Ατομική Διπλωματική Εργασία

# ΑΛΓΟΡΙΘΜΟΙ ΕΛΕΓΧΟΥ ΚΙΝΗΤΙΚΟΤΗΤΑΣ ΣΕ ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ (WSN)

**Κυριάκος Πέτρου**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**Μάιος 2012**

# ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

## ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ


**Αλγόριθμοι Ελέγχου Κινητικότητας σε**
**Ασύρματα Δίκτυα Αισθητήρων (WSN)**


**Κυριάκος Πέτρου**

Επιβλέπων Καθηγητής
Βάσος Βασιλείου



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

Μάιος 2012

## Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Βάσο Βασιλείου για την υποστήριξη και καθοδήγηση που μου πρόσφερε κατά την διάρκεια της υλοποίησης αυτής της εργασίας, αλλά και για την επίτευξη μιας άψογης συνεργασίας. Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου που με στήριξε καθ' όλη την διάρκεια των σπουδών μου.

## Περίληψη

Τα ασύρματα δίκτυα αισθητήρων είναι ένα από τα πιο σημαντικά θέματα μελέτης τα τελευταία χρόνια. Έχουν υποστεί μεγάλη ανάπτυξη χάρη στις καινούριες τεχνολογίες τόσο υλικά όσο και λογισμικά. Παρόλα αυτά όμως εξακολουθούν να αντιμετωπίζουν αρκετά προβλήματα τα οποία τα αποτρέπουν από το να χρησιμοποιηθούν σε ένα μεγαλύτερο φάσμα εφαρμογών. Οι ειδικοί στο θέμα, επειδή γνωρίζουν τις τεράστιες προοπτικές που έχουν τα ασύρματα δίκτυα αισθητήρων, συνεχίζουν τις έρευνες και μελέτες για καλυτέρευση τους. Σε αυτή τη διπλωματική εργασία θα ασχοληθώ με θέματα που έχουν να κάνουν με έλεγχο κινητικότητας σε Ασύρματα Δίκτυα Αισθητήρων. Πιο συγκεκριμένα θα αναπτύξω δύο αλγόριθμους για ομαδοποίηση αισθητήρων και για κίνηση αισθητήρων σε δρομολόγιο με βάση την κίνηση που κάνει ο αρχηγός. Θα αναπτυχθούν δυο εφαρμογές, οι οποίες θα στηρίζονται στους πιο πάνω αλγορίθμους. Η πρώτη εφαρμογή αφορά την εύρεση κοινών φίλων («Find your friends»). Αυτή η εφαρμογή αφορά την συμπεριφορά των κινητών κόμβων (mobile nodes) σε στρατιωτικό περιβάλλον όπου μη επανδρωμένα οχήματα προσπαθούν να συγκλίνουν σε μια περιοχή έτσι ώστε να ομαδοποιηθούν, καθώς την ίδια στιγμή προσπαθούν να αποφύγουν τον εχθρό. Προσομοιώνοντας αυτή την εφαρμογή θα μας αποδείξει την αυτονομία που έχουν οι κόμβοι αισθητήρων σε ένα Ασύρματο Δίκτυο Αισθητήρων, με το να κινούνται προς μια κατεύθυνση, δηλαδή αλλάζοντας την θέση τους, με σκοπό να συγκλίνουν όλοι οι αισθητήρες σε ένα τυχαίο σημείο. Η δεύτερη εφαρμογή αφόρα την κίνηση του αρχηγού (leader sensor) και την ακολουθία όλων των υπόλοιπων αισθητήρων στην κίνηση που έκανε ο αρχηγός («Follow the leader»). Αυτή η εφαρμογή αφόρα στην προσομοίωση μιας πιθανής χρησιμοποίησης των Ασύρματων Δικτύων Αισθητήρων σε ένα Ευφυή Σύστημα Μεταφοράς (Intelligent Transportation System) όπου τα οχήματα σε ένα αυτοκινητόδρομο (highway) ακολουθούν το ένα το άλλο αυτόματα και δεν χάνουν τα οχήματα που βρίσκονται μπροστά τους. Αυτό, επίσης, μπορεί να χρησιμοποιηθεί στο να δείξουμε την σημαντικότητα του να διατηρούμε την σωστή απόσταση από το όχημα μπροστά μας ούτος ώστε να αποφεύγουμε τα τροχαία δυστυχήματα σε μεγάλο βαθμό. Προσομοιώνοντας αυτή την εφαρμογή θα μας δείξει την αυτονομία και την συνεργασία

που έχουν οι κόμβοι αισθητήρων σε ένα Ασύρματο Δίκτυο Αισθητήρων, όπου με την χρήση των κατάλληλων αισθητήρων ακλουθούν το ένα το άλλο με βάση την κίνηση του αρχηγού τους. Επιπρόσθετα, θα μελετηθεί υπάρχουσα ερευνητική και πειραματική εργασία για την κινητικότητα σε Ασύρματα Δίκτυα Αισθητήρων και στη συνέχεια θα παρουσιαστούν αναλυτικά οι δυο αλγόριθμοι κίνησης.

# Περιεχόμενα

# Κεφάλαιο 1

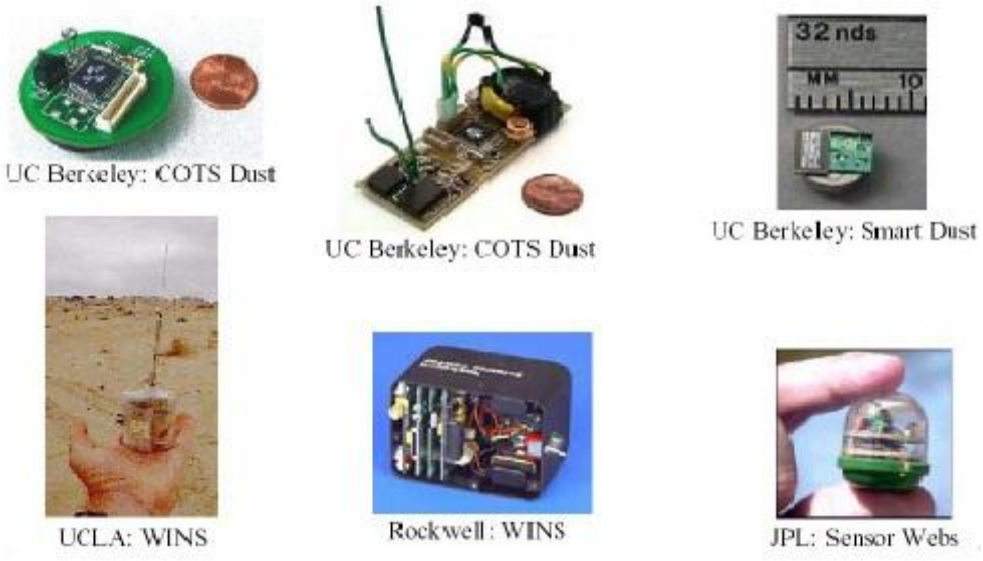
## Εισαγωγή



### 1.1 Γενικά

Ένα ασύρματο δίκτυο αισθητήρων είναι ένα δίκτυο υπολογιστών που αποτελείται από αυτόνομες συσκευές κατανεμημένες στο χώρο οι οποίες χρησιμοποιούν αισθητήρες με σκοπό τη συλλογική απεικόνιση φυσικών ή περιβαλλοντολογικών μεγεθών, όπως η θερμοκρασία, ο ήχος, η δόνηση, η πίεση και η κίνηση. Η ανάπτυξη των ασύρματων δικτύων αισθητήρων αρχικά ξεκίνησε για στρατιωτικές εφαρμογές όπως την παρακολούθηση των πεδίων βολής. Στην συνέχεια εξαιτίας της ραγδαίας ανάπτυξης των ασύρματων επικοινωνιών και των μικρομηχανικών συστημάτων (MEMs), έγινε εφικτή η κατασκευή χαμηλού κόστους, χαμηλής κατανάλωσης ενέργειας, πολυλειτουργικών και μικροσκοπικών αισθητήρων. Οι αισθητήρες αυτοί έχουν την δυνατότητα να «αισθανθούν» το περιβάλλον, να κάνουν επεξεργασία των δεδομένων και να επικοινωνήσουν μεταξύ τους σε μικρές αποστάσεις. Πλέον χρησιμοποιούνται από το ευρύτερο κοινό σε μια πληθώρα εφαρμογών, που περιλαμβάνει παρακολούθηση περιβάλλοντος και κατοικίας, ιατρικές εφαρμογές, οικιακούς και βιομηχανικούς αυτοματισμούς και έλεγχο κυκλοφορίας.

Ένα τυπικό δίκτυο αισθητήρων αποτελείται πολλές φορές από χιλιάδες τέτοιους κόμβους, κατανεμημένους στο χώρο που θα παρακολουθούν είτε τυχαία είτε σύμφωνα με κάποια προκαθορισμένη στατιστική κατανομή. Το κόστος των κόμβων είναι εξίσου κυμαινόμενο, μεταξύ μερικών χιλιάδων δολαρίων έως μερικά cents, ανάλογα με το μέγεθος του δικτύου και την πολυπλοκότητα των μεμονωμένων κόμβων. Οι περιορισμοί μεγέθους και κόστους έχουν σαν αποτέλεσμα αντίστοιχους περιορισμούς στην ενέργεια, την μνήμη, την υπολογιστική ισχύ και το εύρος ζώνης.




Σχήμα 1.1: Διάφορα Είδη Αισθητήρων

Οι εφαρμογές των δικτύων αισθητήρων είναι πολλές και διάφορες. Χρησιμοποιούνται σε εμπορικές και βιομηχανικές εφαρμογές για να συλλέξουν πληροφορίες που θα ήταν είτε δύσκολο είτε οικονομικά ασύμφορο να συλλεχτούν χρησιμοποιώντας καλωδιωμένους αισθητήρες. Μπορούν να εγκατασταθούν σε απομακρυσμένες περιοχές όπου θα μπορούσε να παραμείνουν για μεγάλο χρονικό διάστημα καταγράφοντας ένα περιβαλλοντολογικό μέγεθος χωρίς να χρειάζεται να αντικατασταθεί ή να φορτισθεί η πηγή ενέργειας τους. Μπορούν να τοποθετηθούν περιμετρικά σε μια ιδιοκτησία και να παρακολουθούν την κίνησης εισβολέων μεταδίδοντας πληροφορία από κόμβο σε κόμβο.

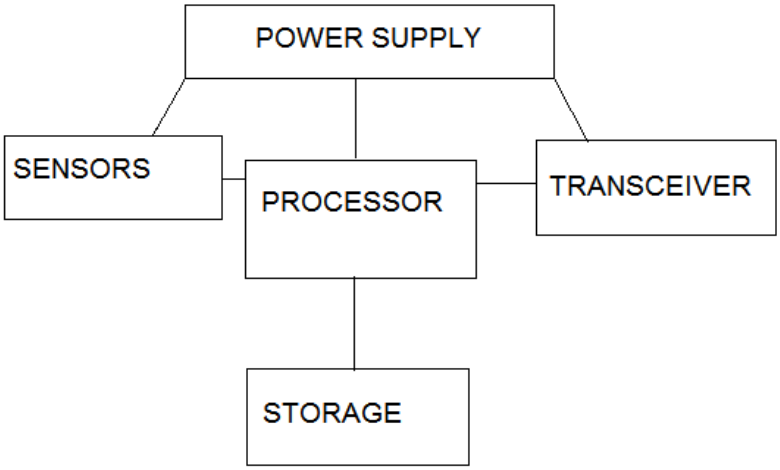
Κάθε δίκτυο περιγράφεται από ένα σύνολο χαρακτηριστικών. Τέτοια είναι το μέγεθος του δικτύου (αριθμός κόμβων), το είδος των κόμβων, ο χώρος που καταλαμβάνει, η τοπολογία του, το μέσο μετάδοσης και τα πρωτόκολλα επικοινωνίας. Μερικά χαρακτηριστικά των δικτύων αυτών είναι τα πιο κάτω:

- Οι κόμβοι κάνουν μετρήσεις στο περιβάλλον και γεγονότα που συμβαίνουν σε αυτό μπορούν να ενεργοποιήσουν συγκεκριμένη μεταφορά δεδομένων στο δίκτυο.
- Οι κόμβοι είναι συνήθως μικροί σε μέγεθος και όμοιοι.
- Πηγές ενέργειας περιορισμένης αντοχής.
- Οι κόμβοι μπορεί να τοποθετηθούν και να μείνουν χωρίς επιτήρηση ή συντήρηση για μεγάλο χρονικό διάστημα.
- Ο χρόνος ζωής των κόμβων εξαρτάται από την χρήση.
- Μεγάλη πυκνότητα κόμβων και περιοχή μετάδοσης που δεν ξεπερνά τα 30 μέτρα.
- Μικρή υπολογιστική ισχύς και περιορισμένη μνήμη.
- Οι κόμβοι μπορεί να μην έχουν καμία δραστηριότητα για μεγάλο χρονικό διάστημα.
- Η επικοινωνία βασίζεται στα δεδομένα.
- Μικρή ροή κυκλοφορίας, κυρίως κατά την εμφάνιση συγκεκριμένων γεγονότων
- Εύρος ζώνης που δεν ξεπερνά τα 250Kbps.
- Η λειτουργία του δικτύου καθορίζεται από την εργασία που πρέπει να πραγματοποιηθεί.

### 1.2 Κόμβοι Αισθητήρων στα Ασύρματα Δικτύα Αισθητήρων

Εκτός από τους αισθητήρες, κάθε κόμβος σε ένα δίκτυο αισθητήρων είναι εξοπλισμένος με ένα αναμεταδότη, έναν μικροεπεξεργαστή, μία μνήμη και μια πηγή ενέργειας, συνήθως μπαταρία. Οι αισθητήρες παράγουν ηλεκτρικά σήματα τα οποία είναι ανάλογα των φυσικών μεγεθών που θα λάβουν μετρήσεις. Ο μικροεπεξεργαστής αναλαμβάνει να επεξεργαστεί και να αποθηκεύσει τις πληροφορίες που λαμβάνει από τον αισθητήρα. Στην συνέχεια ο αναμεταδότης αναλαμβάνει την μετάδοση της πληροφορίας προς άλλους

κόμβους ή τον κεντρικό σταθμό βάσης(sink) και λαμβάνει επίσης δεδομένα από άλλους κόμβους ή και τον ίδιο τον sink.




Σχήμα 1.2: Βασική δομή αρχιτεκτονικής ενός κόμβου-αισθητήρα [18]

Τα κύρια μέρη ενός κόμβου-αισθητήρα, είναι ο μικροεπεξεργαστής, ο πομποδέκτης, η εξωτερική μνήμη, η πηγή ενέργειας και ένας ή περισσότεροι αισθητήρες [12]:

- Ο **μικροεπεξεργαστής (microprocessor)** είναι ο πυρήνας του κόμβου. Δέχεται τα δεδομένα από τους αισθητήρες ή από τους άλλους κόμβους, τα επεξεργάζεται και αποφασίζει πότε και που θα τα στείλει. Χρησιμοποιούνται μικροεπεξεργαστές επειδή είναι η καλύτερη λύση όσο αφορά εξοικονόμηση ενέργειας, αφού μπορούμε να τους προγραμματίσουμε εύκολα να μπαίνουν σε κατάσταση ύπνου (sleep mode), όταν δεν είναι ανάγκη να δουλέψουν και έτσι δεν σπαταλιέται άσκοπα ενέργεια.
- Ο **πομποδέκτης (transceiver)** είναι υπεύθυνος για την αμφίδρομη επικοινωνία των κόμβων. Εκτός από την απλή μετάδοση των δεδομένων, πολλοί πομποδέκτες αναλαμβάνουν και άλλες εργασίες που αφορούν το πρωτόκολλο επικοινωνίας, μειώνοντας έτσι το φόρτο εργασίας του μικροεπεξεργαστή. Υπάρχουν διάφοροι τρόποι επικοινωνίας μεταξύ των κόμβων. Οι πιο διαδεδομένοι είναι με ραδιοσυχνότητα (RF), υπέρυθρες ακτίνες (infrared) και laser. Ο πιο κατάλληλος για επικοινωνία από τους τρεις τρόπους

είναι με την χρήση των ραδιοσυχνοτήτων. Οι άλλοι δύο τρόποι, αν και δεν χρειάζονται αντένα για να αποστείλουν και να λάβουν τα δεδομένα, εντούτοις έχουν περιορισμένη ραδιοφωνική αναμετάδοση και χρειάζονται αρκετές ρυθμίσεις για να μπορούν να λειτουργήσουν καλά (ευθυγράμμιση κατά κύριο λόγο). Από την άλλη με τις ραδιοσυχνότητες μπορούμε να έχουμε μετάδοση και παραλαβή πακέτων προς όλες τις κατευθύνσεις χωρίς πρόβλημα.

- Η **μνήμη** σε ένα κόμβο αισθητήρα χρησιμοποιείται για να προγραμματίσουμε τον κόμβο, για να φυλάξει δεδομένα της εφαρμογής καθώς και για να αποθηκεύονται εκεί αναγνωριστικά του κόμβου. Συνήθως η μνήμη αυτή εμφανίζεται σαν ενσωματωμένη στον μικροεπεξεργαστή ή ακόμα και σαν επιπρόσθετη flash μνήμη.
- Η **πηγή ενέργειας** σε ένα κόμβο αισθητήρα μπορεί να είναι είτε κάποια μπαταρία είτε κάποιοι πυκνωτές που συγκρατούν ενέργεια. Τα τελευταία χρόνια έχουν αναπτυχθεί κόμβοι με τη δυνατότητα να παίρνουν ενέργεια από το περιβάλλον τους και να ανανεώνουν τα αποθέματα που έχουν, όπως με χρήση της ηλιακής ενέργειας.

Όσον αφορά τους πομποδέκτες, αυτοί έχουν τέσσερις βασικές λειτουργικές καταστάσεις:

- **Κατάσταση Εκπομπής (Transmit state):** ο πομποδέκτης είναι ενεργός και αποστέλλει τα δεδομένα που έχει ο κόμβος αισθητήρα. Βασικά ενεργό είναι το μέρος που στέλνει τα δεδομένα και όχι το μέρος που λαμβάνει.
- **Κατάσταση Λήψης (Receive state):** πάλι ο πομποδέκτης είναι ενεργός, όμως σε αυτή την περίπτωση ενεργό είναι το μέρος του πομποδέκτη που είναι υπεύθυνο για την λήψη μηνυμάτων από τους άλλους κόμβους. Έτσι γίνεται η παραλαβή των δεδομένων.
- **Ανενεργή Κατάσταση (Idle state):** Σε αυτή την κατάσταση ο πομποδέκτης αν και είναι έτοιμος να λάβει δεδομένα, δεν λαμβάνει τίποτα. Ορισμένα μέρη του είναι ενεργά αλλά και αρκετά είναι τελείως ανενεργά.
- **Κατάσταση Ύπνου (Sleep state):** εδώ έχουμε τα περισσότερα μέρη του πομποδέκτη ανενεργά. Σε αντίθεση με την idle state, εδώ ο πομποδέκτης δεν είναι έτοιμος να λάβει δεδομένα. Ξυπνά όμως ανά τακτά χρονικά διαστήματα και βλέπει αν υπάρχει τίποτα που τον ενδιαφέρει και μετά ξαναμπαίνει στην κατάσταση ύπνου, για όσο το δυνατόν μεγαλύτερη εξοικονόμηση ενέργειας.

Επιπρόσθετα μπορούμε να κατατάξουμε τους αισθητήρες σε τρεις ομάδες:

- **Παθητικοί - omnidirectional:** είναι οι αισθητήρες που μετρούν ένα φυσικό μέγεθος προς όλες τις κατευθύνσεις, χωρίς να επέμβουν στο περιβάλλον.
- **Παθητικοί – narrow-beam:** σε αντίθεση με την πρώτη ομάδα, σε αυτή το φυσικό μέγεθος που μετριέται περιορίζεται σε μια γωνία ή κατεύθυνση. Πιο χαρακτηριστικό παράδειγμα είναι όταν έχουμε μια κάμερα που κοιτάζει μόνο σε ένα χώρο.
- **Ενεργητικοί:** αντίθετα με τις δύο πιο πάνω ομάδες, αυτοί οι αισθητήρες για να πάρουν μια μέτρηση πρέπει να επέμβουν στο περιβάλλον (πχ. Αισθητήρες σεισμικών δονήσεων).

## 1.3 Τα Στρώματα στα Ασύρματα Δικτύα Αισθητήρων

Στα ασύρματα δίκτυα αισθητήρων είναι δομημένα σε στρώματα (layers) για την σωστή οργάνωση και σχεδίαση πρωτοκόλλων [10].

- **Application Layer:** Παρόλο που έχουν οριστεί πολλές περιοχές εφαρμογών για ασύρματα δίκτυα αισθητήρων, δεν έχει ερευνηθεί μεγάλος αριθμός πρωτοκόλλων για το application layer. Μερικά πρωτόκολλα του application layer τα οποία βρίσκονται υπό έρευνα είναι: Sensor Management Protocol (SMP), Task Assignment and Data advertisement Protocol (TADAP) και το Sensor Query and Data Dissemination Protocol (SQDDP)
- **Transport Layer:** Αυτό το layer είναι χρήσιμο κυρίως όταν το σύστημα είναι σχεδιασμένο έτσι ώστε να μπορεί να έχει πρόσβαση στο διαδίκτυο ή σε άλλα εξωτερικά δίκτυα. Μια προσέγγιση παρόμοια με αυτήν του TCP μπορεί να φανεί χρήσιμη για την επικοινωνία με άλλου είδους δίκτυα όπως το διαδίκτυο. Η επικοινωνία μεταξύ κόμβων του δικτύου δεν μπορεί να υποστηριχθεί με πρωτόκολλα των οποίων η προσέγγιση είναι παρόμοια με του πρωτοκόλλου TCP, λόγω της περιορισμένης μνήμης που διαθέτουν οι κόμβοι αισθητήρων.

- **Network layer:** Παραδοσιακές ad-hoc τεχνικές δρομολόγησης συνήθως δεν ταιριάζουν στις απαιτήσεις των ασύρματων δικτύων αισθητήρων. Το network layer των δικτύων αισθητήρων είναι σχεδιασμένο σύμφωνα με τις πιο κάτω αρχές:
  - Πρέπει να διαχειρίζονται αποδοτικά τα διαθέσιμα ποσά ενέργειας τους.
  - Πρέπει να είναι data-centric.
  - Η συνάθροιση των δεδομένων είναι χρήσιμη μόνο όταν δεν εμποδίζεται η συνεργασία μεταξύ των κόμβων αισθητήρων.
  - Σε ένα ιδανικό δίκτυο αισθητήρων, οι διευθύνσεις των κόμβων είναι attribute-based και δεν γνωρίζουν την ακριβή τοποθεσία τους οι κόμβοι.

Τα δρομολόγια τα οποία είναι αποδοτικά σε ενέργεια (energy efficient) μπορούν να βρεθούν βάση της διαθέσιμης εναπομένουσας ενέργειας σε κάθε κόμβο (available power - PA) ή της ενέργειας που απαιτείται για τη μετάδοση των δεδομένων μέσω των δρομολογίων. Οι πιο γνωστοί αλγόριθμοι για την επιλογή του πιο αποδοτικού δρομολογίου όσον αφορά την ενέργεια είναι οι πιο κάτω:

- **Maximum PA route:** Επιλέγεται το δρομολόγιο το οποίο έχει τη μεγαλύτερη συνολική εναπομένουσα ενέργεια, η οποία υπολογίζεται με το άθροισμα της εναπομένουσας ενέργειας του κάθε κόμβου ο οποίος ανήκει στο συγκεκριμένο δρομολόγιο.
- **Minimum energy (ME) route:** Επιλέγεται το δρομολόγιο το οποίο απαιτεί την ελάχιστη ενέργεια για τη μεταφορά των πακέτων μεταξύ των sinks και των κόμβων αισθητήρων.
- **Minimum hop route (MH):** Επιλέγεται το δρομολόγιο το οποίο θα έχει τον ελάχιστο αριθμό hops μέχρι το sink.
- **Maximum minimum PA note route**: Επιλέγεται το δρομολόγιο του οποίου η ελάχιστη εναπομένουσα ενέργεια θα είναι μεγαλύτερη από την ελάχιστη εναπομένουσα ενέργεια των υπολοίπων δρομολογίων.

Μια από τις υπευθυνότητες του network layer είναι να παρέχει διασύνδεση με εξωτερικά δίκτυα (πχ. άλλα δίκτυα αισθητήρων ή το διαδίκτυο). Σε ένα σενάριο το sink μπορεί να

χρησιμοποιηθεί ως το gateway προς άλλα δίκτυα.

- **Data link layer:** Το data link layer είναι υπεύθυνο να κάνει multiplexing την ροή των δεδομένων, να ανιχνεύει τα data frames καθώς και για medium access και error control. Επίσης, διαφυλάσσει αξιόπιστες point to point και point to multipoint επικοινωνίες στο δίκτυο. Το πρωτόκολλο MAC, σε ένα ασύρματο multihop και αυτό-οργανωτικό δίκτυο αισθητήρων, έχει ως στόχους τη δημιουργία της υποδομής του δικτύου και το πιο δίκαιο και αποδοτικό διαχωρισμό των πόρων επικοινωνίας μεταξύ των κόμβων του δικτύου. Οι δύο γνωστοί μέθοδοι για error control είναι οι forward error correction (FEC) και automatic repeat request (ARQ). Η χρησιμότητα του ARQ σε multihop δίκτυα αισθητήρων είναι περιορισμένη από το επιπλέον κόστος ενέργειας αναμετάδοσης και το overhead. Από την άλλη, η πολυπλοκότητα αποκωδικοποίησης του FEC είναι μεγαλύτερη αφού απαιτείται η ενσωμάτωση ικανοτήτων error control.
- **Physical layer:** Το Physical layer είναι υπεύθυνο για την επιλογή της συχνότητας, την παραγωγή συχνότητας μετάδοσης, την ανίχνευση σήματος, τη διαμόρφωση του και για την κρυπτογράφηση των δεδομένων. Μέχρι σήμερα, είναι από όλους αποδεκτές οι συχνότητες των 816 MHz, 915 MHz και 2.4 GHz ISM για τα ασύρματα δίκτυα αισθητήρων.
- **Power management plane:** Είναι υπεύθυνο για τη διαχείριση της ενέργειας ενός κόμβου αισθητήρα. Για παράδειγμα, για την αποφυγή παραλαβής διπλών μηνυμάτων, ο κόμβος αισθητήρα μπορεί να κλείσει το δέκτη του μετά από παραλαβή ενός μηνύματος από κάποιο από τους γείτονες του. Ακόμη, όταν ο κόμβος αισθητήρα δεν έχει μεγάλα αποθέματα ενέργειας, ενημερώνει τους γείτονες του για να μην συμμετάσχει σε μελλοντική δρομολόγηση πακέτων και έτσι η εναπομένουσα ενέργεια που διαθέτει θα χρησιμοποιείται μόνο για την ανίχνευση γεγονότων.
- **Mobility management plane:** Εντοπίζει και καταγράφει την κίνηση των κόμβων αισθητήρων έτσι ώστε η δρομολόγηση πίσω προς το χρήστη να είναι πάντα δυνατή και έτσι ώστε οι κόμβοι να γνωρίζουν ποιοι είναι οι γείτονές τους.
- **Task management plane:** Ισοζυγίζει και προγραμματίζει τις διεργασίες για αίσθηση σε μια συγκεκριμένη περιοχή. Η ευελιξία, η μεγάλη ανοχή σε λάθη, η υψηλή αξιοπιστία αίσθησης, το μικρό κόστος και η γρήγορη ανάπτυξη ενός δικτύου αισθητήρων

είναι μερικοί από τους σημαντικότερους παράγοντες που δημιουργούν πολλές καινούριες και συναρπαστικές περιοχές για ανάπτυξη εφαρμογών. Σύντομα αυτό το μεγάλο εύρος εφαρμογών θα κάνει τα ασύρματα δίκτυα αισθητήρων μέρος της καθημερινής μας ζωής.

