==true)
	break;


bool is1,is2;
if(j==MAX){
	is1=areTangent(r->c[i].x,r->c[i].y,r->c[i].ch[0]);
	is2=areTangent(r->c[i].x,r->c[i].y,r->c[i].ch[j-1]);
}
else if(j==0){
	is1=areTangent(r->c[i].x,r->c[i].y,r->c[i].ch[j+1]);
	is2=areTangent(r->c[i].x,r->c[i].y,r->c[i].ch[MAX-1]);
}
else{
	is1=areTangent(r->c[i].x,r->c[i].y,r->c[i].ch[j+1]);
	is2=areTangent(r->c[i].x,r->c[i].y,r->c[i].ch[j-1]);
}

if(is1==true && is2==true){
	pthread_mutex_lock(&lock);
	p[0]=r->c[i].x;
	p[1]=r->c[i].y;
	pthread_mutex_unlock(&lock);
	exitF=true; //tha termatisi...
}

if(exitF==true)
	break;

int pos;
pos=0;
int numC;
numC=connectedComponent(r->c[i].cc,r->c[i].visible,r-
>c[i].numV,c,0,0,&pos);
bool t1;
t1=areTangent(r->c[i].x,r->c[i].y,cm);
if(j==MAX){
	cr[0]=r->c[i].ch[0][0]; cr[1]=r->c[i].ch[0][1];
}
else{
	cr[0]=r->c[i].ch[j+1][0];	cr[1]=r->c[i].ch[j+1][1];
}
bool t2;
float point[2];
t2=areTangent(r->c[i].x,r->c[i].y,cr);
if(numC==1 && t1==false && t2==false){
	float a=findLine(cm,cr,0);
	float a1;
	if(a==0)
		a1=0;
	else
```

```
            a1=-(1.0/a);
      float x=findPointIntersectsTheXaxis(a1,c);
      float c2[2],c5[2];
      c2[0]=x;    c2[1]=0.0;
      c5[0]=0.0;  c5[1]=0.0;
      double angle=getAngleofTwoLines(c,c2,c2,c5);
      point[0]=c[0]+((1.0/MAX)*cos(angle));
      point[1]=c[1]+((1.0/MAX)*sin(angle));

      pthread_mutex_lock(&lock);
      p[0]=point[0];
      p[1]=point[1];
      pthread_mutex_unlock(&lock);
}
else{
      int numl=inLargestComponent(r->c[i].visible,r-
>c[i].numV,c);
      if(numl==1){
            pthread_mutex_lock(&lock);
            p[0]=r->c[i].x;
            p[1]=r->c[i].y;
            pthread_mutex_unlock(&lock);
      }
      else if(numl==2){
            int numh=howMuchDistance(r->c[i].visible,r-
>c[i].numV,c);
            if(numh==1){
                  if(j==MAX){
                        point[0]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,0);
                        point[1]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,1);
                  }
                  else{
                        point[0]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,0);
                        point[1]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,1);
                  }
                  pthread_mutex_lock(&lock);
                  p[0]=point[0];
                  p[1]=point[1];
                  pthread_mutex_unlock(&lock);
            }
            else if(numh==2){
                  int pos=0;
                  int n=connectedComponent(r->c[i].cc,r-
>c[i].visible,r->c[i].numV,c,0,0,&pos);
                  float c3[2],c4[2];
                  c3[0]=r->c[i].cc[pos][0];     c3[1]=r-
>c[i].cc[pos][1];
                  c4[0]=r->c[i].cc[pos][0];     c4[1]=r-
>c[i].cc[pos][1];
                  float a=findLine(c3,c4,0);
                  float x=findPointIntersectsTheXaxis(a,c);
                  float c2[2],c5[2];
                  c2[0]=x;    c2[1]=0.0;
                  c5[0]=0.0;  c5[1]=0.0;
                  double angle=getAngleofTwoLines(c,c2,c2,c5);
                  point[0]=c[0]+((1.0/MAX)*cos(angle));
```

```
                  point[1]=c[1]+((1.0/MAX)*sin(angle));

                  pthread_mutex_lock(&lock);
                  p[0]=point[0];
                  p[1]=point[1];
                  pthread_mutex_unlock(&lock);

            }
            else{ //numh==3
                  pthread_mutex_lock(&lock);
                  p[0]=r->c[i].x;
                  p[1]=r->c[i].y;
                  pthread_mutex_unlock(&lock);
            }
      }
      else{ //numl==3
            int nums=inSmallestComponent(r->c[i].visible,r-
>c[i].numV,c);
            if(nums==1){
                  if(j==MAX){
                        point[0]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,0);
                        point[1]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,1);
                  }
                  else{
                        point[0]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,0);
                        point[1]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,1);
                  }
                  pthread_mutex_lock(&lock);
                  p[0]=point[0];
                  p[1]=point[1];
                  pthread_mutex_unlock(&lock);
            }
            else if(nums==2){
                  int numh=howMuchDistance(r->c[i].visible,r-
>c[i].numV,c);
                  if(numh==1){
                        if(j==MAX){
                              point[0]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,0);
                              point[1]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,1);
                        }
                        else{
                              point[0]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,0);
                              point[1]=moveToPoint(r->c[i].x,r-
>c[i].y,r->c[i].ch[0][0],r->c[i].ch[0][0],r->c[i].chNumV,1);
                        }
                        pthread_mutex_lock(&lock);
                        p[0]=point[0];
                        p[1]=point[1];
                        pthread_mutex_unlock(&lock);
                  }
                  else if(numh==2){
                        int pos=0;
```

```
                        int n=connectedComponent(r->c[i].cc,r-
>c[i].visible,r->c[i].numV,c,0,0,&pos);
                        float c3[2],c4[2];
                        c3[0]=r->c[i].cc[pos][0];     c3[1]=r-
>c[i].cc[pos][1];
                        c4[0]=r->c[i].cc[pos][0];     c4[1]=r-
>c[i].cc[pos][1];
                        float a=findLine(c3,c4,0);
                        float
x=findPointIntersectsTheXaxis(a,c);
                        float c2[2],c5[2];
                        c2[0]=x;    c2[1]=0.0;
                        c5[0]=0.0;  c5[1]=0.0;
                        double
angle=getAngleofTwoLines(c,c2,c2,c5);
                        point[0]=c[0]+((1.0/MAX)*cos(angle));
                        point[1]=c[1]+((1.0/MAX)*sin(angle));

