ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΡΟΜΠΟΤ
ΓΙΑ ΑΥΤΟΝΟΜΗ ΠΕΡΙΗΓΗΣΗ ΣΤΟ ΧΩΡΟ

Της φοιτήτριας

Κωνσταντίνας Χριστοφορίδου

Αρ. Μητρώου: 06/2990

Επιβλέπων καθηγητής

κ. Κωνσταντίνος Διαμαντάρας

Θεσσαλονίκη 2012

ΠΡΟΛΟΓΟΣ

Η παρούσα εργασία συνιστά μία εισαγωγή στον γεμάτο ενδιαφέρον κόσμο των ρομπότ. Η ιδέα για τη δημιουργία αυτόνομων μηχανημάτων που θα εκτελούν εργασίες στη θέση των ανθρώπων έχει τις ρίζες στα αρχαία ακόμη χρόνια. Στην εποχή μας, τα ρομπότ υπάρχουν στη ζωή μας και χρησιμοποιούνται καθημερινά καλύπτοντας πολλές από τις ανάγκες μας. Αν και τα ρομπότ υπάρχουν γύρω μας, είναι λίγες οι φορές που έχουμε τη δυνατότητα αντί να τα χρησιμοποιήσουμε απλά, να τα κατασκευάσουμε και να τα προγραμματίσουμε όπως θέλουμε εμείς. Η εταιρία Lego κάνει πραγματικότητα αυτή την επιθυμία πολλών φίλων της ρομποτικής με το ρομπότ της, το NXT.

Σκοπός της εργασίας μας είναι η κατασκευή και ο προγραμματισμός ενός ρομποτικού συστήματος, με τη χρήση του εκπαιδευτικού παιχνιδιού Lego Mindstorms. Η ρομποτική κατασκευή θα έχει τη δυνατότητα να περιφέρεται αυτόνομα στο χώρο, αποφεύγοντας τα τεχνητά εμπόδια που θα τοποθετούνται τυχαία σ’ αυτόν αντιδρώντας με συγκεκριμένο τρόπο ανάλογα με το χρώμα του εμποδίου.

Το ρομπότ της Lego είναι ένα υπέροχο εργαλείο στα χέρια οποιουδήποτε έχει μεγάλη φαντασία και του αρέσει να δημιουργεί. Το γεγονός ότι μπορούμε να ενώσουμε απλά κομμάτια και να δημιουργήσουμε ότι θέλουμε είναι από μόνο του πολύ ενδιαφέρον. Το γεγονός όμως ότι μας δίνεται η δυνατότητα να κάνουμε τις δημιουργίες μας να κινούνται και να αντιδρούν σε διάφορα ερεθίσματα είναι καταπληκτικό.

Δεν είναι τυχαίο το γεγονός, ότι ενώ το θέμα της πτυχιακής μου ήταν σαφές, και το ρομπότ θα έπρεπε να έχει τη δυνατότητα να κινείται, αρχικά κατασκεύασα αρκετά στατικά ρομπότ ελέγχοντας τις πάρα πολλές δυνατότητες που έχει το NXT. Έπειτα έφτιαξα πρώτα ένα αυτοκίνητο, ένα τανκ, ένα κροκόδειλο και ένα ρομπότ που έμοιαζε με άνθρωπο μέχρι να καταλήξω ότι το ρομπότ μου θα είναι τελικά ένας σκύλος. Ο χρόνος και ο κόπος που απαιτούνται για την κατασκευή και τον προγραμματισμό ενός ρομπότ δεν είναι τίποτα σε σχέση με την αίσθηση ικανοποίησης και ευχαρίστησης που παίρνουμε όταν βλέπουμε τη σκέψη μας να γίνεται πραγματικότητα.

ΠΕΡΙΛΗΨΗ

Η πτυχιακή εργασία αυτή πραγματεύεται τη δημιουργία ενός ρομποτικού συστήματος με τη χρήση του εκπαιδευτικού ρομπότ της Lego, του Lego Mindstorms NXT 2.0 και τον προγραμματισμό του με στόχο την υλοποίηση ενός συγκεκριμένου προγράμματος. Ξεκινάει με μία γενική περιγραφή των ρομπότ, της ιστορίας και των δυνατοτήτων τους, έπειτα ακολουθεί η περιγραφή του ρομπότ της Lego και τέλος τρόποι συγγραφής κώδικα και το πρόγραμμά μας.

Αν και η ιστορία των ρομπότ ξεκινάει από πολύ παλιά, χρειάστηκαν πολλά χρόνια μέχρι να καθιερωθεί ο όρος ρομπότ και να καθοριστεί τι ακριβώς είναι το ρομπότ. Όσο περνούν τα χρόνια, τόσο περισσότερο τα ρομπότ εντάσσονται στην καθημερινότητά μας.

Το ρομπότ της Lego είναι ένα διασκεδαστικό και χρήσιμο εργαλείο χωρίς συγκεκριμένη μορφή. Αποτελείται από πολλά πλαστικά κομμάτια τα οποία μπορούμε να συνδυάσουμε με όποιο τρόπο εμείς θέλουμε. Με τη χρήση τους δημιουργήσαμε ένα ρομπότ με τη μορφή σκύλου. Για να μπορούμε να κατασκευάσουμε ξανά το ίδιο ρομπότ σε περίπτωση που χρειαστεί, χρησιμοποιήσαμε το πρόγραμμα Lego Digital Designer που μας επιτρέπει να δημιουργήσουμε ένα τρισδιάστατο εικονικό ρομπότ, ίδιο με το πραγματικό. Επιπλέον, αποθηκεύει οδηγίες βήμα-βήμα για την επανακατασκευή του.

Το βασικό σενάριο του προγράμματος μας είναι το ρομπότ σκύλος να μπορεί να περιφέρεται αυτόνομα στο χώρο αποφεύγοντας τα εμπόδια που υπάρχουν σε αυτόν. Επιπλέον, κάθε φορά που ένα εμπόδιο εντοπίζεται, το ρομπότ ελέγχει αν υπάρχει πάνω του βέλος και στρίβει προς τη φορά του βέλους. Σε περίπτωση που κρίνει ότι δεν υπάρχει στρίβει προς τη μεριά που μπορεί να κινηθεί περισσότερο χωρίς να συναντήσει ξανά εμπόδιο. Τέλος, το πρόγραμμά τελειώνει όταν ο σκύλος μας εντοπίσει εμπόδιο με κόκκινο χρώμα.

Για τη συγγραφή του προγράμματος αυτού, χρησιμοποιήθηκε η Java η οποία είναι γλώσσα προγραμματισμού υψηλού επιπέδου. Για να μπορέσει να γίνει αυτό, χρειάστηκε να εγκαταστήσουμε το λογισμικό LeJOS στο NXT τούβλο που αποτελεί τον εγκέφαλο του ρομπότ μας. Το LeJOS εκτός από τη δυνατότητα να γράφουμε κώδικα για το NXT τούβλο σε Java, μας παρέχει και πολλά χρήσιμα εργαλεία που κάνουν τον προγραμματισμό ακόμη πιο εύκολο. Πολλά από αυτά περιγράφονται στην πτυχιακή αυτή.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Lego Mindstorms, NXT τούβλο, LeJOS, Labview, NXT-2.0, LeJOS Java, αισθητήρας, κινητήρας, NXT firmwares, γλώσσες προγραμματισμού NXT, εντοπισμός εμποδίων, αναγνώριση χρωμάτων, μέθοδοι LeJOS, ρομπότ-σκύλος, ιστορία ρομπότ, είδη ρομπότ.

ABSTRACT

This thesis deals with the creation of a robotic system using the Lego's educational robot, Lego Mindstorms NXT 2.0 and with its programming to implement a particular program. It begins with a general description of robots, robot's history and robot's potentials, then it continues with the description of Lego's robot and ends with programming methods and our own program.

Although, the history of the robots starts long ago, it took us many years to introduce the term robot and to determine what a robot is. As years passes, more and more robots become parts of our daily lives.

Lego's robot is an entertaining and useful tool with no specific form. It consists of several plastic pieces that can be combined in any way we want. We used them to create a robot that looks like a dog. In order to be able to rebuild the same model again if necessary, we used the program Digital Lego Designer, which allows us to create a three-dimensional visual robot, identical to the real one. In addition, it stores step by step instructions for its reconstruction.

The main concept of our program is the dog-robot to be able to move around autonomously avoiding obstacles. Moreover, each time an obstacle is detected, the robot checks if there is an arrow on it and turns in the direction that the arrow shows. In case that the robot decides that there is no arrow, it turns to the side where it can move for more distance finding no other obstacle. Finally, the program ends when our dog detects a red colored obstacle.

To write this program, we used Java, which is a high level programming language. In order to do this, we installed the LeJOS software on the NXT brick, which constitutes the brain of our robot. LeJOS software except of the ability to programming our NXT brick using Java, it also provides us with many useful tools that make development even easier. Many of these tools are described in this thesis.

KEYWORDS

Lego Mindstorms, NXT brick, LeJOS, Labview, NXT-2.0, LeJOS Java, sensor, motor, NXT firmwares, programming languages NXT, obstacle tracker, colour identification, LeJOS methods, dog-robot, robot's history, robot types.

ΕΥΧΑΡΙΣΤΙΕΣ

Με την ολοκλήρωση της πτυχιακής μου εργασίας θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή κ. Κωνσταντίνο Διαμαντάρα για την εμπιστοσύνη και την υπομονή του, καθώς και για τον εξοπλισμό που μου παρείχε. Επίσης, θέλω να ευχαριστήσω την οικογένεια μου για την υποστήριξή και την συμπαράστασή της όλο αυτό τον καιρό. Τέλος, θέλω να ευχαριστήσω την φίλη μου Κοκκοτού Βασιλική και τον συμφοιτητή μου Ορέστη Δανιηλίδη για την κατανόηση και την στήριξή τους.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο σχημάτων

Ευρετήριο πινάκων

ΕΙΣΑΓΩΓΗ

Η πτυχιακή αυτή βασίζεται στο ρομπότ της Lego το οποίο θα έχει την ικανότητα να περιφέρεται αυτόνομα στο χώρο αποφεύγοντας εμπόδια.

Το πρώτο κεφάλαιο, μας ταξιδεύει στον μαγικό κόσμο της ρομποτικής. Η περιήγησή μας ξεκινάει με την ιστορία των ρομπότ που αρχίζει από τα αρχαία ακόμη χρόνια. Ακολουθεί ο ορισμός των ρομπότ ο οποίος είναι μεταγενέστερος, τα πλεονεκτήματά τους, τα μέρη από τα οποία αποτελούνται, καθώς και τα διάφορα είδη που υπάρχουν.

Στο δεύτερο κεφάλαιο, συνεχίζουμε το ταξίδι μας περιγράφοντας το ρομπότ της εταιρίας Lego. Ξεκινάμε με την ιστορία του και συνεχίζουμε με την περιγραφή του NXT τούβλου, των αισθητήρων και των κινητήρων του. Τέλος, αναφερόμαστε σε firmwares και διάφορες γλώσσες προγραμματισμού που μπορούν να χρησιμοποιηθούν για να προγραμματίσουμε το NXT ρομπότ μας.

Το τρίτο κεφάλαιο αναφέρεται στο firmware LeJOS το οποίο χρησιμοποιήσαμε για να μπορούμε να προγραμματίσουμε το ρομπότ μας σε Java. Σε αυτό περιγράφονται οι δυνατότητες και τα πλεονεκτήματά του, διάφορές μέθοδοι και βοηθητικά προγράμματα που το συνοδεύουν και πολλές χρήσιμες πληροφορίες σχετικά με τον προγραμματισμό ενός NXT ρομπότ.

Το τέταρτο και τελευταίο κεφάλαιο, αποτελείται από το πρόγραμμά μας αναλυτικά, των κώδικα που γράψαμε και επεξηγήσεις.

Επιπλέον, στην πτυχιακή μας υπάρχουν τέσσερα βοηθητικά παραρτήματα. Το πρώτο μας περιγράφει το περιβάλλον σχεδίασης NXT-2.0 που συνοδεύει το ρομπότ της Lego με την αγορά του. Το δεύτερο, μας παρέχει οδηγίες εγκατάστασης του λογισμικού LeJOS στο NXT τούβλο του Eclipse και του LeJOS στο Eclipse. Το τρίτο, μας δίνει πληροφορίες σχετικά με τα εργαλεία που μας παρέχονται με το LeJOS και τη χρήση τους. Και τέλος, το τέταρτο περιέχει ολοκληρωμένο τον κώδικά μας με σχόλια.

ΚΕΦΑΛΑΙΟ 1

ΤΑ ΡΟΜΠΟΤ

ΕΙΣΑΓΩΓΗ

Στην εποχή μας πολλές εργασίες γίνονται με τη χρήση ρομπότ τα οποία έχουν πολλές ικανότητες. Εργασίες που φάνταζαν πολύ δύσκολες, ή ακόμη και ακατόρθωτες, τώρα μπορούν να πραγματοποιηθούν μέσα σε ελάχιστο χρόνο. Η αυτοματοποίηση είναι μέρος της καθημερινής μας ζωής.

Η ρομποτική είναι ο επιστημονικός κλάδος που ασχολείται με τη σύλληψη, το σχεδιασμό, την κατασκευή και τη λειτουργία των ρομπότ. Τα ρομπότ, ουσιαστικά, είναι μηχανήματα των οποίων η χρήση έχει ως στόχο την αντικατάσταση του ανθρώπου σε διάφορες εργασίες, τόσο σε φυσικό όσο και σε επίπεδο λήψης αποφάσεων.

1.1 ΙΣΤΟΡΙΑ ΤΩΝ ΡΟΜΠΟΤ

Ο άνθρωπος δημιούργησε τα ρομπότ στην προσπάθειά του για δημιουργία όλο και τελειότερων μέσων παραγωγής με περισσότερες δυνατότητες. Η εξέλιξη τους στο χρόνο, μας επιτρέπει να τα χωρίσουμε σε τρεις γενιές, ενδεικτικές του επιπέδου της τεχνολογίας:

- Πρώτη γενιά - Δεν είχαν δυνατότητα υπολογισμού και αίσθησης.
- Δεύτερη γενιά - Περιορισμένη υπολογιστική ισχύ, αισθητήρες ανατροφοδότησης και γλώσσες προγραμματισμού υψηλού επιπέδου.
- Τρίτη γενιά - Διαθέτουν νοημοσύνη, δηλαδή είναι ικανά να παίρνουν αποφάσεις και να αντιμετωπίζουν προβλήματα κατά τη διάρκεια εργασίας τους. Τις δυνατότητες αυτές τις αποκτούν μέσω εφαρμογών της τεχνητής νοημοσύνης σε συνδυασμό με τους εξελιγμένους αισθητήρες που χρησιμοποιούν.

Το πρώτο αυτόνομο σύστημα στην ανθρώπινη ιστορία εμφανίζεται στην ελληνική μυθολογία. Είναι ο Τάλως, ο χάλκινος γίγαντας με ανθρώπινη μορφή που κατασκευάστηκε από τον θεό Ήφαιστο με σκοπό να προστατεύει την Κρήτη από τους εισβολείς. Ο Ήφαιστος αφού τον κατασκεύασε του έδωσε ζωή γεμίζοντας την μοναδική φλέβα που είχε και ξεκινούσε από τον αυχένα ως τους αστραγάλους του με “ιχώρ” και έκλεισε την κατασκευή του με ένα καρφί. Λέγεται ότι ο Τάλως έκανε το γύρο του νησιού τρεις φορές κάθε μέρα και σε περίπτωση που εχθρικά πλοία πλησίαζαν το νησί τα βύθιζε πετώντας τεράστιους βράχους. Το τέλος του γίγαντα ήρθε όταν η μάγισσα Μήδεια του έκανε μάγια και τάζοντας του την αθανασία τον έπεισε να βγάλει μόνος του τη βίδα που συγκρατούσε μέσα του το “ιχώρ”. Το υγρό χύθηκε και το σώμα του, χωρίς ζωή, έπεσε και βυθίστηκε στη θάλασσα. (1)

Εικόνα 1 "Ο χάλκινος γίγαντας Τάλως"

Ο όρος ρομπότ όμως είναι μεταγενέστερος. Προέρχεται από την Τσεχοσλοβάκικη γλώσσα και πιο συγκεκριμένα από τη λέξη robota, η οποία σημαίνει καταναγκαστική εργασία. Η παρούσα έννοια του όρου δόθηκε από τον Τσεχοσλοβάκο συγγραφέα Karel Capek το 1921 στο θεατρικό έργο Rossum's Universal Robots, όπου στην υπόθεση του έργου εμφανίζεται ένα μηχανικό κατασκεύασμα, το ρομπότ. Το έργο σατιρίζει την εξάρτηση της κοινωνίας από τους μηχανικούς εργάτες της τεχνολογικής εξέλιξης (τα ρομπότ) που τελικά εξοντώνουν τους δημιουργούς τους. Σε πολλές σύγχρονες σλαβικές γλώσσες (π.χ. την πολωνική) χρησιμοποιείται σαν έκφραση της καθημερινότητας με την έννοια της σκληρής δουλειάς.

Μερικά χρόνια αργότερα, τη δεκαετία του 1940 ο Isaac Asimov, συγγραφέας βιβλίων επιστημονικής φαντασίας, περιγράφει το ρομπότ ως ένα ανθρωπόμορφο αυτόματο σύστημα στο οποίο ο άνθρωπος υπαγορεύει πως θα συμπεριφερθεί/αντιδράσει υπό ορισμένες συνθήκες. Το ρομπότ έχει μυαλό και μνήμη για να υπακούει στις εντολές που του δίνει ο άνθρωπος και δεν έχει καθόλου συναισθήματα. Δύο χρόνια μετά, το 1942, στο διήγημα του "Runaround" διατυπώνει για πρώτη φορά τους νόμους των ρομπότ που είναι οι εξής:

- Το ρομπότ δε θα κάνει κακό σε άνθρωπο, ούτε με την αδράνειά του θα επιτρέψει να βλαφτεί ανθρώπινο όν.

- Το ρομπότ πρέπει να υπακούει τις διαταγές που του δίνουν οι άνθρωποι, εκτός αν αυτές οι διαταγές έρχονται σε αντίθεση με τον πρώτο νόμο.

- Το ρομπότ οφείλει να προστατεύει την ύπαρξή του, εφόσον αυτό δεν συγκρούεται με τον πρώτο και τον δεύτερο νόμο. (2)

Ο όρος βιομηχανικό ρομπότ καθιερώθηκε το 1954 από τον G.C Devol ο οποίος περιέγραψε πώς μπορεί να κατασκευαστεί ένα ελεγχόμενο μηχανικό χέρι, το οποίο μπορεί να εκτελεί διάφορες εργασίες στη βιομηχανία. Το πρώτο βιομηχανικό ρομπότ κατασκευάσθηκε και τέθηκε σε λειτουργία το 1961 από την εταιρεία Unimation.

1.2 ΟΡΙΣΜΟΣ

Υπάρχουν πολλοί διαφορετικοί ορισμοί για τον όρο ρομπότ. Γενικά, ένα ρομπότ αποτελείται από διάφορα μέρη τα οποία μπορεί να κινεί για να εκτελεί συγκεκριμένες εργασίες. Επιπλέον, μπορεί να έχει έναν ή περισσότερους αισθητήρες για να αντιλαμβάνεται το περιβάλλον και μπορεί να δράσει κάτω από τον απευθείας έλεγχο ενός ανθρώπου ή αυτόνομα κάτω από τον έλεγχο ενός προ-προγραμματισμένου υπολογιστή. Κάθε ρομπότ αποτελείται από τρεις συνιστώσες:

- Μηχανολογικό υποσύστημα – τα μέρη από τα οποία αποτελείται

Αυτό δίνει την δυνατότητα στο ρομπότ να εκτελεί κάποιο έργο. Αποτελείται από διάφορους μηχανισμούς που επιτρέπουν στο ρομπότ να κινείται, όπως κινητήρες, αρθρώσεις κ.α.

- Αισθητήρες – ανιχνεύουν την κατάσταση του περιβάλλοντος

Αποτελούν το μοναδικό τρόπο αντίληψης του περιβάλλοντος που έχουν τα ρομπότ. Μέσω των αισθητήρων αντλούν πληροφορίες σχετικά με την κατάσταση του περιβάλλοντος χώρου καθώς και τη δική τους κατάσταση.

Σύστημα ελέγχου – το οποίο ελέγχει το μηχανολογικό σύστημα βάσει της κατάστασης του περιβάλλοντος όπως ανιχνεύτηκε από τους αισθητήρες.

Συνδυάζει τα δεδομένα που λαμβάνει μέσω των αισθητήρων του με την αντίστοιχη δράση που πρέπει να πραγματοποιήσει σύμφωνα με τις εντολές που του δόθηκαν. Πιο συγκεκριμένα ο ελεγκτής είναι η μονάδα που μας δίνει την δυνατότητα να προγραμματίζουμε το ρομπότ και αποτελείται από το Hardware και το Software.

- Hardware: Συνήθως ένας υπολογιστής, όπου αποθηκεύεται το πρόγραμμα που θα εκτελεστεί.

- Software: Το λογισμικό για τη δημιουργία των κατάλληλων σημάτων ελέγχου, σύμφωνα με κάποιον αλγόριθμο, παίρνοντας υπόψη διάφορες μεταβλητές που λαμβάνει μέσω των αισθητήρων του.

Ο Διεθνής Οργανισμός Τυποποίησης (ISO) ορίζει το ρομπότ ως έναν επαναπρογραμματιζόμενος μηχανικός βραχίονας πολλαπλών λειτουργιών που έχει μερικούς άξονες κίνησης, ικανός να κινεί υλικά, κομμάτια, εργαλεία ή ειδικές συσκευές μέσω μεταβλητών προγραμματισμένων λειτουργιών για την εκτέλεση μιας ποικιλίας εργασιών.

Σύμφωνα με το Robot Institute of America, ρομπότ είναι ένα μηχάνημα σχεδιασμένο ώστε μέσω προγραμματισμένων κινήσεων να μεταφέρει υλικά, τεμάχια, εργαλεία ή εξειδικευμένες συσκευές με σκοπό την διεξαγωγή διάφορων εργασιών. Το μηχάνημα αυτό είναι γενικά σχεδιασμένο για την εκτέλεση επαναληπτικών εργασιών και μπορεί να προσαρμοστεί στην εκτέλεση νέων.

1.3 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΧΡΗΣΗΣ ΡΟΜΠΟΤ

Μέχρι σήμερα έχουν τεθεί σε λειτουργία εκατομμύρια ρομπότ στην Αμερική, την Ιαπωνία και την Ευρώπη. Τα πλεονεκτήματα της χρήσης των ρομπότ ιδιαίτερα στη βιομηχανία είναι πολλά και σε πολλές περιπτώσεις καθιστούν τη χρήση τους απαραίτητη. Τα βασικότερα πλεονεκτήματα είναι:

- Μειωμένο κόστος εργατικών εξόδων
- Αυξημένη παραγωγικότητα
- Αυξημένη ακρίβεια
- Πραγματοποίηση κουραστικών, επαναλαμβανόμενων ή/και επικινδύνων εργασιών
- Εργασία σε αντίξοες συνθήκες όπως ραδιενεργό περιβάλλον, υπερβολικά υψηλές ή χαμηλές θερμοκρασίες και επισκευή δορυφόρων.

1.4 ΕΙΔΗ ΡΟΜΠΟΤ

Τα ρομπότ μπορούν να κατηγοριοποιηθούν με πολλούς διαφορετικούς τρόπους. Οι πιο συνηθισμένοι τρόποι κατηγοριοποίησης είναι ανάλογα με:

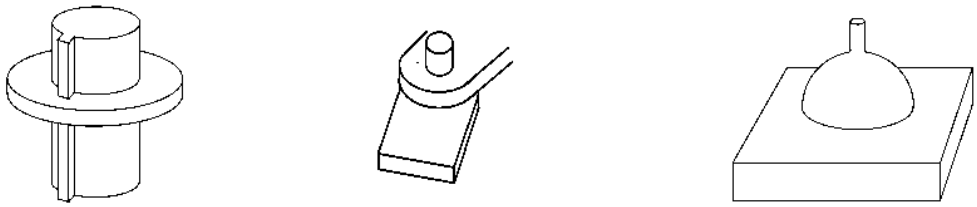
- τη χρήση τους (βιομηχανικά, μη βιομηχανικά)
- το μέγεθός τους (μικρά, μεγάλα)
- το είδος παροχής ισχύος (ηλεκτρικά, πνευματικά, υδραυλικά)
- τη γενιά σχεδιασμού τους (πρώτη, δεύτερη ή τρίτη γενιά)
- τη δυνατότητα κίνησης (σταθερής βάσης, κινούμενα)
- το περιβάλλον εργασίας (επίγεια, εναέρια, υποβρύχια, διαστημικά)
- τη μορφή και τον εξωτερικό σχεδιασμό (ρομποτικοί βραχίονες, ανθρωποειδή κ.λπ.)
- τον αριθμό αρθρώσεων και το είδος των ελεγκτών των κινητήρων τους (στους ρομποτικούς βραχίονες)

Παρακάτω θα δούμε αναλυτικά δύο από τις πιο ενδιαφέρουσες κατηγορίες. Η πρώτη χωρίζει τα ρομπότ ανάλογα με τη δυνατότητα κίνησης που έχουν και η δεύτερη ανάλογα με τον τομέα στον οποίο χρησιμοποιούνται και την εφαρμογή τους.

1.4.1 ΑΝΑΛΟΓΑ ΜΕ ΤΗ ΔΥΝΑΤΟΤΗΤΑ ΚΙΝΗΣΗΣ

Σε σχέση με τη δυνατότητα κίνησης υπάρχουν δύο βασικά είδη ρομπότ, τα σταθερής βάσης και τα κινούμενα.

Τα σταθερής βάσης αποτελούνται από διαδοχικά τμήματα που συνδέονται μέσω αρθρώσεων σχηματίζοντας μία κινηματική αλυσίδα. Η αλυσίδα αυτή έχει το ένα της άκρο συνδεδεμένο σταθερά σε κάποιο σημείο του περιβάλλοντος χώρου. Οι αρθρώσεις μπορεί να είναι: πρισματικές, περιστροφικές ή σφαιρικές.



Εικόνα 2 "Αρθρώσεις - Αριστερά απεικονίζεται μία πρισματική άρθρωση, στη μέση μία περιστροφική άρθρωση και στα δεξιά μία σφαιρική άρθρωση"

Ένα από τα πιο γνωστά ρομπότ του είδους αυτού είναι το PUMA (Programmable Universal Machine for Assembly or Programmable Universal Manipulation Arm). Είναι ένας βιομηχανικός ρομποτικός βραχίονας που αναπτύχθηκε από τον Victor στην εταιρία Unimation, πρωτοπόρο στα ρομπότ. Διαθέτει 6 άξονες κίνησης που του δίνουν τη δυνατότητα να τοποθετήσει το άκρο του σε οποιαδήποτε θέση και προσανατολισμό μέσα στο χώρο εργασίας του.

Εικόνα 3 "Ρομποτικός βραχίονας PUMA - Ένα από τα πιο χαρακτηριστικά παραδείγματα του είδους του. Ο τρόπος με τον οποίο είναι σχεδιασμένο του επιτρέπει να κινηθεί προς όλες τις κατευθύνσεις και είναι σε θέση να φτάσει και να μετακινήσει όλα τα αντικείμενα που φαίνονται στην εικόνα"

Ως κινητά ρομπότ χαρακτηρίζονται όλα εκείνα τα ρομπότ που έχουν τη δυνατότητα να μετακινούνται στο χώρο. Την δυνατότητα αυτή την αποκτούν μέσω ειδικών συστημάτων προώθησης όπως τροχούς, προπέλες, μηχανικά πόδια και άλλα. Τα κινούμενα ρομπότ μπορούμε να τα χωρίσουμε σε επιμέρους κατηγορίες ανάλογα με το βαθμό αυτονομίας που έχουν.

Οι υποκατηγορίες είναι οι εξής:

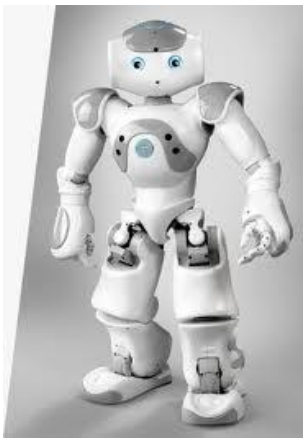
- AGVs – Automatic Guided Vehicles ή Αυτόματα κατευθυνόμενα οχήματα στα ελληνικά. Έχουν περιορισμένη αυτονομία κίνησης και η διαδρομή τους είναι προκαθορισμένη. Συνήθως καθορίζεται μέσω πομπών στον περιβάλλοντα χώρο, καλωδίων, ραγών ή χρωματιστών γραμμών στο έδαφος.

Εικόνα 4 "Αυτόματα κατευθυνόμενο όχημα - Το εικονιζόμενο όχημα κινείται διαβάζοντας τις κίτρινες γραμμές που υπάρχουν στο πάτωμα "

- Αυτόνομα έντροχα ρομπότ ή πιο απλά αυτόνομα ρομπότ με ρόδες. Έχουν αρκετά μεγάλη αυτονομία. Μπορούν να κινούνται στο χώρο ακολουθώντας εντολές υψηλού επιπέδου.

Εικόνα 5 "Έντροχο ρομπότ - Το συγκεκριμένο έχει τρεις τροχούς, ένα πίσω και δύο μπροστά και μπορεί να κινείται σε ευθεία και να στρίβει"

- Ρομπότ που βαδίζουν. Διαθέτουν μηχανικά πόδια και το πλεονέκτημά τους σε σχέση με τα έντροχα είναι ότι μπορούν να αναρριχηθούν σε ανώμαλες επιφάνειες και μη επίπεδα εδάφη. Πιο συνηθισμένα είναι τα δίποδα και τετράποδα αλλά υπάρχουν και με περισσότερα πόδια.



Εικόνα 6 "Ρομπότ με πόδια - Μπορεί να κινείται με τον τρόπο που κινείται και ένας άνθρωπος"

- Εναέρια ρομπότ. Πρόκειται για μη επανδρωμένα ιπτάμενα ρομπότ τα οποία συχνά χρησιμοποιούνται για στρατιωτικούς σκοπούς.

Εικόνα 7 "Ιπτάμενο ρομπότ"

- AUVs - Autonomous Underwater Vehicles ή στα ελληνικά Αυτόνομο υποβρύχιο όχημα. Για τις ανάγκες τροφοδοσίας τους χρησιμοποιούνται ειδικές μπαταρίες, πράγμα που περιορίζει την αυτονομία τους. Διαθέτουν όμως τορπίλες και μπορούν να κινούνται με αρκετά μεγάλες ταχύτητες.

Εικόνα 8 "Υποβρύχιο ρομπότ"

1.4.2 ΑΝΑΛΟΓΑ ΜΕ ΤΗΝ ΕΦΑΡΜΟΓΗ

Ένας άλλος αξιοσημείωτος τρόπος κατηγοριοποίησης των ρομπότ είναι ανάλογα με τον τρόπο χρήσης τους. Έτσι έχουμε:

- Βιομηχανικά ρομπότ

Σύμφωνα με τον ISO, ως βιομηχανικό ρομπότ ορίζεται ένα αυτόματα ελεγχόμενο σύστημα με πολλούς βραχίονες και τρείς ή περισσότερους άξονες. Συνήθως χρησιμοποιούνται για τυπικές επαναλαμβανόμενες εργασίες που απαιτούν ακρίβεια και ταχύτητα αλλά και για εργασίες σε αντίξοες συνθήκες (πολύ υψηλές ή χαμηλές θερμοκρασίες, μολυσμένο αέρα κ.α.). Τέτοιες εργασίες είναι: η συσκευασία, η παλετοποίηση, η επιλογή, η τοποθέτηση, η συναρμολόγηση, η συγκόλληση, ο σχεδιασμός, ο έλεγχος αλλά και συνδυασμοί όλων των προηγουμένων. Οι δράσεις που θα πραγματοποιήσει ένα ρομπότ κατά την λειτουργία του καθορίζονται από προγραμματιζόμενες ρουτίνες που ορίζουν τη σειρά των συντονισμένων επαναλαμβανόμενων κινήσεων καθώς και την ταχύτητα, την κατεύθυνση, την επιτάχυνση ή επιβράδυνση, και την απόσταση. Γνωστά ρομπότ της κατηγορίας αυτής είναι τα PUMA, τα SCARA και τα Cartesian Coordinate.

- Ιατρικά ρομπότ

Τα ιατρικά ρομπότ ίσως αποτελούν την πιο σπουδαία εφαρμογή των ρομπότ. Στις μέρες μας είναι ευρέως διαδεδομένη η τεχνική των ζευγαριών γιατρός και ρομπότ. Για παράδειγμα ένας χειρούργος και κάποιο είδος μηχανισμού (ρομπότ) επιτρέπουν την πραγματοποίηση χειρουργικών επεμβάσεων με πολύ μικρές τομές, μειώνοντας έτσι αισθητά τον κίνδυνο για τους ασθενείς. Η ικανότητα του χειρούργου να ελέγχει το ρομπότ ενισχύεται με την παροχή ανάδρασης που προσφέρεται, δίνοντας του τη δυνατότητα να έχει την αίσθηση της αφής. Αυτού του είδους τα ρομπότ που χρησιμοποιούνται σε επεμβάσεις αλλά δεν είναι ανεξάρτητα ονομάζονται teleoperated συσκευές. Άλλο σημαντικό επίτευγμα της ιατρικής είναι η δημιουργία τεχνητών μηχανικών ακρών, όπως χεριών και ποδιών, που μπορούν να αλληλεπιδράσουν με το ανθρώπινο βιολογικό σώμα. Το να μπορεί κάποιος άνθρωπος με τεχνητά πόδια να περπατήσει ή να αγγίζει και να σηκώνει αντικείμενα με τεχνητά χέρια είναι ένα θαύμα από μόνο του. Επιπλέον, οι ιατρικές κοινότητες ερευνούν τη δυνατότητα αντικατάστασης και άλλων μερών του σώματος με τεχνητά εμφυτεύματα όπως μάτια, αυτιά αλλά και καρδιά. Ειδικά όσο αφορά την καρδιά είναι χιλιάδες τα άτομα που πεθαίνουν περιμένοντας για το σωτήριο εμφύτευμα, οπότε η δημιουργία τεχνητού εμφυτεύματος καρδιάς, αν και βρίσκεται ακόμη σε πειραματικό στάδιο, έχει δώσει ελπίδες σε πολλούς ανθρώπους. Τέλος, άλλη μία πολύ σημαντική χρήση των ρομπότ στην ιατρική είναι η παροχή βοήθειας σε άτομα με σοβαρούς κινητικούς περιορισμούς παρέχοντας τους τη δυνατότητα για έστω και κάποιου είδους κίνησης, αλλά και άμεση ειδοποίηση ιατρικής βοήθειας σε περίπτωση που κάτι δεν πάει καλά.

- Στρατιωτικά ρομπότ

Στην κατηγορία αυτή συμπεριλαμβάνονται και αυτόνομα και τηλεχειριζόμενα ρομπότ σχεδιασμένα για στρατιωτικές εφαρμογές. Ένα χαρακτηριστικό παράδειγμα είναι ένα τετράτροχο ρομπότ εξοπλισμένο με πολλές κάμερες, ραντάρ και συχνά ένα πυροβόλο όπλο το οποίο εκτελεί αυτόματα τυχαίες ή προγραμματισμένες περιπολίες σε μία συγκεκριμένη περιοχή, για παράδειγμα σε μία στρατιωτική βάση. Όταν το ρομπότ ανιχνεύσει κίνηση σε περιοχή όπου απαγορεύεται ειδοποιεί τον χειριστή. Αυτός, αφού ελέγξει από τις κάμερες τι ήταν αυτό που κινούνταν, μπορεί είτε να αναθέσει στο ρομπότ να αγνοήσει το γεγονός και να συνεχίσει την περιπολία του ή να αναλάβει ο ίδιος το χειριστήριο για να έχει τον έλεγχο της περιπολίας ή να ασχοληθεί προσωπικά με τον εισβολέα.

- Εξερευνητικά ρομπότ

Ο άνθρωπος πάντα ήθελε να ανακαλύπτει και να εξερευνεί νέα μέρη. Πολλές φορές όμως τα μέρη αυτά κρύβουν μεγάλους κινδύνους, όπως ο ωκεανός και το διάστημα. Για να μπορεί να έχει πρόσβαση σε τέτοια μέρη χωρίς να κινδυνεύει δημιούργησε ρομπότ τα οποία μπορούν να φτάσουν σε αυτές τις τοποθεσίες και να συλλέξουν πληροφορίες για τη σύσταση του περιβάλλοντος και τη θερμοκρασία, να συλλέξουν δείγματα και να βγάλουν φωτογραφίες. Ένα παράδειγμα υποβρύχιου ρομπότ είναι το «Οδύσσεια ΙΙ», το οποίο αναπτύχθηκε από τους επιστήμονες του MIT για την εξερεύνηση των ωκεανών, ενώ ένα παράδειγμα ρομπότ που ταξίδεψε στο διάστημα είναι το "Sojourner" που προσγειώθηκε στην επιφάνεια του Άρη στις 4 Ιουλίου 1998.

- Οικιακά ρομπότ

Οικιακά ρομπότ λέγονται αυτά που χρησιμοποιούνται για τις δουλειές του σπιτιού. Μέχρι στιγμής κυκλοφορούν στην αγορά μόνο μερικά μοντέλα, αλλά είναι πολλοί αυτοί που πιστεύουν ότι σε λίγα χρόνια τα οικιακά ρομπότ θα είναι πολύ συνηθισμένα. Χαρακτηριστικά παραδείγματα είναι η αυτόματη ηλεκτρική σκούπα, η οποία σκουπίζει μόνη της όλο το σπίτι και επιστρέφει στη βάση της για να φορτιστεί όταν χρειάζεται, με ποιο γνωστές τις Electrolux trilobite, Roomba και Neato αλλά και ένα ανθρωποειδές ρομπότ που μαζεύει τα πιάτα από το τραπέζι και τα τοποθετεί στο πλυντήριο πιάτων που αναπτύχθηκε από την Sharp το 2006.

- Ψυχαγωγικά ρομπότ

Τα ψυχαγωγικά ρομπότ, όπως υποδηλώνει και το όνομά τους, προορίζονται για την ψυχαγωγία μας. Τα ρομπότ αυτά χρησιμοποιούν μοτέρ, αερομηχανική και υδραυλικά συστήματα για τη δημιουργία κίνησης. Ένα πολύ καλό παράδειγμα των ρομπότ αυτών είναι οι κούνιες που υπάρχουν στη Disneyland. Επίσης μπορεί να σχετίζονται με την εικονική πραγματικότητα και διαδραστικά παιχνίδια.

1.5 ΑΙΣΘΗΤΗΡΕΣ

Ένα από τα πιο σημαντικά και χρήσιμα εφόδια ενός ρομπότ είναι οι αισθητήρες του. Αποτελούν βασικά, το μοναδικό τρόπο αντίληψης του περιβάλλοντος. Υπάρχουν πολλών ειδών αισθητήρες που χωρίζονται σε τρεις κύριες κατηγορίες. Τους αισθητήρες περιβάλλοντος, που στέλνουν στο ρομπότ σήματα περιγράφοντας τι συμβαίνει γύρω του, τους αισθητήρες ανάδρασης που καθορίζουν πώς θα αντιδράσει κάτω από συγκριμένες συνθήκες και τους αισθητήρες επικοινωνίας που επιτρέπουν σε έναν εξωτερικό παράγοντα να παρέχει πληροφορίες στο ρομπότ για τον τρόπο με τον οποίο θα αντιδράσει. Οι αισθητήρες όμως δεν είναι τέλειοι. Όταν χρησιμοποιούμε έναν αισθητήρα σε ένα ρομπότ θα υπάρξουν πολλές περιπτώσεις που δεν θα έχουμε το αναμενόμενο αποτέλεσμα. Για παράδειγμα το γεγονός ότι μπορούμε να δούμε εμείς ένα αντικείμενο, δεν σημαίνει απαραίτητα ότι θα το ανιχνεύσει και το ρομπότ μας. Για να έχουμε σωστά αποτελέσματα πρέπει να γνωρίζουμε πως ακριβώς λειτουργούν και τι μετράνε οι αισθητήρες που χρησιμοποιούμε. Μερικοί από τους πιο σημαντικούς αισθητήρες είναι:

- Αισθητήρες ανίχνευσης εμποδίων – απόστασης

Αισθητήρες υπερήχων: Μπορούν να υπολογίσουν την απόσταση μεταξύ ενός εμποδίου και του ρομπότ. Η εμβέλειά τους κυμαίνεται από 0 έως 300 εκατοστά περίπου. Αν χρησιμοποιούνται περισσότεροι από ένας τέτοιους αισθητήρες στον ίδιο χώρο τα αποτελέσματά τους μπορεί να είναι αλλοιωμένα, αφού αλληλεπιδρούν μεταξύ τους.

Αισθητήρες υπερύθρων: Δεν έχουν τη δυνατότητα να υπολογίσουν την απόσταση από ένα εμπόδιο αλλά μπορούν να καταλάβουν αν υπάρχει εμπόδιο σε μία περιοχή περίπου 10 εκατοστών. Ένα πλεονέκτημα των αισθητήρων αυτών είναι ότι δεν επηρεάζονται από τις διαχεόμενες υπέρυθρες του περιβάλλοντος.

Διακόπτες - αισθητήρες επαφής: Ανιχνεύουν την ύπαρξη εμποδίου όταν συγκρουστούν με αυτό.

- Αισθητήρες κλίσης

Αυτοί είναι είτε ολοκληρωμένα κυκλώματα που μετράνε την κλίση, είτε διακόπτες υδραργύρου που μπορούν να αντιληφθούν αν έχει ξεπεραστεί κάποιο συγκεκριμένο επίπεδο κλίσης.

- Αισθητήρες θέσης

Σε αυτούς συμπεριλαμβάνονται τα γνωστά μας GPS, οι αισθητήρες πυξίδας, και τα γυροσκόπια που είναι αισθητήρες προσανατολισμού.

- Αισθητήρες κίνησης

Σε αυτή την κατηγορία ανήκουν τα οδόμετρα, οι αισθητήρες επιτάχυνσης και οι αισθητήρες μέτρησης γωνιακής ταχύτητας.

- Άλλοι αισθητήρες

Για παράδειγμα κάμερες, αισθητήρες ήχου, χρώματος, θερμοκρασίας, πίεσης, εντοπισμού αερίων, ραδιενέργειας, φωτός και πολλά άλλα.

1.6 ΚΙΝΗΤΗΡΕΣ

Οι κινητήρες είναι αυτοί που δίνουν τη δυνατότητα στα ρομπότ να κινούν διάφορα μέρη τους ή ακόμη και κινούνται τα ίδια στο χώρο. Τα είδη των κινητήρων είναι:

- Ηλεκτρικοί Σερβοκινητήρες (DC/AC motors, servos)

Για την κίνηση τους απαιτείται ηλεκτρική ενέργεια η οποία μπορεί να παρέχεται από το ηλεκτρικό ρεύμα, αν το ρομπότ μας μπορεί να βρίσκεται συνδεδεμένο με κάποια πρίζα, ή από κάποια άλλη πηγή όπως μπαταρίες.

- Βηματικοί κινητήρες (stepper motors)

Είναι σύγχρονοι ηλεκτρικοί κινητήρες που μπορούν να χωρίσουν τη μία περιστροφή σε μικρότερα βήματα δίνοντάς μας τη δυνατότητα για μετακινήσεις με μεγάλη ακρίβεια.

- Αεροσυμπιεστές (air muscles)

Λειτουργούν με τη βοήθεια πεπιεσμένου αέρα.

- Υδραυλικοί κινητήρες

Κινούνται με έμβολα λαδιού ή άλλων υγρών.

- Πιεζοηλεκτρικοί κινητήρες

Ονομάζονται και υπερηχητικοί κινητήρες. Βασίζονται στο αντίστροφο πιεζοηλεκτρικό φαινόμενο το οποίο προκαλεί περιστροφική κίνηση.

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο αυτό ασχοληθήκαμε με τα ρομπότ. Ξεκινήσαμε με μία σύντομη ιστορική αναδρομή και στη συνέχεια αναφερθήκαμε στην προέλευση της λέξης ρομπότ και στον ορισμό του. Έπειτα, είδαμε τα πλεονεκτήματα και τα είδη των ρομπότ και τέλος, αναφέραμε κάποια γενικά στοιχεία σχετικά με τους αισθητήρες και τους κινητήρες. Ήρθε λοιπόν η ώρα να μάθουμε μερικά πράγματα σχετικά με το ρομπότ που χρησιμοποιήσαμε, το ρομπότ της Lego.

ΚΕΦΑΛΑΙΟ 2

ΤΟ ΡΟΜΠΟΤ ΤΗΣ LEGO

ΕΙΣΑΓΩΓΗ

Η Lego είναι μία πολύ γνωστή εταιρεία που κατασκευάζει πλαστικά συναρμολογούμενα παιχνίδια που αποτελούνται από πλαστικά αλληλοσυνδεόμενα μέρη. Τα Lego Mindstorms είναι μια γραμμή παραγωγής της Lego που δίνει τη δυνατότητα σε ερασιτέχνες, λάτρεις της ρομποτικής, κάθε ηλικίας να κατασκευάσουν και να προγραμματίσουν αληθινά μίνι ρομπότ μέσα σε λίγη ώρα. Ο συνδυασμός του προγραμματιζόμενου τούβλου (ή και περισσότερων) με απλά τουβλάκια Lego, τεχνικά κομμάτια (όπως άξονες, γρανάζια κ.α.), κινητήρες και αισθητήρες είναι αυτό που τα κάνει πραγματικά ξεχωριστά. Πολλά διαφορετικά είδη, από παιχνίδια μέχρι και βιομηχανικά ρομπότ, μπορούν να δημιουργηθούν με τη χρήση των Lego Mindstorms. Επίσης, η ελευθερία που παρέχεται στη δημιουργία και τον προγραμματισμό είναι πολύ μεγάλη. Το μόνο ίσως εμπόδιο είναι η έλλειψη φαντασίας του χρήστη.

- Στην πρώτη γενιά, τα πλαστικά κομμάτια ήταν μόνο τουβλάκια που μπορούσαν να αλληλοσυνδεθούν το ένα πάνω από το άλλο και /ή πάνω σε μια πλατφόρμα, σχηματίζοντας στατικά κτήρια και οχήματα.

- Η δεύτερη γενιά συνοδεύεται από τη δημιουργία μιας νέας σειράς δομικών πλαστικών κομματιών που μπορούσαν να συνδυαστούν με περισσότερους τρόπους. Η νέα σειρά αυτή ονομάστηκε Lego Technic. Ο συνδυασμός αυτών των νέων δομικών υλικών επιτρέπει τη μετάδοση κίνησης με γρανάζια, άξονες και ρόδες που περιστρέφονται. Γενικά, αυτά τα νέα υλικά είναι πολύ περισσότερα και δίνουν μεγαλύτερη ευελιξία στο δημιουργό. Μάλιστα, είναι δυνατόν να κατασκευαστούν πιο προχωρημένες μηχανικές κατασκευές, που αποτελούνται από περίπλοκα κινούμενα μέρη. Επίσης, η δομή τους είναι τέτοια ώστε να υπάρχει μεγαλύτερη αναλογία με πραγματικές ανθρώπινες κατασκευές όπως γέφυρες, οχήματα, γερανούς κ.α.

- Στην επόμενη γενιά, την τρίτη, χαρακτηριστική ήταν η συμβολή των ερευνητών στο Εργαστηρίου Πολυμέσων MIT (Media Laboratory), οι οποίοι χρησιμοποιούσαν για πολλά χρόνια τη δεύτερη γενιά για τη δημιουργία μηχανικών προτύπων. Αυτοί συνειδητοποίησαν πως πολλές φορές το να προγραμματίσουν και να ελέγξουν τα πρότυπα που δημιουργούσαν θα ήταν πολύ χρήσιμο. Η Lego έδειξε ενδιαφέρον για αυτή την ιδέα τους και χρηματοδότησε τη προσπάθεια του MIT Αυτό ήταν που τελικά οδήγησε στην τρίτη γενιά της Lego, στην οποία εκτός των δομικών υλικών περιλαμβάνονται επίσης κινητήρες, αισθητήρες και ένας μικροελεγκτής, το προγραμματιζόμενο τούβλο, όπως είναι γνωστό.

2.1 ΙΣΤΟΡΙΑ ΤΗΣ LEGO

Η πρώτη λιανική έκδοση των Lego Mindstorms κυκλοφόρησε το 1998 και πωλήθηκε εμπορικά με την επωνυμία Robotics Invention System (RIS). Η τρέχουσα έκδοση κυκλοφόρησε στα τέλη του 2006 ως Lego Mindstorms NXT.

