Ατομική Διπλωματική Εργασία

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ LEGO MINDSTORM NXT ΓΙΑ ΥΠΟΣΤΗΡΙΞΗ ΚΙΝΗΤΙΚΟΤΗΤΑΣ ΣΕ ΑΣΥΡΜΑΤΑ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ (WSN)

Γεώργιος Χαραλάμπους

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Προγραμματισμός LEGO Mindstorm NXT για υποστήριξη κινητικότητας σε ασύρματα δίκτυα αισθητήρων (WSN)

Γεώργιος Χαραλάμπους

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 2009

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 δούμε διάφορα προβλήματα που έχουν τα ασύρματα δίκτυα αισθητήρων στον τομέα του ελέγχου τοπολογίας και στη συνέχεια πώς μπορούν να επιλυθούν με τη χρήση ενός ρομπότ. Το ρομπότ αυτό θα εξυπηρετεί το δίκτυο σε θέματα κινητικότητας και θα είναι προγραμματισμένο με τους ανάλογους αλγορίθμους. Επομένως θα συγκρίνουμε διάφορους αλγόριθμους σε όλους τους τομείς του ελέγχου τοπολογίας όπως εντοπισμός, έλεγχος κάλυψης, κινητικότητα και εντοπισμός στόχου.

Περιεχόμενα

Κεφάλαιο 1

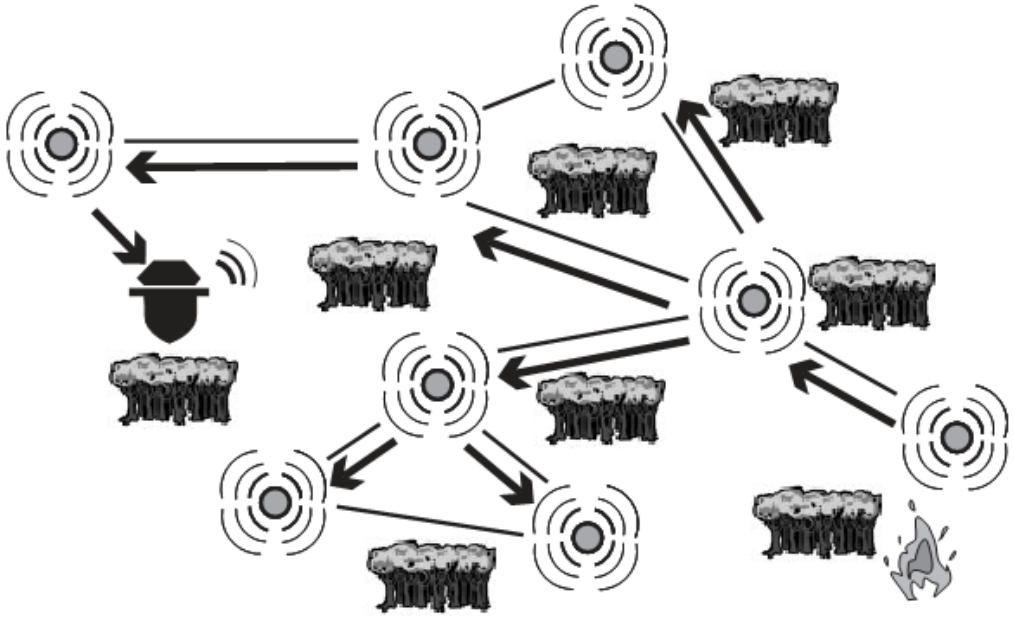
Εισαγωγή

1.1. Ασύρματα Δίκτυα αισθητήρων

Mobile ad hoc networks (σε συντομία MANETs) είναι δίκτυα από ασύρματες μονάδες οι οποίες επικοινωνούν μεταξύ τους μέσο ράδιο-δέκτες. Τις πλείστες φορές ένα μήνυμα για να πάει στο προορισμό του περνά από πολλές μονάδες (multihop paths). Χρησιμοποιούνται σε περιπτώσεις όπου δεν είναι εφικτή ή οικονομικά ασύμφορη η χρήση και υλοποίηση ενσύρματου δικτύου. Για παράδειγμα χρησιμοποιείται για την επικοινωνία σε διάφορες εκδηλώσεις όπως συνέδρια, εκθέσεις, συναυλίες και για τον συντονισμό διάσωσης μετά από μια καταστροφή.

Τα ασύρματα δίκτυα αισθητήρων (wireless sensor networks, WSNs) [5] είναι μια ειδική υποκατηγορία των MANETs όπου οι ασύρματες μονάδες είναι ασύρματοι αισθητήρες. Αυτοί οι αισθητήρες συλλέγουν τα δεδομένα που τους ενδιαφέρουν όπως θερμοκρασία, ατμοσφαιρική πίεση και πολλά άλλα, και στη συνέχεια τα αποστέλλουν μέσο του δικτύου είτε σε μια κεντρική μονάδα που θα τα επεξεργαστεί, είτε σε άλλους αισθητήρες μέσα στο δίκτυο. Τα WSN είναι χρήσιμα σε πάρα πολλούς τομείς. Για παράδειγμα στον έλεγχο καταστάσεων σε απομακρυσμένες ή απρόσιτες περιοχές, παρακολούθηση της κίνησης

ζώων, πρόγνωση καιρού. Προβλέπεται ότι στο σύντομο μέλλον θα αποτελούν ένα σημαντικό μέρος στη ζωή του ανθρώπου, γι' αυτό γίνονται πολλές μελέτες τον τελευταίο καιρό για καλυτέρευση των WSNs.



Σχήμα 1.1: Ασύρματο Δίκτυο Αισθητήρων για τον εντοπισμό φωτιάς [26]

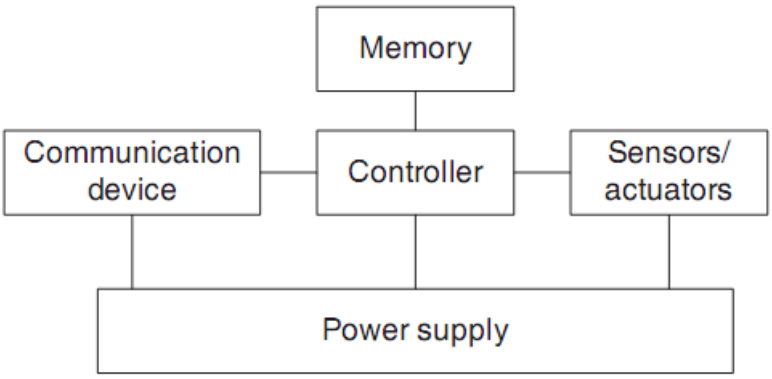
Τα κύρια χαρακτηριστικά των WSN είναι τα εξής:

- Περιορισμένη αποθηκευμένη ενέργεια
- Αντοχή σε αντίξοες καιρικές συνθήκες
- Ικανότητα να αντιμετωπίζει την διακοπή λειτουργίας κόμβων
- Κινητικότητα των κόμβων
- Δυναμική τοπολογία του δικτύου
- Διακοπές στην επικοινωνία
- Ετερογένεια στους κόμβους
- Επέκταση του δικτύου σε μεγάλη κλίμακα
- Αυτόνομη λειτουργία

Υπάρχει πληθώρα εφαρμογών για τα ασύρματα δίκτυα αισθητήρων. Τις πλείστες φορές αφορούν κάποιο είδος παρακολούθησης και έλεγχο. Συγκεκριμένες εφαρμογές αποτελούν την παρακολούθηση ενός κτιρίου, εντοπισμό αντικειμένου, εντοπισμό φωτιάς και παρακολούθηση της κυκλοφοριακής κίνησης. Παρακολούθηση περιοχής είναι η πιο συνηθισμένη εφαρμογή για ένα ασύρματο δίκτυο αισθητήρων. Σε αυτή την εφαρμογή σκορπίζουμε τους κόμβους αισθητήρων στην περιοχή που θέλουμε να παρακολουθούμε και παίρνουμε μετρήσεις για το φαινόμενο που μας ενδιαφέρει. Για παράδειγμα σκορπίζουμε τους κόμβους αισθητήρων στο πεδίο μάχης για των εντοπισμό εχθρικών οχημάτων. Όταν εντοπιστεί κάποιο συμβάν από τους αισθητήρες πρέπει να μεταφερθεί το μήνυμα στη βάση για να λάβει τα απαραίτητα μέτρα. Για παράδειγμα να σημάνει συναγερμός.

1.2. Κόμβοι Αισθητήρων

Οι αισθητήρες (κόμβοι) έχουν τη δυνατότητα μερικής επεξεργασίας, μάζεμα πληροφοριών και επικοινωνίας με τους υπόλοιπους κόμβους μέσα στο δίκτυο. Η εξέλιξη των κόμβων αισθητήρων ξεκίνησε από το 1998 με το πρόγραμμα Smartdust. Είχε σκοπό να δημιουργήσει αυτόνομους αισθητήρες που επικοινωνούσαν μέσα σε ένα κυβικό χιλιοστόμετρο. Παρόλο που το πρόγραμμα τερματίστηκε, έδωσε το έναυσμα για μελέτη στο συγκεκριμένο τομέα. Οι κόμβοι είναι χαρακτηριστικά μικροί σε μέγεθος και έχουν αρκετά περιορισμένη υπολογιστική ισχύ, αποθηκευτικό χώρο και ενέργεια. Η πιο συνηθισμένη αρχιτεκτονική φαίνεται στο σχήμα 1.2.



Σχήμα 1.2: Αρχιτεκτονική ενός κόμβου αισθητήρα [17]

Τα κύρια μέρη ενός κόμβου, όπως φαίνονται στην εικόνα, είναι ο μικροεπεξεργαστής (microcontroller), ο πομποδέκτης (transceiver), η εξωτερική μνήμη (external memory), η πηγή ενέργειας (power source) και ένας ή περισσότεροι αισθητήρες (sensors) [17].

- Ο μικροεπεξεργαστής εκτελεί εργασίες, επεξεργάζεται δεδομένα και ελέγχει τις λειτουργίες όλων των κομματιών που είναι ενωμένα πάνω στο κόμβο. Μικροεπεξεργαστές είναι η καλύτερη επιλογή για ενσωματωμένα συστήματα αφού μπορούν να προγραμματιστούν, να ενωθούν με άλλες συσκευές και να τεθούν σε κατάσταση "ύπνου" για να καταναλώνουν ελάχιστη ενέργεια.

- Υπάρχουν διάφοροι τρόποι επικοινωνίας μέσα σε ένα WSN. Ραδιοσυχνότητα (RF), Infrared και Laser. Το Laser απαιτεί λιγότερη ενέργεια αλλά απαιτεί οι κόμβοι να βρίσκονται σε ευθεία γραμμή και είναι αρκετά ευαίσθητο. Το Infrared όπως και το Laser, δεν χρειάζεται αντένα αλλά έχει περιορισμένη ραδιοφωνική αναμετάδοση. Η Ραδιοσυχνότητα (RF) είναι η πιο κατάλληλη για WSN. Χρησιμοποιεί συχνότητες από 433 MHz μέχρι 2.4 GHz. Σύγχρονοι πομποδέκτες έχουν 4 διαφορετικούς τρόπους λειτουργίας. Αποστολή, Λήψη, Ανενεργό (idle), Εκτός λειτουργίας (sleep). Ο λόγος είναι για να εξοικονομούν όσο το δυνατό περισσότερη ενέργεια.

- Η μνήμη σε ένα κόμβο χρησιμοποιείται κυρίως για δύο λόγους. Πρώτο για την αποθήκευση δεδομένων της εφαρμογής. Δεύτερο για προγραμματιστική μνήμη που χρησιμοποιείται για να προγραμματίσουμε τον κόμβο. Η πιο συνηθισμένη είναι μνήμη ενσωματωμένη στο μικροεπεξεργαστή ή μνήμη flash. Επίσης χρησιμοποιείται για να κρατά αναγνωριστικές πληροφορίες του κόμβου.

- Ένας κόμβος καταναλώνει ενέργεια για τους αισθητήρες, για την επικοινωνία και για την επεξεργασία δεδομένων. Η ενέργεια αποθηκεύεται σε μπαταρίες ή πυκνωτές. Πιο πρόσφατοι κόμβοι σχεδιάζονται για να μπορούν να ανανεώνουν την ενέργεια τους από άλλες πηγές όπως τον ήλιο.

Αισθητήρες είναι συσκευές οι οποίες παράγουν μετρήσιμα αποτελέσματα σε αλλαγές μιας φυσικής κατάστασης όπως τη θερμοκρασία και την πίεση. Γίνεται μία συνεχής μετατροπή του αναλογικού σήματος σε ψηφιακό για να μπορεί να αποσταλεί και να επεξεργαστεί.

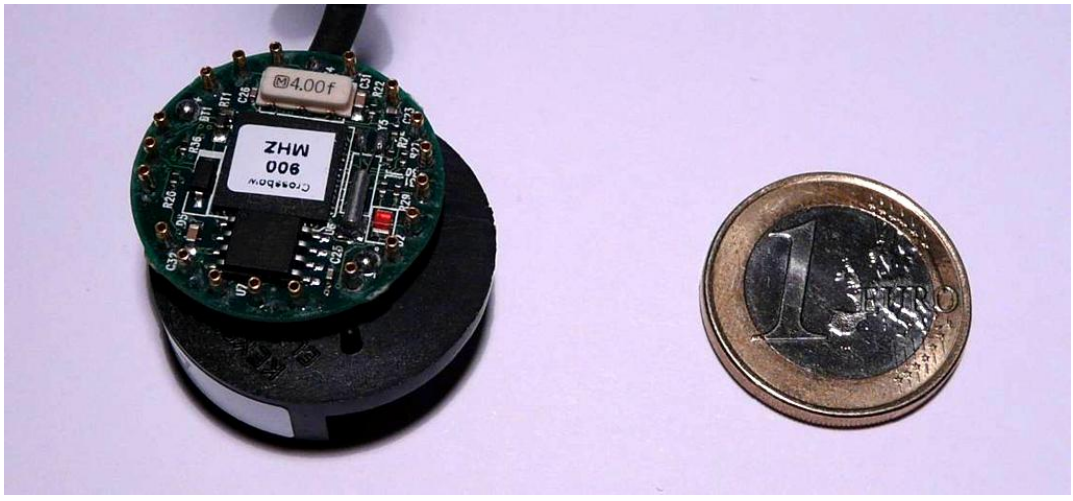
Ορισμένα από τα χαρακτηριστικά των αισθητήρων είναι τα εξής:

- Μικρά σε μέγεθος.
- Κατανάλωση ελάχιστης ενέργειας.
- Λειτουργία σε μεγάλη πυκνότητα.
- Αυτόνομα.
- Προσαρμογή στο περιβάλλον.

Οι αισθητήρες χωρίζονται σε τρεις κατηγορίες:

- Passive, Omni Directional Sensors
- Passive, narrow-beam sensors
- Active Sensors

Στην μελέτη μας θα χρησιμοποιήσουμε τους πρώτους όπου ο κάθε αισθητήρας ελέγχει μία καθορισμένη περιοχή ανάλογη της εμβέλειας του.



Σχήμα 1.3: Παράδειγμα ασύρματου κόμβου αισθητήρα και σύγκριση μεγέθους

1.3. Προβλήματα στα ασύρματα δίκτυα αισθητήρων

Τα WSNs έχουν πάρα πολλά προβλήματα [15]. 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 ‘’ύπνο’’ 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.

- Τέλος, ένα πρόβλημα που θα ασχοληθούμε μαζί του και στη συνέχεια, είναι η πυκνότητα των αισθητήρων. Μερικά WSNs για να λειτουργήσουν σωστά πρέπει να έχουν μια ελάχιστη πυκνότητα από αισθητήρες. Σε περίπτωση μεγάλης πυκνότητας, το δίκτυο υπόκειται σε πολύ θόρυβο και λάθη στη μετάδοση. Απλοί και ανέξοδοι αλγόριθμοι / πρωτόκολλα απαιτούνται για την οργάνωση δικτύων με μεγάλο αριθμό από αισθητήρες.

Κεφάλαιο 2

Βασικές έννοιες στα Ασύρματα δίκτυα αισθητήρων

2.1. Έλεγχος τοπολογίας

2.1.1. Σκοπός

Κύριος στόχος του έλεγχου τοπολογίας είναι η σχεδίαση αποδοτικών αλγορίθμων που θα διατηρούν την διασύνδεση του δικτύου και να βελτιστοποιούν τις μετρήσεις επίδοσης όπως την διάρκεια λειτουργίας του δικτύου και την ταχύτητα μεταφοράς δεδομένων [7]. Η χρήση ελέγχου τοπολογίας είναι απαραίτητη λόγω του ότι τα WSNs τροφοδοτούνται με ενέργεια από μπαταρίες οι οποίες περιορίζουν σε μεγάλο βαθμό τις δυνατότητες του. Μερικές από τις χρήσεις ενός ελέγχου τοπολογίας είναι η αποδοτική επέκταση δικτύων αισθητήρων, ο προσδιορισμός της πυκνότητας αισθητήρων και της δύναμης μετάδοσης των αισθητήρων, προσδιορισμός της καλύτερης εμβέλειας μετάδοσης και διευκόλυνση της διάδοσης μηνυμάτων ελέγχου με έναν αποδοτικό τρόπο.

Λόγο του ότι τα WSNs συνήθως αποτελούνται από μεγάλο αριθμό αισθητήρων και καλύπτουν μεγάλες εκτάσεις, οι επικοινωνίες γίνονται με την αναμετάδοση μίας πληροφορίας από πολλούς κόμβους μέχρι να φτάσει στον κεντρικό κόμβο (multi-hop). Η αναμετάδοση αυτή είναι μία από τις πιο απαιτητικές εργασίες ενός κόμβου αισθητήρα όσο αφορά κατανάλωση ενέργειας. Γι' αυτό το λόγο η μελέτη αλγορίθμων εστιάζεται κυρίως στην καλυτέρευση του τρόπου αναμετάδοσης μίας πληροφορίας από ένα αισθητήρα στον κεντρικό κόμβο.

Υπάρχουν πολλοί τρόποι να προσεγγίσει κανείς ένα πρόβλημα ελέγχου τοπολογίας. Γενικά όμως τα προβλήματα χωρίζονται σε δύο μεγάλες κατηγορίες όπου παίζει ρόλο το είδος του δικτύου που προορίζονται [21]. Μεταξύ των στάσιμων και κινητών δικτύων. Τα πρώτα έχουν σταθερούς κόμβους ενώ στα δεύτερα οι αισθητήρες μπορούν να κινούνται κάνοντας το πρόβλημα αρκετά πιο περίπλοκο.

Στα στάσιμα δίκτυα εκτελείτε μία φορά ο έλεγχος τοπολογίας στην αρχή λειτουργίας του και στη συνέχεια περιοδικά για να λάβει υπόψη αισθητήρες που προστίθενται ή αφαιρούνται από το δίκτυο. Αυτό μας δίνει την δυνατότητα να έχουμε πιο ακριβή και ορθά πρωτόκολλα και αλγόριθμους αφού δεν επηρεάζουν κατά πολύ την ομαλή λειτουργία του δικτύου. Το αντίθετο συμβαίνει στην περίπτωση κινητού δικτύου. Ο έλεγχος τοπολογίας πρέπει να τρέχει συνεχώς για να λαμβάνει υπόψη και τις θέσεις των αισθητήρων που αλλάζουν με την πάροδο του χρόνου.

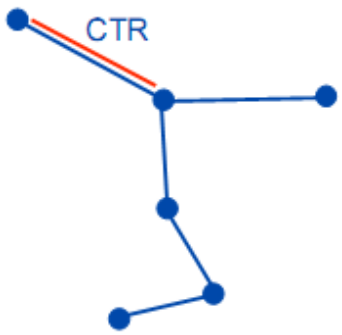
Στη δική μας περίπτωση θα έχουμε κάτι το ενδιάμεσο. Οι αισθητήρες θα είναι σταθεροί αλλά το ρομπότ θα κινείται μέσα στο δίκτυο και θα μεταφέρει μαζί του ένα αισθητήρα. Επίσης θα μετακινεί του ς σταθερούς αισθητήρες όταν το βρίσκει αναγκαίο. Γι' αυτό το λόγο ο έλεγχος τοπολογίας θα εκτελείται μία φορά στην αρχή και μετά κάθε φορά που θα κάνει μία ενέργεια το ρομπότ.

2.1.2. Μέθοδοι ελέγχου τοπολογίας

Στατικά δίκτυα

- Κρίσιμη εμβέλεια μετάδοσης (Critical Transmitting Range)

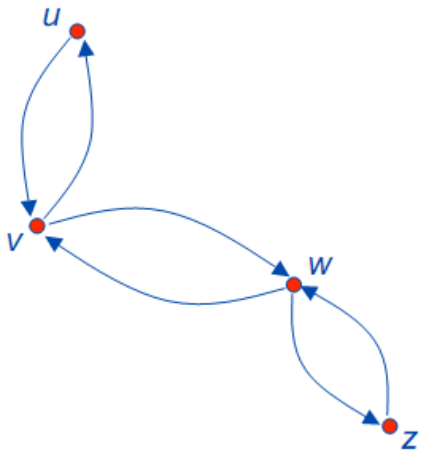
Ένας καλός τρόπος να βελτιώσουμε το χρόνο ζωής του δικτύου είναι να μειώσουμε όσο το δυνατό περισσότερο την εμβέλεια μετάδοσης των αισθητήρων. Αυτό όμως δεν πρέπει να επηρεάσει τη συνδεσιμότητα του δικτύου. Η κρίσιμη εμβέλεια μετάδοσης είναι η ελάχιστη τιμή που μπορεί να έχει η εμβέλεια μετάδοσης διατηρώντας την συνδεσιμότητα. Αυτό μπορεί εύκολα να επιλυθεί αν γνωρίζουμε εκ των προτέρων τις θέσεις των αισθητήρων. Η κρίσιμη εμβέλεια μετάδοσης θα είναι αυτή που χρειάζεται να ενωθεί ο πιο απομακρυσμένος αισθητήρας όπως φαίνεται στο σχήμα 2.1.



Σχήμα 2.1: Κρίσιμη εμβέλεια μετάδοσης [26]

- Ανάθεση εμβέλειας (Range Assignment)

Σε πιο ρεαλιστικές περιπτώσεις όπου η θέση των αισθητήρων δεν είναι γνωστή εκ των προτέρων και η εμβέλεια μετάδοσης διαφέρει από αισθητήρα σε αισθητήρα χρησιμοποιούμε μεθόδους όπως τη ανάθεση εμβέλειας. Όπως φαίνεται στο σχήμα 2.2 αναθέτουμε την ελάχιστη εμβέλεια σε κάθε αισθητήρα για να συνδεθεί στον πιο κοντινό του γείτονα και στη συνέχεια το ανάποδο για να μην προκύψουν μονόπλευρες συνδέσεις.



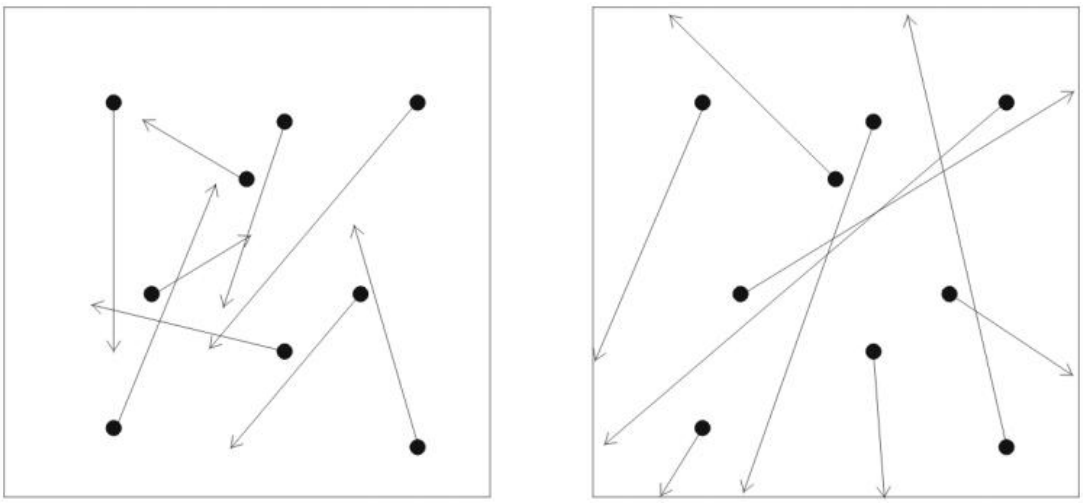
Σχήμα 2.2: Ανάθεση εμβέλειας [26]

Κινητά δίκτυα

- Random waypoint model: Είναι η πιο συνηθισμένη μέθοδος για έλεγχο τοπολογίας σε κινητά δίκτυα αισθητήρων. Κάθε κινητός αισθητήρα επιλέγει τυχαία τον προορισμό του μέσα στην ελεγχόμενη περιοχή και αρχίζει να κινείται προς εκεί με τυχαία ταχύτητα. Όταν φτάσει στον προορισμό του μένει εκεί για ένα χρονικό διάστημα και στη συνέχεια επαναλαμβάνεται το ίδιο.

- Random direction model: Οι κινητοί αισθητήρες επιλέγουν τυχαία μία κατεύθυνση από 0 – 360 μοίρες και αρχίζουν να κινούνται προς εκείνη την κατεύθυνση με τυχαία ταχύτητα. Μετά από ένα ορισμένο χρονικό διάστημα σταματάνε και επαναλαμβάνουν την διαδικασία. Οι κινήσεις στο Random direction model είναι πιο ανεξέλεγκτες λόγο του ότι οι αισθητήρες δεν σέβονται τα όρια της ελεγχόμενης περιοχής όπως φαίνεται στο σχήμα 2.3.

- Brownian motion: Η θέση του κινητού αισθητήρα για την επόμενη χρονική στιγμή επιλέγεται τυχαία μέσα σε μια κυκλική εμβέλεια με κέντρο την τωρινή θέση του αισθητήρα.



(α) (β)

Σχήμα 2.3: Γραφική αναπαράσταση του Random waypoint model(α) και Random direction model(β) [26]

2.2. Καθορισμός θέσης (localization)

2.2.1. Σκοπός

Η δυνατότητα καθορισμού θέσης στα WSNs είναι ιδιαίτερα επιθυμητή σε περιπτώσεις όπως στην ανίχνευση πυρκαγιάς ή στον έλεγχο ποιότητας του νερού, αφού οι πληροφορίες που παίρνουμε θα είναι άχρηστες αν δεν γνωρίζουμε από πού προήλθαν. Οι αλγόριθμοι καθορισμού θέσης στα ασύρματα δίκτυα [12] υπολογίζουν την θέση ενός συγκεκριμένου αισθητήρα βάση της απόστασης του από άλλους κόμβους λαμβάνοντας υπόψη και προς ποια διεύθυνση είναι. Για να είναι αυτό δυνατό υπάρχουν ορισμένοι κόμβοι με γνωστή θέση που την παίρνουν συνήθως από ένα σύστημα GPS.

2.2.2. Μέθοδοι καθορισμού τοποθεσίας για κόμβους αισθητήρα

Τεχνικές Μέτρησης:

- Γωνία προέλευσης (Angle-of-arrival). Η τεχνική αυτή χρησιμοποιεί δύο αντένες οι οποίες μετρούν την διαφορά χρόνου μέχρι να λάβουν ορισμένα πακέτα υπολογίζοντας έτσι την γωνία προέλευσης και στη συνέχεια, αφού γνωρίζει την κατεύθυνση της και απόσταση από ένα γνωστό σημείο, υπολογίζει την θέση της.

- Σχέση απόστασης (Distance related). Σε αυτή τη περίπτωση ο κόμβος στέλνει μηνύματα σε γειτονικούς κόμβους και βάση της χρονικής καθυστέρησης του μηνύματος υπολογίζει την απόσταση του από τους άλλους κόμβους. Στη συνέχεια με την μέθοδο cross-correlation υπολογίζει την θέση του. Η τεχνική αυτή μπορεί να υλοποιηθεί με διάφορες μεθόδους. Οι μέθοδοι αυτοί είναι:
    - One-way propagation time and roundtrip propagation time measurements
    - Lighthouse approach to distance measurements
    - Time-difference-of-arrival measurements
    - Distance estimation via received signal strength measurements
    - RSS profiling measurements

Τεχνικές βασισμένες στη συνδεσιμότητα

- Οι τεχνικές αυτές δεν λαμβάνουν καθόλου υπόψη μετρήσεις μεταξύ κόμβων. Υπολογίζουν την θέση τους μέσα στο δίκτυο βάση του ποιος κόμβος είναι ενωμένος σε ποιο. Δηλαδή χρησιμοποιούν πληροφορίες όπως, ποίοι κόμβοι είναι μέσα στην εμβέλεια άλλων. Η αρχή της τεχνικής αυτής βασίζετε στο ότι, όταν ένας κόμβος είναι μέσα στην εμβέλεια ενός άλλου, τότε καθορίζεται η εγγύτητα μεταξύ των κόμβων που μπορεί να χρησιμοποιηθεί για τον εντοπισμό.

Τεχνικές βασισμένες στη απόσταση

Χωρίζονται σε δύο κατηγορίες. Συγκεντρωτικοί και Κατανεμημένοι αλγόριθμοι. Στους συγκεντρωτικούς αλγόριθμους έχουμε ένα κεντρικό κόμβο ο οποίος λαμβάνει όλες τις πληροφορίες και δημιουργεί το χάρτη για όλο το δίκτυο. Στη περίπτωση των κατανεμημένων αλγόριθμων κάθε κόμβος μέσα στο δίκτυο αναλαμβάνει τον δικό του εντοπισμό.

- Συγκεντρωτικοί αλγόριθμοι. Χρησιμοποιούνται κυρίως σε δίκτυα τα οποία ήδη έχουν μία συγκεντρωτική αρχιτεκτονική όπως έλεγχος φώτων τροχαίας, περιβαλλοντική και ιατρική παρακολούθηση. Σε αυτές τις περιπτώσεις μαζεύονται όλες οι πληροφορίες σε μία κεντρική μονάδα όπου επεξεργάζονται. Ένα σημαντικό πλεονέκτημα σε σχέση με τους κατανεμημένους αλγόριθμους είναι η ακρίβεια στις τοποθεσίες.

- Κατανεμημένοι αλγόριθμοι. Είναι αρκετά πιο δύσκολοι στην υλοποίηση και έχουν λιγότερη ακρίβεια στον υπολογισμό της θέσης τους. Όμως ένας κατανεμημένος αλγόριθμος μπορεί να υλοποιηθεί για κάθε συγκεντρωτικό πρόβλημα αλλά ένας συγκεντρωτικός αλγόριθμος δεν μπορεί να υλοποιηθεί για κάθε κατανεμημένο πρόβλημα.

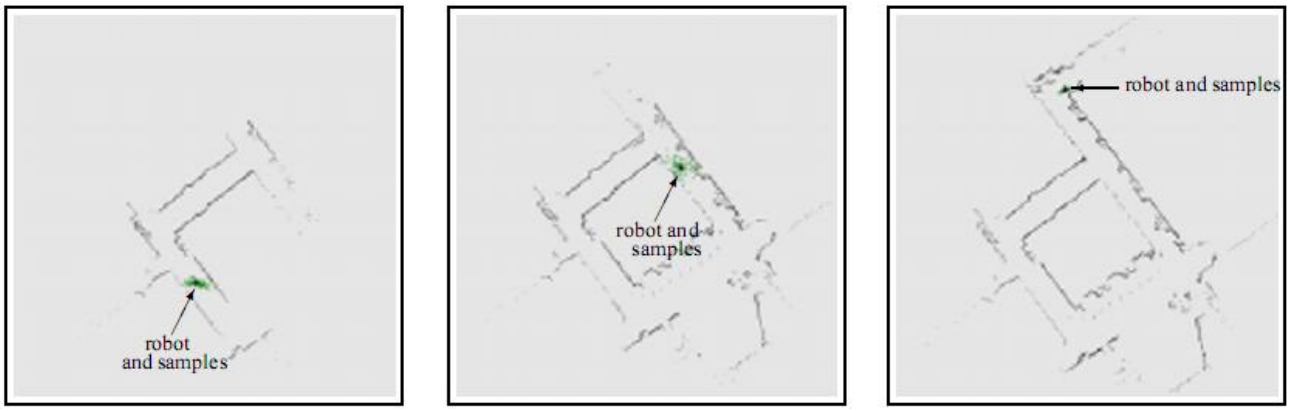
Τεχνική διανύσματος απόστασης άλματος (Distance Vector Hop - DVHop) [13]

Η τεχνική αυτή αποτελείται από διάφορα βήματα. Πρώτα κάθε σταθερός κόμβος που γνωρίζει τη θέση του την αποστέλλει σε όλους τους κόμβους του δικτύου. Κάθε κόμβος που λαμβάνει ένα τέτοιο μήνυμα κρατά αυτό που χρειάστηκε τις λιγότερες αναμεταδόσεις μέχρι να φτάσει εκεί (hop-count). Εν συνεχεία, όταν ένας σταθερός κόμβος λάβει το hop-count ενός άλλου σταθερού κόμβου, υπολογίζει τον μέσο όρο για το μέγεθος μίας αναμετάδοσης (hop-size) το οποίο αποστέλλει σε όλο το δίκτυο. Τώρα κάθε κόμβος όταν λάβει το hop-size, το πολλαπλασιάζει με το hop-count που ήδη γνωρίζει και έτσι υπολογίζει την απόσταση του από τους σταθερούς κόμβους.

2.2.3. Μέθοδοι καθορισμού τοποθεσίας για ρομπότ με αισθητήρες

Τεχνική Simultaneous localization and mapping (SLAM)

Είναι η τεχνική την οποία χρησιμοποιούν ρομπότ και αυτόνομα οχήματα για να χτίζουν χάρτη μέσα σε ένα άγνωστο περιβάλλων και παράλληλα να γνωρίζουν την τοποθεσία τους μέσα στο χάρτη.



Σχήμα 2.4: Σταδιακή δημιουργία χάρτη από ρομπότ με την τεχνική SLAM [22]

Για να λαμβάνει πληροφορίες από το περιβάλλον χρησιμοποιεί κάμερες, λέιζερ και σόναρ. Είναι αρκετά δύσκολο στην εκτέλεση αφού το παραμικρό λάθος στις μετρήσεις απόστασης και διεύθυνση φοράς θα οδηγήσουν στη δημιουργία λάθος χάρτη. Έτσι το ρομπότ δεν θα γνωρίζει την ακριβή του τοποθεσία. Υπάρχουν πολλοί τρόποι να διορθωθεί αυτό το πρόβλημα. Θα μπορούσε για παράδειγμα, να αναγνωρίζει αντικείμενα τα οποία βρήκε προηγουμένως για επιβεβαίωση ότι προχωρεί σωστά μέσα στον χάρτη. Το SLAM παρόλο που δεν έχει τελειοποιηθεί, άρχισε να υλοποιείται για αυτόνομα εναέρια και υποβρύχια οχήματα και ρομπότ εσωτερικής χρήσης.