                        pthread_mutex_lock(&lock);
                        p[0]=point[0];
                        p[1]=point[1];
                        pthread_mutex_unlock(&lock);

                  }
                  else{ //numh==3
                        pthread_mutex_lock(&lock);
                        p[0]=r->c[i].x;
                        p[1]=r->c[i].y;
                        pthread_mutex_unlock(&lock);
                  }
            }
            else{ //nums==3
                  pthread_mutex_lock(&lock);
                  p[0]=r->c[i].x;
                  p[1]=r->c[i].y;
                  pthread_mutex_unlock(&lock);
            }

      }
}//klini to else
r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;

break;
```

Β.2.6 Διαδικασία ΌχιΠλήρηςΟρατότητα

```
float l,l1,c2[2],c3[2];
double x,angle,angle1;
float a[2],b[2],c[2],cl[2],cr[2],cm[2],D[2],vertex[4],vertey[4];
bool flag1,flag2,flag3;
flag1=false;
flag2=false;
flag3=false;
for(j=0;j<r->c[i].chNumV;j++)
      if(r->c[i].x==r->c[i].ch[j][0] && r->c[i].y==r-
>c[i].ch[j][1])
            break;
```

```
//case1: cm=ci
if(j==0){
      cl[0]=r->c[i].ch[j+1][0];
      cl[1]=r->c[i].ch[j+1][1];
      cr[0]=r->c[i].ch[r->c[i].chNumV-1][0];
      cr[1]=r->c[i].ch[r->c[i].chNumV-1][1];
      cm[0]=r->c[i].x;
      cm[1]=r->c[i].y;
}
else if(j==r->c[i].chNumV-1){
      cl[0]=r->c[i].ch[0][0];       cl[1]=r->c[i].ch[0][1];
      cr[0]=r->c[i].ch[j-1][0];     cr[1]=r->c[i].ch[j-1][1];
      cm[0]=r->c[i].x;        cm[1]=r->c[i].y;
}
else{
      cl[0]=r->c[i].ch[j+1][0];     cl[1]=r->c[i].ch[j+1][1];
      cr[0]=r->c[i].ch[j-1][0];     cr[1]=r->c[i].ch[j-1][1];
      cm[0]=r->c[i].x;        cm[1]=r->c[i].y;
}


c3[0]=0;    c3[1]=0;
l=findLine(cl,cr,0);
if(l==0)
      l1=0;
else
      l1=-(1.0/l);
x=findPointIntersectsTheXaxis(l1,cl);
c2[0]=x;    c2[1]=0;
angle=getAngleofTwoLines(cl, c2,c2, c3);

x=findPointIntersectsTheXaxis(l1,cl);
c2[0]=x;    c2[1]=0;
angle1=getAngleofTwoLines(cr, c2,c2, c3);

a[0]=cl[0]+((1.0/MAX)*cos(angle));
a[1]=cl[1]+((1.0/MAX)*sin(angle));
b[0]=cl[0]+((-1.0/MAX)*cos(angle)); b[1]=cl[1]+((-
1.0/MAX)*sin(angle));
c[0]=cr[0]+((1.0/MAX)*cos(angle1));
c[1]=cr[1]+((1.0/MAX)*sin(angle1));
D[0]=cr[0]+((-1.0/MAX)*cos(angle1));      D[1]=cr[1]+((-
1.0/MAX)*sin(angle1));

vertex[0]=a[0];   vertex[0]=b[0];   vertey[0]=c[0];
vertey[0]=D[0];
vertey[1]=a[1];   vertey[1]=b[1];   vertey[1]=c[1];
vertey[1]=D[1];

flag1=onPolygon(4,vertex,vertey,cm[0], cm[1]);

//case2: cl=ci
if(j==r->c[i].chNumV-1){
      cm[0]=r->c[i].ch[0][0]; cm[1]=r->c[i].ch[0][1];
      cr[0]=r->c[i].ch[1][0]; cr[1]=r->c[i].ch[1][1];
      cl[0]=r->c[i].x;        cl[1]=r->c[i].y;
}
else{
      cm[0]=r->c[i].ch[j+1][0];     cm[1]=r->c[i].ch[j+1][1];
```

```
      cr[0]=r->c[i].ch[j+2][0];     cr[1]=r->c[i].ch[j+2][1];
      cl[0]=r->c[i].x;              cl[1]=r->c[i].y;
}

c3[0]=0;    c3[1]=0;
l=findLine(cl,cr,0);

if(l==0)
      l1=0;
else
      l1=-(1.0/l);
x=findPointIntersectsTheXaxis(l1,cl);
c2[0]=x;    c2[1]=0;
angle=getAngleofTwoLines(cl, c2,c2, c3);

x=findPointIntersectsTheXaxis(l1,cl);
c2[0]=x;    c2[1]=0;
angle1=getAngleofTwoLines(cr, c2,c2, c3);

a[0]=cl[0]+((1.0/MAX)*cos(angle));
a[1]=cl[1]+((1.0/MAX)*sin(angle));
b[0]=cl[0]+((-1.0/MAX)*cos(angle)); b[1]=cl[1]+((-
1.0/MAX)*sin(angle));
c[0]=cr[0]+((1.0/MAX)*cos(angle1));
c[1]=cr[1]+((1.0/MAX)*sin(angle1));
D[0]=cr[0]+((-1.0/MAX)*cos(angle1));      D[1]=cr[1]+((-
1.0/MAX)*sin(angle1));

vertex[0]=a[0];   vertex[0]=b[0];   vertey[0]=c[0];
vertey[0]=D[0];
vertey[1]=a[1];   vertey[1]=b[1];   vertey[1]=c[1];
vertey[1]=D[1];

flag2=onPolygon(4,vertex,vertey,cm[0], cm[1]);

