Πανεπιστήμιο Δυτικής Αττικής

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

ΚΟΚΚΙΝΟΣ ΣΩΚΡΑΤΗΣ

Α.Μ. 46893

Εισηγητής: ΙΩΑΝΝΗΣ ΒΟΓΙΑΤΖΗΣ, ΚΑΘΗΓΗΤΗΣ

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

(Κενό φύλλο)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

ΚΟΚΚΙΝΟΣ ΣΩΚΡΑΤΗΣ

Α.Μ. 46893

Εισηγητής:

ΙΩΑΝΝΗΣ ΒΟΓΙΑΤΖΗΣ, ΚΑΘΗΓΗΤΗΣ

Ioannis Voyiatzis

Digitally signed by Ioannis Voyiatzis
Date: 2022.10.31 11:39:05 +02'00'

Εξεταστική Επιτροπή:

«ΜΑΤΙΑΤΟΣ, ΣΠΥΡΙΔΩΝ»

Spyridon Matiatos

Digitally signed by Spyridon Matiatos
Date: 2022.10.31 13:27:35 +02'00'

«ΕΥΣΤΑΘΙΟΥ, ΚΩΝΣΤΑΝΤΙΝΟΣ»

Konstantinos Efstathiou

Digitally signed by Konstantinos Efstathiou
Date: 2022.10.31 14:36:47 +02'00'

Ημερομηνία εξέτασης: 07/10/2022

(Κενό φύλλο)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

«Βεβαιώνω ότι είμαι συγγραφέας της παρούσας διπλωματικής εργασίας και ότι έχω αναφέρει ή παραπέμψει σε αυτή, ρητά και συγκεκριμένα, όλες τις πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, προτάσεων ή λέξεων, είτε αυτές μεταφέρονται επακριβώς (στο πρωτότυπο ή μεταφρασμένες) είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για την συγκεκριμένη διπλωματική εργασία»

Ο Δηλών

(Κενό φύλλο)

Ευχαριστίες

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε στο Πανεπιστήμιο Δυτικής Αττικής, στο τμήμα Μηχανικών Πληροφορικής και Υπολογιστών. Με την ολοκλήρωση αυτής της εργασίας λοιπόν, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες σε αυτούς που συνέβαλαν στην εκπόνησή της.

Αρχικά θέλω να ευχαριστήσω το επιβλέπων καθηγητή μου, κύριο Ιωάννη Βογιατζή που μου έδωσε την ευκαιρία να αναλάβω αυτό το θέμα, το οποίο ήταν σίγουρο πως θα χρειαζόταν αρκετός χρόνος για να υλοποιηθεί. Επίσης, ευχαριστώ και τον υποψήφιο διδάκτορα Δημήτρη Ψιλιά που ανέλαβε να βοηθήσει στην επίβλεψη της εργασίας, για την υπομονή και το χρόνο που διέθεσε.

Επιπλέον, θα ήθελα να δώσω τις ευχαριστίες μου στην ομάδα Talos του Πα.Δ.Α. και στα μέλη που την απαρτίζουν καθώς συνέβαλαν στην ορθή πραγματοποίηση μερών τις εργασίας, με τις συζητήσεις που κάναμε σχετικά με το θέμα. Επίσης, πρέπει να ευχαριστήσω ξεχωριστά το φίλο και μεταπτυχιακό πλέον φοιτητή, Χρήστο Μουργελά που ήταν παρών από την αρχή της λήψης αυτής της ιδέας και κατά τη διάρκεια τον προπτυχιακών σπουδών μας, ως συμφοιτητές, δουλέψαμε μαζί σε παρόμοιες εργασίες με το συγκεκριμένο αυτό θέμα.

Τέλος, με πολύ αγάπη, θέλω να πω ένα μεγάλο ευχαριστώ στους γονείς μου, Κωνσταντίνο και Αντωνία, στην αδερφή μου Αθηνά, καθώς και στη γυναίκα μου Νίκη, οι οποίοι με στήριξαν ενεργά και υπομονετικά κατά τη διάρκεια όλων των σπουδών μου.

(Κενό φύλλο)

Περίληψη

Στις μέρες μας η χρήση των μη επανδρωμένων πτητικών μέσων (UAVs) ή αλλιώς Drones, έχει αυξηθεί εκθετικά σε πολλές κοινωνικές, επαγγελματικές ή ερευνητικές δραστηριότητες. Έτσι αποφασίστηκε η ενασχόληση με ένα βασικό πρόβλημα της χρήσης αυτών των συστημάτων, που δεν είναι άλλο από το περιορισμένο χρόνο πτήσης. Σε αυτή την εργασία θα γίνει μια προσπάθεια αντιμετώπισης αυτού του προβλήματος με χρήση ενός αυτόνομου συστήματος φόρτισης των μπαταριών των drone. Θα κατασκευαστεί ένας αυτόματος σταθμός φόρτισης ο οποίος θα είναι οικονομικός, φορητός, και εύκολος στη χρήση από τον οποιονδήποτε. Τα drone θα μπορούν εύκολα και αυτόματα να προσγειωθούν και να φορτίσουν τις μπαταρίες τους και στη συνέχεια να συνεχίσουν την αποστολή τους. Μελλοντικός στόχος αυτού του εγχειρήματος είναι να υπάρχουν πληθώρα τέτοιων σταθμών ώστε οποιοδήποτε drone, όπου και αν βρίσκεται, να μπορεί εύκολα να κάνει στάση για φόρτιση.

Abstract

In our days, the usage of Unmanned Aerial Vehicles (UAVs) known as Drones has exponentially increased into many social, professional or research activities. Thus, it was decided to engage with a major problem of usage those systems, where is not else but the limited flight duration. In this work, an attempt will be made to deal with this problem by using an autonomous drone battery charging system. An automatic charging station will be built which will be economical, portable, and easy to use by anyone. The drones will be able to easily and autonomously land and charge their batteries and then continue their mission. The future goal of this project is to have a plethora of such stations so that any drone, wherever it is located, can easily make a stop for charging.

Περιεχόμενα

Πίνακας εικόνων

Κεφάλαιο 1. Εισαγωγή

Στις μέρες μας, η χρήση των μη επανδρωμένων πτητικών μέσων (unmanned aerial vehicles - UAVs) έχει γίνει αναπόσπαστο κομμάτι της ζωής μας. Η χρήση τους έχει διευρυνθεί σε όλους τους τομείς της κοινωνίας, από την απλή ψυχαγωγία μέχρι στρατιωτικές επιχειρήσεις. Τα UAVs ή όπως έχει επικρατήσει πλέον, τα drones, έχουν ήδη αποδείξει την αξία τους σε βιομηχανικές και εμπορικές εφαρμογές, όπως για παράδειγμα στην επιτήρηση και διαχείριση αποθηκών, στην υποστήριξη γεωργικών καλλιεργειών μεγάλων εκτάσεων καθώς και σε μεταφορές προϊόντων σε δυσπρόσιτες περιοχές και μη. Ακόμα, παίζουν σημαντικό ρόλο σε αποστολές διάσωσης, επιτήρησης και παρακολούθησης καθώς και σε στρατιωτικές επιχειρήσεις. Η λίστα αυτή δεν έχει τελειωμό, αφού υπάρχει η δυνατότητα εγκατάστασης τεχνολογικού εξοπλισμού σε ένα drone και η απογείωσή του με σκοπό κάποια αποστολή.

Υπάρχει όμως ένα βασικό μειονέκτημα με τα drones, το οποίο είναι ένα αγκάθι στην κατά τ’ άλλα συνεχώς αυξανόμενη χρήση τους. Αυτό, δεν είναι άλλο από τον χρόνο πτήσης. Καθώς τα drone χρησιμοποιούν κατά βάση μπαταρίες για την λειτουργία τους, πρέπει να επιστρέφουν πίσω στη βάση τους για φόρτιση ή αλλαγή μπαταριών. Επίσης ο χρόνος πτήσης είναι περίπου 25 λεπτά. Αυτός ο χρόνος δεν είναι συχνά αρκετός για να ολοκληρωθεί πλήρως μια αποστολή. Και φυσικά, πέρα από τα παραπάνω, υπάρχει η ανάγκη του ανθρώπινου παράγοντα για την αλλαγή ή φόρτιση των μπαταριών.

Έτσι, η επιστημονική κοινότητα αναζητά συνεχώς λύσεις για αυτό το πρόβλημα. Μία καλή περίπτωση θα ήταν η ανάπτυξη καλύτερων μπαταριών που θα κρατάνε περισσότερο και θα είναι ελαφρύτερες για τη μείωση του φορτίου και την αύξηση του χρόνου πτήσης. Εναλλακτική λύση είναι να βρεθούν τρόποι αυτόνομης φόρτισης των μπαταριών χωρίς την ανθρώπινη παρέμβαση.

Σε αυτή την εργασία, θα επιχειρήσω τη μελέτη μιας πλήρους αυτόνομης φόρτισης των drones με χρήση αυτόματου σταθμού φόρτισης. Έχουν γίνει ήδη κάποιες δημοσιεύσεις σχετικά με συστήματα αυτόματης φόρτισης τα τελευταία χρόνια [1], το οποίο είναι πολύ θετικό. Στις εργασίες αυτές, κυρίως επικεντρώνονται στον αποδοτικότερο τρόπο φόρτισης, είτε με ασύρματο τρόπο με χρήση επαγωγικών μεθόδων [2] [3] [4] [5] [6], είτε εξ επαφής με χρήση ειδικών αγώγιμων επαφών [7] [8]. Επίσης, υπάρχουν προτάσεις πρωτότυπων μεθόδων ανίχνευσης του drone όταν προσγειωθεί στο σταθμό με χρήση αισθητήρων [9].

Σε αυτή την εργασία θα γίνει μελέτη, σχεδίαση και κατασκευή του σταθμού φόρτισης, για φόρτιση εξ επαφής και επιπλέον, θα μελετηθούν τρόποι ανίχνευσης και προσγείωσης του drone στο σταθμό. Ο σταθμός θα απευθύνεται σε πολυκόπτερα drones διαφόρων μεγεθών μέχρι τη διάμετρο του ενός μέτρου. Επίσης θα είναι φορητός και θα εγκαθίσταται εύκολα. Καθώς η φόρτιση θα γίνεται εξ επαφής, ο σταθμός θα έχει 100% απόδοση στη μεταφορά ενέργειας σε σύγκριση με τη ασύρματη φόρτιση που η απόδοση μειώνεται λόγω διαφόρων απωλειών. Η ανίχνευση του drone κατά τη προσγείωση θα γίνεται από ένα σύστημα αισθητήρων πίεσης που κατασκευάστηκε για αυτό το σκοπό. Τέλος, ο σταθμός θα έχει τη δυνατότητα σύνδεσης στο internet μέσω wifi ή ενσύρματα και θα υπάρχει σύστημα επικοινωνίας LoRa για την διασύνδεση με τα drones και με άλλους σταθμούς που πιθανώς να υπάρχουν εγκατεστημένα στη περιοχή.

Στα κεφάλαια που θα ακολουθήσουν θα γίνει προσπάθεια ανάλυσης των επιμέρους συστημάτων του σταθμού, ανάλυση των κυκλωμάτων που χρησιμοποιήθηκαν και θα παρουσιαστούν διάφορες δοκιμές και μετρήσεις για να ελεγχθεί η λειτουργικότητα.

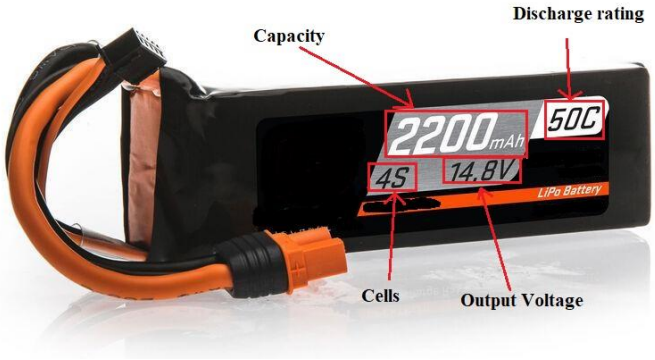
Κεφάλαιο 2. Θεωρητική περιγραφή

Όπως αναφέρθηκε και στην εισαγωγή το μεγαλύτερο πρόβλημα που συναντάμε στη χρήση των drone είναι ο μικρός χρόνος αυτονομίας στη πτήση και η ανάγκη επιστροφής στο χώρο απογείωσης για την επαναφόρτιση ή αλλαγή μπαταριών.

2.1 Μπαταρίες LiPo

Οι μπαταρίες που χρησιμοποιούνται συνήθως για χρήση σε τηλεκατευθυνόμενα μη επανδρωμένα οχήματα είναι τεχνολογίας ιόντων-λιθίου πολυμερών (Lithium-ion Polymer ή LiPo). Οι μπαταρίες LiPo είναι επαναφορτιζόμενες και χρησιμοποιούν πολυμερή ηλεκτρολύτη αντί για κάποιον υγρό. Αυτού του είδους η τεχνολογία παρέχει αρκετά προνόμια, τα σημαντικότερα εκ των οποίων είναι, το μικρότερο βάρος σε σύγκριση με μπαταρίες άλλων τεχνολογιών (μολύβδου, καδμίου, κλπ) , η υψηλότερη τάση των στοιχείων (cells) που της αποτελούν και η μεγάλη ενεργειακή πυκνότητά τους. Επίσης το σχήμα τους μπορεί να προσαρμοστεί ανάλογα με τις ανάγκες της κατασκευής λόγω του πολυμερούς ηλεκτρολύτη.

Για την αναζήτηση της κατάλληλης μπαταρίας LiPo πρέπει να γνωρίζουμε τις παραμέτρους που την χαρακτηρίζουν. Συνήθως μπορούμε να διακρίνουμε ότι είναι απαραίτητο να γνωρίζουμε, επάνω στο περιτύλιγμα της μπαταρίας (Εικόνα 1).



Εικόνα 1: Μπαταρία LiPo

2.1.1 Τάση (voltage) και στοιχεία (cells)

Μια μπαταρία LiPo αποτελείται από ορθογώνια στοιχεία (cells) συνδεδεμένα μεταξύ τους. Κάθε στοιχείο αποτελεί από μόνο του μια ξεχωριστή μπαταρία η οποία έχει την τάση των 3.7 Volts. Συνδέοντας τέτοια στοιχεία μεταξύ τους σε σειρά προκύπτει μια μπαταρία που η τάση στα άκρα της είναι το άθροισμα της τάσης των στοιχείων που την αποτελούν.

3.7V battery = 1 cell x 3.7V= 1S battery
7.4V battery = 2 cell x 3.7V= 2S battery
11.1V battery = 3 cell x 3.7V= 3S battery
14.8V battery = 4 cell x 3.7V= 4S battery
18.5V battery = 5 cell x 3.7V= 5S battery
22.2V battery = 6 cell x 3.7V= 6S battery
29.6V battery = 8 cell x 3.7V= 8S battery
37.0V battery = 10 cell x 3.7V= 10S battery
44.4V battery = 12 cell x 3.7V= 12S battery

Πρέπει να αναφερθεί πως η ελάχιστη τάση που ένα στοιχείο μπορεί να έχει πριν χρειαστεί φόρτιση είναι τα 3.0 Volts, ενώ η τιμή που το στοιχείο θεωρείται πλήρως φορτισμένο είναι τα 4.2 Volts. Έτσι για παράδειγμα, μια μπαταρία LiPo με 4 cells χρειάζεται φόρτιση όταν πέσει η τάση στα 4 x 3.0V = 12 Volts και είναι πλήρως φορτισμένη όταν φτάσει η τάση 4 x 4.2V = 16.8 Volts.

2.1.2 Χωρητικότητα (Capacity)

Η χωρητικότητα της μπαταρίας αντιπροσωπεύει το χρόνο που μπορεί η μπαταρία να παρέχει ενέργεια μέχρι να χρειαστεί να την ξαναφορτίσουμε. Η χωρητικότητα μετριέται σε Ah (amper hours). Σε αυτές τις μπαταρίες συνηθίζεται να μετριέται σε mAh.

$$1000mAh = 1Ah$$

Όσο μεγαλύτερη τιμή χωρητικότητας έχει η μπαταρία, τόσο περισσότερη ώρα μπορεί να κρατήσει. Όμως αυξάνεται και το βάρος της, οπότε ο χρόνος πτήσης ενός drone δεν αυξάνεται γραμμικά με την αύξηση της χωρητικότητας της μπαταρίας καθώς το φορτίο σε kg που πρέπει να σηκώσει, μεγαλώνει και αυτό.

2.1.3 Βαθμός εκφόρτισης (Discharge rating)

Ο βαθμός εκφόρτισης (C Rating), δείχνει το πόσο γρήγορα μπορεί να παρέχει ενέργεια η μπαταρία με ασφάλεια. Ποιο απλά, πόσο ρεύμα μπορεί να δίνει συνεχόμενα κατά τη λειτουργία της χωρίς να υποστεί κάποια ζημιά. Πολλαπλασιάζοντας την τιμή του C με τη τιμή της χωρητικότητας της μπαταρίας, μαθαίνουμε το μέγιστο ρεύμα που μπορεί η μπαταρία να παρέχει συνεχόμενα με ασφάλεια.

Για παράδειγμα, η μπαταρία στην Εικόνα 1, που έχει 2200mAh χωρητικότητα και 50C βαθμό εκφόρτισης, μπορεί να δώσει συνεχόμενο ρευμα:

$$2200mAh * 50 \rightarrow 2.2Ah * 50 \rightarrow \mathbf{110Ah}$$

Θεωρητικά λοιπόν, αν αυτή τη μπαταρία την εκφορτίσουμε με ρεύμα 110 A, θα κρατήσει για μία ώρα μέχρι να χρειαστεί ξανά φόρτιση.

Από τα παραπάνω φαίνεται πως η επιλογή της μπαταρίας για ένα drone δεν είναι απλή υπόθεση και χρειάζεται να λάβουμε υπόψη όλες αυτές τις παραμέτρους κατά την αγορά.

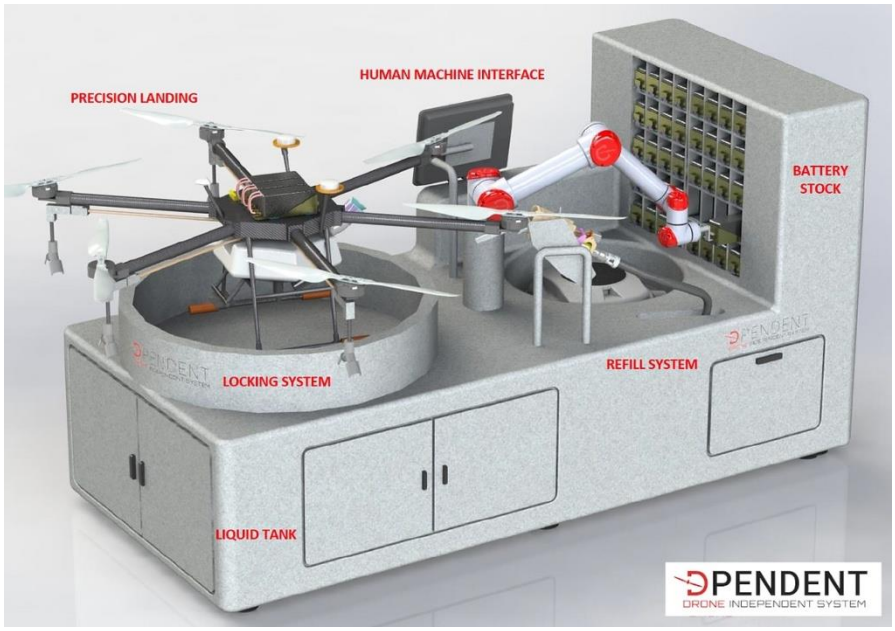
2.2 Μέθοδοι φόρτισης

Γενικά η παροχή ενέργειας για τα μη επανδρωμένα πτητικά μέσα δημιουργεί πρόβλημα σε πολλούς που σκέφτονται να επενδύσουν σε κάποιο drone, είτε για ψυχαγωγία, είτε για επαγγελματική χρήση.

Υπάρχουν διάφορες λύσεις, που κατά περίπτωση είναι χρήσιμες για τους χρήστες, όπως η συνεχής τροφοδότηση ενός drone με καλώδιο ρεύματος το οποίο είναι συνδεδεμένο σε μόνιμη βάση (tethered) με το drone και με το έδαφος σε σημείο παροχής ενέργειας. Αυτή η μέθοδος όμως ακυρώνει σε μεγάλο βαθμό το αρχικό ρόλο της αποστολής ενός drone, δηλαδή το να ταξιδεύει μακριά και γρήγορα προς έναν προορισμό. Είναι όμως μια αποδοτική λύση για κάποιον που θέλει να κάνει μετρήσεις ή να παρακολουθεί μια περιοχή από κάποιο ύψος, κ.α. Έχουν δημοσιευτεί διάφορες ενδιαφέρουσες εργασίες σχετικά με εφαρμογές που χρησιμοποιούν tethered drones, όπως σχετικές με αναμετάδοση σημάτων για δίκτυα τηλεπικοινωνιών [10] [11], καθώς και εργασίες που ερευνούν και αναπτύσσουν καλύτερα συστήματα σχετικά με τα tethered drones [12] [13].

Εικόνα 2: Σύστημα συνδεδεμένου (tethered) drone [14]

Την τελευταία δεκαετία τουλάχιστον, δημοσιεύονται συνεχώς μελέτες σχετικά με σταθμούς φόρτισης, όπου το drone προσγειώνεται και φορτίζει αυτόματα χωρίς την ανθρώπινη παρέμβαση. Μερικές εργασίες προτείνουν αυτόματα συστήματα αλλαγής μπαταριών [15] [16]. Δηλαδή ειδικούς σταθμούς στο έδαφος, που το drone προσγειώνετε και ένας αυτόματος μηχανισμός αλλάζει την τελειωμένη μπαταρία με μία πλήρως φορτισμένη. Αυτό δίνει την δυνατότητα στο drone να συνεχίσει την αποστολή του ενώ η άδεια μπαταρία θα επαναφορτισθεί στο σταθμό. Αυτού του είδους η λύση φαίνεται να απαιτεί συγκεκριμένα μοντέλα drone διότι ο εκάστοτε μηχανισμός στον σταθμό πρέπει να είναι προσαρμοσμένος για κάποιο συγκεκριμένο μοντέλο.



Εικόνα 3: Σταθμός αυτόματης αλλαγής μπαταρίας [17]

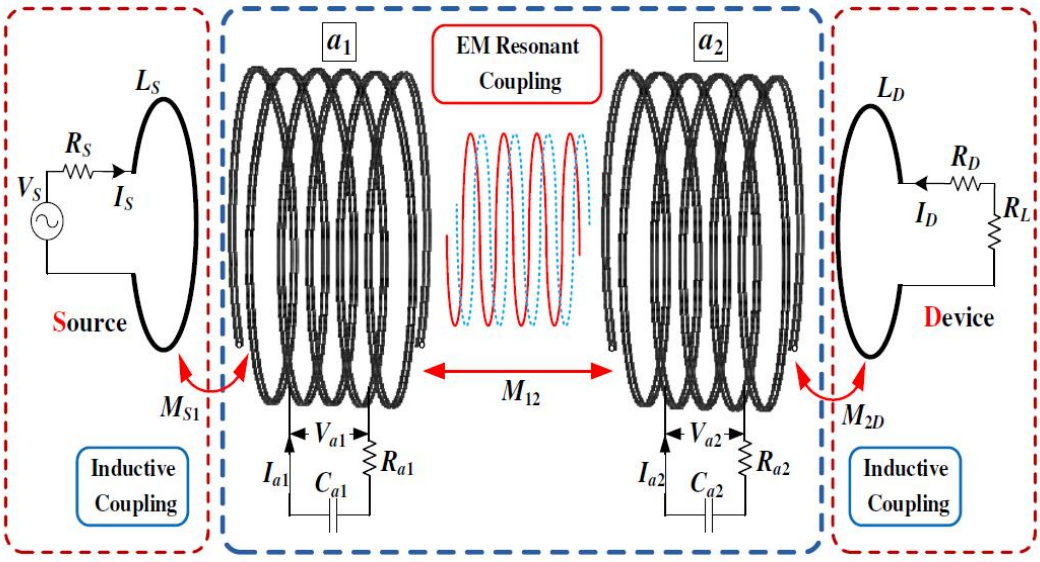
Εκτός της μεθόδου με την αλλαγή μπαταριών, έχουν δημοσιευτεί αρκετές εργασίες που παρουσιάζουν την φόρτιση των μπαταριών του drone, ενώ η μπαταρία βρίσκεται επάνω του. Μπορούμε να ξεχωρίσουμε δύο βασικούς τρόπους φόρτισης. Την φόρτιση με αγώγιμη επαφή και την ασύρματη φόρτιση με χρήση επαγωγικών μεθόδων. Η χρήση τέτοιων σταθμών επιλύει το πρόβλημα που απαιτεί τη προσγείωση συγκεκριμένου μοντέλου drone, καθώς η αρχή λειτουργίας τους στηρίζεται κατά βάση σε μια πλατφόρμα συγκεκριμένων διαστάσεων όπου προσγειώνεται οποιοδήποτε drone χωράει και φέρει τον εξοπλισμό που απαιτεί ο σταθμός για να μπορεί να ξεκινήσει η φόρτιση.

2.2.1 Σταθμοί ασύρματης φόρτισης και φόρτισης με επαφή

Η τεχνολογία της ασύρματης φόρτισης μπαταριών είναι ήδη γνωστή και χρησιμοποιείται τα τελευταία χρόνια σε συσκευές όπως, κινητά τηλέφωνα, έξυπνα ρολόγια, οδοντόβουρτσες και πολλά άλλα.

Γενικά στην ασύρματη φόρτιση, η μεταφορά ενέργειας γίνεται με επαγωγή, χρησιμοποιώντας συνήθως ειδικά σχεδιασμένα πηνία. Υπάρχουν διάφορα κυκλώματα και μέθοδοι επαγωγικής φόρτισης, αλλά η αποδοτικότερη φαίνεται να είναι η σύζευξη μαγνητικού συντονισμού (Magnetic resonant coupling).