2.3. Έλεγχος κάλυψης

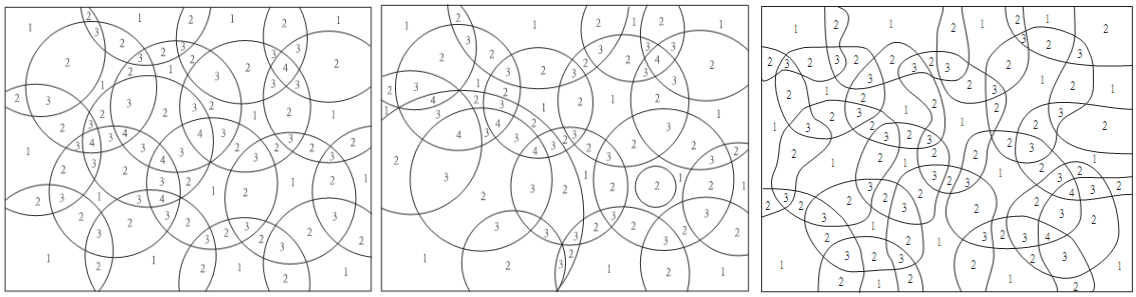
2.3.1. Σκοπός

Ο έλεγχος κάλυψης μας δείχνει κατά πόσο μία περιοχή μέσα στο δίκτυο καλύπτετε από τους υφιστάμενους αισθητήρες και σε πιο βαθμό [3]. Όταν γνωρίζουμε αυτές τις πληροφορίες μπορούμε να δούμε αν υπάρχουν περιοχές που υπερκαλύπτονται από πολλούς αισθητήρες και να θέσουμε ορισμένους σε κατάσταση ‘’ύπνου’’ έτσι ώστε να αυξήσουμε την διάρκεια ζωής του δικτύου κρατώντας σταθερή την κάλυψη του.

Υπάρχουν ορισμένα πιο κρίσιμα δίκτυα τα οποία πρέπει να καλύπτοντα από ένα ελάχιστο αριθμό αισθητήρων σε κάθε σημείο. Με τον έλεγχο κάλυψης μπορούμε να δούμε πού χρειάζεται περισσότερη κάλυψη και να προσθέσουμε αισθητήρες στη συγκεκριμένη περιοχή. Η υπερβολική κάλυψη όμως μπορεί να προκαλεί πολύ θόρυβο στις επικοινωνίες και υπερβολική κατανάλωση ενέργειας. Σε τέτοιες περιπτώσεις πρέπει να αφαιρέσουμε αισθητήρες από τις περιοχές.

Για να θεωρείται μία περιοχή καλυμμένη πρέπει να ισχύει το πιο κάτω. Ένα σημείο μέσα στην ελεγχόμενη περιοχή καλύπτεται όταν βρίσκεται μακριά από ένα αισθητήρα το πολύ όσο η εμβέλεια του. Αν όλα τα σημεία μέσα στην περιοχή πληρούν αυτό το κριτήριο, τότε έχουμε ολική κάλυψη.

Σε πραγματικές καταστάσεις είναι αρκετά δύσκολο να υπολογίσουμε την κάλυψη λόγο του ότι η εμβέλεια των αισθητήρων επηρεάζεται από εμπόδια όπως τοίχους και διαφέρει λόγο έλλειψης ενέργειας. Αν λάβουμε υπόψη όλες τις παρεμβολές μέσα στο δίκτυο θα αυξηθεί δραματικά η πολυπλοκότητα και θα υπάρχουν πολλές εξαρτήσεις [3]. Στο σχήμα 2.5 βλέπουμε τις διαφορές στον έλεγχο κάλυψης ανάλογα με το πόσες παραμέτρους λαμβάνουμε υπόψη.



(α) (β) (γ)

Σχήμα 2.5: Έλεγχος κάλυψης. Οι αριθμοί δείχνουν πόσο καλύπτεται μία περιοχή [3]
(α) όλοι οι αισθητήρες έχουν την ίδια ακτίνα εμβέλειας χωρίς εμπόδια
(β) κάθε αισθητήρας έχει ξεχωριστή ακτίνα εμβέλειας χωρίς εμπόδια
(γ) κάθε αισθητήρας έχει ξεχωριστή ακτίνα εμβέλειας με εμπόδια

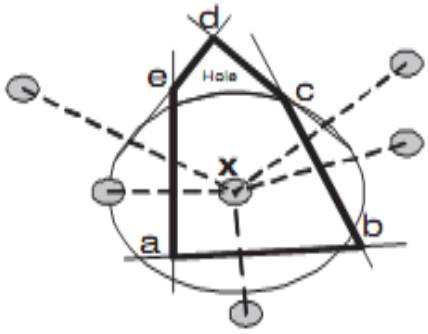
2.3.2. Μέθοδοι ελέγχου κάλυψης

Υπάρχουν διάφορες τεχνικές για εντοπισμό και διόρθωση ακάλυπτων περιοχών μέσα σε ένα WSN. Οι τεχνικές αυτές χωρίζονται βάση του αν οι αισθητήρες μέσα στο δίκτυο μπορούν να κινούνται ή όχι.

Δίκτυα με κινούμενους αισθητήρες

Στη περίπτωση που οι αισθητήρες μέσα στο δίκτυο έχουν τη δυνατότητα να κινούνται [1][4], υπάρχουν τεχνικές για να απλωθούν μέσα στην ελεγχόμενη περιοχή για να αυξήσουν την έκταση κάλυψης ή και να καλύψουν ακάλυπτες περιοχές.

- Μία τεχνική είναι με την χρήση των διαγραμμάτων βορονόι (voronoi diagrams) [24]. Για να πετύχει όμως, κάθε αισθητήρας πρέπει να γνωρίζει τις τοποθεσίες των γειτόνων του ούτος ώστε να μπορεί να δημιουργήσει το διάγραμμα βορονόι και να βρει τυχόν κενά όπως φαίνεται στο σχήμα. Αφού εντοπιστεί το κενό, ο αισθητήρας κινείται προς την ακμή που βρίσκετε πιο μακριά όπως φαίνεται στο σχήμα 2.6, σε αυτή την περίπτωση την d, για να καλύψει το μεγαλύτερο κενό.



Σχήμα 2.6: Διάγραμμα βορονόι για τον εντοπισμό ακάλυπτων περιοχών [24]

Υβριδικά δίκτυα

Σε αυτή τη περίπτωση έχουμε περιορισμένο αριθμό αισθητήρων που μπορούν να κινηθούν μέσα στην ελεγχόμενη περιοχή. Χρησιμοποιούμε αυτούς τους αισθητήρες για να καλύψουμε κενά μέσα στο δίκτυο ή για να ενώσουμε απομονωμένα κομμάτια του δικτύου.

- Ένας τρόπος είναι να έχουμε ένα ρομπότ το οποίο κινείται συνεχώς μέσα στην ελεγχόμενη περιοχή. Αυτό θα επικοινωνεί με τους γειτονικούς αισθητήρες κάθε φορά που αλλάζει τοποθεσία. Αποφασίζει προς τα πού θα κινηθεί βάση της έντασης των σημάτων που λαμβάνει. Σε περίπτωση που δεν υπάρχει επικοινωνία σημαίνει ότι η περιοχή είναι ακάλυπτη έτσι το ρομπότ μπορεί να τοποθετήσει εκεί ένα επιπλέον αισθητήρα.

- Μία δεύτερη προσέγγιση χρησιμοποιεί τα διαγράμματα βορονόι όπως είδαμε προηγουμένως. Η μόνη διαφορά είναι ότι δεν μπορούν όλοι οι αισθητήρες να κινηθούν. Γι' αυτό κάθε σταθερός αισθητήρας όταν εντοπίσει ένα κενό, στέλνει μήνυμα σε ένα κινητό αισθητήρα. Αυτός με την σειρά του επιλέγει να πάει προς τον αισθητήρα που έχει το μεγαλύτερο κενό. Εάν η μετακίνηση αυτή δεν δημιουργεί μεγαλύτερο κενό στην θέση που ήταν ο κινητός αισθητήρας, εκτελείται.

Δίκτυα με σταθερούς αισθητήρες

Δίκτυα με σταθερούς αισθητήρες είναι η πιο συνηθισμένη και απλή μορφή WSN. Το πρόβλημα του ελέγχου κάλυψης όμως εξακολουθεί να έχει την δυσκολία του. Ο έλεγχος θα πρέπει να επιλέγει τον ελάχιστο αριθμό από αισθητήρες που θα βρίσκονται σε υπηρεσία και θα καλύπτουν επαρκώς τη περιοχή. Κύριος στόχος είναι η εξοικονόμηση ενέργειας και επέκταση του χρόνου ζωής του δικτύου χωρίς να χάνουμε από την κάλυψη.

- Μία μέθοδος είναι το πρωτόκολλο ρύθμισης κάλυψης (Coverage Configuration Protocol - CCP) [24]. Κάθε αισθητήρας του δικτύου τρέχει τον αλγόριθμο κάλυψη επιλεξιμότητας (coverage eligibility) ο οποίος ελέγχει αν κάθε σημείο της περιοχής που καλύπτει, καλύπτεται και από άλλο αισθητήρα. Σε περίπτωση που ισχύει, ο αισθητήρας μπαίνει σε κατάσταση 'ύπνου'. Το μειονέκτημα όμως είναι ότι αυτός ο έλεγχος δεν εγγυείται την συνδεσιμότητα του δικτύου.

- Μία άλλη μέθοδος υπολογίζει αν κάθε σημείο μέσα στην ελεγχόμενη περιοχή καλύπτεται από τον απαραίτητο αριθμό αισθητήρων [24]. Υπάρχουν δύο εκδοχές αυτής της μεθόδου. Ο αλγόριθμος k-UC ο οποίος υποθέτει ότι η εμβέλεια κάθε αισθητήρα είναι κυκλική και ο k-NC ο οποίος υποθέτει ότι η εμβέλεια κάθε κόμβου δεν είναι κυκλική. Κάθε αισθητήρας ελέγχει αν η περίμετρος της περιοχής που καλύπτει, καλύπτεται από τους γειτονικούς αισθητήρες. Αυτές οι πληροφορίες αποστέλλονται σε μία κεντρική μονάδα που θα εκτελεί τον αλγόριθμο για να βρίσκει κενά μέσα στο δίκτυο.

- Η μέθοδος OGDC (Optimal Geographical Density Control) [24] έχει ως σκοπό να ελαχιστοποιήσει την υπερκάλυψη μέσα στην ελεγχόμενη περιοχή. Όλοι οι αισθητήρες ξεκινούν στην κατάσταση 'αναποφάσιστο'. Ένας τυχαίος αισθητήρας στέλνει σε όλους τους γείτονες του μήνυμα αφύπνισης. Εάν η περιοχή που καλύπτουν είναι ακόμη ακάλυπτη τότε μπαίνουν σε κατάσταση λειτουργίας και στέλνουν το μήνυμα αφύπνισης στους δικούς τους γείτονες. Αυτό συνεχίζεται μέχρι να λάβουν όλοι οι αισθητήρες μήνυμα.

2.4. Εντοπισμός στόχου

2.4.1. Σκοπός

Ο εντοπισμός στόχου μέσα σε ένα WSN είναι μία αρκετά απαιτητική εργασία [27]. Αυτό οφείλεται στο ότι το δίκτυο πρέπει να είναι σε συνεχή επικοινωνία με τον πραγματικό κόσμο αλλά παράλληλα πρέπει να ελέγχουμε και την διάρκεια ζωής του δικτύου [18]. Δηλαδή δεν θα ήταν σωστό να έχουμε συνεχώς σε λειτουργία όλους τους αισθητήρες, για να πετύχουμε την γρήγορη ανταπόκριση του δικτύου, επειδή θα προκαλέσει την γρήγορη εξάντληση της ενέργειας των αισθητήρων. Επίσης οι αισθητήρες δεν έχουν αρκετή υπολογιστική ισχύ για να μπορούν να υπολογίσουν σωστά και άμεσα την πορεία του στόχου και να ειδοποιήσουν τους επόμενους αισθητήρες οι οποίοι μπορεί να είναι σε κατάσταση εξοικονόμησης ενέργειας.

2.4.2. Μέθοδοι εντοπισμού στόχου από τους κόμβους αισθητήρων

Ένας αλγόριθμος για εντοπισμό στόχου είναι ο εξής [33]:

1 *Αρχική ενεργοποίηση*: Σε περίπτωση που η κάλυψη του δικτύου είναι σταθερή, τότε με την είσοδο του στόχου μέσα στο δίκτυο έχουμε άμεση ενεργοποίηση. Αν οι αισθητήρες μπαίνουν σε κατάσταση εξοικονόμησης ενέργειας τότε μπορεί να έχουμε μια μικρή καθυστέρηση μέχρι να γίνει η αρχική ενεργοποίηση.

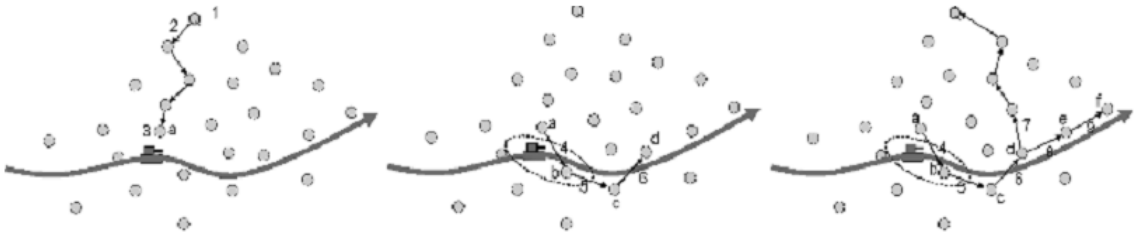
2 *Αρχικός εντοπισμός του στόχου*: Αφού γίνει η ενεργοποίηση χρειάζεται λίγος χρόνος μέχρι να επιβεβαιώσει ο αισθητήρας ότι πράγματι εντόπισε το στόχο. Η καθυστέρηση αυτή οφείλεται στο ότι χρειάζεται ένα ορισμένο αριθμό από πακέτα για να γίνει η επιβεβαίωση.

3 "*Ξύπνημα"*: Στη συνέχεια ο αισθητήρας ο οποίος εντόπισε το στόχο "ξυπνά" γειτονικούς αισθητήρες που μπορεί να είναι σε κατάσταση εξοικονόμησης ενέργειας για να γίνει πιο ακριβής ο εντοπισμός.

4 *Ομαδική συνάθροιση*: Όταν ενεργοποιηθούν οι αισθητήρες, όσοι από αυτούς μπορούν να εντοπίσουν το στόχο μπαίνουν στην ίδια ομάδα. Κάθε ομάδα που δημιουργείται έχει ένα αρχηγό ο οποίος παίρνει αναφορές από τους υπόλοιπους όσο αφορά το στόχο.

5 *Αναφορά βάσης*: Στη συνέχεια, αφού μαζέψει αρκετά δεδομένα ένας αρχηγός, τα στέλλει στη βάση. Μπορεί να υπάρχουν πολλές βάσεις οι οποίες χωρίζουν το δίκτυο σε τομείς.

6 *Επεξεργασία δεδομένων*: Από τη στιγμή που έχει τα απαραίτητα δεδομένα μία βάση μπορεί να υπολογίσει την διαδρομή που ακολούθησε ο στόχος, την κατεύθυνση του και την ταχύτητα του.



(α) (β) (γ)

Σχήμα 2.7: Εντοπισμός στόχου [2]

(α) Εντοπισμός στόχου από ασύρματο δίκτυο αισθητήρων

(β) Αφύπνιση γειτονικών αισθητήρων

(γ) Ενημέρωση βάσης.

2.4.3. Μέθοδοι εντοπισμού στόχου από ρομπότ με αισθητήρες

Σε εφαρμογές όπως η κατ'οίκον παρακολούθηση ηλικιωμένων και έξυπνα περιβάλλοντα, είναι απαραίτητος ο εντοπισμός αντικειμένων, ανθρώπων και στόχων από τα ρομπότ [29][30][32]. Τα αυτόνομα ρομπότ πρέπει να έχουν αποδοτικές και αξιόπιστες μεθόδους στο να εντοπίζουν και να ακολουθούν ένα στόχο. Δύο στρατηγικές που θα δούμε πιο κάτω είναι η Διαδικασία Απόφασης Μερικής Αίσθησης Markov (Partially Observable Markov Decision Process - POMDP trackers) [6] και Άπληστοι Ακολουθητές (Greedy followers) [31].

- POMDP Trackers

Ο εντοπισμός των στόχων έχει δύο παραλλαγές που μελετώνται συχνά ανεξάρτητα με τις διαφορετικές προσεγγίσεις: ο εντοπισμός στόχων που απαιτεί ένα ρομπότ να βρει ένα στόχο, αρχικά μη ορατό, και την παρακολούθηση του στόχο όπου ένα ρομπότ πρέπει να διατηρήσει τη ορατότητα με τον στόχο, αρχικά ορατό. Χρησιμοποιούμε τη μέθοδο POMDP για να χτίσουμε ένα ενιαίο πρότυπο που ενοποιεί τον εντοπισμό στόχων και την ακολουθία στόχων. Η παρακολούθηση στόχου με τη μέθοδο POMDP μειώνει την αβεβαιότητα στη θέση των στόχων, και δεν συμβιβάζεται η απόδοση παρακολούθησης.

- Greedy Followers

Η μέθοδος POMDP ενσωματώνει τις πληροφορίες για τη συμπεριφορά στόχων και το περιβάλλον για να κάνει τη βέλτιστη απόφαση. Όταν λίγα είναι γνωστά για τη συμπεριφορά των στόχων ή το περιβάλλον, μια τοπική greedy ("άπληστη") στρατηγική είναι αποτελεσματικότερη. Το κλειδί για τον αλγόριθμό είναι ο καθορισμός μιας μεθόδου η οποία προσπαθεί να συλλάβει το στόχο όταν πάει να διαφύγει από την περιοχή που καλύπτουν οι αισθητήρες του ρομπότ.

Για να επιλέξει ποια ενέργεια θα εκτελέσει, το ρομπότ πρέπει να ισορροπήσει μεταξύ του βραχυπρόθεσμου στόχου που είναι να μην υπάρξει η άμεση απώλεια του στόχου και του μακροπρόθεσμου στόχου η οποία είναι να κρατήσει τον στόχο σε ορατότητα το μέγιστο

πιθανό χρονικό διάστημα.. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας μόνο την τοπική διαθέσιμη πληροφορία στους αισθητήρες του ρομπότ. Με την ανάλυση της τοπικής γεωμετρίας, ο αλγόριθμός επιλέγει μία δράση για να ελαχιστοποιηθεί ο κίνδυνος να χάσουμε το στόχο με ένα "άπληστο" τρόπο.

Δεδομένου ότι ο αλγόριθμος χρησιμοποιεί μόνο την τοπική γεωμετρική διαθέσιμη πληροφορία στους οπτικούς αισθητήρες του ρομπότ, δεν απαιτεί έναν χάρτη όλης της περιοχής και παρακάμπτει έτσι τη δυσκολία του καθορισμού τοποθεσίας μέσα σε έναν ολικό χάρτη. Επιπλέον, η ανακρίβεια στις μετρήσεις των αισθητήρων του ρομπότ δεν προκαλούν κανένα πρόβλημα. Αυτό βελτιώνει την αξιοπιστία της παρακολούθησης στόχου. Αυτή η προσέγγιση μπορεί να αναπτυχθεί δισδιάστατη και τρισδιάστατη.

2.5. Κινητικότητα

2.5.1. Σκοπός

Η κινητικότητα σε ένα WSN μπορεί να μας εξυπηρετήσει σε διάφορους τομείς. Οι κινητοί αισθητήρες μπορούν να βελτιώνουν τη θέση τους ώστε να έχουν καλύτερη κάλυψη, να μπορούν να στείλουν δεδομένα με λιγότερη ενέργεια και να λύσουν τα προβλήματα κάλυψης του δικτύου. Παρόλα αυτά όμως, η ενέργεια των αισθητήρων είναι αρκετά περιοριστική. Αντί αυτού, μπορούμε να έχουμε κινητές μονάδες που μόνος τους σκοπός θα είναι μεταφορά δεδομένων σε/από απομακρυσμένα τμήματα του δικτύου βελτιώνοντας έτσι τη συνδεσιμότητα του δικτύου. Αυτό βοηθά και στη εξοικονόμηση ενέργειας στους αισθητήρες αφού τα δεδομένα θα περνούν από λιγότερους αισθητήρες για να φτάσουν στο προορισμό τους (multihop) [35].

2.5.2. Μέθοδοι κινητικότητας

Οι μέθοδοι κινητικότητας μέσα σε ένα WSN [8] μπορούν να χωριστούν βάση της ιδιότητας του κινούμενου κόμβου και του τρόπου μετάδοσης των πληροφοριών μέσα στο δίκτυο.

Κινητή βάση

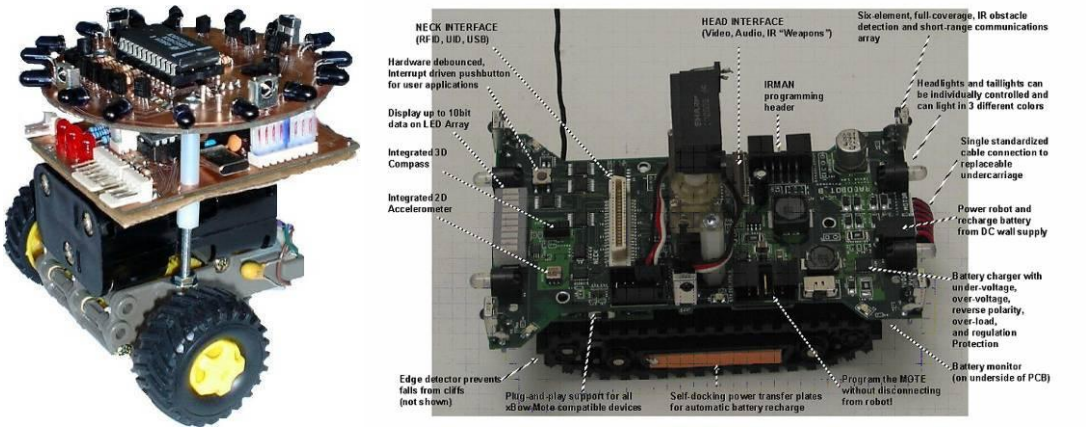
Κατά τη διάρκεια λειτουργίας του δικτύου, η κεντρική βάση μπορεί να κινείται μέσα στο δίκτυο και να λαμβάνει πληροφορίες από τους αισθητήρες χωρίς καθυστερήσεις.

Κινητός συλλέκτης πληροφοριών

Ένας κινητός συλλέκτης πληροφοριών [14] επισκέπτεται τους αισθητήρες μέσα στην ελεγχόμενη περιοχή με το πιο σύντομο μονοπάτι [25]. Συλλέγει πληροφορίες και τις κρατά μαζί του μέχρι να επισκεφτεί μία βάση. Όταν επισκεφτεί την βάση, της μεταφέρει απευθείας όλες τις πληροφορίες που μάζεψε.

Μεταφορά δεδομένων με συνάντηση

Η μέθοδος αυτή είναι η ένωση των δύο προηγούμενων. Οι αισθητήρες στέλλουν τις πληροφορίες τους σε σημεία συνάντησης απ' όπου θα περάσει ένας κινούμενος συλλέκτης πληροφοριών. Ο συλλέκτης αυτός παίρνει τις πληροφορίες από το σημείο συνάντησης και τις μεταφέρει στη βάση.

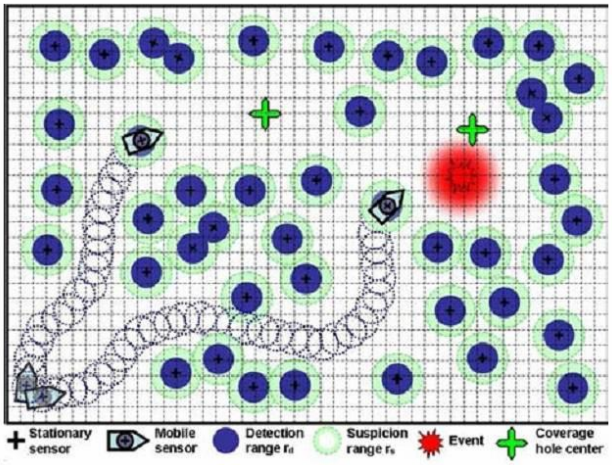


Σχήμα 2.8: Παραδείγματα ασύρματων κόμβων αισθητήρα με δυνατότητα κίνησης [16]

Sens-r και Ragobot

Κινητοί αισθητήρες

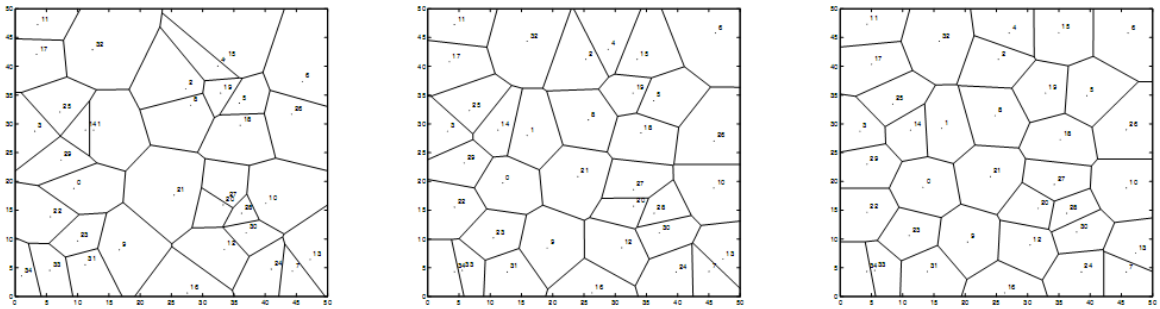
Ο έλεγχος μίας μεγάλης περιοχής απαιτεί μεγάλο αριθμό από αισθητήρες. Γι'αυτό μπορούν να χρησιμοποιηθούν κινητοί αισθητήρες που θα καλύπτουν τα κενά των σταθερών αισθητήρων. Με αυτό το τρόπο μειώνουμε κατα μεγάλο βαθμό τον αριθμό αισθητήρων που χρειαζόμαστε αλλά παράλληλα όλη η περιοχή είναι καλυμμένη. Το μειονέκτημα είναι ότι δεν θα έχουμε συνεχή κάλυψη σε όλη την περιοχή. Πράγμα που μας αποτρέπει τη χρήση του σε κρίσιμα δίκτυα όπου απαιτείται η συνεχής παρακολούθηση όλης της περιοχής.



Σχήμα 2.9: Κάλυψη της περιοχής με κινητούς αισθητήρες [19]

Κίνηση αισθητήρων για αρχική τοποθέτηση

Σε περιπτώσεις όπου η τοποθέτηση των αισθητήρων δεν μπορεί να γίνει χειροκίνητα, οι αισθητήρες θα πρέπει να έχουν τη δυνατότητα κίνησης για να οργανωθούν και να τοποθετηθούν στο κατάλληλο σημείο. Όπως είδαμε και στο κεφάλαιο 2.3 για τον έλεγχο κάλυψης, έτσι και εδώ οι αισθητήρες με δυνατότητα κίνησης χρησιμοποιούν τα διαγράμματα βορονόι για να εντοπίσουν το κενό κοντά τους και να μετακινηθούν προς τα εκεί. Οι αισθητήρες συνεχίζουν να κινούνται κατ'αυτό το τρόπο μέχρι να συμπληρωθούν όλα τα κενά.



Σχήμα 2.10: Κίνηση αισθητήρων για αρχική τοποθέτηση με τη χρήση διαγραμμάτων βορονόι [10][11]

Όπως φαίνεται στο σχήμα 2.10, έχουμε αρχικά μια τυχαία τοποθέτηση των αισθητήρων. Στη συνέχεια δημιουργούν τα διαγράμματα voronoi για να εντοπιστούν κενά. Όσοι από τους αισθητήρες εντόπισαν κενά κινούνται προς το μεγαλύτερο για να το καλύψουν. Καθώς κινούνται υπολογίζουν συνεχώς τα διαγράμματα voronoi βάση της καινούριας τοπολογίας του δικτύου και προσαρμόζουν τον προορισμό τους ανάλογα με το κενό. Αυτό συνεχίζεται μέχρι να εξαπλωθούν οι αισθητήρες μέσα στην περιοχή και να καλύψουν όλα τα κενά.

Κεφάλαιο 3

Πρόβλημα προς επίλυση

3.1. Παρουσίαση προβλήματος προς επίλυση

Σε αυτή τη διπλωματική θα επιχειρήσουμε να προγραμματίσουμε ένα Lego Mindstorm NXT robot με κατάλληλους αλγορίθμους ούτος ώστε να μπορεί αυτόνομα να υποστηρίζει την κινητικότητα σε ένα ασύρματο δίκτυο αισθητήρων.

Το robot θα πρέπει να είναι σε θέση να κάνει τα εξής:

- Πρώτα από όλα είναι ο εντοπισμός (localization). Πρέπει το robot να ξέρει πού βρίσκονται οι αισθητήρες μέσα στη περιοχή που ελέγχουμε και κατ' επέκταση να υπολογίζει τη θέση του σε σχέση με τους αισθητήρες. Η επικοινωνία θα γίνεται με ένα δέκτη ο οποίος βρίσκεται πάνω στο robot με πρωτόκολλο επικοινωνίας I2C.

- Στη συνέχεια έχουμε τον έλεγχο κάλυψης (coverage control). Αφού γνωρίζει τις τοποθεσίες των αισθητήρων, τρέχει ένα αλγόριθμο ο οποίος είναι υπεύθυνος να βρει κενά μέσα στην ελεγχόμενη περιοχή. Δηλαδή χώρους οι οποίοι δεν καλύπτονται από τους υφιστάμενους αισθητήρες. Ο αλγόριθμος αυτός λαμβάνει υπόψη, εκτός από τις τοποθεσίες

των αισθητήρων, την εμβέλεια τους. Παράλληλα μπορεί να βρει και περιοχές οι οποίες υπερκαλύπτονται.

- Τέλος έχουμε την κινητικότητα (mobility) μέσα στο δίκτυο. Τώρα που γνωρίζει πού υπάρχει κενό και πού βρίσκεται το ίδιο, μπορεί να υπολογίσει τη διαδρομή που θα ακολουθήσει για να μεταφέρει ένα αισθητήρα ώστε να βελτιωθεί η γενική κάλυψη της περιοχής. Το αντίστοιχο συμβαίνει όταν έχουμε υπερκάλυψη. Σε περίπτωση που έχουμε και τις δύο περιπτώσεις μέσα στην ελεγχόμενη περιοχή (κενό και υπερκάλυψη), το robot θα πρέπει να υπολογίσει την διαδρομή προς το σημείο της υπερκάλυψης και αφού πιάσει τον επιπλέον αισθητήρα, να συνεχίσει από εκεί στο σημείο του κενού για να αφήσει εκεί τον αισθητήρα.

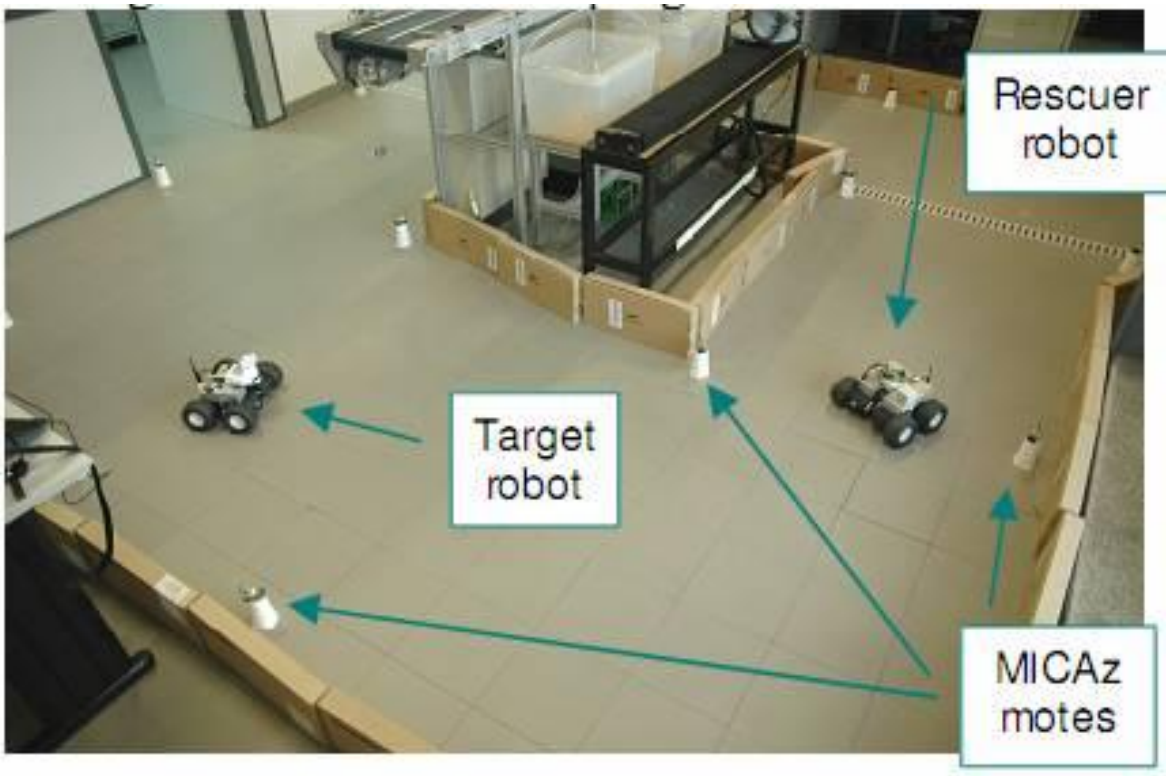
- Επιπλέον το robot θα πρέπει να υπολογίζει, βάση του αλγόριθμου πλανόδιου πωλητή (traveling salesman protocol) την βέλτιστη διαδρομή για να επισκεφτεί όλους τους αισθητήρες μέσα στο δίκτυο για σκοπούς ελέγχου. Σε περιπτώσεις που 2 ή περισσότεροι αισθητήρες καλύπτουν την ίδια περιοχή (cluster) το robot μπορεί απλά να επισκεφτεί το συγκεκριμένο σημείο για να ελέγξει πολλούς αισθητήρες αποφεύγοντας άσκοπες διαδρομές.

- Επίσης το δίκτυο θα μπορεί να εντοπίζει τη θέση του robot στόχου κάθε στιγμή (target tracking) και να αποστέλλει τις πληροφορίες τις πορείας του σε ένα δεύτερο robot. Αυτό θα υπολογίζει την πορεία, ταχύτητα και πιθανή μελλοντική θέση του στόχου. Στη συνέχεια είτε θα το ακολουθεί, είτε θα πηγαίνει απευθείας στη μελλοντική του θέση για να το συναντήσει.

3.2. Προηγούμενη δουλειά

3.2.1. Εφαρμογή εντοπισμού και καταδίωξης στόχου μέσω ενός WSN

Στο πολυτεχνικό ινστιτούτο του Πόρτο στη Πορτογαλία μία ομάδα ερευνητών υλοποίησε μία εφαρμογή εντοπισμού και καταδίωξης στόχου [28]. Χρησιμοποιούν ένα ασύρματο δίκτυο αισθητήρων για τον εντοπισμό του στόχου και για να καθοδηγούν το δεύτερο ρομπότ προς τον στόχο. Οι αισθητήρες που χρησιμοποιήθηκαν είναι οι MicaZ με το λειτουργικό σύστημα TinyOS, και τα ρομπότ είναι δύο WifiBot. Στην εικόνα (α) βλέπουμε τα ρομπότ εν δράση.

