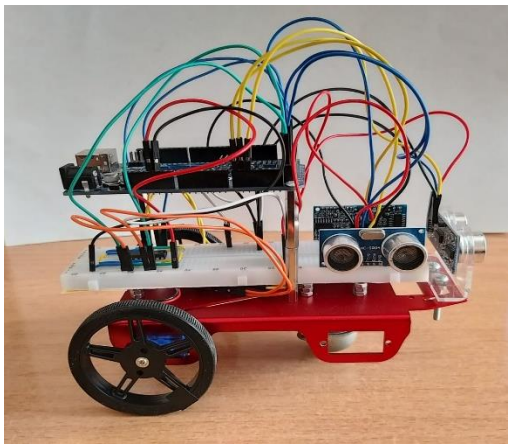
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ

ΘΕΜΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

"ΑΥΤΟΚΙΝΟΥΜΕΝΟ ΟΧΗΜΑ ΕΥΡΕΣΗΣ ΒΕΛΤΙΣΤΗΣ ΔΙΑΔΡΟΜΗΣ ΛΑΒΥΡΙΝΘΟΥ"



ΟΝΟΜΑΤΑ ΦΟΙΤΗΤΩΝ:

ΕΥΔΑΙΜΩΝ ΝΙΚΟΛΑΟΣ

ΝΙΚΟΛΟΥΔΑΚΟΣ ΒΑΣΙΛΕΙΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

ΜΙΧΑΗΛ ΠΑΠΟΥΤΣΙΔΑΚΗΣ

ΑΙΓΑΛΕΩ, ΑΠΡΙΛΙΟΣ 2019

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Οι κάτωθι υπογεγραμμένοι Ευδαίμων Νικόλαος του Γεωργίου και Νικολουδάκος Βασίλειος του Δημητρίου, φοιτητές του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής του Πανεπιστημίου Δυτικής Αττικής, πριν αναλάβουν την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκαν για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε, ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα, σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασή της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση Π.Ε με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσής της.

Ο Δηλών **Ο Δηλών** **Ημερομηνία**

Θα θέλαμε να ευχαριστήσουμε ιδιαίτερα τους καθηγητές μας· κύριο Παπουτσιδάκη Μιχάλη και κύριο Χατζόπουλο Μάκη για την πολύτιμη και πραγματικά ανεκτίμητη βοήθεια που μας προσέφεραν κατά την διάρκεια της εργασίας μας. Επίσης, θα θέλαμε να ευχαριστήσουμε τη Σοφία Λαλέα που μας βοήθησε, ως φιλόλογος, μετά τη συγγραφή της πτυχιακής μας, διορθώνοντάς μας εκφραστικά, ορθογραφικά και συντακτικά λάθη. Τέλος θα θέλαμε να ευχαριστήσουμε θερμά την Αναστασία Ευδαίμων που μας βοήθησε πάρα πολυ, χάρη στο ταλέντο της στη ζωγραφική και στο σχεδιασμό, με τα μηχανολογικά σχέδια του ρομποτικού αμαξιού μας (εικόνες 5 και 6).

ΠΕΡΙΛΗΨΗ

Το θέμα της παρούσης πτυχιακής εργασίας είναι η εύρεση και επίλυση της βέλτιστης διαδρομής ενός λαβυρίνθου, χρησιμοποιώντας ένα ρομποτικό αμαξάκι το οποίο είναι προγραμματισμένο με βάση το μικροελεγκτή Arduino. Σαν εργασία, η επίλυση ενός λαβυρίνθου μπορεί να έχει μεγάλη χρησιμότητα σε πολλές εφαργογές. Παράδειγμα εφαρμογής θα μπορούσε να είναι η πλοήγηση: ας πούμε ότι έχει πιάσει κάπου φωτιά και πρέπει ιπτάμενα μη επανδρωμένα αεροσκάφη (drones) να πάνε από την πιο γρήγορη διαδρομή για να τη σβήσουν. Άλλο παράδειγμα εφαρμογής θα μπορούσε να είναι η εξερεύνηση και πλοήγηση ταυτοχρόνως: ενδεχομένως σε αρχαιολογικές ανασκαφές σε σπήλαια. Θα μπορούσε να χρησιμοποιηθεί και απο το στρατό ή την πυροσβεστική για αποστολές εντοπισμού και διάσωσης. Όσον αφορά τη δική μας εργασία τώρα, ο λαβύρινθος δεν έχει κάποια έξοδο· στην ουσία ορίζεται μέσω του κώδικα μας με συντεταγμένες ένα αρχικό σημείο εκκίνησης, το οποίο είναι το ίδιο για κάθε περίπτωση, καθώς και ένα τελικό σημείο (όποιο θέλουμε εμείς) μέσα στο λαβύρινθο, στο οποίο το αμαξάκι θα πρέπει να καταλήξει και μάλιστα επιλέγοντας την ταχύτερη διαδρομή.

Η μέθοδος που εφαρμόστηκε για να λειτουργήσει σωστά ο κώδικας μας είναι η συνεργασία δύο αλγορίθμων, του Flood Fill [1] και του αλγόριθμου εύρεσης επόμενης κίνησης. Το Flood Fill ορίζει σε όλες τις θέσεις του λαβυρίνθου έναν αριθμό, που δείχνει την απόσταση του από τον τελικό του προορισμό (ο οποίος μετά από κάθε κίνηση του αυτοκινήτου αλλάζει κατάλληλα για το επόμενο βήμα), οπότε το όχημα γνωρίζει προς τα πού πρέπει να κατευθυνθεί και με τον αλγόριθμο εύρεσης επόμενης κίνησης εκτελεί τις κατάλληλες κινήσεις. Όμως για να λειτουργήσει σωστά αυτός ο αλγόριθμος πρέπει να γνωρίζουμε τον χάρτη του λαβυρίνθου, κάτι που στην αρχή της διαδικασίας δεν ισχύει. Άρα, αφού καταφέρει και χαρτογραφήσει όλους τους τοίχους που υπάρχουν στον λαβύρινθο, γυρίζει στην αρχική του θέση για να υπολογίσει, αλλά και να εκτελέσει την βέλτιστη διαδρομή από τα δύο σημεία.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Ρομπότ, μικροελεγκτής Arduino, Εκπαιδευτική πλατφόρμα, Επίλυση λαβυρίνθου

ABSTRACT

The theme of our dissertation is to solve a labyrinth with the optimal (faster) route using a robotic car that is programmed based on the Arduino microcontroller. In particular, the labyrinth does not have an exit, in fact we define through our code coordinates with an initial starting point which is the same for each instance and then choose an endpoint (whichever we want) within the labyrinth in which the robotic car should end up and even choose the fastest route. The method we applied for our code to function properly is the collaboration of two algorithms, the flood fill and the next move finder algorithm. The flood fill defines, in all the positions of the maze, a number indicating its distance from its final destination (which after each movement of the car changes appropriately for the next step), so the vehicle knows where to go and the next move finder algorithm performs these appropriate moves. However, for this algorithm to work properly, we need to know the labyrinth map, which is not the case at the beginning of the process. So after the robotic car maps all the walls within the maze, it returns to its original position to calculate and execute the optimal path of the two points.

KEYWORDS

Robot, Arduino, Educational platform, Maze Solving

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ

ΕΙΣΑΓΩΓΗ

Η πτυχιακή μας εργασία, όπως φαίνεται και απο τον τίτλο, είναι η επίλυση ενός λαβυρίνθου από ένα ρομποτικό αμαξάκι με την καλύτερη και πιο γρήγορη διαδρομή. Σαν εργασία είναι κατά βάση εκπαιδευτική, γιατί ερχόμαστε σε επαφή με διάφορα είδη μικροελεγκτών και μαθαίνουμε τον τρόπο με τον οποίο δουλεύον αλλά και πώς προγραμματίζονται για να δουλεύουν όπως θέλουμε εμείς: π.χ. η γλώσσα προγραμματισμού που χρησιμοποιείται καθώς και οι εντολές και οι βιβλιοθήκες που εφαρμόζονται στην εκάστοτε γλώσσα. Στην προκειμένη εργασία, όπως προαναφέραμε και στην περίληψη, χρησιμοποιούμε τον μικροελεγκτή Arduino που προγραμματίζεται σε γλώσσα C/C++. Επίσης μαθαίνουμε να φτιάχνουμε ηλεκτρονικά κυκλώματα, μαθαίνουμε τη λειτουργεία του κάθε εξαρτήματος (π.χ ένα "ολοκληρωμένο" τι είναι; ή τι είναι ένας αισθητήρας υπερύθρων;) και πειραματιζόμαστε με αυτα. Ασχολούμαστε επίσης και με ένα μηχανολογικό κομμάτι της όλης κατασκευής, τους κινητήρες, που πρέπει να γνωρίζουμε πώς λειτουργούν και ποιο είδος πρέπει να χρησιμοποιήσουμε κάθε φορά, αναλόγως το τι θέλουμε να φτιάξουμε. Στην προκειμένη πτυχιακή λοιπόν, η επίλυση του λαβυρίνθου έγινε με τη χρήση και τη συνεργασία δύο αλγορίθμων. Τον αλγόριθμο Flood Fill που τον χρησιμοποιήσαμε για την εύρεση της βέλτιστης θέσης, αλλά και σαν βοήθημα γενικά για τις κινήσεις που θα κάνει το ρομποτικό μας αμαξάκι, και τον αλγόριθμο εύρεσης επόμενης κίνησης, που τον χρησιμοποιούμε μέτα απο το Flood Fill, εφόσον δουλεύει με τα στοιχεία που θα παίρνει απο το Flood Fill, για να βρει τη βέλτιστη διαδρομή.

ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Ένα ρομποτικό αμαξάκι που επιλύει ένα λαβύρινθο σαν εργασία, για να δουλέψει σωστά, απαιτεί κατά βάση πολύ καλές γνώσεις προγραμματισμού. Πρέπει κάποιος να είναι πολύ καλά εξοικειωμένος με τις υπάρχουσες εκπαιδευτικές πλατφόρμες-μικροελεγκτές. Για παράδειγμα, στη συγκεκριμένη εργασία χρησιμοποιούμε Arduino. Ανάλογα με το μικροελεγκτή που χρησιμοποιούμε, κατ' επέκταση προγραμματίζουμε κατάλληλα· (π.χ. το Arduino προγραμματίζεται σε γλώσσα C/C++ , άρα χρειάζεται καλή γνώση της C/C++, ενώ αν χρησιμοποιούσαμε μικροελεγκτή Raspberry θα το προγραμματίζαμε σε γλώσσα μηχανής Python). Σε αυτό το κομμάτι επίσης, πρέπει να προσέξουμε τις διάφορες εκδόσεις γλωσσών προγραμματισμού και τις αναβαθμίσεις τους ή αναβαθμίσεις σε ήδη υπάρχουσες εκδόσεις που κυκλοφορούν. Αυτό συμβαίνει διότι, εντολές που μπορεί να χρησιμοποιούμε ή διάφορες βιβλιοθήκες που χρησιμοποιούμε, μπορεί να μην δουλεύουν πλέον σωστά ή να μην λειτουργούν στην εκδοση που χρησιμοποιούμε εμείς ή να ορίζονται με διαφορετικό τρόπο. Στον προγραμματισμό, στο μεγαλύτερο βαθμό των γλωσσών που υπάρχουν (αν όχι όλων), ο ορισμός διαφόρων εντολών και η σύνταξή τους παίζει πολύ μεγάλο ρόλο. Επίσης είναι πολύ χρήσιμο να χρησιμοποιούμε πάντα τις τελευταίες εκδόσεις βιβλιοθηκών και εντολών, γιατί είναι σχεδόν πάντα πιο εύκολες στη χρήση και μας διευκολύνουν περισσότερο γιατί είναι πιο λειτουργικές. Επιπρόσθετα, είναι απαραίτητες βασικές γνώσεις ηλεκτρολογικών και ηλεκτρονικών, αφού θα χρειαστεί να συνδέσουμε σωστά τα διάφορα εξαρτήματα που χρησιμοποιούμε. Ακόμα, είναι πολύ βασικό να γνωρίζουμε πώς να φτιάχνουμε ηλεκτρονικά κυκλώματα, διότι έτσι αποφεύγονται πολλά συνδεσμολογικά λάθη και είναι πολύ πιο εφικτό, αν έχουμε κάνει κάποιο λάθος, να το εντοπίσουμε πιο εύκολα και γρήγορα. Επίσης, γνωρίζοντας καλά ηλεκτρονικά, μπορούμε να καταλάβουμε καλύτερα κάθε εξάρτημα τι κάνει, ακόμη και αν δεν το έχουμε ξαναχρησιμοποιήσει ποτέ. Επιπλέον, μπορούμε να κατανοήσουμε κάθε εξάρτημα τι απαιτήσεις έχει: π.χ. πόσο ρεύμα και τάση καταναλώνει και πόσο από τα δύο χρειάζεται αντιστοίχως για να λειτουργήσει, χωρίς να το «κάψουμε». Η γνώση βασικών εννοιών και ενδείξεων ηλεκτρολογικών, (π.χ. A=ampere, V=Volt που είναι συμβολισμοί ρεύματος και τάσης), θα μας βοηθήσει να καταλάβουμε ποια θα πρέπει να είναι η συνδεσμολογία μας.