//case3: cr=ci
if(j==0){
      cm[0]=r->c[i].ch[r->c[i].chNumV-1][0];    cm[1]=r-
>c[i].ch[r->c[i].chNumV-1][1];
      cl[0]=r->c[i].ch[r->c[i].chNumV-2][0];    cl[1]=r-
>c[i].ch[r->c[i].chNumV-2][1];
      cr[0]=r->c[i].x;        cr[1]=r->c[i].y;
}
else{
      cm[0]=r->c[i].ch[j-1][0];     cm[1]=r->c[i].ch[j-1][1];
      cl[0]=r->c[i].ch[j-2][0];     cl[1]=r->c[i].ch[j-2][1];
      cr[0]=r->c[i].x;        cr[1]=r->c[i].y;
}

c3[0]=0;    c3[1]=0;
l=findLine(cl,cr,0);

if(l==0)
      l1=0;
else
      l1=-(1.0/l);
x=findPointIntersectsTheXaxis(l1,cl);
c2[0]=x;    c2[1]=0;
angle=getAngleofTwoLines(cl, c2,c2, c3);
```

```
x=findPointIntersectsTheXaxis(l1,cl);
c2[0]=x;    c2[1]=0;
angle1=getAngleofTwoLines(cr, c2,c2, c3);

a[0]=cl[0]+((1.0/MAX)*cos(angle));
a[1]=cl[1]+((1.0/MAX)*sin(angle));
b[0]=cl[0]+((-1.0/MAX)*cos(angle)); b[1]=cl[1]+((-
1.0/MAX)*sin(angle));
c[0]=cr[0]+((1.0/MAX)*cos(angle1));
c[1]=cr[1]+((1.0/MAX)*sin(angle1));
D[0]=cr[0]+((-1.0/MAX)*cos(angle1));      D[1]=cr[1]+((-
1.0/MAX)*sin(angle1));

vertex[0]=a[0];   vertex[0]=b[0];   vertey[0]=c[0];
vertey[0]=D[0];
vertey[1]=a[1];   vertey[1]=b[1];   vertey[1]=c[1];
vertey[1]=D[1];

flag3=onPolygon(4,vertex,vertey,cm[0], cm[1]);

if(flag1==true || flag2==true || flag3==true)
      r->c[i].statec=9;
else
      r->c[i].statec=6;

break;
```

B.2.7 Διαδικασία ΔενΕίναιΣεΕυθεία

```
int f;
f=0;
if(r->c[i].chNumV==MAX){
      r->c[i].statec=7;
      f=1;
}

if(f==0){
      if(r->c[i].numV==MAX){
            int j;
            double d;
            for(j=1;j<MAX;j++){
                  d=findDistanceOfTwoPoints(r->c[i].ch[j-1],r-
>c[i].ch[j]);
                  if(d>=2.0){ //exist
                        r->c[i].statec=7;
                        break;
                  }
                  else //not exist
                        r->c[i].statec=8;
            }
      }
      else{
            float ch2[MAX][2];
            int j,k,n;
            //copy ch in ch2
            for(j=0;j<r->c[i].chNumV;j++){
                  ch2[j][0]=r->c[i].ch[j][0];
                  ch2[j][1]=r->c[i].ch[j][1];
            }
```

```
            int flag=0;
            for(j=0;j<r->c[i].numV;j++){
                  for(k=0;k<r->c[i].chNumV;k++){
                        if(r->c[i].visible[j][0]==r-
>c[i].ch[k][0] && r->c[i].visible[j][1]==r->c[i].ch[k][1])
                              flag=1;
                  }

                  if(flag==1)
                        flag=0;
                  else
                        break; //found point in visible and not
onCH
            }
            //briskume ton epomeno tu j pou vriskete onCH
            flag=0;
            for(k=j+1;k<r->c[i].numV;k++){
                  for(n=0;n<r->c[i].chNumV;n++){
                        if(r->c[i].visible[k][0]==r-
>c[i].ch[n][0] && r->c[i].visible[k][1]==r->c[i].ch[n][1])
                              flag=1;
                  }
                  if(flag==1)
                        break;
            }
            //to n kai n-
1 einai ta stixia pano sto ch pou tha vrume to
x stin eutheia ts!

            float x[2],c[2];
            c[0]=r->c[i].x;
            c[1]=r->c[i].y;
            if(n==0){
                  x[0]=intersectionPoint(c,r->c[i].visible[j],r-
>c[i].ch[n],r->c[i].ch[r->c[i].numV-1],0);
                  x[1]=intersectionPoint(c,r->c[i].visible[j],r-
>c[i].ch[n],r->c[i].ch[r->c[i].numV-1],1);
            }
            else{
                  x[0]=intersectionPoint(c,r->c[i].visible[j],r-
>c[i].ch[n],r->c[i].ch[n-1],0);
                  x[1]=intersectionPoint(c,r->c[i].visible[j],r-
>c[i].ch[n],r->c[i].ch[n-1],1);
            }
            //put point x on the CH2
            for(k=r->c[i].chNumV+1;k<n;k++){
                  ch2[k][0]=ch2[k-1][0];
                  ch2[k][1]=ch2[k-1][1];
            }

            ch2[n][0]=x[0];
            ch2[n][1]=x[1];

            double d;
            for(j=1;j<r->c[i].chNumV+1;j++){
                  d=findDistanceOfTwoPoints(ch2[j-1],ch2[j]);
                  if(d>=2.0){ //exist
                        r->c[i].statec=7;
                        break;
                  }
```

```
                  else //not exist
                        r->c[i].statec=8;
            }
      }
}

break;
```

B.2.8 Διαδικασία ΥπάρχειΧώροςΓιαΆλλους

```
float cl[2],cr[2],cm[2];
int posi,pos;
pos=0;
posi=0;
for(j=0;j<r->c[i].chNumV;j++){
      if(r->c[i].ch[j][0]!=r->c[i].x && r->c[i].ch[j][1]!=r-
>c[i].y){
            bool is=areTangent(r->c[i].x,r->c[i].y,r-
>c[i].ch[j]);
            if(is==true){
                  pos=1;
            }
      }
      else{
            posi=j;
      }
}

if(pos==1){
      if(posi==0){
            cl[0]=r->c[i].ch[r->c[i].chNumV-1][0];    cl[1]=r-
>c[i].ch[r->c[i].chNumV-1][1];
            cr[0]=r->c[i].ch[1][0]; cr[1]=r->c[i].ch[1][1];
      }
      else if(posi==r->c[i].chNumV-1){
            cl[0]=r->c[i].ch[posi-1][0];  cl[1]=r->c[i].ch[posi-
1][1];
            cr[0]=r->c[i].ch[0][0]; cr[1]=r->c[i].ch[0][1];
      }
      else{
            cl[0]=r->c[i].ch[posi-1][0];  cl[1]=r->c[i].ch[posi-
1][1];
            cr[0]=r->c[i].ch[posi+1][0];  cr[1]=r-
>c[i].ch[posi+1][1];
      }

      if((cl[0]!=r->c[i].ch[j][0] && cl[1]!=r->c[i].ch[j][1]) &&
(cr[0]!=r->c[i].ch[j][0] && cr[1]!=r->c[i].ch[j][1])){
            float l[2];
            l[0]=findLine(cl,cr,0); //a
            l[1]=findLine(cl,cr,1); //b

            float c[2];
            c[0]=r->c[i].x;         c[1]=r->c[i].y;
            float a=-1.0/l[0];
            float x=findPointIntersectsTheXaxis(a,c);
            float c2[2],c3[2];
            c2[0]=x;          c2[1]=0;
            c3[0]=0;          c3[1]=0;
```

