ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

# Σχεδίαση Λογισμικού και Υλικού για Ρομπότ Επίλυσης Λαβυρίνθων

Μακρόπουλος Νικόλαος

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Καθηγητής Διονύσιος Πνευματικάτος (συνεπιβλέπων)
Καθηγητής Μιχαήλ Γ. Λαγουδάκης (συνεπιβλέπων)
Επίκουρος Καθηγητής Γεώργιος Χαλκιαδάκης

Χανιά, Νοέμβρης 2013

# Περίληψη

Το θέμα της παρούσας διπλωματικής εργασίας είναι η σχεδίαση και η κατασκευή του υλικού και του λογισμικού ενός μικρού ρομποτικού οχήματος βασισμένου στην πλατφόρμα arduino, ικανού να επιλύει ένα λαβύρινθο, δηλαδή να εξερευνά και να χαρτογραφεί πλήρως έναν λαβύρινθο, ώστε κατόπιν να είναι σε θέση να πλοηγηθεί προς οποιοδήποτε σημείο του σχεδιάζοντας και ακολουθώντας βέλτιστες (πιο σύντομες) διαδρομές. Το ρομποτικό όχημα διαστάσεων 15cm x 10cm κατασκευάστηκε εξ αρχής χρησιμοποιώντας ηλεκτρονικά υλικά γενικού σκοπού και εξοπλίσθηκε με δύο τροχούς που παρέχουν διαφορική κίνηση, με μία συστάδα αισθητήρων φωτός για την αναγνώριση γραμμών στο δάπεδο, και με ασύρματη σύνδεση RF για επικοινωνία με υπολογιστή. Ο έλεγχος όλων των ηλεκτρονικών στοιχείων πάνω στο όχημα επιτυγχάνεται μέσω της πλατφόρμας arduino, η οποία προγραμματίσθηκε κατάλληλα για να τα υποστηρίζει. Λόγω της περιορισμένης υπολογιστικής ικανότητας της πλατφόρμας arduino, προτείνουμε μια προσέγγιση συνεργατικής λειτουργίας του ρομποτικού οχήματος μέσω της ασύρματης σύνδεσης με λογισμικό που εκτελείται σε απομακρυσμένο υπολογιστή και έχει υλοποιηθεί στο περιβάλλον processing. Το λογισμικό αυτό είναι υπεύθυνο για την καταγραφή του χάρτη και για τον σχεδιασμό όλων των διαδρομών, ενώ παράλληλα παρέχει και γραφικό περιβάλλον όπου ο χρήστης μπορεί να παρακολουθήσει σε πραγματικό χρόνο την πορεία της εξερεύνησης, καθώς και την αποτύπωση του λαβυρίνθου, ενώ έχει την δυνατότητα να υποδείξει σημεία του λαβυρίνθου όπου επιθυμεί να κινηθεί το όχημα. Για αποτελεσματικότερη εξερεύνηση και χαρτογράφηση προτείνεται η χρήση του αλγόριθμου του Tremaux με μια ευριστική μέθοδο που χρησιμοποιεί την τοπικότητα ως κριτήριο, σε συνδυασμό με τον αλγόριθμο του Dijkstra ο
οποίος λύνει το πρόβλημα της εύρεσης βέλτιστων (συντομότερων) μονοπατιών από την τρέχουσα θέση του ρομπότ στον λαβύρινθο προς όλες τις άλλες (single source shortest path). Το υλοποιημένο σύστημα ρομπότ-υπολογιστή δοκιμάστηκε με επιτυχία σε λαβυρίνθους σχεδιασμένους με μαύρες γραμμές σε λευκό δάπεδο που περιείχαν πολλούς κόμβους, κύκλους και εναλλακτικές διαδρομές.

Η εργασία αυτή αφιερώνεται στον Θανάση, την Αλίκη, την Τατιάνα γιατί θα μου δείχνουν για πάντα τον δρόμο...

Αφιερώνεται στους συντρόφους και τους φίλους που με έμαθαν στη ζωή να στέκομαι όρθιος και να παλεύω...

Τέλος με ιδιαίτερη αγάπη, αφιερώνεται στην παρέα που ξενύχτησε στο πλάι μου, τον Γιάννη, τον Πάνο και προφανώς την Αρχοντούλα...

# Περιεχόμενα

# Κεφάλαιο 1

## Εισαγωγή

Πολύ πριν αποφασιστεί το θέμα της εργασίας, επιδιώκαμε να βρούμε ένα τρόπο να συνδυάσουμε την απτή φύση του υλικού με την υπολογιστική ισχύ του λογισμικού και να δούμε το αποτέλεσμα σε ένα σενάριο, που θα ήταν πρόκληση και για τα δύο αυτά πεδία. Αντικειμενικά καταλήξαμε στο χώρο της ρομποτικής, τον κατεξοχήν χώρο όπου λογισμικό και υλικό γίνονται "ένα" και αρχίζουν να προσεγγίζουν είτε την ανθρώπινη συμπεριφορά είτε έχουν σαν αποτέλεσμα μηχανές και ρομπότ με ένα πολύ μεγάλο εύρος δυνατοτήτων.

Το σενάριο θα έπρεπε λοιπόν, να θέτει ένα αλγοριθμικό πρόβλημα από την μία και από την άλλη να θέτει το ερώτημα του ποιος θα εκτελεί τις κινήσεις που ορίζει αυτός ο αλγόριθμος. Δεν ήταν βέβαια μόνο δικιά μας αυτή η αγωνία οπότε εμπνευστήκαμε από πολλές διαφορετικές μικρές η μεγαλύτερες εργασίες που είχαν τον ίδιο προσανατολισμό. Καταλήξαμε λοιπόν στην κατασκευή ενός μικρού ρομπότ το οποίο θα είχε σαν αποστολή κυρίως να λύνει λαβυρίνθους. Αυτό γιατί από την μία η έννοια του λαβύρινθου είχε από πάντα μια μαθηματική διάσταση και άρα θέτει ένα αλγοριθμικό πρόβλημα και από την άλλη γιατί σκεφτήκαμε ότι σε ένα τέτοιο περιβάλλον που θα φτιάχναμε εμείς, θα μπορούσαμε ίσως να θέσουμε και άλλα ζητήματα με τα οποία συνήθως ασχολείται η ρομποτική.

Έτσι λοιπόν αποφασίσαμε ότι θέλουμε να φτιάξουμε ένα μικρό ρομπότ που θα φαίνεται ότι σκέφτεται έξυπνα και θα λύνει λαβυρίνθους όσο δύσκολοι και περίπλοκοι και να είναι και από την άλλη θέλαμε να μπορεί να αποκτάει γνώση για το περιβάλλον του και να το χαρτογραφεί. Ύστερα σκεφτήκαμε ότι θα ήταν άλλη μια πρόκληση να μπορούμε να βλέπουμε και γραφικά αυτό που χαρτογραφεί και αν μπορούμε να του δίνουμε εντολές για να αρχίσει να κάνει πλοήγηση στο περιβάλλον του βρίσκοντας κάθε φορά τη συντομότερη διαδρομή που πρέπει να ακολουθήσει.

Έμπνευση για την υλοποίηση αυτής της εργασίας ήταν αρχικά η ενασχόληση σε προπτυχιακό επίπεδο με την *Τεχνητή Νοημοσύνη* και τη *Ρομποτική* και ύστερα η γνωριμία με το περιβάλλον του arduino, που παρέχει στον χρήστη μία εύκολη και παράλληλα μία πολύ ισχυρή μέθοδο να αναπτύξει εφαρμογές σε επίπεδο υλικού.

Η δουλειά μας χωρίζεται σε τρεις κύριες ενότητες. Η πρώτη ήταν η κατασκευή του ρομπότ, ακολουθούμενη βέβαια από την διαδικασία έρευνας για τα ποια θα είναι τα κατάλληλα ηλεκτρονικά και μηχανικά μέρη που θα το αποτελούν. Η δεύτερη ενότητα περιλαμβάνει τον προγραμματισμό του arduino board για τον έλεγχο της κίνησης, την επεξεργασία των δεδομένων από το περιβάλλον και την επικοινωνία με τον υπολογιστή. Η τρίτη και τελική ενότητα αφορά την υλοποίηση των αλγορίθμων απόφασης, δηλαδή το κομμάτι αυτό που επεξεργάζεται τόσο την τρέχουσα κίνηση του οχήματος, όσο και τις μελλοντικές, αποφασίζοντας για κάθε περίπτωση ποια είναι η καλύτερη δυνατή επιλογή για να εκτελέσει το ρομπότ. Αυτό μπορεί να σημαίνει ότι χρειάζεται να παρθεί μια απόφαση για να γλιτώσει το ρομπότ να επαναλαμβάνει διαδρομές που ήδη έχει διασχίσει ή να χρειάζεται μια απόφαση για το ποια είναι η συντομότερη διαδρομή ώστε να φτάσει σε έναν κόμβο που θέλει να εξερευνήσει. Στην ενότητα αυτή υλοποιούνται και όλες οι βοηθητικές συναρτήσεις για την επεξεργασία των δεδομένων που στέλνει το ρομπότ, όσο και οι συναρτήσεις που υλοποιούν το γραφικό περιβάλλον αλλά και το κομμάτι αυτό που είναι υπεύθυνο για την αξιόπιστη επικοινωνία μεταξύ του ρομπότ και του υπολογιστή.

Επιδιώκουμε μέσα από αυτήν την εργασία να προτείνουμε έναν τρόπο με τον οποίο καταφέρνουμε να συνδυάσουμε προβλήματα αλγοριθμικής και ρομποτικής φύσης. Υπάρχει πάντα η δυνατότητα να θέσουμε ακόμα μεγαλύτερους στόχους και θέλαμε το αποτέλεσμα της εργασίας μας είναι μια βάση προς αυτή την κατεύθυνση.

## 1.1 Ζητήματα που θέτουμε

Ένα από τα βασικά ζητήματα της εργασίας μας είναι ο τρόπος με τον οποίο αποφασίσαμε να διανέμουμε τον κώδικα που “τρέχει” πίσω από οτιδήποτε βλέπουμε. Συνήθως, σε πολλά παραδείγματα ρομποτικών κατασκευών ο κώδικας για το κομμάτι που αφορά τον σχεδιασμό των κινήσεων και αυτός που αφορά την κίνηση αυτή καθαυτή υλοποιούνται μαζί χρησιμοποιώντας την ίδια κοινή μνήμη και τους ίδιους κοινούς πόρους. Αυτό σημαίνει ότι δημιουργείται μια εσωτερική ιεραρχία στον κώδικα μεταξύ των διάφορων συναρτήσεων του, ενώ αν το υλικό που έχουμε στη διάθεση μας θέτει κάποιους περιοριστικούς παράγοντες όπως συχνά συμβαίνει, τότε ίσως να έχουμε και κάποιο κόστος που αφορά την υλοποίηση, όπως πχ την ταχύτητα των κινήσεων . Στη δικιά μας προσέγγιση αυτό που επιδιώξαμε ήταν να διαχωρίσουμε τον κώδικα που αφορά τις χαμηλότερες σε ιεραρχία κινήσεις που κάνει ένα ρομπότ (συλλογή πληροφοριών από το περιβάλλον) από το κομμάτι εκείνο που βρίσκεται σε πιο υψηλό ιεραρχικό επίπεδο και είναι υπεύθυνο για την επίλυση του λαβυρίνθου ή για την εύρεση των συντομότερων μονοπατιών. Για την πρώτη περίπτωση ο κώδικας “τρέχει” πάνω στο ρομπότ και είναι υπεύθυνος για να ελέγχει την κίνηση του ή να

διαβάζει τους αισθητήρες ενώ από την άλλη το κομμάτι του κώδικα που είναι απαιτητικό σε μνήμη και ισχύ "τρέχει" στον υπολογιστή. Κάθε ένα από τα κομμάτια, δεν μπορεί να λειτουργήσει από μόνο του, οπότε το σύστημα μας αποκτά αυτονομία μόνο αν το δούμε σαν σύνολο. Τελικά αυτό που υλοποιήσαμε δουλεύει και τώρα έχουμε δύο διαφορετικά προγράμματα να συνεργάζονται μεταξύ τους δίνοντας ένα αποτέλεσμα που επιδιώκαμε.

Ένα δεύτερο θέμα στο οποίο προσπαθήσαμε να συμβάλλουμε με τη δικιά μας εργασία είναι να βρούμε έναν τρόπο να δούμε το πρόβλημα της επίλυσης του λαβυρίνθου μαζί με αυτό της χαρτογράφησης του, σαν ένα ενιαίο πρόβλημα. Αυτό σημαίνει ότι θέλουμε να ελαχιστοποιήσουμε τις συνολικές κινήσεις οι οποίες εκτελούνται μέχρι να ικανοποιηθούν όλες οι κινήσεις. Όταν δηλαδή, έχει βρεθεί σημείο εξόδου από τον λαβύρινθο και έχουν εξερευνηθεί όλα τα σημεία του. Προσπαθήσαμε να βρούμε έναν αλγόριθμο που θα μας βοηθούσε να το κάνουμε αυτό αλλά παράλληλα προσπαθήσαμε να βρούμε και άλλους τρόπους για να βελτιώσουμε το αποτέλεσμα που θα παίρναμε. Η μέθοδος που χρησιμοποιήσαμε πιστεύουμε ότι βοηθάει πολύ στη λύση του προβλήματος.

Ένα τελευταίο στοιχείο συμβολής είναι ότι μέσα από το γραφικό περιβάλλον που υλοποιήσαμε δίνεται η δυνατότητα στο χρήστη και να παρακολουθεί σε πραγματικό χρόνο την διαμόρφωση του χάρτη αλλά και να ορίζει αυτός τελικά σημεία-στόχους.

## 1.2 Επισκόπηση κειμένου

Στο κεφάλαιο 2 παρουσιάζουμε τα βασικά ψηφιακά και μηχανικά μέρη που αποτελούν το ρομποτικό όχημα που υλοποιήσαμε και περιγράφουμε τις βασικές αρχές λειτουργίας τους. Κατόπιν κάνουμε μια αναφορά στον τρόπο με τον οποίο αναπαριστούμε μαθηματικά τον λαβύρινθο και στο ποιοι είναι οι αλγόριθμοι με τον οποίους μπορούμε να τον επιλύσουμε. Στο κεφάλαιο 3 περιγράφουμε τα προβλήματα στη γενική τους μορφή και τα συνδέουμε με τα δικά μας ζητούμενα. Παρουσιάζουμε κάποιες εργασίες αντίστοιχες σε ένα βαθμό με την δικιά μας και τονίζουμε διαφορές και ομοιότητες. Στο κεφάλαιο 4 γίνεται η περιγραφή της δουλειάς μας και περιγράφουμε για όλα τα θέματα που μας απασχόλησαν τον τρόπο που τα αντιμετωπίσαμε. Στο κεφάλαιο 5 παρουσιάζουμε τα αποτελέσματα της δουλειάς μας και τέλος, στο κεφάλαιο 6 προτείνουμε κάποια θέματα και ιδέες για μελλοντική εργασία.

# Κεφάλαιο 2

## Υπόβαθρο

## 2.1 Η πλατφόρμα του Arduino και το περιβάλλον ανάπτυξης

Όταν αναφερόμαστε στο **arduino** [1] εννοούμε μία επαναπρογραμματιζόμενη ψηφιακή πλακέτα με εισόδους και εξόδους, είτε αναλογικές είτε ψηφιακές, έναν μικροελεγκτή που είναι η καρδιά της πλακέτας, αλλά παράλληλα αναφερόμαστε και στο προγραμματιστικό περιβάλλον που στηρίζεται (**arduino IDE**), το οποίο είναι συνεχώς εξελισσόμενο και έχει φτάσει σε σημείο να μπορεί να ανταποκριθεί στις απαιτήσεις χιλιάδων προγραμματιστών είτε σε ερασιτεχνικό είτε σε επαγγελματικό επίπεδο. Η πλακέτα όσο και το προγραμματιστικό περιβάλλον του arduino ανήκουν στον χώρο του open-source software and hardware και υποστηρίζουν την ιδέα του **prototyping**. Αυτό περιγράφει, τη δυνατότητα να μπορεί μια ιδέα να μετεξελιχθεί γρήγορα σε κάτι υλοποιήσιμο, να μπορεί να κοινοποιηθεί στην κοινότητα και από εκεί και πέρα να εξελιχθεί ή να αποτελέσει τη βάση για κάτι καινούριο. Οι εφαρμογές που βασίζονται στο arduino μπορούν είτε να γίνουν αυτόνομες (με τις κατάλληλες προϋποθέσεις όπως τροφοδοσία), είτε μπορούν να "τρέχουν" σε συνεργασία με κάποιο πρόγραμμα στον υπολογιστή.

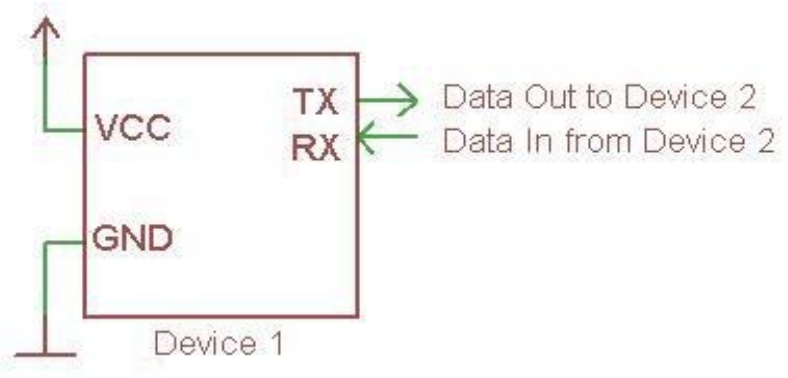
### 2.1.1 Τα τεχνικά χαρακτηριστικά του arduino

Στο επίπεδο του υλικού διακρίνουμε μια πλακέτα ανοιχτού κώδικα (open source hardware) η οποία έχει έναν 8-bit Atmel AVR μικροελεγκτή [2] ή έναν 32 bit Atmel ARM [3], έναν ηλεκτρονικό ταλαντωτή ή έναν κεραμικό πυκνωτή για παραγωγή 16MHz συχνότητας, ένα ρυθμιστή τάσης στα 5V και συνήθως δεκατέσσερις (14) ψηφιακές εισόδους/εξόδους. Τα I/O pins έχουν και τη δυνατότητα να παρέχουν σαν εξόδους Παλμούς Διαμόρφωσης Πλάτους (PWM) και επίσης 6 από αυτά έχουν τη

δυνατότητα να διαβάζουν αναλογικές εισόδους χρησιμοποιώντας έναν αναλογικό σε ψηφιακό μετατροπέα (ADC). Όλες οι πλακέτες arduino είναι προγραμματισμένες για σειριακή επικοινωνία (serial communication) με το πρότυπο RS-232, επικοινωνία δηλαδή όπου τα δεδομένα είναι μια ροή από 0 και 1 σε ακολουθία. Ο χαρακτηρισμός "σειριακή" καταδεικνύει τον τρόπο με τον οποίο γίνεται η αποστολή και η ανάγνωση των δεδομένων και τονίζει την διαφορά μεταξύ της παράλληλης ροής των δεδομένων που χρησιμοποιεί μια παράλληλη επικοινωνία. Ο προγραμματισμός της πλακέτας συνήθως γίνεται μέσω μιας θύρας USB, και αυτό επιτυγχάνεται χρησιμοποιώντας έναν μετατροπέα USB to Serial. Υπάρχουν βέβαια και εναλλακτικές μέθοδοι ασύρματου προγραμματισμού της πλακέτας όπως χρησιμοποιούμε στην παρούσα εργασία και αυτό γίνεται μέσω συσκευών Bluetooth ή ραδιοσυχνοτήτων (RF modules) [4].

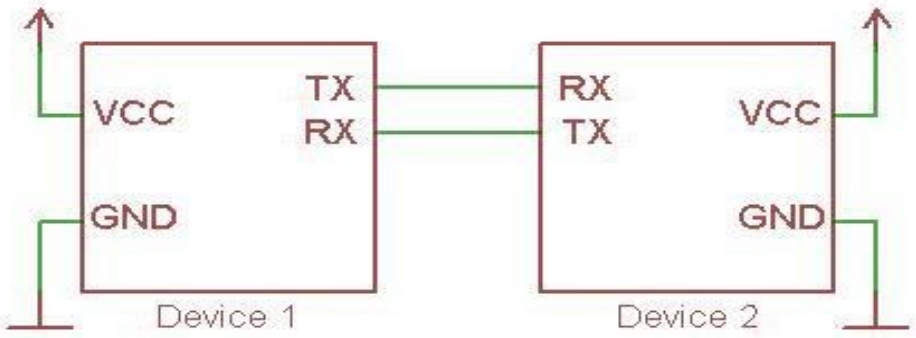
### 2.1.2 Η σειριακή επικοινωνία στο Arduino

Κάθε πλακέτα arduino έχει μία σειριακή θύρα γνωστή και ως UART ή USART (Universal Asynchronous Receiver/Transmitter) η οποία λειτουργεί χρησιμοποιώντας ένα πρωτόκολλο ασύγχρονης επικοινωνίας. Όταν μια συσκευή επικοινωνεί χρησιμοποιώντας ένα τέτοιο πρωτόκολλο, σημαίνει ότι η θύρα δέχεται εισερχόμενα δεδομένα από τη γραμμή Rx (receiver) η αλλιώς το Pin0 και στέλνει δεδομένα χρησιμοποιώντας τη γραμμή Tx (tranceiver) η αλλιώς το Pin1 της πλακέτας. Στην εικόνα 1 που ακολουθεί δίνουμε ένα παράδειγμα με τη ροή δεδομένων σε μια θύρα USART



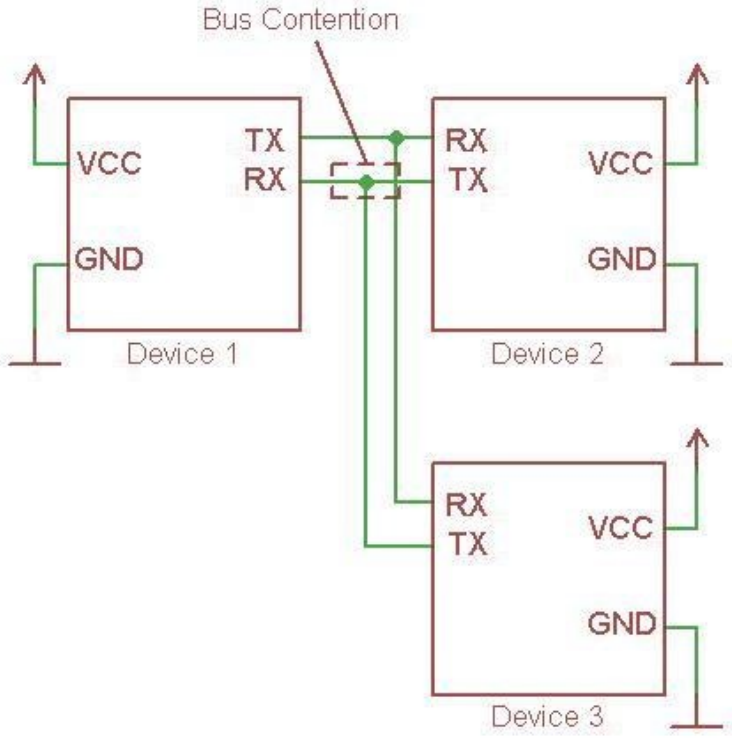

Εικ.1 Ροή δεδομένων από και προς τη θύρα USART

Όταν δύο συσκευές ανταλλάσουν δεδομένα μεταξύ τους χρησιμοποιώντας το ίδιο πρωτόκολλο, η γραμμή Tx της μίας συσκευής συνδέεται με τη γραμμή Rx της δεύτερης συσκευής και αντίστοιχα η γραμμή Rx της πρώτης, με τη γραμμή Tx της δεύτερης. Στην εικόνα 2 φαίνεται η σωστή διάταξη μεταξύ δύο USART συσκευών.




Εικ.2 Η σωστή διάταξη για ασύγχρονη επικοινωνία μεταξύ δύο συσκευών.

Κάθε φορά, μόνο μία συσκευή μπορεί να έχει τον έλεγχο μιας γραμμής USART. Αν δύο ή παραπάνω συσκευές προσπαθούν να στείλουν δεδομένα στη γραμμή Rx μίας συσκευής τότε παράγεται ένα λάθος που ονομάζεται “bus contention”. Αυτό έχει σαν αποτέλεσμα να υπάρχει ανταγωνισμός για το ποια από τις δύο συσκεύες θα καταφέρει να επικοινωνήσει με την τρίτη με αποτέλεσμα να διακοπεί συνολικά η επικοινωνία. Στην εικόνα 3 που ακολουθεί, δίνουμε ένα παράδειγμα τέτοιου λάθους.




Εικ.3 Δύο συσκευές ταυτόχρονα, στέλνουν δεδομένα στην γραμμή Rx μιας τρίτης

## 2.1.3 Οι είσοδοι/έξοδοι του Arduino

Πέρα από τα pin0 (Rx) και pin1 (Tx) που αναφέραμε τα περισσότερα pins χρησιμοποιούνται για λειτουργίες αναλογικής και ψηφιακής ανάγνωσης ή εγγραφής δεδομένων. Στη συνέχεια θα κάνουμε μια περιγραφή των αναλογικών εισόδων σε μια πλακέτα arduino (Atmega8, Atmega168, Atmega328) και θα δείξουμε πώς τις χρησιμοποιούμε για την τεχνική της Διαμόρφωσης Πλάτους Παλμών.

## Μετατροπέας Αναλογικού σήματος σε Ψηφιακό (A/D Converter)

Οι μικροελεγκτές της οικογένειας Atmega που χρησιμοποιούνται στα arduino χρησιμοποιούν έναν 6-κάναλο ADC ανάλυσης 10 bit. Αυτό δίνει τη δυνατότητα να μπορεί να διαβαστεί μια αναλογική είσοδος και αυτή να μεταφραστεί σε μια ψηφιακή τιμή από 0-1023. Τα αναλογικά Pins μπορούν όμως να χρησιμοποιηθούν και ως είσοδοι/έξοδοι γενικού σκοπού (General Purpose I/O).

## Analog Pin mapping

Κάθε pin της πλακέτας έχει μια κωδική ονομασία αποτελείται από ένα κεφαλαίο χαρακτήρα και έναν αριθμό. Συγκεκριμένα για τα αναλογικά χρησιμοποιείται το A (Analog) και ένας αριθμός από το 0-5. Έτσι για παράδειγμα όταν θέλουμε να αναφερθούμε στο Pin A0 και να το δηλώσουμε σαν είσοδο χρησιμοποιούμε την παρακάτω δήλωση:

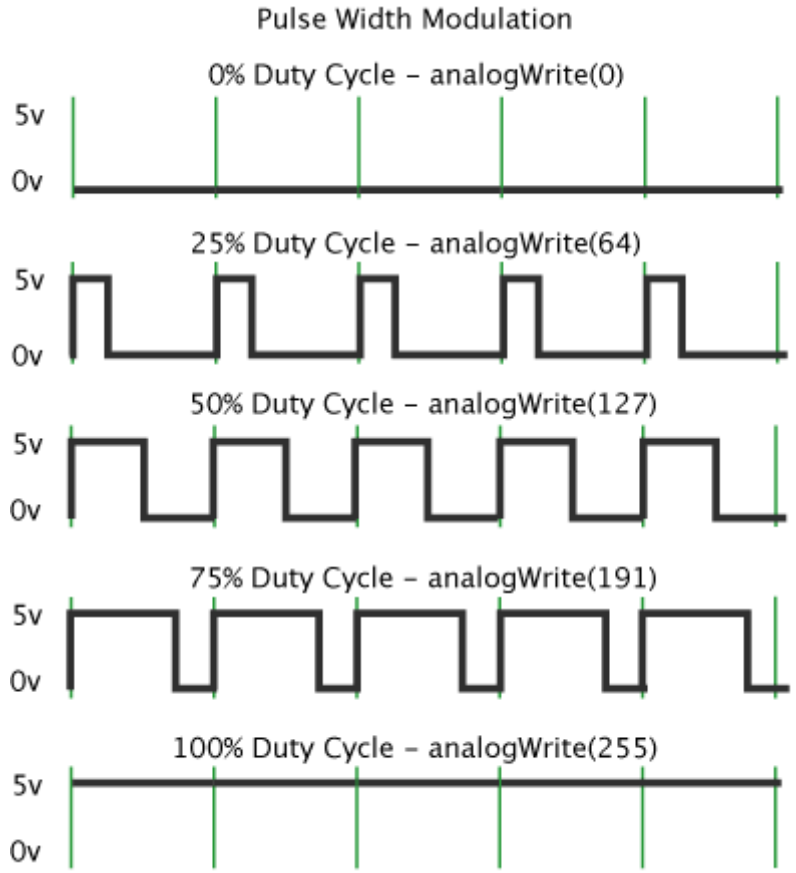
```
pinMode (A0, INPUT);
```

## Παλμοί Διαμόρφωσης Πλάτους (PWM)

Η διαμόρφωση πλάτους είναι μια τεχνική για να παίρνουμε αναλογικά αποτελέσματα από ψηφιακά μέσα. Η ψηφιακή λογική χρησιμοποιείται για την παραγωγή τετραγωνικών παλμών, δηλαδή ένα σήμα που κινείται μεταξύ on και off ή μεταξύ HIGH-LOW ή αλλιώς μεταξύ λογικού 0 και λογικού 1. Αυτή η ακολουθία on-off μπορεί να χρησιμοποιηθεί για να πάρουμε τελικά ένα αποτέλεσμα με τιμές ενδιάμεσες από το λογικό 1 π.χ. 5 V και το λογικό 0, δηλαδή 0 V. Αυτό το επιτυγχάνουμε αλλάζοντας το χρόνο σε ένα παλμό που το σήμα είναι στην υψηλή κατάσταση (HIGH) σε σχέση με το χρόνο που το σήμα είναι στην χαμηλή κατάσταση (LOW). Η τεχνική αυτή χρησιμοποιείται πολύ συχνά όταν θέλουμε να τροφοδοτούμε ένα στοιχείο (πχ ένα LED ) με μια σταθερή τιμή μεταξύ 0 και Vcc.

Στο σχήμα 4 που ακολουθεί, οι πράσινες γραμμές οριοθετούν μια σταθερή τιμή του χρόνου. Η περίοδος του χρόνου είναι η αντίστροφη τιμή της συχνότητας του

παλμού. Δηλαδή αν η συχνότητα PMW του arduino είναι 500Hz, τότε η πράσινη γραμμή έχει περίοδο 2ms. Με την εντολή analogWrite() σε ένα pin, με εύρος τιμής 0-255, έτσι ώστε analogWrite(255) να σημαίνει 100% κύκλο λειτουργίας (duty cycle) και analogWrite(127) να σημαίνει 50% κύκλο λειτουργίας, παράγουμε αναλογικές τιμές μεταξύ 0-5V.




Εικ.4 Ένα παράδειγμα PWM με τη χρήση της analogWrite()

### 2.1.4 Το προγραμματιστικό περιβάλλον του Arduino (IDE)

Το προγραμματιστικό περιβάλλον του Arduino (IDE) προσφέρει έναν εύκολο και γρήγορο τρόπο για σχεδίαση και φόρτωση στην πλακέτα προγραμμάτων που ονομάζονται σχέδια (sketches). Το IDE το ίδιο είναι γραμμένο στη Java και η γλώσσα προγραμματισμού που υποστηρίζει βασίζεται στην **Wiring** η οποία είναι μια διαδεδομένη γλώσσα προγραμματισμού μικροελεγκτών AVR όπως ο Atmega και είναι στην ουσία μια παραλλαγή της C/C++.

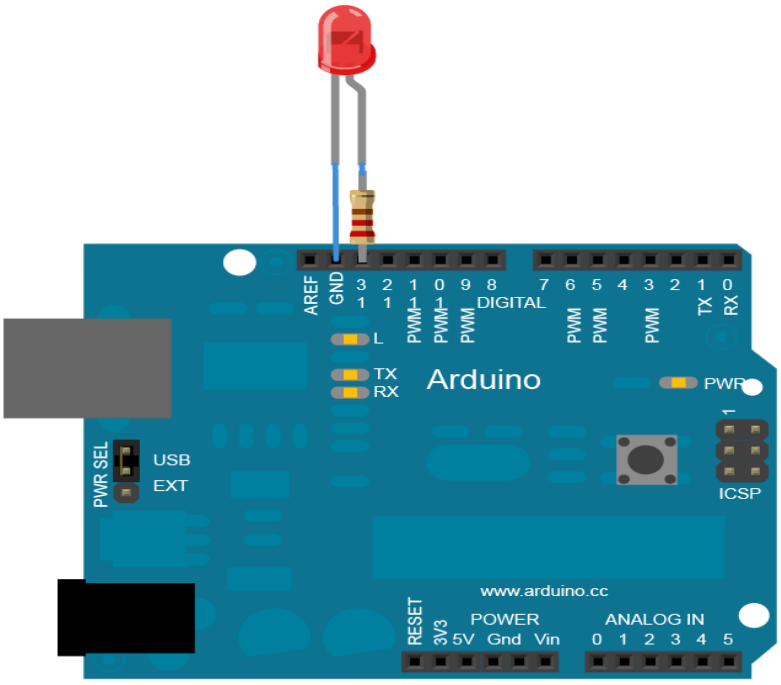
Η βασική δομή ενός sketch φαίνεται παρακάτω, όπου οι εντολές μέσα στην setup() τρέχουν μία μόνο φορά, ενώ οι εντολές μέσα στην loop() τρέχουν συνέχεια.

```
// Ενσωματώσεις βιβλιοθηκών, δηλώσεις μεταβλητών...

void setup ()
{
  // ...
}

void loop ()
{
  // ...
}

// Υπόλοιπες συναρτήσεις...
```

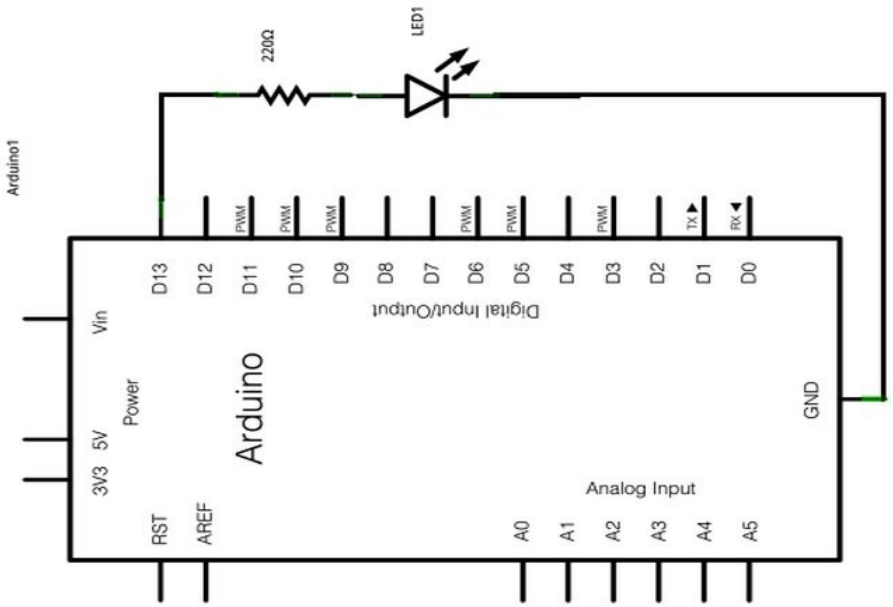
Το αντίστοιχο πρόγραμμα "hello world!" του arduino φαίνεται στο παρακάτω παράδειγμα ακολουθούμενο από το κύκλωμα που αυτό υλοποιεί.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
```

```
int led = 13;

// the setup routine runs once when you press reset:

void setup () {

  // initialize the digital pin as an output.
  PinMode (led, OUTPUT);

}

// the loop routine runs over and over again forever
void loop () {

  digitalWrite (led, HIGH);   // turn the LED on (HIGH is the voltage level)

  delay (1000);     // wait for a second

  digitalWrite (led, LOW);   // turn the LED off by making the voltage LOW

  delay (1000);     // wait for a second
}
```



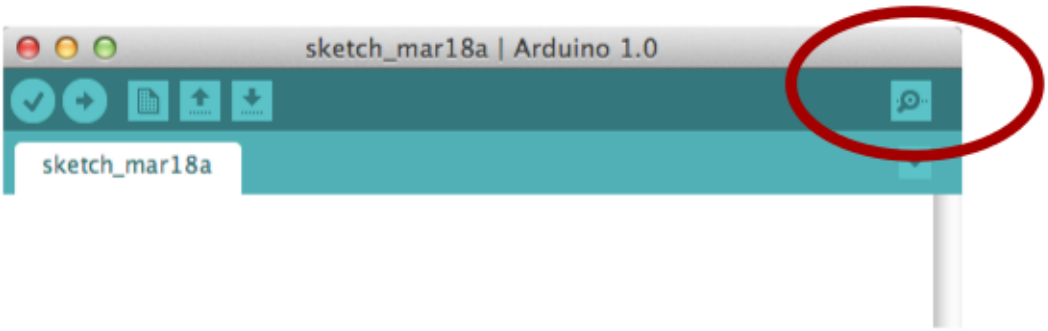

Εικ.5 Η συνδεσμολογία μεταξύ του arduino και μιας αντίστασης 220Ω σε σειρά με ένα led, που αναβοσβήνει κάθε ένα 1 sec (από την ενότητα tutorials στην ιστοσελίδα arduino.cc)




Εικ.6 Το σχηματικό διάγραμμα του παραπάνω κυκλώματος.

## Πρόσβαση στα δεδομένα της σειριακής θύρας μέσω του IDE

Όπως αναφεράμε και προηγουμένως το IDE προσφέρει τη δυνατότητα στο χρήστη μέσω του Serial Monitor να διαβάσει τα δεδομένα εισόδου/εξόδου ακόμα και να στείλει δεδομένα μέσα από αυτό.




Εικ.7 Πατώντας στο σημείο που δείχνει η εικόνα, ανοίγει το Serial monitor (εικόνα από την ιστοσελίδα www.arduino.cc)

Για τον παρακάτω κώδικα βλέπουμε το αποτέλεσμα στο Serial monitor.