Συχνά, τα Mindstorms ρομπότ πωλούνται και χρησιμοποιούνται ως εκπαιδευτικά εργαλεία. Αυτό έγινε πρώτη φορά μέσω μιας συνεργασίας μεταξύ της Lego και του Εργαστηρίου Πολυμέσων MIT. Κάθε ρομπότ συνοδεύεται από το απλό στη χρήση αλλά με πολλές δυνατότητες γραφικό λογισμικό προγραμματισμού NXT-G, που αναπτύχθηκε στο Πανεπιστήμιο Tufts χρησιμοποιώντας ως μηχανή το LabVIEW της National Instruments. Το λογισμικό αυτό περιγράφεται αναλυτικά στο Παράρτημα Ι.

2.1.1 RIS – RCX ΤΟΥΒΛΟ

Η πρώτη γενιά Lego Mindstorms χτίστηκε γύρω από το κομμάτι της Lego με τη μορφή τούβλου γνωστό ως RCX (Robotic Control X). Το κομμάτι αυτό αποτελεί τον εγκέφαλο του ρομπότ. Περιέχει έναν μικροελεγκτή Renesas H8/300 ως εσωτερική κεντρική μονάδα επεξεργασίας του. Το τούβλο προγραμματίζεται με τη φόρτωση ενός προγράμματος από έναν ηλεκτρονικό υπολογιστή στη RAM του τούβλου μέσω μιας ειδικής υπέρυθρης διεπαφής (IR). Έπειτα, χρησιμοποιεί τους αισθητήρες που παρέχονται ως είσοδο από το περιβάλλον του, επεξεργάζεται τα στοιχεία που λαμβάνει και ανάλογα με τις οδηγίες που του δόθηκαν αντιδράει, για παράδειγμα δίνει κίνηση στους κινητήρες.

Εικόνα 9 "RCX ΤΟΥΒΛΟ"

Επιπλέον, δύο ή περισσότερα τούβλα RCX μπορούν να επικοινωνήσουν το ένα με το άλλο μέσω της διεπαφής IR, επιτρέποντας τη συνεργασία ή τον ανταγωνισμό μεταξύ τους. Εκτός από τη θύρα IR, υπάρχουν επιπλέον τρεις θύρες εισαγωγής αισθητήρων, τρεις θύρες σύνδεσης κινητήρων και μία LCD οθόνη που μπορεί να εμφανίζει το επίπεδο φόρτισης των μπαταριών, ποιο πρόγραμμα εκτελείται και άλλες πληροφορίες. Τα τούβλα RCX έκδοσης 1.0 διαθέτουν παροχή ρεύματος για να επιτρέπουν τη συνεχή λειτουργία, αντί της λειτουργίας περιορισμένου χρόνου με τη χρήση μπαταριών. Στην έκδοση RCX 2.0, η παροχή ρεύματος αφαιρέθηκε. Τα τούβλα RCX με παροχή ρεύματος είναι δημοφιλή για τα στατικά προγράμματα ρομποτικής, όπως τα ρομπότ βραχίονες.

2.1.2 NXT ΤΟΥΒΛΟ

Η νέα σειρά LEGO Mindstorm NXT κάνει την κατασκευή ρομπότ πιο γρήγορη και πιο εύκολη. Βασίζεται στο επιτυχημένο RCX τούβλο και ουσιαστικά μπορούμε να πούμε ότι αποτελεί την εξέλιξη του. Όπως και το RCX μπορεί να προγραμματιστεί μέσω ηλεκτρονικού υπολογιστή και έχει τρεις θύρες για την ένωση των κινητήρων (θύρες A, B και C), τέσσερις για την ένωση αισθητήρων (θύρες 1, 2, 3 και 4) και μία θύρα USB για τη σύνδεση με τον υπολογιστή, ώστε να μπορεί ο χρήστης να μεταφέρει και να τρέχει τα προγράμματά του. Επίσης, έχει κι αυτό μια LCD οθόνη όπου μπορούμε να δούμε πληροφορίες σχετικά με την κατάσταση του ρομπότ. Έχει όμως μεγαλύτερη σημασία να δούμε τι καινούριο υπάρχει. Το NXT τούβλο έχει βελτιωθεί με την προσθήκη νέων τεχνολογιών και αισθητήρων με περισσότερες ικανότητες. Οι κυριότερες διαφορές ανάμεσα στις δύο σειρές, όσο αφορά το προγραμματιζόμενο τούβλο, είναι ότι το NXT τούβλο είναι περίπου 15 φορές πιο γρήγορο, αφού ο μικροεπεξεργαστής του είναι 32 bit (σε αντίθεση με τα 16 bit του RCX), έχει μεγαλύτερη μνήμη, και υποστηρίζει τη χρήση Bluetooth.

Εικόνα 10 "NXT ΤΟΥΒΛΟ"

Εικόνα 11 "Το κουτί του ρομπότ της Lego - Το εικονιζόμενο κουτί ανήκει στο ρομπότ Mindstorms NXT έκδοση 2.0"

Η έκδοση που χρησιμοποιήσαμε είναι η 2.0 και αποτελείται από 619 Technic κομμάτια (τουβλάκια, γρανάζια, ρόδες, λάστιχα), 1 προγραμματιζόμενο τούβλο, 3 κινητήρες με ενσωματωμένο αισθητήρα περιστροφών, 4 αισθητήρες (1 αισθητήρα υπερήχων που χρησιμεύει ως μετρητής απόστασης αλλά και ως ανιχνευτής κίνησης, 2 επαφής που δίνουν τη δυνατότητα στο ρομπότ να αντιλαμβάνεται αν ήρθε σε επαφή με κάποιο αντικείμενο και 1 αισθητήρα χρώματος του που επιτρέπει να αντιλαμβάνεται χρώματα, φωτεινότητα και να λειτουργεί ως λαμπάκι, 7 καλώδια διασύνδεσης των αισθητήρων και των κινητήρων με τον προγραμματιζόμενο τούβλο, ένα καλώδιο USB για την επικοινωνία του προγραμματιζόμενου τούβλου με τον Η/Υ, ένα Test Pad για να ελέγξουμε τις δυνατότητες του ρομπότ, ένα έντυπο με οδηγίες για τη δημιουργία του πρώτου μας ρομπότ και πληροφορίες για το λογισμικό NXT-G που το συνοδεύει και τέλος CD-Rom με το λογισμικό NXT-G και drivers.

Εικόνα 12 "Το NXT τούβλο - Στη μέση της εικόνας απεικονίζεται το NXT τούβλο, επάνω οι τρεις κινητήρες και κάτω με σειρά από αριστερά προς τα δεξιά οι δύο αισθητήρες επαφής, ο αισθητήρας χρώματος και ο αισθητήρας υπερήχων"

Όπως φαίνεται και στην προηγούμενη εικόνα υπάρχουν τέσσερα κουμπιά πάνω στο NXT τούβλο. Το πορτοκαλί τετράγωνο χρησιμοποιείται για να το θέσουμε σε λειτουργία ή σε περίπτωση που θέλουμε να πατήσουμε το ”enter” σε κάποια επιλογή του μενού, τα ανοιχτόχρωμα γκρι βέλη χρησιμοποιούνται για να περιηγούμαστε στο μενού δεξιά και αριστερά και τέλος το σκουρόχρωμο γκρι ορθογώνιο χρησιμοποιείτε για να πάμε πίσω ή για την απενεργοποίηση του τούβλου.

Το NXT αποτελείται από τις εξής συσκευές εισόδου – εξόδου:

- Τρεις θύρες παραγωγής για την ένωση των κινητήρων – θύρες A, B και C θύρες αισθητήρων.
- Τέσσερις θύρες εισαγωγής για την ένωση των αισθητήρων - θύρες 1, 2, 3 και 4.
- Μια θύρα USB για την επικοινωνία του NXT με τον υπολογιστή προκειμένου να φορτώσει τα προγράμματα από τον υπολογιστή στο NXT (ή το αντίστροφο). Μπορεί επίσης να χρησιμοποιήσει την ασύρματη σύνδεση Bluetooth για την επικοινωνία με τον υπολογιστή.
- Μεγάφωνο για αναπαραγωγή πραγματικών ήχων.
- Οθόνη υγρών κρυστάλλων ανάλυσης 100 x 64 εικονοστοιχείων για απεικόνιση βασικών λειτουργιών και μηνυμάτων.

Τεχνικά χαρακτηριστικά και λεπτομέρειες υλικού του τούβλου NXT 2.0: (3)

- 32-bit ARM7 μικροελεγκτής
- 256 Kbytes Flash και 64 Kbytes RAM
- 8-bit AVR μικροελεγκτής
- 4 Kbytes Flash και 512 Byte RAM
- Bluetooth για ασύρματη επικοινωνία (Bluetooth class II V2.0)
- Θύρα USB (με ταχύτητα 12 Mbit/s)
- Ηχείο με δυνατότητα αναπαραγωγής με ποιότητα ήχου 8 KHz
- Πηγή ενέργειας: 6 μπαταρίες AA

2.2 ΑΙΣΘΗΤΗΡΕΣ NXT

Οι αισθητήρες είναι ο μοναδικός τρόπος αντίληψης του περιβάλλοντος για ένα ρομπότ. Η Lego συνοδεύει το NXT 2.0 με έναν αισθητήρα υπερήχων, έναν αισθητήρα χρώματος και δύο αισθητήρες επαφής. Επειδή, όμως γνωρίζει το πόσο σημαντικοί είναι οι αισθητήρες δεν σταματάει εκεί, αλλά μας παρέχει πολλούς ακόμη, τους οποίους μπορούμε να αγοράσουμε σε προσιτές τιμές. Επίσης, νέοι, πιο εξελιγμένοι αισθητήρες κυκλοφορούν κάθε τόσο.

2.2.1 ΑΙΣΘΗΤΗΡΑΣ ΥΠΕΡΗΧΩΝ

Ο αισθητήρας υπερήχων (Ultrasonic Sensor) είναι ένας αισθητήρας που δίνει όραση στο ρομπότ μας. Επιτρέπει την ανίχνευση αντικειμένων, και έτσι μπορεί να χρησιμοποιηθεί για την αποφυγή αντικειμένων, για την μέτρηση της απόστασης αλλά και για την ανίχνευση κίνησης. Επίσης, αν τοποθετήσουμε τον αισθητήρα στο περιβάλλον και όχι πάνω στο ρομπότ μπορούμε να το χρησιμοποιήσουμε για να κατευθύνουμε το ρομπότ στην πηγή των υπερήχων.

Έχει τη δυνατότητα μέτρησης σε ίντσες και σε εκατοστά και οι μετρήσεις του μπορεί να κυμαίνονται από 0 έως 170 εκατοστά, με ακρίβεια τριών εκατοστών. Αποτελείται εσωτερικά από έναν πομπό που εκπέμπει τα κύματα και ένα δέκτη που τα λαμβάνει. Τα κύματα που εκπέμπονται είναι ηχητικά κύματα υψηλών συχνοτήτων, γνωστά ως υπερηχητικά κύματα και είναι σύντομα σε διάρκεια (40kHz). Ο υπολογισμός της απόστασης γίνεται υπολογίζοντας το χρόνο που χρειάζεται ένα κύμα για να χτυπήσει ένα αντικείμενο και να επιστρέψει, δηλαδή δουλεύει ακριβώς με τον ίδιο τρόπο με τον οποίο βλέπει η νυχτερίδα. Φυσικά η απόσταση που διανύει συνολικά ένα κύμα θα είναι διπλάσια από την απόσταση στην οποία βρίσκετε το αντικείμενο. Σε περίπτωση που υπάρχουν περισσότερα από ένα αντικείμενα στο οπτικό του πεδίο επιστρέφει την απόσταση του πιο κοντινού. Τα αντικείμενα με μεγάλο μέγεθος και σκληρή επιφάνεια επιστρέφουν καλύτερα τις ανακλάσεις από ότι τα κυρτά ή μαλακά ή μικρά αντικείμενα. Σημαντικό είναι να αναφέρουμε ότι διαθέτει δικό του επεξεργαστή για να κάνει τους υπολογισμούς που χρειάζονται.

Εικόνα 13 "Αισθητήρας υπερήχων"

Εικόνα 14 "Τρόπος λειτουργίας αισθητήρα υπερήχων - Ένα κύμα στέλνεται από τον πομπό (βλέπε κόκκινο χρώμα) και λαμβάνεται από το δέκτη (βλέπε μπλε χρώμα) "

2.2.2 ΑΙΣΘΗΤΗΡΑΣ ΕΠΑΦΗΣ

Ο αισθητήρας επαφής (Touch Sensor) δίνει στο ρομπότ μας την αίσθηση της αφής. Μπορεί να αντιληφθεί πότε πιέζεται και πότε απελευθερώνεται. Χρησιμοποιείτε συνήθως σε ρομπότ που κινούνται στο χώρο για να μπορούν να αντιληφθούν πότε συγκρούονται με ένα αντικείμενο ή σε ρομποτικούς βραχίονες για να μπορούν να ελέγχουν εάν υπάρχει ή όχι κάτι στο βραχίονα.

Εικόνα 15 "Αισθητήρας Επαφής"

2.2.3 ΑΙΣΘΗΤΗΡΑΣ ΦΩΤΟΣ

Ο αισθητήρας φωτός (Light Sensor), όπως και ο αισθητήρας υπερήχων, είναι ένας αισθητήρας που δίνει όραση στο ρομπότ μας. Επιτρέπει την μέτρηση του φωτισμού σε ένα χώρο, έτσι μπορούμε να καταλάβουμε αν έχει σκοτάδι ή φως αλλά μας δίνει επίσης τη δυνατότητα με το κατάλληλο πρόγραμμα να διαβάσουμε μία ασπρόμαυρη εικόνα. Επιπλέον μπορούμε με τη χρήση του αισθητήρα αυτού να κάνουμε το ρομπότ μας να ακολουθεί μία γραμμή (για παράδειγμα μία μαύρη γραμμή σε ένα άσπρο χαρτόνι).

Εικόνα 16 "Αισθητήρας Φωτός"

2.2.4 ΑΙΣΘΗΤΗΡΑΣ ΧΡΩΜΑΤΟΣ

Ο αισθητήρας χρώματος είναι κι αυτός, όπως του φωτός και των υπερήχων, ένας αισθητήρας που δίνει την δυνατότητα της όρασης στο ρομπότ μας. Αυτός ο αισθητήρας έχει 3 διαφορετικές λειτουργίες:

- Μπορεί να διακρίνει 6 διαφορετικά χρώματα
    - άσπρο
    - μαύρο
    - κόκκινο
    - μπλε
    - πράσινο
    - κίτρινο.
- Λειτουργεί ως αισθητήρας φωτός ξεχωρίζοντας τις φωτεινές από τις σκοτεινές περιοχές.
- Λειτουργεί ως απλό led λαμπάκι που μπορεί να ανάβει σε διάφορα χρώματα.

Εικόνα 17 "Αισθητήρας Χρώματος"

2.2.5 ΑΙΣΘΗΤΗΡΑΣ ΗΧΟΥ

Ο αισθητήρας ήχου (Sound Sensor) είναι αυτός που δίνει τη δυνατότητα ακοής στο ρομπότ μας. Έχει τη δυνατότητα να ανιχνεύει decibels (DB – περιλαμβάνουν ήχους που μπορεί να ακούσει το ανθρώπινο αυτί, καθώς και υψηλότερους και χαμηλότερους από αυτούς), αλλά και ρυθμισμένα decibels (DBA – ήχοι που μπορεί να ακούσει το αυτί του ανθρώπου). Μετράει τα επίπεδα έντασης κυμάτων που ταξιδεύουν στην ατμόσφαιρα από 2Hz έως 20kHz με μεγάλη ακρίβεια, ενώ η μέγιστη ένταση που οι προδιαγραφές του επιτρέπουν να ανιχνεύσει είναι τα 90dB. Διαθέτει δύο καταστάσεις λειτουργίας. Κατάσταση λειτουργίας που μετράει την ένταση σε dB και κατάσταση λειτουργίας που μετράει την ένταση σε dBA. Επιπλέον έχει τη

δυνατότητα να ξεχωρίζει τα διάφορα επίπεδα έντασης. Επειδή τα επίπεδα έντασης είναι αρκετά περίπλοκα και εμφανίζονται σε ποσοστό τοις εκατό. Όσο μικρότερο είναι το ποσοστό τόσο πιο χαμηλός είναι και η ένταση του ήχου και το ανάποδο. Πιο συγκεκριμένα είναι:

- 4-5% μπορεί να ανιχνεύσει τον θόρυβο από την αναπνοή μας ή από το βάδισμα
- 5-10% μπορεί να αντιληφθεί κάποιον που μιλάει από μακριά
- 10-30% ανιχνεύει την κανονική ομιλία δίπλα στον αισθητήρα και μουσική που παίζει σε κανονικό επίπεδο
- 30-100% σε αυτό το ποσοστό συμπεριλαμβάνονται η δυνατή μουσική αλλά και κάποιος που φωνάζει

Εικόνα 18 "Αισθητήρας Ήχου"

2.2.6 ΑΙΣΘΗΤΗΡΑΣ ΠΥΞΙΔΑ

Ο αισθητήρας της πυξίδας (Compass Sensor) μπορεί και υπολογίζει το μαγνητικό πεδίο της γης και έτσι μπορεί να υπολογίζει και μας ενημερώνει για την κατεύθυνση προς την οποία είναι στραμμένο το ρομπότ. Επιπλέον στην πυξίδα συμπεριλαμβάνεται ένας βαθμονομητής (calibrator) για να βοηθάει στη μείωση του σφάλματος από μαγνητικές παρεμβολές άλλων πηγών.

Εικόνα 19 "Αισθητήρας Πυξίδα"

2.2.7 ΑΙΣΘΗΤΗΡΑΣ ΕΠΙΤΑΧΥΝΣΗΣ

Ο αισθητήρας επιτάχυνσης (Accelerometer Sensor) μας επιτρέπει να γνωρίζουμε ποια μεριά του ρομπότ μας είναι η πάνω μεριά και πότε γέρνει δεξιά ή αριστερά. Επίσης, υπολογίζει και μας δίνει πληροφορίες για την επιτάχυνση.

Εικόνα 20 "Αισθητήρας επιτάχυνσης"

2.2.8 ΑΙΣΘΗΤΗΡΑΣ ΓΥΡΟΣΚΟΠΙΟ

Ο αισθητήρας γυροσκόπιο (Gyroscopic Sensor) δίνει τη δυνατότητα στο ρομπότ μας να ανιχνεύει την περιστροφή με αποτέλεσμα τα προγράμματά μας να έχουν μεγάλη ακρίβεια όσο αναφορά την κατεύθυνση του ρομπότ. Η ανίχνευση περιστροφής γίνεται με τη χρήση αντηχείου χαλαζία. Είναι πολύ χρήσιμο όταν θέλουμε να διατηρήσουμε σταθερό προσανατολισμό.

Εικόνα 21 "Αισθητήρας Γυροσκόπιο"

2.2.9 ΑΙΣΘΗΤΗΡΑΣ ΕΥΡΕΣΗΣ

Ο αισθητήρας εύρεσης (Seeker Sensor) δίνει τη δυνατότητα στο ρομπότ μας να ανιχνεύει υπέρυθρες πηγές φωτός και να καθορίζει την κατεύθυνση τους αλλά και τη σχετική δύναμη του φωτός χάρη στους 5 ανιχνευτές υπέρυθρων που διαθέτει παρατεταμένους ανά διαστήματα 60º.

Εικόνα 22 "Αισθητήρας Εύρεσης"

2.3 ΚΙΝΗΤΗΡΕΣ NXT

Οι κινητήρες είναι εξίσου σημαντικοί με τους αισθητήρες αφού δίνουν τη δυνατότητα στο ρομπότ μας να κινεί διάφορα μέρη του αλλά και να κινείται το ίδιο στο χώρο. Το NXT διαθέτει σερβομοτέρ και όχι απλά μοτέρ. Γενικά στη ρομποτική οι σερβομηχανισμοί χρησιμοποιούνται πολύ συχνά και είναι συσκευές οι οποίες εκτός από τη βασική λειτουργία που επιτελούν, έχουν και έναν αισθητήρα σφάλματος τον οποίο χρησιμοποιούν για ανατροφοδότηση (feedback) με σκοπό τη βελτίωση της λειτουργίας και τη μείωση του τελικού σφάλματος.

Εικόνα 23 "Κινητήρας της Lego"

Πιο συγκεκριμένα όσο αφορά τους κινητήρες έχουμε σημαντικές διαφορές στην απόδοση ενός κοινού μοτέρ και ενός σερβομοτέρ (όπως αυτά που έχει το NXT). Αν και στα δύο δώσουμε την ίδια εντολή, να κινηθούν για κάποιο συγκεκριμένο χρονικό διάστημα μπορεί να έχουμε διαφορετικό αποτέλεσμα. Αυτό συμβαίνει επειδή αν στο κοινό μοτέρ η κίνηση εμποδίζεται από κάτι ή η τάση δεν είναι αρκετή για να το κινήσει, επειδή είναι για παράδειγμα σε μία ανηφόρα, αυτό θα μείνει κολλημένο στην ίδια θέση χωρίς να το γνωρίζει το πρόγραμμά μας, με αποτέλεσμα να συνεχίσει παρακάτω σαν να είχε κινηθεί το ρομπότ. Με τα σερβομοτέρ όμως είναι διαφορετικά. Ο αισθητήρας σφάλματος (πρόκειται ουσιαστικά για ένα ταχομετρητή - οδομετρητή) που περιέχεται, παρακολουθεί συνέχεια τη θέση του μοτέρ και ανάλογα με τα αποτελέσματα αυξάνει ή μειώνει την τάση. Για παράδειγμα, αν κάποια ρόδα κολλήσει σε ένα σημείο, θα αυξηθεί η τάση για να ξεκολλήσει. Έτσι, με αυτό τον τρόπο, μπορούμε να διατηρήσουμε σταθερή την ταχύτητα περιστροφής στους τροχούς του ρομπότ μας και να εκτελέσουμε εύκολα ομαλή επιτάχυνση και επιβράδυνση.

Εικόνα 24 "Εσωτερικός μηχανισμός του κινητήρα"

Στο σερβομοτέρ του NXT η καταγραφή της περιστροφής γίνεται με ακρίβεια μίας μοίρας. Ο τρόπος με τον οποίο λειτουργεί είναι παρόμοιος με αυτός που λειτουργούσαν παλιά τα ποντίκια των υπολογιστών που είχαν περιστρεφόμενη μπίλια. Πιο συγκεκριμένα, καθώς περιστρέφεται το μοτέρ, περιστρέφει μαζί του και μία ακολουθία από γρανάζια, όπως φαίνεται στην εικόνα παραπάνω. Ένα από αυτά τα γρανάζια είναι ειδικά κατασκευασμένο, με μία σειρά από ειδικές οπές μέσα από τις οποίες περνάει μία δέσμη φωτός. Κάθε φορά που αυτή η δέσμη διακόπτεται έχουμε περιστροφή μίας μοίρας. Ο έλεγχος της γωνίας περιστροφής γίνεται από τον ελεγκτή του προγραμματιζόμενου τούβλου και προκειμένου η κίνηση να είναι ακριβής γίνονται διαρκείς έλεγχοι του ταχογράφου. Αυτή η δουλεία είναι χαμηλότερου επιπέδου και την αναλαμβάνει το firmware που έχουμε εγκατεστημένο. Επιπλέον, είναι χρήσιμο να αναφέρουμε ότι όταν θέλουμε να συγχρονίσουμε πολλά μοτέρ μεταξύ τους χρησιμοποιείται το κεντρικό ρολόι του συστήματος για τον συγχρονισμό τους.

2.4 BLUETOOTH

Το Bluetooth είναι αυτό που επιτρέπει την ασύρματη επικοινωνία μεταξύ του υπολογιστή μας και του NXT. Αυτό είναι πολύ σημαντικό επειδή πολλές φορές χρειάζεται να στείλουμε δεδομένα στον υπολογιστή είτε για να τα επεξεργαστεί πιο γρήγορα και να μας επιστρέψει τα αποτελέσματα, είτε επειδή ο όγκος των δεδομένων που έχουμε είναι πολύ μεγάλος για να χωρέσει στη μνήμη του NXT.

Εικόνα 25 "Εικονίδιο Bluetooth"

2.5 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΤΟΥ NXT

Το προγραμματιζόμενο τούβλο NXT όπως έχουμε ήδη αναφέρει αποτελεί τον εγκέφαλο του ρομπότ και ο χρήστης μπορεί να γράψει και να τρέξει τα δικά του προγράμματα. Εκτός από των κώδικα όμως, στο τούβλο υπάρχει πάντα ένα ενδιάμεσο κομμάτι λογισμικού, το firmware που είναι πρόγραμμα το οποίο ελέγχει εσωτερικά τις λειτουργίες μίας συσκευής. Το firmware αναφέρεται συχνά στα ελληνικά ως υλικολογισμικό και είναι συνήθως γραμμένο σε γλώσσα μηχανής ή σε συμβολική γλώσσα. Στην περίπτωση αυτή όμως, επειδή τα firmwares φορτώνουν τα προγράμματα που έχουμε γράψει και θέλουμε να εκτελέσουμε, μπορούν να θεωρηθούν μινιμαλιστικά λειτουργικά συστήματα. Γενικά ο στόχος τους είναι να αναλαμβάνουν τον έλεγχο του hardware, την επικοινωνία με τα περιφερειακά, να προσφέρουν βιβλιοθήκες (με έτοιμες κλάσεις και μεθόδους) ώστε να μπορεί να γίνει ο προγραμματισμός του ρομπότ σε κάποια γλώσσα υψηλότερου επιπέδου, να γίνεται η διαχείριση της μνήμης και διαχείριση των αρχείων που έχουν φορτωθεί. Ένα πολύ σημαντικό χαρακτηριστικό του NXT είναι ότι αν το αγοράζουμε συνοδεύεται από το firmware της Lego, δεν είναι κλειδωμένο, οπότε μπορούμε να εγκαταστήσουμε οποιοδήποτε άλλο θέλουμε εμείς. Σημαντικό είναι να γνωρίζουμε ότι η εγκατάσταση ενός νέου firmware θα διαγράψει όλα τα αρχεία που υπήρχαν στο ρομπότ μας. Αν θελήσουμε να ξαναχρησιμοποιήσουμε το γραφικό περιβάλλον που παρέχεται από τη LEGO θα πρέπει απλά να χρησιμοποιήσουμε το software που υπήρχε στο κουτί του LEGO ρομπότ μας.

2.6 NXT FIRMWARES ΚΑΙ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Τα firmwares για το NXT που υπάρχουν διαθέσιμα είναι πολλά και αφορούν διαφορετικές γλώσσες προγραμματισμού υψηλού επιπέδου. Παρακάτω παρουσιάζονται τα πιο σημαντικά και ευρέως χρησιμοποιούμενα από αυτά:

- NXT-G

Είναι το firmware και το λογισμικό το οποίο συνοδεύει το NXT με την αγορά του και έχει ως στόχο την εύκολή δημιουργία απλών προγραμμάτων χωρίς να απαιτούνται γνώσεις προγραμματισμού. Ο προγραμματισμός είναι μόνο γραφικός. Ουσιαστικά, ο χρήστης δημιουργεί ένα διάγραμμα από blocks-εντολές σύροντας τα και φτιάχνοντας συνδέσμους μεταξύ τους. Το πρόγραμμα είναι πολύ εύκολο στην εκμάθηση και μπορεί να χρησιμοποιηθεί και από παιδιά. Είναι πολύ καλή επιλογή για να γίνουν δοκιμές που θα ελέγχουν τους αισθητήρες και θα κινούν το ρομπότ μας αλλά δεν είναι κατάλληλο για την ανάπτυξη κάποιου πολυπλοκότερου προγράμματος. Αυτό συμβαίνει επειδή στα πολύ μεγάλα προγράμματα η εικόνα που δημιουργείται είναι περίπλοκη και οι διορθώσεις είναι πολύ δύσκολες. Επιπλέον, τα εκτελέσιμα που δημιουργούνται έχουν μεγάλο μέγεθος με αποτέλεσμα την αργή αρχικοποίηση των προγραμμάτων. Τέλος, άλλο ένα μειονέκτημα είναι ότι δε λειτουργεί σε Linux, αλλά μόνο σε Windows και Mac.

- LabVIEW

Το firmware είναι αντίστοιχο με το προηγούμενο που περιγράψαμε παραπάνω και η μόνη διαφορά του είναι στο γραφικό περιβάλλον που χρησιμοποιεί. Πιο συγκεκριμένα, υπάρχουν δύο τόσο παρόμοια λογισμικά, επειδή στην αρχή η Lego για τη δημιουργία του NXT-G συνεργάστηκε με την National Instrument, η οποία αναπτύσσει το LabVIEW (ένα εργαλείο visual programming). Η συνεργασία τους οδήγησε στη δημιουργία του ROBOLAB το οποίο μετονομάστηκε έπειτα στο γνωστό μας NXT-G. Κατά τη συνεργασία αυτή, η National Instrument έκανε και το δικό της εργαλείο για το NXT, το LabVIEW. Αν και, όταν κυκλοφόρησε είχε περισσότερες δυνατότητες από το αντίστοιχο της Lego, δεν έχει ανανεωθεί από το 2006.

- LeJOS NXJ

Το LeJOS είναι ένα ανοιχτού κώδικα firmware, αρκετά διαφορετικό από αυτό της Lego. Δεν έχει γραφικό περιβάλλον και απαιτεί ο χρήστης να έχει γνώσεις προγραμματισμού. Ειδικότερα θα πρέπει να μπορεί να προγραμματίσει στην υψηλού επιπέδου γλώσσα προγραμματισμού Java. Οι βιβλιοθήκες που προσφέρονται είναι πλήρεις, περιλαμβάνοντας μεθόδους για τους κινητήρες, όλους τους αισθητήρες αλλά και επιπλέον βοηθητικές μεθόδους, για παράδειγμα για την εύκολη πλοήγηση του ρομπότ μας. Επίσης, είναι από τα λίγα firmwares που αξιοποιούν τη δυνατότητα χρήσης των Bluetooth που διαθέτει το NTX. Μπορούμε να χρησιμοποιήσουμε οποιοδήποτε περιβάλλον ανάπτυξης θέλουμε όπως το Eclipse, το Netbeans, το JCreator κ.α. Επιπλέον, μας δίνεται η δυνατότητα να δημιουργήσουμε όσο περίπλοκα προγράμματα θέλουμε, ενώ ταυτόχρονα η ταχύτητα εκτέλεσης είναι ικανοποιητική και δεν γίνεται κατασπατάληση της μνήμης.

- NBC/NXC

Η NBC (Next Byte Codes) είναι μία γλώσσα ανοιχτών προδιαγραφών παρόμοια με την assembly. Επειδή όμως ο προγραμματισμός σε μία τέτοια γλώσσα, που είναι χαμηλού επιπέδου, είναι δύσκολος συνήθως η NBC δε χρησιμοποιείται μόνη της, αλλά σε συνδυασμό με την NXC (Not eXactly C). Η NXC είναι μία υψηλότερου επιπέδου γλώσσα που μοιάζει αρκετά με τη C. Ουσιαστικά, η NXC έχει χτιστεί πάνω στην NBC και συχνά χρησιμοποιεί συναρτήσεις που είναι γραμμένες σε assembly. Χρησιμοποιείται ευρέως και είναι από τις πιο δημοφιλής μετά την NXT-G της Lego. Για να είναι πιο εύκολη η συγγραφή και η μεταγλώττιση του κώδικα μας, έχει αναπτυχθεί και ένα γραφικό περιβάλλον, το BricxCC (Bricx Command Center). Το μεγαλύτερο πλεονέκτημά της είναι ότι χρησιμοποιείται από πάρα πολλά άτομα και είναι πολύ γνωστή. Τα βασικότερα μειονεκτήματά της είναι ότι δεν υποστηρίζει δισδιάστατους πίνακες ή άλλες δομές και στη βιβλιοθήκη της δεν υπάρχουν άλλες βοηθητικές μέθοδοι, πέρα από αυτές που αφορούν τους αισθητήρες και τους κινητήρες.

- Matlab & Simulink

Η Matlab είναι μία γλώσσα προγραμματισμού υψηλού επιπέδου που αν και χρησιμοποιείται κυρίως για ανάπτυξη περίπλοκων αλγορίθμων, την ανάλυση δεδομένων και μαθηματικά υψηλών απαιτήσεων, μπορεί να χρησιμοποιηθεί και για τον προγραμματισμό του NXT. Το πιο μεγάλο πλεονέκτημα της Matlab είναι ότι μπορούμε να τη χρησιμοποιήσουμε σε συνδυασμό με το Simulink, το οποίο είναι ένα περιβάλλον ανάπτυξης που μας δίνει τη δυνατότητα να δημιουργήσουμε μοντέλα εξομοίωσης. Έτσι, μπορούμε αντί να εκτελούμε κάθε φορά στο ρομπότ τον κώδικά μας που έχει γραφτεί σε Matlab, να κάνουμε εξομοίωσή του στο Simulink.

- RobotC

Το RobotC μας δίνει τη δυνατότητα να προγραμματίζουμε χρησιμοποιώντας τη γλώσσα προγραμματισμού C. Ο χρήστης πρέπει να γνωρίζει τη γλώσσα για να μπορεί να προγραμματίσει αν και υπάρχουν πολλά παραδείγματα για να τον βοηθήσουν να μάθει στην αρχή. Δεν υπάρχει κάποια αλλοίωση στη γλώσσα, είναι η κοινή εμπορική C που χρησιμοποιείται συχνά με επιπλέον μεθόδους για τον έλεγχο των αισθητήρων και των κινητήρων του ρομπότ μας.

- PyNXC

Ο κώδικας γράφεται σε Python και στη συνέχεια μετατρέπεται σε NXC. Υπάρχουν πολλά παραδείγματα για να βοηθήσουν τον ενδιαφερόμενο να ξεκινήσει να προγραμματίζει σε Python. Δεν χρησιμοποιείται συχνά στη χώρα μας, μιας και η γλώσσα αυτή δεν χρησιμοποιείται ακόμη ευρέως.

- NXT-Python

Για τη συγγραφή των προγραμμάτων χρησιμοποιείται η γλώσσα προγραμματισμού Python. Υποστηρίζει και τη χρήση του Bluetooth που έχει το NXT, ενώ ταυτόχρονα έχει πολλές βοηθητικές μεθόδους και για αισθητήρες που μπορούν να αγοραστούν μετέπειτα.

- BrickOS & nxtOSEK

Το BrickOS και το nxtOSEK μας παρέχει τη δυνατότητα να προγραμματίζουμε το ρομπότ μας σε C και σε C++. Ο χρήστης απαιτείται να έχει γνώσεις προγραμματισμού και πιο συγκεκριμένα να μπορεί να προγραμματίζει σε C ή /και C++.

- Άλλες γλώσσες

Άλλες γνωστές γλώσσες που χρησιμοποιούνται είναι η Visual Basic (μέσω του COM+ interface), η pbFORTH (επέκταση της γλώσσας προγραμματισμού Forth), η Ruby (με τη χρήση της βιβλιοθήκης ruby-nxt), και η C σε συνδυασμό με τα Microsoft Robotics Developer Studio, BricxCC και NXTGCC.

ΕΠΙΛΟΓΟΣ

Το αντικείμενο μελέτης του κεφαλαίου αυτού ήταν το ρομπότ της εταιρίας Lego, το Lego Mindstorm NXT. Αρχικά, αναφερθήκαμε στην ιστορία του εκπαιδευτικού αυτού παιχνιδιού και είδαμε αναλυτικά τα μέρη από τα οποία αποτελείται ένα NXT ρομπότ. Έπειτα, αναφερθήκαμε στους αισθητήρες και τους κινητήρες που υπάρχουν διαθέσιμοι στην αγορά και τις δυνατότητές τους. Τέλος, είδαμε τον τρόπο με τον οποίο μπορούμε να προγραμματίσουμε το NXT τούβλο, τα διάφορα firmwares που υπάρχουν και τις γλώσσες προγραμματισμού τις οποίες μπορούμε να χρησιμοποιήσουμε. Από τα πολλά διαθέσιμα λογισμικά, εμείς επιλέξαμε να χρησιμοποιήσουμε το LeJOS που μας δίνει τη δυνατότητα να προγραμματίσουμε σε Java.

ΚΕΦΑΛΑΙΟ 3

ΤΟ LEJOS

ΕΙΣΑΓΩΓΗ

Για τη δημιουργία του προγράμματός μας χρησιμοποιήσαμε το LeJOS. Η επιλογή αυτή βασίζεται κυρίως στα πλεονεκτήματα που έχει σε σχέση με άλλα firmwares, τα οποία αναφέρονται παρακάτω, αλλά και το γεγονός ότι η γλώσσα προγραμματισμού είναι η Java, η οποία είναι μία γλώσσα υψηλού επιπέδου, αντικειμενοστραφής, με σχετικά εύκολη απασφαλμάτωση που υποστηρίζει πολυδιάστατους πίνακες και άλλες δομές.

Για την ανάπτυξη της εφαρμογής μας, θα χρησιμοποιήσουμε το Eclipse. Το Eclipse είναι ένα ισχυρό περιβάλλον προγραμματισμού με πολλές δυνατότητες. Ο κυριότερος λόγος που το χρησιμοποιούμε είναι ότι στο Eclipse υπάρχει διαθέσιμο ένα plugin για το LeJOS. Αυτό μας δίνει τη δυνατότητα να δημιουργούμε project τύπου NXJ και επιπλέον μας παρέχει έναν πολύ απλό τρόπο για να κάνουμε upload το firmware του LeJOS, πατώντας απλά ένα κουμπί. Επιπλέον, η μεταγλώττιση του κώδικα, η μεταφορά του στο NXT τούβλο καθώς και η απασφαλμάτωση είναι ευκολότερη. Τέλος, άλλο ένα πλεονέκτημα είναι ότι όταν δημιουργούμε ένα NXJ project, αυτόματα σε αυτό προστίθεται το αρχείο classes.jar, δηλαδή η βιβλιοθήκη που χρειαζόμαστε για να δημιουργήσουμε προγράμματα για το ρομπότ μας. Αυτό μας γλιτώνει από τον κόπο να προσθέσουμε χειροκίνητα τη βιβλιοθήκη και μας δίνει τη δυνατότητα να αρχίσουμε να γράφουμε άμεσα τον κώδικά μας. Αναλυτικές οδηγίες για όσα αναφέραμε υπάρχουν στο παράρτημα ΙΙ.

3.1 ΠΕΡΙΓΡΑΦΗ ΤΟΥ LEJOS

Όπως ήδη έχουμε αναφέρει, το LeJOS είναι ένα firmware ανοιχτού κώδικα που μας παρέχει τη δυνατότητα συγγραφής προγραμμάτων για το NXT ρομπότ μας σε Java. Ουσιαστικά πρόκειται για ένα μινιμαλιστικό λειτουργικό σύστημα και αυτό συμβολίζουν και τα δύο τελευταία γράμματα του ονόματός του (OS: Operating System). Στα αγγλικά, το όνομά του είναι παρόμοιο με τη λέξη Legos, εκτός από το γεγονός ότι στη θέση του γράμματος g υπάρχει ένα j το οποίο προέρχεται από τη λέξη Java. Η σωστή προφορά θα ήταν Ley-J-oss.

Το προγραμματιστικό περιβάλλον LeJOS αναπτύχθηκε εξολοκλήρου από τον Jose Solorzano, αλλά τώρα συντηρείται από τους Paul Andrews και Jürgen Stuber. Ο πυρήνας του και ο χαμηλού επιπέδου κώδικας είναι γραμμένος σε C, ενώ το μεγαλύτερο μέρος της βιβλιοθήκης του αλλά και τα προγράμματα που γράφει ο χρήστης χρησιμοποιούν τη Java. Συγκεκριμένα αποτελείται από:

- Ένα εναλλακτικό firmware για το NXT τούβλο που υποστηρίζει τη χρήση της γλώσσας προγραμματισμού Java.

- Μια βιβλιοθήκη με κλάσεις και πολλές χρήσιμες μεθόδους για τους αισθητήρες και τους κινητήρες καθώς και επιπλέον βοηθητικές μεθόδους για την κίνηση των ρομπότ και τη συμπεριφορά τους.
- Εργαλεία για τη φόρτωση των προγραμμάτων και για πολλές άλλες λειτουργίες.
- Άφθονα παραδείγματα που είναι πολύ βοηθητικά στην αρχή.

3.2 ΠΛΕΟΝΕΚΤΗΜΑΤΑ LEJOS

Υπάρχουν πολλά πλεονεκτήματα του LeJOS συγκριτικά με το NXT-G και με άλλα περιβάλλοντα προγραμματισμού του NXT:

- Χρησιμοποιεί τη γνωστή-τυποποιημένη γλώσσα της Java
- Μας παρέχει αντικειμενοστραφή προγραμματισμό και προηγμένες δυνατότητες όπως αυτόματη διαχείριση μνήμης
- Είναι γρηγορότερο από το NXT-G
- Είναι ανοιχτού κώδικα
- Υποστηρίζει όλα τα λειτουργικά συστήματα: Windows, Mac Os και Linux
- Περιέχει μία πολύ πλούσια βιβλιοθήκη η οποία υποστηρίζει αρκετές από τις συναρτήσεις του Java SDK και περιέχει μεθόδους για τους κινητήρες και τους αισθητήρες
- Μας δίνει τη δυνατότητα να ελέγξουμε τους κινητήρες με ιδιαίτερη ακρίβεια
- Υποστηρίζει πλήρως τη χρήση των Bluetooth
- Τα προγράμματα και το ίδιο το firmware δε χρειάζονται πολλή μνήμη
- Περιέχει βοηθητικές μεθόδους για την πλοήγηση και μερικούς κλασικούς αλγόριθμους ρομποτικής, όπως αλγόριθμους πλοήγησης Monte Carlo
- Μας παρέχει εργαλεία για τον ευκολότερο χειρισμό σύνθετων συμπεριφορών του ρομπότ
- Υποστηρίζει αισθητήρες που δεν συμπεριλαμβάνονται στο NXT, αλλά μπορεί κάποιος να προμηθευτεί από τη Lego
- Μας παρέχει τριγωνομετρικές μεθόδους και άλλες μαθηματικές λειτουργίες

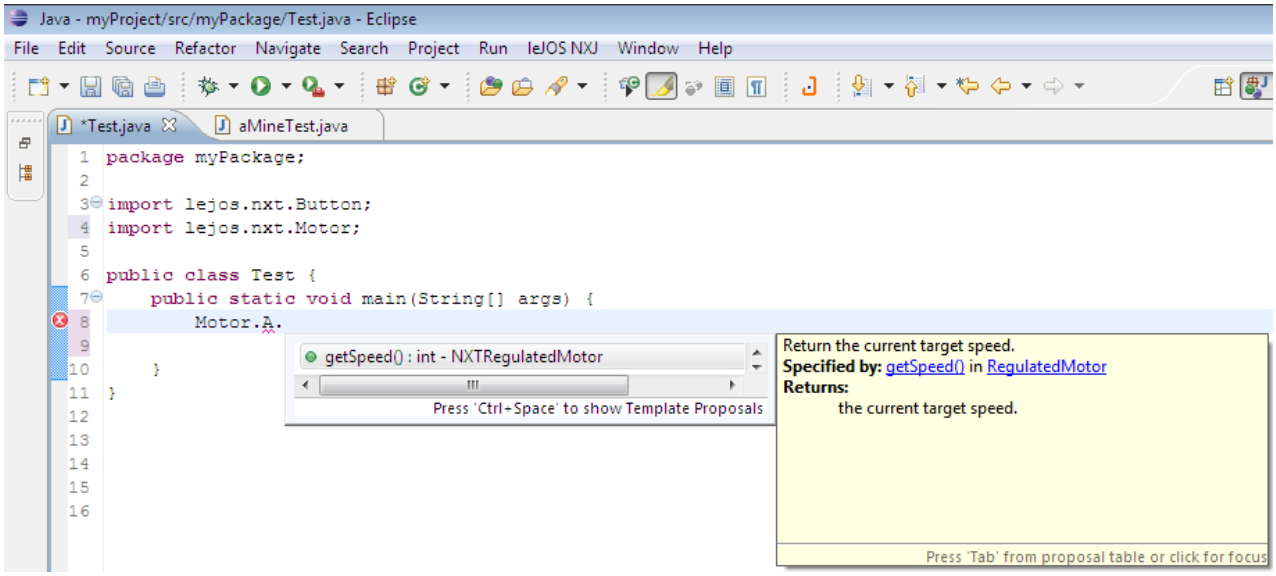
- Υποστηρίζει το J2ME LCD UI συμπεριλαμβανομένων πολλών λειτουργιών γραφικής παράστασης
- Υποστηρίζει ταυτόχρονες διεργασίες
- Μας παρέχει πολλά παραδείγματα προγραμμάτων
- Υποστηρίζει επικοινωνία μεταξύ δύο NXT μέσω Bluetooth
- Υποστηρίζει την αναπαραγωγή αρχείων pcm mono WAV των 8bits
- Υποστηρίζει μακρινή εκτέλεση από το PC χρησιμοποιώντας iCommand
- Υπάρχει φόρουμ για την παροχή βοήθειας στους χρήστες αλλά και την επικοινωνία με την ομάδα ανάπτυξης
- Υποστηρίζει telerobotics μέσω τυποποιημένων υποδοχών TCP/*IP
- Έχει εύχρηστο σύστημα επιλογών
- Χρησιμοποιείται ευρέως από τα πανεπιστήμια και από άλλα ιδρύματα εκπαίδευσης σε όλο τον κόσμο

3.3 ΒΑΣΙΚΕΣ ΜΕΘΟΔΟΙ ΤΟΥ LEJOS

Η συγγραφή κώδικα για το ρομπότ μας με τη χρήση του LeJOS, είναι απλή, αρκεί να γνωρίζουμε Java, γιατί μας παρέχει όλες τις μεθόδους που μπορεί να χρειαστούμε για τον χειρισμό των κινητήρων και των αισθητήρων του ρομπότ μας, ακόμη και αισθητήρων που δεν συμπεριλαμβάνονται στο κουτί του NXT με την αγορά του, αλλά μπορούμε να προμηθευτούμε από τη Lego. Επίσης, υποστηρίζει πλήρως τη χρήση του Bluetooth και περιλαμβάνει και επιπλέον μεθόδους για να μας βοηθήσει για παράδειγμα στον έλεγχο κινούμενων ρομπότ. Παρακάτω, θα παρουσιάσουμε κάποιες από τις πιο σημαντικές και συχνότερα χρησιμοποιούμενες μεθόδους που υπάρχουν. Φυσικά, όποιος ενδιαφέρεται μπορεί να βρει όλα τα πακέτα, τις κλάσεις και τις μεθόδους που είναι διαθέσιμα στην ιστοσελίδα του LeJOS μαζί με πολλές πληροφορίες, επεξηγήσεις και παραδείγματα. Πιο συγκεκριμένα, αξίζει κανείς να επισκεφτεί την ιστοσελίδα: http://lejos.sourceforge.net/nxt/nxj/api/index.html όπου θα βρει αναλυτικά οτιδήποτε χρειάζεται.

Είναι πολύ χρήσιμο να γνωρίζουμε ότι αν θέλουμε να δούμε όλες τις διαθέσιμες μεθόδους που υπάρχουν για κάποιο συγκεκριμένο στοιχείο, για παράδειγμα για τους κινητήρες στο LeJOS, αρκεί να γράψουμε στο Eclipse *Motor.A*. (που δηλώνει σε ποια θέση είναι συνδεδεμένος ο κινητήρας) και ένα παράθυρο με όλες τις πληροφορίες που θέλουμε, διαθέσιμες μεθόδους και την περιγραφή τους, θα εμφανιστεί όπως φαίνεται στην εικόνα 26.

Ακολουθεί η περιγραφή των πιο σημαντικών κλάσεων που μας παρέχονται καθώς και πολλών χρήσιμων μεθόδων και έτοιμων προγραμμάτων που υπάρχουν για να κάνουν τη δουλειά μας ως προγραμματιστές ακόμη πιο εύκολη.