```
            double angle=getAngleofTwoLines(c,c2,c2,c3);

            double e=(1.0/(2*MAX))-0.002;
            p[0]=c[0]+(e*cos(angle));
            p[1]=c[1]+(e*sin(angle));

      }
      else{
            pthread_mutex_lock(&lock);
            p[0]=r->c[i].x;
            p[1]=r->c[i].y;
            pthread_mutex_unlock(&lock);
      }


}
else{
      pthread_mutex_lock(&lock);
      p[0]=r->c[i].x;
      p[1]=r->c[i].y;
      pthread_mutex_unlock(&lock);
}


r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;
break;
```

Β.2.9 Διαδικασία ΔενΥπάρχειΧώροςΓιαΆλλους

```
float mid[2],point1[2],cl[2],cr[2],cm[2];
for(j=0;j<r->c[i].chNumV;j++)
      if(r->c[i].ch[j][0]==r->c[i].x && r->c[i].ch[j][1]==r-
>c[i].y)
            break;

if(j==0){
      cl[0]=r->c[i].ch[r->c[i].chNumV-1][0];    cl[1]=r-
>c[i].ch[r->c[i].chNumV-1][1];
      cr[0]=r->c[i].ch[j+1][0];     cr[1]=r->c[i].ch[j+1][1];
}
else if(j==r->c[i].chNumV-1){
      cl[0]=r->c[i].ch[r->c[i].chNumV-2][0];    cl[1]=r-
>c[i].ch[r->c[i].chNumV-2][1];
      cr[0]=r->c[i].ch[j+1][0];     cr[1]=r->c[i].ch[j+1][1];
}
else{
      cl[0]=r->c[i].ch[j-1][0];     cl[1]=r->c[i].ch[j-1][1];
      cr[0]=r->c[i].ch[j+1][0];     cr[1]=r->c[i].ch[j+1][1];
}

mid[0]=(cl[0]+cr[0])/2;
mid[1]=(cl[1]+cr[1])/2;
float a3;
a3=findLine(cl,cr,0);
double x=findPointIntersectsTheXaxis(a3,mid);
float c2[2],c3[2],point[2];
c2[0]=x;    c2[1]=0;
```

```
c3[0]=0;    c3[1]=0;
double angle=getAngleofTwoLines( cm, c2,c2, c3);

point[0]=mid[0]+(((1.0/(2*MAX))-0.0002)*cos(angle));
point[1]=mid[1]+(((1.0/(2*MAX))-0.0002)*sin(angle));

//find p' which is between m and p!
point1[0]=mid[0]+(0.1*cos(angle));
point1[1]=mid[1]+(0.1*sin(angle));

p[0]=point1[0];
p[1]=point1[1];

r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;
break;
```

B.2.10 Διαδικασία ΣεΕυθεία

```
float
cl[2],cr[2],cm[2],a[2],b[2],c[2],D[2],vertex[2],vertey[2],c2[2],
c3[2],l,l1;
double x,angle,angle1;
bool flag1,flag2,flag3;
flag1=false;
flag2=false;
flag3=false;
for(j=0;j<r->c[i].chNumV;j++)
      if(r->c[i].x==r->c[i].ch[j][0] && r->c[i].y==r-
>c[i].ch[j][1])
            break;

//case1: cm=ci
if(j==0){
      cl[0]=r->c[i].ch[j+1][0];     cl[1]=r->c[i].ch[j+1][1];
      cr[0]=r->c[i].ch[r->c[i].chNumV-1][0];    cr[1]=r-
>c[i].ch[r->c[i].chNumV-1][1];
      cm[0]=r->c[i].x;              cm[1]=r->c[i].y;
}
else if(j==r->c[i].chNumV-1){
      cl[0]=r->c[i].ch[0][0]; cl[1]=r->c[i].ch[0][1];
      cr[0]=r->c[i].ch[j-1][0];     cr[1]=r->c[i].ch[j-1][1];
      cm[0]=r->c[i].x;              cm[1]=r->c[i].y;
}
else{
      cl[0]=r->c[i].ch[j+1][0];     cl[1]=r->c[i].ch[j+1][1];
      cr[0]=r->c[i].ch[j-1][0];     cr[1]=r->c[i].ch[j-1][1];
      cm[0]=r->c[i].x;              cm[1]=r->c[i].y;
}

c3[0]=0;    c3[1]=0;
l=findLine(cl,cr,0);

if(l==0)
      l1=0;
else
      l1=-(1.0/l);
x=findPointIntersectsTheXaxis(l1,cl);
```

```
c2[0]=x;    c2[1]=0;
angle=getAngleofTwoLines(cl, c2,c2, c3);

x=findPointIntersectsTheXaxis(l1,cl);
c2[0]=x;    c2[1]=0;
angle1=getAngleofTwoLines(cr, c2,c2, c3);

a[0]=cl[0]+((1.0/MAX)*cos(angle));
a[1]=cl[1]+((1.0/MAX)*sin(angle));
b[0]=cl[0]+((-1.0/MAX)*cos(angle)); b[1]=cl[1]+((-
1.0/MAX)*sin(angle));
c[0]=cr[0]+((1.0/MAX)*cos(angle1));
c[1]=cr[1]+((1.0/MAX)*sin(angle1));
D[0]=cr[0]+((-1.0/MAX)*cos(angle1));      D[1]=cr[1]+((-
1.0/MAX)*sin(angle1));

vertex[0]=a[0];   vertex[0]=b[0];   vertey[0]=c[0];
vertey[0]=D[0];
vertey[1]=a[1];   vertey[1]=b[1];   vertey[1]=c[1];
vertey[1]=D[1];

flag1=onPolygon(4,vertex,vertey,cm[0], cm[1]);

if(flag1==true)
      r->c[i].statec=11;
else
      r->c[i].statec=10;
break;
```

B.2.911 Διαδικασία ΒλέπειΈναΡομπότ