```
void setup () {
   Serial.begin (9600);     //establish a connection at 9600 baud rate with serial monitor
}
void loop () {      //run forever
   Serial.println ("Hello!");   //print to the serial monitor the string “Hello”
  delay (1000);  //stop everything for 1sec
}
```


Εικ.8 Τα δεδομένα που αποστέλλονται εμφανίζονται στο serial monitor

## 2.2 Ηλεκτρονικά και μηχανικά μέρη που χρησιμοποιήσαμε

Για την κατασκευή του ρομπότ ώστε να μπορούμε να ολοκληρώσουμε τους στόχους της εργασίας δεν μας αρκεί μόνο ένας μικροελεγκτής όπως το arduino που περιγράψαμε παραπάνω. Χρειάζονται αισθητήρες ώστε να μπορούμε να δεχόμαστε ερεθίσματα από τα δεδομένα που μας παρέχει το περιβάλλον του λαβυρίνθου που φτιάξαμε, όπως επίσης χρειαζόμαστε και ένα τρόπο να κάνουμε το ρομπότ να μπορεί να κινηθεί στο χώρο για αυτό και χρειαζόμαστε και δύο dc-motors που εξυπηρετούν αυτό το σκοπό. Χρειαζόμαστε και ένα motor controller για ακόμα πιο αποτελεσματικό έλεγχο πάνω στα motors αλλά και για λόγους που θα εξηγήσουμε παρακάτω. Τέλος χρειαζόμαστε και μία συσκευή που θα μας επιτρέπει να επικοινωνούμε με ασύρματο τρόπο με τον υπολογιστή μιας και οι βασικοί αλγόριθμοι τρέχουν, όπως αναφέραμε στον υπολογιστή, στο περιβάλλον της Processing.

### 2.2.1 QTR-8 Analog Reflectance Sensor Array

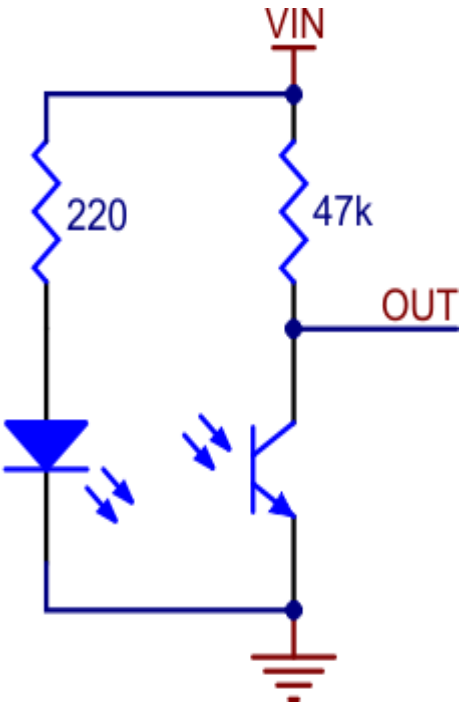
Ο λαβύρινθος τον οποίο διασχίζει το ρομπότ αποτυπώνεται με μαύρες λωρίδες πάνω σε άσπρο φόντο. Είναι σύνηθες να χρησιμοποιούνται αισθητήρες οι οποίοι έχουν σαν σκοπό να διακρίνουν τις διαφορές του άσπρου και του μαύρου χρώματος, όπως και των ενδιάμεσων χρωμάτων, για να είναι τα “μάτια” ενός ρομπότ που ακολουθεί μια γραμμή. Βασική προϋπόθεση είναι η γραμμή ανεξαρτήτως χρώματος, να μπορεί να κάνει αντίθεση σε σχέση με το περιβάλλον της, ώστε να μπορούμε να πάρουμε αξιόπιστα αποτελέσματα.

#### Αρχές Λειτουργίας

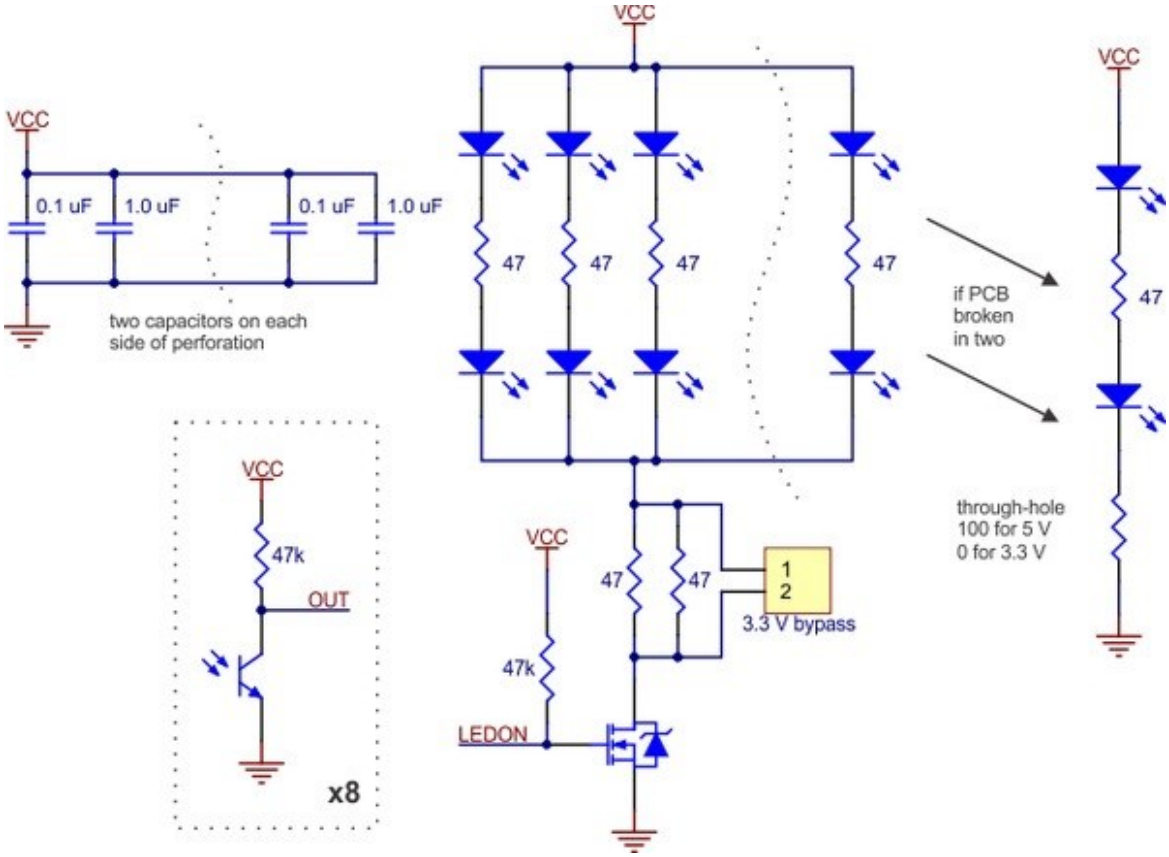
Ο QTR αισθητήρας βασίζεται στην αρχή του **φωτοηλεκτρικού φαινομένου** [5]. Κατά γενικό τρόπο, το φαινόμενο αυτό περιγράφει την απελευθέρωση ηλεκτρικών φορτίων μέσα στο υλικό όταν αυτό βρίσκεται κάτω από την επίδραση φωτεινής

ακτινοβολίας με μήκος κύματος κατώτερο από μία τιμή κατωφλίου, χαρακτηριστική του υλικού. Ο συγκεκριμένος αισθητήρας χρησιμοποιεί μία διάταξη από 8 ζευγάρια υπέρυθρων (IR) πομπών και δεκτών ακτινοβολίας. Οι πομποί ακτινοβολίας είναι φωτοδίοδοι (LED) και εκπέμπουν στα 940 nm. Οι δέκτες ακτινοβολίας είναι φωτοτρανζίστορ τα οποία έχουν την δομή ενός κλασικού *n-p-n* τρανζίστορ. Χαρακτηρίζονται από το γεγονός ότι όταν στη βάση προσπίπτει φως, τότε το φωτο-ρεύμα που παράγεται προστίθεται στο ρεύμα βάσης του τρανζίστορ. Στο σχήμα 9 που ακολουθεί βλέπουμε το σχηματικό διάγραμμα για ένα μόνο ζεύγος LED και φωτοτρανζίστορ.

Φαίνεται ότι η βάση του τρανζίστορ είναι ελεύθερη, οπότε το ρεύμα βάσης εξαρτάται κυρίως από την ακτινοβολία που λαμβάνει. Όσο μεγαλύτερη η ακτινοβολία τόσο μεγαλύτερο είναι και το ρεύμα που περνάει από τη βάση του τρανζίστορ, στον πομπό. Οπότε και η τάση Vout θα είναι μικρότερη. Αντίστροφα τώρα, όταν η ακτινοβολία είναι μικρή, δηλαδή όταν ο αισθητήρας είναι πάνω από μία σκοτεινή επιφάνεια η οποία δεν αντανακλά το φως, το ρεύμα από τη βάση στον πομπό θα είναι και αυτό πολύ μικρό. Αυτό σημαίνει ότι η τάση Vout θα πλησιάζει την τιμή της τάσης Vin, όσο πιο σκούρα θα είναι η επιφάνεια. Αυτή την τεχνική εκμεταλλευόμαστε για να διαβάζουμε αν το ρομπότ είναι πάνω από τη μαύρη γραμμή ή πάνω από την άσπρη επιφάνεια.




Εικ. 9 Διάταξη ενός ζεύγους LED και φωτοτρανζίστορ




Εικ.10 Το κυκλωματικό διάγραμμα του αισθητήρα Qtr από την ιστοσελίδα της κατασκευάστριας εταιρίας.

## 2.2.2 H-Bridge Motor Controller and Dc Motors

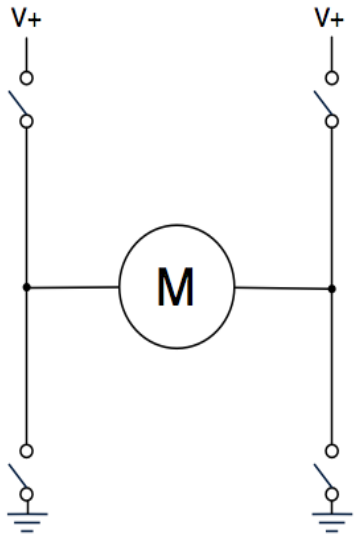
Ο λόγος που χρειαζόμαστε ένα επιπλέον κύκλωμα για να ελέγχουμε τα δύο motors που χρησιμοποιούμε ίσως δεν είναι προφανής από την αρχή και αυτό γιατί ο μικροελεγκτής που χρησιμοποιούμε θεωρητικά μπορεί να μας καλύψει την ανάγκη αυτή. Ένας όμως βασικός κανόνας που αφορά την κατασκευή ηλεκτρονικών κυκλωμάτων είναι να χρησιμοποιούμε στο βαθμό που αυτό είναι δυνατό, διαφορετικές διατάξεις μεταξύ συσκευών, όπως είναι μια πλακέτα μικροελεγκτή και μηχανικών μερών, όπως είναι ένα dc motor. Αυτό γίνεται κυρίως για λόγους ασφαλείας, γιατί η ανάγκη σε ρεύμα για να κινηθεί ένα μηχανικό μέρος είναι αρκετές φορές πολύ μεγαλύτερη από την αντίστοιχη για να λειτουργεί μία μικρή ψηφιακή πλακέτα. Οπότε σε μια περίπτωση υπέρτασης που είναι αρκετά συνηθισμένη προτιμούμε να καεί ένα περιφερειακό κύκλωμα και το υψηλό ρεύμα να μην περάσει και στα υπόλοιπα κυκλώματα, όπως είναι ο μικροελεγκτής και τυχόν αισθητήρες συνδεδεμένους σε αυτόν. Ο δεύτερος λόγος είναι γιατί ένας controller ειδικά σχεδιασμένος για τον έλεγχο των motors, πέρα από την παραπάνω ευελιξία που προσφέρει  είναι και ικανός να παρέχει την απαιτούμενη τάση και ρεύμα που ένα μικροελεγκτής δεν μπορεί.

### Ο τρόπος λειτουργίας δύο τυπικών dc motors

Στην παρούσα εργασία χρησιμοποιούμε δύο dc motors τα οποία έχουν δύο ακροδέκτες, με περιοχή λειτουργίας γύρω στα 6V και καταναλώνουν περίπου 40 mA σε ελεύθερη τροχιά, ενώ σε κατάσταση όπου είναι αδύνατη η κίνηση τους λόγω μεγάλης αντίρροπης δύναμης καταναλώνουν 360 mA. Η κίνηση μεταφέρεται στους τροχούς μέσα από μια διάταξη με γρανάζια σε έναν εξωτερικό άξονα ο οποίος είναι συνδεδεμένος με τον τροχό. Η εναλλαγή της τάσης στους ακροδέκτες αποφέρει και εναλλαγή στην διεύθυνση της κίνησης. Η κίνηση γίνεται προς μία μόνο διεύθυνση και την αντίθετη της. Οπότε, οποιαδήποτε στροφή επιθυμούμε, θα πρέπει να γίνει ρυθμίζοντας την τάση, άρα και την ταχύτητα, σε κάθε motor διαφορετικά, οπότε και προκύπτει διαφορά στην ταχύτητα περιστροφής της κάθε ρόδας. Περιγράφοντας λοιπόν ποιες είναι οι ανάγκες για τον έλεγχο των motors, της ταχύτητας και της διεύθυνσης της κίνησης εισάγουμε την έννοια του H bridge κυκλώματος.

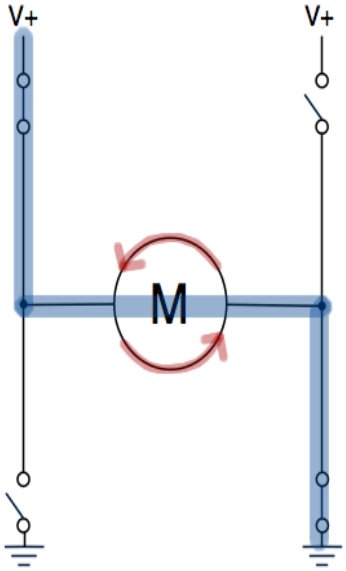
Το κύκλωμα H bridge πήρε το όνομα του από το κεφαλαίο Η, που προσομοιάζει το κύκλωμα αν το δούμε αφαιρετικά σαν τέσσερις διακόπτες που στη μέση βρίσκεται ένα motor, όπως ίσως φαίνεται και στο διπλανό σχήμα.

Κάθε ακροδέκτης μπορεί να βρίσκεται σε τρεις διαφορετικές καταστάσεις. Η πρώτη είναι να είναι συνδεδεμένος με θετική τάση, η δεύτερη να είναι συνδεδεμένος με τη γείωση και η τρίτη να είναι ελεύθερος.



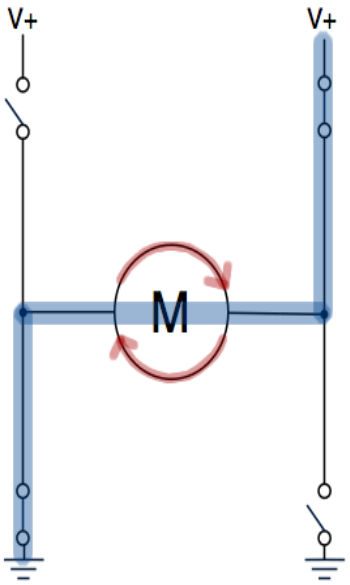

Εικ.11 Η Bridge κύκλωμα με ανοικτούς όλους τους διακόπτες

Στο σχήμα ο πάνω αριστερά και ο κάτω δεξιά διακόπτης είναι κλειστοί, οπότε δημιουργείται κύκλωμα με τη φορά του ρεύματος να είναι από το V+ στη γείωση. Αυτό έχει σαν αποτέλεσμα την αριστερόστροφη περιστροφή του τροχού.




Εικ.12 Η Bridge κύκλωμα με δύο ανοικτούς διακόπτες για περιστροφή

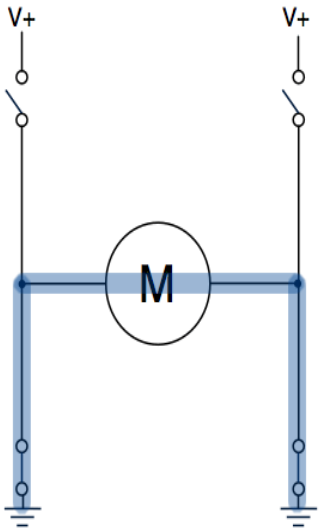
Στο διπλανό σχήμα φαίνεται η δεύτερη βασική περίπτωση όπου οι διακόπτες που ήταν πριν ανοικτοί, τώρα είναι κλειστοί, με αποτέλεσμα στο κύκλωμα που δημιουργείται να έχει αντίθετη φορά το ρεύμα και ο τροχός να κινείται δεξιόστροφα.




Εικ.13 H Bridge κύκλωμα με δύο ανοικτούς διακόπτες για περιστροφή

Σε αυτή την περίπτωση δείχνουμε και την κατάσταση όπου μόνο οι κάτω διακόπτες είναι κλειστοί. Τώρα δεν υπάρχει καμία ροή ρεύματος και αυτό το κύκλωμα οδηγεί σε ένα ομαλό σταμάτημα του τροχού. Υπάρχει και η περίπτωση όπου μόνο οι δύο πάνω διακόπτες είναι κλειστοί οπότε και εφαρμόζεται ισοδύναμη τάση στους ακροδέκτες του motor. Επίσης και εδώ δεν υπάρχει ροή ρεύματος, αλλά αυτό το κύκλωμα οδηγεί σε απότομο σταμάτημα και προτιμάται να αποφεύγεται.

Η περίπτωση όπου ο πάνω και ο κάτω διακόπτης της ίδιας πλευράς, είναι κλειστοί, από τους σύγχρονους controllers δεν δύναται σαν επιλογή αν και θεωρητικά υπάρχει.

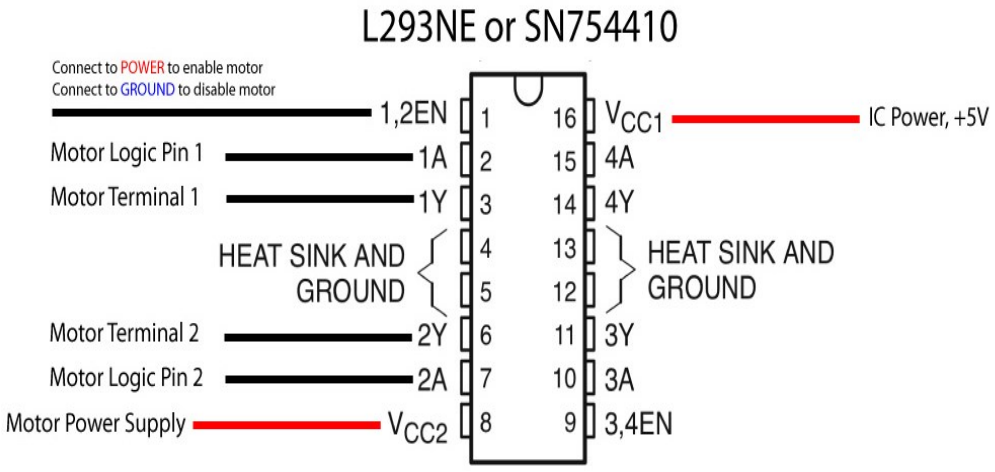



Εικ.14 H Bridge κύκλωμα με 2 ανοικτούς διακόπτες για σταμάτημα

Στην πραγματικότητα όμως ένα κύκλωμα H bridge δεν είναι τόσο απλό όσο παρουσιάσαμε αλλά οι βασικές αρχές του είναι αυτές. Οι διακόπτες υλοποιούνται με τη χρήση τρανζίστορ (διπολικά ή MOSFET). Για να έχουμε μια καλύτερη εικόνα που βοηθάει στη χρήση του SN754410 φτιάχνουμε και ένα πίνακα καταστάσεων για εισόδους και εξόδους για ένα motor.

| Enable | 1A | 2A | ΠΕΡΙΣΤΡΟΦΗ |
|---|---|---|---|
| HIGH | LOW | HIGH | Δεξιόστροφη |
| HIGH | HIGH | LOW | Αριστερόστροφη |
| HIGH | LOW | LOW | Σταδιακό φρένο |
| HIGH | HIGH | HIGH | Απότομο φρένο |
| LOW | X | X | Ελεύθερη |

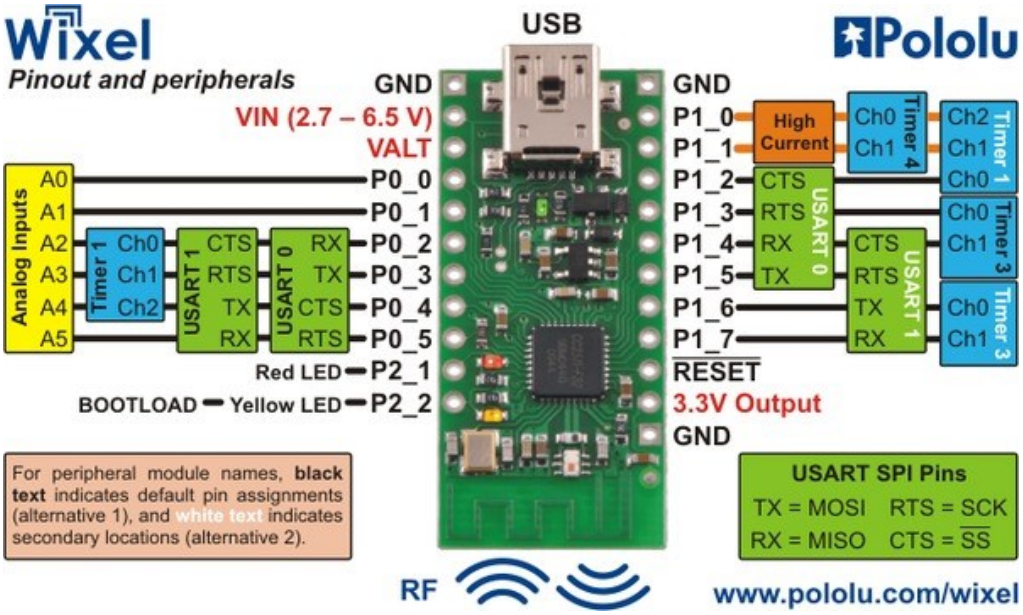
Εικ. 15 πίνακας αληθείας για το H Bridge




Εικ. 16 Διάγραμμα συνδέσεων για τον motor controller

## 2.2.3 Wixel and Wixel Shield

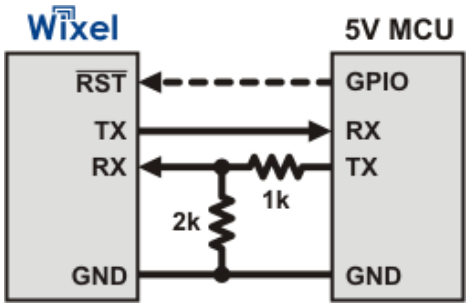
Η επόμενη συσκευή που εξετάζουμε, εξυπηρετεί τον ασύρματο προγραμματισμό του ρομπότ όπως και την επικοινωνία του με τον υπολογιστή. Όπως αναφέραμε και προηγουμένως είναι πολύ βασική προϋπόθεση να υπάρχει ασύρματη επικοινωνία, γιατί ανάλογα με τα δεδομένα που στέλνονται στο πρόγραμμα στον υπολογιστή, αποφασίζεται και η επόμενη κίνηση του ρομπότ. Βέβαια η ασύρματη λειτουργία εξυπηρετεί και πρακτικούς σκοπούς όπως είναι η ελεύθερη κίνηση στο χώρο κάτι που θα ήταν πολύ δύσκολο με τη χρήση καλωδίου. Υπάρχουν αρκετοί τρόποι ασύρματης επικοινωνίας με το arduino αλλά εμείς επιλέξαμε την επικοινωνία με μια συσκευή ραδιοσυχνοτήτων (RF device) και αυτό γιατί η συγκεκριμένη παρέχει ένα εύκολο τρόπο για να φτιάξει κανείς μια ασύρματη επικοινωνία, όπως επίσης και ένα περιβάλλον για τυχόν ανάπτυξη κώδικα για πιο προχωρημένες εφαρμογές. Βασικό επίσης είναι ότι στην ουσία, μία Wixel συσκευή είναι και αυτή ένας μικροελεγκτής, πιο αδύναμος από το arduino, αλλά μπορεί να χρησιμοποιηθεί σαν συμπληρωματικό στοιχείο. Τέλος έχει μια πολύ καλή σχέση απόδοσης-τιμής, κάτι που το χρειαζόμασταν για να κρατήσουμε το κόστος σε λογικά επίπεδα.



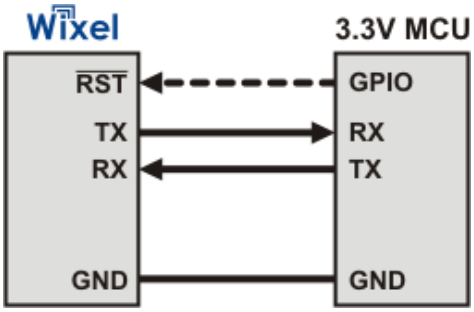

Εικ. 17 Περιγραφή των ακροδεκτών ενός wixel

Για την πραγματοποίηση μιας σύνδεσης ισχύει και σε αυτή την περίπτωση ότι είναι απαραίτητος ένας πομπός και ένας δέκτης. Τα Wixel για αυτό το λόγο δουλεύουν σε ζευγάρια, αν και με λίγες μετατροπές στον κώδικα τους, υπάρχει η δυνατότητα σε ομάδες Wixel να επικοινωνούν μεταξύ τους. Οι συσκευές αυτές εκπέμπουν ραδιοκύματα σε συχνότητα 2.4 MHz και έχουν εύρος επικοινωνίας μέχρι περίπου 15 μέτρα. Είναι βασισμένες στον CC2511F32 μικροελεγκτή ο οποίος έχει ενσωματωμένο πομποδέκτη, 32 Kbyte flash memory, 4 Kbyte RAM και USB διασύνδεση για την χρήση με τον υπολογιστή όπως επίσης και 15 εισόδους/εξόδους γενικού σκοπού. Στην ουσία είναι ασύρματοι USB to TTL σειριακοί μετατροπείς και αυτό σημαίνει ότι καταφέρνουν και μεταφράζουν την πληροφορία από το εσωτερικό των τρανζίστορ σε πληροφορία ικανή να διαβαστεί από μια σειριακή θύρα του υπολογιστή κάτι το οποίο είναι πολύ χρήσιμο στις ρομποτικές κατασκευές. Ένα Wixel έχει τρεις διαφορετικές καταστάσεις λειτουργίας που μπορεί να βρίσκεται. Η πρώτη είναι να είναι συνδεδεμένο μέσω USB σε έναν υπολογιστή. Τότε βρίσκεται στην λειτουργία *USB to Radio*. Η δεύτερη είναι να είναι συνδεδεμένο με έναν μικροελεγκτή οπότε η τάση τροφοδοσίας του έρχεται από τη Vin είσοδο που διαθέτει. Σε αυτή την περίπτωση βρίσκεται στη λειτουργία *UART to Radio*. Η τρίτη λειτουργία είναι στην περίπτωση που είναι συνδεδεμένο σε έναν μικροελεγκτή και η τροφοδοσία του παρέχεται και από την είσοδο Vin και από την USB θύρα. Στην τελική αυτή περίπτωση βρίσκεται στη λειτουργία *USB-to-UART*. Εμείς ενδιαφερόμαστε μόνο για τις δύο πρώτες περιπτώσεις.

Για να επιτευχθεί μια επικοινωνία μεταξύ των δύο πλευρών πολύ βασικό στοιχείο είναι η συμφωνία στο ρυθμό μετάδοσης των δεδομένων (baud rate). Δηλαδή στο πόσα bits μεταδίδονται ανά δευτερόλεπτο. Ένα Wixel χρησιμοποιεί ρυθμό μετάδοσης bit ίσο με 350Kbps και είναι ικανό να λαμβάνει ή να μεταδίδει δεδομένα ύψους 10KB/s. Η επικοινωνία μέσω Wixel επιτρέπει τη χρήση baud rate μεταξύ των ακέραιων τιμών 110 to 115200 bits per second. Τα εισερχόμενα δεδομένα λαμβάνονται στην Rx γραμμή του Wixel και τα απεσταλμένα μεταδίδονται από την Tx γραμμή. Οπότε σε μια σύνδεση ενός Wixel με το arduino η συνδεσμολογία πρέπει να ακολουθεί τον τρόπο που περιγράψαμε προηγουμένως. Για να μπορεί το Wixel να θέσει το arduino σε *bootloader mode*, δηλαδή να μεταφέρει δεδομένα για να το επαναπρογραμματίσει, πρέπει επιπλέον να συνδεθεί η γραμμή Reset του arduino με τη γραμμή Reset του Wixel. Οι είσοδοι ενός Wixel λειτουργούν στα 3.3V οπότε μια σύνδεση με έξοδο από 5V χρειάζεται να μετατραπεί και αυτή στα 3.3V. Αυτό επιτυγχάνεται συνήθως με έναν απλό διαιρέτη τάσης.



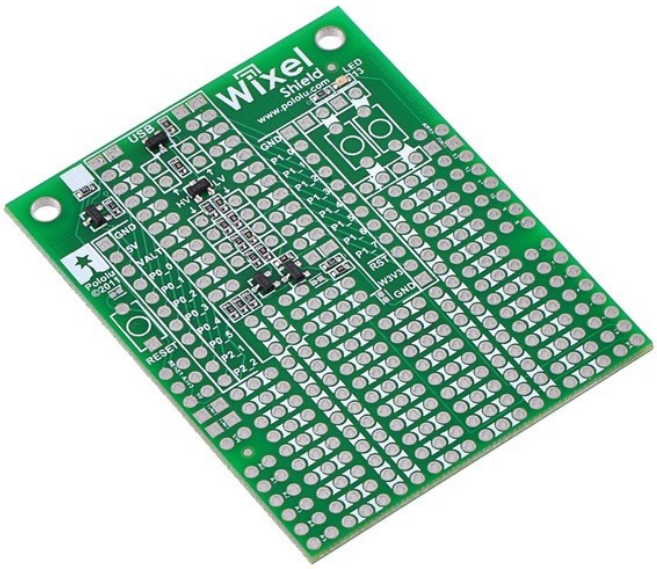

Εικ. 18 η συνδεσμολογία με έναν μικροελεγκτή 5 V




Εικ. 19 η συνδεσμολογία με έναν μικροελεγκτή 3.3 V

## Wixel shield

Το Wixel shield δεν είναι τίποτα παραπάνω από μία τυπωμένη πλακέτα, η οποία είναι ειδικά σχεδιασμένη για την καλύτερη διασύνδεση μεταξύ ενός Wixel και μιας πλακέτας arduino επίσημης έκδοσης. Επεκτείνει τη λειτουργικότητα ενός Wixel παρέχοντας μακρύτερες διασυνδέσεις για τις εισόδους/εξόδους του και έχει και έναν χώρο κατάλληλο για ανάπτυξη καινούργιων ηλεκτρονικών κυκλωμάτων (prototyping space). Αυτός είναι και ο κύριος λόγος που το χρησιμοποιούμε και στην παρούσα εργασία.



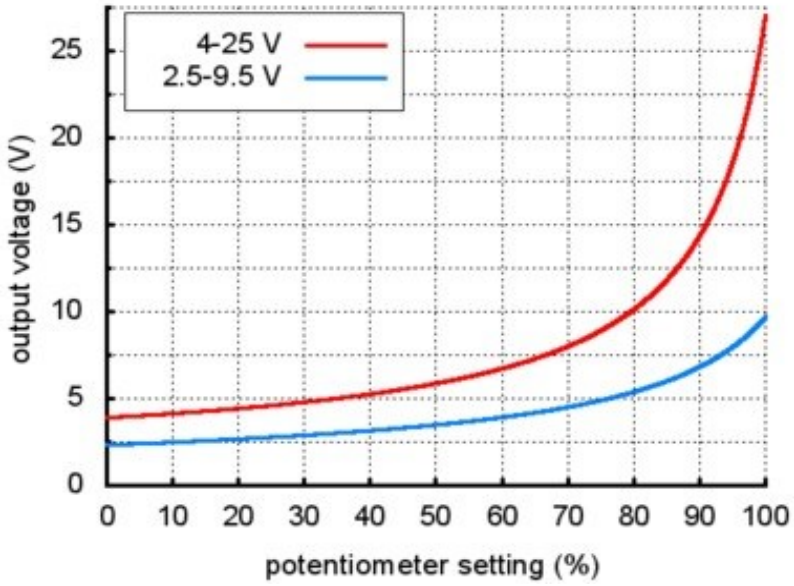

Εικ. 20 Το wixel shield

## 2.2.4 Adjustable Boost Regulator 2.5 – 9 V

Το τελευταίο εξάρτημα που εξετάζουμε εξυπηρετεί κυρίως τις ενεργειακές ανάγκες που προκύπτουν για την αποτελεσματική λειτουργία όλων των παραπάνω από μία κοινή πηγή τάσης. Για την τροφοδοσία του ρομπότ χρησιμοποιούμε 4 AAA επαναφορτιζόμενες μπαταρίες με 1.2V ονομαστική τιμή για την κάθε μία, που είναι ικανές να παρέχουν συνολικά 4000mA ανά ώρα. Παρέχουν αρκετή ισχύ για τη σωστή λειτουργία όλων των εξαρτημάτων αλλά οι ενεργειακές ανάγκες που χρειάζονται τα δύο motors, τα οποία είναι και τα υψηλότερα σε απαιτήσεις ισχύος, κάνουν απαραίτητη τη χρήση μιας ηλεκτρονικής διάταξης ικανής να ανταποκριθεί στις ενεργειακές ανάγκες που έχουμε. Αυτό πρακτικά σημαίνει ότι με το πέρασμα του χρόνου όσο και αν πέφτει η αποδοτικότητα των μπαταριών μας, η ισχύς που παίρνουμε στην έξοδο είναι σταθερή, οπότε μέχρι η τάση να πέσει κάτω από το σημείο λειτουργίας του DC μετατροπέα, δεν παρατηρούμε καμία αλλαγή στην κίνηση και τη γενικότερη συμπεριφορά του ρομπότ, κάτι που προηγουμένως το βλέπαμε αρκετά συχνά και αποτελούσε μία μεγάλη δυσκολία.

### Συνδεσμολογία και τρόπος λειτουργίας του boost regulator

Boost regulator ή αλλιώς μετατροπέας DC σε DC είναι γενικά μία ηλεκτρονική διάταξη η οποία παρέχει τάση εξόδου (Vout) μεγαλύτερη από την τάση εισόδου (Vin). Επειδή η σχέση ισχύος $P=VI$ πρέπει να ισχύει σε όλες τις περιπτώσεις, σημαίνει ότι στην έξοδο του μετατροπέα θα έχουμε μικρότερη τιμή ρεύματος ($I_{out}$) από ότι στην είσοδο ($I_{in}$). Δηλαδή έχουμε κέρδος σε τάση αλλά χάνουμε στη διάρκεια που είναι ζωντανές οι μπαταρίες αφού θα τραβάνε τώρα πολύ περισσότερο ρεύμα. Ο συγκεκριμένος μετατροπέας δίνει μεγάλη ευχέρεια στη χρήση του και υλοποιεί ένα κύκλωμα για το οποίο όμως, η κατασκευάστρια εταιρεία δεν δίνει τα σχηματικά διαγράμματα. Παρ' όλα αυτά μια τυπική απλοϊκή διάταξη DC σε DC μετατροπέα φαίνεται παρακάτω.

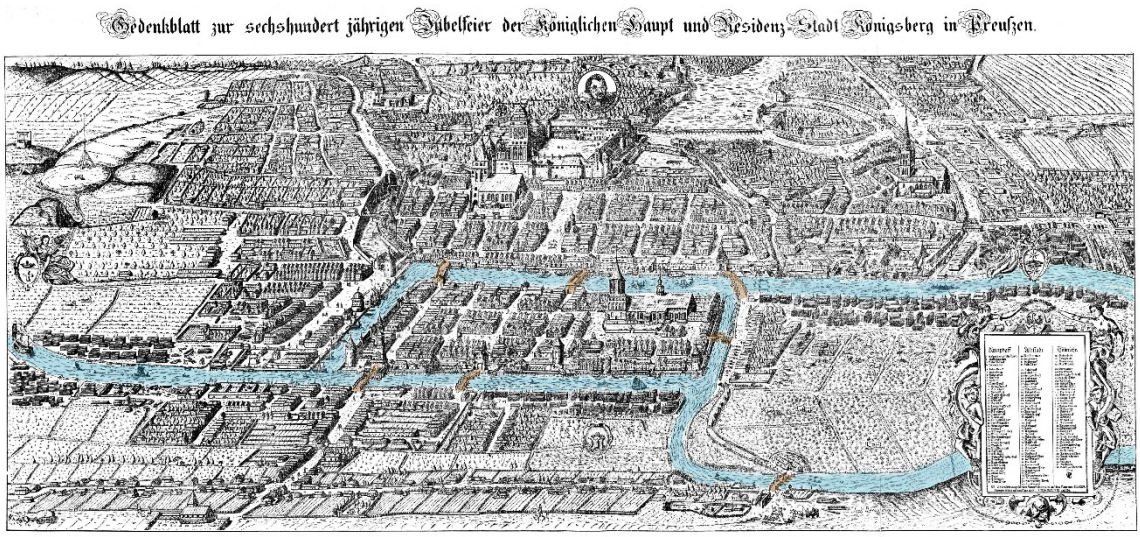



Εικ. 21 Μια απλή διάταξη μετατροπέα DC με χρήση πηνίου

Ο μετατροπέας που χρησιμοποιούμε είναι ικανός να μετατρέπει μια τάση χαμηλή όσο 1.5 V σε 9 V. Η τάση εξόδου καθορίζεται από ένα μικρό ποτενσιόμετρο που βρίσκεται πάνω στην πλακέτα και βασική προϋπόθεση για την ασφαλή λειτουργία του είναι η τάση εισόδου να μην ξεπερνά την τάση εξόδου. Έχει τρεις ακροδέκτες, μία για την τάση Vin, μία για την τάση Vout και μία τρίτη για τη γείωση. Στα παρακάτω σχήματα φαίνονται και οι χαρακτηριστικές καμπύλες της τάσης και του ρεύματος εξόδου για διαφορετικές τιμές της Vin. Η αποδοτικότητα του αγγίζει το 80%-90% όταν η έξοδος είναι διπλάσια της εισόδου κα το ρεύμα εξόδου μεταξύ 100-500 mA.

## 2.3 Μαθηματική αναπαράσταση γράφου

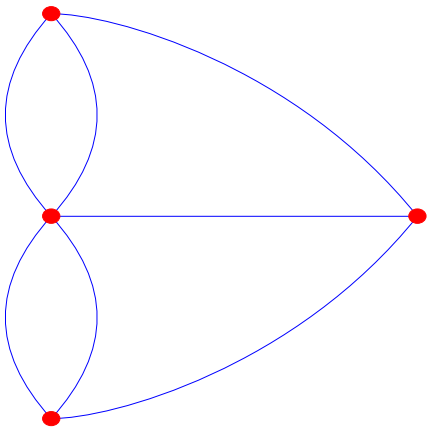
Το 1736 ο Leonard Euler προσπάθησε να λύσει ένα πρόβλημα που ταλαιπωρούσε τους κατοίκους μιας πόλης της σημερινής δυτικής Ρωσίας. Η πόλη του Konigsberg διασχιζόταν από ένα ποτάμι που την χώριζε στη μέση, ενώ στο ποτάμι υπήρχαν και δύο κατοικημένα νησιά. Για αυτό το λόγο υπήρχαν εφτά γέφυρες σε διαφορετικά σημεία για την επικοινωνία μεταξύ των δύο πλευρών και των δύο νησιών. Για χρόνια οι κάτοικοι προσπαθούσαν να βρουν, αν μπορεί να υπάρξει μία ενιαία διαδρομή που θα περνάει από όλα τα σημεία και στην οποία κάθε γέφυρα θα διασχίζεται το πολύ μία μόνο φορά.




Εικ. 22 Πίνακας που απεικονίζει την πόλη του Konigsberg και τις 7 γέφυρες

Η λύση ξεκίνησε αναπαριστώντας τα τέσσερα διαφορετικά μέρη της πόλης, με τέσσερα σημεία ή αλλιώς **κόμβους** και τις εφτά γέφυρες με γραμμές ή αλλιώς **ακμές**. Αυτή η αφαιρετική αναπαράσταση ονομάζεται πλέον γράφος [6]. Ο αριθμός των ακμών που τερματίζουν σε κάθε κόμβο ονομάζεται βαθμός του κόμβου. Το πρόβλημα τώρα συνίσταται στην εύρεση ενός συνεχόμενου μονοπατιού μεταξύ δύο κόμβων (αρχικού και τελικού), οι οποίοι μπορεί να ταυτίζονται, αλλά θα κάνουν χρήση κάθε ακμής μία και μόνο φορά.

Ο Euler παρατήρησε ότι για να μπορεί να υπάρχει μια τέτοια διαδρομή, θα πρέπει κάθε μη τερματικός κόμβος να είναι άρτιου βαθμού, καθώς μόνο δύο διαφορετικές ακμές μπορούν να χρησιμοποιούνται κάθε φορά που ένας μη τερματικός κόμβος προσπελαύνεται. Οπότε καμία τέτοια διαδρομή δεν μπορεί να υπάρξει καθώς όλοι οι κόμβοι είναι περιττής βαθμού. Και έτσι, με τη συμβολή του Euler, η θεωρία των γράφων γεννιέται.

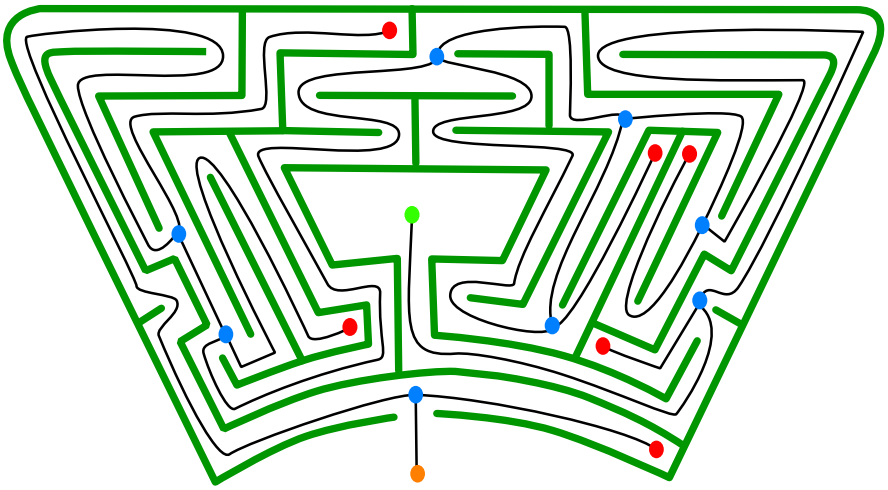


Εικ. 23 Ο γράφος του προβλήματος με τις 7 γέφυρες

## 2.3.1 Αναπαράσταση ενός λαβυρίνθου σε γράφο

Ορίζουμε σαν λαβύρινθο μια δομή που αποτελείται από μονοπάτια, κάθε ένα από τα οποία μπορεί να διασταυρώνεται με ένα ή και παραπάνω άλλα μονοπάτια. Τα μονοπάτια μπορούν να σχηματίζουν διασταυρώσεις, να οδηγούν σε αδιέξοδα, δηλαδή

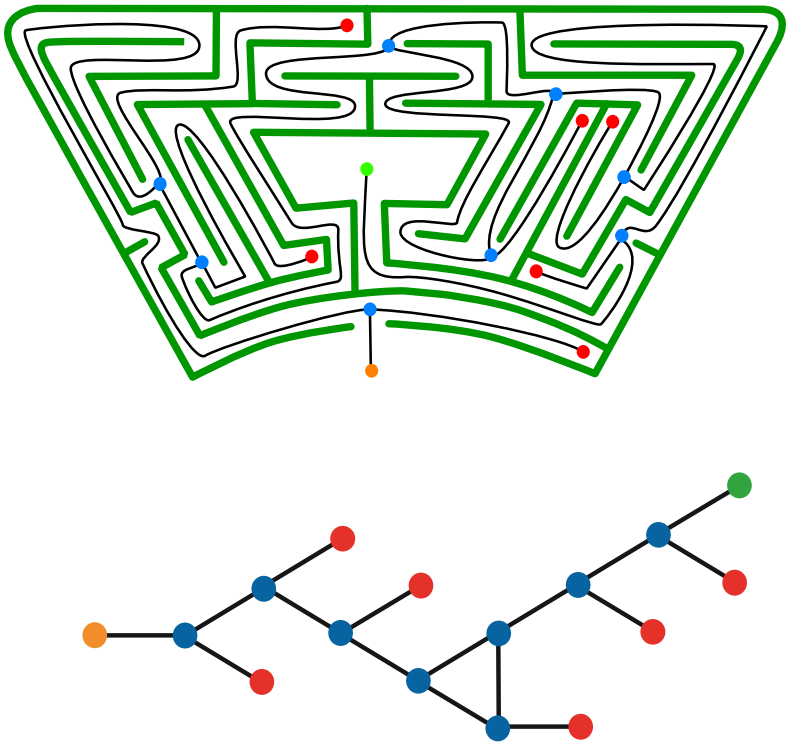
σε σημεία που δεν συναντάται καμία άλλη διαδρομή ή και να σχηματίζουν και κύκλους δηλαδή διαδρομές που καταλήγουν στο ίδιο σημείο. Κάθε λαβύρινθος έχει τουλάχιστον ένα αρχικό σημείο εισόδου ή έναρξης και τουλάχιστον ένα σημείο εξόδου ή τερματισμού. Τα συνηθέστερα προβλήματα πάνω στον λαβύρινθο είναι η διάσχιση με αναζήτηση του τερματικού σημείου ή η διάσχιση με κάποιο γνωστό τερματικό σημείο. Κάθε λαβύρινθος ορίζουμε ότι μπορεί να έχει τέσσερις διαφορετικές καταστάσεις σημείων κάθε μία από τις οποίες αναπαρίσταται με ένα διαφορετικό χρώμα όπως φαίνεται και στο σχήμα που ακολουθεί.



Εικ. 24 Λαβύρινθος στη διαδικασία αναπαράστασης σε γράφο

- Αρχικό σημείο ή σημείο εισόδου
- Διασταύρωση όπου τρία ή περισσότερα μονοπάτια συναντιόνται
- Αδιέξοδο όπου δεν υπάρχει κάποια καινούργια διαδρομή
- Στόχος ή αλλιώς σημείο τερματισμού

Για την αναπαράσταση σε γράφο, κάθε σημείο που ορίσαμε είναι ένας κόμβος του γράφου και κάθε μονοπάτι μία ακμή του, η οποία συνδέει δύο κόμβους.



Εικ. 25 Ο λαβύρινθος και ο αντίστοιχος γράφος που τον αναπαριστά

Ορίζουμε σαν γράφο [7] μια δομή που αποτελείται από ένα σύνολο V το οποίο έχει σαν στοιχεία κόμβους και ένα σύνολο E το οποία έχει σαν στοιχεία ακμές οι οποίες είναι υποσύνολα δύο στοιχείων V (δηλαδή, μια ακμή σχετίζεται με δύο κορυφές και η σχέση απεικονίζεται ως μη ταξινομημένο ζεύγος των κορυφών σε σχέση με τη συγκεκριμένη ακμή). Για να αποφευχθούν οι αμφισημίες, αυτός ο τύπος γραφήματος μπορεί να περιγραφεί με ακρίβεια ως μη-κατευθυνόμενο και απλό. Για το παραπάνω γράφο θα έχουμε.

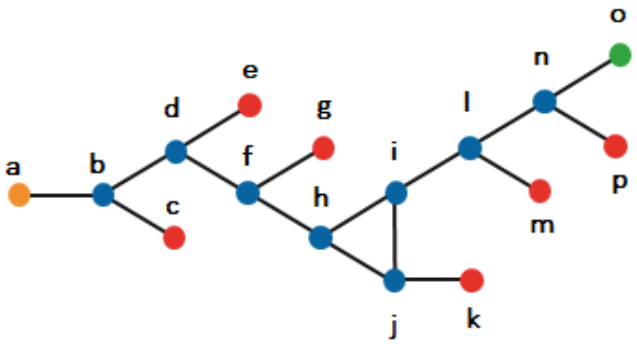
V= {a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p}

και

E={(a, b), (b, d), (b, c), (d, e), (d, f), (f, g), (f, h ), (h, i), (h, j), (i, j),

(j, k), (i, l), (l, m), (l, n), (n, o), (o, p)}

Για τον γράφο που φαίνεται στην εικόνα 26 ορίσαμε τα σύνολα V και E.




Εικ. 26 Ο προηγούμενος γράφος με γράμματα που αντιστοιχούν στους κόμβος του

## 2.3.2 Αλγόριθμοι διάσχισης γράφου

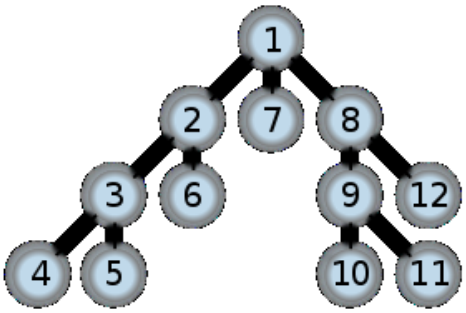
Στην θεωρία των γράφων έχουν αναπτυχθεί πολλοί αλγόριθμοι διάσχισης και αυτό γιατί τις πιο πολλές φορές ένας γράφος δεν είναι τόσο μικρός όσος στο παράδειγμα που παρουσιάσαμε. Συνήθως ένας γράφος ειδικά εκεί όπου υπάρχει και πρακτική εφαρμογή, όπως για παράδειγμα στις τηλεπικοινωνίες, αποτελείται από χιλιάδες κόμβους οπότε η αποδοτικότητα της μεθόδου που θα διασχιστεί ένας γράφος είναι πολύ σημαντική. Στην παρούσα εργασία θα περιοριστούμε μόνο στην παρουσίαση των πιο γνωστών ίσως αλγορίθμων, πάνω στους οποίους έχουν βασιστεί και οι περισσότεροι πιο πολύπλοκοι, όπως αναφέραμε.

### Αναζήτηση Κατά Βάθος (Depth First Search-DFS)

Ο αλγόριθμος διάσχισης κατά βάθος (DFS) είναι ίσως ο πιο γνωστός και αποτελεί την βάση για πολλούς άλλους. Ο DFS σαν βασική αρχή έχει ότι ξεκινώντας από έναν αρχικό κόμβο (root), τον οποίο υποθέτουμε στην κορυφή του γραφήματος, επισκέπτεται πρώτα έναν ανεξερεύνητο γειτονικό κόμβο ο οποίος βρίσκεται χαμηλότερα από τον προηγούμενο του. Αυτή η επανάληψη συνεχίζει μέχρι να φτάσει σε ένα κόμβο ο οποίος δεν έχει άλλον γειτονικό σε χαμηλότερο βάθος. Έπειτα επιστρέφει στον αμέσως προηγούμενο, υψηλότερο κόμβο από εκεί που σταμάτησε, για να επεκτείνει την αναζήτηση σε βάθος, των γειτονικών κόμβων που δεν είχε

εξερευνήσει. Όταν όλοι οι κόμβοι σημειωθούν ως προσπελασμένοι, ο αλγόριθμος σταματάει.

Ένα παράδειγμα διάσχισης ενός γράφου αν ο DFS επιλέγει πρώτα έναν αριστερό κόμβο φαίνεται στο παράδειγμα της παρακάτω εικόνας.




Εικ. 27 Ο τρόπος προσπέλασης ενός γράφου με τη χρήση του DFS

Στην πιο απλή του μορφή ο DFS, μπορεί να υποφέρει από συνθήκη μη τερματισμού, αν για παράδειγμα το μονοπάτι που ακολούθησε είναι άπειρο ή αν δεν θυμάται τους προηγούμενους κόμβους που επισκέφθηκε όπως φαίνεται στο παράδειγμα που ακολουθεί.

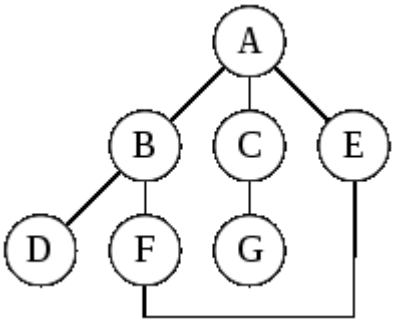
Στην εικόνα που ακολουθεί φαίνεται ένας μικρός γράφος με αρχικό κόμβο τον Α που βρίσκεται στην κορυφή. Αν ο DFS κρατάει σε μια δομή τους κόμβους που έχει εξερευνήσει τότε η ακολουθία προσπέλασης είναι η εξής. Η σειρά προτίμησης είναι και εδώ από αριστερά προς τα δεξιά.

Οπότε θα έχουμε: *A-> B-> D-> F-> E-> C-> G*

Αν όμως δεν θυμάται την ακολουθία την οποία ακολούθησε, τότε η σειρά με την οποία θα διασχίζει τους κόμβους είναι η εξής:

*A-> B-> D-> F-> E-> A-> B-> D-> F-> E->…*

Μπαίνει δηλαδή σε έναν ατέρμονο κύκλο, μη διασχίζοντας ποτέ τους κόμβους C και G.




Εικ 28 Γράφος με κύκλο

Για να αποφύγει ο DFS τέτοιες περιπτώσεις χρησιμοποιούνται αλγόριθμοι με μικρές αλλαγές όπως η *επαναληπτική εμβάθυνση(iterative deepening)*. Δεν είναι όμως στους στόχους του κειμένου να προχωρήσει παραπέρα σε ανάλυση για τις διάφορες

παραλλαγές του DFS, αλλά μόνο να εισάγουμε τις βασικές αρχές του αλγορίθμου και την ανάγκη πολλές φορές να τον χρησιμοποιούμε με αυτές.

Αντίστοιχα γνωστός αλγόριθμος είναι η Αναζήτηση Κατά Πλάτος (Breadth First Search-BFS), για τον οποίο μόνο θα αναφέρουμε ότι, όπως προϊδεάζει και το όνομα του δίνει προτεραιότητα στη διάσχιση των κόμβων που είναι στο ίδιο επίπεδο με τον τρέχοντα κόμβο πριν ξεκινήσει την προσπέλαση κόμβων σε πιο χαμηλά επίπεδα.

## Αλγόριθμος του Dijkstra

Ο αλγόριθμος του Dijkstra, πήρε το όνομα του από τον Ολλανδό μαθηματικό Edsger Dijkstra και δημοσιεύθηκε το 1959. Ο αλγόριθμος αυτός επιλύει το πρόβλημα εύρεσης των συντομότερων μονοπατιών (*single-source shortest path problem*) για ένα γράφο (κατευθυνόμενο ή μη), με μη αρνητικά βάρη στις ακμές του. Ο Dijkstra ανήκει στην κατηγορία των άπληστων αλγορίθμων (*greedy algorithms*), επειδή κάθε φορά επιλέγει την τοπικά βέλτιστη λύση, ώστε στο τελευταίο βήμα να συνθέτει την συνολικά βέλτιστη λύση. Ο αλγόριθμος αυτός έχει σαν έξοδο ένα δέντρο ελάχιστου μονοπατιού *(minimum spanning tree*). Στην βασική του μορφή τα αποτελέσματα που παίρνουμε από τον Dijkstra είναι τα ελάχιστα βάρη για το σύνολο των ακμών του γράφου που συνθέτουν τα ελάχιστα μονοπάτια για κάθε υποδέντρο του γράφου. Οπότε με κάποιες μικρές προσθήκες μπορούμε να συνθέσουμε το μονοπάτι που μας ενδιαφέρει.

### Περιγραφή λειτουργίας του Dijkstra σε βήματα

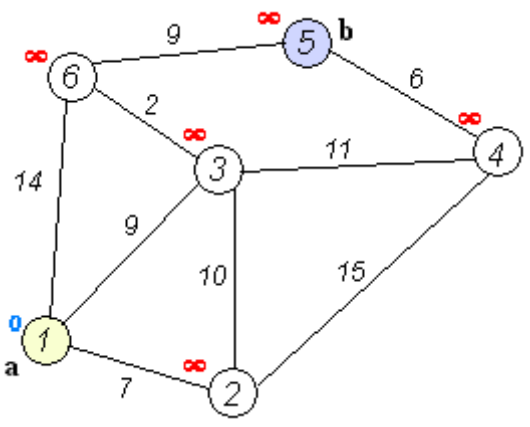
Ορίζουμε έναν γράφο G ( V, E ), όπου V το σύνολο των κόμβων του και E το σύνολο των ακμών του. Επίσης, έχουμε μια συνάρτηση βάρους w : E → $R^+$ υ {0} ορισμένη στις ακμές του γράφου. Αυτό σημαίνει ότι για να πάμε από έναν κόμβο του γράφου σε έναν άλλο, θα πάμε με κάποιο κόστος. Είναι σημαντικό, όπως θα φανεί παρακάτω, τα βάρη να μην είναι αρνητικά, επειδή διαφορετικά ο αλγόριθμος δεν δίνει σωστό αποτέλεσμα. Ο αλγόριθμος του Dijkstra βρίσκει τα μονοπάτια που πρέπει να