### 1.4 Προβλήματα στα ασύρματα δίκτυα αισθητήρων

Τα Ασύρματα Δίκτυα Αισθητήρων έχουν πάρα πολλά προβλήματα [16]. Τα προβλήματα αυτά πηγάζουν κυρίως από τη φύση των αισθητήρων λόγω του ότι είναι μικροί σε μέγεθος και πρέπει να λειτουργούν αυτόνομα.

- Το βασικότερο πρόβλημα είναι ο περιορισμός σε πόρους. Έχουν μειωμένη ενέργεια, ισχύ στον επεξεργαστή, μνήμη και εμβέλεια. Αυτό μπορεί να εξομαλυνθεί με ένα καλό σύστημα διαχείρισης ενέργειας που θα θέτει τον αισθητήρα σε κατάσταση ύπνου (sleep mode) όταν δεν απαιτείται, να αποφασιστεί πότε θα κάνει μετάδοση με υψηλότερη ισχύ για κρίσιμα πακέτα ή ακόμα και να κινηθεί για να πάρει ένα καλύτερο σήμα.
- Εξίσου σημαντικό είναι το πρόβλημα της μη προβλεπτικότητας. Οι αισθητήρες λειτουργούν τις πλείστες φορές σε περιβάλλοντα με ανεξέλεγκτες καταστάσεις. Οι ασύρματη επικοινωνία μπορεί να έχει πολλά εμπόδια ή λάθη. Οι διασυνδέσεις και η τοπολογία του δικτύου αλλάζει δυναμικά με την μετακίνηση, την πρόσθεση ή την αφαίρεση αισθητήρων μέσα στο δίκτυο. Για να αντιμετωπιστεί αυτό το πρόβλημα πρέπει οι αισθητήρες να οργανώνονται, να βελτιστοποιούν και να διορθώνουν την κατάσταση τους χωρίς την παρέμβαση εξωτερικών παραγόντων. Κάτι αρκετά δύσκολο να υλοποιηθεί.
- Άλλο πρόβλημα προκύπτει από το ότι τα δίκτυα αυτά λειτουργούν στον πραγματικό κόσμο και χρειάζεται η άμεση ανταπόκριση τους. Η επίτευξη αυτού καθίσταται δύσκολη λόγω της μεγάλης κλίμακας του θορύβου που υπάρχει. Γίνονται έρευνες για επίλυση αυτού του προβλήματος όπως έλεγχο ανατροφοδότησης (feedback control).
- Το θέμα της ασφάλειας προβληματίζει αρκετούς. Το χαμηλό κόστος και η απλότητα των αισθητήρων τους καθιστά τρωτούς. Συγκεκριμένα επειδή χρησιμοποιούν ένα ενιαίο εύρος ζώνης (bandwidth) για να επικοινωνήσουν το κάνει πολύ εύκολο για κάποιο να «κρυφακούσει». Επίσης οποιοσδήποτε μπορεί να εισάγει δικό του κόμβο μέσα

στο δίκτυο προκαλώντας την παράλυση του δικτύου (denial of service attack). Για την επίλυση των θεμάτων ασφάλειας πρέπει να θέσουμε σε εφαρμογή διάφορα σχέδια που θα αυξάνουν την ασφάλεια αλλά όχι την πολυπλοκότητα των αισθητήρων.

- Τα ασύρματα δίκτυα αισθητήρων είναι ευάλωτα σε «θανάτους» κόμβων τους, συχνά από εξάντληση των αποθεμάτων ενέργειας ή άλλων περιβαλλοντικών αλλαγών που μπορούν να τα επηρεάσουν. Έτσι οι εφαρμογές πρέπει να είναι έτοιμες να αντιμετωπίσουν αποτελεσματικά την αλλαγή στην τοπολογία του δικτύου και να μπορούν δυναμικά να διαχειρίζονται τις απώλειες χωρίς μεγάλα προβλήματα.
- Το πόσο πυκνοί είναι οι αισθητήρες μας μέσα στο δίκτυο μπορεί επίσης να επηρεάσει την επίδοση του. Αν έχουμε περισσότερους από ότι θα έπρεπε, θα παρατηρούνται φαινόμενα συμφόρησης μέσα στο ίδιο το δίκτυο με αποτέλεσμα να χάνουμε πακέτα, άρα να έχουμε περισσότερα λάθη στα αποτελέσματα μας.

Η ατομική αυτή διπλωματική εργασία θα ασχοληθεί με δυο εφαρμογές. Η πρώτη εφαρμογή αφορά την εύρεση κοινών φίλων («Find your friends»). Αυτή η εφαρμογή αφορά την συμπεριφορά των κινητών κόμβων (mobile nodes) σε στρατιωτικό περιβάλλον όπου μη επανδρωμένα οχήματα προσπαθούν να συγκλίνουν σε μια περιοχή έτσι ώστε να ομαδοποιηθούν, καθώς την ίδια στιγμή προσπαθούν να αποφύγουν τον εχθρό. Η δεύτερη εφαρμογή αφόρα την κίνηση του αρχηγού (leader sensor) και την ακολουθία όλων των υπόλοιπων αισθητήρων στην κίνηση που έκανε ο αρχηγός («Follow the leader»). Αυτή η εφαρμογή αφόρα στην προσομοίωση μιας πιθανής χρησιμοποίησης των Ασύρματων Δικτύων Αισθητήρων σε ένα Ευφυή Σύστημα Μεταφοράς (Intelligent Transportation System) όπου τα οχήματα σε ένα αυτοκινητόδρομο (highway) ακολουθούν το ένα το άλλο αυτόματα και δεν χάνουν τα οχήματα που βρίσκονται μπροστά τους. Αυτό, επίσης, μπορεί να χρησιμοποιηθεί στο να δείξουμε την σημαντικότητα του να διατηρούμε την σωστή απόσταση από το όχημα μπροστά μας ούτος ώστε να αποφεύγουμε τα τροχαία δυστυχήματα σε μεγάλο βαθμό. Αυτές οι δύο εφαρμογές είναι πολύ σημαντικές διότι θα μας αναδείξουν την αυτονομία που έχουν οι κόμβοι αισθητήρων σε ένα Ασύρματο Δίκτυο Αισθητήρων, με το να κινούνται προς μια κατεύθυνση, δηλαδή αλλάζοντας την θέση τους, με σκοπό να

συγκλίνουν όλοι οι αισθητήρες σε ένα τυχαίο σημείο και ότι μέσα από την συνεργασία που έχουν οι αισθητήρες πάρα πολλά προβλήματα από την καθημερινότητα μας μπορούν να επιλυθούν, όπως για παράδειγμα η πρόληψη ατυχημάτων. Όλοι οι αλγόριθμοί έχουν αναπτυχθεί σε γλώσσα προγραμματισμού JAVA. Γενικά όλες τις προσομοιώσεις έχουν δημιουργηθεί πάνω στην πλατφόρμα Eclipse η οποία υποστηρίζει την γλώσσα προγραμματισμού JAVA.

# Κεφάλαιο 2

## Κινητικότητα στα Ασύρματα Δίκτυα Αισθητήρων



### 2.1 Γενικά

Στις μέρες μας όλο και πιο σύγχρονοι κόμβοι αισθητήρων αναπτύσσονται. Όμως δεν είναι μόνο οι κόμβοι που αναπτύσσονται, αλλά και γενικά τα ασύρματα δίκτυα αισθητήρων. Πλέον, έχουμε την εισαγωγή της κινητικότητας μέσα σε αυτά τα δίκτυα, πράγμα που μας δίνει ακόμη περισσότερες δυνατότητες στα χέρια μας για να τις εκμεταλλευτούμε και να βάλουμε ακόμη περισσότερο στις ζωές μας τα WSNs. Έχουμε κόμβους τώρα να εφαρμόζονται πάνω σε ρομπότ, ζώα, αυτοκίνητα, ακόμη και ανθρώπους που σκοπό έχουν την καταγραφή διαφόρων περιβαλλοντικών αλλά και βιολογικών μεγεθών για παρατήρηση και μελέτη σε πολλούς τομείς.

Η κινητικότητα μπορεί να βοηθήσει σε πολλούς τομείς ένα ασύρματο δίκτυο αισθητήρων. Μπορεί να δώσει λύσεις στα προβλήματα κάλυψης ενός δικτύου καθώς και να προσφέρει πιο εύκολα κάλυψη σε ένα δίκτυο στο οποίο δεν υπάρχουν αρκετοί

αισθητήρες για να καλύψουν στατικά ολόκληρο το χώρο που τους αναλογεί. Από την άλλη μπορούμε να συλλέξουμε πληροφορίες από απομακρυσμένους κόμβους με μια κινητή μονάδα συλλογής πληροφοριών αντί να αναγκάζουμε τους κόμβους να κάνουν μακρινές αποστολές δεδομένων με αποτέλεσμα να χάνουν άσκοπα πολύτιμη ενέργεια. Με την κίνηση ακόμη μπορούμε να ακολουθούμε ένα κινούμενο στόχο, από τον οποίο θέλουμε να συλλέξουμε πληροφορίες και όμως δεν μπορούμε να τον πλησιάσουμε για να του βάλουμε πάνω του αισθητήρες. Ακόμη μπορούμε να χρησιμοποιήσουμε ζώα σαν κινητήρια δύναμη, ενσωματώνοντας πάνω τους αισθητήρες για να μπορούμε να ελέγχουμε το φυσικό τους περιβάλλον, τα δάση για περιπτώσεις πυρκαγιάς καθώς και τα ίδια τα ζώα.

## 2.2 Μορφές κινητικότητας

Την κινητικότητα μπορούμε να την συναντήσουμε σε πολλές μορφές σε ένα ασύρματο δίκτυο αισθητήρων, όπως προαναφέρθηκε πιο πάνω. Σύμφωνα με το [12] μπορεί να έχουμε τρεις βασικές μορφές:

- **Node mobility:** σε αυτή την περίπτωση έχουμε τους κόμβους-αισθητήρα να είναι από μόνοι τους κινητοί. Εκτελούν αυτοί την κίνηση στο δίκτυο, αλλάζοντας την τοπολογία του πολύ συχνά. Ένα απλό παράδειγμα αυτής της μορφής κίνησης είναι όταν αντί να καλύπτουμε ένα χώρο με πάρα πολλούς κινητούς κόμβους, χρησιμοποιούμε αισθητά πιο λίγος κινητούς κόμβους, που με κάποιες προκαθορισμένες τροχιές, καλύπτουν πάλι ολόκληρο το χώρο.
- **Sink mobility:** αντί να έχουμε κίνηση από όλους τους κόμβους έχουμε κίνηση μόνο από τον sink, τον κόμβο δηλαδή που συλλέγει τα αποτελέσματα. Με αυτό τον τρόπο μπορούμε να εξοικονομήσουμε ενέργεια αφού δεν πρέπει έτσι να σπαταλούμε τόση ενέργεια στο να αποστέλλουμε σε μακρινές αποστάσεις τα μηνύματα απομακρυσμένων κόμβων, αλλά απλά να τα στέλνουμε μόνο όταν φτάσει το sink κοντά στον node. Αυτή η μορφή κινητικότητας μπορεί να έχει πολύ καλή εφαρμογή στην συλλογή πληροφοριών

από κόμβους που έχουν αναπτυχθεί για παράδειγμα σε ένα δάσος για την παρακολούθηση διαφόρων ζώων ή την συλλογή πληροφοριών για το δάσος. Ένα ρομπότ, που ήδη γνωρίζει τις θέσεις των αισθητήρων θα στέλνεται για να συλλέξει ότι πληροφορία έχουν οι κόμβοι και θα επιστρέφει μετά στη βάση.

- **Event mobility:** η τρίτη και τελευταία βασική μορφή κινητικότητας [2], σε ένα δίκτυο ασύρματων αισθητήρων είναι όταν μετακινείται το ίδιο το «πρόβλημα» που θέλουμε να παρακολουθήσουμε. Έχουμε για παράδειγμα ένα ζώο που θέλουμε να παρακολουθούμε συνέχεια σε μια μεγάλη περιοχή που καλύπτεται από πάρα πολλούς κόμβους. Το ζώο αυτό δεν είναι δυνατό να βρίσκεται σε ολόκληρη την περιοχή σε μια χρονική στιγμή. Ανά πάσα στιγμή μπορεί να κινηθεί και να βρεθεί σε άλλο σημείο. Έτσι ενεργεί είναι μόνο οι κόμβοι που βρίσκονται κοντά στο ζώο, αποστέλλοντας τις πληροφορίες που θέλουμε. Σε αντίθετη περίπτωση, όταν δηλαδή το ζώο αυτό φύγει από τον τομέα τους, μπαίνουν σε κατάσταση ύπνου, εξοικονομώντας έτσι πολύτιμη ενέργεια. Ακόμη εκτός από την παρακολούθηση ζώων μπορεί να βρούμε παραδείγματα για αυτό το είδος κίνησης σε στρατιωτικές εφαρμογές παρακολούθησης. Βασικά όπου έχουμε ένα γεγονός που κινείται μέσα στο δίκτυο και θέλουμε να καταγράφουμε τις κινήσεις του και την συμπεριφορά του, μπορούμε να βρούμε και μια εφαρμογή με event mobility.

Με το να έχουμε κινητικότητα στα ασύρματα δίκτυα αισθητήρων μπορούμε να πετύχουμε πολλά πράγματα. Πράγματα που μας βοηθούν να βελτιώσουμε τόσο την επίδοση όσο και την αξιοπιστία και μακροζωία των δικτύων μας. Μπορούμε εύκολα να μεγαλώσουμε το δίκτυο μας με την χρήση της κίνησης. Δεν είμαστε αναγκασμένοι να τοποθετούμε εμείς τους κόμβους αλλά από μόνοι τους πλέον μπορούν να μπουν στις θέσεις που θέλουμε αφού πλέον έχουν τη δυνατότητα να κινηθούν από μόνοι τους. Ακόμη μπορεί να κάνουν και την αρχική τοποθέτηση σε ένα χώρο από μόνοι τους, για παράδειγμα αν τους ρίξουμε με αεροπλάνο σε ένα δάσος που καίγεται για να παίρνουμε στοιχεία για την πυρκαγιά, και βάζοντας τους δεδομένο τα όρια το δάσους θα μπορούν οι κόμβοι να ακροβολιστούν στο δάσος από μόνοι τους δίνοντας μας τις μετρήσεις που χρειαζόμαστε. Σε αυτό τον τομέα μπορούμε να προσθέσουμε την ικανότητα ενός δικτύου να μπορεί να επιδιορθωθεί από μόνο του σε περιπτώσεις νεκρώσεις κάποιων κόμβων του. Με την προσθήκη κάποιον

επιπλέον κινητών κόμβων κατά τη δημιουργία του δικτύου, όταν εντοπιστεί μια τρύπα θα μπορούν με την κίνηση τους οι επιπλέον κόμβοι να την καλύπτουν για να μπορεί το δίκτυο να λειτουργεί χωρίς προβλήματα.

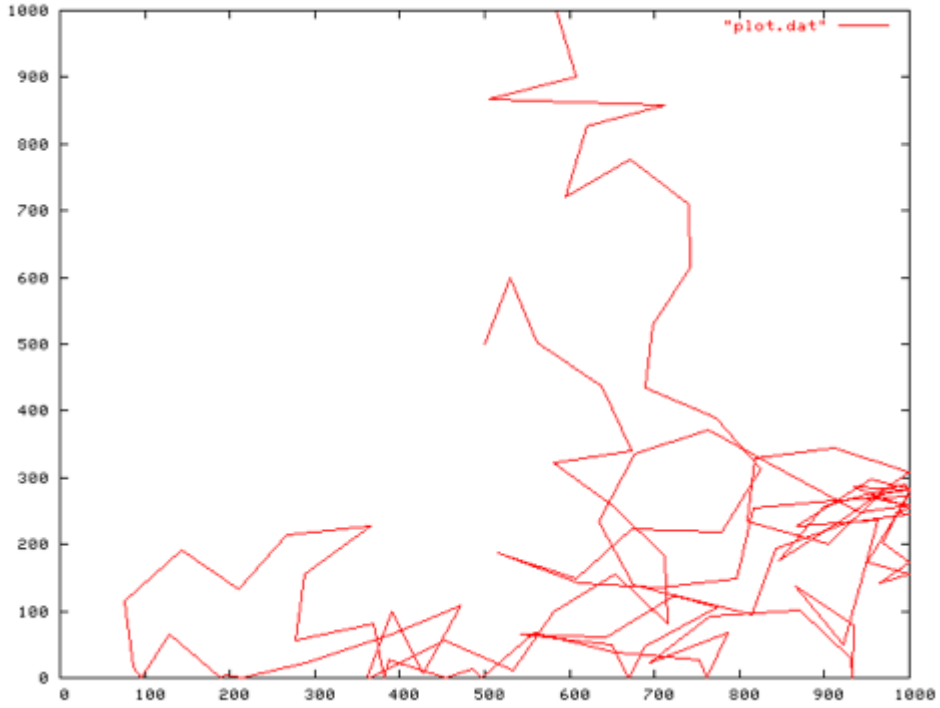
Ακόμη μπορούμε να καλύψουμε με λίγους κινητούς κόμβους μεγάλες περιοχές. Αν αυτές τις περιοχές τις καλύπταμε με τους συμβατικούς ακίνητους κόμβους τότε θα θέλαμε εξαιρετικά πιο μεγάλο αριθμό κόμβων από ότι με τους κινητούς. Καλύπτουμε βασικά το πλήθος αυτών των κόμβων με την κίνηση που θα εκτελούν στο χώρο οι κινητοί.

### 2.3 Μοντέλα Κινητικότητας

Τα μοντέλα κινητικότητας αντιπροσωπεύουν την κίνηση των κινητών κόμβων και το πώς η θέση τους, η ταχύτητα και η επιτάχυνση τους αλλάζουν με την πάροδο του χρόνου. Τα μοντέλα αυτά χρησιμοποιούνται συχνά για προσομοιώσεις, όταν νέες τεχνικές στον τομέα της επικοινωνίας και της πλοήγησης εξερευνώνται. Τα διαφορά μοντέλα διαχείρισης της κινητικότητας (mobility management schemes) για κινητές επικοινωνίες κάνουν χρήση τα διάφορα μοντέλα κινητικότητας για να προβλέψουν τις μελλοντικές θέσεις τους. Στο χώρο όπου εφαρμόζονται μοντέλα κινητικότητας, η συμπεριφορά της κίνησης κάποιου κόμβου μπορεί να περιγραφεί χρησιμοποιώντας είτε analytical είτε simulations μοντέλα. Η είσοδος για ένα analytical μοντέλο αφορά απλουστευμένες υποθέσεις που αφορούν την κίνηση των κόμβων. Τέτοια μοντέλα μπορούν να είναι αποδοτικά για απλές περιπτώσεις μέσα από μαθηματικούς υπολογισμούς. Σε αντίθεση με τα simulations μοντέλα, τα οποία θεωρούνται πιο λεπτομερείς και ρεαλιστικά. Τέτοια μοντέλα μπορούν να δώσουν λύσεις σε περίπλοκες περιπτώσεις. Τα πιο τυπικά μοντέλα κινητικότητας είναι τα εξής:

- **Random Walk Mobility Model**

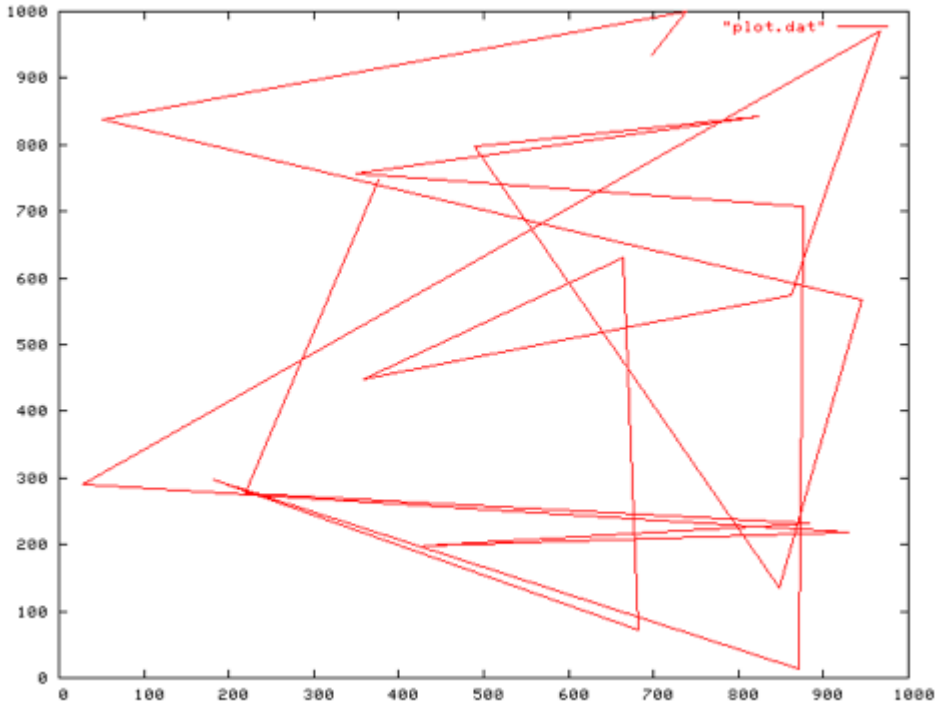
Σε αυτό το μοντέλο κινητικότητας, ένας κινητός κόμβος κινείται από την αρχική του θέση σε μια νέα θέση με το να διαλέξει τυχαία την κατεύθυνση και την ταχύτητα που θα κινείται. Η νέα ταχύτητα και κατεύθυνση καθορίζεται από καθορισμένα όρια, τα οποία κυμαίνονται από [min-speed, max-speed] και [0,2*pi], αντίστοιχα. Κάθε κίνηση σε αυτό το μοντέλο κινητικότητας, προκύπτει είτε από κάποιο σταθερό χρόνο είτε από κάποια σταθερή απόσταση που ταξιδεύει ο κόμβος.




Σχήμα 2.1 Η κίνηση του κόμβου στο Random Walk Mobility Model

- **Random Waypoint Mobility Model**

Το Random Waypoint Mobility Model συμπεριλαμβάνει παύσεις μεταξύ των αλλαγών στην κατεύθυνση και ταχύτητα. Ένας mobile node ξεκινά με το να μένει σε μια τοποθεσία για κάποια περίοδο χρόνου. Όταν αυτός ο χρόνος παρέρθει, ο κόμβος επιλέγει ένα τυχαίο προορισμό στο χώρο της προσομοίωσης και αυξάνει την ταχύτητα του μεταξύ ενός ορίου [min-speed, max-speed]. Ο mobile node ταξιδεύει προς το νέο προορισμό στην ταχύτητα που έχει επιλέξει. Με το που φτάνει στην συγκεκριμένη τοποθεσία, κάνει στάση για μια περίοδο χρόνου μέχρι να επιλέξει τον επόμενο προορισμό.