Εικόνα 26 "Προγραμματισμός LeJOS σε Eclipse - Για να δούμε όλες τις διαθέσιμες μεθόδους για έναν κινητήρα μπορούμε να γράψουμε απλά Motor.A. και το Eclipse θα μας εμφανίσει ένα παράθυρο με όλες τις διαθέσιμες μεθόδους και την περιγραφή τους"

3.3.1 ΚΙΝΗΤΗΡΕΣ

Για τον έλεγχο ενός κινητήρα, το LeJOS έχει έτοιμη για εμάς μία κλάση με όνομα Motor η οποία μας παρέχει πολλές χρήσιμες μεθόδους. Όπως ήδη έχουμε αναφέρει, πάνω στο NXT τούβλο υπάρχουν τρεις θύρες για τη σύνδεση των κινητήρων, η θύρα A, η B και η C. Έτσι, για τη διευκόλυνση των προγραμματιστών, και στον κώδικα αναφερόμαστε σε αυτά με αυτό τον τρόπο. Για παράδειγμα, για να αναφερθούμε στον κινητήρα που είναι συνδεδεμένος στη θύρα C, γράφουμε *Motor.C*.

Οι μέθοδοι που μας παρέχονται χρησιμεύουν για να ελέγχουμε έναν κινητήρα αλλά και για να μαθαίνουμε ποια είναι η τωρινή του κατάσταση.

- *forward()* - κάνει τον κινητήρα να κινείται προς τα μπροστά
- *backward()* - κάνει τον κινητήρα να κινείται προς τα πίσω
- *stop()* - σταματάει τον κινητήρα
- *rotate(int angle)* - περιστρέφει τον κινητήρα όσες μοίρες ορίσουμε και κάνει return όταν ολοκληρωθεί η περιστροφή
- *rotateTo(int angle)* - περιστρέφει τον κινητήρα όσες μοίρες ορίσουμε και κάνει return αμέσως ενώ ο κινητήρας συνεχίζει να κινείται μέχρι να ολοκληρωθεί η περιστροφή

- *getTachoCount()* - μας επιστρέφει σε μοίρες το πόσο έχει κινηθεί ο κινητήρας

- *setSpeed(int speed)* - θέτουμε την ταχύτητα (μοίρες ανά δευτερόλεπτο) με την οποία θέλουμε να κινηθεί ο κινητήρας. Η μέγιστη ταχύτητα που μπορεί να διατηρηθεί σταθερά είναι ίση με περίπου 110 φορές της ισχύς της μπαταρίας

- *isRotating()* - Χρησιμοποιείται για να ελέγξουμε αν ένας κινητήρας είναι σταματημένος. Αν έχει σταματήσει μας επιστρέφει false

- *isMoving()* - Μας δίνει τη δυνατότητα να ελέγχουμε αν ο κινητήρας κινείτε ή είναι σταματημένος. Επιστρέφει true όταν κινείτε, ενώ false όταν έχει σταματήσει.

- *getLimitAngle()* - Μας επιστρέφει τις μοίρες στις οποίες βρίσκετε αυτή τη στιγμή ο κινητήρας

- *getSpeed()* - Μας επιστρέφει την ταχύτητα του κινητήρα

- *isStalled()* - Μας επιστρέφει true όταν ο κινητήρας είναι stalled (δηλαδή για κάποιο λόγο δεν έχει τη δυνατότητα να κινηθεί – έχει κολλήσει).

- *ResetTachoCount()* - Θέτει ίση με το μηδέν την τιμή της μεταβλητής που υπολογίζει πόσες μοίρες έχει κινηθεί ο κινητήρας

- *setAcceleration(int acceleration)* - Μας επιτρέπει να ελέγχουμε το πόσο γρήγορα θα επιταχύνει το ρομπότ μας (μοίρες ανά δευτερόλεπτο στο τετράγωνο)

- *getAcceleration()* - Μας επιστρέφει την επιτάχυνση του κινητήρα τη στιγμή εκείνη

- *suspendRegulation()* - Σταματάει τον αυτόματο έλεγχο (regulation off) που γίνεται στα σερβομοτέρ για την ανίχνευση σφάλματος και τη διόρθωση του. Μπορούμε να τα χρησιμοποιούμε και σαν απλούς κινητήρες όταν αυτό εξυπηρετεί το πρόγραμμά μας

- *setPower(int aPower)* - Χρησιμοποιείτε για να θέσουμε μόνοι μας απευθείας την τιμή της ενέργειας που θα παρέχεται στους κινητήρες για την κίνηση. Δεν μπορεί να χρησιμοποιηθεί όταν έχουμε ανίχνευση σφάλματος (regulation on)

- *getPower()* - Μας επιστρέφει έναν ακέραιο αριθμό από το 1 έως το 100 που αντιπροσωπεύει την τιμή της ενέργειας που χρησιμοποιείτε για την κίνηση των μοτέρ

- *wait(long timeout)* - Μας δίνει τη δυνατότητα να έχουμε κάποια παύση πριν από την επόμενη ενέργεια. Η τιμή που θα δώσουμε στην μεταβλητή timeout

είναι ο χρόνος που θα διαρκέσει η παύση μας μετρημένος σε milliseconds. Αν θέσουμε την τιμή της timeout ίση με το μηδέν, τότε θα περιμένει για πάντα.

- *waitComplete()* - Με τη μέθοδο αυτή μπορούμε να κάνουμε το πρόγραμμά μας να περιμένει μέχρι να ολοκληρωθεί η προηγούμενη κίνηση και έπειτα να συνεχίσει παρακάτω.

Αν θέλουμε ο κινητήρας που είναι συνδεδεμένος στη θέση A να κινηθεί προς τα μπροστά θα πρέπει να γράψουμε: *Motor.A.forward().*

Επιπλέον καλό είναι να γνωρίζουμε ότι σε πολλές από τις μεθόδους που σχετίζονται με την κίνηση του ρομπότ μπορούμε να χρησιμοποιήσουμε ένα επιπλέον όρισμα τύπου Boolean, που συχνά ονομάζεται immediateReturn, για παράδειγμα *rotate(int angle,boolean immediateReturn).* Αν το όρισμα immediateReturn είναι true η μέθοδος κάνει return κατευθείαν ενώ αν είναι false κάνει return αφού ολοκληρωθεί η κίνηση, στο συγκεκριμένο παράδειγμα η περιστροφή.

3.3.2 ΑΙΣΘΗΤΗΡΕΣ

Το NXT 2.0 συνοδεύεται από ένα αισθητήρα υπερήχων, δύο αισθητήρες αφής και έναν αισθητήρα χρώματος. Το LeJOS μας παρέχει μεθόδους για όλους αυτούς τους αισθητήρες και επιπλέον για όσους μπορούμε να αγοράσουμε από την επίσημη σελίδα της Lego.

- ΑΙΣΘΗΤΗΡΑΣ ΥΠΕΡΗΧΩΝ

Ο αισθητήρας υπερήχων μας επιτρέπει την ανίχνευση αντικειμένων από απόσταση χωρίς να χρειάζεται να έρθουμε σε επαφή με αυτά. Για να χρησιμοποιήσουμε τον αισθητήρα υπερήχων πρώτα ορίζουμε σε ποια θύρα είναι συνδεδεμένος. Αν τον συνδέσουμε στην θύρα ένα θα γράψουμε:

UltrasonicSensor myUS = new UltrasonicSensor(SensorPort.S1);

Υπάρχουν δύο διαφορετικά είδη ανάλογα με το πόσο συχνά ο αισθητήρας μας στέλνει κύματα για να ελέγξει αν υπάρχει κάποιο εμπόδιο.

Το συνεχόμενο (που είναι και το default) που στέλνει ένα καινούριο κύμα αμέσως μόλις μπορέσει με αποτέλεσμα να είναι ενημερωμένο συνέχεια για το αν υπάρχουν εμπόδια μπροστά του. Μπορούμε να καλέσουμε τη μέθοδο *getDistance()* η οποία θα μας δώσει τα αποτελέσματα του τελευταίου κύματος που στάλθηκε. Σε περίπτωση που δεν υπάρχει εμπόδιο μας επιστρέφει την τιμή 255, ενώ αν υπάρχει μας επιστρέφει την απόσταση στην οποία βρίσκεται σε εκατοστά, με μέγιστη ακτίνα μέτρησης τα 170 εκατοστά περίπου.

Το ελεγχόμενο στο οποίο εμείς με τη μέθοδο *ping()* κάνουμε τον αισθητήρα να στείλει ένα κύμα και μέχρι και 12 τιμές αποθηκεύονται. Μπορούμε να διαβάσουμε τις τιμές αυτές καλώντας τη μέθοδο *getDistances(int [] distances).* Πριν καλέσουμε όμως τη

μέθοδο αυτή πρέπει να έχουμε δηλώσει έναν πίνακα τύπου int στον οποίο θα αποθηκευτούν οι τιμές που θα μας επιστρέψει η μέθοδος *getDistances(int [] distances).* Επιπλέον, πρέπει να εξασφαλίσουμε ότι υπάρχει μία καθυστέρηση περίπου 20 milliseconds ανάμεσα στο κάλεσμα της μεθόδου *ping()* και της μεθόδου *getDistances(int [] distances),* γιατί διαφορετικά υπάρχει περίπτωση να μας επιστρέψει σφάλμα ή να μη μας επιστρέψει καμία τιμή. Οι επιτρεπτές τιμές που μπορούμε να δεχτούμε ώ αποτελέσματα είναι από 0 ως 170 που αντιστοιχούν στην απόσταση του εμποδίου μετρημένη σε εκατοστά και την τιμή 255 που σημαίνει ότι δεν ανιχνεύτηκε κάποιο εμπόδιο.

Αν καλέσουμε τη μέθοδο *ping(),* αυτόματα θα σταματήσουμε να βρισκόμαστε στο πρώτο είδος που ο αισθητήρας στέλνει συνέχεια κύματα. Αν θέλουμε να πάμε ξανά στο συνεχόμενο πρέπει να καλέσουμε τη μέθοδο *continuous().* Εκτός από τις μεθόδους *ping(),continuous(), getDistance()* και *getDistances(int [] distances)* που ήδη αναφέραμε υπάρχουν και άλλες χρήσιμες μέθοδοι που αφορούν αυτό τον αισθητήρα:

- *reset()* - Μας επιτρέπει να κάνουμε reset τον αισθητήρα, δηλαδή να τον επαναφέρουμε στην αρχική του κατάσταση σε συνεχόμενου κύματος.

- *off()* - Μας δίνει τη δυνατότητα να κλείσουμε τον αισθητήρα. Όσο ο αισθητήρας είναι κλειστός δεν στέλνει κύματα ακόμη και αν ήταν σε κατάσταση συνεχόμενου κύματος. Για να λειτουργήσει ξανά ο αισθητήρας μπορούμε να καλέσουμε την μέθοδο reset(), ping() ή continuous().

- *setContiniousInterval()* - Μας επιτρέπει να ορίσουμε πόση θα είναι η χρονική περίοδος που θα περιμένει ο αισθητήρας μας από τη στιγμή που θα στείλει ένα κύμα μέχρι να στείλει το επόμενο. Σε περίπτωση που το νούμερο που βάλουμε είναι αποδεκτό, η μέθοδος θα μας επιστρέψει την τιμή μηδέν ενώ αν δεν είναι αποδεκτό κάποια άλλη τιμή.

- *getContiniousInterval()* - Μας επιστρέφει την τιμή του χρονικού διαστήματος που μεσολαβεί από τη στιγμή που θα στείλει ένα κύμα μέχρι να στείλει το επόμενο.

- *getMode()* - Μας επιστρέφει την τιμή 0 αν ο αισθητήρας είναι κλειστός, 1 αν είναι σε κατάσταση ελεγχόμενου κύματος, 2 αν είναι σε κατάσταση συνεχόμενου σήματος, 3 αν είναι σε κατάσταση capture (αυτή η κατάσταση μπορεί να υπάρξει μόνο άμα έχουμε δύο αισθητήρες υπερήχων) και -1 αν υπάρχει κάποιο σφάλμα.

- ΑΙΣΘΗΤΗΡΑΣ ΕΠΑΦΗΣ

Οι αισθητήρες επαφής που μας δίνουν τη δυνατότητα να αντιλαμβανόμαστε τυχόν εμπόδια που υπάρχουν μέσω της επαφής/σύγκρουσης με αυτά. Το LeJOS μας παρέχει την κλάση TouchFeatureDetector για να μας βοηθήσει να χειριστούμε τους

αισθητήρες αυτούς. Πριν χρησιμοποιήσουμε τον αισθητήρα αφής πρέπει πρώτα να ορίζουμε σε ποια θύρα είναι συνδεδεμένος. Αν τον συνδέσουμε στην θύρα δύο θα γράψουμε:

TouchSensor myTS = new TouchSensor(SensorPort.S2);

Η μόνη μέθοδος που αφορά τον αισθητήρα αφής είναι η *isPressed()* η οποία ελέγχει αν ο αισθητήρας είναι πατημένος και τότε μας επιστρέφει true. Σε περίπτωση που δεν είναι μας επιστρέφει false.

- ΑΙΣΘΗΤΗΡΑΣ ΧΡΩΜΑΤΟΣ

Ο αισθητήρας χρώματος έχει τρεις διαφορετικές λειτουργίες. Μπορεί να λειτουργήσει ως αισθητήρας χρώματος που ανιχνεύει χρώματα, ως αισθητήρας φωτός που αντιλαμβάνεται τα φωτεινά και τα σκοτεινά σημεία αλλά και ως λαμπάκι led. Για να μπορέσουμε να χρησιμοποιήσουμε τον αισθητήρα χρώματος πρέπει πρώτα να ορίσουμε σε ποια θύρα βρίσκεται συνδεδεμένος. Έστω ότι τον έχουμε συνδεδεμένο στη θύρα τρία θα πρέπει να γράψουμε:

ColorSensor myCS = new ColorSensor(SensorPort.S3);

- ΑΙΣΘΗΤΗΡΑΣ ΧΡΩΜΑΤΟΣ

Οι πιο σημαντικές μέθοδοι για τον αισθητήρα χρώματος είναι οι ακόλουθες:

- *getColor()* - Μας επιστρέφει μία τιμή τύπου int που αντιστοιχεί σε κάποιο χρώμα όπως φαίνεται στον πίνακα παρακάτω.
-

Πίνακας 1 "Αντιστοιχία χρωμάτων και αριθμών όπως τα αντιλαμβάνεται ο αισθητήρας χρωμάτων του ρομπότ NXT"

ΑΡΙΘΜΟΣ	ΧΡΩΜΑ
0	Κόκκινο
1	Πράσινο
2	Μπλε
3	Κίτρινο
6	Άσπρο
7	Μαύρο

- *getBlue()* - Μας επιστρέφει μία τιμή από 0-255 που αντιπροσωπεύει σε κλίμακα sRGB την ποσότητα του μπλε χρώματος που υπάρχει σε αυτό που πήρε ως είσοδο ο αισθητήρας..

- *getGreen()* - Το ίδιο με το προηγούμενο αλλά για το χρώμα πράσινο.

- *getRed()* - Το ίδιο για το κόκκινο χρώμα.

- ΑΙΣΘΗΤΗΡΑΣ ΦΩΤΟΣ

Οι κυριότερες μέθοδοι για τον αισθητήρα χρώματος ως αισθητήρα φωτός είναι οι εξής:

- *setLow(low)* - Μας δίνει τη δυνατότητα να ορίσουμε τη μεταβλητή low. Από μόνη της είναι 0 και αν μας την επιστρέψει η μέτρηση του αισθητήρα σημαίνει τελείως σκοτεινή περιοχή.

- *setHigh(high)* - Μας επιτρέπει να δώσουμε όποια τιμή θέλουμε στη μεταβλητή high. Η τιμή από μόνη της είναι 100 και όταν μας την επιστρέψει ο αισθητήρας σημαίνει τελείως φωτεινή περιοχή.

- *getLow()* - Μας επιστρέφει την τιμή της μεταβλητής low.

- *getHigh()* - Μας επιστρέφει την τιμή της μεταβλητής high.

- *getLightValue()* - Μας επιστέφει μία τιμή από 0-100 (εκτός αν έχουμε αλλάξει τις τιμές των μεταβλητών low και high οπότε η τιμή θα κυμαίνεται από low-high). Όσο πιο μεγάλη τιμή μας επιστρέψει η μέθοδος, τόσο πιο φωτεινό σημαίνει ότι ήταν δείγμα που μετρήσαμε.

- ΛΑΜΠΑΚΙ LED

Μερικές μέθοδοι για τον αισθητήρα χρώματος ως λαμπάκι Led φαίνονται παρακάτω:

- *setFloodlight(boolean floodlight*) - Μας δίνει τη δυνατότητα να ανάψουμε ή αν σβήσουμε το κόκκινο χρώμα στο λαμπάκι led. Αν η μεταβλητή floodlight είναι true τότε το λαμπάκι ανάβει, ενώ αν είναι false το λαμπάκι σβήνει.

- *setFloodlight(int color)* - Μας επιτρέπει να ορίσουμε τι χρώμα θέλουμε να ανάψει το λαμπάκι. Το LeJOS για να μας διευκολύνει αντί για τους αριθμούς int που αντιστοιχούν στο χρώμα που θέλουμε μας επιτρέπει να γράφουμε το χρώμα που θέλουμε. Για παράδειγμα αν θέλουμε πράσινο μπορούμε να γράψουμε *setFloodlight(Color.GREEN)* και το λαμπάκι led θα ανάψει πράσινο. Σε περίπτωση που το χρώμα που γράψουμε δεν υποστηρίζεται από τον αισθητήρα που χρησιμοποιούμε η μέθοδος μας επιστρέφει false.

- *getFloodlight()* - Μας επιστρέφει το χρώμα που έχει το λαμπάκι led αν είναι αναμμένο (πχ. αν έχει κόκκινο χρώμα θα μας επιστρέψει *Color.RED*), ή αν δεν είναι αναμμένο μας επιστρέφει *Color.NONE*.

- *isFloodlightOn()* - Ελέγχει αν το λαμπάκι είναι αναμμένο και μας επιστρέφει true είναι αναμμένο και false αν είναι κλειστό.

- ΑΙΣΘΗΤΗΡΕΣ ΓΙΑ INPUT ΚΑΙ OUTPUT

Εδώ θα δούμε όλες τις κλάσεις που σχετίζονται με input και output που είναι η οθόνη, τα κουμπιά που βρίσκονται πάνω στο NXT τούβλο και το μεγάφωνο που διαθέτει.

LCD ΟΘΟΝΗ

Η οθόνη του NXT μας επιτρέπει να αναπαραστήσουμε κείμενο και γραφικά. Όσο αφορά το κείμενο, έχει πλάτος 16 χαρακτήρες και ύψος 8. Είναι ρυθμισμένη να χρησιμοποιεί τις συντεταγμένες x και y. Το σημείο (0,0) βρίσκεται πάνω αριστερά. Το x μπορεί να πάρει τιμές από 0 ως 15 και το y από 0 ως 7. Οι μέθοδοι με τις οποίες μπορούμε να γράψουμε κάτι πάνω στην οθόνη είναι οι ακόλουθες:

- *drawString(String str, int x, int y)* - Με τη χρήση αυτής της μεθόδου μπορούμε να γράψουμε ένα κείμενο τύπου String που θα ξεκινάει στις συντεταγμένες της οθόνης (x,y)

- *drawInt(int i, int x, int y)* - Το ίδιο με την πρώτη μέθοδο αλλά χρησιμοποιείται για αριθμούς int.

- *drawInt(int i, int places, int x, int y)* - Είναι ίδια με την προηγούμενη μέθοδο με τη διαφορά ότι περιέχει ένα ακόμη όρισμα το places. Το όρισμα αυτό καθορίζει σε πόσες θέσεις θα γράψουμε. Έτσι αν έχουμε θέσει το places ίσο με 3 και βάλουμε ως τιμή i τον αριθμό 1234 στην οθόνη μας θα δούμε 123.

- *clear()* – Σβήνει ότι έχουμε γράψει μέχρι τότε στην οθόνη μας.

- *setAutoRefresh(int i)* - Η οθόνη του ρομπότ μας κάνει από μόνη της ανανέωση περιεχομένου. Μπορούμε να αλλάξουμε την τιμή αυτή για να ανανεώνεται όποτε εμείς θέλουμε ή ακόμη και να θέσουμε το i ίσο με το 0 για αν μην κάνει αυτόματη ανανέωση ποτέ.

- *refresh()* - Αν έχουμε σταματήσει την αυτόματη ανανέωση καλώντας αυτή τη μέθοδο κάνουμε την ίδια δουλειά χειροκίνητα όποτε χρειάζεται.

Επίσης, καλό είναι να γνωρίζουμε ότι μπορούμε να γράψουμε κάτι στην οθόνη χρησιμοποιώντας την κλασική μέθοδο της Java *System.out.println(String str).* Τα κείμενό μας θα εμφανιστεί στην τελευταία σειρά της οθόνης και αν δεν χωράει εκεί θα ξεκινήσει να γράφεται από πιο πάνω.

Όσο αφορά την αναπαράσταση γραφικών, η οθόνη μας έχει 100 pixels πλάτος και 64 pixels ύψος. Χρησιμοποιούνται πάλι οι συντεταγμένες (x, y) και το σημείο (0, 0) βρίσκεται πάνω αριστερά. Το x παίρνει τιμές από 0 ως 99 ενώ το y από 0 ως 63. Για να σχεδιάσουμε γραφικά μπορούμε να χρησιμοποιήσουμε την κλάση Graphics με την οποία μας δίνεται η δυνατότητα να σχεδιάσουμε γραμμές, κύκλους, τετράγωνα,

ορθογώνια, γωνίες ή ακόμη και να ζωγραφίσουμε κάτι θέτοντας ποια pixels θα είναι μαύρα.

ΚΟΥΜΠΙΑ NXT

Το NXT τούβλο έχει τέσσερα κουμπιά στα οποία μπορούμε να αναφερθούμε με τον ακόλουθο τρόπο:

- *Button.ENTER* - το πορτοκαλί τετράγωνο
- *Button.ESCAPE* - το σκούρο γκρι ορθογώνιο
- *Button.LEFT* - το γκρι τρίγωνο που δείχνει προς τα αριστερά
- *Button.RIGHT* - το γκρι τρίγωνο που δείχνει προς τα δεξιά

Οι μέθοδοι που σχετίζονται με τα κουμπιά είναι:

- *isPressed()* - Μας επιστρέφει true όταν το κουμπί είναι πατημένο.
- *waitForPressAndRelease()* - Περιμένει μέχρι ένα συγκεκριμένο κουμπί να πατηθεί και να απελευθερωθεί πάλι.
- *waitForPress()* - Περιμένει να πατηθεί κάποιο από τα κουμπιά και μας επιστρέφει τον κωδικό του κουμπιού που πατήθηκε. (1 = enter, 2 = left, 4 = right, 8 = escape)
- addButtonListener (ButtonListener aListener) - Μας επιτρέπει να βάλουμε ένα listener
- *readButtons()* - Επιστρέφει μία τιμή int η οποία αποτελεί το άθροισμα των κωδικών που αντιστοιχούν στα κουμπιά που είναι πατημένα. Και αφού ισχύει: 1 = enter, 2 = left, 4 = right, 8 = escape αν μας επιστρέψει για παράδειγμα 9 ξέρουμε ότι είναι πατημένα το κουμπί enter και το κουμπί escape.
- ΗΧΟΣ

Η κλάση Sound είναι αυτή στην οποία περιέχονται όλες οι μέθοδοι που σχετίζονται με το μεγάφωνο που υπάρχει στο NXT τούβλο. Στις πιο γνωστές ανήκει η μέθοδος:

- *playTone(int aFrequency, int aDuration)* - Παίζει μία νότα η οποία αντιστοιχεί στην τιμή του ορίσματος aFrequency για συγκεκριμένη χρονική διάρκεια την οποία ορίζουμε με την μεταβλητή aDuration.

Υπάρχουν δύο τρόποι αναπαραγωγής ήχων συστημάτων. Ο πρώτος είναι χρησιμοποιώντας τη μέθοδο *systemSound(boolean aQueued, int aCode)* η οποία έχει δύο ορίσματα. Το πρώτο όρισμα, το aQueued χρησιμοποιείται μόνο από το RXC τούβλο, ενώ στην περίπτωση του NXT τούβλου, το οποίο και χρησιμοποιούμε, απλά

αγνοείται. Το δεύτερο όρισμα, το aCode το οποίο παίρνει τιμές από 0 ως 4. Κάθε κωδικός αντιστοιχεί σε έναν συγκεκριμένο ήχο, όπως φαίνεται και στον παρακάτω πίνακα.

Πίνακας 2 "Αντιστοιχία αριθμών με ήχους συστήματος σύμφωνα με την κλάση Sound του LeJOS"

ΑΡΙΘΜΟΣ	ΗΧΟΣ
0	Μικρό beep
1	Διπλό beep
2	Φθίνουσα σειρά από beep
3	Αύξουσα σειρά από beep
4	Συνεχόμενο χαμηλό buzz

Ο δεύτερος είναι με τη χρήση ξεχωριστών μεθόδων για κάθε ήχο συστήματος, με σαφή ονόματα για να ξέρουμε βρίσκουμε αμέσως τον ήχο που θέλουμε να χρησιμοποιήσουμε χωρίς να χρειάζεται να θυμόμαστε απέξω τον πίνακα με τις αντιστοιχίες των κωδικών:

- *beep()* - Μικρό beep
- *twoBeeps()* - Διπλό beep
- *beepSequence()* - Φθίνουσα σειρά από beep
- *beepSequenceUp()* - Αύξουσα σειρά από beep
- *buzz()* - Συνεχόμενο buzz

Επιπλέον, οι ακόλουθες μέθοδοι είναι διαθέσιμες:

- *playSample(File aWAVfile)* - Χρησιμοποιείται για την αναπαραγωγή ενός αρχείου τύπου .wav το οποίο πρέπει όμως για να μπορεί να το διαβάσει θα πρέπει να είναι 8-bit, mono, pcm και 8KHz. Μας επιστρέφει μία τιμή τύπου int που αντιπροσωπεύει το χρονικό διάστημα για το οποίο ακούστηκε ο ήχος. Αν η τιμή είναι μικρότερη ή ίση του μηδενός τότε υπάρχει κάποιο σφάλμα.
- *playSample(File aWAVfile, int volume)* - Είναι ίδια με την παραπάνω μέθοδο με τη μόνη διαφορά ότι σε αυτή υπάρχει και ένα δεύτερο όρισμα, το volume με το οποίο μπορούμε να ρυθμίσουμε την ένταση του ήχου.
- *playNote(int[] inst,int freq, int len)* - Χρησιμοποιούμε αυτή τη μέθοδο για την αναπαραγωγή μίας μουσικής νότας. Υπάρχουν ήδη ορισμένα μερικά μουσικά όργανα, όπως το πιάνο, το ξυλόφωνο και το φλάουτο αλλά μπορούμε να φτιάξουμε και δικά μας. Το πρώτο όρισμα είναι ένας πίνακας που περιέχει πιο ειδικά στοιχεία όπως η δύναμη της νότας και αν είναι ύφεση ή δίεση, το

δεύτερο όρισμα είναι η συχνότητα της νότας και το τρίτο όρισμα καθορίζει τη χρονική διάρκεια για την οποία θα ακουστεί η νότα.

- *setVolume()* - Μας δίνει τη δυνατότητα να ρυθμίσουμε την ένταση στην οποία θα ακουστεί ο ήχος από 0- αθόρυβο μέχρι 100-όσο πιο δυνατά γίνεται.
- *getVolume()* - Μας επιστρέφει την τωρινή τιμή της έντασης από 0 έως 100.
- *getTime()* - Μας επιστρέφει πόση ώρα (σε milliseconds) μένει ακόμη μέχρι να τελειώσει ο ήχος που ακούμε τώρα.

3.3.3 ΜΠΑΤΑΡΙΑ

Υπάρχουν δύο μέθοδοι οι οποίες μας δίνουν πληροφορίες σχετικά με την ενέργεια της μπαταρίας του ρομπότ μας:

- *getVoltageMilliVolt()* - Μας επιστρέφει την ένταση της μπαταρίας σε milliVolt.
- *getVoltage()* - Μας επιστρέφει την ένταση της μπαταρίας σε Volt.

3.4 ΒΟΗΘΗΤΙΚΑ ΠΡΟΓΡΑΜΜΑΤΑ LEJOS

Τα βοηθητικά προγράμματα του LeJOS, είναι εφαρμογές γραμμένες σε Java που υλοποιούν διάφορα συνηθισμένα προβλήματα που καλείται να λύσει ο προγραμματιστής. Μπορούμε να χρησιμοποιήσουμε απευθείας όλες τις μεθόδους των εφαρμογών που ακολουθούν σε κάποιο πρόγραμμά μας. Τα βοηθητικά προγράμματα αποτελούν ένα βασικό πλεονέκτημα για τον χρήστη αφού με τη χρήση τους εξοικονομούμε κόπο και χρόνο.

3.4.1 ΚΙΝΗΣΗ ΡΟΜΠΟΤ

Ένας από τους πιο συνηθισμένους τύπους ρομπότ είναι αυτά που έχουν δύο τροχούς ανεξάρτητους μεταξύ τους για να κινούνται. Στη βιβλιοθήκη του LeJOS υπάρχουν βοηθητικές μέθοδοι για να χειριστούμε με ευκολία τέτοιου είδους ρομπότ. Σε πρώτο επίπεδο υπάρχει η κλάση DifferentialPilot που μας παρέχει μεθόδους για να μπορούμε να κάνουμε επιτόπου αναστροφή, να κινούμαστε σε ευθεία γραμμή ή με συγκεκριμένη κλίση. Στο επόμενο επίπεδο υπάρχει η Navigator η οποία χρησιμοποιεί την DifferentialPilot για να μας βοηθήσει να κινήσουμε το ρομπότ μας μέσα σε ένα περίπλοκο προσχεδιασμένο μονοπάτι. Για να μπορέσουμε να περιηγηθούμε με επιτυχία στο χώρο πρέπει να γνωρίζουμε την ακριβή του τοποθεσία και την κατεύθυνση προς την οποία βρίσκεται στραμμένο πριν ξεκινήσουμε. Το ρομπότ ενημερώνει κάθε τόσο αυτές τις πληροφορίες χρησιμοποιώντας την κλάση OdometryPoseProvider. Οι σχέσεις μεταξύ των κλάσεων που προαναφέρθηκαν φαίνονται συνοπτικά στον παρακάτω πίνακα.

Πίνακας 3 "Σχέσεις μεταξύ κλάσεων που σχετίζονται με την κίνηση του ρομπότ μας"

CLASS	USER
Navigator	DifferentialPilot OdometryPoseProvider
OdometryPoseProvider	DifferentialPilot
DifferentialPilot	RegulatedMotor

Πιο αναλυτικά, η ροή ελέγχου πηγαίνει από πάνω προς τα κάτω, δηλαδή η κλάση Navigator ελέγχει την DifferentialPilot που ελέγχει τους κινητήρες, ενώ η ροή των πληροφοριών πηγαίνει από κάτω προς τα πάνω, δηλαδή η κλάση DifferentialPilot χρησιμοποιεί πληροφορίες από τους κινητήρες για τους ελέγξει. Η κλάση OdometryPoseProvider με τις πληροφορίες που παίρνει από την DifferentialPilot για να έχει συνέχεια τις πληροφορίες για τις συντεταγμένες του ρομπότ και την κατεύθυνσή του ενημερωμένες με τις πιο πρόσφατες τιμές. Τέλος, η κλάση Navigator χρησιμοποιεί τα δεδομένα αυτά για τη θέση του ρομπότ για να υπολογίζει την απόσταση και την κατεύθυνση του προορισμού που έχουμε προεπιλέξει.

- DifferentialPilot

Η κλάση αυτή, όπως έχουμε ήδη αναφέρει χρησιμοποιείται για τον έλεγχο ενός κινούμενο σε τροχούς ρομπότ με 2 τροχούς κίνησης ανεξάρτητους μεταξύ τους, δηλαδή των καθένα με δικό του κινητήρα. Μπορεί να κάνει το ρομπότ μας να στρίψει ελέγχοντας την ταχύτητα και την κατεύθυνση περιστροφής των κινητήρων του.

Ένα αντικείμενο αυτής της κλάσης για να μπορέσει να λειτουργήσει σωστά χρειάζεται κάποιες πληροφορίες, όπως σε ποιές θύρες είναι συνδεδεμένοι οι κινητήρες, αν η περιστροφή των κινητήρων προς τα μπροστά κάνει το ρομπότ να κινείται προς τα μπροστά ή προς τα πίσω, δηλαδή αντίστροφα, την διάμετρο από τις ρόδες και την απόσταση που έχουν οι δύο ρόδες μεταξύ τους.

Η DifferentialPilot χρησιμοποιεί την διάμετρο των τροχών για να υπολογίζει την απόσταση που διένυσε το ρομπότ και την απόσταση μεταξύ των τροχών για να ελέγχει το πόσο έχει περιστραφεί. Προφανώς και οι δύο αυτές παράμετροι πρέπει να είναι δηλωμένες στην ίδια μονάδα μέτρησης, οποιαδήποτε προτιμούμε εμείς, αν και πιο συνηθισμένα είναι τα εκατοστά και οι ίντσες. Αν θέσουμε σωστά τις παραμέτρους που περιγράψαμε παραπάνω τα λάθη στην απόσταση που διανύει και στην περιστροφή γύρω από τον εαυτό του το ρομπότ μας είναι περίπου 2%.

Constructors ονομάζονται οι ειδικές υπορουτίνες, με όνομα ίδιο με της κλάσης, που καλούνται για τη δημιουργία αντικειμένων. Προετοιμάζουν το νέο αντικείμενο για χρήση, ορίζοντας τις μεταβλητές του, ενώ συνήθως δεν επιστρέφουν τίποτα. Υπάρχουν τρεις διαφορετικοί constructors για την DifferentialPilot:

- DifferentialPilot(float wheelDiameter, float trackWidth, Motor leftMotor, Motor rightMotor)

Το πρώτο όρισμα είναι η διάμετρος των τροχών που πρέπει να είναι μετρημένη στην ίδια μονάδα μέτρησης με το δεύτερο όρισμα που είναι η απόσταση μεταξύ των τροχών. Το τρίτο και τέταρτο όρισμα δηλώνουν σε ποιες θύρες είναι συνδεδεμένοι ο αριστερός και ο δεξής κινητήρας αντίστοιχα.

- DifferentialPilot(float wheelDiameter, float trackWidth, Motor leftMotor, Motor rightMotor, boolean reverse)

Τα τέσσερα πρώτα ορίσματα είναι ίδια με τα παραπάνω αλλά υπάρχει και ένα ακόμη που είναι false αν όταν κινούνται οι κινητήρες προς τα μπροστά κινείται και το ρομπότ μας μπροστά ή true όταν συμβαίνει το ανάποδο, δηλαδή οι κινητήρες να κινούνται προς τα μπροστά το ρομπότ μας κινείται προς τα πίσω.

- DifferentialPilot(float leftWheelDiameter, float rightWheelDiameter, float trackWidth, Motor leftMotor, Motor rightMotor, boolean reverse)

Χρησιμοποιείται σε περίπτωση που οι ρόδες που χρησιμοποιήσαμε είναι διαφορετικές. Μας επιτρέπει να ορίσουμε διαφορετική διάμετρο για κάθε τροχό.

- Κίνηση δίτροχου ρομπότ

Αφού πρώτα δημιουργήσουμε ένα αντικείμενο της κλάσης DifferentialPilot προσέχοντας να έχουμε σωστές τιμές στα ορίσματα μπορούμε να χρησιμοποιήσουμε τις παρακάτω μεθόδους για να κάνουμε το ρομπότ μας να κινηθεί σε ευθεία γραμμή:

- *setTravelSpeed(double travelSpeed)* - Μας δίνει τη δυνατότητα να ορίσουμε την ταχύτητα με την οποία θα κινείται το ρομπότ μας.

- *forward()* - Κάνει το ρομπότ μας να κινηθεί προς τα μπροστά.

- *backward()* - Κάνει το ρομπότ μας να κινηθεί προς τα πίσω.

- *stop()* - Σταματάει την κίνηση του ρομπότ μας.

- *travel(double distance)* - Ορίζει την απόσταση την οποία θέλουμε να διανύσει το ρομπότ μας. Αν βάλουμε αρνητική τιμή το ρομπότ μας θα κινηθεί προς τα πίσω.

- *getMovement().getDistanceTraveled()* - Μας επιστρέφει την απόσταση που έχει διανύσει το ρομπότ μας από την τελευταία φορά που καλέσαμε την μέθοδο *resetTachoCount()* η οποία μηδενίζει την τιμή της μεταβλητής που υπολογίζει πόσες μοίρες έχει κινηθεί ο κινητήρας.

Οι παρακάτω μέθοδοι μας δίνουν τη δυνατότητα να περιστρέφουμε επιτόπου το ρομπότ μας:

- *rotateLeft()* - Δίνει την εντολή στο ρομπότ μας να κάνει επιτόπου αναστροφή προς τα αριστερά. Συνεχίζει να περιστρέφεται μέχρι να του πούμε εμείς να σταματήσει.

- *rotateRight()* - Δίνει την εντολή στο ρομπότ μας να κάνει επιτόπου αναστροφή προς τα δεξιά μέχρι να το σταματήσουμε.

- *rotate(double angle)* - Αν θέλουμε να περιστρέψουμε το ρομπότ μας συγκεκριμένη γωνία χρησιμοποιούμε αυτή τη μέθοδο και δίνουμε ως όρισμα τις μοίρες που θέλουμε να περιστραφεί. Αν βάλουμε θετική τιμή το ρομπότ μας θα στρίψει προς τα αριστερά, ενώ αν βάλουμε αρνητική τιμή θα στρίψει προς τα δεξιά.

Οι μέθοδοι που ακολουθούν μας επιτρέπουν να κινούμε το ρομπότ μας έτσι ώστε να διαγράφει ένα κύκλο:

- *steer(double turnRate)* - Η μεταβλητή turnRate καθορίζει την ακτίνα του κύκλου που θα διαγράψει με την πορεία του το ρομπότ μας. Αν η τιμή που θα δώσουμε είναι θετική τότε το κέντρο του κύκλου θα βρίσκεται στην αριστερή μεριά του ρομπότ, δηλαδή το ρομπότ μας θα ξεκινήσει να στρίβει προς την αριστερή πλευρά, ενώ αν είναι αρνητική θα στρίψει προς τα δεξιά. Η απόλυτη τιμή της μεταβλητής μπορεί να είναι από 0 έως 200. Με τιμή 0 το ρομπότ μας ταξιδεύει σε ευθεία γραμμή, με τιμή 200 κάνει επιτόπου αναστροφή, ενώ με οποιαδήποτε ενδιάμεση τιμή κινείται σε κυκλική πορεία. Η κίνηση συνεχίζεται μέχρι να τη σταματήσουμε εμείς.

- *steer(double turnRate, int angle)* - Η πρώτη μεταβλητή είναι αυτή που περιγράψαμε παραπάνω ενώ η μεταβλητή angle καθορίζει την γωνία στην οποία θέλουμε να σταματήσει η περιστροφή. Αν βάλουμε αρνητική τιμή στην turnRate το ρομπότ μας θα περιστραφεί προς τη φορά που θέτει η μεταβλητή turnRate αλλά πηγαίνοντας προς τα πίσω.

- *arcBackward(double radius)* - Το ρομπότ μας ξεκινάει να περιστρέφεται προς τα πίσω. Η μεταβλητή radius καθορίζει το πόσο μεγάλος θα είναι ο κύκλος και αν είναι θετική το κέντρο του κύκλου θα βρίσκεται από την αριστερή το πλευρά. Η κίνηση σταματάει όταν εμείς τη διακόψουμε.

- *arcForward(double radius)* - Το ίδιο με το πάνω αλλά το ρομπότ μας ξεκινάει να περιστρέφεται προς τα μπροστά.

- *arc(double radius, double angle)* - Η μεταβλητή radius περιγράφεται παραπάνω ενώ η μεταβλητή angle μας βοηθάει να ορίσουμε σε ποια γωνία θέλουμε να σταματήσουμε την περιστροφή.

Σε περίπτωση που θέλουμε να έχουμε περιστροφή αλλά με μεγάλη ακρίβεια πρέπει να μειώσουμε την ταχύτητα περιστροφής χρησιμοποιώντας τη μέθοδο *setRotateSpeed(double speed).* Όσο μικρότερη η ταχύτητα τόσο μικρότερο και το σφάλμα στη περιστροφή από την τάση των αντικειμένων να διατηρούν σταθερή την κινητική τους κατάσταση.

- OdometryPoseProvider

Η κλάση OdometryPoseProvider περιέχει πληροφορίες για την θέση που βρίσκεται το ρομπότ μας αλλά και για την κατεύθυνση στην οποία είναι στραμμένο. Για να μπορεί να συμβεί αυτό, η κλάση αυτή θα πρέπει να ενημερώνεται για κάθε κίνηση που γίνεται από την κλάση DifferentialPilot. Αυτό επιτυγχάνεται με τη χρήση ενός listener της κλάσης OdometryPoseProvider που προσθέτουμε στην κλάση DifferentialPilot με την μέθοδο *addListener().* Ο listener καλεί αυτόματα την μέθοδο *moveStarted()* κάθε φορά που το ρομπότ μας ξεκινάει να κινείται και την μέθοδο *moveStopped()* κάθε φορά που το ρομπότ μας είναι ακίνητο. Για να μένει ενημερωμένη η OdometryPoseProvider καλεί την μέθοδο *getMovement()* στην DifferentialPilot για να πάρει τις απαραίτητες πληροφορίες.

- Compass Pilot

Η κλάση CompassPilot είναι αποτελεί επέκταση της DifferentialPilot. Υλοποιεί τις ίδιες μεθόδους, αλλά χρησιμοποιεί τον αισθητήρα πυξίδα για να ελέγχει την κατεύθυνση προς την οποία βρίσκεται στραμμένο το ρομπότ μας, ώστε είναι σίγουρο ότι είναι ίδια με αυτή που μας υποδεικνύει η κλάση DifferentialPilot. Ο αισθητήρας πυξίδα όπως ήδη έχουμε αναφέρει δεν συμπεριλαμβάνεται στο κουτί του Lego Mindstorm, αλλά υπάρχει διαθέσιμος και μπορεί να αγοραστεί από την επίσημη σελίδα της Lego.

Οι διαθέσιμοι constructors είναι ίδιοι με αυτούς της DifferentialPilot με μόνη διαφορά το ότι υπάρχει μία επιπλέον παράμετρος που αφορά τον αισθητήρα πυξίδα ορίζοντας σε ποια θύρα βρίσκεται συνδεδεμένος:

- CompassPilot(SensorPort compassPort, float wheelDiameter, float trackWidth, Motor leftMotor, Motor rightMotor)

- CompassPilot(SensorPort compassPort, float wheelDiameter, float trackWidth, Motor leftMotor, Motor rightMotor, boolean reverse)

Επιπλέον, υπάρχουν διαθέσιμες και οι παρακάτω μέθοδοι στην κλάση CompassPilot:

- *calibrate()* - Κάνει calibrate (βαθμονομεί) τον αισθητήρα πυξίδα γυρνώντας αργά το ρομπότ μας μία φορά επιτόπου (δηλαδή 360μοίρες).

- *setHeading(int angle)* - Μας επιτρέπει να θέσουμε την κατεύθυνση στην οποία θα είναι στραμμένο το ρομπότ μας. Όσο αυξάνουμε την τιμή της μεταβλητής angle, τόσο πιο αριστερά στρίβει το ρομπότ μας. Η τιμή 0 σημαίνει ότι το ρομπότ μας έχει την κατεύθυνση προς την μεριά που βρίσκονται οι θετικοί αριθμοί στον άξονα x του καρτεσιανού συστήματος και η τιμή 90 ότι έχει την κατεύθυνση προς την μεριά που βρίσκονται οι θετικοί αριθμοί στον άξονα y.

- *getHeading()* - Μας επιστρέφει την επιθυμητή τιμή της κατεύθυνσης του ρομπότ μας.

- *getAngle()* - Μας επιστρέφει την τιμή της κατεύθυνσης του ρομπότ μας.

- OdometryPoseProvider

Η πιο βασική ευθύνη που έχει αυτή η κλάση είναι ότι πρέπει να διατηρεί πληροφορίες για την τωρινή θέση του ρομπότ μας στο χώρο αλλά και την κατεύθυνση προς την οποία είναι στραμμένο.

Υπάρχει μόνο ένας constructor για αυτή την κλάση:

- *OdometryPoseProvider(MoveProvider mp)* - Κατασκευάζει ένα αντικείμενο της κλάσης αυτής που λειτουργεί ως listener.

Σχετικές μέθοδοι με τη κλάση αυτή που είναι αρκετά χρήσιμες είναι οι:

- *setPose(Pose aPose)* - Τη χρησιμοποιούμε για να θέσουμε τιμή στο aPose (τωρινή θέση του ρομπότ μας).

- *Pose getPose()* - Η μέθοδος αυτή καλείτε όσο το ρομπότ μας κινείται και εγγυάται ότι το Pose έχει τη σωστή τιμή.

- Navigator

Η κλάση Navigator χρησιμοποιεί και την Pilot για να μπορεί να ελέγχει την κίνηση του ρομπότ και την PoseProvider για να γνωρίζει την τωρινή θέση του. Χρησιμοποιούμε αυτή την κλάση για να κατευθύνουμε το ρομπότ μας σε ένα συγκεκριμένο μονοπάτι, από σημείο σε σημείο. Κάθε τέτοιο σημείο είναι ένα instance της κλάσης Waypoint, ενώ όλο το μονοπάτι που σχηματίζεται από τα σημεία είναι ένα instance της κλάσης Path. Το μονοπάτι συμπεριφέρεται όπως ακριβώς και μία λίστα FIFO (first in, first out). Μόλις το ρομπότ μας φτάσει ένα σημείο Waypoint, το αφαιρεί από το μονοπάτι και συνεχίζει πηγαίνοντας προς το επόμενο. Επιπλέον έχουμε τη δυνατότητα να προσθέτουμε στο τέλος της λίστας όσα ακόμη σημεία θέλουμε οποιαδήποτε στιγμή.

Η κλάση Navigator έχει δύο διαφορετικούς constructors:

- *Navigator(MoveController aPilot)* - Αν χρησιμοποιήσουμε αυτό τον constructor εμείς χρειάζεται να φτιάξουμε μόνο ένα pilot και αυτό δημιουργεί το δικό του OdometryPoseProvider.

- *Navigator(MoveController pilot, PoseProvider poseProvider)* - Το χρησιμοποιούμε όταν θέλουμε να κάνουμε χρήση κάποιας δικής μας PoseProvider κλάσης.

Μέθοδοι για την κλάση Navigator:

- *setPath(Path aPath)* - Με τη μέθοδο αυτή δημιουργούμε ένα καινούριο μονοπάτι στο οποίο μπορούμε να προσθέσουμε νέα σημεία είτε με τη χρήση των μεθόδων *addWaypoint(Waypoint aWaypoint)* και *addWaypoint(float x, float y)* που απλά προσθέτουν στο τέλος του μονοπατιού ένα ακόμη σημείο είτε με την *addWaypoint(float x, float y , heading)* που προσθέτει ακόμη ένα σημείο στο τέλος του μονοπατιού αλλά ορίζει και την επιθυμητή κατεύθυνση του ρομπότ μας (δηλαδή μόλις φτάσει στο συγκεκριμένο σημείο το ρομπότ μας περιστρέφεται μέχρι να αποκτήσει την κατεύθυνση που του ορίσαμε).

- *followRoute(Path aRoute)* - Το ρομπότ μας ξεκινάει να κινείται ακολουθώντας το μονοπάτι aRoute.

- *stop()* - Κάνει το ρομπότ μας να σταματήσει αλλά θυμάται ποιο μονοπάτι ακολουθούσε και πιο είναι το επόμενο σημείο που πρέπει να πάει.

- *followRoute()* - Το ρομπότ μας ξεκινάει να κινείται ακολουθώντας ένα ήδη γνωστό μονοπάτι. Χρησιμοποιείται για να ξεκινήσουμε ξανά την κίνηση όταν για κάποιο λόγο είχαμε καλέσει προηγουμένως τη μέθοδο *stop().*

- *goTo(WayPoint destination)* - Προσθέτει ένα νέο σημείο στο τέλος του μονοπατιού και ξεκινάει την κίνηση του ρομπότ μας στο μονοπάτι.

- *goTo(double x, double y)* – Λειτουργεί όπως και η παραπάνω μέθοδος.

- *goTo(double x, double y , double heading)* - Χρησιμοποιείται ακριβώς όπως η προηγούμενη μέθοδος με τη μόνη διαφορά ότι μόλις φτάσει στο συγκεκριμένο σημείο το ρομπότ μας περιστρέφεται μέχρι να αποκτήσει την κατεύθυνση που του ορίσαμε στο όρισμα heading.

- *isMoving()* - Μας επιστρέφει true αν το ρομπότ μας κινείται κατευθυνόμενο προς ένα σημείο.