Στην ασύρματη φόρτιση, η απόδοση του συστήματος είναι πολύ σημαντική γιατί υπάρχει κατά κανόνα απώλεια ενέργειας που εξαρτάται από τη επαγωγική σύζευξη των πηνίων, των δύο πλευρών (πομπός – δέκτης). Σε μια κλασική μέθοδο μαγνητικής σύζευξης δύο πηνίων, η απόδοση της μεταφοράς ενέργειας εξαρτάται σε μεγάλο βαθμό από την απόσταση που έχουν μεταξύ τους και από το πόσο έκκεντρα βρίσκονται. Όσο μικρότερη η απόσταση και όσο λιγότερο έκκεντρα είναι, τόσο καλύτερη απόδοση έχουμε στην μεταφορά. Στη σύζευξη μαγνητικού συντονισμού συμπεριλαμβάνεται στο σύστημα και η επίδραση της συχνότητας ρεύματος που διαρρέει τα πηνία. Με την προσθήκη πυκνωτών στο κύκλωμα καταφέρνει να ρυθμίσει τη συχνότητα με τέτοιο τρόπο ώστε τα δύο μέρη (πομπός – δέκτης) να συντονίζουν με αποτέλεσμα η μεταφορά ενεργείας να είναι πολύ ποιο αποδοτική και σε μεγαλύτερες αποστάσεις σε συνάρτηση και με τη συχνότητα. Έχουν δημοσιευτεί αρκετές εργασίες με σχετικό περιεχόμενο [18] [19].



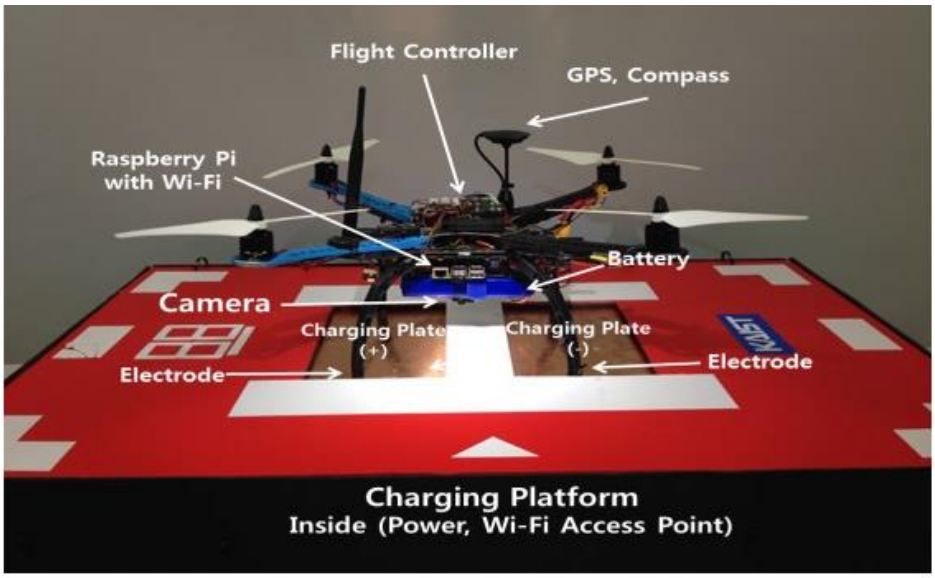
Εικόνα 4: Αρχή λειτουργίας ασύρματης φόρτισης [20]

Με τους σταθμούς ασύρματης φόρτισης καταφέρνουμε να φορτίσουμε τα drone χωρίς να υπάρχει φυσική επαφή των δύο μερών. Αυτό κάνει το σύστημα ασφαλέστερο σε φθορές και φυσικά φαινόμενα καθώς

δεν χρειάζεται να υπάρχουν αγώγιμα μέρη εκτεθειμένα. Πρέπει να αναφερθεί όμως, πως αυτού του είδους οι σταθμοί έχουν μεγάλο κόστος κατασκευής λόγο των πολύπλοκων συστημάτων φόρτισης και ανίχνευσης του πηνίου-δέκτη για την καλύτερη σύζευξη, καθώς προσθέτουν και ένα υπολογίσιμο επιπλέον βάρος στο drone καθώς θα πρέπει να εγκατασταθεί επάνω του το σύστημα του δέκτη με το πηνίο και με το ηλεκτρονικό κύκλωμα.

Από την άλλη μεριά υπάρχουν οι σταθμοί που η φόρτιση γίνεται με αγώγιμη επαφή των δύο μερών. Στη μεριά του drone υπάρχει μόνο ο φορτιστής του που συνδέεται με τη μπαταρία και η τροφοδοσία του φορτιστή γίνεται με κάποιους αυτοσχέδιους αγωγούς που καταλήγουν να έρθουν σε επαφή με το σταθμό φόρτισης.

Στη μεριά του σταθμού υπάρχει συνήθως κάποιο σύστημα αγώγιμων επαφών ανάλογα με τον κατασκευαστή. Έχουν γίνει πολύ ενδιαφέρουσες δημοσιεύσεις σχετικά με το τρόπο σύνδεσης του drone στα σημεία τροφοδοσίας [7] [8]. Η πρόκληση σε αυτού του είδους τους σταθμούς είναι ο τρόπος προσγείωσης του drone και η καλή επαφή με τους αγωγούς τροφοδοσίας, καθώς πρέπει να ληφθεί υπόψη η πολικότητα (αφού έχουμε DC ρεύμα) και ο προσανατολισμός του drone.



Εικόνα 5: Σταθμός φόρτισης με επαφή [21]

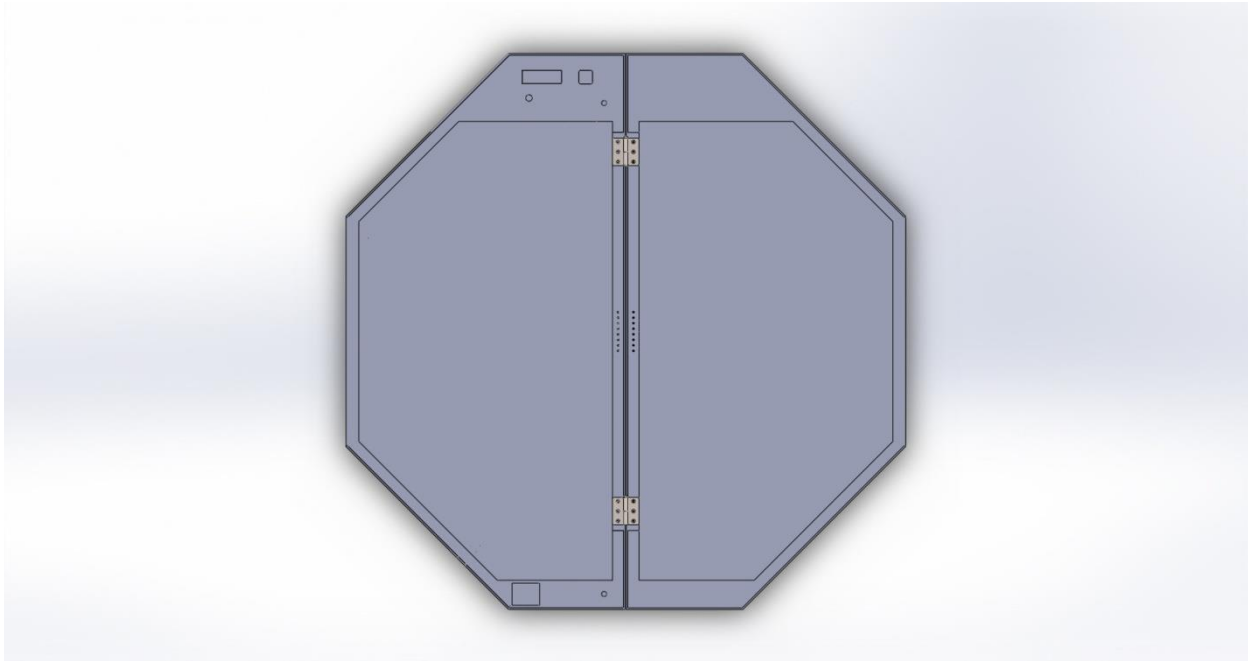
Σε αυτή την εργασία θα κατασκευάσω έναν σταθμό φόρτισης με αγώγιμη επαφή. Θα δημιουργηθεί ένα πρωτότυπο μοντέλο για την πειραματική διερεύνηση των συστημάτων και θα παρουσιαστούν τα αποτελέσματα. Θα χρησιμοποιήσω ένα σύστημα αυτοσχέδιων αισθητήρων πίεσης για την ανίχνευση του drone και θα συμπεριλάβω και σύστημα επικοινωνίας με LoRa καθώς και σύνδεση στο internet. Στο επόμενο κεφάλαιο θα παρουσιαστεί ο τρόπος σχεδίασης και κατασκευής του σταθμού καθώς και τα κυκλώματα που συμπεριλήφθηκαν στην κατασκευή.

Κεφάλαιο 3. Σχεδίαση και υλοποίηση

Σε αυτό το κεφάλαιο θα γίνει απόπειρα αναλυτικής προσέγγισης της διαδικασίας κατασκευής του σταθμού. Θα παρουσιάζεται το υλικό (hardware) που χρησιμοποιήθηκε. Η κατασκευή αυτοσχέδιων αισθητήρων πίεσης, σχηματικά διαγράμματα των ηλεκτρονικών κυκλωμάτων και οτιδήποτε άλλο σχετικό. Οι αλγόριθμοι λειτουργίας των μερών του σταθμού και ο κώδικας του λογισμικού (software) που υλοποιήθηκε, θα υπάρχουν σε παράρτημα της εργασίας, για να γίνει καλύτερα κατανοητή η όλη λειτουργία του συστήματος.

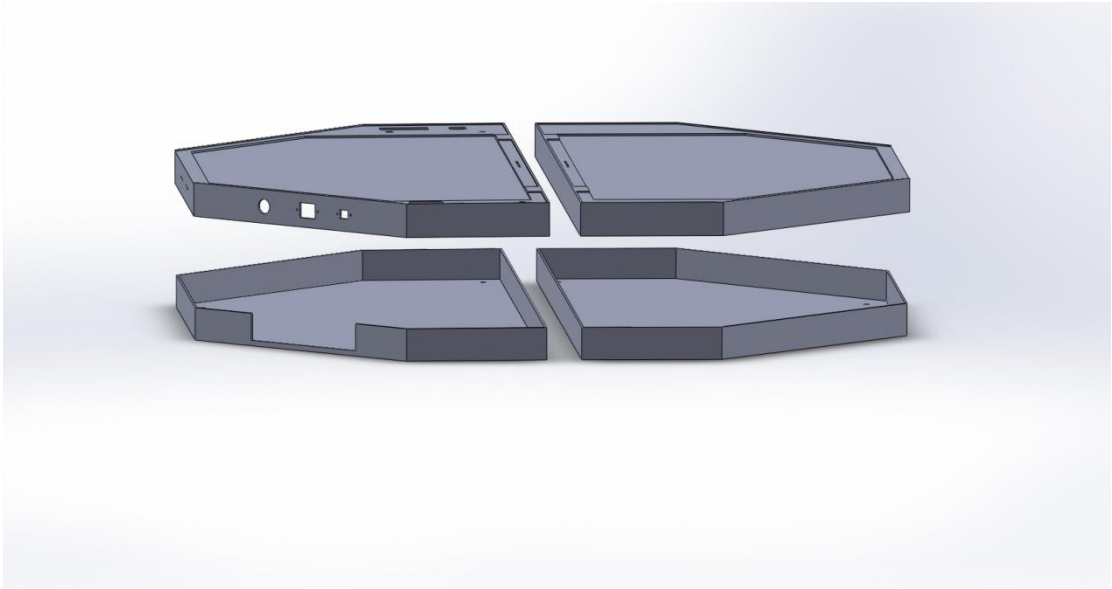
3.1 Κατασκευή του σταθμού φόρτισης

Κατά τη σχεδίαση του σταθμού, δόθηκε έμφαση στο να κατασκευαστεί ένα σύστημα που θα είναι όσο το δυνατόν ελαφρύτερο και εύκολο στη μεταφορά. Για το λόγο αυτό αποφασίστηκε ο σταθμός να έχει εξαγωνικό σχήμα διαμέτρου ενός μέτρου και να διπλώνει στη μέση με σκοπό τη μεταφορά του ως χειραποσκευή. Στην Εικόνα 6 φαίνεται η κάτοψη του σταθμού.

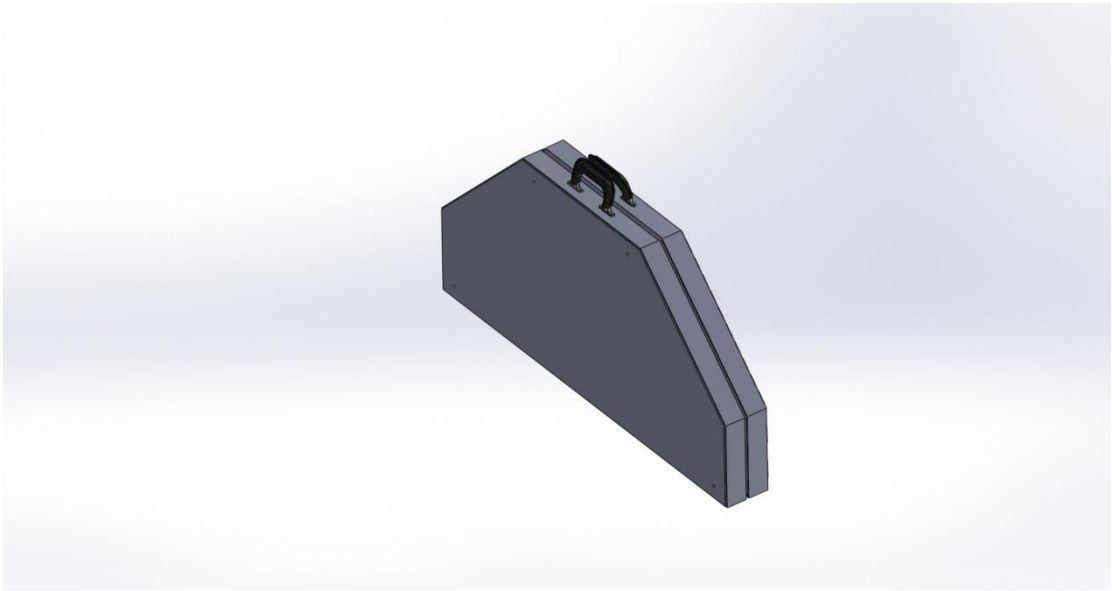


Εικόνα 6: Κάτοψη σταθμού φόρτισης

Ο σκελετός αποτελείται από τέσσερα ξεχωριστά κομμάτια, όπως φαίνεται στην Εικόνα 7, έτσι ώστε να τοποθετηθούν στο εσωτερικό τα ηλεκτρονικά που θα χρειαστούν. Επίσης το πάχος του σταθμού είναι περίπου στα 50mm. Επιπλέον τοποθετήθηκαν και δύο χερούλια για να μπορεί κάποιος να τον μεταφέρει σαν βαλίτσα όταν είναι διπλωμένος όπως φαίνεται στην Εικόνα 8.



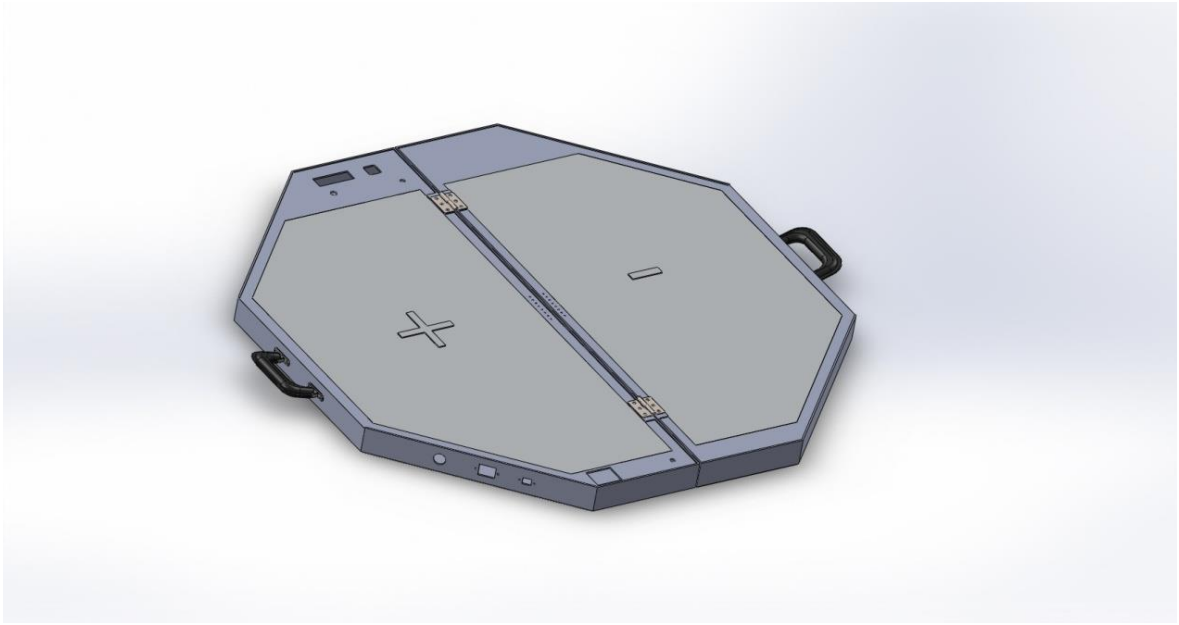
Εικόνα 7: Σκελετός σταθμού φόρτισης



Εικόνα 8: Ο σταθμός φόρτισης κατά τη μεταφορά του

Για τις ανάγκες παρουσίασης του σταθμού, το πρωτότυπο κατασκευάστηκε από αλουμινένιες γωνίες και plexiglass ώστε να φαίνεται το εσωτερικό της κατασκευής. Φυσικά μπορεί ολόκληρος ο σκελετός να είναι αλουμινένιος ή ακόμα και από ανθρακόνημα για μείωση του βάρους, αλλά το κόστος θα αυξηθεί ανάλογα.

Για την διαδικασία της φόρτισης τοποθετήθηκαν δύο φύλλα από αλουμίνιο στο πάνω μέρος του σταθμού. Ένα για τον θετικό και ένα για τον αρνητικό πόλο της DC τροφοδοσίας όπως φαίνεται στην Εικόνα 9.



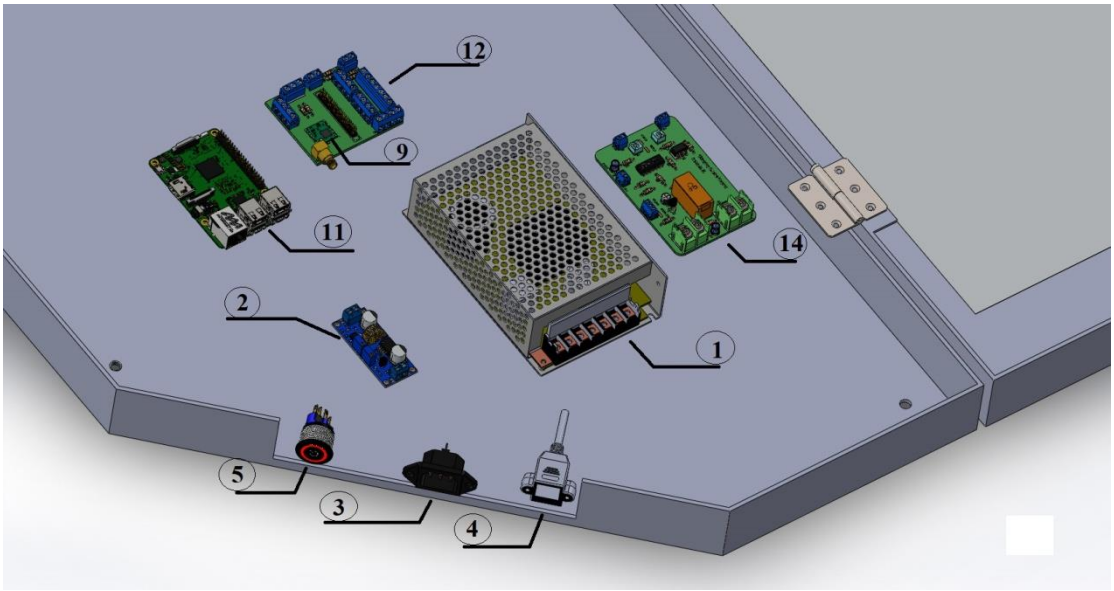
Εικόνα 9: Επαφές για τη τροφοδοσία κατά τη φόρτιση ενός drone

Οι επαφές φόρτισης αποφασίστηκε να είναι από αλουμίνιο αντί για χάλκινες για να αποφευχθεί η φθορά τους από διάβρωση και λόγω τις δυσκολίας που συνάντησα στην αναζήτηση φύλλων χαλκού. Επιπλέον επιτυγχάνεται χαμηλότερο κόστος και βάρος.

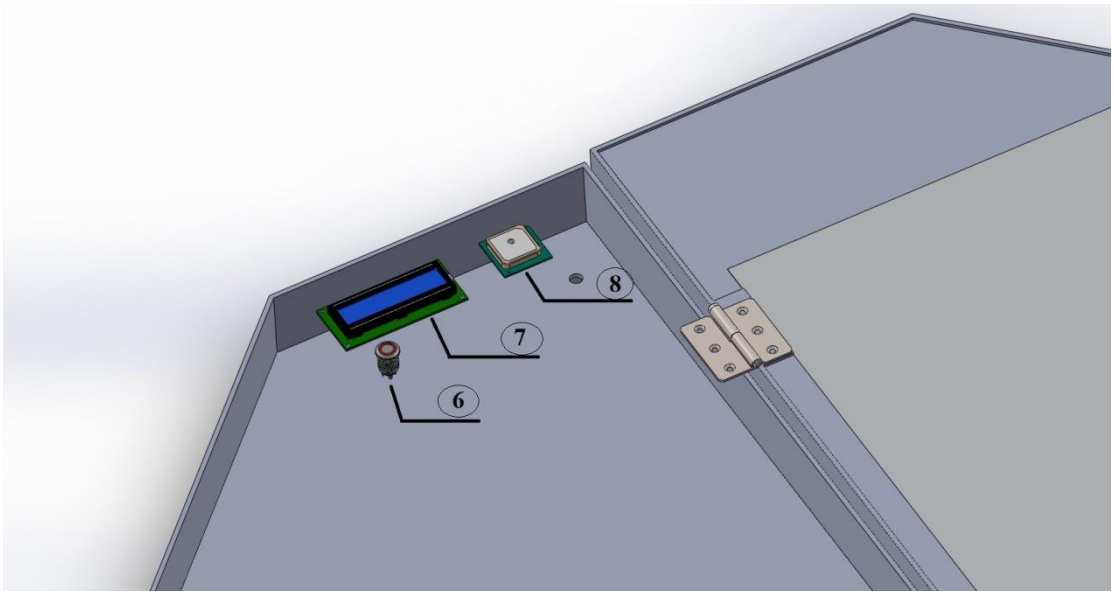
Αφού παρουσιάστηκαν τα βασικά δομικά στοιχεία του σκελετού της βάσης, στη συνέχεια θα παραθέσω μια λίστα με τα υπόλοιπα εξαρτήματα που χρησιμοποιήθηκαν στο σταθμό:

1. *Τροφοδοτικό 12V/10A*
2. *Μετατροπέας DC/DC 12V/5V*
3. *Φις τροφοδοσίας AC 230V*
4. *Φις δικτύου RJ45*
5. *Διακόπτης επαναφοράς 1 (push button 1)*
6. *Διακόπτης επαναφοράς 2 (push button 2 – menu_button)*
7. *Οθόνη LCD 16x2*
8. *Δέκτης GPS*
9. *Πλακέτα επικοινωνίας LoRa*
10. *Κεραία για το σύστημα LoRa*
11. *Raspberry Pi 3 Model B*
12. *Πλακέτα διανομής σημάτων*
13. *Αυτοσχέδιοι αισθητήρες πίεσης*
14. *Πλακέτα ελέγχου αισθητήρων πίεσης*

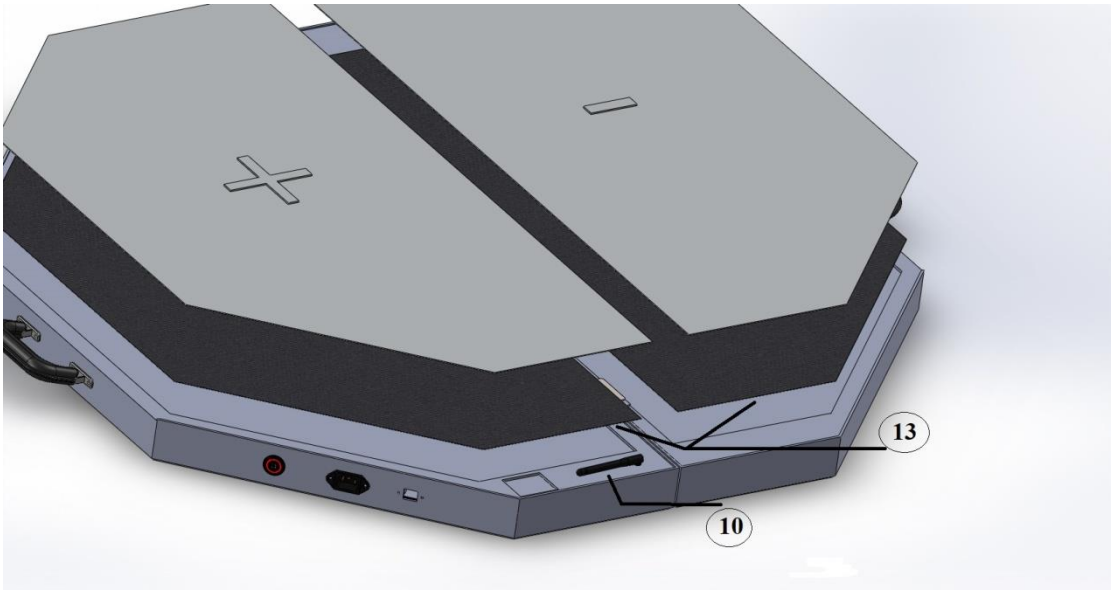
Όλα τα παραπάνω στη λίστα φαίνονται στην Εικόνα 10, Εικόνα 11 και Εικόνα 12 με τους αντίστοιχους αριθμούς.