Σχήμα 2.2 Η κίνηση του κόμβου στο Random Waypoint Mobility Model

- **Boundless Simulation Area Mobility Model**

Στο Boundless Simulation Area Mobility Model υπάρχει μια σχέση μεταξύ της προηγούμενης κατεύθυνσης και ταχύτητας που κινείται ο κόμβος με τις αντίστοιχες τωρινές. Το διάνυσμα της ταχύτητας V=(v,O), χρησιμοποιείται για να περιγράψει την ταχύτητα το κόμβου v προς την κατεύθυνση O. Η θέση του κόμβου περιγράφεται από δυο σημεία (χ,y). Το διάνυσμα της ταχύτητας και η θέση του κόμβου αναβαθμίζονται σε κάθε delta χρόνο, σύμφωνα με τα πιο κάτω:

$$v(t+delta(t)) = min[max(v(t) + delta(v), 0), Vmax]$$

$$O(t+delta(t)) = O(t) + delta(O)$$

$$x(t+delta(t)) = x(t) + v(t) * cos(O(t))$$

$$y(t+delta(t)) = y(t) + v(t) * sin(O(t))$$

v = velocity

Vmax: είναι η μέγιστη ταχύτητα

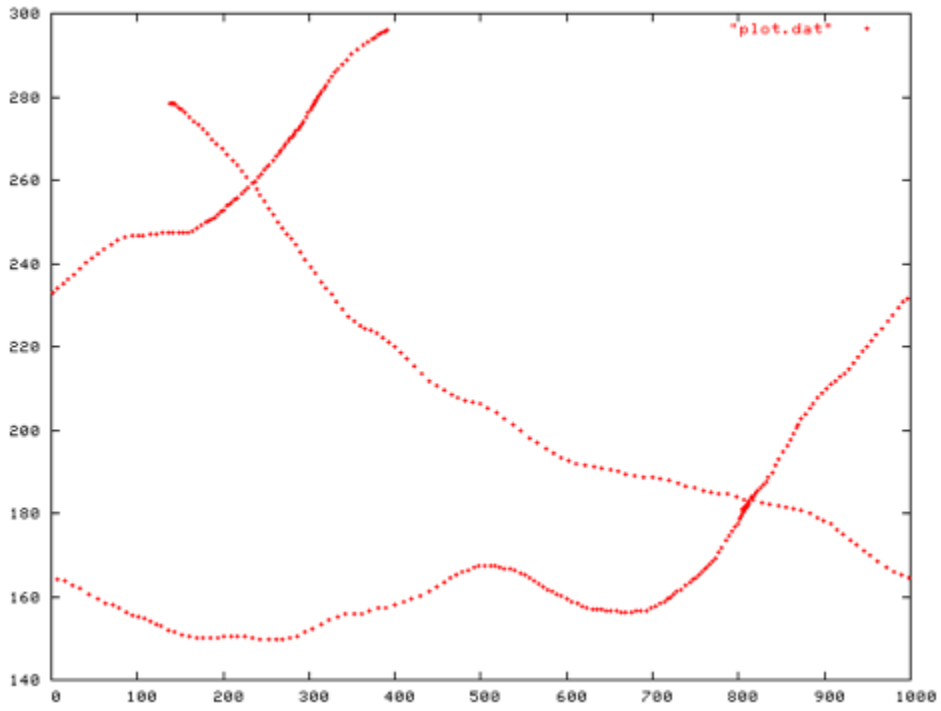
Delta (v): παίρνει τιμές μεταξύ [-Amax*delta (t), Amax*delta (t)]

Amax: είναι η μέγιστη επιτάχυνση

O: κατεύθυνση

Delta (O): αλλάζει κατεύθυνση μέσα στα όρια μεταξύ [-alpha*delta (t), alpha*delta (t)]

Alpha: είναι η μέγιστη γωνιακή μεταβολή (angular change)




Σχήμα 2.3 Η κίνηση του κόμβου στο Boundless Simulation Area Mobility Model

- **Gauss-Markov Mobility Model**

Το Gauss-Markov Mobility Model σχεδιάστηκε στο να προσαρμόζεται σε διαφορετικά επίπεδα τυχαιότητας με την αλλαγή κάποιον παραμέτρων. Αρχικά, σε κάθε κινητό κόμβο ανατίθεται μια αρχική ταχύτητα και μια κατεύθυνση. Σε κάθε σταθερή πάροδο του χρόνου, η κίνηση προέκυπτε ενημερώνοντας τις ταχύτητες και κατευθύνσεις κάθε κόμβου. Ειδικότερα, η τιμή της ταχύτητας και της κατεύθυνσης στο $n^{th}$ στιγμιότυπο, υπολογιζόταν στις τιμές της ταχύτητας και κατεύθυνσης στο στιγμιότυπο $(n-1)^{th}$ και σε μια τυχαία μεταβλητή χρησιμοποιώντας τις πιο κάτω εξισώσεις:

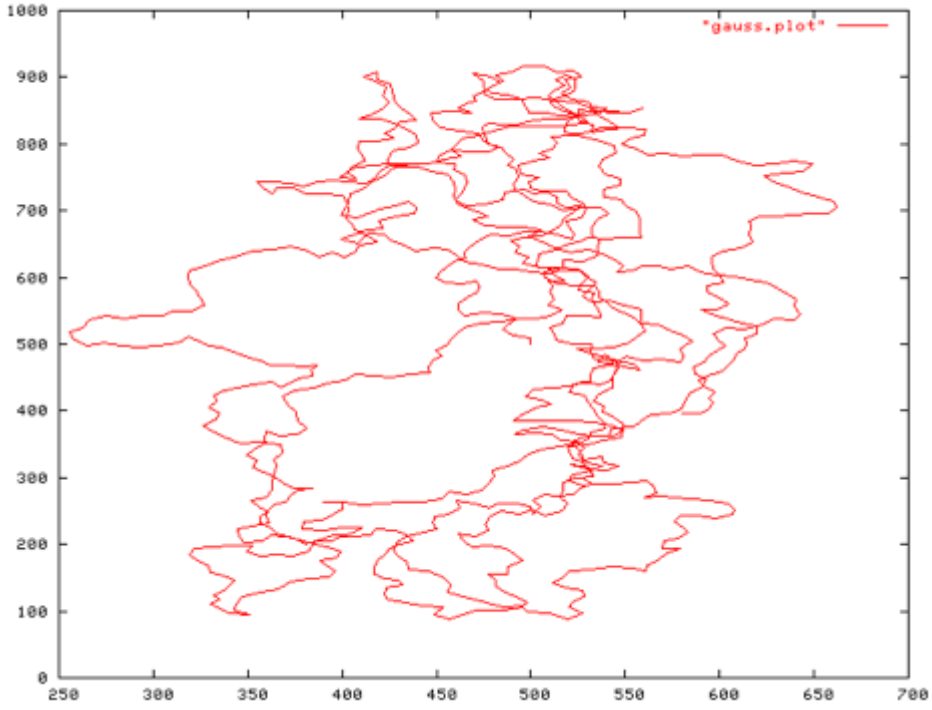
Sn = alpha * Sn-1 + ( 1 + alpha )*S + sqrt(1 -alpha 2 )*SXn - 1

Dn = alpha * Sn-1 + ( 1 + alpha )*D + sqrt(1 -alpha 2) *DXn – 1,

Όπου το Sn και Dn είναι οι νέες τιμές της ταχύτητας και της κατεύθυνσης σε χρόνο n, όπου 0<alpha<1, είναι η παράμετρος που ρυθμίζει την τυχαιότητα των S και D, που είναι σταθερές και αναπαριστούν την μέση τιμή της ταχύτητας και της κατεύθυνσης όταν n-> άπειρο. Οι μεταβλητές SXn-1 και το DXn-1, είναι τυχαίες μεταβλητές από κάποια Gaussian Distribution. Σε κάθε χρόνο η επόμενη θέση καθορίζεται από την τωρινή θέση, την ταχύτητα και την κατεύθυνση της κίνησης. Ειδικότερα σε χρόνο n, η θέση που θα έχει ο κινητός κόμβος είναι η εξής:

Xn = Xn-1 + Sn-1 Cos(Dn-1)

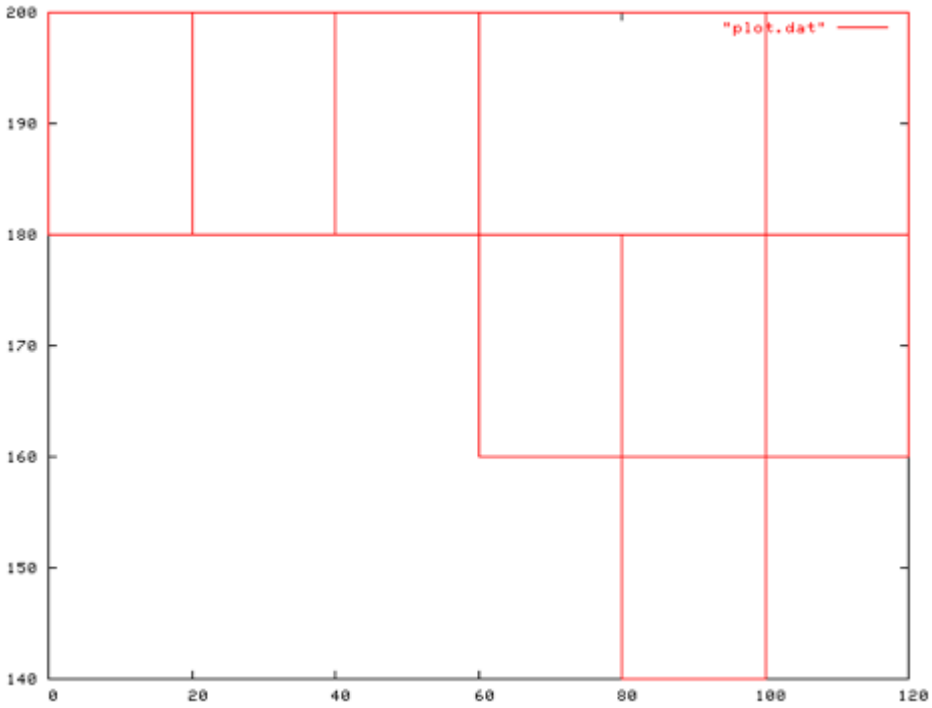
Yn = Yn-1 + Sn-1 Sin(Dn-1)




Σχήμα 2.4 Η κίνηση του κόμβου στο Gauss-Markov Mobility Model

- **City Section Mobility Model**

Στο City Section Mobility Model, η περιοχή που προσομοιώνουμε είναι ένα δίκτυο που αναπαριστά ένα section μιας πόλης. Κάθε mobile node αρχίζει την προσομοίωση της κίνησης του σε ένα προκαθορισμένο σημείο. Ο αλγόριθμος της κίνησης από κάποιο προορισμό σε κάποιο νέο προορισμό, δημιουργεί ένα shortest path μεταξύ των δύο σημείων. Επιπρόσθετα υπάρχουν χαρακτηριστικά οδήγησης όπως όριο ταχύτητας και η ελάχιστη απόσταση μεταξύ των κόμβων. Καθώς ο κόμβος φτάνει προς τον προορισμό του, κάνει στάση για μια περίοδο χρόνου μέχρι να επιλέξει τον επόμενο προορισμό.




Σχήμα 2.5 Η κίνηση του κόμβου στο City Section Mobility Model

## 2.3 Πρωτόκολλα Ομαδοποίησης

Το να ομαδοποιήσουμε ομάδες κόμβων είναι ένα από τα πιο σημαντικά προβλήματα που έχουμε. Ένας απλός ορισμός του clustering θα μπορούσε να είναι: η διαδικασία της οργάνωσης διάφορων αντικειμένων σε ομάδες, των οποίων τα μέλη είναι κοινά με κάποιο τρόπο. Στα WSNs, με αυτή την ομαδοποίηση στοχεύουμε στην μείωση της κατανάλωσης ενέργειας. Αυτό γιατί με την ομαδοποίηση των αισθητήρων, μια ομάδα θα στέλνει στον συντονιστή της και αυτό θα είναι υπεύθυνος για την παραπέρα διάδοση των δεδομένων τους. Έχουν προταθεί πολλών ειδών αλγόριθμοι, με τους περισσότερους να είναι Heuristics και να στοχεύουν στην δημιουργία του μικρότερου αριθμού από ομάδες έτσι ώστε κάθε κόμβος να είναι το πολύ d βήματα μακριά από τον επικεφαλής της ομάδας. Οι περισσότεροι από αυτούς έχουν πολυπλοκότητα O(n), με n τον αριθμό των κόμβων. Ακόμη αρκετοί από αυτούς απαιτούν συγχρονισμό μεταξύ των αισθητήρων, πράγμα που τους κάνει κατάλληλους μόνο για δίκτυα με μικρό αριθμό κόμβων.

Πιο κάτω θα μελετήσουμε μερικούς από αυτούς τους αλγορίθμους ομαδοποίησης (clustering algorithms).

- **Linked Cluster Algorithm(LCA & LCA2)**
  Σε αυτόν τον αλγόριθμο (LCA) [5] ένας κόμβος γίνεται ο επικεφαλής της ομάδας εάν έχει την μεγαλύτερη τιμή για ταυτότητα από τους γείτονες του που βρίσκονται ένα Hop μακριά του. Αυτός ο αλγόριθμος βελτιώνεται από τον LCA2 [2], όπου σε αυτόν διαλέγεται σαν επικεφαλής ο κόμβος με το μικρότερο Id από όλους τους κόμβους, που δεν είναι ούτε επικεφαλείς ούτε απέχουν ένα βήμα μακριά από άλλους επικεφαλείς.

- **Weighted Clustering Algorithm (WCA)**
  Ο WCA επιλέγει ένα κόμβο σαν επικεφαλής ομάδας βάση στον αριθμό των γειτόνων, την ενέργεια μετάδοσης, την μπαταρία και τον βαθμό κινητικότητας του

κόμβου. Επίσης περιορίζει τον αριθμό των κόμβων σε ένα cluster έτσι ώστε η απόδοση των πρωτοκόλλων MAC να μην υποβαθμίζεται [13].

- **Distributed Clustering Algorithm (DCA)**

  Αυτό ο αλγόριθμος χρησιμοποιεί βάρη για να διαλέξει τους επικεφαλείς κάθε ομάδας [17]. Αυτά τα βάρη είναι γενικά και μπορούν να προσδιοριστούν ανάλογα με την εφαρμογή. Διαλέγει τον κόμβο με το ψηλότερο βάρος μεταξύ των γειτόνων με απόσταση ένα βήμα σαν αρχηγό της ομάδας αυτής. Είναι κατάλληλος ο DCA για δίκτυα στα οποία οι κόμβοι είναι στατικοί ή κινούνται με μικρές ταχύτητες. Ο DMAC (Distributed and mobility-adaptive clustering) τροποποιεί τον DCA για να επιτρέπει την κίνηση των κόμβων, στη διάρκεια ή μετά την φάση της δημιουργίας των clusters.

- **Algorithm for Cluster Establishment (ACE)**

  Σε αυτό τον αλγόριθμο, που αναπτύσσετε στο [9], έχουμε ομαδοποίηση των κόμβων μέσα σε ένα σταθερό αριθμό επαναλήψεων, χρησιμοποιώντας τον βαθμό του κόμβου σαν κύρια παράμετρο. Μια από τις αδυναμίες του ACE είναι ότι διαλέγει τυχαία τον υποψήφιο κόμβο σε κάθε επανάληψη, γεγονός το οποίο δημιουργεί διαφορετικά αποτελέσματα κάθε φορά στο ίδιο δίκτυο. Επίσης χρησιμοποιώντας δύο παραμέτρους που προσαρμόζονται από τον χρήστη ελέγχει τον σχηματισμό καινούριων ομάδων, με αποτέλεσμα η απόδοσή του να βασίζεται σε αυτές, που καθορίζονται βάση του μεγέθους και του σχήματος του δικτύου αισθητήρων.

- **Low Energy Adaptive Clustering Hierarchy (LEACH)**

  Στόχος του είναι να παρέχει συνάθροιση πληροφοριών για δίκτυα αισθητήρων με το να παρέχει επικοινωνία που να είναι ενεργειακά επαρκής χωρίς να αγνοεί μερικούς κόμβους περισσότερο από άλλους. Σε κάθε κύκλο διαλέγονται διαφορετικοί επικεφαλείς για τις ομάδες που υπάρχουν. Ο κάθε κόμβος τρέχει ένα στοχαστικό αλγόριθμο για να αποφασίσει αν θα πρέπει να γίνει επικεφαλής σε κάθε συγκεκριμένο κύκλο. Οι κόμβοι που ήδη ήταν επικεφαλείς δεν μπορούν να

ξαναγίνουν για P χρόνο, όπου P το επιθυμητό ποσοστό αρχηγών στο δίκτυο. Στο τέλος κάθε γύρου όσοι δεν είναι επικεφαλείς, διαλέγουν τη πιο κοντινή ομάδα και μπαίνουν σε αυτή. Έπειτα οι επικεφαλείς των ομάδων διαμορφώνουν ένα πρόγραμμα για το πώς κάθε κόμβος στην ομάδα τους θα μεταδώσει τα δεδομένα του.

# Κεφάλαιο 3

## Προβλήματα προς επίλυση



### 3.1 Παρουσίαση προβλημάτων προς επίλυση

Σε αυτό το κεφάλαιο θα παρουσιάσω τα προβλήματα-εφαρμογές προς επίλυση κατά την διάρκεια της διπλωματικής μου εργασίας, όπου αφορούσαν την κινητικότητα των κινητών κόμβων αισθητήρων και την ομαδοποίηση τους με βάση συγκεκριμένα χαρακτηριστικά.

Η πρώτη εφαρμογή αφορά την εύρεση κοινών φίλων («Find your friends»). Αυτή η εφαρμογή αφορά την συμπεριφορά των κινητών κόμβων (mobile nodes) σε στρατιωτικό περιβάλλον όπου μη επανδρωμένα οχήματα προσπαθούν να συγκλίνουν σε μια περιοχή έτσι ώστε να ομαδοποιηθούν, καθώς την ίδια στιγμή προσπαθούν να αποφύγουν τον εχθρό. Προσομοιώνοντας αυτή την εφαρμογή θα μας αποδείξει την αυτονομία που έχουν οι κόμβοι αισθητήρων σε ένα Ασύρματο Δίκτυο Αισθητήρων, με το να κινούνται προς μια κατεύθυνση, δηλαδή αλλάζοντας την θέση τους, με σκοπό να συγκλίνουν όλοι οι αισθητήρες σε ένα τυχαίο σημείο.

Η δεύτερη εφαρμογή αφόρα την κίνηση του αρχηγού (leader sensor) και την ακολουθία όλων των υπόλοιπων αισθητήρων στην κίνηση που έκανε ο αρχηγός («Follow the leader»). Αυτή η εφαρμογή αφόρα στην προσομοίωση μιας πιθανής χρησιμοποίησης των Ασύρματων Δικτύων Αισθητήρων σε ένα Ευφυή Σύστημα Μεταφοράς (Intelligent Transportation System) όπου τα οχήματα σε ένα αυτοκινητόδρομο (highway) ακολουθούν το ένα το άλλο αυτόματα και δεν χάνουν τα οχήματα που βρίσκονται μπροστά τους. Αυτό, επίσης, μπορεί να χρησιμοποιηθεί στο να δείξουμε την σημαντικότητα του να διατηρούμε την σωστή απόσταση από το όχημα μπροστά μας ούτος ώστε να αποφεύγουμε τα τροχαία δυστυχήματα σε μεγάλο βαθμό. Προσομοιώνοντας αυτή την εφαρμογή θα μας δείξει την αυτονομία και την συνεργασία που έχουν οι κόμβοι αισθητήρων σε ένα Ασύρματο Δίκτυο Αισθητήρων, όπου με την χρήση των κατάλληλων αισθητήρων ακλουθούν το ένα το άλλο με βάση την κίνηση του αρχηγού τους.

### 3.2 Προηγούμενη Εργασία

Από την μελέτη που έκανα με βάση προηγούμενες διπλωματικές και επιστημονικές έρευνες, η αρχική εντύπωση που μου δημιουργήθηκε είναι ότι τα ασύρματα δίκτυα αισθητήρων είναι μια ενδιαφέρουσα και πολλά υποσχόμενη τεχνολογία. Οι εφαρμογές που έχουν αναπτυχτεί κατά καιρούς είναι πάρα πολλές τόσο σε επίπεδο προσομοίωσης χρησιμοποιώντας γραφικές απεικονίσεις των εφαρμογών, όσο και σε επίπεδο ρομπότ, προσομοιώνοντας μια πιο ρεαλιστική εφαρμογή των διάφορων πρωτοκόλλων.

#### 3.2.1 Προηγούμενες Διπλωματικές Εργασίες

Σε επίπεδο ρομπότ έχει γίνει έρευνα γύρω από την σπουδαιότητα της συνεργασίας ρομπότ με κάποιο ασύρματο δίκτυο (sensor network) στο θέμα του καθορισμού της θέσης κάποιου στόχου. Αυτό έχει μεγάλη σημασία διότι κάποιο ρομπότ πολύ εύκολα μπορεί να χάσει τον προσανατολισμό του και αυτό συμβαίνει διότι το ρομπότ κινείται συνεχώς και

αλλάζει πορεία και προορισμούς, οπόταν θα πρέπει το δίκτυο να του παρέχει πληροφορίες ώστε να γνωρίζει ανά πάσα στιγμή την θέση του. Επιπλέον μια άλλη εφαρμογή στο επίπεδο ρομπότ, είναι και η εύρεση του ελάχιστου μονοπατιού, η οποία με την βοήθεια κάποιον heuristics είχε πολύ μεγάλες πιθανότητες το μονοπάτι που ανακάλυπτε να ήταν το ελάχιστο. Επιπρόσθετα, ερευνήθηκαν και θέματα κάλυψης περιοχής, βρίσκοντας αλγόριθμο όπου μειωνόταν κατά πολύ το φαινόμενο της «μαύρης τρύπας» και αλγόριθμοί ακολούθησης στόχου.

Σε επίπεδο προσομοίωσης χρησιμοποιώντας γραφικές απεικόνισης των εφαρμογών, αναπτύχθηκαν αρκετές εφαρμογές. Μια από τις εφαρμογές, είναι αύτη της μελέτης του περιμετρικού ελέγχου μιας περιοχής όπου εξάχθηκε το συμπέρασμα ότι με λιγότερους κινητούς κόμβους μπορεί να καλυφθεί πιο εύκολα μια περιοχή σε σχέση με τους συμβατικούς στατικούς κόμβους, όπου θα χρειαζόταν τεράστιος αριθμός κόμβων αισθητήρων. Επιπλέον μια άλλη κατηγορία εφαρμογών αφορά θέματα ομαδοποίησης των κινητών κόμβων μεταξύ τους. Η ομαδοποίηση είναι παρά πολύ σημαντική διότι τις περισσότερες φορές πρέπει να εγκαταστήσουμε ένα ασύρματο δίκτυο αισθητήρων σε χώρους όπου δεν έχει πρόσβαση άνθρωπος. Έτσι χρειάζεται για παράδειγμα να τους ρίξουμε από αεροπλάνο. Με ένα απλό αλγόριθμο ομαδοποίησης θα μπορούμε να ομαδοποιήσουμε τους κόμβους, και μετά να τους αφήσουμε να συντονιστούν για να πραγματοποιήσουν τις μετρήσεις που θέλουμε.

Μελετώντας τις προηγούμενες εργασίες που έχουν γίνει με βοήθησε να πάρω μια ιδέα για την φύση των προβλημάτων που σχετίζονται με δίκτυα ασύρματων αισθητήρων αλλά και να πάρω κάποιες από τις υπάρχουσες τεχνικές και να τις εφαρμόσω στην επίλυση των προβλημάτων που μελετώ. Στο επόμενο κεφάλαιο θα επεξηγήσω με ακρίβεια ποιες από τις υπάρχουσες ιδέες χρησιμοποίησα στους δικούς μου αλγορίθμους άλλα το πιο σημαντικό θα καταγράψω τις βελτιώσεις που θα εισάγω στους δικούς μου αλγορίθμους κάνοντας τους πιο αποτελεσματικούς και εύχρηστους σε θέματα κινητικότητας και ομαδοποίησης.

Η προηγούμενη εργασία σε θέματα κινητικότητας και ομαδοποίησης, υποστήριζε την δημιουργία ενός τετραγωνισμένου πλέγματος το οποίο αναπαριστούσε την περιοχή που θα τοποθετούνταν οι κόμβοι αισθητήρων. Στο πλέγμα αυτό βρίσκονταν μόνο δυο ομάδες κόμβων και οι κινήσεις τους παραγόνταν από το μοντέλο κινητικότητας Manhattan. Ανατίθεται στον κάθε κόμβο μια τυχαία αρχική διεύθυνση για κίνηση. Μόλις ένας κόμβος ανακαλύψει ότι μέσα στην εμβέλεια του έχει εισέλθει κάποιος κόμβος, ελέγχει πρώτα από όλα αν αυτός ο κόμβος ανήκει στην ίδια ομάδα με αυτόν. Αν όχι αλλάζουν και οι δύο την πορεία τους. Στην περίπτωση που είναι της ίδιας ομάδας με αυτόν, τότε πραγματοποιεί ακόμη ένα επιπλέον έλεγχο. Βλέπει αν ο κόμβος που έχει συναντήσει είναι ο αρχηγός του ή κόμβος που έχει ήδη ομαδοποιηθεί μαζί με τον αρχηγό τους. Αν όντως ισχύει αυτό το σενάριο τότε μπαίνει και αυτός στην ομάδα, αλλάζοντας την φορά της κίνησής του, κάνοντας την ίδια με την κίνηση της ομάδας. Αν πάλι βρεθεί με κάποιο κόμβο της ομάδας του, που όμως δεν ομαδοποιήθηκε με τον αρχηγό τους, τότε αλλάζουν και οι δύο κατευθύνσεις και συνεχίζουν το ψάξιμο για εύρεση της ομάδας.

Από αυτά θεώρησα σαν στοιχεία που έπρεπε να υπήρχαν στους αλγορίθμους μου τα εξής:

- κάθε κόμβος θα έπρεπε να είχε προορισμό σε τυχαίο σημείο κάθε φορά.
- κάθε κόμβος είχε την εμβέλεια που κάλυπτε (Cover Range).
- ύπαρξη αρχηγών σε κάθε ομάδα.
- αλλαγή κατεύθυνσης σε περίπτωση που δυο κόμβοι δεν ανήκαν στη ίδια ομάδα.

Επιπλέον από τις διάφορες επεκτάσεις και εισηγήσεις που αφορούσαν αλγορίθμους κινητικότητας και ομαδοποίησης θεώρησα σωστότερο να προσθέσω τα εξής στοιχεία στους αλγορίθμους μου:

- η κίνηση κάθε κόμβου να στηριζόταν στο μοντέλο κινητικότητας Random Waypoint έτσι ώστε να προσομοιώνονταν όλες οι πιθανές κινήσεις των κόμβων μέσα στην προκαθορισμένη περιοχή (area).

- κάθε κόμβος θα έχει και εμβέλεια αίσθησης (Sense Range), ούτως ώστε να ανιχνεύει σε μεγαλύτερες αποστάσεις κινητούς κόμβους.
- Δημιουργία υπο-ομάδων και συνενωσή τους σε κάποια φάση της κίνησης που κάνουν.
- Η προσομοίωση στους αλγόριθμους θα γίνεται με περισσότερες από δυο ομάδες με ισάριθμο αριθμό αρχηγών.

### 3.2.1 Επιστημονικές Έρευνες

Η επιστημονική κοινότητα έχει ασχοληθεί το τελευταίο καιρό πάρα πολύ με θέματα κινητικότητας σε Ασύρματα Δίκτυα Αισθητήρων βασιζόμενη κυρίως στην συμπεριφορά της φύσης και των ζώων. Πιο κάτω θα αναφερθούν αλγόριθμοι που προέκυψαν από την μελέτη της φύσης γενικότερα. Οι αλγόριθμοί είναι η εξής:

- **Hybrid- AC&A[26]:** Είναι ένας υβριδικός αλγόριθμος ο οποίος βασίζεται στο τρόπο όπου τα μυρμηγκιά δημιουργούν αποικίες και σε agglomerate αλγόριθμους ομαδοποίησης. Ο αλγόριθμός αυτός προέκυψε μετά από την μελέτη του τρόπου όπου τα μυρμήγκια ρίχνουν το φορτίο που κουβαλούν και τον τρόπο που αποκτούν το φορτίο τους. Οι επιδόσεις του Hybrid- AC&A σε σχέση με άλλους αλγορίθμους, υποδηλώνουν ότι ο αλγόριθμος βελτιώνει την πτυχή της αποδοτικότητας του χρόνου για να γίνει η ομαδοποίηση.

- **Flocking clustering algorithm[25]:** Αυτός ο αλγόριθμος χρησιμοποιεί stochastic και heuristic αρχές οι οποίες έχουν ανακαλυφθεί από την παρατήρηση πουλιών (bird flocks) και από τα ψάρια. Ό αλγόριθμος δημιουργεί μια συσπείρωση δεδομένων σε ένα πλέγμα δυο διαστάσεων για εύκολη ανάκτηση των αποτελεσμάτων της ομαδοποίησης. Ό αλγόριθμός είναι εμπνευσμένος από την συμπεριφορά στον τρόπο αυτό-οργάνωσης των πουλιών.

- **A Hybrid Clustering Algorithm based on Honey Bees Mating Optimization and Greedy Randomized Adaptive Search Procedure [24]:** Είναι ένας υβριδικός αλγόριθμος ο οποίος βασίζεται Εμπνευσμένος από την φύση ο οποίος βασίζεται στην βελτιστοποίηση ζευγαρώματος των μελισσών και σε ένα άπληστο αλγόριθμο για την βελτιστοποίηση στην ομαδοποίηση N αντικειμένων σε K ομάδες. Η απόδοση του συγκρίνεται με άλλα δημοφιλή μοντέλα όπως swarm optimization, ant colony optimization και genetic algorithms.

Οι πιο πάνω αλγόριθμοι αποτέλουν μερικά παραδειγματα το πόσο μέγάλη επιδραση έχει η φύση πάνω στον καθορισμο της κίνησης σε Ασύρματα δίκτυα Αισθητηρων. Οι αλγοριθμοί που παραθέτω πιο κάτω δεν έχουν κάποια χαρακτηριστικα από τέτοιους είδους αλγορίθμους.

# Κεφάλαιο 4

## Σχεδιασμός και Υλοποίηση Αλγορίθμων



### 4.1 Γενικά- Παραδοχές- Υποθέσεις

Σε αυτό το κεφάλαιο θα αναπτυχθούν τα δύο προβλήματα με τα οποία ασχολήθηκα στην διπλωματική εργασία. Με λίγα λόγια είχα να υλοποιήσω ένα αλγόριθμο για κάθε μια περιοχή που μελέτησα καθ' όλη την διάρκεια της εργασίας μου. Πιο συγκεκριμένα έπρεπε να αναπτύξω μια εφαρμογή η οποία ομαδοποιούσε τους κόμβους κατά ομάδες, τοποθετώντας τους αρχικά σε τυχαίες θέσεις στο χώρο και μια εφαρμογή κατά την οποία όλοι οι κόμβοι ακολουθούσαν τον κόμβο-αρχηγό σε κάθε του κατεύθυνση αφότου πρώτα ομαδοποιούνταν σε προκαθορισμένες θέσεις στο χώρο.

