ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΩΝ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΙΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Αριθμός 1289

Σύστημα κίνησης κεραίας WiFi σε δύο άξονες και αυτόματη εύρεση βέλτιστης θέσης με βάση την ισχύ σήματος ή τις γεωγραφικές συντεταγμένες δύο σημείων.

ΣΠΟΥΔΑΣΤΗΣ:
ΦΟΙΒΟΣ – ΕΥΑΓΓΕΛΟΣ ΓΙΟΥΡΓΑΣ

ΕΙΣΗΓΗΤΗΣ:
Δρ. ΕΥΑΓΓΕΛΟΣ ΤΟΠΑΛΗΣ

ΠΑΡΓΑ ΝΟΕΜΒΡΙΟΣ 2012

Πρόλογος

Η παρούσα πτυχιακή εργασία εκπονήθηκε στα πλαίσια της ολοκλήρωσης των σπουδών μου στο τμήμα Ηλεκτρολογίας της Σχολής Τεχνολογικών Εφαρμογών του ΤΕΙ Πατρών.

Με την ακόλουθη πτυχιακή εργασία, όπου μελετάται αναλυτικά και κατασκευάζεται ένα σύστημα αυτόματης στόχευσης μίας κατευθυντικής κεραίας WiFi με την χρήση προγραμματιζόμενου μικροελεγκτή, διεύρυνα τις γνώσεις μου στον τομέα των μικροελεγκτών, των κεραιών και των ασύρματων δικτύων. Τομείς με έντονο ερευνητικό ενδιαφέρον και άπειρες πρακτικές εφαρμογές.

Κλείνοντας θα ήθελα να ευχαριστήσω θερμά τον εισηγητή μου κο. Ευάγγελο Τοπάλη για την επίβλεψη, και την καθοδήγηση του. Την κοπέλα μου Μαίρη για την υπομονή, και την πολύτιμη βοήθεια της καθ' όλη την διάρκεια της εργασίας αυτής. Τέλος ευχαριστώ όλους όσους συνέβαλλαν στον εμπλουτισμό των γνώσεών μου κατά τη διάρκεια της φοιτητικής μου πορείας και όσους στάθηκαν δίπλα μου τα χρόνια αυτά.

Παργα Φεβρουάριος 2013

Φοίβος-Ευάγγελος Γιούργας

"Within a few years a simple and inexpensive device, readily carried about, will enable one to receive on land or sea the principal news, to hear a speech, a lecture, a song or play of a musical instrument, conveyed from any other region of the globe. The invention will also meet the crying need for cheap transmission to great distances, more especially over the oceans. The small working capacity of the cables and the excessive cost of messages are now fatal impediments in the dissemination of intelligence which can only be removed by transmission without wires."

Nikola Tesla 1905

ΠΕΡΙΛΗΨΗ

Αντικείμενο της εργασίας αυτής είναι η κατασκευή ενός συστήματος που θα κινεί μία κατευθυντική κεραία Wi-Fi (τύπου πάνελ, grid, yagi κτλ) σε δύο άξονες και θα είναι σε θέση να προσδιορίζει την βέλτιστη θέση της κεραίας με βάση την ισχύ του σήματος που λαμβάνει.

Επίσης δίνοντας τις γεωγραφικές συντεταγμένες και το υψόμετρο του συστήματος καθώς και μίας κεραίας στόχου θα υπολογίζει την απόσταση του στόχου, το αζιμούθιο, την γωνία στον κατακόρυφο άξονα βάση της διαφοράς υψομέτρου των δύο σημείων και θα κινεί την κεραία προς τον στόχο.

Η υλοποίηση του συστήματος αυτού έχει ως σκοπό τη γρήγορη επίτευξη ασύρματης σύνδεσης δύο σημείων (point to point link) και η γρήγορη εναλλαγή μεταξύ διαφορετικών σημείων σύνδεσης. Ένας ακόμα τομέας εφαρμογής, είναι σε περιπτώσεις όπου οι κεραίες δεν είναι σε σταθερό σημείο αλλά μετακινούνται όπως σε πλοία, και αυτοκίνητα ή σε μη επανδρωμένα οχήματα UAV Drones.

Για την κατασκευή, χρησιμοποιήθηκε ως κεραία ένα πάνελ Nanostation M2 που διαθέτει ενσωματωμένο router board. Για την κίνηση, χρησιμοποιήθηκαν δύο βηματικοί κινητήρες με οδήγηση από το ολοκληρωμένο της Allegro A4988. Τον έλεγχο του συστήματος αναλαμβάνει ένα arduino με επεξεργαστή AtMega328 και με επέκταση για την υποστήριξη δικτύου Ethernet. Επίσης το σύστημα διαθέτει μια οθόνη LCD 16x2 για την άμεση προβολή πληροφοριών, αισθητήρα θερμοκρασίας και αισθητήρες φωτοδιόδου (Photomicrosensor) στους άξονες.

Η επικοινωνία του arduino με την κεραία καθώς και με τον χρήστη γίνεται μέσω του πρωτοκόλλου telnet.

Αξίζει επίσης να σημειωθεί ότι πολλά από τα υλικά της κατασκευής προήλθαν από την ανακύκλωση παλιών συσκευών όπως πχ: εκτυπωτές.

ΠΕΡΙΕΧΟΜΕΝΑ

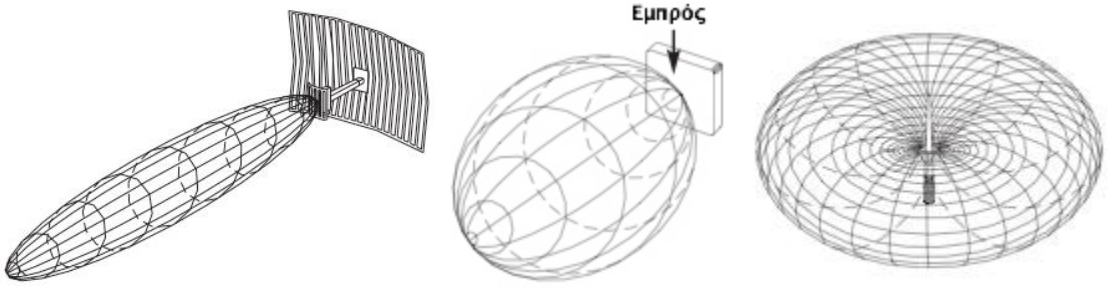
ΚΕΦΑΛΑΙΟ 1 : ΕΙΣΑΓΩΓΗ

1.1 Ασύρματα Δίκτυα και τα Είδη Κεραιών

Η ασύρματη μεταφορά δεδομένων έχει γίνει αναπόσπαστο κομμάτι της καθημερινής μας ζωής. Τα ασύρματα δίκτυα μπορεί να καλύπτουν τις ανάγκες σύνδεσης μεταξύ συσκευών σε οικιακό ή εργασιακό περιβάλλον είτε σε δίκτυα μεγάλων αποστάσεων όπου οι συσκευές μπορεί να βρίσκονται χιλιόμετρα μακριά.

Ένα από τα πολλά μέρη που απαρτίζουν ένα ασύρματο δίκτυο μεταφοράς δεδομένων και με το οποίο θα ασχοληθούμε σε αυτήν την εργασία είναι οι κεραίες. Η κεραία δεν είναι τίποτα άλλο παρά μια διάταξη με αγωγούς που επιτρέπει αποτελεσματικά εκπομπή ή/και λήψη ραδιοκυμάτων βάση του φαινόμενου της ηλεκτρομαγνητικής επαγωγής. Όταν λειτουργεί ως τμήμα του δέκτη, λαμβάνει ραδιοκύματα και τα μετατρέπει σε εναλλασσόμενο ρεύμα και όταν λειτουργεί ως τμήμα του πομπού, λαμβάνει εναλλασσόμενο ρεύμα και το μετατρέπει αντίστοιχα σε ραδιοκύματα. Οι κεραίες που χρησιμοποιούμε για την δημιουργία ασύρματων ζεύξεων έχουν διπλό ρόλο και αφορά και λήψη και εκπομπή από την ίδια κεραία (reciprocity).

Η επιλογή της κεραίας μας γίνεται ανάλογα με το είδος εξωτερικής ζεύξης που θέλουμε να πραγματοποιήσουμε. Όταν θέλουμε να κάνουμε σύνδεση ενός σταθερού σημείου με πολλαπλά σημεία σε διαφορετικές κατευθύνσεις ή με μη σταθερά σημεία (Point to Multipoint: Access Points, Hot Spots), τότε μας ενδιαφέρουν κεραίες οι οποίες εκπέμπουν παγκατευθυντικά. Αντίθετα όταν θέλουμε να συνδέσουμε δύο σταθερά σημεία (Point to point συνδέσεις), μας ενδιαφέρουν κεραίες που ενισχύουν την εκπομπή προς μία συγκεκριμένη κατεύθυνση αυξάνοντας έτσι και την εμβέλεια. Με αυτόν τον τρόπο χρειαζόμαστε μικρότερη ισχύ εκπομπής και παράλληλα εξασφαλίζουμε μικρότερο θόρυβο και παρεμβολές.

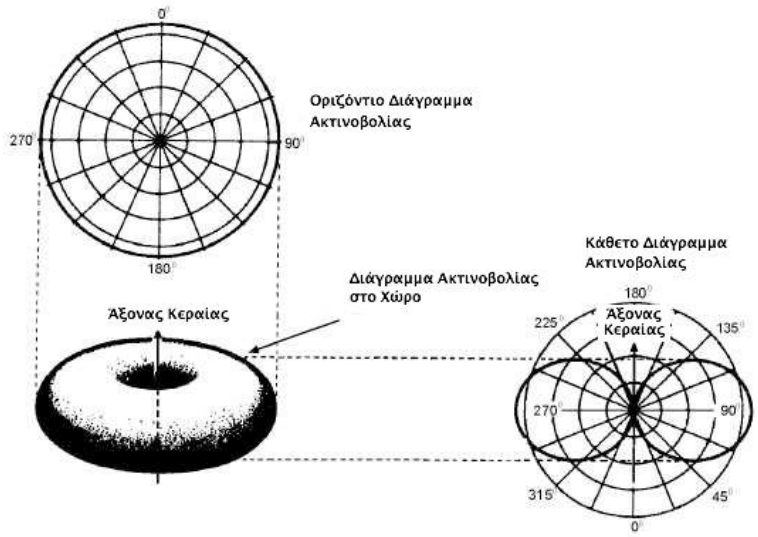


Εικ. 1: Κατευθυντική Ημι-κατευθυντική Παγκατευθυντική

Το βασικό πρόβλημα που προκύπτει με την χρήση κατευθυντικών κεραιών και το οποίο αυξάνεται όσο πιο κατευθυντικές γίνονται, δηλαδή στενεύει η δέσμη εκπομπής, είναι η σωστή στόχευση τους. Σε περιπτώσεις στόχευσης σημείου σε σημείο σε μεγάλες αποστάσεις και η πιο μικρή απόκλιση μπορεί να υποβαθμίσει την στάθμη του σήματος. Η στόχευση είναι μία διαδικασία που απαιτεί χρόνο και μεγάλη ακρίβεια. Σε αυτήν την εργασία γίνεται μία προσπάθεια αυτοματοποίησης αυτής της διαδικασίας.

1.2 Διαγράμματα Ακτινοβολίας και Ισχύς Σήματος

Όλες οι κεραίες έχουν συγκεκριμένο τρόπο για το πως εκπέμπουν (άρα και το πως λαμβάνουν) στον χώρο. Επομένως είναι σημαντικό όταν θέλουμε να επιλέξουμε μια κεραία να εξετάσουμε το διάγραμμα ακτινοβολίας της στον χώρο (ή αλλιώς πρότυπο εκπομπής – radiation pattern - aperture), το οποίο μας δίνει μια γρήγορη εικόνα σχετικά με το πως συμπεριφέρεται η κεραία [εικ. 1]. Για την ευκολία μας η ανάλυση του τρισδιάστατου διαγράμματος ακτινοβολίας αναλύεται σε δύο διαστάσεις κάθετα (vertical/elevation ή E-Plane) και οριζόντια (horizontal/azimuth ή H-Plane) συνθέτοντας αντίστοιχα, το κάθετο και το οριζόντιο διάγραμμα ακτινοβολίας [εικ. 3].



Εικ. 3: Αναπαράσταση 3d-Διαγράμματος Ακτινοβολίας Διπόλου σε 2d (οριζόντιο & κάθετο).

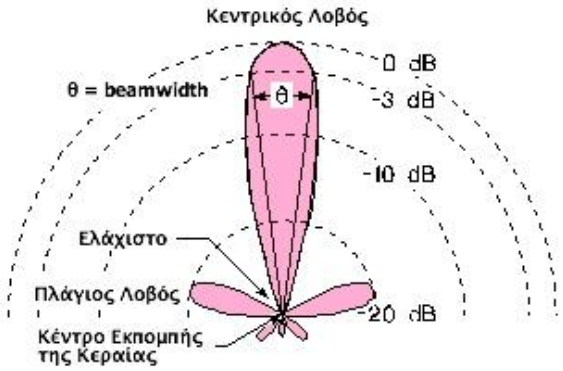
Ίσως το πρώτο πράγμα που βλέπουμε σε ένα διάγραμμα ακτινοβολίας είναι η κατευθυντικότητα της κεραίας. Πρόκειται όμως για εστίαση της εκπομπής προς συγκεκριμένες κατευθύνσεις περισσότερο από κάποιες άλλες και όχι για ενίσχυση της ισχύς του συνολικού σήματος.

Το πόσο κατευθυντική είναι μια κεραία φαίνεται στα χαρακτηριστικά της με δύο τρόπους.

Ο πρώτος είναι η ένδειξη της απολαβής της κεραίας (gain), που συνήθως μετριέται βάση μιας γνωστής τιμής είτε σε λογαριθμικές μονάδες dBi (deciBel over isotrop) είτε σε dBd (deciBel over dipole). Τα dBi της κεραίας καθορίζονται από το gain που έχει, συγκριτικά με μια ιδανική κεραία η οποία έχει μηδενικό gain και λέγεται isotropic dipole antenna (δεν υπάρχει, δεν μπορεί να κατασκευαστεί, αλλά είναι χρήσιμη εργαστηριακά ως σημείο αναφοράς στους υπολογισμούς μας). Τα dBd καθορίζονται και αυτά όπως τα dBi. Αλλά ως σημείο αναφοράς αντί για την ισοτροπική κεραία χρησιμοποιείται το δίπολο του Hertz. Η προσεχτική παρατήρηση στο πως αναγράφεται η απολαβή είναι σημαντική αφού 2dB λιγότερο είναι περίπου 65% χαμηλότερη ισχύ προς το σημείο με την μέγιστη κατευθυντικότητα. Η μετατροπή ωστόσο των dBd σε dBi είναι εύκολη και δεν έχουμε παρά να αφαιρέσουμε 1.76 μονάδες από τα dBd. Για παράδειγμα η κατευθυντικότητα μιας κεραίας η οποία έχει gain 30 dBd είναι ίδια με μια κεραία που έχει κατευθυντικότητα 28.24 dBi.

Εκτός και εάν έχει ειπωθεί διαφορετικά το gain μιας κεραίας μετριέται στον κύριο λοβό (mainlobe ή mainbeam) του διαγράμματος ακτινοβολίας της. Κύριος λοβός όπως φαίνεται και στην [εικ. 4] είναι το διακριτό τμήμα στο διάγραμμα ακτινοβολίας, που ορίζεται από το μέγιστο gain της κεραίας και δύο σημαντικά ελάχιστα δεξιά και αριστερά.

Ο δεύτερος τρόπος υπολογισμού της κατευθυντικότητας μια κεραίας είναι το έυρος δέσμης (θ= beamwidth) στον οριζόντιο και στον κάθετο άξονα του κύριου λοβού της κεραίας, που ορίζεται από τα σημεία τομής στον κύριο λοβό στη μισή τιμή ισχύος (-3dB) [εικ. 4]. Το εύρος δέσμης μετράται σε μοίρες ανάμεσα σε δύο σημεία, γνωστά ως σημεία μισής ισχύος, στις δύο πλευρές από το σημείο μέγιστης έντασης σήματος. Όσο μικρότερο είναι το εύρος δέσμης τόσο πιο κατευθυντική είναι η κεραία και το αντίθετο. Το εύρος δέσμης είναι συγκριτικά μικρότερο όσο υψηλότερη είναι η κεντρική συχνότητα.



Εικ. 4: Σχηματική αναπαράσταση διαγράμματος ακτινοβολίας μιας κατευθυντικής κεραίας και του κεντρικού λοβού της.

Σε σύνθετες κατασκευές κατευθυντικών κεραιών είναι σύνηθες να δημιουργούνται συμπληρωματικοί λοβοί σε άλλες μη ωφέλιμες κατευθύνσεις πέρα από την κεντρική δημιουργώντας τον οπίσθιο και τους πλαϊνούς λοβούς. Τις περισσότερες φορές αυτό είναι ένα ανεπιθύμητο χαρακτηριστικό και πολλές τεχνικές έχουν αναπτυχθεί με τα χρόνια έτσι ώστε να μειωθούν ή ακόμα και να εξαλειφθούν οι επιπλέον λοβοί.

<u>Υποσημείωση:</u> *Η θεωρία που περιλαμβάνεται στην παραπάνω εισαγωγή έχει προέλθει από τον «ΠΡΑΚΤΙΚΟ ΟΔΗΓΟ ΚΕΡΑΙΩΝ WiFi Σωκράτης Πανουσίου (socrates.. @.. awmn.net)(2002-2007: Athens Wireless Metropolitan Network)» και μπορεί να διαβαστεί εκτενέστερα σε αυτόν.*

ΚΕΦΑΛΑΙΟ 2: ΑΥΤΟΜΑΤΟΠΟΙΗΣΗ ΣΤΟΧΕΥΣΗΣ ΚΕΡΑΙΑΣ

2.1 Στόχοι

Θέλοντας να κατασκευάσουμε ένα σύστημα αυτόματης στόχευσης μίας κεραίας είναι απαραίτητο να ορίσουμε τις ελάχιστες απαιτήσεις που έχουμε από το σύστημα, τις λειτουργίες που θα πραγματοποιεί, αλλά και να κατανοήσουμε τους περιορισμούς που θα προκύψουν από τις επιλογές μας.

Στην περίπτωση, η κατασκευή θέλουμε να έχει τα ακόλουθα χαρακτηριστικά.

- Περιστροφή της κεραίας οριζόντια κατά 360 μοίρες (pan).
- Περιστροφή της κεραίας κάθετα κατά 180 μοίρες (tilt).
- Η κεραία (πομπός) να είναι σύμφωνη με το πρότυπο IEEE 802.11 b/g/n.
- Router board με λειτουργία Ethernet bridge.
- Η συσκευή να είναι συμβατή με πλήθος router board με λογισμικό Linux.
- Δυνατότητα εύκολης μελλοντικής αναβάθμισης του υλικού και λογισμικού.
- Αυτονομία. Το σύστημα να μην εξαρτάτε από λογισμικό σε κάποιον υπολογιστή.
- Απομακρυσμένη διαχείριση και λειτουργία.

Όσον αφορά τις λειτουργίες το σύστημα θα πρέπει:

- Να εντοπίζει την βέλτιστη θέση της κεραίας βάση της ισχύος του σήματος που λαμβάνει.
- Να υπολογίζει την στόχευση της κεραίας δίνοντας του τις γεωγραφικές συντεταγμένες του συστήματος και ενός απομακρυσμένου σημείου σύνδεσης.
- Μη αυτόματη επιλογή θέσης.
- Απομακρυσμένη ενημέρωση για την κατάσταση του συστήματος και προαιρετικά κάποιων αισθητήρων π.χ. θερμοκρασίας.

2.2 Επιλογές

Για να επιτύχουμε τα παραπάνω έπρεπε να γίνει μία σειρά επιλογών όσων αφορά την μορφή του συστήματος και τις συσκευές που θα το αποτελούν.