Εικόνα 10: Ηλεκτρονικά και εξαρτήματα του σταθμού 1



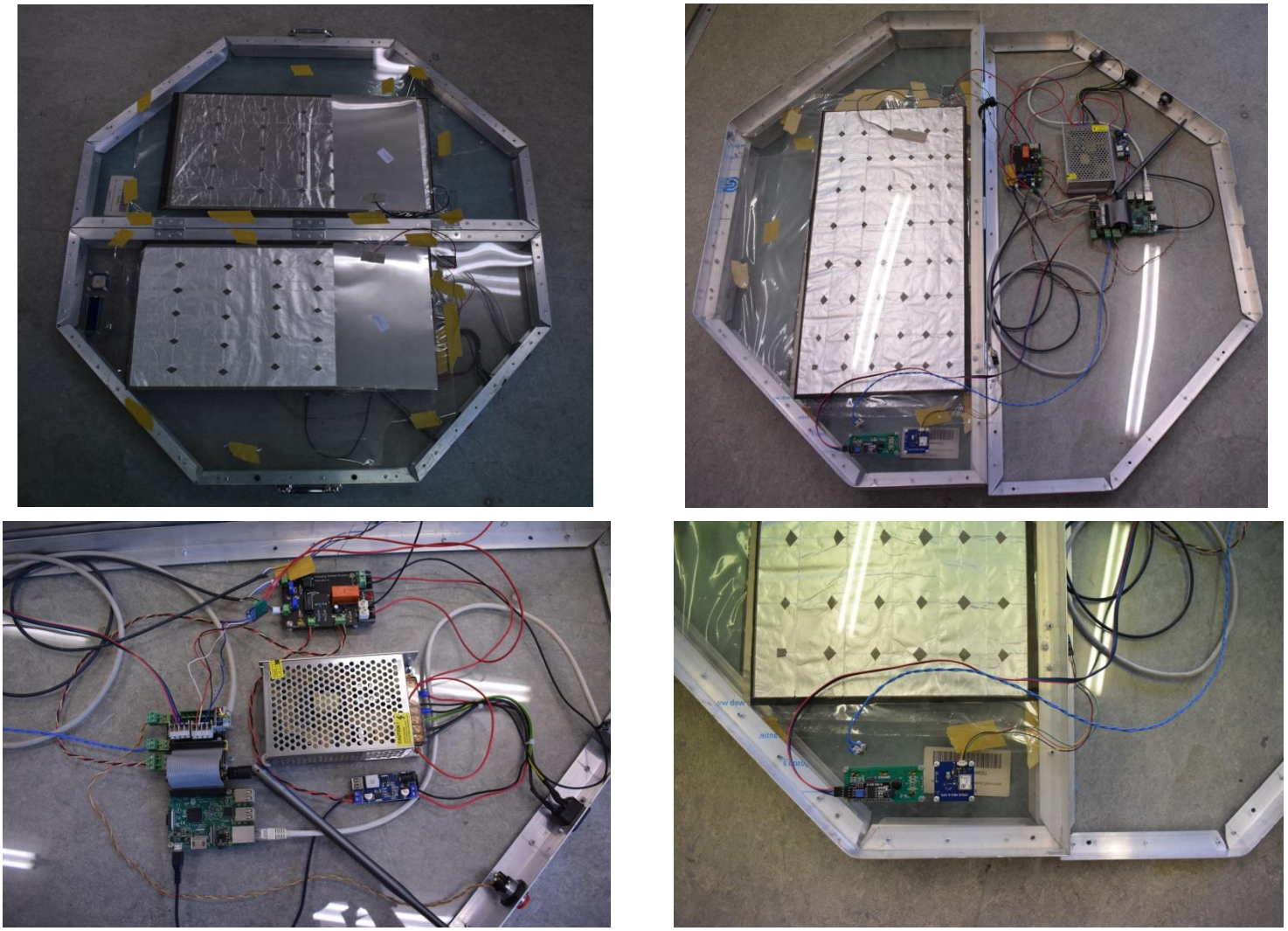
Εικόνα 11: Ηλεκτρονικά και εξαρτήματα του σταθμού 2



Εικόνα 12: Ηλεκτρονικά και εξαρτήματα του σταθμού 3

Όπως φαίνεται στις εικόνες, όλα τα ηλεκτρονικά έχουν τοποθετηθεί στο αριστερό μισό του σταθμού. Αυτό έγινε με σκοπό το δεξιό μέρος το σταθμού να έχει ελεύθερο χώρο ώστε να χρησιμοποιηθεί για αποθήκευση εργαλείων και ειδών χρήσιμων σε περίπτωση τεχνικής υποστήριξης καθώς και χώρο για μπαταρίες για χρήση σε μέρος χωρίς παροχή ρεύματος.

Στην Εικόνα 13 παρουσιάζεται το πρωτότυπο μοντέλο του σταθμού που κατασκευάστηκε για τις ανάγκες της εργασίας.



Εικόνα 13: Πρωτότυπη υλοποίηση του σταθμού

3.2 Αισθητήρες πίεσης με χρήση Velostat

Η επιλογή του τρόπου με τον οποίο θα γινόταν η ανίχνευση του drone από τη βάση κατά τη προσγείωση ώστε να ξεκινήσει το σύστημα φόρτισης ήταν ένα θέμα που με απασχόλησε αρκετά. Σε άλλες αντίστοιχες εργασίες [4] [9] υπήρχαν διάφορες λύσεις με αισθητήρες και κλαπέτα που μετακινούνται φέρνοντας το drone σε συγκεκριμένο σημείο, αλλά κάτι αντίστοιχο θα έκανε το όλο σύστημα αρκετά ογκώδες και πολύπλοκο. Η ιδανική λύση για να πληρούνται οι προϋποθέσεις για μικρό όγκο και βάρος είναι κάποιου είδους, αισθητήρας πίεσης, που όταν το drone ακουμπάει στο σταθμό, τότε θα ενεργοποιείται η φόρτιση του.

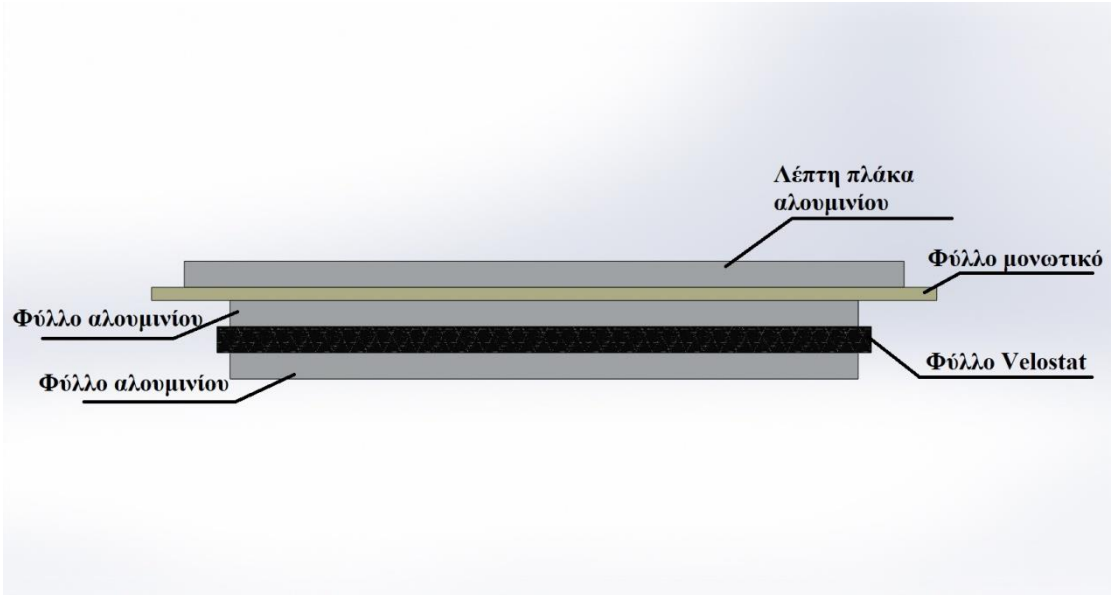
Δυστυχώς, οι επίπεδοι αισθητήρες πίεσης που υπάρχουν στην αγορά έχουν κατά βάση μικρή επιφάνεια, οπότε θα έπρεπε να φτιάξω συστοιχίες ώστε να καλύπτεται μεγαλύτερη επιφάνεια του σταθμού. Αυτό θα ανέβαζε αρκετά το κόστος κατασκευής.

Η λύση τελικά βρέθηκε σε μία τάση των τελευταίων ετών, τα wearable, τα οποία είναι έξυπνες συσκευές τις οποίες μπορεί κάποιος να τις φορέσει. Για την δημιουργία τέτοιων συσκευών χρησιμοποιούνται

διαφόρων ειδών υλικά, από ηλεκτρικά αγώγιμες κλωστές μέχρι πολυμερή υλικά που αλλάζουν τα ηλεκτρικά χαρακτηριστικά τους αναλόγως την επεξεργασία που δέχονται.

Στη γκάμα αυτών των υλικών βρήκα ένα λεπτό και εύκαμπτο φύλλο από πολυμερές υλικό το οποίο ονομάζεται Velostat ή Linqstat και είναι ηλεκτρικά αγώγιμο. Το σημαντικότερο όμως είναι πως αλλάζει η ηλεκτρική του αντίσταση μεταξύ της μίας και της άλλης επιφάνειας όταν ασκείται πίεση μεταξύ τους. Λόγω αυτού του χαρακτηριστικού κατάφερα να κατασκευάσω αισθητήρες πίεσης με ελάχιστο βάρος, οικονομικούς και εύκολους στη κατασκευή. Στην Εικόνα 14 φαίνεται ένα σχέδιο με τη δομή της κατασκευής των αισθητήρων.

Ο αισθητήρας βρίσκεται κάτω από το μονωτικό φύλλο και οι επαφές του είναι τα δύο φύλα αλουμινίου. Η λεπτή πλάκα αλουμινίου, πάνω από το μονωτικό φύλλο είναι η επιφάνεια που ακουμπάει το drone για να φορτίσει.



Εικόνα 14: Δομή κατασκευής αισθητήρων πίεσης

Στο σταθμό κατασκευάστηκαν δύο τέτοιοι αισθητήρες, ένας για κάθε πλευρά (Εικόνα 10), όπου καλύπτουν το μεγαλύτερο μέρος του σταθμού έτσι ώστε οποιουδήποτε μεγέθους drone να μπορεί να ανιχνευτεί. Επιπλέον η φόρτιση θα ξεκινάει μόνο όταν και οι δύο αισθητήρες ανιχνεύσουν αντικείμενο επάνω τους με σκοπό να αποφευχθούν περιπτώσεις εσφαλμένης ανίχνευσης.

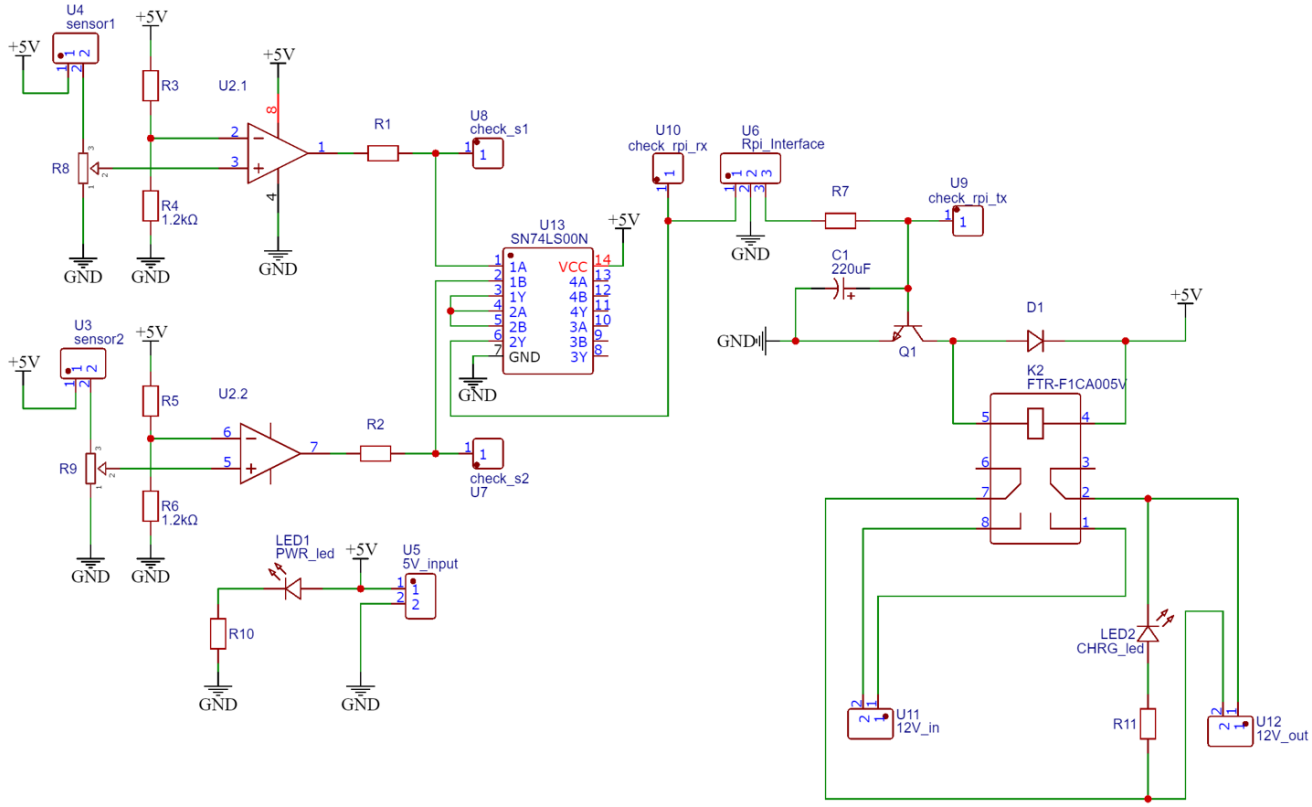
Τα φύλλα Velostat έχουν τα παρακάτω χαρακτηριστικά σύμφωνα με το κατάστημα που τα προμηθεύτηκα:

- *Dimensions: 11" x 11" (280mm x 280mm)*
- *8 mil / 0.2mm thick*
- *Weight: 18.66g*
- *Temperature Limits : -45°C to 65°C (-50°F to 150°F)*
- *Heat Sealable : Yes*
- *Volume Resistivity : <500 ohm-cm*
- *Surface Resistivity : < 31,000 ohms/sq.cm*

Ενώ τα φύλλα μπορούν να κοπούν στο σχήμα και μέγεθος που χρειάζεται, στο πρωτότυπο του σταθμού για τις δοκιμές χρησιμοποίησα από δύο φύλλα Velostat για κάθε αισθητήρα στο μέγεθος που τα αγόρασα.

3.3 Κύκλωμα ελέγχου αισθητήρων πίεσης

Για τον έλεγχο των αισθητήρων πίεσης χρειάστηκε να σχεδιαστεί ένα κύκλωμα ελέγχου. Το ηλεκτρονικό σχέδιο του κυκλώματος φαίνεται στην Εικόνα 15.



Εικόνα 15: Κυκλωματικό διάγραμμα πλακέτας ελέγχου αισθητήρων πίεσης

Το κύκλωμα αποτελείται από δύο συγκριτές με χρήση τελεστικού ενισχυτή, έναν για κάθε αισθητήρα. Ο έλεγχος αλλαγής κατάστασης του κάθε αισθητήρα γίνεται με χρήση γέφυρας Wheatstone που συνδέεται στον εκάστοτε συγκριτή και η ευαισθησία του ελέγχου ρυθμίζεται από μια μεταβλητή αντίσταση (trimmer).

Η έξοδος των συγκριτών οδηγείται σε μία πύλη AND δύο εισόδων ώστε να επιτευχθεί η ταυτόχρονη ανίχνευση και των δύο αισθητήρων. Τέλος, η έξοδος της πύλης ελέγχει ένα GPIO στο Raspberry Pi. Με αυτή τη συνδεσμολογία επιτυγχάνουμε την ταυτόχρονη ανίχνευση και των δύο αισθητήρων με χρήση μόνο ενός GPIO του ελεγκτή.

Επιπλέον στο ίδιο κύκλωμα υπάρχει και ένα relay το οποίο ελέγχεται από το Raspberry Pi με σκοπό να ενεργοποιεί το κύκλωμα φόρτισης. Ουσιαστικά οι επαφές του relay διακόπτουν τη τροφοδοσία των 12V από το τροφοδοτικό στις αλουμινένιες πλάκες φόρτισης στην επιφάνεια του σταθμού.

Όταν ένα drone προσγειωθεί πάνω στο σταθμό, οι αισθητήρες πίεσης θα αλλάξουν τιμή ηλεκτρικής αντίστασης με αποτέλεσμα οι συγκριτές να δώσουν σήμα στη πύλη NAND και από εκεί στο GPIO του Raspberry Pi. Το σήμα αυτό θα επεξεργαστεί ανάλογα στον ελεγκτή και αυτός θα ενεργοποιήσει το relay ώστε να ξεκινήσει η φόρτιση του drone.

3.4 Σύστημα επικοινωνίας LoRa

Για την επικοινωνία του σταθμού με τα drone ή με άλλους σταθμούς, αποφάσισα να χρησιμοποιήσω τεχνολογία LoRa (Long Range). Αυτή η τεχνολογία χρησιμοποιεί μια τεχνική διαμόρφωσης ευρείας περιοχής χαμηλής κατανάλωσης και λειτουργεί σε μπάντες ραδιοφωνικών συχνοτήτων (RF). Στην Ευρώπη, η νόμιμη μπάντα είναι η EU863-870 (863-870/873 MHz).

Συστήματα με LoRa χρησιμοποιούνται κυρίως για επικοινωνία συστημάτων που λειτουργούν με μπαταρία, λόγω της χαμηλής κατανάλωσης που μπορεί να προσφέρει. Επίσης η δυνατότητα επικοινωνίας σε μεγάλες αποστάσεις την κάνει ιδανική για χρήση σε μη επανδρωμένα πτητικά μέσα και κατά συνέπεια σε αυτή την εργασία.

Ακόμα έχει δημιουργηθεί ένα πρωτόκολλο επικοινωνίας, το LoRaWAN, το οποίο συνδέει πολλαπλές συσκευές που χρησιμοποιούν LoRa με το internet, ώστε να μεταφέρονται δεδομένα, κυρίως από συστήματα αισθητήρων, παντού. Φυσικά υπάρχει η δυνατότητα δημιουργίας προσωπικού δικτύου με χρήση LoRa, με χρήση gateways σε διάφορα σημεία, ακόμα και πάνω σε ένα drone, έτσι ώστε να μεταφέρονται τα δεδομένα σε ένα μεγαλύτερο κλειστό δίκτυο.

Σε αυτή την εργασία θα χρησιμοποιηθεί απευθείας επικοινωνία του σταθμού με ένα drone για το proof of concept, όμως μελλοντικά θα επιδιωχθεί η δημιουργία ενός κλειστού δικτύου με πολλαπλούς σταθμούς και drones.

Το ολοκληρωμένο κύκλωμα που χρησιμοποιήθηκε στη εργασία για το σύστημα LoRa είναι ένα RFM96 module, όπου φαίνεται στην Εικόνα 16.

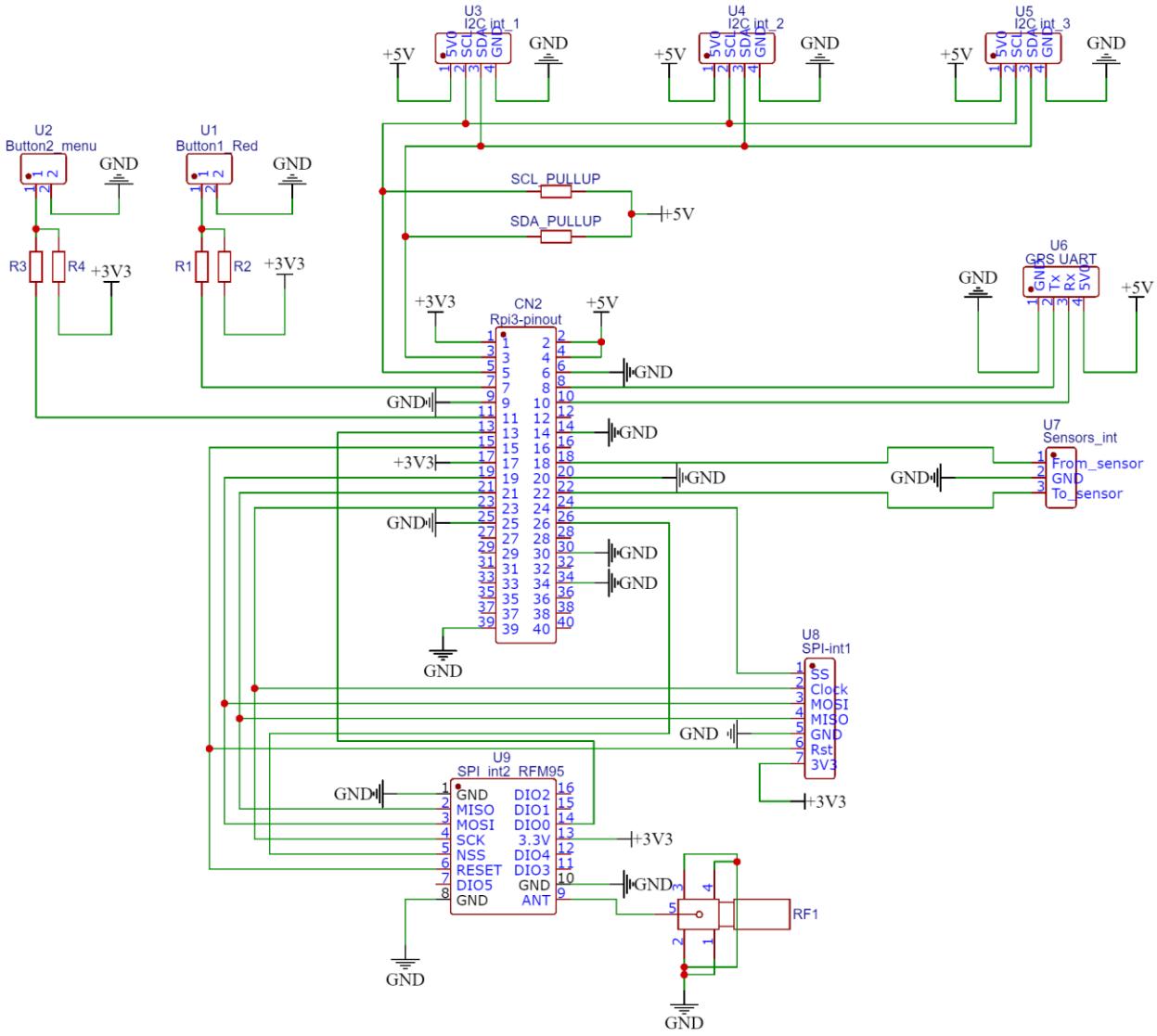
Εικόνα 16: RFM96 LoRa module 868MHz

Αυτό το module τοποθετήθηκε στη πλακέτα διανομής των σημάτων από και προς το Raspberry Pi, την οποία θα παρουσιάσω παρακάτω. Χρησιμοποιεί πρωτόκολλο SPI για την επικοινωνία με τον ελεγκτή, η τροφοδοσία του είναι στα 3.3V και η κεραία που χρησιμοποιεί είναι τοποθετημένη στην επιφάνεια του σταθμού, όπως φαίνεται στην Εικόνα 13 και συνδεδεμένη με το module με κατάλληλο ομοαξονικό καλώδιο και επαφές σύνδεσης SMA.

Σε πρώτη φάση τα μηνύματα που θα ανταλλάσουν ο σταθμός και το drone θα είναι επιβεβαιώσεις για την ταυτότητά τους για την διαδικασία της φόρτισης για λόγους ασφαλείας. Στη συνέχεια όμως θα επικοινωνούν επιπλέον για μετάδοση πληροφοριών που σχετίζονται με συλλογή δεδομένων από αισθητήρες, ενημέρωση της κίνησης στον εναέριο χώρο μεταξύ των drones, κ.α.

3.5 Πλακέτα διανομής σημάτων

Για την μεταφορά των σημάτων από και προς τον ελεγκτή του σταθμού, δηλαδή το Raspberry Pi, χρειάστηκε να σχεδιαστεί ένα κύκλωμα διανομής, όπως φαίνεται στην Εικόνα 17.



Εικόνα 17: Κυκλωματικό διάγραμμα πλακέτας διανομής σημάτων

Στον παρακάτω πίνακα εξηγούνται οι συνδέσεις του κυκλώματος:

Label	***Description***
CN2	*Σύνδεσμος 40 pin για την σύνδεση του Raspberry Pi με την πλακέτα διανομής*
U1	*Επαφή 2 pin για την σύνδεση push button αρχικοποίησης σταθμού*
U2	*Επαφή 2 pin για την σύνδεση push button ελέγχου λειτουργιών*
U3	*Επαφή 4 pin για τη σύνδεση LCD οθόνης (I2C protocol)*
U4, U5	*Επαφές 4 pin για τη σύνδεση επιπλέων συσκευών (I2C protocol)*
U6	*Επαφή 5 pin για τη σύνδεση του GPS module (UART protocol)*
U7	*Επαφή 3 pin για τη σύνδεση του κυκλώματος ελέγχου των αισθητήρων πίεσης*
U8	*Επαφή 7 pin για τη σύνδεση συσκευής με SPI protocol*
U9	*Θέση για το RFM96 LoRa module*
RF1	*Επαφή SMA για σύνδεση κεραίας LoRa*

3.6 Τεχνικά χαρακτηριστικά και τελικές επισημάνσεις

Ηλεκτρικά χαρακτηριστικά:

- *Τάση εισόδου : 230V AC, 50Hz*
- *Τάση εξόδου: 12V DC*

Φυσικά χαρακτηριστικά:

- *Διαστάσεις: Επιφάνεια 6 πλευρών 410mm/πλευρά, ύψος 55mm*
- *Τρόπος μεταφοράς: Αναδίπλωση και χρήση χειρολαβών*
- *Βάρος: 13,5 kg*

Συνδεσιμότητα και επικοινωνία:

- *RJ45 επαφή για καλώδιο Ethernet*
- *Wifi*
- *LoRa*
- *GPS*

Προορίζεται για:

- *Drone διαμέτρου από 150mm έως 950mm με τον απαραίτητο εξοπλισμό*

Ένα ζήτημα το οποίο θα λυθεί σε μελλοντική εργασία είναι η ανίχνευση του σταθμού από το drone. Το εκάστοτε drone θα γνωρίζει τις συντεταγμένες του σταθμού αλλά θα πρέπει να μπορέσει να προσγειωθεί με όσο το δυνατόν καλύτερη ακρίβεια επάνω του. Συνήθως για μια ακριβής προσγείωση χρησιμοποιείται μια κάμερα κάτω από το drone που «βλέπει» το έδαφος. Ανάλογα με τον αλγόριθμο που έχει σχεδιαστεί στον ελεγκτή που βρίσκεται στο drone, μπορεί να ανιχνευθεί κάποιο συγκεκριμένο μοτίβο ή κάποιο φώς, συνήθως υπέρυθρο (IR beacon) ώστε το drone να μετακινείται μέχρι να κεντράρει στην εικόνα τον στόχο.

