ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αναβάθμιση και Προγραμματισμός Delta Robot Ώστε να Αναγνωρίζει και να Μετακινεί Αντικείμενα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Δ. Γεωργούσης

Επιβλέπων: Γεώργιος Καμπουράκης
Αν. Καθηγητής ΕΜΠ

Αθήνα, Οκτώβριος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αναβάθμιση και Προγραμματισμός Delta Robot Ώστε να Αναγνωρίζει και να Μετακινεί Αντικείμενα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Δ. Γεωργούσης

Επιβλέπων: Γεώργιος Καμπουράκης
Αν. Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24η Οκτωβρίου 2016.

.....	.....	.....
Γεώργιος Καμπουράκης	Βασίλειος Λούμος	Ηλίας Κουκούτσης
Αν. Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Επ. Καθηγητής ΕΜΠ

Αθήνα, Οκτώβριος 2016

……

Γεώργιος Δ. Γεωργούσης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών ΕΜΠ

Περίληψη

Σκοπός αυτής της εργασίας είναι η αναβάθμιση και ο προγραμματισμός ενός προϋπάρχοντος Delta Robot προκειμένου να αναγνωρίζει οπτικά και να μετακινεί αντικείμενα.

Στο εργαστήριο υπήρχε ένα Delta Robot το οποίο κατασκευάστηκε στα πλαίσια παλαιότερης διπλωματικής εργασίας. Ο χώρος εργασίας του αποτελείται από 9 τετράγωνα διατεταγμένα σε τρεις τριάδες. Αρχικά η λειτουργία που μπορούσε να επιτελεί ήταν να μαζεύει 9 μεταλλικές κεφαλές πινεζών με τη χρήση ηλεκτρομαγνήτη και να τις συγκεντρώνει σε ένα κουτί. Σκοπός της συγκεκριμένης εργασίας ήταν να αναβαθμιστεί και να προγραμματιστεί το ρομπότ κατάλληλα ώστε να μπορεί να παίζει τρίλιζα αναγνωρίζοντας οπτικά, με χρήση κάμερας, τις θέσεις που βρίσκονται τοποθετημένα τα πιόνια προκειμένου να αντιλαμβάνεται τις κινήσεις του αντιπάλου. Επιπλέον έπρεπε να αναγνωρίζει τυχόν μη έγκυρες κινήσεις από τον παίκτη, «κλέψιμο» ή «ζαβολιά», και να επαναφέρει τα πιόνια στις προηγούμενες έγκυρες θέσεις τους.

Για την υλοποίηση του Delta Robot είχε χρησιμοποιηθεί μικροελεγκτής Arduino Mega ο οποίος στα πλαίσια αυτής της εργασίας επαναπρογραμματίστηκε κατάλληλα ώστε να μπορεί να δέχεται συντεταγμένες μέσω σειριακής θύρας και να μετακινεί την κεφαλή του ρομπότ στη ζητούμενη θέση. Επίσης με την χρήση χειριστηρίων κατάλληλα κολλημένων σε Arduino Shield είναι δυνατό να δέχεται απ' ευθείας εντολές από τον χρήστη. Για τις κύριες λειτουργίες της εργασίας αυτής, όπως η οπτική αναγνώριση, η απόφαση για την επόμενη κίνηση και η αποστολή των συντεταγμένων στον Arduino, χρησιμοποιήθηκε μικροϋπολογιστής Raspberry Pi με λειτουργικό σύστημα Raspbian, η βιβλιοθήκη OpenCV καθώς και η επίσημη κάμερα για Raspberry Pi.

Λέξεις κλειδιά

Delta Robot, οπτική αναγνώριση αντικειμένων, μετακίνηση αντικειμένων, τρίλιζα, Arduino Mega, Raspberry Pi, Raspbian, OpenCV

Abstract

The objective of this thesis was to upgrade and program a pre-existing Delta Robot in order to optically recognize and move objects.

There was a Delta Robot in the laboratory which had been manufactured during an older thesis. Its work space was consisted of 9 squares arranged in three groups of three. Initially it had been programmed to pick 9 metallic pin heads using an electromagnet and gather them into a box. The objective of this thesis was to upgrade and program the robot so as to play tic tac toe by optically recognizing the positions of the pawns using a camera in order to apprehend the opponent moves. Moreover, it had to recognize any invalid player moves, such as cheating, and restore the game board by putting the pawns to their previous valid positions.

An Arduino Mega microcontroller had been used to control the Delta Robot. For this project it was reprogrammed so as to move the head of the robot to the coordinates received via serial port. Additionally, it was given the ability to accept commands directly from user via suitable controllers soldered on an Arduino Shield. For the main functions of this project, such as the optical recognition, the decision for the next move and the transmission of the coordinates to the Arduino, a Raspberry Pi microcomputer running Raspbian operating system was used with the OpenCV library and an official Raspberry Pi camera installed.

Keywords

Delta Robot, optical object recognition, move objects, Tic Tac Toe, Arduino Mega, Raspberry Pi, Raspbian, OpenCV

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Αν. Καθηγητή ΕΜΠ κύριο **Γεώργιο Καμπουράκη**, επιβλέποντα της εργασίας αυτής, για την πολύ ενδιαφέρουσα διπλωματική εργασία που μου ανέθεσε καθώς και για την πολύτιμη βοήθειά του κατά την εκπόνηση αυτής, αλλά και για τις ενδιαφέρουσες γνώσεις που μου μετέδωσε σχετικά με διάφορα θέματα επιστημονικού ενδιαφέροντος.

Πίνακας περιεχομένων

Εισαγωγή

Σκοπός αυτής της διπλωματικής εργασίας είναι η αναβάθμιση και ο προγραμματισμός ενός προϋπάρχοντος Delta Robot προκειμένου να αναγνωρίζει οπτικά και να μετακινεί αντικείμενα. Πιο συγκεκριμένα, ο κύριος στόχος της διπλωματικής αυτής είναι να μπορεί το ρομπότ να αλληλεπιδρά με το χρήστη παίζοντας τρίλιζα αναγνωρίζοντας οπτικά τις θέσεις των πιονιών του αντιπάλου και να αποφασίζει για την επόμενη κίνηση.

Στο εργαστήριο υπήρχε ένα Delta Robot ως αποτέλεσμα προηγούμενης διπλωματικής εργασίας (Διπλωματική Εργασία Γεωργίου Βαγενά) η οποία είχε σαν βασικό σκοπό την κατασκευή του ρομπότ και την δυνατότητα να υπολογίζει τις γωνίες των σερβοκινητήρων με δεδομένες τις συντεταγμένες που ήταν επιθυμητό να πάει η κεφαλή του. Ο έλεγχος των σερβοκινητήρων γινόταν από έναν Arduino Mega, ο οποίος προγραμματίζεται μέσω ηλεκτρονικού υπολογιστή με τη χρήση κατάλληλα προσαρμοσμένης γλώσσας C++. Το ρομπότ μπορούσε να εκτελεί μία απλή εργασία επίδειξης των δυνατοτήτων του όπου μάζευε 9 μεταλλικές κεφαλές πινεζών από 9 τετράγωνα στην επιφάνεια εργασίας του και τις συγκέντρωνε σε ένα σημείο με τη βοήθεια ενός ηλεκτρομαγνήτη που ήταν τοποθετημένος στην κεφαλή του.

Στα πλαίσια αυτής της διπλωματικής αυτό αναβαθμίστηκε και προγραμματίστηκε ώστε να μπορεί να δέχεται είτε από την σειριακή θύρα είτε από κατάλληλα χειριστήρια (ποτενσιόμετρα), τα οποία κολλήθηκαν σε πλακέτα Arduino Shield, τις επιθυμητές καρτεσιανές συντεταγμένες (x, y, z) καθώς και την επιθυμητή κατάσταση του ηλεκτρομαγνήτη (on ή off) και κατόπιν να τοποθετεί την κεφαλή στην αντίστοιχη θέση και να ανάβει ή να σβήνει τον ηλεκτρομαγνήτη αντίστοιχα. Η επιλογή για το αν θα δέχεται συντεταγμένες τοπικά από τα ποτενσιόμετρα ή απομακρυσμένα από την σειριακή θύρα γίνεται με τη βοήθεια κατάλληλων μπουτόν που επίσης κολλήθηκαν στο Arduino Shield μαζί με ενδείκτες LED για να πληροφορούν το χρήστη για την επιλεγμένη μέθοδο εισαγωγής συντεταγμένων καθώς και για άλλες παραμέτρους.

Για την κύρια λειτουργία της διπλωματικής, δηλαδή για να μπορεί να παίζει τρίλιζα, χρησιμοποιήθηκε μικροϋπολογιστής Raspberry Pi 2 Model B με λειτουργικό σύστημα Raspbian Jessie, που αποτελεί έκδοση του Debian Linux, μία κάμερα για Raspberry Pi καθώς και η βιβλιοθήκη OpenCV. Αυτό συνδέεται μέσω της σειριακής θύρας με τον Arduino Mega προκειμένου να του στέλνει τις συντεταγμένες της κεφαλής και την κατάσταση του ηλεκτρομαγνήτη. Η χρήση του Raspberry ήταν απαραίτητη γιατί ο Arduino Mega δεν έχει τη δυνατότητα να συνδεθεί με κάμερα, ούτε αρκετή μνήμη και υπολογιστική ισχύ ώστε να κάνει την οπτική αναγνώριση και

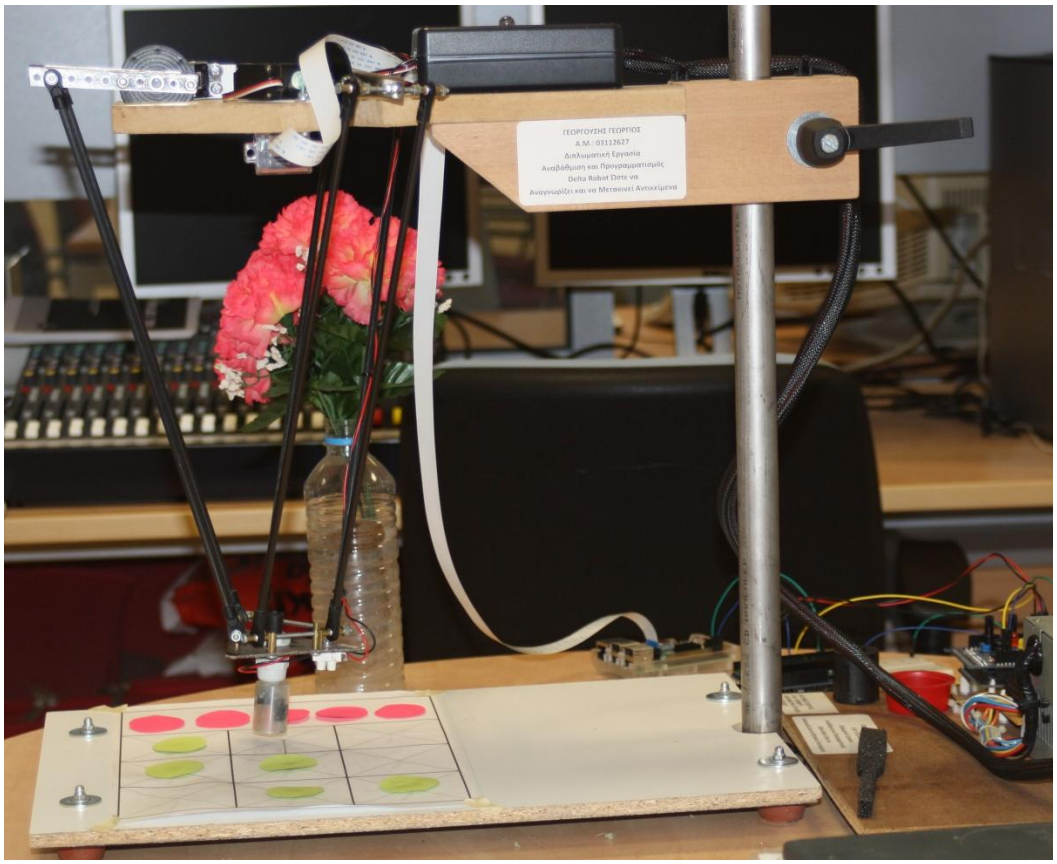
να προβεί στις υπόλοιπες ενέργειες που απαιτούνται για το παιχνίδι, συνεπώς δεν μπορούσε να υλοποιηθεί η διπλωματική εργασία εξολοκλήρου σε αυτόν.

Με τις αλλαγές που έγιναν στο Delta Robot, αυτό πλέον είναι σε θέση να δέχεται συντεταγμένες και εντολές από οποιαδήποτε συσκευή μπορεί να συνδεθεί στη σειριακή θύρα. Άρα μπορεί να επικοινωνήσει με μία ευρεία γκάμα συσκευών, π.χ. Arduino, Raspberry Pi, προσωπικός υπολογιστής κλπ, στις οποίες μπορεί να τρέχει μία τεράστια γκάμα προγραμμάτων. Έτσι είναι δυνατό να χρησιμοποιηθεί σε μία πολύ μεγάλη ποικιλία εφαρμογών.

Στα κεφάλαια που ακολουθούν αναλύεται τόσο η λειτουργία του Delta Robot, κεφάλαιο 1, όσο και ο κώδικας που αφορά το κυρίως μέρος της διπλωματικής εργασίας, κεφάλαιο 2. Στο κεφάλαιο 3 περιγράφεται το τελικό αποτέλεσμα της εργασίας αυτής, προτείνονται πιθανές μελλοντικές εφαρμογές και παρατίθενται γενικότερα σχόλια. Στο τέλος της εργασίας υπάρχουν οι βιβλιογραφικές αναφορές καθώς και οι αναφορές στο διαδίκτυο. Στις διαδικτυακές αναφορές αναφέρεται η διεύθυνση της αρχικής σελίδας κάθε ιστότοπου γιατί όταν έχουν ληφθεί δεδομένα από διάφορες σελίδες του είναι δύσκολο να παρατεθούν όλες λόγω πολύ μεγάλου πλήθους. Έτσι παρατίθεται μόνο η αρχική, πχ. www.arduino.cc, και ο αναγνώστης μπορεί να πλοηγηθεί με ευκολία στο κομμάτι που τον ενδιαφέρει. Αντίθετα, όταν από έναν ιστότοπο έχει αντληθεί περιεχόμενο από μία συγκεκριμένη ιστοσελίδα του μόνο, τότε στην αναφορά στο τέλος παρατίθεται η πλήρης διαδρομή της διεύθυνσής της προκειμένου ο αναγνώστης να έχει εύκολη πρόσβαση σε αυτήν και να μην ψάχνει ξεκινώντας από την αρχική.

Κεφάλαιο 1

Το Delta Robot



Κεφάλαιο 1. Το Delta Robot

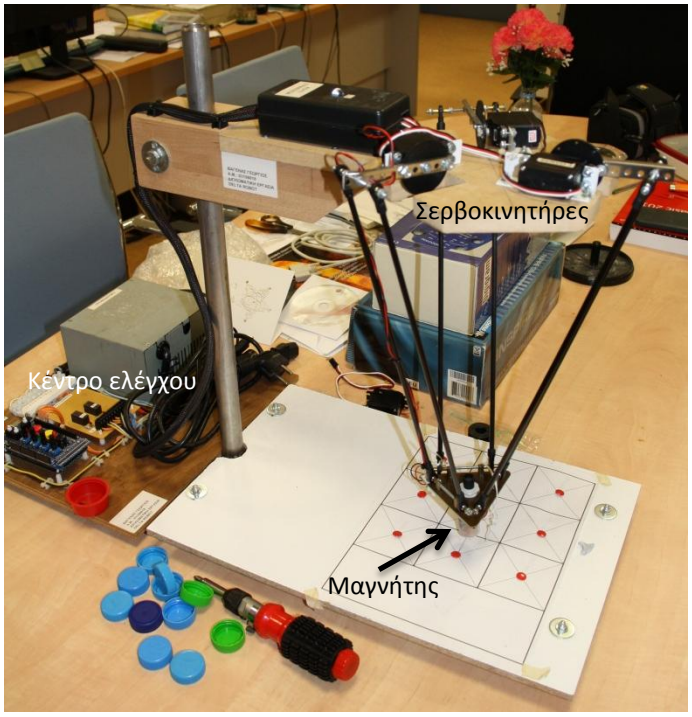
Στο κεφάλαιο αυτό αρχικά παρουσιάζεται το Delta Robot που υπήρχε στο εργαστήριο από προηγούμενη διπλωματική καθώς και οι λειτουργίες του. Κατόπιν παρουσιάζονται οι τροποποιήσεις και προσθήκες που έγιναν σε αυτό προκειμένου να υλοποιηθούν οι σκοποί αυτής της εργασίας. Τέλος παρατίθεται και αναλύεται ο κώδικας που γράφτηκε για τους σκοπούς αυτής της διπλωματικής.

1.1 Περιγραφή του Delta Robot που προϋπήρχε

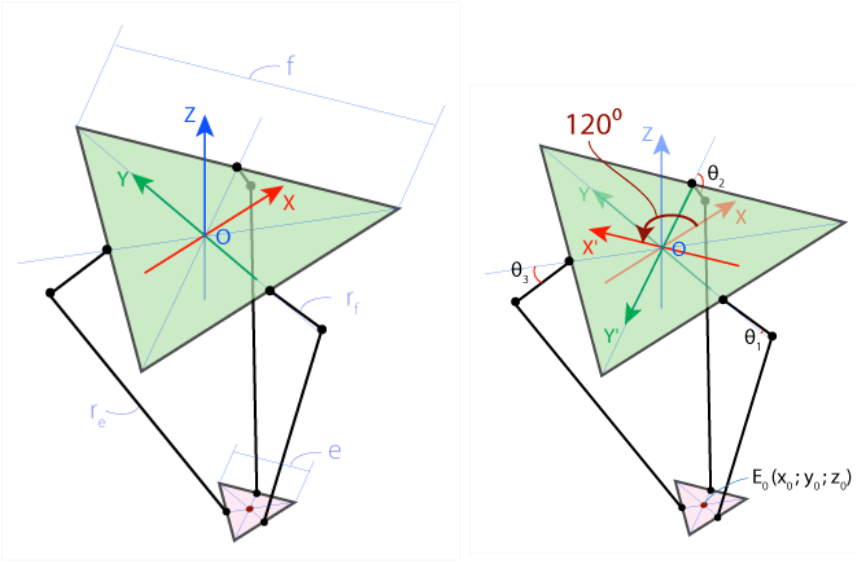
Στο εργαστήριο είχε κατασκευαστεί ένα Delta Robot σαν αποτέλεσμα προηγούμενης διπλωματικής εργασίας (διπλωματική εργασία Γεωργίου Βαγενά), από όπου μπορεί κανείς να αντλήσει περισσότερες πληροφορίες. Αυτό μπορούσε να εκτελεί μία απλή εργασία επίδειξης των δυνατοτήτων του και συγκεκριμένα να μαζεύει 9 μεταλλικές κεφαλές πινεζών από τον πάγκο εργασίας του, ο οποίος είχε χωριστεί σε 9 τετράγωνα διατεταγμένα σε τρεις σειρές και στήλες οριζοντίως και καθέτως. Τις πινέζες αυτές το ρομπότ τις μάζευε μία-μία με τη βοήθεια ηλεκτρομαγνήτη που ήταν προσαρμοσμένος στην κεφαλή του και τις τοποθετούσε μέσα σε ένα ορθογώνιο παραλληλόγραμμο.

Το Delta Robot είναι ένα είδος παράλληλου ρομπότ. Αποτελείται από τρεις βραχίονες που συνδέονται με αρθρώσεις με τη βάση. Οι βραχίονές του κινούνται με τη βοήθεια τριών σερβοκινητήρων SPRINGRC SM-S4315M οι οποίοι ελέγχονται από μία πλακέτα Arduino Mega. Ο ηλεκτροκινητήρας των σερβοκινητήρων είναι συνεχούς ρεύματος, DC, μονίμων μαγνητών, που ενδείκνυται για αυτή τη χρήση λόγω εύκολου ελέγχου της περιστροφής του και μικρής κατανάλωσης ρεύματος (Stephen Umans). Για αυτή την εφαρμογή δεν απαιτούνταν μεγάλη ακρίβεια για να επιλεγεί ένα stepper motor που θα είχε μεγαλύτερη παραγωγή θερμότητας (Μαρία Ιωαννίδου) και επιπλέον θα απαιτούσε πιο περίπλοκο χειρισμό και αντίστοιχο κώδικα συγκριτικά με τον σερβοκινητήρα.

Το βασικό χαρακτηριστικό του Delta Robot είναι ότι οι βραχίονές του κινούνται με τέτοιο τρόπο ώστε καθώς μετακινούν την κεφαλή στις επιθυμητές συντεταγμένες να διατηρείται ο προσανατολισμός του μαγνητικού πεδίου του ηλεκτρομαγνήτη πάντα κάθετος προς το επίπεδο εργασίας. Στην εικόνα 1.1 απεικονίζεται το Delta Robot όπου φαίνεται ο πάγκος εργασίας του, ενώ στην εικόνα 1.2 παρουσιάζεται ένα σκαρίφημά του με τις διαστάσεις. Οι γωνίες θ_i είναι οι γωνίες που σχηματίζει ο κάθε βραχίονας που είναι συνδεδεμένος στον δρομέα του κάθε σερβοκινητήρα με το οριζόντιο επίπεδο και μεταβάλλεται κατά την κίνηση του ρομπότ. Οι βραχίονες αυτοί και συνεπώς οι άξονες των σερβοκινητήρων είναι συμμετρικά τοποθετημένοι γύρω από τον κατακόρυφο άξονα και σχηματίζουν μεταξύ τους γωνία 120 μοιρών.



Εικόνα 1.1. Το Delta Robot.



Εικόνα 1.2. Σκαρίφημα του ρομπότ (Πηγή: Διπλωματική Εργασία Γεωργίου Βαγενά).

Οι τιμές των παραμέτρων είναι:

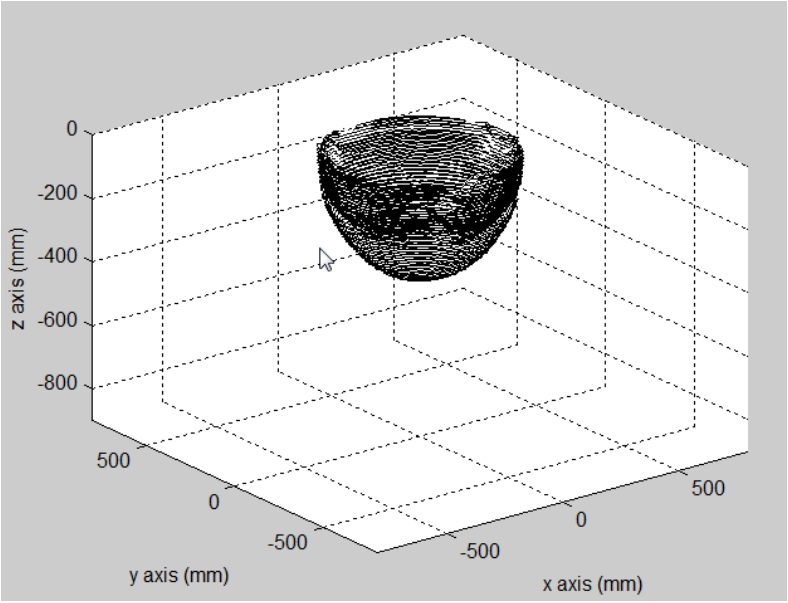
$f = 233mm$ $e = 80mm$ $\theta_{max} = 70°$ ακτίνα $R_{χώρου εργασίας} = 200mm@h = -347mm$

$r_f = 60mm$ $r_e = 350mm$ $\theta_{min} = -110°$ επίπεδο $h_{πάγκου εργασίας} = -347mm$

Εδώ πρέπει να αναφερθεί ότι το σύστημα συντεταγμένων των σερβοκινητήρων δεν συμπίπτει με αυτό του ρομπότ. Δηλαδή όταν ο βραχίονας έχει γωνία θ=0 ως προς το οριζόντιο επίπεδο, στον σερβοκινητήρα αυτή αντιστοιχεί σε 110 μοίρες

περίπου. Αυτό συμβαίνει γιατί ο κάθε σερβοκινητήρας έχει εύρος λειτουργίας από 0 μέχρι 180 μοίρες, ενώ οι γωνίες θ_i παίρνουν και αρνητικές τιμές. Οπότε έπρεπε οι σερβοκινητήρες να τοποθετηθούν υπό τέτοια γωνία ώστε η γωνία θ=0 του συστήματος συντεταγμένων του ρομπότ να αντιστοιχεί σε γωνία 110 μοιρών περίπου ως προς το σύστημα συντεταγμένων του σερβοκινητήρα. Επίσης η θετική φορά μέτρησης της γωνίας για τους σερβοκινητήρες είναι αντίθετη από αυτή των θ_i.

Ένα άλλο σημαντικό στοιχείο είναι ο χώρος εργασίας του ρομπότ. Δηλαδή τα όρια μέσα στα οποία μπορεί να κινηθεί η κεφαλή του. Σύμφωνα με την διπλωματική εργασία του Γεωργίου Βαγενά μπορούσε να κινείται μέσα σε χώρο σχήματος ημισφαιρίου όπως φαίνεται στην εικόνα 1.3. Στην περίπτωσή μας ο πάγκος εργασίας του ήταν στο z=-347mm περίπου, λαμβάνοντας υπ' όψιν και τον ηλεκτρομαγνήτη, και το ρομπότ μπορούσε να καλύψει χώρο ακτίνας περισσότερο από 200mm στο επίπεδο του πάγκου εργασίας.



Εικόνα 1.3. Το πεδίο δράσης του ρομπότ (Πηγή: Διπλωματική Εργασία Γεωργίου Βαγενά).

Τέλος υπάρχει μία βάση που την είχε ονομάσει κέντρο ελέγχου πάνω στην οποία βρίσκονται το τροφοδοτικό, ο Arduino Mega και μία πλακέτα με διάφορα βοηθητικά ηλεκτρονικά κυκλώματα, την οποία είχε ονομάσει πλακέτα εφαρμογών. Από το κέντρο ελέγχου ξεκινάνε καλώδια που καταλήγουν σε ένα κουτί συνδέσεων που βρίσκεται στο κυρίως σώμα του ρομπότ και εκεί συνδέονται με τους σερβοκινητήρες και τον ηλεκτρομαγνήτη. Επειδή το κουτί συνδέσεων βρίσκεται στο υψηλότερο σημείο της κατασκευής, έχει ενσωματωμένη μία μεγάλη λάμπα LED (LED ασφαλείας) που αναβοσβήνει για να προειδοποιεί το χρήστη όταν το ρομπότ είναι σε λειτουργία προκειμένου να αποφευχθεί κάποιος τραυματισμός. Τόσο η λειτουργία αυτού του LED ασφαλείας όσο και του ηλεκτρομαγνήτη ελέγχονται από

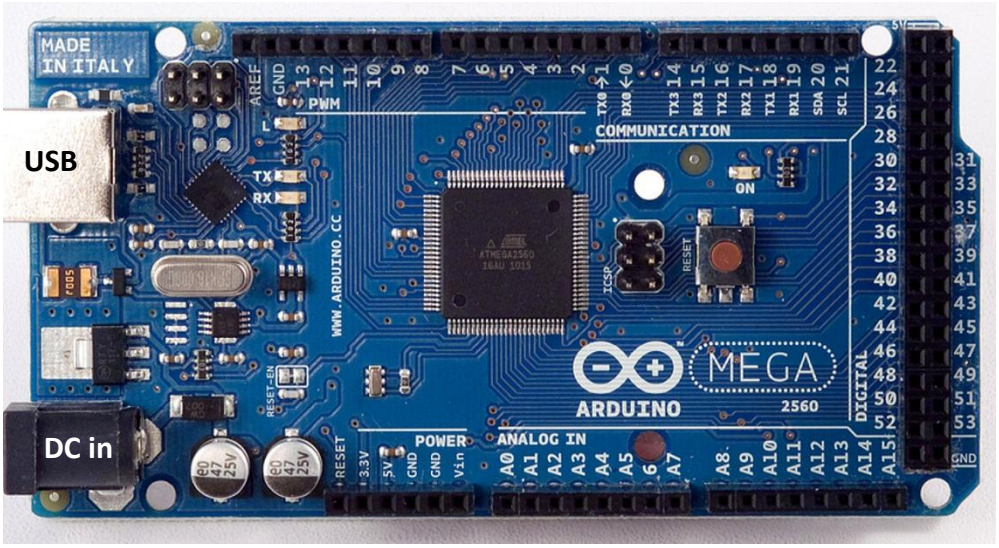
τον Arduino μέσω δύο ηλεκτρονόμων που βρίσκονται στη πλακέτα εφαρμογών. Εκεί υπάρχουν και δύο μικρά LEDs που ενημερώνουν για τη κατάσταση των επαφών αυτών των ηλεκτρονόμων (εντός ή εκτός), άρα, συνεπακόλουθα, και των στοιχείων που αυτά τροφοδοτούν με ρεύμα, δηλαδή του ηλεκτρομαγνήτη και του LED ασφαλείας. Όλη αυτή η διάταξη τροφοδοτείται με ηλεκτρική ισχύ μέσω ενός κατάλληλα διαμορφωμένου τροφοδοτικού ηλεκτρονικού υπολογιστή.

Πάνω στην πλακέτα του Arduino Mega ήταν τοποθετημένη μία διάτρητη πλακέτα (Arduino Shield) όπου μπορούν να κολληθούν διάφορα εξαρτήματα και αυτά να επικοινωνούν με τις εισόδους-εξόδους (pins) του Arduino. Στον Arduino Mega, μεταξύ άλλων, υπάρχει κολλημένο από την κατασκευή ένα SMD LED που είναι μόνιμα συνδεδεμένο με το pin 13. Παρόμοιο LED, αλλά μεγαλύτερο και φωτεινότερο, υπάρχει και στο Shield επίσης συνδεδεμένο με το pin 13. Αυτό έχει κόκκινο χρώμα, ενώ υπάρχει και ένα όμοιο πράσινο που ανάβει όταν υπάρχει τροφοδοσία ρεύματος στο Shield.

Ο κώδικας που ήταν φορτωμένος στον Arduino είχε το εξής σκεπτικό: Το ρομπότ σε κάθε βήμα μάζευε μία πινέζα, την τοποθετούσε σε ένα συγκεκριμένο σημείο στην άκρη και ύστερα μάζευε την επόμενη μέχρι να τις μαζέψει όλες. Κατόπιν επαναλάμβανε από την αρχή την ίδια διαδικασία. Για αυτό το σκοπό υπήρχε ένας πίνακας με τις επιθυμητές καρτεσιανές συντεταγμένες που έπρεπε να πάει ο μαγνήτης καθώς και κατάλληλες συναρτήσεις που υπολόγιζαν τις γωνίες θ_i του κάθε βραχίονα του Delta Robot βάσει των συντεταγμένων αυτών. Δηλαδή οι συναρτήσεις αυτές επίλυαν το αντίστροφο κινηματικό πρόβλημα (inverse kinematics). Κατόπιν υπολογιζόταν η διάρκεια παλμού που αποστελλόταν στον σερβοκινητήρα βάση κάποιου πίνακα βαθμονόμησης που είχε δημιουργηθεί για κάθε έναν σερβοκινητήρα. Επιπλέον, υπήρχε και συνάρτηση για την επίλυση του ευθέος κινηματικού προβλήματος (forward kinematics), δηλαδή για την εύρεση των συντεταγμένων με γνωστές τις γωνίες θ_i. Αυτή δεν ήταν απαραίτητη, ωστόσο υπήρχε για μελλοντική χρήση.

1.2 Λίγα λόγια για τον Arduino

Για τον έλεγχο του Delta Robot χρησιμοποιήθηκε πλακέτα Arduino Mega, που απεικονίζεται στην εικόνα 1.4. Αυτή αποτελείται από έναν μικροελεγκτή ATmega2560 της Atmel, ένα κύκλωμα για επικοινωνία με ηλεκτρονικό υπολογιστή μέσω θύρας USB, κύκλωμα τροφοδοσίας και προστασίας, καθώς και κάποια άλλα βοηθητικά κυκλώματα. Εδώ παρουσιάζεται η πλακέτα περιληπτικά με έμφαση στα σημεία που αφορούν τη συγκεκριμένη διπλωματική, όπως είναι η σειριακή θύρα. Περισσότερες λεπτομέρειες για το προϊόν μπορεί να αντλήσει κανείς από τον επίσημο ιστότοπο του κατασκευαστή, www.arduino.cc.



Εικόνα 1.4. Η πλακέτα Arduino Mega 2560.

Ο μικροελεγκτής που είναι κολλημένος πάνω στην πλακέτα είναι ο ATmega2560 και αποτελείται από επεξεργαστή AVR 8bit RISC αρχιτεκτονικής Harvard χρονισμένο στα 16MHz με μνήμη SRAM 8Kbytes και 256Kbytes μνήμη flash για αποθήκευση του κώδικα. Επίσης πάνω στο chip υπάρχει μνήμη EEPROM 4KB. Μεταξύ άλλων, όπως ψηφιοαναλογικοί μετατροπείς κλπ, υπάρχουν στο chip και 4 USART κυκλώματα για τις σειριακές θύρες επικοινωνίας, USART 0 έως USART 3, τα οποία μπορούν να λαμβάνουν και να στέλνουν δεδομένα ασύγχρονα και ανεξάρτητα από τον κύριο επεξεργαστή. Το καθένα έχει δύο ενδιάμεσες μνήμες, buffer, μία για τα εισερχόμενα και μία για τα εξερχόμενα bytes μεγέθους 64 bytes η κάθε μία.

Έτσι το κάθε chip μόλις λάβει ένα μήνυμα το αποθηκεύει στην αντίστοιχη buffer και ο κύριος επεξεργαστής μπορεί να το διαβάσει σε μεταγενέστερο χρόνο. Ομοίως όταν υπάρχει ένα εξερχόμενο μήνυμα στην buffer, τότε αυτό αναλαμβάνει την αποστολή του, ενώ ο επεξεργαστής μπορεί να συνεχίσει να εκτελεί τις επόμενες εντολές χωρίς να πρέπει να περιμένει να ολοκληρωθεί η αποστολή για να συνεχίσει, εκτός αν του ζητηθεί να περιμένει με κατάλληλη εντολή. Αν γεμίσει η buffer μνήμη εισερχομένων, τότε τα επόμενα bytes που έρχονται θα χαθούν. Επίσης αν γεμίσει η buffer μνήμη εξερχομένων, τότε όταν χρειαστεί να στείλει καινούρια bytes, θα περιμένει μέχρι να αποσταλούν μερικά από τα υπάρχοντα στην buffer προκειμένου να υπάρξει διαθέσιμός χώρος σε αυτήν για τα καινούρια. Έτσι θα καθυστερήσει η εκτέλεση του κώδικα. Σε παρακάτω κεφάλαια που αναφέρονται στην σειριακή μετάδοση δεδομένων αναλύεται πως αντιμετωπίστηκαν αυτά τα προβλήματα.

Πάνω στην πλακέτα υπάρχουν 70 θηλυκοί ακροδέκτες (pins) για είσοδο-έξοδο που είναι συνδεδεμένοι με αντίστοιχους ακροδέκτες εισόδου-εξόδου του μικροελεγκτή. Έτσι επικοινωνεί η πλακέτα με άλλες συσκευές ή εξαρτήματα. Αυτές μπορούν να λειτουργήσουν είτε σαν λογική είσοδος είτε σαν λογική έξοδος κατά το

δοκούν. Επίσης υπάρχουν κάποια επιπλέον pins για την τροφοδοσία με ρεύμα, V_{in} και GND, για επανεκκίνηση, reset pin, καθώς και κάποια για άλλες λειτουργίες. Η τροφοδοσία με ρεύμα μπορεί να γίνει και με βύσμα τύπου barrel jack, καθώς και από τη USB. Η τάση τροφοδοσίας του πρέπει να είναι μεταξύ 7 και 12V όταν τροφοδοτείται από το pin V_{in} ή από το barrel jack, ενώ καλό είναι να μην τροφοδοτείται ταυτόχρονα από δύο πηγές, δηλαδή πχ. από τη USB και τη V_{in}. Τα pins της γης, GND, χρησιμοποιούνται και για να συνδεθούν με την αντίστοιχη γείωση εξωτερικών συσκευών με τις οποίες θέλουμε να επικοινωνήσουμε, προκειμένου να υπάρχει κοινή τάση αναφοράς. Ο Arduino Mega θεωρεί ως λογικό HIGH τα 5V και ως λογικό LOW τα 0V, ωστόσο υπάρχουν κάποιες ανοχές. Κάθε pin εισόδου-εξόδου αντέχει μέχρι ρεύμα 20mA, ενώ υπάρχουν και δύο pins που μπορούν να τροφοδοτήσουν με τάση 5V και 3.3V αντίστοιχα μία εξωτερική συσκευή με μέγιστο ρεύμα τα 50mA. Τέλος υπάρχει ένα SMD LED μόνιμα συνδεδεμένο στο pin 13. Έτσι ο χρήστης μπορεί να επιλέξει πότε αυτό θα ανάψει και πότε θα σβήσει ελέγχοντας την έξοδο 13, δηλαδή αν θα έχει 5V ή 0V αντίστοιχα.

Η κύρια χρήση αυτού του μικροελεγκτή είναι να μπορέσει να ελέγξει άλλα κυκλώματα ή να επικοινωνήσει με άλλες συσκευές ή αισθητήρες επιτρέποντας την υλοποίηση μίας πληθώρας εφαρμογών. Για παράδειγμα μπορεί να τοποθετηθεί το ένα πόδι ενός LED σε ένα pin εξόδου και το άλλο στη γη και να ανάβει ή να σβήνει κατά το δοκούν, ή να τοποθετηθεί ένας διακόπτης σε ένα pin εισόδου κοκ. Άλλη απλή εφαρμογή είναι να συνδεθεί η πύλη ενός MOSFET για έλεγχο κάποιου εξωτερικού κυκλώματος. Σε αυτά τα pins μπορεί να συνδεθεί οποιοδήποτε ψηφιακό κύκλωμα, όπως λογικές πύλες κλπ., αρκεί η τάση λειτουργίας του να είναι 5V και να έχουν κοινή γη (κοινό voltage reference). Αυτά πρακτικά σημαίνουν ότι πρέπει να είναι βραχυκυκλωμένο το pin GND (γη) του Arduino με την αντίστοιχη γη της εξωτερικής συσκευής και το λογικό 1 να είναι στα 5V.

Κάποια από αυτά τα pins εισόδου εξόδου έχουν επιπλέον δυνατότητες. Έτσι 16 από αυτά, τα A0 έως A15, έχουν 10bit αναλογικό-ψηφιακό μετατροπέα, δηλαδή έχουν τη δυνατότητα να διαβάσουν αναλογικό σήμα, από 0 έως 5V, και να το μετατρέψουν σε ψηφιακό με εύρος 10bit δηλαδή το αντιστοιχούν σε έναν ακέραιο αριθμό από το 0 έως και το 1023. Επίσης 15 pins, από 2 έως 13 και από 44 έως 46, έχουν τη δυνατότητα εξόδου PWM παλμών συχνότητας 490Hz και duty cycle που κυμαίνεται από 0% έως 100% όπου το 100% αντιστοιχεί στον αριθμό 255, δηλαδή έχουν ανάλυση 8bit.

Τέλος μέσω κάποιων από τα pins υλοποιείται η επικοινωνία με άλλες συσκευές μέσω συγκεκριμένων σειριακών πρωτοκόλλων. Έτσι έχουμε δυνατότητα SPI, TWI ή I2C επικοινωνία καθώς και 4 σειριακές θύρες UART. Για αυτή την εργασία γίνεται χρήση των UART 0 και 1, δηλαδή των Serial και Serial1 συναρτήσεων αντίστοιχα. Η UART 0 επικοινωνεί με τα pins 0 και 1 με το 0 να είναι για τη λήψη δεδομένων

(receive ή RX) και το 1 για την αποστολή (transmit ή TX). Αυτά είναι συνδεδεμένα εσωτερικά και με το κύκλωμα που αναλαμβάνει την επικοινωνία με υπολογιστή μέσω USB. Έτσι μέσω αυτών μπορεί να στείλει και να λάβει ένα μήνυμα προς και από το PC, αλλά μέσω αυτών γίνεται και ο προγραμματισμός του μικροελεγκτή. Για αυτό το λόγο όταν είναι συνδεδεμένος στον υπολογιστή μέσω USB δεν μπορεί να συνδεθεί κάποια συσκευή στα pins 0 και 1 γιατί δεν μπορεί να επικοινωνεί ταυτόχρονα με δύο συσκευές μέσω του ίδιου UART. Επίσης για τη σειριακή 1, UART 1, χρησιμοποιούνται τα pins 18 για αποστολή (transmit) και 19 για λήψη (receive) δεδομένων.

Ο Arduino Mega προγραμματίζεται μέσω της θύρας USB και κατάλληλου λογισμικού, Arduino IDE, που ενσωματώνει μία τροποποιημένη έκδοση της γλώσσας C/C++ και κατάλληλες βιβλιοθήκες. Αυτό είναι διαθέσιμο στην ιστοσελίδα του κατασκευαστή μαζί με αναλυτική τεκμηρίωση και λεπτομέρειες για το προϊόν (www.arduino.cc). Το περιβάλλον ανάπτυξης του Arduino (Arduino IDE) περιλαμβάνει κατάλληλες συναρτήσεις για τον έλεγχο κάθε μίας εισόδου-εξόδου ξεχωριστά. Δηλαδή μπορεί να καθοριστεί το επίπεδο τάσης σε μία έξοδο ή να διαβαστεί το επίπεδο τάσης από μία είσοδο. Επίσης μπορεί να αποσταλεί μία ακολουθία bytes μέσω της σειριακής θύρας ή να διαβαστεί μία που ήδη έχει παραληφθεί μέσω της σειριακής θύρας. Τέλος υπάρχουν ενσωματωμένες και βιβλιοθήκες για πιο περίπλοκα πράγματα, όπως για τον έλεγχο ενός σερβοκινητήρα ή μιας οθόνης υγρών κρυστάλλων (LCD), ενώ μπορεί κανείς να κατεβάσει και πιο εξειδικευμένες βιβλιοθήκες από την επίσημη ιστοσελίδα τόσο του κατασκευαστή όσο και τρίτων.

Στον μικροελεγκτή αυτόν δεν τρέχει κάποιο λειτουργικό σύστημα παρά μόνο κάποιος κώδικας στην εκκίνηση, Bootloader, για να γίνει ο προγραμματισμός εύκολα μέσω της USB. Μετά την ολοκλήρωση του προγραμματισμού τρέχει μόνο ο κώδικας που έγραψε ο χρήστης και τυχόν κώδικας από βιβλιοθήκες που έχει συμπεριλάβει. Η απουσία λειτουργικού συστήματος σε συνδυασμό με το ότι το μέγεθος της μνήμης και η ταχύτητα του επεξεργαστή είναι πολύ μικρότερα από αυτά ενός συνηθισμένου μικροϋπολογιστή έχουν αντίκτυπο στο κώδικα που μπορεί να τρέξει και στις τεχνικές προγραμματισμού που μπορούν να χρησιμοποιηθούν (Wayne Wolf), πχ. αποφυγή χρήσης δυναμικής διαχείρισης μνήμης (malloc) κλπ.

Ο κώδικας που καλείται να γράψει ο χρήστης αποτελείται από δύο τουλάχιστον συναρτήσεις, την void setup(void) και τη void loop(void) και βέβαια όσες άλλες επιθυμεί, οι οποίες μπορούν να κληθούν μέσα από αυτές τις δύο. Η setup() εκτελείται μία φορά στην αρχή μετά την τροφοδοσία με ρεύμα ή την επανεκκίνηση και συνήθως περιλαμβάνεται κώδικας αρχικοποίησης όπως δήλωση pins ως εισόδων ή εξόδων άνοιγμα της σειριακής επικοινωνίας κλπ. Η loop() εκτελείται συνέχεια μέχρι την διακοπή της τροφοδοσίας με ρεύμα και συνήθως εκεί γράφεται

το κυρίως κομμάτι του κώδικα. Αυτές οι δύο συναρτήσεις πρέπει υποχρεωτικά να υπάρχουν στον πηγαίο κώδικα κάθε έργου ακόμα και αν είναι άδειες, αλλιώς δεν θα γίνει μεταγλώττιση. Επειδή αυτές οι δύο δεν επιστρέφουν τίποτα και δεν έχουν ορίσματα, αν πρέπει να επικοινωνήσουν μεταξύ τους αυτό γίνεται με καθολικές (global) μεταβλητές οι οποίες συνήθως δηλώνονται στην αρχή μετά από τυχόν οδηγίες προς τον προεπεξεργαστή.