Ως αποτέλεσμα, γνωρίζοντας τα παραπάνω, θα πρέπει να είμαστε σε θέση να ξέρουμε να διαβάζουμε το εγχειρίδιο απο κάθε εξάρτημα, για να γνωρίζουμε τις απαιτήσεις του αλλά και τον τρόπο με τον οποίο δουλεύει. Ακόμα, θα χρειαστεί να κατέχουμε και κάποιες βασικές μηχανολογικές γνώσεις, κυρίως όσον αφορά το κομμάτι των κινητήρων. Είναι πολύ σημαντικό να γνωρίζουμε τα είδη των κιντήρων που θα χρησιμοποιήσουμε, αλλα και τον τρόπο με τον οποίο δουλεύουν. Στην προκειμένη περίπτωση, χρησιμοποιούμε DC κινητήρες σε συσκευασία σερβομηχανισμού, γιατί είναι πολύ πιο ακριβείς σε σύγκριση με τους απλούς DC κινητήρες, δουλεύουν καλύτερα και χωρίς καθυστερήσεις, καθώς είναι συνεχείς, συγκριτικά με τους απλούς Servo κινητήρες

που δουλεύουν με παλμό, (όπου μπορεί να έχουμε απώλεια σημάτων) και μπορούμε επίσης να ελέγξουμε πιο εύκολα την ταχύτητα τους με τη χρήση PWM.

Τέλος, απαραίτητη είναι και η γνώση υπολογιστικής νοημοσύνης, τόσο για την σύνταξη των αλγορίθμων που χρησιμοποιούνται από το όχημα, όσο και για την κατανόηση των κινήσεων που εκτελούνται κατά τη διάρκεια της διαδικασίας. Στην προκειμένη εργασία, χρησιμοποιήσαμε δύο αλγορίθμους σε συνδυασμό: τον αλγόριθμο Flood Fill και τον αλγόριθμο εύρεσης επόμενης κίνησης. Γενικά, κάποιοι άλλοι τρόποι επίλυσης είναι: ο ακολουθητής τοίχου ή αλλιώς κανόνας αριστερού ή δεξιού χεριού (Wall Follower [2]). Η λογική αυτού το αλγόριθμου στηρίζεται στο εξής: οι τοίχοι του λαβύρινθου πρέπει να είναι συνδεδεμένοι μεταξύ τους από την είσοδο μέχρι την έξοδο, έτσι ώστε να τους ακολουθεί το αμαξάκι και να είναι τοποθετημένοι με συγκεκριμένο τρόπο, είτε αριστερά είτε δεξιά, έτσι ώστε το αμαξάκι, αφού επιλέξει να στρίβει μόνο δεξιά ή αριστερά,(αναλόγως πώς έχει οριστεί απο τον κώδικά του), ακολουθώντας τους τοίχους, θα βρεί κάποια στιγμή την έξοδο. Στην δική μας περίπτωση, δεν ήταν εφικτή αυτή η μέθοδος, διότι οι τοίχοι μας δεν είναι συνδεδεμένοι μεταξύ τους (οπότε το αμαξάκι θα παγιδευόταν μέσα στο λαβύρινθο σε ατελείωτους κύκλους εξαιτίας τών κενών μεταξύ των τοίχων). Άλλωστε, θέλαμε την επίλυση του λαβυρίνθου με τη βέλτιστη διαδρομή. Άλλη μια μέθοδος είναι οι «τυχαίες κινήσεις» (Random Mouse / Robot algorithm [3]). Πρόκειται για έναν πολύ απλό αλγόριθμο, γιατί το αμαξάκι ακολουθεί συνεχώς τυχαίες διαδρομές, χωρίς να κρατάει μνήμη των διαδρομών που έχει ακολουθήσει, και επιλέγει πάντα τυχαίες κατευθύνσεις σε διασταυρώσεις που μπορεί να συναντήσει. Παρόλο που θα βρεί την έξοδο εν τέλει μετά από πολλά περάσματα, είναι πολύ χρονοβόρα διαδικασία και στην δική μας περίπτωση θέλαμε τη βέλτιστη και ταχύτερη διαδρομή.

ΜΕΘΟΔΟΛΟΓΙΑ

Όπως προαναφέραμε και στην περίληψη της πτυχιακής, οι δύο μέθοδοι που εφαρμόσαμε για την αντιμετώπιση του πρόβληματος, δηλαδή την επίλυση του λαβυρίνθου, είναι ο αλγόριθμος Flood Fill και ο αλγόριθμος εύρεσης επόμενης κίνησης. Αυτοί οι δύο αλγόριθμοι συνεργάζονται για να μπορεί κάθε φορά το όχημα να εκτελεί την σωστή κίνηση. Πριν προχωρήσουμε όμως, ας δούμε πρώτα πώς λειτουργεί ο κάθε αλγόριθμος ξεχωριστά. Η βασική λειτουργία του Flood Fill είναι να γεμίσει έναν πίνακα με τιμές, που αντιστοιχούν στην απόσταση του κάθε τετραγώνου του πίνακα από ένα τετράγωνο που του ορίζουμε εμείς (το ονομάζουμε θέση ενδιαφέροντος). Να αναφέρουμε πως κάθε τετράγωνο του πίνακα αντιστοιχεί σε ένα κελί του πραγματικού λαβυρίνθου. Παρακάτω ακολουθεί ένα απλό παράδειγμα (χωρίς εμπόδια) με έναν πίνακα 4x4. Διαλέγουμε σαν θέση ενδιαφέροντος το (3,3) και ο αλγόριθμος τοποθετεί σε αυτή την θέση την τιμή 0.

		0	

Πίνακας 1 - 1ο Παράδειγμα Flood Fill αριθμός 0

Επόμενο βήμα είναι να τοποθετηθούν οι αμέσως μεγαλύτερες τιμές από το 0 στα γειτονικά τετράγωνα. Οπότε θα έχουμε

		1	
	1	0	1
		1	

Πίνακας 2 - 1ο Παράδειγμα Flood Fill αριθμός 1

Και συνεχίζουμε με παρόμοιο τρόπο έως ότου γεμίσει πλήρως ο πίνακας με τιμές

		2	
	2	1	2
2	1	0	1
	2	1	2

Πίνακας 3 - 1ο Παράδειγμα Flood Fill αριθμός 2

	3	2	3
3	2	1	2
2	1	0	1
3	2	1	2

Πίνακας 4 - 1ο Παράδειγμα Flood Fill αριθμός 3

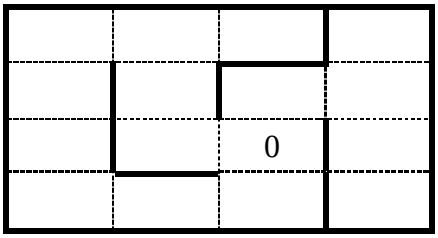
Καταλήγοντας στο

4	3	2	3
3	2	1	2
2	1	0	1
3	2	1	2

Πίνακας 5 - 1ο Παράδειγμα Flood Fill αριθμός 4

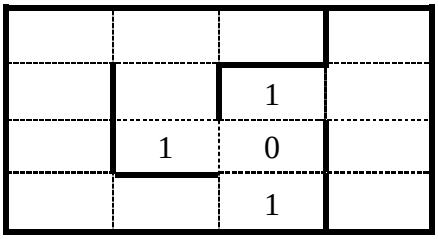
Οπότε πλέον γνωρίζουμε την απόσταση όλων των τετραγώνων από την θέση ενδιαφέροντος (3,3).

Όμως, σε ένα πρακτικό πρόβλημα θα υπάρχουν και εμπόδια ανάμεσα στα τετράγωνα (τοίχοι), οι οποίοι αλλάζουν την πολυπλοκότητα του αλγορίθμου. Ακολουθεί και ένα παράδειγμα με τοίχους, πάλι όμως σε πίνακα 4x4 με θέση ενδιαφέροντος το σημείο (3,3). (Με έντονη μαύρη γραμμή έχουμε τοίχο, ενώ με διακεκομμένη γραμμή ελεύθερο πέρασμα).



Πίνακας 6 - 2ο παράδειγμα Flood Fill αριθμός 0

Αυτή την φορά ο αλγόριθμος ‘‘βλέπει’’ τους τοίχους και τοποθετεί ανάλογα τους αριθμούς.



Πίνακας 7 - 2ο παράδειγμα Flood Fill αριθμός 1

Παρατηρούμε πως αυτή την φορά δεν προστέθηκε τιμή στο τετράγωνο δεξιά εξαιτίας του τοίχου που υπάρχει ανάμεσα. Και ο αλγόριθμος συνεχίζει να προσθέτει τιμές.

	2	1	2
	1	0	
	2	1	

Πίνακας 8 - 2ο παράδειγμα Flood Fill αριθμός 2

	3		3
	2	1	2
	1	0	3
3	2	1	

Πίνακας 9 - 2ο παράδειγμα Flood Fill αριθμός 3

4	3	4	3
	2	1	2
4	1	0	3
3	2	1	4

Πίνακας 10 - 2ο παράδειγμα Flood Fill αριθμός 4

Και καταλήγουμε στο

4	3	4	3
5	2	1	2
4	1	0	3
3	2	1	4

Πίνακας 11 - 2ο παράδειγμα Flood Fill αριθμός 5

Οπότε βλέπουμε ότι η ύπαρξη τοίχων “μεγαλώνει” τις αποστάσεις των τετραγώνων και κάνει πιο πολύπλοκο τον λαβύρινθο.

Να σημειωθεί πως, σαν μοναδική συνθήκη στην δημιουργία του λαβυρίνθου, είχαμε πει ότι θα έπρεπε και τα 49 τετράγωνα να είναι προσβάσιμα τουλάχιστον από μία πλευρά.

Τώρα που είδαμε τον αλγόριθμο Flood Fill, ας προχωρήσουμε και στον επόμενο, που είναι ο αλγόριθμος εύρεσης επόμενης κίνησης. Για να λειτουργήσει σωστά, χρειάζεται τα στοιχεία από τον Flood Fill, οπότε θα χρησιμοποιήσουμε τον τελικό πίνακα από το προηγούμενο παράδειγμα. Σε αυτό το σενάριο έχουμε το όχημα μας στην θέση (4,1) και θέλουμε να βρούμε την αλληλουχία κινήσεων για να φτάσουμε στο (3,3). (Με βελάκι συμβολίζεται η θέση και κατεύθυνση του οχήματος)

4	3	4	3
5	2	1	2
4	1	0	3
↑	2	1	4

Πίνακας 12 - Παράδειγμα Εύρεσης Επόμενης Κίνησης αρχική θέση

Το όχημα ‘‘διαβάζει’’ τις τιμές των τεσσάρων γειτονικών τετραγώνων. Σε περίπτωση που υπάρχει τοίχος σε κάποια πλευρά, τότε δίνει σαν τιμή τετραγώνου έναν μεγάλο αριθμό για να μην επηρρεάσει τις υπόλοιπες τιμές και διαλέγει το τετράγωνο με την μικρότερη τιμή σαν επόμενη θέση του. Οπότε στην περίπτωση μας έχει να διαλέξει μεταξύ του πάνω τετραγώνου (με τιμή 4) και του δεξιά (με τιμή 2). Διαλέγει το τετράγωνο δεξιά, γιατί έχει την μικρότερη τιμή, αλλά πριν καταφέρει να μεταφερθεί σωστά, πρέπει να λάβει υπόψιν και την τρέχουσα κατεύθυνση του. Στο παράδειγμα το όχημα “κοιτάει” βόρεια, άρα θα πρέπει πρώτα να στρίψει δεξιά και μετά να πραγματοποιήσει ευθεία κίνηση. Σε άλλη περίπτωση θα μπορούσε να ήταν στραμμένο ανατολικά, οπότε να ήταν άσκοπη η δεξιά στροφή και να χρειαζόταν μόνο η ευθεία κίνηση.

Γενικά, το όχημα πραγματοποιεί τις εξής κινήσεις: Ευθεία κίνηση 30 εκατοστά, δεξιά στροφή 90^o, αριστερή στροφή 90^o και αναστροφή 180^o.

Οπότε αφού εκτελέσει τις δύο κινήσεις θα βρεθεί στην παρακάτω θέση.

4	3	4	3
5	2	1	2
4	1	0	3
3	→	1	4

Πίνακας 13 - Παράδειγμα Εύρεσης Επόμενης Κίνησης 1η κίνηση

Με παρόμοιο τρόπο θα ψάξει να βρει την επόμενη κίνησή του. Αυτή την φορά έχει να επιλέξει μεταξύ του αριστερού τετραγώνου, από όπου ήρθε (με τιμή 3) και του δεξιού (με τιμή 1). Αφού επιλέξει το δεξί και δει και την κατεύθυνση του, θα αποφασίσει απλά να κάνει ευθεία κίνηση μιας και η επόμενη θέση είναι στην πορεία της κατεύθυνσής του. Έτσι βρισκόμαστε σε αυτή την θέση.

4	3	4	3
5	2	1	2
4	1	0	3
3	2	→	4

Πίνακας 14 - Παράδειγμα Εύρεσης Επόμενης Κίνησης 2η κίνηση

Πλέον έχει φτάσει ένα τετράγωνο μακριά από την τελική θέση, την οποία και επιλέγει σαν επόμενη κίνηση. Βάσει της θέσης του, θα πρέπει να εκτελέσει πρώτα αριστερή στροφή και μετά ευθεία κίνηση καταλήγοντας στην τελική θέση.