- *pathCompleted()* - Μας επιστρέφει true αν το ρομπότ μας έχει φτάσει στο τελικό σημείο του μονοπατιού.

- *rotateTo(double direction)* - Κάνει το ρομπότ μας να περιστραφεί προς την τιμή του ορίσματος direction. Όπως ήδη αναφέραμε, οι περιστροφές γίνονται με βάση το καρτεσιανό σύστημα. Η τιμή 0 σημαίνει ότι το ρομπότ μας έχει την κατεύθυνση προς την μεριά που βρίσκονται οι θετικοί αριθμοί στον άξονα x του καρτεσιανού συστήματος και η τιμή 90 ότι έχει την κατεύθυνση προς την μεριά που βρίσκονται οι θετικοί αριθμοί στον άξονα y.

- *waitForStop()* - Περιμένει μέχρι το ρομπότ μας να σταματήσει να κινείται και μετά κάνει return οπότε το πρόγραμμά μας μπορεί να συνεχίσει παρακάτω εκτελώντας την επόμενη γραμμή κώδικα.

- *singleStep(boolean yes)* - Καλούμε αυτή τη μέθοδο με όρισμα true αν θέλουμε το ρομπότ μας να σταματάει σε κάθε σημείο. Για να πάει στο επόμενο σημείο καλούμε τη μέθοδο followRoute().

- ShortestPathFinder

Ας υποθέσουμε ότι το ρομπότ μας βρίσκεται σε μία γνωστή τοποθεσία και στόχος μας είναι να φτάσουμε σε ένα προορισμό. Στο δρόμο προς τον προορισμό μας υπάρχουν εμπόδια τα οποία βρίσκονται τοποθετημένα σε γνωστά σημεία. Τα σημεία αυτά αναπαρίστανται πάνω σε ένα χάρτη. Πιο συγκεκριμένα ο χάρτης αυτός αποτελείται από ίσιες γραμμές που μας δείχνουν όλα τα αντικείμενα-εμπόδια που υπάρχουν στο χώρο στον οποίο αναφέρεται ο χάρτης καθώς και τα όρια του. Ο constructor της κλάσης ShortestPathFinder δέχεται ως όρισμα ένα χάρτη. Με τη χρήση της κλάσης αυτής μπορούμε να βρούμε την πιο σύντομη διαδρομή για τον προορισμό μας (η διαδρομή είναι ένα μονοπάτι, δηλαδή μία λίστα από σημεία).

Ο constructor της κλάσης ShortestPathFinder είναι:

- *ShortestPathFinder(LineMap map)* - Δέχεται ως όρισμα ένα χάρτη.

Μέθοδοι της κλάσης αυτής:

- *Path findRoute(Pose start, WayPoint finish*) - Τη χρησιμοποιούμε όταν θέλουμε να βρούμε το πιο σύντομο μονοπάτι για έναν προορισμό. Το όρισμα start περιέχει πληροφορίες για την αρχική θέση και την κατεύθυνση του ρομπότ μας και το όρισμα finish για την τελική θέση που πρέπει να φτάσει το ρομπότ μας. Ο χάρτης που χρησιμοποιείται είναι αυτός που ορίστηκε στον constructor της κλάσης

- *Path findRoute(Pose start, WayPoint finish, LineMap theMap)* - Είναι ίδια με την προηγούμενη μέθοδο με τη μόνη διαφορά ότι περιέχει και ένα τρίτο όρισμα που μας επιτρέπει να χρησιμοποιήσουμε διαφορετικό χάρτη από αυτόν που ορίζει ο constructor.

Όσο αφορά στην εύρεση του πιο σύντομου μονοπατιού, επειδή το ρομπότ μας έχει κάποιες διαστάσεις και πολλές φορές, αν και θεωρητικά υπάρχει μονοπάτι το ρομπότ μπορεί να μην χωράει να περάσει από κάποιο σημείο. Για να αποφύγουμε τέτοιου είδους προβλήματα χρησιμοποιούμε τη μέθοδο *lengthenLines(float delta).*

3.4.2 ΑΝΙΧΝΕΥΣΗ ΑΝΤΙΚΕΙΜΕΝΩΝ

Με αυτή τη λειτουργία ένα ρομπότ έχει τη δυνατότητα να εντοπίζει αντικείμενα και να αντιδράει με κάποιο τρόπο που εμείς ορίζουμε, για παράδειγμα να τα αποφεύγει. Για την ανίχνευση των αντικειμένων χρησιμοποιούμε συνήθως έναν αισθητήρα, όπως ο αισθητήρας επαφής ή ο αισθητήρας υπερήχων. Για να χρησιμοποιήσουμε κάποιον από τους δύο δημιουργούμε ένα FeatureDetector που μπορεί να είναι είτε RangeFeatureDetector για ανίχνευση αντικειμένων από μακριά, είτε TouchFeatureDetector για ανίχνευση αντικειμένων με επαφή. Φυσικά σε ένα πρόγραμμα μπορούμε να έχουμε συνδυασμό και των δύο τρόπων ανίχνευσης αντικειμένων ή και πολλούς αισθητήρες ίδιου τύπου (πχ αφής) σε διάφορα σημεία. Για το συνδυασμό διαφόρων FeatureDetectors μπορούμε να χρησιμοποιήσουμε την κλάση FusorDetector.

Ένα πολύ σημαντικό πλεονέκτημα των κλάσεων FeatureDetector είναι ότι μπορούν αυτόματα να ενημερώνουν άλλες κλάσεις όταν ανιχνευθεί κάποιο αντικείμενο, με τη χρήση ενός listener. Αφού έχουμε αρχικοποιήσει το FeatureDetector μπορούμε να προσθέσουμε σε αυτό ένα FeatureListener:

FeatureDetector.addListener(FeatureListener)

Έπειτα, μέσω της μεθόδου *FetureListener.featureDetected()* θα ενημερώνεται ο FeatureListener κάθε φορά που ανιχνεύεται ένα αντικείμενο.

- RangeFeatureDetector

Το LeJOS μας παρέχει την κλάση RangeFeatureDetector για να κάνει πιο εύκολη τη χρήση του αισθητήρα υπερήχων. Η κλάση αυτή μας επιτρέπει να ορίσουμε παραμέτρους όπως την απόσταση στην οποία θέλουμε να «βλέπει» το ρομπότ μας και το χρονικό διάστημα ανά το οποίο θα γίνεται έλεγχος για ανίχνευση αντικειμένων. Για να χρησιμοποιήσουμε τον αισθητήρα υπερήχων πρέπει πρώτα να ορίσουμε στο ρομπότ σε ποια θύρα είναι συνδεδεμένος. Αν τον συνδέσουμε στην θύρα ένα θα γράψουμε:

UltrasonicSensor myUS = new UltrasonicSensor(SensorPort.S1);

Έπειτα θα αρχικοποιήσουμε το FeatureDetector. Μπορούμε να διαλέξουμε ανάμεσα από δύο constructors. Στον πρώτο που δέχεται τρία ορίσματα, οπού myUS είναι ο αισθητήρας υπερήχων, maxDistance είναι η μεγαλύτερη απόσταση στην οποία θέλουμε να «βλέπει» ο αισθητήρας μας και delay είναι το χρονικό διάστημα που θα περιμένει ο αισθητήρας για να ελέγξει ξανά αν υπάρχει εμπόδιο.

float maxDistance = 50; (σε εκατοστά)

*int delay = 500; (*σε milliseconds)

RangeFeatureDetector(myUS, maxDistance, delay)

Ο δεύτερος constructor έχει ένα επιπλέον όρισμα το angle, που μας δείχνει τη γωνία στην οποία είναι στραμμένος ο αισθητήρας.

double angle = 0; (σε μοίρες, το μηδέν σημαίνει ότι είναι στραμμένος προς τα μπροστά, τα θετικά νούμερα προς τα αριστερά και τα αρνητικά προς τα δεξιά)

RangeFeatureDetector(myUS, maxDistance, delay, angle)

Τώρα μπορούμε να χρησιμοποιήσουμε τον ανιχνευτή αντικειμένων. Είναι πολύ σημαντικό να ελέγχουμε κάθε φορά αν το αντικείμενο result είναι διάφορο του null γιατί σε περίπτωση που είναι null και προσπαθήσουμε να καλέσουμε κάποια μέθοδο θα μας πετάξει ένα null pointer exception:

Feature result = myFD.scan();

if(result != null)

System.out.println(“Range: “ + result.getRangeReading().getRange());

Μέθοδοι της κλάσης αυτής:

- *getMaxDistance()* - Μας επιστρέφει την μεγαλύτερη απόσταση στην οποία μπορεί ο αισθητήρας μας να ανιχνεύσει αντικείμενα.

- *scan()* - Ο αισθητήρας στέλνει ένα κύμα και μας επιστρέφει την πιο κοντινή απόσταση στην οποία ανιχνεύτηκε κάποιο αντικείμενο. Σε περίπτωση που η τιμή που θα μας επιστρέψει είναι ίδια με αυτή που μας επέστρεψε η μέθοδος getMaxDistance(), αυτό σημαίνει ότι δεν υπάρχει κάποιο εμπόδιο μέχρι τη συγκεκριμένη απόσταση.

- *setMaxDistance()* - Μας επιτρέπει να ορίσουμε την μεγαλύτερη απόσταση στην οποία μπορεί ο αισθητήρας μας να ανιχνεύσει αντικείμενα.

- TouchFeatureDetector

Ο TouchFeatureDetector επιτρέπει σε έναν αισθητήρα επαφής να χρησιμοποιηθεί ως αισθητήρας απόστασης αναφέροντας τη θέση του αισθητήρα αφής για το API ανίχνευσης αντικειμένων. Για να χρησιμοποιήσουμε έναν αισθητήρα επαφής αρχικά πρέπει να ορίσουμε σε ποια θύρα είναι συνδεδεμένος. Αν τον συνδέσουμε στην θύρα δύο θα γράψουμε:

TouchSensor myTS = new TouchSensor(SensorPort.S2);

Έπειτα θα αρχικοποιήσουμε το FeatureDetector. Μπορούμε να χρησιμοποιήσουμε τον απλό constructor με τον οποίο υποθέτουμε ότι ο αισθητήρας βρίσκεται ακριβώς στο κέντρο του ρομπότ μας:

FeatureDetector myFD = new TouchFeatureDetector(myTS);

Ή αν θέλουμε μεγάλη ακρίβεια στις κινήσεις μας μπορούμε να χρησιμοποιήσουμε αυτό τον constructor ο οποίος μας επιτρέπει να θέσουμε σε ποια θέση ακριβώς βρίσκεται ο αισθητήρας σε σχέση με το κέντρο του ρομπότ:

double xOffset = 30; (σε χιλιοστά)

double yOffset = 40; (σε χιλιοστά)

FeatureDetector myFD = new TouchFeatureDetector(myTS, xOffset, yOffset);

Τώρα είμαστε έτοιμοι να χρησιμοποιήσουμε τον ανιχνευτή επαφής αντικειμένων. Πρέπει να γνωρίζουμε ότι ο ανιχνευτής μας ενημερώνει ότι κάποιο αντικείμενο βρέθηκε μόλις πατηθεί το κουμπί του αισθητήρα. Το κουμπί πρέπει να απελευθερωθεί και να ξαναπατηθεί για να μας στείλει σήμα ότι υπάρχει και δεύτερο αντικείμενο. Επιπλέον όπως και στον RangeFeatureDetector, είναι πολύ σημαντικό να ελέγχουμε κάθε φορά αν το αντικείμενο είναι διάφορο του null για να αποφύγουμε το null pointer exception.

Μέθοδοι της κλάσης αυτής:

- *scan()* - Ελέγχει αν υπάρχει κάποιο αντικείμενο. Σε περίπτωση που υπάρχει μας επιστρέφει ένα αντικείμενο τύπου Feature και μπορούμε με τη χρήση της μεθόδου *getRangeReading()* να πάρουμε πληροφορίες για την απόσταση του αντικειμένου ή αν δεν υπάρχει μας επιστρέφει null.

- *notifyListeners*(*Feature* feature) - Μας επιτρέπει να ενημερώνουμε τους listeners ότι ανιχνεύτηκε κάποιο αντικείμενο.

3.4.3 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ ΕΛΕΓΧΟ ΣΥΜΠΕΡΙΦΟΡΑΣ

Όταν προγραμματίζουμε σε Java, λόγω συνήθειας, είναι πιο απλό να σκεφτόμαστε με λογική σειρά τα if και then που θα αποτελέσουν το πρόγραμμά μας. Αυτός ο τρόπος προγραμματισμού είναι εύκολος όταν ξεκινάμε να γράφουμε ένα πρόγραμμα γιατί δεν απαιτεί πολύ σκέψη στην αρχή, ούτε κάποιο προσχέδιο. Μπορούμε απλά να ξεκινήσουμε να προγραμματίζουμε αμέσως. Το πρόβλημα με την μέθοδο συγγραφής κώδικα που μόλις περιγράψαμε είναι ότι σύντομα ο κώδικάς μας γίνεται περίπλοκος και η επέκτασή του ή απλές αλλαγές σε αυτόν είναι δύσκολες.

Το πιο σημαντικό ίσως εφόδιο που μας παρέχει το LeJOS για τη δημιουργία των προγραμμάτων μας είναι ο έλεγχος συμπεριφοράς με το interface Behavior και την κλάση Arbitrator. Σε αντίθεση με τα if και then, εδώ απαιτείται να σκεφτούμε και να κάνουμε ένα προσχέδιο πριν ξεκινήσουμε να γράφουμε κώδικα, αλλά αυτό θα μας

γλιτώσει από πολύ κόπο αργότερα. Ο κώδικάς μας θα είναι χωρισμένος σε συμπεριφορές, εύκολα κατανοητός και με απλή δομή. Επίσης, το να προσθέσουμε και να αφαιρέσουμε συμπεριφορές είναι πολύ εύκολο αφού δεν επηρεάζεται ο υπόλοιπος κώδικάς μας, ενώ ταυτόχρονα μας δίνεται και η δυνατότητα να ελέγχουμε και να τρέχουμε στο ρομπότ μας την κάθε μία συμπεριφορά μόνη της ανεξάρτητα από τις υπόλοιπες με αποτέλεσμα την πιο γρήγορη και εύκολη απασφαλμάτωση.

Ο τρόπος υλοποίησης του ελέγχου συμπεριφοράς στο LeJOS είναι αρκετά απλός. Οι πιο βασικοί κανόνες στους οποίους βασίζεται είναι οι ακόλουθοι:

- Κάθε συμπεριφορά έχει τη δική της ξεχωριστή κλάση
- Μόνο μία συμπεριφορά τη φορά μπορεί να είναι ενεργή και να ελέγχει το ρομπότ μας
- Κάθε συμπεριφορά γνωρίζει πότε πρέπει να πάρει τον έλεγχο
- Κάθε συμπεριφορά έχει τη δική της προτεραιότητα και αυτή που είναι ενεργή έχει πάντα την υψηλότερη προτεραιότητα από όσες πληρούν τις προϋποθέσεις για να πάρουν τον έλεγχο.
- Το interface Behavior

Το interface Behavior είναι αυτό στο οποίο ορίζουμε με ποιο τρόπο θα αντιδράσει το ρομπότ μας σε ορισμένα ερεθίσματα που δέχεται από τους αισθητήρες ή σε καταστάσεις στις οποίες μπορεί να βρεθεί. Αποτελείται από τρείς public μεθόδους:

- *takeControl()* - Μας επιστρέφει true όταν η συμπεριφορά αυτή πρέπει να γίνει ενεργή. Για παράδειγμα αν θέλουμε όταν ο αισθητήρας υπερήχων εντοπίσει κάποιο εμπόδιο σε απόσταση μικρότερη ή ίση των 10 εκατοστών να γίνει κάποια συγκεκριμένη ενέργεια από το ρομπότ μας, πρέπει να κάνουμε την μέθοδο αυτή να μας επιστρέφει true όταν εντοπιστεί εμπόδιο στη συγκεκριμένη απόσταση.

  ```
  public boolean takeControl() {
      return distance.getDistance() <= 10;
  }
  ```

Αν θέλουμε κάποια συγκεκριμένη ενέργεια να γίνεται συνέχεια, για παράδειγμα το ρομπότ μας να κινείται σε ευθεία, ο κώδικας της takeControl() θα είναι:

```
public boolean takeControl() {
    return true;
}
```

- *action()* - Ο κώδικας τον οποίο θα γράψουμε μέσα σε αυτή τη μέθοδο θα εκτελεστεί όταν η συγκεκριμένη συμπεριφορά γίνει ενεργή, δηλαδή όταν η takeControl() μας επιστρέψει true. Στο παράδειγμά μας, αυτή η μέθοδος θα κληθεί αυτόματα μόλις ο αισθητήρας υπερήχων εντοπίσει κάποιο εμπόδιο σε απόσταση μικρότερη ή ίση από 10 εκατοστά και η αντίδραση του ρομπότ μας θα είναι να ανάψει το λαμπάκι του αισθητήρα χρώματος πράσινο. Η συμπεριφορά θα είναι ενεργή για όση ώρα εκτελείται ο κώδικας της μεθόδου *action()* ή μέχρι να κληθεί κάποια συμπεριφορά με μεγαλύτερη προτεραιότητα.

```
public void action() {
        color.setFloodlight(Color.GREEN);
}
```

- *suppress()* - Η μέθοδος αυτή καλείται όταν η μέθοδος *takeControl()* κάποιας άλλης συμπεριφοράς με υψηλότερη προτεραιότητα επιστρέψει true. Με τη μέθοδο *suppress()* πρέπει να σταματήσουμε άμεσα τον κώδικα της *action()* που τρέχει τώρα. Μπορούμε επίσης να τη χρησιμοποιήσουμε για να ενημερώσουμε τυχόν μεταβλητές που έχουν αλλάξει πριν τερματίσουμε τον κώδικα. Η πιο συνηθισμένη και απλή χρήση της μεθόδου είναι να δηλώσουμε μία μεταβλητή, τη suppressed. Η μεταβλητή αυτή μας βοηθάει να γνωρίσουμε ότι η μέθοδος κλήθηκε χωρίς προβλήματα και προτείνεται να τη χρησιμοποιούμε σε κάθε συμπεριφορά.

```
boolean suppressed = false;

public void suppress() {
        suppressed = true;
}
```

Στο παράδειγμά μας, έχουμε μόνο μία συμπεριφορά οπότε η μέθοδος αύτη δε θα κληθεί ποτέ. Παρόλα αυτά, για να ικανοποιήσουμε το interface Behavior πρέπει να υπάρχει αλλά μπορεί να είναι κενή:

```
public void suppress() {}
```

- Η κλάση Arbitrator

Αφού δημιουργήσουμε όλες τις συμπεριφορές που θέλουμε για το ρομπότ μας, χρησιμοποιούμε την κλάση Arbitrator για να ελέγχουμε ποια συμπεριφορά θα έχει τον έλεγχο. Θα μπορούσαμε να χαρακτηρίσουμε την κλάση αυτή ως διαιτητή των συμπεριφορών, αφού ελέγχει ποια συμπεριφορά πρέπει να εκτελεστεί και έπειτα της δίνει τον έλεγχο.

Έχουμε δύο constructors της κλάσης:

- *Arbitrator(Behavior[] behaviors)* - Έχει ένα όρισμα που είναι ένας πίνακας από όλες τις συμπεριφορές που θα έχει το πρόγραμμά μας με σειρά προτεραιότητας. Στη θέση 0 του πίνακα βρίσκεται η συμπεριφορά με τη χαμηλότερη προτεραιότητα.

- *Arbitrator(Behavior[] behaviors, boolean returnWhenInactive)* - Έχει δύο ορίσματα. Το πρώτο είναι αυτό που περιγράψαμε παραπάνω και το δεύτερο όρισμα αν είναι true, τότε σε περίπτωση που δεν υπάρχει καμία συμπεριφορά που η μέθοδος της *takeControl()* να είναι true για να εκτελεστεί κάποιο action, τερματίζει το πρόγραμμά μας. Διαφορετικά το πρόγραμμά μας τρέχει συνέχεια μέχρι να βρεθεί κάποια *takeControl()* που να πληρούνται οι προϋποθέσεις της. Αν θέλουμε μπορούμε να τερματίσουμε το πρόγραμμα χειροκίνητα την ώρα που τρέχει πατώντας ταυτόχρονα τα κουμπιά Enter και Escape.

Η κλάση αυτή έχει μία μέθοδο:

- *start()* - Όταν καλέσουμε τη μέθοδο αυτή το πρόγραμμά μας αρχίζει τη διαδικασία ελέγχου για να βρει ποια συμπεριφορά πρέπει να είναι ενεργή κάθε στιγμή, ανάλογα με τα δεδομένα που έχει. Οι μέθοδοι *takeControl()* από όλες τις συμπεριφορές καλούνται με σειρά προτεραιότητας ξεκινώντας από αυτές με την υψηλότερη προτεραιότητα και ελέγχει αν η τιμή που θα επιστραφεί είναι true. Μόλις βρεθεί κάποια *takeControl()* που επιστρέψει true καλείται η αντίστοιχη *action(),* άρα η συμπεριφορά αυτή γίνεται ενεργή. Αν κάποια άλλη συμπεριφορά έτρεχε πριν, αυτόματα θα κληθεί η μέθοδος *suppress()* για εκείνη τη μέθοδο.

Για να καταλάβουμε πως λειτουργεί καλύτερα ο έλεγχος συμπεριφοράς ακολουθεί ένα ολοκληρωμένο παράδειγμα το οποίο περιλαμβάνει και το παράδειγμα που περιγράψαμε παραπάνω. Έστω ότι έχουμε έναν αισθητήρα υπερήχων συνδεδεμένο στη θύρα 1 του ρομπότ μας και έναν αισθητήρα χρώματος στη θύρα 2. Το ρομπότ μας θα έχει συνέχεια αναμμένο το λαμπάκι του αισθητήρα χρώματος κόκκινο και όταν εντοπίσει κάποιο αντικείμενο σε απόσταση μικρότερη ή ίση των 10 εκατοστών το λαμπάκι θα γίνεται κόκκινο. Μπορούμε να δοκιμάσουμε το πρόγραμμα εύκολα βάζοντας και βγάζοντας το χέρι μας μπροστά από τον αισθητήρα υπερήχων για να αλλάξουμε την απόσταση που αντιλαμβάνεται.

Πίνακας 4"Παράδειγμα προγραμματισμού με τη χρήση συμπεριφορών"

```
package myPackage;

import lejos.nxt.*; // για να μπορούμε να χρησιμοποιήσουμε όλες τις διαθέσιμές
                    // μεθόδους για το NXT
import lejos.robotics.subsumption.*; // για χρησιμοποιήσουμε το interface
                                     // Behavior και την κλάση Arbitrator
import lejos.robotics.Color; // για να χρησιμοποιήσουμε το Color. για τη δήλωση
```

```
                                        // του χρώματος που θέλουμε αντί για έναν αριθμό
public class Test {
        public static UltrasonicSensor distance = new UltrasonicSensor(
                        SensorPort.S1); // αισθητήρας υπερήχων
        public static ColorSensor color = new ColorSensor(
                        SensorPort.S2); //αισθητήρα χρώματος
        public static void main(String[] args) throws Exception {
                Behavior b1 = new Example(); // πρώτη συμπεριφορά
                Behavior b2 = new Example2(); // δεύτερη συμπεριφορά
                Behavior[] behav = { b1, b2 }; // η b2 έχει την μεγαλύτερη προτεραιότητα
                Arbitrator arbi = new Arbitrator(behav); // δημιουργία Arbitrator
                arbi.start(); // ξεκινάμε το έλεγχο
        }
}
class Example implements Behavior { // πρώτη συμπεριφορά
        boolean suppressed = false; // δήλωση μεταβλητής suppressed με τιμή false
        public boolean takeControl() {
                return true; // θα γίνεται πάντα, όσο οι συμπεριφορές με υψηλότερη
                                //προτεραιότητα έχουν την takeControl() τους false
        }
        public void action() {
                Test.color.setFloodlight(Color.RED); // ανάβει το λαμπάκι κόκκινο
                        while (!suppressed) {
                        Thread.yield();// αυτό γίνεται συνέχεια μέχρι να κληθεί η suppress()
                         }
        }
        public void suppress() {
                suppressed = true; // δεν έχουμε κάτι να αποθηκεύσουμε ή να
                                        // σταματήσουμε οπότε απλά θέτουμε τη μεταβλητή
                                        // suppressed ίση με true
        }
}
class Example2 implements Behavior {// δεύτερη συμπεριφορά
        boolean suppressed = false; // δήλωση μεταβλητής suppressed με τιμή false
        public boolean takeControl() {
                return Test.distance.getDistance() <= 10; // θα επιστρέψει true όταν
                                                        //ο αισθητήρας υπερήχων
                                                        // εντοπίσει εμπόδιο σε
                                                        //απόσταση μικρότερη ή ίση
                                                        // των 10 εκατοστών
        }
        public void action() {
                Test.color.setFloodlight(Color.GREEN); // το λαμπάκι ανάβει πράσινο
        }
        public void suppress() { // η μέθοδος αυτή δε θα κληθεί ποτέ αφού αυτή
                                        //είναι η συμπεριφορά με την πιο μεγάλη
                                        //προτεραιότητα για αυτό την αφήνουμε κενή
        }
}
```

3.4.4 ΒΟΗΘΗΤΙΚΑ ΠΡΟΓΡΑΜΜΑΤΑ JAVA

Ένα επιπλέον πολύ χρήσιμο στοιχείο του LeJOS είναι ότι υποστηρίζει αρκετές από τις δομές δεδομένων και τα βοηθητικά προγράμματα που υπάρχουν στη Java όπως:

- Lists
- ArrayLists
- BitSets
- Hashtables
- Queues
- Stacks
- Ventors
- Timers
- Random
- Properties

3.4.5 ΔΙΑΧΕΙΡΙΣΗ ΣΦΑΛΜΑΤΩΝ

Το LeJOS μας βοηθάει να χειριστούμε τα διάφορα σφάλματα που μπορεί να προκύψουν κατά την εκτέλεση του προγράμματός μας. Υποστηρίζει όλα τα exceptions που υπάρχουν και στην κλασική γλώσσα Java, ενώ ταυτόχρονα μας δίνεται η δυνατότητα να δημιουργήσουμε τα δικά μας exceptions. Ο κώδικας που ακολουθεί είναι το παράδειγμα που χρησιμοποιείται στη σελίδα του LeJOS. (http://lejos.sourceforge.net/nxt/nxj/tutorial/ErrorHandlingAndDebugging/ErrorHandling_and_debugging.htm). (4)

Πίνακας 5"Παράδειγμα κώδικα για διαχείριση σφαλμάτων"

```
1. public class ExceptionTest {
2.     static void m1()
3.     {
4.         int test[] = new int[2];
5.         // δημιουργούμε επίτηδες ένα λάθος που θα οδηγήσει σε exception
6.         test[0] = test[1] + test[2];
7.     }
8.
9.     static void m2()
10.    {
11.        m1();
12.    }
13.
14.    public static void main(String[] args) throws Exception
15.    {
16.        System.out.println("Running");
17.        m2();
18.    }
19. }
```

Σε περίπτωση που κατά τη διάρκεια εκτέλεσης του κώδικα μας συμβεί κάποιο exception που δεν έχουμε προβλέψει, η εκτέλεση του προγράμματός μας θα σταματήσει και στην οθόνη του NXT τούβλου θα εμφανιστεί ένα σχετικό μήνυμα. Στο παράδειγμα που χρησιμοποιούμε το λάθος είναι πολύ προφανές. Ο πίνακας test έχει μόνο δύο θέσεις την test[0] και την test[1]. Εμείς προσπαθούμε να χρησιμοποιήσουμε και μια Τρίτη θέση την test[2]. Αυτό θα προκαλέσει ArrayIndexOutOfBoundsException. Στην οθόνη του NXT τούβλου μας θα μας εμφανιστεί το παρακάτω μήνυμα:

Πίνακας 6 "Η έξοδος της οθόνης του τούβλου μας όταν συμβεί κάποιο Ecxeption"

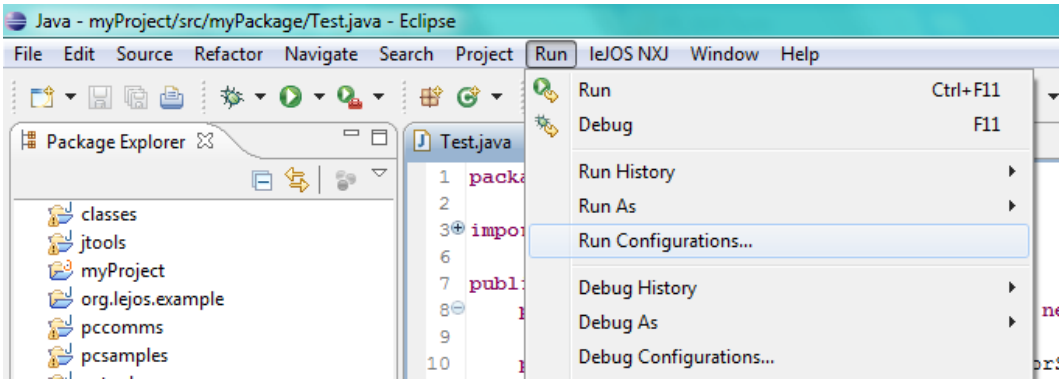
```
Exception: 28
at: 20(11)
at: 21(1)
at: 22(9)
```

Αν και βλέποντας αυτό το μήνυμα δεν βγάζουμε κανένα άμεσο συμπέρασμα δεν είναι δύσκολο να καταλάβουμε τι σημαίνει αν γνωρίζουμε ότι:

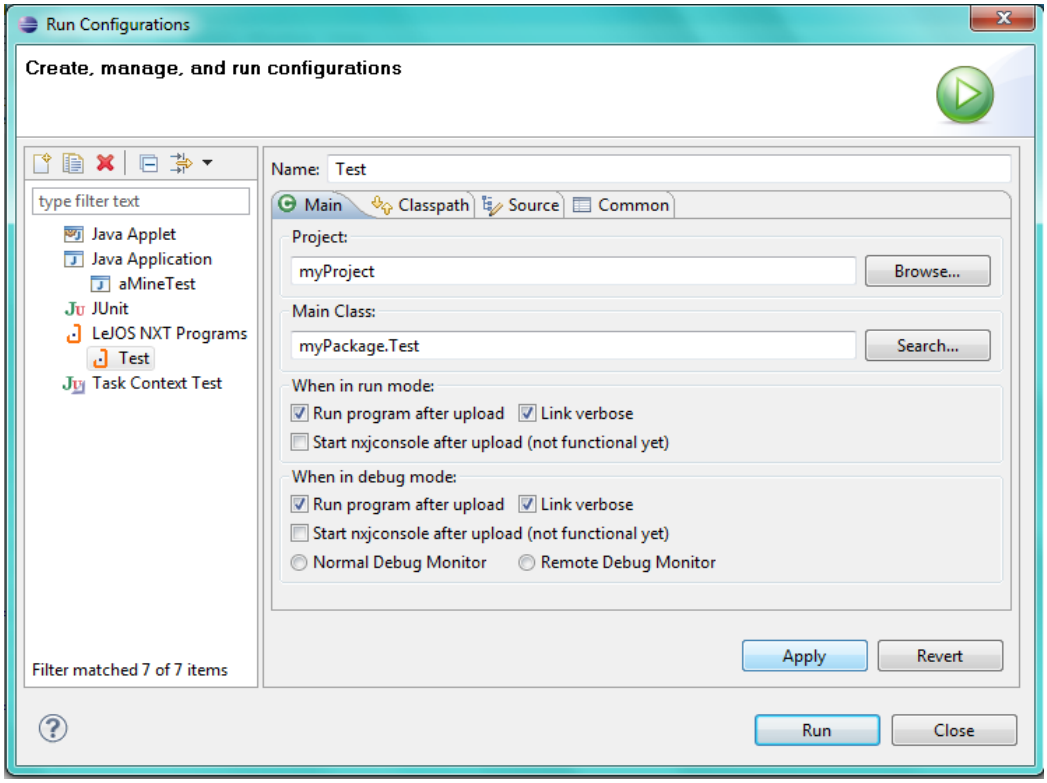
- Η πρώτη γραμμή που εμφανίζεται, δηλαδή το «Exception: 28» στο συγκεκριμένο παράδειγμα, μας δείχνει το είδος του exception.

- Οι τρεις γραμμές που ακολουθούν σχετίζονται με την μέθοδο στην οποία υπάρχει το λάθος που προκάλεσε το συγκεκριμένο exception. Η πρώτη γραμμή σημαίνει ότι το σφάλμα βρίσκεται στη μέθοδο νούμερο 20 στην τοποθεσία 11. Οι σειρές που ακολουθούν αποτελούν το μονοπάτι προς τη μέθοδο αυτή, δηλαδή η μέθοδος 20 κλήθηκε από τη μέθοδο 21 στην τοποθεσία 1, που κλήθηκε από τη μέθοδο 22 στην τοποθεσία 9.

Τώρα που είδαμε τι σημαίνουν όλα αυτά τα νούμερα, ας δούμε πως σχετίζονται με τα πραγματικά exceptions και με τις μεθόδους που υπάρχουν στο παράδειγμά μας. Για να γίνει αυτό χρειαζόμαστε ένα πίνακα που να αντιστοιχεί τους αριθμούς αυτούς στα αληθινά στοιχεία. Αρκεί να πάμε στο Eclipse και να επιλέξουμε από το Run → Run Configurations και να επιλέξουμε το Link Verbose όπως δείχνουν οι εικόνες.



Εικόνα 27 " Οδηγίες για Eclipse, πηγαίνουμε στο Run --> Run Configurations"



Εικόνα 28 "Οδηγίες για Eclipse, επιλέγουμε τα δύο κουτάκια που γράφουν Link verbose και πατάμε το κουμπί Apply"

Όταν τρέξουμε το πρόγραμμά μας, στην κονσόλα του Eclipse θα εμφανιστεί ο πίνακας με τις αντιστοιχήσεις που θέλουμε. Στο συγκεκριμένο παράδειγμα ο πίνακας που θα εμφανιστεί είναι αυτός που ακολουθεί.

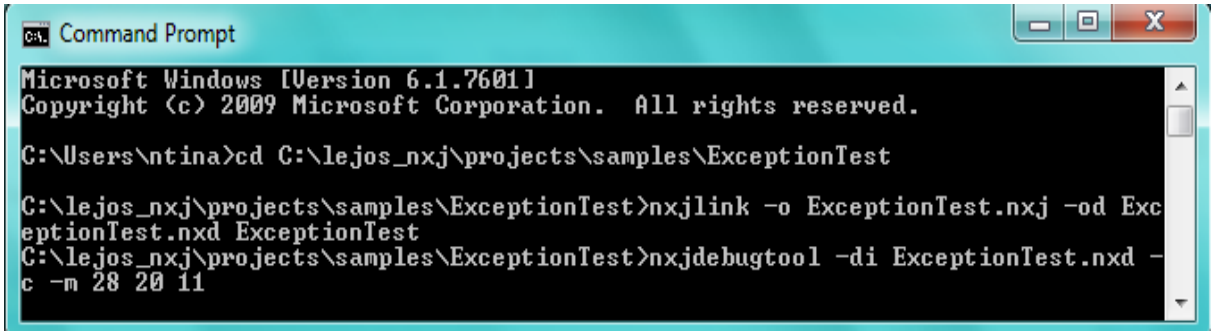
Πίνακας 7 "Αντιστοιχίες αριθμών που εμφανίζονται στα σφάλματα με τις κλάσεις του προγράμματός μας"

Class 1: java.lang.Throwable
Class 2: java.lang.Error
Class 3: java.lang.OutOfMemoryError
Class 4: boolean
Class 5: char
Class 6: float
Class 7: double
Class 8: byte
Class 9: short
Class 10: int
Class 11: long
Class 12: void
Class 13: java.lang.Object[]
Class 14: java.lang.NoSuchMethodError
Class 15: java.lang.StackOverflowError
Class 16: java.lang.NullPointerException
Class 17: boolean[]
Class 18: char[]

```
Class 19: float[]
Class 20: double[]
Class 21: byte[]
Class 22: short[]
Class 23: int[]
Class 24: long[]
Class 25: reserved
Class 26: java.lang.ClassCastException
Class 27: java.lang.ArithmeticException
Class 28: java.lang.ArrayIndexOutOfBoundsException
Class 29: java.lang.IllegalArgumentException
Class 30: java.lang.InterruptedException
Class 31: java.lang.IllegalStateException
Class 32: java.lang.IllegalMonitorStateException
Class 33: java.lang.ThreadDeath
.... other classes omitted
Method 0: java.lang.Object.<init>() PC 1976 Signature id 2
Method 1: java.lang.Object.getClass() PC 1977 Signature id 121
Method 2: java.lang.Object.toString() PC 1985 Signature id 123
Method 3: java.lang.Throwable.i<init>() PC 2013 Signature id 2
Method 4: java.lang.Throwable.<init>(java.lang.String) PC 2023 Signature id 124
Method 5: java.lang.Throwable.getLocalizedMessage() PC 2038 Signature id 125
Method 6: java.lang.Throwable.getMessage() PC 2043 Signature id 29
Method 7: java.lang.Throwable.toString() PC 2048 Signature id 123
Method 8: java.lang.Throwable.fillInStackTrace() PC 2096 Signature id 126
Method 9: java.lang.NullPointerException.<init>() PC 2109 Signature id 2
Method 10: java.lang.Class.isInterface() PC 2114 Signature id 133
Method 11: java.lang.Class.toString() PC 2131 Signature id 123
Method 12: java.lang.String.<init>(int) PC 2177 Signature id 128
Method 13: java.lang.String.<init>(char[], int, int) PC 2189 Signature id 141
Method 14: java.lang.String.charAt(int) PC 2206 Signature id 145
Method 15: java.lang.String.length() PC 2231 Signature id 155
Method 16: java.lang.String.toString() PC 2237 Signature id 123
Method 17: java.lang.String.valueOf(java.lang.Object) PC 2239 Signature id 167
Method 18: java.lang.Thread.run() PC 2253 Signature id 1
Method 19: java.lang.Thread.currentThread() Native id 12
Method 20: ExceptionTest.m1() PC 2273 Signature id 175
Method 21: ExceptionTest.m2() PC 2288 Signature id 176
Method 22: ExceptionTest.main(java.lang.String[]) PC 2292 Signature id 0
Method 23: lejos.nxt.VM.<clinit> PC 2304 Signature id 3
Method 24: lejos.nxt.VM.<init>() PC 2323 Signature id 2
.... other methods omitted
```

Τώρα είναι όλα κατανοητά. Σύμφωνα με τον πίνακα, το Exception 28 αντιστοιχεί στο γνωστό μας java.lang.ArrayIndexOutOfBoundsException, που σημαίνει ότι προσπαθήσαμε να χρησιμοποιήσουμε μία θέση πίνακα που δεν υπάρχει. Όσο αφορά τις μεθόδους, το νούμερο 20 αντιστοιχεί στην μέθοδο ExceptionTest.m1, το 21 στην μέθοδο ExceptionTest.m2 και το 22 στην μέθοδο ExceptionTest.main. Αυτό σημαίνει ότι το σφάλμα που προκάλεσε το Exception βρίσκεται στην μέθοδο m1, που κλήθηκε από την μέθοδο m2, η οποία με τη σειρά κλήθηκε από την μέθοδο main. Σε ένα μικρό πρόγραμμα όπως αυτό το παράδειγμα αυτά τα στοιχεία είναι αρκετά για να βρούμε και να διορθώσουμε το σφάλμα. Σε ένα μεγάλο και περίπλοκο πρόγραμμα όμως, τα νούμερα που βρίσκονται στις παρενθέσεις είναι απαραίτητα. Αυτά προσδιορίζουν τη γραμμή του κώδικα στην οποία βρίσκονται οι μέθοδοι που αναφέραμε. Το LeJOS μας παρέχει ένα εργαλείο που αποκωδικοποιεί τα νούμερα αυτά δίνοντάς μας τη σειρά του κώδικα που βρίσκεται η μέθοδος. Αρκεί να ανοίξουμε το Command Prompt του υπολογιστή μας και να πληκτρολογήσουμε τις ακόλουθες γραμμές (στο παράδειγμά μας, η κλάση ExceptionTest βρίσκεται στο path: C:\lejos_nxj\projects\samples\ExceptionTest\ExceptionTest.java).

Οι εντολές που πρέπει να γράψουμε για το συγκεκριμένο παράδειγμα είναι οι ακόλουθες:



Εικόνα 29 "Οι εντολές που πρέπει να γράψουμε στο command promt για να μας εμφανίσει αναλυτικές πληροφορίες για κάποιο exception που προκλήθηκε"

Αυτό θα μας επιστρέψει τα παρακάτω, δηλαδή αναλυτικά όλες τις πληροφορίες που θέλαμε:

The class number 28 refers to:

java.lang.ArrayIndexOutOfBoundsException
(ArrayIndexOutOfBoundsException.java)

The method number 20 refers to:

ExceptionTest.m1()V (ExceptionTest.java)

PC 11 refers to:

line 6 in ExceptionTest.java

Κάποιες φορές, κατά τη διάρκεια ανεβάσματος ή εκτέλεσης του κώδικά μας, μπορεί το ρομπότ μας να σταματήσει να λειτουργεί και να μην ανταποκρίνεται. Τότε στην οθόνη του NXT τούβλου μας εμφανιστεί ένα μήνυμα παρόμοιο με το ακόλουθο:

DATA ABORT

PC 00140BAC

AASR 1831BF01

ASR 00020601

OPCODE ???

DEBUG1 00020010

DEBUG2 00000000

Αυτό σημαίνει ότι το firmware LeJOS έχει σταματήσει να λειτουργεί κανονικά. Οι πιο συνηθισμένες αιτίες που μας οδηγούν σε DATA ABORT είναι το ανέβασμα αρχείου που δεν είναι NXJ αρχείο ή ανέβασμα και προσπάθεια εκτέλεσης κάποιου ημιτελούς προγράμματος. Η μοναδική λύση όταν συμβεί αυτό, είναι να βγάλουμε τις μπαταρίες από το NXT τούβλο μας. Αυτό είναι πολύ πιθανό να το επαναφέρει στην αρχική του κατάσταση και να χρειαστεί να εγκαταστήσουμε ξανά το LeJOS.

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο αυτό είδαμε τι είναι το LeJOS και αναφέραμε τα πλεονεκτήματά του που ήταν και ο λόγος για τον οποίο το επιλέξαμε. Έπειτα, ασχοληθήκαμε με τον προγραμματισμό. Αναφέραμε τις διάφορες μεθόδους που υπάρχουν για τους κινητήρες και τους αισθητήρες του, τα πιο βασικά βοηθητικά προγράμματα που μας παρέχει και τέλος τον τρόπο διαχείρισης σφαλμάτων. Στο επόμενο κεφάλαιο θα εφαρμόσουμε πολλά από όσα αναφέραμε εδώ για να γράψουμε το δικό μας πρόγραμμα.

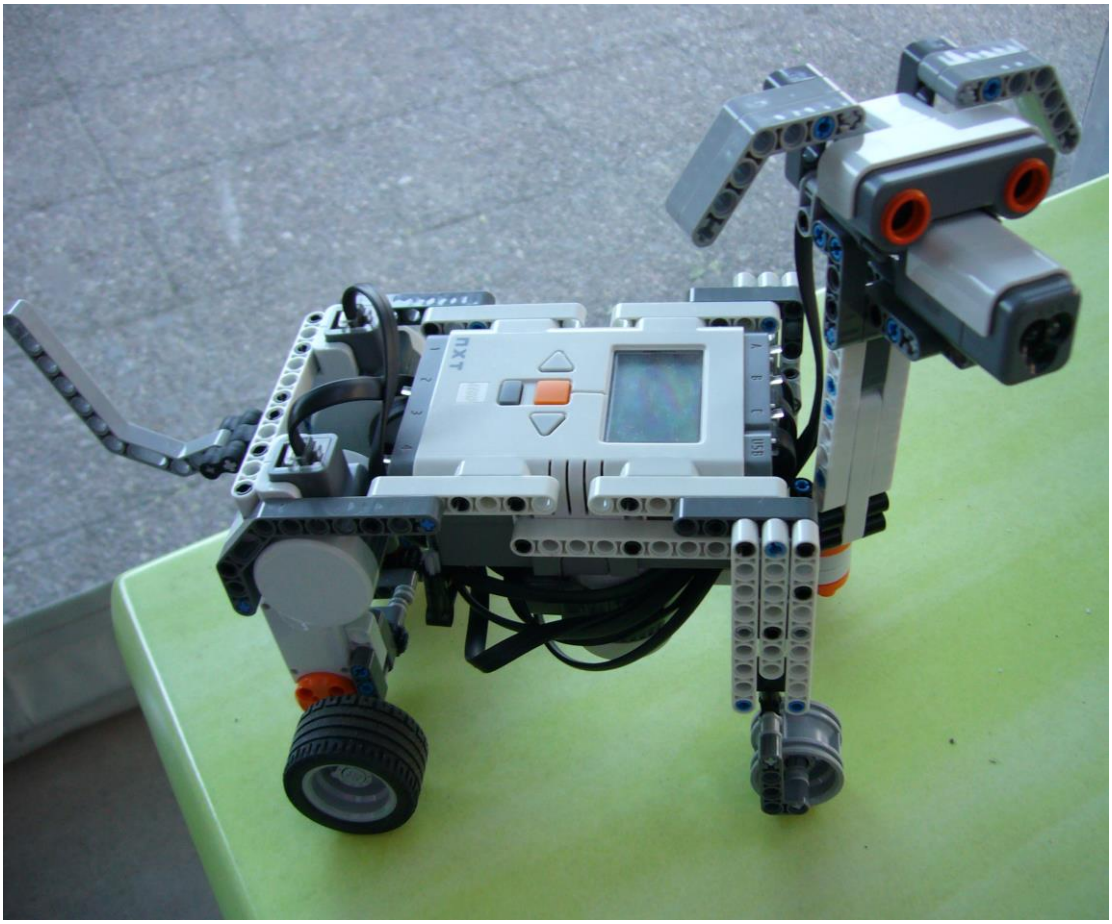
ΚΕΦΑΛΑΙΟ 4

ΠΡΟΓΡΑΜΜΑ ΚΑΙ ΚΩΔΙΚΑΣ

ΕΙΣΑΓΩΓΗ

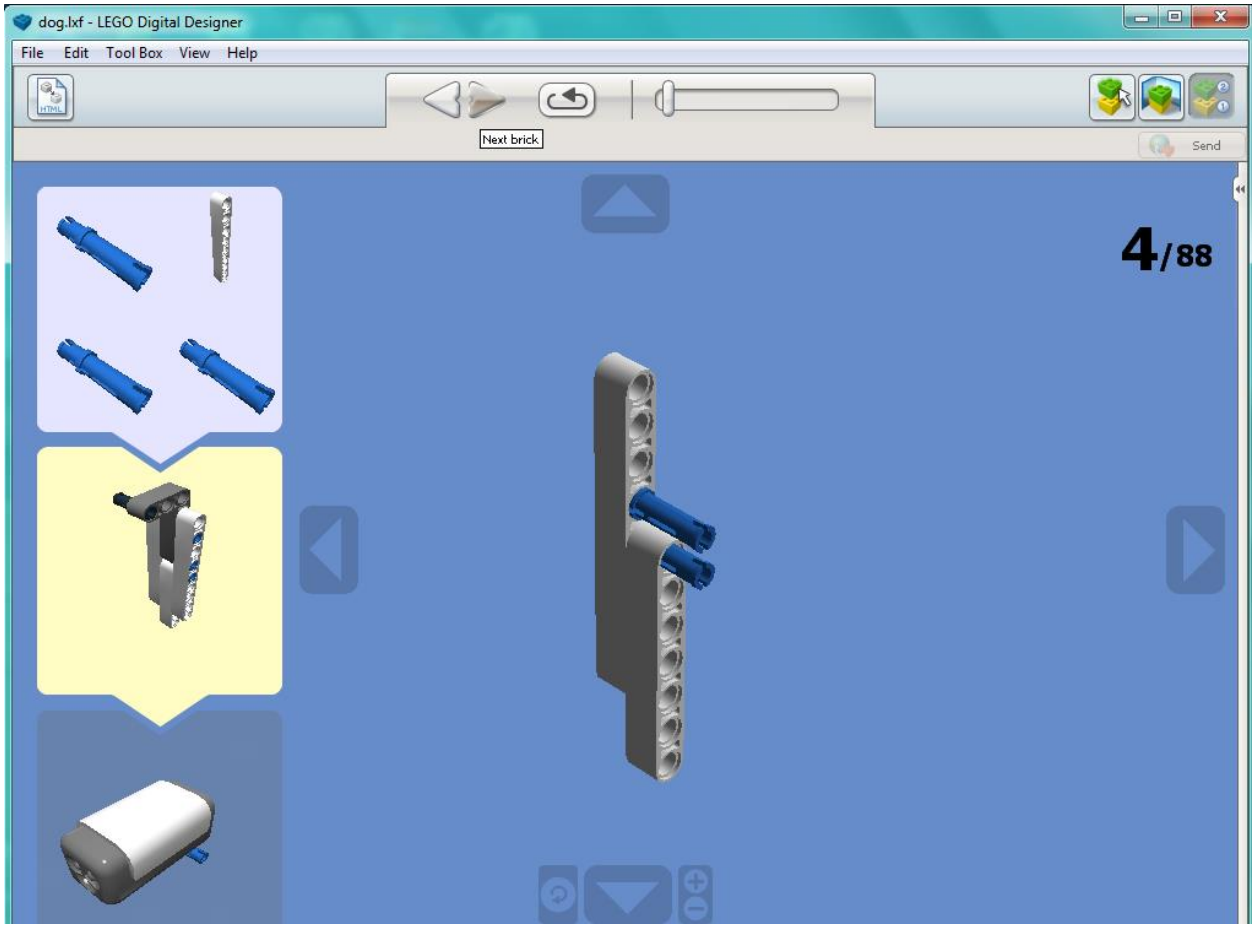
Αφού έχουμε αναφερθεί στην ιστορία των ρομπότ, τα είδη τους, το ρομπότ της Lego, το firmware LeJOS και τις μεθόδους του, ήρθε η ώρα να δημιουργήσουμε ένα δικό μας ολοκληρωμένο πρόγραμμα.