2.2.1 Κεραία

Το σύστημα που επιλέχθηκε αποτελείται από μία κεραία τύπου πάνελ με ενσωματωμένο router board, της εταιρίας ubiquity. Καλύπτει όλες τις λειτουργίες που θέλουμε για την ασύρματη σύνδεση του δικτύου ενώ προσφέρει δύο εξόδους Ethernet. Επίσης δίνει την δυνατότητα επικοινωνίας και ρύθμισης του μέσω του πρωτοκόλλου επικοινωνίας telnet που κάνει πολύ εύκολη την σύνδεση του με μία μονάδα μικροελεγκτή χωρίς την χρήση πολύπλοκων πρωτοκόλλων ασφαλείας. Συγκεκριμένα το μοντέλο που επιλέχθηκε είναι το Nanostation M2 με τα ακόλουθα τεχνικά χαρακτηριστικά.

SYSTEM INFORMATION	
Processor Specs	Atheros MIPS 24KC, 400MHz
Memory Information	32MB SDRAM, 8MB Flash
Networking Interface	2 X 10/100 BASE-TX (Cat. 5, RJ-45) Ethernet Interface

REGULATORY / COMPLIANCE INFORMATION	
Wireless Approvals	FCC Part 15.247, IC RS210, CE
RoHS Compliance	YES

OPERATING FREQUENCY 2412MHz-2462MHz

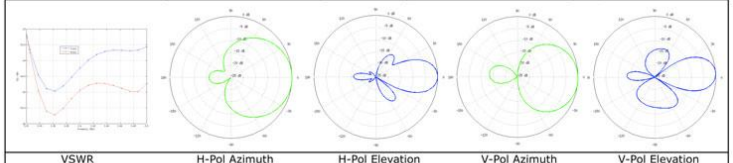
TX POWER SPECIFICATIONS			
	DataRate	Avg. TX	Tolerance
11b/g	1-24Mbps	28 dBm	+/-2dB
	36Mbps	26 dBm	+/-2dB
	48Mbps	25 dBm	+/-2dB
	54Mbps	24 dBm	+/-2dB
11n / Airmax	MCS0	28 dBm	+/-2dB
	MCS1	28 dBm	+/-2dB
	MCS2	28 dBm	+/-2dB
	MCS3	28 dBm	+/-2dB
	MCS4	27 dBm	+/-2dB
	MCS5	25 dBm	+/-2dB
	MCS6	23 dBm	+/-2dB
	MCS7	22 dBm	+/-2dB
	MCS8	28 dBm	+/-2dB
	MCS9	28 dBm	+/-2dB
	MCS10	28 dBm	+/-2dB
	MCS11	28 dBm	+/-2dB
	MCS12	27 dBm	+/-2dB
	MCS13	25 dBm	+/-2dB
	MCS14	23 dBm	+/-2dB
	MCS15	22 dBm	+/-2dB

RX SPECIFICATIONS			
	DataRate	Sensitivity	Tolerance
11b/g	1-24Mbps	-97 dBm min	+/-2dB
	36Mbps	-80 dBm	+/-2dB
	48Mbps	-77 dBm	+/-2dB
	54Mbps	-75 dBm	+/-2dB
11n / Airmax	MCS0	-96 dBm	+/-2dB
	MCS1	-95 dBm	+/-2dB
	MCS2	-92 dBm	+/-2dB
	MCS3	-90 dBm	+/-2dB
	MCS4	-86 dBm	+/-2dB
	MCS5	-83 dBm	+/-2dB
	MCS6	-77 dBm	+/-2dB
	MCS7	-74 dBm	+/-2dB
	MCS8	-95 dBm	+/-2dB
	MCS9	-93 dBm	+/-2dB
	MCS10	-90 dBm	+/-2dB
	MCS11	-87 dBm	+/-2dB
	MCS12	-84 dBm	+/-2dB
	MCS13	-79 dBm	+/-2dB
	MCS14	-78 dBm	+/-2dB
	MCS15	-75 dBm	+/-2dB

PHYSICAL / ELECTRICAL / ENVIRONMENTAL	
Enclosure Size	29.4 cm x 8 cm x 3cm
Weight	0.4kg
Enclosure Characteristics	Outdoor UV Stabalized Plastic
Mounting Kit	Pole Mounting Kit included
Max Power Consumption	8 Watts
Power Supply	24V, 0.5A surge portection integrated POE adapter included
Power Method	Passive Power over Ethernet (pairs 4,5+; 7,8 return)
Operating Temperature	-30C to +80C
Operating Humidity	5 to 95% Condensing
Shock and Vibration	ETSI300-019-1.4

INTEGRATED 2x2 MIMO ANTENNA

Frequency Range	2.32-2.55 GHz	Max VSWR	1.6:1
Gain	10.4-11.2 dBi	H-pol Beamwidth	55 deg.
Polarization	Dual Linear	V-pol Beamwidth	53 deg.
Cross-pol Isolation	23dB minimum	Elevation Beamwidth	27 deg.



Εικ. 5 Τεχνικά Χαρακτηριστικα Nanostation

Όπως παρατηρούμε από τα διαγράμματα ακτινοβολίας, το εύρος δέσμης της κεραίας είναι 53 μοίρες κάτι που είναι λογικό αφού πρόκειται για κεραία τύπου πάνελ, αλλά την κάνει να μην είναι και η καλύτερη επιλογή για την εφαρμογή μας αφού δεν είναι πολύ κατευθυντική. Πειραματικά παρά την έλλειψη κατευθυντικότητας το σύστημα δείχνει καλά αποτελέσματα στον σωστό προσδιορισμό της θέσης βέλτιστου σήματος.

2.2.2 Υπολογιστική Πλατφόρμα - Μικροελεγκτής

Το «μυαλό» του συστήματος αποτελεί ένας μικροελεγκτής και βασίζεται στην πλατφόρμα Arduino. Η επικοινωνία του μικροελεγκτή με την κεραία καθώς και με τον χρήστη γίνεται μέσω μίας πρόσθετης μονάδας που παρέχει στον μικροελεγκτή σύνδεση Ethernet. Ο προγραμματισμός έγινε στην γλώσσα wiring (απλοποιημένη C++) και ο μικροελεγκτής είναι σε θέση να εκτελεί όλες τις λειτουργίες που θέσαμε ως στόχο, χωρίς να χρειάζεται την υποστήριξη κάποιου άλλου υπολογιστικού συστήματος.

Εικ. 6: Arduino Uno και Arduino Ethernet Shield

Το Arduino είναι μια υπολογιστική πλατφόρμα ανοιχτού υλικού και λογισμικού. Βασίζεται σε μια αναπτυξιακή πλακέτα που ενσωματώνει επάνω έναν μικροελεγκτή και συνδέεται με τον Η/Υ για να τον προγραμματίσουμε μέσα από ένα απλό περιβάλλον ανάπτυξης. Το μοντέλο που θα χρησιμοποιήσουμε είναι το UNO με τα ακόλουθα τεχνικά χαρακτηριστικά:

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Βέβαια ο κώδικας της εφαρμογής είναι συμβατός με όλα τα μοντέλα Arduino αρκεί να έχουν την απαραίτητη μνήμη και εισόδους - εξόδους για να υποστηρίξουν το σύστημα.

2.2.3 Κινητήριο σύστημα

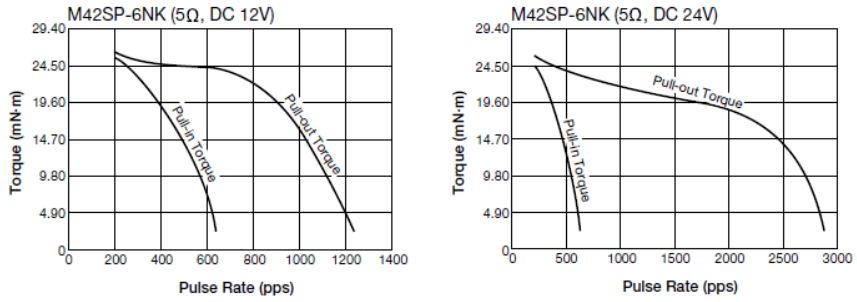
Για την κίνηση του συστήματος επιλέχθηκε η χρήση βηματικών κινητήρων καθώς και γραναζιών για την μετάδοση της κίνησης στους άξονες περιστροφής. Οι βηματικοί κινητήρες είναι εύκολο να ελεγχθούν από τον μικροελεγκτή μέσω παλμών που στέλνονται σε μία μονάδα οδήγησης. Προσφέρουν πολύ μεγάλη ακρίβεια στον έλεγχο της περιστροφής τους και αρκετή ροπή για να κινήσουν το σύστημα ακόμα και σε πολύ χαμηλό αριθμό στροφών. Η χρήση άλλων κινητήρων απορρίφθηκε καθώς θα έπρεπε να χρησιμοποιηθούν επιπλέον αισθητήρες για τον προσδιορισμό της ακριβούς περιστροφής του άξονα. Επίσης οι σερβοκινητήρες απορρίφθηκαν γιατί προσφέρουν μόνο 180 μοίρες περιστροφής.

Οι βηματικοί κινητήρες που χρησιμοποιήθηκαν προήλθαν από την ανακύκλωση παλιών εκτυπωτών και έχουν τα ακόλουθα ηλεκτρικά χαρακτηριστικά.

SPECIFICATIONS

Item	M42SP-6NK	
Rated Voltage	DC 12V	DC 24V
Working Voltage	DC 10.8~13.2V	DC 21.6~26.4V
Rated Current/Phase	400mA(PEAK)	
No. of Phase	2 Phase	
Coil DC Resistance	5Ω/phase±7%	
Step Angle	7.5°/step	
Excitation Method	2-2 Phase excitation (Bipolar driving)	
Insulation Class	Class E insulation	
Holding Torque	41.2mN·m	41.2mN·m
Pull-out Torque	21.6mN·m/800pps	17.6mN·m/2,000pps
Pull-in Torque	25.5mN·m/200pps	22.5mN·m/200pps
Max. Pull-out Pulse Rate	1,200pps	2,900pps
Max. Pull-in Pulse Rate	640pps	640pps

CHARACTERISTICS



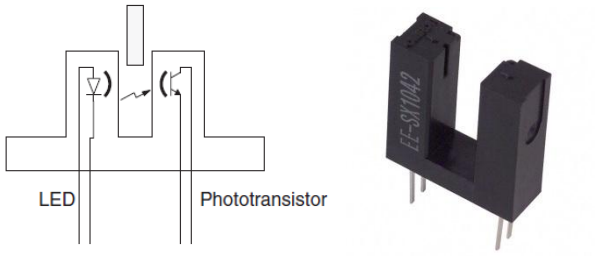
Εικ. 7: Ηλεκτρικά χαρακτηριστικά βηματικών κινητήρων

Τα χαρακτηριστικά στα οποία θα πρέπει να δώσουμε ιδιαίτερη σημασία είναι η τάση λειτουργίας, το μέγιστο επιτρεπόμενο ρεύμα λειτουργίας το οποίο θα πρέπει να ρυθμίσουμε στην συσκευή οδήγησης ώστε να μην το ξεπεράσουμε, και ο μέγιστος αριθμός παλμών ανά δευτερόλεπτο που μπορούμε να στείλουμε. Ο αριθμός των παλμών καθορίζει και την ταχύτητα του κινητήρα ο οποίος δεν θα λειτουργήσει εάν βρισκόμαστε εκτός των ορίων του.

Για την οδήγηση των βηματικών κινητήρων επιλέχθηκε το chip A4988 της εταιρίας Alegro. Είναι τεχνολογίας DMOS έχει πολύ μικρό μέγεθος και μικρές θερμικές απώλειες κάνοντας

περιττή την χρήση ψήκτρας για την χρήση που το θέλουμε. Μπορεί να οδηγήσει κινητήρες έως 35V με ρεύμα 2A και διαθέτει ρυθμιζόμενη προστασία υπερέντασης του κινητήρα.

Η ρύθμιση της αρχικής θέσης των κινητήρων και η ευθυγράμμιση των αξόνων περιστροφής μπορεί να πραγματοποιηθεί με την βοήθεια αισθητήρων φωτοδιόδου (Photomicrosensor). Οι αισθητήρες τοποθετούνται στους άξονες περιστροφής και στέλνουν ένα σήμα στην μονάδα του μικροελεγκτή όταν ο άξονας περάσει από ένα προκαθορισμένο σημείο. Η αρχή λειτουργίας τους είναι πολύ απλή, μία δέσμη φωτός (LED) διεγείρει ένα φωτοτρανζίστορ. Εάν η δέσμη διακοπεί από ένα αντικείμενο που περνά από την περιοχή μεταξύ της φωτοδιόδου και του φωτοτρανζίστορ, το τελευταίο αλλάζει κατάσταση στέλνοντας ένα σήμα στον μικροελεγκτή.



Εικ. 8: Transmissive Photomicrosensor

2.2.4 Οθόνη

Κρίθηκε απαραίτητη η τοποθέτηση στο σύστημα μίας οθόνης για την άμεση προβολή πληροφοριών της κατάστασης λειτουργίας του συστήματος. Ίσως η πιο βασική πληροφορία είναι η διεύθυνση IP με την οποία έχει συνδεθεί ο μικροελεγκτής στο δίκτυο. Η διεύθυνση αυτή δίνεται αυτόματα από το δίκτυο μέσω του πρωτοκόλλου DHCP και μπορεί να αλλάζει κατά διαστήματα. Είναι απαραίτητο να την γνωρίζουμε προκειμένου να συνδεθούμε στο τερματικό TELNET του μικροελεγκτή και να πραγματοποιήσουμε οποιονδήποτε χειρισμό. Επίσης η οθόνη μπορεί να μας παρέχει άλλες πληροφορίες για το σύστημα, αλλά και να χρησιμοποιηθεί και σαν ένα ισχυρό εργαλείο αποσφαλμάτωσης (debugging).

Ως οθόνη επιλέχθηκε μία οθόνη τύπου LCD δύο γραμμών των 16 χαρακτήρων η κάθε μία. Η οθόνη ενσωματώνει τον οδηγό HD44780 ο οποίος για να ελεγχθεί απαιτεί τουλάχιστον 4 κανάλια εξόδου από τον μικροελεγκτή. Για να περιορίσουμε τον αριθμό των καναλιών σε μόλις δύο χρησιμοποιούμε ως ενδιάμεσο ένα Shift Register 74HC595 και κάνουμε μία μικρή τροποποίηση στον κώδικα του μικροελεγκτή.

Εικ. 9: Οθόνη LCD 1602

2.2.5 Τροφοδοσία και αισθητήρες

Η τροφοδοσία όλων των συσκευών επιλέχθηκε να γίνει με ένα τροφοδοτικό δύο εξόδων. 240V AC σε 12V 2A για τους κινητήρες και 5V 2A για όλα τα υπόλοιπα. Η μονάδα της κεραίας (Nanostation) διαθέτει δική της τροφοδοσία μέσω του καλωδίου δικτύου (Power Over Ethernet).

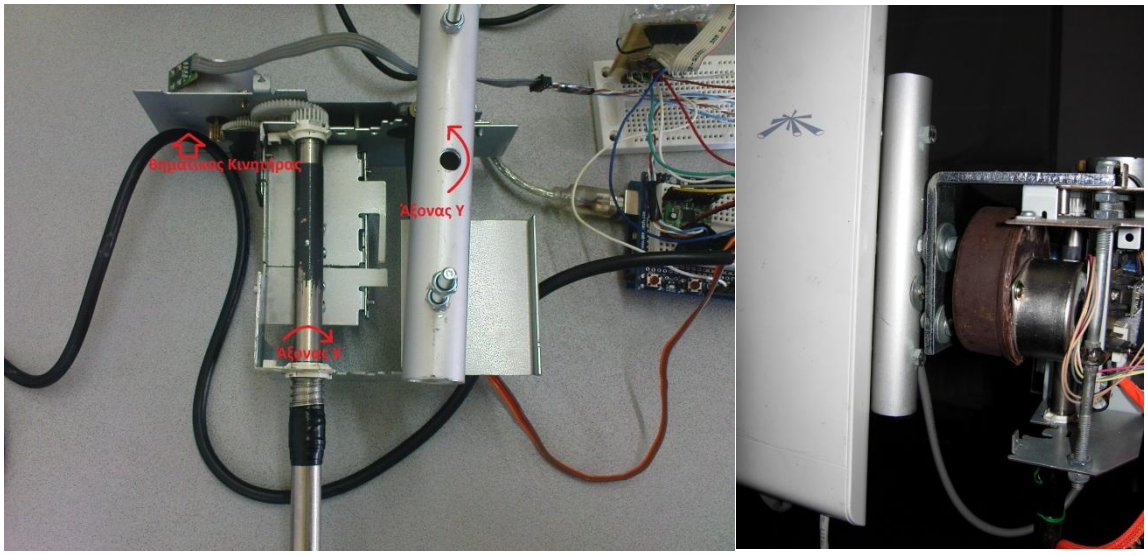
Προαιρετικά τοποθετήθηκε ένας αισθητήρας μέτρησης της θερμοκρασίας περιβάλλοντος την οποία μπορούμε να δούμε απομακρυσμένα από το τερματικό, αλλά στην θέση του θα μπορούσε να τοποθετηθεί και οποιοδήποτε άλλο αισθητήριο χρειαστεί. Το μοντέλο είναι το DS18S20 της εταιρίας Maxim και επικοινωνεί με τον μικροελεγκτή μέσω πρωτοκόλλου 1-Wire.

ΚΕΦΑΛΑΙΟ 3 : ΥΛΙΚΟ (HARDWARE)

Στο κεφάλαιο αυτό θα ασχοληθούμε με τον σχεδιασμό του μηχανολογικού υλικού που θα αποτελεί την βάση του συστήματος, πάνω στην οποία θα τοποθετηθούν όλες οι επιμέρους συσκευές. Στην συνέχεια θα μελετήσουμε την ηλεκτρική σύνδεση των συσκευών και τον τρόπο που θα επικοινωνούν μεταξύ τους.

3.1 Κατασκευή και Συναρμολόγηση Μηχανικών Μερών

Ένας από τους σκοπούς της εργασίας ήταν το χαμηλό κόστος των υλικών και η ανακύκλωση υλικών όπου αυτό ήταν δυνατό. Έτσι η βάση στην οποία στηρίζεται όλο το σύστημα προήλθε από την μονάδα τροφοδοσίας χαρτιού ενός χαλασμένου εκτυπωτή. Ύστερα από την κοπή, τροποποίηση και συγκόλληση διαφόρων μεταλλικών επιφανειών από το σασί του εκτυπωτή κατέληξα στην παρακάτω κατασκευή.