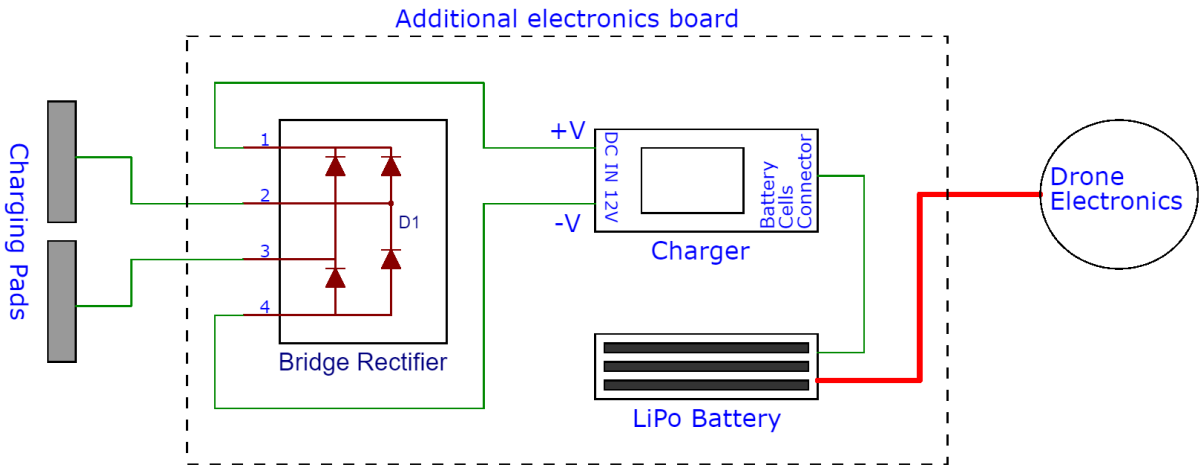
Ο σταθμός έχει σχεδιαστεί με τέτοιο τρόπο ώστε το σχήμα του να έχει αρκετά ξεχωριστό μοτίβο για ανίχνευση καθώς επίσης μπορούν εύκολα να εγκατασταθούν υπέρυθρα LED στο κέντρο του για ανίχνευση της φωτεινής δέσμης.

Επίσης θα προβλέπεται και η τοποθέτηση του σταθμού προς συγκεκριμένη κατεύθυνση, για παράδειγμα προς το βορρά. Η γνώση της τιμής του προσανατολισμού θα βοηθάει το drone να προσγειώνεται κατά μήκος των αγώγιμων επαφών του σταθμού.

3.7 Απαραίτητος εξοπλισμός για το drone

Για να μπορέσει κάποιο drone να χρησιμοποιήσει το σταθμό φόρτισης θα πρέπει να έχει εξοπλιστεί με κάποια συγκεκριμένα συστήματα. Αρχικά θα πρέπει να έχει πάνω του τον φορτιστή των μπαταριών του, συνδεδεμένο με τις μπαταρίες και να έχει φροντίσει να υπάρχουν αγώγιμα μέρη εκεί που το drone ακουμπάει με το έδαφος. Κατά κανόνα αυτά τα σημεία βρίσκονται στα πέλματα των ποδιών του drone. Έτσι θα μπορεί να μεταφερθεί ενέργεια από τον σταθμό που θα προσγειωθεί το drone μέχρι τον φορτιστή των μπαταριών.

Επιπλέον, θα πρέπει να υπάρχει και κάποιο κύκλωμα αλλαγής πολικότητας ώστε να μην υπάρχει ανάγκη το drone να προσγειώνεται με συγκεκριμένο προσανατολισμό αλλά ο φορτιστής να δέχεται τη σωστή πολικότητα. Στην Εικόνα 18 φαίνεται το κυκλωματικό διάγραμμα για παραπάνω σύστημα.



Εικόνα 18: Κυκλωματικό διάγραμμα εξοπλισμού drone

3.8 Αυτόματη προσγείωση του drone

Για την προσέγγιση και προσγείωση του drone στον σταθμό φόρτισης είναι ανάγκη να έχει γίνει κάποια πρόβλεψη για έναν μηχανισμό ανίχνευσης του σταθμού από το εκάστοτε drone. Καθώς το κάθε drone που είναι εγγεγραμμένο στο δίκτυο των σταθμών γνωρίζει τις συντεταγμένες που είναι τοποθετημένος ο κάθε σταθμός, μπορεί εύκολα να κινηθεί προς αυτή τη τοποθεσία με χρήση αυτών των συντεταγμένων και του προσωπικού του GPS. Αυτός ο τρόπος είναι ο ιδανικός για να μπορέσει ένα drone να προσεγγίσει έναν σταθμό σε απόσταση τέτοια, που ο σταθμός να είναι αρκετά κοντά ώστε να τεθεί σε λειτουργία κάποιο σύστημα για προσγείωση με ακρίβεια.

Η δυσκολία έγκειται όταν το drone έχει πλησιάσει το σταθμό για να προσγειωθεί επάνω του. Σε αυτή τη περίπτωση υπάρχουν διάφορες επιλογές, ώστε η προσγείωση να είναι όσο το δυνατόν ποιο ακριβής. Ιδανική επιλογή είναι η τοποθέτηση μίας κάμερας κάτω από το drone, όπου με χρήση κατάλληλου λογισμικού επεξεργασίας εικόνας (image processing) και ανίχνευσης αντικειμένων (object detection) μπορεί το drone να ανιχνεύσει το σταθμό φόρτισης και να προσγειωθεί. Στη συνέχεια θα παρουσιαστούν δύο

προτάσεις, όπου χρησιμοποιούν την τεχνική της επεξεργασίας εικόνας, ως παραδείγματα προς μελλοντικές εργασίες σχετικά με το θέμα.

3.8.1 Ανίχνευση και αναγνώριση προτύπου

Όπως αναφέρθηκε και σε προηγούμενη παράγραφο, ο σταθμός φόρτισης σχεδιάστηκε με τέτοιο τρόπο ώστε να είναι εύκολα ανιχνεύσιμος από λογισμικά αναγνώρισης αντικειμένων. Με τις κατάλληλες τροποποιήσεις θα μπορούσε να προσαρμοστεί επάνω στην επιφάνειά του ένα ξεχωριστό σχήμα για την αναγνώρισή του από κάποιο drone. Χρησιμοποιώντας έτσι το ιδιαίτερο σχήμα του, όπως φαίνεται στην Εικόνα 19, δημιουργείται ένα μαύρο μοτίβο που θα μπορούσε να ανιχνευτεί από κάποιο drone για την προσγείωσή του με ακρίβεια.

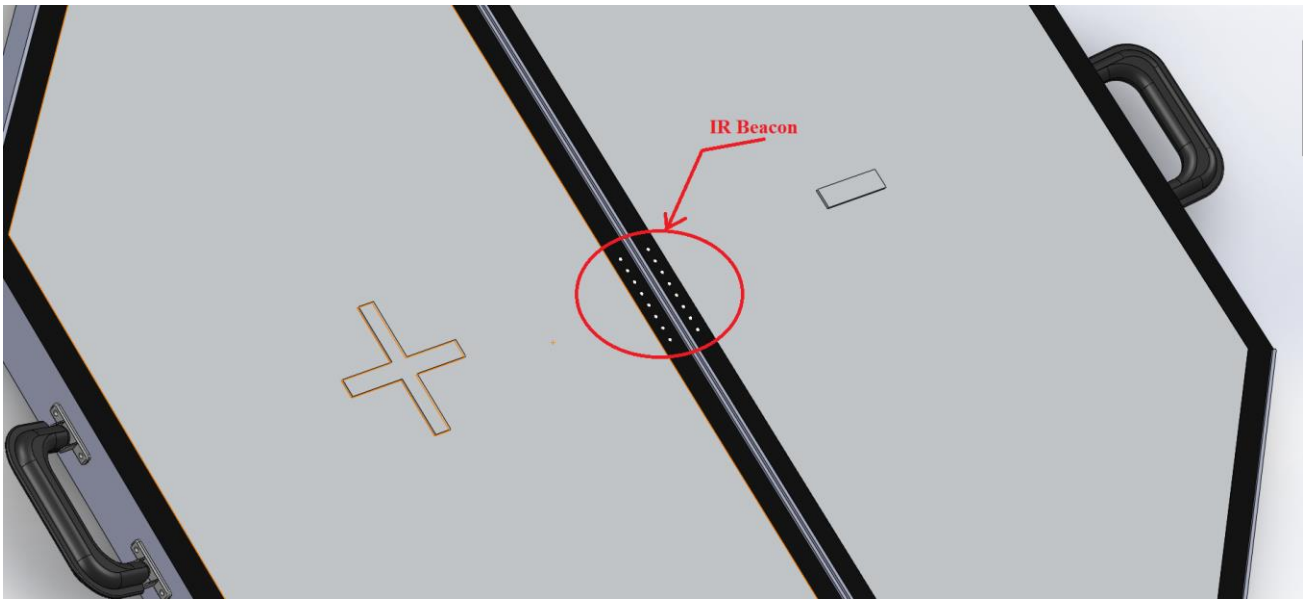
Εικόνα 19: Μοτίβο αναγνώρισης κατά τη προσγείωση

Θα ήταν έτσι δυνατό να εκπαιδευτεί ένα μοντέλο μηχανικής μάθησης και να ενσωματωθεί στο λογισμικό του drone. Όταν αυτό θα φτάσει με τη χρήση του GPS κοντά στο σταθμό ώστε να έχει οπτική επαφή, θα ενεργοποιήσει την κάμερα και κινούμενο κατάλληλα θα φέρνει το σταθμό στην μέση της εικόνας μέχρι την τελική προσγείωση.

3.8.2 IR Beacon – Φάρος υπερύθρων

Με τη χρήση του παραπάνω μοτίβου συναντάμε πρόβλημα κατά της νυχτερινές ώρες που δεν υπάρχει φώς για την καλή ανίχνευση του σταθμού. Αν και το drone μπορεί να έχει κάποια πηγή φωτός για να φωτίσει τον σταθμό, υπάρχει η λύση του IR Beacon. Ακολουθώντας την ίδια τεχνολογία ανίχνευσης με την κάμερα στο drone, αντί για μοτίβο πάνω στο σταθμό, θα μπορούσαν να υπάρχουν ενσωματωμένα υπέρυθρα φώτα

LED στη μέση του σταθμού (Εικόνα 20) όπου η κάμερα του drone θα τα έβλεπε, ίσως και από ποιο μακριά και κεντράροντας τη πηγή του φωτός στην εικόνα θα μπορούσε να προσγειωθεί με πολύ καλή ακρίβεια ακόμα και στο σκοτάδι.



Εικόνα 20: IR Beacon στο κέντρο του σταθμού φόρτισης

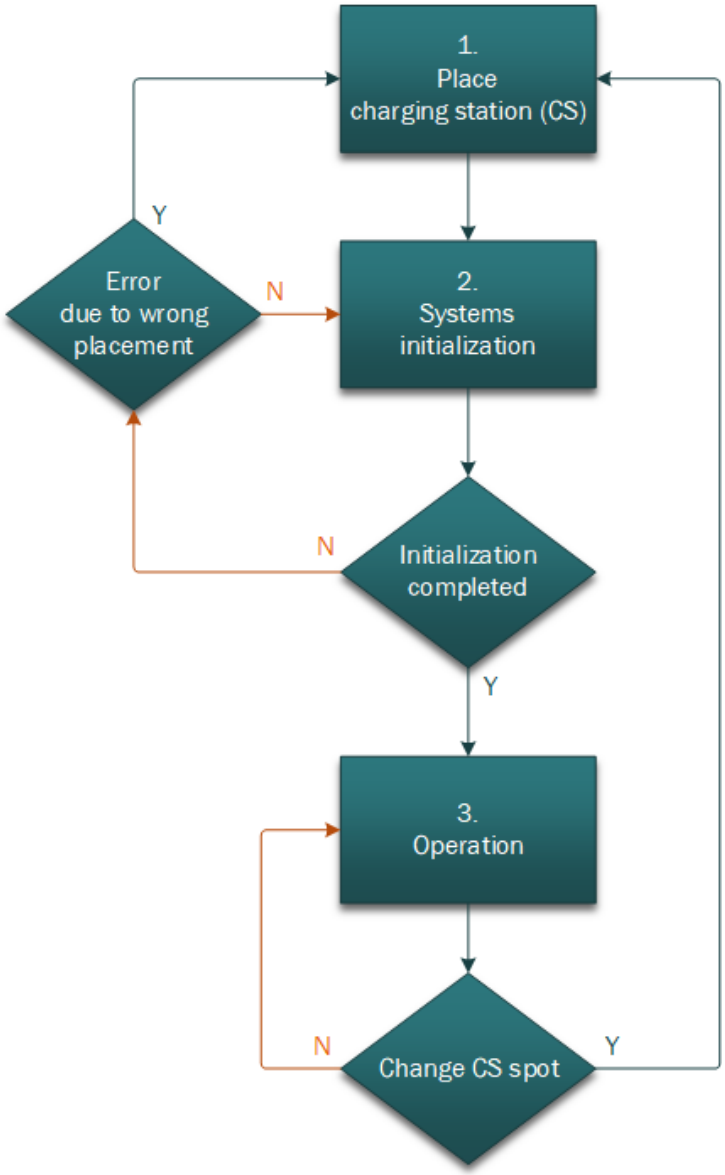
Ένα ακόμα πλεονέκτημα της χρήσης υπέρυθρου φωτός είναι πως ο άνθρωπος δεν μπορεί να το δει, ενώ είναι απολύτως ορατό από οποιαδήποτε κάμερα που δεν έχει φίλτρο υπέρυθρου φωτός. Με αυτό το τρόπο δεν υπάρχει κάποια περίπτωση να γίνει ενοχλητικό το φώς κατά τη διάρκεια της νύχτας για τον οποιονδήποτε.

Επίσης πρέπει να σημειωθεί πως αν και οποιαδήποτε κάμερα θα μπορούσε να ανιχνεύσει το φώς των υπερύθρων, υπάρχουν και κάμερες που εξειδικεύονται σε αυτή ακριβώς την δουλειά με χρήση ειδικών φίλτρων που απομονώνουν το υπέρυθρο φως. Έτσι η ανίχνευση και τελικά η προσγείωση μπορεί να γίνει με τη μέγιστη ακρίβεια.

Κεφάλαιο 4. Λειτουργία του σταθμού φόρτισης

Σε αυτό το κεφάλαιο θα παρουσιαστούν οι αλγόριθμοι λειτουργίας του σταθμού φόρτισης. Θα αναλυθούν με χρήση διαγραμμάτων ροής για την καλύτερη κατανόηση, η σωστή τοποθέτηση, η αρχικοποίηση των επιμέρους συστημάτων και οι διαδικασίες που θα εκτελούνται κατά την διάρκεια λειτουργίας του.

Αρχικά στην Εικόνα 21 φαίνεται ο κύκλος λειτουργίας του σταθμού, ο οποίος αποτελείται από τρία μέρη που θα αναλυθούν στη συνέχεια.



Εικόνα 21: Διάγραμμα κύκλου λειτουργίας του σταθμού φόρτισης

Το πρώτο μέρος αποτελείται από την διαδικασία τοποθέτησης του σταθμού. Το δεύτερο μέρος έχει να κάνει με την αρχικοποίηση όλων των συστημάτων και των επικοινωνιών και το τρίτο μέρος περιλαμβάνει την κανονική λειτουργία του σταθμού, δηλαδή τις διεργασίες που εκτελούνται κατά την αναμονή του σταθμού για την προσέγγιση κάποιου drone καθώς και την μεταφορά πληροφοριών σχετικά με το δίκτυο των σταθμών και την συλλογή δεδομένων από τους αισθητήρες του.

4.1 Τοποθέτηση του σταθμού φόρτισης

Όταν ένας χρήστης προμηθευτεί τον σταθμό φόρτισης θα πρέπει να βρει το σωστό μέρος να τον τοποθετήσει. Ένα καλό μέρος θα ήταν μια ταράτσα πολυκατοικίας ή μια αυλή χωρίς ψηλή και πυκνή βλάστηση. Όμως, σε οποιαδήποτε επίπεδη επιφάνεια όπου να υπάρχει ελεύθερη πρόσβαση για ένα drone να προσγειωθεί θα ήταν δυνατόν ένας σταθμός να εγκατασταθεί δίχως κανένα πρόβλημα.

Στο χώρο που θα γίνει η εγκατάσταση, θα πρέπει ο χρήστης να έχει προβλέψει να υπάρχει μία πρίζα ρεύματος, τάσης 230V AC για την μόνιμη τροφοδότηση του σταθμού και προαιρετικά ένα καλώδιο δικτύου για την σύνδεση του σταθμού στο internet. Σε περίπτωση που δεν υπάρχει καλώδιο θα πρέπει ο χρήστης να έχει ενημερώσει τον πάροχο για τα στοιχεία σύνδεσης του ασύρματου δικτύου του (wifi), ώστε να γίνουν οι απαραίτητες ρυθμίσεις του σταθμού για να συνδεθεί ασύρματα στο internet. Αν για κάποιο λόγο ο χρήστης δεν έχει ή δεν επιθυμεί σύνδεση στο internet τότε ο σταθμός θα λειτουργήσει με κάποιες δυνατότητες απενεργοποιημένες. Όπως την ενημέρωση των αλλαγών του δικτύου των σταθμών σε πραγματικό χρόνο.

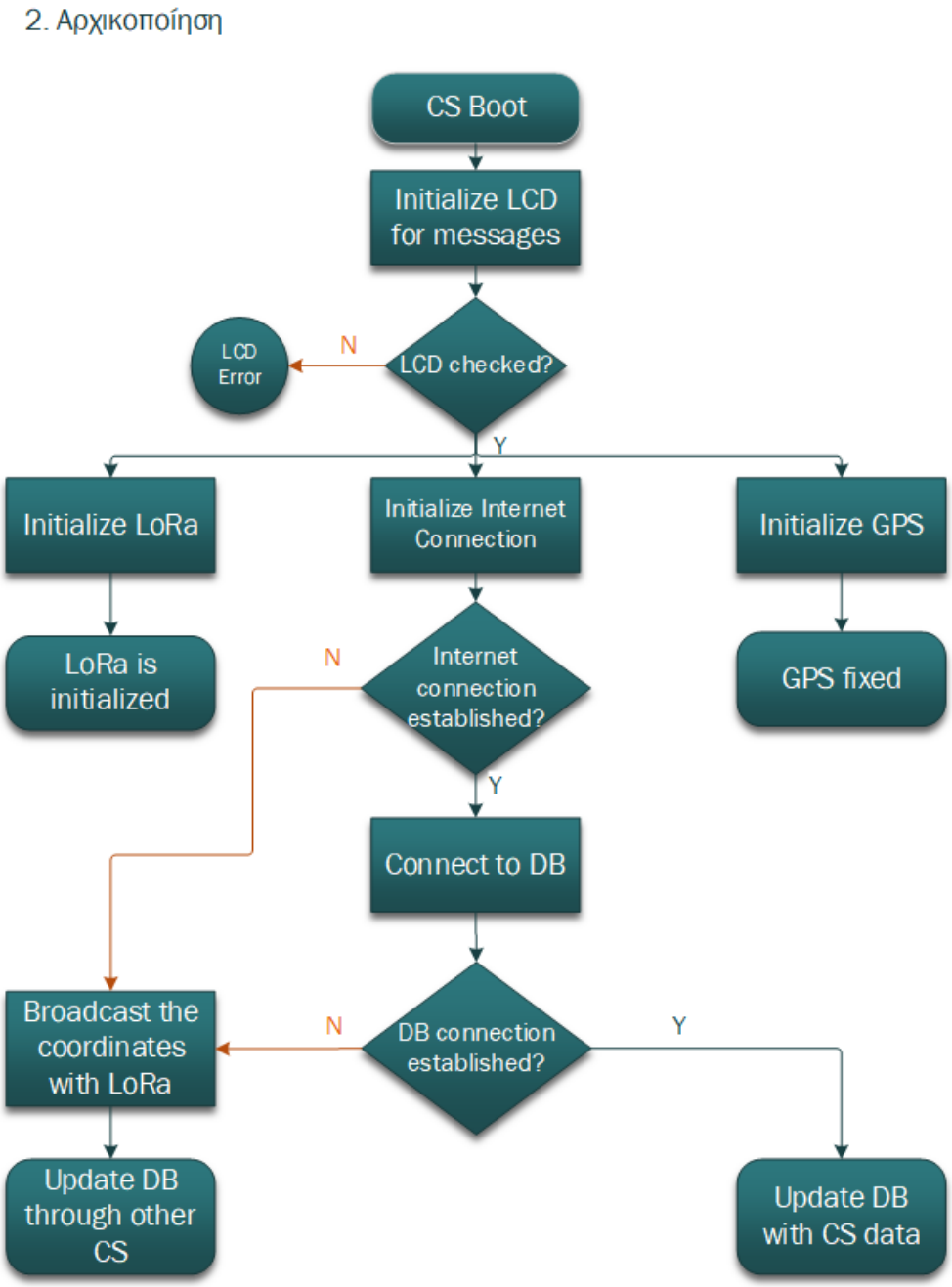
Ένα τελευταίο προαπαιτούμενο για την σωστή εγκατάσταση του σταθμού είναι η τοποθέτηση του με προσανατολισμό προς τον βορρά. Αυτό θα είναι πολύ εύκολο με την προσθήκη μιας αναλογικής πυξίδας πάνω στο σταθμό. Ο προσανατολισμός αυτός θα βοηθήσει στην ακριβέστερη προσγείωση του drone και θα αποτρέψει το σύστημα από διάφορα σφάλματα κατά τη προσγείωση.

Όταν ολοκληρωθεί η τοποθέτηση του σταθμού ο χρήστης πρέπει να συνδέσει αρχικά το καλώδιο δικτύου (αν υπάρχει) στο φις δικτύου RJ45 του σταθμού (Εικόνα 11– No 4). Στη συνέχεια πρέπει να συνδέσει την μία άκρη του καλωδίου ρεύματος στο φις τροφοδοσίας AC 230V του σταθμού (Εικόνα 11– No 3) και την άλλη άκρη στη πρίζα. Τότε ο σταθμός θα εκκινήσει και θα αρχίσει η αρχικοποίηση των συστημάτων.

Ο χρήστης θα χρειαστεί να παρακολουθεί την οθόνη LCD του σταθμού (Εικόνα 12–No 7) και να ακολουθήσει τα βήματα που θα εμφανιστούν εκεί.

4.2 Αρχικοποίηση συστημάτων

Στην Εικόνα 22 φαίνεται το διάγραμμα ροής της διαδικασίας αρχικοποίησης των επιμέρους συστημάτων και συσκευών κατά την εκκίνηση του σταθμού.



Εικόνα 22: Διάγραμμα ροής αρχικοποίησης των συστημάτων του σταθμού φόρτισης

Αφού ο σταθμός εκκινήσει, η πρώτη συσκευή που πρέπει να αρχικοποιηθεί είναι η οθόνη LCD ώστε να μπορεί ο χρήστης να βλέπει τα μηνύματα κατάστασης και να ακολουθεί τις οδηγίες που θα εμφανιστούν αν υπάρχει ανάγκη.

Σκοπός τις αρχικοποίησης των συστημάτων, εκτός από την εκκίνηση της λειτουργίας τους, είναι να μπορέσει ο σταθμός να ενημερώσει το δίκτυο για την ύπαρξή του, ώστε να μπορεί κάποιο drone να τον χρησιμοποιήσει.

Αυτό απαιτεί την γνώση των συντεταγμένων όπου είναι τοποθετημένος, με χρήση του δέκτη GPS και κάποιον τρόπο ώστε να μεταφερθούν οι πληροφορίες στη βάση δεδομένων του δικτύου.

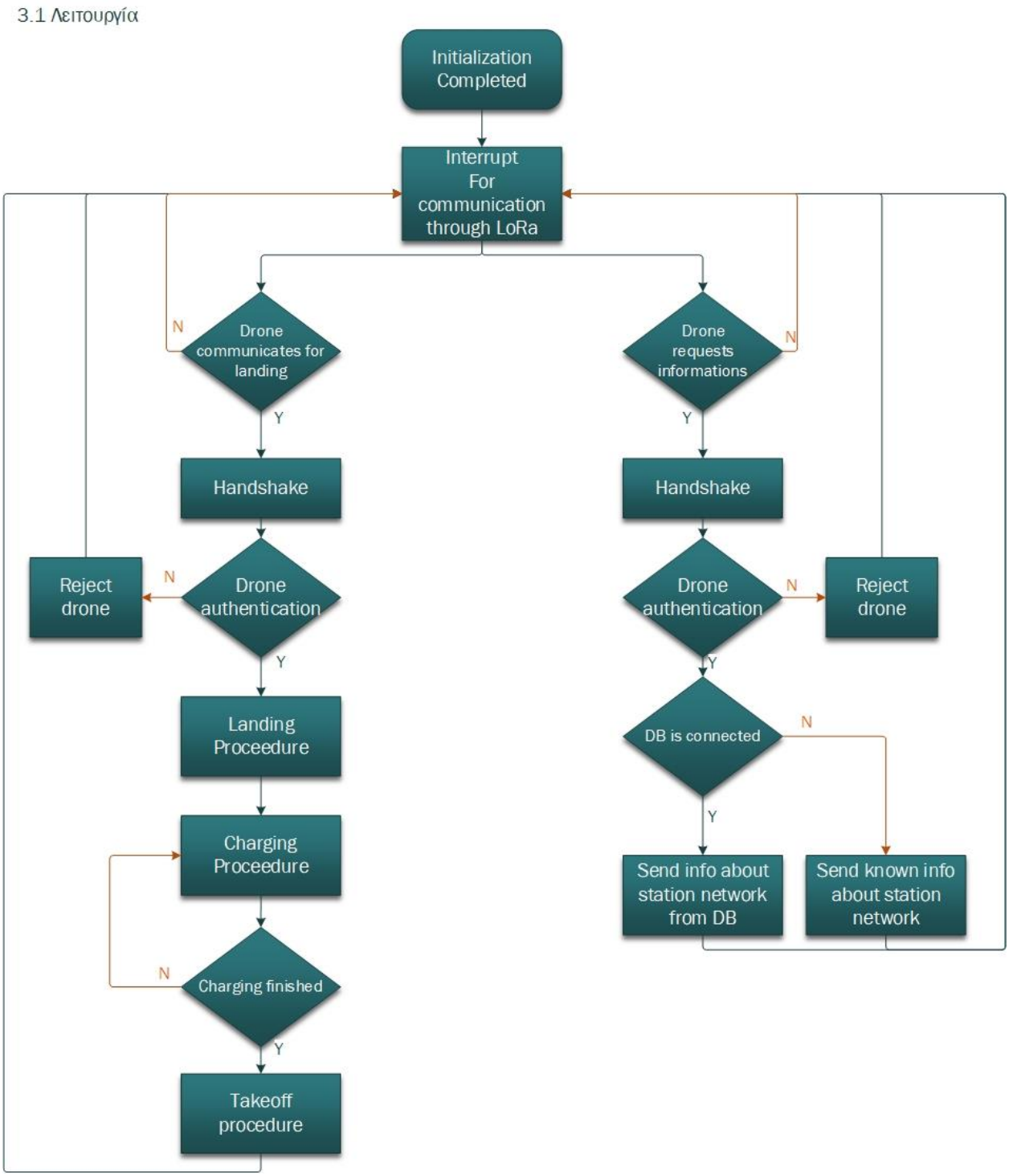
Ο ευκολότερος τρόπος μεταφοράς των πληροφοριών προς τον server του δικτύου είναι μέσω internet. Σε περίπτωση όμως που ο χρήστης δεν έχει επιλέξει τη σύνδεση του σταθμού στο internet, θα υπάρχει η δυνατότητα να σταλούν οι πληροφορίες με χρήση LoRa, σε κάποιον άλλο, κοντινό σταθμό ο οποίος με τη σειρά του μπορεί να ενημερώσει τη βάση δεδομένων.