4	3	4	3
5	2	1	2
4	1	↑	3
3	2	1	4

Πίνακας 15 - Παράδειγμα Εύρεσης Επόμενης Κίνησης τελική θέση

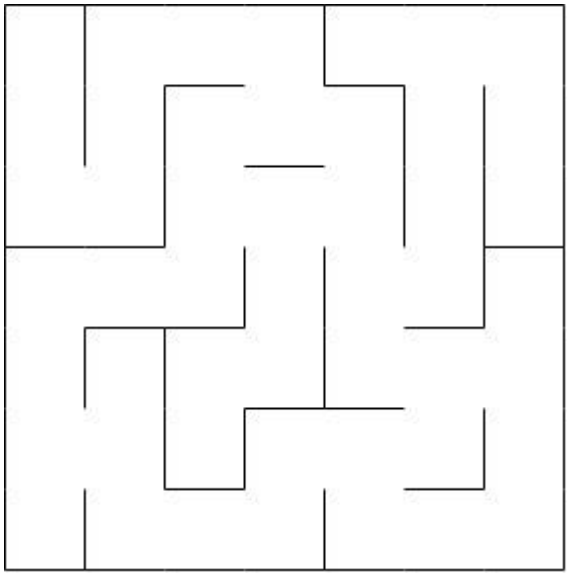
Τα συγκεκριμένα παραδείγματα είναι απλά, αλλά αντικατοπτρίζουν πλήρως την λογική που χρησιμοποιείται στην κύρια διαδικασία μας.

Παρακάτω ακολουθεί παράδειγμα με την χρήση της μεθόδου στον πραγματικό λαβύρινθο.

Στην αρχή της διαδικασίας, το όχημα έχει στην μνήμη του σημειωμένους μόνο τους εξωτερικούς τοίχους του λαβυρίνθου, με τις εσωτερικές θέσεις να είναι κενές. Αυτές θα γεμίζουν κατά την διάρκεια της χαρτογράφησης με δεδομένα από τα αισθητήρια του οχήματος. Τα αισθητήρια διαβάζονται κάθε φορά πριν εκτελεστεί μία κίνηση από το όχημα. Επίσης, το όχημα είναι προγραμματισμένο να περάσει τουλάχιστον μία φορά από το κάθε τετράγωνο του λαβυρίνθου, αρχίζοντας να μετράει από την πάνω αριστερή γωνία (1,1) και καταλλήγοντας στην κάτω δεξιά (7,7) σημειώνοντας παράλληλα και στην μνήμη του από ποιές θέσεις έχει ήδη περάσει για να αποφύγει περιττές κινήσεις. Όταν πέρασει και από τις 49 θέσεις και έχει χαρτογραφήσει πλήρως τον λαβύρινθο, τελειώνει το πρώτο και μεγαλύτερο στάδιο της διαδικασίας. Στο δεύτερο στάδιο θα πρέπει να επιστρέψει από την τρέχουσα θέση του στην αρχική. Αυτό επιτυγχάνεται με την χρήση του Flood Fill μία φορά, έχοντας σαν θέση ενδιαφέροντος το (7,1) που είναι η αρχική θέση, καθώς και με την εκτέλεση του αλγορίθμου εύρεσης επόμενης κίνησης, τόσες φορές όση είναι και η απόσταση του οχήματος από την θέση ενδιαφέροντος. Όταν καταφέρει και επιστρέψει, τελειώνει και το δεύτερο στάδιο και μπαίνουμε πλέον στο τελευταίο και σημαντικότερο που είναι η εκτέλεση της βέλτιστης διαδρομής από το σημείο (7,1) μέχρι την τελική θέση. Πάλι το όχημα θα χρησιμοποιήσει τον Flood Fill για να βρει την διαδρομή και θα εκτελέσει τις κινήσεις με την βοήθεια της εύρεσης επόμενης κίνησης.

Παρακάτω ακολουθούν αναλυτικότερα τα βασικά βήματα της επίλυσης της διαδικασίας.

Αρχικά δίνεται η απεικόνιση/κάτοψη του λαβυρίνθου μας.



Εικόνα 1 - Λαβύρινθος

Μετά, προκειμένου να επιλυθεί, μετασχηματίζεται σε έναν πίνακα 7x7, όπου κάθε κελί αντιστοιχεί σε ένα τετράγωνο του λαβυρίνθου. Και έτσι έχουμε τον παρακάτω πίνακα (όπου έχουμε ↑ είναι η θέση του οχήματος).

↑						

Πίνακας 16 - Κύρια δαδικασία αρχική θέση

Παρατηρούμε πως στον χάρτη μας δεν έχουμε στοιχεία, γιατί ακόμα δεν έχει γίνει χαρτογράφηση. Πρώτη δουλειά που κάνει το όχημα, είναι να σαρώσει με τα τρία αισθητήρια που διαθέτει το περιβάλλον του, για την ύπαρξη τοίχων γύρω από το τετράγωνο που βρίσκεται. Οπότε θα έχουμε αυτό:

↑						

Πίνακας 17 - Κύρια διαδικασία 1η σάρωση

Μετά την σάρωση, ακολουθεί η εκτέλεση κάποιας κίνησης. Αυτή ορίζεται από δύο εμφωλευμένες for loop που έχουμε στον κώδικα. Συγκεκριμένα πρέπει να ξεκινήσει να διαβάζει από την θέση (1,1) και να καταλήξει στην (7,7), αγνοώντας όμως όποιες θέσεις έχει ήδη περάσει από το αντίστοιχο τετράγωνο σε κάποια προηγούμενη διαδρομή. Άρα αυτή την στιγμή πρέπει να πάει το όχημα στην θέση (1,1), οπότε εκτελώντας τον Flood Fill με θέση ενδιαφέροντος το (1,1) έχουμε τον ακόλουθο πίνακα

0	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
↑	7	8	9	10	11	12

Πίνακας 18 - Κύρια διαδικασία 1ο FloodFill

Βάσει του αλγορίθμου εύρεσης επόμενης κίνησης, το όχημα θα κινηθεί προς τα πάνω και θα εκτελέσει την επόμενη σάρωση.

0	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
↑	6	7	8	9	10	11
6	7	8	9	10	11	12

Πίνακας 19 - Κύρια διαδικασία 1η κίνηση

Αφού δεν αλλάζουν τα δεδομένα του χάρτη, το όχημα θα συνεχίσει τις κινήσεις του σαρώνοντας παράλληλα σε κάθε νέα θέση που βρίσκεται.

0	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
↑	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Πίνακας 20 - Κύρια διαδικασία 3ο FloodFill

Προσπερνώντας κάποια βήματα βρισκόμαστε πλέον στην θέση (1,1)

↑	5	6	7	10	11	12
1	4	9	8	9	10	11
2	3	10	11	10	11	12
13	12	11	12	11	12	13
14	15	14	13	12	13	14
15	16	15	14	13	14	15
16	17	16	15	14	15	16

Πίνακας 21 - Κύρια διαδικασία θέση (1,1)

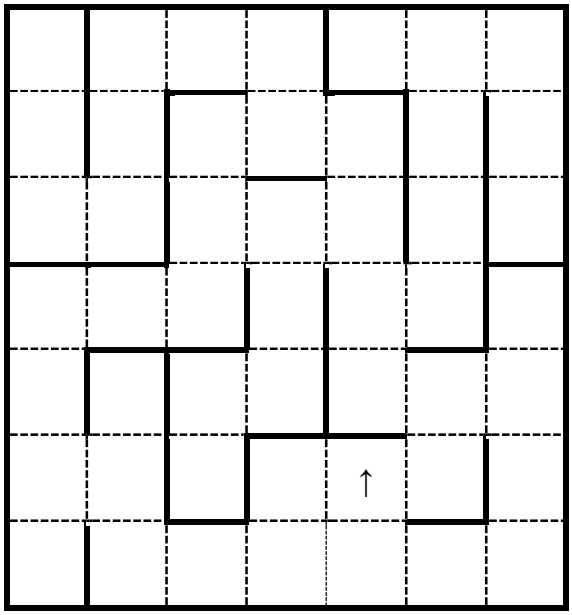
Βλέπουμε πως ο λαβύρινθος έχει αρχίσει να σχηματίζεται και στον χάρτη μας. Επόμενο βήμα είναι να πάει το όχημα στην θέση (1,2), αλλά έχει ήδη περάσει από την συγκεκριμένη θέση, οπότε την αγνοεί, όπως και τις (1,3) και (1,4). Με στόχο πλέον την (1,5) εκτελεί Flood Fill με θέση ενδιαφέροντος την (1,5).

↑	5	4	3	0	1	2
9	6	3	2	1	2	3
8	7	4	3	2	3	4
7	6	5	4	3	4	5
8	7	6	5	4	5	6
9	8	7	6	5	6	7
10	9	8	7	6	7	8

Πίνακας 22 - Κύρια διαδικασία θέση (1,1) FloodFill

Το όχημα υπολογίζει την ταχύτερη διαδρομή και την εκτελεί μέχρι το (1,5).

Στο τέλος θα πρέπει η εικόνα του χάρτη μας να είναι ίδια με τον πραγματικό λαβύρινθο



Πίνακας 23 - Κύρια διαδικασία τέλος 1ου σταδίου

Εδώ τελειώνει το πρώτο στάδιο και το όχημα περνάει στο δεύτερο: πρέπει δηλαδή να επιστρέψει στην αρχική θέση, οπότε εκτελεί Flood Fill με θέση ενδιαφέροντος το (7,1) με το παρακάτω αποτέλεσμα.

16	11	10	9	14	13	14
15	12	7	8	9	12	15
14	13	6	7	8	11	16
3	4	5	8	9	10	11
2	3	10	9	10	9	10
1	2	11	6	↑	8	9
0	3	4	5	6	7	8

Πίνακας 24 - Κύρια διαδικασία FloodFill για επιστροφή στην αρχική θέση

Άρα το όχημα έχει βρει την διαδρομή που πρέπει να εκτελέσει και την ακολουθεί για να τελειώσει και το δεύτερο στάδιο.

Στο τελικό στάδιο, θα πρέπει να μετακινηθεί από την αρχική του θέση (7,1) στην τελική (5,6). Οπότε εκτελεί για μια τελευταία φορά τον Flood Fill με θέση ενδιαφέροντος το (5,6).

13	8	7	6	7	6	7
12	9	6	5	4	5	8
11	10	5	4	3	4	9
8	7	6	5	2	3	2
9	8	7	6	1	0	1
8	7	8	3	2	1	2
↑	6	5	4	3	4	3

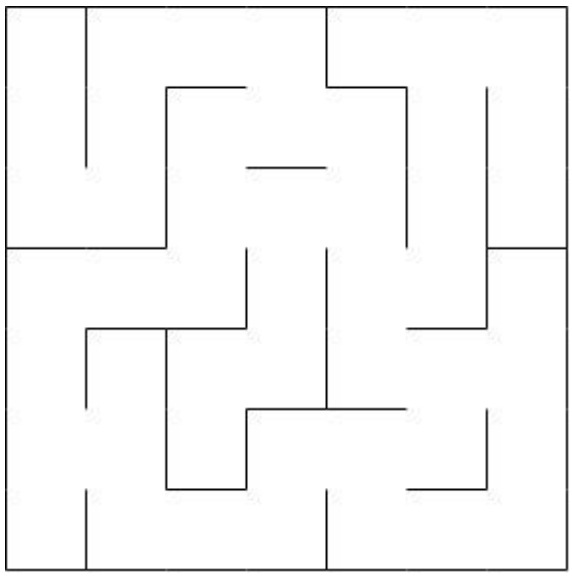
Πίνακας 25 - Κύρια διαδικασία τελικό στάδιο

Πλέον είναι εύκολο να υπολογίσει και να εκτελέσει την βέλτιστη διαδρομή, κάτι που θα φέρει την διαδικασία μας εις πέρας.