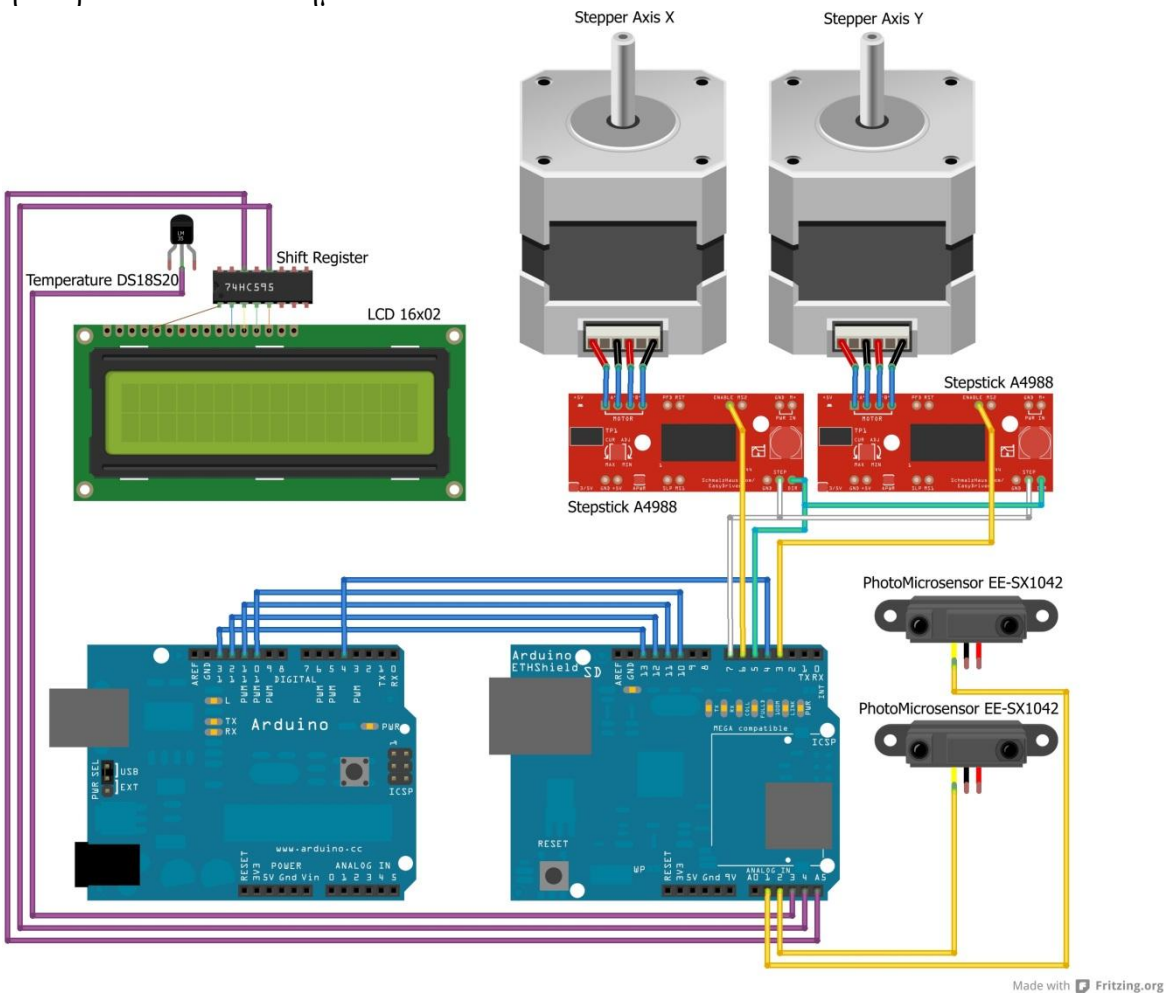
Εικ. 10: Μεταλλικό σασί και σύστημα κίνησης

Ο άξονας Χ μπορεί να προσδεθεί σε ιστό κεραίας, οπότε με την βοήθεια του βηματικού κινητήρα όλη η κατασκευή μπορεί να περιστραφεί γύρω από αυτόν. Στον άξονα Υ βλέπουμε έναν σωλήνα αλουμινίου που αποτελεί τη βάση στήριξης της κεραίας τύπου πάνελ. Στον αρχικό σχεδιασμό, τον άξονα αυτόν περιέστρεφε ένας σερβοκινητήρας, όμως λόγω προβλημάτων από κραδασμούς που δημιουργούσε αντικαταστάθηκε από βηματικό κινητήρα, όπως φαίνεται στην δεύτερη φωτογραφία.

Υποσημείωση: *Για λόγους ανακύκλωσης ο βηματικός κινητήρας προήλθε και πάλι από εκτυπωτή ενώ το κιβώτιο υποβιβασμού και μετάδοσης της κίνησης προήλθε από ηλεκτρική σούβλα.*

3.2 Ηλεκτρικά και Ηλεκτρονικά

Το δεύτερο στάδιο της εργασίας ήταν ο σχεδιασμός της διασύνδεσης των ηλεκτρικών και ηλεκτρονικών υποσυστημάτων.



Εικ. 11: Διασύνδεση μικροελεγκτή

Όλες οι συσκευές που αποτελούν το σύστημα θα πρέπει να συνδεθούν με τον κεντρικό μικροελεγκτή (Arduino), και η κάθε μία έχει τις δικές της ιδιαιτερότητες.

Το Arduino Uno που επιλέξαμε να χρησιμοποιήσουμε διαθέτει 16 ψηφιακές εξόδους-εισόδους (0-13) γενικής χρήσης και 6 αναλογικές (A0-A5). Αφού όλες οι συσκευές που χρησιμοποιούμε είναι ψηφιακές μπορούμε να χρησιμοποιήσουμε και τις αναλογικές ως ψηφιακές. Θα πρέπει να προσέξουμε ότι οι θύρες 0 και 1 χρησιμοποιούνται από το Arduino για σειριακή επικοινωνία και προγραμματισμό του μικροελεγκτή οπότε καλό είναι να μην χρησιμοποιηθούν.

3.2.1 Ethernet Shield

Η πρόσθετη μονάδα Ethernet Shield παρέχει στο Arduino μία έξοδο RJ45 και μέσω του Ethernet Controller: W5100 που διαθέτει του δίνει την δυνατότητα να συνδεθεί στο δίκτυο

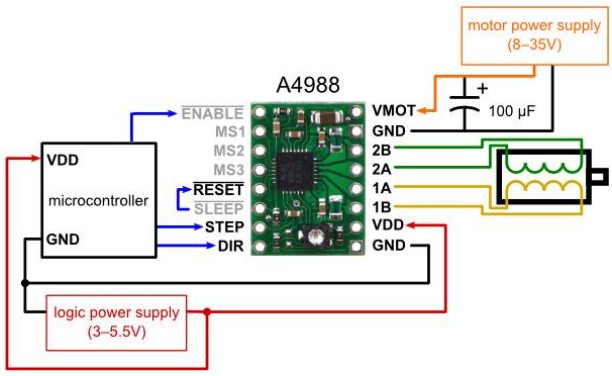
Ethernet. Επίσης παρέχει μία υποδοχή για προσθήκη αποθηκευτικού χώρου σε μορφή κάρτας SD την οποία δεν θα χρησιμοποιήσουμε στην παρούσα εφαρμογή.

Η μονάδα τοποθετείται απευθείας πάνω στο Arduino και συνδέεται με αυτό μέσω του πρωτοκόλλου SPI. Το SPI δημιουργεί έναν δίαυλο επικοινωνίας περιφερειακών συσκευών. Χρησιμοποιεί τρία κανάλια επικοινωνίας τα οποία είναι κοινά για όλες τις συσκευές, συν ένα για κάθε επιπλέον συσκευή το οποίο κάνει την επιλογή της. Τον ρόλο των τριών κοινών καναλιών τον έχουν οι θύρες 11,12,13 του Arduino, ενώ η 10 επιλέγει τον Ethernet Controller: W5100 και η 4 την κάρτα SD. Οι θύρες αυτές δεν μπορούν να χρησιμοποιηθούν για άλλη χρήση.

Υποσημείωση: *περισσότερες πληροφορίες για το Ethernet Shield υπάρχουν στην ιστοσελίδα* *http://arduino.cc/en/Main/ArduinoEthernetShield*

3.2.2 Οδήγηση Βηματικών Κινητήρων

Για την οδήγηση των βηματικών κινητήρων επιλέχθηκε το ολοκληρωμένο A4988. Επειδή έχει πολύ μικρό μέγεθος και είναι δύσκολο να κολληθεί αγοράστηκαν έτοιμα «breakout-board».



Εικ. 12: Τυπική σύνδεση βηματικών κινητήρων

Τα τυλίγματα του βηματικού κινητήρα συνδέονται στους ακροδέκτες 2B,2A,1A,1B του οδηγού. Ο οδηγός τροφοδοτείται με 12V για τις ανάγκες του κινητήρα και 5V για την τροφοδοσία του ολοκληρωμένου. Η τροφοδοσία γίνεται από το γενικό τροφοδοτικό του συστήματος. Η σύνδεση με τον μικροελεγκτή γίνεται μέσω τριών καλωδίων. Η θύρα 7 του Arduino συνδέεται με τον ακροδέκτη STEP και στέλνει έναν παλμό κάθε φορά που θέλουμε να εκτελέσουμε ένα βήμα του κινητήρα. Η θύρα 6 συνδέεται στον ακροδέκτη DIR και επιλέγει την κατεύθυνση προς την οποία θέλουμε να εκτελεστεί το βήμα. Τέλος ο ακροδέκτης ENABLE ενεργοποιεί τον οδηγό.

Επειδή το σύστημα διαθέτει δύο βηματικούς κινητήρες και άρα δύο οδηγούς, μπορούμε να βραχυκυκλώσουμε τους αντίστοιχους ακροδέκτες STEP και DIR των δύο οδηγών και να ενεργοποιούμε μέσω διαφορετικών θυρών τον οδηγό που θέλουμε να λειτουργήσουμε. Χρησιμοποιούμε την θύρα 6 για την ενεργοποίηση του οδηγού στον άξονα X και την θύρα 3 στον άξονα Y [Εικ. 11].

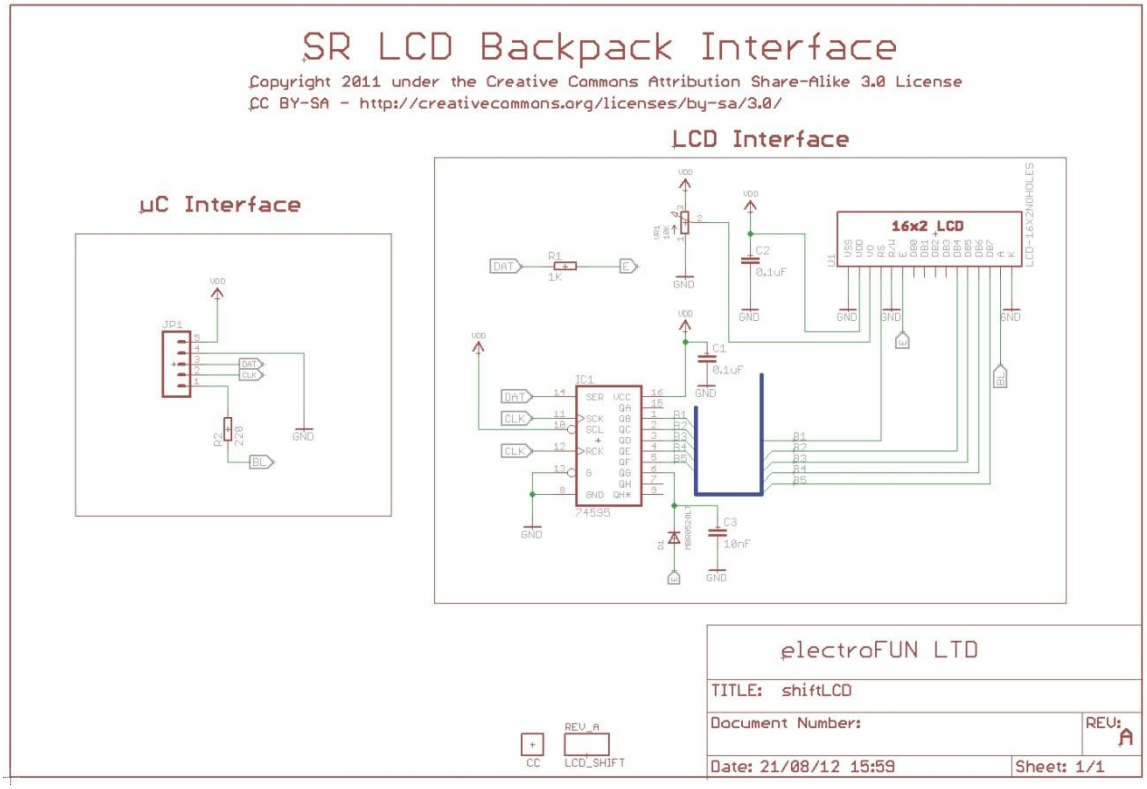
Υποσημείωση: *περισσότερες πληροφορίες για τον οδηγό των βηματικών κινητήρων υπάρχουν στην ιστοσελίδα http://reprap.org/wiki/StepStick*

3.2.3 Οθόνη LCD

Η οθόνη είναι μία τυπική 2 γραμμών επί 16 χαρακτήρων LCD. Συνήθως για την σύνδεση της με έναν μικροελεγκτή σε 4-bit mode χρειάζονται 6 θύρες επικοινωνίας. Οι 4 είναι για τα δεδομένα και δύο για τον έλεγχο της οθόνης.

Υπάρχουν αρκετοί τρόποι για τον περιορισμό των απαιτούμενων θυρών. Επιλέχθηκε η χρήση ενός Shift-Register ως ενδιάμεση επικοινωνία μεταξύ της οθόνης και του μικροελεγκτή. Με αυτόν τον τρόπο μειώνονται οι απαιτούμενες θύρες σε μόλις δύο.

Το ολοκληρωμένο Shift-Register είναι το 74HC595 και η σύνδεση του έγινε σύμφωνα με το παρακάτω σχέδιο.



Εικ. 13: Two wire latch Shift Register Schematic (https://bitbucket.org/fmalpartida/new-liquidcrystal/wiki/schematics#!shiftregister-connection)

Η σύνδεση με το Arduino έγινε στους ακροδέκτες A4(Clock Pin) και A5(Data/Enable Pin).

3.2.4 Αισθητήρες

Χρησιμοποιήθηκαν δύο ειδών αισθητήρες, ένας αισθητήρας θερμοκρασίας και δύο αισθητήρες φωτοδιόδου (Photomicrosensor) [κεφάλαιο 2.2.3].

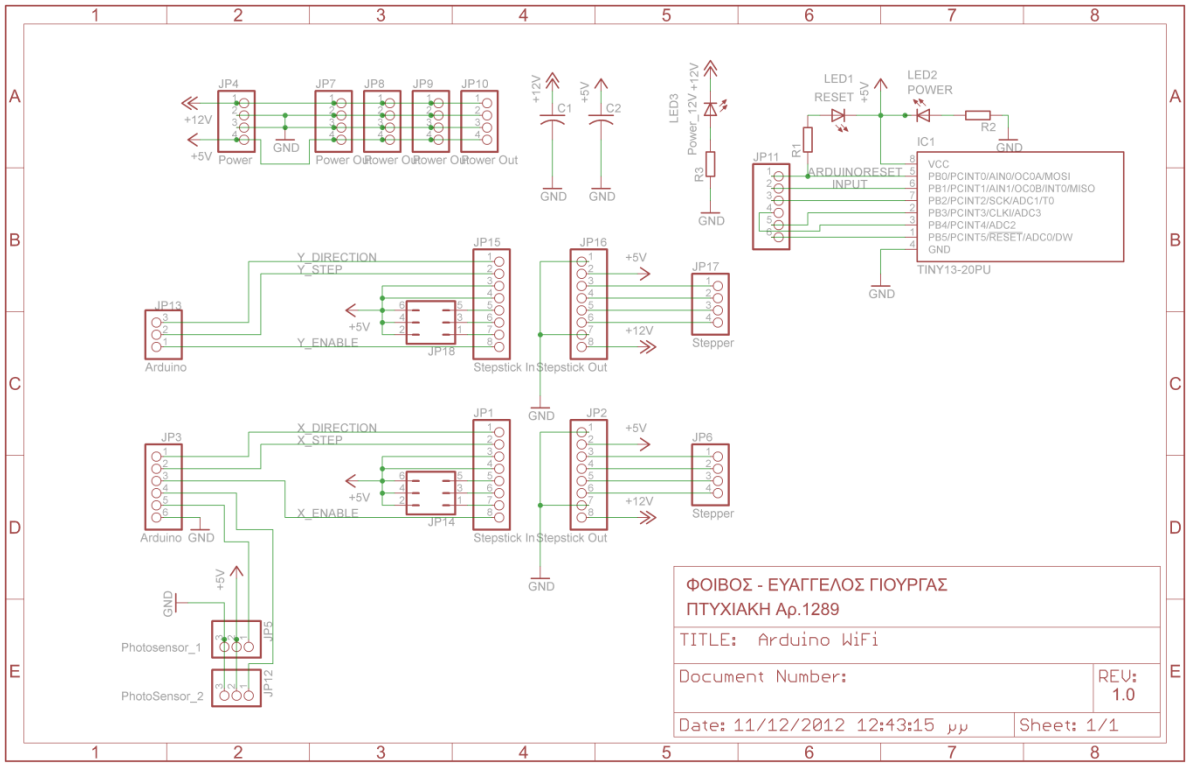
Ο αισθητήρας θερμοκρασίας DS18S20 επικοινωνεί μέσω του πρωτοκόλλου 1-Wire . Όπως υποδηλώνει το όνομα του απαιτεί ένα καλώδιο για να στείλει δεδομένα στον μικροελεγκτή

και συνδέθηκε στην θύρα A3. Βεβαίως χρειάζεται και τροφοδοσία η οποία παρέχεται από το γενικό τροφοδοτικό στα 5V.

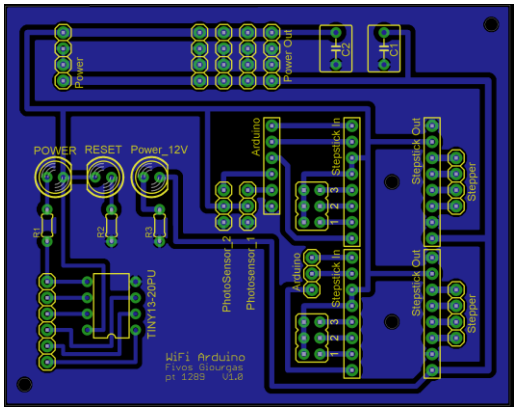
Οι αισθητήρες Photomicrosensor συνδέθηκαν στην θύρα A1 για τον άξονα X και A2 για τον άξονα Y. Η τροφοδοσία τους είναι επίσης 5V.

3.2.5 Μητρική πλακέτα

Για να διευκολυνθεί η σύνδεση των συσκευών σχεδιάστηκε μία μητρική πλακέτα η οποία συνδέεται με τον μικροελεγκτή. Αυτή φιλοξενεί τους δύο οδηγούς των βηματικών κινητήρων διανέμει την τροφοδοσία προς τις συσκευές και τους αισθητήρες, ενώ διαθέτει υποδοχή για έναν προαιρετικό δεύτερο μικροελεγκτή.



Εικ. 14: Σχέδιο μητρικής πλακέτας



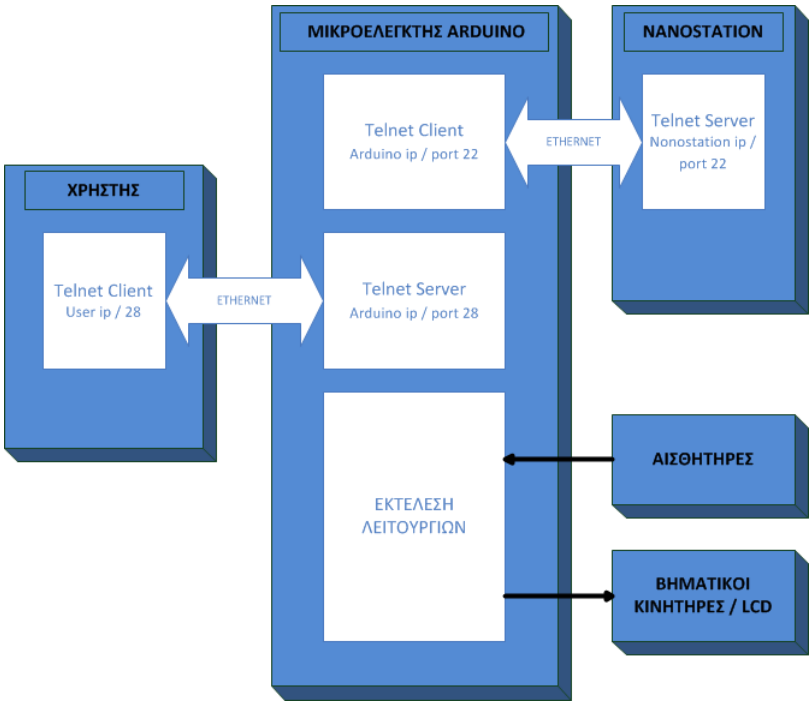
Εικ. 15 Τυπωμένο κύκλωμα μητρικής

ΚΕΦΑΛΑΙΟ 4 : ΛΟΓΙΣΜΙΚΟ (SOFTWARE)

Σε αυτό το κεφάλαιο παρουσιάζεται η αρχή λειτουργίας του συστήματος, γίνεται ανάλυση του πηγαίου κώδικα που «τρέχει» στον μικροελεγκτή, καθώς και των λειτουργιών που εκτελεί.

4.1 Αρχή Λειτουργίας του Συστήματος και Προγραμματισμός

4.1.1 Αρχή λειτουργίας



Εικ. 16 : Διάγραμμα λειτουργίας του συστήματος

Το διάγραμμα της εικόνας 16 παρουσιάζει την αρχή λειτουργίας του συστήματος. Ο μικροελεγκτής (Arduino) θα πρέπει να πραγματοποιεί δύο συνδέσεις Ethernet.

Η πρώτη σύνδεση είναι προς τον Telnet Server του Nanostation. Μέσω αυτής στέλνοντας τις κατάλληλες εντολές θα μπορεί να λαμβάνει πληροφορίες όπως είναι η ισχύς του σήματος που λαμβάνει η κεραία.