Αντίθετα, δεν υπάρχει συνάρτηση main() όπως συμβαίνει σε ένα τυπικό πρόγραμμα C/C++. Για την ακρίβεια υπάρχει αλλά δεν την γράφει ο χρήστης, αλλά προστίθεται αυτόματα από το περιβάλλον ανάπτυξης όταν ο χρήστης ζητήσει να γίνει μεταγλώττιση του κώδικα και προγραμματισμός του μικροελεγκτή. Αυτή περιλαμβάνει μεταξύ άλλων μία κλήση της setup() και έναν ατέρμονα βρόγχο κατά τον οποίο γίνεται κλήση της loop() (δηλαδή for(;;){loop();}). Για αυτό και ο χρήστης δεν πρέπει να γράψει συνάρτηση με το όνομα main().

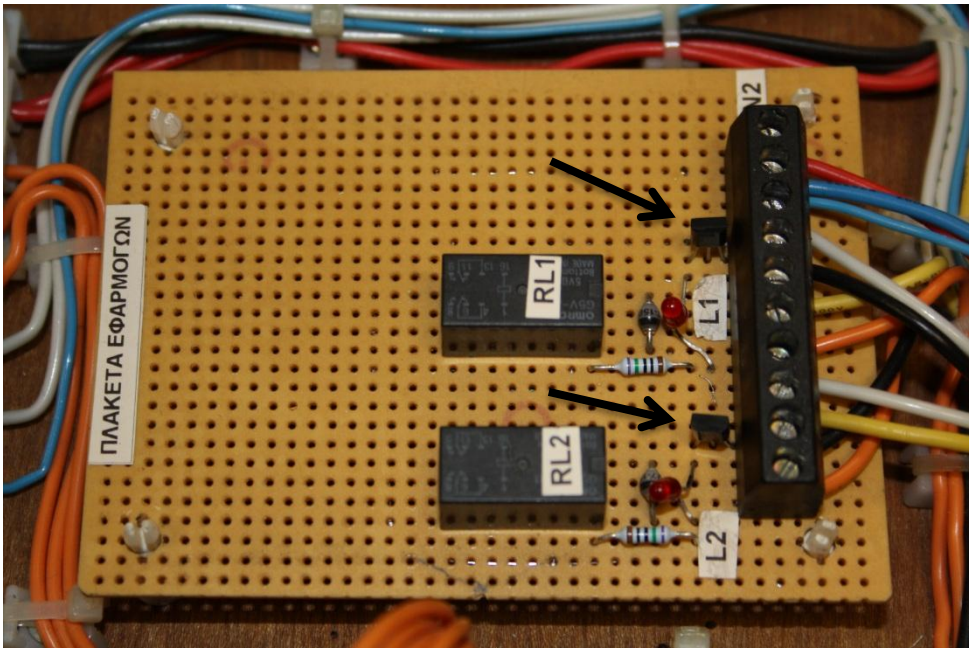
Ένα άλλο χρήσιμο χαρακτηριστικό που συνοδεύει το Arduino IDE είναι το σειριακό τερματικό (Serial Terminal). Αυτό έχει πρόσβαση στην σειριακή θύρα COM που επικοινωνεί μέσω USB με τον Arduino. Σε αυτό τυπώνεται οποιοδήποτε μήνυμα στέλνει ο Arduino στο PC μέσω της USB αρκεί να αποτελείται από εκτυπώσιμους ASCII χαρακτήρες. Επίσης υπάρχει πεδίο όπου ο χρήστης μπορεί να εισάγει από το πληκτρολόγιο κάποιο μήνυμα και να το στείλει στον Arduino. Βέβαια στον κώδικα που τρέχει στον Arduino πρέπει να υπάρχουν οι αντίστοιχες εντολές για σειριακή αποστολή ή και λήψη δεδομένων.

Τέλος μία άλλη πλακέτα της ίδιας εταιρίας που χρησιμοποιήθηκε είναι ο Arduino UNO. Αυτός χρησιμοποιήθηκε συνεπικουρικά και σκοπό είχε να τυπώνει σε μία οθόνη υγρών κρυστάλλων την έξοδο του Delta Robot. Περισσότερα για την έξοδο αναλύονται σε επόμενα κεφάλαια. Για τον Arduino UNO ισχύουν ότι έχει αναλυθεί παραπάνω για τον Mega με κάποιες διαφορές ως προς τις δυνατότητές του. Έτσι έχει λιγότερα pins, 20, μικρότερο μέγεθος και μία σειριακή USART. Ο μικροελεγκτής που υπάρχει στην πλακέτα είναι ATmega328P της Atmel. Αυτός έχει 32KBytes μνήμης flash, 2KB μνήμης SRAM, 1KB μνήμης EEPROM και 8bit RISC μικροεπεξεργαστή AVR χρονισμένο στα 16MHz αρχιτεκτονικής Harvard.

1.3 Οι τροποποιήσεις και οι προσθήκες που έγιναν στο Delta Robot

Για την υλοποίηση αυτής διπλωματικής χρειάστηκε να γίνουν κάποιες μικρές τροποποιήσεις στο υπάρχων ρομπότ. Κατ' αρχήν, το πηνίο του κάθε ηλεκτρονόμου τροφοδοτούταν με ρεύμα κατ' ευθείαν από την αντίστοιχη ψηφιακή έξοδο (pin) του Arduino. Όμως το ρεύμα που τραβάει το κάθε πηνίο είναι σχετικά κοντά στο μέγιστο ρεύμα που αντέχει η κάθε ψηφιακή έξοδος (pin) του Arduino. Γι' αυτό κρίθηκε σκόπιμο το κάθε ένα πηνίο να τροφοδοτηθεί με ρεύμα από το τροφοδοτικό μέσω

ενός MOSFET, η πύλη του οποίου να ελέγχεται από τον Arduino. Έτσι εξασφαλίζεται ότι θα περάσουν πολύ μικρές ποσότητες ρεύματος από τις αντίστοιχες εξόδους του Arduino. Γι' αυτό το λόγο στην πλακέτα εφαρμογών τοποθετήθηκαν 2 MOSFETs, ένα για κάθε ηλεκτρονόμο, όπως φαίνεται στην εικόνα 1.5.



Εικόνα 1.5. Η πλακέτα εφαρμογών στην τελική της μορφή μετά την τοποθέτηση των MOSFETs.

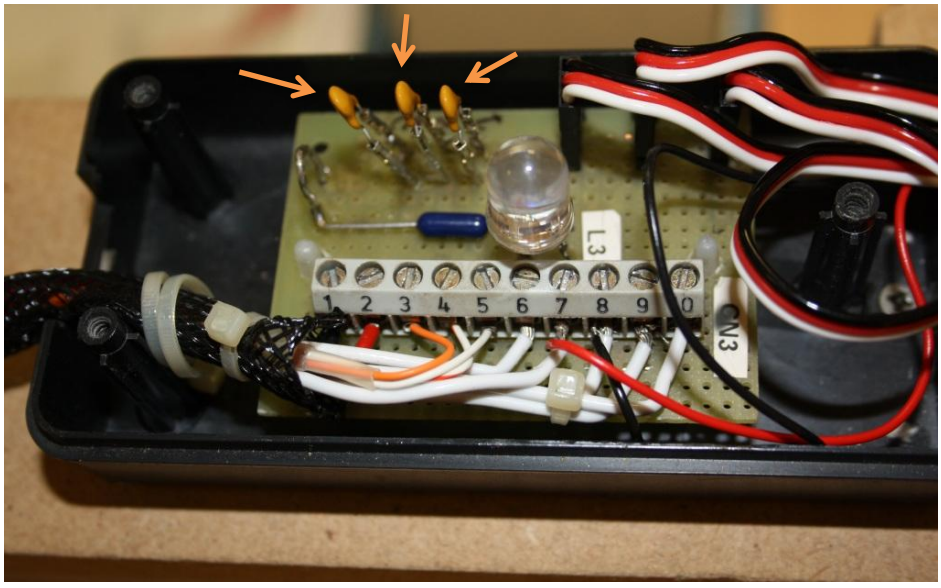
Μία άλλη τροποποίηση που χρειάστηκε να γίνει ήταν η προσθήκη ηλεκτρικής ασφάλειας στους σερβοκινητήρες. Αυτοί τροφοδοτούνταν με ρεύμα από το τροφοδοτικό χωρίς να μεσολαβεί κάποια ηλεκτρική ασφάλεια ούτε είχαν ενσωματωμένη κάποια ασφάλεια ή κάποιο άλλο είδος προστασίας, σε αντίθεση με τον ο Arduino που έχει ενσωματωμένα κυκλώματα προστασίας. Η ασφάλεια που είναι ενσωματωμένη στο τροφοδοτικό διακόπτει το κύκλωμα για ρεύματα πολύ μεγαλύτερα από αυτά που μπορούν να αντέξουν οι σερβοκινητήρες.

Μετρήθηκε ότι το ρεύμα ακινητοποιημένου δρομέα, πχ. λόγω μεγάλου μηχανικού φορτίου που δεν τον αφήνει να πάει στην επιθυμητή θέση, είναι περίπου 2A, τιμή που μπορεί να υποστηρίξει το τροφοδοτικό, αλλά όμως μπορεί να δημιουργήσει πρόβλημα ή και καταστροφή στον σερβοκινητήρα, κυρίως αν αυτό τον διαρρέει για παρατεταμένο χρονικό διάστημα. Υπό κανονικές συνθήκες το ρεύμα που διαρρέει τον σερβοκινητήρα μετρήθηκε ότι κυμαίνεται μεταξύ 50 με 500mA αναλόγως με το μηχανικό φορτίο και αν είναι στάσιμος ο δρομέας ή κινείται. Έτσι κρίθηκε σκόπιμο η τοποθέτηση τριών ασφαλειών, μία για κάθε έναν, των 750mA, τύπου polyfuse.

Αυτές λειτουργούν κυρίως θερμικά. Δηλαδή όταν η θερμοκρασία υπερβεί κάποια τιμή, αυξάνεται πάρα πολύ η ωμική αντίσταση της ασφάλειας, ενώ όταν

μειωθεί μειώνεται και η αντίσταση. Συνεπώς, όταν το ρεύμα που την διαρρέει υπερβεί κάποια τιμή για κάποιο μικρό χρονικό διάστημα, πχ. εξ' αιτίας της ακινητοποίησης του δρομέα, τότε, παρά την μικρή της αντίσταση, παράγεται σ' αυτήν αρκετή θερμότητα ώστε να αυξήσει την ωμική της αντίσταση σημαντικά, της τάξης των kΩ, περιορίζοντας δραστικά το ρεύμα. Παρά το ότι η τιμή του ρεύματος είναι πλέον πολύ μικρή, η παραγωγή θερμότητας στην ασφάλεια λόγω της μεγάλης της αντίστασης είναι αρκετή ώστε να διατηρηθεί η υψηλή της θερμοκρασία, άρα και η μεγάλη αντίσταση. Μετά την άρση του γεγονότος που οδήγησε σε μεγάλο ρεύμα, πχ. μετά την άρση της ακινητοποίησης του δρομέα, το ρεύμα πλέον είναι τόσο μικρό που δεν μπορεί να διατηρηθεί η θερμοκρασία σε υψηλά επίπεδα, οπότε αρχίζει να ψύχεται με αποτέλεσμα η ασφάλεια να επανέρχεται σε κανονική κατάσταση λειτουργίας, δηλαδή με πάρα πολύ μικρή ωμική αντίσταση, της τάξης των mΩ. Έτσι η πτώση τάσης στα άκρα της είναι αμελητέα και το κύκλωμα που προστατεύει τροφοδοτείται πρακτικά με την ονομαστική τιμή τάσης, στην περίπτωσή μας 5V.

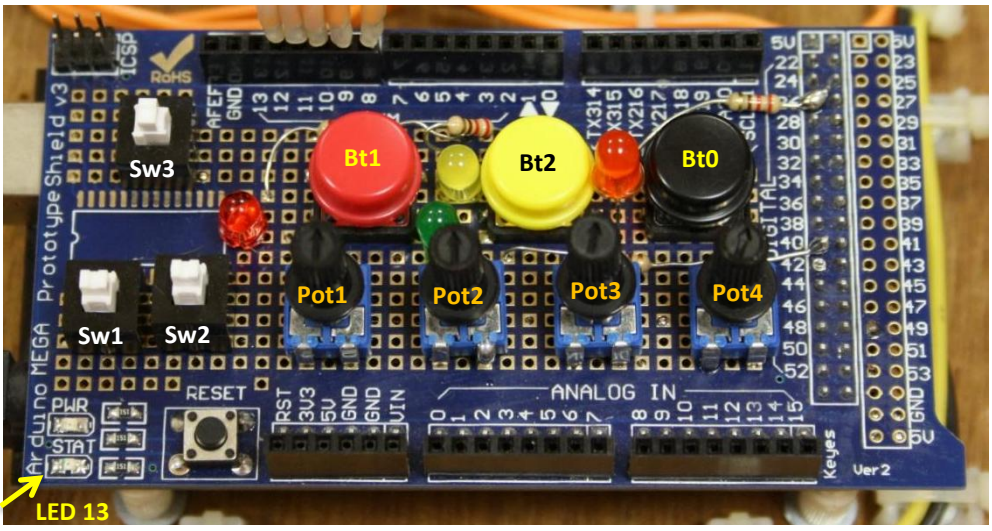
Αυτό μπορεί να επαναληφθεί πολλές φορές στη διάρκεια της ζωής μιας τέτοιας ασφάλειας. Παρ' όλα αυτά αυτές δεν κολλήθηκαν στην πλακέτα, αλλά κολλήθηκαν βάσεις στις οποίες τοποθετήθηκαν οι ασφάλειες κουμπωτά, έτσι ώστε να μπορούν να αντικατασταθούν αν χρειαστεί όπως φαίνεται στην εικόνα 1.6. Στην εικόνα 1.7 απεικονίζονται και οι σερβοκινητήρες που βρίσκονται στο πάνω μέρος της κατασκευής και δίπλα τους το κουτί συνδέσεων της εικόνας 1.6.



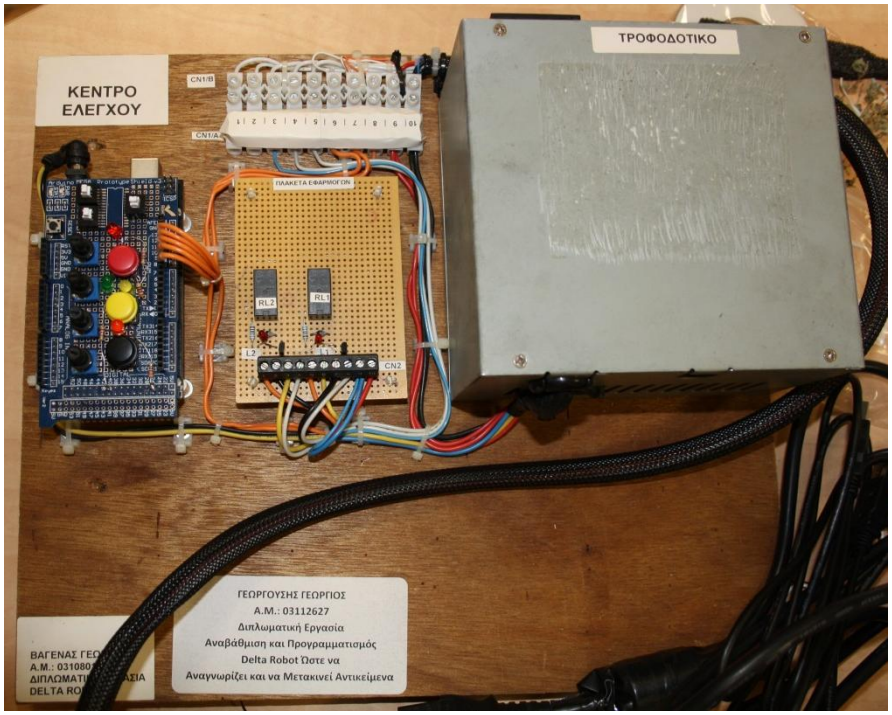
Εικόνα 1.6. Το κουτί συνδέσεων στην τελική του μορφή μετά την τοποθέτηση των ασφαλειών.

Εικόνα 1.7. Οι τρεις σερβοκινητήρες.

Τέλος, στην διάτρητη πλακέτα Arduino Shield που ήταν τοποθετημένη πάνω στον Arduino Mega κολλήθηκαν κατάλληλα ποτενσιόμετρα, διακόπτες και LEDs για να μπορεί να ελέγχεται το ρομπότ τοπικά από τον χρήστη όταν αυτός το επιθυμεί. Εκεί κολλήθηκαν και λάμπες LED που πληροφορούν τον χρήστη για διάφορες παραμέτρους που αφορούν τη λειτουργία του, όπως απεικονίζεται στην εικόνα 1.8. Στην εικόνα 1.9 απεικονίζεται το κέντρο ελέγχου στην τελική του μορφή. Περισσότερες λεπτομέρειες παρουσιάζονται στις παραγράφους που ακολουθούν. Στην εικόνα 1.10 απεικονίζεται ο ηλεκτρομαγνήτης την στιγμή που σηκώνει μία μεταλλική κεφαλή πινέζας.



Εικόνα 1.8. Το Arduino Shield στην τελική του μορφή.



Εικόνα 1.9. Το κέντρο ελέγχου στην τελική του μορφή.

Εικόνα 1.10. Ο ηλεκτρομαγνήτης που σηκώνει μία μεταλλική κεφαλή πινέζας.

1.4 Η λειτουργία του Delta Robot

Για την υλοποίηση του κυρίως κομματιού αυτής της διπλωματικής εργασίας έπρεπε το Delta Robot να επικοινωνεί με κάποια εξωτερική συσκευή προκειμένου να δέχεται τις συντεταγμένες και τη κατάσταση του ηλεκτρομαγνήτη. Αυτό ήταν απαραίτητο γιατί ο Arduino δεν έχει τις δυνατότητες που απαιτούνται για να γίνει η υλοποίηση εξ ολοκλήρου σ' αυτόν. Επιπλέον καλό ήταν σαν αποτέλεσμα αυτής της εργασίας το Delta Robot στην τελική του μορφή να έχει μία μορφή πολυλειτουργικότητας και να μην είναι μόνο για μία συγκεκριμένη χρήση. Δηλαδή είναι καλύτερη λύση να μπορεί να επικοινωνεί με οποιαδήποτε συσκευή για να έχει ένα μεγάλο εύρος εφαρμογών παρά να κάνει μόνο μία δουλειά, πχ μόνο να παίζει τρίλιζα. Όμως για την περίπτωση που δεν θα ήταν συνδεδεμένη κάποια εξωτερική συσκευή καθώς και κατά την ανάπτυξη κάποιας εφαρμογής, πχ. αυτής της διπλωματικής εργασίας, κρίθηκε ότι ήταν σκόπιμο αλλά και απαραίτητο να έχει επιπλέον τη δυνατότητα να δέχεται εντολές κατευθείαν από το χρήστη μέσω κάποιων χειριστηρίων. Έτσι το ρομπότ πλέον μπορεί να λειτουργεί σε τρεις καταστάσεις λειτουργίας (modes), το mode 0, το mode 1 και το mode 2 που εναλλάσσονται όποτε ο χρήστης το επιθυμεί. Στο mode 0 το ρομπότ δέχεται εντολές από εξωτερική συσκευή, στην περίπτωσή μας από το Raspberry Pi. Στο mode 1 το ρομπότ ακινητοποιείται και σταματά τη λειτουργία του μέχρι ο χρήστης να αλλάξει mode. Τέλος στο mode 2 το ρομπότ δέχεται εντολές από το χρήστη μέσω των χειριστηρίων. Η αρχική λειτουργία είναι το mode 0. Δηλαδή μετά την εκκίνηση μπαίνει σε mode 0 και περιμένει οδηγίες απομακρυσμένα.

1.5 Τρόπος επικοινωνίας του Delta Robot με εξωτερική συσκευή

Το Delta Robot για να λαμβάνει τις συντεταγμένες επικοινωνεί με την εξωτερική συσκευή σειριακά μέσω της σειριακής θύρας 1, UART 1, του Arduino Mega, δηλαδή τα pins 18 (Transmit) και 19 (Receive), κάνοντας χρήση του ενσωματωμένου κυκλώματος UART (ή USART) με BAUD rate 9600. Η επικοινωνία γίνεται ασύγχρονα χωρίς να ελέγχει ο αποστολέας αν ο παραλήπτης έχει διαβάσει ή όχι τα bytes που έστειλε. Αν γεμίσει η ενδιάμεση μνήμη (buffer) των εισερχομένων τότε τα υπόλοιπα που έρχονται χάνονται. Γι' αυτό το κάθε μήνυμα δεν πρέπει να έχει μέγεθος μεγαλύτερο από 64 bytes, και η εξωτερική συσκευή δεν πρέπει να στέλνει συνεχώς μηνύματα, αλλά να περιμένει πρώτα να διαβαστεί το κάθε μήνυμα πριν στείλει το επόμενο.

Έτσι αποφασίστηκε η επικοινωνία να γίνεται ως εξής:

1. Η εξωτερική συσκευή στέλνει ένα μήνυμα στο ρομπότ. Μόλις αυτό το λάβει εκτελεί την εντολή, δηλαδή τοποθετεί την κεφαλή στις συντεταγμένες που έλαβε και αν δεν υπάρχει άλλο μήνυμα στην buffer μνήμη στέλνει τον χαρακτήρα «@»

(ASCII 64). Τότε η εξωτερική συσκευή μπορεί να προχωρήσει στην αποστολή του επόμενου μηνύματος. Δηλαδή η εξωτερική συσκευή πρέπει να περιμένει μέχρι να λάβει το «@» προκειμένου να προχωρήσει στην αποστολή κάποιου μηνύματος. Έτσι δεν θα έρχονται μηνύματα με ρυθμό μεγαλύτερο από αυτόν που μπορεί να τα διαβάσει και εκτελέσει το ρομπότ.

2. Το εισερχόμενο μήνυμα έχει την εξής δομή: Στην αρχή του μηνύματος πρέπει να υπάρχει ο χαρακτήρας «@» (ASCII 64), κατόπιν να ακολουθούν οι ASCII κωδικοί των ψηφίων της συντεταγμένης x, ύστερα ένας τουλάχιστον ASCII χαρακτήρας που δεν είναι αριθμητικό ψηφίο ούτε τα «@» και «$», μετά οι ASCII κωδικοί των ψηφίων της συντεταγμένης y, ύστερα ένας τουλάχιστον ASCII χαρακτήρας που δεν είναι αριθμητικό ψηφίο ούτε τα «@» και «$», μετά ομοίως για την συντεταγμένη z, μετά μη αριθμητικός χαρακτήρας, ύστερα ο ASCII κωδικός του 0 ή του 1 (το 48 ή 49 αντίστοιχα) που αφορά την κατάσταση του ηλεκτρομαγνήτη, on ή off αντίστοιχα, και τέλος ο χαρακτήρας «$» (ASCII 36). Το συνολικό μήκος του εισερχόμενου μηνύματος δεν πρέπει να ξεπερνάει τα 64 bytes.

Οι συντεταγμένες x,y,z είναι ακέραιες ποσότητες και είναι οι επιθυμητές συντεταγμένες σε cm πολλαπλασιασμένες με 100. Δηλαδή για το σημείο (x,y,z)=(-10.05,4.54,-32.00) θα πρέπει να αποσταλούν οι αριθμοί (-1005,454,-3200). Άρα όλη η ακολουθία αν θέλουμε τον μαγνήτη εκτός (off) θα είναι:

@-1005,454,-3200,0$

Οπότε οι αντίστοιχοι ASCII κωδικοί, που τελικά θα μεταβιβαστούν για το παραπάνω μήνυμα θα είναι:

64 45 49 48 48 53 44 52 53 52 44 45 51 50 48 48 44 48 36

Ή τα αντίστοιχα bytes σε δυαδική μορφή θα είναι:

01000000 00101101 00110001 00110000 00110000 00110101 00101100 00110100 00110101 00110100 00101100 00101101 00110011 00110010 00110000 00110000 00101100 00110000 00100100

Ανάμεσα στα ψηφία των αριθμών πρέπει να υπάρχει τουλάχιστον ένας μη αριθμητικός χαρακτήρας εκτός από τους «@» και «$» για να ξεχωρίζουν μεταξύ τους οι διαφορετικοί αριθμοί. Σε αντίθετη περίπτωση θα εκληφθούν όλοι σαν ένας αριθμός και δεν θα μπορεί να εξαχθεί συμπέρασμα για το που χωρίζονται μεταξύ τους. Επίσης μπορούν να διαχωρίζονται από περισσότερους από έναν χαρακτήρες είτε αλφαβητικούς είτε κενά κλπ είτε συνδυασμούς τους, αλλά όχι αριθμητικούς ούτε τα «@» και «$». Για παράδειγμα το παρακάτω μήνυμα είναι αποδεκτό:

@ X=1015, Y=1354, Z=-3321, Electromagnet=1, Everything is OK. $

Θα εκληφθεί ως (x,y,z)=(10.15,13.54,-33.21) και ο ηλεκτρομαγνήτης εντός (on).

Το παρακάτω μήνυμα δεν είναι αποδεκτό γιατί δεν υπάρχει διαχωρισμός μεταξύ των αριθμών:

@10151354-3321$

Το παρακάτω θα έχει διαφορετικά αποτελέσματα από τα επιθυμητά γιατί υπάρχει η τελεία μεταξύ του 10 και του 15 και τους διαχωρίζει:

@ X=10.15 Y=1354, Z=-3321, Electromagnet=1, OK. $

Έτσι θα εκληφθεί ως (x,y,z)=(0.1,0.15,13.54) και μαγνήτης=-3321, ενώ τα υπόλοιπα θα αγνοηθούν. Το παρακάτω μήνυμα επίσης είναι μη αποδεκτό:

@ X1=1015, Y2=1354, Z3=-3321, Electromagnet=1. $

Θα το εκλάβει ως (x,y,z)=(0.01,10.15,0.02) και μαγνήτης=1354 εξαιτίας των 1 και 2 δίπλα από τα X και Y αντίστοιχα.

Επίσης η ύπαρξη των «@» και «$» μέσα στο μήνυμα θα δημιουργήσει πρόβλημα γιατί αυτοί οι δύο χαρακτήρες είναι για την αρχή και το τέλος του μηνύματος. Τέλος αν απουσιάζει ένας από τους «@» ή «$» ή και οι δύο από την αρχή και το τέλος αντίστοιχα, τότε θα αγνοηθεί το μήνυμα. Αυτοί οι χαρακτήρες λειτουργούν σαν σύστημα αναφοράς έτσι ώστε να ξεχωρίζει ποιος αριθμός είναι ο πρώτος ποιος ο δεύτερος κλπ, άρα ποιος αριθμός αναπαριστά ποιο μέγεθος. Δηλαδή ο πρώτος είναι το x, ο δεύτερος το y κοκ.

Ο λόγος που αποφασίστηκε να χρησιμοποιηθούν τα «@» (ASCII 64) και «$» (ASCII 36) σαν χαρακτήρες αρχής και τέλους, αντί για τους ASCII κωδικούς που υπάρχουν για αυτό το λόγο, είναι επειδή κατά την ανάπτυξη η αποσφαλμάτωση γίνεται ευκολότερα καθότι οι «@» και «$» είναι εκτυπώσιμοι χαρακτήρες και υπάρχουν και στο πληκτρολόγιο. Οι λόγοι που αποφασίστηκε να μεταβιβάζονται οι ASCII κωδικοί των ψηφίων των αριθμών και όχι οι ίδιοι οι αριθμοί σε δυαδική απεικόνιση είναι δύο. Ο πρώτος είναι ότι η εξωτερική συσκευή μπορεί να έχει διαφορετική απεικόνιση από τον Arduino, πχ. περισσότερα bytes για έναν αριθμό, περιπλέκοντας τον κώδικα που θα έπρεπε να γραφτεί. Ο δεύτερος είναι για να μπορούν να εκτυπωθούν σε ένα σειριακό τερματικό έτσι ώστε να γίνει ευκολότερα η αποσφαλμάτωση κατά την ανάπτυξη της εφαρμογής.

1.6 Ο χειρισμός του Delta Robot

Στο Arduino Shield έχουν κολληθεί τρία μεγάλα κυκλικά κουμπιά για να επιλέγει ο χρήστης μεταξύ των τριών modes λειτουργίας του ρομπότ. Το μαύρο κουμπί, Bt0, αφορά το mode 0, το κόκκινο, Bt1, το mode 1 και το κίτρινο, Bt2, το

mode 2. Επίσης έχουν κολληθεί τρία LED για να πληροφορείται ο χρήστης το mode λειτουργίας. Το πράσινο ανάβει στο mode 0, το κόκκινο στο mode 1 και το κίτρινο στο mode 2. Τέλος έχουν κολληθεί τρεις διακόπτες, Sw1 έως 4, τέσσερα ποτενσιόμετρα, Pot 1 έως 4, και ένα πορτοκαλί LED, ενώ γίνεται και χρήση του ενσωματωμένου SMD LED (LED 13). Για την χρήση τους δίνονται αναλυτικές πληροφορίες παρακάτω.

1.6.1 Το Mode 0

Στο mode 0 το ρομπότ δέχεται συντεταγμένες από εξωτερική συσκευή, απομακρυσμένος χειρισμός, πχ. από το Raspberry Pi. Οι καρτεσιανές συντεταγμένες λαμβάνονται μέσω της σειριακής θύρας 1 (Serial 1, pins 18 και 19) μαζί με την κατάσταση του ηλεκτρομαγνήτη, εντός ή εκτός, όπως αναλύθηκε προηγουμένως. Το LED ασφαλείας δεν μπορεί να ελεγχθεί εξωτερικά και αναβοσβήνει μόνιμα. Μόλις λάβει κάποιο μήνυμα από τη σειριακή θα σβήσει το πορτοκαλί LED. Όταν εκτελέσει την εντολή και δεν υπάρχουν άλλα δεδομένα στην ενδιάμεση μνήμη, τότε θα ανάψει και πάλι το πορτοκαλί LED. Δηλαδή αυτό το LED πληροφορεί το χρήστη ότι αναμένει νέο μήνυμα.

Αν οι συντεταγμένες που έλαβε παραπέμπουν σε σημείο εκτός του χώρου εργασίας του Delta Robot που εικονίζεται στην εικόνα 1.3 ή εκτός του εύρους των γωνιών είτε των θ_i είτε αυτών των σερβοκινητήρων, τότε δεν εκτελεί την εντολή ενώ παράλληλα σβήνει και το ενσωματωμένο SMD LED 13 για να πληροφορήσει το χρήστη για αυτό το γεγονός. Όταν ληφθούν συντεταγμένες που παραπέμπουν σε έγκυρο σημείο, τότε μόνο εκτελεί την κίνηση και ανάβει το LED 13.

Στο επίπεδο z=-34.77 περίπου βρίσκεται ο πάγκος εργασίας του ρομπότ. Αυτός είναι μέσα στον ημισφαιρικό χώρο που εικονίζεται στην εικόνα 1.3. Συνεπώς για να μην ακουμπήσει η κεφαλή στον πάγκο με δύναμη υπάρχει ακόμα ένας περιορισμός, ώστε αν η συντεταγμένη z είναι μικρότερη από -34.77, τότε την αντικαθιστά με την τιμή -34.77 και πηγαίνει στο επίπεδο του πάγκου. Δηλαδή σε αυτή την περίπτωση εκτελεί την κίνηση, αλλά δεν πηγαίνει στο επίπεδο που ζητήθηκε, αλλά στο οριακό. Και εδώ σβήνει το LED 13 όσο το z είναι μικρότερο από την οριακή τιμή και ανάβει όταν επανέλθει σε μεγαλύτερες τιμές. Επειδή μπορεί να υπάρχουν αντικείμενα στον πάγκο, δίνεται η δυνατότητα στον χρήστη να αλλάξει αυτήν την τιμή σε μεγαλύτερη, πιο ψηλότερο επίπεδο, και να αποθηκεύσει τη νέα τιμή στην μνήμη EEPROM του Arduino για μελλοντική χρήση.

Αν πατηθεί το μαύρο κουμπί ενώ το ρομπότ είναι ήδη σε mode 0 και έχει περάσει περισσότερο από 1 δευτερόλεπτο από τη στιγμή που είχε προηγουμένως πατηθεί και αφεθεί, τότε το ρομπότ στέλνει τον χαρακτήρα «@» στην εξωτερική συσκευή, πχ. στο Raspberry Pi. Το ένα δευτερόλεπτο μετράει από τη στιγμή που

αφήνεται το κουμπί και έχει το νόημα να μην γίνεται μαζική αποστολή πολλών «@» λόγω παρατεταμένου πατήματος του κουμπιού. Αυτό έχει πρακτική εφαρμογή στην περίπτωση που το ρομπότ έχει στείλει κανονικά το «@» και περιμένει το μήνυμα, άλλα για κάποιο λόγο το Raspberry Pi δεν έχει λάβει το «@» και το περιμένει, πχ. όταν αποσυνδεθεί το καλώδιο ή δεν είχε ξεκινήσει η εφαρμογή όταν έγινε η αποστολή του. Δηλαδή έτσι γίνεται ξανά ο συγχρονισμός των δύο συσκευών. Ο χρήστης καταλαβαίνει ότι έχει προκύψει αυτό το πρόβλημα όταν βλέπει το πορτοκαλί LED αναμμένο στο ρομπότ και το Raspberry Pi να τυπώνει μήνυμα στην οθόνη ότι το ρομπότ δεν αποκρίνεται, ενώ στην πραγματικότητα αυτό έχει αποκριθεί, ειδάλλως δεν θα είχε ανάψει το πορτοκαλί LED.

1.6.2 Το Mode 1

Το mode 1 είναι παύση λειτουργίας του Robot. Ο ηλεκτρομαγνήτης και το LED ασφαλείας σβήνουν και η κεφαλή του μεταβαίνει στη θέση (x,y,z)=(0,0,-33) και δεν μετακινείται μέχρι να αλλάξει mode λειτουργίας. Εδώ να πούμε ότι δεν σβήνει το τροφοδοτικό της διάταξης ούτε πηγαίνει σε κατάσταση αναμονής. Επίσης και οι σερβοκινητήρες εξακολουθούν να τροφοδοτούνται με ρεύμα και με τις γωνίες που πρέπει να έχουν οι δρομείς τους. Έτσι μπορούν να διατηρήσουν τη κεφαλή σε σταθερή θέση ακόμα και αν ασκηθεί κάποια δύναμη σε αυτήν. Για να γυρίσει το ρομπότ σε mode 1 πρέπει να πατηθεί το κόκκινο κουμπί, Bt1, που είναι κολλημένο στο Arduino Shield. Όσο είναι σε mode 1 είναι αναμμένο το κόκκινο LED που είναι κολλημένο στο Shield ενώ είναι σβησμένα όλα τα άλλα, εκτός βέβαια από αυτό της τροφοδοσίας. Καλό είναι πριν το σβήσιμο της τροφοδοσίας του ρομπότ, ο χρήστης να επιλέγει το mode 1 προκειμένου να μεταβούν σε οριζόντια θέση οι άνω βραχίονες και η κεφαλή στο σημείο (x,y,z)=(0,0,-33).

1.6.3 Το Mode 2

Στο mode 2 το ρομπότ δέχεται εντολές τοπικά από το χρήστη, τοπικός χειρισμός. Εδώ ο χρήστης έχει τη δυνατότητα μέσω ποτενσιόμετρων να εισάγει είτε τις καρτεσιανές συντεταγμένες είτε κατ' ευθείαν τις γωνίες των σερβοκινητήρων στο τοπικό τους σύστημα, όχι τις γωνίες θ_i. Επίσης έχει τη δυνατότητα να ανάβει και να σβήνει τον ηλεκτρομαγνήτη και το LED ασφαλείας. Για να ενεργοποιηθεί το mode 2 πρέπει να πατηθεί το κίτρινο κουμπί, Bt2, που βρίσκεται στο Shield. Όσο είναι σε mode 2 είναι αναμμένο το κίτρινο LED που βρίσκεται στο Shield. Επίσης ανάβει το LED 13 όταν ο χρήστης δίνει έγκυρες καρτεσιανές συντεταγμένες, ενώ σβήνει όταν αυτές παραπέμπουν σε σημείο εκτός της σφαίρας της εικόνας 1.3 ή κάτω του επιπέδου του πάγκου ή αυτού που έχει ορίσει ο χρήστης. Για να εισάγει ο χρήστης τις επιθυμητές καρτεσιανές συντεταγμένες υπάρχουν τρία ποτενσιόμετρα, ένα για κάθε συντεταγμένη για την x το Pot1, για την y το Pot2 και για τη z το Pot3.

Δεξιά τους υπάρχει και ένα τέταρτο, Pot4, για να επιλέγει το εύρος μέσα στο οποίο κινούνται οι συντεταγμένες x και y, αλλά όχι η z. Δηλαδή κάνει κάτι σαν περιορισμό του εύρους κίνησης της κεφαλής για τις x και y, ενώ για την z ισχύει η ίδια διαδικασία που περιγράφτηκε και παραπάνω με τον ορισμό του κατακόρυφου ορίου.

Ο διακόπτης που βρίσκεται πάνω αριστερά, Sw3, ελέγχει το LED ασφαλείας, αυτός που είναι κάτω αριστερά, Sw1, ελέγχει τον ηλεκτρομαγνήτη, ενώ με τον διπλανό του, Sw2, ο χρήστης επιλέγει τον τρόπο εισαγωγής συντεταγμένων, δηλαδή καρτεσιανές συντεταγμένες ή κατ' ευθείαν γωνίες σερβοκινητήρων, όχι θ_i. Στην περίπτωση των γωνιών, μόνο τα τρία ποτενσιόμετρα, Pot1, Pot2 και Pot3, λειτουργούν και μέσω αυτών εισάγονται οι γωνίες των τριών σερβοκινητήρων με εύρος από 0 έως 180 μοίρες. Σε αυτή τη περίπτωση το LED 13 είναι μόνιμα σβηστό. Επίσης, σε αυτή τη περίπτωση δεν γίνεται έλεγχος ορίων της σφαίρας της εικόνας 1.3 ούτε του κατακόρυφου ορίου, μόνο του εύρους των σερβοκινητήρων, 0 ως 180 μοίρες, για αυτό πρέπει να χρησιμοποιείται με προσοχή. Ο λόγος ύπαρξης της δυνατότητας εισαγωγής κατευθείαν των γωνιών είναι για να ελεγχθεί από το χρήστη η ορθή λειτουργία των σερβοκινητήρων και η σωστή τοποθέτησή τους και δεν προορίζεται για να χρησιμοποιείται κατά τη συνήθη λειτουργία του ρομπότ.

1.6.4 Η αλλαγή του κατακόρυφου ορίου

Όπως έχει αναφερθεί παραπάνω, υπάρχει ένας περιορισμός στην κατακόρυφη θέση της κεφαλής για να μην ακουμπήσει στον πάγκο εργασίας. Αυτό το όριο μπορεί να μεταβληθεί από το χρήστη. Κατά την εκκίνηση είτε κατά τη τροφοδοσία είτε λόγω επανεκκίνησης, reset, του Arduino Mega, ανάβουν όλα τα λαμπάκια για να διαπιστώσει ο χρήστης την ορθή λειτουργία τους. Όταν όλα έχουν ανάψει, περιμένει λίγο, μετά σβήνουν και τέλος ξεκινάει η κανονική λειτουργία του όπως έχει περιγραφεί παραπάνω.

Αν και οι δύο διακόπτες που είναι κάτω αριστερά, Sw1 και Sw2, είναι πατημένοι κατά την εκκίνηση ή πατηθούν πριν ανάψουν όλα τα LEDs, τότε το ρομπότ μπαίνει σε κατάσταση ρύθμισης του κατακόρυφου ορίου. Σε αυτή τη περίπτωση με τη χρήση του δεξιού ποτενσιόμετρου, Pot4, ο χρήστης μπορεί να ρυθμίσει αυτό το κατακόρυφο όριο. Η κεφαλή του ρομπότ δεν μετακινείται, ωστόσο κάθε φορά η τιμή στην οποία αντιστοιχεί η θέση του διακόπτη είναι διαθέσιμη στην έξοδο, με τον τρόπο που περιγράφεται στην επόμενη παράγραφο και αφορά γενικότερα τα μηνύματα εξόδου του ρομπότ. Κατά τη διαδικασία αυτή αναβοσβήνει το LED 13. Μόλις ο χρήστης αποφασίσει ποια τιμή προτιμά, ανοίγει τους διακόπτες. Τότε επιλέγεται η νέα τιμή η οποία αποθηκεύεται και στην EEPROM για μελλοντική χρήση.

Αν γίνει επανεκκίνηση, τότε επανέρχεται η προκαθορισμένη τιμή του ορίου δηλαδή τα -34.77cm και όχι αυτή του χρήστη. Αν όμως ο χρήστης επιθυμεί να φορτώσει την αποθηκευμένη τιμή, τότε κατά την εκκίνηση και πριν ανάψουν όλα τα LEDs πατάει τον πάνω αριστερά διακόπτη, Sw3. Τότε πάλι όπως και πριν αναβοσβήνει το LED 13 και στέλνεται στην έξοδο η αποθηκευμένη τιμή για ανάγνωση από το χρήστη. Εδώ δεν μπορεί να μεταβληθεί η αποθηκευμένη στην EEPROM τιμή, μόνο φορτώνεται και περιμένει μέχρι ο χρήστης να ανοίξει τον διακόπτη. Μόλις το κάνει το ρομπότ συνεχίζει κανονικά τη λειτουργία του χρησιμοποιώντας σαν οριακή τιμή αυτή που μόλις φόρτωσε.

1.7 Το μήνυμα εξόδου

Το Delta Robot κατά τη λειτουργία του στέλνει ένα μήνυμα εξόδου μέσω της σειριακής θύρας UART 0, δηλαδή μέσω του pin 1 (Transmit) με BAUD rate 9600. Σκοπός αυτού του εξερχόμενου μηνύματος είναι να μπορεί ο χρήστης μέσω κάποιας άλλης συσκευής να πληροφορείται τη θέση της κεφαλής καθώς και άλλες παραμέτρους. Η αποστολή γίνεται μέσω του pin 1 γιατί αυτό είναι συνδεδεμένο εσωτερικά με το κύκλωμα που επικοινωνεί με τον υπολογιστή μέσω θύρας USB. Έτσι ο χρήστης μπορεί να βλέπει την έξοδο σε κάποιο σειριακό τερματικό όπως είναι αυτό που είναι ενσωματωμένο στο Arduino IDE, δηλαδή στο περιβάλλον ανάπτυξης.

Εναλλακτικά, μπορεί να συνδεθεί με κάποια άλλη συσκευή που έχει κύκλωμα UART όπως είναι κάποιος άλλος Arduino. Για την συγκεκριμένη διπλωματική στο pin 1 του ρομπότ είχε συνδεθεί ένας Arduino UNO στον οποίο είχε τοποθετηθεί Shield που περιελάμβανε οθόνη LCD για την εκτύπωση του μηνύματος εξόδου του ρομπότ. Έτσι ο UNO λάμβανε στο pin 0 (Receive) το μήνυμα εξόδου και το τύπωνε στην οθόνη LCD. Αυτή είναι και η συνήθη συνδεσμολογία για επικοινωνία μέσω σειριακής UART ή USART, δηλαδή το pin Tx της μίας συσκευής συνδέεται με το pin Rx της άλλης.