```
pthread_mutex_lock(&lock);
p[0]=r->c[i].x;
p[1]=r->c[i].y;
pthread_mutex_unlock(&lock);
r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;
break;
```

B.2.12 Διαδικασία ΒλέπειΔύοΡομπότ

```
float p1[2],p2[2],in[2],cl[2],cr[2],cm[2];
for(j=0;j<r->c[i].chNumV;j++)
      if(r->c[i].x==r->c[i].ch[j][0] && r->c[i].y==r-
>c[i].ch[j][1])
            break;
//case: cm=ci
if(j==0){
      cl[0]=r->c[i].ch[j+1][0];     cl[1]=r->c[i].ch[j+1][1];
      cr[0]=r->c[i].ch[r->c[i].chNumV-1][0];    cr[1]=r-
>c[i].ch[r->c[i].chNumV-1][1];
      cm[0]=r->c[i].x;              cm[1]=r->c[i].y;
}
else if(j==r->c[i].chNumV-1){
      cl[0]=r->c[i].ch[0][0]; cl[1]=r->c[i].ch[0][1];
      cr[0]=r->c[i].ch[j-1][0];     cr[1]=r->c[i].ch[j-1][1];
      cm[0]=r->c[i].x;              cm[1]=r->c[i].y;
}
```

```
else{
      cl[0]=r->c[i].ch[j+1][0];     cl[1]=r->c[i].ch[j+1][1];
      cr[0]=r->c[i].ch[j-1][0];     cr[1]=r->c[i].ch[j-1][1];
      cm[0]=r->c[i].x;              cm[1]=r->c[i].y;
}

//find point p1, starts from point cm and goes vertical to
line clcr
float a1,a2;
a1=findLine(cl,cr,0);
if(a1==0)
      a2=0;
else
      a2=-1.0/a1;
float x=findPointIntersectsTheXaxis(a2,cm);
float c2[2],c3[2];
c2[0]=x;    c2[1]=0;
c3[0]=0;    c3[1]=0;
double angle=getAngleofTwoLines( cm, c2,c2, c3);

p1[0]=cm[0]+(((1.0/(2*MAX))-0.0001)*cos(angle));
p1[1]=cm[1]+(((1.0/(2*MAX))-0.0001)*sin(angle));

//find the intersection point of line clcr and line cmp1
in[0]=intersectionPoint(cl, cr,p1, cm,0);
in[1]=intersectionPoint(cl, cr,p1, cm,1);

x=findPointIntersectsTheXaxis(a2,in);
c2[0]=x;    c2[1]=0;
c3[0]=0;    c3[1]=0;
angle=getAngleofTwoLines( in, c2,c2, c3);

//find point2 from the line clcr vertical to it
p2[0]=in[0]+((1.0/MAX)*cos(angle));
p2[1]=in[1]+((1.0/MAX)*sin(angle));

//find distances of the two points and find which is nearest
to cm
double d1,d2;
d1=findDistanceOfTwoPoints(cm,p1);
d2=findDistanceOfTwoPoints(cm,p2);

if(d1<d2){
      p[0]=p1[0];
      p[1]=p1[1];
}
else{
      p[0]=p2[0];
      p[1]=p2[1];
}

r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;
break;
```

B.2.13 ΔιαδικασίαΌχιΠάνωΣτοΚυρτόΠερίβλημα

```
bool is;
```

```
is=false;

for(j=0;j<r->c[i].numV;j++){
      if(!(r->c[i].x==r->c[i].visible[j][0] && r->c[i].y==r-
>c[i].visible[j][1])){
            is=areTangent(r->c[i].x,r->c[i].y,r-
>c[i].visible[j]);
            if(is==true)
                  break;
      }
}

if(is==true)
      r->c[i].statec=13;
else
      r->c[i].statec=14;
break;
```

B.2.14 Διαδικασία Κολλημένο

```
float points[100][2];
int count=findPoint(r->c[i].ch,points,r->c[i].chNumV);
if(count>=1){
float c[2];
double d;

pthread_mutex_lock(&lock);
c[0]=r->c[i].x;
c[1]=r->c[i].y;
pthread_mutex_unlock(&lock);

//find the most near point from the points the function returned
double min=findDistanceOfTwoPoints(c,points[0]);
int k,minp=0;
for(k=0;k<count;k++){
      d=findDistanceOfTwoPoints(c,points[k]);
      if(d<min){
            min=d;
            minp=k;
      }
}

float tpoints[100][2];
int count1;
double mind;
count1=findTouchingPoints(c,r->c[i].visible,tpoints,r-
>c[i].numV);
mind=findMinDistance(tpoints,points[minp],count1);

if(min>mind){
      pthread_mutex_lock(&lock);
      p[0]=r->c[i].x;
      p[1]=r->c[i].y;
      pthread_mutex_unlock(&lock);
}

float spoints[100][2];
int samed;
```

```
double distnace;
distnace=findDistanceOfTwoPoints(points[minp],c);
samed=findSameDistance(points[minp],tpoints,spoints,count1,distn
ace);
if(samed>=1){
      int pos=findRightMostPoint(c,spoints,samed);
      if(pos==-1){
            int j;
            float cl[2],cr[2];
            for(j=0;j<r->c[i].numV;j++){
                  if(r->c[i].x==r->c[i].visible[j][0] && r-
>c[i].y==r->c[i].visible[j][1])
                        break;
            }

            if(j==0){
                  cl[0]=r->c[i].visible[r->c[i].numV-1][0];
cl[1]=r->c[i].visible[r->c[i].numV-1][0];
                  cr[0]=r->c[i].visible[j+1][0];      cr[1]=r-
>c[i].visible[j+1][0];
            }
            else if(j==MAX-1){
                  cl[0]=r->c[i].visible[j-1][0];      cl[1]=r-
>c[i].visible[j-1][0];
                  cr[0]=r->c[i].visible[0][0];  cr[1]=r-
>c[i].visible[0][0];
            }
            else{
                  cl[0]=r->c[i].visible[j-1][0];      cl[1]=r-
>c[i].visible[j-1][0];
                  cr[0]=r->c[i].visible[j+1][0];      cr[1]=r-
>c[i].visible[j+1][0];
            }

            if((cl[0]!=points[minp][0] &&
cl[1]!=points[minp][1]) && (cr[0]!=points[minp][0] &&
cr[1]!=points[minp][1])){
                  float in[2];

in[0]=intersectionPoint(cl,cr,points[minp],c,0);

in[1]=intersectionPoint(cl,cr,points[minp],c,1);

                  p[0]=in[0]+(0.1*cos(90.0));
                  p[1]=in[1]+(0.1*sin(90.0));