Η δεύτερη σύνδεση είναι προς τον χρήστη του συστήματος. Δημιουργώντας έναν Telnet Server στον οποίο ο χρήστης θα μπορεί να συνδεθεί και να εκτελεί χειρισμούς του συστήματος ή να λαμβάνει πληροφορίες.

Όταν ο μικροελεγκτής δεχθεί μία εντολή θα πρέπει να την επεξεργάζεται και να εκτελεί την αντίστοιχη ρουτίνα.

4.1.2 Γλώσσα Προγραμματισμού

Ο κώδικας γράφτηκε στο αναπτυξιακό περιβάλλον Arduino IDE, με χρήση της γλώσσας προγραμματισμού Wiring. Η γλώσσα Wiring αποτελεί μία απλοποιημένη μορφή της C++. Περισσότερες πληροφορίες υπάρχουν στην επίσημη ιστοσελίδα του Arduino http://www.arduino.cc .

4.1.3 Δομή του Προγράμματος

Στην αρχή του προγράμματος δηλώνονται οι βιβλιοθήκες που χρησιμοποιεί το πρόγραμμα και οι καθολικές μεταβλητές. Οι καθολικές μεταβλητές έχουν εμβέλεια σε ολόκληρο το πρόγραμμα.

Στην συνέχεια ακολουθεί η λειτουργία **Setup.** Στο Setup πραγματοποιούνται οι ρυθμίσεις παραμέτρων του μικροελεγκτή. Ορίζονται οι θύρες του μικροελεγκτή ως είσοδοι ή έξοδοι καθώς και οι τιμή τους. Επίσης γίνεται έναρξη ορισμένων βιβλιοθηκών.

Τα πρώτα δύο μέρη του προγράμματος εκτελούνται μόνο κατά την εκκίνηση του μικροελεγκτή. Εκτελούνται ξανά εάν γίνει επανεκκίνηση ή επαναφορά του μικροελεγκτή.

Ακολουθεί ο βασικός βρόχος (**main loop**) του προγράμματος ο οποίος και επαναλαμβάνεται κατά την διάρκεια λειτουργίας του μικροελεγκτή.

Τέλος λόγω της πολυπλοκότητας του προγράμματος θα γίνει χρήση της τεχνικής του τμηματικού προγραμματισμού. Το πρόγραμμα θα χωριστεί σε υπορουτίνες (global functions), οι οποίες θα εκτελούν συναρτήσεις ή διαδικασίες του προγράμματος. Οι υπορουτίνες έχουν καθολικό χαρακτήρα και άρα μπορεί να γίνει κλήση τους από οποιοδήποτε μέρος του προγράμματος.

Οι υπορουτίνες που θα χρησιμοποιηθούν είναι οι παρακάτω.

Υπορουτίνες (Global Functions)

command_prompt: εμφάνιση προτροπής για εισαγωγή εντολής (telnet)
help: εμφάνιση βοήθειας (telnet)
show_status: εμφάνιση κατάστασης συσκευής (telnet)
connectToNanostation: εκτέλεση συνδέσεων δικτύου
point_location: ρύθμιση με χρήση γεωγραφικών συντεταγμένων
azimuth: υπολογισμός αζιμούθιου
differencial_altitude_angle: υπολογισμός γωνίας λόγω διαφοράς υψομέτρου
distance_between: υπολογισμός απόστασης
getSignal: λήψη τιμής ισχύος σήματος από το nanostation
getTemp: λήψη τιμής θερμοκρασίας
lcd_update: ανανέωση πληροφοριών οθόνης LCD
move: κίνηση των δύο αξόνων
moveX: κίνηση του άξονα X
moveY: κίνηση του άξονα Y
optimal: αναζήτηση βέλτιστης θέσης στον άξονα X
optimal_Y: αναζήτηση βέλτιστης θέσης στον άξονα Y

scan_function: σάρωση του άξονα X
scan_function_Y: σάρωση του άξονα Y
sort: χρησιμοποιείται για την ταξινόμηση των τιμών ισχύος σήματος.
Calibrate: ρύθμιση βηματικών κινητήρων

4.2 Ανάλυση του Κώδικα

Κατά την ανάλυση του προγράμματος θα παρατίθενται τμήματα του κώδικα και θα περιγράφεται η λειτουργία τους.

4.2.1 Έναρξη και Setup

Αρχικά γίνεται η εισαγωγή των βιβλιοθηκών που χρησιμοποιεί το πρόγραμμα.

```
#include <stdlib.h>
#include <SPI.h>
#include <Ethernet.h>
#include <Servo.h>
#include <MemoryFree.h>
#include <LiquidCrystal_SR.h>
#include <OneWire.h>
```

Ορίζονται οι μεταβλητές που αντιστοιχούν στους αριθμούς των θυρών του μικροελεγκτή. Θα χρησιμοποιηθούν στην συνέχεια για την δήλωση χρήσης των θυρών.

```
//pinout variables
const byte steppin = 7;  //stepstick step pin
const byte dirpin = 5;    //stepstick direction pin
const byte enable_X = 6;  //stepstick enable pin
const byte enable_Y = 3;  //stepstick 2 enable pin
const byte Photomicrosensor_X = A1; //Omron EE-SX1042 Photomicrosensor X pin
const byte Photomicrosensor_Y = A2; //Omron EE-SX1042 Photomicrosensor Y pin
const byte reset_out_pin = A0;
const byte DS18S20_Pin = A3; //DS18S20
```

Δήλωση στην βιβλιοθήκη της οθόνης LCD ότι θα χρησιμοποιηθεί σύνδεση της οθόνης μέσω Shift Register στις αντίστοιχες θύρες. Προσοχή η επίσημη βιβλιοθήκη δεν υποστηρίζει αυτήν την σύνδεση, έτσι γίνεται χρήση της βιβλιοθήκης New_ LiquidCrystal.

```
LiquidCrystal_SR lcd(A4,A5,TWO_WIRE);
        //                |    |
        //                |    \-- Clock Pin
        //                \---- Data/Enable Pin
```

Έναρξη της βιβλιοθήκης OneWire για την επικοινωνία του αισθητήρα θερμοκρασίας.

```
//Initiate Temperature chip i/o
OneWire  ds (DS18S20_Pin);
```

Ρύθμιση παραμέτρων δικτύου Ethernet.

Είναι απαραίτητο να ορίσουμε μία διεύθυνση mac που θα χρησιμοποιεί ο μικροελεγκτής. Επίσης μπορεί να οριστεί και διεύθυνση ip εάν δεν θέλουμε να δοθεί αυτόματα μέσω DHCP.

```
// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
//IPAddress ip(192, 168, 100, 69);
//IPAddress gateway(192, 168, 100, 254);
//IPAddress subnet(255, 255, 255, 0);
```

Ορίζεται η διεύθυνση ip του Nanostation και δημιουργούνται οι Ethernet Client (σύνδεση προς το Nanostation) και Ethernet Server (σύνδεση του χρήστη στην θύρα 28). Επίσης δημιουργούμε μία κενή μεταβλητή για την ip του χρήστη.

```
// Enter the IP address of the Nanostation you're connecting to:
byte nano[] = {
  192,168,1,20};
// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 23 is default for telnet;
EthernetClient nanostation;
EthernetServer server(28);
EthernetClient client = 0; // Client needs to have global scope so it can be called
// from functions outside of loop, but we don't know
// what client is yet, so creating an empty object
```

Ορίζουμε τις καθολικές μεταβλητές

```
///////////////global variables//////////////
int curStepperPos_X = 0;        // current position of stepper motor (steps)
int possition_X = 0;            // position (degrees)
long curStepperPos_Y = 34740;       // current position of stepper motor (steps)
int possition_Y = 90;           // position (degrees)
byte resetcounter = 0;
float TemperatureSum;
boolean logged = false;     // whether you are logged to nanostation
boolean gotAMessage = false; // whether or not you got a message from the client yet
byte lcd_flag = 1;          // change lcd displayed data
float coordinates[6];

///////////////Timers//////////////////////
unsigned long lcd_timer = 0;//
unsigned long lastAttemptTime = 0;     // last attempt to connect to the server, in
milliseconds
const int requestInterval = 10 * 1000; // delay between connect attempts.
unsigned long client_disconnect_timer = 0; //disconect client after some time inactive
```

Για την αποθήκευση των τιμών της ισχύς σήματος, σε συνδυασμό με την αντίστοιχη γωνία περιστροφής ή κλήσης δημιουργούμε μία νέα μεταβλητή τύπου δομής. Συγκεκριμένα δημιουργούμε δύο πίνακες των 13 στοιχείων. Ο κάθε πίνακας αντιστοιχεί σε έναν άξονα περιστροφής. Το κάθε στοιχείο του πίνακα περιέχει δύο μεταβλητές τύπου integer, όπου η πρώτη μεταβλητή εκφράζει την γωνία και η δεύτερη την ισχύ σήματος.

```
//////////////Data Stuctures////////////////
struct Data{
  int angle;
  int power;
};
Data scan[13];
Data scanY[13];
int signal; //Signal level
```

Έναρξη του Setup

```
void setup() {
```

Με την εντολή pinMode δηλώνουμε την χρήση των θυρών του μικροελεγκτή ως εισόδους ή εξόδους. Επιπρόσθετα με την εντολή digitalWrite θέτουμε την αρχική κατάσταση της θύρας σε υψηλή ή χαμηλή κατάσταση.

```
  pinMode(Photomicrosensor_X,INPUT); //Calibration Sensor
  pinMode(Photomicrosensor_Y,INPUT); //Calibration Sensor
```

Η κάρτα αποθήκευσης SD που υπάρχει στο Ethernet Shield δεν θα χρησιμοποιηθεί. Είναι ωστόσο απαραίτητο να απενεργοποιηθεί θέτοντας την θύρα επιλογής της σε υψηλή στάθμη. Σε διαφορετική περίπτωση το Ethernet μπορεί να μην δουλεύει σωστά.

```
  //explicitly deselect SD card reader
  pinMode(4, OUTPUT);
  digitalWrite(4, HIGH);

  //Setup StepStick
  pinMode(steppin, OUTPUT);
  pinMode(dirpin, OUTPUT);
  pinMode(enable_X, OUTPUT);
  pinMode(enable_Y, OUTPUT);
  digitalWrite(dirpin, HIGH);
  digitalWrite(steppin, LOW);
  digitalWrite(enable_X, HIGH);
  digitalWrite(enable_Y, HIGH);
```

Ορισμός των διαστάσεων και έναρξη της οθόνης LCD, ακολουθούμενη από εμφάνιση μηνύματος καλωσορίσματος.

```
  lcd.begin(16,2);     // initialize the lcd
  lcd.home ();         // go home
  lcd.print("Arduino WiFi");
```

Στο τέλος του Setup γίνεται κλήση της υπορουτίνας connectToNanostation().

```
  //Connect to Ethernet
  void connectToNanostation();
}
```

4.2.2 Συνδέσεις Ethernet (υπορουτίνα connectToNanostation())

Η υπορουτίνα connectToNanostation() είναι υπεύθυνη για την δημιουργία των συνδέσεων δικτύου Ethernet.
Σε πρώτο στάδιο εκκινεί το δίκτυο, χρησιμοποιώντας την διεύθυνση mac που του έχουμε δώσει. Αναμένει από το δίκτυο να του δοθεί διεύθυνση ip και την εμφανίζει στην οθόνη LCD. Εάν υπάρχει κάποιο πρόβλημα εμφανίζει μήνυμα σφάλματος.

```
void connectToNanostation(){

  lcd.setCursor( 0 , 1 );
  lcd.print(F("connecting..."));
  logged = false;

  if(Ethernet.begin(mac) == 0) { // start ethernet using mac & DHCP
    lcd.clear();
    lcd.home();
    lcd.print(F("DHCP Failed"));
    while(true) // no point in carrying on, so stay in endless loop:
      ;
  }
  delay(1000); // give the Ethernet shield a second to initialize

  // print your local IP address:
  lcd.clear();
  lcd.home();
  lcd.print(Ethernet.localIP());
```

Σε δεύτερο στάδιο προσπαθεί να συνδεθεί στην θύρα 23 του Nanostation, στην οποία βρίσκεται ο Telnet Server. Εάν καταφέρει να συνδεθεί, διαβάζει τα δεδομένα όπου θα πρέπει να υπάρχει προτροπή για εισαγωγή ονόματος και κωδικού ασφαλείας. Σε αυτήν την περίπτωση δίνει τα στοιχεία και περιμένει την ταυτοποίηση.
Στην συνέχεια εκκινεί τον Telnet Server που εδρεύει στην μονάδα του μικροελεγκτή, ειδάλλως εμφανίζει μήνυμα σφάλματος και σημειώνει τον τρέχοντα χρόνο.
Τέλος επιστρέφει στον κεντρικό βρόχο(loop).

```
  // if you get a connection, login:
  if (nanostation.connect(nano, 23)>0) {
    // Login to nanostation
    nanostation.print("\n");
    if (nanostation.find("login:"));
    {       //if prompt to login
      nanostation.print("ubnt\n");
    }       //write username
```

```
  if (nanostation.find("Password:"));
  {    //if prompt for password
    nanostation.print("ubnt\n");
  }       //write password
  logged = (nanostation.find("BusyBox")); //if command prompt logged=true
  server.begin();                  //Begin Server
 }
 else {
  // if you didn't get a connection to the server:
  lcd.println(F("connection failed"));
 }
 // note the time of this connect attempt:
 lastAttemptTime = millis();
```

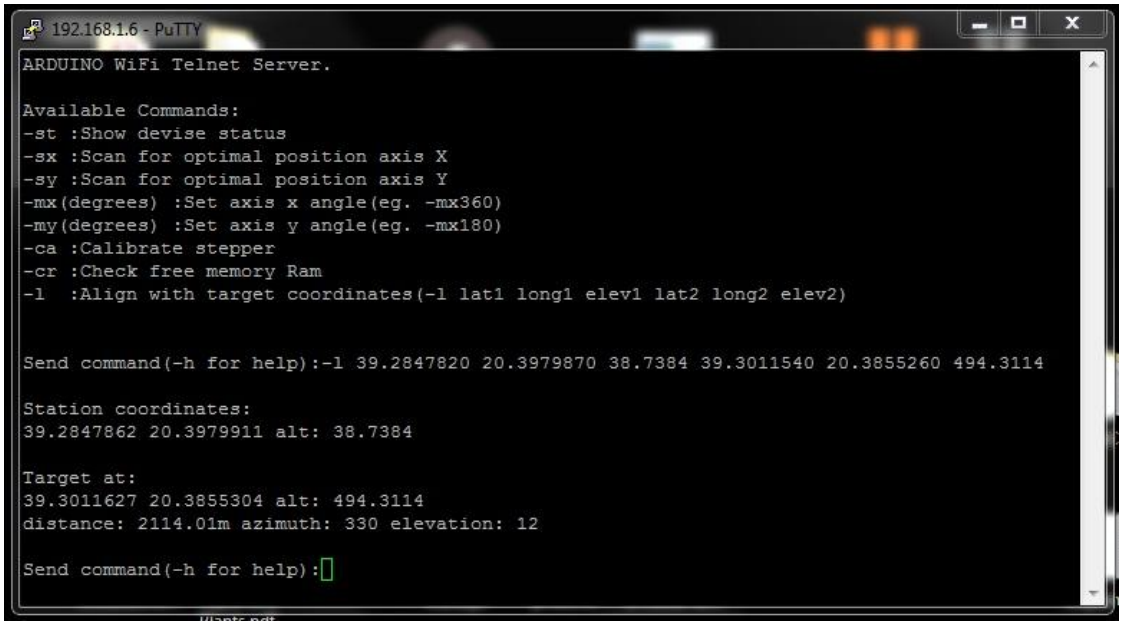
Χρειάστηκε μεγάλη προσπάθεια να βρεθεί, ότι η προεπιλεγμένη ταχύτητα μεταφοράς δεδομένων στον δίαυλο SPI είναι πολύ υψηλή. Αυτό προκαλούσε προβλήματα και ο μικροελεγκτής έπαυε να αποκρίνεται, μετά από κάποιο χρονικό διάστημα. Το πρόβλημα διορθώνει η παρακάτω εντολή, η οποία μειώνει την ταχύτητα στο ένα δέκατο έκτο της ταχύτητας του μικροελεγκτή.

```
 SPI.setClockDivider(SPI_CLOCK_DIV16); //slow down SPI bus (Without this the system hangs after a while)
 }
```

4.2.3 Κεντρικός Βρόχος (loop)

Ο κεντρικός βρόχος του προγράμματος επαναλαμβάνεται διαρκώς, όσο είναι ενεργοποιημένος ο μικροελεγκτής. Η λειτουργίες που θέλουμε να πραγματοποιεί είναι:

- Να ελέγχει εάν ο μικροελεγκτής παραμένει συνδεδεμένος στο δίκτυο Ethernet. Σε διαφορετική περίπτωση να εκτελεί την ρουτίνα σύνδεσης.
- Να ελέγχει εάν ο χρήστης έχει συνδεθεί στον Telnet Server. Τότε να του εμφανίζει το μενού βοήθειας [Εικ. 17].
- Όταν ο χρήστης στείλει μία εντολή να την διαβάζει και να εκτελεί την αντίστοιχη ρουτίνα.
- Τέλος να εκτελεί την ρουτίνα ανανέωσης της οθόνης LCD.



Εικ. 17 : Μενού βοηθείας του Telnet Server και εκτέλεση εντολής

```
void loop()
{
```

Η εντολή Ethernet_maintain στέλνει αίτηση στον διαχειριστή DHCP για διατήρηση της διεύθυνσης ip. Εάν δεν την συμπεριλάβουμε, ανάλογα με τις ρυθμίσεις του δικτύου, υπάρχει περίπτωση αποσύνδεσης. Με την δεύτερη εντολή εξετάζουμε εάν παραμένει η σύνδεση δικτύου. Εάν όχι, το πρόγραμμα προσπερνά τις υπόλοιπες εντολές και πάει στην τελευταία διαδικασία. Σε αυτήν ελέγχει τον χρόνο που έχει περάσει από την τελευταία σύνδεση και εάν είναι μεγαλύτερος από τον προκαθορισμένο εκτελεί την ρουτίνα connectToNanostation().

```
Ethernet.maintain();
if (nanostation.connected() && logged) {
```

Στην περίπτωση που το δίκτυο λειτουργεί, καλείται η ρουτίνα ανανέωσης της οθόνης LCD. Στην συνέχεια ελέγχεται εάν υπάρχει συνδεδεμένος χρήστης. Τότε καλείται η ρουτίνα help()[Εικ. 17] της οποίας ο ρόλος είναι να στείλει ένα μενού βοηθείας στον χρήστη.

```
lcd_update(); // update the lcd display data every 4 second
EthernetClient client = server.available();  // wait for a new client:
// when the client sends the first byte, say hello:
if (client) {
  client_disconnect_timer = millis();
  if (!gotAMessage) {
    gotAMessage = true;
    help();
  }
```

Επόμενο στάδιο είναι ο έλεγχος των εισερχόμενων δεδομένων και η αποκωδικοποίηση των εντολών. Για να διευκολυνθεί η αναγνώριση των εντολών ορίστηκαν κάποιοι κανόνες.

- Όλες οι εντολές θα πρέπει να ξεκινούν με το σύμβολο (-)
- Να αποτελούνται από δύο χαρακτήρες
- Σε περίπτωση που χρειάζεται,να ακολουθούνται από αριθμητικές τιμές.

Έτσι, το πρόγραμμα ελέγχει σειριακά τα εισερχόμενα δεδομένα. Όταν εντοπιστεί το σύμβολο(-) αποθηκεύει τους δύο επόμενους χαρακτήρες. Τότε γίνεται σύγκριση των δύο χαρακτήρων με τις διαθέσιμες εντολές και εάν ταιριάζουν, καλείται η ρουτίνα που αντιστοιχεί στην εντολή.

Στις περιπτώσεις εντολών που απαιτούν την χρήση αριθμητικής τιμής αυτή διαβάζεται με την εντολή parseInt() ή parseFloat. Όλα τα δεδομένα που μεταφέρονται στο δίκτυο Telnet είναι σε μορφή κώδικα ASCII πχ το 0 αντιστοιχεί στον αριθμό 48. Οι εντολές αυτές μετατρέπουν τους ASCII χαρακτήρες σε integer ή float αντίστοιχα.