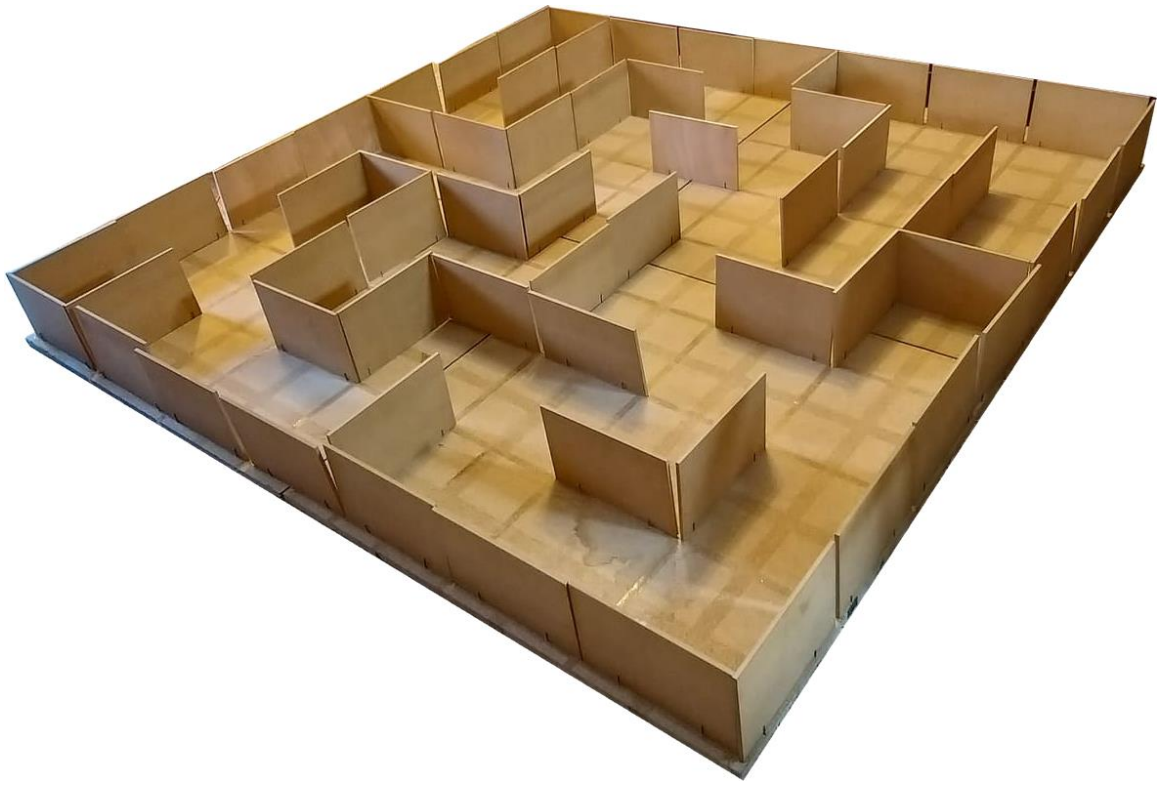
ΣΧΕΔΙΑΣΗ ΚΑΙ ΚΑΤΑΣΚΕΥΗ ΥΛΙΚΟΥ ΜΕΡΟΥΣ (HARDWARE)

Λειτουργικά Μέρη Κατασκευής

Ο λαβύρινθος αποτελείται από τέσσερα μεγάλα ξύλα σε σχήμα τετραγώνου, τύπου νοβοπάν και με διαστάσεις 110x110 εκατοστά το καθένα, τα οποία αποτελούν τη βάση του λαβυρίνθου μας (σύνολο 220x220 εκατοστά). Στα εσωτερικά μήκη κάθε ξύλου που συνυπάρχουν μεταξύ τους, για να δημιουργήσουν ένα μεγάλο ενιαίο τετραγάγωνο (βάση), έχουμε ανοίξει τρύπες με τρυπάνι, αντιστοιχώντας τη μία με την άλλη, (συνολο:16) και έχουμε τοποθετήσει δύο μικρά και μακρόστενα ξυλάκια που ονομάζονται καβίλιες σε κάθε τετράδα τρυπών που ανοίξαμε (σύνολο 8). Επίσης χρησιμοποιήσαμε 25 μικρά ξύλα, τύπου κόντρα πλακέ θαλάσσης, με διαστάσεις 20x30 εκατοστά, που αναπαριστούν τους τοίχους του λαβύρινθου μας και τοποθετήθηκαν με βάση μια τετράδα καρφιών ανα ξύλο.



Εικόνα 2 - Σχέδιο Λαβυρίνθου



Εικόνα 3 - Διαγώνια όψη λαβυρίνθου

Εικόνα 4 - Εμπρόσθια όψη λαβυρίνθου

Το ρομποτικό αυτοκίνητο το συναρμολογήσαμε έχοντας αγοράσει τα υλικά του μέρη και τοποθετώντας πάνω του τον μικροελεγκτή μας (Arduino) και ενα breadboard. Επίσης, τοποθετήσαμε πάνω στο αμαξάκι μας δύο DC κινητήρες σε συσκευασία σερβομηχανισμού και στα άκρα τους δύο ρόδες. Ακόμα χρησιμοποιήσαμε και τρία υπέρηχα αισθητήρια περιμετρικά, (μπροστά, δεξιά και αριστερά) για να μετράμε αποστάσεις του αμαξιού απο τους τοίχους που το περιβάλλουν, για αποφυγή συγκρούσεων αλλά και αναγνώριση του λαβυρίνθου. Επιπρόσθετα, πάνω στο breadboard εχουμε τοποθετήσει ενα ολοκληρωμένο που ονομάζεται L293D 1A και το χρησιμοποιούμε για να ελέγξουμε τους δύο κινητήρες. Όλα αυτά συνδέονται μέσω ειδικών καλωδίων (jumpers) στο breadboard και το breadboard συνδέεται μέσω του ίδιου τύπου καλωδίων με το Arduino από όπου τροφοδοτείται και κατ' επέκταση τροφοδοτεί και όλα τα υπόλοιπα υλικά που προαναφέραμε.

Μηχανολογικό σχέδιο

Η μηχανολογική κατασκευή του ρομποτικού μας αμαξιού αποτελείται από μία πλατφόρμα αλουμινίου (ως κύριο σώμα) χρώματος κόκκινου που φέρει:

- δύο ανεξάρτητες υποδοχές για τροχούς, έτσι ώστε να είναι πιο εύκολη η τοποθέτηση τους πάνω στο όχημα. Αυτοί οι τροχοί είναι DC κινητήρες σε συσκευασία σερβομηχανισμού, έτσι ώστε το όχημα να πραγματοποιεί διαφορική κίνηση με ακρίβεια (βλέπε Εικόνα 6)

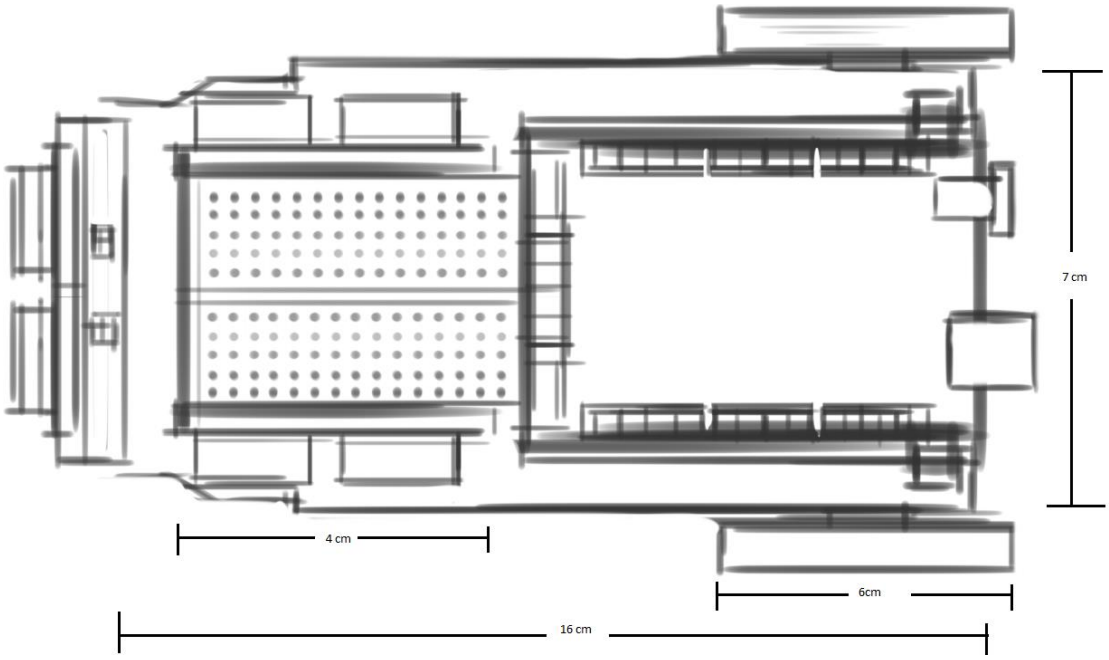
- Μια μεταλλική στρογγυλή μπίλια στο μπροστινό μέρος του αμαξιού, όπου χρησιμοποιείται μόνο για υποβοήθεια και την στήριξη (βλέπε Εικόνα 6) του οχήματος.

- Μια πλακέτα γενικής χρήσης breadboard, όπου πραγματοποιήσαμε τις συνδεσμολογίες μας (βλέπε Εικόνα 5 και 6)

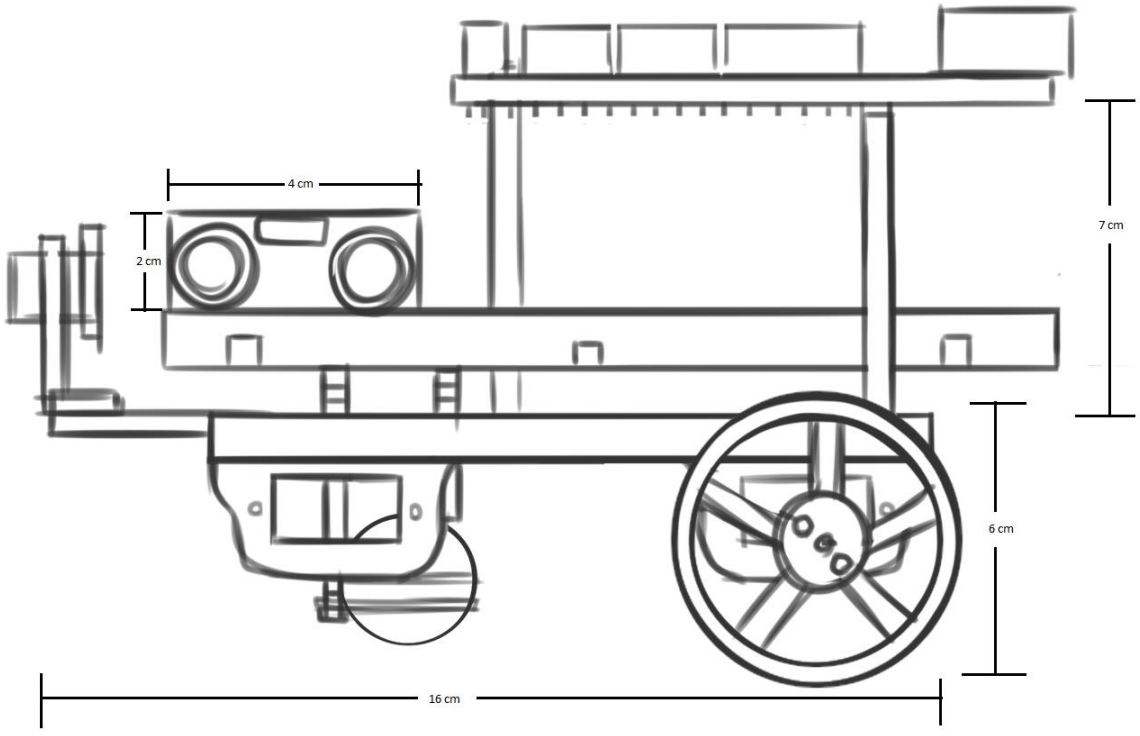
- Ένα ολοκληρωμένο οδηγό L293D που χρησιμοποιήθηκε για την ρύθμιση και τον έλεγχο των ταχυτήτων των κινητήρων μας.

- Τον μικροελεγκτή Arduino Mega 2560 όπου στην ουσία είναι ο ‘‘εγκέφαλος’’ του ρομποτικού μας αμαξιού, όπου βρίσκεται αποθηκευμένος ο κώδικας μας, και γενικά ελέγχουμε ολόκληρο το ρομποτικό αμαξάκι (βλέπε Εικόνα 5 και 6)

- Τρία υπέρηχα αισθητήρια, όπου μας βοηθάνε στην χαρτογράφηση του λαβυρίνθου κυρίως και φυσικά για την αποφυγή συγκρούσεων του αμαξιού με τους γύρω τοίχους που το περιβάλλουν (βλέπε Εικόνα 5 και 6).