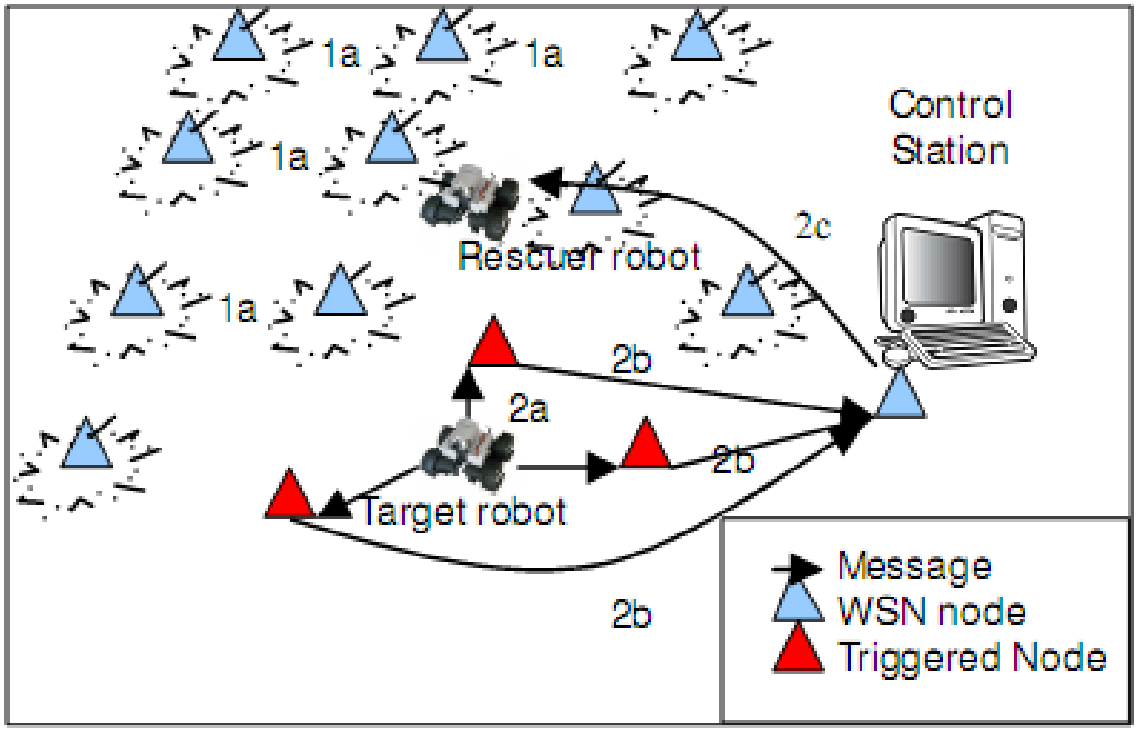


Σχήμα 3.1: Εφαρμογή εντοπισμού και καταδίωξης στόχου εν δράση [28]

Η εφαρμογή λειτουργεί ως εξής. Το πρώτο ρομπότ, το οποίο θεωρείται ως ο στόχος (target robot), κινείται τηλεκατευθυνόμενα μέσα στο δίκτυο από ένα χειριστή. Ο κόμβος αισθητήρα που βρίσκετε πάνω στο ρομπότ στέλνει κατά τακτά χρονικά διαστήματα μηνύματα για την ύπαρξη του τα οποία φτάνουν στη βάση (control station).

Στη συνέχεια η βάση παίρνει τα μηνύματα και υπολογίζει διάφορες πληροφορίες για τον στόχο όπως τοποθεσία και κατεύθυνση. Αμέσως μετά, ειδοποιά το δεύτερο ρομπότ (rescuer robot) για την πιο πρόσφατη γνωστή τοποθεσία του στόχου. Αυτό ξεκινά να κινείται προς τον στόχο με αυτόνομο τρόπο πλοήγησης.

Όλη αυτή η διαδικασία επαναλαμβάνεται συνεχώς μέχρι το ρομπότ να φτάσει κοντά στο στόχο. Μέχρι στιγμής η εφαρμογή υποστηρίζει μόνο ένα ρομπότ στόχο και ένα ρομπότ κυνηγό. Οι ερευνητές κάνουν προσπάθειες για να προσθέσουν κι άλλα ρομπότ και στις δύο ομάδες.



Σχήμα 3.2: Διακίνηση μηνυμάτων μέσα στο δίκτυο [28]

3.2.2. Εφαρμογή CLARITY

Στο τμήμα πληροφορικής του University College Dublin διεξάγεται έρευνα για την συνεργασία ενός ασύρματου δικτύου αισθητήρων με κινούμενα ρομπότ και διάφορες ασύρματες συσκευές όπως κινητά τηλέφωνα και PDAs [9].

Για τα πειράματα χρησιμοποιήθηκαν τα πιο κάτω υλικά:

- 70 Berkley motes αισθητήρες με δυνατότητα παρακολούθησης υγρασίας, θερμοκρασίας και φωτός
- 10 κινητά ρομπότ με πολλούς αισθητήρες τελευταίας τεχνολογίας όπως USB cameras, laser απόστασης, υπερήχους, υπέρυθρες, οδόμετρα και αισθητήρες επαφής
- Ασύρματες συσκευές (κινητά τηλέφωνα, PDAs)

Σχήμα 3.3: Τα ρομπότ του CLARITY [9]

Μεγάλη έμφαση δόθηκε στους ακόλουθους τρεις τομείς:

- Ολοκλήρωση μεταξύ των ρομπότ και του ασύρματου δικτύου αισθητήρων

Αυτός ο τομέας ερευνά την ασύρματη δικτύωση, τη συγκομιδή πληροφοριών και την ανάλυση των δεδομένων αυτών μέσα σε ένα ασύρματο δίκτυο αισθητήρων. Τα ρομπότ είναι μία τάξη χρηστών των πληροφοριών που παίρνουν, είτε από τους δικούς τους αισθητήρες, είτε από τους αισθητήρες του δικτύου.
Τα ρομπότ εκτός από το να ενεργούν μέσα στο περιβάλλων του δικτύου, μπορούν να το βοηθούν με την πρόσθεση, τον προγραμματισμό και την συντήρηση των αισθητήρων. Επίσης να βοηθούν στον εντοπισμό των αισθητήρων και την αναμετάδοση πληροφοριών.

- Προσαρμοστικές και αυτό-οργανωτικές αρχιτεκτονικές λογισμικού

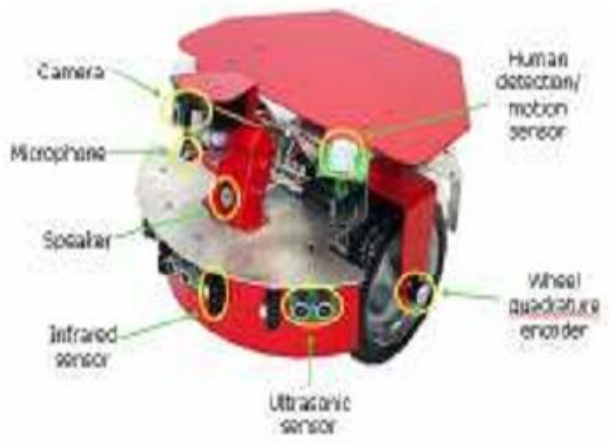
Μέσα σε άγνωστα περιβάλλοντα τα ρομπότ απαιτούν μία δυναμική προσέγγιση που να τους επιτρέπει να προσαρμόζονται και να αναδιοργανώνονται.
Το CBSE (Component-Based Software Engineering) και το AOSE (Agent-Oriented Software Engineering) είναι δύο υποψήφιες λύσεις οι οποίες προσφέρουν αρχιτεκτονική που οργανώνει δυναμικά τις διαφορετικές λειτουργίες των συστημάτων.

- Προσωπικά και κοινωνικά βοηθητικά ρομπότ

Τα ρομπότ εκτός από το να επικοινωνούν με το δίκτυο και να εκτελούν διάφορες περίπλοκες εργασίες, θα πρέπει να συμπεριφέρονται με ένα κοινωνικά αποδεκτό τρόπο και να αντιμετωπίζουν όλες τις απαιτήσεις του κάθε χρήστη. Αυτό οφείλετε στο ότι τα ρομπότ και οι άνθρωποι θα συνυπάρχουν στον ίδιο χώρο οπότε η επικοινωνία και η διαπροσωπεία μεταξύ τους θα έχει μεγάλη σημασία.

3.2.3. Wireless Sensor-driven Intelligent Navigation Robots for Indoor Construction Site Security and Safety

Στο ινστιτούτο Peter Kiewit στη Omaha, Nebraska γίνετε μελέτη για την ανάπτυξη έξυπνων ρομπότ που θα προσφέρουν μία υψηλού επιπέδου πλοήγηση υποβοηθούμενοι από ένα ασύρματο δίκτυο αισθητήρων [34]. Τα ρομπότ είναι εξοπλισμένα με πολλούς αισθητήρες όπως βλέπουμε στην εικόνα. Απώτερος σκοπός των μελετητών είναι να χρησιμοποιήσουν τα ρομπότ και το ασύρματο δίκτυο αισθητήρων σε αποθήκες, γραφεία, και οικοδομές όπου θα βρίσκουν ανωμαλίες και θα προσφέρουν κάποιο είδος ασφάλειας.



Σχήμα 3.4: Αισθητήρες του κινητού ρομπότ [34]

Για να πετύχουν τον σκοπό τους, τα ρομπότ θα πρέπει να είναι αυτόνομα και να εξερευνούν την περιοχή την οποία βρίσκονται. Η ποιότητα όμως εξαρτάτε από πολλούς παράγοντες. Την ακρίβεια στην αίσθηση, το χτίσιμο του χάρτη και την πλοήγηση.

Ο εντοπισμός είναι ένας σημαντικός τομέας στη κίνηση του ρομπότ. Η sensor-based εξερεύνηση επιτρέπει σε ένα ρομπότ να εντοπίσει τη θέση του, να ερευνήσει ένα περιβάλλον και να χτίσει ένα χάρτη χρησιμοποιώντας sonar και λέιζερ. Εντούτοις, αυτές οι τεχνολογίες απαιτούν γραμμή θέασης για να καταγράψει τη θέση του το ρομπότ σε έναν χάρτη και περιορίζουν τη δυνατότητα εφαρμογής μόνο σε ανοιχτούς χώρους.

Για αυτή τη μελέτη, χρησιμοποιήθηκαν πέντε κινητά ρομπότ όπως αυτό στο σχήμα 3.4. Κάθε ρομπότ εξοπλίστηκε με μία σειρά από έξι αισθητήρες sonar, εννέα infrared για να μετρούν απόσταση, δύο αισθητήρες που μετρούν την περιστροφή των τροχών, και δύο αισθητήρες ανίχνευσης κινήσεων. Επίσης, μία φωτογραφική μηχανή με ανάλυση 353x288 pixels και ταχύτητα λειτουργίας μέχρι και 15fps, μικρόφωνο και μεγάφωνο.

Κάθε ρομπότ έχει τη δική του διεύθυνση IP και ενσωματωμένη κάρτα Wi-Fi 802.11 ασύρματης επικοινωνίας. Με αυτό το τρόπο το ρομπότ μπορεί να διαβιβάζει όλα τα στοιχεία των αισθητήρων σε μία κεντρική μονάδα για επεξεργασία. Εντολές και στοιχεία μπορούν να σταλούν στα ρομπότ μέσω της ίδιας ασύρματης σύνδεσης για έλεγχο και πρόσβαση σε πραγματικό χρόνο. Σε αυτήν την μελέτη, τα ρομπότ προγραμματίστηκαν να κινούνται είτε αυτόνομα είτε καθοδηγούμενα από ένα χειριστή και να αποφεύγουν στατικά και κινητά εμπόδια.

Κεφάλαιο 4

Πιλοτικές εφαρμογές

4.1. Παρουσίαση υλικού

4.1.1. Lego Mindstorm NXT robot

Το NXT είναι ο εγκέφαλος του MINDSTORM robot. Αυτό είναι υπεύθυνο για τους υπολογισμούς και αποφάσεις του robot. Περιέχει:

- Τρεις θύρες όπου ενώνουμε κινητήρες – Θύρες A, B, C
- Τέσσερις θύρες όπου ενώνουμε αισθητήρες – Θύρες 1, 2, 3, 4
- Θύρα USB όπου ενώνουμε ένα ηλεκτρονικό υπολογιστή για να κατεβάσουμε τα προγράμματα μας στο NXT. Αυτό μπορεί να γίνει και ασύρματα μέσο Bluetooth.
- Μεγάφωνο για το όταν αναπαράγει αρχεία ήχου
- οθόνη LCD

- 32-bit ARM7 microcontroller
- 256 Kbytes FLASH, 64 Kbytes RAM
- 8-bit AVR microcontroller
- 4 Kbytes FLASH, 512 Byte RAM
- Bluetooth wireless communication (Bluetooth Class II V2.0 compliant)
- USB full speed port (12 Mbit/s)
- 4 input ports, 6-wire cable digital platform (One port includes a IEC 61158 Type 4/EN 50 170 compliant expansion port for future use)
- 3 output ports, 6-wire cable digital platform
- 100 x 64 pixel LCD graphical display
- Loudspeaker - 8 kHz sound quality.
- Power source: 6 AA batteries

Σχήμα 4.1: Lego Mindstorm NXT

4.1.2. MicaZ αισθητήρες

Συσκευές ασύρματων δικτύων αισθητήρων (motes) κατασκευάζονται από εταιρίες όπως την Intel, Crossbow, Dust Networks, Millennial Net, Arched Rock, Ember και άλλες. Για αυτή τη διπλωματική εργασία είχαμε πρόσβαση στους MicaZ motes μαζί με το programming board τους.

Τεχνητά χαρακτηριστικά των MicaZ:

- IEEE 802.15.4, tiny, wireless measurement system
- Designed specifically for deeply embedded sensor networks
- 250 kbps, high data rate radio
- Wireless communications with every node as router capability
- Expansion connector for light, temperature, RH, barometric pressure, acceleration / seismic, acoustic, magnetic, and other Crossbow sensor boards

Τεχνητά χαρακτηριστικά των MIB510 Programming board:

- Base station for wireless sensor networks via standard MICA2 processor radio board
- Serial port programming for all MICA hardware platforms

Σχήμα 4.2: Κόμβος MicaZ

4.2. Παρουσίαση λογισμικού

Παρ' όλη την εξέλιξη που είχαμε τα τελευταία χρόνια στους αισθητήρες, το λογισμικό είναι αυτό που έχει τον τελευταίο λόγο για να πετύχει ένα δίκτυο. Οι αλγόριθμοι και τα πρωτόκολλα που θα χρησιμοποιηθούν πρέπει να μεγιστοποιούν την διάρκεια ζωής του δικτύου, να μπορούν να αντιμετωπίσουν την καταστροφή αισθητήρων και να οργανώνουν το δίκτυο χωρίς εξωτερική βοήθεια.

Μερικά θέματα, όσο αφορά το λογισμικό, ακόμα υστερούν αρκετά και απασχολούν πολλούς μελετητές. Θέματα όπως την ασφάλεια των δεδομένων και η κινητικότητα των αισθητήρων και των βάσεων. Πιο κάτω βλέπουμε το λογισμικό που χρησιμοποιήθηκε στην περίπτωση μας για να υλοποιηθούν οι αλγόριθμοι του ρομπότ και οι επικοινωνίες μεταξύ του ρομπότ και του δικτύου ασύρματων αισθητήρων.

4.2.1. Eclipse

Eclipse είναι ένα περιβάλλον ανάπτυξης εφαρμογών. Υποστηρίζει ένα μεγάλο φάσμα από γλώσσες προγραμματισμού μέσο των plug-ins. Η κύρια γλώσσα προγραμματισμού που υποστηρίζει είναι η Java την οποία χρησιμοποιήσαμε για τον προγραμματισμό του Lego Mindstorm NXT. Χάρη στην επεκτασιμότητα του eclipse μπορέσαμε να εισάγουμε τις βιβλιοθήκες του ρομπότ και να το προγραμματίζουμε απευθείας από το περιβάλλον του eclipse.

Σχήμα 4.3: Λογότυπο του προγράμματος Eclipse

4.2.2. Lejos firmware

Το Lejos είναι εναλλακτική μέθοδος για να προγραμματίζεται το Lego Mindstorm NXT. Περιέχει Java virtual machine και επιτρέπει στο ρομπότ να δέχεται προγράμματα Java. Τα πιο κύρια χαρακτηριστικά του που μας ώθησαν στο να χρησιμοποιήσουμε αυτή τη μέθοδο αντί την βασική μέθοδο που έρχεται προ-εγκατεστημένη στο ρομπότ είναι τα εξής:

- Αντικειμενοστραφής γλώσσα προγραμματισμού (Java)
- Threads
- Πολυδιάστατους πίνακες
- Αναδρομή
- Συγχρονισμό
- Εξαιρέσεις
- Βοηθητικό API στο διαδίκτυο

Σχήμα 4.4: Λογότυπο Lejos

4.2.3. TinyOS

TinyOS είναι ένα open-source λειτουργικό σύστημα σχεδιασμένο ειδικά για embedded συστήματα με περιορισμένους πόρους όπως τους αισθητήρες MicaZ που χρησιμοποιούμε. Χρησιμοποιεί την γλώσσα προγραμματισμού NesC η οποία είναι μία επέκταση της C με παρόμοια σύνταξη. Ως embedded λειτουργικό σύστημα, είναι σχεδιασμένο να εκτελεί όλες τις λειτουργίες που απαιτούνται από ένα κόμβο μέσα σε ένα ασύρματο δίκτυο αισθητήρων χρησιμοποιώντας το λιγότερο δυνατό πόρους.

Σχήμα 4.5: Λογότυπο του λειτουργικού συστήματος TinyOS

4.3. Προκαταρκτικά υλοποίησης

4.3.1. Πειραματισμός με το Lego Mindstorm NXT

Ο προγραμματισμός του Lego NXT με Java απαιτούσε κάποια βήματα για να αλλάξουμε το λογισμικό του από το Mindstorm στο Lejos και να ρυθμίσουμε τον ηλεκτρονικό υπολογιστή για να αναγνωρίζει το Lego NXT και να το προγραμματίζουμε μέσο του Eclipse. Τα βήματα που ακολουθούν έγιναν σε ηλεκτρονικό υπολογιστή με λειτουργικό σύστημα Windows XP.

1. Εγκατάσταση της Java στον υπολογιστή

Η Java προσφέρεται δωρεάν από την Sun Microsystems. Κατεβάζουμε το Java SE (standard edition) JRE (Java Runtime Environment) από τη σελίδα της Sun και το εγκαθιστούμε στον υπολογιστή μας ακολουθώντας τα βήματα της εφαρμογής. Χρειαζόμαστε τουλάχιστο την έκδοση 5 του JRE

2. Εγκατάσταση του Lego NXT USB driver στον υπολογιστή

Το NXT μπορεί να ενωθεί στον υπολογιστή μέσο USB ή Bluetooth. Η επικοινωνία μέσο USB είναι πιο αξιόπιστη και πιο γρήγορη από αυτή του Bluetooth. Επιπλέον, το Bluetooth εξαρτάται και από το μοντέλο του υπολογιστή για να δουλέψει σωστά. Πρώτα εγκαθιστούμε το driver και στη συνέχεια μπορούμε να ενωθούμε με το NXT χρησιμοποιώντας το καλώδιο USB. Το USB driver είναι διαθέσιμο στην ιστοσελίδα της Mindstorm. Για να ελέγξουμε αν πέτυχε η εγκατάσταση βλέπουμε μέσα στο Device Manager αν υπάρχει το Lego NXT όταν το ενώσουμε.

3. Εγκατάσταση του Lejos στον υπολογιστή

Το Lejos είναι διαθέσιμο από την ιστοσελίδα sourceforge. Αφού κατεβάσουμε το συμπιεσμένο αρχείο (zip file) το αποσυμπιέζουμε σε ένα φάκελο μέσα στο σκληρό μας δίσκο. Στην περίπτωση μας ήταν `"C:\lejos_nxt\"`. Δεν απαιτεί κάποια εγκατάσταση

αλλά πρέπει να ενημερώσουμε τον υπολογιστή μας για την ύπαρξη του. Αυτό το επιτυγχάνουμε με την δημιουργία System variables. Δημιουργούμε την System variable "LEJOS_HOME" η οποία περιέχει το "C:\lejos_nxt\" και στη συνέχεια προσθέτουμε μέσα στην System variable "Path" το "%LEJOS_HOME%\bin". Τώρα είμαστε σε θέση να τρέξουμε εντολές του Lejos μέσα σε ένα Command prompt.

4. Εγκατάσταση του Libusb στον υπολογιστή

Αυτό το πρόγραμμα επιτρέπει στα Windows να έχουν πρόσβαση σε οποιαδήποτε συσκευή USB. Βρίσκετε μέσα στο φάκελο του lejos που εγκαταστήσαμε προηγουμένως στο " C:\lejos_nxt\3rdparty\lib\"

5. Εγκατάσταση του Lejos στο Lego NXT

Για να εγκαταστήσουμε το Lejos στο Lego NXT πρέπει πρώτα να σβήσουμε το ήδη εγκατεστημένο λογισμικό. Αυτό γίνετε με το πάτημα ενός (καλά κρυμμένου) κουμπιού στο πίσω μέρος του Lego NXT. Στη συνέχεια ενώνουμε το Lego NXT Με τον υπολογιστή μας μέσω ενός καλωδίου USB και εκτελούμε την εντολή "lejosfirmdl" μέσα σε ένα Command prompt. Όταν ολοκληρωθεί η διαδικασία το Lego NXT είναι έτοιμο να δεχτεί προγράμματα Java

6. Εγκατάσταση και ρύθμιση του Eclipse στον υπολογιστή

Το Eclipse όπως και το Lejos το βρίσκουμε σε μορφή συμπιεσμένου αρχείου (zip file) και το μόνο που απαιτεί είναι να το αποσυμπιέζουμε σε ένα φάκελο μέσα στο σκληρό μας δίσκο. Όταν τρέξουμε για πρώτη φορά το Eclipse δημιουργούμε το χώρο εργασίας όπου θα αποθηκεύονται όλα μας τα projects.

Όταν δημιουργήσουμε ένα καινούργιο project πρέπει να του ενσωματώσουμε τις βιβλιοθήκες του Lejos. Μέσα στις επιλογές των βιβλιοθηκών προσθέτουμε το αρχείο "C:\lejos_nxt\lib\classes.jar"

Για να κάνουμε compile ένα πρόγραμμα και να το κατεβάσουμε στο Lego NXT, ρυθμίζουμε τα External Tools που βρίσκονται κάτω από την επιλογή Run του μενού. Στο "Location" βάζουμε το "C:\lejos_nxt\bin\lejosdl.bat", μέσα στο "Working Directory" γράφουμε "${project_loc}\bin" και στα "Arguments" γράφουμε "${java_type_name}" Αποθηκεύουμε αυτές τις επιλογές και το εκτελούμε κάθε φορά που θέλουμε να κατεβάσουμε ένα πρόγραμμα στο Lego NXT.

Μετά από αυτά τα βήματα είμαστε σε θέση να γράψουμε προγράμματα σε Java μέσα στο Eclipse και να τα μεταγλωττίσουμε και να τα κατεβάσουμε στο Lego NXT από τον ίδιο τόπο. Για να ελέγξουμε την λειτουργικότητα αυτών των προγραμμάτων και ρυθμίσεων και επίσης για να εξοικειωθούμε με όλα αυτά και να γνωρίσουμε το API του Lejos, δημιουργήσαμε ένα απλό πρόγραμμα για το Lego NXT. Σκοπός του προγράμματος είναι το Lego NXT να ακολουθεί μία μαύρη γραμμή πάνω στο έδαφος μη λαμβάνοντας υπόψη τυχόν εμπόδια. Για να το πετύχουμε αυτό χρησιμοποιήσαμε ένα αισθητήρα φωτός ο οποίος ήταν στραμμένος προς τα κάτω όπως φαίνεται στο σχήμα 4.6.

Σχήμα 4.6: Ακολουθία γραμμής από το Lego NXT με τη χρήση ενός αισθητήρα φωτός

Αλγόριθμος εφαρμογής ακολουθία μαύρης γραμμής

Στροφή 360 μοίρες και διάβασμα τιμών αισθητήρα φωτός

Χαμηλότερη τιμή = Μαύρη γραμμή

Όσο η τιμή του αισθητήρα είναι χαμηλή

Προχώρα μπροστά

Αν η τιμή του αισθητήρα είναι μεγάλη

Σταμάτα και ψάξε δεξιά και αριστερά για πιο χαμηλή τιμή

Αν δεν βρεθεί, αύξησε τη γωνία ελέγχου

Αν βρεθεί συνέχισε μπροστά

Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα Α.1.

4.3.2. Πειραματισμός με αισθητήρες MicaZ και TinyOS

Το δεύτερο κομμάτι που χρειαζόταν να εγκατασταθεί και να ρυθμιστεί ήταν το λειτουργικό των αισθητήρων MicaZ, TinyOS. Ήταν απαραίτητη η χρήση τις έκδοσης 2.x για λόγους συμβατότητας με το Sensor board του MicaZ. Επίσης, επειδή το λειτουργικό TinyOS είναι συμβατό μόνο σε περιβάλλοντα Unix, χρειάζεται και την εγκατάσταση του Cygwin για να τρέχει μέσα στα Windows. Όπως και προηγουμένως, τα βήματα που ακολουθούν έγιναν σε ηλεκτρονικό υπολογιστή με λειτουργικό σύστημα Windows XP.

1. Εγκατάσταση του εργαλείου ανάπτυξης Java (Java Development Kit - JDK)

Η Java χρειάζεται για την επικοινωνία με τους κόμβους και βάσεις κόμβων οι οποίες είναι ενωμένες πάνω στον ηλεκτρονικό υπολογιστή. Το Java JDK το βρίσκουμε στη σελίδα της Sun. Από εκεί κατεβάζουμε ένα εκτελέσιμο αρχείο (.exe) και όταν το τρέξουμε και ακολουθήσουμε τις οδηγίες έχουμε το JDK πάνω στον υπολογιστή μας.

2. Εγκατάσταση του Cygwin

Όπως αναφέραμε και πριν το TinyOS λειτουργεί μόνο σε περιβάλλοντα Unix. Το Cygwin προσφέρει αυτό το περιβάλλον μέσα σε λειτουργικό Windows καθώς και όλα τα εργαλεία τα οποία χρησιμοποιεί το TinyOS όπως perl και shell scripts. Το Cygwin είναι διαθέσιμο σε εκτελέσιμο αρχείο από την σελίδα του.

3. Εγκατάσταση native compilers

Αφού εγκαταστήσουμε το Cygwin πρέπει να κατεβάσουμε αρκετά native compilers. Αυτά τα compilers δημιουργούν τον κατάλληλο κώδικα σε assembly για συσκευές όπως τους κόμβους αισθητήρων. Στην περίπτωση μας που χρησιμοποιούμε τους MicaZ πρέπει να κατεβάσουμε την σειρά εργαλείων Amtel AVR. Συγκεκριμένα είναι αρχεία .rpm για τα ακόλουθα εργαλεία: avr-binutils, avr-gcc, avr-libc, avarice, insight (avr-gdb).
Εγκαθιστούμε αυτά τα εργαλεία μέσω του Cygwin με την εντολή

```
"rpm –ivh ονομα_αρχείου.rpm"
```

4. Εγκατάσταση του nesC compiler

Ο προγραμματισμός των MicaZ γίνεται με την γλώσσα προγραμματισμού nesC. Εγκαθίσταται με τον ίδιο τρόπο όπως τα προηγούμενα εργαλεία. Κατεβάζουμε τα αρχεία rpm nesC και tinyos-tools και τα εγκαθιστούμε μέσω του Cygwin.

5. Εγκατάσταση του TinyOS source tree

Τώρα που έχουμε όλα τα εργαλεία εγκατεστημένα χρειάζεται μόνο να εγκαταστήσουμε το TinyOS source tree και να καθορίσουμε τα environment variables. Κατεβάζουμε το αντίστοιχο rpm του TinyOS ανάλογα με την έκδοση που θέλουμε και το εγκαθιστούμε μέσω του Cygwin. Στη συνέχεια καθορίζουμε τα πιο κάτω environment variables:

```
TOSROOT = "/opt/tinyos-2.x"
TOSDIR = "$TOSROOT/tos"
CLASSPATH = "C:\tinyos\cygwin\opt\tinyos-2.x\support\sdk\java\tinyos.jar;."
MAKERULES = "$TOSROOT/support/make/Makerules"
```

6. Εγκατάσταση του εργαλείου απεικόνισης Graphviz

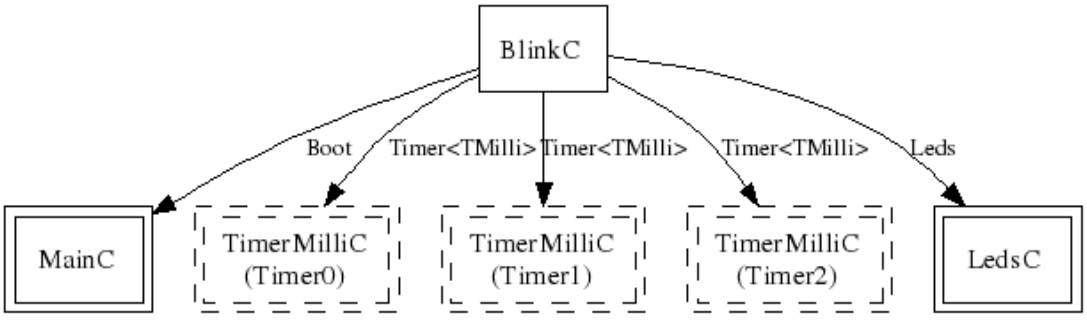
Μέσα στο περιβάλλων του TinyOS περιέχεται το εργαλείο "nesdoc" το οποίο αυτόματα δημιουργεί έγγραφα σε μορφή HTML απευθείας από πηγαίο κώδικα. Ένα μέρος αυτής τις διαδικασίας είναι η σχεδίαση διαγραμμάτων τα οποία δείχνουν σχέσεις μεταξύ διάφορων συστατικών του TinyOS. Το Graphviz είναι ένα εργαλείο ανοιχτού κώδικα το οποίο χρησιμοποιεί το "nesdoc" για αυτή την απεικόνιση. Για περιβάλλοντα Windows με Cygwin υπάρχει εκτελέσιμο αρχείο στη σελίδα του Graphviz το οποίο εγκαθιστά το εργαλείο.

Μετά την εγκατάσταση του TinyOS στον υπολογιστή χρησιμοποιήσαμε ένα απλό πρόγραμμα για να ελέγξουμε τη λειτουργικότητα του περιβάλλοντος, την συνδεσιμότητα με τους κόμβους MicaZ και για να εξοικειωθούμε με την γλώσσα προγραμματισμού nesC. Το πρόγραμμα που χρησιμοποιήσαμε είναι το Blink το οποίο βρίσκεται μέσα στα παραδείγματα του TinyOS. Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα C.1.

Σκοπός της εφαρμογής είναι να αναπαριστά ένα χρονόμετρο πάνω στα 3 LEDs που έχουν οι MicaZ. Τα LEDs αναβοσβήνουν στα 0.25Hz, 0.5Hz και 1Hz αντίστοιχα. Το αποτέλεσμα είναι να έχουμε μία δυαδική αναπαράσταση από το 0 μέχρι το 7 κάθε δύο δευτερόλεπτα.

Για να προγραμματίσουμε ένα κόμβο MicaZ πρέπει να τον ενώσουμε πάνω στο programming board MIB510 το οποίο είναι ενωμένο με το ρεύμα και τον υπολογιστή με DB9-serial-to-USB καλώδιο. Όταν προγραμματίζουμε ένα κόμβο, πρέπει να είναι σβηστός. Από το φάκελο της εφαρμογής Blink μέσα στο Cygwin εκτελούμε την εντολή `"make micaz install mib510,/dev/ttyS3"`

Όταν προγραμματίσουμε τους κόμβους MicaZ με την εφαρμογή Blink, τους ανάβουμε και, αν όλα πήγαν καλά, βλέπουμε τα LEDs των MicaZ να αναβοσβήνουν αναπαριστώντας δυαδικά χρονόμετρα από το μηδέν μέχρι το εφτά. Το εργαλείο nesdoc μας προσφέρει μία γραφική απεικόνιση τις εφαρμογής με την εντολή `"make platform docs"`



4.7: Γραφική απεικόνιση της εφαρμογής Blink από το nesdoc

4.3.3. Πειραματισμός με το πρωτόκολλο I2C

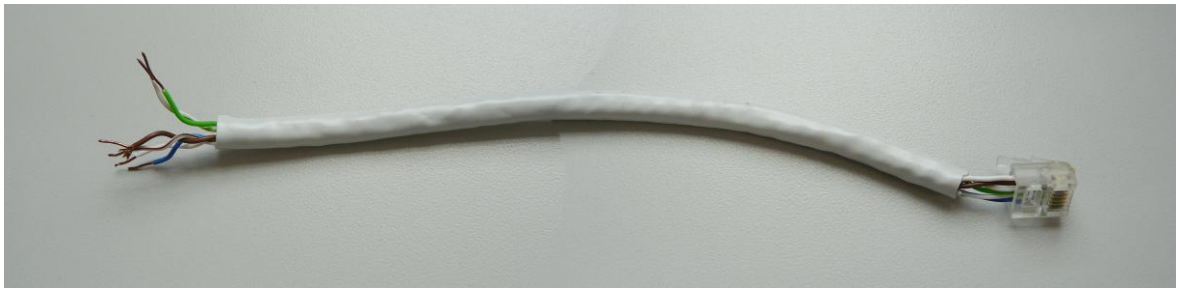
Μέχρι στιγμής μπορούμε να γράφουμε προγράμματα σε Java για το Lego NXT και προγράμματα σε nesC για τους κόμβους MicaZ. Πρέπει με κάποιο τρόπο να επικοινωνήσουν αυτά τα δύο κομμάτια. Μετά από αρκετή μελέτη βρέθηκε πως το μόνο πρωτόκολλο επικοινωνίας που υποστηρίζουν και τα δύο είναι το πρωτόκολλο I2C. Λόγω του ότι είναι αρκετά χαμηλού επιπέδου επικοινωνία χρειάστηκε να δημιουργήσουμε ειδικά καλώδια τα οποία θα μας έβγαζαν τις εξόδους και εισόδους που θέλαμε από το Lego NXT και τον κόμβο MicaZ.

- Δημιουργία καλωδίου NXT port to I2C

Για τη δημιουργία αυτού του καλωδίου, για να μην καταστρέψουμε τα καλώδια του Lego NXT, χρησιμοποιήσαμε ένα καλώδιο τηλεφώνου. Τροποποιήσαμε το socket για να ταιριάζει με αυτό του Lego NXT και ενώσαμε τα έξι καλώδια (pins) που έχει μέσα το καλώδιο τηλεφώνου. Για να ελέγξουμε το ρεύμα του κάθε χρωματιστού καλωδίου (pin) μέσα στο καλώδιο τηλεφώνου, μετρούμε το ρεύμα με ένα αμπερόμετρο του καθενός την ώρα που βρισκόταν ενωμένο στην πρώτη θήρα του αναμμένου Lego NXT. Πιο κάτω αναγράφονται οι μετρήσεις των καλωδίων και το είδος του κάθε pin..