Μετά την εκτέλεση της ρουτίνας που επέλεξε ο χρήστης, στέλνεται μήνυμα προτροπής για νέα εντολή (command_prompt()). Τότε η διαδικασία loop τερματίζεται και επαναλαμβάνεται στο επόμενο κύκλο.

```
while (client.connected()) {
  if (client.available()) {
    if(client.findUntil("-", "\n")){
      char type1 = client.read();
      char type2 = client.read();
      switch(type1){
      case 's':
        if( type2 == 't'){
          show_status();
        }
        else if (type2 == 'x'){
          optimal();
          command_prompt();
        }
        else if (type2 == 'y'){
          optimal_Y();
          command_prompt();
        }
        break;
      case 'm':
        if (type2 == 'x'){
          int val = client.parseInt();
          moveX(val);
          command_prompt();
        }
        else if (type2 =='y'){
          int val = client.parseInt();
          moveY(val);
          command_prompt();
        }
        break;
      case 'c':
        if (type2 == 'a'){
          Calibrate();
          command_prompt();
        }
        else if (type2 =='r'){
          server.print(F("freeMemory()="));
```

```
            server.println(freeMemory());
            command_prompt();
           }
           break;
         case 'h':
           help();
           break;
         case 'l':
           for(int field=0; field < 6; field++){  //Parse coordinates
            coordinates[field] = client.parseFloat();
           }
           point_location();
           break;
         default:
           server.println(F("Unknown command type -h for help")); //if wrong type
print erroe msg
          }
        }
        break;
      }
     }
    }
    else {
     if (millis() - client_disconnect_timer > 180*1000){  // Disconect client after 3min
of inactivity
      client.stop();
      gotAMessage = false;
     }
    }
  }
```

Εδώ καταλήγει το πρόγραμμα εάν δεν λειτουργούν οι συνδέσεις δικτύου.

```
  else if (millis() - lastAttemptTime > requestInterval) {
   // if you're not connected, and time has passed since
   // your last connection, then attempt to connect again:
   resetcounter++;
   lcd.setCursor(15,1);
   lcd.print(resetcounter);
   if (resetcounter >= 3) {
    digitalWrite(reset_out_pin, HIGH);
   } //Pulling the RESET pin HIGH triggers the reset.
   nanostation.stop();
   connectToNanostation();
  }
 }
```

4.2.4 Βοηθητικές Ρουτίνες του Telnet Server

command_prompt():

Τυπώνει στον Server το μήνυμα προτροπής για εισαγωγή νέας εντολής

```
void command_prompt(){
  server.println();
  server.print(F("Send command(-h for help):"));
}
```

help():

Τυπώνει στον Server την οθόνη βοηθείας. Σε αυτήν περιέχεται το όνομα του Server και όλες οι εντολές που έχει ο χρήστης στην διάθεση του. void help()

```
{
  server.println(F("ARDUINO WiFi Telnet Server."));
  server.println();
  server.println(F("Available Commands:"));
  server.println(F("-st :Show devise status"));
  server.println(F("-sx :Scan for optimal position axis X"));
  server.println(F("-sy :Scan for optimal position axis Y"));
  server.println(F("-mx(degrees) :Set axis x angle(eg. -mx360)"));
  server.println(F("-my(degrees) :Set axis y angle(eg. -mx180)"));
  server.println(F("-ca :Calibrate stepper"));
  server.println(F("-cr :Check free memory Ram"));
  server.println(F("-l  :Align with target coordinates(-l lat1 long1 elev1 lat2 long2
elev2)"));
  server.println();
  command_prompt();
}
```

show_status():

Τυπώνει στον Server την κατάσταση της συσκευής. Δηλαδή την ισχύ σήματος της κεραίας, την θέση των αξόνων και την θερμοκρασία.

```
void show_status(){
  getSignal();
  getTemp();
  server.print(F("Signal: "));
  server.print(signal);
  server.print(F("db |"));
  server.print(F(" axis X: "));
  server.print(possition_X);
  server.print(F(" axis Y: "));
  server.print(possition_Y);
  server.print(F(" (degree) | temp: "));
  server.print(TemperatureSum,1);
  server.println("C");
  command_prompt();
}
```

4.2.5 Κίνηση

Για να ελέγξουμε τους οδηγούς των βηματικών κινητήρων χρειάζονται τρία σήματα ελέγχου [Εικ. 11-12]. Το πρώτο σήμα προέρχεται από την θύρα 5 την οποία στο πρόγραμμα ονομάζουμε dirpin. Αυτή συνδέεται στους οδηγούς και των δύο κινητήρων. Αναλόγως την στάθμη της (υψηλή ή χαμηλή) επιλέγει την φορά περιστροφής του κινητήρα (δεξιόστροφη αριστερόστροφη). Το δεύτερο σήμα προέρχεται από τις θύρες 6 και 3, τις οποίες ονομάζουμε enable_X και enable_Y αντίστοιχα. Η κάθε μία συνδέεται στην επαφή enable ενός οδηγού, και όταν έχει χαμηλή στάθμη ενεργοποιεί τον εκάστοτε οδηγό. Το τελευταίο σήμα προέρχεται από την θύρα 7, που ονομάζουμε steppin. Σε αυτήν τη θύρα δημιουργούμε παλμούς, όπου ο κάθε παλμός αντιστοιχεί σε ένα βήμα του κινητήρα. Μεταβάλλοντας την συχνότητα των παλμών μπορούμε να ρυθμίσουμε την ταχύτητα περιστροφής.

Οπότε η διαδικασία που πρέπει να εξακολουθήσουμε είναι η εξής:

- Επιλέγουμε και ενεργοποιούμε τον βηματικό κινητήρα θέτοντας το enable_X ή enable_Y σε χαμηλή στάθμη.
- Επιλέγουμε την φορά περιστροφής θέτοντας το dirpin σε χαμηλή ή υψηλή στάθμη.
- Δημιουργούμε παλμούς στο steppin, όσους και τα βήματα που θέλουμε να πραγματοποιήσει ο βηματικός κινητήρας.

Η ρουτίνα:

Είσοδος της ρουτίνας είναι μία μεταβλητή, που εκφράζει την θέση στην οποία θέλουμε να κινηθεί ο άξονας. Η θέση μετριέται σε μοίρες από την αρχική θέση του άξονα περιστροφής.

```
void moveX(int possitionX)
{
  int i, j, steps;
  i = 1800;                //Set start speed
```

Αρχικά θέτουμε ένα όριο στις δυνατές τιμές, ώστε να αποκλείσουμε την κίνηση σε μη επιθυμητές γωνίες.

```
  possitionX = constrain(possitionX, 0, 360);
  possition_X = possitionX;   //save position to a global variable for other use
```

Η εντολή map κάνει μία αντιστοίχιση των μοιρών περιστροφής σε βήματα που πρέπει να εκτελέσει ο κινητήρας.

```
  digitalWrite(enable_X, LOW);                  //enabling stepper
  int posValue = map(possitionX, 0, 360, 0, 4320); //map position from degrees to
steps
```

Έχοντας αποθηκευμένη σε μία μεταβλητή την τελευταία θέση στην οποία βρισκόταν ο άξονας, μπορούμε να υπολογίσουμε:

Την φορά κίνησης που πρέπει να εξακολουθήσουμε για να βρεθούμε στην νέα θέση και τα βήματα που απαιτούνται.
Για να πραγματοποιηθεί η κίνηση έχουμε δύο περιπτώσεις. Στην πρώτη ο κινητήρας κινείται δεξιόστροφα και στην δεύτερη αριστερόστροφα.

```
// moving to position
if (posValue > curStepperPos_X){
  steps = posValue - curStepperPos_X;
  curStepperPos_X =curStepperPos_X + steps;
  digitalWrite(dirpin, LOW);
```

Για κάθε παλμό που θέλουμε να δημιουργήσουμε θέτουμε σε υψηλή στάθμη το steppin για 2μs. Στην συνέχεια το επαναφέρουμε σε χαμηλή και εισάγουμε μία καθυστέρηση. Η καθυστέρηση είναι αντιστρόφως ανάλογη της ταχύτητας περιστροφής.
Για να ελέγξουμε την επιτάχυνση, ξεκινώντας θέτουμε μία καθυστέρηση 1800μs την οποία σταδιακά μειώνουμε έως ότου φτάσουμε την επιθυμητή ταχύτητα.

```
  for (j=0; j<steps; j++)
  {
    digitalWrite(steppin, HIGH);
    delayMicroseconds(2);
    digitalWrite(steppin, LOW);
    delayMicroseconds(i);
    if (steps - j < 200){
      i=i+6;
    }
    else {
      i=i-10;
    }
    i=constrain(i,1000,1800);
  }
}
else if (posValue < curStepperPos_X){
  steps = curStepperPos_X - posValue;
  curStepperPos_X =curStepperPos_X - steps;
  digitalWrite(dirpin, HIGH);

  for (j=0; j<steps; j++)
  {
    digitalWrite(steppin, HIGH);
    delayMicroseconds(2);
    digitalWrite(steppin, LOW);
    delayMicroseconds(i);
    if (steps - j < 200){
      i=i+6;
    }
    else {
      i=i-10;
    }
    i=constrain(i,1000,1800);
```

Στο τέλος απενεργοποιούμε τον οδηγό του βηματικού κινητήρα με μία μικρή καθυστέρηση. Η καθυστέρηση φρενάρει ηλεκτρικά τον βηματικό κινητήρα.

```
    }
  }
  delay(100);
  digitalWrite(enable_X, HIGH);      //disabling stepper
}
```

Η ρουτίνα που περιγράφηκε αφορά τον άξονα X. Στον άξονα Y η ρουτίνα είναι ελαφρώς τροποποιημένη. Οι σημαντικότερες διαφορές είναι στα ονόματα των μεταβλητών, τον αριθμό βημάτων για μία πλήρη περιστροφή του άξονα και την απουσία ρύθμισης της επιτάχυνσης.

```
void moveY(int possitionY)
{
  int i;
  long j, steps;
  i = 680;                //Set start speed
  possitionY = constrain(possitionY, 0, 180);
  possition_Y = possitionY;   //save position to a global variable for other use
  digitalWrite(enable_Y, LOW);                  //enabling stepper
  long posValue = map(possitionY, 0, 180, 0, 69480); //map position from degrees to
steps

  // moving to possition
  if (posValue > curStepperPos_Y){
    steps = posValue - curStepperPos_Y;
    curStepperPos_Y =curStepperPos_Y + steps;
    digitalWrite(dirpin, HIGH);
    for (j=0; j<steps; j++)
    {
      digitalWrite(steppin, HIGH);
      delayMicroseconds(2);
      digitalWrite(steppin, LOW);
      delayMicroseconds(i);
    }
  }
  else if (posValue < curStepperPos_Y){
    steps = curStepperPos_Y - posValue;
    curStepperPos_Y =curStepperPos_Y - steps;
    digitalWrite(dirpin, LOW);
    for (j=0; j<steps; j++)
    {
      digitalWrite(steppin, HIGH);
      delayMicroseconds(2);
      digitalWrite(steppin, LOW);
      delayMicroseconds(i);
    }
  }
  delay(100);
```

4.2.6 Είσοδοι (ισχύς σήματος και θερμοκρασία)

Ισχύς σήματος:

Η ισχύς σήματος λαμβάνεται μέσω του Telnet Server στο Nanostation. Το λειτουργικό σύστημα του Nanostation ονομάζεται AirOs και είναι μία διανομή Linux. Οπότε εκτελώντας την εντολή iwconfig μπορούμε να δούμε τις πληροφορίες που μας ενδιαφέρουν. Πιο συγκεκριμένα η εντολή που χρησιμοποιούμε είναι η iwconfig |grep Signal με την οποία περιορίζουμε τα αποτελέσματα.

```
int getSignal(){
```

Πρώτα ελέγχουμε εάν λειτουργεί η σύνδεση δικτύου.

```
 if (nanostation.connected()) {
   signal = 0;
//    int freeSpace = nanostation.free();
```

Στην συνέχεια εκτελούμε την εντολή iwconfig |grep Signal και αναζητούμε την γραμμή Signal level=-. Εάν βρεθεί διαβάζουμε και αποθηκεύουμε την τιμή ισχύος σήματος. Για να αδειάσει η προσωρινή μνήμη buffer. Διαβάζουμε και τα υπόλοιπα εισερχόμενα δεδομένα χωρίς να τα αποθηκεύσουμε.

```
   int write_b = nanostation.println(F("iwconfig |grep Signal"));
   delay(1);
   if (nanostation.find("Signal level=-"));
   {
    signal = nanostation.parseInt();
    while (nanostation.available() >0){
     nanostation.read();
    }
   }
   nanostation.flush();
   return signal;
  }
```

Εάν το δίκτυο δεν λειτουργεί εκτελούμε την ρουτίνα σύνδεσης

```
  else{
   lcd.setCursor(15,1);
   lcd.print(resetcounter);
   nanostation.stop();
   connectToNanostation();
  }
}
```

Θερμοκρασία:

Ο αισθητήρας θερμοκρασίας επικοινωνεί μέσω του πρωτοκόλλου 1-Wire κάνοντας χρήση της βιβλιοθήκης OneWire.h . Ο παρακάτω κώδικας αναζητεί την διεύθυνση του αισθητήρα

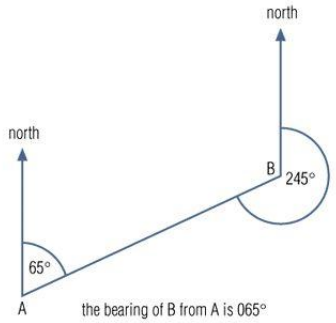
ds18b20, στην συνέχεια διαβάζει την θερμοκρασία και την επιστρέφει σαν μία μεταβλητή τύπου float. Η θερμοκρασία είναι σε βαθμούς κελσίου.

```
float getTemp(){
  //returns the temperature from one DS18S20 in DEG Celsius
  byte data[12];
  byte addr[8];

  if ( !ds.search(addr)) {
    //no more sensors on chain, reset search
    ds.reset_search();
    return -1000;
  }
  if ( OneWire::crc8( addr, 7) != addr[7]) {
    return -1000;
  }
  if ( addr[0] != 0x10 && addr[0] != 0x28) {
    return -1000;
  }

  ds.reset();
  ds.select(addr);
  ds.write(0x44,1); // start conversion, with parasite power on at the end
  byte present = ds.reset();
  ds.select(addr);
  ds.write(0xBE); // Read Scratchpad
  for (int i = 0; i < 9; i++) { // we need 9 bytes
    data[i] = ds.read();
  }
  ds.reset_search();
  byte MSB = data[1];
  byte LSB = data[0];
  float tempRead = ((MSB << 8) | LSB); //using two's compliment
  TemperatureSum = tempRead / 16;
  return TemperatureSum;
}
```

Υποσημείωση: *περισσότερες πληροφορίες υπάρχουν στην ιστοσελίδα* *http://bildr.org/2011/07/ds18b20-arduino*

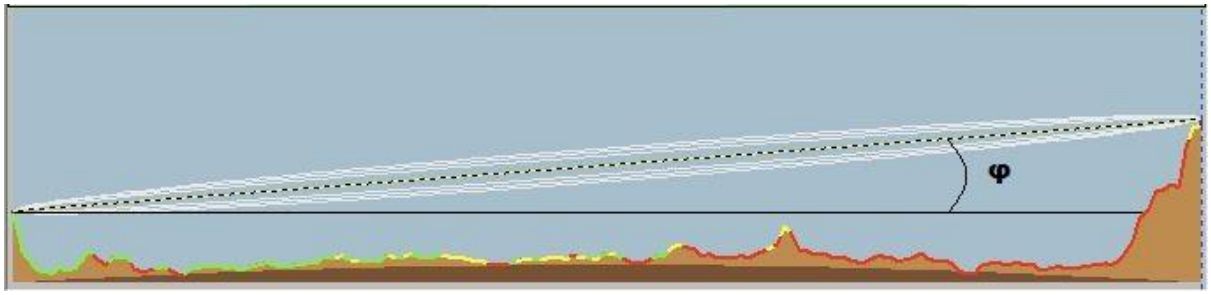
4.2.7 Στόχευση Κεραίας με Χρήση Γεωγραφικών Συντεταγμένων

Κατά την στόχευση της κεραίας με βάση τις γεωγραφικές συντεταγμένες θα πρέπει να υπολογίσουμε δύο γωνίες. Η πρώτη γωνία εκφράζει την απόκλιση της νοητής ευθείας που ενώνει τα δύο σημεία, σε σχέση με τον βορά [Εικ. 17 : Αζιμούθιο]. Σύμφωνα με την γωνία αυτή περιστρέφουμε την κεραία στον άξονα X.



Εικ. 18 : Αζιμούθιο

Η δεύτερη γωνία οφείλεται στην υψομετρική διαφορά που μπορεί να έχουν τα δύο σημεία [Εικ. 19] . Σύμφωνα με αυτήν περιστρέφουμε τον άξονα Y.



Εικ. 19 : Γωνία φ λόγο υψομετρικής διαφοράς των δύο σημείων

Για την στόχευση χρησιμοποιείται η ρουτίνα **point_location()** η οποία χρησιμοποιεί τρείς υπορουτίνες. Αυτές είναι οι:

1. **azimuth:** υπολογισμός αζιμούθιου
2. **differencial_altitude_angle:** υπολογισμός γωνίας λόγο διαφοράς υψομέτρου
3. **distance_between:** υπολογισμός απόστασης μεταξύ των σημείων

Η ρουτίνα point_location αναλαμβάνει να στείλει τα δεδομένα που έχει δώσει ο χρήστης, και αφορούν τις γεωγραφικές συντεταγμένες, στις υπορουτίνες που θα κάνουν τους απαραίτητους υπολογισμούς. Τα δεδομένα αυτά είχαν αποθηκευθεί από την ρουτίνα loop στον πίνακα coordinates με συγκεκριμένη σειρά.

```
void point_location(){
  // point axis x
   int x = azimuth(coordinates[0], coordinates[1], coordinates[3], coordinates[4]) + 0.5;
// we add 0.5 to round float to int

 // pont axis y
```

```
  float distance = distance_between(coordinates[0], coordinates[1], coordinates[3],
coordinates[4], 1);
  int y = differencial_altitude_angle(coordinates[2], coordinates[5], distance) + 0.5;
```

Αφού γίνουν οι υπολογισμοί, τα αποτελέσματα εκτυπώνονται στον server και εκτελούνται οι ρουτίνες που περιστρέφουν του άξονες.

```
  server.println();
  server.println(F("Station coordinates:"));
  server.print(coordinates[0], 7);
  server.print(F(" "));
  server.print(coordinates[1],7);
  server.print(F(" alt: "));
  server.println(coordinates[2],4);
  server.println();
  server.println(F("Target at: "));
  server.print(coordinates[3],7);
  server.print(F(" "));
  server.print(coordinates[4],7);
  server.print(F(" alt: "));
  server.println(coordinates[5],4);
  server.print(F("distance: "));
  server.print(distance);
  server.print(F("m azimuth: "));
  server.print(x);
  server.print(F(" elevation: "));
  server.println(y);
  command_prompt();

  moveX(x);
  moveY(y + 90); // servo center is at 90 degrees
}
```

Η πρώτη υπορουτίνα που καλείται είναι αυτή που υπολογίζει το αζιμούθιο και είναι γραμμένη από τον Maarten Lamers (http://www.maartenlamers.com). Δέχεται ως εισόδους το γεωγραφικό πλάτος και μήκος των δύο σημείων και επιστρέφει το αζιμούθιο σε μοίρες. Επειδή το αποτέλεσμα έχει δεκαδικά ψηφία, στην παραπάνω ρουτίνα τα στρογγυλοποιούμε.

```
float azimuth (float lat1, float long1, float lat2, float long2) {
  // returns initial course in degrees (North=0, West=270) from
  // position 1 to position 2, both specified as signed decimal-degrees
  // latitude and longitude.
  float dlon = radians(long2-long1);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  float a1 = sin(dlon) * cos(lat2);
  float a2 = sin(lat1) * cos(lat2) * cos(dlon);
  a2 = cos(lat1) * sin(lat2) - a2;
  a2 = atan2(a1, a2);
  if (a2 < 0.0) {
```

```
    a2 += TWO_PI;                    // modulo operator doesn't seem to work on
floats
  }
  return degrees(a2);
}
```