Κατά το mode 0 το ρομπότ στέλνει μέσω της σειριακής θύρας 0, pin 1, τις συντεταγμένες x, y, z που έλαβε από το Raspberry Pi. Κατόπιν στέλνει χαρακτήρες κειμένου που πληροφορούν το χρήστη αν όλα είναι εντάξει ή αν έχει υπάρξει κάποιο πρόβλημα. Τέλος στέλνει έναν αριθμό που αφορά το αν οι συντεταγμένες παραπέμπουν σε έγκυρη θέση ή όχι. Αν αυτός ο αριθμός είναι 0, τότε όλα είναι καλά. Αν είναι αρνητικός, τότε σημαίνει πως κάποιοι περιορισμοί έχουν παραβιαστεί, οπότε οι συντεταγμένες παραπέμπουν σε μη έγκυρο σημείο και η κεφαλή δεν έχει μετακινηθεί. Αν έχει την τιμή 1 σημαίνει ότι υπήρξε πρόβλημα στις επικοινωνίες και δεν έλαβε όλο το μήνυμα, δηλαδή κάποιο τμήμα του δεν ήρθε, ενώ αν είναι 2, τότε σημαίνει ότι η συντεταγμένη z είναι μικρότερη από την ελάχιστη τιμή που πρέπει να έχει. Σε αυτή τη περίπτωση αυτή αντικαθίσταται από

την ελάχιστη και η κεφαλή μετακινείται στο επίπεδο αυτό. Και εδώ χρησιμοποιούνται οι χαρακτήρες «@» και «$» για να υποδείξουν την αρχή και το τέλος του μηνύματος κάνοντας έτσι πιο εύκολο τον κώδικα που πρέπει να τρέχει στην συσκευή εξόδου αν δεν είναι το τερματικό, πχ. στον Arduino UNO, προκειμένου να ξεχωρίσει ποιος αριθμός αναπαριστά ποιο μέγεθος.

Ομοίως και για το mode 2 όταν ο τρόπος εισόδου είναι σε καρτεσιανές συντεταγμένες. Αντίθετα όταν ο τρόπος εισόδου είναι σε γωνίες, τότε στέλνονται οι γωνίες αυτές χωρίς όμως άλλον αριθμό στο τέλος. Σε αυτήν την περίπτωση όμως μετά τον χαρακτήρα «@» ακολουθεί και ο «%» (ASCII 37) για να ενημερώσει τον Arduino UNO ότι οι αριθμοί που ακολουθούν είναι γωνίες και όχι καρτεσιανές συντεταγμένες έτσι ώστε να τυπωθεί στην οθόνη το κατάλληλο μήνυμα. Στο τέλος του μηνύματος υπάρχει επίσης ο «$».

Στο mode 1 στέλνεται στο pin 1 μήνυμα που λέει ότι είναι σε Pause Mode. Και εδώ χρησιμοποιούνται οι «@» και «$» με τη μόνη διαφορά ότι μετά το «@» ακολουθεί ο χαρακτήρας «#» (ASCII 35). Έτσι πληροφορείται ο Arduino UNO για την κατάσταση mode 1 και τυπώνει κατάλληλο μήνυμα στην οθόνη.

Τέλος αν είναι στην κατάσταση επιλογής κατακόρυφου ορίου κατά την εκκίνηση, όπως αναλύθηκε στην προηγούμενη παράγραφο, τότε για όσο χρονικό διάστημα είναι σε αυτή τη κατάσταση αποστέλλεται ένα μήνυμα με τον αριθμό που θα επιλεγεί ή έχει επιλεγεί για όριο. Εδώ μετά τον «@» ακολουθεί το «!» (ASCII 33) για να ενημερώσει κατάλληλα τον UNO. Στο τέλος του μηνύματος υπάρχει ο «$».

1.8 Η δομή του πηγαίου κώδικα που γράφτηκε για τον Arduino Mega

Σε προηγούμενες παραγράφους παρουσιάστηκε η λειτουργία του Delta Robot καθώς και ο χειρισμός του, όπως η χρήση των κουμπιών που υπάρχουν σε αυτό, τα LEDs, τα μηνύματα που μεταβιβάζονται μέσω των σειριακών κλπ. Εδώ παρουσιάζεται ο κώδικας που τρέχει στον Arduino Mega για να πραγματοποιηθούν όλα αυτά. Η γενική ιδέα έχει ως εξής: Αρχικά κάνει αρχικοποιήσεις παραμέτρων, όπως είναι η σειριακή επικοινωνία. Κατόπιν εξετάζει αν είναι πατημένοι οι διακόπτες Sw1 και Sw2 ταυτόχρονα ώστε να μπει σε κατάσταση ρύθμισης και αποθήκευσης του ελάχιστου κατακόρυφου ορίου όπως αυτό έχει αναλυθεί σε προηγούμενες παραγράφους. Ύστερα εξετάζει αν είναι πατημένος ο διακόπτης Sw3 για να φορτώσει το ελάχιστο κατακόρυφο όριο που είναι αποθηκευμένο στην EEPROM. Μετά θέτει το ρομπότ σε mode 0 για λήψη συντεταγμένων μέσω σειριακής και στέλνει στη σειριακή έναν χαρακτήρα «@» για να ζητήσει τη λήψη δεδομένων.

Αφού ολοκληρωθούν αυτά που βρίσκονται στη setup(), πηγαίνει στη loop() όπου βρίσκεται το κυρίως κομμάτι του κώδικα και επαναλαμβάνεται συνεχώς όσο

υπάρχει τροφοδοσία με ηλεκτρική ισχύ. Εκεί σε κάθε επανάληψη εξετάζει αρχικά αν είναι πατημένο κάποιο κουμπί από τα Bt0, Bt1 και Bt2. Αν πατηθεί κάποιο από αυτά αλλάζει το ρομπότ στο αντίστοιχο mode, πχ. mode 0 για το Bt0, mode 1 για το Bt1, mode 2 για το Bt2 και ανάβει ή σβήνει τα αντίστοιχα LEDs (πράσινο, κόκκινο κίτρινο) καθώς και το LED ασφαλείας. Ειδάλλως συνεχίζει να έχει το mode που είχε στον προηγούμενο κύκλο. Το κουμπί Bt0 έχει και μία δεύτερη χρήση: Αν βρίσκεται το ρομπότ ήδη σε mode 0 και πατηθεί το Bt0, τότε στέλνεται στη σειριακή 1, UART1 στο pin 19, ένας χαρακτήρας «@» για να ξαναγίνει ο συγχρονισμός με την εξωτερική συσκευή, όπως αναλύθηκε σε προηγούμενη παράγραφο. Όμως για να συμβεί αυτό πρέπει όταν πατηθεί το Bt0 να έχει περάσει τουλάχιστον ένα δευτερόλεπτο από τη στιγμή που είχε πατηθεί και αφεθεί το κουμπί αυτό σε προγενέστερο χρόνο. Αυτό συμβαίνει για να μην στέλνονται συνεχώς «@» λόγω παρατεταμένου πατήματος ή σπινθηρισμών. Έτσι κάθε φορά που πατιέται και αν είναι ήδη το ρομπότ σε mode 0, στέλνεται μόνο ένα «@» και πρέπει να αφεθεί για 1sec τουλάχιστον και να ξαναπατηθεί για να σταλεί κι άλλο.

Κατόπιν εισέρχεται στο αντίστοιχο block αναλόγως σε τι mode βρίσκεται. Δηλαδή, αν είναι σε mode 0 ελέγχει αν υπάρχουν δεδομένα στη σειριακή 1, UART1 μέσω του pin 18. Αν δεν υπάρχουν ανάβει το πορτοκαλί LED και εκτελεί τον επόμενο βρόγχο της loop. Αν υπάρχουν το σβήνει και ψάχνει για το χαρακτήρα «@». Μόλις το βρει ψάχνει για 4 ακέραιους αριθμούς που αντιστοιχούν στις συντεταγμένες και στην κατάσταση του ηλεκτρομαγνήτη με τον τρόπο που έχει αναλυθεί σε προηγούμενο κεφάλαιο και αφορά την επικοινωνία. Αν η συντεταγμένη z είναι μικρότερη από το όριο, τότε την αντικαθιστά με το όριο αυτό. Τέλος ψάχνει το χαρακτήρα «$». Αν δεν τον βρει θεωρεί ότι η μετάδοση του μηνύματος ήταν ελλιπής ή προβληματική και ότι τα δεδομένα που μόλις διάβασε δεν είναι έγκυρα έτσι δεν μετακινεί την κεφαλή. Αν ο «$» έχει βρεθεί, τότε θεωρεί ότι το μήνυμα είναι ολοκληρωμένο και σωστό και θέτει τον ηλεκτρομαγνήτη εντός ή εκτός ανάλογα. Κατόπιν ελέγχει αν οι συντεταγμένες που έλαβε είναι μέσα στη σφαίρα που μπορεί να κινηθεί, εικόνα 1.3, και υπολογίζει τις γωνίες θ_i. Κατόπιν τις στρογγυλοποιεί στον πλησιέστερο ακέραιο και αυτόν τον αφαιρεί από το offset, που είναι 110 μοίρες περίπου, καθότι για θ=0 οι δρομείς των σερβοκινητήρων βρίσκονται στις 110 μοίρες περίπου ως προς το σύστημα συντεταγμένων του σερβοκινητήρα αλλά με αντίθετη φορά. Αν ο τελικός αριθμός προκύψει να είναι μέσα στο εύρος του σερβοκινητήρα, δηλαδή μεταξύ 0 και 180 μοίρες, και εφ’ όσον είναι διαφορετικός από τη θέση που βρίσκεται ήδη ο δρομέας, τότε μετακινούνται οι δρομείς στην κατάλληλη θέση και ανάβει το SMD LED 13, ειδάλλως η κεφαλή δεν μετακινείται και το LED 13 σβήνει. Αφού ολοκληρωθούν όλα αυτά και εφ’ όσον δεν υπάρχουν άλλα δεδομένα στη σειριακή 1, στέλνει ένα «@» στη σειριακή 1, UART1 μέσω του pin 19, για να ζητήσει νέα δεδομένα.

Αν βρίσκεται σε mode 1 δεν κάνει κάτι, απλά εκτελεί διαρκείς επαναλήψεις της loop μέχρι να αλλάξει mode. Αυτό που συμβαίνει όταν πατηθεί το κουμπί Bt1 είναι να πηγαίνει στη θέση (x,y,z)=(0,0,-33) όπου οι γωνίες $\theta_1=\theta_2=\theta_3=0$ και σβήνει τον ηλεκτρομαγνήτη και όλα τα LEDs, εκτός από το κόκκινο, και παραμένει εκεί για όσο το mode λειτουργίας είναι το 1.

Αν βρίσκεται σε mode 2, τότε αν και όσο είναι κλειστός, πατημένος, ο διακόπτης Sw3, ενεργοποιείται το LED ασφαλείας, ενώ αν και για όσο διάστημα είναι κλειστός, πατημένος, ο Sw1, ενεργοποιείται ο ηλεκτρομαγνήτης. Τέλος αν ο Sw2 είναι ανοιχτός, τότε διαβάζει το ποτενσιόμετρο Pot3 και την τιμή αυτή την αντιστοιχεί στη συντεταγμένη z καθώς και τα ποτενσιόμετρα Pot1 και Pot2 και τα αντιστοιχεί σε συντεταγμένες x και y αντίστοιχα σα συνάρτηση του πολλαπλασιαστικού παράγοντα που διαβάζει από το Pot4. Για να γίνουν αυτά κάνει παράλληλα και μετατροπή του εύρους καθότι τα ποτενσιόμετρα εισάγουν τιμές από 0 έως 1023, ενώ οι συντεταγμένες κυμαίνονται από -22 έως +22 cm για τις x και y και από -29 έως -40 cm για τη z. Κατόπιν γίνεται έλεγχος αν το z είναι μικρότερο από -34.77 και αν ναι αντικαθίσταται με την τιμή -34.77. Ύστερα γίνεται έλεγχος αν οι συντεταγμένες παραπέμπουν μέσα στο χώρο της εικόνας 1.3 και κατόπιν με παρόμοιο τρόπο με την περίπτωση του mode 0 γίνεται η μετατροπή των καρτεσιανών συντεταγμένων σε γωνίες και μετακινείται η κεφαλή αν δεν είναι ήδη εκεί από προγενέστερη εκτέλεση του βρόγχου. Ο έλεγχος για το αν βρίσκεται εκεί ήδη η κεφαλή ή όχι γίνεται γιατί ο χρήστης μπορεί να μην μετακινεί συνέχεια τα ποτενσιόμετρα και για πολλές επαναλήψεις του βρόγχου αυτά να παραπέμπουν στο ίδιο σημείο.

Αν ο Sw2 είναι κλειστός τότε σβήνει το LED 13 και διαβάζει από τα ποτενσιόμετρα Pot1, Pot2 και Pot3 τις γωνίες των σερβοκινητήρων, τα αντιστοιχεί σε εύρος από 0 έως 180 και στέλνει τον κατάλληλο παλμό στους σερβοκινητήρες. Εδώ να πούμε ότι δεν γίνεται έλεγχος ορίων, παρά μόνο έλεγχος αν οι τελικές τιμές των γωνιών είναι μέσα στο εύρος των σερβοκινητήρων δηλαδή μεταξύ 0 και 180 μοιρών.

Στο τέλος κάθε επανάληψης του βρόγχου loop στέλνει στην σειριακή 0, UART0, μέσω του pin 1 το μήνυμα εξόδου που αναλύθηκε σε προηγούμενο κεφάλαιο. Αυτό το μήνυμα αποστέλλεται σε κάθε περίπτωση, αλλά είναι διαφορετικό για κάθε mode όπως έχει ήδη αναλυθεί. Για να μην καθυστερεί η εκτέλεση του κώδικα το μήνυμα εξόδου δεν αποστέλλεται διαρκώς, παρά μόνο όταν υπάρχει αρκετός χώρος στην ενδιάμεση μνήμη εξόδου. Για αυτό και ο κώδικας πρώτα κάνει έλεγχο κενών θέσεων στη buffer και αν είναι γεμάτη δεν θα αποσταλούν δεδομένα κατά τον τρέχοντα βρόγχο, αλλά σε επόμενο όταν θα υπάρχουν αρκετές κενές θέσεις. Έτσι κάποια δεδομένα θα χαθούν αλλά δεν υπάρχει ιδιαίτερο πρόβλημα καθότι πρόκειται για δεδομένα εξόδου που προορίζονται για πληροφόρηση του χρήστη και

η όλη διαδικασία ανανέωσης των δεδομένων είναι τόσο γρήγορη που ακόμα και να μην αποσταλεί κάποιο μήνυμα δεν θα δημιουργηθεί ιδιαίτερο πρόβλημα στο χρήστη.

Εδώ να πούμε ότι οι διακόπτες και τα κουμπιά όταν είναι πατημένα, κλειστές οι επαφές, ισοδυναμούν με λογικό LOW γιατί τροφοδοτούν την είσοδο με 0V επειδή είναι γειωμένα. Όταν είναι ανοικτές οι επαφές, τότε η είσοδος τροφοδοτείται με 5V, λογικό HIGH, γιατί έχει ενεργοποιηθεί η εσωτερική Pull Up αντίσταση. Έτσι ακόμα και αν έχει αφαιρεθεί το Arduino Shield πριν την εκκίνηση, δεν θα προκύψει κάποιο πρόβλημα στη λειτουργία του ρομπότ γιατί θα ισοδυναμεί με την κατάσταση που κανένα κουμπί ή διακόπτης δεν έχει πατηθεί. Έτσι θα λειτουργεί κανονικά στο default mode που είναι το 0, οπότε θα δέχεται εντολές απομακρυσμένα, αλλά δεν θα μπορεί να το χειριστεί ο χρήστης τοπικά. Αυτός είναι και ο λόγος που αποφασίστηκε να χρησιμοποιηθεί το ενσωματωμένο LED 13 για να πληροφορήσει την εγκυρότητα ή μη των συντεταγμένων ή του μηνύματος. Και αυτό γιατί στον Arduino Mega υπάρχει επίσης ένα LED 13 με αποτέλεσμα, ακόμα και αν αφαιρεθεί το Arduino Shield, ο χρήστης να μπορεί να ξέρει για την εγκυρότητα ή μη του μηνύματος ή του σημείου που λαμβάνεται από την εξωτερική συσκευή. Αυτή η πληροφορία θεωρήθηκε πιο σημαντική από την πληροφορία που μπορεί να αντλήσει από τα υπόλοιπα LEDs που είναι κολλημένα στο Shield και απλά πληροφορούν για το mode λειτουργίας πράγμα που είναι ήδη γνωστό στο χρήστη όταν δεν υπάρχει το Shield γιατί δεν έχει διαθέσιμα τα κουμπιά που μπορούν να το αλλάξουν.

1.9 Οι πίνακες βαθμονόμησης των σερβοκινητήρων

Στην αρχή του κώδικα υπάρχουν τρεις πίνακες με 181 στοιχεία ο καθένας. Αυτοί είναι πίνακες βαθμονόμησης των σερβοκινητήρων. Για να αναλυθεί καλύτερα η χρήση τους πρέπει πρώτα να γίνει μία σύντομη περιγραφή των σερβοκινητήρων που χρησιμοποιήθηκαν αντλώντας δεδομένα από τη διπλωματική του Γεωργίου Βαγενά ο οποίος και τους βαθμονόμησε και παρουσιάζονται παρακάτω.

Οι σερβοκινητήρες που χρησιμοποιήθηκαν μπορούν να τοποθετήσουν τον δρομέα τους σε όποια θέση τους ζητηθεί από 0 μέχρι 180 μοίρες. Αυτό το πετυχαίνουν με τη βοήθεια κατάλληλων ηλεκτρονικών κυκλωμάτων που υπάρχουν εσωτερικά καθώς και με ένα ποτενσιόμετρο μηχανικά συνδεδεμένο με το δρομέα. Έτσι ξέρουν μέσω ηλεκτρικής ανάδρασης από το ποτενσιόμετρο κάθε στιγμή τη θέση του δρομέα και με βάση την επιθυμητή θέση τροφοδοτείται με ρεύμα ένας κινητήρας συνεχούς ρεύματος. Για υποβιβασμό της κίνησης υπάρχουν κατάλληλα γρανάζια πετυχαίνοντας έτσι μεγάλη ακρίβεια στην τοποθέτηση του δρομέα, μεγάλες τιμές ροπής αλλά και ικανοποιητική ταχύτητα λόγω του πολύστροφου DC κινητήρα.

Για να λάβουν την επιθυμητή γωνία από τον Arduino Mega, αυτός στέλνει μία σειρά ηλεκτρικών παλμών που μεταφράζονται από τον ελεγκτή του σερβοκινητήρα σε θέση του δρομέα. Οι παλμοί αυτοί έχουν συχνότητα 50Hz και αποκωδικοποιούνται σε μοίρες βάση της χρονική διάρκειας που είναι ο παλμός σε λογική κατάσταση 1 (HIGH), 5V. Δηλαδή για 0 μοίρες ο παλμός έχει τιμή HIGH για διάστημα γύρω στα 600μsec. Οι 180 μοίρες αντιστοιχούν σε εύρος 2300μs ή 2.3ms κοκ. Οτιδήποτε μικρότερο εκλαμβάνεται ως 0 μοίρες, ενώ οτιδήποτε μεγαλύτερο ως 180 μοίρες, δηλαδή αντιστοιχούν στις ακραίες θέσεις που μπορεί να πάρει ο δρομέας. Κάθε φορά δεν στέλνεται ένας παλμός αλλά στέλνονται συνέχεια παλμοί για όσο τρέχει ο κώδικας. Αυτό που αλλάζει είναι το εύρος της άνω παρειάς του κάθε παλμού ανάλογα με την επιθυμητή θέση.

Η βιβλιοθήκη Servo που χρησιμοποιήθηκε μετατρέπει τις μοίρες σε χρονική διάρκεια άνω παρειάς παλμού για ευκολία του χρήστη. Όμως οι σερβοκινητήρες δεν παρουσίασαν αμιγώς γραμμική συμπεριφορά όσον αφορά την αντιστοιχία χρονικής διάρκειας παλμών με μοίρες, ενώ δεν είχαν και οι τρεις παρόμοια συμπεριφορά. Παρά το ότι οι διαφορές μεταξύ τους καθώς και η μη γραμμικότητα ήταν μικρές, αποφασίστηκε για μεγαλύτερη ακρίβεια στη λειτουργία του ρομπότ να γίνει βαθμονόμηση όπως περιγράφεται αναλυτικά στην διπλωματική του Γεωργίου Βαγενά.

Έτσι δημιουργήθηκε ένας πίνακας 181 στοιχείων για κάθε έναν σερβοκινητήρα όπου αντιστοιχίζεται η κάθε τιμή μοιρών σε μία διάρκεια παλμού σε μsec. Συνεπώς το πρώτο στοιχείο του, servo1[0], αντιστοιχεί στη χρονική διάρκεια της κατάστασης HIGH του παλμού που πρέπει να μεταβιβαστεί στον σερβοκινητήρα 1 προκειμένου ο δρομέας του να πάει στη θέση με γωνία 0 μοίρες. Το servo1[180], αφορά τις 180 μοίρες κλπ. Οπότε η θέση σε μοίρες που επιθυμούμε να βρεθεί ο δρομέας δεν μεταβιβάζεται στη συνάρτηση servo.write…, αλλά πρόκειται για θέση στοιχείου πίνακα, η τιμή του οποίου είναι η χρονική διάρκεια σε μsec που θα πρέπει να μεταβιβαστεί στη συνάρτηση. Για αυτό το λόγο στον κώδικα που γράφτηκε στα πλαίσια της παρούσας διπλωματικής γίνεται έλεγχος αν η επιθυμητή γωνία ανήκει στο εύρος από 0 έως 180 γιατί σε αντίθετη περίπτωση θα προσπελαστεί θέση μνήμης που δεν αντιστοιχεί σε στοιχείο του πίνακα με άγνωστες συνέπειες.

Ένα άλλο σημείο που πρέπει να αναφερθεί είναι ότι επειδή οι συγκεκριμένοι πίνακες είναι μεγάλοι, για οικονομία μνήμης RAM αυτοί οι πίνακες δεν αντιγράφονται ποτέ στην μνήμη RAM, όπως συμβαίνει γενικά με τις μεταβλητές, αλλά προσπελαύνονται κατευθείαν από τη μνήμη flash. Αυτό μπορεί να γίνει γιατί τα δεδομένα τους είναι σταθερά και δεν μεταβάλλονται κατά την εκτέλεση. Αν μεταβάλλονταν, αυτό δεν θα μπορούσε να συμβεί γιατί κατά την εκτέλεση του κώδικα επιτρέπεται μόνο ανάγνωση από την flash, οπότε θα έπρεπε να αντιγραφούν στην RAM κατά την εκκίνηση του Arduino. Από την άλλη, χρειάζονται 3

κύκλοι του επεξεργαστή για να προσπελαστεί η flash ενώ 2 κύκλοι για τη RAM. Όμως αυτή η αύξηση στον χρόνο προσπέλασης είναι αμελητέα και δεν δημιουργεί προβλήματα στο τελικό χρόνο εκτέλεσης του κώδικα κυρίως αν συνυπολογιστεί ότι αυτοί οι πίνακες δεν προσπελαύνονται συνεχώς παρά μόνο όταν υπάρχουν νέες συντεταγμένες πράγμα που στην πράξη γίνεται ανά μεγάλα χρονικά διαστήματα και όχι σε κάθε επανάληψη της loop. Από την άλλη, το κέρδος σε οικονομία στη RAM είναι σημαντικό και αφήνει αρκετό ελεύθερο χώρο στη μνήμη.

Για να υλοποιηθεί αυτό, αυτοί οι πίνακες έχουν δηλωθεί ως PROGMEM και η προσπέλασή τους δεν γίνεται με τον κλασσικό τρόπο, πχ. servo1[4], αλλά με τη χρήση της συνάρτησης pgm_read_word_near(). Δηλαδή αντί για την έκφραση a=servo1[4] έχουμε την εντολή a=pgm_read_word_near(servo1+4).

1.10 Ο πηγαίος κώδικας που εκτελείται στον Arduino Mega

Παρακάτω αναλύεται ο πηγαίος κώδικας που γράφτηκε στα πλαίσια της παρούσας διπλωματικής για τη λειτουργία του Delta Robot όπως αυτή έχει περιγραφτεί. Στα επόμενα παρατίθεται και επεξηγείται ο κώδικας τμηματικά σε blocks για καλύτερο σχολιασμό και κατανόηση, ενώ στο τέλος του κεφαλαίου υπάρχει όλος μαζί συγκεντρωμένος για καλύτερη ανάγνωση. Σε κάθε υποπαράγραφο, πριν και μετά τον κώδικα παρατίθενται σχόλια πάνω σε αυτόν, ενώ υπάρχουν σχόλια και μέσα σε αυτόν. Κάθε υποπαράγραφος ξεκινά από καινούρια σελίδα για να είναι ευκολότερη η ανεύρεσή του από τον αναγνώστη. Κάποια τμήματα κώδικά προϋπήρχαν από την προηγούμενη διπλωματική του Γεωργίου Βαγενά.

1.10.1 Οι πίνακες βαθμονόμησης και οι σταθερές

Αρχικά ο κώδικας περιέχει τις οδηγίες προς τον προεπεξεργαστή και τους πίνακες βαθμονόμησης των τριών σερβοκινητήρων. Επίσης υπάρχουν κάποιες σταθερές που αφορούν τα γεωμετρικά χαρακτηριστικά. Για λόγους συντομίας παρακάτω παρατίθενται κάποια από τα στοιχεία των πινάκων βαθμονόμησης. Στο τέλος του κεφαλαίου υπάρχει όλος ο κώδικας μαζί όπου εκεί οι πίνακες αυτοί είναι οι πλήρεις. Το παρακάτω τμήμα του κώδικα εκτός από την χρήση της βιβλιοθήκης EEPROM (οδηγία #include <EEPROM.h>) προϋπήρχε και έχει ληφθεί από τη διπλωματική του Γεωργίου Βαγενά:

```
#include <EEPROM.h>
#include <Servo.h> // Servo library

Servo servo_1; // create servo object to control a servo
Servo servo_2; // a maximum of eight servo objects can be created
Servo servo_3;

const int servo1[]PROGMEM =
 {641,651,659,668,676,680,690,779,788,796,808,815,825,833,843,854,…
 ……, // 181 στοιχεία συνολικά
 ……2288,2296,2305};

const int servo2[]PROGMEM =
 {587,594,600,608,612,620,627,633,641,651,659,662,672,…
                    // 181 στοιχεία συνολικά
 ……..,2296};

const int servo3[]PROGMEM =
 {587,596,603,609,616,626,632,640,648,657,665,671,679,688,696,701,708,716,722,729,…
                    // 181 στοιχεία συνολικά
 ……..2208};


// robot geometry
const float e = 8.0;     // end effector
const float f = 23.32;     // base
const float re = 35.0;
const float rf = 6.0;

// trigonometric constants
const float sqrt3 = sqrt(3.0);
const float pi = 3.141592653;    // PI
const float sin120 = sqrt3/2.0;
```

```
const float cos120 = -0.5;
const float tan60 = sqrt3;
const float sin30 = 0.5;
const float tan30 = 1.0/sqrt3;

const int offset_1 =111; // rf are horizontal
const int offset_2 =110;
const int offset_3 =116;

const int em = 12; // Defines the pin which controls the electromagnet
const int led = 8; // Defines the pin which controls the LED of the Delta Robot
```

Εδώ να πούμε ότι τα offset_1 έως 3 είναι η διαφορά στην γωνία τοποθέτησης των σερβοκινητήρων ως προς το οριζόντιο επίπεδο, όπως έχει αναφερθεί και σε προηγούμενες παραγράφους. Δηλαδή για να είναι ο βραχίονας 1 σε γωνία θ=0 πρέπει ο σερβοκινητήρας 1 να έχει τον δρομέα του στις 111 μοίρες ως προς το δικό του σύστημα συντεταγμένων. Άρα για να προκύψει το μήκος παλμού πρέπει να προσπελαστεί από τον κώδικα το στοιχείο 111 του πίνακα servo1, δηλαδή το servo1[111], κοκ. Επίσης να επαναλάβουμε ότι η φορά μέτρησης των γωνιών των σερβοκινητήρων είναι αντίθετη από την φορά μέτρησης των γωνιών θ_i. Άρα οι θ_i πρέπει να αφαιρεθούν από το offset για να προκύψει η γωνία του δρομέα. Πχ.:

$$\text{για } \theta_1 = -10°, \text{έχω } \text{angle\_1} = \text{offset\_1} - \theta_1 \Leftrightarrow \text{angle\_1} = 111 + 10 = 121°$$

1.10.2 Οι υπόλοιπες σταθερές

Εδώ δηλώνονται κάποιες σταθερές που πληροφορούν ποια pins (είσοδοι ή έξοδοι) του Arduino Mega κάνουν ποια δουλειά. Γενικά αυτό δεν είναι απαραίτητο, αλλά γίνεται για ευκολία στον προγραμματισμό και στην αναγνωσιμότητα του κώδικα. Πχ. στο pin 52 θα συνδέσουμε το μαύρο κουμπί, στο 6 το κόκκινο, μεταβλητή pinForStop, στο 48 το κίτρινο. Δηλαδή ο κάθε αριθμός έχει αντιστοιχιστεί σε μία σταθερά, της οποίας το όνομα δίνει πληροφορία (λέει κάτι) στον αναγνώστη. Οπότε κάθε φορά μέσα στον κώδικα δεν χρειάζεται να ξέρουμε σε ποιο pin είναι πιο κουμπί ή LED κλπ, αρκεί να βάλουμε το όνομα της κατάλληλης σταθεράς, και έτσι θα προκύψει ο σωστός αριθμός. Έτσι έχουμε τις παρακάτω δηλώσεις:

```
const int pinForRemoteControl=52; // Low to control via external device
const int pinForStop=6;  // Low to stop
const int pinForLocalControl=48; // Low to control delta Robot via potensiometers
const int pinToControlMagnet=5; // Input Pin that defines magnet state when controlMode is 2.
                                //Low for On, High for Off.
const int pinToControlLED=7; // Input Pin that defines LED state when controlMode is 2.
                                // Low for On, High for Off.
const int pinToToggleInputMode=4;  // When controlMode is 2 defines whether input
                                values are angles or coordinates. Low for Angle, High for Coordinates
const int redLed=3;   // Pin for red led. Led is On at stop mode
const int yellowLed=2;  // Pin for yellow led. Led is On at local control mode
const int orangeLed=26; // Pin for orange led. Led is On when waiting to receive
                                //data at remote control mode
const int greenLed=40;   // Pin for green led. Led is On at remote control mode
const int led13=13;   // Pin for Arduino built in LED 13. Led 13 is On when the position is
                                // valid and off otherwise or in Pause mode.
const int pinForXorAngle1Potensiometer=1; // Pin connected to the potensiometer that
                                // defines X coordinate or Angle1
const int pinForYorAngle2Potensiometer=5; // Pin connected to the potensiometer that
                                // defines Y coordinate or Angle2
const int pinForZorAngle3Potensiometer=8; // Pin connected to the potensiometer that
                                // defines Z coordinate or Angle3
const int pinForZoomPotensiometer=15; // Pin connected to the potensiometer that
                                // defines Zoom factor for the other three
```

Αν για κάποιο λόγο ο χρήστης αλλάξει τη συνδεσμολογία και συνδέσει κάποιο στοιχείο (LED, ποτενσιόμετρο κλπ) σε κάποιο άλλο pin, αρκεί να αλλάξει την τιμή της αντίστοιχης μεταβλητής-σταθεράς και να δώσει τον αριθμό του νέου pin. Έτσι δεν χρειάζεται να ψάχνει μέσα στον κώδικα όλα τα σημεία που αναφέρονται στο παλιό και στο καινούριο pin και να τα αλλάξει. Πχ. αν το κόκκινο LED που είναι στο pin 3, συνδεθεί στο 4, τότε αρκεί η εντολή «const int redLed=3;» να γίνει «const int redLed=4;».

1.10.3 Οι υπόλοιπες μεταβλητές

Παρακάτω ακολουθούν και οι υπόλοιπες μεταβλητές που έχουν δηλωθεί και αποθηκεύονται διάφορες παράμετροι του κώδικα:

```
const float tableLevel=-34.77;  // The z coordinate of the table
float verticalLimit=tableLevel; // The lower vertical limit that delta robot head is allowable to reach
int verticalFlag=0; // If the given z coordinate is lower than vertical Limit then vertical flag=1 else =0.

const unsigned long intervalBetweenButtonPress=1000; // For button debouncing
unsigned long currentTime=0, previousTime=0;

float t1;
float t2;
float t3;
int angle_1=0,angle_2=0,angle_3=0;          //  Desired servo angles
int oldAngle1=0, oldAngle2=0, oldAngle3=0;// Current servo angles

float x=0.0, y=0.0, z=-33.0;          //  x,y,z coordinates
float multipl=1.0;
int electromagnet=0, ledState=0, magStatus=0, ledStatus=0;
            // ...State= the desired condition (On or Off), ...Status=the current condition.
int retStat=0; // Return status of angles calculation. 0 If everything is
                        //… OK, -1, -2, -3 if one two or three angles are bad.
int controlMode=0; // Defines who controls the servo. 0 for external serial device, 1 to
                        // stop (Pause mode) and 2 for potensiometers
int coord=HIGH;
int validPosition=0; // Equals 0 if the given x,y,z coordinates correspond to a valid
                        // position, negative if not and 1 when bad data is received.
                        // If it is 2 then the given z coordinate is less than vertical limit.
```

Η διαφορά των μεταβλητών ledState από ledStatus είναι ότι η ledStatus αποθηκεύει την επιθυμητή κατάσταση λειτουργίας του LED ασφαλείας, on ή off. Αντίθετα η ledState αποθηκεύει την τρέχουσα κατάσταση. Η χρήση τους θα γίνει κατανοητή μέσα στον κώδικα που παρατίθεται σε επόμενες υποπαραγράφους.

1.10.4 Οι συναρτήσεις επίλυσης του inverse kinematics problem

Οι συναρτήσεις που ακολουθούν αφορούν στον υπολογισμό του αντιστρόφου κινηματικού προβλήματος (inverse kinematics). Δηλαδή την εύρεση των γωνιών θ_i από τις συντεταγμένες x, y, z. Αυτές υπήρχαν στη διπλωματική του Γεωργίου Βαγενά όπου υπήρχε αναφορά στο διαδίκτυο (forums.trossenrobotics.com) σαν πηγή προέλευσής τους. Η πλήρης διαδρομή παρατίθεται στην βιβλιογραφία στο τέλος. Παρακάτω παρατίθενται αυτές οι δύο συναρτήσεις:

```
int delta_calcAngleYZ(float x0, float y0, float z0, float &theta)
{
  float y1 = -0.5 * 0.57735 * f; // f/2 * tg 30
  //float y1 = yy1;
  y0 -= 0.5 * 0.57735    * e;    // shift center to edge
  // z = a + b*y
  float a = (x0*x0 + y0*y0 + z0*z0 +rf*rf - re*re - y1*y1)/(2*z0);
  float b = (y1-y0)/z0;
  // discriminant
  float d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
  if (d < 0) return -1; // non-existing point
  float yj = (y1 - a*b - sqrt(d))/(b*b + 1); // choosing outer point
  float zj = a + b*yj;
  theta = 180.0*atan(-zj/(y1 - yj))/pi + ((yj>y1)?180.0:0.0);
  if ((theta < -180) || (theta > 180))
     return -1;
  return 0;
}

// inverse kinematics: (x0, y0, z0) -> (theta1, theta2, theta3)
// returned status: 0=OK, negative=non-existing position, the negative number is the
//number of wrong angles
int delta_calcInverse(float x0, float y0, float z0, float &theta1, float &theta2, float &theta3)
{
  theta1 = theta2 = theta3 = 0;
  int stat1 = delta_calcAngleYZ(x0, y0, z0, theta1);
  int stat2 = delta_calcAngleYZ(x0*cos120 + y0*sin120, y0*cos120-x0*sin120, z0, theta2);
                          // rotate coords to +120 deg
  int stat3 = delta_calcAngleYZ(x0*cos120 - y0*sin120, y0*cos120+x0*sin120, z0, theta3);
                          // rotate coords to -120 deg
  return stat1+stat2+stat3;
}
```

Η delta_calcAngleYZ(…) υπολογίζει τη γωνία θ συναρτήσει των συντεταγμένων x,y,z και η τιμή αποθηκεύεται στην μεταβλητή του ορίσματος κλήσης της συνάρτησης λόγω της κλήσης της με αναφορά. Επιστρέφει 0 αν όλα είναι καλά ή -1 αν το σημείο είναι εκτός της σφαίρας της εικόνας 1.3.

Η int delta_calcInverse(…) δέχεται σαν ορίσματα τις συντεταγμένες x,y,z, και καλεί την delta_calcAngleYZ τρεις φορές μία για κάθε σερβοκινητήρα, αφού περιστρέψει το κατακόρυφο επίπεδο 120 μοίρες για κάθε έναν σερβοκινητήρα, γιατί τόσες μοίρες είναι η γωνία που σχηματίζουν οι σερβοκινητήρες μεταξύ τους. Οι τελικές τιμές των γωνιών θ_i αποθηκεύονται στις μεταβλητές των ορισμάτων όπως και πριν λόγω της αναφορικής κλήσης. Επιστρέφει 0 αν όλα είναι καλά ή έναν αρνητικό αριθμό αναλόγως το πόσες γωνίες δεν υπάρχουν βάση της κλήσης της delta_calcAngleYZ. Περισσότερες πληροφορίες για αυτές τις δύο συναρτήσεις μπορεί κανείς να αντλήσει από τη διπλωματική του Γεωργίου Βαγενά.

1.10.5 Η συνάρτηση setup()

Η συνάρτηση setup() κάνει αρχικοποίηση των σειριακών θυρών και δηλώνει ποια pins είναι είσοδοι και ποια έξοδοι, και που υπάρχει σερβοκινητήρας, ενώ στις εισόδους ενεργοποιεί τις Pull Up αντιστάσεις. Αφού γίνουν αυτά πηγαίνει τους βραχίονες στη θέση (x,y,z)=(0,0,-33), όπου $\theta_1=\theta_2=\theta_3=0$. Ύστερα ανάβει όλα τα LEDs και ελέγχει αν ο χρήστης έχει πατήσει τους διακόπτες Sw1 και Sw2 για να αλλάξει το οριζόντιο όριο και να το αποθηκεύσει στην θέση 3 της EEPROM. Μετά ελέγχει αν είναι πατημένος ο Sw3 για να φορτώσει από την θέση 3 της EEPROM το αποθηκευμένο όριο. Εδώ να πούμε ότι η αρίθμηση των θέσεων-διευθύνσεων της EEPROM ξεκινάει από το 0 και πάει μέχρι το 1023. Άρα το EEPROM[3] είναι το τέταρτο στοιχείο. Μετά σβήνει όλα τα LEDs και τέλος πηγαίνει σε mode 0, στέλνει ένα «@» στη UART1 και ανάβει ή σβήνει τα αντίστοιχα LEDs. Παρακάτω παρατίθεται ο πηγαίος κώδικας της setup():

```
void setup()
 {
  Serial.begin(9600);   // To output data to Serial monitor or any other connected device.
  Serial1.begin(9600);  // Communication with external device for remote control.
                                  //Used for data input, such as coordinates.
  pinMode(led13, OUTPUT);
  pinMode(em, OUTPUT);
  pinMode(led, OUTPUT);
  digitalWrite(led13,HIGH);
  digitalWrite(em,LOW);
  digitalWrite(led,LOW);

  servo_1.attach(9,500,2500); // attaches the servo on pin 9 to the servo object
  servo_2.attach(10,400,2500); // attaches the servo on pin 10 to the servo object
  servo_3.attach(11,400,2500); // attaches the servo on pin 11 to the servo object

   //all servos horizontal (x,y,z)=(0,0,-31)
  servo_1.writeMicroseconds(pgm_read_word_near(servo1+offset_1));
  oldAngle_1=offset_1;
  servo_2.writeMicroseconds(pgm_read_word_near(servo2+offset_2));
  oldAngle_2=offset_2;
  servo_3.writeMicroseconds(pgm_read_word_near(servo3+offset_3));
  oldAngle_3=offset_3;

  delay(100);
  pinMode(redLed,OUTPUT);
  digitalWrite(redLed,HIGH);
  delay (100);
  pinMode(yellowLed,OUTPUT);
```

```
  digitalWrite(yellowLed,HIGH);
  delay (100);
  pinMode(greenLed,OUTPUT);
  digitalWrite(greenLed,HIGH);
  delay(100);
  pinMode(orangeLed,OUTPUT);
  digitalWrite(orangeLed,HIGH);
  delay(100);
  pinMode(pinForRemoteControl,INPUT_PULLUP);
  pinMode(pinForStop,INPUT_PULLUP);
  pinMode(pinForLocalControl,INPUT_PULLUP);
  pinMode(pinToControlMagnet,INPUT_PULLUP);
  pinMode(pinToControlLED,INPUT_PULLUP);
  pinMode(pinToToggleInputMode,INPUT_PULLUP);

  delay(500);

   // If both switches are pressed during boot, then store desired vertical limit offset to EEPROM
  if (digitalRead(pinToControlMagnet)==LOW && digitalRead(pinToToggleInputMode)==LOW)
  {
        Τα περιεχόμενα αυτού του block βρίσκονται στην παράγραφο 1.10.5.1
 }

  // If switch is pressed during boot then load vertical limit offset from EEPROM
  if (digitalRead(pinToControlLED)==LOW)
  {
        Τα περιεχόμενα αυτού του block βρίσκονται στην παράγραφο 1.10.5.2
 }

  digitalWrite(led13,HIGH);  // Led 13 is on because servos are in valid position (0,0,-31)

  digitalWrite(redLed,LOW);
  delay(100);
  digitalWrite(yellowLed,LOW);
  delay(100);
  digitalWrite(greenLed,LOW);
  delay(100);
  digitalWrite(orangeLed,LOW);
  delay(100);
  digitalWrite(led,HIGH); // Turn on Delta Robot's LED
  ledStatus=1;
  digitalWrite(greenLed,HIGH);  // Default mode is 0 (Remotely controlled)
  Serial1.write('@'); // Informs external device that delta robot is ready to receive data.
}
```

1.10.5.1 Η πρώτη if της setup()

Εδώ διαβάζει συνεχώς από το ποτενσιόμετρο Pot4 για όσο διάστημα είναι πατημένοι οι Sw1 και Sw2 και τυπώνει στην UART0 την τιμή που διαβάζει από τα ποτενσιόμετρα. Μόλις αφεθούν η τιμή αποθηκεύεται στην EEPROM και το οριζόντιο όριο γίνεται όσο είναι το default, -34.77, αν προστεθεί η τιμή που διαβάστηκε, 0 έως 255, διαιρεμένη δια του 200. Δηλαδή προστίθεται ένας αριθμός μεταξύ του 0 και του 1.27 περίπου. Άρα το όριο μπορεί να γίνει από -34.77 μέχρι -33.5cm. Παρακάτω παρατίθεται ο κώδικας που βρίσκεται μέσα στο block της πρώτης if και αφορά τον καθορισμό κατακόρυφου ορίου:

```
// If both switches are pressed during boot, then store desired vertical limit offset to EEPROM
if (digitalRead(pinToControlMagnet)==LOW && digitalRead(pinToToggleInputMode)==LOW)
{
 int value=0;
 unsigned long past=0;
 do
 {
  if (millis()-past>500)
  {
   digitalWrite(led13,!digitalRead(led13));  // Built in LED is flashing during procedure.
   past=millis();
  }
  value=analogRead(pinForZoomPotensiometer)/4;
                                  // read potensiometer and map from 0-1023 to 0-255
  Serial.println("@!");
  Serial.print("Table Level= ");
  Serial.println(tableLevel+(float)value/200.0);
  Serial.println('$');
 } while (digitalRead(pinToControlMagnet)==LOW||digitalRead(pinToToggleInputMode)==LOW);
       // When both switches are released, then save the result
 EEPROM[3]=(byte)value;
 verticalLimit=tableLevel+(float)value/200.0;
}
```

Το LED 13 αναβοσβήνει κατά τη διάρκεια αυτής της διαδικασίας. Η millis() επιστρέφει το χρόνο που έχει περάσει από την εκκίνηση του Arduino. Άρα το LED 13 αναβοσβήνει με συχνότητα ενός δευτερολέπτου γιατί ανάβει ή σβήνει ανά 500ms.