ακολουθήσουμε από έναν κόμβο-αφετηρία προς όλους τους υπόλοιπους, με το ελάχιστο κόστος.

Χρησιμοποιούμε ένα διάνυσμα *dist[]* μεγέθους |V|=n στο οποίο κρατάμε τις υπολογισμένες σε κάθε βήμα του αλγορίθμου αποστάσεις για κάθε κόμβο. Αρχικοποιούμε το διάνυσμα έτσι ώστε dist[s]=0 και dist[u]=∞ όπου u ≠ s και s είναι ο κόμβος αφετηρίας. Επίσης χρησιμοποιούμε ένα διάνυσμα Q, για να κρατάμε τους κόμβους που έχουν προτεραιότητα για να εξεταστούν από τον αλγόριθμο και ένα διάνυσμα S στο οποίο κρατάμε τους κόμβους για τους οποίους έχει βρεθεί η ελάχιστη διαδρομή και ο αλγόριθμος δεν θα ασχοληθεί ξανά μαζί τους. Επίσης για κάθε στοιχείο του V κρατάμε και μία τιμή *prev* η οποία δείχνει στον προηγούμενο κοντινότερο κόμβο στο μονοπάτι προς τον s. Ο αλγόριθμος σε κάθε βήμα, εξάγει από το διάνυσμα Q το στοιχείο που έχει προτεραιότητα και τον εισάγει στον πίνακα S, συγκρίνοντας για κάθε ένα από τα γειτονικά του στοιχεία αν d[y] > d[x] + w(x,y), όπου y το στοιχείο που εξετάζεται και x το γειτονικό στοιχείο το οποίο δεν πρέπει να ανήκει στον πίνακα S. Αν ισχύει η ανισότητα τότε η τιμή d[y] γίνεται ίση με d[x] + w(x,y) και *prev[*y*]*=x. Δηλαδή αν υπολογιστεί μια μικρότερη απόσταση για το στοιχείο y από την ήδη υπολογισμένη τιμή d[y] τότε αυτή η τιμή ενημερώνεται με τη νέα υπολογισμένη απόσταση.

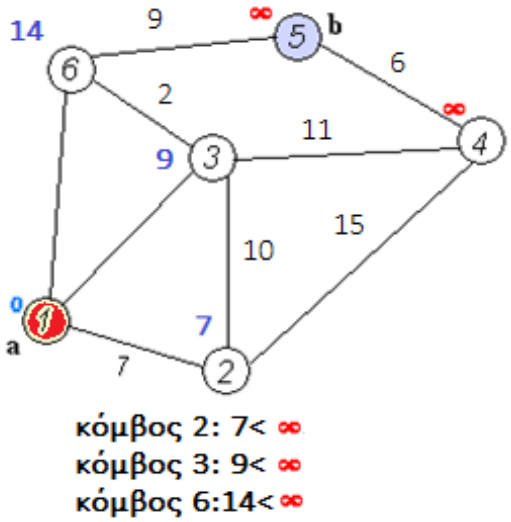
Σημειώνουμε ότι προτεραιότητα έχει ο κόμβος u που στο δεδομένο βήμα έχει την μικρότερη τιμή dist[u] και δεν έχει μπει ακόμα στο διάνυσμα S.

Για τον γράφο της εικόνας 29 ο κόμβος αφετηρίας είναι ο a και τα βάρη είναι αυτά που φαίνονται πάνω από τις ακμές του γράφου. Ο πίνακας dist[] για τον κόμβο a έχει τιμή ίση με το 0 και για όλους τους υπόλοιπους τιμή ίση με ∞. Οπότε στον πίνακα Q προτεραιότητα έχει ο κόμβος 1. Ψάχνουμε να βρούμε το μονοπάτι με το ελάχιστο κόστος από τον κόμβο a στον b.



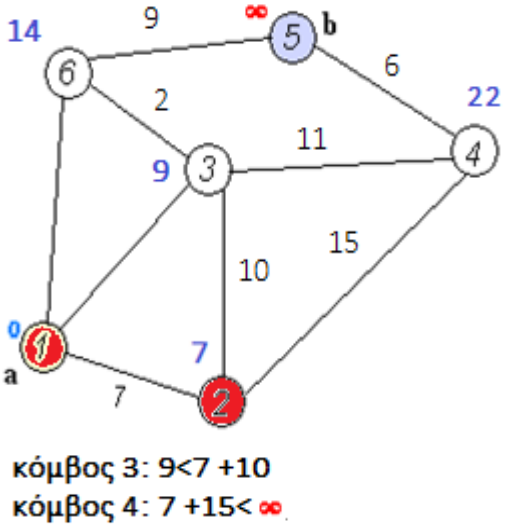

Εικ. 29 Αρχικοποίηση του πίνακα dist ενός γράφου με βάρη, με τη χρήση του Dijkstra

Αφού οριστεί η προτεραιότητα για κάποιον κόμβο, εξετάζονται οι γείτονες του. Ο κόμβος 2 έχει για την ακμή που τον ενώνει με τον 1, βάρος ίσο με 7. Η σύγκριση που γίνεται είναι μεταξύ της τιμής dist[2] και του βάρους της ακμής. Οπότε 7<∞ και τώρα η τιμή dist[2] γίνεται ίση με 7 (dist[2]=7) και prev[2]=1. Αντίστοιχα ο αλγόριθμος για τα επόμενα δύο βήματα συνεχίζει και για τους δύο άλλους κόμβους-γείτονες του 1 και παίρνουμε σαν αποτελέσματα dist[3]=9, dist[6]=14 και prev[3]=1, prev[6]=1. Ο κόμβος 1 εξάγεται από την Q σημειώνεται πλέον ως κλειστός και εισάγεται στον πίνακα *S*.



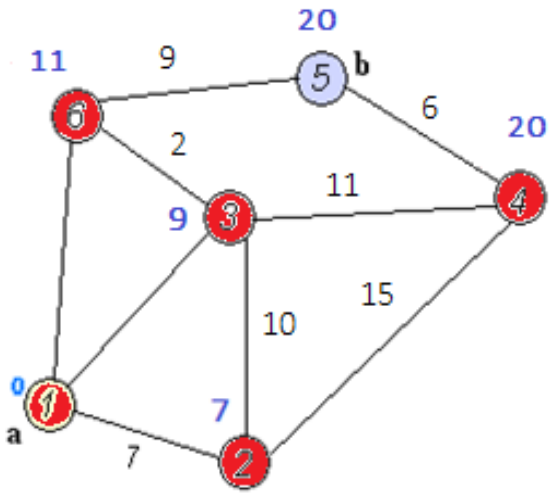

Εικ. 30 Τα πρώτα 3 βήματα του Dijkstra

Ο επόμενος κόμβος του Q που έχει προτεραιότητα, είναι αυτός με την μικρότερη τιμή dist[u], δηλαδή ο 2 με dist[2]=7. Οι γειτονικοί κόμβοι που εξετάζονται είναι ο 3 και 4 και όχι ο 1 ο οποίος είναι τώρα κλειστός. Οι συγκρίσεις που γίνονται φαίνονται στο διπλανό σχήμα και ενημέρωση του πίνακα dist γίνεται μόνο για τον κόμβο 4 εφόσον το μονοπάτι από τον 4 στον 1 μέσω του 2 έχει κόστος 7+15=22<∞. Όταν ολοκληρωθεί η εξέταση όλων των γειτονικών κόμβων του 2 και υπολογιστούν τα ελάχιστα μέχρι στιγμής βάρη, για τους κόμβους αυτούς, ο 2 σημειώνεται και αυτός ως κλειστός εξάγεται από τον πίνακα Q και εισάγεται στον S




Εικ. 31 Τα επόμενα 2 βήματα του Dijkstra

Αντίστοιχα λοιπόν συνεχίζει ο Dijkstra να εξετάζει τους υπόλοιπους κόμβους και να εφαρμόζει την ενημέρωση του πίνακα dist εκεί που πρέπει. Στο τέλος όλοι οι κόμβοι θα είναι κλειστοί και τα νέα βάρη θα δηλώνουν το ελάχιστο κόστος για το μονοπάτι μέχρι τον 1 περνώντας από τους κόμβους που η μεταβλητή prev δηλώνει. Για τον κόμβο 5 θα έχουμε κόστος ίσο με 20, prev[5]=6 και prev[6]=3 prev[3]=1. Αν τώρα αντιστρέψουμε τη σειρά των κόμβων θα ανασκευάσουμε το ζητούμενο μονοπάτι το οποίο είναι 1→3→6→5. Το αποτέλεσμα που παίρνουμε είναι εγγυημένα το ελάχιστο.




Εικ. 32 Ο γράφος την τελική κατάσταση του Dijkstra

## 2.3.3 Αλγόριθμοι επίλυσης λαβυρίνθου

Όπως αναφέραμε προηγουμένως ένας λαβύρινθος μπορεί να αναπαρασταθεί επιτυχώς σαν ένας γράφος. Οπότε όπως είναι λογικό οι αλγόριθμοι που χρησιμοποιούνται για τη διάσχιση ενός γράφου είναι συναφείς με τους αλγορίθμους επίλυσης λαβυρίνθου [8]. Η διαφορά εδώ όμως είναι η εξής. Ένας αλγόριθμος επίλυσης λαβυρίνθου έχει μια συνθήκη τερματισμού η οποία συνήθως είναι η εύρεση ενός τερματικού κόμβου. Οπότε αν πάρουμε για παράδειγμα τον DFS, αυτός θα ξεκίναγε τη διάσχιση του γράφου με τον τρόπο που περιγράψαμε προηγουμένως και θα σταματούσε όταν έβρισκε το σημείο εξόδου. Τελικά θα έβρισκε τη λύση, παίρνοντας βέβαια υπόψιν ότι ένας λαβύρινθος μπορεί να έχει και κυκλικές διαδρομές οπότε θα έπρεπε να κάνουμε και τις κατάλληλες αλλαγές στην υλοποίηση του DFS για να μην μπει σε μια κατάσταση μη τερματισμού. Αυτό σημαίνει ότι ένας λαβύρινθος έχει και κάποιες τοπολογικές ιδιότητες τις οποίες θα πρέπει να παίρνουμε υπόψιν και μια αναζήτηση με τον τρόπο που θα είχαμε με τον DFS, δεν θα εγγυόταν πάντα και το καλύτερο αποτέλεσμα, αν και τελικά συνήθως θα έβρισκε τη λύση. Μια άλλη σημαντική διαφορά είναι, ότι συνήθως ένα πρόβλημα επίλυσης λαβυρίνθου έχει σαν αρχή ότι η πληροφορία για το εσωτερικό του, δεν είναι εκ των προτέρων γνωστή οπότε ένας αλγόριθμος σαν τον Dijkstra δεν θα μπορούσε αποδοτικά να εφαρμοστεί, εφόσον

για να εκτελεί κάθε φορά την τοπικά βέλτιστη λύση, θα έπρεπε να έχει μια εκ των προτέρων γνώση του κόστους μετάβασης από κόμβο σε κόμβο. Για τους παραπάνω λόγους οι αλγόριθμοι επίλυσης λαβυρίνθων είναι συνήθως τυποποιημένες παραλλαγές των αλγορίθμων διάσχισης ενός γράφου η ενός δέντρου. Παρουσιάζουμε παρακάτω κάποιους από τους πιο γνωστούς αλγόριθμους και τους κατατάσσουμε σε δύο βασικές κατηγορίες. Τα κριτήρια μας είναι η αποδοτικότητα ενός αλγορίθμου, το εύρος εφαρμογής τους και η πολυπλοκότητα τους αν και υπάρχουν και άλλοι μέθοδοι κατηγοριοποίησης [9].

Στην πρώτη κατηγορία αλγορίθμων ανήκουν με βάση τα κριτήρια που θέσαμε ο αλγόριθμος τυχαίας προσπέλασης(random mouse walk) ο αλγόριθμος wall-follower και ο αλγόριθμος του Pledge.

### *Αλγόριθμος τυχαίας προσπέλασης*

Ο αλγόριθμος αυτός αποτελεί μια αρκετά τετριμμένη λύση επίλυσης λαβυρίνθου καθώς το μόνο που κάνει, είναι κάθε φορά που φτάνει σε μια διασταύρωση να κάνει μια τυχαία επιλογή για την επόμενη κίνηση του. Ενδεχομένως να φτάσει σε λύση αλλά είναι υπερβολικά κοστοβόρος και χρονοβόρος οπότε δεν συνίσταται σχεδόν ποτέ.

### *Ο αλγόριθμος wall-follower*

Αυτός ο αλγόριθμος χρησιμοποιεί τον κανόνα του αριστερού χεριού ή του δεξιού. Σημειώνουμε ότι δεν έχει καμία σημασία ποιος από τους δύο θα επιλεγεί. Η λογική που στηρίζεται είναι ότι αν σε ένα λαβύρινθο κάποιος ακουμπάει συνέχεια με το αριστερό του χέρι τον τοίχο και σε κάθε διασταύρωση που συναντάει επιλέγει πάντα την στροφή αριστερά, τότε κάποια στιγμή θα φτάσει στο σημείο εξόδου. Βασική προϋπόθεση πρέπει είναι ο λαβύρινθος να είναι συνδεδεμένος ή διαφορετικά αν δεν είναι, τότε η έξοδος να μην βρίσκεται σε κάποιο εσωτερικό σημείο γιατί ποτέ δεν θα το φτάσει και παράλληλα να μην ξεκινάει η αναζήτηση από το ένα εσωτερικό σημείο.

Αυτό για έναν λαβύρινθο σημαίνει ότι κάθε τοίχος που συναντάμε είναι κομμάτι ενός μεγαλύτερου τοίχου και άρα θεωρητικά ο λαβύρινθος είναι σαν ένα τσαλακωμένο χαρτί το οποίο αν το ανοίξουμε, η διάσχιση του λαβυρίνθου θα γίνεται μόνο περιφερειακά. Αν δεν είναι συνδεδεμένος και η έξοδος βρίσκεται σε ένα περιφερειακό σημείο τότε δεν θα έχει εξερευνήσει όλες τις πιθανές διαδρομές, αλλά τελικά θα βρει την έξοδο λόγω της συγκεκριμένης τοπολογικής διάταξης. Είναι πολύ διαδεδομένος λόγω της απλής του λογικής και χρησιμοποιείται κυρίως στις ρομποτικές κατασκευές, όπου οι λαβύρινθοι, είναι συχνά τέτοιας φύσης. Παρ' όλα αυτά επειδή ούτε και αυτός ο αλγόριθμος εγγυάται σίγουρα μια λύση αν και είναι σαφώς πιο ισχυρός από τον πρώτο που παρουσιάσαμε τον κατατάσσουμε στην πρώτη κατηγορία.

### *Ο αλγόριθμος του Pledge*

Ο αλγόριθμος αυτός χρησιμοποιείται κυρίως για αποφυγή εμποδίων σε ένα λαβύρινθο που αναπαρίσταται σαν μη συνδεδεμένος γράφος. Επειδή και αυτός ο αλγόριθμος προσπαθεί να κρατάει το εμπόδιο-τοίχο στην αριστερή (ή δεξιά) πλευρά όπως και ο wall follower στην ουσία επιλύει το πρόβλημα που περιγράψαμε πιο πάνω όπου η αναζήτηση ξεκινάει από ένα εσωτερικό σημείο του λαβυρίνθου και το μονοπάτι αυτό, είναι αποσυνδεμένο από τα υπόλοιπα. Σε αυτή την περίπτωση ο wall follower θα έμπαινε σε έναν ατέρμονο κύκλο (βρόγχο) από τον οποίο δεν θα μπορούσε να ξεφύγει. Ο τρόπος με τον οποίο ο *Pledge,* λύνει αυτό το πρόβλημα είναι χρησιμοποιώντας μια μεταβλητή που μετράει τις στροφές που εκτελεί καθώς και το συνολικό άθροισμα σε μοίρες. Ενώ μπορεί να ξεφύγει από ένα εσωτερικό σημείο του λαβυρίνθου και να συνεχίσει την πορεία του προς την έξοδο, δεν είναι ικανός να κάνει το αντίστροφο. Δηλαδή από ένα αρχικό σημείο στο εξωτερικό του λαβυρίνθου δεν μπορεί να βρει την έξοδο αν αυτή βρίσκεται σε κάποιο αποσυνδεμένο κομμάτι του λαβυρίνθου. Κατατάσσουμε και αυτόν τον αλγόριθμο σε αυτή την κατηγορία, παρότι είναι αρκετά πιο αποτελεσματικός από τους δύο προηγούμενους, δεν εγγυάται όμως ότι θα βρει την λύση αν αυτή υπάρχει.

### *Ο αλγόριθμος του Tremaux*

Ο αλγόριθμος αυτός, αναπτύσσει μια πολύ καλή μέθοδο επίλυσης λαβυρίνθων που βασίζεται στο μαρκάρισμα των μονοπατιών και εγγυάται να βρει τη λύση για κάθε λαβύρινθο που έχει καλά ορισμένα μονοπάτια. Τον παρουσιάζουμε με την μορφή τεσσάρων κανόνων [10]. Σημειώνουμε ότι νέο μονοπάτι ή κόμβος είναι αυτό το οποίο

για πρώτη φορά διασχίζουμε ή συναντάμε και παλιό είναι αυτό που το έχουμε διασχίσει ή το έχουμε συναντήσει ξανά στο παρελθόν.

***Κανόνας νο1.*** Κανένα μονοπάτι δεν μπορεί να διασχιστεί πάνω από δύο φορές.

***Κανόνας νο2.*** Όταν φτάνουμε σε ένα νέο κόμβο επιλέγουμε όποιο μονοπάτι θέλουμε, ή αποφασίζουμε στην τύχη.

***Κανόνας νο3.*** Όταν από ένα καινούργιο μονοπάτι φτάνουμε σε ένα παλιό ή όταν φτάνουμε σε ένα αδιέξοδο, γυρίζουμε πίσω από τη διαδρομή που φτάσαμε εκεί.

***Κανόνας νο4.*** Όταν από ένα παλιό μονοπάτι φτάνουμε σε ένα παλιό κόμβο τότε επιλέγουμε ελεύθερα ένα καινούργιο μονοπάτι αν υπάρχει. Αλλιώς ακολουθούμε το πρώτο μονοπάτι που μας είχε οδηγήσει στον κόμβο εξαρχής.

Αν δεν υπάρχει λύση ο αλγόριθμος θα μας φέρει ξανά στον πρώτο κόμβο με όλα τα μονοπάτια μαρκαρισμένα δύο φορές.

### *Ο αλγόριθμος Dead End Filling*

Ο αλγόριθμος είναι και αυτός απλός στο σκεπτικό του και εγγυάται να βρει τη λύση αν αυτή υπάρχει με την προϋπόθεση ο λαβύρινθος να έχει αδιέξοδα. Αποτελείται από δύο κανόνες. Ο πρώτος κανόνας ορίζει ότι, όταν συναντήσουμε ένα αδιέξοδο, τότε μαρκάρουμε με μια γραμμή όλη τη διαδρομή μέχρι να βρούμε μια διασταύρωση. Όταν το κάνουμε αυτό για όλα τα μονοπάτια που οδηγούν σε αδιέξοδο τότε το μόνο μονοπάτι που δεν θα έχει μαρκαριστεί θα είναι η λύση. Αυτός ο αλγόριθμος προϋποθέτει να γνωρίζουμε εξαρχής που είναι τα αδιέξοδα για να μπορεί να εφαρμοστεί.

Υπάρχουν και άλλοι αλγόριθμοι που ίσως θα μπορούσαμε να αναφέρουμε. Δεν το κάνουμε όμως γιατί στην ουσία αποτελούν κυρίως παραλλαγές των προηγούμενων.

# Κεφάλαιο 3

## Παρουσίαση του προβλήματος

Το πρόβλημα που καλούμαστε να επιλύσουμε αφορά τόσο την κατασκευή του ρομπότ βασισμένο στο arduino, (σωστή συμπεριφορά του στο περιβάλλον του λαβυρίνθου, επικοινωνία με υπολογιστή) όσο και το πρόβλημα της χαρτογράφησης αλλά και του σχεδιασμού των κινήσεων που χρειάζονται για να βρεθεί τελικά μια λύση για το λαβύρινθο σε συνδυασμό με την επιθυμία μας να γίνεται αυτό με έναν τρόπο ελαχιστοποίησης των αναγκαίων κινήσεων.

## 3.1 Το πρόβλημα του σχεδιασμού

Γενικά το πρόβλημα του σχεδιασμού (planning) [11] αφορά το κομμάτι εκείνο, που είναι υπεύθυνο για την κατάστρωση όλων των απαραίτητων ενεργειών που πρέπει να γίνουν για την επίτευξη του στόχου που έχει οριστεί. Ο σχεδιασμός εμπεριέχει το στοιχείο της λογικής για την αναπαράσταση των καταστάσεων όπως επίσης και το στοιχείο της αναζήτησης για την εύρεση μιας ικανοποιητικής λύσης. Πάντα πριν κάνουμε μια κίνηση πρέπει να έχουμε μια εικόνα του τι αποτελέσματα αυτή μπορεί να επιφέρει έτσι ώστε αν χρειάζεται να την αποφύγουμε ή αλλιώς να την επιδιώξουμε. Στη συγκεκριμένη εργασία το πρόβλημα αυτό αφορά κυρίως τη διαδικασία μέσω στης οποίας βρίσκουμε τη λύση για το λαβύρινθο αποφεύγοντας για παράδειγμα κυκλικές διαδρομές και καταφέρνοντας να εξερευνήσουμε όλα τα πιθανά μονοπάτια. Παίρνει επίσης υπόψιν της η διαδικασία αυτή να προσπαθεί να κάνει την διάσχιση του λαβυρίνθου με τέτοιο τρόπο ώστε να μην αφήνει πίσω της πολλά ανεξερεύνητα μονοπάτια τα οποία θα χρειαστούν αργότερα να εξερευνηθούν κάτι το οποίο θα αυξήσει τις κινήσεις του ρομπότ και άρα θα καθυστερήσει και το πρόβλημα της χαρτογράφησης. Τέλος, το επόμενο θέμα που αφορά το πρόβλημα του σχεδιασμού είναι η εύρεση των ελαχίστων διαδρομών. Με ποιο τρόπο δηλαδή αποφασίζουμε, ότι για να πάμε από ένα αρχικό σημείο σε ένα άλλο τελικό, η διαδρομή που επιλέγουμε θα είναι και η μικρότερη, κάτι το οποίο είναι ένα από τα βασικότερα θέματα της εργασίας.

## 3.2 Το πρόβλημα της χαρτογράφησης

Το πρόβλημα της χαρτογράφησης (mapping) αφορά τον τρόπο με τον οποίο αναπαριστούμε τον φυσικό χώρο έτσι ώστε να μπορούμε να εντοπίσουμε τη θέση μας μέσα σε αυτόν. Η αναπαράσταση του χώρου μπορεί να είναι είτε μετρική είτε τοπολογική. Η μετρική αναπαράσταση υποθέτει ένα δισδιάστατο μοντέλο του χώρου όπου τοποθετεί όλα τα αντικείμενα και προσομοιάζει τον τρόπο με τον οποίο οι άνθρωποι αντιλαμβάνονται τον χώρο. Η τοπολογική αναπαράσταση υποθέτει σημεία και τη σχέση μεταξύ αυτών. Για παράδειγμα μια τέτοια αναπαράσταση είναι ένας γράφος όπου κάθε κόμβος είναι μια πόλη και οι ακμές που συνδέουν τους κόμβους η απόσταση μεταξύ των πόλεων. Στη συγκεκριμένη εργασία ψάχνουμε να βρούμε ένα τρόπο να αναλύουμε τα δεδομένα που παίρνουμε από τον λαβύρινθο, ώστε να φτιάξουμε ένα σύστημα χαρτογράφησης ικανό να μας δίνει τα απαραίτητα στοιχεία για να ξέρουμε κάθε φορά ποια είναι η θέση του οχήματος στο λαβύρινθο αλλά και προς τα πού μπορεί κάθε φορά να κινηθεί [12][13].

## 3.3 Το ρομποτικό σύστημα

Με τον όρο ρομπότ (robot) εννοούμε εκείνο το σύστημα που έχει τη δυνατότητα να λαμβάνει ερεθίσματα από το περιβάλλον του και να μπορεί να επενεργεί σε αυτό με σκοπό την επίτευξη του στόχου για το οποίο κατασκευάστηκε. Μπορούμε να χωρίσουμε τα ρομποτικά συστήματα σε δύο κατηγορίες: τα στατικά και τα κινητά. Στατικά είναι τα ρομποτικά συστήματα που έχουν ένα σταθερό σημείο αναφοράς και οι κινήσεις των στοιχείων δράσης τους βασίζονται στην αναφορά αυτή. Κινητά είναι τα ρομποτικά συστήματα που το σημείο αναφοράς τους μεταβάλλεται συνεχώς. Ένα ρομπότ μπορεί να είναι αυτόνομο ή ημιαυτόνομο όπου σε αυτή την περίπτωση χρειάζεται κάποια εξωτερική παρέμβαση για να λειτουργήσει. Στην περίπτωση μας καλούμαστε να φτιάξουμε λοιπόν ένα κινητό ρομποτικό όχημα το οποίο εφόσον για να λειτουργεί σωστά πρέπει να συνεργάζεται με το πρόγραμμα στον υπολογιστή οπότε και ανήκει στην κατηγορία των ημιαυτόνομων κατασκευών.

## 3.4 Σχετικές εργασίες

Υπάρχουν χιλιάδες εργασίες με θέμα αυτόνομα ή ημιαυτόνομα ρομποτικά οχήματα. Για να μπορούμε να συγκρίνουμε την εργασία μας με άλλες πρέπει πρώτα να κάνουμε μια κατηγοριοποίηση και να δούμε που ανήκει η δικιά μας. Καταρχήν το όχημα που αναπτύξαμε στο πλαίσιο της εργασίας μας είναι βασισμένο στον προγραμματισμό μικροελεγκτή και δεν έχει ένα ενσωματωμένο ολοκληρωμένο υπολογιστικό σύστημα, οπότε η σύγκριση με όλα αυτά τα ρομπότ που έχουν για παράδειγμα ισχυρούς επεξεργαστές θα ήταν λάθος. Δεύτερο, η εργασία μας έχει σκοπό την επίλυση λαβυρίνθου τυπωμένο σε χαρτί οπότε το πλαίσιο το οποίο χαρτογραφεί δεν μπορεί να συγκριθεί για παράδειγμα με τον φυσικό κόσμο που κάποιες άλλες ρομποτικές συσκευές χαρτογραφούν. Η δικιά μας δουλειά περιγράφει ένα ημιαυτόνομο όχημα που συνεργάζεται με κάποιο πρόγραμμα στον υπολογιστή για την επίλυση του λαβυρίνθου. Χρησιμοποιεί ασύρματη επικοινωνία για να το επιτύχει όπως και για να κάνει αποτύπωση του χάρτη σε ένα γραφικό πρόγραμμα. Χρησιμοποιεί τον προγραμματισμό ενός AVR μικροεπεξεργαστή και ανήκει στην κατηγορία των μεσαίων ταχυτήτων.

Παρακάτω παρουσιάζουμε τρία γνωστά παραδείγματα επίλυσης λαβύρινθου τα οποία αποτέλεσαν και έμπνευση για τη δική μας δουλειά.

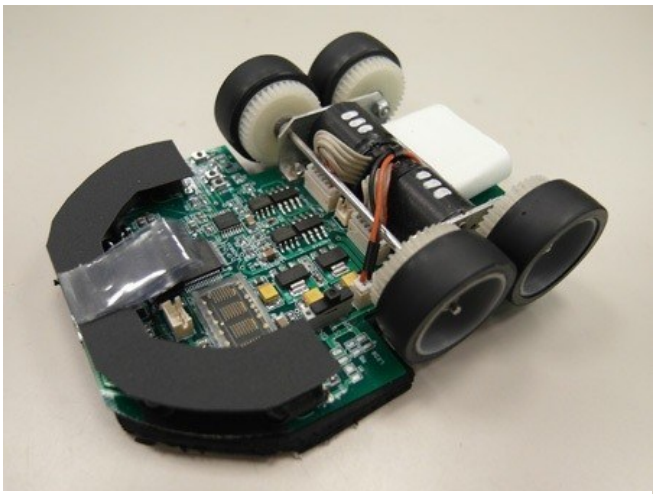
Το πρώτο είναι το NXT Mindstorm της LEGO το οποίο έχει μια βασική διαφορά ότι χρησιμοποιεί έναν αισθητήρα υπερήχων για να αντιμετωπίζει εμπόδια και όχι για να ακολουθεί μια γραμμή όπως κάνουμε εμείς. Όλα τα παραδείγματα που είδαμε λύνουν μεν τον λαβύρινθο με τα εμπόδια αλλά αφορούν πάντα πολύ μικρούς λαβυρίνθους χωρίς κυκλικές διαδρομές. Επίσης όλα τα παραδείγματα υλοποιούσαν τον αλγόριθμο wall follower ή μια απλή έκδοση του DFS. Η υλοποίηση αυτή σημειώνουμε ότι αφορά αργά ρομποτικά οχήματα και δεν μπορεί να αναπτύξει μεγάλες ταχύτητες. Ο προγραμματισμός αφορά έναν επεξεργαστή και όχι μικροελεγκτή.

Εικ. 33 Το NXT μέσα σε ένα λαβύρινθο

Το δεύτερο είναι το 3pi της Pololu. Το συγκεκριμένο είναι κατασκευασμένο για να ακολουθεί γραμμή και μοιάζει στη δικιά μας προσέγγιση πιο πολύ από το NXT επειδή προγραμματίζει έναν μικροελεγκτή υπεύθυνο για την κίνηση του. Μπορεί να λύνει μεγάλους λαβυρίνθους πολύ γρήγορα αλλά δεν έχουμε δει ακόμα κάποιο παράδειγμα που να αντιμετωπίζει τα προβλήματα των κυκλικών διαδρομών. Επίσης είναι προγραμματισμένο για δουλεύει μόνο του χωρίς τη συνεργασία με κάποιο άλλο πρόγραμμα όπως κάνουμε στη δική μας εργασία. Δίνει όμως τη δυνατότητα στο χρήστη να τα επαναπρογραμματίσει και αν κάποιος ήθελε θα μπορούσε να ακολουθήσει τη δική μας προσέγγιση χρησιμοποιώντας το 3pi. Το συγκεκριμένο το κατατάσσουμε στα πολύ γρήγορα ρομποτικά οχήματα.

Εικ. 34 Το 3pi πάνω σε μία μαύρη γραμμή

Το τρίτο που παρουσιάζουμε είναι το παράδειγμα των ρομποτικών οχημάτων που χρησιμοποιούν στο διαγωνισμό micromouse. Αυτό ίσως είναι το εξελιγμένο παράδειγμα που έχουμε δει γιατί καταφέρνουν να λύνουν πολύ περίπλοκους και πολύ μεγάλους λαβυρίνθους σε πραγματικά ελάχιστο χρόνο. Ο προγραμματισμός τους γίνεται κυρίως από επαγγελματίες αν



Εικ. 35 Εικόνα ενός ρομπότ που συμμετέχει στο διαγωνισμό micromouse

και οποιοσδήποτε θα μπορούσε να προσπαθήσει να φτιάξει μια απλή έκδοση. Χρησιμοποιούν υπέρυθρους αισθητήρες όχι για τον εντοπισμό γραμμής αλλά για τον εντοπισμό όρθιων εμποδίων. Δυστυχώς τα καλύτερα παραδείγματα προγραμματισμού αυτών των ρομπότ κρατάνε τον κώδικα κλειστό γιατί όπως είπαμε συμμετέχουν σε παγκόσμιους διαγωνισμούς όπου και η παραμικρή διαφορά στον κώδικα θα κάνει τη διαφορά. Για την αναζήτηση χρησιμοποιούν πολλούς διαφορετικούς αλγορίθμους και όπως επίσης κάνουν και χρήση των αλγορίθμων εύρεσης συντομότερων μονοπατιών. Η κατηγορία αυτή ανήκει στα πολύ γρήγορα ρομποτικά οχήματα. Ούτε για αυτή την κατηγορία είδαμε να υπάρχει κάποιο παράδειγμα ασύρματης επικοινωνίας με τον υπολογιστή.

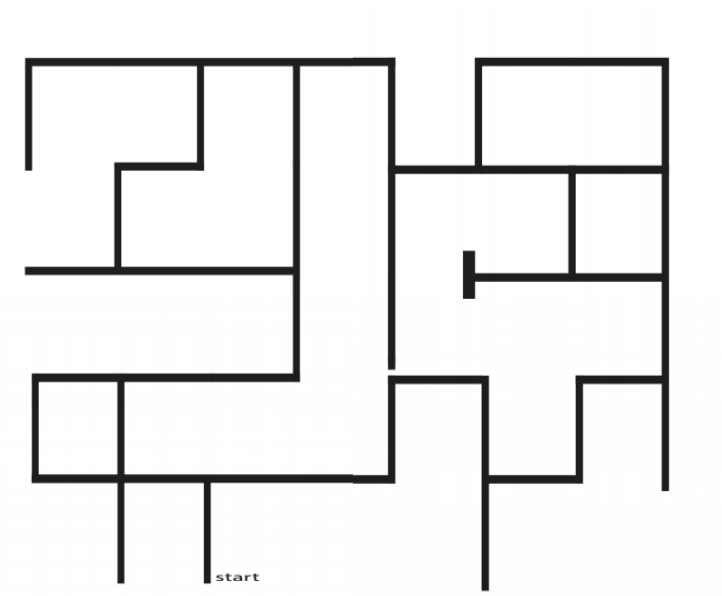
# Κεφάλαιο 4

## Η δική μας προσέγγιση

### 4.1 Ο λαβύρινθος που κατασκευάσαμε

Για να ξεκινήσουμε την περιγραφή της προσέγγισης μας θα πρέπει πρώτα να ορίσουμε το φυσικό χώρο κίνησης, πάνω στον οποίο εκτελούνται όλες οι ενέργειες. Ο λαβύρινθος αποτυπώνεται με μαύρες λωρίδες πάνω σε μια άσπρη επιφάνεια. Ένας τρόπος υλοποίησης είναι με τη χρήση μαύρης κολλητικής ταινίας πάνω σε μια σκληρή επίπεδη επιφάνεια και ο άλλος ο τρόπος, αυτός που χρησιμοποιούμε και στην παρούσα εργασία, είναι η τύπωση των μαύρων γραμμών σε άσπρες σελίδες. Συγκεκριμένα χρησιμοποιούμε τέσσερα φύλλα Α0 αλλά αυτό δεν είναι καθόλου δεσμευτικό για κάποιον που θέλει να χρησιμοποιήσει έναν μεγαλύτερο χώρο κίνησης. Βασικό στοιχείο είναι η επιφάνεια να είναι λεία, επίπεδη και να ασκεί σχετικά καλή τριβή ώστε να μην υπάρχουν φαινόμενα ολίσθησης ή αλλά απρόβλεπτα που μπορούν να προκαλέσουν σημαντική αστάθεια ή και ζημιά στο όχημα που χρησιμοποιούμε.

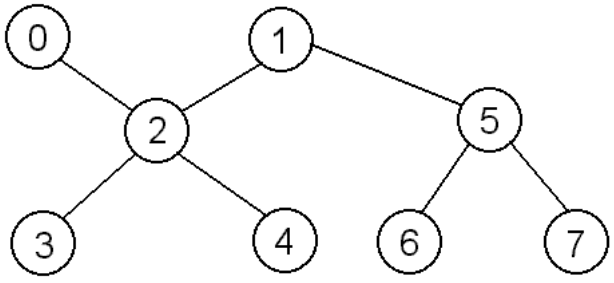
Η τοπολογία του λαβυρίνθου χαρακτηρίζεται από ευθείες γραμμές πάχους 22 χιλιοστών εκατοστών και μήκους 30 εκατοστών. Οι γραμμές ενώνονται μεταξύ τους είτε σε οριζόντια είτε σε κάθετη διάταξη. Δεν επιτρέπονται να δημιουργούνται σχήματα τύπου “Y” ή να χρησιμοποιούνται καμπύλες γραμμές. Κάθε γραμμή μπορεί να ενώνεται με μια άλλη με οποιονδήποτε από τους δύο τρόπους που αναφέραμε με την προϋπόθεση η ένωση να ξεκινάει εκεί που τελειώνει η προηγούμενη.



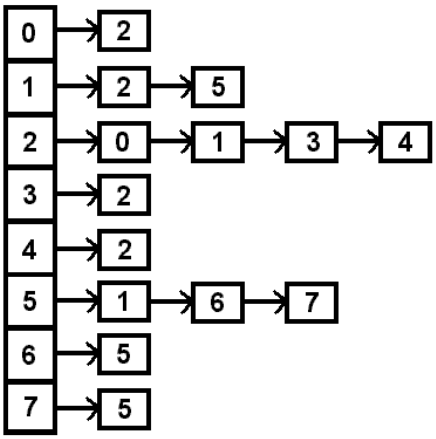

Εικ. 36 Ο λαβύρινθος που κατασκευάσαμε

Τον λαβύρινθο τον επεξεργαζόμαστε αλγοριθμικά σαν ένα γράφο, σύμφωνα με τον τρόπο που περιγράψαμε στην ενότητα 2.3. Οπότε, από εδώ και στο εξής όταν αναφερόμαστε στο γράφο, θα εννοούμε την αλγοριθμική αποτύπωση του λαβυρίνθου και θα χρησιμοποιούμε τις έννοιες του κόμβου, της διάσχισης γράφου κτλ. και όταν αναφερόμαστε στο λαβύρινθο θα εννοούμε τον φυσικό χώρο που αυτός περιγράφει.

Για τον γράφο χρησιμοποιούμε ένα διάνυσμα 1×N, όπου N ο αριθμός των κόμβων, στο οποίο αποθηκεύουμε κάθε κόμβο ξεχωριστά και μέσω αυτού έχουμε πρόσβαση σε όλους τους υπόλοιπους γειτονικούς του κόμβους (*adjacency list*). Ένα παράδειγμα χρήσης αυτής της δομής για έναν γράφο με N=8 φαίνεται στην εικόνα 38.




Εικ. 37 Γράφος 8 κόμβων




Εικ. 38 Η λίστα γειτνίασης γράφου 8 κόμβων

## 4.2 Κίνηση και ανάγνωση του φυσικού χώρου του λαβυρίνθου

Το σύνολο των ενεργειών που λαμβάνουν χώρο στο λαβύρινθο χωρίζεται σε δύο βασικές κατηγορίες. Η πρώτη κατηγορία αφορά την κίνηση που κάνει το όχημα και η δεύτερη κατηγορία αφορά την ανάγνωση δεδομένων από το περιβάλλον του. Όλος ο κώδικας που είναι υπεύθυνος για τον έλεγχο της κίνησης, δηλαδή το πόσο θα στρίψει ή αν θα διορθώσει την πορεία του εκτελείται πάνω στο arduino οπότε θα κάνουμε και παραπομπές σε βασικά κομμάτια του κώδικα για να δείξουμε και τι γίνεται στο εσωτερικό του arduino.

### 4.2.1 Η κίνηση πάνω στο λαβύρινθο

Υπάρχουν τέσσερις βασικές κινήσεις που υλοποιούμε και δύο κινήσεις διόρθωσης της πορείας. Οι βασικές είναι η κίνηση σε ευθεία, η στροφή αριστερά, η στροφή δεξιά και η αναστροφή. Οι κινήσεις διόρθωσης αφορούν δύο συναρτήσεις που καλούνται κάθε φορά που υπάρχει ελαφρά μετατόπιση από το κέντρο της γραμμής και αφορούν τη διόρθωση της πορείας προς τα δεξιά ή προς τα αριστερά. Μια στροφή αριστερά ή δεξιά είναι μια κίνηση περίπου $90^0$ μοιρών και εκτελείται σε δύο στάδια. Το πρώτο εκτελείται μέχρι να βρεθεί η καινούργια γραμμή που θα ακολουθήσουμε και το δεύτερο μέχρι να βρεθεί το ρομπότ ακριβώς πάνω από την γραμμή. Αυτό γίνεται για δύο λόγους, οι οποίοι θα φανούν καλύτερα πιο κάτω, στην ενότητα που θα περιγράψουμε τον τρόπο ανάγνωσης της πληροφορίας από τους αισθητήρες. Στην ουσία όμως γίνεται για λόγους καλύτερης ακρίβειας, σε συνδυασμό με την ταχύτητα εκτέλεσης της στροφής. Οι συναρτήσεις διόρθωσης της πορείας μπορούν να εκτελεστούν οποιαδήποτε στιγμή το ρομπότ είτε έχει ολοκληρώσει μια στροφή $90^0$ ή $180^0$ μοιρών και οποτεδήποτε οι αισθητήρες δεν βλέπουν κάποια διασταύρωση, οπότε και η κίνηση είναι ευθεία, προς μια συγκεκριμένη κατεύθυνση. Η αναστροφή γίνεται σε τρεις κινήσεις. Εδώ ίσως χάνουμε παραπάνω σε ταχύτητα αλλά επειδή είναι πιο περίπλοκη κίνηση, προτιμούμε να επενδύσουμε στην ακρίβεια. Η πρώτη αφορά μια γρήγορη ημιαναστροφή περίπου $90^0$ μοιρών, η δεύτερη είναι μία πιο αργή

ημιαναστροφή περίπου $90^0$ μοιρών προς την ίδια όμως κατεύθυνση, ακολουθούμενη από ανάγνωση των αισθητήρων και αν χρειάζεται μία πιο ειδική διόρθωση της πορείας που υλοποιείται εσωτερικά, στη συνάρτηση που κάνει την αναστροφή.

### 4.2.2 Μια ματιά στο εσωτερικό του arduino

Το πρώτο πράγμα που γίνεται στον κώδικα είναι οι δηλώσεις μεταβλητών. Οι μεταβλητές δηλώνονται έξω από την κύρια συνάρτηση την loop() και πριν την setup(). Όπως αναφέραμε και στην ενότητα που περιγράψαμε τον motor controller και τα dc motors, κάθε ένα motor έχει δύο ακροδέκτες για την τροφοδοσία που συνδέονται σε δύο pins του controller. Αυτοί οι ακροδέκτες συνδέονται με μια τάση εξόδου από τον controller και τη γείωση. Αυτό που προγραμματίζουμε και φαίνεται στον παρακάτω κώδικα, είναι μεταβλητές λογικής-TTL 5V που είναι έξοδοι από το **arduino** και είσοδοι στον **controller** και ελέγχουν αν σε κάποιο ακροδέκτη θα εφαρμοστεί τάση (logic HIGH) ή όχι (logic LOW). Η μεταβλητή enablePin είναι ένα επίπεδο πιο πάνω και ορίζει αν ένα motor θα είναι ενεργό ή όχι. Οπότε αν πάρουμε για παράδειγμα το motor_right οι τρείς πρώτες μεταβλητές έχουν για πίνακα αληθείας την εικόνα 15.