Και οι δύο εφαρμογές έχουν αναπτυχθεί σε γραφικό περιβάλλον, σε γλώσσα προγραμματισμού JAVA, χρησιμοποιώντας την κλάση Frame, η οποία μπορεί να δημιουργεί γραφικές αναπαραστάσεις αντικείμενων. Περισσότερες πληροφορίες για την χρήση της συγκεκριμένης κλάσης υπάρχει στην ιστοσελίδα της JAVA (http://docs.oracle.com/javase/1.3/docs/api/java/awt/Frame.html). Ο γραφικός χώρος ο

οποίος χρησιμοποιείται στις εφαρμογές είναι ένα Frame συγκεκριμένων διαστάσεων (παράδειγμα 1024X860), για πιο εύκολη κατανόηση των αλγορίθμων και για εύκολη αναγνώριση των γνωρισμάτων του προβλήματος κάθε φορά. Οι εφαρμογές όλες είναι σχεδιασμένες και δοκιμασμένες στην πλατφόρμα Eclipse, η οποία υποστηρίζει την γλώσσα προγραμματισμού JAVA.

Κατά το σχεδιασμό και υλοποίηση των αλγορίθμων για τις εφαρμογές έπρεπε πρώτιστα να καθοριστεί το όλο περιβάλλον στο οποίο θα στηρίζονταν οι προσομοιώσεις των προβλημάτων. Σκοπός ήταν να γίνει μια γραφική αναπαράσταση η οποία θα ήταν όσο το δυνατό πιο ακριβής στο να παρουσιάσει την επίλυση του προβλήματος και όχι να λάβει υπόψη πραγματικές συνθήκες οι οποίες στη πραγματικότητα θα αντιμετώπιζε ένα κόμβος π.χ. ανωμαλίες στο έδαφος, παρεμβολές σημάτων.

Λαμβάνοντας υπόψη ότι σκοπός ήταν να σχεδιαστούν και να υλοποιηθούν εφαρμογές οι οποίες θα επίλυαν τα προβλήματα που παρουσιάστηκαν στο κεφάλαιο 3.1, δημιούργησα ένα γραφικό περιβάλλον το οποίο θα ήταν πολύ απλό ώστε να είναι και πιο κατανοητό. Όπως ανάφερα και πιο πάνω στην αρχή δημιούργησα ένα Frame το οποίο αναπαριστούσε την περιοχή στην οποία υπήρχαν οι κόμβοι αισθητήρων. Στην συνέχεια με μικρά κόκκινα τετράγωνα αναπαριστούσα τους κόμβους. Κάθε κόμβος περιτριγυριζόταν από ένα κύκλο όπου αναπαριστούσε την εμβέλεια του. Ανάλογα της κίνησης κάθε κόμβου μετακινούνταν και τα σχήματα στις κατάλληλες θέσεις. Θεώρησα σαν παραδοχές το γεγονός ότι οι κόμβοι θα κινούνταν σε μια επιφάνεια χωρίς ανωμαλίες στο έδαφος και χωρίς εμπόδια. Τα μοναδικά εμπόδια που βρίσκονταν στο χώρο ήταν οι άλλοι κόμβοι. Επιπλέον θέματα κατανάλωσης ενέργειας δεν μελέτησα σε αυτές τις προσομοιώσεις και θεωρούσα ότι υπάρχει πάντα ικανοποιητικό ποσοστό ενέργειας ώστε να μπορούν οι αισθητήρες να στέλλουν τα διαφορά σήματα τους. Επιπρόσθετα υπόθεσα ότι ο Sink κόμβος γνωρίζει τις διαστάσεις της περιοχής του δικτύου, έχει μεγάλη υπολογιστική ισχύ, μνήμη και μεγάλα αποθέματα ενέργειας. Τέλος για τους κόμβους αισθητήρων υπόθεσα ότι κάθε κόμβος γνωρίζει τη θέση του σε σχέση με την περιοχή του δικτύου και μπορεί να βρίσκεται σε μια από τις τέσσερις καταστάσεις (κατάσταση

μετάδοσης μηνύματος, κατάσταση λήψης μηνύματος, κατάσταση αίσθησης γεγονότος, κατάσταση μετακίνησης ).

Η κατανάλωση της ενεργείας επικεντρώνεται σε θέματα που σχετίζονται με την κίνηση των κόμβων και της ανταλλαγής μηνυμάτων μεταξύ των κόμβων στην υπό- ομάδα που δημιουργείτε. Θέματα κατανάλωσης ενέργειας που σχετίζονται με άλλες λειτουργίες του αισθητήρα, όπως επεξεργασία δεδομένων, ενέργεια που σχετίζεται με το εύρος της εμβέλειας των κόμβων κ.α., δεν λαμβάνονται υπόψη στα πλαίσια αυτής της διπλωματικής.

### 4.2 Αλγοριθμός για την εφαρμογή «Find Your Friends»

#### 4.2.1 Γενικά για το πρόβλημα

Σε αυτήν την εφαρμογή το γενικό σενάριο είναι ότι, υπάρχει μια περιοχή η οποία καλύπτεται από κάποιο συγκεκριμένο αριθμό αισθητήρων. Οι θέσεις των αισθητήρων καθορίζονται τυχαία σε όλη την έκταση της περιοχής, δίνοντας τους μια τυχαία αρχική κατεύθυνση και ταχύτητα. Για λογούς πιο εύκολης κατανόησης και ομοιομορφίας στην ομαδοποίηση των αισθητήρων υπέθεσα ότι κάθε ομάδα θα είχε τον ίδιο αριθμό αισθητήρων. Αφού πλέον οι αισθητήρες καθορίζονταν σε μια τυχαία θέση, ο αλγόριθμος προσπαθούσε ανάλογα με την κίνηση τους να καταφέρει να ομαδοποίηση τους κόμβους που ήταν στην ίδια ομάδα σε κάποια υπό-περιοχή της αρχικής περιοχής. Αφού έκανε κάποιους ελέγχους (θα επεξηγήσω με λεπτομέρεια στο 4.2.3) για να καταφέρει να δημιουργήσει τις ομάδες, τερμάτιζε όταν πλέον οι ομάδες είχαν συγκεντρωθεί σε ξεχωριστές υπό-περιοχές.

### 4.2.2 Αλγόριθμος

1. Καθορισμός αριθμών κόμβων ανά ομάδα, αριθμό ομάδων, Cover Range/Sense Range, speed και όρια περιοχής (area).

2. Με βάση τις πιο πάνω τιμές γίνεται ο καθορισμός θέσεων κόμβων στην περιοχή με τυχαίο τρόπο με συγκεκριμένες συντεταγμένες.

3. While (not in teams) {

3.1 Ξεκίνησε την κίνηση κάθε κόμβου χρησιμοποιώντας το mobility model Random Waypoint (Set a random destination).

3.2 Έλεγχος αν κάποιος κόμβος κατά την κίνηση του βρίσκεται στο Sense Range κάποιου άλλου μέσω της Ευκλείδειας Απόστασης.

3.2.1 if (Ευκλείδεια Απόστασης< Sense Range && ίδιας ομάδας){

Οι κόμβοι θα μετακινηθούν ο ένας προς τον άλλον

If (Ευκλείδειας Απόστασης< Cover Range)

Οι κόμβοι θα ενωθούν μαζί θα δημιουργήσουν υπό- ομάδα και θα κινούνται ενιαία.

} else if (Ευκλείδεια Απόστασης< Sense Range && όχι ίδιας ομάδας){

Οι κόμβοι θα μετακινηθούν σε διαφορετικές κατεύθυνσης

} else

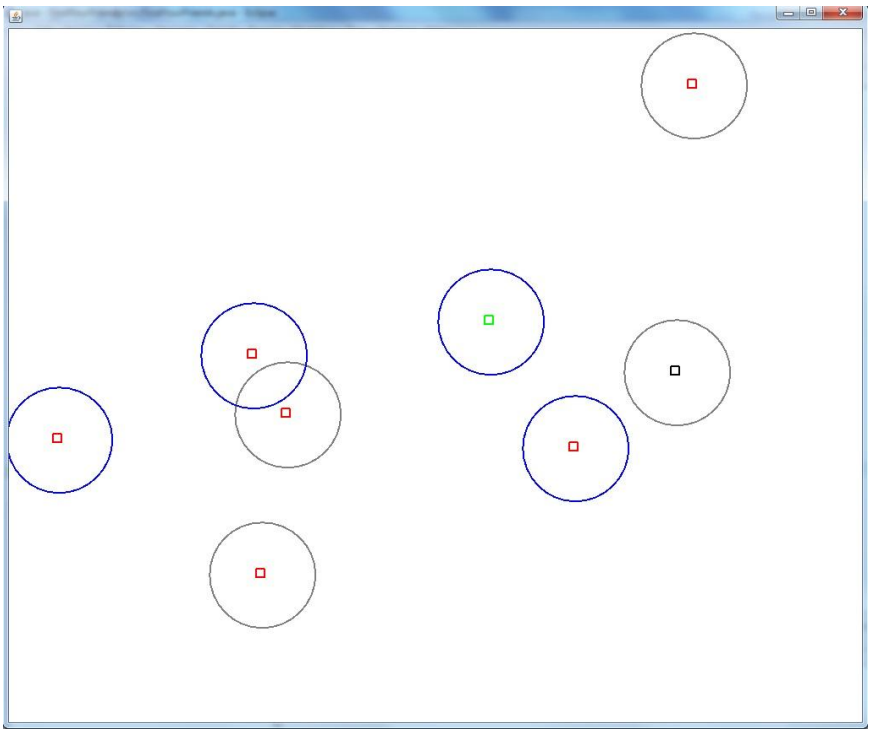
Οι κόμβοι συνεχίζουν προς το random destination

3.3 Σε κάθε 50 κινήσεις των κόμβων αύξησε τους την ταχύτητα που κινούνται ώστε να επιταχυνθεί η ομαδοποίηση τους.

}

### 4.2.3 Επεξήγηση αλγορίθμου

Όπως είδαμε από το 4.2.2 η εφαρμογή μας ξεκίνα με το να καθορίσουμε κάποιες παραμέτρους που διαδραματίζουν σημαντικό ρόλο στην όλη φιλοσοφία της κίνησης και ομαδοποίησης των κόμβων. Στην αρχή καθορίσαμε τα όρια της περιοχής που στα οποία θα προσομοιωνόταν ο αλγόριθμος. Στην συνέχεια θα έπρεπε να καθοριστούν τα κυριότερα χαρακτηριστικά των κόμβων, τα οποία είναι το Cover Range/Sense Range και το μέγεθος του αισθητήρα. Με βάση αυτά, οι κόμβοι μας θα διασκορπίζονταν μέσα στην περιοχή με ένα πιο ομοιόμορφο τρόπο ούτος ώστε να δείχνει μια πιο ρεαλιστική κάλυψη της περιοχής.



Σχήμα 4.1 Αρχικές θέσεις των κόμβων για δύο ομάδες

Στην συνέχεια καθορίσαμε των αριθμών των ομάδων που θα θέλαμε να δημιουργήσουμε και τους αναθέσαμε μια αρχική ταχύτητα. Με βάση αυτά τα στοιχειά δημιουργήσαμε μια συνάρτηση/μέθοδο η οποία λάμβανε υπόψη τα πιο πάνω και μας έκανε μια γραφική απεικόνιση (βλέπε Σχήμα 4.1) των αρχικών θέσεων των κόμβων στην περιοχή.

Στην συνέχεια ξεκινούσε η κίνηση των κόμβων χρησιμοποιώντας το μοντέλο κινητικότητας Random Waypoint. Με βάση αυτό το μοντέλο κινητικότητας σε κάθε κόμβο ανατίθεται ένας τυχαίος προορισμός στη περιοχή. Η κίνηση που παρήγαγε ο κόμβος βασιζόταν στις τριγωνομετρικές συναρτήσεις sin (θ) και cos (θ), αφού πρώτα υπολογιζόταν η γωνία θ. Αυτή η γωνιά θ υπολογιζόταν ως εξής:

$\lambda_{\varepsilon\varphi}=(Y_{destination\_position}-Y_{current\_position})/\ (X_{destination\_position}-X_{current\_position})$

Ξερώντας από την αναλυτική γεωμετρία ότι η λεφ= εφ (θ)

$\theta^{o}=$ τοξεφ($\lambda_{\varepsilon\varphi}$),

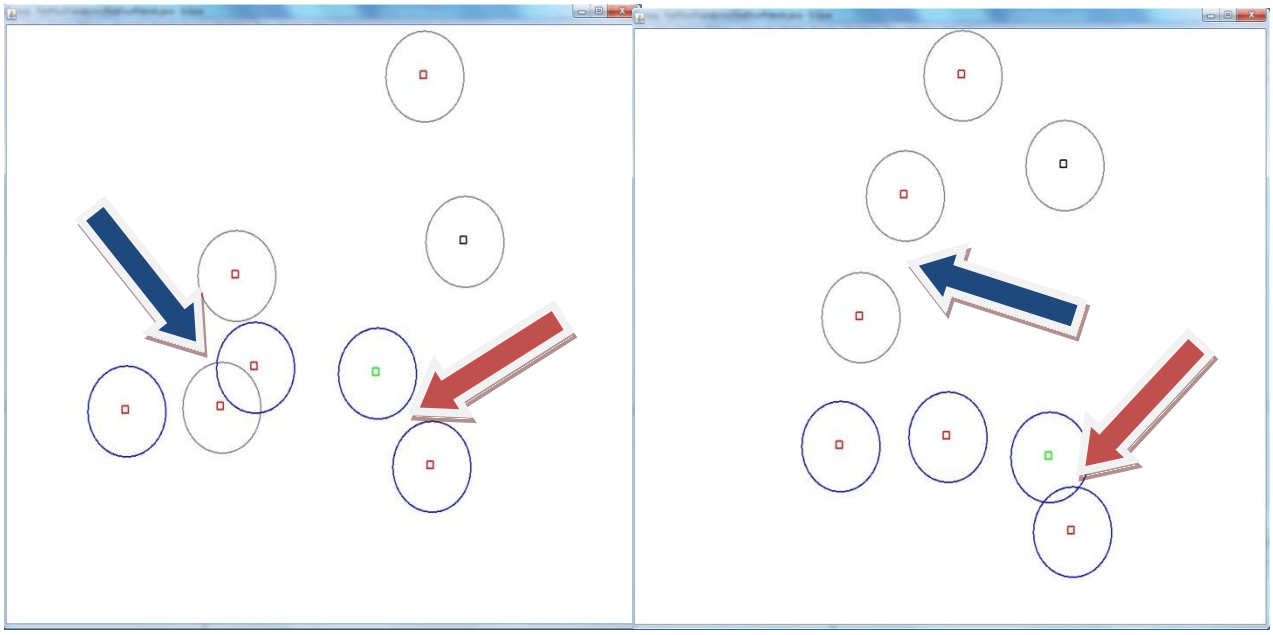
Οπόταν υπολογίζοντας την γωνιά θ, εύκολα υπολογιζόταν το sin (θ)*speed για να καθορίσω την κίνηση του X και το cos (θ) *speed για να καθορίσω την κίνηση του Y.

Κατά την διάρκεια της κίνησης των κόμβων γίνονταν διάφοροι έλεγχοι ώστε να επιτευχθεί ο στόχο. Στην αρχή έλεγχα την παράμετρο του Sense Range. Αν δυο κόμβοι είχαν απόσταση μικρότερη του Sense Range και βρίσκονταν σε διαφορετικές ομάδες τότε άλλαζα την κατεύθυνση της κίνησης τους προς άλλους προορισμούς, αλλιώς συνέχιζαν την αρχική τους πορεία. Στην περίπτωση που οι αποστάσεις τους ήταν μικρότερη του Cover Range και αφού ήταν της ίδιας ομάδας τότε συνενωνόταν μαζί και κινούνταν αυτόνομα μέσα στην περιοχή. Ο καθορισμός του κόμβου που θα αναλάμβανε την κίνηση γινόταν με βάση το μικρότερο node id, δηλαδή αν δυο κόμβοι, οι 2 και 4 δημιουργούσαν υπό-ομάδα τότε ο 2 θα κινούταν προς κάποιο τυχαίο προορισμό και απλά ο 4 θα τον ακολουθούσε. Γενικά κάναμε τους εξής ελέγχους:

- Αν δυο κόμβοι έπρεπε να συνενωθούν, με βάση τα πιο πάνω, τότε ο «οδηγός» θα ήταν αυτός με το μικρότερο id.
- Αν μια υπό- ομάδα Α και πιο συγκεκριμένα ο «οδηγός» αυτής συνενωνόταν με κάποιο άλλο κόμβο Β, τότε υπήρχαν δυο περιπτώσεις:
    i. Αν ο «οδηγός» είχε μικρότερο id από τον κόμβο Β τότε ο κόμβος Β ακολουθούσε την υπό- ομάδα Α
    ii. Αν ο κόμβος Β μικρότερο id από τον «οδηγός», τότε ολόκληρη η υπό- ομάδα Α ακολουθούσε κόμβο Β.

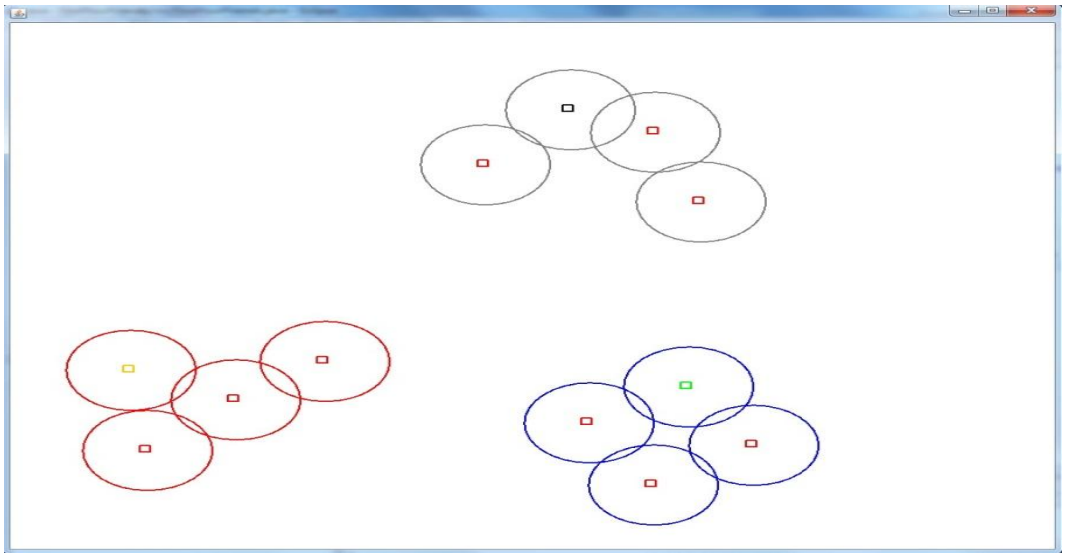
- Αν μια υπό- ομάδα Α και πιο συγκεκριμένα κάποιος κόμβος που δεν ήταν ο «οδηγός» αυτής συνενωνόταν με κάποιο άλλο κόμβο Β, τότε υπήρχαν δυο περιπτώσεις:
    i. Αν ο «οδηγός» είχε μικρότερο id από τον ο κόμβος Β τότε ο κόμβος Β ακολουθούσε την υπό- ομάδα Α.
    ii. Αν ο κόμβος Β μικρότερο id από τον «οδηγός», τότε ολόκληρη η υπό- ομάδα Α συν του οδηγού της ακολουθούσε κόμβο Β.
- Αν μια υπό- ομάδα Α και πιο συγκεκριμένα ο «οδηγός» αυτής συνενωνόταν με κάποια άλλη υπό- ομάδα Β, τότε υπήρχαν δυο περιπτώσεις:
    i. Αν ο «οδηγός» της υπό- ομάδα Α είχε μικρότερο id από τον «οδηγό» της υπό- ομάδα Β τότε ολόκληρη η υπό- ομάδα Β συν του οδηγού της ακολουθούσε την υπό- ομάδα Α.
    ii. Αν ο «οδηγός» της υπό- ομάδα Β είχε μικρότερο id από τον «οδηγό» της υπό- ομάδα Α τότε ολόκληρη η υπό- ομάδα Α συν του οδηγού της ακολουθούσε την υπό- ομάδα Β.

- Αν μια υπό- ομάδα Α και πιο συγκεκριμένα κάποιος κόμβος που δεν ήταν ο «οδηγός» αυτής συνενωνόταν με κάποια άλλη μια υπό- ομάδα Β και πιο συγκεκριμένα κάποιος κόμβος που δεν ήταν ο «οδηγός» αυτής, τότε ο έλεγχος γινόταν μεταξύ των «οδηγων».



Σχήμα 4.2 Δημιουργία υπό-ομάδων(κόκκινα βέλη) και αλλαγής προορισμού (μπλε βέλη)

Επιπρόσθετα, όταν οι κόμβοι κάνουν αριθμό κινήσεων πάνω από 50 και δεν έχουν φτάσει στο προορισμό τους τότε τους αυξάνω την παράμετρο speed η οποία θα δώσει πιο μεγάλη μεταβολή στην αλλαγή των συντεταγμένων X και Y, τις οποίες όπως ανάφερα και πιο πάνω προκύπτουν από το sin(θ)*speed και cos(θ) *speed αντίστοιχα. Τέλος, ο αλγόριθμος τερματίζει όταν όλες οι ομάδες έχουν ομαδοποιηθεί σε κάποιο χώρο της περιοχής που ορίσαμε.



Σχήμα 4.3 Δημιουργία τριών ομάδων

## 4.3 Αλγοριθμός για την εφαρμογή «Follow The Leader»

### 4.3.1 Γενικά για το πρόβλημα

Σε αυτήν την εφαρμογή το γενικό σενάριο είναι ότι, υπάρχει μια περιοχή η οποία καλύπτεται από κάποιο συγκεκριμένο αριθμό αισθητήρων. Οι θέσεις των αισθητήρων καθορίζονται τυχαία σε όλη την έκταση της περιοχής δίνοντας τους μια τυχαία αρχική κατεύθυνση και ταχύτητα. Σε αυτήν την εφαρμογή υπάρχει η έννοια του «αρχηγού». Ο «αρχηγός» αναμένει σε ένα προκαθορισμένο σημείο της περιοχής τους άλλους κόμβους, ούτως ώστε να τοποθετεί ο ένας κόμβος πίσω από τον άλλο. Κάθε κόμβος κινείται ανεξάρτητα προς τον «αρχηγό» υπολογίζοντας την κίνηση του, βασιζόμενο στις τριγωνομετρικές συναρτήσεις sin (θ) και cos (θ), αφού πρώτα υπολογιζόταν η γωνία θ (βλέπε 4.2.3). Στην συνέχεια αφού όλοι οι κόμβοι μπουν σε μια ευθεία γραμμή τότε ο «αρχηγός» ξεκίνα να κινείται σε τυχαίους προορισμούς μέσα στην περιοχή και ταυτόχρονα όλοι οι άλλοι κόμβοι τον ακολουθούν σε κάθε του κίνηση κρατώντας συνάμα και μια απόσταση ασφαλείας.

### 4.3.2 Αλγόριθμος

1. Καθορισμός αριθμών κόμβων, Cover Range/ Sense Range, speed και όρια περιοχής(area).

2. Με βάση τις πιο πάνω τιμές γίνεται ο καθορισμός των θέσεων των κόμβων με τυχαίο τρόπο σε συγκεκριμένες συντεταγμένες.

3. While (Μέχρι οι κόμβοι δεν είναι σε μια ευθεία πίσω από τον «αρχηγό») {

   3.1 Ξεκίνησε την κίνηση κάθε κόμβου προς την προκαθορισμένη

θέση που βρίσκεται ο «αρχηγός» χρησιμοποιώντας sin (θ) και cos (θ) για να μεταβάλεις τις συντεταγμένες X και Y αντίστοιχα.

}

RandomDest= Set a random destination ();

4. While (true) {

If («αρχηγός» δεν έφτασε στο RandomDest) {

4.1 Ξεκίνησε την κίνηση του κόμβου- «αρχηγού» χρησιμοποιώντας το μοντέλο κινητικότητας Random Waypoint

4.2 Κάθε κίνηση που κάνει ο «αρχηγός» ενημερώνεται ο κόμβοςΑ που βρίσκεται πίσω του με την προηγούμενη θέση του «αρχηγού». Μέτα ο κόμβος Α κάνει το ίδιο στον κόμβο Β που βρίσκεται πίσω του και ούτω κάθε εξής. Μέχρι τον τελευταίο κόμβο της ευθείας.