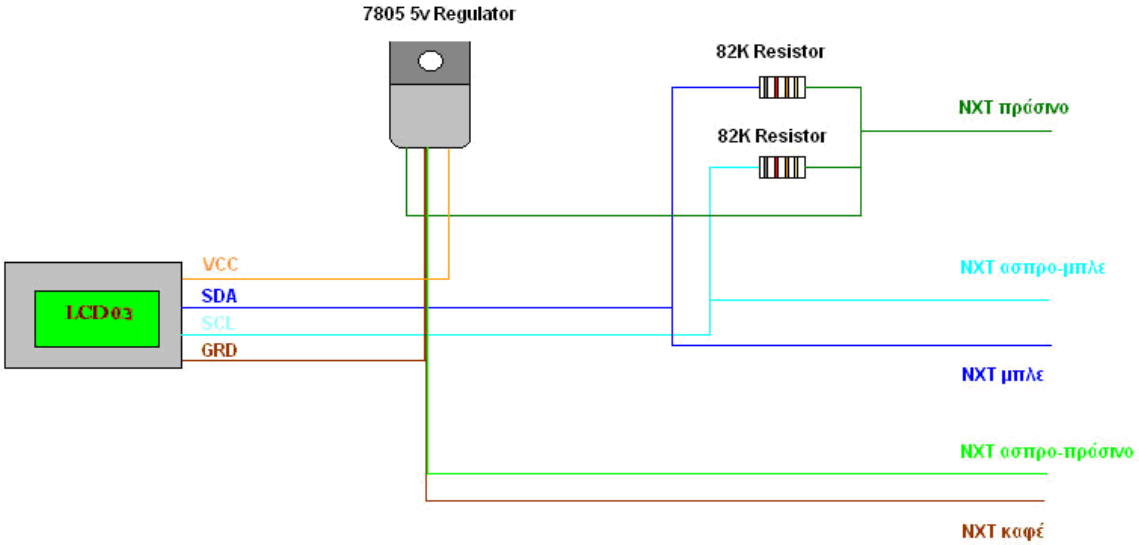
Αριθμός και χρώμα καλωδίου (pin)	**Μέτρηση ρεύματος**	**Είδος εξόδου**
Pin 1 (άσπρο/καφέ)	4.9V	VCC Ρεύμα
Pin2 (καφέ)	0V	GND Γείωση
Pin 3 (άσπρο/πράσινο)	0V	GND Γείωση
Pin 4 (πράσινο)	4.5V	VCC Ρεύμα
Pin 5 (άσπρο/μπλε)	0V	SCL Ρολόι
Pin 6 (μπλε)	0V	SDA Δεδομένα

Πίνακας 4.1: Μετρήσεις ρεύματος για τα pins του NXT-I2C καλωδίου



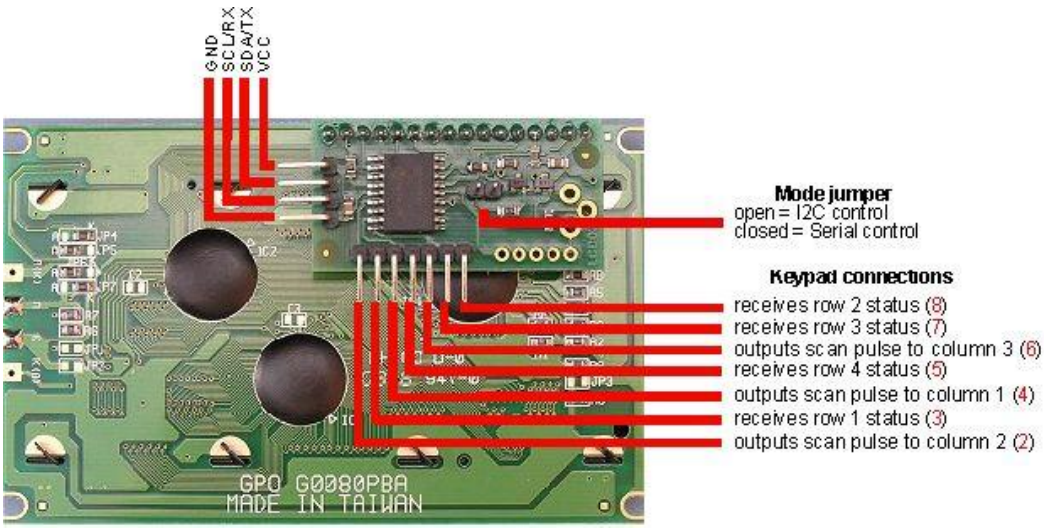
Σχήμα 4.8: Καλώδιο NXT-I2C

Για να δοκιμάσουμε το πρωτόκολλο I2C και τα καλώδια, ενώσαμε το Lego NXT με μία LCD03. Τις απαραίτητες ενώσεις τις κάναμε μέσω ενός breadboard. Πάνω στην LCD03, όπως φαίνεται στο σχήμα 4.10, χρησιμοποιήσαμε τις ενώσεις GND (γείωση), SCL/RX (ρολόι), SDA/TX (δεδομένα), VCC (ρεύμα) και το Mode jumper ήταν ανοιχτό για να λειτουργεί με το I2C. Από το Lego NXT πήραμε τα αντίστοιχα καλώδια για (γείωση), SCL/RX (ρολόι), SDA/TX (δεδομένα), VCC (ρεύμα) και τα ενώσαμε ως εξής:



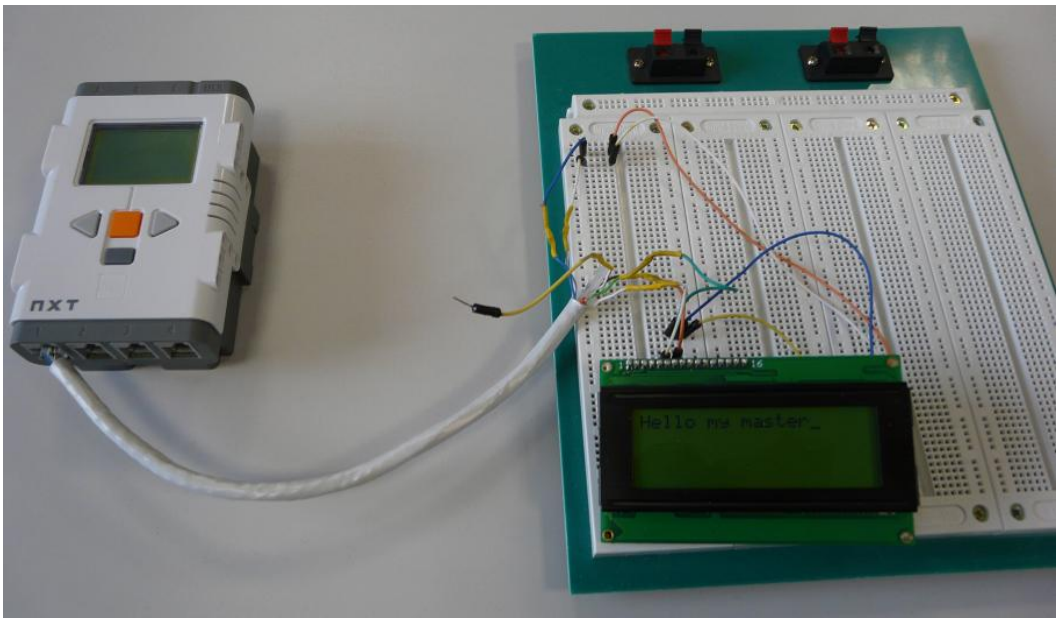
Σχήμα 4.9: Συνδέσεις μεταξύ Lego NXT και LCD03

Ο λόγος που χρησιμοποιήσαμε το πράσινο καλώδιο για VCC αντί του άσπρο/καφέ, είναι γιατί με την χρήση τις σταθεράς TYPE_LOWSPEED_9V μέσα στο πρόγραμμα το άσπρο/καφέ καλώδιο βγάζει 9V. Αυτό μπορεί να δημιουργήσει πρόβλημα στην LCD.



Σχήμα 4.10: Ενώσεις του LCD03 στο πίσω μέρος [20]

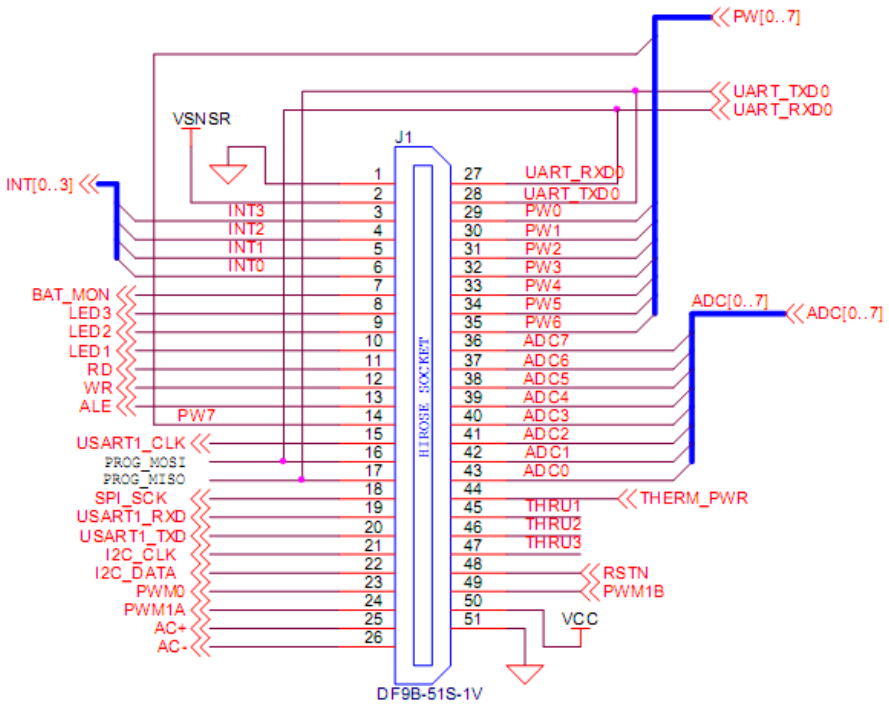
Μετά την ένωση τρέχουμε ένα πρόγραμμα Java πάνω στο Lego NXT το οποίο στέλνει ένα-ένα χαρακτήρα στην οθόνη μέσω του I2C και εμφανίζει το μήνυμα πάνω στην οθόνη του LCD03. Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα Α.2.



Σχήμα 4.11: Εφαρμογή για επικοινωνία μεταξύ Lego NXTκαι LCD03 μέσω I2C

- Δημιουργία καλωδίου MicaZ to I2C

Στη περίπτωση του MicaZ η δημιουργία καλωδίου I2C ήταν σχετικά πιο εύκολη αλλά απαιτούσε αρκετή ακρίβεια στις ενώσεις. Εδώ το μόνο που χρειαζόμασταν ήταν να βρούμε τα κατάλληλα pin για I2C CLK (ρολόι), I2C DATA (δεδομένα) VCC (ρεύμα) και GND (γείωση) και να ενώσουμε σύρματα πάνω τους με την χρήση καλάι. Όπως φαίνεται στο σχήμα 4.12 είναι τα pins 21, 22, 50 και 51 αντίστοιχα.



Σχήμα 4.12: Διάγραμμα εξόδων από το 51-pin expansion slot του MicaZ [23]

4.4. Υλοποίηση εφαρμογών

4.4.1. Κινητικότητα

4.4.1.1. Κίνηση προς ένα αισθητήρα

Η πρώτη εφαρμογή που δημιουργήσαμε εμπίπτει στην κατηγορία της κινητικότητας. Είναι το βασικό πρόγραμμα κίνησης το οποίο θα χρησιμοποιείται στη συνέχεια από της υπόλοιπες εφαρμογές.

Αλγόριθμος εφαρμογής

Διάβασε τις συντεταγμένες του προορισμού
Βάση των συντεταγμένων και κατεύθυνση του ρομπότ
    Υπολόγισε την γωνία και απόσταση μέχρι τον προορισμό
Ξεκίνα να προχωράς ευθεία προς τον προορισμό
    Αν βρεθεί εμπόδιο
        Ψάξε δεξιά και αριστερά για τυχόν άλλα εμπόδια
        Απόφυγε το εμπόδιο από την πιο καθαρή πλευρά
        Συνέχισε προς τον προορισμό

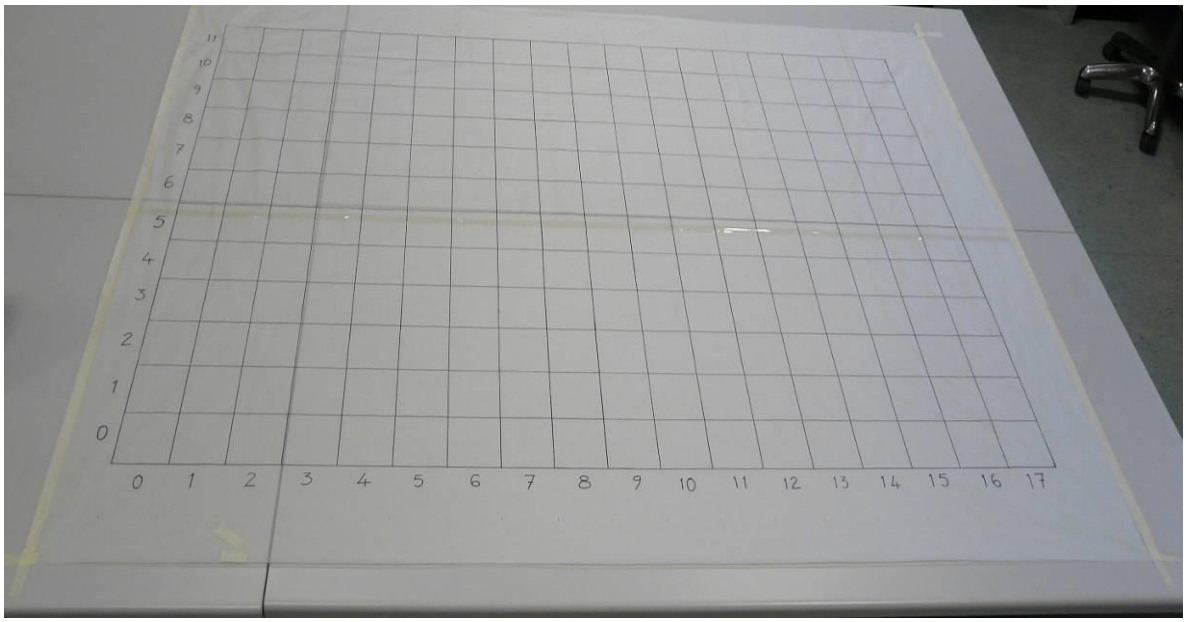
Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα Α.3.

Επεξήγηση αλγόριθμου

Το Lego NXT βρίσκεται σε μία θέση μέσα στο δίκτυο, γνωστή από προηγουμένως. Για παράδειγμα στο (0,0) και φορά κατεύθυνσης προς τον θετικό άξονα ψ. Λόγο του ότι δεν υλοποιούμε αλγόριθμο καθορισμού θέσης μέσα στο δίκτυο, δεν είναι δυνατή η εκκίνηση από ένα τυχαίο σημείο. Το Lego NXT λαμβάνει ως είσοδο τις συντεταγμένες προορισμού. Είτε προέρχονται από την εφαρμογή έλεγχου κάλυψης, είτε από τον εντοπισμό στόχου.

Στη συνέχεια έχουμε δύο επιλογές για τη δημιουργία του μονοπατιού. Προκαθορισμένο μονοπάτι σε περίπτωση που το Lego NXT γνωρίζει από προηγουμένως τις θέσεις όλων των MicaZ μέσα στο δίκτυο και δυναμική κίνηση σε περίπτωση πού δεν γνωρίζει για την ύπαρξη άλλων αισθητήρων. Εμείς υλοποιήσαμε την δεύτερη περίπτωση η οποία μας προσφέρει μια πιο ρεαλιστική λύση μιας και μέσα σε ένα δίκτυο ασύρματων αισθητήρων γίνονται συχνά αλλαγές στην τοπολογία και υπάρχουν φυσικά εμπόδια εκτός από τους αισθητήρες καθεαυτό.

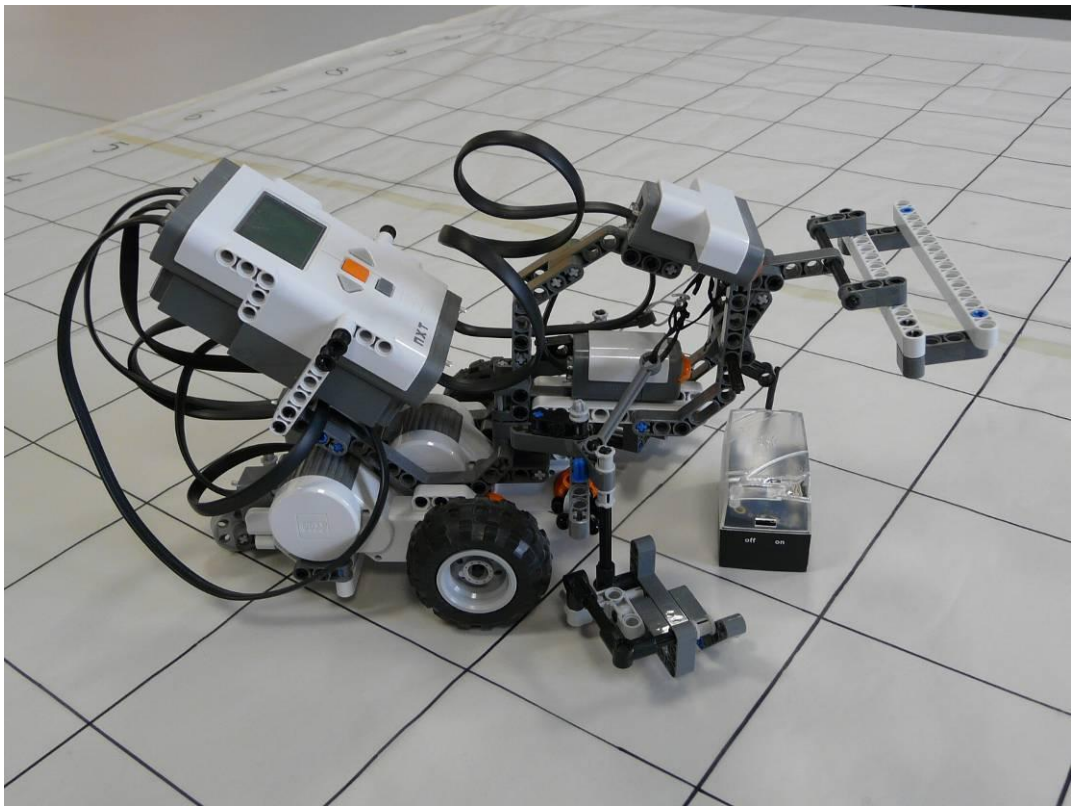
Το Lego NXT υπολογίζει την γωνία και απόσταση την οποία πρέπει να καλύψει για να φτάσει στον προορισμό του και ξεκινά να κινείται προς εκεί παρατηρώντας ταυτόχρονα για τυχών εμπόδια στο δρόμο με τους αισθητήρες επαφής και ultrasonic. Σε περίπτωση που βρει εμπόδιο το αποφεύγει και συνεχίζει πάλι προς τον προορισμό του. Για να γνωρίζει πού βρίσκεται κάθε στιγμή, μετρά συνεχώς την περιστροφή των τροχών του και υπολογίζει την θέση του σε σχέση με την αρχική.



Σχήμα 4.13: Επιφάνεια δοκιμών. Κάθε τετράγωνο έχει διαστάσεις 10x10 cm

Αποτελέσματα

Για να ελέγξουμε την ορθότητα του αλγορίθμου και τον τρόπο που αντιμετωπίζει την εύρεση εμποδίων βάλαμε το Lego NXT να μεταφέρει ένα κόμβο MicaZ σε μία συγκεκριμένη τοποθεσία την οποία υπολογίσαμε από πριν. Μέσα στην περιοχή τοποθετήσαμε και εμπόδια. Αφού τρέξαμε την εφαρμογή αρκετές φορές αλλάζοντας κάθε φορά τον προορισμό καταγράψαμε θετικά αποτελέσματα αλλά όχι αρκετή ακρίβεια. Αυτό οφείλεται στο ότι το Lego NXT δεν μας προσφέρει αρκετή ακρίβεια στις κινήσεις του και στις μετρήσεις της περιστροφής των τροχών. Επειδή τη θέση του την υπολογίζει βάση αυτών των μετρήσεων, κάθε λάθος που κάνει προστίθεται στα προηγούμενα με αποτέλεσμα να νομίζει ότι βρίσκεται σε ένα σημείο την στιγμή που είναι σε άλλο. Το πρόβλημα αυξάνεται όταν εξασθενήσουν λίγο οι μπαταρίες του.



Σχήμα 4.14: Εφαρμογή κίνησης προς ένα σημείο και τοποθέτηση αισθητήρα

4.4.1.2. Καθορισμός μονοπατιού προς όλους τους αισθητήρες

Η δεύτερη εφαρμογή πάλι είναι μέσα στην κατηγορία της κινητικότητας. Σε αυτή την περίπτωση το Lego NXT θα λαμβάνει ως είσοδο τις συντεταγμένες όλων των κόμβων MicaZ μέσα στο δίκτυο και να υπολογίζει το συντομότερο μονοπάτι για να επισκεφτεί όλους τους κόμβους. Το Lego NXT θα λαμβάνει τις πληροφορίες αυτές από τον κόμβο MicaZ ο οποίος θα βρίσκεται πάνω στο Lego NXT ενωμένο με I2C και θα συμπεριφέρεται ως root για να λαμβάνει τις πληροφορίες από τους υπόλοιπους κόμβους μέσα στο δίκτυο.

Αλγόριθμος εφαρμογής

Διάβασε όλες τις τοποθεσίες των αισθητήρων
Υπολόγισε τα κόστη (αποστάσεις) μεταξύ όλων των αισθητήρων και του ρομπότ
Υπολόγισε το πιο σύντομο μονοπάτι (δύο τρόποι)

1. Πρόσθεσε τη θέση του ρομπότ στη λίστα
   Επέλεξε τον πιο κοντινό αισθητήρα από την τελευταία θέση του ρομπότ και πρόσθεσε τον στη λίστα
      Καθόρισε τις καινούργιες συντεταγμένες ως η θέση του ρομπότ
      Επανέλαβε μέχρι να μπουν όλοι οι αισθητήρες στη λίστα
   Πρόσθεσε την αρχική θέση του ρομπότ στο τέλος της λίστας
2. Δημιούργησε όλα τα δυνατά μονοπάτια που ξεκινούν και τελειώνουν στη θέση του ρομπότ και περνούν από όλους τους αισθητήρες
   Υπολόγισε το συνολικό κόστος του κάθε μονοπατιού
   Επέλεξε το μονοπάτι με το πιο μικρό κόστος

Κάλεσε την εφαρμογή κίνηση προς ένα σημείο με παραμέτρους την τωρινή θέση του ρομπότ και στόχο τον επόμενο αισθητήρα μέσα στη λίστα μέχρι να συμπληρωθεί το μονοπάτι.

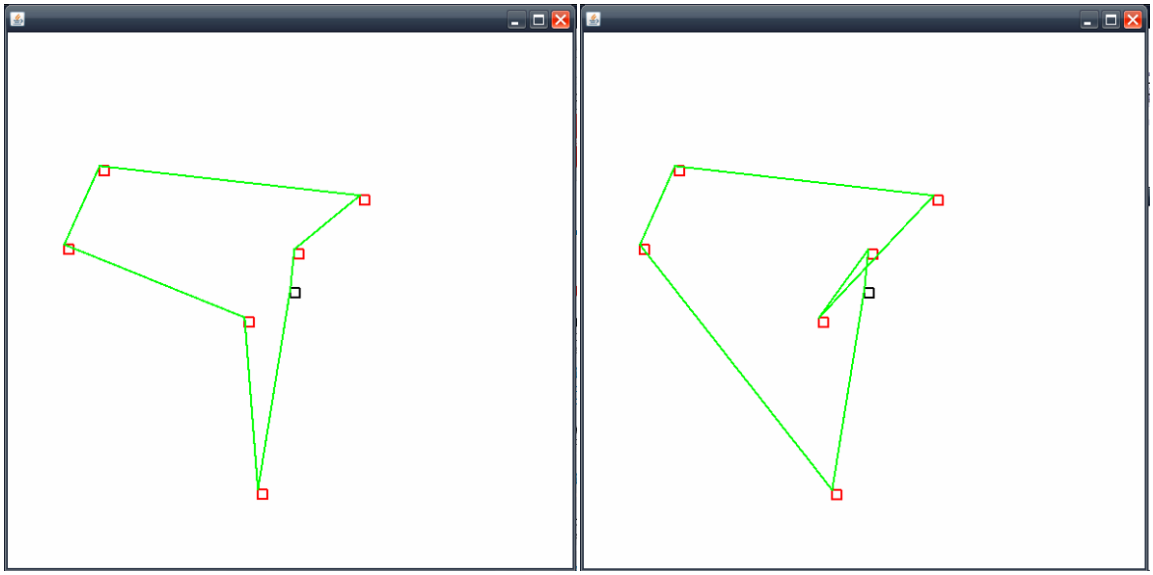
Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα Α.4.

Επεξήγηση αλγόριθμου

Η συλλογή των πληροφοριών μπορεί να γίνει με δύο τρόπους. Να γίνει στην αρχή η συλλογή των συντεταγμένων από όλους τους κόμβους και να υπολογιστεί το συντομότερο μονοπάτι. Αλλιώς μπορεί να γίνει σταδιακά η συλλογή μόνο από γειτονικούς κόμβους κάθε φορά και να επιλέγει τον πιο κοντινό κόμβο για να πάει προς τα εκεί. Η πρώτη επιλογή είναι πιο ορθή στον υπολογισμό του συντομότερου μονοπατιού ενώ η δεύτερη είναι πιο γρήγορη. Εμείς υλοποιήσαμε και τους δύο πιο πάνω τρόπους.

Και στις δύο περιπτώσεις η συλλογή γίνεται στην αρχή από όλους τους κόμβους μέσα στο δίκτυο. Αυτό μας εξασφαλίζει ότι δεν θα αγνοηθούν τυχόν απομακρυσμένοι κόμβοι. Στη συνέχεια χρησιμοποιούμε το γνωστό πρόβλημα του πλανόδιου πωλητή (Travelling Salesman Problem) για να υπολογίσουμε το συντομότερο μονοπάτι. Ο πρώτος τρόπος βρίσκει κάθε φορά τον πιο κοντινό κόμβο από την τελευταία θέση του ρομπότ και τον προσθέτει μέσα σε μία σειρά. Αυτό επαναλαμβάνεται μέχρι να μπουν όλοι οι κόμβοι μέσα στη σειρά. Με τον δεύτερο τρόπο υπολογίζονται όλα τα δυνατά μονοπάτια και επιλέγουμε το πιο σύντομο.

Στη συνέχεια καλείται η προηγούμενη εφαρμογή η οποία κινεί το Lego NXT προς συγκεκριμένες συντεταγμένες ξεκινώντας από την αρχή της σειράς μέχρι να φτάσει στον τελευταίο κόμβο και να επιστρέψει στην αρχική του θέση. Αυτή η εφαρμογή μπορεί να εκτελείται περιοδικά για τον έλεγχο του δικτύου.



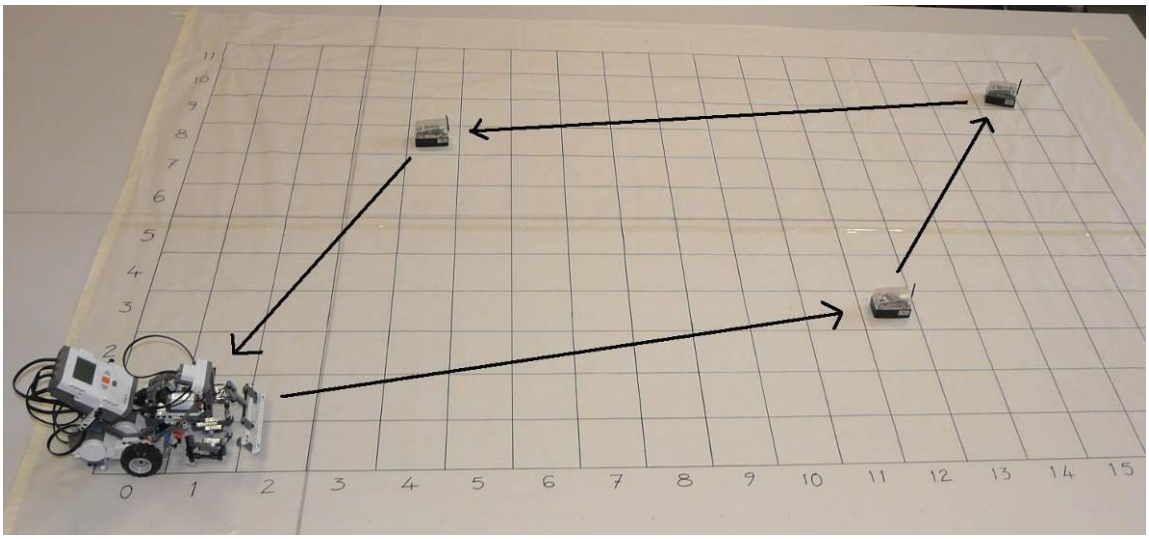
(α) (β)

Σχήμα 4.15: Γραφική απεικόνιση του καθορισμού μονοπατιού προς όλους τους κόμβους
(α) Υπολογισμός όλων των δυνατών μονοπατιών
(β) Επιλογή πλησιέστερου γειτονικού κόμβου

Αποτελέσματα

Για σκοπούς ελέγχου δημιουργήσαμε ένα πρόγραμμα Java το οποίο δημιουργεί ένα αρχείο κειμένου στο οποίο αναγράφονται οι τοποθεσίες όλων των αισθητήρων. Αυτές οι τοποθεσίες επιλέγονται τυχαία αλλά βρίσκονται μέσα στην ελεγμένη περιοχή. Ο κώδικας της εφαρμογής για την τοποθέτηση των αισθητήρων βρίσκεται στο Παράρτημα Β.1. Στη συνέχεια υπολογίζουμε το συντομότερο μονοπάτι και το απεικονίζουμε γραφικά όπως φαίνεται στο σχήμα 4.15. Αφού τρέξαμε την εφαρμογή αρκετές φορές και συγκρίναμε τους δύο τρόπους, παρατηρήσαμε τις περιπτώσεις που υστερεί ο καθένας. Ο κώδικας της εφαρμογής για απεικόνιση του πρώτου τρόπου βρίσκεται στο Παράρτημα Β.3.1 και του δεύτερου στο Παράρτημα Β.3.2.

Ο πρώτος τρόπος που επιλέγει πάντα τον πιο κοντινό γείτονα υστερεί σε ακρίβεια. Σε ορισμένες περιπτώσεις, όπως αυτή στο σχήμα 4.15, δεν βρίσκει το συντομότερο μονοπάτι. Ο μεν δεύτερος τρόπος, μπορεί να βρει το συντομότερο μονοπάτι αλλά υστερεί στο γεγονός ότι απαιτεί πάρα πολύ μνήμη αποτρέποντας το να βρίσκει μονοπάτια όταν έχουμε πολλούς αισθητήρες. Αυτό οφείλεται στο ότι υπάρχουν n! συνδυασμοί (όπου n ο αριθμός των αισθητήρων) σε αντίθεση με τον πρώτο τρόπο που βρίσκει μόνο ένα μονοπάτι σε n*n χρόνο.



Σχήμα 4.16: Εφαρμογή καθορισμού μονοπατιού προς όλους τους κόμβους

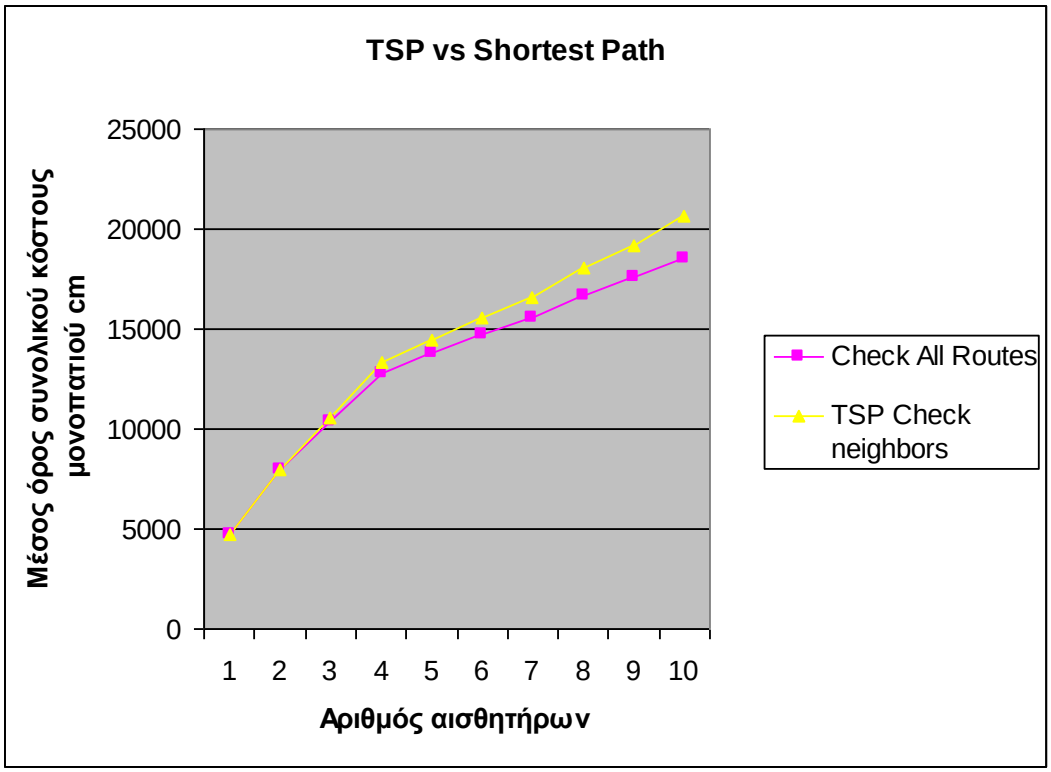
Αφού είδαμε γραφικά τι γίνεται, δοκιμάσαμε τους αλγόριθμους πρακτικά. Τοποθετήσαμε τους κόμβους MicaZ σε γνωστές τοποθεσίες όπως φαίνεται στο σχήμα 4.16 και τις περάσαμε στο Lego NXT. Αυτό με την σειρά του υπολόγισε το συντομότερο μονοπάτι ξεκίνησε να το διανύει. Το πρόβλημα που αντιμετωπίσαμε εδώ είναι το ότι το Lego NXT προσπαθούσε να πάει στις συντεταγμένες του κάθε κόμβου με αποτέλεσμα να το βρίσκει σαν εμπόδιο και να εκτελεί επιπλέον αχρείαστες κινήσεις για να τα αποφύγει.

Προσπαθήσαμε να το διορθώσουμε λέγοντας του να σταματά σε κάποια απόσταση από τους κόμβους. Για παράδειγμα να σταματά όταν μπει μέσα στην εμβέλεια του κόμβου. Αυτό βοήθησε αρκετά εκτός από τις περιπτώσεις όπου ο πιο κοντινός κόμβος είναι σε ευθεία γραμμή. Και να σταματήσει μέσα στην εμβέλεια του πρώτου κόμβου, πάλι θα τον βρει σαν εμπόδιο στην προσπάθεια του να πάει στον επόμενο κόμβο.

Σύγκριση αλγορίθμων

Για να συγκρίνουμε τους δύο αλγορίθμους τους τρέξαμε αρκετές φορές αλλάζοντας τον αριθμό των αισθητήρων μέσα στο δίκτυο με ελεγχόμενη περιοχή μεγέθους 680x440 cm. Κάθε φορά μετρούσαμε το συνολικό κόστος του μονοπατιού που επέλεγε ο κάθε αλγόριθμος ως το πιο σύντομο. Στη γραφική παράσταση φαίνεται η διαφορά στο κόστος μεταξύ των δύο αλγορίθμων. Λόγο του ότι ο αλγόριθμος που ελέγχει όλα τα δυνατά μονοπάτια απαιτούσε πολύ μνήμη, περιοριστήκαμε μέχρι τους 10 αισθητήρες.