```
const int motor_right1 = 6;    //arduino pin 6 to H-bridge leg 1 (pin 2, 1A)
const int motor_right2 = 4;    // arduino pin 4 to H-bridge leg 2 (pin 7, 2A)
const int enablePin1 = 11;    // arduino pin 11 to H-bridge enable pin for motor_right


const int motor_left1 = 7;    // arduino pin 7 to H-bridge leg 1 (pin 15, 4A)
const int motor_left2 = 8;    // arduino pin 8 to H-bridge leg 2 (pin 10, 3A)
const int enablePin2 = 3;    // arduino pin 3 to H-bridge enable pin for motor left
```

Το ίδιο ισχύει φυσικά και το δεύτερο motor και τώρα είμαστε έτοιμοι να περάσουμε στην πρώτη βασική συνάρτηση κάθε προγράμματος arduino που είναι η setup(). Εδώ ορίζουμε τι είδους θέλουμε να είναι οι μεταβλητές που δηλώσαμε πιο πάνω, δηλαδή αν θα είναι έξοδοι ή είσοδοι. Είπαμε και προηγουμένως ότι θέλουμε να ελέγχουμε τη λειτουργία του controller με βάση τον πίνακα αληθείας που παρουσιάσαμε, οπότε οι μεταβλητές αυτές θα είναι **έξοδοι** από το arduino και **είσοδοι**

στον controller. Έτσι στο εσωτερικό της setup(), η οποία να θυμίσουμε ότι εκτελείται μία και μόνο φορά σε αντίθεση με την loop() που εκτελείται διαρκώς, θα έχουμε:

```
void setup () {

    // set all the other pins you're using as outputs:
    pinMode (motor_left1, OUTPUT);
    pinMode (motor_left2, OUTPUT);
    pinMode (motor_right1, OUTPUT);
    pinMode (motor_right2, OUTPUT);
    pinMode (enablePin1, OUTPUT);
    pinMode (enablePin2, OUTPUT);


    // set enablePin 1 and 2 HIGH so that motor can turn on:
    digitalWrite (enablePin1, HIGH);
    digitalWrite (enablePin2, HIGH);
}
```

Αφού ορίσαμε τώρα, ότι οι μεταβλητές μας είναι έξοδοι, μπορούμε να χρησιμοποιήσουμε τις εντολές digitalWrite() ή analogWrite() για τα pins που θέλουμε να έχουν ψηφιακή έξοδο ή αναλογική αντίστοιχα. Όπως αναφέραμε και προηγουμένως αναλογική έξοδο παίρνουμε από ένα pin ψηφιακής λογικής, χρησιμοποιώντας την τεχνική του **PWM** σύμφωνα με την οποία ορίζουμε τον κύκλο λειτουργίας του παλμού, δηλαδή πόση ώρα θα είναι σε λογικό επίπεδο **HIGH** σε σχέση με την ώρα που θα βρίσκεται σε λογικό επίπεδο **LOW.** Η γρήγορη αυτή εναλλαγή μας δίνει αποτελέσματα αναλογικά, μεταξύ της τάσης Vcc και της γείωσης και έχει σαν αποτέλεσμα τον έλεγχο της ταχύτητας ενός motor, εφόσον πτώση της τάσης που εφαρμόζουμε έχει σαν αποτέλεσμα ελάττωση της ταχύτητας.

Στο παράδειγμα κώδικα που ακολουθεί, στη συνάρτηση loop(), θέτουμε τις κατάλληλες τιμές για τις παραπάνω μεταβλητές ώστε να έχουμε σαν αποτέλεσμα την κίνηση προς τα εμπρός.

```
void loop () {

   // motor left forward
   digitalWrite (motor_left1, HIGH);   // set motor_left1 to HIGH
   digitalWrite (motor_left2, LOW);   );   // set motor_left2 to LOW


   // motor right forward
   digitalWrite (motor_right1, LOW);   // set motor_right1 to LOW
   digitalWrite (motor_right2, HIGH);   // set motor_right1 to HIGH

}
```

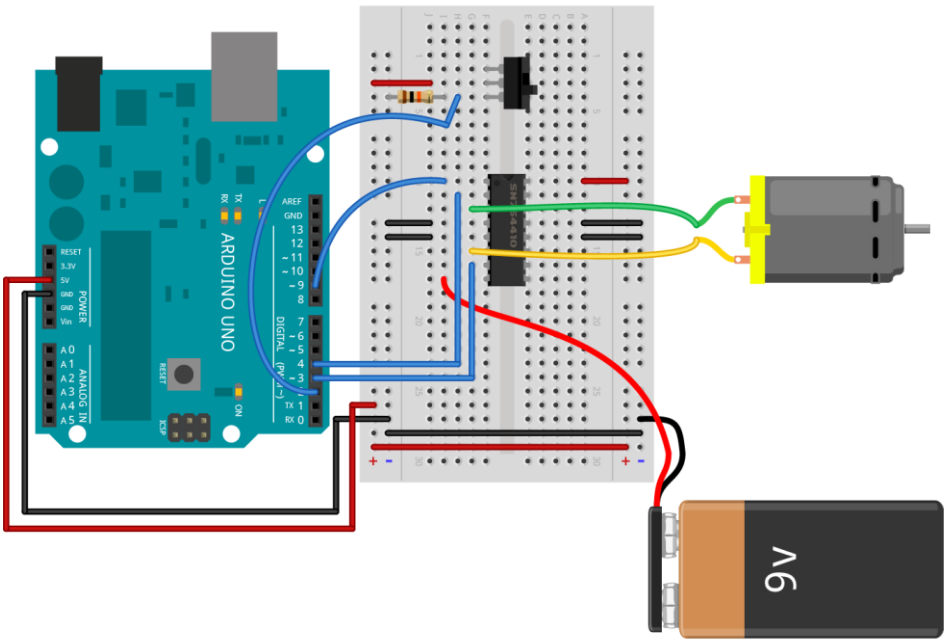
Για αυτό το κομμάτι κώδικα τρία πράγματα θα θέλαμε να σημειώσουμε. Πρώτον ότι οι εντολές για να θέσουμε τα δύο motors σε λειτουργία δεν είναι ακριβώς ίδιες. Είναι σαν αντανάκλαση η μία της άλλης και αυτό γίνεται απλά επειδή τα motors είναι τοποθετημένα το ένα απέναντι από το άλλο. Αν θέλαμε να το αλλάξουμε αυτό, και να ακολουθούν την ίδια ακολουθία HIGH, LOW, HIGH, LOW θα μπορούσαμε απλά να συνδέσουμε τους ακροδέκτες του motor_left ανάποδα στον motor_controller. Το δεύτερο είναι ότι δεν φαίνεται πουθενά αυτό που αναφέραμε προηγουμένως για έλεγχο της ταχύτητας με την *PWM*. Στην ουσία όμως, θέσαμε προηγουμένως με τις εντολές digitalWrite (enablePin1, HIGH) και digitalWrite (enablePin2, HIGH) στο κομμάτι του setup() ότι οι μεταβλητές enablePin1 και enablePin2, θα έχουν κύκλο λειτουργίας 100% οπότε η τάση που εφαρμόζεται, είναι η μέγιστη τάση που μας δίνει η τροφοδοσία. Αν θέλαμε να το αλλάξουμε αυτό και πάρουμε για παράδειγμα την μισή τάση θα έπρεπε να είχαμε τοποθετήσει τις εντολές analogWrite(enablePin1, 173) και analogWrite(enablePin2, 173) στην αρχή της *loop*(). Η αλλαγή που κάναμε στη μία περίπτωση με την digitalWrite() με όρισμα την τιμή HIGH και στη δεύτερη περίπτωση με την analogWrite και όρισμα το 173, είναι η υλοποίηση του μηχανισμού PWM που έχουμε περιγράψει. Το τελευταίο που σημειώνουμε είναι ότι στην πραγματικότητα, όλες οι παραπάνω εντολές είναι προτιμότερο να δηλώνονται στο σώμα μιας συνάρτησης πχ της forward() η οποία θα καλείται όποτε κρίνουμε εμείς. Από τη στιγμή που κάποιο pin πάρει μια τιμή HIGH ή LOW αυτό δεν αλλάζει μέχρι κάποια άλλη εντολή να το αλλάξει ή να κοπεί η σύνδεση στο κύκλωμα.

Στο παράδειγμα που ακολουθεί δίνουμε ένα ολοκληρωμένο παράδειγμα κώδικα που ορίζει την κίνηση ευθεία για 2 sec, το σταμάτημα της κίνησης για 1 sec και μετά το ξεκίνημα με αργή κίνηση ευθεία, για άλλα 2 sec. Οι δηλώσεις των μεταβλητών και η setup() μένουν ίδια με προηγουμένως. Ο κώδικας έχει σαν αποτέλεσμα αυτή η ακολουθία εντολών να εκτελείται η μία μετά την άλλη διαρκώς.