Αν όλα γίνουν χωρίς πρόβλημα στην διαδικασία αρχικοποίησης τότε θα ξεκινήσει η κανονική λειτουργία του σταθμού. Σε κάθε άλλη περίπτωση θα έχει συμβεί κάποιο σφάλμα το οποίο δεν αφήνει το σταθμό να συνεχίσει τη λειτουργία του. Σε κάθε περίπτωση θα υπάρχουν κατάλληλα μηνύματα στην οθόνη LCD του σταθμού για την ενημέρωση του χρήστη.

4.3 Κανονική λειτουργία

Κατά την κανονική λειτουργία του, ο σταθμός έχει δύο βασικές λειτουργίες να επιτελέσει. Αρχικά, καλείται να διαχειρίζεται τα αιτήματα από τα drone μέσω επικοινωνίας LoRa. Τα αιτήματα αυτά μπορεί να είναι, είτε το drone να ζητήσει άδεια για προσγείωση ώστε να φορτίσει την μπαταρία του, είτε να ζητήσει πληροφορίες σχετικά με δεδομένα που αφορούν το δίκτυο των σταθμών.

Στην Εικόνα 23 παρουσιάζεται το διάγραμμα ροής για την περίπτωση που ο σταθμός εξυπηρετεί κάποιο drone.

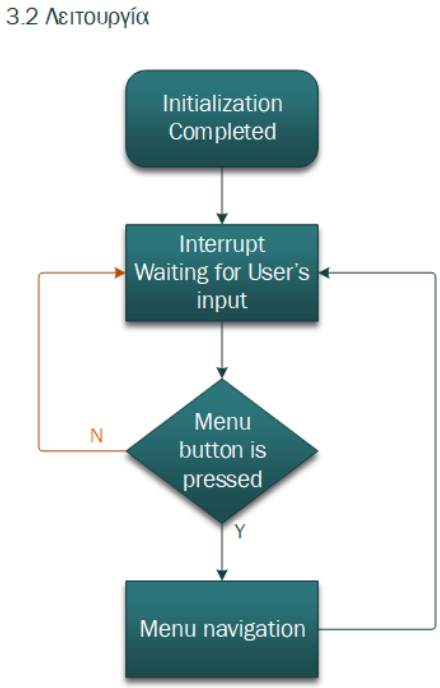


Εικόνα 23: Διάγραμμα ροής κανονικής λειτουργίας του σταθμού φόρτισης (Περίπτωση 1)

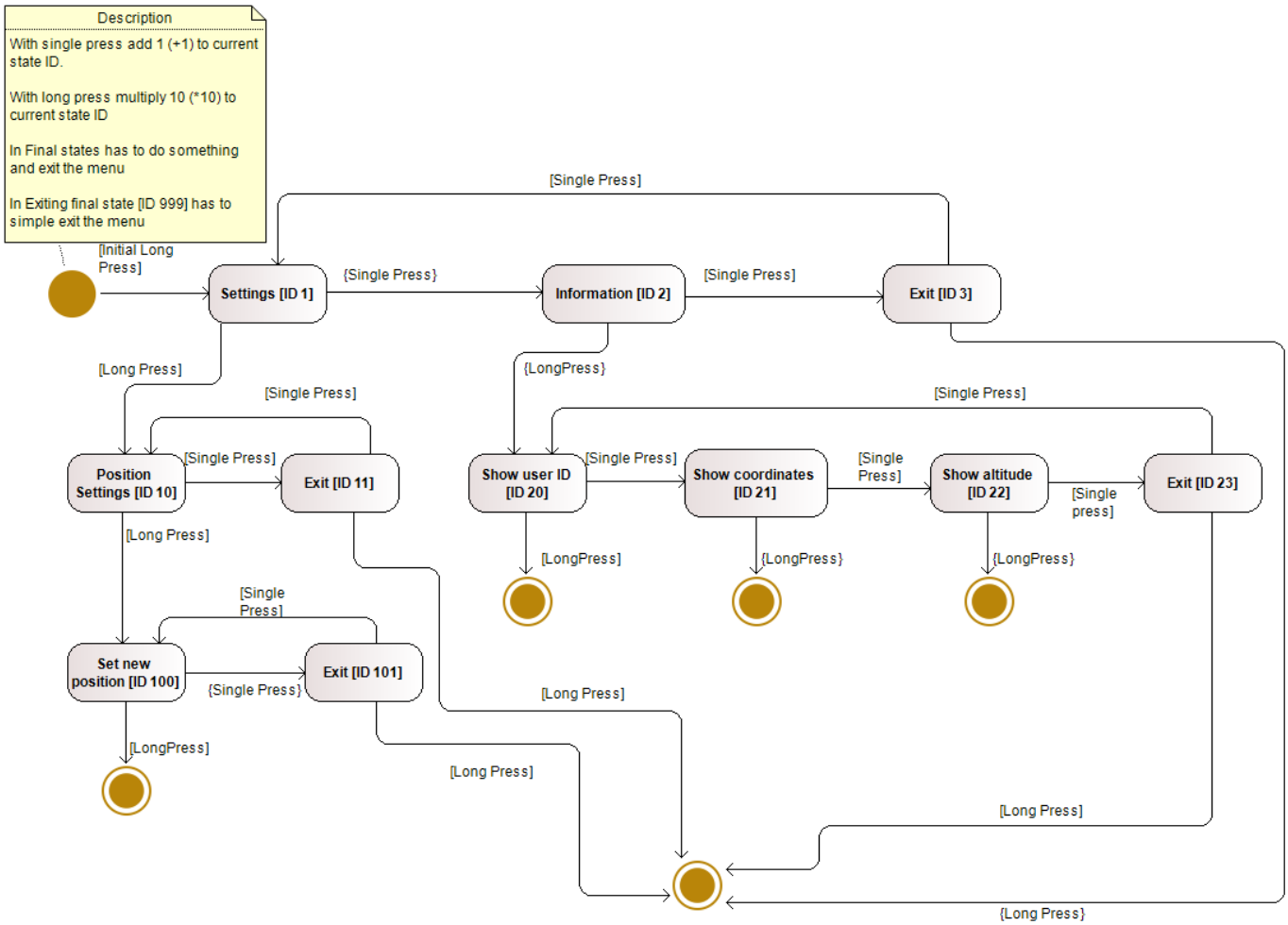
Στη συνέχεια, μέσω ενός μενού επιλογών ο χρήστης θα μπορεί να μάθει μερικές πληροφορίες για τον σταθμό, όπως τις συντεταγμένες που τον έχει τοποθετήσει, το προσωπικό του ID σε περίπτωση που ζητηθεί από κάποιον εξουσιοδοτημένο τεχνικό σε περίπτωση βλάβης ή ακόμα να επιλέξει μεταφορά του σταθμού και να ενημερώσει την βάση δεδομένων με τις νέες συντεταγμένες.

Στην Εικόνα 24 παρουσιάζεται το διάγραμμα ροής για την περίπτωση που ο σταθμός εξυπηρετεί τον χρήστη μέσω του μενού επιλογών. Επίσης στην Εικόνα 25 φαίνεται το διάγραμμα ροής του μενού επιλογών με χρήση του menu_button (Εικόνα 12– No 6). Το μενού έχει σχεδιαστεί ως μια μηχανή καταστάσεων, όπου

κάθε κατάσταση είναι μια επιλογή και έχει έναν μοναδικό αριθμό. Η μεταφορά από κατάσταση σε κατάσταση γίνεται με το απλό πάτημα (Single Press) του button είτε με παρατεταμένο πάτημα (Long Press). Με αυτή την τεχνική της μηχανής καταστάσεων είναι εύκολο να προστεθούν περισσότερες επιλογές στο μενού ανάλογα με τις μελλοντικές ανάγκες.

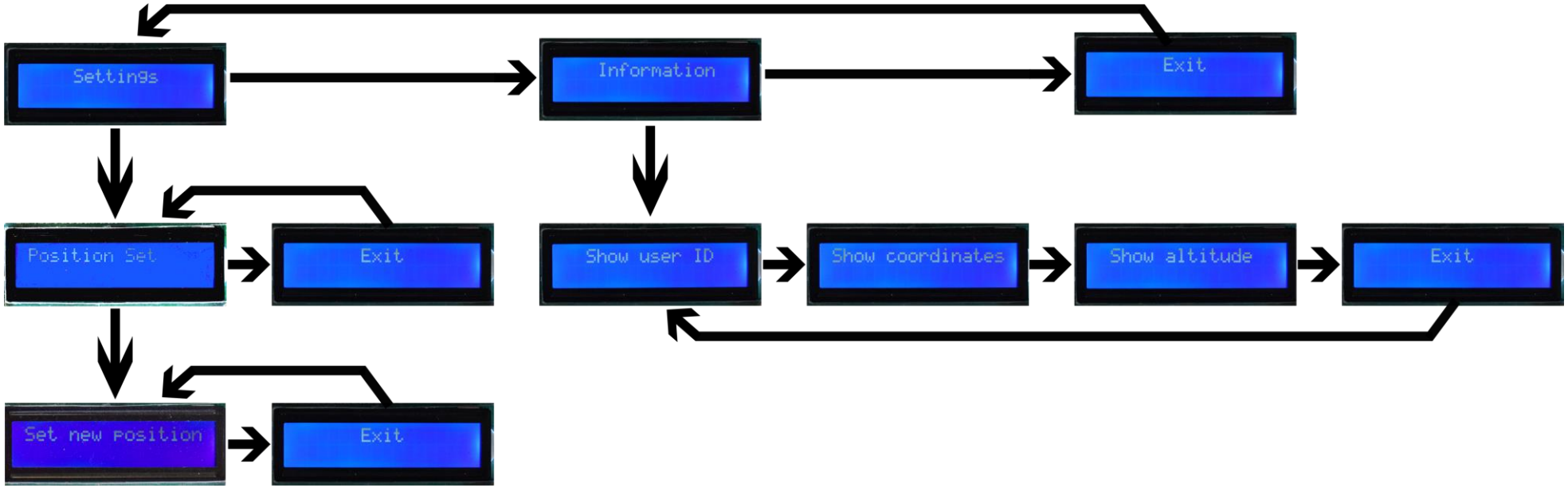


Εικόνα 24: Διάγραμμα ροής κανονικής λειτουργίας του σταθμού φόρτισης (Περίπτωση 2)



Εικόνα 25: Διάγραμμα ροής του μενού επιλογών

Στην Εικόνα 26 παρουσιάζεται η πλοήγηση στο μενού με εικόνες από το πρωτότυπο μοντέλο που κατασκευάστηκε.



Εικόνα 26: Παρουσίαση της πλοήγησης στο μενού επιλογών

Κεφάλαιο 5. Πειραματική αξιολόγηση

Σε αυτό το κεφάλαιο ο σταθμός φόρτισης θα τεθεί σε λειτουργία και θα γίνει παρουσίαση και έλεγχος των επιμέρους λειτουργιών. Η εκτέλεση των διαδικασιών θα ακολουθεί την ροή που παρουσιάζεται από την Εικόνα 21 έως Εικόνα 25.

Η τοποθέτηση του σταθμού έγινε σε υπαίθριο χώρο, όπως φαίνεται στην Εικόνα 27 και χρησιμοποιώντας το απαραίτητο καλώδιο ρεύματος έγινε η εκκίνηση λειτουργίας, συνδέοντάς το στο σταθμό. Μετά την εκκίνηση του λειτουργικού συστήματος του Raspberry Pi ο σταθμός συνδέεται, είτε ασύρματα στο προεπιλεγμένο Access Point ή μέσω καλωδίου Ethernet, ώστε να υπάρχει σύνδεση στο internet και να μπορεί να γίνει η σύνδεση μέσω SSH για την παρακολούθηση και καταγραφή της λειτουργίας για τους σκοπούς της εργασίας.

Εικόνα 27: Τοποθέτηση σταθμού φόρτισης στο έδαφος

5.1 Αρχικοποίηση συστημάτων

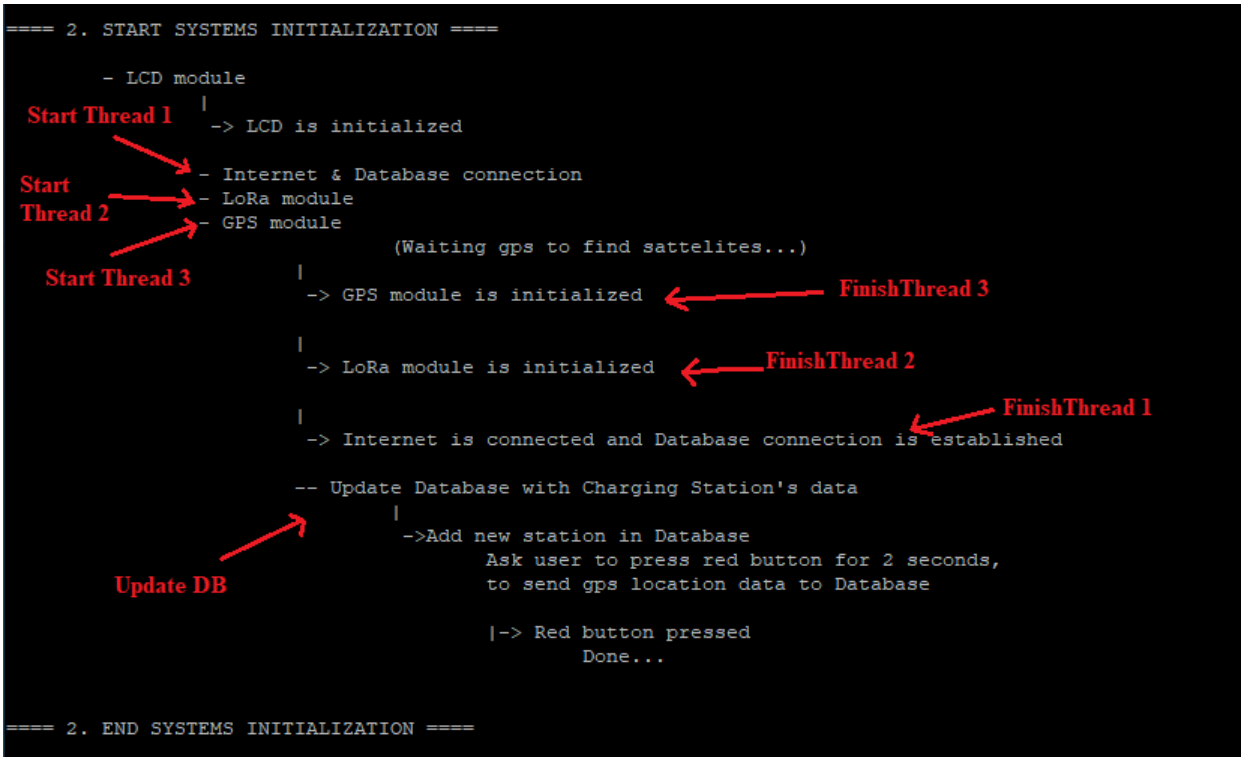
Εκτελώντας το script “main.py” ξεκινάει η αρχικοποίηση των συστημάτων του σταθμού. Σε αυτή τη λειτουργία περιλαμβάνεται αρχικά η ενεργοποίηση της οθόνης LCD, όπου θα φαίνονται κατάλληλα μηνύματα στον χρήστη. Αμέσως μετά γίνεται με χρήση threading η ενεργοποίηση της συσκευής επικοινωνίας με πρωτόκολλο LoRa στα 868Mhz, η εκκίνηση αναζήτησης δορυφόρων του δέκτη GPS και η σύνδεση με τον server που βρίσκεται η βάση δεδομένων του δικτύου των σταθμών φόρτισης. Στην Εικόνα 28 φαίνεται το CLI με τα μηνύματα των λειτουργιών, ενώ στην Εικόνα 29 τα αντίστοιχα μηνύματα στην οθόνη LCD που μπορεί να δει ο χρήστης. Έγινε προσπάθεια να παρουσιάζουν τη δομή του διαγράμματος ροής όπως στην Εικόνα 22.

Επίσης πρέπει να σημειωθεί πως σε κάθε εκκίνηση του σταθμού ελέγχεται αν ο σταθμός, ο οποίος έχει ένα μοναδικό κωδικό, υπάρχει ήδη στη βάση δεδομένων. Αν δεν υπάρχει, τότε καταχωρείται ως νέος

σταθμός και ο χρήστης, ακολουθώντας τις οδηγίες στην οθόνη LCD, ενημερώνει τη βάση δεδομένων για τη τοποθεσία.

Στην περίπτωση που ο σταθμός υπάρχει ήδη, τότε μπορεί ο χρήστης να τον μεταφέρει σε κάποιο άλλο μέρος, έτσι με κατάλληλα μηνύματα και μέσω του μενού επιλογών ο χρήστης καλείται να ενημερώσει τη βάση δεδομένων για τη νέα τοποθεσία.

Στο παράδειγμα που ακολουθεί φαίνονται οι λειτουργίες για την εγκατάσταση ενός νέου σταθμού.



Εικόνα 28: Διαδικασία αρχικοποίησης συστημάτων – αποτύπωση στο CLI

Εικόνα 29: Σειρά μηνυμάτων στην LCD οθόνη κατά την αρχικοποίηση συστημάτων

Μετά την δημιουργία του νέου σταθμού φόρτισης και την αρχικοποίηση των συστημάτων θα έχει ενημερωθεί η βάση δεδομένων με τις εγγραφές που φαίνονται στην Εικόνα 30. Εκεί παρουσιάζονται οι πίνακες που ενημερώθηκαν (βλ. Παράρτημα 2) μετά την παραπάνω διαδικασία.

charging_stations

STATION_ID	user_id	occupied	lora_address
60322001	12	0	2
NULL	NULL	NULL	NULL

users

user_id	user_name	user_email	user_tel	user_address	station_id
12	testuser	user@test.gr	69765xxxxx	Athens	60322001
NULL	NULL	NULL	NULL	NULL	NULL

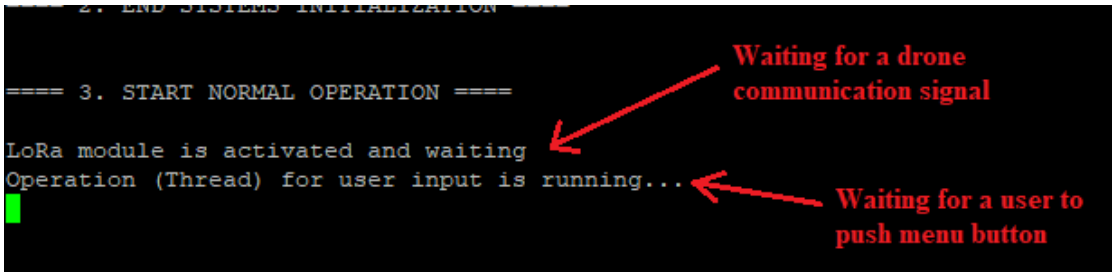
stations_location

STATION_ID	nmea_sentence_type	UTC	latitude	N_or_S	longitude	E_or_W	is_fixed	satellites_number	horizontal_dilution_of_precision	altitude_over_sea	units_of_altitude	geoidal_separation	units_of_geoidal	DGPS_data_age	DGPS_station_ID
60322001	$GPGGA	162840.11	38[redacted]07209	N	02[redacted]27768	E	1	06	1.71	189.9	M	33.2	M		*56

Εικόνα 30: Πίνακες στη βάση δεδομένων μετά τη δημιουργία του πρώτου σταθμού φόρτισης

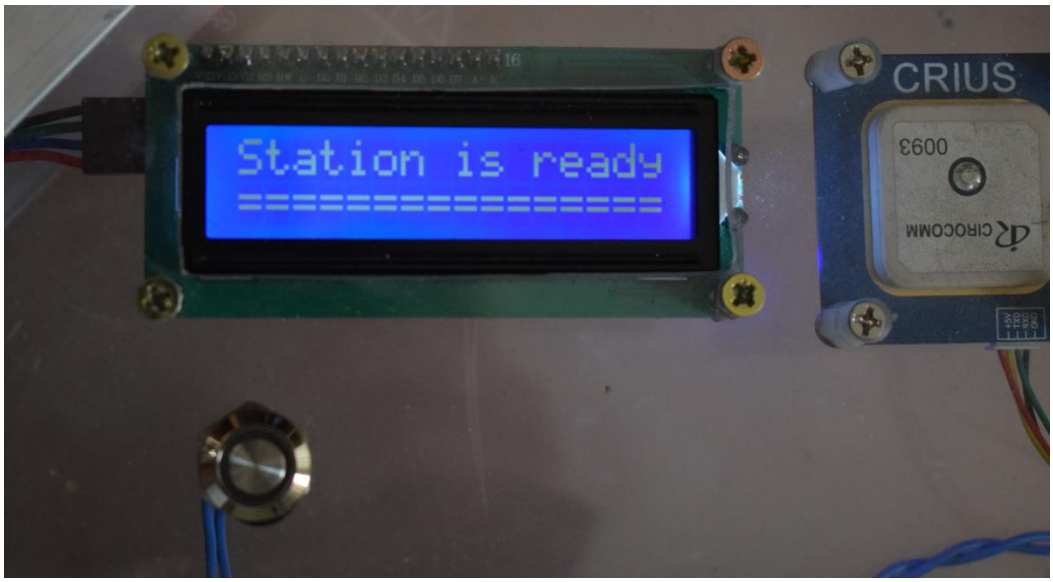
5.2 Κανονική λειτουργία σταθμού

Όταν η διαδικασία της αρχικοποίησης ολοκληρωθεί, θα εκτελεστούν δύο threads όπως φαίνεται στην Εικόνα 31, ένα για κάθε περίπτωση λειτουργίας (Εικόνα 23,Εικόνα 24), ενώ το πρόγραμμα θα μπει σε έναν ατέρμονα βρόγχο, περιμένοντας κάποια από τις δύο αυτές λειτουργίες να λάβει χώρα.



Εικόνα 31: Έναρξη κανονικής λειτουργίας του σταθμού – αποτύπωση στο CLI

Στην οθόνη LCD θα εμφανίζεται μόνιμα το μήνυμα επιβεβαίωσης πως ο σταθμός είναι έτοιμος για χρήση, όπως φαίνεται στην Εικόνα 32.



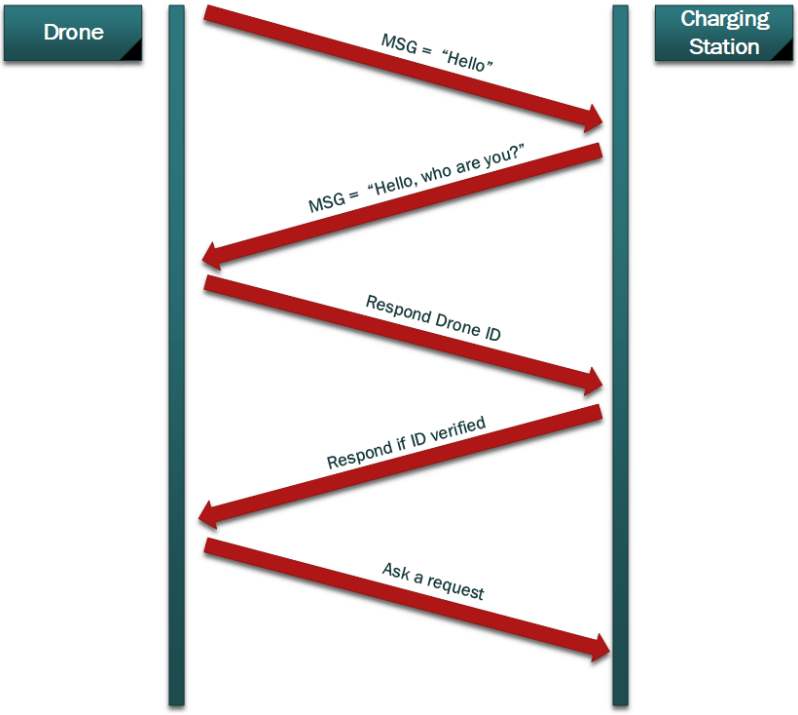
Εικόνα 32: Μήνυμα στην LCD οθόνη μετά και την αρχικοποίηση συστημάτων

5.3 Χρήση του σταθμού από κάποιο drone

Σε αυτό το σημείο πρέπει να σημειωθεί πως στη βάση δεδομένων του δικτύου των σταθμών φόρτισης, υπάρχουν εγγεγραμμένα κάποια drone τα οποία ανήκουν σε κάποιον χρήστη. Αυτά τα drone γνωρίζουν και ενημερώνονται περιοδικά για τις τοποθεσίες που βρίσκονται οι σταθμοί φόρτισης του δικτύου καθώς το σύστημα είναι δυναμικό και μπορεί να προστίθενται συνεχώς νέοι σταθμοί ή κάποιος χρήστης να αλλάξει την τοποθεσία του δικού του.

Στη περίπτωση που κάποιο drone χρειαστεί να προσεγγίσει τον σταθμό, θα πρέπει να επικοινωνήσει πρώτο, στέλνοντας ένα μήνυμα μέσω LoRa και στη συνέχεια να ανταλλάξουν μηνύματα με τον σταθμό ώστε να γίνει η ταυτοποίησή του, δηλαδή η αναζήτηση του drone στη βάση δεδομένων. Αν το drone είναι

εγγεγραμμένο στο δίκτυο των σταθμών μπορεί τελικά να αιτηθεί τον λόγο που προσέγγισε τον σταθμό. Η ροή αυτής της επικοινωνίας φαίνεται στην Εικόνα 33.