Πρώτα από όλα, χρησιμοποιώντας το NXT τούβλο, τα τουβλάκια, τα γρανάζια, τις ρόδες, τα λάστιχα, τους τρείς κινητήρες και τους αισθητήρες (υπερήχων και χρώματος) κατασκευάσαμε το ρομπότ μας, που μοιάζει με σκύλο. Για να μπορέσουμε να αποθηκεύσουμε τις οδηγίες για την κατασκευή του σκύλου μας χρησιμοποιήσαμε το πρόγραμμα κατασκευής ρομπότ Lego, το Lego Digital Designer. Το NXT τούβλο αποτελεί το σώμα του σκύλου μας. Οι δύο κινητήρες βρίσκονται στα πίσω πόδια του, ένας στο κάθε πόδι, για να μπορεί το ρομπότ μας να κινείται σε ευθεία πορεία και να στρίβει. Ο τρίτος κινητήρας βρίσκεται στο κάτω μέρος του σώματος του και συνδέεται με τον λαιμό του για να έχει τη δυνατότητα να γυρίζει το κεφάλι του δεξιά και αριστερά. Ο αισθητήρας υπερήχων βρίσκεται στο κεφάλι του σκύλου και αποτελεί τα μάτια του λόγω σχήματος και μας δίνει τη δυνατότητα να ανιχνεύουμε εμπόδια που μπορεί να υπάρχουν μπροστά στο ρομπότ.



Εικόνα 30 "Ο σκύλος ρομπότ"

Τέλος, ο αισθητήρας χρώματος, αποτελεί τη μύτη του και επιτρέπει στο ρομπότ μας να αντιλαμβάνεται χρώματα και φωτεινότητα αλλά και να αλλάζει το χρώμα από το λαμπάκι που υπάρχει εκεί. Η διαδικασία δημιουργίας του ρομπότ σκύλου υπάρχει αναλυτικά στο cd της πτυχιακής εργασίας στο αρχείο dog.lxf. Για να ανοίξουμε το αρχείο αυτό πρέπει πρώτα να κατεβάσουμε στον υπολογιστή μας το πρόγραμμα Lego Digital Designer, το οποίο είναι δωρεάν και μπορούμε εύκολα να βρούμε στο διαδίκτυο. Για να μπορέσουμε να δούμε βήμα-βήμα τη διαδικασία και να την επαναλάβουμε αρκεί αφού ανοίξουμε το αρχείο να πάμε στο View → Building Guide Mode (F7). Στην οθόνη μας θα δούμε κάτι παρόμοιο με την εικόνα που ακολουθεί. Όπως φαίνεται και στην εικόνα μας, το πρόγραμμα μας δείχνει αναλυτικά πιο τουβλάκι πρέπει να συνδέσουμε με ποιο. Πατώντας το βελάκι αριστερά πηγαίνουμε στο προηγούμενο βήμα, ενώ πατώντας το βελάκι στα δεξιά πάμε στο επόμενο βήμα.



Εικόνα 31 "Το πρόγραμμα Lego Digital Designer"

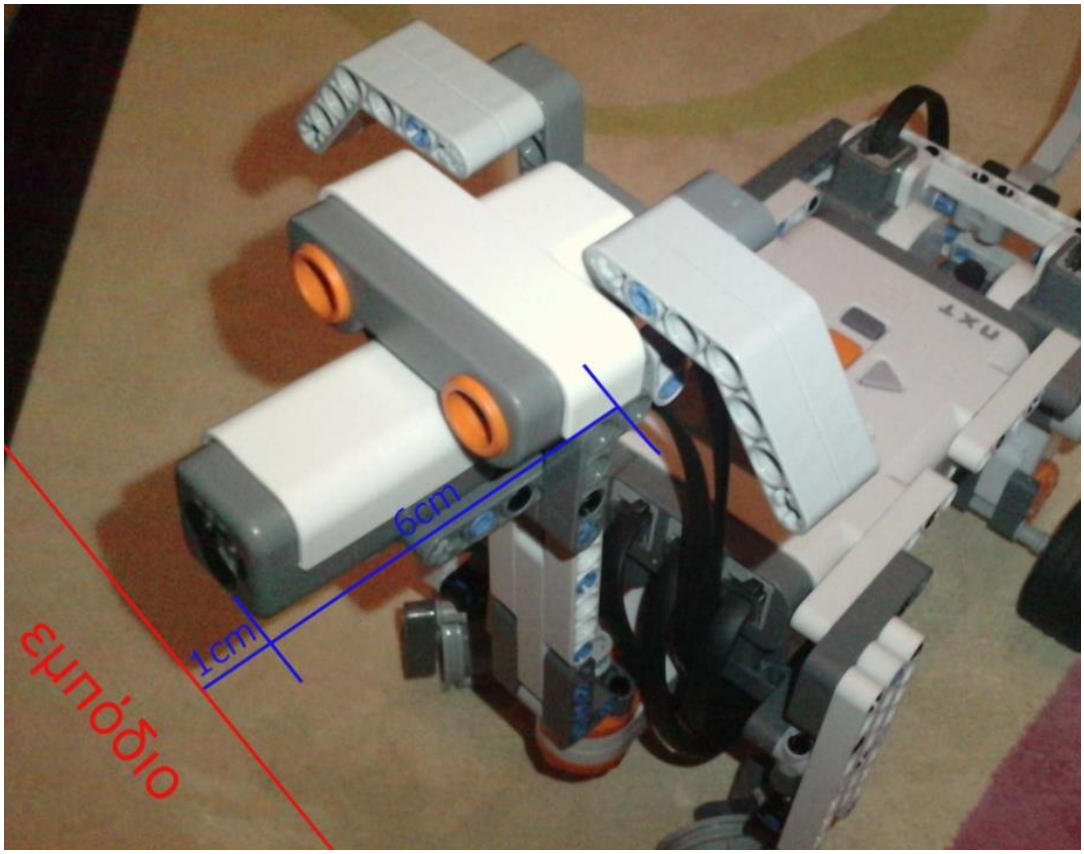
4.1 ΤΟ ΣΕΝΑΡΙΟ

Το κυρίως θέμα είναι η ελεύθερη περιήγηση στο χώρο και η αποφυγή εμποδίων. Με λίγα λόγια ο σκύλος ρομπότ θα κινείται σε ευθεία γραμμή και κάθε φορά που συναντάει ένα εμπόδιο θα ελέγχει αν υπάρχει βέλος πάνω του. Σε περίπτωση που υπάρχει θα στρίβει ανάλογα με τη φορά του βέλους, αν δεν υπάρχει θα ελέγχει την απόσταση αριστερά και δεξιά και θα στρίβει προς τη μεριά που μπορεί να κινηθεί

περισσότερο, χωρίς να συναντήσει εμπόδιο. Το πρόγραμμα θα σταματάει όταν το ρομπότ εντοπίσει κόκκινο χρώμα.

Για να μπορέσουμε να φτιάξουμε ένα τέτοιο πρόγραμμα, πρέπει να δούμε τις δυνατότητες που έχει το ρομπότ μας και να σκεφτούμε πως θα καταλαβαίνει αν υπάρχει βέλος στο εμπόδιο και πώς θα το διαβάζει.

Όπως ήδη αναφέραμε, για να μπορέσει το ρομπότ μας να κινείται έχουμε προσαρμόσει στα πίσω πόδια από έναν κινητήρα. Αυτό μας επιτρέπει την κίνηση σε ευθεία όταν μετακινούμε και τους δύο κινητήρες προς την ίδια φορά, τη στροφή με κίνηση μόνο του ενός κινητήρα ανάλογα με τη μεριά που θέλουμε να στρίψουμε, αλλά και την επιτόπου αναστροφή με αντίθετη κίνηση των κινητήρων. Στην αρχή θα κινείται σε ευθεία γραμμή μέχρι να εντοπίσει κάποιο εμπόδιο με τον αισθητήρα υπερήχων. Τότε, θέλουμε να ο αισθητήρας χρώματος να διαβάσει τι χρώμα έχει το εμπόδιο. Για να συμβεί αυτό με επιτυχία, ο αισθητήρας χρώματος πρέπει να απέχει περίπου 1 εκατοστό από το εμπόδιο και επιπλέον το ποσοστό της σωστής ανάγνωσης χρώματος αυξάνεται όταν ο αισθητήρας φωτίζει το εμπόδιο με το λαμπάκι led που διαθέτει με άσπρο χρώμα. Λόγω της κατασκευής του σκύλου, αυτό πρακτικά σημαίνει ότι ο αισθητήρας υπερήχων πρέπει να μας ενημερώσει μόλις η απόσταση από το εμπόδιο είναι 6 εκατοστά και τότε εμείς να δώσουμε εντολή στους κινητήρες να σταματήσουν να κινούνται.



Εικόνα 32 "Απόσταση από εμπόδιο αισθητήρα χρώματος και αισθητήρα υπερήχων λόγω κατασκευής"

Μόλις σταματήσει η κίνηση του ρομπότ, ο αισθητήρας χρώματος μας επιστρέφει την τιμή του χρώματος που διάβασε. Αν το χρώμα είναι άσπρο, τότε ελέγχει την απόσταση αριστερά και δεξιά και στρίβει προς τη μεγαλύτερη και κινείται ξανά ευθεία μέχρι να βρει το επόμενο εμπόδιο (σε περίπτωση που είναι ίσες στρίβει αριστερά). Σε περίπτωση που το χρώμα είναι κόκκινο, ο σκύλος μας γαυγίζει και ανάβει το λαμπάκι της μύτης του σε 3 διαφορετικά χρώματα (μπλε, κόκκινο και πράσινο). Έπειτα το πρόγραμμά μας τερματίζει και το NXT τούβλο απενεργοποιείται. Στη περίπτωση που το χρώμα δεν είναι ούτε άσπρο, ούτε κόκκινο, το πρόγραμμά μας υποθέτει ότι υπάρχει βέλος στο εμπόδιο και προσπαθεί να το διαβάσει. Το βέλος είναι χρωματισμένο με δύο διαφορετικά χρώματα για να μπορεί να αναληφθεί το ρομπότ μας προς τα ποια μεριά δείχνει. Τα χρώματα που χρησιμοποιήσαμε για τα βέλη είναι κίτρινο, πράσινο και μπλε. Ο σκύλος μας έχει τη δυνατότητα με τον τρίτο κινητήρα να στρίβει το κεφάλι του αριστερά και δεξιά και με αυτό τον τρόπο μπορεί να διαβάσει ποιο χρώμα βρίσκεται στα δεξιά του και ποιο στα αριστερά του. Κάθε χρώμα έχει και μία τιμή. Όλα τα βέλη είναι σχεδιασμένα έτσι ώστε το χρώμα με τη μεγαλύτερη τιμή να βρίσκεται στη μύτη του βέλους. Έτσι, ο σκύλος μας στρίβει το κεφάλι του αριστερά και έπειτα ξεκινάει το στρίβει αργά προς τα δεξιά. Κατά τη διάρκεια κίνησης του κεφαλιού δεξιά ο αισθητήρας χρώματος παίρνει 8 δείγματα από το χρώμα που αντιλαμβάνεται. Παίρνουμε πολλά δείγματα για να είμαστε σίγουροι για το αποτέλεσμα, επειδή λόγω της περιστροφής του κεφαλιού, η απόσταση του αισθητήρα χρώματος από το εμπόδιο αυξομειώνεται. Αν η απόσταση είναι πολύ μικρή ή πολύ μεγάλη μας επιστρέφει είτε άσπρο είτε μαύρο. Έπειτα αποθηκεύουμε αυτές τις τιμές σε έναν πίνακα (σε περίπτωση που η μέτρηση είναι ίση με 6 που αντιστοιχεί στο χρώμα άσπρο ή 7 που αντιστοιχεί στο μαύρο στη θέση εκείνη του πίνακα βάζουμε την τιμή 0). Αν το άθροισμα των πρώτων τεσσάρων θέσεων του πίνακα είναι μεγαλύτερο από αυτό των τεσσάρων επόμενων, τότε το βέλος μας δείχνει προς τα αριστερά ενώ αν το πρώτο άθροισμα είναι μικρότερο από το δεύτερο τότε το βέλος δείχνει δεξιά. Έπειτα, το ρομπότ μας στρίβει προς τη μεριά που δείχνει το βέλος κάνοντας επιτόπου στροφή. Για να συμβεί αυτό κινεί τον ένα κινητήρα προς τα μπροστά και τον άλλο προς τα πίσω, ανάλογα με τη μεριά στην οποία πρέπει να στρίψει. Σε περίπτωση που τα δύο αθροίσματα είναι ίσα, το ρομπότ κάνει άλλη μία φορά την διαδικασία ανάγνωσης του βέλους. Σε περίπτωση που αυτή τη φορά κάποιο είναι μεγαλύτερο στρίβει προς τη μεγαλύτερη τιμή. Σε περίπτωση που πάλι είναι ίσα, αντιλαμβάνεται ότι δεν υπάρχει βέλος στο εμπόδιο απλά το εμπόδιο έχει κάποιο χρώμα διαφορετικό από άσπρο ή κόκκινο. Τότε ο σκύλος μας ελέγχει απλά την απόσταση δεξιά και αριστερά και στρίβει προς τη μεριά που μπορεί να κινηθεί περισσότερο χωρίς να συναντήσει ξανά εμπόδιο.

Για να το κάνουμε πιο ενδιαφέρον, το ρομπότ μας θα κινείται και θα τοποθετούμε εμπόδια μπροστά του εκείνη τη στιγμή ή μπορούμε να φτιάξουμε έναν λαβύρινθο και το ρομπότ μας να κινείται μέσα σε αυτόν.

4.2 Ο ΚΩΔΙΚΑΣ

Για τη δημιουργία του κώδικά μας χρησιμοποιήσαμε τις μεθόδους για τους κινητήρες και τους αισθητήρες που μας παρέχει το LeJOS. Για να είναι πιο απλό και πιο κατανοητό από άλλους προγραμματιστές το πρόγραμμά μας χρησιμοποιήσαμε τον έλεγχο συμπεριφοράς, τον οποίο περιγράψαμε παραπάνω. Έτσι, το πρόγραμμά μας είναι χωρισμένο σε συμπεριφορές, οργανωμένο και εύκολα τροποποιήσιμο. Έχουμε τρείς συμπεριφορές τις MoveForward, FindWallReadColor και DoAction, οι οποίες θα εξηγήσουμε τι ακριβώς κάνουν ενώ περιγράφουμε τον κώδικα. Επιπλέον, για να μην έχουμε πολλές γραμμές κώδικα στις μεθόδους action() των συμπεριφορών δημιουργήσαμε τις μεθόδους CountDistance_Turn(), Turn(**int** a, **int** b) και ReadArrow(). Αλλά ας δούμε αναλυτικά τα μέρη από τα οποία αποτελείται ο κώδικάς μας.

4.2.1 IMPORTS

Στην αρχή του κώδικα μας υπάρχουν τα απαραίτητα import. Με τη χρήση της εντολής import μπορούμε να εισάγουμε τις κλάσεις που περιέχονται σ' ένα άλλο πακέτο στο δικό μας πρόγραμμα. Σ' αυτήν την περίπτωση χρειαζόμαστε κάποια από τα πακέτα της βιβλιοθήκης που μας παρέχει το LeJOS και ένα από αυτά που μας παρέχει η Java. Πιο αναλυτικά έχουμε πέντε import:

- **import** java.io.File;

Αυτό το μας δίνει τη δυνατότητα να δημιουργήσουμε ένα αντικείμενο τύπου file και το χρειαζόμαστε για να μπορέσουμε να εισάγουμε στον κώδικα μας το αρχείο ήχου με το γαύγισμα του σκύλου που θα ακούγεται όταν ο σκύλος εντοπίσει εμπόδιο χρώματος κόκκινου που σημαίνει το τέλος του προγράμματός μας.

- **import** lejos.robotics.Color;

Όλες οι μέθοδοι που σχετίζονται με τον αισθητήρα χρώματος δέχονται ορίσματα τύπου int. Αυτό σημαίνει ότι πρέπει να γνωρίζουμε και να θυμόμαστε ποιος αριθμός αντιστοιχεί σε ποιο χρώμα. Για να κάνουμε τον κώδικά μας πιο εύκολο το LeJOS μας παρέχει αυτή την κλάση η οποία κάνει την αντιστοίχιση για εμάς και αντί για νούμερα μας επιτρέπει να γράφουμε στον κώδικά μας τα χρώματα με τη μορφή Color.GREEN το οποίο αντιστοιχεί φυσικά στο πράσινο.

- **import** lejos.robotics.subsumption.*;

Αυτό το import μας επιτρέπει να χρησιμοποιήσουμε στον κώδικά μας τον έλεγχο συμπεριφοράς και να δημιουργήσουμε τις δικές μας συμπεριφορές.

- **import** lejos.nxt.*;

Συμπεριλαμβάνει όλες τις βασικές μεθόδους για το NXT ρομπότ μας για τους κινητήρες και τους αισθητήρες που έχουν δημιουργηθεί επίσημα από τη Lego για NXT.

- **import** lejos.robotics.navigation.DifferentialPilot;

Για να κάνουμε πιο εύκολη την περιήγηση του ρομπότ μας στο χώρο και να μπορεί ο σκύλος μας να κινείται σε ευθεία και να στρίβει με ακρίβεια, χρησιμοποιούμε ένα DifferentialPilot που μας δίνει τη δυνατότητα να αντιμετωπίζουμε το ρομπότ σαν ένα όχημα. Αυτό σημαίνει ότι όταν θέλουμε ο σκύλος μας να κάνει επιτόπου αναστροφή δε χρειάζεται να προγραμματίσουμε τον ένα κινητήρα να κινηθεί προς τα μπροστά και τον άλλο προς τα πίσω ταυτόχρονα και να υπολογίσουμε εμείς πόσες στροφές πρέπει να κάνει ο κάθε κινητήρας για να στρίψει το ρομπότ μας 360 μοίρες. Αρκεί απλά να μετρήσουμε τις διαστάσεις του ρομπότ μας και τη διάμετρο των τροχών του και να τα χρησιμοποιήσουμε ως ορίσματα στον constructor της κλάσης DifferentialPilot, την οποία έχουμε ήδη περιγράψει στο προηγούμενο κεφάλαιο. Αφού δημιουργήσουμε τον δικό μας DifferentialPilot μπορούμε να προγραμματίσουμε το ρομπότ μας να κάνει επιτόπου αναστροφή όπως φαίνεται στο παράδειγμα παρακάτω:

```
DifferentialPilot pilot = new DifferentialPilot(4.32f, 7.9f, Motor.A, Motor.B);
pilot.rotate(360);
```

Έπειτα ακολουθούν τέσσερις κλάσεις. Η μία είναι η κύρια κλάση του προγράμματός μας που περιέχει και την μέθοδο main, ενώ οι άλλες τρεις αντιστοιχούν στις τρεις συμπεριφορές που θα έχει το ρομπότ μας.

4.2.2 Η ΚΛΑΣΗ MAIN

Η κυρίως κλάση του προγράμματός μας ονομάζεται MyDog, χάρη στο σχήμα του ρομπότ μας. Η κλάση αυτή περιέχει τις δηλώσεις του DifferentialPilot, του αισθητήρα υπερήχων και του αισθητήρα χρώματος, αρκετών μεταβλητών που θα χρησιμοποιήσουμε αργότερα, καθώς και την μέθοδο main η οποία είναι η βασική μέθοδος του προγράμματός μας, αφού στη Java όταν εκτελούμε μία κλάση, η εκτέλεση αρχίζει με την κλήση της μεθόδου με όνομα main.

Οι δηλώσεις που γίνονται στην κλάση MyDog είναι:

- **static** UltrasonicSensor distance = **new** UltrasonicSensor(SensorPort.S4);

Δηλώνουμε ότι υπάρχει ένας αισθητήρας υπερήχων συνδεδεμένος στη θύρα νούμερο τέσσερα του NXT τούβλου, τον οποίο θα ονομάζουμε distance.

- **static** ColorSensor color = **new** ColorSensor(SensorPort.S2);

Δηλώνουμε ότι υπάρχει ένας αισθητήρας χρώματος συνδεδεμένος στη θύρα νούμερο δύο, τον οποίο θα ονομάζουμε color.

- **static** ColorSensor.Color clr;

Δηλώνουμε τη μεταβλητή clr η οποία θα χρησιμοποιείται για να αποθηκεύουμε την τιμή του χρώματος που διαβάζουμε από τον αισθητήρα χρώματος.

- **static** DifferentialPilot *pilot*;

Στο πρόγραμμά μας θα υπάρχει ένας DifferentialPilot για να κάνει πιο απλή την κίνηση του ρομπότ μας με όνομα pilot.

- **static int** *mycolor* = -1;

Η μεταβλητή τύπου int με όνομα mycolor χρησιμοποιείται για να καταλαβαίνει η συμπεριφορά DoAction πότε ο αισθητήρας χρώματος έχει διαβάσει κάποιο χρώμα, οπότε πρέπει να αναλάβει δράση και να καλέσει την δική της μέθοδο action. Μόλις η μεταβλητή αυτή πάρει κάποια τιμή διαφορετική του -1,αυτό σημαίνει ότι ο αισθητήρας χρώματος διαβάσει ένα χρώμα.

- **static String**[] *colors* = { "RED", "GREEN", "BLUE", "YELLOW", "none", "none", "WHITE", "BLACK" };

Για να μπορέσουμε να τυπώνουμε κατανοητά μηνύματα για το χρώμα που έχει διαβάσει ο αισθητήρας χρώματος χρησιμοποιούμε τον πίνακα colors με τον οποίο ο κάθε αριθμός συνδέεται με το χρώμα που αντιπροσωπεύει. Πιο συγκεκριμένα, το 0 είναι το κόκκινο, το 1 το πράσινο, το 2 το μπλε, το 3 το κίτρινο, το 4 και το 5 δεν υπάρχουν, το 6 το άσπρο και το 7 το μαύρο. Τα χρώματα στον πίνακα είναι γραμμένα στα αγγλικά, όπως και τα μηνύματα που εμφανίζουμε στην οθόνη του NXT τούβλου μας.

- static boolean *retry* = false;

Η μεταβλητή retry μας επιτρέπει να δίνουμε τη δυνατότητα στο ρομπότ μας, σε περίπτωση που προσπαθήσει να διαβάσει ένα βέλος αλλά οι τιμές των leftColor και rightColor είναι ίσες, να δοκιμάσει άλλη μία φορά.

- **static** File *filename* = **new** File("bark.wav");

Δηλώνουμε ένα αντικείμενο τύπου File, με το οποίο μπορούμε να συμπεριλάβουμε στον κώδικά μας ένα αρχείο. Στην περίπτωση μας, είναι ένα αρχείο ήχου, για να έχει τη δυνατότητα ο σκύλος μας να γαυγίζει. Το αρχείο αυτό, για να μπορεί να χρησιμοποιηθεί από το NXT ρομπότ μας, πρέπει να είναι τύπου wav, pcm, mono και 8 bits.

Στη μέθοδο main της κλάσης MyDog υπάρχουν οι εξής γραμμές κώδικα:

- *color*.setFloodlight(Color.*WHITE*);

Θέτουμε ότι το χρώμα από το λαμπάκι του αισθητήρα χρώματος θα έχει χρώμα άσπρο.

- *pilot* = **new** DifferentialPilot(4.32f, 7.9f, Motor.*A*, Motor.*B*, **true**);

Χρησιμοποιούμε τον constructor της κλάσης DifferentialPilot για να αρχικοποιήσουμε τον δικό μας pilot. Το πρώτο όρισμα είναι η διάμετρος των τροχών που πρέπει να είναι μετρημένη στην ίδια μονάδα μέτρησης με το δεύτερο όρισμα που είναι η απόσταση μεταξύ των τροχών. Η διάμετρος των τροχών βρίσκεται γραμμένη πάνω στα λάστιχα που συνοδεύουν το NXT. Στο τρίτο και τέταρτο όρισμα δηλώνουμε σε ποιες θύρες είναι συνδεδεμένοι ο αριστερός και ο δεξιός κινητήρας αντίστοιχα. Το τελευταίο όρισμα σημαίνει ότι όταν οι κινητήρες κινούνται προς τα μπροστά, το ρομπότ μας κινείται προς τα πίσω και το ανάποδο. Οπότε αν δεν υπήρχε αυτό το όρισμα όταν λέγαμε *pilot*.forward() αντί το ρομπότ μας να κινηθεί προς τα μπροστά όπως είναι το λογικό, θα κινούνταν προς τα πίσω και αυτό μπορεί να μας μπέρδευε κάποια στιγμή.

- *pilot*.setRotateSpeed(30);

Θέτουμε την ταχύτητα με την οποία θα κινείται ο pilot σε 30 μοίρες ανά δευτερόλεπτο.

- Behavior b1 = **new** MoveForward();

  Behavior b2 = **new** FindWallReadColor();

  Behavior b3 = **new** DoAction();

Με τις τρεις αυτές γραμμές δηλώνουμε ότι στο πρόγραμμά μας υπάρχουν τρεις διαφορετικές συμπεριφορές με ονόματα MoveForward, FindWallReadColor και DoAction. Κάθε μία από αυτές τις συμπεριφορές έχει τη δική της κλάση και περιγράφεται αναλυτικά παρακάτω.

- Behavior[] behav = { b1, b2, b3 };

Δημιουργούμε έναν πίνακα συμπεριφορών με όνομα behav στον οποίο εισάγουμε τις συμπεριφορές με αύξουσα σειρά προτεραιότητας. Αυτό σημαίνει ότι η συμπεριφορά b1 έχει τη μικρότερη προτεραιότητα και η b3 την πιο μεγάλη.

- Arbitrator arbi = **new** Arbitrator(behav);

Δημιουργούμε έναν καινούριο Arbitrator με όνομα arbi που δέχεται ως όρισμα των πίνακα με τις προτεραιότητες των συμπεριφορών που δημιουργήσαμε παραπάνω, τον behav. Αυτός είναι ο διαιτητής του προγράμματός μας, ο οποίος είναι υπεύθυνος για να γίνεται ενεργή κάθε φορά η συμπεριφορά που πρέπει. Όπως ήδη αναφέραμε στο προηγούμενο κεφάλαιο, η κάθε συμπεριφορά αποτελείται από τρεις μεθόδους: την takeControl() η οποία καθορίζει πότε πρέπει η συμπεριφορά αυτή να πάρει τον

έλεγχο, την action() η οποία εκτελείται όταν η συμπεριφορά έχει τον έλεγχο και την suppress() η οποία καλείται όταν μία άλλη συμπεριφορά πάρει τον έλεγχο για να σταματήσει την μέθοδο action() που εκτελείται και να αποθηκεύσει τυχόν αλλαγές σε μεταβλητές που θέλουμε. Η δουλειά του διαιτητή μας, είναι να ελέγχει αν πληρούνται οι προϋποθέσεις της μεθόδου takeControl() με σειρά φθίνουσα σειρά προτεραιότητας (αυτή με τη μεγαλύτερη προτεραιότητα πρώτα) και να δίνει τον έλεγχο στην συμπεριφορά που πρέπει.

- arbi.start();

Με αυτή τη γραμμή του κώδικα κάνουμε τον διαιτητή να ξεκινήσει να κάνει τους απαραίτητους ελέγχους και το πρόγραμμά μας ξεκινάει.

4.2.3 ΜΕΘΟΔΟΙ

Παρακάτω θα δούμε τις τρεις μεθόδους οι οποίες μας βοηθούν να είναι πιο συμμαζεμένος ο κώδικάς μας μέσα στις κλάσεις των συμπεριφορών.

- **public static void** CountDistance_Turn()

Η μέθοδος αυτή καλείται όταν το ρομπότ μας εντοπίσει κάποιο εμπόδιο με άσπρο χρώμα ή όταν το ρομπότ μας προσπαθήσει να διαβάσει ένα βέλος δύο φορές και αποτύχει και τις δύο. Τότε στην οθόνη του NXT τούβλου μας, θα εμφανιστεί το μήνυμα «Get distance» και ο σκύλος μας θα στρίψει το κεφάλι του αριστερά. Με τον αισθητήρα υπερήχων θα μετρήσει την απόσταση μέχρι το πρώτο εμπόδιο και θα εμφανίσει στην οθόνη μας την μέτρησή του (πχ. LEFT: 24). Έπειτα θα στρίψει το κεφάλι δεξιά και θα επαναλάβει τη διαδικασία τυπώνοντας μας στην οθόνη το αντίστοιχο μήνυμα (πχ RIGHT: 10). Μετά το ρομπότ γυρίζει ξανά το κεφάλι του μπροστά και σβήνει τα προηγούμενα μηνύματα από την οθόνη. Έπειτα, ελέγχει αν η απόσταση στα αριστερά είναι μεγαλύτερη ή ίση σε σχέση με αυτή στα δεξιά. Αν ναι στρίβει προς τα αριστερά ενώ αν όχι προς τα δεξιά, ώστε να μπορεί να διανύσει μεγαλύτερη απόσταση χωρίς να συναντήσει εμπόδιο. Επιπλέον, τυπώνει το αντίστοιχο μήνυμα στην οθόνη. Τέλος, θέτει τη μεταβλητή mycolor ίση με -1 που θα κάνει τον διαιτητή μας να δώσει τον έλεγχο στην συμπεριφορά b1. Η μέθοδος αποτελείται από τις παρακάτω γραμμές κώδικα:

Πίνακας 8 "Κώδικας μεθόδου CountDistance_Turn"

```
double left;
double right;
Motor.C.setSpeed(100);
Motor.C.rotate(100);
LCD.clear(7);
LCD.drawString("Get distance", 0, 1);
left = MyDog.distance.getDistance();
LCD.drawString("LEFT: " + Double.toString(left), 0, 5);
```

```
Motor.C.rotate(-200);
right = MyDog.distance.getDistance();
LCD.drawString("RIGHT: " + Double.toString(right), 0, 6);
Motor.C.rotate(100);
LCD.clear();
if (left >= right) {
    LCD.drawString("turning LEFT!", 0, 4);
        MyDog.pilot.rotate(-90);
         }
else {
    LCD.drawString("turning RIGHT!", 0, 4);
        MyDog.pilot.rotate(90);
         }
MyDog.mycolor = -1;
LCD.clear();
```

Πιο αναλυτικά, στην 1η και τη 2η γραμμή δηλώνουμε δύο μεταβλητές τύπου double στις οποίες θα αποθηκεύουμε την απόσταση που μετρήθηκε από τον αισθητήρα υπερήχων. Η μεταβλητή left θα αποθηκεύει την απόσταση προς τα αριστερά και η right την απόσταση προς τα δεξιά. Η 3η γραμμή θέτει την ταχύτητα περιστροφής του κινητήρα που είναι συνδεδεμένος στη θύρα C, δηλαδή τον κινητήρα του κεφαλιού σε 100 μοίρες ανά δευτερόλεπτο. Η 4η γραμμή περιστρέφει τον κινητήρα C 100 μοίρες, το οποίο πρακτικά σημαίνει ότι το κεφάλι του σκύλου μας θα περιστραφεί 90 μοίρες στα αριστερά. Στην 5η γραμμή σβήνουμε την έβδομή σειρά από την οθόνη μας και στην 6η γράφουμε στη δεύτερη σειρά της οθόνης μας το μήνυμα «Get distance». Στην 7η ο αισθητήρας υπερήχων μετράει την απόσταση από το πιο κοντινό εμπόδιο και την αποθηκεύει στη μεταβλητή left και στην 8η εμφανίζουμε στην έκτη σειρά της οθόνης μας το μήνυμα «LEFT: “απόσταση που μετρήθηκε”». Η 9η γραμμή περιστρέφει τον κινητήρα C -200 μοίρες, το οποίο πρακτικά σημαίνει ότι το κεφάλι του σκύλου μας θα περιστραφεί 180 μοίρες στα δεξιά, οπότε τώρα κοιτάζει προς τα δεξιά. Στην 10η ο μετράμε την απόσταση από το πιο κοντινό εμπόδιο και την αποθηκεύουμε το αποτέλεσμα στη μεταβλητή right και στην 11η εμφανίζουμε στην έβδομη σειρά της οθόνης μας το μήνυμα «RIGHT: “απόσταση που μετρήθηκε”». Η 12η περιστρέφει τον κινητήρα C 100 μοίρες, το οποίο πρακτικά σημαίνει ότι το κεφάλι του σκύλου μας θα περιστραφεί 90 μοίρες στα αριστερά, άρα τώρα θα κοιτάζει ευθεία, ενώ η 13η σβήνει όλες τις πληροφορίες που είχαμε τυπώσει στην οθόνη μας. Στη γραμμή 14 ελέγχουμε αν η μεταβλητή left έχει μεγαλύτερή ή ίση τιμή από την right και αν αυτό ισχύει εκτελούνται οι γραμμές 15 (τυπώνει στην οθόνη μας, στη σειρά πέντε το μήνυμα «turning LEFT!») και 16(με τη χρήση του pilot κάνουμε το ρομπότ μας να περιστραφεί επιτόπου 90 μοίρες προς τα αριστερά). Αν όχι εκτελείται ο κώδικας που υπάρχει στο else δηλαδή στις γραμμές 19 (τυπώνει στην οθόνη μας, στη σειρά πέντε το μήνυμα «turning RIGHT!») και 20 (κάνουμε το ρομπότ μας να περιστραφεί επιτόπου 90 μοίρες προς τα δεξιά). Στη 22η γραμμή θέτουμε την τιμή

της μεταβλητής mycolor ίση με -1 το οποίο θα κάνει τον διαιτητή arbi να δώσει τον έλεγχο σε κάποια άλλη συμπεριφορά, την b2 αν υπάρχει εμπόδιο σε απόσταση ίση ή μικρότερη των 6 εκατοστών ή αλλιώς την b1. Τέλος, με την γραμμή 23 σβήνουμε όλα τα μηνύματα από την οθόνη μας.

- public static void Turn(int a, int b)

Η μέθοδος αυτή καλείται όταν το ρομπότ μας έχει διαβάσει κάποιο βέλος και θέλει να στρίψει ανάλογα με τη φορά του βέλους. Το a παίρνει την τιμή της μεταβλητής leftColor ενώ το b της rightColor. Το ρομπότ μας εμφανίζει στην οθόνη του αν είναι βέλος προς τα δεξιά ή προς τα αριστερά ανάλογα με τις τιμές που δέχτηκε ως ορίσματα η μέθοδος. Αν η πρώτη είναι πιο μεγάλη, το βέλος είναι αριστερό βέλος, ενώ αν η δεύτερη είναι πιο μεγάλη είναι δεξιό βέλος. Έπειτα εμφανίζεται στην οθόνη το αντίστοιχο μήνυμα που μας ενημερώνει για την φορά του βέλους. Μετά το ρομπότ μας στρίβει προς την κατεύθυνση. Τέλος, η μεταβλητή mycolor παίρνει την τιμή -1 με αποτέλεσμα να αναλάβει τον έλεγχο η συμπεριφορά b2 αν υπάρχει εμπόδιο σε απόσταση μικρότερη των 6 εκατοστών ή διαφορετικά η b1. Ακολουθεί ο κώδικας της μεθόδου:

Πίνακας 9 "Κώδικας μεθόδου Turn(int a, int b)"

```
int left;
int right;
left = a;
right = b;
LCD.clear();
if (left > right) {
        LCD.drawString("LEFT arrow!!!", 0, 3);
        LCD.drawString("turning LEFT!", 0, 4);
        MyDog.pilot.rotate(-90);
         }
else {
        LCD.drawString("RIGHT arrow!!!", 0, 3);
        LCD.drawString("turning RIGHT!", 0, 4);
        MyDog.pilot.rotate(90);
         }
        MyDog.mycolor = -1;
```

Στις 2 πρώτες γραμμές της μεθόδου αυτής δηλώνουμε δύο μεταβλητές (left και right) τύπου int οι οποίες θα πάρουν τις τιμές των ορισμάτων a και b αντίστοιχα που είναι ο τιμές με τις οποίες καλούμε την μέθοδο. Η 5η γραμμή σβήνει ότι ήταν γραμμένο στην οθόνη μας. Στην 6η γραμμή ελέγχουμε αν το left είναι μεγαλύτερο από το right. Σε περίπτωση που είναι θα εκτελεστούν οι 7η (εμφανίζει στην τέταρτη γραμμή του πίνακα το μήνυμα «LEFT arrow!!!»), 8η (εμφανίζει στην πέμπτη γραμμή του πίνακα το μήνυμα «turning LEFT!»), και 9η γραμμή του κώδικα (χρησιμοποιείται ο pilot για να

στρίψουμε το ρομπότ μας επιτόπου προς τα αριστερά 90 μοίρες). Σε περίπτωση που δεν είναι θα εκτελεστούν οι γραμμές 12 (εμφανίζει στην πέμπτη γραμμή του πίνακα το μήνυμα «turning RIGHT!»), 13 (εμφανίζει στην πέμπτη γραμμή του πίνακα το μήνυμα «turning RIGHT!»), και 14 (κάνουμε το ρομπότ μας να στρίψει επιτόπου δεξιά 90 μοίρες). Τέλος, στη 15η γραμμή θέτουμε την τιμή της mycolor ίση με -1 το οποίο θα κάνει τον διαιτητή arbi να δώσει τον έλεγχο σε κάποια άλλη συμπεριφορά.

- public static void ReadArrow()

Αυτή η μέθοδος καλείται όταν ο σκύλος μας εντοπίσει κάποιο εμπόδιο με χρώμα διαφορετικό από το άσπρο και το κόκκινο. Τότε προσπαθεί να διαβάσει την κατεύθυνση του βέλους που υποθέτει ότι υπάρχει πάνω στο εμπόδιο. Σε περίπτωση που διαβάσει το βέλος με επιτυχία στρίβει προς την κατεύθυνση που του δείχνει. Σε περίπτωση που αποτύχει προσπαθεί άλλη μία φορά και αν πάλι δεν τα καταφέρει υποθέτει ότι τελικά δεν υπάρχει βέλος στο εμπόδιο και καλεί την μέθοδο CountDistance_Turn() που περιγράψαμε πιο πάνω. Τα βέλη είναι όλα χρωματισμένα με δύο διαφορετικά χρώματα Τα χρώματα που χρησιμοποιούνται είναι το πράσινο, το μπλε και το κίτρινο. Κάθε χρώμα έχει και μία τιμή (πράσινο = 2, μπλε = 3, κίτρινο = 4). Όλα τα βέλη είναι σχεδιασμένα έτσι ώστε το χρώμα με τη μεγαλύτερη τιμή να βρίσκεται στη μύτη του βέλους. Αυτό πρακτικά σημαίνει ότι αν έχουμε ένα πράσινο-κίτιρνο βέλος στη μύτη θα πρέπει να είναι το κίτρινο, αν έχουμε ένα πράσινο-μπλε, στη μύτη θα είναι το μπλε, ενώ αν έχουμε ένα μπλε-κίτρινο η μύτη θα είναι στο κίτρινο. Ο σκύλος μας μπορεί να διαβάσει ποιο χρώμα βρίσκεται στα δεξιά του και ποιο στα αριστερά του γυρίζοντας το κεφάλι του. Έτσι, ο σκύλος μας στρίβει το κεφάλι του αριστερά και έπειτα ξεκινάει το στρίβει αργά προς τα δεξιά. Κατά τη διάρκεια της περιστροφής προς τα δεξιά ο αισθητήρας χρώματος παίρνει 8 δείγματα από το χρώμα που αντιλαμβάνεται. Παίρνουμε πολλά δείγματα για να είμαστε σίγουροι για το αποτέλεσμα, επειδή λόγω της περιστροφής του κεφαλιού, η απόσταση του αισθητήρα χρώματος από το εμπόδιο αυξομειώνεται. Αν το άθροισμα των πρώτων 4 θέσεων του πίνακα είναι μεγαλύτερο από αυτό των 4 επόμενων, τότε το βέλος μας δείχνει προς τα αριστερά ενώ αν το πρώτο άθροισμα είναι μικρότερο από το δεύτερο τότε το βέλος δείχνει δεξιά. Έπειτα, το ρομπότ μας στρίβει προς τη μεριά που δείχνει το βέλος κάνοντας επιτόπου στροφή χρησιμοποιώντας τον pilot. Ο κώδικας της μεθόδου είναι ο εξής:

Πίνακας 10"Κώδικας μεθόδου ReadArrow()"

```
int [] colorarray = new int[8];
int leftColor = 0;
int rightColor = 0;
Motor.C.setSpeed(20);
Motor.C.rotate(28);
LCD.drawString("something found!", 0, 0);
LCD.drawString("try read arrow...", 0, 3);
        for (int i = 0; i < 8; i++) {
```

```
        clr = MyDog.color.getColor();
        if (clr.getColor() == 1)
                colorarray[i] = clr.getColor();
        else if (clr.getColor() == 2)
                colorarray[i] = 100;
        else if (clr.getColor() == 3)
                colorarray[i] = 10000;
        else
                colorarray[i] = 0;
        Motor.C.rotate(-7);
        }
LCD.clear();
leftColor = colorarray[0] + colorarray[1] + colorarray[2] + colorarray[3];
rightColor = colorarray[4] + colorarray[5] + colorarray[6] + colorarray[7];
LCD.drawInt(leftColor, 6, 6);
LCD.drawInt(rightColor, 7, 7);
Motor.C.rotate(28);
if (leftColor == rightColor) {
    if (MyDog.retry == false) {
        MyDog.retry = true;
        MyDog.ReadArrow();
        } else {
        MyDog.retry = false;
        LCD.drawString("No arrow found!", 0, 0);
        MyDog.CountDistance_Turn();
        }
}else
         MyDog.Turn(leftColor, rightColor);
```

Στην 1η γραμμή δηλώνουμε πίνακα τύπου int με όνομα colorarray που αποτελείται από 8 θέσεις. Τον πίνακα αυτό θα τον χρησιμοποιήσουμε για να αποθηκεύουμε τις τιμές που διαβάζει ο αισθητήρας χρώματος την ώρα που προσπαθεί να καταλάβει προς ποια κατεύθυνση δείχνει το βέλος. Όπως ήδη αναφέραμε, παίρνουμε πολλά δείγματα για να είμαστε σίγουροι για το αποτέλεσμα, επειδή λόγω της περιστροφής του κεφαλιού, η απόσταση του αισθητήρα χρώματος από το εμπόδιο αυξομειώνεται και υπάρχουν κάποιες πιθανότητες λάθους. Στις γραμμές 2 και 3 δηλώνουμε δύο μεταβλητές για να αποθηκεύσουμε τα αθροίσματα των θέσεων του πίνακα colorarray τα οποία βοηθούν στο να καταλάβουμε την κατεύθυνση στην οποία δείχνει το βέλος. Επειδή την ώρα που διαβάζουμε το βέλος, το κεφάλι στρίβει από αριστερά προς τα δεξιά, το άθροισμα των πρώτων τεσσάρων μετρήσεων θα είναι η τιμή της μεταβλητής leftColor, ενώ το άθροισμα των τεσσάρων τελευταίων η τιμή της μεταβλητής rightColor. Το βέλος είναι σχεδιασμένο ώστε να δείχνει προς το μεγαλύτερο άθροισμα. Στην 4η γραμμή θέτουμε την ταχύτητα του κινητήρα C ίση με

20 μοίρες το δευτερόλεπτο. Στη 5η γραμμή περιστρέφουμε τον κινητήρα C 28 μοίρες προς τα αριστερά. Η 3η γραμμή τυπώνει στην οθόνη μας το μήνυμα «something found!» και στην 4η το μήνυμα «try read arrow...». Στην 6η γραμμή ξεκινάει ένα loop δηλαδή ένας βρόγχος επανάληψης. Το loop αυτό επαναλαμβάνεται 8 φορές και κάθε φορά χρησιμοποιώντας τον αισθητήρα χρώματος διαβάζουμε το χρώμα που υπάρχει μπροστά του και στρίβουμε το κεφάλι 7 μοίρες προς τα δεξιά. Κάθε φορά που κάνει μία μέτρηση ανάλογα με το χρώμα αποθηκεύει και έναν αριθμό στον πίνακα (για κίτρινο την τιμή 1000, για πράσινο 100 και για μπλε 10). Αφού τελειώσει το loop, σβήνουμε ότι υπήρχε γραμμένο στην οθόνη μας. Στην 21η και 22η γραμμή δίνουμε τιμές στις μεταβλητές leftColor και rightColor. Στις γραμμές 23 και 24 εμφανίζουμε στην οθόνη μας τις τιμές που έχουν τελικά οι μεταβλητές leftColor και rightColor. Στην 25η γραμμή στρίβουμε το κεφάλι του σκύλου 28 μοίρες προς τα αριστερά με αποτέλεσμα να κοιτάζει ευθεία. Στη γραμμή 26 με ένα ελέγχουμε αν οι δύο μεταβλητές (leftColor και rightColor) έχουν την ίδια τιμή. Αυτό σημαίνει ότι το ρομπότ μας δεν μπόρεσε να διαβάσει το βέλος που μπορεί να συμβεί αν έχουμε ένα μονόχρωμο εμπόδιο, για παράδειγμα ένα κίτρινο εμπόδιο ή αν την ώρα που ο ξεκινάει το loop αφαιρέσουμε το εμπόδιο μπροστά από το σκύλο μας. Σε περίπτωση που οι μεταβλητές έχουν την ίδια τιμή και είναι η πρώτη προσπάθεια για αναγνώριση του βέλους (δηλαδή αν η μεταβλητή retry είναι false) εκτελούνται οι γραμμές 28 (θέτει τη μεταβλητή retry ίση με true) και 29 (καλεί ξανά την μέθοδο ReadArrow()). Αν έχει ήδη προσπαθήσει μία φορά αποτυχημένα εκτελούνται οι γραμμές 31 (κάνει τη μεταβλητή retry false), 32 (εμφανίζει στην οθόνη στην πρώτη σειρά το μήνυμα « No arrow found! ») και 33 (αφού δεν υπάρχει βέλος, καλεί τη μέθοδο *CountDistance_Turn*() που είδαμε πιο πάνω για να στρίψει ανάλογα με την απόσταση). Τέλος, αν δεν έχουν την ίδια τιμή στη γραμμή 36 καλείται η μέθοδος Turn(leftColor, rightColor) την οποία είδαμε προηγουμένως.

4.2.4 ΣΥΜΠΕΡΙΦΟΡΕΣ

Ο κώδικας μας, όπως ήδη αναφέραμε έχει τρεις συμπεριφορές τις οποίες ήρθε η ώρα να δούμε αναλυτικά. Κάθε συμπεριφορά αποτελεί μία ξεχωριστή κλάση η οποία υλοποιεί (δηλαδή στη γλώσσα της Java κάνει implement) μία συμπεριφορά δηλαδή ένα Behavior. Και οι τρεις κλάσεις αποτελούνται υποχρεωτικά και από τις τρεις μεθόδους που υπάρχουν στην κλάση Behavior ακόμη και αν δεν χρειάζονται. Μία τέτοια περίπτωση αποτελεί πάντα η μέθοδος suppress() στην συμπεριφορά με την μεγαλύτερη προτεραιότητα. Αυτό συμβαίνει επειδή δεν υπάρχει συμπεριφορά με πιο μεγάλη προτεραιότητα για να γίνει ενεργή και να χρειαστεί να σταματήσει η μέθοδος action() που εκτελείται με την κλήση της μεθόδου suppress(). Συνήθως την αφήνουμε κενή αφού δεν θα κληθεί ποτέ, αλλά πρέπει να υπάρχει για να δουλέψει ο κώδικάς μας.