```
void loop () {

      forward ();
        delay (2000);

      motor_stop ();
        delay (1000);

      slow_forwrad ();
        delay (2000);

}
void forward ()
{

    // motor left forward
    digitalWrite (motor_left1, HIGH);   // set motor_left1 to HIGH
    digitalWrite (motor_left2, LOW);   );   // set motor_left2 to LOW


    // motor right forward
    digitalWrite (motor_right1, LOW);   // set motor_right1 to LOW
    digitalWrite (motor_right2, HIGH);   // set motor_right1 to HIGH
}
```

```
void slow_forward ()
{

   analogWrite (enablePin1, 173); //PWM results to setting Vcc to Vcc/2
   analogWrite (enablePin2, 173); // PWM results to setting Vcc to Vcc/2
   // motor left forward
    digitalWrite (motor_left1, HIGH);   // set motor_left1 to HIGH
    digitalWrite (motor_left2, LOW);    // set motor_left2 to LOW

    // motor right forward
    digitalWrite (motor_right1, LOW);   // set motor_right1 to LOW
    digitalWrite (motor_right2, HIGH);   // set motor_right1 to HIGH

}
void motor_stop () {
      digitalWrite (motor_left1, LOW);
      digitalWrite (motor_left2, LOW);
      digitalWrite (motor_right1, LOW);
      digitalWrite (motor_right2, LOW);
}
```

Δείξαμε πώς περίπου λειτουργεί ένας κώδικας στο arduino για τον έλεγχο δύο motors, την κίνηση ευθεία και την αλλαγή της ταχύτητας. Αν τώρα κάνουμε ένα συνδυασμό για το ποιοι ακροδέκτες θα είναι ενεργοί κα ποιοι όχι, όπως και για πόση ώρα ή με πόση ταχύτητα θα κινούνται μπορούμε να συνθέσουμε και τις υπόλοιπες κινήσεις (πχ τη στροφή δεξιά). Στην εικόνα που ακολουθεί φαίνεται η συνδεσμολογία ενός arduino, ενός motor_controller και ενός motor υπό τον έλεγχο ενός διακόπτη.




Εικ. 39 Η συνδεσμολογία μεταξύ ενός arduino του controller motor και δύο motors

## 4.3 Λειτουργία των αισθητήρων

Η λειτουργία των αισθητήρων είναι πολύ σημαντική για τη συνολική λειτουργία και συμπεριφορά του ρομποτικού οχήματος. Υπάρχουν τρεις διαφορετικές περιπτώσεις στις οποίες διαβάζουμε δεδομένα από το περιβάλλον, κάθε μία από τις οποίες εξυπηρετεί ένα διαφορετικό σκοπό.

Όπως αναφέραμε ο QTR είναι μια σειρά από έξι υπέρυθρους αισθητήρες ανάκλασης του φωτός. Είναι ικανός να ξεχωρίζει μια άσπρη από μια μαύρη επιφάνεια και επιστρέφει μία αναλογική τιμή μεταξύ της τάσης τροφοδοσίας Vcc και της γείωσης. Αυτή η τιμή μεταφράζεται από τους ακροδέκτες αναλογικών εισόδων του arduino σε μια τιμή μεταξύ 0 και 1023. Όταν ο αισθητήρας είναι πάνω από μία σκούρα επιφάνεια τότε η τιμή που θα διαβάζουμε είναι υψηλή και όταν είναι πάνω από μία λευκή, η τιμή είναι χαμηλή. Στην πιο καλή περίπτωση η πρώτη τιμή πλησιάζει το 1023 και στην άλλη περίπτωση το 0. Επειδή όμως συχνά, μπορεί να υπάρχει θόρυβος ή ο αισθητήρας να μην είναι σε κατάλληλη απόσταση από το έδαφος, συνήθως θεωρούμε ότι είμαστε πάνω από μία μαύρη επιφάνεια αν η τιμή που διαβάζουμε είναι πάνω από 850 και

διαφορετικά θεωρούμε ότι είμαστε πάνω από μία λευκή επιφάνεια αν η τιμή που διαβάζουμε είναι κάτω από το 40. Υπάρχει και μια άλλη διαφορετική παράμετρος που μπορεί να μας δώσει λανθασμένα αποτελέσματα αν δεν κάνουμε καλή χρήση της διαδικασίας ανάγνωσης των τιμών των αισθητήρων. Αυτή η παράμετρος είναι η κίνηση. Προτιμάμε να μην διαβάζουμε τις τιμές των αισθητήρων κάνοντας χρήση της analogWrite() σε έναν αναλογικό ακροδέκτη, γιατί η ανάγνωση με αυτό τον τρόπο έχει έναν μεγάλο παράγοντα θορύβου που μερικές φορές οδηγεί σε λάθος αποτελέσματα. Χρησιμοποιούμε μια έτοιμη συνάρτηση της βιβλιοθήκης που δίνει η κατασκευάστρια εταιρεία, η οποία παίρνει υπόψιν της το θόρυβο και τον απαλείφει σε έναν μεγάλο βαθμό, κάνοντας δειγματοληψία στις τιμές που διαβάζει και επιστρέφει έναν μέσο όρο. Αυτό όμως σημαίνει ότι καθυστερούμε πιο πολύ τώρα να διαβάσουμε τις τιμές (περίπου 25ms για ανάγνωση σειριακά των τιμών, λόγω ADC, δηλαδή 100 ms αν δειγματολειπτούμε τέσσερις φορές) . Οπότε οι τιμές που αναφέραμε προηγουμένως ισχύουν, αν το ρομπότ είναι σχεδόν ακίνητο ή αν βρίσκεται σε κίνηση και δεν υπάρχει γρήγορη εναλλαγή στις τιμές που διαβάζει. Όταν βρίσκεται σε κίνηση, για παράδειγμα σε μια στροφή, τότε μέσα σε μερικά εκατοστά του δευτερολέπτου μπορεί να βλέπει μία άσπρη επιφάνεια και μετά μία μαύρη. Οπότε η τιμή 500 μπορεί να σημαίνει ότι μόλις πέρασε γρήγορα πάνω από μία άσπρη περιοχή και τώρα περνάει πάνω από μία μαύρη. Είναι λοιπόν και θέμα της υλοποίησης του κώδικα η σωστή ανάγνωση των δεδομένων.

### 4.3.1 Υπολογισμός θέσης και εύρεση γραμμής

Αναφέραμε και προηγουμένως ότι εμείς χρησιμοποιούμε τους έξι από τους οκτώ αισθητήρες που μας δίνει τη δυνατότητα ο QTR. Αυτό σημαίνει ότι όταν οι δύο μεσαίοι, δηλαδή ο 3 και ο 4 αισθητήρας βρίσκονται ακριβώς πάνω από τη γραμμή και οι υπόλοιποι πάνω από την άσπρη επιφάνεια, οι τιμές που θα διαβάζουμε θα είναι περίπου αυτές [ 0, 0, 1000, 1000, 0, 0 ]. Οπότε μπορούμε να υπολογίσουμε την θέση του ρομπότ σε σχέση με τη γραμμή ως εξής:

Η συνάρτηση readLine() της βιβλιοθήκης που χρησιμοποιούμε, αποθηκεύει τις τιμές που διαβάζουμε σε έναν πίνακα π.χ. τον qtr[6} και επιστρέφει την μεταβλητή position. Έτσι ώστε position= (0 × qtr [0] + 1 × qtr [1] + 2 × qtr [2] + 3 × qtr [3] + 4 × qtr [4] + 5 × qtr [5])/2

= (**0** × 0 + **1** × 0 + **2** × 1000 + **3** × 1000 + **4** × 0 + **5** × 0)/2 = **2500**.

Δηλαδή όταν η μεταβλητή position παίρνει τιμή ίση με 2500 τότε βρισκόμαστε ακριβώς πάνω από τη γραμμή και μπορούμε να συνεχίσουμε την πορεία μας ευθεία χωρίς να χρειάζεται να γίνει κάποια διόρθωση. Ένα καλό περιθώριο για το οποίο θεωρούμε ότι δεν χρειάζεται κάποια διόρθωση η πορεία είναι μεταξύ 2450 και 2550. Στην εικόνα 40 που ακολουθεί δείχνουμε τη θέση που βρίσκονται οι αισθητήρες όταν έχουμε για την μεταβλητή position=2500. Αντίστοιχα τώρα αν διαβάσουμε ότι position= 1500 τότε σημαίνει ότι είμαστε πολύ αριστερά εκτός πορείας, γιατί τότε θα είναι πάνω από τη μαύρη γραμμή, ο αισθητήρας 2 και ο 3.

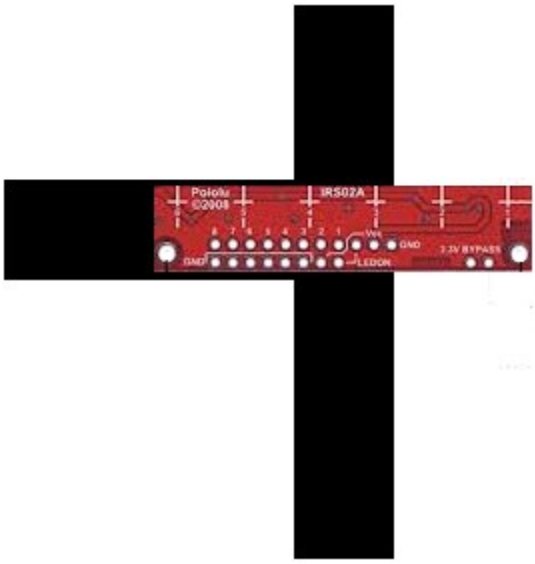

Εικ. 40 Ο αισθητήρας QTR ακριβώς πάνω από μία γραμμή

Η μέθοδος αυτή μας βοηθάει όταν βρισκόμαστε πάνω από μία μαύρη γραμμή και θέλουμε να ελέγξουμε την θέση μας για να δούμε αν χρειάζονται να γίνει κάποια διόρθωση προς τα αριστερά ή τα δεξιά. Όταν όμως βρισκόμαστε πάνω από μία διασταύρωση όπου και οι έξι αισθητήρες μπορεί να έχουν την ίδια τιμή ή όταν είμαστε τελείως εκτός γραμμής όπως όταν γίνεται αναστροφή δεν μας βοηθάει σε κάτι αυτή συνάρτηση.

Στην περίπτωση που το ρομπότ έχει βγει τελείως εκτός πορείας είτε από λάθος είτε επειδή όπως στην αναστροφή ίσως να μην καταφέρει να βρει τη γραμμή, θέλουμε να ξεκινήσει η αναζήτηση της. Αυτό σημαίνει ότι, αν η γραμμή είναι στα αριστερά του αισθητήρα η αναζήτηση είναι στην ουσία μια εντολή while () με συνθήκη πχ qtr[5]<800 και εντολή να στρίβει μέχρι να μην ισχύει η συνθήκη.

## 4.3.2 Λειτουργία σε διασταύρωση ή αδιέξοδο

Στην περίπτωση μιας διασταύρωσης ο αισθητήρας θα έχει για τις μεταβλητές qtr[5] ή qtr[0] ή και για τις δύο, ανάλογα με τόσες γραμμές θα βρει, τιμές κοντά στο 1000 (οι ενδιάμεσοι αισθητήρες προς το παρόν δεν μας ενδιαφέρουν). Οπότε τώρα, από τις τιμές αυτές θα ξέρουμε ότι δεν πρόκειται για λάθος στην πορεία αλλά για μια εύρεση διακλάδωσης. Αυτό σημαίνει ότι πρέπει να γίνουν τα κατάλληλα βήματα, ώστε να δοθεί σήμα στο πρόγραμμα του υπολογιστή για να μπορεί να αποφασιστεί το αν θα στρίψει, και προς τα πού. Επειδή όμως η διακλάδωση δεν έχει μόνο στροφές αριστερά ή δεξιά, αλλά μπορεί να έχει και κάποιο μονοπάτι ευθεία, το πρόγραμμα στο arduino πριν αποφασίσει τι πληροφορία θα στείλει, δίνει εντολή για ένα μικρό βήμα προς τα εμπρός, για να ξαναδιαβαστούν οι τιμές από τον αισθητήρα. Αν μετά το βήμα αυτό, οι μεσαίοι αισθητήρες qtr[2] και qtr[3] έχουν τιμή κοντά στο 1000 σημαίνει ότι βρέθηκε και άλλο υποψήφιο μονοπάτι, οπότε αλλάζει και η πληροφορία που θα αποσταλεί. Η περίπτωση του αδιεξόδου είναι πολύ πιο απλή, μιας και αδιέξοδο σημαίνει ότι ξαφνικά οι αισθητήρες βλέπουν μόνο μια άσπρη επιφάνεια. Οπότε τώρα όλες οι τιμές είναι πολύ χαμηλές και δεν υπάρχει ανάγκη για περαιτέρω ελέγχους.

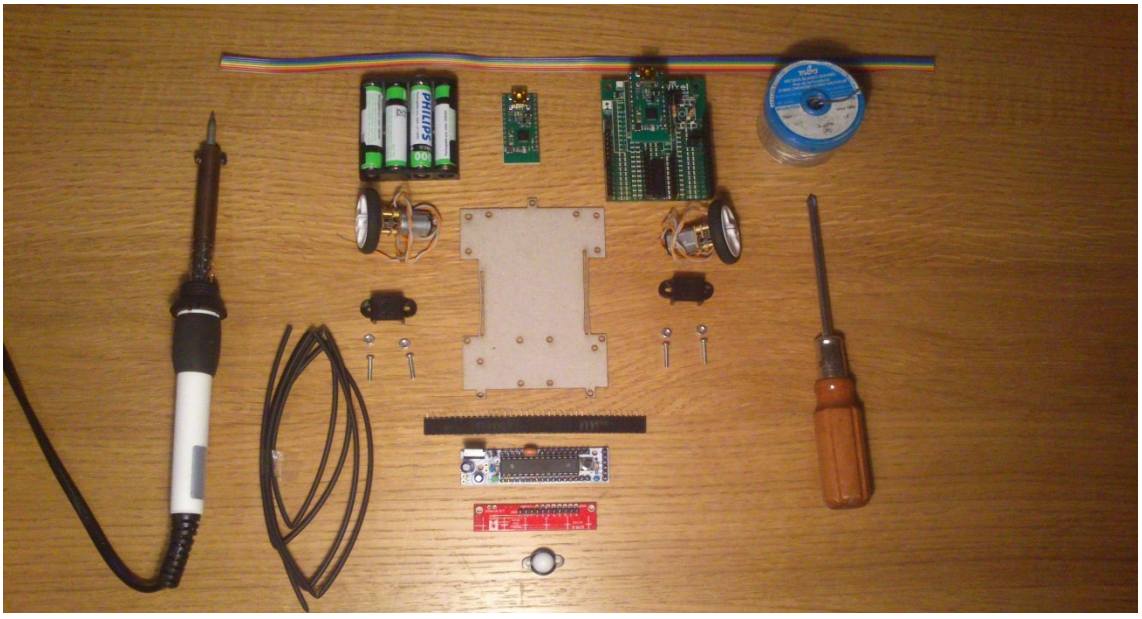



Εικ. 41 Ο αισθητήρας QTR πάνω από μία διασταύρωση

Κάπου εδώ τελειώνει και ο ρόλος που παίζει ο αισθητήρας και ξαναρχίζει πάλι όταν μπει στη διαδικασία να ακολουθήσει το καινούργιο μονοπάτι. Αμέσως επόμενο είναι να δούμε πως κωδικοποιείται η πληροφορία στο arduino και στην processing και πως μας βοηθάει αυτό στη πρόβλημα της χαρτογράφησης και του σχεδιασμού.

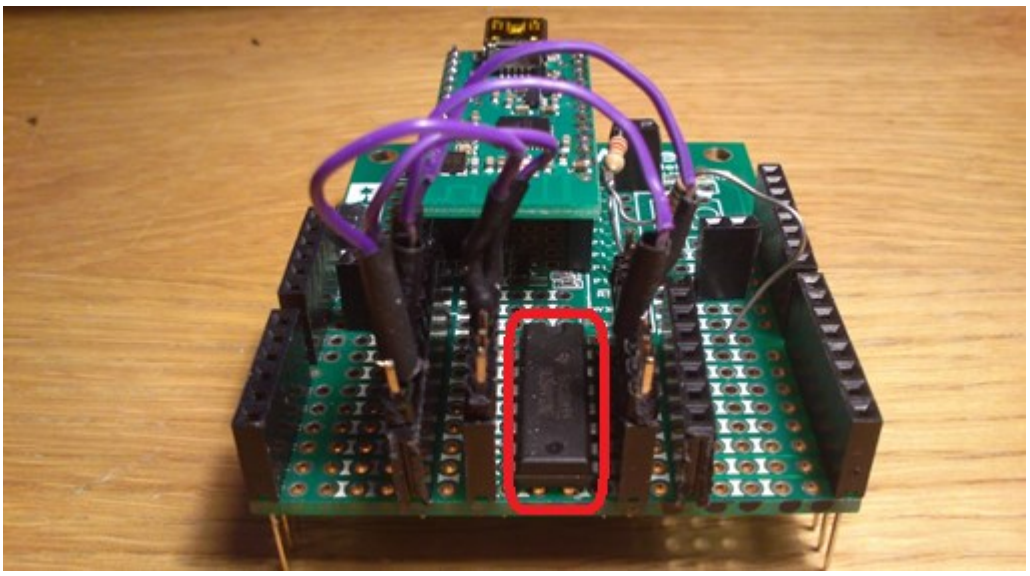
## 4.4 Η κατασκευή του ρομποτικού οχήματος

Στο κεφάλαιο 2 αναφέραμε τα βασικά ηλεκτρονικά και μηχανικά μέρη από τα οποία κατασκευάστηκε το ρομποτικό όχημα. Η διαδικασία της συνδεσμολογίας ήταν ένα από τα πολύ σημαντικά στοιχεία της εργασίας μας, διότι υπήρχαν πολλοί παράγοντες που έπρεπε να πάρουμε υπόψιν. Σίγουρα, ο πιο βασικός ήταν η ασφάλεια όλων των κυκλωματικών στοιχείων για αυτό και κάθε καινούργια σύνδεση γινόταν με ιδιαίτερη προσοχή. Επίσης πολύ σημαντική παράμετρος ήταν ο χώρος που είχαμε στη διάθεση μας. Όπως αναφέραμε χρησιμοποιήσαμε την πλακέτα Wixel shield σαν το βασικό μας χώρο συνδεσμολογίας, εφόσον αυτή παρέχει μια τέτοια δυνατότητα. Οι πιο πολλές συνδέσεις γίνονται στο όπως είναι λογικό στο arduino και στον motor controller μιας και ο τελευταίος έχει 16 εισόδους/εξόδους. Στην αρχή είχαμε επιλέξει τη μέθοδο της ηλεκτροκόλλησης για τη δημιουργία συνδέσεων αλλά αποδείχτηκε ασύμφορη και χρονοβόρα διαδικασία, οπότε στο τελικό αποτέλεσμα χρησιμοποιούμε καλώδια με θηλυκούς και αρσενικούς ακροδέκτες κατάλληλα για συνδεσμολογίες. Η βάση του οχήματος ξεκίνησε από σκληρό χαρτόνι και κατέληξε σε πλεξιγκλάς. Το σχέδιο έγινε με τη χρήση του adobe illustrator και η κοπή έγινε σε μηχάνημα laser cutter. Στην εικόνα 42 παρουσιάζονται όλα τα εργαλεία και τα ηλεκτρονικά στοιχεία που χρησιμοποιήσαμε, με εξαίρεση τον ανυψωτή τάσης DC που προσθέσαμε αργότερα.



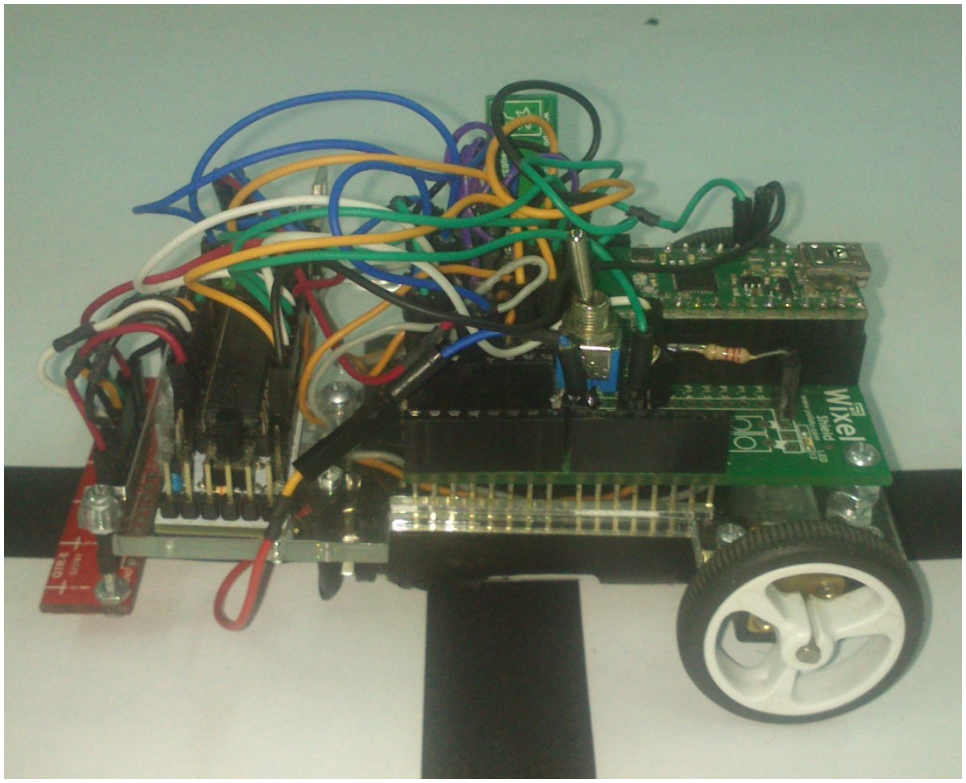

Εικ. 42 Το ρομποτικό όχημα πριν τη συναρμολόγηση

Στην εικόνα 43, φαίνεται από κοντά η πλακέτα wixel shield που χρησιμοποιούμε, στην οποία έχουμε τοποθετήσει το ένα wixel στο πίσω μέρος της και στο πλαίσιο που σημειώνεται με κόκκινο, βρίσκεται ο motor controller. Στην πλακέτα φαίνονται κάποιες από τις γραμμές κοινής τάσης οι οποίες χωρίζονται σε οριζόντιες, με τρεις εισόδους/εξόδους και σε κάθετες, με οκτώ εισόδους/εξόδους. Οι γραμμές που είναι μεγαλύτερες σε μήκος εξυπηρετούν να χρησιμοποιηθούν σαν κοινή γείωση ή κοινή τάση εισόδου. Στη συγκεκριμένη εικόνα, βλέπουμε να καταλήγουν τέσσερα καλώδια γείωσης από τον motor controller στη γραμμή γείωσης.



Εικ. 43 Η πλακέτα wixel shield με τον motor controller

Στην εικόνα 44 φαίνεται το ρομποτικό όχημα όπως είναι στο τελικό του στάδιο και είναι τοποθετημένο πάνω από μία γραμμή του λαβυρίνθου. Δυστυχώς η παρουσία τόσων καλωδίων δυσκολεύει την παρατήρηση των συνδέσεων για αυτό και τις παρουσιάζουμε καλύτερα στο κυκλωματικό διάγραμμα στην ενότητα 4.5.




Εικ. 44 Το ρομποτικό όχημα στο τελικό στάδιο της κατασκευής του

## 4.5 Το κυκλωματικό διάγραμμα του ρομποτικού οχήματος

Πριν περάσουμε στις υπόλοιπες ενότητες στις οποίες περιγράφουμε το σκεπτικό και τη λύση για τα προβλήματα της εργασίας μας, παρουσιάζουμε το κυκλωματικό διάγραμμα που δείχνει όλες τις συνδέσεις των ηλεκτρονικών και μηχανικών μερών του οχήματος.

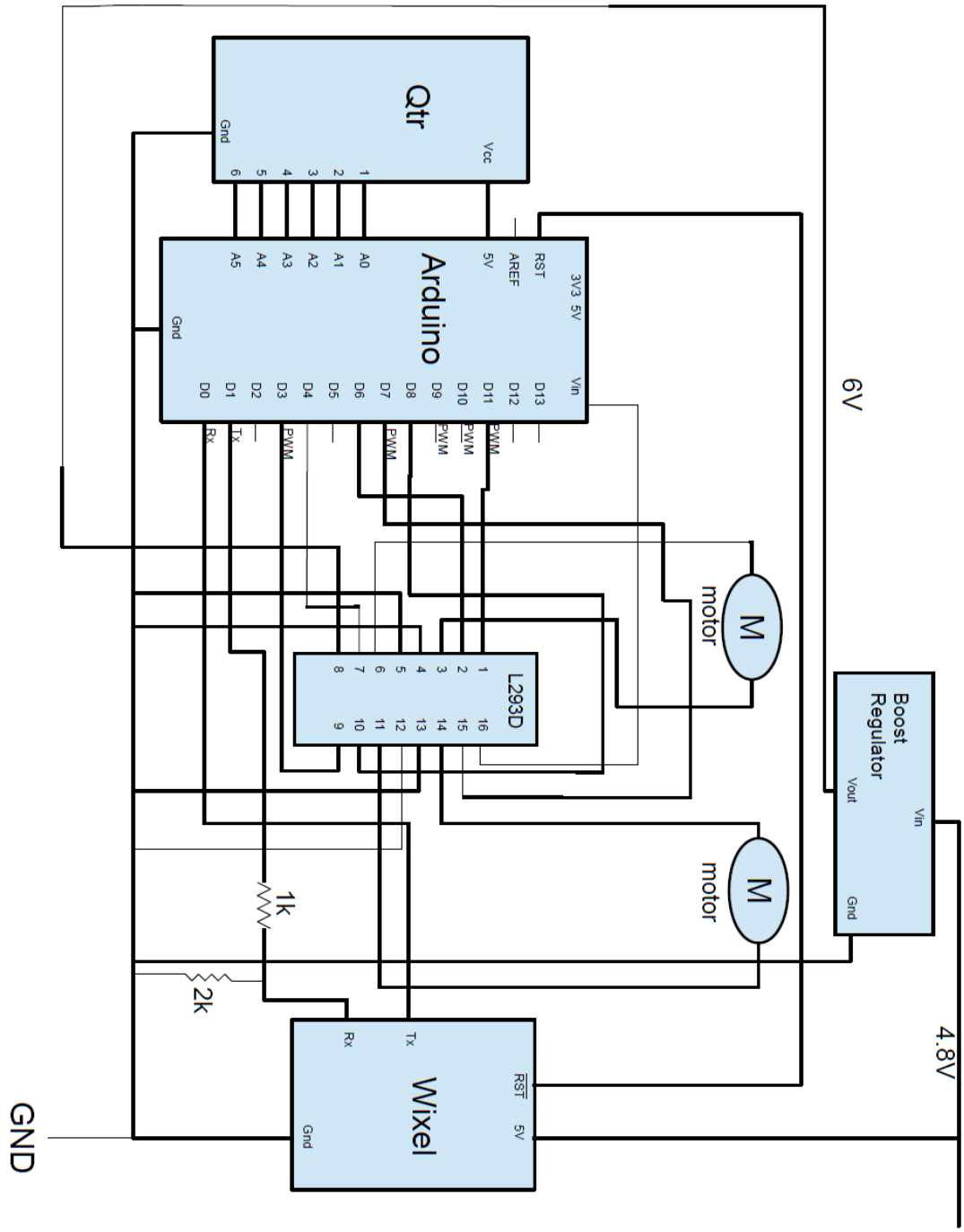
Στο παρακάτω διάγραμμα η τάση 4.8 V είναι η τάση τροφοδοσίας του κυκλώματος που παίρνουμε από τις μπαταρίες και συνδέονται σαν τάσεις εισόδου μόνο για το κύκλωμα του Wixel και για τον μετατροπέα ανύψωσης τάσης (boost converter).

Ο boost converter με τη σειρά του τροφοδοτεί με τάση 6.5 V από την έξοδο του Vout την πλακέτα του arduino και τα δύο motors, μέσω των αντίστοιχων εισόδων Vin του arduino και του motor_controller η οποία βρίσκεται στο Pin 8 του τελευταίου. Το

μέγιστο ρεύμα εξόδου 2A αρκεί για να καλύψει τις ανάγκες των στοιχείων που τροφοδοτεί. Το arduino με τη σειρά του, τροφοδοτεί τον αισθητήρα Qtr, με τάση 5V διαμέσου της αντίστοιχης εξόδου του και της εισόδου Vcc του αισθητήρα, όπως επίσης και τον motor_controller διαμέσου της δεύτερης εξόδου 5V που διαθέτει και μέσω της εισόδου Vcc του controller που βρίσκεται στο Pin 16.

Για τον motor_controller η συνδεσμολογία όπως δείξαμε και στο σχήμα 16 είναι η εξής: Τα pins 4,5,12 και 13 συνδέονται στη γείωση. Τα pins 2,7,10,15 αφορούν τη λογική κατάσταση (HIGH,LOW) στην οποία θα βρίσκονται οι αντίστοιχοι ακροδέκτες του κάθε motor, δηλαδή αν σε αυτούς θα εφαρμοστεί τάση ή όχι και ελέγχονται από τους ακροδέκτες 6,4,8 και 7 του arduino αντίστοιχα. Στα pins 3,6 συνδέουμε τους ακροδέκτες του ενός δεξιού motor και στα pins 14 και 11 συνδέουμε τους ακροδέκτες του αριστερού.Τα pins 1 και 9 ενεργοποιούν το δεξί και το αριστερό motor αντίστοιχα και συνδέονται το πρώτο με το pin 11 και το δεύτερο με το pin 3 του arduino. Σημειώνουμε ότι τα pins 3 και 11 του arduino τα χρησιμοποιούμε για τον έλεγχο της ταχύτητας των motors σύμφωνα με το μηχανισμό της Διαμόρφωσης Πλάτους Παλμού που έχουμε περιγράψει.

Για τον Qtr κάθε ένας από τους αισθητήρες (έξοδοι 1-6) συνδέονται στα αναλογικά pins του arduino Analog(0-5) αντίστοιχα. Για τη συνδεσμολογία μεταξύ του Wixel και του arduino υλοποιήσαμε τον διαιρέτη τάσης μεταξύ του arduino-Tx και του Wixel-Rx που δείξαμε και στο σχήμα 18 ενώ το arduino-Tx το συνδέσαμε με το Wixel-Rx. Τέλος, σημειώνουμε ότι χρησιμοποιούμε γείωση κοινής αναφοράς για όλα τα μέρη του κυκλώματος.




Εικ. 45 Το κυκλωματικό διάγραμμα του ρομποτικού οχήματος

## 4.6 Η πληροφορία μεταξύ arduino και processing

Ένα από τα πιο σημαντικά σημεία της εργασίας είναι η ανταλλαγή μηνυμάτων μεταξύ του arduino και της processing αφού όπως είπαμε η όλη διαδικασία του σχεδιασμού και της χαρτογράφησης γίνεται με συνεργατικό τρόπο μεταξύ των δύο. Από την πλευρά του arduino, τι μήνυμα στέλνει στην processing, διακρίνουμε δύο περιπτώσεις. Η πρώτη είναι η επιβεβαίωση ότι λήφθηκε ένα μήνυμα και η δεύτερη είναι η πληροφορία που αφορά το χαρακτήρα μιας διασταύρωσης. Η πρώτη περίπτωση αφορά πιο πολύ τον μηχανισμό ανταλλαγής μηνυμάτων και δεν μας απασχολεί προς το παρόν. Σε αυτή την ενότητα θα επικεντρωθούμε κυρίως στο πως κωδικοποιούμε την πληροφορία όταν βρίσκουμε μια διακλάδωση και τι σημαίνει αυτό, για το πρόγραμμα στην processing. Από την πλευρά του υπολογιστή, δηλαδή της processing το μήνυμα που στέλνει κάθε φορά είναι για το που να στρίψει και ένας έλεγχος για να δει σε τι κατάσταση βρίσκεται το ρομπότ.

Αναφέραμε προηγουμένως ότι όταν βρίσκουμε μια διασταύρωση ο πίνακας των τιμών του αισθητήρα έχει τιμές σαν [1000 1000 1000 1000 0 0 ] ή [1000 1000 1000 1000 1000 1000]. Για παράδειγμα αυτές οι τιμές είναι για μια διασταύρωση που έχει στροφή αριστερά και για μια, που έχει στροφή και αριστερά και δεξιά. Βέβαια δεν μας αρκεί αυτό γιατί όπως είπαμε, πρέπει το όχημα να κάνει και ένα μικρό βήμα προς τα εμπρός, όπου αν υπάρχει μονοπάτι τότε μόνο οι δύο μεσαίοι αισθητήρες θα είχαν τιμές κοντά στο 1000 και οι υπόλοιποι θα είχαν στο 0. Φτιάξαμε λοιπόν μια κωδικοποίηση που αντιστοιχεί σε αυτές και στις υπόλοιπες περιπτώσεις η οποία βασίζεται στο να στέλνει 1 ή 0, έτσι που το 1 να αντιστοιχεί στο ότι είδαμε μια γραμμή και το 0 στο ότι δεν είδαμε. Βέβαια δεν το κάνουμε αυτό για κάθε αισθητήρα, αλλά μόνο για τους δύο ακριανούς και μία τιμή για τους δύο μεσαίους, μιας και αυτοί είτε θα βλέπουν μαζί μία γραμμή, είτε όχι. Η κωδικοποίηση αυτή παίρνει υπόψιν πρώτα το τι βλέπουμε δεξιά, μετά αριστερά και μετά ευθεία. Στέλνουμε άλλη μια τιμή η οποία στην προκειμένη περίπτωση δεν παίζει κάποιο ρόλο απλά επειδή μας βολεύει να στέλνουμε τέσσερις τιμές (θα εξηγήσουμε παρακάτω), έχουμε προσθέσει και αυτή. Υποτίθεται ότι αντιστοιχεί στο τι βλέπουμε προς τα πίσω, αλλά αυτό δεν έχει κάποιο νόημα, οπότε την θέτουμε πάντα στο 0. Έτσι λοιπόν ο κανόνας right left front back ορίζει ότι αν μετά τις κινήσεις που κάνουμε, βρούμε μία διασταύρωση που έχει για παράδειγμα στροφή δεξιά και ευθεία, τότε θα στείλουμε {1 0 1 0}. Σε μια διαφορετική περίπτωση που θα είχε μόνο στροφή αριστερά, θα στέλναμε {0 1 0 0}. Είναι προφανές ότι στην περίπτωση του αδιεξόδου θα στέλναμε {0 0 0 0 }. Στην ουσία αυτός ο τρόπος δηλώνει απλά τι βλέπουν οι αισθητήρες σε σχέση με τη θέση του στο χώρο και είναι προφανές, ότι την ίδια διασταύρωση αν την δούμε από διαφορετικούς προορισμούς θα έχει και διαφορετικά αποτελέσματα.