Εικόνα 5 - Κάτοψη μηχανολογικού σχέδιο οχήματος

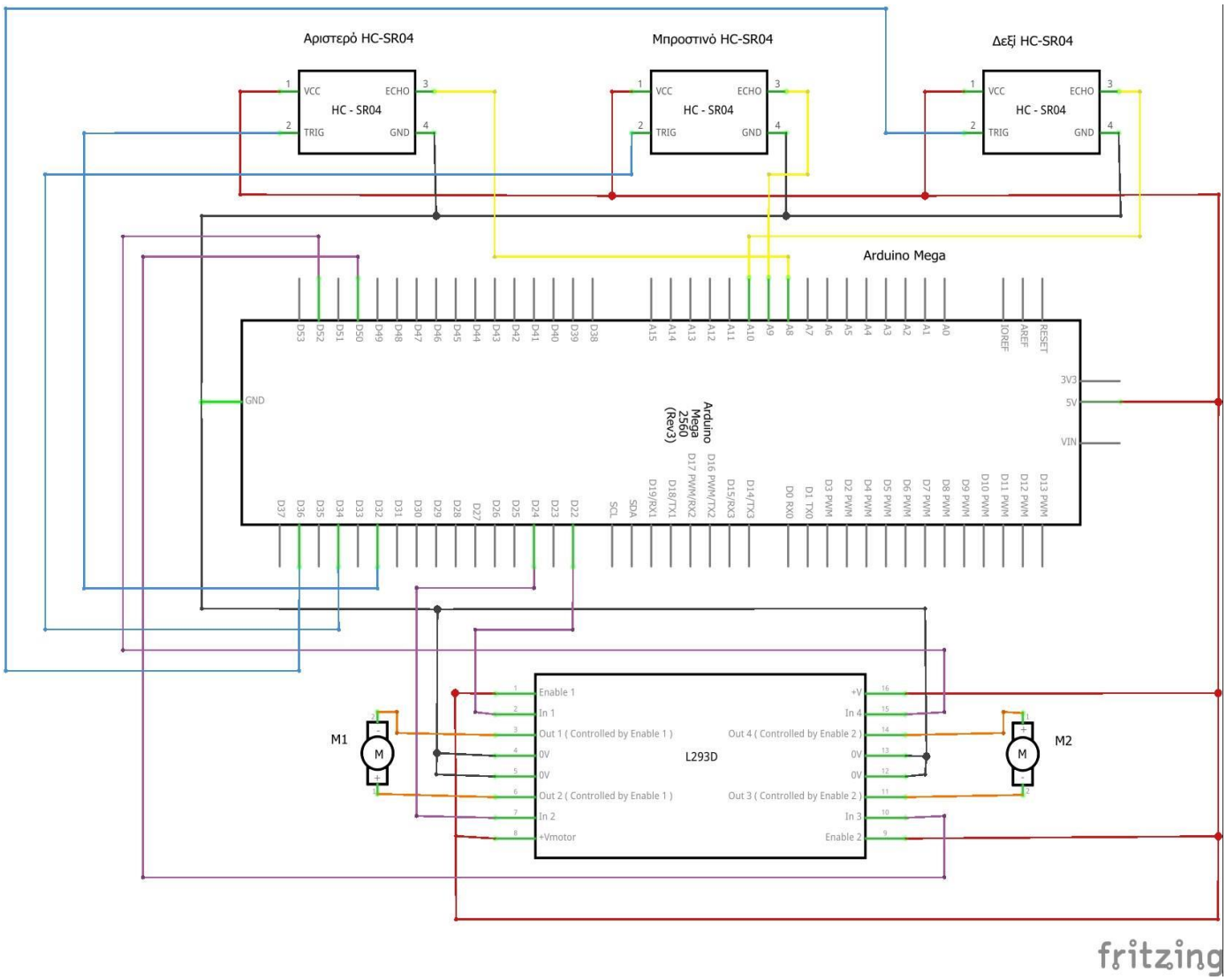


Εικόνα 6 - Αριστερή όψη μηχανολογικού σχεδίου οχήματος

Ηλεκτρονικό κύκλωμα

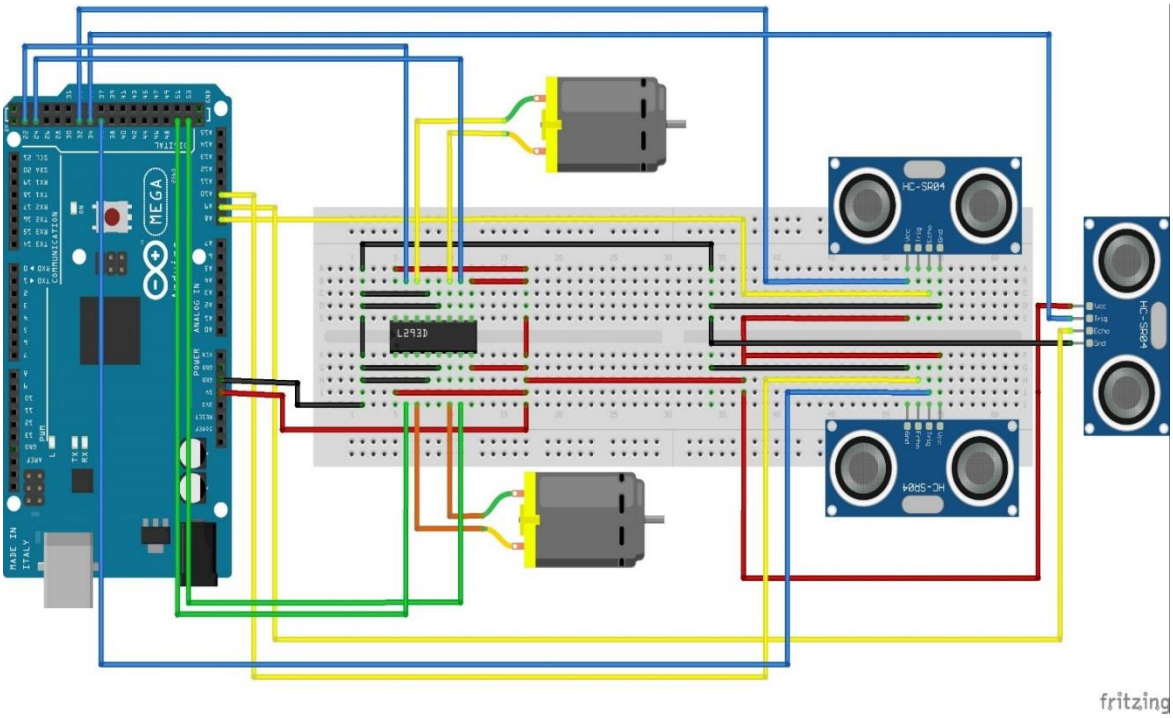
Ο έλεγχος του ρομποτικού μας αμαξιού υλοποιείται από το παρακάτω ηλεκτρονικό κύκλωμα ελέγχου. Κύριο βασικό εξάρτημα του είναι ο μικροελεγκτής Arduino Mega (βλέπε Εικόνα 9). Ο μικροελεγκτής είναι υπεύθυνος για την συλλογή δεδομένων, δηλαδή τις μετρήσεις των αισθητηρίων από το περιβάλλον και την οδήγηση των ενεργοποιητών (κινητήρων) του ρομποτικού μας αμαξιού μέσω του ολοκληρωμένου L293D. Η τροφοδοσία όλων των εξαρτημάτων γίνεται από την θύρα των 5V του Arduino. Οι ψηφιακές θύρες του Arduino στέλνουν τα σήματα για τον έλεγχο των κινητήρω στον οδηγό, και από εκεί συνδέονται με τους κινητήρες για την ενεργοποίηση τους. Τέλος για την επικοινωνία με τους αισθητήρες χρησιμοποιούνται οι αναλογικές είσοδοι για το Echo και οι ψηφιακές έξοδοι για το Trigger.

Στην παρακάτω εικόνα φαίνεται αναλυτικά το ηλεκτρονικό κύκλωμα του οχήματος



Εικόνα 7 - Ηλεκτρονικό κύκλωμα [9]

Και εδώ φαίνεται η συνδεσμολογία με τα πραγματικά εξαρτήματα για την καλύτερη κατανόηση του κυκλώματος.



Εικόνα 8 - Σχέδιο συνδεσμολογίας [9]

Λίστα υλικών

Τα υλικά που χρησιμοποιήσαμε για την πτυχιακή μας εργασία είναι τα εξής:

*Στις παρενθέσεις αναγράφεται η ποσότητα των συγκεκριμένων υλικών που χρησιμοποιήσαμε για την κατασκευή μας.

Μικροελεγκτής Arduino Mega 2560 (1)

Εικόνα 9 - Arduino Mega 2560 [4]

Ηλεκτρικά χαρακτηριστικά [10]:

- Τάση λειτουργίας: 5 VDC
- Τάση εισόδου 7-12 V προτεινόμενη
- Τάση εισόδου 6-20 V (min-max ορίου)
- Ψηφιακές είσοδοι/έξοδοι 54
- PWM Ψηφιακές είσοδοι/έξοδοι 16
- Αναλογικές είσοδοι 16
- Ρεύμα ανα είσοδο/έξοδο 20mA
- Ρεύμα ανα είσοδο/έξοδο 3.3V 50mA
- Μνήμη Flash 256 KB, από τα οποία 8 KB χρησιμοποιούνται για το σύστημα
- Μνήμη SRAM 8 KB
- Μνήμη EEPROM 4 KB
- Ταχύτητα 16 MHz

DC Motors FM90 (2)

Εικόνα 10 - DC κινητήρας σε θήκη Servo [5]

Ηλεκτρικά χαρακτηριστικά [11]:

- Τάση τροφοδοσίας 4-6 V
- Ταχύτητα χωρίς φορτίο
  110RPM (4.8v)
  130RPM (6v)
- Διερχόμενο ρεύμα (χωρίς φορτίο):
  100mA (4.8v)
  120mA (6v)
- Μέγιστη ροπή (4.8v): 1.3 kg.cm / 18.09 oz.in
- Μέγιστη ροπή (6v): 1.5 kg.cm / 20.86 oz.in
- Στατικό ρεύμα :
  550mA (4.8v)
  650mA (6v)

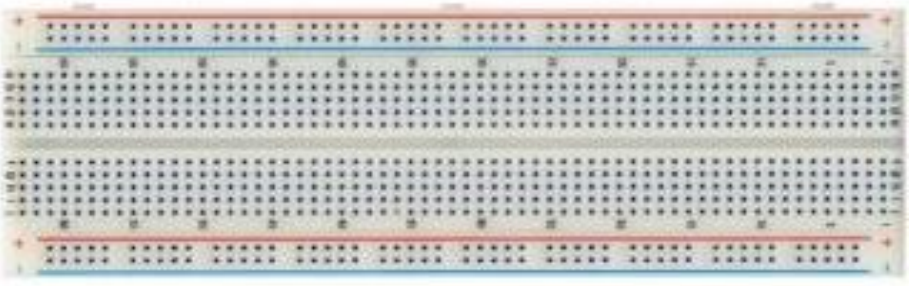
Αισθητήρες Υπερήχων (HC-SR04) (3)

Εικόνα 11 - Αισθητήριο HC-SR04 [6]

Ηλεκτρικά χαρακτηριστικά [12]:

- Τάση λειτουργίας – 5V
- Ρεύμα λειτουργίας –15mA
- Διασύνδεση – Ψηφιακή
  - Trigger pin: παλμός εισόδου
  - Echo pin: παλμός εξόδου
- Χωρίς πρωτόκολλο επικοινωνίας

Breadboard

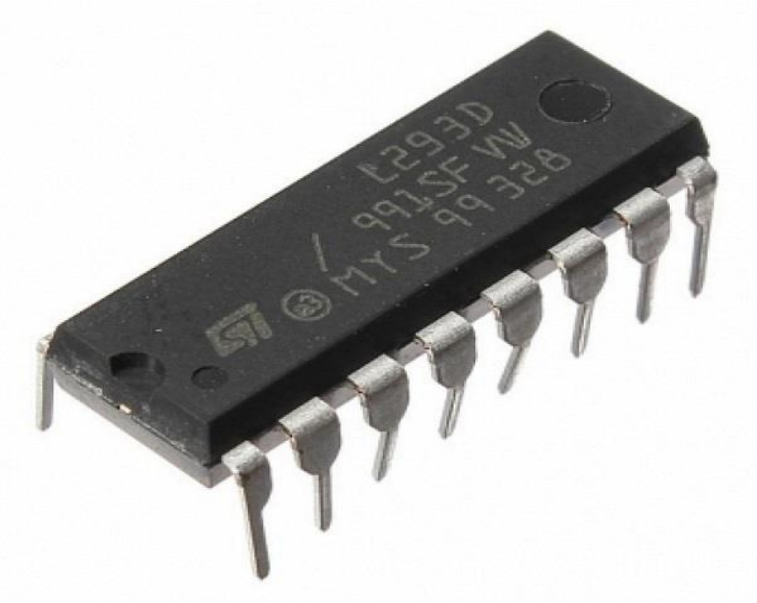


Εικόνα 12 – Breadboard [7]

Ηλεκτρικά χαρακτηριστικά [13]:

- Αριθμός pin: 630
- Μέγιστη Τάση: 300V
- Μέγιστη Ένταση: 3-5A

Motor Driver L293D 1A (1)



Εικόνα 13 - L2935 1A [8]

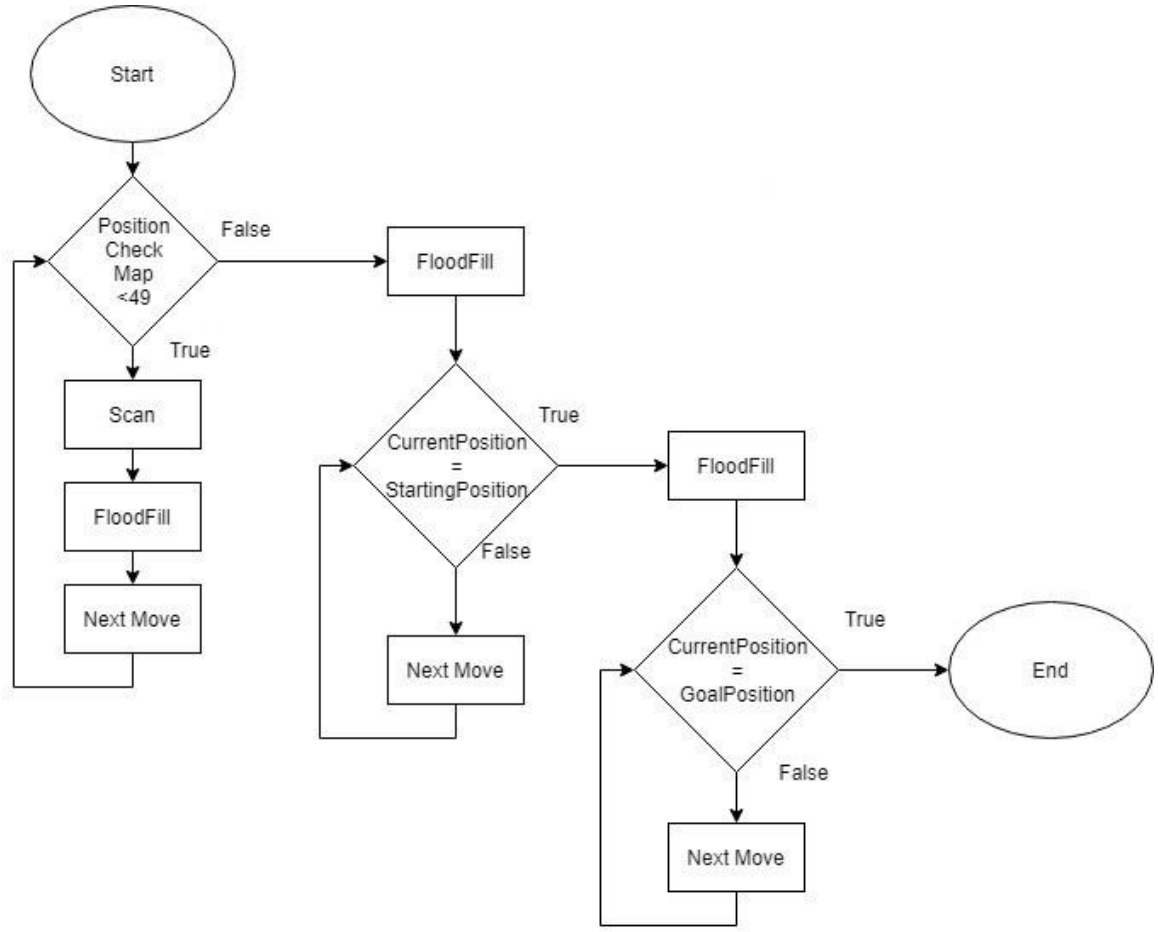
Ηλεκτρικά χαρακτηριστικά [14]:

- Εύρος τάσης : 4.5 V to 36 V
- Ξεχωριστές είσοδοι για τροφοδοσία λογικών καναλιών
- Εσωτερική ESD προστασία
- Προστασία από υπερβολική θερμότητα
- Ρεύμα εξόδου 1 A για κάθε κανάλι (600 mA for L293D)
- Μέγιστο ρεύμα εξόδου 2 A για κάθε κανάλι (1.2 A for L293D)
- Εξωτερικές θύρες διόδου για επαγωγιμότητα

ΑΝΑΛΥΣΗ ΚΑΙ ΣΥΓΓΡΑΦΗ ΛΟΓΙΣΜΙΚΟΥ (SOFTWARE)

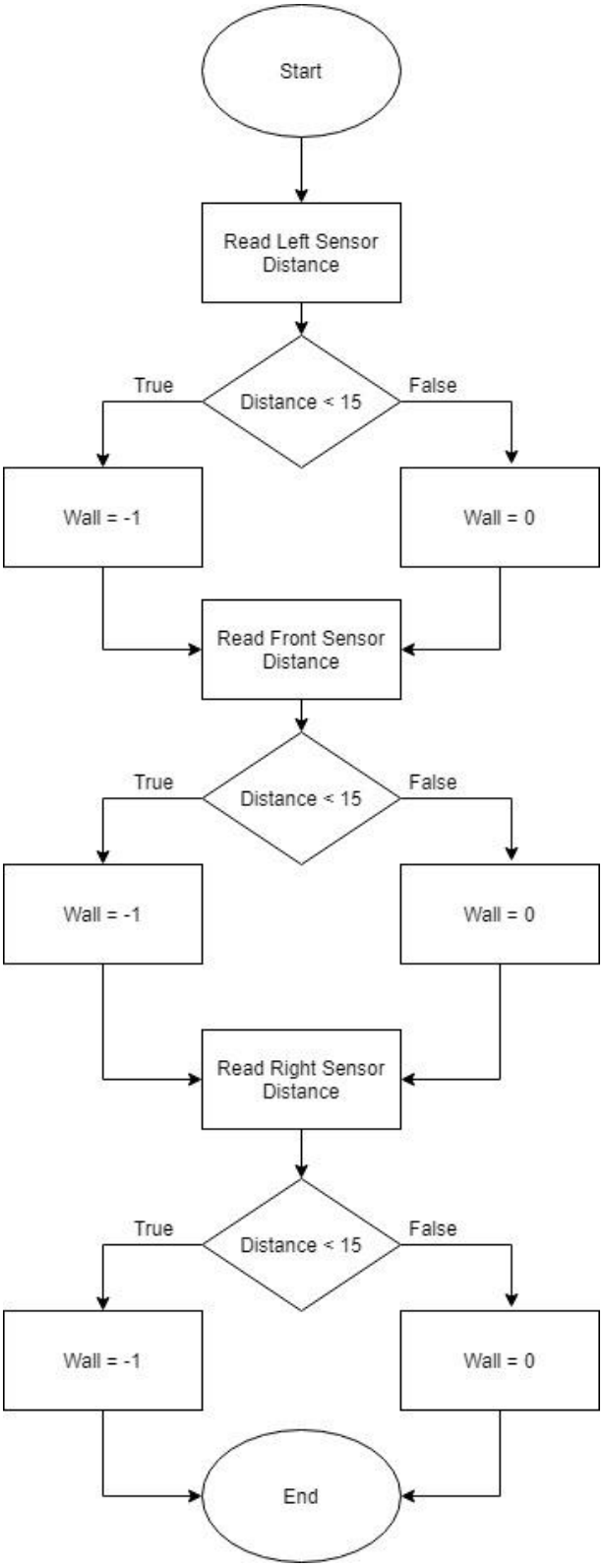
Τόσο η κίνηση του ρομπότ όσο και η συνολική «ευφυΐα» που είναι απαραίτητη για την επίλυση του λαβύρινθου υλοποιείται στο λογισμικό που έχουμε συγγράψει για το μικροελεγκτή Arduino. Στο παρακάτω σχήμα παρουσιάζεται το διάγραμμα ροής του βασικού κώδικα

Διάγραμμα Ροής (Flow Chart)

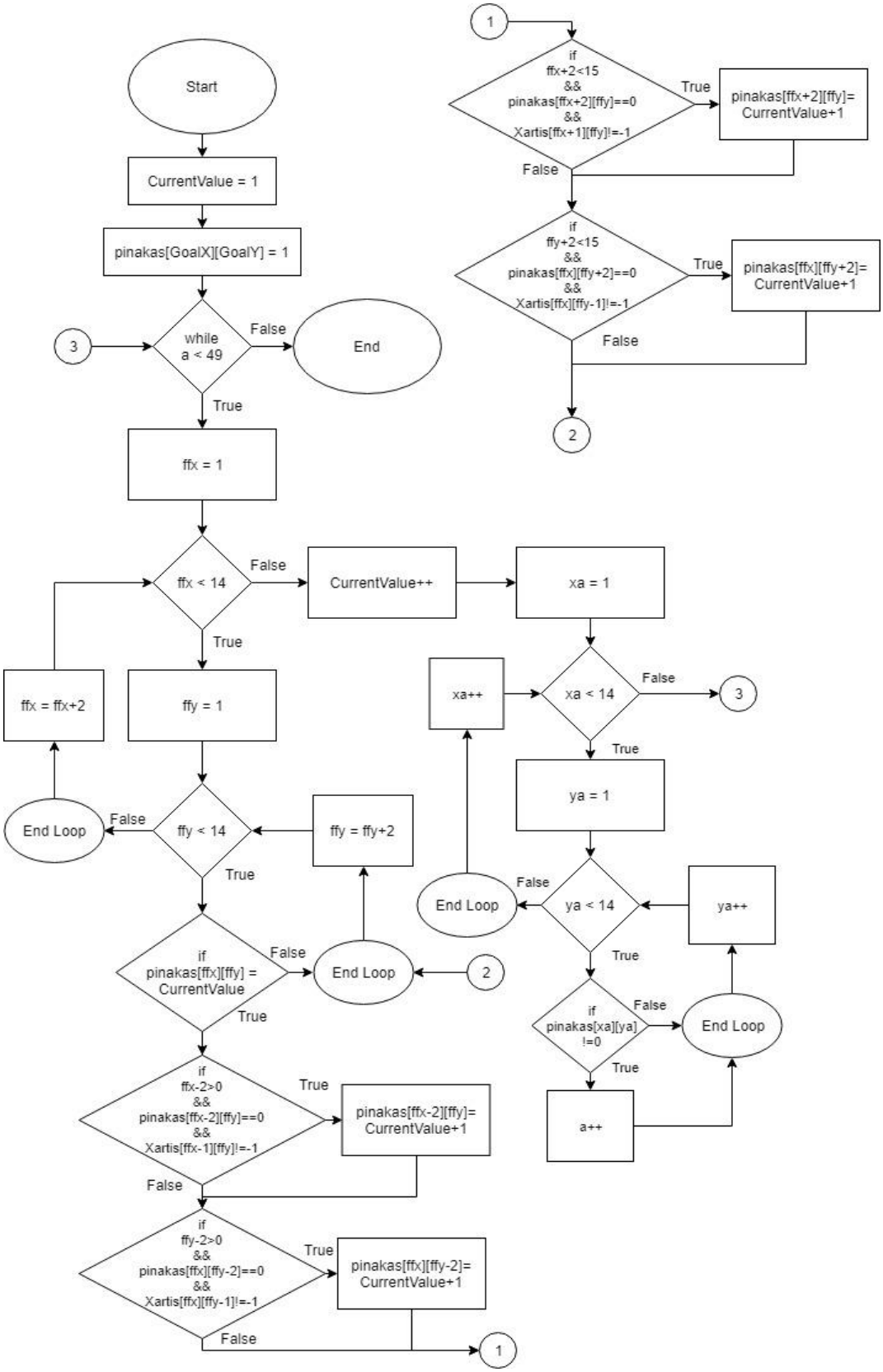


Εικόνα 14 - Διάγραμμα Ροής βασικής λειτουργίας του οχήματος

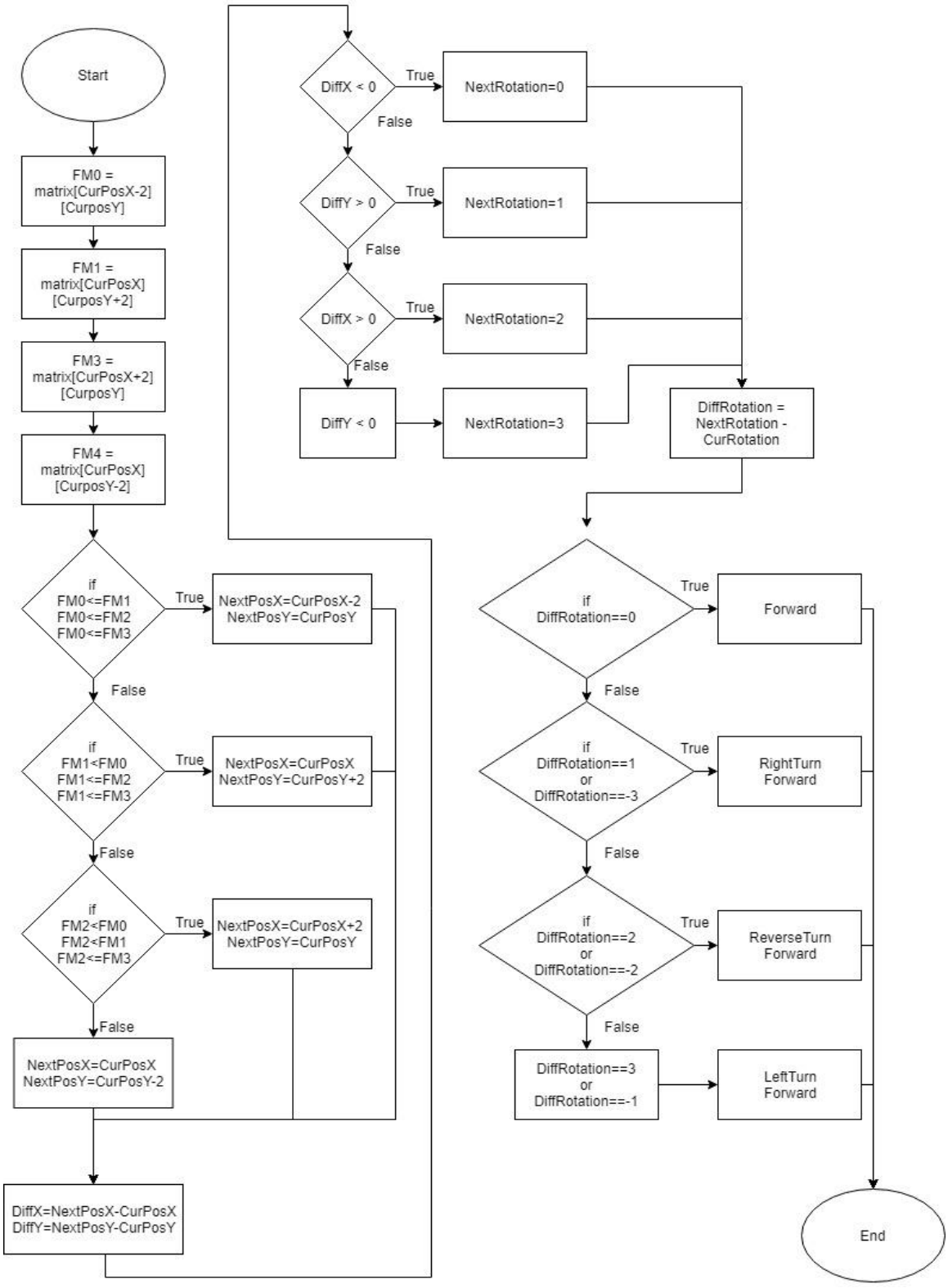
Και παρακάτω ακολουθούν τα διαγράμματα ροής των βοηθητικών λειτουργιών του κώδικα.



Εικόνα 15 - Διάγραμμα ροής Scan



Εικόνα 16 - Διάγραμμα ροής Flood Fill



Εικόνα 17 - Διάγραμμα ροής Εύρεσης Επόμενης Κίνησης

Στην λειτουργία Scan, το όχημα απλά διαβάζει τις τιμές των τριών αισθητηρίων για να προσθέσει στον χάρτη του λαβυρίνθου τις τοποθεσίες των τοίχων.