            }
            else{
                  p[0]=points[minp][0]+(0.1*cos(90.0));
                  p[1]=points[minp][1]+(0.1*sin(90.0));
            }
      }
      else{
            pthread_mutex_lock(&lock);
            p[0]=r->c[i].x;
            p[1]=r->c[i].y;
            pthread_mutex_unlock(&lock);
      }
}
else{
```

```
      pthread_mutex_lock(&lock);
      p[0]=r->c[i].x;
      p[1]=r->c[i].y;
      pthread_mutex_unlock(&lock);
}

}//if count>=1
else{ //the function find points didn't return any points...
int k;
float nearp[2][2],c[2];
c[0]=r->c[i].x;         c[1]=r->c[i].y;
bool is=findNearestTwoPoints(c,nearp,r->c[i].ch,r->c[i].chNumV);

if(is == false){
      pthread_mutex_lock(&lock);
      p[0]=r->c[i].x;
      p[1]=r->c[i].y;
      pthread_mutex_unlock(&lock);
}
else{
      float tpoints[100][2];
      int count=findTouchingPoints(c,r->c[i].visible,tpoints,r-
>c[i].numV);
      double min[2];
      double **distance=(double
**)malloc(sizeof(double*)*count);
      for(i=0; i<count; i++){
            distance[i] = (double *)malloc(sizeof(double)*2);
      }

      min[0]=findDistanceOfTwoPoints(nearp[0],c);
      min[1]=findDistanceOfTwoPoints(nearp[1],c);

      for(k=0;k<count;k++){

distance[i][0]=findDistanceOfTwoPoints(nearp[0],tpoints[k]);

distance[i][1]=findDistanceOfTwoPoints(nearp[1],tpoints[k]);
      }
      int minp;
      double min1,min2;
      min1=distance[0][0];
      min2=distance[0][1];
      minp=0;
      for(k=0;k<count;k++){
            if(min1>distance[k][0] && min2>distance[k][1]){
                  min1=distance[0][0];
                  min2=distance[0][1];
                  minp=k;
            }
      }

      if(min1<min[0] && min2<min[1]){
            pthread_mutex_lock(&lock);
            p[0]=r->c[i].x;
            p[1]=r->c[i].y;
            pthread_mutex_lock(&lock);
      }
      else{
            float spoints[100][2];
```

```
            int
sd1=findSameDistance(nearp[0],tpoints,spoints,count,min[0]);
            int
sd2=findSameDistance(nearp[1],tpoints,spoints,count,min[1]);

            if(sd1>=1 && sd2>=1){
                  int pos=findRightMostPoint(c,spoints,count);
                  if(pos==-1){
                        p[0]=(nearp[0][0]+nearp[1][0])/2;
                        p[1]=(nearp[0][1]+nearp[1][1])/2;
                  }
                  else{
                        pthread_mutex_lock(&lock);
                        p[0]=r->c[i].x;
                        p[1]=r->c[i].y;
                        pthread_mutex_unlock(&lock);
                  }
            }
            else{
                  pthread_mutex_lock(&lock);
                  p[0]=r->c[i].x;
                  p[1]=r->c[i].y;
                  pthread_mutex_unlock(&lock);
            }
      }

}//else is=true!

}//exoteriko else!

r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;
break;
```

B.2.15 Διαδικασία ΌχιΚολλημένο

```
float points[MAX][2];
int count=findPoint(r->c[i].ch,points,r->c[i].chNumV);
if(count>=1)
r->c[i].statec=16;
else
r->c[i].statec=15;
break;
```

B.2.16 Διαδικασία ΘαΠροκαλέσειΑλλαγή

```
int k,table[MAX];
double din;
k=0;
for(j=0;j<r->c[i].numV;j++){
if(j==(r->c[i].numV)-1)
      din=findDistanceOfTwoPoints(r->c[i].visible[j],r-
>c[i].visible[0]);
else
      din=findDistanceOfTwoPoints(r->c[i].visible[j],r-
>c[i].visible[j+1]);
```

```
if(din>=2.0){
      table[k]=1;
      k++;
}
}

if(k==0){
pthread_mutex_lock(&lock);
p[0]=r->c[i].x;
p[1]=r->c[i].y;
pthread_mutex_unlock(&lock);
}
else{
int minp1,minp2;
minp1=findNearestPoint(r->c[i].x,r->c[i].y,r->c[i].visible,-1);

minp2=findNearestPoint(r->c[i].x,r->c[i].y,r-
>c[i].visible,minp1);

p[0]=((r->c[i].visible[minp1][0])+(r-
>c[i].visible[minp2][0]))/2;
p[1]=((r->c[i].visible[minp1][1])+(r-
>c[i].visible[minp2][1]))/2;
}

r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;
break;
}
```

Β.2.17 Διαδικασία ΔενΘαΠροκαλέσειΑλλαγή

```
float points[MAX][2],c[2];
int minp;
double mind;

int count=findPoint(r->c[i].ch,points,r->c[i].chNumV);
c[0]=r->c[i].x;
c[1]=r->c[i].y;
mind=findDistanceOfTwoPoints(c,points[0]);
minp=0;
for(j=1;j<count;j++){
double d1=findDistanceOfTwoPoints(c,points[j]);
if(mind>d1){
      mind=d1;
      minp=j;
}

}

//to pio kontino simio sto ci einai to point[minp]
//thelo na vro to left kai to right t sto convex hull
int pos=0;
float mid[2],point[2];
for(j=1;j<r->c[i].chNumV;j++){
mid[0]=(r->c[i].ch[j-1][0]+r->c[i].ch[j][0])/2;
mid[1]=(r->c[i].ch[j-1][1]+r->c[i].ch[j][1])/2;
```