Από μόνος του αυτός ο τρόπος δεν μας λύνει τα χέρια, γιατί στο περιβάλλον της processing εκεί που κρατάμε και τη δομή του γράφου που σχηματίζεται σιγά σιγά, δεν παίρνουμε καμία πληροφορία από το arduino για το ποια είναι αυτή η διασταύρωση που είδαμε. Το θέμα αυτό θα το εξετάσουμε αμέσως παρακάτω όπου αναλύεται ο τρόπος με τον

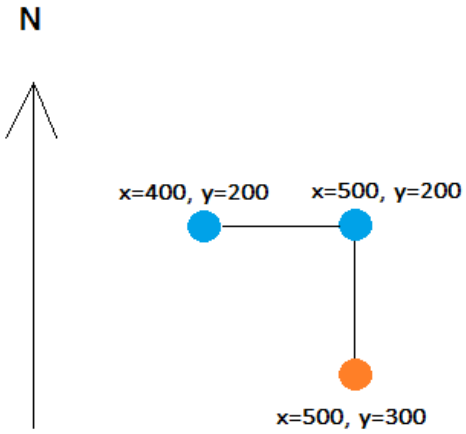
οποίο δημιουργείται το γράφημα του λαβυρίνθου και το πώς αξιοποιούμε την πληροφορία που στάλθηκε.

Για να κλείσουμε την ενότητα αυτή, απλά αναφέρουμε και την πληροφορία που στέλνει η processing στο arduino. Όπως είπαμε έχει σχέση αποκλειστικά με το ποια κίνηση πρόκειται να εκτελέσει το ρομπότ. Στέλνει δηλαδή τιμές από το 7 έως το 10, κάθε μία από τις οποίες αντιστοιχεί σε μία από τις τέσσερις κινήσεις. Το ότι στέλνουμε αριθμούς και όχι για παράδειγμα μια συμβολοσειρά έχει σχέση με τον τρόπο που γίνεται η επικοινωνία μεταξύ arduino και processing. Στην ουσία όμως γίνεται, επειδή ένας αριθμός σε  bit είναι πιο μικρός από μια συμβολοσειρά και επιπλέον ο αριθμός δεν χρειάζεται περαιτέρω επεξεργασία. Το πιο σημαντικό ίσως είναι ότι, πριν στείλει η processing έναν αριθμό που αντιστοιχεί σε κάποια κίνηση, πρώτα κοιτάει να δει αν είναι το ρομπότ έτοιμο να λάβει μια τέτοια εντολή. Και αυτό όμως άπτεται πιο πολύ στην υλοποίηση της επικοινωνίας και όχι στο περιεχόμενο της οπότε θα το εξηγήσουμε παρακάτω και αυτό.

## 4.7 Το πρόβλημα της χαρτογράφησης (mapping)

Βασικό στοιχείο στο πρόβλημα της χαρτογράφησης είναι να μπορούμε να έχουμε μια αντίληψη του χώρου έτσι ώστε κάθε φορά να ξέρουμε που βρισκόμαστε και να έχουμε και γνώση του προσανατολισμού μας. Έτσι λοιπόν, όπως όλοι οι χάρτες έχουν σημειωμένα τα τέσσερα σημεία του ορίζοντα, κάναμε και εμείς το ίδιο. Θέτουμε σαν αρχή ότι όλες οι κινήσεις θα γίνονται βόρεια, νότια, δυτικά ή ανατολικά. Αυτό αντιστοιχεί και στην τοπολογία του λαβυρίνθου, αφού αυτός δεν επιτρέπει κινήσεις σε ενδιάμεσους προσανατολισμούς. Για κάθε μία από αυτές τις κινήσεις, φτιάξαμε μια συνάρτηση και την ονομάσαμε north, west , east και south. Οι συναρτήσεις αυτές είναι υπεύθυνες για την αποτύπωση της κίνησης του ρομπότ, στο γραφικό περιβάλλον και μέσα σε αυτές γίνεται και η δημιουργία κάποιου καινούργιου κόμβου, αν χρειάζεται. Το σύστημα αναφοράς μας είναι η οθόνη του υπολογιστή έτσι ώστε όταν βλέπουμε να ζωγραφίζεται ο λαβύρινθος, η κίνηση προς τα πάνω, είναι η εκτέλεση της εντολής north, αριστερά της west και πάει λέγοντας. Έχουμε πάρει σαν σύμβαση, ότι πάντα, η πρώτη κίνηση που κάνει το όχημα μας θα είναι μια κίνηση προς τα πάνω, ανεξάρτητα προφανώς από το πώς εμείς την βλέπουμε στο λαβύρινθο. Αυτό είναι μια μορφή αρχικοποίησης του συστήματος αναφοράς και έτσι έχουμε μια βάση για να γνωρίζουμε και τον προσανατολισμό του ρομπότ κάθε στιγμή.

Τώρα που ορίσαμε το σύστημα αναφοράς μπορούμε να ορίσουμε και το χώρο κινήσεων, αυτή τη φορά με υπολογιστικούς όρους. Επιλέξαμε την αναπαράσταση του λαβυρίνθου με γράφο. Αυτό σημαίνει ότι το περιβάλλον μας είναι ένας διακριτοποιημένος χώρος (*grid world*), όπου η μετάβαση γίνεται από κελί σε κελί ή από κόμβο σε κόμβο και δεν έχουμε τρόπο να αντιστοιχήσουμε την επακριβή θέση του ρομπότ όταν αυτό κινείται ανάμεσα από δύο σημεία. Μπορούμε μόνο να ξέρουμε ότι πήγε σε ένα σημείο ή ότι είναι στη διαδικασία να πηγαίνει. Στο συγκεκριμένο παράδειγμα λαβυρίνθου που παρουσιάζουμε έχουμε έναν χώρο με 8 × 7 πιθανά διακριτά σημεία. Τώρα, για να μπορούμε να αναγνωρίσουμε ότι ένας κόμβος του γράφου είναι διαφορετικός από έναν άλλον και επίσης για να έχουμε και πληροφορία για το που είναι αυτός ο κόμβος, φτιάξαμε και ένα σύστημα με συντεταγμένες. Στην πραγματικότητα εκμεταλλευτήκαμε το περιβάλλον της processing, που είναι κατασκευασμένο έτσι ώστε κάθε γραφιστικό πρόγραμμα να τρέχει σε ένα χώρο με συντεταγμένες. Το πόσο μεγάλος θα είναι και πως θα τον χειριζόμαστε, αφήνεται στην ευχέρεια μας. Εμείς θέσαμε ότι το γραφιστικό περιβάλλον θα έχει διαστάσεις 800×700 pixels. Έτσι λοιπόν ορίσαμε ότι ο πρώτος κόμβος θα ξεκινάει από κάποιες συγκεκριμένες συντεταγμένες. Αυτό προϋποθέτει μια μερική γνώση του λαβυρίνθου ως προς το που τοποθετείται το αρχικό σημείο εισόδου, έτσι ώστε να υπάρχει και μια ομοιόμορφη αποτύπωση του λαβυρίνθου στην οθόνη. Αν για παράδειγμα ξεκινάμε την αναζήτηση από κάτω και αριστερά, τότε θέτουμε και σαν αρχικές συντεταγμένες αυτές που αντιστοιχούν σε ένα σημείο κάτω και αριστερά. Προφανώς δεν είναι καθόλου υποχρεωτικό απλά γίνεται για λόγους ομοιόμορφης αποτύπωσης όπως αναφέραμε. Έτσι λοιπόν αφού έχουμε θέσει τις συντεταγμένες του πρώτου κόμβου, τον εισάγουμε στη λίστα γειτνίασης και όταν ολοκληρωθεί η north, η οποία ζωγραφίζει ένα μονοπάτι μήκους 100 pixels, όταν αυτή τελειώσει, ξέρουμε ότι ο νέος κόμβος, αν βρεθεί νέος, θα έχει new_x=start_x και new_y=start_y-100 (η processing χρησιμοποιεί ανάποδο σύστημα συντεταγμένων, έτσι που η πάνω αριστερή γωνία έχει x=0 και y=0). Οπότε κάθε κόμβος που θα δημιουργείται θα έχει συντεταγμένες και προσανατολισμό που θα υπολογίζεται σε σχέση με τον προηγούμενό του κόμβο. Έτσι λοιπόν σταδιακά δημιουργείται ο γράφος, όπου κάθε κόμβος θα έχει ένα διαφορετικό ζευγάρι συντεταγμένων.



Εικ. 46 Οι συντεταγμένες των τριών πρώτων κόμβων με την ακολουθία εντολών north → west

Στην εικόνα που ακολουθεί φαίνεται ένα παράδειγμα για τους τρείς πρώτους κόμβους αν ακολουθηθεί μια σειρά εντολών σαν την north → west, με τον αρχικό κόμβο να έχει συντεταγμένες (500,300).

Στην ουσία, μέχρις στιγμής έχουμε περιγράψει τη διαδικασία με την οποία, κάθε κόμβος αποκτά μία μοναδική τιμή συντεταγμένων αλλά ακόμα δεν έχει ξεκαθαριστεί πως γίνεται

γνωστή η παραπάνω πληροφορία για τον κόμβο. Δηλαδή το πώς εκμεταλλευόμαστε την πληροφορία που μας δίνουν οι αισθητήρες για την κάθε διασταύρωση που συναντάμε και πως αποφασίζεται τελικά το αν πρέπει να εκτελεστεί μία στροφή ή όχι. Αυτό μας εισάγει στην ενότητα της διαδικασίας σχεδιασμού των απαραίτητων κινήσεων που πρέπει να κάνουμε για να λύσουμε το πρόβλημα που έχουμε θέσει, την οποία παρουσιάζουμε παρακάτω.

## 4.8 Η διαδικασία σχεδιασμού των διαδρομών (planning)

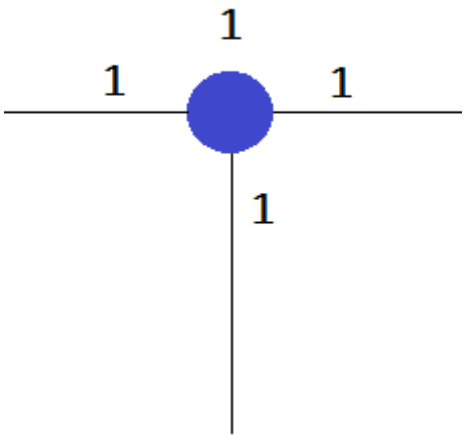
Σε αυτό το σημείο, έχουμε περιγράψει μέχρι στιγμής τη διαδικασία των κινήσεων του ρομπότ που εκτελείται στο πρόγραμμα του arduino, τον τρόπο με τον οποίο διαβάζουμε την πληροφορία μέσα από τους αισθητήρες και πως την κωδικοποιούμε, συνεχίσαμε με την περιγραφή του χώρου κίνησης αλγοριθμικά με την αναπαράσταση σε γράφο και τη διαδικασία με την οποία κάνουμε τη χαρτογράφηση και τέλος μας μένει η ενότητα όπου αναλύεται ο τρόπος σχεδιασμού των κινήσεων που κάνουμε για να λύσουμε το πρόβλημα που κάθε φορά αντιμετωπίζουμε. Δηλαδή το κομμάτι αυτό που αποφασίζει πως λύνεται ο λαβύρινθος και πως αντιμετωπίζεται το πρόβλημα εύρεσης των συντομότερων διαδρομών. Αν το δούμε αφαιρετικά αυτό το κομμάτι της εργασίας είναι το υψηλότερο σε επίπεδο αφού χρειάζεται όλα τα προηγούμενα να δουλεύουν για να μπορούμε να πάρουμε τις σωστές αποφάσεις.

Από τη στιγμή που δείξαμε πως ένας κόμβος αποκτά μια τιμή συντεταγμένων που τον ορίζει στον χώρο, δείξαμε ένα μόνο από τα χαρακτηριστικά του. Από εκεί και πέρα υπάρχουν και άλλα πολύ βασικά χαρακτηριστικά που έχουν κυρίως σχέση με τους δεσμούς που αποκτά ένας κόμβος με άλλους γειτονικούς του όπως επίσης και το χαρακτηρισμό σαν “ανοικτό” ή “κλειστό” το οποίο μας χρησιμεύει για το πρόβλημα εύρεσης συντομότερων μονοπατιών. Για την περιγραφή και ανάλυση των κόμβων δημιουργήσαμε μια κλάση στην processing τη node η οποία έχει μεταβλητές και συναρτήσεις οι οποίες μας βοηθούν για τα παραπάνω προβλήματα. Τα βασικά χαρακτηριστικά που έχει ένας κόμβος είναι μεταβλητές που μας ενημερώνουν για τα ποια μονοπάτια έχει κλειστά ή ανοικτά, τέσσερις μεταβλητές της ίδιας κλάσης για να συνδέονται όλοι οι κόμβοι στη λίστα γειτνίασης, μια μεταβλητή που κρατάει το πόσες φορές αυτός ο κόμβος έχει προσπελαστεί και τις συναρτήσεις που επενεργούν πάνω σε αυτές τις μεταβλητές.

## 4.9 Δημιουργία και επεξεργασία κόμβου

Η πρώτη μας ενέργεια όσον αφορά έναν νέο κόμβο είναι η δημιουργία του. Ο πρώτος κόμβος, δημιουργείται όπως αναφέραμε από την αρχή του προγράμματος και αποθηκεύεται στη λίστα γειτνίασης σαν το πρώτο στοιχείο. Από εκεί πέρα όμως, κάθε νέος κόμβος δημιουργείται υπό προϋποθέσεις και στο βαθμό που γνωρίζουμε κάποιον γειτονικό του, αμέσως τους συνδέουμε. Όπως είναι λογικό κάποιος κόμβος δημιουργείται όταν φτάνουμε σε μια διασταύρωση, αλλά αυτό από μόνο του δεν αρκεί, γιατί υπάρχει πιθανότητα να ξαναπεράσουμε από την ίδια διασταύρωση οπότε η δημιουργία ξανά του ίδιου κόμβου θα ήταν λάθος. Παρουσιάζουμε ένα σενάριο με το παράδειγμα του πρώτου κόμβου που δημιουργείται αμέσως μετά τον αρχικό μας, για να γίνει πιο κατανοητό. Όπως είπαμε η πρώτη διαδρομή που σχεδιάζεται είναι ένα μονοπάτι αποτέλεσμα της *north*. Οπότε αν έχουμε βρεθεί σε διασταύρωση λαμβάνουμε ένα μήνυμα για παράδειγμα το {1 1 0 0}. Αυτό σημαίνει σύμφωνα με τον κανόνα *right-left-front-back,* σημαίνει ότι είμαστε σε διασταύρωση με μονοπάτι δεξιά και αριστερά. Χωρίς να μας ενδιαφέρει ακόμα το γιατί, ο αλγόριθμος απόφασης λέει ότι πρέπει να γίνει στροφή αριστερά. Η εντολή εκτελείται και το σενάριο λέει ότι φτάνουμε σε αδιέξοδο, οπότε τώρα, πρέπει να γυρίσουμε πίσω με αποτέλεσμα να ξαναφτάσουμε στον ίδιο κόμβο με πριν. Τώρα όμως αυτό που βλέπουν οι αισθητήρες θα είναι μια διαφορετική εικόνα, με στροφή δεξιά και ευθεία, οπότε το μήνυμα που θα λάβουμε θα είναι το {1 0 1 0}. Ο τρόπος για να καταλάβουμε αν είμαστε στον ίδιο κόμβο δεν είναι περίπλοκος. Απλά κάνουμε μια αναζήτηση στη λίστα γειτνίασης και ψάχνουμε να βρούμε αν υπάρχει άλλος κόμβος με κλειδί τις τρέχουσες συντεταγμένες. Αυτό όμως που μένει αναπάντητο είναι το ποια χαρακτηριστικά του κόμβου θα εκμεταλλευτούμε, ώστε να αποφύγουμε να στρίψουμε δεξιά και να γυρίσουμε πάλι στο μονοπάτι από το οποίο ξεκινήσαμε. Έτσι λοιπόν, χρησιμοποιούμε μια βοηθητική μεταβλητή τη node_state η οποία περιγράφει αν τα μονοπάτια γύρω από τον κόμβο είναι ανοικτά ή κλειστά. Ανοικτό μονοπάτι σημαίνει ότι το μονοπάτι δεν έχει προσπελαστεί και κλειστό σημαίνει είτε ότι δεν υπάρχει καθόλου μονοπάτι, είτε ότι έχει προσπελαστεί τουλάχιστον μία φορά. Η node_state είναι στην πραγματικότητα τέσσερις μεταβλητές και τη χρησιμοποιούμε λίγο αφαιρετικά στο κείμενο. Οι πραγματικές είναι η north_state, η west_state, η east_state και η south_state. Οι τιμές που μπορούν να πάρουν είναι 0 και 1, με το 0 να σημαίνει κλειστό μονοπάτι και το 1 να σημαίνει ανοικτό. Όταν λοιπόν δημιουργείται ένας κόμβος αρχικοποιούνται και αυτές οι μεταβλητές με την τιμή 1. Θεωρούμε αυθαίρετα λοιπόν ότι στην αρχή ο κόμβος είναι ανοικτός προς όλες τις κατευθύνσεις, αν και σίγουρα δεν είναι από την κατεύθυνση από την οποία ήρθαμε, αφού μόλις τη διασχίσαμε.

Στην εικόνα που ακολουθεί, δείχνουμε τις τιμές που έχει η node_state τη στιγμή που δημιουργείται ένας νέος κόμβος, πριν ακόμα πάρουμε την πληροφορία από τους αισθητήρες. Είπαμε ότι για το συγκεκριμένο σταυροδρόμι το μήνυμα που θα λάβουμε είναι το {1 1 0 0}. Βλέπουμε λοιπόν ότι η μορφή του μηνύματος με τη μορφή των μεταβλητών που ορίζουν την κατάσταση του κόμβου είναι αρκετά όμοιες. Τέσσερα ψηφία και στις δύο περιπτώσεις με τιμές 0 ή 1. Αν τώρα εφαρμόσουμε την λογική πράξη AND μεταξύ αυτών των τιμών, έτσι ώστε να πάρουμε κατά ζεύγη την μεταβλητή east_state με την τιμή right, την μεταβλητή north_state με την τιμή της up, την west_state με την left και την south_state με την back και αποθηκεύσουμε τα αποτελέσματα στις αντίστοιχες node μεταβλητές θα πάρουμε σαν αποτέλεσμα το εξής:



Εικ. 47 Ένας κόμβος στο στάδιο αρχικοποίησης του

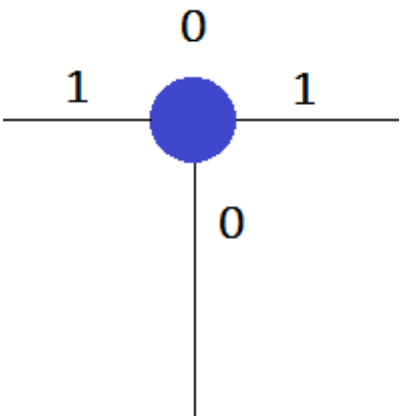
east_state=**1** AND 1 = 1

west_state**=1** AND 1 = 1

north_state=**1** AND 0 = 0

south_state=**1** AND 0 = 0

Έτσι λοιπόν τώρα, όπως φαίνεται και στο σχήμα που ακολουθεί, τα μονοπάτια που όντως είναι ανοικτά έχουν τιμή 1 και αυτά που έχουν προσπελαστεί ή δεν υπάρχουν, έχουν τιμή 0 (το νότιο και το βόρειο αντίστοιχα). Οι νέες τιμές των μεταβλητών node_state δείχνουν τώρα την πραγματική κατάσταση του κόμβου η οποία υπολογίζεται μόνο αφού διαβαστούν οι τιμές από τους αισθητήρες.



Εικ. 48 Ένας κόμβος αφού πάρει σωστές τιμές όταν φτάνουμε σε αυτόν από τη νότια πλευρά

Τώρα θα δούμε τι γίνεται, όταν αφού επιλέξει ο αλγόριθμος απόφασης την στροφή αριστερά και επιστρέψουμε στον ίδιο κόμβο μετά το αδιέξοδο. Όπως είπαμε το μήνυμα που θα λάβουμε αυτή τη φορά θα είναι το {1 0 1 0}. Δεν μας απασχολεί να ξαναφτιάξουμε νέο

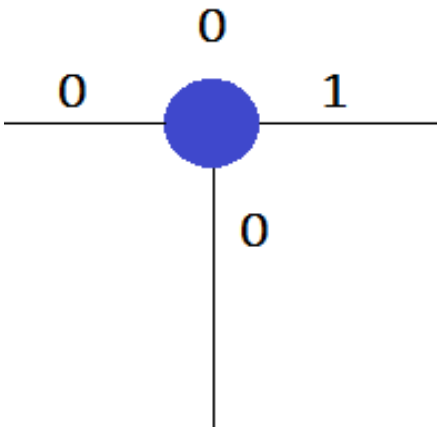
κόμβο, αφού όπως είπαμε με μια αναζήτηση στον πίνακα γειτνίασης, βλέπουμε ότι ήδη υπάρχει. Θα μας απασχολήσει μόνο το κομμάτι του να θέσουμε στον κόμβο τις σωστές τιμές κατάστασής του. Αυτό για το συγκεκριμένο παράδειγμα σημαίνει ότι εφόσον κινούμαστε ανατολικά μετά το αδιέξοδο, θα συναντήσουμε τη δυτική πλευρά του κόμβου. Θέλει όμως προσοχή εδώ, γιατί προηγουμένως δεν αναφέραμε ότι η επιλογή των ζευγαριών που κάναμε την πράξη AND ήταν λίγο αυθαίρετη και πέτυχε επειδή ταυτιζόταν ο προσανατολισμός της κίνησης του ρομπότ με το σύστημα αναφοράς που χρησιμοποιούμε. Δηλαδή αυτό που έβλεπε στα δεξιά του ο αισθητήρας, ταυτιζόταν με την ανατολική πλευρά του κόμβου. Και η ανατολική πλευρά του κόμβου δεν είναι σχετική έννοια, όπως είναι αυτό που βλέπει ο αισθητήρας. Στη συγκεκριμένη περίπτωση αυτό που βλέπει στα δεξιά του ο αισθητήρας είναι η νότια πλευρά του κόμβου η οποία και είναι κλειστή, αφού την διαβήκαμε με την πρώτη κίνηση που κάναμε. Άρα λοιπόν, πρέπει πρώτα να εξετάσουμε τον προσανατολισμό που έχουμε σε σχέση με το σύστημα αναφοράς ώστε να έρθουμε σε συμφωνία μαζί του. Οπότε η πράξη AND πρέπει να γίνει μεταξύ της τιμής right και της τιμής south_state, μεταξύ της τιμής front και της east_state, της left και της north_state και τέλος της back με την west_state. Είναι στην ουσία ένας μηχανισμός κυκλικής ολίσθησης που πρέπει κάθε φορά να γίνεται μεταξύ των τιμών του αισθητήρα και των τιμών που περιγράφουν την κατάσταση του κόμβου. Οι καινούργιες τιμές φαίνονται παρακάτω και στο σχήμα που ακολουθεί.

east_state=**1** AND 1 = 1

west_state**=1** AND 0 = 0

north_state=**0** AND 0 = 0

south_state=**0** AND 1 = 0




Εικ. 49 Ο ίδιος κόμβος αφού πάρει σωστές τιμές όταν φτάνουμε σε αυτόν ξανά, από τη δυτική πλευρά