Όπως φαίνεται στη γραφική παράσταση, ακόμα και για τόσο μικρό αριθμό από αισθητήρες, ο αλγόριθμος του πλανόδιου πωλητή που επιλέγει τον πιο κοντινό γείτονα, έχει μεγαλύτερο κόστος από το πιο σύντομο μονοπάτι. Η διαφορά στο κόστος θα αυξάνεται καθώς αυξάνεται ο αριθμός αισθητήρων μέσα στο δίκτυο.



Σχήμα 4.17: Γραφική παράσταση σύγκρισης κόστους αλγορίθμων για εύρεση του πιο σύντομου μονοπατιού

4.4.2. Έλεγχος κάλυψης

Η εφαρμογή αυτή είναι υπεύθυνη στο να εντοπίζει τυχόν κενά κάλυψης από τους αισθητήρες μέσα στο δίκτυο. Το Lego NXT δέχεται ως είσοδο τις συντεταγμένες όλων των κόμβων μέσα στο δίκτυο και εκτελεί τον αλγόριθμο ελέγχου κάλυψης που υλοποιήσαμε. Στη συνέχεια, αν βρεθεί κενό, καλούμε την πρώτη εφαρμογή με της συντεταγμένες του κενού για να πάει μέχρι εκεί και να αφήσει ένα επιπλέον κόμβο MicaZ.

Αλγόριθμος εφαρμογής

Διάβασε όλες τις τοποθεσίες των αισθητήρων
Για κάθε αισθητήρα του οποίου η εμβέλεια μπαίνει μέσα στην περιοχή την οποία ελέγχουμε
    Αφαίρεσε το εμβαδό της κάλυψης του αισθητήρα η οποία είναι μέσα στην περιοχή από το ολικό εμβαδό της περιοχής
Αν το ακάλυπτο κομμάτι που έμεινε είναι μεγαλύτερο από το ελάχιστο επιτρεπτό όριο ακάλυπτης περιοχής
    Χώρισε την περιοχή στα 4 και επανέλαβε τον έλεγχο για κάθε κομμάτι όσο η περιοχή έχει διαστάσεις μεγαλύτερες από το ελάχιστο όριο διαστάσεων της ελεγχόμενης περιοχής
Αφού βρεθούν όλα τα κενά στην περιοχή
Υπολόγισε σε πιο σημείο πρέπει να τοποθετηθεί ο επόμενος αισθητήρας (δύο τρόποι)

1. Βάση των συντεταγμένων του ρομπότ
    Υπολόγισε την απόσταση του κάθε κενού από το ρομπότ
    Επέλεξε το κενό πιο κοντά στο ρομπότ
2. Υπολόγισε πόσα γειτονικά κενά έχει το κάθε κενό
    Βάση των συντεταγμένων του κενού που ελέγχουμε
        Υπολόγισε πόσα κενά έχει δίπλα του
        Αν υπάρχουν: Υπολόγισε πόσα κενά υπάρχουν δίπλα από αυτά.
    Επέλεξε το κενό με τα περισσότερα γειτονικά κενά

Κάλεσε την εφαρμογή κίνηση προς ένα σημείο με παραμέτρους την τωρινή θέση του

ρομπότ και στόχο το σημείο του κενού όπου θα αφήσει τον αισθητήρα και στη συνέχεια να επιστρέψει στην αρχική του θέση.

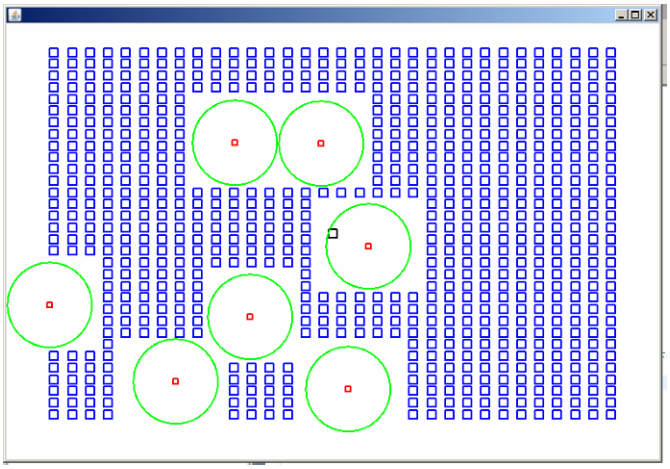
Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα Β.2

Επεξήγηση αλγόριθμου

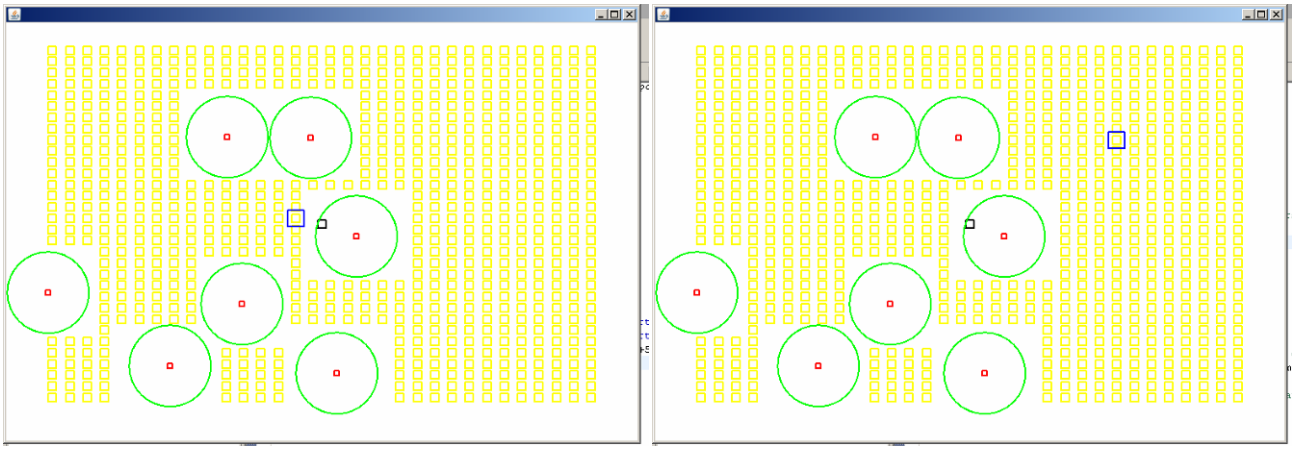
Για να δουλέψει σωστά ο αλγόριθμος πρέπει το Lego NXT να γνωρίζει ορισμένα πράγματα εκ των προτέρων. Τις διαστάσεις της ελεγχόμενης περιοχής, την εμβέλεια των κόμβων αισθητήρων και το ελάχιστο ποσοστό κάλυψης για το οποίο θεωρούμε μία περιοχή ως καλυμμένη.

Η μέθοδος που ελέγχουμε μία περιοχή αν είναι καλυμμένη είναι η εξής. Αφού γνωρίζουμε τις συντεταγμένες τις περιοχής και των αισθητήρων και την εμβέλεια τους, όσες περιοχές απέχουν από όλους τους αισθητήρες απόσταση μεγαλύτερη από την εμβέλεια των αισθητήρων, τότε είναι ακάλυπτες.

Ο αλγόριθμος λειτουργεί με αναδρομή. Κάθε φορά ελέγχουμε μια περιοχή με διαστάσεις τις οποίες προκαθορίζουμε. Για παράδειγμα 10x10 cm. Στην αρχή καλείται η συνάρτηση για την περιοχή η οποία ξεκινά από το (0, 0) και ελέγχεται η κάλυψη της. Αφού βρεθεί η κάλυψη της περιοχής, προχώρα στην διπλανή περιοχή και συνεχίζεται μέχρι να ελεγχθούν όλες. Όπως φαίνεται στο σχήμα 4.17, με την κλήση αυτής της μεθόδου μπορούν να εντοπιστούν πολλά κενά μέσα στην ελεγχόμενη περιοχή. Βάση όλων των ακάλυπτων περιοχών υπολογίζουμε αυτή που είναι η πιο κρίσιμα ακάλυπτη για να τοποθετήσουμε εκεί τον επιπλέον αισθητήρα. Δηλαδή την περιοχή που γύρο της έχει τις περισσότερες ακάλυπτες περιοχές, είτε την ακάλυπτη περιοχή που βρίσκεται πιο κοντά σε ένα προκαθορισμένο σημείο. Στην περίπτωση μας είναι η θέση του ρομπότ.



(α)



(β) (γ)

Σχήμα 4.17: Γραφική απεικόνιση του έλεγχου κάλυψης

(α) Εντοπισμός όλων των ακάλυπτων περιοχών (μπλε = ακάλυπτη περιοχή)

(β) Υπολογισμός της πιο κοντινής στο ρομπότ ακάλυπτης περιοχής

(γ) Υπολογισμός της πιο κρίσιμα ακάλυπτης περιοχής

(κίτρινο = ακάλυπτη περιοχή, μπλε = επόμενη τοποθέτηση αισθητήρα)

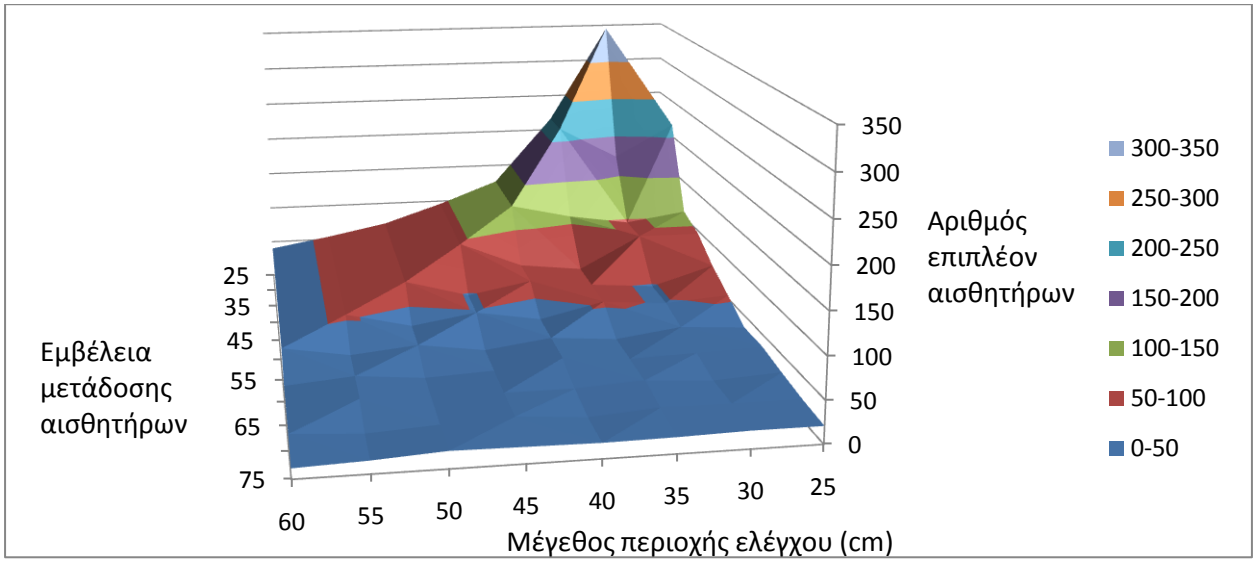
Αποτελέσματα

Για να ελέγξουμε αυτή την εφαρμογή δεν την κατεβάσαμε πάνω στο Lego NXT. Αντί αυτού, τρέξαμε την εφαρμογή μέσα στο Eclipse σαν εφαρμογή Java και προσθέσαμε κώδικα ο οποίος απεικονίζει το αποτέλεσμα της εφαρμογής όπως φαίνεται πιο πάνω στο σχήμα 4.17. Τρέξαμε το πρόγραμμα με διάφορες τοπολογίες και αριθμό από αισθητήρες.

Τα αποτελέσματα ήταν αρκετά θετικά. Ο κώδικας της εφαρμογής για απεικόνιση της εύρεσης κενού βρίσκεται στο Παράρτημα Β.2.5

Σύγκριση αλγορίθμων

Για να συγκρίνουμε τους δύο αλγορίθμους τους τρέξαμε αρκετές φορές αλλάζοντας την εμβέλεια των αισθητήρων μέσα στο δίκτυο με ελεγχόμενη περιοχή μεγέθους 340x220 cm και αρχικά κενή. Ο κάθε αλγόριθμος έτρεχε μέχρι να καλυφθούν όλα τα κενά μέσα στην περιοχή. Σε κάθε περίπτωση μετρούσαμε τον αριθμό αισθητήρων που χρειάστηκε.



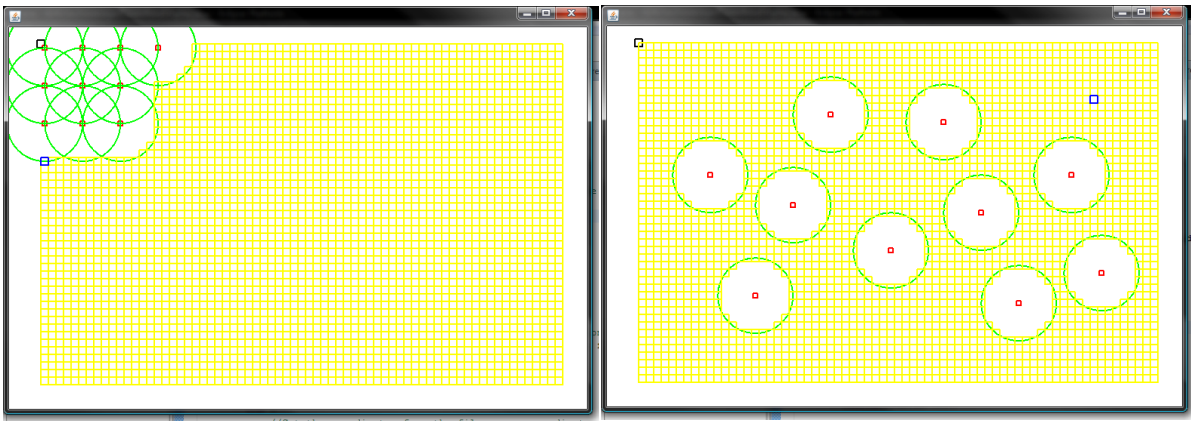
Σχήμα 4.18: Γραφική παράσταση επίδοσης αλγορίθμων για την κάλυψη των κενών μίας περιοχής

Όπως παρατηρήσαμε από τις μετρήσεις και οι δύο αλγόριθμοι είχαν την ίδια επίδοση όσο αφορά επιπλέον αισθητήρες. Όσο πιο μικρή είναι η περιοχή ελέγχου και η εμβέλεια των αισθητήρων, τόσο αυξάνεται και ο αριθμός αισθητήρων που προσθέτουμε.

Αν προσέξουμε όμως την γραφική αναπαράσταση των δύο αλγορίθμων στα σχήματα 4.19, βλέπουμε ότι το πλεονέκτημα του δεύτερου αλγορίθμου είναι ότι ανα πάσα στιγμή έχουμε μερική κάλυψη όλης της περιοχής σε αντίθεση με τον πρώτο αλγόριθμο που έχουμε ολική κάλυψη κοντά στο ρομπότ και καθόλου στην υπόλοιπη περιοχή.

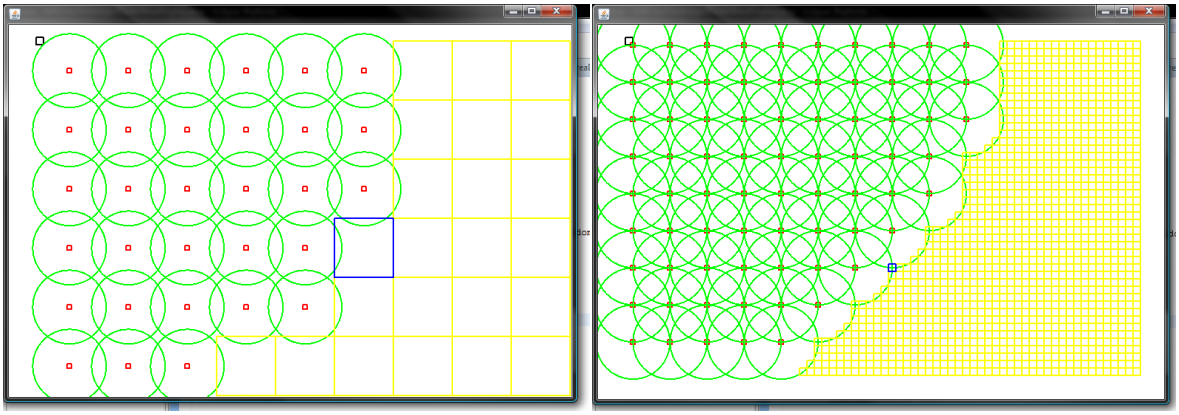
Στο σχήμα 4.20, βλέπουμε ότι όσο πιο μεγάλη είναι η περιοχή ελέγχου τόσο πιο λίγους αισθητήρες προσθέτουμε και έχουμε λιγότερη υπερκάλυψη. Στην περίπτωση που η περιοχή ελέγχου είναι μεγαλύτερη από την εμβέλεια του αισθητήρα, τότε δεν έχουμε καθόλου υπερκάλυψη.

Ο κώδικας της γραφικής αναπαράστασης της κάλυψης των κενών βρίσκεται στο Παράρτημα Β.2.6.



(α) (β)

Σχήμα 4.19: Γραφική αναπαράσταση για την κάλυψη των κενών μίας περιοχής βάση (α) ενός συγκεκριμένου σημείου (θέση του ρομπότ) και (β) της πιο ακάλυπτης περιοχής μετά από την τοποθέτηση 10 αισθητήρων



(α) (β)

Σχήμα 4.20: Γραφική αναπαράσταση για την κάλυψη των κενών μίας περιοχής βάση ενός συγκεκριμένου σημείου (θέση του ρομπότ) με περιοχή ελέγχου
(α) 80x80 cm και (β) 10x10 cm

4.4.3. Εντοπισμός στόχου

4.4.3.1. Εντοπισμός φωτεινού στόχου από το ρομπότ

Σε αυτή την εφαρμογή υλοποιήσαμε τον εντοπισμό στόχου ανεξάρτητα από το ασύρματο δίκτυο αισθητήρων. Το Lego NXT με τη χρήση δύο αισθητήρων φωτός ψάχνει για να βρει μία πηγή φωτός και την ακολουθεί. Είναι παρόμοιο σκεπτικό με την πειραματική εφαρμογή που δημιουργήσαμε η οποία ακολουθούσε μία μαύρη γραμμή. Εδώ οι αισθητήρες είναι στραμμένοι προς τα εμπρός για να έχει μεγαλύτερη εμβέλεια και αυτό που κυνηγούμε είναι τη μεγαλύτερη μέτρηση τους σε αντίθεση με την ακολουθία γραμμής που έψαχνε για την πιο χαμηλή.

Αλγόριθμος εφαρμογής

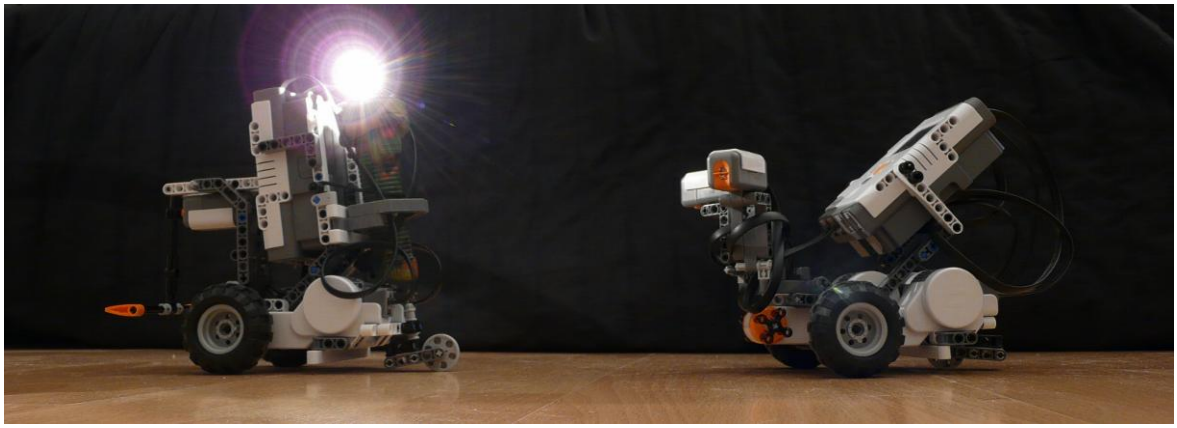
Στροφή 360 μοιρών και διάβασμα τιμών από τους αισθητήρες φωτός
Ψηλότερη τιμή = στόχος
Όσο η τιμή και των δύο αισθητήρων είναι ψηλή
    Προχώρα μπροστά
    Αν η τιμή του αριστερά αισθητήρα είναι χαμηλή
        Ψάξε για τον στόχο στα δεξιά
    Αν η τιμή του δεξιά αισθητήρα είναι χαμηλή
        Ψάξε για τον στόχο στα αριστερά
    Αν η τιμή και των δύο αισθητήρων είναι χαμηλή
        Ψάξε για τον στόχο με στροφή 360 μοιρών
Όταν βρεθεί ο στόχος συνέχισε ευθεία

Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα Α.5.

Επεξήγηση αλγόριθμου

Το Lego NXT γυρίζει μία φορά για να πάρει μετρήσεις (calibrate) και να εντοπίσει προς τα που βρίσκεται ο στόχος (φως). Αρχίζει και κατευθύνεται προς τον στόχο παρατηρώντας συνεχώς τις μετρήσεις των αισθητήρων. Σε περίπτωση που η αριστερή μέτρηση είναι πιο μεγάλη από την δεξιά, σημαίνει ότι ο στόχος είναι στα αριστερά του ρομπότ. Ισχύει το αντίστοιχο στα δεξιά. Το Lego NXT ψάχνει δεξιά και αριστερά ανάλογα με τις μετρήσεις με σκοπό να μην χάσει ποτέ από μπροστά του τον στόχο.

Σε περίπτωση που χαθεί ο στόχος πάει να πει ότι απομακρύνθηκε, είτε είναι πίσω του, είτε πίσω από κάποιο εμπόδιο. Σε έτσι περίπτωση κάνει ξανά ένα γύρο για να εντοπίσει τον στόχο.



Σχήμα 4.21: Εφαρμογή εντοπισμού φωτεινού στόχου από το ρομπότ

Αποτελέσματα

Η εφαρμογή αυτή απαιτεί αρκετό σκοτάδι ούτος ώστε να ξεχωρίζει το φως του στόχου. Επίσης δεν πρέπει να υπάρχουν έντονες αντανακλάσεις κοντά στην περιοχή όπως καθρέφτες, επειδή μπορούν να παραπλανήσουν το Lego NXT. Για να ελέγξουμε την εφαρμογή τοποθετήσαμε μία λάμπα πάνω στο ένα Lego NXT, όπως φαίνεται στο σχήμα 4.18, το οποίο θα ήταν ο στόχος και το αφήσαμε να κινείται τυχαία μέσα σε μία ελεγχόμενη περιοχή. Το δεύτερο Lego NXT έτρεχε τον αλγόριθμο εντοπισμού στόχου και προσπαθούσε να φτάσει στο πρώτο Lego NXT.

Όπως είδαμε από της δοκιμές, η εφαρμογή υστερούσε σε αρκετούς τομείς. Όταν ο στόχος κρυβόταν πίσω από κάποιο εμπόδιο, ο ‘‘κυνηγός’’ δεν μπορούσε να τον εντοπίσει όσο βρισκόταν πίσω από το εμπόδιο. Δηλαδή για να μπορεί να ακολουθεί τον στόχο απαιτεί να είναι συνεχώς μέσα στην εμβέλεια όρασης του. Άλλο πρόβλημα προκύπτει όταν αυξήσουμε την ταχύτητα του στόχου και μειώσουμε αυτή του κυνηγού. Σε έτσι περίπτωση μπορεί να απομακρυνθεί εύκολα ο στόχος και το Lego NXT θα αδυνατεί να τον εντοπίζει. Αυτά τα προβλήματα μπορούν να επιλυθούν με τη βοήθεια ενός ασύρματου δικτύου αισθητήρων για τον εντοπισμό στόχου.

4.4.3.2. Εντοπισμός φωτεινού στόχου με τη βοήθεια του WSN

Αυτή η εφαρμογή έχει ακριβώς τον ίδιο σκοπό με την προηγούμενη. Η διαφορά είναι στον τρόπο υλοποίησης. Εδώ θα παίρνουν της μετρήσεις του φωτός οι κόμβοι MicaZ και θα τις στέλνουν στο Lego NXT το οποίο θα υπολογίζει πού είναι ο στόχος. Για τους υπολογισμούς χρειάζεται τις μετρήσεις των αισθητήρων μαζί με τις συντεταγμένες του κόμβου που τις έστειλε και σε ποια χρονική στιγμή πάρθηκαν.

Αλγόριθμος εφαρμογής

Κάθε φορά που διαβάζει τις τιμές των αισθητήρων

Επιλέγει τους τρείς αισθητήρες με τις πιο ψηλές μετρήσεις

Υπολογίζει βάση των συντεταγμένων και των μετρήσεων τους την πιθανή θέση του στόχου

Ακολουθία στόχου (δύο τρόποι)

- Αλλαγή κατεύθυνσης προς τον στόχο και κίνηση προς αυτόν
  Επανάληψη μέχρι να φτάσει κοντά στο στόχο
- Πρόσθεση τοποθεσίας στόχου σε ουρά
  Αλλαγή κατεύθυνσης προς τον επόμενο στόχο και κίνηση προς αυτόν βάση της σειράς που αποθηκεύονται στην ουρά
  Επανάληψη μέχρι να εκτελέσει την διαδρομή του στόχου

Ο κώδικας της εφαρμογής βρίσκεται στο Παράρτημα Β.4.

Επεξήγηση αλγόριθμου

Το Lego NXT θα λαμβάνει τις μετρήσεις των αισθητήρων μαζί με τις συντεταγμένες του κόμβου που τις έστειλε κάθε ένα δευτερόλεπτο για να προσαρμόζει κατάλληλα την πορεία του σύμφωνα με τις πιο πρόσφατες μετρήσεις. Ο αισθητήρας με την πιο ψηλή μέτρηση έχει κοντά του τον στόχο εκείνη τη χρονική στιγμή. Σε περίπτωση που πολλοί αισθητήρες έχουν ψηλές τιμές σημαίνει πως ο στόχος βρίσκεται κάπου στο κέντρο αυτών των αισθητήρων.

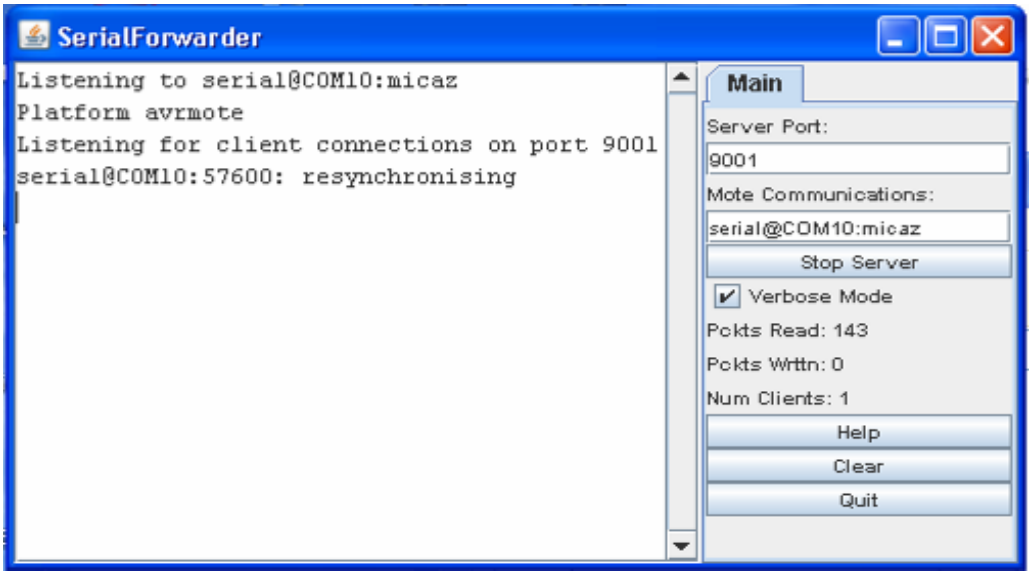
Μπορεί με αρκετό πειραματισμό και μετρήσεις να βρεθεί η απόσταση του στόχου από τον αισθητήρα ανάλογα με την τιμή του φωτός. Με αυτό τον τρόπο θα μπορούμε να χρησιμοποιούμε τους τρεις κόμβους με τις πιο ψηλές τιμές και να υπολογίζουμε ακριβώς τη θέση του στόχου με τη μέθοδο του triangulation. Φυσικά θα υπάρχει κάποιο σφάλμα ανάλογα με την ποιότητα και ποσότητα των μετρήσεων που πάρθηκαν για να υπολογιστεί η σχετική απόσταση.

Στη δική μας περίπτωση υπολογίζουμε την θέση του στόχου βάση των τριών αισθητήρων με τις πιο ψηλές μετρήσεις την συγκεκριμένη χρονική στιγμή. Για να έχουμε λίγη περισσότερη ακρίβεια, υπολογίζουμε το κέντρο αυτών των αισθητήρων μαζί με τα βάρη των μετρήσεων τους. Δηλαδή ο στόχος θα είναι πιο κοντά στον αισθητήρα με την ψηλότερη από τις τρείς μετρήσεις.

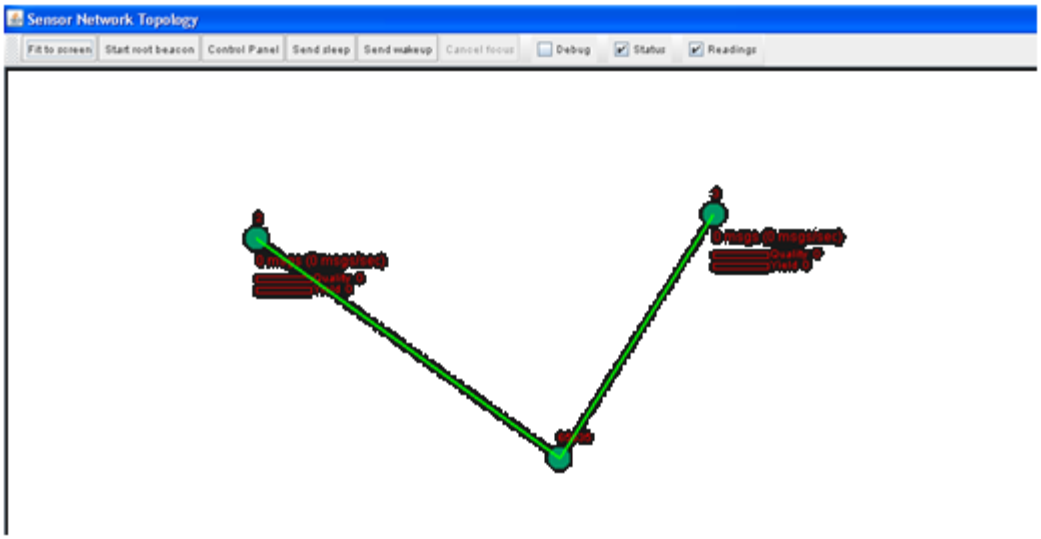
Αποτελέσματα

Για να ελεγχθεί το πρώτο κομμάτι της εφαρμογής, δηλαδή η συλλογή των μετρήσεων από τους αισθητήρες, υλοποιήσαμε την εφαρμογή Surge. Ο κόμβος που θα συλλέγει τις πληροφορίες και θα τις δίνει στο Lego NXT έτρεχε το πρόγραμμα TOSbase, ενώθηκε πάνω στον υπολογιστή με σειριακό καλώδιο και τρέξαμε το Java πρόγραμμα SerialForwarder και Surge για να έχουμε απεικόνιση στον υπολογιστή όλων των μετρήσεων που λαμβάνει ο κόμβος. Οι υπόλοιποι κόμβοι έτρεχαν το πρόγραμμα Surge_mst_light το οποίο παίρνει μετρήσεις από τον αισθητήρα φωτός και τις στέλνει στη βάση. Σκορπίσαμε τους κόμβους μέσα στην ελεγχόμενη περιοχή και κινούσαμε την λάμπα μέσα στη περιοχή.

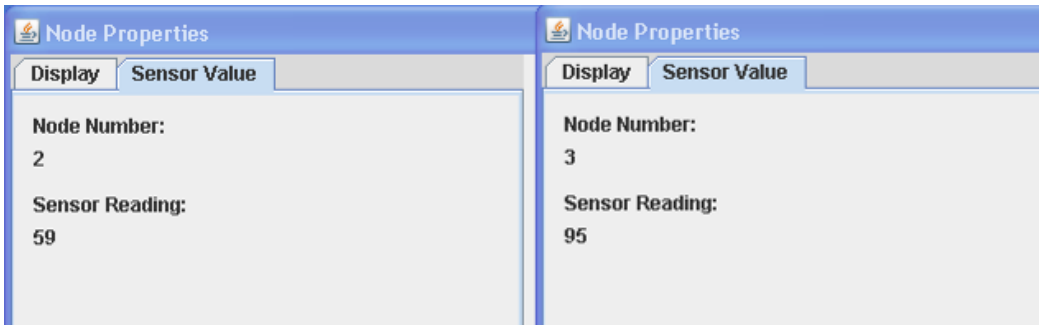
(α)



(β)



(γ)



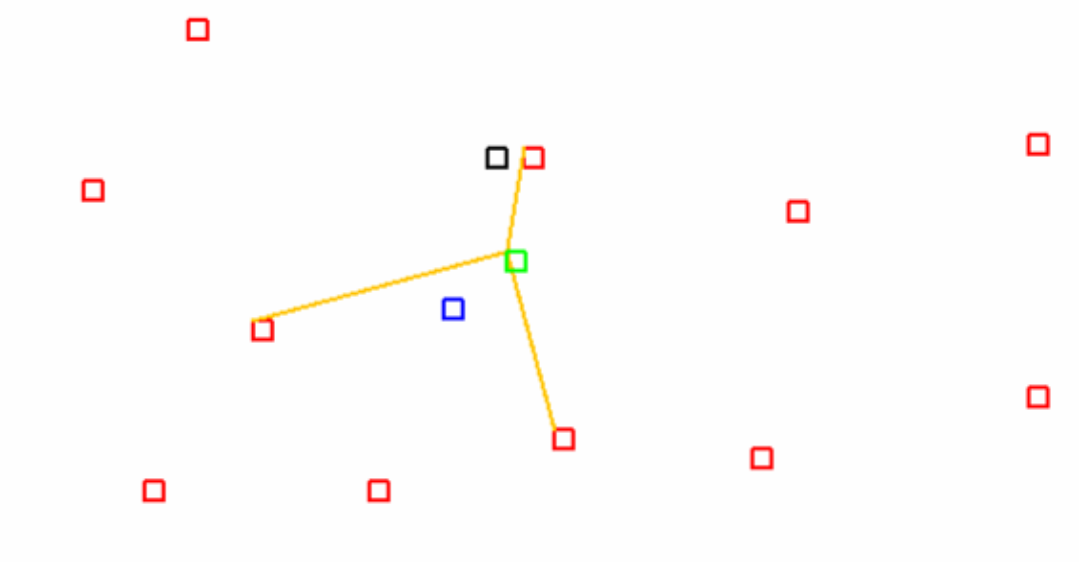
Σχήμα 4.22: Συλλογή μετρήσεων από τους αισθητήρες

(α) Serial Forwarder. Λήψη πακέτων από τους αισθητήρες

(β) Surge. Απεικόνιση των ενωμένων αισθητήρων

(γ) Surge. Προβολή των μετρήσεων του κάθε αισθητήρα

Για να το δεύτερο κομμάτι της εφαρμογής, δηλαδή η συλλογή των μετρήσεων από το ρομπότ και ο υπολογισμός της τοποθεσίας του στόχου, δημιουργήσαμε ένα πρόγραμμα Java το οποίο άλλαζε συνεχώς τη θέση του στόχου και ενημέρωνε τις μετρήσεις των αισθητήρων βάση της απόστασης τους από τον στόχο. Στη συνέχεια το ρομπότ διάβαζε το αρχείο με τις μετρήσεις και υπολόγιζε που βρίσκεται ο στόχος. Το ρομπότ κάθε φορά που υπολόγιζε την καινούρια τοποθεσία του στόχου είτε άλλαζε τον προορισμό του για να πάει προς τον στόχο από την πιο σύντομη διαδρομή, είτε πρόσθετε την τοποθεσία μέσα σε μία σειρά για να εκτελέσει την ίδια διαδρομή που ακολούθησε και ο στόχος.