} else {

4.3 Όρισε καινούργιο Random Destination

RandomDest= Set a random destination ();

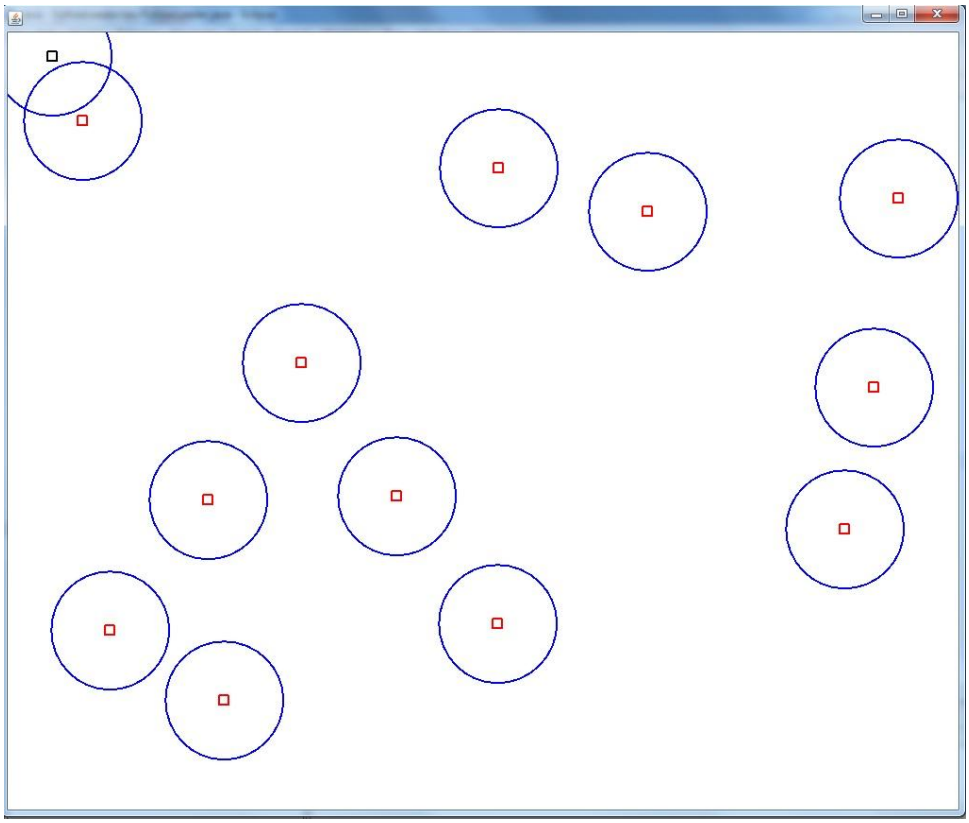
}

}

### 4.3.3 Επεξήγηση αλγορίθμου

Παρόμοια με το 4.3.2 η εφαρμογή μας ξεκίνα με το να καθορίσουμε κάποιες παραμέτρους που διαδραματίζουν σημαντικό ρόλο στην όλη φιλοσοφία της κίνησης και ομαδοποίησης των κόμβων. Στην αρχή καθορίσαμε τα όρια της περιοχής που θα

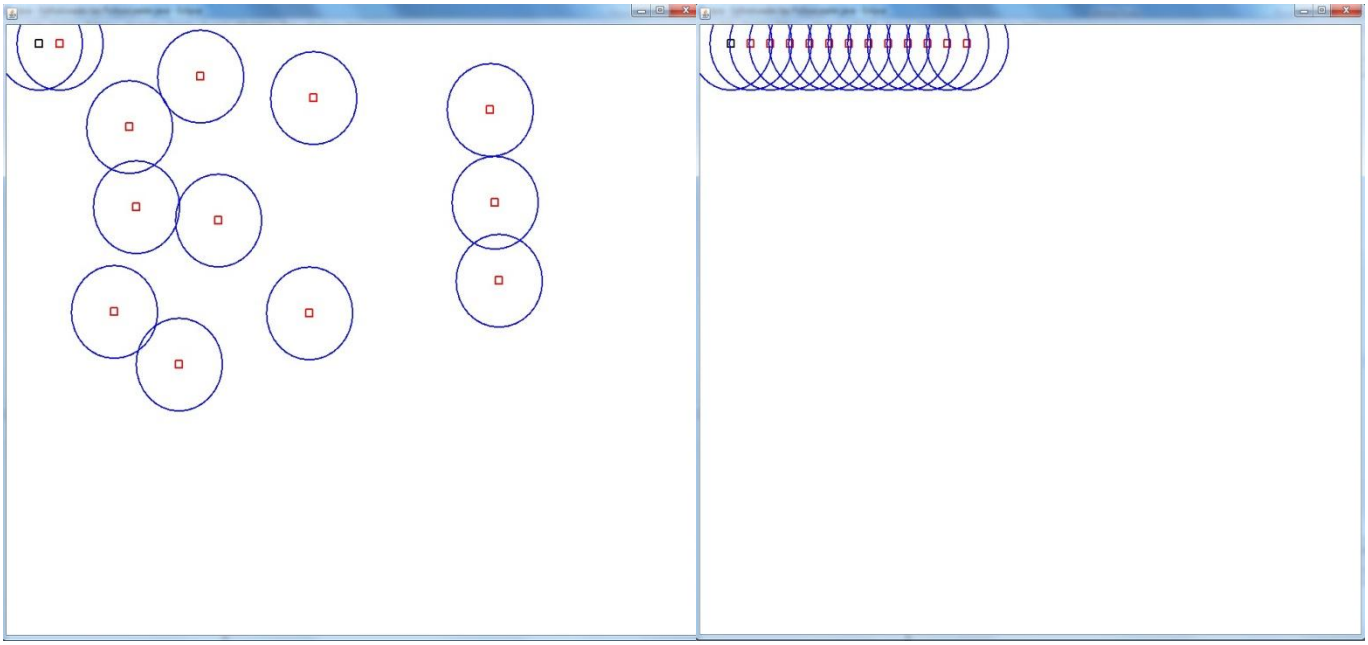
κινούνταν οι κόμβοι. Στην συνέχεια θα έπρεπε να καθοριστούν τα κυριότερα χαρακτηριστικά των κόμβων, τα οποία είναι το Cover Range/Sense Range και το μέγεθος του αισθητήρα. Με βάση αυτά, οι κόμβοι μας θα διασκορπίζονταν μέσα στην περιοχή με ένα πιο ομοιόμορφο τρόπο ούτος ώστε να δείχνει μια πιο ρεαλιστική κάλυψη της περιοχής.



Σχήμα 4.4 Αρχικές θέσεις των 12 κόμβων

Αφού έχουν τοποθετηθεί οι κόμβοι σε τυχαίες θέσεις μέσα στην περιοχή (βλέπε Σχήμα 4.4), θα πρέπει να κινηθούν προς συγκεκριμένο προορισμό έτσι ώστε να μπουν όλοι σε μια ευθεία. Ο καθορισμός του προορισμού για κάθε κόμβο έκτος από αυτό του «αρχηγού», γινόταν με βάση την ευκλείδεια απόσταση των κόμβων από τον «αρχηγό». Από τις υποθέσεις που έκανα στο κεφάλαιο 3.2, θεώρησα ότι υπάρχει κάποιος κόμβος sink ο οποίος γνωρίζει όλες τις θέσεις των κόμβων μέσα σε μια περιοχή και ενημερώνει

κάθε κόμβο ανάλογα με την απόσταση που βρίσκεται από τον αρχηγό, σε ποια θέση της ευθείας που θα δημιουργηθεί από όλους τους κόμβους θα τοποθετηθεί (βλέπε Σχήμα 4.5). Ο «αρχηγός» (με μαύρο χρώμα) τοποθετείται αρχικά σε προκαθορισμένη θέση μέχρι όλοι κόμβοι μα μπουν σε μια ευθεία.



Σχήμα 4.5 Κίνηση των κόμβων μέχρι να μπουν σε ευθεία

Η κίνηση των κόμβων στηρίζεται στις τριγωνομετρικές συναρτήσεις sin (θ) και cos (θ), αφού πρώτα υπολογιζόταν η γωνία θ. Αυτή η γωνιά θ υπολογιζόταν ως εξής:

$\lambda_{\varepsilon\varphi}=(Y_{destination\_position}-Y_{current\_position})/\ (X_{destination\_position}-X_{current\_position})$

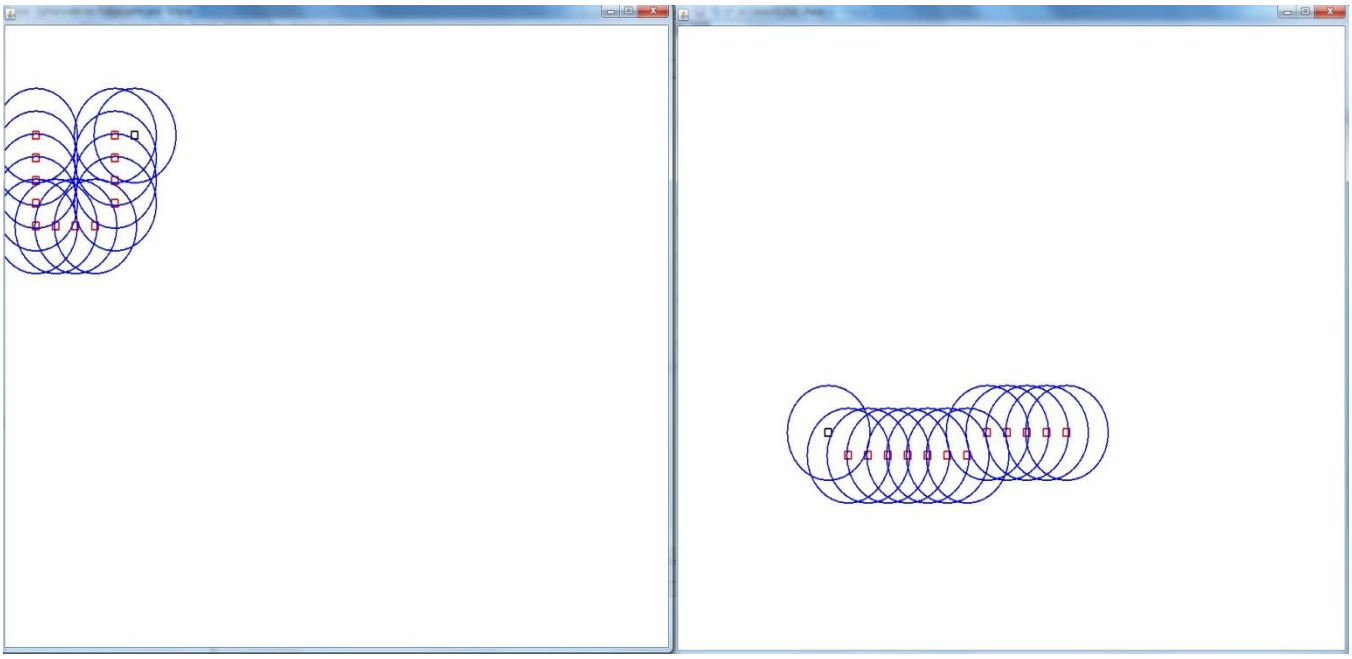
Ξερώντας από την αναλυτική γεωμετρία ότι η λεφ= εφ (θ)

$\theta^{o}=$ τοξεφ($\lambda_{\varepsilon\varphi}$),

Οπόταν υπολογίζοντας την γωνιά θ χρησιμοποιούσαμε το sin (θ)*speed για να καθορίσουμε την κίνηση του X και το cos (θ) *speed για να καθορίσουμε την κίνηση του Y.

Όταν πλέον τοποθετηθούν οι κόμβοι, όπως δείχνει το Σχήμα 4.5 (δεξιά), αρχίζει την κίνηση του ο «αρχηγός». Η κίνηση που κάνει ο «αρχηγός» στηρίζεται στο μοντέλο κινητικότητας Random Waypoint όπου του ανατίθεται ένας τυχαίος προορισμός. Κατά την κίνηση του «αρχηγού» γίνεται μετάδοση προς τα πίσω όλων των κινήσεων των κόμβων μέχρι τον τελευταίο κόμβο που έχει ομαδοποιηθεί. Δηλαδή την προηγούμενη θέση του αρχηγού θα την πάρει ο κόμβος Α που βρίσκεται πίσω του, την προηγούμενη θέση του κόμβου Α θα την πάρει ο κόμβος Β που βρίσκεται πίσω του και ούτω κάθε εξής. Γίνονται έλεγχοι με βάση το γεγονός ότι ο αρχηγός δεν μπορεί να κάνει κίνηση προς τα πίσω (να κάνει όπισθεν) αλλά ούτε να διαπεράσει κάποιο άλλο κόμβο. Πάντα κινείται σε συντεταγμένες που δεν είναι καταλυμένες από άλλους κόμβους.

Σκοπός της εφαρμογής ήταν για να δείξουμε την ομαδοποίηση των κόμβων σε προκαθορισμένες θέσεις και η ακολούθησης του αρχηγού. Για αυτό το λόγο δεν θέσαμε τον αλγόριθμο να σταματά σε κάποιο σημείο αλλά να τρέχει συνεχώς.



Σχήμα 4.6 Κινήσεις των κόμβων

## 4.4 Αποτελέσματα-Ανάλυση αποτελεσμάτων

### 4.4.1 Αλγόριθμος «Find Your Friend»

Για την αξιολόγηση του αλγορίθμου που αναπτύχθηκέ, εκτέλεσα αρκετές μετρήσεις για να εξετάσω την αποτελεσματικότητα του. Για τις ανάγκες αυτής της αξιολόγησης, έτρεξα διάφορα αντιπροσωπευτικά σενάρια, μετρώντας κάθε φορά το χρόνο που χρειάστηκε ο αλγόριθμός για να τερματίσει. Το σημαντικότερο από την διαδικασία αυτή, ήταν ο καθορισμός των παραμέτρων και η δημιουργία σεναρίων για εξαγωγή αποτελεσμάτων. Οι παράμετροι που χρησιμοποιήσαμε για να συλλέξουμε τα δεδομένα ήταν:

- Το εμβαδό της περιοχής (Area)
- Η εμβέλεια κάλυψης και αίσθησης (Cover/Sense Range)
- Ο αριθμός των ομάδων (Number of Teams)
- Η ταχύτητα των κόμβων (Speed)
- Κατανάλωση Ενέργειας (Joule)

Σκοπός μας ήταν να αναλύσουμε την συμπεριφορά των αλγορίθμων. Γι αυτό τον λόγο δεν περιοριστικά μόνο σε απλές γραφικές παραστάσεις, π.χ. Area Vs Time, άλλα σε γραφικές παραστάσεις οι οποίες δείχνουν την ταυτόχρονη μεταβολή δύο παραμέτρων σε σχέση με το χρόνο, π.χ. Area Vs Time Increasing Cover/Sense Range.

Στα πιο κάτω σενάρια που ακλουθούν, οι μετρήσεις αφορούν το χρόνο που χρειάστηκε ο αλγόριθμος για να τερματίσει, δηλαδή είναι ο χρόνος μέχρι οι κόμβοι αισθητήρων ομαδοποιηθούν σε ομάδες με κοινά χαρακτηριστικά και την κατανάλωση της ενέργειας που χρειάζονταν οι κόμβοι. Λόγω της κίνησης που κάνουν οι κόμβοι με βάση το μοντέλο κινητικότητας Random Waypoint, έπαιρνα μέσους χρόνους (average time) τερματισμού του αλγόριθμου. Αυτό το έκανα διότι λόγω της τυχαιότητας των προορισμών των κόμβων, κάποιοι χρόνοι δεν θα ήταν αντιπροσωπευτική. Οπόταν παίρνοντας μέσους

χρόνους θα είχαμε μια καλύτερη κατανομή των αποτελεσμάτων.

Για τον υπολογισμό του μέσου χρόνου (average time), έτρεχα ένα σενάριο με τις ίδιες τίμες για είκοσι (20) συνεχόμενες φόρες και στην συνέχεια το συνολικό χρόνο τερματισμού των διαιρούσα δια είκοσι(20). Στην συνέχεια άλλαζα τις τίμες των παραμέτρων που εξέταζα και εφάρμοζα την ίδια διαδικασία για να πάρω μέσο χρόνο για τις νέες τιμές των παραμέτρων. Συνολικά δημιούργησα πέντε ζεύγη τιμών των παραμέτρων που εξέταζα και έτρεχα των αλγόριθμο δέκα φορές για κάθε ζεύγος.

Όσον αφόρα τον τρόπο υπολογισμού της κατανάλωσης της ενέργειας, αυτός προέκυπτε από τον υπολογισμό της συνολικής απόστασης που διανύσαν οι κόμβοι μέχρι να ομαδοποιηθούν και των σημάτων μετάδοσης (transmit) και παραλαβής (receive) που αντάλλαζαν οι κόμβοι κατά την διάρκεια που δημιουργούσαν τις διάφορες υπό-ομάδες. Η κατανάλωσης ενέργειας χωριζόταν σε τρία είδη, σύμφωνα με τα πιο κάτω [8]:

- Για κίνηση των κόμβων η κατανάλωση υπολογιζόταν στα 28 J/m.
- Για μετάδοσης κάποιου σήματος χρειαζόταν 17.4 mA. Η κατανάλωσης της ενέργειας προέκυπτε από το εξής:

  $(3600x17.4x10^{-3})/3.7V= 16.92$ Joules
- Για παραλαβή κάποιου σήματος χρειαζόταν 19.4 mA. Η κατανάλωσης της ενέργειας προέκυπτε από το εξής:

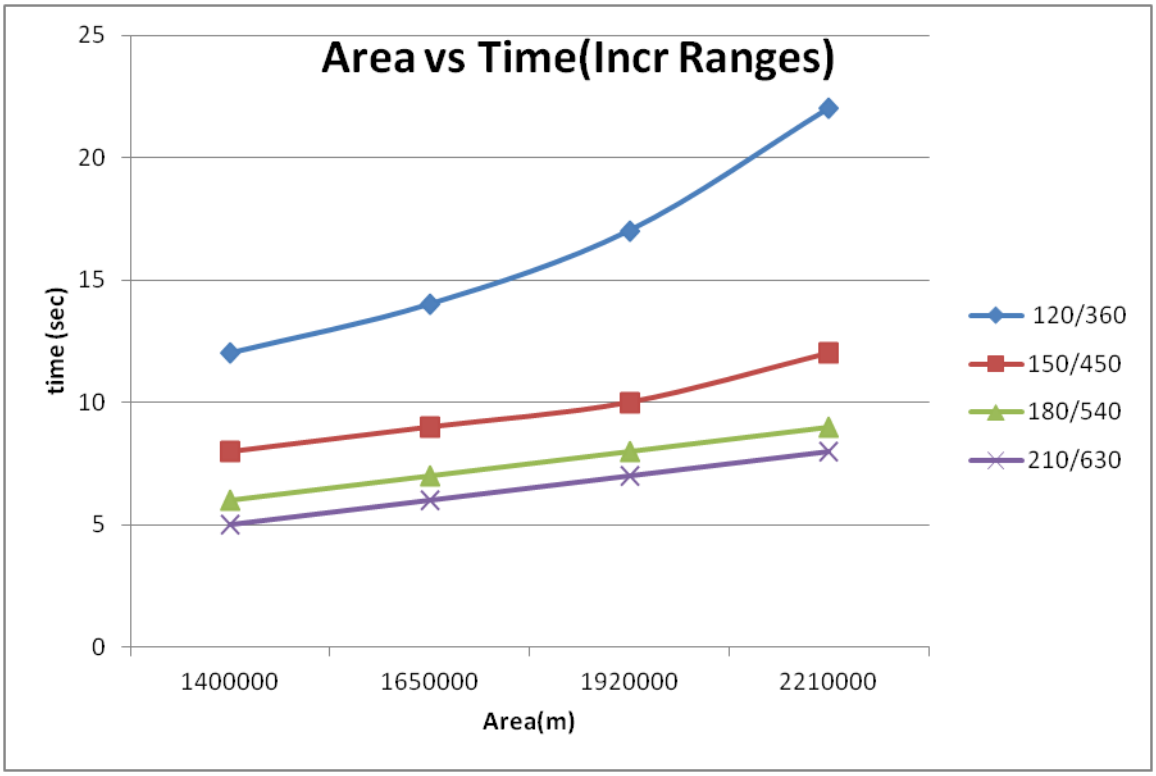
  $(3600x19.4x10^{-3})/3.7V= 18.87$ Joules

**Σενάριο στο οποίο μεταβάλλονται οι παράμετροι Area και Cover/Sense Range**

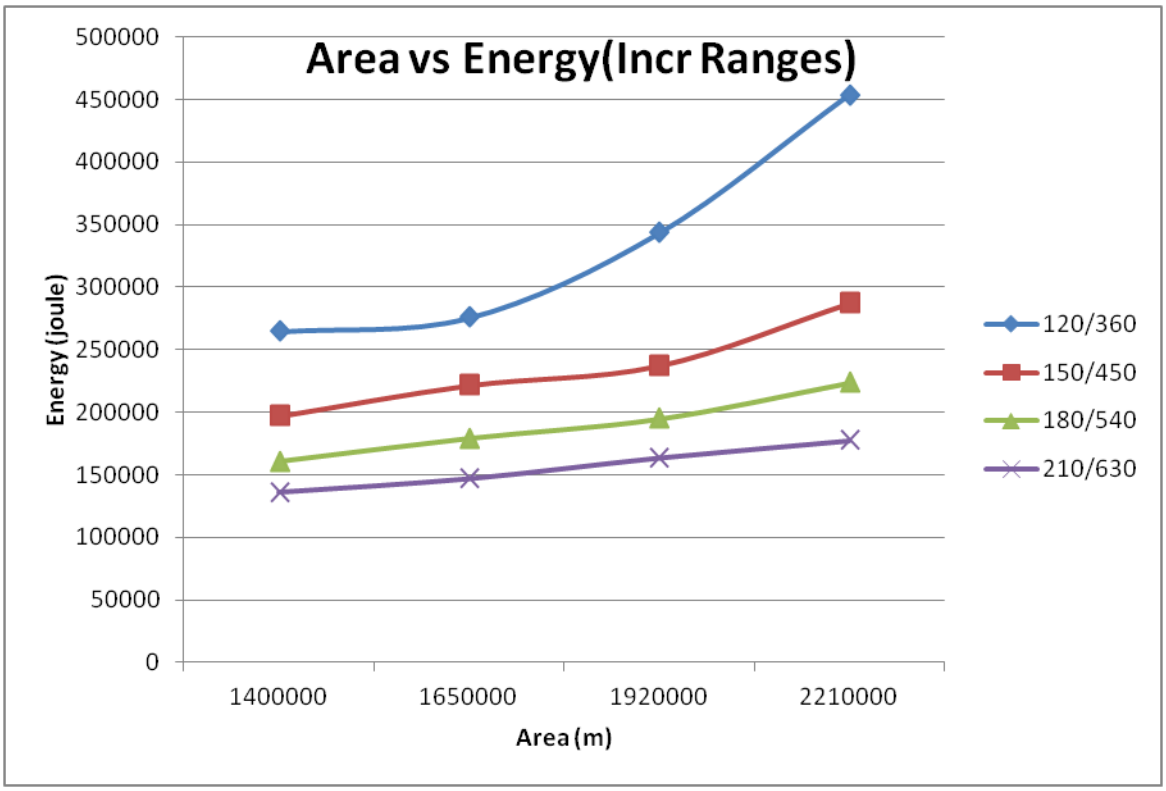
Σε αυτό το σενάριο προσομοίωσα την συμπεριφορά των κόμβων όταν μεταβάλλονται οι παράμετροι Area και Cover/Sense Range σε σχέση με τον χρόνο τερματισμού του αλγορίθμου και την ενέργεια που κατανάλωσαν. Στην αρχή για κάθε περιοχή υπολόγιζα το χρόνο που θα τερμάτιζε ο αλγόριθμος για διαφορετικά Cover/Sense Range. Πρακτικά, για κάθε μία από τις τέσσερις περιοχές που όρισα υπολόγιζα τέσσερις μέσους χρόνους (average time) για τέσσερα διαφορετικά Cover/Sense Range. Οι γραφικές παραστασείς που προέκυψαν φαίνονται πιο κάτω.

Ανάθεση τιμών στις παραμέτρους:

| | |
|---|---|
| Area | Var |
| Range | Var |
| speed | 25 m/s |
| #Teams | 3 |




Διάγραμμα 4.1: Area Vs Time (Incr Ranges)




Διάγραμμα 4.2: Area Vs Energy (Incr Ranges)

Οι πιο πάνω γραφικές παραστάσεις αφορούν στο χρόνο τερματισμού του αλγόριθμού χρησιμοποιώντας διαφορετικές τιμές στην παράμετρο Cover/Sense Range (Διάγραμμα 4.1) και στην συνολική κατανάλωση της ενέργειας που υπήρξε από όλους τους κόμβους στο δίκτυο (Διάγραμμα 4.2). Αυτό που παρατηρούμε είναι ότι για όλες τις τιμές της παραμέτρου Cover/Sense Range, καθώς αυξάνουμε το εμβαδό της περιοχής, αυξάνεται και ο χρόνος μέχρι να τερματίσει ο αλγόριθμος. Αυτό προκύπτει από το γεγονός ότι μεγαλώνοντας την περιοχή οι κόμβοι μπορούν να κινηθούν σε πιο μακρινές θέσεις , οπόταν αυξάνεται ο χρόνος της ομαδοποίησης. Το σημαντικότερο όμως είναι το γεγονός ότι αυξάνοντας την τιμή της παραμέτρου Cover/Sense Range παρατηρούμε ότι ο χρόνος αυτός μειώνεται. Αυτό συμβαίνει διότι όσο πιο μεγάλη είναι η τιμή της παραμέτρου Cover/Sense Range τόσο πιο εύκολα μπορούν να ομαδοποιηθούν οι κόμβοι μας διότι θα μπορούν να κάνουν Sense σε μεγαλύτερες αποστάσεις. Επιπρόσθετα, για το διάγραμμα της ενέργειας παρατηρούμε ότι η αύξηση της παραμέτρου Cover/Sense Range επιφέρει μείωση στην ενέργεια που καταναλώνουν οι κόμβοι για να παράγουν την κίνηση τους και να αλλάξουν μηνύματα όταν δημιουργούν υπό- ομάδες. Αυτό συμβαίνει διότι με το να

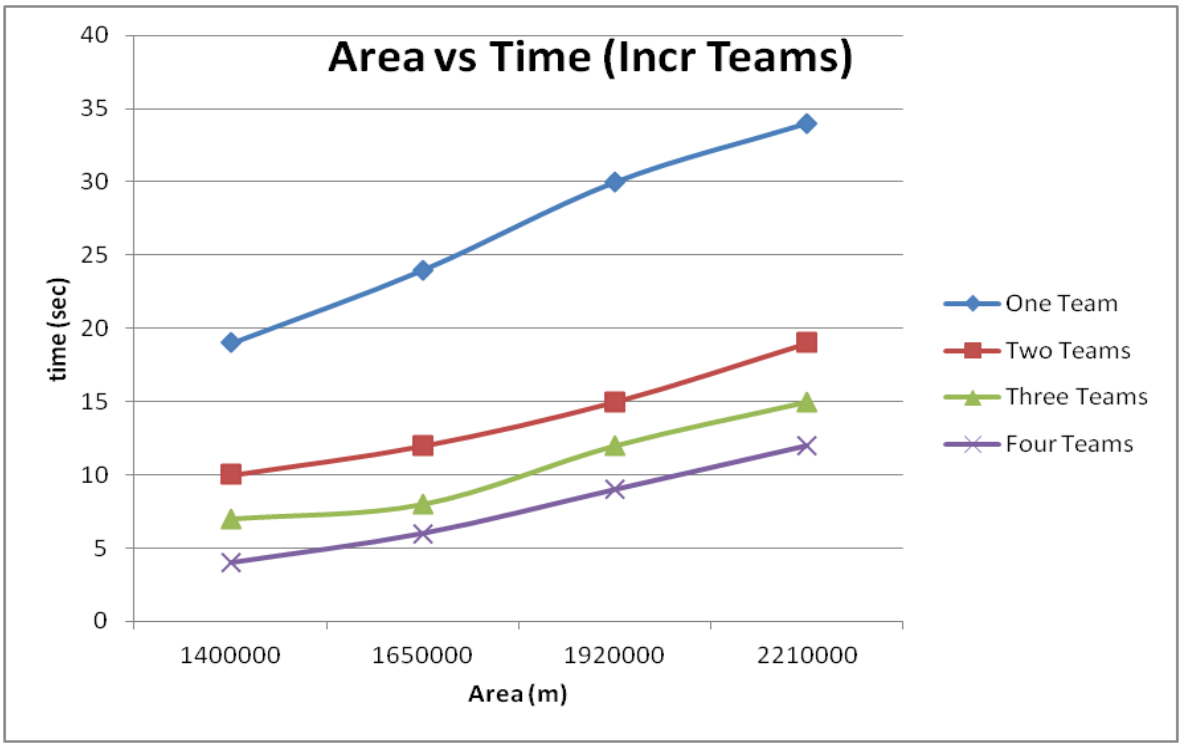
γίνεται Sense σε μεγαλύτερη ακτίνα, μειώνονται κατά πολύ οι κινήσεις που θα χρειαστούν να κάνουν οι κόμβοι για να ομαδοποιηθούν με αποτέλεσμα να μειώνεται η συνολική ενέργεια που χρησιμοποιούν.