1.10.5.2 Η δεύτερη if της setup()

Στον παρακάτω κώδικα παρατίθεται η δεύτερη εντολή if της setup() και αφορά την φόρτωση του αποθηκευμένου κατακόρυφου ορίου:

```
// If switch is pressed during boot then load vertical limit offset from EEPROM
 if (digitalRead(pinToControlLED)==LOW)
 {
  int value=EEPROM[3];
  unsigned long past=0;
  do
  {
   if (millis()-past>500)
   {
    digitalWrite(led13,!digitalRead(led13));  // Built in LED is flashing during procedure.
    past=millis();
   }
   Serial.println("@!");
   Serial.print("Table Level= ");
   Serial.println(tableLevel+(float)value/200.0);
   Serial.println('$');
  } while (digitalRead(pinToControlLED)==LOW);
  verticalLimit=tableLevel+(float)value/200.0;
 }
```

Εδώ απλά όταν πατηθεί ο Sw3 τυπώνεται στην σειριακή UART0 η τιμή που είναι αποθηκευμένη στην EEPROM και όταν αφεθεί ορίζεται ως νέο κατακόρυφο όριο αυτό που υπάρχει αποθηκευμένο στην EEPROM με παρόμοιο τρόπο με πριν.

1.10.6 Η συνάρτηση loop()

Στη συνάρτηση loop() πραγματοποιείται η κύρια λειτουργία του ρομπότ. Αρχικά ελέγχεται αν είναι πατημένο κάποιο από τα τρία μεγάλα κουμπιά και δίνει την κατάλληλη τιμή στην μεταβλητή controlMode ανάλογα με ποιο κουμπί πατήθηκε. Αν πατηθεί το μαύρο παίρνει τη τιμή 0, ενώ αν είναι ήδη mode 0 και πατηθεί το μαύρο στέλνεται ένα «@» στη σειριακή 1, UART1, αρκεί να μην έχει περάσει ένα δευτερόλεπτο από την προηγούμενη φορά που πατήθηκε και αφέθηκε. Αν πατηθεί το κόκκινο, τότε παίρνει την τιμή 1, εκτελεί τη μετακίνηση στη θέση (0,0,-33) σβήνοντας το μαγνήτη και μετά επαναλαμβάνει τη loop() μέχρι να αλλάξει το mode. Αν πατηθεί το κίτρινο παίρνει τη τιμή 2. Μετά, ανάλογα με τη τιμή της μεταβλητής αυτής μπαίνει και στο αντίστοιχο if block. Τέλος στέλνει στην UART0 κατάλληλο μήνυμα.

Εδώ να πούμε ότι στις μεταβλητές oldAngle_1, 2, 3 αποθηκεύονται οι θέσεις των δρομέων των σερβοκινητήρων και όχι οι γωνίες θ_i. Επίσης διευκρινίζεται ότι η Serial1 αφορά τη UART1 με την οποία γίνεται η επικοινωνία με την εξωτερική συσκευή, πχ. Raspberry Pi, ενώ η Serial αφορά τη UART0 στην οποία αποστέλλεται το μήνυμα εξόδου. Παρακάτω ακολουθεί ο κώδικας της συνάρτησης loop():

```
void loop()
  {
   if (digitalRead(pinForRemoteControl)==LOW)
            // If the Black button is pressed then change control mode to 0 (Remotely Controlled)
   {
    digitalWrite(led,HIGH);
    ledStatus=HIGH;

    currentTime=millis();
    // If button pressed when already in controlMode 0, then send a @, but
            //avoid sending multiple @s.
    if (controlMode==0&&currentTime-previousTime>intervalBetweenButtonPress)
                                    Serial1.write('@');
    previousTime=currentTime;

    controlMode=0;
    digitalWrite(yellowLed,LOW);
    digitalWrite(redLed,LOW);
    digitalWrite(greenLed,HIGH);
   }

   if (digitalRead(pinForStop)==LOW)
   {        // If stop (Red Button) is pressed turn off magnet and go horizontal (Mode 1)
    controlMode=1;
```

```
  digitalWrite(yellowLed,LOW);
  digitalWrite(greenLed,LOW);
  digitalWrite(orangeLed,LOW);
  digitalWrite(redLed,HIGH);
  digitalWrite(led13,LOW);
  digitalWrite(em,LOW);
  magStatus=0;

  servo_1.writeMicroseconds(pgm_read_word_near(servo1+offset_1));
  oldAngle1=offset_1;
  servo_2.writeMicroseconds(pgm_read_word_near(servo2+offset_2));
  oldAngle2=offset_2;
  servo_3.writeMicroseconds(pgm_read_word_near(servo3+offset_3));
  oldAngle3=offset_3;

  digitalWrite(led,LOW);  // Also turn off Delta Robot's LED
  ledStatus=LOW;
 }

 if (digitalRead(pinForLocalControl)==LOW)
 {        // Locally controlled with potensiometers (Mode 2)
  controlMode=2;
  digitalWrite(redLed,LOW);
  digitalWrite(greenLed,LOW);
  digitalWrite(orangeLed,LOW);
  digitalWrite(yellowLed,HIGH);
 }
 if (controlMode==0&&!Serial1.available()) digitalWrite(orangeLed,HIGH); // No data received yet
 if (controlMode==0&&Serial1.available())   // If there is data in Serial buffer, ...
 {
        Τα περιεχόμενα αυτού του block βρίσκονται στην παράγραφο 1.10.6.1
 } // Endif for control mode 0

 if (controlMode==2)   // Locally controlled with potensiometers
 {
        Τα περιεχόμενα αυτού του block βρίσκονται στην παράγραφο 1.10.6.2
 } // Endif for control mode 2

  // If there is enough space in buffer then write in order to avoid delay in code execution.
 if (Serial.availableForWrite()>62)
   {
        Τα περιεχόμενα αυτού του block βρίσκονται στην παράγραφο 1.10.6.3
   }
}
```

1.10.6.1 Το if block για το mode 0

Σε αυτή την υποπαράγραφο παρατίθεται ο κώδικας του block που αφορά την περίπτωση του mode 0, δηλαδή της λήψης συντεταγμένων από το Raspberry Pi. Αρχικά γίνεται έλεγχος αν υπάρχουν δεδομένα στη σειριακή. Αν δεν υπάρχουν επαναλαμβάνεται ο βρόγχος. Αν δεν γίνει αυτός ο έλεγχος και εκτελεστεί η Serial1.find που ακολουθεί, τότε αυτή θα περιμένει δεδομένα για 1 δευτερόλεπτο μέχρι να επιστρέψει τον αριθμό 0 στην if. Έτσι θα καθυστερεί ο κώδικας με αποτέλεσμα αν ο χρήστης θελήσει να αλλάξει mode, τότε θα είναι το σύστημα προσωρινά μη αποκρίσιμο. Βέβαια αυτός ο χρόνος του 1 δευτερολέπτου μπορεί να αλλάξει με τη «Serial1.setTimeout()», αλλά αν τον μικρύνουμε, τότε μπορεί να υπάρξει πρόβλημα με τις «Serial1.parseInt()» που ακολουθούν.

Η «Serial1.parseInt()» διαβάζει από τη σειριακή ένα-ένα τα ψηφία του αριθμού μέχρι να συναντήσει μη αριθμητικό χαρακτήρα αγνοώντας τυχόν αρχικούς μη αριθμητικούς χαρακτήρες που βρίσκονται πριν τα αριθμητικά ψηφία. Κατόπιν κάνει τη μετατροπή του από κείμενο σε ακέραιο και τον επιστρέφει. Αν από τη στιγμή που ξεκινήσει να εκτελείται δεν έχει συναντήσει καθόλου αριθμητικούς χαρακτήρες για ένα δευτερόλεπτο, τότε επιστρέφει 0. Αν συναντήσει αριθμητικούς χαρακτήρες και μετά δεν έρθουν νέα δεδομένα για ένα δευτερόλεπτο, τότε θεωρεί ότι δεν υπάρχουν άλλα και επιστρέφει τον αριθμό που έχει διαβάσει μέχρις στιγμής. Δηλαδή για να επιστρέψει έναν αριθμό πρέπει να υπάρχει ένας μη αριθμητικός χαρακτήρας μετά το τελευταίο ψηφίο του αριθμού ή να περάσει ένα δευτερόλεπτο χωρίς να έρθουν καθόλου δεδομένα.

Αυτός είναι και ο λόγος που αποφασίστηκε να υπάρχουν μη αριθμητικοί χαρακτήρες μεταξύ των συντεταγμένων. Σε αντίθετη περίπτωση δεν θα μπορούσε να ξεχωρίσει που τελειώνει ο ένας αριθμός και που αρχίζει ο άλλος. Για τον ίδιο λόγο αποφασίστηκε να υπάρχει ο «$» στο τέλος του μηνύματος, αλλιώς θα καθυστερούσε η εκτέλεση του κώδικα για 1 δευτερόλεπτο μέχρι να καταλάβει ότι δεν υπάρχει άλλο ψηφίο για την μεταβλητή «electromagnet». Με αυτή τη τεχνική συναντώντας το «$» καταλαβαίνει ότι ήρθαν όλα τα ψηφία του αριθμού και επιστρέφει χωρίς να περιμένει να περάσει το 1 δευτερόλεπτο. Βέβαια το «$» εν γένει σηματοδοτεί το τέλος του μηνύματος.

Συνεπώς, αν η συσκευή που στέλνει τις συντεταγμένες είναι πολύ αργή, και έχει ρυθμιστεί η «Serial.setTimeout()» σε μικρό χρόνο, τότε μπορεί να μην περιμένει αρκετά ώστε να έρθουν όλα τα ψηφία του αριθμού με αποτέλεσμα η λήψη των συντεταγμένων να είναι εσφαλμένη. Για αυτό πρέπει ο χρόνος να μην είναι μικρός και να γίνεται έλεγχος της ενδιάμεσης μνήμης πριν εκτελεστούν αυτές οι εντολές προκειμένου να μην καθυστερούν την εκτέλεση του κώδικα. Βέβαια αν καθυστερήσει ο κώδικας γιατί διαβάζουν από κάποιο αργό μηχάνημα, τότε αυτό

είναι κάτι που δεν μπορεί να αποφευχθεί γιατί ειδάλλως δεν θα είχαμε καθόλου επικοινωνία με την εξωτερική συσκευή.

Αν λοιπόν υπάρχουν bytes στη buffer, αρχικά αναζητείται ο χαρακτήρας «@». Αν δεν βρεθεί, τότε βγαίνει από το block και επαναλαμβάνεται ο βρόγχος. Αν βρεθεί, τότε ψάχνει 4 ακέραιους για τις συντεταγμένες και τον ηλεκτρομαγνήτη και μετατρέπει τις συντεταγμένες σε δεκαδικούς και cm διαιρώντας με 100 γιατί η εξωτερική συσκευή τις μεταδίδει πολλαπλασιασμένους με 100 και χωρίς δεκαδικά ψηφία. Αν η τιμή της z είναι μικρότερη από το οριζόντιο όριο, τότε γίνεται ίση με αυτό. Κατόπιν ψάχνει για το «$». Αν δεν το βρει, σημαίνει πως το μήνυμα δεν είναι σωστό και εκτελείται η «else validPosition=1». Αν τον βρει τότε θέτει κατάλληλα τον ηλεκτρομαγνήτη, εντός ή εκτός, και υπολογίζει και ελέγχει αν οι συντεταγμένες αντιστοιχούν σε έγκυρο σημείο. Αν όχι η «validPosition» γίνεται αρνητική. Ειδάλλως μηδέν.

Μετά στρογγυλοποιεί τις γωνίες, τις κάνει ακέραιο αριθμό, τις αφαιρεί από το offset και ελέγχει αν αντιστοιχούν σε θέση των δρομέων των σερβοκινητήρων μεταξύ 0 και 180 μοιρών. Αν όχι τότε η «validPosition» γίνεται -4. Ειδάλλως γίνεται 0 και ελέγχει αν οι δρομείς των σερβοκινητήρων βρίσκονται ήδη στην επιθυμητή θέση, «oldAngle_1,2,3», από προηγούμενη εκτέλεση της loop(). Αν όχι προσπελαύνει το στοιχείο του πίνακα «servo1,2,3», 0 ως 180, που αντιστοιχεί στην επιθυμητή θέση του δρομέα σε μοίρες, 0 ως 180, προκειμένου να προκύψει το εύρος σε μsec της άνω παρειάς του παλμού που θα σταλεί στον κάθε σερβοκινητήρα. Για την προσπέλαση αυτή χρησιμοποιείται η συνάρτηση «pgm_read_word_near()» γιατί οι πίνακες «servo1,2,3» έχουν οριστεί ως «PROGMEM», όπως αναφέρθηκε σε προηγούμενο κεφάλαιο. Με τη «servo_1,2,3.writeMicroseconds()» αλλάζει κατάλληλα το εύρος παλμού που στέλνεται στους σερβοκινητήρες. Αν έχει χρησιμοποιηθεί το όριο της z, τότε η «validPosition» γίνεται 2. Τέλος αν δεν υπάρχουν άλλα δεδομένα στη buffer εισερχομένων της UART1, στέλνεται ένα «@» στη UART1 για να ζητήσει νέα δεδομένα. Ο κώδικας αυτός παρουσιάζεται παρακάτω:

```
if (controlMode==0&&!Serial1.available()) digitalWrite(orangeLed,HIGH); // No data received yet
if (controlMode==0&&Serial1.available())
 {                      // If there is data in Serial buffer, ...
  digitalWrite(orangeLed,LOW);
  if (Serial1.find("@"))      // ..., then search for start character,@, and read x,y,z,electromagnet
   {
    x=(float)Serial1.parseInt()/100.0;
    y=(float)Serial1.parseInt()/100.0;
    z=(float)Serial1.parseInt()/100.0;
    electromagnet=Serial1.parseInt();
```

```
if (z<verticalLimit)
            // If the z coordinate is lower than the vertical limit then z is equal to that limit
{
 z=verticalLimit;
 verticalFlag=1;
}
else verticalFlag=0;

if (Serial1.find("$"))    // If end character, $, is found, then no data is missing.
 {
  if (electromagnet==0&&magStatus==1)
    {              // If magnet should be off and is on, then turn it off
     digitalWrite(em,LOW);
     magStatus=0;
    }
  if (electromagnet==1&&magStatus==0)
    {              // If magnet should be on and is off, then turn it on
     digitalWrite(em,HIGH);
     magStatus=1;
    }

   retStat=delta_calcInverse(x,y,z,t1,t2,t3); // Calculate theta1, theta2 and theta3 angles
   angle_1=offset_1-round(t1);
   angle_2=offset_2-round(t2);
   angle_3=offset_3-round(t3);

   // If return Status is 0 (OK), all the angles are within the limits (0 to 180), then
    //... write to the corresponding servos
   if (retStat==0 && angle_1>=0 && angle_1<=180 && angle_2>=0 &&
    angle_2<=180 && angle_3>=0 && angle_3<=180)
     {
       validPosition=0;    // If everything is OK then validPosition=0
       if (oldAngle1!=angle_1)
                  // If the desired new angle (angle_1) is different from the current angle
        {         // (oldAngle1) then write it to the servo1
          servo_1.writeMicroseconds(pgm_read_word_near(servo1+angle_1));
          oldAngle1=angle_1;
        }
       if (oldAngle2!=angle_2)
                  // If the desired new angle (angle_2) is different from the current angle
        {         //(oldAngle2) then write it to the servo2
          servo_2.writeMicroseconds(pgm_read_word_near(servo2+angle_2));
          oldAngle2=angle_2;
        }
```

```
            if (oldAngle3!=angle_3)
                    // If the desired new angle (angle_3) is different from the current angle
             {      //(oldAngle3) then write it to the servo3
               servo_3.writeMicroseconds(pgm_read_word_near(servo3+angle_3));
               oldAngle3=angle_3;
             }
          }
        else validPosition=retStat==0?-4:retStat;  // If return Status==0(OK) but any of the above
                                    //… limitations is not met then validPosition=-4, else =retStat
       } // Endif for Serial.find("$")
      else validPosition=1;
                    // If bad data has been received or data is missing, then validPosition is 1
      if (validPosition==0 && verticalFlag==1) validPosition=2;
         // If everything else is OK but vertical limit has been violated then validPosition=2
      digitalWrite(led13,validPosition==0?HIGH:LOW);
    } // Endif for Serial.find("@")

  if (!Serial1.available()) Serial1.write('@'); // Send @ to ask for new data
} // Endif for control mode 0
```

1.10.6.2 Το if block για το mode 2

Εδώ διαβάζει από τα ποτενσιόμετρα την αντίστοιχη θέση των δρομέων. Αν ο διακόπτης Sw1 είναι κλειστός, λογικό LOW, τότε ανάβει τον ηλεκτρομαγνήτη, ενώ αν είναι ανοικτός, λογικό HIGH, τότε τον σβήνει. Ομοίως αν ο Sw3 είναι κλειστός, τότε αναβοσβήνει το LED ασφαλείας, ενώ αν είναι ανοικτός, τότε αυτό σβήνει.

Αν ο Sw2 είναι ανοιχτός, λογικό HIGH, τότε διαβάζει από τα ποτενσιόμετρα Pot1 έως Pot4 με τη χρήση της «analogRead()» τις τιμές των συντεταγμένων και τις προσαρμόζει κατάλληλα, όπως έχει αναλυθεί σε προηγούμενο κεφάλαιο. Κατόπιν κάνει παρόμοιους ελέγχους όπως και στην περίπτωση του mode 0 και μετακινεί την κεφαλή στην αντίστοιχη θέση. Η διαδικασία είναι παρόμοια με αυτή που αναλύθηκε στην προηγούμενη υποπαράγραφο με την μόνη διαφορά ότι δεν διαβάζει από τη σειριακή θύρα.

Αν ο Sw2 είναι κλειστός, λογικό LOW, τότε διαβάζει από τα ποτενσιόμετρα Pot1 έως Pot3 τις τιμές των γωνιών των σερβοκινητήρων και αφού ελέγξει ότι είναι εντός του εύρους των 0 έως 180 μοιρών, τότε προχωράει στην αναζήτηση του παλμού από τον αντίστοιχο πίνακα βαθμονόμησης και τον στέλνει στον αντίστοιχο σερβοκινητήρα. Παρακάτω ακολουθεί ο κώδικας του block αυτού:

```
if (controlMode==2)   // Locally controlled with potensiometers
 {
  ledState=digitalRead(pinToControlLED); // Inverse Logic
  if (ledState==HIGH&&ledStatus==1)
    {                   // If Delta Robot LED is on and the switch is open, then turn the LED off
      digitalWrite(led,LOW);
      ledStatus=0;
    }
  if (ledState==LOW&&ledStatus==0)
    {                   // If Delta Robot LED is off and the switch is closed, then turn the LED on
      digitalWrite(led,HIGH);
      ledStatus=1;
    }

  coord=digitalRead(pinToToggleInputMode);  // Coordinates or angles input
  if (coord==HIGH)
  // If switch is open, then read the x,y,z coordinates from potensiometers and act accordingly
  {
    multipl=5.0+17.0*(float)analogRead(pinForZoomPotensiometer)/1023.0;
 // Multiplier factor that defines the scale and range of x,y values that read the potensiometers
    x=(float)analogRead(pinForXorAngle1Potensiometer)*2.0*multipl/1023.0-multipl;
                        // Reads the x value and map it according to multiplier factor
```

```
y=(float)analogRead(pinForYorAngle2Potensiometer)*2.0*multipl/1023.0-multipl;
                                    // Reads the y value and map it according to multiplier factor
z=(float)analogRead(pinForZorAngle3Potensiometer)*11.0/1023.0-40.0;
                          // Reads the z value and map without taking into account the multiplier factor

if (z<verticalLimit)
{                 // If the z coordinate is lower than the vertical limit then z is equal to that limit
 z=verticalLimit;
 verticalFlag=1;
}
else verticalFlag=0;

retStat=delta_calcInverse(x,y,z,t1,t2,t3);
                                    // Calculate theta1, theta2 and theta3 angles and do as above
angle_1=offset_1-round(t1);
angle_2=offset_2-round(t2);
angle_3=offset_3-round(t3);
if (retStat==0 && angle_1>=0 && angle_1<=180 && angle_2>=0 &&
                        angle_2<=180 && angle_3>=0 && angle_3<=180)
  {
    validPosition=0;
    if (oldAngle1!=angle_1)
    {
      servo_1.writeMicroseconds(pgm_read_word_near(servo1+angle_1));
      oldAngle1=angle_1;
    }
    if (oldAngle2!=angle_2)
    {
      servo_2.writeMicroseconds(pgm_read_word_near(servo2+angle_2));
      oldAngle2=angle_2;
    }
    if (oldAngle3!=angle_3)
    {
      servo_3.writeMicroseconds(pgm_read_word_near(servo3+angle_3));
      oldAngle3=angle_3;
    }
  }
  else validPosition=retStat==0?-4:retStat;  // If return Status==0(OK) but any of the above
                                    // ...limitations is not met then validPosition=-4, else =retStat
  if (validPosition==0 && verticalFlag==1) validPosition=2;
            // If everything else is OK but vertical limit has been violated then validPosition=2
  digitalWrite(led13,validPosition==0?HIGH:LOW);
}  // Endif coord High.
```

```
 if (coord==LOW)
 {       // If switch is closed, then read the desired angles of the servos from potensiometers
   digitalWrite(led13,LOW);
         // There is no reason to be on. Every angle between 0 to 180 can be achieved by servos.
   angle_1=analogRead(pinForXorAngle1Potensiometer)*180.0/1023.0;
   angle_2=analogRead(pinForYorAngle2Potensiometer)*180.0/1023.0;
   angle_3=analogRead(pinForZorAngle3Potensiometer)*180.0/1023.0;

   if (angle_1>=0 && angle_1<=180 && angle_2>=0 &&
                        angle_2<=180 && angle_3>=0 && angle_3<=180)
     {
       if (oldAngle1!=angle_1)
       {
         servo_1.writeMicroseconds(pgm_read_word_near(servo1+angle_1));
         oldAngle1=angle_1;
       }
       if (oldAngle2!=angle_2)
       {
         servo_2.writeMicroseconds(pgm_read_word_near(servo2+angle_2));
         oldAngle2=angle_2;
       }
       if (oldAngle3!=angle_3)
       {
         servo_3.writeMicroseconds(pgm_read_word_near(servo3+angle_3));
         oldAngle3=angle_3;
       }
     }
 }  // Endif coord Low

 electromagnet=digitalRead(pinToControlMagnet); // Inverse Logic
 if (electromagnet==HIGH&&magStatus==1)
   {                    // If magnet switch is open and magnet is on, then turn it off
     digitalWrite(em,LOW);
     magStatus=0;
   }
 if (electromagnet==LOW&&magStatus==0)
   {                    // If magnet switch is closed and magnet is off, then turn it on
     digitalWrite(em,HIGH);
     magStatus=1;
   }
} // Endif for control mode 2
```

1.10.6.3 Το block εγγραφής του μηνύματος εξόδου

Σε αυτή την υποπαράγραφο παρουσιάζονται οι εντολές για την αποστολή στην σειριακή θύρα UART0 του αντίστοιχου μηνύματος εξόδου αναλόγως την περίπτωση, όπως έχει αναλυθεί σε προηγούμενη παράγραφο:

```
// If there is enough space in buffer then write in order to avoid delay in code execution.
if (Serial.availableForWrite()>62)
  {
    if (controlMode==2&&coord==LOW)
      {
        Serial.println("@#");
        Serial.print("Servo1= ");
        Serial.println(angle_1);
        Serial.print("Servo2= ");
        Serial.println(angle_2);
        Serial.print("Servo3= ");
        Serial.println(angle_3);
        Serial.println('$');
      }
    else if (controlMode==1)
      {
        Serial.println("@%");
        Serial.println(" Paused!");
        Serial.println('$');
      }
    else
      {
        Serial.println('@');
        Serial.print("X= ");
        Serial.println(x);
        Serial.print("Y= ");
        Serial.println(y);
        Serial.print("Z= ");
        Serial.println(z);
        if (validPosition==0) Serial.println(" OK");
        if (validPosition<0) Serial.println(" Bad Position!");
        if (validPosition==1) Serial.println(" Bad Data!");
        if (validPosition==2) Serial.println(" z Limit!");
        Serial.print(validPosition);
        Serial.println('$');
      }
  }
```

1.11 Ολόκληρος ο πηγαίος κώδικας του Delta Robot

Επειδή στην προηγούμενη παράγραφο, 1.10, ο κώδικας παρουσιάζεται τμηματικά για καλύτερη ανάλυση, εδώ παρουσιάζεται ξανά, αλλά συγκεντρωτικά χωρίς σχόλια:

```
#include <EEPROM.h>
#include <Servo.h> // Servo library

Servo servo_1; // create servo object to control a servo
Servo servo_2; // a maximum of eight servo objects can be created
Servo servo_3;

const int servo1[]PROGMEM =
 {641,651,659,668,676,680,690,700,706,714,722,730,737,746,754,763,772,779,
 788,796,808,815,825,833,843,854,862,871,878,888,896,
 903,910,919,932,943,951,960,969,976,986,996,1007,1016,1022,1032,1043,1050,
 1060,1071,1081,1089,1097,1108,1116,1127,1137,1147,
 1158,1169,1180,1189,1198,1206,1216,1226,1236,1248,1258,1268,1274,1285,1300,
 1307,1317,1328,1338,1348,1357,1364,1374,1386,1395,
 1406,1413,1421,1432,1443,1454,1463,1474,1483,1492,1498,1508,1520,1529,1538,
 1547,1557,1566,1578,1586,1596,1607,1615,1625,1631,
 1639,1655,1664,1675,1683,1694,1703,1711,1726,1737,1746,1756,1765,1770,
 1780,1791,1801,1812,1821,1830,1839,1851,1861,1867,1877,
 1887,1894,1902,1911,1924,1933,1940,1948,1956,1968,1977,1985,1994,2004,2011,
 2023,2030,2040,2049,2057,2065,2075,2084,2093,2103,
 2109,2119,2129,2137,2146,2153,2161,2170,2180,2188,2199,2208,2220,2228,2234,
 2242,2252,2264,2271,2279,2288,2296,2305};

const int servo2[]PROGMEM =
 {587,594,600,608,612,620,627,633,641,651,659,662,672,680,688,695,
 703,713,721,730,738,744,752,761,768,777,783,791,802,809,819,
 827,835,847,856,861,873,883,889,896,907,916,925,935,941,953,961,970,976,
 984,995,1003,1012,1024,1033,1043,1050,1059,1069,1077,
 1086,1100,1105,1117,1128,1136,1146,1156,1165,1178,1185,1194,1206,1213,1225,
 1232,1241,1251,1261,1268,1275,1288,1298,1307,1321,
 1330,1335,1348,1356,1369,1373,1384,1396,1408,1418,1426,1437,1449,1460,
 1469,1476,1488,1498,1507,1520,1531,1540,1549,1560,1572,
 1583,1593,1601,1614,1623,1637,1645,1656,1672,1682,1691,1700,1710,1717
 ,1729,1743,1754,1764,1776,1789,1800,1812,1818,1829,1841,
 1851,1862,1869,1882,1888,1898,1911,1920,1931,1943,1954,1961,1970,1976,1988,
 2000,2008,2016,2029,2038,2047,2056,2064,2080,2090,
 2099,2109,2120,2132,2139,2147,2159,2170,2178,2187,2198,2205,2215,2228,
 2239,2249,2256,2265,2276,2282,2296};
```

```
const int servo3[]PROGMEM =
 {587,596,603,609,616,626,632,640,648,657,665,671,679,688,696,701,708,716,722,
 729,739,749,755,764,771,780,787,794,806,814,820,
 827,834,843,852,861,870,878,888,896,904,911,923,931,941,952,959,968,980,987,
 995,1003,1014,1023,1034,1042,1052,1060,1071,1080,
 1088,1096,1108,1118,1128,1138,1146,1158,1167,1175,1185,1196,1206,
 1215,1224,1235,1243,1254,1263,1272,1284,1294,1303,1315,1325,
 1333,1346,1356,1365,1376,1386,1395,1402,1413,1423,1432,1440,1451,1460,
 1469,1477,1486,1496,1508,1518,1524,1534,1544,1554,1563,
 1573,1584,1596,1603,1610,1621,1630,1637,1652,1661,1670,1680,1688,1698,1706,
 1719,1728,1738,1746,1757,1769,1779,1788,1793,1802,
 1813,1821,1830,1843,1852,1862,1874,1882,1893,1902,1912,1920,1931,1938,
 1946,1957,1964,1972,1981,1991,1998,2010,2017,2027,2036,
 2042,2053,2060,2070,2079,2086,2095,2102,2111,2119,2128,2137,2146,2153,2162,
 2169,2175,2184,2193,2202,2208};


// robot geometry
const float e = 8.0;     // end effector
const float f = 23.32;     // base
const float re = 35.0;
const float rf = 6.0;

// trigonometric constants
const float sqrt3 = sqrt(3.0);
const float pi = 3.141592653;    // PI
const float sin120 = sqrt3/2.0;
const float cos120 = -0.5;
const float tan60 = sqrt3;
const float sin30 = 0.5;
const float tan30 = 1.0/sqrt3;

const int offset_1 =111; // rf are horizontal
const int offset_2 =110;
const int offset_3 =116;

const int em = 12; // Defines the pin which controls the electromagnet
const int led = 8; // Defines the pin which controls the LED of the Delta Robot

const int pinForRemoteControl=52; // Low to control via external device
const int pinForStop=6;  // Low to stop
const int pinForLocalControl=48; // Low to control delta Robot via potensiometers
const int pinToControlMagnet=5; // Input Pin that defines magnet state when controlMode is 2.
                                //Low for On, High for Off.
```

```
const int pinToControlLED=7; // Input Pin that defines LED state when controlMode is 2.
                             // …Low for On, High for Off.
const int pinToToggleInputMode=4;  // When controlMode is 2 defines whether input
                           //… values are angles or coordinates. Low for Angle, High for Coordinates
const int redLed=3;   // Pin for red led. Led is On at stop mode
const int yellowLed=2;  // Pin for yellow led. Led is On at local control mode
const int orangeLed=26; // Pin for orange led. Led is On when waiting to receive data at
                                    //remote control mode
const int greenLed=40;   // Pin for green led. Led is On at remote control mode
const int led13=13;   // Pin for Arduino built in LED 13. Led 13 is On when the position is valid and
                                    // off otherwise or in Pause mode.
const int pinForXorAngle1Potensiometer=1; // Pin connected to the potensiometer that defines X
                                                // coordinate or Angle1
const int pinForYorAngle2Potensiometer=5; // Pin connected to the potensiometer that defines Y
                                                // coordinate or Angle2
const int pinForZorAngle3Potensiometer=8; // Pin connected to the potensiometer that defines Z
                                                // coordinate or Angle3
const int pinForZoomPotensiometer=15; // Pin connected to the potensiometer that defines
                                                // Zoom factor for the other three

 /* Also built in led 13 is on when received coordinates are valid and off when bad
  * coordinates have been received at control mode 0 (remote)
  * or at by the user at control mode 2 when receiving coordinates. At any other cases
  * (mode 1 or mode 2 when receiving angles) led 13 is off.
  */

const float tableLevel=-34.77;  // The z coordinate of the table
float verticalLimit=tableLevel; // The lower vertical limit that delta robot head is allowable to reach
int verticalFlag=0;
          // If the given z coordinate is lower than vertical Limit then vertical flag=1 else =0.

const unsigned long intervalBetweenButtonPress=1000; // For button debouncing
unsigned long currentTime=0, previousTime=0;

float t1;
float t2;
float t3;
int angle_1=0,angle_2=0,angle_3=0;          //  Desired servo angles
int oldAngle1=0, oldAngle2=0, oldAngle3=0;// Current servo angles

float x=0.0, y=0.0, z=-33.0;         //  x,y,z coordinates
float multipl=1.0;
int electromagnet=0, ledState=0, magStatus=0, ledStatus=0;
                  // ...State= the desired condition (On or Off),...Status =the current condition.
```

```
int retStat=0; // Return status of angles calculation. 0 If everything is OK, -1, -2, -3 if
               // one two or three angles are bad.
int controlMode=0; // Defines who controls the servo. 0 for external serial device, 1 to
                   //stop (Pause mode) and 2 for potensiometers
int coord=HIGH;
int validPosition=0; // Equals 0 if the given x,y,z coordinates correspond to a valid position,
                // … negative if not and 1 when bad data is received.
                            // If it is 2 then the given z coordinate is less than vertical limit.



int delta_calcAngleYZ(float x0, float y0, float z0, float &theta)
{
 float y1 = -0.5 * 0.57735 * f; // f/2 * tg 30
 //float y1 = yy1;
 y0 -= 0.5 * 0.57735    * e;    // shift center to edge
 // z = a + b*y
 float a = (x0*x0 + y0*y0 + z0*z0 +rf*rf - re*re - y1*y1)/(2*z0);
 float b = (y1-y0)/z0;
 // discriminant
 float d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
 if (d < 0) return -1; // non-existing point
 float yj = (y1 - a*b - sqrt(d))/(b*b + 1); // choosing outer point
 float zj = a + b*yj;
 theta = 180.0*atan(-zj/(y1 - yj))/pi + ((yj>y1)?180.0:0.0);
 if ((theta < -180) || (theta > 180))
    return -1;
 return 0;
}

// inverse kinematics: (x0, y0, z0) -> (theta1, theta2, theta3)
// returned status: 0=OK, negative=non-existing position, the negative number is
  // the number of wrong angles
int delta_calcInverse(float x0, float y0, float z0, float &theta1, float &theta2, float &theta3)
{
 theta1 = theta2 = theta3 = 0;
 int stat1 = delta_calcAngleYZ(x0, y0, z0, theta1);
 int stat2 = delta_calcAngleYZ(x0*cos120 + y0*sin120, y0*cos120-x0*sin120, z0, theta2);
                                                        // rotate coords to +120 deg
 int stat3 = delta_calcAngleYZ(x0*cos120 - y0*sin120, y0*cos120+x0*sin120, z0, theta3);
                                                        // rotate coords to -120 deg
 return stat1+stat2+stat3;
}
```

```
void setup()
 {
  Serial.begin(9600);   // To output data to Serial monitor or any other connected device.
  Serial1.begin(9600);  // Communication with external device for remote control.
                                          // Used for data input, such as coordinates.
  pinMode(led13, OUTPUT);
  pinMode(em, OUTPUT);
  pinMode(led, OUTPUT);

  digitalWrite(led13,HIGH);
  digitalWrite(em,LOW);
  digitalWrite(led,LOW);

  servo_1.attach(9,500,2500); // attaches the servo on pin 9 to the servo object
  servo_2.attach(10,400,2500); // attaches the servo on pin 10 to the servo object
  servo_3.attach(11,400,2500); // attaches the servo on pin 11 to the servo object

   //all servos horizontal (x,y,z)=(0,0,-31)
  servo_1.writeMicroseconds(pgm_read_word_near(servo1+offset_1));
  oldAngle_1=offset_1;
  servo_2.writeMicroseconds(pgm_read_word_near(servo2+offset_2));
  oldAngle_2=offset_2;
  servo_3.writeMicroseconds(pgm_read_word_near(servo3+offset_3));
  oldAngle_3=offset_3;

  delay(100);

  pinMode(redLed,OUTPUT);
  digitalWrite(redLed,HIGH);
  delay (100);
  pinMode(yellowLed,OUTPUT);
  digitalWrite(yellowLed,HIGH);
  delay (100);
  pinMode(greenLed,OUTPUT);
  digitalWrite(greenLed,HIGH);
  delay(100);
  pinMode(orangeLed,OUTPUT);
  digitalWrite(orangeLed,HIGH);
  delay(100);

  pinMode(pinForRemoteControl,INPUT_PULLUP);
  pinMode(pinForStop,INPUT_PULLUP);
  pinMode(pinForLocalControl,INPUT_PULLUP);
  pinMode(pinToControlMagnet,INPUT_PULLUP);
```

```
pinMode(pinToControlLED,INPUT_PULLUP);
pinMode(pinToToggleInputMode,INPUT_PULLUP);

delay(500);

 // If both switches are pressed during boot, then store desired vertical limit offset to EEPROM
if (digitalRead(pinToControlMagnet)==LOW && digitalRead(pinToToggleInputMode)==LOW)
{
  int value=0;
  unsigned long past=0;
  do     // When both switches are released, then save the result
  {
    if (millis()-past>500)
    {
      digitalWrite(led13,!digitalRead(led13));  // Built in LED is flashing during procedure.
      past=millis();
    }
    value=analogRead(pinForZoomPotensiometer)/4;
                                    // read potensiometer and map from 0-1023 to 0-255
    Serial.println("@!");
    Serial.print("Table Level= ");
    Serial.println(tableLevel+(float)value/200.0);
    Serial.println('$');
  } while (digitalRead(pinToControlMagnet)==LOW||digitalRead(pinToToggleInputMode)==LOW);
          // When both switches are released, then save the result
  EEPROM[3]=(byte)value;
  verticalLimit=tableLevel+(float)value/200.0;
}

// If switch is pressed during boot then load vertical limit offset from EEPROM
if (digitalRead(pinToControlLED)==LOW)
{
  int value=EEPROM[3];
  unsigned long past=0;
  do
  {
    if (millis()-past>500)
    {
      digitalWrite(led13,!digitalRead(led13));  // Built in LED is flashing during procedure.
      past=millis();
    }
    Serial.println("@!");
    Serial.print("Table Level= ");
    Serial.println(tableLevel+(float)value/200.0);
    Serial.println('$');
```

```
    } while (digitalRead(pinToControlLED)==LOW);
    verticalLimit=tableLevel+(float)value/200.0;
   }

   digitalWrite(led13,HIGH);  // Led 13 is on because servos are in valid position (0,0,-31)

   digitalWrite(redLed,LOW);
   delay(100);
   digitalWrite(yellowLed,LOW);
   delay(100);
   digitalWrite(greenLed,LOW);
   delay(100);
   digitalWrite(orangeLed,LOW);
   delay(100);


   digitalWrite(led,HIGH); // Turn on Delta Robot's LED
   ledStatus=1;
   digitalWrite(greenLed,HIGH);  // Default mode is 0 (Remotely controlled)
   Serial1.write('@'); // Informs external device that delta robot is ready to receive data.

  }


void loop()
 {

  if (digitalRead(pinForRemoteControl)==LOW)

  {          // If the Black button is pressed then change control mode to 0 (Remotely Controlled)
   digitalWrite(led,HIGH);
   ledStatus=HIGH;

   currentTime=millis();
                        // If button pressed when already in controlMode 0, then send a @, but
                        //avoid sending multiple @s.
   if (controlMode==0&&currentTime-previousTime>intervalBetweenButtonPress)
                                    Serial1.write('@');
   previousTime=currentTime;

   controlMode=0;
   digitalWrite(yellowLed,LOW);
   digitalWrite(redLed,LOW);
   digitalWrite(greenLed,HIGH);
  }
```

```
if (digitalRead(pinForStop)==LOW)
{          // If stop (Red Button) is pressed turn off magnet and go horizontal (Mode 1)
  controlMode=1;
  digitalWrite(yellowLed,LOW);
  digitalWrite(greenLed,LOW);
  digitalWrite(orangeLed,LOW);
  digitalWrite(redLed,HIGH);
  digitalWrite(led13,LOW);
                        // There is no reason examining whether the position is valid or not since
                        // it is already known that it is valid

  digitalWrite(em,LOW);
  magStatus=0;

  servo_1.writeMicroseconds(pgm_read_word_near(servo1+offset_1));
  oldAngle_1=offset_1;
  servo_2.writeMicroseconds(pgm_read_word_near(servo2+offset_2));
  oldAngle_2=offset_2;
  servo_3.writeMicroseconds(pgm_read_word_near(servo3+offset_3));
  oldAngle_3=offset_3;

  digitalWrite(led,LOW);  // Also turn off Delta Robot's LED
  ledStatus=LOW;
}

if (digitalRead(pinForLocalControl)==LOW)
{                       // Locally controlled with potensiometers (Mode 2)
  controlMode=2;
  digitalWrite(redLed,LOW);
  digitalWrite(greenLed,LOW);
  digitalWrite(orangeLed,LOW);
  digitalWrite(yellowLed,HIGH);
}

if (controlMode==0&&!Serial1.available()) digitalWrite(orangeLed,HIGH); //No data received yet
if (controlMode==0&&Serial1.available())   // If there is data in Serial buffer, ...
  {

    digitalWrite(orangeLed,LOW);

    if (Serial1.find("@"))     // ..., then search for start character,@, and read x,y,z,electromagnet
     {
       x=(float)Serial1.parseInt()/100.0;
       y=(float)Serial1.parseInt()/100.0;
```

```
z=(float)Serial1.parseInt()/100.0;
electromagnet=Serial1.parseInt();

if (z<verticalLimit)
{         // If the z coordinate is lower than the vertical limit then z is equal to that limit
 z=verticalLimit;
 verticalFlag=1;
}
else verticalFlag=0;

if (Serial1.find("$"))    // If end character, $, is found, then no data is missing.
 {
  if (electromagnet==0&&magStatus==1)
     {              // If magnet should be off and is on, then turn it off
      digitalWrite(em,LOW);
      magStatus=0;
     }
  if (electromagnet==1&&magStatus==0)
     {              // If magnet should be on and is off, then turn it on
      digitalWrite(em,HIGH);
      magStatus=1;
     }

   retStat=delta_calcInverse(x,y,z,t1,t2,t3); // Calculate theta1, theta2 and theta3 angles
   angle_1= offset_1-round(t1);
   angle_2= offset_2-round(t2);
   angle_3= offset_3-round(t3);

   // If return Status is 0 (OK), all the angles are within the limits (0 to 180), then
   // write to the corresponding servos
   if (retStat==0 && angle_1>=0 && angle_1<=180 && angle_2>=0 &&
    angle_2<=180 && angle_3>=0 && angle_3<=180)
     {
       validPosition=0;    // If everything is OK then validPosition=0
       if (oldAngle1!=angle_1)
                              // If the desired new angle (angle_1) is different from the
        {                     // current angle (oldAngle1) then write it to the servo1
          servo_1.writeMicroseconds(pgm_read_word_near(servo1+angle_1));
          oldAngle1=angle_1;
        }
       if (oldAngle2!=angle_2)
                              // If the desired new angle (angle_2) is different from the
        {                     // current angle (oldAngle2) then write it to the servo2
          servo_2.writeMicroseconds(pgm_read_word_near(servo2+angle_2));
          oldAngle2=angle_2;
```

```
            }
            if (oldAngle3!=angle_3)
                                    // If the desired new angle (angle_3) is different from the
            {                       // current angle (oldAngle3) then write it to the servo3
              servo_3.writeMicroseconds(pgm_read_word_near(servo3+angle_3));
              oldAngle3=angle_3;
            }
          }
         else validPosition=retStat==0?-4:retStat;
                        // If return Status==0(OK) but any of the above limitations is
                        //not met then validPosition=-4, else =retStat
       } // Endif for Serial.find("$")
       else validPosition=1;   // If bad data has been received or data is missing,
                                //then validPosition is 1
       if (validPosition==0 && verticalFlag==1) validPosition=2;
          // If everything else is OK but vertical limit has been violated then validPosition=2
       digitalWrite(led13,validPosition==0?HIGH:LOW);
     } // Endif for Serial.find("@")

   if (!Serial1.available()) Serial1.write('@'); // Send @ to ask for new data
 } // Endif for control mode 0

 if (controlMode==2)   // Locally controlled with potensiometers
 {
  ledState=digitalRead(pinToControlLED); // Inverse Logic
  if (ledState==HIGH&&ledStatus==1)
    {                   // If Delta Robot LED is on and the switch is open, then turn the LED off
      digitalWrite(led,LOW);
      ledStatus=0;
    }
  if (ledState==LOW&&ledStatus==0)
    {      // If Delta Robot LED is off and the switch is closed, then turn the LED on
      digitalWrite(led,HIGH);
      ledStatus=1;
    }

  coord=digitalRead(pinToToggleInputMode);  // Coordinates or angles input
  if (coord==HIGH)
// If switch is open, then read the x,y,z coordinates from potensiometers and act accordingly
  {
   multipl=5.0+17.0*(float)analogRead(pinForZoomPotensiometer)/1023.0;
   // Multiplier factor that defines the scale and range of x,y values that read the potensiometers
   x=(float)analogRead(pinForXorAngle1Potensiometer)*2.0*multipl/1023.0-multipl;
            // Reads the x value and map it according to multiplier factor
```