Εικόνα 33: Ροή επικοινωνίας με LoRa μεταξύ drone και σταθμού φόρτισης

Στη βάση δεδομένων έχουν γίνει ήδη δύο εγγραφές στο πίνακα με τα εγγεγραμμένα drone ώστε να υπάρχει κάποιο για ταυτοποίηση και δοκιμές όπως φαίνεται στην Εικόνα 34.

drone_Code	comments
HEXA123	A big hexacopter drone
QUAD001	The first drone in the fleet
NULL	NULL

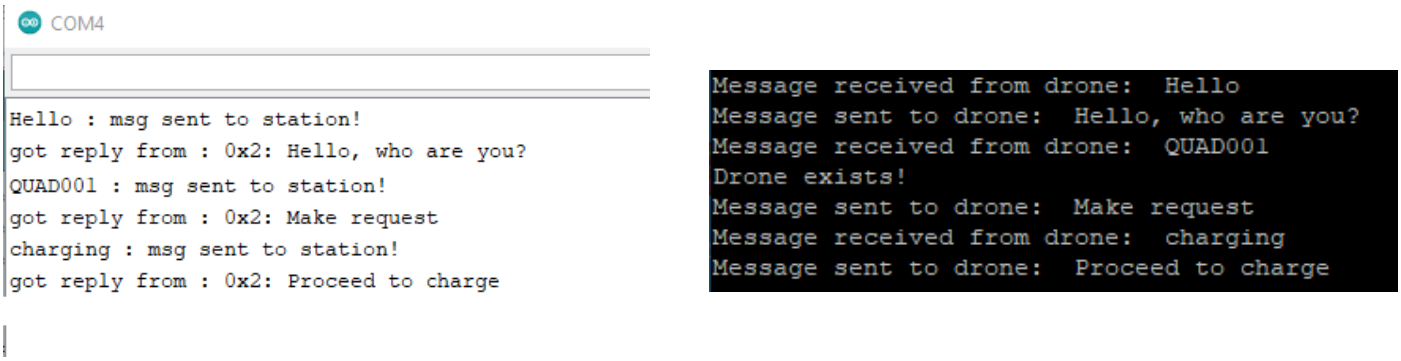
Εικόνα 34: Πίνακας εγγεγραμμένων drone στη βάση δεδομένων

Μετά το πέρας της παραπάνω επικοινωνίας και αν το drone είναι μέλος του δικτύου των σταθμών, μπορεί να αιτηθεί είτε την προσγείωσή του για φόρτιση, είτε να ζητήσει κάποιες πληροφορίες. Στην εργασία αυτή θα μελετήσουμε μόνο τη διαδικασία φόρτισης του drone.

5.4 Φόρτιση Drone

Για να ζητήσει ένα drone άδεια προσγείωσης για φόρτιση, αφού έγινε πρώτα η επιτυχής ταυτοποίησή του, στέλνει ειδικό μήνυμα στο σταθμό για το αίτημά του να φορτίσει. Για της ανάγκες της εργασίας αποστέλλεται το μήνυμα "charging", αλλά θα μπορούσε να ήταν κάποιος κωδικός ή κάποιο κρυπτογραφημένο μήνυμα σύμφωνα με το πρωτόκολλο επικοινωνίας που θα υλοποιηθεί μελλοντικά. Μόλις ο σταθμός λάβει το μήνυμα ελέγχει αν είναι ήδη κατειλημμένος από άλλο drone και δίνει άδεια για προσγείωση στο drone, με το μήνυμα "proceed".

Η διαδικασία που περιγράφηκε παραπάνω φαίνεται στην Εικόνα 35, μέσω του CLI του raspberry. Επίσης πρέπει να αναφερθεί πως για τις ανάγκες παρουσίασης της εργασίας, ο σταθμός επικοινωνούσε με ένα arduino στο οποίο είχε τοποθετηθεί ένα LoRa shield και προσομοίωνε τα μηνύματα που θα στελνόντουσαν από τον flight controller ενός drone .



Εικόνα 35: Επικοινωνία του drone με τον σταθμός φόρτισης

Στη συνέχεια το drone προσεγγίζει το σταθμό και προσγειώνεται, όπως φαίνεται στην Εικόνα 36.*

Εικόνα 36: Προσγείωση του drone στο σταθμό φόρτισης

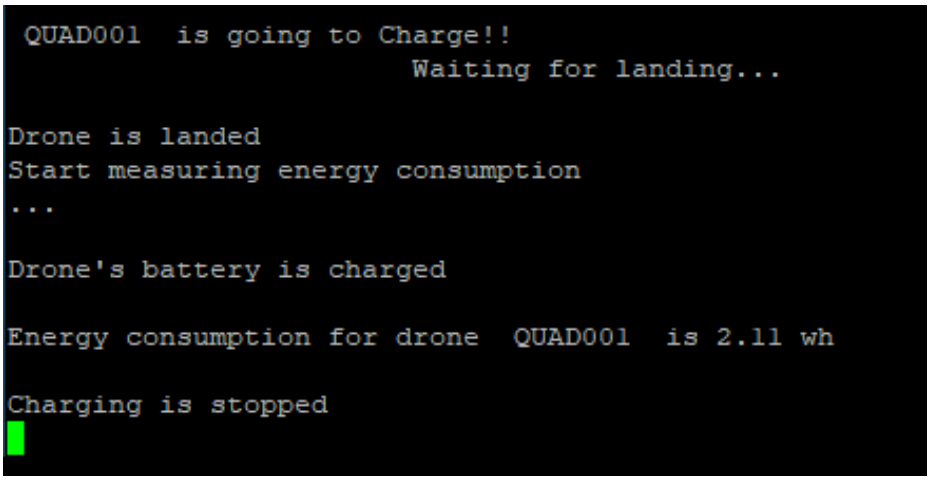
Ο σταθμός ανιχνεύει την σωστή θέση του drone με τη βοήθεια των αισθητήρων πίεσης και ξεκινάει τη παροχή ρεύματος στο drone μέσω των αγώγιμων επαφών. Κατά τη διάρκεια της φόρτισης γίνεται και η καταγραφή της ενέργειας που παρέχεται στο drone.

Όταν το drone ολοκληρώσει τη φόρτιση, στέλνει ένα μήνυμα στο σταθμό για να απενεργοποιήσει την παροχή ενέργειας. Το μήνυμα που αποστέλλεται για τις ανάγκες της εργασίας είναι το “finished”. Ο σταθμός

* Η διαδικασία προσγείωσης βαραίνει τον χρήστη του drone ο οποίος πρέπει να έχει προβλέψει τους μηχανισμούς ανίχνευσης που θα είναι εγκαταστημένοι στο drone, σύμφωνα με τα χαρακτηριστικά που ο πάροχος του σταθμού έχει ορίσει.

λαμβάνει αυτό το μήνυμα, διακόπτει τη παροχή ενέργειας και περιμένει να απογειωθεί το drone ώστε να ενημερώσει τη βάση δεδομένων με τα στοιχεία της φόρτισης.

Στην Εικόνα 37 παρουσιάζεται η καταγραφή της διαδικασίας φόρτισης από τη προσγείωση του drone μέχρι και την απογείωση, ενώ στην Εικόνα 38 φαίνεται η εγγραφή στη βάση δεδομένων, με τα στοιχεία της φόρτισης του drone, όπου αναφέρεται μεταξύ των άλλων και η ενέργεια, σε wh, που χρησιμοποιήθηκε για τη φόρτιση. Οι εγγραφές που φαίνονται στο πίνακα είναι και από προηγούμενες δοκιμές, ενώ η δοκιμή στην Εικόνα 37 είναι στη τελευταία εγγραφή του πίνακα με event_ID 21.

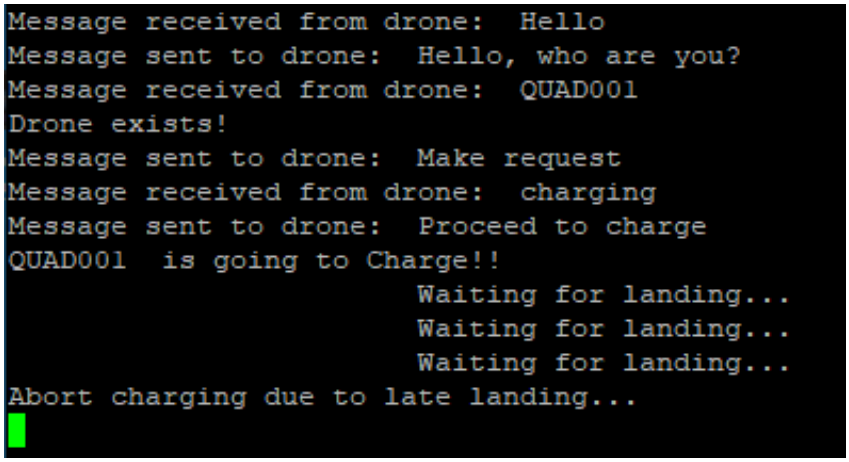


Εικόνα 37: Κύκλος φόρτισης του drone (Προσγείωση - Απογείωση)

event_ID	station_ID	drone_Code	starting_time	stoping_time	landing_date	energy_consumption
16	60322001	HEXA123	22:08:54	22:10:33	2022-05-10	5.10842
17	60322001	QUAD001	20:54:06	20:55:22	2022-05-16	4.06018
18	60322001	QUAD001	22:26:33	22:27:21	2022-05-16	0.79201
19	60322001	QUAD001	22:29:13	22:29:34	2022-05-16	0.00500
20	60322001	QUAD001	22:31:38	22:32:05	2022-05-16	0.21145
21	60322001	QUAD001	22:44:27	22:45:32	2022-05-16	2.11327
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Εικόνα 38: Ενημέρωση της Βάσης Δεδομένων μετά τη φόρτιση του drone

Σε μια περίπτωση που το drone κατά τη διάρκεια της προσγείωσης περάσει ένα καθορισμένο χρόνο που ο σταθμός περιμένει για την ολοκλήρωση της διαδικασίας, τότε αναβάλλεται η προσγείωση και πρέπει το drone να επικοινωνήσει εκ νέου αν ακόμα επιθυμεί να προσγειωθεί για φόρτιση. Στην Εικόνα 39 φαίνεται η διαδικασία που αναφέρθηκε στο CLI του raspberry.



Εικόνα 39: Αναβολή διαδικασίας προσγείωσης λόγο καθυστέρησης – time out

5.5 Περίπτωση χρήσης του σταθμού από τον χρήστη

Κατά τη διάρκεια λειτουργίας του σταθμού μπορεί ο χρήστης να χρησιμοποιήσει το μενού επιλογών που διαθέτει ο σταθμός οποιαδήποτε στιγμή, χωρίς να επηρεάζει τις υπόλοιπες λειτουργίες.

Μπορεί μέσω του μενού να μάθει διάφορες πληροφορίες όπως, το μοναδικό κωδικό του σταθμού, τις συντεταγμένες που είναι τοποθετημένος, το υψόμετρο που βρίσκεται, κ.α. Επίσης έχει τη δυνατότητα από τις ρυθμίσεις να ενημερώσει τη βάση δεδομένων για νέες συντεταγμένες, εάν μετέφερε το σταθμό σε άλλο σημείο. Στη παράγραφο 4.3 παρουσιάζονται οι λεπτομέρειες σχετικά με το μενού.

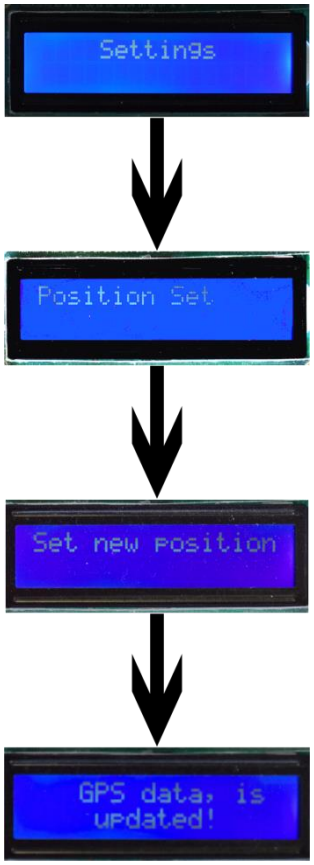
Σε αυτή τη παράγραφο θα παρουσιαστεί με εικόνες, η κάθε μια από τις επιλογές του μενού και τα αποτελέσματα που θα πάρει ο χρήστης για κάθε επιλογή. Θα γίνει προσπάθεια να παρουσιαστεί μια ρεαλιστική περίπτωση χρήσης για τις επιλογές ώστε να φανεί η ανάγκη ύπαρξης του μενού επιλογών στον σταθμό φόρτισης.

5.5.1 Περίπτωση 1: “Settings → Position Settings → Set new position”

Όπως αναφέρθηκε σε προηγούμενες παραγράφους, κάποιος χρήστης μπορεί να θέλει να μετακινήσει το σταθμό φόρτισης από την αρχική του τοποθεσία, σε κάποια άλλη. Όταν γίνει η μεταφορά και θέσει ο χρήστης σε λειτουργία τον σταθμό, θα ξεκινήσει η αρχικοποίηση των συστημάτων όπως προβλέπεται. Σε αυτή τη διαδικασία θα ενημερωθεί από την οθόνη LCD πως ο σταθμός υπάρχει ήδη στη βάση δεδομένων όπως είναι αναμενόμενο. Επίσης θα ενημερωθεί πως αν έχει αλλάξει θέση το σταθμό πρέπει να πάει στις ρυθμίσεις για την ενημέρωση της νέας τοποθεσίας. Στην Εικόνα 40 παρουσιάζεται η διαδικασία που περιγράφηκε σε αντιστοιχία με την Εικόνα 29 όπου είχε γίνει τοποθέτηση νέου σταθμού.

Εικόνα 40: Τοποθέτηση και αρχικοποίηση, ήδη εγγεγραμμένου σταθμού στο δίκτυο

Ο χρήστης λοιπόν ακολουθεί το μενού όπως φαίνεται στην Εικόνα 41 και επιλέγει να ενημερώσει τη βάση δεδομένων για τις νέες συντεταγμένες του σταθμού φόρτισης.



Εικόνα 41: Μενού επιλογών - Set new position

Στην Εικόνα 42 φαίνεται η προηγούμενη και η νέα ενημερωμένη τοποθεσία του σταθμού μετά την ολοκλήρωση της διαδικασίας.

OLD POSITION

STATION_ID	nmea_sentence_type	UTC	latitude	N_or_S	longitude	E_or_W	is_fixed	satellites_number	horizontal_dilution_of_precision	altitude_over_sea	units_of_altitude
60322001	$GPGGA	162840.11	3[redacted]2.07209	N	02[redacted]5.27768	E	1	06	1.71	189.9	M

NEW POSITION

STATION_ID	nmea_sentence_type	UTC	latitude	N_or_S	longitude	E_or_W	is_fixed	satellites_number	horizontal_dilution_of_precision	altitude_over_sea	units_of_altitude
60322001	$GPGGA	162803.89	3[redacted]0.11891	N	02[redacted]0.58044	E	1	05	3.79	145.6	M

Εικόνα 42: Συντεταγμένες σταθμού φόρτισης, πριν και μετά την αλλαγή τοποθεσίας

5.5.2 Περίπτωση 2: "Information → Show user ID"

Σε αυτή και στις επόμενες περιπτώσεις χρήσης θεωρούμε πως υπάρχει κάποιου είδους βλάβη στα συστήματα του σταθμού. Σε μια τέτοια περίπτωση ο χρήστης καλείται να επικοινωνήσει με τον εξουσιοδοτημένο τεχνικό για την αναφορά και επίλυση της βλάβης.

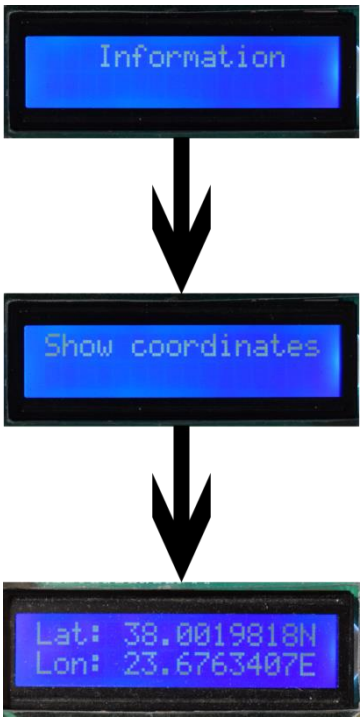
Ο τεχνικός για να μπορέσει να συνδεθεί απομακρυσμένα στο σταθμό θα ζητήσει από τον χρήστη να του πει το μοναδικό κωδικό του σταθμού. Έτσι ο χρήστης θα πρέπει να ακολουθήσει το μενού στις πληροφορίες και να επιλέξει "Show user ID", όπως παρουσιάζεται στην Εικόνα 43. Ο κωδικός που εμφανίζεται στην οθόνη είναι ο μοναδικός κωδικός που έχει καταχωρηθεί ο συγκεκριμένος σταθμός φόρτισης στη βάση δεδομένων.

Εικόνα 43: Μενού επιλογών – Show user ID

5.5.3 Περίπτωση 3: “Information → Show Coordinates” και “Information → Show Altitude”

Αντίστοιχα και σε αυτή τη περίπτωση μπορεί ο χρήστης να χρειάζεται να ενημερώσει κάποιον τεχνικό, που δεν έχει απομακρυσμένη πρόσβαση στο σταθμό, για τη τοποθεσία και το υψόμετρο που είναι τοποθετημένος ο σταθμός. Με αυτά τα δεδομένα ο τεχνικός θα μπορούσε να επισκεφτεί το σταθμό για να τον επαναφέρει σε λειτουργική κατάσταση.

Έτσι ο χρήστης θα πρέπει να πλοηγηθεί στο μενού και στις πληροφορίες να επιλέξει “Show coordinates” καθώς και “Show altitude” και να ενημερώσει τον τεχνικό με τις πληροφορίες που εμφανιστήκαν. Τα αποτελέσματα των ενεργειών αυτών φαίνονται στην Εικόνα 44 και Εικόνα 45.



Εικόνα 44: Μενού επιλογών – Show coordinates

Για την σωστή αποτύπωση των συντεταγμένων της τοποθεσίας του σταθμού, θα πρέπει να υπολογιστούν σωστά οι τιμές από τα δεδομένα που ελήφθησαν από το GPS. Τα δεδομένα της τοποθεσίας είναι της μορφής ddmm.mmmm για το latitude και dddmm.mmmm για το longitude. Για να υπολογιστούν οι τιμές που θα βάζαμε σε έναν ηλεκτρονικό χάρτη ώστε να δώσει τη θέση μας θα πρέπει να ακολουθήσουμε τις παρακάτω σχέσεις.

$$dd + mm.mmmm/60 \ \textit{για latitude}$$

$$ddd + mm.mmmm/60 \ \textit{για longitute}$$

Εικόνα 45: Μενού επιλογών – Show altitude

Αυτές ήταν οι επιλογές που έχουν καταχωρηθεί στο μενού για τις ανάγκες της εργασίας. Φυσικά είναι ενδεικτικές και μπορούν εύκολα να προστεθούν περισσότερες όπως ήδη έχει αναφερθεί.

Κεφάλαιο 6. Τελικές παρατηρήσεις και μελλοντικές ιδέες

Στόχος της παρούσας διπλωματικής εργασίας ήταν η δημιουργία ενός αυτόματου σταθμού φόρτισης για drone με σκοπό την εύκολη φόρτιση των μπαταριών του, χωρίς να είναι απαραίτητη η ανθρώπινη παρέμβαση. Επιπλέον, υπήρχε η ανάγκη να ενσωματωθούν λειτουργίες επικοινωνίας και διασύνδεσης με κάποια Βάση Δεδομένων καθώς και δημιουργία αυτοματισμών για την ανίχνευση και εξυπηρέτηση του drone. Λόγω της πολυπλοκότητας του θέματος, είναι προφανές πως τα επιμέρους τμήματα της εργασίας δεν είναι ολοκληρωμένα, παρά όσο είναι απαραίτητο για το proof of concept.

Για παράδειγμα, η επικοινωνία του σταθμού με το drone προσομοιώθηκε με χρήση ενός arduino module αντί για ένα κανονικό drone (βλ. Παράρτημα 3) και τα μηνύματα που ανταλλάχτηκαν δεν ήταν κρυπτογραφημένα. Έτσι, καθίσταται η επικοινωνία μη ασφαλή, σε μεγάλο βαθμό.

Το σύστημα ανίχνευσης του drone με τους αισθητήρες πίεσης, καθώς και η φόρτιση, δούλεψαν καλά και χωρίς προβλήματα, τροφοδοτώντας το drone με ρεύμα, μόνο αφού είχε γίνει σωστά η προσγείωση. Ενώ, η μέτρηση της καταναλισκόμενης ενέργειας κατά τη φόρτιση δεν υπολογίζεται με ακρίβεια καθώς είναι εγκατεστημένος μόνο αισθητήρας ανίχνευσης ρεύματος και η τιμή της τάσης ενσωματώνεται ως σταθερά, στον Πηγαίο Κώδικα. Αυτό όμως διορθώνεται εύκολα με μελλοντική αναβάθμιση του hardware προσθέτοντας τον κατάλληλο αισθητήρα, ώστε ρεύμα και τάση να υπολογίζονται ταυτόχρονα κάθε δεδομένη στιγμή με αποτέλεσμα την ακριβή μέτρηση της ενέργειας.

Σχετικά με τα μέρη της εργασίας που δεν σχετίζονται με την αυτόματη φόρτιση, πρέπει να αναφέρω την επιτυχημένη επικοινωνία του σταθμού με Βάση Δεδομένων (ΒΔ) σε απομακρυσμένο server και την επιτυχή καταχώρηση και ανάγνωση δεδομένων σε πραγματικό χρόνο. Ο σταθμός κατά την πρώτη λειτουργία του στέλνει στη ΒΔ τα στοιχεία της τοποθεσίας του με χρήση του GPS, για την ορθή καταχώρησή του στο δίκτυο των σταθμών.

Κατά τη διάρκεια της λειτουργίας του ενημερώνει για το ιστορικό φόρτισης των drone, με τα στοιχεία του εκάστοτε drone, τα στοιχεία που αφορούν τους χρόνους προσγείωσης και απογείωσης, καθώς και τη ενέργεια που χρησιμοποιήθηκε κατά τη φόρτιση.

Τέλος, το μενού επιλογών φαίνεται πολύ χρήσιμο και εύκολο στη χρήση και σίγουρα χρειάζεται να προστεθούν μελλοντικά επιπλέον επιλογές. Φυσικά οι επιλογές που έχουν συμπεριληφθεί είναι απλά για τη παρουσίαση της εργασίας, όμως η επιλογή για την αλλαγή τοποθεσίας του σταθμού είναι σημαντική καθώς έχουμε να κάνουμε με έναν φορητό σταθμό, πράγμα που απαιτεί μια λειτουργία σαν και αυτή, ώστε να αποστέλλονται οι νέες συντεταγμένες προς τη ΒΔ και να ενημερώνεται το δίκτυο για την αλλαγή που έγινε.

Το βάρος του σταθμού είναι χαμηλό και ο τρόπος αναδίπλωσης και οι χειρολαβές κάνουν τη μεταφορά ευκολότερη. Δυστυχώς ο τρόπος κατασκευής του πρωτότυπου μοντέλου δεν αποδίδει σωστά τα πλεονεκτήματα της σχεδίασης που έχει το τρισδιάστατο μοντέλο (Εικόνα 6 – Εικόνα 12). Ο σταθμός είναι σχεδιασμένος για να είναι αδιάβροχος και υποτίθεται ότι οι αγώγιμες πλάκες και τα αισθητήρια πίεσης θα είναι κατασκευασμένα ως ένα ενιαίο κομμάτι.

Επίσης, η κεραία που χρησιμοποίησα για την επικοινωνία (Εικόνα 13) δεν είναι σωστή και δεν ακολουθεί τη διαρρύθμιση του τρισδιάστατου μοντέλου, αλλά κατά τη διάρκεια των δοκιμών υπήρξε ένα πρόβλημα και δεν κατάφερα να βρω τα απαραίτητα υλικά για να το διορθώσω. Έτσι, έκανα τις δοκιμές με αυτή τη κεραία ώστε να προχωρήσει η ροή της εργασίας.

Αυτή η εργασία αποτελεί πρακτικά περίληψη των διαφορετικών θεμάτων που προκύπτουν από το εν λόγω θέμα. Κάθε διαφορετικό κομμάτι θα μπορούσε να μελετηθεί αναλυτικότερα ως διπλωματική εργασία.

Με τη σκέψη αυτή, θα προχωρήσω σε μία λίστα με τα μελλοντικά θέματα που μπορούν να λάβουν χώρα στη μελέτη και την ανάπτυξη του δικτύου των σταθμών φόρτισης των drone:

- Κατασκευή drone με τον απαραίτητο εξοπλισμό για την αυτόνομη προσγείωση στο σταθμό φόρτισης και ενσωμάτωση συστήματος επικοινωνίας LoRa και σχετικών αλγορίθμων.
- Αναβάθμιση του Πηγαίου Κώδικα του σταθμού για τη δημιουργία fail safe ρουτινών καθώς και τη διαχείριση των σφαλμάτων κατά τη λειτουργία του σταθμού.
- Ενσωμάτωση αναδιπλούμενου συστήματος ανανεώσιμων πηγών ενέργειας για χρήση του σταθμού σε μέρη χωρίς παροχή ενέργειας από το δίκτυο ηλεκτροδότησης και εγκατάσταση μπαταριών για αποθήκευση της ενέργειας, στο δεξί μέρος του σταθμού το οποίο είναι κενό για τέτοιου είδους χρήσεις.
- Ρουτίνες επικοινωνίας μεταξύ σταθμών για την περίπτωση απώλειας σύνδεσης στο internet σε έναν από αυτούς, με σκοπό την μεταφορά πληροφοριών από σταθμό σε σταθμό μέχρι να φτάσει η πληροφορία στη ΒΔ.
- Μελέτη και δημιουργία ιδιωτικού LoRa WAN και ολοκληρωμένου πρωτόκολλου επικοινωνίας με όλα τα μέλη του δικτύου.
- Δημιουργία εφαρμογής για την ενημέρωση των χρηστών, για τις θέσεις των σταθμών και του drone τους, καθώς και πλατφόρμα διαχείρισης και εγγραφής χρηστών στο δίκτυο.
- Ενσωμάτωση αλγορίθμου χρέωσης της φόρτισης του drone και διαχείριση του δικτύου μέσω συστήματος blockchain για ασφάλεια επικοινωνιών και συναλλαγών μεταξύ των κόμβων.