**Σενάριο στο οποίο μεταβάλλονται οι παράμετροι Area και Number Of Teams**

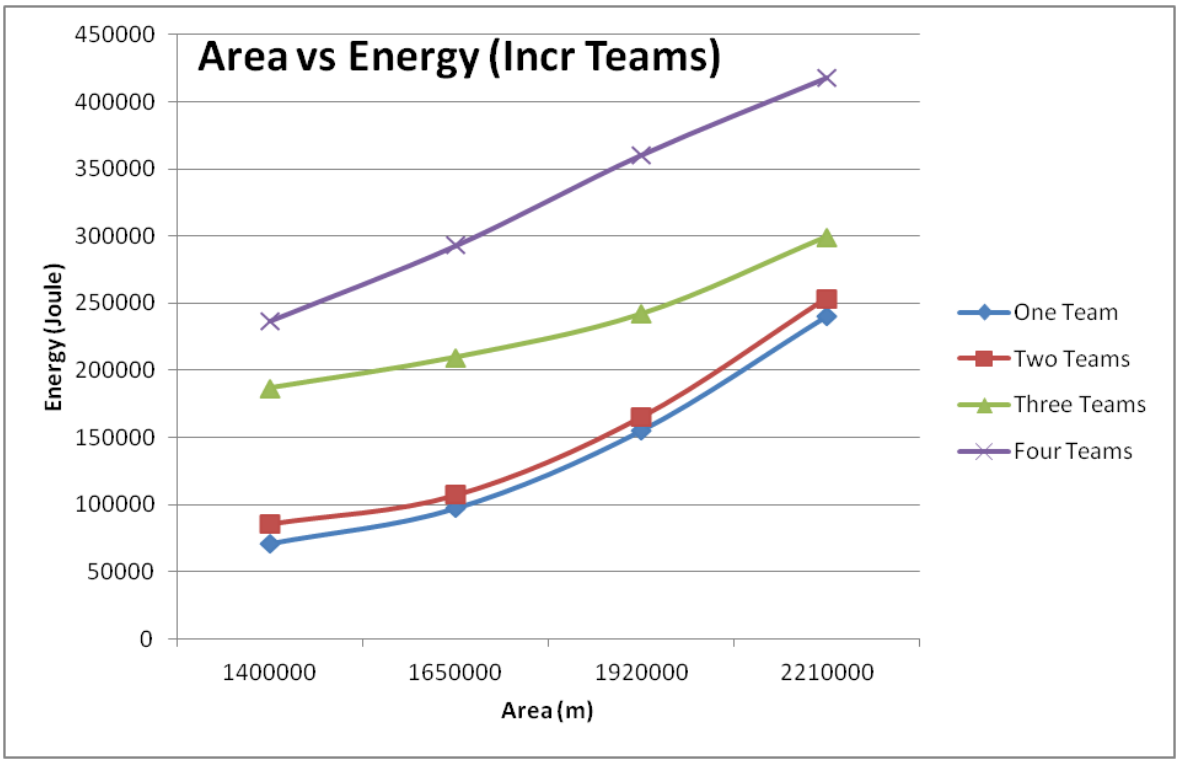
Σε αυτό το σενάριο προσομοίωσα την συμπεριφορά των κόμβων όταν μεταβάλλονται οι παράμετροι Area και Number Of Teams σε σχέση με τον χρόνο τερματισμού του αλγορίθμου και την ενέργεια που κατανάλωσαν. Στην αρχή για κάθε περιοχή υπολόγιζα το χρόνο που θα τερμάτιζε ο αλγόριθμος για διαφορετικά Number Of Teams. Πρακτικά, για κάθε μία από τις τέσσερις περιοχές που όρισα, υπολόγιζα τέσσερις μέσους χρόνους (average time) για τέσσερις διαφορετικές ομάδες. Οι ομάδες αποτελούνταν από ίσον αριθμό κόμβων. Οι γραφικές παραστασείς που προέκυψαν φαίνονται πιο κάτω.

Ανάθεση τιμών στις παραμέτρους:

| | |
|---|---|
| Area | Var |
| Range | 120/360 |
| speed | 25 m/s |
| #Teams | Var |




Διάγραμμα 4.3: Area Vs Time (Incr Number of Teams)




Διάγραμμα 4.4: Area Vs Energy (Incr Number of Teams)

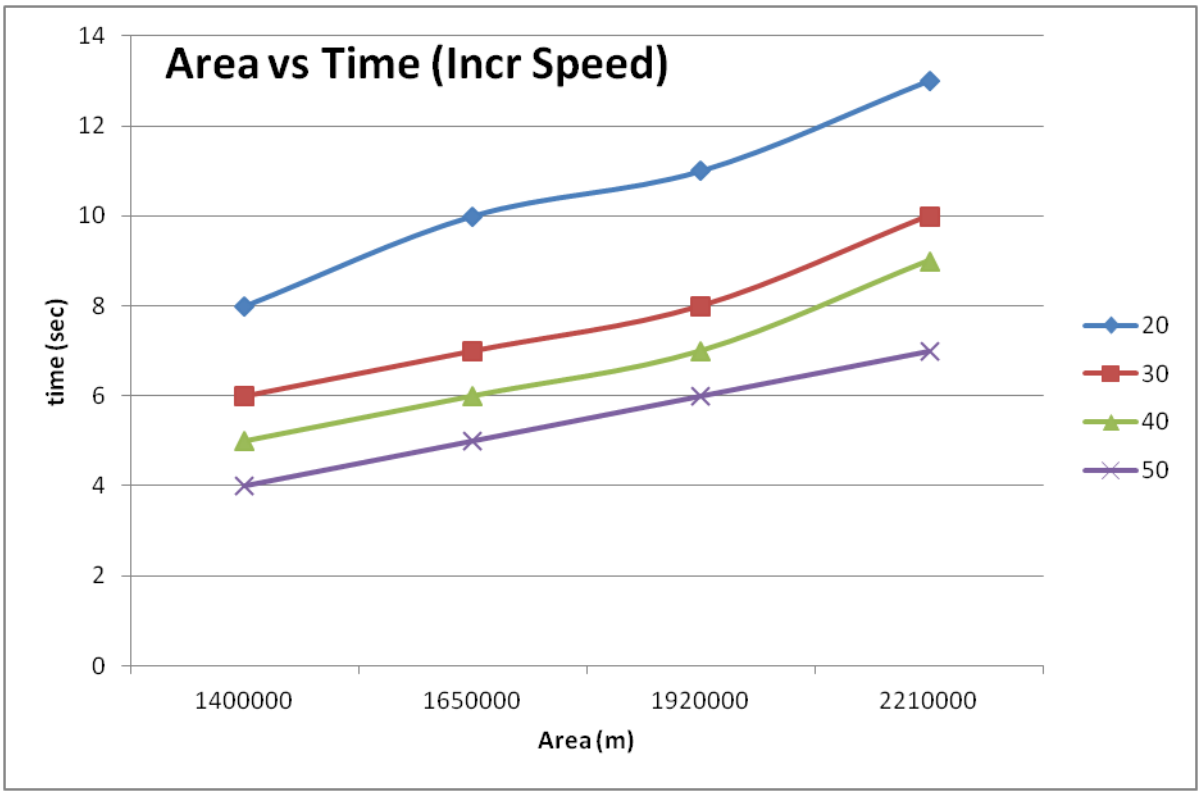
Οι πιο πάνω γραφικές παραστάσεις αφορούν στο χρόνο τερματισμού του αλγόριθμού χρησιμοποιώντας διαφορετικές τιμές στην παράμετρο #Teams (Διάγραμμα 4.3) και στην συνολική κατανάλωση της ενέργειας που υπήρξε από όλους τους κόμβους στο δίκτυο (Διάγραμμα 4.4). Από το Διάγραμμα 4.3 προκύπτει ότι χρησιμοποιώντας μόνο μια ομάδα μέσα σε μια περιοχή είναι πολύ χρονοβόρο μέχρι οι κόμβοι αυτής της ομάδας να ομαδοποιηθούν. Αυτό το φαινόμενο δημιουργείται από το γεγονός ότι οι κόμβοι υλοποιώντας το μοντέλο κινητικότητας Random Waypoint μπορούν να κινούνται σε διαφορετικούς προορισμούς. Όμως δεν έρχονται σε επαφή με άλλους κόμβους άλλης ομάδας (αφού δεν υπάρχουν), έτσι ώστε να μπορούν να αλλάζουν την κατεύθυνση τους, με αποτέλεσμα ο μόνος τρόπος να αλλάξουν κατεύθυνση είναι να φτάσουν στο προορισμό τους (destination point), έτσι ώστε να θέσουν νέο προορισμό. Πράγμα που δημιουργεί καθυστέρηση στο τερματισμό του αλγορίθμου. Η καθυστέρηση αυξάνεται όταν μεγαλώνει ακόμη περισσότερο η παράμετρος Area αφού πλέον οι κόμβοι μπορούν να θέτουν ακόμη μεγαλύτερα destination points. Τα πράγματα διαφοροποιούνται όταν προσθέσω περισσότερες ομάδες σε μια περιοχή. Παρατηρώ ότι οι χρόνοι μειώνονται δραστικά και αυτό οφείλεται στο γεγονός ότι οι κόμβοι διαφορετικών ομάδων ένεκεν της πιθανής εισδοχής του ενός στην εμβέλεια του άλλου, αλλάζουν κατεύθυνση και γίνεται πιο γρήγορα η ομαδοποίηση των ομάδων. Από την πλευρά της κατανάλωσης της ενέργειας τα αποτελέσματα δείχνουν μια διαφορετική συμπεριφορά. Αυτό που παρατηρείται είναι ότι μεγαλώνοντας τον αριθμό των ομάδων μέσα στο δίκτυο, αυξάνεται και η συνολική ενέργεια που θα πρέπει να χρησιμοποιήσουν όλοι οι κόμβοι για να κινηθούν και να ανταλλάξουν μηνύματα. Αυτό είναι πολύ λογικό διότι προσθέτοντας κόμβους μέσα στο δίκτυο οι συνολικές ανάγκες σε ενέργεια αυξάνονται, οπόταν, ναι μεν μειώνεται ο χρόνος ομαδοποίησης αλλά από την άλλη αυξάνεται η κατανάλωσης της ενέργειας.

**Σενάριο στο οποίο μεταβάλλονται οι παράμετροι Area και Speed**

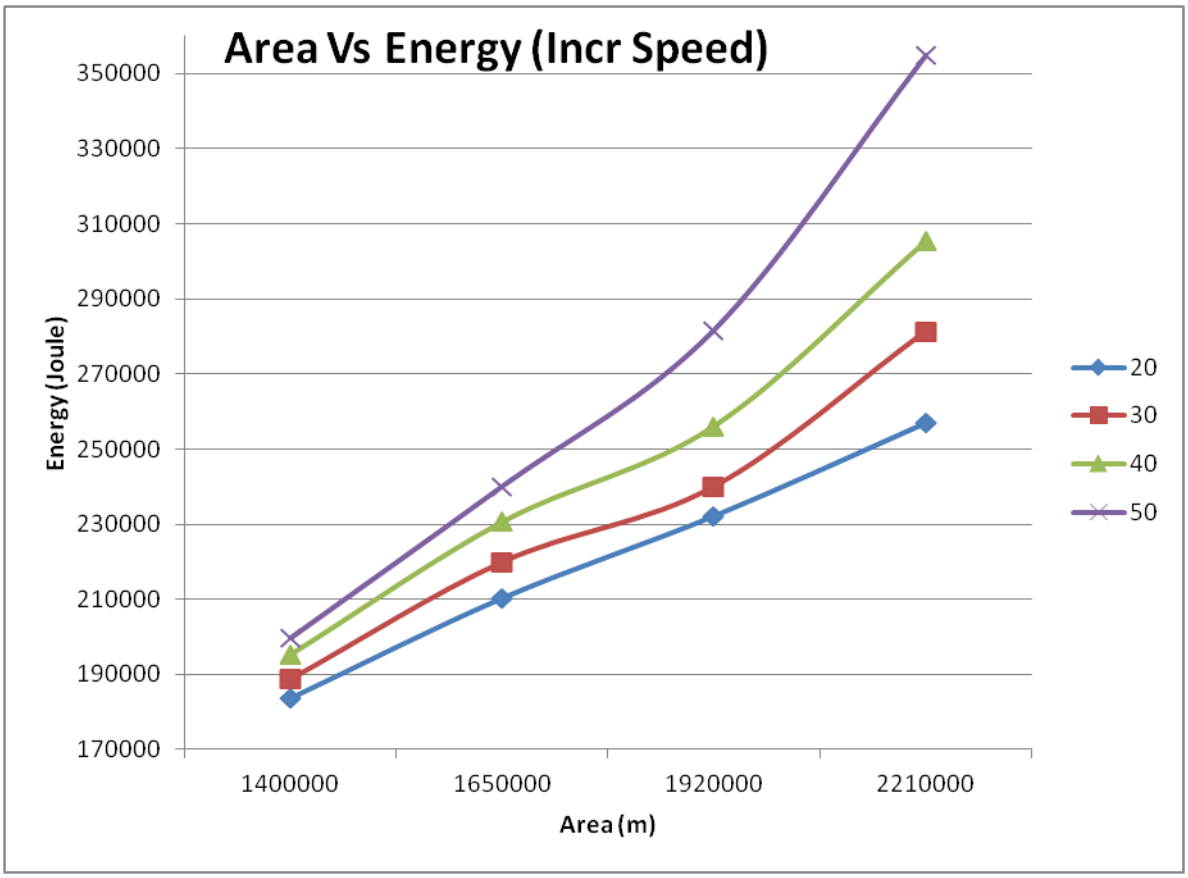
Σε αυτό το σενάριο προσομοίωσα την συμπεριφορά των κόμβων όταν μεταβάλλονται οι παράμετροι Area και Speed σε σχέση με τον χρόνο τερματισμού του αλγορίθμου και την ενέργεια που κατανάλωσαν. Στην αρχή για κάθε περιοχή υπολόγιζα το χρόνο που θα τερμάτιζε ο αλγόριθμος για διαφορετικά Speeds. Πρακτικά, για κάθε μία από τις τέσσερις περιοχές που όρισα υπολόγιζα τέσσερις μέσους χρόνους (average time) για τέσσερις διαφορετικές ταχύτητες. Οι γραφικές παραστασείς που προέκυψαν φαίνονται πιο κάτω.

Ανάθεση τιμών στις παραμέτρους:

| | |
|---|---|
| Area | Var |
| Range | 120/360 |
| speed | Var |
| #Teams | 3 |




Σχήμα 4.5: Area Vs Time (Incr Speed)




Σχήμα 4.6: Area Vs Energy (Incr Speed)

Οι πιο πάνω γραφικές παραστάσεις αφορούν στο χρόνο τερματισμού του αλγόριθμού χρησιμοποιώντας διαφορετικές τιμές στην παράμετρο speed (Διάγραμμα 4.5) και στην συνολική κατανάλωση της ενέργειας που υπήρξε από όλους τους κόμβους στο δίκτυο (Διάγραμμα 4.6). Από το Διάγραμμα 4.5 προκύπτει ότι στην αρχική τιμή της παραμέτρου Area, οι χρόνοι τερματισμού για όλες τις τιμές της παραμέτρου Speed είναι πολύ κοντινές μεταξύ τους. Αυτό οφείλεται στο γεγονός ότι για την συγκριμένη τιμή δεν παίζει και πολύ μεγάλο ρόλο η τιμή της ταχύτητας  διότι η περιοχή είναι σχετικά μικρή. Το ενδιαφέρον προκύπτει καθώς η τιμή της παραμέτρου Area αυξάνεται. Αυτό που παρατηρούμε είναι ότι στα τέσσερα γραφήματα των τιμών των ταχυτήτων έχουμε μια αύξηση στο χρόνο τερματισμού του αλγορίθμου. Αυτό οφείλεται στο γεγονός ότι μεγαλώνει η περιοχή που έχει ως συνέπεια οι κόμβοι να κινούνται σε πιο μεγάλες αποστάσεις μέχρι να ομαδοποιηθούν. Επιπρόσθετα προκύπτει ότι όσο πιο μεγάλη ταχύτητα αποκτούν οι κόμβοι στην περιοχή τόσο πιο μικρή καθυστέρηση στην ομαδοποίηση θα υπάρξει. Από

την πλευρά της κατανάλωσης της ενέργειας τα αποτελέσματα δείχνουν μια διαφορετική συμπεριφορά. Αυτό που παρατηρείται είναι ότι μεγαλώνοντας την τιμή της ταχύτητας αυξάνεται κατά πολύ η συνολική ενέργεια που θα πρέπει να χρησιμοποιήσουν όλοι οι κόμβοι για να κινηθούν και να ανταλλάξουν μηνύματα. Αυτό προκύπτει από το γεγονός ότι αυξάνοντας την ταχύτητα η ενεργεία που χρειάζεται ώστε να διατηρείτε αύτη η ταχύτητα είναι μεγαλύτερη σε σχέση με μια ταχύτητα που έχει μικρότερη τιμή.

Κάνοντας μια σύνοψη από τα πιο πάνω, αυτό που μπορούμε να συμπεράνουμε είναι ότι για να μειώσουμε το χρόνο ομαδοποίησης των κόμβων μέσα σε ένα Ασύρματο Δίκτυο Αισθητήρων θα πρέπει είτε να αυξήσουμε την ταχύτητα που κινούνται οι κόμβοι , είτε να αυξήσουμε τις ομάδες μέσα στο δίκτυο είτε να αυξήσουμε την εμβέλεια των κόμβων. Συμπεριλαμβάνοντας όμως και τα θέματα κατανάλωσης ενέργειας, που αποτελούν το σημαντικότερο θέμα στα Ασύρματα Δίκτυα Αισθητήρων, συμπεραίνουμε ότι αυξάνοντας την ταχύτητα και τον αριθμό των ομάδων, προκύπτει ότι οι κόμβοι χρειάζονται περισσότερα ποσοστά ενέργειας για να παράγουν την κίνηση τους και να αλλάξουν μηνύματα όταν δημιουργούν υπό- ομάδες. Σε αντίθεση με το να αυξήσουμε την εμβέλεια των κόμβων, όπου δεν παρατηρείτε το φαινόμενο να αυξάνεται η ενέργεια όσο μεγαλώνει η εμβέλεια τους.

### 4.4.2 Αλγόριθμος «Follow the Leader»

Όσον αφορά τον αλγόριθμο «Follow the Leader», τα αποτελέσματα από τις πιο κάτω γραφικές παραστάσεις αφορούσαν κυρίως το χρόνο που θα έπαιρνε μέχρι οι κόμβοι να ομαδοποιηθούν σε μία ευθεία πίσω από τον κόμβο- «αρχηγό» και την ενέργεια που καταναλώνουν οι κόμβοι παράγοντας την κίνηση τους προς το προκαθορισμένο προορισμό τους.

Ξερώντας την σπουδαιότητας της εξοικονόμησης ενέργειας σε τέτοιου είδους δίκτυα, θα έπρεπε κατά κάποιο τρόπο να εξεταστεί κατά πόσο ο πιο πάνω αλγόριθμος περιόριζε τις σπάταλες ενέργειας. Για να εξετάσω αυτήν την πτυχή δημιούργησα δυο σενάρια. Το πρώτο σενάριο αφορούσε την επιλογή με τυχαίο τρόπο της θέσης κάθε κόμβου πίσω από τον κόμβο-αρχηγό» χωρίς να λάμβανα υπόψη την απόσταση μεταξύ που βρισκόταν ο κόμβος ο όποιος θα κινείτο προς τον κόμβο-αρχηγό». Στο δεύτερο σενάριο εφάρμοσα τον αλγόριθμο «Follow the Leader», όπως ανάφερα και πιο πάνω, λαμβάνει υπόψη την απόσταση του κάθε κόμβου από τον κόμβο αρχηγό.

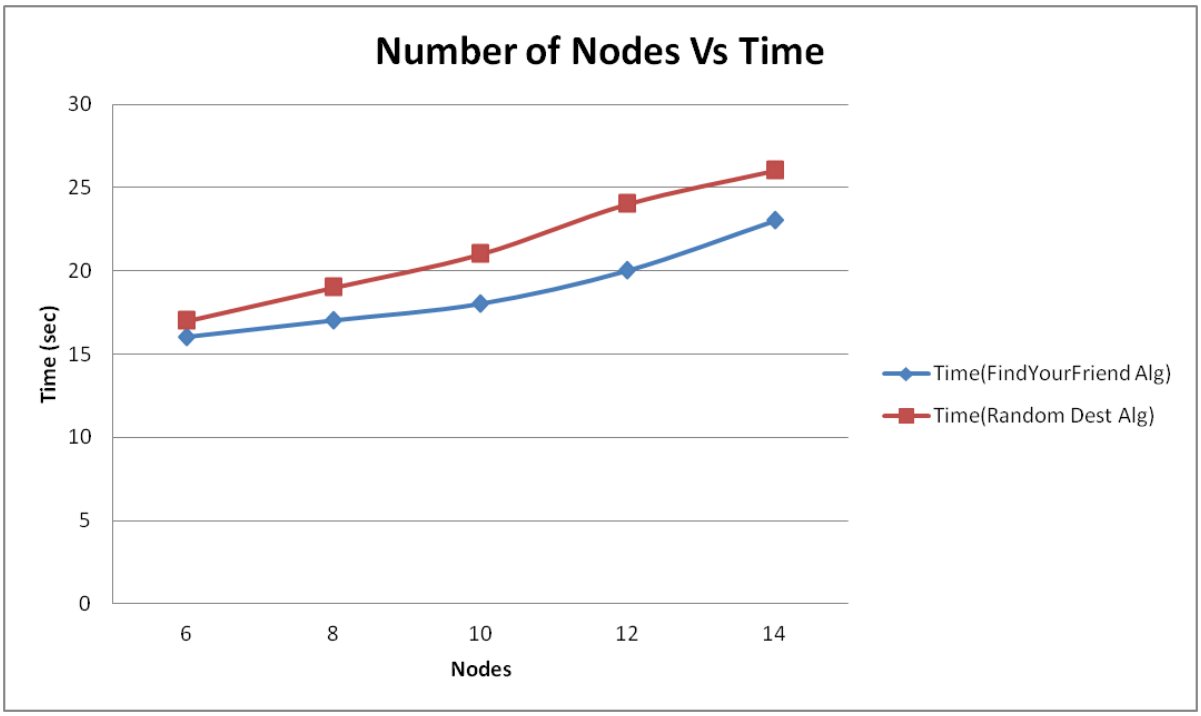
Στις πιο κάτω γραφικές παραστάσεις που ακλουθούν, οι μετρήσεις αφορούν το χρόνο που χρειάστηκε ο αλγόριθμος για να τερματίσει, δηλαδή είναι ο χρόνος μέχρι οι κόμβοι αισθητήρων ομαδοποιηθούν σε μια ευθεία πίσω από τον κόμβο- «αρχηγό». Λόγω της κίνησης που κάνουν οι κόμβοι με βάση το μοντέλο κινητικότητας Random Waypoint, έπαιρνα μέσους χρόνους (average time) τερματισμού του αλγόριθμου. Αυτό το έκανα διότι λόγω της τυχαιότητας των προορισμών των κόμβων, κάποιοι χρόνοι δεν θα ήταν αντιπροσωπευτική. Οπόταν παίρνοντας μέσους χρόνους θα είχαμε μια καλύτερη κατανομή των αποτελεσμάτων.

Για τις ανάγκες της εξαγωγής αποτελεσμάτων χρησιμοποίησα τις εξής παραμέτρους:

- Speed=15 m/s
- Energy Consumption= 28 Joule/m [8]
- Time (sec)

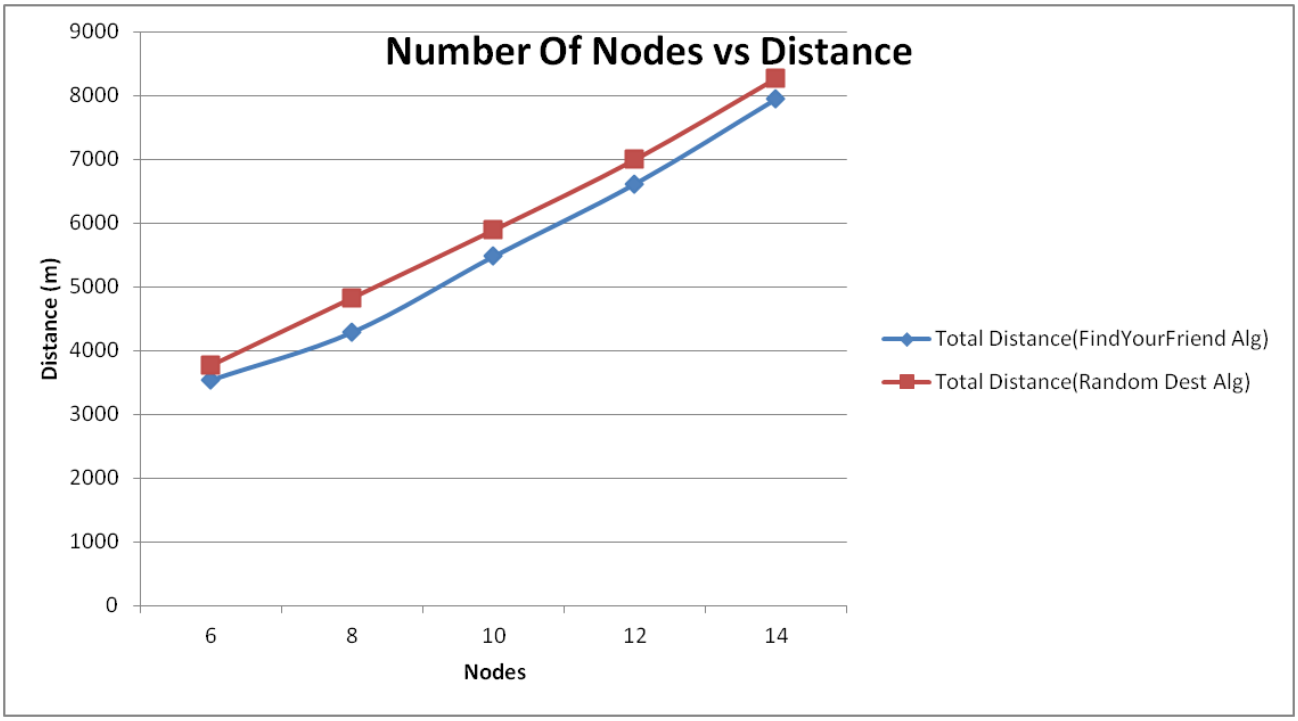
- Total Distance Traveled (m)
- Total Energy (J)

**Ανάλυση Αποτελεσμάτων:**

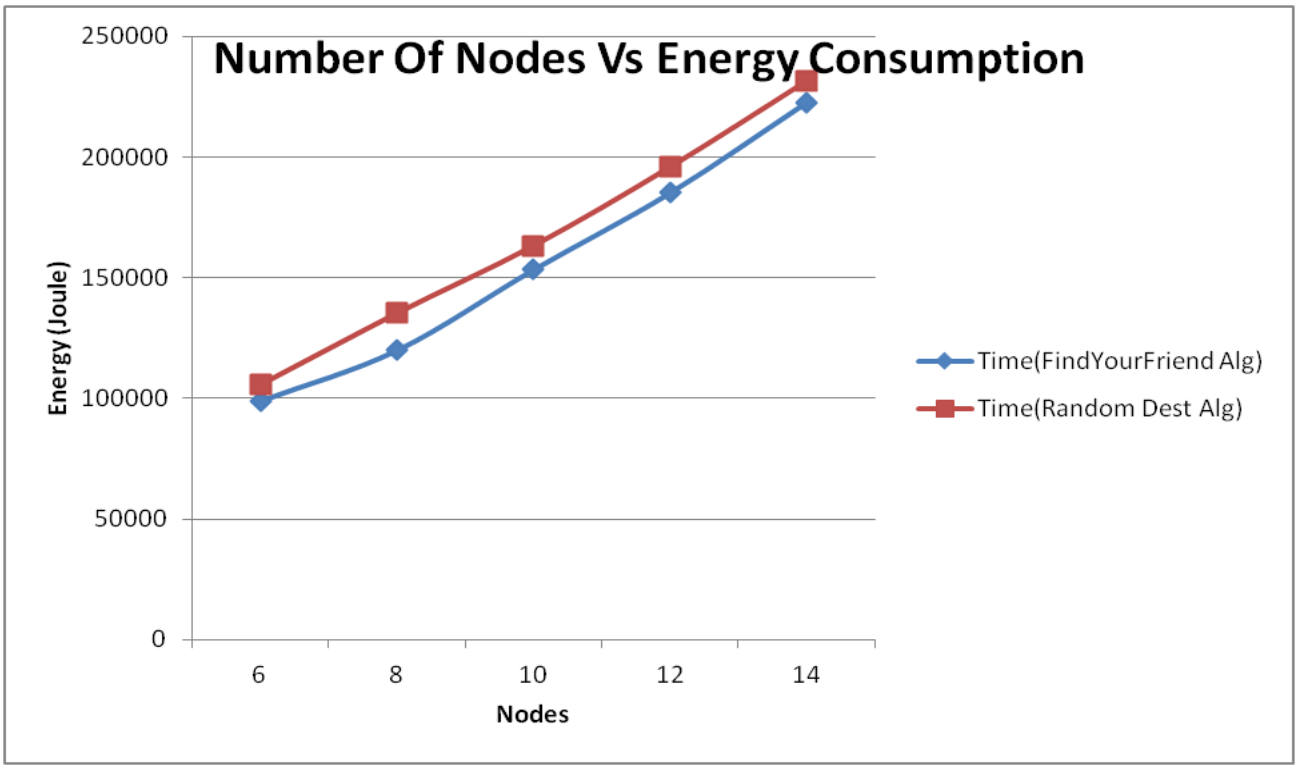



Σχήμα 4.6: Number of Nodes Vs Time

Η πιο πάνω γραφική παράσταση αφορά το χρόνο που χρειάστηκαν οι κόμβοι να ομαδοποιηθούν σε μια ευθεία πίσω από τον κόμβο-«αρχηγό». Αυτό που προκύπτει είναι ότι ο χρόνος που χρειάζονται οι κόμβοι να ομαδοποιηθούν είναι μικρότερος χρησιμοποιώντας τον αλγόριθμό «Follow the Leader» σε σχέση με κάποιο αλγόριθμό τυχαίας επιλογής θέσης. Αν και προσθέτωντας περισσότερους κόμβους μέσα στην περιοχή, αυξάνεται και ο χρόνος μέχρι να ομαδοποιηθούν οι κόμβοι λόγω των συγκρούσεων μέταξυ των κόμβων, εντουτοις οι χρόνοι παραμένουν μικρότερη για τον αλγόριθμο Follow the Leader». Άρα αυτό που προκύπτει είναι ότι διαδραματίζει σημαντικό ρόλο το γεγονός ότι οι κόμβοι κινούνται σε θέσεις με βάση την απόσταση που έχουν από την αρχική θέση που έχουν τοποθετηθεί.