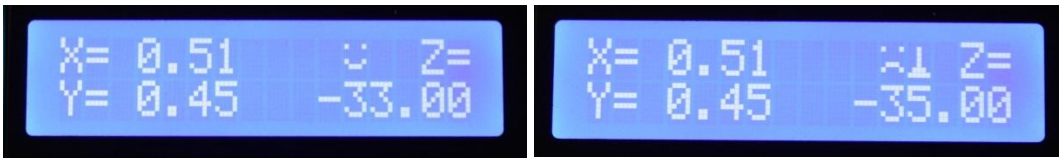
```
y=(float)analogRead(pinForYorAngle2Potensiometer)*2.0*multipl/1023.0-multipl;
        // Reads the y value and map it according to multiplier factor
z=(float)analogRead(pinForZorAngle3Potensiometer)*11.0/1023.0-40.0;
        // Reads the z value and map without taking into account the multiplier factor

if (z<verticalLimit)   limit
{                       // If the z coordinate is lower than the vertical limit then z is equal to that
 z=verticalLimit;
 verticalFlag=1;
}
else verticalFlag=0;

retStat=delta_calcInverse(x,y,z,t1,t2,t3);
                        // Calculate theta1, theta2 and theta3 angles and do as above
angle_1=offset_1-round(t1);
angle_2=offset_2-round(t2);
angle_3=offset_3-round(t3);
if (retStat==0 && angle_1>=0 && angle_1<=180 && angle_2>=0 &&
                        angle_2<=180 && angle_3>=0 && angle_3<=180)
  {
    validPosition=0;
    if (oldAngle1!=angle_1)
    {
      servo_1.writeMicroseconds(pgm_read_word_near(servo1+angle_1));
      oldAngle1=angle_1;
    }
    if (oldAngle2!=angle_2)
    {
      servo_2.writeMicroseconds(pgm_read_word_near(servo2+angle_2));
      oldAngle2=angle_2;
    }
    if (oldAngle3!=angle_3)
    {
      servo_3.writeMicroseconds(pgm_read_word_near(servo3+angle_3));
      oldAngle3=angle_3;
    }
  }
  else validPosition=retStat==0?-4:retStat;
  // If return Status==0(OK) but any of the above limitations is not met then
        //... validPosition=-4, else =retStat
  if (validPosition==0 && verticalFlag==1) validPosition=2;
        // If everything else is OK but vertical limit has been violated then validPosition=2
  digitalWrite(led13,validPosition==0?HIGH:LOW);
}  // Endif coord High.
```

```
 if (coord==LOW)   // If switch is closed, then read the desired angles of the servos from
                   // … potensiometers
 {
   digitalWrite(led13,LOW);   // There is no reason to be on. Every angle between 0 to 180
                              // can be achieved by servos.
   angle_1=analogRead(pinForXorAngle1Potensiometer)*180.0/1023.0;
   angle_2=analogRead(pinForYorAngle2Potensiometer)*180.0/1023.0;
   angle_3=analogRead(pinForZorAngle3Potensiometer)*180.0/1023.0;

   if (angle_1>=0 && angle_1<=180 && angle_2>=0 &&
                     angle_2<=180 && angle_3>=0 && angle_3<=180)
     {
       if (oldAngle1!=angle_1)
       {
         servo_1.writeMicroseconds(pgm_read_word_near(servo1+angle_1));
         oldAngle1=angle_1;
       }
       if (oldAngle2!=angle_2)
       {
         servo_2.writeMicroseconds(pgm_read_word_near(servo2+angle_2));
         oldAngle2=angle_2;
       }
       if (oldAngle3!=angle_3)
       {
         servo_3.writeMicroseconds(pgm_read_word_near(servo3+angle_3));
         oldAngle3=angle_3;
       }
     }
 }  // Endif coord Low

 electromagnet=digitalRead(pinToControlMagnet); // Inverse Logic
 if (electromagnet==HIGH&&magStatus==1)
    {        // If magnet switch is open and magnet is on, then turn it off
     digitalWrite(em,LOW);
     magStatus=0;
   }
 if (electromagnet==LOW&&magStatus==0)
   {                  // If magnet switch is closed and magnet is off, then turn it on
     digitalWrite(em,HIGH);
     magStatus=1;
   }
} // Endif for control mode 2

 // If there is enough space in buffer then write in order to avoid delay in code execution.
if (Serial.availableForWrite()>62)
```

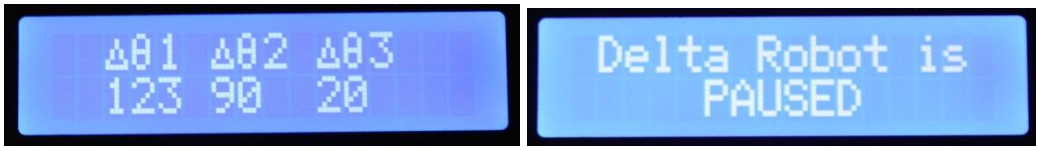
```
    {
      if (controlMode==2&&coord==LOW)
        {
          Serial.println("@#");
          Serial.print("Servo1= ");
          Serial.println(angle_1);
          Serial.print("Servo2= ");
          Serial.println(angle_2);
          Serial.print("Servo3= ");
          Serial.println(angle_3);
          Serial.println('$');
        }
      else if (controlMode==1)
        {
          Serial.println("@%");
          Serial.println(" Paused!");
          Serial.println('$');
        }
      else
        {
          Serial.println('@');
          Serial.print("X= ");
          Serial.println(x);
          Serial.print("Y= ");
          Serial.println(y);
          Serial.print("Z= ");
          Serial.println(z);
          if (validPosition==0) Serial.println(" OK");
          if (validPosition<0) Serial.println(" Bad Position!");
          if (validPosition==1) Serial.println(" Bad Data!");
          if (validPosition==2) Serial.println(" z Limit!");
          Serial.print(validPosition);
          Serial.println('$');
        }
    }
}
```

1.12 Ο Arduino UNO

Όπως αναφέρθηκε χρησιμοποιήθηκε ένας Arduino UNO για να τυπώνει το μήνυμα εξόδου του Delta Robot σε μία οθόνη χαρακτήρων LCD. Εδώ παρατίθεται ο κώδικας που τρέχει στον Arduino UNO και έχει σκοπό να τυπώσει τις συντεταγμένες και άλλες πληροφορίες στην οθόνη που υπάρχει στο LCD Shield. Ο κώδικας είναι απλός, διαβάζει τα δεδομένα που έρχονται από τη σειριακή και τα τυπώνει στην οθόνη υγρών κρυστάλλων. Αφού διαβάσει το «@», ελέγχει αν μετά ακολουθεί το «#» ή το «%» ή το «!» για να διαμορφώσει ανάλογα την οθόνη. Δηλαδή το «#» υποδηλώνει ότι οι αριθμοί είναι γωνίες, το «%» ότι το ρομπότ έχει σταματήσει «Pause Mode», ενώ το «!» ότι γίνεται η ρύθμιση του κατακόρυφου ορίου. Στην περίπτωση που δεν υπάρχει κάποιο από αυτά τότε έχουμε καρτεσιανές συντεταγμένες. Σε αυτήν την περίπτωση τυπώνει στην οθόνη και ένα χαμογελαστό πρόσωπο αν όλα είναι εντάξει, δηλαδή αν η validPosition είναι μηδέν. Αλλιώς το πρόσωπο είναι θλιμμένο και δίπλα του αναγράφεται η αιτία. Έτσι αν η μεταβλητή validPosition είναι αρνητική, τότε δίπλα του τυπώνεται ο αριθμός αυτός χωρίς το πρόσημό του. Αν είναι θετική και 1, τότε τυπώνεται ένα θαυμαστικό για να επιστήσει την προσοχή στον χρήστη ότι πιθανότατα υπάρχει πρόβλημα με τις επικοινωνίες. Αν είναι 2, τότε τυπώνεται ένα σύμβολο που υποδεικνύει ότι πήγε να παραβιαστεί το κατακόρυφο όριο. Στις παρακάτω εικόνες απεικονίζονται τα μηνύματα που τυπώνονται στην οθόνη για διάφορες περιπτώσεις:



Εικόνα 1.11. Η οθόνη εξόδου (α) αν δεν έχει και (β) αν έχει παραβιαστεί το κατακόρυφο όριο



Εικόνα 1.12. Η οθόνη εξόδου (α) όταν δείχνει τις γωνίες, (β) κατά την παύση του ρομπότ

Εικόνα 1.13. Η οθόνη εξόδου (α) κατά τη ρύθμιση του κατακόρυφου ορίου, (β) κατά την εκκίνηση

Παρακάτω ακολουθεί και ο κώδικας που τρέχει στον Arduino UNO:

```
#include<LiquidCrystal.h>

LiquidCrystal lcd(7,6,5,4,3,2);


    byte smiley[8] =
    {
      B00000,
      B10001,
      B00000,
      B00000,
      B10001,
      B01110,
      B00000,
      B00000
    };


    byte bad[8] =
    {
      B00000,
      B10001,
      B00000,
      B00000,
      B01110,
      B10001,
      B00000,
      B00000
    };


    byte bump[8] =
    {
      B00000,
      B00100,
      B00100,
      B00100,
      B00100,
      B01110,
      B11111,
      B00000
    };
```

```
float x=0.0,y=0.0,z=0.0,limit=-34.77;
float oldx=0.0, oldy=0.0, oldz=0.0, oldlimit=0.0;
int angle1=0, angle2=0, angle3=0;
int old1=0, old2=0, old3=0;
int prevLayout=-1;
int layout=-1;
int validPosition=0;
int nextChar=0;


void setup()
   {
    Serial.begin(9600);
    pinMode(13, OUTPUT);
    digitalWrite(13,HIGH);

    lcd.createChar(1,smiley);
    lcd.createChar(2,bad);
    lcd.createChar(3,bump);

    lcd.begin(16,2);
    lcd.clear();
    lcd.print("Waiting For Data");
    lcd.setCursor(0,1);
    lcd.print("Baud Rate= 9600");
   }


void loop()
  {

    if(Serial.available()>=4)
     {
       Serial.find("@");
       nextChar=Serial.peek();
       if (nextChar=='#') layout=2;
       else if (nextChar=='%') layout=1;
       else if (nextChar=='!') layout=3;
       else layout=0;

       if (layout==0)
       {
        x=Serial.parseFloat();
```

```
 y=Serial.parseFloat();
 z=Serial.parseFloat();
 validPosition=Serial.parseInt();

 if (x!=oldx||y!=oldy||z!=oldz||prevLayout!=0)
    {
       lcd.clear();
       lcd.print("X= ");
       lcd.print(x);
       lcd.setCursor(11,0);
       lcd.write(validPosition==0?1:2);
       if (validPosition==1) lcd.write('!');
       if (validPosition<0) lcd.print(-1*validPosition);
       if (validPosition==2) lcd.write(3);
       lcd.setCursor(14,0);
       lcd.print("Z= ");
       lcd.setCursor(0,1);
       lcd.print("Y= ");
       lcd.print(y);
       lcd.setCursor(10,1);
       lcd.print(z);

       oldx=x;
       oldz=z;
       oldy=y;
       prevLayout=0;
    }

}
else if (layout==1)
{
 if (prevLayout!=1)
 {
   lcd.clear();
   lcd.print(" Delta Robot is");
   lcd.setCursor(5,1);
   lcd.print("PAUSED");
   prevLayout=1;
 }
}
else if (layout==2)
{
 Serial.find('=');
 angle1=Serial.parseInt();
 Serial.find('=');
```

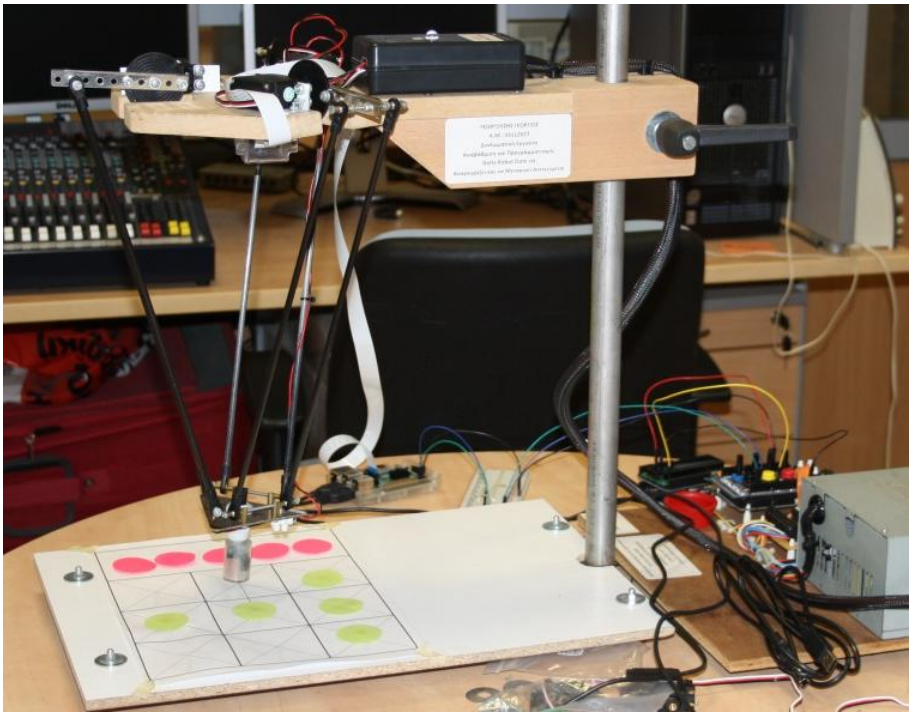
```
 angle2=Serial.parseInt();
 Serial.find('=');
 angle3=Serial.parseInt();

 if (angle1!=old1||angle2!=old2||angle3!=old3||prevLayout!=2)
   {

     lcd.clear();

     lcd.setCursor(2,0);
     lcd.write(127);
     lcd.write(230);
     lcd.write('1');
     lcd.setCursor(6,0);
     lcd.write(127);
     lcd.write(230);
     lcd.write('2');
     lcd.setCursor(10,0);
     lcd.write(127);
     lcd.write(230);
     lcd.write('3');

     lcd.setCursor(2,1);
     lcd.print(angle1);
     lcd.setCursor(6,1);
     lcd.print(angle2);
     lcd.setCursor(10,1);
     lcd.print(angle3);

     old1=angle1;
     old2=angle2;
     old3=angle3;
     prevLayout=2;
   }

}
else
{
   limit=Serial.parseFloat();

   if (limit!=oldlimit||prevLayout!=3)
   {

     lcd.clear();
```

```
            lcd.print(" Lower Limit =");

            lcd.setCursor(5,1);
            lcd.print(limit);

            oldlimit=limit;
            prevLayout=3;
          }
      }

      Serial.find("$");
  }         // Endif (Serial.available()>=4)
}
```

Κεφάλαιο 2.

Το Delta Robot

Παίζει Τρίλιζα

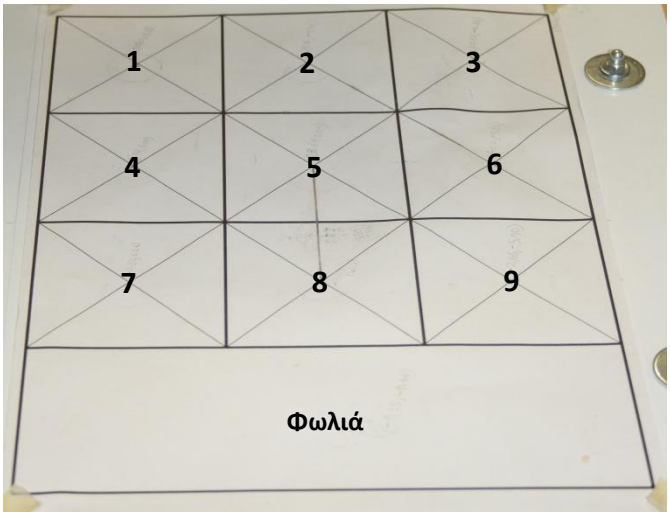


Κεφάλαιο 2. Το Delta Robot Παίζει Τρίλιζα

Ο κύριος σκοπός αυτής της διπλωματικής εργασίας ήταν να μπορεί το Delta Robot να παίζει τρίλιζα με οπτική αναγνώριση των πιονιών με τη χρήση κάμερας, χωρίς να δίνει ο χρήστης δεδομένα από το πληκτρολόγιο ή άλλη συσκευή. Επειδή το Arduino Mega δεν είχε τη δυνατότητα να συνδεθεί με κάμερα ούτε να εκτελέσει απαιτητικούς υπολογισμούς σε μνήμη και επεξεργαστική ισχύ, χρησιμοποιήθηκε ένας υπολογιστής Raspberry Pi. Αυτός μπορεί να εκτελέσει αυτή τη δουλειά και να στείλει μέσω της σειριακής του θύρας τις συντεταγμένες και την κατάσταση του ηλεκτρομαγνήτη στον Arduino Mega που ελέγχει το Delta Robot. Σε αυτό το κεφάλαιο παρουσιάζεται το Raspberry Pi και ο κώδικας που γράφτηκε. Οι παράγραφοι που ακολουθούν, μετά από μία σύντομη περιγραφή του hardware και του software, επικεντρώνονται στα χαρακτηριστικά εκείνα που αφορούν τη συγκεκριμένη εργασία. Περισσότερες πληροφορίες για το Raspberry Pi και τα παρελκόμενά του μπορεί κανείς να αντλήσει από τον επίσημο ιστότοπο, www.raspberrypi.org, καθώς από τους συνδέσμους που υπάρχουν σε αυτόν.

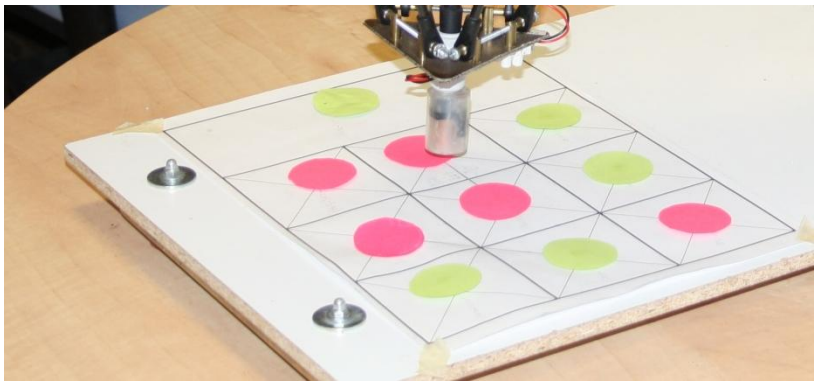
2.1 Το παιχνίδι της τρίλιζας

Η τρίλιζα είναι ένα παιχνίδι που παίζεται με δύο παίκτες. Ο χώρος που διεξάγεται χωρίζεται σε 9 τετράγωνα τοποθετημένα σε 3 τριάδες οριζόντια και κάθετα. Στην εικόνα 2.1 απεικονίζεται ο χώρος εργασίας του Delta Robot όπως διαμορφώθηκε για το παιχνίδι της τρίλιζας. Οι διαστάσεις του είναι 20x20cm, ενώ αν συμπεριλάβουμε και τη φωλιά είναι 26.7X26cm. Το κάθε τετράγωνο έχει διαστάσεις 6.7X6.7cm και η φωλιά 6.7X20cm.



Εικόνα 2.1. Ο πάγκος εργασίας του Delta Robot χωρισμένος σε 9 αριθμημένα τετράγωνα

Όταν η τρίλιζα παίζεται στο χαρτί ο κάθε παίκτης ζωγραφίζει ένα Χ ή ένα Ο στο τετράγωνο που επιθυμεί εφ' όσον είναι κενό. Εδώ για το παιχνίδι αυτό χρησιμοποιήθηκαν πιόνια. Έτσι το Delta Robot έχει τα κόκκινα πιόνια και ο παίκτης τα πράσινα. Η φωλιά προορίζεται για την τοποθέτηση των πιονιών του ρομπότ πριν την εκκίνηση του παιχνιδιού, από όπου το ρομπότ παίρνει τα πιόνια του και τα τοποθετεί στον πάγκο κατά τη διάρκεια του παιχνιδιού. Τα πιόνια, εικόνα 2.2, δημιουργήθηκαν από χρωματιστό χαρτί στο οποίο είχαν κολληθεί οι μεταλλικές κεφαλές των πινεζών που είχαν χρησιμοποιηθεί κατά την προηγούμενη διπλωματική εργασία. Έτσι το Delta Robot μπορούσε να αναγνωρίζει οπτικά σε ποιο τετράγωνο βρίσκονται τα πιόνια του αντιπάλου, αλλά και τα δικά του.



Εικόνα 2.2. Τα πιόνια του παιχνιδιού

Σκοπός του παιγνιδιού είναι να τοποθετηθούν 3 πιόνια ενός παίκτη σε μία ευθεία γραμμή είτε οριζόντια είτε κάθετα είτε διαγώνια. Όποιος παίκτης το καταφέρει πρώτος κερδίζει το παιχνίδι. Αν δεν το καταφέρει κανένας και δεν υπάρχει άλλο κενό τετράγωνο, τότε το παιχνίδι είναι ισόπαλο. Πχ. στην εικόνα 2.2 έχει κερδίσει το ρομπότ, διαγώνια. Το πράσινο πιόνι στη φωλιά δεν υπήρχε κατά τη φάση του παιχνιδιού, καθότι εκεί τοποθετούνται τα κόκκινα, αλλά τοποθετήθηκε μετά για τη φωτογράφιση.

Το Delta Robot είναι σε θέση να αναγνωρίζει τι κινήσεις έχουν γίνει, να αντιλαμβάνεται αν πρόκειται για έγκυρες ή όχι και να δρα κατάλληλα. Αν πρόκειται για μη έγκυρη κίνηση, ζαβολιά, επαναφέρει τα πιόνια στις προηγούμενες έγκυρες θέσεις τους, ενώ αν πρόκειται για έγκυρη κίνηση αποφασίζει για την επόμενη κίνησή του βάσει των κανόνων του παιγνιδιού. Τέλος αντιλαμβάνεται αν έχει κερδίσει, αν έχει χάσει ή αν το παιγνίδι έληξε ισόπαλο.

Εδώ αποφασίστηκε ο αλγόριθμος βάσει του οποίου θα αποφασίζει την επόμενη κίνησή του να είναι τέτοιος ώστε ο παίκτης ούτε να κερδίζει δύσκολα ούτε όμως και πολύ εύκολα έτσι ώστε να μην χάνει το ενδιαφέρον του. Αν είχαμε ένα ρομπότ που συνέχεια κερδίζει ή συνέχεια χάνει, τότε ο παίκτης θα βαριόταν γρήγορα και θα έχανε το ενδιαφέρον του. Ομοίως, αν σε περίπτωση ζαβολιάς του

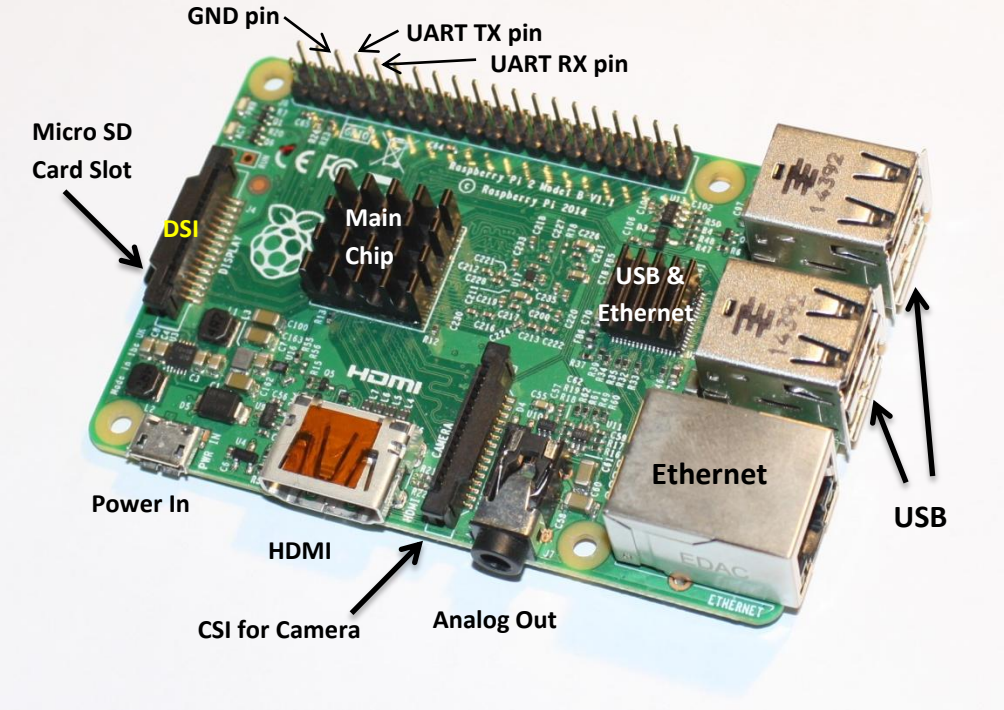
παίκτη, το ρομπότ αποφάσιζε να διακόψει την παρτίδα, τότε ο παίκτης θα έχανε το ενδιαφέρον του μετά από μερικές παρτίδες και θα σταματούσε. Αντιθέτως τώρα μπορεί να έχει περισσότερο ενδιαφέρον να εκτελεί ο παίκτης εσκεμμένα κάποιες εσφαλμένες κινήσεις προκειμένου να παρατηρήσει την αντίδραση του ρομπότ.

2.2 Το Raspberry Pi

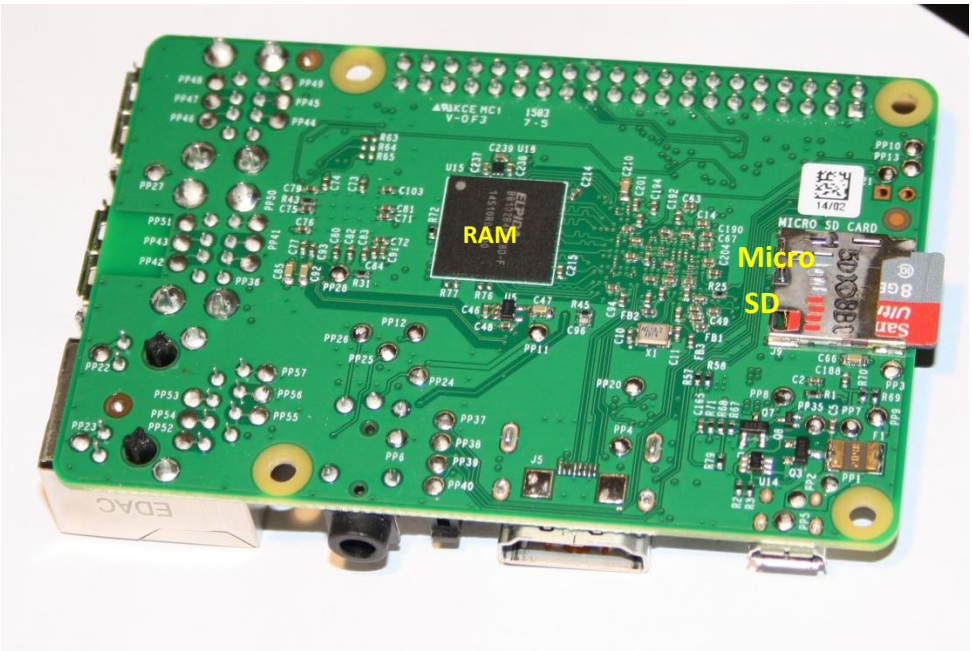
Το Raspberry Pi είναι ένας ηλεκτρονικός υπολογιστής μεγέθους πιστωτικής κάρτας. Το μοντέλο που χρησιμοποιήθηκε για την εργασία αυτή είναι το Raspberry Pi 2 Model B. Αυτό έχει τετραπύρηνο 32-bit επεξεργαστή ARM Cortex-A7 χρονισμένο στα 900MHz, μνήμη RAM 1GB, ενώ έχει και επεξεργαστή γραφικών VideoCore IV που διαμοιράζεται την κύρια μνήμη με τον κεντρικό επεξεργαστή. Όλα τα παραπάνω εκτός από τη μνήμη βρίσκονται σε ένα chip, SoC (System on Chip), κατασκευασμένο από τη Broadcom και είναι τοποθετημένο πάνω στην πλακέτα. Η μνήμη βρίσκεται σε ξεχωριστό chip τοποθετημένο στην κάτω όψη της πλακέτας. Εκεί υπάρχει και θύρα για κάρτα μνήμης Micro SD.

Πάνω στην πλακέτα υπάρχουν επίσης 4 θύρες USB, μία θύρα Ethernet, μία θύρα HDMI με ανάλυση εικόνας έως Full High Definition, καθώς και ένας θηλυκός ακροδέκτης για βύσμα τύπου καρφί 3.5mm για αναλογική έξοδο στερεοφωνικού ήχου και εικόνας. Για τον έλεγχο των διαύλων USB και Ethernet υπάρχει ξεχωριστό chip τοποθετημένο στην πλακέτα. Επιπλέον υπάρχουν κάποιες θύρες Camera Serial Interface (CSI) και Display Serial Interface (DSI) για να συνδεθούν η κάμερα και η οθόνη αφής TFT. Αυτές οι δύο θύρες προορίζονται για να συνδεθούν τα επίσημα εξαρτήματα για Raspberry Pi που έχουν συγκεκριμένες προδιαγραφές. Τέλος υπάρχουν ακροδέκτες GPIO που αναλύονται σε επόμενη παράγραφο.

Η ηλεκτρική τροφοδοσία γίνεται από εξωτερικό τροφοδοτικό ονομαστικής τάσης 5.1V και ρεύματος 2.5A μέσω θύρας micro USB η οποία προορίζεται μόνο για την τροφοδοσία με ρεύμα και όχι για δεδομένα. Στις εικόνες 2.3 και 2.4 απεικονίζονται οι δύο όψεις της πλακέτας αυτής. Στην υποδοχή για κάρτα Micro SD έχει τοποθετηθεί μία κάρτα SanDisk χωρητικότητας 8GB. Επίσης πάνω στο κυρίως chip και στο chip ελέγχου των θυρών USB και Ethernet έχουν τοποθετηθεί ψύκτρες για αποτελεσματικότερη ψύξη.



Εικόνα 2.3. Η πλακέτα Raspberry Pi



Εικόνα 2.4. Η πίσω όψη της πλακέτας Raspberry Pi

Το λειτουργικό του σύστημα είναι το Raspbian Jessie που είναι βασισμένο στην Debian Jessie διανομή του Linux κατάλληλα προσαρμοσμένη στο συγκεκριμένο μηχάνημα. Το περιβάλλον λειτουργίας είναι παραθυρικό αλλά και κονσόλας. Το λειτουργικό σύστημα εγκαθίσταται στη κάρτα Micro SD και εκτελείται από εκεί. Έχει όλα τα χαρακτηριστικά, τις βιβλιοθήκες και τις εφαρμογές του Debian Linux που είναι απαραίτητα για την υλοποίηση της συγκεκριμένης διπλωματικής εργασίας. Στην κάρτα αποθηκεύονται και όλα τα εγκατεστημένα προγράμματα και τα δεδομένα του χρήστη. Μερικά από αυτά που περιλαμβάνονται στο αρχικό πακέτο είναι διερμηνευτής για τη γλώσσα Python, μεταγλωττιστής για την γλώσσα C/C++, δηλαδή οι gcc και g++, κ.ά. Επίσης είναι διαθέσιμα από την επίσημη ιστοσελίδα αρκετά προγράμματα για κατέβασμα και εγκατάσταση με απλό τρόπο είτε μέσω κατάλληλης παραθυρικής εφαρμογής που υπάρχει εγκατεστημένη είτε με τη χρήση εντολών κονσόλας (sudo apt-get install … ή remove … για απεγκατάσταση). Με αυτό τον τρόπο εγκαταστάθηκε η βιβλιοθήκη OpenCV που περιγράφεται σε επόμενη παράγραφο, καθώς και το περιβάλλον ανάπτυξης Geanny.

2.3 Οι είσοδοι-έξοδοι γενικού σκοπού, GPIO pins

Πάνω στην πλακέτα του Raspberry Pi υπάρχουν 40 είσοδοι-έξοδοι, pins, General Purpose Input Output pins (GPIO), που λειτουργούν παρόμοια με αυτές του Arduino, με τρεις κύριες διαφορές. Πρώτον, είναι αρσενικά βύσματα, ενώ του Arduino θηλυκά. Δεύτερον, το λογικό 1, HIGH, αντιστοιχεί σε 3.3V και όχι σε 5V όπως συμβαίνει στον Arduino Mega. Συνεπώς για να επικοινωνήσει μαζί του υπήρχε κατάλληλο κύκλωμα σε Breadboard που έκανε τη μετατροπή του επιπέδου τάσης. Τρίτον, αντέχουν πολύ μικρότερα ρεύματα, της τάξης των 3 με 5mA, σε αντίθεση με του Arduino Mega που το όριο είναι 20mA. Από αυτά τα 40 pins τα 14 είναι για να τροφοδοτούν εξωτερικές συσκευές με ηλεκτρική ισχύ, δηλαδή 5V, 3.3V και γείωση. Αυτά αντέχουν πολύ μεγαλύτερα ρεύματα από τα υπόλοιπα 26 που υλοποιούν ψηφιακές εισόδους-εξόδους.

Δύο από αυτά τα 26 pins, τα 8 και 10 μπορούν να συνδεθούν εσωτερικά με το κύκλωμα UART προκειμένου να υλοποιηθεί σειριακή επικοινωνία με άλλη συσκευή που έχει τέτοια δυνατότητα. Κάποια άλλα μπορούν να χρησιμοποιηθούν για σειριακή επικοινωνία SPI, άλλα για TWI κλπ. Στα pins που υλοποιούν εισόδους-εξόδους μπορεί να καθοριστεί το επίπεδο τάσης από το λειτουργικό σύστημα με κατάλληλες εντολές. Όμως υπάρχει ενσωματωμένη μία εύχρηστη βιβλιοθήκη για γλώσσα C/C++, η wiringPi, η οποία μπορεί να ελέγξει αυτά τα pins με απλές εντολές όμοιες με αυτές του Arduino, δηλαδή digitalRead(), digitalWrite() κλπ. Επιπλέον με τη wiringSerial ο χρήστης μπορεί να υλοποιήσει εύκολα σειριακή μετάδοση και λήψη δεδομένων μέσω της UART με απλές εντολές. Στη συγκεκριμένη εργασία αυτή η δυνατότητα χρησιμοποιήθηκε για την επικοινωνία με τον Arduino Mega του Delta Robot και τη μετάδοση των συντεταγμένων.

2.4 Ο τρόπος επικοινωνίας του Raspberry Pi με το Delta Robot

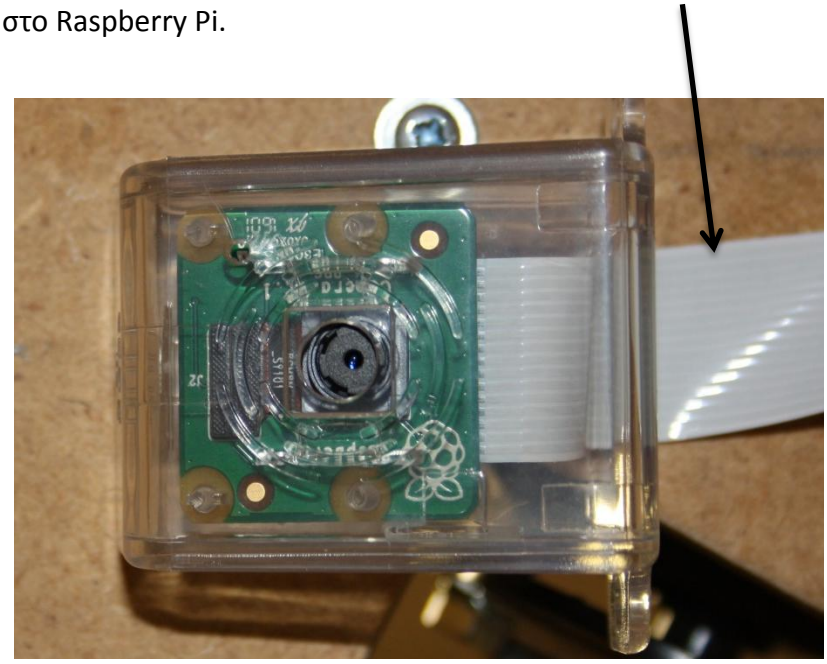
Η επικοινωνία μέσω UART γίνεται με κατάλληλο κύκλωμα που είναι ανεξάρτητο από τον κυρίως επεξεργαστή με τρόπο παρόμοιο με αυτόν του Arduino Mega που έχει αναλυθεί στο προηγούμενο κεφάλαιο. Η διαφορά είναι ότι η ενδιάμεση μνήμη είναι πολύ μεγαλύτερη και ότι παρεμβαίνει το λειτουργικό σύστημα στην όλη διαδικασία. Στην ουσία το λειτουργικό δημιουργεί ένα εικονικό αρχείο στη θέση «/dev/» συνήθως το «ttyAMA0», δηλαδή το «/dev/ttyAMA0», και επιστρέφει έναν file descriptor σαν να ήταν κανονικό αρχείο. Το ίδιο κάνει και με άλλες συσκευές πχ. με την κάμερα. Αυτό το αρχείο μπορεί να προσπελαστεί με τις κλήσεις συστήματος που προορίζονται για εγγραφή και ανάγνωση από αρχεία («write()», «read()», κλπ) και έτσι να αποσταλεί ή να ληφθεί το μήνυμα. Ωστόσο, η βιβλιοθήκη «wiringSerial» απλοποιεί πολύ την διαδικασία.

Τα pins για τη σειριακή επικοινωνία είναι στη θέση 8 για την αποστολή (TX) και στη θέση 10 για τη λήψη (RX) και αφορά την φυσική αρίθμηση των pins πάνω στην πλακέτα. Όμως επειδή κάποια pins είναι για ηλεκτρική τροφοδοσία, δεν είναι όλα τα pins είσοδοι-έξοδοι, οπότε δεν είναι όλα συνδεδεμένα με το chip της Broadcom. Συνεπώς η αρίθμηση αυτών των δύο pins όσον αφορά το πώς τα «βλέπει» το chip είναι η θέση 14 για την αποστολή (TX) και η θέση 15 για τη λήψη (RX) δεδομένων. Έτσι για την επικοινωνία με το Delta Robot, το pin 8, TX (ή BCM14), συνδέεται με το pin 19, RX, του Arduino Mega και το pin 10, RX (ή BCM15), συνδέεται με το pin 18, TX, του Arduino Mega. Μεταξύ τους παρεμβάλλεται το κύκλωμα αλλαγής του επιπέδου τάσης. Τέλος συνδέεται κάποιο pin γείωσης του Raspberry Pi πχ. το pin 6 (GND) με το αντίστοιχο pin GND του Arduino προκειμένου να έχουν κοινή αναφορά.

Εδώ να αναφέρουμε ότι η UART και τα pins 8 και 10 αρχικά είναι δεσμευμένα από το λειτουργικό για να μπορεί να δέχεται εντολές και να στέλνει την έξοδο το τερματικό από εξωτερική σειριακή συσκευή στην περίπτωση που πρέπει το Raspberry Pi να λειτουργήσει χωρίς πληκτρολόγιο ή και χωρίς οθόνη. Στην περίπτωσή μας υπήρχε συνδεδεμένο πληκτρολόγιο, ποντίκι, μέσω USB, και οθόνη μέσω HDMI, ενώ ήταν διαθέσιμη και η επίσημη οθόνη αφής που συνδέεται στην θύρα DSI και αναγνωρίζεται εγγενώς από το λειτουργικό αντικαθιστώντας τα παραπάνω. Έτσι δεν ήταν απαραίτητη αυτή η δυνατότητα ελέγχου του τερματικού μέσω της UART. Αντιθέτως δημιουργούσε προβλήματα γιατί τη δέσμευε και δεν μπορούσε να χρησιμοποιηθεί από άλλη εφαρμογή. Συνεπώς έπρεπε να απενεργοποιηθεί για να μπορέσει να χρησιμοποιηθεί από την εφαρμογή που δημιουργήθηκε στα πλαίσια αυτής της εργασίας για να επικοινωνήσει με τον Arduino. Αυτό γίνεται με δύο τρόπους, είτε από το παραθυρικό περιβάλλον μέσω κατάλληλης εφαρμογής ρυθμίσεων που υπάρχει ενσωματωμένη είτε από το περιβάλλον κονσόλας μέσω της επίσης ενσωματωμένης εφαρμογής «raspi config» (εντολή: «sudo raspi-config»).

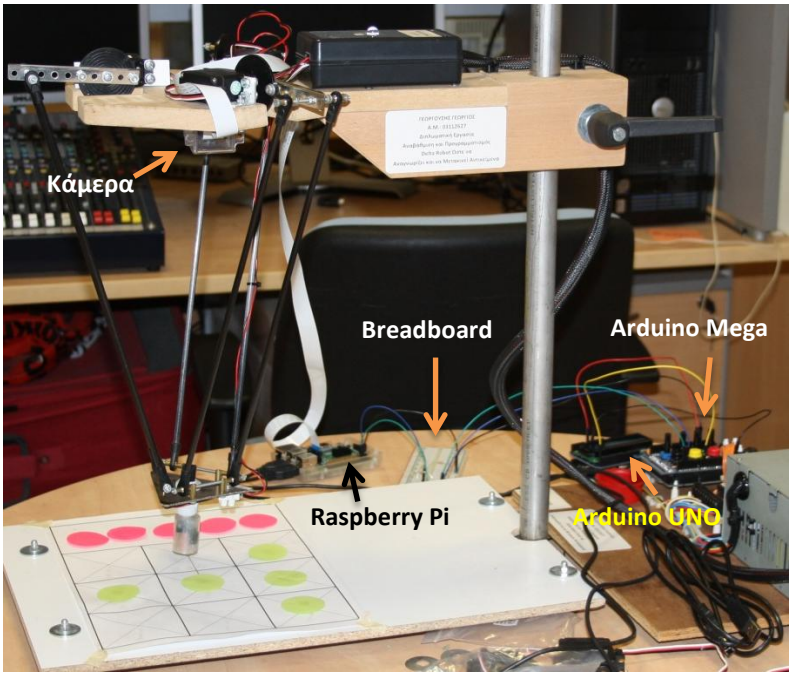
2.5 Η κάμερα

Για να αναγνωρίζει τα πιόνια και τις θέσεις τους οπτικά είχε τοποθετηθεί στην υποδοχή CSI η επίσημη κάμερα για Raspberry Pi. Αυτή αναγνωρίζεται αυτόματα από το λειτουργικό, γιατί περιέχει τους απαραίτητους drivers για την λειτουργία της. Έχει αισθητήρα της SONY 8Megapixels με ανάλυση 3280x2464 pixels. Για την εργασία αυτή χρησιμοποιήθηκε ανάλυση 640x480, για οικονομία μνήμης και χρόνου υπολογισμών, άλλα και γιατί κρίθηκε ότι ήταν αρκετή για την εφαρμογή που θέλαμε να υλοποιήσουμε. Στην εικόνα 2.5 απεικονίζεται η κάμερα τοποθετημένη μέσα στη θήκη της η οποία είναι κολλημένη στο Delta Robot και κοιτάζει προς τον πάγκο εργασίας του ρομπότ. Διακρίνεται επίσης η καλωδιοταινία που μεταφέρει τα δεδομένα στο Raspberry Pi.