Όπως φαίνεται τώρα, μόνο η ανατολική πλευρά έχει τιμή 1 και αυτό είναι σωστό, γιατί μόνο αυτή δεν έχει ακόμα προσπελαστεί. Την τεχνική αυτή την χρησιμοποιούμε και για τις υπόλοιπες περιπτώσεις και είναι πολύ σημαντική για να παίρνουμε κάθε φορά τη σωστή πληροφορία για τον κόμβο που εξετάζουμε, που όπως θα δείξουμε είναι προϋπόθεση για τη σωστή απόφαση που αφορά την επόμενη κίνηση. Σημειώνουμε ότι για τον παραπάνω κόμβο σε όλες τις περιπτώσεις που τον επεξεργαστήκαμε τον θεωρούμε ανοικτό, επειδή έχει τουλάχιστον ένα μονοπάτι που δεν έχει εξερευνηθεί. Όταν προσπελαστεί και το ανατολικό μονοπάτι τότε θα θεωρηθεί κλειστός κόμβος αφού όλες οι μεταβλητές node_state θα είναι πλέον 0. Διακρίνουμε όμως δύο διαφορετικές περιπτώσεις για την κατάσταση του κόμβου. Αυτή που μόλις περιγράψαμε αφορά μόνο τον αλγόριθμο της

επίλυσης του λαβύρινθου και της χαρτογράφησης και όχι του αλγόριθμου επίλυσης του προβλήματος των συντομότερων μονοπατιών.

## 4.10 Η διαδικασία απόφασης (planning)

Στην ενότητα αυτή παρουσιάζουμε τον τρόπο επίλυσης ενός από τα δύο βασικά θέματα που μας απασχολούν στην παρούσα εργασία. Η διαδικασία απόφασης του τρόπου με τον οποίο λύνουμε το πρόβλημα, αφορά το κομμάτι της επίλυσης του λαβυρίνθου σε συνδυασμό με την διαδικασία χαρτογράφησης. Για το κομμάτι του λαβυρίνθου αυτό είναι να μπορεί να βρεθεί ένα τερματικό σημείο το οποίο το θεωρούμε σαν την έξοδο, ξεκινώντας από ένα αρχικό σημείο το οποίο θεωρούμε σαν είσοδο (επίλυση προβλήματος με αναζήτηση [13]). Το αρχικό σημείο θεωρούμε ότι πρέπει να είναι ανοικτό μόνο προς μία κατεύθυνση, δηλαδή με όρους γράφου, να είναι κόμβος με βαθμό τη μονάδα. Αυτό για το λαβύρινθο, μπορεί μόνο να σημαίνει, ότι ξεκινάμε την αναζήτηση από ένα αδιέξοδο. Για τη διαδικασία της χαρτογράφησης ο σχεδιασμός των κινήσεων ταυτίζεται σε πολύ μεγάλο βαθμό με τη διαδικασία επίλυσης του λαβυρίνθου, αφού αυτά τα δύο γίνονται παράλληλα, με μόνη διαφορά ότι η συνθήκη τερματισμού της χαρτογράφησης είναι να εξερευνηθούν όλα τα μονοπάτια και να μην σταματήσει όταν βρεθεί το σημείο εξόδου του λαβυρίνθου. Προφανώς όπως είπαμε, θέλουμε αν μπορούμε να εξετάσουμε και έναν τρόπο να βελτιστοποιήσουμε τη διαδικασία αυτή μετά το πέρας της επίλυσης του λαβυρίνθου.

Ο λαβύρινθος που εξετάζουμε είπαμε ότι περιέχει κυκλικές διαδρομές και η λύση του βρίσκεται σε κάποιο εσωτερικό σημείο. Αυτό το ζήτημα μεγαλώνει τη δυσκολία του προβλήματος γιατί ο τρόπος με τον οποίο γίνεται η αναζήτηση δεν μπορεί να αρκεστεί μόνο στα εξωτερικά σημεία, όπως για παράδειγμα ο αλγόριθμος wall follower θα έκανε. Επίσης η ύπαρξη κύκλων πρέπει να αντιμετωπιστεί γιατί υπάρχει και ο κίνδυνος επανάληψης τέτοιων διαδρομών και ίσως, ο ατέρμονος εγκλωβισμός σε μία από αυτές.

Επίσης, μπορεί ο συγκεκριμένος λαβύρινθος που φτιάξαμε να είναι μικρός σε μέγεθος αλλά θέλουμε η αναζήτηση να γίνεται με τρόπο που θα εκμεταλλεύεται την τοπικότητα γιατί θέλουμε να μπορεί να δουλέψει και σε μεγαλύτερα περιβάλλοντα χωρίς μεγάλο κόστος. Αυτό πρακτικά σημαίνει, ο αλγόριθμος να μην αφήνει πίσω του πολλά ανοικτά μονοπάτια, που ίσως μελλοντικά χρειαστεί να ξαναπεράσει από αυτά για να τα εξερευνήσει, όπως για παράδειγμα θα έκανε ο DFS στην απλή του μορφή και

στο παράδειγμα που είχαμε παρουσιάσει στο κεφάλαιο 2.3.2. Αυτό σε έναν μεγαλύτερο λαβύρινθο θα είχε πολύ μεγάλο χρονικό κόστος και αυτό θέλουμε να το αποφύγουμε.

Παρουσιάζοντας προηγουμένως τους αλγόριθμος διάσχισης γράφου [14], προσπαθήσαμε να τονίσουμε τα δυνατά και αδύναμα σημεία, καθενός από αυτούς. Καταλήξαμε ότι αυτός που θα μας βοηθούσε καλύτερα στην επίλυση του προβλήματος μας, είναι ο αλγόριθμος του Tremaux και αυτό γιατί οι περισσότεροι από τους υπόλοιπους, είτε δεν προσέφεραν εγγυημένη λύση είτε χρειάζονταν μια προηγούμενη γνώση για το εσωτερικό του λαβυρίνθου κάτι το οποίο δεν επιδιώκαμε. Στην ουσία ο Tremaux είναι βασισμένος στον DFS και είναι τροποποιημένος έτσι ώστε να σημειώνει τα μονοπάτια που έχει ήδη διαβεί, αποφεύγοντας έτσι την αδυναμία που παρουσιάζει ο DFS στην απλή του μορφή, που δεν έχει μνήμη για τους κόμβους που προσπέλασε. Αποφεύγεται με αυτό τον τρόπο να πιαστεί ο αλγόριθμος σε έναν ατέρμονο κύκλο. Δεύτερο σημαντικό στοιχείο διαφοράς με τον DFS, πάντα συγκρίνοντας τον με την απλή του μορφή, είναι ότι δίνει τη δυνατότητα για χρήση μιας ευριστικής μεθόδου την οποία εμείς εκμεταλλευόμαστε ιδιαίτερα.

Ο κανόνας νο2 και νο4 του Tremaux, ορίζει ότι αν φτάσουμε σε ένα καινούργιο κόμβο ή σε έναν παλιό κόμβο από παλιό μονοπάτι μπορούμε να κάνουμε ελεύθερα μία κίνηση, με την προϋπόθεση βέβαια να υπάρχει κάποια δυνατότητα καινούργιας κίνησης. Εμείς εκμεταλλευτήκαμε αυτή την ελευθερία που μας δίνει ο αλγόριθμος για να υλοποιήσουμε μια **ευριστική μέθοδο** [15]. Ένα κανόνα δηλαδή που βρίσκει έναν τρόπο για βελτίωση της λύσης. Θα προσπαθήσουμε να δείξουμε γιατί αυτή η μέθοδος βελτιώνει τη λύση στο συνολικό πρόβλημα της χαρτογράφησης και της επίλυσης του λαβυρίνθου.

Η ευριστική μέθοδος για το πρόβλημα μας σημαίνει, ότι κάθε φορά επιδιώκουμε η αναζήτηση της λύσης να ξεκινάει από μια συγκεκριμένη περιοχή του λαβυρίνθου συνεχίζοντας ύστερα και στις υπόλοιπες. Τον τρόπο με τον οποίο γίνεται αυτή η επιλογή και που αποσκοπεί θα τα δείξουμε παρακάτω. Συγκεκριμένα θέσαμε έναν κανόνα που λέει ότι η αναζήτηση θέλουμε να ξεκινάει όσο πιο κάτω και αριστερά μπορούμε. Δηλαδή, αν έχουμε σαν σημείο εκκίνησης ένα το οποίο βρίσκεται σε μια περιοχή νότια και δυτικά του λαβυρίνθου, υποθέτουμε αυθαίρετα, ότι η λύση μπορεί να βρίσκεται κοντά σε εκείνη την περιοχή οπότε εξαντλούμε στο βαθμό που είναι δυνατό τα μονοπάτια που οδηγούν όσο πιο νότια μπορούμε και όσο πιο δυτικά. Δίνουμε δηλαδή προτεραιότητα σε κάποιες διαδρομές εις βάρος των υπολοίπων. Αν για παράδειγμα κινούμαστε προς μια βόρεια κατεύθυνση και φτάσουμε σε σταυροδρόμι με όλες τις επιλογές ανοικτές, επιλέγουμε πρώτα ένα μονοπάτι που

οδηγεί προς τα δυτικά εις βάρος ενός άλλου που οδηγεί βόρεια και ακόμα χειρότερα ενός που θα οδηγούσε ανατολικά. Με την προϋπόθεση πάντα, να υπάρχει δυνατότητα να το κάνουμε αυτό, το οποίο σημαίνει να υπάρχει το μονοπάτι και επίσης να είναι ανεξερεύνητο ή αν το θέσουμε διαφορετικά ο κόμβος να είναι ανοικτός από εκείνη την πλευρά. Με αυτό τον τρόπο η αναζήτηση ξεκινάει από μία περιοχή και προσπαθεί να την εξαντλεί με προτεραιότητα από τη νότια προς τη βόρεια πλευρά, και ύστερα από δυτικά προς ανατολικά. Με αυτό τον τρόπο η αναζήτηση γίνεται με έναν πιο συντεταγμένο και μαζεμένο τρόπο σε σύγκριση με την περίπτωση που αφήναμε κάθε φορά την επιλογή του μονοπατιού στην τύχη, όπως μας δίνει τη δυνατότητα ο Tremaux. Το αποτέλεσμα είναι να μην αφήνουμε πολλά “κενά” στην εξερεύνηση μας κάτι το οποίο μας βοηθάει σε αυτό που θα περιγράψουμε λίγο πιο κάτω.

Αν μπορούσαμε να παρομοιάζαμε τη μέθοδο αυτή με τη ζωγραφική θα ήταν σαν πιάναμε την κάτω αριστερή γωνία του χαρτιού και να αρχίζαμε να ζωγραφίζουμε το χαρτί από αυτό το σημείο, με διαδοχικές κινήσεις πάνω-κάτω και με πιο αργό ρυθμό από αριστερά προς τα δεξιά. Θα ζωγραφίζαμε τελικά όλο το χαρτί και όταν θα φτάναμε στην κάτω δεξιά γωνία θα είχαμε αφήσει λίγα μόνο κενά να συμπληρώσουμε.

Στον πίνακα που ακολουθεί δείχνουμε για κάθε κίνηση τι επιλέγει ο αλγόριθμος με φθίνουσα προτεραιότητα από αριστερά προς τα δεξιά. Παρατηρούμε ότι η τελευταία στήλη, έχει πάντα την αντίστροφη κίνηση από την τρέχουσα και αυτό γιατί αντίστροφη κίνηση, γίνεται μόνο στις περιπτώσεις που ο Tremaux δεν μας δίνει περιθώρια να χρησιμοποιήσουμε την ευριστική μέθοδο. Δεν την παίρνουμε υπόψιν μας στην ουσία αφού είναι μια αναγκαστική κίνηση δηλαδή δεν εμπίπτει στην κατηγορία των περιπτώσεων που έχουμε δυνατότητα να αποφασίσουμε ελεύθερα για τι θα κάνουμε. Ανήκει δηλαδή στον κανόνα νο3 και για αυτό και τη σημειώνουμε με κόκκινο χρώμα. Για όλες τις υπόλοιπες επιλογές, η προτεραιότητα έχει νόημα με την προϋπόθεση να υπάρχουν τέτοια μονοπάτια και ο κόμβος να είναι ανοικτός προς αυτή την κατεύθυνση.

| Τρέχουσα Κατεύθυνση | Επιλογή καινούργιας κατεύθυνσης | | | |
|---|---|---|---|---|
| Βόρεια | Δυτικά | Βόρεια | Ανατολικά | Νότια |
| Δυτικά | Νότια | Δυτικά | Βόρεια | Ανατολικά |
| Ανατολικά | Νότια | Βόρεια | Ανατολικά | Δυτικά |
| Νότια | Δυτικά | Νότια | Ανατολικά | Νότια |

Εικ. 50 Πίνακας που δείχνει την προτεραιότητα των κατευθύνσεων

Θα μπορούσε κάποιος να ισχυριστεί ότι η μέθοδος που χρησιμοποιούμε έχει και ένα κόστος. Αυτό είναι ότι, αν η λύση τελικά δεν βρίσκεται κοντά στην περιοχή που επιδιώκουμε να εξερευνήσουμε, τότε κάνουμε άσκοπες αναζητήσεις εφόσον αυτή βρίσκεται κάπου μακριά. Αυτό βέβαια δεν μπορούμε ποτέ να το ξέρουμε μιας και η γνώση που έχουμε για τον λαβύρινθο είναι μηδενική, οπότε το κόστος είναι φαινομενικό και όχι πραγματικό. Ένας άλλος λόγος που το κόστος που αναφέραμε είναι φαινομενικό, δικαιολογείται επειδή σκοπός της εργασίας μας δεν είναι μόνο η εύρεση ενός τερματικού σημείου στο λαβύρινθο αλλά και η χαρτογράφηση του στο σύνολο του με έναν αρκετά συντεταγμένο τρόπο. Δηλαδή να έχουμε εικόνα για όλα τα μονοπάτια, για το που βρίσκονται και το που οδηγούνε. Έτσι λοιπόν, μετά το πέρας της αναζήτησης της εξόδου συνεχίζουμε την αναζήτηση όλων των υπόλοιπων ανοικτών μονοπατιών. Δηλαδή τώρα, το τερματικό σημείο που είχαμε πριν από λίγο που ήταν ένα και μοναδικό τώρα αλλάζει και ανανεώνεται συνεχώς, αφού κάθε ανοικτό μονοπάτι είναι και στόχος προς εξερεύνηση. Οπότε στη χειρότερη περίπτωση που η έξοδος του λαβυρίνθου είναι μακριά από την περιοχή που εμείς πιστεύουμε και ο αλγόριθμος κάνει άσκοπες αναζητήσεις στην περιοχή αυτή, το κόστος που πληρώσαμε το αντισταθμίζουμε αμέσως γιατί εκείνη τη στιγμή ξεκινάει η καινούργια αναζήτηση η οποία αφορά όλα τα υπόλοιπα ανοικτά μονοπάτια. Έτσι θα έχουμε αφήσει πίσω μας μια περιοχή εξερευνημένη κατά ένα μεγάλο ποσοστό και δεν θα χρειάζεται να κάνουμε πολλά πισωγυρίσματα για να εξερευνήσουμε ανοικτά μονοπάτια. Άρα και η χαρτογράφηση θα τελειώσει πιο σύντομα.

Καταλήγουμε λοιπόν στο συμπέρασμα ότι η ευριστική μέθοδος που χρησιμοποιούμε όχι μόνο δεν επιφέρει κάποιο κόστος αλλά βελτιώνει τη λύση του συνολικού προβλήματος συνδυάζοντας την εύρεση της λύσης για τον λαβύρινθο με τον δομημένο τρόπο που ανακαλύπτει τα καινούργια μονοπάτια και άρα την επιτάχυνση της διαδικασίας της χαρτογράφησης. Συγκρίνοντας τη χειρότερη περίπτωση μεταξύ του Tremaux που δεν χρησιμοποιεί την ευριστική μέθοδο που περιγράψαμε και της δικιάς μας προσέγγισης μετρήσαμε 81 βήματα για την πρώτη περίπτωση και 69 για τη δεύτερη. Τα βήματα αυτά αφορούν τις κινήσεις που κάναμε από κόμβο σε κόμβο μέχρι να βρούμε το σημείο εξόδου προσθέτοντας τα βήματα μέχρι να εξερευνηθούν όλα τα μονοπάτια του λαβυρίνθου. Προφανώς η απλή υλοποίηση του Tremaux που αφήνει στην τύχη την επόμενη κίνηση, υπάρχει πιθανότητα να βρει πιο γρήγορα τη λύση του λαβυρίνθου αλλά σίγουρα μετά θα καθυστερήσει παραπάνω για τη διαδικασία της χαρτογράφησης.

## 4.11 Το πρόβλημα εύρεσης συντομότερων μονοπατιών-SPP

Το πρόβλημα που έχουμε θέσει από την αρχή, δεν έχει τελειώσει σε αυτό το σημείο, μιας και μέχρι στιγμής δεν έχουμε δει να καλύπτεται η συνθήκη τερματισμού που έχουμε θέσει για την διαδικασία της χαρτογράφησης. Στην προηγούμενη ενότητα εξηγήσαμε πως τροποποιήσαμε τον αλγόριθμο του Tremaux για να βρίσκει έναν καλύτερο τρόπο στη λύση του συνδυαστικού προβλήματος επίλυσης ενός λαβύρινθου και τη διαδικασία της χαρτογράφησης.

Σε αυτή την ενότητα εξετάζουμε τα βήματα που γίνονται μετά την εύρεση του τερματικού σημείου του λαβυρίνθου και κοιτάμε αν μπορούμε να βελτιστοποιήσουμε ακόμα παραπάνω τη διαδικασία της χαρτογράφησης.

Αν τα πάρουμε τα πράγματα από την αρχή, μέχρι στιγμής έχουμε δείξει τις λειτουργίες και τους μηχανισμούς με τους οποίους ψάχνουμε τη λύση σε έναν λαβύρινθο, ενώ παράλληλα κρατάμε πληροφορία για όλους τους κόμβους και τα μονοπάτια που βρίσκουμε, κάνοντας χαρτογράφηση. Ο ίδιος τρόπος με πριν θα μπορούσε πάλι να χρησιμοποιηθεί, κάνοντας δηλαδή χρήση του Tremaux μέχρι να βρεθούν όλα τα μονοπάτια και να τελειώσει η χαρτογράφηση. Αυτή η λύση σίγουρα θα δούλευε και εν μέρει την χρησιμοποιούμε αλλά είχαμε δηλώσει στην αρχή ότι θέλουμε να την βελτιστοποιήσουμε στο βαθμό που αυτό είναι δυνατό.

Όταν μπήκαμε στη διαδικασία να συγκρίνουμε τους υποψήφιους αλγόριθμους που θα χρησιμοποιήσουμε αποκλείσαμε από τη μία πλευρά, αυτούς που δεν εγγυούνταν μία σίγουρη λύση, γιατί για παράδειγμα δεν μπορούσαν να αντιμετωπίσουν τις κυκλικές διαδρομές και από την άλλη αποκλείσαμε μια πολύ ισχυρή κατηγορία αλγορίθμων οι οποίοι προϋπόθεταν τη γνώση πληροφορίας για τη δομή του λαβυρίνθου [16]. Και στην αρχή της αναζήτησης μας πράγματι δεν υπάρχει καμία γνώση για τον λαβύρινθο, παρατηρούμε όμως ότι διασχίζοντας τον σιγά σιγά, αποκτούμε πληροφορία αρκετή. Ειδικά μετά το πέρας της επίλυσης του λαβυρίνθου αν έχουμε ξεκινήσει από κάποιο μακρινό σημείο θα έχουμε αποκαλύψει ένα μεγάλο μέρος του. Οπότε τώρα διαπιστώνουμε ότι υπάρχει η δυνατότητα οι αναζητήσεις που

κάνουμε, να γίνουν με έναν καλύτερο και γρηγορότερο τρόπο, αν αξιοποιήσουμε αυτή την πληροφορία [17].

Δείξαμε ότι ο αλγόριθμος του Tremaux είναι πολύ καλός όταν ο λαβύρινθος είναι ανεξερεύνητος και χρησιμοποιεί μια πολύ αποτελεσματική μέθοδο για τον τρόπο που κάνει την εξερεύνηση. Αυτό που ίσως δεν έχει φανεί μέχρι τώρα είναι η αδυναμία του αλγορίθμου να ανταπεξέλθει στην περίπτωση που θέλει να ξαναπάει σε ένα σημείο που έχει αφήσει προηγουμένως ανοικτό σύμφωνα με τον κανόνα νο4 στη δεύτερη του περίπτωση, Δηλαδή όταν φτάνουμε σε ένα παλιό κόμβο από ένα παλιό μονοπάτι και δεν υπάρχει κάποιο άλλο καινούργιο μονοπάτι να ακολουθήσει οπότε και θα πρέπει να γυρίσουμε πίσω στο μονοπάτι που έφερε στο σημείο αυτό εξαρχής. Ο κανόνας αυτός όμως δεν παίρνει υπόψιν του καθόλου το κόστος για να φτάσουμε στο σημείο που μας υποδεικνύει. Και αν υπάρχει κάποιος συντομότερος δρόμος για να το κάνουμε αυτό, δεν απασχολεί καθόλου τον αλγόριθμο. Όπως επίσης δεν τον απασχολεί αν υπάρχει και κάποια άλλο μονοπάτι ανοικτό από κάποια άλλη διαδρομή, αλλά τυχαίνει αυτό να είναι πολύ κοντά στο σημείο που βρισκόμαστε τώρα. Έτσι η αναζήτηση αυτών των "κενών" μπορεί τελικά να επιτυγχάνεται αλλά έχει έναν πολύ μεγάλο παράγοντα κόστους. Βέβαια ο Tremaux επινοήθηκε με σκοπό να εγγυηθεί λύση για ένα λαβύρινθο και όχι για να βρίσκει συντομότερες διαδρομές από κόμβο σε κόμβο. Αυτό το πρόβλημα το λύνει ο αλγόριθμος του Dijkstra όπως αναφέραμε στο κεφάλαιο 2.3.2 και θα τον χρησιμοποιήσουμε σε δύο περιπτώσεις που θα περιγράψουμε πιο κάτω.

Έτσι λοιπόν παίρνοντας υπόψιν την πολύ αργή μέθοδο που χρησιμοποιεί ο Tremaux για να καλύψει τα "κενά" που άφησε πίσω του σε συνδυασμό με το γεγονός ότι πλέον έχουμε πληροφορία για τη δομή του γράφου που αναπαριστά τον λαβύρινθο μπορούμε να συνδυάσουμε και τον Dijkstra στις περιπτώσεις σαν και αυτές που περιγράψαμε. Προφανώς όταν τελειώσει η επίλυση του λαβυρίνθου δεν σημαίνει ότι κάθε κόμβος που θα βρίσκουμε θα είναι αυτής της κατηγορίας. Πρακτικά αυτό συμβαίνει όταν η χαρτογράφηση του λαβύρινθου έχει σχεδόν ολοκληρωθεί οπότε τα καινούργια μονοπάτια είναι πολύ λίγα και άρα μεγαλώνει και η πιθανότητα να φτάσουμε σε ένα μονοπάτι τέτοιας κατηγορίας. Για το συγκεκριμένο παράδειγμα λαβυρίνθου ο Dijkstra χρειάστηκε να χρησιμοποιηθεί τρεις με τέσσερις φορές μόνο και η χαρτογράφηση κατά κύριο λόγω ολοκληρώθηκε με τη χρήση του Tremaux που περιγράψαμε. Αυτό συνέβη επειδή με τον αλγόριθμο του Tremaux είχαμε αφήσει μόνο 4 ανοικτά μονοπάτια, όταν επιστρέψαμε κοντά στο σημείο που ξεκινήσαμε.