Η δεύτερη υπορουτίνα επίσης γραμμένη από τον Maarten Lamers επιστρέφει την απόσταση μεταξύ των δύο σημείων. Για τον υπολογισμό χρησιμοποιείται το θεώρημα great-circle για το οποίο πληροφορίες υπάρχουν στην ιστοσελίδα http://en.wikipedia.org/wiki/Great-circle_distance . Το αποτέλεσμα μπορεί να είναι σε μέτρα ή σε άλλη μονάδα, σύμφωνα με την τιμή της μεταβλητής units_per_meter.

```
float distance_between (float lat1, float long1, float lat2, float long2, float
units_per_meter) {
  // returns distance in meters between two positions, both specified
  // as signed decimal-degrees latitude and longitude. Uses great-circle
  // distance computation for hypothised sphere of radius 6372795 meters.
  // Because Earth is no exact sphere, rounding errors may be upto 0.5%.
  float delta = radians(long1-long2);
  float sdlong = sin(delta);
  float cdlong = cos(delta);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  float slat1 = sin(lat1);
  float clat1 = cos(lat1);
  float slat2 = sin(lat2);
  float clat2 = cos(lat2);
  delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
  delta = sq(delta);
  delta += sq(clat2 * sdlong);
  delta = sqrt(delta);
  float denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
  delta = atan2(delta, denom);
  return delta * 6372795 * units_per_meter;
}
```

Τέλος υπολογίζουμε την γωνία που θα πρέπει να έχει ο άξονας Y. Η γωνία αυτή ισούται με την εφαπτομένη της υψομετρικής διαφοράς και της απόστασης των σημείων. Για αποστάσεις μικρότερες των 30 χιλιομέτρων η καμπύλη της γης θεωρείται αμελητέα και δεν την υπολογίζουμε.

```
float differencial_altitude_angle (float elev1, float elev2, float distance) {
  // returns y axis angle based on altitude of two locations and the
  // distance between them.
  float dif = elev2 - elev1; //calculate the altitude difference
  float w = atan2(dif,distance);
  return degrees (w);
}
```

4.2.8 Στόχευση Κεραίας με Χρήση Γεωγραφικών Συντεταγμένων

Για να βρεθεί η βέλτιστη θέση ακολουθείται η παρακάτω διαδικασία, ξεχωριστά για κάθε άξονα. Το πρώτο βήμα είναι να ληφθούν τιμές ισχύος για διάφορες γωνίες του άξονα περιστροφής. Περιστρέφουμε τον άξονα σε βήματα των 60 μοιρών όπου διαβάζουμε την τιμή ισχύος. Οι τιμές αποθηκεύονται σε έναν πίνακα μεταβλητών τύπου δομής. Οι μεταβλητές που έχουμε κατασκευάσει έχουν δύο στοιχεία, την ισχύ σήματος και την αντίστοιχη γωνία περιστροφής σε μοίρες. Στην συνέχεια γίνεται ταξινόμηση του πίνακα ως προς την καλύτερη τιμή. Ακολουθεί μία δεύτερη σάρωση ανάμεσα στις δύο καλύτερες τιμές. Ο νέος πίνακας που προκύπτει ταξινομείται ξανά και η κεραία μετακινείται στην νέα καλύτερη θέση.

Η λειτουργία της υπορουτίνας scan_function είναι η σάρωση του άξονα X και στην συνέχεια η ταξινόμηση του πίνακα δεδομένων που συλλέχθηκε. Είσοδοι της υπορουτίνας είναι η θέση από την οποία θέλουμε να ξεκινήσει η σάρωση, η τελική θέση, το βήμα που θα ακολουθήσει καθώς και το μέγεθος του πίνακα που θα παραχθεί.

```
void scan_function(int pos1, int pos2, int scan_step, byte matrix_size){
 byte n=0;
 for (pos1; pos1 <= pos2;  pos1 += scan_step )
```

Για κάθε βήμα η κεραία κινείται στην κατάλληλη θέση και αφήνει λίγο χρόνο για την σταθεροποίηση του σήματος. Στην συνέχεια το διαβάζει και το αποθηκεύει μαζί με την αντίστοιχη γωνία περιστροφής στην οποία βρίσκεται.

```
{
  moveX(pos1);
  delay(5000);
  scan[n].angle = pos1;
  scan[n].power = getSignal();
  server.print (scan[n].angle);
  server.print (" ");
  server.print (scan[n].power);
  server.println();
  n++;
 }
```

Ο πίνακας που έχει δημιουργηθεί ταξινομείται με την εντολή qsort. Η qsort (Quicksort ή partition-exchange sort) είναι ένας αλγόριθμος φθίνουσας ή αύξουσας ταξινόμησης, με πολύ καλά αποτελέσματα στην ταχύτητα αλλά και την χαμηλή υπολογιστική ισχύ που απαιτεί.

```
 qsort (scan, matrix_size, sizeof(struct Data), sort);
 server.println(F("qsorted array"));
 for (n=0; n < matrix_size; n++){
  server.print (scan[n].angle);
  server.print (" ");
  server.println (scan[n].power);
 }
}
```

Η qsort χρειάζεται μία βοηθητική ρουτίνα που υποδεικνύει τον τρόπο με τον οποίο θέλουμε να γίνει η σύγκριση των δεδομένων.

```
int sort(const void *x, const void *y)
{
  const struct Data
    *data1 = (struct Data *)x,
    *data2 = (struct Data *)y;
  if ( data1->power > data2->power ) return 1;
  if ( data1->power < data2->power ) return -1;
  return 0;
}
```

Υποσημείωση: *περισσότερες πληροφορίες για την qsort υπάρχουν στις ιστοσελίδες* *http://en.wikipedia.org/wiki/Quicksort* *και* *http://www.cplusplus.com/reference/cstdlib/qsort*

Η υπορουτίνα optimal εκτελεί την λειτουργία ανεύρεσης της βέλτιστης θέσης. Αρχικά εκτελεί μία σάρωση με τιμές από 0 έως 300 μοίρες. Επειδή οι 0 με τις 360 μοίρες συμπίπτουν είναι περιττό να διαβάσουμε την ισχύ στις 360. Στην συνέχεια διαβάζει τα δύο καλύτερα αποτελέσματα και εκτελεί μία πιο ακριβή σάρωση ανάμεσα σε αυτά. Σε αυτό το σημείο θα πρέπει να προσέξουμε ότι εάν τα αποτελέσματα είναι 0 και 300 μοίρες θα πρέπει η σάρωση να γίνει στο τμήμα 300 έως 360 μοίρες. Δηλαδή να έχει εύρος 60 μοίρες και όχι 0 έως 300 όπου το εύρος είναι 300 μοίρες. Για τον λόγο αυτό γίνεται μία σύγκριση των αποτελεσμάτων με αυτές τις τιμές καθώς και η απαραίτητη διόρθωση εάν χρειάζεται.

```
void optimal(){
  moveY(90);
  server.println(F("X axis First Scan Array"));
  scan_function (0, 300, 60, 6);
  int pos1,pos2;
  if (scan[0].angle==0 && scan[1].angle == 300){
    scan[0].angle=360;
  }
  if (scan[1].angle==0 && scan[0].angle == 300){
    scan[1].angle=360;
  }
```

Το εύρος στην δεύτερη σάρωση θα είναι η διαφορά των δύο καλύτερων θέσεων διαιρεμένη σε έξη βήματα.

```
  int scan_step = (abs(scan[1].angle - scan[0].angle))/6;
  pos1 = min(scan[0].angle, scan[1].angle);
  pos2 = max(scan[0].angle, scan[1].angle);
  server.println(F("Second Scan"));
  server.print(F("Scan Step: "));
  server.println(scan_step);
  server.print(F("Start position: "));
  server.println(pos1);
  server.print(F("End position: "));
  server.println(pos2);
```

```
 server.println(F("Array"));
 scan_function (pos1, pos2, scan_step, 7);
 moveX(scan[0].angle);
}
```

Τέλος εκτυπώνονται τα αποτελέσματα στον Server και η κεραία κινείται στην νέα καλύτερη θέση.

Οι αντίστοιχες υπορουτίνες υπάρχουν και για τον άξονα Y όπου αλλάζουν απλά οι ρυθμίσεις.

```
void optimal_Y(){
 server.println(F("Y axis First Scan Array"));
 scan_function_Y(30, 150, 20, 7);
 int pos1,pos2;
 int scan_step = (abs(scanY[1].angle - scanY[0].angle))/5;
 pos1 = min(scanY[0].angle, scanY[1].angle);
 pos2 = max(scanY[0].angle, scanY[1].angle);
 server.println(F("Second Scan"));
 server.print(F("Scan Step: "));
 server.println(scan_step);
 server.print(F("Start position: "));
 server.println(pos1);
 server.print(F("End position: "));
 server.println(pos2);
 server.println(F("Array"));
 scan_function_Y (pos1, pos2, scan_step, 6);
 moveY(scanY[0].angle);
}
void scan_function(int pos1, int pos2, int scan_step, byte matrix_size){
 byte n=0;
 for (pos1; pos1 <= pos2;  pos1 += scan_step )
 {
  moveX(pos1);
  delay(5000);
  scan[n].angle = pos1;
  scan[n].power = getSignal();
  server.print (scan[n].angle);
  server.print (" ");
  server.print (scan[n].power);
  server.println();
  n++;
 }
 qsort (scan, matrix_size, sizeof(struct Data), sort);
 server.println(F("qsorted array"));
 for (n=0; n < matrix_size; n++){
  server.print (scan[n].angle);
  server.print (" ");
  server.println (scan[n].power);
 }}
```

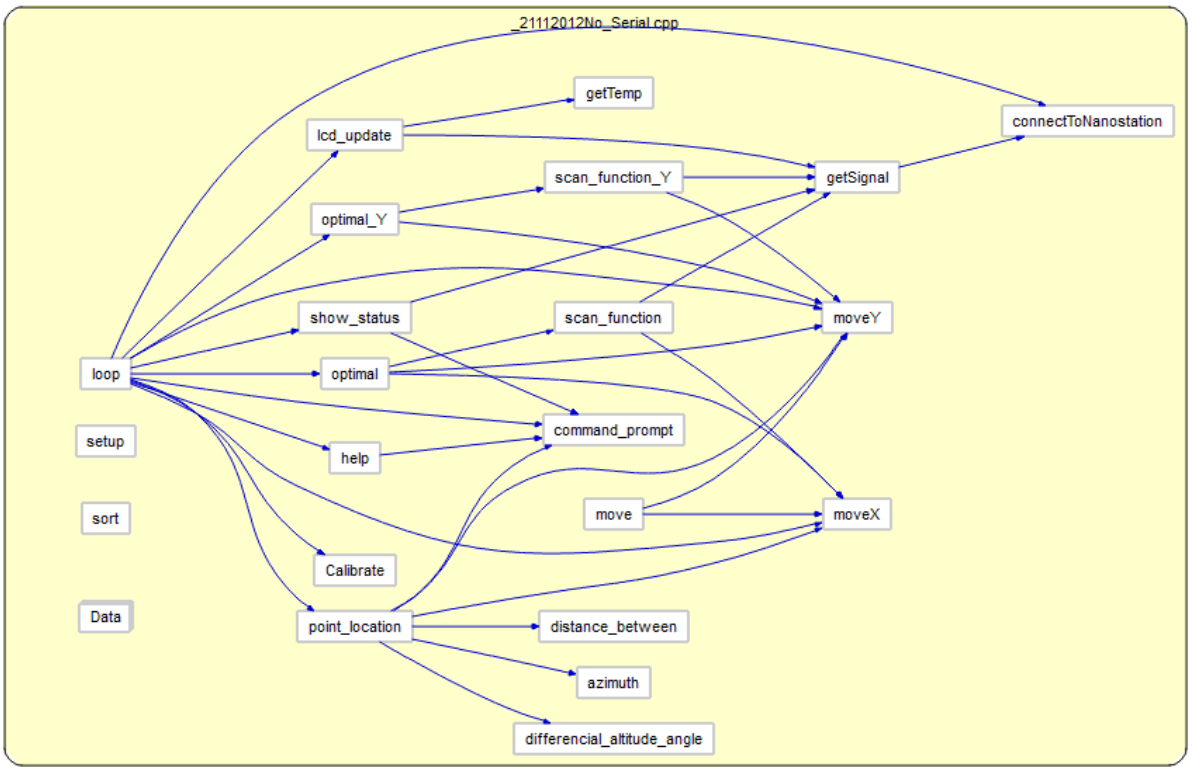
4.2.9 Πληροφορίες Οθόνης LCD

Η υπορουτίνα ανανέωσης των πληροφοριών στην οθόνη LCD χρησιμοποιεί έναν timer και έτσι καλείται κάθε 4 δευτερόλεπτα. Υπάρχει η δυνατότητα να εναλλάσσουμε ομάδες δεδομένων που θα προβάλλονται εναλλάξ. Για να γίνει αυτό χρησιμοποιείται η εντολή switch και μία σημαία που αλλάζει κάθε φορά που έχει εκτελεστεί μία ομάδα, ώστε να δείχνει προς μία άλλη. Η διεύθυνση ip που έχει τυπωθεί στην οθόνη είναι σημαντική και για τον λόγο αυτό δεν σταματά να προβάλλεται. Τα υπόλοιπα δεδομένα που ανανεώνονται είναι η θερμοκρασία, η ισχύς του σήματος, η θέση των αξόνων και οποιαδήποτε άλλη πληροφορία θα θέλαμε να προσθέσουμε.

```
void lcd_update()
{
  if(millis() - lcd_timer > 4000) {
    lcd_timer = millis();
    switch (lcd_flag){
    case 1:
      getTemp();
      getSignal();
      lcd.clear();
      lcd.print(Ethernet.localIP());
      lcd.setCursor(0,1);
      lcd.print("X:");
      lcd.print(possition_X);
      lcd.print(" Y:");
      lcd.print(possition_Y);
      lcd.setCursor(11,1);
      lcd.print(TemperatureSum,1);
      lcd.print("C");
      lcd.setCursor(12,0);
      lcd.print(signal);
      lcd.print("db");
     // lcd_flag = 2; //uncomment to display second screen
      break;
    case 2:
      lcd.clear();
      lcd.print(millis() / 60000);
      lcd.print("min");
      lcd_flag = 1;
      break;
    }
  }
}
```

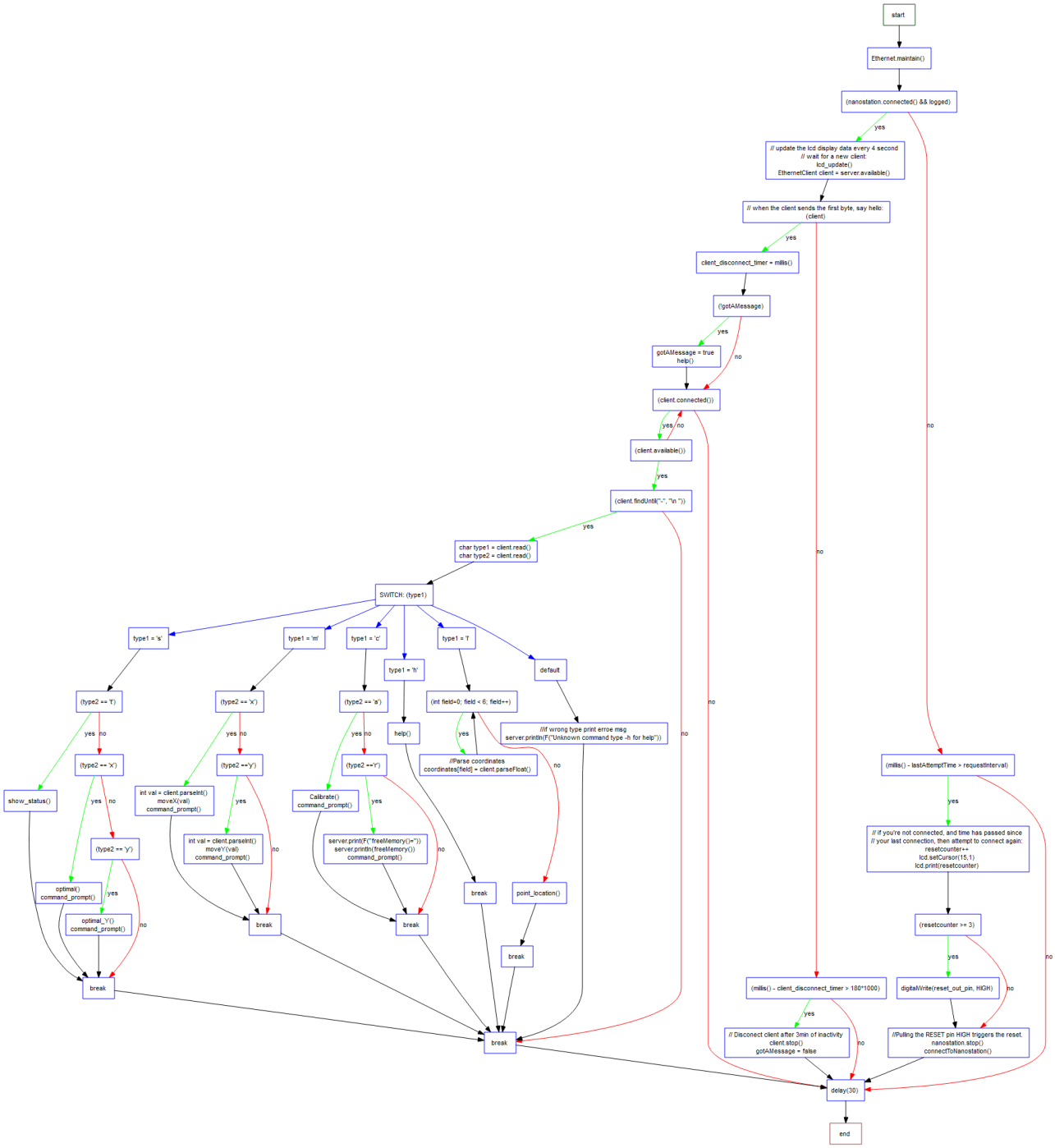
4.3 Αναθεώρηση του Κώδικα Διαγράμματα Ροής

Το διάγραμμα κλήσης ρουτινών [Εικ. 20] δείχνει αναλυτικά το πώς αλληλεπιδρούν οι ρουτίνες μεταξύ τους. Παρατηρούμε ότι ο κεντρικός βρόγχος loop καλεί μία σειρά από υπορουτίνες οι οποίες για να λειτουργήσουν καλούν τις υπόλοιπες.



Εικ. 20 : Διάγραμμα κλήσης ρουτινών

Παρακάτω φαίνεται [Εικ. 21] το διάγραμμα ροής του κεντρικού βρόγχου loop.



Εικ. 21 : Διάγραμμα ροής του κεντρικού βρόγχου loop

ΚΕΦΑΛΑΙΟ 5 : ΣΥΜΠΕΡΑΣΜΑ

5.1 Αποτελέσματα

Επαγωγικά σκεπτόμενοι φτάσαμε στην υλοποίηση του στόχου μας, ο οποίος είναι η κατασκευή ενός συστήματος κίνησης κεραίας WiFi σε δύο άξονες και η αυτόματη εύρεση της βέλτιστης θέσης, με βάση την ισχύ σήματος ή τις γεωγραφικές συντεταγμένες δύο σημείων. Το σύστημα δουλεύει σταθερά για μεγάλο χρονικό διάστημα χωρίς να παρουσιάζει εμπλοκές. Υπάρχει μεγάλη ακρίβεια στην ανεύρεση του σωστού σημείου στόχευσης προσδιοριζόμενο από τις γεωγραφικές συντεταγμένες. Ωστόσο όσον αφορά την ανεύρεση του σωστού σημείου με βάση την ισχύ, επειδή η κεραία μας είναι τύπου πάνελ και δεν είναι πολύ κατευθυντική, μπορεί να υπάρχει κάποια μικρή απόκλιση. Σε αυτό συμβάλει στο ότι η έρευνα μας έγινε ως επί το πλείστον σε κλειστό χώρο ή σε εξωτερικό αστικό περιβάλλον. Δηλαδή ο περιβάλλον χώρος δημιουργούσε ανακλάσεις και παρεμβολές στο σήμα.