Εικόνα 2.5. Η κάμερα μέσα στη θήκη της που είναι κολλημένη στο Delta Robot

Στην εικόνα 2.6 απεικονίζεται τη σύνολο της διάταξης συνδεδεμένο. Διακρίνεται η κάμερα που είναι τοποθετημένη πάνω από τον πάγκο εργασίας του ρομπότ και είναι συνδεδεμένη με το Raspberry Pi. Επίσης διακρίνεται ο Arduino Mega με το Shield του που επικοινωνεί με το Raspberry καθώς και με το ρομπότ. Δίπλα του διακρίνεται η οθόνη LCD τοποθετημένη πάνω στον Arduino UNO που είναι συνδεδεμένος με το ρομπότ για την έξοδο. Τέλος διακρίνονται τα πιόνια του ρομπότ, κόκκινα, τοποθετημένα στη φωλιά πριν την έναρξη του παιχνιδιού. Κατά τη διάρκεια του παιχνιδιού το ρομπότ τα παίρνει από εκεί ένα-ένα και τα τοποθετεί στα τετράγωνα. Τα πιόνια του παίκτη, πράσινα, κανονικά δεν πρέπει να βρίσκονται στα τετράγωνα, όμως τοποθετήθηκαν για τις ανάγκες τις φωτογράφησης.



Εικόνα 2.6. Το Raspberry Pi συνδεδεμένο με το Delta Robot και την κάμερα

Για να λειτουργήσει η κάμερα υπάρχουν κατάλληλες εντολές κονσόλας και βιβλιοθήκες κυρίως της Python. Από αυτές μπορεί να γίνει λήψη βίντεο ή φωτογραφίας και να αποθηκευτούν σε κάποιο αρχείο. Επίσης μέσα από το περιβάλλον της Python μπορεί να γραφτεί κώδικας που να υλοποιεί πιο σύνθετες εφαρμογές. Όμως η Python είναι διερμηνευόμενη γλώσσα και είναι πιο αργή στην εκτέλεση του κώδικα σε σχέση με την C/C++ που είναι μεταγλωττιζόμενη και μπορεί να εκτελεστεί ο κώδικας πολύ πιο γρήγορα. Έτσι αν λάβουμε υπ' όψιν μας τις μικρές δυνατότητες του Raspberry Pi και τις μεγάλες απαιτήσεις της οπτικής αναγνώρισης, τότε προκύπτει σημαντική διαφορά στην ταχύτητα εκτέλεσης. Αυτό παρατηρήθηκε και στην πράξη μετά από δοκιμές που έγιναν. Γι' αυτό αποφασίστηκε αυτή η εργασία να γίνει σε C++.

2.6 Η βιβλιοθήκη OpenCV

Για τις ανάγκες αυτής της εργασίας χρησιμοποιήθηκε η βιβλιοθήκη OpenCV για C++. Αυτή αρχικά δεν είναι εγκατεστημένη στο Raspberry αλλά μπορεί να κατέβει και να εγκατασταθεί σε αυτό με τη χρήση της εντολής:

```
sudo apt-get install libopencv-dev
```

Η OpenCV είναι εύχρηστη βιβλιοθήκη και μεταξύ άλλων μπορεί να αιχμαλωτίσει εικόνα από την κάμερα και να την αποθηκεύσει σε αντικείμενα της κλάσης «Mat» η οποία υπάρχει στη βιβλιοθήκη και είναι κατάλληλη για επεξεργασία εικόνας κυρίως και όχι μόνο. Κατόπιν με κατάλληλες συναρτήσεις και μεθόδους της κλάσης αντικειμένων «Mat» που περιέχει η βιβλιοθήκη μπορεί να

γίνει επεξεργασία εικόνας, να προσπελαστούν συγκεκριμένα pixels κοκ. Επίσης υπάρχει διαθέσιμη και βιβλιοθήκη OpenCV για Python.

Τέλος για να τρέξει το εκτελέσιμο αρχείο που κάνει χρήση της OpenCV και περιέχει εντολές πρόσβασης στην κάμερα πρέπει πρώτα να εκτελεστεί από το τερματικό η εντολή:

sudo modprobe bcm2835-v4l2

Αυτή ζητάει από το λειτουργικό να ενεργοποιήσει τον driver της κάμερας και αυτό δημιουργεί ένα εικονικό αρχείο στο φάκελο «/dev/» με όνομα συνήθως «video0». Αυτό το αρχείο προσπελαύνει ο κώδικας του εκτελέσιμου αρχείου προκειμένου να λάβει εικόνα από την κάμερα.

2.7 Το σκεπτικό του κώδικα της εφαρμογής

Ο πάγκος παιχνιδιού έχει χωριστεί σε 9 τετράγωνα αριθμημένα από το 1, πάνω αριστερά, μέχρι το 9, κάτω δεξιά. Στον κώδικα έχει δημιουργηθεί ένα αντικείμενο που έχει ονομαστεί «Boxes» και περιέχει έναν πίνακα μέλος του 10 στοιχείων, box[10], που έχει σκοπό να αποθηκεύει το περιεχόμενο αυτών των κουτιών. Έτσι αν στο πάνω αριστερά τετράγωνο, κουτί 1, υπάρχει πιόνι του ρομπότ, τότε αυτό (box[1]) έχει την τιμή 2, ενώ αν υπάρχει πιόνι του παίκτη, τότε έχει τιμή 1. Αν είναι άδειο έχει την τιμή 0. Στο «box[0]» αποθηκεύεται ο αριθμός των πιονιών του ρομπότ που υπάρχουν στη φωλιά. Υπενθυμίζεται ότι η φωλιά είναι το μεγάλο ορθογώνιο παραλληλόγραμμο κουτί μέσα στο οποίο αποθηκεύονται τα 5 πιόνια του ρομπότ και από εκεί παίρνει πιόνια κάθε φορά για να παίξει. Στη φωλιά αρχικά αποθηκεύονται 5 πιόνια του ρομπότ από τον παίκτη γιατί αυτός είναι και ο μέγιστος αριθμός που χρειάζεται. Αφού τα τετράγωνα είναι 9 και οι παίκτες παίζουν εναλλάξ, τότε αν γεμίσουν όλα τα κουτιά με πιόνια, αυτά θα είναι 5 από τον ένα παίκτη και 4 από τον άλλον. Συνεπώς είναι αδύνατον να χρειαστούν περισσότερα από 5 για τον κάθε παίκτη για να ολοκληρωθεί το παιχνίδι σύμφωνα με τους κανόνες του.

Αρχικά ο κώδικας εκτελεί την εντολή «sudo modprobe bcm2835-v4l2» για τους λόγους που αναλύθηκαν σε προηγούμενη παράγραφο. Εδώ υπάρχει μία παρατήρηση. Αν ο χρήστης εκτελέσει το εκτελέσιμο αρχείο με όρισμα «!», τότε δεν θα εκτελεστεί η παραπάνω εντολή. Δηλαδή αν υποθέσουμε ότι το όνομα του εκτελέσιμου αρχείο είναι «TicTacToe», τότε για να μην εκτελεστεί η παραπάνω εντολή ο χρήστης πρέπει να πληκτρολογήσει:

./TicTacToe !

Και αυτό γιατί δεν είναι απαραίτητο να εκτελείται κάθε φορά η εντολή «modprobe…» αρκεί να έχει εκτελεστεί μία φορά μετά την εκκίνηση του Raspberry και εξακολουθεί να είναι σε ισχύ. Από την άλλη δεν υπάρχει πρόβλημα αν την

εκτελέσει ο χρήστης πολλές φορές. Ο λόγος όμως ύπαρξης αυτής της δυνατότητας είναι γιατί μπορεί για κάποιο λόγο όταν εκτελείται αυτή η εντολή να κολλήσει και να μην μπορεί να συνεχιστεί το παιχνίδι. Έτσι αν ο χρήστης έχει εκτελέσει νωρίτερα την εντολή αυτή με επιτυχία, τότε του δίνεται η δυνατότητα να προσπαθήσει να παίξει χωρίς την εκτέλεση της εντολής αυτής. Για βοήθεια ο χρήστης μπορεί να τρέξει το εκτελέσιμο με όρισμα «?» ή «H» ή «h» ή ολόκληρη τη λέξη «Help» ή «help». Δηλαδή:

./TicTacToe ? ή ./TicTacToe Help κοκ.

Αφού ολοκληρωθούν όλα αυτά ανοίγει η κάμερα και ξεκινάει η σειριακή επικοινωνία με το Delta Robot. Δηλαδή περιμένει να λάβει ένα «@» και του στέλνει τις πρώτες συντεταγμένες. Για να συμβεί όμως αυτό πρέπει να έχει ξεκινήσει η εφαρμογή «TicTacToe» πριν από το Delta Robot, ειδάλλως το «@» που θα έχει στείλει το ρομπότ θα χαθεί και η εφαρμογή θα περιμένει. Αν περάσουν 5 δευτερόλεπτα και το ρομπότ δεν αποκριθεί, τότε τυπώνει σχετικό μήνυμα στην οθόνη για να πληροφορήσει το χρήστη. Ένας άλλος λόγος να χαθεί ο συγχρονισμός είναι να αποσυνδεθεί το καλώδιο επικοινωνίας. Σε όλες αυτές τις περιπτώσεις για να επανέλθει η κανονική λειτουργία αρκεί ο χρήστης να πατήσει το μαύρο κουμπί στο Arduino Shield, όπως έχει αναλυθεί στο προηγούμενο κεφάλαιο. Αν δεν υπήρχε η δυνατότητα αυτή θα έπρεπε κάθε φορά που εκτελείται η εφαρμογή της τρίλιζας να γίνεται επανεκκίνηση του Arduino Mega στο ρομπότ.

Όταν ολοκληρωθούν αυτοί οι έλεγχοι επιτυχώς, τότε αποφασίζει με τυχαίο τρόπο ποιος από τους δύο παίζει πρώτος, το ρομπότ ή ο παίκτης. Κατόπιν ξεκινάει το παιχνίδι. Το κυρίως μέρος της εφαρμογής περιλαμβάνεται σε δύο βρόγχους φωλιασμένους ο ένας μέσα στον άλλον. Στον εσωτερικό βρόγχο πραγματοποιείται η οπτική αναγνώριση των θέσεων των πιονιών με τη κλήση κατάλληλης συνάρτησης που έχει γραφτεί για αυτό το σκοπό. Κατόπιν γίνεται έλεγχος αν έχει πραγματοποιηθεί κάποια κίνηση από τον παίκτη. Για να γίνει αυτό έχει αποθηκευμένο σε ένα αντικείμενο, ή στιγμιότυπο, της κλάσης «Boxes» με το όνομα «validGameBoad» την τελευταία έγκυρη θέση των πιονιών, δηλαδή ποιο πιόνι είναι σε κάθε κουτί. Σε ένα άλλο στιγμιότυπο της «Boxes» με όνομα «gameBoad» έχουν αποθηκευτεί οι θέσεις των πιονιών που ανιχνεύτηκαν οπτικά. Έτσι συγκρίνει τα δύο αυτά αντικείμενα αν είναι ίδια ή αν υπάρχουν αλλαγές στα περιεχόμενά τους. Αυτό επαναλαμβάνεται μέχρι να ανιχνευθεί κάποια αλλαγή. Αυτή η επανάληψη δεν γίνεται συνέχεια αλλά κάθε ένα δευτερόλεπτο γιατί δεν έχει νόημα να σαρώνει συνεχώς τον πάγκο του παιγνιδιού, αφού ο παίκτης θέλει τουλάχιστον ένα δευτερόλεπτο για να μετακινήσει ένα πιόνι.

Όταν ανιχνεύσει αλλαγή ελέγχει αν αυτή είναι έγκυρη κίνηση, δηλαδή αν ο παίκτης έχει παίξει σύμφωνα με τους κανόνες του παιχνιδιού ή αν έχει «κλέψει». Αν έχει κλέψει, τότε επαναφέρει τα πιόνια στις προηγούμενες θέσεις τους και ζητάει

από τον παίκτη να ξαναπαίξει. Όλα αυτά πραγματοποιούνται στον εσωτερικό βρόγχο. Αν η κίνηση είναι έγκυρη, τότε βγαίνει από αυτόν και συνεχίζει την εκτέλεση του εξωτερικού βρόγχου όπου αποφασίζει την επόμενή του κίνηση και την εκτελεί, εκτός αν έχει νικήσει ο αντίπαλος ή αν δεν υπάρχουν κενές θέσεις. Κατόπιν επαναλαμβάνει από την αρχή τον εξωτερικό βρόγχο. Αν το παιχνίδι έχει τελειώσει, τότε δεν τον εκτελεί και τυπώνει μήνυμα για το ποιος κέρδισε ή για τον αν υπάρχει ισοπαλία, ενώ παράλληλα τυπώνει και σε ξεχωριστό παράθυρο τις τελικές θέσεις των πιονιών και πάνω από τα πιόνια του αντιπάλου έχει ένα πράσινο Χ, ενώ πάνω από τα δικά του ένα κόκκινο Ο.

2.8 Η οπτική αναγνώριση

Για την οπτική αναγνώριση υπάρχουν πολλοί τρόποι. Η βιβλιοθήκη OpenCV έχει κάποιες συναρτήσεις αναγνώρισης κάποιου συγκεκριμένου μοτίβου, αναγνώρισης γραμμών κλπ. Επιπλέον υπάρχει η συνάρτηση HoughCircles() που αναγνωρίζει κυκλικά αντικείμενα στην εικόνα. Όμως είναι αρκετά απαιτητική σε υπολογισμούς και επιλέχθηκε να μην χρησιμοποιηθεί εδώ. Αντιθέτως, γράφτηκε μία συνάρτηση, η «detectPawns», που πηγαίνει στο κέντρο του κάθε κουτιού και ψάχνει μέσα σε μία τετραγωνική περιοχή 20Χ20 pixels για κάποιο πιόνι χρώματος κόκκινου ή πράσινου, αναλόγως με ποιανού τα πιόνια ψάχνει, του ρομπότ ή του παίκτη. Στην ουσία ελέγχει κάθε ένα από τα 400 εικονοστοιχεία γύρω από το κέντρο κάθε κουτιού και αν βρει τουλάχιστον 300 από κάποιο χρώμα, τότε θεωρεί ότι εκεί υπάρχει το αντίστοιχο πιόνι. Πιο αναλυτικά αυτή η διαδικασία περιγράφεται παρακάτω.

Η οπτική αναγνώριση γίνεται ως εξής: Αρχικά δηλώνονται 3 αντικείμενα «Boxes» με τα ονόματα «temp1», «temp2» και «temp3» και αντικείμενα της κλάσης «Mat», με ονόματα «DeltaRobotPawns» και «playerPawns». Μετά, αποθηκεύει στη μνήμη ένα καρέ από την κάμερα. Ύστερα, μετατρέπει το αρχείο από έγχρωμο BGR, 3 καναλιών (Blue Green Red), σε έγχρωμο HSV, 3 καναλιών (Hue Saturation Value). Κατόπιν το HSV το μετατρέπει σε ασπρόμαυρο (B&W) ενός καναλιού. Αυτό δεν είναι αποχρώσεις του γκρι (Grayscale), αλλά είτε άσπρο είτε μαύρο. Δηλαδή σε κάθε εικονοστοιχείο αποθηκεύεται είτε το μαύρο, 0 (00000000), είτε το άσπρο, 255 (11111111).

Για να γίνει αυτό χρησιμοποιείται η συνάρτηση «inRange()» της βιβλιοθήκης OpenCV, η οποία μετατρέπει μία έγχρωμη εικόνα σε B&W βάση κάποιων παραμέτρων που εισάγονται σε αυτήν σαν ορίσματα. Αυτοί οι παράμετροι λένε στην συνάρτηση εντός ποιων τιμών των H, S και V καναλιών αντίστοιχα να θεωρείται το εικονοστοιχείο άσπρο. Έτσι από μία έγχρωμη εικόνα μπορούμε να κάνουμε «αόρατα», μαύρα, όλα τα αντικείμενα που δεν έχουν κάποιο επιθυμητό χρώμα, είναι εκτός των ορίων αυτών, ενώ να κάνουμε ορατά, άσπρα, όλα τα αντικείμενα που έχουν το επιθυμητό χρώμα. Έτσι μπορούμε να απομονώσουμε σε

έναν «πίνακα», αντικείμενο κλάσης «Mat», τα πιόνια του ρομπότ, «DeltaRobotPawns», και σε ένα άλλο αυτά του παίκτη, «playerPawns». Αφού γίνουν αυτά εφαρμόζεται ένα φίλτρο Gaussian Blur σε κάθε ένα για να μειωθεί ο θόρυβος.

Κατόπιν πηγαίνει στο αντικείμενο «Mat» που αφορά τα πιόνια του παίκτη και προσπελαύνει τα 400 εικονοστοιχεία που υπάρχουν σε ένα τετράγωνο διαστάσεων 20 επί 20 pixels γύρω από το κέντρο του κάθε ενός κουτιού του πάγκου παιγνιδιού και ελέγχει την τιμή τους, δηλαδή αν είναι άσπρα ή μαύρα. Αν είναι άσπρα σημαίνει ότι εκεί υπάρχει πιόνι, ενώ αν είναι μαύρα όχι. Επειδή μπορεί να υπάρχει θόρυβος ή λόγω φωτισμού να υπάρχουν κάποια pixels μαύρα εκεί που θα έπρεπε να είναι άσπρα και το αντίθετο, αποφασίστηκε ότι αν βρει περισσότερα από 300 άσπρα εικονοστοιχεία, από σύνολο 400, να θεωρήσει ότι υπάρχει πιόνι. Αν βρει λιγότερα ή κανένα, τότε θεωρεί ότι δεν υπάρχει πιόνι στο συγκεκριμένο κουτί. Αυτό το επαναλαμβάνει και για τα 9 κουτιά. Κατόπιν κάνει το ίδιο και για τα πιόνια του ρομπότ. Η φωλιά δεν ελέγχεται γιατί εμποδίζεται από τους βραχίονες του ρομπότ.

Για παράδειγμα έστω το αντικείμενο της κλάσης «Mat» που είναι αποθηκευμένα σε B&W τα pixels του ρομπότ, που ονομάζεται «DeltaRobotPawns», αν από τα 400 εικονοστοιχεία που είναι γύρω από το κέντρο του πρώτου κουτιού βρεθούν 300 τουλάχιστον λευκά, τότε θεωρεί ότι εκεί υπάρχει πιόνι του ρομπότ. Άρα στο αντικείμενο της κλάσης «Boxes» με το όνομα «temp1» θα αλλάξει την τιμή του στοιχείου box[1] από 0 σε 2 (temp1.box[1]=2). Αν ήταν στο «Mat» «playerPawns», τότε το box[1] θα γινόταν 1 (temp1.box[1]=1) υποδηλώνοντας ότι εκεί υπάρχει πιόνι του παίκτη. Ομοίως και για τα υπόλοιπα 8 κουτιά.

Επειδή μπορεί για κάποιο λόγο να υπάρξουν σφάλματα στην διαδικασία αυτή, για σιγουριά αποφασίστηκε να θεωρούνται ακριβή τα αποτελέσματα όταν επιβεβαιωθούν άλλες δύο φορές. Έτσι αυτή η διαδικασία επαναλαμβάνεται από την αρχή όπου αρχικά τα στοιχεία του «temp2» αντιγράφονται στο «temp3» και του «temp1» στο «temp2», ενώ το «temp1» μηδενίζεται και εκεί καταχωρούνται τα καινούρια δεδομένα που θα προκύψουν από την επανάληψη της διαδικασίας. Έτσι όταν βρεθούν 3 συνεχόμενες επαναλήψεις με παρόμοιο αποτέλεσμα, δηλαδή όταν τα «temp1», «temp2» και «temp3» έχουν ίδια δεδομένα, τότε θεωρείται ότι έχουμε επιτυχή αναγνώριση και επιστρέφει η συνάρτηση καταχωρώντας στο αντικείμενο «Board» το αποτέλεσμα. Αυτό στην ουσία τα καταχωρεί στο αντικείμενο «gameBoard» που είναι δηλωμένο στη main() γιατί η συνάρτηση «detectPawns» έχει στο όρισμα αναφορά, κλήση με αναφορά. Αν ο παίκτης κάνοντας την κίνησή του καλύψει τα πιόνια από την κάμερα, αυτό θα έχει ως αποτέλεσμα να θεωρηθεί εσφαλμένα ότι τα πιόνια δεν είναι στη θέση τους. Για αυτό κάθε επανάληψη πραγματοποιείται ανά δευτερόλεπτο και όχι συνεχώς έτσι ώστε να έχει μετακινηθεί το χέρι του και να υπάρξουν τρία διαφορετικά αποτελέσματα μεταξύ των διαδοχικών επαναλήψεων και όχι όμοια. Αυτό θα έχει ως αποτέλεσμα να συνεχιστεί

η διαδικασία μέχρις ότου να απομακρυνθεί τελείως το χέρι και βρεθούν τρία όμοια. Στο τέλος κάθε επανάληψης αποτυπώνεται στο παράθυρο η εικόνα από την κάμερα μαζί με τα σημεία Χ ή Ο πάνω από τα αντίστοιχα πιόνια.

2.9 Οι πίνακες θέσης των κουτιών

Για να την αποθήκευση της θέσης του κάθε κουτιού ως προς το σύστημα συντεταγμένων της κάμερας, υπάρχουν δύο πίνακες με τις θέσεις των κεντρικών εικονοστοιχείων του κάθε κουτιού, ο «XcentralPix[9]» και ο «YcentralPix[9]» αντίστοιχα. Παραδείγματος χάρη στο XcentralPix[3] και YcentralPix[3] είναι αποθηκευμένες οι συντεταγμένες (405 και 320) του εικονοστοιχείου που βρίσκεται στο κέντρο του 4^{ου} κουτιού, δηλαδή δεύτερη σειρά πρώτη στήλη. Αυτά δεν είναι σε εκατοστά αλλά σε pixels όπως τα βλέπει η κάμερα και τα αποθηκεύσει στο αντικείμενο «Mat».

Ομοίως, ως προς το σύστημα συντεταγμένων του Delta Robot υπάρχουν άλλοι τρεις πινακες «Xcenters[14]», «Ycenters[14]» και «ZcloseToPawns[14]» όπου αποθηκεύονται οι Χ, Υ συντεταγμένες του κάθε κουτιού καθώς και η Z συντεταγμένη χαμηλά κοντά στα πιόνια. Αντιθέτως για ψηλά, δηλαδή όταν είναι μακριά από τα πιόνια δεν χρειάζεται κάποιος πίνακας, για αυτό υπάρχει η σταθερά «up». Αυτοί οι πίνακες έχουν 14 στοιχεία γιατί αποθηκεύουν και τις συντεταγμένες των 5 θέσεων της φωλιάς. Πχ. το σημείο Xcenters[13], Ycenters[13], ZcloseToPawns[13], δηλαδή το (843,-669,-3450), είναι η θέση που πρέπει να βρεθεί η κεφαλή του Delta Robot για να πιάσει το πιόνι που είναι στη δεξιά άκρη της φωλιάς. Οι τιμές τους είναι σε εκατοστά πολλαπλασιασμένα επί 100 και αφορούν το σύστημα συντεταγμένων του ρομπότ. Τέλος υπάρχουν οι μεταβλητές «awayX» και «awayY» που αφορούν τη θέση της κεφαλής για να μην κρύβει τα 9 κουτιά από την κάμερα.

Συνεπώς οι δύο πρώτοι πίνακες προσπελαύνονται κατά την ανίχνευση και αναγνώριση των πιονιών και της θέσης που έχει το καθένα. Αντιθέτως οι τρεις τελευταίοι πίνακες προσπελαύνονται όταν πρέπει να σταλεί στο Delta Robot η θέση στην οποία πρέπει να μετακινηθεί η κεφαλή του με τον ηλεκτρομαγνήτη. Έτσι σε κάθε κουτί του παιγνιδιού αντιστοιχούν δύο ομάδες συντεταγμένων, αυτές που αφορούν την κάμερα και αυτές που αφορούν το ρομπότ.

2.10 Η πρόσβαση στους πίνακες

Επειδή ο κώδικας είναι πολύπλοκος υπήρχε το ενδεχόμενο λόγω τυχόν προγραμματιστικού λάθους να προσπελαστεί στοιχείο πίνακα που δεν είναι έγκυρο, πχ ZcloseToPawns[14] ή gameBoard.box[10]. Έτσι αποφασίστηκε οι πίνακες με τις συντεταγμένες να μην είναι καθολικής εμβέλειας (global) αλλά τοπικής (local) μέσα

σε κάποια συνάρτηση και η πρόσβαση να γίνεται μέσω της συνάρτησης αυτής. Έτσι γίνεται έλεγχος αν το ζητούμενο στοιχείο είναι μέσα στα όρια του κάθε πίνακα ή όχι. Αν είναι έξω από αυτά, τυπώνεται μήνυμα σφάλματος στην οθόνη και σταματάει η εκτέλεση του κώδικα.

Αυτό βοήθησε πολύ στην αποσφαλμάτωση, γιατί σε αντίθετη περίπτωση θα υπήρχε ο κίνδυνος η μη έγκυρη θέση πίνακα να αντιστοιχεί σε κάποια άλλη μεταβλητή, με αποτέλεσμα το λειτουργικό σύστημα να μην διακόψει την εκτέλεση του κώδικα. Αυτό θα μπορούσε να έχει σοβαρές συνέπειες κυρίως όσον αφορά την αποσφαλμάτωση καθότι το πρόβλημα μπορεί να έβγαινε στην επιφάνεια σε μεταγενέστερο χρόνο και όχι τη στιγμή πρόσβασης στους πίνακες, δυσκολεύοντας έτσι την εύρεση του λάθους. Αντιθέτως με τη παρούσα υλοποίηση το πρόβλημα γίνεται αντιληπτό μόλις επιχειρηθεί η πρόσβαση στην μη έγκυρη θέση κάνοντας πιο εύκολο να βρεθεί το σημείο του κώδικα που βρίσκεται το λάθος.

Συνεπώς δημιουργήθηκαν οι συναρτήσεις «XcentralPixel()», «YcentralPixel()», «X()», «Y()» και «down()» για πρόσβασή στους πίνακες «XcentralPix», «YcentralPix», «Xcenters», «Ycenters» και «ZcloseToPawns» αντίστοιχα. Αυτές δέχονται σαν όρισμα τον αριθμό του κουτιού, από 1 έως και 9, ενώ οι τρεις τελευταίες δέχονται επιπλέον και το 0 και αρνητικό αριθμό από -5 έως και -1. Οι αρνητικοί αριθμοί είναι για τις θέσεις της φωλιάς από αριστερά προς τα δεξιά, ενώ το 0 αφορά τις συντεταγμένες για την θέση της κεφαλής μακριά από την κάμερα. Τέλος όλες επιστρέφουν την τιμή του αντίστοιχου στοιχείου του πίνακα.

Όσον αφορά τον πίνακα «box[]» που είναι μέλος της κλάσης «Boxes», αυτός έχει δηλωθεί ως ιδιωτικό στοιχείο, private, για τον ίδιο λόγο με τους άλλους. Η πρόσβαση σε αυτόν γίνεται μέσω των υπερφορτωμένων τελεστών αγκύλες «[]» και παρενθέσεις «()» που είναι δημόσια μέλη, public, της κλάσης.

Οι υπερφορτωμένες αγκύλες δέχονται σαν ορίσματα ένα αντικείμενο «Boxes» αριστερά τους, γιατί είναι μέλος της κλάσης αυτής, και έναν ακέραιο αριθμό μέσα τους για να γίνει η πρόσβασή στο αντίστοιχο μέλος box[i]. Δηλαδή για πρόσβαση στο «gameBoard.box[5]» που είναι το τετράγωνο δεύτερη γραμμή δεύτερη στήλη, η εντολή είναι «gameBoard[5]». Επιστρέφουν αναφορά στο αντικείμενο αυτό και όχι την τιμή του προκειμένου να μπορεί να γίνει και αλλαγή της τιμής του. Δηλαδή τόσο η εντολή «a=gameBoard[5]» όσο και η «gameBoard[5]=1» είναι αποδεκτές.

Οι υπερφορτωμένες παρενθέσεις δέχονται ένα αντικείμενο «Boxes» σαν όρισμα αριστερά τους, γιατί είναι μέλος της κλάσης αυτής, και δύο ορίσματα ακέραιους αριθμούς μέσα τους για να γίνεται η πρόσβασή στο μέλος box[i] σαν να ήταν δισδιάστατος πίνακας. Επιστρέφει αναφορά στο στοιχείο αυτό όπως και οι αγκύλες. Δηλαδή για πρόσβαση στο «gameBoard.box[5]» που είναι το τετράγωνο δεύτερη γραμμή δεύτερη στήλη, η εντολή είναι «a=gameBoard(2,2)».

2.11 Η βαθμονόμηση

Για να προκύψουν οι συντεταγμένες που είναι αποθηκευμένες στους πίνακες που προαναφέρθηκαν, καθώς και οι παράμετροι των χρωμάτων που μεταβιβάζονται στη συνάρτηση «inRange» γράφτηκαν δύο βοηθητικές εφαρμογές σε γλώσσα C++ με χρήση της βιβλιοθήκης OpenCV.

2.11.1 Βαθμονόμηση χρώματος

Για να βρεθούν οι παράμετροι των χρωμάτων που μεταβιβάζονται στη συνάρτηση «inRange» δημιουργήθηκε μία εφαρμογή που άνοιγε ένα παράθυρο στο οποίο ήταν τοποθετημένα sliders με τη χρήση κατάλληλων συναρτήσεων της βιβλιοθήκης OpenCV. Τα sliders είναι διακόπτες ολίσθησης που μπορούν να μετακινούνται με το ποντίκι για να αλλάζουν τις τιμές μεταβλητών ανάλογα με τη θέση τους. Έτσι με τη μετακίνησή τους άλλαζαν οι τιμές των ορισμάτων που μεταβιβάζονταν στη συνάρτηση «inRange» στην οποία επίσης μεταβιβαζόταν και η εικόνα από την κάμερα αφού είχε μετατραπεί σε HSV. Το τελικό αποτέλεσμα εμφανίζονταν σε ένα παράθυρο σε B&W. Έτσι με δοκιμές προέκυψαν οι σωστές τιμές των παραμέτρων της συνάρτησης. Οι τιμές που έκαναν «ορατά», άσπρα, τα πιόνια του ρομπότ και «αόρατα», μαύρα, όλα τα υπόλοιπα εικονοστοιχεία ήταν αυτές που χρησιμοποιήθηκαν για την ανίχνευση των πιονιών του ρομπότ. Ομοίως αυτές που έκαναν «ορατά» τα πιόνια του παίκτη και «αόρατα» όλα τα υπόλοιπα εικονοστοιχεία, ήταν αυτές που χρησιμοποιήθηκαν για την ανίχνευση των πιονιών του παίκτη.

2.11.2 Βαθμονόμηση θέσης στο σύστημα αναφοράς της κάμερας

Για να βρεθεί σε ποια θέση βρισκόταν το κάθε τετράγωνο όσον αφορά τα εικονοστοιχεία στο σύστημα συντεταγμένων της κάμερας, πίνακες «XcentralPix» και «YcentralPix», δημιουργήθηκε μία άλλη εφαρμογή. Αυτή εμφάνιζε στην οθόνη την εικόνα από την κάμερα στην οποία είχε προσθέσει κύκλους με κέντρο που καθοριζόταν από δύο sliders, ένα για κάθε συντεταγμένη. Με δοκιμές προέκυψαν οι συντεταγμένες για τις οποίες το κέντρο του κύκλου συνέπιπτε με το κέντρο του κάθε κουτιού. Αυτή η βαθμονόμηση πραγματοποιήθηκε αφού είχε κολληθεί μόνιμα η θήκη της κάμερας στο ρομπότ. Οι τιμές που προέκυψαν για κάθε κουτί αποθηκεύτηκαν στους αντίστοιχους πίνακες.

2.11.3 Βαθμονόμηση θέσης στο σύστημα αναφοράς του ρομπότ

Εδώ δεν χρειάστηκε κάποια εφαρμογή. Απλά με τη βοήθεια των χειριστηρίων του ρομπότ μετακινιόταν η κεφαλή στο κέντρο του κάθε τετραγώνου. Στην οθόνη του Arduino UNO ήταν διαθέσιμες οι καρτεσιανές συντεταγμένες της κεφαλής κάθε

στιγμή. Έτσι ήταν γνωστό κάθε επιθυμητό σημείο σε ποιες συντεταγμένες αντιστοιχεί. Ομοίως ενεργοποιώντας τον ηλεκτρομαγνήτη και πλησιάζοντάς τον κοντά στα πιόνια μπορούσε να εξαχθεί συμπέρασμα για την συντεταγμένη Z της κεφαλής που ήταν αρκετά κοντά στα πιόνια για να τα σηκώσει ο ηλεκτρομαγνήτης χωρίς να ακουμπάει σε αυτά. Κατόπιν αυτές οι τιμές αποθηκεύτηκαν στους πίνακες «Xcenters», «Ycenters» και «ZcloseToPawns» αντίστοιχα. Με δοκιμές βρέθηκε η θέση της κεφαλής που δεν εμποδίζει την κάμερα και αποθηκεύτηκε στις μεταβλητές «awayX», «awayY», και «up».

2.12 Η στρατηγική επιλογής της επόμενης κίνησης

Κάθε φορά που παίζει το ρομπότ, η επόμενή του κίνηση γίνεται με βάση το σκεπτικό να μην κερδίζει εύκολα ο αντίπαλος, αλλά ούτε και να κερδίζει τις περισσότερες φορές το ρομπότ, προκειμένου ο χρήστης να μην χάνει γρήγορα το ενδιαφέρον του. Έτσι δεν αναζητήθηκε ο βέλτιστος δυνατός αλγόριθμος όσον αφορά την λήψη της απόφασης για την επόμενη κίνηση του ρομπότ. Το ρομπότ αποφασίζει ως εξής:

Πρώτα ελέγχει μήπως υπάρχουν δύο πιόνια του συνεχόμενα οριζόντια, κάθετα και διαγώνια και αν είναι κενό το διπλανό τετράγωνο, προκειμένου να τοποθετήσει εκεί το τρίτο πιόνι του και να κερδίσει. Αν δεν συμβαίνει αυτό, κατόπιν ελέγχει αν συμβαίνει το ίδιο με τον αντίπαλο, οπότε τοποθετεί στο κενό τετράγωνο ένα πιόνι του για να «κλείσει» τον αντίπαλο και μην μπορέσει να συμπληρώσει τριάδα και κερδίσει ο αντίπαλος. Αν ούτε και αυτό συμβαίνει, τότε ελέγχει τα 4 τετράγωνα που βρίσκονται στις γωνίες, τα 1, 3, 7 και 9, αν είναι άδεια και τοποθετεί εκεί ένα πιόνι. Κατόπιν ελέγχει τα ενδιάμεσα, τα 2, 4, 6 και 8 και τέλος το κεντρικό, το 5.

Η συνάρτηση που υλοποιεί αυτή τη διαδικασία ελέγχει και άλλα πράγματα. Δηλαδή ελέγχει μήπως ο αντίπαλος έχει κερδίσει μετά την τελευταία του κίνηση και επιστρέφει 0. Όταν αποφασίσει την επόμενη κίνηση του ρομπότ, τότε επιστρέφει το κουτί που πρέπει να τοποθετηθεί το πιόνι, πχ. επιστρέφει τον αριθμό 5 για το κεντρικό κουτί. Αν η κίνηση του ρομπότ είναι η τελευταία και με αυτήν κερδίζει, τότε επιστρέφει τη θέση με αρνητικό πρόσημο. Δηλαδή για το κουτί 5, θα επιστρέψει τον αριθμό -5. Αν δεν υπάρχουν κενές θέσεις και δεν έχει κερδίσει κανείς, οπότε έχουμε ισοπαλία, τότε επιστρέφει τον αριθμό -10. Αν υπάρχει μόνο ένα κενό κουτί, αλλά με την τοποθέτηση πιονιού του εκεί δεν κερδίζει, τότε ο αριθμός είναι θετικός και στην επόμενη επανάληψη του βρόγχου θα ανακαλύψει ότι έχουμε ισοπαλία. Για να μην συμβεί αυτό, μόλις επιστρέφει η συνάρτηση γίνεται έλεγχος μέσα από τη main() για κενή θέση. Αν δεν υπάρχει κενή θέση τότε τερματίζεται η παρτίδα ισόπαλα. Αν δεν γινόταν αυτό, τότε θα περίμενε πρώτα να παίξει ο αντίπαλος και ύστερα θα συνειδητοποιούσε ότι δεν υπάρχει κενό. Όμως επειδή ο αντίπαλος δεν θα μπορούσε να παίξει, τότε θα κόλλαγε η εφαρμογή.

2.13 Ο πηγαίος κώδικας της εφαρμογής

Στις υποπαραγράφους που ακολουθούν αναλύεται ο πηγαίος κώδικας της εφαρμογής της τρίλιζας. Για την ανάπτυξή του χρησιμοποιήθηκε η εφαρμογή Geanny που είναι ένα γραφικό παραθυρικό περιβάλλον ανάπτυξης εφαρμογών σε διάφορες γλώσσες προγραμματισμού. Ανάλογα με την κατάληξη του πηγαίου κώδικα, «.cpp», «.c», «.py» κλπ. καταλαβαίνει σε ποια γλώσσα αναφέρεται και διαμορφώνει κατάλληλα το περιβάλλον. Τέλος μπορεί να κάνει και τη μεταγλώττιση καλώντας το g++ χωρίς να χρειάζεται να καταφεύγει ο χρήστης συνεχώς στην κονσόλα. Επίσης μπορεί να εκτελεί τον μεταγλωττισμένο κώδικα σε παράθυρο τερματικού. Στην τελευταία έκδοση του Raspbian Jessie, το Geanny είναι ήδη προεγκατεστημένο. Ωστόσο, επειδή ο κώδικας για την εργασία αυτή αναπτύχθηκε νωρίτερα από τη διάθεση αυτής της έκδοσης του λειτουργικού, το Geanny δεν ήταν προεγκατεστημένο. Έτσι εγκαταστάθηκε με την εντολή:

sudo apt-get install geanny

Για την μεταγλώττιση του κώδικα της εφαρμογής «TicTacToe» αυτής της εργασίας πρέπει ο «g++» να κληθεί με τα ορίσματα «$(pkg-config --cflags opencv)» για τη μεταγλώττιση, και «$(pkg-config --libs opencv)» και «-lwiringPi» για την διασύνδεση, λόγω της χρήσης των βιβλιοθηκών OpenCV και wiringPi αντίστοιχα. Παρακάτω ακολουθούν ξεχωριστοί υποπαράγραφοι για κάθε συνάρτηση του κώδικα και η κάθε μία ξεκινάει από καινούρια σελίδα για να την βρίσκει πιο εύκολα ο αναγνώστης.

2.13.1 Οι βιβλιοθήκες

Εδώ παρατίθενται οι βιβλιοθήκες και οι χώροι ονομάτων που χρησιμοποιήθηκαν για την εφαρμογή αυτή. Παρακάτω ακολουθούν αυτά:

```
#include <iostream>
#include <wiringSerial.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/time.h>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;
```

2.13.2 Οι συναρτήσεις elpasedTime() και Pause()

Η συνάρτηση double elapsedTime() χρησιμοποιήθηκε για να επιστρέφει τον χρόνο σε δευτερόλεπτα που έχει περάσει από την εκκίνηση του Raspberry. Ο λόγος είναι ότι η gettimeofday που χρησιμοποιείται δεν επιστρέφει την τιμή αλλά την αποθηκεύει σε μία δομή. Έτσι είναι κάπως δύσχρηστη να χρησιμοποιείται κάθε φορά μέσα στη συνάρτηση που την καλεί. Αντιθέτως η elapsedTime() που γράφτηκε παίρνει αυτή τη δομή και την αποθηκεύει σε έναν δεκαδικό αριθμό που είναι πιο εύχρηστος. Όμως επειδή η elapsedTime() είναι πολύ μικρή συνάρτηση, κρίθηκε σκόπιμο να δηλωθεί ως «inline» προκειμένου να τοποθετηθεί μέσα στον κώδικα της συνάρτησης που την καλεί έτσι ώστε να αποφευχθούν άσκοπες καθυστερήσεις κατά την εκτέλεση από τις πάρα πολλές κλήσεις της που υπάρχουν. Ωστόσο η δήλωση «inline» αποτελεί προτροπή προς τον μεταγλωττιστή και όχι υποχρέωση. Δηλαδή, ο μεταγλωττιστής μπορεί να κρίνει για κάποιο λόγο ότι δεν μπορεί να μπει ένθετη μέσα στον κώδικα, οπότε σε αυτή τη περίπτωση θα συμπεριφέρεται σαν κανονική συνάρτηση (Νίκος Χατζηγιαννάκης).