Ο αλγόριθμος του Dijkstra εγγυάται ότι η εύρεση του μονοπατιού μεταξύ ενός κόμβου και ενός άλλου θα είναι και το μικρότερο, οπότε αν κάθε φορά που βρισκόμαστε σε μια περίπτωση οπισθοδρόμησης χρησιμοποιούμε αυτή τη μέθοδο, σίγουρα θα έχουμε βελτιώσει την διαδικασία χαρτογράφησης. Η χειρότερη περίπτωση εκτέλεσης του Dijkstra είναι αυτή η οποία, η συντομότερη διαδρομή που βρίσκει, τυγχάνει να είναι ίδια με αυτή που θα ακολουθούσαμε αν πηγαίναμε με βάση τον Tremaux. Αυτή η περίπτωση είναι υπαρκτή αλλά είναι κυρίως αποτέλεσμα της τύχης και που θα μπορούσαμε να αποφύγουμε να το βλέπουμε συχνά αν για παράδειγμα, είχαμε έναν πολύ μεγάλο και ιδιαίτερα περίπλοκο λαβύρινθο. Στη δικιά μας εργασία ο Dijkstra βοήθησε στο να γλυτώσουμε το πολύ τέσσερις κινήσεις κάτι το οποίο δεν είναι σίγουρα μεγάλο νούμερο αλλά από την άλλη η μέθοδος που ακολουθούμε δεν είναι περιορισμένη μόνο για μικρούς λαβυρίνθους και σίγουρα θα βλέπαμε πολύ καλύτερα τη σύγκριση αν είχαμε φτιάξει έναν αρκετά μεγαλύτερο.

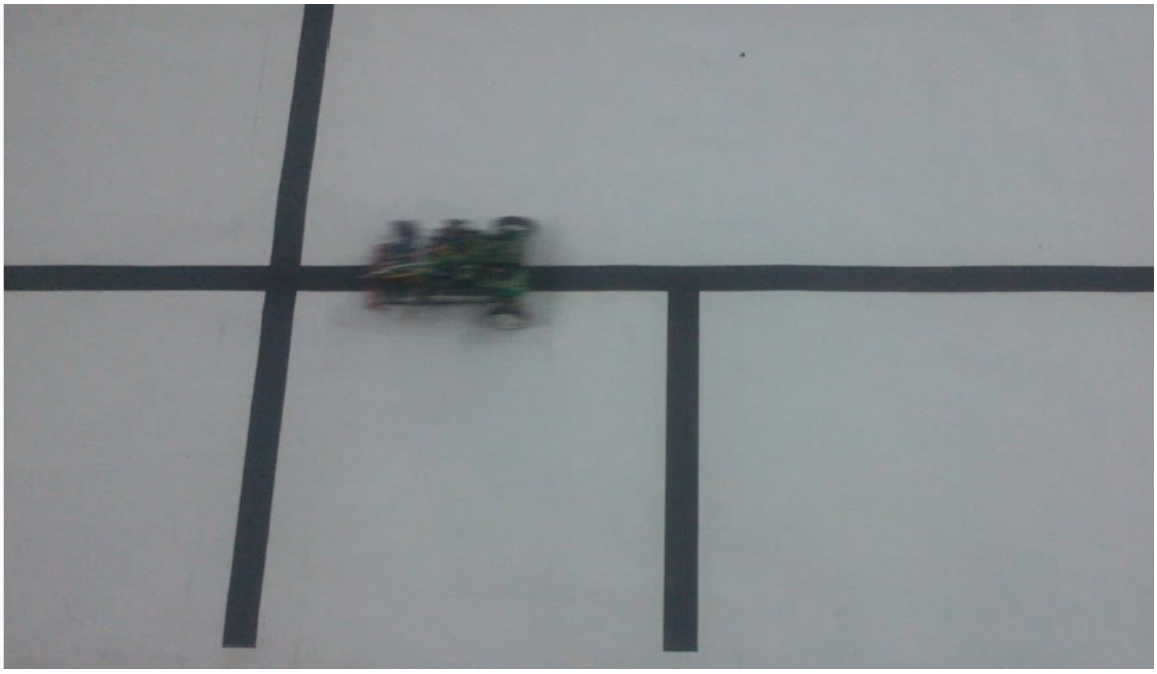
Η δεύτερη περίπτωση χρησιμοποίησης του Dijkstra είναι αυτή που αφορά τον τελικό στόχο της εργασίας. Δηλαδή τη δυνατότητα που έχει ο χρήστης, όταν έχει ολοκληρωθεί η διαδικασία χαρτογράφησης και άρα έχουμε συνολική πληροφορία για τον λαβύρινθο, να δίνει μέσα από το γραφικό περιβάλλον που υλοποιήσαμε, εντολές για το που θέλει να κατευθυνθεί το ρομπότ. Αυτό που έχει απλά να κάνει ο χρήστης είναι να πατήσει με ένα κλικ το σημείο που θέλει να θέσει σαν στόχο και τότε ο Dijkstra υπολογίζει το συντομότερο μονοπάτι που πρέπει να ακολουθηθεί. Σημειώνουμε ότι σε αυτή την περίπτωση όλοι κόμβοι έχουν προσπελαστεί οπότε προφανώς δεν χρησιμοποιούμε καθόλου τον αλγόριθμο του Tremaux.

## 4.12 Το Γραφικό περιβάλλον

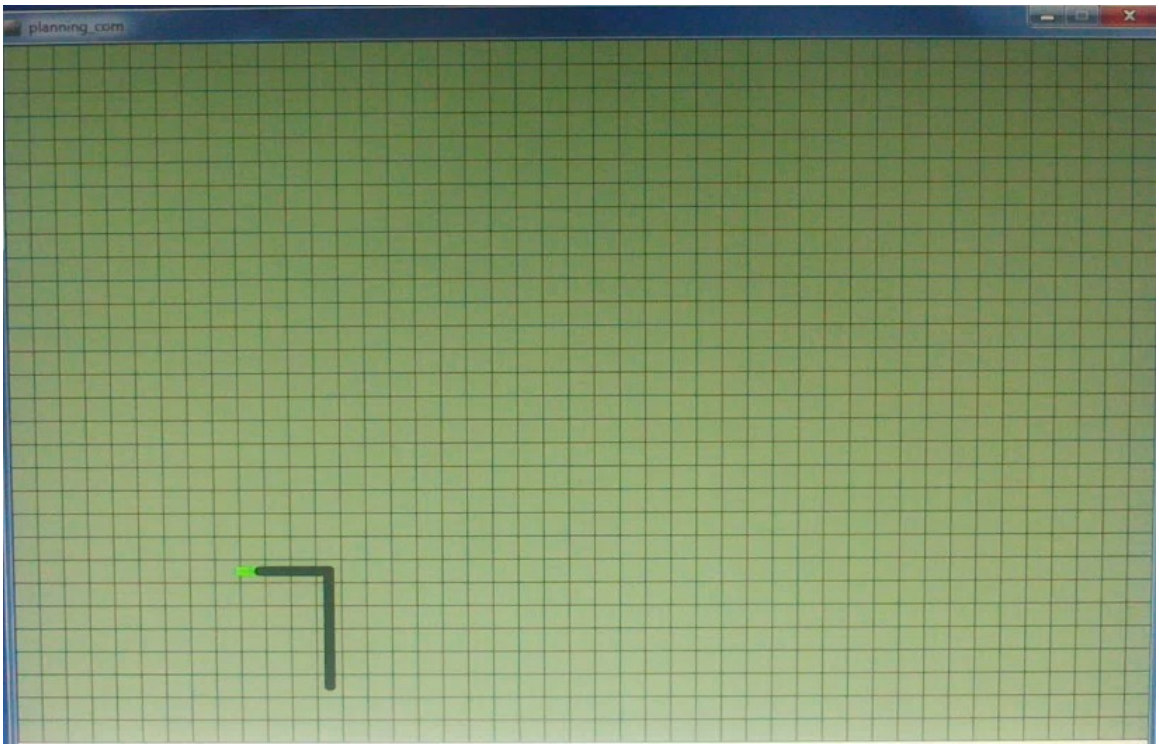
Το γραφικό περιβάλλον όπως αναφέραμε το υλοποιήσαμε στην Processing [18] και ήταν πολύ βασικό κομμάτι της εργασίας μας. Μέσω αυτού, αποτυπώνεται κάθε κίνηση που γίνεται στο χώρο του λαβυρίνθου ενώ οι συναρτήσεις που είναι υπεύθυνες για την αποτύπωση κάθε διαδρομής χρησιμοποιούν μια τεχνική για να δημιουργείται δυναμικά η εικόνα, δίνοντας μια ψευδαίσθηση κίνησης. Στο γραφικό περιβάλλον, κάθε μονοπάτι έχει μήκος 100 pixels και αυτό αντιστοιχεί σε μία γραμμή του λαβυρίνθου. Όταν τελειώσει η αναζήτηση του τερματικού σημείου του λαβυρίνθου, στην περίπτωση που χρησιμοποιηθεί ο αλγόριθμος του Dijkstra για την εύρεση ενός

ελαχίστου μονοπατιού τότε η διαδρομή αυτή τονίζεται με χρώμα μπλε. Τέλος μέσω του γραφικού περιβάλλοντος, έχουμε τη δυνατότητα να επιλέξουμε έναν κόμβο του λαβυρίνθου και αυτός θα τεθεί αυτόματα σαν στόχος ο οποίος θα πρέπει να φτάσει το όχημα μας, μέσω της συντομότερης διαδρομής. Προϋπόθεση για να γίνει αυτό είναι να έχει τελειώσει η χαρτογράφηση του λαβυρίνθου και το πάτημα του κουμπιού να γίνει σε μια κοντινή απόσταση (μέχρι 30 pixels) από το σημείο που υποδηλώνεται ένας κόμβος. Σημειώνουμε ότι για να αντιμετωπίσουμε το γεγονός ότι δύο γραμμές του λαβυρίνθου σε σειρά, πρέπει να φανούν και στο γραφικό περιβάλλον σαν μια γραμμή διπλάσιου μήκους από τις υπόλοιπες, αντιμετωπίζουμε τα σημεία που ενώνονται οι δύο γραμμές σαν κόμβους, ακριβώς όπως θα κάναμε αν βρίσκαμε μία διασταύρωση. Αυτό έχει σαν αποτέλεσμα να μπορούμε να διαλέξουμε και σημεία που βρίσκονται στη μέση μιας μεγάλης, σε μήκος, γραμμής αν και στην πραγματικότητα το σημείο αυτό δεν είναι κόμβος. Στη συνέχεια δίνουμε δύο παραδείγματα εικόνων της λειτουργίας του γραφικού περιβάλλοντος.

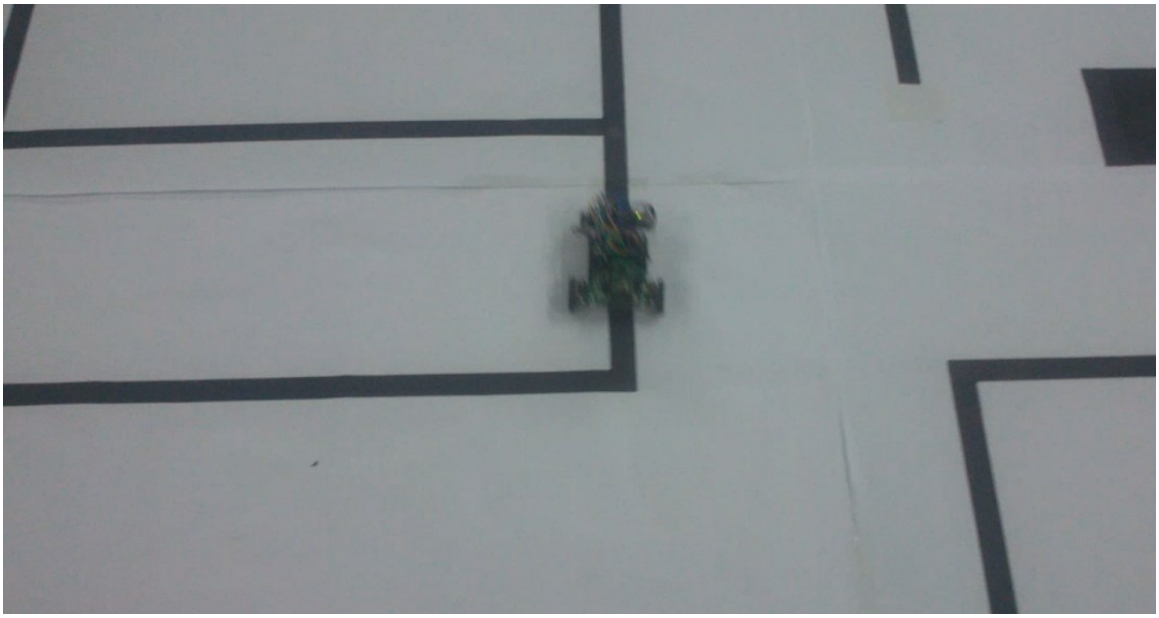
Στις εικόνες που ακολουθούν δείχνουμε ένα παράδειγμα με στιγμιότυπα από την πορεία που ακολουθεί το όχημα στην περίπτωση χρήσης του αλγόριθμου του Tremaux με την ευριστική συνάρτηση και τη χρήση του Dijkstra. Στην εικόνα 51 φαίνεται η θέση του οχήματος μετά από την αρχική κίνηση north ακολουθούμενη από τη west.



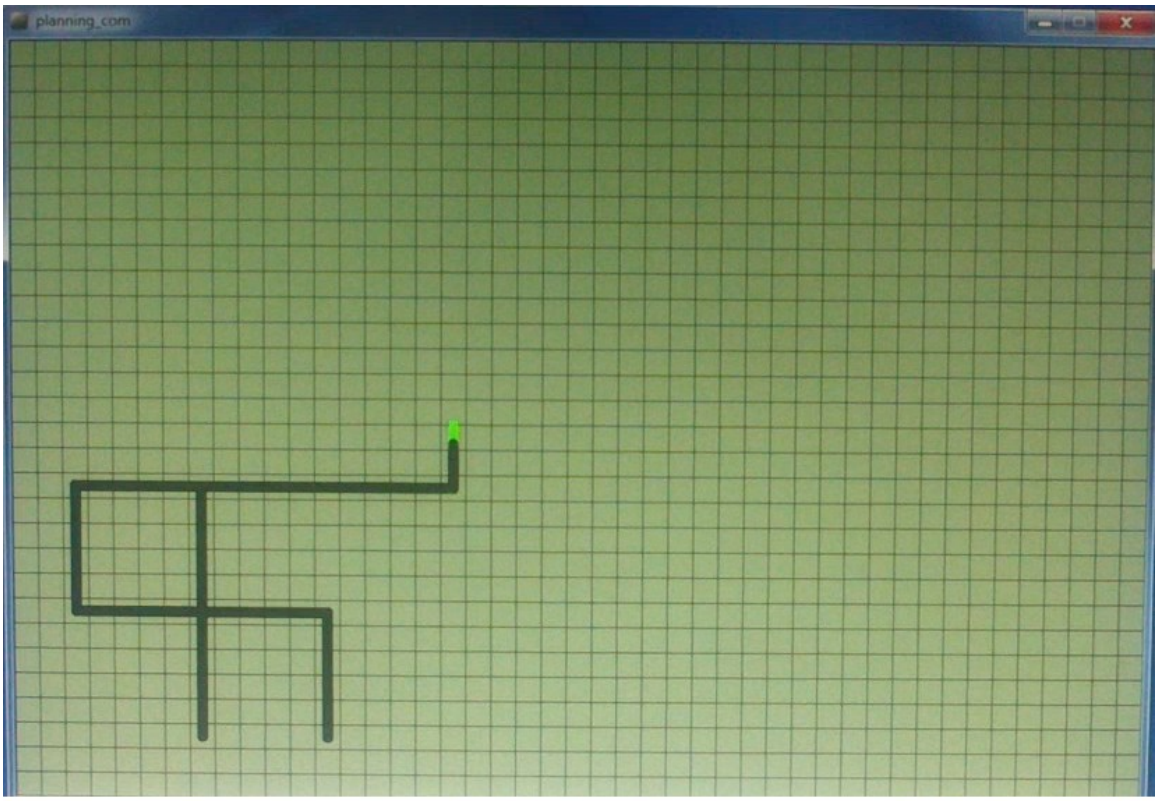
Εικ. 51  Η θέση του οχήματος λίγο μετά την έναρξη



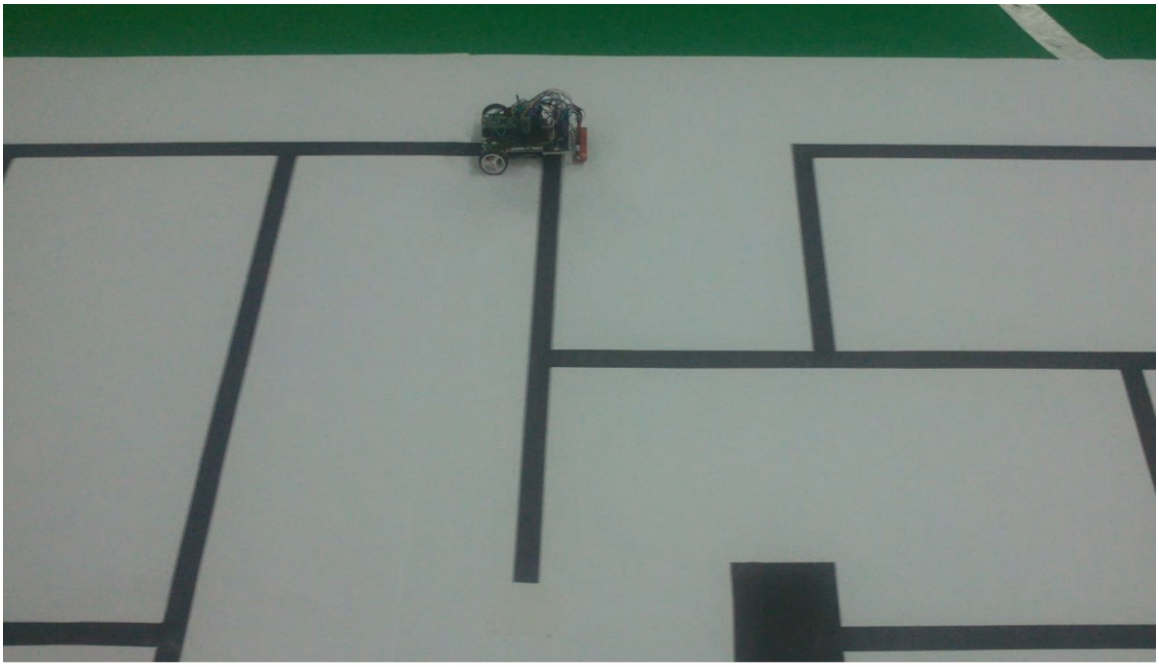

Εικ. 52 Στιγμιότυπο του γραφικού περιβάλλοντος μετά τις εντολές north και west.



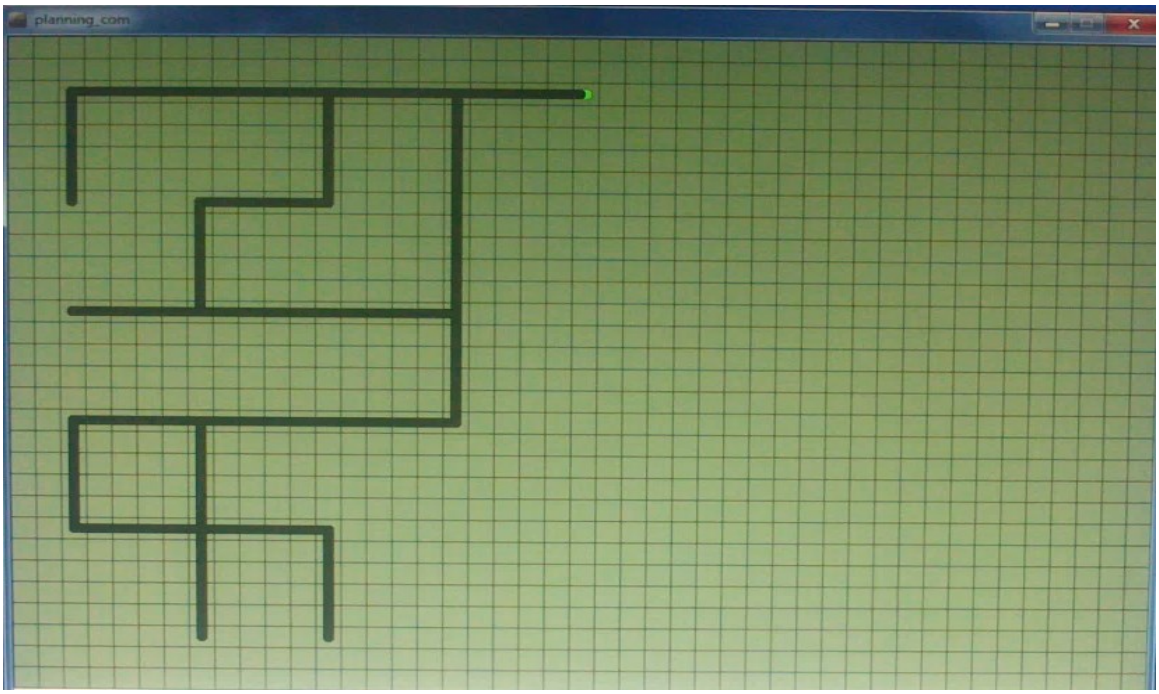
Εικ. 53 Η θέση του οχήματος στο δεύτερο στιγμιότυπο της κίνησης με την ευριστική συνάρτηση.



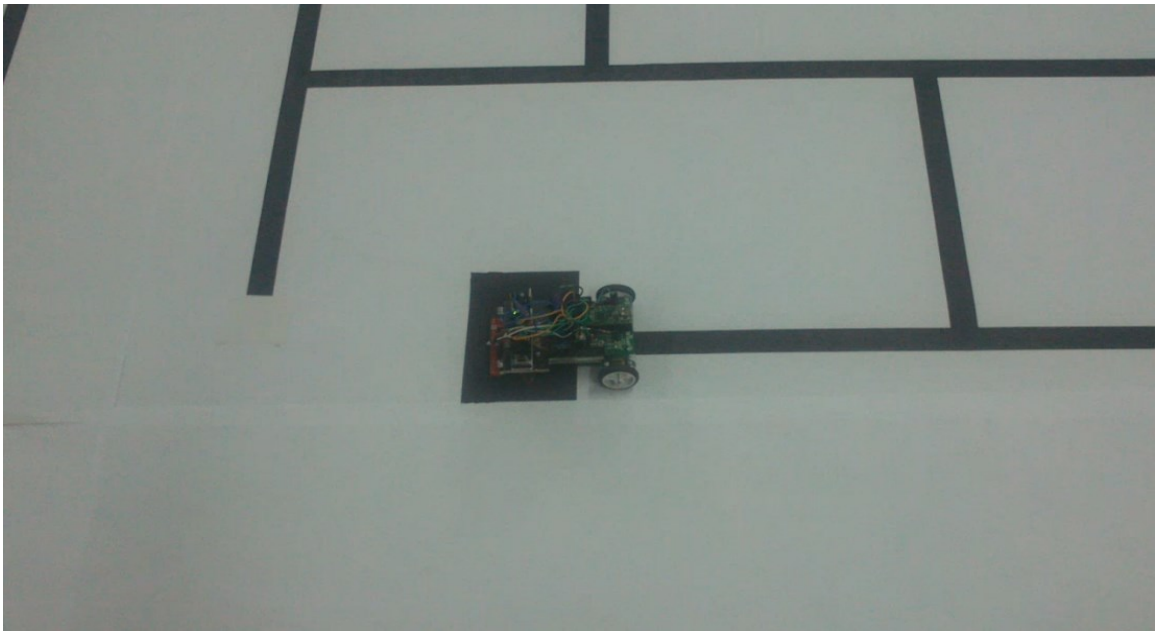

Εικ. 54 Το δεύτερο στιγμιότυπο στο γραφικό περιβάλλον, της κίνησης με την ευριστική συνάρτηση.



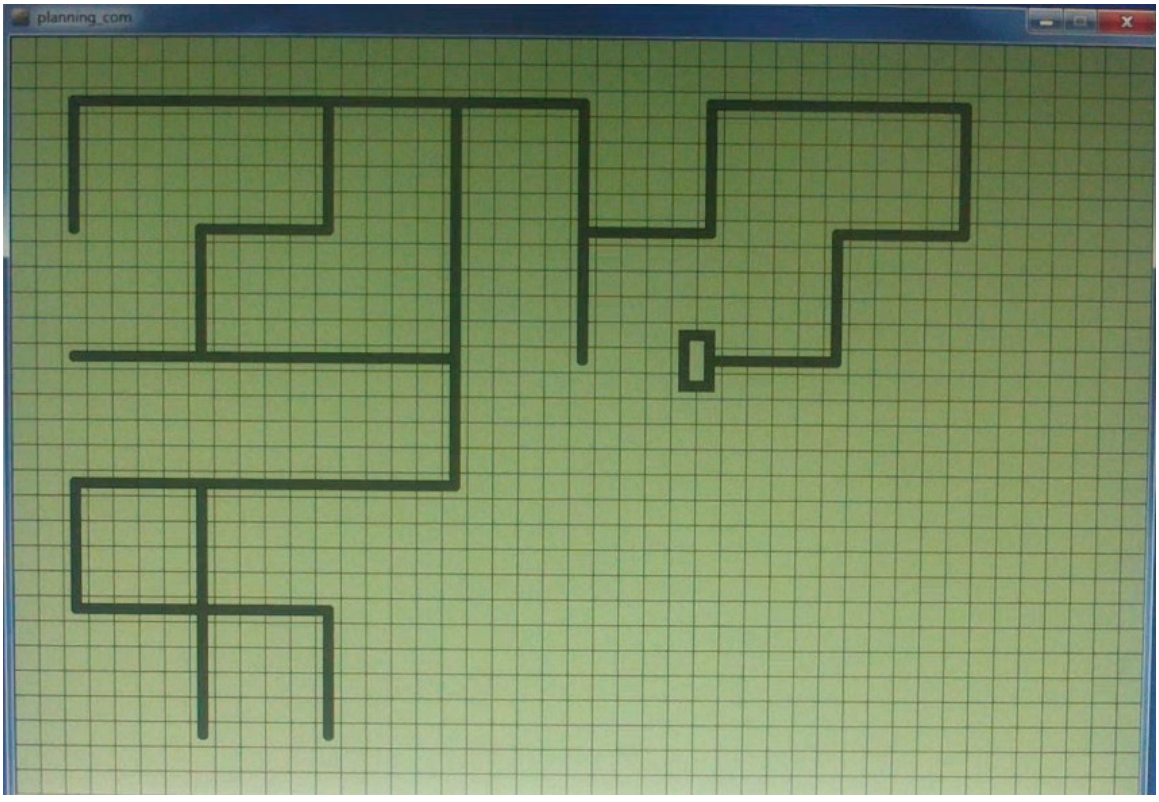
Εικ. 55 Η θέση του οχήματος στο τρίτο στιγμιότυπο της κίνησης με την ευριστική συνάρτηση.




Εικ. 56 Το τρίτο στιγμιότυπο στο γραφικό περιβάλλον με την ευριστική συνάρτηση.

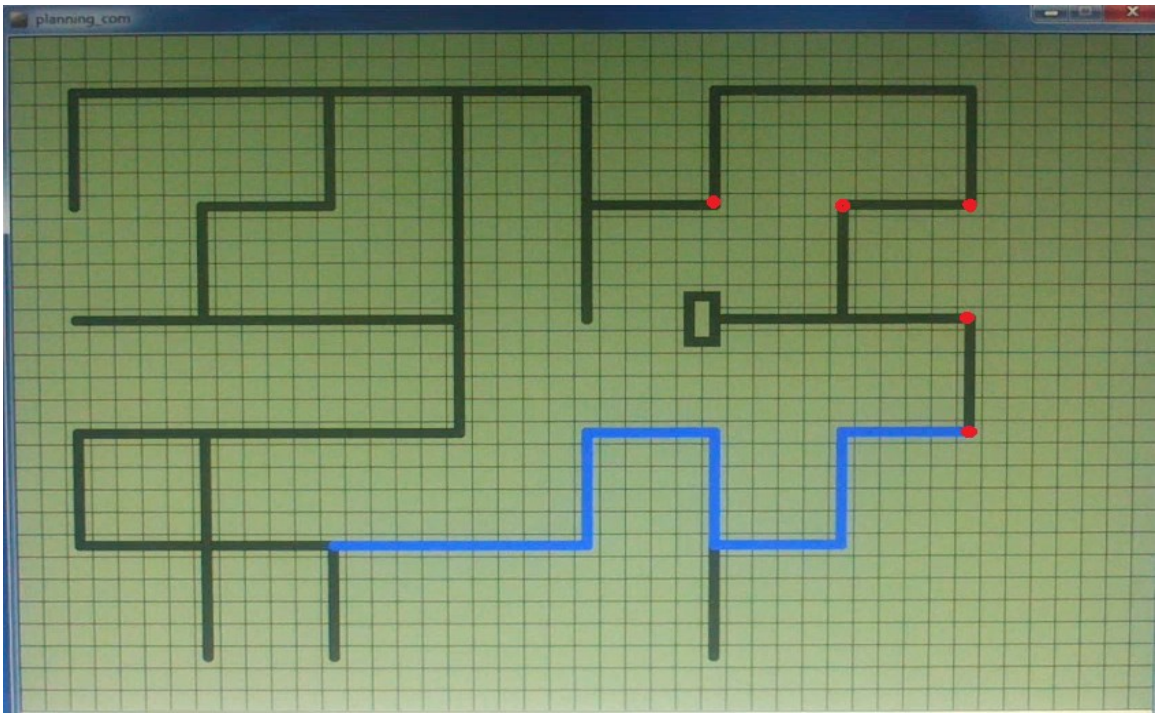


Εικ. 57 Η θέση του οχήματος στο τέταρτο στιγμιότυπο της κίνησης με την ευριστική συνάρτηση




Εικ.58 Το τέταρτο στιγμιότυπο στο γραφικό περιβάλλον της κίνησης με την ευριστική συνάρτηση.

Εικ. 59 Η θέση του οχήματος στο πέμπτο στιγμιότυπο της κίνησης με την ευριστική συνάρτηση

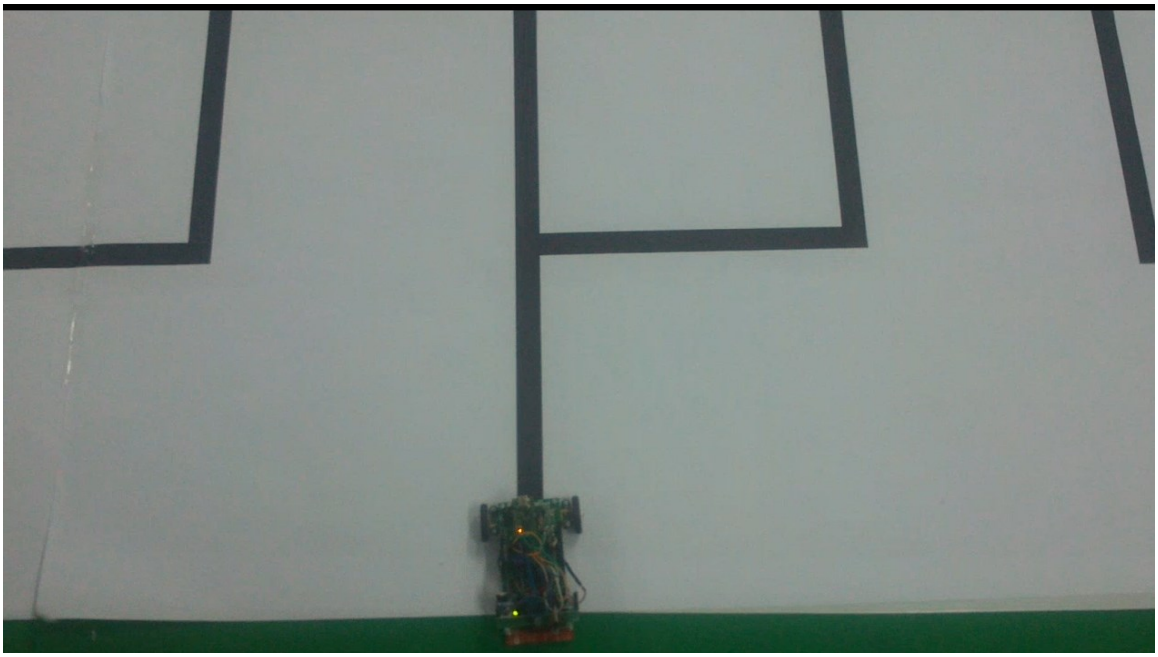



Εικ.60 Το πέμπτο στιγμιότυπο στο γραφικό περιβάλλον της κίνησης με την ευριστική συνάρτηση.

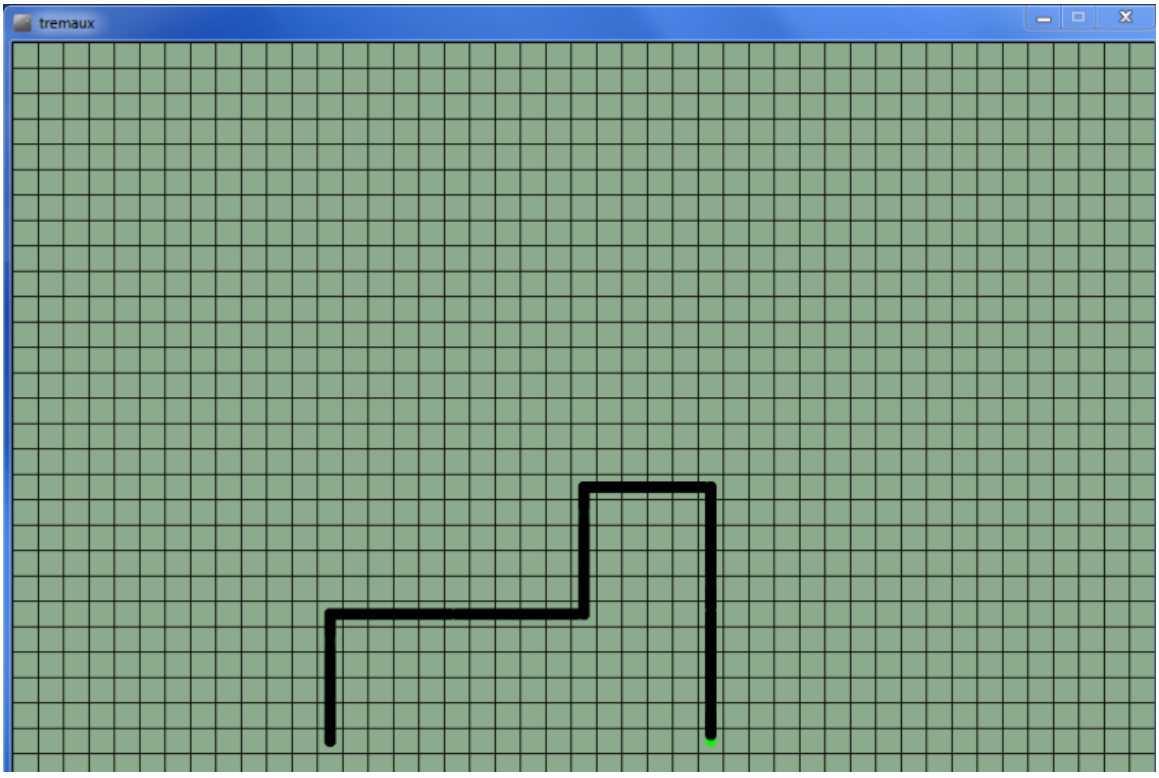
Στην εικόνα 60 η διαδρομή που τονίζεται με μπλε χρώμα είναι και η συντομότερη διαδρομή από το σημείο που βρίσκεται το όχημα σε έναν από τους

ανεξερεύνητους κόμβους που φαίνονται με κόκκινο χρώμα. Η συντομότερη διαδρομή εμφανίζεται γιατί το σημείο που φαίνεται και στην εικόνα 59 έχει προσπελαστεί στο παρελθόν και καμία καινούργια διαδρομή δεν υπάρχει που να ξεκινάει από αυτό. Από το σημείο αυτό ξεκινάει το ρομποτικό όχημα οπισθοδρόμηση για περάσει από τους ανοικτούς κόμβους μέχρι να τους καλύψει όλους. Για το παράδειγμα αυτό και με τη χρήση της ευριστικής συνάρτησης, η χαρτογράφηση ολοκληρώθηκε σε εξηνταεννέα (69) βήματα.

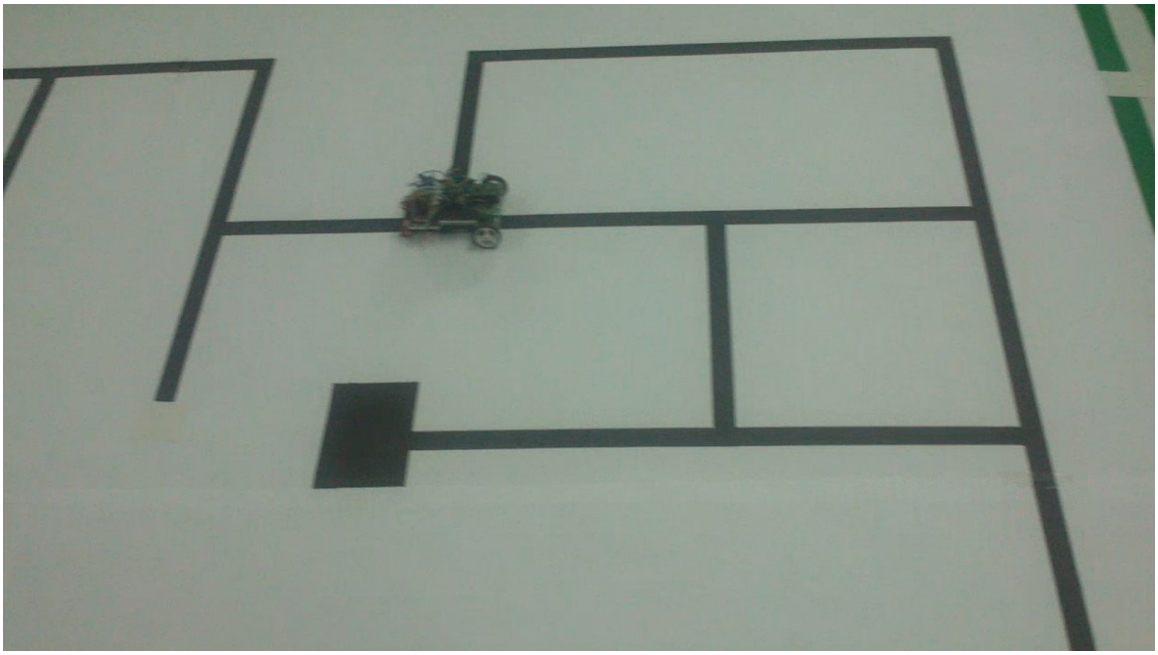
Στο παράδειγμα που ακολουθεί εφαρμόζουμε τον αλγόριθμο του Tremaux χωρίς την ευριστική μέθοδο, οπότε κάθε κίνηση που μπορεί να γίνει ελεύθερα, όπως ορίζει ο αλγόριθμος, τώρα γίνεται στην τύχη. Το σημείο έναρξης είναι το ίδιο με προηγουμένως. Για το συγκεκριμένο παράδειγμα μετρήσαμε 79 βήματα μέχρι να ολοκληρωθεί η χαρτογράφηση. Κατά τη διαδικασία διάσχισης του λαβυρίνθου φαίνεται η μεγάλη διαφορά των δύο εκδοχών του αλγορίθμου, όσον αφορά το πλήθος των κόμβων που μένουν ανοικτοί. Στην εικόνα 61 το στιγμιότυπο είναι παρμένο μετά από την ακολουθία εντολών, north, east, east, north, east, south, south.



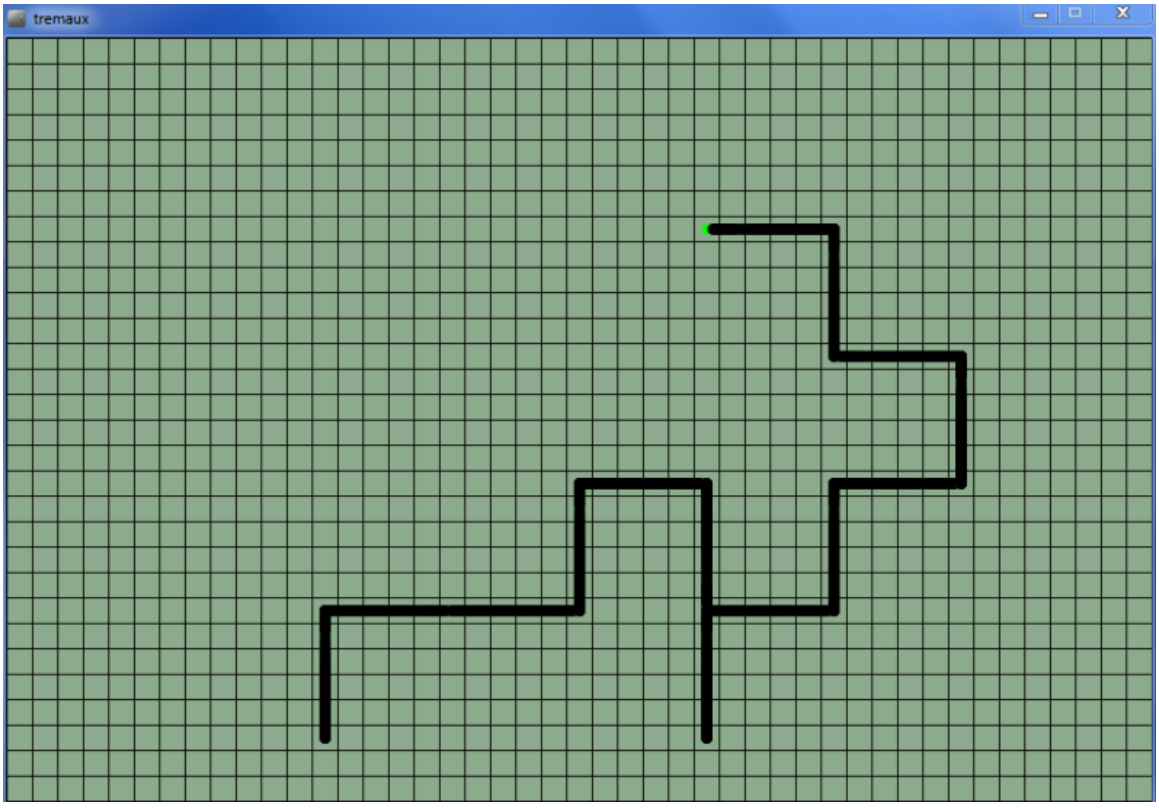
Εικ.61 Η θέση του οχήματος στο πρώτο στιγμιότυπο της κίνησης χωρίς την ευριστική συνάρτηση.



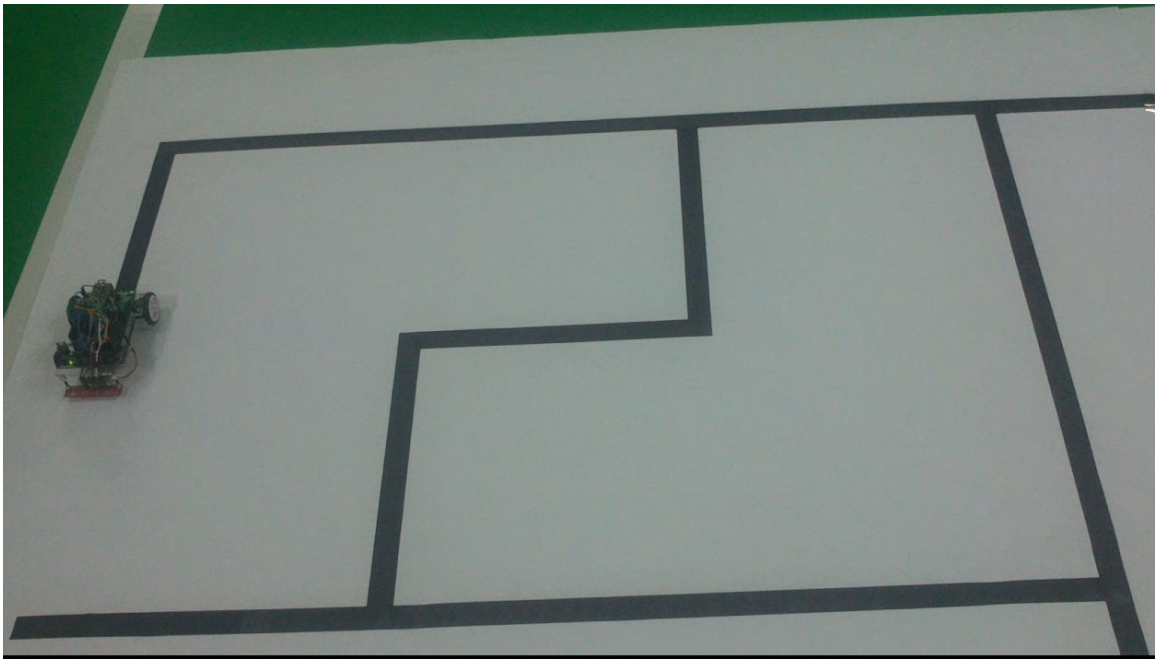

Εικ.62 Το πρώτο στιγμιότυπο στο γραφικό περιβάλλον της κίνησης χωρίς την ευριστική συνάρτηση.



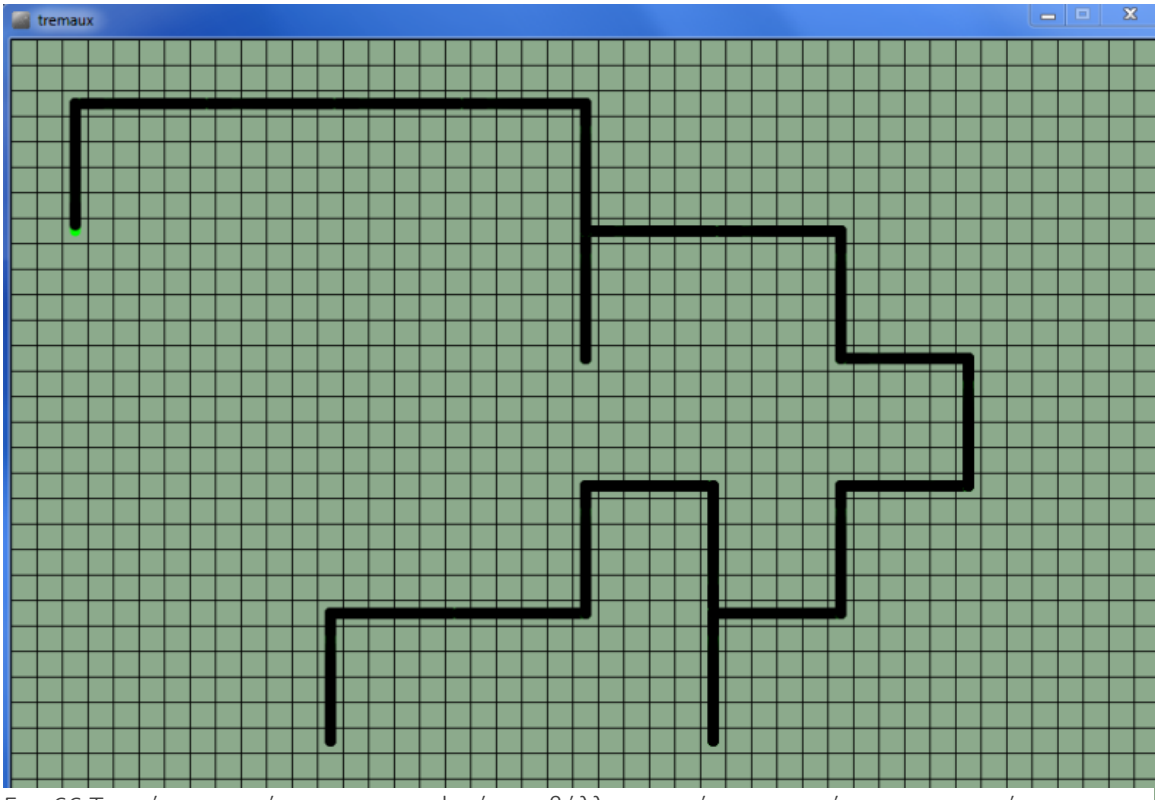
Εικ. 63 Η θέση του οχήματος στο δεύτερο στιγμιότυπο της κίνησης χωρίς την ευριστική συνάρτηση.



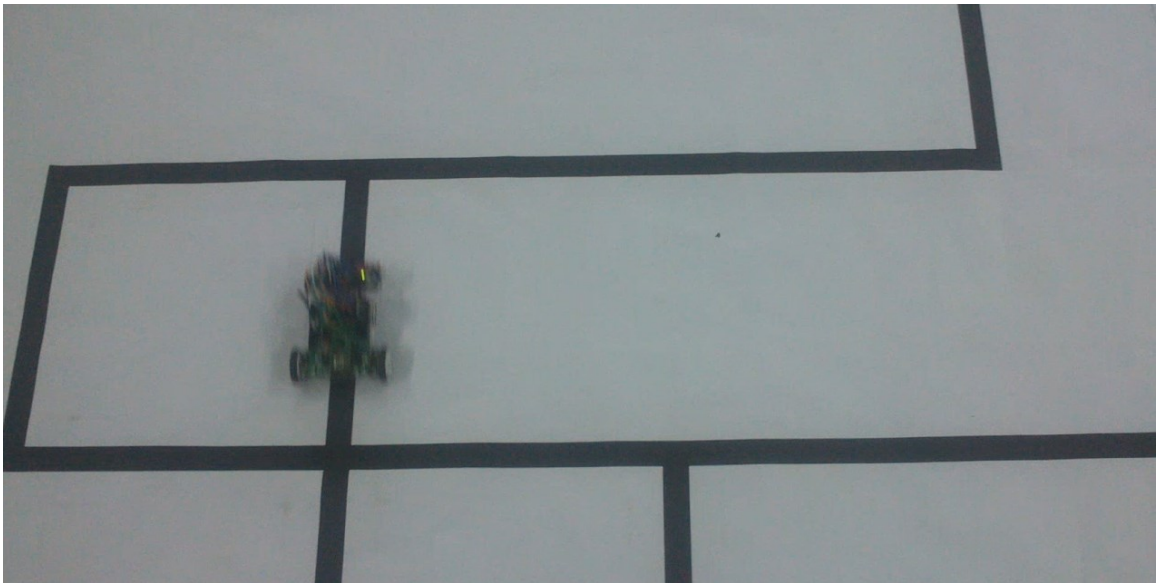

Εικ. 64 Το δεύτερο στιγμιότυπο στο γραφικό περιβάλλον της κίνησης χωρίς την ευριστική συνάρτηση.



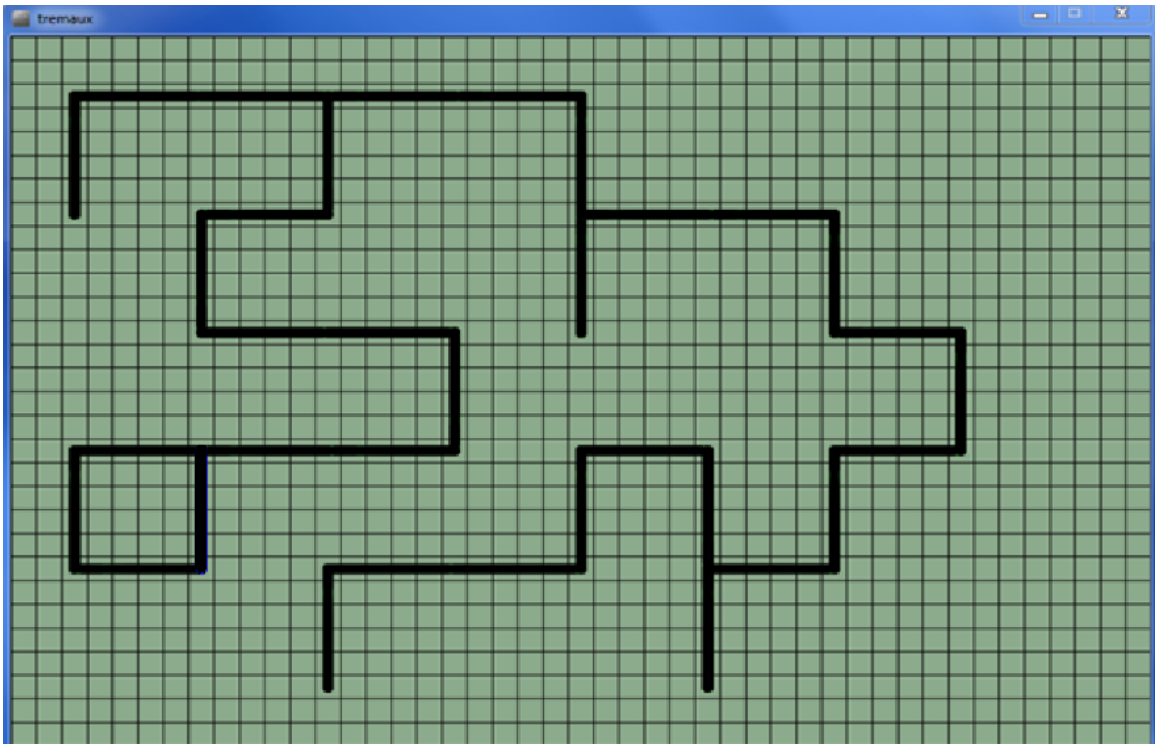
Εικ. 65 Η θέση του οχήματος στο τρίτο στιγμιότυπο της κίνησης χωρίς την ευριστική συνάρτηση.



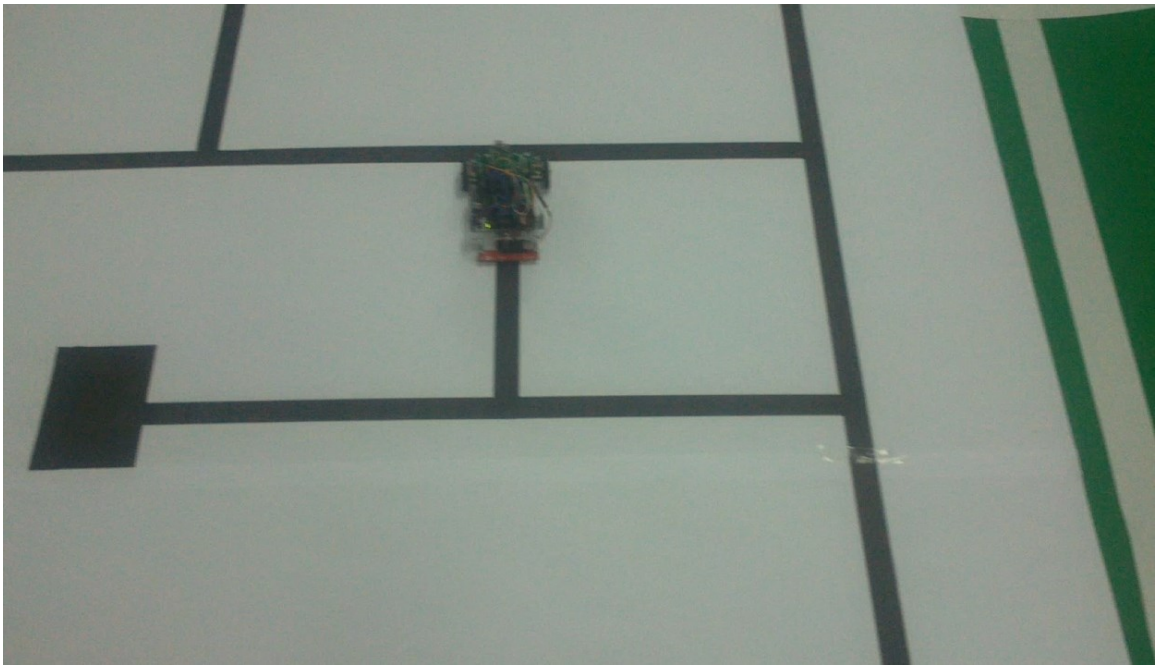

Εικ. 66 Το τρίτο στιγμιότυπο στο γραφικό περιβάλλον της κίνησης χωρίς την ευριστική συνάρτηση.



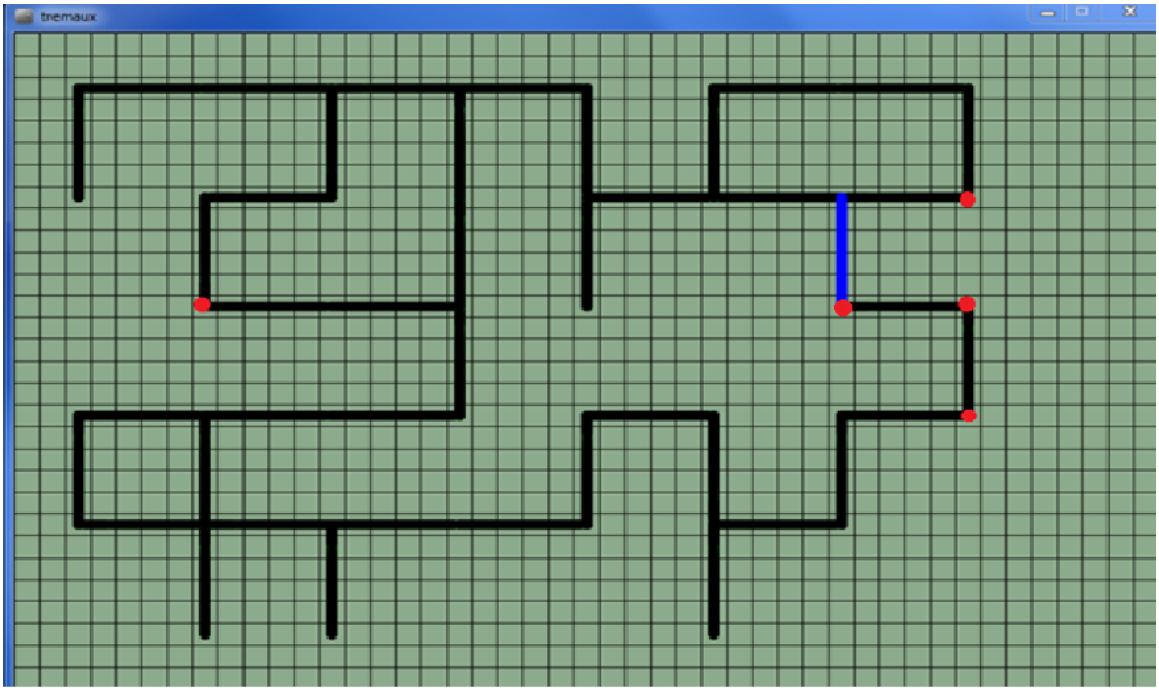
Εικ. 67 Η θέση του οχήματος στο τέταρτο στιγμιότυπο της κίνησης χωρίς την ευριστική συνάρτηση.




Εικ. 68 Το τέταρτο στιγμιότυπο στο γραφικό περιβάλλον της κίνησης χωρίς την ευριστική συνάρτηση.



Εικ. 69 Η θέση του οχήματος στο πέμπτο στιγμιότυπο της κίνησης χωρίς την ευριστική συνάρτηση.




Εικ. 70 Το πέμπτο στιγμιότυπο στο γραφικό περιβάλλον της κίνησης χωρίς την ευριστική συνάρτηση.

# Κεφάλαιο 5

## Αποτελέσματα

Σε αυτό το κεφάλαιο παρουσιάζουμε τα αποτελέσματα της δουλειάς μας τόσο για την κατασκευή του ρομποτικού οχήματος που φτιάξαμε όσο και για το κομμάτι της συνεργατικής λειτουργίας του μαζί με τους αλγορίθμους και τις μεθόδους που υλοποιήσαμε για το πρόβλημα της επίλυσης του λαβυρίνθου, της χαρτογράφησης και του προβλήματος SPP. Δυστυχώς δεν υπάρχει ένας εύκολος τρόπος να μετρήσουμε τα αποτελέσματα της δουλειάς μας με έναν ενιαίο και τυποποιημένο τρόπο γιατί προσπαθήσαμε να ακουμπήσουμε πολλά διαφορετικά πεδία με σκοπό να φτιάξουμε κάτι πρωτότυπο. Αυτό όμως δεν έχει και πολύ σημασία γιατί τους σκοπούς της εργασίας πιστεύουμε ότι τους πετύχαμε αφήνοντας πίσω μια πολύ καλή βάση για περαιτέρω εξέλιξη και πειραματισμό.

## 5.1 Η ρομποτική κατασκευή που υλοποιήσαμε

Όσον αφορά τη ρομποτική κατασκευή αυτή αποτέλεσε ένα πολύ σημαντικό στοιχείο της εργασίας. Συνδυάζοντας ένα μικροελεγκτή με αισθητήρες και τα υπόλοιπα περιφερειακά στοιχεία υλοποιήσαμε ένα ρομποτικό όχημα, το οποίο εκτέλεσε το ρόλο που του αναθέσαμε και με το παραπάνω. Πιστεύουμε ότι έχει μια αρκετά καλή συμπεριφορά και με λίγες γραμμές κώδικα παραπάνω θα μπορεί να έχει ακόμα μεγαλύτερες δυνατότητες. Κατάφερε να κινείται στο χώρο με τρόπο που θέλαμε, να μπορεί να παίρνει με έναν πολύ αξιόπιστο τρόπο τα ερεθίσματα από το περιβάλλον του και το πιο σημαντικό είναι ότι έχει δυνατότητες για πολλά παραπάνω. Δεν εξαντλήσαμε σε καμία περίπτωση την υπολογιστική του ισχύ, παρόλες τις δυσκολίες και περιορισμούς που μπορεί να έχει ο προγραμματισμός ενός μικροελεγκτή. Με λίγες αλλαγές στον κώδικα του και ίσως με μετατροπές στο υλικό του θα μπορούσε να αποκτήσει και πολύ μεγαλύτερη ταχύτητα και να αποκτήσει τη δυνατότητα για πλήρη αυτονομία.

## 5.2 Η αλγοριθμική επεξεργασία που υλοποιήσαμε

Στο κομμάτι αυτό αξιολογούμε τη διαδικασία αναπαράστασης του λαβυρίνθου σε γράφο καθώς ήταν ένα ιδιαίτερα σημαντικό κομμάτι της εργασίας μας. Καταφέραμε να φτιάξουμε μια δομή αποτύπωσης του λαβυρίνθου η οποία δεν υλοποιείται με στατικό τρόπο καθώς δεδομένα για νέους κόμβους και μονοπάτια έρχονται συνεχώς κάτι το οποίο ήταν ένας παράγοντας δυσκολίας παραπάνω. Φτιάξαμε ένα αξιόπιστο σύστημα κωδικοποίησης της πληροφορίας που παίρνουμε από τους αισθητήρες και με έναν γρήγορο τρόπο θέτουμε κάθε φορά τις σωστές τιμές κατάστασης για κάθε κόμβο.

## 5.3 Το πρόβλημα του σχεδιασμού

Στην ενότητα αυτή παρουσιάζουμε τα αποτελέσματα για τους αλγόριθμους που χρησιμοποιήσαμε για το κομμάτι της επίλυσης του λαβυρίνθου και της διαδικασίας της χαρτογράφησης του. Καθώς επίσης και για το πρόβλημα εύρεσης των συντομότερων μονοπατιών. Πιστεύω ότι η επιλογή του αλγόριθμου του Tremaux ήταν η καλύτερη δυνατή για την επίλυση ενός λαβύρινθου σαν και αυτόν που υλοποιήσαμε. Από μόνος του ο Tremaux πάντα θα βρίσκει τη λύση αλλά εμείς προτείναμε τη χρήση του υλοποιώντας μια ευριστική μέθοδο που χρησιμοποιεί την τοπικότητα σαν κριτήριο αναζήτησης. Για το συνδυαστικό πρόβλημα που είχαμε θέσει, η προσέγγιση μας έδωσε πολύ καλά αποτελέσματα αν και δεν είχαμε τη δυνατότητα να συγκρίνουμε τις δύο διαφορετικές εκδοχές σε μια προσομοίωση για να δούμε τα αποτελέσματα σε μεγαλύτερη κλίμακα. Για τον δικό μας λαβύρινθο η χειρότερη απόδοση που μετρήσαμε με τον Tremaux στην απλή του μορφή ήταν 81 κινήσεις και για τη δικιά μας προσέγγιση μετρήσαμε 69 και ίσως να υπάρχει και χειρότερο σενάριο που δεν το υπολογίσαμε. Ήταν δηλαδή κατά 15% καλύτερος με τη χρήση της μεθόδου που υλοποιήσαμε πάντα για το δικό μας περιβάλλον.

Για το πρόβλημα του SPP, τον αλγόριθμο του Dijkstra δεν τον τροποποιήσαμε καθόλου αλλά αυτό που προτείναμε είναι η συνεργατική λειτουργία του Dijkstra με τον Tremaux στις περιπτώσεις που θέλουμε να αποφύγουμε την αναζήτηση κάποιου

ανοικτού μονοπατιού με τον τρόπο που ορίζει ο Tremaux στον κανόνα νούμερο 4 στη δεύτερη περίπτωση. Αποδείξαμε ότι αυτή η προσέγγιση δουλεύει αλλά δυστυχώς η τοπολογία του λαβυρίνθου και το μέγεθος του δεν μας δείχνει τη βελτίωση με αποτελεσματικό τρόπο. Μετρήσαμε μόνο 4 βήματα στην καλύτερη περίπτωση που γλυτώσαμε με την προσέγγιση αυτή και σε πολλές περιπτώσεις το αποτέλεσμα του Dijkstra συμπίπτει με το αποτέλεσμα που θα έδινε ο Tremaux. Σημειώνουμε ότι με μικρές αλλαγές στην τοπολογία του λαβυρίνθου κρατώντας το ίδιο μέγεθος μπορούμε να γλιτώσουμε και πάνω από δέκα κινήσεις. Όσον αφορά τη δεύτερη περίπτωση που χρησιμοποιούμε τον Dijkstra εκεί δηλαδή που ο χρήστης ορίζει ένα σημείο-στόχο, ο Dijkstra μας δίνει πάντα τη σωστή διαδρομή.

# Κεφάλαιο 6

## Συμπεράσματα

Η ανάπτυξη ρομποτικών εφαρμογών είναι ένα πεδίο της τεχνολογίας που παρουσιάζει για μας ιδιαίτερο ενδιαφέρον και σίγουρα αυτή η εργασία ήταν ένα ενθαρρυντικό βήμα για παραπάνω ενασχόληση με το πεδίο αυτό. Προσπαθήσαμε να φτιάξουμε μια εργασία όπου θα παρουσιάζει ένα τρόπο ανάπτυξης μιας σχετικά απλής αλλά αποτελεσματικής εφαρμογής βάζοντας την να λύσει προβλήματα αλγοριθμικής φύσης. Ελπίζουμε να αποτελέσει έμπνευση για κάποιον που θα ήθελε να ακολουθήσει τον δικό μας τρόπο προσέγγισης αλλά ακόμα καλύτερα αν θέλει και να τον αναπτύξει περεταίρω.

Μελλοντικά σίγουρα θα μπορούσαν να γίνουν πολλά για να βελτιώσουν σημεία που κρίνουμε ότι θα μπορούσαν να εξελιχθούν. Σίγουρα ένα πολύ βασικό στοιχείο είναι η ταχύτητα την οποία και περιορίσαμε για να έχουμε καλύτερο έλεγχο. Οπότε ένα διαφορετικό μοντέλο ελέγχου βασισμένο στον PID έλεγχο θα ήταν ίσως το πρώτο πράγμα που θα κάναμε. Δεύτερο σημείο που θα μπορούσε να εξελιχθεί θα ήταν πέρα από το θέμα της χαρτογράφησης, της επίλυσης λαβυρίνθου και της εύρεσης των ελάχιστων διαδρομών να βάζαμε και το πρόβλημα της εύρεσης της συνολικής ελάχιστης διαδρομής που περνάει από όλα τα ανοικτά σημεία που έχει αφήσει πίσω του ενδεχομένως ο Tremaux. Να λύνει δηλαδή ο αλγόριθμος, το πρόβλημα του Πλανόδιου Πωλητή. Βέβαια αυτό είναι ένα μόνο παράδειγμα προβλήματος και είμαστε σίγουροι πως και πολλά άλλα θα μπορούσαν να τεθούν. Ίσως από τις πιο ενδιαφέρουσες δουλειές θα ήταν το σύστημα του λαβυρίνθου να μετεξελιχθεί σε ένα κατευθυνόμενο σύστημα, δηλαδή να έχει διαδρομές που επιτρέπεται η προσπέλαση τους μόνο από μία κατεύθυνση. Αυτό θα μπορούσε να γίνει χρησιμοποιώντας λωρίδες διαφορετικού χρώματος αλλά αυτό θα έβαζε αρκετή επιπλέον δυσκολία. Τέλος θα μπορούσαμε μελλοντικά να χρησιμοποιούσαμε διαφορετικές τεχνικές υλοποίησης της χαρτογράφησης και αλγορίθμους εκτίμησης της θέσης.

# Βιβλιογραφική Αναφορά


[1] Russell and Peter Norvig. 2005. *Τεχνητή Νοημοσύνη, μια σύγχρονη προσέγγιση*, Εκδόσεις Κλειδάριθμος, (Κεφ. 3 σ. 92-125) 77, (Κεφ. 4 σ. 131-169) 82, (Κεφ. 11 σ. 433-474) 46

[2] Latombe, J.C. 1990. *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991, Appendix C (p. 603-609) 78

[3] M. Babula, "Simulated maze solving algorithms through unknown mazes," in Proceedings of the XVIIIth Concurrency, Specification and Programming Workshop, 2009, pp. 13-22. 42

[4] Wikipedia, the free encyclopedia. [Online] http://en.wikipedia.org/wiki/Maze_solving_algorithm 42

[5] Steven M. Lavalle, *Planning Algorithms*, Cambridge University Press (Chapter 4 p 127-148), (Chapter 11 p. 560-575) 47

[6] Παναγιώτης Δ. Μποζάνης, 2003. *Αλγόριθμοι,* Εκδόσεις Τζιόλα 33

[7] David Russel-Mitchell Thornton, *Introduction to Embedded Systems: Using ANSI C and the Arduino Development Environment (Synthesis Lectures on Digital Circuits and Systems)* 9

[8] Heath, Steve (2003). *Embedded systems design* 9

[9] Barnett, Cox and o'Cull, 2006. *Embedded C Programming and the Atmel AVR* 9

[10] University of Vermont, http://www.cems.uvm.edu/~rsnapp/teaching/cs32/lectures/tremaux.pdf 44

[11] Emil Keyder & Blai Bonet, Universitat Pompeu Fabra & Universidad Simon Bolivar, Heuristics for Planning, http://icaps09.uom.gr/tutorials/tut1.pdf 78

[12] Kane Bonnette and Stephen Hilber, Information Scarce Maze Solving, Georgia College of Tech Computing, http://www.cc.gatech.edu/~mstilman/class/RIP08/FINAL_PROJECTS/KaneStephen.pdf

[13] Elgar P. 2000. *Αισθητήρες μέτρησης και ελέγχου* 20

[14] Daniel Shiffman, 2008. *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction* 85

[15] National Instruments, 2011, Introduction to RF & Wireless Communications Systems [Online] http://www.ni.com/white-paper/3541/en/ 10