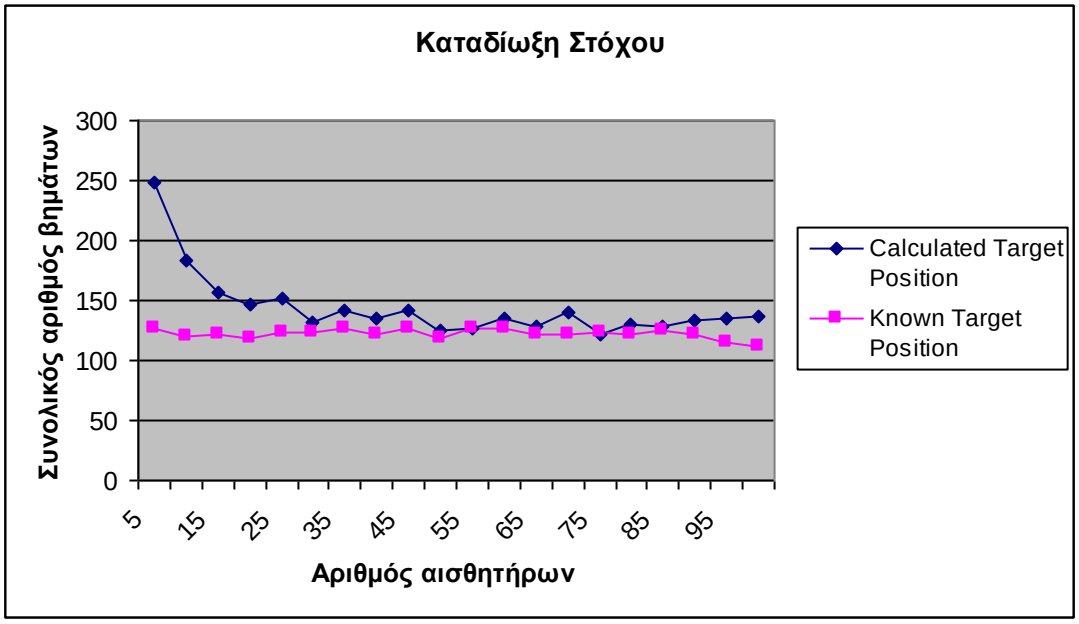
Σχήμα 4.23: Γραφική ακολουθία στόχου από το ρομπότ

(κόκκινο = αισθητήρες, πράσινο = πραγματική θέση στόχου, μπλε = υπολογισμένη θέση στόχου, μαύρο = ρομπότ, πορτοκαλί γραμμές = 3 πιο κοντινοί αισθητήρες στο στόχο)

Σύγκριση αλγορίθμων

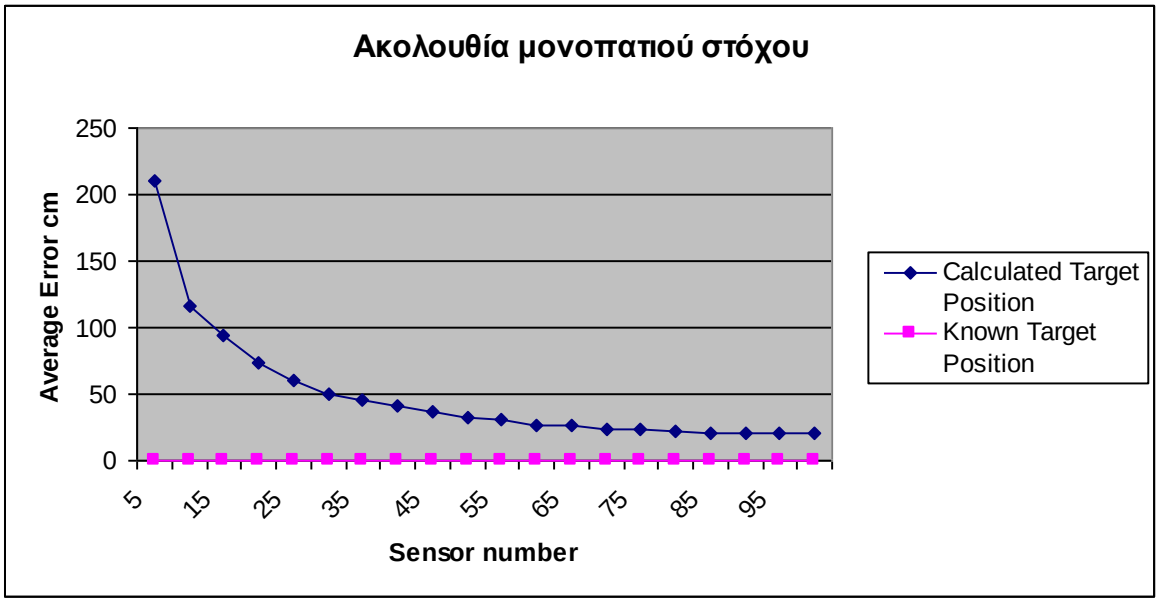
Για να συγκρίνουμε την επίδοση των αλγορίθμων τους τρέξαμε αρκετές φορές αλλάζοντας τον αριθμό αισθητήρων μέσα στο δίκτυο με ελεγχόμενη περιοχή μεγέθους 680x440 cm και παράλληλα τρέχαμε το ίδιο πρόγραμμα με τη διαφορά ότι το ρομπότ γνώριζε συνεχώς που ήταν ο στόχος.

Στην πρώτη περίπτωση έχουμε την συνεχή ακολουθία του στόχου. Μετρούμε πόσα βήματα χρειάστηκε για να φτάσει τον στόχο και τον συγκρίνουμε με την ιδανική περίπτωση που γνωρίζει που είναι ο στόχος και δεν χρειάζεται υπολογισμούς. Όπως φαίνεται στη γραφική παράσταση, σχήμα 4.24 η ιδανική περίπτωση είναι σταθερή, ενώ η κανονική χρειάζεται παραπάνω βήματα όταν είναι λίγοι οι αισθητήρες. Αυτό οφείλεται στο ότι δεν έχει αρκετά δεδομένα το ρομπότ για να υπολογίσει την σωστή θέση του στόχου.



Σχήμα 4.24: Γραφική παράσταση για την επίδοση της καταδίωξης στόχου

Στην δεύτερη περίπτωση έχουμε την ακολουθία του μονοπατιού που έκανε ο στόχος. Μετρούμε τον μέσο όρο της διαφοράς θέσης της πραγματικής θέσης του στόχου και της υπολογισμένης θέσης. Η ιδανική περίπτωση που γνωρίζει που είναι ο στόχος και δεν χρειάζεται υπολογισμούς έχει πάντα μηδενικό σφάλμα. Όπως φαίνεται στη γραφική παράσταση, σχήμα 4.24 η ιδανική περίπτωση είναι πάντα μηδέν, ενώ η κανονική έχει μεγαλύτερο σφάλμα όσο πιο λίγοι είναι οι αισθητήρες. Αυτό οφείλεται στο ότι δεν έχει αρκετά δεδομένα το ρομπότ για να υπολογίσει την σωστή θέση του στόχου.



Σχήμα 4.25: Γραφική παράσταση για το λάθος στον υπολογισμό της θέσης του στόχου

4.5. Προβλήματα στην υλοποίηση

Το κύριο πρόβλημα που αντιμετωπίσαμε στην υλοποίηση ήταν στην συνδεσιμότητα του Lego NXT μαζί με το ασύρματο δίκτυο αισθητήρων MicaZ. Προσπαθήσαμε διάφορους τρόπους για να πετύχουμε επικοινωνία αλλά συνεχώς βρίσκαμε αδιέξοδα. Η μέθοδος με το πρωτόκολλο I2C δεν δούλεψε λόγο του ότι είχαμε ασυμβατότητα μεταξύ των διαφόρων εκδόσεων του TinyOS. Χρειαζόμασταν την έκδοση 1.x να τρέχει πάνω σε όλους τους κόμβους MicaZ που θα έπαιρναν μετρήσεις από την ελεγχόμενη περιοχή. Ο λόγος είναι γιατί μόνο στην έκδοση 1.x υπήρχαν drivers που υποστήριζαν το μοντέλο του sensor board το οποίο είχαν πάνω οι MicaZ. Όσο αφορά τον κόμβο MicaZ ο οποίος θα επικοινωνούσε με το Lego NXT μέσω I2C χρειαζόμασταν την έκδοση 2.x. Ο λόγος είναι γιατί το πρωτόκολλο I2C υποστηρίζεται μόνο από την έκδοση 2.x. Μετά από δοκιμές αποδείχτηκε ότι τα πακέτα που στέλλουν οι κόμβοι με έκδοση 1.x και 2.x διαφέρουν. Επομένως απορρίφθηκε η επιλογή I2C.

Επιχειρήσαμε να βρούμε ένα άλλο τρόπο επικοινωνίας πιο ψηλού επιπέδου και καταλήξαμε στο Bluetooth. Επειδή τα MicaZ υποστηρίζουν Zigbee και όχι Bluetooth, έπρεπε να προσθέσουμε ακόμη ένα συστατικό μέσα στο δίκτυο για να εκτελεί καθήκοντα "μεταφραστή". Προφανή λύση ήταν η χρήση του φορητού υπολογιστή. Το σκεπτικό ήταν να έχουμε ενωμένο τον κόμβο MicaZ, που θα επικοινωνούσε με το Lego NXT, πάνω στον υπολογιστή με σειριακό καλώδιο. Στη συνέχεια θα προσθέταμε κώδικα μέσα στον πηγαίο κώδικα της εφαρμογής Surge, η οποία αναπαριστά τα πακέτα που λαμβάνει ο ενωμένος κόμβος, και θα δημιουργούσαμε ένα αρχείο κειμένου με τις πληροφορίες που χρειαζόταν το Lego NXT για τους υπολογισμούς του. Με ένα απλό πρόγραμμα Java που θα έτρεχε παράλληλα πάνω στον υπολογιστή, θα στέλναμε αυτό το αρχείο σε τακτά χρονικά διαστήματα μέσω Bluetooth στο Lego NXT, το οποίο με τη σειρά του θα διάβαζε της πληροφορίες που ήθελε.

Δυστυχώς, παρά όλες τις προσπάθειες, δεν καταφέραμε την επικοινωνία μεταξύ Lego NXT και MicaZ. Η χρήση άλλων μοντέλων αισθητήρων ή και ρομπότ με περισσότερες (και κοινές) επιλογές στους τρόπους συνδεσιμότητας θα έλυναν αυτό το πρόβλημα.

Κεφάλαιο 5

Μελλοντική δουλειά

Τα ασύρματα δίκτυα αισθητήρων είναι σχετικά καινούργια τεχνολογία η οποία άρχισε να αναπτύσσεται ραγδαία τα τελευταία χρόνια. Είναι σίγουρο πως τα επόμενα χρόνια θα συνεχιστεί αυτή η ανάπτυξη και τα ασύρματα δίκτυα αισθητήρων θα εισβάλουν την καθημερινότητα μας όπως έγινε πριν λίγα χρόνια με τα κινητά τηλέφωνα. Υπάρχουν πολλοί τομείς οι οποίοι πρέπει να μελετηθούν. Ο τομέας ο οποίος μας απασχολεί εμάς, που είναι η εξυπηρέτηση των ασύρματων δικτύων αισθητήρων από ρομπότ, έχει από μόνος του πολλούς τομείς και μεγάλο περιθώριο ανάπτυξης.

Στη δική μας περίπτωση υπάρχουν αρκετοί τομείς που μπορούν να αναπτυχθούν ή να εξελιχθούν.

- Πρώτα απ'όλα πρέπει να βρεθεί τρόπος επικοινωνίας μεταξύ του ρομπότ και του ασύρματου δικτύου αισθητήρων. Αυτό μπορεί να γίνει με τη χρήση κάποιου άλλου πρωτοκόλλου επικοινωνίας, είτε με τη χρήση κατάλληλου συνδετικού κρίκου η ακόμη και με την χρήση άλλων αισθητήρων και ρομπότ. Αφού υλοποιηθεί αυτό θα μπορούν όλες οι προσομοιώσεις που δημιουργήσαμε για αυτή τη διπλωματική εργασία να γίνουν στην πράξη.

- Μπορεί να γίνει ένας αλγόριθμος καθορισμού θέσης. Είτε μέσω του δικτύου από τους αισθητήρες, είτε από το ρομπότ και να γίνει κάποια σύγκριση για την ακρίβεια και πόσο χρόνο χρειάστηκαν. Ακόμη μπορεί να γίνει χρήση ενός Cricket mode localization το οποίο θα αποτελεί την βάση σύγκρισης των άλλων τρόπων καθορισμού θέσης.

- Στον έλεγχο κάλυψης μπορούν να βρεθούν περισσότεροι τρόποι επιλογής της επόμενης τοποθέτησης του αισθητήρα. Έτσι θα μπορεί να βρεθεί ο πιο αποδοτικός τρόπος ο οποίος θα λιγοστεύει την συνολική απόσταση που θα ταξιδεύει το ρομπότ για να τοποθετήσει όλους τους επιπλέων αισθητήρες, και παράλληλα θα κάνει μία δίκαιη κατανομή των αισθητήρων και να προσφέρει ομοιόμορφη κάλυψη. Με την επικοινωνία του ρομπότ και των αισθητήρων, δεν θα χρειάζεται να λάβει όλες της πληροφορίες για τις τοποθεσίες των αισθητήρων από την αρχή. Αντί αυτού, θα μπορεί να κυκλοφορεί μέσα στο δίκτυο και ανάλογα με τα σήματα που λαμβάνει να βρίσκει το κενό και να αφήνει τον επιπλέον αισθητήρα όπως είδαμε στο κεφάλαιο 2.3.

- Όσο αφορά τον εντοπισμό στόχου μπορεί να γίνει περισσότερη επεξεργασία των δεδομένων που λαμβάνει το ρομπότ από τους αισθητήρες. Σκοπός αυτής της επεξεργασίας θα είναι ο υπολογισμός της ταχύτητας και κατεύθυνσης του στόχου ούτος ώστε ο καταδιώκτης να μην ακολουθεί τον στόχο από πίσω αλλά να προσπαθεί να τον αποκόψει. Σύγκριση των δύο αλγορίθμων θα δείξει κατα πόσο πιο αποδοτικός είναι ο δεύτερος.

- Σε θέματα κινητικότητας όπως την κίνηση προς ένα σημείο και κίνηση προς όλους τους αισθητήρες, μπορεί το ρομπότ να λαμβάνει υπόψη του όλους τους αισθητήρες και ως εμπόδια στη δημιουργία μονοπατιού για να μην τα βρίσκει μπροστά του μετά όταν εκτελεί τη διαδρομή. Με αυτό το τρόπο θα γλιτώσει πολλές άσκοπες κινήσεις για αποφυγή εμποδίων. Επίσης στον καθορισμό μονοπατιού προς όλους τους αισθητήρες μπορεί να λάβει υπόψη όσους αισθητήρες βρίσκονται κοντά δημιουργώντας ομάδες αισθητήρων (clusters) όπου θα έχει τη δυνατότητα να επισκεφτεί μόνο ένα κεντρικό τους σημείο που καλύπτεται από όλους τους αισθητήρες τις ομάδας αντί να επισκεφτεί ξεχωριστά τον κάθε ένα.

Σε γενικές γραμμές, τα ρομπότ θα παίζουν ένα πολύ σημαντικό ρόλο για τα ασύρματα δίκτυα αισθητήρων γιατί και αυτά τυγχάνουν ραγδαία ανάπτυξη τα τελευταία χρόνια. Θα μπορούν να παίξουν το ρόλο της αλληλεπίδρασης του ανθρώπου και του ασύρματου δικτύου αισθητήρων. Σε εφαρμογές όπως τα έξυπνα κτίρια, τα οποία ελέγχονται από

αισθητήρες, τα ρομπότ θα μπορούν να παίρνουν πληροφορίες από το δίκτυο και να ενημερώνουν τους ανθρώπους για διάφορες καταστάσεις, να πηγαίνουν σε περιοχές όπου υπάρχει πρόβλημα, να καθοδηγούν τους ανθρώπους στον προορισμό τους για μία συνάντηση ή στην έξοδο όταν εντοπίσει πυρκαγιά από το δίκτυο αισθητήρων. Επίσης με τη χρήση των δικών τους αισθητήρων θα μπορούν να εξυπηρετούν το δίκτυο ως ένας επιπλέον κινητός αισθητήρας. Θα δίνει πληροφορίες για την θερμοκρασία και φωτισμό για να μπορούν να ρυθμιστούν ανάλογα. Οι μετρήσεις του ρομπότ θα είναι πιο ρεαλιστικές γιατί θα κινείται μέσα στο περιβάλλων των ανθρώπων οπότε θα μετρά ακριβός αυτό που νιώθουμε.

Σε νοσοκομεία όπου όλοι οι ασθενείς και το ιατρικό προσωπικό θα έχουν προσωπικό κόμβο αισθητήρα, τα ρομπότ θα μπορούν να συνδυάζουν συμπτώματα των ασθενών και τις ειδικότητες του ιατρικού προσωπικού, και να βρίσκουν τον πλησιέστερο διαθέσιμο γιατρό σε περίπτωση προβλήματος. Επίσης θα μπορούν να ελέγχουν τα φάρμακα που χρειάζεται κάθε ασθενής και να του τα προσφέρουν στις κατάλληλες χρονικές στιγμές. Στο πεδίο μάχης όπου θα υπάρχουν σκορπισμένοι αισθητήρες για την παρακολούθηση του εχθρού, τα ρομπότ θα έχουν πρωταγωνιστικό ρόλο. Προβλέπεται ότι στο μέλλον δεν θα υπάρχει ο ανθρώπινος παράγοντας μέσα στο πεδίο μάχης.

Στο οδικό δίκτυο θα υπάρχουν αισθητήρες που να ελέγχουν την κίνηση και να ενημερώνουν τους οδηγούς και να κάνουν εισηγήσεις για δρομολόγια με λιγότερη κίνηση. Επίσης κινητά ρομπότ θα μπορούν να κινηθούν μέσα στο οδικό δίκτυο καθοδηγούμενα από τους ασύρματους αισθητήρες. Αυτά τα κινητά ρομπότ θα μπορεί να είναι οχήματα καθαρισμού του δρόμου, ή ακόμη και σκυβαλοσυλλεκτες τα οποία θα εκτελούν προκαθορισμένα δρομολόγια χωρίς την ανάγκη οδηγού. Αν σκεφτούμε ακόμη πιο μακριά, τα κινητά ρομπότ θα είναι όλων μας τα αυτοκίνητα τα οποία θα κινούνται μέσα στο οδικό δίκτυο αποφασίζοντας την καλύτερη διαδρομή προς τον προορισμό μας βάση των πληροφοριών του ασύρματου δικτύου αισθητήρων.

Αυτές είναι μόνο λίγες από τις χιλιάδες εφαρμογές που μπορούν να αναπτυχθούν και να υπάρχει συνεργασία ρομπότ με ασύρματα δίκτυα αισθητήρων.

Κεφάλαιο 6

Συμπεράσματα

6.1. Σύνοψη

Σε αυτή τη διπλωματική εργασία προσπαθήσαμε να προγραμματίσουμε κατάλληλα ένα κινητό ρομπότ ούτως ώστε να μπορεί να εξυπηρετεί αυτόνομα ένα ασύρματο δίκτυο αισθητήρων. Είδαμε καταρχάς τι είναι τα ασύρματα δίκτυα αισθητήρων. Πιο συγκεκριμένα είδαμε πως λειτουργούν, πού μπορούν να χρησιμοποιηθούν και τα διάφορα προβλήματα τους. Επίσης έγινε επεξήγηση για τις βασικές έννοιες των ασύρματων δικτύων αισθητήρων όπως τον έλεγχο τοπολογίας, τον καθορισμό θέσης, τον έλεγχο κάλυψης, τον εντοπισμό στόχου και την κινητικότητα. Για κάθε ένα αναφέρθηκαν μέθοδοι και προσεγγίσεις που χρησιμοποιούνται σήμερα. Έγινε επεξήγηση του τι θα υλοποιήσουμε και αναφορά σε προηγούμενες εφαρμογές πάνω στο ίδιο θέμα. Στη συνέχεια έχουμε αναλυτική περιγραφή των βημάτων της υλοποίησης και των εφαρμογών καθώς και τα αποτελέσματα αυτών των εφαρμογών. Τέλος γίνεται αναφορά στα προβλήματα που αντιμετωπίσαμε κατά την διάρκεια της υλοποίησης και προοπτικές για μελλοντική δουλειά πάνω στο θέμα.

6.2. Συμπεράσματα

Τα ασύρματα δίκτυα αισθητήρων είναι μία ενδιαφέρουσα και πολύ υποσχόμενη τεχνολογία. Μέσα από τη μελέτη που έγινε για τα ασύρματα δίκτυα αισθητήρων παρατηρήθηκε ότι ακόμη υστερούν πολύ, κυρίως λόγο των περιορισμένων πόρων που έχουν διαθέσιμους οι αισθητήρες. Αυτό το πρόβλημα πάντα θα υπάρχει γιατί όσο αναπτύσσεται η τεχνολογία, τόσο μικραίνουν οι αισθητήρες σε μέγεθος. Επομένως πρέπει να βρεθούν άλλοι τρόποι για να αντισταθμιστεί το πρόβλημα. Ένας τρόπος είναι η δημιουργία πιο αποδοτικών αλγορίθμων και πρωτοκόλλων. Άλλος τρόπος είναι η χρήση ρομπότ μέσα στο δίκτυο που θα εξυπηρετούν το δίκτυο σε όσο το δυνατό περισσότερους τομείς.

Με την χρήση ρομπότ μέσα σε ένα ασύρματο δίκτυο αισθητήρων επωφελούνται και οι δύο πλευρές. Όπως είδαμε με την εφαρμογή εντοπισμού στόχου, το ρομπότ από μόνο του δυσκολευόταν να βρει τον στόχο του σε ορισμένες περιπτώσεις όπως όταν μπει μεταξύ τους ένα εμπόδιο. Στη συνέχεια όμως όταν προσθέσαμε και το ασύρματο δίκτυο αισθητήρων, παρατηρήσαμε ότι η καταδίωξη ήταν πιο αποτελεσματική στην προσομοίωση που εκτελέσαμε. Για να ισχύει όμως αυτό, πρέπει η περιοχή να είναι επαρκώς καλυμμένη αφού όπως είδαμε από τις μετρήσεις, όσο λιγότεροι ήταν οι αισθητήρες, τόσο περισσότερη ώρα ήθελε το ρομπότ για να πετύχει τον στόχο του. Δηλαδή σε ακάλυπτες περιοχές δεν μπορεί να υπολογίσει την θέση του στόχου. Δηλαδή πολλές εφαρμογές λειτουργούν καλύτερα με τον συνδυασμό των δύο τεχνολογιών επειδή η μία καλύπτει τα κενά της άλλης. Με τους ασύρματους κόμβους αισθητήρων έχουμε μία ολική κάλυψη της περιοχής ενώ με το ρομπότ έχουμε αλληλεπίδραση με το περιβάλλων και δυνατότητα επέκτασης.

Ο καθορισμός θέσης μέσα σε ένα ασύρματο δίκτυο αισθητήρων παίζει πρωταρχικό ρόλο στο αν θα πετύχει η εφαρμογή. Ακόμη πιο σημαντικός είναι ο καθορισμός θέσης ενός κινητού ρομπότ. Επειδή το ρομπότ κινείται συνεχώς και αλλάζει πορεία και προορισμούς ανάλογα με τις πληροφορίες του δικτύου, πρέπει να γνωρίζει ανα πάσα στιγμή την θέση του. Μέθοδοι όπου το ρομπότ δημιουργεί χάρτη της περιοχής απαιτεί μεγάλη ακρίβεια, όμως μετά τη δημιουργία του χάρτη μπορεί εύκολα να κατατοπιστεί και να γνωρίζει την

θέση του. Όπως είδαμε σε όλες τις εφαρμογές που δημιουργήσαμε αντιμετωπίζαμε πρόβλημα προσανατολισμού. Αυτό οφειλόταν στο ότι το ρομπότ δεν ανανέωνε την θέση του βάση το τι είχε γύρο του, αλλά βάση της αρχικής του θέσης και της διαδρομής που κάλυψε. Αυτό θα ήταν αποδεκτό σε εφαρμογές όπου οι κινήσεις του ρομπότ είναι σταθερές μέσα σε ελεγχόμενο περιβάλλον.

Όσο αφορά τον καθορισμό μονοπατιού παρατηρήσαμε πως καμία από της μεθόδους που χρησιμοποιήσαμε δεν ήταν τέλεια. Η μία μέθοδος που έλεγχε όλα τα δυνατά μονοπάτια, έβρισκε μεν το συντομότερο αλλά απαιτούσε πάρα πολλή μνήμη. Η δεύτερη μέθοδος που έβρισκε τον κοντινότερο γείτονα κάθε φορά, υστερούσε στο ότι όσο αυξάνονταν οι αισθητήρες, τόσο μεγαλύτερο σφάλμα είχε στην δημιουργία του μονοπατιού, πράγμα που το αποτρέπει από το να χρησιμοποιηθεί σε δίκτυα μεγάλης κλίμακας. Για να επιλυθεί αυτό το θέμα μπορούν να χρησιμοποιηθούν εριστικές (heuristics) μέθοδοι οι οποίες θα βρίσκουν όσο το δυνατό πιο σύντομο μονοπάτι χωρίς να υπερφορτώνουν την μνήμη και επεξεργαστή.

Στον έλεγχο κάλυψης παρατηρήσαμε ότι αφού βρούμε όλη την ακάλυπτη περιοχή, μπορούμε να επιλέξουμε την θέση για τον επόμενο αισθητήρα βάση διαφόρων παραγόντων. Όταν η επιλογή της θέσης είναι πάντα βάση μίας συγκεκριμένης τοποθεσίας, πετυχαίνουμε μια στοιχισμένη και ομοιόμορφη κάλυψη του δικτύου. Σε περίπτωση που είναι βάση άλλων παραγόντων όπως το κέντρο της περιοχής με τη λιγότερη κάλυψη, τότε η κάλυψη δεν είναι στοιχισμένη αλλά λαμβάνει υπόψη όλη την έκταση της περιοχής προσφέροντας μια πιο δίκαιη κατανομή που δεν ευνοεί μόνο ένα κομμάτι της ολικής περιοχής. Η χρήση μίας ενδιάμεσης λύσης θα μπορούσε να πάρει τα θετικά και από τους δύο πιο πάνω τρόπους. Δηλαδή να έβρισκε το κέντρο της περιοχής με τη λιγότερη κάλυψη και η τοποθέτηση να γινόταν βάση μίας σχάρας (grid) για να έχουμε δίκαιη κατανομή και ομοιόμορφη κάλυψη ταυτοχρόνος.

Βιβλιογραφία

[1] Andrew Howard, Maja J Mataric, and Gaurav S Sukhatme, "Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem", Distrib Auton Robot Syst 5, pp. 299-308, 2002.

[2] Ashima Gupta, Chao Gui, Prasant Mohapatra, "Mobile Target Tracking Using Sensor Networks", Mobile, Wireless, and Sensor Networks, John Wiley & Sons, pp. 173-196 2006

[3] Chi-Fu Huang and Yu-Chee Tseng, "The Coverage Problem in a Wireless Sensor Network", International Workshop on Wireless Sensor Networks and Applications, pp. 115-121, 2003.

[4] Cortes J., Martinez S., Karatas T. and Bullo F., "Coverage control for mobile sensing networks", Robotics and Automation IEEE Transactions on Vol. 20, Issue 2, pp.243-255, April 2004.

[5] Culler D., Estrin D. and Srivastava M., "Guest Editors Introduction: Overview of Sensor Networks", IEEE Computer Society, vol. 37, Issue 8, Aug. 2004, pp. 41 - 49, 2004.

[6] D. Hsu, W.S. Lee, and N. Rong, "A point-based POMDP planner for target tracking", In Proc. IEEE Int. Conf. on Robotics & Automation, pp. 2644–2650, 2008.

[7] Debashis Saha and Arunava Saha Dalal, "Topology Control", Sensor Networks and Configuration, pp. 119-141, 2007.

[8] Eylem Ekici, Yaoyao Gu, and Doruk Bozdag, "Mobility-Based Communication in Wireless Sensor Networks", IEEE Communications Magazine, pp. 56-62, July 2006.

[9] Gregory O'Hare, Mauro Dragone and Jennifer Treanor, "The CLARITY Ubiquitous Robotic Testbed", ERCIM News 76, Issue January 2009, pp. 34-35, 2009.

[10] Guiling Wang, Guohong Cao, and Tom La Porta, "A Bidding Protocol for Deploying Mobile Sensors", Network Protocols, 11th IEEE International Conference on 4-7 November 2003, pp. 315-324, 2003.

[11] Guiling Wang, Guohong Cao, and Tom La Porta, "Movement-assisted sensor deployment", Mobile Computing, IEEE Transactions on, Vol. 5, Issue 6, pp. 640-652, June 2006.

[12] Guoqiang Mao, Baris¸ Fidan and Brian D.O. Anderson, "Wireless Sensor Network Localization Techniques", Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 51, pp. 2529-2553, 2007.

[13] Hongyang CHEN, Kaoru SEZAKI, Ping DENG and Hing Cheung SO, "An Improved DV-Hop Localization Algorithm with Reduced Node Location Error for Wireless Sensor Networks", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, IEICE TRANS. FUNDAMENTALS, VOL.E91–A, 2008.

[14] Ioannis Chatzigiannakis, Athanasios Kinalis and Sotiris Nikoletseas, "Sink mobility protocols for data collection in wireless sensor networks", Proceedings of the 4th ACM international workshop on Mobility management and wireless access, pp. 52-59, 2006.

[15] John A. Stankovic, "Research Challenges for Wireless Sensor Networks", ACM SIGBED Review, vol. 1, Issue 2 July, pp. 9-12, 2004.

[16] Jonathan Friedman, David Lee, Ilias Tsigkogiannis, Sophia Wong, Dennis Chao, David Levin, William Kaiser and Mani Srivastava, "RAGOBOT: A New Platform for Wireless Mobile Sensor Networks", Distributed Computing in Sensor Systems, Vol. 3560/2005, pp.412, 2005.

[17] Karl Holger and Andreas Willig, "Protocols and Architectures for Wireless Sensor Networks", John Wiley and Son, 2007

[18] Khin Thanda Soe, "Increasing Lifetime of Target Tracking Wireless Sensor Networks", Proceedings of world academy of science, Engineering and Tecknology, vol. 32, August 2008.

[19] Lambrou T.P. and Panayiotou C.G., "Collaborative event detection using mobile and stationary nodes in sensor networks", Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2007. International Conference on 12-15 November, pp. 106 - 115, 2007.

[20] LCD03-I2C/Serial LCD Technical Documentation, Devantech

[21] Mo Li and Baijian Yang, "A Survey on Topology issues in Wireless Sensor Network", ICWN 2006, pp. 503-512, 2006.

[22] Montemerlo Michael, "Fastslam: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics", pp. 1-10, 2007

[23] MPR-MIB Users Manual, Crossbow Technology Inc., Revision A, June 2007

[24] Nadeem Ahmed, Salil S. Kanhere and Sanjay Jha, "The holes problem in wireless sensor networks: a survey", ACM SIGMOBILE Mobile Computing and Communications Review archive vol. 9 , Issue 2, pp. 4-18, 2005.

[25] Nesamony S., Vairamuthu M.K. and Orlowska M.E., "On Optimal Route of a Calibrating Mobile Sink in a Wireless Sensor Network", Networked Sensing Systems, Fourth International Conference on 6-8 June 2007 pp. 61-64, 2007.

[26] Paolo Santi, "Topology control in wireless ad hoc and sensor networks", John Wiley and Son, July 2005

[27] Rickard Karlsson, "Simulation Based Methods for Target Tracking", Linkoping Studies in Science and Technology Thesis No. 930, 2002.

[28] Severino Ricardo and Alves Mairio, "Engineering a Search and Rescue Application with a Wireless Sensor Network - based Localization Mechanism", World of Wireless, Mobile and Multimedia Networks, WoWMoM 2007, IEEE International Symposium on 18-21 June, pp. 1-4, 2007.

[29] T. Bandyopadhyay, Ang M.H. and D. Hsu, "Motion planning for 3-D target tracking among obstacles", In Proc. Int. Symp. on Robotics Research, 2007.

[30] T. Bandyopadhyay, Hsu D. and Ang M.H., "Motion strategies for people tracking in cluttered dynamic environments", In Proc. Int. Symp. on Experimental Robotics, 2008.

[31] T. Bandyopadhyay, Yuanping Li, Ang M.H. and Hsu D., "A greedy strategy for tracking a locally predictable target among obstacles", Robotics and Automation, ICRA 2006 Proceedings IEEE International Conference on 15-19 May, pp. 2342-2347, 2006.

[32] T. Bandyopadhyay, Yuanping Li, Ang M.H. and Hsu D., "Stealth tracking of an unpredictable target among obstacles" In M. Erdmann and others, Algorithmic Foundations of Robotics VI---Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 43–58, 2004.

[33] Tian He, Pascal Vicaire, Ting Yan, Liqian Luo, Lin Gu, Gang Zhou, Radu Stoleru, Qing Cao, John A. Stankovic, Tarek Abdelzaher, "Achieving Real-Time Target Tracking Using Wireless Sensor Networks," rtas, pp.37-48, 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06), 2006.

[34] Yong K. Cho and Jong-Hoon Youn, "Wireless Sensor-driven Intelligent Navigation Robots for Indoor Construction Site Security and Safety", ISARC2006, pp. 493-498, 2006.

[35] Zoltán Vincze and Rolland Vida, "Multi-hop wireless sensor networks with mobile sink", International Conference on Emerging Networking Experiments and Technologies, pp. 302-303, 2005.

Παράρτημα Α

Σε αυτό το παράρτημα έχουμε όλους τους κώδικες που γράφτηκαν στο Eclipse και απευθύνονταν για το Lego NXT.