```
point[0]=mid[0]+((1.0/MAX)*cos(90.0));
point[1]=mid[1]+((1.0/MAX)*sin(90.0));
if(point[0]==points[minp][0] && point[1]==points[minp][1])
      pos=j-1;

}

mid[0]=(r->c[i].ch[pos][0]+r->c[i].ch[pos+1][0])/2;
mid[1]=(r->c[i].ch[pos][1]+r->c[i].ch[pos+1][1])/2;

p[0]=mid[0]+(0.1*cos(90.0));
p[1]=mid[1]+(0.1*sin(90.0));
r->c[i].statec=-1;
r->c[i].state='m';
flagc=false;
break;
}
}
}
break;
```

Παράρτημα Γ

Στο παράρτημα αυτό θα δοθεί λεπτομερής κώδικας για τις βοηθητικές συναρτήσεις που χρησιμοποιήθηκαν στο πρόγραμμα, που ουσιαστικά βοήθησαν σε κάποιες μαθηματικές πράξεις που είχε ο αλγόριθμος.

Γ.1 ΌχιΣτοΟπτικόΠεδίο

```
bool isNotInVisible(float v[MAX][2],float c[2],int count,int inside){
	if(inside==0 && count==0)
		return true;

	int i;
	bool flag=true;
	for(i=0;i<count;i++){
		if(c[0]==v[i][0] && c[1]==v[i][1]){
			flag=false;
			break;
		}
	}

	if(flag==false)
		return false;
	else
		return true;
}
```

Γ.2 ΕύρεσηΕυθείας

```
float findLine(float c1[2], float c2[2],int c){
	float a,b,d,e;

	d=(c2[1]-c1[1]);
	e=(c2[0]-c1[0]);
	if(e==0 && c==0){
		return -1;
	}
	else if(e==0 && c==1){
		return c1[0];
	}
	else if(e!=0 && c==0){
		a=d/e;
		return a;
	}
	else{
		a=d/e;
		b= c1[1]-(a*c1[0]);

		return b;
	}
}
```

Γ.3 ΣημείοΤομής

```
float intersectionPoint(float c1[2], float c2[2],float c3[2], float c4[2],int c){
	float p[2],d,pre, post;

	d=((c1[0]-c2[0])*(c3[1]-c4[1]))-((c1[1]-c2[1])*(c3[0]-c4[0]));
	if(d==0)
		return -500.0;

	pre=(c1[0]*c2[0])-(c1[1]*c2[1]);
	post=(c3[0]*c4[0])-(c3[1]*c4[1]);

	p[0]=(pre*(c3[0]-c4[0])-((c1[0]-c2[0])*post));
	p[1]=(pre*(c3[1]-c4[1])-((c1[1]-c2[1])*post));

	if(c==0)
		return p[0];
	else
		return p[1];
}
```

Γ.4 ΕύρεσηΑπόστασηςΜεταξύΕυθείαςΚαιΣημείου

```
double foundDistanceofPointFromLine(float c1[2],float c2[2], float c[2]){
	float A[2],B[2], AB,AP,BP,a,b;
	double PC;

	a=findLine(c1,c2,0);
	b=findLine(c1,c2,1);

	A[0]=c[0];
	A[1]=(a*A[0])+b;
	B[1]=c[1];
	B[0]=(B[1]-b)/a;

	AB=sqrt(pow(A[0]-B[0],2) + pow(A[1]-B[1],2));
	AP=sqrt(pow(c[0]-A[0],2) + pow(c[1]-A[1],2));
	BP=sqrt(pow(c[0]-B[0],2) + pow(c[1]-B[1],2));

	PC=(AP*BP)/AB;

	return PC;
}
```

Γ.5 ΕύρεσηΑπόστασηςΜεταξύΔύοΣημείων

```
double findDistanceOfTwoPoints(float c1[2],float c2[2]){
	double d;

	d=sqrt(pow(c2[0]-c1[0],2) + pow(c2[1]-c1[1],2));

	return d;
}
```

Γ.6 ΠάνωΣτηνΕυθεία

```
bool isOnLine(float a, float b, float c[2]){
	if(a==-1){
		if(c[0]==b)
			return true;
		else
			return false;
	}

	float d,e;
	d=c[0] * a;
	e=d+b;

	if(c[1]==e)
		return true;
	else
		return false;

}
```

Γ.7 Εφάπτονται

```
bool areTangent(float x, float y, float p[2]){
	double a= sqrt(x-p[0])+sqrt(y-p[1]);

	if(a==0)
		return true;
	else
		return false;
}
```

Γ.8 ΕύρεσηΚοντινότερουΣημείου

```
int findNearestPoint(float x, float y,float visible[MAX][2],int pos){
	float pc[2];
	double dif,mind;
	int minp,j;
	pc[0]=x;
	pc[1]=y;

	if(pos==-1){ //kalite gia proti fora i sinartisi kai thelo to pio kontino
robot
		mind=findDistanceOfTwoPoints(pc,visible[0]);
		for(j=1;j<MAX;j++){
			dif=findDistanceOfTwoPoints(pc,visible[j]);
			if(dif<mind){
				mind=dif;
				minp=j;
			}
		}
	}
	else{//kalite gia na vro to deftero stoixeio!
		if(pos==0){
```

```
                        mind=findDistanceOfTwoPoints(pc,visible[1]);
                        for(j=2;j<MAX;j++){
                                dif=findDistanceOfTwoPoints(pc,visible[j]);
                                if(dif<mind){
                                        mind=dif;
                                        minp=j;
                                }
                        }
                }
                else{
                        mind=findDistanceOfTwoPoints(pc,visible[0]);
                        for(j=1;j<MAX;j++){
                                if(j!=pos){
                                        dif=findDistanceOfTwoPoints(pc,visible[j]);
                                        if(dif<mind){
                                                mind=dif;
                                                minp=j;
                                        }
                                }
                        }
                }
        }

        return minp;

}
```

Γ.9 ΠάνωΣτοΠολύγωνο

```
bool onPolygon(int nvert, float *vertex, float *vertey,float testx, float testy){
        bool c=false;
        int i,j;

        for(i=0,j=nvert-1;i<nvert;j=i++){
                if(((vertey[i]>testy)!=(vertey[j]>testy)) && (testx<((vertex[j]-
vertex[i])*((testy-vertex[i])/(vertey[j]-vertey[i]))+vertex[i])))
                        c=!c;
        }
        return c;
}
```

Γ.10 ΕύρεσηΚολλημένωνΣημείων