- MoveForward

Η πρώτη συμπεριφορά είναι η MoveForward. Η συμπεριφορά αυτή είναι η b1 δηλαδή αυτή που έχει και την μικρότερη προτεραιότητα. Αυτό σημαίνει ότι είναι η

συμπεριφορά που έχει συνέχεια τον έλεγχο μέχρι η μέθοδος takeControl() κάποιας συμπεριφοράς να μας επιστρέψει την τιμή true και να πάρει αυτή τον έλεγχο. Κάνει το ρομπότ σκύλο μας να κινείται συνέχεια ευθεία μέχρι κάποια συμπεριφορά με μεγαλύτερη προτεραιότητα να πάρει τον έλεγχο. Ακολουθεί ο κώδικας της συμπεριφοράς αυτής:

Πίνακας 11"Κώδικας συμπεριφοράς MoveForward "

```
boolean suppressed = false;
public boolean takeControl() {
    return true;
}
public void suppress() {
    suppressed = true;
}
public void action() {
    LCD.clear();
    suppressed = false;
    MyDog.pilot.setTravelSpeed(5);
    MyDog.pilot.forward();
    while (!suppressed) {
        Thread.yield();
    }
    MyDog.pilot.stop();
}
```

Στην 1η γραμμή του κώδικα της συμπεριφοράς αυτής δηλώνουμε μία μεταβλητή τύπου boolean με όνομα suppressed ίση με false την οποία θα χρησιμοποιήσουμε για την μέθοδο suppress(). Στη 2η γραμμή βρίσκεται η μέθοδος takeControl() η οποία αποτελείται από τη γραμμή 3 που μας επιστρέφει πάντα true, δηλαδή θα παίρνει πάντα τον έλεγχο. Στην 5η γραμμή βρίσκεται η μέθοδος suppress() που αποτελείται από τη γραμμή 6 (κάνει την τιμή της μεταβλητής suppressed ίση με false). Στη γραμμή 8 είναι η μέθοδος action() που αποτελείται από τις γραμμές 9 (σβήνει ότι υπήρχε γραμμένο στην οθόνη μας), 10 (θέτει τη μεταβλητή suppressed ίση με true), 11 (δηλώνουμε ότι η ταχύτητα του σκύλου μας θα είναι 5 μοίρες ανά δευτερόλεπτο), 12 (κάνουμε το ρομπότ μας να κινείται σε ευθεία γραμμή), 13 και 14 (δηλώνουμε ότι αυτή η δράση θα γίνεται συνέχεια μέχρι να κληθεί η μέθοδος suppress() και τέλος στην 16η γραμμή που εκτελείται μόλις κληθεί η μέθοδος suppress() σταματάμε την κίνηση).

- FindWallReadColor

Η δεύτερη συμπεριφορά είναι η FindWallReadColor. Αυτή παίρνει τον έλεγχο όταν βρεθεί κάποιο εμπόδιο σε απόσταση μικρότερη ή ίση των 6 εκατοστών από τον

αισθητήρα υπερήχων. Όταν συμβεί αυτό ο αισθητήρας χρώματος διαβάζει το χρώμα του εμποδίου που εντοπίστηκε και αποθηκεύει την τιμή του στη μεταβλητή mycolor.

Πίνακας 12 "Κώδικας συμπεριφοράς FindWallReadArrow"

```
boolean suppressed = false;
public boolean takeControl() {
    return MyDog.distance.getDistance() <= 6;
}
public void suppress() {
    suppressed = true;
}
public void action() {
    suppressed = false;
    LCD.clear();
    MyDog.clr = MyDog.color.getColor();
    LCD.drawString("    COLOR:  ", 0, 1);
    LCD.drawString(MyDog.colors[MyDog.clr.getColor()], 7, 1);
    while (!suppressed) {
        MyDog.mycolor = MyDog.clr.getColor();
    }
}
```

Στη γραμμή 1 δηλώνουμε την μεταβλητή τύπου boolean με όνομα suppressed ίση με false την οποία θα χρησιμοποιήσουμε για την μέθοδο suppress(). Στη 2η γραμμή βρίσκεται η μέθοδος takeControl() η οποία αποτελείται από τη γραμμή 3 που μας επιστρέφει true και δίνει τον έλεγχο σε αυτή τη συμπεριφορά όταν η απόσταση από τον αισθητήρα υπερήχων είναι μικρότερη ή ίση με 6 εκατοστά. Μόλις αυτή η συμπεριφορά πάρει των έλεγχο καλείται η μέθοδος suppress() της προηγούμενης συμπεριφοράς σταματώντας την κίνηση του ρομπότ και έπειτα καλείται η action() αυτής της συμπεριφοράς. Στη γραμμή 5 βρίσκεται η μέθοδος suppress() η οποία αποτελείται από τη γραμμή 6 και θέτει την τιμή της μεταβλητής suppressed ίση με false. Στην 8η γραμμή υπάρχει η μέθοδος action() που περιέχει τις γραμμές κώδικα 9 (θέτει τη μεταβλητή suppressed ίση με false),10 (σβήνει ότι ήταν γραμμένο στην οθόνη μας),11 (διαβάζουμε το χρώμα που υπάρχει στο εμπόδιο),12 και13 (εμφανίζουμε στην οθόνη το χρώμα που διαβάσαμε),14 και 15 (όσο δεν έχει κληθεί κάποια μέθοδος με μεγαλύτερη προτεραιότητα διαβάζουμε το χρώμα και το αποθηκεύουμε στ μεταβλητή mycolor).

- DoAction

Η τελευταία συμπεριφορά είναι η DoAction η οποία έχει τη μεγαλύτερη προτεραιότητα. Παίρνει τον έλεγχο μόλις η μεταβλητή mycolor πάρει κάποια τιμή διάφορη του -1, που σημαίνει ότι ο αισθητήρας χρώματος έχει διαβάσει κάποιο χρώμα. όταν πάρει τον έλεγχο, αντιδράει ανάλογα με το χρώμα που διαβάσαμε. Σε

περίπτωση που είναι άσπρο καλείται η μέθοδος *CountDistance_Turn*(), αν είναι οποιοδήποτε άλλο χρώμα καλείται η μέθοδος *ReadArrow*() ενώ αν είναι κόκκινο σταματάει το πρόγραμμά μας.

Πίνακας 13 "Κώδικας συμπεριφοράς DoAction"

```
public boolean takeControl() {
    return MyDog.mycolor != -1;
}
public void suppress() {
}
public void action() {
    LCD.clear();
    if (MyDog.mycolor == 6) {
        LCD.drawString("WHITE wall found", 0, 0);
        MyDog.CountDistance_Turn();
    } else if (MyDog.mycolor == 0) {
        LCD.drawString("THE END!", 4, 5);
        Sound.playSample(MyDog.filename, 100);
        for (int i = 0; i < 3; i++) {
            MyDog.color.setFloodlight(Color.BLUE);
            MyDog.color.setFloodlight(Color.RED);
            MyDog.color.setFloodlight(Color.GREEN);
        }
        NXT.shutDown();
    } else {
        MyDog.mycolor = -1;
        MyDog.ReadArrow();
    }
}
```

Στην 1η γραμμή έχουμε τη μέθοδο takeControl() η οποία περιέχει μία γραμμή κώδικα, την 2η που ορίζει ότι η συμπεριφορά αυτή θα πάρει τον έλεγχο μόλις η μεταβλητή mycolor πάρει κάποια τιμή διαφορετική από το -1. Στην 4η γραμμή υπάρχει η μέθοδος suppress() η οποία είναι κενή μιας και δεν θα κλιθεί ποτέ αφού η συμπεριφορά DoAction είναι η συμπεριφορά με τη μεγαλύτερη προτεραιότητα. Στην 6η γραμμή βρίσκεται η μέθοδος action() που καλείται όταν αυτή η συμπεριφορά πάρει τον έλεγχο αμέσως μετά την μέθοδο suppress() της συμπεριφοράς που είχε πριν τον έλεγχο. Η μέθοδος action() αποτελείται από τις γραμμές 7 έως 22. Η 7η σβήνει ότι υπήρχε στην οθόνη μας. Η 8η ελέγχει αν το χρώμα είναι άσπρο και αν είναι εκτελούνται οι γραμμές 9 (εμφανίζεται στην οθόνη μας το μήνυμα «WHITE wall found») και 10 (καλεί τη μέθοδο *CountDistance_Turn*()). Η 11η ελέγχει αν το χρώμα είναι κόκκινο και αν είναι εκτελούνται οι γραμμές 12 (εμφανίζει το μήνυμα «THE END!» στην οθόνη μας), 13 (κάνει το σκύλο μας να γαυγίσει αναπαράγοντας το

αρχείο ήχου filename), 14 με 17 (εναλλάσσονται τρεις φορές τα χρώματα μπλε, κόκκινο, πράσινο στο λαμπάκι του αισθητήρα χρώματος) και 19 που απενεργοποιεί το NXT τούβλο. Στην 21η και 22η γραμμή αν το χρώμα δεν είναι ούτε άσπρο, ούτε κόκκινο θέτουμε τη μεταβλητή mycolor ίση με -1 και καλούμε τη μέθοδο ReadArrow().

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο αυτό είδαμε αναλυτικά το πρόγραμμα το οποίο γράψαμε. Αρχικά αναφέραμε ότι χρησιμοποιώντας τα τουβλάκια της Lego φτιάξαμε ένα ρομπότ το οποίο έχει τη μορφή σκύλου. Μετά, είδαμε το σενάριο του προγράμματος μας και τέλος τον κώδικα μαζί με αναλυτικές επεξηγήσεις.

ΣΥΜΠΕΡΑΜΑΤΑ

Η ρομποτική είναι ένας δημιουργικός κλάδος με πολύ ενδιαφέρον. Είναι γεγονός ότι οι άνθρωποι, από τα πολύ παλιά χρόνια, είχαν στο μυαλό τους τη δημιουργία μηχανημάτων για να αυτοματοποιήσουν διάφορες εργασίες και να ζούνε πιο ξεκούραστα. Αργότερα κάποιοι οραματίστηκαν ανθρωπόμορφες μηχανές που θα μπορούσαν να εκτελούν όποια εντολή τους δίνουμε. Για να μπορέσει όμως να ελεγχθεί η κατάσταση φαντάστηκαν ότι οι μηχανές αυτές δε θα πρέπει σε καμία περίπτωση να βλάπτουν την ανθρώπινη ύπαρξη αλλά αντιθέτως θα πρέπει να την προστατεύουν. Αργότερα οι μηχανές αυτές ονομάστηκαν ρομπότ. Στις μέρες μας τα ρομπότ βρίσκονται στην καθημερινότητά μας. Υπάρχουν πολλά διαφορετικά είδη κα με λίγη υπερβολή θα μπορούσαμε να πούμε ότι οι δυνατότητές τους είναι απεριόριστες.

Η εταιρεία Lego μας δίνει τη μοναδική ευκαιρία να γίνουμε κι εμείς μέρος του υπέροχου κόσμου των ρομπότ με το εκπαιδευτικό της παιχνίδι το Lego Mindstorms NXT. Μπορούμε να του δώσουμε όποια μορφή θέλουμε, να συνδυάσουμε περισσότερα από ένα NXT τούβλα για να φτιάξουμε ένα μεγαλύτερο με πιο πολλές δυνατότητες ρομπότ αλλά και να αγοράσουμε καινούριους αισθητήρες. Πέρα όμως από την κατασκευή, μπορούμε και να το προγραμματίσουμε με πολλούς τρόπους, για να κάνει ότι εμείς θέλουμε, με μεγαλύτερο περιορισμό την έλλειψη φαντασίας.

Μερικές ιδέες για ενδιαφέροντα προγράμματα, τα οποία μπορούν να υλοποιηθούν με τη χρήση ενός Lego Mindstorm NXT, είναι:

- η αναγνώριση αντικειμένων και προσώπων με τη χρήση του αισθητήρα χρώματος, ο οποίος θα παίρνει δείγματα από όλη την εικόνα

- η δημιουργία ενός robot "βοηθός service" με τη χρήση των κλάσεων για navigation και δημιουργία χαρτών. Το ρομπότ αυτό θα βρίσκεται σταματημένο σε ένα σημείο και η κατασκευή του θα μας επιτρέπει να στηρίζουμε πάνω του διάφορα αντικείμενα π.χ. ένα ποτήρι νερό. Το ρομπότ θα είναι σε θέση να αντιλαμβάνεται ότι υπάρχει πάνω του κάποιο αντικείμενο χρησιμοποιώντας τους αισθητήρες επαφής. Θα μπορεί να πηγαίνει σε κάποια προκαθορισμένη διαδρομή, όπου και θα σταματάει, περιμένοντας κάποιος να αφαιρέσει το αντικείμενο από πάνω του. Έπειτα θα επιστρέφει στην αρχική του θέση, περιμένοντας να παραλάβει κάποιο νέο αντικείμενο

- χρήση Bluetooth και σύνδεση με android, που μας δίνει τη δυνατότητα να ελέγξουμε το ρομπότ μας από μακριά (remote controler)

- στατικό ρομπότ που θα χρησιμοποιεί τους κινητήρες, για να μπορεί να κουνάει κάποιο στυλό ή μολύβι, για να γράφει

- επίλυση διάφορων παιχνιδιών, όπως sudoku, puzzle και τον κύβο του Rubik

Επίσης, η δημιουργία μεγαλύτερων ρομπότ με περισσότερες δυνατότητες αποτελεί μία ακόμα ελκυστική ιδέα. Μεγάλη πρόκληση μπορεί να αποτελέσει η δημιουργία μιας ποδοσφαιρικής ομάδας με ρομπότ Lego και η συμμετοχή του ΑΤΕΙ Θεσσαλονίκης στο Robocup, παγκόσμιο πρωτάθλημα ποδοσφαίρου για ρομπότ.

Στην παρούσα πτυχιακή ασχοληθήκαμε με τον προγραμματισμό σε Java και το firmware LeJOS, όμως αξίζει να αναφέρουμε ότι το ρομπότ της Lego μπορεί να λειτουργήσει και με το πρόγραμμα Matlab, το οποίο μας προσφέρει απίστευτα πολλές δυνατότητες και ευκολίες, όπως για παράδειγμα το Simulink, το οποίο μας δείχνει μία προσομοίωση του κώδικά μας.

Τελειώνοντας, θα ήθελα να αναφέρω ότι η πτυχιακή αυτή, ήταν η πιο ενδιαφέρουσα και δημιουργική εργασία που έχω κάνει μέχρι σήμερα. Εύχομαι πολλά άτομα ακόμη να ασχοληθούν με το συγκεκριμένο αντικείμενο και να νιώσουν τη χαρά της δημιουργίας όπως την ένιωσα εγώ.

ΑΝΑΦΟΡΕΣ

1. (n.d.). Retrieved Ιανουάριος 2012, from wikipedia org: http://el.wikipedia.org/wiki/%CE%A4%CE%AC%CE%BB%CF%89%CF%82_(%CE%BC%CF%85%CE%B8%CE%BF%CE%BB%CE%BF%CE%B3%CE%AF%CE%B1)

2. (n.d.). Retrieved Ιανουάριος 2012, from wikipedia org: http://el.wikipedia.org/wiki/%CE%A4%CF%81%CE%B5%CE%B9%CF%82_%CE%BD%CF%8C%CE%BC%CE%BF%CE%B9_%CF%84%CE%B7%CF%82_%CF%81%CE%BF%CE%BC%CF%80%CE%BF%CF%84%CE%B9%CE%BA%CE%AE%CF%82

3. (n.d.). Retrieved Φεβρουάριος 2012, from Lego Minstorms: http://mindstorms.lego.com/en-us/whatisnxt/default.aspx

4. (n.d.). Retrieved Φεβρουάριος 2012, from LeJOS: http://lejos.sourceforge.net/nxt/nxj/tutorial/ErrorHandlingAndDebugging/ErrorHandling_and_debugging.htm

5. (n.d.). Retrieved Ιανουάριος 2012, from LeJOS: http://lejos.sourceforge.net/nxt/nxj/tutorial/PC_GUI/PCGUITools.htm

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Βασιλειάδης, Χ. (2010), Προγραμματισμός Αυτόματου Ρομποτικού Συστήματος για την εύρεση κατάλληλου χώρου και πραγματοποίηση της ευθείας και αντίστροφης διαδικασίας στάθμευσης, Μεταπτυχιακή διπλωματική εργασία (Α.Π.Θ), Θεσσαλονίκη

[2] Νικολής, Α. (2010), Αυτόματο Ρομποτικό Σύστημα για τη Συλλογή και Επεξεργασία Δεδομένων, Μεταπτυχιακή διπλωματική εργασία (Α.Π.Θ), Θεσσαλονίκη

[3] Αρμενόπουλος, Γ., Λιάκος, Χ. (2010), Επίλυση προβλημάτων αυτόματου ελέγχου σε τετράτροχο όχημα χρησιμοποιώντας την πλατφόρμα LEGO MINDSTORMS NXT, Πτυχιακή εργασία (ΤΕΙ Σερρών), Σέρρες

[4] Αλεξανδρίδης, Ζ. (2010), Επίλυση του Προβλήματος Sudoku με Χρήση Ευφυών Τεχνικών από Εκπαιδευτικό Ρομπότ, Πτυχιακή εργασία (Πανεπιστήμιο Πατρών), Πάτρα

[5] Bagnall, B. (2007), Maximum LEGO NXT: Building Robots with Java Brains, Variant Press

[6] Alfonso, D., Ferrari, G., Ferrari, M. (2007), Building Robots with LEGO MINSTORMS NXT, Syngress Publishing Inc., Elsevier Inc., 30 Corporate Drive, Burlington

[7] (n.d.). Retrieved Φεβρουάριος 2012, from Java tools for LeJOS: https://cs.marlboro.edu/term/spring06/programming_workshop/lejos/lejos_2_1_0/JavaTools.README

[8] Retrieved Ιανουάριος 2012, from LeJOS: http://mindstorms.lego.com/en-us/whatisnxt/default.aspx

[9] Retrieved Ιανουάριος 2012, from LeJOS: http://lejos.sourceforge.net/nxt/nxj/tutorial/index.html

[10] Retrieved Ιανουάριος 2012, from LeJOS: http://lejos.sourceforge.net/nxt/nxj/tutorial/MotorTutorial/ControllingMotors.html

[11] Retrieved Ιανουάριος 2012, from LeJOS: http://lejos.sourceforge.net/nxt/nxj/tutorial/PC_GUI/PCGUITools.html

[12] Retrieved Ιανουάριος 2012, from LeJOS: http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/UsingEclipse.htm

[13] Retrieved Ιανουάριος 2012, from LeJOS: http://lejos.sourceforge.net/nxt/nxj/api/index.html

[14] Retrieved Ιανουάριος 2012, from LeJOS: http://lejos.sourceforge.net/tutorial/essential/errors/index.html

[15] Retrieved Φεβρουάριος 2012, from LeJOS error handling: http://books.google.gr/books?id=r-ADYbe9ANoC&pg=PA237&lpg=PA237&dq=lejos+error+handling&source=bl&ots=27am-cwhTW&sig=bWhlCYFXKNUgLx76RStdjPZYGfc&hl=el&sa=X&ei=ggp_T_zxC4jAtAav2tCuBA&ved=0CFMQ6AEwBQ#v=onepage&q=lejos%20error%20handling&f=false

[16] Retrieved Φεβρουάριος 2012, from LEGO: http://shop.lego.com/en-US/NXT-Intelligent-Brick-9841

[17] Retrieved Φεβρουάριος 2012, from ρομποτική με χρήση του πακέτου Lego Mindstorms NXT: http://users.sch.gr/kyrgeo/material/nxt/02-MindstormsNXT.pdf

[18] Retrieved Φεβρουάριος 2012, from Robot Sensors: http://www.servomagazine.com/media-files/727/Then%26Now-0707.pdf

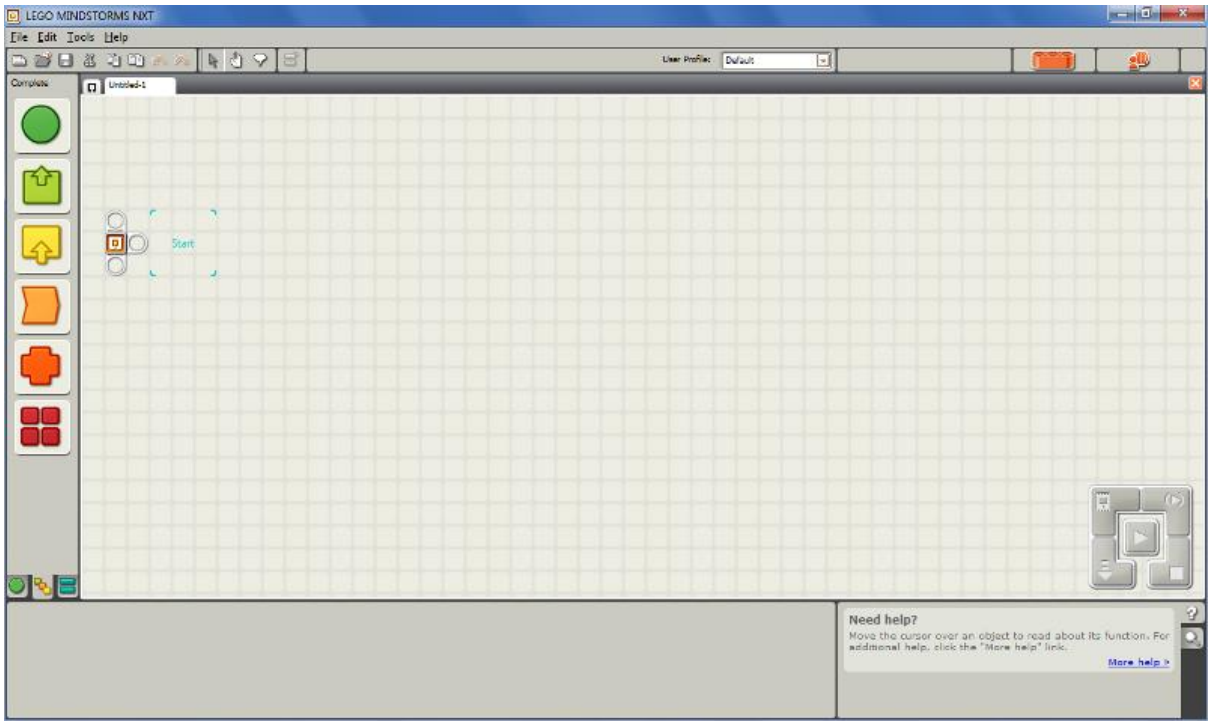
[19] Retrieved Φεβρουάριος 2012, from wikipedia: http://el.wikipedia.org/wiki/%CE%92%CE%B9%CE%BF%CE%BC%CE%B7%CF%87%CE%B1%CE%BD%CE%B9%CE%BA%CE%AC_%CF%81%CE%BF%CE%BC%CF%80%CF%8C%CF%84

ΠΑΡΑΡΤΗΜΑΤΑ

ΠΑΡΑΡΤΗΜΑ Ι – ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΣΧΕΔΙΑΣΗΣ NXT 2.0

Για την υλοποίηση ενός project, η LEGO, σε συνεργασία με τη National Instruments, δίνει την ευκαιρία στους χρήστες του NXT να σχεδιάσουν ένα πρόγραμμα, χωρίς να έχουν γνώσεις προγραμματισμού, βασιζόμενοι στο σύστημα των εικονικών οργάνων που έχει αναπτυχτεί στη σουίτα Labview.

Ο χρήστης προγραμματίζει το ρομπότ του με τη χρήση δομικών μονάδων (blocks) που αντιστοιχούν σε λειτουργίες κάποιου κινητήρα ή αισθητήρα.



Όπως φαίνεται στην πιο πάνω εικόνα, αριστερά μέσα στο ανοιχτό γκρι πλέγμα υπάρχει ένα πορτοκαλί σχήμα. Αυτό είναι το σημείο εκκίνησης από όπου μπορούν να ξεκινήσουν μία ή και περισσότερες διεργασίες. Αριστερά στο μενού του προγράμματος, εμφανίζεται η βιβλιοθήκη του προγράμματος που αποτελείται από 6 κατηγορίες με blocks. Παρακάτω, παρατίθενται επεξηγήσεις των βασικών blocks του προγράμματος NXT-2.0, χωρίς όμως αναλυτικές λεπτομέρειες, αφού σκοπός μας είναι η κατανόηση των βασικών λειτουργιών και όχι η εκμάθηση του λογισμικού.

Το πρώτο κουμπί  (common) αποτελείται από τις εξής εντολές-blocks:

Κίνησης (move). Με αυτή την εντολή-block ελέγχουμε τους κινητήρες. Μπορούμε να ελέγξουμε ποιοι κινητήρες θα χρησιμοποιηθούν καθώς και να ρυθμίσουμε την ταχύτητα, την απόσταση, τη διάρκεια και την κατεύθυνση προς την οποία επιθυμούμε να κινηθεί ο κινητήρας.

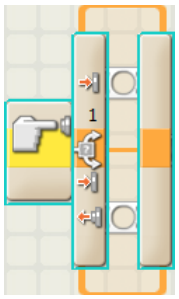
Εγγραφής (record). Αυτή η εντολή-block χρησιμοποιείται για να γράψουμε και να επαναλάβουμε μία συγκεκριμένη κίνηση. Αποθηκεύει σε μία μεταβλητή τις κινήσεις που πραγματοποιήθηκαν από τους κινητήρες που μας ενδιαφέρουν για χρονικό διάστημα που εμείς ορίζουμε και μπορούν να επαναληφτούν.

Ήχου (sound). Μας δίνει τη δυνατότητα να αναπαράγουμε κάποιον ήχο ή τόνο που βρίσκεται αποθηκευμένος. Μπορούμε να ρυθμίσουμε την ένταση και να αποφασίσουμε αν ο ήχος θα είναι επαναλαμβανόμενος ή θα ακουστεί μόνο μία φορά.

Προβολής στην οθόνη (display). Μπορούμε να προβάλουμε μία εικόνα στην οθόνη του ρομπότ μας επιλέγοντας ανάμεσα από μία μεγάλη συλλογή από εικόνες ή σχεδιάζοντας τη δική μας εικόνα ή γράφοντας το δικό μας κείμενο. Μπορούμε να προβάλλουμε ταυτόχρονα πολλές εικόνες και κείμενα.

Αναμονής (wait). Αυτή η εντολή-block μα επιτρέπει να περιμένουμε πριν εκτελεστεί η επόμενη εντολή-block. Μπορεί να περιμένει είτε για ορισμένο χρονικό διάστημα που ορίζουμε εμείς, είτε μέχρι να αλληλεπιδράσουμε με κάποιον από τους αισθητήρες του.

Επανάληψης (loop). Μπορεί να χρησιμοποιηθεί για να επαναλάβουμε ξανά πολλές εντολές-blocks γρήγορα και εύκολα. Εμείς ελέγχουμε πότε θα σταματήσει η επανάληψη. Οι επιλογές που δίνονται είναι: μετά από κάποια ώρα, μετά από συγκεκριμένο αριθμό επαναλήψεων, μετά από αλληλεπίδραση ενός αισθητήρα, ανάλογα με το αν μία συνθήκη είναι αληθής ή ψευδής και τέλος να γίνεται για πάντα.



Επιλογής(switch). Μας δίνει τη δυνατότητα να έχουμε διαφορετικές εντολές ανάλογα με το αν έχει συμβεί κάτι ή όχι. Αυτό μπορεί να είναι ένας αισθητήρας, μία μεταβλητή ή ένα χρονόμετρο που έφτασαν σε ορισμένη τιμή. Για παράδειγμα χρησιμοποιώντας τον αισθητήρα επαφής μπορούμε να καταλάβουμε αν τράκαρε το ρομπότ μας. Αν έχει τρακάρει, δηλαδή έχει είναι πατημένος ο αισθητήρας επαφής θα του πούμε να κάνει ένα βήμα πίσω. Αν όχι θα κάνει ένα βήμα μπροστά. Μπορούμε να συνδυάσουμε το παράδειγμα αυτό με ένα loop που περιγράψαμε προηγουμένως.

Το δεύτερο κουμπί  (action) περιέχει τις παρακάτω εντολές-blocks:

Κίνησης (move). Με αυτή την εντολή-block ελέγχουμε έναν κινητήρα τη φορά. Μπορούμε ρυθμίσουμε την ταχύτητα, την απόσταση, τη διάρκεια και την κατεύθυνση προς την οποία επιθυμούμε να κινηθεί ο κινητήρας.

Ήχου (sound). Το ίδιο ακριβώς με αυτό που υπάρχει στο πρώτο κουμπί (common).

Προβολής στην οθόνη (display). Το ίδιο ακριβώς με αυτό που υπάρχει στο πρώτο κουμπί (common).

Αυτή η εντολή-block μπορεί να χρησιμοποιηθεί για την αποστολή μηνυμάτων χωρίς να χρειάζεται να χρησιμοποιήσω καλώδια. Με αυτό τον τρόπο μπορώ να στείλω μηνύματα από ένα NXT σε κάποιο άλλο. Τα μηνύματα μπορεί να είναι περιέχουν κείμενο ή/και αριθμούς ή την τιμή μίας λογικής μεταβλητής.

Μας δίνει τη δυνατότητα να κάνουμε τον αισθητήρα φωτός να ανάψει σε ένα από τα τρία χρώματα που διαθέτει (πράσινο, μπλε ή κόκκινο). Για να σβήσουμε το λαμπάκι πρέπει να χρησιμοποιήσουμε ξανά την ίδια εντολή-block και να επιλέξουμε το σβήσιμο στο ίδιο χρώμα που είχαμε ανάψει.

Το τρίτο κουμπί  (sensor) περιέχει τις παρακάτω εντολές-blocks:

Αυτή η εντολή-block αναφέρεται στον αισθητήρα επαφής. Επιστρέφει μία λογική τιμή (αληθές ή ψευδές). Μπορούμε να επιλέξουμε πότε θέλουμε να μας επιστρέφεται αληθές ανάμεσα από τρείς επιλογές: όταν είναι πατημένος ο αισθητήρας, όταν δεν έχει πατηθεί καθόλου ή όταν πατήθηκε για λίγο.

Ο ανιχνευτής ήχου απεικονίζεται με αυτή την εντολή-block. Μπορεί να αναγνωρίσει ήχους επιστρέφοντας μας μία λογική τιμή. Οι ήχοι αναγνωρίζονται κατά κύριο λόγο από τη διάρκειά τους.

Αυτή η εντολή-block διαχειρίζεται τον αισθητήρας φωτός. Μπορεί να μας επιστρέψει μία τιμή που αντιστοιχεί στην φωτεινότητα που ανιχνεύτηκε ή μία λογική τιμή ανάλογα με τον τρόπο που θα το ρυθμίσουμε.

Αυτή η εντολή-block αναφέρεται στον αισθητήρα απόστασης. Η απόσταση στην οποία μπορεί να «δει» είναι περίπου 250 εκατοστά (100 ίντσες). Με αυτό μπορούμε να ανιχνεύσουμε εμπόδια κινητά και ακίνητα και να αντιδράσουμε με κάποιο τρόπο.

Αυτή η εντολή-block διαχειρίζεται τα κουμπιά που υπάρχουν πάνω στο NXT brick. Μας επιστρέφει τη λογική τιμή «αληθές» αν έχει πατηθεί κάποιο κουμπί.

Ο ανιχνευτής περιστροφής εμφανίζεται με αυτή την εντολή-block. Μας βοηθάει να μετράμε πόσες πλήρεις περιστροφές έκανε ένας κινητήρας. Μπορεί να μας επιστρέψει μία λογική τιμή μόλις εκτελεστούν οι προκαθορισμένες στροφές.

Με αυτή την εντολή-block ελέγχουμε τα 3 χρονόμετρα που υπάρχουν σε κάθε πρόγραμμα που γράφουμε. Και τα τρία ξεκινούν ταυτόχρονα μόλις ξεκινήσει το πρόγραμμά μας και μπορούμε να τα μηδενίσουμε και να διαβάσουμε την τιμή που έχουν όποτε θέλουμε.

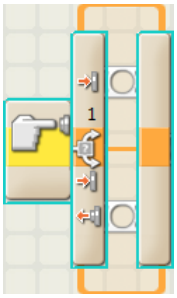
Αυτή η εντολή-block μπορεί να χρησιμοποιηθεί για την λήψη μηνυμάτων χωρίς να χρειάζεται να χρησιμοποιήσω καλώδια. Με αυτό τον τρόπο μπορώ να λάβω μηνύματα από ένα άλλο NXT. Τα μηνύματα μπορεί να είναι περιέχουν κείμενο ή/και αριθμούς ή την τιμή μίας λογικής μεταβλητής. Μπορώ να χρησιμοποιήσω το μήνυμα όπως το έλαβα ή να το συγκρίνω με κάποιο αποθηκευμένο μήνυμα και να επιστρέψω μία λογική τιμή.

Με αυτή την εντολή-block μπορούμε να ελέγξουμε τον αισθητήρα χρώματος. Ο αισθητήρας αυτός μπορεί να μας βοηθήσει να ανιχνεύσουμε διάφορα χρώματα ή την φωτεινότητα του περιβάλλοντος.

Το τρίτο κουμπί  (flow) αποτελείται από τις πιο κάτω εντολές-blocks:

Αναμονής (wait). Το ίδιο ακριβώς με αυτό που υπάρχει στο πρώτο κουμπί (common).

Επανάληψης (loop). Το ίδιο ακριβώς με αυτό που υπάρχει στο πρώτο κουμπί (common).



Επιλογής (switch). Το ίδιο ακριβώς με αυτό που υπάρχει στο πρώτο κουμπί (common).

Με αυτή την εντολή-block μας δίνεται η δυνατότητα να σταματήσουμε όλα όσα συμβαίνουν στο σύστημά μας. Σταματάει τους κινητήρες, τις λάμπες και τους ήχους.

Το τέταρτο κουμπί  (data) στο οποίο περιέχονται οι ακόλουθες εντολές-blocks:

Με τη χρήση αυτής της εντολής-block μπορούμε να κάνουμε λογικές πράξεις. Δέχεται ως είσοδο λογικές τιμές (αληθές ή ψευδές) και μας επιστρέφει το αποτέλεσμα τις λογικής πράξης που επιλέξαμε.

Αυτή η εντολή-block μας βοηθάει να εκτελούμε απλές αριθμητικές όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση.

Χρησιμοποιώντας αυτή την εντολή-block μπορούμε να κάνουμε συγκρίσεις μεταξύ αριθμών, δηλαδή μπορούμε να δούμε αν ένας αριθμός είναι μεγαλύτερος, μικρότερος ή ίσος με κάποιον άλλο.

Αυτή την εντολή-block μας βοηθάει να καταλάβουμε αν ένας αριθμός ανήκει ή όχι σε ένα συγκεκριμένο σύνολο αριθμών.

Αυτή η εντολή-block μας επιστρέφει έναν τυχαίο αριθμό από ένα εύρος που εμείς καθορίζουμε.

Αυτή η εντολή-block μας επιτρέπει να χρησιμοποιούμε μεταβλητές στο πρόγραμμά μας.

Με αυτή την εντολή-block μας δίνεται η δυνατότητα να χρησιμοποιήσουμε σταθερές μεταβλητές.

Το πέμπτο και τελευταίο κουμπί  (advanced) περιέχει τις επόμενες εντολές-blocks:

Αυτή η εντολή-block μας βοηθάει να μετατρέψουμε έναν αριθμό (πχ. το αποτέλεσμα μίας πράξης ή την τιμή ενός αισθητήρα) σε κείμενο το οποίο μπορούμε να εμφανίσουμε στην οθόνη του ρομπότ.

Αυτή η εντολή-block μας δίνει την δυνατότητα να δημιουργήσουμε μεγάλα κομμάτια κειμένου (προτάσεις) χρησιμοποιώντας μικρότερα.

Με αυτή την εντολή-block μπορούμε να εμποδίσουμε το ρομπότ να μπει σε κατάσταση ύπνου (sleep mode) όταν το πρόγραμμα μου πρέπει να περιμένει περισσότερο από τον χρόνο που το ρομπότ χρειάζεται για να μπει σε κατάσταση ύπνου.

Αυτή η εντολή-block χρησιμεύει για τη δημιουργία ενός εγγράφου με τη χρήση του ρομπότ. Αφού δημιουργήσουμε ένα έγγραφο πρέπει πρώτα να το κλείσουμε και μετά έχουμε τη δυνατότητα να διαβάσουμε τι γράψαμε ή να το σβήσουμε.

Αυτή η εντολή μας βοηθάει να ρυθμίσουμε από 0% που είναι το ελάχιστο ως 100% που είναι το μέγιστο τις τιμές τις οποίες θα αντιλαμβάνονται ο αισθητήρας ήχου και ο αισθητήρας φωτός.

Αυτή η εντολή-block μας επιτρέπει να κλείσουμε την αυτόματη διόρθωση σφάλματος κίνησης που υπάρχει στους κινητήρες.

Με αυτή την εντολή-block μπορούμε να ανοίγουμε και να κλείνουμε τα Bluetooth.

Επιπλέον υπάρχουν και οι δύο παρακάτω εντολές-block:

Αυτές μας δίνουν τη δυνατότητα να δημιουργούμε τις δικές μας ρουτίνες κώδικα που μπορούμε να ξαναχρησιμοποιήσουμε ή να κατεβάσουμε ήδη υπάρχουσες ρουτίνες από το διαδίκτυο.

ΠΑΡΑΡΤΗΜΑ ΙΙ – ΟΔΗΓΙΕΣ ΕΓΚΑΤΑΣΤΑΣΗΣ LEJOS ΚΑΙ ECLIPSE

Οδηγίες εγκατάστασης του λογισμικού LeJOS στο NXT τούβλο του Eclipse και του LeJOS στο Eclipse.

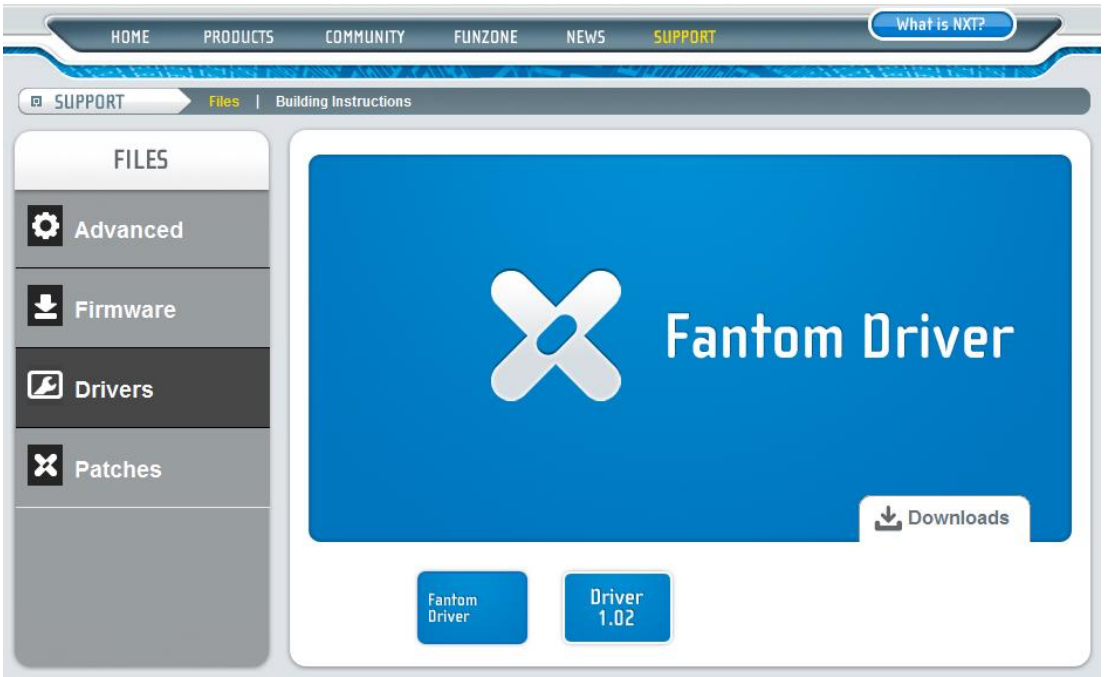
- Οι παρακάτω οδηγίες εξασφαλίζουν την επιτυχή εγκατάσταση του LeJOS στο NXT τούβλο. Χρησιμοποιήθηκε υπολογιστής με Windows 7.

Βήμα 1:

Αρχικά, πρέπει να εγκαταστήσουμε στον υπολογιστή μας Java JDK. Η έκδοση που χρησιμοποιήθηκε ήταν η πιο πρόσφατη που υπήρχε όταν δημιουργήθηκε αυτό το έγγραφο (JDK 7). Η εγκατάσταση του είναι απλή και μπορούμε να το κατεβάσουμε από την ιστοσελίδα www.oracle.com/technetwork/java/javase/downloads/index.html/. Επιπλέον, πρέπει να έχουμε στον υπολογιστή μας κάποιο πρόγραμμα αποσυμπίεσης αρχείων .zip όπως το WinRar.

Βήμα 2:

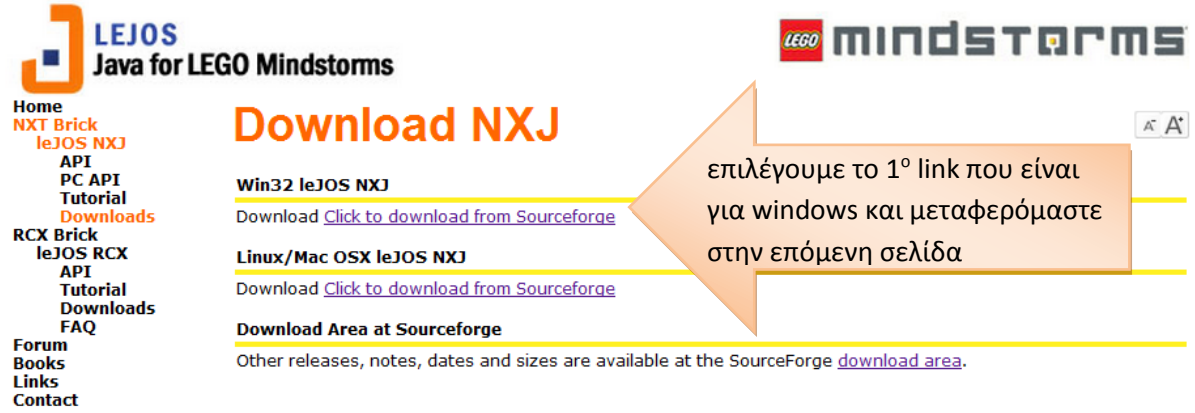
Αν έχετε ήδη εγκαταστήσει το περιβάλλον σχεδίασης NXT και τους drivers από το cd που υπάρχει στη συσκευασία του ρομπότ τότε μπορείτε να προχωρήσετε στο επόμενο βήμα. Αυτό επειδή κάποια αρχεία είναι απαραίτητα. Εναλλακτικά, μπορείτε να κατεβάσετε από την ιστοσελίδα της Lego τον Fantom Driver (http://mindstorms.lego.com/en-us/support/files/default.aspx#Driver)

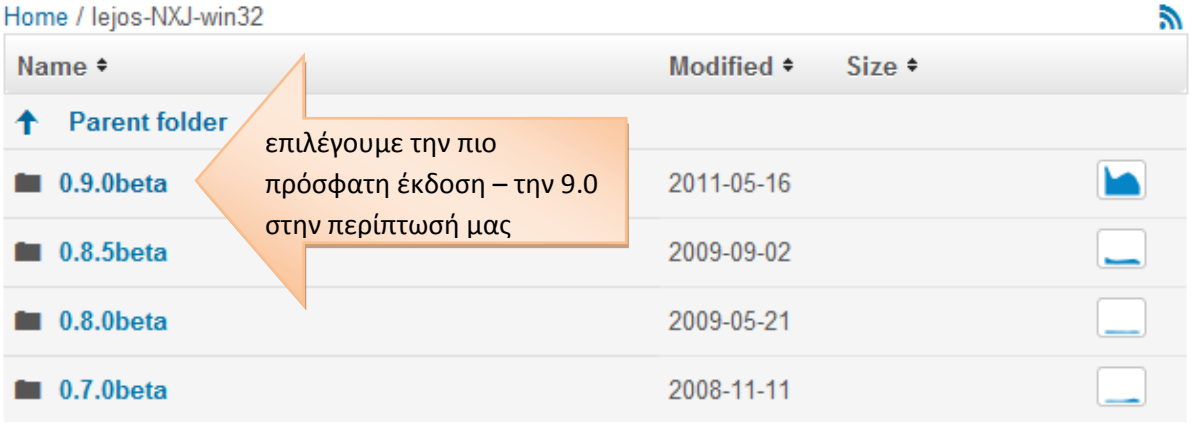


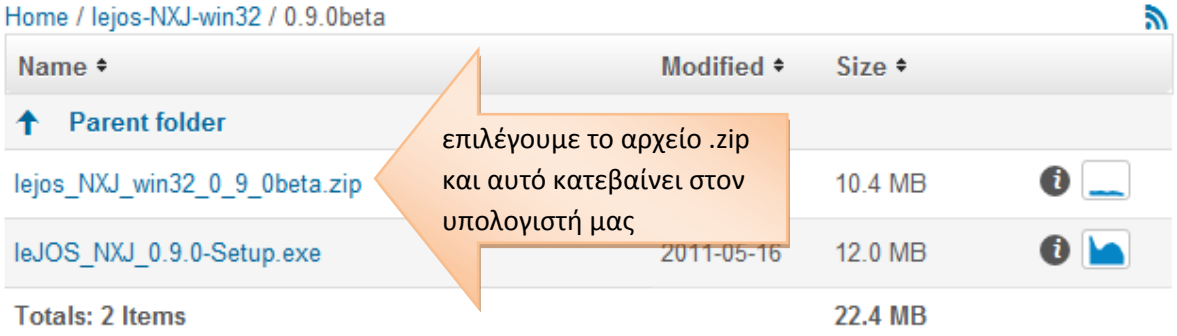
Βήμα 3:

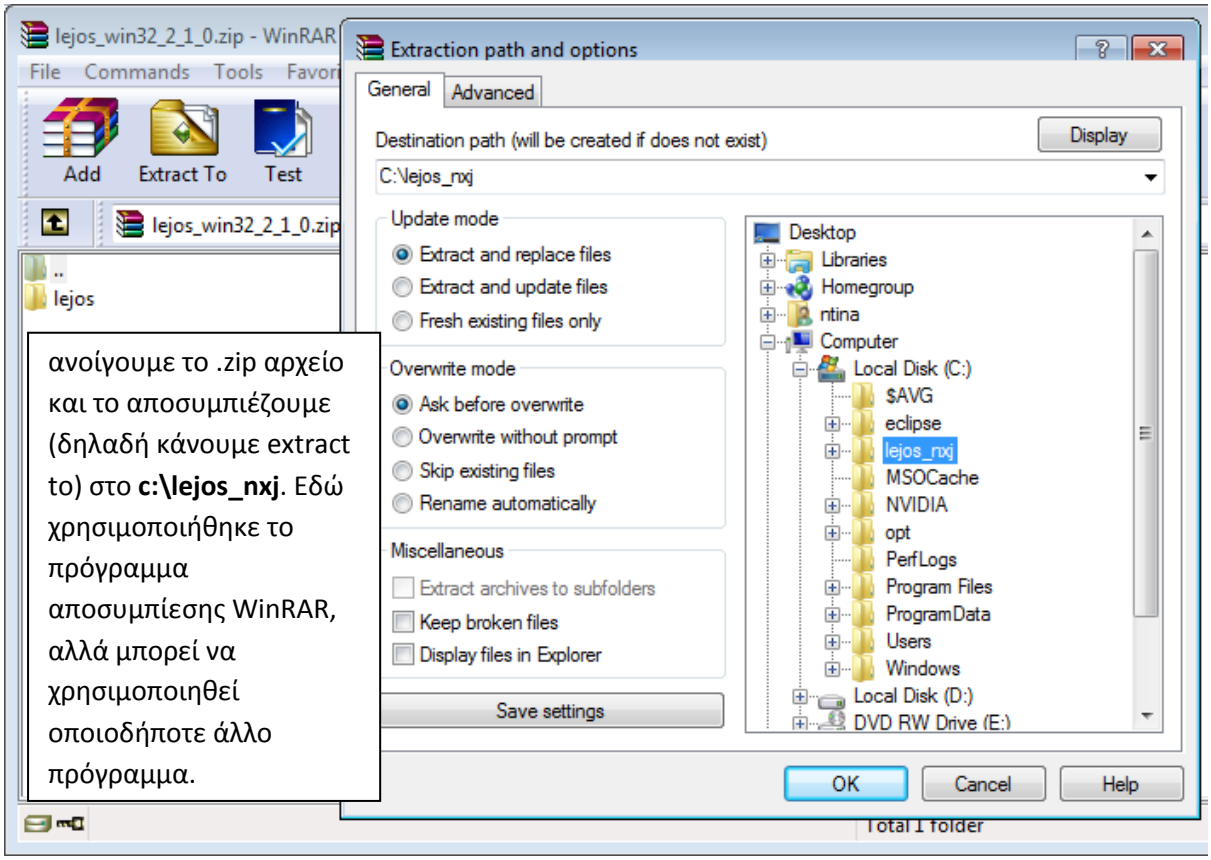
Κατεβάζουμε την πιο πρόσφατη έκδοση του LeJOS. Η σελίδα στην οποία μπορούμε να βρούμε τις διαθέσιμες εκδόσεις αλλά και πληροφορίες σχετικά με το LeJOS είναι η http://lejos.sourceforge.net.