Στο Flood Fill αρχικά δίνεται στο τετράγωνο που αντιστοιχεί στην τελική θέση η τιμή ένα (1). Από εκεί και πέρα μέσω δύο εμφωλευμένων for προστίθενται και μεγαλύτερες τιμές (2, 3, κλπ.) στα γειτονικά τετράγωνα, έως ότου να γεμίσει όλος ο πίνακας με τιμές. Αυτή η διαδικασία τελειώνει όταν μία μεταβλητή που υπάρχει για μετρητής (a) φτάσει στην τιμή 49, δηλαδή όσα είναι και τα τετράγωνα του λαβυρίνθου. Ο έλεγχος επιτυγχάνεται με άλλες δύο εμφωλεύμενες for, όπου δίνουν +1 στο a όταν βρίσκουν κάποια τιμή εντός του κάθε τετραγώνου του χάρτη.

Στον αλγόριθμο εύρεσης επόμενης κίνησης, το όχημα, με βάση την τωρινή θέση του, διαβάζει τις τέσσερις γειτονικές τιμές, που έχουν προέλθει από το Flood Fill, και βρίσκει την μικρότερη. Οπότε πλέον ξέρει προς ποιό τετράγωνο πρέπει να κινηθεί. Επόμενο βήμα είναι να κάνει μία απλή πράξη, μεταξύ της τωρινής κατεύθυνσης του και της επόμενης, και να βρει τις κατάλληλες κινήσεις για την σωστή μετακίνηση του, τις οποίες και εκτελεί.

Κώδικας Προγράμματος

Το όχημα ξεκινάει από την κάτω αριστερή γωνία του λαβυρίνθου (7,1). Από εκεί και πέρα εκτελεί κινήσεις μέχρι να περάσει από όλες θέσεις με σκοπό την χαρτογράφηση. Αυτό επιτυγχάνεται με χρήση δύο εμφωλευμένων for και ενός βοηθητικού χάρτη που σημειώνει από ποιές θέσεις έχει περάσει το όχημα. Όταν καταφέρει να περάσει και από τις 49 θέσεις, τότε τελείωνει το πρώτο και μεγαλύτερο κομμάτι της διαδικασίας, οπότε περνάει στο δεύτερο, που είναι η επιστροφή του στην αρχική θέση. Τις κινήσεις τις υπολογίζει με την χρήση του Flood Fill, όπου παίρνει σαν θέση ενδιαφέροντος την αρχική θέση του λαβυρίνθου (7,1) σε σχέση με την τωρινή του θέση. Αφού υπολογίσει και εκτελέσει τις κινήσεις (με την βοήθεια του αλγορίθμου εύρεσης επόμενης κίνησης) και πλέον είναι στην αρχική θέση (7,1) μπαίνει σε λειτουργία το τρίτο στάδιο και βασικός σκοπός της εργασίας, που είναι ο υπολογισμός και εκτέλεση της βέλτιστης διαδρομής. Στο συγκεκριμένο παράδειγμα έχουμε ως αρική θέση το (7,1) και ως τελική το (5,6), οπότε υπολογίζει τις κινήσεις με τον ίδιο τρόπο που έκανε και στο δεύτερο βήμα της διαδικασίας (Flood Fill και εύρεση επόμενης κίνησης).

Βοηθητικές συναρτήσεις στον κώδικα είναι: η μέτρηση απόστασης από τα τρία υπέρηχα αισθητήρια, η εκτέλεση κινήσεων (ευθεία, αριστερή και δεξιά στροφή, αναστροφή, στόπ) με την χρήση των δύο κινητήρων που έχουμε, καθώς και οι δύο αλγόριθμοι (Flood Fill και Εύρεση Επόμενης Κίνησης) που τους αναλύσαμε στο κομμάτι με τα διαγράμματα ροής.

```
/********** Αυτοκινούμενο όχημα εύρεσης βέλτιστης διαδρομής λαβυρίνθου
 *  Σκοπός
 *    Ένα όχημα βρίσκει και εκτελεί την ταχύτερη διαδρομή μεταξύ 2 σημείων
 *   του λαβυρίνθου, αφού πρώτα κάνει την πλήρη χαρτογράφηση του
 *  Hardware
 *    Στο arduino είναι συνδεδεμένα 3 υπέρηχα αισθητήρια HC-SR04, καθώς
 *    και 2 DC  κινητήρες σε θήκη servo μέσω ενός οδηγού L293
 *  Software
 *    Γίνεται χρήση 2 αλγορίθμων: Flood Fill και εύρεση επόμενης κίνησης,
 *    καθώς και απλών συναρτήσεων για την εκτέλεση των μηχανικών
 *   κινήσεων και την λήψη τιμών των αισθητηρίων.
 *  Reference
 *    final version Νικόλαος Ευδαίμων - Βασίλειος Νικολουδάκος, Απρίλιος 2019.
 **********/
```

```
//Motors
int in1 = 22;   //Ορισμός κινητήρων
int in2 = 24;
int in3 = 44;
int in4 = 46;

//Ultrasound Sensors
int TriggerPin1 = 32;      //Αριστερό Αισθητήριο
int EchoPin1 = A8;
int TriggerPin2 = 34;      //Μπροστινό Αισθητήριο
int EchoPin2 = A9;
int TriggerPin3 = 36;      //Δεξιό Αισθητήριο
int EchoPin3 = A10;

//Misc
int VehicleRotation = 0;   //Μεταβλητή Κατεύθυνσης(Αρχική Θέση: Βόρεια(0), Υπόλοιπες θέσεις:Δυτικά(1), Νότια(2), Ανατολικά(3))
int Xartis[15][15] = {{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
                      {-1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1},
                      {-1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1},
                      {-1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1},
                      {-1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1},
                      {-1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1},
                      {-1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1},
                      {-1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1},
                      {-1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1},
                      {-1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1},
                      {-1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1},
                      {-1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1},
                      {-1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1},
                      {-1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1},
                      {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}};

//Χάρτης από αναγνώριση αυτοκινήτου, με -1 σημειώνονται τοίχοι και απροσπέλαστα σημεία, //ενώ με 0 ανοικτοί δρόμοι καθώς και άγνωστα σημεία
int PositionCheckMap[7][7] = {{0, 0, 0, 0, 0, 0, 0},
                              {0, 0, 0, 0, 0, 0, 0},
                              {0, 0, 0, 0, 0, 0, 0},
                              {0, 0, 0, 0, 0, 0, 0},
                              {0, 0, 0, 0, 0, 0, 0},
                              {0, 0, 0, 0, 0, 0, 0},
                              {0, 0, 0, 0, 0, 0, 0}};
```

```
//Χρησιμοποιείται για να ξέρει το όχημα από ποιές θέσεις έχει περάσει

int PositionSum=0;
int StartingPositionX=13; //Αρχική Θέση οχήματος
int StartingPositionY=1;
int GoalPositionX=9;      //Τελική θέση οχήματος
int GoalPositionY=11;
int CurrentPositionX;     //Τρέχουσα θέση οχήματος
int CurrentPositionY;
int NextPositionX;            //Επόμενη θέση οχήματος(μετά από εκτέλεση αλγορίθμου εύρεσης επόμενης κίνησης)
int NextPositionY;
int FM0,FM1,FM2,FM3;      //Τιμές γειτονικών τετραγώνων
int DifferenceX;
int DifferenceY;
int NextRotation;
int DifferenceRotation;
int pinakas[15][15];      //Βοηθητικός πίνακας
int LeftSensor,FrontSensor,RightSensor;    //Τιμές από υπέρηχα αισθητήρια



void setup() {
  Serial.begin(9600);
  //Motors
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  //Ultrasound Sensors
  pinMode(TriggerPin1,OUTPUT);
  pinMode(TriggerPin2,OUTPUT);
  pinMode(TriggerPin3,OUTPUT);
  pinMode(EchoPin1,INPUT);
  pinMode(EchoPin2,INPUT);
  pinMode(EchoPin3,INPUT);

}

void loop() {
```

```
        //MAIN CODE
        CurrentPositionX=StartingPositionX;
        CurrentPositionY=StartingPositionY;
        Scanning();
        PositionCheck();
        //Πρώτο Μέρος (Αναγνώριση λαβυρίνθου και χαρτογράφηση του)
        //Εκτέλεση κινήσεων μέχρι να περάσει το όχημα από όλες τις θέσεις
        while(PositionSum<49){
          for(int x=1;x<14;x+=2){
            for(int y=1;y<14;y+=2){
              if(PositionCheckMap[(x-1)/2][(y-1)/2]==0){
                Scanning();
                FloodFill(Xartis,x,y);
                PositionCheck();
                while((CurrentPositionX!=x)||(CurrentPositionY!=y)){
                  PositionCheck();
                  FindMove(pinakas);
                  Scanning();
                  PositionCheck();
                  FloodFill(Xartis,x,y);
                }
              }
            }
          }
        }
        //Δεύτερο Μέρος (Επιστροφή στην αρχική θέση)
        ReturnToStart();
        //Τρίτο Μέρος (Βέλτιστη διαδρομή προς την τελική θέση που του έχουμε
ορίσει)
        FloodFill(Xartis,GoalPositionX,GoalPositionY);

while((CurrentPositionX!=GoalPositionX)||(CurrentPositionY!=GoalPositionY)){
          FindMove(pinakas);
        }
      }



      //FUNCTIONS
```

```
//Position Checks
void PositionCheck(){  //Συνάρτηση ελέγχου για αναγνώριση λαβυρίνθου
  PositionSum=0;
  if(PositionCheckMap[(CurrentPositionX-1)/2][(CurrentPositionY-
1)/2]==0){
    PositionCheckMap[(CurrentPositionX-1)/2][(CurrentPositionY-1)/2]=1;
  }
  for(int d=0;d<7;d++){
    for(int e=0;e<7;e++){
      if(PositionCheckMap[d][e]==1){
        PositionSum++;
      }
    }
  }
}



//Return to Start
void ReturnToStart(){ //Συνάρτηση επιστροφής στην αρχική θέση
  FloodFill(Xartis,13,1);
  //Εκτέλεση κινήσεων μέχρι να φτάσει το όχημα στην αρχική θέση του
  while((CurrentPositionX!=13)||(CurrentPositionY!=1)){
    FindMove(pinakas);
  }
  ReverseTurn();
}



//Ultrasound Distance
long ultrasound_distance(long trigPin,long echoPin){//Συνάρτηση για
μέτρηση απόστασης
  long duration, distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(.3);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(1);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = (duration/2) / 29.1;
  delay(60);
  return distance;
```

```
}

//Movements
void Forward(){
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
  delay(1300);
  Stop();
  switch (VehicleRotation){
    case 0 :
    CurrentPositionX=CurrentPositionX-2;
    break;
    case 1 :
    CurrentPositionY=CurrentPositionY+2;
    break;
    case 2 :
    CurrentPositionX=CurrentPositionX+2;
    break;
    case 3 :
    CurrentPositionY=CurrentPositionY-2;
    break;
  }
  delay(2000);
}



void RightTurn(){
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
  delay(450);
  Stop();
  VehicleRotation=VehicleRotation+1;
  if(VehicleRotation>3){//Έλεγχος για να μην φύγει εκτός "κύκλου κατευθύνσεων"
    VehicleRotation-=4;  //(0-1-2-3)
  }
```

```
  delay(2000);
 }


 void LeftTurn(){
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
  delay(450);
  Stop();
  VehicleRotation=VehicleRotation-1;
  if(VehicleRotation<0){//Έλεγχος για να μην φύγει εκτός "κύκλου κατευθύνσεων"
    VehicleRotation+=4; //(0-1-2-3)
  }
  delay(2000);
 }


 void ReverseTurn(){
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
  delay(900);
  Stop();
  VehicleRotation=VehicleRotation+2;
  if(VehicleRotation>3){//Έλεγχος για να μην φύγει εκτός "κύκλου κατευθύνσεων"
    VehicleRotation-=4; //(0-1-2-3)
  }
  delay(2000);
 }



 void Stop(){
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
```

```
  digitalWrite(in4, LOW);
}



//WALL SCANNING
void Scanning(){
  //Διάβασμα των τριών αισθητηρίων για την εύρεση τοίχων περιμετρικά του
  //οχήματος και εγγραφή των στοιχείων στον χάρτη
  LeftSensor=ultrasound_distance(TriggerPin1,EchoPin1);
  FrontSensor=ultrasound_distance(TriggerPin2,EchoPin2);
  RightSensor=ultrasound_distance(TriggerPin3,EchoPin3);
  if (LeftSensor<20){//Αριστερό Αισθητήριο
    switch (VehicleRotation){
      case 0 :
      if(Xartis[CurrentPositionX][CurrentPositionY-1]==0){
        Xartis[CurrentPositionX][CurrentPositionY-1]=-1;
      }
        break;
      case 1 :
      if(Xartis[CurrentPositionX-1][CurrentPositionY]==0){
        Xartis[CurrentPositionX-1][CurrentPositionY]=-1;
      }
        break;
      case 2 :
      if(Xartis[CurrentPositionX][CurrentPositionY+1]==0){
        Xartis[CurrentPositionX][CurrentPositionY+1]=-1;
      }
        break;
      case 3 :
      if(Xartis[CurrentPositionX+1][CurrentPositionY]==0){
        Xartis[CurrentPositionX+1][CurrentPositionY]=-1;
      }
        break;
    }
  }
  if (FrontSensor<20){//Μπροστινό Αισθητήριο
    switch (VehicleRotation){
      case 0 :
      if(Xartis[CurrentPositionX-1][CurrentPositionY]==0){
```

```
      Xartis[CurrentPositionX-1][CurrentPositionY]=-1;
    }
      break;
    case 1 :
    if(Xartis[CurrentPositionX][CurrentPositionY+1]==0){
      Xartis[CurrentPositionX][CurrentPositionY+1]=-1;
    }
      break;
    case 2 :
    if(Xartis[CurrentPositionX+1][CurrentPositionY]==0){
      Xartis[CurrentPositionX+1][CurrentPositionY]=-1;
    }
      break;
    case 3 :
    if(Xartis[CurrentPositionX][CurrentPositionY-1]==0){
      Xartis[CurrentPositionX][CurrentPositionY-1]=-1;
    }
      break;
  }
}
if (RightSensor<20){//Δεξιό Αισθητήριο
  switch (VehicleRotation){
    case 0 :
    if(Xartis[CurrentPositionX][CurrentPositionY+1]==0){
      Xartis[CurrentPositionX][CurrentPositionY+1]=-1;
    }
      break;
    case 1 :
    if(Xartis[CurrentPositionX+1][CurrentPositionY]==0){
      Xartis[CurrentPositionX+1][CurrentPositionY]=-1;
    }
      break;
    case 2 :
    if(Xartis[CurrentPositionX][CurrentPositionY-1]==0){
      Xartis[CurrentPositionX][CurrentPositionY-1]=-1;
    }
      break;
    case 3 :
    if(Xartis[CurrentPositionX-1][CurrentPositionY]==0){
      Xartis[CurrentPositionX-1][CurrentPositionY]=-1;
```

```
        }
          break;
      }
    }
  }
```

```
  //FLOODFILL ALGORITHM
  void FloodFill(int FFmatrix[15][15],int GoalX,int GoalY){
    for(int copy1=0;copy1<15;copy1++){
      for(int copy2=0;copy2<15;copy2++){
        pinakas[copy1][copy2]=FFmatrix[copy1][copy2];    //Αντιγραφή πίνακα
για
      }                                    //αποφυγή αλλοίωσης βασικού χάρτη
    }
    int a=0;    //Μεταβλητή για την εκτέλεση του αλγορίθμου. Όταν φτάσει 49
θα
    int CurrentValue=1;           //έχουν μπει τιμές σε όλες τις θέσεις
    pinakas[GoalX][GoalY]=CurrentValue;  //Τελική θέση FloodFill
    //Πρέπει να ισχύουν τρεις συνθήκες
    //1η:Αν είναι μέσα στα όρια του πίνακα
    //2η:Αν είναι η τιμή της 0 (δεν είναι μέρος συντομότερης διαδρομής)
    //3η:Αν δεν υπάρχει τοίχος ανάμεσα (-1 δείχνει τοίχο)
    while(a<49){
      for(int ffx=1;ffx<14;ffx+=2){
        for(int ffy=1;ffy<14;ffy+=2){
          if(pinakas[ffx][ffy]==CurrentValue){
            if((ffx-2>=0)&&(pinakas[ffx-2][ffy]==0)&&(Xartis[ffx-1][ffy]!=-
1)){
              pinakas[ffx-2][ffy]=CurrentValue+1;
            }
            if((ffy-2>=0)&&(pinakas[ffx][ffy-2]==0)&&(Xartis[ffx][ffy-1]!=-
1)){
              pinakas[ffx][ffy-2]=CurrentValue+1;
            }
            if((ffx+2<15)&&(pinakas[ffx+2][ffy]==0)&&(Xartis[ffx+1][ffy]!=-
1)){
              pinakas[ffx+2][ffy]=CurrentValue+1;
            }
```

```
                if((ffy+2<15)&&(pinakas[ffx][ffy+2]==0)&&(Xartis[ffx][ffy+1]!=-
1)){
                  pinakas[ffx][ffy+2]=CurrentValue+1;
                }
              }
            }
          }
          CurrentValue=CurrentValue+1;
          a=0;
          for(int xa=1;xa<14;xa=xa+2){           //Όποια θέση έχει τιμή τότε
προστίθεται
            for(int ya=1;ya<14;ya=ya+2){     //στην μεταβλητή a μέχρι να φτάσει
το 49
              if(pinakas[xa][ya]!=0){        //(7x7=49)
                a++;
              }
            }
          }
        }
      }



      //FIND NEXT MOVE ALGORITHM
      void FindMove(int FMmatrix[15][15]){//Του δίνουμε πίνακα μετά από Flood
Fill για να βρει την μικρότερη τιμή των τεσσάρων γειτονικών τετραγώνων από την
τρέχουσα θέση
        //Εύρεση τιμών των τεσσάρων γειτονικών τετραγώνων
        if((CurrentPositionX-2>0)&&(Xartis[CurrentPositionX-
1][CurrentPositionY]==0)&&(FMmatrix[CurrentPositionX-2][CurrentPositionY]!=0)){
//Έλεγχος πάνω τετραγώνου
          FM0=FMmatrix[CurrentPositionX-2][CurrentPositionY];    //Τιμή    πάνω
τετραγώνου
        }
        else{
          FM0=100;  //Το 100 είναι μια τυχαία "μεγάλη" τιμή, μακριά από τις δικές
μας (1->15)
        }

if((CurrentPositionY+2<14)&&(Xartis[CurrentPositionX][CurrentPositionY+1]==0)&&(
FMmatrix[CurrentPositionX][CurrentPositionY+2]!=0)){  //Έλεγχος δεξιά τετραγώνου
          FM1=FMmatrix[CurrentPositionX][CurrentPositionY+2];   //Τιμή   δεξιού
τετραγώνου
        }
```

```
        else{
          FM1=100;
        }

if((CurrentPositionX+2<14)&&(Xartis[CurrentPositionX+1][CurrentPositionY]==0)&&(
FMmatrix[CurrentPositionX+2][CurrentPositionY]!=0)){  //Έλεγχος κάτω τετραγώνου
          FM2=FMmatrix[CurrentPositionX+2][CurrentPositionY];    //Τιμή    κάτω
τετραγώνου
        }
        else{
          FM2=100;
        }
        if((CurrentPositionY-2>0)&&(Xartis[CurrentPositionX][CurrentPositionY-
1]==0)&&(FMmatrix[CurrentPositionX][CurrentPositionY-2]!=0)){          //Έλεγχος
αριστερά τετραγώνου
          FM3=FMmatrix[CurrentPositionX][CurrentPositionY-2];  //Τιμή  αριστερού
τετραγώνου
        }
        else{
          FM3=100;
        }
        //Σύγκριση των τιμών για την εύρεση της επόμενης θέσης
        if((FM0<=FM1)&&(FM0<=FM2)&&(FM0<=FM3)){ //Αν η FM0 είναι η μικρότερη τιμή
τότε
          NextPositionX=CurrentPositionX-2;    //το όχημα θα πρέπει να πάει πάνω
          NextPositionY=CurrentPositionY;
        }
        if((FM1<FM0)&&(FM1<=FM2)&&(FM1<=FM3)){  //Αν η FM1 είναι η μικρότερη τιμή
τότε
          NextPositionX=CurrentPositionX;       //το όχημα πρέπει να πάει δεξιά
          NextPositionY=CurrentPositionY+2;
        }
        if((FM2<FM0)&&(FM2<FM1)&&(FM2<=FM3)){   //Αν η FM2 είναι η μικρότερη τιμή
τότε
          NextPositionX=CurrentPositionX+2;     //το όχημα πρέπει να πάει κάτω
          NextPositionY=CurrentPositionY;
        }
        if((FM3<FM0)&&(FM3<FM1)&&(FM3<FM2)){    //Αν η FM3 είναι η μικρότερη τιμή
τότε
          NextPositionX=CurrentPositionX;      //το όχημα πρέπει να πάει αριστερά
          NextPositionY=CurrentPositionY-2;
        }
        //Υπολογισμός των κατάλληλων κινήσεων για την μετάβαση στην επόμενη θέση
        DifferenceX=NextPositionX-CurrentPositionX;
```

```
        DifferenceY=NextPositionY-CurrentPositionY;
        if(DifferenceX<0){
          NextRotation=0;
        }
        if(DifferenceY>0){
          NextRotation=1;
        }
        if(DifferenceX>0){
          NextRotation=2;
        }
        if(DifferenceY<0){
          NextRotation=3;
        }
        DifferenceRotation=NextRotation-VehicleRotation;
        if(DifferenceRotation<0){
          DifferenceRotation+=4;
        }
        //Εκτέλεση κινήσεων
        if(DifferenceRotation==0){ //Δεν υπάρχει αλλαγή στην κατεύθυνση, άρα προχωράει ευθεία το όχημα
          Forward();
        }
        if((DifferenceRotation==1)||(DifferenceRotation==-3)){ //Το όχημα πρέπει να κάνει δεξιά στροφή
          RightTurn();
          Forward();
        }
        if((DifferenceRotation==2)||(DifferenceRotation==-2)){ //Το όχημα πρέπει να κάνει αναστροφή
          ReverseTurn();
          Forward();
        }
        if((DifferenceRotation==3)||(DifferenceRotation==-1)){ //Το όχημα πρέπει να κάνει αριστερή στροφή
          LeftTurn();
          Forward();
        }

      }
```