```
int findTouchingPoints(float c[2], float v[MAX][2], float tpoints[][2],int pos){
        int count=0,i;
        for(i=0;i<pos;i++){
                bool is=areTangent(c[0],c[1],v[i]);
                if(is==true){
                        tpoints[count][0]=v[i][0];
                        tpoints[count][1]=v[i][1];
                        count++;
                }
        }
        return count;
}
```

Γ.11 ΕύρεσηΜικρότερηςΑπόστασης

```
double findMinDistance(float tpoints[][2], float c[2], int count){
	double mind=findDistanceOfTwoPoints(tpoints[0],c),d;
	int i;
	for(i=0;i<count;i++){
		d=findDistanceOfTwoPoints(tpoints[i],c);
		if(d<mind)
			mind=d;
	}
	return mind;
}
```

Γ.12 ΕύρεσηΣημείωνΊδιαςΑπόστασης

```
int findSameDistance(float p[2],float tpoints[100][2], float spoints[100][2],int
count,double din){
	int count1=0,i;
	double distance;
	for(i=0;i<count;i++){
		distance=findDistanceOfTwoPoints(p,tpoints[i]);
		if(distance==din){
			spoints[count][0]=tpoints[i][0];
			spoints[count][1]=tpoints[i][1];
			count1++;
		}
	}
	return count1;
}
```

Γ.13 ΕύρεσηΑριστερότερουΣημείου

```
int findRightMostPoint(float p[2],float spoints[][2],int count){
	float maxx=p[0];
	int maxp=-1,i;
	for(i=0;i<count;i++){
		if(spoints[i][0]>maxx){
			maxx=spoints[i][0];
			maxp=i;
		}
	}
	return maxp;
}
```

Γ.14 ΕύρεσηΔύοΚοντινότερωνΣημείων

```
bool findNearestTwoPoints(float c[2],float nearp[2][2],float ch[MAX][2], int
num){
	int count=-1,i;
```

```
	double distance[MAX],d;

	for(i=0;i<num;i++){
		d=findDistanceOfTwoPoints(c,ch[i]);
		if(d>=2){
			count++;
			distance[count]=d;
		}
	}

	if(count == -1)
		return false;
	else{
		double mind=distance[0],mind1;
		int minp=0,minp1;
		for(i=1;i<count;i++){
			if(mind>distance[i]){
				mind=distance[i];
				minp=i;
			}
		}

		if(minp==0){
			mind1=distance[1];
			minp1=1;
			for(i=2;i<count;i++){
				if(mind>distance[i]){
					mind=distance[i];
					minp=i;
				}
			}
		}
		else{
			double mind1=distance[0];
			int minp1=0;
			for(i=1;i<count;i++){
				if(mind>distance[i] && i!=minp){
					mind=distance[i];
					minp=i;
				}
			}
		}

	nearp[0][0]=ch[minp][0];
	nearp[0][1]=ch[minp][1];
	nearp[1][0]=ch[minp1][0];
	nearp[1][1]=ch[minp1][1];

	}

	return true;
}
```

Γ.15 ΕύρεσηΕφαπτόμενηςΕυθείας

```
bool findTangentLine(float c1[2],float c2[2],float r){
	double d_sq=(c1[0]-c2[0])*(c1[0]-c2[0])+(c1[1]-c2[1])*(c1[1]-c2[1]);

	if(d_sq<=0)
```

```
		return false;
	else
		return true;

}
```

Γ.16 ΔενΒρίσκεταιΣεΣυγκέντρωση

```
bool isNotInComponent(float v[2],float components[MAX][2],int count){
	int i;
	bool flag=true;

	for(i=0;i<count;i++){
		if(v[0]==components[i][0] && v[1]==components[i][1]){
			flag=false;
			break;
		}
	}

	if(flag==false)
		return false;
	else
		return true;

}
```

Γ.17 ΕύρεσηΣημείουΤομήςΕυθείαςΚαιΆξοναΧ

```
float findPointIntersectsTheXaxis(float a,float c[2]){
	float b=c[1]-(a*c[0]);
	float x;
	if(a!=0)
		x=-b/a;
	else
		x=-b;

	return x;

}
```

Γ.18 ΕύρεσηΓωνιαςΔυοΕυθειών

```
double getAngleofTwoLines(float c1[2], float c2[2],float c3[2], float c4[2]){
	double d,dx1,dx2,dy1,dy2,l2;
	double angle;

	dx1=c2[0]-c1[0];
	dx2=c3[0]-c4[0];
	dy1=c2[1]-c1[1];
	dy2=c3[1]-c4[1];

	d=dx1*dx2+dy1*dy2;
	l2=((dx1*dx1)+(dy1*dy1))*((dx2*dx2)+(dy2*dy2));
```

```
	angle=acos(d/sqrt(l2));
	return angle;

}
```

Γ.19 ΕύρεσηΣυγκέντρωσης

```
int findComponent(float cc[MAX][6],int size){
	int i;
	for(i=0;i<size;i++){
		if(cc[i][5]==1.0)
			break;
	}


	return i;

}
```

Γ.20 ΜέτρησηΣυγκεντρώσεων

```
int countComponents(int r, int l, int num){
	int i,pos,pos1,count=1;

	for(i=0;i<num;i++){
		pos=(r+i)%num;
		if(pos==l)
			break;

		pos1=(r+i+1)%num;
		count++;
	}

	return count;

}
```

Γ.21 ΒρίσκεταιΣταΌριαΤωνΣυγκεντρώσεων

```
bool isInComponentLimits(int r,int l,int t, int num){
	int i,pos;
	for(i=0;i<num;i++){
		pos=(r+i)%num;

		if(pos==t)
			return true;

		if(pos==l && pos==t)
			return true;

		if(pos==l && pos!=t)
			return false;
	}
```

```
	return false;
}
```

Γ.21 ΕίναιΤοCΣταΌριαΤηςΣυγκέντρωσης

```
float isCInComponentLimits(int r,int l,int t, int num){
	int i,pos;
	for(i=0;i<num;i++){
		pos=(r+i)%num;

		if(pos==t)
			return 1.0;

		if(pos==l && pos==t)
			return 1.0;

		if(pos==l && pos!=t)
			return 0.0;
	}

	return 0.0;
}
```