Κεφάλαιο 7. Παραρτήματα

Στα επόμενα παραρτήματα θα παρουσιαστούν κάποια βασικά δομικά στοιχεία της εργασίας, που χωρίς αυτά δεν θα μπορούσε να λειτουργήσει ο σταθμός φόρτισης.

Στο παράρτημα 1 θα επισυνάψω το περιεχόμενο των αρχείων με τον Πηγαίο Κώδικα που εκτελείται κατά τη διάρκεια λειτουργίας του σταθμού. Στο παράρτημα 2 θα γίνει μια παρουσίαση του Σχεσιακού Μοντέλου Οντοτήτων της Βάσης Δεδομένων και στο παράρτημα 3 θα δείξω το υλικό (hardware) που χρησιμοποίησα για τη προσομοίωση της επικοινωνίας του σταθμού με το drone.

7.1 Παράρτημα 1 – Πηγαίος Κώδικας

Ο Πηγαίος Κώδικας της εργασίας αποτελείται από 6 αρχεία σε γλώσσα python όπως περιγράφονται στη συνέχεια:

1:	ChargingStation.py	→ Αρχείο με τη βασική κλάση του σταθμού φόρτισης και της μεθόδους λειτουργίας του
2:	init_methods.py	→ Αρχείο με τις μεθόδους αρχικοποίησης των συστημάτων του σταθμού
3:	drone_operations.py	→ Αρχείο με τις μεθόδους που εκτελούνται για την εξυπηρέτηση ενός drone
4:	wait_for_user_operation.py	→ Αρχείο με τις μεθόδους που εκτελούνται κατά τη χρήση του μενού επιλογών από τον χρήστη
5:	_constants.py	→ Αρχείο ρυθμίσεων με βασικές σταθερές που χρησιμοποιούνται στα υπόλοιπα αρχεία
6:	main.py	→ Αρχείο εκτέλεσης του προγράμματος

Επίσης περιλαμβάνονται και 2 αρχεία κειμένου με κάποιες απαραίτητες πληροφορίες:

1:	database_info.txt	→ Αρχείο με τα στοιχεία σύνδεσης στη βάση δεδομένων
2:	clients_info.txt	→ Αρχείο με τα στοιχεία του χρήστη που θα έχει στη κατοχή του τον σταθμό. Περιέχεται επίσης και ο μοναδικός αριθμός του σταθμού

Σε πραγματικές συνθήκες χρήσης αυτά τα αρχεία χρειάζεται να είναι κρυπτογραφημένα για λόγους ασφάλειας.

- ChargingStation.py

```python
#import libraries and modules
import time
from _constants import *
import pynmea2
import serial
from threading import Thread
import mysql.connector
from mysql.connector import errorcode
import RPi.GPIO as GPIO
from ina219 import INA219, DeviceRangeError

#import from other python scripts
from _constants import *
import init_methods as init
import drone_operations as op1
import wait_for_user_operation as op2

class ChargingStation(object):

#initialize the charging station with info from a config file
    def __init__(self,station_id,name="NULL",
                      client_email="NULL",
                      client_tel=0,
                      client_address="NULL",
                      lora_address = LORA_ADDRESS,
                      SHUNT_OHMS = INO219_SHUNT_OHMS,
                      MAX_EXPECTED_AMPS = INO219_MAX_EXPECTED_AMPS):
        self.__station_id = station_id
        self.userName = name
        self.client_email = client_email
        self.client_tel = client_tel
        self.client_address = client_address
        self.lora_address = int(lora_address)
        self.__lcd = None
        self.__gps = None
        self.__lora = None
        self.__db_con = None
        self.__LORA_PAYLOAD = None
        self.__ina219 = INA219(SHUNT_OHMS, MAX_EXPECTED_AMPS)
        self.__ina219.configure(self.__ina219.RANGE_16V, self.__ina219.GAIN_AUTO)
        self.__charging_flag = False
        self.threading_list = []
        self.communication_flag=False

```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
45
46      def get_DB_config(self):
47          try:
48              f = open('database_info.txt','r')
49              config = {}
50
51              for row_data in f:
52                  data = row_data.split()
53                  data = data[0].split(":")
54                  config[data[0]] = data[1]
55              return config
56          except Exception as e:
57              print(e)
58              return None
59
60  #connect to database for operations
61      def connect_DB(self):
62          try:
63              config = self.get_DB_config()
64              self.connection = mysql.connector.connect(**config)
65              #print("Connected to DB!")
66              time.sleep(1)
67              self.cursor = self.connection.cursor()
68              return self.connection,self.cursor
69          except mysql.connector.Error as err:
70              if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
71                  print("Something is wrong with your user name or password")
72                  return None
73              elif err.errno == errorcode.ER_BAD_DB_ERROR:
74                  print("Database does not exist")
75                  return None
76              else:
77                  print(err)
78                  return None
79
80  #add a new charging station if not exists
81      def add_new_station(self):
82          query1 = """INSERT INTO charging_stations (STATION_ID,occupied,lora_address)
83                  VALUES ('{}','{}');""".format(self.__station_id, 0, self.lora_address)
84
85          query2 = """INSERT INTO users (user_name,user_email,user_tel,user_address,station_id)
86                  VALUES ('{}','{}','{}','{}','{}');""".format(self.userName,
87                                                               self.client_email,
88                                                               self.client_tel,
89                                                               self.client_address,
90                                                               self.__station_id)
91
92          result1 = self.DB_insert_update_query(query1)
93          result2 = self.DB_insert_update_query(query2)
94          return not result1, not result2
95
```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
96 #database query for insert and update operations
97     def DB_insert_update_query(self,query):
98         try:
99             connection,cursor = self.connect_DB()
100            cursor.execute(query)
101            connection.commit()
102            cursor.close()
103            connection.close()
104            return True
105
106        except mysql.connector.Error as err:
107            if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
108                print("Something is wrong with your user name or password")
109                return False
110            elif err.errno == errorcode.ER_BAD_DB_ERROR:
111                print("Database does not exist")
112                return False
113            else:
114                print(err)
115                return False
116
117    def DB_select_query(self,query):
118        try:
119            connection,cursor = self.connect_DB()
120            cursor.execute(query)
121            myresult = cursor.fetchall()
122            cursor.close()
123            connection.close()
124            if myresult:
125                return myresult[0]
126            else:
127                return False
128        except mysql.connector.Error as err:
129            if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
130                print("Something is wrong with your user name or password")
131                return False
132            elif err.errno == errorcode.ER_BAD_DB_ERROR:
133                print("Database does not exist")
134                return False
135            else:
136                print(err)
137                return False
138
```

```
#get data from gps if fixed and return dictionary with all GPGGA sentence's parts
    def get_GPS_data(self,ser):
        ser.open()
        #print("searching for gps data!")
        gps_dict = None
        #ser = self.serial_init()
        for _ in range(5):
            time.sleep(0.1)
            if ser.isOpen():
                #print("port is open")
                try:
                    data = ser.readline()
                    data = data.decode("utf-8")
                    #print(type(data))
                    if data[0:6]=="$GPGGA":
                        #print("GPGGA message found")
                        data_splitted = data.split(",")
                        if not data_splitted[6] == '0':
                            #print("GPS is fixed")
                            ser.close()
                            gps_dict = {"sentence_ID":data_splitted[0],
                            "UTC_time":data_splitted[1],
                            "latitude":data_splitted[2],
                            "N_or_S":data_splitted[3],
                            "longitude":data_splitted[4],
                            "E_or_W":data_splitted[5],
                            "is_fixed":data_splitted[6],
                            "satellites_number":data_splitted[7],
                            "horizontal_dilution_of_precision":data_splitted[8],
                            "altitude_over_sea":data_splitted[9],
                            "units_of_altitude":data_splitted[10],
                            "geoidal_separation":data_splitted[11],
                            "units_of_geoidal":data_splitted[12],
                            "DGPS_data_age":data_splitted[13],
                            "DGPS_station_ID":data_splitted[14]}
                            #print(gps_dict)
                            ser.close()
                            return gps_dict
                        else:
                            fixed = False
                            time.sleep(3.1)
                            continue
                except Exception as e:
                    #print("Error in GPS module: ",e)
                    continue
        #print("port closed")
        ser.close()
        return None

```

```
#send gps location data to database
    def gps_data_to_DB(self):
        try:
            data = self.get_GPS_data(self.__gps)
            if data != None:
                query = """UPDATE stations_location
                        SET nmea_sentence_type = '{}',
                            UTC = '{}',latitude = '{}',
                            N_or_S = '{}',longitude = '{}',
                            E_or_W = '{}',is_fixed = '{}',
                            satellites_number = '{}',
                            horizontal_dilution_of_precision = '{}',
                            altitude_over_sea = '{}',
                            units_of_altitude = '{}',
                            geoidal_separation = '{}',
                            units_of_geoidal = '{}',
                            DGPS_data_age = '{}',
                            DGPS_station_ID = '{}'
                        WHERE STATION_ID = '{}' ;""".format(data["sentence_ID"],
                                                            data["UTC_time"],
                                                            data["latitude"],
                                                            data["N_or_S"],
                                                            data["longitude"],
                                                            data["E_or_W"],
                                                            data["is_fixed"],
                                                            data["satellites_number"],
                                                            data["horizontal_dilution_of_precision"],
                                                            data["altitude_over_sea"],
                                                            data["units_of_altitude"],
                                                            data["geoidal_separation"],
                                                            data["units_of_geoidal"],
                                                            data["DGPS_data_age"],
                                                            data["DGPS_station_ID"],
                                                            self.__station_id)
                result = self.DB_insert_update_query(query)
                return result
            return False
        except mysql.connector.Error as err:
            if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
                print("Something is wrong with your user name or password")
                return False
            elif err.errno == errorcode.ER_BAD_DB_ERROR:
                print("Database does not exist")
                return False
            else:
                print(err)
                return False

    def get_station_ID(self):
        return self.__station_id

    def get_station_coordinates(self):
        query = """SELECT latitude,N_or_S,longitude,E_or_W
                    FROM stations_location
                    WHERE STATION_ID = '{}';""".format(self.__station_id)
        result = self.DB_select_query(query)
        lat = "Lat: {:.7f}{}".format((float(result[0][:2]) + float(result[0][2:])/60),result[1])
        lon = "Lon: {:.7f}{}".format((float(result[2][:3]) + float(result[2][3:])/60), result[3])

        return lat,lon

```

```
    def get_station_altitude(self):
        query = """SELECT altitude_over_sea
                    FROM stations_location
                    WHERE STATION_ID = '{}';""".format(self.__station_id)
        result = self.DB_select_query(query)
        alt = result[0]

        return alt

#init_systems with threads
    def init_systems(self):

        #initialize lcd display firstly
        init.LCD_init()

        time.sleep(1)

        dbcon_thread = Thread(target=init.database_init, args=(self,))  # define thread for db_init
        lora_thread = Thread(target=init.lora_init, args =(self,self.lora_address,)) # define thread for lora_init
        gps_thread = Thread(target=init.serial_init, args=(self,))      # define thread for serial_init

        dbcon_thread.start()   # Start threads
        lora_thread.start()    # Start threads
        gps_thread.start()     # Start threads

        dbcon_thread.join()    # Wait for every thread to finish its job
        lora_thread.join()     # Wait for every thread to finish its job
        gps_thread.join()      # Wait for every thread to finish its job

        self.__lcd = init.my_LCD.get()     # Get for lcd the returned value through queue
        self.__gps = init.my_GPS.get()     # Get for gps the returned value through queue
        self.__lora = init.my_LoRa.get()   # Get for lora the returned value through queue
        self.__lora.on_recv = self.lora_on_recv
        self.__db_con = init.my_DB.get()   # Get for db the returned value through queue

    def get_lcd(self):
        return self.__lcd
    def get_gps(self):
        return self.__gps
    def get_lora(self):
        return self.__lora
    def get_db_con(self):
        return self.__db_con

    def get_ina219(self):
        return self.__ina219

    def set_charging_flag(self,option):
        if option == True or option == False:
            self.__charging_flag = option
        else:
            print("Argument on set_charging_flag have to be True or False...")

    def get_charging_flag(self):
        return self.__charging_flag

    def get_lora_payload(self):
        return self.__LORA_PAYLOAD

    def reset_lora_payload():
        self.__LORA_PAYLOAD = None

    def lora_send_msg(self,msg,to_address):
        while not self.get_lora().send_to_wait(msg, to_address, retries=1):
            time.sleep(0.5)
        print("Message sent to drone: ",msg)

```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
316     def catch_lora_payload(self,packet):
317         self.__LORA_PAYLOAD = packet
318 
319     def lora_on_recv(self,payload):
320         try:
321             #print("Just received a message")
322             self.catch_lora_payload(payload)
323             if not self.communication_flag:
324                 self.communication_flag = True
325                 new_thread = Thread(target=self.communication_procedure)
326                 new_thread.start()
327                 self.threading_list.append(new_thread)
328 
329         except Exception as e:
330             print("lora_on_recv ERROR: ", e)
331 
332     def set_lora_mode_cad(self):
333         print("LoRa module is activated and waiting")
334         self.get_lora().set_mode_cad()
335 
336     def communication_procedure(self):
337         status = 0 #in status 0 there is no open communication
338         drone_id = None #define drone id as None to use later
339         while True:
340             if self.__LORA_PAYLOAD is not None: #if a lora message received
341                 packet = self.__LORA_PAYLOAD
342                 msg = packet.message.decode("utf-8").strip('\x00')
343 
344                 print("Message received from drone: ", msg)
345 
346                 if status==0 and msg == HELLO_MSG_FROM_DRONE: #Handshake msg from drone to station
347                     respond_msg = HELLO_MSG_FROM_CS
348                     self.lora_send_msg(respond_msg,packet.header_from)
349                     status = 1 #handshake is done
350                     self.__LORA_PAYLOAD = None
351                 elif status == 1: #in status 1 there is a correct handshake and the msg from drone is its code for
    validation
352                     db_query = """SELECT * FROM registered_drones WHERE drone_Code = '{}';""".format(msg)
353                     if self.DB_select_query(db_query):
354                         print("Drone exists!")
355                         drone_id = msg #catch drone id
356                         respond_msg = ASK_REQ_FROM_DRONE
357                         self.lora_send_msg(respond_msg,packet.header_from)
358                         status = 2
359                         self.__LORA_PAYLOAD = None
360                     else: #If drone code not exists in db then status returns to 0 and communication is over
361                         print("Drone not exists in database and rejected...")
362                         status = 0
363                         self.__LORA_PAYLOAD = None
364                         break
365                 elif status == 2 and msg == DRONE_REQ_CHARGE: #in status 2, drone validation is succeed/drone asks
    for charging
366                     charging_proccess = Thread(target=op1.charging_procedure, args=
    (self,packet.header_from,drone_id,)) #create a thread for charging
367                     #check if station is occupied
368                     db_query = """SELECT occupied FROM charging_stations WHERE STATION_ID =
    '{}';""".format(self.__station_id)
369                     occupied = self.DB_select_query(db_query)
370                     if not occupied[0]:
371                         db_query = """UPDATE charging_stations SET occupied=1 WHERE STATION_ID =
    '{}';""".format(self.__station_id)
372                         self.DB_insert_update_query(db_query)
373                         respond_msg = PROCEED_TO_CHARGE
374                         self.lora_send_msg(respond_msg,packet.header_from)
375                         self.__LORA_PAYLOAD = None
376                         status = 0
377                         charging_proccess.start()
378                         charging_proccess.join()
379                         db_query = """UPDATE charging_stations SET occupied=0 WHERE STATION_ID =
    '{}';""".format(self.__station_id)
```

```
                        self.DB_insert_update_query(db_query)
                        self.communication_flag = False
                        self.__LORA_PAYLOAD = None
                        break
                    else:
                        print("Station is not available!")
                        respond_msg = OCCUPIED_MSG
                        self.lora_send_msg(respond_msg,packet.header_from)
                        status = 0
                        self.__LORA_PAYLOAD = None
                        break
                elif status == 2 and msg == DRONE_REQ_INFO: #in status 2, drone validation is succeed/drone asks for information
                    respond_msg = ASK_WHAT_INFO
                    self.lora_send_msg(respond_msg,packet.header_from)
                    status = 3
                    self.__LORA_PAYLOAD = None
                elif status == 3: #in status 3, drone asks for information
                    #list of possible info requests
                    print("Information list under construction")
                    status = 0
                    self.__LORA_PAYLOAD = None
                    break
                else:
                    print("Wrong message")
                    status = 0
                    self.__LORA_PAYLOAD = None
                    break


    def user_input_procedure(self,menu_btn):
        print("Operation (Thread) for user input is running...")
        GPIO.add_event_detect(menu_btn, GPIO.FALLING, callback=lambda x: op2.myMenu(self,menu_btn), bouncetime=500)
        while True:
            pass
```

- init_methods.py

```
#All methods for initialize various systems in charging station, such as:
#Serial port for GPS
#Lora module
#Database connection
#LCD screen

from _constants import *
import time
import pynmea2
from serial import Serial
from pyLoraRFM9x import LoRa, ModemConfig
import mysql.connector
from mysql.connector import errorcode
import RPi.GPIO as GPIO
from rpi_lcd import LCD
import queue
import requests


GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

```

```
#Start Queue procedures to getting values from threads
my_LCD = queue.Queue()
my_DB = queue.Queue()
my_LoRa = queue.Queue()
my_GPS = queue.Queue()

def storeInLCDQueue(f):
    def wrapper(*args):
        my_LCD.put(f(*args))
    return wrapper
def storeInDBQueue(f):
    def wrapper(*args):
        my_DB.put(f(*args))
    return wrapper
def storeInLoRaQueue(f):
    def wrapper(*args):
        my_LoRa.put(f(*args))
    return wrapper
def storeInGPSQueue(f):
    def wrapper(*args):
        my_GPS.put(f(*args))
    return wrapper
#End Queue procedures

#initialize serial port
@storeInGPSQueue #Tell in following function where to return its value
def serial_init(aStation):
    print("\t\t- GPS module")
    port = SERIAL_PORT
    baudrate = SERIAL_BAUDRATE
    gps_fixed = None
    try:
        ser = Serial(port,baudrate,timeout = 2)
        time.sleep(0.1)
        ser.close()
    except Exception as e:
        print("\t\t- Serial init error: ",e)
        ser = None

    if ser is not None:
        try:
            while True:
                gps_fixed = aStation.get_GPS_data(ser)
                #print("Gps_fixed: ", gps_fixed)
                print("\t\t\t\t(Waiting gps to find sattelites...)")
                if gps_fixed is not None:
                    gps_f = gps_fixed["is_fixed"]
                    if gps_f == '1' or gps_f == '2':
                        break
                time.sleep(5)
        except Exception as e:
            print("Fixed gps - Error: ",e)
            ser = None

    if ser is not None:
        print("\t\t\t|")
        print("\t\t\t -> GPS module is initialized\n")

    return ser
```

```
 82
 83 #Initialize LoRa device with SPI interface
 84 @storeInLoRaQueue #Tell in following function where to return its value
 85 def lora_init(self,address):
 86     print("\t\t- LoRa module")
 87     try:
 88         lora = LoRa(1,27,address,
 89                         reset_pin=22,
 90                         freq=LORA_FREQ,
 91                         modem_config=ModemConfig.Bw125Cr45Sf128,
 92                         tx_power=14,
 93                         acks=True)
 94     except:
 95         lora = None
 96
 97     time.sleep(1)
 98     if lora is not None:
 99         print("\t\t\t|")
100         print("\t\t\t -> LoRa module is initialized\n")
101
102     return lora
103
104 # mysql database connection self testing
105 @storeInDBQueue #Tell in following function where to return its value
106 def database_init(aStation):
107
108     print("\t\t- Internet & Database connection")
109     try:
110         request = requests.get("https://www.google.com", timeout=5)
111     except (requests.ConnectionError, requests.Timeout) as exception:
112         print(exception)
113         return None
114
115     connection, cursor = aStation.connect_DB()
116
117     if connection is not None:
118         print("\t\t\t|")
119         print("\t\t\t -> Internet is connected and Database connection is established\n")
120         cursor.close()
121         connection.close()
122         return True
123     else:
124         return None
125
```

```
126 #Initialize and check LCD display with i2c interface
127 @storeInLCDQueue #Tell in following function where to return its value
128 def LCD_init():
129     print("\t- LCD module")
130     try:
131         lcd = LCD()
132         #print("\nLCD object: ",lcd,"\n")
133         lcd.text("LCD display",1)
134         lcd.text("works good",2)
135         time.sleep(1)
136         lcd.clear()
137         lcd.text("Initializing",1)
138         lcd.text("systems...",2)
139 
140     except:
141         print("\nLCD init error!\n")
142         lcd = None
143 
144     if lcd is not None:
145         print("\t\t|")
146         print("\t\t -> LCD is initialized\n")
147     return lcd
```

- drone_operations.py

```
1 import time
2 from _constants import *
3 import RPi.GPIO as GPIO
4 from threading import Thread
5 from datetime import datetime,date
6 
7 GPIO.setmode(GPIO.BCM)
8 GPIO.setup(PIN_FROM_SENSOR_BOARD,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
9 GPIO.setup(PIN_TO_SENSOR_BOARD,GPIO.OUT)
10 
11 power_sample_list = []
12 
```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
def measure_power(aStation):
    ina219 = aStation.get_ina219()
    lcd = aStation.get_lcd()
    try:
        print("Start measuring energy consumption\n...")
        while True:
            if aStation.get_charging_flag():
                current = "{:.2f}".format(ina219.current()/1000)
                power_sample_list.append(12.2 * ina219.current())
                #print("Power: ",12.2 * current , " W")
                lcd.text("Current: "+ current+"A",2)
                time.sleep(1)
            else:
                print("Charging is stopped")
                break

    except Exception as e:
        print("Power meter error: ", e)

def calculate_energy_consumption():
    pass

def clear_values(station):
    GPIO.output(PIN_TO_SENSOR_BOARD,0)
    db_query = """UPDATE charging_stations
                    SET occupied=0
                    WHERE STATION_ID = '{}';""".format(
                                                station.get_station_ID())
    station.DB_insert_update_query(db_query)
    station.reset_lora_payload

def charging_procedure(aStation,aDrone_address,aDrone_Code):
    try:
        lcd = aStation.get_lcd()
        print("\n\n",aDrone_Code, " is going to Charge!!")
        GPIO.output(PIN_TO_SENSOR_BOARD,0)

        landing_date = datetime.today().strftime('%Y-%m-%d')
        starting_time = datetime.today().strftime('%H:%M:%S')

        measuring_power = Thread(target = measure_power, args=(aStation,))

        lcd.clear()
        lcd.text(" Waiting drone,",1)
        lcd.text(" for landing...",2)

        #Drone start landing procedure and station is waiting it
        for tries in range(3): # trying 3 times for "LANDING_WAITING_TIME" if something goes wrong in
landing
            print("\t\t\tWaiting for landing...")
            count_wait_time = 0

            while GPIO.input(PIN_FROM_SENSOR_BOARD) == 0: # while the drone is in landing process
                count_wait_time = count_wait_time + 1     # wait for some time
                time.sleep(1)
                if count_wait_time > LANDING_WAITING_TIME:# if time exceeds LANDING_WAITING_TIME seconds
then abort charging
                    break

```

```
            time.sleep(0.1)
            if GPIO.input(PIN_FROM_SENSOR_BOARD) == 1:    # check once again if drone is landed correctly
                lcd.clear()
                lcd.text("Drone is",1)
                lcd.text("       landed!",2)
                GPIO.output(PIN_TO_SENSOR_BOARD,1)        # and start charging
                time.sleep(2)
                break
            elif tries == 2:
                print("Abort charging due to late landing...")
                lcd.clear()
                lcd.text("Landing is",1)
                lcd.text("       aborted!",2)
                clear_values(aStation)
                return

        #Drone is landed and station check for any error in connection until charge is finished
        aStation.set_charging_flag(True)
        aStation.communication_flag = False
        time.sleep(1)
        lcd.clear()
        lcd.text("Charging...",1)
        print("\nDrone is landed")
        measuring_power.start() #start thread to measure power

        while True:
            if GPIO.input(PIN_FROM_SENSOR_BOARD) == 1: #drone is in place
                ########here start the record in charging history for drone with station_ID
                if aStation.get_lora_payload() is not None:
                    packet = aStation.get_lora_payload()
                    msg = packet.message.decode("utf-8").strip('\x00')
                    print("Message received from drone: ", msg)

                    if msg == DRONE_CHARGE_FINISH: #charging is finished and have to update the DB and close
    the charging procedure
                        print("Drone's battery is charged")
                        GPIO.output(PIN_TO_SENSOR_BOARD,0)
                        aStation.set_charging_flag(False)
                        energy = (sum(power_sample_list)*(len(power_sample_list)/3600))/1000 #energy in wh
                        print("\nEnergy consumption for drone ",aDrone_Code," is {:.2f}
    wh\n".format(energy))
                        stoping_time = datetime.today().strftime('%H:%M:%S')
                        power_sample_list.clear()
                        lcd.clear()
                        lcd.text("  Charging is,",1)
                        lcd.text("    finished",2)

                        while True:
                            time.sleep(2) # some time to take off
                            if GPIO.input(PIN_FROM_SENSOR_BOARD) == 0: #if drone took off
        #here stop the record in charging history for drone with drone_Code at Station with station_ID#
                                db_query1 = """UPDATE charging_stations
                                               SET occupied=0
                                               WHERE STATION_ID = '{}';""".format(
                                                                    aStation.get_station_ID())
                                db_query2 = """INSERT INTO charging_history(station_ID,
                                                                    drone_Code,
                                                                    starting_time,
                                                                    stoping_time,
                                                                    landing_date,
                                                                    energy_consumption)
                                               VALUES('{}','{}','{}','{}','{}','{}');""".format(
                                                                     aStation.get_station_ID(),
                                                                     aDrone_Code,
                                                                     starting_time,
                                                                     stoping_time,
                                                                     landing_date,
                                                                     energy)
```

```
                                aStation.DB_insert_update_query(db_query1)
                                aStation.DB_insert_update_query(db_query2)
                                aStation.reset_lora_payload
                                return
                        else:
                            #print("Wrong message from drone while charging...")
                            #add code to handle the message
                            time.sleep(0.5)
                    else:
                        #do nothing
                        pass

                else:
                    #add code to handle the error
                    print("Drone is not in place")
                    time.sleep(1)

            measuring_power.join()

        except Exception as e:
            print("Error in charging_procedure loop: ", e)
            clear_values(aStation)
        except KeyboardInterrupt:
            print ('KeyboardInterrupt exception is caught')
            clear_values(aStation)
        finally:
            lcd.clear()
            lcd.text("Station is ready",1)
            lcd.text("================",2)
```