Θα ήταν παράλειψη εάν δεν αναφέραμε ότι η σωστή λειτουργία του συστήματος βασίστηκε στην άψογη λειτουργία του μικροελεγκτή και των υποσυστημάτων του, καθώς και στην μεγάλη ακρίβεια και αξιοπιστία που προσέφεραν οι βηματικοί κινητήρες και το σύστημα μετάδοσης με γρανάζια.

5.2 Προτάσεις για Αναβάθμιση

Η πρώτη αλλαγή που θα μπορούσε να γίνει είναι να χρησιμοποιηθεί μία πιο κατευθυντική κεραία έτσι ώστε να έχουμε μεγαλύτερη ακρίβεια στην εύρεση της βέλτιστης θέσης. Με περισσότερο κεφάλαιο και την ανάλογη επιπλέον εργασία θα μπορούσαμε να προσθέσουμε αρχικά έναν αισθητήρα GPS έτσι ώστε να αυτοματοποιήσουμε την εισαγωγή των γεωγραφικών συντεταγμένων της κεραίας. Επιπλέον μπορούμε να προσθέσουμε μία ψηφιακή πυξίδα ώστε το σύστημα να προσανατολίζεται με τον μαγνητικό βορά. Τέλος ένας γυροσκοπικός αισθητήρας τριών αξόνων θα μας εξασφάλιζε πιο εύκολη ευθυγράμμιση του συστήματος με το επίπεδο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

ΠΡΑΚΤΙΚΟΣ ΟΔΗΓΟΣ ΚΕΡΑΙΩΝ WiFi Σωκράτης Πανουσίου (socrates.. @.. awmn.net)(2002-2007: Athens Wireless Metropolitan Network)

IEEE Standard Definitions of Terms for Antennas, IEEE Std 145-1983

Getting Started with Arduino By Massimo Banzi, Publisher: O'Reilly Media / Make, December 2008

Making Things Talk Practical Methods for Connecting Physical Objects, By Tom Igoe, Publisher: O'Reilly Media / Make, September 2007

Arduino Cookbook, 2nd Edition Recipes to Begin, Expand, and Enhance Your Projects, By Michael Margolis, Publisher: O'Reilly Media, December 2011

http://www.arduino.cc 15/02/2013

http://playground.arduino.cc 15/02/2013

http://reprap.org/wiki/StepStick 15/02/2013

http://bildr.org/2011/07/ds18b20-arduino 15/02/2013

http://www.maartenlamers.com 15/02/2013

http://en.wikipedia.org/wiki/Great-circle_distance 15/02/2013

http://en.wikipedia.org/wiki/Quicksort 15/02/2013

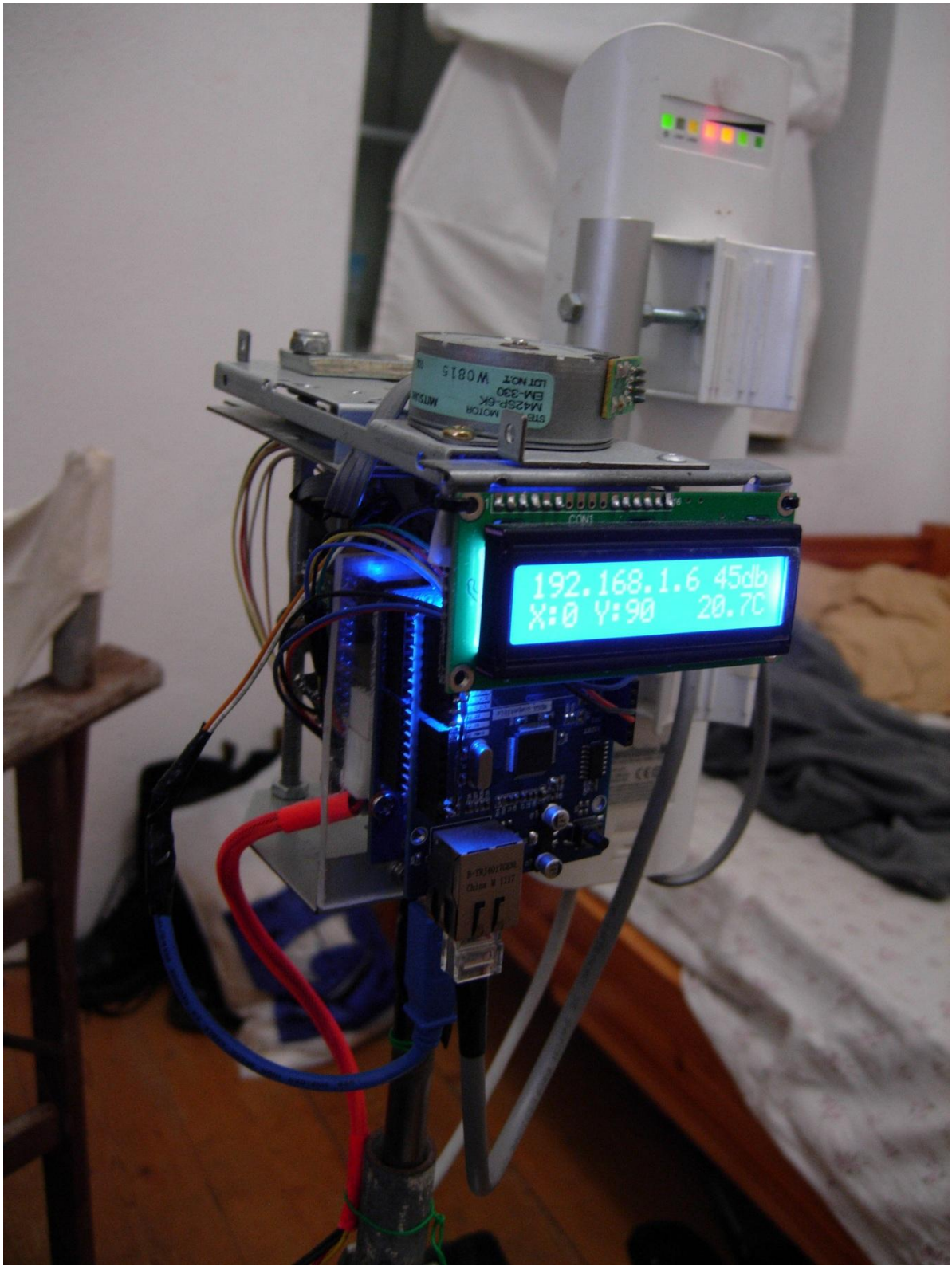
http://www.cplusplus.com/reference/cstdlib/qsort 15/02/2013

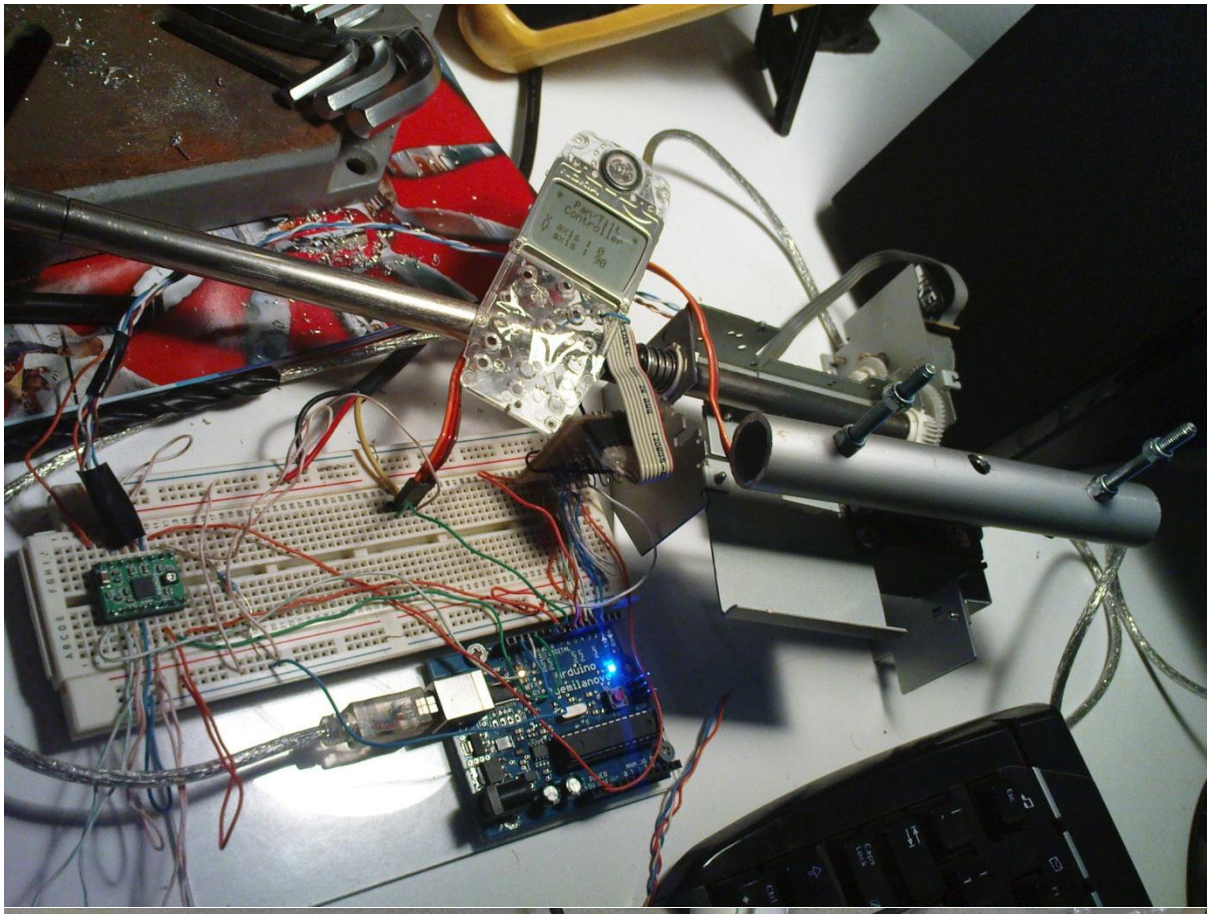
http://www.ubnt.com/airmax#nanostationm 15/02/2013