Α.1 Ακολουθία γραμμής

```
//Οι βιβλιοθήκες που χρησιμοποιεί το πρόγραμμα
import lejos.navigation.Pilot;
import lejos.nxt.Button;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.LightSensor;

//Η κλάση followLineSimple η οποία περιέχει μόνο μία μέθοδο, την main
public class followLineSimple {
public static void main(String[] args) {

	//Χειριστήριο το οποίο μας επιτρέπει να κινούμε το NXT
	//λαμβάνει ως παραμέτρους την διάμετρο των τροχών, την μεταξύ
	//τους απόσταση και το πού είναι ενωμένα.
	Pilot pilot = new Pilot(5.6f,11.6f,Motor.B, Motor.C);
	//Χειριστήριο το οποίο μας επιτρέπει να λαμβάνουμε τιμές από
	//τον αισθητήρα φωτός
	LightSensor line= new LightSensor(SensorPort.S1, true);
	//Μεταβλητές τις οποίες χρησιμοποιεί το πρόγραμμα
	int i,low,high, black;
	int flag = 0;
	int turn=20;
	boolean cont=false;
	//διάβασμα αρχικής τιμής φωτός
	i=line.readValue();
	low=i; high=i;
	//στροφή 360 μοιρών για να διαβάσει τις τιμές το ρομπότ και
	//να αποφασίσει το threshhold μεταξύ της γραμμής (μαύρο, χαμηλή τιμη)
	//και το έδαφος (άσπρο, μεγάλη τιμή)
	pilot.rotate(360,true);
	while(pilot.isMoving()){
		i=line.readValue();
		if (i>high){ high=i; }
		else if  (i<low) { low = i; }
	}
	//το threshhold μεταξύ της γραμμής και του εδάφους.
	black = (high + low ) / 2;
	do{	//το ρομπότ προχωρά μπροστά και διαβάζει τιμές
		//μέχρι να διαβάσει τιμή πιο μικρή από το threshold
		//δηλαδή να ξεφύγει από την μαύρη γραμμή
		pilot.forward();
		i=line.readValue();
		turn=20;
		cont = false;
		if(i> black ){
		//όταν ξεφύγει από τη μαύρη γραμμή σταματά και ελέγχει
		//τη μία δεξιά και τη μία αριστερά μέχρι να βρει τη γραμμή
		pilot.stop();
		while (cont==false) {
			if(flag==0 && cont==false){
			pilot.rotate(turn, true);	//στροφή αριστερά
			flag=1;
			while(pilot.isMoving()){	//έλεγχος για εντοπισμό
				i=line.readValue();	//της γραμμής
				if(i<=black){
					pilot.stop();
					cont=true;
					flag=0;
				}
			}
			}

			else if (flag==1 && cont==false) {
```

```
                                pilot.rotate(-turn, true);    //στροφή δεξιά
                                flag=0;
                                while(pilot.isMoving()){//έλεγχος για εντοπισμό
                                        i=line.readValue();//της γραμμής
                                        if(i<=black){
                                                pilot.stop();
                                                cont=true;
                                                flag=1;
                                        }
                                  }
                                }
                            //αν δέν εντοπίσει τη μαύρη γραμμή αυξάνει την γωνιά ελέγχου
                             if (cont== false)
                                    turn+=45;
                            }
                    }
            //έξοδος από το πρόγραμμα με το πάτημα του κουμπιού
            }while(!Button.ESCAPE.isPressed());
}
}
```

A.2 Επικοινωνία Lego NXT και LCD03

```
package i2c2;
import lejos.nxt.I2CSensor;
import lejos.nxt.LCD;
import lejos.nxt.SensorPort;
import lejos.nxt.*;

public class Hello {
           public static void main(String[] args) throws Exception{
                   I2CSensor i2c = new I2CSensor(SensorPort.S1);
                   i2c.setAddress(0x63);
                   i2c.sendData(0, (byte)12);        //Clear screen
                   i2c.sendData(0, (byte)72);        //H
                   i2c.sendData(0, (byte)101);       //e
                   i2c.sendData(0, (byte)108);       //l
                   i2c.sendData(0, (byte)108);       //l
                   i2c.sendData(0, (byte)111);       //o
                   i2c.sendData(0, (byte)32);       //(space)
                   i2c.sendData(0, (byte)109);       //m
                   i2c.sendData(0, (byte)121);       //y
                   i2c.sendData(0, (byte)32);        //(space)
                   i2c.sendData(0, (byte)109);       //m
                   i2c.sendData(0, (byte)97);        //a
                   i2c.sendData(0, (byte)115);       //s
                   i2c.sendData(0, (byte)116);       //t
                   i2c.sendData(0, (byte)101);       //e
                   i2c.sendData(0, (byte)114);       //r
                   Thread.sleep(3000);
            }
}
```

A.3 Κίνηση Lego NXT προς ένα σημείο

```
public class GoTo {
	static TachoNavigator pilot;
	//touch and ultrasonic sensors
	static TouchSensor touch = new TouchSensor(SensorPort.S2);
	static UltrasonicSensor see= new UltrasonicSensor(SensorPort.S1);

	public static void goTo(float[] args) {
		//Robot and target coordinates
		float xFrom=args[0], yFrom=args[1];
		float xTo=args[2], yTo=args[3];
		//The different behaviors of the robot.
		//Drive to destination, Hit or see obstacle
		pilot.setPosition(xFrom,yFrom,args[4]);
		pilot.setSpeed(200);
		//Start the behaviors
		do{
			//GoTo.pilot.updatePosition();
			pilot.goTo(xTo, yTo, true);
			while(pilot.isMoving()){

				if(touch.isPressed() || see.getDistance() < 20){
					GoTo.pilot.stop();
					int temp1, temp2;
					//if the robot hits an object it backs up
					if(touch.isPressed()){
						GoTo.pilot.travel(-10);

					}
				//check to see which direction has more clear space
					GoTo.pilot.rotate(45);
					temp1=see.getDistance();
					GoTo.pilot.rotate(-90);
					temp2=see.getDistance();
					//try to avoid obstacle from the direction
					//with less obstacles
					if(temp1>temp2){
						GoTo.pilot.rotate(90);
						GoTo.pilot.travel(25,true);
						while(GoTo.pilot.isMoving()){
							if(touch.isPressed())
								GoTo.pilot.stop();
						}
					}
					else{
						GoTo.pilot.travel(25,true);
						while(GoTo.pilot.isMoving()){
							if(touch.isPressed())
								GoTo.pilot.stop();
						}
					}
					GoTo.pilot.stop();
				}
			}
			GoTo.pilot.stop();
		}while(GoTo.pilot.distanceTo(xTo, yTo)>15);
	}
```

A.4 Καθορισμός μονοπατιού προς όλους τους αισθητήρες

```
public class CheckSensors{

	public static void main(String[] args) {
		//array of int holding the sensors and robot coordinates
		int[] xySenso = new int[10];
		xySenso[0]=0; xySenso[1]=0;
		xySenso[2]=0; xySenso[3]=20;
		xySenso[4]=40; xySenso[5]=20;
		xySenso[6]=40; xySenso[7]=0;
		xySenso[8]=0; xySenso[9]=0;
		//the number of sensors including the robot
		int sensor_num=xySenso.length/2;
		//array holding the shortest path
		int[] shortpath = new int[sensor_num];
		//array indicating if a sensor has been visited
		int[] visited = new int[sensor_num];
		//variable for the last visited sensor
		int lastVisited;
		//////////////////////////
		//Create array with costs
		int cost=0;
		int max=0;
		int min;
		double temp1=0, temp2=0;
		int[][] costs = new int[sensor_num][sensor_num];
		//for each sensor calculate its distance from the rest
		for(int i=0; i<sensor_num; i++){
			for(int j=0; j<sensor_num; j++){
				if(i!=j){
					temp1=xySenso[i*2]-xySenso[j*2];
					temp2=xySenso[i*2+1]-xySenso[j*2+1];
					temp1=temp1*temp1;
					temp2=temp2*temp2;
					temp1=temp1+temp2;
					cost=(int)Math.sqrt(temp1);
					costs[i][j]=cost;
					if(cost>max){
						max=cost;
					}
				}
				else
					costs[i][j]=0;
			}
		}

		//////////////////////////////////
		//calculate the shortes path
		for(int i=0;i<sensor_num;i++){
			visited[i]=0;
		}
		visited[0]=1;
		shortpath[0]=0;
		lastVisited=0;

		int next = 0,count=1;
		while(count<sensor_num){
			min=max*5;
			for(int i=0; i<sensor_num; i++){
				if(visited[i]== 0){
					if(i==sensor_num-1 && count!=sensor_num-1){}
					else {
						if(min>costs[lastVisited][i]){
							min=costs[lastVisited][i];
							next=i;
						}
					}
				}
			}
			shortpath[count]=next;
			visited[next]=1;
			lastVisited=next;
			count++;
		}
```

```
                ////////////////////////////////////
                //go to sensors
                float[] arg= new float[4];
                //Robot controler
                GoTo.pilot = new TachoNavigator(5.6f,11.6f,Motor.A, Motor.C);
                GoTo.pilot.setPosition(xySenso[0],xySenso[1], 90);
                for(int i=1; i<shortpath.length;i++){
                   arg[0]=xySenso[shortpath[i-1]*2];arg[1]=xySenso[shortpath[i-1]*2+1];
                   arg[2]=xySenso[shortpath[i]*2];arg[3]=xySenso[shortpath[i]*2+1];
                   arg[4]=GoTo.pilot.getAngle();
                   GoTo.goTo(arg);
                   Sound.playTone(100, 1000);
                }
        }
}
```

A.5 Εντοπισμός στόχου με φως

```
public class FollowLight {
        static Pilot pilot;
        static LightSensor lightR;
        static LightSensor lightL;
        static int midle;


public static void calibrate() {
        int i,j,highR,highL,turnR,turnL,turn;
        pilot.setSpeed(300);
        highR=0;highL=0;turnR=0;turnL=0;
        pilot.rotate(360,true);
        while(pilot.isMoving()){
                LCD.clear();
                i=lightR.readValue();
                j=lightL.readValue();
                LCD.drawInt(i ,0,1);
                LCD.drawInt(j ,0,2);
                LCD.refresh();
                if(i>=highR){
                        highR=i;
                        turnR=pilot.getAngle();
                }
                if(j>highL){
                        highL=j;
                        turnL=pilot.getAngle();
                }
        }
        midle=(highR+highL)/4;
        if(turnL>turnR){
                turn=(turnR+turnL)/2;
                if(turn>180){
                        turn-=360;
                }
                pilot.rotate(turn);
        }
        pilot.setSpeed(200);
}
```

```
public static void main(String[] args) {
	// TODO Auto-generated method stub
	pilot = new Pilot(5.6f,11f,Motor.A, Motor.C);
	lightR= new LightSensor(SensorPort.S1, false);
	lightL= new LightSensor(SensorPort.S2, false);
	boolean lost=false;
	int i,j;
	try {
		Thread.sleep(2000);
	} catch (InterruptedException e) {}
	calibrate();
	Motor.A.forward();
	Motor.C.forward();
	do{
		LCD.clear();
		i=lightR.readValue();
		j=lightL.readValue();
		LCD.drawInt(i ,0,1);
		LCD.drawInt(j ,0,2);
		LCD.refresh();
		if(i<j-3){
			Motor.A.forward();
			Motor.C.flt();
		}
		else if(i-3>j){
			Motor.A.flt();
			Motor.C.forward();
		}
		else{
			Motor.A.forward();
			Motor.C.forward();
		}
		if(i<midle && j<midle){
			Motor.A.stop();
			Motor.C.stop();
			lost=true;
			LCD.clear();
			LCD.drawString("Target lost" ,0,2);
			LCD.refresh();
			Sound.beep();
			pilot.setSpeed(300);
			pilot.rotate(360,true);
			while(pilot.isMoving()){
				LCD.clear();
				i=lightR.readValue();
				j=lightL.readValue();
				LCD.drawInt(i ,0,1);
				LCD.drawInt(j ,0,2);
				LCD.refresh();
				if(i>=midle || j>=midle){
					pilot.stop();
					lost=false;
				}
			}
			pilot.setSpeed(200);
			Motor.A.forward();
			Motor.C.forward();
		}
	}while(!Button.ESCAPE.isPressed() && !lost);
}


}
```

Παράρτημα Β

Σε αυτό το παράρτημα έχουμε όλους τους κώδικες που γράφτηκαν στο Eclipse για γραφική απεικόνιση των αλγορίθμων

Β.1 Καθορισμός θέσης αισθητήρων και ρομπότ μέσα στο δίκτυο

```
public class RandomizeSensors {

	//integer holding the cover range of the sensors
	public static int cover_range=25;


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

Β.1.1 Καθορισμός και εγγραφή θέσεων αισθητήρων και σε αρχείο

```
	//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 sensor_number=7,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();
		//coordinates of the robot
		writer.write("250 250");
		writer.newLine();
		//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()*500);
				sensory=(int)(Math.random()*500);
				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==3){
							break;
						}
					}
				}
			}while(close_sensors==3 && 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
        System.out.println("Sensors inside the area: " + actual_number);
        writer.write("250 250");
        writer.close();
}
```

B.1.2 Διάβασμα θέσεων αισθητήρων και ρομπότ μέσα στο δίκτυο από αρχείο

```
        //method for reading the coordinates from a txt file
        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());
                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;
        }


}
```

B.2 Έλεγχος κάλυψης

B.2.1 Περιοχή η οποία θα ελεγχθεί

```
// the class area represents the area being checked
class Area{
	float startx, starty, finishx, finishy;
	float volume;
	boolean covered,valid;
	int cover_range;
	float uncover_threshhold,covered_volume;
	//the constructor sets the boundaries of the area
	//and initializes its atributes
	Area(float x1,float y1,float x2,float y2){
		startx=x1;
		starty=y1;
		finishx=x2;
		finishy=y2;
		volume = (x2-x1) * (y2-y1);
		covered=false;
		covered_volume=0;
		valid=true;
		cover_range=RandomizeSensors.cover_range;
		uncover_threshhold=volume;
	}
```

B.2.2 Εντοπισμός όλων των ακάλυπτων περιοχών

```
//recursive method that checks the coverage of an area
ArrayList checkCoverage(ArrayList areas, int[] sensors,int areaThreshold,int threshold){

	int[] visited = new int[sensors.length-2];
	for(int i=0; i<visited.length;i++){
		visited[i]=0;
	}
	//for each sensor
	for(int i=2; i<sensors.length-2;i+=2){
		//check the sensors
		if((sensors[i]>=startx+cover_range && sensors[i]<=finishx-cover_range &&
		sensors[i+1]>=starty+cover_range && sensors[i+1]<=finishy-cover_range)){
			//add 1
			covered_volume+=(Math.PI+ Math.pow(cover_range, 2));
			}
		else if((sensors[i]>startx-cover_range && sensors[i]<finishx+cover_range &&
		sensors[i+1]>starty-cover_range && sensors[i+1]<finishy+cover_range)){
			visited[i]=1;
			//add 1/2
			if((sensors[i]<startx+cover_range || sensors[i]>finishx-cover_range) &&
			sensors[i+1]>=starty+cover_range && sensors[i+1]<=finishy-cover_range){
				covered_volume+=(Math.PI+ Math.pow(cover_range, 2))/2;
			}
			else if(sensors[i]>=startx+cover_range && sensors[i]<=finishx-
			cover_range && (sensors[i+1]>finishy-cover_range ||
			sensors[i+1]<starty+cover_range)){
				covered_volume+=(Math.PI+ Math.pow(cover_range, 2))/2;
			}
			//add 1/4
			else{
				covered_volume+=(Math.PI+ Math.pow(cover_range, 2))/4;
			}
		}
	}
```

```
        uncover_threshhold=volume*threshold/100;
        //if the area NOT covered is larger than our threshold, then we divide the area
        //for further analysis
        if((volume-covered_volume)>=uncover_threshhold){
                //divide the area in 4 and check again each one with a higher threshold
                if((finishx-startx)/2>=areaThreshold && (finishy-starty)/2>=areaThreshold){
                        valid=false;
////////////////////////////////////////////////////////////
areas.add(new Area(startx, starty, ((startx+finishx)/2), ((starty+finishy)/2)));
((Area)areas.get(areas.size()-1)).checkCoverage(areas, sensors,areaThreshold,threshold);
areas.add(new Area((startx+finishx)/2, starty, finishx,(starty+finishy)/2));
((Area)areas.get(areas.size()-1)).checkCoverage(areas, sensors,areaThreshold,threshold);
areas.add(new Area(startx, (starty+finishy)/2, (startx+finishx)/2,finishy));
((Area)areas.get(areas.size()-1)).checkCoverage(areas, sensors,areaThreshold,threshold);
areas.add(new Area((startx+finishx)/2, (starty+finishy)/2, finishx, finishy));
((Area)areas.get(areas.size()-1)).checkCoverage(areas, sensors,areaThreshold,threshold);
                }
        }
        else{
                covered = true;
        }
        return areas;
}
```

B.2.3 Εντοπισμός της πιο κρίσιμα ακάλυπτης περιοχής

```
//recursive method that finds the criticaly uncovered area
ArrayList findUncovered(ArrayList uncoveredArea, ArrayList areas,int areaThreshold){
        //counters holding the number of sensors near each corner
        float newX=0,newY=0;
        float counter=0,distancex=0,distancey=0;
        //for each sensor we calculate the distance from each sensor and add it to
        //the correct counter
        for(int i=0;i<areas.size();i++){
                if(((Area)areas.get(i)).covered==false && ((Area)areas.get(i)).valid==true &&
                ((Area)areas.get(i)).startx>=startx && ((Area)areas.get(i)).finishx<=finishx
                && ((Area)areas.get(i)).starty>=starty &&
                ((Area)areas.get(i)).finishy<=finishy){
                        counter++;
                        newX+=(((Area)areas.get(i)).startx+((Area)areas.get(i)).finishx)/2;
                        newY+=(((Area)areas.get(i)).starty+((Area)areas.get(i)).finishy)/2;
                }
        }
        distancex=(finishx-startx)/2-areaThreshold;
        distancey=(finishy-starty)/2-areaThreshold;
        newX/=counter;
        newY/=counter;
        //while we havent reached the area threshold
        //we check the area again with the new boundaries
        if(distancex*2>areaThreshold && distancey*2>areaThreshold){
                startx=newX-distancex;
                finishx=newX+distancex;
                starty=newY-distancey;
                finishy=newY+distancey;
                valid=false;
                uncoveredArea.clear();
                uncoveredArea.add(new Area(startx,starty, finishx, finishy));
                ((Area)uncoveredArea.get(uncoveredArea.size()-1)).findUncovered(uncoveredArea,
                areas,areaThreshold);
        }
        return uncoveredArea;
}


}
```

B.2.4 Γραφική απεικόνιση όλης της ακάλυπτης περιοχής

```
public class coverageDisplay extends Frame {

	private static final long serialVersionUID = 1L;
	//drawing variables
	static Stroke drawingStroke;
	static RoundRectangle2D  robot;
	static RoundRectangle2D[]  sensors;
	static RoundRectangle2D[]  cover;
	static RoundRectangle2D[]  uncover;

	//method for drawing the sensors and the coverage of the wsn
	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.green);
		for(int i=0;i<cover.length;i++){
			ga.draw(cover[i]);
		}
		ga.setPaint(Color.blue);
		for(int i=0;i<uncover.length;i++){
			ga.draw(uncover[i]);
		}
	}
	public static void main(String[] args) {
		//variables holding the cover range of the sensors and the
		//area threshold
		int cover_range =RandomizeSensors.cover_range;
		int areaThreshold=10;
		//integers holding the start and end coordinates of the controled area
		float startx=0, starty=0, finishx=500, finishy=500;
		//array with doubles holding the sensors coordinates
		int[] xySenso = null;
		try {
			//Get the coordinates from the file sensor_coordinates.txt
			xySenso = RandomizeSensors.getIntegersFromFile
			("sensor_coordinates.txt");
		} catch (IOException e) {
			e.printStackTrace();
		}
		//the number of sensors
		int sensor_num=xySenso.length/2-2;
		int threshold=100;
		// list holding the areas with their coverage
		ArrayList areas;
		areas = new ArrayList();
		// start the algorithm with the whole area
		do{
			Area area1 = new Area(startx, starty, finishx, finishy);
			areas = area1.checkCoverage(areas, xySenso, areaThreshold,
			threshold);
			threshold-=10;
		}while(areas.size()==0 && threshold>0);

		//print all the areas that are uncovered
		int counter=0;
		for(int i=0; i<areas.size();i++){
			if(((Area)areas.get(i)).valid==true &&
			((Area)areas.get(i)).covered==false){
				counter++;
			}
		}
```

```
                ////////////////////////////////////////
                //create shapes
                drawingStroke = new BasicStroke(2);
                //rectangle for each sensor and the robot
                robot=new RoundRectangle2D.Float();
                robot.setRoundRect(xySenso[0]-5+50,xySenso[1]-5+50,10, 10, 0, 0 );
                sensors= new RoundRectangle2D[sensor_num];
                for(int i=0;i<sensor_num;i++){
                        sensors[i]=new RoundRectangle2D.Float();
                        sensors[i].setRoundRect(xySenso[(i+1)*2]-
                        3+50,xySenso[(i+1)*2+1]-3+50,6, 6, 0, 0 );
                }
                //circle indicating the area covered by each sensor
                cover= new RoundRectangle2D[sensor_num];
                for(int i=0;i<sensor_num;i++){
                        cover[i]=new RoundRectangle2D.Float();
                        cover[i].setRoundRect(xySenso[(i+1)*2]-
                        cover_range+50,xySenso[(i+1)*2+1]-cover_range+50,cover_range*2,
                        cover_range*2,cover_range*2, cover_range*2 );
                }
                //rectangles indicating the uncovered area
                double temp1=0, temp2=0;
                uncover= new RoundRectangle2D[counter];
                counter=0;
                for(int i=0; i<areas.size();i++){
                        if(((Area)areas.get(i)).valid==true &&
                        ((Area)areas.get(i)).covered==false){
                                uncover[counter]=new RoundRectangle2D.Float();
                                temp1=(((Area)areas.get(i)).startx+((Area)areas.get(i)).
                                finishx)/2;
                                temp2=(((Area)areas.get(i)).starty+((Area)areas.get(i)).
                                finishy)/2;
                                uncover[counter].setRoundRect(temp1-
                                +areaThreshold/2+50,temp2-
                                areaThreshold/2+50,areaThreshold,areaThreshold,0,0);
                                counter++;
                        }
                }
                Frame frame = new coverageDisplay();
                frame.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
                });
                frame.setSize(600, 600);
                frame.setVisible(true);
        }
}
```

B.2.5 Γραφική απεικόνιση της πιο κρίσιμα ακάλυπτης περιοχής

```
public class criticalUncoverdAreaDisplay extends Frame {

	private static final long serialVersionUID = 1L;
	//drawing variables
	static Stroke drawingStroke;
	static RoundRectangle2D  robot;
	static RoundRectangle2D[]  sensors;
	static RoundRectangle2D[]  cover;
	static RoundRectangle2D  uncover;

	//method for drawing the route of the robot
	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.green);
		for(int i=0;i<cover.length;i++){
			ga.draw(cover[i]);
		}
		ga.setPaint(Color.blue);
		ga.draw(uncover);
	}

	public static void main(String[] args) {

		//variables holding the cover range of the sensors and the
		//area threshold
		int cover_range =RandomizeSensors.cover_range;
		int areaThreshold=10;
		//integers holding the start and end coordinates of the controled area
		float startx=0, starty=0, finishx=500, finishy=500;
		//array with doubles holding the sensors coordinates
		int[] xySenso = null;
		try {
			//Get the coordinates from the file sensor_coordinates.txt
			 xySenso = RandomizeSensors.getIntegersFromFile
			 ("sensor_coordinates.txt");
		} catch (IOException e) {
			e.printStackTrace();
		}
		//the number of sensors
		int sensor_num=xySenso.length/2-2;
		int threshold=100;
		// list holding the areas with their coverage
		ArrayList areas;
		areas = new ArrayList();
		// start the algorithm with the whole area
		do{
			Area area1 = new Area(startx, starty, finishx, finishy);
			 areas = area1.checkCoverage(areas,
			 xySenso,areaThreshold,threshold);
			 threshold-=10;
		}while(areas.size()==0 && threshold>0);

		//print all the areas that are uncovered
		int counter=0;
		for(int i=0; i<areas.size();i++){
			if(((Area)areas.get(i)).valid==true &&
			 ((Area)areas.get(i)).covered==false){
				counter++;
			}
		}
		// list holding the areas with their coverage
		ArrayList uncoveredArea;
		uncoveredArea = new ArrayList();
```

```
                // start the algorithm with the whole area
                Area area1 = new Area(startx, starty, finishx, finishy);
                uncoveredArea = area1.findUncovered(uncoveredArea, areas,
                areaThreshold);

                System.out.println(((Area)uncoveredArea.get(0)).startx+"
                "+((Area)uncoveredArea.get(0)).starty+"
                "+((Area)uncoveredArea.get(0)).finishx+"
                "+((Area)uncoveredArea.get(0)).finishy);

                ////////////////////////////////////////
                //create shapes
                drawingStroke = new BasicStroke(2);
                //rectangle for each sensor and the robot
                robot=new RoundRectangle2D.Float();
                robot.setRoundRect(xySenso[0]+50,xySenso[1]+50,10, 10, 0, 0 );
                sensors= new RoundRectangle2D[sensor_num];
                for(int i=0;i<sensor_num;i++){
                        sensors[i]=new RoundRectangle2D.Float();
                        sensors[i].setRoundRect(xySenso[(i+1)*2]-
                        3+50,xySenso[(i+1)*2+1]-3+50,6, 6, 0, 0 );
                }
                //circle indicating the area covered by each sensor
                cover= new RoundRectangle2D[sensor_num];
                for(int i=0;i<sensor_num;i++){
                        cover[i]=new RoundRectangle2D.Float();
                        cover[i].setRoundRect(xySenso[(i+1)*2]-
                        cover_range+50,xySenso[(i+1)*2+1]-cover_range+50,cover_range*2,
                        cover_range*2, cover_range*2, cover_range*2 );
                }
                //rectangle indicating the criticaly uncovered area
                double temp1=0, temp2=0;
                uncover=new RoundRectangle2D.Float();
                temp1=(((Area)uncoveredArea.get(uncoveredArea.size()-
                1)).startx+((Area)uncoveredArea.get(uncoveredArea.size()-
                1)).finishx)/2;
                temp2=(((Area)uncoveredArea.get(uncoveredArea.size()-
                1)).starty+((Area)uncoveredArea.get(uncoveredArea.size()-
                1)).finishy)/2;
                uncover.setRoundRect(temp1-areaThreshold+50,temp2-
                areaThreshold+50,areaThreshold*2,areaThreshold*2,0,0);

                Frame frame = new criticalUncoverdAreaDisplay();
                frame.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
                });
                frame.setSize(600, 600);
                frame.setVisible(true);

        }
}
```

B.2.6 Γραφική απεικόνιση της κάλυψης περιοχής

```
public class CoverAreaDisplay extends Frame {

	private static final long serialVersionUID = 1L;
	//drawing variables
	static Stroke drawingStroke;
	static RoundRectangle2D  robot;
	static RoundRectangle2D[]  sensors;
	static RoundRectangle2D[]  cover;
	static RoundRectangle2D[]  uncovered;
	static RoundRectangle2D  uncover;




	//method for drawing the route of the robot
	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.green);
		for(int i=0;i<cover.length;i++){
			ga.draw(cover[i]);
		}
		ga.setPaint(Color.yellow);
		for(int i=0;i<uncovered.length;i++){
			ga.draw(uncovered[i]);
		}
		ga.setPaint(Color.blue);
		ga.draw(uncover);
	}

	public static void main(String[] args) {

		//variables holding the cover range of the sensors and the
		//area threshold
		int type=1;
		int counter;
		int nextSensor = 0;
		Frame frame = new CoverAreaDisplay();
		float[] target = new float[2];
		boolean first=true;
		int cover_range =RandomizeSensors.cover_range;
		int areaThreshold=10;
		//integers holding the start and end coordinates of the controled area
		float distance,startx=0, starty=0, finishx=RandomizeSensors.distanceX,
		finishy=RandomizeSensors.distanceY;
		//array with doubles holding the sensors coordinates
		int[] xySenso = null;
		int[] xySensoOld = null;
		try {
			//Get the coordinates from the file sensor_coordinates.txt
		xySenso = RandomizeSensors.getIntegersFromFile
		("sensor_coordinates.txt");
		} catch (IOException e) {
			e.printStackTrace();
		}
		do{
			//the number of sensors
			int sensor_num=xySenso.length/2-2;
			int threshold=50;
			// list holding the areas with their coverage
			ArrayList areas;
			areas = new ArrayList();
			// start the algorithm with the whole area
			do{
				Area area1 = new Area(startx, starty, finishx, finishy);
				 areas = area1.checkCoverage(areas,
```

```
        xySenso,areaThreshold,threshold);
        threshold-=10;
}while(areas.size()==0 && threshold>=0);
//print all the areas that are uncovered
counter=0;
for(int i=0; i<areas.size();i++){
        if(((Area)areas.get(i)).valid==true &&
        ((Area)areas.get(i)).covered==false){
                counter++;
        }
}
// list holding the areas with their coverage
ArrayList uncoveredArea;
uncoveredArea = new ArrayList();
// start the algorithm with the whole area
if(counter!=0){
       Area area1 = new Area(startx, starty, finishx, finishy);
        uncoveredArea = area1.findUncovered(uncoveredArea,
        areas,areaThreshold);
////////////////////////////////////////
//create shapes
drawingStroke = new BasicStroke(2);
//rectangle for each sensor and the robot
robot=new RoundRectangle2D.Float();
robot.setRoundRect(xySenso[0]-5+50,xySenso[1]-5+50,10, 10, 0, 0
);
sensors= new RoundRectangle2D[sensor_num];
for(int i=0;i<sensor_num;i++){
        sensors[i]=new RoundRectangle2D.Float();
        sensors[i].setRoundRect(xySenso[(i+1)*2]-
        3+50,xySenso[(i+1)*2+1]-3+50,6, 6, 0, 0 );
}
//circle indicating the area covered by each sensor
cover= new RoundRectangle2D[sensor_num];
for(int i=0;i<sensor_num;i++){
        cover[i]=new RoundRectangle2D.Float();
        cover[i].setRoundRect(xySenso[(i+1)*2]-
        cover_range+50,xySenso[(i+1)*2+1]-
        cover_range+50,cover_range*2, cover_range*2,
        cover_range*2, cover_range*2 );
}
double temp1=0, temp2=0;
uncovered= new RoundRectangle2D[counter];
counter=0;
for(int i=0; i<areas.size();i++){
        if(((Area)areas.get(i)).valid==true &&
        ((Area)areas.get(i)).covered==false){
                uncovered[counter]=new RoundRectangle2D.Float
                temp1=(((Area)areas.get(i)).startx+((Area)areas.g
                et(i)).finishx)/2;
                temp2=(((Area)areas.get(i)).starty+((Area)areas.g
                et(i)).finishy)/2;
                uncovered[counter].setRoundRect(temp1-
                +areaThreshold/2+50,temp2-
                areaThreshold/2+50,areaThreshold,areaThreshold,0,
                0);
                counter++;
        }
}
//rectangle indicating the criticaly uncovered area
//double temp1=0, temp2=0;
uncover=new RoundRectangle2D.Float();
if(counter!=0){
        temp1=(((Area)uncoveredArea.get(uncoveredArea.size()-
        1)).startx+((Area)uncoveredArea.get(uncoveredArea.size()
        -1)).finishx)/2;
        temp2=(((Area)uncoveredArea.get(uncoveredArea.size()-
        1)).starty+((Area)uncoveredArea.get(uncoveredArea.size()
        -1)).finishy)/2;
        uncover.setRoundRect(temp1-areaThreshold+50,temp2-
        areaThreshold+50,areaThreshold*2,areaThreshold*2,0,0);
        if(type==1){
                target[0]=(startx+finishx)/2;
                target[1]=(starty+finishy)/2;
        }
        else{
                target[0]=(float) temp1;
```

```
                                target[1]=(float) temp2;
                        }
                        ////////////////////////////////
                        //find next placement
                        float min = RandomizeSensors.distanceX+
                        RandomizeSensors.distanceY;
                        for(int i=0;i<areas.size();i++){
                                if(((Area)areas.get(i)).valid==true &&
                                ((Area)areas.get(i)).covered==false){
                                        temp1=(((Area)areas.get(i)).startx+((Area
                                        )areas.get(i)).finishx)/2;
                                        temp2=(((Area)areas.get(i)).starty+((Area
                                        )areas.get(i)).finishy)/2;
                                       distance=(float)
                                        Math.sqrt(Math.pow(temp2-
                                        target[1],2)+Math.pow(temp1-
                                        target[0],2));
                                       if(distance<min){
                                               min=distance;
                                               nextSensor=i;
                                       }
                                }
                        }
                        //add new sensor
                        xySensoOld = xySenso;
                        xySenso = new int[xySensoOld.length+2];
                        for(int i=0;i<xySensoOld.length-2;i++){
                               xySenso[i]=xySensoOld[i];
                        }
                        temp1=(((Area)areas.get(nextSensor)).startx+((Area)areas
                        .get(nextSensor)).finishx)/2;
                        temp2=(((Area)areas.get(nextSensor)).starty+((Area)areas
                        .get(nextSensor)).finishy)/2;
                        uncover.setRoundRect(temp1-areaThreshold+50,temp2-
                        areaThreshold+50,areaThreshold*2,areaThreshold*2,0,0);
                       xySenso[xySenso.length-4]=(int) temp1;
                       xySenso[xySenso.length-3]=(int) temp2;
                       xySenso[xySenso.length-2]=xySenso[0];
                       xySenso[xySenso.length-1]=xySenso[1];
                }
                else{
                        uncover.setRoundRect(0,0,0,0,0,0);
                }
                if(first){
                        first=false;
                        frame.addWindowListener(new WindowAdapter(){
                                public void windowClosing(WindowEvent we){
                                        System.exit(0);
                                }
                        });
                        frame.setSize(RandomizeSensors.distanceX+100,
                        RandomizeSensors.distanceY+100);
                        frame.setVisible(true);
                }
                else
                        frame.repaint();

                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                System.out.println(counter);
        }while(counter!=0);
    }
}
```

B.3 Γραφική απεικόνιση του πιο σύντομου μονοπατιού

B.3.1 Επιλογή του πιο κοντινού γειτονικού κόμβου

```
public class CheckNeighbors extends Frame{

	private static final long serialVersionUID = 1L;
	//drawing variables
	static Stroke drawingStroke;
	static RoundRectangle2D  robot;
	static RoundRectangle2D[]  sensors;
	static Line2D[]  lines;

	//method for drawing the route of the robot
	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.green);
		for(int i=0;i<lines.length;i++){
			ga.draw(lines[i]);
		}
	}

	public static void main(String[] args) {
		//array of int holding the sensors and robot coordinates
		int[] xySenso = null;
		try {
			//Get the coordinates from the file sensor_coordinates.txt
			 xySenso = RandomizeSensors.getIntegersFromFile
			 ("sensor_coordinates.txt");
		} catch (IOException e) {
			e.printStackTrace();
		}
		//the number of sensors including the robot
		int sensor_num=xySenso.length/2;
		//array holding the shortest path
		int[] shortpath = new int[sensor_num];
		//array indicating if a sensor has been visited
		int[] visited = new int[sensor_num];
		//variable for the last visited sensor
		int lastVisited;
		////////////////////////////
		//Create array with costs
		int cost=0;
		int max=0;
		int min;
		double temp1=0, temp2=0;
		int[][] costs = new int[sensor_num][sensor_num];
		//for each sensor calculate its distance from the rest
		for(int i=0; i<sensor_num; i++){
			for(int j=0; j<sensor_num; j++){
				if(i!=j){
					temp1=xySenso[i*2]-xySenso[j*2];
					temp2=xySenso[i*2+1]-xySenso[j*2+1];
					temp1=temp1*temp1;
					temp2=temp2*temp2;
					temp1=temp1+temp2;
					cost=(int)Math.sqrt(temp1);
					costs[i][j]=cost;
					if(cost>max){
						max=cost;
					}
				}
				else
```