Σχήμα 4.7: Number of Nodes Vs Distance




Σχήμα 4.8: Number of Nodes Vs Time

Οι πιο πάνω γραφικές παραστάσεις αφορούν τη συνολική απόσταση που κάλυψαν οι κόμβοι μέχρι να ομαδοποιηθούν (Σχήμα 4.7) και τη συνολική κατανάλωση ενέργειας που χρηστήκαν για να καταφέρουν να κινηθούν ώστε να φθάσουν στο προορισμό τους (Σχήμα 4.8). Αυτό που παρατηρούμε και στις δυο γραφικές παραστάσεις είναι το γεγονός ότι στον αλγόριθμο «Follow the Leader», οι κόμβοι κάλυψαν πιο λίγη απόσταση και κατ' επέκταση κατανάλωσαν πιο λίγη ενέργεια μέχρι να ομαδοποιούν σε σχέση με κάποιο αλγόριθμό τυχαίας επιλογής θέσης. Αυτό είναι ένα ακόμα δείγμα ότι ο αλγόριθμός λαμβάνει υπόψη θέματα κατανάλωσης ενέργειας με το να μειώνει τις κινήσεις που κάνουν οι κόμβοι μέχρι να ομαδοποιηθούν σε μια ευθεία πίσω από τον κόμβο – «αρχηγό».

# Κεφάλαιο 5

## Μελλοντική Εργασία



### 5.1 Γενικές βελτιώσεις-επεκτάσεις-εισηγήσεις

Υπάρχουν αρκετές βελτιώσεις που μπορούν να γίνουν στους αλγόριθμους που έχουν αναπτυχθεί σε αυτή την ατομική διπλωματική εργασία. Μια σημαντική επέκταση και κατά κάποιο τρόπο επιτακτική να γίνει είναι η εφαρμογή τους σε πραγματικό δίκτυο αισθητήρων για να διαφανεί πόσο αποτελεσματικοί είναι σε ένα πιο ρεαλιστικό περιβάλλον. Αυτό θα υποδείξει προβλήματα που μπορεί να υπάρχουν και τα οποία δεν μπορούν να εμφανιστούν σε ένα περιβάλλον προσομοίωσης, όπως για παράδειγμα παρεμβολές που μπορεί να υπάρχουν μεταξύ των κόμβων-αισθητήρων. Επίσης με την χρήση του γραφικού περιβάλλοντος στη γλώσσα προγραμματισμού JAVA, δεν προκύπτει κανένα αποτέλεσμα όσον αφορά τα πόσα ενέργειας που καταναλώνονται από τους κόμβους. Φυσικά κατά το σχεδιασμό των αλγορίθμων έχει ληφθεί υπόψη το δεδομένο ότι δεν υπάρχει μεγάλη πηγή ενέργειας και αποφεύγονται σε όλες τις περιπτώσεις οι περιττές κινήσεις. Όμως αυτό δεν φτάνει για μια σωστή και έγκυρη εκτίμηση των τελικών ενεργειακών σπαταλών που θα γίνουν.

Και στις δύο εφαρμογές που αναπτύχθηκαν ο τρόπος κίνησης των κόμβων στην περιοχή στηριζόταν αποκλειστικά στο Random Waypoint μοντέλο κινητικότητας. Από το κεφάλαιο 2.3, υπάρχουν αρκετά διαφορετικά μοντέλα κινητικότητας που μπορούμε να υλοποιήσουμε. Σαν μελλοντική εργασία θα ήταν κάλο να εξεταστεί κατά πόσο οι πιο

πάνω εφαρμογές θα είχαν καλύτερα αποτελέσματα χρησιμοποιώντας κάποιο άλλο μοντέλο κινητικότητας.

Μελετώντας την υπάρχουσα ερευνητική και πειραματική εργασία αντιλήφθηκα ότι για πολλά προβλήματα που αφορούν τα ασύρματα δίκτυα αισθητήρων έχουν προταθεί πάρα πολύ αλγόριθμοι. Έχουν υλοποιηθεί αλγόριθμοι για κίνηση προς συγκεκριμένο σημείο και δρομολόγιο, ακολούθησης στόχου, έλεγχου περιμέτρου και αλγόριθμοι ομαδοποίησης. Μια καινοτόμα επέκταση είναι ο συνδυασμός κάποιων από τους αλγόριθμούς έτσι ώστε να δημιουργούμε υβριδικά συστήματα τα οποία θα είναι πιο αποδοτικά και πιο ρεαλιστικά. Προτείνω σαν μελλοντική εργασία να συμπτυχθούν σε ένα υβριδικό αλγόριθμο, ο αλγόριθμος που έχω σχεδιάσει και υλοποιήσει για την εφαρμογή «Find your Friend» μαζί με τους αλγόριθμους ελέγχου περιμέτρου και κάλυψης περιοχής από την υπάρχουσα εργασία. Αυτό θα μας βοηθούσε στο να έχουμε στην αρχή τυχαίο αριθμό κόμβων από διαφορετικές ομάδες στο χώρο. Η μια από τις ομάδες αυτές να είναι υπεύθυνη για την περιμετρική κάλυψη του χώρου. Έτσι να ξεκινά πρώτα ο αλγόριθμος για την εύρεση των διαφορετικών ομάδων και μετά, η ομάδα που πρέπει να κάνει την περιμετρική κάλυψη να συνεχίζει με τον αλγόριθμο που έχει αναπτυχθεί για αυτό τον σκοπό. Οι άλλες ομάδες θα πρέπει είτε να μένουν στην θέση τους είτε, μπορούμε να τις εκμεταλλευτούμε με άλλο τρόπο, όπως για παράδειγμα να καλύψουμε την περιοχή μέσα στον χώρο που έχουμε. Να γίνει εισαγωγή και ενός αλγόριθμου κάλυψης περιοχής, έτσι ώστε οι ομάδες να διασκορπίζονται μέσα στο χώρο, στις κατάλληλες τοποθεσίες για να γίνεται καλύτερος έλεγχος ολόκληρης την περιοχή.

Και οι δυο αυτές εφαρμογές μπορούν να χρησιμοποιηθούν στην πράξη για διάφορες εφαρμογές. Σχετικά με τον αλγόριθμο της εύρεσης κοινών φίλων («Find your friends») μπορεί να χρησιμοποιηθεί σε στρατιωτικό περιβάλλον όπου μη επανδρωμένα οχήματα προσπαθούν να συγκλίνουν σε μια περιοχή έτσι ώστε να ομαδοποιηθούν, καθώς την ίδια στιγμή προσπαθούν να αποφύγουν τον εχθρό. Αυτό θα δώσει περισσότερηη ευελίξια στην κίνηση των οχημάτων και πλέον δεν θα χρειάζεται ο ανθρώπινος παράγοντας για την καθοδήγηση των οχημάτων. Μια άλλη χρησιμότητα είναι ότι μπορούμε να έχουμε

κατασκοπικά ρομπότ τα οποία να μπορούν να εισχωρούν σε αντίπαλες περιοχές και να περισυλλέγουν διάφορα δεδομένα. Όταν βρεθούν αντιμέτωπα με εχθρικές δυνάμεις, δηλαδή να αισθανθούν μέσα στην εμβέλεια τους οτιδήποτε εκτός από άλλα ρομπότ τις δικής τους ομάδας να αλλάζουν την κίνησή τους για να μην τα βρει ο αντίπαλος. Από την άλλη όταν βρουν τον αρχηγό της ομάδας να κινηθούν μαζί του έτσι ώστε να γίνει και η συλλογή των πληροφοριών και να μπορούν να επιστρέψουν πίσω όλα μαζί, φέροντας τις πολύτιμες τους πληροφορίες για επεξεργασία.

Όσον αφορά τον αλγόριθμο της ακολούθησης αρχηγού ( «Follow the leader») μπορεί να χρησιμοποιηθεί σε μια πιθανή χρησιμοποίηση των ασύρματων δικτύων αισθητήρων (wireless sensors networks) σε ένα Intelligent Transportation System όπου τα οχήματα σε ένα αυτοκινητόδρομο (highway) ακολουθούν το ένα το άλλο αυτόματα και δεν χάνουν τα οχήματα που βρίσκονται μπροστά τους. Αυτό, επίσης, μπορεί να χρησιμοποιηθεί στο να δείξουμε την σημαντικότητα του να διατηρούμε την σωστή απόσταση από το όχημα μπροστά μας ούτος ώστε να αποφεύγουμε τα τροχαία δυστυχήματα σε μεγάλο βαθμό.

Βασική προϋπόθεση για την χρήση των αλγορίθμων όμως είναι η βελτίωση τους και ο πειραματισμός τους σε πραγματικές συνθήκες. Δεν μπορούν διαφορετικά να βγουν ασφαλή συμπεράσματα για το πόσο καλές και χρήσιμες μπορεί να φανούν.

# Κεφάλαιο 6

## Συμπεράσματα



### 6.1 Γενική σύνοψη

Η ατομική διπλωματική αυτή εργασία καταπιάστηκε με θέματα γύρω από το χώρο των ασύρματων δικτύων αισθητήρων. Πιο συγκεκριμένα είδαμε στην αρχή μια γενική αναφορά για το τι είναι ένα ασύρματο δίκτυο αισθητήρα και από τι είναι ένας κόμβος αισθητήρα. Αναλύθηκαν τα χαρακτηριστικά των δικτύων αυτών, τα στρώματα στα Ασύρματα Δικτύα Αισθητήρων, προβλήματα στα ασύρματα δίκτυα αισθητήρων καθώς και τομείς που χρησιμοποιούνται σήμερα τα WSNs. Για τους αισθητήρες είδαμε σε τι ομάδες μπορούμε να τους κατατάξουμε, την βασική αρχιτεκτονική τους και μια περιγραφή του κάθε ενός βασικού στοιχείου τους καθώς και επίσης βασικές τους λειτουργίες.

Είδαμε τα ανοικτά ζητήματα που υπάρχουν στο χώρο αυτό και επικεντρωθήκαμε στην κινητικότητα στα ασύρματα δίκτυα αισθητήρων, με την οποία και ασχολείται η εργασία αυτή. Παρουσιάστηκαν οι τρόποι με τους οποίους μπορεί να παρουσιαστεί η κινητικότητα μέσα σε ένα ασύρματο δίκτυο αισθητήρων και ποιες δυνατότητες προσδίδουν αυτοί στο δίκτυο.

Μετέπειτα αναλύθηκαν τα δυο διαφορετικά υποπροβλήματα που υπήρχαν για το σκοπό της εργασίας αυτής. Για κάθε ένα από αυτά παρουσιάστηκαν οι λύσεις τους υλοποιημένες σε γραφικό περιβάλλον με τη βοήθεια της γλώσσας JAVA και αναλυτική περιγραφή του αλγόριθμου που χρησιμοποιήθηκε για να γίνουν. Έγινε ανάλυση των αποτελεσμάτων από δοκιμαστικές προσομοιώσεις που έγιναν και σχολιασμός των αποτελεσμάτων.

Τέλος αναφέρθηκε η μελλοντική δουλειά που μπορεί να γίνει σχετικά με τα θέματα που ασχολήθηκε η ατομική διπλωματική αυτή εργασία καθώς και τομείς που μπορεί να εφαρμοστούν οι συγκεκριμένοι αλγόριθμοι.

### 6.2 Συμπεράσματα

Από την μελέτη που έχει γίνει για τις ανάγκες αυτής της διπλωματικής εργασίας, μπορώ να πω ότι τα ασύρματα δίκτυα αισθητήρων είναι μια ενδιαφέρουσα και πολλά υποσχόμενη τεχνολογία. Βέβαια υπάρχουν ακόμη πολλά προβλήματα, ειδικά στα θέματα κατανάλωσης ενέργειας, που όμως με την συνεχή ανάπτυξη της τεχνολογίας τα πράγματα θα γίνονται όλο και καλύτερα. Ακόμη με την εισαγωγή της κινητικότητας μπορούμε να βοηθήσουμε πάρα πολύ την αποτελεσματικότητα και τις δυνατότητες των WSNs.

Με την κίνηση πλέον οι αισθητήρες μπορούν να κάνουν πολλά περισσότερα πράγματα και ακόμη μπορεί να τα εκτελέσουν και πολύ καλύτερα από το να είναι πάντα σταθεροί σε ένα σημείο. Έχουμε μια πιο σφαιρική και ολοκληρωμένη κατανόηση του γεγονότος που θέλουμε να μετρήσουμε με την κίνηση των κόμβων, αφού μπορούμε να κάνουμε πολύ περισσότερα πράγματα με την κίνηση.

Αναφορικά με την ομαδοποίηση των κόμβων σε μια περιοχή, χωρίς την κινητικότητα αυτό είναι σχεδόν αδύνατο έως ακατόρθωτο. Με την ομαδοποίηση των κόμβων

μπορούμε να πετύχουμε στη συνέχεια πολλά πράγματα. Αυτό γιατί τις περισσότερες φορές πρέπει να εγκαταστήσουμε ένα ασύρματο δίκτυο αισθητήρων σε χώρους όπου δεν έχει πρόσβαση άνθρωπος. Έτσι χρειάζεται για παράδειγμα να τους ρίξουμε από αεροπλάνο. Με ένα απλό αλγόριθμο ομαδοποίησης, όπως αυτό που αναπτύχθηκε στην εργασία αυτή θα μπορούμε να ομαδοποιήσουμε τους κόμβους, και μετά να τους αφήσουμε να συντονιστούν για να πραγματοποιήσουν τις μετρήσεις που θέλουμε.

Τα ασύρματα δίκτυα αισθητήρων μπορούν κάλλιστα στο άμεσο μέλλον να βοηθήσουν έμπρακτα τον άνθρωπο σε χιλιάδες πρακτικές εφαρμογές, προσδίδοντας νέα δεδομένα και νέες προ κλείσεις σε πάρα πολλούς τομείς. Με τη συνεχή έρευνα σε αυτό το πεδίο μπορούμε να προκαλέσουμε μια γενική πρόοδο, που βάση την να έχει αυτούς τους μικρούς κόμβους-αισθητήρες. Βασική προϋπόθεση φυσικά είναι να μπορούμε να εκμεταλλευτούμε τις ευκαιρίες που μας δίνει η τεχνολογία και να τις ενσωματώσουμε στα δίκτυα αυτά.

# Βιβλιογραφία


[1] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyr. "*Recent advances on multi-agent patrolling*". Lecture Notes in Computer Science, 3171:474–483, 2004

[2] A. Ephremides, J.E. Wieselthier and D. J. Baker, "*A Design concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling*", Proceeding of IEEE, Vol. 75, No. 1, pp. 56-73, 1987.

[3] A. Howard, M. J Mataric, and G. S. Sukhatme. "*Mobile sensor network deployment using potential fields:A distributed, scalable solution to the area coverage problem*." In 6th International Symposium on Distributed Autonomous Robotics Systems (DARS02), June 2002.

[4] C. Huang and Y. Tseng. "*The Coverage Problem in a Wireless Sensor Network*", In Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE, 29 Nov.- 3 Dec. 2004, 3182 - 3186 Vol.5

[5] D. J. Baker and A. Ephremides, "*The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm*", IEEE Transactions on Communications, Vol. 29, No. 11, pp. 1694-1701, November 1981.

[6] G. Amitabha. "*Estimating Coverage Holes and Enhancing Coverage in Mixed Sensor Networks*", In Local Computer Networks, 2004. 29th Annual IEEE International Conference on, pp 68 – 76, 16-18 Nov. 2004.

[7] G. Wang, G. Cao, and T. La Porta." *A bidding protocol for deploying mobile sensors*." In 11th IEEE International Conference on Network Protocol ICNP "03, pp. 315–324, Nov 2003.

[8] G. Wang, G. Cao, T. La Porta and W. Zhang. "*Sensor relocation in Mobile Sensor Network*", In IEEE INFOCOM 2004, June 2004.

[9] H. Chan, A. Perrig, "*ACE: An Emergent Algorithm for Highly Uniform Cluster Formation*". In 2004 European Workshop on Sensor Networks. pp. 154-171.

[10] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci "*A Survey on Sensor Networks.*", "IEEE Communications Magazine, vol.40, No.8, pp.102-114, August 2002.

[11] J. S. Shieh and T. W. Calvert. "*View and route planning for patrol and exploring robots*". Advanced Robotics, 6(4):399–430, 1992.

[12] K. Holger and A. Willig, "*Protocols and Architectures for Wireless Sensor Networks*", John Wiley and Son, 2007

[13] M. Chatterjee, S. K. Das, and D. Turgut, "*WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks*", Journal of Cluster Computing, Special issue on Mobile Ad hoc Networking, No. 5, 2002, pp. 193-204.

[14] N. Heo and P. K. Varshney." *An intelligent deployment and clustering algorithm for a distributed mobile sensor network*." In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, volume 5, pp. 4576–4581, Oct 2003.

[15] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus. "*An efficient heuristic approach for security against multiple adversaries*". In *AAMAS*, 2007.

[16] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. "*Security in multiagent systems by policy randomization*". In *AAMAS*, 2007.

[17] S. Basagni, "*Distributed Clustering for Ad Hoc Networks*", in Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks, pp. 310-315, June 1999.

[18] S. Ganeriwal, A. Kansal, and M. B.Srivastava." *Self aware actuation for fault repair in sensor networks*." In Proceedings of the IEEE International Conference on Robotics and Automation(ICRA), May 2004.

[19] T. Yan, T. He, and J. Stankovic. "*Differentiated surveillance for sensor networks*.", In Proceedings of the ACM SenSys 03, Nov 2003.

[20] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, "*Energy-efficient communication protocols for wireless microsensor networks*", in: Proceedings of the 33rd Hawaiian International Conference on Systems Science (HICSS-33), Maui, HI, USA, 4–7 January 2000.

[21] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless and C. Gill. "*Integrated coverage and connectivity configuration in wireless sensor networks.*" In Proceedings of the ACM SenSys ‟03, Nov 2003, pp 28–39.

[22] Y. Chevaleyre. "*Theoretical analysis of the multi-agent patrolling problem*". In Proceedings of Intelligent Agent Technology (IAT*)*, pp 302-308, Sept 2004.

[23] Y. Elmaliach, N. Agmon, and G. A. Kaminka. "*Multi-robot area patrol under frequency constraints*". Robotics and Automation, 2007 IEEE International Conference on, 10-14 April 2007, pp 385 – 390.

[24] Yiannis Marinakis, Magdalene Marinaki, Nikolaos Matsatsinis, "*A Hybrid Clustering Algorithm based on Honey Bees Mating Optimization and Greedy Randomized Adaptive Search Procedure* ". Lecture Notes in Computer Science, 2008, Volume 5313/2008, 138-152, DOI: 10.1007/978-3-540-92695-5_11

[25] Y. Ke Cheng, Prithviraj Dasguta and Yi Wang, "*Distrubuted Area Using Robot Flocks*". In proceeding of: Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, 01/2010; DOI: 10.1109/NABIC.2009.5393461

[26] Y. Xiaohua Wang, Jie Shen and Hongjun Tang,*"Novel Hybrid Document Clustering Algorithm Based on Ant Colony and Agglomerate."*. In Proceeding KAM '09 Proceedings of the 2009 Second International Symposium on Knowledge Acquisition and Modeling - Volume 03 Pages 65-68.

# Παράρτημα Α

**Κώδικας στην JAVA για δημιουργία τυχαίων θέσεων των κόμβων σε μια περιοχή**

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.StringWriter;
import java.util.ArrayList;
import java.util.StringTokenizer;


public class RandomizeSensors {

	//integer holding the cover range of the sensors
	public static int sensor_number=12,
cover_range=150,distanceX=1200,distanceY=800,area_size=10;

	//method that writes the coordinates of the sensors randomly
	public static void writeIntegersToFile(String fileName) throws IOException {
		//number of the sensors in the area
		int actual_number=0;
		int sensorx=0;
		int sensory=0;
		int close_sensors,count = 0;
		double temp1,temp2,distance;
		int[] sensors = new int[sensor_number*2];
		BufferedWriter writer = new BufferedWriter(new FileWriter(fileName));
		writer.flush();
			//add the sensors in random places inside the area
		for(int i=0; i<sensor_number ; i++){
			count=0;
```

```
        do{
                count++;
                close_sensors=0;
                sensorx=(int)(Math.random()*distanceX);
                sensory=(int)(Math.random()*distanceY);
                sensors[i*2]=sensorx;
                sensors[i*2+1]=sensory;
                for(int j=0;j<i;j++){
                        temp1=sensors[i*2]-sensors[j*2];
                        temp2=sensors[i*2+1]-sensors[j*2+1];
                        temp1=temp1*temp1;
                        temp2=temp2*temp2;
                        temp1=temp1+temp2;
                        distance=(int)Math.sqrt(temp1);
                        //make sure there are no collisions
                        if(distance<cover_range*2){
                                close_sensors++;
                                if(close_sensors==10){
                                        break;
                                }
                        }

                }
        }while(close_sensors==10 && count<1000);

        if(count<1000){
                //write the coordinates in the text file
                writer.write(sensors[i*2] + " " + sensors[i*2+1]);
                actual_number++;
                writer.newLine();
        }
}
//display the actual number of sensors added in the area
writer.close();
```

```
}

//method for reading the coordinates from a txt file
@SuppressWarnings("unchecked")
public static int[] getIntegersFromFile(String fileName) throws IOException {
	StringWriter writer = new StringWriter();
	BufferedReader reader = new BufferedReader(new FileReader(fileName));
	for (String line = reader.readLine(); line != null; line = reader.readLine()) {
		writer.write(line + " ");
	}
	StringTokenizer tokens = new StringTokenizer(writer.toString());
	@SuppressWarnings("rawtypes")
	ArrayList list = new ArrayList();
	while (tokens.hasMoreTokens()) {
		String str = tokens.nextToken();
		try {
			list.add(new Integer(str));
		} catch (NumberFormatException e) {
			System.out.println("Error '" + str
					+ "' is not an integer.");
		}
	}
	int[] array = new int[list.size()];
	for( int i = 0; i < array.length; i++){
		array[i] = ((Integer)list.get(i)).intValue();
	}

	return array;
}

public static void main(String[] args){
	try {
		writeIntegersToFile("sensor_coordinates.txt");
	} catch (IOException e) {
		// TODO Auto-generated catch block
```