Όταν μπούμε στη σελίδα του LeJOS πηγαίνουμε στα Downloads.

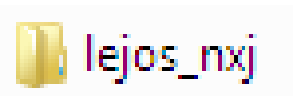








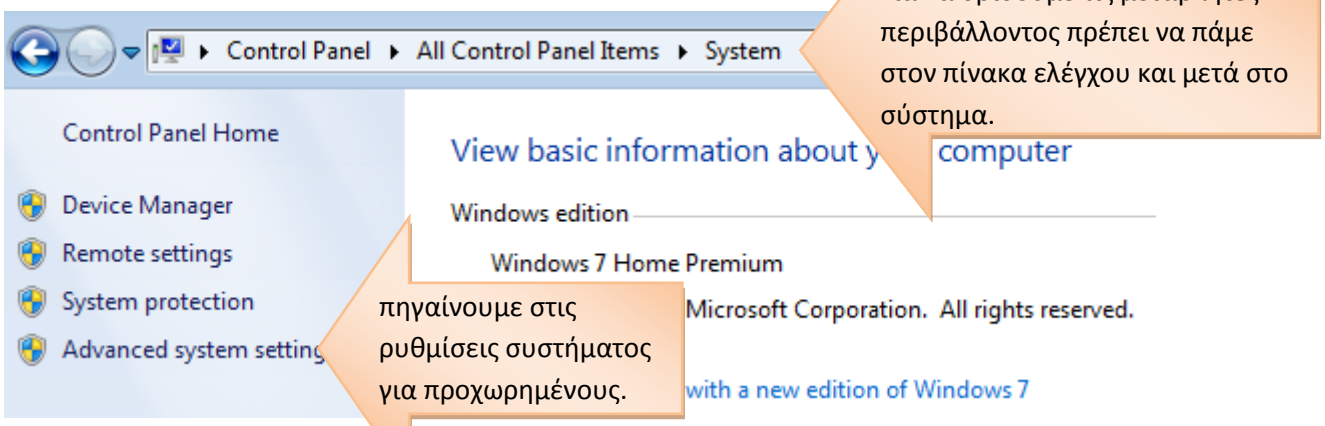
ανοίγουμε το .zip αρχείο και το αποσυμπιέζουμε (δηλαδή κάνουμε extract το) στο **c:\lejos_nxj**. Εδώ χρησιμοποιήθηκε το πρόγραμμα αποσυμπίεσης WinRAR, αλλά μπορεί να χρησιμοποιηθεί οποιοδήποτε άλλο πρόγραμμα.



Αφού αποσυμπιεστεί το .zip θα πρέπει να μπορούμε να δούμε το φάκελο της διπλανής εικόνας στο: έναρξη → Ο Υπολογιστής Μου →c (start → My Computer → c).

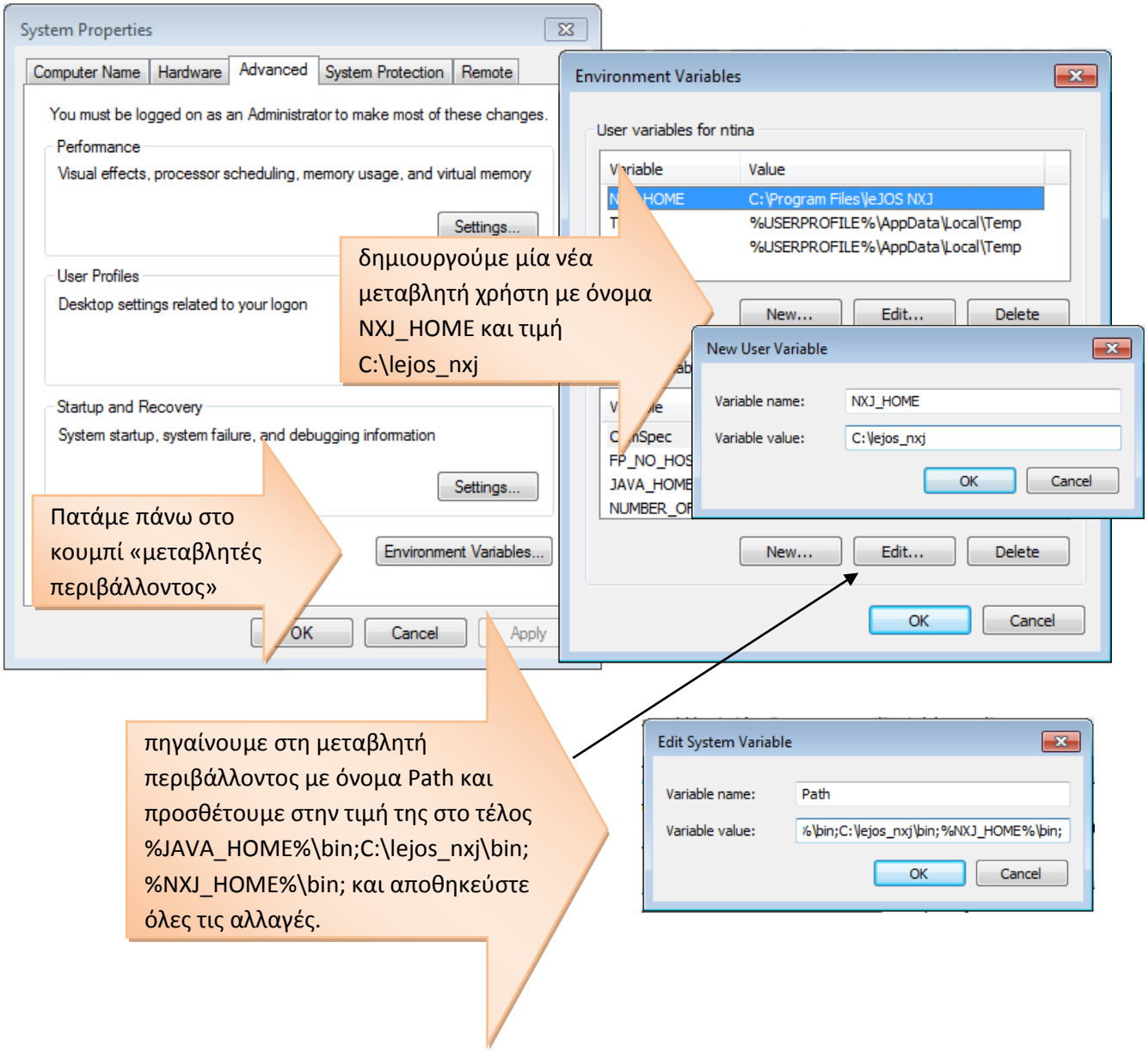
Βήμα 4: Ορισμός μεταβλητών περιβάλλοντος

Αν έχετε windows 7 βεβαιωθείτε ότι όταν είστε στον πίνακα ελέγχου έχετε επιλέξει την κλασική προβολή.



Για να ορίσουμε τις μεταβλητές περιβάλλοντος πρέπει να πάμε στον πίνακα ελέγχου και μετά στο σύστημα.

πηγαίνουμε στις ρυθμίσεις συστήματος για προχωρημένους.

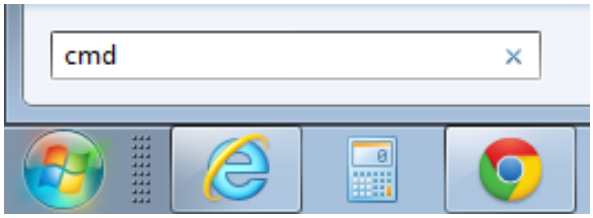


Βήμα 6:

Κάνουμε reset το NXT τούβλο. Για να γίνει αυτό το γυρνάμε ανάποδα και με κάτι λεπτό όπως μια οδοντογλυφίδα πατάμε το ασημί κουμπί που υπάρχει μέσα στην τρύπα για 5 δευτερόλεπτα. Η οθόνη του τούβλου θα σβήσει και θα ακούγεται ένας ήχος «τικ» σχεδόν ανά δευτερόλεπτο. Όπως φαίνεται στην εικόνα το κουμπί βρίσκεται στην ακριανή τρύπα στην πλευρά που συνδέεται το καλώδιο σύνδεσης με τον υπολογιστή.

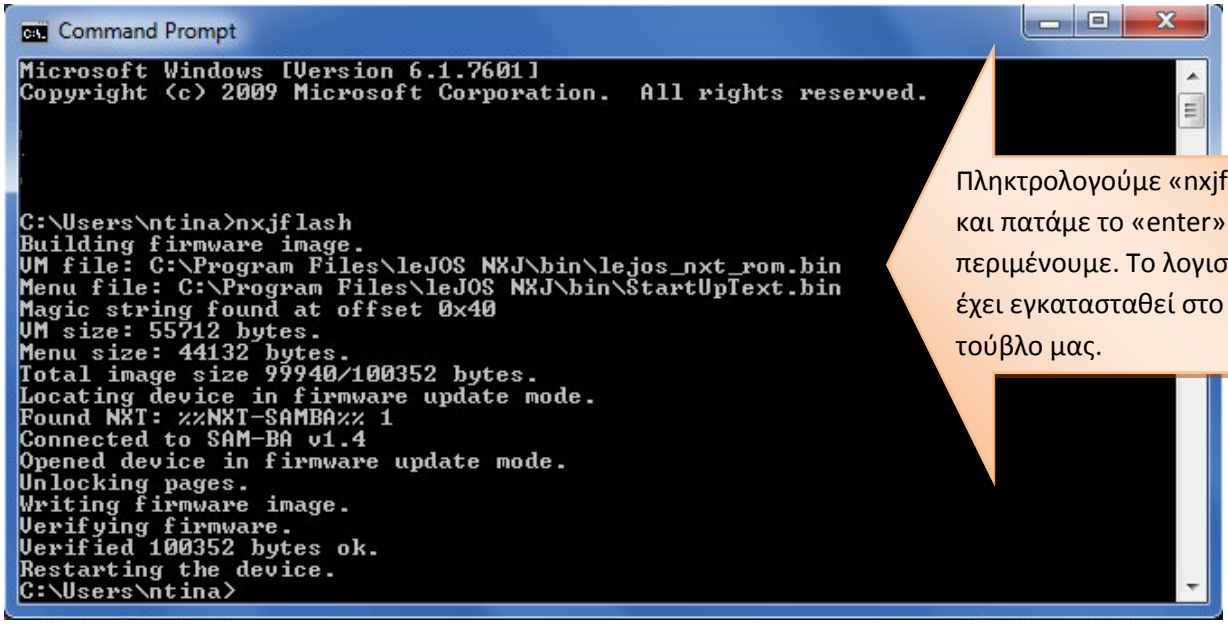
Βήμα 7: Εγκατάσταση του LeJOS στο NXT τούβλο

Συνδέουμε το NXT τούβλο με τον υπολογιστή μέσω της θύρας USB.



Στα windows 7, πατάμε στην έναρξη και πληκτρολογούμε στην αναζήτηση «cmd» και πατάμε το ender. Αυτό θα μας ανοίξει το command prompt.

Βήμα 8:

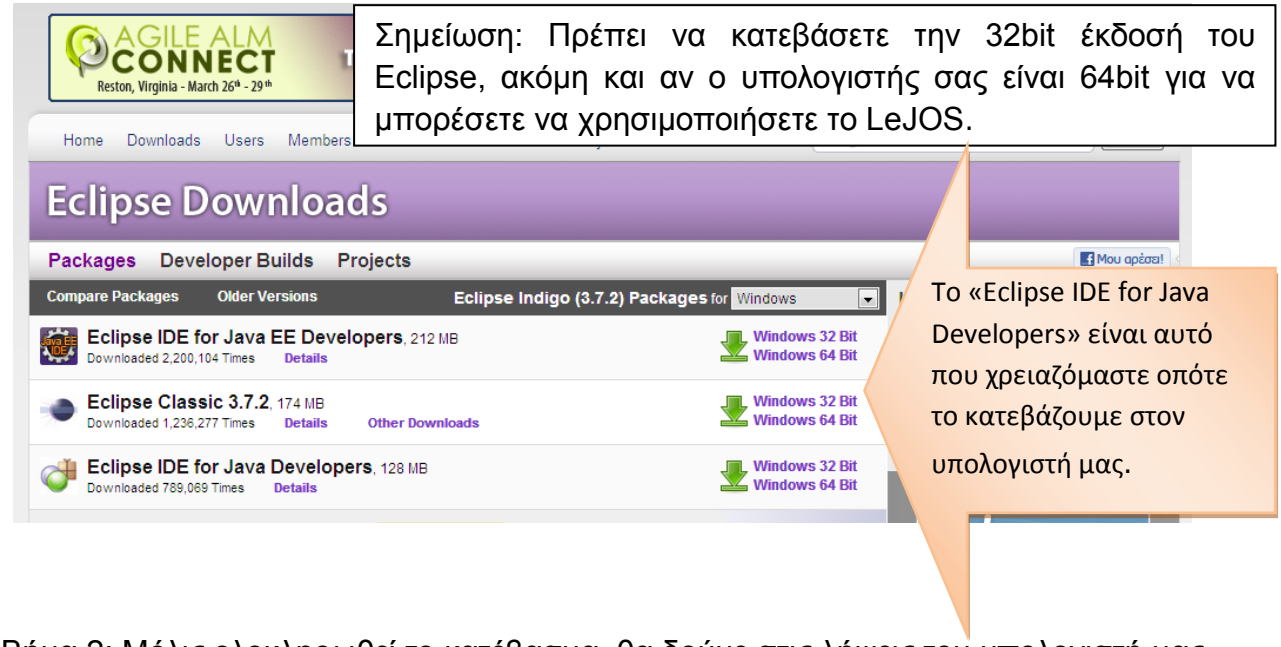


Πληκτρολογούμε «nxjflash» και πατάμε το «enter» και περιμένουμε. Το λογισμικό έχει εγκατασταθεί στο τούβλο μας.

Μόλις πατήσουμε το πορτοκαλί κουμπί πάνω στο NXT τούβλο, η οθόνη θα ανάψει και θα δούμε το λογότυπο του LeJOS. Αυτό σημαίνει ότι η εγκατάστασή μας

- Για την εγκατάσταση του Eclipse στον υπολογιστή μας:

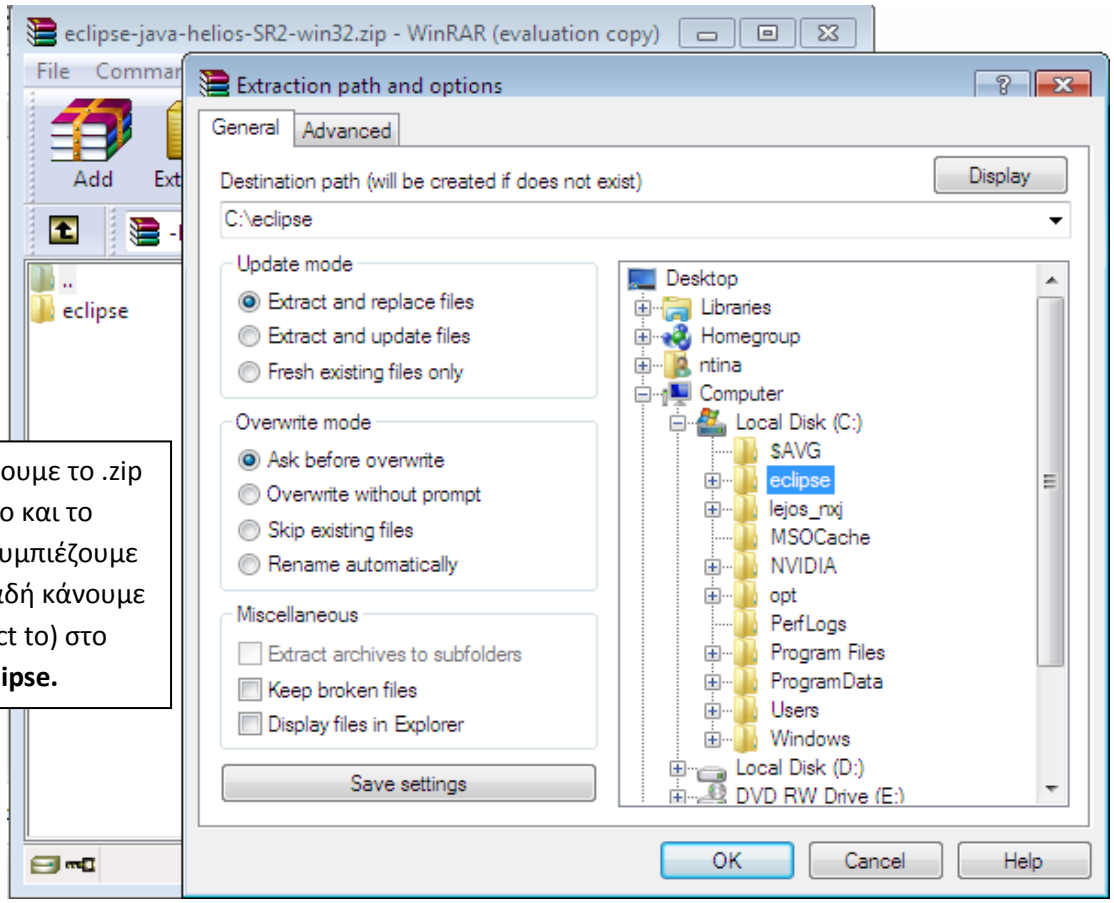
Βήμα 1: Πρώτα πηγαίνουμε στη σελίδα του Eclipse στα downloads (http://www.eclipse.org/downloads/) για να κατεβάσουμε το Eclipse.



Σημείωση: Πρέπει να κατεβάσετε την 32bit έκδοσή του Eclipse, ακόμη και αν ο υπολογιστής σας είναι 64bit για να μπορέσετε να χρησιμοποιήσετε το LeJOS.

Το «Eclipse IDE for Java Developers» είναι αυτό που χρειαζόμαστε οπότε το κατεβάζουμε στον υπολογιστή μας.

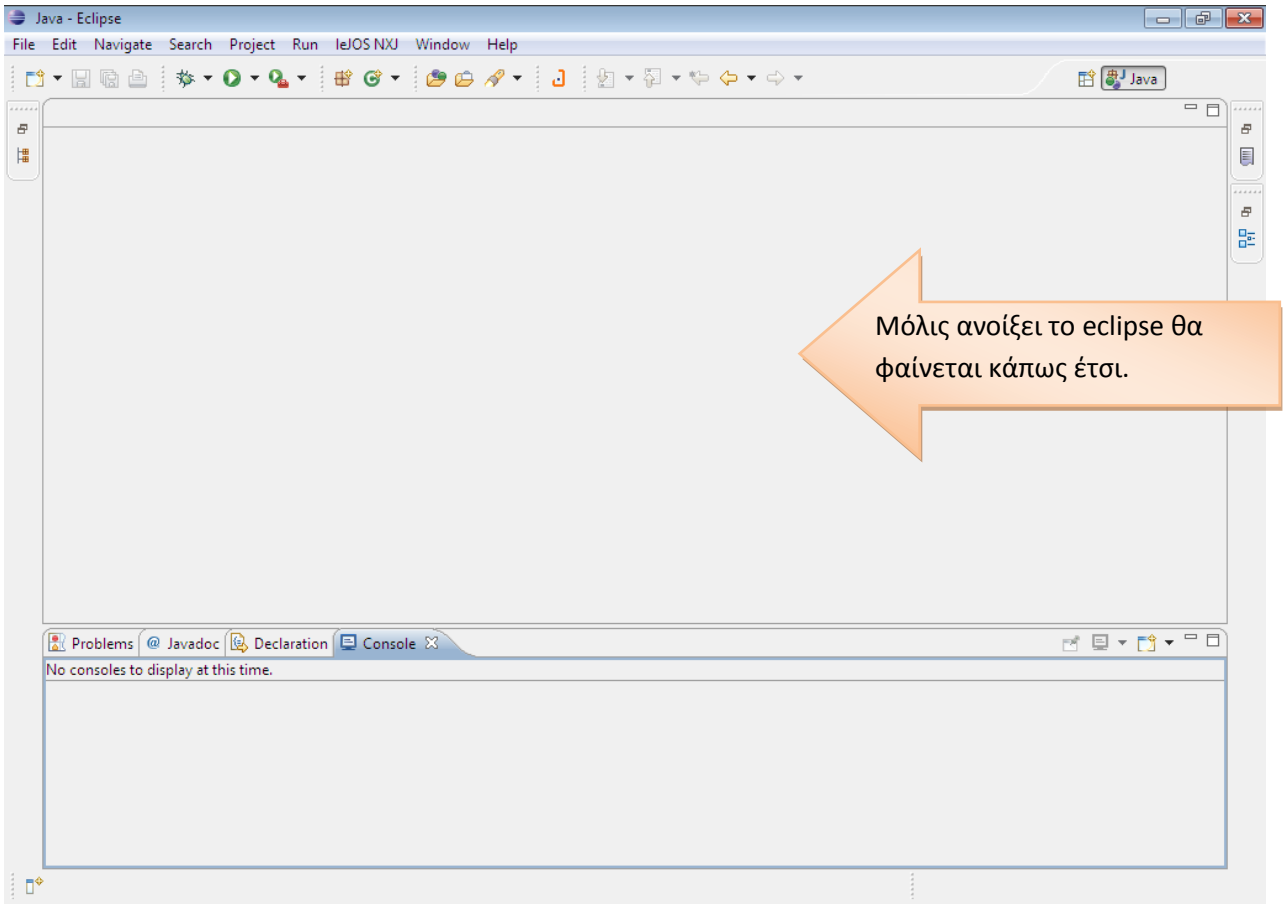
Βήμα 2: Μόλις ολοκληρωθεί το κατέβασμα, θα δούμε στις λήψεις του υπολογιστή μας ένα αρχείο τύπου .zip το οποίο πρέπει να κάνουμε unzip.



ανοίγουμε το .zip αρχείο και το αποσυμπιέζουμε (δηλαδή κάνουμε extract το) στο **c:\eclipse.**

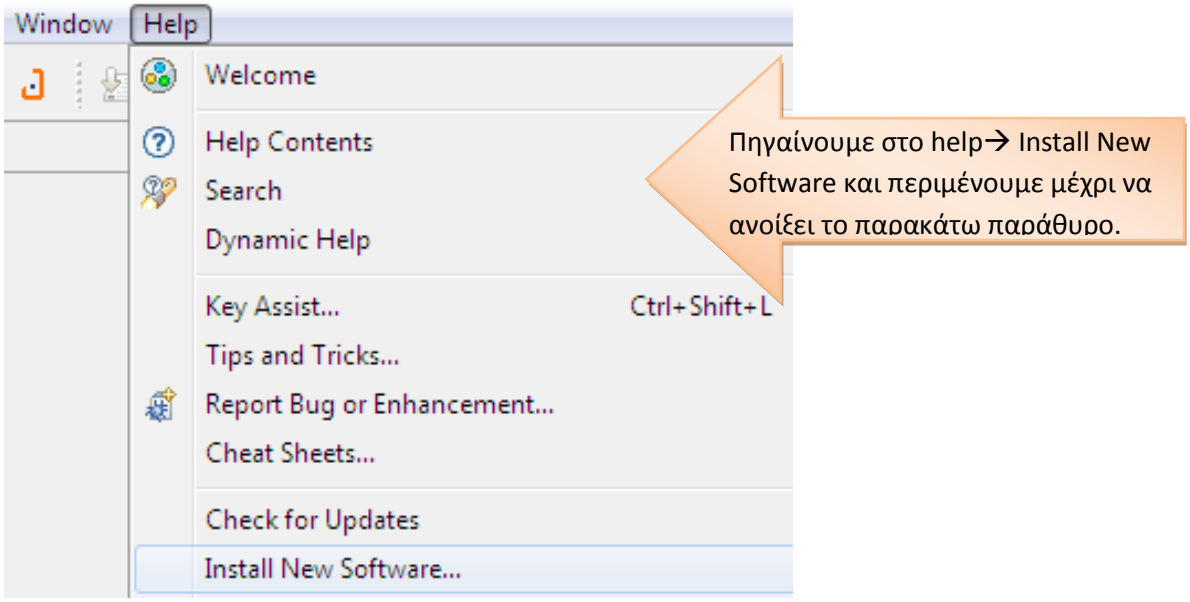
Αφού αποσυμπιεστεί το .zip θα πρέπει να μπορούμε να δούμε το φάκελο της διπλανής εικόνας στο: έναρξη → Ο Υπολογιστής Μου →c (start → My Computer → c).

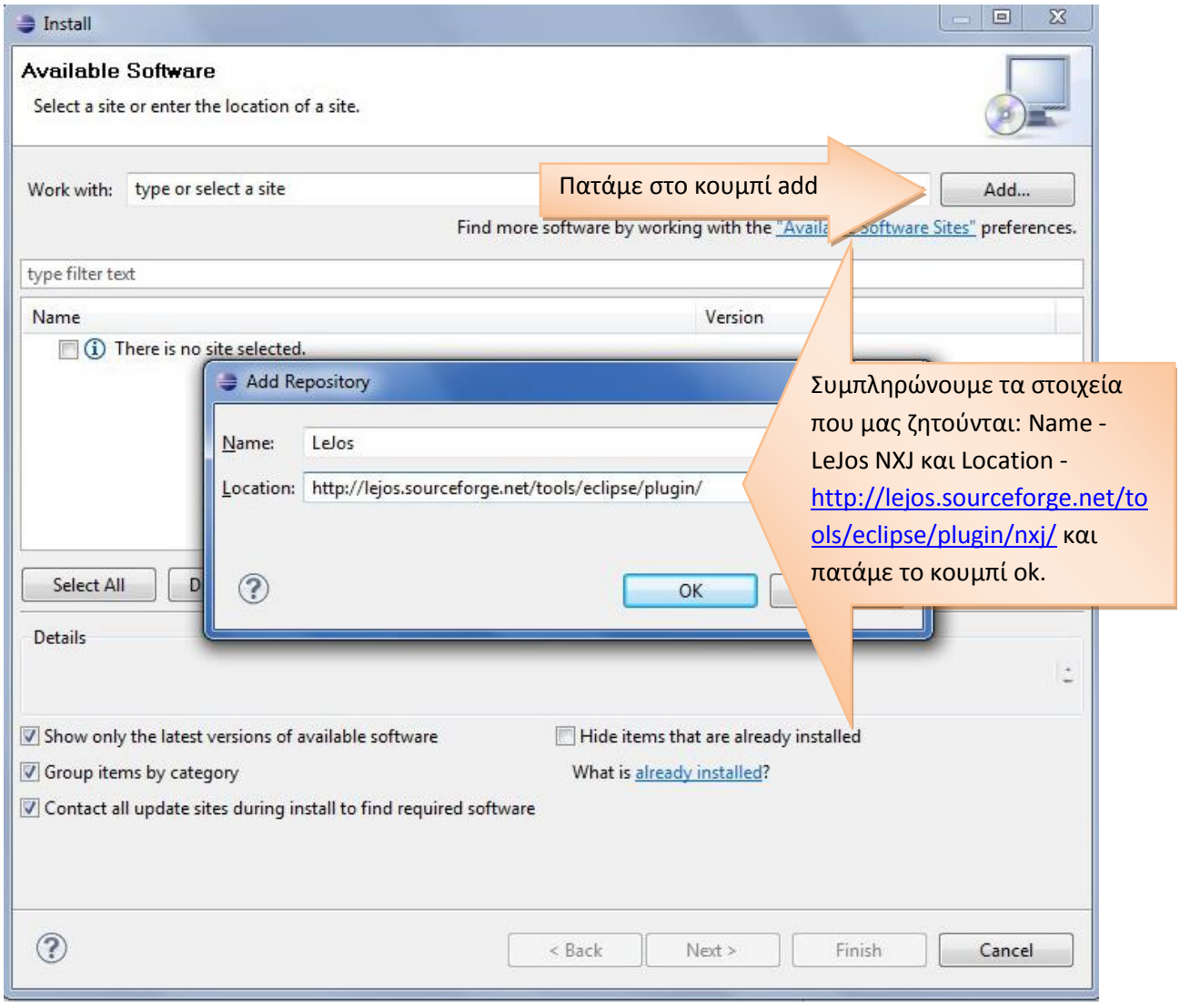
Βήμα 3: Για να ολοκληρώσουμε την εγκατάσταση ανοίγουμε τον φάκελο «eclipse» και κάνουμε διπλό κλικ στο εκτελέσιμο αρχείο με τίτλο «eclipse» και το «eclipse» ανοίγει. Την πρώτη φορά που θα το ανοίξουμε θα μας ζητήσει να ορίσουμε το workspace, δηλαδή τον χώρο στον οποίο θέλουμε να αποθηκεύονται τα project μας. Αν θέλουμε μπορούμε να το αφήσουμε όπως είναι.

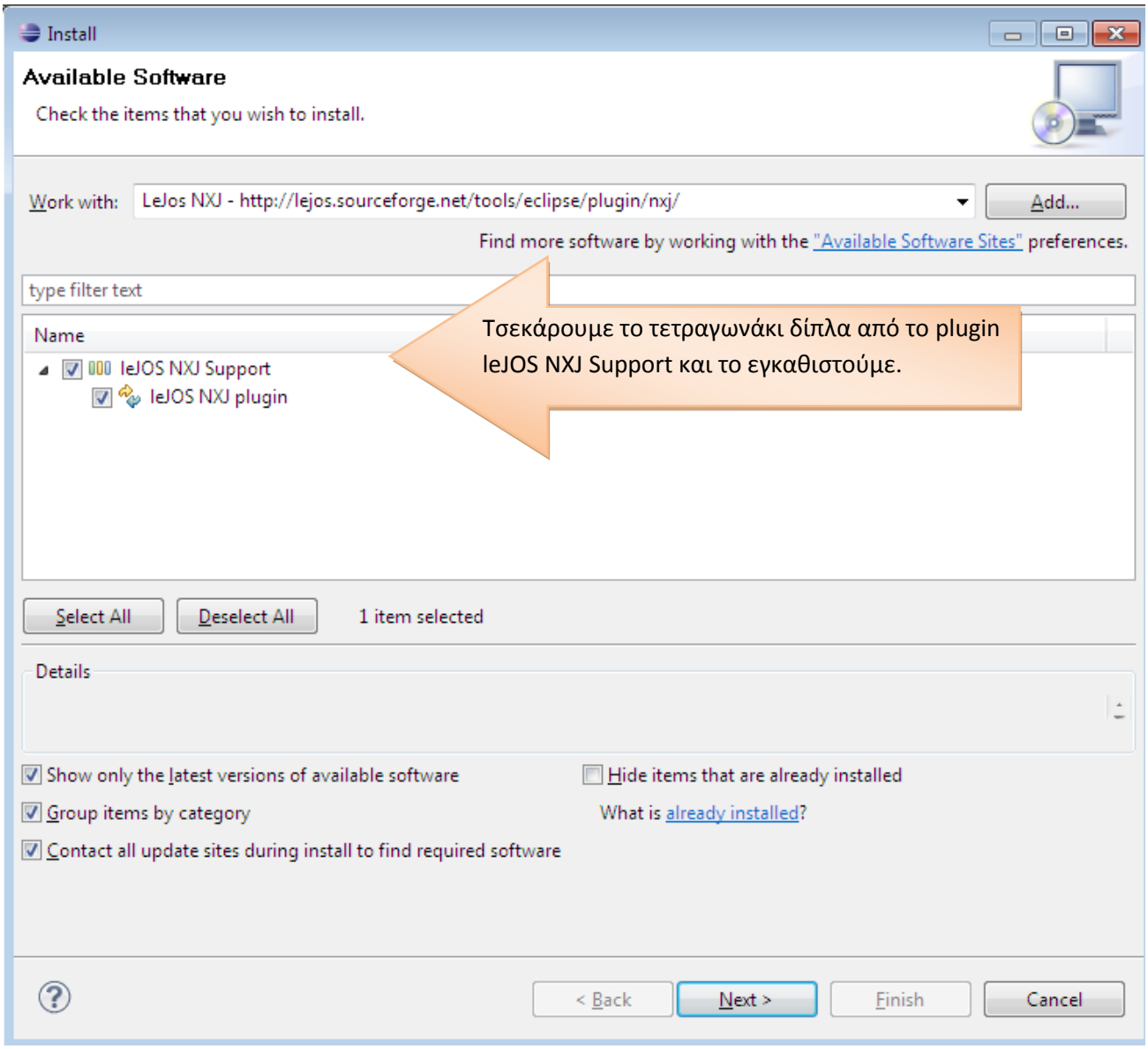


- Για την εγκατάσταση του LeJOS στο eclipse, ο πιο εύκολος και προτεινόμενος τρόπος είναι να χρησιμοποιήσουμε το plugin για το LeJOS που υπάρχει διαθέσιμο. Αυτό μας επιτρέπει να δημιουργούμε γρήγορα και χωρίς κόπο NXT projects στα οποία θα χρησιμοποιούμε τη Java ως γλώσσα προγραμματισμού.

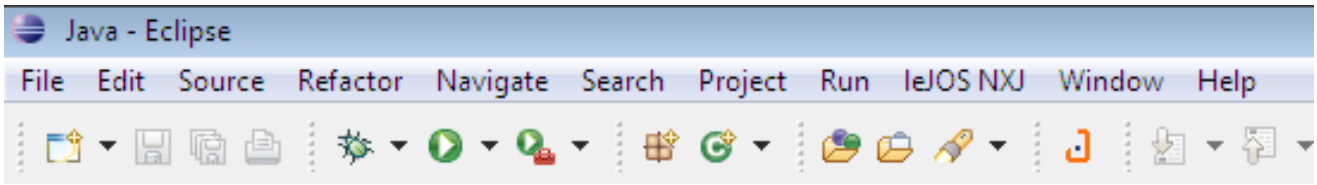
Βήμα 1: Κατεβάζουμε και εγκαθιστούμε το plugin για το LeJOS



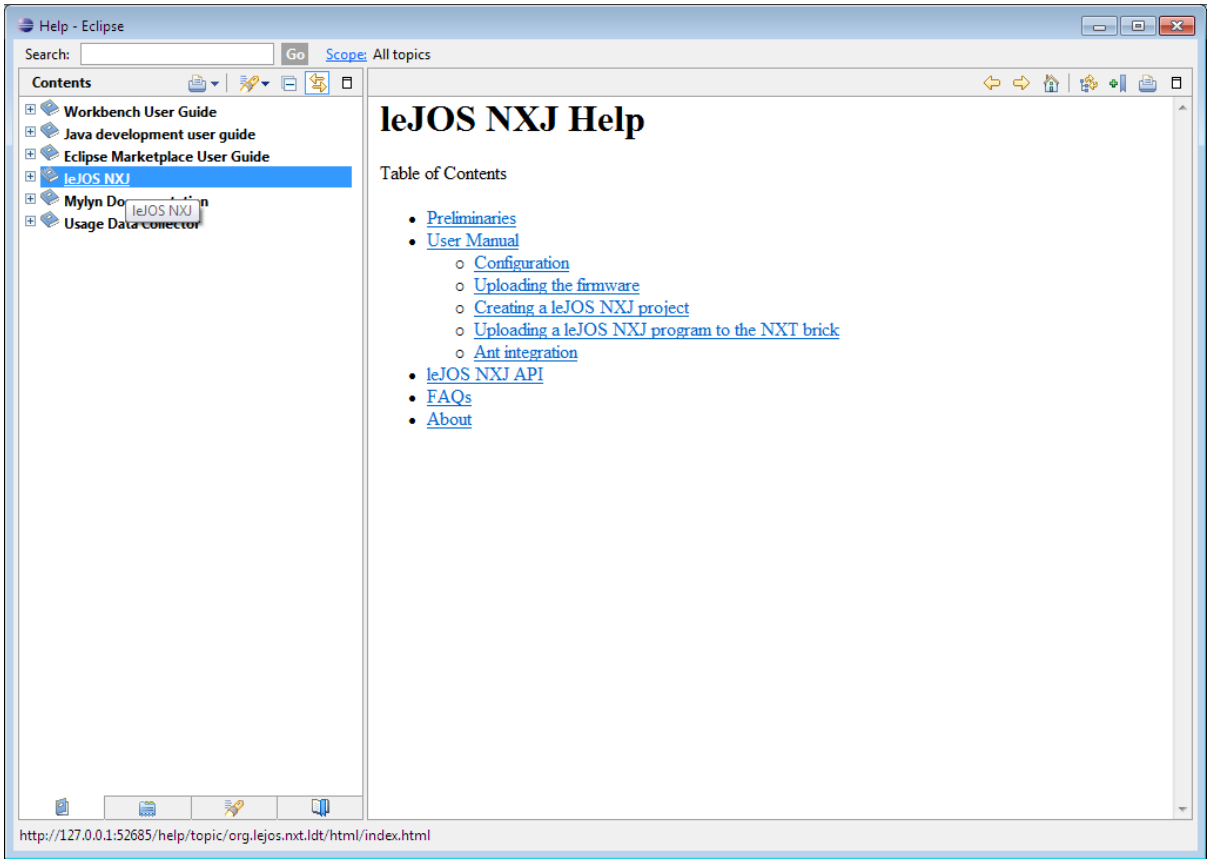




Μόλις ολοκληρωθεί η εγκατάσταση θα μας ζητήσει να κλείσουμε και να ανοίξουμε ξανά το eclipse. Αφού το κάνουμε αυτό θα είναι διαθέσιμο στο μενού του Eclipse το LeJOS NXJ, όπως φαίνεται στην παρακάτω εικόνα.

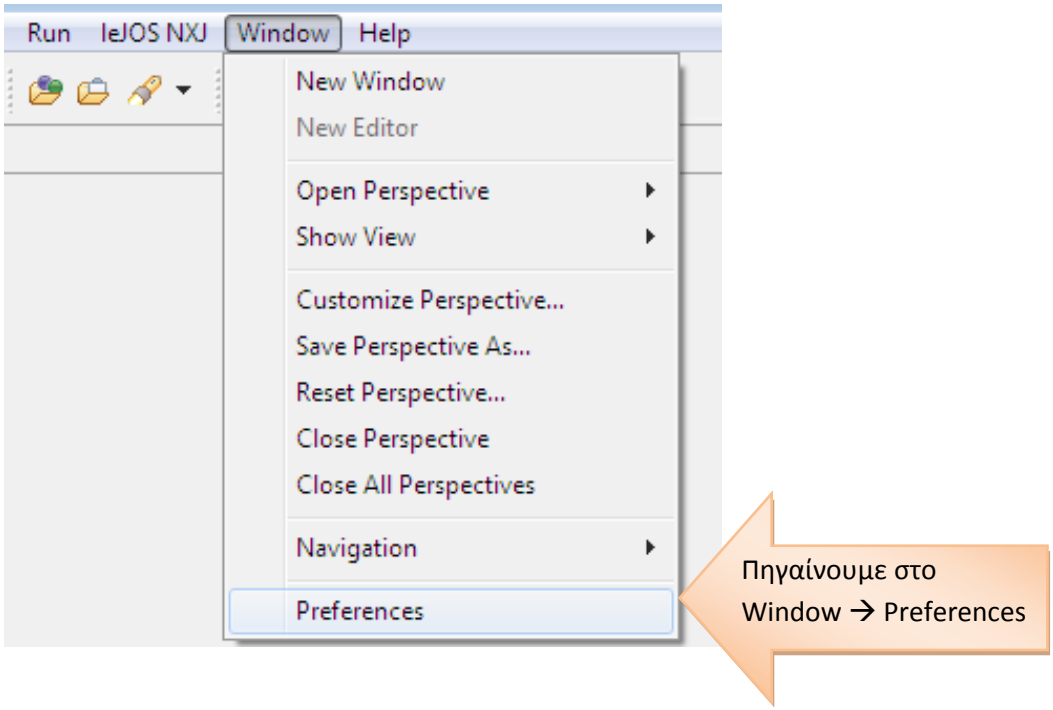


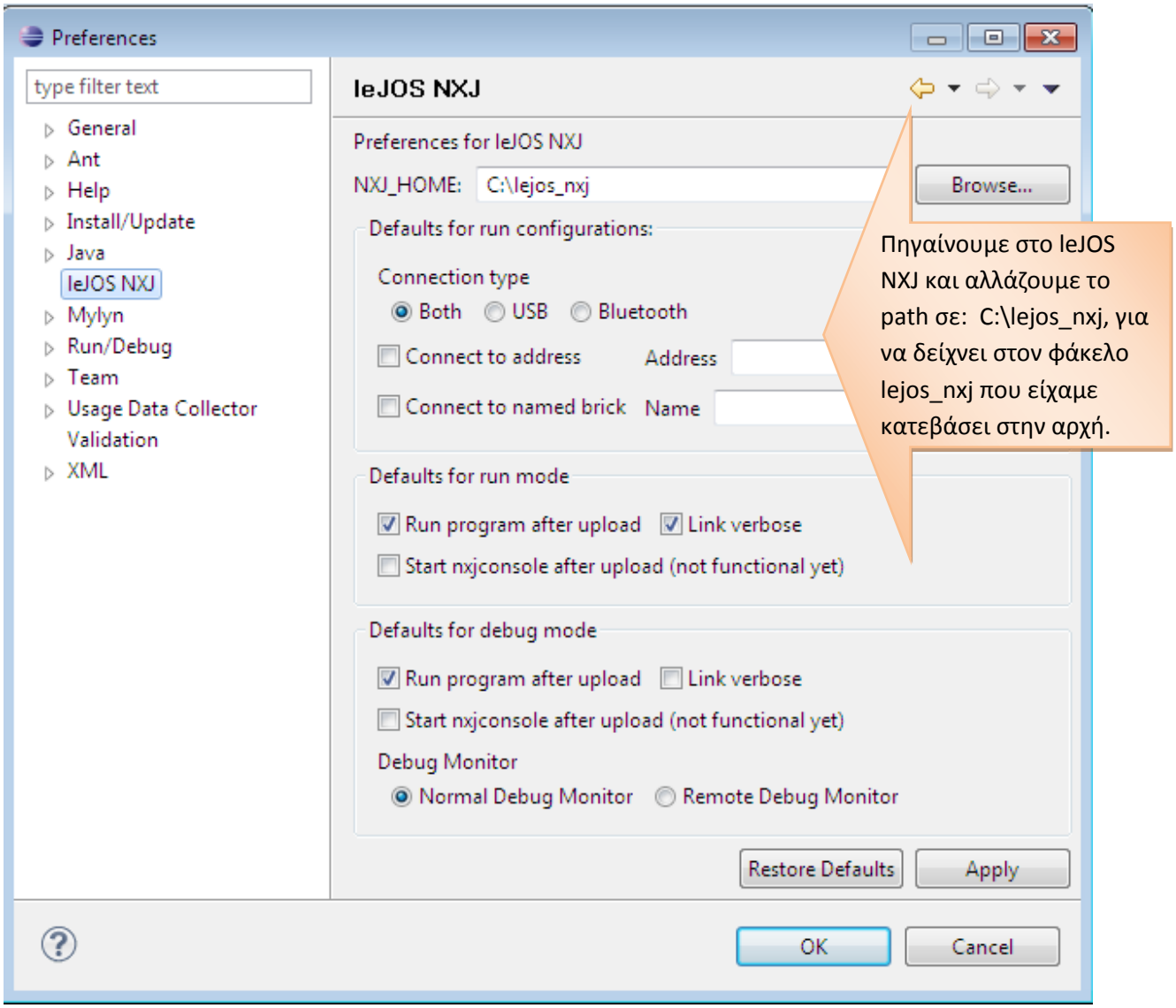
Επιπλέον, καλό είναι να γνωρίζουμε ότι μπορούμε να πάμε στο help → Help Contents και να βρούμε χρήσιμες πληροφορίες και βοήθεια για το LeJOS.



Καλό είναι να διαβάσετε τις πληροφορίες για το LeJOS από το Help Contents επειδή είναι εύκολες στην κατανόηση και θα σας βοηθήσουν να εξοικειωθείτε με το plugin γρήγορα.

Βήμα 2: Θέτουμε τις παραμέτρους

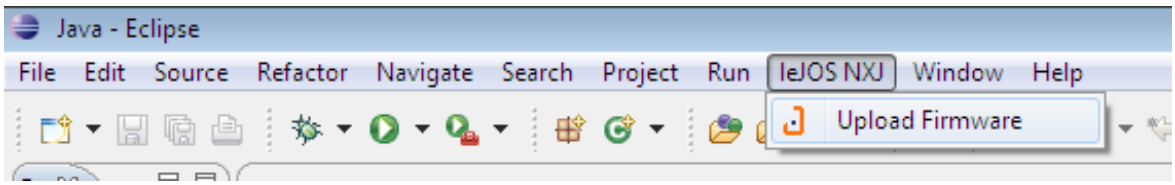




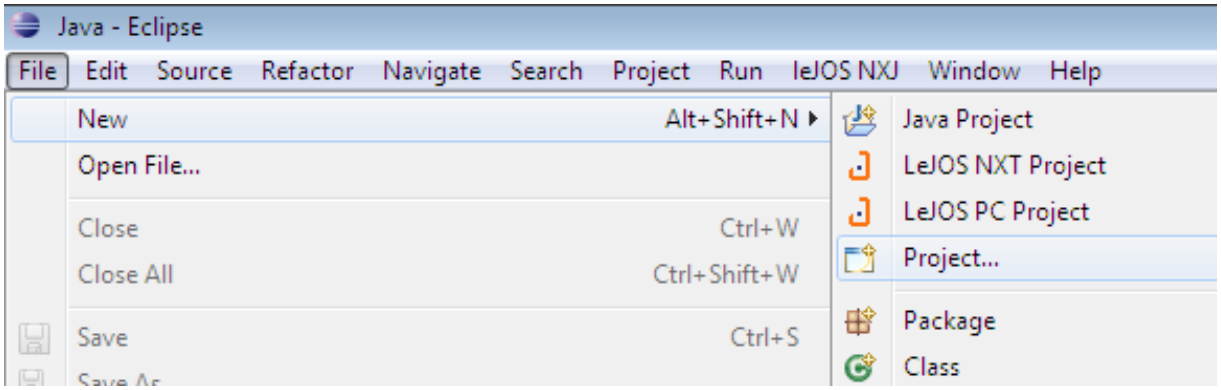
Τώρα το plugin είναι έτοιμο για χρήση.