Η Pause(double) σταματάει την εκτέλεση του κώδικα για τουλάχιστον τόσα δευτερόλεπτα, όση είναι η τιμή του ορίσματός της. Δεν έχει την αξιοπιστία άλλων παρόμοιων συναρτήσεων που υπάρχουν, αλλά εδώ δεν ήταν το ζητούμενο η αξιοπιστία γιατί δεν πρόκειται για χρονικά κρίσιμο κομμάτι κώδικα. Η sleep() απορρίφθηκε γιατί θέλει ακέραια τιμή δευτερολέπτων, ενώ θέλαμε να μπορούμε να εισάγουμε και μικρότερες τιμές. Αντιθέτως η delay() της βιβλιοθήκης wiringPi δέχεται milliseconds σαν όρισμα με αποτέλεσμα για αναμονή πολλών δευτερολέπτων να προκύπτουν μεγάλα νούμερα. Παρακάτω ακολουθεί ο πηγαίος κώδικας αυτών των συναρτήσεων:

```
inline double elapsedTime (void)
{               /* This function returns system time in seconds as double precision floating number. */
   struct timeval time;
   gettimeofday(&time,NULL);
   return (double)time.tv_sec+0.000001*(double)time.tv_usec;
}
void Pause (double pause=1.0)
{  //This function pauses program execution at least for the seconds given as argument.
    // Default pause duration is 1 second.
   double init=elapsedTime(), now=init;
   while (now-init < pause)
   {
      now = elapsedTime();
      if (init > now) init=now;// If system time counter has started all over again, reset variable "now"
   }
   return;
}
```

2.13.3 Η κλάση αντικειμένων Boxes

Εδώ παρουσιάζεται η κλάση «Boxes». Αυτή έχει περιγραφεί σε προηγούμενα κεφάλαια. Αποτελείται από το ιδιωτικό μέλος πίνακα box[10] και από τις δημόσιες μεθόδους clear() για καθορισμό τιμής στα στοιχεία του box καθώς και για αρχικό καθορισμό τιμής από την συνάρτηση δόμησης. Επίσης έχουν υπερφορτωθεί οι τελεστές αγκύλης και παρένθεσης για την πρόσβαση στον πίνακα box με τον τρόπο που περιγράφτηκε. Επιπλέον έχει υπερφορτωθεί ο τελεστής «=» για αντιγραφή όλων των τιμών του πίνακα box[i] από ένα αντικείμενο Boxes σε ένα άλλο εξαιρώντας τη φωλιά, box[0]. Επίσης έχει υπερφορτωθεί το «==» για να συγκρίνει δύο αντικείμενα Boxes αν έχουν ίδιες τιμές όλα τα στοιχεία του πίνακα box[] ή όχι εξαιρώντας τη φωλιά. Τέλος έχει υπερφορτωθεί το «-» που συγκρίνει δύο αντικείμενα Boxes, επίσης εξαιρώντας το box[0], και επιστρέφει θετικό αριθμό αν στο πρώτο έχει αλλάξει μόνο ένα στοιχείο του box[i] σε σχέση με το δεύτερο και όταν αυτό από 0 έχει γίνει 1. Επιστρέφει την θέση του στοιχείου από 1 μέχρι 9. Αν και τα δύο αντικείμενα έχουν ίδια στοιχεία, επιστρέφει 0. Στην ουσία έχει το νόημα να βρίσκει σε ποιο τετράγωνο έχει τοποθετήσει ο αντίπαλος δικό του πιόνι, ποιο είναι αυτό το τετράγωνο, και τέλος αν έχει γίνει ζαβολιά. Στην περίπτωση της ζαβολιάς επιστρέφεται ένας αρνητικός αριθμός, αναλόγως το πλήθος των μεταβολών που έχουν γίνει, αλλά δεν εξετάζεται τι συμβαίνει παραπέρα γιατί αυτό είναι αντικείμενο άλλης συνάρτησης. Παρακάτω ακλουθεί το αντικείμενο Boxes:

```
/* The contents of the nine boxes of the game board will be stored into the Boxes object.
 *For this program Box or Boxes is/are the square(s)
 * where Delta Robot or its opponent place their pawns.
 *If there is a player's pawn inside the i-th box, then box[i] is 1.
 * If there is a Delta Robot's pawn, then box[i] is 2. If it is empty then box[i] is 0.
 * The box[1] represents the upper left box and box[9] represents the lower right box.
 * Element box[0] is to store the number of pawns inside the nest. Nest is the big rectangle
 *where Delta Robot's own pawns are stored.
 */
class Boxes
{

private:

  int box[10];   // The contents of the boxes. They are private so as to access them by member
                 //... functions which check if accessing inside reserved matrix space

public:

  int clear(int value=0)
  {          // Method to change Boxes contents to any value or to 0 if there is no argument
```

```
   for (int i=0; i<10; i++) box[i]=value;
   return value;
}

Boxes(int init=0)
{          // The Constructor writes the initial values to boxes or 0 if there is no argument
   clear(init);
   return;
}


int& operator [] (int i)
{   // Operator [] overload to access private member box[i] checking for valid memory position
   if ( i>=0 && i<10 ) return box[i];
   else
   {
      cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                                    terminate."<<endl;
      exit(1);
   }
}

int& operator () (int i, int j)
     // Operator () overload to access box[i] as a two dimensional matrix. (3 rows 3 columns)
     // (1,1) for upper left box and (3,3) for lower right. Nest (box[0]) cannot be accessed.
{
   if ( i>=1 && i<=3 && j>=1 && j<=3) return box[3*(i-1)+j];
   else
   {
      cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                                    terminate."<<endl;
      exit(1);
   }
}

void operator = (Boxes bx)
{          // Operator = overload to copy member box[i] contents omitting box[0]
   for (int i=1; i<10; i++) box[i]=bx.box[i];
   return;
}
```

```
  /* Operator == overload checks whether all elements are equal or not. Member box[0] is omitted.
   * If all elements are equal, it returns true, else false.
   */
  bool operator == (Boxes bx)
  {
     for (int i=1; i<10; i++) if (box[i]!=bx.box[i]) return false;
     return true;
  }


  /* This operator - overload checks whether there is ONLY ONE change
    *from 0 to 1 (this->box[i]=1 and bx.box[i]=0) for only one i
    * and for the rest values of i this->box[i]==bx.box[i].
    *In that case it returns the position (i) of the element that has changed.
    * If all elements are equal, it returns 0,
    *whereas in any other case it returns <0 (minus number of changes).
   */
  int operator - (Boxes bx)
  {
     int counter=0;
     int place=0;
     for (int i=1; i<10; i++)
     {
        if (box[i]!=bx.box[i])
        {
           counter++;
           place=i;
        }
     }
     if ( counter==1 && box[place]==1 && bx.box[place]==0 ) return place;
     else return -counter;
  }
};  // End of class Boxes declaration.
```

2.13.4 Η πρόσβαση στους πίνακες αποθήκευσης συντεταγμένων

Παρακάτω παρουσιάζονται οι συναρτήσεις πρόσβασης στους πίνακες συντεταγμένων όπως έχουν περιγραφεί, καθώς και η καθολική σταθερά «up»:

```
  /* This function returns the X coordinate of a given box. i=1 is for the upper left box
   *and i=9 for the lower right box.
   * The returned coordinate refers to the reference system of the camera,
   * not the Delta Robot and the units are in pixels.
   */
int XcentralPixel(int place)
{
  const int XcentralPix[9]=
  {
    407, 309, 210,
    405, 312, 214,
    405, 312, 217
  };

  if (place>=1 && place<=9) return XcentralPix[place-1];
  else
  {
    cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                        terminate."<<endl;
    exit(1);
  }
}

  /* This function returns the Y coordinate of a given box. i=1 is for the upper left box
   *and i=9 for the lower right box.
   * The returned coordinate refers to the reference system of the camera,
   * not the Delta Robot and the units are in pixels.
   */
int YcentralPixel(int place)
{
  const int YcentralPix[9]=
  {
    422, 419, 417,
    320, 318, 318,
    228, 225, 222
  };

  if (place>=1 && place<=9) return YcentralPix[place-1];
  else
  {
```

```
      cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                                                terminate."<<endl;
      exit(1);
   }
}


   /* This function returns the X coordinate of a given box. i=1 is for the upper left box
    * and i=9 for the lower right box.
    * If i is 0 then it returns the X coordinate of the position where the camera
    * has full view of the game board.
    * If i is negative then it returns the X coordinate of the respective nest position. -1 is for
    * the right place and -5 for the left.
    * The returned coordinate refers to the reference system of the Delta Robot,
    * not the camera and the units are centimeters multiplied by 100 (not pixels).
    */
int X (int i)
{
      /* Matrix "Xcenters" stores the X coordinate of the center of each box.
       *For example Xcenters[0] is the X coordinate of the upper left box.
       * The Xcenters[8] is the X coordinate of the lower right box.
       * The Xcenters[9] till [13] is for the 5 places of the nest (big rectangle on the bottom).
       *9 is the left 13 the right one.
       * These coordinates refer to the reference system of the Delta Robot, not the camera
       *and the units are centimeters multiplied by 100 (not pixels).
       */
   const int Xcenters[14]=
   {
      1087, 1159, 1121,
      439, 437, 500,
      -270, -270, -270,
      -874,-871,-874,-859,-843
   };

   const int awayX= -568;  // The X coordinate of the position where the camera has
                           // ...full view of the game board (cm*100)

   if (i>=1 && i<=9) return Xcenters[i-1];
   else if (i<=-1 && i>=-5) return Xcenters[-i+8];
   else if (i==0) return awayX;
   else
   {
      cout << "\n ERROR!\nTried to access a non existing matrix element. Program will terminate."
                        <<endl;
      exit(1);
```

```
    }
}

    /* This function returns the Y coordinate of a given box. i=1 is for the upper left box
     * and i=9 for the lower right box.
     * If i is 0 then it returns the Y coordinate of the position where the camera
     *has full view of the game board.
     * If i is negative then it returns the Y coordinate of the respective nest position.
     * -1 is for the right place and -5 for the left.
     * The returned coordinate refers to the reference system of the Delta Robot,
     * not the camera and the units are centimeters multiplied by 100 (not pixels).
     */
int Y (int i)
{
      /* Matrix "Ycenters" stores the Y coordinate of the center of each box.
       *For example Ycenters[0] is the Y coordinate of the upper left box.
       * The Ycenters[8] is the Y coordinate of the lower right box.
       * The Ycenters[9] till [13] is for the 5 places of the nest (big rectangle on the bottom).
       *9 is the left 13 the right one.
       * These coordinates refer to the reference system of the Delta Robot, not the camera
       *and the units are centimeters multiplied by 100 (not pixels).
       */
    const int Ycenters[14]=
    {
       578, -30, -609,
       611, -1, -653,
       624, 9, -547,
       645, 300, 17, -286, -669
    };

    const int awayY= 1011;  // The Y coordinate of the position where the camera has full view
                            // … of the game board (cm*100)

    if (i>=1 && i<=9) return Ycenters[i-1];
    else if (i<=-1 && i>=-5) return Ycenters[-i+8];
    else if (i==0) return awayY;
    else
    {
       cout << "\n ERROR!\nTried to access a non existing matrix element. Program will terminate."
            <<endl;
       exit(1);
    }
}
```

```
const int up = -3122;  /* Z coordinate at a high level, much higher than game board level.
                        * It refers to the reference system of the Delta Robot,
                        * not the camera and the units are centimeters multiplied by 100 (not pixels).
                        */


   /* This function returns the Z coordinate of a given box close to pawn level.
    *i=1 is for the upper left box and i=9 for the lower right box.
    * If i is negative then it returns the Z coordinate of the respective nest position at pawn level.
    *-1 is for the right place and -5 for the left.
    * The returned coordinate refers to the reference system of the Delta Robot,
    * not the camera and the units are centimeters multiplied by 100 (not pixels).
    */
int down (int i)
{
      /* Matrix "ZcloseToPawns" stores the Z coordinate of pawn level for each box.
       *For example ZcloseToPawns[0] is the Z coordinate for the upper left box.
       * The ZcloseToPawns[8] is the Z coordinate for the lower right box.
       * The ZcloseToPawns[9] till [13] is for the 5 places of the nest (big rectangle on the bottom).
       *9 is the left 13 the right one.
       * These coordinates refer to the reference system of the Delta Robot, not the camera
       *and the units are centimeters multiplied by 100 (not pixels).
       */
   const int ZcloseToPawns[14]=
   {
      -3371, -3372, -3350,
      -3399, -3405, -3427,
      -3423, -3427, -3446,
      -3450, -3450, -3450, -3450, -3450
   };

   if (i>=1 && i<=9) return ZcloseToPawns[i-1];
   else if (i<=-1 && i>=-5) return ZcloseToPawns[-i+8];
   else
   {
      cout << "\n ERROR!\nTried to access a non existing matrix element. Program will terminate."
              <<endl;
      exit(1);
   }
}
```

2.13.5 Η συνάρτηση εκτέλεσης του driver της κάμερας

Παρακάτω ακολουθεί η συνάρτηση εκτέλεσης της εντολής «sudo modprobe bcm2835-v4l2» για τη λειτουργία της κάμερας:

```
int executeModprobe(void)   // Executes the sudo modprobe bcm2835-v4l2 command
{
  int status=1;

  if (fork()==0)
  {
    char executable[] = "/usr/bin/sudo";
    char argument1[] = "modprobe";
    char argument2[] = "bcm2835-v4l2";
    char *arguments[] = { executable, argument1, argument2, NULL };
    execve(executable, arguments, NULL);
    perror("execve");   // Execve never returns. If it does, then an error may have occurred.
    exit(1);
  }

  wait(&status);

 /* Parent is waiting for driver to be executed before going further.
  *In normal cases child returns 0. Otherwise an error has occurred.
  * However the driver may have been already loaded from a previous run or from bash by the user.
  *For that reason this program will NOT stop as the video driver may be running properly.
  */

  if (status!=0)
  {
    cout << "\n Modprobe was NOT executed properly. If any problems occur,
                        then press Ctrl+C and run this program again.\n";
    cout << " If problems insist, then execute from terminal the command
          \"sudo modprobe bcm2835-v4l2\" and then run this program with argument \"!\".\n\n";
  }

  return status;  // Returns 0 if everything is OK
}
```

2.13.6 Η συνάρτηση εκτύπωσης του μηνύματος βοήθειας

Παρακάτω ακολουθεί η συνάρτηση εκτύπωσης του μηνύματος βοήθειας:

```
void printHelpMessage(void)
{
   cout << "\n  Help:" << endl;
   cout << " In normal cases no arguments are needed to execute this program." << endl;
   cout << " Use argument \"?\" or \"H\" or \"h\" for help."<<endl;
   cout << " Use argument \"!\" if you do NOT want to execute modprobe driver.\n";
   cout << "\n   Caution!!"<<endl;
   cout << " In that case be sure that you have already executed from terminal the command
                              \"sudo modprobe bcm2835-v4l2\" otherwise ";
   cout << "this program will not run properly.\n This option should only be used if there are
                              problems in running modprobe from this program.\n"<<endl;
}
```

2.13.7 Η συνάρτηση ανίχνευσης πιονιών

Εδώ παρουσιάζεται η συνάρτηση ανίχνευσης πιονιών. Ο τρόπος λειτουργίας της έχει αναλυθεί σε προηγούμενη παράγραφο. Τα τρία προσωρινά αντικείμενα «temp1», «temp2» και «temp3» αρχικοποιούνται με τιμές 0, 2 και 3 αντίστοιχα προκειμένου αρχικά να μην έχουν ίδιες τιμές. Έτσι εξασφαλίζεται ότι ο βρόγχος while θα εκτελεστεί τουλάχιστον 3 φορές. Με τη μέθοδο «at» της κλάσης «Mat» γίνεται η πρόσβαση στη τιμή κάποιου εικονοστοιχείου. Πχ. Η «playerPawns.at<uchar>(YcentralPixel(i),XcentralPixel(i))» επιστρέφει αναφορά στο αντίστοιχο εικονοστοιχείο, όχι δείκτη. Στα ορίσματα δίνεται η συντεταγμένη Υ πρώτα και ύστερα η Χ γιατί η Υ είναι οι γραμμές του καρέ, η μικρή διάσταση, και η Χ οι στήλες, η μεγάλη διάσταση.

Για την πρόσβαση στα εικονοστοιχεία υπάρχουν δύο ξεχωριστά blocks με βρόγχους, ένα για αυτά που αφορούν τα πιόνια του ρομπότ και ένα άλλο για αυτά του παίκτη. Θα μπορούσε αυτή η πρόσβαση στο κάθε ένα αντικείμενο, το «playerPawns» και το «DeltaRobotPawns», να γινόταν στην ίδια for αντί σε δύο ξεχωριστές. Όμως επιλέχτηκε αυτή η λύση για καλύτερη εκμετάλλευση της μνήμης cache του επεξεργαστή. Δηλαδή, κάποια γειτονικά στοιχεία του πίνακα playerPawns βρίσκονται πιθανότατα στην cache κατά την πρώτη αναφορά σε αυτόν. Έτσι η πρόσβαση στα γειτονικά τους κατά τις επόμενες επαναλήψεις των εσωτερικών βρόγχων είναι γρηγορότερη. Αντιθέτως, αν αμέσως μετά ακολουθούσε η πρόσβαση στον DeltaRobotPawns, τότε πιθανότατα ο playerPawns να έβγαινε από τη cache λόγω έλλειψης χώρου και στην επόμενη επανάληψη να έπρεπε να ξαναμπεί για να γίνει η πρόσβαση στο επόμενο στοιχείο του. Σαν αποτέλεσμα θα είχαμε μεγαλύτερους χρόνους εκτέλεσης, λόγων των πολλών προσβάσεων στη κεντρική μνήμη RAM. Για αυτό αποφασίστηκε η χρήση δύο ξεχωριστών for loops.

Σαν ορίσματα δέχεται το αντικείμενο που έχει δημιουργηθεί στη main() για τη λειτουργία της κάμερας και ένα αντικείμενο κλάσης «Boxes» για την αποθήκευση των θέσεων των πιονιών. Παρακάτω ακολουθεί η συνάρτηση ανίχνευσης πιονιών:

```
void detectPawns (VideoCapture& cameraFrame, Boxes &Board)
{               // Detects and Identifies which pawn is in which box.

   const int lowHueRobot=113;
   const int highHueRobot=204;
   const int lowSatRobot=108;
   const int highSatRobot=223;
   const int lowValueRobot=81;
   const int highValueRobot=202;

   const int lowHuePlayer=21;
```

```
const int highHuePlayer=65;
const int lowSatPlayer=93;
const int highSatPlayer=240;
const int lowValuePlayer=69;
const int highValuePlayer=201;

double previousTime=0.0, currentTime=0.0;

bool flag=false;

Mat originalImage, hsvImage, DeltaRobotPawns, playerPawns;

Boxes temp1, temp2(2), temp3(3);
                        // Three temporary Boxes objects to store pawn location on the game board

while (!flag)
{
  currentTime=elapsedTime();

    // If system time counter has started all over again, then reset variable "previousTime"
  if (currentTime < previousTime) previousTime=currentTime;

  if ( currentTime-previousTime > 1.0 || previousTime==0.0)
        // Leave a time interval between successive frames.
  {
    previousTime=currentTime;

    temp3=temp2;
    temp2=temp1;
    temp1.clear();  // Initially every element of temp1 becomes zero

    for (int i=0; i<6; i++) cameraFrame >> originalImage;
        // Empty camera buffer, capture the latest frame and copy it to an object of Mat type

    cvtColor(originalImage, hsvImage, COLOR_BGR2HSV);
        // Transform the BGR image to HSV

      // Transform HSV color image in black and white according to the given HSV values
    inRange(hsvImage, Scalar(lowHuePlayer,lowSatPlayer,lowValuePlayer),
                        Scalar(highHuePlayer,highSatPlayer,highValuePlayer), playerPawns);
    inRange(hsvImage, Scalar(lowHueRobot,lowSatRobot,lowValueRobot),
                        Scalar(highHueRobot,highSatRobot,highValueRobot), DeltaRobotPawns);

      // Apply Gaussian Blur filter and then detect any pawns.
    GaussianBlur( playerPawns, playerPawns, Size(9, 9), 2.0, 2.0 );
```

```
GaussianBlur( DeltaRobotPawns, DeltaRobotPawns, Size(9, 9), 2.0, 2.0 );


for( int i = 1; i <= 9; i++ )
{     // In the Player pawns matrix, for each white pixel found inside a
      // …square of 20x20 pixels surrounding the center of i-th box, then counterPlayer++

   int counterPlayer=0;
   for (int j=-10; j<=10; j++)
   {
      for (int k=-10; k<=10; k++)
      {
         if (playerPawns.at<uchar>(YcentralPixel(i)+j,XcentralPixel(i)+k)) counterPlayer++;
      }

   }
   if (counterPlayer>=300)
   {                 // If there are more than 300 white pixels then there is a pawn in the i-th box
      temp1[i]=1;
      Point pix1(XcentralPixel(i)-15,YcentralPixel(i)-15);
      Point pix2(XcentralPixel(i)+15,YcentralPixel(i)+15);
      Point pix3(XcentralPixel(i)-15,YcentralPixel(i)+15);
      Point pix4(XcentralPixel(i)+15,YcentralPixel(i)-15);
      line (originalImage, pix1, pix2, Scalar(0,255,0),3,CV_AA);          // Draws a green X
      line (originalImage, pix3, pix4, Scalar(0,255,0),3,CV_AA);
   }

}

for( int i = 1; i <= 9; i++ )
{     // In the Delta Robot pawns matrix, for each white pixel found inside a
      // …square of 20x20 pixels surrounding the center of i-th box, then counterRobot++

   int counterRobot=0;
   for (int j=-10; j<=10; j++)

   {
      for (int k=-10; k<=10; k++)
      {
         if (DeltaRobotPawns.at<uchar>(YcentralPixel(i)+j,XcentralPixel(i)+k)) counterRobot++;
      }

   }

   if (counterRobot>=300)
```

```
            {                   // If there are more than 300 white pixels then there is a pawn in the i-th box
               temp1[i]=2;
               Point center(XcentralPixel(i), YcentralPixel(i));
               circle( originalImage, center, 3, Scalar(255,0,0), -1, CV_AA );    // Draws a blue dot
               circle( originalImage, center, 20, Scalar(0,0,255), 3, CV_AA );   // Draws a red O
            }

         }

           // Show image
         imshow ("Captured Image", originalImage);  // Show image on the window.
         for (int i=0; i<10; i++) waitKey(1);    // This is required to show image.

         // Keep scanning until you find three successive frames with identical Boxes elements
         flag = temp1==temp2 && temp2==temp3;

      }  // Endif

   }  // End while loop

   Board=temp1;    // Return a Boxes object with pawn location on the game board
                   // Nest (box[0]) will not be affected.

   return;
}
```

2.13.8 Η συνάρτηση απόφασης της επόμενης κίνησης

Εδώ παρουσιάζεται η συνάρτηση απόφασης της επόμενης κίνησης όπως έχει αναλυθεί σε προηγούμενη παράγραφο. Δέχεται σαν όρισμα τις τρέχουσες θέσεις των πιονιών και επιστρέφει τη θέση που πρέπει να τοποθετήσει το ρομπότ το επόμενο πιόνι του όπως έχει περιγραφεί σε προηγούμενη παράγραφο. Παρακάτω ακολουθεί η συνάρτηση απόφασης:

```
/* Decides Delta Robot's next move and returns the number of the box.
 *If player has won it returns 0.
 * If it is the last move of Delta Robot and robot wins, then it returns a negative number which is
 * the place of the next box negated. If there is no box empty and there is no winning combination,
 * then nobody wins (tie) and returns -10.
 */
int decideNextMove ( Boxes Board )
{

  /* If three player's pawns are diagonal or three in a raw or in a column, then player wins. */
  if (Board(1,1)==1 && Board(2,2)==1 && Board(3,3)==1) return 0;
  if (Board(1,3)==1 && Board(2,2)==1 && Board(3,1)==1) return 0;
  for (int i=1; i<=3; i++)
  {
     if (Board(i,1)==1 && Board(i,2)==1 && Board(i,3)==1) return 0;
     if (Board(1,i)==1 && Board(2,i)==1 && Board(3,i)==1) return 0;
  }

  /* If there are two Delta Robot's pawns in a row or column or diagonal,
   * then place a third pawn in the third place, if empty. Return negative because Robot wins.
   */
  for (int i=1; i<=3; i++)
  {
     if (Board(i,1)==0 && Board(i,2)==2 && Board(i,3)==2) return 3*(1-i)-1;
     if (Board(i,1)==2 && Board(i,2)==0 && Board(i,3)==2) return 3*(1-i)-2;
     if (Board(i,1)==2 && Board(i,2)==2 && Board(i,3)==0) return 3*(1-i)-3;
     if (Board(1,i)==0 && Board(2,i)==2 && Board(3,i)==2) return -i;
     if (Board(1,i)==2 && Board(2,i)==0 && Board(3,i)==2) return -3-i;
     if (Board(1,i)==2 && Board(2,i)==2 && Board(3,i)==0) return -6-i;
  }

  if (Board(1,1)==0 && Board(2,2)==2 && Board(3,3)==2) return -1;
  if (Board(1,1)==2 && Board(2,2)==0 && Board(3,3)==2) return -5;
  if (Board(1,1)==2 && Board(2,2)==2 && Board(3,3)==0) return -9;
  if (Board(3,1)==0 && Board(2,2)==2 && Board(1,3)==2) return -7;
  if (Board(3,1)==2 && Board(2,2)==0 && Board(1,3)==2) return -5;
  if (Board(3,1)==2 && Board(2,2)==2 && Board(1,3)==0) return -3;
```

```
    /* If there are two player's pawns in a row or column or diagonal,
     * then place a pawn in the third place, if empty, to avoid losing from player.
     */
  for (int i=1; i<=3; i++)
  {
    if (Board(i,1)==0 && Board(i,2)==1 && Board(i,3)==1) return 3*(i-1)+1;
    if (Board(i,1)==1 && Board(i,2)==0 && Board(i,3)==1) return 3*(i-1)+2;
    if (Board(i,1)==1 && Board(i,2)==1 && Board(i,3)==0) return 3*(i-1)+3;
    if (Board(1,i)==0 && Board(2,i)==1 && Board(3,i)==1) return i;
    if (Board(1,i)==1 && Board(2,i)==0 && Board(3,i)==1) return 3+i;
    if (Board(1,i)==1 && Board(2,i)==1 && Board(3,i)==0) return 6+i;
  }

  if (Board(1,1)==0 && Board(2,2)==1 && Board(3,3)==1) return 1;
  if (Board(1,1)==1 && Board(2,2)==0 && Board(3,3)==1) return 5;
  if (Board(1,1)==1 && Board(2,2)==1 && Board(3,3)==0) return 9;
  if (Board(3,1)==0 && Board(2,2)==1 && Board(1,3)==1) return 7;
  if (Board(3,1)==1 && Board(2,2)==0 && Board(1,3)==1) return 5;
  if (Board(3,1)==1 && Board(2,2)==1 && Board(1,3)==0) return 3;

    /* In any other case, put a pawn to a corner, if empty.
     * If not, put it somewhere inside.
     */
  if (Board[1]==0) return 1;
  if (Board[3]==0) return 3;
  if (Board[7]==0) return 7;
  if (Board[9]==0) return 9;
  for (int i=2; i<=8; i+=2) if (Board[i]==0) return i;
  if (Board[5]==0) return 5;

  return -10;    // There is no empty space or winning combination.
}
```

2.13.9 Η συνάρτηση αποστολής δεδομένων

Εδώ παρουσιάζεται η συνάρτηση που αναλαμβάνει να αποστείλει στο Delta Robot μέσω της σειριακής θύρας UART τις επιθυμητές συντεταγμένες της κεφαλής του και την επιθυμητή κατάσταση του ηλεκτρομαγνήτη που αποτελούν και ορίσματά της. Αρχικά ελέγχει αν υπάρχουν δεδομένα στην μνήμη εισερχομένων. Αν δεν υπάρχουν ή αν υπάρχουν αλλά δεν είναι ο χαρακτήρας «@», τότε δεν κάνει τίποτα και περιμένει μέχρι το Delta Robot να στείλει το «@», ενώ παράλληλα τυπώνει μήνυμα στην οθόνη ανά τακτά διαστήματα. Μόλις τον λάβει, στέλνει στη σειριακή τα δεδομένα όπως έχει αναλυθεί στο κεφάλαιο 1. Δέχεται σαν πρώτο όρισμα τον file descriptor που έχει επιστραφεί από το λειτουργικό σύστημα από την κλήση της συνάρτησης μέσα από την main() που ανοίγει τη σειριακή επικοινωνία. Παρακάτω ακολουθεί η συνάρτηση αυτή:

```
  // Function serialSend sends to the Delta Robot the coordinates and magnet status via UART.
  // Numbers must be integers in centimeters multiplied by 100.
   // For example for (x,y,z)=(12.32,-6.78,-34.09) the numbers must be (x,y,z)=(1232,-678,-3409)
void serialSend ( int fd, int x=0, int y=0, int z=-3100, bool mag=false )
{
  double currentTime, oldTime=elapsedTime();
  while(1)
  {
    if ( serialDataAvail(fd) )  // If Delta Robot has sent a '@' then transmit coordinates. Else loop.
    {
      if ( serialGetchar(fd) == '@' )
      {
        serialFlush (fd);
        serialPutchar (fd,'@');
        serialPrintf (fd,"X= %d\nY= %d\nZ= %d\nElectromagnet= %d\n",x,y,z,(int)mag);
        serialPutchar (fd,'$');
        Pause(0.4);  // Wait a little bit for Delta Robot to execute command.
        return;
      }
    }
    currentTime=elapsedTime();
    if ( currentTime<oldTime ) oldTime=currentTime;
          // If system time counter has started all over again, then reset variable "oldTime"
    if ( currentTime-oldTime > 5.0 )
    {
      cout << "\n\n Warning!!\n Delta Robot is not responsive for too long." << endl;
      cout << " Check connection and press the black button on the Arduino Shield." << endl;
      oldTime=currentTime;
    }
  }
}
```

2.13.10 Η συνάρτηση μετακίνησης πιονιού

Εδώ παρουσιάζεται η συνάρτηση που αναλαμβάνει να μετακινήσει ένα πιόνι από ένα μέρος σε ένα άλλο με ορίσματα τα αντίστοιχα κουτάκια. Δηλαδή το κουτί 1 είναι το πάνω αριστερά και το 9 το κάτω δεξιά. Οι αρνητικοί αριθμοί αφορούν τις θέσεις της φωλιάς, ενώ το 0 τη θέση μακριά από την κάμερα, όπως έχει αναλυθεί σε προηγούμενη παράγραφο. Το πρώτο όρισμα είναι ο file descriptor που αντιστοιχεί στη σειριακή θύρα και τα άλλα δύο ορίσματα είναι η θέση προέλευσης και προορισμού αντίστοιχα. Παρακάτω ακολουθεί η συνάρτηση αυτή:

```
  /* This function moves a pawn from a box to another box. Box is a number between 1 to 9.
   * 1 is for upper left and 9 for lower right. If it is negative then it implies the nest.
   * -5 is the right place and -1 for the left. If it is zero then it means far away.
   */
void movePawn(int fd, int from, int to)
{
  const double pause = 0.4;

  serialSend (fd, X(from), Y(from), up, false);   // Go on top of the pawn
  Pause(pause);
  serialSend (fd, X(from), Y(from), down(from), true);    // And grab it (turn magnet on)
  Pause(pause);
  serialSend (fd, X(from), Y(from), up, true);

  serialSend (fd, X(to), Y(to), up, true);    // Then go to destination
  Pause(pause);
  serialSend (fd, X(to), Y(to), up, false);   // And release the pawn (turn magnet off)

  Pause(pause);

  return;
}
```

2.13.11 Η συνάρτηση επαναφοράς του παιχνιδιού

Εδώ παρουσιάζεται η συνάρτηση που αναλαμβάνει να επαναφέρει τα πιόνια στην προηγούμενη έγκυρη θέση τους. Αυτή καλείται μετά από μη έγκυρη κίνηση του παίκτη. Δέχεται σαν ορίσματα την τρέχουσα και την επιθυμητή κατάσταση του πάγκου του παιχνιδιού και επιστρέφει τον αριθμό των πιονιών μέσα στη φωλιά μετά την ολοκλήρωση της διαδικασίας.

Αρχικά ελέγχει ένα-ένα τα τετράγωνα και για κάθε ένα στο οποίο υπάρχει πιόνι του ρομπότ ενώ δεν θα έπρεπε, το αφαιρεί. Αν έπρεπε να υπάρχει σε άλλο τετράγωνο πιόνι του ρομπότ και δεν υπάρχει, τότε το τοποθετεί εκεί εκτός και αν υπάρχει σε αυτό κάποιο πιόνι του αντιπάλου. Αν υπάρχει πιόνι του αντιπάλου ή αν δεν λείπει πιόνι του ρομπότ από κάποια άλλη θέση, τότε το τοποθετεί στη φωλιά. Αν η φωλιά είναι γεμάτη, τότε το τοποθετεί έξω από τον πάγκο του παιγνιδιού. Κατόπιν επαναλαμβάνει την ίδια διαδικασία για τα πιόνια του παίκτη, μόνο που εδώ αυτά που πρέπει να απομακρυνθούν από το παιγνίδι δεν τοποθετούνται στη φωλιά, αλλά έξω από τον πάγκο. Μετά από αυτή τη διαδικασία, στον πάγκο του παιγνιδιού δεν υπάρχουν πιόνια σε θέσεις που δεν θα έπρεπε. Κατόπιν τοποθετεί στον πάγκο τα πιόνια του ρομπότ σε θέσεις που θα έπρεπε να υπάρχουν αυτά και λείπουν από εκεί. Όσον αφορά τα πιόνια του παίκτη, επειδή δεν υπάρχει αντίστοιχη φωλιά για τα δικά του πιόνια, τυπώνει μήνυμα στην οθόνη που του ζητάει να τα τοποθετήσει υποδεικνύοντάς του τη σωστή θέση. Ομοίως αν αδειάσει η φωλιά του ρομπότ ζητάει από τον παίκτη να τη γεμίσει και κατόπιν συνεχίζει.

Μετά το τέλος αυτής της διαδικασίας υπάρχουν τα σωστά πιόνια στις σωστές τους θέσεις. Αν όμως αυτό δεν έχει συμβεί, τότε επειδή θα επαναληφθεί ο εσωτερικός βρόγχος που υπάρχει στη main(), θα κληθεί ξανά η detectPawns, με αποτέλεσμα να ξαναγίνει έλεγχος. Έτσι εξασφαλίζεται ότι το παιγνίδι δεν πάει παρακάτω αν αυτό δεν επανέλθει στην προηγούμενη αποδεκτή κατάσταση. Παρακάτω ακολουθεί η συνάρτηση αυτή:

```
  /* This function puts pawns where they were previously.
   * It returns the number of pawns inside the nest.
   */
int restoreBoardFromTo (int fd, Boxes currentBoard, Boxes desiredBoard)
{

  serialSend(fd, X(-2), Y(-2), up);   // Move head a little bit
  Pause(0.4);

      /* First check for Delta Robot's pawns that should be removed from a box */
  for (int i=1; i<10; i++)
  {
```

```
   if ( currentBoard[i]==2 && desiredBoard[i]!=2)
   {     // If there should not be a Delta Robots's pawn then take it from i box and...
      bool foundPlace=false;
      for (int j=1; j<10 && !foundPlace; j++)
      {
         // If there should be a Delta Robot's pawn, then put it there (j box)
         if (currentBoard[j]==0 && desiredBoard[j]==2)
         {
            movePawn(fd,i,j);
            foundPlace=true;
            currentBoard[j]=2;
         }
      }
      if (!foundPlace)    // If there is no place, then put it inside the nest
      {     // currentBoard[0] must be negated to inform function movePawn…
           // …that it should be placed inside the nest
         if (currentBoard[0]<5) movePawn(fd,i,-(++currentBoard[0]));   // And update nest.
         else movePawn(fd,i,0);  // If nest is full, then dump it
      }
      currentBoard[i]=0;
   }
}

   /* Now check for player's pawns that should be removed from a box */
for (int i=1; i<10; i++)
{
   if ( currentBoard[i]==1 && desiredBoard[i]!=1)
   {     // If there should not be a player's pawn then take it from i box and...
      bool foundPlace=false;
      for (int j=1; j<10 && !foundPlace; j++)
      {
         // If there should be a player's pawn, then put it there (j box)
         if (currentBoard[j]==0 && desiredBoard[j]==1)
         {
            movePawn(fd,i,j);
            foundPlace=true;
            currentBoard[j]=1;
         }
      }
      if (!foundPlace) movePawn(fd,i,0);
                        // If there is no place to move the pawn, dump it (move it away)
      currentBoard[i]=0;
   }
}
```

```
    /* Now check for Delta Robot's pawns that should be placed on game Board */
  for (int i=1; i<10; i++)
  {
    if ( currentBoard[i]==0 && desiredBoard[i]==2)
    {     // If there should be a Delta Robots's pawn then take it from nest
      if (currentBoard[0]<=0)
      {
        cout << " \n Nest is empty. Please refill nest.\n" << endl;
        Pause(15.0);    // Wait to refill nest.
        currentBoard[0]=5;
      }
      movePawn(fd,-currentBoard[0]--,i);
      currentBoard[i]=2;
    }
  }

  serialSend(fd, X(0), Y(0), up);     // Move away from camera

    /* Now check for player's pawns that should be placed on gameBoard */
  for (int i=1; i<10; i++)
  {
    if ( currentBoard[i]==0 && desiredBoard[i]==1)
    {
      cout << "\n ATTENTION. Pawns are missing. \n Please put a pawn on the " << i << "th box.\n"
                    << endl;
      Pause (15.0);
      currentBoard[i]=1;
    }
  }

  return currentBoard[0];
}
```

2.13.12 Η συνάρτηση main()

Η main() έχει αναλυθεί σε προηγούμενη παράγραφο που αφορούσε το σκεπτικό του κώδικα. Η main() αρχικά κάνει αρχικοποιήσεις, όπως να τρέξει τον driver modprobe, να ανοίξει την κάμερα και τη σειριακή επικοινωνία και να ελέγξει αν όλα είναι καλά. Κατόπιν επιλέγει με τυχαίο τρόπο ποιος θα παίξει πρώτος και εκτελεί το παιγνίδι όπως έχει περιγραφεί. Για τον τρόπο επιλογής του ποιος παίζει πρώτος, ελέγχει αν ο χρόνος που έχει περάσει από την εκκίνηση του Raspberry Pi σε δευτερόλεπτα είναι άρτιος ή περιττός αριθμός. Παρακάτω ακολουθεί η main():

```
int main(int argc, char** argv)
{

  if (argc==1) executeModprobe(); // If no arguments passed, run the driver for the video camera

  if (argc==2)    // Argument "?" or "H" or "h" for help. The whole word "Help" or "help" is accepted
  {
    if (argv[1][0]=='?'||argv[1][0]=='H'||argv[1][0]=='h')
    {
      printHelpMessage();
      return 0;
    }

    else if (argv[1][0]=='!') // Argument "!" is if the user does NOT want to execute the driver
    {
      cout << " \n Modprobe will NOT be executed.\n";
      cout << " Be sure that it has already been executed either from previous run of this program";
      cout << " or from terminal with command \"sudo modprobe bcm2835-v4l2\".\n" << endl;
    }

    else
    {
      cout<< " \n BAD argument!! Program will terminate." <<endl;
      printHelpMessage();
      return 0;
    }
  }

  if (argc>2)
  {
    cout<< " \n Too many arguments!! Program will terminate." <<endl;
    printHelpMessage();
    return 0;
  }
```

```
VideoCapture camera(0); // open the default camera
if(!camera.isOpened())  // check if we succeeded
{
   cout << " Camera could not be opened!" << endl;
   return -1;
}
   // If frame resolution is not 640x480 pixels then set camera resolution to 640x480
if ( camera.get(CV_CAP_PROP_FRAME_WIDTH) != 640.0 )
                        camera.set(CV_CAP_PROP_FRAME_WIDTH, 640.0);
if ( camera.get(CV_CAP_PROP_FRAME_HEIGHT) != 480.0 )
                        camera.set(CV_CAP_PROP_FRAME_HEIGHT, 480.0);

   // Show windows and a first picture of game board.
for (int i=0; i<10; i++)
{
   Mat temp;
   camera >> temp;
   imshow("Captured Image",temp);
   for (int i=0; i<10; i++) waitKey(1);
}

int fd=serialOpen("/dev/ttyAMA0",9600); // Open the serial port to communicate with Arduino
if (fd<0)
{
   cout << " Serial port could not be opened!" << endl;
   return -1;
}
cout << " Waiting for Delta Robot to connect..." << endl;
serialSend(fd);    // Move Delta Robot's Head to the center of the game board
cout << " Serial Connection is OK." << endl;

cout << "\n Initializing game. Deciding who plays first...\n" << endl;

Boxes gameBoard, lastValidGameBoard; // The object boxes is to store the status of the table

   // Initially there are 5 pawns in the nest. (box[0] is the number of pawns inside the nest)
gameBoard[0]=5;
lastValidGameBoard[0]=5;

   // Decide who starts first by calculating if the elapsed seconds are even or odd.
if ( (int)elapsedTime()%2 )
{
   cout << " I play first..." << endl;
   Pause(1.0);
```

```
      // Move a pawn from nest to the first box and update nest and game board
    movePawn(fd, -5, 1);
    gameBoard[0]=4;
    lastValidGameBoard[0]=4;
    gameBoard[1]=2;
    lastValidGameBoard[1]=2;
}
else cout << " You play first." << endl;

int nextMove=0;     // Delta Robot's next move (number of box from 1 to 9). <0 if it is the last move.
int playerMove=0;   // Player's move (number of box from 1 to 9) or 0 if has not played yet
                    // ...and <0 if player has cheated
bool flag=true;     // It is false when game is over and true when it must continue.

while (flag)
{

    serialSend (fd, X(0), Y(0), up, false);     // Move Delta Robot away from camera's view
    cout << " It is your turn now..." << endl;

    double currentTime=0.0, previousTime=0.0, printMsg=0.0;

    // While there is no change to the table, keep scanning until...
    // ...something is changed. (Player's move)
    // But do not scan continuously. Give a little time between successive scans.
    do
    {

        currentTime = elapsedTime();

        // If system time counter has started all over again, then...
        // ... reset variables "previousTime" and "printMsg"
        if (currentTime < previousTime) previousTime=currentTime;
        if (currentTime < printMsg) printMsg=currentTime;

        /* If some time has passed since previous prompt message...
         * ...then inform player that they must play.
         */
        if (currentTime-printMsg > 4.0)
        {
            cout << " \n Waiting for your move..." << endl;
            printMsg=currentTime;
        }
```

```
   /* If the desired time has passed since previous scan or if it is the first inner do-while loop…
    * …after finishing the outer while loop, then perform board scan and pawn detection.
    */
   if (currentTime-previousTime > 1.0 || previousTime == 0.0)
   {
      previousTime=currentTime;
      detectPawns(camera, gameBoard);
                        // Detect all pawns on game board and store it on gameBoard
   }

   playerMove= gameBoard-lastValidGameBoard;

   //If player has made an invalid move, then…
   // … move pawns to their last valid position and wait for player to play again.
   if (playerMove<0)
   {
      cout <<"\n Invalid move! Restoring game board to previous valid one...\n" << endl;
      gameBoard[0]=restoreBoardFromTo(fd,gameBoard,lastValidGameBoard);
   }

} while (playerMove<=0);

   // Player has made a valid move.

cout << " You moved to box: " << playerMove << endl;
cout << " It is my turn now..." << endl;

Pause(0.3);

   /* Decide Delta Robot's next move and execute.
    * If decideNextMove>0, then move a pawn from nest to the respective box.
    * If <0 then move from nest to the respective box, but this is the last move and
    * Delta Robot wins.
    * if =0 then do not move because player has won.
    */
nextMove=decideNextMove(gameBoard);

if (nextMove > 0)
{

   if (gameBoard[0]<=0)
   {
      cout << "\n Nest is empty. Please refill nest.\n" << endl;
```

```
      Pause(15.0); // Wait to refill nest.
      gameBoard[0]=5;
    }

    cout << " I move to box: " << nextMove << endl;

    /* Move pawn from nest to selected box. gameBoard[0] must be negated in order
     * to be interpreted by movePawn function as a move from nest and not from a box.
     */
    movePawn(fd, -gameBoard[0], nextMove);
    gameBoard[0]--;    // Nest (which is box[0]) has one less pawn now
    gameBoard[nextMove]=2;  // Update gameBoard with the Delta Robot's new move
    lastValidGameBoard=gameBoard;
    lastValidGameBoard[0]=gameBoard[0];
                        // Nest will not be updated by overloaded =. It must be done manually.

          /* If there is no empty box after Robot's move and there is not a winning combination,
           * then it is a tie. If it were a winning combination then nextMove would have been <=0.
           */
    bool empty=false;
    for (int i=1; i<=9; i++) if (gameBoard[i]==0) empty=true;
    if (empty==false)
    {
      flag=false;  // So, game has ended.
      nextMove=-10;   // And it is a tie.
    }

}


    /* If nextMove < 0 then it is the last move and Delta Robot wins.
     *However, it must become positive,
     * otherwise it will be interpreted by movePawn function as move to nest and not to a box.
     */
else if (nextMove<0 && nextMove>-10)
{
    if (gameBoard[0]<=0)
    {
      cout << "\n Nest is empty. Please refill nest.\n" << endl;
      Pause(15.0); // Wait to refill nest.
      gameBoard[0]=5;
    }
    cout << " I move to box: " << -nextMove << endl;
    movePawn(fd, -gameBoard[0], -nextMove);  // Move pawn from nest to selected box
    gameBoard[-nextMove]=2;
```

```
            flag=false;
        }

        else flag=false;    // If nextMove==0 or -10.

    }  // If next move<=0 then Game is Over.

    cout << " \n\tGAME OVER\n";
    if (nextMove==0) cout << "\n CONGRATULATIONS ! YOU WON !\n" << endl;
    else if (nextMove==-10) cout << "\n Nobody won. It is a TIE !\n" << endl;
    else if (nextMove<0) cout << "\n DELTA ROBOT WON !\n" << endl;

    Pause(0.2);
    serialSend(fd); // Park Delta Robot

    serialClose (fd);

    cout << " Thank you for playing. I hope to see you again.\n" <<endl;

        // Show game board after game has ended.
    Mat finalImage;
    for (int i=0; i<6; i++) camera >> finalImage;   // Empty camera buffer and capture the latest image
    for (int i=1; i<=9; i++)
    {
        if (gameBoard[i]==1)    // Draw a green X on the top of Player's pawns.
        {
            Point pix1(XcentralPixel(i)-15,YcentralPixel(i)-15);
            Point pix2(XcentralPixel(i)+15,YcentralPixel(i)+15);
            Point pix3(XcentralPixel(i)-15,YcentralPixel(i)+15);
            Point pix4(XcentralPixel(i)+15,YcentralPixel(i)-15);
            line (finalImage, pix1, pix2, Scalar(0,255,0),3,CV_AA);
            line (finalImage, pix3, pix4, Scalar(0,255,0),3,CV_AA);
        }
        else if (gameBoard[i]==2)   // Draw a red circle with a blue dot on top of Delta Robot's pawns.
        {
            Point center(XcentralPixel(i), YcentralPixel(i));
            circle( finalImage, center, 3, Scalar(255,0,0), -1, CV_AA );
            circle( finalImage, center, 20, Scalar(0,0,255), 3, CV_AA );
        }
    }
    imshow ("Captured Image", finalImage);    // Show captured image with drawn Xs and Os.
    for (int i=1; i<10; i++) waitKey(1);    // This is required to show image.
    Pause(10.0);
    return 0;
}
```