```
                                costs[i][j]=0;
                        }
                }
                ////////////////////////////////////////
                //create shapes
                drawingStroke = new BasicStroke(2);
                //rectangle for each sensor and the robot
                robot=new RoundRectangle2D.Float();
                robot.setRoundRect(xySenso[0]+50,xySenso[1]+50,10, 10, 0, 0 );
                sensors= new RoundRectangle2D[sensor_num];
                for(int i=0;i<sensor_num;i++){
                        sensors[i]=new RoundRectangle2D.Float();
                        sensors[i].setRoundRect(xySenso[(i)*2]+50,xySenso[(i)*2+1]+50,10
                        , 10, 0, 0 );
                }
                //lines for the path
                lines= new Line2D[shortpath.length-1];
                for(int i=0;i<shortpath.length-1;i++){
                        lines[i]=new Line2D.Float();
                        lines[i].setLine(xySenso[shortpath[i]*2]+50,
                        xySenso[shortpath[i]*2+1]+50,xySenso[shortpath[i+1]*2]+50,
                        xySenso[shortpath[i+1]*2+1]+50);
                }
                Frame frame = new CheckNeighbors();
                frame.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
                });
                frame.setSize(600, 600);
                frame.setVisible(true);


                ////////////////////////////////
                //calculate the shortes path
                for(int i=0;i<sensor_num;i++){
                        visited[i]=0;
                }
                visited[0]=1;
                shortpath[0]=0;
                lastVisited=0;
                int next = 0,count=1;
                while(count<sensor_num){
                        min=max*5;
                        for(int i=0; i<sensor_num; i++){
                                if(visited[i]== 0){
                                        if(i==sensor_num-1 && count!=sensor_num-1){}
                                        else {
                                                if(min>costs[lastVisited][i]){
                                                        min=costs[lastVisited][i];
                                                        next=i;
                                                }
                                        }
                                }
                        }

                        shortpath[count]=next;
                        lines[count-1]=new Line2D.Float();
                        lines[count-1].setLine(xySenso[shortpath[count-1]*2]+50,
                        xySenso[shortpath[count-
                        1]*2+1]+50,xySenso[shortpath[count]*2]+50,
                        xySenso[shortpath[count]*2+1]+50);
                        visited[next]=1;
                        lastVisited=next;
                        count++;
                        try {
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        frame.repaint();
                }
        }
}
```

B.3.2 Υπολογισμός όλων των μονοπατιών

```
public class CheckAllRoutes extends Frame{

	private static final long serialVersionUID = 1L;
	//drawing variables
	static Stroke drawingStroke;
	static RoundRectangle2D  robot;
	static RoundRectangle2D[]  sensors;
	static Line2D[]  lines;

	//method for drawing the route of the robot
	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.green);
		for(int i=0;i<lines.length;i++){
			ga.draw(lines[i]);
		}
	}

	public static void main(String[] args) {
		//array of int holding the sensors and robot coordinates
		int[] xySenso = null;
		try {
			//Get the coordinates from the file sensor_coordinates.txt
			xySenso = RandomizeSensors.getIntegersFromFile
			("sensor_coordinates.txt");
		} catch (IOException e) {
			e.printStackTrace();
		}
		//the number of sensors including the robot
		int sensor_num=xySenso.length/2;
		//the number of posible route combinations

		if(sensor_num-2>9){
			System.out.println("too many sensors");
			return;
		}

		int comb=sensor_num-2;
		for(int i=sensor_num-3;i>0;i--){
			comb*=i;
		}
		//array holding the shortest path
		int[] shortpath = new int[sensor_num];

		////////////////////////////
		//Create array with costs
		int cost=0;
		int max=0;
		int min;
		double temp1=0, temp2=0;
		int[][] costs = new int[sensor_num][sensor_num];
		//for each sensor calculate its distance from the rest
		for(int i=0; i<sensor_num; i++){
			for(int j=0; j<sensor_num; j++){
				if(i!=j){
					temp1=xySenso[i*2]-xySenso[j*2];
					temp2=xySenso[i*2+1]-xySenso[j*2+1];
					temp1=temp1*temp1;
					temp2=temp2*temp2;
					temp1=temp1+temp2;
					cost=(int)Math.sqrt(temp1);
					costs[i][j]=cost;
					if(cost>max){
						max=cost;
					}
				}
```

```
                else
                        costs[i][j]=0;
        }
}
////////////////////////////////
//create all possible paths
int temp = 0,temp3,temp4,temp5;
int k;boolean check = false;
int[][] path = new int[comb][sensor_num-2];
temp3=sensor_num-2;
temp5=comb;
for(int i=0;i<sensor_num-4;i++){
        temp=0;
        temp4=0;
        for(int j=1;j<=comb;j++){
                if(temp==sensor_num-2)
                        temp=0;
                temp++;
                for(k=0;k<(temp5/temp3);k++){
                        //check if is visited
                        do{
                                check=true;
                                for(int v=0;v<i;v++){
                                        if(path[temp4][v]==temp){
                                                check=false;break;
                                        }
                                }
                                if(check==true){
                                        path[temp4][i]=temp;
                                        temp4++;
                                }
                                else{
                                        if(temp==sensor_num-2)
                                                temp=0;
                                                temp++;
                                }
                        }while(check==false);
                }
                j=temp4;
        }
        temp5=temp5/temp3;
        temp3-=1;
}
if(sensor_num>3){
        int count=0;
        for(int j=sensor_num-4; j<sensor_num-2; j++){
                for(int i=0; i<comb; i++){
                        do{
                                if(count==sensor_num-2)
                                        count=0;
                                count++;
                                check=true;
                                for(int v=0;v<j;v++){
                                        if(path[i][v]==count){
                                                check=false;break;
                                        }
                                }
                                if(check==true){
                                        path[i][j]=count;
                                }
                        }while(check==false);
                }
        }
}

////////////////////////////////////////
//create shapes
drawingStroke = new BasicStroke(2);
//rectangle for each sensor and the robot
robot=new RoundRectangle2D.Float();
robot.setRoundRect(xySenso[0]+50,xySenso[1]+50,10, 10, 0, 0 );
sensors= new RoundRectangle2D[sensor_num-2];
for(int i=1;i<sensor_num-1;i++){
        sensors[i-1]=new RoundRectangle2D.Float();
        sensors[i-1].setRoundRect(xySenso[(i)*2]+50,
        xySenso[(i)*2+1]+50,10, 10, 0, 0 );
}
```

```
		//lines for the path
		lines= new Line2D[sensor_num-1];
		for(int i=0;i<sensor_num-1;i++){
			lines[i]=new Line2D.Float();
			 lines[i].setLine(xySenso[shortpath[i]*2]+50,
			 xySenso[shortpath[i]*2+1]+50,xySenso[shortpath[i+1]*2]+50,
			 xySenso[shortpath[i+1]*2+1]+50);
		}
		Frame frame = new CheckAllRoutes();
		frame.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent we){
				System.exit(0);
			}
		});
		frame.setSize(600, 600);
		frame.setVisible(true);

		////////////////////////////////////////
		//find the shortest path
		if(sensor_num>3){
			int[] distance =new int[comb];
			for(int i=0;i<comb;i++){
				distance[i]=costs[0][path[i][0]];
				for(int j=0;j<sensor_num-3;j++){
					distance[i]+=costs[path[i][j]][path[i][j+1]];
				}
				distance[i]+=costs[path[i][sensor_num-3]][sensor_num-1];
			}
			min=sensor_num*50000;
			for(int i=0; i<comb; i++){
				if(min>distance[i]){
					min=distance[i];
					temp=i;
				}
			}
		}
		shortpath[0]=0;

		for(int j=1;j<sensor_num-1;j++){
			shortpath[j]=path[temp][j-1];

		}
		shortpath[sensor_num-1]=sensor_num-1;

		if(sensor_num-2==1){
			shortpath[1]=1;
		}

		for(int j=1;j<sensor_num;j++){
			lines[j-1]=new Line2D.Float();
			 lines[j-1].setLine(xySenso[shortpath[j-1]*2]+50,
			 xySenso[shortpath[j-1]*2+1]+50,xySenso[shortpath[j]*2]+50,
			 xySenso[shortpath[j]*2+1]+50);

			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
			frame.repaint();}
	}


}
```

B.4 Εντοπισμός στόχου

B.4.1 Καθορισμός αρχικής θέσης στόχου και μετρήσεις αισθητήρων

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


public static void initializeTarget(String fileName) throws IOException {
	float reading;
	////////////////////////////////////////////////
	//read the sensor coordinates
	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());
	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();
	}
	////////////////////////////////////////////////
	//set a random position for the target
	targetx=(int)(Math.random()*500);
	targety=(int)(Math.random()*500);
	System.out.println(targetx + "  " + targety);
	System.out.println();
	fileName="sensor_readings.txt";
	BufferedWriter writerr = new BufferedWriter(new FileWriter(fileName));
	writerr.flush();
	//coordinates of the robot
	writerr.write(array[0] +" " +array[1]);
	writerr.newLine();
	//add the sensors in random places inside the area
	for(int i=2; i<array.length-2 ; i+=2){
		//calculate the reading according to the distance
		reading=(float) Math.sqrt(Math.pow(targetx-array[i], 2) +
		Math.pow(targety-array[i+2], 2));
		//write the coordinates in the text file
		writerr.write(array[i] + " " + array[i+1] + " " + (int)reading);
		writerr.newLine();
	}
	//display the actual number of sensors added in the area
	writerr.close();
}
```

B.4.2 Ανανέωση θέσης στόχου μέσα στο αρχείο

```
public static void updateSensorReadings(String fileName) throws IOException {
        float reading,tempx= 0,tempy = 0;
        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());
        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();
        }
        do{
                if(dir==1){
                        tempx=(float) 7.07;
                        tempy=(float) 7.07;
                }
                else if(dir==2){
                        tempx=(float) -7.07;
                        tempy=(float) 7.07;
                }
                else if(dir==3){
                        tempx=(float) -7.07;
                        tempy=(float) -7.07;
                }
                else if(dir==4){
                        tempx=(float) 7.07;
                        tempy=(float) -7.07;
                }
                if(tempx+targetx>500 || tempx+targetx<0 || tempy+targety>500 ||
                tempy+targety<0){
                        if(dir==4)
                                dir=0;
                        dir++;
                }
        }while(tempx+targetx>500 || tempx+targetx<0 || tempy+targety>500 ||
        tempy+targety<0);
        targetx+=tempx;
        targety+=tempy;
        BufferedWriter writerr = new BufferedWriter(new FileWriter(fileName));
        writerr.flush();
        //coordinates of the robot
        writerr.write(array[0] +" " +array[1]);
        writerr.newLine();
        //add the sensors in random places inside the area
        for(int i=2; i<array.length ; i+=3){
                //write the coordinates in the text file
                reading=(float) Math.sqrt(Math.pow(targetx-array[i], 2) +
                Math.pow(targety-array[i+2], 2));
                writerr.write(array[i] + " " + array[i+1] + " " + (int)reading);
                writerr.newLine();
        }
        //display the actual number of sensors added in the area
        writerr.close();
}
```

B.4.3 Γραφική απεικόνιση εύρεσης και ακολουθίας στόχου

```
public class FindTarget  extends Frame {

        private static final long serialVersionUID = 1L;
        //drawing variables
        static Stroke drawingStroke;
        static RoundRectangle2D  robot;
        static RoundRectangle2D[]  sensors;
        static RoundRectangle2D  target;
        static RoundRectangle2D  close_sensor;
        //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.green);
                ga.draw(target);
                ga.setPaint(Color.blue);
                ga.draw(close_sensor);
        }

        public static void main(String[] args) {
                int speed=5;
                //array with int holding the sensors coordinates
                int[] senso = null,temp={0,0,0};
                int max,index = 0;
                float targetx,targety,robotX,robotY,a;
                try {
                        //Get the coordinates from the file sensor_coordinates.txt
                        senso = RandomizeSensors.getIntegersFromFile
                        ("sensor_readings.txt");
                } catch (IOException e) {
                        e.printStackTrace();
                }
                robotX=senso[0];
                robotY=senso[1];
                try {
                        updateTargetPosition.initializeTarget("sensor_coordinates.txt");
                        senso = RandomizeSensors.getIntegersFromFile
                        ("sensor_readings.txt");
                } catch (IOException e) {
                        e.printStackTrace();
                }
                ////////////////////////////////////////
                //create shapes
                drawingStroke = new BasicStroke(2);
                //rectangle for each sensor and the robot
                robot=new RoundRectangle2D.Float();
                robot.setRoundRect(robotX+50,robotY+50,10, 10, 0, 0 );
                sensors= new RoundRectangle2D[senso.length/3];
                int count=0;
                for(int i=2; i<senso.length;i+=3){
                        sensors[count]=new RoundRectangle2D.Float();
                        sensors[count].setRoundRect(senso[(i)]+50,
                                        senso[(i)+1]+50,10, 10, 0, 0 );
                        count++;
                }
                //target
                target=new RoundRectangle2D.Float();
                target.setRoundRect(updateTargetPosition.targetx+50,
                                updateTargetPosition.targety+50,10, 10, 0, 0 );
                close_sensor=new RoundRectangle2D.Float();
                close_sensor.setRoundRect(0,0,0,0, 0, 0 );
                Frame frame = new FindTarget();
                frame.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
                });
```

```
                frame.setSize(600, 600);
                frame.setVisible(true);
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e1) {
                        e1.printStackTrace();
                }
                count=0;
                do{
                        count++;
                        if(count==100){
                                speed+=5;
                                count=0;
                        }
                        try {
                                Thread.sleep(200);
                                updateTargetPosition.updateSensorReadings
                                ("sensor_readings.txt");
                                //Get the coordinates from the file
                                sensor_coordinates.txt
                                senso = RandomizeSensors.getIntegersFromFile
                                ("sensor_readings.txt");
                        } catch (IOException e) {
                                e.printStackTrace();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        targetx=0;
                        targety=0;
                        for(int j=0;j<3;j++){
                                max=0;
                                for(int i=2; i<senso.length;i+=3){
                                        if(max<senso[i+2] && i!=temp[0] && i!=temp[1]){
                                                max=senso[i+2];
                                                temp[j]=i;
                                                index=i;
                                        }
                                }
                                targetx+=senso[index];
                                targety+=senso[index+1];
                        }
                        targetx/=3;
                        targety/=3;
                        a=(targety-robotY)/(targetx-robotX);
                        a=(float) Math.atan(a);
                        if(targety>robotY && targetx>robotX ||
                           targety<robotY && targetx>robotX){
                        robotX+=(float) (Math.cos(a)*speed);
                        robotY+=(float) (Math.sin(a)*speed);
                        }
                        else if(targety<robotY && targetx<robotX ||
                                 targety>robotY && targetx<robotX){
                                robotX-=(float) (Math.cos(a)*speed);
                                robotY-=(float) (Math.sin(a)*speed);
                        }
                        //robot
                        robot=new RoundRectangle2D.Float();
                        robot.setRoundRect(robotX+50,robotY+50,10, 10, 0, 0 );
                        //target
                        target=new RoundRectangle2D.Float();
                        target.setRoundRect(updateTargetPosition.targetx+50,
                                        updateTargetPosition.targety+50,10, 10, 0, 0 );
                        //robot
                        close_sensor=new RoundRectangle2D.Float();
                        close_sensor.setRoundRect(targetx+50,targety+50,10, 10, 0, 0 );
                        frame.repaint();
                }while(Math.abs(robotX-updateTargetPosition.targetx)>20 ||
                        Math.abs(robotY-updateTargetPosition.targety)>20);
        }
}
```

B.4.4 Γραφική απεικόνιση εύρεσης και ακολουθίας μονοπατιού στόχου

```
public class FindTargetRoot  extends Frame {

        private static final long serialVersionUID = 1L;
        //drawing variables
        static Stroke drawingStroke;
        static RoundRectangle2D  robot;
        static RoundRectangle2D[]  sensors;
        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.green);
                ga.draw(target);

        }

        public static void main(String[] args) {
                int root_length=50,path_size=root_length;
                float[] pathX =new float[path_size];
                float[] pathY =new float[path_size];
                int next;
                float robotX,robotY,a,targetx,targety;
                int max,index = 0;
                //array with doubles holding the sensors coordinates
                int[] senso = null, temp={0,0,0};

                try {
                        //Get the coordinates from the file sensor_coordinates.txt
                        senso = RandomizeSensors.getIntegersFromFile
                        ("sensor_readings.txt");
                } catch (IOException e) {
                        e.printStackTrace();
                }
                robotX=senso[0];
                robotY=senso[1];
                try {
                        updateTargetPosition.initializeTarget("sensor_coordinates.txt");
                        senso = RandomizeSensors.getIntegersFromFile
                        ("sensor_readings.txt");
                } catch (IOException e) {
                        e.printStackTrace();
                }
                ////////////////////////////////////////
                //create shapes
                drawingStroke = new BasicStroke(2);
                //rectangle for each sensor and the robot
                robot=new RoundRectangle2D.Float();
                robot.setRoundRect(robotX+50,robotY+50,10, 10, 0, 0 );
                sensors= new RoundRectangle2D[senso.length/3];
                int count=0;
                for(int i=2; i<senso.length;i+=3){
                        sensors[count]=new RoundRectangle2D.Float();
                        sensors[count].setRoundRect(senso[(i)]+50,senso[(i)+1]+50,10,
                        10, 0, 0 );
                        count++;
                }
                //target
                target=new RoundRectangle2D.Float();
                target.setRoundRect(updateTargetPosition.targetx+50,updateTargetPositio
                n.targety+50,10, 10, 0, 0 );
                Frame frame = new FindTargetRoot();
                frame.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
```

```
		});
		frame.setSize(600, 600);
		frame.setVisible(true);
		try {
			Thread.sleep(1000);
		} catch (InterruptedException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		count=0;
		next=0;
		do{
			try {
				Thread.sleep(100);
			} catch (InterruptedException e1) {
				e1.printStackTrace();
			}
			if(count<root_length){
				try {
	updateTargetPosition.updateSensorReadings("sensor_readings.txt");
		//Get the coordinates from the file sensor_coordinates.txt
	senso = RandomizeSensors.getIntegersFromFile("sensor_readings.txt");
				} catch (IOException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
				targetx=0;
				targety=0;
				for(int j=0;j<3;j++){
					max=0;
					for(int i=2; i<senso.length;i+=3){
					if(max<senso[i+2] && i!=temp[0] && i!=temp[1]){
							max=senso[i+2];
							temp[j]=i;
							index=i;
						}
					}
					targetx+=senso[index];
					targety+=senso[index+1];
				}
				targetx/=3;
				targety/=3;
				 if(count>0 && pathX[count-1]==senso[index] &&
				 pathY[count-1]==senso[index+1]){}
				else{
					pathX[count]=targetx;
					pathY[count]=targety;
					count++;
				}
			}
			a=(pathY[next]-robotY)/(pathX[next]-robotX);
			a=(float) Math.atan(a);
			if(pathY[next]>robotY && pathX[next]>robotX ||
			pathY[next]<robotY && pathX[next]>robotX ){
				robotX+=(float) (Math.cos(a)*10);
				robotY+=(float) (Math.sin(a)*10);
			}
			else if(pathY[next]<robotY && pathX[next]<robotX ||
			pathY[next]>robotY && pathX[next]<robotX){
				robotX-=(float) (Math.cos(a)*10);
				robotY-=(float) (Math.sin(a)*10);
			}
			if(Math.abs(pathY[next]-robotY)<10 && Math.abs(pathY[next]-
			robotY)<10){
				next++;
			}
			//robot
			robot=new RoundRectangle2D.Float();
			robot.setRoundRect(robotX+50,robotY+50,10, 10, 0, 0 );
			//target
			target=new RoundRectangle2D.Float();
			 target.setRoundRect(updateTargetPosition.targetx+50,updateTarget
			 Position.targety+50,10, 10, 0, 0 );
			frame.repaint();
		}while(next<root_length-1);
	}
}
```

Παράρτημα Γ

Σε αυτό το παράρτημα έχουμε όλους τους κώδικες που γράφτηκαν στο TinyOS και απευθύνονταν για τους κόμβους MicaZ..

C.1 Αναπαράσταση χρονομέτρου με τις λάμπες του MicaZ

Αρχείο “BlinkAppC.nc”

```
#Δήλωση ότι αυτό είναι configuration αρχείο με το όνομα BlinkAppC
configuration BlinkAppC {
}

#Η υλοποίηση του configuration
implementation {
        #Τα συστατικά που χρησιμοποιεί
        components MainC, BlinkC, LedsC;
        components new TimerMilliC() as Timer0;
        components new TimerMilliC() as Timer1;
        components new TimerMilliC() as Timer2;

        #Ένωση των συστατικών με τα interface του αρχείου BlinkC.
        #Είναι ένα είδος καλωδίωσης μέσα στην εφαρμογή
        BlinkC -> MainC.Boot;
        BlinkC.Timer0 -> Timer0;
        BlinkC.Timer1 -> Timer1;
        BlinkC.Timer2 -> Timer2;
        BlinkC.Leds -> LedsC;
}
```

Αρχείο “BlinkC.nc”

```
#include "Timer.h"

#Δήλωση ότι αυτό είναι ένα module αρχείο με το όνομα BlinkC
module BlinkC @safe(){
   #Δήλωση όλων των interface που χρησιμοποιεί
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
#Η υλοποίηση του module
implementation{
#καθορισμός τις περιόδου κάθε Timer σε milliseconds
  event void Boot.booted(){
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }
   #κάθε φορά που ένα Timer “πυροβολεί”, καλείται το αντίστοιχο
   #event το οποίο γράφει το χρόνο και αναβοσβήνει το κατάλληλο LED
   event void Timer0.fired(){
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    call Leds.led0Toggle();
  }
  event void Timer1.fired(){
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
    call Leds.led1Toggle();
  }
  event void Timer2.fired(){
    dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
    call Leds.led2Toggle();
  }
}
```

C.2 Μέτρηση φωτός από τους αισθητήρες

Αρχείο “Makefile”

```
# Use Surge for micaz
PLATFORMS=mica mica2 mica2dot micaz pc
COMPONENT=Surge

SENSORBOARD=mts400

PFLAGS= -I%T/lib/Route -I%T/lib/Queue -I%T/lib/Broadcast

include ../Makerules
```

Αρχείο “Surge.nc”

```
includes Surge;
includes SurgeCmd;
includes MultiHop;

configuration Surge {
}
implementation {
  components Main, SurgeM, TimerC, LedsC, NoLeds, RandomLFSR,
    GenericCommPromiscuous as Comm, Bcast, MultiHopRouter as multihopM,
QueuedSend,TaosPhoto/*, Sounder*/;

  Main.StdControl -> SurgeM.StdControl;
  //Main.StdControl -> Photo;
  Main.StdControl -> Bcast.StdControl;
  Main.StdControl -> multihopM.StdControl;
  Main.StdControl -> QueuedSend.StdControl;
  Main.StdControl -> TimerC;
  Main.StdControl -> Comm;

  //to create
  SurgeM.photoControl ->TaosPhoto;
  //connect to the  light sensor of mts400
  SurgeM.ADC -> TaosPhoto.ADC[0];

  //  multihopM.CommControl -> Comm;

  //SurgeM.ADC   -> Photo;
  SurgeM.Timer -> TimerC.Timer[unique("Timer")];
  SurgeM.Leds  -> LedsC; // NoLeds;
  //SurgeM.Sounder -> Sounder;

  SurgeM.Bcast -> Bcast.Receive[AM_SURGECMDMSG];
  Bcast.ReceiveMsg[AM_SURGECMDMSG] -> Comm.ReceiveMsg[AM_SURGECMDMSG];

  SurgeM.RouteControl -> multihopM;
  SurgeM.Send -> multihopM.Send[AM_SURGEMSG];
  multihopM.ReceiveMsg[AM_SURGEMSG] -> Comm.ReceiveMsg[AM_SURGEMSG];
  //multihopM.ReceiveMsg[AM_MULTIHOPMSG] -> Comm.ReceiveMsg[AM_MULTIHOPMSG];
}
```

Αρχείο “SurgeM.nc”

```
includes Surge;
includes SurgeCmd;

/*
 *  Data gather application
 */
module SurgeM {
  provides {
    interface StdControl;
  }
```

```
  uses {
    interface ADC;
    interface Timer;
    interface Leds;
    //interface StdControl as Sounder;
    interface Send;
    interface Receive as Bcast;
    interface RouteControl;

       //
       interface SplitControl as photoControl;
    //interface ADC as photo;
  }
}
implementation {

  enum {
    TIMER_GETADC_COUNT = 1,            // Timer ticks for ADC
    TIMER_CHIRP_COUNT = 10,            // Timer on/off chirp count
  };

  bool sleeping;                     // application command state
  bool focused;
  bool rebroadcast_adc_packet;

  TOS_Msg gMsgBuffer;
  norace uint16_t gSensorData;              // protected by gfSendBusy flag
  bool gfSendBusy;


  int timer_rate;
  int timer_ticks;
  /*************************************************************************
   * Initialization
   *************************************************************************/

  static void initialize() {
    timer_rate = INITIAL_TIMER_RATE;
    atomic gfSendBusy = FALSE;
    sleeping = FALSE;
    rebroadcast_adc_packet = FALSE;
    focused = FALSE;
  }

  task void SendData() {
    SurgeMsg *pReading;
    uint16_t Len;
    dbg(DBG_USR1, "SurgeM: Sending sensor reading\n");

    if (pReading = (SurgeMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) {
      pReading->type = SURGE_TYPE_SENSORREADING;
      pReading->parentaddr = call RouteControl.getParent();
      pReading->reading = gSensorData;

      if ((call Send.send(&gMsgBuffer,sizeof(SurgeMsg))) != SUCCESS)
        atomic gfSendBusy = FALSE;
    }

  }

  command result_t StdControl.init() {
  call photoControl.init();
    call Leds.init();
    initialize();
    return SUCCESS;
  }

  command result_t StdControl.start() {
    call photoControl.start();
    return call Timer.start(TIMER_REPEAT, timer_rate);
    return SUCCESS;
  }

  command result_t StdControl.stop() {
   call photoControl.stop();
    return call Timer.stop();
  }
```

```
  /*****************************************************************
   * Commands and events
   *****************************************************************/
  event result_t Timer.fired() {
    dbg(DBG_USR1, "SurgeM: Timer fired\n");
    timer_ticks++;
    if (timer_ticks % TIMER_GETADC_COUNT == 0) {
      call ADC.getData();
    }
    // If we're the focused node, chirp
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
      //call Sounder.start();
    }
    // If we're the focused node, chirp
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
     // call Sounder.stop();
    }
    return SUCCESS;
  }

  async event result_t ADC.dataReady(uint16_t data) {
    //SurgeMsg *pReading;
    //uint16_t Len;
    dbg(DBG_USR1, "SurgeM: Got ADC reading: 0x%x\n", data);
    atomic {
      if (!gfSendBusy) {
       gfSendBusy = TRUE;
       gSensorData = data;
       post SendData();
      }
    }
    return SUCCESS;
  }

  event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
    dbg(DBG_USR2, "SurgeM: output complete 0x%x\n", success);
    //call Leds.greenToggle();
    atomic gfSendBusy = FALSE;
    return SUCCESS;
  }


  /* Command interpreter for broadcasts
   *
   */
  event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload, uint16_t payloadLen)
{
    SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload;

    dbg(DBG_USR2, "SurgeM: Bcast  type 0x%02x\n", pCmdMsg->type);

    if (pCmdMsg->type == SURGE_TYPE_SETRATE) {       // Set timer rate
      timer_rate = pCmdMsg->args.newrate;
      dbg(DBG_USR2, "SurgeM: set rate %d\n", timer_rate);
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, timer_rate);

    } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
      // Go to sleep - ignore everything until a SURGE_TYPE_WAKEUP
      dbg(DBG_USR2, "SurgeM: sleep\n");
      sleeping = TRUE;
      call Timer.stop();
      call Leds.greenOff();
      call Leds.yellowOff();

    } else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
      dbg(DBG_USR2, "SurgeM: wakeup\n");

      // Wake up from sleep state
      if (sleeping) {
       initialize();
        call Timer.start(TIMER_REPEAT, timer_rate);
       sleeping = FALSE;
      }

    } else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
      dbg(DBG_USR2, "SurgeM: focus %d\n", pCmdMsg->args.focusaddr);
```

```
      // Cause just one node to chirp and increase its sample rate;
      // all other nodes stop sending samples (for demo)
      if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
        // OK, we're focusing on me
        focused = TRUE;
        //call Sounder.init();
        call Timer.stop();
        call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
      } else {
        // Focusing on someone else
        call Timer.stop();
        call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
      }

    } else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
      // Return to normal after focus command
      dbg(DBG_USR2, "SurgeM: unfocus\n");
      focused = FALSE;
      //call Sounder.stop();
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, timer_rate);
    }
    return pMsg;
  }

  event result_t photoControl.startDone() {
      return SUCCESS;
  }
  event result_t photoControl.initDone() {
      return SUCCESS;
  }
  event result_t photoControl.stopDone() {
      return SUCCESS;
  }
}
```

C.3 Συλλογή μετρήσεων από όλους τους αισθητήρες

Αρχείο “TOSbase.nc”

```
configuration TOSBase {
}
implementation {
  components Main, TOSBaseM, RadioCRCPacket as Comm, FramerM, UART,
LedsC,CC2420RadioC;

  Main.StdControl -> TOSBaseM;

  TOSBaseM.UARTControl -> FramerM;
  TOSBaseM.UARTSend -> FramerM;
  TOSBaseM.UARTReceive -> FramerM;
  TOSBaseM.UARTTokenReceive -> FramerM;

  TOSBaseM.CC2420Control -> CC2420RadioC.CC2420Control;


  TOSBaseM.RadioControl -> Comm;
  TOSBaseM.RadioSend -> Comm;
  TOSBaseM.RadioReceive -> Comm;

  TOSBaseM.Leds -> LedsC;

  FramerM.ByteControl -> UART;
  FramerM.ByteComm -> UART;
}
```

Αρχείο “TOSbaseM.nc”

```
#ifndef TOSBASE_BLINK_ON_DROP
#define TOSBASE_BLINK_ON_DROP
#endif

module TOSBaseM {
  provides interface StdControl;
  uses {
    interface StdControl as UARTControl;
    interface BareSendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;
    interface TokenReceiveMsg as UARTTokenReceive;
    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;

    interface Leds;

        //interface gia miwsi tou transmission power
        interface CC2420Control;
  }
}

implementation
{
  enum {
    UART_QUEUE_LEN = 12,
    RADIO_QUEUE_LEN = 12,
  };

  TOS_Msg    uartQueueBufs[UART_QUEUE_LEN];
  uint8_t    uartIn, uartOut;
  bool       uartBusy, uartCount;
  TOS_Msg    radioQueueBufs[RADIO_QUEUE_LEN];
  uint8_t    radioIn, radioOut;
  bool       radioBusy, radioCount;
  task void UARTSendTask();
  task void RadioSendTask();

  void failBlink();
  void dropBlink();
  void processUartPacket(TOS_MsgPtr Msg, bool wantsAck, uint8_t Token);

  command result_t StdControl.init() {
    result_t ok1, ok2, ok3;

    uartIn = uartOut = uartCount = 0;
    uartBusy = FALSE;
    radioIn = radioOut = radioCount = 0;
    radioBusy = FALSE;
    ok1 = call UARTControl.init();
    ok2 = call RadioControl.init();
    ok3 = call Leds.init();
    dbg(DBG_BOOT, "TOSBase initialized\n");
    return rcombine3(ok1, ok2, ok3);
  }

  command result_t StdControl.start() {
    result_t ok1, ok2;
    call CC2420Control.SetRFPower(3);
    ok1 = call UARTControl.start();
    ok2 = call RadioControl.start();
    return rcombine(ok1, ok2);
  }

  command result_t StdControl.stop() {
    result_t ok1, ok2;
    ok1 = call UARTControl.stop();
    ok2 = call RadioControl.stop();
    return rcombine(ok1, ok2);
  }
```

```
event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr Msg) {
  dbg(DBG_USR1, "TOSBase received radio packet.\n");
  if ((!Msg->crc) || (Msg->group != TOS_AM_GROUP))
    return Msg;
  if (uartCount < UART_QUEUE_LEN) {
    memcpy(&uartQueueBufs[uartIn], Msg, sizeof(TOS_Msg));
    uartCount++;
    if( ++uartIn >= UART_QUEUE_LEN ) uartIn = 0;
    if (!uartBusy) {
     if (post UARTSendTask()) {
       uartBusy = TRUE;
     }
    }
  } else {
    dropBlink();
  }
  return Msg;
}

task void UARTSendTask() {
  dbg (DBG_USR1, "TOSBase forwarding Radio packet to UART\n");
  if (uartCount == 0) {
    uartBusy = FALSE;
  } else {
    if (call UARTSend.send(&uartQueueBufs[uartOut]) == SUCCESS) {
     call Leds.greenToggle();
    } else {
     failBlink();
     post UARTSendTask();
    }
  }
}

event result_t UARTSend.sendDone(TOS_MsgPtr msg, result_t success) {
  if (!success) {
    failBlink();
  } else {
    uartCount--;
    if( ++uartOut >= UART_QUEUE_LEN ) uartOut = 0;
  }
  post UARTSendTask();
  return SUCCESS;
}

event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr Msg) {
  processUartPacket(Msg, FALSE, 0);
  return Msg;
}

event TOS_MsgPtr UARTTokenReceive.receive(TOS_MsgPtr Msg, uint8_t Token) {
  processUartPacket(Msg, TRUE, Token);
  return Msg;
}

void processUartPacket(TOS_MsgPtr Msg, bool wantsAck, uint8_t Token) {
  bool reflectToken = FALSE;
  dbg(DBG_USR1, "TOSBase received UART token packet.\n");
  if (radioCount < RADIO_QUEUE_LEN) {
    reflectToken = TRUE;
    memcpy(&radioQueueBufs[radioIn], Msg, sizeof(TOS_Msg));
    radioCount++;
    if( ++radioIn >= RADIO_QUEUE_LEN ) radioIn = 0;
    if (!radioBusy) {
     if (post RadioSendTask()) {
       radioBusy = TRUE;
     }
    }
  } else {
    dropBlink();
  }
  if (wantsAck && reflectToken) {
    call UARTTokenReceive.ReflectToken(Token);
  }
}
```

```
  task void RadioSendTask() {
    dbg(DBG_USR1, "TOSBase forwarding UART packet to Radio\n");
    if (radioCount == 0) {
      radioBusy = FALSE;
    } else {
      radioQueueBufs[radioOut].group = TOS_AM_GROUP;
      if (call RadioSend.send(&radioQueueBufs[radioOut]) == SUCCESS) {
       call Leds.redToggle();
      } else {
       failBlink();
       post RadioSendTask();
      }
    }
  }

  event result_t RadioSend.sendDone(TOS_MsgPtr msg, result_t success) {
    if (!success) {
      failBlink();
    } else {
      radioCount--;
      if( ++radioOut >= RADIO_QUEUE_LEN ) radioOut = 0;
    }
    post RadioSendTask();
    return SUCCESS;
  }

  void dropBlink() {
#ifdef TOSBASE_BLINK_ON_DROP
    call Leds.yellowToggle();
#endif
  }

  void failBlink() {
#ifdef TOSBASE_BLINK_ON_FAIL
    call Leds.yellowToggle();
#endif
  }
}
```