Χρήσιμες πληροφορίες:

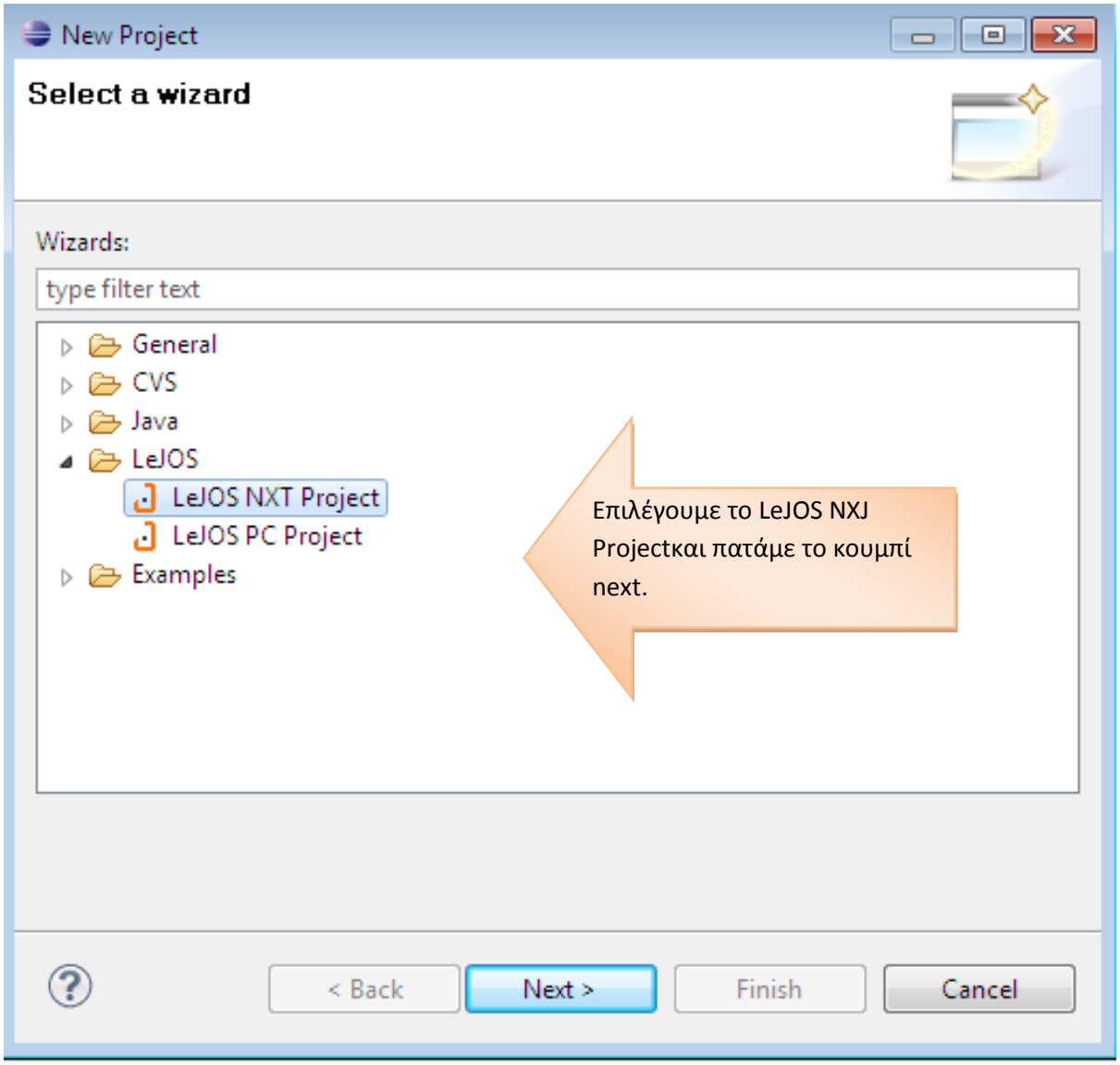
- Μπορούμε να κάνουμε update το firmware το LeJOS μέσω του eclipse απλά πατώντας leJOS NXJ→ Upload Firmware.

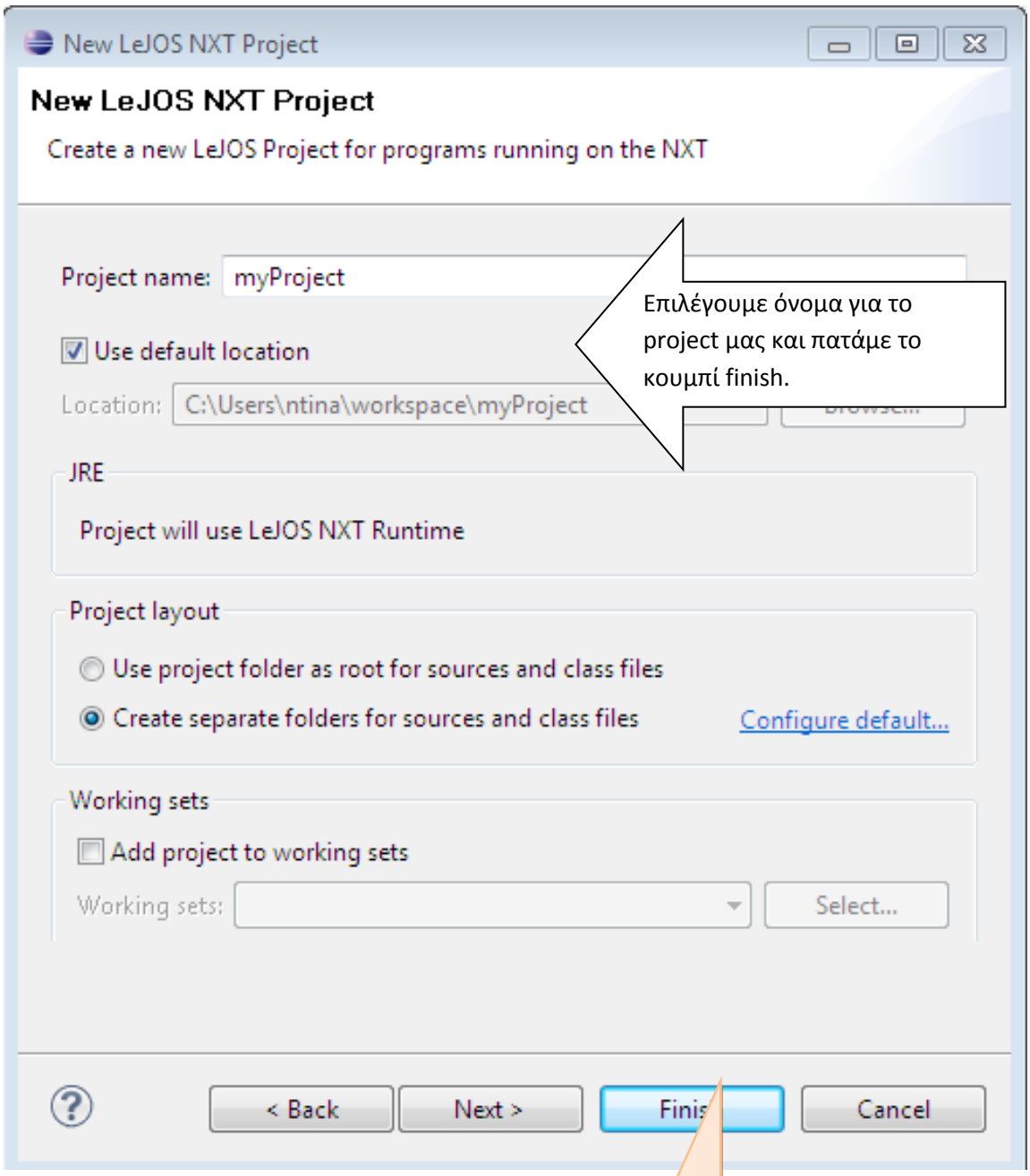


- Για τη δημιουργία ενός νέου project, πηγαίνουμε στο File → New → Project.

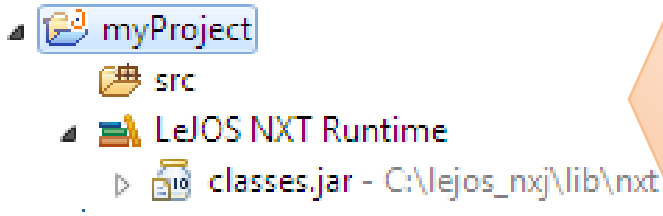


Ο παρακάτω οδηγός δημιουργίας project εμφανίζεται.



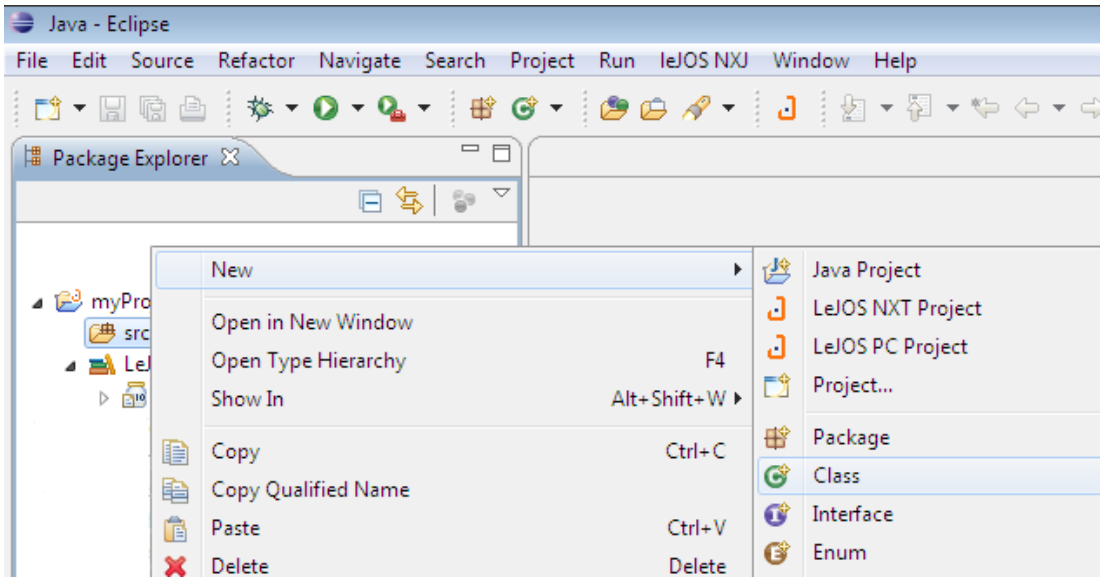


Επιλέγουμε όνομα για το project μας και πατάμε το κουμπί finish.

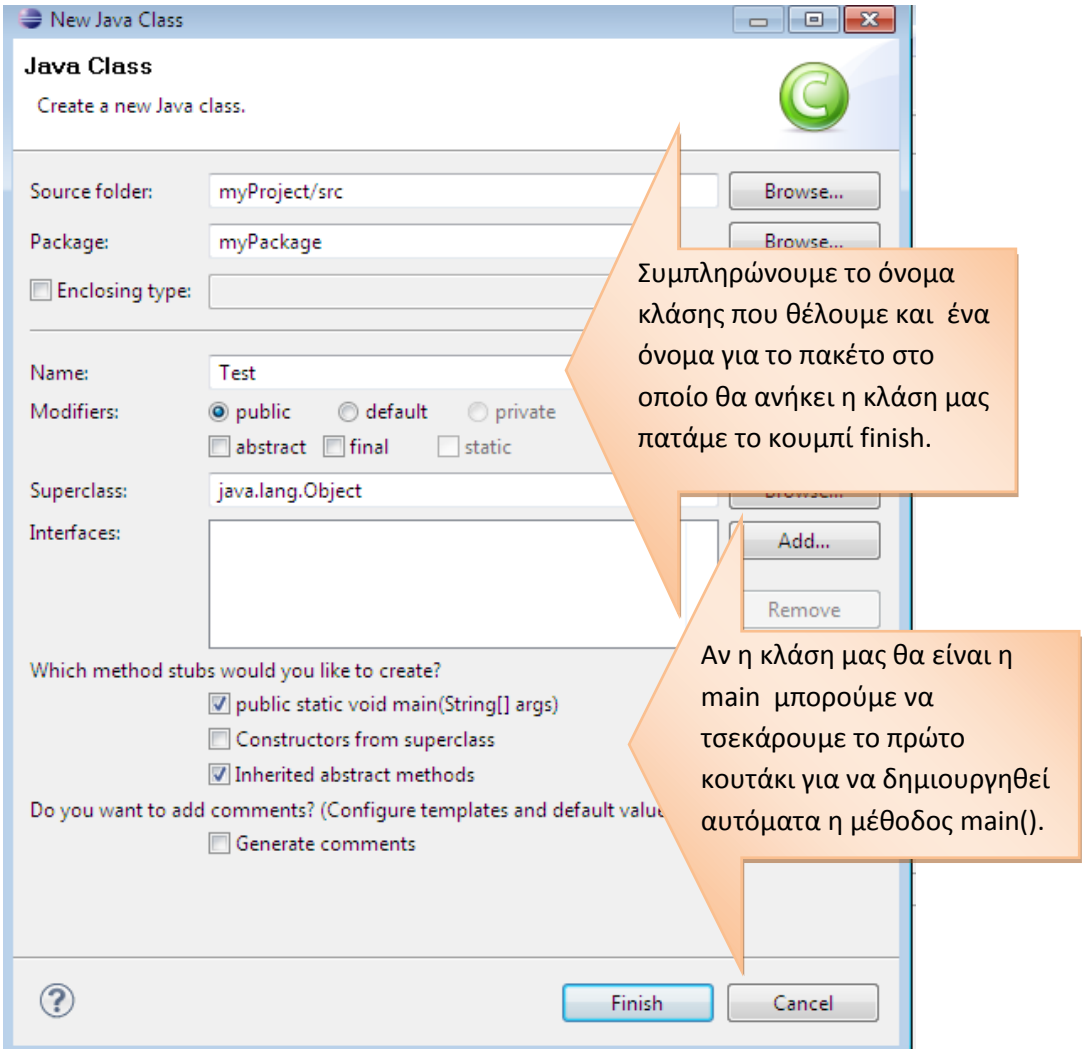


Έτσι εμφανίζεται στα αριστερά ένα καινούριο project με το όνομα που επιλέξαμε. Το πορτοκαλί σύμβολο που μοιάζει με το γράμμα «j» μας δείχνει ότι είναι ένα NXT project. Επιπλέον, μπορούμε να δούμε ότι υπάρχει ήδη μέσα στο project μας το classes.jar στις οποίες μπορούμε να βρούμε όλες τις διαθέσιμες κλάσεις που μπορούμε να χρησιμοποιήσουμε για να κάνουμε τα δικά μας προγράμματα.

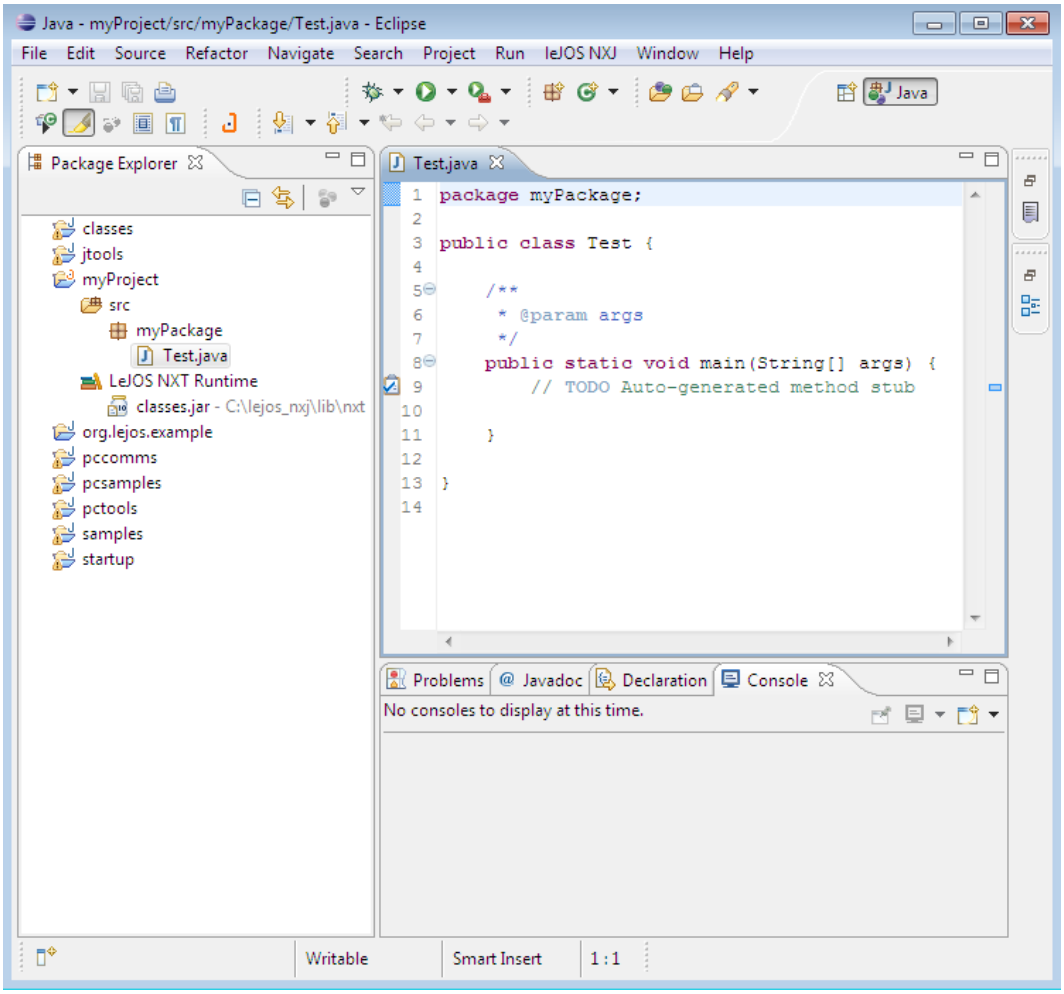
- Για να δημιουργήσουμε μία καινούρια κλάση κάνουμε δεξί κλικ στον φάκελο src και επιλέγουμε new → class.



Ο παρακάτω οδηγός δημιουργίας project εμφανίζεται.



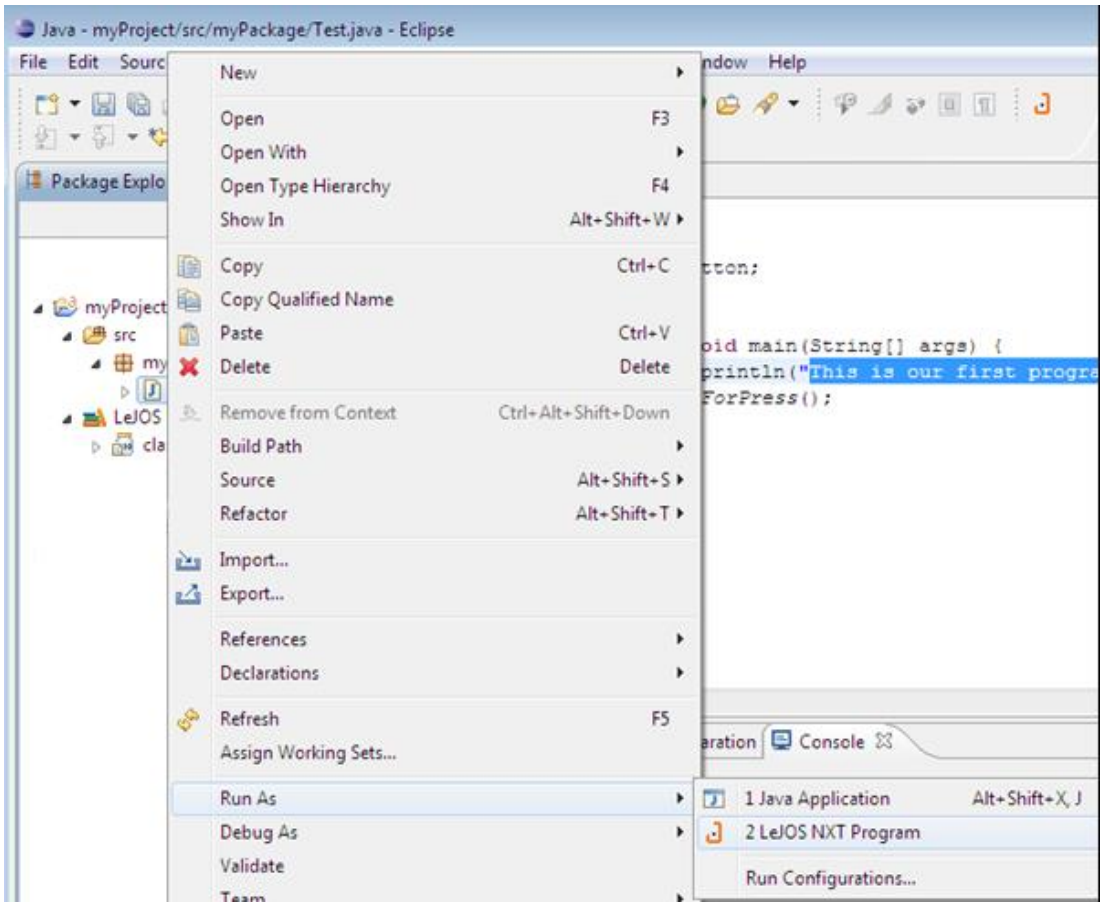
Αφού πατήσουμε το κουμπί finish θα δούμε την παρακάτω εικόνα:



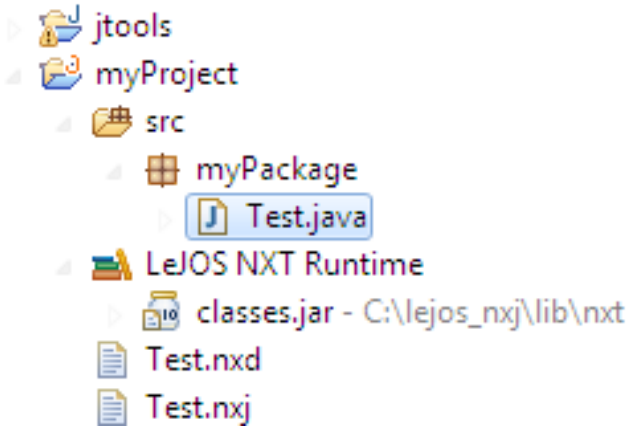
- Η διαδικασία για να ανεβάσουμε ένα πρόγραμμα στο NXT τούβλο είναι απλή. Ας γράψουμε ένα απλό πρόγραμμα για να δούμε τη διαδικασία. Το πρόγραμμα που ακολουθεί θα εμφανίσει στην οθόνη του NXT τούβλου μας την πρόταση «This is our first program!» και θα περιμένει μέχρι να πατήσουμε οποιοδήποτε κουμπί.

```
package myPackage;

import lejos.nxt.Button;

public class Test {
    public static void main(String[] args) {
        System.out.println("This is our first program!");
        Button.waitForPress();
    }
}
```

Για να ανεβάσουμε το πρόγραμμά μας κάνουμε δεξί κλικ στο όνομα της κλάσης μας και επιλέγουμε Run As → LeJOS NXT Program.

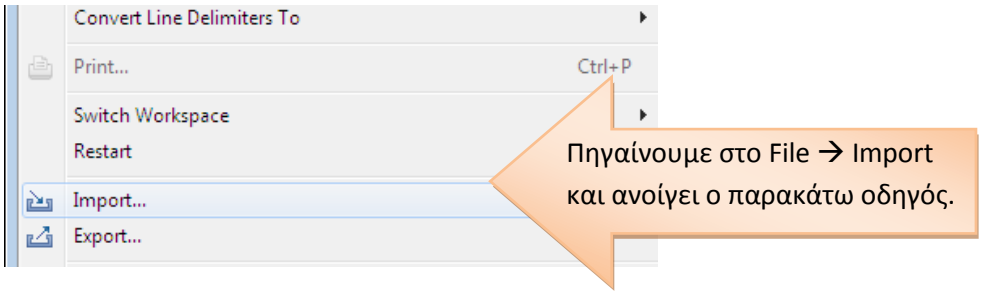


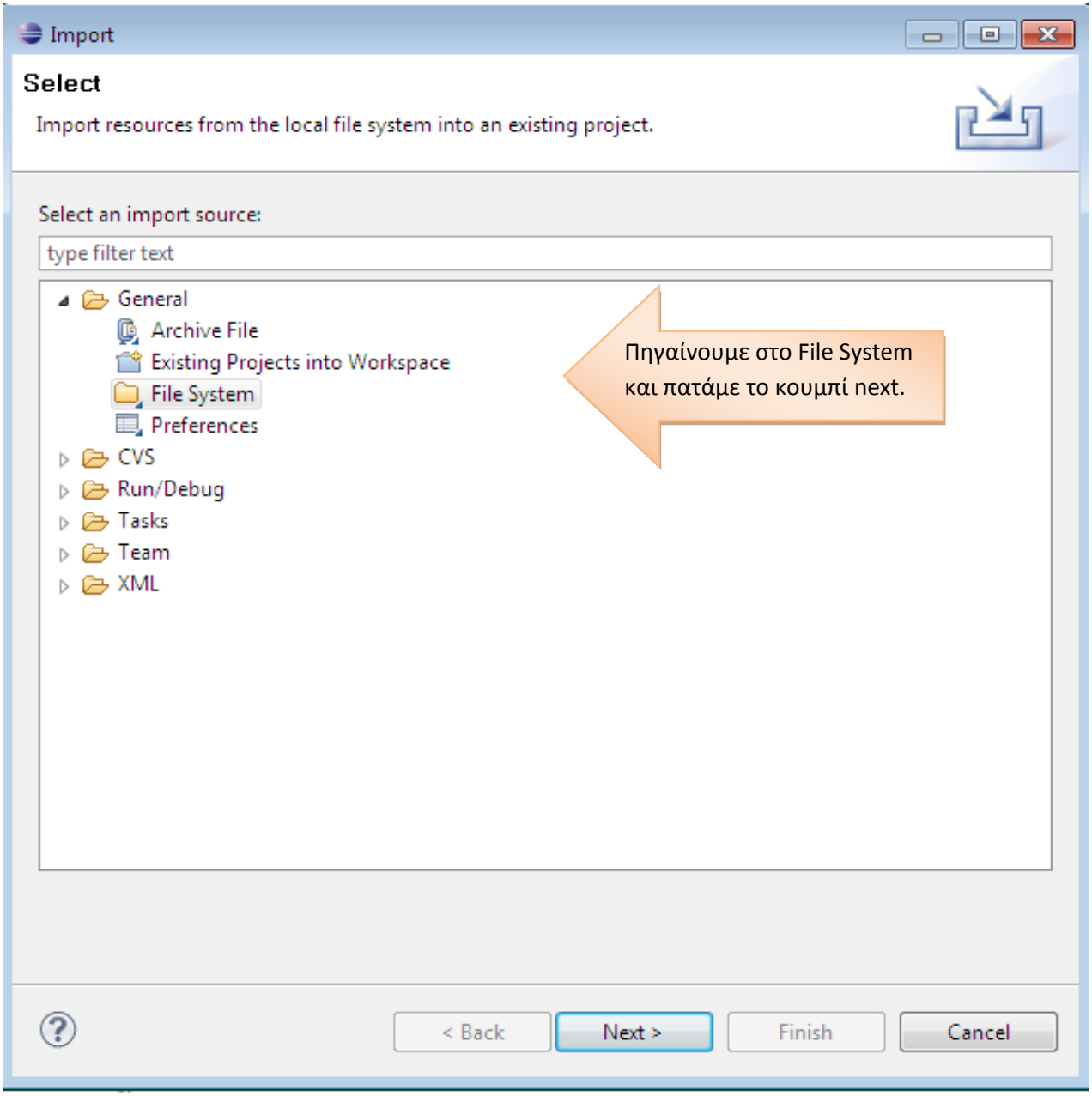
Το αποτέλεσμα θα είναι να δούμε το πρόγραμμά μας να εκτελείται και δύο νέα αρχεία όπως φαίνεται και στην εικόνα παρακάτω δημιουργούνται:

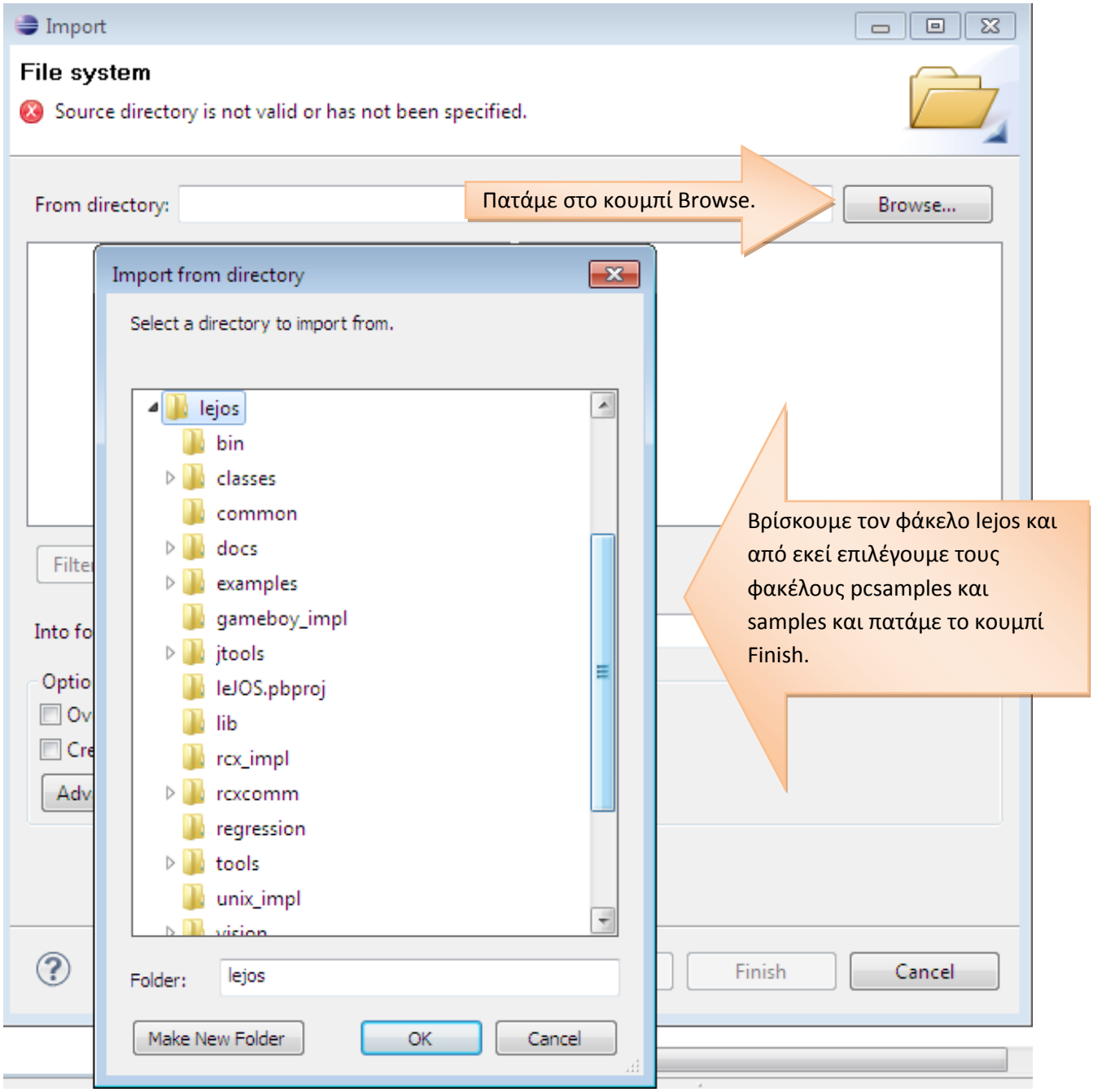


Τα δύο νέα αρχεία είναι το Test.nxd το οποίο χρησιμοποιείται για την απασφαλμάτωση του προγράμματός μας και Test.nxj το οποίο είναι το αρχείο που ουσιαστικά μεταφέρεται στο NXT τούβλο μας.

- Ένα πολύ χρήσιμο βήμα είναι να προσθέσουμε στο Eclipse τα παραδείγματα που μας δίνει το LeJOS για να τα μελετήσουμε. Είναι ίσως ο πιο εύκολος τρόπος να ξεκινήσει κάποιος να γράφει προγράμματα για το ρομπότ του.





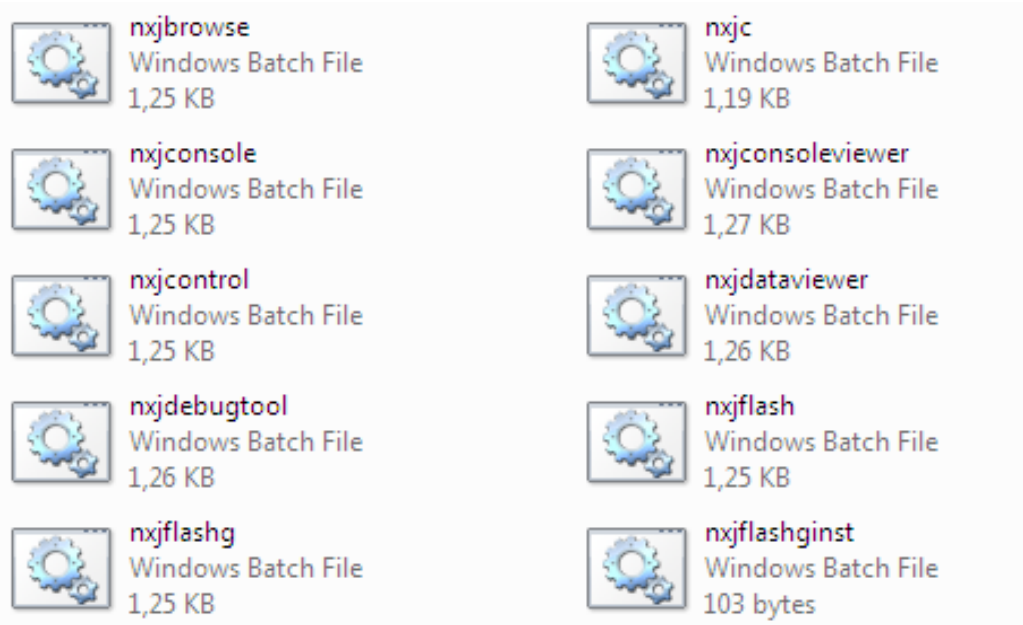


Τώρα μπορούμε να χρησιμοποιήσουμε τα παραδείγματα για να εξερευνήσουμε τις δυνατότητες του ρομπότ μας.

ΠΑΡΑΡΤΗΜΑ ΙΙΙ – ΕΡΓΑΛΕΙΑ LEJOS

Εργαλεία που μας παρέχονται με το LeJOS (5)

Μετά την εγκατάσταση του LeJOS, που μπορούμε να κάνουμε ακολουθώντας τα βήματα του παραρτήματος ΙΙ, αν πάμε στον φάκελο lejos_nxj → bin θα δούμε ότι περιέχει αρχεία τύπου Windows Batch File (δηλαδή αρχεία με κατάληξη .bat). Όλα τα αρχεία αυτά είναι χρήσιμα εργαλεία που μας παρέχει το LeJOS για να είναι ακόμη πιο φιλικό στον χρήστη.



Τα εργαλεία που μας παρέχονται είναι περιληπτικά:

- nxjflash - φορτώνει το firmware και το μενού του LeJOS στο NXT τούβλο
- nxjflashg - διεπαφή χρήστη (user interface) που φορτώνει το firmware και το μενού του LeJOS στο NXT τούβλο
- nxjc - κάνει compile ένα πρόγραμμα Java για το LeJOS NXJ
- nxj - συνδέει, ανεβάζει και προαιρετικά τρέχει ένα LeJOS NXJ πρόγραμμα
- nxjlink - συνδέει ένα πρόγραμμα
- nxjupload - ανεβάζει και προαιρετικά τρέχει το πρόγραμμα
- nxjbrowse - διεπαφή χρήστη για την εξερεύνηση, ανέβασμα και διαγραφή των αρχείων στο NXT τούβλο
- nxjmonitor - διεπαφή χρήστη για την απομακρυσμένη παρακολούθηση και εντοπισμό του ρομπότ μας μέσω του Bluetooth

- nxjconsole - επιτρέπει τον απομακρυσμένο εντοπισμό σφαλμάτων μέσω USB ή Bluetooth

- nxjconsoleviewer - διεπαφή χρήστη για τον απομακρυσμένο εντοπισμό σφαλμάτων μέσω USB ή Bluetooth

- nxjdataviewer - απομακρυσμένος ελεγκτής DataLogger αρχείων

Ακολουθεί μία σύντομη περιγραφή των εργαλείων που έχουν εγκατασταθεί και αναφέρθηκαν.

- nxjflash και nxjflashg:

Αν έχουμε ήδη εγκαταστήσει το LeJOS στο NXT τούβλο μας, τότε δε θα χρειαστούμε αυτό το εργαλείο παρά μόνο σε περίπτωση που αναγκαστούμε να κάνουμε hard reset (δηλαδή μία διαδικασία που θα επαναφέρει το τούβλο μας στην κατάσταση στην οποία το αγοράσαμε). Μπορεί να αναγκαστούμε να κάνουμε hard reset αν το ρομπότ μας δεν ανταποκρίνεται, δηλαδή έχει κολλήσει, με αποτέλεσμα να διαγραφούν τα πάντα άρα και το LeJOS. Η εντολή nxj_flash επικοινωνεί μέσω USB με τον επεξεργαστή του NXT τούβλου και αναλαμβάνει να εγκαταστήσει το LeJOS firmware στη μνήμη του τούβλου. Με την εκτέλεση του προηγούμενα αρχεία που είναι αποθηκευμένα είναι πολύ πιθανό να χαθούν για πάντα. Το nxjflashg κάνει την ίδια ακριβώς δουλειά με το nxjflash με τη διαφορά ότι διαθέτει γραφικό περιβάλλον που κάνει τη χρήση του πιο εύκολη.

Σημείωση: Οποιοδήποτε firmware και να χρησιμοποιούμε υπάρχει περίπτωση για κάποιο λόγο να πάψει να λειτουργεί επειδή σβήστηκε από τη μνήμη. Σε αυτή την περίπτωση όταν θα ανοίγουμε τον επεξεργαστή, αυτός δεν θα εμφανίζει τίποτα στην οθόνη και θα κάνει έναν επαναλαμβανόμενο λεπτό ήχο. Η μόνη λύση για αυτό το πρόβλημα είναι να ξαναπεραστεί το firmware με αυτό το πρόγραμμα.

- nxjbrowse

Το πρόγραμμα αυτό εμφανίζει μία λίστα με τα διαθέσιμα NXT στα οποία μπορεί να γίνει σύνδεση είτε μέσω USB είτε με Bluetooth. Αφού ο χρήστης επιλέξει την επιθυμητή συσκευή και αφού γίνει η σύνδεση με επιτυχία, εμφανίζεται ένα νέο παράθυρο που περιέχει λίστα όλων των προγραμμάτων - αρχείων που υπάρχουν στο NXT. Αυτή η γραφική διεπαφή μπορεί να χρησιμοποιηθεί για να γίνει upload, download ή ακόμη και διαγραφή προγραμμάτων. Επίσης, είναι δυνατόν να δοθεί εντολή για την εκκίνηση εκτέλεσης ενός προγράμματος καθώς ακόμη και για defrag NxjBrowse δηλαδή μεταφορά των προγραμμάτων στην αρχή της μνήμης flash του NXT.

- nxjc, nxjlink, nxjupload, nxj

Το nxjc κάνει compile το πρόγραμμα που είναι γραμμένο σε Java. Το αποτέλεσμα αυτής της εντολής είναι η δημιουργία ενός αντίστοιχου αρχείου κατάληξης class, το οποίο περιέχει Java Bytecode. Για παράδειγμα:

$ nxjc main.java file2.java

$ ls

main.class main.java file2.class file2.java

Το nxjlink κάνει link τα αρχεία που περιέχουν Java Bytecode με την βιβλιοθήκη του LeJOS. Το αρχείο εξόδου είναι και το τελικό πρόγραμμα. Σημειώνουμε ότι η παράμετρος του nxjlink πρέπει να είναι μόνο εκείνο το class αρχείο που περιέχει την main. Επίσης υποχρεωτική παράμετρος είναι ένα αρχείο εξόδου. Παράδειγμα χρήσης:

$ *nxjlink* main -o main.nxj

Το nxjupload αναλαμβάνει να ανεβάσει στο NXT το εκτελέσιμο πρόγραμμα, στην περίπτωση μας το “main.nxj”. Το upload του αρχείου μπορεί να γίνει είτε μέσω Bluetooth είτε μέσω USB, ανάλογα την παράμετρο που θέτουμε:

$ nxjupload main.nxj --usb

Τα τρία παραπάνω βήματα μπορούν να εκτελεστούν αυτόματα δίνοντας μόνο μία εντολή, χρησιμοποιώντας την nxj. Συγκεκριμένα με την παρακάτω εντολή δημιουργούμε το αρχείο “main.nxj” και στη συνέχεια γίνεται upload:

$ nxj main

- nxjmonitor

Το nxjmonitor μπορεί να χρησιμοποιηθεί για να παρακολουθείται η εκτέλεση ενός προγράμματος στο NXT. Το πρόγραμμα που παρακολουθείται θα πρέπει να τρέχει το νήμα LCPBTResponter. Όταν στο NXT δεν εκτελεί κώδικα το nxjmonitor μπορεί να στέλνει εντολές. Έτσι αποτελεί ένα καλό τρόπο για τον πειραματισμό του χρήστη. Άλλες πληροφορίες που γίνονται διαθέσιμες από αυτό το πρόγραμμα είναι η κατάσταση της μπαταρίας και των αισθητήρων, συμπεριλαμβανομένου και των ταχομετρητών των σερβομοτέρ.

- nxjconsole/nxjconsoleviewer

Αυτά τα δύο προγράμματα είναι κονσόλες που εμφανίζουν μηνύματα από προγράμματα που εκτελούνται στο NXT. Αναλυτικότερα, ο κώδικας (του NXT) για την αποστολή μηνυμάτων έχει ως εξής:

Rconsole.openBluetooth(); ή Rconsole.openUSB(0); ή RConsole.open();

Rconsole.println("Hello World.");

Rconsole.close();

Αν τρέξουμε τον παραπάνω κώδικα, το NXT θα σταματήσει στην πρώτη γραμμή μέχρι να εκτελέσουμε στο PC το nxjconsole. Σε αυτό μάλιστα θα εμφανιστεί το μήνυμα Hello World. Το nxjconsoleviewer είναι ίδιο με το nxjconsole με τη διαφορά ότι έχει γραφικό περιβάλλον.

Σημείωση: Αν το εκτελέσιμο στο NXT τερματίσει για κάποιο λόγο προτού κάνει close την σύνδεση, θα υπάρξει πρόβλημα την επόμενη φορά που θα εκτελεστεί ο ίδιος κώδικας. Για να τα αποφύγουμε αρκεί να κλείσουμε και να ανοίξουμε το NXT.

- nxjdataviewer

Η λειτουργία αυτού του προγράμματος είναι αντίστοιχη του nxjconsole που είδαμε παραπάνω. Γίνεται δηλαδή η ανταλλαγή δεδομένων μεταξύ NXT με το PC και βοηθάει κυρίως στην απασφαλμάτωση. Λειτουργεί ως εξής: επιλέγουμε στον κώδικα NXT ένα σύνολο μεταβλητών το οποίο επιθυμούμε να αποσταλεί στο PC και στη συνέχεια το μεταδίδουμε με την εντολή transmit. Πχ:

Datalogger dl = new Datalogger();

dl.writeLog(4.0); dl.writeLog(1.0);

dl.transmit();

Αν εκτελέσουμε τον παραπάνω κώδικα, το NXT θα σταματήσει στην πρώτη γραμμή και θα περιμένει να εκτελέσουμε στο pc το nxjdataviewer και να κάνουμε connect. Τότε θα εμφανιστούν οι τιμές 4.0 και 1.0.

ΠΑΡΑΡΤΗΜΑ IV - ΚΩΔΙΚΑΣ

```
import java.io.File;
import lejos.robotics.Color;
import lejos.robotics.subsumption.*;
import lejos.nxt.*;
import lejos.robotics.navigation.DifferentialPilot;

public class MyDog {
	public static UltrasonicSensor distance = new UltrasonicSensor(
			SensorPort.S4);
	public static ColorSensor color = new ColorSensor(SensorPort.S2);
	public static ColorSensor.Color clr;
	public static DifferentialPilot pilot;
	public static int mycolor = -1;
	public static String[] colors = { "RED", "GREEN", "BLUE", "YELLOW", "none",
			"none", "WHITE", "BLACK" };
	public static boolean retry = false;
	public static File filename = new File("bark.wav");

	public static void main(String[] args) throws Exception {
		color.setFloodlight(Color.WHITE);
		pilot = new DifferentialPilot(4.32f, 7.9f, Motor.A, Motor.B, true);
		pilot.setRotateSpeed(30);
		Behavior b1 = new MoveForward();
		Behavior b2 = new FindWallReadColor();
		Behavior b3 = new DoAction();
		Behavior[] behav = { b1, b2, b3 };
		Arbitrator arbi = new Arbitrator(behav);
		arbi.start();
	}

	public static void CountDistance_Turn() {
		double left;
		double right;
		Motor.C.setSpeed(100);
		Motor.C.rotate(100);
		LCD.clear(7);
		LCD.drawString("Get distance", 0, 1);
		left = MyDog.distance.getDistance();
		LCD.drawString("LEFT: " + Double.toString(left), 0, 5);
		Motor.C.rotate(-200);
		right = MyDog.distance.getDistance();
		LCD.drawString("RIGHT: " + Double.toString(right), 0, 6);
```

```
            Motor.C.rotate(100);
            LCD.clear();
            if (left >= right) {
                  LCD.drawString("turning LEFT!", 0, 4);
                  MyDog.pilot.rotate(-90);
            } else {
                  LCD.drawString("turning RIGHT!", 0, 4);
                  MyDog.pilot.rotate(90);
            }
            MyDog.mycolor = -1;
            LCD.clear();
      }

      public static void Turn(int a, int b) {
            int left;
            int right;
            left = a;
            right = b;
            LCD.clear();
            if (left > right) {
                  LCD.drawString("LEFT arrow!!!", 0, 3);
                  LCD.drawString("turning LEFT!", 0, 4);
                  MyDog.pilot.rotate(-90);
            } else {
                  LCD.drawString("RIGHT arrow!!!", 0, 3);
                  LCD.drawString("turning RIGHT!", 0, 4);
                  MyDog.pilot.rotate(90);
            }
            MyDog.mycolor = -1;
      }

      public static void ReadArrow() {
            int[] colorarray = new int[8];
            int leftColor = 0;
            int rightColor = 0;
            Motor.C.setSpeed(20);
            Motor.C.rotate(28);
            LCD.drawString("something found!", 0, 0);
            LCD.drawString("try read arrow...", 0, 3);
            for (int i = 0; i < 8; i++) {
                  clr = MyDog.color.getColor();
```

```
if (clr.getColor() == 1)
                colorarray[i] = clr.getColor();
            else if (clr.getColor() == 2)
                colorarray[i] = 100;
            else if (clr.getColor() == 3)
                colorarray[i] = 10000;
            else
                colorarray[i] = 0;
            Motor.C.rotate(-7);
        }
        Motor.C.rotate(28);
        LCD.clear();
        leftColor = colorarray[0] + colorarray[1] + colorarray[2]
                + colorarray[3];
        rightColor = colorarray[4] + colorarray[5] + colorarray[6]
                + colorarray[7];
        LCD.drawInt(leftColor, 6, 6);
        LCD.drawInt(rightColor, 7, 7);
        if (leftColor == rightColor) {
            if (MyDog.retry == false) {
                MyDog.retry = true;
                MyDog.ReadArrow();
            } else {
                MyDog.retry = false;
                LCD.drawString("No arrow found!", 0, 0);
                MyDog.CountDistance_Turn();
            }
        } else
            MyDog.Turn(leftColor, rightColor);
    }
}

class MoveForward implements Behavior {
    boolean suppressed = false;

    public boolean takeControl() {
        return true;
    }

    public void suppress() {
        suppressed = true;
    }
```

```
	public void action() {
		LCD.clear();
		suppressed = false;
		MyDog.pilot.setTravelSpeed(5);
		MyDog.pilot.forward();
		while (!suppressed) {
			Thread.yield();
		}
		MyDog.pilot.stop();
	}
}

class FindWallReadColor implements Behavior {
	boolean suppressed = false;

	public boolean takeControl() {
		return MyDog.distance.getDistance() <= 6;
	}

	public void suppress() {
		suppressed = true;
	}

	public void action() {
		suppressed = false;
		LCD.clear();
		MyDog.clr = MyDog.color.getColor();
		LCD.drawString("     COLOR:  ", 0, 1);
		LCD.drawString(MyDog.colors[MyDog.clr.getColor()], 7, 1);
		while (!suppressed) {
			MyDog.mycolor = MyDog.clr.getColor();
		}
	}
}

class DoAction implements Behavior {

	public boolean takeControl() {
		return MyDog.mycolor != -1;
	}

	public void suppress() {
	}
```

```
	public void action() {
		LCD.clear();
		if (MyDog.mycolor == 6) { // white
			LCD.drawString("WHITE wall found", 0, 0);
			MyDog.CountDistance_Turn();
		} else if (MyDog.mycolor == 0) { // red

			LCD.drawString("THE END!", 4, 5);
			Sound.playSample(MyDog.filename, 100);
			for (int i = 0; i < 3; i++) {
				MyDog.color.setFloodlight(Color.BLUE);
				MyDog.color.setFloodlight(Color.RED);
				MyDog.color.setFloodlight(Color.GREEN);
			}
			NXT.shutDown();
		} else {
			MyDog.mycolor = -1;

			MyDog.ReadArrow();
		}
	}
}
```