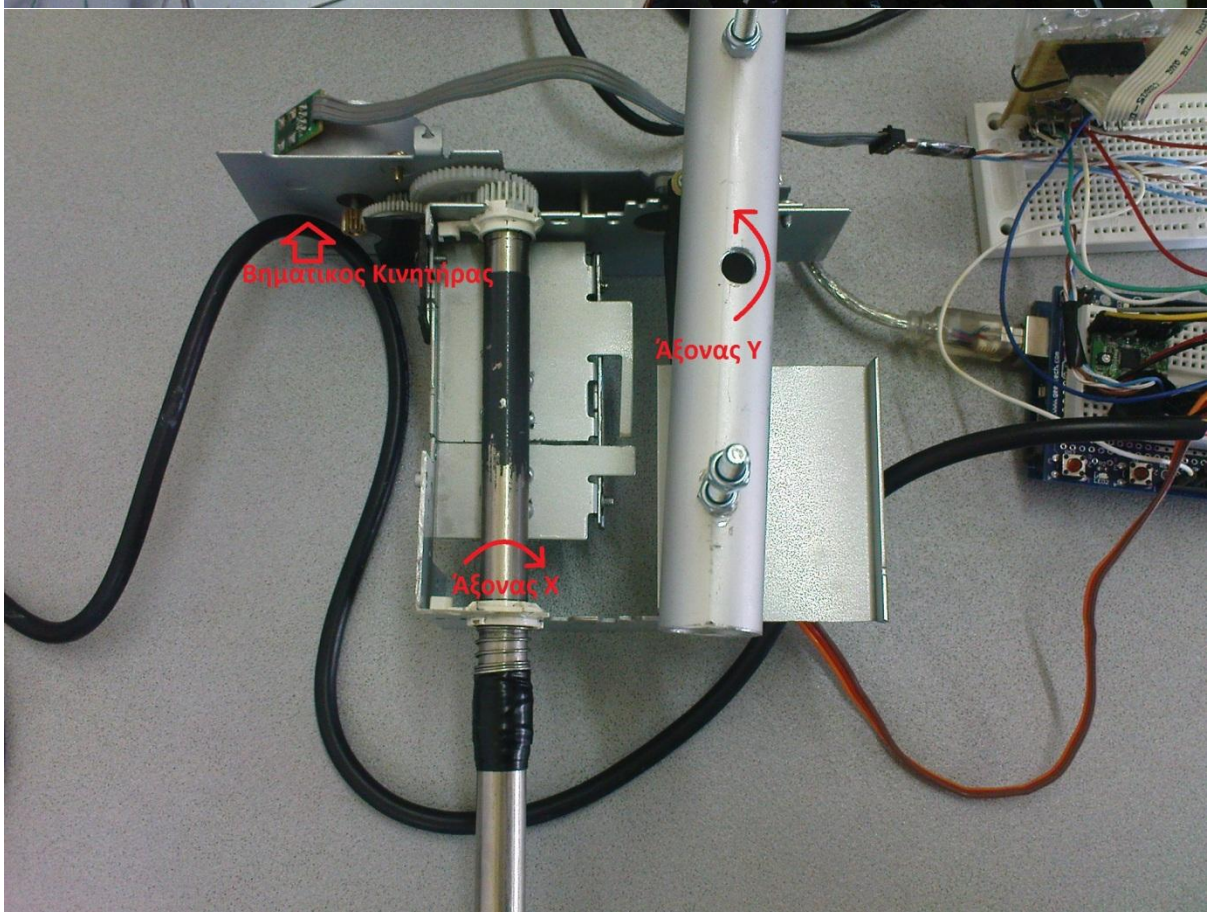
ΠΑΡΑΡΤΗΜΑ 1: ΦΩΤΟΓΡΑΦΙΕΣ

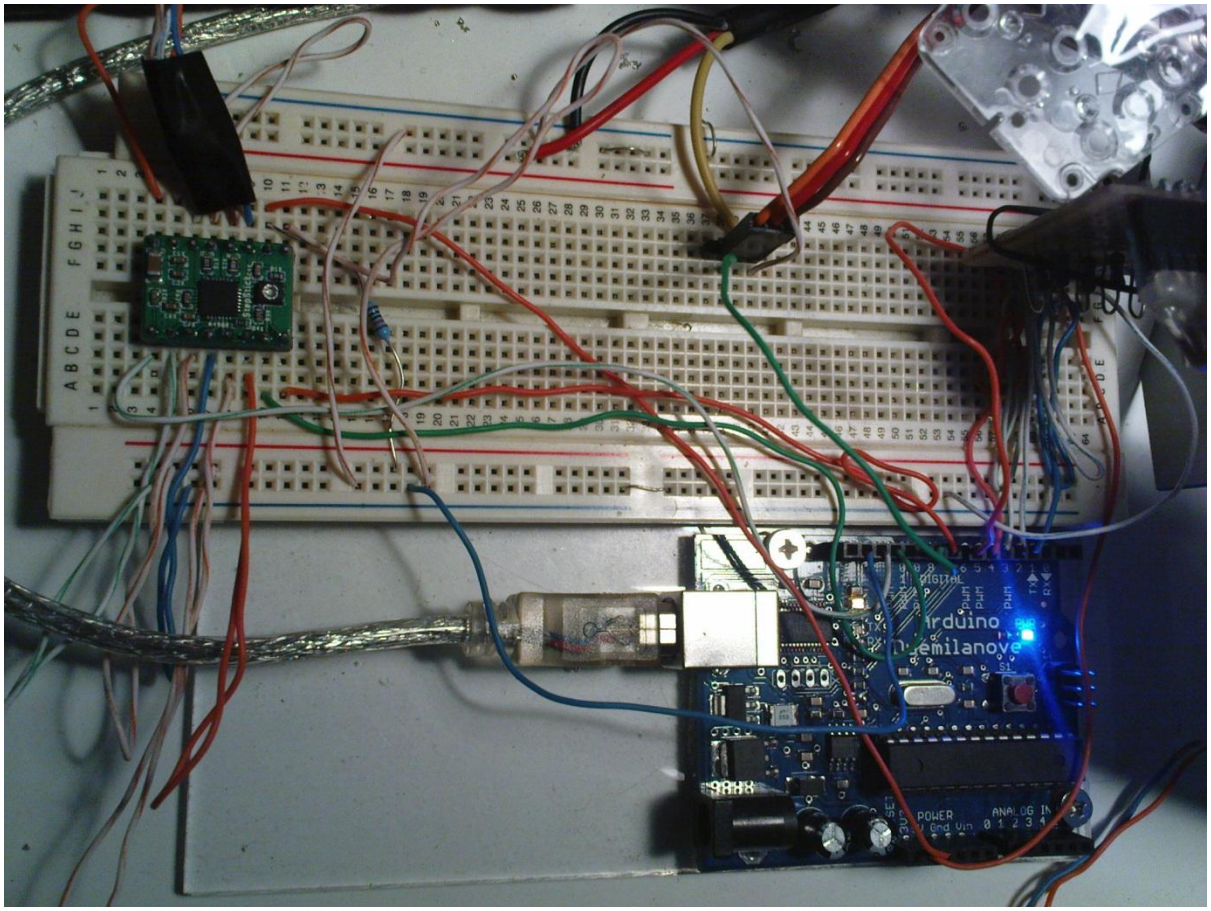

MITSUMI


NanoStation M2


192.168.1.6 45db
X:0 Y:90 20.7C




Βηματικος Κινητήρας
Άξονας Υ
Άξονας Χ


ABCDE FGHIJ
DIGITAL
POWER
ANALOG IN

ΠΑΡΑΡΤΗΜΑ 2: ΚΩΔΙΚΑΣ

```
/*
   Telnet Server and Nanostation client

 This sketch connects to a Nanostation telnet server
 using an Arduino Wiznet Ethernet shield.
 User is able to simultaneously connect to Arduino via Telnet
 in order to execute commands, change nanostation position or track
 WiFi networks.

 Circuit:
 * Ethernet shield attached to pins ISP(11, 12, 13) SS(10)
 * SD attached to pins ISP(11, 12, 13) SS(4)
 * Stepstick attached to pins (5, 6, 7)
 * Servo attached to pin 3
 * LCD attached to pins (A4, A5)
 * DS18S20 attached to pin (A3)
 * Omron EE-SX1042 Photomicrosensor X (A1)
 * Omron EE-SX1042 Photomicrosensor Y (A2)
 * Reset output (A0)

 created 9 Dec 2011
 last edit 18/11/2012
 by Fivos Giourgas

 */

#include <stdlib.h>
#include <SPI.h>
#include <Ethernet.h>
#include <Servo.h>
#include <MemoryFree.h>
#include <LiquidCrystal_SR.h>
#include <OneWire.h>

//pinout variables
const byte steppin = 7;  //stepstick step pin
const byte dirpin = 5;    //stepstick direction pin
const byte enable_X = 6;  //stepstick enable pin
const byte enable_Y = 3;  //stepstick 2 enable pin
const byte Photomicrosensor_X = A1; //Omron EE-SX1042 Photomicrosensor X pin
const byte Photomicrosensor_Y = A2; //Omron EE-SX1042 Photomicrosensor Y pin
const byte reset_out_pin = A0;
const byte DS18S20_Pin = A3; //DS18S20

LiquidCrystal_SR lcd(A4,A5,TWO_WIRE);
//                  | |
//                  | \-- Clock Pin
//                  \---- Data/Enable Pin

//Initiate Temperature chip i/o
OneWire ds(DS18S20_Pin);

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
//IPAddress ip(192, 168, 100, 69);
//IPAddress gateway(192, 168, 100, 254);
//IPAddress subnet(255, 255, 255, 0);

// Enter the IP address of the Nanostation you're connecting to:
byte nano[] = {
  192,168,1,20};

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 23 is default for telnet;
// if you're using Processing's ChatServer, use  port 10002):
EthernetClient nanostation;
EthernetServer server(28);
EthernetClient client = 0; // Client needs to have global scope so it can be called
// from functions outside of loop, but we don't know
```

```
// what client is yet, so creating an empty object

///////////////global variables//////////////

int curStepperPos_X = 0;         // current position of stepper motor (steps)
int possition_X = 0;             // position (degrees)
long curStepperPos_Y = 34740;         // current position of stepper motor (steps)
int possition_Y = 90;             // position (degrees)
byte resetcounter = 0;
float TemperatureSum;
boolean logged = false;      // whether you are logged to nanostation
boolean gotAMessage = false; // whether or not you got a message from the client yet
byte lcd_flag = 1;           // change lcd displayed data
float coordinates[6];

///////////////Timers//////////////////////

unsigned long lcd_timer = 0;//
unsigned long lastAttemptTime = 0;     // last attempt to connect to the server, in milliseconds
const int requestInterval = 10 * 1000; // delay between connect attempts.
unsigned long client_disconnect_timer = 0; //disconect client after some time inactive

//////////////Data Stuctures////////////////
struct Data{
  int angle;
  int power;

};
Data scan[13];
Data scanY[13];
int signal; //Signal level


////////////////////////////////SETUP////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
void setup() {

  //reset pin trigger (via ATTiny85)
  pinMode(reset_out_pin,OUTPUT);
  digitalWrite(reset_out_pin, LOW);

  pinMode(Photomicrosensor_X,INPUT); //Calibration Sensor
  pinMode(Photomicrosensor_Y,INPUT); //Calibration Sensor

  //explicitly deselect SD card reader
  pinMode(4, OUTPUT);
  digitalWrite(4, HIGH);

  //Setup StepStick
  pinMode(steppin, OUTPUT);
  pinMode(dirpin, OUTPUT);
  pinMode(enable_X, OUTPUT);
  pinMode(enable_Y, OUTPUT);
  digitalWrite(dirpin, HIGH);
  digitalWrite(steppin, LOW);
  digitalWrite(enable_X, HIGH);
  digitalWrite(enable_Y, HIGH);

  lcd.begin(16,2);     // initialize the lcd
  lcd.home ();         // go home
  lcd.print("Arduino WiFi");

  //Connect to Ethernet
  void connectToNanostation();
}
//////////////////////BEGIN LOOP//////////////////////////
//////////////////////////////////////////////////////////
void loop()
{
  Ethernet.maintain();
  if (nanostation.connected() && logged) {
    lcd_update(); // update the lcd display data every 4 second
    EthernetClient client = server.available();  // wait for a new client:
    // when the client sends the first byte, say hello:
    if (client) {
```

```
    client_disconnect_timer = millis();
    if (!gotAMessage) {
      gotAMessage = true;
      help();
    }
    while (client.connected()) {
      if (client.available()) {
        if(client.findUntil("-", "\n")){
          char type1 = client.read();
          char type2 = client.read();
          switch(type1){
          case 's':
            if( type2 == 't'){
              show_status();
            }
            else if (type2 == 'x'){
              optimal();
              command_prompt();
            }
            else if (type2 == 'y'){
              optimal_Y();
              command_prompt();
            }
            break;
          case 'm':
            if (type2 == 'x'){
              int val = client.parseInt();
              moveX(val);
              command_prompt();
            }
            else if (type2 =='y'){
              int val = client.parseInt();
              moveY(val);
              command_prompt();
            }
            break;
          case 'c':
            if (type2 == 'a'){
              Calibrate();
              command_prompt();
            }
            else if (type2 =='r'){
              server.print(F("freeMemory()="));
              server.println(freeMemory());
              command_prompt();
            }
            break;
          case 'h':
            help();
            break;
          case 'l':
            for(int field=0; field < 6; field++){  //Parse coordinates
              coordinates[field] = client.parseFloat();
            }
            point_location();
            break;
          default:
            server.println(F("Unknown command type -h for help")); //if wrong type print erroe msg
          }
        }
        break;
      }
    }
  }
  else {
    if (millis() - client_disconnect_timer > 180*1000){  // Disconect client after 3min of inactivity
      client.stop();
      gotAMessage = false;
    }
  }
}
else if (millis() - lastAttemptTime > requestInterval) {
  // if you're not connected, and time has passed since
  // your last connection, then attempt to connect again:
  resetcounter++;
```

```
    lcd.setCursor(15,1);
    lcd.print(resetcounter);
    if (resetcounter >= 3) {
      digitalWrite(reset_out_pin, HIGH);
    } //Pulling the RESET pin HIGH triggers the reset.
    nanostation.stop();
    connectToNanostation();
  }
}
//////////////////////////END LOOP//////////////////////////////////////
//////////////////////////////////////////////////////////////////////////

////////////////////////Point to target location//////////////////////////////
void point_location(){
  // point axis x
  int x = azimuth(coordinates[0], coordinates[1], coordinates[3], coordinates[4]) + 0.5; // we add 0.5 to round float to int

  // pont axis y
  float distance = distance_between(coordinates[0], coordinates[1], coordinates[3], coordinates[4], 1);
  int y = differencial_altitude_angle(coordinates[2], coordinates[5], distance) + 0.5;

  server.println();
  server.println(F("Station coordinates:"));
  server.print(coordinates[0], 7);
  server.print(F(" "));
  server.print(coordinates[1],7);
  server.print(F(" alt: "));
  server.println(coordinates[2],4);
  server.println();
  server.println(F("Target at: "));
  server.print(coordinates[3],7);
  server.print(F(" "));
  server.print(coordinates[4],7);
  server.print(F(" alt: "));
  server.println(coordinates[5],4);
  server.print(F("distance: "));
  server.print(distance);
  server.print(F("m azimuth: "));
  server.print(x);
  server.print(F(" elevation: "));
  server.println(y);
  command_prompt();

  moveX(x);
  moveY(y + 90); // servo center is at 90 degrees
}

//////////////////////////Refresh LCD data//////////////////////////////////////////
void lcd_update()
{
  if(millis() - lcd_timer > 4000) {
    lcd_timer = millis();
    switch (lcd_flag){
    case 1:
      getTemp();
      getSignal();
      lcd.clear();
      lcd.print(Ethernet.localIP());
      lcd.setCursor(0,1);
      lcd.print("X:");
      lcd.print(possition_X);
      lcd.print(" Y:");
      lcd.print(possition_Y);
      lcd.setCursor(11,1);
      lcd.print(TemperatureSum,1);
      lcd.print("C");
      lcd.setCursor(12,0);
      lcd.print(signal);
      lcd.print("db");
     // lcd_flag = 2; //uncomment to display second screen
      break;
    case 2:
      lcd.clear();
      lcd.print(millis() / 60000);
      lcd.print("min");
```

```
      lcd_flag = 1;
      break;
    }
  }
}
//////////////////////Telnet Show Devise Status//////////////////////////
void show_status(){
  getSignal();
  getTemp();
  server.print(F("Signal: "));
  server.print(signal);
  server.print(F("db |"));
  server.print(F(" axis X: "));
  server.print(possition_X);
  server.print(F(" axis Y: "));
  server.print(possition_Y);
  server.print(F(" (degree) | temp: "));
  server.print(TemperatureSum,1);
  server.println("C");
  command_prompt();
}

//////////////////////Calibrate Function//////////////////////////////////////////////////////
void Calibrate(){
  digitalWrite(enable_Y, LOW);                    //enabling stepper
  digitalWrite(dirpin, HIGH);
  while (digitalRead(Photomicrosensor_Y) == HIGH){
    digitalWrite(steppin, HIGH);
    delayMicroseconds(2);
    digitalWrite(steppin, LOW);
    delayMicroseconds(600);
  }
  curStepperPos_Y = 0;
  possition_Y = 0;
  digitalWrite(enable_Y, HIGH);                    //disabling stepper
  long st=0;
  digitalWrite(enable_Y, LOW);                    //enabling stepper
  digitalWrite(dirpin, LOW);
  while (digitalRead(Photomicrosensor_Y) == LOW)
  {
    st++;
    digitalWrite(steppin, HIGH);
    delayMicroseconds(2);
    digitalWrite(steppin, LOW);
    delayMicroseconds(600);
  }
  server.println(st);
  while (digitalRead(Photomicrosensor_Y) == HIGH){
    st++;
    digitalWrite(steppin, HIGH);
    delayMicroseconds(2);
    digitalWrite(steppin, LOW);
    delayMicroseconds(600);
  }
  curStepperPos_Y = 180;
  possition_Y = 180;
  server.println(st);
  digitalWrite(enable_Y, HIGH);                    //disabling stepper
}

//////////////////////Print telnet home help Screen//////////////////////////////////////////
void help()
{
  server.println(F("ARDUINO WiFi Telnet Server."));
  server.println();
  server.println(F("Available Commands:"));
  server.println(F("-st :Show devise status"));
  server.println(F("-sx :Scan for optimal position axis X"));
  server.println(F("-sy :Scan for optimal position axis Y"));
  server.println(F("-mx(degrees) :Set axis x angle(eg. -mx360)"));
  server.println(F("-my(degrees) :Set axis y angle(eg. -mx180)"));
  server.println(F("-ca :Calibrate stepper"));
  server.println(F("-cr :Check free memory Ram"));
  server.println(F("-l  :Align with target coordinates(-l lat1 long1 elev1 lat2 long2 elev2)"));
  server.println();
```

```
  command_prompt();
}
////////////////Print command prompt//////////////////////////////
void command_prompt(){
  server.println();
  server.print(F("Send command(-h for help):"));
}
////////////////Establish ethernet Connections//////////////////////////////
void connectToNanostation(){

  lcd.setCursor( 0 , 1 );
  lcd.print(F("connecting..."));
  logged = false;

  if(Ethernet.begin(mac) == 0) { // start ethernet using mac & DHCP
    lcd.clear();
    lcd.home();
    lcd.print(F("DHCP Failed"));
    while(true) // no point in carrying on, so stay in endless loop:
      ;
  }
  delay(1000); // give the Ethernet shield a second to initialize

  // print your local IP address:
  lcd.clear();
  lcd.home();
  lcd.print(Ethernet.localIP());

  // if you get a connection, login:
  if (nanostation.connect(nano, 23)>0) {
    // Login to nanostation
    nanostation.print("\n");
    if (nanostation.find("login:"));
    {       //if prompt to login
      nanostation.print("ubnt\n");
    }       //write username
    if (nanostation.find("Password:"));
    {    //if prompt for password
      nanostation.print("ubnt\n");
    }       //write password
    logged = (nanostation.find("BusyBox")); //if command prompt logged=true
    server.begin();                     //Begin Server
  }
  else {
    // if you didn't get a connection to the server:
    lcd.println(F("connection failed"));
  }

  // note the time of this connect attempt:
  lastAttemptTime = millis();

  SPI.setClockDivider(SPI_CLOCK_DIV16); //slow down SPI bus to 2MHz (Without this the system hangs after a while)
}

//////////////////////Get signal power from nanostation//////////////////////////////

int getSignal(){
  if (nanostation.connected()) {
    signal = 0;
//    int freeSpace = nanostation.free();
    int write_b = nanostation.println(F("iwconfig |grep Signal"));
    delay(1);
    if (nanostation.find("Signal level=-"));
    {
      signal = nanostation.parseInt();
      while (nanostation.available() >0){
        nanostation.read();
      }
    }
    nanostation.flush();
    return signal;
  }
  else{
    digitalWrite(reset_out_pin, HIGH); //Pulling the RESET pin HIGH triggers the reset.
    resetcounter++;
```

```
    lcd.setCursor(15,1);
    lcd.print(resetcounter);
    nanostation.stop();
    connectToNanostation();
   }
}

//////////////////Function to move axis X to the desired position/////////////////////////
void moveX(int possitionX)
{
  int i, j, steps;
  i = 1800;                //Set start speed
  possitionX = constrain(possitionX, 0, 360);
  possition_X = possitionX;   //save position to a global variable for other use
  digitalWrite(enable_X, LOW);                    //enabling stepper
  int posValue = map(possitionX, 0, 360, 0, 4320); //map position from degrees to steps

  // moving to possition

  if (posValue > curStepperPos_X){
    steps = posValue - curStepperPos_X;
    curStepperPos_X =curStepperPos_X + steps;
    digitalWrite(dirpin, LOW);

    for (j=0; j<steps; j++)
    {
      digitalWrite(steppin, HIGH);
      delayMicroseconds(2);
      digitalWrite(steppin, LOW);
      delayMicroseconds(i);
      if (steps - j < 200){
        i=i+6;
      }
      else {
        i=i-10;
      }
      i=constrain(i,1000,1800);
    }
  }
  else if (posValue < curStepperPos_X){
    steps = curStepperPos_X - posValue;
    curStepperPos_X =curStepperPos_X - steps;
    digitalWrite(dirpin, HIGH);

    for (j=0; j<steps; j++)
    {
      digitalWrite(steppin, HIGH);
      delayMicroseconds(2);
      digitalWrite(steppin, LOW);
      delayMicroseconds(i);
      if (steps - j < 200){
        i=i+6;
      }
      else {
        i=i-10;
      }
      i=constrain(i,1000,1800);
    }
  }
  delay(100);
  digitalWrite(enable_X, HIGH);      //disabling stepper
}

//////////////////Function to move axis Y to the desired position/////////////////////////
void moveY(int possitionY)
{
  int i;
  long j, steps;
  i = 680;                //Set start speed
  possitionY = constrain(possitionY, 0, 180);
  possition_Y = possitionY;   //save position to a global variable for other use
  digitalWrite(enable_Y, LOW);                    //enabling stepper
  long posValue = map(possitionY, 0, 180, 0, 69480); //map position from degrees to steps

  // moving to possition
```

```
  if (posValue > curStepperPos_Y){
    steps = posValue - curStepperPos_Y;
    curStepperPos_Y =curStepperPos_Y + steps;
    digitalWrite(dirpin, HIGH);

    for (j=0; j<steps; j++)
    {
      digitalWrite(steppin, HIGH);
      delayMicroseconds(2);
      digitalWrite(steppin, LOW);
      delayMicroseconds(i);

    }
  }
  else if (posValue < curStepperPos_Y){
    steps = curStepperPos_Y - posValue;
    curStepperPos_Y =curStepperPos_Y - steps;
    digitalWrite(dirpin, LOW);

    for (j=0; j<steps; j++)
    {
      digitalWrite(steppin, HIGH);
      delayMicroseconds(2);
      digitalWrite(steppin, LOW);
      delayMicroseconds(i);

    }
  }
  delay(100);
  digitalWrite(enable_Y, HIGH);      //disabling stepper
}

////////////////////function to move both axes/////////////////////////////////////////////
void move(int possitionX,int possitionY){
  moveX(possitionX);
  moveY(possitionY);
}

/////////////////compare function used for qsort()////////////////////
int sort(const void *x, const void *y)
{
  const struct Data
    *data1 = (struct Data *)x,
  *data2 = (struct Data *)y;

  if ( data1->power > data2->power ) return 1;
  if ( data1->power < data2->power ) return -1;

  return 0;
}

///////////////Scan for the optimal position X////////////////////////////
void optimal(){
  moveY(90);
  server.println(F("X axis First Scan Array"));
  scan_function (0, 300, 60, 6);
  int pos1,pos2;
  if (scan[0].angle==0 && scan[1].angle == 300){
    scan[0].angle=360;
  }
  if (scan[1].angle==0 && scan[0].angle == 300){
    scan[1].angle=360;
  }
  int scan_step = (abs(scan[1].angle - scan[0].angle))/6;
  pos1 = min(scan[0].angle, scan[1].angle);
  pos2 = max(scan[0].angle, scan[1].angle);
  server.println(F("Second Scan"));
  server.print(F("Scan Step: "));
  server.println(scan_step);
  server.print(F("Start position: "));
  server.println(pos1);
  server.print(F("End position: "));
  server.println(pos2);
  server.println(F("Array"));
```

```
  scan_function (pos1, pos2, scan_step, 7);
  moveX(scan[0].angle);
}

//////////////Scan for optimal position Y////////////////////
void optimal_Y(){
  server.println(F("Y axis First Scan Array"));
  scan_function_Y(30, 150, 20, 7);
  int pos1,pos2;
  int scan_step = (abs(scanY[1].angle - scanY[0].angle))/5;
  pos1 = min(scanY[0].angle, scanY[1].angle);
  pos2 = max(scanY[0].angle, scanY[1].angle);
  server.println(F("Second Scan"));
  server.print(F("Scan Step: "));
  server.println(scan_step);
  server.print(F("Start position: "));
  server.println(pos1);
  server.print(F("End position: "));
  server.println(pos2);
  server.println(F("Array"));
  scan_function_Y (pos1, pos2, scan_step, 6);
  moveY(scanY[0].angle);
}

//////////////scan function for axis X////////////////////
void scan_function(int pos1, int pos2, int scan_step, byte matrix_size){
  byte n=0;
  for (pos1; pos1 <= pos2;  pos1 += scan_step )
  {
    moveX(pos1);
    delay(5000);
    scan[n].angle = pos1;
    scan[n].power = getSignal();
    server.print (scan[n].angle);
    server.print (" ");
    server.print (scan[n].power);
    server.println();
    n++;
  }
  qsort (scan, matrix_size, sizeof(struct Data), sort);
  server.println(F("qsorted array"));
  for (n=0; n < matrix_size; n++){
    server.print (scan[n].angle);
    server.print (" ");
    server.println (scan[n].power);
  }
}

//////////////scan function for axis Y////////////////////
void scan_function_Y(int pos1, int pos2, int scan_step, byte matrix_size){
  byte n=0;
  for (pos1; pos1 <= pos2;  pos1 += scan_step )
  {
    moveY(pos1);
    delay(1000);
    scanY[n].angle = pos1;
    scanY[n].power = getSignal();
    server.print (scanY[n].angle);
    server.print (" ");
    server.print (scanY[n].power);
    server.println();
    n++;
  }
  qsort (scanY, matrix_size, sizeof(struct Data), sort);
  server.println(F("qsorted array"));
  for (n=0; n < matrix_size; n++){
    server.print (scanY[n].angle);
    server.print (" ");
    server.println (scanY[n].power);
  }
}

////////////////////DS18B20 GET TEMP//////////////////////////////
float getTemp(){
  //returns the temperature from one DS18S20 in DEG Celsius
```

```
  byte data[12];
  byte addr[8];

  if ( !ds.search(addr)) {
    //no more sensors on chain, reset search
    ds.reset_search();
    return -1000;
  }

  if ( OneWire::crc8( addr, 7) != addr[7]) {
    return -1000;
  }

  if ( addr[0] != 0x10 && addr[0] != 0x28) {
    return -1000;
  }

  ds.reset();
  ds.select(addr);
  ds.write(0x44,1); // start conversion, with parasite power on at the end

  byte present = ds.reset();
  ds.select(addr);
  ds.write(0xBE); // Read Scratchpad


  for (int i = 0; i < 9; i++) { // we need 9 bytes
    data[i] = ds.read();
  }

  ds.reset_search();

  byte MSB = data[1];
  byte LSB = data[0];

  float tempRead = ((MSB << 8) | LSB); //using two's compliment
  TemperatureSum = tempRead / 16;

  return TemperatureSum;

}

float distance_between (float lat1, float long1, float lat2, float long2, float units_per_meter) {
  // returns distance in meters between two positions, both specified
  // as signed decimal-degrees latitude and longitude. Uses great-circle
  // distance computation for hypothised sphere of radius 6372795 meters.
  // Because Earth is no exact sphere, rounding errors may be upto 0.5%.
  float delta = radians(long1-long2);
  float sdlong = sin(delta);
  float cdlong = cos(delta);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  float slat1 = sin(lat1);
  float clat1 = cos(lat1);
  float slat2 = sin(lat2);
  float clat2 = cos(lat2);
  delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
  delta = sq(delta);
  delta += sq(clat2 * sdlong);
  delta = sqrt(delta);
  float denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
  delta = atan2(delta, denom);
  return delta * 6372795 * units_per_meter;
}

float azimuth (float lat1, float long1, float lat2, float long2) {
  // returns initial course in degrees (North=0, West=270) from
  // position 1 to position 2, both specified as signed decimal-degrees
  // latitude and longitude.
  float dlon = radians(long2-long1);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  float a1 = sin(dlon) * cos(lat2);
  float a2 = sin(lat1) * cos(lat2) * cos(dlon);
```

```
  a2 = cos(lat1) * sin(lat2) - a2;
  a2 = atan2(a1, a2);
  if (a2 < 0.0) {
    a2 += TWO_PI;                                   // modulo operator doesn't seem to work on floats
  }
  return degrees(a2);
}

float differencial_altitude_angle (float elev1, float elev2, float distance) {
  // returns y axis angle based on altitude of two locations and the
  // distance between them.
  float dif = elev2 - elev1; //calculate the altitude difference
  float w = atan2(dif,distance);
  return degrees (w);
}
```