```
                    e.printStackTrace();
                }
        }
}
```

**Κώδικας στην JAVA για δημιουργία την εφαρμογή «Find Your Friends»**

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Stroke;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.geom.Ellipse2D;
import java.awt.geom.RoundRectangle2D;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;

public class FindYourFriends extends Frame {

        private static final long serialVersionUID = 1L;
        // drawing variables
        static int Cover_Range = 120;
        static int Sense_Range = Cover_Range*3;
        static int Area_Length = 1400;
        static int Area_Windth = 1000;
        static int Speed = 20;

        static Stroke drawingStroke;
```

```
static RoundRectangle2D Leader1;
static RoundRectangle2D[] sensors;
static RoundRectangle2D Leader2;
static RoundRectangle2D Leader3;
static Ellipse2D[] RangeTeamA;
static Ellipse2D[] RangeTeamB;
static Ellipse2D[] RangeTeamC;
static Ellipse2D RangeL1;
static Ellipse2D RangeL2;
static Ellipse2D RangeL3;

// method for drawing
public void paint(Graphics g) {
	Graphics2D ga = (Graphics2D) g;
	ga.setStroke(drawingStroke);
	ga.setPaint(Color.red);
	for (int i = 0; i < sensors.length; i++) {
		ga.draw(sensors[i]);
	}
	ga.setPaint(Color.black);
	ga.draw(Leader1);
	ga.setPaint(Color.green);
	ga.draw(Leader2);
	ga.setPaint(Color.orange);
	ga.draw(Leader3);
	ga.setPaint(Color.gray);

	for (int i = 0; i < sensors.length / 3; i++) {
		ga.draw(RangeTeamA[i]);
	}
	ga.setPaint(Color.blue);
	for (int i = 0; i < sensors.length / 3; i++) {
		ga.draw(RangeTeamB[i]);
	}
	ga.setPaint(Color.red);
```

```
            for (int i = 0; i < sensors.length / 3; i++) {
                ga.draw(RangeTeamC[i]);
            }
            ga.setPaint(Color.gray);
            ga.draw(RangeL1);
            ga.setPaint(Color.blue);
            ga.draw(RangeL2);
            ga.setPaint(Color.red);
            ga.draw(RangeL3);

        }

        static public double Euclidean_distance(float x0, float y0, float x1,
                float y1) {

            return Math.sqrt(((x0 - x1) * (x0 - x1)) + ((y0 - y1) * (y0 - y1)));

        }

        @SuppressWarnings({ "unchecked" })
        static public void SensorsGrouping(SensorsCharacteristics[] table) {
            int temp;
            int counter = 0;
            for (int i = 0; i < table.length; i++) {

                for (int j = i; j < table.length; j++)

                    if (i != j
                            && ((table[i].nodeId != table[j].Driver) ||
(table[j].nodeId != table[i].Driver))
                            && table[i].teamId == table[j].teamId
                            && Euclidean_distance(table[i].coordX,
table[i].coordY,
                                    table[j].coordX,
table[j].coordY) <= Cover_Range) {
```

```
                        if (table[j].Driver == -1 && table[j].list.isEmpty()
                                && table[i].Driver == -1 &&
table[i].list.isEmpty()) {
                            table[j].Driver = table[i].nodeId;
                            table[i].list.add(table[j].nodeId);
                            // System.out.println("Press1");
                        } else if (table[i].Driver == -1 &&
table[i].list.isEmpty()
                                && table[j].Driver == -1
                                && !table[j].list.isEmpty()) {
                            table[j].Driver = table[i].nodeId;
                            table[i].list.add(table[j].nodeId);
                            while (!table[j].list.isEmpty()) {
                                temp = (Integer)
table[j].list.removeFirst();
                                table[i].list.add(temp);
                                table[temp].Driver = table[i].nodeId;
                            }
                            // System.out.println("Press2");
                        } else if (table[j].Driver == -1 &&
table[j].list.isEmpty()
                                && table[i].Driver == -1
                                && !table[i].list.isEmpty()) {
                            table[j].Driver = table[i].nodeId;
                            table[i].list.add(table[j].nodeId);
                            // System.out.println("Press3");
                        } else if (table[j].Driver == -1
                                && !table[j].list.isEmpty()
                                && table[i].Driver == -1
                                && !table[i].list.isEmpty()) {
                            table[j].Driver = table[i].nodeId;
                            table[i].list.add(table[j].nodeId);
                            while (!table[j].list.isEmpty()) {
                                temp = (Integer)
table[j].list.removeFirst();
                                table[i].list.add(temp);
```

```
                                        table[temp].Driver = table[i].nodeId;

                                    }
                                    // System.out.println("Press4");
                                } else if ((table[i].Driver == -1 && table[j].Driver !=
-1 && table[i].nodeId != table[j].Driver
                                /*
                                 * && table[i].list.isEmpty() && table[j].list
.isEmpty()
                                 */)) {
                                    int size;
                                    if (table[i].nodeId > table[j].Driver) {
                                        // System.out.println("Press  5P");
                                        table[i].Driver = table[j].Driver;

        table[table[j].Driver].list.add(table[i].nodeId);
                                        size = table[i].list.size();
                                        while (counter < size) {
                                            temp = (Integer)
table[i].list.pop();

        table[table[j].Driver].list.push(temp);

                                            table[temp].Driver =
table[j].Driver;
                                            counter++;
                                        }

                                    } else {
                                        // System.out.println("Press  5Q");
                                        int size1;
                                        counter = 0;
                                        table[table[j].Driver].Driver =
table[i].nodeId;
                                        table[i].list.push(table[j].Driver);
                                        size1 =
table[table[j].Driver].list.size();
```

```
                                        while (counter < size1) {
                                                temp = (Integer)
table[table[j].Driver].list
                                                                .pop();
                                                table[i].list.push(temp);

                                                table[temp].Driver =
table[i].nodeId;
                                                counter++;
                                        }

                                }

                        } else if ((table[j].Driver == -1 && table[i].Driver !=
-1 && table[j].nodeId != table[i].Driver
                        /*
                         * && table[j].list.isEmpty() && table[i].list
.isEmpty()
                         */)) {
                                int size1;
                                if (table[j].nodeId > table[i].Driver) {
                                        // System.out.println("Press  6P");
                                        table[j].Driver = table[i].Driver;

        table[table[i].Driver].list.add(table[j].nodeId);
                                        size1 = table[j].list.size();
                                        ;
                                        while (counter < size1) {
                                                temp = (Integer)
table[j].list.pop();

        table[table[i].Driver].list.push(temp);

                                                table[temp].Driver =
table[i].Driver;
                                                counter++;
                                        }
```

```
                                        } else {
                                                // System.out.println("Press  6Q");
                                                int size;
                                                counter = 0;
                                                table[table[i].Driver].Driver =
table[j].nodeId;
                                                table[j].list.push(table[i].Driver);
                                                size = table[table[i].Driver].list.size();
                                                while (counter < size) {
                                                        temp = (Integer)
table[table[i].Driver].list
                                                                        .pop();
                                                        table[j].list.push(temp);

                                                        table[temp].Driver =
table[j].nodeId;
                                                        counter++;
                                                }
                                        }

                                } else if ((table[i].Driver != -1 && table[j].Driver !=
-1
                                                && table[i].nodeId != table[j].Driver
                                                && table[i].Driver != table[j].Driver
                                                && table[i].list.isEmpty() &&
table[j].list
                                                .isEmpty())) {
                                        if (table[i].Driver > table[j].Driver) {
                                                int counter1 = 0;
                                                // System.out.println("Press  7Q");
                                                table[table[i].Driver].Driver =
table[j].Driver;
                                                table[table[j].Driver].list

        .add(table[table[i].Driver].nodeId);
                                                int size =
```

```
table[table[j].Driver].list.size();
                                                        while (counter1 < size) {
                                                                temp = (Integer)
table[table[i].Driver].list
                                                                        .removeFirst();

        table[table[j].Driver].list.add(temp);
                                                                table[temp].Driver =
table[j].Driver;
                                                                counter1++;
                                                        }
                                                } else {
                                                        // System.out.println("Press  7P");
                                                        table[table[j].Driver].Driver =
table[i].Driver;
                                                        table[table[i].Driver].list

        .add(table[table[j].Driver].nodeId);
                                                        int size =
table[table[j].Driver].list.size();
                                                        int counter1 = 0;
                                                        while (counter1 < size) {
                                                                temp = (Integer)
table[table[j].Driver].list
                                                                        .removeFirst();

        table[table[i].Driver].list.add(temp);
                                                                table[temp].Driver =
table[i].Driver;
                                                                counter1++;
                                                                // System.out.println("counter:
" + counter1);
                                                        }

                                                }
                                        }
                                }
```

```
		}
	}

	static boolean checkMoveCorrectness(float x, float y,
			SensorsCharacteristics[] table, int id, float prevcoordX,
			float prevcoordY) throws InterruptedException {

		float diffX = x - prevcoordX;
		float diffY = y - prevcoordY;
		boolean bflag = true;
		/*
		 * Ginetai elegxos kata poso i kinisi kapoiou komvou vgazi ekktos oriou
		 * tou frame, komvoys p pithanon na einai proskolimeni se auton
		 */
		for (int j = 0; j < table.length; j++)
			if (id != table[j].nodeId && id == table[j].Driver)
				if (!((table[j].coordX + diffX) < 950
						&& (table[j].coordX + diffX) > 60
						&& (table[j].coordY + diffY) < 750 &&
(table[j].coordY + diffY) > 60)) {
					bflag = false;
					break;

					// System.out.println("OUTOFFRAME: "+id+"
"+table[j].nodeId);

				}
		/*
		 * Elexei kata posos i kinisi tou komvou metakinite se thesi opou
		 * vrisketai kapoios allos komvos
		 */
		for (int i = 0; i < table.length; i++)

			if (x == table[i].coordX && y == table[i].coordY && bflag) {

				return false;
```

```
                }
            return true;

        }

        public static void main(String[] args) throws InterruptedException,
                FileNotFoundException {

            // array with int holding the sensors coordinates
            int counterForAverage;
            int numberOfResults = 1;
            long elapsedTime, temporary = 0;
            String strFilePath =
"C://Users//kpetro02//Desktop//workspaceUML//ThreeNumberOfTeams//results.txt";
            FileOutputStream fos = new FileOutputStream(strFilePath);
            PrintStream dos = new PrintStream(fos);

            while (numberOfResults <= 4) {
                counterForAverage = 1;
                while (counterForAverage <= 5) {
                    long start = System.currentTimeMillis();

                    boolean finish = false;
                    int counter = 0;
                    int[] senso = null;
                    int iterate = 1;
                    int count = 0, countTeamA = 0, countTeamB = 0,
countTeamC = 0;
                    ;
                    try {
                        // Get the coordinates from the file
                        // sensor_coordinates.txt
                        senso = RandomizeSensors

        .getIntegersFromFile("sensor_coordinates.txt");
                    } catch (IOException e) {
```

```
                e.printStackTrace();
            }

            /*
             * Diaxwrismos se dyo omadas me ison arithmo komvwn.
             */
            SensorsCharacteristics[] SensoTable = new
SensorsCharacteristics[senso.length / 2];
            for (int i = 0; i < senso.length; i++) {
                senso[i] = senso[i] - (senso[i] % 10) + 50;
            }
            SensoTable[0] = new SensorsCharacteristics(true, 1,
senso[0],
                    senso[1], 0);
            SensoTable[1] = new SensorsCharacteristics(true, 2,
senso[2],
                    senso[3], 1);
            SensoTable[2] = new SensorsCharacteristics(true, 3,
senso[4],
                    senso[5], 2);
            /**
             * For the teams
             */
            counter = 3;

            for (int i = 6; i < senso.length; i += 2) {
                if (iterate == 1)
                    SensoTable[counter] = new
SensorsCharacteristics(false,
                            1, senso[i], senso[i + 1],
counter);
                else if (iterate == 2) {
                    SensoTable[counter] = new
SensorsCharacteristics(false,
                            2, senso[i], senso[i + 1],
counter);
```

```
                } else if (iterate == 3) {
                        SensoTable[counter] = new
SensorsCharacteristics(false,
                                        3, senso[i], senso[i + 1],
counter);
                        iterate = 0;
                }
                counter++;
                iterate++;
        }

        // //////////////////////////////////////
        // create shapes
        drawingStroke = new BasicStroke(2);

        Leader1 = new RoundRectangle2D.Float();
        Leader2 = new RoundRectangle2D.Float();
        Leader3 = new RoundRectangle2D.Float();
        RangeL1 = new Ellipse2D.Float();
        RangeL2 = new Ellipse2D.Float();
        RangeL3 = new Ellipse2D.Float();

        RangeL1.setFrame(SensoTable[0].coordX - 55,
                        SensoTable[0].coordY - 55, 125, 125);
        RangeL2.setFrame(SensoTable[1].coordX - 55,
                        SensoTable[1].coordY - 55, 125, 125);
        RangeL3.setFrame(SensoTable[2].coordX - 55,
                        SensoTable[2].coordY - 55, 125, 125);

        Leader1.setRoundRect(SensoTable[0].coordX,
                        SensoTable[0].coordY, 10, 10, 0, 0);
        Leader2.setRoundRect(SensoTable[1].coordX,
                        SensoTable[1].coordY, 10, 10, 0, 0);
        Leader3.setRoundRect(SensoTable[2].coordX,
                        SensoTable[2].coordY, 10, 10, 0, 0);
```

```
                        RangeTeamA = new Ellipse2D.Float[(SensoTable.length -
2) / 2];
                        RangeTeamB = new Ellipse2D.Float[(SensoTable.length -
2) / 2];
                        RangeTeamC = new Ellipse2D.Float[(SensoTable.length -
2) / 2];

                        /**
                         * For teams
                         */
                        sensors = new RoundRectangle2D[SensoTable.length - 3];

                        count = 0;
                        countTeamA = 0;
                        countTeamB = 0;
                        countTeamC = 0;

                        for (int i = 3; i < SensoTable.length; i++) {
                                sensors[count] = new RoundRectangle2D.Float();
                                sensors[count].setRoundRect(SensoTable[i].coordX,
                                                SensoTable[i].coordY, 10, 10, 0, 0);
                                if (SensoTable[i].teamId == 1) {
                                        RangeTeamA[countTeamA] = new
Ellipse2D.Float();
                                        RangeTeamA[countTeamA].setFrame(
                                                        SensoTable[i].coordX - 55,
                                                        SensoTable[i].coordY - 55,
125, 125);
                                        countTeamA++;
                                } else if (SensoTable[i].teamId == 2) {
                                        RangeTeamB[countTeamB] = new
Ellipse2D.Float();
                                        RangeTeamB[countTeamB].setFrame(
                                                        SensoTable[i].coordX - 55,
                                                        SensoTable[i].coordY - 55,
125, 125);
                                        countTeamB++;
```

```
                } else if (SensoTable[i].teamId == 3) {
                        RangeTeamC[countTeamC] = new
Ellipse2D.Float();
                        RangeTeamC[countTeamC].setFrame(
                                SensoTable[i].coordX - 55,
                                SensoTable[i].coordY - 55,
125, 125);
                        countTeamC++;
                }

                count++;
        }

        Frame frame = new FindYourFriends();
        frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent we) {
                        System.exit(0);
                }
        });
        frame.setSize(Area_Length, Area_Windth);
        frame.setVisible(true);
        try {
                Thread.sleep(500);

        } catch (InterruptedException e) {
                e.printStackTrace();
        }// System.exit(0);
        do {
                try {
                        Thread.sleep(200);

                } catch (InterruptedException e) {
                        e.printStackTrace();
                }

                /*
```

```
                                        * Ginetai elexos kata posos oi komvoi mporoun na
                                        * omadopoihthoun
                                        */
                                       SensorsGrouping(SensoTable);

                                       /*
                                        * Elexos tis kinisis kathe komvou o opoios den
einai
                                        * proskolimenos se kapoio allon
                                        */
                                       for (int i = 0; i < SensoTable.length; i++) {
                                              if (SensoTable[i].Driver == -1)
                                                     SensoTable[i]

        .checkMove(frame.getSize(), SensoTable);
                                       }

                                       // ////////////////////////////////////
                                       // create shapes
                                       drawingStroke = new BasicStroke(2);

                                       Leader1 = new RoundRectangle2D.Float();
                                       Leader2 = new RoundRectangle2D.Float();
                                       Leader3 = new RoundRectangle2D.Float();
                                       RangeL1 = new Ellipse2D.Float();
                                       RangeL2 = new Ellipse2D.Float();
                                       RangeL3 = new Ellipse2D.Float();

                                       RangeL1.setFrame(SensoTable[0].coordX - 55,
                                                     SensoTable[0].coordY - 55, 125,
125);
                                       RangeL2.setFrame(SensoTable[1].coordX - 55,
                                                     SensoTable[1].coordY - 55, 125,
125);
                                       RangeL3.setFrame(SensoTable[2].coordX - 55,
                                                     SensoTable[2].coordY - 55, 125,
```

```
125);

                                        Leader1.setRoundRect(SensoTable[0].coordX,
                                                SensoTable[0].coordY, 10, 10, 0, 0);
                                        Leader2.setRoundRect(SensoTable[1].coordX,
                                                SensoTable[1].coordY, 10, 10, 0, 0);
                                        Leader3.setRoundRect(SensoTable[2].coordX,
                                                SensoTable[2].coordY, 10, 10, 0, 0);

                                        RangeTeamA = new
Ellipse2D.Float[(SensoTable.length - 2) / 2];
                                        RangeTeamB = new
Ellipse2D.Float[(SensoTable.length - 2) / 2];
                                        RangeTeamC = new
Ellipse2D.Float[(SensoTable.length - 2) / 2];

                                        /**
                                         * For teams
                                         */
                                        sensors = new
RoundRectangle2D[SensoTable.length - 3];

                                        count = 0;
                                        countTeamA = 0;
                                        countTeamB = 0;
                                        countTeamC = 0;

                                        for (int i = 3; i < SensoTable.length; i++) {
                                                sensors[count] = new
RoundRectangle2D.Float();

        sensors[count].setRoundRect(SensoTable[i].coordX,
                                                        SensoTable[i].coordY, 10, 10,
0, 0);
                                                if (SensoTable[i].teamId == 1) {
                                                        RangeTeamA[countTeamA] = new
Ellipse2D.Float();
```

```
        RangeTeamA[countTeamA].setFrame(
                                                        SensoTable[i].coordX
- 55,
                                                        SensoTable[i].coordY
- 55, 125, 125);
                                                countTeamA++;
                                            } else if (SensoTable[i].teamId == 2) {
                                                RangeTeamB[countTeamB] = new
Ellipse2D.Float();

        RangeTeamB[countTeamB].setFrame(
                                                        SensoTable[i].coordX
- 55,
                                                        SensoTable[i].coordY
- 55, 125, 125);
                                                countTeamB++;
                                            } else if (SensoTable[i].teamId == 3) {
                                                RangeTeamC[countTeamC] = new
Ellipse2D.Float();

        RangeTeamC[countTeamC].setFrame(
                                                        SensoTable[i].coordX
- 55,
                                                        SensoTable[i].coordY
- 55, 125, 125);
                                                countTeamC++;
                                            }

                                            count++;
                                        }

                                        frame.repaint();

                                        /*
                                         * Molis ginei i omadopoiisi energopoieitai to flag
gia
                                         * termatismo
```

```
                                        */
                                       /**
                                        * for teams
                                        */
                                       for (int i = 3; i < SensoTable.length; i++) {
                                              finish = false;
                                              if (SensoTable[i].Driver == -1) {
                                                     finish = true;
                                                     break;
                                              }
                                       }
                                } while (finish);
                                elapsedTime = System.currentTimeMillis() - start;
                                counterForAverage++;
                                System.out.println(counterForAverage);


                                temporary += (elapsedTime / 1000f);
                         }
                         dos.println("\t" + Speed + "\t"
                                       + temporary / counterForAverage);
                         Speed+=10;

                         temporary = 0;
                         System.out.println("Loop " + numberOfResults);
                         numberOfResults++;
                  }
                  System.out.println("FINISH");

            }
}
```

**Κώδικας στην JAVA για δημιουργία την εφαρμογή «Follow the Leader»**

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Stroke;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.geom.Ellipse2D;
import java.awt.geom.RoundRectangle2D;
import java.io.IOException;

public class FollowLeader extends Frame {

	static int Cover_Range = 150;
	private static final long serialVersionUID = 1L;
	// drawing variables
	static Stroke drawingStroke;
	static RoundRectangle2D robot;
	static RoundRectangle2D[] sensors;
	static Ellipse2D[] Range;

	// static RoundRectangle2D target;

	// method for drawing
	public void paint(Graphics g) {
		Graphics2D ga = (Graphics2D) g;
		ga.setStroke(drawingStroke);
		ga.setPaint(Color.red);
		for (int i = 0; i < sensors.length; i++) {
			ga.draw(sensors[i]);
		}
```

```
        ga.setPaint(Color.black);
        ga.draw(robot);
        ga.setPaint(Color.blue);

        for (int i = 0; i < sensors.length + 1; i++) {
            ga.draw(Range[i]);
        }

}

public static boolean move_back(Sensor[] table, float coordX, float coordY) {

        for (int i = 1; i < table.length; i++)

            if (coordX == table[i].coordX && coordY == table[i].coordY){
                System.out.println("false");
                return false;

            }
        return true;

}

public static void bubbleSort(double[] arr, int[] array) {
        boolean swapped = true;
        int j = 0;
        double tmp;
        int temp;
        while (swapped) {
            swapped = false;
            j++;
            for (int i = 0; i < arr.length - j; i++) {
                if (arr[i] > arr[i + 1]) {
                    tmp = arr[i];
                    temp = array[i];
```

```
                                arr[i] = arr[i + 1];
                                array[i] = array[i + 1];
                                arr[i + 1] = tmp;
                                array[i + 1] = temp;
                                swapped = true;
                        }
                }
        }
}

static public boolean checkIfInLine(float[] getInLine) {

        float temp = 0;

        for (int i = 0; i < getInLine.length; i++) {
                if (temp < getInLine[i])
                        return false;
        }

        return true;
}

static public double Euclidean_distance(float x0, float y0, float x1,
                float y1) {

        return Math.sqrt(((x0 - x1) * (x0 - x1)) + ((y0 - y1) * (y0 - y1)));

}

/*
 * static boolean checkMove(float x, float y, float[] table) {
 *
 * for (int i = 0; i < table.length; i += 2)
 *
 * if (x == table[i] && y == table[i + 1]) { // System.out.println(table[i]
```

```
 * + " " + table[i + 1]); return false; } return true;
 *
 * }
 */

public void sub_move(int coordX, int coordY, int targetX, int targetY) {

}

public static void main(String[] args) throws IOException,
                InterruptedException {

        // array with int holding the sensors coordinates
        int[] senso = null;
        int iter = 0;
        int positionX = 0;
        boolean flag = true;
        try {
                // Get the coordinates from the file sensor_coordinates.txt
                senso = RandomizeSensors
                                .getIntegersFromFile("sensor_coordinates.txt");
        } catch (IOException e) {
                e.printStackTrace();
        }
        int count = 1;
        Sensor[] tableNodes = new Sensor[senso.length / 2];
        double[] temp = new double[tableNodes.length - 1];
        int[] array = new int[tableNodes.length - 1];
        for (int i = 0; i < array.length; i++) {
                array[i] = count;
                count++;
        }
        for (int i = 0; i < senso.length; i += 2) {
                tableNodes[iter++] = new Sensor(senso[i], senso[i + 1]);
```

```
}
count = 1;
for (int i = 0; i < temp.length; i++) {

        temp[i] = Euclidean_distance(tableNodes[0].coordX,
                        tableNodes[0].coordY, tableNodes[count].coordX,
                        tableNodes[count].coordY);
        count++;
}

bubbleSort(temp, array);

for (int i = 0; i < array.length; i++) {
        positionX += 30;
        tableNodes[array[i]].targetX = positionX;

}

// Gia na xerw pios einai o leader
tableNodes[0].leader = true;
tableNodes[0].targetX = 0;
tableNodes[0].targetY = 240;
// ////////////////////////////////////
// create shapes
drawingStroke = new BasicStroke(2);
// rectangle for each sensor and the robot
robot = new RoundRectangle2D.Float();
robot.setRoundRect(tableNodes[0].coordX + 50,
                tableNodes[0].coordY + 50, 10, 10, 0, 0);

sensors = new RoundRectangle2D[tableNodes.length - 1];
Range = new Ellipse2D.Float[tableNodes.length];

for (int i = 1; i < tableNodes.length; i++) {
        sensors[i - 1] = new RoundRectangle2D.Float();
```

```
        sensors[i - 1].setRoundRect(tableNodes[i].coordX + 50,
                        tableNodes[i].coordY + 50, 10, 10, 0, 0);
}
for (int i = 0; i < tableNodes.length; i++) {
        Range[i] = new Ellipse2D.Float();
        Range[i].setFrame(tableNodes[i].coordX - 7,
                        tableNodes[i].coordY - 7, 125, 125);
}
Frame frame = new FollowLeader();
frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent we) {
                System.exit(0);
        }
});
frame.setSize(1024, 860);
frame.setVisible(true);
try {
        Thread.sleep(1000);
} catch (InterruptedException e1) {
        e1.printStackTrace();
}

boolean bflag = false;
do {
        try {
                Thread.sleep(500);
        } catch (InterruptedException e) {
                e.printStackTrace();
        }

        for (int i = 1; i < tableNodes.length; i++) {
                if (tableNodes[i].inLine == false) {
                        bflag = true;
                        break;
                }
```

```
}
if (bflag) {
	for (int i = 1; i < tableNodes.length; i++) {
		if (!tableNodes[i].inLine)
			tableNodes[i].Movement(tableNodes);

	}

} else {
	int counter = 0;
	positionX = 0;
	if (flag)// topothetisis akrivws stin thesis ts
		for (int i = 1; i < tableNodes.length; i++) {
			positionX += 30;
			tableNodes[i].coordX = positionX;
			flag = false;
		}
	float[] tableY = new float[tableNodes.length];
	float[] tableX = new float[tableNodes.length];

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

		tableX[i] = tableNodes[i].coordX;
		tableY[i] = tableNodes[i].coordY;
		// System.out.println(tableNodes[i].coordX + " "
		// + tableNodes[i].coordY);
	}

	if (tableNodes[0].targetX > tableNodes[0].coordX) {
		// System.out.println("Press1");
		if (move_back(tableNodes, tableNodes[0].coordX +
30,
				tableNodes[0].coordY)) {
			tableNodes[0].coordX += 30;
			counter = 0;
```

```
                                        for (int i = 1; i < tableNodes.length; i++) {
                                                tableNodes[i].coordX =
tableX[counter];
                                                tableNodes[i].coordY =
tableY[counter];
                                                counter++;
                                        }
                                } else {
                                        tableNodes[0].setRandomWaypoint();
                                }

                        }

                        if (tableNodes[0].targetX < tableNodes[0].coordX) {
                                // System.out.println("Press2");
                                if (move_back(tableNodes, tableNodes[0].coordX -
30,
                                                tableNodes[0].coordY)) {
                                        tableNodes[0].coordX -= 30;
                                        counter = 0;
                                        for (int i = 1; i < tableNodes.length; i++) {
                                                tableNodes[i].coordX =
tableX[counter];
                                                tableNodes[i].coordY =
tableY[counter];
                                                counter++;
                                        }
                                } else {
                                        tableNodes[0].setRandomWaypoint();
                                }
                        }
                        if (tableNodes[0].targetX == tableNodes[0].coordX
                                        && tableNodes[0].targetY >
tableNodes[0].coordY) {
                                if (move_back(tableNodes, tableNodes[0].coordX,
                                                tableNodes[0].coordY + 30)) {
```

```
                                        // System.out.println("Press3");
                                        tableNodes[0].coordY += 30;
                                        counter = 0;
                                        for (int i = 1; i < tableNodes.length; i++) {
                                                tableNodes[i].coordX =
tableX[counter];
                                                tableNodes[i].coordY =
tableY[counter];
                                                counter++;
                                        }
                                } else {
                                        tableNodes[0].setRandomWaypoint();
                                }
                        }
                        if (tableNodes[0].targetX == tableNodes[0].coordX
                                        && tableNodes[0].targetY <
tableNodes[0].coordY) {
                                // System.out.println("Press4");
                                if (move_back(tableNodes, tableNodes[0].coordX,
                                                tableNodes[0].coordY - 30)) {
                                        tableNodes[0].coordY -= 30;
                                        counter = 0;
                                        for (int i = 1; i < tableNodes.length; i++) {
                                                tableNodes[i].coordX =
tableX[counter];
                                                tableNodes[i].coordY =
tableY[counter];
                                                counter++;
                                        }
                                } else {
                                        tableNodes[0].setRandomWaypoint();
                                }

                        }
                        if (tableNodes[0].targetX == tableNodes[0].coordX
                                        && tableNodes[0].targetY ==
```

```
tableNodes[0].coordY) {
                                        // System.out.println("Press5");
                                        tableNodes[0].setRandomWaypoint();
                                }
                        }
                        bflag = false;
                        // ////////////////////////////////////
                        // create shapes
                        drawingStroke = new BasicStroke(2);
                        // rectangle for each sensor and the robot
                        robot = new RoundRectangle2D.Float();
                        robot.setRoundRect(tableNodes[0].coordX + 50,
                                        tableNodes[0].coordY + 50, 10, 10, 0, 0);

                        sensors = new RoundRectangle2D[tableNodes.length - 1];
                        Range = new Ellipse2D.Float[tableNodes.length];

                        for (int i = 1; i < tableNodes.length; i++) {
                                sensors[i - 1] = new RoundRectangle2D.Float();
                                sensors[i - 1].setRoundRect(tableNodes[i].coordX + 50,
                                                tableNodes[i].coordY + 50, 10, 10, 0, 0);
                        }
                        for (int i = 0; i < tableNodes.length; i++) {
                                Range[i] = new Ellipse2D.Float();
                                Range[i].setFrame(tableNodes[i].coordX - 7,
                                                tableNodes[i].coordY - 7, 125, 125);
                        }

                        frame.repaint();

                } while (true);
        }
}
```