ΑΠΟΤΕΛΕΣΜΑΤΑ

Μετά το πέρας της εργασίας, διαπιστώσαμε ότι το όχημα εκπλήρωσε σωστά τον βασικό μας στόχο, που ήταν ο υπολογισμός και η εκτέλεση της βέλτιστης διαδρομής μεταξύ των δύο σημείων που είχαμε θέσει, αλλά αφού πρώτα είχε αναγνωρίσει και χαρτογραφήσει πλήρως τον λαβύρινθο. Άλλωστε, χωρίς αυτή την διαδικασία, ο βασικός στόχος δεν θα μπορούσε να επιτευχθεί με ακρίβεια. Επίσης, επαληθεύτηκαν και τα αποτελέσματα από τις προσομοιώσεις που είχαμε κάνει στην αρχή της εργασίας με τη χρήση της σειριακής επικοινωνίας του Arduino.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Καταλήξαμε στα εξής συμπεράσματα:

1) Ο αλγόριθμος Flood Fill είναι κατάλληλος για επίλυση λαβυρίνθου, όταν αυτός έχει συγκεκριμένα χαρακτηριστικά. Στην περίπτωσή μας είχαμε τετράγωνο λαβύρινθο χωρισμένο σε ίσα τετράγωνα σημεία, κάτι που κάνει εύκολη την χαρτογράφηση του. Σε περίπτωση που ο λαβύρινθος ήταν κυκλικός ή τυχαίου σχήματος δεν θα είχαμε τόσο μεγάλη επιτυχία στην επίλυση του. Για αυτές τις περιπτώσεις υπάρχουν άλλες τεχνικές επίλυσης.

2) Μελλοντικές βελτιώσεις που θα μπορούσαν να γίνουν στην εργασία είναι κυρίως στο υλικό κομμάτι, δηλαδή στο όχημα. Μερικές προσθήκες είναι η χρήση encoders στους τροχούς για την ακριβή μέτρηση των κινήσεων, η τοποθέτηση γυροσκοπίου και επιταχυνσιόμετρου για την εξακρίβωση της θέσης του οχήματος στους τρείς άξονες (X,Y,Z),καθώς και η χρήση υπέρυθρων ακολουθητών γραμμής και η αντίστοιχη μαύρη ταινία στο έδαφος για σταθεροποίηση των κινήσεων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Elshmarka Ibrahim and A. B. S. Saman, "Design and Implementation of a Robot for Maze-Solving using Flood Fill Algorithm," *International Journal of Computer Applications (0975 – 8887)*, vol. 56, no. 5, pp. 8-13, 2012. 4 4 2019.

[2] J. R. B. D. Rosario, J. G. Sanidad, A. M. Lim, P. S. L. Uy, A. J. C. Bacar, M. A. D. Cai and A. Z. A. Dubouzet, "Modelling and Characterization of a Maze-Solving Mobile Robot Using Wall Follower Algorithm," *Applied Mechanics and Materials Vols*, pp. 1245-1249, 2014.

[3] Random Mouse Algorithm, 4 4 2019. [Online] Available: https://en.wikipedia.org/wiki/Maze_solving_algorithm

[4] Arduino Mega 2560, Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/images/detailed/0/11061-01b.jpg

[5] DC Motors FM90, Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/dc-motor-in-micro-servo-body.html

[6] Αισθητήρες Υπερήχων (HC-SR04), Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/ultrasonic-sensor-sr04.html

[7] Breadboard, Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/breadboard-830-tie-point-classic.html

[8] Motor Driver L293D 1A, Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/motor-driver-l293d-1a.html

[9] Ηλεκτρονικό κύκλωμα και σχέδιο μηχανολογίας, 4 4 2019 Available : http://fritzing.org/home/

[10] Ηλεκτρικά χαρακτηριστικά , Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/arduino-mega-2560-rev3.html

[11] Ηλεκτρικά χαρακτηριστικά , Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/dc-motor-in-micro-servo-body.html

[12] Ηλεκτρικά χαρακτηριστικά , Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/ultrasonic-sensor-sr04.html

[13] Ηλεκτρικά χαρακτηριστικά , Grobotronics.com, 4 4 2019. [Online]. Available:https://grobotronics.com/breadboard-830-tie-point-classic.html

[14] Ηλεκτρικά χαρακτηριστικά , Grobotronics.com, 4 4 2019. [Online]. Available: https://grobotronics.com/motor-driver-l293d-1a.html