2.14 Ο πηγαίος κώδικας συγκεντρωτικά

Στην παράγραφο αυτή παρουσιάζεται όλος ο πηγαίος κώδικας της εφαρμογής TicTacToe. Υπενθυμίζουμε το hardware και το software που χρησιμοποιήθηκαν. Χρησιμοποιήθηκε η πλακέτα Raspberry Pi 2 Model B, με την έκδοση Jessie του λειτουργικού Raspbian και η δεύτερη έκδοση της επίσημης κάμερας για Raspberry Pi με αισθητήρα της Sony 8MP. Κατόπιν ενεργοποιήθηκε η χρήση της κάμερας και απενεργοποιήθηκε η χρήση του κυκλώματος UART από το τερματικό προκειμένου να είναι διαθέσιμο για την εφαρμογή. Αυτές οι δύο ενέργειες γίνονται τόσο από το «raspi-config», «sudo raspi-config», όσο και από το μενού από το παραθυρικό περιβάλλον. Τέλος έγιναν οι τελευταίες αναβαθμίσεις του firmware και του λειτουργικού που συμπεριλαμβάνουν και την βιβλιοθήκη wiringPi και εγκαταστάθηκε η βιβλιοθήκη OpenCV. Αυτά έγιναν με τις εξής εντολές:

```
sudo rpi-update

sudo apt-get update

sudo apt-get upgrade

sudo reboot

sudo apt-get install libopencv-dev
```

Για την ανάπτυξη της εφαρμογής εγκαταστάθηκε το Geanny που είναι ένα εύχρηστο γραφικό περιβάλλον ανάπτυξης εφαρμογών σε γλώσσα C, C++, Python κ.ά.. Βέβαια αυτό δεν ήταν απαραίτητο άλλα λόγω της πολυπλοκότητας της εφαρμογής κρίθηκε σκόπιμο η χρήση του για ταχύτερη και ευκολότερη εγγραφή και αποσφαλμάτωση του κώδικα. Αυτό εγκαθίσταται με την εντολή:

```
sudo apt-get install geanny
```

Για τη μεταγλώττιση και τη διασύνδεση του κώδικα πρέπει να εκτελεστεί ο g++ με κάποιες παραμέτρους που αφορούν τη χρήση των βιβλιοθηκών wiringPi και OpenCV, δηλαδή η πλήρης εντολή είναι:

```
g++ -Wall -o TicTacToe TicTacToe.cpp $(pkg-config --cflags --libs opencv) -lwiringPi
```

Αν αποφασίσει ο χρήστης να κάνει τη μεταγλώττιση και τη διασύνδεση ξεχωριστά, τότε οι αντίστοιχες εντολές θα είναι:

```
g++ -Wall -c TicTacToe.cpp $(pkg-config --cflags opencv)

g++ -Wall -o TicTacToe TicTacToe.o $(pkg-config --libs opencv) -lwiringPi
```

Αν όλα έχουν πάει καλά τότε θα τρέξει απρόσκοπτα η εφαρμογή με την εντολή:

```
./TicTacToe
```

Παρακάτω ακολουθεί ο πηγαίος κώδικας:

```
#include <iostream>
#include <wiringSerial.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/time.h>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;


   /* This function returns system time in seconds as double precision floating number. */
inline double elapsedTime (void)
{
   struct timeval time;
   gettimeofday(&time,NULL);
   return (double)time.tv_sec+0.000001*(double)time.tv_usec;
}


   /* This function pauses program execution at least for the seconds given as argument.
    *The default is 1 second. Its argument is a double precision floating number. It is not pauses
    *exactly for the seconds given.
    */
void Pause (double pause=1.0)
{
   double init=elapsedTime(), now=init;
   while (now-init < pause)
   {
      now = elapsedTime();
      if (init > now) init=now;
            // If system time counter has started all over again, then reset variable "now"
   }
   return;
}


/* The contents of the nine boxes of the game board will be stored into the Boxes object.
 *For this program Box or Boxes is/are the square(s)
 * where Delta Robot or its opponent place their pawns.
 *If there is a player's pawn inside the i-th box, then box[i] is 1.
 * If there is a Delta Robot's pawn, then box[i] is 2. If it is empty then box[i] is 0.
 * The box[1] represents the upper left box and box[9] represents the lower right box.
```

```
 * Element box[0] is to store the number of pawns inside the nest. Nest is the big rectangle
 *where Delta Robot's own pawns are stored.
 */

class Boxes
{

private:

   int box[10];    // The contents of the boxes. They are private so as to access them by member
                           //... functions which check if accessing inside reserved matrix space

public:

   int clear(int value=0)
   {            // Method to change Boxes contents to any value or to 0 if there is no argument
      for (int i=0; i<10; i++) box[i]=value;
      return value;
   }

   Boxes(int init=0)
   {            // The Constructor writes the initial values to boxes or 0 if there is no argument
      clear(init);
      return;
   }



   int& operator [] (int i)
   {   // Operator [] overload to access private member box[i] checking for valid memory position
      if ( i>=0 && i<10 ) return box[i];
      else
      {
         cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                                          terminate."<<endl;
         exit(1);
      }
   }

   int& operator () (int i, int j)
        // Operator () overload to access box[i] as a two dimensional matrix. (3 rows 3 columns)
       // (1,1) for upper left box and (3,3) for lower right. Nest (box[0]) cannot be accessed.
   {
      if ( i>=1 && i<=3 && j>=1 && j<=3) return box[3*(i-1)+j];
      else
      {
```

```
        cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                                        terminate."<<endl;
        exit(1);
      }
   }

   void operator = (Boxes bx)
   {          // Operator = overload to copy member box[i] contents omitting box[0]
      for (int i=1; i<10; i++) box[i]=bx.box[i];
      return;
   }

   /* Operator == overload checks whether all elements are equal or not. Member box[0] is omitted.
    * If all elements are equal, it returns true, else false.
    */
   bool operator == (Boxes bx)
   {
      for (int i=1; i<10; i++) if (box[i]!=bx.box[i]) return false;
      return true;
   }

   /* This operator - overload checks whether there is ONLY ONE change
     *from 0 to 1 (this->box[i]=1 and bx.box[i]=0) for only one i
     * and for the rest values of i this->box[i]==bx.box[i].
     *In that case it returns the position (i) of the element that has changed.
     * If all elements are equal, it returns 0,
     *whereas in any other case it returns <0 (minus number of changes).
    */
   int operator - (Boxes bx)
   {
      int counter=0;
      int place=0;
      for (int i=1; i<10; i++)
      {
         if (box[i]!=bx.box[i])
         {
            counter++;
            place=i;
         }
      }
      if ( counter==1 && box[place]==1 && bx.box[place]==0 ) return place;
      else return -counter;
   }
};  // End of class Boxes declaration.
```

```
   /* This function returns the X coordinate of a given box. i=1 is for the upper left box
    *and i=9 for the lower right box.
    * The returned coordinate refers to the reference system of the camera,
    * not the Delta Robot and the units are in pixels.
    */
int XcentralPixel(int place)
{
   const int XcentralPix[9]=
   {
      407, 309, 210,
      405, 312, 214,
      405, 312, 217
   };

   if (place>=1 && place<=9) return XcentralPix[place-1];
   else
   {
      cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                              terminate."<<endl;
      exit(1);
   }
}


   /* This function returns the Y coordinate of a given box. i=1 is for the upper left box
    *and i=9 for the lower right box.
    * The returned coordinate refers to the reference system of the camera,
    * not the Delta Robot and the units are in pixels.
    */
int YcentralPixel(int place)
{
   const int YcentralPix[9]=
   {
      422, 419, 417,
      320, 318, 318,
      228, 225, 222
   };
   if (place>=1 && place<=9) return YcentralPix[place-1];
   else
   {
      cout<<"\n ERROR!\nTried to access a non existing matrix element. Program will
                                                terminate."<<endl;
      exit(1);
   }
}
```

```
  /* This function returns the X coordinate of a given box. i=1 is for the upper left box
   * and i=9 for the lower right box.
   * If i is 0 then it returns the X coordinate of the position where the camera
   * has full view of the game board.
   * If i is negative then it returns the X coordinate of the respective nest position. -1 is for
   * the right place and -5 for the left.
   * The returned coordinate refers to the reference system of the Delta Robot,
   * not the camera and the units are centimeters multiplied by 100 (not pixels).
   */
int X (int i)
{
     /* Matrix "Xcenters" stores the X coordinate of the center of each box.
      *For example Xcenters[0] is the X coordinate of the upper left box.
      * The Xcenters[8] is the X coordinate of the lower right box.
      * The Xcenters[9] till [13] is for the 5 places of the nest (big rectangle on the bottom).
      *9 is the left 13 the right one.
      * These coordinates refer to the reference system of the Delta Robot, not the camera
      *and the units are centimeters multiplied by 100 (not pixels).
      */
  const int Xcenters[14]=
  {
    1087, 1159, 1121,
    439, 437, 500,
    -270, -270, -270,
    -874,-871,-874,-859,-843
  };

  const int awayX= -568;  // The X coordinate of the position where the camera has
                          // ...full view of the game board (cm*100)

  if (i>=1 && i<=9) return Xcenters[i-1];
  else if (i<=-1 && i>=-5) return Xcenters[-i+8];
  else if (i==0) return awayX;
  else
  {
    cout << "\n ERROR!\nTried to access a non existing matrix element. Program will terminate."
                    <<endl;
    exit(1);
  }
}

  /* This function returns the Y coordinate of a given box. i=1 is for the upper left box
   * and i=9 for the lower right box.
   * If i is 0 then it returns the Y coordinate of the position where the camera
   *has full view of the game board.
```

```
    * If i is negative then it returns the Y coordinate of the respective nest position.
    * -1 is for the right place and -5 for the left.
    * The returned coordinate refers to the reference system of the Delta Robot,
    * not the camera and the units are centimeters multiplied by 100 (not pixels).
    */
int Y (int i)
{
     /* Matrix "Ycenters" stores the Y coordinate of the center of each box.
      *For example Ycenters[0] is the Y coordinate of the upper left box.
      * The Ycenters[8] is the Y coordinate of the lower right box.
      * The Ycenters[9] till [13] is for the 5 places of the nest (big rectangle on the bottom).
      *9 is the left 13 the right one.
      * These coordinates refer to the reference system of the Delta Robot, not the camera
      *and the units are centimeters multiplied by 100 (not pixels).
      */
  const int Ycenters[14]=
  {
    578, -30, -609,
    611, -1, -653,
    624, 9, -547,
    645, 300, 17, -286, -669
  };

  const int awayY= 1011;  // The Y coordinate of the position where the camera has full view
                          // ... of the game board (cm*100)

  if (i>=1 && i<=9) return Ycenters[i-1];
  else if (i<=-1 && i>=-5) return Ycenters[-i+8];
  else if (i==0) return awayY;
  else
  {
    cout << "\n ERROR!\nTried to access a non existing matrix element. Program will terminate."
         <<endl;
    exit(1);
  }
}

const int up = -3122;  /* Z coordinate at a high level, much higher than game board level.
                        * It refers to the reference system of the Delta Robot,
                        * not the camera and the units are centimeters multiplied by 100 (not pixels).
                        */


  /* This function returns the Z coordinate of a given box close to pawn level.
   *i=1 is for the upper left box and i=9 for the lower right box.
```

```
    * If i is negative then it returns the Z coordinate of the respective nest position at pawn level.
    *-1 is for the right place and -5 for the left.
    * The returned coordinate refers to the reference system of the Delta Robot,
    * not the camera and the units are centimeters multiplied by 100 (not pixels).
    */
int down (int i)
{
     /* Matrix "ZcloseToPawns" stores the Z coordinate of pawn level for each box.
      *For example ZcloseToPawns[0] is the Z coordinate for the upper left box.
      * The ZcloseToPawns[8] is the Z coordinate for the lower right box.
      * The ZcloseToPawns[9] till [13] is for the 5 places of the nest (big rectangle on the bottom).
      *9 is the left 13 the right one.
      * These coordinates refer to the reference system of the Delta Robot, not the camera
      *and the units are centimeters multiplied by 100 (not pixels).
      */
  const int ZcloseToPawns[14]=
  {
    -3371, -3372, -3350,
    -3399, -3405, -3427,
    -3423, -3427, -3446,
    -3450, -3450, -3450, -3450, -3450
  };

  if (i>=1 && i<=9) return ZcloseToPawns[i-1];
  else if (i<=-1 && i>=-5) return ZcloseToPawns[-i+8];
  else
  {
    cout << "\n ERROR!\nTried to access a non existing matrix element. Program will terminate."
         <<endl;
    exit(1);
  }
}

int executeModprobe(void)   // Executes the sudo modprobe bcm2835-v4l2 command
{
  int status=1;

  if (fork()==0)
  {
    char executable[] = "/usr/bin/sudo";
    char argument1[] = "modprobe";
    char argument2[] = "bcm2835-v4l2";
    char *arguments[] = { executable, argument1, argument2, NULL };
    execve(executable, arguments, NULL);
    perror("execve");   // Execve never returns. If it does, then an error may have occurred.
```

```
        exit(1);
    }

    wait(&status);

    /* Parent is waiting for driver to be executed before going further.
     *In normal cases child returns 0. Otherwise an error has occurred.
     * However the driver may have been already loaded from a previous run or from bash by the user.
     *For that reason this program will NOT stop as the video driver may be running properly.
     */

    if (status!=0)
    {
        cout << "\n Modprobe was NOT executed properly. If any problems occur,
                                then press Ctrl+C and run this program again.\n";
        cout << " If problems insist, then execute from terminal the command
                \"sudo modprobe bcm2835-v4l2\" and then run this program with argument \"!\".\n\n";
    }

    return status;  // Returns 0 if everything is OK
}


void printHelpMessage(void)
{
    cout << "\n  Help:" << endl;
    cout << " In normal cases no arguments are needed to execute this program." << endl;
    cout << " Use argument \"?\" or \"H\" or \"h\" for help."<<endl;
    cout << " Use argument \"!\" if you do NOT want to execute modprobe driver.\n";
    cout << "\n   Caution!!"<<endl;
    cout << " In that case be sure that you have already executed from terminal the command
                                \"sudo modprobe bcm2835-v4l2\" otherwise ";
    cout << "this program will not run properly.\n This option should only be used if there are
                                problems in running modprobe from this program.\n"<<endl;
}


void detectPawns (VideoCapture& cameraFrame, Boxes &Board)
{               // Detects and Identifies which pawn is in which box.

    const int lowHueRobot=113;
    const int highHueRobot=204;
    const int lowSatRobot=108;
    const int highSatRobot=223;
    const int lowValueRobot=81;
```

```
const int highValueRobot=202;

const int lowHuePlayer=21;
const int highHuePlayer=65;
const int lowSatPlayer=93;
const int highSatPlayer=240;
const int lowValuePlayer=69;
const int highValuePlayer=201;

double previousTime=0.0, currentTime=0.0;

bool flag=false;

Mat originalImage, hsvImage, DeltaRobotPawns, playerPawns;

Boxes temp1, temp2(2), temp3(3);
                        // Three temporary Boxes objects to store pawn location on the game board

while (!flag)
{
   currentTime=elapsedTime();

      // If system time counter has started all over again, then reset variable "previousTime"
   if (currentTime < previousTime) previousTime=currentTime;

   if ( currentTime-previousTime > 1.0 || previousTime==0.0)
            // Leave a time interval between successive frames.
   {
      previousTime=currentTime;

      temp3=temp2;
      temp2=temp1;
      temp1.clear();  // Initially every element of temp1 becomes zero

      for (int i=0; i<6; i++) cameraFrame >> originalImage;
            // Empty camera buffer, capture the latest frame and copy it to an object of Mat type

      cvtColor(originalImage, hsvImage, COLOR_BGR2HSV);
            // Transform the BGR image to HSV

        // Transform HSV color image in black and white according to the given HSV values
      inRange(hsvImage, Scalar(lowHuePlayer,lowSatPlayer,lowValuePlayer),
                        Scalar(highHuePlayer,highSatPlayer,highValuePlayer), playerPawns);
      inRange(hsvImage, Scalar(lowHueRobot,lowSatRobot,lowValueRobot),
                        Scalar(highHueRobot,highSatRobot,highValueRobot), DeltaRobotPawns);
```

```
   // Apply Gaussian Blur filter and then detect any pawns.
GaussianBlur( playerPawns, playerPawns, Size(9, 9), 2.0, 2.0 );
GaussianBlur( DeltaRobotPawns, DeltaRobotPawns, Size(9, 9), 2.0, 2.0 );


for( int i = 1; i <= 9; i++ )
{     // In the Player pawns matrix, for each white pixel found inside a
      // …square of 20x20 pixels surrounding the center of i-th box, then counterPlayer++

   int counterPlayer=0;
   for (int j=-10; j<=10; j++)
   {
      for (int k=-10; k<=10; k++)
      {
         if (playerPawns.at<uchar>(YcentralPixel(i)+j,XcentralPixel(i)+k)) counterPlayer++;
      }

   }
   if (counterPlayer>=300)
   {                  // If there are more than 300 white pixels then there is a pawn in the i-th box
      temp1[i]=1;
      Point pix1(XcentralPixel(i)-15,YcentralPixel(i)-15);
      Point pix2(XcentralPixel(i)+15,YcentralPixel(i)+15);
      Point pix3(XcentralPixel(i)-15,YcentralPixel(i)+15);
      Point pix4(XcentralPixel(i)+15,YcentralPixel(i)-15);
      line (originalImage, pix1, pix2, Scalar(0,255,0),3,CV_AA);          // Draws a green X
      line (originalImage, pix3, pix4, Scalar(0,255,0),3,CV_AA);
   }

}

for( int i = 1; i <= 9; i++ )
{     // In the Delta Robot pawns matrix, for each white pixel found inside a
      // …square of 20x20 pixels surrounding the center of i-th box, then counterRobot++

   int counterRobot=0;
   for (int j=-10; j<=10; j++)

   {
      for (int k=-10; k<=10; k++)
      {
         if (DeltaRobotPawns.at<uchar>(YcentralPixel(i)+j,XcentralPixel(i)+k)) counterRobot++;
      }

   }
```

```
            if (counterRobot>=300)
            {                   // If there are more than 300 white pixels then there is a pawn in the i-th box
               temp1[i]=2;
               Point center(XcentralPixel(i), YcentralPixel(i));
               circle( originalImage, center, 3, Scalar(255,0,0), -1, CV_AA );   // Draws a blue dot
               circle( originalImage, center, 20, Scalar(0,0,255), 3, CV_AA );   // Draws a red O
            }

         }


           // Show image
         imshow ("Captured Image", originalImage);  // Show image on the window.
         for (int i=0; i<10; i++) waitKey(1);    // This is required to show image.

         // Keep scanning until you find three successive frames with identical Boxes elements
         flag = temp1==temp2 && temp2==temp3;

      }  // Endif


   }  // End while loop

   Board=temp1;    // Return a Boxes object with pawn location on the game board
                   // Nest (box[0]) will not be affected.

   return;
}



   /* Decides Delta Robot's next move and returns the number of the box.
    *If player has won it returns 0.
    * If it is the last move of Delta Robot and robot wins, then it returns a negative number which is
    * the place of the next box negated. If there is no box empty and there is no winning combination,
    * then nobody wins (tie) and returns -10.
    */
int decideNextMove ( Boxes Board )
{

   /* If three player's pawns are diagonal or three in a raw or in a column, then player wins. */
   if (Board(1,1)==1 && Board(2,2)==1 && Board(3,3)==1) return 0;
   if (Board(1,3)==1 && Board(2,2)==1 && Board(3,1)==1) return 0;
   for (int i=1; i<=3; i++)
   {
```

```
   if (Board(i,1)==1 && Board(i,2)==1 && Board(i,3)==1) return 0;
   if (Board(1,i)==1 && Board(2,i)==1 && Board(3,i)==1) return 0;
}


/* If there are two Delta Robot's pawns in a row or column or diagonal,
 * then place a third pawn in the third place, if empty. Return negative because Robot wins.
 */
for (int i=1; i<=3; i++)
{
   if (Board(i,1)==0 && Board(i,2)==2 && Board(i,3)==2) return 3*(1-i)-1;
   if (Board(i,1)==2 && Board(i,2)==0 && Board(i,3)==2) return 3*(1-i)-2;
   if (Board(i,1)==2 && Board(i,2)==2 && Board(i,3)==0) return 3*(1-i)-3;
   if (Board(1,i)==0 && Board(2,i)==2 && Board(3,i)==2) return -i;
   if (Board(1,i)==2 && Board(2,i)==0 && Board(3,i)==2) return -3-i;
   if (Board(1,i)==2 && Board(2,i)==2 && Board(3,i)==0) return -6-i;
}

if (Board(1,1)==0 && Board(2,2)==2 && Board(3,3)==2) return -1;
if (Board(1,1)==2 && Board(2,2)==0 && Board(3,3)==2) return -5;
if (Board(1,1)==2 && Board(2,2)==2 && Board(3,3)==0) return -9;
if (Board(3,1)==0 && Board(2,2)==2 && Board(1,3)==2) return -7;
if (Board(3,1)==2 && Board(2,2)==0 && Board(1,3)==2) return -5;
if (Board(3,1)==2 && Board(2,2)==2 && Board(1,3)==0) return -3;

   /* If there are two player's pawns in a row or column or diagonal,
    * then place a pawn in the third place, if empty, to avoid losing from player.
    */
for (int i=1; i<=3; i++)
{
   if (Board(i,1)==0 && Board(i,2)==1 && Board(i,3)==1) return 3*(i-1)+1;
   if (Board(i,1)==1 && Board(i,2)==0 && Board(i,3)==1) return 3*(i-1)+2;
   if (Board(i,1)==1 && Board(i,2)==1 && Board(i,3)==0) return 3*(i-1)+3;
   if (Board(1,i)==0 && Board(2,i)==1 && Board(3,i)==1) return i;
   if (Board(1,i)==1 && Board(2,i)==0 && Board(3,i)==1) return 3+i;
   if (Board(1,i)==1 && Board(2,i)==1 && Board(3,i)==0) return 6+i;
}

if (Board(1,1)==0 && Board(2,2)==1 && Board(3,3)==1) return 1;
if (Board(1,1)==1 && Board(2,2)==0 && Board(3,3)==1) return 5;
if (Board(1,1)==1 && Board(2,2)==1 && Board(3,3)==0) return 9;
if (Board(3,1)==0 && Board(2,2)==1 && Board(1,3)==1) return 7;
if (Board(3,1)==1 && Board(2,2)==0 && Board(1,3)==1) return 5;
if (Board(3,1)==1 && Board(2,2)==1 && Board(1,3)==0) return 3;
```

```
    /* In any other case, put a pawn to a corner, if empty.
     * If not, put it somewhere inside.
     */
  if (Board[1]==0) return 1;
  if (Board[3]==0) return 3;
  if (Board[7]==0) return 7;
  if (Board[9]==0) return 9;
  for (int i=2; i<=8; i+=2) if (Board[i]==0) return i;
  if (Board[5]==0) return 5;

  return -10;    // There is no empty space or winning combination.
}



  /* Function serialSend sends to the Delta Robot the coordinates and electromagnet
   * status via serial port (UART).
   * Numbers must be integers in centimeters multiplied by 100.
   * For example for (x,y,z)=(12.32,-6.78,-34.09) the numbers must be
   * (x,y,z)=(1232,-678,-3409)
   */
void serialSend ( int fd, int x=0, int y=0, int z=-3100, bool mag=false )
{
  double currentTime, oldTime=elapsedTime();

  while(1)
  {
    if ( serialDataAvail(fd) )  // If Delta Robot has sent a '@' then transmit coordinates. Else loop.
    {
      if ( serialGetchar(fd) == '@' )
      {
        serialFlush (fd);
        serialPutchar (fd,'@');
        serialPrintf (fd,"X= %d\nY= %d\nZ= %d\nElectromagnet= %d\n",x,y,z,(int)mag);
        serialPutchar (fd,'$');
        Pause(0.4);  // Wait a little bit for Delta Robot to execute command.
        return;
      }
    }

    currentTime=elapsedTime();
    if ( currentTime<oldTime ) oldTime=currentTime;
          // If system time counter has started all over again, then reset variable "oldTime"
    if ( currentTime-oldTime > 5.0 )
    {
```

```
            cout << "\n\n Warning!!\n Delta Robot is not responsive for too long." << endl;
            cout << " Check connection and press the black button on the Arduino Shield." << endl;
            oldTime=currentTime;
        }
    }
}



    /* This function moves a pawn from a box to another box. Box is a number between 1 to 9.
     * 1 is for upper left and 9 for lower right. If it is negative then it implies the nest.
     * -5 is the right place and -1 for the left. If it is zero then it means far away.
     */
void movePawn(int fd, int from, int to)
{
    const double pause = 0.4;

    serialSend (fd, X(from), Y(from), up, false);   // Go on top of the pawn
    Pause(pause);
    serialSend (fd, X(from), Y(from), down(from), true);    // And grab it (turn magnet on)
    Pause(pause);
    serialSend (fd, X(from), Y(from), up, true);

    serialSend (fd, X(to), Y(to), up, true);    // Then go to destination
    Pause(pause);
    serialSend (fd, X(to), Y(to), up, false);   // And release the pawn (turn magnet off)

    Pause(pause);

    return;
}



    /* This function puts pawns where they were previously.
     * It returns the number of pawns inside the nest.
     */
int restoreBoardFromTo (int fd, Boxes currentBoard, Boxes desiredBoard)
{

    serialSend(fd, X(-2), Y(-2), up);   // Move head a little bit
    Pause(0.4);

        /* First check for Delta Robot's pawns that should be removed from a box */
    for (int i=1; i<10; i++)
```

```
{
   if ( currentBoard[i]==2 && desiredBoard[i]!=2)
   {     // If there should not be a Delta Robots's pawn then take it from i box and...
      bool foundPlace=false;
      for (int j=1; j<10 && !foundPlace; j++)
      {
         // If there should be a Delta Robot's pawn, then put it there (j box)
         if (currentBoard[j]==0 && desiredBoard[j]==2)
         {
            movePawn(fd,i,j);
            foundPlace=true;
            currentBoard[j]=2;
         }
      }
      if (!foundPlace)    // If there is no place, then put it inside the nest
      {     // currentBoard[0] must be negated to inform function movePawn...
           // ...that it should be placed inside the nest
         if (currentBoard[0]<5) movePawn(fd,i,-(++currentBoard[0]));   // And update nest.
         else movePawn(fd,i,0);  // If nest is full, then dump it
      }
      currentBoard[i]=0;
   }
}

      /* Now check for player's pawns that should be removed from a box */
for (int i=1; i<10; i++)
{
   if ( currentBoard[i]==1 && desiredBoard[i]!=1)
   {     // If there should not be a player's pawn then take it from i box and...
      bool foundPlace=false;
      for (int j=1; j<10 && !foundPlace; j++)
      {
         // If there should be a player's pawn, then put it there (j box)
         if (currentBoard[j]==0 && desiredBoard[j]==1)
         {
            movePawn(fd,i,j);
            foundPlace=true;
            currentBoard[j]=1;
         }
      }
      if (!foundPlace) movePawn(fd,i,0);
                        // If there is no place to move the pawn, dump it (move it away)
      currentBoard[i]=0;
   }
}
```

```
    /* Now check for Delta Robot's pawns that should be placed on game Board */
  for (int i=1; i<10; i++)
  {
    if ( currentBoard[i]==0 && desiredBoard[i]==2)
    {     // If there should be a Delta Robots's pawn then take it from nest
      if (currentBoard[0]<=0)
      {
        cout << " \n Nest is empty. Please refill nest.\n" << endl;
        Pause(15.0);    // Wait to refill nest.
        currentBoard[0]=5;
      }
      movePawn(fd,-currentBoard[0]--,i);
      currentBoard[i]=2;
    }
  }

  serialSend(fd, X(0), Y(0), up);    // Move away from camera


    /* Now check for player's pawns that should be placed on gameBoard */
  for (int i=1; i<10; i++)
  {
    if ( currentBoard[i]==0 && desiredBoard[i]==1)
    {
      cout << "\n ATTENTION. Pawns are missing. \n Please put a pawn on the " << i << "th box.\n"
                    << endl;
      Pause (15.0);
      currentBoard[i]=1;
    }
  }

  return currentBoard[0];
}



int main(int argc, char** argv)
{

  if (argc==1) executeModprobe(); // If no arguments passed, run the driver for the video camera

  if (argc==2)    // Argument "?" or "H" or "h" for help. The whole word "Help" or "help" is accepted
  {
    if (argv[1][0]=='?'||argv[1][0]=='H'||argv[1][0]=='h')
    {
```

```
      printHelpMessage();
      return 0;
   }

   else if (argv[1][0]=='!') // Argument "!" is if the user does NOT want to execute the driver
   {
      cout << " \n Modprobe will NOT be executed.\n";
      cout << " Be sure that it has already been executed either from previous run of this program";
      cout << " or from terminal with command \"sudo modprobe bcm2835-v4l2\".\n" << endl;
   }

   else
   {
      cout<< " \n BAD argument!! Program will terminate." <<endl;
      printHelpMessage();
      return 0;
   }
}

if (argc>2)
{
   cout<< " \n Too many arguments!! Program will terminate." <<endl;
   printHelpMessage();
   return 0;
}


VideoCapture camera(0); // open the default camera
if(!camera.isOpened())  // check if we succeeded
{
   cout << " Camera could not be opened!" << endl;
   return -1;
}
   // If frame resolution is not 640x480 pixels then set camera resolution to 640x480
if ( camera.get(CV_CAP_PROP_FRAME_WIDTH) != 640.0 )
                        camera.set(CV_CAP_PROP_FRAME_WIDTH, 640.0);
if ( camera.get(CV_CAP_PROP_FRAME_HEIGHT) != 480.0 )
                        camera.set(CV_CAP_PROP_FRAME_HEIGHT, 480.0);

   // Show windows and a first picture of game board.
for (int i=0; i<10; i++)
{
   Mat temp;
   camera >> temp;
   imshow("Captured Image",temp);
```

```
   for (int i=0; i<10; i++) waitKey(1);
}


int fd=serialOpen("/dev/ttyAMA0",9600); // Open the serial port to communicate with Arduino
if (fd<0)
{
   cout << " Serial port could not be opened!" << endl;
   return -1;
}
cout << " Waiting for Delta Robot to connect..." << endl;
serialSend(fd);    // Move Delta Robot's Head to the center of the game board
cout << " Serial Connection is OK." << endl;


cout << "\n Initializing game. Deciding who plays first...\n" << endl;


Boxes gameBoard, lastValidGameBoard; // The object boxes is to store the status of the table

   // Initially there are 5 pawns in the nest. (box[0] is the number of pawns inside the nest)
gameBoard[0]=5;
lastValidGameBoard[0]=5;


   // Decide who starts first by calculating if the elapsed seconds are even or odd.
if ( (int)elapsedTime()%2 )
{
   cout << " I play first..." << endl;
   Pause(1.0);


      // Move a pawn from nest to the first box and update nest and game board
   movePawn(fd, -5, 1);
   gameBoard[0]=4;
   lastValidGameBoard[0]=4;
   gameBoard[1]=2;
   lastValidGameBoard[1]=2;
}
else cout << " You play first." << endl;


int nextMove=0;    // Delta Robot's next move (number of box from 1 to 9). <0 if it is the last move.
int playerMove=0;  // Players move (number of box from 1 to 9) or 0 if has not played yet
                   // ...and <0 if player has cheated
```

```
bool flag=true;    // It is false when game is over and true when it must continue.

while (flag)
{

   serialSend (fd, X(0), Y(0), up, false);    // Move Delta Robot away from camera's view
   cout << " It is your turn now..." << endl;

   double currentTime=0.0, previousTime=0.0, printMsg=0.0;

   // While there is no change to the table, keep scanning until…
   // …something is changed. (Player's move)
   // But do not scan continuously. Give a little time between successive scans.
   do
   {

      currentTime = elapsedTime();

      // If system time counter has started all over again, then…
      // … reset variables "previousTime" and "printMsg"
      if (currentTime < previousTime) previousTime=currentTime;
      if (currentTime < printMsg) printMsg=currentTime;


      /* If some time has passed since previous prompt message…
       * …then inform player that they must play.
       */
      if (currentTime-printMsg > 4.0)
      {
         cout << " \n Waiting for your move..." << endl;
         printMsg=currentTime;
      }


      /* If the desired time has passed since previous scan or if it is the first inner do-while loop…
       * …after finishing the outer while loop, then perform board scan and pawn detection.
       */
      if (currentTime-previousTime > 1.0 || previousTime == 0.0)
      {
         previousTime=currentTime;
         detectPawns(camera, gameBoard);
                        // Detect all pawns on game board and store it on gameBoard
      }
      playerMove= gameBoard-lastValidGameBoard;
```

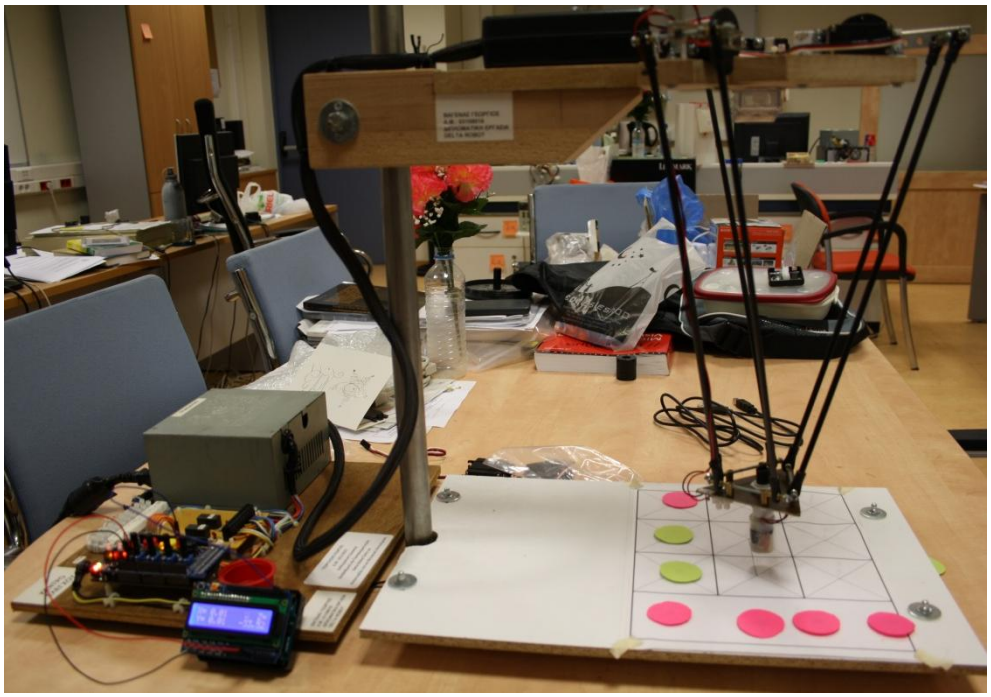
```
   //If player has made an invalid move, then…
   // … move pawns to their last valid position and wait for player to play again.
   if (playerMove<0)
   {
      cout <<"\n Invalid move! Restoring game board to previous valid one...\n" << endl;
      gameBoard[0]=restoreBoardFromTo(fd,gameBoard,lastValidGameBoard);
   }

} while (playerMove<=0);


   // Player has made a valid move.

cout << " You moved to box: " << playerMove << endl;
cout << " It is my turn now..." << endl;

Pause(0.3);

   /* Decide Delta Robot's next move and execute.
    * If decideNextMove>0, then move a pawn from nest to the respective box.
    * If <0 then move from nest to the respective box, but this is the last move and
    * Delta Robot wins.
    * if =0 then do not move because player has won.
    */
nextMove=decideNextMove(gameBoard);

if (nextMove > 0)
{

   if (gameBoard[0]<=0)
   {
      cout << "\n Nest is empty. Please refill nest.\n" << endl;
      Pause(15.0); // Wait to refill nest.
      gameBoard[0]=5;
   }

   cout << " I move to box: " << nextMove << endl;

   /* Move pawn from nest to selected box. gameBoard[0] must be negated in order
    * to be interpreted by movePawn function as a move from nest and not from a box.
    */
   movePawn(fd, -gameBoard[0], nextMove);
   gameBoard[0]--;    // Nest (which is box[0]) has one less pawn now
   gameBoard[nextMove]=2;  // Update gameBoard with the Delta Robot's new move
   lastValidGameBoard=gameBoard;
```

```
      lastValidGameBoard[0]=gameBoard[0];
                        // Nest will not be updated by overloaded =. It must be done manually.


          /* If there is no empty box after Robot's move and there is not a winning combination,
           * then it is a tie. If it were a winning combination then nextMove would have been <=0.
           */
      bool empty=false;
      for (int i=1; i<=9; i++) if (gameBoard[i]==0) empty=true;
      if (empty==false)
      {
        flag=false;  // So, game has ended.
        nextMove=-10;   // And it is a tie.
      }

    }

      /* If nextMove < 0 then it is the last move and Delta Robot wins.
       *However, it must become positive,
       * otherwise it will be interpreted by movePawn function as move to nest and not to a box.
       */
    else if (nextMove<0 && nextMove>-10)
    {
      if (gameBoard[0]<=0)
      {
        cout << "\n Nest is empty. Please refill nest.\n" << endl;
        Pause(15.0); // Wait to refill nest.
        gameBoard[0]=5;
      }
      cout << " I move to box: " << -nextMove << endl;
      movePawn(fd, -gameBoard[0], -nextMove);  // Move pawn from nest to selected box
      gameBoard[-nextMove]=2;
      flag=false;
    }

    else flag=false;    // If nextMove==0 or -10.

  }  // If next move<=0 then Game is Over.

  cout << " \n\tGAME OVER\n";
  if (nextMove==0) cout << "\n CONGRATULATIONS ! YOU WON !\n" << endl;
  else if (nextMove==-10) cout << "\n Nobody won. It is a TIE !\n" << endl;
  else if (nextMove<0) cout << "\n DELTA ROBOT WON !\n" << endl;

  Pause(0.2);
```

```
    serialSend(fd); // Park Delta Robot

    serialClose (fd);

    cout << " Thank you for playing. I hope to see you again.\n" <<endl;

      // Show game board after game has ended.
    Mat finalImage;
    for (int i=0; i<6; i++) camera >> finalImage;   // Empty camera buffer and capture the latest image
    for (int i=1; i<=9; i++)
    {
      if (gameBoard[i]==1)    // Draw a green X on the top of Player's pawns.
      {
        Point pix1(XcentralPixel(i)-15,YcentralPixel(i)-15);
        Point pix2(XcentralPixel(i)+15,YcentralPixel(i)+15);
        Point pix3(XcentralPixel(i)-15,YcentralPixel(i)+15);
        Point pix4(XcentralPixel(i)+15,YcentralPixel(i)-15);
        line (finalImage, pix1, pix2, Scalar(0,255,0),3,CV_AA);
        line (finalImage, pix3, pix4, Scalar(0,255,0),3,CV_AA);
      }
      else if (gameBoard[i]==2)   // Draw a red circle with a blue dot on top of Delta Robot's pawns.
      {
        Point center(XcentralPixel(i), YcentralPixel(i));
        circle( finalImage, center, 3, Scalar(255,0,0), -1, CV_AA );
        circle( finalImage, center, 20, Scalar(0,0,255), 3, CV_AA );
      }
    }
    imshow ("Captured Image", finalImage);    // Show captured image with drawn Xs and Os.
    for (int i=1; i<10; i++) waitKey(1);    // It is required to show image.

    Pause(10.0);

    return 0;
}
```

Κεφάλαιο 3

Τελικός Απολογισμός



Κεφάλαιο 3. Τελικός Απολογισμός.

Μετά το τέλος αυτής της διπλωματικής εργασίας και σαν αποτέλεσμα αυτής, αυτό που μένει στο εργαστήριο είναι ο κώδικας που έτρεχε στο Raspberry Pi, που αφορά το κυρίως κομμάτι αυτής της εργασίας, καθώς και ένα πλήρως λειτουργικό Delta Robot με τον κώδικά του. Αυτό μπορεί να δέχεται εντολές είτε τοπικά από τα χειριστήριά του είτε απομακρυσμένα από μία άλλη συσκευή με τον τρόπο που περιγράφτηκε. Συνεπώς, μέσα στα πλαίσια των δυνατοτήτων του είναι ικανό να υλοποιήσει μία ευρεία ποικιλία εφαρμογών χωρίς να γίνουν τροποποιήσεις.

Όσον αφορά την εφαρμογή της τρίλιζας, η δομή του κώδικά της είναι τέτοια που επιτρέπει τη δυνατότητα αναβάθμισής του για πραγματοποίηση παιγνιδιών όπως η ντάμα, όπου επίσης έχουμε πιόνια δύο διαφορετικών χρωμάτων, αλλά τα τετράγωνα είναι 64, 8 επί 8, αντί 9. Οπότε, με τη τροποποίηση κάποιων πινάκων και την αλλαγή ή την προσθαφαίρεση κάποιων συναρτήσεων θα μπορούσε να παίξει ντάμα. Μία άλλη πιο περίπλοκη μελλοντική εφαρμογή είναι το σκάκι όπου θα απαιτούνταν πιο πολύπλοκος κώδικας αναγνώρισης των πιονιών και διεξαγωγής του παιγνιδιού.

Αυτή η εργασία είχε σκοπό να κάνει το Delta Robot να αναγνωρίζει αντικείμενα και να τα μετακινεί. Έτσι αυτό αλληλεπιδρά με τον άνθρωπο και εν γένει με το εξωτερικό του περιβάλλον δεχόμενο ερεθίσματα από αυτό και αντιδρώντας σε αυτά με προκαθορισμένο τρόπο. Η δυνατότητα να παίζει τρίλιζα είναι στην ουσία μία πρακτική έκφανση αυτής της γενικότερης ιδέας. Η αλληλεπίδραση με το περιβάλλον είναι το πλαίσιο πάνω στο οποίο στηρίζονται πολλά τεχνολογικά επιτεύγματα της σύγχρονης εποχής.

Μια πιο ενδιαφέρουσα περίπτωση είναι αν μια μηχανή μπορεί να αντιδρά στα εξωτερικά ερεθίσματα με ΜΗ προκαθορισμένο τρόπο. Αυτό, υπό προϋποθέσεις, θα μπορούσε να θέσει βαθύτερα ερωτήματα που αφορούν την έννοια της ζωής και της ύπαρξης γενικότερα (George Cambourakis).

Βιβλιογραφία

George Cambourakis, Another Paradigm for Existence and Life, EANA16 Astrobiology International Conference, Athens, Greece, 2016.

Stephen Umans, Fitzgerald & Kingsley's Electric Machinery, Seventh Edition, McGraw-Hill International Edition, 2014.

Wayne Wolf, Οι υπολογιστές ως Συστατικά Στοιχεία, Εκδόσεις Νέων Τεχνολογιών, 2008.

Γεώργιος Βαγενάς, Κατασκευή Delta Robot, Διπλωματική Εργασία, ΕΜΠ, 2014.

Μαρία Γ. Ιωαννίδου, Ηλεκτρικά Συστήματα Κίνησης: Γ. Συστήματα Ειδικών Κινητήρων, Εκδόσεις Ιωαννίδου Μ.-Π., 2013.

Νίκος Μ. Χατζηγιαννάκης, Η Γλώσσα C++ σε Βάθος, Εκδόσεις Κλειδάριθμος, 2014.

Πηγές από το διαδίκτυο

www.arduino.cc

www.raspberrypi.org

opencv.org

http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/