- wait_for_user_operation.py

```
import RPi.GPIO as GPIO
import time
import sys
from ChargingStation import *

################################################
#Menu states, just printing methods
def Settings(station):
    print("Settings")
    pass

def Information(station):
    print("Information")
    pass

def Exit(station):
    print("Exit")
    return 999

def Position_Settings(station):
    print("Position Settings")
    pass

def Set_new_position(station):
    print("Set new position")
    pass

def Show_user_ID(station):
    print("Show user ID")
    pass
```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
31
32 def Show_coordinates(station):
33     print("Show Station coordinates")
34     pass
35
36 def Show_altitude(station):
37     print("Show station altitude")
38     pass
39
40 #do something for menu final states
41 def Exitting(station):
42     return -1
43
44 def Send_GPS_data(station): #Final state
45     lcd = station.get_lcd()
46     res = station.gps_data_to_DB()
47     if res:
48         lcd.clear()
49         lcd.text("   GPS data, is",1)
50         lcd.text("    updated!",2)
51         time.sleep(2)
52     else:
53         lcd.clear()
54         lcd.text("GPS data, is NOT",1)
55         lcd.text("    updated!",2)
56     return -1
57
58 def Get_station_ID(station): #Final state
59     #DB query
60     lcd = station.get_lcd()
61     station_id = station.get_station_ID()
62     lcd.text("Station ID:",1)
63     lcd.text(station_id,2)
64     time.sleep(5)
65     return -1
66
67 def Get_station_coor(station): #Final state
68     lcd = station.get_lcd()
69     lcd.clear()
70     lat, lon = station.get_station_coordinates()
71     print(lat,"\n",lon)
72     lcd.text(lat,1,'left')
73     lcd.text(lon,2,'left')
74     time.sleep(5)
75     return -1
76
```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
77 def Get_station_alt(station): #Final state
78     lcd = station.get_lcd()
79     lcd.clear()
80     altitude = station.get_station_altitude()
81     print("Show station altitude: ", altitude)
82     lcd.text("Altitude",1,'center')
83     lcd.text(altitude+" m",2,'center')
84     time.sleep(5)
85     return -1
86 
87 
88 menu_states = {1:[Settings,"    Settings"],
89                2:[Information,"   Information"],
90                3:[Exit,"      Exit"],
91                10:[Position_Settings,"Position Set"],
92                11:[Exit,"      Exit"],
93                100:[Set_new_position,"Set new position"],
94                1000:[Send_GPS_data," "],
95                101:[Exit,"      Exit"],
96                20:[Show_user_ID,"  Show user ID"],
97                200:[Get_station_ID," "],
98                21:[Show_coordinates,"Show coordinates"],
99                210:[Get_station_coor," "],
100               22:[Show_altitude," Show altitude"],
101               220:[Get_station_alt," "],
102               23:[Exit,"      Exit"],
103               999:[Exitting,"Exitting"]}
104 
105 exit_states = [3,11,101,23] #the final submenu states ids
106 
107 ################################################
108 #functions for menu selections
109 
110 def btn_pressing_time(channel):
111         #tune the sleep times and counter adding according to better results
112     time.sleep(0.1)
113     counter = 0
114     while GPIO.input(channel) == 0: # Wait for button pressing
115         time.sleep(0.1)
116         counter = counter + 0.5
117         if counter > 2:
118             break
119     return counter    # How long was the button pressed?
120 
121 def do_something(state,station):
122 
123     lcd = station.get_lcd()
124 
125     if state in menu_states:
126         lcd.clear()
127         lcd.text(menu_states[state][1],1)
128         handler = menu_states[state][0]
129         return handler(station)
130     else:
131         lcd.clear()
132         lcd.text("ErrorErrorError!",1)
133         print("Error")
134         return None
135 
```

```
136 def is_exit_state(state):
137 
138     if state in exit_states:
139         int_length = len(str(state))
140         if int_length == 1:
141             return 1
142         else:
143             s = list(str(state))
144             s[int_length-1] = '0'
145             return int(''.join(s))
146     else:
147         return None
148 
149 def select_state(state):
150     if state == 999:
151         return state
152     return state*10
153 
154 def next_state(state):
155     final_state = is_exit_state(state)
156     if final_state == None:
157         return state+1
158     return final_state
159 
160 
161 #interrupt callback function for circle menu button
162 def myMenu(myStation,channel):
163 
164     lcd = myStation.get_lcd()
165     buttonTime = btn_pressing_time(channel)
166 
167     print("Button time: ", buttonTime)
168     if buttonTime >= 2:
169         lcd.clear()
170         try:
171             current_state = 1
172             returned_value = do_something(current_state,myStation)
173             while True:
174                 if GPIO.input(channel) == 0:
175                     buttonTime = btn_pressing_time(channel)
176                     if buttonTime >= 2:
177                                             #print("Select from state: ",current_state)
178                         if returned_value == 999:
179                             current_state = 999
180                         new_state = select_state(current_state)
181 
182                     elif .1 <= buttonTime < 2:
183                                             #print("Go to next state from state: ",current_state)
184                         new_state = next_state(current_state)
185                     else:
186                                             #print("Else with time: ",buttonTime)
187                         continue
188 
189                     returned_value = do_something(new_state,myStation)
190                                         #print(returned_value)
191                     if returned_value==-1: #pressed exit menu
192                         lcd.clear()
193                         break
194                     current_state = new_state
195 
196         except Exception as e:
197             print("Menu Error: ",e)
198 
199     time.sleep(0.1)
200     lcd.clear()
201     print("Exit menu")
202     lcd.text("Station is ready",1)
203     lcd.text("================",2)
204 
```

- _constants.py

```
#RPI PINOUT
RED_BUTTON = 4
MENU_BUTTON = 17
PIN_FROM_SENSOR_BOARD = 24
PIN_TO_SENSOR_BOARD = 25

#SENSOR CONSTANTS
SERIAL_PORT = "/dev/ttyAMA0"
SERIAL_BAUDRATE = 115200

LORA_FREQ = 868 #set lora module frequency in MHerz
LORA_ADDRESS = 2 #set lora module address for this station

INO219_SHUNT_OHMS = 0.1
INO219_MAX_EXPECTED_AMPS = 3.0

#MESSAGES FROM DRONE TO CS
HELLO_MSG_FROM_DRONE = "Hello"
DRONE_REQ_CHARGE = "charging"
DRONE_REQ_INFO = "information"
DRONE_CHARGE_FINISH = "finished"
ASK_FOR_UPDATE_STATION_NETWORK = "update station network"

#MESSAGES FROM CS TO DRONE
HELLO_MSG_FROM_CS = "Hello, who are you?"
ASK_REQ_FROM_DRONE = "Make request"
PROCEED_TO_CHARGE = "Proceed to charge"
OCCUPIED_MSG = "Occupied"
ASK_WHAT_INFO = "What info you want?"

LANDING_WAITING_TIME = 10 #time in seconds for waiting drone to land
```

- main.py

```python
#IMPORT LIBRARIES
from _constants import *
import RPi.GPIO as GPIO
import time
from threading import Thread
from ChargingStation import *

#------------------------------------------------------------------------
#DEFINE GPIOs AND VARIABLES

GPIO.setmode(GPIO.BCM)
GPIO.setup(RED_BUTTON,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(MENU_BUTTON,GPIO.IN,pull_up_down=GPIO.PUD_UP)

#Define some methods
def btn_pressing_time(channel):
        #tune the sleep times and counter adding according to better results
    time.sleep(0.1)
    counter = 0
    while GPIO.input(channel) == 0: # Wait for the button up
        time.sleep(0.1)
        counter = counter + 0.5
        if counter > 2:
            break
    return counter


#get the client config data
f = open('clients_info.txt','r')
client_dict = {}

for row_data in f:
    data = row_data.split()
    data = data[0].split(":")
    client_dict[data[0]] = data[1]
        #print(client_dict)

station = ChargingStation(client_dict['charging_station_unique_ID'],
                          client_dict['client_name'],
                          client_dict['client_email'],
                          client_dict['client_tel'],
                          client_dict['client_address'])


#------------------------------------------------------------------------
```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
#2. SYSTEMS INITIALIZATION
print("\n\n==== 2. START SYSTEMS INITIALIZATION ====\n")
station.init_systems() #method from ChargingStation class to initialize systems

lcd = station.get_lcd()
db_con = station.get_db_con()
lora = station.get_lora()
gps = station.get_gps()

if lcd is not None:
    if lora is not None:
        lcd.clear()
        lcd.text("LoRa is",1)
        lcd.text("checked...",2)
        time.sleep(4)
    else:
        print("\t\t- LoRa module error!")
        lcd.clear()
        lcd.text("Lora ERROR",1)
        time.sleep(2)
    if gps is not None:
        lcd.clear()
        lcd.text("GPS is",1)
        lcd.text("checked...",2)
        time.sleep(4)
    else:
        print("gps error!")
        lcd.clear()
        lcd.text("GPS ERROR...",1)
        time.sleep(2)
    if db_con is not None:
        lcd.clear()
        lcd.text("Database is",1)
        lcd.text("checked...",2)
        time.sleep(4)
    else:
        print("db_con error!")
        lcd.clear()
        lcd.text("Database",1)
        lcd.text("ERROR...",2)
        time.sleep(2)
else:
    print("LCD error!")

```

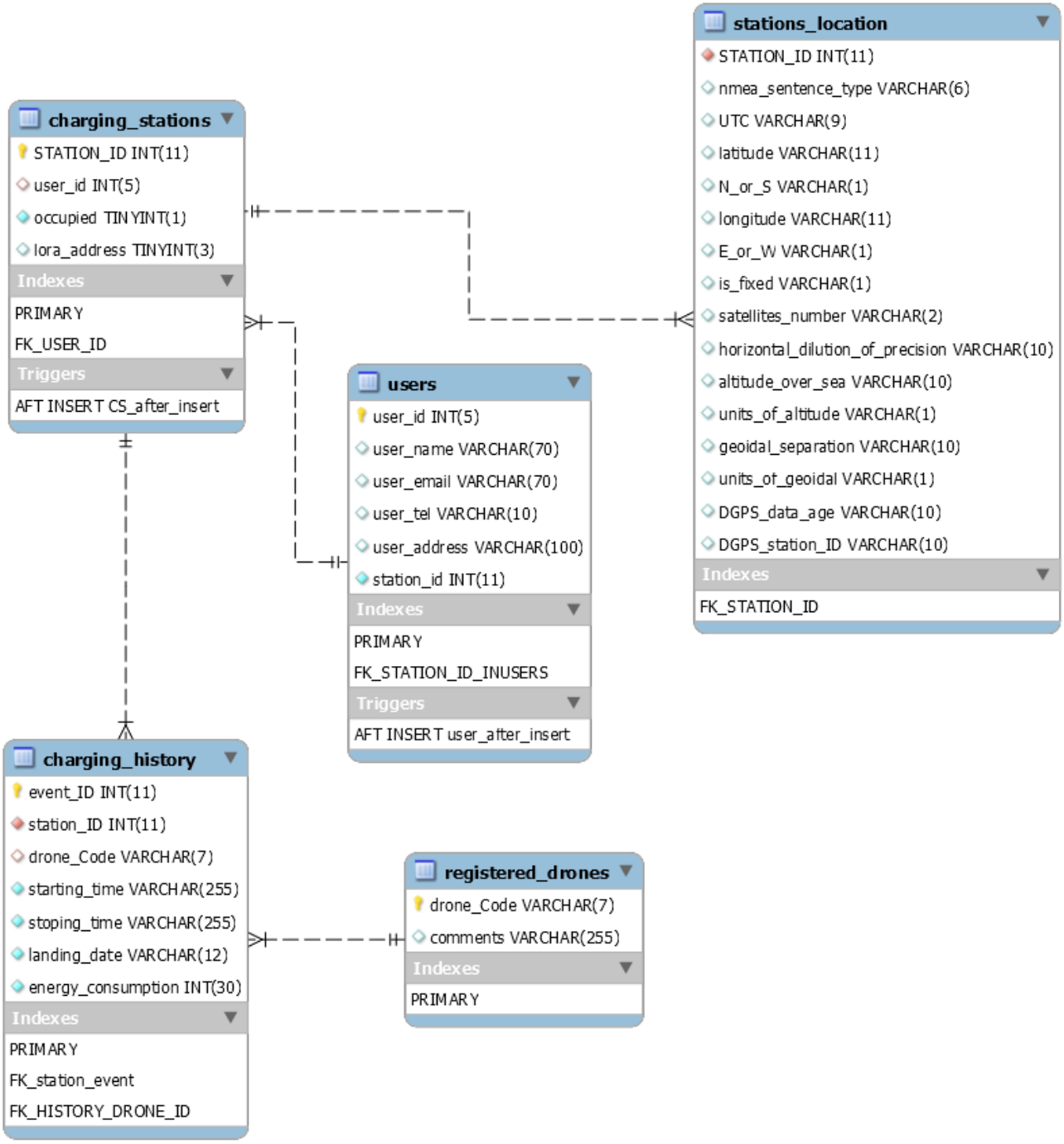
```
#Check database if station exists and if not, create new station in database
print("\t\t\t-- Update Database with Charging Station's data")
try:
    if db_con is not None: #if there is connection with database
        sql_query = """SELECT * FROM charging_stations
                       WHERE STATION_ID={};""".format(station.get_station_ID()) #check if station already exists
        res = station.DB_select_query(sql_query)
        if res: #Station exists
            print("\t\t\t\t|")
            print("\t\t\t\t|->Station already exists")
            print("\t\t\t\t ->If location has changed,\n\t\t\t\t\t\tset new position from settings")
            lcd.text("    Station!",1)
            lcd.text(" already exists",2)
            time.sleep(2)
            lcd.clear()
            lcd.text("If location has",1)
            lcd.text("changed, set new",2)
            time.sleep(3)
            lcd.clear()
            lcd.text(" position from",1)
            lcd.text("menu -> settings",2)
            time.sleep(3)
            lcd.clear()
        else: #Add new station
            print("\t\t\t\t|")
            print("\t\t\t\t ->Add new station in Database")
            lcd.text("  New Station!",1)
            lcd.text("================",2)
            station.add_new_station()
            print("\t\t\t\t\tAsk user to press red button for 2 seconds,")
            time.sleep(4)
            lcd.clear()
            lcd.text("To send position",1)
            lcd.text(" to database...",2)
            print("\t\t\t\t\tto send gps location data to Database\n")
            time.sleep(5)
            lcd.clear()
            lcd.text("     Press",1)
            lcd.text("   RED BUTTON",2)
            time.sleep(0.1)
            while True: #Wait for redbutton pressing and send gps data to db
                if GPIO.input(RED_BUTTON) == 0:
                    buttonTime = btn_pressing_time(RED_BUTTON)
                    if buttonTime >= 2:
                        print("\t\t\t\t\t|-> Red button pressed")
                        lcd.clear()
                        lcd.text("Please wait...",1)
                        time.sleep(3)
                        lcd.text("Sending data...",2)
                        if station.gps_data_to_DB():
                            time.sleep(4)
                            lcd.clear()
                            print("\t\t\t\t\t\tDone...\n")
                            lcd.text("Data sent...",1)
                            time.sleep(2)
                            lcd.clear()
                            break
                        else:
                            print("Sending GPS data: Failed!")
                            lcd.clear()
                            lcd.text("    Failed!",1)
                            lcd.text("Press Again",2)

    else: #if there is not connection with database save data to a variable and use lora to broadcasting
        gps_data = {}
        gps_data = station.get_GPS_data(gps)
        print(gps_data)

        #routine to broadcast with LoRa, charging station's position
        #...
        pass

except Exception as e:
    print("Error: ",e)


lcd.text("Station is ready",1)
lcd.text("================",2)

print("\n==== 2. END SYSTEMS INITIALIZATION ====\n")
#INITIALIZATION COMPLETED
```

Βάσεις φόρτισης για μη επανδρωμένα οχήματα - αυτόματη προσέγγιση σε βάση φόρτισης

```
171
172 #-------------------------------------------------------------------------
173 #3. OPERATION
174
175 print("\n==== 3. START NORMAL OPERATION ====\n")
176
177 #Activate lora module
178 station.set_lora_mode_cad()
179
180 #Starting thread for waiting user input from menu button
181 user_input_thread = Thread(target=station.user_input_procedure, args=(MENU_BUTTON,))
182 station.threading_list.append(user_input_thread)
183 user_input_thread.start()
184
185 while True:
186     try:
187         time.sleep(2)
188         if not len(station.threading_list) == 0:
189             #print(station.threading_list)
190             for t in station.threading_list:
191                 #print("Thread: ",t,"-->", t.is_alive())
192                 if not t.is_alive():
193                     #print("Thread: ", t ," is not alive")
194                     station.threading_list.remove(t)
195         else:
196             print("Exit Program")
197             lcd.clear()
198             break
199
200     except KeyboardInterrupt:
201         print ('KeyboardInterrupt exception is caught in main.py and terminate the program!')
202         lcd.clear()
203         break
```

7.2 Παράρτημα 2 – Βάση Δεδομένων

Με τη βάση δεδομένων που σχεδιάστηκε για την εργασία, καλύπτονται οι ανάγκες διαχείρισης και ελέγχου πολλαπλών σταθμών φόρτισης, καθώς επίσης γίνεται και καταγραφή ιστορικού φόρτισης των drones και κατανάλωσης ενέργειας. Στην Εικόνα 46 παρουσιάζεται το Σχεσιακό Μοντέλο Οντοτήτων της βάσης δεδομένων το οποίο δημιουργήθηκε μέσω Reverse Engineering από το εργαλείο MySQL Workbench όπου χρησιμοποιούσα ως UI για την διαχείριση της βάσης.



Εικόνα 46: Μοντέλο οντοτήτων συσχετίσεων βάσης δεδομένων σταθμού φόρτισης

Ο server όπου λειτουργεί η βάση δεδομένων είναι ένα εικονικό μηχάνημα με κάποια από τα χαρακτηριστικά να φαίνονται στον παρακάτω πίνακα:

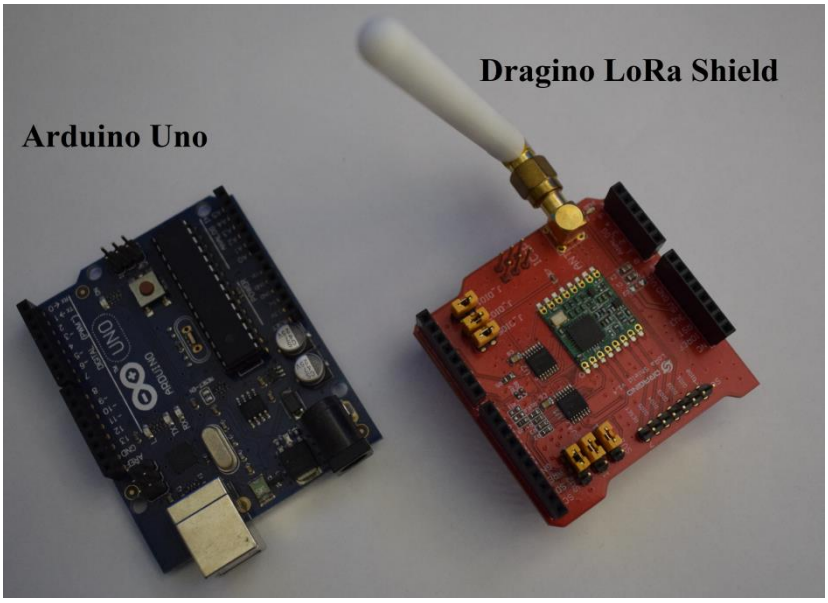
Virtualization Manager	: Oracle VirtualBox 6.1
Kernel/Operation System	: Linux/Centos 7
MySQL version	: mysql 8.0.13
MySQL listen port	: 3306/tcp

Ο host βρίσκεται σε προσωπικό χώρο και μέσω port forwarding υλοποιείται απομακρυσμένη πρόσβαση από το σταθμό φόρτισης με χρήση προσωπικού domain name. Τα στοιχεία της σύνδεσης βρίσκονται στο αρχείο “database_info.txt” (βλ. Παράρτημα 1). Με αυτό το τρόπο έγινε και ο έλεγχος ορθής επικοινωνίας μεταξύ σταθμού και server.

7.3 Παράρτημα 3 – Βοηθητικός εξοπλισμός

Για την παρουσίαση της λειτουργίας του σταθμού φόρτισης υπήρξε η ανάγκη επικοινωνίας με κάποιο drone μέσω LoRa. Καθώς δεν υπήρχε διαθέσιμο drone με εξοπλισμό τέτοιας επικοινωνίας, χρησιμοποιήθηκε ένα Dragino LoRa Shield 868MHz.

Το Dragino (Εικόνα 47) είναι μια πλακέτα με ενσωματωμένο το LoRa RFM96 module στα 868 MHz που έχει τη κατάλληλη διαρρύθμιση για την τοποθέτησή του πάνω σε Arduino (Εικόνα 48).



Εικόνα 47: Arduino Uno (Αριστερά) - Dragino LoRa Transceiver Shield 868MHz (Δεξιά)

Εικόνα 48: Το Dragino τοποθετημένο πάνω σε Arduino Uno

Με χρήση της βιβλιοθήκης RadioHead, στην οποία βασίζεται και η βιβλιοθήκη που χρησιμοποιήθηκε και στον σταθμό σε γλώσσα python, έγραψα ένα πρόγραμμα στο Arduino IDE, όπου έδινα στο Serial Monitor το μήνυμα που έπρεπε να σταλεί στο σταθμό και περίμενα στη συνέχεια την ανάλογη απάντηση (Εικόνα 35).

Επίσης ο λόγος που δεν χρησιμοποίησα άλλου είδους εξοπλισμό και προτίμησα το Arduino, είναι πως οι συνηθέστεροι flight controller που χρησιμοποιούνται σε αυτοσχέδια drone είναι βασισμένοι στο Arduino (π.χ. Ardupilot) και τα προγράμματά τους γραμμένα σε C/C++. Άρα ήταν μια καλή ευκαιρία να δοκιμάσω την επικοινωνία μεταξύ διαφορετικών αναπτυξιακών κυκλωμάτων και γλωσσών προγραμματισμού.

Βιβλιογραφία - Αναφορές

C.Mourgelas, S.Kokkinos, A.Milidonis and I.Voyiatzis, "Autonomous drone charging stations: A survey".*24th Pan-Hellenic Conference on Informatics (PCI 2020).* [1]

S. Aldhaher, P. D. Mitcheson, J. M. Arteaga, G. Kkelis and D. C. Yates, "Light-Weight Wireless Power Transfer for Mid-Air Charging of Drones," *11th European Conference on Antennas and Propagation (EUCAP),* 2017. [2]

A. B. Junaid, A. Konoiko, Y. Zweiri, M. N. Sahinkaya and L. Seneviratne, "Autonomous Wireless Self-Charging for Multi-Rotor Unmanned Aerial Vehicles," *Energies,* 2017. [3]

T. Campi, S. Cruciani, M. Feliziani and F. Maradei, "High Efficiency and Lightweight Wireless Charging System for Drone Batteries," *2017 AEIT International Annual Conference,* 2017. [4]

T. Campi, S. Cruciani and M. Feliziani, "Wireless Power Transfer Technology Applied to an Autonomous Electric UAV with a Small Secondary Coil," *Energies,* 2018. [5]

C. Yang, Y. He, H. Qu, J. Wu, Z. Hou, Z. Lin and C. Cai, "Analysis, design and implement of asymmetric coupled wireless power transfer systems for unmanned aerial vehicles," *AIP Advances 9,* 2019. [6]

D. Dominique and F. Dominique, "A position and orientation independent drone charging system," *The Journal of Engineering,* 2017. [7]

A. Compagnin, A. Cenzato, E. Lungavia, N. Bagarello, A. Rossi, A. Re, L. Olivieri, F. Sansone, R. Mantellato and A. Francesconi, "Autoport Project: a Docking Station for Planetary Exploration Drones," *AIAA SciTech Forum, Grapevine, Texas, 55th AIAA Aerospace Sciences Meeting,* January 2017. [8]

C. H. Choi, H. J. Jang, S. G. Lim, H. C. Lim, S. H. Cho and I. Gaponov, "Automatic wireless drone charging station creating essential environment for continuous drone operation," *Researchgate,* 2016. [9]

M. A. Kishk, A. Bader and M.-S. Alouini, "Aerial Base Station Deployment in 6G Cellular Networks Using Tethered Drones," *IEEE vehicular technology magazine,* 2020. [10]

O. M. Bushnaq, M. A. Kishk, A. Celik, M.-S. Alouini and T. Y. Al-Naffouri, "Optimal Deployment of Tethered Drones for Maximum Cellular Coverage in User Clusters," *IEEE transactions on wireless communications, Vol20, No.3,* 2021. [11]

B. Kosarnovsky and S. Arogeti, "Geometric and constrained control for a string of tethered drones," *Elsevier,* 2020. [12]

W. Walendziuk, P. Falkowski and K. Kulikowski, "The Analysis of Power Supply Topologies for Tethered Drone Applications," *Proceedings,* 2020. [13]

Dronesurvey Asia, "Dronesurvey Asia," [Online]. Available: www.dronesurvey.asia.

14]

D. Lee, J. Zhou and W. T. lin, "Autonomous Battery Swapping System for Quadcopter," 15] *International Conference on Unmanned Aircraft Systems (ICUAS), Denver Marriott Tech Center, Denver, Colorado, USA,* 2015.

Z.-n. Liu, X.-q. Liu, L.-j. Yang, D. Leo and H.-w. Zhao, "An Autonomous Dock and Battery 16] Swapping System for Multirotor UAV," *Researchgate,* 2018.

Dpendent, "dpendent.ch," Dpendent, [Online]. Available: www.dpendent.ch/products.
17]

R. Pudur, V. Hanumante, S. Shukla and K. Kumar, "Wireless Power Transmission: A Survey," 18] *IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014),,* 2014.

X. Mou and H. Sun, "Wireless Power Transfer: Survey and Roadmap," *IEEE 81st Vehicular* 19] *Technology Conference (VTC Spring),* 2015.

X. Zhang, H. Meng, B. Wei, S. Wang and Q. Yang, "An improved three-coil wireless power link 20] to increase spacing distance and power for magnetic resonant coupling system," *EURASIP Journal on Wireless Communications and Networking,* 2018.

C. Woo, S. Kang, H. Ko, H. Song and J. O. Kwon, "Auto charging platform and algorithms for 21] long-distance flight of drones," *IEEE International Conference on Consumer Electronics (ICCE),* 2017.